NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

SHIPS' TRAJECTORIES PREDICTION USING RECURRENT NEURAL NETWORKS BASED ON AIS DATA

by

Shay Paz Liraz

September 2018

Thesis Advisor:	Lyn R. Whitaker
Co-Advisor:	Matthew Norton
Second Reader:	Robert A. Koyak

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2018	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE SHIPS' TRAJECTORIES PREDICTION USING RECURRENT NEURAL NETWORKS BASED ON AIS DATA	**5. FUNDING NUMBERS**
6. AUTHOR(S) Shay Paz Liraz	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

The objective of this research is to develop a method for predicting the future behavior of ships and detecting anomalous behavior based on their past location coordinates and a set of context features. We use a Recurrent Neural Network model with inputs extracted from Automated Information System (AIS) data. This data includes ship coordinates, speed and course, and the ship's call sign, size, and type. These features are appropriately encoded to amplify significant predictive structures within the data. The ability to automate the task of track prediction and the process of detecting anomalous ship behavior serves to increase maritime domain awareness and aid security analysts in deciding how to best allocate limited resources. Furthermore, these capabilities enable the investigation of potential threats, prevention of collisions, and planning for search-and rescue missions.

14. SUBJECT TERMS Recurrent Neural Networks, AIS, trajectories prediction, maritime assistance, RNN, embedding, LSTM, TensorFlow	**15. NUMBER OF PAGES** 97
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

SHIPS' TRAJECTORIES PREDICTION USING RECURRENT NEURAL NETWORKS BASED ON AIS DATA

Shay Paz Liraz
Captain, Israel Army
BS, Hebrew University, 2011

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 2018

Approved by: Lyn R. Whitaker
Advisor

Matthew Norton
Co-Advisor

Robert A. Koyak
Second Reader

W. Matthew Carlyle
Chair, Department of Operations Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The objective of this research is to develop a method for predicting the future behavior of ships and detecting anomalous behavior based on their past location coordinates and a set of context features. We use a Recurrent Neural Network model with inputs extracted from Automated Information System (AIS) data. This data includes ship coordinates, speed and course, and the ship's call sign, size, and type. These features are appropriately encoded to amplify significant predictive structures within the data. The ability to automate the task of track prediction and the process of detecting anomalous ship behavior serves to increase maritime domain awareness and aid security analysts in deciding how to best allocate limited resources. Furthermore, these capabilities enable the investigation of potential threats, prevention of collisions, and planning for search-and rescue missions.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

ADAM	Adaptive Moment Estimation
AIS	Automated Information System
BOEM	Bureau of Ocean Energy Management
COG	Course over Ground
CPU	Central Processing Unit
DOE	Design of Experiments
GPU	Graphical Processing Unit
HPC	High Performance Computing
IMO	International Maritime Organization
kNN	k Nearest Neighbors
LA	Los Angeles
LSTM	Long Short-Term Memory
METOC	Meteorological and Oceanographic
MMSI	Maritime Mobile Service Identity
NaN	Not a Number
NOAA	National Oceanic and Atmospheric Administration
NN	Neural Network
NOLH	Nearly Orthogonal Latin Hypercube
NPS	Naval Postgraduate School
PCA	Principal Component Analysis
ReLu	Rectified Linear Unit
RMSprop	Root Mean Square Propagation
RNN	Recurrent Neural Network
SF	San Francisco
SGD	Stochastic Gradient Descent
SLURM	Simple Linux Utility for Resource Management
SOG	Speed over Ground
UTC	Coordinated Universal Time
UTM	Universal Transverse Mercator

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

In this work, we use Recurrent Neural Networks (RNN) to predict vessel movements based on recent travel history in the form of geographic time series data. Using Automated Information System (AIS) data, we construct models that make accurate short- (1-minute), medium- (30-minute), and long-term (100-minute) predictions of future vessel location given recent travel history. We implement two primary approaches for model construction, posing the predictive task first as a classification problem and second as a regression problem. After implementing multiple variations of a deep RNN, we find that a classification approach, predicting discretized bearing and distance classes, works best, achieving the most accurate predictions of future position with a mean miss distance of 8, 670, and 2,795 meters for a 1-minute, 30-minute and 100-minute prediction horizon, respectively. Nevertheless, we still find that a successful regression approach can be formulated based on a clustering scheme, where regression targets are limited to a convex combination of engineered landmark points. By nature of the regression problem, the predicted vessel tracks are much smoother than those made by the discretized classification approach.

For our neural network (NN) design, we use a Nearly Orthogonal Latin Hypercube (NOLH) design for hyper-parameter selection using a two-phase search process. We implement the models using the Python language and the Keras library with a TensorFlow backend. Training the models is done using the Naval Postgraduate School (NPS) High Performance Computing center facilities, including a computing cluster running the training jobs in parallel over dozens of nodes, using 16 Nvidia Graphical Processing Units (GPUs) with 120 gigabytes.

This type of model, which uses widely available AIS data to predict the future vessel behavior, has applications for enhancement of maritime awareness. For example, collision prevention and assistance with search-and-rescue lost vessel missions are two such applications. We briefly highlight the use of such predictive systems for another purpose, specifically anomaly detection, briefly exploring the use of prediction error as a real-time tracking mechanism for detecting improper vessel activity. Such ability to detect abnormal behavior is useful in automatically classifying suspicious ships that might engage in criminal

activity, piracy, or terror and ships suffering from an emergency that prevents them from following their normal conduct.

Overall, we find RNNs to be a viable method for predicting maritime activity. With RNNs proving highly flexible, the potential for extending this work and incorporating new data sources is high, which is expected to improve the RNN's predictive power further, making it a useful and valuable tool for enhancement of maritime domain awareness.

ACKNOWLEDGMENTS

I would like to thank my wife for the great love and support I never stopped receiving throughout the research period (and always). I thank Bruce Chiarelli for his tremendous support with using the HPC, and Professor Samuel Buttrey for data preprocessing.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

With the world's oceans covering approximately 71 percent of the earth's surface, it is no surprise that over 90 percent of the world's trade is carried by sea, as reported by the International Maritime Organization (IMO) (Tu et al. 2016). With global trade volume on the rise and maritime traffic increasing correspondingly, the risks associated with maritime operations are growing. Congestion, for example, increases the likelihood of collisions and presents exploitative opportunities for terrorists and piracy groups. The clutter of "normal" ships might disguise maritime criminal activities and veil ships that are under emergency conditions. To combat such activity, security analysts and maritime domain awareness experts search for automated algorithms to predict future trajectories of ships and to detect anomalous behaviors to help them decide how to best allocate their limited resources.

The development of modern automated algorithms has been driven by the availability of rich data resources and collection mechanisms. For the maritime domain, the Automated Information System (AIS), an information system that accumulates location transmissions from ships and aircraft around the globe, provides one such resource. According to Al-Molegi et al. (2016), starting from 2002, due to the requirements of the IMO, this system provides the past and current whereabouts of most ships accessible (i.e., passenger ships of all sizes and any other ships heavier than 300 tons) and includes their velocity, bearing and other information. While the system's foremost use is preventing collisions, AIS is also important and useful in investigating accidents, increasing maritime awareness, and managing traffic.

The AIS data provides an incredibly large set of observations, as more than 70,000 ships around the globe are continuously transmitting their information (Marine Traffic 2018). This data is a leading enabler in developing advanced algorithms to predict future behavior of ships. In particular, this enables the development of "data-hungry" models such as neural networks. These highly flexible universal approximators, while often proving too flexible for small-data regimes, have been shown to excel in prediction of complex behaviors when given large amounts of data (Leca et al. 2015; Tu et al. 2016).

A Recurrent Neural Network (RNN) is a type of neural network model that holds internal memory which can be used to process input sequences of arbitrary length and is therefore suitable for time series data. RNNs are widely used in automated speech and handwriting recognition as well as rhythm learning and even music composition (Eck and Schmidhuber 2002; Gers et al. 2002; Graves et al. 2009). Thus, RNNs are a natural candidate for predicting the behaviors of vessels at sea.

We use RNNs to predict the future behavior of ships based on their past location coordinates or tracks and a set of "context" features associated with the ship extracted from the AIS data. We focus on formulating a model that can be easily enhanced to include future data sources and an enriched input space.

B. LITERATURE REVIEW

The following passages describe artificial neural networks, their common types and categories, the key challenges in using them, and short review of applications of artificial neural networks to track-like data.

