Latent Space Modelling of Unsteady Flow Subdomains

Thesis Report

B. Mulder

Technische Universiteit Delft

TU Delft

Latent Space Modelling of Unsteady Flow Subdomains

Thesis Report

by

B. Mulder

Student number: 4100794
Supervisor: Prof. dr. S.J. Hulshoff, TU Delft

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

TU Delft

Abstract

Very complex flows can be expensive to compute using current Computational Fluid Dynamics (CFD) techniques. In this thesis, models based on deep learning were used to replace certain parts of the flow domain, with the objective of replacing well-known regions with simplified models to increase efficiency. To keep the error produced by the deep learning model bounded, a traditional CFD model and deep learning model were coupled using a boundary overlap area. In this overlap area, the flow computed by the traditional CFD model was used by the deep learning model as an input.

It was demonstrated that since traditional CFD model continuously feeds in reliable information into the deep learning domain, the error remains bounded. Furthermore, it was found that the accuracy of the deep learning models depends significantly on the random initial weights. Therefore, deep learning models trained differently must be carefully compared.

Contents

List of Symbols

A Amplitude

a Output of a perceptron

b Bias of a perceptron

E Internal Energy

f Frequency

g Activation function

h_t Hidden state of time at time t

m_s Length of the latent space vector

o Number of inputs for the RNN

ρ Density

τ Time shift at which a wave enters the domain.

u Velocity

W_i Weight applied to perceptrons input i

W_{hh} Weight matrix of of a RNN applied to the hidden state of the previous time step, which results in the hidden state

W_{hx} Weight matrix of of a RNN applied to the input, which results in the hidden state

W_{hy} Weight matrix of of a RNN applied to the hidden state, which results in the output of the RNN layer

x_t Input of a neural network at time t

Abbreviations

AE Autoencoder

ANN Artificial Neural Network

CFD Computational Fluid Dynamics

CNN Convolutional Neural Network

DL Deep Learning

EIA an Euler code for Internal Aeroacoustics

GRU Gated Recurrent Unit

LSTM Long Short Term Memory

MLP Multilayer Perceptron

MSE Mean squared error

NS Navier-Stokes

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

ROM Reduced Order Model

1 Introduction

Complex flows at high Reynolds numbers often require so much computational effort that they are practically impossible to compute. As designing many vehicles and tools involves fluid dynamics, improving the CFD would lead to improvements in design and/or a reduction in design costs.

Currently, CFD techniques often use resources for solving parts of the flow domain where the behaviour is well known and is not of primary interest, yet exhibit complex behaviour necessary for resolving the complete flow. In certain cases it may be possible replace these areas with more efficient models to reduce computation costs. In this thesis, models based on Deep Learning (DL) are developed for this task. DL techniques show promise in many areas such as self-driving cars, playing a variety of games at a superhuman level, image recognition, linguistic modelling, etc. In reference Wiewel et al. [24] it was shown that DL can help to lower the CPU costs of the CFD algorithm. An Autoencoder (AE) was used to reduce the flow dimensionality into a latent space. A Recurrent Neural Network (RNN) was used to predict the next latent space state, based on a few previous latent space states. This model, however, needs a short history of the fluid state as a startup, which might not be feasible in situations where the problem is too complex. The error in their solution also grows exponentially over time. This was solved by running a traditional algorithm each few time steps, which leads to higher computation costs and again might not be possible for situations where the problem is too complex. The question this thesis helps to answer is: Can a DL approach be used to replace a traditional CFD approach in parts of the domain to do the time evolution of the full state of a fluid?

Chapter 2 discusses the relevant literature this thesis uses to built upon and discusses what can and needs to be improved to make a functional DL aided CFD model. The research outline is described in Chapter 3. The method for testing various aspects of the proposed architecture are discussed in Chapter 4. The results and conclusions are described in Chapter 5. The conclusions are presented in Chapter 6. Finally, recommendations for future work is described in Chapter 7.

2

Relevant Literature

This chapter discusses the relevant literature used as inspiration. DL is discussed in Section 2.1. A method of using DL for fluid flow predictions is discussed in Section 2.2

2.1. Deep Learning Networks

This section will dive into the components and successfully used Artificial Neural Network (ANN) architectures. First the perceptron, the main building block of the ANN, is discussed in Section 2.1.1. The Multilayer Perceptron (MLP) layer is discussed in Section 2.1.2. The exploding or vanishing gradient problem is described in Section 2.1.3. The Convolutional Neural Network (CNN) and RNN layers are discussed in Section 2.1.4 and 2.1.5, respectively. An application of the described layers, the AE, is discussed in Section 2.1.6

2.1.1. Perceptron

The main building block for ANN's is the perceptron. As described by Rosenblatt [19], the perceptron has the form depicted in Figure 2.1. A perceptron has a certain number of inputs, x_1 up to x_n. These inputs are

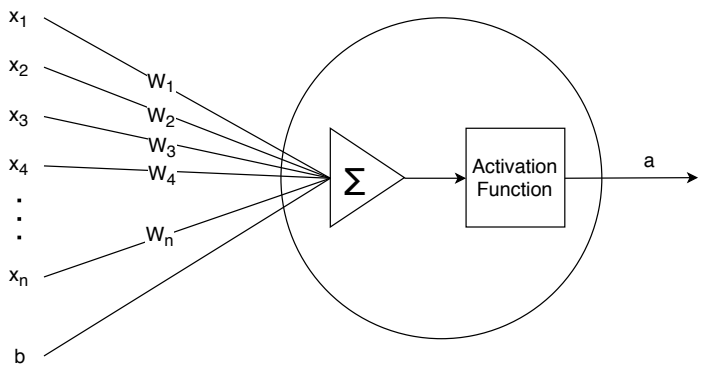

Figure 2.1: Schematic overview of a perceptron.

multiplied by their own weight W_1 up to W_n and consequently summed. Finally, the bias b is added and an activation function is applied, which yields a. The output for a perceptron can thus be calculated using Equations 2.1.

$$a = g\left(\sum_{i=0}^{i=n} x_i W_i + b\right) \tag{2.1}$$

Where g usually was a sigmoid or tanh function, but more recently a Rectified Linear Unit (ReLU) has become more popular as it allows for training deeper networks Nair and Hinton [17]. With a training set mapping x to a, the weights can be adjusted, with a gradient descent algorithm for example, so a desired output gets computed by the perceptron.

2.1.2. Multi-Layered Perceptron

A single perceptron can only predict linear separable problems. It can't approximate a XOR function, for example. It was found that chaining together the perceptron into an MLP solved this issue Rumelhart et al. [21]. A schematic overview of an MLP is given in Figure 2.2.

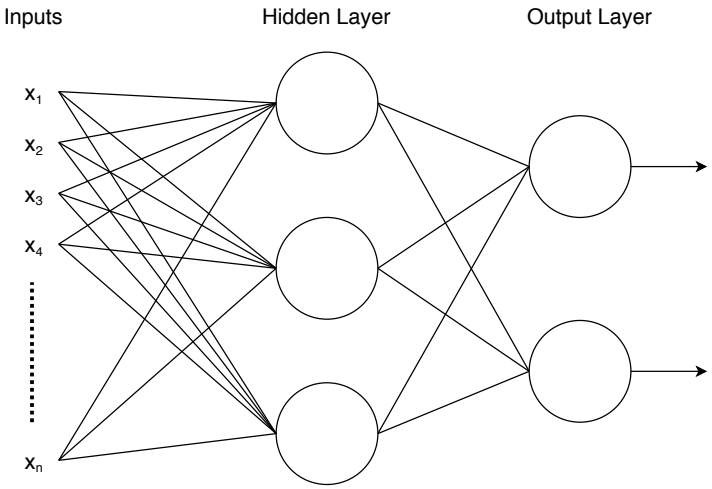

Figure 2.2: Schematic overview of an MLP.

The MLP consists of perceptrons coupled together. Each individual node in Figure 2.2 is a perceptron as depicted in Figure 2.1. Furthermore, each perceptron has its own bias, which isn't indicated in the figure. The MLP has at least a single so called hidden layer, but there can be many. This layer is not interesting as an output but is necessary for approximating non linearly separable functions. The hidden layers eventually feed information to the output layer to give a certain desired output. The desired output can be obtained by setting the right weights and biases for each individual perceptron. An important feature of this setup is that the weights and biases can be trained by giving the ANN examples, using backpropagation. A gradient descent method can be used to set the desired weights for the network. The equation for calculating the output for any layer is given in Equation 2.1

$$\mathbf{a_i} = g(\mathbf{W_i a_{i-1}} + \mathbf{b_i}) \tag{2.2}$$

Where g is the activation function of the perceptron in layer i, W_i is the weight matrix of layer i and b_i is a bias vector of layer i and a_i the output vector of layer i.

2.1.3. The Exploding and Vanishing Gradient Problem

A sidestep from DL is made for the explanation of the exploding and vanishing gradients problem as this has prevented researchers a long time from successfully training deep networks. Neural networks multiply their weights by an activation function. For a one hidden layer network, like the one in Figure 2.2 the output would be,

$$\mathbf{y} = g(\mathbf{Wx} + \mathbf{b})$$

Setting the biases to zero for now and a linear activation function the equation becomes,

$$\mathbf{y} = \mathbf{Wx}$$

When the number of layers increases to an bigger number called l, this equation becomes,

$$\mathbf{y} = \mathbf{W_0 W_1 W_2 \cdots W_{l-2} W_{l-1} W_l x}$$

If the weights would now be bigger than 1, the equation explodes exponentially when increasing the number of layers. If the weights would be smaller than 1, assuming positive weights, the equation exponentially decays to 0. The gradients are linked to these weights, which results in very big or small weight updates.

When the activation function are not linear but sigmoids, historically often used, this always leads to vanishing gradient, as the output of each layer is always lower than one. ReLU activation functions are often

used nowadays as the activation function can be lower and higher than one and therefore tend to have less problems with exploding or vanishing gradients.

Furthermore, weights initialisation can also help to prevent the exploding and vanishing gradient problem as suggested by Glorot and Bengio [6]. Many DL toolboxes apply these automatically for convenience.

2.1.4. Convolutional Neural Network

Consider the MLP of Figure 2.2. If one would want this network to classify, for example, if there is a cat or a dog on a greyscale image of 1000 x 1000 pixels. This would mean there are 1 million inputs for this network and hence 3 million weights to be trained for the first layer. To make the hidden layer more complicated and thus the network able to approximate a more complex function, the hidden layer size could be increased to e.g. a 1000 nodes. This creates 1 billion trainable parameters. Besides the computational burden, using too much trainable parameters usually leads to overfitting and poor generalisation of neural networks. Therefore this network is not suitable for image classification or similar problems.

Bottou et al. [2] solved this issue by using CNN's. These use filters of a certain width and height. The depth of the filter has to be equal to the depth of the input. Figure 2.3 shows the filters and how they are convolved over the, in this case, image. The pixels in the filters are the filter specific weights and these are trained using backpropagation. When convolving over an image, the filter is placed over the image as depicted on the top left. Each of the pixels of the input image is multiplied by the filter weight it is overlapping with. These resulting values are summed, a bias is added to this number and finally an activation function is applied to the resulting number, as is done with a normal perceptron. The final value on the first layer of the output matrix, as it is the first filter, and is placed in the top left. Now the filter is moved by a number of pixels to the right, which is called the stride (2 in this case), as can be seen in the top right of Figure 2.3. The process described above is repeated here resulting in a number which is placed in the output matrix on the first layer, first row and second column. The filter is moved again 2 pixels to the right and the same process is repeated, placing the output number on the first layer, first row and third column. Now a problem occurs as the filter can't move 2 pixels to the right anymore as the filter would partially fall outside of the input image. Therefore, the image is moved to the next row using the same stride and moved all the way to the left. The output from this is placed in the first layer, second row and first column. The same process is repeated until the whole image is convolved. This is repeated for the number of filters that are chosen as indicated in bottom of Figure 2.3. As can be seen, the output of the convolution process has shrunk in width and height in comparison to the input. This is partially caused by setting the stride to 2, so if the stride would have been 1 the output width and height would have been larger. However, padding should be used to keep the output the same size as the input (if a stride of 1 is used). Padding works as follows, if the filter falls partially outside of the input image, the image is padded with zeros (zero padding) or with the same pixels as the border (same padding). Now these areas can also be used for the convolution process resulting in the output width and height being the same as the input image for a stride of 1. If the stride is bigger than one this would mean the image size would shrink by the factor of the stride rounded up to an integer.

