Neural Network Based Generation of Reactionary Dance Improvisations

M'AI have this dance?
or
How to train your DRA-GAN

H. N. Basien

MSc. thesis of H. N. Basien

Set within the '*AI-man*' project, in collaboration with
Stichting *Another Kind of Blue* and *CompactCopters UG*

July 27, 2021

Neural Network Based Generation of Reactionary Dance Improvisations

M'AI have this dance?
or
How to train your DRA-GAN

by

H. N. Basien

to obtain the degree of Master of Science
at the Delft University of Technology,
Faculty of Aerospace Engineering - Department of Control & Operations,
to be defended publicly on Tuesday August 24, 2021 at 14:00.

Student number:	4207653		
Project duration:	February 10, 2020 – May 16, 2021		
Thesis committee:	Prof. Dr. Ir. J. M. Hoekstra	Chairman	[TU Delft]
	Dr. Ir. J. Ellerbroek	Supervisor	[TU Delft]
	Dr. Ir. A. Bombelli	External	[TU Delft]
	Dr. Ir. A. Jamshidnejad	Additional	[TU Delft]

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Abstract

This MSc. thesis presents the extensive research conducted on designing neural networks capable of generating reactionary dance motions. The research was conducted in collaboration with *Stichting Another kind of Blue* (*AKOB*) and *CompactCopters UG* (*CC*).

It is set within the industry context of the '*AI-man*' project, developed by *AKOB* and *CC*; a novel dance show intended to create a truly interactive duet between man and machine. It is the follow-up to the successful '*Airman*' project; a dance show utilizing a swarm of 12 drones to perform with a human dancer on stage. *Airman* enabled the drone swarm to represent a human figure and copy the improvised dance motions of the human dancer, recorded via a motion capture (MoCap) system in real-time. The *AI-man* project aims to take this to the next level, by placing an AI motion controller in between the live MoCap recording and the desired humanoid pose passed to the drone swarm. By doing so, the system should be responsive to the human dancer's motion and generate reactionary dance motions to created a shared improvised duet.

Long-term motion generation is a very challenging problem and currently actively researched by various AI researchers. For reference, many researchers consider the prediction or generation of motion longer than one second, or even just 500 ms, 'long-term' [Ghosh et al., 2018; Gui et al., 2018a; Tang et al., 2017]. In contrast to this, the interactive segment in the *Airman* show was about one minute long. During the initial literature study, no comparable research was found attempting to generate motion in reaction to full-body human input.

To facilitate the research, a custom MoCap dataset was acquired containing over 9 hours of improvised dance motions between two dancers. In comparison to current publicly available datasets, the acquired dataset appears to be second longest MoCap dataset to date.

To achieve the desired objective, two models were created:
1) An initial regression model, successfully mimicking the human motion in the training dataset, but very unstable in a real-time environment.
2) A generative model, intended to generate a variety of reactionary motions in real-time, in reaction to an arbitrary motion input.

For the generative model, a novel neural network architecture, for the generation of long-term reactionary motions, is proposed: The 'Differential Recurrent Attention GAN' (DRA-GAN). Utilizing the training methodology of 'Generative Adversarial Networks' (GANs) [Goodfellow et al., 2014], in combination with design elements from 'Recurrent Neural Networks' (RNNs) [Recurrent units: LSTM [Hochreiter and Schmidhuber, 1997] & GRU [Cho et al., 2014]], attention mechanisms [Bahdanau et al., 2015], differential equations and 'Principal Component Analysis' (PCA) [Jolliffe, 2002].

The proposed model showcases promising training progression, but has so far failed to generate true long-term output. This is because the training halts in a common failure mode for GANs; 'mode collapse' [Liu and Tuzel, 2016].

While the model has so far not succeeded in generating the desired results, it cannot be concluded that the model is unable to generate the desired results. This is because, the full potential of the model has not yet been explored, e.g. by means of 'Bayesian Hyperparameter Optimization' (BHPO).

The framework to achieve BHPO was developed, but the extremely long training times ($\approx$ 2 weeks), on the limited hardware available, have prevented the evaluation of a sufficient number of model variations. To facilitate further research, an extensive list of methodologies was compiled that could potentially resolve the current problems of the model.

Preface

We, as *TU Delft* students, are incentivised to reinvent the wheel and take bold leaps every now and then. As Sir Isaac Newton once proclaimed:

> *No great discovery was ever made without a bold guess.*
>
> *Isaac Newton*

This notion describes the origin of this research very well:
"Let's teach a computer how to dance with humans!" Well, it is easier said than done.

Throughout this research I have learned much about neural networks and artificial intelligence, but also about learning in general. Seeing my little nephew and niece slowly grow and learn has taught me a few valuable lessons: Learning takes time and teaching is no easy feat either. Humans are widely considered the pinnacle of evolution, but it still takes us roughly a whole year to even learn how to balance and many more to produce something we would classify as dance.

On the one hand, I think that we can cut our neural networks some slack, for not being perfect within a few hours or even weeks of learning. But on the other hand, waiting that long, to know if a model has succeeded or not, is extremely impracticable and significantly slows down the research process.

> *If I have seen further than others, it is by standing upon the shoulders of giants.*
>
> *Isaac Newton*

A cliché, I know, but I had wished this to be true for myself after completion of this thesis. However, even after my in depth study of neural networks and their inner workings, I still feel very much dwarfed by these giants and the wealth of knowledge that there is to discover, as well as the infinite possibilities and pitfalls that this fascinating technology provides.

I think that, given my initial naivity when embarking on this research, I have slowly but steadily passed through many stages of the 'Dunning-Kruger Effect' [Kruger and Dunning, 1999]. However, I believe that with respect to deep learning, I have finally passed the 'valley of despair' and am now slowly working my way up the long-winded 'slope of enlightenment'.

Ultimately I believe that, the true strength of a good engineer or scientist comes from the humility to question your own assumptions and to never stop asking questions.

> *Ipse se nihil scire id unum sciat. (I know that I know nothing.)*

I am grateful for the many that have guided and aided me in this research and the road towards it:

- **Thanks to my loving parents**
  , for giving me the gift of life and supporting me throughout it.
- **Thanks to my peers Alessandro, Antonin, Karlo and Raoul**
  , for their support and listening to my problems.
- **Thanks to my reviewers Andres, Joachim, Johanna, Marcus and Max**
  , for enduring all my many (<,> & <'>) mistakes.
- **Thanks to my supervisor, Dr. Ir. Joost Ellerbroek, and all committee members**
  , for guiding me and allowing me to finally graduate.
- **Thanks to Another Kind of Blue**
  , for facilitating the data acquisition and the long years of collaboration culminating in this research.
  And last, but most certainly not least:
- **Thanks to my love Johanna**
  , for your love, care and always being there for me, through the good and the hard times.
  I love you and I look forward to spending more time with you!

Ultimately, thanks to all the people and experiences that have gotten me to where and who I am today!

I have always joked about the 16 semesters that my father took to finish his aerospace diploma and I never thought that I would surpass that one day. However, by now it has been almost 18 semesters since my first day at the *TU Delft* and while taking ever so slightly longer to complete my study, than I had originally anticipated; 'life happened': Starting my own company, working in other start-ups, being a board member of Yoroshi, traveling and meeting new people, having a myriad of other interesting projects on the side and having found a loving and caring woman for life ... if I had the choice, I would do it all again. My only hope is that my children do not attempt to bump the record to 20 semesters...

I am honored, after all these years, to finally graduate under the same professor that gave the very first lecture I attended at the TU Delft, on the 3rd of September 2012 at 08:45; Prof. Dr. Ir. Jacco Hoekstra.

H. N. Basien
Delft, May 2021

Contents

Abstract III

Preface V

Table of Contents VI

List of Abbreviations VIII

List of Symbols X

List of Figures XII

List of Tables XIV

List of Equations XV

List of Algorithms XVI

I Thesis Context & Summary 2

1 Introduction 4
- 1.1 Industry Context . . . 4
- 1.2 Problem Statement . . . 6
- 1.3 The 'Creative Problem' . . . 6
- 1.4 Thesis Structure . . . 6

2 Research Planning 8
- 2.1 Research Goals . . . 8
- 2.2 Research Objective . . . 8
- 2.3 Research Questions . . . 9
- 2.4 Research Hypothesis . . . 9
- 2.5 Research Framework . . . 9

3 Research Summary 10
- 3.1 Data Handling . . . 10
- 3.2 Training . . . 13
- 3.3 Neural Network Design . . . 13
- 3.4 Results . . . 18
- 3.5 Live Interaction . . . 18
- 3.6 Discussion . . . 18

II Research & Development 20

4 Data Handling 22
- 4.1 Data Acquisition . . . 22
- 4.2 Data Structure . . . 25
- 4.3 Data Cleaning . . . 30
- 4.4 Data Preparation . . . 35
- 4.5 Dataset Overview . . . 42

5 *DanceNet-BHPO* Framework 44

5.1 Training Framework . . . 44
5.2 Optimization Framework . . . 47
5.3 Cluster Framework . . . 51
5.4 Data Output . . . 53

6 Regression Model 56
6.1 Design Choices . . . 56
6.2 Implementation . . . 57
6.3 Results . . . 60
6.4 Feedback Dilemma . . . 64

7 Generative Model 68
7.1 Generative Adversarial Network . . . 68
7.2 Initial Design Problems . . . 74
7.3 Model Design Improvements . . . 75
7.4 DRA-GAN . . . 87
7.5 Results . . . 95

III Application & Relevance 100

8 Live Interaction 102
8.1 Live Simulation . . . 102
8.2 Live Visualization . . . 106

9 Future Work 110
9.1 Data . . . 110
9.2 Architecture . . . 112
9.3 Training . . . 114
9.4 Optimization . . . 115
9.5 Validation . . . 117

10 Conclusions 120
10.1 Scientific Contribution . . . 121
10.2 Discussion . . . 123

IV Appendices 126

Bibliography 128

A Coordinate System Definitions 132
A.1 OptiTrack . . . 132
A.2 BVH . . . 132
A.3 Blender . . . 132
A.4 Unreal Engine 4 . . . 132

B Statistical Human Motion Model 134
B.1 Histograms . . . 134
B.2 PDFs . . . 135
B.3 Edge-case 'Hips-ψ' . . . 136
B.4 Regression Parameters . . . 136

C BHPO GUI 142

D Algorithms 146

E DRA-GAN Results 150
E.1 V1.0-23 | 6D-ATT-DI-DO-LSTM-GAN | e406d70f24 . . . 150
E.2 V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | d9be67f6a7 . . . 154
E.3 V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | 42241955b0 . . . 158
E.4 V1.2 | 6D-PCA-ATT-DI-DO-LSTM-Pain-GAN | 96142329a6 . . . 162
E.5 V1.3-7 | 6D-PCA-ATT-DI-DO-LSTM-Pain-LJ-GAN | 7d2d6588e0 . . . 166
E.6 V1.3-15 | 6D-PCA-ATT-DI-DO-GRU-Pain-LJ-GAN | 89ba1481a7 . . . 170

F Preliminary Report 174

List of Abbreviations

General Abbreviations:

BMI	Body Mass Index
COG	Center of Gravity
COVID	Coronavirus Disease
DOF	Degrees of Freedom
GT	Grounded Theory
ID	Identifier
IR	InfraRed
MoCap	Motion Capture
MSc	Master of Science
R&D	Research & Development
TTMS	Turning Test for Motion Synthesis

Databases:

AMASS	*Archive of Motion Capture as Surface Shapes*
CMU	*CMU Graphics Lab Motion Capture Database*
H3.6M	*Human3.6M(illion poses) Dataset*
DB	Database

Entities:

AKOB	*Stichting Another Kind of Blue*
CC	*CompactCopters UG*
CMU	*Carnegie Mellon University*
HREC	*Human Research Ethics Committee*
ICLR	*International Conference on Learning Representations*
NIPS	*Neural Information Processing Systems*
SIGGRAPH	*Special Interest Group on Computer Graphics and Interactive Techniques*
TU Delft	*Delft University of Technology*
EU	European Union
NL	The Netherlands

File Types:

BVH	*BioVision* Hierarchy
C3D	Medical motion experiment format
CSV	Comma Separated Values
FBX	*Filmbox*
JSON	JavaScript Object Notation
TAK	*OptiTrack* 'take'
TRC	'Motion Analysis' filetype

Computing Hardware:

CPU	Central Processing Unit
GPU	Graphics Processing Unit
PC	Personal Computer
RAM	Random-Access Memory
VM	Virtual Machine
VR	Virtual Reality

Mathematics:

BDF	Backward Differentiation Formula
EMG	Exponentially modified Gaussian distribution
PCA	Principle Component Analysis
PDF	Probability Density Function
SLERP	Spherical Linear intERPolation

Machine-/Deep learning:

6D	6 dimensional rotation representation
Adam	Adaptive Moment Estimation
AI	Artificial Intelligence
ATT	soft-self-ATTention
BCE	Binary Cross Entropy
BHPO	Bayesian Hyperparameter Optimization
BP	BackPropagation
BPTT	BackPropagation Through Time
CNN	Convolutional Neural Network
cuDNN	*NVIDIA CUDA®* Deep Neural Network library
DCGAN	Deep Convolutional GAN
DI	Derivative Input
DL	Deep Learning
DNC	Differentiable Neural Computer
DO	Derivative Output
DRA-GAN	Differential Recurrent Attention GAN
EB	EarlyBreak
GAN	Generative Adversarial Network
GRU	Gated Recurrent Unit
HP	Hyperparameter
LJ	Limited Judgement
LSTM	Long Short-Term Memory
MHU	Modified High-way Unit

ML	Machine Learning
MLP	Multilayer Perceptron
MSE	Mean Squared Error
NN	Neural Network
NTM	Neural Turing Machine
PFNN	Phase-Functioned Neural Networks
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
UCB	Upper Confidence Bound

Software and Networking:

ASCII	American Standard Code for Information Interchange
ESXi	*VMware ESXi* Hypervisor
GUI	Graphical User Interface
LAN	Local Area Network
NaN	Not a Number
OS	Operating System
SDK	Software Development Kit
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UE4	*Unreal Engine 4*
WAN	Wide Area Network
WLAN	Wireless LAN

Units:

BPM	Beats per Minute
FPS	Frames Per Second
SI	Système international (d'unités)

For PDF readers:
All abbreviations in the text are equipped with a
hidden hyper-link to this page. - Example link: EXMPL

Return to the original page:
Adobe Reader - Right-click→'Previous View'

List of Symbols

General:

ϕ	Roll angle
θ	Pitch angle
ψ	Yaw angle
μ	Mean
σ	Standard deviation
p_{t-1}	Previous Pose
p_t	Current Pose
p_{t+1}	Next Pose

Chapter 4 [Data Handling]:

Filtering:

σ_p	Mean Pose
$\hat{p_t}$	Filtered Pose at time t
R	Discrete time exponential filter ratio
r	Continous time exponential decay ratio
$t_{1/2}$	Half-life
$t_{1/r}$	Time for exponential decay to reach r ('$\frac{1}{r}$-life')
M	Parameter defining the relation between R and r for a given $t_{1/r}$
$\hat{R}$	Equivalent discrete time exponential filter ratio at 1 FPS

6D Rotations:

R_{6D}	6D rotation vector
R_{2D}	2D rotation vector
$\vec{x_r}$,$\vec{y_r}$	Original rotation vectors
$\vec{x_n}$,$\vec{y_n}$,$\vec{z_n}$	Normalized rotation vectors
R_M	Rotation matrix

Chapter 5 [*DanceNet-BHPO* Framework]:

Reality Gap:

RG	Reality Gap
$\text{Loss}_{\text{Training}}$	Loss on the training dataset
$\text{Loss}_{\text{Test}}$	Loss on the test dataset

Utility:

$\mu_{\text{est}}(P)$	Estimated mean of Gaussian process at point P
$\sigma_{\text{est}}(P)$	Estimated standard deviation of Gaussian process at point P
μ_{max}	Currently evaluated optimum
κ	UCB tuning parameter

Chebyshev's inequality:

x	Random variable x
X	Realization of random variable x
μ_x	Mean of random variable x
σ_x	Standard deviation of random variable x
k	Number of standard deviations from the mean

Chapter 6 [Regression Model]:

Loss:

Loss_{μ}	Mean loss
p_{act_i}	Actual pose
p_{pred_i}	Predicted pose
$\hat{\alpha_{\mu}}$	Estimated mean angular error

Weights & Biases:

N	Number of values in pose
H	Number of values in hidden vector
W_X	Weights of layer X
B_X	Biases of layer X
WB_X	Weights and Biases of layer X
$\underset{Y \times Z}{WB}$	Weights and Biases of size $Y \cdot Z$

Chapter 7 [Generative Model]:

G	Generator
D	Discriminator

BCE Loss:

g_{pred}	Predicted label - Discriminator guess
g_{act}	Ground truth label - Discriminator answer
Loss_{BCE}	Binary cross entropy loss

Discriminator Metrics:

μ_{Real}	Real-Mean : Average discriminator guess for all real data
μ_{Fake}	Fake-Mean : Average discriminator guess for all fake data
μ_{Bias}	Mean-Bias : Average discriminator guess for all data
$\Delta\mu$	Mean-Distance : Absolute average diff. between guesses on real and fake data
$\Delta\mu_{\max}$	Max-Distance : Maximum $\Delta\mu$ for current training run
d_{Real}	Real-CenterOffset : Absolute average diff. between guesses on real data and random guessing
d_{Fake}	Fake-CenterOffset : Absolute average diff. between guesses on fake data and random guessing
d	Mean-CenterOffset : Combined absolute diff. from random guessing for real and fake data
$\text{Loss}_{\text{Generator}}$	Augmented generator loss
$\text{Loss}_{\text{Pain}}$	Pain criterion loss
$\text{Loss}_{\text{Total}}$	Combined generator loss

Chapter 9 [Future Work]:

$\text{SR}_{\max}$	Maximum (desired) success ratio

Appendix B [Statistical Human Motion Model]:

#Frames	Total number of frames in dataset
#Bins	Number of histogram bins
α_{Range}	Total range of possible angles
α_{Res}	Angular resolution
$c[i]$	Histogram count at index i
$\{p_i\}$	Optional extra parameters of PDF number i
l_i	Translation parameter of PDF #i
s_i	Scaling parameter of PDF #i
w_i	Skewing parameter of PDF #i

List of Figures

1.1 *Newton's Duet* (2015) . . . 5
1.2 *Airman* (2018) . . . 5

2.1 Research Framework . . . 9

3.1 Skeletal rig used by *AKOB* . . . 12
3.2 *DanceNet* Architecture: Regression Model . . . 14
3.3 *DanceNet* Architecture: Regression Model with Feedback . . . 14
3.4 Neural network Architecture: Original GAN . . . 15
3.5 *DanceNet* Architecture: Generative Model . . . 15

4.1 Behind the scenes at *AKOB*s studio, during data acquisition . . . 23
4.2 *AKOB* Dataset: Distribution of take durations . . . 24
4.3 *AKOB* dataset: Distribution of labels . . . 25
4.4 Rigged example character mesh in *T-Pose* . . . 26
4.5 Kinematic Chain Example - 3 Bones in 2D . . . 27
4.6 *ArangoDB*: Data acquisition → Recording day → Takes . . . 29
4.7 *ArangoDB*: Take → Recordings → Skeleton + Motion . . . 29
4.8 Parameter Distributions: Relative joint channel histograms . . . 30
4.9 Parameter Distributions: LeftArm histograms . . . 31
4.10 Example of a discontinuous representation of rotations in 2D . . . 35
4.11 N-D Rotations: Visualization of 2D representation of 1D rotations . . . 37
4.12 PCA: Example of PCA on Gaussian scatter data in 2D . . . 38
4.13 PCA: PC #14 - Spine bending right → left . . . 40
4.14 PCA: Zero/Mean Pose . . . 41
4.15 PCA: Influence Ranking . . . 41
4.16 PCA: Data Dimensionality Reduction . . . 42

5.1 BHPO: Neural Network Training Framework . . . 45
5.2 BHPO: Hyperparameter Optimization Outline . . . 48
5.3 Bayesian Optimization: Utility function example . . . 49
5.4 BHPO: Cluster Computing Framework . . . 52
5.5 BHPO: Resultant Data Structure . . . 54

6.1 Neural network Architecture: LSTM Unit . . . 57
6.2 Neural network Architecture: GRU Unit . . . 58
6.3 *DanceNet* Architecture: Regression Model . . . 59
6.4 *DanceNet* Results: Regression Model . . . 61
6.5 BHPO Losses (Sorted): GRU . . . 62
6.6 BHPO Losses (Sorted): LSTM . . . 62
6.7 Regression Model: Reality Gap . . . 63
6.8 BHPO: Success-Rate vs Chebyshev's Inequality . . . 64
6.9 *DanceNet* Architecture: Regression Model with Feedback . . . 65
6.10 *DanceNet* Results: Regression Model with Feedback . . . 65

7.1 *DanceNet* Architecture: Generative Model . . . 69
7.2 GAN: Mode Collapse Example . . . 73
7.3 Neural network Architecture: Stacked RNN Units . . . 75

7.4 Neural network Architecture: Fully-Connected Final Layer . . . 77
7.5 Neural network Architecture: Derivative Output . . . 78
7.6 Derivative Output: Vanishing State Problem . . . 79
7.7 Neural network Architecture: Derivative Input . . . 80
7.8 Neural network Architecture: Soft-Self-Attention . . . 81
7.9 Attention: Attention Weights Example . . . 82
7.10 Pain: Loss Example . . . 83
7.11 Pain: Losses over Time Example . . . 84
7.12 Neural network Architecture: Limited Judgement . . . 86
7.13 Neural network Architecture: DRA-GAN DanceNet . . . 89
7.14 DRA-GAN: Network Parameters Visualization Example . . . 92
7.15 DRA-GAN: Discriminator Guesses Example . . . 93
7.16 DRA-GAN: Discriminator Metrics Example . . . 94
7.17 DRA-GAN Results: Final Pose [Epoch #20] . . . 96
7.18 DRA-GAN Results: Twisted Poses [Epoch #{5,10&15}] . . . 96
7.19 DRA-GAN Results: Output Motion Example . . . 97

8.1 *DanceNet* Architecture: Inference Model . . . 103
8.2 *OptiTrack*: *Motive* Streaming Settings . . . 105
8.3 Live Simulation: *OptiTrack* Reconstruction Bug . . . 106
8.4 *DanceNet* Architecture: Live Model . . . 107
8.5 Live Visualization: Unreal Engine T-Pose Example . . . 108
8.6 Live Visualization: Unreal Engine Rotations Bug . . . 109

9.1 BDF: Approximation Errors over Time . . . 111
9.2 BDF: Mean Approximation Errors . . . 111
9.3 Gradient Clipping . . . 116
9.4 DRA-GAN: Minimized Network Parameters Visualization Example . . . 118

A.1 Coordinate system definitions . . . 133

B.1 Example PDFs . . . 135
B.2 Parameter Distributions: Histogram . . . 140
B.3 Parameter Distributions: Best Fit PDFs . . . 141

C.1 BHPO GUI Examples: MainScreen . . . 143
C.2 BHPO GUI Examples: Graphs - Training/Testing Loss & Reality Gap . . . 144
C.3 BHPO GUI Examples: Graphs - Training/Testing ΔLoss & dt . . . 145

E.1 V1.0-23 | 6D-ATT-DI-DO-LSTM-GAN | e406d70f24: Network Weights . . . 151
E.2 V1.0-23 | 6D-ATT-DI-DO-LSTM-GAN | e406d70f24: Discriminator History . . . 152
E.3 V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | d9be67f6a7: Network Weights . . . 155
E.4 V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | d9be67f6a7: Discriminator History . . . 156
E.5 V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | d9be67f6a7: Pain History . . . 157
E.6 V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | 42241955b0: Network Weights . . . 159
E.7 V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | 42241955b0: Discriminator History . . . 160
E.8 V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | 42241955b0: Pain History . . . 161
E.9 V1.2 | 6D-PCA-ATT-DI-DO-LSTM-Pain-GAN | 96142329a6: Network Weights . . . 163
E.10 V1.2 | 6D-PCA-ATT-DI-DO-LSTM-Pain-GAN | 96142329a6: Discriminator History . . . 164
E.11 V1.2 | 6D-PCA-ATT-DI-DO-LSTM-Pain-GAN | 96142329a6: Pain History . . . 165
E.12 V1.3-7 | 6D-PCA-ATT-DI-DO-LSTM-Pain-LJ-GAN | 7d2d6588e0: Network Weights . . . 167
E.13 V1.3-7 | 6D-PCA-ATT-DI-DO-LSTM-Pain-LJ-GAN | 7d2d6588e0: Discriminator History . . . 168
E.14 V1.3-7 | 6D-PCA-ATT-DI-DO-LSTM-Pain-LJ-GAN | 7d2d6588e0: Pain History . . . 169
E.15 V1.3-15 | 6D-PCA-ATT-DI-DO-GRU-Pain-LJ-GAN | 89ba1481a7: Network Weights . . . 171
E.16 V1.3-15 | 6D-PCA-ATT-DI-DO-GRU-Pain-LJ-GAN | 89ba1481a7: Discriminator History . . . 172
E.17 V1.3-15 | 6D-PCA-ATT-DI-DO-GRU-Pain-LJ-GAN | 89ba1481a7: Pain History . . . 173

Any Figure that does not have an explicit 'Source' mentioned, under the caption, was custom made for this report.

List of Tables

4.1 Parameter Distributions: Dataset limits and statistics per channel . . . 32
4.2 PCA: Human Interpretation of Principal Components . . . 39
4.3 *AKOB* MoCap Dataset Summary . . . 42
4.4 MoCap Dataset Comparison . . . 43

5.1 Hyperparameters: Training Framework . . . 47
5.2 Exploration vs Exploitation . . . 49
5.3 Hyperparameter Re-parametrization Trade-off . . . 51

6.1 Hyperparameters: Regression Model . . . 59
6.2 Regression Model: BHPO Run Statistics & Results . . . 60
6.3 Regression Model: (Partially) Optimized Hyperparameters for GRU & LSTM . . . 60

7.1 BCE Loss: Simplification and Limits . . . 69
7.2 Hyperparameters: Output Ratios . . . 77
7.3 Hyperparameters: Limited Judgment . . . 86
7.4 Hyperparameters: DRA-GAN . . . 90
7.5 DRA-GAN: Parameter Distribution Example . . . 90

8.1 Live Simulation: Modes and Speed Test . . . 103
8.2 Live Simulation: UDP Message components . . . 104

9.1 DRA-GAN: Minimized Parameter Distribution Example . . . 118

10.1 Methodology comparison between this research and Huang et al. [2020] . . . 123

B.1 PDF regression arguments - Single . . . 138
B.2 PDF regression arguments - Double . . . 139

E.1 V1.0-23 | 6D-ATT-DI-DO-LSTM-GAN | e406d70f24: General Settings . . . 150
E.2 V1.0-23 | 6D-ATT-DI-DO-LSTM-GAN | e406d70f24: Hyperparameters . . . 150
E.3 V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | d9be67f6a7: General Settings . . . 154
E.4 V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | d9be67f6a7: Hyperparameters . . . 154
E.5 V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | 42241955b0: General Settings . . . 158
E.6 V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | 42241955b0: Hyperparameters . . . 158
E.7 V1.2 | 6D-PCA-ATT-DI-DO-LSTM-Pain-GAN | 96142329a6: General Settings . . . 162
E.8 V1.2 | 6D-PCA-ATT-DI-DO-LSTM-Pain-GAN | 96142329a6: Hyperparameters . . . 162
E.9 V1.3-7 | 6D-PCA-ATT-DI-DO-LSTM-Pain-LJ-GAN | 7d2d6588e0: General Settings . . . 166
E.10 V1.3-7 | 6D-PCA-ATT-DI-DO-LSTM-Pain-LJ-GAN | 7d2d6588e0: Hyperparameters . . . 166
E.11 V1.3-15 | 6D-PCA-ATT-DI-DO-GRU-Pain-LJ-GAN | 89ba1481a7: General Settings . . . 170
E.12 V1.3-15 | 6D-PCA-ATT-DI-DO-GRU-Pain-LJ-GAN | 89ba1481a7: Hyperparameters . . . 170

List of Equations

4.1 Filtering: Exponential filtering . . . 33
4.2 Filtering: Exponential decay time-ratio . . . 34
4.3 Filtering: Parameterized dynamic exponential filtering parameter . . . 34
4.4 Filtering: Pose estimation, by first-order approximation, if p_t is invalid . . . 35
4.5 Filtering: Pose estimation, by return to mean, if $p_t \& (p_{t-1} \| p_{t-2})$ are invalid . . . 35
4.6 N-D Rotations: Definition of 6D representation of 3D rotations . . . 36
4.7 N-D Rotations: Definition of 2D representation of 1D rotations . . . 36

5.1 Reality Gap . . . 46
5.2 Utility Function: Upper Confidence Bound . . . 49
5.3 Chebyshev's inequality & 'Upper Confidence Bound' Utility Criterion . . . 50

6.1 Regression Model: Loss Function - Mean Squared Error . . . 59
6.2 Number of weights and biases per Gate . . . 60
6.3 LSTM: Number of weights and biases . . . 60
6.4 GRU: Number of weights and biases . . . 60
6.5 Regression Model: Model Loss → Angular Error . . . 61

7.1 Generative Model: Loss Function - Binary Cross Entropy . . . 69
7.2 Neural network Architecture: Hidden layer size . . . 77
7.3 Generative Model: Discriminator Metrics . . . 93
7.4 Generative Model: Advanced Loss Function . . . 94

9.1 Noisy Labels . . . 112
9.2 κ tuning . . . 117

B.1 Joint Histogram Resolution . . . 134
B.2 PDF transformation . . . 135
B.3 Double PDF definition . . . 136

List of Algorithms

5.1 Early Breaking - Simplified . . . 46
5.2 Bayesian Hyperparameter Optimization . . . 48

7.1 GAN Training Algorithm - Simplified . . . 70
7.2 PainLoss . . . 83
7.3 DRA-GAN Processing Framework . . . 87

8.1 Live Simulation: *OptiTrack* Message Decoding . . . 104
8.2 Live Simulation: *OptiTrack* Packet Injection . . . 108

D.1 Early Breaking . . . 146
D.2 Neural network Architecture: Soft-Self-Attention Layer . . . 146

Disclaimer:

All parameters and functions are actually handled inside the larger scope of the BHPO framework classes. The following algorithms are stand-alone representations of various algorithms and are purely for illustrative purposes and conveying the primary logic. To this extend not every algorithm is 100% executable and holds some pseudo-code elements. These simplifications are required, as the full code would be to extensive to print in this report.

I

Thesis Context & Summary

1 Introduction

The following research project is set within the context of the *AI-man* project; a collaboration between *Another Kind of Blue (AKOB)*, a Dutch dance company, and *CompactCopters (CC)*, a German drone R&D company. The *AI-man* project desires to create a duet between man and machine by means of utilizing a drone-swarm in combination with artificial intelligence (AI). Ultimately, it is intended to result in a theater performance to be enjoyed by audiences all over the world.

For this thesis a neural network was designed which generates human motion, in reaction to a human dancer's input. This neural network is intended to be utilized as the drive algorithm for the prior developed drone swarm, representing an artificial 'dancer'.

1.1. Industry Context

The research to be performed will contribute to the modern dance performance *AI-man*, by *AKOB*, intended to be shown in theaters to general audiences in 2022. Apart from the scientific contribution, the following thesis research intends to directly serve a clear purpose for an industry application. Understanding the industry context of this thesis allows for a better understanding of some of the design choices made.

1.1.1. Industry Partners

Stichting Another Kind of Blue (AKOB) is a The Hague based dance company, specializing in the combination of art and technology. They create visually stunning performances by augmenting modern dance with digital technologies, such as projectors, drones, motion tracking and VR.

CompactCopters UG (CC) is a German drone R&D company, co-founded by the author of this thesis. *CC* has been involved in projects such as developing a solar-powered drone for industrial inspection, drone swarms for theater performances, load-analysis and flight-testing tools for small aircraft and data-processing tools for drone-based inspection of plants in greenhouses.

1.1.2. Past Projects

AKOB had been experimenting with the usage of aircraft and drones for stage performances for many years, however the earliest attempts were all manually controlled, before the transition to full automation was made.

The *AI-man* project is the logical conclusion of two previous drone-dance projects created by *AKOB* & *CC* (among other partners), over the course of the past 6 years:

1. *Newton's Duet* (2015): [Figure 1.1] http://y2u.be/fI20Mxd8vLc
   - 2 drones and 2 dancers
   - Pre-planned choreography
   - Tour around NL[1]
2. *Airman* (2018): [Figure 1.2] http://y2u.be/NuP4YhMzt3k

[1] Also performed during the 174th Dies Natalis of the TU Delft

- A swarm of 12 Drones, representing the human form, able to copy the improvisation of 1 Dancer
- Pre-planned choreography + interactive improvisation
- Tour around NL + global shows

For *Newton's Duet*, an automated control system was developed, which allowed two drones to follow a pre-choreographed set of 3D reference positions over time (see Figure 1.1).

In *Airman*, a solo-dancer was being recorded live by an *OptiTrack* motion capture (MoCap) system. From the MoCap data, key points were extracted and transformed, to be used as reference positions for a larger-than-life skeletal replica, formed by the drone swarm in front of the dancer (see Figure 1.2).

Figure 1.1: *Newton's Duet* (2015)
Source: *Stichting Another Kind of Blue*

Figure 1.2: *Airman* (2018)
Source: *Stichting Another Kind of Blue*

1.1.3. *AI-man*

AKOB's choreographer's motivation behind the *AI-man* project can be summarized by the following set of philosophical questions[2]:

> *What is free will? Do humans have it? Can machines have it?*
>
> *David Middendorp - Choreographer, Another Kind of Blue*

To explore these questions, without trying to find a final answer, it was decided to create a duet between a human and a drone swarm, in which both performers have as much 'free will' as possible. This means, unlike most conventional show pieces, no fixed choreography is defined.

All previous projects have explored various levels of man-machine interaction in an artistic environment. However, none have yet achieved a true 'duet' in the classical sense of the word:

> *An intricate dance between man and machine where both partners can be 'free' in their decisions of motion and yet bound to each other by a common dance language to complement each other's expression.*
>
> *AI-man Project Vision*

To achieve this, an 'artificially intelligent' larger-than-life virtual dancer (*'output dancer'*) is to be created that can react to a human dancer (*'leading/input dancer'*), by means of a neural network that transforms the live recorded motion-capture data into a reactionary dance.

[2]These questions represent the artistic background behind the showpiece and this research, but this research will not further delve into the philosophical discourse resulting thereout.

1

1.2. Problem Statement

Given the wider context of the *AI-man* project, the primary missing link between the *Airman* and the *AI-man* project is the development of an AI-powered algorithm, capable of generating novel dance motion in real-time.

The *AI-man* project strives to reuse the drone-swarm hardware and control algorithms used in *Airman*. For *Airman*, the reference skeleton, from which the reference positions for each drone are being extracted, is the direct output of the MoCap system. For *AI-man*, this direct throughput is intended to be replaced by an algorithm, that receives the live MoCap skeleton of the dancer as input and generates an entirely new skeleton in reaction to the input dancer, rather than replicating the input dancer's motion.

By choice of *AKOB*, the desired solution utilizes AI technology, in order to allow the drone-swarm to showcase the illusion of 'free will'. This illusion would be very difficult to achieve with conventional control theory and hard-coded pre-defined sets of motions, as it either reuses pre-set motion segments, quickly becoming repetitive, or requires a clear set of control laws defining 'artistic intent'.

To enable a live performance, the design objective can be defined as:

> *Integrating an AI algorithm, capable of dynamically transforming MoCap skeleton frames into reactionary skeleton frames, at* > 100 *Hz*[3] *in real-time, into the Airman framework.*
>
> *AI-man Project Objective*

1.3. The 'Creative Problem'

Due to the artistic nature of the problem at hand, the engineering challenges and methodology for this project differ considerably from the more 'conventional' thesis topics researched at the *TU Delft* Faculty of Aerospace Engineering.

The primary difference lies in the nature of the desired solution. Commonly, an engineering design or research tries to solve a clearly defined problem, or investigate the influence of a certain parameter on a system. For this objective however, the desired final output of the required algorithm has, by definition, no ground-truth, as the resulting motions are intended to be novel and unique for every input motion.

Therefore, the success of the project is divided into two categories: First the successful implementation of the AI algorithm, and second the largely subjective visual quality of the resulting duet. The subjective nature of the desired result makes it very hard, but not impossible, to quantify the quality of the motion data. The quantifiable parameters are primarily restricted to measuring adherence to physical limits and not e.g. how 'beautiful' a resulting motion is.

This creative nature of the desired result is also reflected by the chosen methodology and key design decisions made throughout the project. Examples of these are the choice of a generative model for the final algorithm and the use of 'Grounded Theory' inspired techniques and inductive reasoning[4].

1.4. Thesis Structure

This report is structured into four main parts:

1. **Part I - Thesis Context & Summary**
   Background and condensed summary of the thesis, to allow for a better understanding of the context before examining the research in detail
2. **Part II - Research & Development**
   Detailed account of all work performed, structured in logical order, rather than chronological
3. **Part III - Application & Relevance**
   Steps taken to apply the research in a real-world environment and retrospect of the thesis in the light of its academic and practical contribution
4. **Part IV - Appendices**
   Bibliography and supplementary material

[3] 100Hz was chosen as the desired update rate for the drone swarm control system and is a direct multiplier of other frame rates commonly used by *AKOB*; 25 and 50 FPS and the desired update rate of the *Airman* system.

[4] Grounded theory is research methodology for qualitative research, as certain aspects of this research are within in the realm of the arts and hard to quantify, contrasting common engineering design problems.

This initial Chapter 1 [Introduction] provides an introduction to the industry context of this research and the general result it intends to achieve. The following Chapter 2 [Research Planning] provides an overview of the general guidelines that governed this research; the research goals and objective to be achieved, the questions to be answered, the hypothesis to be proven and the framework guiding the research. To avoid the reader getting lost in the details, before seeing the big picture; Chapter 3 [Research Summary] presents a condensed summary to provide the reader a broad overview of the entire research.

Before any neural network can be trained it requires data to be trained with; Chapter 4 [Data Handling] details the process of acquiring, cleaning and preparing the dataset for training the deep learning (DL) models. Neural network by themselves do not learn, they require a framework for training and optimization; Chapter 5 [*DanceNet-BHPO* Framework] details the training, optimization and cluster frameworks designed to facilitate training the models in this research. Chapter 6 [Regression Model] introduces the initial DL model; while succeeding at achieving its task, in hindsight it has proven to not be the appropriate model to achieve the desired research objective. Chapter 7 [Generative Model] details the design of a novel architecture for generating long-term reactionary motions: The 'Differential Recurrent Attention GAN' (DRA-GAN).

To facilitate the utilization of the model on stage, Chapter 8 [Live Interaction] describes the design of a live simulation framework, as well as the corresponding live visualizations to test it. Chapter 9 [Future Work] introduces various potential improvements to the current model, as a guideline for future researchers intending to build upon this research. Finally, the research is concluded in Chapter 10 [Conclusions], by retrospecting the results of the research and providing a final discussion.

2

Research Planning

The most important elements, which define the premise and guidelines for the research, are stated in this chapter to provide a clearer picture of the overall scope and intent of the research presented.

Before setting out to perform the actual research an extensive literature study was performed, which resulted in a detailed planning guiding the research to follow. These elements are presented in the preliminary thesis/literature study report "*AI-man*: Preliminary Report", provided as annex to this thesis in Appendix F, and will not be explicitly reiterated in this thesis.

2.1. Research Goals

The external goal of the research is focused on the development of a neural network based algorithmic framework for generating human poses resembling the natural motions of an interactive dancer, to be used as reference positions for a drone swarm.

The primary challenge of achieving this goal is finding an answer to the question when the neural network's output is optimal and how to quantify 'natural motions'. Even though a reduction in the model's loss-function on the test and validation datasets may indicate an improvement in performance, this is not necessarily the case, as the research context is set within the arts. A mathematically 'better' outcome is not necessarily represented by a 'nicer', 'more natural' or 'more beautiful' result. Finding a solution to this ambiguity is essential for *AKOB* to deliver a stunning visual performance to their international audiences.

The internal goal of the research is therefore, to solve the ambiguity between objective and subjective evaluation, and how to design a neural network architecture that can in fact achieve the desired output of 'transforming a dancer's improvised motions into appropriate reactionary motions'.

2.2. Research Objective

The resulting research objective of this practice-oriented master's thesis is:

> *To design an algorithm that generates a virtual skeleton in reaction to motion-captured dance improvisations, by comparing existing and creating new algorithms, within the domain of deep learning architectures.*
>
> *Research Objective*

2.3. Research Questions

The following set of research questions have guided the research trajectory:

1. **Data:** What data is required to teach a neural network to generate human motion?
   (a) Quantity: How much motion capture (MoCap) data is required?
   (b) Transformation: Which data representations are beneficial for training neural networks?
2. **Methodology:** Which deep learning methodologies can process and generate motion capture data?
   (a) Architecture: Which neural network architectures are capable of handling multi-dimensional time series data?
   (b) Task: Which function is to be optimized, quantifying the quality of generated human motion?
3. **Optimization:** How can the systems performance be optimized?
   (a) Training: How is the neural network to be trained to generate long-term motions?
   (b) Hyperparameters: How can the initial network architecture design be optimized?

The presented research aims to fulfill the internal goal, by finding well-reasoned answers to these questions. The related research and development is presented in Part II and concluded in Chapter 10.

2.4. Research Hypothesis

To bridge the gap between objective and subjective evaluation a final test was to be issued, where the experts of *AKOB* would be shown various motion outputs, either ground-truth (real) data or generated (fake) data, and have to decide if the data shown is real or fake.

The following research hypothesis is to be proven:

> *Neural networks are capable of generating motion in reaction to live full-body human input, which cannot be distinguished from human motion by human experts.*
>
> *Research Hypothesis*

If this test would result in a distribution of feedback that is similar (with statistical significance) to the distribution that would result from random guessing, it could be asserted that the hypothesis is confirmed.

2.5. Research Framework

A top level overview of the proposed research framework is given in Figure 2.1. It visualizes the interrelations between the tasks and results of the research.

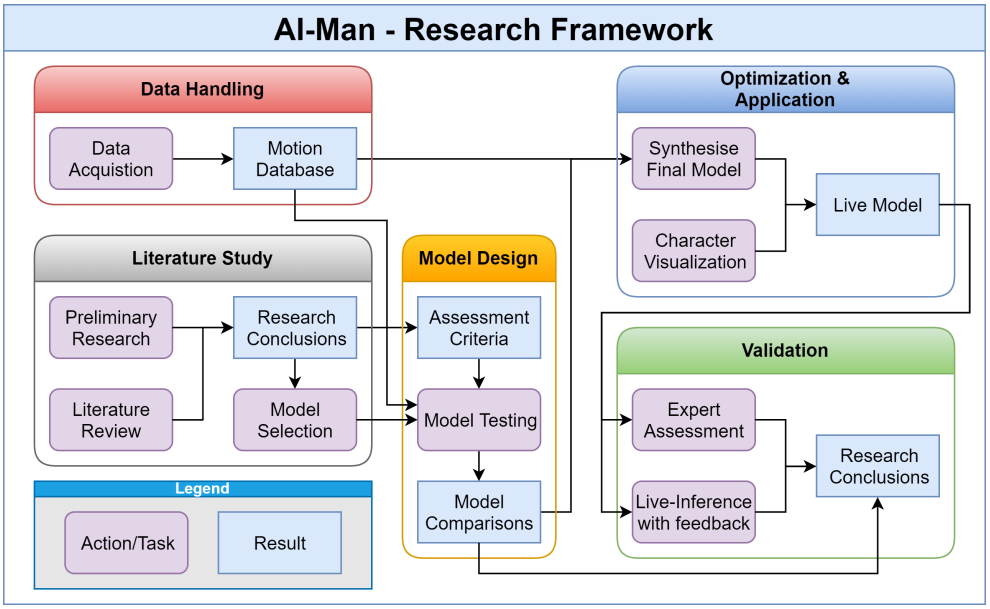


Figure 2.1: Research Framework

3

Research Summary

In this research, insight into the mathematical representations of the human shape, as well as state-of-the-art knowledge of neural networks, is gathered. Combining these yields a novel online virtual motion synthesis algorithm, which at a later stage is intended to be interfaced with the drone hardware of the *Airman* project.

The summary presented in this chapter is intended to function as a guide for the reader to become familiar with the overall research and grant a high-level overview, before going into detail for each research aspect individually. It is possible that, at this point, not all concepts and methods described in this chapter are fully comprehended by the reader; the following Part II is intended to describe everything in-depth and elaborate on the details of this research.

Unlike the detailed account in the subsequent part, this chapter focuses on a chronological order of events, as the research is highly based on inductive reasoning in an attempt to iteratively improve the final model, based on prior insight gained.

3.1. Data Handling

> *Data is the new science. Big Data holds the answers.*
>
> *Pat Gelsinger - CEO, Intel*

"Machine learning algorithms without data to feed them, are as useless as fish without water to swim in." Therefore, the first step in any machine learning (ML)/deep learning (DL) research project is to acquire data to train the models with. Given the fact that some DL architectures are better suited for some types of data than others, the question which comes first, the data or the model, is a bit of a 'chicken and the egg' problem; however, one needs to start somewhere.

3.1.1. Data Acquisition

Based on the Project Objective, as stated in Section 1.2, it is clear that data that needs to be acquired should be motion capture (MoCap) data, fully encompassing the human stature, as well as its motion. More specifically, 'dance motion' needs to be digitized and analyzed. The following sub-categorization shows the layers of specialization required for the required data:

All datasets →MoCap →Dancing →Modern dance →Two interacting dancers

There are several publicly available datasets containing some amount of dance data [Alemi, 2017; Chan et al., 2019; CMU Graphics Lab, 2016; Harvey et al., 2020; Huang et al., 2020; University Of Cyprus, 2012]. However, not all of these utilize MoCap data, instead several models use video recordings instead, which are not applicable for this research. Furthermore, very little research has been done on the interaction of two or more humans in an interactive environment [Aristidou et al., 2018b; Liu et al., 2019; Manzi et al., 2018]. The only dataset that contains two interacting dancers, contain salsa dancers at close distance and account for only a fraction of the total dataset [CMU Graphics Lab, 2016]. Ultimately, there was no publicly available dataset found that matches all categories.

As a result, a custom dataset was created, consisting of 9.2 hours recorded in 147 takes, using the *OptiTrack* MoCap system of *AKOB* located at their studio in The Hague. The data was recorded at 100 Hz for a total of 6.647.486 data frames.

In terms of recording time this makes it the second largest MoCap dataset, compared to the 18 datasets making up the world's largest MoCap data collection to date; the 'Archive of Motion Capture as Surface Shapes ' (*AMASS*) database [Mahmood et al., 2019].

3.1.2. Data Structure

Multiple takes of two dancers performing were recorded. Each take consist of two dancers improvising simultaneously. One of the dancers is defined prior to have the 'leading' role and is free to choose any motion desired. The other dancer is defined to be the 'following' dancer and is tasked with creating a reactionary improvisation to the leading dancer in real-time. Both dancers are tasked to move within the 'dance language' and limits previously agreed upon with the choreographer[1].

The data was saved in anonymized <.BVH> files, according to the regulations of the Human Research Ethics Committee of the TU Delft.

The format in which the human body was digitized consists of 156 floats per frame:

- 6-DOF global transform of the body's root (hips):
  - 3 floats representing the 3D position $[m]$
  - 3 floats representing the global Euler rotations [°]
- 3 floats, for each joint/bone, representing the relative Euler rotations from it to the next [°]
  - This model defines 50 joints/bones for the entire human body:
    - $4 + (4 + 4) \cdot 2 \quad = 20$ joints/bones define the spine (4) and the four limbs $(4 \cdot 4)$
    - $(5 \cdot 3) \cdot 2 \quad = 30$ joints/bones define the details of all 5 fingers, on both hands

This format of interdependent rotations is largely scale independent, and therefore allows the data to be generalized and combined for various humans statures. A visual representation of this rig is presented in Figure 3.1.

The data was combined in a local *ArangoDB* database, for easy access by the DL models and other analysis tools.

3.1.3. Data Preparation

Given the *OptiTrack* tracking system's sub-millimeter accuracy, the data contains almost no noise due to positional in-accuracy. However, raw MoCap data is known for rarely being perfect, as the system can lose tracking of (parts of) the body and create spikes in the data. Hence, the data needs to be cleaned and pre-processed before applying it in an neural network.

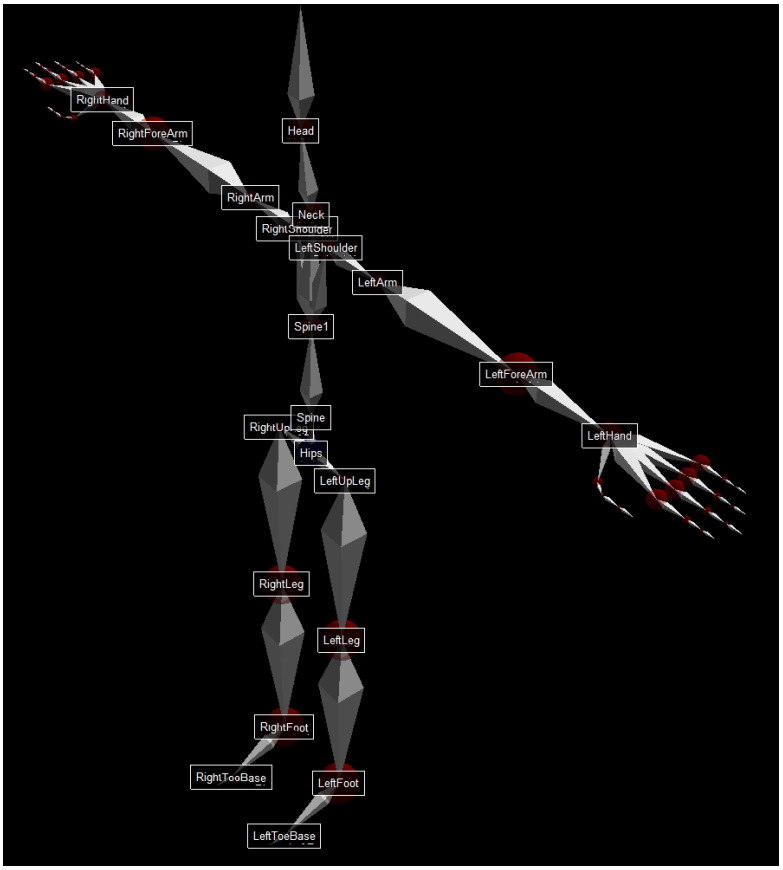
Filtering

The outliers in the data were identified by plotting each channel's data distribution in a histogram, to extract the angular limits for each joint (see Figure B.2). The data was subsequently cleaned, by applying exponential filtering and 'spherical linear interpolation' (SLERP).

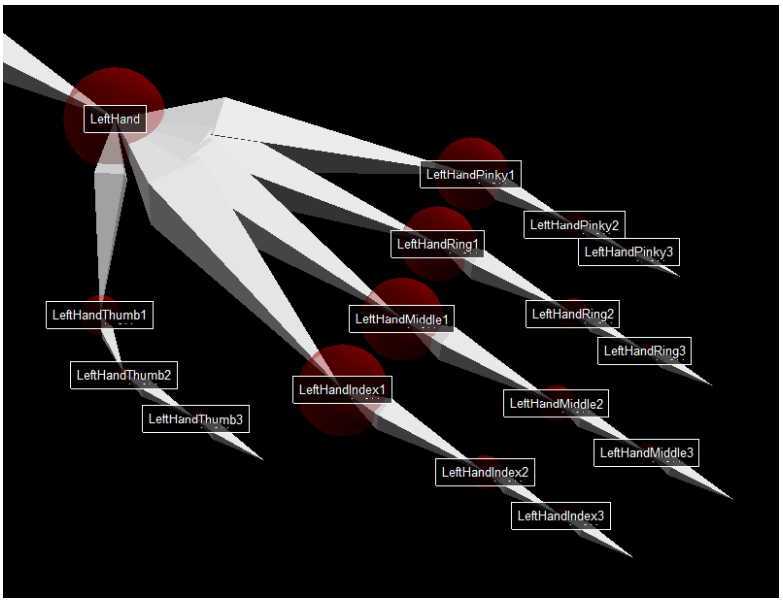
Finger Removal

During the analysis of each joint's limits it was discovered that for the finger joints only $\frac{1}{3}$ of the channels contained data (1D rotation), as each finger segment can only rotate along a singular axis and that these channels lost tracking very frequently ($20 - 70\%$). Due to the noise in the data and the additional data complexity that the finger joints entailed, it was decided to remove the fingers from the dataset and only keep the general direction of the hand. This reduced each frame's data size from 50 to 20 joints per frame or 156 to 66 floats per frame respectively; Considerably reducing the required model complexity and computation time. For the final result the loss of the finger data is not problematic, as the drone swarm is incapable of representing the fine details of the fingers.

[1]The 'dance language' has not been formalized and was left to the expertise of the choreographer.



(a) Full body Rig



(b) Hand Rig

Figure 3.1: Skeletal rig used by *AKOB*
The red spheres represent the joints of the rig
The white double pyramids, connecting every joint, represent the bones of the rig
Each of the 51 joints are labeled individually
The root of the rig is located at the 'Hips' joint

Dataset Splitting

The complete final dataset was split into three parts for the training, testing and validation of the neural networks:

1. Training (80%):
   Dataset used to train the neural network
2. Test (10%):
   Dataset used to test the neural network's performance, and optimize the hyperparameters
3. Validation (10%):
   Dataset used to validate the neural network's performance, after hyperparameter optimization

Normalization

Prior to feeding the data into any model, a final normalization step was added, which normalized the range of possible values to the range between $[-1, +1]$.

3.2. Training

A crucial initial step in designing a DL model is the decision on the framework in which the model is to be created. Based on prior experience, a large pre-existing codebase and community, the popular Python-based DL library *PyTorch* was used.

On top of the *PyTorch* library a custom framework, featuring a graphical user interface (GUI), was developed:
The 'BHPO' (Bayesian Hyperparameter Optimization - Framework).

This framework primarily aids in the following aspects:

- Live Visualization of training progress and performance
- Hyperparameter optimization
- Distributed/Cluster training

3.3. Neural Network Design

Once sufficient data was gathered and a framework for training was established, the next consideration in designing a DL algorithm was the architecture and type of learning to utilize.

Based on the literature research, it became apparent that 'Recurrent Neural Networks' (RNNs) are the best fit for the desired task. This is due to their temporal nature and their ability to 'remember', which allows the network to be fed each new data frame in real-time, without the need to store and process a large chunk containing multiple data frames.

Vanilla RNNs have a significant number of problems when it comes to long-term dependencies, therefore two alternative RNN architectures were investigated:

- Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997]
  Network with internal cell state, controlled by information flow control gates
- Gated Recurrent Unit (GRU) [Cho et al., 2014]
  Simplification of the LSTM, by removing the cell state and one of the gates

Various models, have been designed to solve the project's objective, each of which have been named *'DanceNet'*; all 13 iterations and their model variations are denoted by their respective version numbers and mode IDs.

3.3.1. Regression Model

As an initial test, a supervised regression model was developed with only a single layer RNN of the chosen type (LSTM or GRU). It was fed the current frame of the 'leading' dancer and the 'following' dancer and was then tasked to predict the 'following' dancer's pose at the subsequent time step. The regression model's architecture is visualized in Figure 3.2.

While the initial configurations performed rather poorly, the system managed to learn to perform the desired task, after a considerable amount of hyperparameter (HP) optimization steps. The best performing model was a configuration utilizing the GRU.

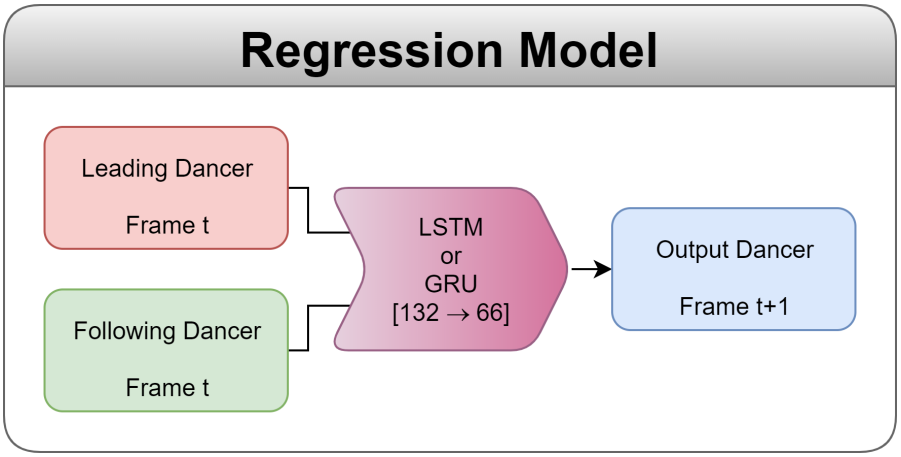


Figure 3.2: *DanceNet* Architecture: Regression Model

Trivial Solution

The main issue with this approach is that, while at first it seems a logical setup, the system does not actually need to learn any underlying complex behavioral patterns. This is because there exists a trivial solution to the problem:

Even though the underlying system is recurrent, the system is fed with ground truth data at every time step and only ever requires to predict the motion 10ms into the future. To achieve the simplest solution to this problem, all the system needs to learn is the 'Backward Euler Method', where it applies the derivative of the previous time step to predict the next state. This reduces the task to a regression of the systems inertia and completely disincentivizes the system to learn any long-term dependencies or interrelations between the two dancers. This aspect is what, in hindsight, gave the regression model its name.

Feedback Test

When placing the system into the live testing environment, there is no pre-recorded 'following' dancer, whose pre-existing pose could be exploited. Instead, the 'following' dancer's pose is replaced by the system's own output (see Figure 3.3). The combination of the overreliance on the 'following' dancer's prior pose and the sudden removal of this 'crutch' in the live test, resulted in the system displaying highly unstable dynamics.

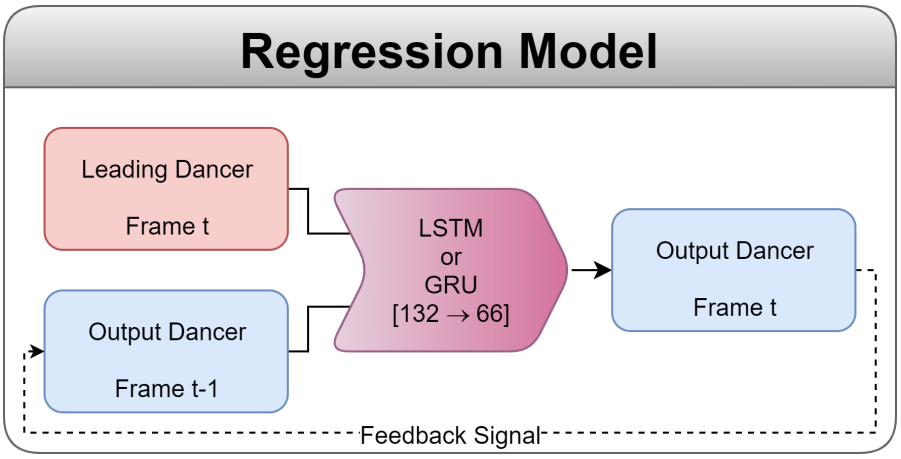


Figure 3.3: *DanceNet* Architecture: Regression Model with Feedback

In ML research the formulation of the exact task given to a system is just as important, if not more important, than the algorithm and data used to solve it. The system will always provide a solution, but it may not always be the desired one. This has proven completely true when considering the results generated by the regression model. As the regression model is inappropriate for generating long-term motions, a new task had to be defined.

3.3.2. Generative Model

As the regression model was clearly asking the wrong question, a new model was required that would be capable of generating new results, rather than 'memorizing' the training dataset, reflecting the creative nature of the desired result.

Generative Adversarial Network

A promising meta-architecture, which performed extremely well on e.g. image generation tasks, is the 'Generative Adversarial Network' (GAN) [Goodfellow et al., 2014; Shrivastava et al., 2016; Xu et al., 2018]. The primary idea behind this model is that there is no complex loss function and no singular correct answer, all that is defined and learned is whether the data is 'real/recorded' or 'fake/generated'. This is solved by training two networks simultaneously; The 'Generator' and the 'Discriminator'. A simplified visual representation of the original GAN model is presented in Figure 3.4.

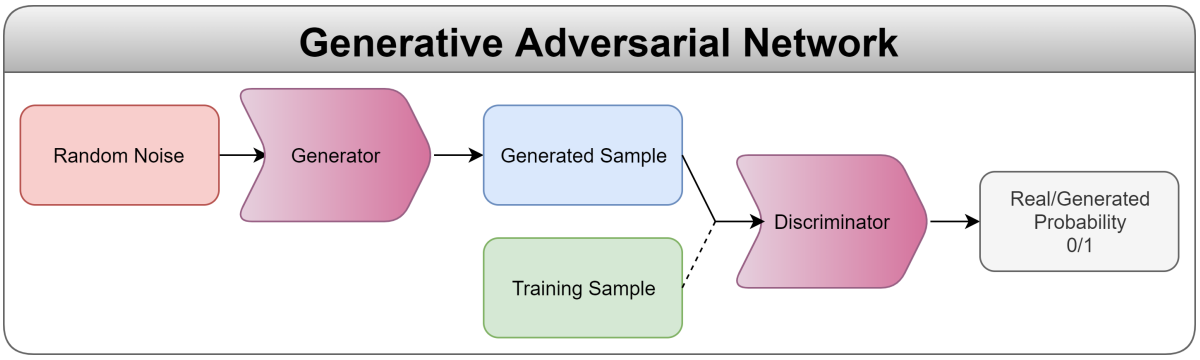


Figure 3.4: Neural network Architecture: Original GAN

This original model had to be adapted to fit the data in this research. The generator can be pictured as the 'following' dancer generating new dance motions, based on the 'leading' dancer's input. The discriminator can be pictured as the choreographer or art critic, which reviews both the original pre-recorded responses and the newly generated responses. Its job is to classify the reviewed take as fake or real (0 or 1). The generator is trained based on the feedback it receives from the discriminator until the discriminator cannot tell the difference between the pre-recorded and generated motions. The generative model's architecture is visualized in Figure 3.5.

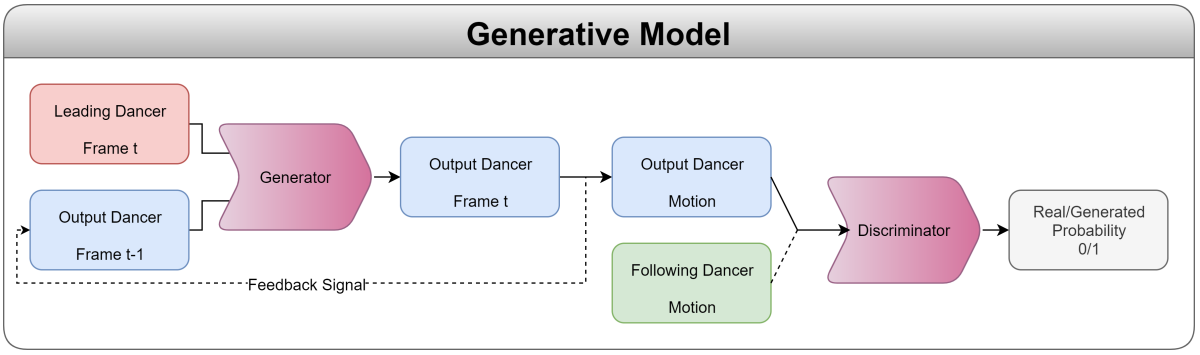


Figure 3.5: *DanceNet* Architecture: Generative Model

A GAN can be an extremely powerful and versatile tool to generate new datasets, which are indistinguishable to the original in their overall characteristics, without replicating the original dataset. The system does not learn to memorize or replicate the original dataset, but rather learns the overall style and relations of the dataset, resulting in the ability to generate truly novel output data. In this it is one of the only algorithms designed for generating truly novel data, as is required by this research's artistic context.

However, in practice, GANs are notoriously hard to train and stabilize. This is due to the 'two player' nature of this meta-architecture. The training method is essentially a game between two opposing entities, where the desired outcome is that 'the student fools the teacher'.

Intuitively, there are two scenarios in which this setup breaks down:

- The student is just too limited to ever satisfy the desires of its teacher, thus never being able to reach its full potential.
- The teacher is just too limited that it is satisfied with basically anything, thus being unable to teach the student anything new.

> *A lot of people want to use GANs, they just don't know that they're unstable, until they get into using them...and then they're kind of stuck there.*
>
> *Soumith Chintala - Facebook AI ['How to train a GAN', NIPS 2016]*

This quote by Soumith Chintala has unfortunately turned out to be true, as the system output initially was rather static and did not want to train at all, with the resulting motions being about as bad as the live feedback results from the regression model.

The largest part of this research was therefore dedicated to iteratively improving the initial GAN architecture, to prove the theoretical applicability of this model on the practical task at hand.

Model Improvements

A number of improvements to the initial GAN have been made, by uncovering and mitigating logical fallacies, such as the one uncovered in the regression model. This was done by means of inductive reasoning, based on the training results achieved after each training run.

The numerous improvements and their exact reasoning are further detailed in Part II. Here, only the changes are stated, including a brief reasoning as to why they were made, providing a chronological overview of the progress throughout the research:

1. ***DanceNet*-V0.1:**
   - Description: Basic RNN based model && Combination of both dancer motions to form singular input
   - Model ID: **GRU**
   - Reasoning: Recurrent networks were deemed most appropriate for handling temporal data. && The network should be able to compute the relation between the two motions.
2. ***DanceNet*-V0.2:**
   - Description: Initial GAN implementation && Adding attention layer
   - Model ID: **ATT-GRU-GAN**
   - Reasoning: The GAN should allow for learning the underlying 'concept' of dance. && The soft-self-attention (ATT) layer allows to focus the input data, by pre-filtering less-relevant parameters.
3. ***DanceNet*-V0.3:**
   - Description: Expansion of hidden state vector && Adding final linear layer
   - Model ID: **ATT-GRU-GAN**
   - Reasoning: The discriminator's output size of 1 resulted in a hidden state size of 1 as well, severely limiting the network's complexity. && The hidden state of the discriminator needed to be expanded in order for the system to learn complex relations over time.
4. ***DanceNet*-V0.4:**
   - Description: Cluster support
   - Model ID: **ATT-GRU-GAN**
   - Reasoning: Support to run the BHPO on multiple PCs and GPUs at once, allowing for faster hyperparameter optimization.
5. ***DanceNet*-V0.5:**
   - Description: Allowing positive gradients in final loss && Switch back to LSTM
   - Model ID: **ATT-LSTM-GAN**
   - Reasoning: Conventional early-stopping algorithms prevented the GAN from training when the loss increases, however this behavior is to be expected from GANs and had to be allowed. && As the GRU has no additional cell-state, with only a singular layer it is impossible to 'remember' temporal relations longer than one time step ago.
6. ***DanceNet*-V0.6:**

- Description: Increase in RNN layer depth
- Model ID: **ATT-LSTM-GAN**
- Reasoning: A singular recurrent layer was sufficient for the regression model, due to the trivial solution, however for learning more complex interdependencies a deeper network structure is required.

7. ***DanceNet*-V0.7:**
   - Description: Pre-conversion of Euler angles, to 6D-rotation representation
   - Model ID: **6D-ATT-LSTM-GAN**
   - Reasoning: Zhou et al. have proven that neural networks perform suboptimal when providing (possibly) non-continuous input parameters; such as Euler angles, but provided a 6D representation of angles in 3D space perform significantly better.
8. ***DanceNet*-V0.8:**
   - Description: Prediction of state derivative, instead of complete pose
   - Model ID: **6D-ATT-DO-LSTM-GAN**
   - Reasoning: 'Derivative Output' (DO): Due to the continuous nature of the desired output, predicting only the state change at each time step instead of the entire pose, should allow the system to focus more on the desired dynamics, while simultaneously avoiding large jumps in the output data.
9. ***DanceNet*-V0.9:**
   - Description: Pre-computation of derivative
   - Model ID: **6D-ATT-DI-DO-LSTM-GAN**
   - Reasoning: 'Derivative Input' (DI): The state derivative is an important metric, as it is by definition the 'motion' which is desired. So instead of feeding the neural network only pose information and forcing it to learn the concept of velocity by itself, the state change from frame to frame is pre-computed and concatenated to the original input vector.
10. ***DanceNet*-V1.0:**
    - Description: Added original input to discriminator input
    - Model ID: **6D-ATT-DI-DO-LSTM-GAN**
    - Reasoning: So far the discriminator was only judging the output motion of the generator, however this prevents it from judging the motion in context. Therefore the original 'leading' dancer's motion was concatenated to the discriminator's input data.
11. ***DanceNet*-V1.1:**
    - Description: 'Pain' criterion
    - Model ID: **6D-ATT-DI-DO-LSTM-Pain-GAN**
    - Reasoning: Humans need to learn to crawl, walk and dance, however they do not need to be explicitly taught their physical limitations. Angular limits of the joints are unavoidable, except by enduring a large amount of pain. By pre-computing the distribution limits for each joint, these limits were mimicked in the model by adding an extremely high loss (pain) when these limits are exceeded.
12. ***DanceNet*-V1.2:**
    - Description: Utilization of 'Principal Component Analysis' PCA
    - Model ID: **6D-PCA-ATT-DI-DO-LSTM-Pain-GAN**
    - Reasoning: The PCA transform was applied, to enable the inspection of the general motions present in the dataset. Additionally, it is an excellent and fast tool to normalize and 'whiten[2]' the data prior to processing.
13. ***DanceNet*-V1.3:**
    - Description: Added 'Limited-Judgement'
    - Model ID: **6D-PCA-ATT-DI-DO-LSTM-Pain-LJ-GAN**
    - Reasoning: So far the discriminator was fed the entire output of the generator from $t = 0$. Due to the initialization of the network the first frame of the generator is always the same, this means that the discriminator was able to immediately pick up this irregularity and judge based on the initial state. To avoid this the discriminator was given 'Limited-Judgement' (LJ) by only feeding him the motion past a certain number of frames of output data, to allow the generator to obtain a realistic pose before being judged.

[2]Covariance matrix is the identity matrix

Given the ever expanding model ID and the fact that the final proposed model turned out to be more complex than initially expected, a new name for the entire model architecture was required. The final name for the proposed model is the 'Differential Recurrent Attention GAN', or **'DRA-GAN'** for short.

3.4. Results

The initial regression model was simple and effective, it performed the desired task accurately and proved that the BHPO training framework is functioning as expected. The GAN has proven to be significantly harder to train than initially expected, however the gradual steps towards the final model have not only improved the output, but also provided insight into the system's behavior.

However, the final output motions, generated by the DRA-GAN model, are not of sufficient quality to be considered long-term and interactive. The output motion is improving throughout the training run, but visually does not appear to be a fully correct human pose. The output motion largely ignores the leading dancers input and outputs mostly the same pose after a few seconds. This failure mode of GANs is commonly known as 'mode collapse' [Liu and Tuzel, 2016].

3.5. Live Interaction

To allow the system to be applied in the *AI-man* showpiece, a simulation was created that processes the pre-trained model in real-time and transmits the computed output motion via a custom UDP broadcast, for other sub-systems to utilize the resulting data.

To achieve this, a module was created that transforms the serialized motion-data stream, as broadcasted by *OptiTrack*'s *NatNet* system, to the desired input, as required by the neural network.

To test and visualize the real-time process, two visualization systems have been developed:

1. VPython:
   A simple debugging GUI for visualizing the skeletal motions
2. Unreal Engine 4:
   A visually appealing high-performance visualization, that could also be applied in a VR environment

Note: Please refer to Appendix A for definitions of all coordinate system definitions used by the various programs throughout the research.

3.6. Discussion

The research field of long-term motion generation by utilizing neural networks is relatively new and at the current cutting edge of technology. Most papers found on the subject are from the last five years, with only few exceptions, and significant advances in the field have even been made throughout the duration of this research. Many researchers consider time frames of more than a second to to be 'long-term' [Ghosh et al., 2018; Gui et al., 2018a; Tang et al., 2017], while this research attempts to generate motions of more than a minute.

Applying the principles of the grounded theory approach, the architecture of the final network was devised iteratively given continuous inductive reasoning, based on insights gathered throughout the research project. Unfortunately, the final output of the model is not as desired, but the research leading up the final model has proven incremental improvements and the full potential of the final DRA-GAN model remains yet unexplored. While originally, a detailed comparison and benchmarking of various neural network architectures was intended, this has proven to be impractical given the time constraints, due to the long training durations of recurrent neural networks of this scale. With training times of multiple weeks, the same holds true for fully evaluating a sufficient amount of hyperparameter sets, to assess the DRA-GAN's full potential.

The contribution to the scientific community lies in the design of a novel neural network architecture, for the application of motion synthesis, as well as various algorithms and criteria to support the training of RNN based GANs. For future researchers an extensive list of potential improvements to the current model has been compiled, to allow for continuation of this research; hopefully enabling the *AI-man* project to be showcased to the world.

II

Research & Development

4

Data Handling

"Data is key. Humans cannot learn without input, neither can machines."

This chapter describes the data used to train the subsequently developed neural networks; the acquisition procedure, as well as the performed data cleaning and pre-processing steps.

The given project objective falls generally under the fields of research 'Motion Synthesis' and 'Motion Prediction'. The application of neural networks in these fields, for the generation or prediction of human motion, is by itself not novel and various papers can be found on this topic [Gaisbauer et al., 2019; Holden et al., 2017a; Martinez et al., 2017; Pavllo et al., 2019; Wang et al., 2020a,b; Zhou et al., 2019a]. Due to the specific nature of the assignment, requiring close interaction of two dancers over an extended period of time, the number of applicable prior research is narrowed down significantly to only a handful[1]. However, these papers only cover one or a few of the required aspects, and never all at once. The same is true for the datasets used in these papers, which is why a custom dataset was acquired.

4.1. Data Acquisition

A total of 9.2 hours of motion capture (MoCap) footage were recorded on 6 days, over the course of 2.5 weeks, in 147 individual takes.

The acquisition of the required dataset was primarily executed by *AKOB*. The researcher was not present at the times of the recording due to COVID-19 restrictions.

4.1.1. Public Datasets

Given the application of the final research in a real-time motion capture (MoCap) environment and the study of the human form, the only logical conclusion was the usage of MoCap data for this research and hence the requirement to source a suitable database.

A lot of research in the field is utilizing pre-existing datasets such as the *'CMU Graphics Lab Motion Capture Database'* (*CMU*) [CMU Graphics Lab, 2016] or the *'Human3.6M Dataset'* (*H3.6M*) [Ionescu et al., 2014]. These datasets mainly consist of short ($< 20s$) takes of individual people performing various clearly defined tasks. Subsequently, they are being used for research, such as classification of the performed action or prediction of the remainder of a cut-off take [Bütepage et al., 2017; Gui et al., 2018a,b; Hernandez et al., 2019; Li et al., 2019; Mao et al., 2019; Martinez et al., 2017; Tang et al., 2017].

Due the typically small size of available MoCap datasets and the lack of standardization between them, the *'Archive of Motion Capture as Surface Shapes'* (*AMASS*) project took 18 commonly used datasets and combined them into a singular database, for easier usage by the research community [Mahmood et al., 2019].

However, out of all these publicly available datasets, there was none which fulfilled all required criteria:

- MoCap
- (Modern) Dancing
- Two interacting people/dancers

Due to this extremely specific nature of the required data, the generation of a custom dataset was required.

[1]The full literature study and the most relevant papers are presented in the preliminary report (see Appendix F).

4.1.2. Procedure

The setup for a recording session is as follows: Two dancers and the choreographer of *AKOB* are located within their motion capture studio. One dancer is considered the *'leading/input dancer'*, while the other is considered the *'following dancer'*. The 'leading dancer' takes the lead in the improvisation, while the 'following dancer' reacts to their partner accordingly. In the actual showpiece the 'leading dancer' will remain human, while the 'following dancer' will be replaced by the 'drone swarm dancer'.

AKOB utilizes a motion capture system by *OptiTrack*, with 16 *Prime 13* infrared (IR) cameras[2], running at $100Hz$. For human motion tracking, Velcro suits with 54 passive IR markers (reflective spheres) are used. The company and dancers are well experienced with the system, and have used it for various other projects in the past.

For each take the dancers would start in the standard 'T-Pose'[3]. The choreographer would pick a music track of his choice and give a few guidelines to the dancers that they have to take into consideration during their 2-6 minute long improvisations. The distribution of take durations is visualized in Figure 4.2.

A visual impression of the data acquisition process is presented in Figure 4.1.

Figure 4.1: Behind the scenes at *AKOB*s studio, during data acquisition
Source: *Stichting Another Kind of Blue*

Dancers:	Leading Dancer 'pushing' following dancer, wearing marker costume[4]
Left Side:	*OptiTrack Prime 13* camera lighting up
Bottom:	*OptiTrack Motive*, for recording the digital MoCap take
Lighting:	As the system used IR, the influx of sunlight was limited to reduce IR interference

4.1.3. Anonymization

In accordance with the requirements of the TU Delft's *Human Research Ethics Committee* (*HREC*), the following measures have been taken to ensure anonymity of all data within the dataset:

1. Replacement of dancer names with alphabetical symbols: Dancer *A* & Dancer *B*
2. No audio or video footage was recorded, only skeletal reconstructions

[2] A system very similar to the one being used at the *CyberZoo* of the TU Delft Faculty of Aerospace Engineering
[3] Standing straight, facing forward, with arms spread to the side, see Figures 3.1 and 4.4

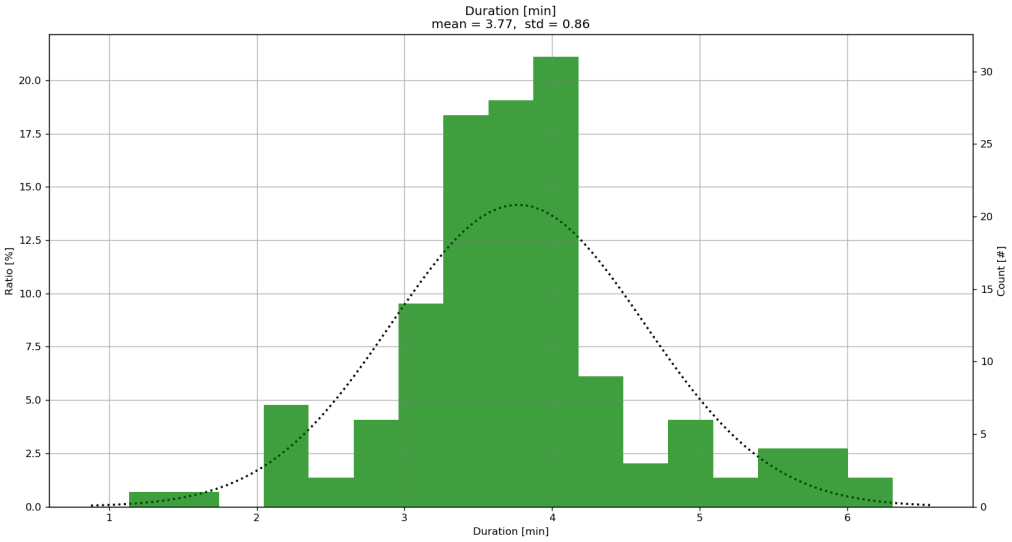


Figure 4.2: *AKOB* Dataset: Distribution of take durations
Dotted line indicates best-fit for a Gaussian distribution on this data

4.1.4. Filename Convention

A consistent naming schema for all takes was defined and communicated with *AKOB*, to ensure that all required information was contained in the filename of each take. The following naming schema was devised:

'Take YYYY-MM-DD HH.MM.SS_Role{L,F}DancerSymbol{A,B}_Label#1,Label#2[,Label#N]_Description(_Take#).EXT'

e.g.

'Take 2020-05-13 11.30.00_FA_ball, slow, mirror_feeling good muse_001.bvh'

The filename contains the following information, in the stated order:

1. Date: **'YYYY-MM-DD'**
2. Time: **'HH.MM.SS'**
3. Dancer Role: **'{L,F (,A)}'** L='Leading', F='Following' and A='Alternating[5]'
4. Dancer Symbol: **'{A,B (,C,D)}'** [6]
5. Labels: Variable number of labels, to store information on the intention behind the take
6. Description: General information from the choreographer, mostly the choice of music for each take
   ↓ Optional ↓
7. Take Number: **'{001,002,...}'** Enumeration, if the same labels and description were used

Labels

The choreographer was asked to develop a set of labels for his intentions, which should be reused to enhance the dataset's uniformity. The addition of these labels would also allow the application of classification algorithms on the data, to augment the motion prediction/generation tasks.

Unfortunately, it was not possible to convince the choreographer to plan ahead and create a limited set of consistent labels, which should be reused approximately in equal fashion. Instead he decided to approach the data acquisition procedure in a creative manner; adding labels 'on-the-go'. This resulted in a highly biased label distribution, as can be seen in Figure 4.3. 14 out of 23 labels are preset in less than 10% of all takes, while e.g. the 'slow' label is present in over 40% of all takes. This inconsistency makes it hard to use the labels as an objective and unbiased training aid.

[5]The label 'Alternating' was required, because the choreographer initially did not properly communicate the role division to the dancers. For training of 'Alternating' takes the leading and following roles were assigned at random.
[6]Dancers 'C' & 'D' were only present in 4 out of 147 takes, which have been removed to keep the dataset consistent')

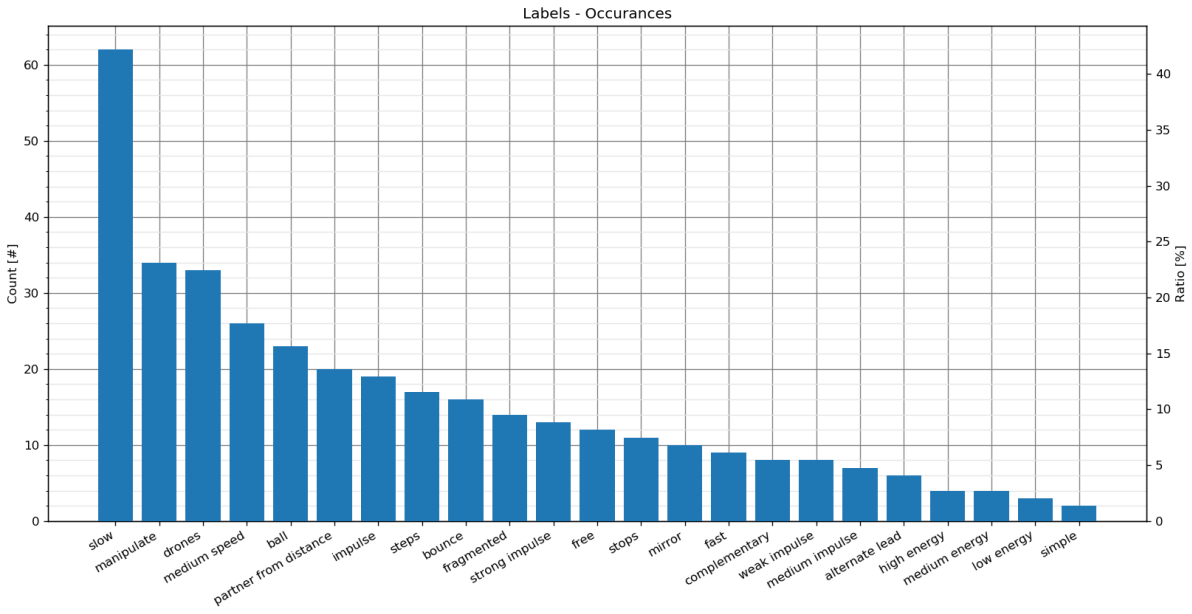


Figure 4.3: *AKOB* dataset: Distribution of labels

4.2. Data Structure

Understanding the format of data used in this research is key to ensure comprehension of the rationale behind subsequent analysis and pre-processing steps. This section provides an overview of the format and standards of the MoCap data used in this research.

4.2.1. The (Digital) Human Body

Character animators and MoCap researchers are at the crossroads of technical and intuitive understanding of the human body and have come up with various means of digitizing human bodies and their motions[7].

For digitizing and animating any dynamic entity two types of data are used:

- Mesh:
  3D tessellation as approximation of the general shape of the entire body, analogous to the outer skin.
- Skeleton/Rig:
  Simplified digital representation of the actual skeleton, reduced to the basic components required for movement.

An example representation of a humanoid character is given in Figure 4.4; the green hull represents the mesh, while the grey polyhedrons and spheres represent the skeleton.

The mesh without a skeleton to move it is called a 'static mesh' and is incapable of changing its own form. A skeleton defined for a specific mesh allows for dynamic manipulation of the static mesh, but is not constrained to that mesh and can be utilized to animate multiple meshes, as long as the general structure and layout is sufficiently similar. A rig defined for a quadruped, such as a dog, will prove hard to transfer to a humanoid mesh.

Unlike related motion generation research, that focuses on the 3D visual aspect for animations [Holden et al., 2015, 2017b; Mahmood et al., 2019; Shen and Yang, 2016; Yamane et al., 2010], this work focuses only on the generation of humanoid motion. This is due to the fact that, the resulting skeleton is independent of the mesh used for visualizing it, as the final product will be represented by drones (points) in space and not by a character on a screen. Therefore, the remainder of this research will focus solely on the skeletal rig.

General Formats

There are various means to define the format for the skeletal rig and its motion, but almost always it is defined in a kinematic chain in a hierarchical fashion. Figure 4.5 illustrates a kinematic chain in 2D and the various

[7]For extended background information, please refer to Section 3.2 ('Character Animation') of the preliminary report (see Appendix F).

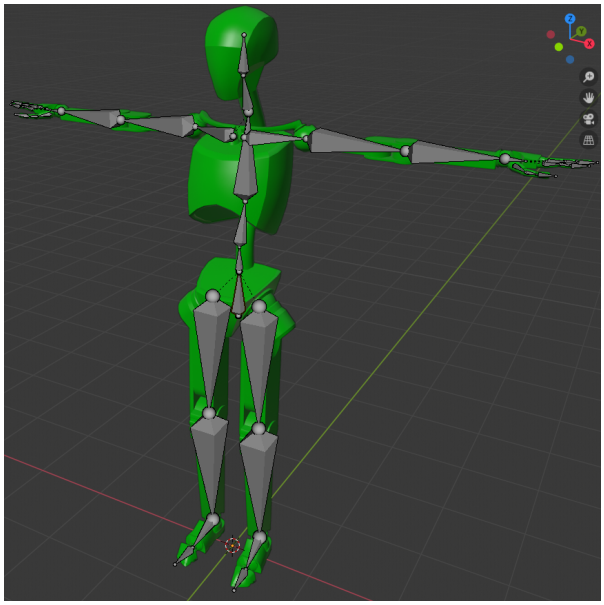


Figure 4.4: Rigged example character mesh in *T-Pose*
Mesh: Outer shape, in bright green
Skeleton/Rig: Inner structure, in grey (overlay)

formats that describe the configuration. For the actual skeletal rig this example has to be extended from 2D to 3D, resulting in three rotational axes per joint, compared to the singular one illustrated here. The second simplification made in this example compared to a humanoid skeletal rig is the absence of chain splitting, e.g. the kinematic chain at the Root ('Hips') splits in three individual segments, one for each of the legs and the spine (see Figure 3.1). The same is the case for the split at 'Spine1', splitting the chain into three individual segments again, one for each of the arms and the head. Up until a split the kinematic chain is shared, independent of the end effector.

Primarily there are two clear distinctions between prominent formats of how the joints or bones are defined:

- Position: 3D positions in space for each joint define the length and orientation of each bone

vs

- Angle: Bone lengths are pre-defined and each joint is fully defined in terms of its orientation

&

- Absolute: The positions/angles are all defined with respect to the global coordinate system

vs

- Relative: The positions/angles are defined relative to their predecessor in a hierarchical layout

Position vs Angle In terms of visualization, a position-based format is desired, as it allows for direct translation of the data to the correct position in space or on screen, while an angular approach requires prior re-computation of the positions before visualization.

A downside of the position-based approach is that the constant length of individual limbs/bones is not guaranteed and has to be enforced separately. Especially for neural networks, which can produce rather unexpected outputs, especially early on in the training process. Here, it is beneficial to utilize formats that prevent breaking physical constraints of the human body as much as possible.

Given the desired versatility of the final product, enabling it to be utilized with various dancers and hence bodies, it was chosen to use a format solely utilizing angles to allow for its application onto various body statures. In dance a certain motion is mostly defined by the relative angles of the limbs and not their absolute position in space. A dwarf and a giant can both showcase a pirouette (ballet turn) and it will still remain a pirouette, regardless of their size. There are limitations to this approach, when it comes to certain distributions of bone lengths that deviate a lot from the average distribution. It can however be reasonably expected that

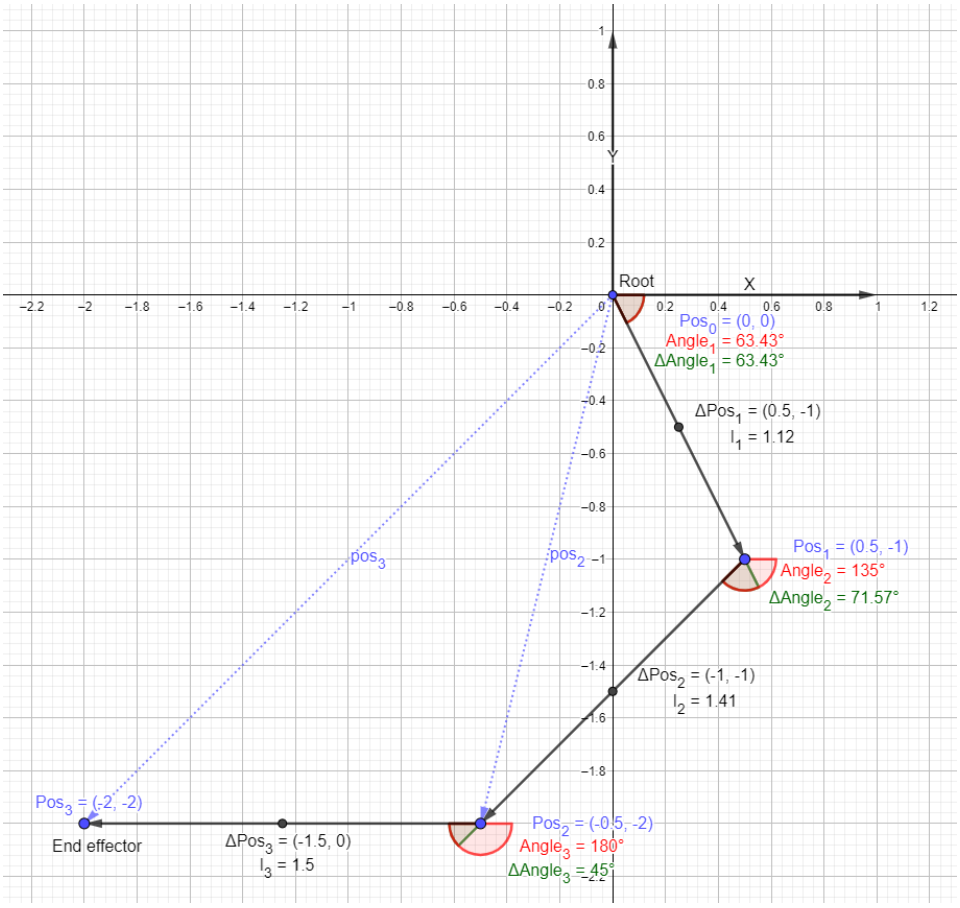


Figure 4.5: Kinematic Chain Example - 3 Bones in 2D
Red: Absolute Angles | Green: Relative Angles | Blue: Absolute Positions | Grey: Relative Positions + Bone Lengths

the research will not be utilized with physically impaired individuals in the present state of the development. Adding information on the exact size of the dancer could be added in future research. The validity of this assumption is further manifested by the fact that in the dataset, as well as the final product, there have been and will be no physical touch-based interactions between the two dancers, hence reducing the requirement for exact synchronous positioning, e.g. touching hands.

Absolute vs Relative Analogous to the 'position vs angle' comparison, absolute values allow for a more direct visualization and require less or no additional pre-computations.

When inspecting the human body, relative angles are easier to comprehend for humans and machines alike. Defining the relative angles between two joints allows for easy verification whether any limits are exceeded. For machine learning, relative angles offer a distribution with a clear mean and limited discontinuities.

4.2.2. Biovision Hierarchy (BVH)

The 'Biovision Hierarchy' (<.BVH>) format was chosen as the initial data format for the data acquisition; out of the 6 formats compared in the preliminary report. Additionally, each take was also stored in *OptiTrack*'s native <.TAK> format, to allow for later conversion to other MoCap formats, if desired.

The <.BVH> format combines numerous advantages that are beneficial to the application of this research:

1. Angle-based formatting: (see Section 4.2.1)
2. Relative angles: (see Section 4.2.1)
3. Easily interchangeable skeleton: Same data can be visualized on multiple meshes
4. Easily adaptable limb sizing: Easy to adapt spacing of drones in swarm
5. Simple 2D array of floats: Pre-formatted for training neural networks - faster parsing

The <.BVH> is an easily readable ASCII based file. Positions are defined in meters, angles are defined as Euler angles in degrees. The file consists of two segments: The '**HIERARCHY**' segment defines the skeletal structure, with each joint defined relative to its predecessor[8], starting at the **ROOT** (Hips). The '**MOTION**' segment defines the actual animation data as an array of floats, defining the relative rotations for each joint in the order defined by the hierarchy.

The structure of the file is as follows[9]:

- **HIERARCHY**: Skeletal Data
  - Hierarchical skeletal definition[10] of the '**ROOT**' and each subsequent '**JOINT**', containing info on:
    - Name of the joint
    - **OFFSET**:
      Vector defining positional offset from the current joint to the previous joint
    - **CHANNELS**:
      Number and order of (positional and [**ROOT** only]) rotational channels for this joint
    - **End Site**:
      If the joint is the last element in its hierarchical chain it defines the '**OFFSET**' of the final bone
- **MOTION**: Animation Data
  - **Frames**: Total number of frames in the file
  - **Frame Time**: Time step between frames
  - Frame-by-frame poses:
    - Separate line for each frame (pose)
    - Array of float separated by spaces
    - Follows the hierarchical definition of channels, as defined in the '**HIERARCHY**'
      - 3 floats for the global position
      - 3 floats for the global orientation
      - 3 floats for the relative angles of each joint defined, ordered down the hierarchical chain

While theoretically the <.BVH> format is capable of storing multiple skeletons in the same file, in practice almost no program is compatible with this functionality. For this reason, two <.BVH> files were created per take, containing the recordings for the leading dancer and the following dancer respectively.

4.2.3. *ArangoDB*

The raw <.BVH> files provided by *AKOB* had to be parsed and processed to enable easier usage in the development stage. For this purpose a local *ArangoDB* database was created. *ArangoDB* is a graph database that allows for storing two types of collections:

- Document Collection: Documents in JSON format.
- Edge Collection: Relations between documents (e.g. hierarchy), for graph visualization and querying

Considering that the size of the final dataset in <.BVH> was over 12 GB, it was deemed practical to separate the large motion data arrays from the smaller meta-data. This allows for querying of meta-properties, like the total number of frames/takes or duration, without loading all the motion data into RAM.

The collections that have been created and the data they contain are as follows; the first level is just for logical separation and for clarifying the use case of each collection:

- Organization:
  - Phases: General information on each recording phase
  - Days: Date and IDs of takes recorded on each day (ID example: **2020.05.01**)
- Data:
  - Takes: General meta-data of each take (ID example: **2020.05.01-15:53**)
    - Date: Date of recording

[8] Example hierarchy: RightShoulder→RightArm→RightForeArm→RightHand
[9] A full description of the <.BVH> format can be found at https://research.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/BVH.html
[10] The hierarchical definition reminds strongly of the <.JSON> format, but is not directly compatible.

