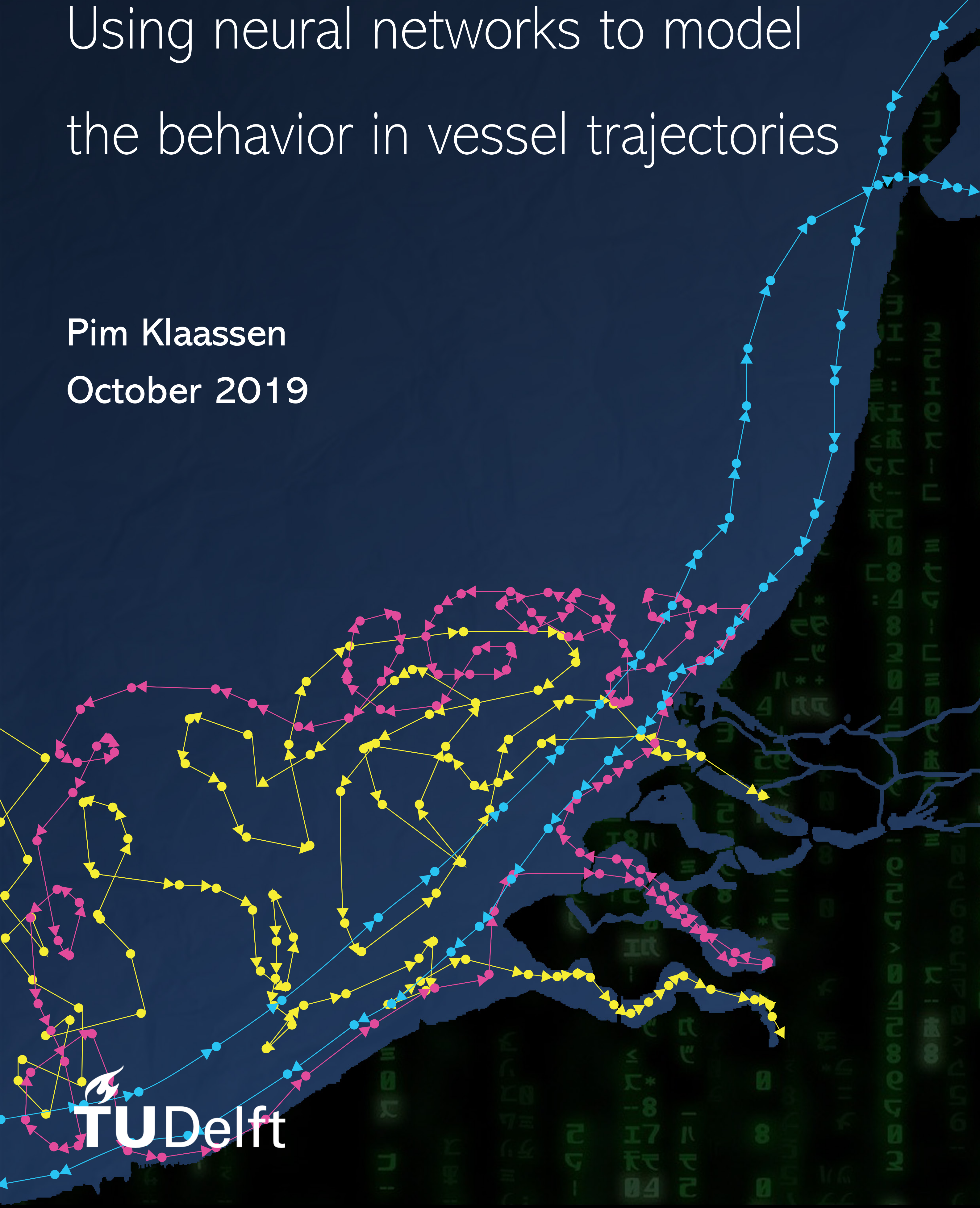
MSc thesis in Geomatics for the Built Environment

Using neural networks to model the behavior in vessel trajectories

Pim Klaassen

October 2019



TU Delft

USING NEURAL NETWORKS TO MODEL THE BEHAVIOR IN VESSEL TRAJECTORIES

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of

Master of Science in Geomatics for the Built Environment

by

Pim Klaassen

September 2019

Supervisors:	Martijn Meijers
	Edward Verbree
Co-reader:	Ihor Smal

ABSTRACT

The contemporary trend shows a shift from rule-based algorithms to *deep learning*. In the last few years, this field has been developing rapidly and its popularity has increased to a large extent. This happened for a good reason, since deep learning was able to solve some of the hardest problems in fields like image recognition, natural language processing and speech recognition. A large proportion of its success has to be credited to the explosion of big data in the past two decades. Structured data sets are essential to deep learning systems.

The rising amount of *automatic identification system* data is a key example in the current big data boom. The automatic identification system produces spatiotemporal movement data of vessels. It was designed as a collision avoidance system, but researchers have been looking into ways to leverage its data for other tasks. Analyzing behavior in the movement data of vessels can help policymakers and monitoring operators with decision making processes. Improving these processes lead to safer and more resilient marine environments.

Unfortunately, the possibility of applying deep learning on vessel movement data is an underexposed topic. This project attempts to explore the research gap in this topic. The objective is therefore to give an overview of the possibilities, complications and opportunities given the current state of the art. Ultimately, this project may serve as a rough guide for those who wish to explore the crossings where deep learning and vessel movement data meet.

ACKNOWLEDGEMENTS

This thesis is the result of almost a year of hard work and the final product of my time as a Geomatics student. Although graduating is an individual process, I could not have finished it without the help and support of my supervisors, family and friends.

First of all I would like to express special thanks to my supervisors, Martijn Meijers and Edward Verbree. During our biweekly meetings, we had long and profound discussions which were very fruitful to the course of this project. Martijn has an unparalleled commitment. I do not recall a moment when he was not available to help me out with a problem, were it the fixing of a bug or discussing about the project itself. Edward often sparked my imagination by delivering accurate input during our sessions. This helped me to focus on the common thread of my thesis and to formulate my conclusions in a more determined fashion.

Then I would like to thank Ihor Smal for co-reading my thesis. His expertise in deep learning was of great value for making decisions throughout the process. We brainstormed with enthusiasm about the possibilities of deep learning and movement data, which was very encouraging.

Parts of this graduation project would not have been as accurate without the assistance of Niels Hintzen. I would like to thank him for informing me about the current state of fisheries research in the Netherlands. His contribution to the data preparation was essential to the progress of this project.

My parents and brother helped me with certain parts of the project, providing feedback and new ideas to try out. But most of all, they offered warm company during the periods I visited them.

Finally, I would like to thank Lize for her mental support in the past few months. Not only did she contribute to the graphical appearance of this thesis, but she also assisted me in the process of formulating my ideas.

CONTENTS

LIST OF FIGURES

ACRONYMS

GNSS	global navigation satellite systems	13
GIS	geographical information system	17
AIS	Automatic Identification System	1
DL	deep learning	1
NN	neural network	1
LSTM	long short-term memory	10
GPU	graphics processing units	1
IMO	International Maritime Organization	12
ML	Machine Learning	1
AI	artificial intelligence	1
NLP	natural language processing	1
IUU	illegal, unreported and unregulated	2
MLP	multilayer perceptron	6
MSE	mean squared error	8
SGD	stochastic gradient descent	8
ReLU	rectified linear unit	7
tanh	hyperbolic tangent	7
RNN	recurrent neural network	9
ADADELTA	adaptive delta	9
MMSI	maritime mobile service identity	12
t-SNE	t-distributed stochastic neighbor embedding	21
DBMS	database management system	17
VMS	vessel monitoring system	29
ROT	rate of turn	12
Adam	adaptive moment estimation	9
COG	course over ground	12
SVM	support vector machines	14
NNS	nearest neighbor search	14
DBSCAN	density based spatial clustering of applications with noise	14
seq2seq	sequence to sequence	15

1 INTRODUCTION

In the past decades, a rising amount of Automatic Identification System (AIS) data has become available. Originally serving as a collision avoidance system, the data that is being produced by AIS is arousing an increasing interest in the fields of surveillance, security and planning [Mascaro et al., 2014]. As the data is high in volume and complex by nature, deep learning (DL) algorithms present themselves as useful tools for analysis, such as different variants of the neural network (NN). These tools have become available recently due to the increasing performance of graphics processing units (GPU) and progress in algorithmic theories. The highly developing field of DL opens new doors in the area of vessel behavior analysis. This project will explore to what extent NN's can contribute to modeling behavior in vessel movement data, by conducting some experiments with real data (e.g. detecting fishing behavior). In the upcoming sections a general overview of the topic and its relevance will be laid out, followed by the formulation of a research question.

1.1 GENERAL BACKGROUND

This project intersects at three major topics: AIS, DL—more specifically NN's— and the analysis of movement data. AIS is a standardized procedure, intended for the automatic exchange of nautical information between vessels and between vessels and shore-based facilities [IMO, 2013a]. The objectives of AIS comprise of enhancing safety and protection of the marine environment as well as safety and efficiency of navigation. The system produces movement data—spatiotemporal information about moving vessels—which can be recorded. Analyzing this data can support decision- and policymakers by giving insight about what vessels did in the past, or what they are up to in the future. In other words, the vessels' behavior.

DL is a means of analyzing data (not in particular movement data). It is a type of Machine Learning (ML) and an approach to artificial intelligence (AI). ML is the capability in computer systems to obtain knowledge by looking for patterns in raw data. Roughly put, DL is capable of finding more complex patterns than ML. DL has become more popular for a number of reasons; the increasing amount of available data, improvement in computer infrastructure and the fact that DL has solved complex problems with increasing accuracy over time [Goodfellow et al., 2016]. One of the main algorithms in DL is the NN, which will be central in this project. DL is being used in a vast amount of applications, among others: image recognition, natural language processing (NLP), medical image analysis, toxicology, recommendation systems and speech recognition. However, movement data analysis is currently not one of the common applications which makes use of DL. [Graser, 2019]

1.2 RELEVANCE

Analyzing recorded AIS data (i.e. the analysis of movement data) serves multiple purposes in the field of surveillance, security and planning. Ever since the volume of AIS data increased, there has been a growing interest in analyzing this data to reveal threats to security, illegal trafficking and other risks [Mascaro et al., 2014].

In the realm of planning, mapping the spatial distribution of fishing effort is a key challenge in marine conservation [de Souza et al., 2016]. The world's oceans are the least observed regions on Earth [NOAA, 2018], yet they are of vital importance for humanity. As pointed out by Kroodsma and Sullivan [2016], about 3 billion people get at least one-fifth of their animal protein from seafood. The lack of observation and the need of marine resources expose the oceans to the risk of over fishing [Kroodsma and Sullivan, 2016]. Based on observations by monitoring systems of fishing effort, this risk can be mitigated by applying marine conservation.

Anomaly detection is a task in marine security and surveillance. It concerns the detection of deviations from patterns in moving vessels. This can be done by a signature-based approach or by creating a model that represents normal behavior and comparing a vessel's behavior to that model [Mascaro et al., 2014].

One of the more recently researched activities in illegal, unreported and unregulated (IUU) fishing are transshipments. It concerns the activity of fishing vessels transferring catch to refrigerated cargo vessels. Transshipments contribute to IUU fishing for multiple reasons. First of all, the fishing vessels can stay in their fishing region to save fuel costs and sell their catch more easily. Secondly, transshipments allow fishermen to mix illegal catch into the legal market, reducing transparency and traceability in fisheries. About 15% of the annual global catch is IUU. A third reason is the linkage to human trafficking, allowing captains to transfer crew members or keeping them on their vessels for months or even years [Miller et al., 2018]. The main challenge is to be able to detect this kind of behavior, to engage in legal action.

In the most general sense, it is useful to detect any type of behavior the surveillance operator, security monitor or marine planner is interested in. In all of these applications, recorded AIS data can be analyzed to serve this goal. The analysis of recorded AIS data is now being considered as a valuable tool [Robards et al., 2016], as opposed to other more expensive techniques like the use of tracking drones, satellite radar or high resolution imagery [Kroodsma and Sullivan, 2016].

1.3 AIM

As described in the previous sections, there is an increase in the application of DL algorithms. The contemporary trends show a shift from rule-based algorithms and ML to DL. It is commonly believed that DL will have many more successes in the future LeCun et al. [2015]. While it is not completely new to the field of geoinformatics it is also not generally applied. As Graser [2019] puts it: 'geoinformatics is largely playing catch-up with the quick development in machine learning (including deep learning) that promise new and previously unseen possibilities'.

Given these observations, there is a research gap in the application of DL in geoinformatics—more specifically, movement data analysis. This project attempts to explore that gap. There is not yet an apparent way of how to deal with movement data in DL. This problem ranges from data representation to what kind of algorithms have to be used. Therefore the ultimate aim of this research is to stipulate guidelines on how to apply DL on vessel movement data.

1.4 RESEARCH QUESTION

To reach the objectives of some of the applications that were mentioned above, mathematical models need to be derived. In technical terms, part of the movement data—that contains consecutive points—which was produced by a vessel is called a *trajectory*. Along the lines of this terminology, the research question central to this project states:

To what extent can neural networks (NN) contribute to modeling the behavior in vessel trajectories?

To answer this question, a research methodology will be proposed in Chapter 3. Following this methodology, multiple experiments will be conducted. Trajectories have a certain length and can be represented in different ways. Not only the geographical component, but also other attributes can be part of a trajectory. Different types of NN's exist, and they can be configured in many ways. In the sign of these observations, the following sub questions attempt to partition the main research question:

- *What is a good way to represent a trajectory for a NN and what features should be included?*
- *How should the trajectories be segmented and how should the modeling be segmented?*
- *Which type of NN is fit to model the behavior in vessel trajectories?*
- *How well do NN's perform at modeling the behavior in vessel trajectories?*

1.5 RESEARCH SCOPE

The topic of this project is new and has not yet been thoroughly researched. This project will therefore be exploratory. Meaning that a broad range of experiments will be conducted. The methodology that is proposed in Chapter 3 is in part already the product of trial and error. Some of the experiments that were conducted before this methodology was proposed will still be commented on in Chapter 6

This project will primarily focus on data representation and the configuration of NN's. The main objective is to explore ways of how NN' can be applied on vessel movement data. All the experiment that will be conducted are dedicated to this objective. However, this project is not a *complete* overview of what types of NN's can be applied to vessel trajectories and how they perform compared to each other. This would rather be a continuation of this research. There is simply not enough time and too much uncertainty in this field to accomplish this. Because of this, some decisions had to be made early on in the project, which will be elaborated on in Chapter 5.

2 THEORETICAL BACKGROUND AND RELATED WORK

In Chapter 1, a general background of the topic was laid out, and the central research question was formulated. In this chapter, a theoretical basis of more detail will be elaborated on. In order to understand the methodology presented in this research, it is essential to know the mathematical basics of NN's. Additionally, a more technical overview of AIS will be provided.

2.1 DEEP LEARNING AND NEURAL NETWORKS

2.1.1 Concepts and terminology

As pointed out in Section 1.1 DL is a type of ML, which is the capability in computer systems to obtain knowledge by looking for patterns in raw data. This capability is reached when algorithms can learn from experience. The algorithms do so by correlating input information with possible output information. Every bit of input information is called a *feature*. Features can be designed manually—as is often the case in ML—or automatically. DL enables automatic feature design, this is useful because manual feature design can be time consuming and complex [Goodfellow et al., 2016].

To demonstrate this with an example, consider a system that has to predict whether a tumor is malignant or benign. A manual approach to feature design would involve a doctor inserting various pieces of relevant information into the system, e.g. the size of the tumor. In DL a whole PET scan image would be the input of the system, which then extracts relevant features from it. The reason why it is called *deep* learning is because the input data is used to construct a nested hierarchy of concepts with different levels of complexity and abstraction. Figure 2.1 illustrates this stacked hierarchy of concepts. The images are visualized features from 5 of the 22 layers of GoogLeNet, an algorithm which can classify images into 1,000 distinct classes [Szegedy et al., 2015]. The hierarchy of concepts is clearly visible, ranging along edges, textures, patterns, parts and objects.

Multiple types of NN's exist, however the central building block of a NN is the *perceptron*. It was invented by psychologist Frank Rosenblatt in 1958 and was defined as '*[...] a set of signal generating units (or "neurons") connected together to form a network.*

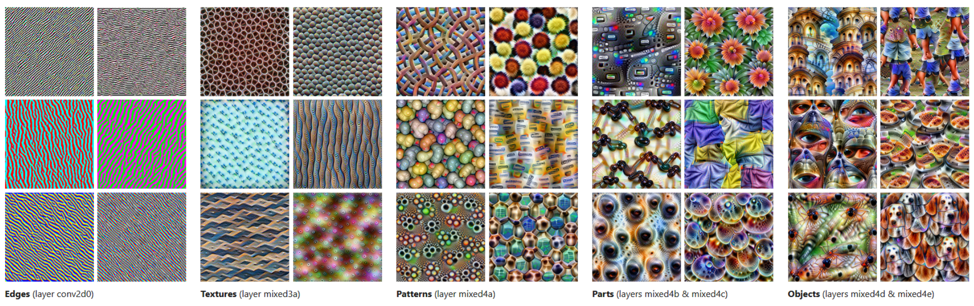

Edges (layer conv2d0)
Textures (layer mixed3a)
Patterns (layer mixed4a)
Parts (layers mixed4b & mixed4c)
Objects (layers mixed4d & mixed4e)

Figure 2.1: "How neural networks build up their understanding of images." [Olah et al., 2017]

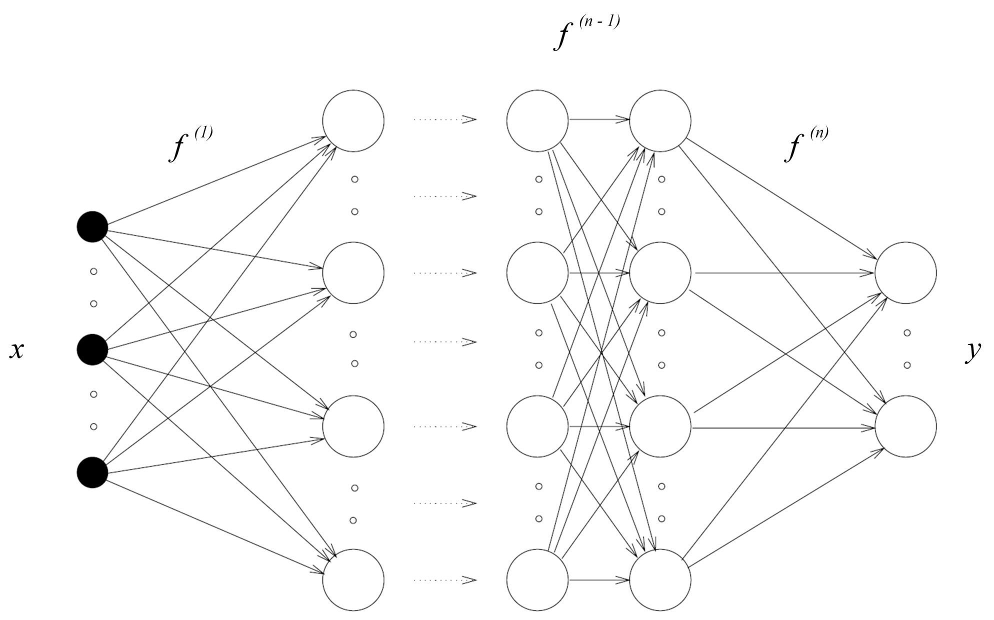


Figure 2.2: A diagrammatic representation of a multilayer perceptron. [Kröse and van der Smagt, 1991]

Rosenblatt then continues: *Each of these units, upon receiving a suitable input signal [...] responds by generating an output signal [...]. Each perceptron includes a sensory input [...] and one or more output units [...].'* [Rosenblatt, 1963]

The modern approach of a perceptron is the multilayer perceptron (MLP). As the name suggests, it consists out of multiple layers: *input layer*, *output layer* and one or more *hidden layers* in between. The number of hidden layers defines the *depth* of the MLP. Each layer consists of an arbitrary amount of units. Throughout this paper, the units will be called *neurons*.

