TECHNICAL UNIVERSITY OF DENMARK

MASTER THESIS

Anomaly Detection for AIS Data Using Deep Neural Networks for Trajectory Predictions

Marie Normann Gadeberg

supervised by

Line K. H. CLEMMENSEN
Kristoffer V. OLESEN
Sune HØRLÜCK

January 24th, 2021

Anomaly Detection for AIS Data Using Deep Neural Networks for Trajectory Predictions

This report was prepared by:
Marie Normann Gadeberg, s153382

Advisors:
Line H. K. Clemmensen, Associate Professor, Technical University of Denmark

Kristoffer V. Olesen, PhD student, Technical University of Denmark

Sune Hørlück, Specialist, Innovation Products, Terma A/S

Copyright:	Reproduction of this publication in whole or in part must include the customary bibliographic citation, including author attribution, report title, etc.
Published by:	DTU, Department of Applied Mathematics and Computer Science, Richard Petersens Plads, Building 324, DK-2800 Kgs. Lyngby `https://www.compute.dtu.dk/english`
Submission Date:	January 24th, 2021

Abstract

Maritime traffic is a dominant way of transport for carrying the worlds trade. Developing reliable systems to help monitor and control the traffic in the world seas is an important task in order to maintain the maritime situation awareness.

This thesis explores the use of deep neural networks with latent space representations of AIS messages received from cargo and tanker ships in the Danish maritime waters to detect anomalous behaviour. By using a Convolutional Autoencoder (CVAE) with the full trajectory image as input it was possible to reliably reconstruct a given route. The reconstruction probability was then used as a metric for anomalous behaviour with some success, however for trajectories of very different lengths an unwanted advantage was given to the shorter paths to pass as normal. One easy implemented solution presented here, was to use local thresholds based on journey duration. This reduced the aforementioned advantage for shorter trajectories but did not solve the problem completely.

Further, given that the model is independent of time, certain anomalies could not be detected and deploying the model in a online setting would also not be feasible.

To include time, a Variational Recurrent Neural Network (VRNN) was implemented. However, the model was unable to learn a useful latent space, postulated to be the result of a posterior collapse. The concept of a posterior collapse is introduced and various methods proposed to alleviate it are presented, none with the wanted effect for the model developed in this thesis.

Resumé

Søfart er en dominerende måde at transportere verdenshandelen på. Udviklingen af robuste systemer til at monitorere og kontrollere trafikken i verdenshavene er en vigtigt opgave for at opnå maritim situationsbevisthed.

Dette speciale udforsker brugen af dybe neurale netværk med en latent repræsentation af AIS beskeder modtaget fra fragt- og tankskibe i de danske farvande for at detektere unormal opførsel. Ved at bruge en Convolutional Autoencoder (CVAE) med billedet af den fulde rute som input, lykkedes det at rekonstruere en given rute pålideligt. Rekonstruktionssandsynligheden blev efterfølgende brugt som indikator for unormal opførsel delvist succesfuldt, dog sås det, at for ruter af varierende længde havde korte ruter en uønsket fordel. En løsning præsenteret her er at bruge lokale tærskelværdier for unormal opførsel baseret på ruternes varighed. Dette reducerede den fordel der førhen blev givet til korte ruter, men løste ikke problemet fuldstændigt.

Derudover, givet at modellen er uafhængig af tid, var der visse anomalier som ikke kunne detekteres, og det ville heller ikke være muligt at køre modellen online.

For at inkludere tid som parameter blev der implementeret et Variational Recurrent Neural Network (VRNN). For denne model var det dog ikke muligt at lære en brugbar latent repræsentation af data, hvilket postuleres at være grundet et "*posterior collapse*". Dette koncept bliver introduceret sammen med forskellige metoder, der har vist sig brugbare til at løse det. Ingen af de præsenterede metoder løste imidlertidig problemet i dette speciale.

Preface

This report has been written to fulfill the requirements to obtain a master's degree in Mathematical Modelling and Computation at the Technical University of Denmark (DTU) by stud. polyt. Marie Normann Gadeberg.

The project was assigned to a workload of 30 ECTS points corresponding to 840 working hours and spanned over a six month period from the 3rd of August 2020 to the 24th of January 2021. The data was received from Terma A/S.

[illegible]

Marie Normann Gadeberg

List of Abbreviations

AE	Autoencoder
AIS	Automatic Information System
AU	Active Units
ANN	Artificial Nerual Networks
BN	Bayesian Network
CNN	Convolutional Nerual Network
COG	Course-Over-Ground
CVAE	Convolutional Variational Autoencoder
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
ELBO	Evidence Lower Bound
EM	Expectation Maximization
GMM	Gaussian Mixture Model
GP	Gaussian Process
GRU	Gated Recurrent Units
HMM	Hidden Markov Model
KDE	Kernel Density Estimation
KL	Kullback-Leibler divergence
LSTM	Long Short-Term Memory
MI	Mutual Information
MMSI	Marine Mobile Service Identifier
MSA	Maritime Situation Awareness
PCA	Principal Component Analysis
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
ROI	Region of Interest
SOG	Speed-Over-Ground
TREAD	Traffic Route Extraction and Anomaly Detection
t-SNE	t-Distributed Stochastic Neighbor Embedding
VAE	Variational Autoencoder
VI	Variational Inference
VRNN	Variational Recurrent Neural Network

Contents

CHAPTER 1

Introduction

Maritime traffic carries around 90% of the worlds trade and is continuously growing at a rate even higher than the world economy (Tu et al. 2018). The large volume of ships makes it crucial to have proper surveillance and monitoring of the sailing routes through the vast sea areas. To instate traffic control, maritime situation awareness (MSA) is necessary, defined in (COM-EU 2009) as "*the effective understanding of activity associated with the maritime domain that could impact the security, safety, economy, or environment [...]*". The general task of predicting ship behaviour holds many use cases including navigation, intelligent collision avoidance systems (Gao, Shi, and Li 2018), optimization of traffic streams and avoidance of illegal activities (Riveiro, Giuliana Pallotta, and Vespe 2018, p. 1).

Monitoring traffic patterns are often performed at the ports by human officers but the extensive amount of data available, and the complexity it holds, makes it prone for human errors to occur (Riveiro, Giuliana Pallotta, and Vespe 2018, p. 2).

To help the operators navigate in the ever-growing data, machine learning methods and data mining has been an advancing topic in research (Cazzanti and G. Pallotta 2015).

The data available for vessel tracking can be divided into two groups; self-reporting or observation-

based. The self-reported positioning data is a voluntary broadcast, either transmitted by the vessel or sent through satellite communications, whereas observation-based positioning data is collected by active or passive sensors (Riveiro, Giuliana Pallotta, and Vespe 2018, p. 6).
Falling within the first category is the Automatic Information System (AIS) data, the data used throughout this thesis. AIS was originally developed as a safety tool used in collision avoidance as well as a general support to vessel traffic services, but was quickly after its introduction in 2002 recognized as a useful aid in the maritime situational awareness.
An AIS message is broadcast from the system periodically, and contains both static and dynamic information. The static information provides the identification of the vessel in the shape of the Marine Mobile Service Identifier (MMSI), name, flag, country, dimension and ship type.
The dynamic information on the other hand is of greater interest for modelling ship behaviour. Among others it includes the position given as longitude and latitude coordinates, the speed-over-ground (SOG) given in knots and the course-over-ground (COG) given in degrees (Cazzanti and G. Pallotta 2015).
There are multiple applications of intelligent analysis of AIS data to increase seafare safety including route estimation, collision avoidance, path planning and detection of anomalous behaviour (Tu et al. 2018) with the latter being the main topic of this thesis.

With the use of deep learning models, this thesis uses AIS messages from cargo and tanker ships in the Danish maritime water around the island of Bornholm to model ship behaviour. It is hypothesized, that any ships exhibiting behaviour difficult to characterize by the model, should be flagged as anomalous.

In Section 1.1 an overview of existing methods used for trajectory prediction and anomaly detection in the maritime domain will be introduced. Chapter 2 introduces the models and machine learning concepts used to produce the results seen in Chapter 3. Finally, a discussion of the findings will be given in Chapter 4 with the final conclusion in Chapter 5.

1.1 Related Work

Maritime anomaly detection can fall within one of two categories; *explicit* or *implicit* anomaly detection (Nguyen et al. 2019b).
The *explicit* methods use a set of rules to catch anomalous behaviour and an example of an extensive list of such rules can be found in (Kazemi et al. 2013). That list was a result of a case study in which several workshops were facilitated to formulate possible anomalous scenarios, after which members of the Swedish coastguard helped narrow down the list to 11 expert rules based on possibility from historical occurrence and degree of interest. Examples of these defined anomalies include, if a vessel A which normally goes between ports X and Y, suddenly goes to port Z or if a vessel has not left a

port according to the port schedule.

One of the main advantages of the explicit methods are the interpretability. Further, they do not rely on historical data, which makes them easily implemented. The big drawback of these methods though, is that making an exhaustive list of anomalies in the maritime domain is challenging since some parameters, like speed, are relative and makes it difficult to implement the rules across systems (Nguyen et al. 2019b; Riveiro, Giuliana Pallotta, and Vespe 2018, p. 9).

The *implicit* methods learn the anomalies from the data. Most of these methods consists of two steps; learning normal events from data, and detecting anomalies as events that differ from this normalcy (Sidibé and Shu 2020). Since the data present in the maritime domain is rarely labelled for anomalies, most methods used to extract the normal representation is unsupervised. However, modelling the trajectory of the ships can be thought of as a supervised method, since the ground truth of the predicted route is just the route itself.

Many different techniques and models have been used to try and model ship trajectories and behaviour. Some early methods tried to learn the abnormal vessel behaviour through the use of Bayesian Networks (BN). A BN is a directed acyclic graph comprised of a set of nodes connected by directed edges. Every node can take on a set of states, or variables, which must be defined (Tu et al. 2018). One of the advantages of using BNs is the explainability making them easily understood by non-experts. Expert knowledge though is easily implemented if available - a strong driver for Johansson and Falkman using Bayesian Networks for detecting vessel anomalies, as they argued that often there is not sufficient data with the characteristics needed for the task. The expert knowledge can be incorporated by including a conditional dependence relation between specific variables. They defined seven states; longitude, latitude, heading, Δheading, speed, Δspeed and object type, where Δ indicates the difference between two succesive messages. They were able to detect some anomalies on simulated data, but others were not possible to detect without also introducing a large number of false positives (Johansson and Falkman 2007).

A more complex implementation of the Bayesian Network was done by Mascaro et al. who included a large number of states including information about weather conditions and vessel interactions (Mascaro, Korb, and Nicholson 2010). They trained two models in which one took the time-series nature of the data into consideration where every time step was associated with a set of states. The other model summarised the full track as a whole. They increased the anomaly performance but showed no increased effect of adding weather information. One big drawback of the Bayesian Networks remain though, namely the sensitivity in assumptions for the defining the states.

Neural networks have also been employed in the maritime anomaly detection task at an early stage. Bomberger et al. used the *fuzzy* ARTMAP algorithm which works as a grid over a geographical area, in their case a harbor area. The junctions of the grid represents nodes in a neural network and the grid edges are the weighted connections. They evaluated a 15 minutes temporal window in which the weights between a start and end in the spatial grid was upweighted the more likely it was to occur (Bomberger et al. 2006). Also using the *fuzzy* ARTMAP algorithm, although slightly modified, was (Rhodes et al. 2005) where the normalcy of the velocity of the vessels was learned by clustering the trajectories in each square. Although the *fuzzy* ARTMAP is an unsupervised method, with relatively low computational cost, it is sensitive to the gridsize of the geographical area of interest. It is also, unlike the BNs, difficult to include static information about the vessels such as type and size (Tu et al. 2018).

The idea of dividing the region of interest into a grid and modelling the behaviour of the vessel within each grid is though widely used in literature and collectively referred to as "Geographic models" (Tu et al. 2018). Another example of this approach is seen in (Laxhammar 2008) where the normalcy in each square is found using a Gaussian Mixture Model (GMM) and the Expectation-Maximazation (EM) algorithm to cluster momentary location, speed and course of the vessels. The results showed that most anomalies detected were vessels crossing sea lanes or travelling in the opposite direction of sea lanes. To find more complex anomalies, they argue that more sophisticated models should be used. It is also, as noted by (Kowalska and Peel 2012), a non-trivial task finding the optimal number of mixing components for the GMM and can often lead to overfitting. Instead of finding anomalous tracks as most methods introduced so far, Kowalska et al. use point-based anomaly detection. They use two separate Gaussian Processes (GP) to predict the velocity in the horizontal and vertical direction respectively, which combined gives them the predicted velocity at an unseen position. The sum of the log-likelihood of the point in the trajectories can then indicate whether the track is behaving abnormally or not. The Gaussian Process is contrary to GMMs non-parametric, albeit like the GMMs have a high computational cost and poor scalability (Tu et al. 2018). In (Kowalska and Peel 2012) this is addressed by Active Learning with some success.

For both the *fuzzy* ARTMAP and the GMMs some form of clustering of the trajectories is used. Kowalska et al. makes a preprocessing simplification in which they must assume only one shipping lane is present at one position, mentioning that to accommodate multiple shipping lanes in one position a clustering is necessary (Kowalska and Peel 2012). According to Nguyen et al. most methods for anomaly detection in the maritime domain contains clustering at some point (Nguyen et al. 2019a).
One of those clustering methods widely used is the DBSCAN (Density-Based Spatial Clustering of Applications with Noise). In its essence DBSCAN has two important hyperparameters; *minPts* (minimum points) and ϵ. The algorithm recognizes points as either core points, density-connected points or outliers. A point is categorized as a core point if at least *minPts* are within a distance of ϵ. Density-

connected points are then either points directly reachable from the core points (i.e. within a distance of ϵ) or reachable through another point (or points) in the cluster through a path to the core point always with a maximum distance of ϵ. Density-connected points are then said to belong to the same cluster, and any points not belonging to a cluster is defined as noise (Giuliana Pallotta, Vespe, and Bryan 2013).
The DBSCAN was one of the key components in the method proposed by Palotta et. al which they named TREAD (Traffic Route Extraction and Anomaly Detection). Here they focus on a pre-specified area and evaluate the events from the AIS messages from the vessels entering the area of interest. They use the clustering of these events to create waypoints (e.g. stationary points, entry points, exit points) which they then link to create a statistical representation of a route (Giuliana Pallotta, Vespe, and Bryan 2013). The clustering is implemented in an incremental manner, and a Kernel Density Estimation (KDE) of the predicted route is used for classifying anomalous behaviour. One of the main disadvantages from the DBSCAN is much like the Baysian Networks the sensitivity of tuning the hyperparameters.

This was addressed in (Zhao and Shi 2019) where they also clustered trajectories using the DBSCAN and subsequently used a Recurrent Neural Network (RNN) to predict the route marking any deviations from the learned normalcy as anomalous. They introduced a statistical approach for estimating the hyperparameters decreasing the possible combinations of optimal choices significantly. They determine the core-distance as the distance to the closest point within the *minPts* and say that for a fixed *minPts*, ϵ should be the mode value of the inverse Gaussian Distribution. Their method though proved sensitive to the quality of the sample vessels used in training. Further, the RNN used proved difficult to implement on shorter vessel tracks such as fishing vessels.
The idea of using the minimum distance in *minPts* as the core distance is a concept of the OPTICS algorithm which in itself is a generalization of the DBSCAN algorithm trying to overcome the hyperparameter sensitivity. Le Guillarme et al. created a model composing of three steps. First they perform trajectory partitioning in which common parts of different behaviours are treated as one trajectory. They then cluster stop and move trajectories separately using the OPTICS algorithm to find distinct points of interest. Lastly, they create a two-scale normalcy model by adopting a graph structure. As in the BNs, the nodes were the states of the model and the edges were the transition between states. They modelled the transitions using a simple Markov Chain and defined the states as a local activity with the connections as a global model - hence, making it two-scale (Le Guillarme and Lerouvreur 2013).

Although the DBSCAN (with its various generalizations i.e. the OPTICS algorithm) is voluminous in literature [see also Coscia et al. 2018; Varlamis et al. 2019] other clustering methods have been imposed also. One such example is (Zissis et al. 2020) who utilized K-means clustering of trajectories. Often methods like DBSCAN is preferable to K-means as the number of clusters does not have to be defined in advance, but Zissis et al. propose simply chosing the number of clusters as the number of origin-to-destination pairs available in data.

Nguyen et al. on the other hand argues that clustering methods results in a significant loss of information and believes that continuous latent states better capture the complexity of AIS data. Further they question the fact that most state of the art methods, such as the TREAD, assume that the normalcy learned by the model is homogeneous in the total areas of interest. To address these points their proposed method *GeoTrackNet* introduces a latent space using a Variational Recurrent Neural Network (VRNN) and an *a contrario* detector subsequently allowing for geographically specific anomalies (Nguyen et al. 2019a; Nguyen et al. 2019b).

This thesis explores two models, both drawing inspiration from Nguyen et al.'s observation of increased performance after an introduction of a latent space. The first model is time invariant, handling each trajectory as an image to be reconstructed through a Variational Autoencoder (VAE). The second model is an implementation of the VRNN presented by Nguyen et al. (Nguyen et al. 2019a) and the following chapter gives a detailed description of the theory behind both.

CHAPTER 2

Theory

2.1 Convolutional Variational Autoencoder

Using deep generative models to reconstruct normal images and using the reconstruction error as a metric for detecting anomalies, has been widely applied in the medical domain. Baur et al. used deep spatial autoencoding models to capture normality of Magnetic Resonance Images (MRIs) of the brain and used the reconstruction error to detect anomalies (Baur et al. 2018). Zimmerer et al. also used variational autoencoders for reconstructing medical images using a density-based scoring in addition to the reconstruction error for anomaly detection (Zimmerer et al. 2018), and Chen et al. also used the VAEs for detecting anomalies in brain images (Chen et al. 2018).
Letting the applications in the medical domain serve as motivation, the first model to be explored in this thesis is a Convolutional Variational Autoencoder (CVAE). The CVAE is the special case of a variational autoencoder where the encoder and decoder consists of convolutional neural networks (CNN). To fully grasp the architecture of this model, the idea behind VAE is explored in the following sections as well as a definition of CNNs. But first, a gentle introduction to Neural Networks, how to

train them and the idea behind backpropagation.

2.1.1 Neural Networks, training and backpropagation

The idea behind Artificial Neural Networks (ANN) originated in 1943 where McCulloch and Pitts, tried to model the information processing of neurons in the human brain (McCulloch and Pitts 1943). The models have since increased in sophistication but the general inspiration from the human brain remains, with the ANNs also consisting of neurons processing information when activated.

The derivations in this section follows those in Bishop 2006, p. 227-244 and Herlau, N. Schmidt, and Mørup 2018, p. 200-205. To simplify notation and derivations the network considered here is a simple feed-forward neural network with an input layer, one hidden layer and an output layer as illustrated in Figure 2.1. The network is categorized as a feed-forward NN because information only flows from the input x, through the hidden layer, to the output y, but does not include any feedback operations in which the output y is fed back to the model. When a feedback is introduced, the neural network becomes **recurrent** which is described in further detail in Section 2.2.1.

If D denotes the number of neurons in the input layer and K the number of neurons in the output layer the neural network can be said to be a mapping such as $\mathbf{f}: \mathbb{R}^D \rightarrow \mathbb{R}^K$.

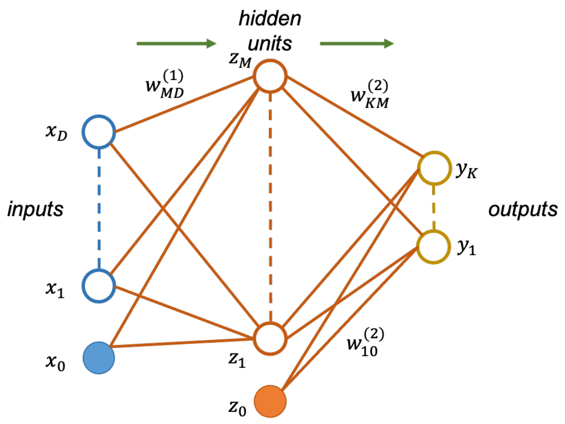


Figure 2.1: Illustration of a simple feed-forward neural network with D input dimensions and K output dimensions and one hidden layer of size M.

Initially every neuron in the first hidden layer is given an activation formulated with inspiration from the simple linear regression, denoted $a_j^{(1)}$,

$$a_j^{(1)} = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{2.1}$$

where the superscript indicates that it is the activation of neuron j in the first hidden layer and w

are the weigths and bias. Next the activation is transformed using a non-linear and differentiable activation function, g,

$$z_j^{(1)} = g(a_j^{(1)}) \tag{2.2}$$

It is the activation functions which introduce the non-linearity into neural networks and thus help increase performance. Many different activation functions exist, but for modern neural networks the rectified linear unit (ReLU) is often used. The ReLU is characterized by $g(a) = \max(0, a)$.