        - Time: Time of recording
        - RecordingIDs: Downwards reference to the two recordings that make up this take
    - Recordings: Meta-data of an individual motion recording (ID example: **2020.05.01-15:53_{L&F}**)
        - TakeID: Upwards reference to the take this recording belongs to
        - SkeletonID: Downwards reference to the skeleton that belongs to this recording
        - MotionID: Downwards reference to the motion that belongs to this recording
        - MetaData: Info on this recording
            - Role: {'Following', 'Leading'(, 'Alternating')}
            - Dancer: {'A', 'B'(, 'C', 'D')}
            - NrFrames: Total number of motion frames
    - Motions: Raw/original motion data [Largest Collection]
        - NrFrames: Total number of motion frames
        - NrValues: Number of floats per motion frame
        - FPS: Frame rate in Hz
        - Motion: 2D array of motion data; Size=[NrFrames x NrValues]
    - Motions_Cleaned: Pre-cleaned copy of 'Motions'; same parameters as 'Motions' [Actual Dataset]
    - Skeletons: Hierarchical body structures for each recording
        - ID: Symbol for the dancer this skeleton belongs to
        - Skeleton: Nested skeletal structure, analogous to the <.BVH> **'HIERARCHY'**
- Analysis
    - AnalysisResults: Results obtained by analysis of the dataset, containing the following documents
        - JointHistogram: Histogram counts for every joint's channels, combining all 'Motions'
        - JointInfo_Original: Automatically extract joint limits, based on the 'JointHistogram' data
        - JointInfo: Manually reviewed and adjusted copy of 'JointInfo_Original' [Used by model]
        - PCA: PCA transformation data, based on 6D 'Motions_Cleaned' [Used by model]
    - Relations: Edge collection, for storing the hierarchy between all documents
- BHPO:
    - BHPO_Results: Hardware info and optimization results for every BHPO run
    - BHPO_Suggestions: HyperParamater suggestions for the BHPO Cluster

A visual representation of the relational graph structure inside the database is presented in Figures 4.6 & 4.7. These figures showcase the hierarchical nature of the data, as the data acquisition has taken place on multiple 'Days', each with multiple 'Takes'. Every 'Take' has exactly two associated 'Recordings', both containing references to the 'Skeleton' and 'Motion' data that define them.

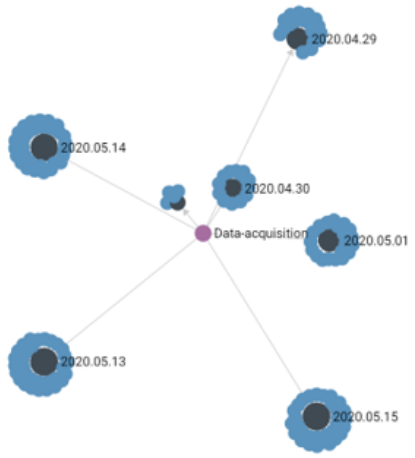


Figure 4.6: *ArangoDB*: Data acquisition → Recording day → Takes

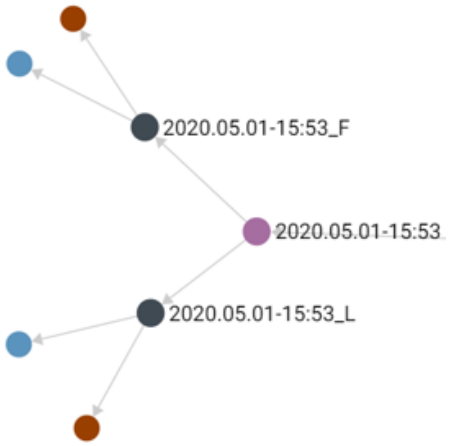


Figure 4.7: *ArangoDB*: Take → Recordings → Skeleton + Motion

4.3. Data Cleaning

Although the current state of MoCap technology is impressive, it is far from perfect and a lot of advanced research is being done to automate cleaning procedures of MoCap data. This includes the application of neural networks for the cleaning and re-rigging of MoCap data [Holden et al., 2015, 2017b; Shen and Yang, 2016; Yamane et al., 2010], however this laborious process is often still done manually. Advanced MoCap cleaning algorithms are however not the core of this research and a simplified algorithm was devised, in an attempt to remove the most problematic outliers in the data, as described in the following section.

Given that most deep learning (DL) algorithms are effectively black-box systems, it is extremely hard to identify and resolve even minor issues after the fact. It is thus paramount that the data, on which the network is being trained on, is analyzed upfront, cleaned and clearly understood.

4.3.1. Limit detection

Determining the distribution and limits of the state of each joint in the body is an important first step for detecting outliers in the data, as well as gaining a better understanding of the statistical properties of our dataset.

The fact that the data is defined as relative angles to each previous joint, allows for an easy channel-by-channel[11] analysis, without the need for prior transformation of the data.

A fully automated algorithm was devised to analyze the data and detect the limits and outliers for each joint. If the data were 100% accurate, detecting the limits would simplify to a min/max operation, however due to the presence of outliers this is not possible. Therefore, a histogram was created for each channel and the limits were chosen based on a probability threshold. The resulting histograms are shown in Figure 4.8, with a full-scale version available in Appendix B (see Figure B.2). A singular example of the 'LeftArm' joint is presented in Figure 4.9.

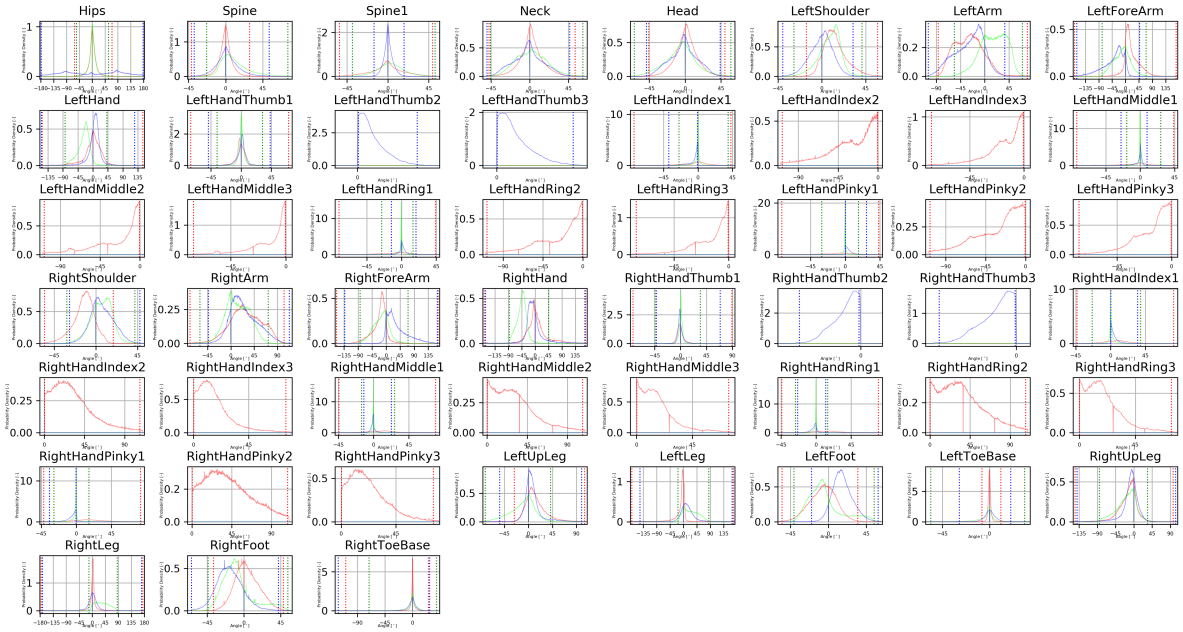


Figure 4.8: Parameter Distributions: Relative joint channel histograms
Full-scale version is available in Appendix B (see Figure B.2)

The first thing to note is that the parameters are not normally distributed. This means that common measures like '$\mu \pm X \cdot \sigma$' cannot reliably be used for outlier detection. Having a clear analytical representation of each parameter's distribution allows for proper quantitative parameterized analysis. An extensive regression analysis was performed, attempting to find the best analytical fit based on Probability Density Functions' (PDFs). Although the distribution of angles per joint is not an independently sampled stochastic variable, this

[11] Each joint consists of three data channels: Roll (ϕ), Pitch (θ) & Yaw (ψ)

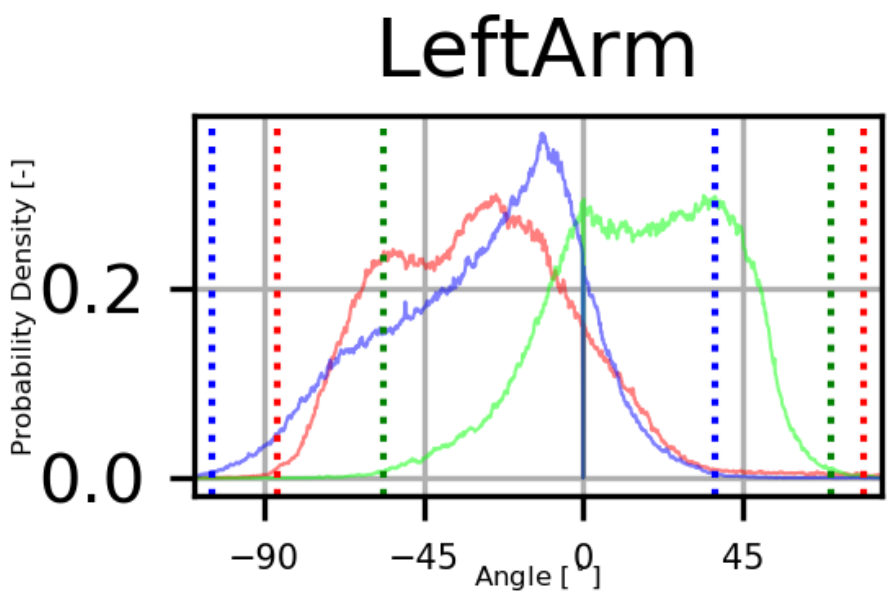


Figure 4.9: Parameter Distributions: LeftArm histograms
Red: Roll (ϕ) | Green: Pitch (θ) | Blue: Yaw (ψ)
Continuous lines: Distributions | Dotted lines: Limits

analysis does confirm that for most joints a standard Gaussian distribution is indeed a sub-optimal fit. In general, symmetrical joints (e.g. head and spine) are best characterized by the symmetrical Laplace distribution; while asymmetrical joints (e.g. RightShoulder) are best described by the (potentially) asymmetrical skew normal or Exponentially modified Gaussian (EMG|ExpoNorm) distributions. The histograms and details of the PDF regression analysis can be found in Appendix B.

The results from this analysis are in line with the intuitive expectations of the human body and are an analytical verification of the integrity of the dataset.

Due to the non-normality of the data, the means of detecting the joint limits was to threshold the resulting histograms at $\pm 0.25\%$ of the total count. For a generic continuous PDF this would be equivalent to the region where the 'Cumulative Distribution Functions' (CDF) is between $[0.25\%, 99.75\%]$[12].

The computed statistics per channel, as well as the percentage of detected outliers can be seen in Table 4.1. Given the large number of samples, these limits provide a good estimation for the overall limits of the human body. Furthermore, the key parameters presented in the table allow for assessing a few characteristics of each joint:

- $|Mean - Peak| \approx 0$: Distribution is symmetrical, with probably a singular mode
- $Min\&Max \approx \pm 180°$: Channel allows for all possible values
- Invalid $\gg 0.5\%$: System had problems tracking this channel, the data should not be trusted

The main observations from this data are that most finger channels are highly unreliable and that the left and right joint channels are not perfectly symmetrical.

Manual Review

Based on inspecting the histograms and detected limits a few logical manual modifications to the detected limits were made:

- Knowing the symmetry of the human body, the limits of the joints were forced to be symmetrical as well, by selecting the extrema of the two options.
  This extended the limits by 4.6° on average.
- Any restrictions to the root joint (hips) were removed, to allow full 6-DOF freedom in space, to allow for extending the dataset with dancers in any position.

[12]...or for a normally distributed parameter a cut-off at $\approx 3 - 3.5\sigma$

JointName	Min [°]	Peak [°]	Mean [°]	Max [°]	Range [°]	Invalid [%]
Hips-ϕ	-61.4	-0.2	1.4	69.6	131.0	0.5
Hips-θ	-55.0	0.4	1.3	57.4	112.4	0.5
Hips-ψ	-178.2	86.4	2.1	178.0	356.2	0.6
Spine-ϕ	-41.4	0.4	-0.5	29.0	70.4	0.5
Spine-θ	-23.0	1.8	11.0	75.6	98.6	0.5
Spine-ψ	-38.2	0.8	2.1	53.0	91.2	0.5
Spine1-ϕ	-53.2	-0.2	-1.1	49.4	102.6	0.5
Spine1-θ	-39.0	2.4	3.4	52.0	91.0	0.5
Spine1-ψ	-15.4	-0.2	0.5	17.8	33.2	0.5
Neck-ϕ	-47.4	1.0	1.9	52.6	100.0	0.5
Neck-θ	-45.6	1.2	2.1	61.6	107.2	0.5
Neck-ψ	-49.6	-1.0	-2.0	46.4	96.0	0.5
Head-ϕ	-45.8	1.6	1.2	49.8	95.6	0.5
Head-θ	-63.8	-1.8	-5.9	57.2	121.0	0.5
Head-ψ	-48.6	-1.6	-2.1	44.0	92.6	0.5
LeftShoulder-ϕ	-16.0	9.2	13.6	57.2	73.2	0.5
LeftShoulder-θ	-28.8	14.6	8.0	41.2	70.0	0.5
LeftShoulder-ψ	-40.2	3.6	-0.5	32.0	72.2	0.6
LeftArm-ϕ	-86.4	-24.6	-29.0	79.0	165.4	0.5
LeftArm-θ	-56.4	34.0	16.2	69.8	126.2	0.5
LeftArm-ψ	-105.0	-11.4	-28.9	37.2	142.2	0.5
LeftForeArm-ϕ	-171.6	6.6	19.7	170.2	341.8	0.5
LeftForeArm-θ	-79.0	-0.6	-12.5	69.0	148.0	0.5
LeftForeArm-ψ	-163.8	-21.8	-40.5	47.4	211.2	0.6
LeftHand-ϕ	-152.0	2.6	-0.5	149.6	301.6	0.6
LeftHand-θ	-83.4	-20.2	-27.6	41.6	125.0	0.5
LeftHand-ψ	-154.4	8.0	4.2	125.6	280.0	0.6
LeftHandThumb1-ϕ	-75.0	-1.6	-1.3	70.8	145.8	0.7
LeftHandThumb1-θ	-36.0	0.2	0.6	31.8	67.8	0.6
LeftHandThumb1-ψ	-49.0	1.6	0.6	43.4	92.4	0.8
LeftHandThumb2-ψ	0.2	1.0	3.5	13.4	13.2	40.8
LeftHandThumb3-ψ	0.2	2.0	7.0	27.4	27.2	39.6
LeftHandIndex1-ϕ	-83.4	3.8	-5.5	43.0	126.4	4.1
LeftHandIndex1-θ	-25.0	0.2	-0.4	39.2	64.2	9.4
LeftHandIndex1-ψ	-38.0	-0.4	-5.7	-0.4	37.6	66.6
LeftHandIndex2-ϕ	-108.0	-1.0	-31.6	-0.4	107.6	49.7
LeftHandIndex3-ϕ	-76.6	-1.0	-17.5	-0.4	76.2	48.4
LeftHandMiddle1-ϕ	-82.8	0.4	-7.4	44.0	126.8	1.1
LeftHandMiddle1-θ	-17.6	-0.4	0.6	27.0	44.6	20.0
LeftHandMiddle1-ψ	-25.0	0.2	-1.3	9.0	34.0	0.9
LeftHandMiddle2-ϕ	-108.2	-1.6	-28.0	-0.4	107.8	30.9
LeftHandMiddle3-ϕ	-75.2	-0.8	-15.4	-0.4	74.8	30.8
LeftHandRing1-ϕ	-81.2	-1.0	-8.1	44.4	125.6	0.8
LeftHandRing1-θ	-25.6	-0.4	-0.0	14.8	40.4	16.0
LeftHandRing1-ψ	-13.2	-0.2	1.1	18.8	32.0	1.0
LeftHandRing2-ϕ	-108.2	-1.4	-29.8	-0.4	107.8	30.6
LeftHandRing3-ϕ	-76.0	-0.8	-16.2	-0.4	75.6	30.4
LeftHandPinky1-ϕ	-82.4	-1.6	-8.8	43.2	125.6	2.3
LeftHandPinky1-θ	-30.4	-0.2	-0.6	17.2	47.6	1.0
LeftHandPinky1-ψ	0.2	0.4	4.8	27.6	27.4	50.7
LeftHandPinky2-ϕ	-107.8	-3.2	-32.9	-0.4	107.4	37.5
LeftHandPinky3-ϕ	-76.4	-1.6	-18.0	-0.4	76.0	36.0
RightShoulder-ϕ	-55.4	-9.8	-13.0	18.8	74.2	0.6
RightShoulder-θ	-31.2	12.6	6.1	42.2	73.4	0.5
RightShoulder-ψ	-28.4	2.4	6.9	47.4	75.8	0.6
RightArm-ϕ	-78.8	21.2	28.2	103.0	181.8	0.5
RightArm-θ	-59.8	-0.8	13.2	72.0	131.8	0.5
RightArm-ψ	-43.6	13.4	25.6	113.0	156.6	0.5
RightForeArm-ϕ	-158.6	-13.8	-20.7	166.2	324.8	0.6
RightForeArm-θ	-83.0	-1.2	-16.0	69.8	152.8	0.5
RightForeArm-ψ	-132.8	16.4	35.4	164.0	296.8	0.6
RightHand-ϕ	-170.8	-2.8	5.2	169.6	340.4	0.5
RightHand-θ	-86.6	-41.4	-44.4	45.4	132.0	0.5
RightHand-ψ	-166.6	-5.4	-4.3	167.0	333.6	0.6
RightHandThumb1-ϕ	-82.6	-1.4	-0.6	89.4	172.0	0.8
RightHandThumb1-θ	-44.2	0.2	0.1	34.6	78.8	0.7
RightHandThumb1-ψ	-42.2	-1.4	-0.1	69.2	111.4	0.7
RightHandThumb2-ψ	-14.2	-1.6	-4.0	-0.4	13.8	42.1
RightHandThumb3-ψ	-28.0	-2.4	-7.8	-0.4	27.6	40.1
RightHandIndex1-ϕ	-43.4	10.6	8.8	81.6	125.0	3.3
RightHandIndex1-θ	-23.6	0.2	1.6	38.8	62.4	4.1
RightHandIndex1-ψ	0.2	0.2	5.2	35.0	34.8	77.6
RightHandIndex2-ϕ	0.2	15.8	32.1	107.0	106.8	35.1
RightHandIndex3-ϕ	0.2	10.6	17.4	75.4	75.2	33.5
RightHandMiddle1-ϕ	-44.2	14.0	11.2	79.2	123.4	1.4
RightHandMiddle1-θ	-15.4	-0.4	0.2	26.8	42.2	19.2
RightHandMiddle1-ψ	-12.6	-0.4	-0.7	22.6	35.2	1.3
RightHandMiddle2-ϕ	0.2	0.8	32.0	107.2	107.0	19.8
RightHandMiddle3-ϕ	0.2	0.4	17.2	74.2	74.0	18.6
RightHandRing1-ϕ	-44.2	17.4	13.4	80.2	124.4	1.0
RightHandRing1-θ	-26.6	-0.2	-0.7	14.8	41.4	1.3
RightHandRing1-ψ	-23.6	-0.4	-3.1	12.2	35.8	1.2
RightHandRing2-ϕ	0.2	28.0	36.4	107.2	107.0	19.8
RightHandRing3-ϕ	0.2	0.2	19.3	74.0	73.8	18.5
RightHandPinky1-ϕ	-41.8	13.8	15.5	82.6	124.4	3.2
RightHandPinky1-θ	-28.8	-0.2	-1.7	16.2	45.0	1.6
RightHandPinky1-ψ	-34.4	-0.4	-6.3	-0.4	34.0	42.7
RightHandPinky2-ϕ	0.4	29.2	41.0	108.2	107.8	26.1
RightHandPinky3-ϕ	0.2	11.4	22.1	75.8	75.6	23.2
LeftUpLeg-ϕ	-26.4	5.0	13.9	105.0	131.4	0.5
LeftUpLeg-θ	-80.6	0.6	-11.5	41.6	122.2	0.5
LeftUpLeg-ψ	-53.6	0.8	7.3	99.6	153.2	0.5
LeftLeg-ϕ	-173.8	-3.8	-12.6	167.2	341.0	0.5
LeftLeg-θ	-18.6	1.4	25.8	84.6	103.2	0.5
LeftLeg-ψ	-145.8	7.0	14.3	163.4	309.2	0.5
LeftFoot-ϕ	-55.0	-4.0	-6.6	34.8	89.8	0.5
LeftFoot-θ	-40.8	-7.0	-4.6	54.4	95.2	0.5
LeftFoot-ψ	-20.0	15.0	16.7	59.2	79.2	0.5
LeftToeBase-ϕ	-7.2	-0.2	-0.0	10.8	18.0	0.6
LeftToeBase-θ	-56.2	-0.2	-3.4	33.6	89.8	0.6
LeftToeBase-ψ	-28.8	0.2	0.2	20.2	49.0	0.7
RightUpLeg-ϕ	-144.8	-2.0	-14.8	97.0	241.8	0.5
RightUpLeg-θ	-85.0	0.6	-18.7	33.6	118.6	0.5
RightUpLeg-ψ	-140.0	-4.8	-8.1	102.8	242.8	0.5
RightLeg-ϕ	-175.8	3.0	2.0	175.0	350.8	0.5
RightLeg-θ	-11.4	8.8	33.7	86.2	97.6	0.6
RightLeg-ψ	-172.0	-0.6	0.3	170.0	342.0	0.5
RightFoot-ϕ	-37.2	-2.6	3.8	48.2	85.4	0.6
RightFoot-θ	-43.8	-11.4	-7.7	53.8	97.6	0.6
RightFoot-ψ	-64.2	-23.8	-17.4	42.2	106.4	0.6
RightToeBase-ϕ	-109.0	0.2	-0.9	26.0	135.0	1.1
RightToeBase-θ	-71.0	-0.6	-2.5	39.2	110.2	0.8
RightToeBase-ψ	-121.4	0.2	-2.3	27.8	149.2	1.3

Table 4.1: Parameter Distributions: Dataset limits and statistics per channel

- The following channels should allow for full 360° rotation, even though the true extrema are rarely occurring;
  e.g. the 'LeftForeArm-ϕ' channel has a detected range of 341.8°, but this restriction is artificial, due to the method chosen for detecting the limits.
  - { Left, Right }Arm-{ ϕ, ψ }
  - { Left, Right }ForeArm-{ ϕ, ψ }
  - { Left, Right }UpLeg-{ ϕ, ψ }
  - { Left, Right }Leg-{ ϕ, ψ }

4.3.2. Skeletal Simplification

Based on the channel limits and spike detection algorithm, as described in Appendix B, outliers in the data were detected. Throughout this process, a few key observations were made with respect to the finger joint channels:

- For the finger segments #2&3 only the 'ϕ' channel (thumb: only 'ψ' channel) are present, all other channels are '0' for all time steps.
  This is anatomically correct, as the final finger segments can only rotate around one axis.
- All finger joints were tracked unreliably.
  The mean and maximum limit-independent outliers[13] were as high as 19.25% and 77.08%, compared to only 0.06% and 0.83% for all other joint channels.

Given the fact that the finger data is not only highly unreliable, but also makes up for more than half of the total dataset, the decision was made to remove the finger joint channels from the dataset. The reduced dataset size means faster processing times, as well as reduced requirements for data storage, which are both beneficial towards reducing the overall hardware requirements for this research. This reduced the number of input parameters per pose from 156 to 66.

4.3.3. Data Filtering

The filtering and cleaning of MoCap data is an entire field of research by itself that increasingly utilizes neural networks to improve accuracy and mostly performance. One of the reasons for the need for automation in this field is that even today a lot MoCap data cleaning is still done manually [Aristidou et al., 2018a; Holden, 2018].

Overall the recorded dataset looks fairly good, based on visual inspection, with the exceptions of the data related to the fingers which has already been removed. However, the remaining outliers still need to be smoothed out, to avoid sudden jumps in the output data, which could have a detrimental effect on the final drone flight path.

To achieve this, an exponential 'spherical linear interpolation' (SLERP) filtering algorithm was created.

Dynamic Exponential Filtering

The decision was made to use an exponential smoothing function with a fixed time-based decay rate. Considering that this algorithm only requires information of the previous time step, it allows for fast real-time filtering, when required.

At each time step each channel was updated according to Equation 4.1.

$$\hat{p_t} = p_t \cdot R + \hat{p_{t-1}} \cdot (1 - R) \tag{4.1}$$

Equation 4.1: Filtering: Exponential filtering

Given that filtering is to occur in discrete time, an effort was made to define an analytical relation between the desired time-based decay rate of the data and the discrete filter-ratio parameter R. This relation is defined in Equation 4.3, where r and $t_{1/r}$ define the desired decay ratio and the time to reach it respectively, as defined in Equation 4.2.

[13] Percentage of outliers −0.5%, attributed to the forced cut-off, due to the limit detection.

$$
\begin{aligned}
p_0 &= 1 \\
p_{>0} &= 0 \\
p_{\hat{t_{1/r}}} &= \frac{1}{r} \\
t_{1/2} &: \text{The system's half-life}
\end{aligned}
\tag{4.2}
$$

Equation 4.2: Filtering: Exponential decay time-ratio

The parameter M defines the relation between the discrete R and continuous r. It was computed numerically through simulation for $r = \{2, 4, 5, 10, 20, 40, 50, 100\}$[14] and approximated by regression of the simulation results, for all other values of r, as defined in Equation 4.3.

$$
\begin{aligned}
R\left(r, t_{1/r}, \text{FPS}\right) &= \frac{\hat{R}}{\text{FPS}} \\
\hat{R}\left(r, t_{1/r}\right) &= \frac{1}{t_{1/r} \cdot M} \\
M\left(r\right) &= 0.190851 + \frac{1}{r \cdot 0.416355} \\
&-- \\
M\left(r = 2\right) &= 1.438106 \\
M\left(r = 10\right) &= 0.436192 \\
M\left(r = 100\right) &= 0.218457
\end{aligned}
\tag{4.3}
$$

Equation 4.3: Filtering: Parameterized dynamic exponential filtering parameter

This ensures comparable time-based behavior, even at re-sampling to different FPS rates or the desire to adapt the decay rate, for more or less aggressive filtering.

The application of a more advanced method, such as a Kalman filter would perhaps have yielded better results, but considering that the data cleaning aspect does not define the core of this research, the deliberate choice was made for an algorithm that was faster to implement.

SLERP

For interpolation of 3D angles, direct interpolation of each Euler-angle channel individually can lead to some undesired visual artifacts; even when accounting for discontinuities manually. The spherical linear interpolation (SLERP) algorithm interpolates smoothly between two sets of 3D rotations. It computes first computes the points on the unit sphere that both sets of angles point to and then interpolates linearly along the great circle arc between these points.

This results in the desired visual effect of angular interpolation. SLERP was utilized in conjunction with the exponential filtering algorithm, by moving a factor of R between the two computed points along the great circle arc, for each time step.

Outlier repair

To filter the motion and its derivative, exponential smoothing was applied. For the filtering of the pose and angular velocity, exponential smoothing with '$t_{1/10} = 0.25$s' was applied. This means that after 250ms only 10% of the current pose is retained in the filtered output. This filtering constant avoids the introduction of extensive lag, for when the MoCap system is performing as expected and does not record any outliers.

If only the current frame is detected as an outlier, but the previous frames are valid, the next pose will be estimated based on a first-order approximation, as defined in Equation 4.4.

[14] In Equation 4.3 on the three most common values of M are presented for $r = \{2, 10, 100\}$, but all other values were used to determine the regression coefficients.

$$\begin{aligned} \hat{v_t} &= (p_{t-1} - p_{t-2}) \\ v_t &= \hat{v_t} \cdot R + v_{t-1} \cdot (1 - R) \\ p_t &= p_{t-1} + v_t \end{aligned} \tag{4.4}$$

Equation 4.4: Filtering: Pose estimation, by first-order approximation, if p_t is invalid

If the current frame, as well as either of the two previous frames have been detected as an outlier, then a slow return to the mean would occur, using a filter with slower convergence $t_{1/10} = 10s$. This could happen e.g. when a body part is blocked from tracking for a while. The process is detailed in Equation 4.5.

$$p_t = \sigma_p \cdot R + p_{t-1} \cdot (1 - R) \tag{4.5}$$

Equation 4.5: Filtering: Pose estimation, by return to mean, if $p_t \& (p_{t-1} \| p_{t-2})$ are invalid

4.4. Data Preparation

Given the newly acquired and filtered dataset, the final steps, before passing the data into the neural network framework, is to transform it into a feature space that neural networks can better process and to split the full dataset for training and testing.

4.4.1. Data Normalization

Considering that the data is to be processed by a type of recurrent network, with 'tanh' and 'sigmoid' activation functions, the data is required to be the range of $[-1, +1]$.

The simplest form of data normalization is to scale the data to fit into the required range. This was achieved by dividing all data by the maximum value possible for each data type:

- Positions: 5 $[m]$
- Angles: 180 $[°]$

However, there are more advanced data normalization techniques available, transforming the original feature space to aid the training process.

6D-Rotations

Euler angles have two major issues: gimbal lock and discontinuities. The usage of quaternions for representing 3D rotations in computers solves these issues, at the cost of requiring constant re-normalization.

Neural networks are suboptimal at learning discontinuous relations in the data, as their underlying functions are fully continuous and differentiable. The fact that a neural network is end-to-end differentiable is a vital requirement for the application of back propagation. Based on this alone it is easy to see that the discontinuity present in Euler angles are hard for neural networks to learn. Because not all combinations of four number define a valid quaternion, using quaternions as the basis to define 3D rotations in neural networks can pose significant issues, as the networks output might result in an invalid configuration.

Given that our dataset is defined as relative angles, rather than absolute, not all channels contain discontinuities, but unfortunately a few still do. Every channel with limits of $[-180, +180]$, is discontinuous at the limits. Figure 4.10 illustrates the issue of rotational discontinuity in 2D.

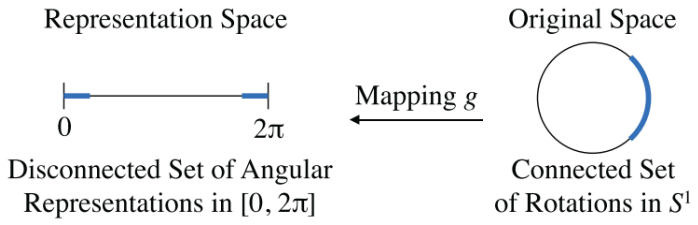


Figure 4.10: Example of a discontinuous representation of rotations in 2D
Source: [Zhou et al., 2019b]

Zhou et al. have researched the performance of neural networks given various means of formatting 3D rotations and found the mean reconstruction error, on a pose estimation task, for Euler angles and quaternions to be 6.98° and 3.32° respectively [Zhou et al., 2019b]. The researchers devised a continuous representation of 3D rotations utilizing 6 independent variables, which had a mean reconstruction error of only 0.49° and outperformed all other rotational representations on two other tasks that it was tested on.

At first, the idea of a 6-dimensional continuous representation for 3D rotations seems unintuitive, but the simplest explanation shows that it is a pre-normalized truncated rotation matrix. Rotations in 3D can also be expressed by a 3 x 3 rotation matrix, which is a set of 3 orthonormal vectors defining a coordinate system in space, with a fixed rotation to the reference frame.

Firstly, consider the fact that orthonormality means that any of the 3 vectors can be reconstructed by the other two based on the cross product. Then, it becomes apparent that any 2 of the 3 vectors contain all the information of the full rotation matrix. Thus $(2 \cdot 3 =)$ 6 values contain all required information.

Secondly, consider the fact that the matrix only needs to contain information on rotation and that scaling and sheering of space is irrelevant. This means that the requirement of vector normality is also not strictly required to hold information on the rotation alone.

Lastly, having two vectors of variable length, it is not required that both vectors are orthogonal to each other, as one can reconstruct the missing 3rd vector, based on the cross product, as long as the vector remains in the same plane.

In the end it is essentially the process of reconstructing an orthonormal coordinate system from any two vectors in space, enabling any 2 vectors to fully define a singular set of 3D rotations in space. The only exception is when both vectors are perfectly parallel. This edge case can be checked for and the limited precision of computers causes floating point vectors to almost never be truly parallel.

The set of equations that allow any 6 parameters to be converted in a valid set of 3D rotations is defined by Equation 4.6. Analogous to this, the set of equations that allow any 2 parameters to be converted in a valid set of 1D rotations is defined by Equation 4.7 and visualized in Figure 4.11.

$$
\begin{aligned}
R_{6D} &= \begin{bmatrix} a & b & c & d & e & f \end{bmatrix} \rightarrow \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix} = [\vec{x_r}\vec{y_r}] \\
\vec{x_n} &= \frac{\vec{x_r}}{|\vec{x_r}|} \\
\vec{z_n} &= \frac{\vec{x_n} \times \vec{y_r}}{|\vec{x_n} \times \vec{y_r}|} \\
\vec{y_n} &= \vec{z_n} \times \vec{x_n} \\
R_M &= [\vec{x_n}\vec{y_n}\vec{z_n}] = \begin{bmatrix} \cos\phi\cos\theta & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\sin\theta\cos\psi - \sin\phi\sin\psi \\ \sin\phi\cos\theta & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\sin\theta\cos\psi + \cos\phi\sin\psi \\ -\sin\theta & \cos\theta\sin\psi & \cos\theta\cos\psi \end{bmatrix}
\end{aligned}
\tag{4.6}
$$

Equation 4.6: N-D Rotations: Definition of 6D representation of 3D rotations

$$
\begin{aligned}
R_{2D} &= \begin{bmatrix} a & b \end{bmatrix} \rightarrow \begin{bmatrix} a \\ b \end{bmatrix} = \vec{x_r} \\
\vec{x_n} &= \frac{\vec{x_r}}{|\vec{x_r}|} = \begin{bmatrix} a_n \\ b_n \end{bmatrix} \\
\vec{y_n} &= \begin{bmatrix} -b_n \\ a_n \end{bmatrix} \\
R_M &= [\vec{x_n}\vec{y_n}] = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}
\end{aligned}
\tag{4.7}
$$

Equation 4.7: N-D Rotations: Definition of 2D representation of 1D rotations

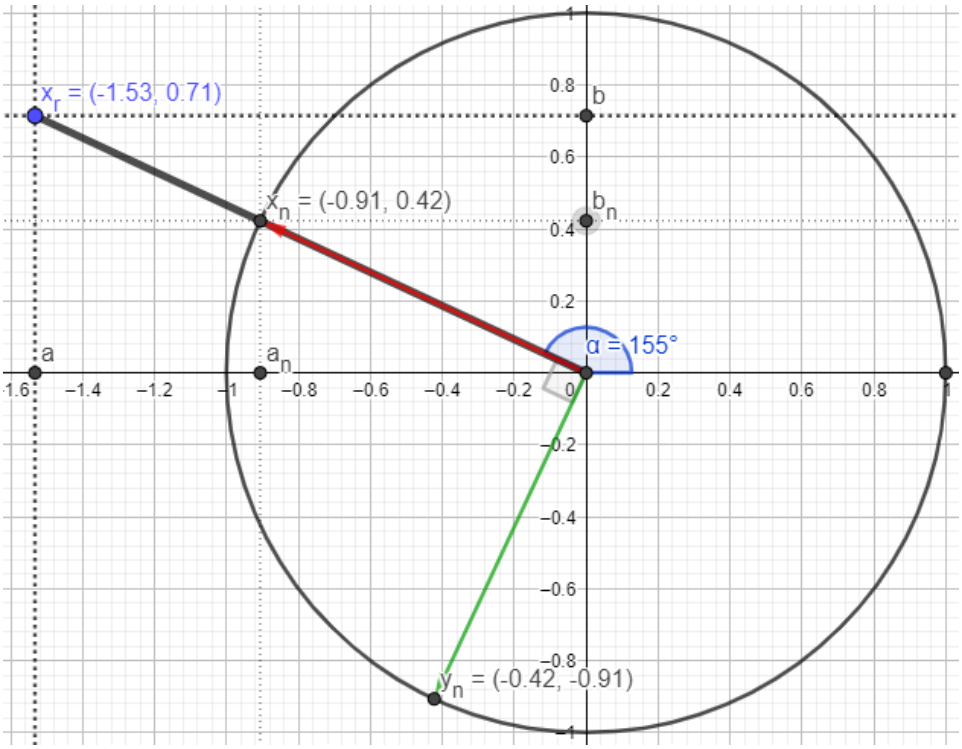


Figure 4.11: N-D Rotations: Visualization of 2D representation of 1D rotations

The benefit of this methodology is the improved final performance of the neural network, at the cost of increasing the number of input parameters to the system by a factor of 2 for every rotation.

This increased the number of input parameters per pose from 66 to 129.

Principal Component Analysis

'Principal Component Analysis' (PCA) was deployed as a secondary data normalization step. PCA is the process of extracting a new set of orthogonal basis vectors, from a n-dimensional dataset, which are ranked according to the explained variance of each component [Jolliffe, 2002]. Each extracted vector is called a principal component (PC).

An example of this is the task of sampling the height and weight of a human. A sample of any two values is most likely not going to be a realistic or valid combination. When plotting every sample of height vs weight for humans, a clear positive correlation between height and weight can be found. Identifying the PCs for this dataset in 2D is analogous to finding the major and minor axis when attempting to fit an ellipse onto the dataset. The resulting pair of principal components could then be labeled as e.g. 'size' (vector following average weight for a given height) and 'fitness' (analogous to BMI). This pair of feature vectors would describe the dataset much better than the original feature space. Figure 4.12 shows an example of PCA on 2D Gaussian scatter data, illustrating the example state above.

Application of the PCA transform matrix requires processing of all data in parallel and cannot be sequentially generated. Due to this restriction the process is limited by the RAM in the processing hardware. Therefore the transform was applied using only 50% of the original dataset.

The resulting transform was whitened by dividing the output vectors by their standard deviations resulting in a set of zero-centered feature vectors normalized to map the $[-\sigma, +\sigma]$ to $[-1, +1]$.

As the final input has to be within the range of $[-1, +1]$, but should include all original data, as an additional step the feature vectors were divided by a factor of 6 to ensure that $[-6\sigma, +6\sigma]$ were mapped, theoretically resulting in 99.9999998% of all data being included within this range.

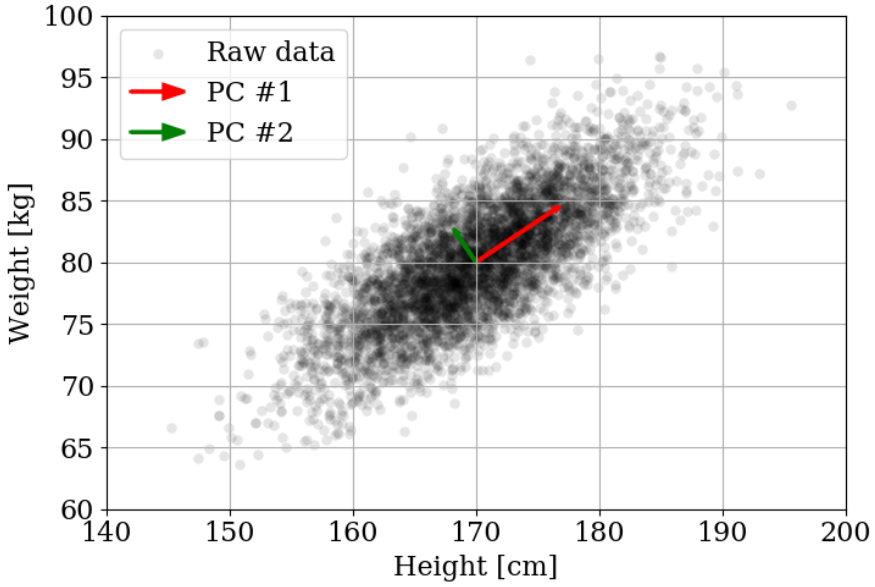


Figure 4.12: PCA: Example of PCA on Gaussian scatter data in 2D
Source: Adapted from Wikimedia Commons

The resulting transform has a multitude of benefits:

- Random-Sampling will likely be within distribution
  - Limits are harder to exceed
  - Output looks more 'natural'
- Whitening of input data
  - Uncorrelated feature parameters
  - Equal standard deviation
  - Zero centered data
- Vectors sorted by influence on final pose

Using the 6D rotations and the PCA pre-processing steps, the original data was successfully transformed into a new feature space that is much easier for the neural network to learn.

Human interpretability of the principal components is another positive side effect of the PCA transform. Humans are extremely adept at detecting and labeling the most discerning factors in all kinds of data; analogous to the PCA transform.

This means that in most cases the resulting feature vectors of the PCA transform can be attempted to be interpreted and labeled by humans. Table 4.2 presents the results of this interpretation.

As an example, Figure 4.13 visualizes PC #14, clearly showing the bending of the spine.

In the default BVH feature space the zero vector is defined to be the T-Pose, but in the transformed PCA feature space the zero vector denotes the mean pose of the dataset. The resulting mean pose can be seen in Figure 4.14.

A few observations can be made based on this mean pose:

- The pose is (almost) symmetrical
- Limbs are slightly bent
- Arms are located forward, not to the side
- Foot contact with the floor is maintained
- (Estimated) COG is above base of support created by feet
- No limits are exceeded

PC#	Human Label
1	Global position - right → left
2	Global position - left → right
3	Reaching down → up
4	Arms downwards → upwards
5	Opposing extremity correlation ('walking')
6	Arms open → close
7	Arms open → close (as well)
8	Same-side extremity correlation
9	Spine arching backwards
10	Leaning to the right
11	!No directly interpretable relation!
12	!No directly interpretable relation!
13	Reaching out with left arm
14	Spine bending right → left
15	Crossing legs (for turn)
...	*PCs are getting less and less tangible*

Table 4.2: PCA: Human Interpretation of Principal Components

Overall all observations stated here are intuitively to be expected from the human body and can be seen as verification that the resultant pose is indeed representative of the average (dancing) human pose.

Data Dimensionality Reduction is another potential benefit achievable through the PCA transform.

Figure 4.15 shows the explained standard deviation for each of the 129 principal components, as well as the cumulative total.

Based on the cumulative total, 50% of the data can be explained by just the first 27 components, 80% by the first 62 and 90% by the first 82 respectively. This means that for the last example ($129 - 82 =$) 47 components can be removed, while retaining 90% of the original information. In other words, 37% of the input can be removed with only 10% of the output being lost.

Figure 4.16 shows the favorable progression of this trade-off in PCA feature space.

This was ultimately not applied as it was not desired to remove information, before the desired end result has been achieved. However it can be a powerful tool when further optimizing the final model.

4.4.2. Dataset Splitting

Last but not least, the data had to be split for the application in the neural networks training framework.

Chunk Splitting

Initially the entire 2-6 minute takes were fed into the neural network, but the decision was made to split each take into smaller chunks. This is due to the 'backpropagation through time' (BPTT) increasing the training time substantially for each additional recurrent pass. Therefore, a trade-off had to be made between:

- Longer chunks → longer training times
- Shorter chunks → less long-term dependencies

After visual inspection of the dance data and consultation with a professional dancer, it was deduced that 10 seconds should be the minimum to include a snapshot of meaningful dance interaction, which holds enough information about multiple motions and thus long-term dependencies. Dance segments are usually expressed in 8-counts, with $7.5 - 26$ counts per chunk (10 seconds per chunk at $45 - 156$ BPM[15]) this results in $\approx 1 - 3$ dance segments per chunk.

Given that not all take durations are evenly divided in chunks of a given size the remainder was discarded. With a chunk duration of 10 seconds, this means that on average 5 seconds of data were discarded towards the end, but as the beginning and end of each take are indicated by a few seconds of holding the T-Pose, this was not deemed considered a substantial loss.

[15] Lento – slow (45–60 BPM) & Allegro – fast (120–156 BPM)

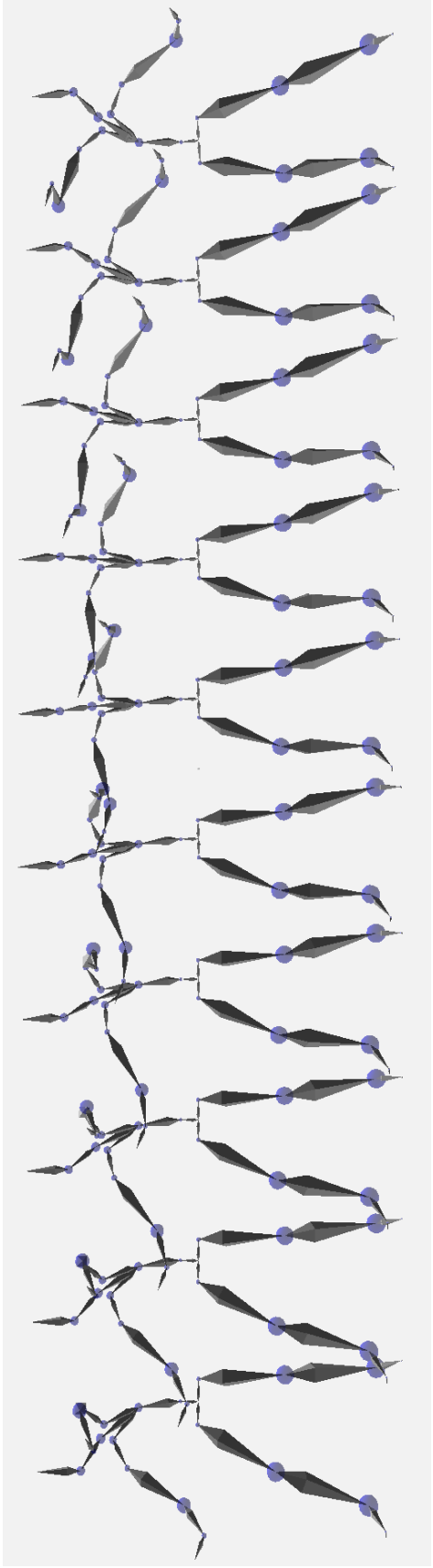


Figure 4.13: PCA: PC #14 - Spine bending right → left

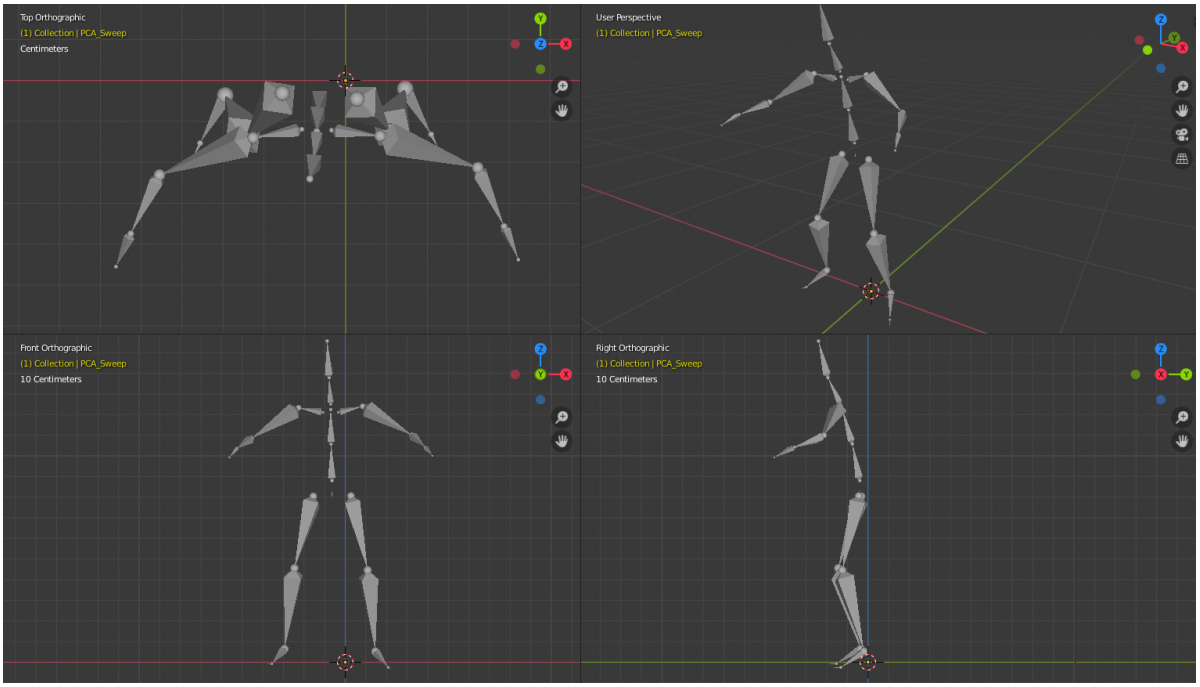


Figure 4.14: PCA: Zero/Mean Pose
Transformation: Rotated 25° along the global upward axis, to align with the principle planes for better visualization.

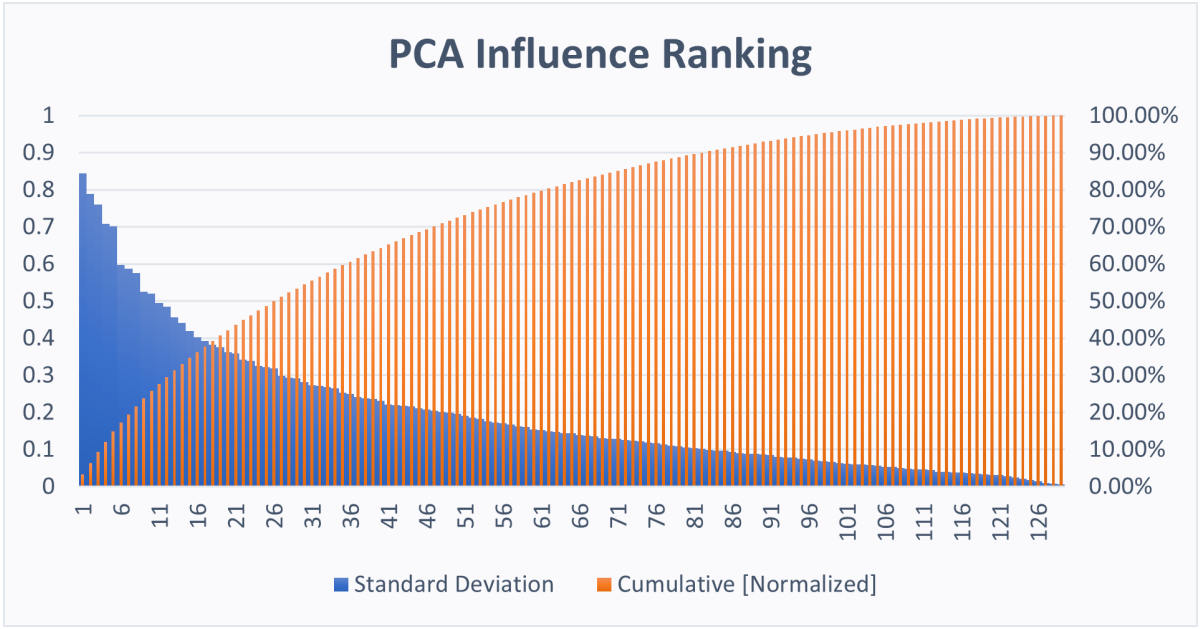


Figure 4.15: PCA: Influence Ranking

Training, Test & Validation Set

To ensure independent testing of the neural network's results, the full dataset was split three-way. A deliberate decision was made to make the split chronological, to ensure that the test and validation data is indeed truly unseen data, as at the time of recording not even the original dancers had envisioned the output.

The data was split according to the following rule set:

1. Training 80% : Used for training the neural network & computing the PCA[16]
2. Test 10% : Used for testing the neural network & determining the EarlyBreak point
3. Validation 10% : Used for testing the BHPO & validating the neural network

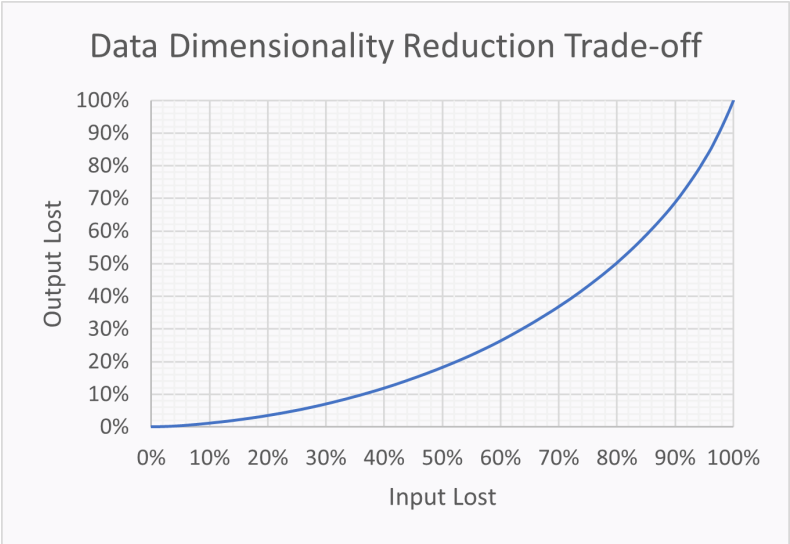


Figure 4.16: PCA: Data Dimensionality Reduction

4.5. Dataset Overview

A summary of the key parameters of our dataset are presented in Table 4.3.

Dataset Summary		
Parameter	**Value**	**Unit**
Motions		
Nr. of Dancers	2	[#]
Nr. of Takes	147	[#]
Nr. of Frames	6,647,486	[#]
Total Duration	554	[min]
Data		
Raw Data Size	12.3	[GB]
Actual Data Rate	22.7	$\left[\frac{\text{MB}}{\text{min}}\right]$
Theoretical Min. Data Rate	7.1	$\left[\frac{\text{MB}}{\text{min}}\right]$
Labels		
Total Nr. of Labels	361	[#]
Avg. Nr. of Labels	2.5	$\left[\frac{\text{Labels}}{\text{Take}}\right]$
Recording		
Frame rate	100	[FPS]
Nr. of Markers	54	[#]
Nr. of Camera	16	[#]
Avg. footage per recording day	79.1	$\left[\frac{\text{min}}{\text{day}}\right]$

Table 4.3: *AKOB* MoCap Dataset Summary

After splitting the data, the following number of 10 second chunks are to be fed into the final network:

1. Training: 2596
2. Test: 324
3. Validation: 324

4.5.1. *AMASS* MoCap Dataset Comparison

As previously mentioned, there appears to be no publicly available MoCap dataset which satisfies the requirements for this research, however this does not mean that there are no valuable insights to be gained from comparing our newly acquired dataset to these publicly available datasets.

[16]Only 62.5% of the training dataset was used for the PCA, corresponding to 50% of the total dataset

Table 4.4 compares our dataset to those MoCap datasets that were used to create the *AMASS* dataset, the largest publicly available MoCap dataset, at the time of writing[17].

Dataset	#Markers	#Subjects	#Motions	#Minutes	Avg. Take Length [s]
AKOB	*54*	*2*	*147*	*554*	*226.1*
AMASS	**N/A**	**460**	**13944**	**2710**	11.7
KIT	100	55	4232	662	9.3
CMU	42	106	2083	552	15.8
BMLrub	41	111	3061	523	10.2
Eyes Japan	37	12	750	364	29.0
BMLmovi	67	86	1801	169	5.6
MPI HDM05	41	4	215	145	40.3
BMLhandball	41	10	649	102	9.4
Total Capture	53	5	37	41	66.6
EKUT	51	4	349	31	5.2
ACCAD	82	20	252	27	6.3
PosePrior	53	3	35	21	35.6
MPI MoSh	89	19	77	17	12.8
SFU	53	7	44	15	20.7
Transitions	49	1	110	15	8.2
DFaust Synthetic	67	10	129	10	4.8
Human Eva	39	3	28	8	18.1
TCD Hands	85	1	62	8	7.7
SSM	86	3	30	2	3.7
Other noteworthy MoCap datasets					
Human3.6M	?	11	?	298[18]	?
LAFAN1	?	5	77	≈276	≈215

Table 4.4: MoCap Dataset Comparison
Source: https://amass.is.tue.mpg.de/dataset, [Ionescu et al., 2014] and [Harvey et al., 2020]

The main take-aways from these comparisons are twofold:

Firstly, in terms of duration (data size) our dataset is the second largest to date, which should give confidence that enough data was acquired to be used in further research.

Secondly, almost all other datasets are focused on short ($<$ 1 min) takes, consisting of a large variety of different motions, which are primarily used for motion classification or short-term motion prediction. In contrast, our dataset consists of long ($>$ 2 min) takes and only a singular motion type; dancing. This leads to the assumption that we have created the largest dance MoCap dataset to date, as well as the only dataset that recorded simultaneous motion of more than one subject on a large scale.

[17] Please note that the values stated here differ from those stated in their official paper, this is because the dataset is still being expanded. The data presented here is taken from https://amass.is.tue.mpg.de/dataset on 11.03.2021 (Login required).
For references to the original data and papers please consult the *AMASS* website at amass.is.tue.mpg.de, or their original paper [Mahmood et al., 2019].

[18] Not explicitly stated in original paper. Estimation based on $3,578,080$ Frames, taken by 4 Cameras at 50 Hz [Ionescu et al., 2014].

5

DanceNet-BHPO Framework

Training neural networks requires a framework to perform and monitor the training process and results. For this research the custom '*DanceNet* - BHPO' framework was developed, built on top of the *PyTorch* machine learning ecosystem, written in Python 3.7.

The framework consists of four major elements, each with their unique requirements and use within the training and testing procedure:

1. *DanceNet* : The (various) neural network architecture(s) to be trained
2. Training Framework: Used for training and monitoring the network
3. Optimization Framework: Used for 'Bayesian Hyperparameter Optimization' (BHPO)
4. Cluster Framework: Used for BHPO on multiple machines in parallel

This chapter describes the inner workings of and custom development performed on the latter three frameworks, while Chapters 6 and 7 detail the respective architectures used for the *DanceNet* neural network framework core. This means that in what follows, only the general developed framework is described. No network-specific parameters or methods are discussed.

5.1. Training Framework

A neural network by itself is essentially a collection of mathematical relations, combined with a set of variable parameters used in these expressions. But the network itself does not learn, it requires a framework to optimize all its parameters.

The framework is split into three stages:

1. Initialization: Setup the network and all required elements
2. Training: Train the network iteratively, until one of the exit criteria are met
3. Post-processing: Evaluate the final network and generate the resulting output motions

The flowchart in Figure 5.1 showcases these stages and the steps taken to train the neural network over the duration of many iterations and epochs.

In addition to the backend code that performs the actual training procedure, a matching front-end was developed to visualize the training procedure to the researcher in real-time. Screenshots of this graphical user interface (GUI) can be found in Appendix C.

The following section describes some parameters and algorithms used in the framework in detail, as they are important to understand some of the major elements of the process.

5.1.1. Reality gap

The reality gap is a performance indicator of the network, intended to represent the divergence of the network's performance on the test dataset. Numerically it is defined by the ratio of the test loss over the training loss, as shown in Equation 5.1.

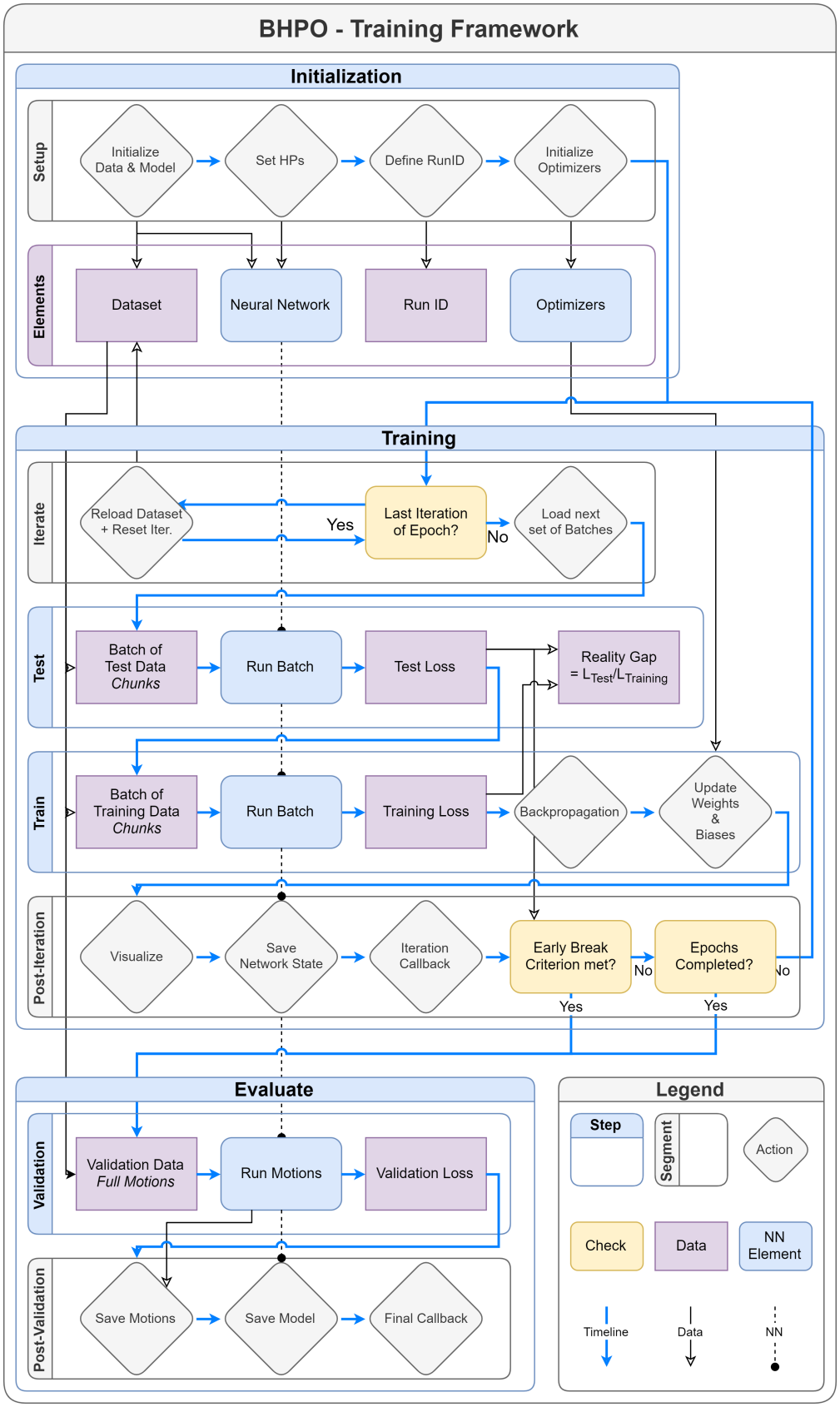


Figure 5.1: BHPO: Neural Network Training Framework

$$RG = \frac{Loss_{Test}}{Loss_{Training}} \tag{5.1}$$

Equation 5.1: Reality Gap

The reality gap metric allows for detecting overfitting. When the training loss keeps decreasing, but the test loss starts diverging and gets worse, the reality gap will start to rise above the desired state of ≈ 1.

The reality gap was close to the desired state of ≈ 1 for all performed experiments, showing a clear significance of the trained model's application on the test dataset.

5.1.2. Early Breaking

To avoid overfitting, and take the guesswork out of selecting an appropriate number of epochs for training, a custom criterion was developed to decide when to stop the training procedure prematurely.

A simplified representation of the 'Early Break' algorithm is defined in Algorithm 5.1, while a more detailed example can be found in Appendix D (see Algorithm D.1).

Algorithm 5.1 Early Break Criterion - Simplified

```
# ! DISCLAIMER: This is partial pseudo-code !

%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
% Filter Data
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

def GetExpAvg(data,p=0.9):
    ExpAvg = []
    for i,val in enumerate(data):
      ExpAvg.append(val if (i==0) else (ExpAvg[-1]*p + val*(1.0-p)))
    return ExpAvg

def Get_d(data):
    return [data[i+1]-data[i] for i in range(len(data)-1)]

def GetdExpAvg_special(data)
    return GetExpAvg(Get_d(GetExpAvg(data)))

%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
% Early Break
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Losses       = {...} # Results from Training
breakcounter = 0

def CheckEarlyBreak(EarlyBreakFactor=10**(-4),NrIterations_Min=NrEpochsPerIteration,
                                              AllowPositiveGradient=False):

  if NrIterations>NrIterations_Min:
    #--- Compute and Filter Derivatives ---
    dLosses      = Get_d(Losses)
    dLoss_ExpAvg = GetdExpAvg_special(Losses)
    #--- Get Threshold ---
    threshold = max(abs(dLosses)) * EarlyBreakFactor
    #--- Check Conditions---
    if abs(dLoss_ExpAvg)<threshold or (not AllowPositiveGradient and dLoss_ExpAvg>0):
      breakcounter+=1
      if breakcounter>=5:
        return True
      return False
    else:
      breakcounter = 0
  return False
```

In a nutshell, the algorithm determines when the change in test loss has become too small to make a significant difference to the final result, thus a local (preferably global) minimum has been reached. For common loss functions with a strict minimum it also avoids divergence by preventing positive loss gradients.

The criterion is applied to a twofold filtered derivative of the test loss. First the loss itself is exponentially filtered and the derivative of this signal is then filtered again; both exponential filters use a smoothing factor of 0.9. This ensures a suitably slow signal that only reacts to true changes in the overall course of the loss function and ignores most of the noise present in the process.

5.1.3. Hyperparameters

Each neural network's design contains certain design decisions, which are mostly expressed in the type of network to use, their arrangement and certain hyperparameters that govern the design of the network or optimization procedure. These hyperparameters are commonly selected by the researcher and are, unlike the weights and biases of the network, not optimized in the training process.

The hyperparameters that are not related to any specific architecture are defined in Table 5.1.

Hyperparameters			
Parameter	**[Default] Value**	**Limits**	**Description**
Optimizers			
Learning Rate	Variable	$[10^{-6}, 10^{-3}]$	Backpropagation step size, relative to loss gradient
Stochastic Gradient Descent (SGD)			
Momentum	0.906	$[0.1, 1.0]$	The exponential decay rate for the gradient
Adaptive Moment Estimation (Adam)			
β_1	0.9	$[0.5, 1 - 10^{-3}]$	The exponential decay rate for the first-moment estimates
β_2	0.999	$[0.9, 1 - 10^{-5}]$	The exponential decay rate for the second-moment estimates
Regularization			
Learning Rate Decay	0.9	$[10^{-2}, 1.0]$	Learning rate reduction factor, once per epoch:
Weight Decay	10^{-2}	$[10^{-10}, 10^{-1}]$	L2 penalty: pulls parameters towards 0, to aid attention
Batch Size	8	$[1, 16]$	Number of motions to be processed in parallel
Early Break			
Nr. of Epochs	25	$[2, 50]$	Maximum number of epochs to train
EB Threshold	10^{-4}	$[10^{-5}, 10^{-1}]$	EarlyBreakFactor to determine threshold (see Algorithm 5.1)

Table 5.1: Hyperparameters: Training Framework

In theory many more hyperparameters are present, as any hard-coded variable set by the researcher could affect the training results. The exponential decay rate and the minimum breakcounter of the EarlyBreak criterion, as defined in Algorithm 5.1, are such examples. The hyperparameters presented here are the most relevant for optimization. The process of optimizing these hyperparameters is described in detail in the next section.

5.2. Optimization Framework

The process of training neural networks, through means of backpropagation and gradient descent, is an advanced optimization technique, which simultaneously improves a large set of weights and biases to achieve a pre-defined goal. Once completed, this process has usually found a local (preferably global) minimum in the loss function in which the optimization process has come to a halt and does not progress any further.

The set of hyperparameters used to run this specific training process, is usually an educated guess by the researcher. Hyperparameters are by definition not optimized by the training process and are hence victim to the researcher's own biases. To take the guesswork out of designing the neural network and its training framework, a secondary optimization loop was designed to automate the process of hyperparameter optimization: The 'Bayesian Hyperparameter Optimization' (BHPO) framework. The general process of hyperparameter optimization is visualized in Figure 5.2.

To perform the hyperparameter optimization various methods can be utilized:
The simplest of which would be to perform a singular parameter sweep to determine the local optimum along the 1-dimensional slice of the feature space, for each hyperparameter. The primary downsides of this method are twofold: Firstly, optimizing along a certain axis requires sampling at (regular) intervals along this axis and hence requires a large number of training runs to find the local optimum. Secondly, given a large number of hyperparameters the optimization space is multi-dimensional and optimizing a singular parameter at a time might not necessarily lead to the global optimum.

Additional complexity is given by the fact that not all hyperparameters are continuous, as some are only able to be represented by integers (e.g. the number of layers), or an even more limited representation such as

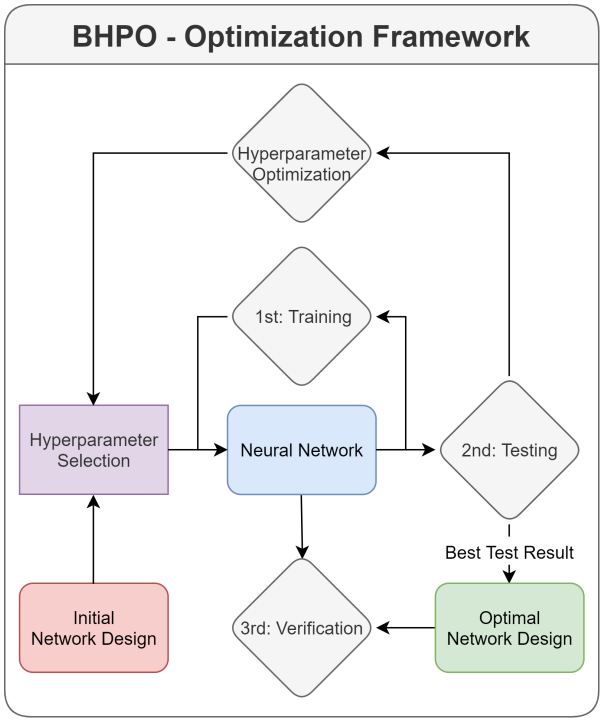


Figure 5.2: BHPO: Hyperparameter Optimization Outline

powers of two (2^X - e.g. batch size, on some machines). This means that the resulting loss landscape is not a continuous manifold, but can contain sharp discontinuities.

Given these issues, a more advanced optimization technique is desired to achieve the goal of hyperparameter optimization.

5.2.1. Bayesian Optimization

As the name already suggests, the BHPO utilizes a Bayesian optimization algorithm. The approach is Bayesian as it utilizes the previous results as a-priori knowledge and keeps on improving its estimation for the mean (μ) and variance (σ^2) of the loss landscape after every new result.

This algorithm is particularly useful for optimizing hyperparameters for neural networks, as it allows for simultaneous optimization of all parameters, can be applied iteratively and performs even for sparse results. The algorithm performs the following steps to iteratively optimize the underlying process:

Algorithm 5.2 Bayesian Hyperparameter Optimization

0. **Initialization:** Initial definition of hyperparameters by researcher OR random guess
1. **Training:** Evaluate process and compute loss
2. **Save Results:** Store hyperparameter set and associated loss as a-priori data-point
3. *If not enough data-points are known yet to estimate loss landscape: Jump to Step 0.*
4. **Update Estimation:** Generate (1st iter.) or update Gaussian process to estimate μ and σ^2 of the loss landscape
5. **Find max. Utility:** Evaluate utility function of estimated loss landscape, to find feature vector at max. utility
6. **Suggest new HPs:** Define new set of hyperparameters equal to the feature vector at maximum utility
7. *Jump to Step 1.*

Utility Function

Based on the estimated loss landscape the predefined utility function determines the utility for each point in the N-dimensional feature space. Various utility functions can be defined, the simplest 'Upper Confidence Bound' (UCB) utility function is defined by Equation 5.2.

$$\text{Utility}\,(P) = \mu_{\text{est}}\,(P) + \kappa \cdot \sigma_{\text{est}}\,(P) \tag{5.2}$$

Equation 5.2: Utility Function: Upper Confidence Bound

The process is hard to visualize in the higher N-dimensional feature space that the algorithm is capable of solving, so a simplified 1-dimensional example is provided in Figure 5.3. The upper plot shows the underlying function, as well as the best estimate and the associated confidence interval, while the lower plot shows the utility for each data-point.

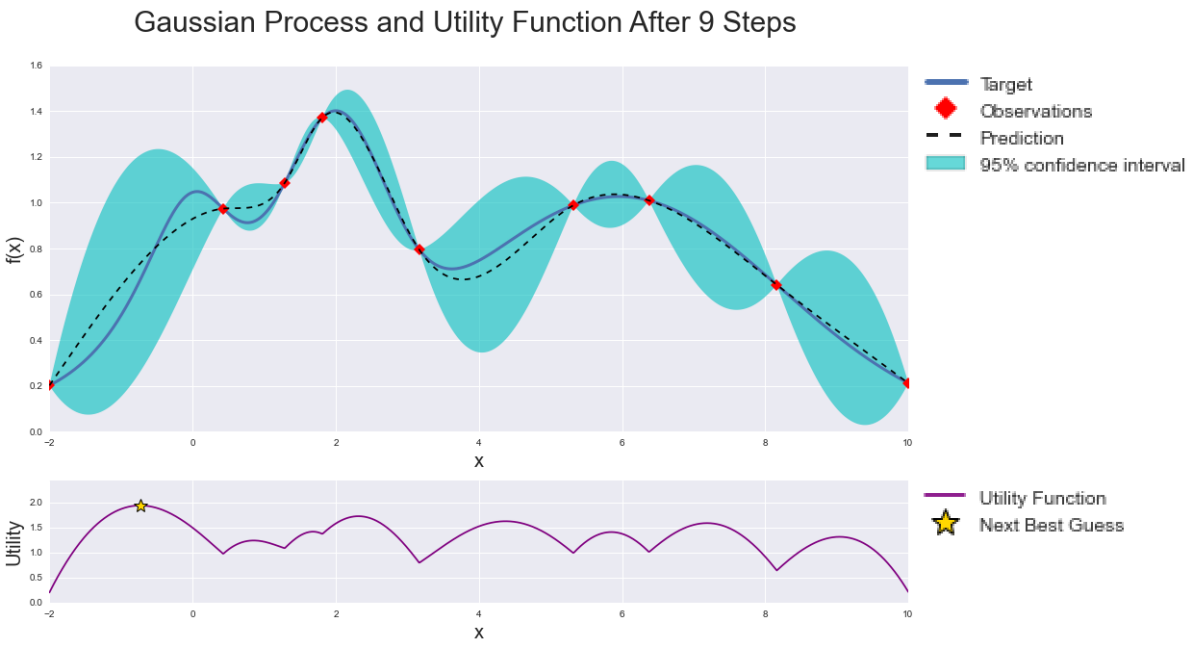


Figure 5.3: Bayesian Optimization: Utility function example
Source: https://github.com/fmfn/BayesianOptimization

Exploration vs Exploitation

The utility function defines a trade-off between the following two search methods:

- Exploration : Checking previously unknown areas of the feature space
- Exploitation : Attempting to further improve the current optimum

Each approach has benefits and downsides, presented in Table 5.2, hence both are important and a proper trade-off is required.

	Exploration	Exploitation
Pro	'Curiosity' : Ability to find 'hidden' global optimum	'Determination' : Higher chance of finding the exact local optimum within a region
Con	'Shooting blanks' : Wasting computational resources on fruitless guesses	'Marginal improvement' : Improving local optimum, but missing global optimum

Table 5.2: Exploration vs Exploitation

Essentially, each time a new point of maximum utility is evaluated, the system makes a bet that the new point will be better than the previous optimum. The value of κ determines how certain the system wants to be of that bet.

Assuming that the underlying distribution cannot always be modeled by means of Gaussian distributions with 100% accuracy, one can make use of the more general Chebyshev's inequality to determine a probability

of finding a new optimum at the point of maximum utility for each value of κ. Considering that in order to reach a new optimum the confidence bound has to be exceeded on one side only, this means that one needs to consider the one-sided Chebyshev's inequality instead. Note the similarity of the one-sided Chebyshev's inequality to the Bayesian UCB utility criterion, as defined in Equation 5.3 [Marshal and Olkin, 1991].

$$
\begin{aligned}
&\text{Chebyshev's inequality:} \\
&\sigma \neq 0 | k > 0 \;\; \& \;\; k \in \mathbb{R} \\
&\Pr\left(k\sigma_x \leq |X - \mu_x|\right) \quad \leq \quad \frac{1}{k^2} \quad \text{[Two Sided]} \\
&\Pr\left(k\sigma_x \leq X - \mu_x\right) \quad \leq \quad \frac{1}{1+k^2} \quad \text{[One Sided]} \\
&\text{Utility Criterion:} \\
&\mu_{\text{est}}(P) + \kappa \cdot \sigma_{\text{est}}(P) \quad > \quad \mu_{\text{max}} \\
&\kappa \cdot \sigma_{\text{est}}(P) \quad > \quad \mu_{\text{max}} - \mu_{\text{est}}(P)
\end{aligned}
\tag{5.3}
$$

Equation 5.3: Chebyshev's inequality & 'Upper Confidence Bound' Utility Criterion

Low values for κ favor exploitation (low risk, low reward) and high values favor exploration (high risk, high reward). In the following research $\kappa = 2.5$ was used. Per evaluation, this results in accepting any guess that provides a minimum of 13.8% (maximum[1]) success-rate, for finding a new optimum.

5

Initialization

The problem with initializing the Bayesian approach is twofold:
Firstly, with no or too few data-points available a Gaussian regression is impossible. Secondly, early in the optimization procedure very little of the full search space has been explored. This causes the system to first guess randomly and then the exploratory aspect of the optimizer visits the various extreme points of the search space. This is because, these points have the highest uncertainty, but are also usually suboptimal.

To resolve this dilemma a hybrid approach between the Bayesian optimization and a conventional grid search was used. For each hyperparameter a very sparse grid search is performed, evaluating 5 points along the axis of each hyperparameter, starting from the initial guess. These 5 points being: Both extreme points, the mid point and two points in between. This allowing for an initial regression of the loss landscape along this hyperparameter.

The downside of this approach is that it takes $N \cdot 5$ full evaluations before the Bayesian algorithm starts coming into effect. If these initial evaluations take too long, it can be opted to instead initialize the Bayesian algorithm with a handful of hyperparameter sets based on an educated guess.

5.2.2. Scope Creep & Re-parameterization

When utilizing a secondary outer hyperparameter optimization routine, such as Bayesian optimization, choosing the right number of hyperparameters is key. Choosing too few and one might miss out on significant improvements, but choosing too many and the *curse of dimensionality* might cause the algorithm to run forever.

This is where scope creep comes into play. By further improving and expanding one's network design it is natural to introduce new network elements (layers, modules, connections, optimizers). However, adding new elements usually comes at the cost of more hyperparameters. The researcher needs to make a trade-off to include the new hyperparameter into the optimization routine, or leave it fixed at a value based on an educated guess.

One solution that can be applied to resolve this issue, is to re-parameterize a subset of hyperparameters, by means of a smaller set of parameters. To illustrate this point, consider a simple feed-forward network with N layers: Given a brute-force approach one could define the number of neurons in each layer as its own hyperparameter, ending up with N discrete parameters. Alternatively, one could define a polynomial function P of power M and define the number of neurons at each layer as '$P\,(Layer\#)$'. The trade-off table between the two approaches is presented in Table 5.3.

To illustrate the benefit of the correlations between HPs, one can imagine the feed-forward network from before and assume the sub-optimal case of '#Inputs $\gg$ $\#\text{Neurons}_{\text{Layer\#1}} \ll \#\text{Neurons}_{\text{Layer\#2}}$'. In this case the first layer is significantly smaller than the number of inputs. This means that the immediate compression will destroy most information contained in the inputs, which will then be over-expanded in the following layer. A

[1] Note the '$\leq$' in Equation 5.3

	Brute-force	Re-parameterization
Pro	'Completeness' : • All parameters are present and global optimum can be found	'Simplification' & 'Correlation' & 'Continuity' : • Less HPs are faster to optimize • Potential correlation between HP can be expressed • New HPs can be turned into continuous factors
Con	'Verbosity' & 'Discretization' : • *Curse of dimensionality* causes extremely large runtime • Hyperparameters may be discrete integers	'Restriction' : • Possible range of solutions may be too restricted to find global optimum

Table 5.3: Hyperparameter Re-parametrization Trade-off

better approach would be to re-parameterize the number of neurons per layers, by defining it relative to the number of input parameters and limit the scaling factor: $\#\text{Inputs} \cdot \text{HP}_{\text{LayerSizeFactor\#N}} = \#\text{Neurons}_{\text{Layer\#N}}$.

Furthermore, parameters with a large power range, e.g. the learning rate, can best be re-parameterized by optimizing for $\log(\text{HP})$ instead, as it linearizes the search space.

5.2.3. Hyper-Hyperparameters

Just as hyperparameters are the parameters that define the training process of the model parameters, the parameters that define the hyperparameters' optimization routine are called hyper-hyperparameters.

In the case of Bayesian optimization the governing hyper-hyperparameters are: κ, which defines the trade-off between exploration and exploitation, and the extend of the prior grid search, which was set to 5 elements per hyperparameter. For other hyperparameter optimization routines more parameters might govern the approach.

Hyper-hyperparameters cause a meta optimization problem that arises from the use of hyperparameter optimization. Optimizing these hyper-hyperparameters does not change the underlying distribution of the original problem, it only changes how the search space will be explored. So in the end it is but the path to the answer that is optimized, while the answer itself remains unaffected.

While it was decided that automated optimization of these hyper-hyperparameters is outside of the scope of this research, it is important to note that there is not necessarily an end to this process and one might very well end up with an endless chain of attempting to optimize the optimizers one level below. Therefore it is still at the researcher's discretion to evaluate the level and scope of automated optimization procedures.

5.3. Cluster Framework

One of the main problems of training neural networks, and 'Recurrent Neural Networks' (RNNs) in particular, is that they take a long time to train. Most models developed for this research took between a few days, up to a few weeks to train till one of the break criteria was met.

Because of this long time horizon, running the BHPO in sequence was deemed impractical and a solution had to be found to speed up the process. While for the training procedure each subsequent step is dependent on the previous iterations, for the hyperparameter optimization each evaluation of a hyperparameter set is independent. This allows for running multiple training runs with different sets of HPs in parallel.

To utilize the ability to parallelize the BHPO process, a third meta framework was developed: The 'Cluster Framework'. The cluster framework allows for running multiple *DanceNet* training runs simultaneously, on multiple hosts, virtual machines (VMs) and GPUs. While, each node still communicates their results to a central server.

A flowchart detailing the procedure is presented in Figure 5.4.

In a nutshell, the framework is split in three major elements, with the following primary functions:

- Database: Store evaluation results and facilitate the communication between server and clients
- ClusterServer: Runs the BHPO instance and generates suggestions for new sets of HPs
- ClusterClient: Runs the *DanceNet* instance and evaluates HP sets, by means of training the neural network

The system was tested using the following configuration, showcasing successful deployment on various operating systems (OSs) and hardware components:

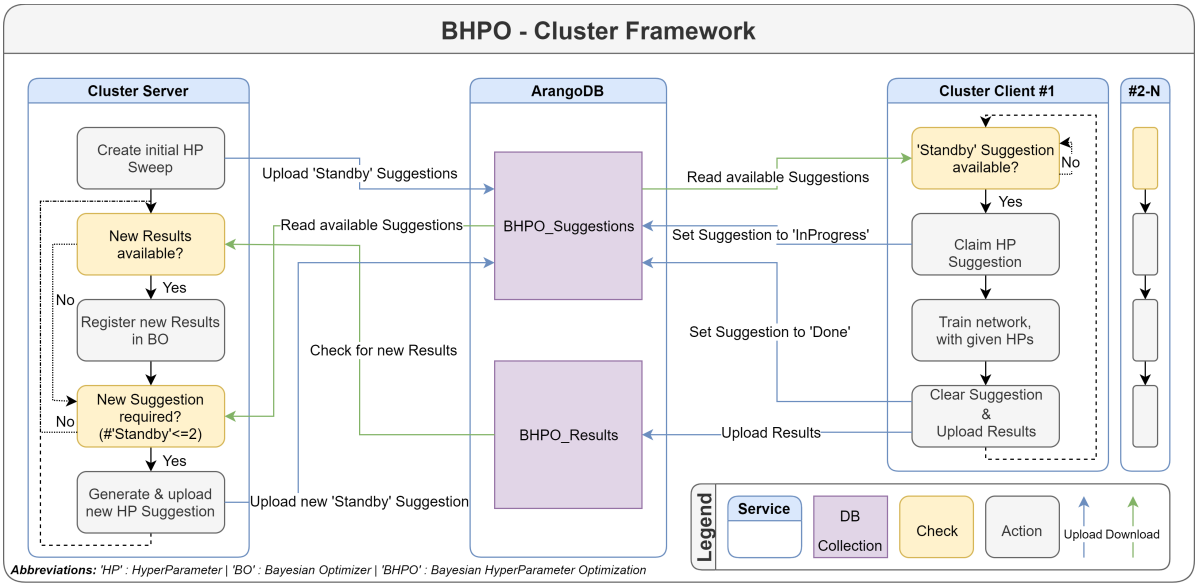


Figure 5.4: BHPO: Cluster Computing Framework

- Host #1:
  - OS : *Windows 10* [Bare metal]
  - LAN: *AKOB* Studio
  - GPUs:
    1. *NVIDIA TITAN X*
    2. *NVIDIA GTX 980*
  - Services:
    1. *ArangoDB*
    2. ClusterServer
    3. ClusterClient #1: on *TITAN X*
    4. ClusterClient #2: on *TITAN X*
    5. ClusterClient #3: on *GTX 980*
    6. ClusterClient #4: on *GTX 980*
- Host #2:
  - OS : Windows 10 [Bare metal]
  - LAN: *AKOB* Studio
  - GPUs:
    1. *NVIDIA GTX 1080 Ti*
  - Services:
    1. *ArangoDB*
    2. ClusterClient #5: on *GTX 1080 Ti*
    3. ClusterClient #6: on *GTX 1080 Ti*
- Host #3:
  - OS : Ubuntu Mate 18.04 [ESXi VM]
  - LAN: Researcher's Home
  - GPUs:
    1. *NVIDIA GTX 1060 [6GB]*
    2. *NVIDIA GTX 660 Ti* (too old to run with *PyTorch*> 0.3.0)
    3. *NVIDIA GTX 660 Ti* (too old to run with *PyTorch*> 0.3.0)
  - Services:
    1. *ArangoDB [Master]*
    2. ClusterClient #7: on *GTX 1060*
    3. ClusterClient #8: on *GTX 1060*

In theory the framework is capable of utilizing loosely networked machines in different sub-nets via the WAN (wide area network); in practice however the maximum upload-rate of the researcher's Internet connection has severely limited the framework's ability to serve the data contained in the central *ArangoDB* to all clients. To solve this issue the database was cloned onto each client's host, prior to training.

Another problem which occurred in practice was the occasional output of NaNs into the *PyTorch* tensors. The occurrence of this problem appears to be proportional to the number of ClusterClient instances utilizing the same GPU. It is assumed to be related to incorrectly configured access routines in *PyTorch*, when multiple processes read and write on the same hardware. To mitigate this issue, safeguards have been put in place to stop and re-evaluate the run when it has failed due to this cause. The problem has not occurred when running only a singular instance per GPU and was infrequent enough when running two instances to justify the risk.

The cluster framework has proven functional in practice. Unfortunately, the additional hardware, provided by *AKOB* [Host #1&2] to run the BHPO framework on, became unusable mid-research, after which the remainder of the runs were executed on a single machine [Host #3].

5.4. Data Output

Independently of the underlying architecture used, the framework saves the configuration and results of the training run to disk for further evaluation and post-processing.

The data generated and stored by the framework can be divided into 5 main categories (some files fall under multiple categories):

- Results: Actual results of the network - Final motions & Model Weights
- Analysis: Data required to analyze the training and optimization performance - 'CSV' Logs
- Visualization: Visual feedback to the researcher - Graphs & Plots
- Failsafe: Data required to pickup the run after a failure - Logs & Model Weights
- Reproducibility: Enable to run the exact same experiment - Hyperparameters, Settings & State

All data is saved in the data structure as presented in Figure 5.5.

Overall the saved data is intended to facilitate the research process and to allow the final results to be used within other frameworks.

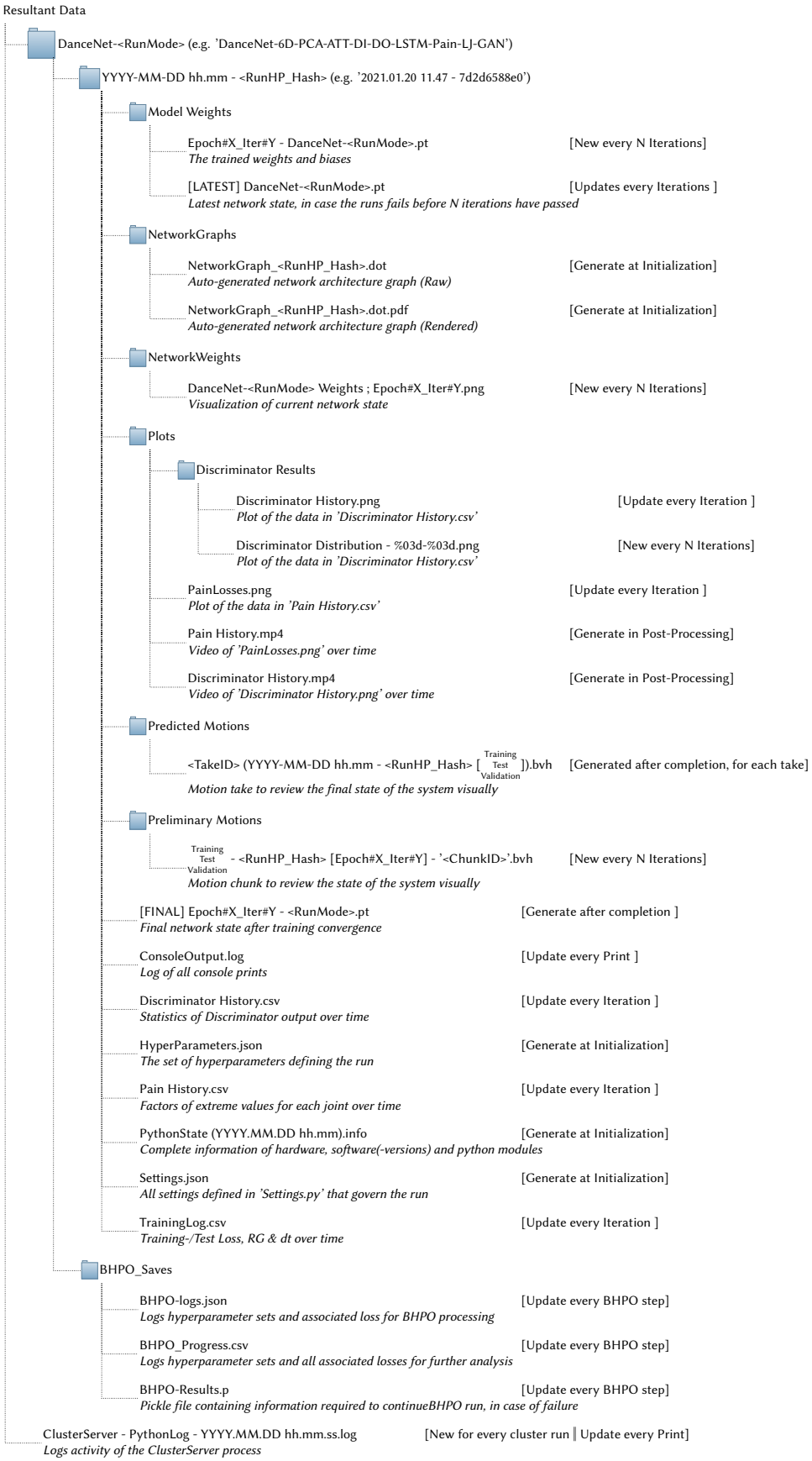


Figure 5.5: BHPO: Resultant Data Structure

6 Regression Model

The core of the entire framework is the neural network that generates novel motion data. This network was named *DanceNet* and describes all the architectural options utilized in this research. The least complex of these options is described in this chapter, detailing its benefits and shortcomings. In hindsight, the system can be categorized as a regression model.

The proposed system predicts the following dancer's next pose, based on the current pose of the input and following dancer. The hypothesis was that learning how to predict the following dancer's motion in the context of the leading dancer will allow for the generation of novel output motions, when feeding the network with novel motion input.

6.1. Design Choices

In order to test the neural network inside the developed framework, using the newly acquired data, it was important to make an informed design decision on the primary architecture of the model.

Given the context of the final result, the network is required to be executable at inference time with a high frame-rate ($>$ 100 Hz) and minimal latency. These requirements governed the trade-offs made, while choosing the appropriate network architecture.

In the preliminary report (see Appendix F) an extensive comparison of the rationale, architecture and relevance of various neural network architectures was performed. The three fundamental architectures compared were the 'Multilayer Perceptron' (MLP), the 'Recurrent Neural Network' (RNN) and the 'Convolutional Neural Network' (CNN).

For utilizing time-series data, such as motion capture (MoCap) data, RNNs appear as the obvious choice, as they are specifically designed for handling temporally sequential data. To validate this intuitive choice, other architectures are considered and compared.

6.1.1. MLP & CNN: Sliding Time-Window Trade-off

Both the MLP and the CNN are one-way systems, where each evaluation is standalone and no prior information on the input is retained. This means that, for these architectures to handle time-series data, it is required to pass a chunk of data over multiple past time steps (sliding window) into the network at once. By doing so, it is required to make a trade-off between passing a larger time window into the network or losing any information prior to the chosen window size. Given the high frame-rate and the desired result to exhibit long-term 'intuitive' interactions, allowing the system to react to input that happened only a couple of seconds in the past would mean having to store and pass hundreds of frames through the network at every time step. Even though CNNs have been proven to be better than RNNs for handling time-series data on certain tasks [Bai et al., 2018], at inference time they still suffer from the problem stated above. While the implementation of these architectures are possible, the additional memory overhead and increased complexity was deemed to be in conflict with the low latency requirement set by the final product.

6.1.2. RNN: LSTM vs GRU

The RNN has the benefit of allowing time-series data to be passed into the model frame-by-frame, without the need to explicitly memorize past inputs within the framework. This is beneficial to the latency of the system as

it can directly pass new motion frames from the MoCap system into the network. However, very deep neural networks and vanilla RNNs suffer from the 'vanishing/exploding gradient problem' [Hochreiter, 1991]. This makes the original RNN models infeasible for memorizing true long-term dependencies in practice.

This problem was tackled by Hochreiter et al., by developing a gated memory unit called the 'Long Short-Term Memory' (LSTM) cell [Hochreiter and Schmidhuber, 1997]. A singular LSTM unit is visualized in Figure 6.1.