1. Artificial Neural Networks

Artificial neural networks (NN) are mathematical models for information processing and pattern recognition. As Graves (2012) describes, an artificial neural network's basic structure is a directed network of independent processing nodes that are joined using weighted connections. The nodes represent neurons and the connections represent the synapses between them. Activation of the network is done by providing an input, which then propagates from the input nodes and through the rest of the network. An important distinction in NNs is between cyclic and acyclic. Cyclic networks are usually referred to as feedback, recursive, or recurrent NN and are used extensively in this research. Acyclic networks are called feedforward NNs, such as the multilayer perceptron networks, developed and researched by Rumelhart et al. (1985), Werbos (1988), and Bishop and Bishop (1995).

NNs are usually described by layers. Each layer of neurons is connected with the previous and the next layer's neurons. Neurons in the same layer do not communicate. The layers between the input layer and the output layer are considered the hidden layers. Each

neuron input is composed of a weighted sum of the output coming from the connected neurons in the previous layer, plus a bias constant. The neuron output to the next layer is its input value fed to its activation function. The activation function can be a linear function, but is more commonly sigmoid, rectified linear (ReLu), or some other nonlinear function.

A common activation function for the output layer in classification NNs is the *softmax* activation function (Bridle 1990). This function "squashes" the output values so that they are in the range (0, 1] and sum to 1, and so that each value represents the predicted probability of that class.

As described in Goodfellow et al. (2016), NNs are universal approximators that typically use supervised learning to train their parameters (weights and biases) to their given task through an optimization process. The optimization process tries to minimize the loss over a set of data points called the training set. Typically, the optimizers use some version of stochastic gradient descent. NNs use a variety of loss functions, including the mean or absolute squared error, categorical cross entropy, and cosine proximity. In many cases customized loss functions are used to accommodate a given problem. Such is the case in this research as shown in Chapter III.

Many types of NN layers have been developed and are used to process images, text, voice, and other types of information. As this work deals with RNNs, an important layer type is a recurrent layer called Long Short-Term Memory (LSTM). As Goodfellow et al. (2016) explain, these units have somewhat more complex structure than a simple neuron. Apart from "looping" some of their output back to themselves as input in the next time step, which is common to all recurrent units in RNNs, LSTMs have a set of gates that control their internal state. LSTMs control how much of the new current input is going to be taken in using the input gate. They govern how much of the previous state will be looped back to the current state using a forget gate and how much of the internal state is going to be output using the output gate. All of these gates are influenced by the internal state of the LSTM unit, which is composed of trainable parameters. A diagram of an LSTM unit is provided in Figure 1.

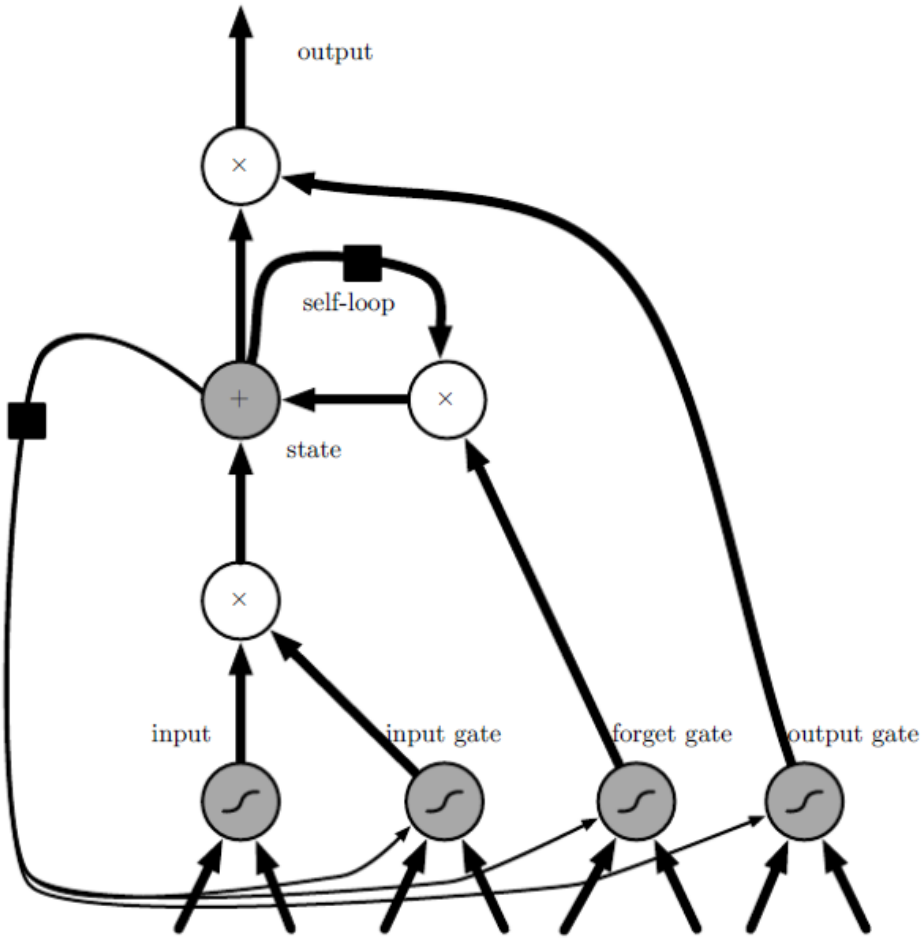

Figure 1. Diagram of an LSTM unit.
Source: Goodfellow et al. (2016)

RNNs can take sequential input such as a time series of stock prices, a sentence or document, a series of image frames from a video, or a sound recording, and they can also take a fixed size input of any length. In the same manner, their output might be a single prediction based on the entire input series, a fixed length series of predictions, or a nonfixed length output. We use Karpathy's (2015) example of the flexibility of RNN architectures in Figure 2. The red rectangles represent the RNN input and the blue rectangles represent the output.

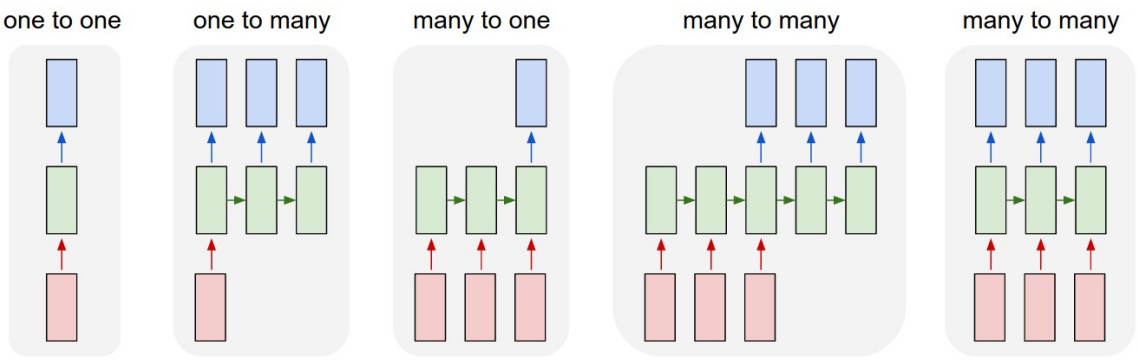

Figure 2. The flexibility of RNN architectures.
Source: Karpathy (2015)

As Karpathy (2015) explains, an RNN can use a one-to-many architecture for image captioning where the input is a single image and the output is a sentence of variable length. It can be a many-to-one architecture, such as sentiment analysis of a sentence. It can also use a many-to-many architecture, such as the case in this work, where the input is a ship track of variable length and the output is a series of predictions whose length is determined by the input track length.

2. Families of Neural Networks

Neural network models can be, roughly speaking, divided into two model families according to their targets and outputs: classification and regression. A classification model outputs a class, or label prediction. For example, a picture might be classified as "Cat" or "Dog," or a sentence can be classified as having "Positive," "Negative," or "Neutral" sentiment. Often, the prediction will be specified in the form of a score vector, with each element of the score vector indicating the probability that the given example belongs to each potential class. The last layer of the network, which provides these probabilities, will typically be the size of the number of classes to predict (two and three, respectively, in the previous examples), and the value at each node will represent the probability that the input will match each of the classes. In many cases the last layer is activated by a softmax function. A regression model's output is numeric. For example, an NN can predict the future price of stocks based on previous values or predict the height of a child based on the child's age and

shoe size. In contrast to a classification model, the output is not a probability but the actual estimated value in question (e.g., the predicted height of the child).