Now, linearly combining these activations in the same way as in Equation 2.1 gives the activations of the output layer,

$$a_k^{(2)} = \sum_{j=1}^{M} w_{kj}^{(2)} z_j^{(1)} + w_{k0}^{(2)} \tag{2.3}$$

Passing these through an output activation function g_{out} gives the output of the network,

$$y_k(\mathbf{x}, \mathbf{w}) = g_{out}\left(\sum_{j=0}^{M} w_{kj}^{(2)} g\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right) \tag{2.4}$$

where $\mathbf{x}$ and $\mathbf{w}$ are input and weight vectors respectively. The output activation function ensures a transformation of the output to a useful quantity. As an example, using a sigmoid function as the output activation would transform the output to range between 0 and 1, and is useful for classification problems, since the output can then be thought of as the probability of a given class. Please note, that the bias parameters in Equation 2.4 are included in the weight parameters by adding an extra input $x_0 = 1$ (Bishop 2006, p. 229).

When training a neural network, optimal weights are sought that minimizes some specific loss function. The general derivation of loss functions will be omitted from this thesis, and an arbitrary, smooth and continuous loss function will simply be denoted as $\mathcal{L}(\mathbf{w})$. The training then consists of finding the best value for $\mathbf{w}$ that minimizes $\mathcal{L}$,

$$w^* = \arg\min_{w} \mathcal{L}(\mathbf{w}) \tag{2.5}$$

Picturing the loss function as a surface located in weight space, a small step in weight space going from $\mathbf{w}$ to $\mathbf{w} + \lambda\mathbf{w}$ would change the loss function as,

$$\lambda\mathcal{L} \simeq \lambda\mathbf{w}^T \nabla\mathcal{L}(\mathbf{w}) \tag{2.6}$$

where $\nabla\mathcal{L}(\mathbf{w})$ indicates the direction of greatest rate of increase in $\mathcal{L}(\mathbf{w})$ and $\mathbf{w}^T$ is the transposed weight vector. Given that the loss function is smooth and continuous the smallest value will occur where $\nabla\mathcal{L}(\mathbf{w}) = 0$, unfortunately it is nearly impossible to find an analytic solution to that equation (Bishop 2006, p. 237). Instead optimization methods are applied, most of which utilizes an initialization of $\mathbf{w}^{(0)}$ and then iteratively moving through weight space such as,

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)} \tag{2.7}$$

where τ denote a given iteration. The choice of the weight vector update, $\Delta\mathbf{w}^{(\tau)}$, gives rise to various optimization algorithms. One of the traditionally most common algorithms is the *gradient descent* where $\Delta\mathbf{w}^{(\tau)}$ is the negative gradient of the loss function,

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla\mathcal{L}(\mathbf{w}^{(\tau)}) \tag{2.8}$$

where η defines the size of the step in the negative gradient direction called the *learning rate*. Often loss functions are a sum over the loss function of every example in the training set, which for very large datasets makes the computation of one gradient step become unfeasible (Goodfellow, Bengio, and Courville 2016, p. 149). To solve this *stochastic gradient descent* is introduced, which updates the weight vector based on one data-point, data-point number n, such that Equation 2.8 becomes,

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla\mathcal{L}_n(\mathbf{w}^{(\tau)}) \tag{2.9}$$

It is also possible to update the weight vector using a subset of samples called minibatches. The stochastic behaviour of stochastic gradient descent comes from the random selection of samples from the training set.

Different implementations of the stochastic gradient descent algorithm has been proposed, and in recent advances in deep learning the Adam optimization has gained traction. The algorithm was introduced in 2015 by Kingma and Ba, in which an exponential moving average of the gradient and the squared gradient are updated with hyperparameters β_1 and β_2 controlling the exponential decay rates (Kingma and Ba 2015). The Adam optimizer is used in this project.

After establishing how to optimize the weights of the network, the only outstanding is finding an efficient method to evaluate the gradients of the loss function throughout the network. To do this, backpropagation is used. To formulate the scheme of backpropagation the definitions of the simple feed-forward neural network introduced by Equation 2.1, 2.2, 2.3 and 2.4 are revisited. Let j be an arbitrary hidden neuron receiving input from activation i in a previous layer. The activation of j can from Equation 2.1 and 2.2 be described as,

$$a_j = \sum_i w_{ji} z_i \tag{2.10}$$

It is then assumed that Equation 2.10 and 2.2 has been successively calculated for all hidden and output layers in the network. Next, the derivative of the loss function with regards to the weights must be evaluated. Using the chain rule for partial derivatives gives,

$$\frac{\partial \mathcal{L}_n}{\partial w_{ji}} = \frac{\partial \mathcal{L}_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} \tag{2.11}$$

made possible since $\mathcal{L}_n$ only depends on the weights from the summed input of a_j to layer j. Introducing the notation $\delta_j \equiv \frac{\partial \mathcal{L}_n}{\partial a_j}$ and utilizing that 2.10 can be written as $\frac{\partial a_j}{\partial w_{ji}} = z_i$ gives,

$$\frac{\partial \mathcal{L}_n}{\partial w_{ji}} = \delta_j z_i \tag{2.12}$$

This shows that to evaluate the gradient, δ_j should simply be calculated for every hidden and output neuron of the network. To perform this calculation, the chain rule is once more applied giving,

$$\delta_j = \sum_k \frac{\partial \mathcal{L}_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \tag{2.13}$$

where k is the dimension of the output layer.

Backsubstituting the definition of δ_j in Equation 2.13 and utilizing the definition of activation given in Equation 2.3 gives the backpropagation formula,

$$\begin{aligned} \delta_j &= \sum_k \delta_k \frac{\partial}{\partial a_j} \left(\sum_j w_{kj} \cdot g(a_j) \right) \\ &= g'(a_j) \sum_k w_{kj} \delta_k \end{aligned} \tag{2.14}$$

The δ's at the output is simply the error for a specific neuron and hence is known. All other δ's of the network can then be found by propagating the errors backward and in combinations with the activation of the given neuron the gradients can be evaluated, thus letting the network train (Bishop 2006, p. 244).

2.1.2 Variational Autoencoder

An Autoencoder is an unsupervised method using neural networks to map a given input from the data space, X, to a latent space, Z, preserving as much information as possible while simultaneously simplifying the data (Lempitsky 2020). The mapping to latent space is referred to as the encoder. The model then seeks to inverse map the latent space back to an output with the same dimension as the input also using a neural network referred to as the decoder. This way the latent representation becomes a dimensionality reduction that can then be used as inputs in various other machine learning tasks (Joshi 2020).

Mathematically this corresponds to learning a latent representation, $z \in Z$, from a given input $x \in X$ as,

$$z = \varphi_{\phi}^{enc}(x) \tag{2.15}$$

where φ_{ϕ}^{enc} can be any neural network with learnable parameters ϕ. The decoder, φ_{θ}^{dec}, then maps $z \in Z$ back to the data space X, making the full autoencoder the composition of the two networks (Lempitsky 2020). An illustration of the autoencoder can been seen in Figure 2.2.

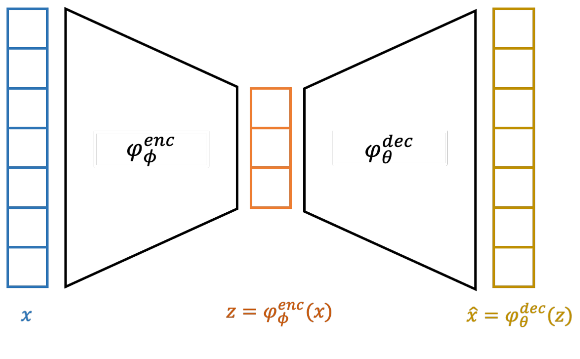

φ_{ϕ}^{enc}
φ_{θ}^{dec}
x
$z = \varphi_{\phi}^{enc}(x)$
$\hat{x} = \varphi_{\theta}^{dec}(z)$

Figure 2.2: Illustration of an autoencoder with an encoding neural network, a latent space of lower dimension than the data space and a decoding neural network.

In order to train the network a reconstruction error is simply applied measuring the dissimilarity, Δ, of the input and output as given in Equation 2.16 (Joshi 2020).

$$\mathcal{L}(\phi, \theta) = \frac{1}{N} \sum_{i=1}^{N} \Delta(x_i, \varphi_\theta^{dec}(\varphi_\phi^{enc}(x_i))) \tag{2.16}$$

where x_i denotes the i'th input of a dataset of size N.

Learning the output from the input is not necessarily useful in itself, but by setting restrictions on the latent space the aim is to capture the most important aspects of the input. Restrictions could include reducing the dimensions of Z compared to the dimension of the input. In the case of a linear decoder and mean squared error as dissimilarity measure, the latent space would learn to span the subspace as a Principal Component Analysis (PCA) (Goodfellow, Bengio, and Courville 2016, p. 500).

Since the latent space is learned directly from the training data, autoencoders often generalize poorly to new data, resulting in reconstructions from the decoder that are implausible given the data space (Cang et al. 2018). This happens because the autoencoders do not try to match the model distribution with the data distribution. Looking at the latent space as a lower dimension manifold on which the data lies and whose structure the encoder seeks to capture, this difficulty arises when the manifold is not very smooth. A very large number of training examples would then be necessary in order to model the structure of the manifold correctly, making it difficult to generalize to unseen data (Goodfellow, Bengio, and Courville 2016). Since many deep learning problems de facto work with highly complex data structures the Variational Autoencoder (VAE) was introduced as a generative model to solve this problem (Kingma and Welling 2014).

Now, instead of mapping the input to a point in latent space, the input is encoded as a distribution over the latent space, $p(z|x)$. The assumption here is, that data is generated by some random process that includes the random variable z and generating an output is then the product of the conditional distribution $p(x|z)$ (Kingma and Welling 2014). It is assumed that the latent variable, z, follows a prior distribution $p(z)$ often chosen as a simple Gaussian (Chung et al. 2015; Kingma and Welling 2014).

Compared to the autoencoder where the goal was to create an output that mapped the input the best, the VAE seeks to maximize the likelihood of x, $p(x)$, as given in Equation 2.17 (Cang et al. 2018; Doersch 2016).

$$p(x) = \int_z p(x|z)p(z)\mathrm{d}z \tag{2.17}$$

In theory it should be possible to approximate $p(x)$ by sampling a large number of samples from Z and making the approximation $p(x) \approx \frac{1}{n}\sum_i p(x|z_i)$ but in practice n would have to be very large making Equation 2.17 intractable (Doersch 2016).

However, it is possible to see that $p(x|z)$ will be close to zero for most z which gives the motivation for the encoder $p(z|x)$ that conditions the latent variable z on the input x, (Cang et al. 2018). This true posterior density function can be expressed using Bayes Theorem,

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} \tag{2.18}$$

However, this posterior function is also intractable as it requires the evaluation of $p(x)$. To overcome this problem Variational Inference (VI) is used to approximate the posterior distribution (Abiri 2019). In VI a complex distribution is approximated using optimization. In the case of the VAE, the approximation of the posterior is made by introducing the distribution $q(z|x)$, an approximate posterior. Through variational inference the distribution that best describes $p(z|x)$ is sought for. In practice this in done by minimizing the Kullback-Leibler divergence between the true posterior and approximated posterior (Blei, Kucukelbir, and McAuliffe 2017),

$$q^*(z|x) = \arg\ \min \text{KL}(q(z|x)||p(z|x)) \tag{2.19}$$

where q^* describes the optimal approximation and the Kullback-Leibler divergence is defines as,

$$\text{KL}(q(z|x)||p(z|x)) = \mathbb{E}[\log(q(z|x))] - \mathbb{E}[\log(p(z|x))] \tag{2.20}$$

where all expectations are taken with respect to $q(z|x)$.

Applying the formulation of $p(z|x)$ given by Bayes Theorem in Equation 2.18, Equation 2.20 can be rewritten as,

$$\text{KL}(q(z|x)||p(z|x)) = \mathbb{E}[\log(q(z|x)) - \log(p(x|z) - \log(p(z))] + p(x) \tag{2.21}$$

Looking at Equation 2.21 it is though clear that this also depends on $p(x)$ which once more makes the calculation intractable. Instead an alternative objective is optimized which is equivalent to the KL up to an added contant. This alternative objective equals the negative KL from Equation 2.21 plus $\log(p(x))$ which is a constant with respect to $q(z|x)$ and is referred to as the Evidence Lower BOund

(ELBO) (Abiri 2019; Blei, Kucukelbir, and McAuliffe 2017),

$$\mathcal{L}_{\theta,\phi}(x) = \mathbb{E}[\log(p_\theta(x|z))] - \mathrm{KL}(q_\phi(z|x)||p(z)) \tag{2.22}$$

where θ refers to the parameters characterizing the decoder network and ϕ is the parameters of the approximate posterior (Kingma and Welling 2014). Maximizing the ELBO is equivalent to minimizing the KL divergence in Equation 2.20 and is used as the objective function during inference when training the Variational Autoencoder (Abiri 2019; Doersch 2016).

Looking at the right hand side of Equation 2.22 the first term measures the reconstruction error between an input x and the reconstruction made from the latent variable z drawn from the conditioned latent distribution $q(z|x)$. The second term quantifies the difference between this encoding distribution $q(z|x)$ and the distribution over the latent variables $p(z)$ and acts as a regularizer (Cang et al. 2018; Kingma and Welling 2014).

The steps of the VAE then includes mapping the input to a distribution over the latent space $q(z|x)$. A sample is then drawn from this distribution, $z \sim q(z|x)$, and passed through the decoder to map back into data space, $p(x|z)$, and calculate the reconstruction error after which the weight of the network can be updated by stochastic gradient descent. All the steps are feasible through the network except backpropagating through the sampling $z \sim q(z|x)$ (Soleimany 2020). To overcome this the *reparameterization trick* is introduced.

The key idea behind the reparameterization trick is to express the random variable z drawn from the latent distribution $q_\phi(z|x)$ as a deterministic variable (Kingma and Welling 2014). This is done by expressing z as a sum of a fixed μ vector and a fixed σ vector scaled by ϵ, where ϵ is a random constant sampled from the prior distribution (Soleimany 2020). Often the approximated posterior is chosen to be a multivariate Gaussian distribution with diagonal covariance structure as,

$$q_\phi(z|x) \sim \mathcal{N}(\mu, \sigma^2 I) \tag{2.23}$$

where the μ and σ are outputs of the encoding non-linear function parameterized by ϕ. This makes the latent variable after the reparameterization trick follow,

$$z \sim \mu + \sigma \odot \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I) \tag{2.24}$$

where $\odot$ represents the element-wise product (Kingma and Welling 2014). The full flow of the Variational Autoencoder can be seen in Figure 2.3.

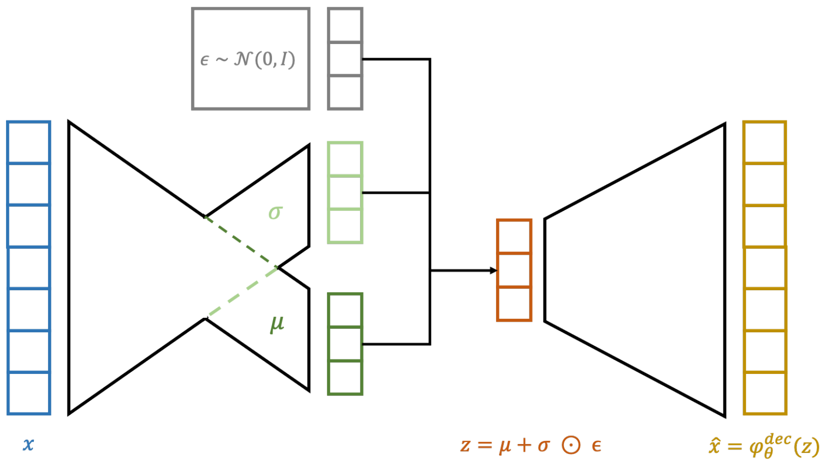

$\epsilon \sim \mathcal{N}(0, I)$
σ
μ
x
$z = \mu + \sigma \odot \epsilon$
$\hat{x} = \varphi_\theta^{dec}(z)$

Figure 2.3: Illustration of the layout for the Variational Autoencoder in which the encoder maps the input to a distribution given by μ and σ (green). The decoder then draws a sample z (orange) from the distribution made by the encoder and passes it through the decoder to create a reconstruction of x (yellow). Since backpropagation is not possible through the step of sampling, the reparameterization trick is used in which a random constant ϵ sampled from the prior (grey) is added.

2.1.3 Convolutional Neural Network

As mentioned in the introduction to this section the first model explored in this thesis is the special case of a VAE in which the encoder and decoder are parameterized by CNNs. To understand the benefits a CNN imposes remember first the fully connected neural network discussed in Section 2.1.1. Consider the case in which the input is a grid-like structure of dimensions 100 $\times$ 100, such as an image, and the first hidden layer contains 1000 neurons. Passing the image to the first layer of the feed forward network would require $100 \cdot 100 \cdot 1000 = 10^7$ weights (and an additional 1000 parameters for the bias terms). Training a neural network of this size would require a substantial computational power, and the size of the training data would have to be very large in order to tune that many parameters (Herlau, N. Schmidt, and Mørup 2018, p. 212). Convolutional neural networks address this issue explicitly by convolving the input with a kernel creating a set of feature maps. In contrary to the fully connected feed-forward neural network, the CNN have sparse interactions, accomplished by making the kernel smaller than the input.

For a 2D image input $I \in \mathbb{R}^{m \times n}$ and 2D kernel $K \in \mathbb{R}^{i \times j}$ the convolution of the two are given as,

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.25}$$

The process of convolution is commutative meaning, that $(I*K)(i,j) = (K*I)(i,j)$. This commutative behaviour is achieved by flipping the kernel as indicated by Equation 2.25, which is not generally an important feature for convolutional neural networks. Instead many libraries for convolutional neural networks use a related operation, the *cross-correlation*, which is also the case for the library Pytorch used for setting the architecture of the neural networks used in this thesis. The cross-correlation between a kernel K and image I is given as,

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \tag{2.26}$$

A given kernel can capture one kind of feature in the image such as edges or corners. For neural networks multiple convolutional operations are run in parallel in order to capture multiple features. For images the dimensions of the input is often 4D, with a dimension for the batchsize, one for the height, one for the width and lastly one for the input channels. For an image in color there will be three input channels for the red, green and blue intensity of each pixel. Grey-scale images on the other hand only have 1 input channel (Goodfellow, Bengio, and Courville 2016, p. 342).
The input kernel usually have the same dimensions as the image and the resulting convolutions in each dimension is summed to a one-depth channel output feature map.
An illustration of convolving an image $I \in \mathbb{R}^{6\times 6}$ with a kernel $K \in \mathbb{R}^{3\times 3}$ is seen in Figure 2.4. Here the kernel starts in the upper left corner and moves one step to the right until the edge of the image is reached, after which it moves back to the left edge but one step down and once more makes its way to the right edge of the image. This process is continued until the kernel has spanned the entire image.

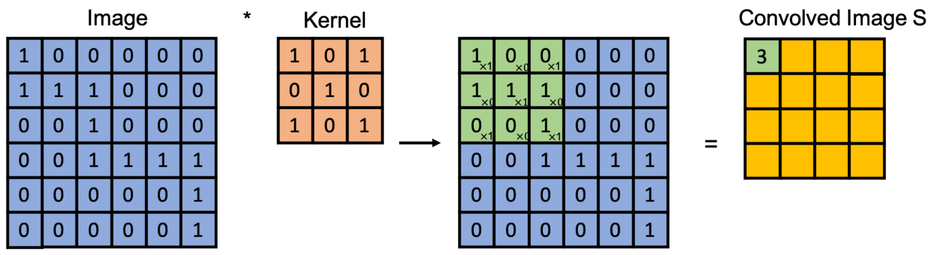


Figure 2.4: Illustration of the convolutional operation used in CNNs. The kernel (orange) is elementwise multiplied and summed with the input (blue) values in an incremental manner starting from the top left corner and moving left to right, up to down until the bottom right corner is reached, finally creating the output (yellow).

In the example in Figure 2.4 the kernel moves with one step at every calculation. The step size in which the kernel moves is referred to as the *stride* and is not limited to the size of 1 and may be different for each direction. It can be shown that any stride length different than 1 will reduce the dimensions of the data by a factor of the stride length (Rebala, Ravi, and Churiwala 2019, p. 189).

Another important feature of the convolutional operation for neural networks is the process of padding in which the width and height of the image are padded with a given number of zeros. Without zero padding the output will shrink with one pixel less than the kernel width (Goodfellow, Bengio, and Courville 2016, p. 343), as seen in Figure 2.4 where the input of dimension 6×6 results in a feature map of dimensions 4×4. This corresponds to a dimension reduction of 2 - one pixel less than the kernel size.