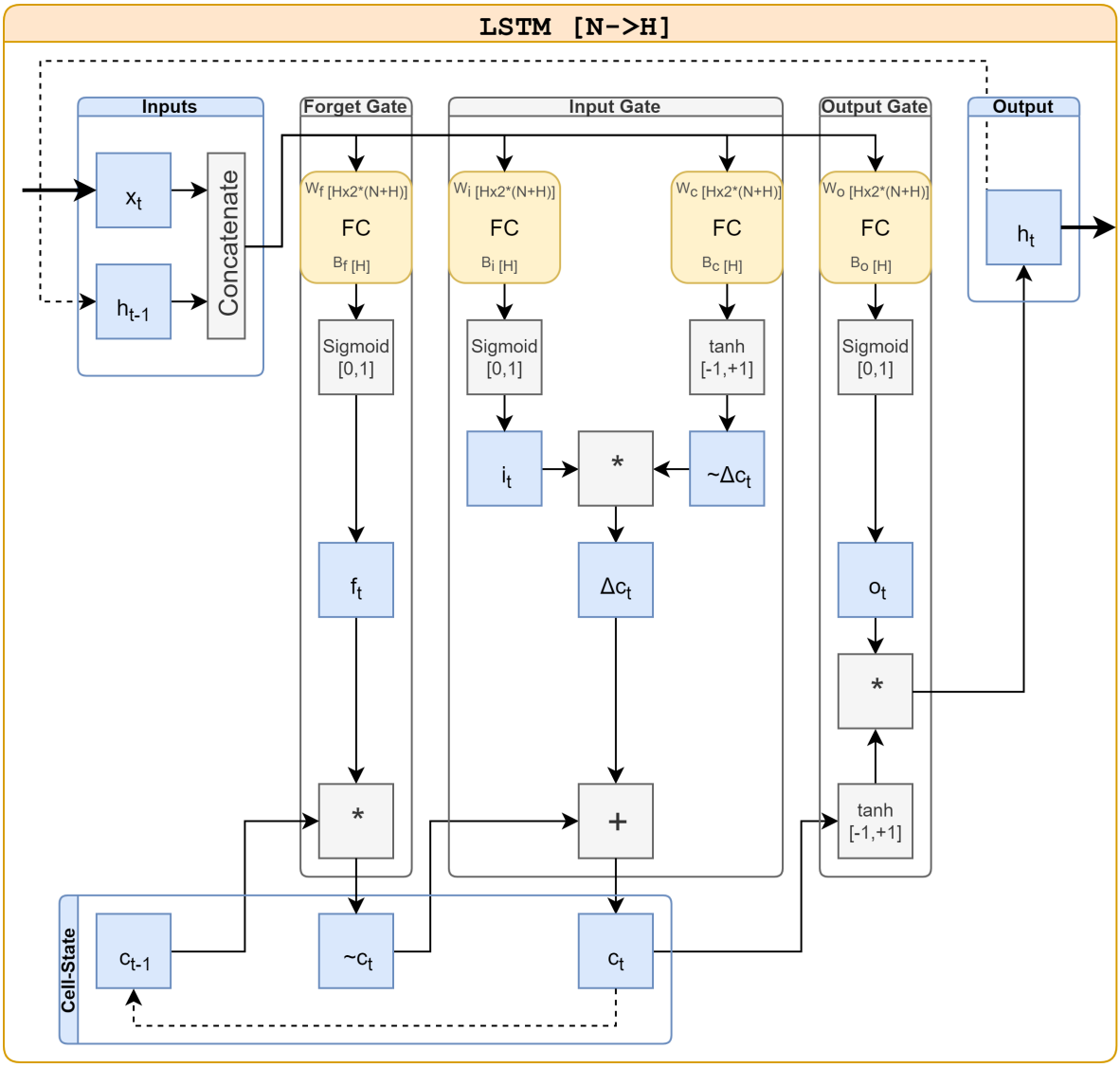


Figure 6.1: Neural network Architecture: LSTM Unit

The primary downside of training RNNs and LSTMs in particular is that the recursive nature of the underlying equations results in very large processing times for backpropagation. To alleviate this problem, Cho et al. proposed a simplified approach, retroactively called, the 'Gated Recurrent Unit' GRU in 2014, reducing the number of parameters and operations by removing the internal cell state and the output gate accordingly [Cho et al., 2014]. A singular GRU unit is visualized in Figure 6.2.

As both architectures can outperform the other, depending on the task [Bai et al., 2018], it was decided that for this research both architectures were to be used and compared.

6.2. Implementation

The regression model was the first model tested and it was intentionally kept as simple as possible. This was done to test the functionality of the entire *DanceNet-BHPO* framework and to adhere to the design philosophy

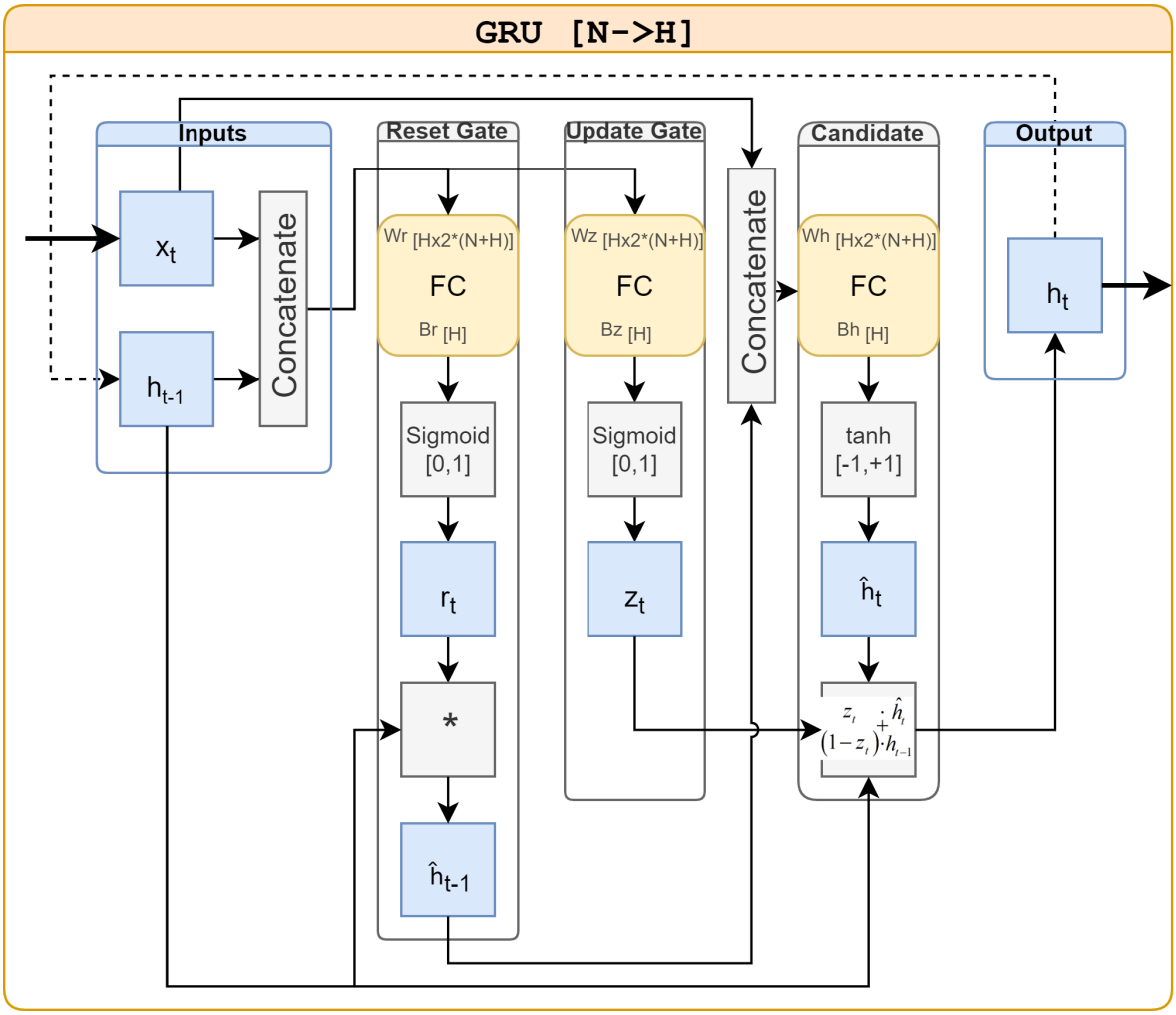


Figure 6.2: Neural network Architecture: GRU Unit

of simplicity as a virtue. Given the end goal in mind, an overcomplicated design would not be beneficial to the speed and latency of the system.

The system receives the recorded frames of the leading and following dancer as input, then it is tasked to predict the next frame of the following dancer. This is done recurrently frame-by-frame for an entire take, from start to finish The resulting architecture is visualized in Figure 6.3.

The task can be classified as a 'motion prediction' problem. The rationale behind it being that when trained this way the system would learn the following dancer's behavior in relation to the leading dancer.

6.2.1. Hyperparameters

Unlike fully connected networks and convolutional networks that can exhibit a number of hyperparameters to govern the design, whereas the design of a recurrent unit is mostly fixed and requires very little parameters to define it.

As the recurrent network is operated in a feedback loop, the same network is passed through multiple times. This results in a very deep network, when unfolded over time. Therefore, unlike their unidirectional counterparts, recurrent networks do not require a large number of layers to be classified as a 'deep learning' technique.

For a single unit architecture not even the number of neurons can be set, as the dimensionality of the in- and output of the system predefine the required underlying architecture.

It is however possible, but not required, to stack multiple units after another to create a multi-layer architecture per time step. This allows for a higher level of abstraction of the data being processed in the network.

Hence, only a single hyperparameter governs the architecture for this model, as defined in Table 6.1.

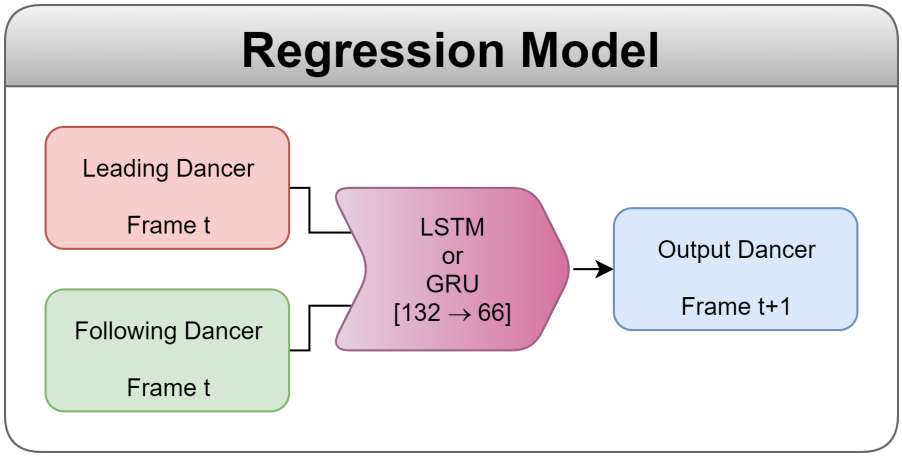


Figure 6.3: *DanceNet* Architecture: Regression Model

Hyperparameters			
Parameter	**[Default] Value**	**Limits**	**Description**
# Layers	1	[1, 3]	Number of stacked recurrent units

Table 6.1: Hyperparameters: Regression Model

Training Framework Hyperparameters used are defined as follows:

- Constant:
  - Batch size = 1 - Full takes of variable length
  - EB Threshold = 0.0001
  - # Epochs = 10 - Maximum, if not halted by EB
- Variable/Optimized by 'Bayesian Hyperparameter Optimization' (BHPO):
  - Learning Rate
  - Learning Rate Decay
  - Momentum (SGD optimizer used)
  - Weight Decay

To make effective use of the Bayesian optimization routine, the number of hyperparameters to be optimized was kept to a minimum, by setting some HPs to a constant value. This resulted in only 5 parameters to be optimized.

6.2.2. Data & Loss Function

For this first model, the only pre-processing performed on the data was basic scaling to normalize the data, as described in Section 4.4.1.

The loss function for this model was the 'Mean Squared Error' (MSE) Loss, comparing the actual following dancer's pose with the predicted one according to Equation 6.1.

$$\text{Loss}_{\mu} = \frac{\sum_{i=0}^{N} \left(p_{\text{act}_i} - p_{\text{pred}_i}\right)^2}{N} | N = 66 \tag{6.1}$$

Equation 6.1: Regression Model: Loss Function - Mean Squared Error

6.2.3. Weights & Biases

The RNN units, as shown in Figures 6.1 & 6.2), are actually meta-architectures connection a set of fully connected networks embedded in each cell. Each gate contains (at least) two fully connected (FC) networks, one processing the external input ($N = 66$) and the other processing the internal hidden state ($H = 66$).

The weights and biases[1] required for each gate are presented in Equation 6.2.

$$\begin{aligned} \underset{(N+H)\times H}{W_X} &= \begin{bmatrix} \underset{N\times H}{W_{X_N}} & \underset{H\times H}{W_{X_H}} \end{bmatrix} \\ \underset{2\times H}{B_X} &= \begin{bmatrix} \underset{1\times H}{B_{X_N}} & \underset{1\times H}{B_{X_H}} \end{bmatrix} \\ \underset{(N+H+2)\times H}{WB_X} &= \begin{bmatrix} \underset{(N+H)\times H}{W_X} & \underset{2\times H}{B_X} \end{bmatrix} \end{aligned} \tag{6.2}$$

Equation 6.2: Number of weights and biases per Gate

For each RNN unit , the computation for the number of the total trainable parameters of the LSTM and GRU based architectures are given in Equation 6.3 and 6.4 respectively.

$$\begin{aligned} &\text{Total:} \\ &\underset{4\cdot(N+H+2)\times H}{WB} = \begin{bmatrix} \underset{(N+H+2)\times H}{WB_f} & \underset{(N+H+2)\times H}{WB_i} & \underset{(N+H+2)\times H}{WB_c} & \underset{(N+H+2)\times H}{WB_o} \end{bmatrix} \\ &\text{\# Parameters:} \\ &(4\cdot(N+H+2))\cdot H = 4\cdot\left(H^2+H\cdot(N+2)\right) \\ &= 4\cdot 8844 = 35.376 \end{aligned} \tag{6.3}$$

Equation 6.3: LSTM: Number of weights and biases

$$\begin{aligned} &\text{Total:} \\ &\underset{3\cdot(N+H+2)\times H}{WB} = \begin{bmatrix} \underset{(N+H+2)\times H}{WB_r} & \underset{(N+H+2)\times H}{WB_z} & \underset{(N+H+2)\times H}{WB_h} \end{bmatrix} \\ &\text{\# Parameters:} \\ &(3\cdot(N+H+2))\cdot H = 3\cdot\left(H^2+H\cdot(N+2)\right) \\ &= 3\cdot 8844 = 26.532 \end{aligned} \tag{6.4}$$

Equation 6.4: GRU: Number of weights and biases

6.3. Results

Both the LSTM and the GRU based model were trained using the BHPO framework for numerous hyperparameter sets. The optimal losses achieved and statistics on both BHPO runs are presented in Table 6.2.

RNN Unit	# Runs	Average Run Time [h]	Total Time [days]	Avg. Reality Gap	Optimal Loss
GRU	27	6.9	7.8	0.941	0.00098
LSTM	40	8.1	13.6	0.907	0.0027

Table 6.2: Regression Model: BHPO Run Statistics & Results

The results were achieved using the (partially) optimized sets of hyperparameters, as given in Table 6.3.

RNN Unit	Learning Rate	Learning Rate Decay	Momentum	Weight Decay	# Layers
GRU	6.281144591	0.43310875	0.905515929	1.0	1
LSTM	8.237621096	0.59898849	0.885980442	0.900098016	1

Table 6.3: Regression Model: (Partially) Optimized Hyperparameters for GRU & LSTM

[1] For each gate two sets of biases are defined: 1 for the external input (B_N) and 1 for the internal hidden state (B_H). 2 sets of biases for the same function are actually redundant and could be expressed by a singular bias set: $B_N + B_H = B$. This is due to the specific *PyTorch* implementation making use of *NVIDIA*s cuDNN, which speeds up computational time.

As illustrated in Figure 6.4, the model has successfully learned to mimic the following dancer's motion.

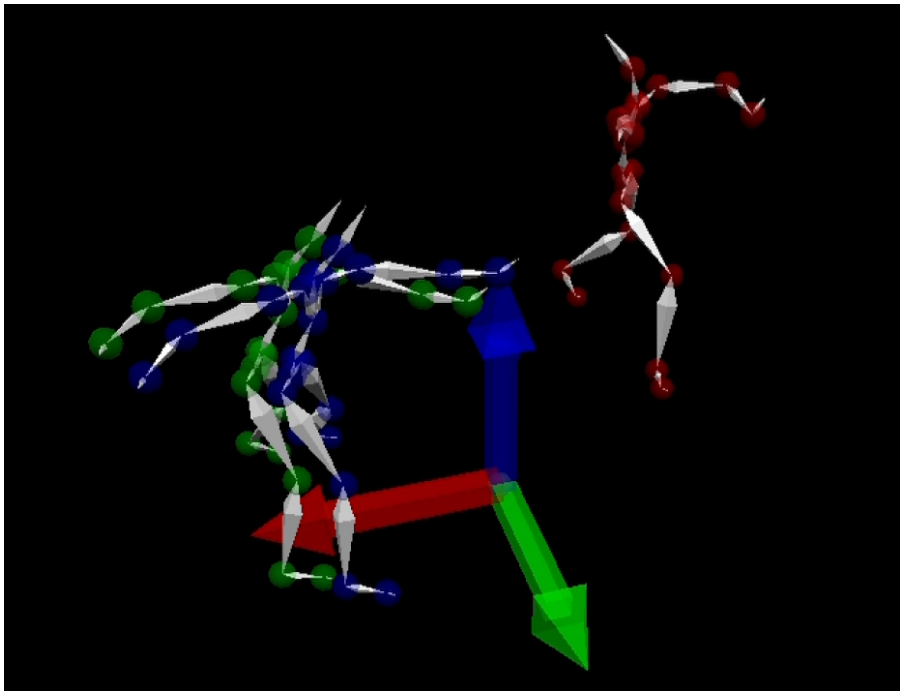


Figure 6.4: *DanceNet* Results: Regression Model
Legend: Red: Leading Dancer | Green: Following Dancer |Blue: Output Dancer

Even though the GRU based model was optimized for fewer hyperparameter sets it still produces better results than the LSTM counterpart. The observation that the GRU appears to be better at this task is also supported by reviewing the average results for all attempted sets of hyperparameters, as seen in the next section.

6.3.1. Loss Progression

Figure 6.5 and 6.6 showcase the final mean losses achieved, sorted by the test loss. The yellow lines display the average loss of all three datasets in temporal order, showcasing the stochastic nature of the Bayesian optimization procedure.

The average loss appears to follow an exponential progression, resulting in diminishing returns for extended BHPO optimization runs. This exponential optimization progression appears to be analogous to the exponential progression displayed by the actual training of the neural network.

6.3.2. Observations

Given the results presented above, a few observations stand out. These are addressed individually in the following sections.

Average Angular Error

While the model's loss cannot completely reconstruct the physical parameters underlying it, one can estimate the average error of each joint by applying Equation 6.5, reversing the process of computing the loss and normalizing the data.

$$\hat{\alpha_\mu} = \sqrt{\text{Loss}_\mu} \cdot 180° \tag{6.5}$$

Equation 6.5: Regression Model: Model Loss → Angular Error

When evaluating the minimum loss of 0.00098 this can be approximated to a mean angular error of ≈ 5.57°. This error appears large at first. However, comparing it to the sanity check for reconstructing Euler angle encoded rotations with a neural network [Zhou et al., 2019b], it actually appears better than the expected

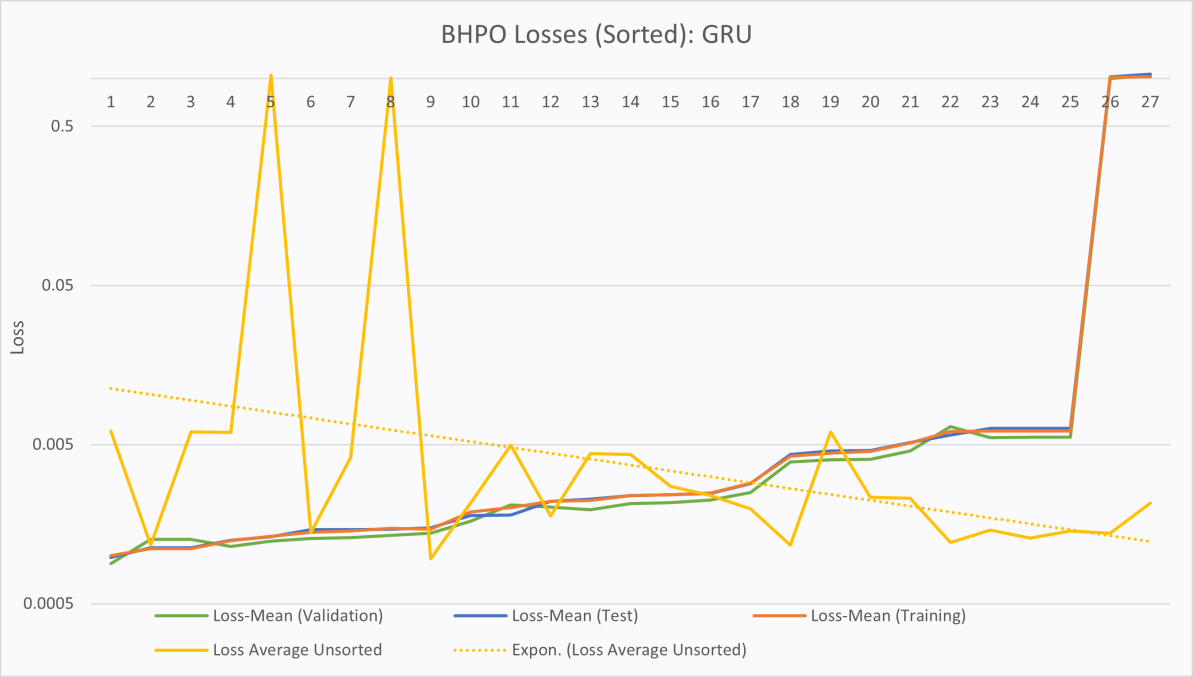


Figure 6.5: BHPO Losses (Sorted): GRU

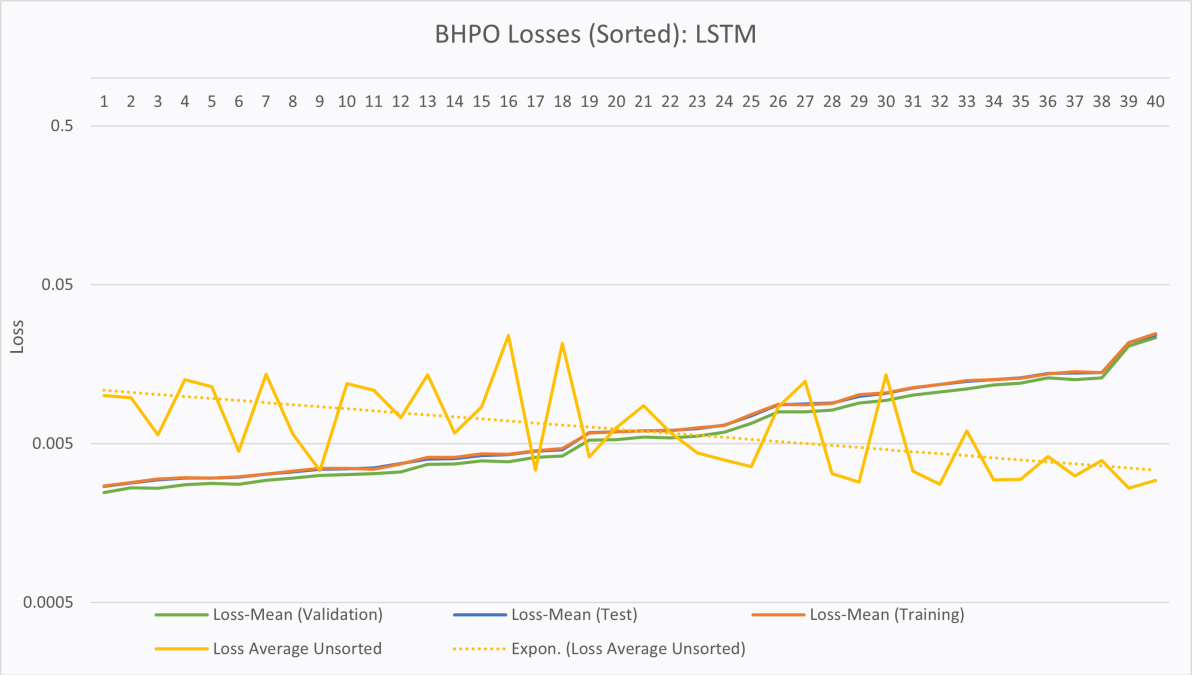


Figure 6.6: BHPO Losses (Sorted): LSTM

accuracy. Their average reconstruction error was evaluated to be '6.98°', compared to this result of ≈ 5.57°. Even though this reference experiment was conducted using an MLP auto-encoder, instead of an RNN based system, it can be seen as an acceptable benchmark for the reconstruction error of Euler angles using neural networks. One possible explanation for this slightly lower error might be the fact that the Euler angles for this research were relative angles with a defined mean, while for the general study any random Euler angle was possible.

Run Time

Even for this minimal configuration the average run time is still 7.5 hours, confirming the expectation that RNN based models take a long time to train. This extensive runtime is also the reason for the limited number of BHPO runs performed.

Expectedly, the GRU based system's training time was lower than the LSTM based system, however only by about 15% on average.

Reality Gap

The reality gap is below 1 for most of the runs, meaning that the model's results on the test dataset are actually better than the results on the training dataset.This is unexpected as usually the model is expected to perform worse on unseen data.

One hypothesis for this is that because the BHPO framework actually optimizes for the test loss that this would mean that a lower test loss would be considered better, independent of the underlying training loss. To test this hypothesis the reality gap (RG) (see Section 5.1.1) was plotted over time for both BHPO runs, as seen in Figure 6.7.

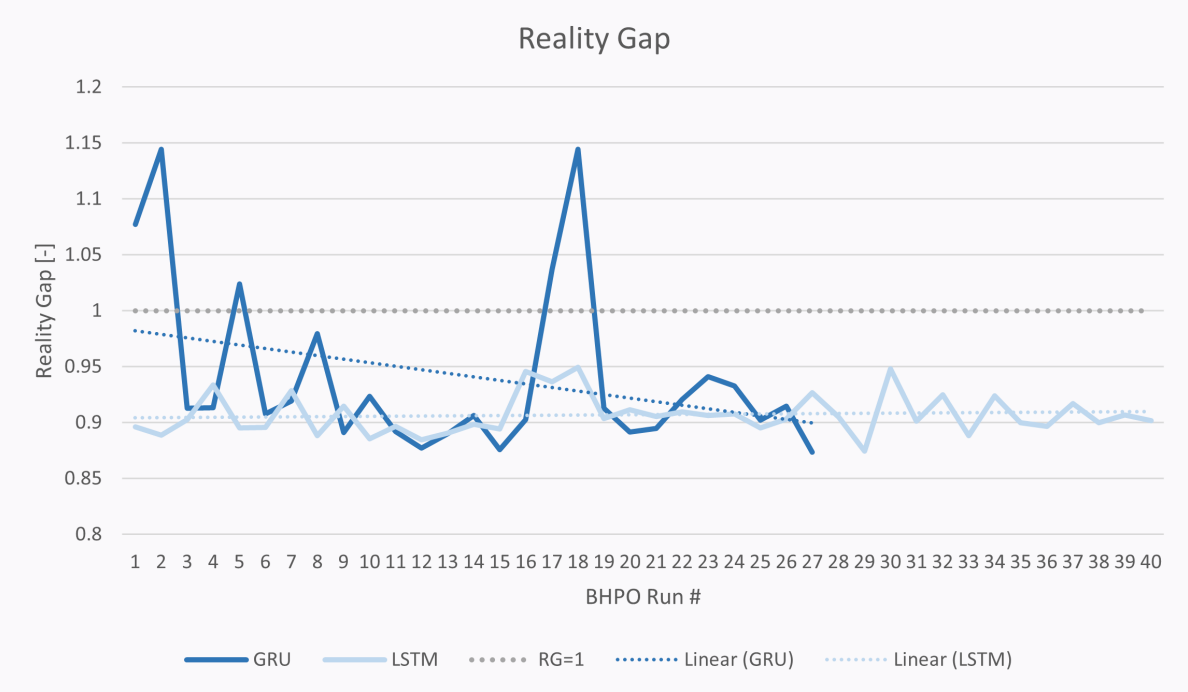


Figure 6.7: Regression Model: Reality Gap

The hypothesized behavior of decreasing RG over time can be observed for the GRU based run, however the LSTM based run displays no such behavior and shows a fairly consistent RG of ≈ 0.9 from the start.

Further research into the origins of this phenomenon will have to be conducted.

Learning Rate

The optimal learning rates for both architectures are unexpectedly high. The upper limit for the learning rate hyperparameter had to be increased from its initially set limit to accommodate for these higher optimal values.

Commonly the learning rate for training neural networks lies several orders of magnitude lower, in order to slowly approach the optimum without overshooting the target. The extremely high learning rate, as well as large momentum, may indicate a trivial underlying solution. One that can be approached with large steps, while still finding the optimum.

Outliers

The two outliers in the GRU run, with losses > 1, can be traced back to the fact that they are the only two runs where the momentum HP was set to the upper limit of 1.0. This effectively means that there is no exponential decay washing out previous gradient velocities, but that all previous gradients are retained indefinitely, which is counterproductive to the training procedure.

6.3.3. Chebyshev's Inequality

An effort was made to verify the practical relevance of Chebyshev's inequality (see Section 5.2.1) for the BHPO framework. A new run is considered to be a 'success' when a new optimum has been reached by evaluating the new hyperparameter set. Plotting the average success rate of the BHPO framework over time, as seen in Figure 6.8, it is indeed approaching the region under the maximum value predicted by Chebyshev's inequality.

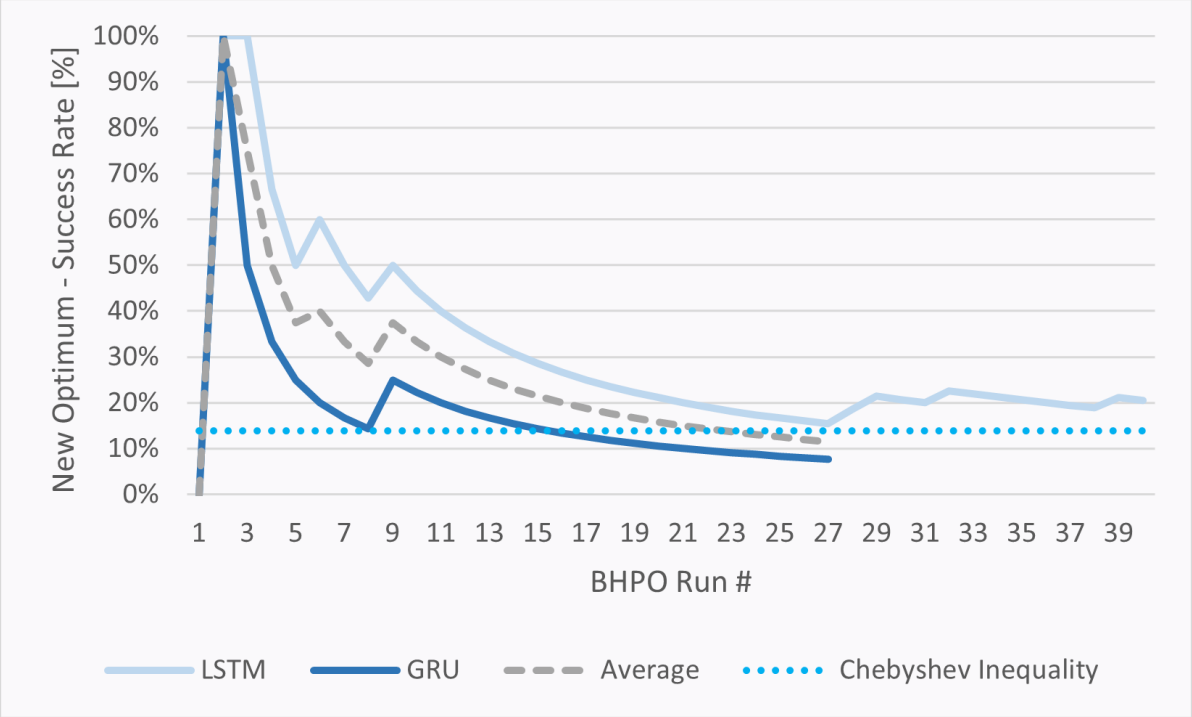


Figure 6.8: BHPO: Success-Rate vs Chebyshev's Inequality

The current sample size of these BHPO runs is not sufficient to make a conclusive statement of statistical significance. However, Chebyshev's inequality does appear to hold true as an acceptable rule of thumb for tuning the hyper-hyperparameter κ.

6.4. Feedback Dilemma

The developed model appears to successfully mimic the following dancer when feeding it data from the dataset. However, the model's setup does not hold true for using the system at inference time. This is due to the resulting feedback loop, caused by the absence of the following dancer.

6.4.1. Model with Feedback

The main thing to consider, when adapting the model for utilization in a live environment, is the absence of a prerecorded following dancer. In the live environment the leading dancer will interact purely with the output dancer generated by the network.

Therefore, the original framework, as visualized in Figure 6.3, had to be adapted for the live utilization. The modification removes the input from the following dancer and replaces it with a feedback loop of the system's output (see Figure 6.9).

It is important to note that, this new system is not intended to replace the training framework, but is purely utilized at inference time. This means that the network is trained using the original model and the pre-trained network is placed in the updated framework for live interaction.

Initialization

As the system no longer receives its input from the prerecorded following dancer, it becomes important to properly initialize the output. It was chosen to initialize the second input as a zero-vector for a few frames, representing the dancer in a T-Pose, as this is also the pose both dancers start each take in.

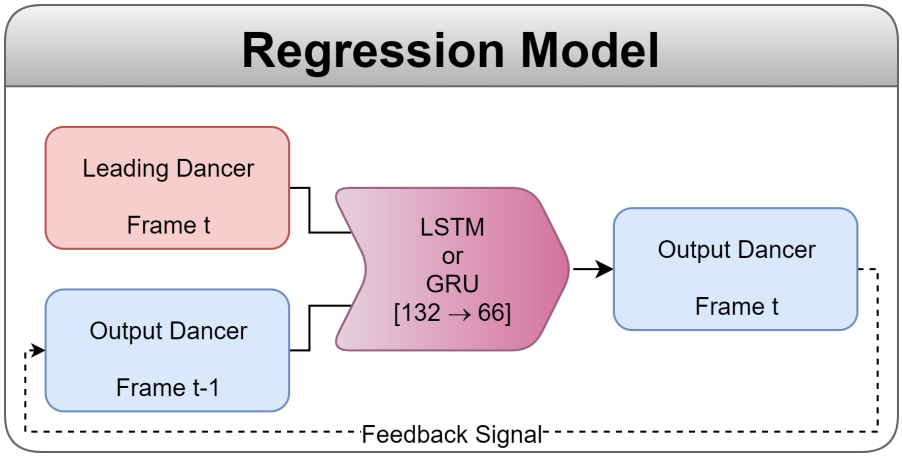


Figure 6.9: *DanceNet* Architecture: Regression Model with Feedback

6.4.2. Results

Running the adapted live system made it very clear that the current state of the system was not capable of performing as desired. The output dancer moved away from the desired position at extreme velocity only to settle a few frames later in the corner of the simulated space in an unnatural contorted pose for the remainder of the take (see Figure 6.10)[2].

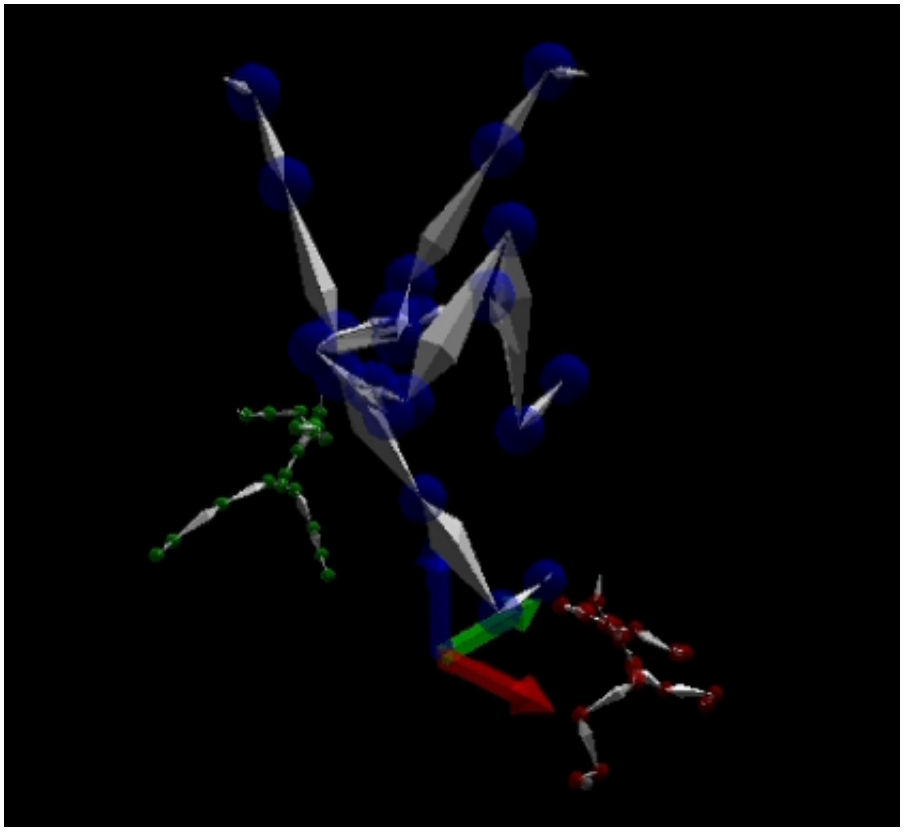


Figure 6.10: *DanceNet* Results: Regression Model with Feedback
Legend: Red: Leading Dancer | Green: Following Dancer |Blue: Output Dancer

While reviewing the data it became apparent that the resulting pose had reached the extreme values ($[-1, +1]$) for many of the feature vector's parameters. This is why the output dancer was hanging in space

at $[+5\text{m}, -5\text{m}, +5\text{m}]$ and with some twisted limbs at $\approx \pm 180°$. [3]

This indicates that the system has become highly unstable through the introduction of the feedback loop, only capped by the tanh activation function.

This also means that the output dancer just copies the following dancer's motions, without learning and internalizing the intention behind these motions.

6.4.3. Trivial Solution

Investigating the cause of the system's instability has brought to light that the question asked of the model was incorrect from the start.

The model generates acceptable results when attempting to predict the following dancer one time step into the future, but completely breaks down when faced with the feedback task. This is because the model has no incentive at all to take the leading dancer's pose into consideration nor to learn any complex relations, as the task that is asked can be simplified to a regression problem of the following dancer's inertia 10ms into the future.

When fed the ground truth values for both dancers at each time step the largest error the system can produce is an error 10ms into the future, so even a relatively large error of a few degrees would still visually look acceptable. However, when placed into the feedback loop the system's error quickly accumulates and becomes unstable.

All the neural network has to do is learn how to approximate the differentiation of the following dancer's motion, which is to say that it needs to learn the 'Backward Differentiation Formula' (BDF). Even for the 0th order case, where the system would assume $P_{t+1} = P_t$, the system's mean angular error would be capped at the mean angular offset between two recording frames. This mean ΔP_t of the dataset was computed to be $0.42°$, which is several times lower than the current system's error. This shows that the current system's performance is much more a measure of how well it is able to reconstruct the Euler angle representation of the data, rather than the actual performance of the regression task asked, exhibiting no complex interaction with the leading dancer.

6.4.4. Training with Feedback

Training the model in a feed-forward manner, to then transfer its application into a feedback system has proven problematic. Therefore, it became apparent that the feedback-loop had to be integrated into the training procedure. However, when considering the current design, two issues became apparent:

Dual Feedback Problem

Examining the feedback model closer in the context of an underlying RNN based architecture, the model actually contains a dual feedback loop. The new feedback model forces an outer feedback loop, by replacing the following dancer's input with the system output, but for the case 'NrLayers $= 1$' the internal RNN unit already feeds its own output back into the RNN unit. So in this adapted setup, the exact same system's output is fed back twice at every time step, resulting in the actual data processed by the RNN unit to be a concatenation of the following 3 poses:

- Regression Model: $[\text{Leading Dancer Pose}_t, \text{Following Dancer Pose}_t, \text{Output Pose}_{t-1}]$
- Regression Model + Feedback: $[\text{Leading Dancer Pose}_t, \text{Output Pose}_{t-1}, \text{Output Pose}_{t-1}]$

While it makes sense to feed the system both dancers' poses at each time step, such that relations between the two can be computed, it is apparent that feeding the system the same data twice is redundant. This is however no longer true when more than one layer is used, as now the 3rd pose is replaced by an unknown feature vector produced by the model's first layer.

So for training the model with feedback either one of the following solutions can be used:

- Remove the outer feedback loop
  by reducing the system's primary input to only the leading dancer's pose.
- Increase the number of layers
  to still benefit from the output pose data, but avoid data redundancy on the first layer.

[3] Some earlier results even had the output dancer completely contorted to an unrecognizable 'ball of limbs'.

Single Solution Problem

However, training the model in this manner causes another problem, which is the implication that for every input there exists only a singular correct output. Given the creative nature of the underlying data this is however not correct. The following dancer could have decided to react in a multitude of ways to the leading dancer's input, and still be considered appropriate and visually pleasing.

6.4.5. Deep Thought

> *That quite definitely is the answer. I think the problem, to be quite honest with you, is that you've never actually known what the question is.*
>
> *Deep Thought - Douglas Adams 'The Hitchhiker's Guide to the Galaxy'*

The problem is that neural networks are great at optimization and will provide you with an answer, however it might not necessarily be the one you're looking for.

Therefore, a new model had to be designed that could cope with the feedback loop built into the training procedure, as well as the notion that there can be multiple correct answers to the same input.

7

Generative Model

Following the insights gained from the initial regression model a new model had to be designed that is capable and incentivised to learn complex relations between the two dancers over an extended period of time.

To achieve this, a generative model was developed, by means of extending the basic regression model with a multitude of improvements.

7.1. Generative Adversarial Network

The 'Generative Adversarial Network' (GAN) has only recently been developed for the generation of novel images from noise [Goodfellow et al., 2014], but this meta-architecture is not limited to the application with 'Convolutional Neural Networks' (CNNs) and has been used prior for motion prediction and generation [Gui et al., 2018a; Hernandez et al., 2019; Kender and Way, 2018; Wang et al., 2020a].

The primary goal of the GAN is the generation of novel data, in contrast to more common regression, interpolation and classification tasks. This is achieved by combining two networks being trained in parallel with opposing/adversarial optimization goals. The training procedure is essentially a two-player game with both networks competing against each other, where both get better over time in an attempt to 'beat' the opponent. The basic GAN framework was adapted to fit the data and goal of this research. This section details the design of the adapted framework.

For an intuitive understanding each network can be envisioned as its own entity with a specific purpose and goal. The generator can be seen as a 'dancer' learning how to dance from scratch. The discriminator can be seen as a 'choreographer' or 'dance critic' learning how to tell amateur (fake/generated) and professional (real/recorded) dancers apart, while providing feedback to the dancer in the process.

- Generator: 'Dancer'
  - Output: Motion frame
  - Purpose: Generate new motions based on the input of the leading dancer
  - Goal: 'Fool' the discriminator
  - Learns through: Feedback given by discriminator
  - Active during: Training & Inference
- Discriminator: 'Choreographer'/'Dance Critic'
  - Output: 1 logit value (0 = Fake, 1 = Real)
  - Purpose: Determine whether a motion sequence is real/recorded or fake/generated
  - Goal: 'Call out' the generator as a 'fraud'
  - Learns through: Being presented ground truth samples
  - Active during: Training only

The discriminator attempts to learn how to tell the recorded and generated motions apart. This is equivalent to a binary classification task, with the only difference being that one of the two classes is a dynamic output from another network, as opposed to static ground truth data.

The generator attempts to generate new motions frame-by-frame. This is equivalent to the mode of operation of the previous model, with the only difference being that it is trained by means of receiving feedback from the discriminator through back propagation, as opposed to regressing ground truth data directly.

This results in an indirect training approach of the generator by means of backpropagation through the discriminator network, rather than defining an error based on direct numerical comparison of the output data with the recorded reference. Intuitively, it can be seen as a teacher learning a subject and then explaining it to the student in a logical manner; As oppose to forcing a student to learn a subject by handing him a large amount of questions with associated answers, but no explanation in between.

Building on the regression models (presented in Figure 6.3 and 6.9), Figure 7.1 shows the schematic flow diagram of the generative model and its incorporation of the following dancer's data outside of the primary motion generation loop.

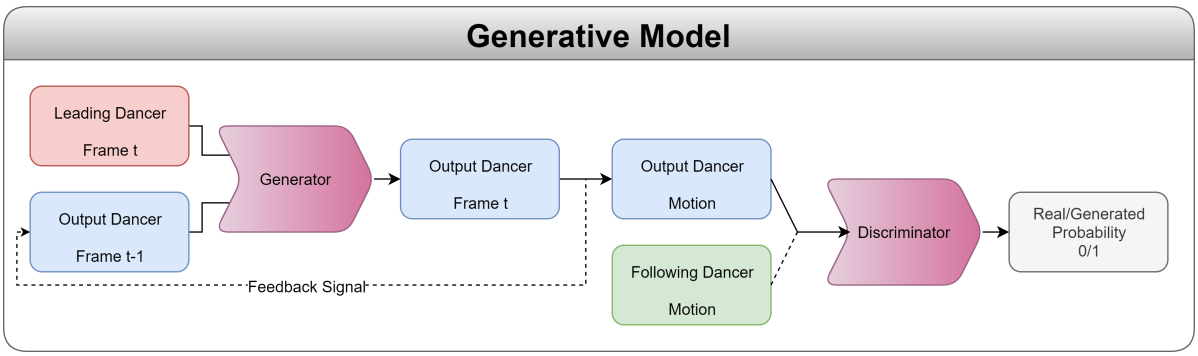


Figure 7.1: *DanceNet* Architecture: Generative Model

7.1.1. Loss Metric

Unlike the regression model where the final output was a motion frame, the final output of the GANs discriminator is a single logit value $(0, 1)$. It defines the system's guess, as well as its confidence in this guess: Fake : $(0, 0.5)$ and Real : $(0.5, 1.0)$

Therefore, the 'Binary Cross Entropy' (BCE) loss function (see Equation 7.1), commonly used for GANs, was used for this model.

$$\begin{aligned} &g_{\text{act}} \in [0,1] \quad , \quad g_{\text{pred}} \in (0,1) \\ &\text{Loss}_{\text{BCE}} = -\left(g_{\text{act}} \cdot \log(g_{\text{pred}}) + (1 - g_{\text{act}}) \cdot \log(1 - g_{\text{pred}})\right) \end{aligned} \tag{7.1}$$

Equation 7.1: Generative Model: Loss Function - Binary Cross Entropy

At first glance the BCE loss looks complex, however the equation is simplified considerably when binary labels are applied and considering the loss for the limits of the prediction, as shown in Table 7.1. Approaching the formula in this manner clearly shows its use case for defining the loss in a binary classification task.

	$g_{\text{act}} = 0$	$g_{\text{act}} = 1$
Loss =	$-\log\left(1 - g_{\text{pred}}\right)$	$-\log\left(g_{\text{pred}}\right)$
$g_{\text{pred}} \to 0$	$\to 0$ (True Negative)	$\to \infty$ (False Negative)
$g_{\text{pred}} \to 1$	$\to \infty$ (False Positive)	$\to 0$ (True Positive)

Table 7.1: BCE Loss: Simplification and Limits

7.1.2. Training Algorithm

Algorithm 7.1 illustrates the procedure of training the GAN, alternating between fake and real data. Due to the recurrent nature of the underlying networks, the entire motion is generated frame-by-frame and only afterwards passed into the discriminator frame-by-frame.

Unlike conventional systems where all of the networks parameters are updated based on the same optimizer and working towards a common goal, the GAN is build up of two separate networks, each with their own set of parameters, optimizer and goal:

Discriminator: The BCE loss is appropriate for scoring a classification task and is hence attempted to be minimized by the discriminator, which is trying to tell the data apart as good as possible. For each optimization step the discriminator's optimizer brings only the discriminator's parameters a step closer to minimizing the final loss.

Generator: The generator's goal is to fool the discriminator into wrongly classifying the fake labels and is hence trying to increase the final loss function of the discriminator. For each optimization step the generator's optimizer brings only the generator's parameters a step closer to maximizing the final loss.

This is why the operational mode of a GAN is also referred to as a 'mini-max game'.

Note that, for the training on real data the generator is not used, as the data to be passed into the discriminator is already given. Similarly, the discriminator is discarded once the training procedure has been completed, as it is only a tool to train the generator and not required to generate new motions at inference time.

7.1.3. Problems with GANs

While GANs allow for the possibility of generating stunning results, in their current form they also have a significant number of drawbacks that make their utilization harder than more conventional models and the training procedure unstable. The general excitement about GANs in the deep learning (DL) community is generally not considering the pitfalls and problems to be encountered when actually implementing a GAN based model. This is perhaps best summed up by the following quote:

> *A lot of people want to use GANs, they just don't know that they're unstable, until they get into using them...and then they're kind of stuck there.*
>
> *Soumith Chintala - Facebook AI ['How to train a GAN', NIPS 2016]*

This quote has unfortunately proven very true for this research. The first versions of the generative model were not showing any signs of progress at all. Therefore, a number of improvements and innovations were developed to alleviate the common problems encountered when (training) GANs.

This section will elaborate on the various problems encountered when training GANs, that are unlikely to occur when using more conventional DL techniques.

Progress Insight

For the regression model from Chapter 6 and other regression or classification models, a lower loss, by definition, equals an improvement in the desired output. In contrast, the loss value from a GAN actually has very little explanatory power with regard to the model's learning progression. As strange as this may seem at first, it is a logical conclusion from the setup as a two player game.

The first problem of this system is that both players are trying to affect the same score, meaning if both parties improve by an equal amount it might be impossible to tell quantitatively based on the final output alone.

The final loss is actually a measure of how well the discriminator is performing, but the actual desired goal is to train the generator, while the discriminator is but a means to an end. So why not define a quantitative measure for the output of the generator directly? If a direct quantitative measure to define the validity of the generator's output would exist, the whole setup of a GAN would not be required and the system could be directly optimized for this new criterion.

For an intuitive understanding , one can think of it as a game of tug of war:

The loss function is the rope, the loss value is the position of the rope, the generator and discriminator are the teams (A & B) pulling on either end of the rope and every new training iteration is a new round of the game. The DL researcher in this scenario can be seen as the coach that is trying to gage the strength of team A, but all he's given is a pinhole camera that can only see the middle of the rope. If both teams are equally strong, the rope does not move. After each round the teams might have become stronger or grown more tired than before, but if both parties improve or worsen by equal amounts, the rope still will not move no matter how strong the teams are pulling.

Algorithm 7.1 GAN Training Algorithm - Simplified

```
# ! DISCLAIMER: This is PyTorch inspired pseudo-code !

#...Load dataset
#...Setup Networks   (Generator, Discriminator)
#     and Optimizers (Generator, Discriminator, Pain)

for (leading_motion,following_motion) in dataset:

  #----------------------------------------
  # Train on fake/generated data
  #----------------------------------------

  #--- Generate new motion ---
  output_motion = []
  output_frame  = None # Initialized to zero vector
  for leading_frame in leading_motion:
    output_frame = Generator([leading_frame,output_frame])
    output_motion.append(output_frame)

  #--- Run through Discriminator ---
  g_pred = []
  for (leading_frame,output_frame) in zip(leading_motion,output_motion):
    g_pred.append(Discriminator([leading_frame,output_frame]))

  #--- Compute loss ---
  g_act_fake = 0                      # Generate Label(s)
  loss = BCE_Loss(g_pred,g_act_fake) # Compute Loss
  loss.backward()                     # Backpropagation

  #--- Perform optimization step ---
  Optimizers["Generator"    ].step()
  Optimizers["Discriminator"].step()

  #--- Apply Pain Criterion  ---
  pain = Pain_Loss(output_motion)
  pain.backward()
  Optimizers["Pain"].step()

  #----------------------------------------
  # Train on real/recorded data
  #----------------------------------------

  #--- Run through Discriminator ---
  g_pred = []
  for leading_frame,following_frame in zip(leading_motion,following_motion):
    g_pred.append(Discriminator([leading_frame,following_frame]))

  #--- Compute loss ---
  g_act_real = 1                      # Generate Label(s)
  loss = BCE_Loss(g_pred,g_act_real) # Compute Loss
  loss.backward()                     # Backpropagation

  #--- Perform optimization step ---
  Optimizers["Discriminator"].step()
```

A true measure for the improvement of both teams would e.g. be the total force exerted on the rope, however this would require advanced analytical equipment, which is currently not available. [1]

So as long as both networks remain in equilibrium the loss function does not change, but progress might still be occurring.

Training Instability

Now assume the case were the system does not remain in equilibrium: The researcher cannot only tell that one of the two systems is getting better than the other, but also which one. This means that insight into the system is temporarily gained, but at the cost of creating an inequality between the two players that might prove detrimental to the training procedure.

For an intuitive understanding , lets build on the analogy with a game of tug of war:

Once the equilibrium is broken it means that two teams of different skill levels are playing against each other. In most competitive sports the teams are put into different leagues based on their level. This is to ensure that teams with comparable levels are playing against each other. Consider the following two scenarios:

- Team A is considerably stronger than team B:
  Beating team B is too easy and requires no skill - No challenge is posed
  → No additional experience is gained
- Team A is considerably weaker than team B:
  Beating team B becomes impossible - Learning curve is too steep
  → No practical experience is gained

While this analogy cannot be applied to the GAN training procedure 1-to-1, the general idea behind it still holds true. The example above considers two players playing a symmetrical game, while in the case of a GAN the relation between the two players can be described much better as one between a student (generator) and a teacher (discriminator). In this relation it is usually common and desired to have an imbalance between the parties, with the teacher outclassing the student, while the student's ultimate goal still remains to surpass its teacher.

One might argue that in theory a perfectly pre-trained discriminator would be beneficial in this scenario. This intuition may however be flawed, as it might focus on specific details, while the generator might be producing results that are not even remotely close to the desired result yet. An important aspect to this progression is that the networks are initialized randomly, therefore resulting in initially random output. It would prove futile to teach a toddler a double pirouette, if it still struggles with the basic concept of balance. While this may initially sound silly, this is very much how the generator network should be regarded at the early stage of the training procedure, a toddler learning how to control the basic motor functions of its body. The main difference being that physical forces, such as gravity, are omitted, as this would reach into the realm of reinforcement learning (RL) instead. So just as the generator needs to adapt to grow from its random initial state to the desired final state, so does the discriminator need to adjust the level of detail it provides in the feedback.

In an optimal scenario the level of details would increase as time goes on and both networks improve, however there is also the chance that they get stuck in an infinite loop. A possible scenario that would cause this would be the following:

1. The discriminator finds criterion A to be out of line with the recorded data
2. Eventually the generator removes the incorrect behavior based on the discriminator's feedback about criterion A
3. Now criterion A is no longer a good discriminatory metric and the discriminator will move on to find another criterion B
4. Eventually the generator removes the incorrect behavior based on the discriminator's feedback about criterion B
5. The incorrect behavior detected by means of criterion A might return to the generator's output, now that the discriminator no longer focuses on criterion A in its feedback
6. The cycle repeats at 1.

[1] A potential solution to obtaining information on the internal state of a GAN and its progression is presented in section 9.5

This problem of selective and sequential optimization of an individual criterion, in contrast to simultaneous improvements of multiple criteria, is one representation of 'mode collapse' [Liu and Tuzel, 2016]. This problem is illustrated in Figure 7.2. The upper row represents the desired progression of learning all 8 classes present in the dataset simultaneously, while the lower row illustrates the problem of mode collapse, as the network jumps from class to class individually.

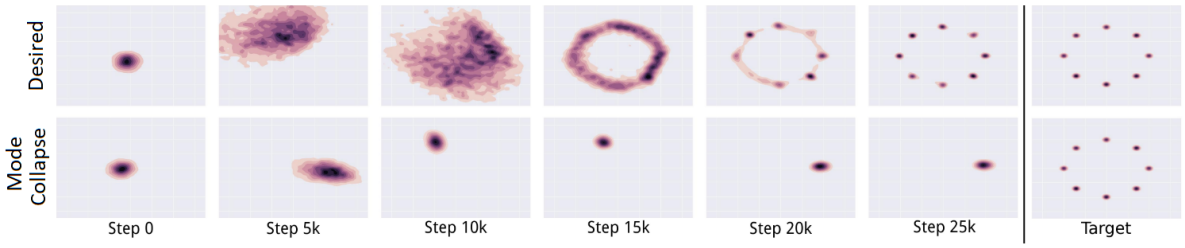


Figure 7.2: GAN: Mode Collapse Example
Source: Adapted from [Liu and Tuzel, 2016]

Balancing the training of the generator and the discriminator, such that they enhance each other complementary rather than outclassing the respective other, is a delicate process that is both hard to predict and hard to maintain.

7.1.4. Best Practices

> *[...] I will tell you all, or some, of the missing details that all research papers omit , because [...] it does not fit into their story , but if you start to train GANs you would need to know some of these things.*
>
> *Soumith Chintala - Facebook AI ['How to train a GAN', NIPS 2016]*

Paraphrasing a great talk by Soumith Chintala, at the *'Conference on Neural Information Processing Systems'* (*NIPS*) 2016, this section is intended to provide a set of practical best practices and the extend to which they have been applied to this research. Chintala's talk is an invaluable resource with respect to the practical implementation of GAN based models, as it includes a lot of 'missing details'.

1. "Normalize the inputs":
   → Implemented
   → see Section 4.4.1 [Data Normalization]
2. "Modify the loss function for the generator": $Min\ (\log(1 - D)) \rightarrow Max\ (\log(D))$
   → Implemented
   → Used *PyTorch* standard BCE loss implementation, but reversed learning rate on the generator optimizer to maximize the loss
3. "Use spherical Z": About sampling of latent vector Z
   → Not Applicable
   → No noise sampling is being done, as random noise input is replaced by input motion
4. "Use batch norm properly": Batch of only real or fake samples, no mixing within a batch
   → Implemented
   → see Section 5.1.3 [Hyperparameters] and Algorithm 7.1
5. "Avoid sparse gradients": e.g. no ReLU, Max-Pool
   → Implemented
   → Primarily for CNNs - All RNN activations are continuous (Sigmoid,TanH)
6. "Use soft and noisy labels":
   → Add stochastic value to labels, so (0+(0,0.3)|1-(0,0.3))
   → Not Implemented
   → see Section 9.1.2

7. DCGAN | hybrid model":
   → Not Applicable | Implemented
   → DCGAN only applicable for CNN based models | Adding the Pain loss and RNN base results in a hybrid model
8. "Apply stability tricks from RL": e.g. keep past models during training and inject them to check against them
   → Not Implemented
   → see Section 9.2.1
9. "Optimizer=Adam": Use Adam (and for discriminator sometimes SGD)
   → (Partially) Implemented
   → Adam is used for Generator and Discriminator
10. "Track failures early": Avoid losing time in training models that do not work
    → Implemented
    → Implemented Early Breaking (see Section 5.1.2 [Early Breaking]), as well live visualizations for continuous insight (see Section 8.2 [Live Visualization])
11. "Do not balance via loss statistics": Do not try to find a (dynamic) schedule for when to train which sub-network
    → Implemented
    → $K = 1$, see Algorithm 7.1
12. "If you have labels, use them": Hybrid model - train the discriminator to also classify the labels
    → Not Implemented
    → The current set of labels are highly biased and should not be used (see Section 4.1.4 [Labels])
13. "Add noise to input, decay over time": Add noise to every layer
    → Partially Implemented
    → Injection of Gaussian noise ($\mu = 0, \sigma = 1$) with a fixed scaling factor of $10^(-3)$ (in real life no two samples are the same)
14. "Train discriminator more (...sometimes)":
    → Partially Implemented
    → Generator is only trained in the fake data step, while Discriminator is trained in both. But, $K = 1$ (see Algorithm 7.1) Done, Training Discriminator/Generator 2/1

This research attempted to adhere to all of these best practices, if applicable. Those that have not been implemented are addressed later in Section 9 [Future Work].

7.2. Initial Design Problems

Building on the BHPO optimized regression model design, the discriminator was added to the model, analogous to the design of the generator, to allow for the GAN based training. As the generated output motions are of the same variable length as the input motions, it was decided to utilize the same basic 'Recurrent Neural Network' (RNN) unit for the discriminator as for the generator. This allows for frame-by-frame processing of the output motion by the discriminator.

This initial GAN showed no sign of training progression at all. In addition to the problems described in Section 7.1.3, this can be attributed to a combination of the following three factors:

1. **Incorrect hyperparameters:**
   - The previous model's task was incorrect and thus the optimization results cannot be transfered 1-to-1 and expected to perform as desired (see Section 6.4 [Feedback Dilemma])
   - Most notably: The learning rate was way too high (> 6)
2. **Too few parameters:**
   - The total number of trainable parameters, with a singular RNN layer, are as follows:
     - Generator: $N = 66$ & $H = 66$
       - LSTM : $4 \cdot \left(66^2 + 66 \cdot (66 + 2)\right) = 35.376$
       - GRU : $3 \cdot \left(66^2 + 66 \cdot (66 + 2)\right) = 26.532$
     - Discriminator: $N = 66$ & $H = 1$
       - LSTM : $4 \cdot \left(1^2 + 1 \cdot (66 + 2)\right) = 276$

- GRU : $3 \cdot \left(1^2 + 1 \cdot (66 + 2)\right) = 207$
- Especially for the discriminator $\approx 200 - 300$ parameters are far too few to learn any complex behavior, for the task given.

3. **Missing memory:**
   - The size of the LSTMs memory cell and the hidden state feedback signal are equal to the size of the output.
     → As the final output of the discriminator is of size 1, so are the states that store the network's memory over time
   - No complex information can be stored over time in a singular value.

Given that this initial design fails to generate any results, incremental improvements to the system were applied to enable training progression, in an attempt to generate novel motions.

7.3. Model Design Improvements

The details of all design improvements, developed to alleviate the previously mentioned issues, are described in this section:

1. Stacked RNN Units
2. Fully-Connected Final Layer
3. Derivative Output
4. Derivative Input
5. Soft-Self-Attention
6. Pain
7. Limited Judgement

The description of each design improvement is structured based on (at least) the following five segments:

1. **Architecture:** Schematic of the design
2. **Problem:** The current model's problem that is to be resolved
3. **Rationale:** The reasoning behind implementing this change, in relation to the problem
4. **Hyperparameters:** Definition of (new) hyperparameters introduced or used
5. **Trade-off:** Reviewing the benefits and downsides of the change

7.3.1. Stacked RNN Units

Architecture

Increasing the number of layers for a RNN results in stacking multiple RNN units on top of each other. Each unit has its own feedback signal into the future, resulting in a feedback loop for each additional unit in the stack. The schematics of a stacked RNN model is given in Figure 7.3.

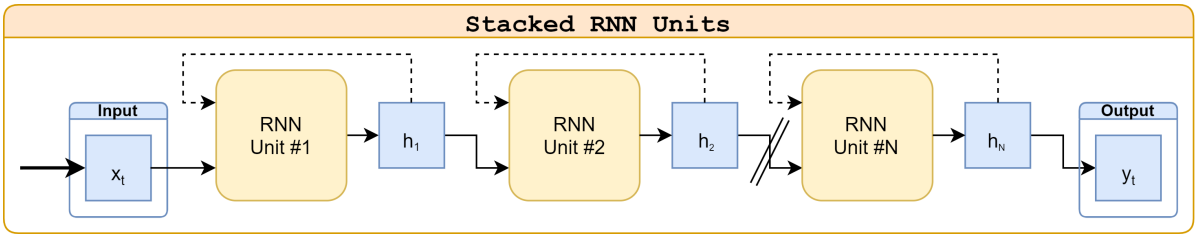


Figure 7.3: Neural network Architecture: Stacked RNN Units

Problem

As presented in Section 6.4.4 [Dual Feedback Problem] , a singular unit with the original design results in a dual feedback loop of the same output data. As presented in Section 7.2 [Initial Design Problems], the current model has no latent memory and too few parameters to learn complex interrelations.

Rationale

The 'deep' in deep learning actually refers to the utilization of multiple layers within the network. A model with a singular layer can therefore not truly be called a deep learning model. However, this is not true for RNNs, even with just a singular unit, due to the feedback loop built into the system. Each unit is passed through multiple times, when unfolded over time. This results in a very deep network, despite the utilization of only a singular RNN unit [Rumelhart et al., 1986].

However, analogous to stacking FC or CNN layers, adding multiple layers for RNNs still allows for multiple layers of abstraction in each processing step, resulting in the ability to learn more complex relations.

The introduction of multiple stacked layers was actually already present in the regression model, but the BHPO optimized resulting models, for both the LSTM and the GRU, required only a singular layer (see Section 6.3 [Results]).

Hyperparameters

For the chosen design, only a singular hyperparameter was added to the system: "# Layers". This parameter has already been introduced in Section 6.2.1 [Hyperparameters].

Stacking multiple RNN units means that the size of the latent vector in between layers can be chosen arbitrarily, analogous to choosing the number of neurons in a fully-connected layer. Making use of the default *PyTorch* implementation for training speed optimization, this means that on the first unit the output size is equal to the final desired output size and remain constant for all subsequent layers. This also avoids extensive HP scope creep, as defined in Section 5.2.2 [Scope Creep & Re-parameterization] If required, additional hyperparameters could be introduced that define the size of each individual unit.

Trade-off

- Pro:
  - \+ Dual feedback problem is resolved, as the first layer now feeds back a latent vector
  - \+ More non-linearities and training parameters: Allows the system to represent a more complex output function
- Con:
  - \- More training parameters and feedback loops: Even longer and increasingly nested output, increasing backpropagation runtime significantly
  - \- More complex systems usually take more data and time to train

Most of the benefits are already achieved with a layer size of 2, but introducing more layers is a trade-off between increased training time and additional parameters for the system to learn. Excessive amounts of layers should also be avoided to avoid over-fitting.

As it is hard to predict the correct number of stacked units upfront, the associated hyperparameter can best be optimized through BHPO.

7.3.2. Fully-Connected Final Layer

Architecture

Instead of using the output of the (stacked) RNN unit(s) as the final output of the model, an additional fully connected layer is placed at the end.

Problem

As presented in Section 7.2, given a final output vector size of 1 the discriminator's RNN units are defined to be of size 1 as well. This problem can be extrapolated to the general fact that the size of the in- and output vectors to the RNN units are fixed by the size of the original in- and output, this means that the number of parameters for the RNN layer are completely pre-determined and cannot be adjusted.

Rationale

Introducing an additional fully connected layer towards the end changes the nature of the RNN output from an output layer to a latent layer. This allows for choosing the size of the RNNs output arbitrarily and hence controlling the size of the latent memory, as well as the total number of parameters of the model.

When applying this to the initial model the drastic inequality between the two networks is alleviated. Using the LSTM based model as an example and setting both the generator's and discriminator's RNN output size equal to the input size of 66 floats, the total number of parameters are as follows:

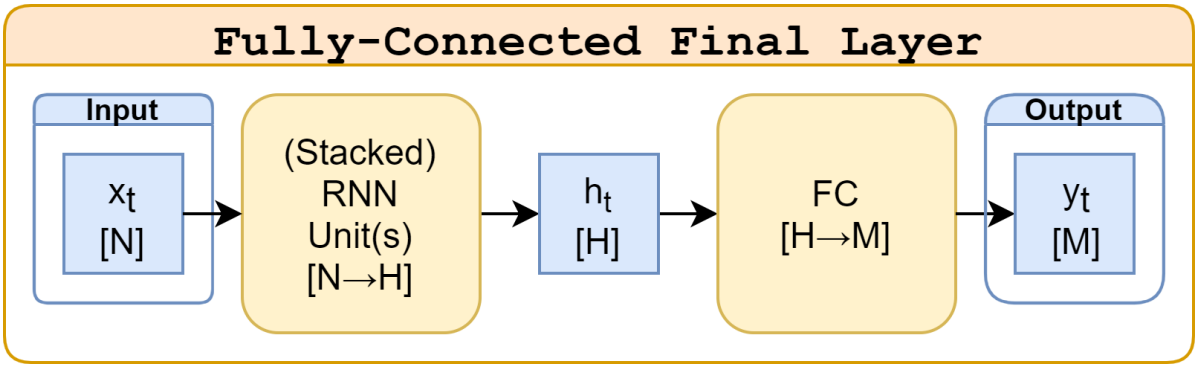


Figure 7.4: Neural network Architecture: Fully-Connected Final Layer

- Original design: (see Section 7.2)
  - Total: 35.652
  - Generator: 35.376 (99.2%)
  - Discriminator: 276 (0.8%)
- Original design + FC layer:
  - Total: 75.241
  - Generator:
    - FC: $66 \cdot 66 + 66 = 4.422$ (5.9%)
    - Total: $35.376 + 4.422 = 39.798$ (52.9%)
  - Discriminator:
    - FC: $66 \cdot 1 + 1 = 67$ ($< 0.1\%$)
    - Total: $35.376 + 67 = 35.443$ (47.1%)

The balance in the parameter count between the generator and the discriminator is restored to $\approx 50 - 50$ instead of the discriminator being dwarfed by the generator's size. This is beneficial as both networks are required to process the same type, dimension and amount of data.

Hyperparameters

For each network the size of the latent vectors can be freely chosen, introducing two new hyperparameters to the system. In practice these hyperparameters were re-parameterized and expressed in ratio to the input size, according to Equation 7.2, and are defined in Table 7.2.

$$H_{\{G,D\}} = \text{int}\left(N \cdot \text{OutputRatio}_{\{G,D\}}\right) \tag{7.2}$$

Equation 7.2: Neural network Architecture: Hidden layer size

Hyperparameters			
Parameter	**[Default] Value**	**Limits**	**Description**
G-OutputRatio	1.1	[1.0, 2.0]	Ratio of generator's latent space to the input
D-OutputRatio	0.75	[0.5, 1.0]	Ratio of discriminator's latent space to the input

Table 7.2: Hyperparameters: Output Ratios

By default the generator's latent size was chosen to be larger than the discriminator's, as the generator's final output is of higher dimensionality.

The ratio between the number of parameters of the generator and discriminator was by default kept at $\approx 33\% vs 66\%$ of the total.

Intuitively, one might also think that it is a lot harder and more complex to learn a new skill from scratch than to expose a flaw in a professionals performance. Generally, one is not required to be a dancer to judge if a performance was good or bad. This inequality in the tasks asked of the two networks is thus reflected in the total parameter count.

Given this defined ratio, the hyperparameters could alternatively be re-parameterized: One HP defines the ratio of the number of parameters between the generator and discriminator ($\approx 2/1$), and the other defines the common multiplier with the input dimensionality.

Trade-off

- Pro:
  - \+ RNN size is independent of output dimension
  - \+ Customizable control over the number of parameters in the network
- Con:
  - \- Introduction of additional hyperparameters required to be optimized

Implementing the final connected layers is essentially a necessity to resolve the extreme inequality between the two networks.

7.3.3. Derivative Output

Architecture

Instead of using the neural network's output as the final output, the neural network generates data related to the change in the final state. The output of the network is not exactly the state change, as this would cause the final state to be unbound. To prevent this, TanH is applied before the final output to keep the output within bounds. The flow diagram is given in Figure 7.5.

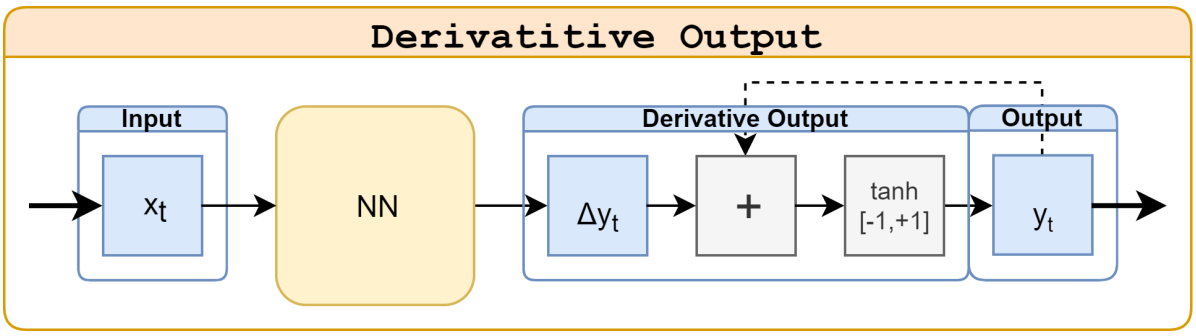


Figure 7.5: Neural network Architecture: Derivative Output

Problem

The neural network is intended to generate motions, but actually generates poses.

Rationale

The generator is intended to generate novel motions and the final output is required to be a full pose at any given time step. At first this may not seem like a contradiction, but looking closer there is a discrepancy between these two requirements. The fact that the final output is a pose is logical, given the final projection onto the drone swarm. Therefore, the final pose is the system's state. But generating novel motions requires the system to generate the change in its state rather than the actual state. Therefore, the network's output should be the state's derivative.

Given the relatively high frame-rate and the fact that a human is a physical system required to obey the laws of physics, such as inertia, it is undesirable to have large jumps between frames. Forcing the system to generate the state's delta instead, allows for finer-grained control and insight into the system: e.g. artificially limiting the system's derivative.

Intuitively, this means that the neural network learns how to 'move' its body, instead of how to 'pose' it.

After the fact, it was discovered that a similar method has been applied by Kratzer et al. [2020].

Hyperparameters

No new hyperparameters are defined by implementing this change. However, application of this modification or not is a decision made by the researcher that affects the overall system architecture. In theory, the usage of this option can be regulated by a boolean hyperparameter that activates it, but this was left a general setting not to be optimized by the BHPO.

Trade-off

- Pro:
  - \+ Next pose will only be a slight change from the previous
    → Ensures continuity
  - \+ Combined with internal delta of the LSTM, it mimics human control
    analogous to force/acceleration based on input
- Con:
  - \- Bounding the final output by means of a TanH function, means a recursive application of the TanH function
    This results in a vanishing state problem[2], which the neural network needs to correct for (see Figure 7.6)

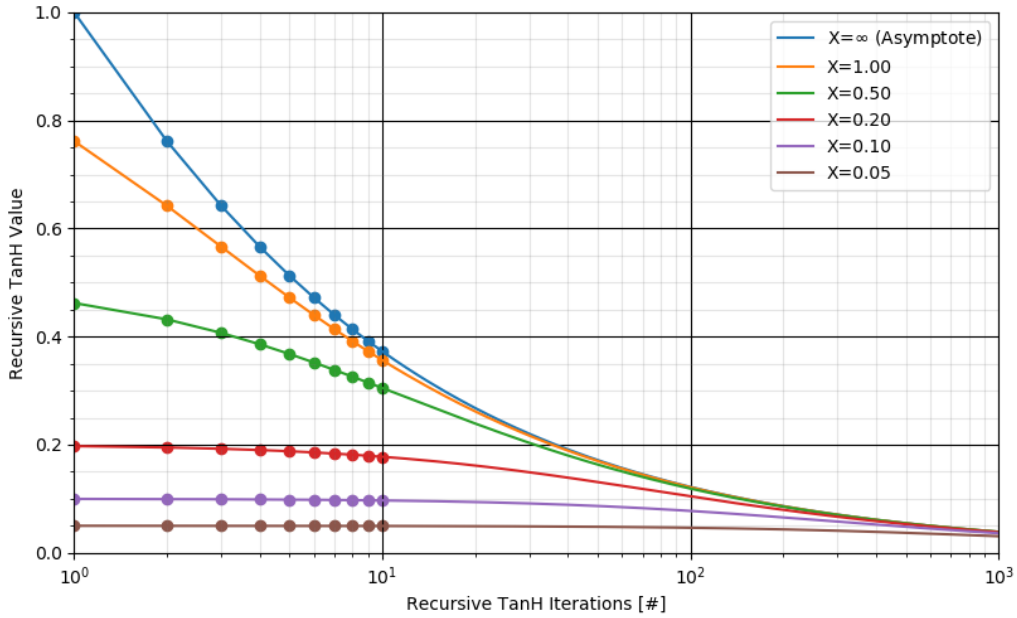


Figure 7.6: Derivative Output: Vanishing State Problem

7.3.4. Derivative Input

Architecture

In addition to the input and output poses used as input to the network, the changes in the in- and output state are included as well. The flow diagram is given in Figure 7.7.

Problem

The system has to learn everything from scratch, while there are certain known derived data representations that are expected to be useful to the network. This wastes computational resources, as the network is required to learns these representations instead of providing them upfront as an input to the system.

Rationale

Building on the trivial solution to the regression problem, as presented in Section 6.4.3, it becomes clear that the system first needs to learn about inertia, before learning more complex interrelations. The 'Backward Differentiation Formulas' (BDFs) provide a clear solution for the best estimate of the system's state derivative

[2]The vanishing state problem is a result of iteratively applying the *tanh* function to the same data. This results in the maximum possible output value (asymptote) to shrink with every recurrent iteration (see Figure 7.6).

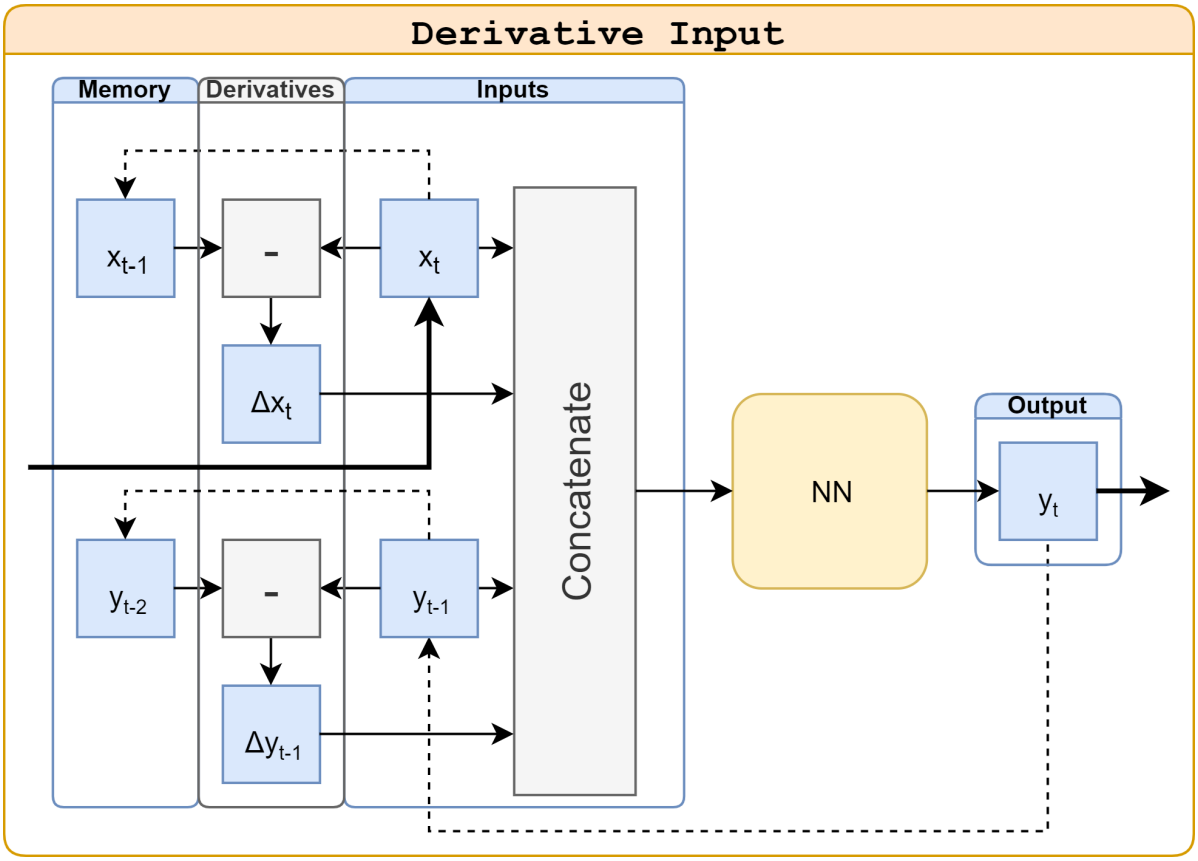


Figure 7.7: Neural network Architecture: Derivative Input

at the next time-step in discrete-time. The BDFs first order solution simplifies to the backward Euler method, where the current derivative is estimate to be equal to the one of the previous time step. Computing the first order discrete derivative prior is trivial and passing it into the network saves the network the trouble of having to learn that this information is relevant.

Intuitively, it can be compared to providing a student with a formula sheet during an exam: The student should be able to learn the formulas by heart and solve the problem without them, but providing them makes solving the actual problem easier. This is because it is not desired for the student to spend resources remembering the formulas by heart, when the actual task merely requires them as a means to an end.

After the fact, it was discovered that a similar method, without the final normalization layer, has been applied by Kratzer et al. [2020].

Hyperparameters

No new hyperparameters are defined by implementing this change. However, application of this modification or not is a decision made by the researcher that affects the overall system architecture. In theory, the usage of this option can be regulated by a boolean hyperparameter that activates it, but this was left a general setting not to be optimized by the BHPO.

Trade-off

- Pro:
  - \+ Feeding network relevant data directly
    Avoids network having to learn it explicitly
- Con:
  - \- Increases number of input- and training parameters by a factor of 2

7.3.5. Soft-Self-Attention

Architecture

The soft-self-attention layer is a neural network based data pre-processing step that passed through the relevant and masks out less relevant parts of the data [Bahdanau et al., 2015]. The flow diagram is given in Figure 7.8 and the exact *PyTorch* code is given in Algorithm D.2.

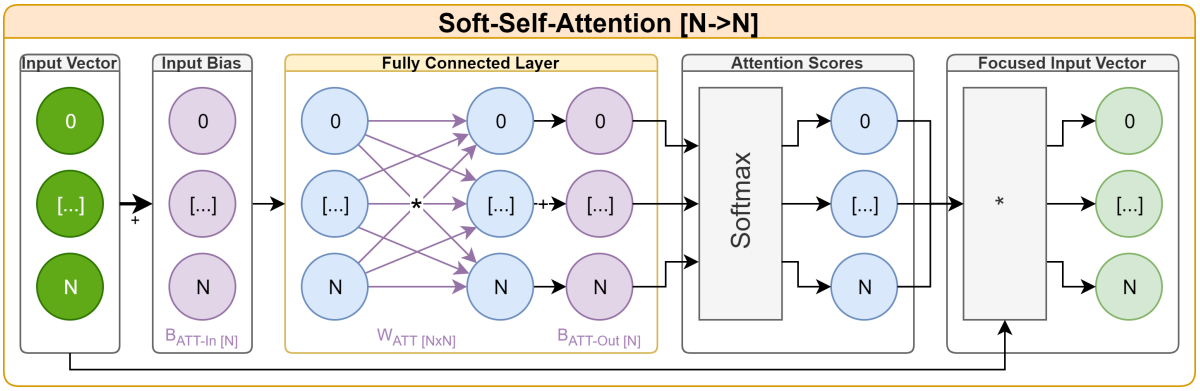


Figure 7.8: Neural network Architecture: Soft-Self-Attention

In addition to the usual configuration of this layer an input bias layer was added, which should allow for learning corrections for non-zero mean input variables (see Table 4.1).

Problem

Neural networks are black-box systems which make it hard for researchers to gain insight into the internal mode of operation of the system and make informed decisions in response.

Based on the derivative-input (see Section 7.3.4) and the 6D rotations (see Section 4.4.1) modifications made to the system, the total number of input parameters has increased considerably: A singular Euler based motion frame consists of 66 floats, but a 6D based motion frame consists of 129 floats. Given both modifications the total input parameters increases from $(2 \cdot 66 =)$ 132 to $(4 \cdot 129 =)$ 516.

Furthermore, applying the 'Principal Component Analysis' (PCA) transform (see Section 4.4.1) changes the motion frame data to a ranked structure, where later PCA components have less influence on the overall motion and might be less relevant for processing.

Rationale

Applying an attention layer prior to processing has the following two advantages:

- Focus: The network focuses on the relevant data, while ignoring the remainder
- Insight: Allows insight into which segments of the input data are actually used by the networks

Attention mechanisms have the unique benefit that they do not directly generate an output, but a mask [sum=1] that is applied to the original input data. As both the input and output of the network are of the same dimension and order, this allows for a human interpretation of the network weights. The weights of the FC layer can be interpreted similar to an autocorrelation matrix. Examining the weights displayed in Figure 7.9 it becomes apparent that ...:

- ... the generator focuses primarily on the Δs, mostly ignores the output pose, but overall uses all data
- ... the discriminator focuses primarily on the output pose, while ignoring most other input data

This behavior is expected as...:

- ... the generator is tasked with generating motions
  → Focus on Δs
- ... the generator has to take in new information of the leading dancer, while the output dancer's pose is previous information
  → Focus on leading pose over output pose
- ... the generator should judge all data in context
  → Overall wide focus

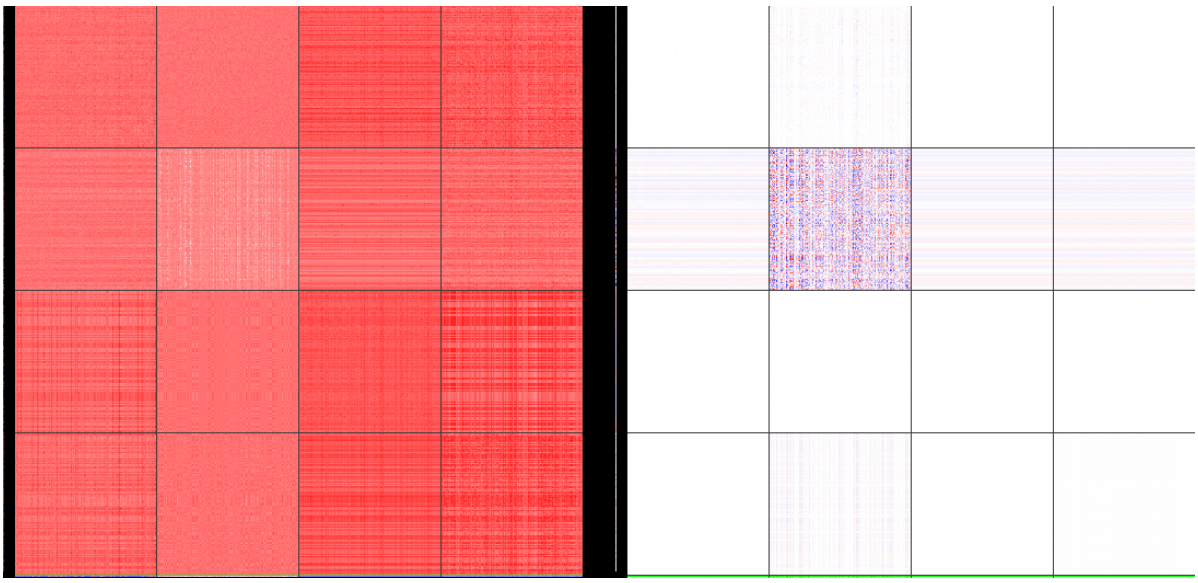


Figure 7.9: Attention: Attention Weights Example
Blue: Negative parameters & White: Zero Parameters & Red: Positive Parameters
Left: Generator Weights & Biases
Right: Discriminator Weights & Biases
Top↓Bottom & Left→Right: Leading pose, Output pose, Δ Leading pose, Δ Output pose
1px line on the left and bottom: Input- and Output Biases

The application of an attention mechanism changes the nature of the system from a pure black-box system to a grey-box system.

Overall, application of attention mechanisms have proven highly beneficial to processing data with neural networks [Bahdanau et al., 2015; Dong and Xu, 2019; Vaswani et al., 2017; Xu et al., 2018].

Hyperparameters

No new hyperparameters are defined by implementing this change. However, application of this modification or not is a decision made by the researcher that affects the overall system architecture. In theory, the usage of this option can be regulated by a boolean hyperparameter that activates it, but this was left a general setting not to be optimized by the BHPO.

Trade-off

- Pro:
  - \+ Attention mask filters less relevant data
  - \+ Dynamic masking, based on input data
  - \+ Enables insight into which parts of the data are being used
- Con:
  - \- Increases complexity and number of parameters
  - \- Softmax layer can cause numerical instability issues

Weight Initialization

Unlike all other weights that are initialized with uniform random distribution, the weights of the attention layer have to be initialized in a non-biased manner.

To prevent undesired discrimination of some parameters, the weights have to be initialized as follows:

- Weights: All weights $= \frac{1}{N}$, to ensure every attention score is equal
- Biases : All zeros, to avoid a direct bias

7.3.6. Pain

Architecture

The pain optimizer introduces an additional optimizer with a custom loss criterion, as defined in Algorithm 7.2. The pain loss makes use of the min/peak/max values for each channel, as defined in Table 4.1.

Algorithm 7.2 Pain Loss

```
def PainLoss(v,min,max,peak,r=8):
    _l =  (v-peak)/(min-peak)*(v<peak) + (v-peak)/(max-peak)*(v>=peak)
    return torch.pow(_l,r)
```

The loss function is designed with the following five properties in mind:

- Continuously differentiable in $\mathbb{R}$
- $= 0$ at the peak
- ≈ 0 within the limits
- $\rightarrow \infty$ when the limits have been exceeded
- $= 1$ at the upper- and lower limits

An example of a channel's histogram, min/peak/max and the associated pain loss function is shown in Figure 7.10.

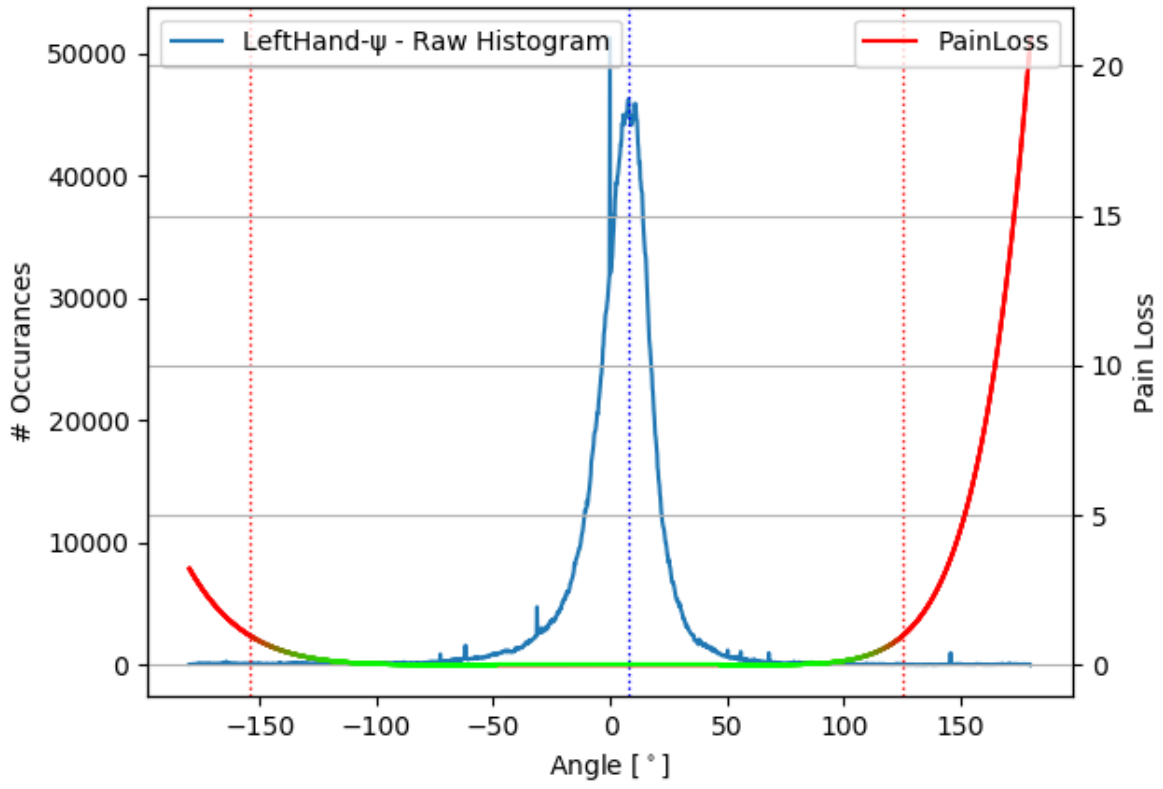


Figure 7.10: Pain: Loss Example

The pain loss is not applied when $min = -180° \& max = +180°$ for a specific channel, as this means that any value is allowed and no penalty should be added.

Problem

The model is initialized randomly with no constraints for the output defined, causing almost all channels in the initial output to exceed the limits. This problem is greatly alleviated when applying the PCA transform (see Section 4.4.1), as it normalizes the data into the regions actually present in the data, but it does not fully avoid the issue of exceeding the physical limits.

Rationale

A human learning how to dance is not required to learn his physical limits, they are known intuitively and met with a sharp pain when exceeded. The pain criterion makes use of biomimicry to transfer the intuitive mechanism of pain into the generator network. Hard limits are difficult to implement in neural network based systems, as they learn by means of backpropagation through a fully differentiable set of equations and hard limits introduce sharp discontinuities. The design of the pain criterion is therefore designed as a soft limit with an exceedingly large gradient and loss the further from the limit the output gets.

Humans instinctively know their limits, why shouldn't the computer? Some say: *"Pain is the best teacher"*.

Furthermore, as presented in Section 7.1.3 the BCE loss is affected by both networks and does not accurately reflect the networks improvement.

Considering the pain loss in conjunction with the BCE loss allows for more insight into the training progression. To illustrate this, some of the training runs had wildly changing pain losses, while the discriminator's output was consistently stagnant and indecisive at 0.5/0.5 for the entire run. In this case, just looking at the BCE loss would lead to the believe that the training is making no progress, while the pain loss clearly shows otherwise.

An example of the progression of joint limits is shown in Figure 7.11. This example does not utilize PCA processing and shows an inconvenient initial condition. On average the limits are exceeded by a factor of $\approx 6-7x$ at the first iteration and a factor of $\approx 2-4x$ during the first 500 iterations. After the chaotic initial phase a clear progression towards the limit can be seen.