The two approaches differ in the types of problems they try to solve, and both have their advantages and disadvantages. Some problems, however, can be modeled by both approaches depending on how the modeler chooses to represent the input data and output (target) data. As we will see, this is the case with predicting ship trajectories. We train RNNs that treat a future ship location as numeric (i.e., with two output units, one for latitude value and one for longitude value). We also train RNNs that treat future ship location as categorical, using a classification scheme.

3. Challenges in Training Neural Networks

Neural networks are flexible models, but with an array of design choices. Networks with only a few hidden layers are universal function approximators. For modern pattern recognition tasks, however, state of the art networks can often be hundreds or even thousands of layers deep (Shazeer et al. 2017). In addition to selecting the number of layers, one must also choose from many types of layers, with each layer being composed of a potentially different number of units. Then, once this aspect of the architecture has been selected, there are many ways to formulate a loss function for a specific problem and a wide variety of optimization techniques (e.g., variations of stochastic gradient descent) to train the network to minimize the chosen loss function via training examples (Pascanu et al. 2013).

The learning itself is also governed by a number of hyper-parameters, among which is the learning rate parameter. A high learning rate means that every gradient descent step may be a large leap in the gradient direction. This might speed up learning initially, but set too high, the learning becomes unstable, failing to converge to any steady state. Many techniques have been developed to deal with this issue, such as the introduction of a momentum term that enforces continued movement partially in the direction of previous gradients. Other innovations include adaptive learning rate techniques and learning rate schedulers, changing the learning rate along the training process (using a decay factor) or in a way that is responsive to the error over the training set and a validation set (Goodfellow et al. 2016).

The performance of a network is sensitive to the choices of hyper-parameters, and thus the process of hyper-parameter tuning is considered to be an important and challenging task. This is ever more so in big networks, which might have many millions of weights to be learned, and training data sets the size of which might be measured in terabytes. Careful selection of these hyper-parameters is made even more important by the fact that training big networks with a single selection of parameters may take days or weeks, even when using high performance computing clusters (Goodfellow et al. 2016).

This calls for methods to help explore the vast space of hyper-parameters. Using experimental design techniques (Kleijnen et al. 2005) can drastically reduce the number of experiments that need to be conducted in order to learn the approximate shape of a response surface in a high dimensional space. Response surface methodologies and analysis techniques can be used to find the "hot-spots" of the hyper-parameters that yield the best networks that can then be explored further (Gunst, 1996).

In general, NNs achieve better performance when their input is rich with contextual information. However, data in its raw form is often suboptimal in its representation, not providing a contextual description of the data that an NN can understand in the context of the given pattern it is attempting to recognize (i.e., learn). This applies to both the input and output of an NN. Thus, engineering appropriate context features for the NN input, and engineering the appropriate features that should be predicted, is critical to NN training. For example, regression problems are often approximated as multi-class classification problems, which can be easier to learn with a more focused prediction task.

To find the right type of input and the best input representation, attention must be given to the specific idiosyncrasies of the data available for learning and in many cases, extensive data processing and feature engineering have to take place to allow effective machine learning (Kubat et al., 1998). Data processing and feature engineering, which are discussed in Chapters II, III and IV of this work, are critical to our contribution.

4. Forecasting AIS Tracks

Related work dealing with motion prediction of vessels at sea using AIS data typically utilize clustering of tracks or probabilistic models. Ristic et al. (2008) utilize adaptive kernel

density estimation, predict motion and try detecting anomalies using a Gaussian sum tracking filter. Bomberger et al. (2006) use an associative learning algorithm that uses a grid of possible ship locations and assign weights to connections between grid locations using gated Hebbian learning. Pallotta et al. (2013; 2014) use clustering algorithms to define waypoints and identify trajectories between them, using only the ship coordinates and ignoring other differences between the ships. They then form predictions based on Ornstein-Uhlenbeck stochastic processes (Gardiner 2009) whose parameters are estimated with the extracted trajectories. Wijaya and Nakamura (2013) use a simple k-nearest neighbors (kNN) clustering to locate similar tracks and predict future location according to them. Mascaro et al. (2014) use dynamic and static Bayesian Networks learned from AIS data to detect anomalous behaviors. Mazzarella et al. (2015) expand this work and use a Bayesian vessel prediction algorithm based on a particle filter and prior knowledge of traffic routes. Young (2017) uses trajectory clustering, random forests and simple fully connected dense NN to predict future locations of ships. He also notes that the Ornstein-Uhlenbeck approach by Pallota et al. (2013) works well for straight ship tracks, but fails for curved tracks.

In related problems, RNNs have been used in recent years for rain precipitation nowcasting (Xingjian et al. 2015), predicting the trajectories of hurricanes in the Atlantic (Kordmahalleh 2015), predicting a person's next location (Al-Molegi et al. 2016; Liu et al. 2016), and classifying ships' fishing activity based on AIS data (Jiang et al. 2017). To our knowledge, using RNNs to predict ships' trajectories based on AIS data is a novel approach that has yet to be considered.

C. RESEARCH METHODS

To predict the future behavior of ships, we use an array of RNN architectures that take as input the ship location, speed, bearing, unique call sign, dimension, and type. In essence, we use RNNs to learn the spatio-temporal dependencies in the data structure.

The data used to train the models is AIS based and includes 17,647 ship tracks from the years 2016 and 2017 in the Los Angeles region (south and central California coast). The data is preprocessed, and two different representations of the input data are used.

Training of the models is done with a computer cluster at the high-performance computing center (HPC) at the Naval Postgraduate School (NPS) using the Python programming language written open-source software Keras (Chollet 2015) with a Tensorflow (Abadi et al. 2016) neural network training engine. The models are then evaluated and compared over a separate test set of ship tracks from the same time frame in the same region.

D. RESULTS

Overall, we show that an RNN can accurately predict vessel movements in the form of geographic time series data. Using AIS data, we construct models that make accurate short, medium, and long-term predictions of future vessel location given recent travel history. We implement two primary approaches for model construction, posing the predictive task first as a classification problem and second as a regression problem. After implementing multiple variations of a deep RNN, we find that a classification approach, predicting discretized bearing and distance classes, works best and achieves the most accurate predictions of future position. Nonetheless, we still find that a successful regression approach can be formulated based on a clustering scheme, where regression outputs are limited to a convex combination of engineered landmark points. By nature of the regression problem, the predicted vessel tracks are much smoother than those made by the discretized classification approach.

We briefly highlight the use of such predictive RNNs for anomaly detection, exploring the use of prediction error as a real-time tracking mechanism for detecting improper vessel activity. We focus on important RNN design choices, such as layer construction, inclusion of secondary data, and specific forms that input features and output features can take. For example, we implement Nearly Orthogonal Latin Hypercubes (NOLH) to perform hyper-parameter selection, and comment upon our findings. Additionally, we find that inclusion of secondary data can be both beneficial and detrimental, and is highly dependent on the overall prediction task.

E. OUTLINE

The remaining chapters are organized as follows. In Chapter II we review the AIS data collection and preparation process, transforming it into a form that fits the neural network model architecture. In Chapter III, we discuss the considerations taken in designing the

models, with emphasis on their input and output representations, and present the final architecture used for prediction of ship trajectories. Chapter IV deals with the experimental set-up and the model analysis and evaluation, where we compare the results of numerous model architectures over the test set. Chapter V holds a discussion about the methods we use and proposes topics for additional research and future work. We present our conclusions in Chapter VI.

II. DATA COLLECTION AND PREPARATION

A. DATA DESCRIPTION

AIS, which stands at the heart of this research, is an information system that accumulates location transmissions from ships and aircraft around the globe. According to Al-Molegi et al. (2016), this system makes the past and current whereabouts of most ships accessible and includes their speed and bearing. While the system's foremost goal is preventing collisions, it is also important and useful in investigating accidents, enhancing maritime awareness, and managing traffic.

Most data extracted from AIS transmissions falls into two categories, static and dynamic. In short, the dynamic data is broadcast in short intervals of two to ten seconds whenever a ship is sailing and once every 180 seconds when anchoring (U.S. Coast Guard Navigation Center 2018). It includes the following major fields:

- Maritime Mobile Service Identity (MMSI): a unique nine-digit numeric identifier used to identify a specific AIS transceiver, seldom changes.
- Coordinates: the ship's position; latitude and longitude in degrees.
- Time stamp: Coordinated Universal Time (UTC) format of the time at transmission.
- Bearing: the ship's direction relative to the magnetic north in degrees.
- Course over ground (COG): the ship's direction relative to the absolute north given in degrees.
- Speed over Ground (SOG): the ship's velocity in nautical miles per hour.