2.1.2 Mathematical concepts

Providing a complete mathematical description of NN's would be both impossible and out of scope for this research project. However, some of the basic mathematical concepts are essential to the understanding of the rest of this document.

A MLP is a *feedforward* network. This means that the information flows from the input layer to the hidden layers and finally to the output layer. The output is not reused as a new input, as would be the case in a network with *feedback* connections. The objective of a NN is to find a function f^*. The NN defines a mapping $\mathbf{y} = f(\mathbf{x}; \Theta)$ for input $\mathbf{x}$ to output $\mathbf{y}$ and approximates the optimal parameters Θ. A NN is actually a composite function, with each layer representing a function. The first layer would be $f^{(1)}$, which maps the input of the network to the output of the first layer. In a network consisting of n layers, the function that maps the output of $f^{(n-1)}$ to the final output is $f^{(n)}$ or simply the *output layer*, subsequently also the last layer. In this fashion, a network which has four layers could be written as a chain of functions:

$$f(\mathbf{x}) = f^{(4)}(f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))) \tag{2.1}$$

The primary objective of *training* a NN is to let $f(\mathbf{x})$ approximate $f^*(\mathbf{x})$. To accomplish this, *training data* is used, which exists out of noisy observations of $f^*(\mathbf{x})$. The observations are called *samples* and constitute input-output examples. Consider the task of finding a function that predicts the location of a satellite at any given time in the future. The training data would consist out of historical samples taken from the satellite trajectory. The samples would be input-output examples; in this case a time

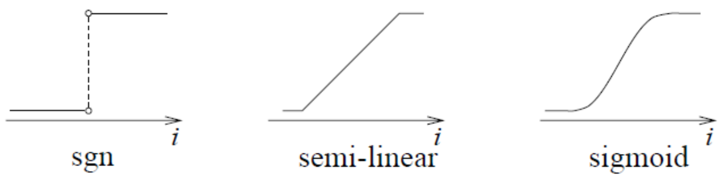


Figure 2.3: Common activation functions. [Kröse and van der Smagt, 1991]

x as input and a 3-dimensional coordinate $\mathbf{y}$ as output where $\mathbf{y} \approx f^*(x)$. The NN uses these training samples to *learn* the parameters Θ. When the network is trained, it can predict outputs of previously unseen inputs. [Goodfellow et al., 2016]

It is now clear how the different layers in a network are stacked, but how do they look like internally? Figure 2.2 shows a diagrammatic representation of a MLP. The black dots are inputs and the white dots are neurons. Consider $\mathbf{x}$ to be a vector with values x_1, x_2 and x_3. The parameters Θ typically consist of two different types: *weights* w and *biases* b. In Figure 2.2, the weights are represented by the arrows. Individual neurons are calculated by taking the weighted sum of inputs and passing it through a function σ:

$$n_1^{(1)} = \sigma(w_1^{(1)}x_1 + w_2^{(1)}x_2 + w_3^{(1)}x_3 + b_1^{(1)}) \tag{2.2}$$

In Equation 2.2 the superscripts indicate the layer of the network and the subscript indicates the index of the input/weight/bias. To simplify things, matrix notation can be used to indicate the calculation of an entire layer:

$$f^{(1)}(\mathbf{x}) = \sigma(\mathbf{w^{(1)}} \cdot \mathbf{x} + \mathbf{b^{(1)}}) \tag{2.3}$$

In the case of Figure 2.2 $\mathbf{w^{(1)}}$ is a 4×3 matrix and $\mathbf{x}$ a column vector with three features.

Function σ is known as an *activation function*. It can be any kind of function—depending on the application—but is usually non-linear. A very simple, non-linear activation function is the *sign* function:

$$sgn(x) = \frac{d}{dx}|x|,\ x \neq 0 \tag{2.4}$$

Figure 2.3 displays some common activation functions. In image recognition, semi-linear functions are widely used, e.g. the rectified linear unit (ReLU):

$$ReLU(x) = max(0,\ x) \tag{2.5}$$

The *sigmoid* function—together with some other non-linear functions, like the hyperbolic tangent (tanh)—are widely used in NLP:

$$S(x) = \frac{e^x}{e^x + 1} \tag{2.6}$$

Each layer of a network can have different activation functions, this is part of the *network architecture*. The activation function decides whether the neuron activates or not. A neuron that results in 0 through its activation function does not propagate any signal to the rest of the network.

2.1.3 Training a neural network

After having seen that a NN uses training data to learn the parameters Θ, the question that still arises is *how* it can learn from training data. The main engine behind training a NN is *backpropagation*. But before elaborating on the concept of backpropagation, an analogy might simplify the section that follows. In its essence, a computer system that learns from training data is not very different from a human that learns from sensory information. Consider a 12- to 19-month-old that learns to walk. During this learning period the infant averages 2368 steps and falls 17 times per hour [Adolph et al., 2012]. Infants undergo a tremendous amount of loco motor experience within this period, resulting in more steps and less falling. The child's learning trajectory is subject to a bonus-malus system, in which walking is rewarded and falling is punished. Whether the child walks successfully or falls, gives a direction to the way connections have to be updated in the brain. The learning process in the infant's brain as well as in the computer system answer the question *how to adjust the internal state so that undesirable consequences minimize?* [1]

A NN initially starts with a random set of parameters Θ, which obviously results in wrong outputs. It then compares those outputs to the correct outputs of the training data, which is called *ground truth*. The comparison is done using a *loss function* L. The loss function gives an indication of how wrong the network is, given the current state of Θ. In a *regression* problem—like the satellite example in Section 2.1.2—the loss function is typically the mean squared error (MSE) or something similar:

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \textit{ where } \hat{y} \textit{ is predicted and } y \textit{ is ground truth.} \tag{2.7}$$

NN's can also solve categorical problems—like the tumor *classification* example in Section 2.1.1. In this case, the output is a vector that represents the class probability distribution. Therefore, *cross-entropy* is often used as a loss function in those types of problems:

$$L = -\frac{1}{n}\sum_{i=1}^{n} y_i \cdot ln(\hat{y}_i) + (1 - y_i) \cdot ln(1 - \hat{y}_i) \tag{2.8}$$

In both of these typical loss functions, deviations from ground truth are penalized. If the network predicts many wrong outputs, the loss is high. Therefore an optimization algorithm is needed to minimize the loss, this can be done through backpropagation. Backpropagation is simply computing the gradients of the loss in the NN with respect to Θ:

$$\triangledown L = \left(\frac{\delta L}{\delta\theta_1}, \frac{\delta L}{\delta\theta_2}, \ldots, \frac{\delta L}{\delta\theta_n}\right), \textit{ where } n \textit{ is the number of parameters.} \tag{2.9}$$

The computed gradients are used in an optimization algorithm of choice to update Θ. A typical *optimizer* is stochastic gradient descent (SGD). SGD is a solution to calculating the gradients for every sample in the training data set, which would be very slow and wasteful. Instead, a random batch of samples is selected and the gradient is computed at once. This way the network can *generalize* over data and learn faster through parallel computation using GPU's. Θ is updated through SGD:

$$\theta_i^{new} = \theta_i - k \cdot \frac{\delta L}{\delta\theta_i}, \textit{ where } k \textit{ is the learning rate.} \tag{2.10}$$

As previously discussed, the network is a large composite function f with nonlinearity. The optimization problem is therefore highly non-convex. Because of this, convergence is reached at a local minimum after many iterations. One iteration constitutes one forward-backward pass of the network. One *epoch* is done when the whole training data set was processed through random batches. The number of epochs has to be decided prior to running the network.

1 This is purely a conceptual comparison. Artificial neural networks are by no means comparable to the unrivaled complexity of the human brain.

2.1.4 hyperparameters

The parameters that need to be set before running the network are known as *hyperparameters*. Different hyperparameters lead to different results in training. Some of the relevant hyperparameters have already been discussed before. This section provides a brief summary of hyperparameters:

- **Learning rate**: the learning rate is the constant in Equation 2.10, usually initially set to 0.01. It is used in SGD and other optimizers, but some optimizers—like adaptive delta (ADADELTA) [Zeiler D., 2012] and adaptive moment estimation (Adam)—autonomously regulate the learning rate. This parameter has a big influence of the learning process. If the learning rate is too large, the optimizer may never converge. If the learning rate is too small, the network may not learn at all.

- **Epochs**: the number of epochs indicates how many times the complete training data set has to be processed. Too many epochs can result in *overfitting*, which means the network will adapt to the training data only and does not generalize. The number of epochs very much depends on the batch size and how many data there are.

- **Batch size**: the batch size is usually set in the range of 8, 16, 32, 64, 128 and 256—to exploit parallel processing abilities in GPU's—but can be any positive integer. A small batch size results in more iterations per epoch and vice versa. Larger batch sizes are computationally more efficient if GPU is available, although it depends on the type of NN that is being used.

- **Layer architecture**: the layer architecture constitutes the number of hidden layers and the number of neurons per layer. But also the types of activation functions inside the neurons. More units in a NN will result in increased computation time and do not guarantee better performance. There is no rational rule of thumb in designing the layer architecture; it tilts more towards trial and error. Often it is recommended to start with a minimalistic design and expand afterwards. The activation functions very much depend on the type of NN, which was discussed in Section 2.1.2 and will be further discussed in Section 2.1.6.

2.1.5 Recurrent neural networks

In the precious section, the MLP was used to demonstrate the components and training of a NN. However, different types of NN's exist, among which the recurrent neural network (RNN). RNN's are designed to process *sequential* data $\mathbf{X} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\tau)}\}$, with *timestep* t ranging from 1 to τ. The key characteristics that were discussed in Section 2.1.2 and Section 2.1.3 remain broadly the same for RNN's, only the network architecture is different. [Goodfellow et al., 2016]

Instead of vectors, the input samples now become sequences of vectors which may have variable lengths. A RNN can be best understood as a MLP that consecutively processes each timestep of the sequence. After the MLP processes a timestep, output $\mathbf{h}^{(t)}$—or the *hidden state*, a fixed-size vector—is used as an additional input together with the next timestep. It recurrently continues this process, until final state $\mathbf{t}^{(\tau)}$ is reached. This way, RNN's have the ability to access relevant information from previous timesteps, i.e. it has memory. Figure 2.4 illustrates this concept, where f is equivalent to the mapping function in Section 2.1.2. It is also possible to create a sequence as output, known as *seq2seq*. In that case, instead of the last hidden state, the intermediate hidden states would together form the output. The dimensions of input and output sequence may vary.

An interesting property of RNN's is that they can be *bidirectional*. Sometimes, the thing that has to be predicted depends not only on the passed states, but on

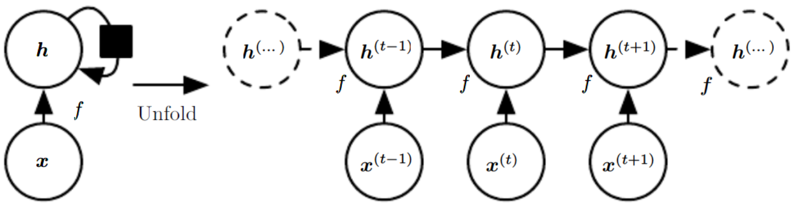


Figure 2.4: Diagrammatic representation of a recurrent neural network. [Goodfellow et al., 2016]

the whole sequence. To facilitate this, RNN's can be designed bidirectionally; they process hidden states from the past *and* the future.

2.1.6 Long short-term memory

To train a RNN, backpropagation (Equation 2.9) can be applied in exactly the same way as it would be applied to a MLP. After this, any optimizer could be used, e.g. SGD or ADADELTA. However, one of the problems that can occur is that as the sequences become longer, the gradients can vanish or explode over time [Goodfellow et al., 2016]. This phenomenon can hinder learning. A solution is provided in the long short-term memory (LSTM) network. LSTM networks have *gates* built into their architecture. The gates are used to 'forget' information if it is irrelevant or let it freely flow to the next time steps if it is relevant. This way, the network can 'regulate' its memory in both long- and short-term.

by design, LSTM networks use sigmoid (Equation 2.5) and tanh activation functions. However, most DL software packages allow to change the activation function. Different activation functions influence to what extent the gradients vanish or explode.

2.1.7 Encoder-decoder networks

Recall from Section 2.1.2 that a NN learns from input-output pairs. This type of learning is known as *supervised* learning. Sometimes the outputs or *labels* are not available. If this is the case, a domain expert could be asked to label data. But sometimes this is very hard to do, as will be seen in Chapter 5.

Encoder-decoder networks (also autoencoders) are *unsupervised* ways of training data. Instead of showing the network what is correct, the network has to figure it out itself. It can do this by making *representations* of the data samples. The representation is just a vector z of arbitrary length. Representations that are close to each other, should share some characteristics in the real world. In this fashion, the *space* where the samples are represented can be used to perform a simple clustering algorithm. This space is abstract and hidden, and is therefore often referred to as the *latent* space. The representations are often called *latent vectors* (or the bottleneck).

The latent vectors can be obtained by applying an encoder-decoder network. Which—as the name implies—contains and encoder network and a decoder network. The encoder network *encodes* a sample into a latent vector and the decoder network *decodes* the latent vector back into the input sample. The input-output pair is therefore actually an input-input pair. This type of network is useful to reduce complex objects (e.g. trajectories of variable length or images) to simple vectors, while preserving their characteristics.

The loss function of encoder-decoder networks is called the *reconstruction loss*, because the network attempts to reconstruct the input sample. Only when this reconstruction is good, the latent vectors will be good representations of the samples. The reconstruction loss is usually an MSE.

There exists a more elaborate version of the regular encoder-decoder network: the variational encoder-decoder network. This network approaches the same problem from a probabilistic perspective. Explaining the mathematics of variational encoder-decoder networks would be beyond the scope of this project. Therefore, an intuitive explanation is presented. The latent space in the regular network might not be continuous, variational encoder-decoder models try to solve this by imposing a probability distribution over the latent vectors. In other words, the bottleneck approximates a multivariate Gaussian distribution. From this distribution, z is sampled and decoded back into x. The distribution is represented by mean vector μ and variance vector σ.

To train this network, not only the reconstruction loss can be used. The probability distribution has to be forced into a unit Gaussian. This part of the loss is simply added as a term to the reconstruction loss. The KL-divergence of the distribution with a unit Gaussian used to do this. KL-divergence is a degree of similarity between probability distributions.

2.1.8 Validation and performance metrics

There are some general performance metrics that will be used throughout this project. The metric can be applied to classification results, both binary and multi-class problems. When an algorithm applies classification, there are four possible outcomes:

- **true positive**: the algorithm correctly classified an object that belongs to class X as belonging to class X.
- **true negative**: the algorithm correctly classified an object that belongs to class Y as *not* belonging to class X.
- **false positive**: the algorithm wrongfully classified an object that belongs to class Y as belonging to class X.
- **false negative**: the algorithm wrongfully classified an object that belongs to class Y as *not* belonging to class Y

Below is a brief overview of these metrics:

- **accuracy** is the most straightforward metric. It is simply the ratio of classes that were predicted correctly and the whole population:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.11}$$

- **precision** takes gives some extra information about the performance. If an algorithm randomly assigns classes in a skewed distribution, the accuracy can still be high. But the performance is bad, because the algorithm did not learn anything. This is where precision can help to assess the algorithms performance:

$$precision = \frac{TP}{TP + FP} \tag{2.12}$$

In words, the precision says what fraction of the objects that are classified positively are actually correct.

- **recall** is the counterpart of precision:

$$precision = \frac{TP}{TP + FN} \tag{2.13}$$

  In words, recall says what fraction of the objects that belong to a certain class, were actually classified correctly.

- **f1-score** is the harmonic mean of precision and recall:

$$f1\ score = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{2.14}$$

2.2 AUTOMATIC IDENTIFICATION SYSTEM

In Section 1.1 was already pointed out that AIS is a system for the exchange of nautical information between vessels and between vessels and shore-based facilities. The AIS information which is sent by vessels consists of three different types: static information, voyage-related information and dynamic information. Static information is entered on installation and contains fields like maritime mobile service identity (MMSI) number, call sign and name, ship type, International Maritime Organization (IMO) number and ship dimensions. Unless the ship undergoes changes in these values - like change of name or change of type - this information stays the same.

Voyage-related information is about specific variables of a voyage the vessel is currently engaged in, such as draught (this changes because of the load), hazardous cargo, destination and estimated time of arrival. These fields are to be manually entered at the beginning of a journey. Dynamic information contains a positional report with accuracy, course over ground (COG), speed, rate of turn (ROT), true heading and navigational status [IMO, 2013b].

These types of information are distributed over several message types. There are 27 types in total [Raymond, 2016], but for this project only a few are relevant. Types 1, 2 and 3, or Position report class A contain dynamic information and are transmitted every 2 to 180 seconds, depending on the vessel's activity [ESA, 2019]. AIS Messages of type 5 contain static and voyage-related information and are transmitted every 6 minutes. These messages are less reliable as they have to be entered manually and sometimes contain a wrong bit length [Raymond, 2016].

The messages are AIVDM data packets, which typically look something like this:

!AIVDM,1,1,,B,13btcbovi<0CMW:Me0WTeCjj28=m,0*17

The messages are comma separated strings with seven fields. When the above message is parsed it results in the following information:

1. **!AIVDM** means that this is an AIVDM packet, which is a specific protocol. AIVDO packets do also exist, the difference is that AIVDM packets are from other vessels and AIVDO packets are from your own vessel.

2. **1** indicates that this message is the first in a possible sequence of messages. There are a limited amount of bits available per packet and therefore sometimes the payload (see below) needs to be distributed over multiple packets.

3. **1** indicates over how many packets the payload will be distributed. This is a single line message.

4. This field is empty in this case, because it would contain a sequence ID. This ID is used concatenate multi line messages.

Bits	Description	Units
0-5	Message Type	Constant: 1-3
6-7	Repeat Indicator	Message repeat count
8-37	MMSI	9 decimal digits
38-41	Navigation Status	
42-49	Rate of Turn (ROT)	decoded 0 to 708 degrees/min 128 = not available
50-59	Speed Over Ground (SOG)	0 to 102 knots 1023 = not available
60-60	Position Accuracy	
61-88	Longitude	Minutes/10000
89-115	Latitude	Minutes/10000
116-127	Course Over Ground (COG)	Relative to true north, degrees
128-136	True Heading (HDG)	0 to 359 degrees, 511 = not available
137-142	Time Stamp	Second of UTC timestamp
143-144	Maneuver Indicator	
145-147	Spare	Not used
148-148	RAIM flag	
149-167	Radio status	

Table 2.1: Payload fields of dynamic messages, or position report class A (types 1, 2 and 3). Raymond [2016]

5. **B** is the radio channel code, this message was sent at a radio frequency of 162.025 Mhz.

6. **13btcbovi<oCMW:MeoWTeCjj28=m** is the payload. This part of the packet needs to be decoded to retrieve the dynamic or static information.

7. **0** is the number of bits that is required to pad the payload to a 6 bit boundary

The suffix ***17** is a data integrity checksum, to validate the authenticity of the message. Table 2.1 contains the fields of information in the message payload of position report class A. The payload is embedded in the messages as an encoded string. Each consecutive sequence of bits is assigned to a field of information and is decoded in its own way. For this project, only a few of the fields will be used.

2.3 RELATED WORK

Since AIS, movement data in general and DL methods are quite new, the related work in this area is as well. This means that most research until now was exploratory by nature. This research focuses on the analysis of historical AIS data in particular, because the data structure of AIS and the behavior of vessels are things in its own. However, there has been research into analyzing movement data in other categories and trajectories in general which could be useful to look into. For example, the analysis of global navigation satellite systems (GNSS) trajectories of pedestrians, taxi-drivers and animals or frameworks for analyzing geographical trajectories. [Andrienko et al., 2008]

Also, this research will focus on NN's as a tool for analysis. Nonetheless—in the same style as argued above—other tools exist and have been explored to perform this task. As those tools are often referred to as traditional methods in comparison to NN's, this terminology will be held on here as well.