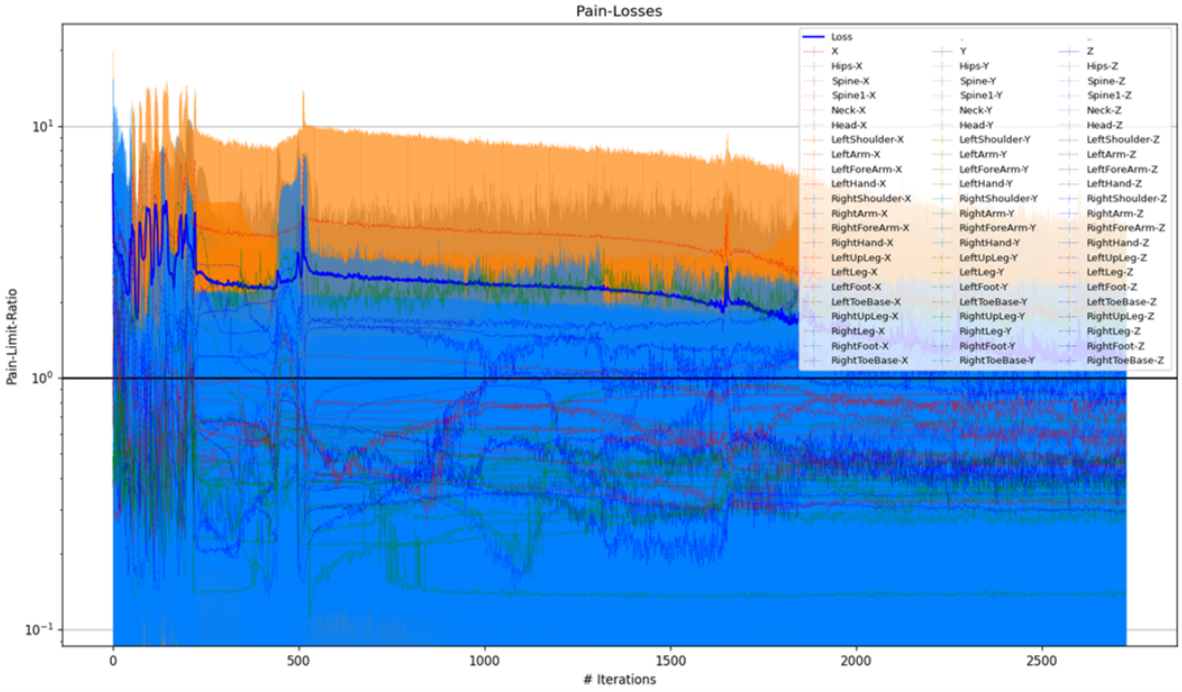


Figure 7.11: Pain: Losses over Time Example

The loss is transformed by $X^{\left(\frac{1}{r}\right)}$ to allow for easier interpretation by humans.
Only the thick blue line is of direct relevance, as it indicates the transformed mean of all individual losses.

The pain criterion alone works well for correcting limits that are exceeded very far, but the decreasing gradients towards the limit causes progression to stagnate. This shortcoming is alleviated by applying the

PCA transform to the data. The PCA transform causes the data to initially stay within limits and the pain criterion corrects for channels shooting out of bounds, as the generator explores the solution space. This synergy can be observed in Figure E.14.

Hyperparameters

The pain criterion introduces two new hyperparameters:

- Pain-Optimizer learning rate: Equal to global learning rate
- Pain-Criterion Exponent: 8

These parameters were not integrated into the BHPO to avoid parameter scope creep.

The pain optimizer's learning rate was set equal to the other two optimizers learning rates to avoid inequalities between them.

The pain-criterion exponent was iteratively improved to the acceptable value of 8. The following cases serve as a guideline for some of the regions for r:

- $r < 1$ Desired gradient progression is inverted
- $r = 1$ Gradients on each side are constant and discontinuous at the peak
- $5 < r < 10$ Acceptable region
- $r \gg 10$ Gradients are too steep, such that corrections overshoot

Trade-off

- Pro:
  - \+ Adds additional penalty to avoid exceeding of physical limits
  - \+ Allows for additional insight into the training progression
- Con:
  - \- Adds additional complexity
  - \- Artificially skews output towards peak
  - \- Only affects generator and not discriminator

7.3.7. Limited Judgement

Architecture

The generator generates a new motion frame-by-frame, which is subsequently passed to the discriminator frame-by-frame. Instead of feeding the entire newly generated motion into the discriminator, the first few frames are removed and this truncated version is passed into the discriminator. A visualization of this procedure is presented in Figure 7.12.

Problem

The final output and hidden states of the network are initialized in the same fashion every time. The initial condition at $t = 0$ is therefore always the same.

This opens up the possibility for a trivial solution for the discriminator, which should be avoided. All the discriminator needs to do to distinguish the generated and the real motions is to the detect the initial configuration of the generated motion. As the system utilizes recurrent units that can be used as memory to store information over time, all the discriminator needs to do is hold this believe for the entire motion. Once correctly detected at $t = 0$, there is no incentive for the discriminator to change the preset believe, as it will remain correct.

Rationale

"It would be unfair to have a dance critic judge a dancer's stage performance from the moment on (s)he's still getting ready in the dressing room." So why is the network doing this?

Any system that has an imperfect initial condition suffers from a transition period until the system converges, e.g. a Kalman filter usually requires a few iterations before the filtered estimate should be trusted.

If the discriminator is allowed to judge this initial period, which is by definition sub-optimal, it opens up a shortcut to the correct answer, which does not reflect the desired solution. If the discriminator learns to

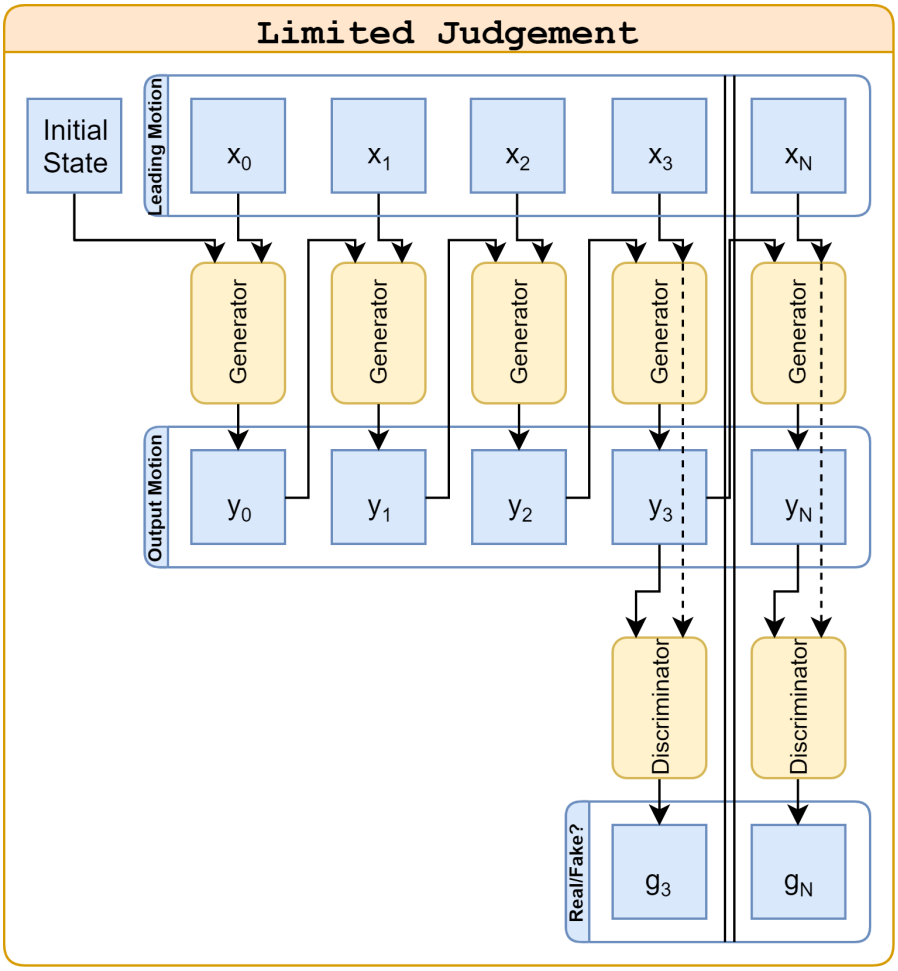


Figure 7.12: Neural network Architecture: Limited Judgement

discriminate on the wrong cues, this means that the generator learns by receiving incorrect feedback. Subsequently the generator would try to correct for the initial offset as fast as possible, but will never be able to outpace the discriminator. This results in an excessive Δ early on, as the generator tries to correct for the initial state by approximating a large step input. This behavior is however in stark contrast to the desired smooth and continuous transitions in the output that the recorded data exhibits, which it is supposed to mimic.

Truncating the transient that corrects for the initial state gives the generator a few frames the time to 'get ready', before being judged by the discriminator.

An intuitive example for thinking about the persistent cue: If one watches a video and is tasked to judge if it is real or fake and there is even a single moment that somebody is floating upside-down, it would be known for the rest of the video that it is fake. Hence, a single cue can be enough to 'make up somebody's mind' for a long time.

Hyperparameters

The number of frames to truncate is the only newly introduce variable (see Table 7.3).

Hyperparameters			
Parameter	**[Default] Value**	**Limits**	**Description**
# Truncated Frames	25	$[1, 100]$	Number of frames to remove for discriminator

Table 7.3: Hyperparameters: Limited Judgment

This hyperparameter could be re-parameterized to be expressed in seconds instead, making it independent of the frame rate. Tuning this parameter is not trivial for the following reasons:

- Truncate too few frames: The problem described prior manifests.
- Truncate too many frames: The generator 'receives' no feedback, over an extended period of time, making it harder to learn and build up the desired motion.

Trade-off

- Pro:
  - \+ Avoids presenting a trivial solution for the discriminator
- Con:
  - \- Introduces another hyperparameter to be optimized
  - \- If too many frames are truncated the initial phase receives limited feedback.

Hidden State Initialization

To understand the need for the limited judgment method, one must consider the way the network state is initialized. Due to the derivative input, the change to the system's output is incremental. Thus, the first few iterations the output is close to the initial state. The system is initialized with two separate poses depending on the data pre-processing that is used:

- <.BVH> Euler angle representation
  - Initial pose: T-Pose (see Figure 3.1)
  - Initial output vector: Zero-vector
- 6D Rotations:
  - Initial pose: T-Pose (see Figure 3.1)
  - Initial output vector: Zero-vector transformed through 6D rotation transform
- 6D Rotations with PCA transform:
  - Initial pose: PCA mean pose (see Figure 4.14)
  - Initial output vector: Zero-vector

Both dancers were instructed to start and end each take in the T-Pose. This makes initialization of the output in the T-Pose is a logical choice.

Once the PCA transform was added, the PCA mean pose was chosen as the initial pose. This is because, it represents the mean pose of the dataset, which is on average the one closest to any desired state of the system.

All other hidden states were initialized as zero-vectors, as not to introduce a bias at T_0.

7.4. DRA-GAN

The final model is comprised of the initial GAN, augmented with all the improvements described in the previous sections. The ID of this proposed model was originally '**6D-PCA-ATT-DI-DO-LSTM-Pain-LJ-GAN**', but replaced in favor of a more descriptive name: The 'Differential Recurrent Attention GAN' (DRA-GAN).

The full diagram of the DRA-GAN framework is presented in Figure 7.13, combining all the elements previously introduced individually.

The algorithm visualized in the diagram is summarized sequentially in Algorithm 7.3, including references to the relevant sections for each element.

In summary, this proposed model...:

- ... combines the strength of various neural network architectures
  (see 6.1.2 [RNN: LSTM vs GRU] & 7.1 [Generative Adversarial Network])
- ... solves the feedback dilemma
  (see 6.4 [Feedback Dilemma])
- ... solves the initial design problems
  (see 7.2 [Initial Design Problems])
- ... adheres to the best practices for training GANs
  (see 7.1.4 [Best Practices])

Algorithm 7.3 DRA-GAN Processing Framework

0. **Initialization:**
   (a) **Load Training Framework:** (see 5.1 [Training Framework])
   (b) **Initialize Hidden States:** (see 7.3.7 [Hidden State Initialization])
1. **Load Data:**
   (a) **Load Raw Motions:** (see 4.2.3 [*ArangoDB*])
   (b) **Remove Fingers:** (see 4.3.2 [Skeletal Simplification])
   (c) **Filter Motions:** (see 4.3.3 [Data Filtering])
   (d) **Transform to 6D Rotations:** (see 4.4.1 [6D-Rotations])
   (e) **Apply PCA Transform:** (see 4.4.1 [Principal Component Analysis])
2. **Run Generator:**
   (a) **Generate Motion Frame-by-Frame:**
      i. **Prepare Input:** (see 7.3.4 [Derivative Input])
      ii. **Focus Attention:** (see 7.3.5 [Soft-Self-Attention])
      iii. **Run RNN Core:** (see 6.1.2 [RNN: LSTM vs GRU] & 7.3.1 [Stacked RNN Units])
      iv. **Reduce Output Dimensionality:** (see 7.3.2 [Fully-Connected Final Layer])
      v. **Integrate and Limit State:** (see 7.3.3 [Derivative Output])
      vi. **Store Generated Frame:**
      vii. *If Motion not Completed: Jump to 'i.'*
   (b) **Run Pain Optimizer:** (see 7.3.6 [Pain])
      i. **Compute Pain Loss:** (see Algorithm 7.2)
      ii. **Backpropagation:**
      iii. **Pain Optimization Step:** (see 5.1.3 [Hyperparameters] for governing HPs)
3. **Run Discriminator:**
   (a) **Select Following or Output Motion:**
   (b) **Discriminate Motion Frame-by-Frame:**
      i. **Prepare Input:** (see 7.3.4 [Derivative Input])
      ii. **Focus Attention:** (see 7.3.5 [Soft-Self-Attention])
      iii. **Run RNN Core:** (see 6.1.2 [RNN: LSTM vs GRU] & 7.3.1 [Stacked RNN Units])
      iv. **Reduce Output Dimensionality:** (see 7.3.2 [Fully-Connected Final Layer])
      v. **Store Generated Guess:**
      vi. *If Motion not Completed: Jump to 'i.'*
   (c) **Run GAN Optimizer:**
      i. **Compute BCE Loss:** (see Equation 7.1) & Table 7.1
      ii. **Backpropagation:**
      iii. **GAN Optimization Step:** (see 5.1.3 [Hyperparameters] for governing HPs)

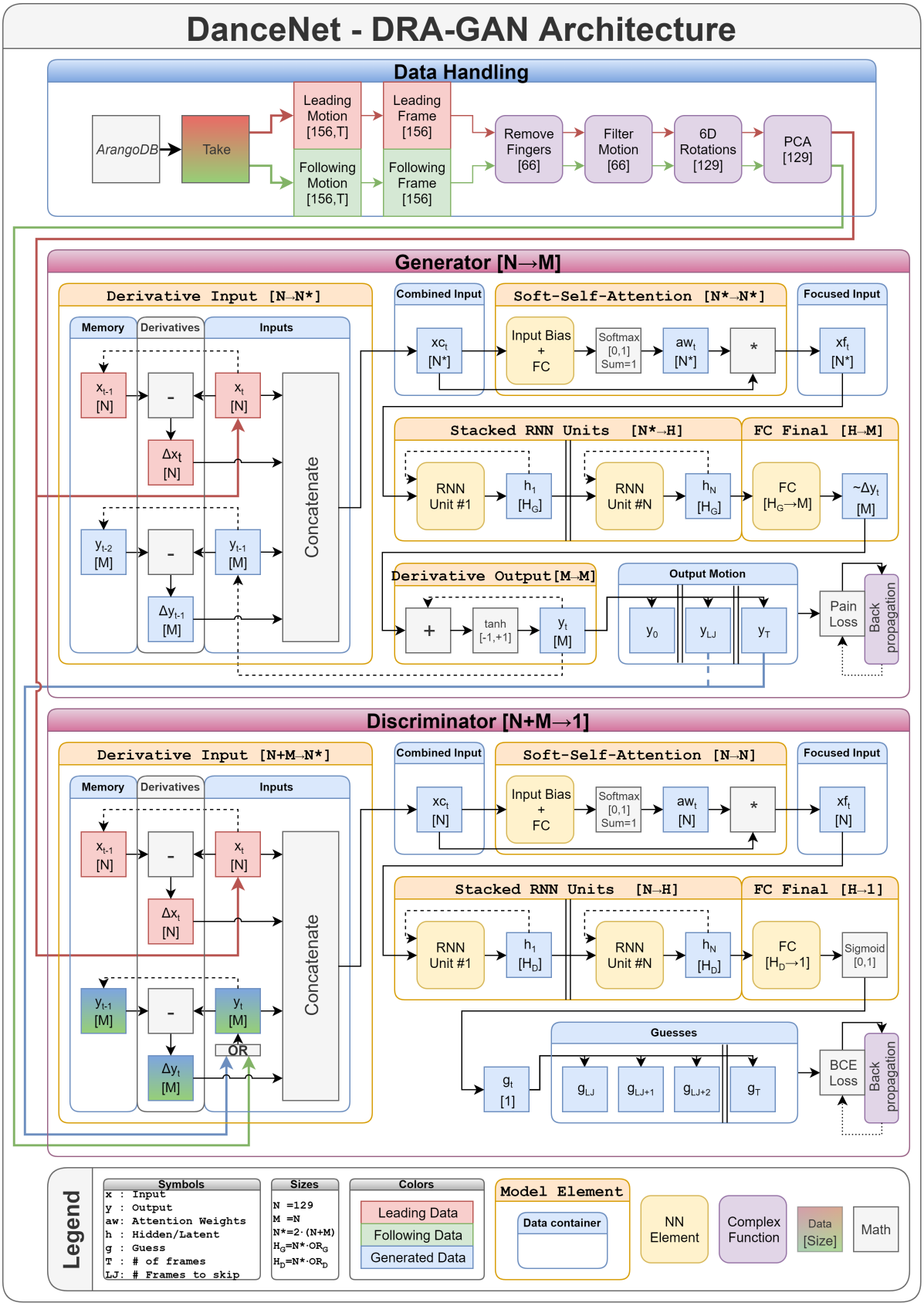


Figure 7.13: Neural network Architecture: DRA-GAN DanceNet

Hyperparameters			
Parameter	**[Default] Value**	**Limits**	**Description**
Network Size			
# Layers	3	[2, 5]	Number of stacked recurrent units
G-OutputRatio	1.1	[1.0, 2.0]	Ratio of generator's latent space to the input
D-OutputRatio	0.75	[0.5, 1.0]	Ratio of discriminator latent space to the input
Pain			
Pain learning rate	Equal to global learning rate	(see Section 5.1.3)	Learning rate of the pain optimizer
Pain Exponent	8	[5.0, 10.0]	Pain loss: limit-exceedance-ratio exponent
Limited Judgement			
# Truncated Frames	25	[1, 100]	Number of frames to remove for discriminator

Table 7.4: Hyperparameters: DRA-GAN

7.4.1. Hyperparameters

Table 7.4 summarized all hyperparameters newly introduced in the DRA-GAN model.

7.4.2. Weights and Biases

Compared to the very small network size of the regression model (see Section 7.2), the full DRA-GAN model contains millions of trainable parameters. This overall size is in line with comparable recurrent networks for motion generation or prediction, such as [Huang et al., 2020; Kratzer et al., 2020; Mao et al., 2020] utilizing models with parameter counts in the millions.

The bulk of the trainable parameters is concentrated in the RNN core, with the attention and final linear layers only accounting for 5% of the total. The number of parameters in the attention and linear layers are completely determined by the dimensionality of the motion frame. They are therefore only influenced by the choices whether or not to use the 6D-Rotations transform and to remove the finger joints. The number of parameters in the RNN core is determined by the number of stacked layers, as well as the set output ratios for the generator and discriminator, respectively. By default the output ratios are chosen such that the generator contains approximately double the number of parameters than the discriminator, as the generator is expected to have the more complex task (see Section 7.3.2). A complete parameter distribution example is provided in Table 7.5.

DRA-GAN # Parameters		
Network	**# Parameters**	**% Total**
Attention Layers:		
Generator	267,288	2.24%
Discriminator	267,288	2.24%
Recurrent Units:		
Generator	7,539,552	63.18%
Discriminator	3,785,888	31.73%
Linear Layers:		
Generator	72,885	0.61%
Discriminator	387	< 0.01%
Combined:		
Generator	7,879,725	66.03%
Discriminator	4,053,563	33.97%
Total	11,933,288	100%

Table 7.5: DRA-GAN: Parameter Distribution Example
3 layer LSTM with G-OutputRatio = 1.1 & D-OutputRatio = 0.75

Figure 7.9 displays how visualization of the attention weights can enable the researcher to gain insight into the network's mode of operation. This visualization was extended to incorporate the complete network's weights and biases. Figure 7.14 visualizes these combined network parameters.

The example given is for a 3 layer LSTM and is intended to introduce the various elements and how to interpret the DRA-GAN weight visualizations. The left half of the figure displays the parameters belonging

to the generator and the right half those belonging to the discriminator. The square $4x4$ grids on top are the attention weights, such as previously displayed in Figure 7.9. The segments enclosed by the blue dotted lines represent the RNN core parameters. Each set of two columns within these represent the parameters of a singular unit in the stack. In the first column are the parameters applied to the input and in the second those affecting the hidden state feedback signal. Each row represents the weights of a specific gate within the RNN unit (4 for LSTMs & 3 for GRUs). The small rectangle, next to the RNN core parameters, indicates the parameters belonging to the final FC layer. As the output dimensionality of the discriminator is 1, so is the width of the discriminator's final layer but a single pixel in width and can therefore not be distinguished in this example. The general flow of data is indicated by the orange dotted line, going through each of the major segments sequentially. The color scheme is the same as for the attention weights with red and blue indicating positive and negative parameters respectively and parameters (close to) zero displayed in white. The green and black lines are merely for visual separation of the various subsegments within a parameter block.

The interpretation of these weights will later be addressed separately in 7.5 [Results].

7.4.3. Advanced Loss

As presented in Section 7.1.3, the GAN based model makes it hard to make an objective quantitative assessment of the model's performance.

In addition to the limited insight, a secondary problem became apparent when training the GAN based model within the training framework (see Section 5.1). The early break criterion (see Section 5.1.2) was originally designed for models with a steadily decreasing loss function. The criterion was adapted to allow for a positive gradient in the final loss, but the same reason that causes limited insight also causes stagnation of the BCE loss at times. When the system is in equilibrium the loss stagnates and the early break criterion triggers, while this might not actually indicate a stagnation in the training progression.

The final issue with the current loss is the numerical equivalence of the loss function at the start of training compared with the desired final state. To understand this, a closer examination of the three stages of training is required: At the start of the training the weights are initialized randomly. With the positive and negative weights in the final layer having equal probability they cancel each other out on average. This results in the final FC output to be ≈ 0, which passed through the sigmoid activation function outputs a default value of ≈ 0.5. So, the discriminator is uncertain about the classification of the data and is essentially guessing randomly '50/50'. During the training, it is desired for the discriminator to be able to correctly classify the motions, as this means that the feedback provided to the generator is correct and results in steps taking in the right direction. At the end of the training, the generator is desired to generated motions so well that they are indistinguishable from the real motions. This desired final state forces the generator back to a state of guessing randomly, as there is no way of telling the real and fake data apart. However, this means that the network is initialized in the same configuration as the final desired state, making them numerically equivalent.

The following three issues have to be resolved:

- Insight into the system's performance and training progression is severely limited
- Stagnation of discriminator loss should not trigger early breaking, if training is still progressing
- 50/50 guessing at initialization and at the end of training should not be treated the same way

To alleviate these issues, a few modifications were made to the default loss function. While this modified loss does not provide an absolute metric for assessing the network's training progress, it serves as a better starting point to access the process' progress.

Discriminator Metrics

At each iteration the discriminator is being trained on one real and one fake sample and produces a series of guesses over time. Analyzing these time series predictions by the discriminator allows for computing a set of metrics from them that go beyond the singular value that the BCE loss provides.

An example of the discriminator's guess for a full motion chunk is presented in Figure 7.15. The lower plot shows the time trace of the discriminator's guess for the real (green) and fake (red) input. The upper plot shows a histogram of the same time traces, as well as the mean and standard deviation for each of the four quadrants. The upper histogram indicates the output for the real data and the lower histogram that of the fake data. In this particular example the discriminator is ...:

Figure 7.14: DRA-GAN: Network Parameters Visualization Example
Blue: Negative parameters & White: Zero Parameters & Red: Positive Parameters
Green & Grey: Logical separators

- ...fairly confident in identifying the real motion, with all guess consistently being true positives.
- ...uncertain in identifying the fake motion, with about half the guesses over time being false negatives. However, considering the time trace it does appear that the discriminator becomes better at identifying the fake data correctly as time progresses.

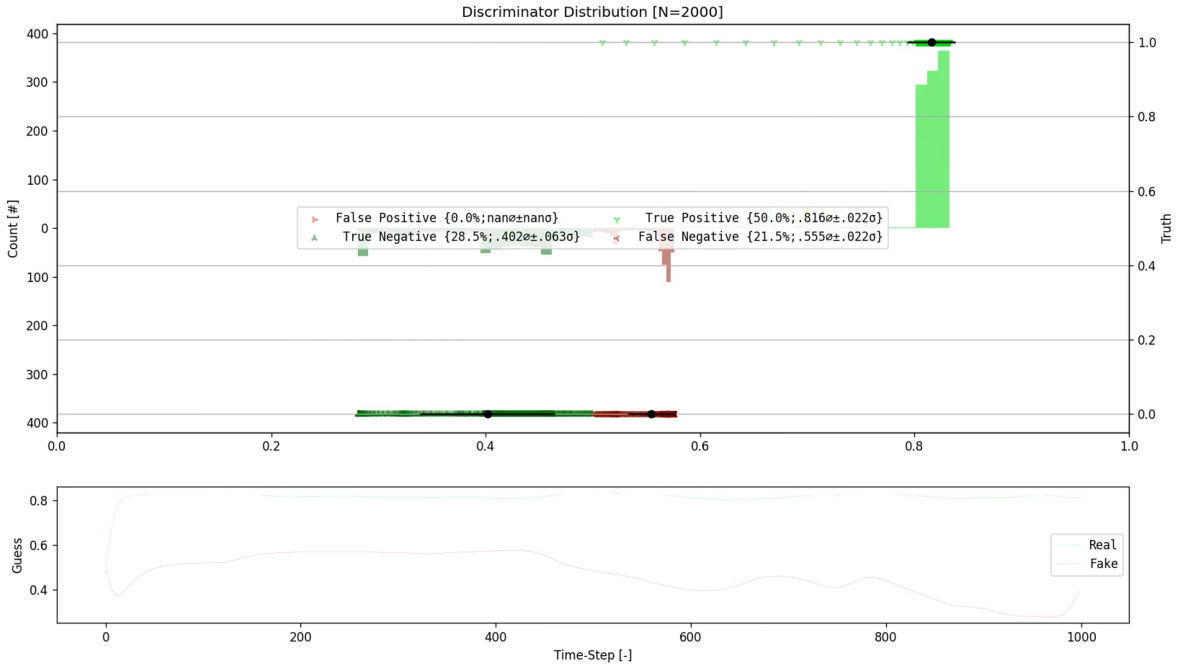


Figure 7.15: DRA-GAN: Discriminator Guesses Example

Based on these time series, the minimum, maximum, median, mean and standard deviation of the guesses can be extracted, for each type of input data (real & fake). Given these metrics an additional set of derived metrics can be computed that reflect the discriminator's performance. Equation 7.3 shows the computation of these metrics. [3]

$$
\begin{aligned}
\mu_{\text{Real}} &= \frac{\sum_{i=0}^{N} g_{\text{pred}_{\text{Real}}}}{N} && \text{| Real-Mean} \\
\mu_{\text{Fake}} &= \frac{\sum_{i=0}^{N} g_{\text{pred}_{\text{Fake}}}}{N} && \text{| Fake-Mean} \\
\mu_{\text{Bias}} &= \frac{\mu_{\text{Real}}+\mu_{\text{Fake}}}{2} && \text{| Mean-Bias} \\
\Delta\mu &= |\mu_{\text{Real}} - \mu_{\text{Fake}}| && \text{| Mean-Distance} \\
\Delta\mu_{\max} &= \max\left(\{\Delta\mu_0, ..., \Delta\mu_t\}\right) && \text{| Max-Distance} \\
d_{\text{Real}} &= |\mu_{\text{Real}} - 0.5| && \text{| Real-CenterOffset} \\
d_{\text{Fake}} &= |\mu_{\text{Fake}} - 0.5| && \text{| Fake-CenterOffset} \\
d &= d_{\text{Real}} + d_{\text{Fake}} && \text{| Mean-CenterOffset} \\
\text{Loss}_{\text{Generator}} &= \frac{d_t+\Delta\mu_t+(1-\Delta\mu_{\max})}{2} && \text{| Loss}
\end{aligned}
\tag{7.3}
$$

Equation 7.3: Generative Model: Discriminator Metrics

All metrics are within the range of $(0, 1)$ and have the following interpretive power:

- Real-Mean: Average performance of the discriminator on classifying real data
- Fake-Mean: Average performance of the discriminator on classifying fake data
- Mean-Bias: Average bias of the discriminator on classifying any data

[3] The computations of the minimum, maximum, median and standard deviations are trivial and have been omitted. The mean was explicitly stated, as it forms the basis for computing the derived metrics.

- Mean-Distance:
  → 0: Real and Fake data appears identical to discriminator
  → 1: Perfect classification
- Max-Distance: Best discriminator performance so far
- Mean-CenterOffset: Overall confidence of the discriminator in its guesses
- Loss: A measure of the performance of the generator

The 'Mean-Distance' and 'Mean-CenterOffset' are equivalent if the discriminator is on average correctly classifying both types of data correctly ($\mu_{\text{Real}} > 0.5$ & $\mu_{\text{Fake}} < 0.5$). A divergence of these two metrics indicate a heavy overall bias towards one of the two classifications.

An example of the discriminator's performance over multiple training iterations, showcasing all metrics presented above, is presented in Figure 7.16. The green and red regions indicate the regions between the minimum and maximum of the respective data; as the discriminator output is always initialized at 0.5 these regions will usually border or incorporate the 0.5 line.

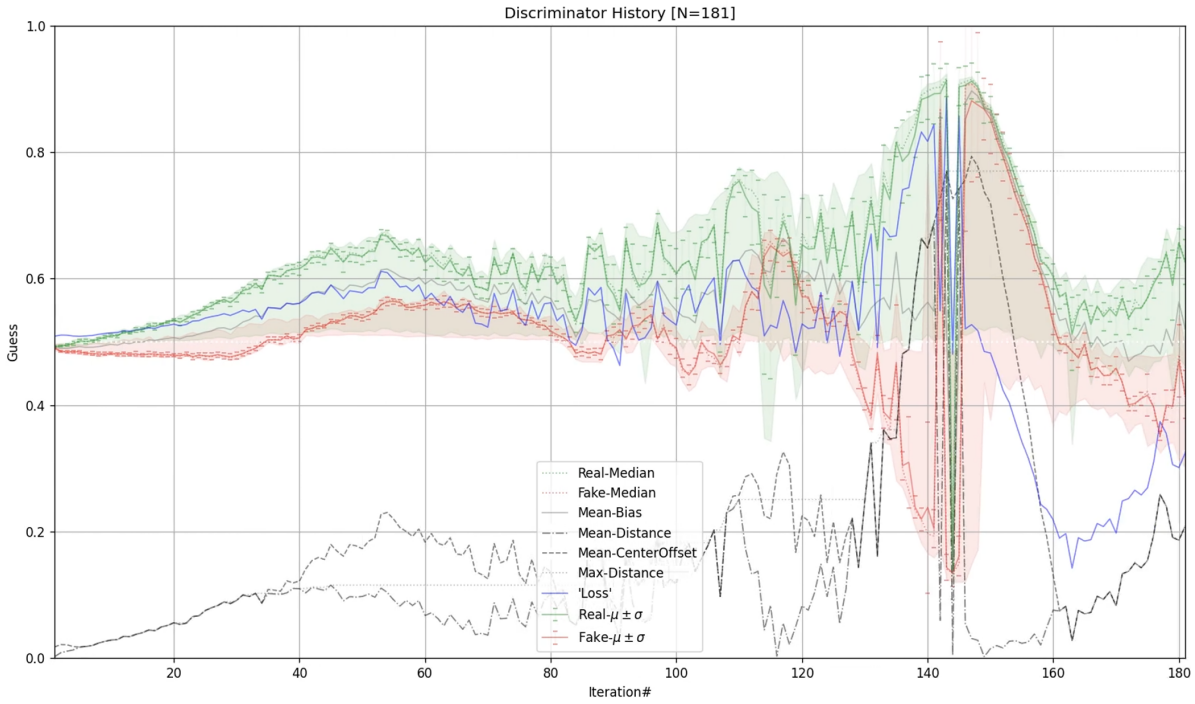


Figure 7.16: DRA-GAN: Discriminator Metrics Example

It becomes apparent that by incorporating the $1-\Delta\mu_{\max}$ term into the loss function, the loss function starts at ≈ 0.5 and can only drops lower once the discriminator has showcased an adequate level of performance. The loss first has to rise before it is able to drop lower than the starting value. This solves the problem of the numerical equality of the start and desired end of the training.

Combined Loss

The final 'loss' is a combination of the generator loss described above and the normalized average pain losses. The normalized pain loss indicates the ratio of exceeding the joint limits, so any value below 1 is acceptable.

It is defined by Equation 7.4.

$$\text{Loss}_{\text{Total}} = \text{Loss}_{\text{Generator}} + \text{Loss}_{\text{Pain}}^{\frac{1}{r}} \tag{7.4}$$

Equation 7.4: Generative Model: Advanced Loss Function

Given that the generator loss is in range $(0, 1)$ and the pain loss should never exceed 1, it becomes clear that the total loss should never exceed a value of 2. Any value < 1 is likely to indicate acceptable limits and adequate performance of the generator.

Unfortunately, neither the generator nor the pain loss have any true expressive power over how good the current output is. It provides merely an indication on the system's performance and the knowledge that any values > 2 indicates unacceptable generator performance.

It is important to note that this loss metric is not directly used for training the network, but indirectly to increase insight and as the metric used by the early break criterion and the BHPO optimization.

7.5. Results

The DRA-GAN framework is capable of successful execution. Multiple model configurations have been trained over the course of this research.

7.5.1. Model Development

As indicated in Section 7.2, the initial design did not show any training progression at all and hence produced no results.

The initial design problems were solved one problem at a time through iterative design, as outlined in Section 3.3.2. Therefore, the initial result was the successful design of GAN for motion generation.

The following results were achieved:

1. Increased insight into the training process:
   (see 7.4.3 [Discriminator Metrics], 7.4.3 [Combined Loss], 7.3.5 [Soft-Self-Attention] & 7.4.2 [Weights and Biases])
2. Resolved training stagnation:
   (see 7.3.2 [Fully-Connected Final Layer] & 7.3.1 [Stacked RNN Units])
3. Avoided exceeding of physical limits:
   (see 7.3.6 [Pain] & 4.4.1 [Principal Component Analysis])
4. Removed Euler angle reconstruction error:
   (see 4.4.1 [6D-Rotations])
5. Prevented trivial solution:
   (see 7.3.7 [Limited Judgement])

Only after resolving these primary issues could the quality of the generated motion be assessed.

7.5.2. Visual Output

Given the lack of an objective numerical metric to assess the final output motion, the final measure of judgement is left to visual inspection of the generated motions by the observer. Figure 7.19 attempts to visualize one such motion, by displaying key frames over time sequentially along a spacial axis. This is at the cost of losing information on the spacial translation, as the temporal displacement is replaced by a spacial one. While far from perfect, this initial motion does appear to have a certain resemblance to an 'arabesque' in ballet, after which it progresses to a strange contorted pose.

In the end the motion settles into a final pose, presented in Figure 7.17, after which it still displays motion of $\approx 30 - 50$ cm and movement of the limbs of a few degrees, but overall holding the same pose. At first glance the pose looks unnatural and strange, but unlike earlier in the training this pose actually resembles a human figure. For comparison, Figure 7.18 shows the exact same output frame in reaction to the input pose, but at three earlier stages in the training. The stark contrast between these poses show that the network has indeed learned to output more realistic poses and motions over time.

Analyzing the final pose of the network the following aspects can be observed:

- Pro:
  + Character is upright
  + Almost all joints are within natural limits
  + Foot contact is made with the floor ($\pm 2 - 3$ cm)
  + COG expected to be above foot base, in the sagittal (sideways) plane
  + Pose looks 'active' (hands up)
  + The pose is located near the origin, where the original dancers have been recorded
- Con:
  - Right leg is twisted unnaturally

- Spine is slightly bend, in the coronal (frontal) plane
- Left leg makes contact on the wrong side of the body

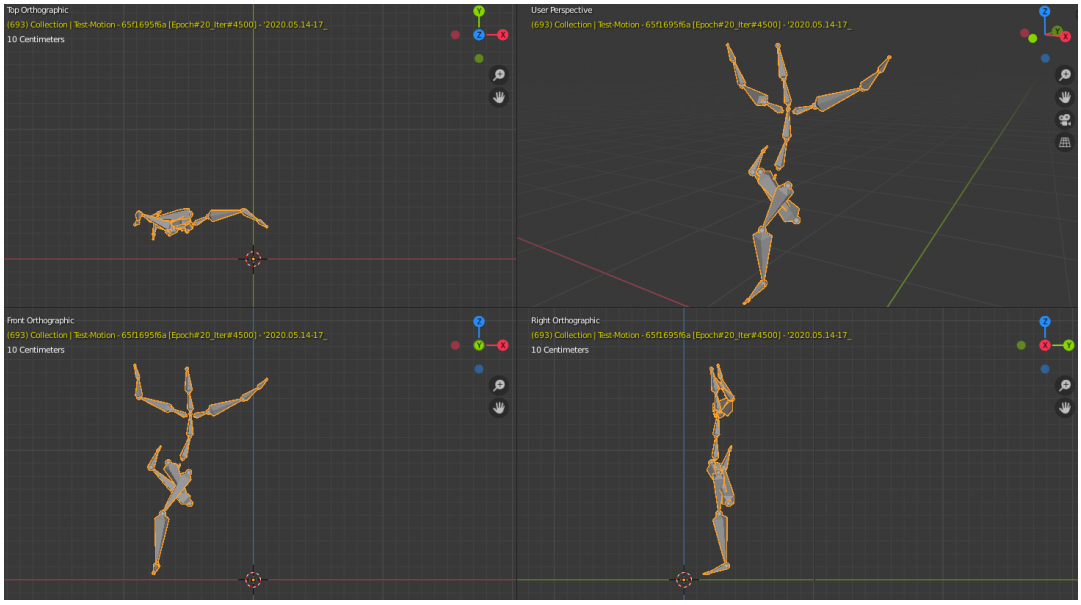


Figure 7.17: DRA-GAN Results: Final Pose [Epoch #20]
Transformation: Rotated 40° along the global upward axis, to align with the principle planes for better visualization.

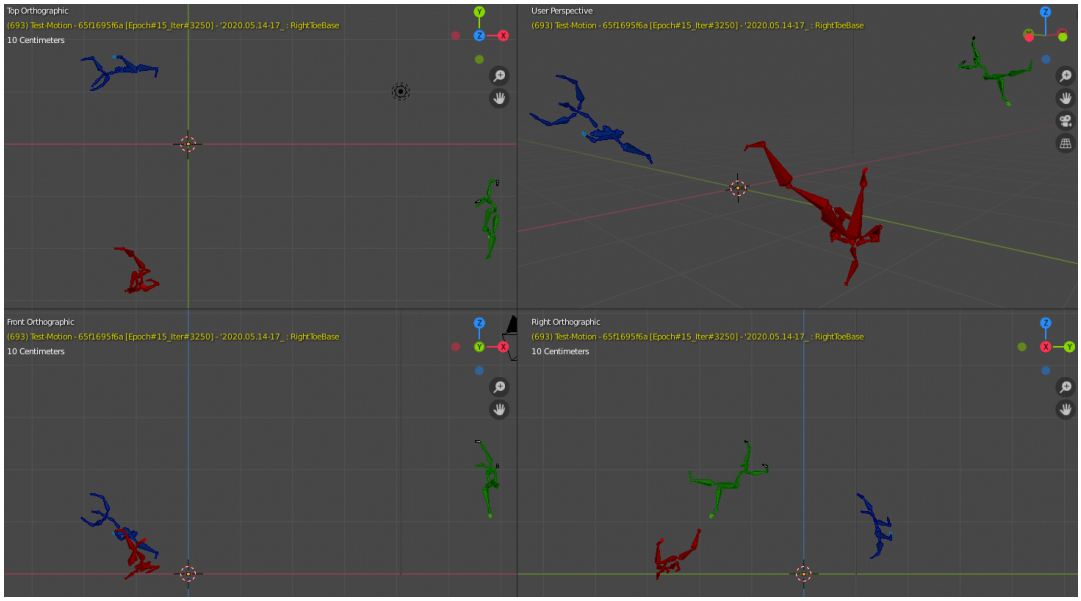


Figure 7.18: DRA-GAN Results: Twisted Poses [Epoch #{5,10&15}]
Red: Epoch #5 | Green: Epoch #10 | Blue: Epoch #15

Overall, given the limited number of hyperparameter sets the model was able to be tested with, it is impossible to say conclusively that the design model is strictly unable to generate the desired results and further BHPO should be performed.

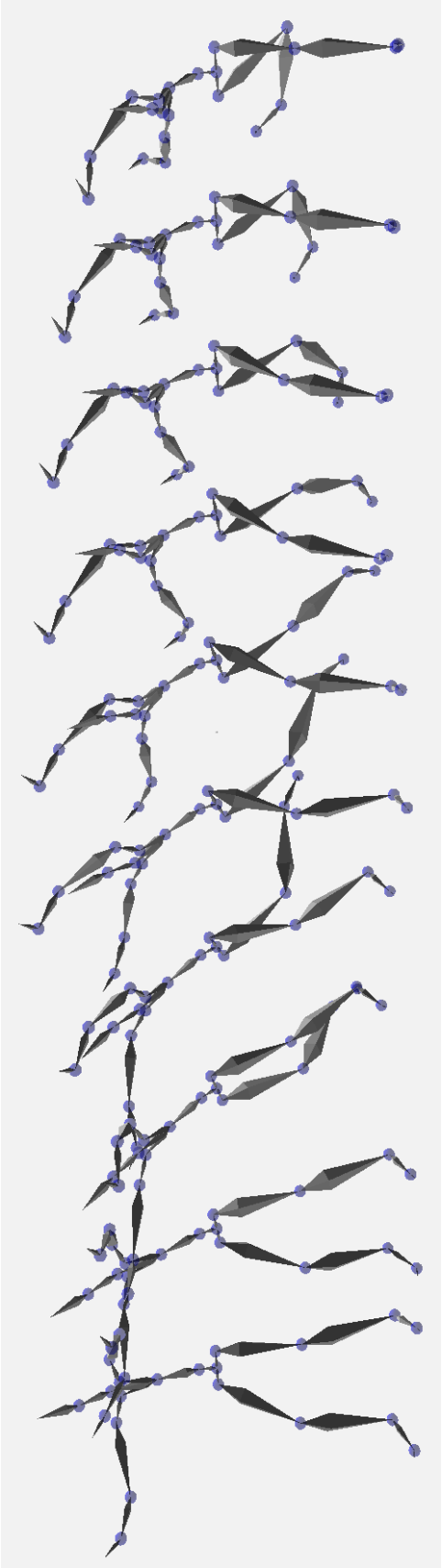


Figure 7.19: DRA-GAN Results: Output Motion Example

7.5.3. Training Progression

The primary reason why only a singular pose is presented as the final pose is the fact that, the output motions are almost completely identical, independent on the input motion selected. The issue of a GAN generating largely the same output, independent of the input data provided, is commonly known as 'mode collapse' and one of the ways in which the GAN's training instability can manifest. The general output pose is almost the same, with only different translational motions and slightly different motions in the joints. This is because the current state of the network largely ignores the leading motion.

This can be seen by the fact that the attention of the discriminator is almost entirely focused on the output motion and does not take the input motion into consideration (see Section 7.3.5).

At first this may seem as an undesirable situation, but is actually a logical conclusion of the GAN based framework. To understand this, one needs to consider the training progression that is to take place:

1. Both networks are initialized randomly
2. The discriminator realizes that the input motion has no discriminatory features, as it is identical independent on whether the output motion is real or fake
3. The discriminator starts to ignore the input motion and focuses on the output motion
4. The generator starts learning based on the generator's feedback on the output pose
   ↑ Current state of the network ↑
   ↓ Desired state of the network ↓
5. The generator will learn to generate output poses that are realistic and within the training distribution
6. The discriminator looses the output pose as a discriminatory feature, as the output poses are indistinguishable of the real poses
7. The discriminator will have to look for a new discriminatory feature:
   - The outputs motion → Δ Output Pose
   - Judge the output pose in context → Input Pose
   - Judge the output motion in context → Δ Input Pose
8. The generator will continue to learn based on the new feedback and improve the overall output piecewise until perfection
9. *Jump to 7. and repeat*

Therefore the current training progression is not incorrect, it just appears to halt after step 4. One reason for this might be the weight decay applied to the attention weights. Usually the attention weights are supposed to be a feature that filter out undesirable data that is not required for learning, but these weights do not change. In the case of a GAN the data that should be focused on changes over time and in the end all data should be taken into consideration. In the current state the attention weights become so small however, that little to no information is passed through anymore. This makes it exceedingly hard to recover the attention on this part of the data.

Intuitively, one might think of it like the self reinforcing suggestion algorithms on modern Internet platforms: The user is only shown new suggestions based on his or her preferences, therefore the user consumes more of this suggested content. Other content is still available, but it becomes exceedingly hard to access and to change the attention to it.

Therefore, a means of recovering attention over time could be a potential solution to this problem.

7.5.4. Extended Results

Multiple runs have been executed, but so far none have generated true long-term interactive motions. For a few selected runs the results have been compiled in Appendix E. For each run the following parameters and results are shown:

- Elements used (for reproducibility):
  - Network Version (see Section 3.3.2)
  - Hyperparameters (see Section 5.1.3 & Section 7.4.1)
- Output generated:
  - Final Network Weights (see Figure 7.14)
  - Discriminator History (see Section 7.4.3)
  - Pain History (see Section 7.3.6)

The following runs are showcased:

1. V1.0-23 | 6D-ATT-DI-DO-LSTM-GAN | e406d70f24
2. V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | d9be67f6a7
3. V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | 42241955b0
4. V1.2 | 6D-PCA-ATT-DI-DO-LSTM-Pain-GAN | 96142329a6
5. V1.3-7 | 6D-PCA-ATT-DI-DO-LSTM-Pain-LJ-GAN | 7d2d6588e0
6. V1.3-15 | 6D-PCA-ATT-DI-DO-GRU-Pain-LJ-GAN | 89ba1481a7

DanceNet models prior to V0.9 did not contain the complete derivative handling and are hence not classified as a DRA-GAN. Runs prior to *DanceNet* V1.0 did not save the plots to disk, but only had live visualization inside the BHPO GUI (see Appendix C). The selected runs show the progression of the model over the various version, as well as the diversity in the resulting output, based on the various hyperparameter sets used. Each result contains a section with a brief analysis that highlights some of the important elements to note.

III

Application & Relevance

8 Live Interaction

The focus of Part II lies in fulfilling the internal goal of this research, through data handling and the development of the frameworks for training neural networks. This chapter will focus on fulfilling the external goal of this research, through the development of a framework for interacting with the pre-trained neural networks.

8.1. Live Simulation

The final network is intended to be used in a live performance on a theater stage. The scope of this research was restricted to a simulated environment only, as the budget did not suffice for testing with the real drone swarm. As a result, a live simulation environment was created, as close to the final product as possible.

The final system has to fulfill a few key requirements:

- Update rate: $\geq$ 100 Hz (same as *OptiTrack* FPS)
- Latency: $\leq$ 10 ms (As low as possible)
- Communication: Publish results for further processing by other algorithms
- Safety: No harm to the audience or dancer

All of these requirements were adhered to in the designed simulation, except the final requirement on safety. It was not taken into account as inside the simulation the drone swarm poses no threat to the user and because the network is restricted to generating novel motion poses, with the drone swarm control system required to handle collision avoidance.

The two main elements of the framework are the model inference module, enabling runtime interaction with the pre-trained neural network, and the *OptiTrack* streaming module, receiving live data from the *OptiTrack* motion capture (MoCap) system.

8.1.1. Model Inference

Once the model is trained, using the frameworks presented in Chapter 5, a lot of framework elements are not required anymore at inference time. For the generative model only the generator is required, with the discriminator only being a means to an end during training.

To utilize the final model, only the trained network parameters (<.pt> files) and the hyperparameters ('`Hyperparameters.json`') related to the generator's design, are required.

A simulation environment was developed to execute the trained model at runtime. The resulting model is shown in Figure 8.1. Adapting the training model (see Figure 7.1), the generator was placed in a separate environment. After selecting a data source for the input motion, the motion is processed sequentially and the results are published as UDP (User Datagram Protocol) messages onto the LAN (local area network).

Data Input Sources

Three basic input sources were defined for testing the runtime speed of the inference model. The modes and their measured update rates are presented in Table 8.1.

[1]Run on the following hardware:: CPU - *Intel i7-8750H* & GPU - *NVIDIA Quadro P1000*

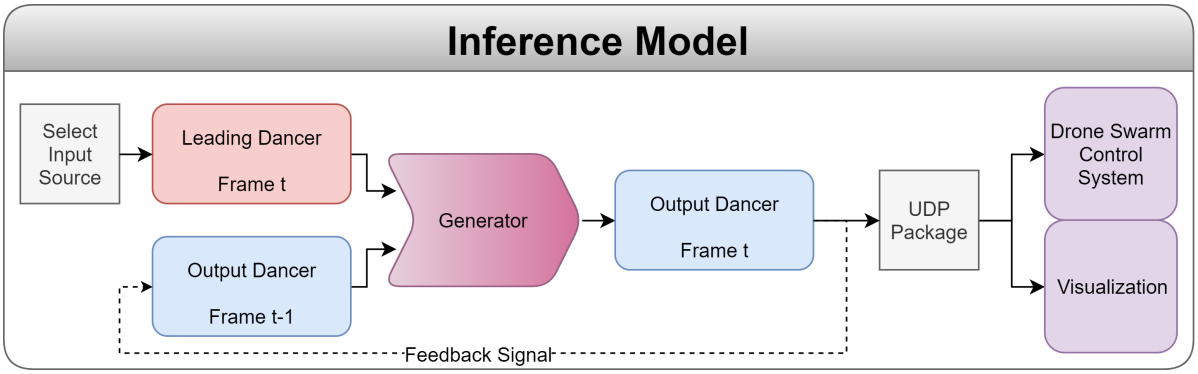


Figure 8.1: *DanceNet* Architecture: Inference Model

Live Simulation Speed Test[1]		
Input Source	**Update Rate**	**Description**
T-Pose	$\approx 500\ [Hz]$ ($\approx 2\ [ms]$)	Continuous T-Pose
Random Pose	$\approx 500\ [Hz]$ ($\approx 2\ [ms]$)	Completely randomized input pose
BVH Playback	$\approx 333\ [Hz]$ ($\approx 3\ [ms]$)	Frame-by-frame playback of <.BVH> file

Table 8.1: Live Simulation: Modes and Speed Test

The test clearly shows that the model is capable of fulfilling the update rate and latency requirements, even on mobile hardware.

The 'T-Pose' and 'Random Pose' input modes are not only intended as a quick speed test tool, but also to see the network's reaction to these two modes of operation. For the safety requirement the network has to behave at least somewhat predictable. Therefore, at least the following two tests should be executed before utilization on stage; The output shall...:

- T-Pose: Start and end of each performance
  → ... remain still when the input motion is still for long enough.
- Random Pose: Simulating loss of tracking or system malfunction
  → ... continue to move in a physically realistic manner and not move out of control.

Given the current state of the system, resulting in an almost static pose, these two requirements are met but not representative.

Communication

The choice was made to transmit the resulting data via UDP to the LAN, as this allows for executing *DanceNet* as a standalone service in a microservices architecture.

UDP vs TCP: The choice was made to utilize UDP (User Datagram Protocol) over TCP (Transmission Control Protocol), as it has the following benefits and resulting capabilities:

- Multicast: → Send data to multiple or all machines and/or services at once
- No handshake: → No timeout and low latency

The problem with TCP is that it requires a three-way handshake, in which the recipient acknowledges the connection. This results in additional network overhead and the necessity to know the IP of the recipient ahead of time.

UDP allows to send-and-forget messages, allowing the recipient to process the incoming packets at their own pace, only using the latest incoming message. The downside of UDP is that some data packets might get lost or be dropped by the recipient, but this is actually intended behavior for real-time inference, as older messages are by definition obsolete once a newer message has arrived.

UDP Message components			
Component	**Data Type**	**Description**	**SI Unit**
`UnixT`	float	Unix Time of sending	$[s]$
`InputMotion`	float $[3 + 63]$	Live/playback/generated motion	$[m]$ & $[^\circ]$
`OutputMotion`	float $[3 + 63]$	DanceNet generated motion	$[m]$ & $[^\circ]$
(**`ReferenceMotion`**)	float $[3 + 63]$	Recorded reference motion	$[m]$ & $[^\circ]$

Table 8.2: Live Simulation: UDP Message components

Message content: Each data message contains the information shown in Table 8.2. It contains at least the time, to check the latency of each message, and the original and generated skeletal poses. When running the system in 'BVH Playback' mode, the 'ReferenceMotion' can be send as well for visualization as a reference.

Each pose is encoded in SI units, using the Euler angle representation. The message is encoded in a JSON inspired key-value format, with a new line for each component: `<key>=<value_string>`

For production this could be fully changed to a JSON based transmission, as this should make parsing on the receiving end easier.

8.1.2. *OptiTrack* Streaming

To enable utilization of the system on stage, the *DanceNet* module is required to receive live data from the *OptiTrack* MoCap system. *OptiTrack* streams the live MoCap data via their *NatNet* SDK. The serialized UDP packets need to be decoded prior to passing the data into the *DanceNet*.

Each NatNet message contains information on all markers, bodies and skeletons present in the tracking volume, but for this module only the skeletal information is of relevance. Unfortunately the skeletal encoding and joint order are not equal to the convention used by the <.BVH> files and has to be converted. Each skeleton is encoded as relative or absolute rotations in quaternion format.

A pre-processing tool[2] was created that decodes the *NatNet* packets and converts them into the appropriate format for further processing. The steps taken to receive and process the data are presented in Algorithm 8.1.

Algorithm 8.1 Live Simulation: *OptiTrack* Message Decoding

1. **Initialization:**
   - (I) **Setup client and connection to server**
   - (II) **Request asset description information**
   - (III) **Save asset description information**
2. **Receive and decode packets:**
   - (I) **Receive packets via *NatNet* SDK** (see Optitrack.com/software/NatNet-SDK)
   - (II) **Decode serialized data**
   - (III) **Extract skeletal data from full information**
   - (IV) **Reorder bones into BVH hierarchy** (see 4.2.2 [Biovision Hierarchy (BVH)])
   - (V) **Remove finger bones** (see 4.3.2 [Skeletal Simplification])
   - (VI) **Convert from quaternions to Euler angles**
   - (VII) **Forward data to *DanceNet***

The system is configured using a multithreaded asynchronous callback structure, for increased performance and to ensure handling of all incoming messages. The system was tested using the streaming settings for *Motive* as defined in Figure 8.2 and the following version for software provided by *OptiTrack*:

- *Motive* 2.2.0
- *NatNet* 3.1.0.0

[2] Build upon the '**`PythonClient`**' example provided by the *NatNet* SDK.

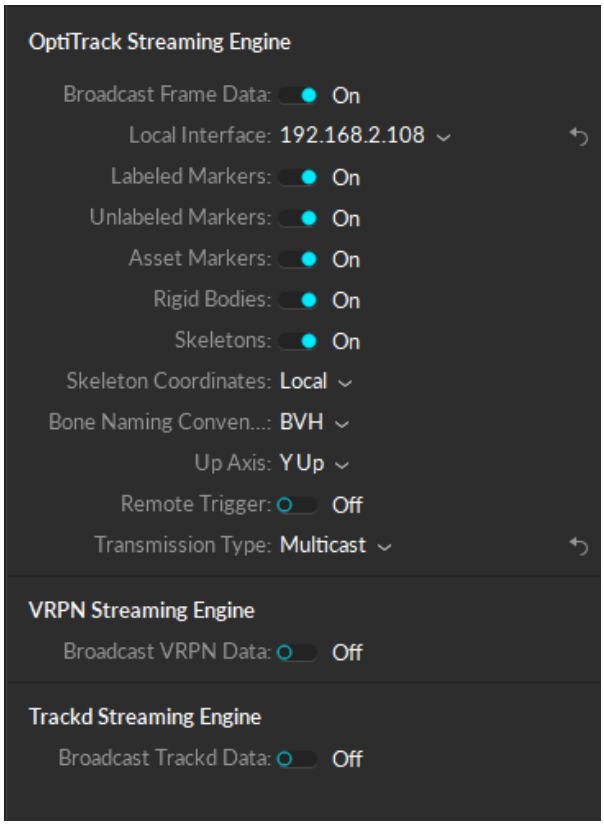


Figure 8.2: *OptiTrack*: *Motive* Streaming Settings

Right Leg Bug

The pre-processing tool works as intended, except for a singular unexplained bug. For unknown reasons the right leg is reconstructed incorrectly, even though all joints are processed by exactly the same code and the remaining joints are reconstructed without error. This phenomenon is presented in Figure 8.3.

It is uncertain if the bug is introduced in the code written by the researcher, or in the Python example provided by *OptiTrack*. Even after multiple calls with the *OptiTrack* customer support the issue remains unresolved. This bug needs to be resolved before utilization on stage.

Speed

The system was initially used over the local WLAN, but conflicts with other devices in the network caused a significant number of packets to be dropped, a highly irregular frame rate and sometimes latencies of > 2000 ms.

After the switch to a separate LAN configuration was made, the system ran at a steady frame rate of ≈ 50 FPS. The tool has the option to save the raw in- and output motions to disk in <.BVH> motion, for verification of the received frames. This however costs additional time and slows down the system. Turning the 'SaveToDisk' option off results in a frame rate of ≈ 80 FPS.

Finally, removing the finger joints right after receiving the *NatNet* message, instead of after conversion, resulted in a the desired update rate of 100 FPS. Receiving and processing almost all messages, with a negligible amount of frames dropped ($< 0.1\%$).

8.1.3. *AI-man* Framework

The *DanceNet* module, including the *OptiTrack* pre-processing module, are only part of a larger framework that will facilitate the final *AI-man* show. The simplified version of the entire live data processing framework for *AI-man*, including the feedback signal from the drone swarm back to the Dancer, is presented in Figure 8.4.

True Feedback

As indicated in Figure 8.4, the final version of this system will include a feedback signal from the drone swarm back to the dancer. Independently on whether the system is executed inside a simulated environment or

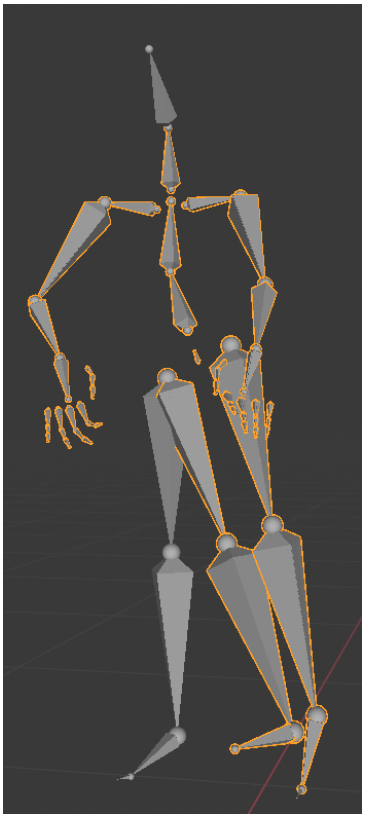


Figure 8.3: Live Simulation: *OptiTrack* Reconstruction Bug
Grey outline: Original pose & Orange outline: Reconstructed pose

utilizing the real drone swarm, as soon as the dancer receives visual feedback from the system, the dancer will react to the output generated by the system.

This is important as this results in a dynamic that the network is not trained for, as this would reach over into the realm of reinforcement learning (RL).

The full dataset is split into the training, as well as the two test and validation datasets, to perform verification of the *DanceNet* module on previously unseen data. These datasets however are static and do not properly represent the real life scenario where both partners react to each other. As they are pre-recorded, the leading motion will never change based on the output motion.

Therefore, a final live test with a dancer is required to validate the system's performance in the desired environment.

This test was originally planned to be performed at *AKOB*'s studio, but was canceled due to the unfortunate lack of interactivity of the current system's output (see Section 7.5.2 [Visual Output]).

8.2. Live Visualization

As this research did not include the means of testing the system with the real drone swarm, two visualization environments were designed for this research.

8.2.1. *VPython*

For aiding in debugging the *DanceNet* framework and the live simulation module, a live visualization environment was created in *VPython*. The environment can be executed independently of the *DanceNet* module, as it receives and processes the UDP messages from the live simulation asynchronously.

The code runs at $\approx$ 60 Hz, but the *VPython* framework's actual frame rate displayed on screen is unfortunately much lower ($<$ 10 FPS). The *VPython* module is a very practical tool for fast visualization and visual debugging of 3D data, but its runtime performance is suboptimal and highly unpredictable at times.

The following figures in this report were made utilizing the *VPython* visualization environment:

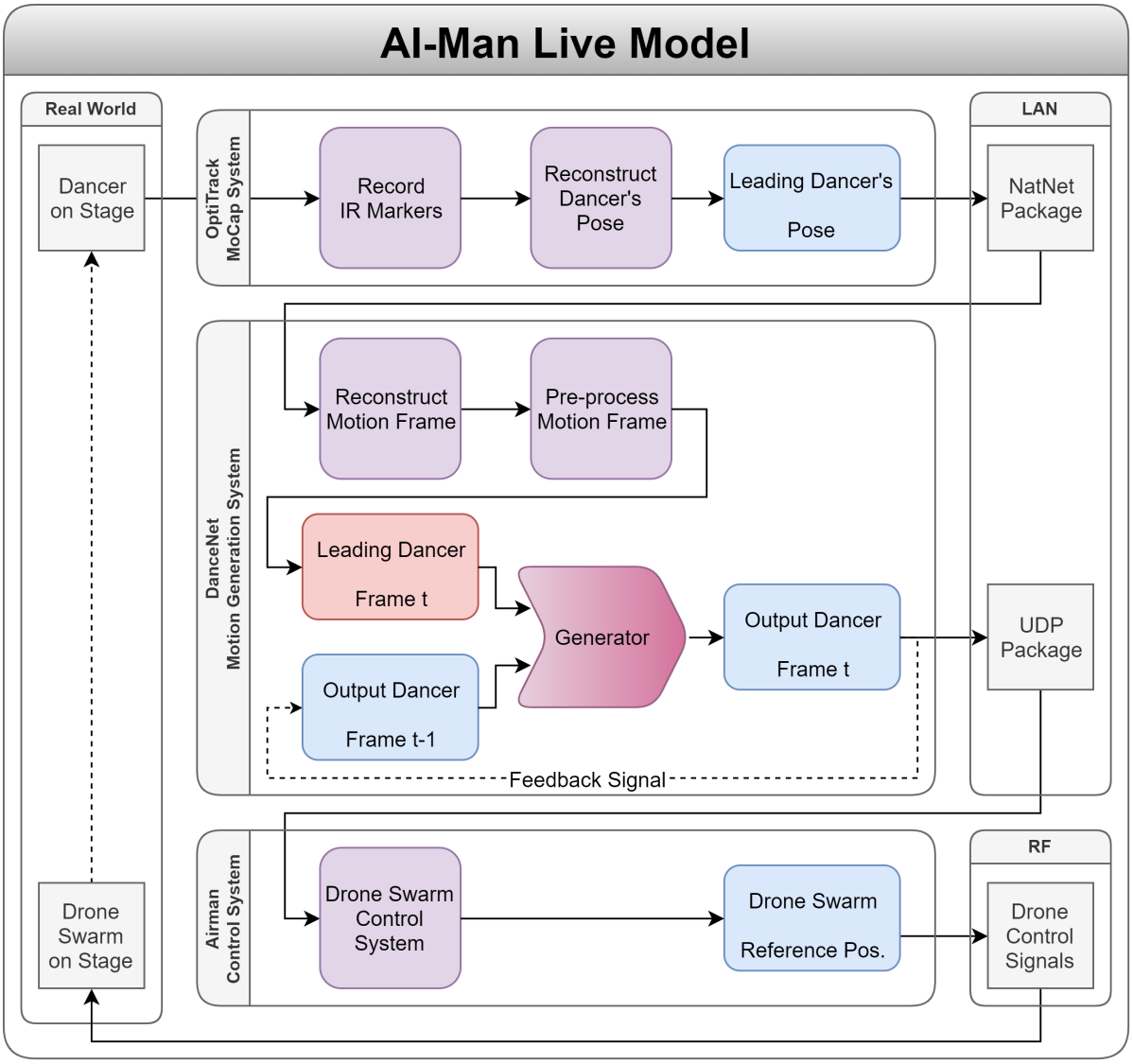


Figure 8.4: *DanceNet* Architecture: Live Model

- 3.1 [Skeletal rig used by *AKOB*]
- 6.4 [*DanceNet* Results: Regression Model]
- 6.10 [*DanceNet* Results: Regression Model with Feedback]
- 7.19 [DRA-GAN Results: Output Motion Example]

8.2.2. Unreal Engine

Given the apparent performance limitations of the *VPython* visualization environment, a new visualization environment had to be developed to facilitate live testing. As no real drones were to be used for the initial system tests, it was considered to use a VR environment to increase the immersion of the dancer, when interacting with his artificial counterpart.

To this end, the 'Unreal Engine 4' (UE4) game engine was chosen for the following reasons:

- High performance — > 100 FPS
- VR compatibility
- C++ based — Ability to interface with custom modules for data handling
- Customizable meshes and textures
- Readily available *OptiTrack* plugin [3]

An example of the resulting visualization can be seen in Figure 8.5.

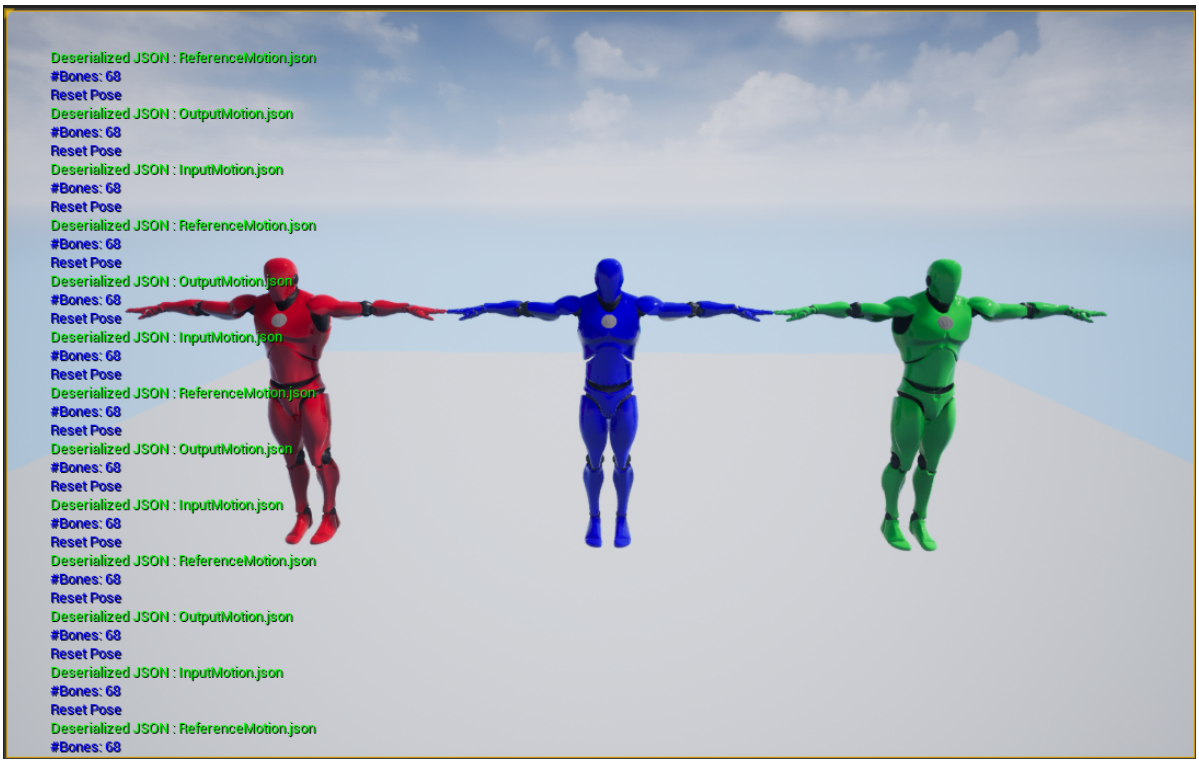


Figure 8.5: Live Visualization: Unreal Engine T-Pose Example

Rotation Conversion Bug

The default *OptiTrack* plugin allows for streaming the recorded skeletons live into the game engine. This is not applicable for the pose generated by the *DanceNet* module, as it does not match the message format used by *NatNet*. Unfortunately, the default character model configuration, as defined in UE4, is different to both the BVH and *OptiTrack* configurations. Therefore, the default character rig could not be directly interfaced and a custom module was created for posing the output. Due to the multitude of various coordinate systems used in the various environments (see Appendix A), combined with the fact that an all zero pose for the UE4 mannequin does not equal a T-Pose, transforming the DanceNet output to the UE4 rig correctly has proven non-trivial.

A representation of the output shown inside the UE4 is given in Figure 8.6. As an initial test, the motion was extracted from a JSON file and saved to disk by the live simulation module. At the current stage in development the live visualization is successfully parsing the motion and transferring it onto the rig, but the reconstructed pose does not match the original, due to each bone having a separate default rotation that needs to be corrected for.

The final visual results generated by *DanceNet*, at the current stage of the research, did not justify the additional resources spend on developing the advanced visualization. Therefore, the development was put on hold until the visual results generated are advanced enough to justify live testing with dancers.

A potential solution to this conundrum would be to perform a man-in-the-middle attack on the NatNet streaming data, injecting custom packets that include the generated motions. This approach is described in Algorithm 8.2. The downside of this approach is that the NatNet packets are encoded in a binary representation, making it harder to re-encode a valid packet. The benefit of this approach would be that both the original *Airman* drone control system and the UE4 *OptiTrack* plugin could directly parse the new packets, without requiring any additional code.

[3] wiki.optitrack.com/index.php?title=OptiTrack_Unreal_Engine_4_Plugin

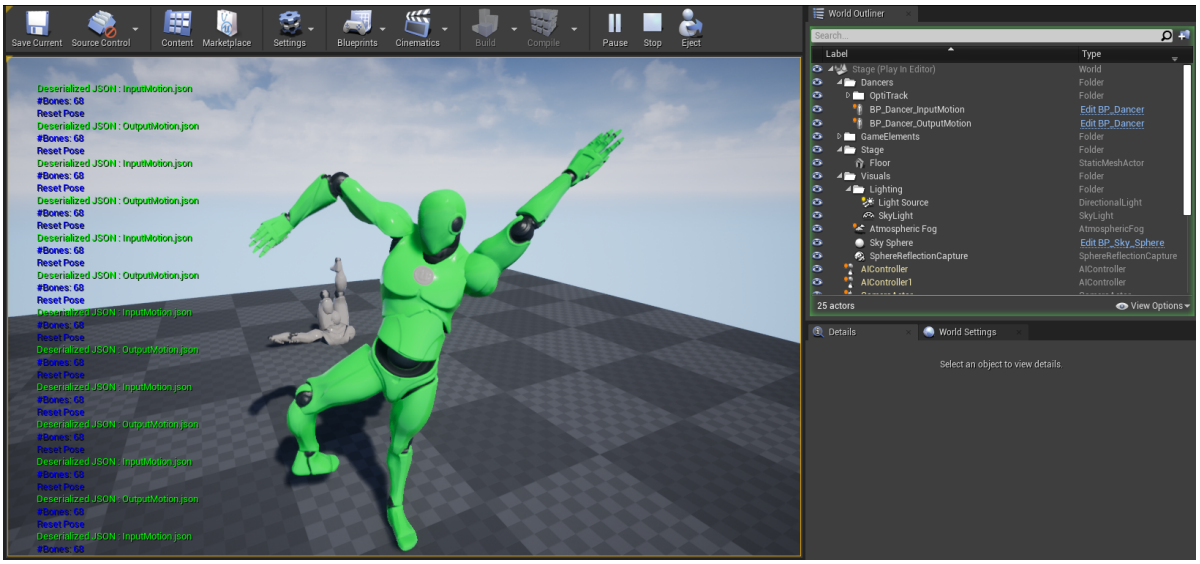


Figure 8.6: Live Visualization: Unreal Engine Rotations Bug

Algorithm 8.2 Live Simulation: *OptiTrack* Packet Injection

1. Receive the original *NatNet* packets
2. Decode the packets
3. Run the live simulation module
4. Convert the output motion to *OptiTrack*'s skeletal representation
5. Inject the new skeleton into the original data
6. Re-encode the *NatNet* packets
7. Forward the modified packets to the original recipients

9 Future Work

The current state of the *DanceNet* 'Differential Recurrent Attention GAN' (DRA-GAN) does not generate the desired results, but there are plenty of unexplored options to potentially improve the system. This section will list and briefly describe all potential improvements that were envisioned throughout this development, but were not executed due to a lack of time.

The potential improvements are categorized by the components of the model they would affect:

- 9.1 [Data]
- 9.2 [Architecture]
- 9.3 [Training]
- 9.4 [Optimization]
- 9.5 [Validation]

Some suggestions are very concrete and can be directly applied, while others indicate a more general approach that might benefit the system. May these serve as an aid to future researchers, to improve the current model and achieve true long-term interactive reactionary motion generation.

9.1. Data

9.1.1. Data Pre-processing

Data Augmentation

Even though the collected dataset is the second largest original publicly available motion capture (MoCap) dataset to date (see 4.5 [Dataset Overview]), complex models require vast amounts of data. Data augmentation allows to generate new data from the original dataset, by applying transformations to the data that do not change the nature of the data with respect to the desired task to be learned.

For this dataset the following two parameter sets can be augmented to expand the available dataset:

- Position in the horizontal plane:
  → Applying an equal translation to both dancers' root positions, bound by the tracking volume
- Yaw Angle:
  → Rotate both dancers by an equal arbitrary angle, around the origins upward axis

As long as the relative distance and angle between both dancers remain the same, many variations of the same motion can be generated.

Extended Chunk Splitting

Currently, each take is split into chunks of equal length, throwing away the end of the take, which does not fit into an entire chunk. This can be avoided and the number of chunks greatly extended, by not cutting each take into 'floor $\left(\frac{\#\text{Frames}}{\text{ChunkLength}}\right)$', but into '#Frames − ChunkLength' instead. This extracts every possible chunk, by shifting the chunk backwards one frame at a time.

Data Shuffling

Currently, the network is trained by passing every chunk of the training dataset in chronological order. This is intended to mimic the exploratory process exhibited by the dancers originally recording the dataset.

However, when utilizing the data augmentation and chunk splitting technique described above, this results in the network processing a large amount of data with only slight variations, before moving on to the next take. This might lead to overfitting on these takes. To avoid this the chunks should be shuffled randomly during the training procedure.

9.1.2. Data Input

Smart Hidden Vector Initialization

Currently, the output vector is initialized as a zero vector at the start of each training iteration (see 7.3.7 [Hidden State Initialization]). This results in the output pose being initialized as either the T-Pose or the 'Principal Component Analysis' (PCA) mean pose, depending on whether or not the PCA transform is applied.

To potentially avoid the trivial solution, which the limited judgment modification attempts to fix (see 7.3.7 [Limited Judgement]), the hidden vectors for each batch should be initialized equal to the first frame of the following dancer's recorded motion.

The downside of this approach is that at inference time there exists no leading pose to initialize in and a default pose, such as the original zero vector, will have to be selected. This however causes a discrepancy again between the training task and the final task to be performed.

BDF Input

The 'Derivative Input' (DI) modification pre-computes the first order derivative approximation, equivalent to the backward Euler method (see 7.3.4 [Derivative Input]). The 'Backward Differentiation Formulas' (BDFs) can be extended to higher orders, to increase the accuracy of the derivative estimate, while only utilizing past data [Iserles, 2008].

The DI modification could be extended to pre-compute the BDF of the Xth order, whereby 'X' indicates a new hyperparameter of the system. This would remove the necessity for the system to learn this metric, allowing for the system to concentrate more on the desired creative reactionary response. This however comes at the cost of extended memory and computational requirements, as the system output needs to be retained over X frames. Given the current inference time of ≈ 2 ms of the *DanceNet* network, it can be safely assumed that this change should not cause the system to break the > 100 FPS requirement.

However, caution needs to be applied as the stability of this method needs to be checked against the size of the time step between frames. A quick experiment was performed approximating the derivative of a simple sinusoid with a frequency of 10 Hz, with the BDFs up to the 6th order. This was done using the same time step, of 10 ms, as used by current dataset. The results are shown in Figures 9.1[1] and 9.2, showing a clear exponential improvement for each additional BDF order[2].

Figure 9.1: BDF: Approximation Errors over Time

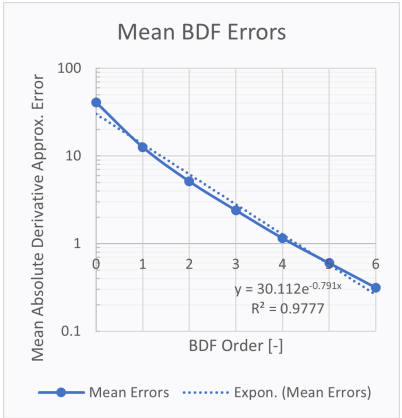


Figure 9.2: BDF: Mean Approximation Errors

[2]Interesting phenomenon in this example is that the peak of a lower order approximation is always equal to the approximation of the order above it (Mean error $4.3 \cdot 10^{-13}$)

[2]The 0th order approximation was defined as $\dot{Y} = 0$.

Noisy Labels

As described in 7.1.4 [Best Practices], applying an element of stochastic noise to the real and fake labels might improve the training procedure. This entails adding a one sided stochastic variable to each originally binary label at runtime, to generate a value between the original and 0.5. One potential realization of this procedure is given in Equation 9.1. This introduces a new hyperparameter (σ), defining the standard deviation of the applied Gaussian noise.

$$\begin{aligned} g_{\text{act}_{\text{Fake}}} &= 0 \quad + \min\left(0.5, \left|\mathcal{N}\left(0, \sigma^2\right)\right|\right) \\ g_{\text{act}_{\text{Real}}} &= 1 \quad - \min\left(0.5, \left|\mathcal{N}\left(0, \sigma^2\right)\right|\right) \end{aligned} \tag{9.1}$$

Equation 9.1: Noisy Labels

Alternatively, instead of using a stochastic modifier for the real labels, the original take could be analyzed and a metric defined to summarize how 'real' the motion is; e.g. applying a penalty for each detected outlier in the original data.

Noise Input

Conventional 'Generative Adversarial Networks' (GANs) utilize random noise as the primary input to the generator, based on which the generator is intended to generate an entirely novel image. Currently, given the task at hand the system uses the leading dancer's motion as input instead of random noise. Potentially concatenating the current generator's input vector with an entirely random input vector may aid in generating variational output motions, given the same leading motion. This is intended to encourage the system to generate various output motions in reaction to each input motion, as in dancing there is no singular correct motion.

9.2. Architecture

9.2.1. Current Model Improvements

Multiple Discriminators

Currently, the network is structure based on the basic GAN structure, using a singular generator and discriminator respectively. One of the problems of the current system is that the system gets stuck, focusing on a singular issue: the output motion.

Intuitively, a student would benefit from having multiple teachers that focus on different, but equally relevant, aspects of the students development.

The architecture replaces the necessity for the discriminator's attention mechanism, by utilizing an ensemble of multiple additional discriminators, each intended to focus on a different aspect of the data. In the current system this would add two smaller discriminator networks, each focusing solely on either the output pose or the output derivative, in addition to the general discriminator that can discriminate the full output in the context of the input motion. If the additional discriminator networks are given a higher learning rate than the general discriminator, this should lead to faster learning of generally realistic poses and motions. The lower relative learning rate of the general discriminator should aid in avoiding overfitting on a singular discriminative component, while the other two discriminators initially keep the general output in check.

The theoretical feasibility, as well as general implementation and benefit of this architecture have been explored by Tolstikhin et al. [2017].

Inject Past Models

As described in 7.1.4 [Best Practices], to avoid mode collapse it might be beneficial to keep the state of previous iterations during training and inject these back into the system. This can be seen similar to the approach of using multiple discriminators, as it temporarily introduces a different network (state) into the system.

Considering the desired process described in 7.5.3 [Training Progression], it is desired for the discriminator to focus on new discriminatory features to improve the level of detail of the final output. This process might also introduce an instability, however, in the form of an infinite loop: The discriminator focuses on feature A until the generator has learned it and then focuses on feature B. If the generator unlearns what it had already learned about feature A in favor of feature B the two players may be stuck in this cycle forever.

This cycle could be broken by injecting the discriminator's state of the previous stage in parallel to the current stage, forcing the generator to focus on both features simultaneously. Intuitively, this could be seen as a 'surprise test' that a teacher issues to its students, asking questions on topics learned some time ago, to refresh their memory.

This can be compared to the rationale behind the 'Unrolled Generative Adversarial Networks' architecture, which avoids mode collapse by means of backpropagation through multiple sequential GAN passes [Liu and Tuzel, 2016].

Residual Connections

Veit [2016] have proven that, by adding residual/skip connections into deep neural networks, it causes the network to behave as an ensemble network, improving the overall performance.This can be interpreted as lots of different smaller networks that are being trained simultaneously, sharing some weights with their 'peers'.

Two potential applications of this technique have been envisioned:

- Stacked RNNs with skip connections:
  Greatly increase the number of layers in the stacked RNN core, while introducing residual highway connections to keep the shortest path to 2-5 layers
- Temporal skip connections:
  Reintroducing the predicted outputs of N iterations ago back into the network, to facilitate learning true long-term dependencies

9.2.2. Alternative Network Models

Alternative Recurrent Units

Currently, two recurrent unit types can be applied in the DRA-GAN model: The LSTM and the GRU. There is however another modern and complex recurrent unit with promising results available: The 'Modified Highway Unit' (MHU), proposed by Tang et al. [2017], specifically for human motion prediction.

Phase Functioned Neural Network

The 'Phase Functioned Neural Network' (PFNN), as proposed by Holden et al. [2017a], is a network design that utilizes a separate set of network weights for multiple segments of the primary motion phase. This technique was primarily applied to locomotion, as a walk cycle has a singular clearly defined phase, but this concept breaks down when applying it to a general set of complex motions.

Two possibilities to utilize this methodology for the task at hand:

- Fourier Phase: Transform the motion into the Fourier domain and apply a separate phase for each discrete frequency
- Music Syncing: Use the current music's beats as the phase of the network

The first option would increase the number of input parameters, as well as the number of network parameters extensively, but might enable the system to achieve a feeling for timing and layer fast and slow motions.

The secondary option is not possible with the current state of the dataset, as the original music was not recorded in sync with the motion, but could be considered for future data acquisition. Music and timing is a key factor for dancers, introducing this element back into the network might greatly improve the reactionary behavior.

Sequential-Hierarchical Network

Currently, the system predicts the output motion holistically, by generating the complete new motion vector in one pass. Similar to the approach taken by Aksan et al. [2019], it could be considered to split the network's core into multiple smaller networks, each responsible for predicting the change to the next joint down the hierarchical chain. So instead of predicting the change to the entire arm at once, first the shoulder is predicted, after which the upper arm, utilizing the information of the shoulder joint.

The downside of this system is that each joint generates its prediction based on limited information, the benefit is that major changes are predicted first, which can be taken into consideration when generating the predictions of the next smaller joint.

Neural Turing Machine

The 'Neural Turing Machine' (NTM) is a novel 'Recurrent Neural Network (RNN) based model, proposed by Graves [2014], intended to mimic the read and write operations to memory that a more conventional logic based computer program utilizes to solve problems.

The architecture is novel and experimental and has not been applied to motion generation/prediction tasks. The ability to read and write to an explicit memory may aid in storing long-term information, such as intent and style of a motion sequence.

A promising variation of the NTM is the 'Differentiable Neural Computer' (DNC) [Graves et al., 2016].

9.2.3. Advanced Layers

Dropout

The dropout layer randomly sets certain values in the vector to zero, preventing information flow of that part of the input vector.

One of the problems encountered by the current system is the overreliance of the discriminator on a singular part of the input; the output pose. Introducing a dropout layer right after the attention layer may help to avoid this behavior.

Instead of applying the dropout to each value of the input vector directly, a higher level dropout layer could be applied that completely filters out one of the four major input vectors. Currently, this would mean that the discriminator is forced to sometimes make a prediction without access to the output pose, purely relying on the relation of the output derivative to the input motion.

Noisy Layers

Introducing noise into the system can be used as data augmentation, as well as to prevent overfitting. Currently, stochastic noise is added to the input of both the generator and the discriminator. However, this only partially satisfies the suggestion as given in 7.1.4 [Best Practices]. This approach could be extended to add a small amount of noise to every layer in the network, instead of just the input.

9.3. Training

9.3.1. Loss Functions

Normalized Power Spectrum Similarity

As previously mentioned it is hard to quantify how good or natural a human motion is. Gopalakrishnan et al. [2018] proposed the 'Normalized Power Spectrum Similarity' (NPSS) as a promising metric to solve this problem.

By transforming the motion into the Fourier domain and comparing the power spectra of real and generated motions a better assessment of the motions quality can be made. The current system could be improved by adding a loss component based on the NPSS to the system, similar to the pain loss introduced in 7.3.6 [Pain].

Label Classification

In 7.1.4 [Best Practices] it is suggested that simultaneous classification of data labels during the GAN training procedure might improve the training, as it requires the system to learn nuances in the data, instead of just the general distribution. The current labels of the dataset are not suited for classification due to the high bias towards a few frequently occurring labels (see 4.1.4 [Labels]).

This is something to take into consideration for future data acquisition, or retroactive relabeling of the data by *AKOB*.

Foot Contact Loss

A relatively common approach used in motion generation and prediction is to use foot contact information, either to determine the phase of the motion, or to directly define a loss [Harvey et al., 2020; Holden et al., 2017a; Starke et al., 2020; Thomas Geijtenbeek et al., 2013]. It is intended to provide a penalty to the system whenever the resulting pose does not have contact with the floor, e.g. if the character is floating or clipping through the floor. When using a position-based encoding, this can be directly implemented by a mean square error term applied to the distance of the toes to the floor. Currently, the system utilizes a relative angle-based encoding resulting in the necessity to first compute the actual position of the toes in space (see 4.2.1 [General Formats]), before being able to apply this loss.

This could be further augmented by taking into account the physics constraints related to the losing foot contact: Even though jumps are rare in the dataset, the body of a person losing foot contact has to abide by gravity and the conservation of energy.

Predict Leading Motion

Currently, the system is primarily focused on the output motion, while largely ignoring the leading motion for context (see 7.5.3 [Training Progression]).

To resolve this problem the system could be forced to take the leading motion into account. Analogous to the regression model in Chapter 6, the system could be forced to predict the leading dancer's motion N poses

into the future. With N being a new hyperparameter to the system. By choosing N sufficiently large, the system is forced to learn and anticipate the leading dancer's intent on a larger timescale. Caution is advised when selecting a value for N that is too large, as at some point the free will of the leading dancer makes predicting the motion impossible, having an adverse affect on the training procedure.

9.3.2. Limits

Higher Derivative Limits

As presented in 4.3.1 [Limit detection] and 7.3.6 [Pain], the system has pre-computed the angular limits of the dataset and enforces them by means of the pain criterion. At some point during the DRA-GAN training procedure the output motion was flying through space at an alarming rate, exhibiting very high frequency oscillation. To prevent this and other unrealistic motions it might be beneficial to extend the pain optimizer to higher derivatives. Firstly, compute the estimate of the velocities and accelerations from the original dataset, by utilizing the central differencing scheme. Secondly, find the limits of the higher derivatives, by means of the same methodology used in 4.3.1 [Limit detection] and B [Statistical Human Motion Model]. Lastly, approximate the output motion's higher derivatives, by means of applying higher order BDFs, and apply separate optimizers for penalizing exceeding the computed limits of the angular velocities and accelerations for each joint.

PCA Histograms

One of the downsides of the current limit detection approach is that it utilizes the Euler angle representation as the basis for detecting angular limits per channel. When reviewing the resulting histograms, as presented in Figure B.2, it becomes apparent that there are a few channels that are highly concentrated around the mean, but still have limits of $\pm 180°$ (e.g. 'Left-/RightLeg' and 'Left-/RightHand'), due the Euler based encoding.

The fact that the limits are encoded in Euler angle representation also results in the necessity to convert the generated output pose from the network internal 6D-PCA encoding back to the original Euler angle encoding before applying the pain loss.

To avoid this, it might be beneficial to compute the dataset histograms per channel on the PCA transformed data. This would have the following benefits:

- PCA transformation outputs non-cyclical zero mean data
  → clear limits will be present
- Less complex transformations during training
  → faster backpropagation and training times

9.4. Optimization

9.4.1. Optimizers

Layered Optimizers

Currently, there are three optimizers present in the system: Two for the generator and discriminator, handling the 'Binary Cross Entropy' (BCE) loss (see 7.1.1 [Loss Metric]), and another for the generator, handling the pain loss (see 7.3.6 [Pain]). All optimizers are governed by the same global learning rate parameter.

Given the current problem of overfocusing on a specific part of the data, a separate optimizer for the attention layers, with a lower learning rate, might alleviate this issue to some extent. Generalizing this approach, separate optimizers can be defined for each of the networks segments (attention, RNN core, FC final layer), while defining individual learning rates for each of them. This allows for a more fine tuned training procedure, at the expense of introducing more hyperparameters.

Gradient Norms

A common problem in vanilla RNNs and other neural network models is the problem of vanishing gradients. The network is trained by taking a step according to the gradients computed during backpropagation, but for certain architectures the gradients at the input layer are so small that training stagnates.

Currently, this does not appear to be an imminent problem encountered by the system, as upon visual inspection the parameters of the first segment (generator attention layer) show immediate progression, even after just a few iterations. Eventually however, after about 15-30 epochs the training stagnates for unknown reasons.

Monitoring the norms of the gradients might provide more insight into the gradient progression throughout the network.

The same visualization as presented in Figure 7.14 can be utilized, by exchanging the parameter matrices for the derivative matrices and adjusting the color scale.

Gradient Clipping

A general best practice when training RNN based models is gradient clipping, as proposed by Pascanu et al. [2013]. This refers to limiting the maximum absolute value of the gradients, to avoid overshooting when encountering steep regions or discontinuities in the loss landscape. This is illustrated in Figure 9.3.

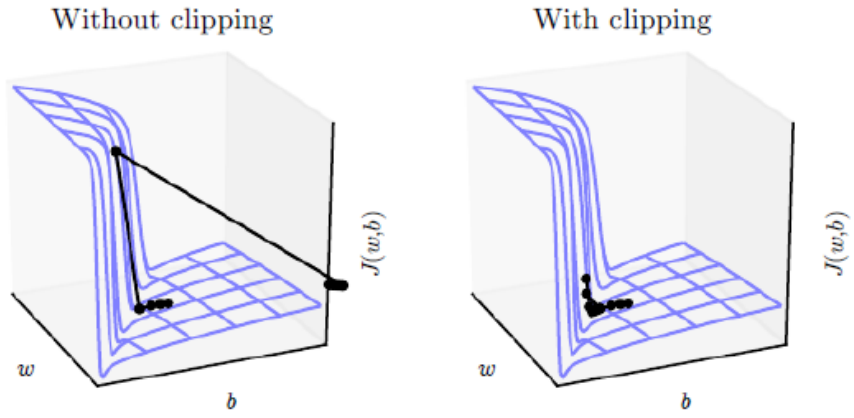


Figure 9.3: Gradient Clipping
Source: Figure adapted from Pascanu et al. [2013], copied from Neptune.AI

Truncated Backpropagation Through Time

Standard 'Backpropagation Through Time' (BPTT) unfolds the recurrent feedback loops over time from T_0 to T_N. This can leads to very large and complex backpropagation equations, that take a long time to process.

Truncated BPTT allows for training RNNs faster by avoiding backpropagation all the back to T_0, but instead only backtracking a few iterations in time [Jaeger, 2005].

9.4.2. BHPO

Practical Application

Currently, the training of a singular generative model takes between 1-2 weeks[3], which makes it impractical to use the BHPO and obtain useful results within a reasonable time frame.

The following options would allow for a practical application of the 'Bayesian Hyperparameter Optimization' (BHPO) on the generative model:

- Faster hardware: e.g. *NVIDIA TITAN X* from *AKOB*
- Cluster run: see 5.3 [Cluster Framework]
- Absolute loss: e.g. 7.4.3 [Advanced Loss] combined with the NPSS loss
- Faster training: e.g. Truncated BPTT

Unfortunately, the additional hardware provided by *AKOB* became unavailable throughout the research, but without a true absolute loss, providing a numerical metric for how good the resulting output is, running the BHPO on the generative model is meaningless. The intention of the BHPO is to provide an automated means of finding the optimal network configuration, but without a metric to define what is optimal the algorithm is prone to optimize the wrong function.

Currently the hyperparameter selection for the DRA-GAN is almost certainly suboptimal and the true potential of the model still remains unknown. Only when these four conditions are met can the true performance of the DRA-GAN model be evaluated.

[3]Run on the following hardware:: CPU - *AMD Opteron 6380* & GPU - *NVIDIA GTX 1060 6GB*

κTuning

As proposed in 5.2.1 [Exploration vs Exploitation], the hyper-hyperparameter κ can be tuned to achieve a certain probability of finding a new optimum. The extreme long training time of the current model (1-2 weeks) combined with the low probability of finding a new optimum (max. 13.8% $\leftarrow \kappa = 2.5$) makes using the BHPO impractical. Adjusting κ through reversing Chebyshev's inequality (see Equation 9.2) adjusted for the average training time, can lead to a metric that defines the expected average number of new optima over a given period of time.

$$\kappa = \sqrt{\frac{1}{\mathrm{SR}_{\max}} - 1} \tag{9.2}$$

Equation 9.2: κ tuning

An example:

- Desired rate of new optima: $1 \left[\frac{1}{\text{month}}\right]$
- Average training time: 1 [week]
- Minimum success rate required: 25 [%]
- Maximum κ: 1.73 [−]

Logarithmic Hyperparameters

As mentioned in 5.2.2 [Scope Creep & Re-parameterization]: Parameters with a large power range, such as the learning rate, can best be re-parameterized by optimizing for $\log(HP)$ instead, as it linearizes the search space.

Expand Hyperparameters

In contrast to the statement made in 5.2.2 [Scope Creep & Re-parameterization] to avoid hyperparameter scope creep, expanding the number of hyperparameters is the only true means of finding the true optimum.

Once the BHPO can be applied to the generative model and the final optimized set of hyperparameters still does not result in the desired visual output, it can be considered to expand the number of hyperparameters to be optimized.

Some adaptations that are likely expected to generate better results:

- Decoupling of generator and discriminator learning rate
- Variable number of limited judgment frames
- Variable batch size
- Variable noise injection
- Variable chunk length

9.5. Validation

Simplified Network

The current system has an option to simplify the input to only the root's position and rotation ($3 + 6 = 9$ floats, with 6D rotations encoding). This was intended to speed up the training runtime to validate the model performance on a simplified problem. The model runs and can be computed, but due to the parallelization of tensor operations on the GPU, reducing the size of the tensors does not notably reduce the training time, it does however reduce the memory load significantly. For this reason the training run was halted, as it was deemed impractical to occupy the hardware for a few weeks to run a simplified network, if a complete run could be run instead.

Provided that the training time is reduced to a reasonable time frame, running this simplified model does still pose as a valuable tool for validation.

The total number of parameters of this minimized DRA-GAN network can be as low as 28.284, as presented in Table 9.1. This is only 0.23% of the number of parameters presented in the example presented in Table 7.5. 28.284 parameters still sounds like a lot to take in at a glance, but considering that all parameters can fit inside a $169x169$ px image shows that this simplification allows for examining the inner workings of the network in more detail at a much smaller scale. Figure 9.4 shows an actual visualization of all parameters in the minimized network example.