A vessel is required to manually broadcast a static message every six minutes. A static message provides information that does not change with the motion of the ship. The full list of attributes of a static message can be obtained at the U.S. Coast Guard Navigation Center website (2018). Of interest to our work are the following fields:

- Maritime Mobile Service Identity (MMSI): same as in the dynamic data.
- Name: the ship's name as shown on its station radio license (not fixed).
- Type of Ship and Cargo: numeric codes to classify ship type.
- Overall Dimensions: a set of distances from the center of the ship to its boundaries.
- Destination: the ship's next destination, updated by the ship's operator.

The accumulation of AIS data over time produces a large data set. As Young (2017) describes and as is evident from online AIS-utilizing websites such as MarineTraffic.com (2018), more than 70,000 ships around the globe transmit AIS data at high rates on any given day.

The AIS data set we use is publicly available through a joint venture between the Bureau of Ocean Energy Management (BOEM) and the National Oceanic and Atmospheric Administration (NOAA). MarineCadastre.gov (2018) provides one-minute interval AIS data along the coasts of the United States, in Universal Transverse Mercator (UTM) zones 1–20 (a third of the globe) for 2009 through 2017.Although MMSIs are encrypted and call signs removed between 2010 and 2014. They are available after 2014, and after 2015, the data is regarded as being of better quality (MarineCadastre.gov 2018).

Since we wish to use the MMSIs and call signs, we work with two years-worth of data from 2016 to 2017. Although in principle the AIS data should be very frequent in time (dynamic data is transmitted every ten seconds or less), we find that the available AIS data is much less frequent. The time gap between transmissions is not constant and while it usually varies between one to three minutes, the gap can be longer than an hour. This variability in transmission intervals requires more preprocessing that is described next.

B. DATA PROCESSING

AIS data requires preprocessing to "clean" the data and to shape it for use as input to an RNN model. We discuss the need for preprocessing and the processing done to meet these

needs in the following passages, while leaving some of the more advanced data processing to Chapter III.

1. Defining Ship Tracks

The raw data is in the form of one record per AIS transmission. For dynamic data, one record includes the MMSI, time stamp and ship location at that time. Since we are interested in the ship's tracks, the first stage of data processing is to assemble a ship's consecutive coordinates into tracks. These tracks might include months of sailing, with multiple anchoring points. For the sake of learning ship behavior, we find it better to partition long tracks into shorter ones, where the ship is moving.

As mentioned earlier, the time stamps are not at fixed intervals. Most our records are from one to three minutes apart, and some are many minutes or even hours apart. We take only the more frequently recorded tracks, excluding tracks where there are time gaps larger than ten minutes. This is more appropriate for the task at hand, which is prediction of short-term ship behavior. To account for the variable time intervals, we interpolate the coordinates, speed, and bearing for every round minute (as in 13:45:00, 13:46:00, 13:47:00, etc.) and use this as the base data set.

The next stage of preprocessing is to focus attention on a particular area of interest. The motivation for restricting the data to a certain geographical region is rooted in the behavioral attributes of ships. We expect ships in different areas to behave differently. A ship in mid-ocean will tend to move in straight lines, while near the coast there are many more constraints that must be considered. Other environmental factors, such as weather (for example, in the Arctic Sea), threats from pirates near Somalia, a tension near the coast of Korea etc., may also affect ship's behavior. Therefore, we focus on learning these local behaviors in one area rather than attempting to learn ships' behaviors globally. Another reason for using only a subset of worldwide AIS data is to reduce the sheer scale of data to a more workable size. As discussed in the following chapters, there are also opportunities to use "transfer learning" to better the learning rate and model results in other geographical areas (Goodfellow et al. 2016).

In this thesis we predict the future behavior of ships based on their past location coordinates and a set of "context" features concerning the ship to help in identifying anomalous behavior at sea. We perceive that the most important zone for anomalous behavior identification is near the coast, near ports. This is where smuggling operations, terror plots, and ships that for some technical difficulty have lost the ability to stay on course might cause the most damage.

For our experiments, we focus on tracks that pass by the Los Angeles (LA) coast region. More accurately, any track that cross the rectangle displayed in Figure 3 is included in the analysis.

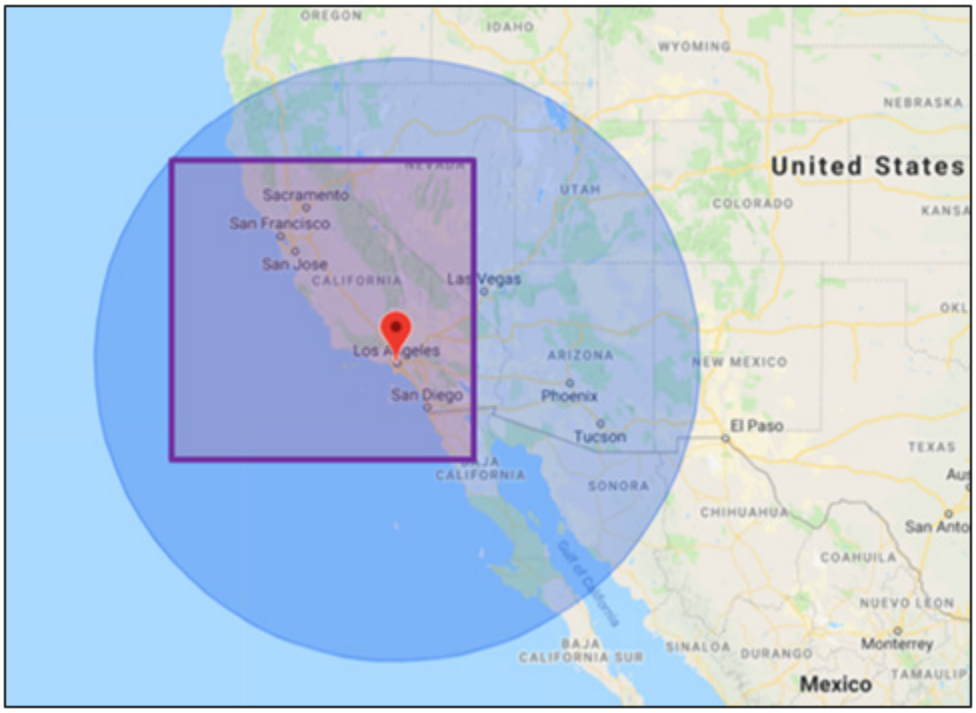

Figure 3. Area of interest where only tracks that cross the rectangle are included in the data and those tracks may extend to the circle boundaries.

Some of the tracks included are from far away or are destined to anchor on the other side of the world. To avoid including the very distant parts of the tracks, we cut out the parts

of every track that are farther than 1,000 km (620 miles) from LA (described by the circle boundaries in Figure 3).

Next, to establish ship type, we take the most recent static data record prior to the time stamp of the beginning of the track and match the MMSIs. We leave only the ship types that are of interest to us, ship types 60–89, 1003–1004, 1012–1017 and which include all types of cargo ships, tankers, and passenger vessels (MarineCadastre.gov 2018). Fishing vessels, high-speed boats, and other special category types are omitted since they behave differently from the majority of ships; they might stop frequently, move in circles and return to their origin port without visiting another. Fishing activity trajectories are sufficiently different from transportation vessels to allow classification based on their behavior, as has been demonstrated by Jiang et al. (2017). This activity is less interesting from our perspective and might be harder to predict. Yet, our methods can easily be applied to ships of any kind, including fishing vessels. We simply focus on the subsets that are of greatest immediate interest for demonstration of the proposed methods.

Ships might anchor for a while for different reasons. They might be stopped at the entrance to a port waiting for their turn to harbor, or for some technical problem, bad weather, etc. We only keep portion of ship tracks where the ship is moving constantly. Any ship missing dynamic records for longer than a two-hour period is a sign to partition the track into two different tracks, before the halt and after. If the resulting segmented tracks are shorter than 300 minutes, we do not use them for the analysis.

2. Dealing with Errors and Missing Data

As with any real data set that is automatically compiled at such large scales, there are many challenges with errors and missing data. Any field of data that is to be manually input (much of the static data) is prone to errors, either by mistake or to intentionally mislead. Within our subset of data, much of the dynamic data is also erroneous.

Harati-Mokhtari et al. (2007) discover various difficulties in the implementation and management of the AIS, which leads to many errors in the resulting data. They point to errors in the MMSI, ship type, ship dimensions, destinations, etc.