2.3.1 Analysis of movement data with traditional machine learning methods

To carry out the applications that were described above, models have to be derived from the specific movement data at hand. This can be done using traditional methods, like *Gaussian mixture models*, support vector machines (SVM) and *random forests*. These are all clustering algorithms with the objective of grouping trajectories in a space. Where distances in this space represent similarities between trajectories.

Li et al. [2006a] developed a method that extracts movement features from the trajectories of objects. Similar movements are clustered and generalized based on these movement features, the objects can than be put into a multi-level feature space to classify them (resembling the latent space of Section 2.1.7). This method is used to detect anomalies, as being trajectories in the feature space which do not fit in a particular cluster.

In past research, nearest neighbor search (NNS) algorithms have been used to query trajectories from a database. NNS search can be used to cluster the trajectories in a feature space and query trajectories that are alike in absolute space. R-trees or TPR-trees (time parametrized tree) are often used as data structures to query from Frentzos et al. [2007].

Gaffney and Smyth [1999] explored probabilistic mixture regression models and the expectation maximization algorithm to cluster trajectories. In this research, trajectories referred to all data with response variables over time, such as moving objects in video frames or response curves in drug therapy monitoring.

A common algorithm for clustering movement behavior of entities is density based spatial clustering of applications with noise (DBSCAN). Tang et al. [2016] used a variant of DBSCAN to model travel behavior for public transit, to provide decision support for urban construction. DBSCAN is also used to model traffic behavior in busy harbors [Li et al., 2017].

Mascaro et al. [2014] argues that many methods to create models are not transparent enough for a human argent, such as a surveillance monitor, to interact with. Therefore they used Bayesian networks for analyzing vessel behavior and the detection of anomalies.

de Souza et al. [2016] proposed a framework to detect fishing activity by using different approaches for different types of fishing vessels (trawlers, long liners and purse seiners). The approaches comprised of hidden Markov models based on vessel speed, a data mining approach which is inspired by studies on animal movement and a multi-layered filtering strategy de Souza et al. [2016].

A more general framework was proposed by Zheng and Yu [2015], coining this topic *trajectory data mining*, as a response to its increasingly important role in a wide range of fields. The framework is an elaborate collection of pattern mining techniques which can be used for trajectory data, like trajectory classification and outlier detection. It also contains a taxonomy of different types of movement data and a description of multiple data representations for trajectories.

2.3.2 Analysis of movement data using neural networks

As discussed in previous sections, the neural network is a fairly new player in contemporary (practical) research. Since large amounts of GNSS/movement data becoming available is also a recent development, research on the analysis of movement data using NN's is limited.

Jiang et al. [2016] explored the performance of various traditional data mining methods compared to neural networks to classify parts of a trajectory as fishing or non-fishing. The research looked into the interpretation of AIS points prior to applying the algorithms. A sliding window method was used and a distinction was made between treating the window as raw points, linearly interpolated points or a grid form. Manually labeled data were used to train encoder-decoder networks, SVM's and random forests.

A more static approach was researched by Russo et al. [2011], attempting to predict gear types of harbor to harbor trajectories of fishermen. They use a fully connected NN that takes as 35 variables as input, which are derived from a full trip, like speed profile, change of heading profile and sea depth profile.

Yao et al. [2017] addresses the problem that many of the techniques described above create models that use spatiotemporal similarity measures which are not space- and time-invariant. As a solution, they propose the use representation learning to generalize clusters of behavior which are similar but occurred on different locations and times. Just like Li et al. [2006b], they extract movement features from the trajectory to create sequences which are better representations of the entity's behavior. After this, they apply sequence to sequence (seq2seq) LSTM encoder-decoder networks—which are widely used in speech recognition—to transform the sequences into fixed-size vector representations. These representations robustly encode movement behavior in latent space and can be clustered into separate groups. Experiments with this method show accurate results in classifying vessel types with historical AIS data as well as classifying movement behavior in synthetic movement data.

It is also possible to model multi-entity behavior using neural networks. For example, Wang et al. [2018] explore the use of representation learning with auto-encoders to cluster similar driving encounters of GNSS-equipped vehicles. This method can be used to classify traffic situations in autonomous driving or detect anomalous situations in traffic encounters.

3 METHODOLOGY

3.1 INTRODUCTION

This chapter will elaborate on a methodology to model behavior in vessel trajectories. Figure 3.1 represents a flowchart of this methodology. The whole process consists of roughly 3 phases: the data ingestion phase, the training phase and the testing phase. The data ingestion phase takes place from retrieving data from the data source until the storage of samples. The samples are bits of information which can be directly used for training. This means that they are normalized, clipped or transformed in an appropriate way for the task at hand. Also, if needed, a subset of the original data is taken.

The flowchart in Figure 3.1 also contains an important taxonomy of problems which introduces the training phase. Behavior can be modeled within a trajectory or between different trajectories. At first sight, these two ways of approaching the problem seem to be the same; one trajectory can be split into multiple trajectories, also making it a sub-trajectory modeling problem. However, there are some practical difficulties which complicate this idea. This will be further discussed in Section 3.3.2. As discussed in Section 2.1.7, DL problems can be supervised or unsupervised. Problems have to be solved in an unsupervised way if there is no labeled data available. In this case the trajectories need to be processed by an encoder-decoder model. When labeled data is available, the trajectories can be processed by a standard NN with categorical loss functions. The choice of network types and architectures will be elaborated on in Section 3.3. The different types of networks in Figure 3.1 require different input-output examples for training.

A trained network has to be tested with previously unseen test data. In the testing phase, the results form a continuous feedback loop for adjustments in the hyperparameters and network architecture. Once the results are satisfactory, the model can be deployed.

3.2 DATA INGESTION PHASE

3.2.1 Database

The data ingestion of AIS data can origin directly from the source or from a historical data dump. In both cases, the data needs to be stored in such a way that it can be restructured efficiently. Therefore, a database management system (DBMS) is essential. The DBMS allows to query subsets of data and apply complex filters. It processes data both reliably and fast, and can be accessed remotely. AIS produces a sufficiently structured data format to integrate it into relational tables.

Only two tables are needed: a table for the dynamic AIS messages and one for the static messages. The tables are related through the MMSI number. Table 3.1 demonstrates some of the relevant field names in the dynamic and static tables.

When enabling spatial objects in the DBMS is possible, a connection to geographical information system (GIS) software can be established. In this project, PostGIS and QGIS were used for this end. This ability supports visual inspection of data, which at some point could be useful when, for instance, mapping results.

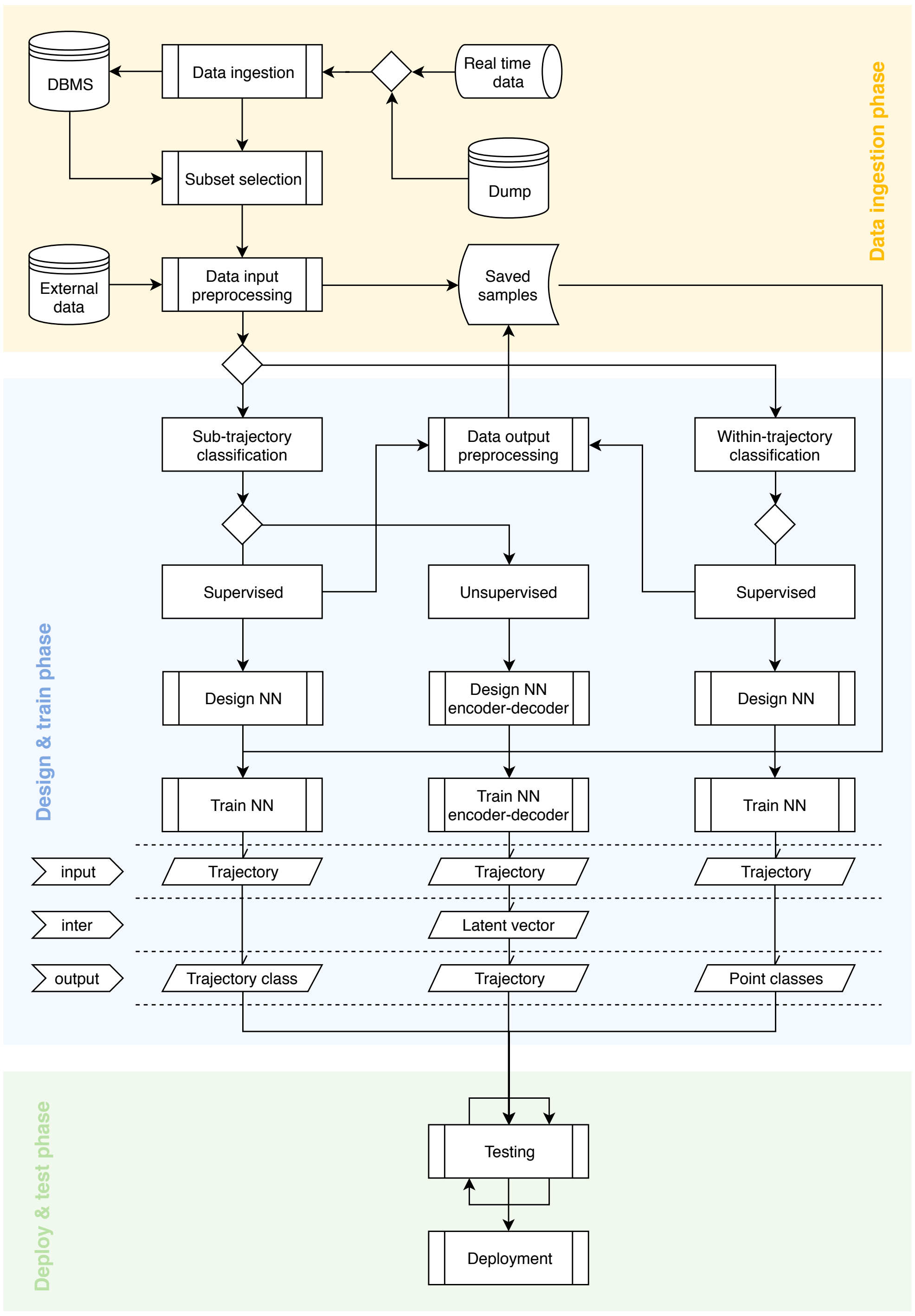


Figure 3.1: Flowchart of methodology.

DBMS table	fields				
dynamic	mmsi	timestamp	longitude	latitude	turn	speed
static	mmsi	timestamp	shipname	shiptype	destination	...

Table 3.1: Field names of the dynamic and static tables.

3.2.2 Data preprocessing

In order for a NN to be trained on data, that data has to be in the correct format. As well the input as the output data (in the case of supervised learning). When the training data is synthetic, it can be created in a way that is already meets the requirements of a format. However, for real AIS data, samples have to be extracted and transformed. The extractions and transformations are morphological and numerical.

Morphological operations concern the shape of the data. The most important factor in this step is to determine the length of a trajectory. This is a tedious decision to make because of multiple reasons. First of all, trajectories in an AIS database do not have a beginning or an end. Vessels have a tremendous range of different activity and it is simply not possible to assign rules to the beginning and ending of individual samples in an automatic way. Furthermore, the trajectories are frequently subject to irregular data gaps, making them discontinuous. In the attempt to choose beginning and end points based on regular time intervals, we may find samples with no points at all. Then there is the problem that NN's usually process a fixed amount of data points across one data set (although solutions to this problem exist). In short, there is a trade off between morphological and temporal continuity.

The numerical operations are straightforward as they comprise of simple normalization operations. However, many different ways of normalizing data exist and its absence can obstruct learning. One of the most common ways of preprocessing data before DL is to subtract the mean and divide by the standard deviation. But before being able to do this, outliers have to be removed. Outliers primarily exist because of GNSS errors and damaged sensors. Custom smoothing filters can be used to correct for outliers.

During data preprocessing is it possible to merge the the AIS data with other external spatial data sources. These data sources might correlate to the behavior that needs to be modeled and altogether increase the NNs predictive performance.

The input samples are stored in separate, binary files. As data sets can become very big, it is desirable that they are partitioned. DL consumes a lot of memory and therefore it is common to load samples per batch. It could be possible to let the samples flow from the DBMS directly. However, the transformation operations that were discussed above should then be integral to that process. This would require a carefully thought out infrastructure, parametrized according to the needs of different tasks.

The data output samples can only be created when some crucial decisions about the network type were made. Note that in a deployed environment the data ingestion phase is not a single event, but rather a continuous cycle. Real rime data would constantly flow into the database, which can subsequently be used to make predictions or update the networks.

3.3 DESIGN & TRAIN PHASE

When all data is ingested and preprocessed, a NN can be designed and trained. Figure 3.1 demonstrates a proposed taxonomy of problems: sub-trajectory classification and within-trajectory classification. These two concepts then branch into supervised and unsupervised problems. All the NN's that are proposed in this methodology contain LSTM networks. This decision stems from the sequential nature of AIS data. Trajectories essentially are time series, and because LSTM networks are good at processing time series (see Section 2.1.6) it seems appropriate to apply them in this situation. Furthermore, the networks are hybrid, as they switch between LSTM and regular layers. The following sections will discuss *general* design of the proposed networks. Specific design parameters will be discussed in Chapter 5.

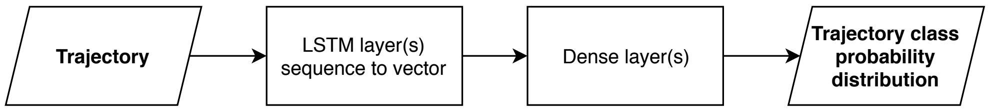

Trajectory
LSTM layer(s) sequence to vector
Dense layer(s)
Trajectory class probability distribution

Figure 3.2: A blueprint for a sub-trajectory supervised learning problem.

3.3.1 Sub-trajectory classification

In a sub-trajectory classification problem, behavior is modeled between samples. In other words, each sample is subject to one kind of behavior. We wish to classify the separate samples into behavioral classes. If this problem is supervised, the classes are predefined and the data is labeled. For instance, AIS includes a field with the ship type (see Section 2.2). This information can be used as a label to classify vessel trajectories into different ship types (the ship type is actually a high-level behavior). If there is no such information available, the problem is unsupervised and predetermining classes is not possible.

Supervised method

Supervised learning in a sub-trajectory classification problem is the most simple case in the proposed methodology. As illustrated in Figure 3.1 this problem requires a trained NN which takes a trajectory as input and transforms it into a class probability. Prior to this, the network has to be designed and trained. Figure 3.2 proposes a blueprint for the network design in this particular case. The first series of layers are LSTM layers that at some point transform the sequences into vectors. The signals are then propagated through regular layers (called *dense* layers in Keras terminology) into a class probability distribution. This distribution contains the probabilities of the trajectory belonging to a certain class. During training, the probability distributions are used to calculate the categorical cross entropy, in the loss function. After training, the arguments of the maxima are the classes that will be assigned to the trajectories.

Figure 3.2 does not make any suggestion about the number of layers, but only determines the main infrastructure. The layer architecture can be designed as deep as needed for the specific task at hand. Technically, the dense layer(s) can be skipped if the LSTM layer directly outputs a class probability distribution. But this would rule out the use of any other dense layer in the rest of the network.

Unsupervised method

Classifying trajectories without labeled data is a challenging task. There are no input-output examples to train on and the derived classes may not resonate with the classes one was initially looking for. In this methodology, representation learning is proposed as a method to cluster trajectories into classes. The output is not directly a class probability distribution, but rather an embedding of trajectories into a multidimensional space. The trajectories are represented in this latent space, and their inter-relational distances represent the degree of difference in behavior in the real world. This concept can be implemented by both regular- as variational encoder-decoder models (see Section 2.1.7 for more details). Although the more complex variational variant has more interesting properties, both are included in the methodology. Figure 3.3 and Figure 3.4 illustrate blueprints for both cases.

Regular encoder-decoder model

The first part of this model has the same infrastructure as the supervised model discussed in Section 3.3.1. It encodes a trajectory into a vector, but this time it is not a class probability distribution but the latent representation z. The network continues in reverse order, to decode z back into the original trajectory. The loss function

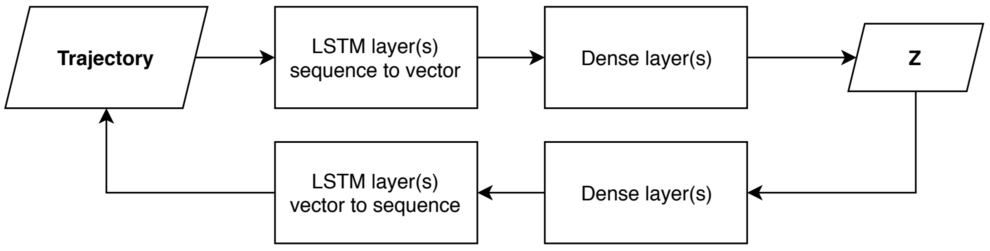

Trajectory
LSTM layer(s) sequence to vector
Dense layer(s)
Z
Dense layer(s)
LSTM layer(s) vector to sequence

Figure 3.3: A blueprint for a sub-trajectory unsupervised learning problem, using a regular encoder-decoder model.

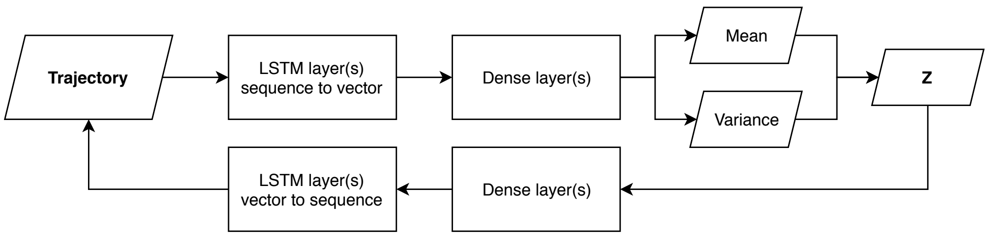

Trajectory
LSTM layer(s) sequence to vector
Dense layer(s)
Mean
Variance
Z
Dense layer(s)
LSTM layer(s) vector to sequence

Figure 3.4: A blueprint for a sub-trajectory unsupervised learning problem, using a variational encoder-decoder model.

in this network is the reconstruction error of x and x'. A trained encoder-decoder model preserves the class difference between trajectories into the latent space. The latent space can have as many dimensions as is desirable. Less dimensions take a toll on the reconstructions loss and too many will cause overfitting. The trajectories can be classified by applying a clustering algorithm on the latent vectors. t-distributed stochastic neighbor embedding (t-SNE) can be applied to visually inspect the latent vectors.

Variational encoder-decoder model

The variational variant is the most complex model proposed in the methodology. It is an expansion of the model in Figure 3.3.1 and is demonstrated in Figure 3.4. As discussed in Section 2.1.7, a variational encoder-decoder model constrains the latent vector to be normally distributed. This constraint introduces a generative property, as vectors can be sampled and decoded into synthetic trajectories. The decoded vectors look like the trajectories from which hidden distributions they were sampled from.

The mean and variance layers are simple dense layers. z Is created by sampling from the values in those layers. The loss function has two terms: the reconstruction error of x and x' and the KL-divergence of hidden distributions with a unit Gaussian.

Just like in the regular encoder-decoder model, the latent vectors can be clustered into behavioral classes. The classes will take up distinct area's in the latent space. The size of this network has to be bigger, as information could be lost on the way. Because the network is bigger, it will take much longer to train it, depending of the size of the input samples.