Minimized DRA-GAN # Parameters		
Network	**# Parameters**	**% Total**
Attention Layers:		
Generator	1,368	4.84%
Discriminator	1,368	4.84%
Recurrent Units:		
Generator	15,984	56.51%
Discriminator	9,204	32.54%
Linear Layers:		
Generator	333	1.18%
Discriminator	27	0.10%
Combined:		
Generator	17,685	62.53%
Discriminator	10,599	37.47%
Total	28.284	100%

Table 9.1: DRA-GAN: Minimized Parameter Distribution Example
Root-only 2 layer GRU with G-OutputRatio = 1.0 & D-OutputRatio = 0.75

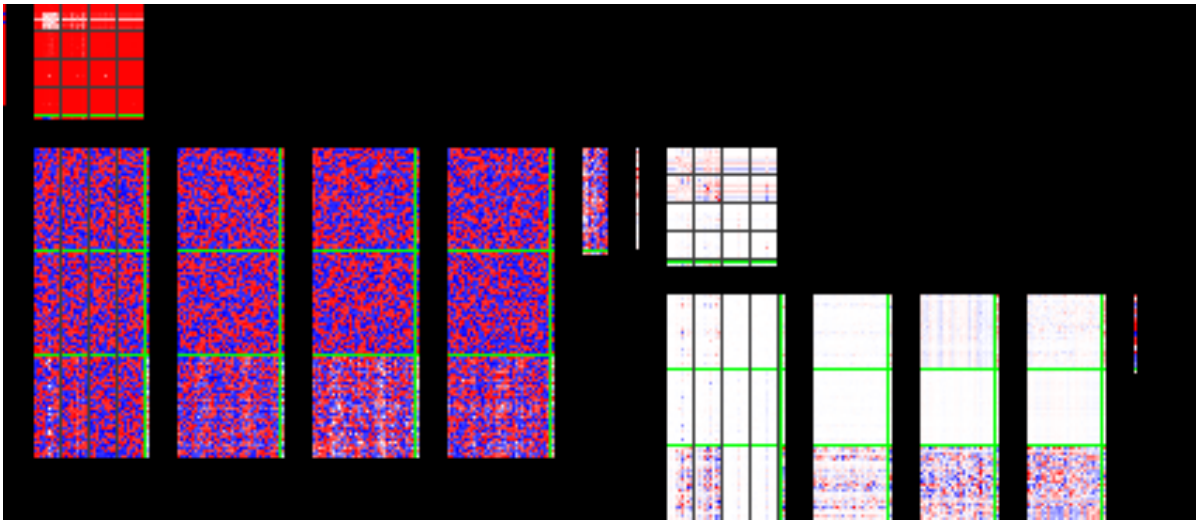


Figure 9.4: DRA-GAN: Minimized Network Parameters Visualization Example
Blue: Negative parameters & White: Zero Parameters & Red: Positive Parameters
Green & Grey: Logical separators

Past Model Progression

As presented in 7.1.3 [Progress Insight], gaining insight into the networks training progression is difficult for any GAN. A potential solution to this issue, that could be applied to any GAN, utilizes an adaptation to the past model injection methodology, as present in Section 9.2.1.

It might be possible to gain insight into the networks' performance improvement over time, by comparing the current performance against their previous counterparts. Borrowing the tug-of-war analogy from 7.1.3 [Progress Insight]: It is impossible to assess the absolute strength of two teams of equal strength that train and progress at equal rates, as every match will be a tie. If it were possible to match up these teams against past versions of their opponents they would manage to win and show that they have indeed progressed since the previous point in time.

Therefore, the proposed technique will iterate over N past models at regular intervals and evaluate the network once for the current generator vs the past discriminator and once for the current discriminator vs the past generators, plotting the final loss of the network for each match up. The gradient of the resulting plot should provide a better indication of the progression of each network over time.

Turning Test for Motion Synthesis

First proposed in the preliminary report, the 'Turning Test for Motion Synthesis' (TTMS) was intended to be one of two final tests to be performed to validate the model's performance.

Analogous to both the original Turing test and the discriminator in a GAN, the TTMS is intended to utilize human judgment to validate the DRA-GAN's output motions. The human participants are to be presented with a random motion chunk, showcasing the leading and output motion together, and are tasked with classifying the chunk as real or fake, substituting the discriminator network. Once the generator would be successful in fooling human experts (e.g. choreographers, dancers, dance critics), the model can be considered a success, ready for stage performances and a paper submission to *SIGGRAPH*.

10

Conclusions

The objective of this research was to design an algorithmic framework capable of generating reactionary dance improvisations for the show '*AI-man*'. Throughout this research a novel neural network architecture, the 'Differential Recurrent Attention GAN' (DRA-GAN), as well as the complete framework to train it were designed to achieve this task. However, due to limited processing power, the question whether the designed framework is capable of succeeding at this task still remains unanswered.

The external goal of this research is partially met: The framework for streaming the generator's output in reaction to live data received from the *OptiTrack* motion capture (MoCap) system is in place (see Section 8.1.2 [*OptiTrack* Streaming]). However, the final visual result and the interactivity of the output are not sufficient for the desired stage performance (see Section 7.5.2 [Visual Output]).

The internal goal of this research is mostly met: The extensive research and iterative design through inductive reasoning have gradually improved the system to a state that should be capable of achieving the desired output (see Section 3.3.2 [Model Improvements]), but is yet to be confirmed after an extensive 'Bayesian Hyperparameter Optimization' (BHPO) run. Solving the ambiguity of subjective and objective evaluation was partially addressed, by means of the advanced loss function presented in 7.4.3 [Advanced Loss], but is yet to be extended by additional means, such as the 'Normalized Power Spectrum Similarity' (NPSS) loss (see Section 9.3.1 [Normalized Power Spectrum Similarity]).

All research questions have been explored and at least partially answered. A sufficiently large dataset has been acquired and (see Section 4.5 [Dataset Overview]) and processed accordingly, to allow training neural networks (see Section 4.4 [Data Preparation]). Various neural network architectures have been researched and the appropriate architecture selected (see Section 6.1 [Design Choices] and Appendix F [Preliminary Report]), as well as the appropriate loss functions for motion generation (see Section 7.1.1 [Loss Metric] and Section 7.4.3 [Advanced Loss]). Appropriate frameworks have been found to optimize the initial design and facilitate the training procedure (see Chapter 5 [*DanceNet-BHPO* Framework]).

The proposed research hypothesis could not be confirmed, as the current state of the final model was unable to generate acceptable motions to showcase to human experts. However, neither could the hypothesis be denied, as the full potential of the model is yet to be explored.

Reflecting on the original research framework, as presented in Figure 2.1, it can be seen that all elements, except for the validation have been executed accordingly:

1. Literature Study:
   90+ papers have been reviewed
   (see Appendix F [Preliminary Report])
2. Data Handling:
   A sufficiently large dataset was acquired and pre-processed
   (see Chapter 4 [Data Handling])
3. Model Design:
   Two types of models have been designed

- Regression Model: Successful training, but solving the wrong problem
  (see Chapter 6 [Regression Model])
- Generative Model: DRA-GAN solves the right problem, but training stagnates in mode collapse
  (see Chapter 7 [Generative Model])

4. Optimization & Application:
   Frameworks have been designed for optimization and live visualization of the models

   - BHPO: Framework is working as desired , but DRA-GAN training times are too long
     (see Chapter 5 [*DanceNet-BHPO* Framework])
   - Live Interaction: Framework is setup and operational, but minor bugs remain
     (see Chapter 8 [Live Interaction])

5. Validation:
   Live testing and Turing test for motion synthesis have been prepared for, but canceled due to the limited quality of the final output motions

Overall, all desired elements are in place, but the full potential of the DRA-GAN model remains yet unexplored, due to limited computing hardware. Finally, given the limited interactivity and variety of the current output motions, both the live interaction and the validation aspects of this research were unable to be executed to the desired extent.

10.1. Scientific Contribution

Many techniques utilized in this research are the result of an extensive literature study, utilizing methodologies presented by other researcher or adaptations thereof. The reader is highly encouraged to consult the original papers, as presented in the bibliography[1], for deeper insight into the key methodologies used throughout this research. However this research is not purely applied research, a few select novel methodologies were developed specifically for this research, as reiterated below.

In addition to these methodologies, the dataset acquired for this research is the second largest MoCap dataset in term of total duration, compared to currently publicly available datasets (see Section 4.5 [Dataset Overview]).

Neither the BHPO framework, nor the *DanceNet* model, nor the *AKOB* dataset are open-source or under public development, they are however freely available for research purposes under specific conditions. In case of legitimate interest in the underlying code or dataset, please contact the author via ' Henricus@Basien.de ' with your request.

Each of the novel methodologies presented in this report are briefly restated, in the context of their contribution to the scientific community:

10.1.1. Deep Learning

GAN Training

As frequently mentioned throughout this report, training 'Generative Adversarial Networks' (GANs) is a non-trivial endeavor (see Section 7.1.3 [Problems with GANs]), with a significant number of best practices to adhere to (see Section 7.1.4 [Best Practices]).

Three methodologies presented in this report attempt to increase the insight into the GAN training procedure, avoid losing time when training unsuccessful models and prevent a trivial solution when training GANs on time-series data:

Improved GAN Insight 7.4.3 [Advanced Loss] presents a novel metric to improve gauging the progress of the GANs training progression. The metrics defined in 7.4.3 [Discriminator Metrics] can be applied to any GAN independent of the data utilized. The addition of the pain loss, defined in 7.4.3 [Combined Loss], indicates the benefit of adding data type specific metrics to the overall loss, improving insight into the progress even further.

[1]The initial literature study is presented in the preliminary report.

Early Break While the concept of early stopping is not novel and frequently utilized in deep learning (DL), the conventional algorithms are only capable of minimizing or maximizing a loss function and are unable to handle loss functions with positive and negative derivatives. The 'Binary Cross Entropy' (BCE) loss utilized by the GAN can have both positive and negative derivatives, depending on whether the generator or discriminator is improving faster relative to the other.

The early break criterion, as presented in 5.1.2 [Early Breaking] and Algorithm D.1, solves this issue by utilizing the absolute value of the double exponentially filtered loss derivative as the primary metric to check for progress stagnation.

Limited Judgement When utilizing a GAN for generating time series data, the problem of initial state detection by the discriminator is present. The methodology described in 7.3.7 [Limited Judgement], alleviates this shortcut to a trivial solution by depriving the discriminator of output data largely contaminated by the initial state of the network. This is a problem that common feed forward based GANs do not suffer from, as they do not retain memory of past iterations over time, but becomes prevalent for GANs utilizing memory elements and feedback loops during training.

DRA-GAN

The 'Differential-Recurrent-Attention GAN' (DRA-GAN), presented in 7.4 [DRA-GAN], is a novel architecture design that is theoretically capable of generating new reactionary data to any differentiable time sequence data. Using the model with other data types requires removing those network elements that are strictly related to (human) motion data, or adapting them to suit the new data type accordingly.

The motion specific network elements are:

- 6D Rotations (see Section 4.4.1 [6D-Rotations])
- Pain Criterion (see Section 7.3.6 [Pain])

Besides the current use case for motion generation, the DRA-GAN could also be utilized for generating:

- Musical accompaniment
  Leading dancer → Main melody & Following dancer → Instrumental accompaniment
- Synthetic bivariate time series datasets
  Leading dancer → Variable #1 & Following dancer → Variable #2
- Body language comprehension and mimicking for humanoid androids, in the future
  Leading dancer → Subject A & Following dancer → Subject B

Derivative Input The DRA-GAN is mostly a combination of various pre-existing network architectures. Even the 'Derivative Output' segment could be seen as an adaptation of the 'Add & Norm' segments in the 'Transformer' network [Vaswani et al., 2017] or a residual connection through time. However, the 'Derivative Input' (DI) segment combines traditional control theory and differential equations with deep learning techniques, by supplying the first derivative of the state, in addition to the state, as an input to the network. This allows the neural network to focus on solving the desired task, without the need to learn how to differentiate the state. If the researcher can reasonably assume that the state's derivative(s) are relevant for solving the problem, DI can assist the system's learning process. With possible improvements to this segment remaining, such as the 'Backward Differentiation Formulas' (BDF) based input (see Section 9.1.2 [BDF Input]) and extending the input to the states second derivative, it offers flexibility in its application.

10.1.2. Miscellaneous

κTuning

Equation 9.2 offers the means of tuning the BHPO's hyper-hyperparameter κ, utilizing Chebyshev's inequality (see Equation 5.3).

Manual hyper-hyperparameter tuning is mostly a tedious and complicated process, but κ tuning provides a simple and quick means of arriving at a range of acceptable values for κ. Enabling an objective and analytical trade-off between exploration and exploitation (see Section 5.2.1 [Exploration vs Exploitation]).

Analytical Model of the Human Body

Table B.4 provides an analytical representation of the distribution of joint angles of the human body. While not directly applied within this research, it may yet serve other researchers in developing better analytical models of the human body and its motions.

10.2. Discussion

The following two papers are examples of the current state of the art in motion generation and their primary achievements, both published during the time frame of this research:

- Dance Revolution: Long-Term Dance Generation with Music via Curriculum Learning [Huang et al., 2020]
  - Conference: *ICLR* 2021
  - Journal: N/A (Preprint)
  - Primary Achievement: 1-minute long generation of dance motions to music in 2D (@15 FPS)
- Robust Motion In-betweening [Harvey et al., 2020]
  - Conference: *SIGGRAPH* 2020
  - Journal: ACM Transactions on Graphics
  - Primary Achievement: Interpolates motion cycles between key frames 3-44 frames apart (@30 FPS)

The first paper is the closest to the objective attempted to be achieved by this research, as it includes true long-term motion generation in reaction to another time series; music. It was written by six researchers from Microsoft STCA and Fudan University. This paper is so bleeding edge that it is still in preprint and has, unfortunately, only been discovered by the researcher mere days before completion of this thesis and is hence only addressed in this section.

The second paper is another example of the current state of the art in motion generation. It was written by four researchers from Polytechnique Montreal, Ubisoft Montreal and McGill University. The generation of new motion frames is restricted to 0.1-1.5 seconds, with a clearly defined boundary at the start and end of each pose. This problem is clearly defined and limited in its scope and yet it is considered state of the art.

Throughout this research the true scope of the desired research objective has become clear; in hindsight given this context, the objective set out to be achieved in this research appears to be on the bleeding edge of the current state of the art and exceed the expected scope of a regular MSc. thesis.

While looking extremely promising, the task presented in the 'Dance Revolution' paper has three primary distinctions from this research: The motion data is presented as 2D points on a screen[2], the context data is music not another dancer and the context data is pre-processed in its entirety, instead of sequence to sequence.

Nevertheless, it presents the closest research to this one to date and much can be learned by comparing the techniques used. Both models show a surprising amount of similarity, as very similar issues had to be addressed. Table 10.1 compares some methodologies used in their paper, to those used by the final DRA-GAN model.

Methodology	**Huang et al. [2020]**	**This Research**
Motion network	3 Layer LSTM with H= 1024	3 Layer LSTM with H= 567
Attention Mechanism	Multi-head self-attention	Soft self-attention
Optimizer	Adam	Adam
Learning rate	$1 \cdot 10^{-4}$	$5 \cdot 10^{-5}$
Objective Metric	Fréchet Inception Distance (FID)	Modified discriminator metrics + Pain
Human Evaluation	3 preference metrics	Binary real/fake classification (Suggested)
Learning Technique	Curriculum learning & Teacher forcing	Full BPTT & GAN

Table 10.1: Methodology comparison between this research and Huang et al. [2020]

This comparison is only meaningful as the content of their paper was only discovered after the DRA-GAN model was completed. It showcases that the DRA-GAN model shares many similarities with their successful model, increasing the confidence that the general approach chosen is indeed correct.

Their research still utilizes some methodologies that were not applied in the DRA-GAN model and are novel to their approach. Most important is the different approach to training the model, which utilizes curriculum learning and a gradually diminishing teacher forcing approach "to alleviate error accumulation of autoregressive models in long motion sequence generation".

Perhaps a combination of the DRA-GAN and their approach may prove successful in achieving the objective of this research.

[2]Essentially a flattened 2D representation of position-based pose information. This is because the data was extracted from videos instead of a real MoCap system.

Despite the fact that the final output of the model does not produce the desired results, in its current state, this thesis provides a broad overview of the methodologies used, common pitfalls and possible solutions to generate human motions with GANs. The current DRA-GAN model is showing a clear training progression and, given appropriate hyperparameter tuning, may yet prove successful in the desired task. Due to the limited hardware available to train the network it can be safely said that the current model has not yet fully realized its full potential.

IV

Appendices

Bibliography

Emre Aksan, Manuel Kaufmann, and Otmar Hilliges. Structured prediction helps 3D human motion modelling. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-Octob(Iccv):7143–7152, 2019. ISSN 15505499. doi: 10.1109/ICCV.2019.00724. arXiv-ID: 1910.09070.

Omid Alemi. GrooveNet : Real-Time Music-Driven Dance Movement Generation using Artificial Neural Networks. *Sigkdd-W*, (July):6, 2017.

A. Aristidou, D. Cohen-Or, J. K. Hodgins, and A. Shamir. Self-similarity analysis for motion capture cleaning. *Computer Graphics Forum*, 37(2):297–309, 2018a. ISSN 14678659. doi: 10.1111/cgf.13362.

Andreas Aristidou, Daniel Cohen-Or, Jessica K. Hodgins, Yiorgos Chrysanthou, and Ariel Shamir. Deep motifs and motion signatures. *SIGGRAPH Asia 2018 Technical Papers, SIGGRAPH Asia 2018*, 37(06), 2018b. ISSN 15577368. doi: 10.1145/3272127.3275038.

Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–15, 2015. arXiv-ID: 1409.0473.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. 2018. arXiv-ID: arXiv:1803.01271v2.

Judith Bütepage, Michael J. Black, Danica Kragic, and Hedvig Kjellström. Deep representation learning for human motion prediction and classification. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:1591–1599, 2017. doi: 10.1109/CVPR.2017.173. arXiv-ID: 1702.07486.

Caroline Chan, Shiry Ginosar, Tinghui Zhou, and Alexei Efros. Everybody dance now. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2019-Octob, 2019. ISBN 9781728148038. doi: 10.1109/ICCV.2019.00603.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pages 1724–1734, 2014. doi: 10.3115/v1/d14-1179. arXiv-ID: 1406.1078.

CMU Graphics Lab. CMU Graphics Lab Motion Capture Database, 2016.

Minjing Dong and Chang Xu. On retrospecting human dynamics with attention. *IJCAI International Joint Conference on Artificial Intelligence*, 2019-Augus:708–714, 2019. ISSN 10450823. doi: 10.24963/ijcai.2019/100.

Felix Gaisbauer, Jannes Lehwald, Janis Sprenger, and Enrico Rukzio. Natural posture blending using deep neural networks. *Proceedings - MIG 2019: ACM Conference on Motion, Interaction, and Games*, 2019. doi: 10.1145/3359566.3360052.

Partha Ghosh, Jie Song, Emre Aksan, and Otmar Hilliges. Learning human motion models for long-Term predictions. *Proceedings - 2017 International Conference on 3D Vision, 3DV 2017*, pages 458–466, 2018. doi: 10.1109/3DV.2017.00059.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS 2014)*, pages 2672–2680, 2014.

Anand Gopalakrishnan, Ankur Mali, Dan Kifer, C. Lee Giles, and Alexander G. Ororbia. A neural temporal model for human motion prediction. *arXiv*, 2018. ISSN 23318422.

Alex Graves. Neural Turing Machines. pages 1–26, 2014. arXiv-ID: arXiv:1410.5401v2.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016. ISSN 14764687. doi: 10.1038/nature20101.

Liang Yan Gui, Yu Xiong Wang, Xiaodan Liang, and José M.F. Moura. Adversarial Geometry-Aware Human Motion Prediction. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11208 LNCS, pages 823–842. Springer Verlag, 2018a. ISBN 9783030012243. doi: 10.1007/978-3-030-01225-0_48.

Liang Yan Gui, Yu Xiong Wang, Deva Ramanan, and José M.F. Moura. Few-shot human motion prediction via meta-learning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11212 LNCS:441–459, 2018b. ISSN 16113349. doi: 10.1007/978-3-030-01237-3_27.

Félix G. Harvey, Mike Yurick, Derek Nowrouzezahrai, and Christopher Pal. Robust motion in-betweening. *ACM Transactions on Graphics*, 39(4), 2020. ISSN 15577368. doi: 10.1145/3386569.3392480.

Alejandro Hernandez, Jurgen Gall, and Francesc Moreno. Human motion prediction via spatio-temporal inpainting. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-Octob(Iccv):7133–7142, 2019. ISSN 15505499. doi: 10.1109/ICCV.2019.00723. arXiv-ID: 1812.05478.

Josef Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. *Iclr*, (April):14, 1991.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997. ISSN 08997667. doi: 10.1162/neco.1997.9.8.1735.

Daniel Holden. Robust solving of optical motion capture data by denoising. *ACM Transactions on Graphics*, 37 (4):1–12, 2018. ISSN 15577368. doi: 10.1145/3197517.3201302.

Daniel Holden, Jun Saito, and Taku Komura. Learning an inverse rig mapping for character animation. In *Proceedings - SCA 2015: 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 165–173, 2015. ISBN 9781450334969. doi: 10.1145/2786784.2786788.

Daniel Holden, Taku Komura, and Jun Saito. Phase-Functioned Neural Networks for Character Control. *ACM Transactions on Graphics*, 36(4), 2017a.

Daniel Holden, Jun Saito, and Taku Komura. Learning Inverse Rig Mappings by Nonlinear Regression. 23(3): 1167–1178, 2017b.

Ruozi Huang, Huang Hu, Wei Wu, Kei Sawada, and Mi Zhang. Dance Revolution: Long Sequence Dance Generation with Music via Curriculum Learning. *arXiv*, pages 1–14, 2020. ISSN 23318422. arXiv-ID: 2006.06119.

Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3. 6M. *Ieee Transactions on Pattern Analysis and Machine in□Ligence*, page 1, 2014. ISSN 01628828.

Arieh Iserles. *A First Course in the Numerical Analysis of Differential Equations*. Cambridge University Press, USA, 2nd edition, 2008. ISBN 0521734908.

Herbert Jaeger. A tutorial on training recurrent neural networks , covering BPPT , RTRL , EKF and the " echo state network " approach. *ReVision*, 2002:1–46, 2005.

I.T. Jolliffe. Principal Component Analysis, 2002. ISSN 10780998.

John Kender and One Microsoft Way. HP-GAN : Probabilistic 3D human motion prediction via GAN Emad Barsoum. 2018. doi: 10.1109/CVPRW.2018.00191.

Philipp Kratzer, Marc Toussaint, and Jim Mainprice. Prediction of Human Full-Body Movements with Motion Optimization and Recurrent Neural Networks. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 1792–1798, 2020. ISSN 10504729. doi: 10.1109/ICRA40945.2020.9197290. arXiv-ID: 1910.01843.

Justin Kruger and David Dunning. Unskilled and unaware of it: How difficulties in recognizing one's own incompetence lead to inflated self-assessments. *Journal of Personality and Social Psychology*, 77(6):1121–1134, 1999. ISSN 00223514. doi: 10.1037/0022-3514.77.6.1121.

Maosen Li, Siheng Chen, Xu Chen, Ya Zhang, Yanfeng Wang, and Qi Tian. Symbiotic Graph Neural Networks for 3D Skeleton-based Human Action Recognition and Motion Prediction. pages 1–19, 2019. arXiv-ID: 1910.02212.

Lucas Liu, Duri Long, Swar Gujrania, and Brian Magerko. Learning movement through human-computer co-creative improvisation. *ACM International Conference Proceeding Series*, 2019. doi: 10.1145/3347122.3347127.

Ming Yu Liu and Oncel Tuzel. Unrolled Generative Adversarial Networks. *Advances in Neural Information Processing Systems*, pages 469–477, 2016. ISSN 10495258. arXiv-ID: 1606.07536.

Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael Black. AMASS: Archive of motion capture as surface shapes. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2019-Octob, 2019. ISBN 9781728148038. doi: 10.1109/ICCV.2019.00554. arXiv-ID: 1904.03278.

Alessandro Manzi, Laura Fiorini, Raffaele Limosani, Paolo Dario, and Filippo Cavallo. Two-person activity recognition using skeleton data. *IET Computer Vision*, 12(1), 2018. ISSN 17519640. doi: 10.1049/iet-cvi.2017.0118.

Wei Mao, Miaomiao Liu, Mathieu Salzmann, and Hongdong Li. Learning trajectory dependencies for human motion prediction. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-Octob(Iccv): 9488–9496, 2019. ISSN 15505499. doi: 10.1109/ICCV.2019.00958. arXiv-ID: 1908.05436.

Wei Mao, Miaomiao Liu, and Mathieu Salzmann. History Repeats Itself: Human Motion Prediction via Motion Attention. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12359 LNCS:474–489, 2020. ISSN 16113349. doi: 10.1007/978-3-030-58568-6_28. arXiv-ID: 2007.11755.

Albert W. Marshal and Ingram Olkin. A One-Sided Inequality of the Chebyshev Type. *Annals of Statistics*, 19 (3):1403–1433, 1991.

Julieta Martinez, Michael J. Black, and Javier Romero. On human motion prediction using recurrent neural networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:4674–4683, 2017. doi: 10.1109/CVPR.2017.497. arXiv-ID: 1705.02445.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training Recurrent Neural Networks. 2013. arXiv-ID: arXiv:1211.5063v2.

Dario Pavllo, Christoph Feichtenhofer, Michael Auli, and David Grangier. Modeling Human Motion with Quaternion-Based Neural Networks. *International Journal of Computer Vision*, 128(4):855–872, 2019. ISSN 15731405. doi: 10.1007/s11263-019-01245-6. arXiv-ID: 1901.07677.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. *MIT Press, Cambridge, MA*, 1(V):318–362, 1986.

Ju Shen and Jianjun Yang. Automatic Human Animation for Non-Humanoid 3D Characters. In *Proceedings - 2015 14th International Conference on Computer-Aided Design and Computer Graphics, CAD/Graphics 2015*, pages 220–221, 2016. ISBN 9781467380201. doi: 10.1109/CADGRAPHICS.2015.31.

Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Josh Susskind, Wenda Wang, and Russ Webb. Learning from Simulated and Unsupervised Images through Adversarial Training. 2016. arXiv-ID: arXiv:1612.07828v2.

Sebastian Starke, Yiwei Zhao, Taku Komura, and Kazi Zaman. Local motion phases for learning multi-contact character movements. *ACM Transactions on Graphics*, 39(4), 2020. ISSN 15577368. doi: 10.1145/3386569.3392450.

Yongyi Tang, Lin Ma, Wei Liu, and Wei-shi Zheng. Long-Term Human Motion Prediction by Modeling Motion Context and Enhancing Motion Dynamic. pages 935–941, 2017.

Thomas Geijtenbeek, Michiel van de Panne, and A. Frank van der Stappen. Flexible Muscle-Based Locomotion for Bipedal Creatures. *ACM Transactions on Graphics*, 32(6), 2013.

Ilya Tolstikhin, Sylvain Gelly, Olivier Bousquet, Carl Johann Simon-Gabriel, and Bernhard Schölkopf. AdaGAN: Boosting generative models. *Advances in Neural Information Processing Systems*, 2017-Decem:5425–5434, 2017. ISSN 10495258. arXiv-ID: 1701.02386.

University Of Cyprus. DanceDB, 2012.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017-Decem (Nips):5999–6009, 2017. ISSN 10495258. arXiv-ID: arXiv:1706.03762v5.

Andreas Veit. Residual Networks Behave Like Ensembles of Relatively Shallow Networks. pages 1–9, 2016. arXiv-ID: arXiv:1605.06431v2.

Qi Wang, Thierry Artières, Mickael Chen, and Ludovic Denoyer. Adversarial learning for modeling human motion. *Visual Computer*, 36(1):141–160, 2020a. ISSN 01782789. doi: 10.1007/s00371-018-1594-7.

Xin Wang, Xiaotao Jiang, Gloria Rumbidzai Regedzai, Haohao Meng, and Lingyun Sun. Gated neural network framework for interactive character control. *Multimedia Tools and Applications*, 2020b. ISSN 15737721. doi: 10.1007/s11042-020-08792-y.

Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1316–1324, 2018. ISSN 10636919. doi: 10.1109/CVPR.2018.00143. arXiv-ID: 1711.10485.

Katsu Yamane, Yuka Ariki, and Jessica Hodgins. Animating non-humanoid characters with human motion data. *Computer Animation 2010 - ACM SIGGRAPH / Eurographics Symposium Proceedings, SCA 2010*, pages 169–178, 2010.

Dongsheng Zhou, Xinzhu Feng, Pengfei Yi, Xin Yang, Qiang Zhang, Xiaopeng Wei, and Deyun Yang. 3D Human Motion Synthesis Based on Convolutional Neural Network. *IEEE Access*, 7:66325–66335, 2019a. ISSN 21693536. doi: 10.1109/ACCESS.2019.2917609.

Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019b. ISBN 9781728132938. doi: 10.1109/CVPR.2019.00589. arXiv-ID: 1812.07035.

A

Coordinate System Definitions

Various data formats and programs have been used throughout this research, all of which are using a different coordinate system. This disparity of conventions is most unfortunate, however it cannot be changed and has to be dealt with. Therefore all coordinate system conventions that are used throughout the researched are listed here, in an effort to clear up some of the confusion around them.

A visualization of the various coordinate systems is presented in Figure A.1.

A.1. OptiTrack

- CS-Type: 3D-Cartesian
  - Axis-Up: Y
  - Axis-Forward: Z
  - Orientation: Right Handed
- Rotation type: Quaternions

A.2. BVH

- CS-Type: 3D-Cartesian
  - Axis-Up: Y
  - Axis-Forward: Z
  - Orientation: Right Handed
- Rotation type: Euler-Intrinsic
  - Rotation Order: Y→X→Z

A.3. Blender

- CS-Type: 3D-Cartesian
  - Axis-Up: Z
  - Axis-Forward: Y
  - Orientation: Right Handed
- Rotation type: Variable {Euler,Quaternions,Axis-Angle}

A.4. Unreal Engine 4

- CS-Type: 3D-Cartesian
  - Axis-Up: Z
  - Axis-Forward: X
  - Orientation: Left Handed
- Rotation type: Euler-Extrinsic
  - Rotation Order: X→Y→Z

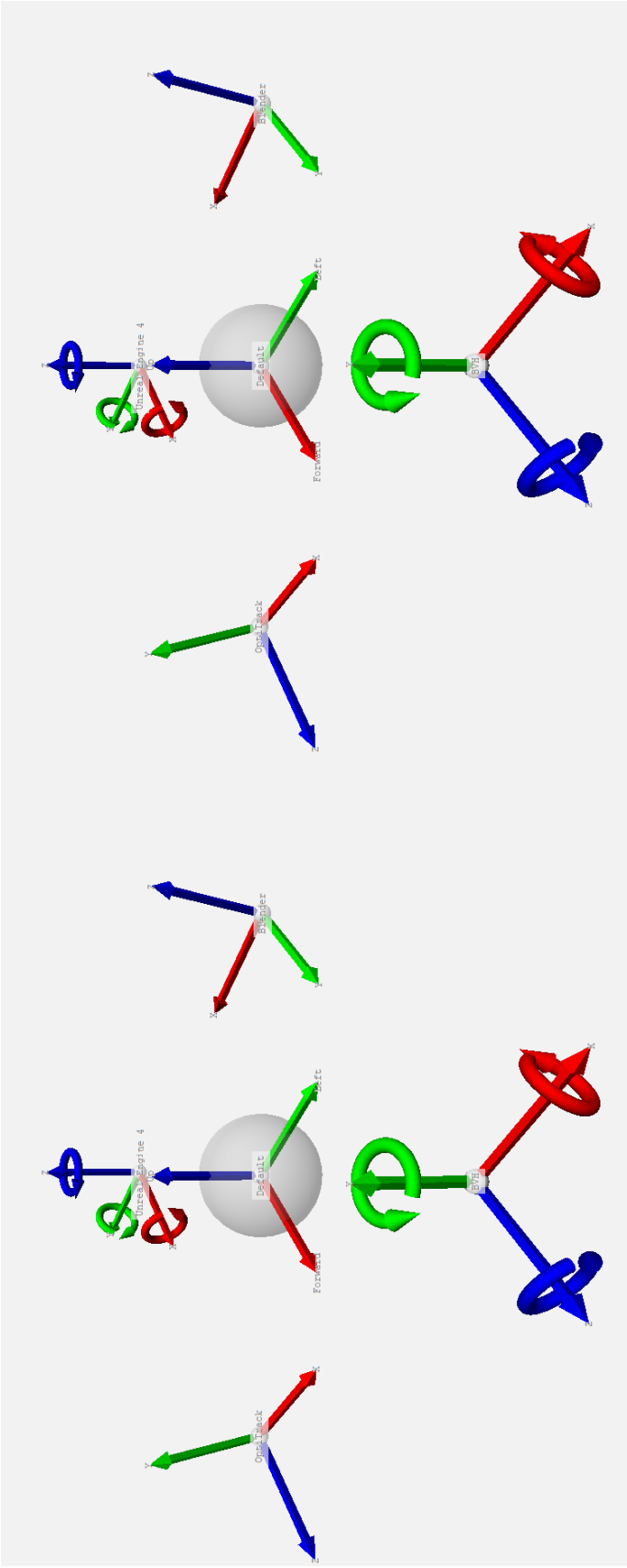


Figure A.1: Coordinate system definitions
Printed in cross-eyed stereoscopy, for better 3D insight.

B

Statistical Human Motion Model

A extensive regression analysis, on the joint histogram data, was performed using a set of applicable 'Probability Density Functions' (PDFs), to find an analytical representation for each recorded variable. The process and results are described in the following appendix.

B.1. Histograms

Full-scale visualization of the joint histograms and their best-fit PDF regression can be seen in Figure B.2 & B.3.

B.1.1. Histogram Resolution

The resolution for the histogram was based on the square root of the number of samples, which satisfies divisions by 2, 5, & 9. This allowed for a optimal visual result and this procedure is detailed in Equation B.1.

$$\begin{aligned} \#\text{Frames} &= 6.482.101 \\ \#\text{Bins} &\leq \text{int}\left(\sqrt{\#\text{Frames}}\right) \text{ where } (\#\text{Bins} \bmod \{2,5,9\} = 1) \\ \alpha_{\text{Range}} &= 2 \cdot 180^{\circ} = 360^{\circ} \\ \alpha_{\text{Res}} &= 0.2^{\circ} \\ \#\text{Bins} &= 1 + \frac{\alpha_{\text{Range}}}{\alpha_{\text{Res}}} = 1801 \end{aligned} \tag{B.1}$$

Equation B.1: Joint Histogram Resolution

B.1.2. Histogram Cleaning

Prior to plotting and regression obvious problems with the histogram have been fixed. Most issues were apparent with the finger joints, which very frequently lost tracking and recorded incorrect default values instead of the actual angles. The pre-cleaning include the following operations:

1. Removal of all zero values, due to *OptiTrack* sometimes setting the value to 0 when tracking is lost.
2. Removal of all 'spikes', due to *OptiTrack* sometimes returning to a previous 'default' when tracking is lost.

Spikes where defined as follows:

- 'Common' spike: $c[i-1] \cdot 2 \leq c[i] \geq c[i+1] \cdot 2$
- 'Lonely' spike: $c[i-1] = c[i+1] = 0 \;\&\; c[i] \neq 0$

For a continuous distribution with a resolution of 0.2°, a sudden doubling of occurrence or random spike within $10ms$ with no means of getting there, were deemed unrealistic and attributed to tracking loss.

B.2. PDFs

All non-finger channels, except for 'Hips-ψ', can be analytically described by a singular PDF, with high confidence '$R^2 >= 96.97\%$'.

The details results of all PDF regressions, of the joint channel histogram data, are presented at the end of this appendix.

B.2.1. Distributions Used

The PDFs of the following distributions were used in the regression analysis:

1. Gamma $[0, \infty]$: $\text{PDF}(x, a) = \frac{x^{a-1} \cdot e^{-x}}{\Gamma(a)}$
2. Beta $[0, 1]$: $\text{PDF}(x, a, b) = \frac{\Gamma(a+b) \cdot x^{a-1} \cdot (1-x)^{b-1}}{\Gamma(a) \cdot \Gamma(b)}$
3. Gaussian: $\text{PDF}(x) = \frac{e^{\frac{-x^2}{2}}}{\sqrt{2\pi}}$
4. Skew Norm: $\text{PDF}(x, a) = 2 \cdot \text{PDF}_{\text{Gaussian}}(x) \cdot \text{CDF}_{\text{Gaussian}}(a \cdot x)$
5. ExpoNorm: $\text{PDF}(x, K) = \frac{e^{\frac{1}{2 \cdot K^2} - \frac{x}{K}}}{2 \cdot K} \cdot \text{erfc}\left(-\frac{x - \frac{1}{K}}{\sqrt{2}}\right)$
6. Cauchy: $\text{PDF}(x) = \frac{1}{\pi \cdot (1 + x^2)}$
7. Laplace: $\text{PDF}(x) = \frac{e^{-|x|}}{2}$

The general shape of each PDF used in the regression analysis can be seen in Figure B.1.

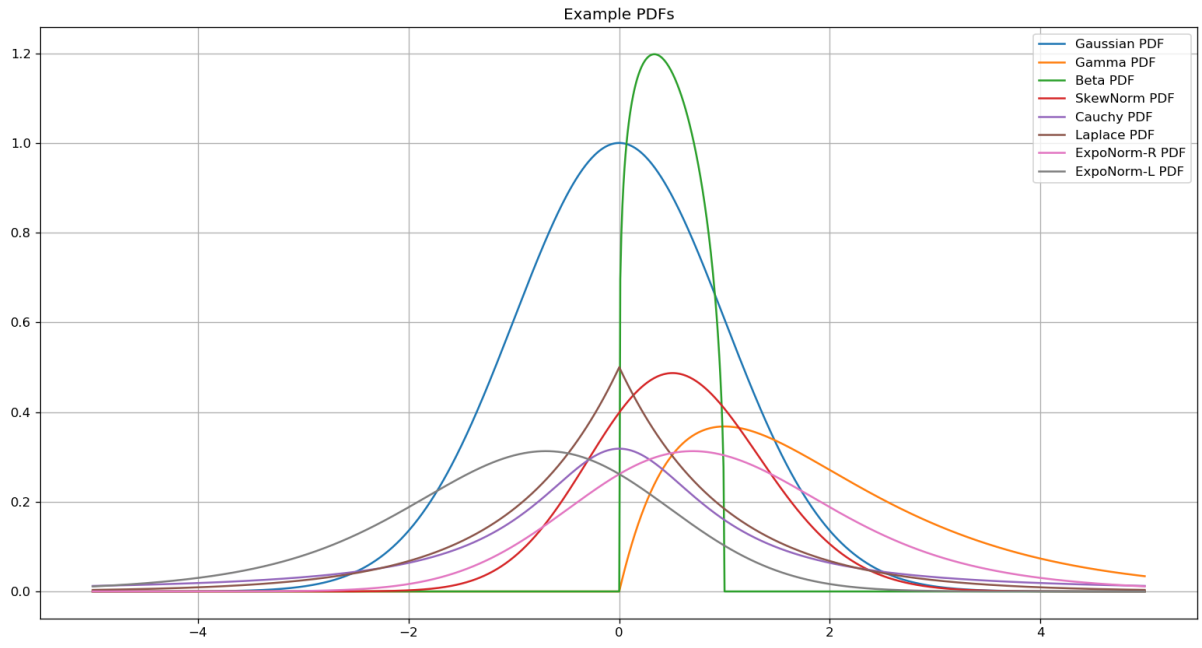


Figure B.1: Example PDFs

Each PDF was given three additional parameters to translate (l), scale (s) and skew (w) the function. The definition of this transform is given in Equation B.2.

$$\text{PDF}_{\text{Final}} = \text{PDF}\left(\frac{x}{w} - l\right) \cdot s \tag{B.2}$$

Equation B.2: PDF transformation

B.2.2. Bi-Modal Regression

Allowing the option for multi-modal parameters a comparison was made between singular PDF and double PDF regression.

The double PDF function is defined in Equation B.3.

$$\text{PDF}_{\text{Double}}\left(x,\{p_1\},\{p_2\},l_1,s_1,w_1,l_2,s_2,w_2\right)=\text{PDF}_{\text{Single}}\left(x,\{p_1\},l_1,s_1,w_1,\right)+\text{PDF}_{\text{Single}}\left(x,\{p_2\},l_2,s_2,w_2,\right) \tag{B.3}$$

Equation B.3: Double PDF definition

Given more parameters, it makes sense that the overall fit becomes better, on average however the double regression only showed a 1.4% improvement over the singular PDF regression. Therefore the presence of a secondary mode could not be asserted, except for a few edge cases, like 'Hips-ψ' and a few finger joint channels.

B.3. Edge-case 'Hips-ψ'

The 'Hips-ψ' channel is a special case, as compared to all other channels, and has 3 distinct modes at $[0°, \pm 90°]$. 'Hips-ψ' is the global yaw angle of the combined dancers and is the only parameter not local to each dancer individually. Both dancers start off by facing the audience at [Mode #1: $0°$] and then turn towards each other keeping this primary direction for most of the take [Mode #2&3: $\pm 90°$]. Being free to turn and face any desired direction, not bound by limits, it is also the only channel showcasing near uniform distribution over the entire $360°$ motion range. For a full analytical PDF to accurately describe this distribution a combination of a uniform distribution plus three Laplace distributions for each mode would probably be required. For general analytical motion modeling of the human body this dataset specific variable is however not of importance and can just be substituted by a uniform distribution to all the skeleton to face any desired direction.

B.4. Regression Parameters

Tables B.4 and B.1 presents all function arguments required to reconstruct the best-fit regression functions for each channel. The following table is intended for usage by future researchers, for the analytical modeling of the human body. The parameters presented here follow the convention as defined in Equations B.2 and B.3. [1]

[1] Caution: The following parameters are based on the regression of experimental data and are not ensured to comply with the requirement for PDFs that $CDF\left(\infty\right)=1$. An additional scaling factor should be applied to ensure this criteria, before using it in an analytical model.

R^2 Results

	Single							Double					Best					
Best-Fit Count Overall→	2	0	0	1	4	1	0	11	57	19	10	8	**Info:**	= Non-converging regression				
Best-Fit Count (Single||Double):→	10	1	0	30	41	13	18	13	59	23	10	8						
JointName ↓	Gamma PDF	Beta PDF	Gaussian PDF	SkewNorm PDF	ExpoNorm PDF	Cauchy PDF	Laplace PDF	Double Gaussian PDF	Double SkewNorm PDF	Double ExpoNorm PDF	Double Cauchy PDF	Double Laplace PDF	Best Single	Best Double	Single-Double Diff	Best R^2	Best Single	Best Double
Average →	57.31%	-8.95%	91.57%	95.55%	92.21%	91.78%	89.48%	96.05%	98.16%	92.23%	84.38%	92.91%	96.71%	98.12%	-1.40%	98.37%	SkewNorm PDF	Double SkewNorm PDF
Hips-x	97.99%	-0.68%	97.93%	98.12%	98.41%	99.47%	99.26%	99.69%	99.78%	96.85%	99.86%	99.82%	99.47%	99.86%	-0.40%	99.86%	Cauchy PDF	Double Cauchy PDF
Hips-y	38.76%	-0.36%	96.83%	97.00%	97.40%	99.57%	99.85%	99.69%	99.72%	97.33%	99.57%	99.93%	99.85%	99.93%	-0.08%	99.93%	Laplace PDF	Double Laplace PDF
Hips-z	21.33%	-722.34%	13.88%	27.03%	-84.17%	11.95%	9.65%			64.39%	-584.98%	10.30%	27.03%	64.39%	-37.36%	64.39%	SkewNorm PDF	Double ExpoNorm-L PDF
Spine-x	23.90%	6.00%	98.28%	98.37%	98.57%	99.46%	99.60%	99.88%	99.82%	96.68%	99.69%	99.84%	99.60%	99.92%	-0.32%	99.92%	Laplace PDF	Double SkewNorm PDF
Spine-y	99.44%	-10.62%	95.81%	99.48%	99.90%	95.30%	94.69%	99.57%	99.93%	99.48%	97.90%	97.94%	99.90%	99.93%	-0.03%	99.93%	ExpoNorm-R PDF	Double SkewNorm PDF
Spine-z	56.27%	-4.84%	97.83%	98.08%	98.39%	98.95%	99.63%	99.79%	99.87%	87.16%	99.13%	99.79%	99.63%	99.87%	-0.24%	99.87%	Laplace PDF	Double SkewNorm PDF
Spine1-x	8.92%	-9.12%	94.36%	94.42%	94.80%	98.16%	99.39%	99.79%	99.85%	87.57%	98.80%	99.59%	99.39%	99.85%	-0.46%	99.85%	Laplace PDF	Double SkewNorm PDF
Spine1-y	99.59%	-10.35%	99.53%	99.63%	99.72%	98.12%	98.28%	99.53%	99.94%	99.72%		99.31%	99.72%	99.94%	-0.22%	99.94%	ExpoNorm-R PDF	Double SkewNorm PDF
Spine1-z	97.34%		97.26%	97.48%	97.83%	99.08%	99.81%	98.80%	99.06%	99.83%	99.12%	99.84%	99.81%	99.84%	-0.03%	99.84%	Laplace PDF	Double Laplace PDF
Neck-x	38.53%	-3.95%	96.91%	97.09%	97.44%	99.10%	99.81%	99.77%	99.86%	90.20%	99.22%	99.88%	99.81%	99.88%	-0.07%	99.88%	Laplace PDF	Double Laplace PDF
Neck-y	99.56%	99.58%	99.51%	99.54%	99.54%	95.83%	96.07%	99.57%	99.87%	99.54%	98.19%	98.53%	99.58%	99.87%	-0.29%	99.87%	Beta PDF	Double SkewNorm PDF
Neck-z	2.72%	3.34%	95.96%	96.05%	96.11%	96.37%	98.30%	97.07%	99.67%	86.11%	97.54%	98.45%	98.30%	99.67%	-1.37%	99.67%	Laplace PDF	Double SkewNorm PDF
Head-x	98.61%	-7.77%	98.61%	98.62%	98.72%	98.93%	99.50%	99.88%	99.89%	97.72%	99.00%		99.50%	99.89%	-0.39%	99.89%	Laplace PDF	Double SkewNorm PDF
Head-y	2.28%		98.06%	98.75%	98.94%	98.23%	98.66%	99.92%	99.70%	99.92%	99.24%	98.66%	98.94%	99.92%	-0.98%	99.92%	ExpoNorm-L PDF	Double Gaussian PDF
Head-z		-14.19%	96.58%	96.72%	96.77%	96.56%	98.34%	99.65%	99.67%	96.77%	97.05%	98.37%	98.34%	99.67%	-1.33%	99.67%	Laplace PDF	Double SkewNorm PDF
LeftShoulder-x	99.38%	-6.78%	98.18%	99.48%	99.79%	97.39%	97.20%	99.11%	99.90%	99.88%		99.21%	99.79%	99.90%	-0.11%	99.90%	ExpoNorm-R PDF	Double SkewNorm PDF
LeftShoulder-y	96.83%	-8.76%	96.86%	99.48%	98.99%	94.48%	93.80%		99.73%	99.75%	98.96%	93.80%	99.48%	99.75%	-0.28%	99.75%	SkewNorm PDF	Double ExpoNorm-L PDF
LeftShoulder-z	99.08%	-8.13%	99.10%	99.73%	99.65%	96.83%	96.94%	99.10%	99.93%	87.99%	98.67%	98.58%	99.73%	99.93%	-0.20%	99.93%	SkewNorm PDF	Double SkewNorm PDF
LeftArm-x	-27.50%	-34.59%	97.68%	97.69%	97.68%	91.76%	91.54%	97.68%	99.85%	96.35%	34.22%	91.54%	97.69%	99.85%	-2.16%	99.85%	SkewNorm PDF	Double SkewNorm PDF
LeftArm-y	96.97%	-30.56%	95.96%	96.64%	95.95%	89.00%	87.61%	99.62%	99.86%	95.96%		97.28%	96.97%	99.86%	-2.89%	99.86%	Gamma PDF	Double SkewNorm PDF
LeftArm-z	92.40%		92.44%	98.39%	97.59%	89.99%	89.68%	99.67%	99.77%	99.35%		89.68%	98.39%	99.77%	-1.39%	99.77%	SkewNorm PDF	Double SkewNorm PDF
LeftForeArm-x	75.49%	-8.88%	87.08%	94.88%	98.08%	92.27%	91.22%	99.41%	99.80%	98.62%		98.73%	98.08%	99.80%	-1.72%	99.80%	ExpoNorm-R PDF	Double SkewNorm PDF
LeftForeArm-y		-35.37%	97.79%	98.70%	98.37%	94.99%	95.38%	99.56%	99.85%	98.37%			98.70%	99.85%	-1.15%	99.85%	SkewNorm PDF	Double SkewNorm PDF
LeftForeArm-z	-32.03%	-41.12%	88.17%	96.55%	97.23%	89.35%	89.10%	97.10%	97.57%	99.43%	92.06%		97.23%	99.43%	-2.19%	99.43%	ExpoNorm-L PDF	Double ExpoNorm-L PDF
LeftHand-x	97.92%	-22.80%	97.92%	97.93%	98.10%	98.20%	98.54%	99.09%	99.52%	71.96%	98.41%	98.54%	98.54%	99.52%	-0.98%	99.52%	Laplace PDF	Double SkewNorm PDF
LeftHand-y	-12.85%	-18.99%	96.10%	98.77%	99.49%	96.85%	96.80%	97.86%	99.86%	-18.17%	98.81%	96.80%	99.49%	99.86%	-0.37%	99.86%	ExpoNorm-L PDF	Double SkewNorm PDF
LeftHand-z	98.56%		98.57%	99.21%	99.52%	98.60%	98.07%	99.78%	99.70%	99.15%	99.63%	99.58%	99.52%	99.78%	-0.27%	99.78%	ExpoNorm-L PDF	Double Gaussian PDF
LeftHandThumb1-x	97.79%		97.70%	97.91%	97.88%	97.48%	97.34%	98.64%	99.67%	99.32%	99.38%	99.13%	97.91%	99.67%	-1.76%	99.67%	SkewNorm PDF	Double SkewNorm PDF
LeftHandThumb1-y	30.36%	51.97%	96.61%	96.65%	96.92%	99.34%	99.59%	99.49%	99.65%	99.05%		99.74%	99.59%	99.74%	-0.15%	99.74%	Laplace PDF	Double Laplace PDF
LeftHandThumb1-z	38.63%	10.62%	97.04%	97.28%	97.72%	99.22%	99.50%	98.68%		98.62%	99.45%	99.73%	99.50%	99.73%	-0.23%	99.73%	Laplace PDF	Double Laplace PDF
LeftHandThumb2-z	99.86%	56.41%	87.55%	99.28%	98.88%	87.64%	86.53%		99.88%	98.88%	92.92%	92.66%	99.86%	99.88%	-0.02%	99.88%	Gamma PDF	Double SkewNorm PDF
LeftHandThumb3-z	99.79%	15.59%	86.99%	99.27%	98.45%	87.05%	85.92%		99.97%	98.45%	92.32%	91.68%	99.79%	99.97%	-0.18%	99.97%	Gamma PDF	Double SkewNorm PDF
LeftHandIndex1-x	92.87%		92.91%	95.71%	96.80%	95.72%	96.06%	99.25%	99.29%	96.80%		98.07%	96.80%	99.29%	-2.50%	99.29%	ExpoNorm-R PDF	Double SkewNorm PDF
LeftHandIndex1-y	50.35%	35.29%	77.14%	77.58%	78.43%	80.22%	80.07%	80.98%	82.16%	78.43%		68.53%	80.22%	82.16%	-1.94%	82.16%	Cauchy PDF	Double SkewNorm PDF
LeftHandIndex1-z	7.27%	-0.52%	82.46%	96.10%	98.89%	83.79%	81.60%	88.80%	95.10%	68.68%	91.19%	90.44%	98.89%	95.10%	3.80%	98.89%	ExpoNorm-L PDF	Double SkewNorm PDF
LeftHandIndex2-x	-20.65%	-22.80%	72.32%	94.07%	97.14%	69.63%	67.90%	80.46%	94.07%	94.03%		97.15%	97.14%	97.15%	0.00%	97.15%	ExpoNorm-L PDF	Double Laplace PDF
LeftHandIndex3-x	-8.72%	-10.95%	75.07%	94.59%	97.78%	72.97%	71.25%	82.47%	94.59%	97.78%	72.86%		97.78%	97.78%	0.00%	97.78%	ExpoNorm-L PDF	Double ExpoNorm-L PDF
LeftHandMiddle1-x	62.45%		91.24%	94.43%	95.97%	95.28%	95.86%	98.95%	99.13%	95.97%		97.40%	95.97%	99.13%	-3.16%	99.13%	ExpoNorm-L PDF	Double SkewNorm PDF
LeftHandMiddle1-y	31.50%	9.59%	74.31%	74.63%	49.16%	73.61%	69.86%	75.37%	76.45%	75.43%		63.29%	74.63%	76.45%	-1.82%	76.45%	SkewNorm PDF	Double SkewNorm PDF
LeftHandMiddle1-z	42.50%		93.99%	94.22%	94.95%	97.98%	97.94%	97.70%	98.35%	98.80%	98.76%	99.29%	97.98%	99.29%	-1.31%	99.29%	Cauchy PDF	Double Laplace PDF
LeftHandMiddle2-x	-13.70%	-5.46%	67.51%	83.72%	93.31%	71.91%	68.51%	74.02%	83.72%	97.88%			93.31%	97.88%	-4.57%	97.88%	ExpoNorm-L PDF	Double ExpoNorm-L PDF
LeftHandMiddle3-x	-3.16%	-6.87%	70.67%	85.30%	67.11%	74.77%	71.61%	76.71%		98.83%	88.07%	94.08%	85.30%	98.83%	-13.54%	98.83%	SkewNorm PDF	Double ExpoNorm-L PDF
LeftHandRing1-x	1.97%	-13.14%	92.41%	95.47%	96.47%	95.38%	96.10%	98.99%	96.99%	83.78%	99.20%	97.28%	96.47%	99.20%	-2.73%	99.20%	ExpoNorm-L PDF	Double Cauchy PDF
LeftHandRing1-y	23.66%		76.24%	76.24%	51.26%	76.43%	46.81%	77.20%	78.20%	68.97%		60.95%	76.43%	78.20%	-1.77%	78.20%	Cauchy PDF	Double SkewNorm PDF
LeftHandRing1-z	35.57%		94.02%	96.09%	96.77%	96.85%	96.12%	98.08%	99.73%	96.69%	99.37%	99.40%	96.85%	99.73%	-2.88%	99.73%	Cauchy PDF	Double SkewNorm PDF
LeftHandRing2-x	-17.83%	-19.96%	66.07%	88.67%	96.62%	70.86%	68.21%	73.57%	88.67%	95.73%	83.53%	96.62%	96.62%	96.62%	0.00%	96.62%	ExpoNorm-L PDF	Double Laplace PDF
LeftHandRing3-x	-5.73%	-9.19%	70.02%	89.70%	69.91%	74.24%	71.66%	76.77%	92.62%	98.51%	86.20%	71.69%	89.70%	98.51%	-8.81%	98.51%	SkewNorm PDF	Double ExpoNorm-L PDF
LeftHandPinky1-x	96.44%		95.06%	96.88%	97.59%	97.19%	97.61%	99.29%	98.12%	98.58%	99.16%	97.61%	97.61%	99.29%	-1.68%	99.29%	Laplace PDF	Double Gaussian PDF
LeftHandPinky1-y	28.24%	3.66%	90.08%	90.72%	44.07%		34.85%	99.05%	97.74%	57.51%	99.38%	50.76%	90.72%	99.88%	-9.16%	99.88%	SkewNorm PDF	Double Cauchy PDF
LeftHandPinky1-z	99.85%	23.45%	85.91%	99.26%	99.31%	85.58%	84.27%	94.20%		83.45%		91.65%	99.85%	94.20%	5.66%	99.85%	Gamma PDF	Double Gaussian PDF
LeftHandPinky2-x	-23.25%	-25.00%	75.22%	96.11%	97.53%	78.37%	77.10%	84.26%	96.12%	98.16%	88.71%	77.10%	97.53%	98.16%	-0.63%	98.16%	ExpoNorm-L PDF	Double ExpoNorm-L PDF
LeftHandPinky3-x	-10.30%	-11.89%	77.84%	96.43%	98.22%	80.86%	79.60%	85.95%	98.88%	98.69%	91.70%		98.22%	98.88%	-0.67%	98.88%	ExpoNorm-L PDF	Double SkewNorm PDF
RightShoulder-x	98.60%	-11.54%	98.61%	99.35%	99.64%	98.07%	97.97%	99.12%	99.35%	99.64%		99.36%	99.64%	99.64%	0.00%	99.64%	ExpoNorm-L PDF	Double ExpoNorm-L PDF
RightShoulder-y	98.40%	50.96%	98.42%	99.13%	98.87%	94.46%	93.56%	98.48%	99.88%	99.01%	98.98%	99.25%	99.13%	99.88%	-0.76%	99.88%	SkewNorm PDF	Double SkewNorm PDF
RightShoulder-z	99.29%	-9.01%	98.31%	99.48%	99.21%	95.89%	95.97%	99.93%	99.94%	99.75%		98.34%	99.48%	99.94%	-0.47%	99.94%	SkewNorm PDF	Double SkewNorm PDF
RightArm-x		-48.02%	97.78%	98.10%	97.89%	93.61%	93.64%	98.93%	99.70%	98.40%			98.10%	99.70%	-1.60%	99.70%	SkewNorm PDF	Double SkewNorm PDF
RightArm-y	97.16%	-24.83%	96.53%	97.16%	96.66%	90.66%	89.76%	99.03%	99.71%	87.36%	98.27%	98.81%	97.16%	99.71%	-2.55%	99.71%	Gamma PDF	Double SkewNorm PDF
RightArm-z	98.58%	-30.11%	94.33%	99.14%	98.99%	93.36%	93.26%	99.90%	99.14%	96.75%		97.60%	99.14%	99.90%	-0.76%	99.90%	SkewNorm PDF	Double SkewNorm PDF
RightForeArm-x	92.62%	-20.03%	92.66%	96.59%	98.61%	96.13%	95.45%	95.24%	96.59%	99.48%		98.13%	98.61%	99.48%	-0.87%	99.48%	ExpoNorm-L PDF	Double ExpoNorm-L PDF
RightForeArm-y		-23.91%	97.46%	99.06%	98.54%	94.59%	94.43%	97.46%	99.82%	99.79%		94.43%	99.06%	99.82%	-0.76%	99.82%	SkewNorm PDF	Double SkewNorm PDF
RightForeArm-z	63.90%	32.97%	85.68%	96.94%	98.53%	88.09%	86.86%	97.11%	99.32%	99.28%	93.56%	86.86%	98.53%	99.32%	-0.79%	99.32%	ExpoNorm-R PDF	Double SkewNorm PDF
RightHand-x	58.41%	-26.89%	94.37%	94.41%	94.88%	98.08%	98.26%	96.21%	97.45%	94.60%	99.03%	98.35%	98.26%	99.03%	-0.77%	99.03%	Laplace PDF	Double Cauchy PDF
RightHand-y	-18.21%	-19.36%	97.46%	99.14%	99.29%	97.28%	97.46%	99.81%	99.14%	99.69%	-16.00%	97.46%	99.29%	99.81%	-0.52%	99.81%	ExpoNorm-L PDF	Double Gaussian PDF
RightHand-z			97.90%	98.75%	97.89%	97.02%	95.54%	99.15%	99.44%	97.87%	99.52%	95.54%	98.75%	99.52%	-0.77%	99.52%	SkewNorm PDF	Double Cauchy PDF
RightHandThumb1-x	37.21%		98.86%	99.07%	99.37%	98.70%	98.19%	99.81%	99.27%	99.76%	99.58%	99.48%	99.37%	99.81%	-0.44%	99.81%	ExpoNorm-R PDF	Double Gaussian PDF
RightHandThumb1-y		78.80%	96.29%	96.39%	96.70%	98.25%	99.07%	97.27%	99.82%	99.59%		99.09%	99.07%	99.82%	-0.75%	99.82%	Laplace PDF	Double SkewNorm PDF
RightHandThumb1-z	25.60%	43.14%	95.86%	96.61%	97.49%	98.93%	99.09%	98.61%	99.89%	99.35%		99.44%	99.09%	99.89%	-0.80%	99.89%	Laplace PDF	Double SkewNorm PDF
RightHandThumb2-z	2.19%	-1.50%	87.20%	99.49%	98.33%	86.66%	85.77%	93.37%		96.61%		91.98%	99.49%	96.61%	2.88%	99.49%	SkewNorm PDF	Double ExpoNorm-L PDF
RightHandThumb3-z	-2.60%	-4.58%	86.24%	99.45%	98.02%	85.71%	84.80%	92.61%	99.45%	0.00%	92.05%	97.81%	99.45%	99.45%	0.00%	99.45%	SkewNorm PDF	Double SkewNorm PDF
RightHandIndex1-x	99.11%	-16.62%	99.11%	99.23%	99.26%	97.75%	97.65%	99.41%	99.60%	98.66%	99.41%	97.61%	99.26%	99.60%	-0.33%	99.60%	ExpoNorm-L PDF	Double SkewNorm PDF
RightHandIndex1-y	43.61%	38.86%	80.63%	81.45%	82.80%	86.12%	86.96%	91.31%	92.81%	82.92%	92.87%	84.22%	86.96%	92.87%	-5.91%	92.87%	Laplace PDF	Double Cauchy PDF
RightHandIndex1-z	97.77%	35.43%	80.90%	96.93%	76.28%	81.94%	80.12%	92.53%		79.54%		90.10%	97.77%	92.53%	5.24%	97.77%	Gamma PDF	Double Gaussian PDF
RightHandIndex2-x	98.18%	-20.95%	92.45%	97.45%	96.38%	89.84%	89.07%	96.52%	99.63%	99.79%			98.18%	99.79%	-1.61%	99.79%	Gamma PDF	Double ExpoNorm-L PDF
RightHandIndex3-x	98.50%	-8.47%	93.16%	97.74%	96.94%	91.05%	90.20%	96.82%	99.65%	99.88%	91.06%	90.20%	98.50%	99.88%	-1.39%	99.88%	Gamma PDF	Double ExpoNorm-R PDF
RightHandMiddle1-x	98.17%		98.19%	98.58%	98.57%	97.20%	97.21%	99.10%	98.58%	98.87%	97.43%	98.22%	98.58%	99.10%	-0.52%	99.10%	SkewNorm PDF	Double Gaussian PDF
RightHandMiddle1-y	20.67%	19.18%	71.94%	72.24%	49.23%	75.30%	45.67%			64.16%		57.42%	75.30%	64.16%	11.14%	75.30%	Cauchy PDF	Double ExpoNorm-L PDF
RightHandMiddle1-z	8.26%	8.81%	89.88%	93.23%	85.63%	94.59%	93.93%	97.11%	99.75%	91.61%	98.16%	97.89%	94.59%	99.75%	-5.15%	99.75%	Cauchy PDF	Double SkewNorm PDF
RightHandMiddle2-x	95.82%		87.99%	98.24%	93.53%	84.51%	83.53%	92.72%	98.24%	99.39%		90.85%	98.24%	99.39%	-1.15%	99.39%	SkewNorm PDF	Double ExpoNorm-R PDF
RightHandMiddle3-x	96.66%	-4.23%	89.72%	98.44%	94.54%	86.68%	85.64%	93.87%	98.44%	99.74%	92.61%	92.22%	98.44%	99.74%	-1.30%	99.74%	SkewNorm PDF	Double ExpoNorm-R PDF
RightHandRing1-x	98.76%	-18.56%	98.77%	98.84%	98.89%	97.80%	97.95%	99.13%	98.84%	99.67%		97.95%	98.89%	99.67%	-0.78%	99.67%	ExpoNorm-L PDF	Double ExpoNorm-R PDF
RightHandRing1-y	3.44%	25.91%	89.99%	91.23%	48.32%		34.14%	98.32%	98.70%	57.30%		55.36%	91.23%	98.70%	-7.47%	98.70%	SkewNorm PDF	Double SkewNorm PDF
RightHandRing1-z	27.08%	10.40%	93.45%	97.49%	98.73%	95.40%	94.92%	95.97%	99.77%	97.83%	99.05%	98.63%	98.73%	99.77%	-1.04%	99.77%	ExpoNorm-L PDF	Double SkewNorm PDF
RightHandRing2-x	95.13%	-23.69%	90.19%	96.90%	89.77%	86.91%	86.51%	94.25%	96.92%	93.00%		86.51%	96.90%	96.92%	-0.02%	96.92%	SkewNorm PDF	Double ExpoNorm-R PDF
RightHandRing3-x	96.08%	-5.82%	91.70%	97.29%	94.30%	89.13%	88.62%	95.14%	97.29%	99.58%	89.13%	88.62%	97.29%	99.58%	-2.29%	99.58%	SkewNorm PDF	Double ExpoNorm-R PDF
RightHandPinky1-x	99.43%		99.38%	99.47%	99.55%	98.16%	98.28%	99.60%	99.47%	99.83%		99.31%	99.55%	99.83%	-0.27%	99.83%	ExpoNorm-R PDF	Double SkewNorm PDF
RightHandPinky1-y	16.58%	1.75%	86.15%	90.54%	54.67%	92.54%	55.88%	96.48%		73.13%	98.62%	74.19%	92.54%	98.62%	-6.08%	98.62%	Cauchy PDF	Double Cauchy PDF
RightHandPinky1-z	0.76%	-2.80%	82.76%	98.23%	99.73%	83.30%	81.80%	89.63%	99.08%	99.73%	91.64%	90.79%	99.73%	99.73%	0.00%	99.73%	ExpoNorm-L PDF	Double ExpoNorm-L PDF
RightHandPinky2-x	98.07%	-34.95%	93.00%	97.66%	96.28%	89.30%	88.75%	97.24%	97.66%	99.03%		94.95%	98.07%	99.03%	-0.96%	99.03%	Gamma PDF	Double ExpoNorm-R PDF
RightHandPinky3-x	98.87%	43.52%	93.65%	98.31%	97.43%	91.12%	90.52%	97.50%	99.77%	99.66%	91.13%	90.52%	98.87%	99.77%	-0.90%	99.77%	Gamma PDF	Double SkewNorm PDF
LeftUpLeg-x	37.09%	-14.79%	97.18%	99.09%	99.64%	97.19%	97.03%	99.72%	99.44%	95.93%		97.03%	99.64%	99.72%	-0.08%	99.72%	ExpoNorm-R PDF	Double Gaussian PDF
LeftUpLeg-y	95.06%	-20.28%	95.10%	98.59%	99.04%	95.16%	95.01%	99.66%	98.59%	82.52%		99.02%	99.04%	99.66%	-0.62%	99.66%	ExpoNorm-L PDF	Double Gaussian PDF
LeftUpLeg-z	99.12%	-3.57%	98.91%	99.21%	99.43%	98.53%	97.93%	99.72%	99.59%	99.44%	99.53%	99.58%	99.43%	99.72%	-0.29%	99.72%	ExpoNorm-R PDF	Double Gaussian PDF
LeftLeg-x	94.90%	-2.21%	94.91%	96.23%	97.21%	95.79%	94.91%	98.14%	96.93%	97.20%	97.92%	97.32%	97.21%	98.14%	-0.93%	98.14%	ExpoNorm-L PDF	Double Gaussian PDF
LeftLeg-y	96.07%	-29.33%	91.49%	97.14%	93.68%	84.59%	83.14%		99.37%	90.15%	95.39%	83.30%	97.14%	99.37%	-2.23%	99.37%	SkewNorm PDF	Double SkewNorm PDF
LeftLeg-z	99.50%		98.98%	99.57%	99.73%	97.67%	96.96%	99.72%	99.66%	99.92%			99.73%	99.92%	-0.19%	99.92%	ExpoNorm-R PDF	Double ExpoNorm-R PDF
LeftFoot-x	99.36%	-9.05%	99.37%	99.85%	99.74%	96.30%	96.37%	99.37%	99.93%	99.92%		98.45%	99.85%	99.93%	-0.08%	99.93%	SkewNorm PDF	Double SkewNorm PDF
LeftFoot-y			95.33%	95.52%	96.33%	96.10%	95.66%	98.55%	99.45%	95.89%		96.54%	96.33%	99.45%	-3.12%	99.45%	ExpoNorm-R PDF	Double SkewNorm PDF
LeftFoot-z	99.74%		99.07%	99.82%	99.86%	97.37%	97.47%	99.64%	99.97%	94.29%	98.56%	97.47%	99.86%	99.97%	-0.10%	99.97%	ExpoNorm-R PDF	Double SkewNorm PDF
LeftToeBase-x	14.10%		98.65%	98.71%	68.28%	99.34%	99.15%	99.89%	99.91%	85.14%		99.83%	99.34%	99.91%	-0.58%	99.91%	Cauchy PDF	Double SkewNorm PDF
LeftToeBase-y	44.57%		93.69%	94.78%	96.50%	98.37%	97.53%	95.01%	96.76%	99.45%	99.76%	99.07%	98.37%	99.76%	-1.39%	99.76%	Cauchy PDF	Double Cauchy PDF
LeftToeBase-z	99.26%	13.48%	99.22%	99.31%	99.41%	98.91%	98.97%	99.97%	99.98%	99.18%	99.41%	99.75%	99.41%	99.98%	-0.56%	99.98%	ExpoNorm-R PDF	Double SkewNorm PDF
RightUpLeg-x	-1.06%	-18.31%	94.97%	98.68%	99.04%	95.30%	94.84%	99.63%	99.86%	98.77%	98.88%	98.58%	99.04%	99.86%	-0.82%	99.86%	ExpoNorm-L PDF	Double SkewNorm PDF
RightUpLeg-y	98.49%		95.30%	99.02%	98.84%	93.66%	92.87%	99.09%	99.77%	98.66%	98.30%	98.17%	99.02%	99.77%	-0.76%	99.77%	SkewNorm PDF	Double SkewNorm PDF
RightUpLeg-z	99.04%	-12.39%	99.03%	99.06%	99.19%	99.02%	98.82%	99.80%	99.89%	99.85%	99.88%	99.14%	99.19%	99.89%	-0.71%	99.89%	ExpoNorm-L PDF	Double SkewNorm PDF
RightLeg-x	96.38%		96.40%	97.37%	98.16%	99.21%	98.41%	98.25%	98.78%	98.73%	99.89%	99.55%	99.21%	99.89%	-0.68%	99.89%	Cauchy PDF	Double Cauchy PDF
RightLeg-y		-32.27%	94.70%	96.75%	95.07%	87.71%	86.82%	97.83%	99.60%	97.16%		86.82%	96.75%	99.60%	-2.85%	99.60%	SkewNorm PDF	Double SkewNorm PDF
RightLeg-z		-9.61%	99.01%	99.06%	99.16%	99.08%	98.72%	99.84%	99.84%	99.20%		98.86%	99.16%	99.84%	-0.68%	99.84%	ExpoNorm-R PDF	Double SkewNorm PDF
RightFoot-x	99.61%		98.79%	99.72%	99.56%	96.34%	96.52%	99.88%	99.90%	89.74%			99.72%	99.90%	-0.18%	99.90%	SkewNorm PDF	Double SkewNorm PDF
RightFoot-y	0.93%	-17.40%	96.65%	96.86%	97.52%	96.87%	96.16%		99.46%	97.52%	96.98%	97.80%	97.52%	99.46%	-1.93%	99.46%	ExpoNorm-R PDF	Double SkewNorm PDF
RightFoot-z	99.84%	-16.83%	99.62%	99.85%	99.86%	96.89%	96.72%	99.62%	99.86%	99.62%	98.81%	98.94%	99.86%	99.86%	0.00%	99.86%	ExpoNorm-R PDF	Double SkewNorm PDF
RightToeBase-x	21.58%	58.16%	94.70%	94.99%	95.84%	99.43%	98.39%	99.53%	99.59%	99.67%	99.70%	99.33%	99.43%	99.70%	-0.27%	99.70%	Cauchy PDF	Double Cauchy PDF
RightToeBase-y	22.00%	84.39%	92.04%	92.07%	93.68%	98.46%	97.84%	96.37%	97.62%	99.83%	99.48%	99.24%	98.46%	99.83%	-1.37%	99.83%	Cauchy PDF	Double ExpoNorm-L PDF
RightToeBase-z	97.78%	10.33%	97.80%	98.17%	98.62%	99.34%	99.45%	97.80%	99.82%	98.81%	99.66%	99.82%	99.45%	99.82%	-0.37%	99.82%	Laplace PDF	Double SkewNorm PDF

B

JointName	Best-Single	Best-Single [Arguments]
Hips-ϕ	Cauchy PDF	$(+1.26E{-}1\ +2.99E{+}0\ +7.63E{+}0)$
Hips-θ	Laplace PDF	$(+1.14E{-}1\ +2.24E{+}0\ +8.88E{+}0)$
Hips-ψ	SkewNorm PDF	$(-1.13E{+}1\ +4.61E{-}1\ +8.75E{-}2\ +3.42E{+}2)$
Spine-ϕ	Laplace PDF	$(-1.43E{-}2\ +3.34E{+}0\ +6.18E{+}0)$
Spine-θ	ExpoNorm-R PDF	$(+2.63E{+}0\ -8.28E{-}1\ +3.08E{+}0\ +6.57E{+}0)$
Spine-ψ	Laplace PDF	$(+5.71E{-}2\ +1.84E{+}0\ +1.13E{+}1)$
Spine1-ϕ	Laplace PDF	$(-4.10E{-}2\ +1.41E{+}0\ +1.45E{+}1)$
Spine1-θ	ExpoNorm-R PDF	$(+7.48E{-}1\ -4.27E{-}1\ +1.71E{+}0\ +1.15E{+}1)$
Spine1-ψ	Laplace PDF	$(+2.31E{-}2\ +5.01E{+}0\ +4.17E{+}0)$
Neck-ϕ	Laplace PDF	$(+1.08E{-}1\ +1.90E{+}0\ +1.08E{+}1)$
Neck-θ	Beta PDF	$(+9.45E{+}0\ +1.28E{+}1\ -4.14E{-}1\ +8.70E{+}0\ +5.77E{-}3)$
Neck-ψ	Laplace PDF	$(-6.95E{-}2\ +1.25E{+}0\ +1.70E{+}1)$
Head-ϕ	Laplace PDF	$(+1.03E{-}1\ +1.71E{+}0\ +1.23E{+}1)$
Head-θ	ExpoNorm-L PDF	$(+1.21E{+}0\ -5.38E{-}1\ +1.57E{+}0\ +1.25E{+}1)$
Head-ψ	Laplace PDF	$(-6.97E{-}2\ +1.26E{+}0\ +1.69E{+}1)$
LeftShoulder-ϕ	ExpoNorm-R PDF	$(+1.53E{+}0\ +4.95E{-}1\ +2.90E{+}0\ +6.92E{+}0)$
LeftShoulder-θ	SkewNorm PDF	$(-3.98E{+}0\ +1.21E{+}0\ +1.16E{+}0\ +1.71E{+}1)$
LeftShoulder-ψ	SkewNorm PDF	$(-1.97E{+}0\ +6.63E{-}1\ +1.20E{+}0\ +1.66E{+}1)$
LeftArm-ϕ	SkewNorm PDF	$(+6.10E{-}1\ -1.39E{+}0\ +6.52E{-}1\ +3.13E{+}1)$
LeftArm-θ	Gamma PDF	$(+7.03E{+}0\ -8.32E{+}0\ +5.19E{-}1\ -9.29E{-}2)$
LeftArm-ψ	SkewNorm PDF	$(-5.21E{+}0\ +1.16E{-}1\ +4.24E{-}1\ +4.71E{+}1)$
LeftForeArm-ϕ	ExpoNorm-R PDF	$(+5.19E{+}0\ -2.63E{-}1\ +3.98E{+}0\ +4.68E{+}0)$
LeftForeArm-θ	SkewNorm PDF	$(-2.18E{+}0\ +3.62E{-}1\ +4.81E{-}1\ +4.16E{+}1)$
LeftForeArm-ψ	ExpoNorm-L PDF	$(+3.67E{+}0\ +1.36E{-}1\ +1.77E{+}0\ +1.17E{+}1)$
LeftHand-ϕ	Laplace PDF	$(+1.30E{-}1\ +1.02E{+}0\ +1.98E{+}1)$
LeftHand-θ	ExpoNorm-L PDF	$(+2.20E{+}0\ +1.47E{+}0\ +2.44E{+}0\ +8.07E{+}0)$
LeftHand-ψ	ExpoNorm-L PDF	$(-1.31E{+}0\ -1.92E{+}0\ +2.46E{+}0\ +7.56E{+}0)$
LeftHandThumb1-ϕ	SkewNorm PDF	$(+1.66E{+}0\ -7.13E{-}1\ +2.78E{+}0\ +6.58E{+}0)$
LeftHandThumb1-θ	Laplace PDF	$(+1.63E{-}1\ +7.05E{+}0\ +2.84E{+}0)$
LeftHandThumb1-ψ	Laplace PDF	$(+3.14E{-}1\ +4.33E{+}0\ +4.64E{+}0)$
LeftHandThumb2-ψ	Gamma PDF	$(+1.29E{+}0\ -7.57E{-}3\ +1.39E{-}1\ +3.51E{-}1)$
LeftHandThumb3-ψ	Gamma PDF	$(+1.29E{+}0\ -6.98E{-}3\ +2.84E{-}1\ +1.72E{-}1)$
LeftHandIndex1-ϕ	ExpoNorm-L PDF	$(+2.46E{+}0\ -1.43E{+}0\ +2.69E{+}0\ +7.13E{+}0)$
LeftHandIndex1-θ	Cauchy PDF	$(+1.12E{-}2\ +1.96E{+}1\ +9.78E{-}1)$
LeftHandIndex1-ψ	ExpoNorm-L PDF	$(+2.80E{+}1\ +1.86E{-}1\ +1.47E{+}2\ +1.23E{-}1)$
LeftHandIndex2-ϕ	ExpoNorm-L PDF	$(+2.70E{+}2\ +2.42E{-}1\ +1.52E{+}2\ +1.38E{-}1)$
LeftHandIndex3-ϕ	ExpoNorm-L PDF	$(+1.94E{+}2\ +6.58E{-}2\ +2.09E{+}2\ +9.83E{-}2)$
LeftHandMiddle1-ϕ	ExpoNorm-L PDF	$(+2.67E{+}0\ -1.31E{+}0\ +2.81E{+}0\ +6.72E{+}0)$
LeftHandMiddle1-θ	SkewNorm PDF	$(-3.48E{+}0\ +5.61E{-}1\ +1.32E{+}1\ +1.19E{+}0)$
LeftHandMiddle1-ψ	Cauchy PDF	$(+3.48E{-}1\ +1.61E{+}1\ +1.26E{+}0)$
LeftHandMiddle2-ϕ	ExpoNorm-L PDF	$(+2.67E{+}2\ +2.00E{-}2\ +2.31E{+}2\ +7.67E{-}2)$
LeftHandMiddle3-ϕ	SkewNorm PDF	$(-9.08E{+}1\ +3.03E{-}3\ +1.86E{+}0\ +6.93E{+}0)$
LeftHandRing1-ϕ	ExpoNorm-L PDF	$(+2.42E{+}0\ -1.02E{+}0\ +2.53E{+}0\ +7.64E{+}0)$
LeftHandRing1-θ	Cauchy PDF	$(-3.66E{-}1\ +2.60E{+}1\ +8.51E{-}1)$
LeftHandRing1-ψ	Cauchy PDF	$(+3.65E{-}1\ +1.09E{+}1\ +2.08E{+}0)$
LeftHandRing2-ϕ	ExpoNorm-L PDF	$(+7.16E{+}2\ -9.77E{-}2\ +4.78E{+}2\ +4.12E{-}2)$
LeftHandRing3-ϕ	SkewNorm PDF	$(-1.77E{+}2\ +4.14E{-}3\ +1.20E{+}0\ +1.44E{+}1)$
LeftHandPinky1-ϕ	Laplace PDF	$(-2.78E{-}1\ +1.28E{+}0\ +1.61E{+}1)$
LeftHandPinky1-θ	SkewNorm PDF	$(-2.16E{+}0\ +2.20E{-}1\ +2.70E{+}1\ +5.41E{-}1)$
LeftHandPinky1-ψ	Gamma PDF	$(+1.16E{+}0\ -7.67E{-}4\ +2.06E{-}1\ +2.38E{-}1)$
LeftHandPinky2-ϕ	ExpoNorm-L PDF	$(+1.52E{+}2\ +4.43E{-}1\ +7.98E{+}1\ +2.65E{-}1)$
LeftHandPinky3-ϕ	ExpoNorm-L PDF	$(+9.46E{+}1\ +3.75E{-}1\ +9.60E{+}1\ +2.17E{-}1)$
RightShoulder-ϕ	ExpoNorm-L PDF	$(+1.30E{+}0\ +5.80E{-}1\ +2.75E{+}0\ +7.18E{+}0)$
RightShoulder-θ	SkewNorm PDF	$(-2.36E{+}0\ +1.05E{+}0\ +1.15E{+}0\ +1.76E{+}1)$
RightShoulder-ψ	SkewNorm PDF	$(+2.53E{+}0\ -3.38E{-}1\ +1.05E{+}0\ +1.90E{+}1)$
RightArm-ϕ	SkewNorm PDF	$(+1.64E{+}0\ +1.91E{-}2\ +4.52E{-}1\ +4.46E{+}1)$
RightArm-θ	Gamma PDF	$(+1.32E{+}1\ -1.09E{+}1\ +3.39E{-}1\ +1.44E{-}1)$
RightArm-ψ	SkewNorm PDF	$(+4.32E{+}0\ -1.43E{-}1\ +4.66E{-}1\ +4.24E{+}1)$
RightForeArm-ϕ	ExpoNorm-L PDF	$(+3.03E{+}0\ +3.20E{-}1\ +2.75E{+}0\ +6.75E{+}0)$
RightForeArm-θ	SkewNorm PDF	$(-2.89E{+}0\ +2.60E{-}1\ +5.13E{-}1\ +3.86E{+}1)$
RightForeArm-ψ	ExpoNorm-R PDF	$(+6.49E{+}0\ -2.49E{-}1\ +3.21E{+}0\ +6.27E{+}0)$
RightHand-ϕ	Laplace PDF	$(-6.62E{-}2\ +9.20E{-}1\ +2.04E{+}1)$
RightHand-θ	ExpoNorm-L PDF	$(+1.61E{+}0\ +3.45E{+}0\ +2.09E{+}0\ +9.42E{+}0)$
RightHand-ψ	SkewNorm PDF	$(+2.30E{+}0\ -1.09E{+}0\ +7.85E{-}1\ +2.24E{+}1)$
RightHandThumb1-ϕ	ExpoNorm-R PDF	$(-1.07E{+}0\ -1.30E{+}0\ +5.30E{+}0\ +3.52E{+}0)$
RightHandThumb1-θ	Laplace PDF	$(+2.42E{-}1\ +9.12E{+}0\ +2.23E{+}0)$
RightHandThumb1-ψ	Laplace PDF	$(-4.47E{-}1\ +6.52E{+}0\ +3.01E{+}0)$
RightHandThumb2-ψ	SkewNorm PDF	$(-4.23E{+}1\ -2.41E{-}3\ +4.21E{+}0\ +4.80E{+}0)$
RightHandThumb3-ψ	SkewNorm PDF	$(-8.02E{+}1\ -1.54E{-}3\ +2.10E{+}0\ +9.55E{+}0)$
RightHandIndex1-ϕ	ExpoNorm-L PDF	$(-7.15E{-}1\ -1.22E{+}0\ +1.49E{+}0\ +1.29E{+}1)$
RightHandIndex1-θ	Laplace PDF	$(+2.52E{-}1\ +1.42E{+}1\ +1.15E{+}0)$
RightHandIndex1-ψ	Gamma PDF	$(+9.63E{-}1\ +2.83E{-}9\ +2.42E{-}1\ +2.05E{-}1)$
RightHandIndex2-ϕ	Gamma PDF	$(+1.99E{+}0\ -1.44E{-}1\ +8.97E{-}1\ +5.45E{-}2)$
RightHandIndex3-ϕ	Gamma PDF	$(+1.91E{+}0\ -1.17E{-}1\ +4.79E{-}1\ +1.04E{-}1)$
RightHandMiddle1-ϕ	SkewNorm PDF	$(-1.83E{+}0\ +1.11E{+}0\ +8.91E{-}1\ +2.13E{+}1)$
RightHandMiddle1-θ	Cauchy PDF	$(-7.58E{-}1\ +3.10E{+}1\ +6.16E{-}1)$
RightHandMiddle1-ψ	Cauchy PDF	$(-7.06E{-}1\ +1.70E{+}1\ +1.22E{+}0)$
RightHandMiddle2-ϕ	SkewNorm PDF	$(+1.51E{+}5\ -3.82E{-}7\ +4.97E{-}1\ +4.06E{+}1)$
RightHandMiddle3-ϕ	SkewNorm PDF	$(+8.47E{+}4\ -1.88E{-}7\ +9.82E{-}1\ +2.01E{+}1)$
RightHandRing1-ϕ	ExpoNorm-L PDF	$(+6.79E{-}1\ -1.52E{+}0\ +1.40E{+}0\ +1.37E{+}1)$
RightHandRing1-θ	SkewNorm PDF	$(-2.66E{+}0\ +2.26E{-}1\ +2.26E{+}1\ +6.25E{-}1)$
RightHandRing1-ψ	ExpoNorm-L PDF	$(+3.50E{+}0\ -4.48E{-}1\ +1.75E{+}1\ +1.10E{+}0)$
RightHandRing2-ϕ	SkewNorm PDF	$(+4.99E{+}2\ +1.74E{-}4\ +4.26E{-}1\ +4.84E{+}1)$
RightHandRing3-ϕ	SkewNorm PDF	$(+3.19E{+}2\ +1.09E{-}4\ +8.38E{-}1\ +2.41E{+}1)$
RightHandPinky1-ϕ	ExpoNorm-R PDF	$(+7.25E{-}1\ +3.53E{-}1\ +1.31E{+}0\ +1.49E{+}1)$
RightHandPinky1-θ	Cauchy PDF	$(-6.26E{-}1\ +3.32E{+}1\ +5.18E{-}1)$
RightHandPinky1-ψ	ExpoNorm-L PDF	$(+3.99E{+}1\ +3.33E{-}1\ +1.26E{+}2\ +1.65E{-}1)$
RightHandPinky2-ϕ	Gamma PDF	$(+2.51E{+}0\ -3.11E{-}1\ +9.77E{-}1\ +4.93E{-}2)$
RightHandPinky3-ϕ	Gamma PDF	$(+2.21E{+}0\ -1.95E{-}1\ +5.64E{-}1\ +8.67E{-}2)$
LeftUpLeg-ϕ	ExpoNorm-R PDF	$(+1.87E{+}0\ -3.16E{-}1\ +2.26E{+}0\ +8.71E{+}0)$
LeftUpLeg-θ	ExpoNorm-L PDF	$(+2.45E{+}0\ -1.00E{+}0\ +1.98E{+}0\ +1.02E{+}1)$
LeftUpLeg-ψ	ExpoNorm-R PDF	$(+1.09E{+}0\ -2.68E{-}1\ +2.85E{+}0\ +6.51E{+}0)$
LeftLeg-ϕ	ExpoNorm-L PDF	$(+1.99E{+}0\ -5.73E{-}1\ +5.00E{+}0\ +3.42E{+}0)$
LeftLeg-θ	SkewNorm PDF	$(+7.81E{+}0\ -1.98E{-}1\ +4.24E{-}1\ +4.91E{+}1)$
LeftLeg-ψ	ExpoNorm-R PDF	$(+1.17E{+}0\ -1.84E{-}1\ +1.52E{+}0\ +1.24E{+}1)$
LeftFoot-ϕ	SkewNorm PDF	$(-1.81E{+}0\ +3.83E{-}1\ +9.00E{-}1\ +2.24E{+}1)$
LeftFoot-θ	ExpoNorm-R PDF	$(+1.36E{+}0\ -2.06E{+}0\ +1.89E{+}0\ +1.01E{+}1)$
LeftFoot-ψ	ExpoNorm-R PDF	$(+1.17E{+}0\ +9.29E{-}1\ +2.43E{+}0\ +8.25E{+}0)$
LeftToeBase-ϕ	Cauchy PDF	$(-1.53E{-}1\ +2.89E{+}1\ +8.70E{-}1)$
LeftToeBase-θ	Cauchy PDF	$(-3.01E{-}1\ +7.45E{+}0\ +2.66E{+}0)$
LeftToeBase-ψ	ExpoNorm-R PDF	$(+7.77E{-}1\ -5.74E{-}1\ +5.48E{+}0\ +3.55E{+}0)$
RightUpLeg-ϕ	ExpoNorm-L PDF	$(+2.81E{+}0\ -5.40E{-}1\ +2.55E{+}0\ +7.65E{+}0)$
RightUpLeg-θ	SkewNorm PDF	$(-4.24E{+}0\ +2.43E{-}1\ +5.40E{-}1\ +3.64E{+}1)$
RightUpLeg-ψ	ExpoNorm-L PDF	$(+7.55E{-}1\ -1.33E{-}1\ +1.81E{+}0\ +1.02E{+}1)$
RightLeg-ϕ	Cauchy PDF	$(+6.58E{-}1\ +5.87E{+}0\ +3.43E{+}0)$
RightLeg-θ	SkewNorm PDF	$(+5.71E{+}0\ -9.18E{-}3\ +4.39E{-}1\ +4.77E{+}1)$
RightLeg-ψ	ExpoNorm-R PDF	$(+7.32E{-}1\ -4.45E{-}1\ +1.80E{+}0\ +9.99E{+}0)$
RightFoot-ϕ	SkewNorm PDF	$(+2.23E{+}0\ -5.28E{-}1\ +8.98E{-}1\ +2.24E{+}1)$
RightFoot-θ	ExpoNorm-R PDF	$(+1.32E{+}0\ -2.39E{+}0\ +1.98E{+}0\ +9.57E{+}0)$
RightFoot-ψ	ExpoNorm-R PDF	$(-7.92E{-}1\ -2.14E{+}0\ +1.56E{+}0\ +1.28E{+}1)$
RightToeBase-ϕ	Cauchy PDF	$(+9.45E{-}2\ +2.17E{+}1\ +9.23E{-}1)$
RightToeBase-θ	Cauchy PDF	$(-1.18E{-}1\ +6.53E{+}0\ +2.93E{+}0)$
RightToeBase-ψ	Laplace PDF	$(-7.91E{-}3\ +3.86E{+}0\ +5.19E{+}0)$