To overcome this, zero padding is introduced as a method to control the kernel width and the size of the output independently. In the case of the convolution seen in Figure 2.4 where no zero-padding is added it is referred to as a *valid* convolution. Since the dimensions shrink for each convolutional operation, it limits the number of convolutional layers applicable to the network. If the kernel size is large, the shrinkage can happen at a larger rate than useful information is learned by the network.

If there is instead added enough zero-padding to the input, such that the output size remains constant, it is referred to as *same* convolution. The down-side in this case is, that the pixels near the edge of the image influence fewer pixels in the output than the pixels in the center of the input does.

Finally, there is the case where enough zeros are added for every pixel to be visited as many times as the dimensions in the kernel. This is referred to as *full* convolution (Goodfellow, Bengio, and Courville 2016, p. 344).

In addition to the convolution operation creating a set of linear activations, two other processes are important in the building blocks of the CNN. After the convolution, the layers are passed through a non-linear activation such as the ReLU. This ensures that negative values are set to 0.

Lastly, the convolutions are pooled. Pooling summarizes nearby output values by a summary statistic to produce a new output of fewer dimension. The name of the pooling technique typically indicates the summary statistic applied - max pooling hence refers to passing only the maximum value in a rectangular neighborhood, and average pooling passes the average of the outputs in a rectangular neighborhood. The pooling operation ensures that the representation learned by the network become approximately invariant to small local translations in the input. Invariance is useful if it is more important whether a feature is present than where exactly it is.

In general for an input with size i, padding p, kernel size k and stride s the size of the output will be (Dumoulin and Visin 2016),

$$o = \frac{i + 2p - k}{s} + 1 \tag{2.27}$$

For image classification tasks, the output from the last convolutional layer is often flattened and passed through one or more fully connected layers creating an output with the same dimensions as categories in the data. Activating the last layer with a softmax function could then give the probability of each category.

2.1.4 Convolutional Variational Autoencoder

The CVAE is, as mentioned previously, a VAE where the encoder and decoder are characterized by convolutional networks. All the same techniques as described in the section above holds for the convolutional generative network, although the convolutional technique in the decoder is transposed. A transposed convolution is obtained by swapping the forward and backward pass of a convolution. Taking the transposed convolution of an input of size 2×2 with a 2×2 zero-padding and a kernel of size 3×3 will give an output of 4×4, which corresponds to fully connected convolution. Generally it holds, that for a convolution with stride $s = 1$, padding $p = 0$ and kernel size k there is an associated transposed convolution described by $k' = k$, $s' = s$ and $p' = k - 1$ and where the output size will be (Dumoulin and Visin 2016),

$$o' = i' + (k - 1) \tag{2.28}$$

Although the convolutional procedure can be reversed using the associated transposed convolution, most pooling functions are invertible making it unfeasible to add an inverse pooling layer in the generative network. One approach to inverse the pooling layers are the "un-pooling" method as explained in (Dosovitskiy et al. 2017). For the implementations in this thesis, no pooling operations are used in the encoding network and thus there is no need to implement inverse pooling operations in the decoder, leaving the explanation of these outside the current scope.

2.2 Variational Recurrent Neural Network

The Variational Recurrent Neural Network (VRNN) was first introduced by Chung et al. in 2015 to solve the problem of speech recognition (Chung et al. 2015) and has since been used for Machine Translation (Su et al. 2018), speech separation (Chien and Kuo 2017), and speech synthesis (Lee et al. 2018) among others. Although the inspiration and application in literature mainly stems from the field of Natural Language Processing (NLP) it has also been applied to other time series problems such as trajectory forecasting using AIS data (Nguyen et al. 2019a).

The VRNN is a Recurrent Neural Network (RNN) that contains a VAE at every step (Chung et al. 2015), so in order to grasp the full VRNN model, the concepts of a RNN must first be explored.

2.2.1 Recurrent Neural Network

The Recurrent Neural Network (RNN) is a neural network designed to deal with sequential data (Rebala, Ravi, and Churiwala 2019, p. 127). Contrary to other neural networks the RNN is dynamic and handles data that changes over time. The architecture is closely related to that of the feed-forward neural network, the only difference is, that the RNN uses the previous state of the model alongside the input to calculate the new state (Joshi 2020, p. 123).

For the feed-forward network presented in the previous section the data considered was $x \in \mathbb{R}^D$, but for the sequential data used for the RNN, data can be represented as $x \in \mathbb{R}^{T \times D}$ where T is the length of the time sequence. To simplify the derivation of the RNN, vector notation will thus be used instead of the summations over neurons as in Section 2.1.1. Using vector notation on Equation 2.4 would give,

$$y = g_{out}(W_2 \cdot g(W_1 \cdot X)) \tag{2.29}$$

where $W_1 \in \mathbb{R}^{M \times D}$ are the weights from input to first hidden layer and $W_2 \in \mathbb{R}^{K \times M}$ are the weights to the output layer.

For the RNN, consider an output at a given time as the activation of a weighted hidden state,

$$y_t = g_{out}(W_{h_2} \cdot h_t) \tag{2.30}$$

where g_{out} is the activation function of the output such as the sigmoid, W_{h_2} is the weights on the output and the hidden state h_t is conditioned on the previous state such as,

$$h_t = g_h(W_x \cdot x_t, W_{h_1} \cdot h_{t-1}) \tag{2.31}$$

The general architecture of the RNN can be seen in Figure 2.5.

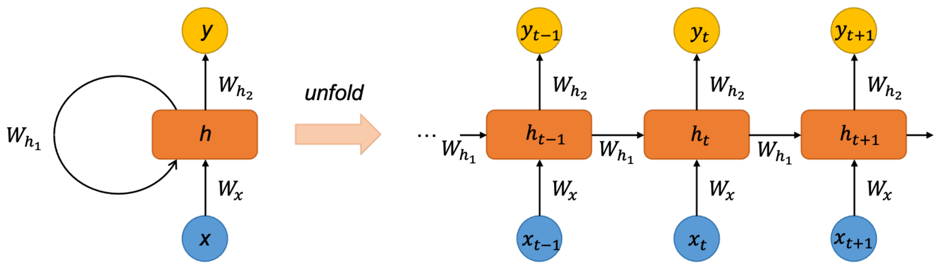


Figure 2.5: Illustration of the architecture of the Recurrent Neural Network in which the previous hidden state is passed to the current timestep to instate a short term memory of the network.

By using the previous hidden state as input in the next timestep, it enables the model to have a short term memory (Mikolov et al. 2011). The network is then trained using the backpropagation defined in Equation 2.14 (Goodfellow, Bengio, and Courville 2016, p. 378; Rebala, Ravi, and Churiwala 2019, p. 132).

One problem persisting with RNNs is the inability to store information for longer sequences. Since every prediction is made based on the last few inputs, which were also predicted by the network, it often becomes unstable, not able to rectify previous mistakes (Graves 2013, p. 2). Mathematically, this is a result of backpropagating the gradients through many stages of the network, which will often tend to make the gradients either vanish or explode.

Say the model consists of repeatedly multiplying by a matrix W, such as the weight matrices in the RNN. Then after t timesteps, this corresponds to multiplying by W^t. If W is a square matrix and has an eigendecomposition such as $W = V\Lambda V^{-1}$, where Λ is the diagonal matrix whose elements are the eigenvalues of W, the following is clear,

$$W^t = (V\Lambda V^{-1})^t = V\Lambda^t V^{-1} \tag{2.32}$$

This shows that for any eigenvalue λ_i it will either explode if it is greater than a near absolute value of 1 or contrarily vanish if it is smaller that 1.

One way to overcome this is by creating some paths through time with derivatives, that neither vanish nor explode, accomplished by the introduction of gated RNNs (Goodfellow, Bengio, and Courville 2016, p. 404). The two main gated RNNs are the Long Short-Term Memory (LSTM) (Hochreiter and Urgen Schmidhuber 1997) and the Gated Recurrent Units (GRU). For the Variational Recurrent Neural Network used in this thesis the LSTM is used.

The LSTM cell introduces three gates; the forget gate, the input gate and the output gate. The output of the LSTM is then the new hidden state and a cell state, both carried to the next sequence. The full flow through the LSTM cell is illustrated in Figure 2.6.

The forget gate determines how much of the previous state should be conserved and how much should be forgotten (Joshi 2020, p. 125). This is done by combining the current input with the previous hidden state and passed through a sigmoid as,

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \tag{2.33}$$

where W_{xf} and W_{hf} are the weights on the input and hidden state respectively for the forget state and

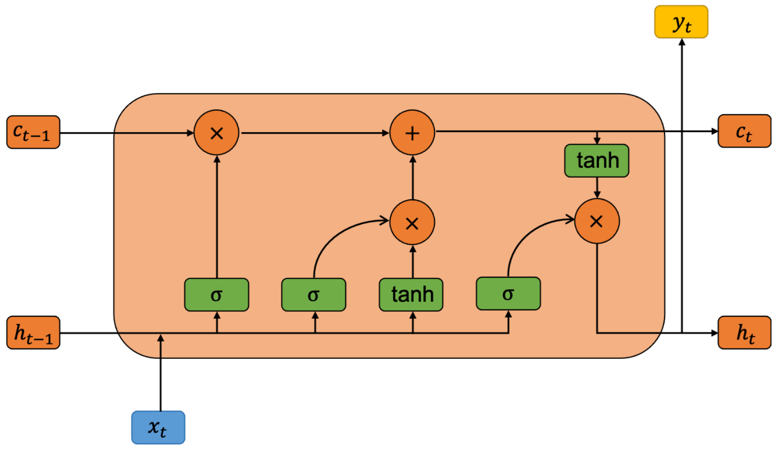


Figure 2.6: Illustration of the flow through the LSTM unit. The flow starts in the bottom left corner where the input is concatenated with the previous hidden state, passed through a sigmoid and multiplied with the previous cell state. This constitutes the *forget* gate. The *input* gate also uses the concatenated input and previous hidden state and passes them through both a tanh and sigmoid function, whereafter the outputs are multiplied. Adding the output from the input gate with the output from the forget gate gives the new cell state. The *output* gate uses the newly created cell state, the current input and previous hidden state to create the new hidden state of the model.

b_f is the bias(Graves 2013, p. 5). This notation of weights remains the same through the formulation of the gates.

Next is the input gate. This determines what new information should be allowed in the state (Joshi 2020, p. 125). Again, the current input and the previous hidden state are combined and passed through both a sigmoid and a tanh activation function after which they are multiplied together,

$$i_t = \sigma(W_{xi_1}x_t + W_{hi_1}h_{t-1} + b_{i_1}) \cdot \tanh(W_{xi_2}x_t + W_{hi_2}h_{t-1} + b_{i_2}) \tag{2.34}$$

The new cell state is then updated as follows,

$$c_t = f_t c_{t-1} + i_t \tag{2.35}$$

The output gate then uses this newly calculated cell state alongside the current input and previous hidden state to generate the output, or new hidden state, of the LSTM unit,

$$h_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \cdot \tanh(c_t) \tag{2.36}$$

2.2.2 Variational Recurrent Neural Network

The VRNN is designed to explicitly model latent dependencies over multiple timesteps while retaining a flexibility to model highly non-linear dynamics (Chung et al. 2015). The VRNN is a RNN that contains a VAE at each step and conditions the VAE on the previous hidden state, making it possible to model the dependencies of the latent random variables through multiple timesteps (Su et al. 2018). Differently from the VAE both the input and latent space are passed through feature extractors, $\varphi_\tau^x(x)$ and $\varphi_\tau^z(z)$ respectively, parameterized by simple linear neural networks. These feature extractors have proven crucial for learning complex sequences.

Moreover, unlike the traditional VAE the prior distribution is not assumed to follow a simple Gaussian but rather is conditioned on the previous hidden state of the model, formally,

$$z_t \sim p(z_t|h_{t-1}) \tag{2.37}$$

The encoder is similarly conditioned on the previous hidden state such as $q(z_t|\varphi(x_t), h_{t-1})$.

Further, the decoder is no longer only conditioned on the latent variable but also on the previous hidden state,

$$x_t \sim p(x_t|\varphi_\tau^z(z_t), h_{t-1}) \tag{2.38}$$

Finally, the RNN then updates its hidden state in the recurrence step as,

$$h_t = f(\varphi_\tau^x(x_t), \varphi_\tau^z(z_t), h_{t-1}) \tag{2.39}$$

The flow through the VRNN is formulated in four steps.

1. Computation of the conditional prior using Equation 2.37
2. Inference of the approximate posterior by sampling z from $q(z_t|\varphi(x_t), h_{t-1})$
3. Running the generating function of Equation 2.38
4. Updating the hidden state of the RNN using Equation 2.39

Like the VAE, the loss function is the ELBO, but this time it is re-written in a step-wise manner (Nguyen et al. 2019b; Su et al. 2018),

$$\mathcal{L}(x_t|h_{t-1}, z_t) = \mathbb{E}[\log(p(x_t|\varphi_\tau^z(z_t), h_{t-1})] - \mathrm{KL}[q(z_t|\varphi(x_t), h_{t-1})||p(z_t|h_{t-1})] \tag{2.40}$$

An illustration of the flow of the VRNN is seen in Figure 2.7.

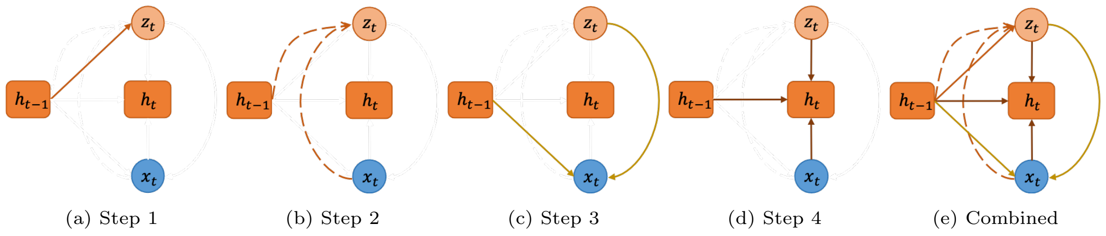


Figure 2.7: Illustration of VRNN flow. In step 1 the prior is computed conditioned on the previous hidden state. In step 2 the approximate posterior is formed conditioned on the input and the previous hidden state. A sample z is drawn from the approximate posterior and together with the previous hidden state it is used to reconstruct x in step 3. Lastly in step 4, the hidden state of the RNN is updated using the sampled latent variable z, the previous hidden state and the input.

2.3 Statistical Tools

During the evaluation of the models, a few statistical tools are used. A short theoretic overview of them will follow.

2.3.1 Monte Carlo Estimate

When training the Variational Autoencoder the ELBO is often approximated by a Monte Carlo Estimate. Monte Carlo sampling is the process of drawing samples from some distribution in order to form a Monte Carlo estimate of a given quantity. Monte Carlo sampling becomes especially useful when the sum or integral in a given calculation cannot be computed exactly, and instead lets the sum or integral be an expectation under some distribution, and then approximate the expectation by an average (Goodfellow, Bengio, and Courville 2016, p. 588). If,

$$s = \sum_x p(x)f(x) = \mathbb{E}_p[f(x)] \tag{2.41}$$

where p is a probability distribution over random variable x, then s can be approximated by drawing

n samples of x from p and calculating the empirical average,

$$s = \frac{1}{n}\sum_{i=1}^{n} f(x^{(i)}) \tag{2.42}$$

For the ELBO used in this thesis, the Monte Carlo estimate of both the reconstruction error and KL divergence is used.

2.3.2 Statistical Tests

It is often useful to be able to state whether there is a difference between the groups, or samples, of two observed variables. To this end, hypothesis testing is used. Here the mean of the samples are compared, and it is tested whether there is a significant difference between the two. Formally a *null hypothesis* H_0 is tested versus an alternative hypothesis H_1 such as,

$$\begin{aligned} H_0 &: \mu_1 - \mu_2 = \delta_0 \\ H_1 &: \mu_1 - \mu_2 \neq \delta_0 \end{aligned} \tag{2.43}$$

where μ_1 is the true mean of group 1, μ_2 is the true mean of group 2 and δ_0 is the difference in means being tested for. Often $\delta_0 = 0$ which would correspond to testing whether the mean of two distributions are equal.

Performing a level α two-sample t-test would then include calculating the test statistic, t_{obs} which follows a t-distribution with υ degrees of freedom using Equation 2.44 and Equation 2.45.

$$t_{obs} = \frac{(\bar{x}_1 - \bar{x}_2) - \delta_0}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{2.44}$$

$$\upsilon = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{(s_1/n_1)^2}{n_1-1} + \frac{(s_2/n_2)^2}{n_2-1}} \tag{2.45}$$

Here $\bar{x}$ is the sample mean and s is the sample standard deviation and n is the number of observations in the group.

If the absolute test statistic is larger than the critical value for a given significance level meaning $|t_{obs}| > t_{1-\alpha/2}$ the null hypothesis is rejected, and the alternative hypothesis can be accepted. For

$\delta_0 = 0$ rejecting the null hypothesis equates stating a significant difference between the means of the two groups investigated (Brockhoff et al. 2018, p. 172).

For this thesis the statistical tests will use a significance level of $\alpha = 0.05$ and for all evaluations the degrees of freedom renders a critical value of 1.96. An assumption for the t-test is, that the samples being investigated must be normally distributed, however the sample size for the calculations in this thesis are large enough to disregard this limitation.

CHAPTER 3

Experiments and Results

For training both models investigated in this thesis the DTU High Performance Computer (HPC) LSF 10 cluster was used. The code was written using Python 3.6 and for all neural networks the Pytorch 1.7.1 framework was used. All code used to produce the results of this thesis is available on Github: `https://github.com/mariegadeberg/AISOutlierDetection.git`.

3.1 Datasets and Preprocessing

Many of the data preprocessing steps taken in this thesis is directly inspired by (Nguyen et al. 2019a). In the paper the data used for anomaly detection was received by an AIS station located in Ushant, an island in the English Channel. Their region of interest (ROI) was a rectangle from 47.5°N to 49.5°N and 4°W to 7°W and was collected from January to March 2017, from July to September 2017 and from January to March 2018, with more than 4.2 million AIS messages in each period.
This project revolves around the Danish maritime water, and to keep the ROI as close in size to that in the paper and similar in seafare traffic type the area chosen spans from 54.5°N to 56.5°N and 13°E to 17°E. This corresponds to the area around the Danish island of Bornholm. The data is collected

from the period January to March 2020 and after preprocessing this period contains 730.000 AIS messages and is comprised of tracks from both tanker and cargo ships with a total of 11.440 individual trajectories. All paths until 10th of March were chosen as training set, paths from 11th of March till 20th of March were the validation set, and paths from 21st of March until 31st of March were the test set. This results roughly in a 8:1:1 split.

Several preprocessing steps where taken to create as homogeneous a dataset as possible. First, discontiguous voyages, identified as voyages with more than 4 hours between two successive AIS messages, were split into separate tracks. Next, paths shorter than 4 hours were discarded and paths were cut when extending beyond the ROI. As in the article the tracks were re-sampled to a resolution of 10-minutes using a linear interpolation.

To spare the RNN for very long sequences a maximum duration for a path was set to 24 hours splitting it into smaller paths if threshold was exceeded. To disregard moored vessels, tracks were discarded if the speed was smaller than 0.1 knots for more than 80% of the time. Any speeds surpassing 30 knots where set to 30 as done in (Nguyen et al. 2019a).

An illustration of some of the tracks in the dataset is shown in Figure 3.1. The processing of the inputs to the CVAE and the VRNN differ, and a detailed description of the separate methods follows.

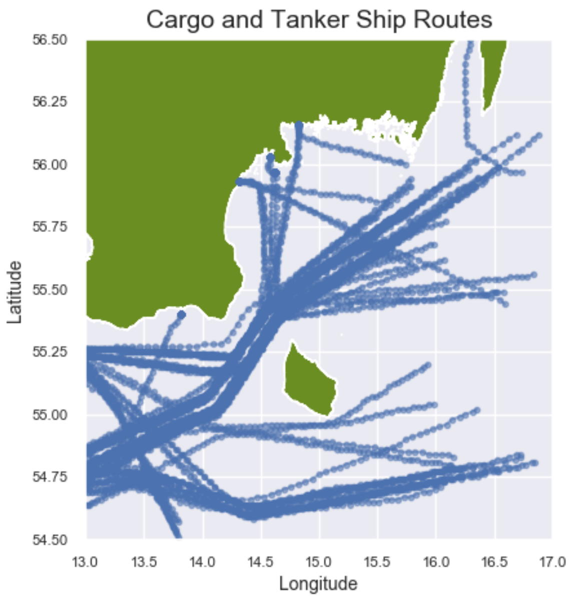


Figure 3.1: Illustration of 100 trajectories from the dataset.