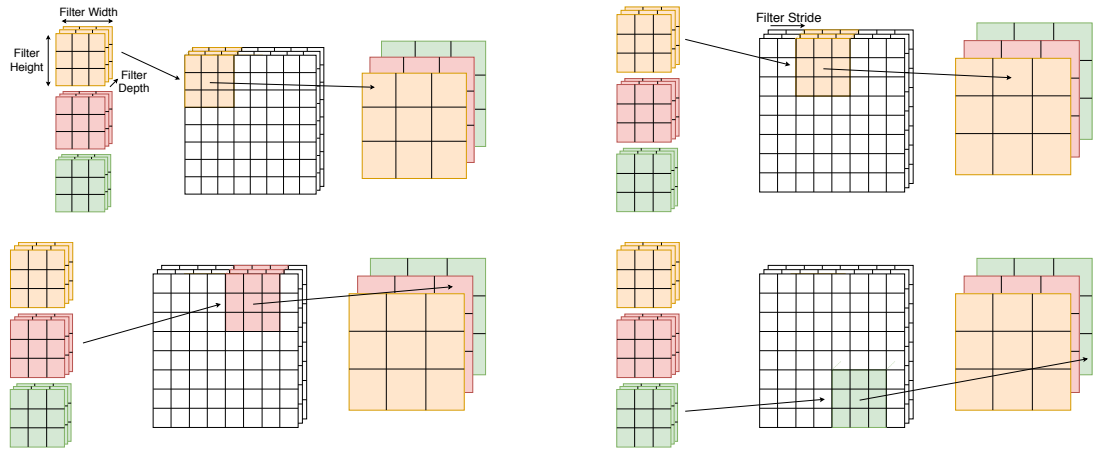

Figure 2.3: Convolution performed by 3 filters using a stride of 2 and no padding.

Figure 2.4 gives an schematic overview of a deep ANN. The image is an example of an RGB image and thus

has 3 channels. A convolution is applied to this 3 features by several filters. After a convolution step, usually a max pooling step is used to reduce the width and height of the filters. The convolution and max pooling steps are repeated a couple of times, after which the filters are flattened and connected to a fully connected layer (i.e. a MLP layer). This is possible now as the dimensions have greatly been reduced. This layer can, for example, classify what is on the image.

If this architecture were to be used on a greyscale image (i.e and image with 1 channel) of a 1000 x 1000 pixels with 64 filters with a width and height of 3 (and a depth of 1 as it is a greyscale image) the number of weights can be calculated. Each filter has 9 filter weights (width * height * depth), which results in 576 weights for all filters. Furthermore each filter has a filter specific bias so 64 has to be added. This results in 640 trainable parameters for this layer. This is significantly lower than a MLP would have. That is why these network setups are favourable in image classification tasks, but also have many other uses where there is a input with large dimensionality.

It should be noted that the "Convolution" steps performed in a convolutional neural network aren't really convolutions. If a real convolution would have been applied the filters' layers should first have been flipped (inverting both the order of the columns and rows). The action applied by the filters is actually called a cross-correlation. For MLP's this does not make a difference as the filters are trained anyway, so the weights are still changing. Due to the similarity, the networks have been called convolutional neural networks while for correctness they should be called cross-correlation neural networks.

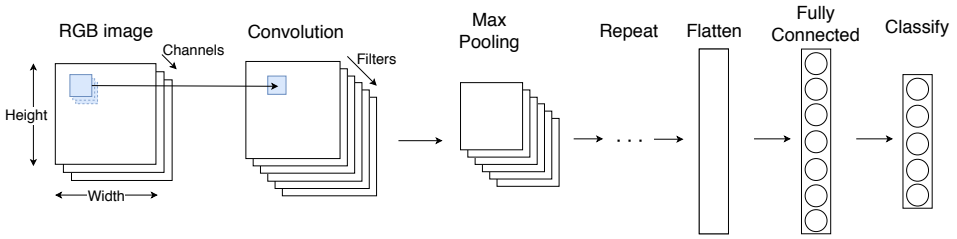

Figure 2.4: Schematic overview of a basic MLP.

Many improvements were made since Bottou et al. [2]. Krizhevsky et al. [15] started a new revolution in DL by using ReLU activation functions for the perceptrons and using dropout to overcome overfitting. Due to the large number of hyper-parameters in MLP's (i.e. number of layers, filters per layer, filter window size, filter stride, pooling size, etc.) it is labour intensive to design the most optimal network architecture. Szegedy et al. [22] overcomes this problem by stacking different outputs of different layer setups together as the input for the next layers, which repeats this process. It is shown that this substantially improves the networks performance and reduces network setup time. They named the network after the movie Inception, as it could achieve deeper network structure than before. Training very deep MLP's remained a problem as very deep MLP's generally performed worse than MLP's with less layers. He et al. [7] overcame this problem by introducing residual networks. In these networks the layers input is added to the output. This way the network only has to predict the residual in each layer. They show that in this way adding layers doesn't harm the MLP's anymore and might even improve the overall network performance. A combination of the above described architectures was used in Szegedy et al. [23] using a residual-inception network which outperformed both of the networks it was based upon. Squeeze-and-excitation units were added to the previously mentioned network to push the performance even further. These units look at the different channels that each residual-inception unit outputs. It determines which of these are most relevant for the task at hand and increases their contribution while lowering the less important channels Hu et al. [10].

2.1.5. Recurrent Neural Network

Some problems might be time dependent or alternatively depend on a previous sequence of words. The previously described networks lose all information after the network completes executing. These networks are therefore unsuitable for any sequence prediction tasks. The solution for this are RNN, first described by Hopfield [9].

The setup for a RNN layer is somewhat different than that from a MLP as can be seen in Figure 2.5. An RNN layer actually has 2 layers of perceptrons inside. The first perceptron layer gets the inputs at time t, $\mathbf{x_t}$, and also takes in a hidden state of the previous time step, $\mathbf{h_{t-1}}$, which both are vectors. $\mathbf{x_t}$ and $\mathbf{h_{t-1}}$ get multiplied by their own weight matrix $\mathbf{W_{hx}}$ and $\mathbf{W_{hh}}$ respectively. A bias is added to the result and an activation function,

g_h, is applied to the the final value. The output from this perceptron layer is called the hidden state, $\mathbf{h_t}$. This hidden state is again multiplied by its own weight matrix, $\mathbf{W_{hy}}$, a bias is added and an activation function, g_y is performed. The output from this layer is the output from the RNN layer, y_t. The hidden state the allows the network to keep information from previous runs. Using this architecture, the ANN can remember its previous states and based on the new input, can do something with knowledge of both. The hidden state can be calculated with Equation 2.3 and the output can then be calculated using 2.4.

$$h_t = g_h\left(\mathbf{W_{hx}x_t} + \mathbf{W_{hh}h_{t-1}} + \mathbf{b_h}\right) \tag{2.3}$$

$$y_t = g_y\left(\mathbf{W_{yh}h_t} + \mathbf{b_y}\right) \tag{2.4}$$

Training RNN's was regarded hard because of the problem of vanishing or exploding gradients Lipton et al. [16]. Looking at the h_t, which is used as h_{t-1} concatenated with $\mathbf{x_t}$, to produce a new h_t. Remembering the h_{t-10} for example is hard for this network due to the vanishing or exploding gradients, as information from h_{t-10} had pass though the first node layer 10 times, causing the information to vanish or explode.

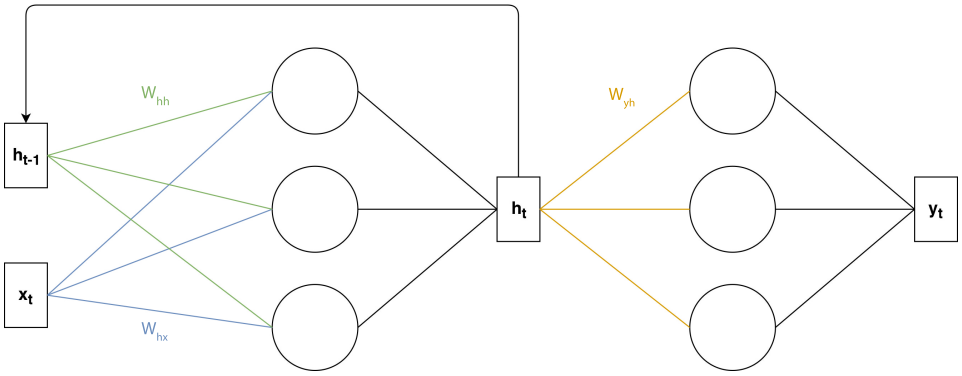

Figure 2.5: Schematic overview of a basic RNN.

The Long Short Term Memory (LSTM) cell mitigated the problem of vanishing or exploding gradients and made it easier to train the RNN Hochreiter and Schmidhuber [8]. Furthermore, a Gated Recurrent Unit (GRU) RNN type was designed more recently as it performs in most cases as good as the LSTM but has lower running cost Chung et al. [5]. As the LSTM and GRU layers are quite complex and technical is not described in this thesis. However, for an overview on how LSTM layers work, it is suggested to look at Olah [18]. For the GRU layer it is suggested to look at Kostadinov [14].

2.1.6. Autoencoder

Dimensionality reduction is commonly used for boosting the efficiency of algorithms in many different fields. ANN's also have a network setup to achieve a dimensionality reduction, called an autoencoder. The autoencoder consists of an encoder network and decoder network. The encoder maps the high dimensional data to a lower dimensional representation, called the latent space. The encoder does this by first taking in the inputs in the input layer. This layer is as big as the high dimensional dataset. Each subsequent hidden layer has fewer and fewer nodes until a satisfactory number of nodes is left. This layer is the output layer for the encoder and the input for the decoder and is also called the "bottleneck". The decoder network transforms the latent space back into the high dimensional data set by increasing the number of nodes again until the number of nodes is equal to the number of inputs. Autoencoders on itself don't have many applications except for denoising images Chaitanya et al. [3]. However, the latent space they produce can be used as a lower the dimensionality of a certain problem, do calculations on, and feed back into the decoder to get the high dimensional output again. A basic layout of a convolutional autoencoder is given in Figure 2.6.

2.2. Applying Deep Learning to CFD

A successful approach DL aided CFD algorithm was found by Wiewel et al. [24]. The paper uses experiments in a 3D fluid environment based on the Navier-Stokes (NS) equations, which has the challenge of having a high dimensionality. Three different data set are used in which two data sets contain fluids with a resolutions of 64^3 and 128^3. The other data set contains smoker simulation with a 128^3 resolution.

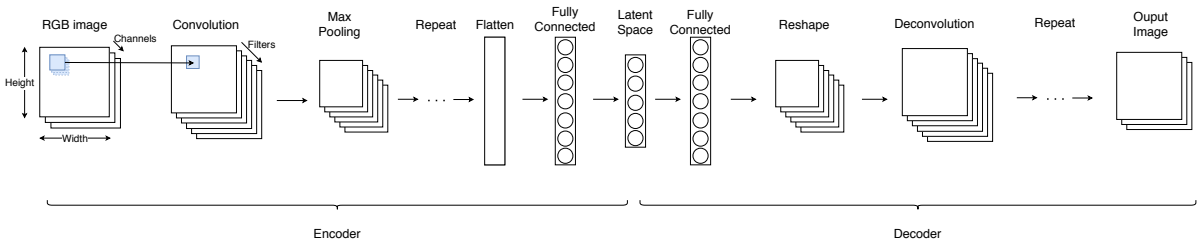

Figure 2.6: Schematic overview of a convolutional Autoencoder.

The problem for a DL approach is that the state should be evolved over time on this high dimensionality. The paper therefore suggests to use an autoencoder to encode that data set into a latent space Rumelhart et al. [20], which compresses the data set by a factor 256. In Wiewel et al. [24] a normal autoencoder was compared to the a variation autoencoder Kingma and Welling [13] as well as fluid properties were encoded to see which would result in best predictions.

Now the dimensionality of the problem has been greatly reduced, a time evolution network can be used. In Wiewel et al. [24] assessment is made for two different time evolution networks. The first is a fully recurrent setup where three layers of LSTM are used. In LSTM's each node in the new layer connects to every node in previous layer. The authors indicate that reducing the amount of weights reduces the chances in overfitting and training times. Therefore the second setup is used where two layers LSTM are used after which a 1D convolution layer is used to provide the output. This greatly reduces the amount of weights and thus the chances of overfitting and training time. These RNN networks use the last few time steps to predict a new time step. This new time step can be used as new input to the RNN resulting in a loop which can continue indefinitely. Predictions of the RNN are in latent dimensions and can be decoded user the decoder network. This results in the time evolution prediction of the fluid state.

Conclusions in this paper are that using the fluid can indeed be predicted using a ANN setup. The best way to do that was by encoding and predicting the pressure. Trails in which the velocity was encoded and predicted resulted in a lower accuracy. Furthermore, the normal autoencoder outperformed the variational autoencoder.

Limitations of this model are that the error increases unbounded over time. To counter this a traditional CFD update is done every few time steps. This increases the accuracy over longer time periods. This increases the computation time again and in certain flow conditions traditional CFD cannot be used. Another limit is that for startup a few time steps of the fluid history are needed. These might also not be available in certain flow conditions. Lastly, it was noticed that the different encoding methods and RNN architectures were trained only once and their results compared. The stochastic nature of DL can have had an impact on the accuracy resulting in an unfair comparison between the different methods and architectures.