Table B.1: PDF regression arguments - Single

JointName	Best-Double	Best-Double [Arguments]
Hips-ϕ	Double Cauchy PDF	(+1.28E+0 +7.26E−1 +6.07E+0 −3.91E−2 +2.71E+0 +6.47E+0)
Hips-θ	Double Laplace PDF	(+1.66E−1 +1.50E+0 +1.03E+1 −1.28E−2 +8.19E−1 +5.70E+0)
Hips-ψ	Double ExpoNorm-L PDF	(+2.87E−2 −3.54E+0 +7.83E−1 +1.75E−1 −8.94E+1 +3.68E+0 +4.22E−1 +1.83E+1)
Spine-ϕ	Double SkewNorm PDF	(−8.92E−1 −1.70E+0 +5.09E−1 +1.24E+0 +1.04E+1 +5.77E−1 +1.50E+0 +4.75E+0)
Spine-θ	Double SkewNorm PDF	(+7.39E+0 +1.28E+0 −2.41E−1 +3.95E−1 +3.01E+1 −3.87E−1 +6.25E−1 +1.31E+1)
Spine-ψ	Double SkewNorm PDF	(+5.99E+0 −2.64E+0 −1.58E−1 +6.29E−1 +1.62E+1 +2.03E−1 +7.53E−1 +1.29E+1)
Spine1-ϕ	Double SkewNorm PDF	(−1.38E+0 −1.23E+0 +4.45E−1 +6.89E−1 +5.87E+0 +5.67E−1 +6.21E−1 +2.60E+1)
Spine1-θ	Double SkewNorm PDF	(+4.29E+0 −8.90E−1 −1.70E−1 +2.95E−1 +2.31E+1 +4.16E−1 +8.77E−1 +1.52E+1)
Spine1-ψ	Double Laplace PDF	(+1.20E−1 +2.07E+0 +4.70E+0 −4.87E−2 +3.01E+0 +3.73E+0)
Neck-ϕ	Double Laplace PDF	(+1.23E−1 +1.85E+0 +1.10E+1 −1.01E+0 +2.14E−1 +1.07E+0)
Neck-θ	Double SkewNorm PDF	(+5.59E+0 −2.76E+0 −1.35E−1 +5.85E−1 +2.05E+1 −2.84E−1 +5.53E−1 +1.45E+1)
Neck-ψ	Double SkewNorm PDF	(−7.82E+0 +7.02E+0 +8.22E−2 +4.96E−1 +2.18E+1 −1.59E−1 +4.78E−1 +2.01E+1)
Head-ϕ	Double SkewNorm PDF	(+2.86E+0 −3.99E+0 −1.30E−2 +6.34E−1 +1.46E+1 +3.00E−1 +6.96E−1 +1.53E+1)
Head-θ	Double Gaussian PDF	(−1.81E+0 +2.16E−1 −7.83E+0 −7.53E+0 +2.89E−1 +2.18E+1)
Head-ψ	Double SkewNorm PDF	(+7.78E−1 −8.02E−1 −6.83E−1 +5.20E−1 +4.03E+0 +3.97E−1 +8.09E−1 +2.27E+1)
LeftShoulder-ϕ	Double SkewNorm PDF	(+3.86E+0 −5.97E−1 +4.89E−1 +5.16E−1 +1.76E+1 +1.23E+0 +1.35E+0 +8.13E+0)
LeftShoulder-θ	Double ExpoNorm-L PDF	(−2.44E+0 −2.98E−2 −1.65E+0 +2.22E+0 +3.77E+0 −2.57E+0 +2.04E+0 +5.67E+0)
LeftShoulder-ψ	Double SkewNorm PDF	(+2.87E+0 −3.05E+0 +4.21E−2 +9.36E−1 +1.05E+1 +7.96E−2 +7.74E−1 +1.32E+1)
LeftArm-ϕ	Double SkewNorm PDF	(+1.42E+0 +2.43E+0 −3.79E+0 +4.07E−1 +1.76E+1 −1.25E+0 +4.37E−1 +2.92E+1)
LeftArm-θ	Double SkewNorm PDF	(−5.38E+0 +8.59E−1 +1.23E+0 +3.99E−1 +4.22E+1 −6.02E−1 +2.57E−1 +1.28E+1)
LeftArm-ψ	Double SkewNorm PDF	(−1.90E+0 +2.43E+0 +3.92E−2 +4.38E−1 +1.95E+1 −1.61E+0 +2.55E−1 +4.56E+1)
LeftForeArm-ϕ	Double SkewNorm PDF	(−6.62E−1 +2.58E+0 +1.37E+0 +3.62E−1 +3.12E+1 +6.83E−3 +7.07E−1 +1.18E+1)
LeftForeArm-θ	Double SkewNorm PDF	(+2.96E+0 +8.57E−1 −5.37E−1 +3.65E−1 +2.74E+1 −1.78E+0 +3.92E−1 +2.56E+1)
LeftForeArm-ψ	Double ExpoNorm-L PDF	(−6.77E−2 +7.56E+0 +1.68E+0 −5.11E−1 +4.90E+0 −2.43E−1 +4.10E+0 +5.56E+0)
LeftHand-ϕ	Double SkewNorm PDF	(+1.56E+0 −2.29E+1 −2.71E−1 +6.39E−1 +2.26E+1 +1.58E−1 +1.28E−1 +4.12E+1)
LeftHand-θ	Double SkewNorm PDF	(−3.37E+0 +1.80E+0 −2.22E−1 +5.27E−1 +3.05E+1 +1.62E+0 +3.98E−1 −9.47E+0)
LeftHand-ψ	Double Gaussian PDF	(+1.87E+0 +1.96E−1 +1.78E+1 +8.16E+0 +5.32E−1 +7.91E+0)
LeftHandThumb1-ϕ	Double SkewNorm PDF	(+3.44E+0 −1.06E+1 −5.72E−1 +2.19E+0 +6.75E+0 −1.72E−1 +5.78E−1 +8.70E+0)
LeftHandThumb1-θ	Double Laplace PDF	(+2.38E−1 +4.18E+0 +3.54E+0 +6.12E−2 +3.31E+0 +1.68E+0)
LeftHandThumb1-ψ	Double Laplace PDF	(+6.63E−1 +2.32E+0 +3.19E+0 +9.94E−2 +2.27E+0 +5.65E+0)
LeftHandThumb2-ϕ	Double SkewNorm PDF	(+3.44E+0 +6.40E+4 +1.82E−1 +2.03E+0 +1.72E+0 −2.84E−7 +3.57E+0 +4.71E+0)
LeftHandThumb3-ψ	Double SkewNorm PDF	(+6.00E+0 +4.20E+1 +2.49E−1 +1.37E+0 +9.13E+0 +2.17E−3 +2.20E+0 +3.50E+0)
LeftHandIndex1-ϕ	Double SkewNorm PDF	(+4.61E+0 −7.95E+0 −1.96E−1 +4.43E−1 +1.69E+1 +2.75E−1 +4.66E−1 +2.68E+1)
LeftHandIndex1-θ	Double SkewNorm PDF	(−4.67E+0 −3.41E+0 +6.06E−1 +7.55E+0 +1.06E+0 +5.60E−1 +2.05E+0 +5.16E+0)
LeftHandIndex1-ψ	Double SkewNorm PDF	(+2.12E+3 −7.26E+0 +8.12E−1 −1.78E+0 +2.46E−1 +5.15E−2 +5.24E+0 +2.99E+0)
LeftHandIndex2-ϕ	Double Laplace PDF	(+5.38E−3 −4.46E+1 +3.72E+1 −7.14E−3 +4.51E+1 +3.72E+1)
LeftHandIndex3-ϕ	Double ExpoNorm-L PDF	(+1.94E+2 −8.28E+2 +6.58E−2 +2.09E+2 +9.83E−2 +1.35E+2 −1.60E+0 −8.79E+2)
LeftHandMiddle1-ϕ	Double SkewNorm PDF	(+5.91E+0 −7.26E+0 −3.12E−1 +4.81E−1 +1.63E+1 +1.97E−1 +4.29E−1 +2.81E+1)
LeftHandMiddle1-θ	Double SkewNorm PDF	(−6.52E+0 −4.07E+0 +5.42E−1 +9.93E+0 +9.39E−1 +6.29E−1 +3.42E+0 +2.43E+0)
LeftHandMiddle1-ψ	Double Laplace PDF	(−1.36E+0 +8.30E−1 +5.16E+0 +3.04E−1 +1.16E+1 +1.43E+0)
LeftHandMiddle2-ϕ	Double ExpoNorm-L PDF	(+6.65E+0 +3.10E+7 +5.40E−1 +5.60E+0 +8.51E−1 −3.64E−2 +1.19E+7 +1.37E−6)
LeftHandMiddle3-ϕ	Double ExpoNorm-L PDF	(+4.58E+1 +1.77E+1 +2.73E+0 +3.07E+1 +4.85E−1 +3.00E−1 +3.87E+1 +1.52E−1)
LeftHandRing1-ϕ	Double Cauchy PDF	(+1.53E−1 +1.93E+0 −9.20E+0 −3.15E+0 +5.46E−1 +8.26E+0)
LeftHandRing1-θ	Double SkewNorm PDF	(+5.64E+0 −1.27E+0 −8.93E−1 +7.23E+0 +9.23E−1 +3.32E−1 +6.64E+0 +1.73E+0)
LeftHandRing1-ψ	Double SkewNorm PDF	(+5.90E+0 −1.55E+0 −3.44E−1 +3.63E+0 +2.40E+0 +7.89E−1 +1.81E+0 +6.56E+0)
LeftHandRing2-ϕ	Double Laplace PDF	(+6.73E−3 −5.22E+1 +2.95E+1 −5.98E−3 +5.28E+1 +2.95E+1)
LeftHandRing3-ϕ	Double ExpoNorm-L PDF	(+2.39E+1 −2.93E−2 +1.00E−1 +3.66E+1 +3.77E−1 −2.80E+0 +5.36E−1 −9.95E+0)
LeftHandPinky1-ϕ	Double Gaussian PDF	(−2.60E+0 +3.25E−1 +6.90E+0 −9.83E+0 +2.62E−1 +2.20E+1)
LeftHandPinky1-θ	Double Cauchy PDF	(−3.56E−1 +1.43E+1 +9.56E−1 −9.13E−1 +5.16E+1 +1.85E−1)
LeftHandPinky1-ψ	Double Gaussian PDF	(+4.72E+0 +1.74E+0 +2.86E+0 +1.33E+0 +2.51E+0 +1.06E+0)
LeftHandPinky2-ϕ	Double ExpoNorm-L PDF	(−2.80E−2 +6.24E+1 +2.70E+0 +1.97E−1 +5.99E+0 +2.77E−1 +3.01E+1 +6.59E−1)
LeftHandPinky3-ϕ	Double SkewNorm PDF	(−1.50E+5 −8.53E−1 −5.87E−7 +6.71E−1 +2.49E+1 −6.55E−1 +7.57E−1 +4.62E+0)
RightShoulder-ϕ	Double ExpoNorm-L PDF	(+1.37E−2 +1.30E+0 +4.96E+0 +3.43E+0 +3.40E+0 +5.80E−1 +2.75E+0 +7.18E+0)
RightShoulder-θ	Double SkewNorm PDF	(−8.63E−1 −5.22E+0 +1.80E+0 +1.50E+0 +8.94E+0 +3.10E−1 +5.24E−1 +1.23E+1)
RightShoulder-ψ	Double SkewNorm PDF	(+4.80E+0 −2.94E+0 +6.90E−1 +6.10E−1 +1.30E+1 +6.73E−1 +1.08E+0 +1.12E+1)
RightArm-ϕ	Double SkewNorm PDF	(−6.27E+0 −8.61E−1 +1.40E+0 +1.94E−1 +5.93E+1 +1.25E+0 +3.70E−1 +2.30E+1)
RightArm-θ	Double SkewNorm PDF	(−2.42E+0 −2.02E+0 +1.27E+0 +4.36E−1 +3.46E+1 +4.36E−1 +3.45E−1 +1.43E+1)
RightArm-ψ	Double Gaussian PDF	(+3.57E+1 +1.78E−1 +2.84E+1 +9.19E+0 +2.34E−1 +1.26E+1)
RightForeArm-ϕ	Double ExpoNorm-L PDF	(−2.99E−2 +7.09E+0 +8.93E−1 +1.06E+0 +7.21E+0 +3.18E+0 +3.16E+0 +3.57E+0)
RightForeArm-θ	Double SkewNorm PDF	(+2.89E+0 −1.62E+0 −5.90E−1 +3.57E−1 +2.09E+1 −3.78E−1 +4.36E−1 +2.84E+1)
RightForeArm-ψ	Double SkewNorm PDF	(+5.52E+0 +1.14E+1 −2.87E−3 +2.05E−1 +6.25E+1 −1.24E−1 +3.10E−1 +2.29E+1)
RightHand-ϕ	Double Cauchy PDF	(+1.28E−2 +1.06E+0 +1.93E+1 −1.09E+0 +4.76E−1 +2.96E+0)
RightHand-θ	Double Gaussian PDF	(−4.90E+1 +2.83E−1 +1.88E+1 −3.96E+1 +3.17E−1 +8.20E+0)
RightHand-ψ	Double Cauchy PDF	(−3.21E+0 +9.21E−1 +6.12E+0 −4.75E−1 +1.30E+0 +1.19E+1)
RightHandThumb1-ϕ	Double Gaussian PDF	(+3.65E+0 −2.24E+0 +4.12E+0 +1.63E+0 +3.19E+0 +5.29E+0)
RightHandThumb1-θ	Double SkewNorm PDF	(+2.55E+0 −1.83E+0 −8.58E−2 +3.29E+0 +8.16E−1 +7.86E−1 +4.34E+0 +3.92E+0)
RightHandThumb1-ψ	Double SkewNorm PDF	(−1.75E+0 +1.94E+0 −2.47E−1 +2.91E+0 +1.69E+0 −7.44E−1 +2.30E+0 +6.34E+0)
RightHandThumb2-ψ	Double ExpoNorm-L PDF	(+2.26E+0 +2.74E−2 +5.17E−1 +1.11E+1 +9.18E−1 +1.32E+0 +2.83E+0 +3.51E+0)
RightHandThumb3-ψ	Double SkewNorm PDF	(−4.87E+1 −8.02E+1 −3.89E+0 0 −2.05E+0 −1.54E−3 +2.10E+0 +9.55E+0)
RightHandIndex1-ϕ	Double SkewNorm PDF	(+2.77E+0 −4.46E+0 +8.22E−2 +8.24E−1 +1.71E+1 −1.76E−2 +3.70E−1 +1.41E+1)
RightHandIndex1-θ	Double Cauchy PDF	(+3.09E−1 +1.20E+1 +1.52E+0 +3.17E+0 +3.15E+1 +8.56E−2)
RightHandIndex1-ψ	Double Gaussian PDF	(+4.02E+0 +1.84E+0 +2.64E+0 +9.47E−1 +3.03E+0 +7.66E−1)
RightHandIndex2-ϕ	Double ExpoNorm-L PDF	(+2.98E+2 +2.12E+0 +2.60E−1 +8.89E+1 −9.95E−2 +1.53E+0 +1.16E+0 −9.94E+0)
RightHandIndex3-ϕ	Double ExpoNorm-R PDF	(+1.72E+0 +2.10E+2 +1.56E+0 +1.88E+0 +5.10E+0 +2.03E−1 +1.22E+2 +8.79E−2)
RightHandMiddle1-ϕ	Double Gaussian PDF	(+9.26E+0 +5.89E−1 +1.40E+1 +3.87E+0 −1.55E−1 +4.95E+0)
RightHandMiddle1-θ	Double ExpoNorm-L PDF	(−6.10E−2 −2.89E−2 +2.79E−1 +5.45E+0 +9.97E−1 +1.89E−1 +6.25E+0 +1.63E+0)
RightHandMiddle1-ψ	Double SkewNorm PDF	(−1.23E+1 +2.54E+0 −9.27E−4 +5.86E+0 +1.22E+0 −9.03E−1 +2.65E+0 +4.79E+0)
RightHandMiddle2-ϕ	Double ExpoNorm-R PDF	(+2.03E+0 +2.75E+2 +2.16E+0 +8.68E−1 +9.75E+0 +9.45E−2 +1.19E+2 +9.87E−2)
RightHandMiddle3-ϕ	Double ExpoNorm-R PDF	(+1.13E+0 +1.68E+2 +2.41E+0 +1.14E+0 +5.00E+0 +9.64E−2 +1.41E+2 +1.02E−1)
RightHandRing1-ϕ	Double ExpoNorm-R PDF	(+1.70E+0 +2.60E+0 −1.08E+0 +1.26E+0 −6.17E+0 +2.07E+0 +2.35E+0 +5.20E+0)
RightHandRing1-θ	Double SkewNorm PDF	(−2.47E+0 −2.26E+0 +3.89E−1 +6.92E+0 +1.85E+0 −4.53E−2 +2.20E+1 +2.92E−1)
RightHandRing1-ψ	Double SkewNorm PDF	(−5.56E+0 −3.25E+0 +5.28E−2 +2.25E+0 +1.72E+0 +2.04E−1 +2.63E+0 +6.02E+0)
RightHandRing2-ϕ	Double SkewNorm PDF	(+8.23E+5 +1.23E+1 +4.14E−3 +4.25E−1 +4.83E+1 −1.74E+0 +9.42E−1 +4.38E−3)
RightHandRing3-ϕ	Double ExpoNorm-R PDF	(+2.67E+0 +3.33E+2 +2.04E+0 +2.39E+0 +5.14E+0 +5.61E−2 +2.20E+2 +3.55E−2)
RightHandPinky1-ϕ	Double ExpoNorm-R PDF	(+5.67E−1 +2.91E−2 +3.03E−1 +8.62E−1 +1.85E+1 +1.32E+0 +3.89E−1 +1.04E+1)
RightHandPinky1-θ	Double Cauchy PDF	(−1.02E+0 +3.23E+1 +2.18E−1 −8.08E−1 +1.18E+1 +1.12E+0)
RightHandPinky1-ψ	Double ExpoNorm-L PDF	(+3.99E+1 −2.52E+1 +3.33E−1 +1.26E+2 +1.65E−1 +1.20E+1 0 −3.38E+0)
RightHandPinky2-ϕ	Double ExpoNorm-R PDF	(−1.91E+0 +2.69E+2 +1.40E+0 +9.61E−1 +1.46E+1 +5.69E−1 +5.10E+1 +1.34E−1)
RightHandPinky3-ϕ	Double SkewNorm PDF	(+2.49E+0 −1.40E+5 +4.31E−2 +9.23E−1 +2.40E+1 +6.41E−7 −3.58E−1 +6.27E+0)
LeftUpLeg-ϕ	Double Gaussian PDF	(+1.90E+1 +1.99E−1 +1.99E+1 +5.15E+0 +4.11E−1 −9.54E+0)
LeftUpLeg-θ	Double Gaussian PDF	(+6.55E−1 +2.58E−1 −9.04E+0 −1.52E+1 +2.26E−1 +2.51E+1)
LeftUpLeg-ψ	Double Gaussian PDF	(+1.37E+1 +9.14E−2 +2.14E+1 +3.46E+0 +7.97E−1 +7.46E+0)
LeftLeg-ϕ	Double Gaussian PDF	(−1.99E+0 +1.12E+0 −4.47E+0 −1.52E+1 +1.83E−1 +1.22E+1)
LeftLeg-θ	Double SkewNorm PDF	(−6.90E−1 +2.58E+0 +2.15E+0 +4.93E−1 +2.40E+1 −4.47E−1 +4.51E−1 +1.87E+1)
LeftLeg-ψ	Double ExpoNorm-R PDF	(−4.96E−1 +1.26E+1 +9.81E−2 +1.05E+0 +1.47E+1 −1.33E+0 +1.05E+0 +4.16E+0)
LeftFoot-ϕ	Double SkewNorm PDF	(+1.16E+0 −1.18E+0 −5.61E−1 +9.51E−1 +1.44E+1 −9.39E−1 +4.48E−1 +1.42E+1)
LeftFoot-θ	Double SkewNorm PDF	(+9.51E+0 −3.27E+0 −7.45E−1 +2.72E−1 +4.44E+1 −7.33E−2 +5.91E−1 +1.39E+1)
LeftFoot-ψ	Double SkewNorm PDF	(−5.18E+0 +2.01E+0 +1.41E+0 +1.83E−1 +1.36E+1 +3.35E−1 +1.02E+0 +1.71E+1)
LeftToeBase-ϕ	Double SkewNorm PDF	(−1.29E+0 +5.39E−1 +3.90E−1 +1.09E+1 +8.25E−1 −3.83E−1 +7.35E+0 +1.67E+0)
LeftToeBase-θ	Double Cauchy PDF	(−2.97E−1 +7.45E+0 +2.24E+0 −1.38E+0 +7.96E−1 +6.24E+0)
LeftToeBase-ψ	Double SkewNorm PDF	(−1.29E+0 −1.01E+0 +7.04E−1 +1.81E+0 +7.47E+0 +4.91E−1 +2.07E+0 +3.22E+0)
RightUpLeg-ϕ	Double SkewNorm PDF	(−2.93E+0 −1.04E+0 +2.95E−1 +4.68E−1 +1.24E+1 −5.88E−2 +5.19E−1 +2.66E+1)
RightUpLeg-θ	Double SkewNorm PDF	(+2.23E+0 −9.52E+0 +1.12E−1 +2.20E−1 +1.36E+1 +1.14E−1 +4.83E−1 +3.53E+1)
RightUpLeg-ψ	Double SkewNorm PDF	(+1.68E+0 −1.03E+1 −8.06E−1 +8.93E−1 +1.63E+1 +5.21E−2 +1.71E−1 +2.96E+1)
RightLeg-ϕ	Double Cauchy PDF	(+1.50E+0 +3.82E+0 +2.13E+0 +1.61E−1 +2.91E+0 +4.05E+0)
RightLeg-θ	Double SkewNorm PDF	(−1.70E+0 −6.89E+0 +2.23E+0 +2.92E−1 +3.06E+1 +8.30E−2 +3.39E−1 −3.35E+1)
RightLeg-ψ	Double SkewNorm PDF	(+2.42E+0 −4.62E+0 −3.85E−1 +7.23E−1 +1.77E+1 +3.79E−1 +2.59E−1 +2.40E+1)
RightFoot-ϕ	Double SkewNorm PDF	(+2.42E+0 −1.75E+0 +6.49E−1 +4.21E−1 +1.48E+1 +4.39E−1 +9.76E−1 +1.42E+1)
RightFoot-θ	Double SkewNorm PDF	(−5.87E+0 −1.42E+0 +1.26E+0 +1.15E−1 +3.78E+1 −3.44E−1 +1.04E+0 +1.52E+1)
RightFoot-ψ	Double SkewNorm PDF	(+2.10E+2 +1.48E+0 −9.68E−1 −1.66E−2 +4.13E−1 −1.50E+0 +9.55E−1 +2.07E+1)
RightToeBase-ϕ	Double Cauchy PDF	(−2.01E−1 +3.81E+1 +9.69E−1 −6.26E−1 −1.99E+1 +7.94E−1)
RightToeBase-θ	Double ExpoNorm-L PDF	(+1.22E+0 −1.88E+0 −6.02E−1 +5.74E+0 +1.38E+0 −1.48E+0 +2.07E+0 +5.72E+0)
RightToeBase-ψ	Double SkewNorm PDF	(−2.48E+0 −2.15E+0 +6.15E−1 +1.22E+0 +1.10E+1 +5.67E−1 +1.53E+0 +3.87E+0)

Table B.2: PDF regression arguments - Double

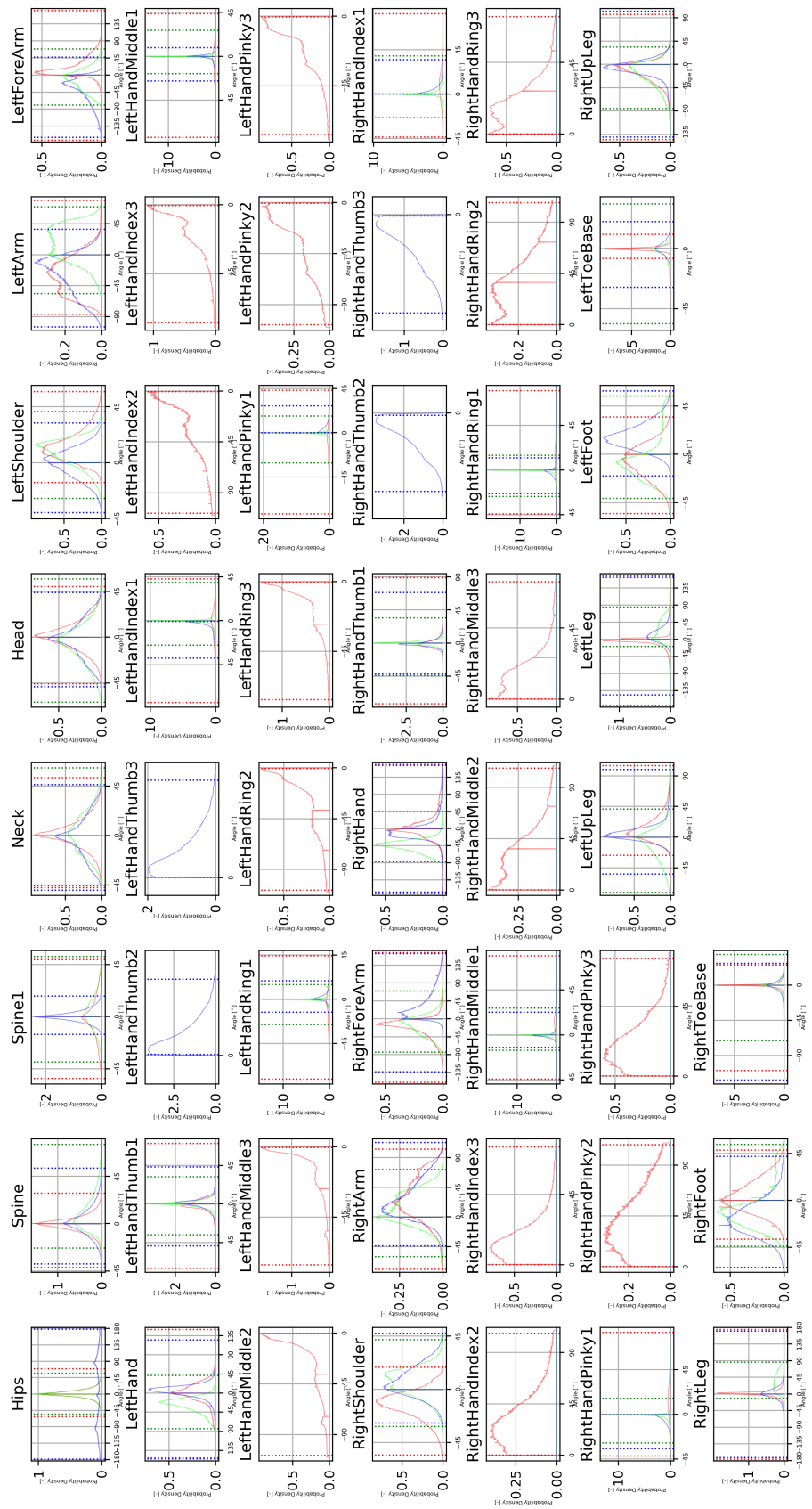


Figure B.2: Parameter Distributions: Histogram
Red: Roll (ϕ) | Green: Pitch (θ) | Blue: Yaw (ψ)
Continuous lines: Distributions | Dotted lines: Limits

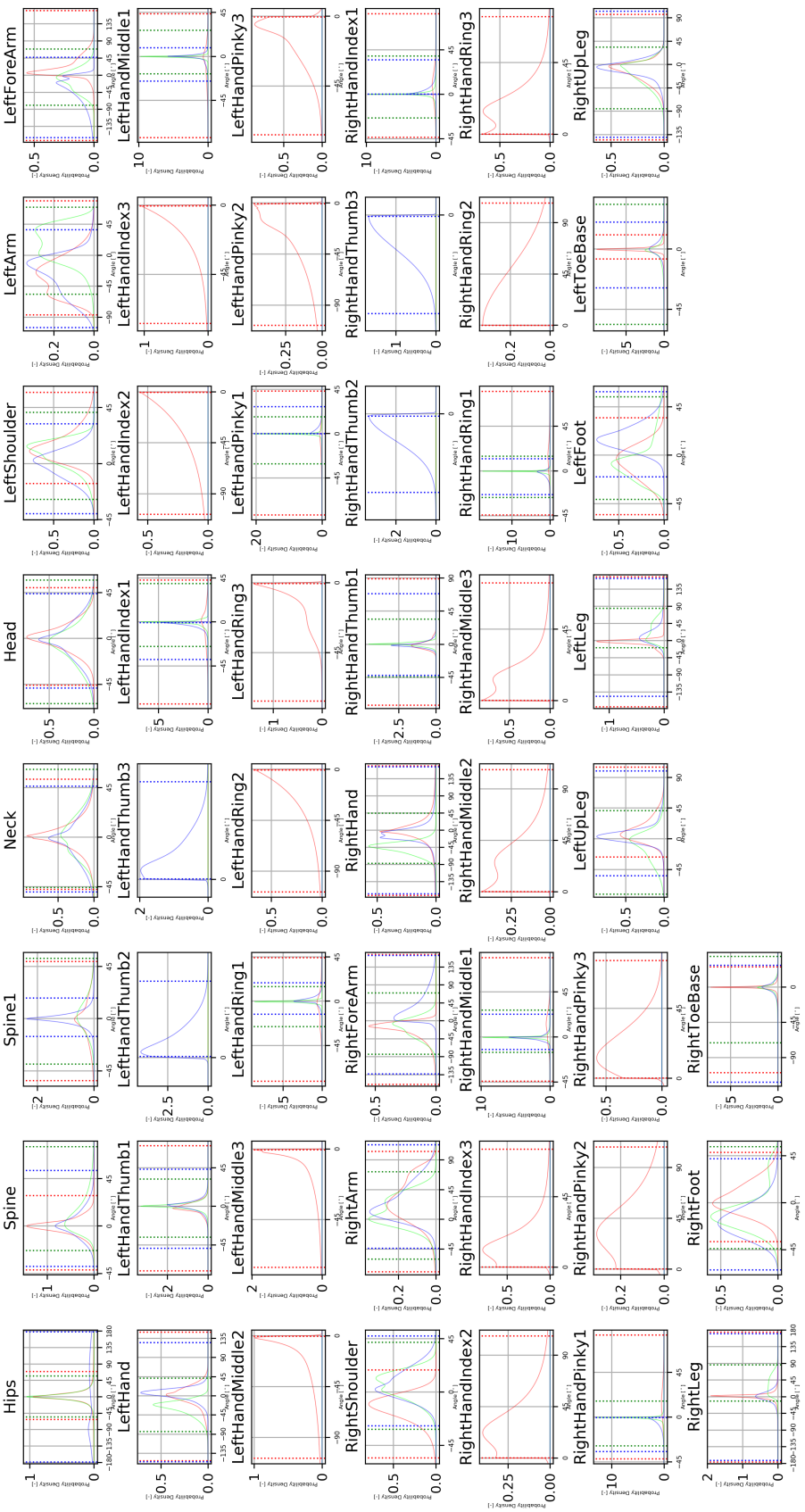


Figure B.3: Parameter Distributions: Best Fit PDFs
Red: Roll (ϕ) | Green: Pitch (θ) | Blue: Yaw (ψ)
Continuous lines: PDFs | Dotted lines: Limits

C

BHPO GUI

The following appendix shows screenshots of the user interface created for the *DanceNet-BHPO* framework.
The legend for reading the graphs is as follows:

- Extrema (Top & Bottom):
  - Red: Maximum value
  - Blue: Minimum value
- Data:
  - Red: Raw data - Every Iteration
  - Light Green: Exponentially filtered data with factor 0.9
  - Dark Green: Optimal regression, based on a variety of template functions
  - Blue: Average Data - Per Epoch

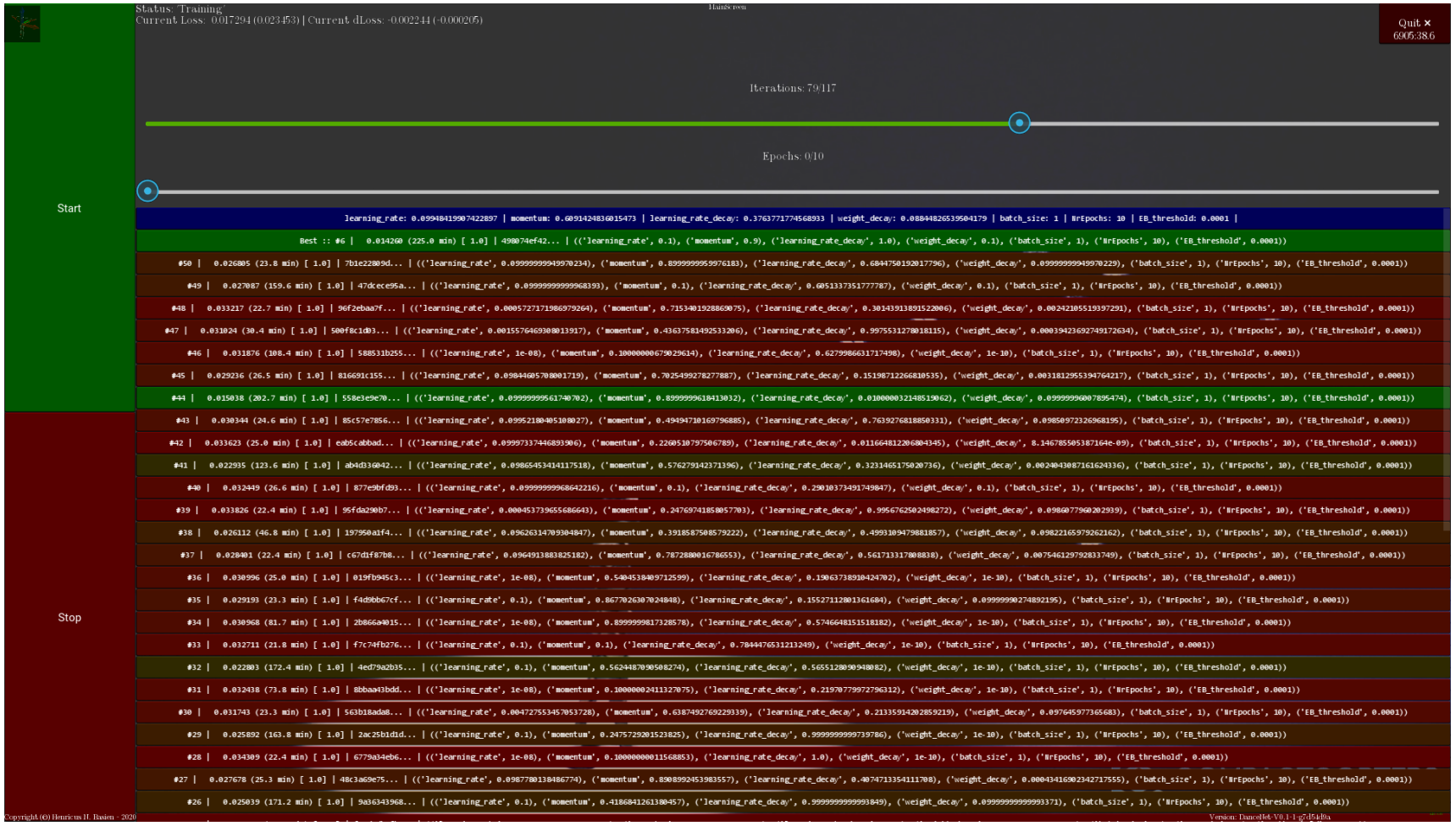


Figure C.1: BHPO GUI Examples: MainScreen
Current run progress bar, Hyperparameter optimization results & Start/Stop

C

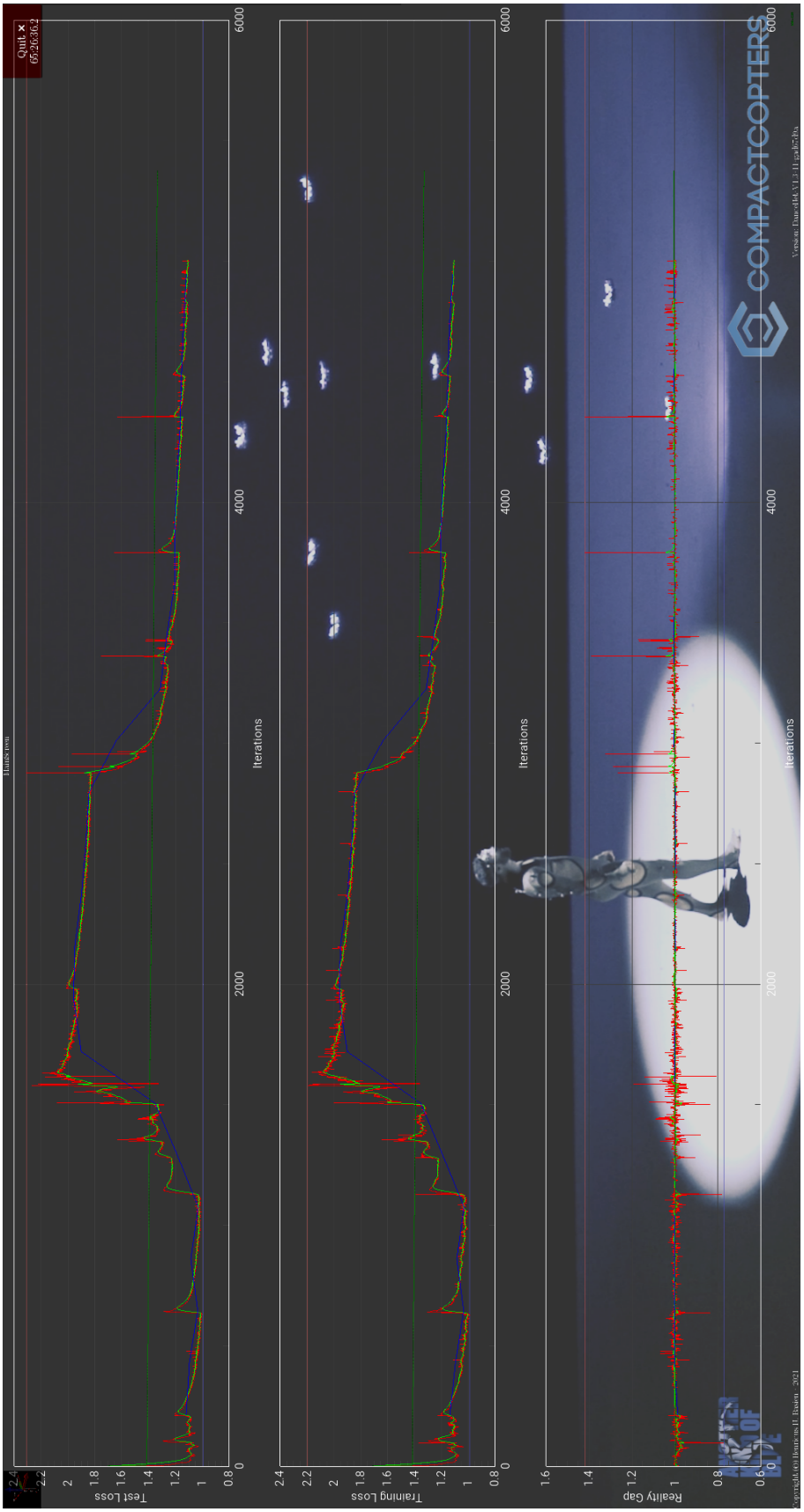


Figure C.2: BHPO GUI Examples: Graphs - Training/Testing Loss & Reality Gap

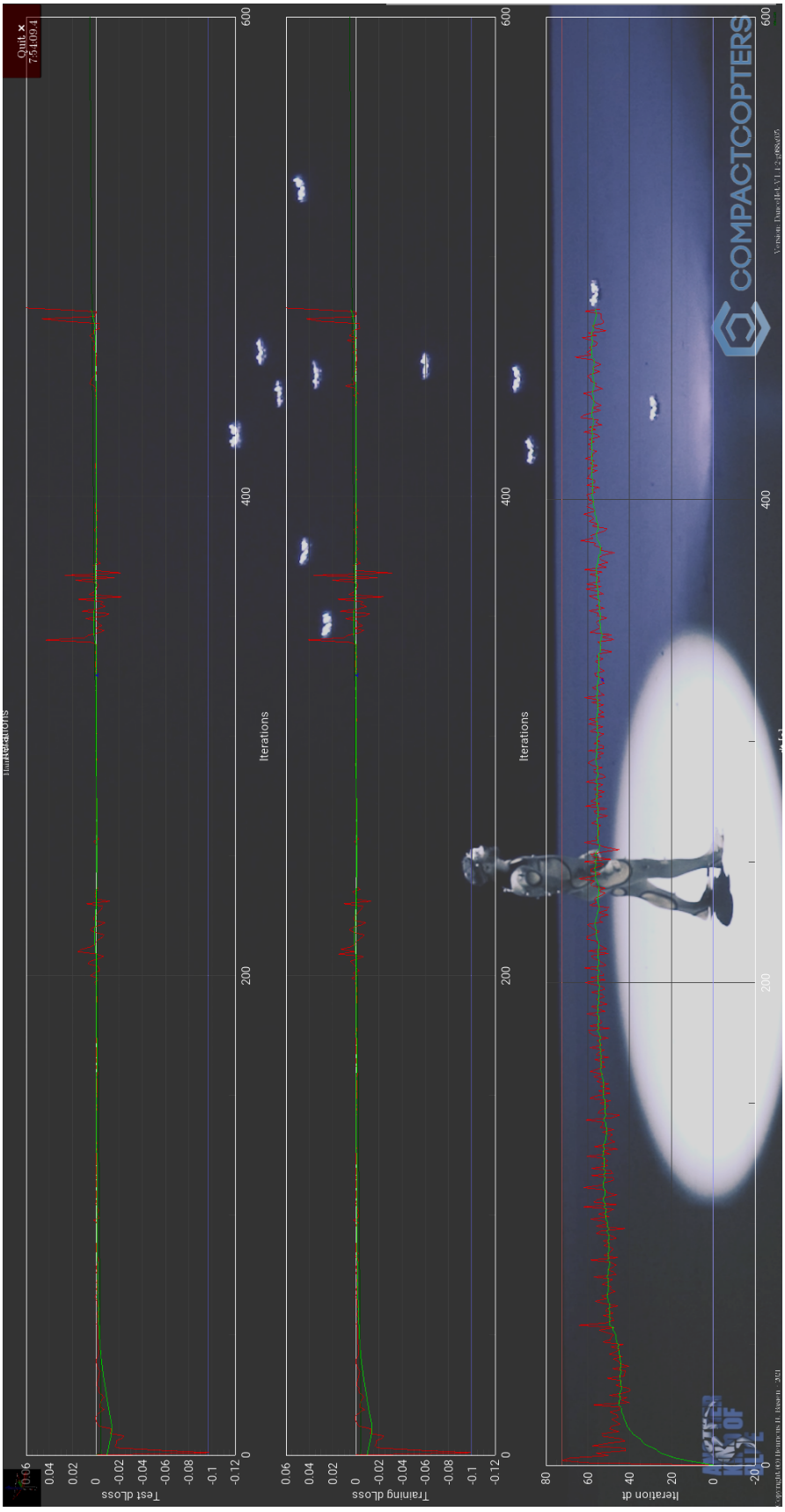


Figure C.3: BHPO GUI Examples: Graphs - Training/Testing ΔLoss & dt

D

Algorithms

The following appendix lists algorithms in Python that were developed for this research.

Disclaimer:

All parameters and functions are actually handled inside the larger scope of the BHPO framework classes. The following algorithms are stand-alone representations of various algorithms and are purely for illustrative purposes and conveying the primary logic. To this extend not every algorithm is 100% executable and holds some pseudo-code elements. These simplifications are required, as the full code would be to extensive to print in this report.

Source Code for Scientific Purposes

Neither the BHPO framework, nor the *DanceNet* model, nor the *AKOB* dataset are open-source or under public development, they are however freely available for research purposes under specific conditions. In case of legitimate interest in the underlying code or dataset, please contact the author via ' Henricus@Basien.de ' with your request.

Algorithm D.1 Early Break Criterion

```
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# Required Parameters
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

NrEpochs             = X # Max number of epochs to run
NrIterationsPerEpoch = Y # Iterations to run to complete 1 Epoch

n_iter          = 0     # Total number of iterations pass
DataLists       = {...} # Dictionary containing the raw and filtered network losses and their derivatives
HyperParameters = {...} # Dictionary containing the training hyperparameters

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# Early Break Check
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

breakcounter = 0

def CheckEarlyBreak(IterMin=NrIterationsPerEpoch,BreakcounterMin=5,IgnoreIterations=2,
                                                    AllowPositiveGradient=False,Debug=False):
    """
    Evaluates exponentially filtered loss derivative, to see if network training is still making progress.
    Ensures halting of training, prior to completing all epochs, if progress has stalled.

    Args:
        IterMin              (int ):
            Min. number of iterations to complete before allowing engaging of break
        BreakcounterMin      (int ):
            Number of iterations for which the criterion should be met, before engaging halt
        IgnoreIterations     (int ):
            Number of iterations to completely ignore, as first results can be unpredictable at times
        AllowPositiveGradient (bool):
            For normal NNs gradients should be negative to improve loss,
            but for e.g. GANs positive loss gradients to not indicate divergence and should be allowed
        Debug                (bool):
            If 'True': Prints additional information

    Returns:
        bool: Returns 'True' when EarlyBreaking condition is met and training should halt else 'False'
    """
    global breakcounter

    #--- Get loss data to evaluate ---
    EB_loss = DataLists["Test Loss"]#["Training Loss"]

    if n_iter>=IgnoreIterations+1: # IF: Allows for removing of the first few iterations
        #--- Retrieve loss derivatives ---
        Min_dLoss = min(EB_loss.d_data[IgnoreIterations:])
        Max_dLoss = max(EB_loss.d_data[IgnoreIterations:])
        Ext_dLoss = abs(Min_dLoss) if not AllowPositiveGradient else max([abs(Min_dLoss),abs(Max_dLoss)])

        if AllowPositiveGradient or Min_dLoss<0: # IF: Ensures loss gradient has been <0 at least once
            #--- Compute and Check Threshold ---
            dExpAvg           = float(EB_loss.GetdExpAvg_special()[-1])          # Get filtered derivative
            dExpAvg_threshold = float(Ext_dLoss*HyperParameters["EB_threshold"]) # Compute Threshold

            if Debug: # Construct Info Text
                dLoss_Info = "dLoss Avg: %f, dLoss Limit: %f [Min=%f,Max=%f]"%(dExpAvg,dExpAvg_threshold,
                                                                    Min_dLoss,Max_dLoss)

            # IF: Checks if primary criterion is met - Gradient has become very small or positive
            if abs(dExpAvg)<dExpAvg_threshold or (not AllowPositiveGradient and dExpAvg>0):
                breakcounter+=1 # Increment counter

                #--- Print Info ---
                if Debug:
                    BreakText = "%i: Breaking point reached! (%s)"%(n_iter,dLoss_Info)
                    BreakText+= "...Counting %i/%i"%(breakcounter,BreakcounterMin)
                    print(BreakText)

                # IF: Check if secondary criterion is met - Ensures minimum number of iterations
                if IterMin==False or n_iter>IterMin:
                    if breakcounter>=BreakcounterMin:
                        if Debug: print("...Engaging Break!")
                        return True # <-------------------------------------- Engages EarlyBreak!
                else:
                    if Debug: print("...but min. NrIteration hasn't completed yet! (%i/%i)"%(n_iter,
                                                                                IterMin))
                return False # Return without resetting breakcounter
            else:
                if Debug: print("Breakpoint not reached (%s)"%dLoss_Info)

    #--- Reset counter and return ---
    breakcounter = 0
    return False

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# Training: Simplified Pseudo code
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

BreakFlag = False
for n_e in range(NrEpochs):
    for n_i in range(NrIterationsPerEpoch):
        n_iter+=1

        # <<< [...] Training Code               >>>
        # <<< [...] Store results in 'DataLists' >>>

        if CheckEarlyBreak():
            BreakFlag = True
            break
    if BreakFlag: break
```

Algorithm D.2 Neural network Architecture: Soft-Self-Attention Layer

```
#=============================================================================
# Imports
#=============================================================================

#--- PyTorch ---
import torch
import torch.nn as nn

#=============================================================================
# Pure Bias Layer
#=============================================================================

class BiasLayer(nn.Module):

    def __init__(self,dim):
        super(BiasLayer, self).__init__()
        self.bias = nn.Parameter(torch.zeros(dim))

    def forward(self,x):
        return self.bias+x

#=============================================================================
# Attention Layer
#=============================================================================

class AttentionLayer(nn.Module):

    def __init__(self,dim,InitWeights=True):
        super(AttentionLayer, self).__init__()

        self.dim        = int(dim)
        self.input_bias = BiasLayer(dim)
        self.linear     = nn.Linear(self.dim, self.dim, bias=True)
        self.softmax    = nn.Softmax(dim=-1)

        if InitWeights: self.InitWeights()

    def InitWeights(self): # Ensure equal softmax output at T=0
        self.linear.weight.data.fill_(1.0/self.dim)
        self.linear.bias  .data.fill_(0.0)

    def forward(self,x):
        x_b = self.input_bias(x)
        attention_scores        = self.linear(x_b)
        attention_weights       = self.softmax(attention_scores)
        if 0: attention_weights = attention_weights*self.dim # Rescale for sum(
                                                             attention_weights)=dim
        y = x*attention_weights
        return y,attention_weights
```

E

DRA-GAN Results

E.1. V1.0-23 | 6D-ATT-DI-DO-LSTM-GAN | e406d70f24

E.1.1. Data

General Settings	
Parameter	**Value**
ModelID	6D-ATT-DI-DO-LSTM-GAN
DanceNet Version	V1.0-23-g62851ab
Hyperparameter Hash	e406d70f24
# Epochs|Iterations	3|1130

Table E.1: V1.0-23 | 6D-ATT-DI-DO-LSTM-GAN | e406d70f24: General Settings

General	
Hyperparameter	**Value**
Data	
Chunk Length	10s
Optimizers	
Learning Rate	0.00005
β_1	0.9
β_2	0.999
Regularization	
Learning Rate Decay	0.5
Weight Decay	0.025
Batch Size	8
Early Break	
Nr. of Epochs	25
EB Threshold	0.0001

Architecture	
Hyperparameter	**Value**
Network Size	
# Layers	2
G-OutputRatio	1.2
D-OutputRatio	0.8
Pain	
Pain Exponent	N/A
Limited Judgement	
# Truncated Frames	N/A

Table E.2: V1.0-23 | 6D-ATT-DI-DO-LSTM-GAN | e406d70f24: Hyperparameters

E.1.2. Brief Analysis

- Pain losses are missing, as pain loss was not implemented yet.
- Generator and discriminator weights appear almost complementary.
- Generator focuses mostly on the derivatives.
- Discriminator focuses on the poses, with more focus on the output pose.
- Discriminator output over time shows interaction between the two networks and appears chaotic at times.

E.1.3. Plots

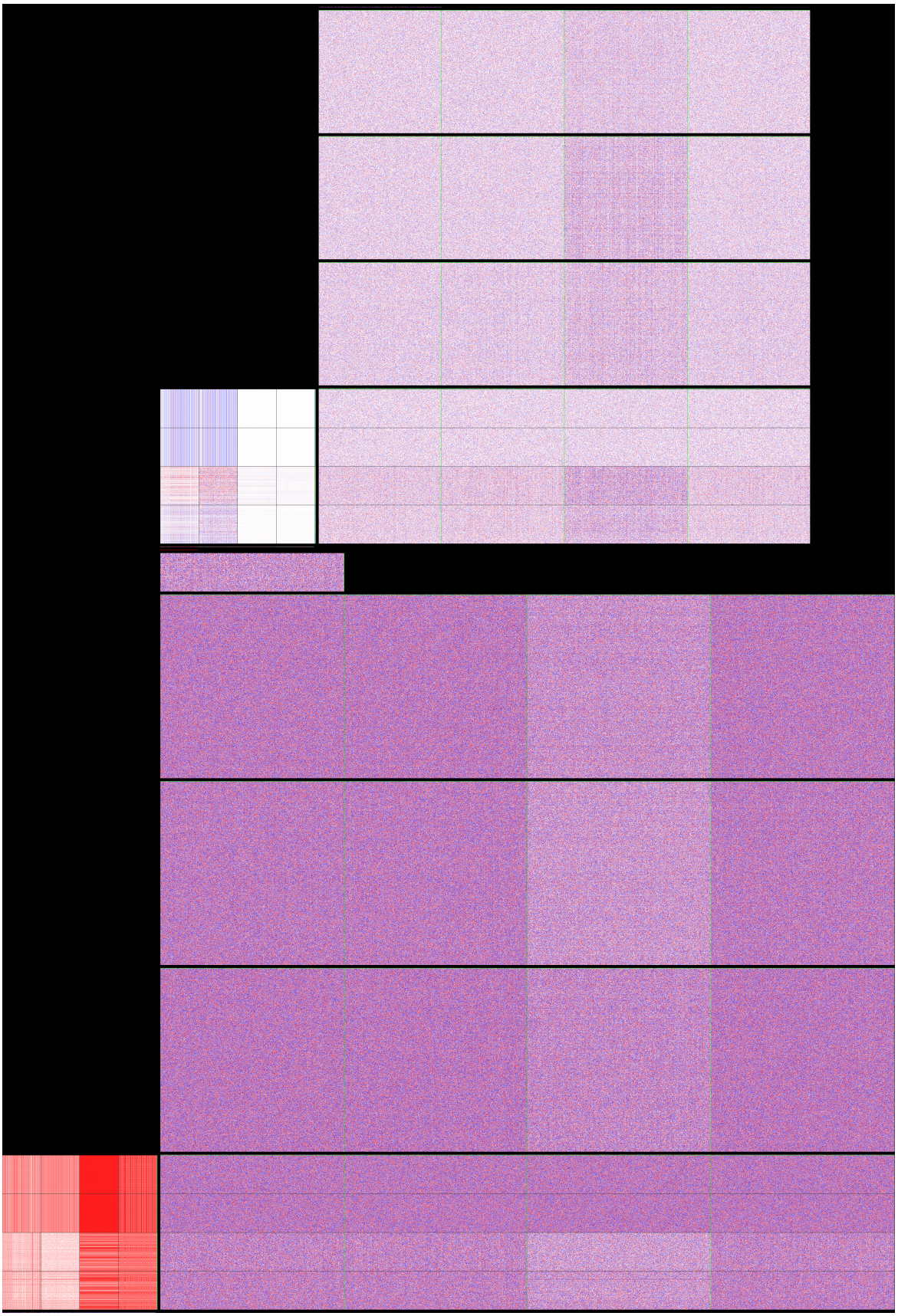


Figure E.1: V1.0-23 | 6D-ATT-DI-DO-LSTM-GAN | e406d70f24: Network Weights

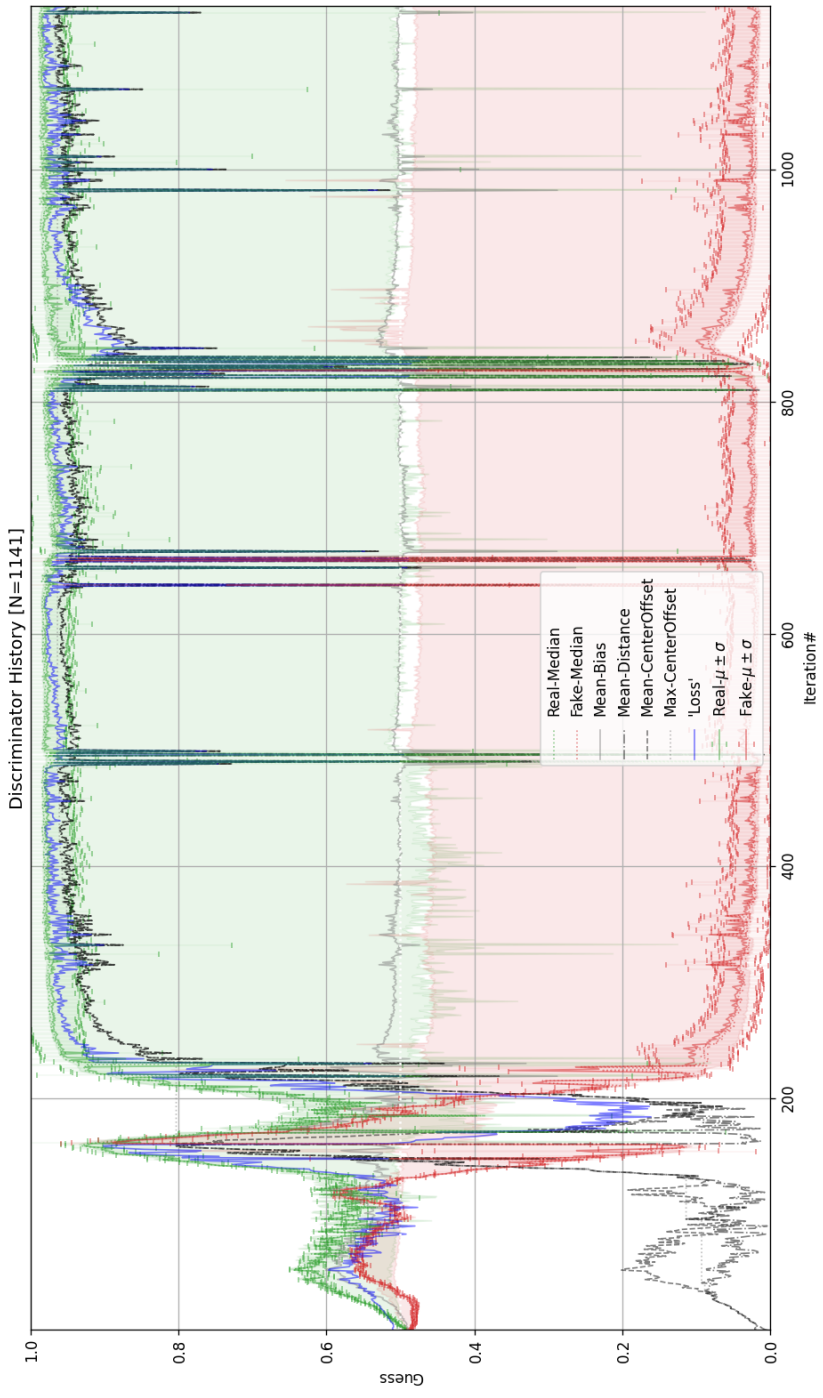


Figure E.2: V1.0-23 | 6D-ATT-DI-DO-LSTM-GAN | e406d70f24: Discriminator History

E.2. V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | d9be67f6a7

E.2.1. Data

General Settings	
Parameter	**Value**
ModelID	6D-ATT-DI-DO-LSTM-Pain-GAN
DanceNet Version	V1.1
Hyperparameter Hash	d9be67f6a7
# Epochs|Iterations	8|2720

Table E.3: V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | d9be67f6a7: General Settings

General	
Hyperparameter	**Value**
Data	
Chunk Length	10s
Optimizers	
Learning Rate	0.00005
β_1	0.9
β_2	0.999
Regularization	
Learning Rate Decay	0.5
Weight Decay	0.025
Batch Size	8
Early Break	
Nr. of Epochs	25
EB Threshold	0.00005

Architecture	
Hyperparameter	**Value**
Network Size	
# Layers	2
G-OutputRatio	1.2
D-OutputRatio	0.8
Pain	
Pain Exponent	8
Limited Judgement	
# Truncated Frames	N/A

Table E.4: V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | d9be67f6a7: Hyperparameters

E.2.2. Brief Analysis

- Pain loss reduces the exceeding of limits over time, but does not fully prevent it.
- Discriminator progression and weights comparable to V1.0-23 | 6D-ATT-DI-DO-LSTM-GAN | e406d70f24, as HPs are identical.

E.2.3. Plots

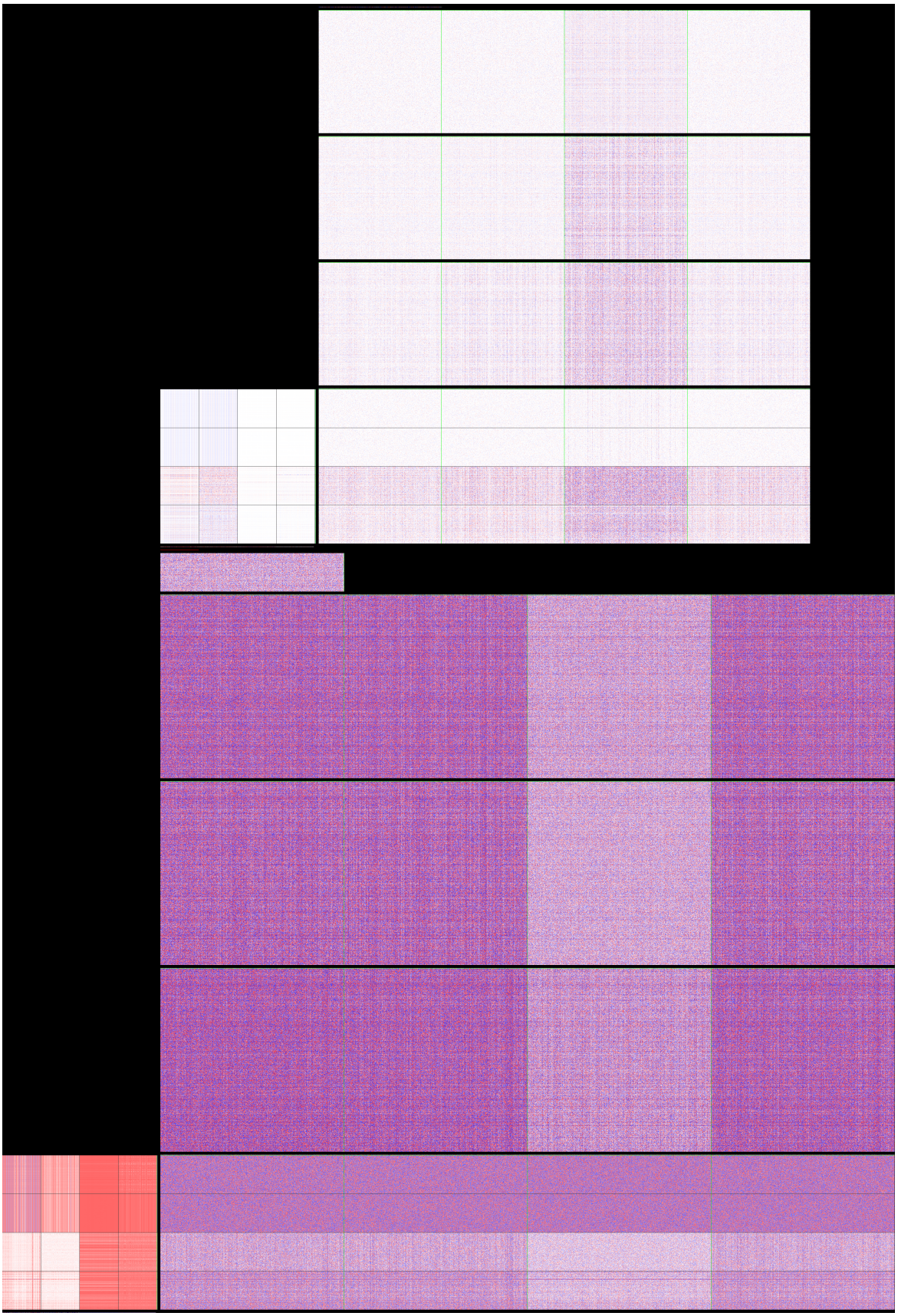


Figure E.3: V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | d9be67f6a7: Network Weights

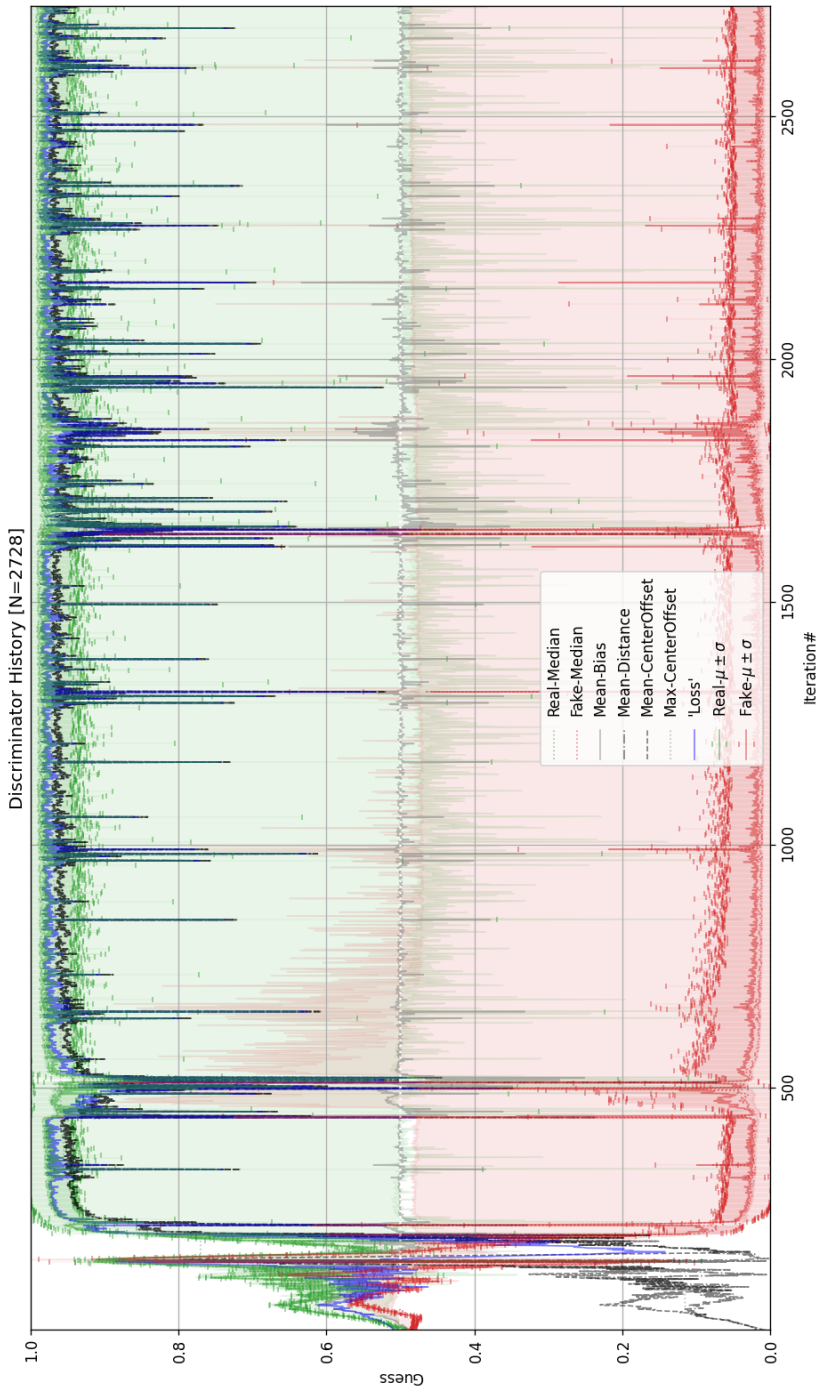


Figure E.4: V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | d9be67f6a7: Discriminator History

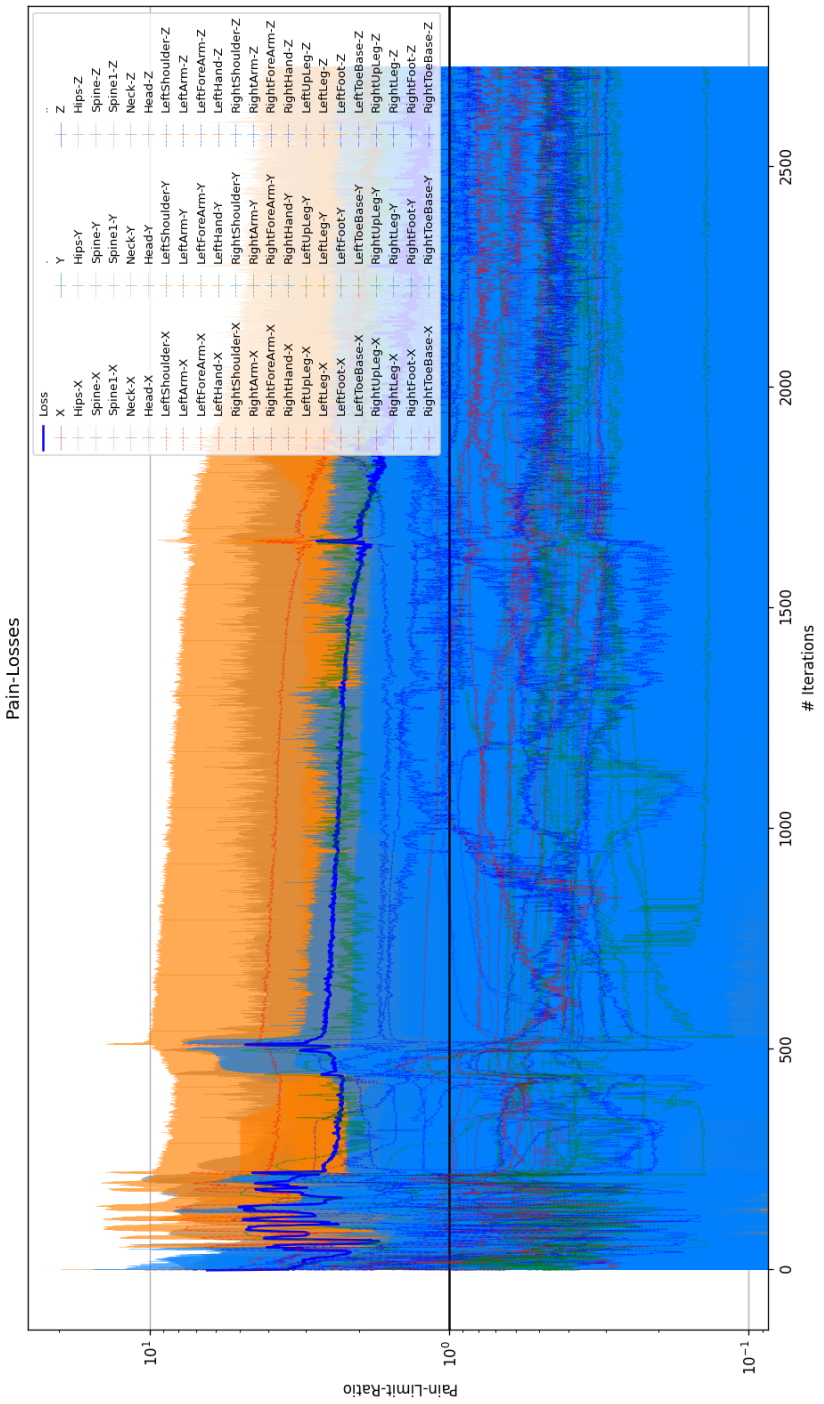


Figure E.5: V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | d9be67f6a7: Pain History

E.3. V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | 42241955b0

E.3.1. Data

General Settings	
Parameter	**Value**
ModelID	6D-ATT-DI-DO-LSTM-Pain-GAN
DanceNet Version	V1.1
Hyperparameter Hash	42241955b0
# Epochs|Iterations	0|410

Table E.5: V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | 42241955b0: General Settings

General	
Hyperparameter	**Value**
Data	
Chunk Length	10s
Optimizers	
Learning Rate	0.00001422362090114059 2
β_1	0.6732695254481775
β_2	0.9148126046862071
Regularization	
Learning Rate Decay	$\frac{1}{\varphi} \approx 0.618$
Weight Decay	0.09818293898364239
Batch Size	4
Early Break	
Nr. of Epochs	25
EB Threshold	0.00005

Architecture	
Hyperparameter	**Value**
Network Size	
# Layers	2
G-OutputRatio	1.848943555312918
D-OutputRatio	1.4268150378762856
Pain	
Pain Exponent	8
Limited Judgement	
# Truncated Frames	N/A

Table E.6: V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | 42241955b0: Hyperparameters

E.3.2. Brief Analysis

- There is higher activity in the hidden layers, than in the input layers, suggesting a potential overreliance on past data.
- Discriminator history shows considerable interplay between generator and discriminator and the partial independence between processing real and fake data.
  - The guesses for real data are continuously increasing.
  - The generator manages to learn from the discriminator and fool it three times.
  - While the discriminator can learn features of real and fake data separately, once the generator fools the discriminator at ≈ 260 iterations, the discriminators confidence in the real guesses drops as well.
- Angular limits are significantly exceeded.

E.3.3. Plots

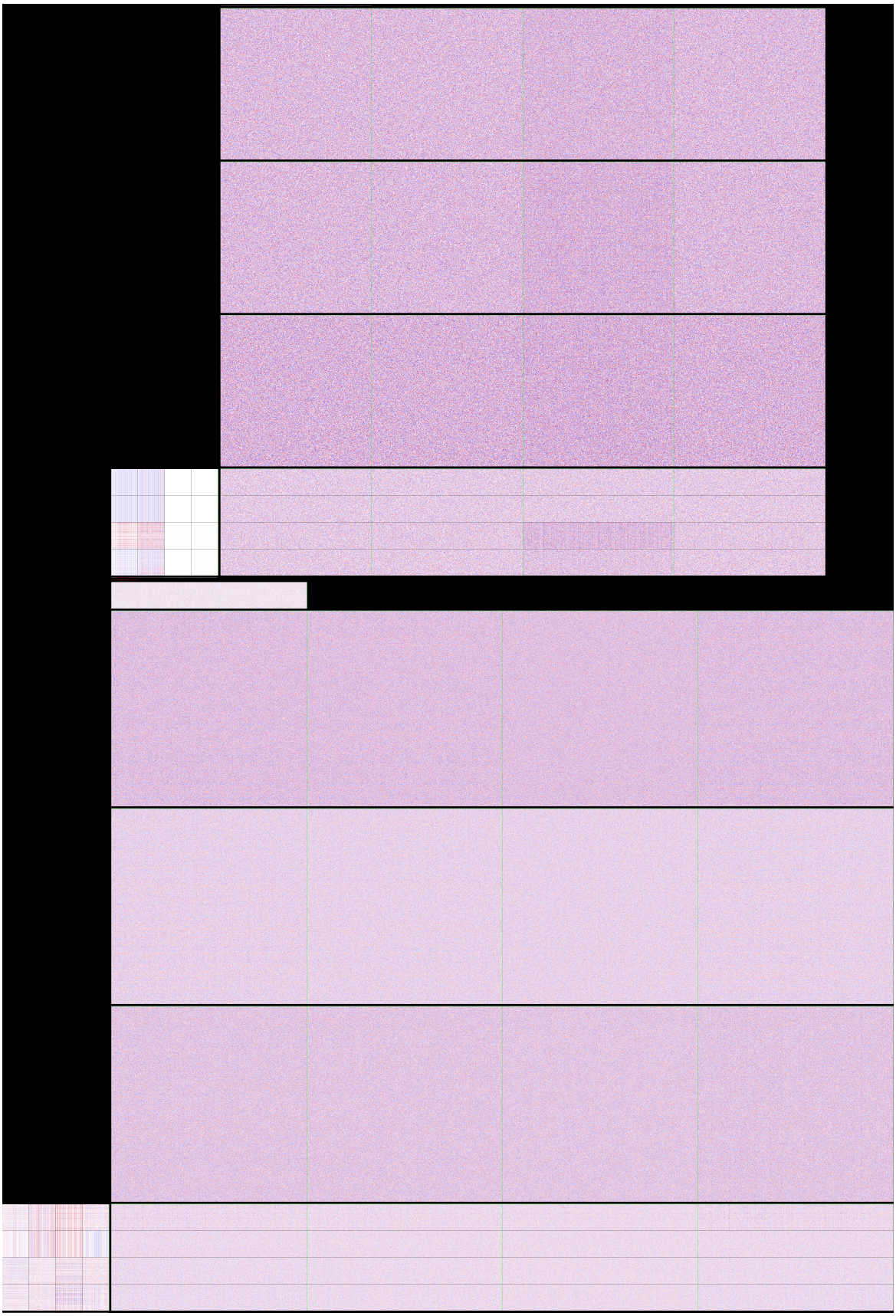


Figure E.6: V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | 42241955b0: Network Weights

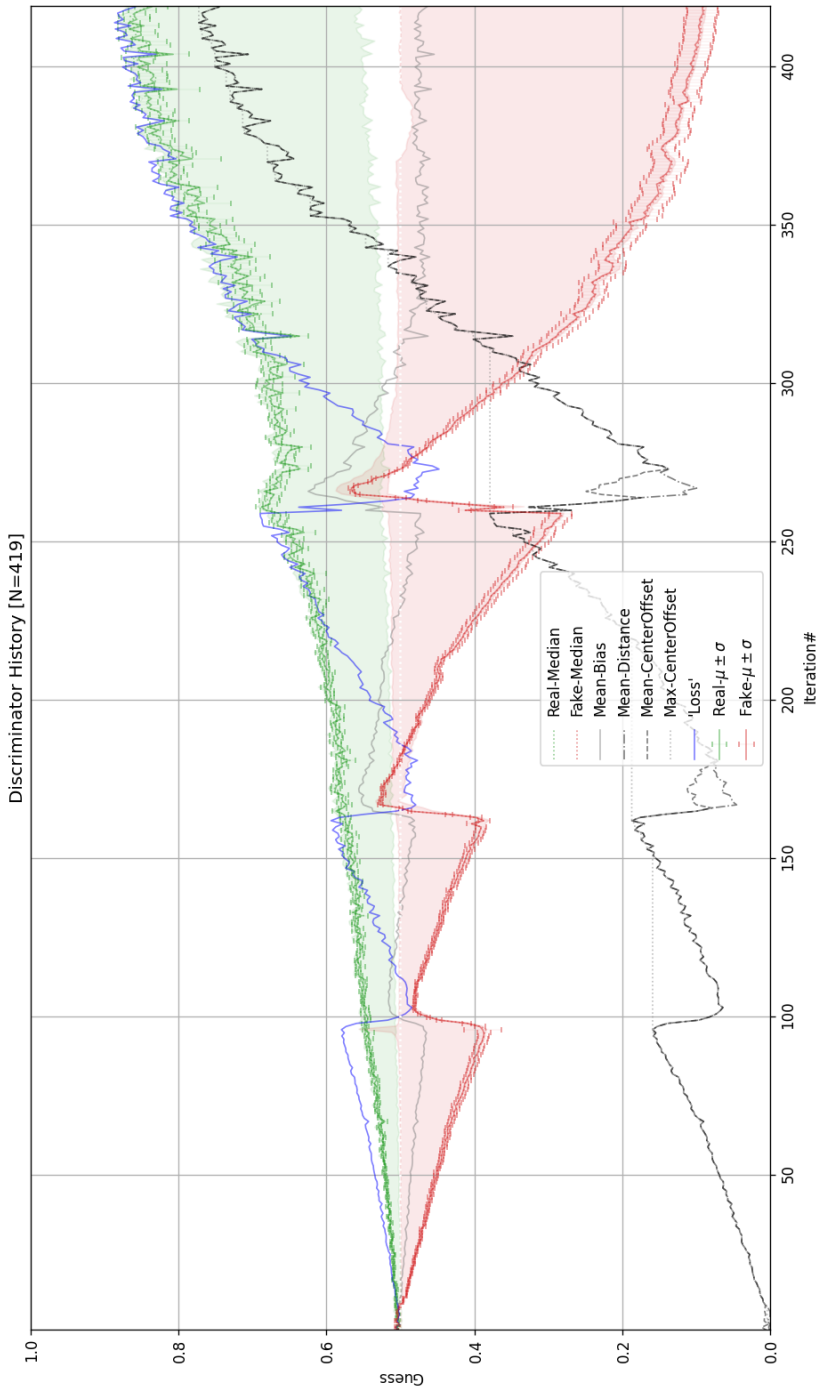


Figure E.7: V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | 42241955b0: Discriminator History

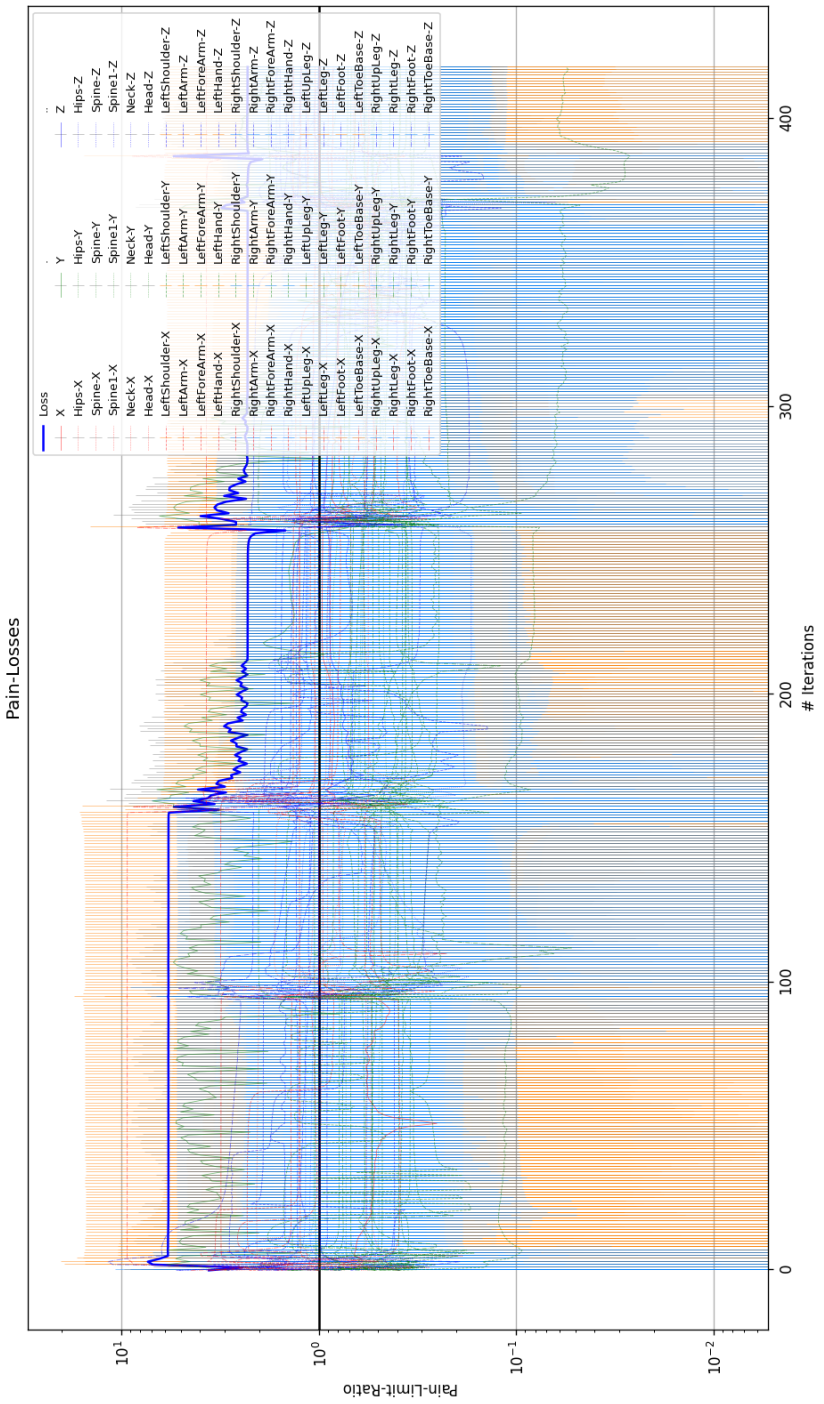


Figure E.8: V1.1 | 6D-ATT-DI-DO-LSTM-Pain-GAN | 42241955b0: Pain History

E.4. V1.2 | 6D-PCA-ATT-DI-DO-LSTM-Pain-GAN | 96142329a6

E.4.1. Data

General Settings	
Parameter	**Value**
ModelID	6D-PCA-ATT-DI-DO-LSTM-Pain-GAN
DanceNet Version	V1.2
Hyperparameter Hash	96142329a6
# Epochs|Iterations	5|1760

Table E.7: V1.2 | 6D-PCA-ATT-DI-DO-LSTM-Pain-GAN | 96142329a6: General Settings

General	
Hyperparameter	**Value**
Data	
Chunk Length	10s
Optimizers	
Learning Rate	0.000222
β_1	0.9
β_2	0.999
Regularization	
Learning Rate Decay	$\frac{1}{\varphi} \approx 0.618$
Weight Decay	0.01
Batch Size	8
Early Break	
Nr. of Epochs	25
EB Threshold	0.00005

Architecture	
Hyperparameter	**Value**
Network Size	
# Layers	3
G-OutputRatio	1.1
D-OutputRatio	0.75
Pain	
Pain Exponent	8
Limited Judgement	
# Truncated Frames	N/A

Table E.8: V1.2 | 6D-PCA-ATT-DI-DO-LSTM-Pain-GAN | 96142329a6: Hyperparameters

E.4.2. Brief Analysis

- Discriminator focuses almost solely on the output pose, while the generator focuses on anything but its own output pose.
- Discriminator is uncertain for a long time, before suddenly finding appropriate discriminatory features, around ≈ 900 iterations.
- Generator initially manages to overreact to the discriminators initial discriminatory features, to the point that the fake data is classified as real with significantly more confidence that the real data.
- The bias towards the fake data, around ≈ 900 iterations, indicates a lag in the detection of real data.
- PCA transform successfully keeps limits within bounds at the start.
- Joint limits are increasing and exceeding the limits, up until the discriminator manages to successfully classify the data, resulting in a steady return of the angles within the limits.

E.4.3. Plots

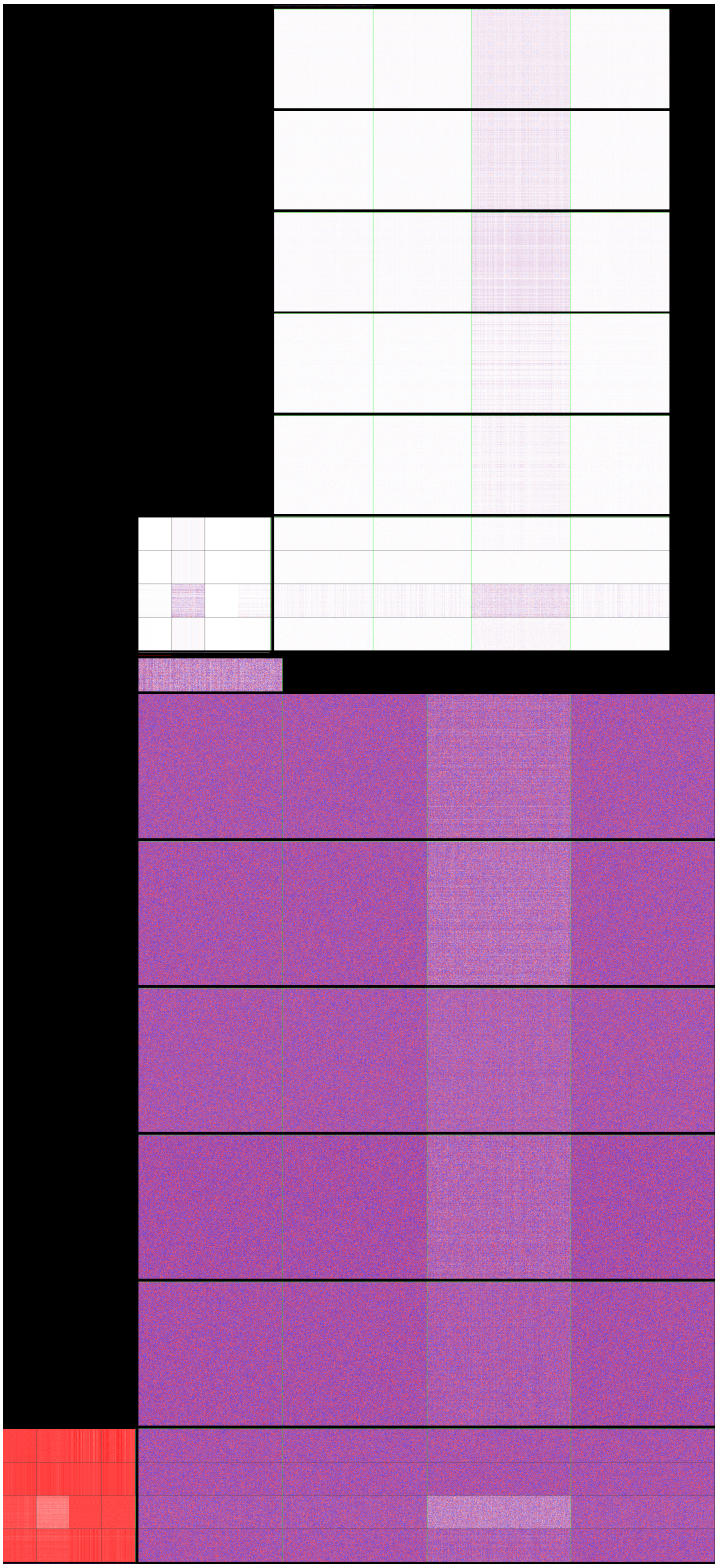


Figure E.9: V1.2 | 6D-PCA-ATT-DI-DO-LSTM-Pain-GAN | 96142329a6: Network Weights

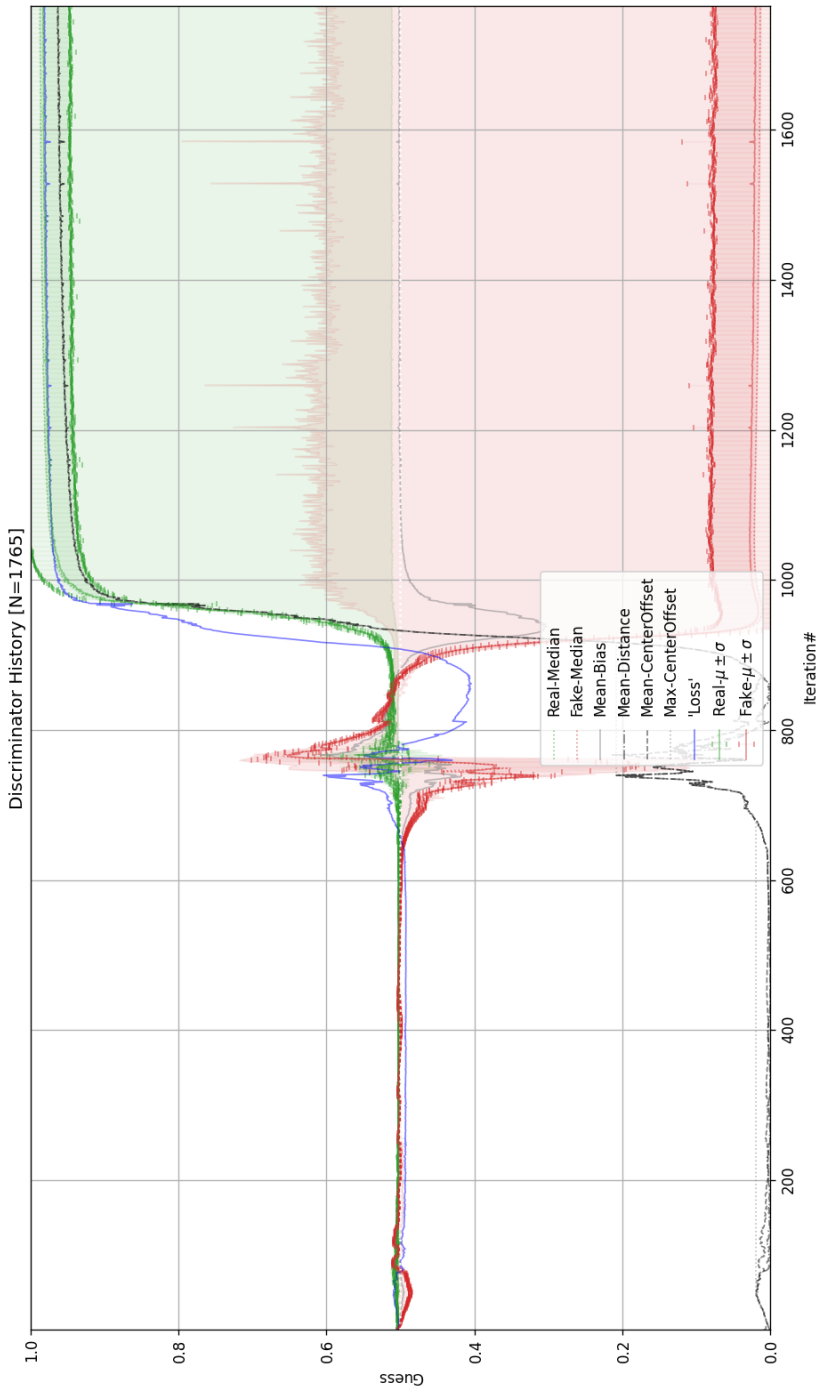


Figure E.10: V1.2 | 6D-PCA-ATT-DI-DO-LSTM-Pain-GAN | 96142329a6: Discriminator History

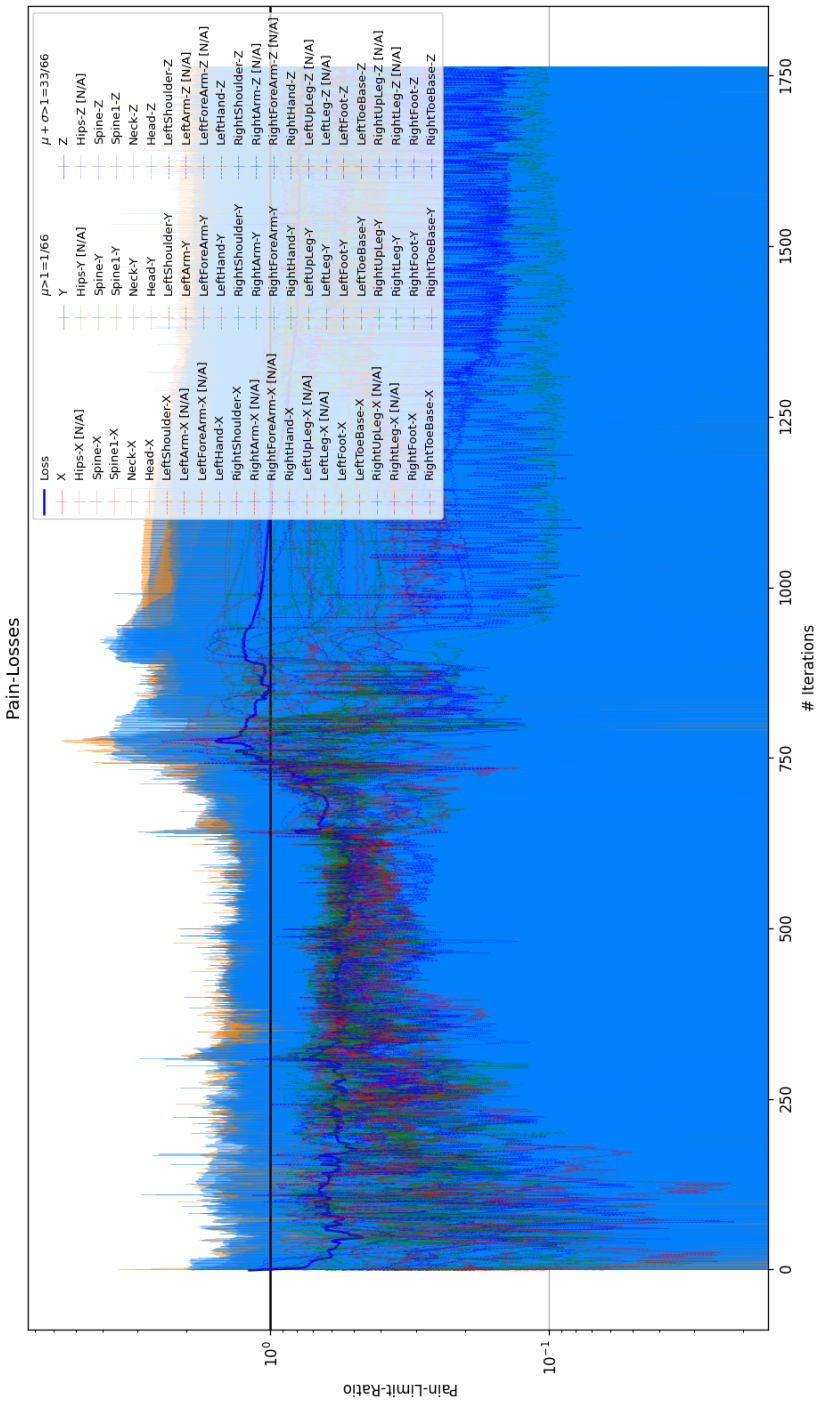


Figure E.11: V1.2 | 6D-PCA-ATT-DI-DO-LSTM-Pain-GAN | 96142329a6: Pain History

E.5. V1.3-7 | 6D-PCA-ATT-DI-DO-LSTM-Pain-LJ-GAN | 7d2d6588e0

E.5.1. Data

General Settings	
Parameter	**Value**
ModelID	6D-PCA-ATT-DI-DO-LSTM-Pain-LJ-GAN
DanceNet Version	V1.3-7-g5d77084
Hyperparameter Hash	7d2d6588e0
# Epochs|Iterations	23|5090

Table E.9: V1.3-7 | 6D-PCA-ATT-DI-DO-LSTM-Pain-LJ-GAN | 7d2d6588e0: General Settings

General	
Hyperparameter	**Value**
Data	
Chunk Length	15s
Optimizers	
Learning Rate	0.00645
β_1	0.9
β_2	0.999
Regularization	
Learning Rate Decay	$\frac{1}{\varphi} \approx 0.618$
Weight Decay	0.01
Batch Size	8
Early Break	
Nr. of Epochs	25
EB Threshold	0.0001

Architecture	
Hyperparameter	**Value**
Network Size	
# Layers	3
G-OutputRatio	1.1
D-OutputRatio	0.75
Pain	
Pain Exponent	8
Limited Judgement	
# Truncated Frames	100

Table E.10: V1.3-7 | 6D-PCA-ATT-DI-DO-LSTM-Pain-LJ-GAN | 7d2d6588e0: Hyperparameters

E.5.2. Brief Analysis

- Discriminator focuses almost solely on the output pose, while the generator focuses primarily on the input derivative.
- Discriminator is uncertain for a long time, before suddenly finding appropriate discriminatory features, around $\approx$ 1500 iterations.
- Overall similar progression than V1.2 | 6D-PCA-ATT-DI-DO-LSTM-Pain-GAN | 96142329a6, except that the limits are 100% within bounds towards the end.
- This has been the best result so far (see Figure 7.17).