3.1.1 Data Representation for CVAE model

The CVAE model takes images as input, and each path must therefore be represented as an image. This is obtained by one-hot encoding the longitude and latitude coordinates at each time-step and multiplying the two vectors to create a matrix spanning the ROI. The matrices at each time-step is then added together, creating a final image in which coordinates where the ship stayed longer have higher pixel values. An illustration of the process can be seen in Figure 3.2. Lastly, the images were normalized making all pixel values span from [0;1]. Three example inputs can be seen in Figure 3.3. The resolutions of the bins in the one-hot vector of the longitude and latitude must be included as hyperparameters. In (Nguyen et al. 2019a) they suggest a resolution of 0.01° for latitude and longitude which is also used here. This make the dimension of the input images 201 $\times$ 402 passed to the model in batches of 32.

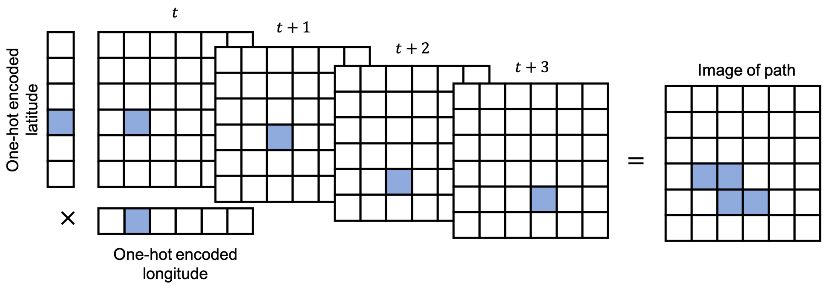


Figure 3.2: Process of generating images of the AIS messages. The one-hot encoding of the longitude and latitude coordinate at each timestep is multiplied and summed over timesteps of the trajectory to create the final image of the path.

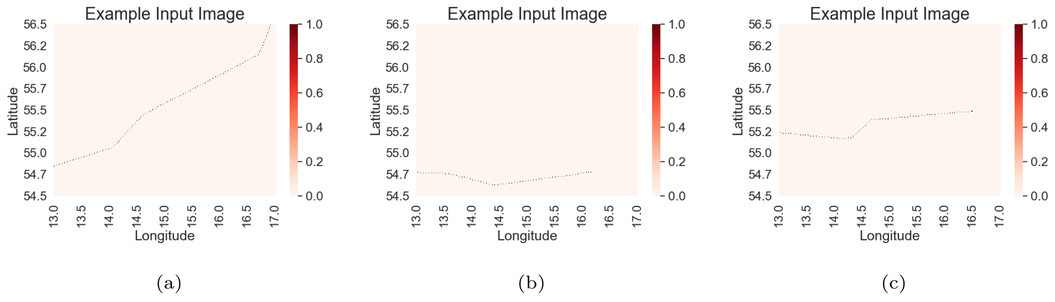


Figure 3.3: Examples of images generated as inputs for the CVAE model

3.1.2 Data Representation for VRNN model

Unlike previous work that uses the AIS messages as a timeseries where the paths are represented by its real-valued position and speed, Nguyen et al. proposes a novel representation of the AIS tracks. They instead represent each AIS point as a ”four-hot” vector concatenating the one-hot encoding of the latitude, longitude, SOG and COG of the message. They argue that this representation helps disentangle both the geometric features and the phase patterns of the tracks. Again, the resolution of the bins are included as hyperparameters. In addition to a resolution of 0.01° for latitude and longitude, they suggest 1 knot for SOG and 5° for COG which is applied here as well. These hyperparameters are a trade-off between capturing critical information without making the network so big, that it becomes too computationally expensive to run and gets prone to overfitting.

Representing the data for the project as four-hot vectors gives a dimension of 707 features. All "four-hot" vectors are normalized before being passed as input and a batchsize of 32 is used. Since the lengths of sequences passed to the LSTM must be equal, all paths in a batch are shortened to equal the length of the shortest path in batch.

3.2 CVAE

The first model to be explored for anomaly detection in the maritime domain, was the CVAE. First step was to train a CVAE to reconstruct a given track, after which anomalies could be identified as routes difficult to reconstruct by the model. Since there is no labelled data for anomalies in Danish waters, a manual inspection of the detected anomalies is necessary. However, to get a metric for the ability to detect outliers, fabricated anomalous tracks are also presented to the model, and the rate at which they are flagged as anomalies is reported.

3.2.1 Neural Network Architecture

The input to the model were greyscale images of dimension $(1 \times 201 \times 402)$, where 1 refers to the greyscale input channel. The encoder consisted of 5 convolutional layers with a stride of 2 and padding of 1 in all layers. The kernel size varied from 3 to 4 and the channels were gradually increased to end with a dimension of $(256 \times 7 \times 13)$. The input was flattened creating a tensor with a dimension of 23.296 which was passed through 2 linear layers resulting in a dimension of 200, namely 2 times the latent state size of 100. All layers were activated using the ReLU activation function.

The decoder deconvolutes the latent variable of size 100 back to the dimension of the input through first two linear layers upscaling the dimensions, then unflattening the result and passing it through 5 transposed convolutional layers with the ReLU activation function between all layers. A simple Gaussian distribution was used to parameterize both the prior and approximate posterior, whereas a multivariate Bernoulli distribution was used in the generative network. An illustration of the architecture can be seen in Figure 3.4. The Adam optimizer was used with a learning rate of 0.0003 and the Monte Carlo estimate of the ELBO was used as the objective function.

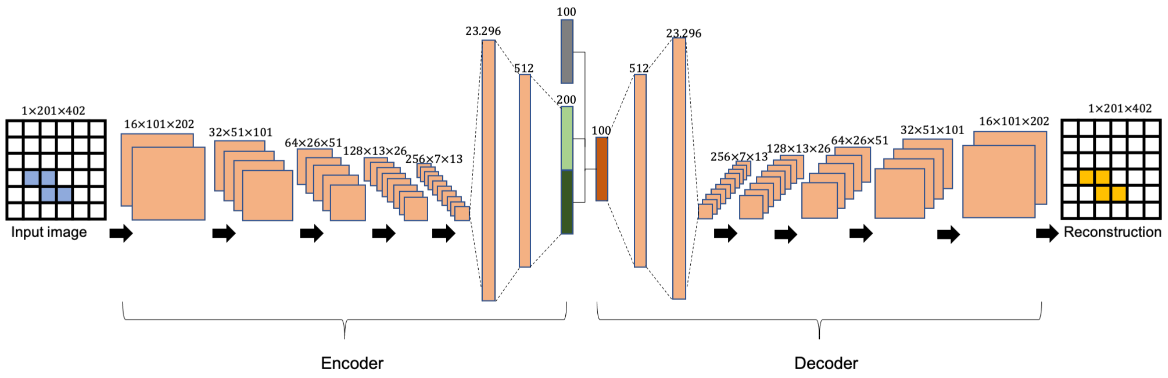


Figure 3.4: Illustration of the architecture of the CVAE model. Five convolutional layers are used to reduce the dimension of the input to 256 × 7 × 13 after which it is flattened and passed through two linear layers. The process is reversed for the decoder using transposed convolution to continuously increase dimensions until that of the input is reached.

3.2.2 Results and Analysis

Trajectory Prediction

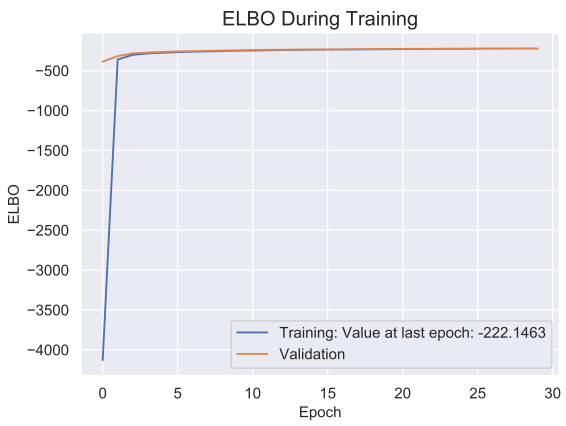
The result of training the model for 30 epochs are visualized in Figure 3.5. Here the ELBO which is sought to be maximized is illustrated (a) alongside the KL divergence (b) which is sought to be minimized. In (c) and (d) the same illustrations are shown without the first epoch to better show nuances of the training process. As expected the ELBO is maximized and the KL is initially minimized. The increase in KL after the first epoch could simply be an indicator that the model learns, that by letting the approximate posterior deviate slightly from the prior, better results are obtained.

To visualize how well the model reconstructs paths, Figure 3.6 shows the original path, the logits created by the decoder to parameterize the Bernoulli distribution and the logits transformed to probability space. Since the logits represent where it is most likely to sample a 1 from the Bernoulli distribution, Figure 3.6 shows that the decoder learns a distribution that is most likely to output 1's where the true path actually lies. The transformation to probability space is done to further illustrate this point, and performed using,

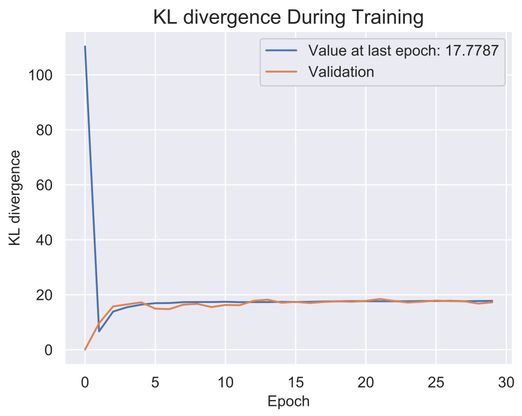
$$p = \frac{e^{(logit)}}{1 + e^{(logit)}} \tag{3.1}$$

where p indicates the probability, and e is the exponential. The complete reconstruction probability is the sum of log probabilities of every pixel in the image. For the top example in Figure 3.6 the log probability of the input is -204.18 and the bottom has a log probability of -157.11. The larger a log probability the more certain the model is of the given input.

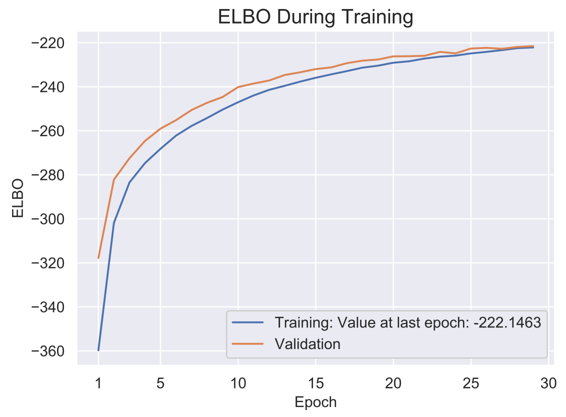
To further test the robustness of the model, the performance on corrupted paths was evaluated as well.



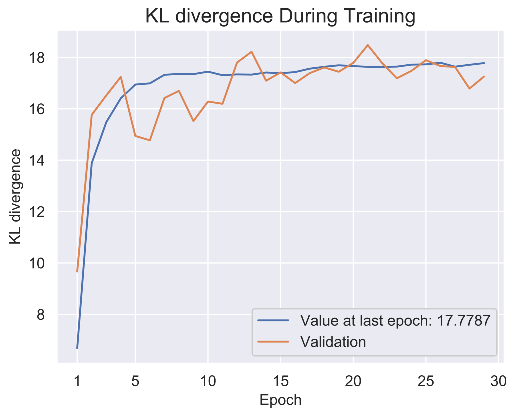
(a)



(b)



(c)



(d)

Figure 3.5: ELBO (a) and KL Divergence (b) during training for 30 epochs, with the first epoch omitted in (c) and (d). As expected the ELBO is maximized and KL minimized, although the increase of the KL after the first epoch could indicate that the model learns to let the approximate posterior deviate slightly from the prior to gain the best results.

Two different corruptions were made, one in which half the path was cut, corresponding to feeding the model with the beginning or end of a path and making it reconstruct the remaining, and one in which a random part of the path was blacked out. Examples of the two corruptions and reconstructed paths are illustrated in Figure 3.7 and Figure 3.8. Here the true paths are shown alongside the corrupted ones with corresponding logit and probability output from the model. The figures show that even though the inputs are corrupted, the probabilities of the true paths does match the location of the true paths. For the input image that has been cut, the model does not seem to predict the full length of the true input, but does goes beyond the length of the corrupted input.

To further compare how the reconstruction of the corrupted paths compare to the reconstruction of the true paths, the distribution of the summed log probabilities of the true inputs when the network is presented with corrupted paths is illustrated in Figure 3.9.

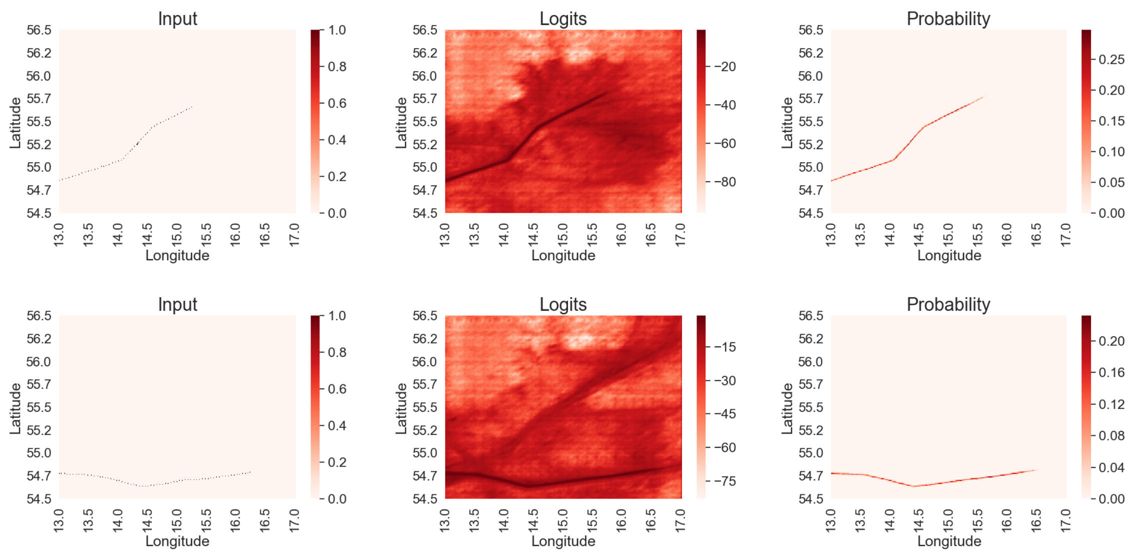


Figure 3.6: Two examples of paths reconstructed using the CVAE model. Far left shows the true paths, middle shows the logits outputted from the decoder parameterizing the Bernoulli distribution for the generative step, and far right shows the logits translated to probability space. The top example has a log probability of the input of -204.18 and the bottom examples has a log probability of the input of -157.11.

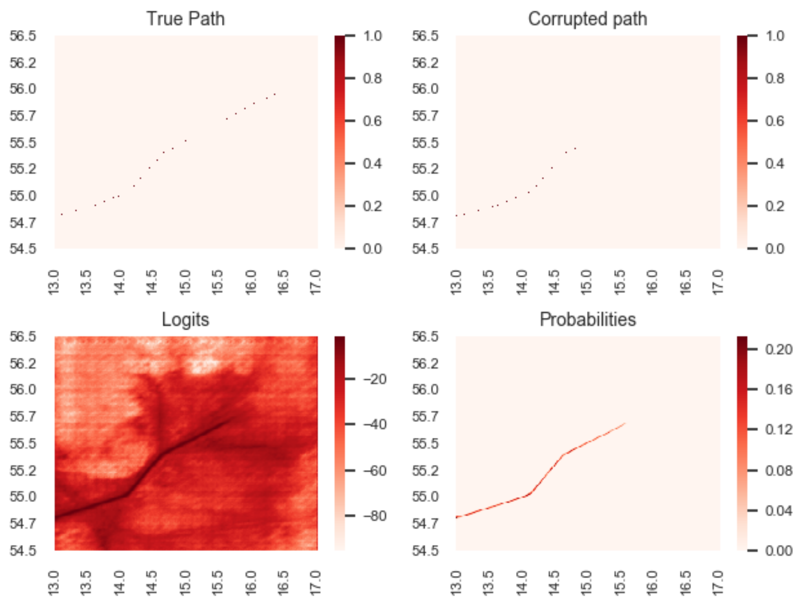


Figure 3.7: Example of an input in which half has been cut before passing to the model. Here the true path is shown alongside the corrupted path in the top of the graph. The bottom shows the logits and probability from the distribution learned by the decoder.

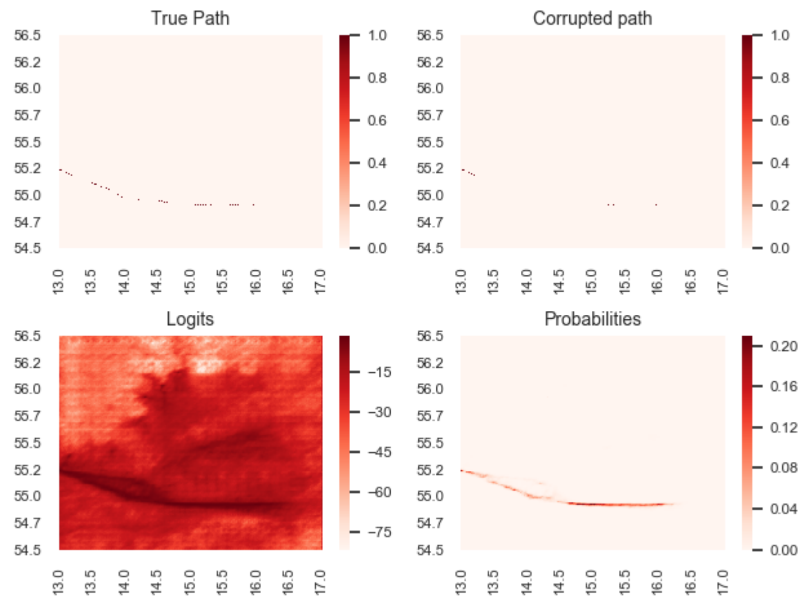


Figure 3.8: Example of an input in which part of the trajectory has been blackened before passing to the model. Here the true path is shown alongside the corrupted path in the top of the graph. The bottom shows the logits and probability from the distribution learned by the decoder.

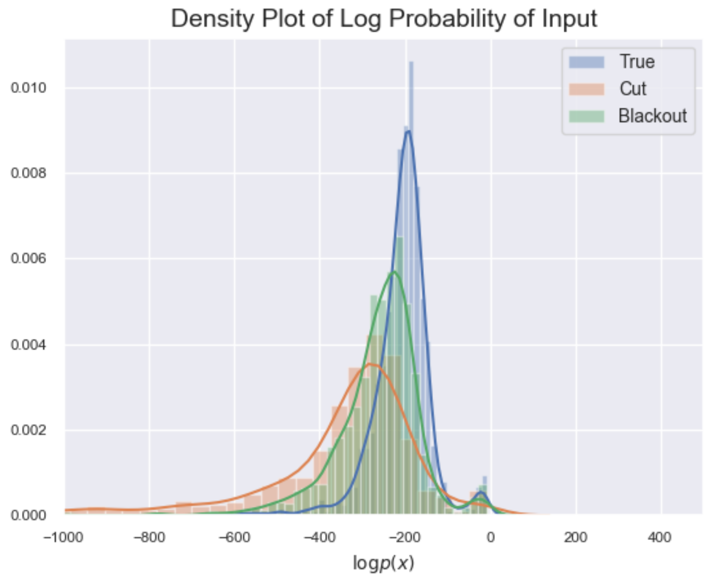


Figure 3.9: Density plot of the log probability of the inputs for the true full path, and when the network has been presented with corrupted input data by either cutting half of the paths out or blackening out part of the path. The distributions for the corrupted paths lie lower than for the true paths, supporting the hypothesis that they are more difficult to reconstruct.

From Figure 3.9 it is clear that the distributions of log probabilities for the corrupted paths have a lower mean and standard deviation than for the true paths, which would be expected. From this it also appears that the model finds it more difficult to reconstruct the path when half is missing, than when a part it blacked out. Again, this may not be surprising, as there are more possible endings for a half completed route, than for a route where the beginning and end is given.
The mean and standard deviation of the distributions are given in Table 3.1 alongside the test statistic when comparing the distribution of log probabilities for the corrupted paths and the true paths.
The test statistic when running a two sample t-test for both corrupted paths are well above the critical value of 1.96, and the mean of the distributions must then be considered different from the mean of the true path distribution.

Table 3.1: Mean and standard deviation of the distributions of log probabilities of the true path when given either the true path as input or a corrupted path. The test statistic for a two-sample t-test t_{obs} indicate that the distributions are significantly different as they are larger than the critical value of 1.96.