3.3.2 Within-trajectory classification

Within-trajectory classification is treated as a separate problem because of segmentation. The main issue is that the break points of different classes of behavior in a trajectory are unknown. Therefore, segmenting the trajectory prior to prediction its class becomes difficult. A solution could be found in the use of a rolling time

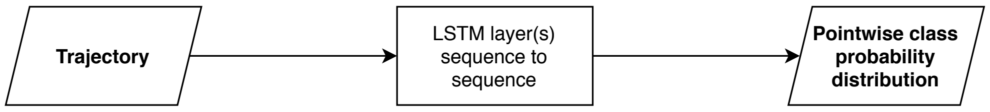

Trajectory
LSTM layer(s) sequence to sequence
Pointwise class probability distribution

Figure 3.5: A blueprint for a within-trajectory supervised learning problem.

window. However, still the length of the segmented parts would be unknown. Additionally, this solution would result in an explosion of data.

The alternative that is proposed in this methodology is a pointwise classification method. Subsequently—in a supervised problem—the labels should be assigned to the individual points. Note that within-trajectory classification has no unsupervised alternative (see Figure 3.1). The reason for this is that pointwise clustering would have to deal with the same segmentation problems as were discussed above.

Figure 3.5 demonstrates that there are no dense layers in the blueprint. The network has a *seq2seq* architecture, as it transforms a sequence into a sequence. The output sequence contains time steps with class probability distributions. In this fashion, it is possible to interpolate class transitions.

3.4 TEST & DEPLOY PHASE

When the data samples are ready, a blueprint has been selected and all the initial hyperparameters have been set, the network can be tested. This is a continuous process in which the testing results are feedback for adjustments in the network parameters that were discussed in Section 2.1.4.

To test the results, various metrics are used. The training- and validation loss show whether the network is learning sufficiently. They also show when the network is under- or overfitting. During training, the loss can be monitored in order to potentially adjust parameters which influence these factors.

After a batch of test samples is passed through a trained network, assessment metrics can be applied to the output. The test samples have to samples that the network has never seen. The reason for this is that only then can be assessed if the network is generalizing properly. Therefore, test samples in this project are always sampled from other ships than the samples in the training set. This eliminates any bias from the testing process. These metrics can indicate the degree of performance of a network. When performance is bad it can be hard to tell exactly why that is the case. There are many factors that can cause bad performance. However, it is sensible to prioritize different solutions. Chances are there was a mistake in the input data rather than the network has to be expanded to increase learning. Therefore it is wise to start out with a small network and overfit on one sample. This rules out possible error sources. From here on, the network can be expanded.

4 DATA, SOFTWARE AND HARDWARE

4.1 DATA

The data that was used in the experiments consists of real world and synthetic data. The real world data is a historical, global recording of AIS from 2016 to 2017. The synthetic data was constructed using simple rules to demonstrate some of the methods in this project.

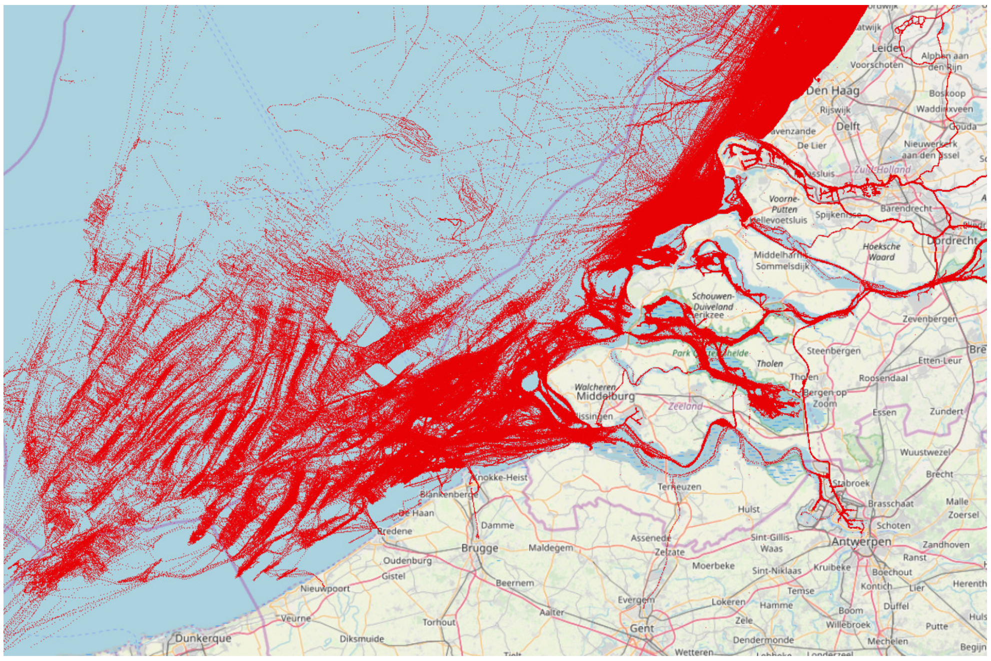


Figure 4.1: AIS data of Dutch fishing vessels.

4.1.1 AIS data set

The AIS data set was retrieved from TU Delft. It contains AIS messages of type 1, 2, 3, 5 and 17 (see Section 2.2 for more information about the message types). In total, there are 1 billion dynamic- and 65 million static messages. In database format, the data set covers approximately 300 gigabytes in volume.

Only a distinct part of this data set was used in the experiments; limited to a selection of Dutch fishing vessels. Dutch ships were selected using a filter on the MMSI numbers, given Dutch MMSI numbers start with 244, 245 or 246. This subset was further filtered to fishing vessels using the *ship type* field in the type 5 messages (see Figure 4.1).

4.1.2 Synthetic data

Synthetic data was created created to conduct some of the experiments in a more simplistic manner. By using synthetic data, the problem of missing or erroneous data can be avoided. Also, it gives some freedom in creating different classes of

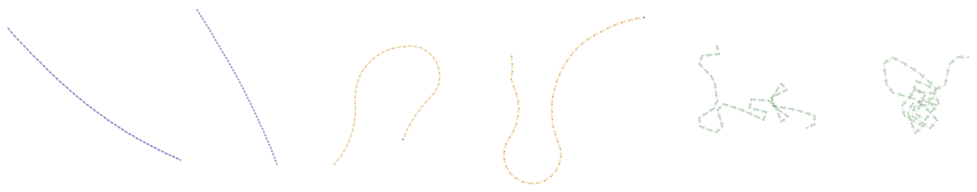


Figure 4.2: Some examples of generated synthetic trajectories.

trajectories. The algorithm that was used to create the synthetic data is straightforward. A randomized process chooses how much coordinates belong to the next segment and how curved this segment will be. Mean and standard deviation of the random process are the parameters of the algorithm. The distances between all coordinates are constant. The problems of this algorithm are discussed in Chapter 6. Some of the generated trajectories are demonstrated Figure 4.2.

4.2 SOFTWARE

The main operations for all experiments are written in *Python*, including the following libraries for specific tasks:

- **NumPy** and **Pandas** for numerical operations and pickling of data points.
- **Matplotlib** and **Seaborn** as plotting engines. Seaborn is a wrapper for Matplotlib to enhance graphics.
- **Tensorflow** and **Keras** as deep learning frameworks. Keras is a high-level interface for Tensorflow, which makes creating NN's modular.
- **Scikit-learn** for the application of t-SNE on multidimensional latent vectors.

4.3 HARDWARE

Two machines were used for different ends during this project. A personal computer was used to conduct all the experiments. A remote machine was used to store the data in a database. This machine could be used to query data from. Running neural networks on the stronger remote machine did not make a lot of difference in efficiency. This is because it had no GPU which could be exploited. However, for data volumes reasons it was still used for storage.

Personal computer

- **CPU** Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz, 2808 Mhz, 4 Core(s), 8 Logical Processor(s)
- **Memory** 8 GB
- **Operating system** Microsoft Windows 10 Home

Remote server

- **CPU** Intel(R) Xeon(R) CPU E5-2690 0 @ 2.90GHz, 8 Core(s), 32 Logical Processor(s)
- **Memory** 128 GB
- **Operating system** Red Hat Enterprise Linux 7.6 (Maipo)

5 EXPERIMENTS AND DISCUSSION

Chapter 3 and Chapter 4 elaborated on the methodology and the data that was used to test it. The coming sections will follow along these lines and describe the experiments that were conducted in this project. After the description of each experiment, an overview of results are discussed. Finally, some of the problems that were encountered will be elaborated on in each section. The structure of this section will follow the same workflow and taxonomy as presented in Figure 3.1.

Conducting experiments was a time consuming part of this project. The main issue was that NN's can take very long to train and they have a lot of hyperparameters. Additionally, the numerical and morphological choices that are made for the input data also influence training results. Before the experiments with NN's will be discussed, a short section of the data representation is included. This section will give a justification of why the trajectories were treated as sequences.

5.1 DATA REPRESENTATION

Before the methodology was created, multiple ways of representing the trajectories were explored. For example, there was some related work that represents the trajectories as raster images. Also, when the trajectories are segmented into fixed parts they could even be used as vectors instead of sequences. Both these ideas were rejected from the beginning for multiple reasons. The former idea was a more convincing candidate than the latter and was briefly experimented with. Figure 5.1 demonstrates how an AIS trajectory is segmented with raster tiles. The tile size is the leading parameter in this process. In each tile, the coordinates are interpolated and rasterized. The consecutive sequences of raster images could then be used by a NN. This method would very mainly leverage the geometrical component of the trajectories. However, multiple bands could be added to include other attributes as well. Figure 5.2 demonstrates some of the rasterized segments of an AIS trajectory.

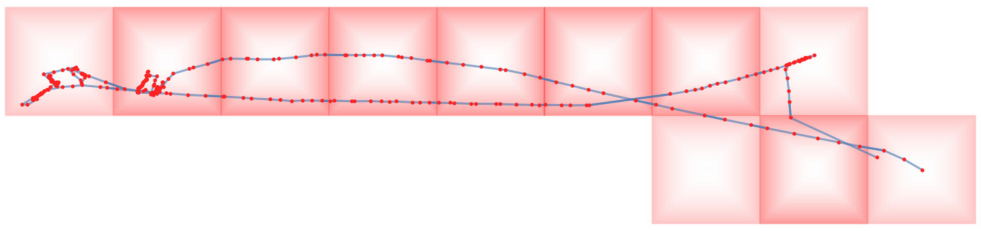


Figure 5.1: Trajectory segmentation for raster conversion in *QGIS*.

This way of representing the trajectories was abandoned after having experimented with it. The largest issue was the waste of memory in the raster images for the surrounding space. This made method very inefficient. Another problem was the segmentation, in relation to coordinate systems. For example, vessels that travel from Tokyo to Rotterdam will have to be treated differently than local journeys. Also, there were implications regarding how to include the other attributes—like speed and ROT—into the images. Because the different attributes would be included into multiple channels in the image, resulting in even more wasted space. Also, the channels would need separate normalization operations.

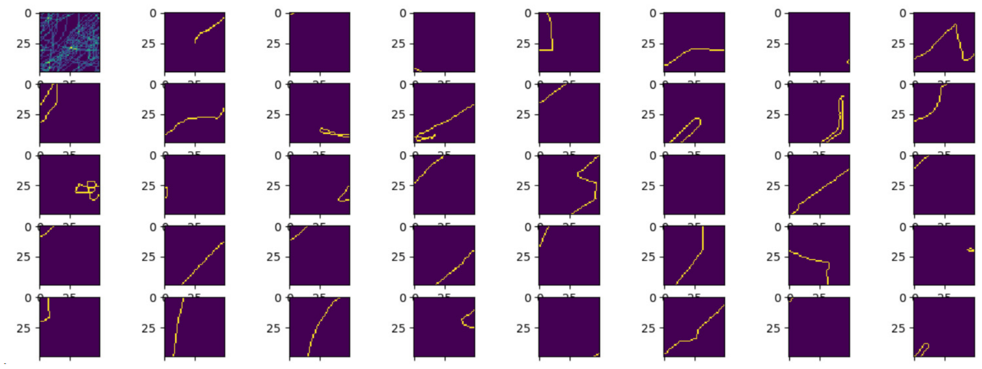


Figure 5.2: A sequence of rasterized trajectory segments.

Treating fixed sized segments as vector inputs was abandoned because of the fact that trajectories can get very long. Long sequences can be processed successively, but vectors are treated as a whole. Long trajectories can therefore occupy a lot of memory. A major problem that this representation shares with the raster image representation is that there is no indication of time direction in the data structure. A time feature could be added, but can not directly be exploited by the types of NN's that would fit these representations. Additionally, there is no obvious way of dealing with temporal gaps. In this sense, the problem of trajectory representation is very much related to Section 6.3, because it is reciprocal to the type of network that is being used.

None of the above methods are however refuted to be working alternatives to the one that was eventually used. They may work, but for the logical reasons that were presented, another path was taken. Unfortunately, testing all the methods was simply not possible within the time provided for the project. Because each method would need its own infrastructure and has its particular implications. Therefore, the trajectories are to be treated as sequences.

5.2 MODELING FISHING ACTIVITY

One of the experiments conducted in this project is the modeling of fishing activity. As discussed in Chapter 1, this is one of the key challenges in marine conservation and a very relevant problem. Therefore, it seems like an interesting opportunity to apply the proposed methodology to this problem. The objectives of this experiment are straightforward; given a historical trajectory of a fishing vessel, where did the vessel fish? By creating a universal model, any trajectory can be analyzed for fishing activity. As well historical data sets can be processed for statistical modeling, as near real-time data to leverage detection systems.

5.2.1 Data ingestion of real world data

The AIS data set was supplied as a raw ASCII file. Typically, each line of characters is an encoded AIS message. However, multi-line messages also occur. As discussed in Section 2.2, the payloads of the messages have to be decoded using specific translation rules. *Python* was used to decode all the raw messages—including multi-line messages—and import them into a *PostreSQL* database. The *Python* script prints COPY statements which are redirected to *psql*.

Static- and dynamic message tables had to be loaded separately because of different table structures. However, the same algorithm to load the data could be applied. The raw AIS messages are comma-separated and contain the information whether they are multi-line or single-line. Errors in payloads reportedly occur at a frequency

DBMS table	# tuples	data volume
global dynamic	1 077 164 571	77 GB
global static	65 860 880	8 546 MB
dutch fishing dynamic	18 918 602	1 381 MB
dutch fishing static	1 651 620	211 MB

Table 5.1: Statistics of various AIS tables.

of about 0.3% [Raymond, 2016]. Because of this, *try/catch* statements were used in the decoding methods. The final frequency of errors resonated with the literature.

Algorithm 5.1 shows how multi-line messages were included in the decoding process. The algorithm uses a queue to keep track of multi-line messages. The messages that are distributed over different lines carry the same message ID. According to their number and total amount of messages, the payloads are concatenated and decoded after the final message of a sequence was found.

Algorithm 5.1: DECODE RAW AIS

Input: A raw ASCII file S
Output: Tuples with decoded attributes for the database

```
begin
    $Q \leftarrow \{\}$;
    for $line \in S$ do
        $ID, number, total, payload \leftarrow parse(line)$;
        if $total < 2$ then
            yield $decode(payload)$;
        else
            if $ID$ in $Q$ then
                if $number = Q[ID][total]$ then
                    $payload \leftarrow Q[ID][payload] + payload$;
                    yield $decode(payload)$;
                else
                    $Q[ID] \leftarrow Q[ID][payload] + payload, total$;
                    continue;
            else
                $Q[ID] \leftarrow payload, total$;
                continue;
```

The data was loaded locally on a remote server (see Chapter 4 for details on the hardware). First, all the dynamic and static messages were loaded into the database. This drastically decreases the data volume. From then on, sub selections could be made using those tables. For example, for one of the experiments, only data of Dutch fishing vessels were used. This sub set was filtered from the main tables. Table 5.1 shows some statistics about the sizes of the data sets.

When the messages were decoded and inspected in *QGIS* it became clear that there is a significant gap between data and reality. First of all, not all ships carry AIS transponders. Even if they are obliged to carry one, they can still turn it off. Secondly, the data has to be collected by a party which owns or hires radio-communication antenna's and satellites. The antenna's are distributed over coastal area's and thus have a limited reception radius. Also, the party might not own antenna's in a region that one is interested in. When a NN is trained, it has to approximate reality. Therefore, the data also has to approximate reality. For the means of this project, a higher and more stable sampling rate would

While engaging into the experiments, this was a big problem. One could say that all the data gaps and errors are part of the system, and including them in the samples is more logical than ignoring them. However, attempts to train with raw data including all the gaps and errors was very fruitless. Only spatiotemporally continuous trajectories showed promising results.

Initially, it would have been interesting to experiment more with the static messages. For example, one of the ideas to experiment with is including the destination of vessels as input data. It seems that this could increase the predictive performance in a vessel type classification problem. By looking further in to this it became clear that this was a difficult implementation. One problem is that boatmen enter the static information into the system themselves. This results in different occurrences of the same object in the system. Table 5.2 demonstrates different string representations of the same destination that were entered into the system.

IJMUIDEN
_IJMUIDEN__________
IJMUIDEN__
IJMUIDEN____________
IJMUIDEN___________B
IJMUIDEN,_HOLLAND___
IJMUIDEN_NL_______
IJMUIDEN-NL_______
IJMUIDEN(NL)________

Table 5.2: Different occurrences of destination 'IJmuiden', where _ denotes a space character.

Of course, this problem could be solved by using regular expressions to filter strings. But additionally there are some problems that can not be solved by this solution. Most of the destination fields were not entered at all. It could also be the case that the destination that was entered is not correct or was not updated after arrival. Many of the entered destinations are random characters or nonsensical information (see Table 5.3).

Not only the static table contains these kind of errors, but also the dynamic table. Some of the vessels that were selected for training samples had defect speedometers and most of the vessels did not include the ROT. Fortunately, these features could be extracted from the coordinates. This complicated the creation of samples even more, because one speed or ROT had to be extracted from two or three consecutive coordinates. It also involved dividing by time, which resulted in the necessary zero division errors.

5.2.2 Data prepocessing

The data preprocessing of real data is a challenging task. The main reason for this is that the data is subject to damage or errors in equipment and modification by boatmen. AIS can be deliberately turned off, leaving large spatiotemporal gaps in the sequences. Some of the vessels may not have been active in the time interval in which data is available. Others might have been used for other purposes then fishing. For example, some of the vessels that were labeled as fishing vessels in

&ISHING GREUNDS_____
LBH[E4H/ZZ/$X
OKTOBERFEST
/_1DJE_VAN_DE_ZAAK__

Table 5.3: Some random or erroneous destinations entered by fishermen, where _ denotes a space character.

Fishing gear	# training samples	# test samples	speed profile
Mechanical dredge	96	16	0.2-1.5 knots
Bream trawler	96	16	2.5-5 knots
Bream trawler 225 kW+	96	16	2.5-7.5 knots
Otter trawler	96	16	2-4 knots
Total	384	64	

Table 5.4: Data set composition.

AIS were stationed in harbor for 3 months. Others were only active in large rivers like the river Rhine, were a fishing activity like beam trawling is not particularly prevalent. These difficulties do cause problems for training. When certain behavior has to be modeled, the NN has to see input examples that actually contain this behavior.

An additional problem is that filtering out these bad examples is hard to automate. The volume of the data is large and the degree of activity in AIS equipment can change over time. Designing rule-based algorithms which segment or select spatiotemporally continuous trajectories can get complex quickly and is out of scope for this project. Therefore, some simple steps were taken to create a selection of samples.

First of all, the data was ordered by MMSI and time, creating continuous trajectories per vessel. Then, the trajectories were simply ordered by the number of points they contain. The trajectories which were dense enough were inspected in *QGIS*. It then became apparent that the vessels can show highly different behavior. Unfortunately, sub classes of fishing vessels are not included as a field in AIS. Differentiating in types of fishing vessels is important for training, because the behavior that vessels show are specific to their vessel type.