3

Research Outline

This chapter defines what current the gaps in knowledge are in Section 3.1 and what research questions follow from that in Section 3.2.

3.1. Gaps In Current Knowledge

As the method presented by Wiewel et al. [24] needs traditional CFD updates to keep the error bounded and a small fluid state history for startup, it does not directly address such problems. An alternate approach, depicted by Figure 3.1, might be able to replace certain regions of the fluid domain complex behaviour is present, which is well known, not of particular interest and high fidelity CFD is too expensive, models based on DL might be able to aid the current CFD models. If such a method could be possible, this could also eliminate the problem of needing a short time history of the fluid state, as a traditional CFD model could introduce the disturbances through the shared boundaries. Furthermore, in this situation the CFD solution on the boundaries might have an bounding effect on the error as well, eliminating the need for traditional update once every few time steps. An overlap area is therefore proposed in which the CFD model makes predictions. The DL model uses the overlap area as well as DL prediction area as an input to make a prediction for the next time step in the DL prediction region.

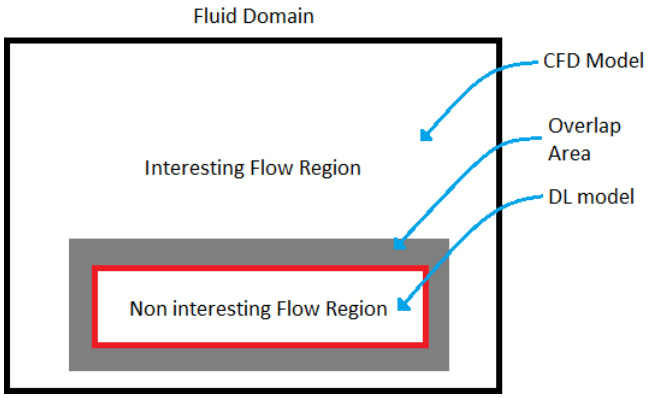

Figure 3.1: Fluid domain containing an interesting flow region and a non interesting flow region.

When developing DL models for an application as described above, it is important to realise that their accuracy depends on the random initial weights and the training process. Correspondingly, the accuracy of the models trained using the approaches by Wiewel et al. [24] could be influenced by the random initial training state, resulting in an unfair comparison. Before continuing with the use of such techniques their mean and standard deviation should be evaluated as it could give an indication whether the comparisons are fair.

Furthermore, generating training data for the training of the DL models is expensive as this needs to be done by the traditional CFD approaches. An analysis of the amount of training data needed for an acceptable accuracy is therefore necessary.

3.2. research Questions

This thesis work consists of three main research questions. The first research question is:

> *"Can an AE-RNN flow model be implemented which does not need any traditional updates and keep the long term prediction error bounded?"*

Sub-questions that follow from first main question are:

1. For a case where the flow structures move through the domain and exit the domain within a relative short time window, can the error be kept bounded without the need for any traditional algorithms?
2. How can the overlap area be included in the latent space by means of altering the encoder setup and execution scheme?
3. What should be the size of the overlap area w.r.t. the prediction area?

The second research question is:

> *"Considering the stochastic nature of DL, can the claimed conclusions in Wiewel et al. [24] comparing different model setups in terms of accuracy be justified?"*

Sub-questions that follow from second main question are:

1. Considering the case where 20 AE's are trained, what is the mean and the standard deviation of the reconstruction MSE for the test set?
2. Considering the case where 20 AE's with each one RNN are trained, what is the mean and the standard deviation of the long term reconstructed prediction MSE on the test set?
3. Considering the case where 1 AE with on 20 RNN's are trained, what are the mean and standard deviation of the long term reconstructed prediction error on the test set?

The third research question is:

> *"What is the sensitivity of the AE-RNN flow model w.r.t. the number of training samples?"*

- What is the sensitivity of the AE reconstruction MSE w.r.t. the number of AE training samples?
- What is the sensitivity of the AE-RNN reconstructed prediction MSE w.r.t. AE training samples?
- What is the sensitivity of the AE-RNN reconstructed prediction MSE w.r.t. RNN training samples?

4

Method

This chapter describes the method that was used during this thesis. The model problem used for training the ANN is described in Section 4.1. The ANN Setup is discussed in Section 4.2. The data sets used for training, validating and testing are described in Section 4.3. The training process is explained in Section 4.4. Finally, details about the inference of the AE-RNN flow model is further explained in Section 4.5.

4.1. Model Problem

A model problem was proposed from which sequences can be obtained in a relative short time window, but contain relatively complex fluid behaviours. Furthermore, the chosen data was one dimensional as training neural networks on higher dimensional data sets requires significantly higher training times. For this an Euler code for Internal Aeroacoustics (EIA) Hulshoff [11] was used, which is able to predict fluid behaviours based on the Euler equations. Here we focused on the one dimensional wave problem in which two waves enter the domain. One wave enters at the left boundary and the other at the right boundary. Both waves propagate through the domain in opposite direction, intersect with each other and leave the domain at the boundary opposing the boundary at which it entered. The waves were specified using their amplitude A, frequency f and the time delay at which they entered the domain τ. Non-linear behaviour is captured by the Euler equations and thus shocks can form when the amplitudes of the waves are large enough. The flow domain was discretised in 1024 points along the x axis. EIA then provides the density ρ, velocity multiplied by the density ρu and the internal energy multiplied by the density ρE. Only $\frac{1}{4}^{th}$ of these points were saved to reduce the size of the sequences. This resulted in 256 points along the x axis which contain the 3 flow properties. All 3 properties were normalised by finding the maximum and minimum in all generated sequences. An example for this is shown in Figure 4.1.

4.2. Artificial Neural Network Setup

For the setup of the ANN inspiration was taken from Wiewel et al. [24]. First the encoder part of an AE was used to reduce the dimensions of the problem, called the latent space. An RNN was given a few previous input time steps latent spaces and from that produced a new time step latent space. This latent space could now be reconstructed to the original dimensions using the decoder part of the autoencoder. This resulted in a prediction of the fluid state in the future. This method will be referred to as the AE-RNN flow model. The AE details will be discussed in Section 4.2.1. Finally, the RNN details will be discussed in Section 4.2.2.

4.2.1. Autoencoder

An autoencoder was used to reduce the dimensions of the flow to an array of floating point numbers with length m_s, called the latent space, and then to transform that back to the original dimensions. The part of the autoencoder that reduces the dimensions to the latent space is called the encoder. The part which reconstructs the original dimensions using the latent space as input is called the decoder. The length of the latent m_s space can be set to anything desired. However, the latent space should be able to contain enough information, so the decoder is able to make an accurate reconstruction. The flow diagram found in Figure 4.2 shows the setup of the AE used during training. The autoencoder is trained to output the exact same fluid

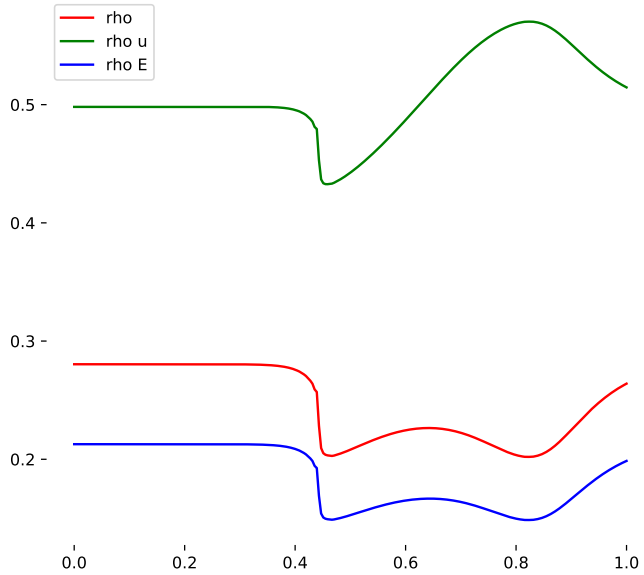

Figure 4.1: 1D Flow model example which was used during the thesis.

state it receives as an input. The encoder and decoder therefore learn to work together in finding an optimal latent space which contains as much information as possible for an as accurate as possible reconstruction.

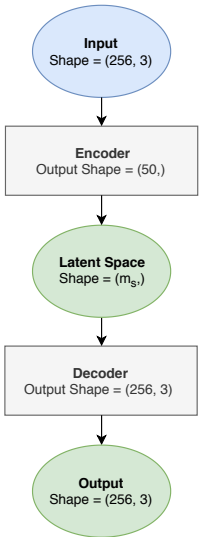

Figure 4.2: Autoencoder setup used during training.

Encoder Block

The encoder block is described by the flow diagram on the left in Figure 4.3. Three layers of convolution blocks with max pooling were used after which the output is flattened. Two dense blocks were applied after which a dense layer with m_s nodes and sigmoid activation was used. The sigmoid activation was used so that the variables in the latent space are forced to be between -1 and 1, to avoid large values in the latent space, which could lead to instabilities during training. The convolution/pooling and dense blocks are explained in Section 4.2.1 and 4.2.1, respectively.

Decoder Block

The decoder block is described by the flow diagram on the right in Figure 4.3. The decoder used a very similar setup as the encoder but instead of reducing dimensions, it increased the dimensions back to the original vector size. It started from the latent space vector and used some dense blocks to increase the dimensions. Next,

the vector was reshaped to a 2D shape, which in turn was used by 3 convolution and upscaling blocks. The last layer was a 1D convolution with 3 filters to ensure the (256, 3) output shape. The convolution/upscaling block is explained in Section 4.2.1.

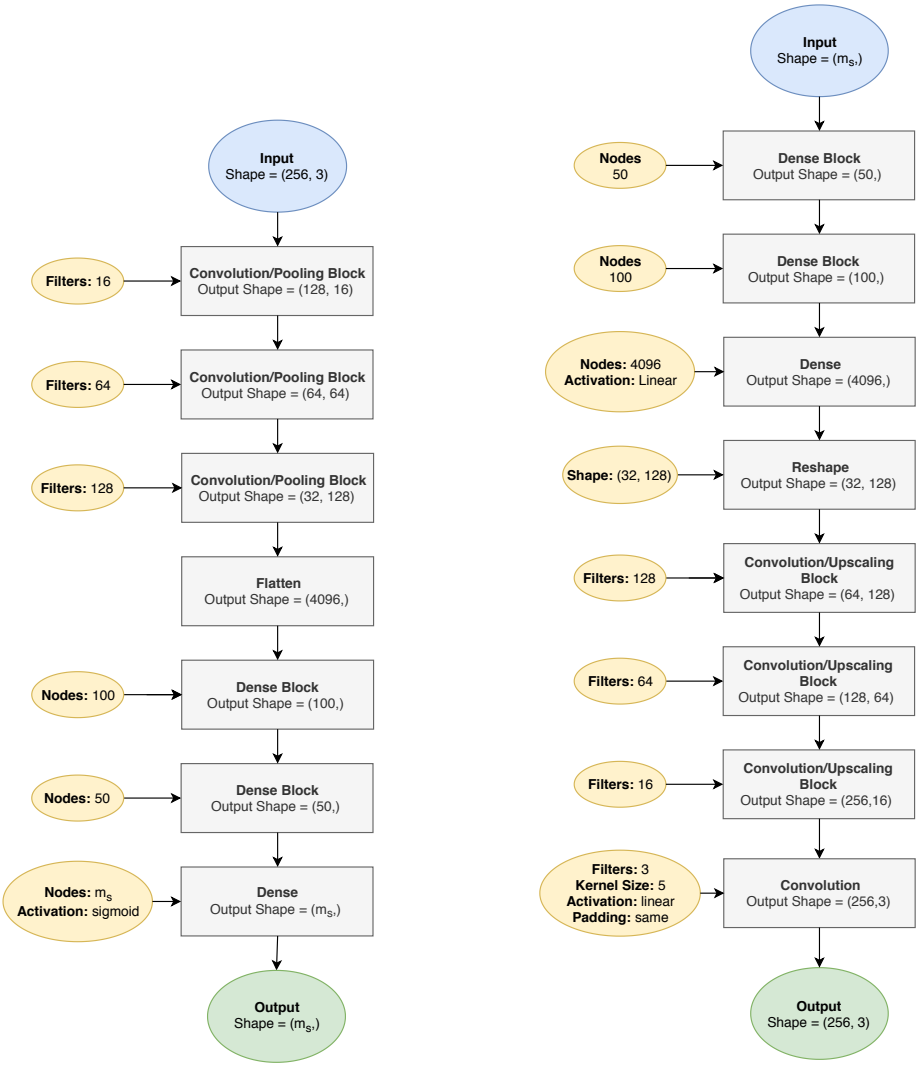

Figure 4.3: Left, Flow diagram of the encoder architecture. Right, Flow diagram of the decoder architecture.