Distribution	**Mean**	**Std**	t_{obs}
True	-206.81	66.26	-
Cut	-357.43	214.59	25.82
Blackout	-261.31	96.41	17.80

Anomaly Detection

To detect anomalies the log probabilities of the training data are used to determine a threshold. Any path that receives a log probability lower than the threshold is flagged as an anomaly such as,

$$\sum_m \sum_n \log p(x) < \tau \rightarrow \textit{anomaly} \tag{3.2}$$

where $m \times n$ is the dimension of x. The threshold is chosen as the 5^{th} percentile of the log probabilities in the training set evaluated at the last epoch of training, which here is **-301.32**. Choosing this threshold corresponds to an estimate of 5% anomalous track in the training set.
Using this threshold on the paths in the test set resulted in 5.7% being flagged as anomalous. In Figure 3.10 paths from the test dataset are illustrated, colored red if they are flagged as anomalies. Here it is seen that the paths marked for anomalies are often routes partly or completely outside the shipping lanes. This illustration also confirms that the marked anomalies are paths not often present in the test set. Some of the marked paths however do appear to follow the shipping lanes, which is common for normal behaviour.

To further support the method of a global threshold to detect outliers, some fabricated outliers were made and the models ability to flag them as anomalies was investigated. Two different altercations

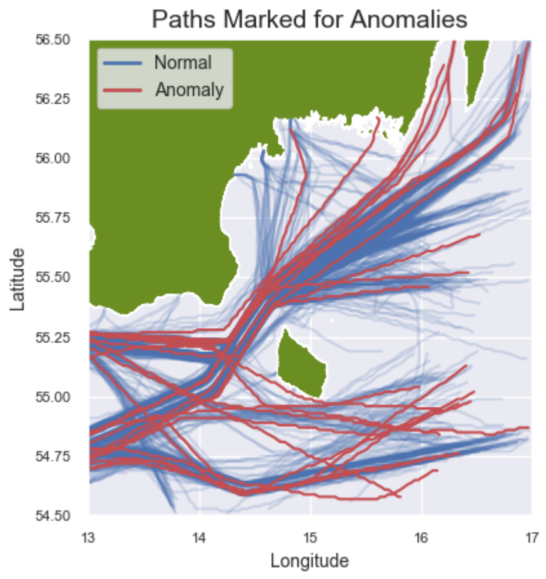


Figure 3.10: Paths from the test set marked as anomaly if the log probability of the input is less than τ. Here it is seen that many of the flagged paths do travel in areas less populated by the test set, however some flagged anomalies seem to exhibit normal behaviour and should be investigated further.

were used to fabricate anomalous tracks. The first method was flipping the routes vertically often creating routes in areas where no data has previously been presented to the model. The hypothesis is that the model should be able to catch most of these anomalies. The second fabrication of anomalies was made by shifting all routes 5 km north, while ensuring that the routes stayed within the ROI. If it was not possible to shift them 5 km north they were shifted 5 km south. This corresponds to a shift of 0.05° latitude. Examples of the original path and the two fabricated anomalies are seen in Figure 3.11.

To investigate the performance on the anomalous datasets the distribution of the log probabilities of the inputs are once more plotted with the log probability of the true inputs as reference. The illustration is seen in Figure 3.12 with corresponding means and standard deviations in Table 3.2. Again, the test statistic for comparing the mean of the anomalous log probability distributions to the true log probability distribution indicate a significant difference in means, as they are both above the critical value of 1.96.

As expected, Figure 3.12 and Table 3.2 clearly shows that the model performs significantly worse on the flipped inputs, and with the threshold for anomalies defined previously, the model flags 96.3% of the paths as anomalous. The shifted routes on the other hand are more difficult for the model to catch as anomalies. One straightforward explanation for this could be, that shifting some routes 5 km may not result in anomalous paths, as some of the shipping lanes has a width of around 5 km. It is however

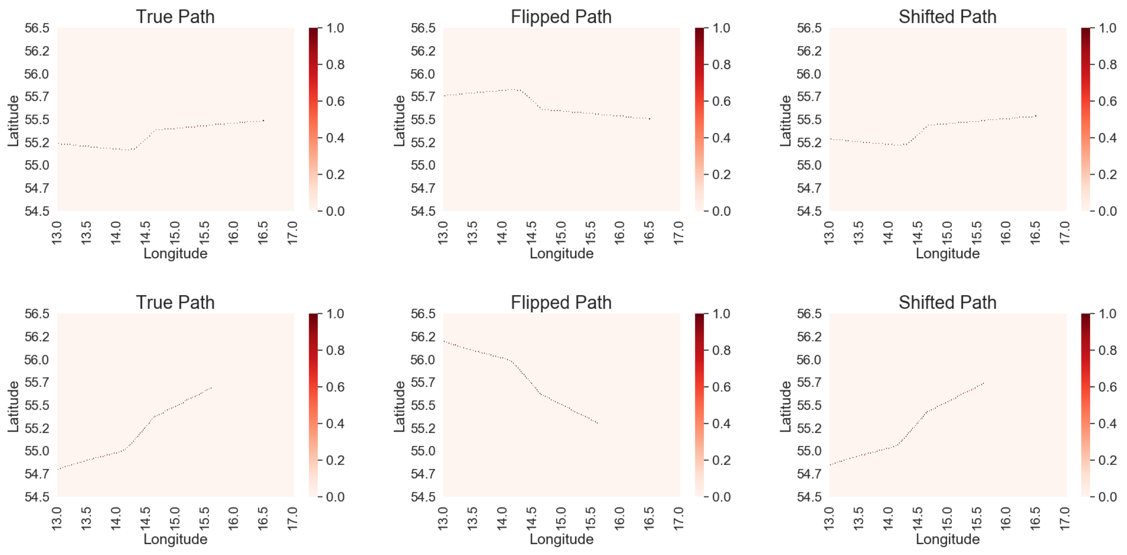


Figure 3.11: Two examples of fabricated anomalies. Far left shows the original path, the middle shows the anomaly created by flipping the route vertically and the far right shows the anomaly created by shifted the path 5km north.

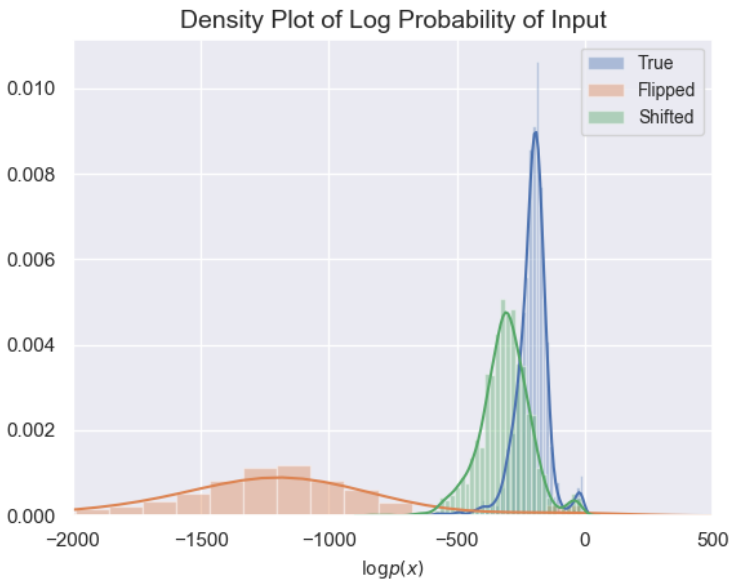


Figure 3.12: Density plot of the log probability of the inputs for the true full path, and when the network has been presented with fabricated anomalies created by either flipping the paths vertically or shifting the path 5km north. The distributions suggest that using the reconstruction probability as a metric to flag outliers would be very useful in the case of the flipped paths, but would not be able to mark all shifted paths as anomalous.

able to catch 53.7% of the shifted paths as anomalous using the threshold defined earlier. Examples of predicted anomalies for both the case of the flipped and shifted routes can be seen in Figure 3.13 and Figure 3.14 respectively.

Table 3.2: Mean and standard deviation of the distributions of log probabilities of the true path or an anomalous track. The test statistic t_{obs} if the result of a two-sample t-test in which the means of the anomalous distributions are compared to the true distribution, and with both test statistics above the critical value of 1.96, the distributions must be considered different.

Distribution	**Mean**	**Std**	t_{obs}
True	-206.81	66.26	-
Flipped	-1503.94	974.70	50.51
Shifted	-311.78	100.96	33.21

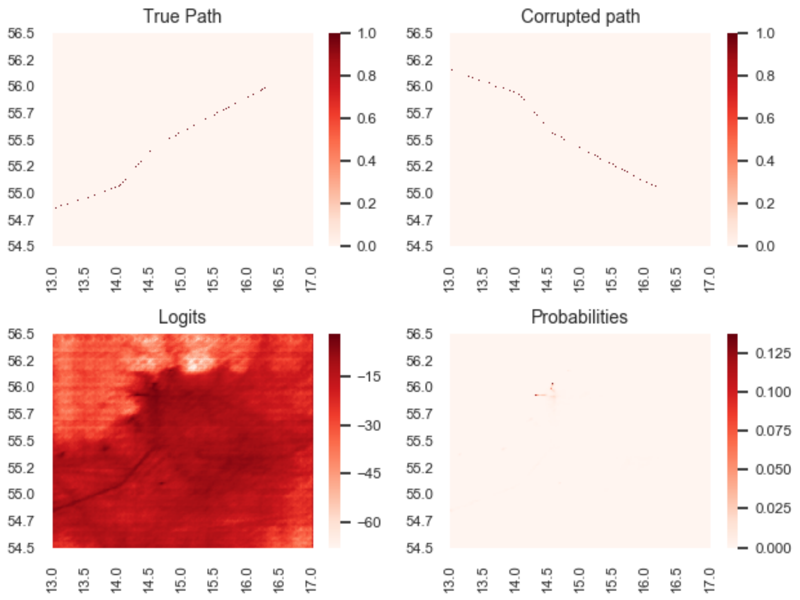


Figure 3.13: Example of an anomaly created by flipping the route vertically. Here the true path is shown alongside the anomalous path in the top of the graph. The bottom shows the logits from the distribution learned by the decoder, and the logits transformed to probability space

However, the fabricated anomalies are often infeasible in reality, and would not be able to serve as an indicator for how the model would deal with real-life anomalies, but merely as a proof of concept. Looking at the flipped route anomalies shown in Figure 3.15a the unfeasibility of the tracks become apparent, as most of them run straight through in-land Sweden and across the island of Bornholm. In Figure 3.15b the shifted route anomalies are shown, and although the trajectories here are not as impractical as the flipped route anomalies, they do also cross land creating routes where no ship would be able to sail in reality.

To address this issue, there is a need for anomalous yet feasible trajectories. For this purpose trajectories from passenger ships were used as the last anomalous behavior presented to the model. The motivation behind using passenger ships as anomalies was, that they present real behaviour in the sea but behave very differently than a tanker or cargo ship. Where it would be expected for many of the passenger ship trajectories to lie in the path from Rønne to Ystad, this track would be highly unlikely

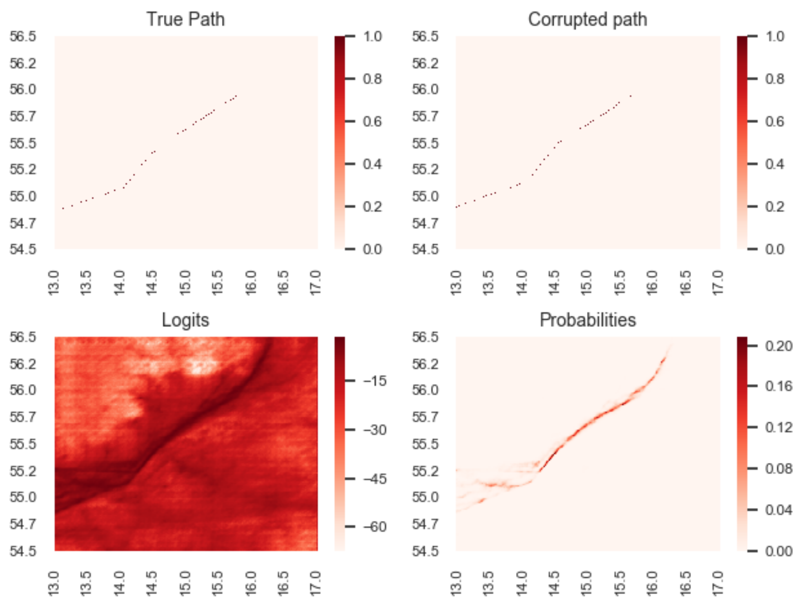


Figure 3.14: Example of an anomaly created by shifting the route 5 km north. Here the true path is shown alongside the anomalous path in the top of the graph. The bottom shows the logits from the distribution learned by the decoder, and the logits transformed to probability space

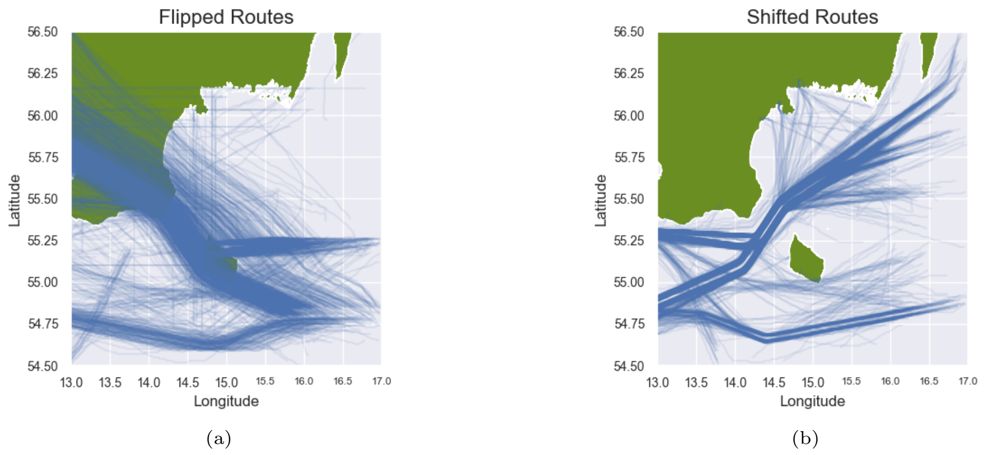


Figure 3.15: Illustration of the dataset of flipped route anomalies (a) and the dataset of the shifted route anomalies (b). The illustration clearly shows that many of the fabricated anomalies are infeasible in reality, and although valid as a proof of concept, not useful as an indicator for the models performance on true anomalies.

for both tanker and cargo ships.

However, since the behaviour of the passenger ships differs from that of the tanker and cargo ships, the data preprocessing cannot be equal either. For passenger ships a trajectory was deemed as two individual paths if the time interval between two successive AIS messages exceeded 30 minutes, in comparison with 4 hours for the tanker and cargo ships. Furthermore, the minimum duration of a trajectory was lowered from 4 hours to 15 minutes, and the maximum duration was lowered from 24 hours to 6 hours. The passenger ship trajectories were collected from January till March 2020 as for the tanker and cargo ships. Figure 3.16 shows the final passenger ship 856 trajectories.

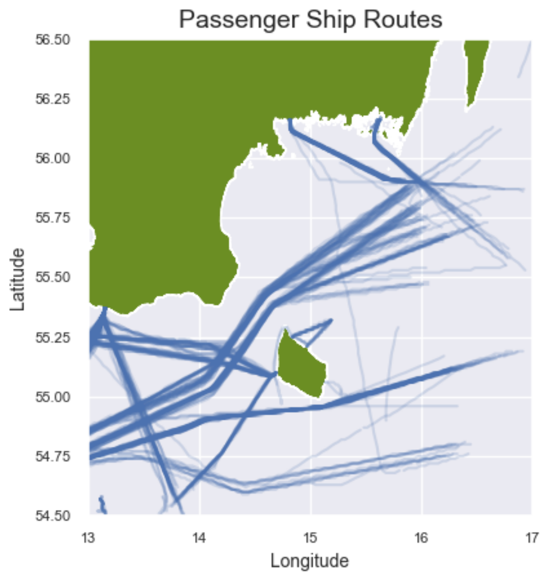


Figure 3.16: Illustration of the routes of passenger ships presented to the model as anomalous behaviour.

Comparing Figure 3.16 to the shipping profiles seen in Figure 3.1 it is clear that although traffic in the shipping lanes north of Bornholm are active in both, the passenger ship trajectories hold many more routes connected to Rønne as well as from Sweden to Germany. If the model has in fact learned a useful representation of the paths in the training set, the hypothesis is that the distribution of log probabilities of the input when presented with passenger ships would be significantly less than that of the true cargo and tanker ships. However, looking at the distribution shown in Figure 3.17 this claim is not supported. Even though the test statistic of 13.30 between the distributions indicate that they are different, it would have been expected that the passenger ships performed worse than the ship types used in training and hence had a distribution of log probabilities lower than the true paths.

Figure 3.18 shows the trajectories in the passenger ship dataset marked as an anomaly if the log probability of the input is less than τ.

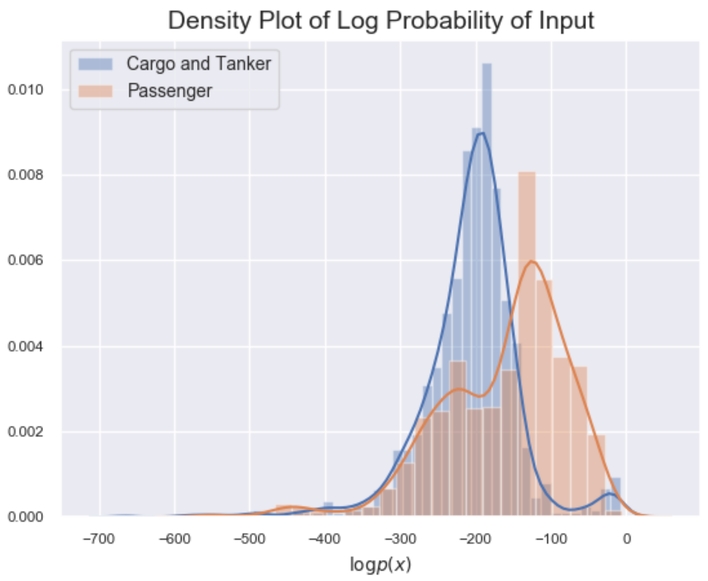


Figure 3.17: Distribution of log probabilities of the input for cargo and tanker ships in the test set againt passenger ships. In addition to not being clearly separable, the probability of the passenger ships actually lies higher than the true ships from the test set.

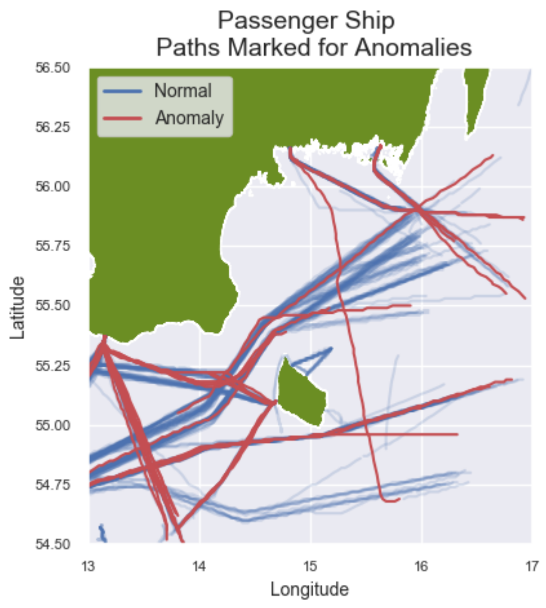


Figure 3.18: Illustration of the passenger ship routes used as feasible anomalous tracks flagged as anomalies if the log probability of the input is less than τ. The long paths deviating significantly from the behaviour of the cargo and tanker ships are as expected flagged as anomalies. However, many of the shorter paths close to Bornholm are not flagged, as would be expected.

Looking at the paths marked as anomalies in Figure 3.18 it is clear that it is in fact the routes out of Rønne or the ones crossing over multiple shipping lanes that are marked. A large majority of the passenger ships routes happen to lie in the shipping lanes north of Bornholm which it would not be expected for the model to flag as anomalous. Nonetheless there are still some tracks around Bornholm that clearly differ from the ones on which the model was trained, and which should be flagged as anomalies as well.
These tracks are though significantly shorter than many of the cargo and tanker ship tracks, which exposes a weakness in the definition of anomalies. Since most of the pixels in the input images are 0 the log probability of a 0 is often much higher than that of a 1. Shorter paths will therefore have a tendency to get a higher log probability of the input, because the number of pixels of which it is likely to make mistakes on are simply less. Looking at the probabilities in Figure 3.6, which shows the probability of a 1 for the true paths in the test set, it shows that even the highest probability of a 1 is only 25%. This feature in itself is not an issue, but if the summed log probability of the pixels in the image is used as a metric to find anomalies, it becomes problematic if the paths vary much in length.

Looking back at Figure 3.10 showing some of the anomalies from the true test set, this may also explain why some of the paths marked as anomalies seem to lie within the shipping lanes. Looking at the lengths of the paths it is seen that they often span from one end of the ROI to the other, and the shear fact that they are very long may be detrimental. Figure 3.19 supports this, showing the distribution of path lengths for the normal and anomalous tracks of the test set with a test statistic when comparing the mean of the two of 14.24 clearly indicating a difference in mean lengths.