To accomplish this goal, Niels Hintzen from Wageningen Marine Research was consulted. The list of current MMSI numbers was joined with vessel monitoring system (VMS) data to add the specific fishing gear and engine power of the vessels. This information was used to group the vessels create samples. The trajectories of vessels with different gear types were inspected in *QGIS* and a sub selection was made, to exclude sparse trajectories. A number of fixed length trajectories was randomly sampled from the vessels, resulting in the data set which is demonstrated in Table 5.4. (the speed profiles will be discussed in the following paragraphs)

Fishing vessels with different gear types have different modi operandi. For example, mechanical dredge fishing is conducted at a lower speed than beam trawling. 225 kW+ indicates that these type of vessels have larger engines, and thus have a larger operational radius. The samples in the test set were taken from other vessels than the samples in the training set, to eliminate any biases in the training process.

One of the assumptions that was made for this experiment is that including external data which correlates with behavior can increase the predictive performance of a NN. Following the methodology in Figure 3.1, that data was merged with the training data in this step. Bathymetry data was chosen to test with, based on the hypothesis that vessels can only fish on tracks with constant ocean depth. The data format was delivered in a raster file and was joined with the samples in *QGIS*.

To apply supervised within-trajectory classification to this data, the samples have to be labeled. The problem is, that these labels do not exist. And as it turns out, they are hard to create. The current research uses simple rule-based algorithms based on *speed* ranges of vessels. If the vessel is cruising within a certain range of speed, it is presumably fishing. Speed is the only dimension that is currently being used. However, fisheries researchers are also looking into other factors, like rate of turn and soil morphology. The samples were therefore labeled with the defined speed ranges per fishing gear. This information was gained from Wageningen Marine Research. The speed profiles are included in Table 5.4.

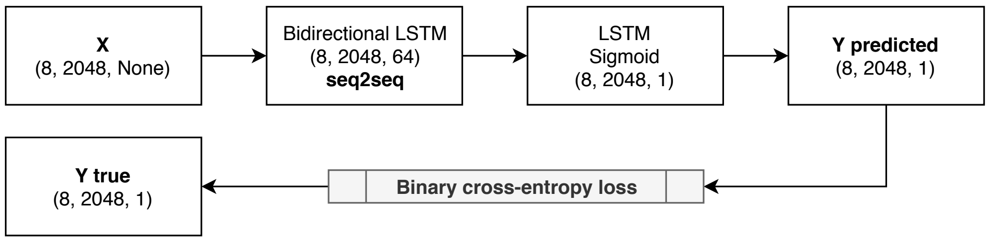


Figure 5.3: Network configuration for supervised within-trajectory classification of fishing activity.

The last step in the data preprocessing was to transform, normalize, clip and remove any outliers if necessary. The absolute coordinates were transformed into relative movement vectors *dx* and *dy* to remove the bias of absolute space. The absolute location should not have an influence on the prediction whether the vessel is fishing or not. All the attributes were normalized to a 0-1 scale. A time attribute was added, to give the network a handle on temporal gaps in the data. Time data of the 24-hour scale was transformed into two coordinates, to add cyclical continuity. Because of GNSS errors, a smoothing filter had to be applied to the relative movement vectors. To accomplish this, the outliers were replaced with the values of a simple averaging kernel. Finally, the samples were stored as binary files, ready to be used as input for the NN.

5.2.3 detecting fishing activity

To design a network to detect fishing activity, the blueprint of Figure 3.5 from the methodology was used. The desired output of this experiment is a sequence of class probability distributions, i.e. a pointwise classification. Figure 5.3 demonstrates the final configuration of the NN that was created. Each rectangle represents a layer (including input- and output layer) and includes the *shape* of the output of that layer. The shape is simply a tuple containing the number of elements per dimension. When a shape contains the keyword *None*, it means the number of elements in that dimension is not defined. The dimensions represent $(samples, time\ steps, features)$, thus the first number is the batch size. Sometimes, additional relevant information is included in the diagram, like the activation function. Also, the loss function is added.

A sequence length of 2048 time steps was used because on average this time interval spans enough variability in fishing activity. Making the sequence length longer cuts the computational efficiency and can give the NN a hard time to learn. The feature dimension is undefined, because multiple subsets of attributes were used train different networks. A batch size of 8 is quite small, but the largest that was computationally possible. A larger batch size can be very memory hungry—especially with larger sequence length—as a batch has to be computed in a single pass.

The rest of the network in Figure 5.3 shows that the first layer is a bidirectional LSTM, that converts the input sequence into another sequence with 64 abstract features. The last hidden layer converts this intermediate output into a binary sequence, using an LSTM layer. All the LSTM layers in this project use the default tanh activation function, with exception of the last layer in this network. A sigmoid function was applied to yield numbers between 0 and 1. This is not exactly a class probability distribution as formulated in Figure 3.5, but because there are only two classes of behavior (i.e. fishing and not fishing) this is possible. This also simplifies the labeling process. The output $y\ predicted$ can be compared to the ground truth $y\ true$. This is done with binary cross-entropy loss. Also, all the networks in this project were optimized using Adam, because its learning rate is self adaptive.

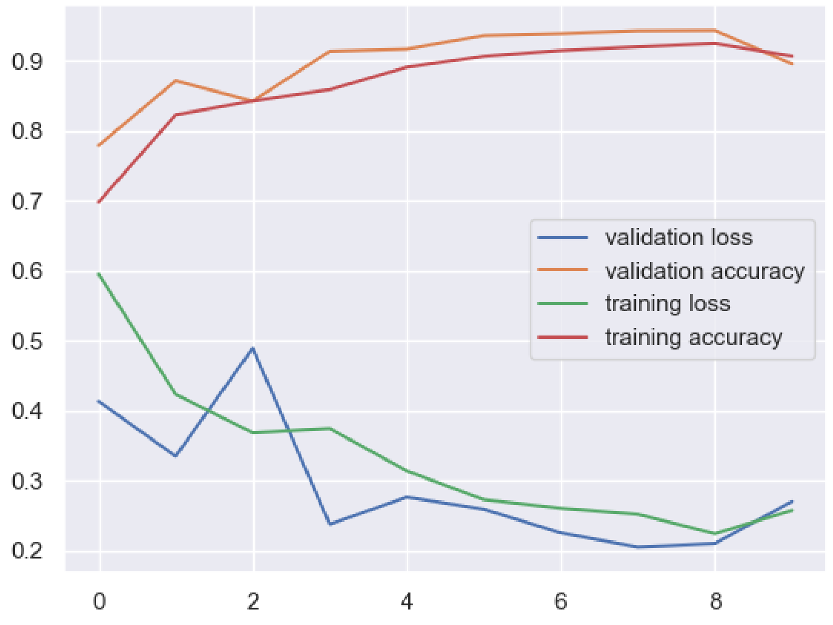


Figure 5.4: Training and validation loss, and accuracy after 10 epochs, batch size 8.

Subset	Loss	Accuracy	Precision	Recall	F1
dx, dy	0.22	0.93	0.86	0.95	0.90
Speed, ROT, time, bath	0.13	0.95	0.92	0.96	0.94
Speed	0.14	0.95	0.91	0.98	0.94
ROT	0.21	0.91	0.83	0.97	0.89
bath	0.66	0.62	0	0	NaN
Speed, ROT, bath	0.15	0.96	0.91	0.98	0.95
Speed, ROT, time	0.12	0.96	0.92	0.98	0.95

Table 5.5: Evaluation of test data for various networks. ROT indicates the rate of turn and bath indicates bathymetry.

Overall, this network is quite simple and not very deep (the *deepness* of a network expresses how many layers it contains). However, training time took about *1 hour* for 10 epochs. The learning process is depicted in Figure 5.4 (using only *dx* and *dy* features, other successful subsets look equivalent). Because the batch size was small, the number of epochs could be reduced. The graph looks healthy, because the loss decreases and the accuracy increases. Training and validation metrics do not differ too much, which means the network is generalizing correctly. The validation metrics are a bit higher than the training metrics. This could result from underfitting (increasing the amount of neurons can solve this), however this phenomenon seems to disappear near epoch 9.

After training, the test data was evaluated for all the different networks. Each network was trained with the same parameters as reported above, but with different subsets of features. Table 5.5 shows an overview of evaluation metrics for these various subsets. The first thing that comes to mind while looking at the evaluation is that bathymetry scores low on all metrics. This can not directly reject the hypothesis of external data and does not mean bathymetry has no predictive power. Obviously, using only speed works well, because the network was labeled using this feature. Furthermore, the rate of turn does a surprisingly well job at distinguishing between fishing and not fishing. Overall, apart from bathymetry, all networks all score well at the metrics. There is a structural imbalance between precision and recall, where the precision scores lower. This means the network is better when it classifies no fishing activity than when it does classify fishing activity.

The low added value of bathymetry data was probably caused by the lack of data in harbors. The harbors of fishing vessels are located just out range for the bathymetry area. As there were many samples distributed over multiple harbors it was hard to automatically set interpolated bathymetry values for the data points in harbors. Therefore NULL values were used for those points. It was assumed that bathymetry data was not needed in harbor area's anyway. Because the ship is not moving in the harbor and this should be enough information for the NN to classify behavior. In any case, this method is either flawed or bathymetry is just not a good predictor for fishing activity.

As discussed before, the task of labeling the trajectories for fishing activity was an obstacle in the research. A couple of shortsighted assumptions were made prior to diving into this effort. One assumption was that fishing activity is visible geometrically and therefore can be labeled by looking at it. This assumption is wrong. Because it turned out that it is primarily vessel speed and to a certain amount the rate of turn which are predictors for fishing activity. The second assumption was that no domain knowledge is needed to label the data. This statement was very quickly refuted by trial. Another faulty assumption was that the samples could be labeled by segment instead of by point. This is wrong, because the transitions between fishing and other activity is not regular. Implementing this method would therefore cause loss of resolution.

In hindsight, it is a pity that there is no correct labeled data available for this. The experiment right now just learns the simple speed profiles and adjusts those to new trajectories. One might argue that using a NN for this simple task is an overkill and not worth the effort. On the other hand, the network can process many trajectories of different fishing gears without the need of any prior knowledge. It can also use only one dimension like ROT and still do a reasonable job.

5.3 SUPERVISED SUB-TRAJECTORY CLASSIFICATION

The previous sections described an experiment for supervised within-trajectory classification. A simpler task is to conduct classification between trajectories. A problem that fits this part of the methodology—like detecting fishing activity—is the modeling of vessel types, based on their trajectory. The objective is to to distinguish different types of fishing vessels (or other types of vessels, for that matter). To test this method, synthetic data was used.

5.3.1 Data ingestion of synthetic data

The synthetic data was created directly, without storing it in a database. However, the trajectories were still preprocessed and stored as separate files. While creating the data it was attempted to mimic some of the typical patterns in real world vessel trajectories. The trajectories contain only a spatial component and no further attributes. One parametric algorithm was made, which can generate unique trajectories of different classes.

Figure 5.5 shows 3 trajectories generated with different parameters in the algorithm. Each trajectory contains 50 data points with equal distances (although this could be changed, see Chapter 6). By looking at the trajectories it is clear that they show different geometrical behavior. It is now the objective of a NN to model this behavior.

The trajectories were transformed into sequences of dx and dy vectors and normalized, to remove the absolute space component. This was done because there is

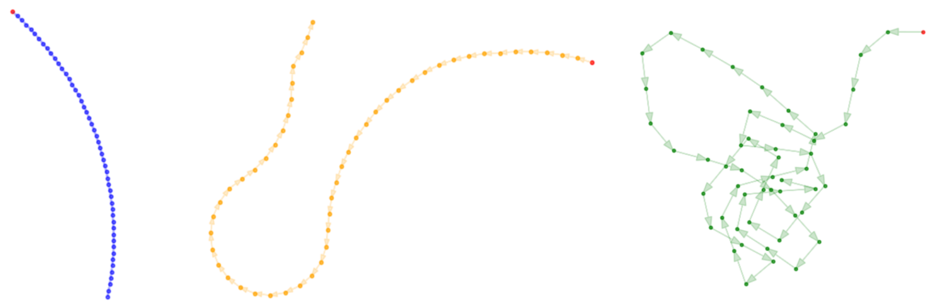


Figure 5.5: 3 trajectories generated in 3 different classes. From left to right, class A, class B and class C.

no correlation between location and behavior (of course, this is not the case with real data, where some vessel behavior might correlate to absolute space). Normalization is a somewhat ambiguous term, which in this case refers to *standardization*. Standardization is transforming the data so that it has a zero-mean and a unit standard deviation. For training, 1000 trajectories per class were generated, with a total of 3 classes. Another 200 trajectories per class were generated for testing purposes.

The process of creating synthetic data faced less technical complications. This is mainly because the whole data cleaning part could be skipped. Besides, a connection to a database was not necessary. The big question concerning the synthetic data is whether it is complex enough to prove the functioning of various networks. To avoid complicating things any further, a simple data generation algorithm was chosen to start with.

The trajectories in Figure 5.5 mimic geographical trajectories, but are very limited. They only contain spatial information as coordinates, and no additional information which AIS offers. In this sense, the trajectories only look like real vessel trajectories from a geometrical perspective. However, there is another problem still. The euclidean distances between all coordinates are constant, which means the trajectory is actually 1-dimensional. The current synthetic trajectories could be reduced to sequences of changes in angle and still contain the same information.

5.3.2 Training and results

Figure 5.6 shows the final configuration of a network to classify synthetic trajectories. The whole setup was created according to the blueprint of Figure 3.2. Because the sequence length of the synthetic trajectories is much smaller, a large batch size of 256 could be chosen. This allows to speed up the training and lets the network generalize.

This time, the first LSTM layer is a sequence to vector layer. From there on, the network continues as a dense network, with a softmax activation on the output. The softmax function transforms the output of the previous layer in a probability density distribution. Because there are three classes, categorical cross-entropy was applied.

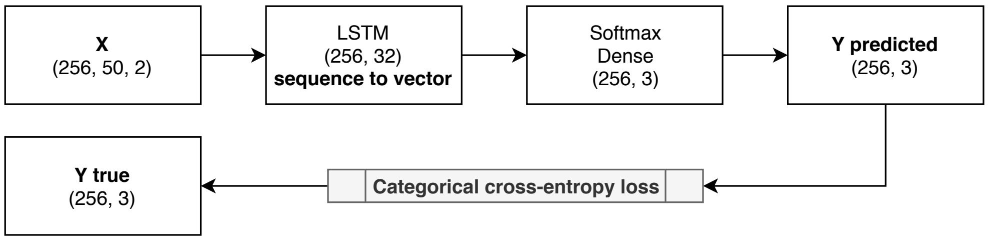


Figure 5.6: Network configuration for supervised sub-trajectory classification.

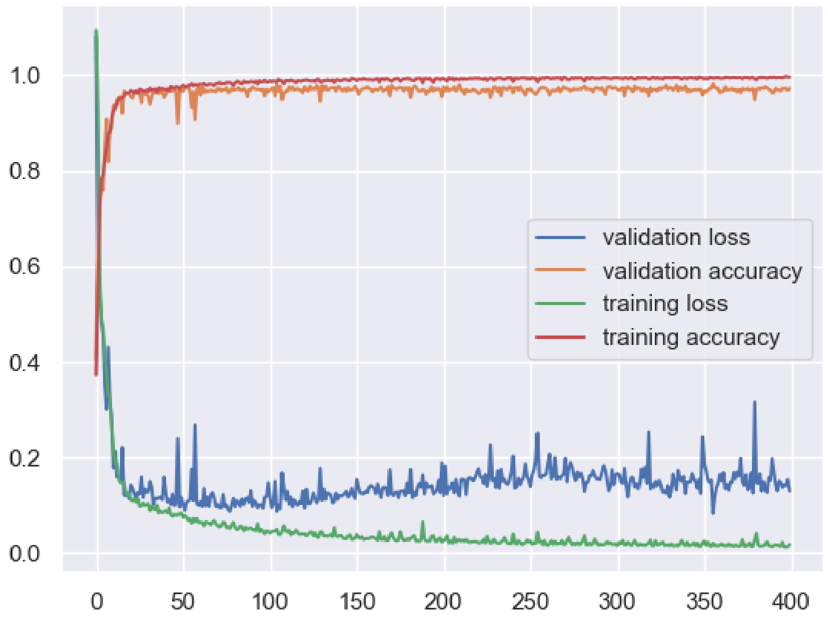


Figure 5.7: Training and validation loss, and accuracy after 400 epochs, batch size 64.

Figure 5.7 illustrates the training process. The network was ran for 400 epochs, which was more than enough to let the loss converge. It seems like there is some overfitting, however the accuracy for the validation data still look good. Table 5.6 shows the evaluation of the test data. A simple network was sufficient to classify almost all test samples correctly.

Loss	Accuracy	Precision	Recall	F1
0.12	0.98	0.97	1	0.99

Table 5.6: Evaluation of test data for supervised sub-trajectory classification of synthetic data.

5.4 MODELING VESSEL TYPES

An excellent experiment to test the ability of NN's to model behavior in vessel trajectories is modeling vessel types. In this supervised experiment, individual trajectories were classified into a fixed amount of possible vessel types. The types that were used for this experiment are cargo, tanker, fishing, tug and passenger vessels. After training the network, it should ideally be able to transform an arbitrary trajectory of a fixed size into a probability density vector, which represents the vessel type it belongs to.

5.4.1 Data ingestion of real world data

The data ingestion phase of this experiment overlapped for a great part with the process described in Section 5.2.1. The only particularity is that the trajectories of different classes had to be compiled in a balanced data set. The labels of the vessel types which were described above are already present in the AIS data.

First, an equal amount of distinct ships per vessel type was selected. In this process, the number of coordinates per vessel in the data base was counted and classified in bins per 10,000, up to over 100,000 points. Ships containing less than 10,000 coordinates were discarded, as they did not contain enough points to sample from. In general, vessels with more points tend to show less activity than vessels with less points. This is because they leave on AIS all the time, even when sitting still in harbor. To rule out any bias during training, not only where the samples distributed equally over vessel types, but also over the number of points. That is, the distribution of samples from vessels of different bins is uniform.

The complete trajectory per vessel was extracted from the database. Each time a number of non-overlapping samples of size 2000 was taken from random locations in the trajectory. Because of this, vessels in higher bins yielded more samples. This caused problems for creating a partition for train, validation and test data (split as 70/15/15). The partitions could not contain overlapping ships, as this causes the network to train in a biased setting. After shuffling with the amounts of vessels per type, bin and partition, a compromise was reached while containing balance in all three factors. In the end, each vessel type contained 8,000 samples in the training set. This should be more than enough to train a network and let it generalize properly.

5.4.2 Data preprocessing

The data preprocessing in this experiment was largely conform Section 5.2.2. An addition was that a median filter of size 3 was applied to the raw coordinates. As this data set is much larger than the fishing data set of Section 5.2.3, this was a good solution to automatically remove outliers. After this, the relative movement vectors were extracted and standardized.

As many of the samples showed inactivity for the greatest part of the trajectory, some had to be removed. This is because the network might see inactivity as a feature of a certain vessel type. Although this is not completely wrong—some vessel types *are* more active than others—this habit obstructs learning. Therefore, all samples which had less than 10% activity were removed from the data set. This threshold is somewhat arbitrary, but left enough samples in the data set to work with. Higher thresholds will remove a substantial proportion of the data set. A simple stay-point detection algorithm was used to remove these points.

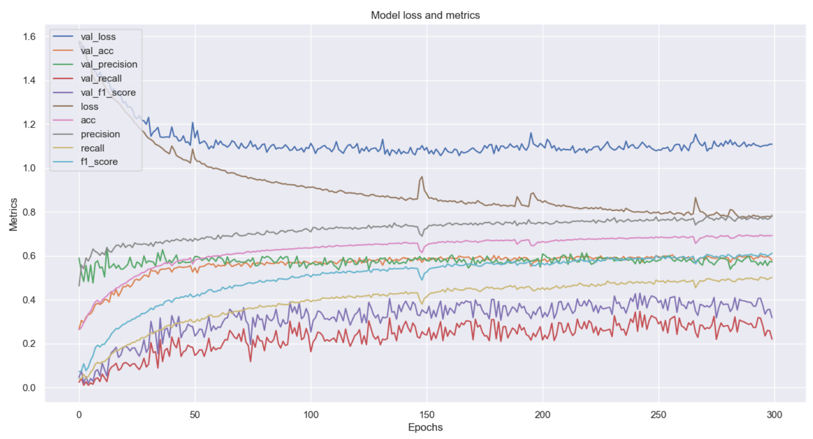


Figure 5.8: Training and validation metrics for 300 epochs, vessel type classification network.