Convolution/Pooling Block

The convolution block/pooling block is described by the flow diagram on the left of Figure 4.4. It used two different inputs, a 2D shape (n_x, n_y) and the number of filters n_f. The inception block Szegedy et al. [23] used these two inputs to create an output with shape (n_x, n_f). This was used as an input together with a squeeze ratio for the Squeeze-and-Excitation block Hu et al. [10], which returned the exact same shape. The original input was added to the output of the Squeeze-and-Excitation block, as suggested by He et al. [7]. Finally, to reduce dimensionality, a 1D Max Pooling layer was used which reduces the x dimensions by a half. This resulted in an output with a shape of $(\frac{n_x}{2}, n_f)$.

Convolution/Upscaling Block

The convolution/upscaling block can be seen on the right of Figure 4.4. It had a similar layout to the convolution/max pooling block, but instead of 1D max pooling layer at the end, an upscaling layer was used at start.

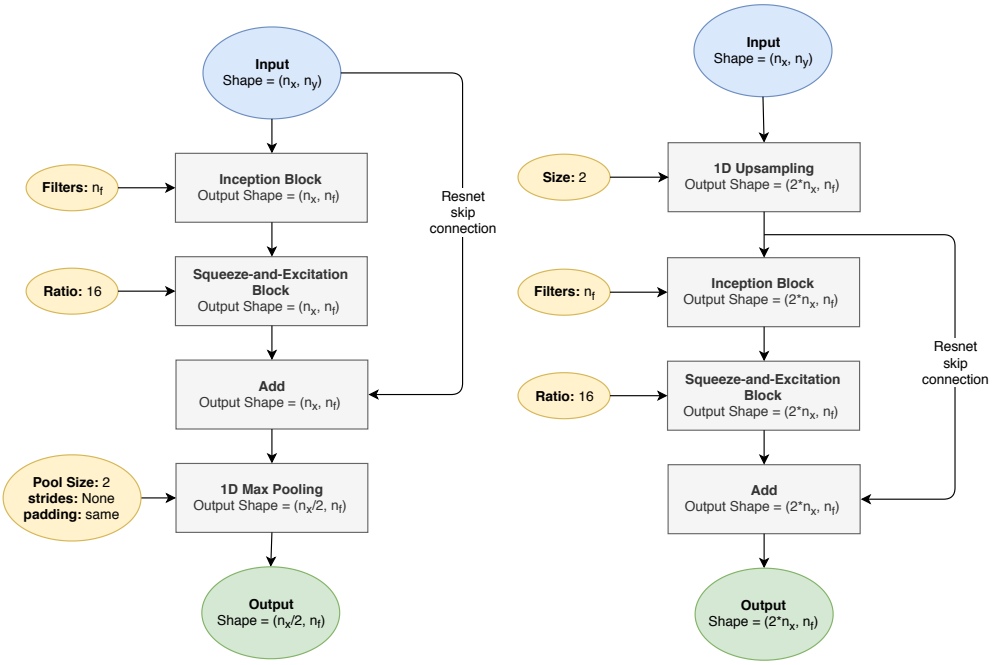

Figure 4.4: Left, flow diagram of the Convolution/Pooling Block. Right, flow diagram of the Convolution/Upscaling Block.

Inception Block

The flow diagram of the inception block Szegedy et al. [23] can be found in Figure 4.5. The diagram looks rather complicated but in fact its design is quite simple. As convolution layers have many hyper parameters, the search for an optimal design can be laboursome. The researchers from Szegedy et al. [23] had the idea to make a single layer in which the ANN can learn which hyperparameters are best suited for the task at hand. The total amount of required filters was therefore split up in 4. Each of the four branches used a different variant of convolutions or max pooling layers. The first 3 branches first squeezed the input as a direct convolution without squeezing requires more computation time. After the squeeze, a convolution with a kernel size of 3, 5 and 7 was used for the three branches, respectively. The last branch used a max pooling layer first, after which a convolution layer with a kernel size of 1 was used. From each of the four branches the outputs were concatenated resulting in the output with a shape of (n_x, n_f). Finally, a batch normalization was used as is suggested by Ioffe and Szegedy [12].

Dense Block

The dense block is shown on the left of Figure 4.6. This is a relative short block as only a dense layer was used after which a batch normalization layer was applied, as suggested by Ioffe and Szegedy [12].

Squeeze-and-Excitation Block

The Squeeze-and-Excitation block Hu et al. [10] is shown on the right in Figure 4.6. First a 1D global average pooling layer was used which finds the average value along the first dimension. This resulted in a one dimensional shape $(n_y,)$. A dense layer was then applied which squeezed the dimensionality for a reduction in computation costs (the ratio defines the squeeze amount). A second dense layer was used to get the original dimensions again. Finally, the input was multiplied by the the output of the second dense layer. This Squeeze-and-Excitation block allowed the network to amplify the features it determined to be most important.

4.2.2. Recurrent Neural Network

Figure 4.7 shows the flow diagram of the RNN. The input for this network was a few time steps o of latent space arrays with length m_s. During the thesis o was set to 5 at all times. An LSTM Hochreiter and Schmidhuber [8] layer was used as it is able to remember the previous inputs it processed and therefore process the sequence of latent space arrays. It only outputs the last state after processing each of the input latent spaces and therefore

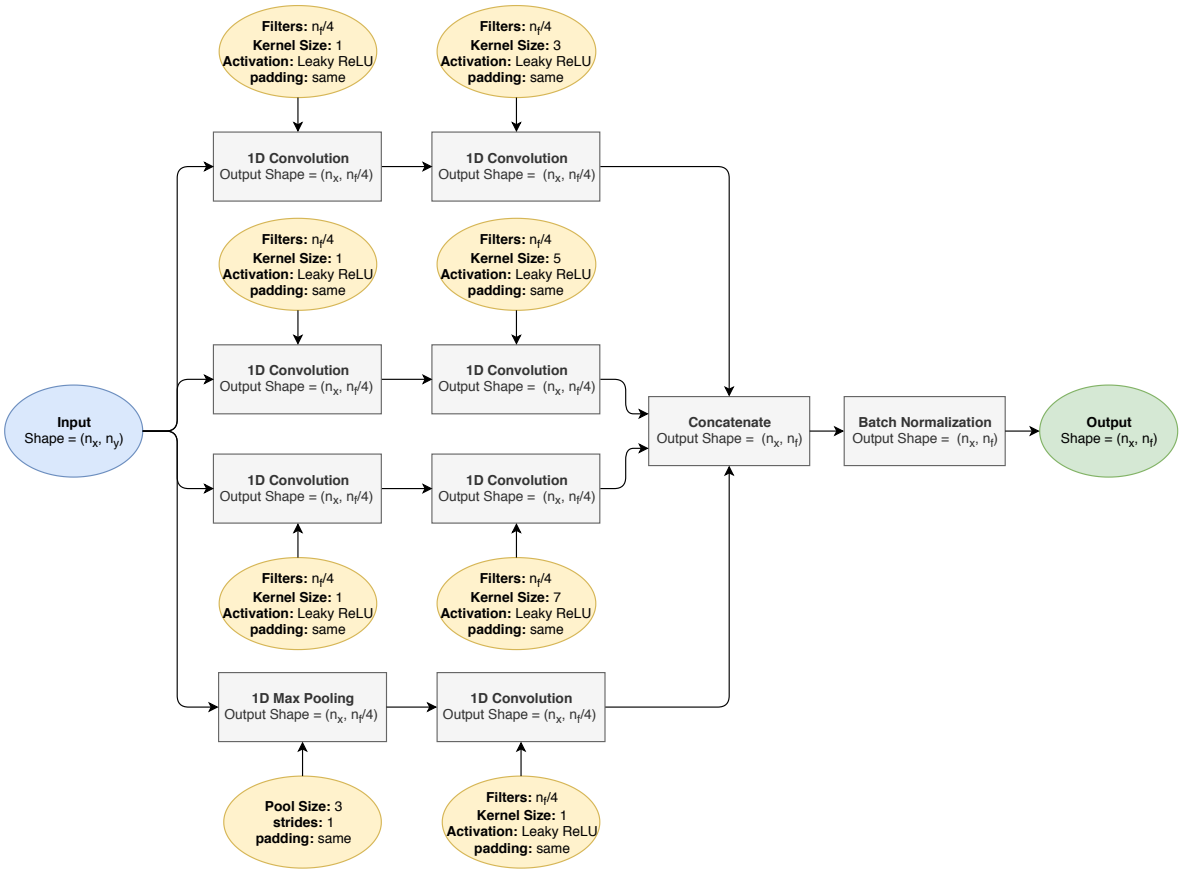

Figure 4.5: The flow diagram of the Inception Block.

outputs a 1D vector. Next, A dropout layer was used to prevent overfitting. A repeat vector was used to transform the output back to a 2D shape again. The amount of repeats was 1 as only one time step is desired to be predicted in advance. This output was processed by another LSTM. This time the state at each iteration is outputted. As it was only one iteration for this case it returns a 2D shape with the length on first dimensions being one. Again a dropout layer was used to prevent overfitting. Finally, a convolution layer was used to return the shape back to $(1, m_s)$, which is a prediction for the next time step in latent dimensions. Note that only one time step predictions were evaluated as Wiewel et al. [24] points out that the one time step predictions worked best for their case.

4.3. Data Sets

To train, validate and evaluate the ANN, three different data sets were used:

- A training data set, which was used during training only and referred to as the training set. The AE and RNN have a separate training set as they have to perform another task. This set was generated from a general sequences data set.

- A validation data set, which was used during training to evaluate how well the ANN's were doing on a set which it isn't trained on. This is referred to as the validation set. The AE and RNN had a separate validation set as they had to perform another task. This set was also generated from a general sequences data set.

- An evaluation set of sequences, which was used at inference to evaluate how well the full AE-RNN flow model is doing on long term predictions. This sequences were generated randomly from EIA so it were completely new sequences which the AE-RNN flow model never had seen before.

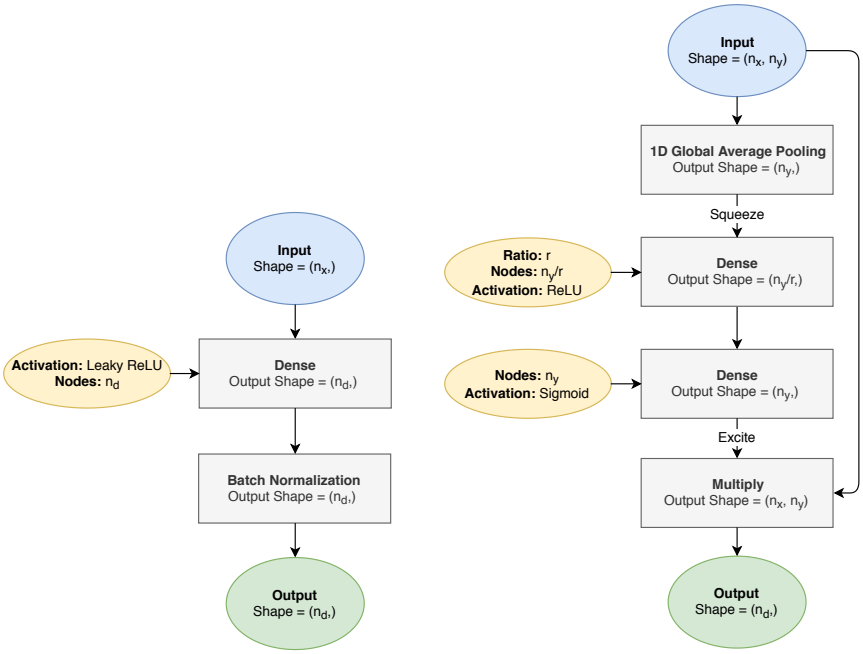

Figure 4.6: Left, flow diagram of the Dense Block. Right, flow diagram of the Squeeze-and-Excitation Block.

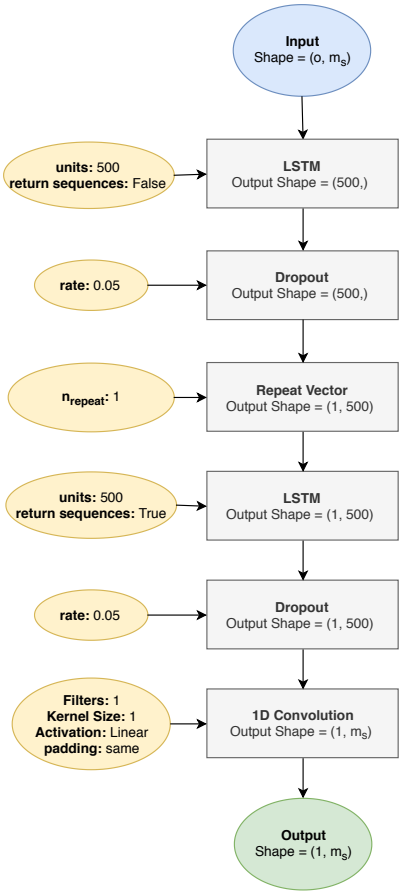

Figure 4.7: Flow diagram of the RNN.