E.5.3. Plots

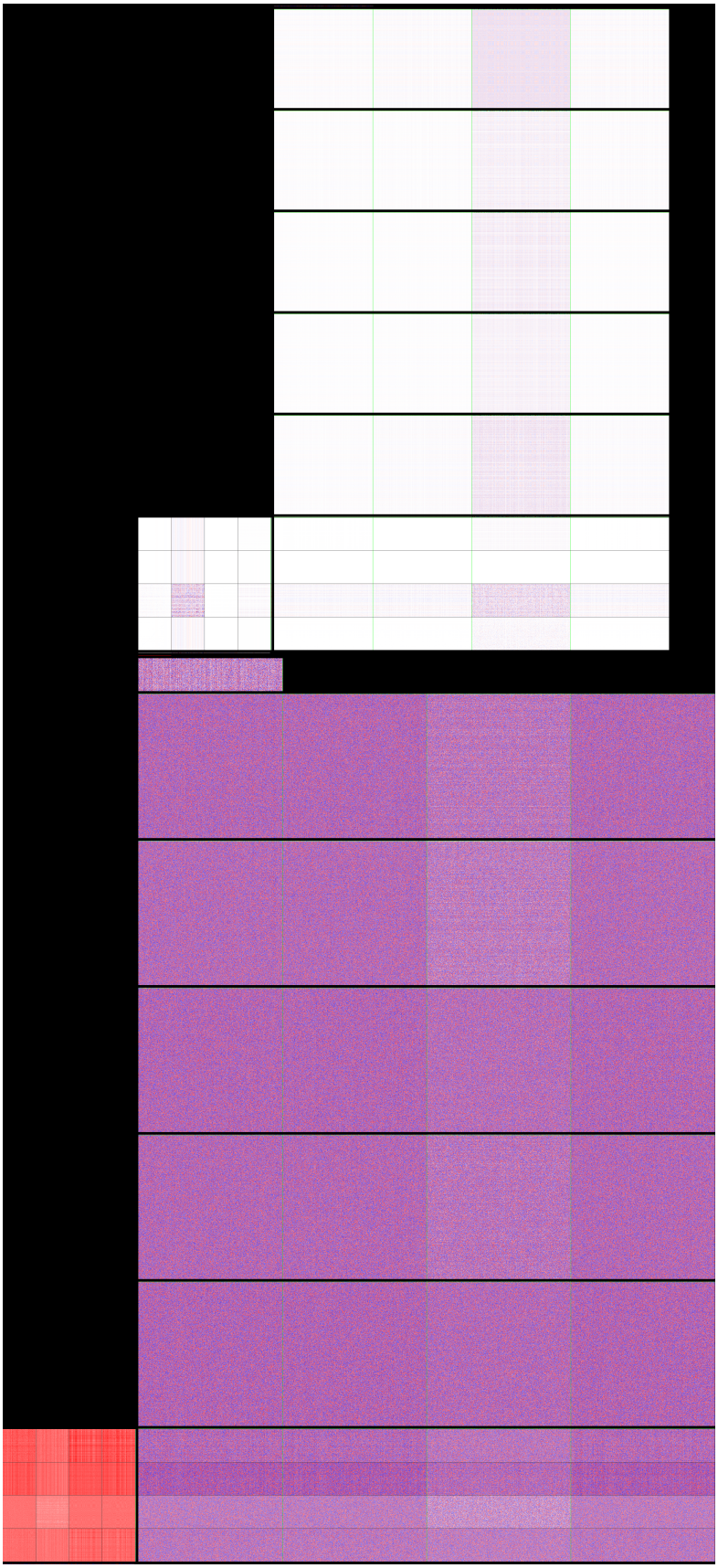


Figure E.12: V1.3-7 | 6D-PCA-ATT-DI-DO-LSTM-Pain-LJ-GAN | 7d2d6588e0: Network Weights

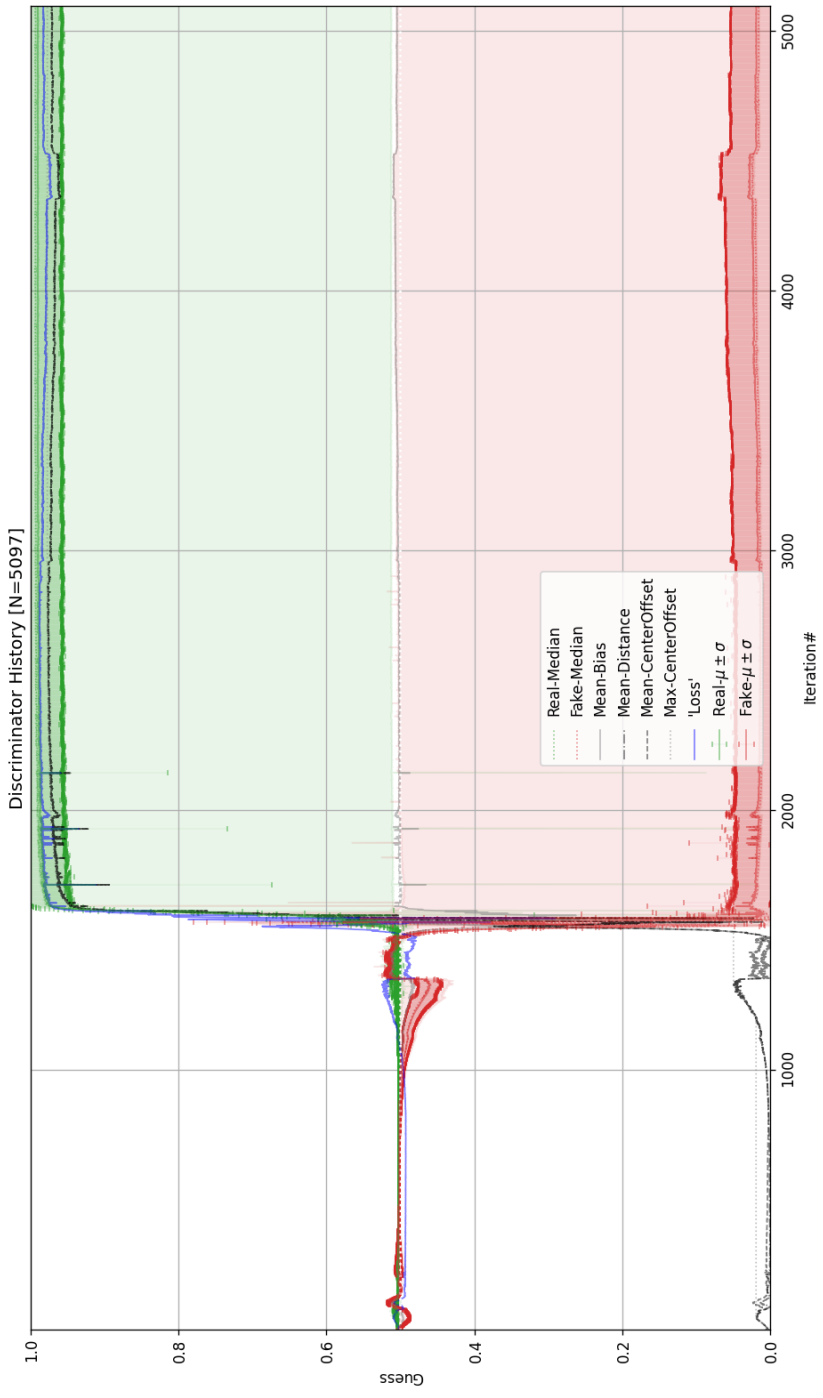


Figure E.13: V1.3-7 | 6D-PCA-ATT-DI-DO-LSTM-Pain-LJ-GAN | 7d2d6588e0: Discriminator History

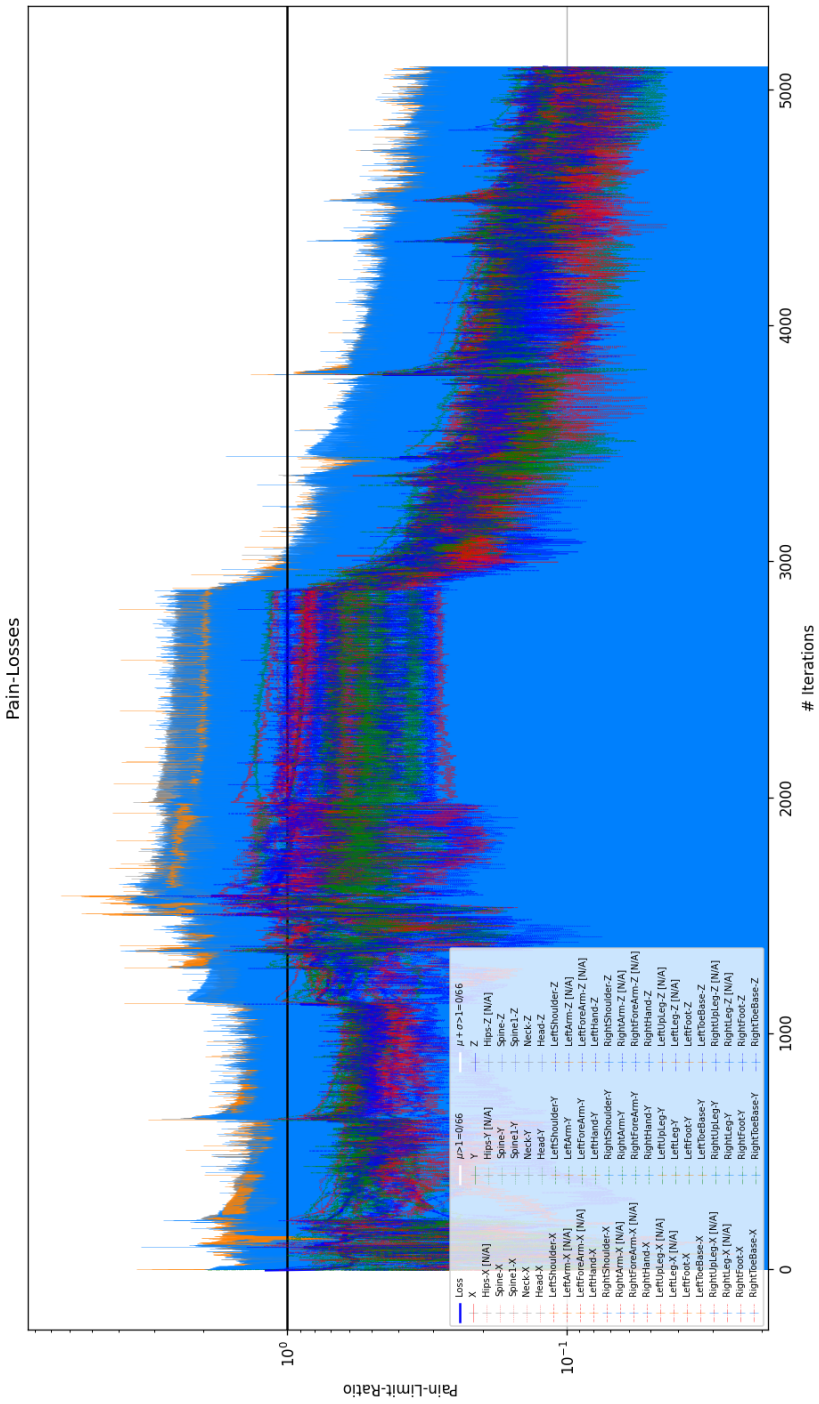


Figure E.14: V1.3-7 | 6D-PCA-ATT-DI-DO-LSTM-Pain-LJ-GAN | 7d2d6588e0: Pain History

E.6. V1.3-15 | 6D-PCA-ATT-DI-DO-GRU-Pain-LJ-GAN | 89ba1481a7

E.6.1. Data

General Settings	
Parameter	**Value**
ModelID	6D-PCA-ATT-DI-DO-GRU-Pain-LJ-GAN
DanceNet Version	V1.3-15-g58336c0
Hyperparameter Hash	89ba1481a7
# Epochs|Iterations	27|8900

Table E.11: V1.3-15 | 6D-PCA-ATT-DI-DO-GRU-Pain-LJ-GAN | 89ba1481a7: General Settings

General	
Hyperparameter	**Value**
Data	
Chunk Length	10s
Optimizers	
Learning Rate	0.000222
β_1	0.9
β_2	0.999
Regularization	
Learning Rate Decay	0.9
Weight Decay	0.005
Batch Size	8
Early Break	
Nr. of Epochs	50
EB Threshold	0.0001

Architecture	
Hyperparameter	**Value**
Network Size	
# Layers	4
G-OutputRatio	1.0
D-OutputRatio	0.75
Pain	
Pain Exponent	8
Limited Judgement	
# Truncated Frames	25

Table E.12: V1.3-15 | 6D-PCA-ATT-DI-DO-GRU-Pain-LJ-GAN | 89ba1481a7: Hyperparameters

E.6.2. Brief Analysis

- An attempt was made to improve the results of V1.3-7 | 6D-PCA-ATT-DI-DO-LSTM-Pain-LJ-GAN | 7d2d6588e0, by ...:
  - ...increasing the number of layers and parameters, to allow the network to learn more complex relations.
  - ...decreasing the weight decay, to avoid the decay of the discriminator weights to almost 0.
  - ...decreasing the learning rate and its decay, to more slowly and steadily approach the desired goal.
  - ...decreasing the chunk size again and switching to the GRU, to significantly decrease the training time.
- The discriminator makes no progress at all.
- The joint limits are highly chaotic.
- → The attempt at improving the previous configuration has failed and it shows how varied the output of the same model can be, based on the hyperparameters used.

E.6.3. Plots

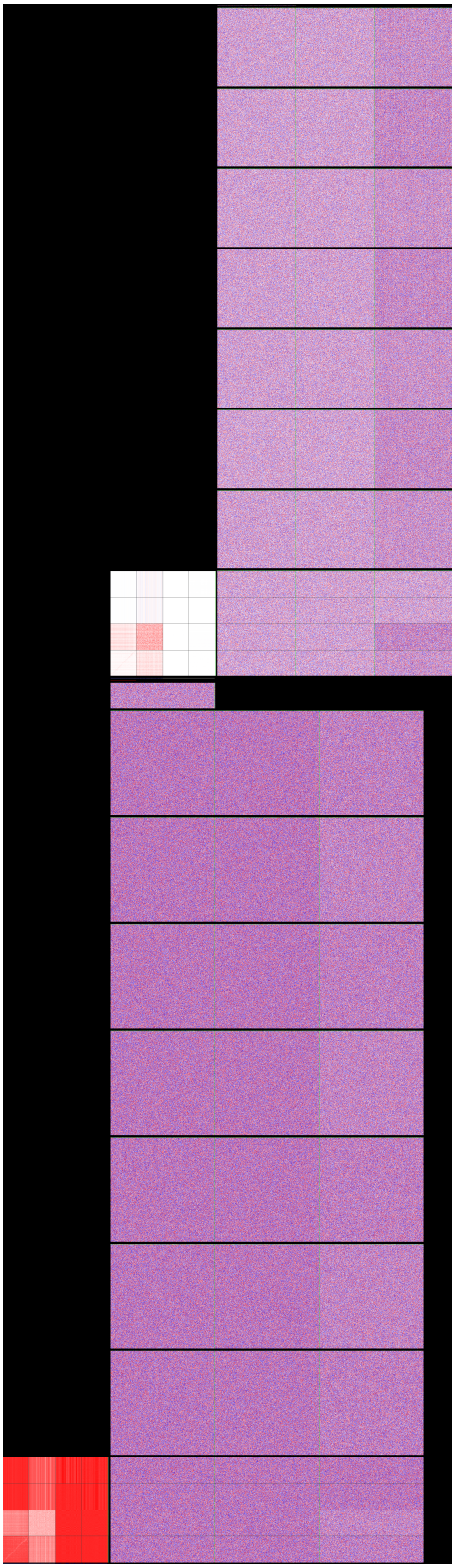


Figure E.15: V1.3-15 | 6D-PCA-ATT-DI-DO-GRU-Pain-LJ-GAN | 89ba1481a7: Network Weights

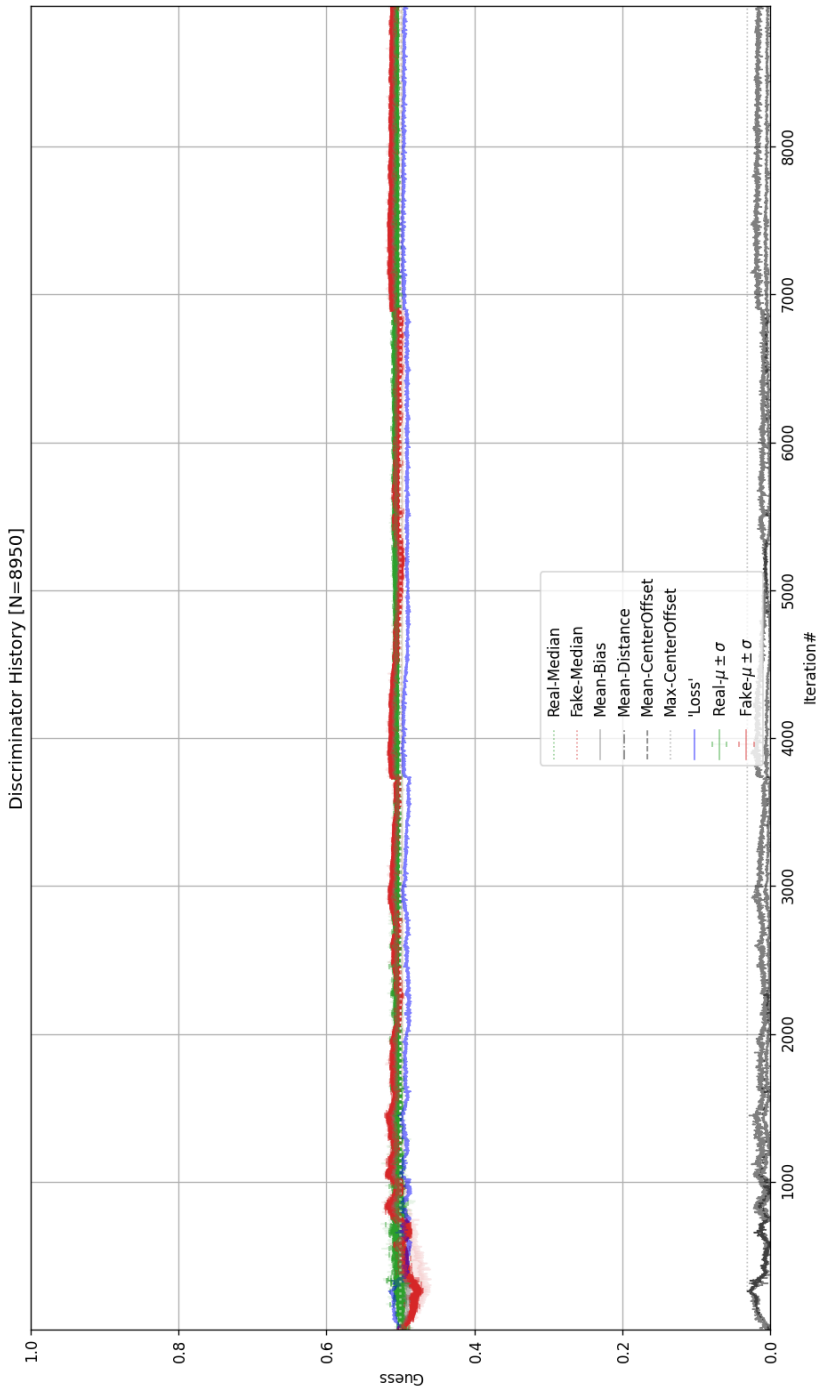


Figure E.16: V1.3-15 | 6D-PCA-ATT-DI-DO-GRU-Pain-LJ-GAN | 89ba1481a7: Discriminator History

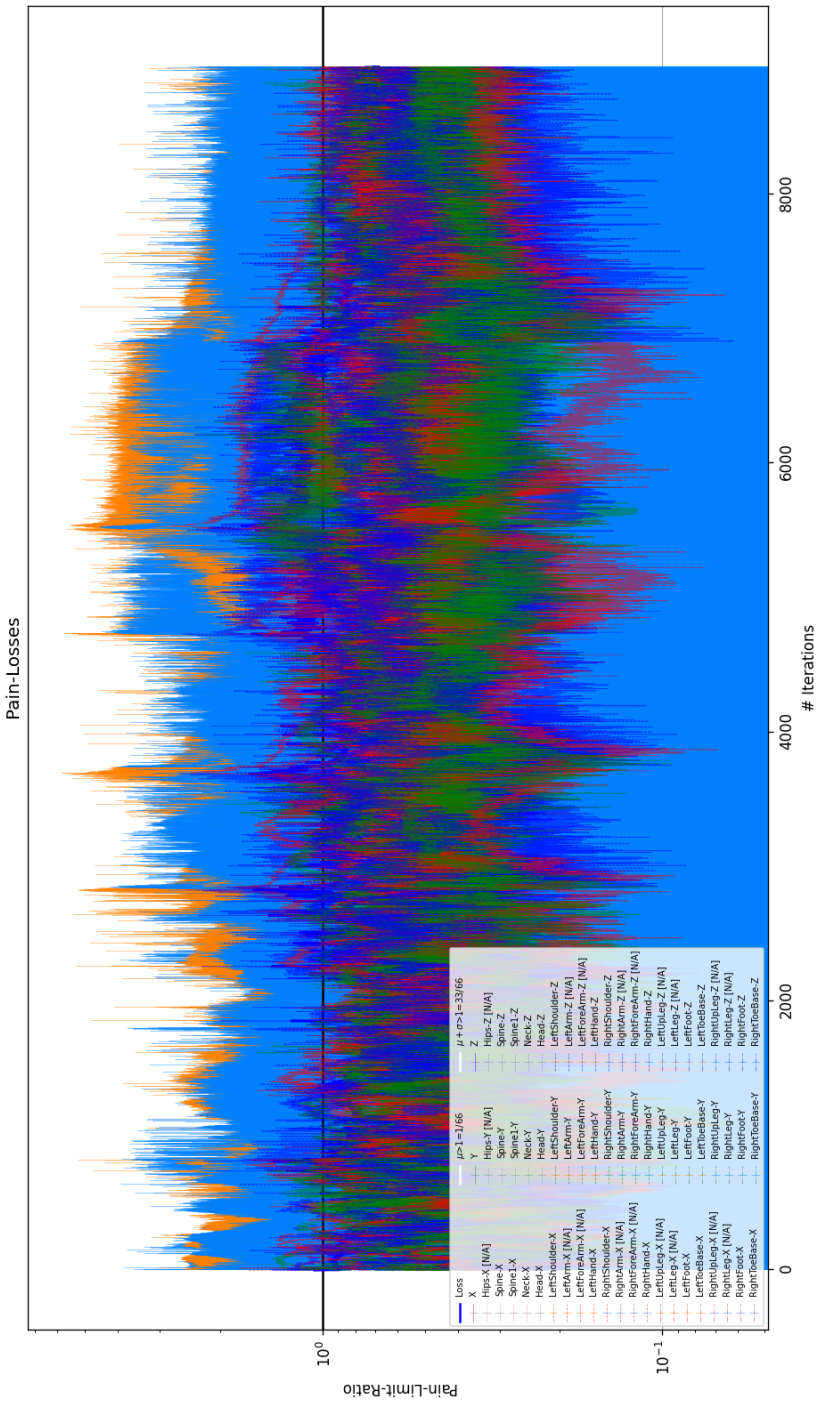


Figure E.17: V1.3-15 | 6D-PCA-ATT-DI-DO-GRU-Pain-LJ-GAN | 89ba1481a7: Pain History

[This page intentionally left blank]

AI-Man

Preliminary Report

H. N. Basien

Preliminary report for MSc. thesis:
'Neural Network based Emergence of reactionary Motion Improvisation'
set within the *'AI-Man'* project, in collaboration with -
Another kind of Blue & *CompactCopters*

TU Delft

AI-Man

Preliminary Report

by

H. N. Basien

to obtain the degree of Master of Science
at the Delft University of Technology,
Faculty of Aerospace Engineering - Department of Control & Operations,
to be defended publicly on **TBA**

Student number:	4207653		
Project duration:	February 10, 2020 – December 18, 2020		
Thesis committee:	Dr. ir. J. Ellerbroek	Supervisor	[TU Delft]
	Prof. J. M. Hoekstra	Chairman	[TU Delft]
	TBA	?	[TU Delft]

An electronic version of the thesis **will be** available at http://repository.tudelft.nl/.

TU Delft

Abstract

The following is the preliminary report for the MSc. thesis *"Neural Network based Emergence of reactionary Motion Improvisation"* of H. N. Basien, under supervision of Dr. ir. J. Ellerbroek.

It comprises of a practice oriented design research, set within the entertainment industry context of the theater drone-dance show *AI-Man*, in collaboration between the dance company *Another kind of Blue (AkoB)* and drone consultants *CompactCopters (CC).*

The premise of the research is to create natural human-machine interaction by utilizing neural networks and motion capture data, enabling a drone swarm resembling a human form, the 'Airman', to have 'free-will' and interact with a dancer on stage. Based on this the following research objective is formulated:

> *'to design an algorithm that generates a virtual skeleton in reaction to motion-captured dance improvisations [...], by comparing existing and creating new algorithms, within the domain of deep-learning architectures, capable of 'fooling' the creator of the original data, beyond reasonable doubt. '*

The literature study has shown that a research gap exists with respect to using neural networks for the motion synthesis of two interacting entities. To tackle the issue at hand a grounded theory approach will be deployed to research, test and refine various neural network architectures found in literature and confront their applicability and performance with one another.

The resulting final model will comprise of an improved versions of the best models tested, combining them where appropriate. This model will generate reactionary motion based on real-time motion capture input. Accompanied by a virtual 3D visualization environment, it will create a showcase for live interaction with a dancer.

Within the arts mathematically 'better' is not necessarily more 'natural' or 'beautiful'. So to bridge the gap between objective and subjective evaluation, at the cutting edge between art and science, a final experiment with the choreographer and dancers of AkoB will be held where they are to judge if a given output motion is 'Real' or 'Generated'. This experiment is to validate the underlying research hypothesis:

> *'Neural networks are capable of generating natural motion in reaction to live full-body human input, which is indistinguishable of real motion, for a human expert.'*

Thereby proposing the *'turing-test'* for motion synthesis, with the underlying assumption:
"If it is good enough to fool the creators, it will be good enough to fool the general public."

After successful completion the results of this research will form the basis for the virtual dancer, the *'AI-Man'*, comprised of 12 drones forming a human skeleton, for the new showpiece of AkoB to be seen in theaters by the end of 2021.

Contents

Abstract III

Table of Contents IV

Nomenclature VI

List of Figures VIII

List of Tables X

I Project Context 2

1 Introduction 4
1.1 Report Overview . . . 4
1.2 Industry Context . . . 4
1.2.1 Past Projects: . . . 5
1.2.2 Additional Challenges: . . . 5
1.3 Research Summary: . . . 5

2 Research Overview 6
2.1 Research Questions . . . 6
2.2 Research Objective . . . 7
2.3 Research Hypothesis. . . . 7
2.4 Research Framework . . . 8

II Literature Study 10

3 Literature Review 12
3.1 Fields of Research . . . 12
3.2 Character Animation . . . 13
3.2.1 Animation Concepts . . . 13
3.2.2 Animation Techniques . . . 13
3.2.3 Human Motion Modeling . . . 16
3.3 Deep Learning . . . 16
3.3.1 AI Fundamentals . . . 16
3.3.2 The ”Black Art” of Machine Learning . . . 17
3.3.3 Selection of Methodology . . . 17
3.3.4 Motion Synthesis & Analysis . . . 17
3.3.5 The Future of Motion Synthesis . . . 18
3.4 Literature Analysis . . . 18
3.4.1 Statistics . . . 18
3.4.2 Most applicable Papers . . . 19
3.4.3 Reflection on Analysis . . . 19

4 Methodology 22
4.1 Grounded Theory Approach . . . 22
4.2 Neural Network Architectures . . . 23
4.2.1 Overview . . . 23
4.2.2 Multilayer Perceptron. . . . 25
4.2.3 Recurrent Neural Networks . . . 27
4.2.4 Convolutional Neural Networks . . . 29
4.2.5 Encoder-Decoder Networks . . . 31
4.2.6 Generative Adversarial Networks . . . 32

4.2.7 Phase-Functioned Neural Networks . . . 34
4.3 Data Pre-processing . . . 35
4.3.1 Derivation . . . 35
4.3.2 Transformation . . . 36
4.3.3 Data Augmentation . . . 37
4.4 Hyperparameter Optimization . . . 37
4.4.1 Grid Search . . . 39
4.4.2 Bayesian Optimization . . . 39
4.5 Human Perception Experiment . . . 39

III Project Plan **42**

5 Development Plan **44**
5.1 Data Handling . . . 44
5.1.1 Data Acquisition . . . 44
5.1.2 Final Dataset . . . 46
5.1.3 Data Pre-processing . . . 47
5.2 Model Development . . . 49
5.2.1 Framework & Hardware . . . 49
5.2.2 Model Design . . . 49
5.2.3 Training & Optimization . . . 49
5.2.4 Model Comparison . . . 50
5.3 Live Interaction . . . 51
5.3.1 'Online' Modifications . . . 51
5.3.2 Simulation . . . 51
5.3.3 Safety Design . . . 51
5.4 Validation . . . 52
5.4.1 'Human-Discriminator' Test . . . 52
5.4.2 Live Inference with Dancer . . . 52

6 Project Planning **54**
6.1 Milestones & Deliverables . . . 54
6.2 Work-Packages . . . 55

7 Conclusions **56**
7.1 Expected Results . . . 56
7.2 Research Gap . . . 57
7.3 Project reflection . . . 57
7.4 Future Development - What is next? . . . 57

IV Addendum **58**

Bibliography **60**

A Machine Learning Algorithms Overiew **66**

B Data-processing Framework **68**

C Gantt Chart **70**

Nomenclature

Abbreviations:

DOF	Degree Of Freedom
GT	Grounded Theory
TBD	To Be Determined

Machine/Deep learning:

AI	Artificial Intelligence
BHPO	Bayesian HyperParameter Optimization
BP	BackPropagation
BPTT	BackPropagation Through Time
CNN	Convolutional Neural Network
DL	Deep Learning
GAN	Generative Adversarial Network
GRU	Gated Recurrent Unit
HP	HyperParameter
HPO	HyperParameter Optimization
LSTM	Long Short-Term Memory
MHU	Modified High-way Unit
ML	Machine Learning
MLP	Multilayer Perceptron
MSE	Mean Squared Error
NN	Neural Network
PCA	Principle Component Analysis
PFNN	Phase-Functioned Neural Networks
ReLU	Rectified Linear Unit
RHN	Recurrent Highway Unit
RL	Reinforcement Learning
RNN	Recurrent Neural Network

Entities:

AkoB	Stichting Another kind of Blue
CC	CompactCopters UG
EU	European Union
NL	Netherlands

File-types:

BVH	BioVision Hierarchy
C3D	Medical motion experiment format
CSV	Comma Separated Values
FBX	Filmbox
TAK	OptiTrack 'take'
TRC	'Motion Analysis' filetype

Hardware:

CPU	Central Processing Unit
GPU	Graphics Processing Unit
PC	Personal Computer
VR	Virtual Reality

Character Animation:

FK	Forward Kinematics
IK	Inverse Kinematics
MoCap	Motion Capture

Software:

FPS	Frames Per Second
NPC	Non-Player-Character
UDP	User Datagram Protocol

[1]

[1]For PDF readers: Any abbreviation in the text is equipped with a hidden hyper-link to this page, like so: EXMPL

List of Figures

2.1 Research Framework for 'AI-Man' . . . 8

3.1 Skeletal rig used by AkoB . . . 15
3.2 Temporal distribution of examined literature . . . 19
3.3 Annual publications of literature related to 'Neural Network's . . . 20
3.4 Distribution of Tags used in examined literature . . . 21
3.5 Occurrence of authors in examined literature . . . 21

4.1 Multilayer Perceptron - Example Architecture . . . 26
4.2 Recurrent Neural Network - Architecture . . . 28
4.3 Long Short-Term Memory - Unit architecture . . . 28
4.4 Gated Recurrent Unit (fully gated) - Unit architecture . . . 29
4.5 Modified High-way Unit - Unit architecture . . . 30
4.6 Convolutional Neural Network - Example architecture . . . 31
4.7 Auto-Encoder - Example architecture . . . 32
4.8 Generative Adversarial Network - Example architecture . . . 33
4.9 Phase-Functioned Neural Network - Network architecture . . . 34
4.10 Hyperparameter optimization flowchart . . . 38

5.1 AkoB Dataset - Key Metrics . . . 46
5.2 AkoB Dataset - Distribution of durations . . . 47
5.3 AkoB Dataset - Distribution of labels . . . 48
5.4 Data-processing Framework Proposal . . . 50

2

[2]Any Figure that does not have an explicit 'Source' mentioned, under the caption, has been custom made for this report.

List of Tables

3.1 Overview of important animation concepts/terminology 14
3.2 Most important papers and their applicability 20

4.1 Overview of applicable neural network architectures 24

5.1 Overview of available motion data file formats 45
5.2 Comparison of common MoCap datasets 49

I

Project Context

1

Introduction

The proposed practice-oriented design is to be executed as the MSc thesis research project of *H. N. Basien*, under the supervision of *Dr. ir. J. Ellerbroek*, assistant professor at the faculty of Aerospace Engineering at Delft University of Technology.

The following research project plan is set within the context of the **AI-Man** project, in collaboration between *Another kind of Blue (AkoB)* and *CompactCopters (CC)*. The AI-Man project intends to create a duet between man and machine by means of utilizing drone-swarm and AI technologies. Ultimately it will result in a theater performance to be enjoyed by audiences all over the world. For this specific project a human motion predictor will be generated, to be utilized as the drive algorithm for the prior mentioned drone swarm, to represent an artificial 'dancer'.

1.1. Report Overview

This preliminary report represents the basis for the continuous process that is the planning of the MSc research project and is structured into three main parts:

1. Project Context: General background and objectives of the research
2. Literature Study: Review of the current state-of-the-art in the field and the methodology to be used
3. Project Plan: Setup of the development phase and general plan of action

The initial chapter 1 gives the reader an introduction to the research and its industry context. The external and internal goals of the research are formalized in chapter 2, outlining the overall scope. To enable deeper insight into the theoretical background and fields of research, it is followed by a literature review summary in chapter 3. The detailing of theories and methodologies to be applied, based on the findings in literature, are highlighted in chapter 4, followed by their intended practical application in chapter 5. To ensure a timely and feasible execution of the outlined planning, chapter 6 will clarify the work-packages, milestones and deliverables required to attain the research objective. The plan is concluded by a the expected outcome of the research and a critical self-reflection in chapter 7.

1.2. Industry Context

The research to be performed will contribute to the modern dance performance *'AI-Man'*, by *Another kind of Blue*, to be shown in theaters to general audiences in 2021.

The choreographer's greater motivation for the proposed research can be summarized by the following set of philosophical questions:

> *What is free will? Do humans have it? Can machines have it?*
>
> *David Middendorp - Choreographer, Another kind of Blue*

Without trying to find a final answer, yet exploring these questions, it was chosen to create a duet between a drone swarm and a human, where both have as much 'free will' as possible.

1.2.1. Past Projects:

The AI-Man project is the logical conclusion of two previous drone-dance projects created by AkoB & CC, over the course of the past 5 years:

1. Newton's Duet (2015): http://y2u.be/fl20Mxd8vLc
   - 2 drones and 2 dancers
   - pre-planned choreography
   - tour around NL
2. Airman (2018): http://y2u.be/NuP4YhMzt3k
   - A swarm of 12 Drones representing the human form copies the improvisation of 1 Dancer
   - pre-planned choreography + interactive improvisation
   - tour around NL + global shows

AkoB had been experimenting with the usage of aircraft and drones way before this, however the earlier attempts where all manually controlled, before the transition to full automation was made.

All of these previous development steps have explored man-machine-interaction in an artistic environment, with various levels of 'interactivity'; though none has yet achieved a true 'duet' in the classical sense of the word: An intricate dance between man and machine where both partners can be 'free' in their decisions of motion and yet bound to each other by a common dance language to complement each others expression. To achieve this, an 'artificially intelligent' larger-than-life virtual dancer (*'output-dancer'*) is to be created that can react to a human dancer (*'input-dancer'*), by means of a neural network that transforms the live-recorded motion-capture data into a reactionary dance.

1.2.2. Additional Challenges:

Aside from the fact that the drive algorithm needs to reflect artistic dance motions, due to real-life constraints, there are also certain additional technical & safety factors to be concerned:

- Constraints
  - **Scaling** of virtual dancers (approx. x2)
  - **Limited** stage/movement **space** - ca. 8x8x8 $[m]$
  - Limited **Vehicle Dynamics** - Inertia & control-loop of drones restrict flight-path execution
- Safety
  - Dynamic **Restricted areas** - e.g. space occupied by dancer
  - **Maximum speed** (close to the boundary), to avoid uncontrolled flight into restricted areas and limit kinetic energy
  - Large **Audience**, which shall not be harmed under any circumstances!

Although most of these safety concerns need to be addressed only once the transition to the physical hardware is made, they should be kept in mind when designing the drive algorithm.

1.3. Research Summary:

In this research insight into the mathematical representations of the human shape, as well as state-of-the-art knowledge of neural networks is gathered. Combing these will yield a novel online virtual motion synthesis algorithm, which at a later stage will be transfered and interfaced with the drone hardware.

The contribution to the scientific community will lie in the structured comparison and benchmarking of various NN architectures, data-processing & optimization techniques, for the application of motion synthesis, while the contribution to AkoB and the artistic/cultural community will be the ability of natural man-machine interaction to generate intricate dance interactions, to delight audiences around the world with a stunning visual performance.

2

Research Overview

The external goal of the research is focused on the development of an algorithm framework for generating reference positions for a drone swarm, that resemble natural motions of an interactive dancer.

The primary issue is however that there is not a clear-cut answer to when a neural network is 'good enough' or has 'natural output'. Even though a reduction in the model's loss-function on the test & validation data-sets may indicate an improvement in performance, this is not necessarily the case as the research context is set within the arts: Mathematically 'better' outcome is not necessarily represented by a 'nicer', more 'natural' or more 'beautiful' result. Finding a solution to this ambiguity is essential to AkoB to deliver a stunning visual performance for their international audiences.

The internal goal of the research is therefore to solve this ambiguity between objective and subjective evaluation and how to design a NN architecture that can in fact achieve the desired output 'transforming a dancers improvised motions into appropriate reactionary motions'. More specifically various NN architectures and processing methods will be compared quantitatively and qualitatively to establish a sound scientific basis for future development in the field. Finally an attempt will be made to de-mystify some of the internal black-box characteristics of various NN models, by gaining a deeper understanding of the in- & output data-streams of each module and their relevance.

2.1. Research Questions

The following set of research questions guide the research trajectory; some questions have a direct external applicability that will be highlighted in brackets:

1. Fundamental Research: Which deep learning techniques are applicable to handling motion capture data?
   (a) Architectures: Which deep learning model architectures are fundamentally capable of handling motion capture data?
   (b) Loss function: How is the loss-function/error between frames to be defined?
   (c) Loss function: How are the various parameters to be weighted?
2. Data Transformation: Which mathematical domain yields the best results in terms of human insight into motion data? *[Better insight into the underlying data may result in improved data dimensionality reduction and runtime performance increase.]*
   (a) Does transformation into the frequency domain improve insight into motion data?
   (b) Does transformation into the Laplace domain improve insight into motion data?
   (c) Does principal component analysis improve insight into motion data?
   (d) Does transformation of Euler rotations into alternative representations (e.g. quaternions) improve the training results? [1]
3. Optimization: Which factors influence the loss function and to what extent?
   (a) Input Data: How does the total amount of input data affect the loss function? *[When is enough data captured for the achieving the desired output?]*

[1] This should avoid issues such as 'gimbal-lock' and cyclical ambiguity. See section 4.3.2

(b) Data Handling: To what extend does preprocessing of the input data improve the loss function?
   i. Time-slices: How does the number of input time-slices affect the loss function? *[Larger input data results in a more complex network and reduces runtime performance.]*
   ii. Derivatives: Does pre-computation of time derivatives improve the loss function?[2]
      A. What derivative order can be filtered from the available motion capture data, while retaining more signal than noise?
      B. How does the addition of derivatives to the input affect the loss function?

(c) Does prior knowledge improve the final results through 'transfer learning'?
   i. Does unsupervised pre-training improve the loss function?
   ii. Does unsupervised pre-training reduce the total amount of data required?

(d) Hyper-parameters: Whats the best method for optimizing the chosen deep learning architecture, when considering the ratio between loss function improvement and total training time?
   i. Which are the hyper-parameters most influencing the loss function?
   ii. What's the best strategy for optimizing these hyper-parameter?

4. Output-applicability: How to ensure that the model results can be applied within the real-world constraints of the show 'AI-man'?

(a) Which technical requirements does the system need to adhere to, to ensure a successful show? *[If these are not met the external validity of the research is in question!]*

(b) Does the model over-fit to an individual dancers body language? *[Will training on dancer A as 'input dancer' be transferable to inference with dancer B?]*

(c) How to ensure temporal continuity within the output data? *[Sudden jumps in position or momentum can be a potential result, which the drones given their inertia will be unable to follow!]*

(d) How is a human experiment to be designed, where the test subject needs to discern between 'Real' and 'Generated' motion data?

The presented research aims to fulfill the internal goal, by finding well-reasoned answers to these questions.

2.2. Research Objective

The research objective of this practice-oriented the master's thesis is:

> "to design an algorithm that generates a virtual skeleton in reaction to motion-captured dance improvisations, as part of the 'AI-Man' showpiece of 'Another kind of Blue' , by comparing existing and creating new algorithms, within the domain of deep-learning architectures, capable of 'fooling' the creator of the original data, beyond reasonable doubt."
>
> *Research Objective*

2.3. Research Hypothesis

To bridge the gap between objective and subjective evaluation a final test is to be issued, where the experts of AkoB will be shown various motion outputs, either ground-truth data or generated data, and are to decide if the data shown in real or fake.

The following research hypothesis is to be proven:

> "*'Neural networks are capable of generating natural motion in reaction to live full-body human input, which is indistinguishable of real motion, for a human expert.'*"
>
> *Research Hypothesis*

If this test results in a distribution of choices close to 50/50 'random' guessing and can be asserted with statistical significance the hypothesis can be regarded as confirmed.

[2] No derivatives and a singular time-slice are expected to yield bad results, as momentum and directionality is completely lost.

2.4. Research Framework

The proposed graphical research framework for this research is presented in Figure 2.1. It presents a top-level overview of the tasks and results of the research, as well as their interrelations.

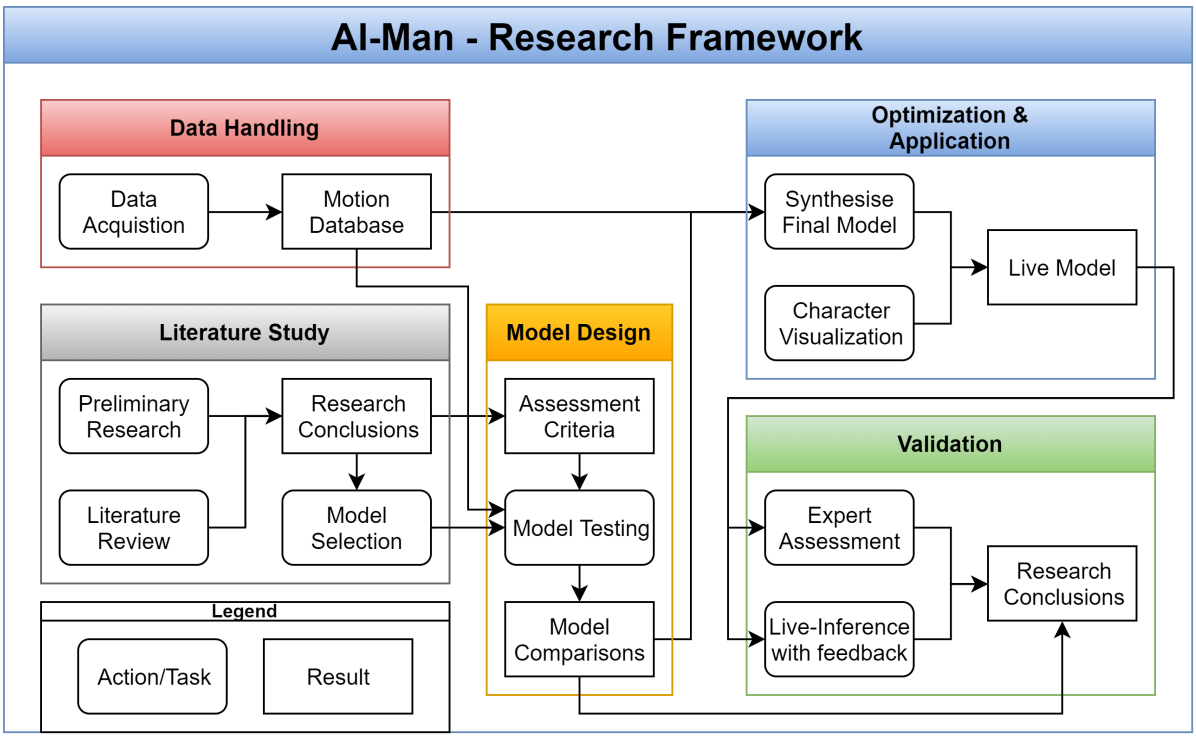


Figure 2.1: Research Framework for 'AI-Man'

II

Literature Study

3

Literature Review

This chapter is dedicated to the review process of state-of-the-art literature related to the project.

It will present work that has already been carried out by other academics in this area, while reflecting on the industries best practices. The researcher first establishes which research areas are relevant, and subsequently reflects on these, with regard to the current understanding, along with any opposing views.

For now this chapter will focus more on the general findings and qualitative assessment, while the next chapter 4 will be focused on the specific methods and techniques, which were extracted from literature to be applied in this research.

3.1. Fields of Research

Given the novel and unconventional approach that AkoB takes towards new developments in the modern dance scene, combining dance & technology, the project at hand is a highly multi-disciplinary project. The background for this project primarily comprising of the following 3 disciplines and related key concepts:

- Computer Science:
  - Artificial intelligence (AI) / Neural networks (NN)
  - Character Animation/Motion capture (MoCap)
  - Signal processing
- Performing Arts:
  - Dance / Choreography
  - Anatomy / Human motion modeling
- Aerospace Engineering:
  - Signal Analysis
  - Drones / Vehicle dynamics
  - Control Theory

Reflecting on these key concepts it is clear that there are vast fields of knowledge that could be researched, beyond the scope of this research. It was decided to focus on a few specific fields, with regard to the requirements of this particular research project:

The primary focus of this research will be the design of a novel neural network for motion synthesis, therefore the focus of this literature review lies primarily within the 'Computer Science' discipline and reflects on state-of-the-art applications of various NNs in the field of character animation.

With regard to the expertise in the field of dance from a qualitative perspective, it has been decided to leave this primarily at the discretion of the professionals at AkoB, thus this research has been restricted to the field of 'human motion modeling' to analyzing how dance and motion can be effectively represented quantitatively.

Given the fact that the current research will be limited to the designing and virtual testing of the motion control system, combined with the fact that the drone hardware and related knowledge is already established from a prior project; it was decided to omit any further research into this field.

In the following first character animation as a whole and its various applications is presented, followed by a reflection of the application of NNs in the field of motion synthesis.

The chapter will be concluded by a quantitative reflection on the completeness of the literature review conducted.

3.2. Character Animation

The research of predicting and analyzing motion using skeletal data falls under the umbrella term of 'character animation'. Various sub disciplines fall under this concept. The general background and applications of character animation has been explored to see which practices can be applied to this research.

Character animation in general is the process of animating artificial entities, generating an illusion of motion and personality in an otherwise static character. Originally animation was done by means of pen and paper, this 2D background, as well as early cinematography, manifested the concept of the discretization of motion into 'frames'.

Evolving from this, 3D character animation is the field of computer aided animation of humanoid and non-humanoid characters, via the manipulation of character meshes via a customized character rig.

For the remainder of this research only 'modern' 3D character animations will be considered.

In practice there are various real-world disciplines in which virtual character animation is being utilized:

- Movies/Videos - Pre-rendered animation of humanoid & non-humanoid characters [Chan et al., 2019; Holden et al., 2015, 2017b; Yamane et al., 2010]
- Gaming - Real-time adaptive character animation and NPC behavior [Bleiweiss et al., 2010; Holden et al., 2017a; Komura et al., 2017; Pejsa and Pandzic, 2010; Starke et al., 2019; Wang et al., 2020b]
- Industrial Simulations - Production plant or crowd simulations [Gaisbauer et al., 2019a, 2020]

As the lines between the applications of movies, 3D games and simulations are starting to blur more and more, it is easy to find various similarities in these fields of 3D character animation.

3.2.1. Animation Concepts

Before getting started on the specific techniques a few of the most important animation concepts are highlighted in table 3.1, to gain a better understanding of these elements when they're used freely in the subsequent text.

Figure 3.1 showcases the skeletal rig that was used in the recording of the dataset by AkoB.

3.2.2. Animation Techniques

There are a few general 'conventional' techniques utilized in the generation of character animations, which are highlighted below.

Rigging

Rigging is the process of preparing a mesh for animation. It involves either fitting an existing rig to a character, or creating a new skeletal rig for a novel type of character. This process is not limited to humanoid characters, but can be applied to various non-humanoid characters, such as dogs, monsters or even inanimate objects, such as flowers and trees e.g. to animate them moving in the wind [Baran and Popović, 2007; Shen and Yang, 2016].

Keyframing

Keyframing involves a designer manually setting a pose for certain frames within an animation. In the most extreme case this is done for every frame by hand. This is analogous to how in 2D animation every frame is drawn with only a slight change to the previous frame to create the illusion of motion. For modern animation it is common that only a few key frames are set manually and the computer automatically interpolates between the key frames smoothly.

To aid the designer in manual manipulation of the rig various techniques are applied. The most common are FK and IK:

Concept	Description
Frame	A discrete timestep snapshot of a scene
FPS	'Frames per second', the update rate of an animation 8-12 FPS is considered the min. for perceiving 'motion', 24 FPS has been the standard for movies, 50-60 FPS is the modern standard for animation and considered to be the limit of human perception [Holcombe, 2009]
Mesh	The 3D tessellated model defining the structure of a character
Rig	The skeletal structure of a character defining its motions A rig can be applied to a mesh to create animated character
Root	The root of a rig defines the center point to which linear and rotational transformations are applied
Bone	A bone represents a static transformation of the root or joint toward the next joint while rotations can usually be freely manipulated bones represent a fixed euclidean distance between joints.
Joint	A point in the rig that can be freely transformed rotationally, it connects two bones together
End-effector	The last link in a kinematic chain (e.g. the hand or fingertip) Term originated from robotics where it represent e.g. the tool at the end of a robotic arm
T-Pose	Default motion capture stance - standing upright with both arms stretched out sideways at 90°
Kinematic Chain	A series of bones connected by joints that are linked together, e.g. an arm or a leg
FK	The process of computing the position of the end-effector, by forward propagation of relative angles on each skeletal joint
IK	Effectively reverse FK; the computation of joint positions and angles given a fixed end-effector position

Table 3.1: Overview of important animation concepts/terminology

Forward Kinematics (FK) is commonly used when the designer wants full control over the rig, as it allows him to control every single joint in the model freely. Working from the root of the rig, the rotations of each joint affect every other bone down the kinematic chain up to the end-effector.

Inverse Kinematics (IK) is commonly used in tasks where points of contact are particularly important. This can be in a walking animation, where gravity demands feet to be connected to the ground, or in a scenario where a character interacts with an object and its hand should e.g. touch the wall and not fade right through it. It also allows the user to manipulate characters more 'naturally', as all links in a kinematic chain move as the end-effector is moved [Holden et al., 2017b; Yamane and Nakamura, 2003].

Motion Capture

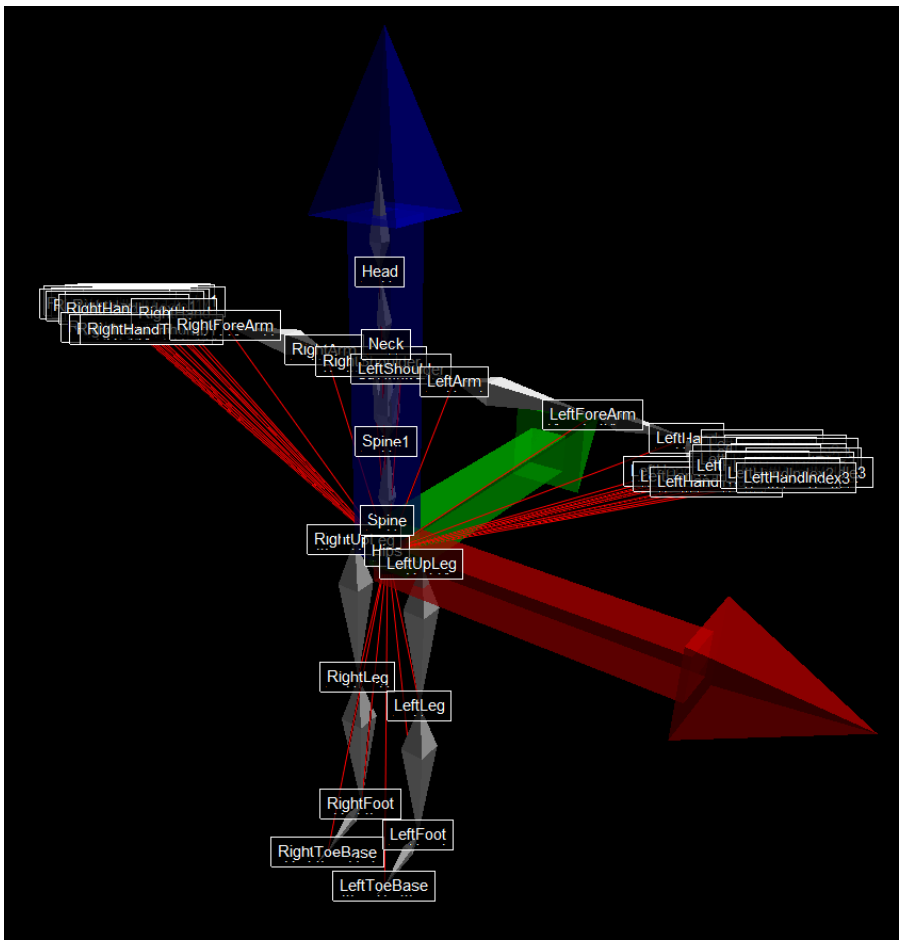
As creating handcrafted animation is usually a time consuming and thus costly process, more and more use is made of 'motion capture' (MoCap) studios to record motion live from a human subject. Various techniques can be used to record human motion, however the probably most common type of motion capture studio consists of a set of cameras and a set of markers distributed over the body of the person to be recorded.

MoCap allows for a quick and easy recording of complex motion that would be difficult to re-create by hand. The benefit is also that the recorded footage can still be augmented by hand-crafted edits. This is frequently required also due to the fact that MoCap recordings tend to be noisy sometimes and incorrect reconstructions of the underlying skeleton need to be corrected [Holden, 2018].

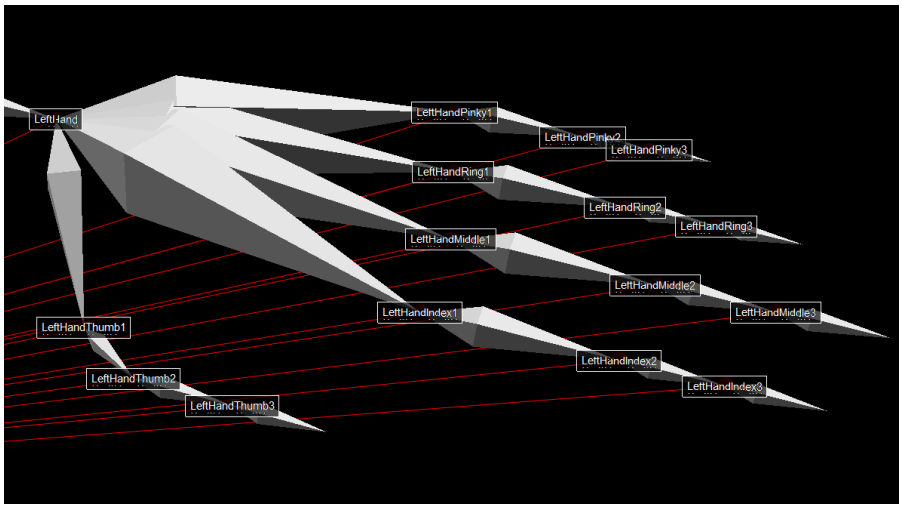
The motion capture studio that was used in the data acquisition for this research is owned by AkoB and holds an 'OptiTrack' tracking system, comprised of 16 'Prime 13' IR cameras, which records 54 reflective IR markers located on a Velcro suit.

Procedural Generation

Especially for interactive environments, such as games or simulations, research is ongoing for developing generative models to automatically create new motions to fit a scenario, without the need for carefully handcrafted



(a) Full body Rig



(b) Hand Rig

Figure 3.1: Skeletal rig used by AkoB

Each of the 51 *joints* is labeled individually
The *root* of the rig is located at the 'Hips'
Red lines go from the *root* to every *joint* as a visual guide
The white double pyramids connecting every *joint* represent the *bones* of the rig

animation cycles. This will be analyzed in detail in the upcoming section 3.3 on deep learning for motion synthesis.

3.2.3. Human Motion Modeling

While it is common and convenient to present the state of a rig in terms of the relative transformations of each joint, it is not the most efficient way of storing the data.

At least for us humans our joints have natural limits and usually our joints don't move completely uncorrelated to each other, commonly specific tasks require a coordinated effort of the limbs to achieve the desired effect. For example there is a high level of correlation between the movements of fingers on the same hand in most motions e.g. gripping. It is also possible that parts of the body move uncorrelated to each other the legs can follow a different goal than the torso and the arms, e.g. while walking several activities can be performed with the arms (holding something, texting, etc...).

The bottom line is that there are more efficient ways to represent human motion, with the means of utilizing various models. This field of human motion modeling can however result in vastly different models depending on the underlying motions it tries to represent.

Wang et al. [2019] explore the DOFs of the various parts of the human body and exploring their effects on pose-estimation tasks.

The development for a simplified human motion model for the dataset of AkoB is highly recommended and potential techniques will be explored in chapter 4.

3.3. Deep Learning - for Motion Synthesis

The research that has been done on deep learning (DL) techniques can roughly be classified into two segments, firstly the fundamentals and background of AI techniques and secondly the specific application of DL for motion synthesis.

3.3.1. AI Fundamentals

When examining the literature available it quickly becomes clear that, the interest in NNs and AI is accelerating and increasingly available to the 'general public'; with annual publications of scientific literature on 'Neural Network(s)' having more than tripled since 2015[1]. Even though the concept of an artificial neural network has conceptually been around since the 1940s and seen first practical application in the 1980s; it is largely thanks to the development of ever increasingly more powerful GPUs developed by e.g. NVIDIA [Huang, 2016] that enable this renewed interest in AI and DL in particular.

It is to be noted that the field of AI is commonly separated into the more 'traditional' ML and into the more 'modern' DL algorithms. An extensive summary of various ML and DL algorithms and their potential applicability to this research can be found in appendix A.

An attempt at an intuitive distinction of these two domains is made below.

Machine Learning algorithms are commonly characterized by a stricter rule-set and theoretical bounds. They generally allow a researcher to more easily modify and tweak the algorithm to achieve a desired result, but can be limited in their ability to learn general relations.

Deep Learning NN algorithms are, in theory, universal function approximators and allows the researcher a lot of freedom in the design and application on various types of data. However, this comes at the price of what's commonly referred to as the 'black-box' approach, which entails that its increasingly hard to understand the inner workings of a NN and even harder to change certain parameters by hand to achieve a desired result.

So despite the current enthusiasm on the subject, the design of neural networks still remains more art than science for the time being (see next section); while there are efforts made to develop empirical optimization models for e.g. hyper-parameter optimization, their application is not yet widely adopted and with limited benchmarking [Eggensperger et al., 2013]. Primarily there is no such thing as 'one-architecture-fits-all'. Every NN based project is highly data and architecture specific, with some architectures being only applicable to some types of data and showing widely varying results for given data types [Liu et al., 2017].

[1] Source: Scopus key-word search 'Neural Network'

3.3.2. The "Black Art" of Machine Learning

Furthermore it should be noted that the algorithms that people consider to be a part of the domains of ML or DL, are sometimes very different depending on the authors opinions. Some researchers consider DL as a part of ML, while others, usually in more popular articles, even freely interchange the terms AI, DL, ML,NN etc. to their liking.

In their great paper 'Troubling Trends in Machine Learning Scholarship', Lipton and Steinhardt [2018] call out (some) modern ML researchers for their lack in scientific rigor and are displeased in the 'overloading of technical terminology', 'speculation over explanation', as well as the trend of incorrect portrayal of AI in 'popular coverage'. In general it can be said that despite all the latest advancements, the field of AI research is still very much in its infancy and the need for more wide-spread standardized procedures, testing & benchmarking is omnipresent.

This fact is nicely summarized by this quote from a, slightly informal, paper by Domingos:

> "[...] developing successful machine learning applications requires a substantial amount of "black art" that is hard to find in textbooks."

Therefore the research, as presented in this report, will make a conscient attempt to...

(a) ...not fall victim to the four fallacies, as present by Lipton and Steinhardt [2018].
(b) ...separate the functionality and use of various stage in the model to be designed, instead of regarding it as a singular 'black-box'.

3.3.3. Selection of Methodology

Examining all ML and DL algorithms in existence in detail, is far outside the scope of this research. Given the fact that prior to the start of the research the premise was set to utilize DL for the task at hand, this is what the focus of the majority of the literature review is based on. However a brief analysis of the applicability of the various other fields, as present in appendix A, has been performed. During this analysis it became apparent that there are certain algorithms from other fields that could successfully augment the DL core; other fields that had been considered non-applicable, by the very nature of their algorithmic approach, have however been omitted from further research.

The algorithms that were deemed 'applicable' are highlighted in detail in chapter 4.

3.3.4. Motion Synthesis & Analysis

This section presents a general overview of the various domains of applicability of DL, with respect to their usage for the processing and synthesis of motion data.

While most relevant literature found is published within the last 5-10 years, it is interesting to see that even as far back as 25 years ago there has already been research on AI and its applicability for motion synthesis [Auslander et al., 1995]; however rudimentary it might have been, research in the field has come a long way since then.

Fields that have previously been complex and labor intensive are increasingly utilizing ML/DL models for automating tasks:

- Re-rigging of animations to new models [Holden et al., 2015, 2017b; Shen and Yang, 2016; Yamane et al., 2010]
- De-noising/Reconstruction of corrupted motion capture data [Cui et al., 2019; Holden, 2018]
- Natural animation of character (loco)motion [Gaisbauer et al., 2019b; Holden et al., 2017a; Starke et al., 2019; Thomas Geijtenbeek et al., 2013]
- Reinforcement learning for non-player agents, to simulate more natural non-scripted behavior [Peng et al., 2015; Thomas Geijtenbeek et al., 2013] [2]

As previously mentioned there is usually not a 'one-size-fits-all' solution when it comes to deep learning,so even within the field of character animation various different DL architectures and parameterizations are utilized [Komura et al., 2017; Pejsa and Pandzic, 2010], specifically with respect to the specific in- & outputs of the models.

[2] Reinforcement learning is not directly applicable to this research, yet a prominent and popular field of research for motion synthesis

A couple of discerning factors found are the following (highlighted in bold are the most applicable domains for this research):

- Static vs **temporal analysis**
- **Pure motion** vs Usage of 'external' inputs, e.g. environmental variables (terrain)
- Pre-processing tools vs **live inference**
- Reinforcement learning vs **fixed data-sets**
- Cyclical motion (e.g. locomotion) vs **generalized complex motion** (e.g. fighting)

3.3.5. The Future of Motion Synthesis

Initially only limited literature on the specific field of deep-learning for motion synthesis was to be found, however there is in fact a strong trend of academia and industry alike to explore the possibilities that this field provides. Some entertainment conglomerates such as 'The Walt Disney Company' even have their own research departments dedicated to advancements in animation and motion research [Yamane et al., 2010], among other subjects.

Final personal Thoughts

Speculating on the future of these fields of research, I belief that, in the near future, they may well lead us into a world where human interaction in VR may becomes immersive enough for e.g. life-like social scenario simulation, compelling interactive storytelling/gaming. Collaborations with mechatronics and robotics engineers, may well lead to more natural and adaptive physiological and facial motion patterns that could lead us into the 'uncanny valley' or beyond. This research is to be utilized on a drone swarm for entertainment purposes, but the more general notion of applying DL algorithms to learn general human interaction patterns may very well be the key enabling promising robotic technology, such as 'Boston Dynamics - ATLAS' or 'Hanson Robotics - Sophia' robots to make the leap towards a general purpose humanoid android.

3.4. Literature Analysis

To reflect on the literature that has been considered for this research, this section is dedicated to a high-level quantitative literature analysis. This is to ensure that not one particular topic or concept has been excessively studied, while others have been completely neglected.

A grand total of 90+ papers have been considered, of which little over half are directly related to the field of animation utilizing DL [3], the remaining are DL fundamental background papers and various other related literature.

For the management of the literature used for this research the reference management software 'Mendeley' has been utilized.

3.4.1. Statistics

For this literature review each paper has been analyzed and categorized by marking each paper with a set of tags that reflect the key concepts that are presented in the paper. Based on the resultant BibTeX file a custom analysis script was created to review the statistics of various quantitative properties, of which the most interesting results are highlighted below.

Temporal Distribution

Figure 3.2 showcases the temporal distribution of examined papers, clustered by year of publication. It showcases that the majority of papers on motion synthesis and DL have been published within the last 5 years The older papers are mostly historical papers that have been added, to investigate the origins and/or fundamentals of neural networks. In general it can be said that DL motion synthesis is a niche, but contemporary topic.

A more general comparison with publication data from Scopus, as seen in figure 3.3, shows that this corresponds to the trend of increasing number of papers on 'Neural Network's within the last lustrum.

Completeness of Research

In figure 3.4 the distribution of all tags, that were used more than once, is shown. Multiple tags per paper are possible, but a single tag can only be used once per paper. The bar labeled 'unknown' represents the amount of papers that have not been tagged.

By examining these the following conclusions can be made:

[3] This was lovingly dubbed 'AI-nimation'

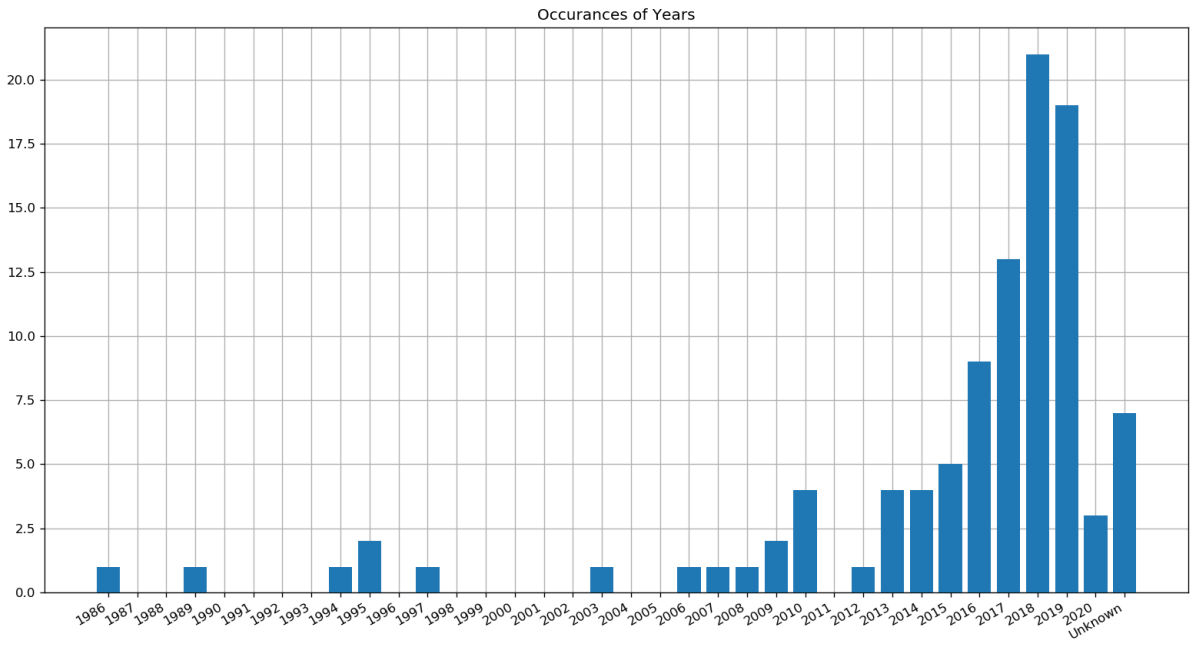


Figure 3.2: Temporal distribution of examined literature

- Sufficient resources have been considered, with regard to the primary NN architectures
- Plenty of papers have been found using standardized datasets for potential benchmarking
- A grand total of 16 papers with publicly available code repositories were found, to ease initial testing

Influential Authors

Figure 3.5 shows the occurrence of various reoccurring authors in the research.

Daniel Holden One author in particular has been standing out over the course of this literature review: 'Daniel Holden'. He might not necessarily be considered the most influential researcher on a larger scale, however this young inspiring machine learning researcher kick-started this literature research with his creative paper on 'Phase-Functioned Neural Networks' (PFNN), which has been recited by various other influential researchers in the field. Furthermore, Holden has published a variety of papers on various character animation and motion analysis related thematics, besides other interesting ML papers.

Some of these are highlighted on his own personal website theorangeduck.com/page/publications.

In case he is interested he seems like a great candidate when considering holding an (online) expert interview for reviewing the final research.

3.4.2. Most applicable Papers

The most applicable papers found are a select few that actually have direct applicability to the special features that define the current research. A summary of these papers is presented in table 3.2.

Ultimately it can be said though that only very few researchers and papers are directly applicable to this new research approach and that the specific information required needs to be extracted piece by piece from these various sources. This shows that there is clearly a research gap with respect to motion synthesis, with regard to the interaction of multiple people.

The details of how the concepts of these papers are relevant to the final research will become clear in the next chapter (4), which will present the usage of these methods in detail.

3.4.3. Reflection on Analysis

While it is to be noted that this reflection to a large extent can be a self fulfilling prophecy, in the sense that due to the small sample size and the largely iterative approach to this literature search, it can't really be seen as objective proof of completeness. As such it can indeed result in a singular researcher standing out, while being objectively less influential e.g. considering a smaller H-Index.

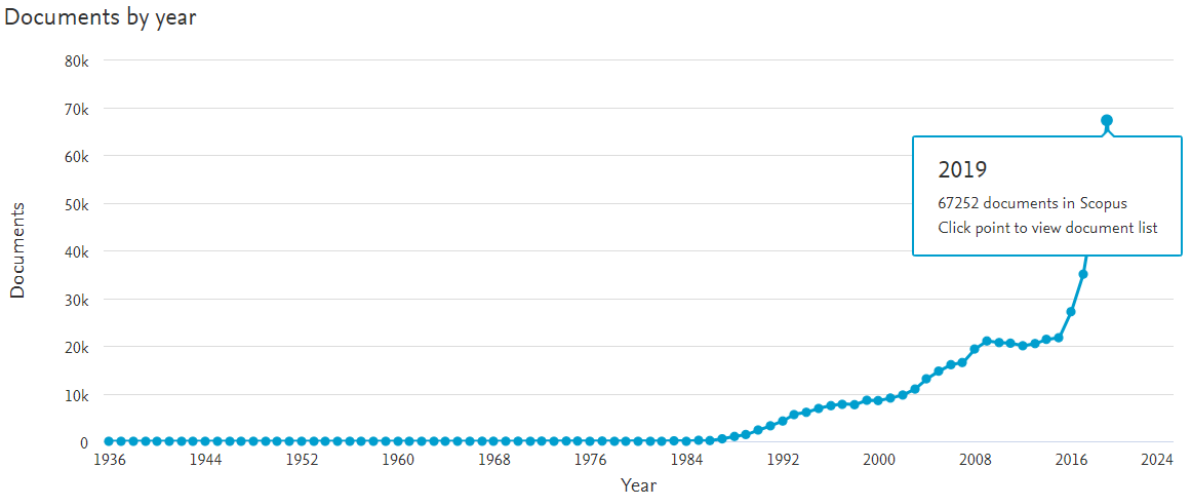


Figure 3.3: Annual publications of literature related to 'Neural Network's

Source: Scopus - Document search

Title Applicability	**Reference**
Two-person activity recognition using skeleton data Classification of 2 people interacting	[Manzi et al., 2018]
Adversarial learning for modeling human motion Combination of GAN and AutoEncoder	[Wang et al., 2020a]
Deep Motifs and Motion Signatures Identification of 'Leader' 'Follower' in interaction activities	[Aristidou et al., 2018b]
On the continuity of rotation representations in neural networks Usage of alternative rotational representation for DL	[Zhou et al., 2019b]
Modeling Human Motion with Quaternion-Based Neural Networks Comparison of various NN models and parametrization for motion prediction	[Pavllo et al., 2019b]
On retrospecting human dynamics with attention Visualization of important body components utilizing attention	[Dong and Xu, 2019]
Phase-Functioned Neural Networks for Character Control Accounting for distinct phases/frequency within data	[Holden et al., 2017a]
Self-similarity analysis for motion capture cleaning Automated cleaning of MoCap data	[Aristidou et al., 2018a]

Table 3.2: Most important papers and their applicability

However the goal of this reflection was less in objectively proving that 'all' literature has been considered and draw statistical conclusions from this, but much rather as a tool for self-reflection to assess if 'enough is enough'. This internal verification allows the researcher to move ahead to the practical phase given the knowledge that everything that appears to be needed has indeed been found.

The system of tags that was put into place to allow for this analysis will be beneficial when it comes to the actual development phase, as it allows the researcher to quickly and accurately consult those papers that are directly related to whatever methodology is currently being implemented.

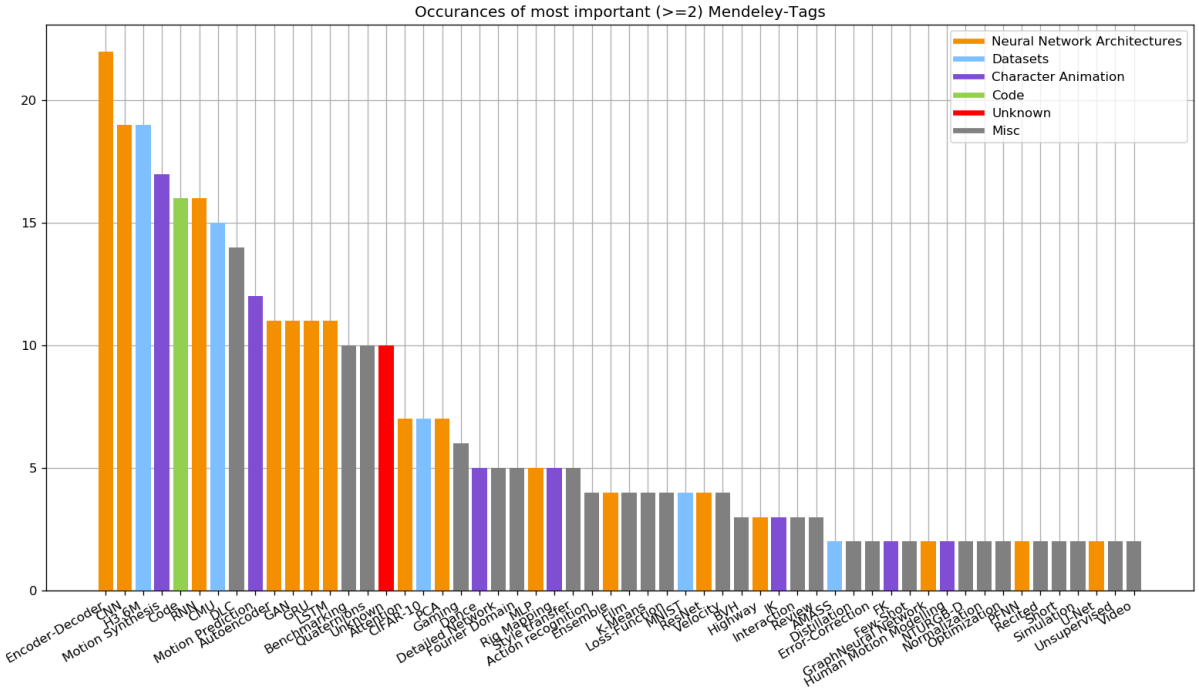


Figure 3.4: Distribution of Tags used in examined literature

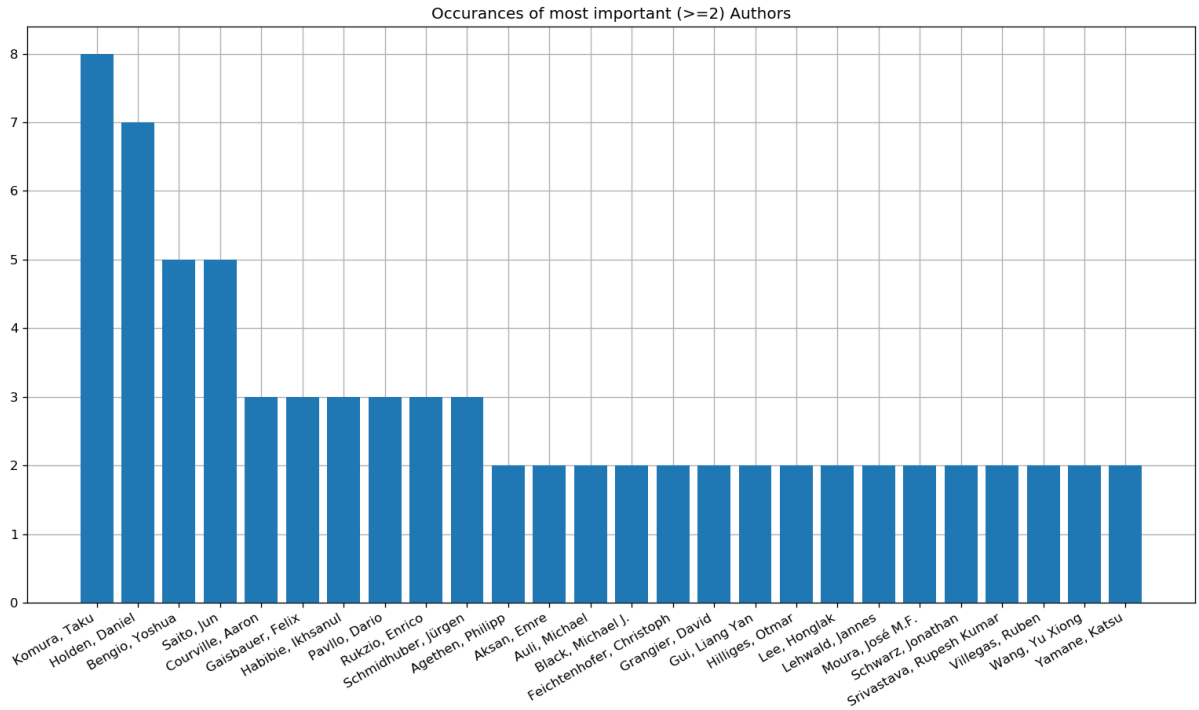


Figure 3.5: Occurrence of authors in examined literature

4

Methodology

Based on the background knowledge obtained and presented in chapter 3, this chapter highlights and explains the primary tools and methods to be utilized over the course of the research.

The chapter starts out with the usage of a 'Grounded theory approach' for guiding the research, followed by an overview of the various DL methods considered, as well as supplementary pre-processing steps and concluded by the rationale behind the final experiment that is to validate the to be developed model.

4.1. Grounded Theory Approach

Grounded Theory (GT) is a research framework that originates from the social sciences, but has recently seen more and more traction in the software development community, though granted in slightly altered variants, which are not always true to the original source methodology [Stol et al., 2016].

The basic rationale behind GT is the building of theory from data as a continuous process. It allows the researcher to observe data and draw conclusions based on similarities found in it and its derived attributes. GT also builds on the belief that the continuous process should not be tainted by prior belief and researcher bias. In some extreme cases it is said that the literature study should be delayed and that defining a research question should also not be done a priori. Regarding strict issues like this, it is important to note that for this research the researcher will use GT as a guideline rather than exactly following the rather specific GT doctrine.

The researcher has chosen to utilize GT methodologies for the following reasons:

- Currently there is no universal theory for how to handle MoCap data using DL
- Initially there is no specific set final goal of this research, rather a larger framework in which the research operates
- The research is highly data specific, so emerging phenomena will be a direct result of the dataset and its underlying properties
- Availability of a large dataset: Grounded theory -> 'Theory that is grounded in the data'

For the problem at hand it was decided prior that a neural network based approach will be chosen. As outlined in chapter 3, this does not entice a singular architecture or method, that is optimal for the task at hand, is predetermined. The design of neural network architectures are very heavily influenced by the underlying data used and are still very much more an art than an exact science. For example a study on the generalization of neural networks using the 'CIFAR-10' dataset has shown that even on a new dataset

> "as close to the original data distribution as possible"

most architectures have shown a significant drop in performance, as over-fitting on data can be a problematic phenomenon [Recht et al., 2018].

Therefore a *grounded theory approach* is utilized, iteratively applying various promising models from literature to the data at hand. The results obtained from these tests will be confronted with each other to result in either a singular, or a hybrid method between the methods discovered.

The following research strategies will be employed that are primarily influenced by the notion of GT.

- Sequential testing of promising DL models on the data, based on prior knowledge

- Continuous objective comparison of various DL methods and their applicability to the data at hand
- Continuous analysis of the data will allow for unexpected structure to arise in the latent space of the data, from which further conclusions could be drawn.

In contrast to the 'true' GT methodology the researcher has already drafted an extensive set of research questions (See chapter 2), as well as a preliminary data-flow diagram (See appendix B), however given that at the current stage only limited information is known on the dataset to be used, these research frameworks are to be a point of origin rather than a destination. Meaning that the researcher will not blindly stick to these frameworks no matter new insights, but rather see them as a living document that can be adapted[1] given new insight and meaning.

4.2. Neural Network Architectures

Based on the insights gain from literature, the following neural network architectures appear to be promising, for utilizing MoCap data.

First a quick overview of these architecture is given, before examining each of them in more detail with respect to their...

- Rationale: The primary thought behind the design of the network
- Architecture: The actual implementation and layout of the network[2]
- Relevance: The applicability of the design for the current research

Disclaimer: The following section is a rather detailed review of several NN architectures, therefore those readers familiar with the general workings of a particular architecture can feel free to directly skip ahead to the *Relevance* section of each architecture.

4.2.1. Overview

On the next page a quick overview of the NN architectures to be considered is presented in table 4.1, showcasing their most important advantages and disadvantages, as well as related papers.

[1] After consideration with the supervisor

[2] For most figures it was intentionally chosen to use images from 'Wikimedia Commons' to ensure they're in the common domain.

Advantages	Disadvantages
Multilayer Perceptron (MLP) [Li et al., 2019; Mar et al., 2017; Villegas and Lee, 2018; Zhou et al., 2019b]	
• Simplest network setup • Theoretical a 'universal function approximator' • Versatile in its usage	• Complexity increases greatly for additional nodes and layers • Easy to loose spacial awareness • Fixed input & output size
Recurrent Neural Networks (RNN,LSTM,GRU,MHU) [Bai et al., 2018; Chung, 2014; Dong and Xu, 2019; Ghosh et al., 2018; Goyal et al., 2016; Gui et al., 2018; Habibie et al., 2017; Martinez et al., 2017; Pavllo and Auli, 2018; Tang et al., 2017; Villegas and Lee, 2018; Zhang et al., 2018]	
• Temporally aware • Able to learn features of variable length • Allows for continuous 'sequence-to-sequence' learning	• Very long training times! • Usage of 'Truncated Backpropagation Through Time' (BPTT) could ignore large parts of the input data
Convolutional Neural Networks (CNN) [Aristidou et al., 2018b; Bai et al., 2018; Gatys et al., 2016; Hernandez et al., 2019; Holden et al., 2016; Li, 2018; Li et al., 2019; Mar et al., 2017; Pavllo et al., 2019a,b; Pratt et al., 2017; Shrivastava et al., 2016; Zhou et al., 2019a]	
• Allows for insight into the networks workings through inspection of the kernels • Sparse representation requires less learnable parameters than its MLP counterpart • Spatially aware	• Doesn't work for data of variable length • Up-convolution may result in rather 'blurry' data • Mostly used for visual/image data
Auto-Encoder [Ghosh et al., 2018; Habibie et al., 2017; Holden et al., 2016; Yan et al., 2018; Zhou et al., 2019a,b]	
• Unsupervised learning • Simple network setup • Enables dimensionality-reduction/compression of data, prior to feeding it into other models. • Allows for cloning of networks weights [(strictly) symmetrical networks]	• Delicate balance between 'blurring'/'generalizing' the output and 'over-fitting' • Additional processing step
Generative Adversarial Network (GAN) [Chan et al., 2019; Goyal et al., 2016; Gui et al., 2018; Hernandez et al., 2019; Kender and Way, 2018; Shrivastava et al., 2016; Wang et al., 2020a; Xu et al., 2018]	
• Usually high quality 'natural' output • Could be an option for a 'human in the loop' approach, by replacing the 'discriminative network' with the choreographer. • Can be used if there is no objective 'correct' data, just 'Real'&'Fake'	• Introduces additional complexity
Phase-Functioned Neural Network (PFNN) [Holden et al., 2017a; Wang et al., 2020b]	
• Allows for temporal component in data, by assigning a 'phase' to the data • Combats 'floating'[3] artifacts	• Only applicable to (mostly) cyclical motion with a single fixed phase: e.g. locomotion (walking, running) • Complex network weighting, due to blending between phase weights.

Table 4.1: Overview of applicable neural network architectures

4.2.2. Multilayer Perceptron

MLP are the earliest and simplest form a NN can take and are sometimes mistakenly referred to as any type of feed-forward NN. They are also commonly referred to as *'fully-connected'* networks.

Rationale

The basic rationale behind the idea an MLP is learning through recombination of features, the introduction of non-linearities through activation functions and learning through back propagation [Lecun et al., 2015]. The non-linearities allow it to transform the feature space of the input into an alternate representation, which allows it to e.g. apply a simple linear boundary to separate feature vectors belonging to complex input classes. This form of learning features/representations from data is called 'representation learning' and can be seen as the basis for the success of NNs as a whole [Bengio et al., 2013].

MLPs are most commonly used for classification tasks, but have since found application as a part of various other meta-architectures. For classification the input to the network is most commonly a normalized feature vector, describing the state of the object to be classified, and combined with a softmax output results in a set of probabilities for the classes to be classified into.

Architecture

The multilayer perceptron is unsurprisingly constructed by stacking multiple Perceptron units together in so called layers. The first layer is called the 'input' layer, analogously the last layer is called the 'output' layer, while any layers in between are referred to as 'hidden' layers. The term 'deep learning' originates from the capability of modern hardware to extend the classical perceptron to allow for more hidden layers and thus creating a 'deeper' network.

A perceptron unit comprises of a singular neuron, which can take any amount of inputs from neurons in the previous layer. It processes data in the following manner:

1. The *'inputs' (i)* are multiplied by the respective *'weights' (W)*, which denote the strength of the connection between two neurons
2. The scaled inputs are summed together
3. Optionally - an offset term called a *'bias' (b)* is added to this sum
4. The sum total is fed into a so called *'activation function' (f)*, a non-linear function distorting the feature space
5. The result of the activation function is called the *'output' (o)*, this can either be used as the input for another neuron or be the result of the output layer

Mathematically speaking the complete operation is represented by equation 4.1.

$$o = f(\sum_{k=0}^{n}(i_k \cdot W_k) + b) \tag{4.1}$$

Matrix operations However equation 4.1 only represents the formula for the output of a singular neuron, while commonly a layer consists of multiple neurons. To make the computations more efficient the entire layer to layer operation can be denoted by matrix operations, which is why running NNs on GPUs is so efficient. For a layer transition from m to n neurons, the respective matrices and vectors will have the following dimensions:

- i: $mx1$ - Column vector
- W: mxn - Matrix
- b: $nx1$ - Column vector
- o: $nx1$ - Column vector

The compact matrix form of equation 4.1 is given by equation 4.2.

$$o = f(W^T i + b) \tag{4.2}$$

A visual representation of this architecture is given in figure 4.1.

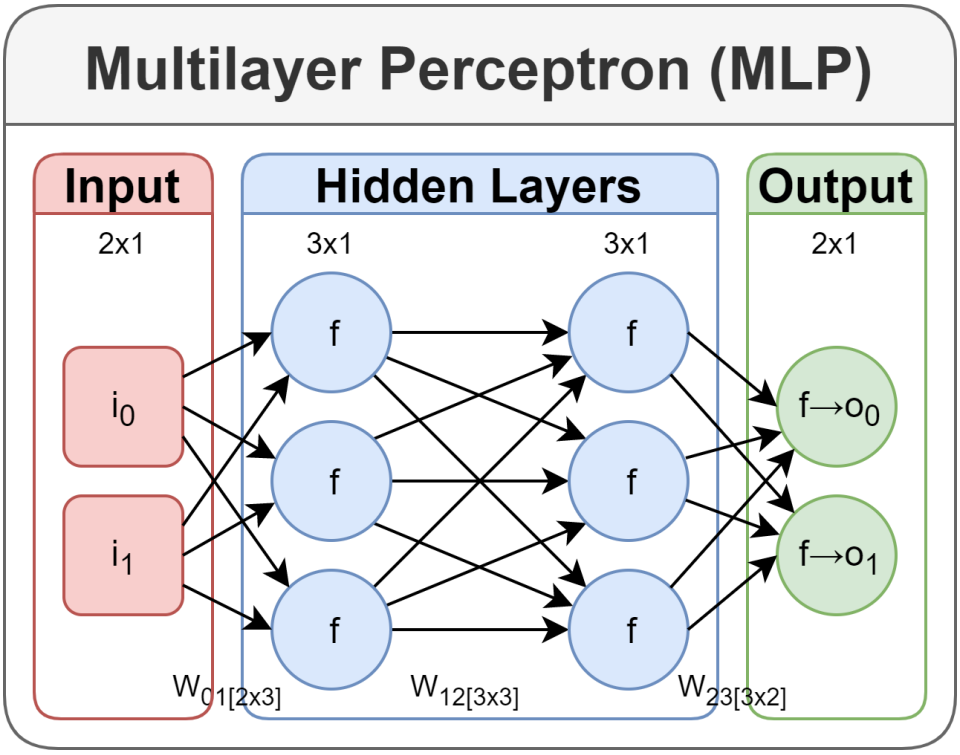


Figure 4.1: Multilayer Perceptron - Example Architecture

Learning For the network to actually learn and improve its predictions two stages are applied sequentially:

1. Feed-forward prediction
   →The feed-forward computation as denoted by equation 4.2
2. Backpropagation training
   →Minimizing the loss-function by propagating the error of the prediction vs the *'ground-truth'* through the network backwards, adapting network weights and biases to result in a better prediction

Summarizing an example BP algorithm, in which the following steps are executed:

1. Compute the error given a specific *'loss-function'*[4]
   (a) Subtract the ground truth from the prediction to get the raw error
   (b) Squaring the raw error to result in a function where 0 is the optimum
2. Attempt to minimize the error by applying an optimization technique, such as (stochastic) gradient descent
   (a) Compute the derivative from the activation function
   (b) Take a step in the negative direction of the derivative to move closer to the minimum, by changing the weights and biases in the opposite direction that contributed to the final error
   (c) Repeat for each layer

The examples above are simplifying a lot of the complex mathematics that is behind the processes and is primarily intended to introduce the concepts of *'activation-function'* *'loss-function'* and optimization algorithms such as *'gradient descent'*. There are many variations to these concepts and which activation-function, loss-function or optimization method is to be applied can heavily influence the performance of the network.

Relevance

With respect to motion synthesis the MLP is probably not the most applicable architecture, at least not directly. Indirectly however the MLP serves as a crucial building block for many promising meta-architectures, thus its

[4]Mean Squared Error (MSE) loss function used in this example

inner workings should be well understood before diving deeper into these derived architectures. With respect to its applicability for this research the most promising place to use MLPs would be in the following meta-architectures:

- The *encoder* and *decoder* network in an *auto-encoder*
- The *discriminator* in an *GAN*

4.2.3. Recurrent Neural Networks

Recurrent neural networks (RNN) are primarily used for sequence analysis and receive their name from the fact that the same nodes are being revisited recurrently at every new input, effectively resulting in a very deep network consisting of the same node(s).

RNNs date back to the publishing of Rumelhart et al. [1986] paper published in nature. Even though the original RNN still sees widespread usage for handling sequential data, various modern adaptations have been made since then; most notably the 'LSTM' [Hochreiter and Schmidhuber, 1997] and the 'GRU' [Cho et al., 2014]. Furthermore, a promising new recurrent unit, the 'Modified High-way Unit' (MHU), specifically designed for human motion data was proposed by Tang et al. [2017].

Rationale

The basic rationale behind the RNN is the ability to retain *'memory'* over time by storing the combined information of all previous inputs in its *'hidden state'*. At every step the previous hidden state, as well as the current input are used to compute the next hidden state.

Long Short-Term Memory The LSTM network as proposed by Hochreiter and Schmidhuber [1997], solves common problems of conventional RNNs, by replacing the simple neurons in an RNN by a more complex node.

Conventional RNNs heavily suffer from the vanishing or exploding gradient problem; as the same network weight is used at every time-step any value below 1 results in exponential decay of information over time, while any value larger than 1 results in exponential growth over time till ∞, eliminating the effect any new information has. This quickly vanishing radiant means that any memory from earlier time-steps is quickly 'forgotten', limiting its short-term memory.

To alleviate these issues the LSTM node, preserves/remembers or forgets previous information in a dynamic fashion. This allows the network to retain information over an extended period of time(-steps). Pre-'long'-ing the time the artificial short-term memory is retained.

Gated Recurrent Unit The GRU as proposed by Cho et al. [2014] takes the complex LSTM and explored if it was possible to reduce the complexity, without reducing its beneficial rational.

The result was a recurrent unit with fewer parameters than the LSTM, which retained similar performance.

Modified High-way Unit The MHU as proposed by Tang et al. [2017] is basically an adaption of the 'Recurrent Highway Unit' (RHN) [Zilly et al., 2017], specifically for motion prediction. The idea is to separately encode the past skeletal states and provide access to this encoded past to allow for better extraction of cyclical motions over a longer time-frame.

Architecture

The simplest of RNNs is presented in figure 4.2, the left representation shows the recurent nature of the RNN, while the right unfolded representation illustrates how data on previous states is propagated through 'time' via the vector denoted 'v'.

For any of the other adaptations the basic layout of the network rarely changes, it is primarily the internal functionality of the hidden node/unit that is exchanged.

Long Short-Term Memory The basic unit structure of an LSTM node is illustrated in figure 4.3.

At first glance the LSTM seems complex and difficult to comprehend, yet when taking a closer look at its individual elements a simple elegance emerges.

Three gates govern the data-flow through the LSTM (following figure 4.3 from left to right):

1. F_t: Forget gate
   Determined if the information in the cell state c is to be 'reset'

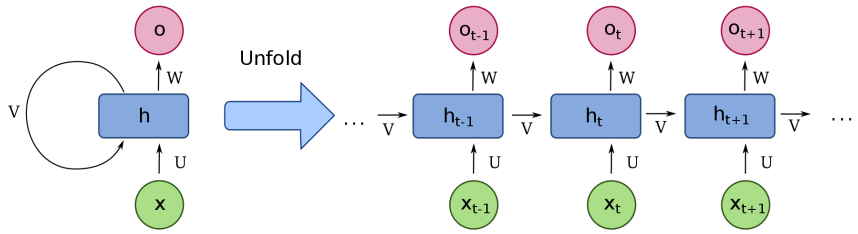


Figure 4.2: Recurrent Neural Network - Architecture

x: Input state, h: Hidden Node, v: Hidden/Cell state, o: Output state

Source: CC BY-SA 4.0 - https://commons.wikimedia.org/wiki/File:Recurrent_neural_network_unfold.svg

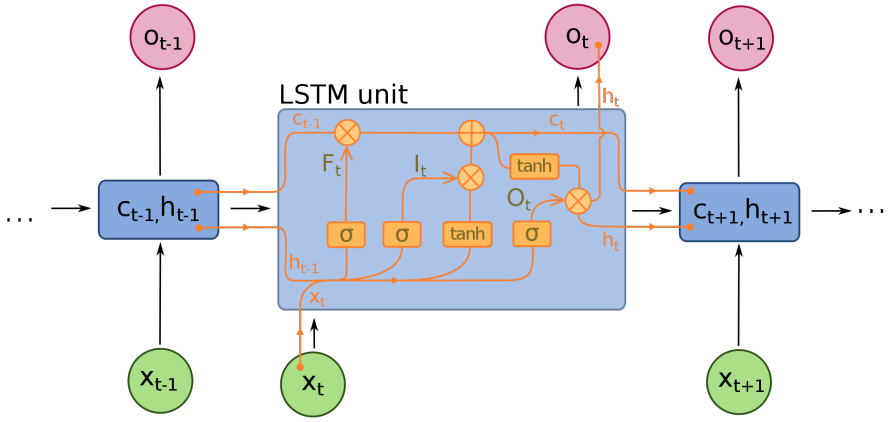


Figure 4.3: Long Short-Term Memory - Unit architecture

σ: Logistic Sigmoid function [0-1], $tanh$: Hyperbolic tangent $[-1\text{-}{+1}]$ | x: Input state, c: Hidden/Cell state, h/o: Hidden/Output state

Source: CC BY-SA 4.0 - https://commons.wikimedia.org/wiki/File:Long_Short-Term_Memory.svg

2. I_t: Input gate
   Determines what and how much of the new input and previous output $x\&h$ will be fed into the cell state c
3. O_t: Output gate
   Determines how much of the current state $x\&h$ will be passed through to the output state $o\&h$

Looking at this model intuitively it becomes clear how it is able to retain data over time, through dynamic gates, effectively learning when to 'remember' and when to 'forget'.