Since the duration of the trajectory has a large influence on the probability of being classified as an anomaly, it could indicate that a global threshold of reconstruction probability is not the best metric to correctly flag anomalies. Inspired by (Nguyen et al. 2019a) in which they set local thresholds for a given geographic area, a simple approach taken in this thesis is setting local thresholds based on path duration. Five separate thresholds were defined, one for each 4 hour interval between 4 hours and 24 hours duration, as preprocessing has ensured that all trajectories lie within this time frame. The 5^{th} percentile of log probabilities of the paths in the training set falling within each interval was chosen as the local thresholds. In Figure 3.20 the threshold for each interval is illustrated clearly showing the linear dependency on duration.

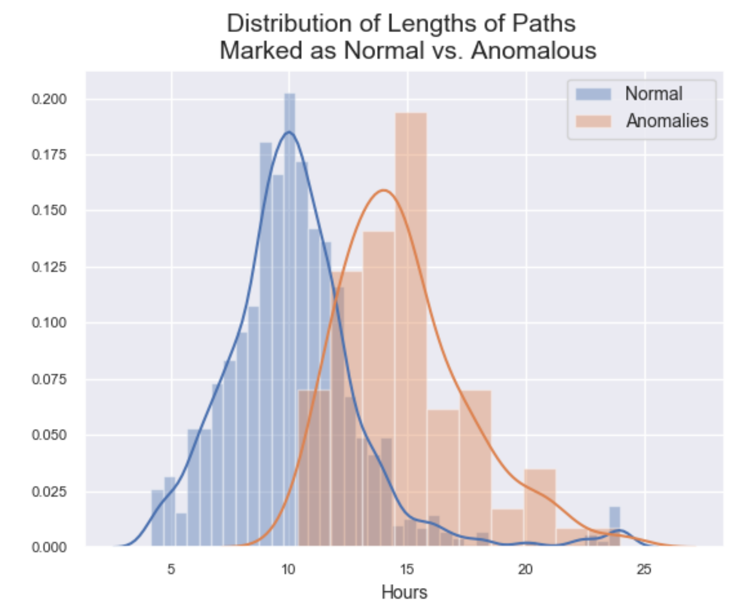


Figure 3.19: Distribution of duration of trajectories split on whether the model flags them as anomalous. Here it becomes clear, that the general length of the anomalous tracks are longer than that of the flagged normal tracks.

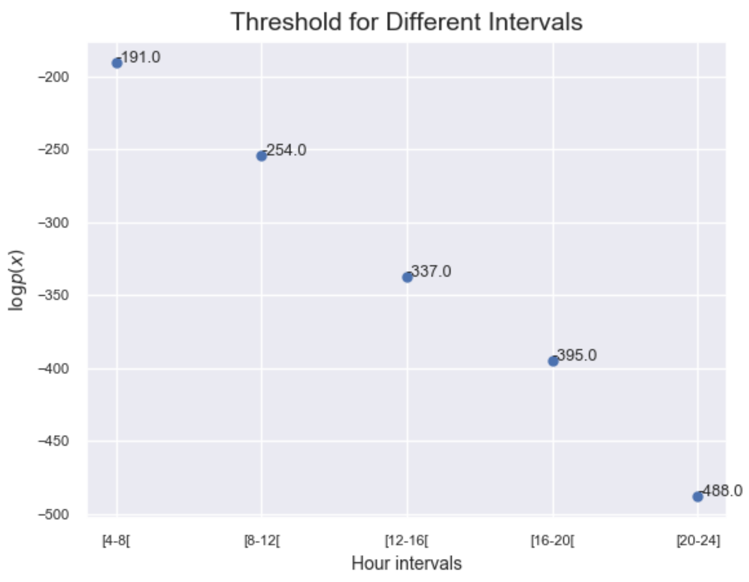


Figure 3.20: Thresholds for each 4 hour interval in the feasible span of duration from 4 hours to 24 hours. The linear dependency between duration and the log probability for and input become very clear as a longer path renders lower log probability.

Applying these local thresholds on the paths from the test set made it possible to flag some of the shorter paths as anomalies too. In Figure 3.21 some of the flagged anomalies are seen. Comparing to Figure 3.10 where the anomalies were a result of a global threshold, Figure 3.21 shows fewer trajectories running in the well-defined shipping lanes flagged as anomalies. Instead the anomalies are predominantly trajectories crossing shipping lanes or deviating from them at points during the journey.

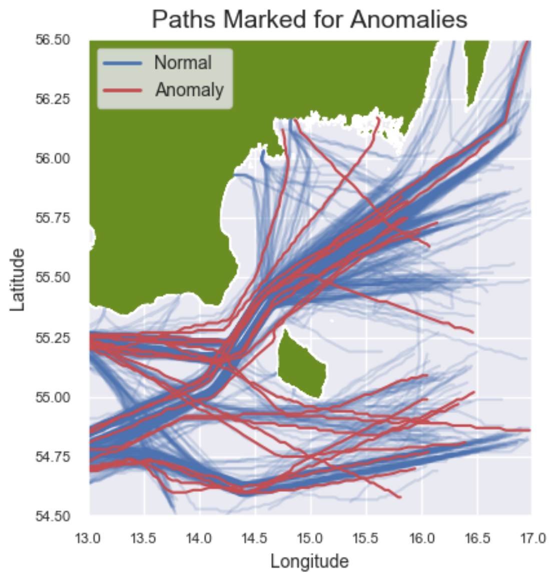


Figure 3.21: Thresholds for each 4 hour interval in the feasible span of duration from 4 hours to 24 hours. The linear dependency between duration and the log probability for and input become very clear as a longer path renders lower log probability.

To further illustrate that the local thresholds lessen the importance of duration, the distribution of log probabilites of paths flagged as normal versus anomalous are seen in Figure 3.22. Although the test statistic when comparing the means of the two distributions are **4.26** indicating a significant difference in mean, the means are closer than for the same distributions when using a global threshold as seen in Figure 3.19.

The results presented in this section indicate that it is possible to use a convolutional VAE to predict ship trajectories, and subsequently use the reconstruction probability as an indicator for whether the track could be classified as anomalous or not. However, certain problems arises when handling the trajectories as images. First, if the lengths of the paths in the dataset are substantially different, a global threshold using the log probabilities of the input will favor shorter paths. This favoritism can be lessened by introducing local thresholds based on duration of trajectories, but does not solve the problem completely.

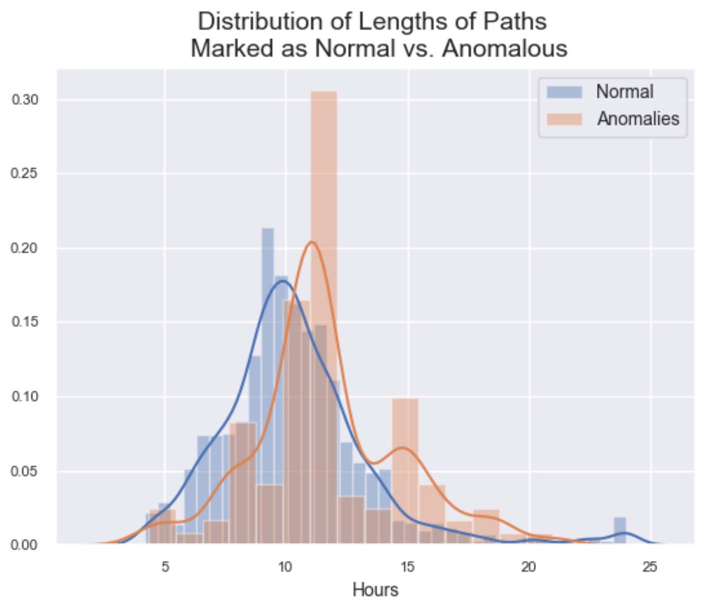


Figure 3.22: Distribution of duration of trajectories split on whether the model flags them as anomalous when using individual thresholds based on durations. The difference in means of the two distributions are less than for duration of anomalous tracks when using a global threshold as seen in Figure 3.19.

Further, as seen when shifting the routes 5 km north, the model only flags around half of the fabricates trajectories as anomalous. One explanation presented for this could be, that shifting the paths does not make them anomalous but merely moves them from one shipping lane to another, such as the ones north of Bornholm. In practice, this would mean that the ship is sailing against traffic, but given that the input is simply an image of the full trajectory, the model has no sense of direction.
Introducing time as a variable in the model would help overcome both these drawbacks seen by the CVAE. By letting the input be one timestep at a time, the length of the route would not be as dominant a factor, as the log probability is always evaluated for a single point. Additionally, the paths shifted from one shipping lane to another would now appear anomalous as the ship would be travelling in the wrong direction compared to the training set.

To incorporate the aspect of time into the model a VRNN is used, and the results are discussed in the following sections.

3.3 VRNN for Trajectory Prediction

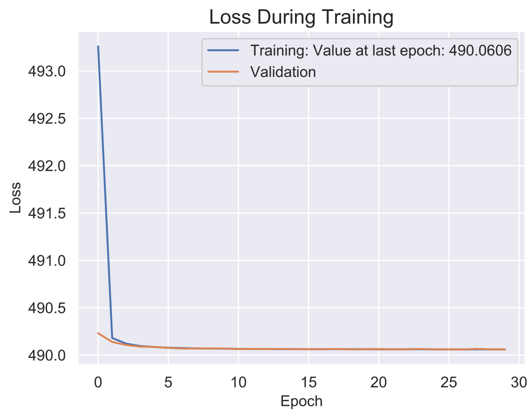
Using a VRNN for trajectory prediction and subsequent anomaly detection, was exactly the task performed by (Nguyen et al. 2019a). The implementation in this thesis follows the approach in the article, and the following sections gives an overview of the experimental setup as well as added modifications.

3.3.1 Neural Network Architecture

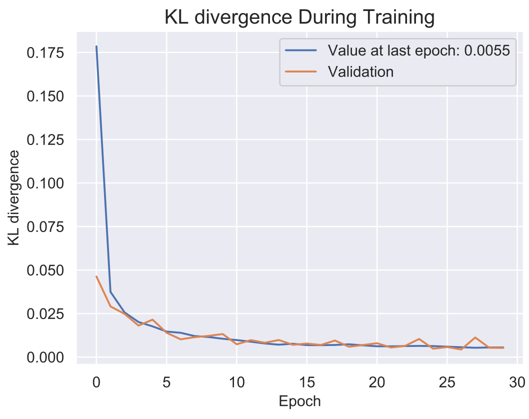
To model the hidden states in the VRNN an LSTM is used for the function f in Equation 2.39. It is parameterized by a single hidden layer of size 100, with z_t being a real-valued vector of the same size. Both the prior and approximate posterior are Gaussian distributions parameterized by two fully connected networks with one hidden layer of size 100. The generative distribution is a multivariate Bernoulli also parameterized by a fully connected network of size 100 and the Monte Carlo estimation of the ELBO is used as the objective function. The Adam optimizer is used with a learning rate of 0.0003.

3.3.2 Results and Analysis

Running the settings as documented from the article on the dataset used in this thesis, did not result in a useful trajectory prediction. The biggest initial problem was that the decoding network predicted logits of 0 no matter the input. Looking at the loss, it is clear that it thresholds very quickly, learning no additional information after the first few epochs. Further, the KL divergence is very small and approaches 0 during training. The loss and KL divergence can be seen in Figure 3.23.



(a)



(b)

Figure 3.23: Loss and KL divergence over 30 epochs for the VRNN. The loss seem to threshold very quickly as the KL approaches 0.

From Figure 3.23 especially the small KL divergence stands out, as it could be an indicator of a *posterior collapse*. For variational autoencoders with strong autoregressive decoders, the problem of KL vanishing or posterior collapse has been a recurring problem (J. He et al. 2019; Long et al. 2020; Zhu et al. 2020). The problem arises when the approximate posterior disregards information from the data and simply collapses to the prior such as,

$$q_\phi(z_t|x_t) \approx p(z_t) \tag{3.3}$$

In this case the latent variables, z, and the data, x, are independent and the variational autoencoder has not learned a good representation of data as most of the modelling power of a VAE comes from a strong coupling of x and z (Dieng et al. 2019; Hoffman D. and Johnson J. 2016). When this happens the decoder gains no useful information from the latent space, and hence quickly learns to ignore it, creating a critical local optimum. This results in little to no gradient signal between the decoder and encoder with a KL term at zero (Bowman et al. 2015).

Besides the KL divergence, two common metrics are used, when investigating the performance of a VAE; the mutual information (MI) between x and z under the approximate posterior, and the number of active units (AU). The mutual information between x and z under the approximate posterior can be determined as (J. He et al. 2019),

$$I_q = \mathbb{E}_{x \sim p_d(x)}[\mathrm{KL}(q_\phi(z|x)||p(z))] - \mathrm{KL}(q_\phi(z)||p(z)) \tag{3.4}$$

where $p_d(x)$ is the empirical distribution, and the aggregated posterior $q_\phi(z) = \mathbb{E}_{p_d(x)}[q_\phi(z|x)]$ (Dieng et al. 2019) which can be approximated by a Monte Carlo estimate. A Monte Carlo estimate can also be used to approximate $\mathrm{KL}(q_\phi(z)||p(z))$ where samples from $q_\phi(z)$ can be obtained by ancestral sampling (J. He et al. 2019). The reason for looking at the mutual information between z and x under the approximated posterior is further motivated in Section 3.3.4. A very small MI would indicate independence between the data and the learned latent space and could therefore be an indicator of a posterior collapse.

The second metric of interest is the number of Active Units (Burda, Grosse, and Salakhutdinov 2016). The motivation behind this stems from observations of VAEs in which the effective dimension of the learned latent space is far below capacity. To quantify this effect Active Units is introduced, following the hypothesis that if a latent dimension has learned useful information about the data, its distribution should change depending on the observation.

The activity of latent dimension i of latent variable z can then be measured as,

$$A_i = \mathrm{Cov}_x(\mathbb{E}_{i \sim q(i|x)}[i]) \tag{3.5}$$

where Cov is the covariance depending on x of the expectation $\mathbb{E}$ of the latent dimension under the approximate posterior q.

A unit is then said to be active if A_i is above some thresholds. For this thesis a threshold of 10^{-2} is used following the approach given in (Burda, Grosse, and Salakhutdinov 2016). Little to no active units indicates that the latent space has not learned useful information about the data and could hence also be an indicator of posterior collapse.

The MI and AU for the implementation of the VRNN illustrated in Figure 3.23 are seen in Table 3.3 both strong indicators that a posterior collapse has indeed taken place.

Table 3.3: Mutual Information and Active Units of the implementation of VRNN

	MI	AU
VRNN	0.014	0

Looking through the code published with the article from (Nguyen et al. 2019a), a few modifications to the model are observed. In the following a brief description and discussion of these modifications will be introduced as well as common methods used to alleviate the situation of a posterior collapse.

3.3.3 Network Wide Skip Connection

First implementation found in the code from (Nguyen et al. 2019a) is an addition of the dataset mean transformed to logit space added to the logits output from the decoder before generating the Bernoulli distribution of the generative network. Second, in the formulation of the approximate posterior the mean from the prior is added to the mean from the encoder, before the Gaussian distribution of the approximate posterior is made.
Both additions could be thought of as network wide skip connections. The use of skip connections, or residual blocks, is one of the key elements in the paper by He et al. from 2015 in which they introduced ResNet (K. He et al. 2015). With their model they addressed a well-known phenomenon seen when training deep neural networks. A degradation problem has been observed in which, as the depth of the network is increased, the accuracy of the network gets saturated whereafter it decreases rapidly. This is hypothesized to be a result of vanishing gradients which creates a difficult optimization problem. To overcome this they add skip connections as a simple identity mapping from one layer in the network to another. This way they explicitly let the layers in their network learn a residual mapping, namely the

difference between the input, x, and the underlying desired mapping, $\mathcal{F}(x)$, and argue that it should make optimization easier. An example of a residual block is illustrated in Figure 3.24.

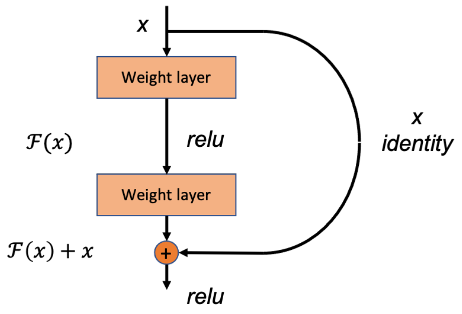


Figure 3.24: Example of a building block in residual learning, in which an identity mapping is added from one layer to another

In (Fraccaro 2018) they use this exact logic to prevent very small KL-terms when training sequential neural models with stochastic layers. Here they add the mean of the prior to the mean of the approximate posterior arguing that learning the residual of the means, improves inference by making it easier for the inference network to discover changes in the generative network.

Adding skip connections to the generative model of a VAE was done in (Dieng et al. 2019), also as a remedy for the posterior collapse, which could prove as motivation for the authors of adding the mean logits to the generative network in (Nguyen et al. 2019a).

3.3.4 Posterior Collapse

One of the most popular methods for dealing with the posterior collapse is by implementing KL annealing (Bowman et al. 2015). Here a variable weight, β, is added to the KL term in the ELBO objective as seen in Equation 3.6.

$$\mathcal{L}_{\theta,\phi}(x) = \mathbb{E}[\log(p_\theta(x|z))] - \beta \mathrm{KL}(q_\phi(z|x)||p(z)) \tag{3.6}$$

The weight is initially set to 0 and gradually increased after each epoch during training until it is 1 and the objective function once more becomes the proper lower bound. By doing this, the authors argue that the model will initially learn to encode as much information in z as possible, and gradually, as the weight is increased, smooth out the encoding and force them towards the prior. It corresponds to annealing from an AE to a VAE. A variant of the KL annealing is the Cyclic Annealing proposed by (Fu et al. 2019) in which β drops back to 0 and increases to 1 a number of times during training.

Looking deeper into the theoretical understanding of the collapse, (J. He et al. 2019) described two collapse states to explain the drivers for the posterior collapse; the model collapse, and the inference collapse. The model collapse happens when the true posterior equals the prior and happens for all VAEs in the beginning of training when x and z are nearly independent under both the approximate posterior and true posterior. In this case there is only one component in the training objective that can install some dependency between x and z, namely the marginal log likelihood of the data. However, it can be overwhelmed by the KL term, as the posteriors are pushed towards the prior to ensure alignment.

The inference collapse is on the other hand when the approximate posterior equals the prior, and they observe that this always happens before the model collapse. Their proposed method to handle the collapse is an aggressive optimization of the inference network, but this has been observed to cost too much training time compared to the orignal VAE (Zhu et al. 2020).

To address the issue of posterior collapse without additional training efforts (Zhu et al. 2020) propose to instead let the KL follow a distribution over the entire dataset, and show that it is sufficient to keep the expectation of this positive to prevent KL vanishing. To derive this a different formulation of the KL divergence is needed. Since the prior of a VAE is often a Gaussian distribution, then the KL between the prior and the approximate posterior can be written as,

$$KL = \frac{1}{2}\sum_{i=1}^{n}(\mu_i^2 + \sigma_i^2 - \log(\sigma_i^2) - 1) \tag{3.7}$$

where μ_i and σ_i are the mean and standard deviations of the approximate posterior.

Given that the KL is often found as the Monte Carlo estimate, Equation 3.7 can be written as,

$$KL = \frac{1}{2}\sum_{i=1}^{n}\left(\frac{\sum_{j=1}^{b}\mu_{ij}^2}{b} + \frac{\sum_{j=1}^{b}\sigma_{ij}^2}{b} - \frac{\sum_{j=1}^{b}\log(\sigma_{ij}^2)}{b} - 1\right) \tag{3.8}$$

The authors of (Zhu et al. 2020) then assume that the μ_i's and σ_i's of each latent dimension i follow a certain distribution with fixed mean and variance. That then turns the KL into a distribution of μ_i's and σ_i's with expectation given as,

$$\mathbb{E}[KL] \geq \frac{1}{2}\sum_{i=1}^{n}(\mathrm{Var}[\mu_i] - \mathbb{E}^2[\mu_i]) \tag{3.9}$$

where Var is the variance and $\mathbb{E}$ indicates the expectation. They argue, that if a positive lower bound of $\mathbb{E}[KL]$ is guaranteed, posterior collapse will be prevented. From Equation 3.9 it is seen that $\mathbb{E}[KL]$ only depends on the number of latent dimensions n and the mean and variance of the μ_i's, which gives the motivation to regularize the distributions of the μ_i's by employing batch normalization. The regularized μ_i can be defined as,

$$\hat{\mu_i} = \gamma \frac{\mu_i - \mu_{\mathcal{B}i}}{\sigma_{\mathcal{B}i}} - \zeta \tag{3.10}$$

where $\mu_{\mathcal{B}i}$ and $\sigma_{\mathcal{B}i}$ are the mean and standard deviation of μ_i for a minibatch and μ_i and $\hat{\mu_i}$ are the means of the approximate posterior before and after regularization respectively. In (Zhu et al. 2020) they fix γ and let ζ be a learnable parameter. By doing this, they ensure that the distribution of μ_i has a mean of ζ and variance of γ^2, allowing them to formulate a new lower bound,

$$\mathbb{E}[KL] = \frac{n \cdot (\gamma^2 + \zeta^2)}{2} \tag{3.11}$$

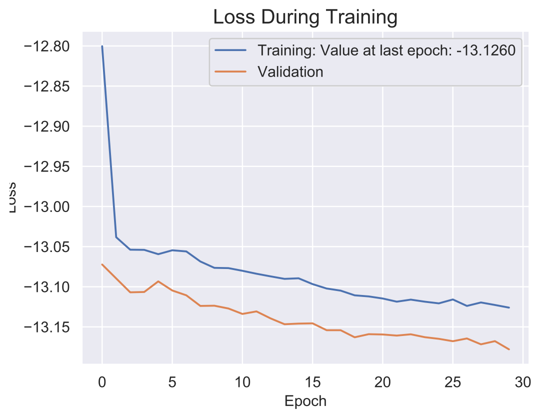
This allows them to control the lower bound of the expectation of KL by γ.