General sequences data set
To generate both the AE as well as the RNN training and validation data, first EIA was used to generate 41000 sequences of intersecting waves. These sequences were generated by using uniformly chosen random parameters for the A, f and τ for the waves as described by table 4.1. In EIA the simulation time step was chosen to be $2.5 \cdot 10^{-4}$ seconds. The sequence was saved every $2.5 \cdot 10^{-2}$ seconds until the 2 seconds time limit was reached. This resulted in sequences with a length of 80 time steps. This data set was sufficiently varying and therefore contains enough cases for the ANN to generalise to all cases within these bounds.

	Minimum	Maximum
a	-0.5	0.5
f	20	200
τ	0.5	1

Table 4.1: Bounds for the random variables for each of the properties of the waves.

Training and validation data sets
The general sequences data set was used to generate both the training as well as the validation set. As the training set was desired to be as diverse as possible, training samples were chosen as far from each other as possible. Meaning that if a certain sequence at a certain time step is chosen as a training sample, a time step close to this time step should not be sample, as it is similar to the other. Therefore, depending on the number of desired training samples, the training samples were selected as far as possible from each other as possible, with random intervals between them. This also eliminated duplicate training samples as each samples was considered only once.

For the AE the training samples were selected as described above. Each training samples had a shape of (256,3), which was the fluid state at a certain time step. The training input was exactly the same as the output, as the AE should be trained to reconstruct the input. This set was finally split up in the training and the validation set.

The selection of training samples for the RNN was slightly different to that of the AE. The number input time steps for the RNN (o) was chosen to be 5, which means that the total length of the training sets should be 6, containing 5 inputs time steps and 1 desired output time step. The selection process described above still holds, but the selected time step was now the first input for the network. The 5 other time steps that come after this time step were also collected. This resulted in input training samples of shape $(5,256,3)$ and output training samples of shape $(1,256,3)$. However, the RNN should be trained on the latent spaces of these samples generated by a trained AE. Therefore all these samples were encoded by the encoder part of a trained AE. This resulted in input training samples with shape $(5, m_s)$ and output training samples with $(1, m_s)$. Finally, these were also split up in a training and validation set. Note that the RNN was trained to predict one time step in advance given a completely correct input sequence.

Evaluation data set
As the AE-RNN flow model was used to predict a complete sequence of a flow, a completely new data set of 5000 sequences was generated, using the same random intervals for the wave properties shown by 4.1. For this set, however, the only difference is that maximum simulation time was set to 4, to be sure that all waves structures have left the domain.

4.4. Training

The flow diagram for the training process can be seen in Figure 4.8. The training process of the AE-RNN flow model was done in two phases. The AE training phase (top flow diagram) and the RNN training phase (bottom flow diagram). Both used a training and validation data set. The training set is only used for optimising the weights in the ANN. During each training epoch samples were shown once to the ANN. The training process usually has more than one epoch as the networks can become more accurate after the training samples have been shown multiple times. After each epoch the ANN was validated, which means that it is shown the validation samples. The weights of the ANN were not optimised during this phase, but only the accuracy was determined. If the best validation accuracy was obtained, the ANN weights were saved. This ensures that the best ANN was saved and not the last, which could possibly perform worse than the ANN during a previous epoch. Due to instabilities in the training process, the accuracy might suddenly rise rapidly. If this happened the best saved state of the network was loaded and the training process continued.

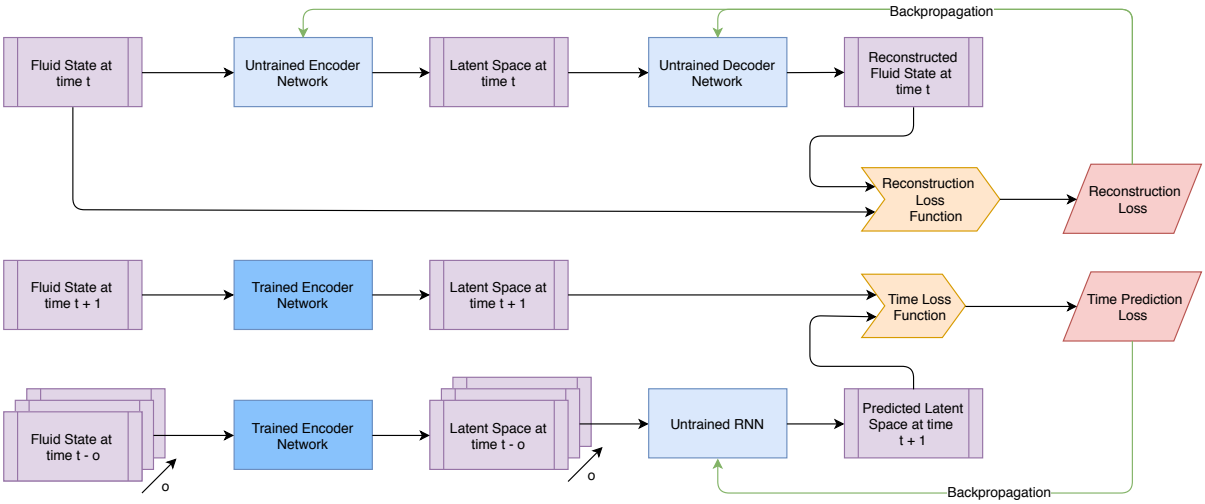

Figure 4.8: Flow diagram for training the AE-RNN flow model.

AE training phase
Using the generated AE training and validation data sets as described in Section 4.3, the AE was trained. A the fluid state of certain time step of a certain sequence is shown to the AE one by one. It was encoded by the encoder to the latent space and then decoded back to the original dimensions using the decoder. Its output was expected to be exactly the same as the input but, especially at the begin of the training process, was not. An Mean squared error (MSE) function was used between the input and output, which results in the reconstruction loss. This allows the AE to be optimised to reconstruct the input given a low dimensional latent space.

RNN training phase
Using the generated RNN training and validation data as described in Section 4.3, the RNN was trained. The RNN was given subsequent time steps encoded to the latent space as an input. The expected output is the next subsequent time step as an output (i.e. a temporal prediction of the latent space). An MSE loss function was used between the actual output and the expected output, which resulted in the time prediction loss. This allows for optimising the RNN weights for predicting the next time step in latent dimensions.

4.5. Inference
The flow diagram for the inference phase can be found in Figure 4.9. The initial 5 time steps were given as an input and encoded to the latent space by the encoder part of the AE. The RNN made a prediction for the latent space in next time step. This latent prediction was decoded by the decoder part of the AE. The latent prediction was also used to as a new input to predict a new next time step. The oldest latent time step was removed and the newest predicted latent time step was added to the inputs for the RNN. This loop could continue as long as desired.

4.6. Boundary Overlap Areas
As described in Chapter 3.1 and by Wiewel et al. [24], the solution for the AE-RNN flow model diverges from the ground truth. A strategy to overcome this was to add boundary overlap areas to the prediction region of the AE-RNN flow model. An example is given in Figure 4.10. The AE-RNN flow model used this overlap area to make predictions of the future state in the DL prediction area. The traditional algorithm (EIA in this case) used the DL prediction area as well as the other flow domain which is not included in the plot to make its prediction of the future state of the overlap areas. The hope was that the error remains bounded due to these overlap areas. The overlap areas required the encoder to be different from the original encoder depicted in Figure 4.3. Figure 4.11 shows the encoder in which each area has its own sub-encoder. The input was split in the overlap regions and the prediction region and fed to the individual encoders. During this thesis the overlap area size was $\frac{1}{8}^{th}$ of the total environment area and thus the overlap areas had a shape of $(32,3)$ and the prediction area had a shape of $(196,3)$.

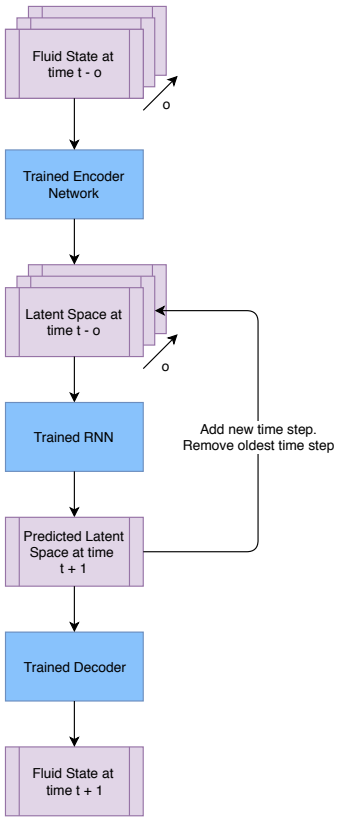

Figure 4.9: Inference of the AE-RNN flow model.

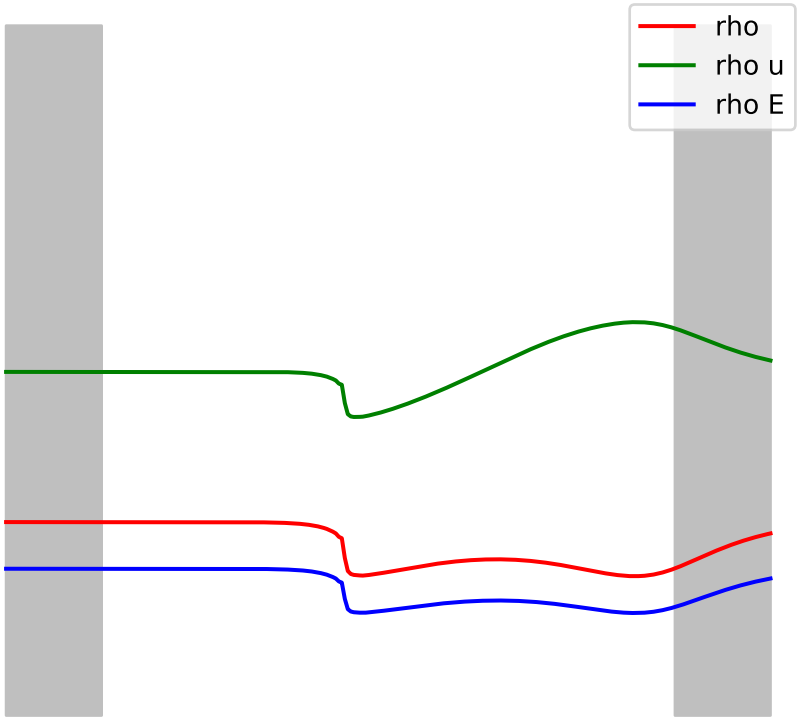

Figure 4.10: Example of overlap areas, which are shown in grey. The prediction area is shown in white.

Figure 4.11: Flow diagram of the encoder which supports boundary overlap areas.

Figure 4.12 shows the layout of the sub-encoders. The layout of the sub-encoders were very similar to the original encoder structure depicted in Figure 4.3. However, the input and output shapes are different and therefore each layer had different output shapes.

Furthermore, the execution scheme was also modified to enforce the EIA solution. Figure 4.13 shows this execution scheme. The fluid state history (which could be a steady flow for this case, as disturbances could be introduced to the prediction domain via the overlap boundaries) was first split up in the left and right overlap areas as well as the prediction area. These were then encoded by their own sub-encoder. All these were concatenated to a latent space with length m_s. A prediction area latent prediction was produced with the RNN. This means that the RNN did not produce a prediction for the overlap areas. This output was concatenated with the latent spaces for the overlap areas and subsequently decoded. With the new fluid state EIA could now make a prediction for the overlap area, which in turn could again be encoded by the two overlap sub-encoders. The prediction area latent prediction was also fed back to the prediction area latent spaces. All latent spaces removed their oldest time step and added the newest time step generated by last iteration. Next, the latent spaces were again concatenated from which the RNN could again make a new prediction. This process could keep iterating as long as desired, which allowed for long term predictions.

4.7. Stochatstics

The stochastic properties of the AE-RNN flow model with boundary overlap areas were assessed. This was done in 2 ways:

- 20 identical AE were trained for 300 epochs using random initial weights and with the exact same training set of 30000 samples and a validation set of 20000 samples. On each of those AE, 1 RNN was trained for 600 epochs with a training set of 50000 samples and a validation set of 20000 samples.
- 1 AE was trained for 300 epochs with a training set of 30000 samples and a validation set of 20000 samples. On that AE, 20 identical RNN were trained for 600 epochs with random initial weights and with the exact same training set of 50000 samples and a validation set of 20000 samples.