5.4.3 Training and results

The partitions of the data were stored in a dictionary, containing the filenames of the stored samples. From this dictionary, the program could generate random batches of 64 samples for each iteration. Generating batches like this was necessary, because it is simply not possible to initially load the full data set into memory. A batch size of 64 showed the best results.

For the network, different methods were tried out incrementally. It would be cumbersome to list all those results here, however some deserve attention. First of all, it was found that using multiple stacked LSTM layers did not work at all. That is, a sequence of LSTM layers. This particular network type took very long to run, and did not learn. However, applying a single *convolutional* layer prior to LSTM layers improved learning. A convolutional layer uses a sliding window similar to kernels in image processing. The parameters of this layer sit in this sliding window.

After testing some more with this configuration it became clear that increasing the size of the sliding window to up to 200 increased learning. Setting the step size between 2 and 6 resulted in better learning. Stacking recurrent layers increased training time but did not significantly improve results. It was also found that the best results were produced by networks that had an equal amount of parameters in the convolutional and recurrent layer.

The configuration that learned best had a sliding window size of 150 and step size of 5, using 128 filters. The convolutional layer was followed by a recurrent layer with 64 units. Only the convolutional layer applies a ReLU activation function. A dropout rate of 0.2 was used in both layers. The training process with all metrics is demonstrated in Figure 5.8. 300 epochs took approximately 14 hours to run. Table 5.7 shows the final results of the test data. Figure 5.9 illustrated the confusion matrix of the network. A confusion matrix is a nice way of visualizing how the model confuses classifications in relation to each other.

Loss	Accuracy	Precision	Recall	F1
1.11	0.57	0.56	0.2	0.3

Table 5.7: Evaluation of test data for vessel type classification network.

It is visible in the confusion matrix that for example cargo and tanker vessels are more often confused than other types. This makes sense, because these two types are more similar than others. Also, it shows that the network quite confidently predicts fishing vessels. Most of the fishing predictions are correct. On the other hand, many samples that were actually fishing vessel were overlooked as something else. It comes to notice that there is an overall tilt towards tug and passenger predictions. This is presumable the case because these vessels are more inactivate, and inactivity is a trend in the complete data set. All of the observations made above explain the imbalance between precision and recall. In general, the reached accuracy of 0.57 shows that the network can learn meaningful representations of vessel trajectories.

5.5 UNSUPERVISED SUB-TRAJECTORY CLASSIFICATION

To the test methodology on unsupervised sub-trajectory classification, encoder-decoder models were designed. One standard encoder-decoder model was designed and tested and then expanded into its variational variant. The mathematical concepts of these models are explained in Section 2.1.7. The same synthetic data as described above was used in these experiments.

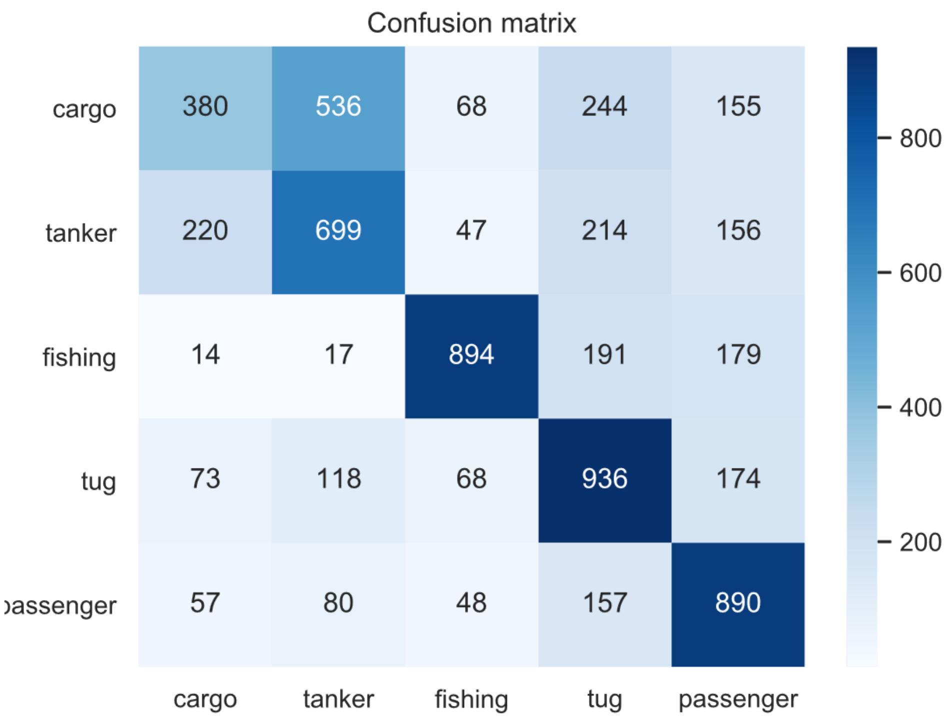


Figure 5.9: Confusion matrix of test data, vessel type classification network. X-axis shows the predicted class, Y-axis shows the true class.

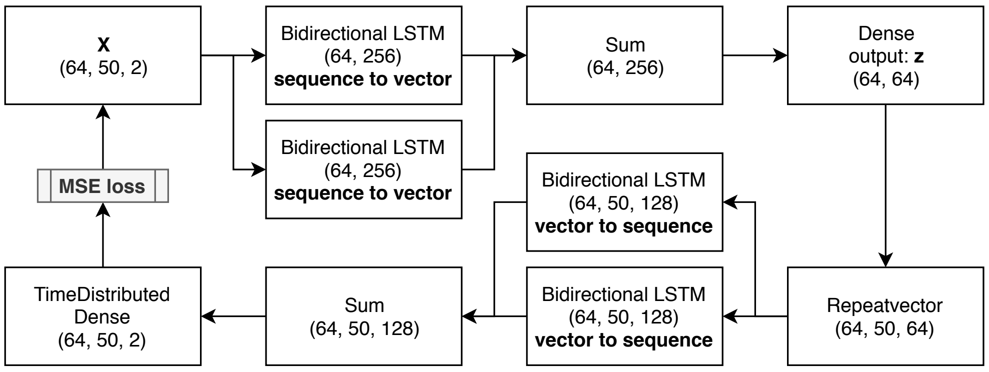


Figure 5.10: Network configuration for standard encoder-decoder model.

5.5.1 Standard encoder-decoder model

To design this network, the methodological blueprint of Figure 3.3 was used as a guideline. A bidirectional LSTM network was chosen (see Section 2.1.5), to use as many information of the trajectory as possible. All the activation functions in the network are tanh functions. Different amounts of neurons per layer, epochs and batch sizes were tested. This network has to reconstruct the input after it was compressed into a single vector (the bottleneck). Therefore, it needs more parameters than the previous networks. The final configuration is illustrated in Figure 5.10.

It is clear that this network is more elaborate. The first layer is a dual bidirectional LSTM layer which converts the input into a vector. The outputs of these layers are summed. The network continues to propagate the signals through dense layers. After this, the vector is converted back into a sequence and is processed by two more bidirectional LSTM layers. In this fashion, the original input is reconstructed. The loss function can be a simple MSE loss.

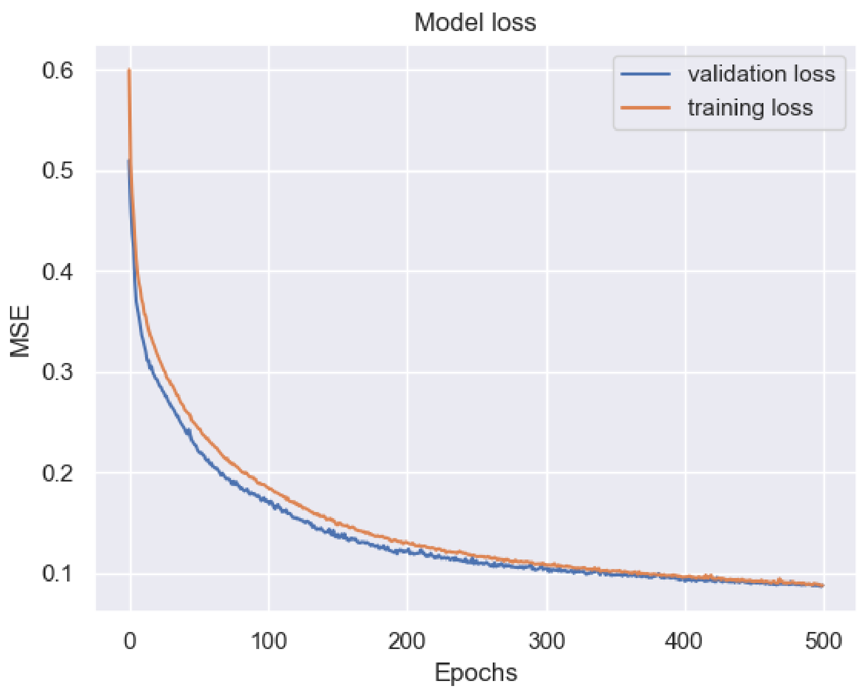


Figure 5.11: Training and validation loss for 500 epochs of standard encoder-decoder model.

Finding the best configuration for this network was harder, because it took longer to train. Therefore, it was first attempted to overfit on one sample, gradually adding different classes. The reconstructions were inspected in *matplotlib*, to check if the network was able to make proper reconstructions. When this worked, the network was expanded. *Dropout* with 10% was applied to the recurrent layers. This is a setting which stochastically 'drops out' a percentage of the inputs at each iteration. Dropout can reduce overfitting.

Figure 5.11 shows the training and validation loss over 500 epochs. Training took approximately *4 hours*. The loss is still not completely converged at epoch 500, so training it for more epochs could improve it. Instead of evaluating with the usual metrics, the latent vectors had to be inspected. However, the latent space is 64-dimensional. Therefore, a dimensionality reduction algorithm has to be used to be able to plot the vectors in 2 dimensions. To do this, t-SNE was applied. Figure 5.12 demonstrates the latent space in 2 dimensions. Each data point in the plot is a latent representation of a trajectory. The points are color-coded with their classes, respective to Figure 5.5.

Figure 5.12 clearly shows clusters of trajectories of the same class. In particular, class A and C seem to differentiate well, while B kind of sits in between. This makes sense when looking at Figure 5.5. These latent representations can be clustered in completely unsupervised settings, to look for patterns in the data.

Some attempts were made to process AIS data in an unsupervised setting for a fishing gear classification task. The hypothesis was that a regular encoder-decoder model of the likes of Figure 5.10 could be altered for the use of real data. The latent space would then demonstrate the same clusters as in Figure 5.12, but the clusters would represent fishing gears. Training this network took very long (around *8 hours*) and the trajectory lengths could not be preserved to 2048. Because of computational burden they had to be reduced to 64. This is a problem because the temporal span is too small. The trajectories will display behavior like 'cruising straight', 'turning' or 'loitering' instead of encompassing the behavior of different fishing gears.

Figure 5.13 and Figure 5.14 demonstrate the results of this experiment. While training, the network systematically underfits the training data. The loss is still

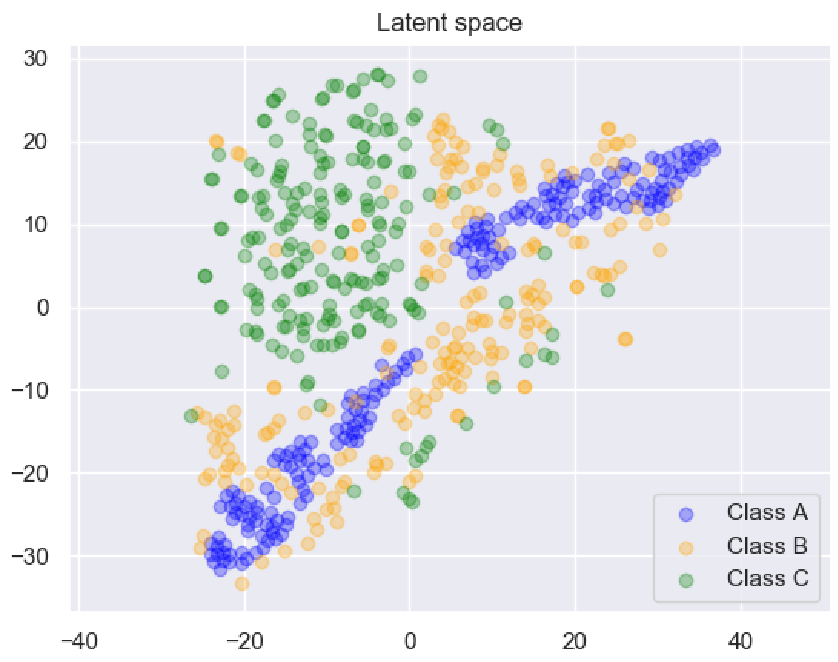


Figure 5.12: The latent representations of trajectories from the testing set, using t-SNE.

decreasing at epoch 100, but due to time limitations and computational resources training for more epochs was not conducted.

The data points in latent space are much more mixed than the structured clusters of the experiments with synthetic data. However, there are some clusters visible. There is a clear accumulation of blue points, which are the mechanical dredges. Also, some of the red points (otter trawlers) accumulate around the blob in the lower left. Why this is the case is still unknown and should be further researched.

5.5.2 Variational encoder-decoder model

To create the variational version of an encoder-decoder the previous model was expanded. Almost all layers of the network are the same, only the bottleneck looks different. Also, the loss function had to be adjusted. In this case, latent output z is not a regular layer. It is sampled from dense layers μ and σ, which represent the means and variances for each dimension in latent space. The objective is find out the latent distribution. During training, μ and σ are forced into a Gaussian distribution. This is done by calculating the KL-divergence with a unit-Gaussian (0 mean, 1 standard deviation) and adding this to the regular reconstruction loss (see Figure 5.15).

Figure 5.16 shows that after 700 epochs the loss is still not converging. This network took approximately *8 hours* to train and was very memory hungry. A batch size of 32 was the highest possible batch size without triggering out of memory errors. The loss is much higher, because the reconstruction term was weighted relative to the KL-divergence loss. This is necessary to prioritize reconstruction. A factor of the number of original dimensions of the input is the rule of thumb. This model had a harder time reconstructing trajectories than the previous one. This is simply because of the extra sampling layer and added loss term. This is also why the network needed much more iterations to train properly.

Figure 5.17 demonstrates the trajectories of the test data in latent space. It is directly visible that the data points are spread out normally. Along the same lines

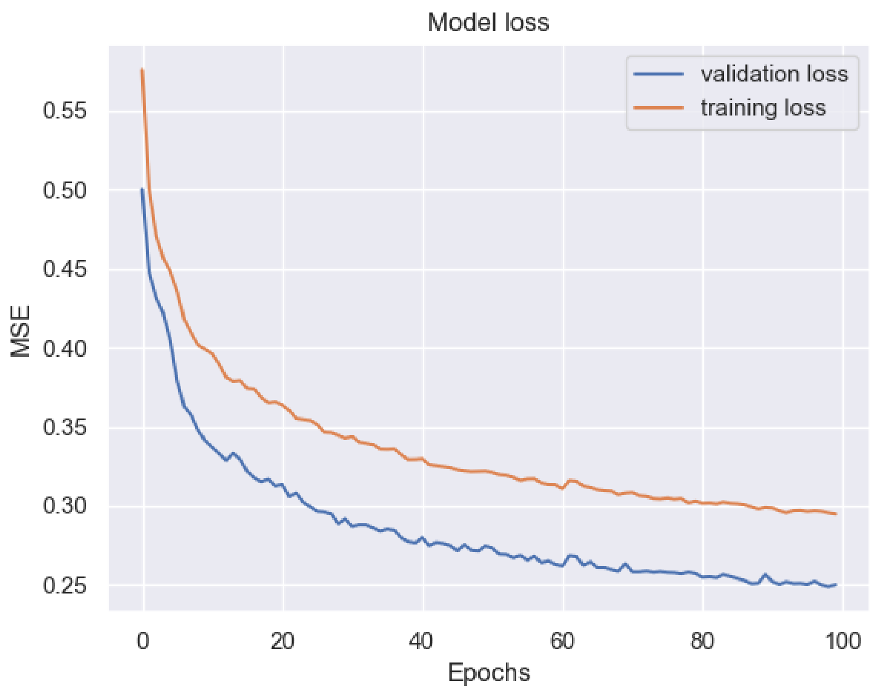


Figure 5.13: Training and validation loss for encoder-decoder model of real world data.

of Figure 5.12, class A and C are very isolated. Class B floats in between the other classes. But all categories occupy distinct area's. The latent space is now continuous and can be explored for variations of the input data. Figure 5.18 shows some of these variations. The blue trajectories were passed through the network to retrieve the parameters of their latent distributions. From these parameters, a new vector z was sampled, which could be decoded by the decoder part of the network. The result is the yellow trajectory. As opposed to the previous networks, this network has a generative aspect. It can generate new trajectories that categorically looks like an input trajectory.

Some of the code that was used in this project is included in appendix A. It contains code snippets of the vessel type classification problem and an example of the variational model discussed above.

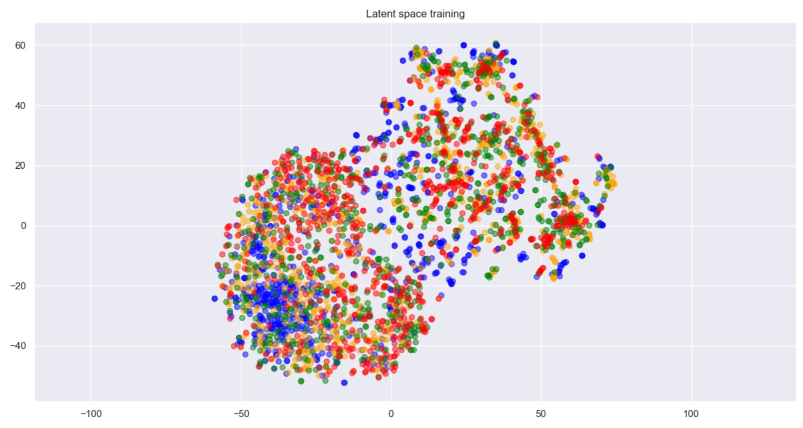


Figure 5.14: Latent space of training data with trajectories of different fishing gears. The blue, orange, green and red data points correspond respectively to mechanical dredges, beam trawlers 225 kW+, beam trawlers and otter trawlers.

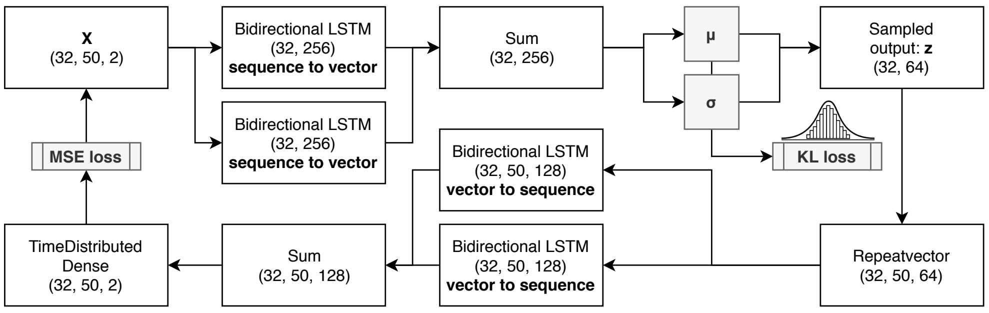


Figure 5.15: Network configuration for variational encoder-decoder model.