Gated Recurrent Unit The GRU is basically an umbrella term for various recurrent units utilizing a singular hidden state, as well as internal gates to regulated data-flow. As such there is no singular architecture to showcase. An example of the 'fully gated' version is shown in figure 4.4.

Modified High-way Unit Excerpt from paper by Tang et al. [2017]:

> "Each historical skeleton is first embedded into one semantic space. At each time step, the motion context modeling summarizes the skeleton embeddings with respect to the last predicted skeleton. Afterwards, MHU works on the motion context and the last estimated skeleton to yield the human motion at each time step."

The associated model is visualized in figure 4.5.

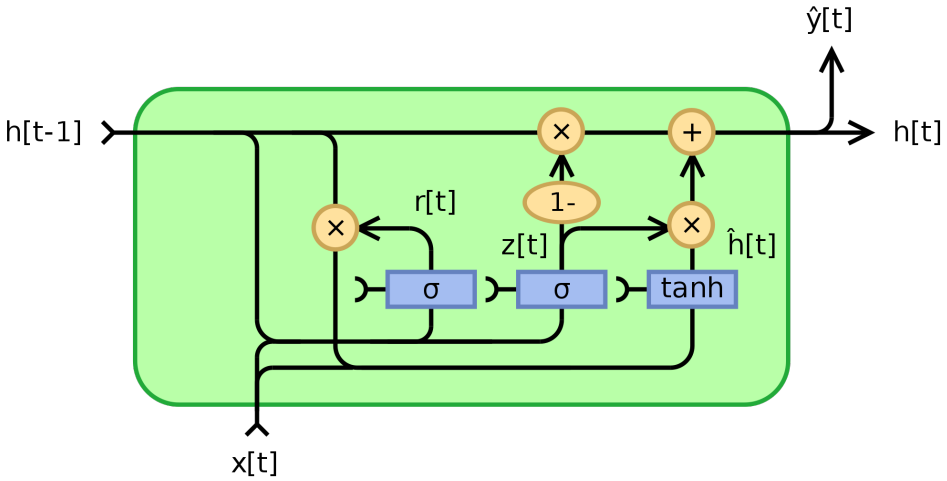


Figure 4.4: Gated Recurrent Unit (fully gated) - Unit architecture

σ: Logistic Sigmoid function [0-1], $tanh$: Hyperbolic tangent [−1-+1] | x: Input state, z: Update gate state, r: Reset gate state , h/y: Output state

Source: CC BY-SA 4.0 - https://commons.wikimedia.org/wiki/File:Gated_Recurrent_Unit,_base_type.svg

Relevance

RNNs seem like the obvious choice for handling motion capture data, as it allows for continuous sequence to sequence prediction, updating the model with each new pose every time-step. It has also been successfully used by various other researchers in the field (please refer to table 4.1).

While the original RNN does not appear to be sufficient to handle the long term relations required for most motion predictions the modern Units as presented prior seem very promising. With respect to comparing performance of LSTMs to GRUs on motion data it appears that ...

- ...GRUs allow for faster training times [Martinez et al., 2017].
- ...there is no benefit in using LSTM over GRU [Pavllo and Auli, 2018].

Especially the approach chosen by the MHU, to introduce a temporal attention unit appears very promising in its application for predicting two skeletons simultaneously, as it allows for cross-attention between the two dancers.

Training As there will be the need for continuous predictions in the final live-inference, basically predicting a new skeleton each time step from a live-stream; it raises the question how exactly the network is to be trained. When training on a full take from beginning to end before back-propagation this would create a network too deep to be handled by the current hardware. To handle this the 'BackPropagation Through Time' (BPTT) might be a valid solution, but further research into this is required.

4.2.4. Convolutional Neural Networks

Convolutional Neural Networks (CNN) are primarily used for image analysis and excel in tasks such as image classification or semantic segmentation.

CNNs have been around since the early 1980s, but recently have gained much attention due to their highly successful usage in the ImageNet competition in 2012 [Lecun et al., 2015].

Rationale

The basic rationale behind the CNN is the ability to comprehend images through the usage of several stages of feature detection. This is achieved through convolving the input with learned *'kernels'*, resulting in subsequent maps indicating where these features are prominent. This approach is based on common computer algorithms, where some of the simplest convolutional kernels can achieve useful tasks such as edge detection or blur operations.

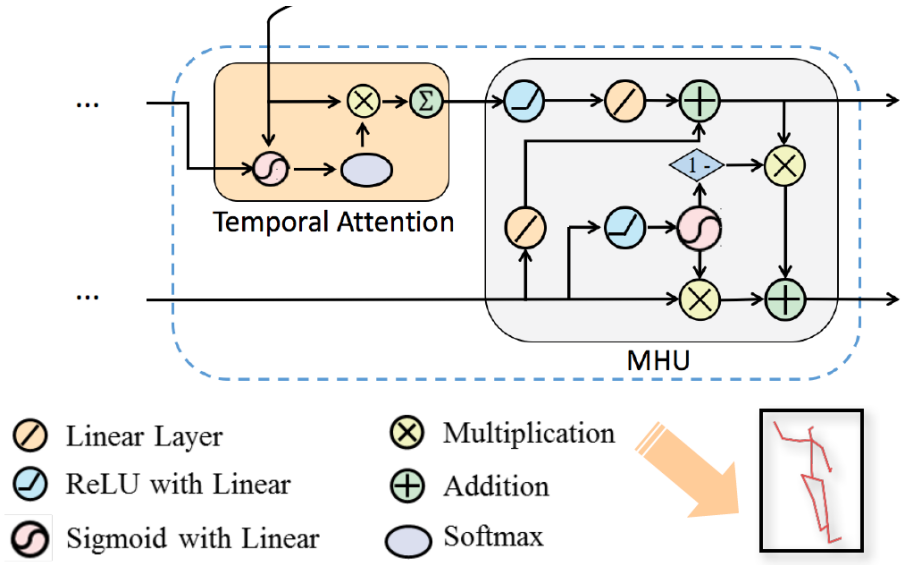


Figure 4.5: Modified High-way Unit - Unit architecture

Source: Excerpt of paper by Tang et al. [2017]

The idea is that each subsequent layer learns more and more complex features of the input. Intuitively this may manifest itself in a chain of features, where each subsequent layer is a combination of the previous features. This is a simplified example of this for animal classification:

- Input Images
- →Basic features: edges, dark/light spots
- →Geometric features: rectangles, circles
- →Textural features: fur, skin
- →Animal specific features: nose, leg
- →Full animals: cat, dog

Architecture

The CNN can also be seen as a regularized version of a conventional MLP, this is because the convolution operator given a specific kernel can also be represented by a sparse matrix operation with shared weights. So in theory any tasks that uses CNNs could be replace by conventional MLPs. So why use a CNN in the first place? The primary benefit lies indeed in this sparse representation, as it allows to get away with much less parameters that need to be trained and more easily allows for deeper layers, while maintaining acceptable training times.

There are 3 important operations that define a conventional CNN:

1. Kernel convolution: Convolves the image with the learned kernels
2. Activation Function: Usually a 'Rectified Linear Unit' (ReLU)
3. Max Pooling: Down/Sub-samples the image by (usually) a factor 2, by taking retaining only the maximum value of a square of 4.

This architecture is visualized in figure 4.6.

Relevance

CNNs are commonly used for image data, so why are they relevant for motion synthesis? At first glance, they aren't. But, CNNs have proven to perform very well on certain data that can be represented in 2D or 3D arrays, as it is able to extract spacial feature relations. This includes sequenced data that would more commonly be associated with RNNs.

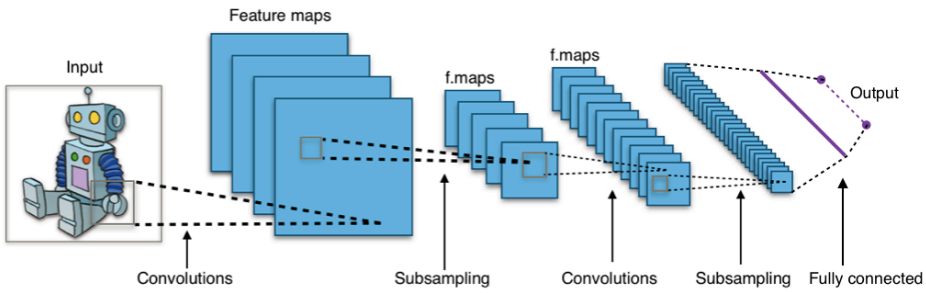


Figure 4.6: Convolutional Neural Network - Example architecture

Source: CC BY-SA 4.0 - https://commons.wikimedia.org/wiki/File:Typical_cnn.png

A great paper by Bai et al. [2018] compares the usage of generic CNNs and RNNs and shows that they are even capable of outperforming RNNs on certain sequential tasks. A similar paper by Pavllo et al. [2019b] specifically compares these two architectures for their application on human motion modeling and finds however that their CNN variant has performed worse than its RNN counterpart.

A significant amount of other researches have however utilized CNNs, or derived architectures, successfully on character motion related tasks [Habibie et al., 2017; Hernandez et al., 2019; Li, 2018; Pavllo et al., 2019a; Yang et al., 2019; Zhou et al., 2019a].

So while it does not necessarily seem like the most applicable architecture at first glance, it certainly is a valid option and could perhaps be as part of a bigger meta-architecture. One such option on how to use CNNs for this research is by transforming the raw skeletal data into a spectrogram and training on this derived data.

4.2.5. Encoder-Decoder Networks

Encoder-Decoder Networks and more specifically *Auto-encoders* have been around since the 1990s and are a type of NN architecture capable of automatically encoding complex data into a more condescend form [Hinton and Zemel, 1994].

Even though auto-encoders are a specific subset of general encoder-decoder networks, this reflection will primarily focus on auto-encoders. The bigger picture of encoder-decoder networks should however still become clear.

Rationale

An auto-encoder network is commonly used for unsupervised self-representation learning, which allows the network to extract hidden feature vectors from unknown data. This is done by funneling the data through a bottleneck, effectively forcing the network to find less complex form for representing the data.

Unlike most network architectures the result that the researcher is interested in, is not the final output of the network, but rather the hidden feature vectors in the intermediate generated latent space representation. These hidden features vectors potentially show correlations and clustering in the latent space that are otherwise not directly obvious from the raw data alone.

Architecture

The auto-encoder is a special case of the more general encoder-decoder networks, with the following (soft) requirements:

- The input is equal to the output
- A symmetrical build between encoder & decoder
- A limited bottleneck layer in between.

The simplest form of an auto-encoder would be a fully connected encoder, which reduces the amount on hidden layer neurons at each new layer until the bottleneck, with the decoder sharing the weights[5]of the encoder and symmetrical increasing the number of neurons again until it is equal to the input. This form is illustrated in figure 4.7.

[5]This is not a strict requirement and may actually hinder the learning process, but is does significantly reduce network complexity

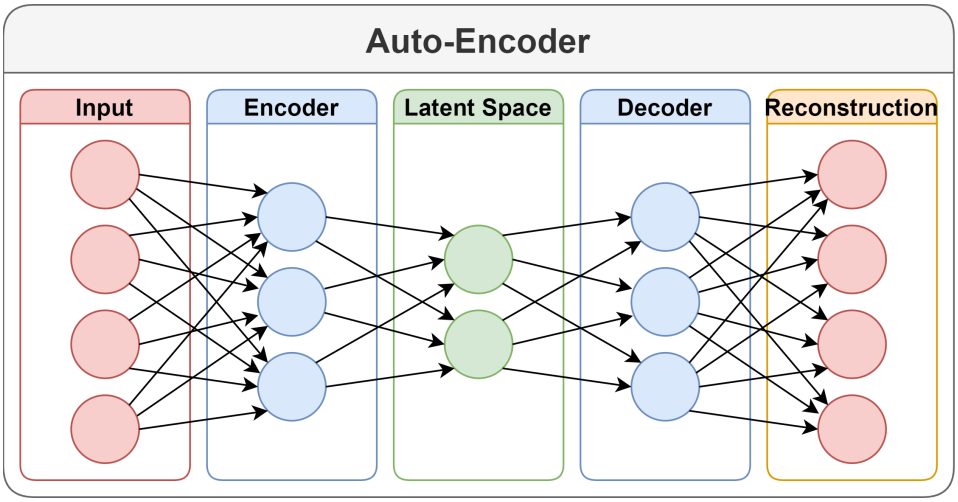


Figure 4.7: Auto-Encoder - Example architecture

The data representation at the bottleneck is commonly referred to as the 'latent space', and is intended to hold all information required to reconstruct the original data, in a compressed form.

Variations Various derivative architectures to this rather strict and limited set of requirements do exist. Some networks labeled as 'auto-encoders', such as 'denoising' auto-encoders, are strictly speaking not 'true' auto-encoders as the input is not exactly equal to the output. However strict auto-encoder only have a limited range of applications and these (slight) derivatives can excel at difficult tasks such as de-noising and in-painting [Xie et al., 2012].

More generally speaking the encoder and decoder networks can be any kind of other NN architecture, which makes the auto-encoder a type of *'meta-architecture'* that other architectures can be build into; e.g. for images it is very common to use CNNs for the en-/decoder step, or for sequences RNNs may be deployed.

Relevance

Auto-encoder are classified as *'unsupervised learning'*, meaning that there is no need for absolute labeling of the data in question. It allows for feature extraction by data alone and may very well uncover hidden relations in the data that can not be previously imagined. In combination with the mentality of GT this allows for a true data-based analysis.

There have also been several studies in which auto-encoders have been utilized for motion synthesis [Habibie et al., 2017; Holden et al., 2016; Yan et al., 2018; Zhou et al., 2019a].

Last but not least the researcher has practical experience with the usage of convolutional auto-encoders from prior work.

4.2.6. Generative Adversarial Networks

GANs are a rather modern addition to the family of NN architectures, only recently developed by Goodfellow et al. [2014].

They allow for the procedural generation of novel data, while maintaining a certain style. While primarily used with image data it has since been utilized for numerous data types.

Rationale

The main idea behind the GAN is a game, more specifically a so called minimax-game. Intuitively one can think of it as two opposing players that both try to turn the tables in their favor. Staying with the original usage of GANs for image generation, we introduce the following two players:

- The 'forger'
- The 'detective'

The forger is in charge of creating new and original artwork given a set of references, think for example of painting in the style of Picasso. His job is to fool the detective.

The detective on the other hand tries to identify whether a artwork that the forger presents him is indeed real or forged/fake, by comparing it to a known set of artworks that are in fact 'real'. His job is to expose the forger.

In the case of GANs the forger a.k.a. the *'generator'* and the detective a.k.a. the *'discriminator'* are drawing their inspiration and references from the same pile of paintings a.k.a. data, it is also possible for the generator to have no prior knowledge and base its output on random noise instead.

The benefit is that it allows for the generation of novel data without a specific quantitative measurement of the output, there are no 'good' or 'bad' results, just 'real' and 'fake'. This allows for the output of unprecedented and creative realistic results when it comes to image/data generation [Shrivastava et al., 2016].

Architecture

Just like the encoder-decoder network the GAN is a meta-architecture and allows the designer to freely choose the models that make up the generator and the discriminator.

The fundamental approach that makes a GAN function is that unlike conventional NN architectures, where the ultimate goal is the minimization of the loss-function, the GAN has two opposing forces trying to affect the same loss-function. As in our 'normal' NN the output of the generator is what we're actually interested in, not so much the final labels, as they are merely a means to an end. Due to this the generator tries to minimize the final loss function, while the discriminator acts as it antagonist and subsequently tries to maximize the final loss function.

This process is visualized on a high-level in figure 4.8.

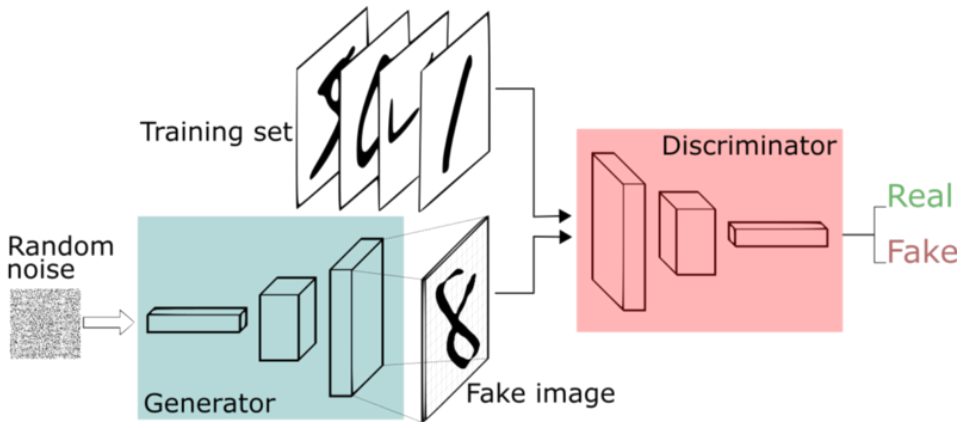


Figure 4.8: Generative Adversarial Network - Example architecture

Source: https://www.freecodecamp.org/news/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394/

Relevance

When taking the final external goal of this research into account, we can see that final goal is to deliver a stunning visual performance, by fooling the general audience into believing that the drone-swarm has a mind of its own. Thus the audience will be the final critic/discriminator of the final result of this research. Similarly we've established that in improvised dance it is hard to mathematically determine if a reactionary motion is good or bad, however we can see if it fits the motion of the partner or not. A good example to illustrate this point is to imagine an expert salsa dancer and an expert ballet dancer and tell them to improvise together, the style and output of each dancer individually can have good or bad form, but it is not the defining feature that allows for a great interaction. If each dancer would just do what they know best, the result is probably a disaster when it comes to teamwork.

In this spirit GANs offer a unique opportunity for creating novel and prior unseen motions that are natural and focused on using the data in a holistic manner.

In practice there have also been very recent studies successfully applying GANs for motion synthesis [Gui et al., 2018; Hernandez et al., 2019; Kender and Way, 2018; Wang et al., 2020a].

4.2.7. Phase-Functioned Neural Networks

The phase-functioned neural network (PFNN) a very novel NN architecture recently proposed by Holden et al. [2017a]. At its core its a novel approach for training NNs on data that has a singular distinct cycle, primarily for human locomotion. Although it could in theory be applied to any dataset that showcases a clear cyclical nature, e.g. weather dataset to account for seasonality or oceanic data to account for the tides.

Rationale

This architecture was created to avoid what is commonly known as 'floating artifacts' in character animation. This happens when contact points with the surface are not preserved and it appears that the character is floating like an ice-skater or a ghost. This is usually a result of a regression to the mean, when temporal continuity and momentum is not preserved. One can think about is as trying to predict the position of a pendulum, without knowing its phase, on average guessing the middle every time will be correct.

To avoid this, the raw data is labeled with the current phase of the primary cycle, prior to training. The main idea is that multiple networks are being trained simultaneously, where each network has a specific phase in the cycle attached to it. Effectively splitting the dataset into smaller chunks, based on the phase and training the network piecewise.

By assigning a fixed phase to each data point and as such effectively training a separate network for each phase, this temporal information is preserved. It can be seen as blending all data from the same phase, rather than blending data from all phases. As a result, the resultant output is cyclically aware and knows which motions are to follow next.

Architecture

Theoretically the rationale of the PFNN can be applied to any NN architecture, classifying it as a meta-architecture or rather as an add-on to an existing network.

When training, the weights of the network are based on this primary phase parameter, by means of interpolation between N discrete states. Holden et al. [2017a] used a generic MLP with 4 discrete phase states α_0-α_3, representing phase values of $0, \frac{\pi}{2}, \pi$ & $\frac{3\pi}{2}$ respectively. For any intermediate phase they blend the discrete phases together to arrive at a unique set of weights that governs the internal structure of the network for the given input.

This structure is illustrated in figure 4.9.

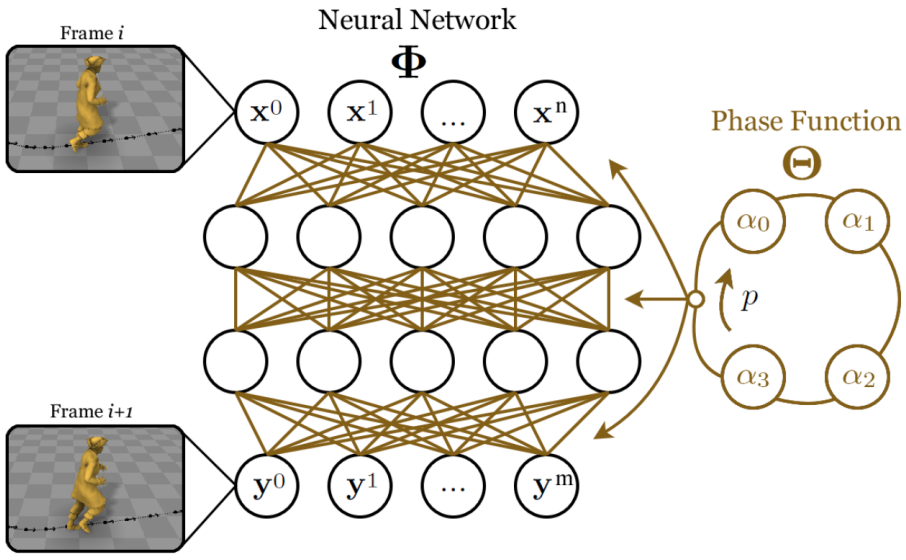


Figure 4.9: Phase-Functioned Neural Network - Network architecture

Source: Excerpt of paper by Holden et al. [2017a]

Relevance

Generalized dance, unlike locomotion, unfortunately does not have a singular determining cycle, from which a phase could be determined. So while the PFNN in its current state cannot effectively be applied to the dataset at hand, there might be an option for creatively applying the underlying concept in a more generalizable manner.

Disclaimer: The following idea is heavily speculative and still needs to be proven!

The proposed hypothesis is as follows: By transforming the raw data into the frequency domain, by means of a fourier transform, this allows for the exact definition of a phase vector over all frequencies. So instead of selecting a singular frequency at which the phase is to be determined the approach will be expanded for all frequencies alike. Effectively training the network on a phase augmented spectrogram of the data, hereby a separate set of weights is given for a discrete set of phases for every frequency to be considered.

The benefit would be that the network can more easily train each network on only the data that's relevant to it, further reducing the potential forming of floating artifacts, and to preserve temporal continuity over all frequencies. It also allows general control laws to be applied to the input and/or output data; e.g. it would allow to apply a low-pass filter to reduce the overall speed and momentum in the data, in case the inertia of the drone hardware can not keep up with certain fast-paced output.

The downside of this approach would be that it most definitely increases the network complexity, through the necessity of computing and processing the entire frequency spectrum and its phases at each time-step.

Ultimately this appears to be a very interesting and promising topic to explore, though the added complexity might put strain on the hardware runtime, as well as the time-line of the project.

4.3. Data Pre-processing

Now that a solid understanding of the various NN architectures and their relevance to the research has been present, focus is shifted on how to 'conventionally' process the data prior to feeding it into the NN.

Various data pre-processing steps can be considered to allow the neural network to more easily interpret the data at hand and to avoid ambiguities and spending network complexity extracting 'trivial' features that can easily be computed prior.

These include, but are not limited to:

- Derivation - Filtering of 'velocity' and 'acceleration' out of the raw data frames
- Transformation - Changing the mathematical domain in which each frame is represented
- Data augmentation - Creating new/additional data from the existing dataset

Here only the methodologies for the generation of new data from, or new representations of, existing data will be addressed. Data cleaning and sanity checks will be addressed in the more practice oriented part of the report, in chapter 5.

4.3.1. Derivation

Derivation of the data is the first important step, as at every frame only the current position and rotational states are preserved and no actual 'motion' data in the form of any information holding any momentum is preserved.

Feeding the neural network not only a static pose, but also its derivatives e.g. velocity & acceleration, should allow the system to better understand the underlying 'motion'. Failing to do so could be, intuitively thought of as, the equivalent of studying dance by means of analyzing a museum full of statues; It is not impossible, but a a lot more tedious than e.g. watching a video in motion.

An argument can be made that e.g. an RNN could extract this information due to its recurrent nature and this is probably true, but why spend valuable training time and network complexity on learning something we can present the network with prior. The downside will be that the initial feature vector will have to be copied for every derivative extracted, as now not only position and rotations, but also their 1st and perhaps 2nd derivative vectors, need to be added. This increases network complexity as well, but may most probably be well worth it.

Several researchers have shown to add 1st order derivatives to their parameterization for motion data handling [Auslander et al., 1995; Holden et al., 2017a; Martinez et al., 2017; Wang et al., 2020b].

An important final note is to not ignore the fact that the final product needs to run on a live-stream of data. This means that only 'past' data points may be considered when computing the derivatives, so the application

of central/forward differencing are out of the question. So backward differencing, and potentially pre-filtering by means of an exponential moving average or Kalman filter will be needed.

4.3.2. Transformation

Changing the data basis with which the data is represented, may allow the NN to better process the data, it furthermore results in extracting alternative feature vectors that might allow for better human insight into the data and perhaps practical dimensionality reduction. Various approaches are considered to aid in the data processing:

- Using alternative representations for rotational data
- Extraction of alternative feature base, using PCA
- Transformation into alternative mathematical domains

Rotation Representation

For a computer the values '0' & '360' & '720' are different, although we know that when applied to angles in degrees these are in fact the same underlying angle which is represented. Conventional Euler-angles, which are prone to this ambiguity and issues such as 'gimbal-lock', are often replaced by *quaternions*. Quaternions are an alternative form for expressing rotation, which avoids cyclical ambiguity. Though conceptually harder to grasp, computers excel at utilizing this 4D number system for representing and computing orientations [Huynh, 2009].

Various researchers already deploy quaternions in their motion models [Aristidou et al., 2018b; Pavllo and Auli, 2018; Pavllo et al., 2019b; Villegas and Lee, 2018].

Further extensive research has been done by Zhou et al. [2019b] on the effect that different rotational representations have on the performance of NNs for motion synthesis. They found

> "that 3D and 4D representations are not ideal for network regression"

and propose novel continuous 5D & 6D representations for rotations, which appear to be particularly suitable for neural networks to learn.

Applying the findings of this paper will hopefully prove useful to the overall performance of final model.

Principle Component Analysis

Principle Component Analysis (PCA) allows for fast identification of the most relevant feature space, by finding primary components within data e.g. for faces a 'smile' vector, or for dancing a 'jump' vector might be a possible result. Based on the data available this can have two beneficial effects:

- Removing feature vectors with the lowest relevance can drastically reduce data dimensionality while minimizing reconstruction error
- Primary features may likely be easily 'human interpretable', allowing for deeper understanding of the data at hand

Initial tests involving PCA have already been conducted and appear promising. Some extracted 'human interpretable' features include 'jump','kick' & 'arm swings'. They clearly show cross-correlation between various body parts for certain motions.

Change of Domain

Initially inspired by the concept of the PFNN, a transformation of the time domain data to the frequency, or even the Laplace domain might be beneficial for 3 possible reasons:

- Improved quantitative human insight into which motions are slow/fast or simple/complex
- Possible applications of conventional filtering/control laws on the model output to custom tailer the final output
- Possible application of the modified PFNN, as proposed in section 4.2.7

Fourier Transform allows for a frequency based visualization of the data and might uncover certain 'peak frequencies' that dominate the data. Furthermore a clear distinction between 'hasty' and 'paced' sections of the dance, could be identified by comparing the average dominant regions of their primary power in the frequency domain.

A slightly unrelated but yet very interesting paper by Pratt et al. [2017] explore the possibilities of Fourier transformations on processing of image data in CNN, must like the 'JPEG' format utilizes a type of Fourier transform for image compression. The bottom line is that various creative and unexpected positive results have come from this simple yet powerful approach.

Laplace/Z Transform , seen as a transient expansion of the Fourier transform, enables the researcher to not only gain the benefits as described above, but additionally identify if a movement is fading-in or fading out, by assessing the magnitude of the real-components of the Laplace transformed dataset.

This addition has not been found in existing literature for its use on motion data and is highly speculative.

4.3.3. Data Augmentation

Data augmentation is a process of creating new data by altering an existing dataset within given limits. There are usually 2 reasons why data augmentation is used:

- The existing dataset is too small for training and needs to be extended, when additional data acquisition is not an option.
- The resulting model is supposed to be invariant to certain features and not learn from false cues.

A famous example of the second point is the paper by Ribeiro et al. [2016], which illustrates that for a classification task between 'husky' and 'wolf' the incorrect cue of the background, 'snow' or 'grass' was used instead of the animal itself. This could have been avoided by augmenting the dataset by random or removed backgrounds, which essentially forces the algorithm to learn on the animals features instead.

This also illustrates some of the dangers that come attached to utilizing NN as a black-box approach and highlight the necessity for *prevention* and improved insight for *validation*.

Data augmentation can aid in the prevention of learning of false cues if considered before training.

In this specific case it is crucial to remove issues with data that is represented with different numerical representations, yet describe the same physical phenomenon. The two most prominent issues that need to be solved are:

- Positional ambiguity
- Rotational ambiguity

Disclaimer: It is important to note that changes to the reference frames are only to be applied once over an entire take and **not** on a frame-by-frame basis, as this would destroy temporal continuity.

Positional Ambiguity

While the relative position of the two dancers to each other is crucial to learning the interaction between them, where exactly this interaction is taking place is irrelevant. For example, if the origin of the reference frame chosen is right under the dancers feet, or 1 km further should not change the interaction between the two dancers. This does entail that changes in reference frame are irrelevant to the dancing motion. This is true at least when moving the coordinate system in the XY (floor) plane and not so much when we start rotating the reference frame, as the gravitational 'upwards' vector will be lost.

Rotational Ambiguity

While maintaining the up/Z vector constant there are 3 DOF that can be freely augmented, just like the X and Y axis are free to change, the heading around the Z axis is as well.

Therefore the combined yaw angle of both dancers can be altered freely as well.

4.4. Hyperparameter Optimization

A crucial element why NN design still involves a significant amount of trial-and-error is the choice that a network designer has when it comes to the so called *hyperparameters* (HP).

In the context of ML/DL hyperparameters represent any parameter of a model that is fixed and is not dynamically learned in the training phase. While not strictly quantifiable the choice of optimization algorithm,

activation function(s) and/or the application of regularization could also be considered as hyperparameters, as these design choices heavily influence the overall performance of a model.

Taking a simple MLP as an example we could identify the following hyperparameters:

- Network structure
  - Number of hidden layers
  - Number of neurons per layer
- Learning parameters
  - Learning rate
  - Batch size
  - Weight decay (Regularization)

The example above is indeed just that, an example; the amount and impact of various hyperparameters can greatly vary based on the network design chosen and influence the networks performance.

It is very important to identify all hyperparameters that affect the system and be aware of their influence.

An attempt to move away from the trial-and-error approach of choosing various hyperparameters is to automate the procedure in a form of secondary optimization loop. This procedure is illustrated in figure 4.10.

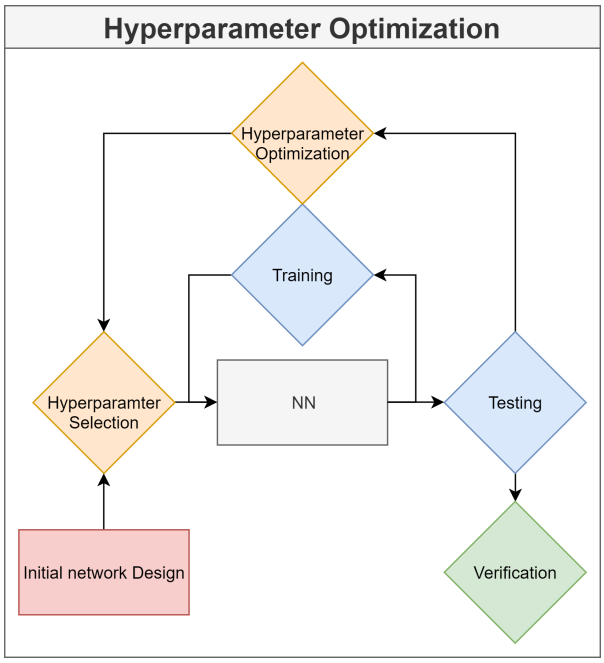


Figure 4.10: Hyperparameter optimization flowchart

So to avoid the rather 'unscientific' guessing work that is involved in picking the exact layout of a neural network by hand, the following two common hyperparameter optimization (HPO) algorithms will be briefly explained below:

- Grid search
- Bayesian optimization (BO)

More optimization techniques do exist, however grid search is the simplest and used to illustrate the technique, while Bayesian optimization appears to be the most promising. Also the researcher already has prior experience with the application of Bayesian hyperparameter optimization (BHPO) for DL.

Verification/Validation dataset: One important thing to note, when applying this approach, is that the inner training loop uses the network performance on the *'training dataset'* to optimize, while the outer hyperparameter optimization loop uses the network performance on the *'testing dataset'* to optimize the chosen

parameters. As this approach infringes the independent nature of the *'testing dataset'* it is required to introduce another truly independent dataset, commonly known as the *'validation dataset'*. Effectively splitting the dataset into 3 separate parts. If this step is skipped, over-fitting on the *'testing dataset'* will be the result.

Disclaimer: For the remainder of the research the *'validation dataset'* will be referred to as the *'verification dataset'*, as it only allows for verification of the model in a closed lab environment. The true *validation* will occur when the model is tested in the real world setting with a live dancer.

4.4.1. Grid Search

The 'grid search' approach simply tries a predetermined set of values for each of the hyperparameters. In the hypothetical case of a 2 hidden-layer MLP, 2 hyperparameters for the amount of neurons in each layer can be introduced. If each layer is to be evaluated at $1, 2, 3$ neurons, this effectively creates a 2D search space with $(3 \cdot 3 =)$ 9 possible combinations, for the set of hyperparameters. After all potential sets of hyperparameters have been evaluated, only the best performing set is kept.

This approach however heavily suffers from the 'curse of dimensionality' and quickly becomes impractical when applied on systems with higher dimensional hyperparameter sets.

4.4.2. Bayesian Optimization

A methodology known as *Bayesian Hyper-parameter Optimization* (BHPO) [Eggensperger et al., 2013] will be utilized. This method is by no means new and has been around and applied on NNs for at least 25 years [Neal, 1995].

BHPO evaluates the network's optimal performance for various sets of hyperparameters, based on a probabilistic utility function it finds the next optimal set of hyperparameters that has the highest probability of potentially being better than the prior one. This might seem a bit confusing at first but effectively is a very powerful method that unlike 'gradient descent' does not keep moving to a potentially local optimum, but much rather views the entire optimization space as a holistic set where the 'unknown' may hold value. In this method a trade-off between *'exploration & exploitation'* [Bondu et al., 2010]:

Exploitation refers to a bias towards trying a new layout close to the current optimum, to make it even better.

Exploration can also be seen as 'curiosity' and has a bias towards trying a layout that is as far away from anything known prior as possible, to potentially find a the real global max-/minimum, rather than a local max-/minimum.

4.5. Human Perception Experiment

The final method to be discussed in this chapter is also the basis for the final experiment after completion of the development phase.

In the spirit of the idea behind the discriminator in a GAN, it is proposed to setup a final experiment that involved real human experts to assess the quality of the final model, in a binary classification task.

As previously mentioned what is mathematically 'better' is not always considered to be of better quality when it comes to creative generative tasks, this is why this approach can be seen as an independent validation of the final result. This predicament has also been observed by another team of researchers that have been working with GANs:

> "a smaller loss does not always indicate a better quality."
>
> [Kender and Way, 2018]

The proposed setup for this experiment is as follows:

1. Preparation
   (a) Finalize the NN model to be evaluated
   (b) Create dataset of generated motions to original input
   (c) Finalize the visualization on screen
   (d) Develop virtual testing environment for experiment

2. Run experiment
   (a) Select participant
   (b) Show a take of 10-15 seconds
   either from the *recorded dataset* or the *generate dataset*
   (c) Present the user with a binary choice-box on screen, to select if he/she thinks if the take was *real* or *fake*[6]
   (d) Repeat to the next take (return to 2.2)
   (e) Repeat with the next participant (return to 2.1)
3. Evaluate the experiment
   (a) Parse the resulting binary classification datasets for all participants
   (b) Compare the distribution of choices for each participant
   (c) Compare the distribution of choices per take
   (d) Check distributions for statistical significance
   (e) Draw conclusions

This method could also be called the *'turing test'* for motion synthesis.

[6]For humans perhaps a more granular scale from 0-10 might be more appropriate, in contrast to the pure binary choice in a GAN

III

Project Plan

5

Development Plan

The practical framework for this design research consist of 4 distinct phases:

1. Data Handling
2. Model Development
3. Live Interaction
4. Validation

Building on the background knowledge acquire in the previous chapter on methodology (4), this chapter outlines each phase of the practical process and their respective tasks.

5.1. Data Handling

The very first step in practical deep learning research is acquiring data. Without data the best designed network wouldn't have much use. For this research the data acquisition is lead by and obtained in collaboration with AkoB. The exact amount of data required is hard to determine prior to testing the models, although as a rule of thumb

> "In practice, results with large data sets are often quite good [...]"

[Marcus, 2018]. Without having a solid benchmark for how much data was required the initial goal had been set to acquire 100h of motion capture footage. After the project's budget had to be reduced, it was decided that 10h was the new goal to strive for. The data-acquisition process has since then been completed and the goal of 10h was almost reached. After comparing the acquired dataset with other publicly available MoCap dataset we can safely assume that indeed more than enough data has been acquired.

This section will outline how the data-acquisition has taken place, what the final dataset looks like and what the desired pre-processing steps are to be taken.

5.1.1. Data Acquisition

The Setup

The setup for a recording session is as follows: Two dancers and the choreographer of AkoB are located within their motion capture studio. One dancer is considered the *'Input dancer'*, while the other is considered the *'Output dancer'*. The 'Input dancer' takes the lead in the improvisation to follow, while the 'Output dancer' reacts to their partner accordingly. In the actual showpiece the 'Input dancer' will remain human, while the 'Output dancer' will be replaced by the 'drone swarm dancer'.

AkoB utilizes a motion capture system called *'OptiTrack'*, with 16 'Prime 13' cameras in place [1]. For human motion tracking Velcro suits with 54 passive IR markers (reflective balls) are used. The company and dancers are well experienced with the system, and have used the same system and techniques for various other projects in the past.

[1] A system very similar to the one being used at our faculties 'CyberZoo'

File Formats

Before recording started, research was done in the applicability of the various motion capture file formats that are available. The research has been limited to the formats that are available by OptiTracks export functionality. The results of this research are summarized in table 5.1.

Advantages	Disadvantages
'.TAK' (Binary) - OptiTrack proprietary data format for *'Motive'*	
• Raw unaltered data format that can be formatted in any other format required.	• Unable to be parsed or converted without a valid license key, of which only 1 is available
'.BVH' (ASCII) - Biovision Hierarchical Data	
• Easy to parse, write & handle • Motion data is a simple list of floats, which is perfect for handling by Neural network • Includes skeletal data	• Fixed time step recording only (However this is actually preferred for RNNs) • Only contains data on 1 skeleton and its motion
'.FBX' (Binary/ASCII) - Autodesk Filmbox	
• Stores 3D-Model data within the same file • Allows for storage of complex additional information • Commonly used format	• Complex data format - Hard to comprehend or parse • A lot of unwanted additional data
'.CSV' (ASCII) - Comma Separated Values	
• Simple to parse • Universal file format	• Oversimplified format • No skeletal data • BVH is basically an improved CSV file
'.TRC' (ASCII) - Format for *'Motion Analysis'*	
• Easy to parse & comprehend	• Not skeleton-based • Contains only positional information
'.C3D' (Binary) - Format by Bethesda National Institutes of Health	
• In the public domain • Ability to store extensive additional experiment information	• Binary data, not easily parseable and writable • Non skeleton based

Table 5.1: Overview of available motion data file formats

Ultimately the '.BVH' format was chosen for the following reasons:

- Ease of use - quick parsing & easy writing
- Clear separation between skeletal and motion data
- Straightforward rotation based numerical data-structure - Can directly be used for NNs

Guidelines

The following guidelines/best-practices were agreed upon and used in the data-acquisition procedure:

- No pre-processing: Deliver the data as raw as possible
  same output that will be available live from the system during the show
- T-Pose before and after every take.
  This allows for easy post processing, e.g. cutting out pre-/post-take data.
- Define reusable labels to classify takes
- Anonymity of participants has been ensured by using Symbols for each dancer A,B
- Consistent file naming schema:
  `"Take YYYY-MM-DD HH.MM.SS_Role{L,F}DancerSymbol{A,B}_Label#1,Label#2[,Label#N]_Description_Take#.EXT"`

  e.g.
  `"Take 2020-04-30 11.37.00_LB_ball,slow_pass energy in ball _01.BVH"`
- Recording frequency: 100Hz
- Storage of raw '.TAK' format as backup, as well as export to '.BVH' format

5.1.2. Final Dataset

The final dataset that has been recorded by AkoB contains a grand total of 9.2h of data split over 147 takes, recorded over the course of 7 recording days. As a supplement to the dataset AkoB has recorded all noteworthy events, such as 'loss of tracking', in a recording sheet.

The following section will present an overview and the preliminary analysis of the dataset, as well as comparing it to other noteworthy MoCap datasets in the field.

Our Dataset

The most important key metrics on our 'dance-interaction' dataset are given in figure 5.1..

AkoB Mocap Dataset – Summary					
Nr. of takes:	147	[#]	#Frames:	6,647,486	[#]
Duration:	554	[min]	→	9.2	[h]
Size:	12582.6	[MB]	→	12.3	[GB]
True DataRate:	22.7	[MB/min]	Theoretical DataRate:	7.1	[MB/min]
Avg. Yield:	79.1	[min/day]	→	16.5	%
Nr. of Labels:	361	[#]	Avg. #Labels:	2.5	[Labels/Take]

Figure 5.1: AkoB Dataset - Key Metrics

Figure 5.2 shows the distribution of the duration for each take in the dataset, showing that the fast majority of takes is between 3-5 minutes in length.

Figure 5.3 showcases the distribution of all labels that were used to classify the data in the dataset. The labels were created in a dynamic fashion by the choreographer, meaning that he made them up on-the-fly and continued to use those techniques and labels that achieved the desired visual results. The majority of labels used have been used in at least 10 different takes, while not the largest sample size it should allow to see if there are indeed similarities between the manual labels and the results of automated clustering procedures such as the K-Means algorithm.

Other Datasets

During the literature review several MoCap databases have been found that are used for motion handling benchmarks.

The by far most extensive dataset found is the *AMASS* dataset. This dataset was put together by Mahmood et al. [2019] it

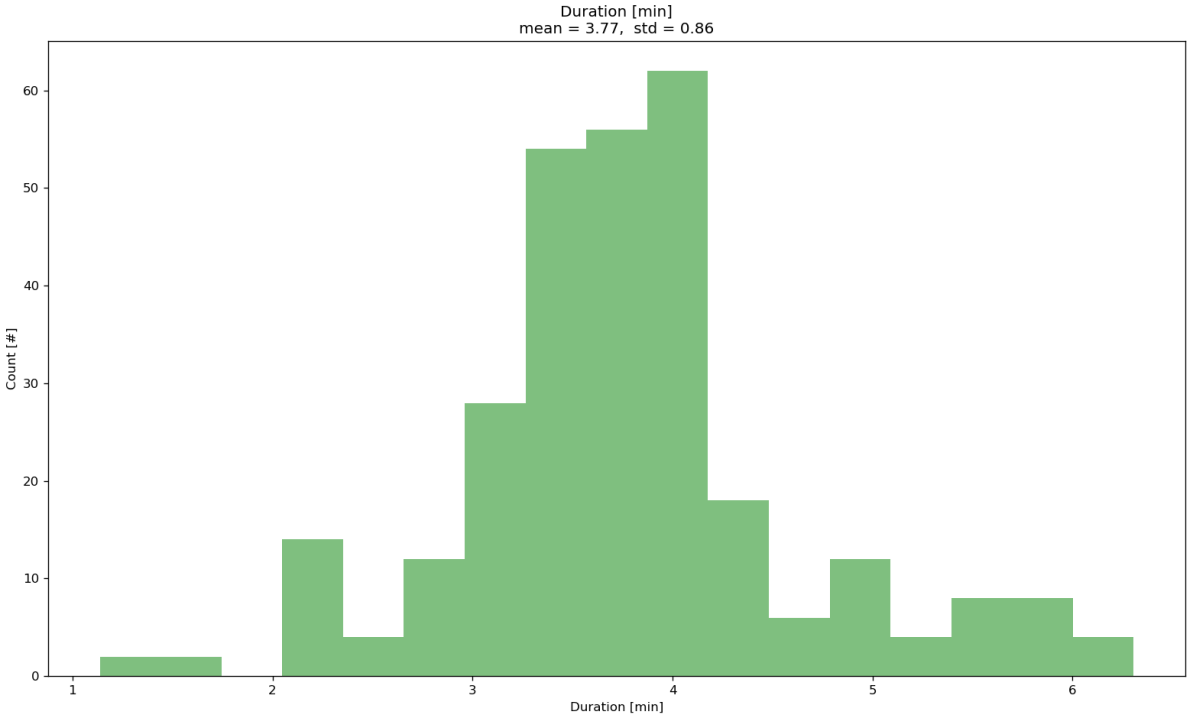


Figure 5.2: AkoB Dataset - Distribution of durations

> ”unifies 15 different optical marker-based MoCap datasets by representing them within a common framework”

All datasets that have been combine in the AMASS dataset have been compared to our dataset in table 5.2 and sorted by their total duration.

This clearly shows that our dataset is in the top 3 of the most common MoCap datasets, which instills great confidence in the fact that indeed more than enough data has been acquired. Usually most takes in these MoCap dataset are intended for classification and are thus much shorter, therefore with respect to the amount of takes/motions in the dataset our dataset falls short, but is in comparison still in the solid middle.

5.1.3. Data Pre-processing

After collection of the raw footage a crucial pre-processing step, prior to applying any deep learning techniques, is to clean and sort the data. Additionally the transformations and augmentations, as outlined in chapter 4.3 are to be applied at this stage.

To avoid issues down the line, and to reduce the manual workload on the researcher, the steps mentioned below are intended to be automated by a pre-processing script written in python.

Data Cleaning

Ensuring that the dataset is clean and correct is important as to not result in a ’garbage in, garbage out’ scenario.

The following tasks are to be performed:

- Check and fix all file(name)s
  - Check for typos in labels [DONE]
  - Check for ambiguous labels and merge then[2] [DONE]
  - Remove excess spaces
- Check the recording sheet and remove any takes in which things went significantly wrong, e.g. loss of tracking, system freeze, etc.

[2]After checking in with the choreographer.

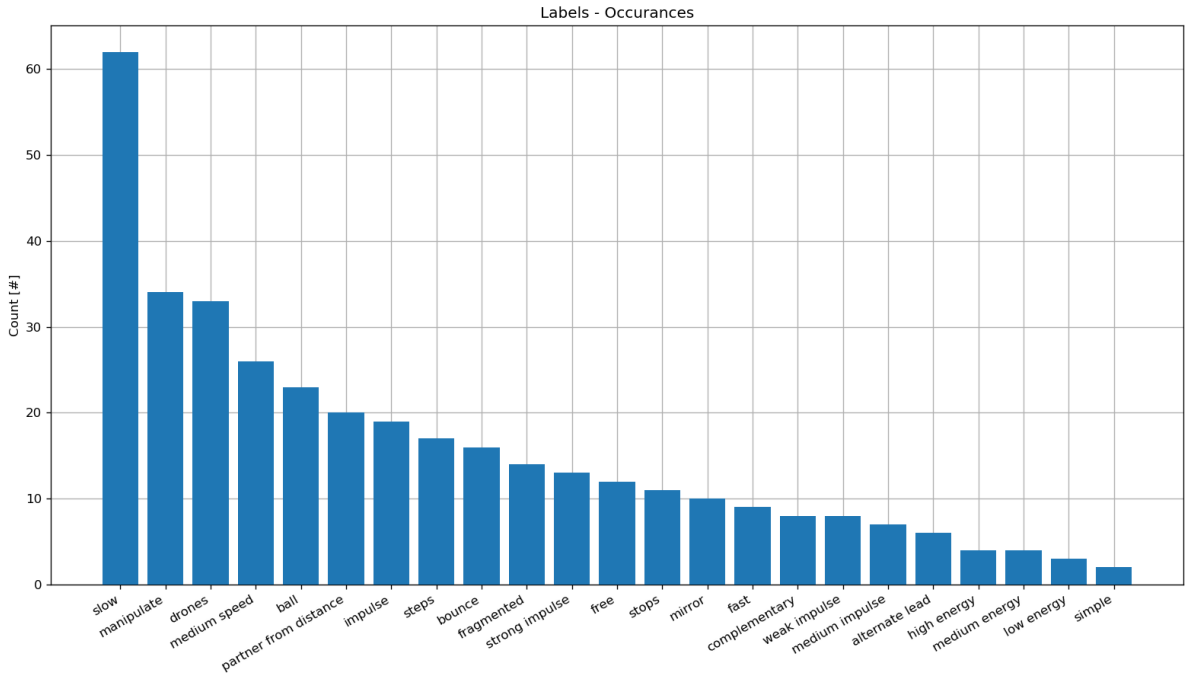


Figure 5.3: AkoB Dataset - Distribution of labels

- Removing any excess material, before or after a take (e.g. the dancer standing still and listening to the instructions of the choreographer), hence the T-Poses.

Any steps that alter the raw original data have been and will be documented strictly, as to ensure full transparency and avoid any charges of data-meddling, cherry-picking or fraud.

Data Correction

Mocap recordings aren't perfect and it can happen regularly that tracking is lost, or that the skeleton is reconstructed incorrectly. Cleaning these errors by hand is a very tedious process, but there are attempts made by researchers (e.g. [Holden, 2018]) to create automated solutions.

It is true that these imperfections will persist when it comes to the live interaction, so actually removing all these problems would result in a dataset that is not representative of the final application. However this is only true with respect to the training data for the 'input-dancer', for the 'output-dancer' we certainly don't want the system to learn that it needs to 'spasm' every once in a while, as we do not want drones flying uncontrollably all over the place. This is also the reason why continuity in the output derivatives needs to be enforced and incorporated into the model's loss-function.

Therefore robust cleaning of the data prior to training is important in particular the following elements are important:

- Removing any excess material, before or after a take (e.g. the dancer standing still and listening to the instructions of the choreographer), hence the T-Poses.
- Check the data for 'impossible' motions:
  - Incorrect reconstruction: skeletal positions outside of human limits (e.g. Twisted limbs, intersecting limbs)
  - (Temporary) loss of tracking: 'Jumps' in data, resulting in accelerations outside of human limits
- Fix any 'missing/corrupted frames' by means of interpolation

Data Augmentation

As a final step the data is augmented by means of the techniques highlighted in the section 4.3.3 and split into 3 parts, the training, testing & verification datasets.

Dataset	#Markers	#Subjects	#Motions	Minutes
AMASS	-	450	13295	2608.29
KIT	100	55	4232	661.84
AkoB	54	4 (2)	147	553.96
CMU	42	106	2083	551.56
BMLrub	41	111	3061	522.69
Eyes Japan	37	12	750	363.64
BMLmovi	67	86	1801	168.99
MPI HDM05	41	4	215	144.54
Total Capture	53	5	37	41.1
EKUT	51	4	349	30.71
ACCAD	82	20	252	26.74
PosePrior	53	3	35	20.82
MPI MoSh	89	19	77	16.53
SFU	53	7	44	15.23
Transitions	49	1	110	15.1
DFaust Synthetic	67	10	129	10.37
Human Eva	39	3	28	8.48
TCD Hands	85	1	62	8.05
SSM	86	3	30	1.87

Table 5.2: Comparison of common MoCap datasets

5.2. Model Development

After the data has been acquired the actual development phase can start.

The following outlines the steps intended to be taken during development of the model.

5.2.1. Framework & Hardware

Development will most likely take place within a NVIDIA docker container, code written in python 3, utilizing the deep learning framework 'pytorch' [3].

With regard to the computational facilities, the server at the researcher's home lab will serve as the primary development hardware, while an additional high-end computer at AkoB can serve as a secondary setup running over-night training and performing BHPO, while the researcher improves the overall algorithm, or tries out alternative models.

5.2.2. Model Design

Given the GT mentality no final framework is set in stone at this point in the research, however the effort has been made to draw up a data-flow diagram illustrating the entire processing pipeline given the current best estimate on the models to be used.

The high-level summary version of this data-flow in presented in figure 5.4, while the full detailed version can be found in appendix B.

The proposed meta-architecture consists primarily of 3 NNs, their architectures and intended use are:

1. **Auto-Encoder*:** Preliminary step for learning hidden representations in the data
2. **Generator*:** A recurrent generator will predict the next pose and velocity at the next time-step
3. **Discriminator:** A simple MLP classifier will, combined with the generator, form a GAN

** can be trained independently*

5.2.3. Training & Optimization

Based on the methods as outline in chapter 4, the following steps are taken for each model:

1. Develop/adapt suitable architecture to handle multi-dimensional temporal data.

[3] 'keras' might also be a possibility

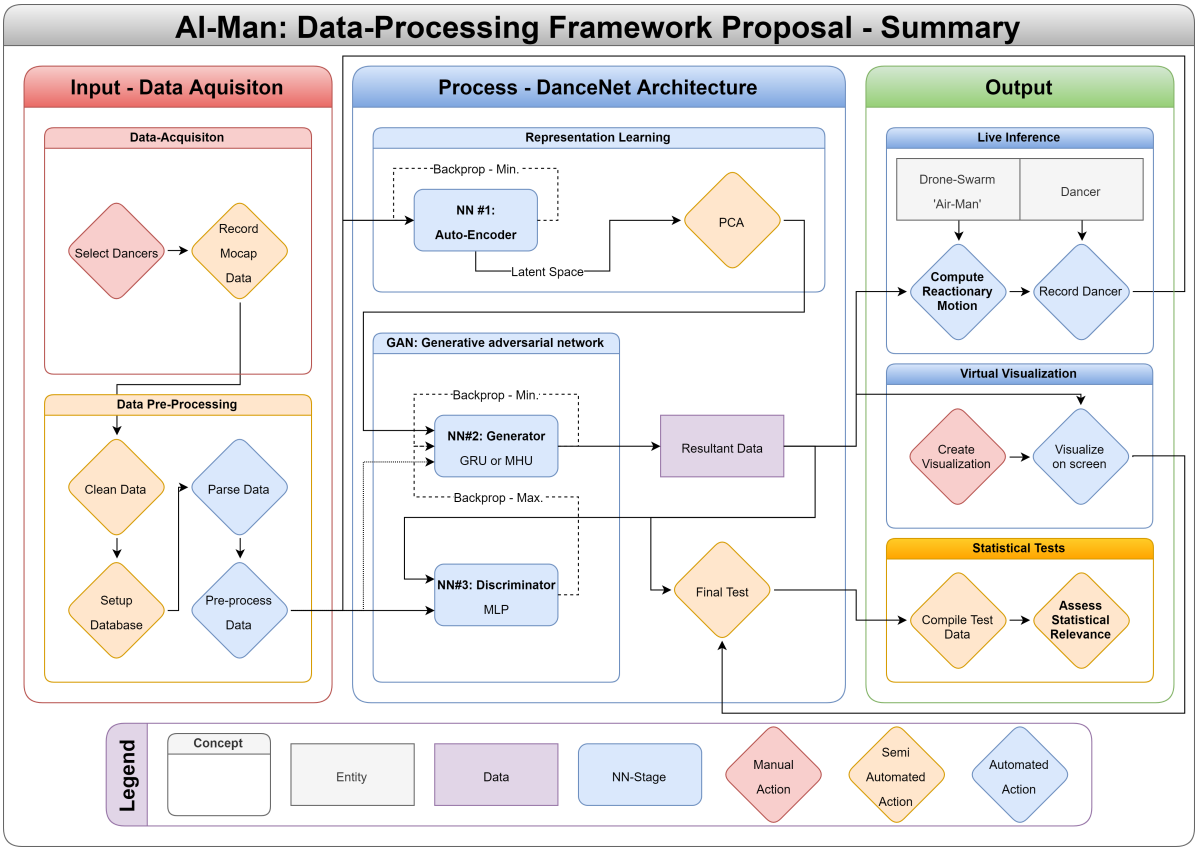


Figure 5.4: Data-processing Framework Proposal

2. Transform the raw data into suitable input for the selected architecture.
3. Implement an adaptive early-break condition to avoid over -fitting, yet allow optimal training results*
4. Train and Test the algorithm with the data provided
5. Perform 'Bayesian Hyper-parameter Optimization' and re-iterate step 4
6. Verify the results with the verification dataset

Early break condtion:* Conventionally NNs are trained for a certain number of *'epochs'*, each epoch indicating that the network has processed every bit of the dataset once. However this approach can lead to over-fitting if the data is being reused over and over.

Generally speaking the performance on the training dataset will keep on increasing the longer the network is trained. Initially this is also true for the performance of the testing dataset, but there is a law of diminishing returns and at some point even a reversal in testing dataset performance. A dynamic early break condition will be used that monitors this divergence of the testing dataset performance, as a cue to stop training the network.

5.2.4. Model Comparison

Strictly speaking only NN #2 would be required to result at a usable output that could satisfy the external goal of this research. However, the internal goal is arrive at a quantitative comparison between various approaches to benchmark these and find the optimum among them.

Therefore, the intention is to train various architectures with certain choices in place or not, possible modules to alter/activate/deactivate are:

- Addition of derivatives
- Preprocessing with NN #1
- Preprocessing with PCA
- Post training with NN #3

- Training networks simultaneously vs in sequence
- Different types of recurrent units
- ...other options as time allows

To ensure that we're not comparing apples with oranges, for each model the intention is to run BHPO on each configuration to allow for the comparison of the 'optimal' layout for each configuration.

After having trained the individual models, a comparison on the various architecture performances and a subsequent selection of the best models is to take place. The final model will be selected an prepared for live inference.

5.3. Live Interaction

There are commonly 2 stages to applying deep learning in practice:

1. Training/optimization (Preparation):
    - The 'learning' process, converting data into knowledge
    - Slow: runs on professional hardware
    - Requires large amounts of data
2. Inference (Application):
    - The 'deployment' process, utilizing the network to achieve a goal
    - Fast: can run on 'consumer' hardware
    - Requires (live) samples to be processed

For the model to be used in the final showpiece, there are 3 main tasks that need to performed:

1. Modifying the network to be usable with 'live' data
2. The development of a virtual simulation environment in which the model output can be observed
3. Creating a safety design that checks the output of the NN, before sending the data to the drone hardware

5.3.1. 'Online' Modifications

After finalization of the NN model, it is taken 'out of the lab and onto the stage'.

The NN is adapted to handle live data input from the OptiTracks UDP broadcast and profiled respectively such that its runtime performance matches, at least, the update rate of the drone hardware. Due to some technical issues with the communication protocol to the drones this update rate is currently as low as 10Hz, although the hardware developers strive for a update rate of 100Hz. This is equal to the frame-rate of the recorded dataset and can be considered the desired update rate for the model in this research.

5.3.2. Simulation

To visualize the results generated by the NN to be assessed by the researcher and to allow for live interaction with the dancer, a simulation environment needs to be developed. This live-visualization environment will be developed within the 'Unreal Engine 4', which allows for visualization of a skeletal rig, as it is applied to and transforming a character model.

While this task is mentioned rather late in this report it might be something that will be developed in parallel to the NN development, as it allows for a better insight into the networks operations during development.

As a final touch the desired view medium for this visualization will be a VR environment, more specifically AkoBs HTC vive setup. This will have the added benefit that the dancer has a visualization of the virtual dancer in a setting that is as natural as possible.

5.3.3. Safety Design

Last but certainly not least the 'Additional Challenges', as mentioned in chapter 1, will need to be addressed.

Even though the goal is to give the drone-dancer as much 'free will' as possible, we'll ultimately still be dealing with real drones and real people, which should **NOT** cross paths under any circumstance.

Therefore a system of final sanity checks needs to be put in place that analyzes the output of the generator network and ensures non of the limits, as described in section 1.2.2 will be violated.

This effectively results in a hybrid approach to the final model, as we can not with 100% certainty tell how the resulting NN will react to truly novel data.

Harm done to the dancer or the audience is **NOT** an option. [4]

5.4. Validation

As a final step to all the previous development there needs to be a form of final validation, which ensures that the developed system performs as intended in the environment that it has been designed for.

For this 2 final tests have been envisioned:

- The 'Human-Discriminator' test
- Live inference with dancer

5.4.1. 'Human-Discriminator' Test

As proposed in section 4.5, an experiment is to be developed and executed that may prove the perceptual validity of the final motion generation.

As the final audience for this research will be the general public, in theory the test can be executed with any participants. In practice the most meaningful results will be achieved by using the creators of the original data as participants. These include the choreographer, as well as the participating dancers.

5.4.2. Live Inference with Dancer

The primary issue with training on pre-existing data is that we're generating new motions for the 'output' dancer, but the 'input' dancer will keep on repeating the same moves independent of the newly generated motions of the 'output' dancer.

In reality two dancers improvising together is a constant feedback loop between the two dancers motions. This effect was attempted to be minimized by the introduction of a specified 'leading' and 'following' dancer in every take, however it can never truly be removed.[5]

Therefore it is expected that the introduction of a live dancer interacting with the generation output motions will result in an additional feedback loop that has not been properly assessed in the models training procedures.

There is enough budget left over for a full day of live inference testing with the dancers to assess the validity of the model in a realistic environment and to see the extent of the existence of a *'reality-gap'* for the final model.

[4] I've seen dancers cut by drones before and it is not pleasant!
[5] Except perhaps with a blindfold on the 'input-dancer'

6

Project Planning

For the planning of this project a work-package based approach is used, specifying the various tasks to be performed. However due to the nature of the grounded theory approach there might be several iterations that arise during the research which may adapt the planning presented here. Therefore this planning is to be utilized as a guidance tool rather than an end in itself.

3 weeks of holiday are planned in the summer periods (June/July), however due to the uncertainties surrounding the corona crisis no exact date has been fixed so far.

6.1. Milestones & Deliverables

The following milestones and their associated deliverables, listed in chronological order, are defining the progression of the research; the various primary stakeholders are indicated by the prefix for each milestone:

1. AkoB - Funding proposals
    - Hand-in funding proposal #1 [EU] (20.06.2019)
    - Hand-in funding proposal #2 [Stimuleringsfonds (NL)] (27.01.2020)
2. TU Delft - Initial Project Approval (25.06.2019)
    - Preliminary project proposal V1 (23.06.2019)

    —- Start of thesis project —-

3. TU Delft - Kick-Off Meeting/Formal starting date (07.01.2020/10.02.2020)
    - Preliminary project proposal V2 (04.01.2020)
    - Thesis Kick-off form (02.03.2020)
4. AkoB - Kick-Off Meeting (21.02.2020)
    - Preliminary project planning
    - Requirements for data acquisition
5. **TU Delft - Preliminary Report Hand-in (31.05.2020)**
    - Literature study report
    - Literature study presentation (12.06.2020)
6. TU Delft - Mid-term Review (21.08.2020)
    - List of requirements for the final model
    - Individual model tests finalized
    - Finalization of model evaluation methods
7. TU Delft - Green-light Review (23.10.2020)
    - Architectural definition of the final model
    - Plan for the final optimization strategy

8. TU Delft - Thesis Hand-In (20.11.2020)
    - MSc. Thesis report
9. AkoB - Model Hand-over (27.11.2020)
    - Finalized & trained neural network model
    - Live Visualization environment
    - Documentation for executing and developing the code
10. TU Delft - Thesis Defense (TBD - latest 18.12.2020)
    - Master Thesis presentation

—- End of thesis project —-

11. 'Stimuleringsfonds' - Public Presentation (TBD - ca.12.2020-01.2021)
    - Live demonstration in front of public audience
12. AkoB - 'AI-Man' Theater performance (TBD - ca. 09.2021)

6.2. Work-Packages

Work packages have been defined and are being maintained within my proprietary project management tool 'TaskScheduler'. As of to date a total of 115 work-packages (93 tasks) have been defined, of which 52 have been completed, which represent 37% of the total 8.5 months (approx. 1360h) workload to be completed.

The major work packages that have been envisioned for the development phase are directly linked to the structure of the development plan, in chapter 5.

The detailed overview and time-line can be found within the Gantt-chart in Appendix C.

7

Conclusions

In conclusion the goal of the research is to design and deploy a NN based drive algorithm for a virtual skeleton that is capable of live interaction with a human dancer.

The related research objective is as follows:

> *'to design an algorithm that generates a virtual skeleton in reaction to motion-captured dance improvisations [...], by comparing existing and creating new algorithms, within the domain of deep-learning architectures, capable of 'fooling' the creator of the original data, beyond reasonable doubt. '*

This is accompanied by the following research hypothesis:

> *'Neural networks are capable of generating natural motion in reaction to live full-body human input, which is indistinguishable of real motion, for a human expert.'*

The external goal of this research will be of practical nature and comprises of an inference NN model, to be run on 'conventional hardware' (e.g. gaming PC) and a 3D visualization environment for the dancer to engage with.

The internal goal of this research is to explore the applicability and performance of NNs for real-time motion synthesis for human-machine interaction.

7.1. Expected Results

The desired results of this research will be a functional demonstrator of a NN driven virtual skeleton, which is capable of choosing appropriate reactions in real-time to an improvising dancer. The skeletal motions will be presented in the same streaming format as the original input motion data, to ensure compatibility with the original drone swarm control system.

In a nutshell it is desired to obtain quantitative knowledge and best practices to train a NN on human movement and generate natural, dynamic and fluent reactionary motions. This will allow not only AkoB, but also future researchers and animators, to bridge the gap between art and technology to create more immersive real-time human-machine interactions.

Ultimately there are 4 resulting products that this research desired to produce:

- **Large public MoCap dataset:**
  A clean and accessible version of the vast dataset, as recorded by AkoB
- **Functional motion model:**
  Including the working knowledge of the underlying principles allowing its operations
- **Functional visualization environment:**
  Allowing the researcher to review the data and a dancer to interact with the system
- **Statistical analysis of the obtained results:**
  Providing a quantifiable measure for the external validity of the research

7.2. Research Gap

While the field of AI and NNs is continuously and rapidly expanding, the applicability of the field is seemingly endless and the highly data depended nature of this methodology makes it hard to find applicable literature for niche applications. This results in a lot of fundamental literature being available, while very little are actually directly applicable to the research at hand. Some researcher show signs of exploration in the direction of live human-machine interactions within a creative setting, but this particular field appears to have a lot of potential to expand and grow.

Particularly most papers on NNs solely base their performance on the optimization of the loss-function alone, while for creative settings the perceptual aspect is completely ignored. This research intends to quantify this 'perceived' performance, bridging the gap between objective and subjective evaluation.

Personally I believe that this research can give deeper insight into how neural networks 'understand' motion and open up a lot of opportunities for artists around the world to engage in creative interaction with technology. Hopefully bridging the gap between art and technology in a creative and exciting manner, such as AkoB has already successfully demonstrated over the past decade.

7.3. Project reflection

Reflecting on the experimental setup and planning, it can be said that in the light of the uncertainty surrounding the exact outcome of the research, the proposed strategy of a grounded theory approach is deemed most appropriate, to develop theories on how the vast field of AI can best be applied to the specific niche application at hand. The basis of this assumption is that by applying various strategies on the same data, new fundamental correlations will be discovered that can provide the researcher with new insights that should lead to the development of new theories and models that can be tested, with the underlying assumption of improving the model at hand.

Once the theoretical basis for the model has been fully established this will allow the researcher to optimize the model for practical application, to the best of his knowledge and based on empirically backed experience.

Scope The researcher is well aware that the scope of the proposed research is large and will require a lot of time and work. This is why some research questions have internally been marked as 'optional' and the intention is to fully define the necessity of some approaches together with the supervisor, during or soon-after the 'preliminary green-light meeting'. All options have been left in this report, as to highlight all possibilities and for the supervisor to reflect on their expected applicability.

7.4. Future Development - What is next?

The final product following this research will be a live virtual performance of a dancer within the OptiTrack system and a projection of the virtual environment in which the drone swarm is to be seen. This will be a public performance towards the end of 2020.

Continuing the previous developments as outlined in chapter 1, the final form of this research will result in the actual showpiece 'AI-Man' to be performed in theaters during the season of 2021, where the control of the physical drones, from 'Airman', will be added to the system.

AI-Man: As an initial step the dancer will be equipped with appropriate VR gear to rehearse together with the new AI in a common environment. The final form that is chosen for the machine is a swarm of drones that can form a human but also other shapes, as utilized prior in 'Airman'. The control algorithms of this swarm will need to be augmented with appropriate anti-collision routines to ensure safe execution of the new and unknown outputs and ensure that no drone were to harm the dancer or the audience. Finally the entire show will be taken on tour around the Netherlands and showcased on international stages as well.

Continuation Summary: After completion of the project the work is intended to be continued, in the form of a public presentation with 'virtual drones' by the end of 2020, and followed the 'real' theater performance 'AI-Man' to be seen in theaters by the end of 2021, made possible by 'Stimuleringsfonds', 'Another kind of Blue' & 'CompactCopters'.

IV

Addendum

Bibliography

A. Aristidou, D. Cohen-Or, J. K. Hodgins, and A. Shamir. Self-similarity analysis for motion capture cleaning. *Computer Graphics Forum*, 37(2):297–309, 2018a. ISSN 14678659. doi: 10.1111/cgf.13362.

Andreas Aristidou, Daniel Cohen-Or, Jessica K. Hodgins, Yiorgos Chrysanthou, and Ariel Shamir. Deep motifs and motion signatures. *SIGGRAPH Asia 2018 Technical Papers, SIGGRAPH Asia 2018*, 37(06), 2018b. ISSN 15577368. doi: 10.1145/3272127.3275038.

Joel Auslander, Alex Fukunaga, Hadi Partovi, Jon Christensen, Lloyd Hsu, Peter Reiss, Andrew Shuman, Joe Marks, and J. Thomas Ngo. Further Experience with Controller-Based Automatic Motion Synthesis for Articulated Figures. *ACM Transactions on Graphics (TOG)*, 14(4):311–336, 1995. ISSN 15577368. doi: 10.1145/225294.225295.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. 2018. arXiv-ID: arXiv:1803.01271v2.

Ilya Baran and Jovan Popovi´. Automatic Rigging and Animation of 3D Characters. *ACM SIGGRAPH*, 2007. ISSN 13000144.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013. ISSN 01628828. doi: 10.1109/TPAMI.2013.50. arXiv-ID: 1206.5538.

Amit Bleiweiss, Dagan Eshar, Gershom Kutliroff, Alon Lerner, Yinon Oshrat, and Yaron Yanai. Enhanced interactive gaming by blending full-body tracking and gesture animation. *ACM SIGGRAPH ASIA 2010 Sketches, SA'10*, pages 2–3, 2010. doi: 10.1145/1899950.1899984.

A. Bondu, V. Lemaire, and M. Boullé. Exploration vs. exploitation in active learning: A Bayesian approach. In *Proceedings of the International Joint Conference on Neural Networks*, 2010. ISBN 9781424469178. doi: 10.1109/IJCNN.2010.5596815.

Caroline Chan, Shiry Ginosar, Tinghui Zhou, and Alexei Efros. Everybody dance now. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2019-Octob, 2019. ISBN 9781728148038. doi: 10.1109/ICCV.2019.00603.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pages 1724–1734, 2014. doi: 10.3115/v1/d14-1179. arXiv-ID: 1406.1078.

Junyoung Chung. Gated Recurrent Neural Networks on Sequence Modeling. pages 1–9, 2014. arXiv-ID: arXiv:1412.3555v1.

Qiongjie Cui, Huaijiang Sun, Yupeng Li, and Yue Kong. A deep bi-directional attention network for human motion recovery. *IJCAI International Joint Conference on Artificial Intelligence*, 2019-Augus:701–707, 2019. ISSN 10450823. doi: 10.24963/ijcai.2019/99.

Pedro Domingos. A Few Useful Things to Know about Machine Learning.

Minjing Dong and Chang Xu. On retrospecting human dynamics with attention. *IJCAI International Joint Conference on Artificial Intelligence*, 2019-Augus:708–714, 2019. ISSN 10450823. doi: 10.24963/ijcai.2019/100.

K Eggensperger, M Feurer, F Hutter, J Bergstra, J Snoek, H Hoos, and K Leyton-Brown. Towards an Empirical Foundation for Assessing Bayesian Optimization of Hyperparameters. *BayesOpt workshop (NIPS)*, 2013.

Felix Gaisbauer, Jannes Lehwald, Philipp Agethen, Julia Sues, and Enrico Rukzio. Proposing a co-simulation model for coupling heterogeneous character animation systems. *VISIGRAPP 2019 - Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, 1(Visigrapp):65–76, 2019a. doi: 10.5220/0007356400650076.

Felix Gaisbauer, Jannes Lehwald, Janis Sprenger, and Enrico Rukzio. Natural posture blending using deep neural networks. *Proceedings - MIG 2019: ACM Conference on Motion, Interaction, and Games*, 2019b. doi: 10.1145/3359566.3360052.

Felix Gaisbauer, Eva Lampen, Philipp Agethen, and Enrico Rukzio. Combining heterogeneous digital human simulations: presenting a novel co-simulation approach for incorporating different character animation technologies. *Visual Computer*, 2020. ISSN 01782789. doi: 10.1007/s00371-020-01792-x.

Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image Style Transfer Using Convolutional Neural Networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-Decem:2414–2423, 2016. ISSN 10636919. doi: 10.1109/CVPR.2016.265.

Partha Ghosh, Jie Song, Emre Aksan, and Otmar Hilliges. Learning human motion models for long-Term predictions. *Proceedings - 2017 International Conference on 3D Vision, 3DV 2017*, pages 458–466, 2018. doi: 10.1109/3DV.2017.00059.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS 2014)*, pages 2672–2680, 2014.

Anirudh Goyal, Alex Lamb, Ying Zhang, Saizheng Zhang, Aaron Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. *Advances in Neural Information Processing Systems*, (Nips 2016):4608–4616, 2016. ISSN 10495258. arXiv-ID: 1610.09038.

Liang Yan Gui, Yu Xiong Wang, Xiaodan Liang, and José M.F. Moura. Adversarial Geometry-Aware Human Motion Prediction. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11208 LNCS, pages 823–842. Springer Verlag, 2018. ISBN 9783030012243. doi: 10.1007/978-3-030-01225-0_48.

Ikhsanul Habibie, Daniel Holden, Jonathan Schwarz, Joe Yearsley, and Taku Komura. A recurrent variational autoencoder for human motion synthesis. *British Machine Vision Conference 2017, BMVC 2017*, 2017. doi: 10.5244/c.31.119.

Alejandro Hernandez, Jurgen Gall, and Francesc Moreno. Human motion prediction via spatio-temporal inpainting. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-Octob(Iccv):7133–7142, 2019. ISSN 15505499. doi: 10.1109/ICCV.2019.00723. arXiv-ID: 1812.05478.

Geoffrey E. Hinton and Richard S. Zemel. Autoencoders, Minimum Description Length and Helmholtz Free Energy. In *Advances in neural information processing systems*, volume 6, pages 3–10, 1994.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997. ISSN 08997667. doi: 10.1162/neco.1997.9.8.1735.

Alex O. Holcombe. Seeing slow and seeing fast: two limits on perception. *Trends in Cognitive Sciences*, 13(5): 216–221, 2009. ISSN 13646613. doi: 10.1016/j.tics.2009.02.005.

Daniel Holden. Robust solving of optical motion capture data by denoising. *ACM Transactions on Graphics*, 37 (4):1–12, 2018. ISSN 15577368. doi: 10.1145/3197517.3201302.

Daniel Holden, Jun Saito, and Taku Komura. Learning an inverse rig mapping for character animation. In *Proceedings - SCA 2015: 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 165–173, 2015. ISBN 9781450334969. doi: 10.1145/2786784.2786788.

Daniel Holden, Jun Saito, and Taku Komura. A deep learning framework for character motion synthesis and editing. *ACM Transactions on Graphics*, 35(4), 2016. ISSN 15577368. doi: 10.1145/2897824.2925975.

Daniel Holden, Taku Komura, and Jun Saito. Phase-Functioned Neural Networks for Character Control. *ACM Transactions on Graphics*, 36(4), 2017a.

Daniel Holden, Jun Saito, and Taku Komura. Learning Inverse Rig Mappings by Nonlinear Regression. 23(3): 1167–1178, 2017b.

Jensen Huang. Accelerating AI with GPUs: A New Computing Model. *Nvidia*, 2016.

Du Q. Huynh. Metrics for 3D rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 2009. ISSN 09249907. doi: 10.1007/s10851-009-0161-2.

John Kender and One Microsoft Way. HP-GAN : Probabilistic 3D human motion prediction via GAN Emad Barsoum. 2018. doi: 10.1109/CVPRW.2018.00191.

Taku Komura, Ikhsanul Habibie, and Jonathan Schwarz. Data-Driven Character Animation Synthesis Taku. *Handbook of Human Motion*, pages 1–29, 2017. doi: 10.1007/978-3-319-30808-1.

Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. 2015. doi: 10.1038/nature14539.

Chen Li. Convolutional Sequence to Sequence Model for Human Dynamics. pages 5226–5234, 2018. doi: 10.1109/CVPR.2018.00548.

Maosen Li, Siheng Chen, Xu Chen, Ya Zhang, Yanfeng Wang, and Qi Tian. Symbiotic Graph Neural Networks for 3D Skeleton-based Human Action Recognition and Motion Prediction. pages 1–19, 2019. arXiv-ID: 1910.02212.

Zachary C Lipton and Jacob Steinhardt. Troubling Trends in Machine Learning Scholarship. pages 1–15, 2018. arXiv-ID: arXiv:1807.03341v2.

Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E. Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 2017. ISSN 18728286. doi: 10.1016/j.neucom.2016.12.038.

Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael Black. AMASS: Archive of motion capture as surface shapes. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2019-Octob, 2019. ISBN 9781728148038. doi: 10.1109/ICCV.2019.00554. arXiv-ID: 1904.03278.

Alessandro Manzi, Laura Fiorini, Raffaele Limosani, Paolo Dario, and Filippo Cavallo. Two-person activity recognition using skeleton data. *IET Computer Vision*, 12(1), 2018. ISSN 17519640. doi: 10.1049/iet-cvi.2017.0118.

M L Mar, Abdelrahman Mohamed, Matthai Philipose, Matt Richardson, and Rich Caruana. Do Deep Convolutional Nets Really Need to be Deep and Convolutional? (2014):1–13, 2017. arXiv-ID: arXiv:1603.05691v4.

Gary Marcus. Deep Learning: A Critical Appraisal. pages 1–27, 2018. arXiv-ID: 1801.00631.

Julieta Martinez, Michael J. Black, and Javier Romero. On human motion prediction using recurrent neural networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:4674–4683, 2017. doi: 10.1109/CVPR.2017.497. arXiv-ID: 1705.02445.

Radford M. Neal. Bayesian Learning for Neural Networks (Lecture Notes in Statistical Vol. 118). *Journal of the American Statistical Association*, 92(438):791, 1995. ISSN 01621459. doi: 10.2307/2965731.

Dario Pavllo and Michael Auli. QuaterNet : A Quaternion-based Recurrent Model for Human Motion. 2018. arXiv-ID: arXiv:1805.06485v2.

Dario Pavllo, Z Eth, Christoph Feichtenhofer, and David Grangier. 3D human pose estimation in video with temporal convolutions and semi-supervised training. pages 7745–7754, 2019a. doi: 10.1109/CVPR.2019.00794.

Dario Pavllo, Christoph Feichtenhofer, Michael Auli, and David Grangier. Modeling Human Motion with Quaternion-Based Neural Networks. *International Journal of Computer Vision*, 128(4):855–872, 2019b. ISSN 15731405. doi: 10.1007/s11263-019-01245-6. arXiv-ID: 1901.07677.

T. Pejsa and I. S. Pandzic. State of the art in example-based motion synthesis for virtual characters in interactive applications. *Computer Graphics Forum*, 29(1):202–226, 2010. ISSN 14678659. doi: 10.1111/j.1467-8659.2009.01591.x.

Xue Bin Peng, Glen Berseth, and Michiel Van De Panne. Dynamic terrain traversal skills using reinforcement learning. *ACM Transactions on Graphics*, 34(4), 2015. ISSN 15577368. doi: 10.1145/2766910.

Harry Pratt, Bryan Williams, Frans Coenen, and Yalin Zheng. FCNN: Fourier Convolutional Neural Networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10534 LNAI:786–798, 2017. ISSN 16113349. doi: 10.1007/978-3-319-71249-9_47.

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and U C Berkeley. Do CIFAR-10 Classifiers Generalize to CIFAR-10 ? pages 1–25, 2018. arXiv-ID: arXiv:1806.00451v1.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should i trust you?" Explaining the predictions of any classifier. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 13-17-Augu:1135–1144, 2016. doi: 10.1145/2939672.2939778. arXiv-ID: arXiv:1602.04938v3.

David E. Rumelhart, Geoffrey E. Hintont, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

Ju Shen and Jianjun Yang. Automatic Human Animation for Non-Humanoid 3D Characters. In *Proceedings - 2015 14th International Conference on Computer-Aided Design and Computer Graphics, CAD/Graphics 2015*, pages 220–221, 2016. ISBN 9781467380201. doi: 10.1109/CADGRAPHICS.2015.31.

Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Josh Susskind, Wenda Wang, and Russ Webb. Learning from Simulated and Unsupervised Images through Adversarial Training. 2016. arXiv-ID: arXiv:1612.07828v2.

Sebastian Starke, He Zhang, Taku Komura, and Jun Saito. Neural state machine for character-scene interactions. *ACM Transactions on Graphics*, 38(6), 2019. ISSN 15577368. doi: 10.1145/3355089.3356505.

Klaas Jan Stol, Paul Ralph, and Brian Fitzgerald. Grounded theory in software engineering research: A critical review and guidelines. *Proceedings - International Conference on Software Engineering*, 14-22-May-(Aug 2015): 120–131, 2016. ISSN 02705257. doi: 10.1145/2884781.2884833.

Yongyi Tang, Lin Ma, Wei Liu, and Wei-shi Zheng. Long-Term Human Motion Prediction by Modeling Motion Context and Enhancing Motion Dynamic. pages 935–941, 2017.

Thomas Geijtenbeek, Michiel van de Panne, and A. Frank van der Stappen. Flexible Muscle-Based Locomotion for Bipedal Creatures. *ACM Transactions on Graphics*, 32(6), 2013.

Ruben Villegas and Honglak Lee. Neural Kinematic Networks for Unsupervised Motion Retargetting. pages 8639–8648, 2018. doi: 10.1109/CVPR.2018.00901.

Jue Wang, Shaoli Huang, Xinchao Wang, and Dacheng Tao. Not all parts are created equal: 3D pose estimation by modeling bi-directional dependencies of body parts. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-Octob:7770–7779, 2019. ISSN 15505499. doi: 10.1109/ICCV.2019.00786.

Qi Wang, Thierry Artières, Mickael Chen, and Ludovic Denoyer. Adversarial learning for modeling human motion. *Visual Computer*, 36(1):141–160, 2020a. ISSN 01782789. doi: 10.1007/s00371-018-1594-7.

Xin Wang, Xiaotao Jiang, Gloria Rumbidzai Regedzai, Haohao Meng, and Lingyun Sun. Gated neural network framework for interactive character control. *Multimedia Tools and Applications*, 2020b. ISSN 15737721. doi: 10.1007/s11042-020-08792-y.

Junyuan Xie, Linli Xu, and Enhong Chen. Image denoising and inpainting with deep neural networks. *Advances in Neural Information Processing Systems*, 1:341–349, 2012. ISSN 10495258.

Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1316–1324, 2018. ISSN 10636919. doi: 10.1109/CVPR.2018.00143. arXiv-ID: 1711.10485.

Katsu Yamane and Yoshihiko Nakamura. Natural motion animation through constraining and deconstraining at will. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):352–360, 2003. ISSN 10772626. doi: 10.1109/TVCG.2003.1207443.

Katsu Yamane, Yuka Ariki, and Jessica Hodgins. Animating non-humanoid characters with human motion data. *Computer Animation 2010 - ACM SIGGRAPH / Eurographics Symposium Proceedings, SCA 2010*, pages 169–178, 2010.

Xinchen Yan, Akash Rastogi, Ruben Villegas, Kalyan Sunkavalli, Eli Shechtman, Sunil Hadap, Ersin Yumer, and Honglak Lee. MT-VAE: Learning Motion Transformations to Generate Multimodal Human Dynamics. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11209 LNCS, pages 276–293. Springer Verlag, 2018. ISBN 9783030012274. doi: 10.1007/978-3-030-01228-1_17. arXiv-ID: 1808.04545.

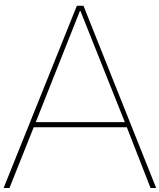
Fan Yang, Yang Wu, Sakriani Sakti, and Satoshi Nakamura. Make skeleton-based action recognition model smaller, faster and better. In *1st ACM International Conference on Multimedia in Asia, MMAsia 2019*, 2019. ISBN 9781450368414. doi: 10.1145/3338533.3366569. arXiv-ID: 1907.09658.

Songyang Zhang, Yang Yang, Jun Xiao, Xiaoming Liu, Yi Yang, Di Xie, and Yueting Zhuang. Fusing geometric features for skeleton-based action recognition using multilayer LSTM Networks. *IEEE Transactions on Multimedia*, 20(9), 2018. ISSN 15209210. doi: 10.1109/TMM.2018.2802648.

Dongsheng Zhou, Xinzhu Feng, Pengfei Yi, Xin Yang, Qiang Zhang, Xiaopeng Wei, and Deyun Yang. 3D Human Motion Synthesis Based on Convolutional Neural Network. *IEEE Access*, 7:66325–66335, 2019a. ISSN 21693536. doi: 10.1109/ACCESS.2019.2917609.

Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019b. ISBN 9781728132938. doi: 10.1109/CVPR.2019.00589. arXiv-ID: 1812.07035.

Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutnik, and Jürgen Schmidhuber. Recurrent highway networks. *34th International Conference on Machine Learning, ICML 2017*, 8:6346–6357, 2017. arXiv-ID: 1607.03474.



Machine Learning Algorithms Overiew

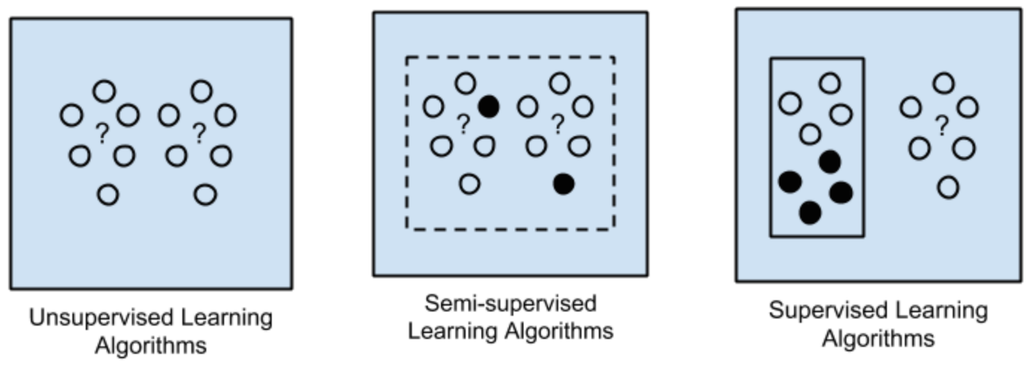
On the next page an overview of the various machine learning algorithms and their applicability is presented.

Machine Learning Algorithms

Based on: https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/

Legend

Not researched	Not applicable	Sub-optimally applicable	Possibly applicable	Applicable

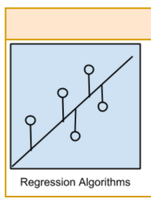


Unsupervised Learning Algorithms

Semi-supervised Learning Algorithms

Supervised Learning Algorithms

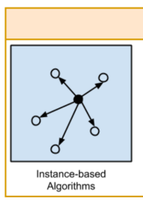
Regression Algorithms



Regression Algorithms

- Ordinary Least Squares Regression (OLSR)
- Linear Regression
- Logistic Regression
- Stepwise Regression
- Multivariate Adaptive Regression Splines (MARS)
- Locally Estimated Scatterplot Smoothing (LOESS)

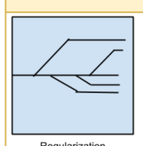
Instance-based Algorithms



Instance-based Algorithms

- K-Nearest Neighbor (KNN)
- Learning Vector Quantization (LVQ)
- Self-Organizing Map (SOM)
- Locally Weighted Learning (LWL)
- Support Vector Machines (SVM)

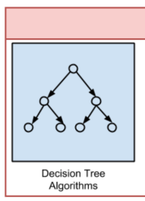
Regularization Algorithms



Regularization Algorithms

- Ridge Regression
- Least Absolute Shrinkage and Selection Operator (LASSO)
- Elastic Net
- Least-Angle Regression (LARS)

Decision Tree Algorithms



Decision Tree Algorithms

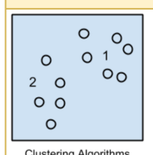
- Classification and Regression Tree (CART)
- Iterative Dichotomiser 3 (ID3)
- C4.5 & C5.0
- Chi-squared Automatic Interaction Detection (CHAID)
- Decision Stump
- M5
- Conditional Decision Trees

Bayesian Algorithms

Bayesian Algorithms

- Naive Bayes
- Gaussian Naive Bayes
- Multinomial Naive Bayes
- Averaged One-Dependence Estimators (AODE)
- Bayesian Belief Network (BBN)
- Bayesian Network (BN)
- Bayesian Optimization

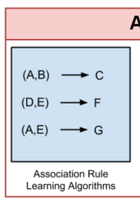
Clustering Algorithms



Clustering Algorithms

- K-Means
- K-Medians
- Expectation Maximisation (EM)
- Hierarchical Clustering

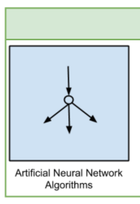
Association Rule Learning Algorithms



Association Rule Learning Algorithms

- Apriori algorithm
- Eclat algorithm

Artificial Neural Network Algorithms



Artificial Neural Network Algorithms

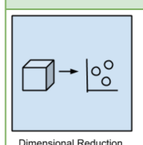
- Perceptron
- Multilayer Perceptrons (MLP)
- Back-Propagation
- Stochastic Gradient Descent
- Hopfield Network
- Radial Basis Function Network (RBFN)

Deep Learning Algorithms

Deep Learning Algorithms

- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Long Short-Term Memory (LSTM)
- (Stacked) Auto-Encoders
- Deep Boltzmann Machine (DBM)
- Deep Belief Networks (DBN)
- Generative adversarial Network (GAN)

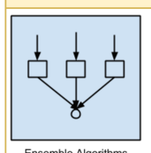
Dimensionality Reduction Algorithms



Dimensional Reduction Algorithms

- Principal Component Analysis (PCA)
- Principal Component Regression (PCR)
- Partial Least Squares Regression (PLSR)
- Sammon Mapping
- Multidimensional Scaling (MDS)
- Projection Pursuit
- Linear Discriminant Analysis (LDA)
- Mixture Discriminant Analysis (MDA)
- Quadratic Discriminant Analysis (QDA)
- Flexible Discriminant Analysis (FDA)

Ensemble Algorithms



Ensemble Algorithms

- Boosting
- Bootstrapped Aggregation (Bagging)
- AdaBoost
- Weighted Average (Blending)
- Stacked Generalization (Stacking)
- Gradient Boosting Machines (GBM)
- Gradient Boosted Regression Trees (GBRT)
- Random Forest

Miscellaneous Algorithms

?

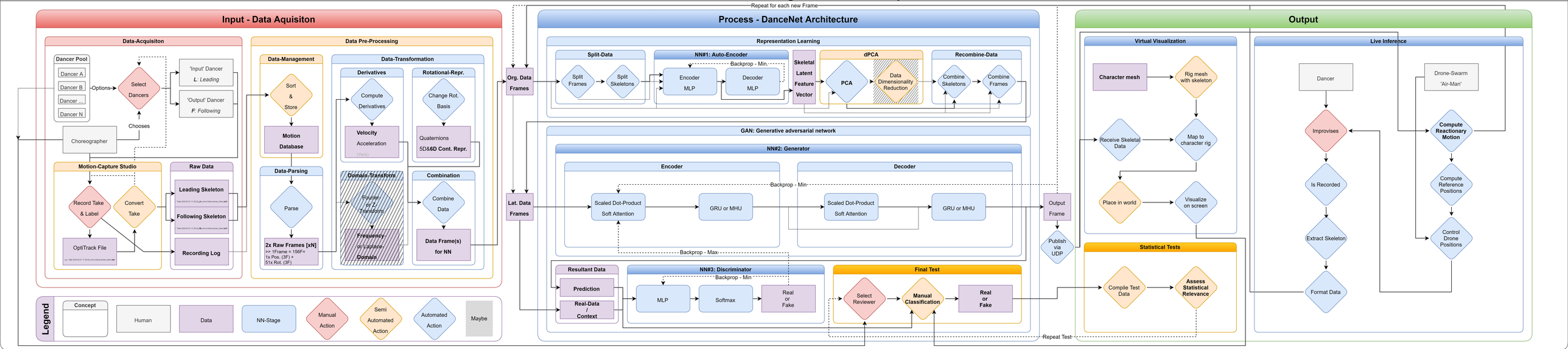
- Reinforcement learning (RL)
- Evolutionary Algorithms
- AdaBoost

B

Data-processing Framework

On the next page the detailed data processing framework for the model to be design is presented. This is the extended version of the summary presented in figure 5.4, in chapter 5.

AI-Man: Data-Processing Framework Proposal



C

Gantt Chart

On the next page an overview of the project time-line, including all work packages to be performed, is summarized in a gantt-chart.

Legend:

- Boxes: Task
  - Blue: Parent Task, contains sub-tasks (creates grey bounding box around sub-tasks)
  - Grey: To be completed
  - Yellow: Currently active
  - Green: Finished
  - Red: Canceled
  - Light Blue-bar: Estimated progress-bar
- Vertical lines:
  - Light Grey - Dashed: Day delimiter
  - Light Grey - Solid: Week delimiter
  - Dark Grey - Solid: Month delimiter
  - Green: Current date

February (2020)
March (2020)
April (2020)
May (2020)
June (2020)
July (2020)
August (2020)
September (2020)
October (2020)
November (2020)
December (2020)
CW#2
CW#3
CW#4
CW#5
CW#6
CW#7
CW#8
CW#9
CW#10
CW#11
CW#12
CW#13
CW#14
CW#15
CW#16
CW#17
CW#18
CW#19
CW#20
CW#21
CW#22
CW#23
CW#24
CW#25
CW#26
CW#27
CW#28
CW#29
CW#30
CW#31
CW#32
CW#33
CW#34
CW#35
CW#36
CW#37
CW#38
CW#39
CW#40
CW#41
CW#42
CW#43
CW#44
CW#45
CW#46
CW#47
CW#48
CW#49
CW#50
CW#51