The AE-RNN combinations were tested on the evaluation set described in Section 4.3. This gives an indication by how much the accuracy of the AE-RNN flow model was influenced by random initial weights. The first case primarily focused on the whole AE-RNN flow model stochastic behaviour, where each AE and RNN were different trained models. The second case keeps the same trained AE and therefore focused on the

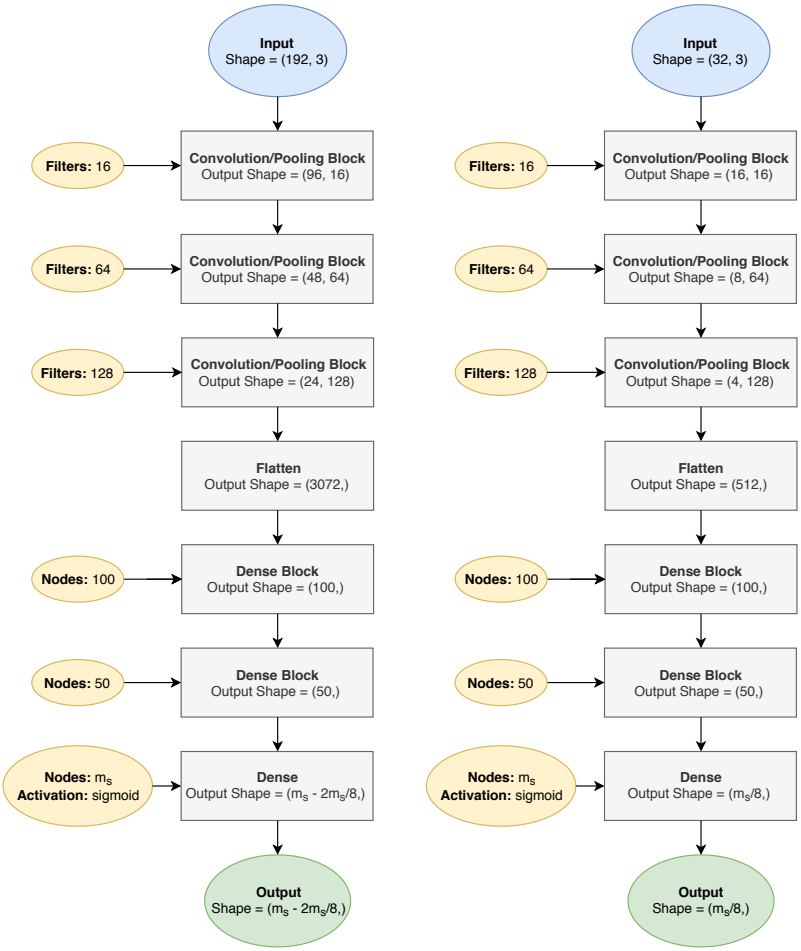

Figure 4.12: Left, flow diagram of the prediction area sub-encoder. Right, flow diagram of the overlap area sub-encoder.

statistics in the trained RNN's on the same AE. Note that a latent space size m_s of 35 was used for all of these cases.

4.8. Number of training samples

The effect of the number of training samples was assessed. This was done two ways:

- An AE was trained on a varying number training samples. The number of training samples that will be evaluated are 70.000, 50.000, 30.000, 10.000 and 5000. A single RNN per AE was trained on this with a constant number of training samples of 50.000.
- An AE was trained on a training set of 30.000 training samples. A few RNN were trained using this AE with a verying number of training samples. The number of training samples that were evaluated were 100.000, 50.000, 30.000, 10.000 and 5000.

With these trained, the accuracy was evaluated on evaluation set. Note that a latent space size m_s of 35 was used for all of these cases.

4.9. Loss definitions

This is a list of loss definition that will be used in the conclusions.

- AE training MSE - The reconstruction MSE of the AE on the training data during a training epoch.
- AE validation MSE - The reconstruction MSE of the AE on the validation data after each epoch.

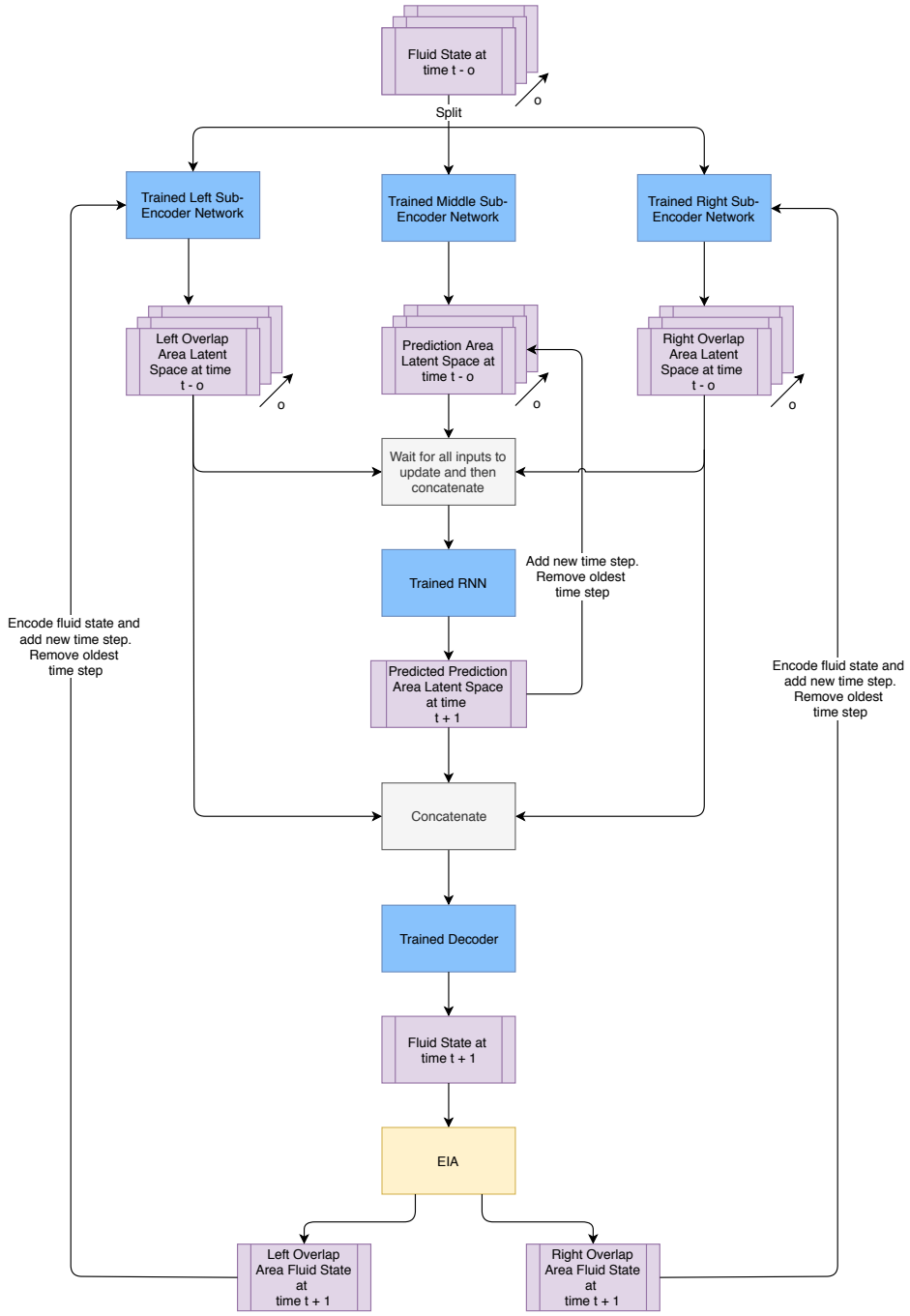

Figure 4.13: Execution scheme where overlap areas are enforced by EIA.

- AE Test MSE - The reconstruction MSE of the AE on the evaluation set.
- Minimum AE Validation MSE - The minimum reconstruction MSE of the AE on the validation set occurred during all epochs.
- RNN training MSE - The latent prediction MSE of the RNN on the training data during a training epoch.
- RNN validation MSE - The latent prediction MSE of the RNN on the validation data after each training epoch.
- RNN Test MSE - The MSE of the RNN on the long term latent space predictions on the test set.

- Minimum RNN validation MSE - The minimum latent space prediction MSE of the RNN on the validation set occurred during all epochs.
- AE-RNN Test MSE - The long term prediction MSE by the AE-RNN flow model of the reconstructed latent space on the evaluation set.

4.10. Deep Learning Tools

To create the DL models keras 2.1.6 Chollet et al. [4] was used with the tensorflow-gpu 1.12.0 Abadi et al. [1] as a backend. The GPU that was used to train the models was the Nvidia Geforce GTX 1070.

5

Results and Discussion

This section presents the results and discusses them. In this case m_s was always chosen to be 128 as this gave the best results in a test run. The boundary overlap areas are discussed in Section 5.1. The error development is discussed in Section 5.2. The influence of the stochastic nature of the models is discussed in Section 5.3. The effect of the number of training samples is discussed in Section 5.4.

5.1. Boundary overlap areas

Figure 5.1 shows multiple examples of two waves, with random amplitudes and frequencies, entering the domain, intersecting and exiting. Each plot has the normalised ρ, ρu and ρE plotted in red, green and blue, respectively. Each column shows a sequence of several predicted time steps. The EIA solution is shown as a solid line, the autoencoded EIA solution is shown as a striped line and the reconstructed prediction is shown as a dotted line. The overlap area used to enforce the solution of EIA on the enter/exit domains are shown in grey. The DL prediction area is shown in white. As can be seen, the method produces almost the same solution as the EIA solution. Interaction between the opposing waves is captured almost perfectly and shock formation is clearly is predicted well.

It should be noted that the overlap area should be large enough for the largest scales occurring in the flow. An example where the overlap area is too small is shown in Figure 5.2. The frequency of the wave entering on the left is low enough to create a wave which is too wide to be encapsulated by the overlap region, causing the AE-RNN based solver to make wrong predictions. As these initial predictions are used to build upon the solution remains faulty for the entire duration the waves are in the prediction domain.

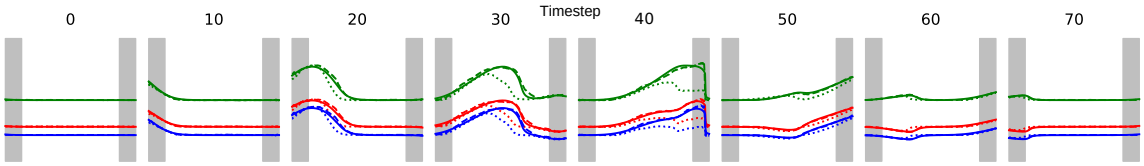

Figure 5.2: Faulty prediction by the AE-RNN flow model, caused by a too wide wave to be encapsulated in the overlap region. A larger overlap region is necessary to get the right prediction.

5.2. Error Development

To evaluate whether the AE-RNN flow model error remains bounded in the 1D environment, a trained AE-RNN was evaluated on the test set of 2000 sequences. The EIA solution was compared with the AE-RNN reconstructed prediction for each time step. Figure 5.3 shows the mean MSE per time step. As can be seen, after a certain time period, the error first quickly rises, which is caused by the waves being imposed in the prediction region. At a certain point the error reaches its peak which at which waves are fully in the prediction domain. After that the error decreases to nearly zero again as the waves move outside the prediction domain. It should be noted that for simplification reasons, the EIA solution is computed and saved for the overlap for all time steps. This means that EIA and the coupling of EIA and the AE-RNN flow model was one way. The