These types of errors persist in more recent AIS data sets, as indicated by Young (2017). Young claims that the Speed over Ground (SOG) field, for example, is particularly erroneous, so instead we interpolate the speed using of the coordinates and the time stamps. Nevertheless, the coordinates might have errors as well, so we ensure the average speed between every two consecutive points in the track is below a reasonable velocity threshold, taken to be 60 km/hour. Exceedingly high speed might also be the result of entering a wrong ship type code, indicating a cargo ship instead of a patrol aircraft. Either way, whether the error is in the ship type, coordinates, or time stamps, we decide to omit these cases.

Missing data is another issue to be dealt with. To maximize the amount of data utilized, we only omit tracks where an important data field is missing. These include MMSI (0.1% missing), coordinates (0.3% missing), time stamp and ship type (5% missing).

The final data set after the cleaning process includes 17,647 tracks in the range of 300 to 3,000 time steps (minutes), which gives 20,942,915 data points. The average track length is 1,175 time stamps and the median is 1,051 time stamps. We work only with passenger, cargo, and tanker ships of which we have 947, 13,239, and 3,461 tracks, respectively. The average speed of the ships is 25.75 km/hour (14 knots). Figure 4 presents the frequency and the cumulative frequency of track lengths.

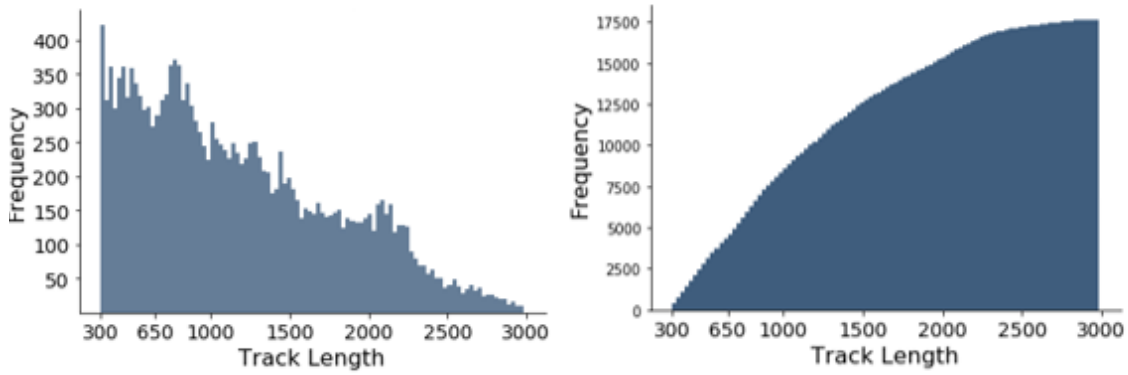

Figure 4. Track length frequency and cumulative frequency distribution

In Figure 5 we plot the entire data set by the tracks' coordinates in a heatmap format. The left map covers the entire area of interest, and the right map takes a closer look at the LA region; warmer colors represent denser areas with more tracks.

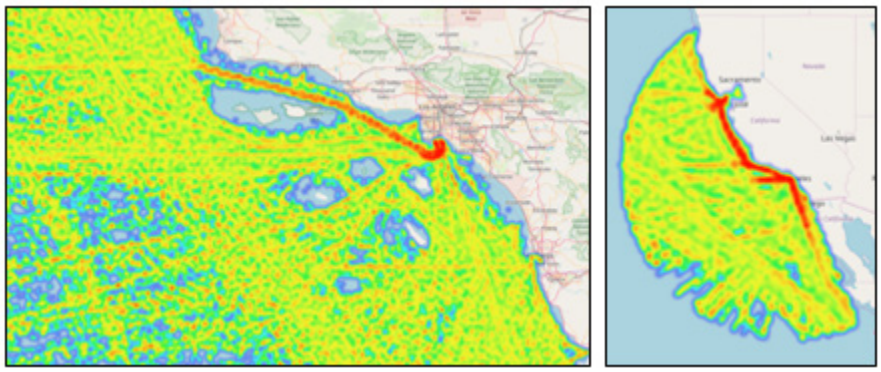

Figure 5. Heatmap of all tracks' coordinates

The ship dimensions are calculated using the AIS static data of ship length and width (given as distances from a fixed point on the ship to fore, aft, port and starboard), that are used to get an estimate of the ship area. The frequency distribution of ship area is given in Figure 6. Missing values (approximately 23%) are considered as zero.

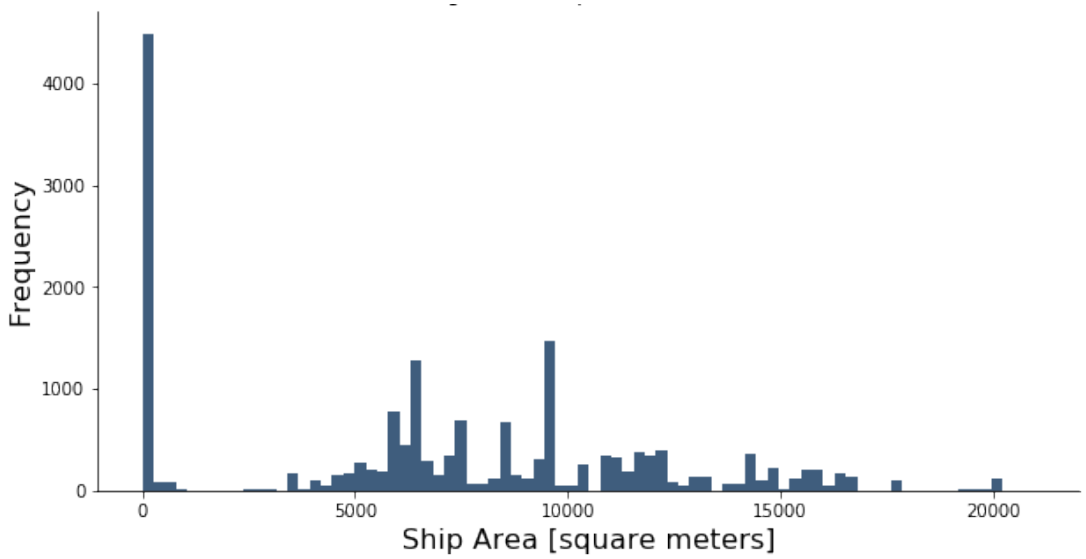

Figure 6. Histogram of ships' area

THIS PAGE INTENTIONALLY LEFT BLANK

III. MODEL DESIGN

The objective of this thesis is to develop RNN models to predict future behavior of ships based on their past location coordinates and a set of "context" features concerning a ship and its surroundings.

The problem of predicting ship trajectories can be modeled as either a regression or classification task, depending upon how the modeler chooses to represent the output. For example, one could attempt to predict the exact coordinates (i.e., as real values) with the regression approach. Alternatively, one could reduce the entire map to a finite number of landmark points and predict which point the ship will be closest to at any given time, using a classification approach. This chapter describes in detail both a classification and a regression approach.

The past and future ship locations input and output representations may be of different data types. A model can take as an input the exact coordinates of the ship in the past and try to predict its next location out of a finite number of landmark points (classification) or conversely, the inputs might be categorical indicating the closest landmark and the outputs exact numeric locations. During the research process we tried many schemes of input and output representations. In this chapter we present the two most successful approaches. Appendix A describes other approaches.

A. INPUT REPRESENTATION

The basic component of our models' input is the time series of coordinates defining a ship's track. Additional inputs are the speed and bearing along the track and static data such as the ship type, area, and call sign (in place of MMSI for ship identification). In both regression and classification models we feed the coordinates as numeric values to the RNN. As recommended by Hastie et al. (2009) and Goodfellow et al. (2016), we standardize the coordinates to have zero mean and unit variance (separately in each dimension).

Following De Brébisson et al.'s (2015) recommendation, we shape the input data so that it is possible to feed more than a single point for each time step. This forms a window of N successive coordinates that shifts along the track by one point as each RNN time step. This

means, for example, that using a window size of 10, at the 134th time step, the input is not only the 134th location of the ship along the track, but the locations at a set of the ten time steps, 125 through 134. This helps the RNN to capture turns, speed, and heading changes.

In addition, we use the speed and bearing as input. The speed is standardized in the same manner as the coordinates, and the bearing is in radians, ranging from 0 to 2π, where 0 is east, going counterclockwise. At each time step we only use the speed and bearing of that time step as input.

The window of coordinates, the speed, and the bearing are then combined to form the "dynamic input." At every time step of the RNN, the dynamic input is of size $2N + 2$. For example, using a window of 15 yields an input size of 32 features.