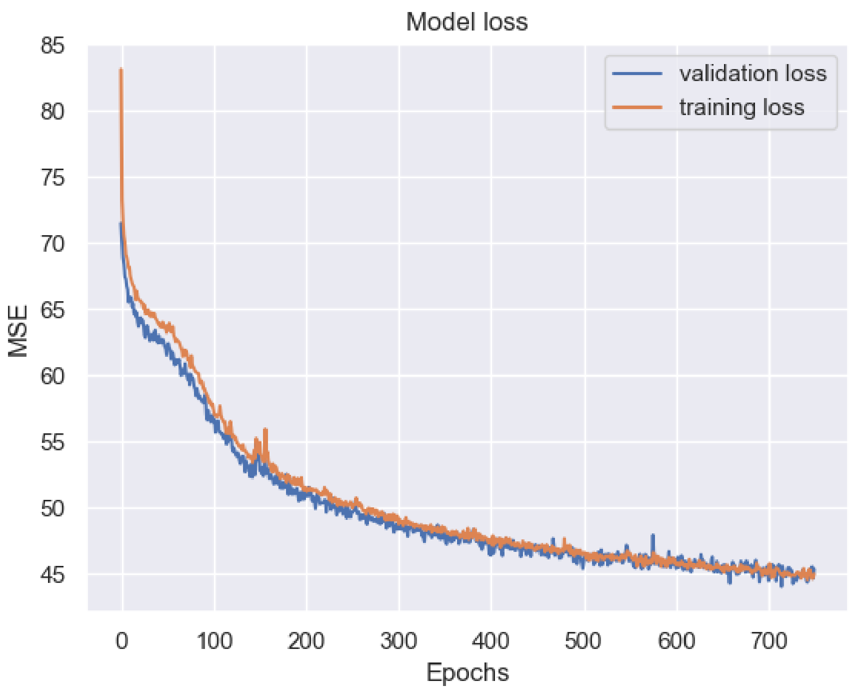


Figure 5.16: Training and validation loss for 750 epochs and batch size of 32.

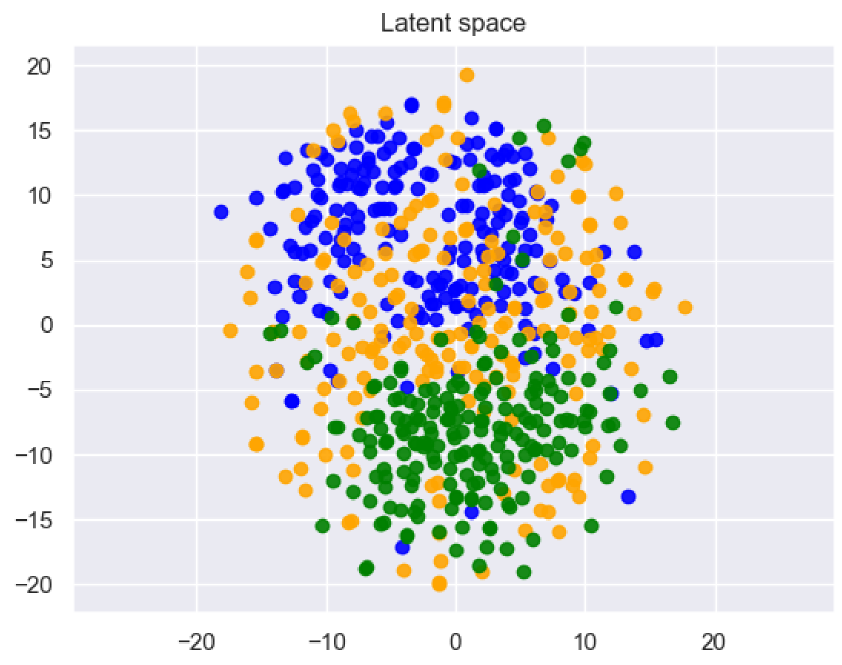


Figure 5.17: The variational latent representations of trajectories from the testing set, using t-SNE.

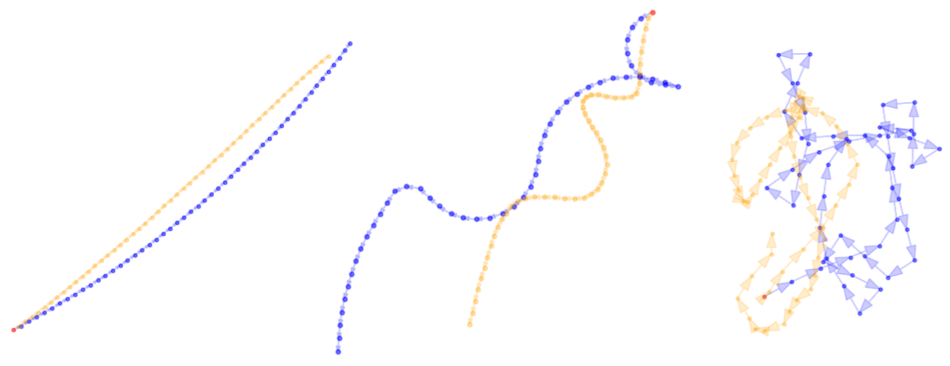


Figure 5.18: Some variations of input trajectories, were the yellow trajectory is a generated vector from the latent distribution of the blue trajectory.

6 CONCLUSION AND RECOMMENDATIONS

The main objective of this project was to explore the use of NN's to model the behavior in vessel trajectories. The main research question states *to what extent can neural networks contribute to modeling the behavior in vessel trajectories?* This question could be partitioned in different topics, such as the representation and segmentation of trajectories, different types of neural networks and their performance. The research proposed an elaborate methodology which was tested on the basis of experiments. Both AIS- and synthetic data were used to conduct these experiments. The results and an overview of the implications were provided in Chapter 5. This last chapter will first go over some of the conclusions that can be made. The main and sub questions will be used as a guideline.

Each of the following sections will discuss a single sub question. After each section, some recommendations regarding the conclusions will be made. The sub questions were formulated prior to the design of the proposed methodology. Therefore, some of the questions may be more relevant than others. Besides this—due to gained knowledge over the course of the project—some questions can not really be answered but rather guessed by logical inference. This resulted in some of the choices that were made in creating a methodology. All in all, the research questions can not be answered with the same simplicity as they were posed. Here follows a quick overview of the sub questions:

- *What is a good way to represent a trajectory for a NN and what features should be included?*
- *How should the trajectories be segmented and how should the modeling be segmented?*
- *Which type of NN is fit to model the behavior in vessel trajectories?*
- *How well do NN's perform at modeling the behavior in vessel trajectories?*

6.1 TRAJECTORIES

The data ingestion phase of this project was a really intensive and complex task to accomplish. The data was in many ways a leading factor in how the experiments were performed. Both the real world data as the synthetic data brought about their own particular difficulties and limitations.

As there is very little research on the application of NN's on trajectories, the problem of how to represent the trajectories had to be explored. NN's can be designed in a way that they are compatible to a certain data structure, but vice versa is also the case. Therefore the following research question was stated: *What is a good way to represent a trajectory for a NN and what features should be included?*.

Regarding the question of what features should be used, there is little gained from the experiments. This is because it is highly dependent on what problem is trying to be solved. The sequences can be expanded to as many dimensions that are available. Whether which ones are successful or not, is determined by the degree of correlation to the phenomenon that has to be modeled. For example, the inclusion of bathymetry data was not successful. But the reason for this is either; that it was implemented in a wrong way; that there is no correlation between bathymetry and fishing activity; or both.

The representation of spatial features is a more complex problem than the inclusion of external data. Throughout the project, the only way the networks learned properly, was by providing *dx*, *dy* vectors rather than *x*, *y* coordinates. Therefore, in combination with the logical argument that absolute space does not influence predictions, was chosen to use relative movement vectors throughout the methodology. This is counter intuitive to the concept of automatic feature design (see Section 2.1.1), because it states that DL should be able to extract features from raw data [Goodfellow et al., 2016]. The same applies to features like speed and ROT, which were all manually designed from the coordinates or already included through AIS. In this aspect, the sub question remains theoretically unanswered. Nonetheless, it can be ascertained that providing the spatial information as relative movement vectors works well from a practical stance.

As the handling of AIS data can get very complex it is desirable to propose a fully automated interface for data representation. For the experiments, the data was extracted from the database and than preprocessed for multiple times. It involved switching between frameworks like *QGIS*, *numpy* and *excel*. This is very undesirable in such a complicated problem. Ideally, there should be a central application which lets data flow from the database to the NN. This application should offer various options for how to represent the data. Including outlier removal, feature selection, rolling window statistics, spatial filters, normalization, etc. This interface should talk directly to the database. It can also save memory if it makes use of the data generator class that *keras* offers. This would simplify things by a lot and saves time for conducting the experiments.

Some problems regarding the synthetic data were already pointed out in this paper. The main recommendation is to include a distance in the relative movement vectors. Now, the synthetic data can be reduced to one dimension. It is also preferable to make the trajectories more complex and longer. Adding complexity to the trajectories could also result in more classes. Of course, this recommendation only holds when training with synthetic data is the only option. Training with real data—either unsupervised or supervised with proper labels—is the best way to further test this methodology.

6.2 SEGMENTATION

In some respects, the segmentation problem closely coheres with the data representation. As seen before, some ways of representing the data put a constraint on how segmentation should happen. Right from the start, decisions had to be made about the length of the trajectories. This task brought about the almost philosophical question of what a trajectory actually is. From a practical point of view, this was formulated as the following sub question: *How should the trajectories be segmented and how should the modeling be segmented?*

The first term of the sub question focuses on how a trajectory is defined and used as an input for a NN. The last term focuses on how the output should be related to the input. For example, it could be desirable to give the network multiple small trajectories of one vessel and get back one classification per trajectory. It could also be desirable to give the network one continuous trajectory of one vessel and get back a pointwise classification. Both methods sort of solve the same problem. What was found in this project is that there is not one 'best' way of segmenting. Again, it depends on the type of problem that needs to solved. Therefore, the proposed taxonomy in Figure 3.1 was made.

Instead of choosing between sub-trajectory and within-trajectory classification, both were implemented. Within-trajectory classification solves the problem of segmentation, because segmentation is inherent to the training process. It however lacks the ability of unsupervised learning (to the best of my knowledge). Sub-trajectory classification solves this, because individual trajectories of predefined length can be projected onto latent space. The supervised variant of sub-trajectory classification has the same holistic approach, the only difference is that labeled data is available.

To summarize, there is no correct way to deal with the segmentation. If the problem allows it and labeled data is available, let the NN do the segmentation through within-trajectory classification. If this is not possible, domain knowledge has to be taken into account. The behavior that needs to be modeled should be present in the whole trajectory. Otherwise, the model will look for lower-level behavior.

6.3 NEURAL NETWORKS

In Chapter 2 was already stated that different NN's exist for different types of information. Different possibilities were therefore to be explored for the methodology. One of the sub research questions therefore stated: *which type of NN is fit to model the behavior in vessel trajectories?*

This question can only be answered in combination with the research question from Section 6.1. Specifically, the data representation strongly suggests what type of network should be used. If the data representation is a raster image, the most logical decision would be to use a convolutional NN. If the inputs are simple vectors, a MLP is the best choice. Consequently, because the trajectories were treated as sequences the chosen networks were all RNN's.

It can not be ruled out that other types of NN's are also fit to model behavior in vessel trajectories. However, the RNN most certainly seems like the most obvious candidate for further research. Some promising results were presented in Chapter 5, acknowledging the potential of this type of network. Nonetheless, only LSTM networks were used. There also exist other types of RNN's, like gated recurrent units and even convolutional LSTM networks. These networks may be interesting to explore in the future.

An idea is to remove arbitrary amounts of coordinates from the trajectories to let networks train with data gaps. This experiment could tell something about the significance of spatiotemporal discontinuous trajectories. By adding more dimensions that resemble the real world data, better conclusions can be made.

6.4 PERFORMANCE

One of the most important sub questions states: *how well do NN's perform at modeling the behavior in vessel trajectories?* To answer this question, the experiments are treated individually first.

The first experiment that was conducted was the detection of fishing activity. Because of issues with the labeling of the data this experiment is a bit suspicious. Almost all metrics score well above 0.85, which is a good performance for this problem. But this is most probably the case because the labeling process was very simplistic. This experiment would most certainly get more interesting with real labeled data (e.g. from sensors on fishing gear). On the other hand, it is notable that training this network with only ROT works well. But this is rather an insight for fisheries studies than for this project.

The supervised sub-trajectory experiment with synthetic data scored very high in performance. Although there were some problems with the synthetic data, this performance shows that simple networks can model spatial behavior. In the same way, the unsupervised experiments showed clear clusters of behavior.

Certainly the experiment with a variational encoder-decoder model was interesting. Figure 5.18 shows convincing imitations of class trajectories. It is not yet clear what this generative aspect could contribute to modeling behavior in vessel trajectories. It is recommended to further look into what could be possible applications for this tool. One suggestion is that it can be used to create realistic simulations, generating variations of real vessel journeys. It might be of use in the military, for spoofing convincing signals of moving objects in general. Generative adversarial networks could also be explored for these kinds of applications.

Besides these well performances, the regular unsupervised experiment with real data that was discussed in Section 5.5.1 did not get very good results. But the network could still be ran for more epochs to check for improvement. Therefore, all things considered, it can be carefully concluded that NN's perform reasonable at modeling behavior in vessel trajectories.

Finally, the experiments with real data were more complex that the ones with synthetic data. Because there was more uncertainty in the real data. Numerous times networks were trained to no avail, because of mistakes in the input data. But the evaluation does not grant this information. The validation and test metrics are the only handles in adjusting the network or input data to reach better results. If a network does not learn, many things can be wrong with it. During the experiments it was always firstly assumed that there is something wrong with the input data, rather than with the network parameters.

Regarding some of the experiments with real data, better hardware with larger networks could be tried. Figure 5.14 already shows some clustering. Increasing the trajectory lengths and expanding the network could lead to a better result. Instead of regular processors, GPU could be exploited for future experiments. This could be useful because the tests can be conducted faster this way. If you have to wait for 10 hours each time a network has to run, it can be tedious to rationalize the effects of the hyperparameters. Also, it just limits the amount of tests that can be run. Furthermore, it is highly advisable to keep a structured logbook of all hyperparameters in combination with their results.

6.5 IN GENERAL

Because the topic is new and not yet thoroughly researched, there are still many directions to head. In general, the contribution of NN's to modeling the behavior in vessel trajectories has been found to be positive. In this fashion, it is advisable to push forward some of the experiments that were done in this project.

While formulating the methodology it has been found that behavior can be modeled within a trajectory or between different trajectories. These are two different problems which can be solved by using DL. While doing this, it is best to treat the trajectories as sequences. At this time, the best NN to process sequences is a recurrent NN. Therefore it can be concluded that LSTM networks are excellent candidates for the job. In fact, by looking at some of the results, it can even be concluded that LSTM networks certainly have potential for modeling the behavior in vessel trajectories.

With respect to segmentation, various conclusions can be made. First of all, NN's can be leveraged to deal with segmentation in within-trajectory classification. Second, while dealing with sub-trajectory classification, the segmentation needs to be done in a way that the segments encompass the targeted behavior. If the periodicity of the behavior is one day, it is recommended to create segments of at least one day. The longer the segments are, the more they will encompass high-level behavior. The shorter they are, the more they will encompass low-level behavior. As of now, this is the only rational justification to choose a length for the trajectories.

It is recommendable to further test the supervised methods of the methodology using data with reliable labels. If these labels are not available, it is not realistic to proceed training in a supervised setting. It could be interesting to treat this problem as a specific side-project. Hereby the main question should be how it would be possible to compile structured, reliable and labeled data sets for the purposes of applying DL to vessel trajectory data. This task should always involve domain experts, to monitor the compilation process and if necessary label the data themselves.

In the famous review 'Deep learning', LeCun et al. [2015] states: 'Unsupervised learning had a catalytic effect in reviving interest in deep learning, but has since been overshadowed by the successes of purely supervised learning. [...] we expect unsupervised learning to become far more important in the longer term. Human and animal learning is largely unsupervised: we discover the structure of the world by observing it, not by being told the name of every object'. If this expectation is true, it would be desirable to continue this research in two branches, a supervised and an unsupervised branch. This way, the focus can be directly pointed at each branch, without interference of its counterpart.

6.6 EPILOGUE

This section is included to reflect on the graduation phase. It is a brief looking back upon the way how was dealt with the project, in relation to the Master of Geomatics, the field of geomatics in practice and society. This is a separate section because I would like to express myself informally.

I started demarcating the problem of this thesis well in advance, about half a year prior to the official start. The reason for this is that demarcation is not one of my strong abilities. I thought that establishing the main problem and pinning it down would avoid later complications with conceptual rumble. A large proportion of my justification for the problem was societal relevance. The idea that research actually contributes to a contemporary problem is very attractive.

There was however a logical error in my motives. I formulated the problem, but had already predetermined how I was going to solve it: by the use of DL. This is wrong, because when a problem is formulated you should look at many different ways to solve it and then choose the best solution. During the process, I gradually came to see that my reasoning was flawed. Eventually, I found out that the problem was actually DL itself, or at least the lack of integration of it with movement data. From then on, my focus was shifted to that direction. The fishing problem was not completely abandoned as it could be used as a case study.

DL is becoming more and more an established field. This clears to air to explore its potential merits in other disciplines. I think geomatics should be one of those disciplines. Large amounts of geographical data, in great need of being analyzed is indeed fertile ground for DL. The only question that remains is how to do it? I think DL should be seen as a tool—like k-D trees, interpolation or spatial operations—which has well proven itself being very effective at solving some hard problems in other fields. On the other hand, DL as a hype presents itself as the solution to all of our problems. Just like all other algorithms in geomatics, DL has only a limited set of problems assigned to it which it can solve.

Many concepts in DL are actually borrowed from regular statistics and labeled with jargon. Some of these concepts were treated during the first year of the master of Geomatics. However, I think that it would be better if DL had a bigger seat at the table. Not particularly in combination with movement data, as the topic is in its infancy still. But the strength of DL could easily be demonstrated in an image classification task like landslide detection. This is clearly a problem where DL could have potential superiority over traditional methods.

All in all, this project was a nice opportunity to explore uncharted waters. As a person who loves to learn new things and work practically, this graduation process was very energizing at points in time. But as a person who finds it difficult to highlight the common thread and shape an ongoing story, this project was much of a learning experience. I think of it as the ultimate exercise, of which the output will be of great benefit for further shaping my career.

BIBLIOGRAPHY

Adolph, K. E., Cole, W. G., Komati, M., Garciaguirre, J. S., Badaly, D., Lingeman, J. M., Chan, G. L. Y., and Sotsky, R. B. (2012). How do you learn to walk? thousands of steps and dozens of falls per day. *Psychological Science*, 23(11):1387–1394. PMID: 23085640.

Andrienko, N., Andrienko, G., Pelekis, N., and Spaccapietra, S. (2008). Basic Concepts of Movement Data. In *Mobility, Data Mining and Privacy*, pages 15–38. Springer Berlin Heidelberg, Berlin, Heidelberg.

de Souza, E. N., Boerder, K., Matwin, S., and Worm, B. (2016). Improving Fishing Pattern Detection from Satellite AIS Using Data Mining and Machine Learning. *PLOS ONE*, 11(7):e0158248.

ESA (2019). Satellite – Automatic Identification System.

Frentzos, E., Gratsias, K., Pelekis, N., and Theodoridis, Y. (2007). Algorithms for Nearest Neighbor Search on Moving Object Trajectories. *GeoInformatica*, 11(2):159–193.

Gaffney, S. and Smyth, P. (1999). Trajectory clustering with mixtures of regression models. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '99, pages 63–72, New York, NY, USA. ACM.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Graser, A. (2019).

IMO (2013a). Regulations for carriage of AIS.

IMO (2013b). Solas Chapter V - Annex 17 - Automatic Identification Systems (AIS).

Jiang, X., Silver, D. L., Hu, B., de Souza, E. N., and Matwin, S. (2016). Fishing Activity Detection from AIS Data Using Autoencoders. pages 33–39. Springer, Cham.