3.3.5 Results After Model Modifications

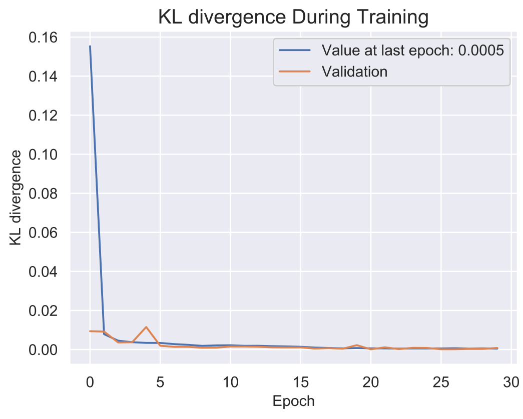
To alleviate the vanishing KL term seen in Figure 3.23 the skip connections identified in the source code were implemented alongside KL annealing. The loss, KL divergence, MI and AU after adding skip connections and KL annealing can be seen in Figure 3.25.

Looking at Figure 3.25 the loss profile indicates that the model does learn throughout training. The increased MI and AU supports this, indicating some useful activation of the latent space. Unfortunately, the KL divergence is even smaller than previously.

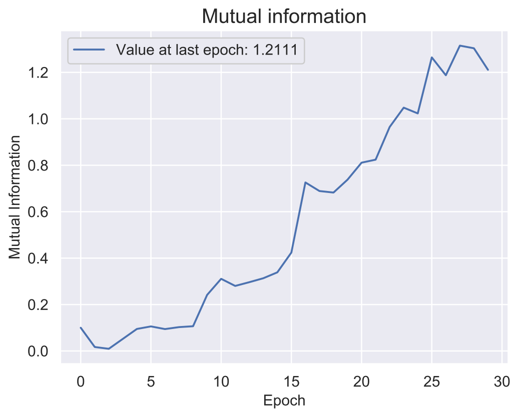
To get a sense of what the model has learned, Figure 3.26a show some reconstructed paths. A total of 32 paths are plotted, each with an opaque factor of 0.1, thus heavier colored nodes and paths are commonly predicted reconstructions. Figure 3.26b shows a heatmap of the matrix multiplication of the mean longitude and mean latitude for comparison. This indicates, that the model has learned to predict the mean of the routes. Further supporting this, is the longitude and latitude profiles seen in Figure 3.27. Here a single path is illustrated, showing the logits of the generative network for the longitude and latitude features respectively. Every timestep is once more plotted with an opaque factor of 0.1, resulting in a heavier colored line as indicator for commonly predicted values. Atop the predicted logits the mean from the dataset is visualized very clearly showing, that most commonly it is the mean profile predicted. At the top of the graph is the one-hot encoding of the true path showing where the peaks of the logit profile should lie, had the model been able to predict the trajectories correct.



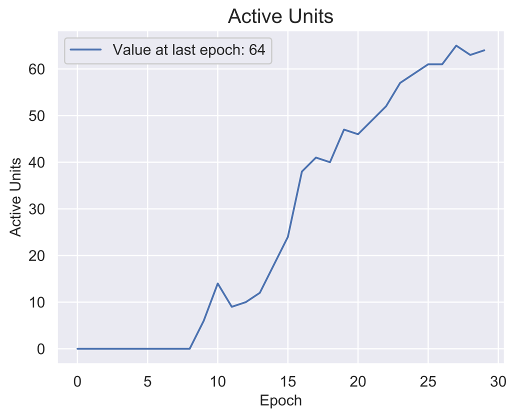
(a)



(b)

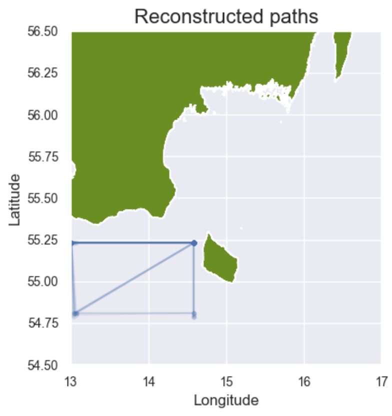


(c)

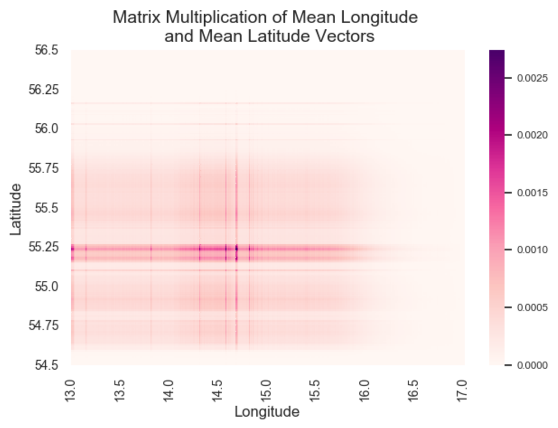


(d)

Figure 3.25: Loss, KL divergence, MI and AU over 30 epochs after adding skip-connections and using KL annealing for the first 10 epochs. Although the loss indicates continous learning, as well as the increased mutual information and active units, the KL converges to 0 within the first few epochs.

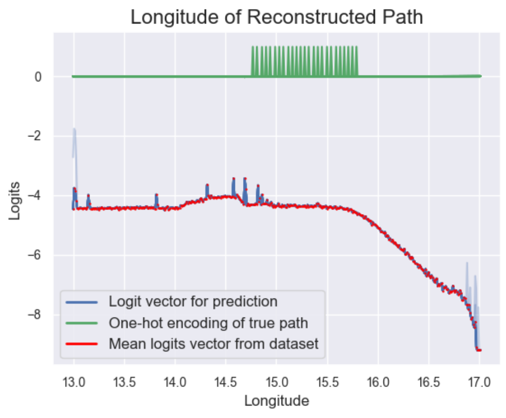


(a)

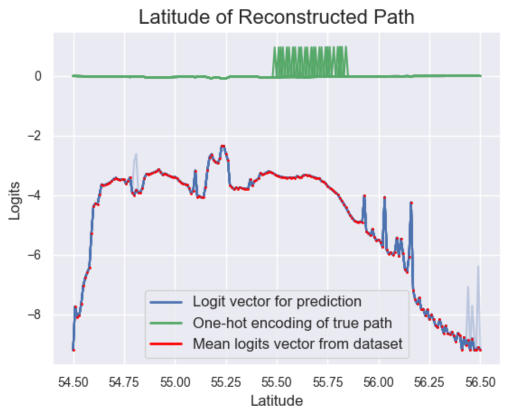


(b)

Figure 3.26: (Left) Reconstructed paths using VRNN model with skip connections trained for 30 epochs. The model predominantly predicts position as the mean of the dataset which can be seen when comparing to a heatmap of the matrix multiplication of the mean longitude and latitude vectors (right).



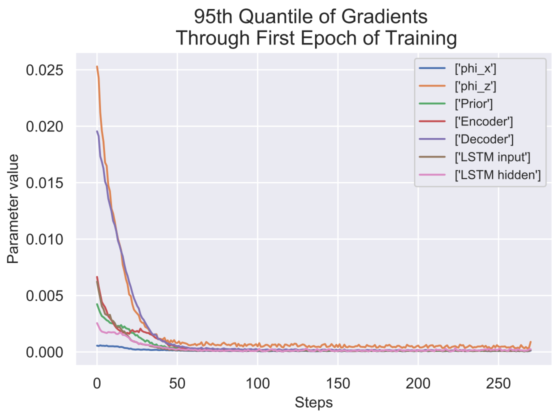
(a)



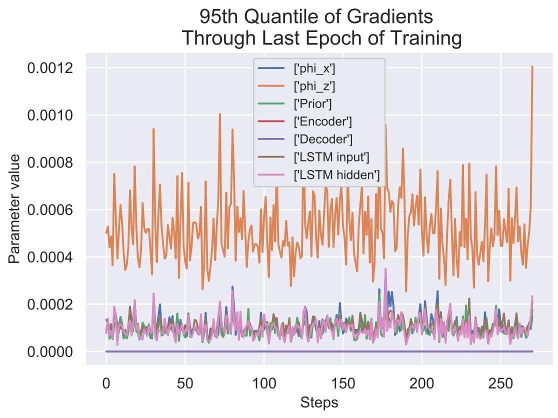
(b)

Figure 3.27: Longitude (left) and latitude (right) profiles of logits for at predicted path. Heavier color indicates a often predicted value, showing that more often that not, the logit profile completely match that of the mean. At a few timesteps other coordinates win indicated by the opaque spikes, but very rarely do they relate to the true coordinate of the path.

To further get a sense of which components in the network have trouble learning, the gradients are explored. As it is especially the presence of exploding or vanishing gradients that are of interest, the 95^{th} quantile of the gradients at each step is visualized, to capture as much information as possible. In Figure 3.28 the 95^{th} quantile of the first and last epoch is seen. All gradients seem to decrease rapidly in the first epoch, which is expected with the initial learning, but become very small very quickly. Most noticeably though is the visualization of the last epoch. Here it appears that although very small there is a gradient signal in all components except the encoder and decoder.



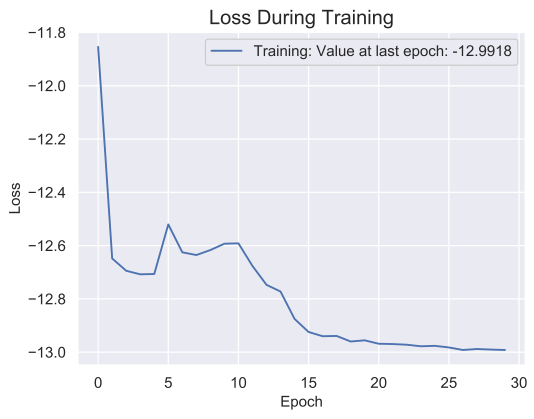
(a)



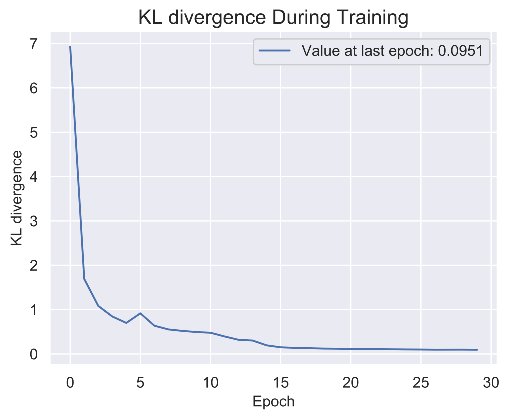
(b)

Figure 3.28: Trace of gradients through first (left) and last (right) epoch of training. Even though all gradients in the end of training are small especially the gradients of the hidden layer of the LSTM stands out. From the first epoch it could appear that it is small from the beginning of training. Please note the different scales of the y-axis.

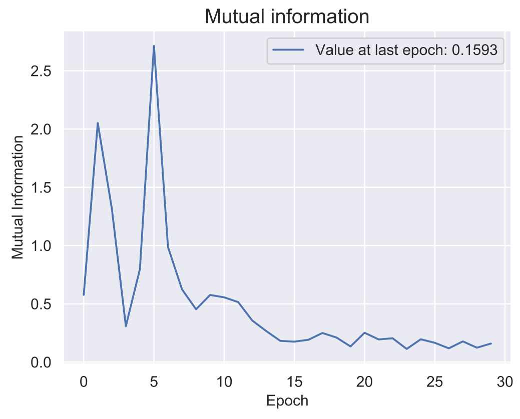
The vanishing gradients of the decoder and encoder motivates a stronger regularization in that part of the network. Thus the batch normalization of the approximate posterior is implemented with the result illustrated in Figure 3.29 where γ is set to 0.6 as in (Zhu et al. 2020). Although the KL term thresholds at a larger value than previously, the profiles of the loss, MI and AU could indicate unstable gradients, especially in the first few epochs. Also, the MI is decreased indicating that the model is not able to maintain the shared information learned between data and the latent space. Further supporting unstable gradients is the 95^{th} quantile of the gradients in the first and last epoch of training. As hypothesized, the gradients of the first epoch appear unstable with near-exploding behaviour. This is though not the case for the last epoch, indicating that the increase of gradients was controlled during training. The profile of the gradients in Figure 3.30 differ from those seen in Figure 3.28 as there is now plenty of signal in both the prior and encoder, indicating that the normalization has indeed added additional information to the latent space. Although it could appear that the remaining components does not learn much with very small gradients, changing the scale so it better compares to that seen in Figure 3.28 gives the result in Figure 3.30c. Here it is clear that the decoder still has very small gradients, and the weights in the hidden state of the LSTM also seem to vanish. Controlling the unstable gradient by gradient clipping resulted in similar results to the model without batch normalization. Further, changing the learning rate did also not solve the problem.



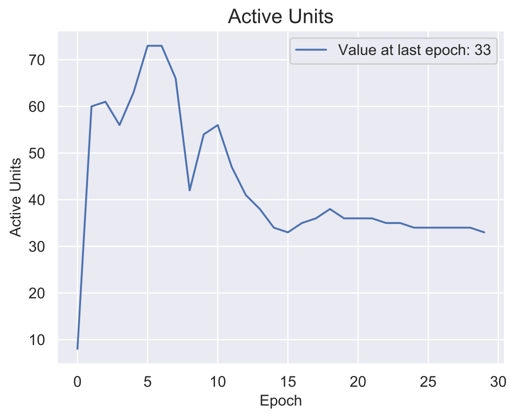
(a)



(b)

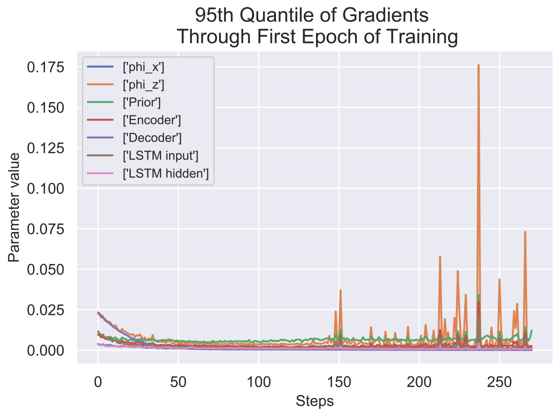


(c)

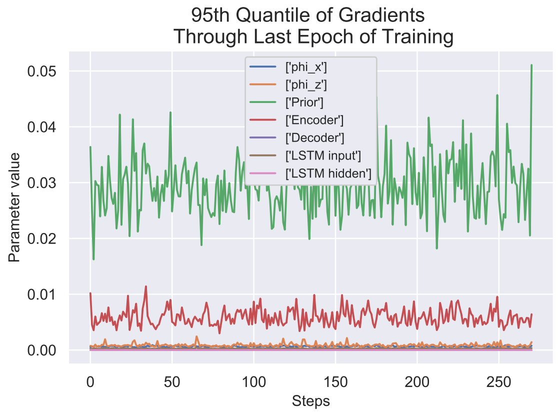


(d)

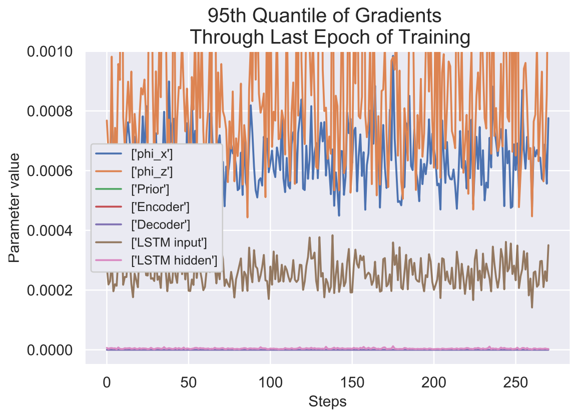
Figure 3.29: Loss, KL divergence, MI and AU over 30 epochs for model with batch normalized inference. Note that the validation curve for the loss and KL divergence has been omitted, as very large values in some of the first epochs corrupted the image.



(a)



(b)



(c)

Figure 3.30: Trace of gradients through first (top left) and last (top right) epoch of training for VRNN model with batch normalized inference network, showing how the gradients seem to initially explode after implementing batch normalization. The gradients in the last epoch are shown at a different scale (bottom) to illustrate the activity in the feature extraction networks as well as the weights on the input of the LSTM. The decoder and weight on the hidden state of the LSTM seem to vanish completely.

CHAPTER 4

Discussion

This thesis has employed two deep neural network models in an attempt to reliably capture useful information about ship trajectories, with the aim to reconstruct the trajectories and subsequently use the reconstruction error as an indicator for anomalous behaviour.
The first model was a Convolutional Variatioal Autoencoder with the trajectory as an image as input. It was shown that the generative network of the model was able to learn separate logit landscapes depending on the input, with a noticeably higher probability of occurrence of the path, in the parts of the image in which the path was indeed present. To further test the robustness of the model, it was presented with corrupted paths and the log probability of the true input was recorded. The distribution of the summed log probabilities was used as a metric for comparison between the reconstruction abilities given the different corrupted paths. As expected the model performs worse when given a corrupted path to reconstruct, however the experiment also showed, that the model had a more difficult time reconstructing paths, when half was missing, than when a given portion was cut out. Intuitively it makes sense, that it is easier for the model to reconstruct a trajectory, when both the beginning and end is given, than when the ending is completely open.

Another simple explanation could lie in the generation of the corrupted paths. Cutting half the path away was a result of setting the last 2° longitude to 0, and hence blackening them. The corrupted paths in which a part was blackened, was created by choosing a point on the trajectory, and blackening everything in a rectangle centering that point of 0.4° latitude and 1° longitude. Depending on the position of the trajectories in the image a larger portion of the image may be cut away for the corruption discarding half the image. This consequently make the part of the trajectory of which the model is uncertain larger hence resulting in a lower log probability of the input.
Even though the two corruption methods may not be directly comparable, both resulted in a log probability distribution significantly different from that of the true inputs, as shown by the two-sample t-test.

Following an assumption that the model had indeed learned a useful representation of the cargo and tanker ship routes, any routes difficult to reconstruct could be indicators of anomalies. To flag these routes, a threshold of the summed log probabilities were set as the 5^{th} percentile of the distribution of summed log probabilities from the training set. A consequence of choosing this threshold is an assumption that 5% of the paths in the training data are anomalies. In Figure 3.10 some of these anomalies are visualized with the remaining test set as reference. Although it does appear that many of the flagged anomalies indeed behave differently than the other tracks in the test set, it is also observed that some of the flagged anomalies lie within the shipping lanes which characterizes the test set. Manual inspection of each of the flagged routes would be necessary in order to determine whether they were indeed anomalous or not. As a remedy to avoid a large quantity of manual inspections, some fabricated anomalies were presented to the model to see how it performed on those. Three different anomaly types were used two of which were fabricated and one which was simply trajectories from a different type of ship.
The performance on the two fabricated anomalies was very different. For the first fabrication the routes from the test set had been flipped vertically, and the model performed notably worse on these. In fact, using the same threshold as applied on the true paths from the test set, let the model flag 96.3% of the flipped routes as anomalous.
The second fabricated anomaly shifted the routes from the test set 5 km north and here the model flagged 53.7% of the fabricated routes as anomalies. One easy explanation for the difference in ability to flag the routes as anomalies, compared to the flipped routes, were hypothesized to be, that shifting the routes 5 km north, for some trajectories simply move them into a different feasible sea traffic area. Shifting a trajectory from one shipping lane to another, should be marked as an anomaly, since the ship in that case would be travelling against traffic, but when the model is independent of time, it does not have a sense of direction, and is thus not able to flag these paths as anomalies.