The static data, ship type, area, and call sign do not change over the course of the sail and have attributes that require a different kind of representation. The ship type is categorical with potentially hundreds of levels. As described in Chapter II, we take only passenger, cargo, and tanker ships. Since there are many sub-classes in each of the three types, we merge all these variants into the main three categories. There are no missing values in the ship type data.

The proxy for ship size, ship area, is then partitioned into 20 equal length intervals (0,1000], (1000,2000],…,(19000,20000], as shown in Figure 7. We dedicate a separate category to missing values, which compose approximately 23% of this data type.

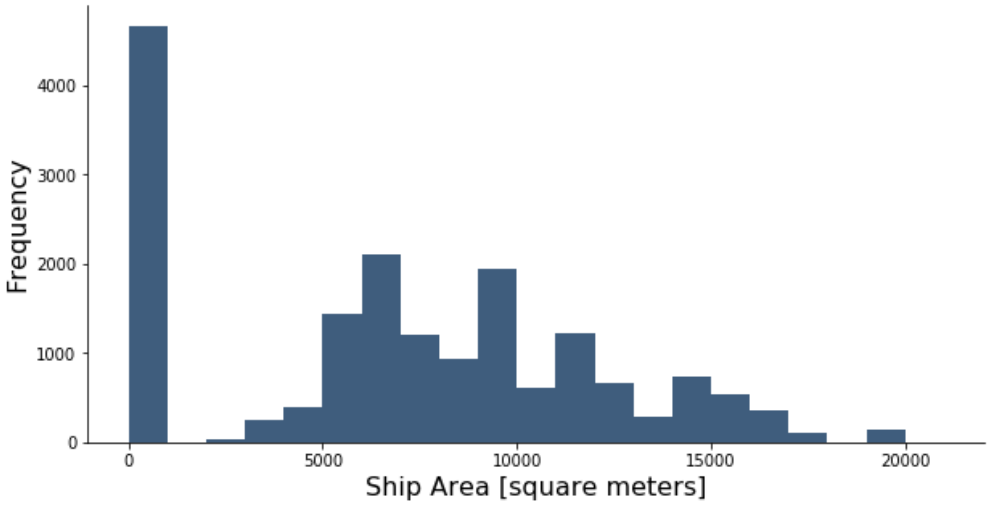

Figure 7. Frequency distribution of ships' area

To identify a ship, one could use either the MMSI or the call sign. There are 1,137 unique MMSI identifiers and 1,141 unique call signs. While MMSIs are not supposed to change, call signs might change in the long run, as happens to up to four ships during the two years' worth of data we are using. Such changes might indicate a change in the ship destined activity, and so we decide to use the call sign and not the MMSI as a ship identifier.

According to Goodfellow et al. (2016), the common practice is to express a categorical feature with *C* classes as a "one-hot" vector of length *C* where all values are zero, except one entry where the value is 1, indicating the class of the given categorical variable. For example, assume there are only three call signs (*C*=3), "9V8009," "V7MT6," and "CQKU." A one-hot vector representation for the three call sighs is given in Table 1.

Table 1. one-hot vector example

Call Sign	one-hot vector representation
"9V8009"	(1, 0, 0)
"V7MT6"	(0, 1, 0)
"CQKU"	(0, 0, 1)

Our approach is to continuously feed the static data into the model at every time step. At each time step we build in linear embedding layer depicted in Figure 8, to reduce the 1,165 features from the combined one-hot static input vector to a vector of 20 features. The static features and the units in the embedding layer are not fully connected. Ship type, ship area and call sign input features only connect to their respective embedded layer units. Table 2 gives the number of embedded layer units of each type.

Table 2. Data types and associated embedding sizes

Data type	Number of possible values	Embedding size
Ship Type	3	3
Ship Area	21	5
Call Sign	1,142	12

The dynamic input features and the embedded static features then feed into the model hidden layers, also depicted in Figure 8.

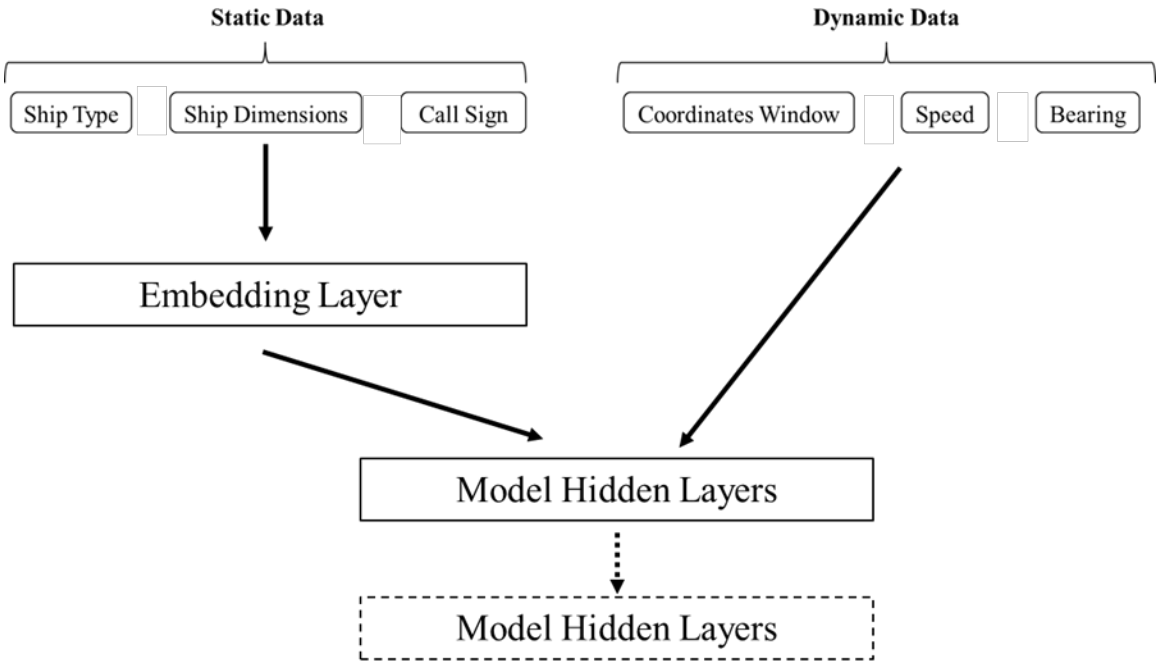

Figure 8. Input layer architecture

We use this input representation for all models. The next sections describe in detail the differences in the output representation between the classification and the regression approaches.

B. CLASSIFICATION MODEL APPROACH

To construct a categorical location target variable, there are two basic options: use "absolute positioning" of the area of interest so that every class represents an actual region or point on the map, or use "relative positioning" classes that represent the next location of the ship relative to its previous coordinates. In an absolute positioning approach, it is natural to partition the area of interest into rectangular regions so that every rectangle is a class that the model can predict. The classification RNN of this chapter uses relative positioning.

Because we believe that there is much to learn from the absolute positioning approach, we summarize those efforts in Appendix A. The rest of Chapter III describes the models that prevailed.

1. Relative Positioning

The idea behind this approach is that the next possible location of a ship highly depends on its current location and is limited by the distance a ship can travel and by the change of bearing it can amass in a single time step. Therefore, the RNN task is to predict the distance and bearing to the next location. This can be done using regression, letting the network predict a numeric value for these two features, or by using classification, letting the network assign probabilities to a set of distance categories and a separate set of bearing categories.

We implement the classification scheme, using seven classes for distance (where distance is measured in meters): [0–110), [110–320), [320–540), etc., , and 180 classes for bearing each of length two degrees: $(359^{\circ} - 1^{\circ}]$, $(1^{\circ} - 3^{\circ}]$, $(3^{\circ} - 5^{\circ}]$, up to $(357^{\circ} - 359^{\circ}]$,. In the event that we predict more than one time step into the future, we increase the number of distance categories by a factor of the number of future time-steps we predict. For example, a model that predicts 30 time steps into the future will have $7 \times 30 = 210$ distance categories. Accordingly, the size of the model's output layer will be the sum of the number of distance and the number of bearing categories.

2. Model Architecture

The core component of an RNN is the recurrent layer(s), which give it the ability to work with time series data and capture temporal relationships. We use Long Short-Term Memory units (LSTM) that have been utilized successfully in many challenging tasks involving serial data and prediction (Gers et al. 2002; Goodfellow et al. 2016; Liu et al. 2016; Tang et al. 2014).