Kroodsma, D. A. and Sullivan, B. (2016). Protecting marine World Heritage from space. In *UNESCO, THe Future of the World Heritage Convention for Marine Conservation*, pages 35–47. Paris.

Kröse, B. J. A. and van der Smagt, P. P. (1991). *An Introduction to Neural Networks*. The University of Amsterdam, Amsterdam, The Netherlands, fourth edition.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521:436–44.

Li, H., Liu, J., Liu, R., Xiong, N., Wu, K., and Kim, T.-h. (2017). A Dimensionality Reduction-Based Multi-Step Clustering Method for Robust Vessel Trajectory Analysis. *Sensors*, 17(8):1792.

Li, X., Han, J., and Kim, S. (2006a). Motion-Alert: Automatic Anomaly Detection in Massive Moving Objects. pages 166–177. Springer, Berlin, Heidelberg.

Li, X., Han, J., and Kim, S. (2006b). Motion-Alert: Automatic Anomaly Detection in Massive Moving Objects. pages 166–177. Springer, Berlin, Heidelberg.

Mascaro, S., Nicholso, A. E., and Korb, K. B. (2014). Anomaly detection in vessel tracks using Bayesian networks. *International Journal of Approximate Reasoning*, 55(1):84–98.

Miller, N. A., Roan, A., Hochberg, T., Amos, J., and Kroodsma, D. A. (2018). Identifying Global Patterns of Transshipment Behavior. *Frontiers in Marine Science*, 5:240.

NOAA (2018).

Olah, C., Mordvintsev, A., and Schubert, L. (2017). Feature visualization. *Distill*.

Raymond, E. S. (2016). AIVDM/AIVDO protocol decoding.

Robards, M., Silber, G., Adams, J., Arroyo, J., Lorenzini, D., Schwehr, K., and Amos, J. (2016). Conservation science and policy applications of the marine vessel Automatic Identification System (AIS)—a review. *Bulletin of Marine Science*, 92(1):75–103.

Rosenblatt, F. F. (1963). Principles of neurodynamics. perceptrons and the theory of brain mechanisms.

Russo, T., Parisi, A., Prorgi, M., Boccoli, F., Cignini, I., Tordoni, M., and Cataudella, S. (2011). When behaviour reveals activity: Assigning fishing effort to métiers based on VMS data using artificial neural networks. *Fisheries Research*, 111(1-2):53–64.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*.

Tang, W., Pi, D., and He, Y. (2016). *A Density-Based Clustering Algorithm with Sampling for Travel Behavior Analysis*, pages 231–239.

Wang, W., Ramesh, A., and Zhao, D. (2018). Clustering of Driving Encounter Scenarios Using Connected Vehicle Trajectories.

Yao, D., Zhang, C., Zhu, Z., Huang, J., and Bi, J. (2017). Trajectory clustering via deep representation learning. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 3880–3887. IEEE.

Zeiler D., M. (2012). Adadelta: An adaptive learning rate method. 1212.

Zheng, Y. and Yu (2015). Trajectory Data Mining. *ACM Transactions on Intelligent Systems and Technology*, 6(3):1–41.

A EXAMPLE CODE

This appendix demonstrates some of the code that was used during the project. The code snippets are presented in order of execution. To start, these are all of the imports used for data preprocessing:

```
from psycopg2 import connect # connection to database
from random import sample # for sampling from trajectory
from scipy.signal import medfilt # for median filter

import numpy as np # numerical operations
import matplotlib.pyplot as plt # for plotting
import seaborn as sns # for enhancement of plots

import json # to store dictionaries as JSON objects
import csv # to open csv files
import math # mathematical operations
import os # to walk directories
```

The first step was to query the occurrences of vessels per type in the static table:

```
query = """SELECT shiptype, COUNT(DISTINCT mmsi) cnt FROM ais_logs_static GROUP
    BY shiptype ORDER BY cnt DESC;"""
```

From this query, the desired vessel types can be chosen. Then, all MMSI numbers per type are exported to a JSON file in the following format:

```
mmsis = {
   'cargo': [477195700, 244730047, ...],
   'tanker': [477904500, 244620978, ...],
   '...': [...]
      }
```

This file is used in the following step to store the amount of coordinates per MMSI number as another JSON file, here *to_disk*. This time:

```
with open('mmsis.json', 'r') as fh:
   mmsis = json.loads(fh.read())

to_disk = {}

for key in mmsis:
   query_format = """SELECT COUNT(mmsi) FROM ais_logs_dynamic WHERE mmsi = """

   for mmsi in mmsis[key]:
      query = query_format + str(mmsi)

      with conn.cursor() as cur:
         cur.execute(query)
         contents = list(cur)

      if contents:
         count = contents[0][0]
      else:
            continue
        to_disk[mmsi] = count
```

I used the two produced JSON files—respectively *mmsis* and *count*—to plot the distribution of coordinate occurrences per vessel type. To do this, I used this simple script:

```
def view_histo(_type):
    data = [count[str(_)] for _ in mmsis[_type]]
    sns.distplot(data, bins=20, kde=False, hist_kws={'cumulative': True})
    plt.title('Trajectory length cum. distribution {} vessels'.format(_type))
    plt.xlabel('Trajectory length')
    plt.ylabel('Occurences')
    plt.show()


def view_plot(_type):
    data = [count[str(_)] for _ in mmsis[_type]]
    data.sort()
    data = np.array(data)
    points = []

    for i in range(0, 120000, 2000):
        length = len(np.where(data > i)[0])
        points.append(length)

    plt.ylabel('Trajectory length')
    plt.xlabel('Number of trajectories larger than y')

    sns.scatterplot(x=points, y=list(range(0, 120000, 2000)), label=_type)


for _type in ['cargo', 'tanker', 'fishing', 'tug', 'passenger']:
    view_plot(_type) # or
    # view_histo(_type)

plt.legend()
plt.show()
```

The information on the plots were used to look at the cumulative distributions, the lower bound decides how many samples can be taken. At least N trajectories in each class which contain more than 40000 coordinates were selected. Another JSON file had to be created to subdivide the data into bins of the numbers of coordinates:

```
final_selection = {
    "40000": {          # coordinate bin (from 40000 to 50000 coordinates)
        "0": [          # vessel type
            [
                244660210, # mmsi
                52994      # number of coordinates
            ],
            [
                244700712,
                51120
            ],
            ...            # other mmsi's
        ]
        "...": [...]     # other vessel types
        }
    "...": {...: ...}   # other coordinate bins
}
```

Statistics on this file were used to create a partition for training, validation and testing whilst preserving an even distribution along the bins and the vessel types. Also, those 3 partitions contain distinct MMSI numbers. But first, the samples had to be extracted from the database and saved as binary files. To do this, a lookup table for data types per table column was created:

```
dtype_lookup = {'longitude': float,
                'latitude': float,
                'ts': np.datetime64,
                'speed': float,
                'turn': float}
```

This table was used in the following function, which is a generator function that yields a number of samples of a certain length from a specific MMSI number:

```
def extract_samples(mmsi,
                    length,
                    n_samples,
                    l_samples,
                    features=['longitude', 'latitude']):

   # create query to retrieve data
   query = """SELECT {0}
           FROM ais_logs_dynamic
           WHERE mmsi = {1}
           ORDER BY ts ASC""".format(', '.join(features), mmsi)

   # create indices for sampling
   indices = range(length - (l_samples - 1) * n_samples)
   offset = 0
   dim = (length,)
   dtype = [(f, dtype_lookup[f]) for f in features]

   # create cursor
   with conn.cursor() as cur:
      cur.execute(query)
      array = np.recarray(dim, dtype=dtype)

      # create array
      for i, _tuple in enumerate(cur):
         array[i] = _tuple

      # sample from array
      for i in sorted(sample(indices, n_samples)):
         i += offset
         yield i, i + l_samples, array[i:i + l_samples]
         offset += l_samples - 1
```

Another simple function was created to save a sample in binary format. While doing so, the name of the file contains all the necessary information to plot the sample on a map if desired:

```
def save_bin(trajectory, path, mmsi, n, start, end, _class, _range):
   fname = path + '_'.join(str(_) for _ in [int(mmsi), n, start, end, _class,
       _range]) + '.npy'
   np.save(fname, trajectory)
   print('saved: ' + fname) # if logging is desired
```

The two previous function are used in the proceeding code snippet, to extract all the samples from the database and store them as binary. The number of samples extracted vary together with the bin size, these numbers were determined after some calculations (there are Ellipsis in this example for demonstrative purposes, don't copy them. Parts of these code snippets are hardcoded):

```
# open the final selection JSON file
with open('final_selection.json', 'r') as f:
   mmsis = eval(f.read())

# root to store samples
ROOT = 'data/'

N_SAMPLES = {40000: 18,
          60000: 28,
          80000: 38,
          100000: 48,
          ...: ...} # don't use this code, this shows more bins are possible

l_samples = 2000 + 1 # because later will be diffed

# for monitoring
ct = 1
total = 39770 # I hardcoded this ...

# loop through all ranges
for _range in mmsis:

   # loop through all classes
   for _class in mmsis[_range]:

      # loop through all mmsi's and lengths
      for mmsi, length in mmsis[_range][_class]:
         mmsi = int(mmsi)

         n_samples = N_SAMPLES[int(_range)]

         # loop through all generated samples
         for n, (start, end, trajectory) in enumerate(extract_samples(mmsi,
             length, n_samples, l_samples)):
            print(ct, end='/{}: '.format(total))
            save_bin(trajectory, ROOT, mmsi, n, start, end, _class, _range)
            ct += 1
```

After the raw trajectories are saved as binary files, they were preprocessed. The preprocessing consists of applying a median filter, extracting relative movement vectors and standardizing. This code was used to apply the filter and extract the relative movement vectors:

```
root = 'data/'
newdir = 'data_diffed/'

stds = [['filename', 'lon_std', 'lat_std']]

for i, fname in enumerate(os.listdir(root)):
   array = np.load(root + fname)

   array['longitude'] = medfilt(array['longitude'], kernel_size=5)
   array['latitude'] = medfilt(array['latitude'], kernel_size=5)

   lon = np.diff(array['longitude'])
   lat = np.diff(array['latitude'])

   stds.append([fname, np.std(lon), np.std(lat)])
   tbr = np.hstack((lon[:, None], lat[:, None]))
   np.save(newdir + fname, tbr)
```

Standardization was done in the following script:

```
root = 'data_diffed/'
newdir = 'data_standardized/'

means = np.array([0., 0.])
div = 0

for fname in os.listdir(root):
   traj = np.load(root + fname)
   div += len(traj)
   means += np.sum(traj, axis=0)

means /= div

stds = np.array([0., 0.])

for fname in os.listdir(root):
   traj = np.load(root + fname)
   stds += np.sum((traj - means)**2, axis=0)

stds /= div
stds = np.sqrt(stds)

for i, fname in enumerate(os.listdir(root)):
   traj = np.load(root + fname)
   traj = (traj - means) / stds
   np.save(newdir + fname, traj)
```

Now, the last JSON file containing the partitions is created in the following format:

```
partition = {
   "train": ["sample_name_containing_mmsi_etc.npy", ...],
   "validate": [...],
   "train": [...]
}
```

This file *partition.json* can be used to create a custom *keras* data generator, which lets the samples flow from a directory (in this case for classification. If the model is an autoencoder, (x, x) has to be returned):

```
ROOT = 'data_standardized/'

class DataGenerator(keras.utils.Sequence):
   def __init__(self, list_IDs, batch_size, dim, n_classes, shuffle=True):
      self.list_IDs = list_IDs
      self.batch_size = batch_size
      self.dim = dim
      self.n_classes = n_classes
      self.shuffle = shuffle
      self.on_epoch_end()

   def __len__(self):
      return int(np.floor(len(self.list_IDs) / self.batch_size))
```

```
    def __getitem__(self, index):
        indexes = self.indexes[index * self.batch_size:(index + 1) *
            self.batch_size]
        list_IDs_temp = [self.list_IDs[k] for k in indexes]
        x, y = self.__data_generation(list_IDs_temp)
        return x, y

    def on_epoch_end(self):
        self.indexes = np.arange(len(self.list_IDs))
        if self.shuffle:
            np.random.shuffle(self.indexes)

    def __data_generation(self, list_IDs_temp):
        x = np.empty((self.batch_size, *self.dim))
        y = np.empty((self.batch_size), dtype=int)

        for i, ID in enumerate(list_IDs_temp):
            data = np.load(ROOT + ID)

            x[i, ] = data
            y[i] = int(ID.split('.')[0].split('_')[4])

        return x, keras.utils.to_categorical(y, num_classes=self.n_classes)
```

This class can then be used as such:

```
from my_datagen import DataGenerator

with open('partition.json', 'r') as f:
    list_IDs = eval(f.read())

params = {
    'batch_size': 64,
    'dim': (2000, 2),
    'n_classes': 5,
    'shuffle': True
}

training_generator = DataGenerator(list_IDs['train'], **params)
```

Finally, a model can be created and used within the *keras* library. Here is an example of the code that was used for the vessel type classification problem:

```
import os
import json
import random
import numpy as np
import sys

import keras_metrics as km

from keras.models import Model, load_model, Sequential
from keras.layers import Dropout, GRU, Activation, BatchNormalization, LSTM,
    Input, Bidirectional, Dense, Conv1D, MaxPooling1D, Flatten
from my_classes import DataGenerator
from sklearn.metrics import confusion_matrix

with open('partition.json', 'r') as f:
    list_IDs = eval(f.read())
```

```python
params = {
   'batch_size': 64,
   'dim': (2000, 2),
   'n_classes': 5,
   'shuffle': True
}

training_generator = DataGenerator(list_IDs['train'], **params)
validation_generator = DataGenerator(list_IDs['validate'], **params)
test_generator = DataGenerator(list_IDs['test'], **params)

# test has to be ordered
test_generator.shuffle = False
test_generator.batch_size = 45 # has to be a factor of the total amount

# important information
features = 2
timestep = 2000

# simple model
model = Sequential()
model.add(Conv1D(filters=128, kernel_size=150, strides=5,
    input_shape=(timestep, features)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(GRU(64, recurrent_dropout=0.2)) # as faster alternative to LSTM
model.add(Dense(5, activation='softmax'))

# compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
    metrics=['acc', km.categorical_precision(), km.categorical_recall(),
    km.categorical_f1_score()])

# this prints a graphic representation of the model layers
print(model.summary())

# generators are used to fit the model
history = model.fit_generator(generator=training_generator,
                       validation_data=validation_generator,
                       epochs=300,
                       use_multiprocessing=True,
                       workers=6,
                       verbose=2)

score = model.evaluate_generator(generator=test_generator,
                         use_multiprocessing=True,
                         workers=6,
                         verbose=0)

# get the predicted and true labels
y_pred = model.predict_generator(generator=test_generator,
    verbose=0).argmax(axis=1)
y_true = np.fromiter(test_generator.get_labels(), dtype=int)

# show confusion matrix
print(confusion_matrix(y_true, y_pred))
print(score)

# save model
model.save('model.h5')
```

```
# dump training history in file to plot if necessary
with open('history.json', 'w', encoding='utf-8') as f:
   json.dump(history.history, f, ensure_ascii=False, indent=4)
```

Creating a variational encoder-decoder model was slightly more complex. Below is an example of how it could look like, using the synthetic data. This time, a custom generator was not necessary because the data was small enough to fit into memory. Also, instead of the Sequential model of Keras, the functional model was used. This allows more flexibility in the architecture of networks:

```
import os
import numpy as np

from keras.models import Model
from keras.layers import LSTM, Input, Bidirectional, Dense, RepeatVector,
    TimeDistributed, Add, Lambda
from keras import objectives
from keras import backend as K


# set up placeholders
x_train = np.zeros((3000, 50, 2)) # hardcoded dimensions
x_test = np.zeros((600, 50, 2))

# load training filenames
train_filenames = []

# retrieve all the file names from directories, this time partition is in the
    directory
root = 'data/train/'
for filename in os.listdir(root):
   train_filenames.append(root + filename)

# load test filenames
test_filenames = []

root = 'data/test/'
for filename in os.listdir(root):
   test_filenames.append(root + filename)

# load the data for training
for n, filename in enumerate(train_filenames):
   x_train[n] = np.load(filename)

# load the data for testing
for n, filename in enumerate(test_filenames):
   x_test[n] = np.load(filename)

# normalize data (hardcoded values)
x_train *= (1 / 0.7071)
x_test *= (1 / 0.7071)

# network parameters
features = 2
sequence_length = 50
originial_dim = features * sequence_length
input_shape = (sequence_length, features)
intermediate_dim = 128
latent_dim = 64
```

```python
# input layer
inputs = Input(shape=input_shape)

# LSTM layers (note bidirectional wrappers)
x_1 = Bidirectional(LSTM(intermediate_dim, activation='tanh', dropout=0.1,
    recurrent_dropout=0.1))(inputs)
x_2 = Bidirectional(LSTM(intermediate_dim, activation='tanh', dropout=0.1,
    recurrent_dropout=0.1))(inputs)

# summed layers
x_3 = Add()([x_1, x_2])

# parameter layers
mean_vector = Dense(latent_dim, activation='relu')(x_3)
variance_vector = Dense(latent_dim, activation='relu')(x_3)


# sampling trick
def sampling(args):
   mean_vector, variance_vector = args
   batch = K.shape(mean_vector)[0]
   dim = K.int_shape(mean_vector)[1]

   epsilon = K.random_normal(shape=(batch, dim),
                       mean=0., stddev=1.)

   return mean_vector + K.exp(0.5 * variance_vector) * epsilon


# latent vector
h = Lambda(sampling, output_shape=(latent_dim,))([mean_vector, variance_vector])

# input shape for decoder
latent_inputs = Input(shape=(latent_dim,))

# decoder
x_4 = RepeatVector(sequence_length)(latent_inputs)

# LSTM layers
x_5 = Bidirectional(LSTM(latent_dim, activation='tanh', dropout=0.1,
    recurrent_dropout=0.1, return_sequences=True))(x_4)
x_6 = Bidirectional(LSTM(latent_dim, activation='tanh', dropout=0.1,
    recurrent_dropout=0.1, return_sequences=True))(x_4)

# summed layers
x_7 = Add()([x_5, x_6])

# timedistributed
outputs = TimeDistributed(Dense(features, activation='tanh'))(x_7)

# create models
encoder = Model(inputs, [mean_vector, variance_vector, h])
decoder = Model(latent_inputs, outputs)
outputs = decoder(encoder(inputs)[2])
autoenc = Model(inputs, outputs)

# reconstruction loss
reconstruction_loss = K.mean(objectives.mse(inputs, outputs))
kl_loss = 1 + variance_vector - K.square(mean_vector) - K.exp(variance_vector)
kl_loss = K.sum(kl_loss, axis=-1)
kl_loss *= -0.5
```

```
# loss
loss = K.mean(kl_loss + (reconstruction_loss * 100))

# compile model
autoenc.add_loss(loss)
autoenc.compile(optimizer='adam')

# train network
history = autoenc.fit(x=x_train, batch_size=32, validation_split=0.2,
    epochs=750, verbose=1, shuffle=True)

# save model
encoder.save('v_encoder.h5')
autoenc.save('v_autoenc.h5')
decoder.save('v_decoder.h5')
```

After the network is trained correctly, the latent vectors can be retrieved and plotted in 2 dimensions. This can be done with the following script:

```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import seaborn as sns

latent = encoder.predict(x_test)[2]

X = TSNE(n_components=2).fit_transform(latent)

A, B, C = X[0:200], X[200:400], X[400:] # hardcoded

plt.title('Latent space')
plt.axis('equal')

plt.scatter(A[:, 0], A[:, 1], color='blue', alpha=0.9)
plt.scatter(B[:, 0], B[:, 1], color='orange', alpha=0.9)
plt.scatter(C[:, 0], C[:, 1], color='green', alpha=0.9)

plt.show()
```

COLOPHON

This document was typeset using $\LaTeX$. The document layout was generated using the `arsclassica` package by Lorenzo Pantieri, which is an adaption of the original `classicthesis` package from André Miede.