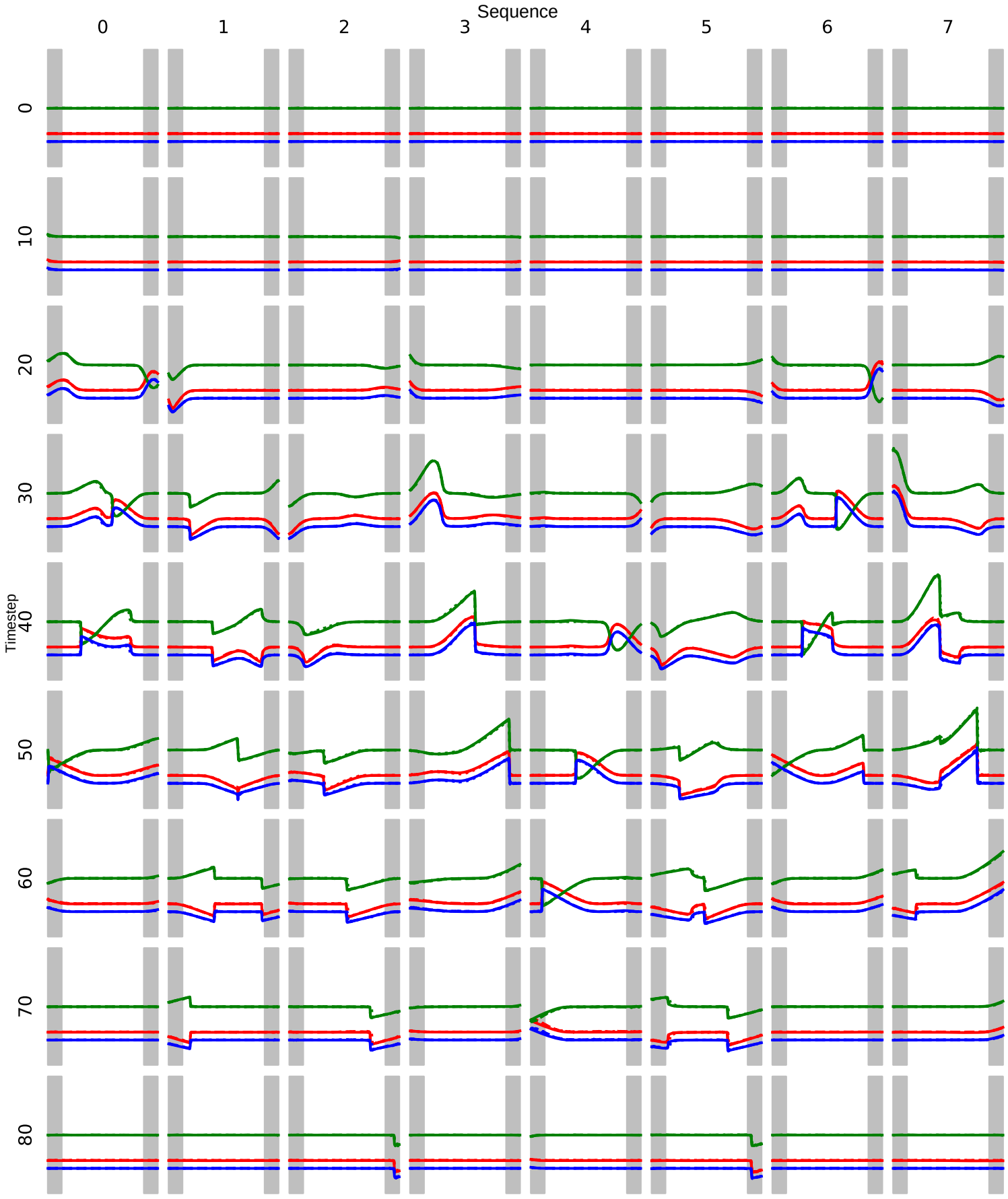

Figure 5.1: Each column is a sequence of two waves, with random frequencies and amplitudes, entering at one of the boundaries, colliding and exiting at the opposing boundaries. The overlap region indicated in grey is enforced by EIA's solution, while the white region is predicted by the AE-RNN flow model. Each figure contains of ρ, plotted in red, ρu, plotted in green and ρE plotted in blue and are all normalised. Solid lines are the EIA solution, striped lines are the autoencoded EIA solution and the dotted line are the reconstructed prediction.

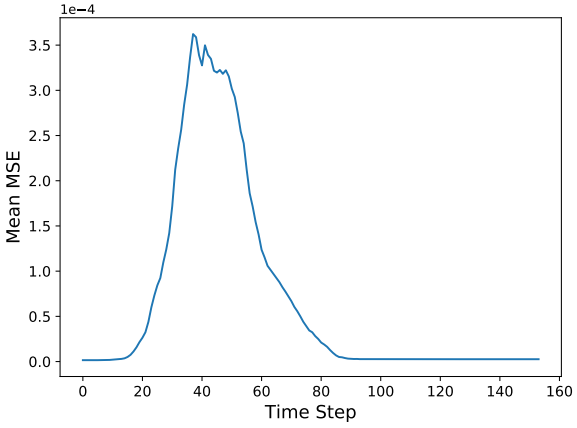

Figure 5.3: Mean reconstructed prediction MSE of 2000 predicted sequences by the AE-RNN flow model.

solution at the overlap areas is therefore always the correct EIA solution and not the solution propagated by the AE-RNN flow model into the overlap domain. Full coupling is anticipated to have similar performance to the one way coupling as the solution in the DL prediction area propagated by the AE-RNN flow model remains very close to the exact counterpart. Another interesting observation is that that if no waves enter the domain the solution remains steady. The errors it makes for these steady solutions apparently do not blow up over time but remain constant.

5.3. Statistics

Figure 5.4 shows the AE validation MSE of the networks after each training epoch and the minimum AE validation MSE encountered during all epochs. It can be noticed the best network is more than twice as accurate on the validation set than the worst performing network, although they all have an absolute error of less then $10e^{-3}$. Table 5.1 shows the mean and standard deviation of these networks evaluated on the test set, on the first row. It can be seen that for the AE's the standard deviation is 14.3 % of the mean.

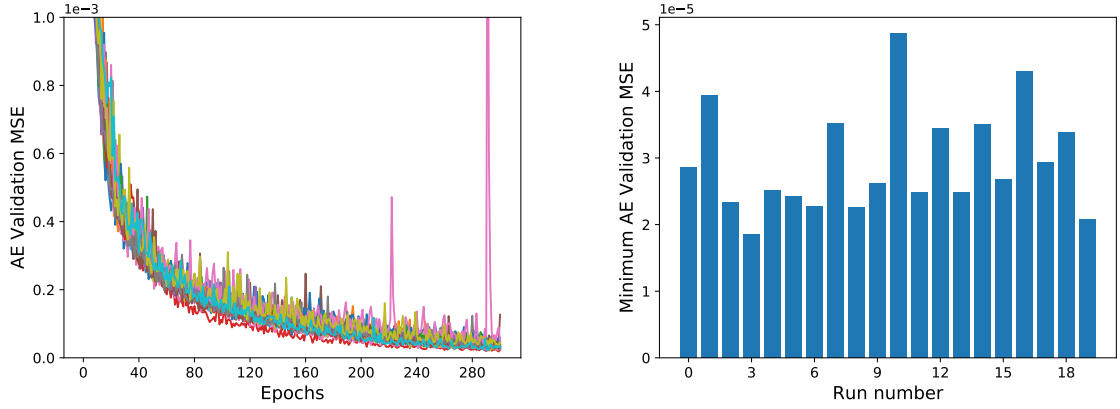

Figure 5.4: Left, AE validation MSE for each of the 20 autoencoders vs epoch number. Right, Minimum AE validation MSE over all epochs, for each of the 20 autoencoders.

When training one RNN to make the time predictions per AE it was expected to see large deviations per training run, as the latent space of each of the autoencoders can be completely different and the magnitude of the latent space variables can also vary in any way. Figure 5.5 shows the validation loss per epoch for each of the 20 RNN's as well as the minimum validation loss per training run. It can be seen that the RNN's perform completely differently per training run on validation set. The RNN's performing worse on the validation set might provide better reconstructed predictions due to the difference in latent space variable magnitudes.

Table 5.1 shows the mean and standard deviation for the RNN's evaluated on the long term test set. It can be seen that the standard deviation is about 50%.

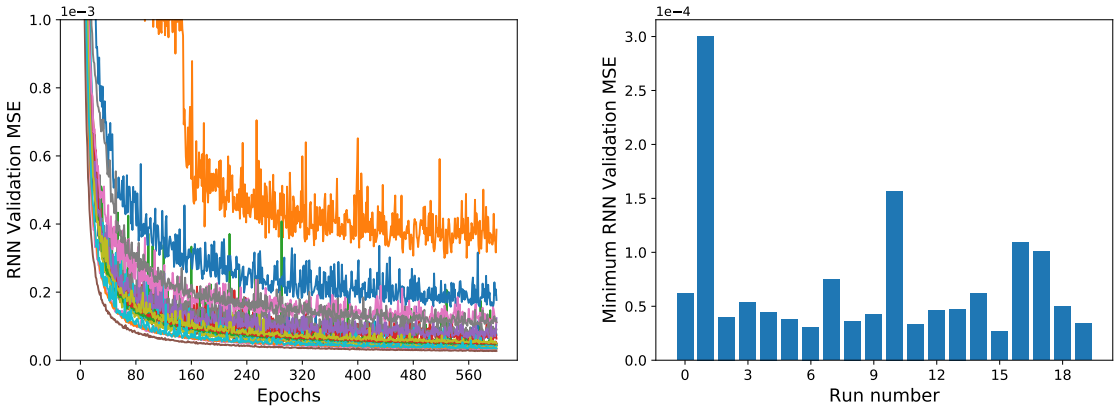

Figure 5.5: Left, RNN Validation MSE for each of the 20 RNN's trained on one of the AE's shown in Figure 5.4 vs epoch number. Right, Minimum RNN Validation MSE for all epochs of each of the RNN's trained per AE.

The mean AE-RNN test MSE over all time steps per trained AE-RNN is plotted in Figure 5.6. It can be seen that the best performs twice as good on the validation set as the worst network. Table 5.1 shows this trend as well for the long term prediction task. The standard deviation is about 15 % of the mean error. It should be noted that the latent spaces are generated by different AE. These all produce latent spaces in which the magnitudes of the variables might differ. This means that the actual reconstructed error might completely different than the error in latent dimensions. In all cases however, the magnitude of the error is relatively low.

It is also interesting to see the stochastic behaviour of training multiple RNN's with the same layout on a single trained AE. 20 RNN's are therefore trained on the same AE to see whether this will result in a more consistent performance. Figure 5.7 shows the validation loss history as well as the lowest validation loss per RNN that was obtained over the whole training process. It can be seen that when using a single AE the RNN's perform more consistently on the validation data than training each RNN on a separate trained AE. Table 5.1 also shows this trend as the standard deviation is only 9% of the mean rather than the 50% observed by training 20 RNN's on different AE.

The mean reconstructed prediction MSE over all time steps per trained network is again plotted in Figure 5.8, to see whether they also perform the similarly in the reconstructed predictions. The figure visually looks similar to the case where 20 different AE's were used. Table 5.1 shows the mean and standard deviation for this case as well. It can be seen the standard deviation is 9% for this case, which is lower than the 20 AE case.

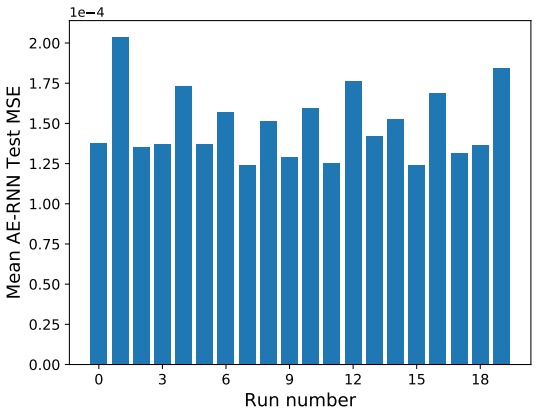

Figure 5.6: AE-RNN long term prediction MSE on the test set.

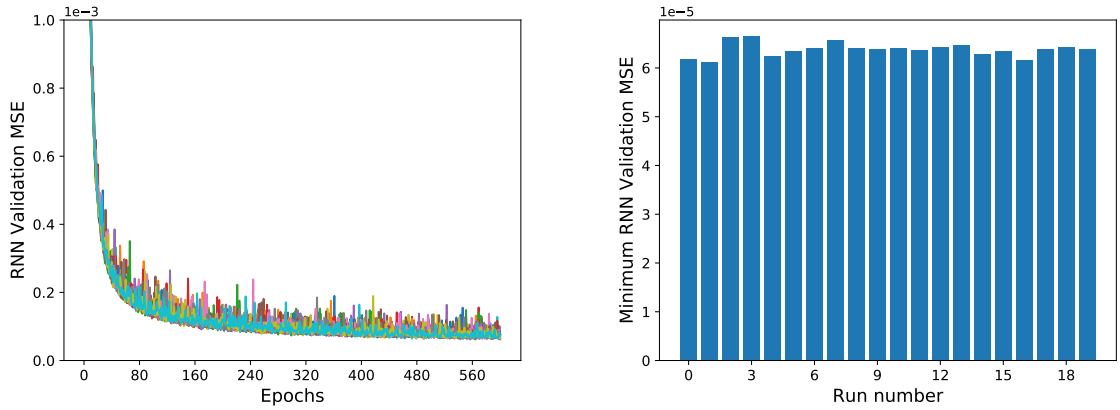

Figure 5.7: Left, Validation loss for each of the 20 RNN's trained on the first of the AE's shown in Figure 5.4 vs epoch number. Right, Minimum validation loss, for each of the 20 RNN's trained the first AE shown in Figure 5.4, occurred during all epochs

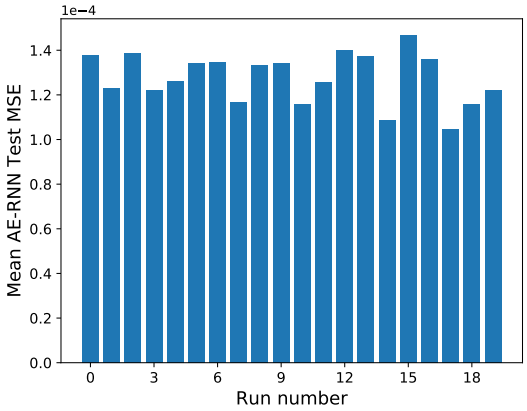

Figure 5.8: AE-RNN long term prediction MSE on the test set.