We also use input processing layers to embed the static data and concatenate it with the dynamic data as described in Section A of this chapter. We use a set of dropout layers meant to avoid overfitting and dense layers with a rectified linear activation function meant to help the model learn complex nonlinear relationships and representations. At the output layers, we use fully connected dense layers with a softmax activation function to obtain the probabilities of each class of distance and bearing. The model architecture is given in Figure 9.

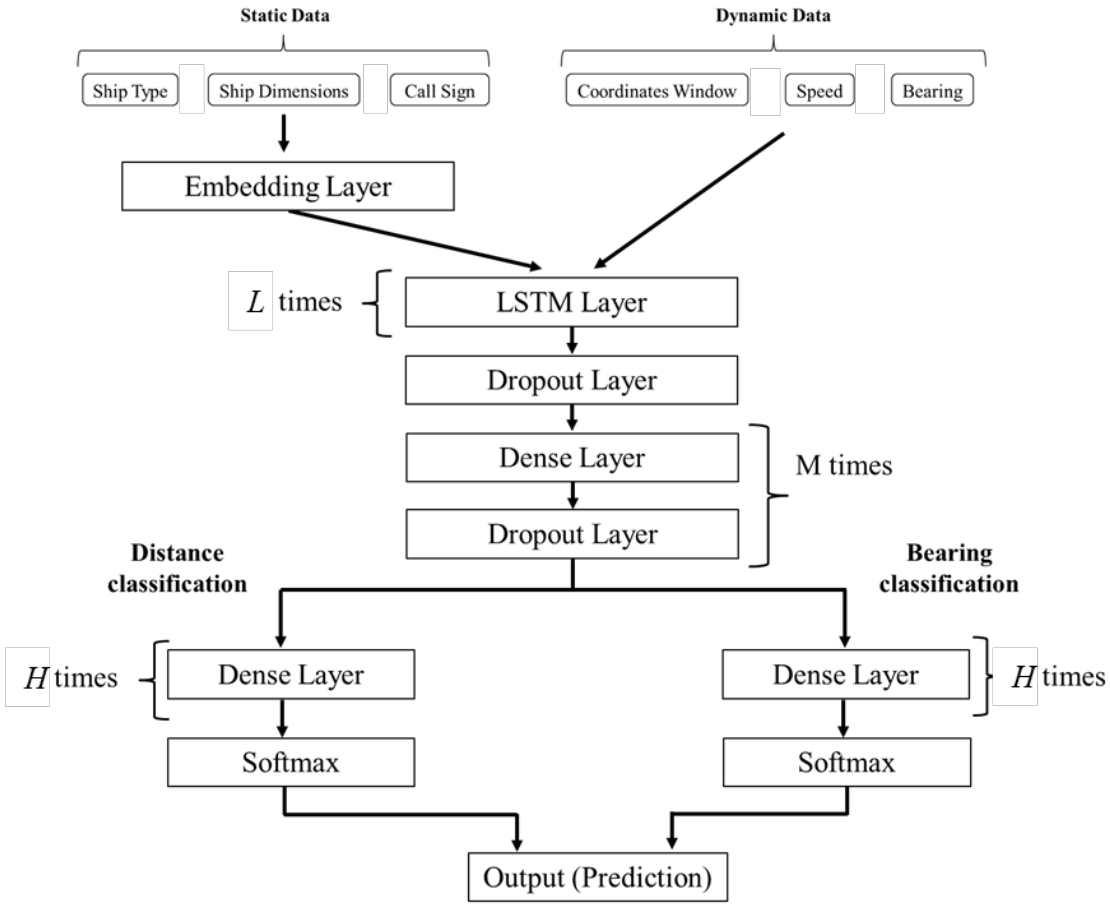

Figure 9. Classification model architecture

In order, after the input there are L consecutive LSTM layers ($L \geq 1$) that return value for the entire sequence of data they process, in a many-to-many scheme. The LSTMs are followed by a dropout layer, which randomly omits a fixed percentage (usually 10–50%) of the connections between the adjacent layers. This is used to force subsequent network unit outputs to be less correlated thus decreasing the risk of overfitting the training data (Hinton et al. 2012; Goodfellow et al. 2016). The dense layer that comes after the dropout layer uses the Keras "Time-Distributed" wrapper so the rest of the network computes, predicts, and measures the loss for every time step separately. This is the preferred approach as training can adjust for losses over many time-step predictions and not just at the last time step in the track. The dense and dropout layers may be stacked M times ($M \geq 1$) but the "Time-Distributed" wrapper is only used on the first dense layer.

At this point the model branches to two separate outputs, one that classifies the distance and one that classifies the bearing of the next predicted location. Since both branches use the same input layer, to add flexibility in turning this input into the two distinct types of predictions, we add H dense layers ($H \geq 0$) at each branch and only then use the final softmax layers. The two softmax layers work separately, predicting the distance and bearing categories so that each of their output vectors sum up to one. They are then concatenated to form the final output layer which is the RNN's prediction.

It is worth mentioning that this output layer is typical for multi-labeling classification problems (Goodfellow et al., 2016). Classification problems might only try to predict one class out of the set of possible classes using one-hot encoding. In our "multi-hot" approach, we require the model to predict a concatenation of two separate probability distributions, thus "multi-labeling." This requires a customized loss function to allow effective learning.

3. Optimization and Loss function

Let C denote the number of classes to predict, for $i = 1, .., C$ let $\mathrm{y_i} \in \{0,1\}$ be the ground-truth for the i^{th} class and $\hat{\mathrm{y}}_\mathrm{i} \in [0,1]$ be the model prediction for the i^{th} class. Classification problems usually use a categorical cross-entropy loss function, which is given by

$$\mathcal{L}(\hat{y}, y) = -\frac{1}{C}\sum_{i=1}^{C} y_i \cdot log(\hat{y}_i)\,,$$

where y and $\hat{y}$ represent respectively the one-hot target vector $(y_1, .., y_C)$ and the vector of predicted probabilities $(\hat{y}_1, .., \hat{y}_C)$. Both y_i and $\hat{y}_i$ sum to one over i. In our multi-hot version, the loss function is the sum of two such loss functions, one corresponding to distance and the other to bearing classification.

Let D denote the number of distance classes to predict (using the previous example, in a model that predicts ten time steps ahead $D = 210$) and B denote the number of bearing classes to predict ($B = 180$). Let $y_{d,i}$ and $\hat{y}_{d,i}$, $i = 1, .., D$ denote the elements of the target and prediction vectors for distance, and $y_{b,i}$ and $\hat{y}_{b,i}$, $i = 1, .., B$ denote the elements of the target and prediction vectors for bearing. The loss function is

$$\mathcal{L}(\hat{y}, y) = -\left(\frac{1}{D}\sum_{i=1}^{D} y_{d,i} \cdot log(\hat{y}_{d,i}) + \frac{1}{B}\sum_{i=1}^{B} y_{b,i} \cdot log(\hat{y}_{b,i})\right),$$

where

$$\sum_{i=1}^{D} y_{d,i} = \sum_{i=1}^{D} \hat{y}_{d,i} = 1,$$

and

$$\sum_{i=1}^{B} y_{b,i} = \sum_{i=1}^{B} \hat{y}_{b,i} = 1,$$

and where y and $\hat{y}$ represent the combined distance and bearing target and prediction vectors.

Choosing an appropriate optimizer to optimize over the loss function is very important as well. We experimented with a few different optimizers such as stochastic gradient descend (Robbins and Monro 1985), adaptive moment estimation (ADAM) developed by Kingma and Ba (2014), root mean square propagation (RMSprop) developed by Tieleman and Hinton (2012), and others. Results with the different alternatives are shown in Chapter IV.

C. REGRESSION MODEL APPROACH

We use the same input design as used in the classification approach. The key difference is in the output representation. Instead of classifying a relative location using distance and bearing, we make the model learn to predict the numeric standardized coordinates of the next location. Therefore, the natural shape of such output layer is two units (vector of size two), one for each of the predicted coordinate components (latitude and longitude).

Yet, as De Brébisson et al. (2015) describe, it can be hard to train such a simple model because it does not consider the underlying distribution of the ship's locations or the structure of the data. To overcome this, one can use a set of "reference points" that cover the area of interest for which the network would assign weights in the prediction process.

Therefore, we implement two types of regression models, one with a simple output layer of two units and another which follows De Brébisson et al.'s (2015) suggestion. To integrate prior knowledge of the ship's trajectories in the data directly in the architecture of the model, instead of predicting the pair of values describing the future location, we use a predefined set of coordinates and a hidden layer to associate a predicted probability to each of these coordinates. We then compute the output to be a weighted average of these coordinates where the predicted probabilities are the weights.