For both the flipped and the shifted paths it holds, that many of the fabricated trajectories become unrealistic, often indicating ships sailing across landmass. In order to draw conclusions about the models ability to detect anomalies for real-world data, a need arose for anomalous yet feasible tracks. To accommodate this, trajectories from passenger ships where presented to the model under the hypothesis that the behaviour of passenger ships differ so much from the behaviour of cargo and tanker ships, that most of the tracks should be marked as anomalies. However, when looking at the distribution of the summed log probabilities for the passenger ships, it was significantly higher than for the true test set.

Figure 3.18 shows the passenger ship trajectories with paths marked as an anomaly if the summed log probability was less than the defined threshold. The figure shows two important things. First, although it was hypothesized that the passenger ship behaviour would differ greatly from the behaviour of cargo and tanker ships, this is not necessarily true for all passenger ship routes. In reality, the figure illustrates that many of the passenger ship trajectories also lie within the well travelled sealanes used by the cargo and tanker ships. The assumption of highly different paths is hence not entirely valid.

Second, the figure shows that although the model only marks 4.5% of the passenger ships as anomalies, it seems to flag the routes that are indeed very different than the cargo and tanker ship behaviour. Examples of this includes trajectories running from Rønne on Bornholm to Ystad in Sweden, or trajectories from Sweden to Germany. This suggest that other factors may be involved in making the summed log probabilities of the passenger ship routes higher.

One hypothesis presented here, is that using the log probability of the input as an indicator for performance comes with a drawback. Since the model most often are presented with 0s the log probability of a 0 is almost always higher than that of a 1, and thus shorter routes will be likely to have a higher log probability. This theory was investigated further by looking at the distribution of lengths for paths marked as normal in the test set versus paths marked as anomalies. A clear difference between the two distributions were seen with the anomalous tracks being generally longer.

In an attempt to solve this side effect of using a global threshold, local thresholds were introduced based on the duration of the trajectories. The local thresholds were set as the 5^{th} percentile of log probabilities of routes with durations falling within the same 4 hour interval. In addition to showing a clear linear dependency between log probabilities and trajectory duration, the local thresholds enabled the model to flag shorter paths as anomalies also. However, the problem was not solved completely as there was still a significant difference in means of durations for normal an anomalous tracks. As for the global threshold, the local thresholds carry an assumption of 5% anomalous paths in each 4 hour interval of the training set, which is problematic in itself, since the number of trajectories in each interval is not the same. To use these local thresholds properly, additional work should ensure, that the data is fairly equal within each chosen interval.

This indicates that it could be beneficial for the performance of the model, to chose a different metric for flagging anomalies than the log probability of the input. Although previous work has showed improved performance when using reconstruction probability over reconstruction error (An and Cho 2015), it may come with some unwanted side effects for the problem sought to be solved in this thesis. In (Santhosh et al. 2018) they used a t-Distributed Stochastic Neighbor Embedding (t-SNE) of the latent space of the Variational Autoencoder marking video object trajectories far from any cluster of trajectories as anomalies. The idea of clustering trajectories was also introduced in Section 1.1 for many of the other approaches to trajectory prediction and anomaly detection for ships. Using a clustering algorithm on the reconstructed paths could possibly alleviate the importance of trajectory duration.

Another big drawback of the CVAE model is, that it disregards time completely. This poses some additional challenges to the problem of anomaly detection using AIS messages. As already mentioned, differentiating between direction of shipping lanes is not possible, when the model is not told from which direction the trajectory begins. In addition to work as a remedy for the unfortunate significance imposed on the duration of trajectories, letting time being a variable in the model would also allow for it to run in an online setting, unlike the CVAE which is required to run offline. This is a necessity for the CVAE as it uses the full trajectory for predictions, and hence the journey must have adjourned before deploying the model.

To make predictions at every timestep the VRNN model was used. This model was used exactly for the purpose of trajectory prediction and subsequent anomaly detection on AIS messages in (Nguyen et al. 2019a), serving as a strong motivation to employ the model in this thesis. Regardless, the model did not produce useful results.
Initially the model did not seem to learn a useful latent representation of the data manifested by a very small KL divergence as well as low mutual information between the input and latent space and no active units recorded. Given that the latent representation did not seem to learn proper information about the input, it was hypothesized that it could be a result of a posterior collapse, a problem common to variational autoencoders with strong autoregressive decoders.
Following this hypothesis, two skip connections were added to the model, and KL annealing was used to kickstart a proper latent representation. Even though the loss was decreased during training and the mutual information increased as the number of active units increased, the latent representation learned of the data, resulted in a prediction of the dataset mean almost exclusively. Since the mutual information and number of active units had increased, it could indicate that a posterior collapse was no longer the problem, alas the KL divergence was still close to zero. Further investigations included looking at the gradients of the model, as an effort to locate the problematic parts of the network.

Here it was clear that neither the encoder nor decoder seemed to have much gradient signal, serving as indicators that the model was having a hard time learning the complexity of the data.
Batch normalization of the approximate posterior was also used, as experiments in the literature suggested that to be a solution. Even though this seemed to ignite some activity in both the prior and encoder, the gradients of the decoder still did not indicate much learning.
More comprehensive methods against the posterior collapse was found in literature, and future work could include implementing some of these. In both (McCarthy et al. 2020) and (Qian and Cheung 2019) they explicitly let the mutual information be a part of the model objective ensuring that it is increased during training.
Another approach was taken in (Singh 2018). Here they used the fact that a VRNN becomes a Hidden Markov Model (HMM) if the autoregressive connections are removed. Since exact inference is possible for HMMs, but not VRNNs, they optimize the inference network of an HMM and use it to initialize the VRNN, effectively solving the presence of a posterior collapse.
All these approaches handles the posterior collapse and should be further investigated to validate, that it is in fact a posterior collapse that causes the difficulties in learning. Nevertheless, the final implementations did indicate that other factors were in play, since the KL did not vanish, the mutual information rose as well as the number of active units. Given that the gradients of the LSTM also seemed to vanish it could indicate that the temporal development of the paths was never learned. Further supporting this is the reconstructions, which appear very discontinuous. Representing the input data as four-hot vectors was a novel approach introduced by (Nguyen et al. 2019a) and not used anywhere else in literature. One future work experiment should hence include changing the data representation to see how the model performs, when the input is the real-values of longitude, latitude, SOG and COG. To further motivate this investigation are the findings in (Ding et al. 2020) where they also used the VRNN on AIS messages, but let the input be Δt, $\Delta long$, Δlat, ΔSOG and ΔCOG, where Δ indicates the difference of the given variable between two successive timesteps. Their results indicate, that they were able to use the VRNN to reliably predict future positions of a ship.
The preprocessing of the data, should for future work be thoroughly investigated, as some unwanted properties of the trajectories could be introduced in the process. One example of such investigation could include the time interval defined to specify individual tracks and the subsequent resampling. For this thesis the paths where split into two individual paths if the time interval between two successive AIS messages surpassed 4 hours. The trajectory was then resampled with a frequency of 10 minutes using a linear interpolation. This could in theory create tracks, where a big part is an interpolation between two points, which could prove difficult for the LSTM. A more thorough validation of the dataset could hence also possibly alleviate some of the difficulties met by the VRNN.

CHAPTER 5

Conclusion

This thesis used deep neural networks for ship trajectory predictions and subsequent anomaly detection using AIS data. Two models were proposed; a Convolutional Variational Autoencoder, and a Variational Recurrent Neural Network. By using the full trajectory image as input, the CVAE was able to learn a latent space representation of the routes, which enabled reliable reconstructions. The reconstruction probability was used as a metric for anomalous behaviour, and tested on different fabricated trajectories. The model proved very capable of detecting anomalies which are infeasible in reality, but was less consistent for feasible trajectories. One explanation provided was that the reconstruction probability could prove difficult to use as metric for anomalous behaviour for trajectories of varying length. As a readily implemented solution to this, local thresholds based on trajectory duration was used, which partially solved the problem.
The second model was a VRNN which sought to predict points on the trajectories at each time-step. The model was not successful and to the extend of this thesis it was not possible to teach the model to predict time-steps with large deviations from the dataset mean. One explanation provided is, that the varitational autoencoder present in every time-step experiences a posterior collapse, making it impossible for the model to learn a useful latent representation of the input.

List of Figures

List of Tables

Bibliography

[1] Najmeh Abiri. *Statistical Inference with Deep Latent Variable Models.* Tech. rep. Lund University, 2019.

[2] Jinwon An and Sungzoon Cho. "Variational Autoencoder based Anomaly Detection using Reconstruction Probability". In: *SNU Data Mining Center* (2015).

[3] Christoph Baur, Benedikt Wiestler, Shadi Albarqouni, and Nassir Navab. "Deep Autoencoding Models for Unsupervised Anomaly Segmentation in Brain MR Images". In: *CoRR* abs/1804.04488 (2018). arXiv: `1804.04488`.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Berlin, Heidelberg: Springer-Verlag, 2006.

[5] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. "Variational Inference: A Review for Statisticians". In: *Journal of the American Statistical Association* 112.518 (Apr. 2017), pp. 859–877.

[6] N. Bomberger, B. Rhodes, M. Seibert, and A. Waxman. "Associative Learning of Vessel Motion Patterns for Maritime Situation Awareness". In: *2006 9th International Conference on Information Fusion* (2006), pp. 1–8.

[7] Samuel R. Bowman et al. "Generating Sentences from a Continuous Space". In: *CoRR* abs/1511.06349 (2015). arXiv: `1511.06349`.

[8] Per B. Brockhoff, Jan K Møller, Elisabeth W Andersen, Peder Bacher, and Lasse E Christiansen. *Introduction to Statistics at DTU.* 2018.

[9] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. "Importance Weighted Autoencoders". In: *ICLR* (2016). arXiv: `1509.00519v4`.

[10] Ruijin Cang, Hechao Li, Hope Yao, Yang Jiao, and Yi Ren. "Improving direct physical properties prediction of heterogeneous materials from imaging data via convolutional neural network and a morphology-aware generative model". In: *Computational Materials Science* 150 (2018), pp. 212–221. arXiv: `1712.03811`.

[11] L. Cazzanti and G. Pallotta. "Mining maritime vessel traffic: Promises, challenges, techniques". In: *OCEANS 2015 - Genova.* 2015, pp. 1–6.

[12] Xiaoran Chen, Nick Pawlowski, Martin Rajchl, Ben Glocker, and Ender Konukoglu. "Deep Generative Models in the Real-World: An Open Challenge from Medical Imaging". In: *CoRR* abs/1806.05452 (2018). arXiv: `1806.05452`.

[13] Jen-Tzung Chien and Kuan-Ting Kuo. "Variational Recurrent Neural Networks for Speech Separation". In: *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH.* Aug. 2017, pp. 1193–1197.

[14] Junyoung Chung et al. "A Recurrent Latent Variable Model for Sequential Data". In: *CoRR* abs/1506.02216 (2015). arXiv: `1506.02216`.

[15] COM-EU. "Towards the integration of maritime surveillance: A common information sharing environment for the EU maritime domain". In: *Commission of the European Communities* 10.538 final (2009).

[16] Pasquale Coscia, Paolo Braca, Leonardo M. Millefiori, Francesco A.N. Palmieri, and Peter Willett. "Multiple Ornstein-Uhlenbeck Processes for Maritime Traffic Graph Representation". In: *IEEE Transactions on Aerospace and Electronic Systems* 54.5 (2018), pp. 2158–2170.

[17] Adji B. Dieng, Yoon Kim, Alexander M. Rush, and David M. Blei. "Avoiding Latent Variable Collapse With Generative Skip Models". In: *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)* (2019). arXiv: `1807.04863`.

[18] Mengzhen Ding et al. "A Novel Approach on Vessel Trajectory Prediction Based on Variational LSTM". In: *2020 IEEE International Conference on Artificial Intelligence and Computer Applications* (2020), pp. 206–211.

[19] Carl Doersch. *Tutorial on Variational Autoencoders.* Tech. rep. 2016. arXiv: `1606.05908v2`.

[20] Alexey Dosovitskiy, Jost Tobias Springenberg, Maxim Tatarchenko, and Thomas Brox. "Learning to Generate Chairs, Tables and Cars with Convolutional Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.4 (2017). arXiv: `1411.5928`.

[21] Vincent Dumoulin and Francesco Visin. *A guide to convolution arithmetic for deep learning.* Tech. rep. 2016. arXiv: `1603.07285`.

[22] Marco Fraccaro. "Deep Latent Variable Models for Sequential Data". In: *DTU* (2018).

[23] Hao Fu et al. "Cyclical Annealing Schedule: A Simple Approach to Mitigating KL Vanishing". In: *CoRR* abs/1903.10145 (2019). arXiv: `1903.10145`.

[24] Miao Gao, Guoyou Shi, and Shuang Li. "Online Prediction of Ship Behavior with Automatic Identification System Sensor Data Using Bidirectional Long Short-Term Memory Recurrent Neural Network". In: *Sensors* 18 (Nov. 2018).

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* `http://www.deeplearningbook.org`. MIT Press, 2016.

[26] Alex Graves. "Generating Sequences With Recurrent Neural Networks". In: *CoRR* (2013). arXiv: `1308.0850`.

[27] Junxian He, Daniel Spokoyny, Graham Neubig, and Taylor Berg-Kirkpatrick. "Lagging Inference Networks and Posterior Collapse in Variational Autoencoders". In: *ICLR* (2019). arXiv: `1901.05534v2`.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: `1512.03385`.

[29] Tue Herlau, Mikkel N. Schmidt, and Morten Mørup. *Introduction to Machine Learning and Data Mining.* 2018.

[30] Sepp Hochreiter and J J Urgen Schmidhuber. "Long Short Term Memory". In: *Neural Computation* (1997).

[31] Matthew Hoffman D. and Matthew Johnson J. "ELBO surgery: yet another way to carve up the variational evidence lower bound". In: *NIPS Workshop on Advances in Approximate Bayesian Inference* (2016).

[32] Fredrik Johansson and Göran Falkman. "Detection of vessel anomalies - A Bayesian network approach". In: *Proceedings of the 2007 International Conference on Intelligent Sensors, Sensor Networks and Information Processing, ISSNIP* (2007), pp. 395–400.

[33] Ameet V Joshi. *Machine Learning and Artificial Intelligence an Introduction.* 1. Springer, 2020.

[34] Samira Kazemi, Shahrooz Abghari, Niklas Lavesson, Henric Johnson, and Peter Ryman. "Open data for anomaly detection in maritime surveillance". In: *Expert Systems with Applications* 40.14 (2013), pp. 5719–5729.

[35] Diederik P Kingma and Jimmy Lei Ba. "Adam: A method for stochastic optimization". In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* (2015), pp. 1–15. arXiv: `1412.6980`.

[36] Diederik P Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: *2nd International Conference on Learning Representations, ICLR*. 2014. arXiv: 1312.6114v10.

[37] Kira Kowalska and Leto Peel. “Maritime anomaly detection using Gaussian Process active learning”. In: *15th International Conference on Information Fusion, FUSION 2012* (2012), pp. 1164–1171.

[38] Rikard Laxhammar. “Anomaly detection for sea surveillance”. In: *Proceedings of the 11th International Conference on Information Fusion, FUSION 2008* (2008).

[39] Nicolas Le Guillarme and Xavier Lerouvreur. “Unsupervised extraction of knowledge from S-AIS data for maritime situational awareness”. In: *Proceedings of the 16th International Conference on Information Fusion, FUSION 2013* (2013), pp. 2025–2032.

[40] Joun Yeop Lee, Sung Jun Cheon, Byoung Jin Choi, Nam Soo Kim, and Eunwoo Song. “Acoustic modeling using adversarially trained variational recurrent neural network for speech synthesis”. In: *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH* September (2018), pp. 917–921.

[41] Victor Lempitsky. “Autoencoder”. In: *Computer Vision* (2020).

[42] Teng Long, Yanshuai Cao, Jackie Chi, Kit Cheung, and Borealis Ai. *On Posterior Collapse and Encoder Feature Dispersion in Sequence VAEs*. Tech. rep. 2020. arXiv: 1911.03976v2.

[43] Steven Mascaro, Kevin B Korb, and Ann E Nicholson. *Learning Abnormal Vessel Behaviour from AIS Data with Bayesian Networks at Two Time Scales*. Tech. rep. 2010.

[44] Arya D. McCarthy, Xian Li, Jiatao Gu, and Ning Dong. “Addressing Posterior Collapse with Mutual Information for Improved Variational Neural Machine Translation”. In: *Association for Computational Linguistics (ACL)* (2020), pp. 8512–8525.

[45] Warren S. McCulloch and Walter H. Pitts. “A Logical Calculus of the Ideas Immanent in Nervous Activity”. In: *Bulletin of Mathematical Biophysics* 5 (1943), pp. 115–133.

[46] Tomas Mikolov, Stefan Kombrink, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. “Extensions of Recurrent Neural Network Language Model”. In: *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2011), pp. 5528–5531.

[47] Duong Nguyen, Rodolphe Vadaine, Guillaume Hajduch, Rene Garello, and Ronan Fablet. “A multi-task deep learning architecture for maritime surveillance using AIS data streams”. In: *Proceedings - 2018 IEEE 5th International Conference on Data Science and Advanced Analytics, DSAA 2018* (2019), pp. 331–340. arXiv: 1806.03972.

[48] Duong Nguyen, Rodolphe Vadaine, Guillaume Hajduch, Rene Garello, and Ronan Fablet. “GeoTrackNet—A maritime anomaly detector using probabilistic neural network representation of AIS tracks and a contrario detection”. In: *CoRR* (2019), pp. 1–10. arXiv: 1912.00682.

[49] Giuliana Pallotta, Michele Vespe, and Karna Bryan. "Vessel pattern knowledge discovery from AIS data: A framework for anomaly detection and route prediction". In: *Entropy* 15.6 (2013), pp. 2218–2245.

[50] Dong Qian and William K. Cheung. "Enhancing variational autoencoders with mutual information neural estimation for text generation". In: *2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing* (2019), pp. 4047–4057.

[51] Gopinath Rebala, Ajay Ravi, and Sanjay Churiwala. *An Introduction to machine learning.* Springer, 2019.

[52] Bradley J. Rhodes, Neil A. Bomberger, Michael Seibert, and Allen M. Waxman. "Maritime situation monitoring and awareness using learning mechanisms". In: *Proceedings - IEEE Military Communications Conference MILCOM* 2005 (2005), pp. 646–652.

[53] Maria Riveiro, Giuliana Pallotta, and Michele Vespe. "Maritime anomaly detection: A review". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8.5 (2018).

[54] Kelathodi Kumaran Santhosh, Debi Prosad Dogra, Partha Pratim Roy, and Adway Mitra. "Video Trajectory Classification and Anomaly Detection Using Hybrid CNN-VAE". In: *CoRR* (2018). arXiv: `1812.07203`.

[55] Abdoulaye Sidibé and Gao Shu. "Study of Automatic Anomalous Behaviour Detection Techniques for Maritime Vessels". In: *THE JOURNAL OF NAVIGATION* 70 (2020), pp. 847–858.

[56] Rachid Singh. *Sequential Discrete Latent Variables for Language Modeling.* Tech. rep. Harvard University, 2018.

[57] Ava Soleimany. *Deep Generative Models.* 2020. eprint: `http://introtodeeplearning.com/slides/6S191{\_}MIT{\_}DeepLearning{\_}L4.pdf`.

[58] Jinsong Su et al. "Variational recurrent neural machine translation". In: *32nd AAAI Conference on Artificial Intelligence, AAAI 2018* (2018), pp. 5488–5495. arXiv: `1801.05119`.

[59] Enmei Tu, Guanghao Zhang, Lily Rachmawati, Eshan Rajabally, and Guang Bin Huang. "Exploiting AIS Data for Intelligent Maritime Navigation: A Comprehensive Survey from Data to Methodology". In: *IEEE Transactions on Intelligent Transportation Systems* 19.5 (2018), pp. 1559–1582. arXiv: `1606.00981v1`.

[60] Iraklis Varlamis, Konstantinos Tserpes, Mohammad Etemad, Amílcar Soares Júnior, and Stan Matwin. "A network abstraction of multi-vessel trajectory data for detecting anomalies". In: *EDBT/ICDT Workshops 2019* (2019).

[61] Liangbin Zhao and Guoyou Shi. "Maritime Anomaly Detection using Density-based Clustering and Recurrent Neural Network". In: *Journal of Navigation* 72.4 (2019), pp. 894–916.

[62] Qile Zhu et al. *A batch normalized inference network keeps the KL vanishing away.* 2020. arXiv: `2004.12585`.

[63] David Zimmerer, Simon A.A. Kohl, Jens Petersen, Fabian Isensee, and Klaus H. Maier-Hein. "Context-encoding variational autoencoder for unsupervised anomaly detection". In: *CoRR* (2018), pp. 1–13. arXiv: `1812.05941`.

[64] Dimitris Zissis, Konstantinos Chatzikokolakis, Giannis Spiliopoulos, and Marios Vodas. "A Distributed Spatial Method for Modeling Maritime Routes". In: *IEEE Access* 8 (2020), pp. 47556–47568.