	Mean MSE	STD (% mean)	Minimum MSE	Maximum MSE
AE Test MSE (20 AE)	$5.26 \cdot 10^{-5}$	14.3%	$4.17 \cdot 10^{-5}$	$6.90 \cdot 10^{-5}$
RNN Test MSE (20 AE, 1 RNN per AE)	$3.65 \cdot 10^{-4}$	51.8%	$1.76 \cdot 10^{-4}$	$1.06 \cdot 10^{-3}$
AE-RNN Test MSE (20 AE, 1 RNN per AE)	$1.49 \cdot 10^{-4}$	14.6%	$1.24 \cdot 10^{-4}$	$2.04 \cdot 10^{-4}$
RNN Test MSE (1 AE, 20 RNN's)	$3.83 \cdot 10^{-4}$	9.03%	$3.12 \cdot 10^{-4}$	$4.82 \cdot 10^{-4}$
AE-RNN Test MSE (1 AE, 20 RNN's)	$1.28 \cdot 10^{-4}$	8.59%	$1.05 \cdot 10^{-4}$	$1.47 \cdot 10^{-4}$

Table 5.1: Stochastic properties of the AE-RNN flow model evaluated on the long term prediction test set.

From this can be concluded that the stochasticity of an AE-RNN combination is high. Two separate trained AE-RNN combinations can have a large accuracy differences (the one could have an accuracy twice the accuracy of the another). Considering these findings, the claims made by Wiewel et al. [24] about the comparison between different AE-RNN setups and methods which differ slightly in accuracy may have been happenstance and not because the one is actually better than the other. To get a better view on the real differences in model accuracy, multiple AE's should be trained, with on each of using multipe RNN's. The mean and standard deviation could give a better indication about the actual accuracy of the model. Furthermore, the models trained by Wiewel et al. [24] could potentially be improved by training multiple AE-RNN combinations. The best performing model could be selected for an optimal performance.

5.4. Training Samples

Figure 5.9 shows the validation loss per epoch as well as the best validation loss obtained per number of samples. As expected it can be seen the number of training samples has a clear impact on the models performance. The more training samples it has, the better it can reconstruct the validation samples.

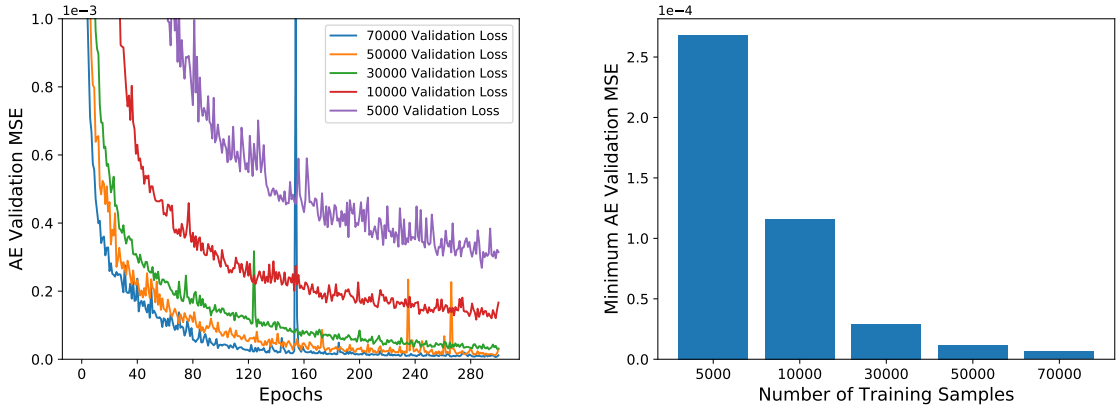

Figure 5.9: Left, AE Validation MSE for each number of training samples settings vs epoch number. Right, Minimum AE validation MSE, for each number of training samples settings, occurred during all epochs

Figure 5.10 shows the AE validation MSE per epoch of the RNN trained with constant samples but on the AE models that with a different number training samples, as well as the minimum AE validation MSE during each run. Interestingly, the number of AE training samples has impact on the AE as well as the RNN even though the number of training samples of the RNN is remained constant. Although a trend can be extracted from these plots, the stochastic nature of this plot shown in Section 5.3 should be taken in mind. This trend could be random.

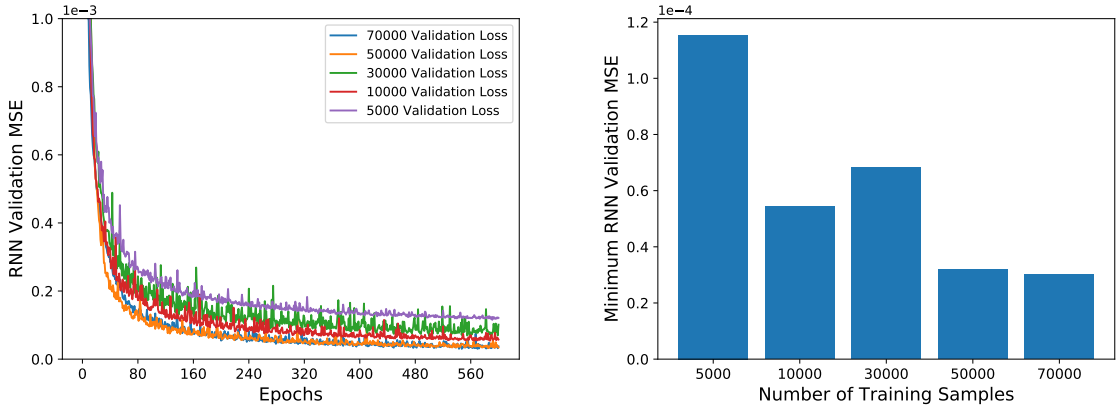

Figure 5.10: Left, RNN Validation loss for each number of AE training samples settings vs epoch number. Right, Minimum RNN validation loss, for each number of AE training samples settings, occurred during all epochs

The reconstructed prediction loss is plotted in Figure 5.11. The number of AE training samples is here also of relative large influence, as the reconstruction accuracy is influenced by the number of training samples.

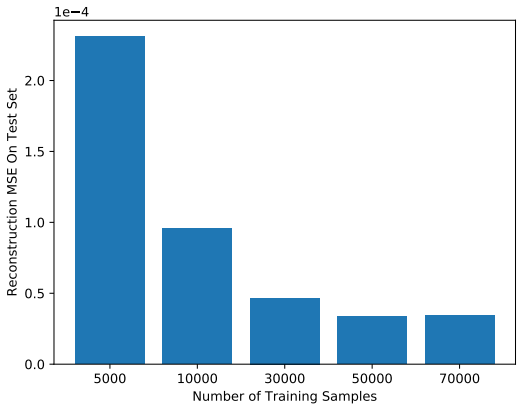

Figure 5.11: Reconstructed long term prediction MSE over 5000 sequences with 153 time steps vs the number of AE training samples

A single AE is now trained with constant training samples and on this different RNN's are trained with a varying number of training samples. Figure 5.12 shows the minimum RNN validation MSE achieved during the whole training process. This also has the expected downward trend if more samples are fed during training.

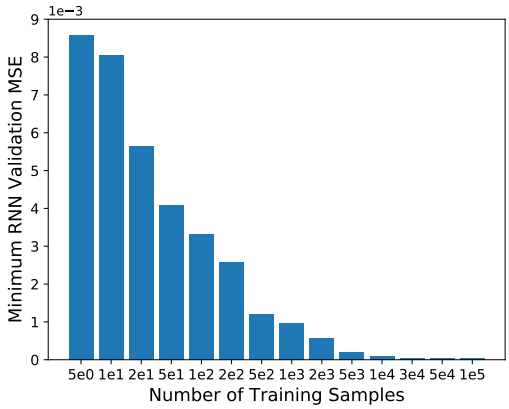

Figure 5.12: Minimum RNN validation loss, for each number of RNN training samples settings, occurred during all epochs

The AE-RNN Test MSE for each of the RNN's shown is shown in Figure 5.13. Interestingly the number of training samples doesn't seem to have much effect on the long term prediction accuracy of the network if more than 5000 training samples are used, even though the validation accuracies for these networks show large differences.

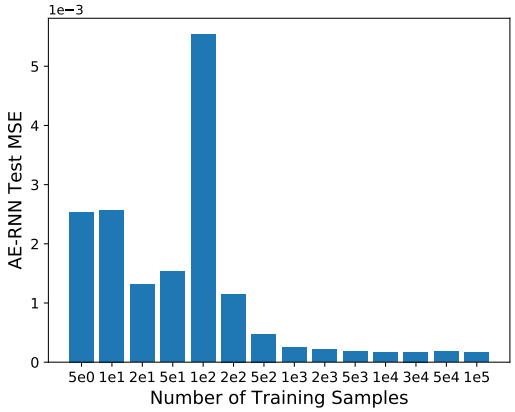

Figure 5.13: Reconstructed long term prediction MSE over 5000 sequences with 153 time steps vs the number of AE training samples

6

Conclusion

First of all it can be concluded that boundary overlap areas can be appears to be an effective way to introduce new flow data into the method presented by Wiewel et al. [24] when using the presented algorithm. The boundary overlap areas do keep the error bounded for the case where fluid structures pass though the domain relatively quickly. The training routine can remain the same but the layout of the encoder part of the AE should be modified. The size of the overlap area should be big enough to contain the largest scales flowing into the DL subdomain. This has potential for the 3D fluid cases as the traditional steps CFD could potentially be avoided in the prediction domain, leading to significant speed ups for the DL prediction area. This is desired for problems where parts of the domain are well known but are necessary to solve the full domain.

The stochastic nature of DL does have an impact on the accuracy of the network. The accuracy for each of the trained network was in the order of 10^{-3}, which is barely noticeable for the naked eye. When training the same setup for 20 times the maximum standard deviation in accuracy was found to be 14%. To do a fair comparison, ideally the models should be trained multiple times to see what the mean and standard deviation of the model are. These should be compared by means of a T-test, for example, to find the probability that two different setups actually differ in accuracy. If the accuracy improves by a much larger margin standard deviation of 14% this is of course less of a necessity. Methods presented by Wiewel et al. [24] should ideally be rerun for several times to get a clearer picture of the underlying statistics. From that a fair comparison between the suggested methods could be made.

It was found that the training data should primarily be generated for training the AE. The AE found a maximum accuracy when trained on 50.000 training samples. The RNN on the other finds a maximum accuracy when trained with 5000 samples only. It should be noted that when generating training samples for the AE, the samples can be reused for the RNN.

7

Recommendations

The AE-RNN flow model with the added overlap areas works well for the 1D case but it is interesting to see how this scales to 2D and 3D problems. How do the minimum required number of training samples scale in 2D and 3D? It is also interesting to see if the current (relatively simple) ANN setup would perform in a 2D case.

It is also recommended to evaluate how well the model performs if flow structures enter the domain constantly, rather than one wave at a time. Can the model make accurate prediction if a waves enters via the overlap domain while another one exits via that overlap domain at the same time?

Furthermore, a sensitivity analysis should be performed to see how well the model performs using other AE and RNN architectures and if it can be optimised further. This could best be done for the 1D case as training times are relatively short for the 1D models.

It would also be a good idea to investigate how the latent space influences the ability for the RNN to make long term prediction. The latent space is currently a black box so this should be investigated further by means of a (partly) supervised latent space. A Reduced Order Model (ROM) could for example be used to make the supervised latent space. An encoder can learn to add specifics which are not captured well by the ROM to create a unsupervised latent space. The supervised and unsupervised latent spaces can be concatenated and be used by the decoder as an input. This would allow for a study in how the RNN makes predictions for future states.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

[2] Léon Bottou, Corinna Cortes, John S Denker, Harris Drucker, Isabelle Guyon, Lawrence D Jackel, Yann LeCun, Urs A Muller, Edward Sackinger, Patrice Simard, et al. Comparison of classifier methods: a case study in handwritten digit recognition. In *Pattern Recognition, 1994. Vol. 2-Conference B: Computer Vision & Image Processing., Proceedings of the 12th IAPR International. Conference on*, volume 2, pages 77–82. IEEE, 1994.

[3] Chakravarty R Alla Chaitanya, Anton S Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM Transactions on Graphics (TOG)*, 36(4):98, 2017.

[4] François Chollet et al. Keras. `https://keras.io`, 2015.

[5] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[6] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[9] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[10] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 7, 2017.

[11] SJ Hulshoff. Eia an euler code for internal aeroacoustics: method description and user's guide. delft, the netherlands: Faculty of aerospace engineering. *Delft University of Technology.(p. xiii, xiv, 69, 70, 72, 139, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 199)*, 2016.

[12] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[14] Simeon Kostadinov. Understanding gru networks. `https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be`, 2017. Accessed: 2019-06-07.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[16] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[17] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[18] Christopher Olah. Understanding lstm networks. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 2015. Accessed: 2019-06-07.

[19] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[20] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[21] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[22] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[23] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.

[24] Steffen Wiewel, Moritz Becher, and Nils Thuerey. Latent-space physics: Towards learning the temporal evolution of fluid flow. *arXiv preprint arXiv:1802.10123*, 2018.