Let K denote the number of coordinates in the pre-defined set used, $\hat{p}_i \in [0,1]$ the model predicted weight of the ith coordinate, $c_i = (Latitude_i, Longitude_i)$ the ith coordinates and where the centroid $\hat{y} = \left(\widehat{Latitude}, \widehat{Longitude}\right)$ is the model prediction for the ship location,

$$\hat{y} = \sum_{i=1}^{K} \hat{p}_i \cdot c_i \, .$$

With appropriate choice of candidates, this "focuses" the model on the area of interest and particularly on the ship-dense areas. This network architecture includes a pre-output layer with K units that will be multiplied by the predefined c_i values. To make the output correspond to a centroid calculation, the hidden values p_i must sum to one. To achieve that, the pre-output layer will be activated by a softmax function.

The predefined set of coordinates starts as a set of cluster centers found by clustering all ship locations in the area of interest. The unique structure of the data includes a dense distribution of coordinates near the shore, especially near the ports, and nothing beyond, since no ship travels on land. This makes it difficult for the model to predict locations that are near the land, as the centroid is confined to the convex hull defined by the cluster centers. This is not a minor issue, as we are particularly interested in predicting ship behavior at these shoreline regions. To solve this problem, we add a set of artificial cluster centers along the shore and inland, as well as a set of peripheral coordinates to allow the model to predict destinations at the edges of the area of interest.

Different clustering algorithms and tuning settings result in a different number of clusters and spatial distribution. As we were not sure what works best, we experiment with a

few configurations. We use the mean shift clustering algorithm varying its bandwidth (Cheng 1995; Fukunaga and Hostetler 1975). In Figure 10, we present such clustering outcomes using a bandwidth of 0.073 over the 20,942,915 coordinates in the data set. The resulting 339 cluster centers are shown in red, superimposed on a heatmap of ship locations.

Figure 10. Cluster centers' spatial distribution in the area of interest

One possible advantage of this regression method over the classification approach is that the accuracy is not limited by the number of classes in the output layer. Also, this method gives weight to the prior information of the data distribution that is not emphasized in the other approach.

1. Model Architecture

The model architecture is similar to the classification model architecture, and only deviates from it in the last layers that deal with the output representation. The model architecture is given in Figure 11.

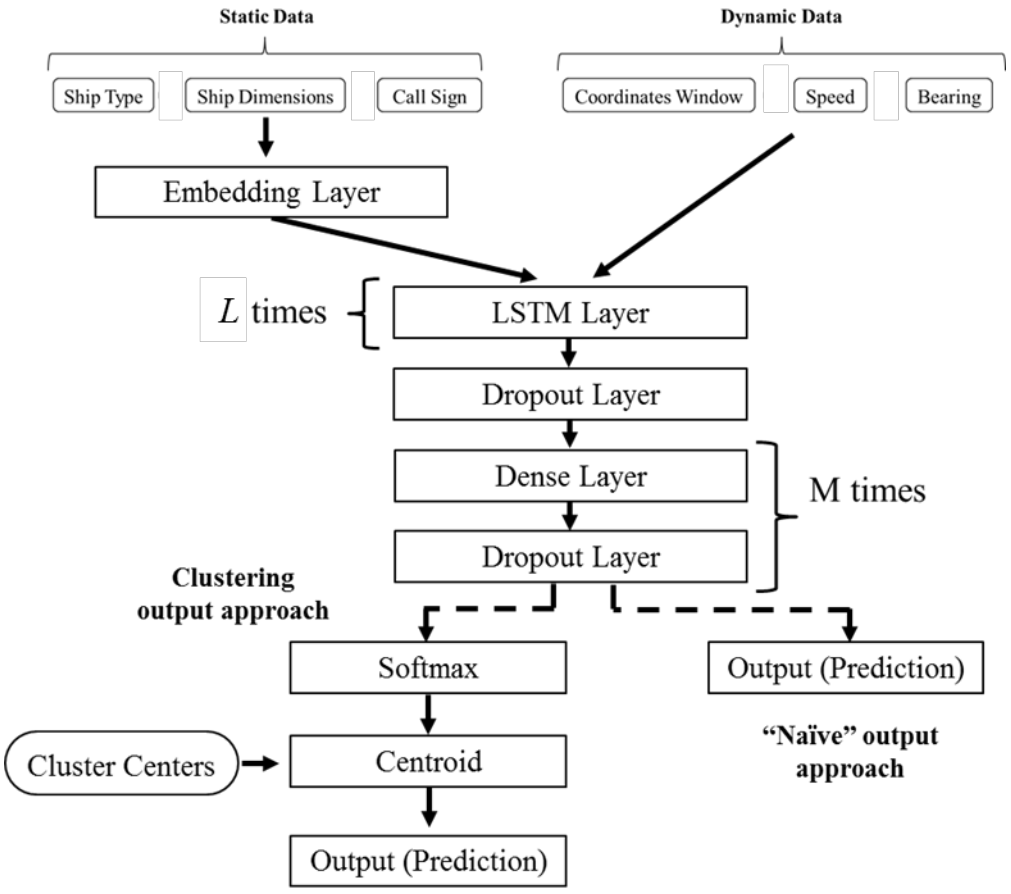

Figure 11. Regression model architecture

In the "naïve" approach the size of the output layer is two, one for each of the latitude and longitude values. In the clustering approach the pre-output layer has K units, one for each cluster center, and is activated by a softmax function. The output layer multiplies these values by the coordinates of the cluster centers. This results in the predicted location output of latitude and longitude coordinates.

2. Optimization and Loss Function

As we are trying to predict locations, we use a geographic distance loss function. An option is to use Haversine distance, which calculates the great-circle distance between two points $\mathrm{p}_1 = (Lat_1, Long_1)$ and $\mathrm{p}_2 = (Lat_2, Long_2)$ on a sphere with radius R, given their longitudes and latitudes and is given by the formula,

$$2R \cdot arcsin\left(\sqrt{sin^2\left(\frac{Lat_2 - Lat_1}{2}\right) + cos(Lat_1)\, cos(Lat_2)\, sin^2\left(\frac{Long_2 - Long_1}{2}\right)}\right)$$

Using this equation as the loss function is computationally inefficient, however, in comparison to a simpler approximation, the Equirectangular distance given by

$$R \cdot \sqrt{\left((Long_2 - Long_1)\, cos\left(\frac{Lat_2 - Lat_1}{2}\right)\right)^2 + (Lat_2 - Lat_1)^2}$$

We use the Equirectangular distance as a loss function and the haversine distance for prediction evaluation.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. MODEL TRAINING AND RESULTS

This chapter presents the training procedure and an experimental design used to search the space of hyper-parameters and architectural configurations. Results are also presented and discussed.

A. TRAINING THE MODELS

The full data set has 17,647 tracks, with variable lengths, from 300 minutes (5 hours) to 3,000 minutes (2 days). We use 70% of it (approximately 12,300 tracks) for training, 15% for validation during the training procedure and 15% for a test set. We feed the models with input that includes the track coordinates as the basic data type, along with additional input such as the speed and bearing, the ship type, area, and call sign.

The models' output is a series of predictions of the next location of the ship. While there are several ways to train a predictive model, we obtain the best predictive power when training a model to predict the ship's location at a single fixed time interval into the future. Predicting a short interval forward and using this prediction to obtain predictions at the next time steps tend to accumulate errors much faster, leading to worse prediction.

When training the models, there is a set of design choices to be made regarding the RNNs' architecture and hyper-parameter configuration. As Goodfellow et al. (2016) describes, the performance of a network is sensitive to these choices, and thus the process of architecture and hyper-parameter tuning is considered to be an important and challenging task. This requires a search in a high-dimensional space and over many orders of magnitude for some of the hyper-parameters.

Moreover, as we are working with thousands of tracks, many millions of data points and large network architectures (millions of parameters), the training time typically ranges from hours to a few days. To tackle this, we use a two-fold solution. We use methods from the field of Design of Experiments (DOE), and we use parallel computing to speed up training time.

a. Experimental Design

We cannot train a NN for every possible choice of hyper-parameters and architecture configuration or even using a coarse grid search, which is known to be an inefficient search method (Nannini and Wan 2011). Using a random search is a common practice in hyper-parameter tuning of NNs. However, a random search is not very efficient for filling a high-dimensional space and leads to correlations between the design columns (Bergstra and Bengio 2012; Sanchez and Wan 2012).

Instead, we use a search method borrowing an experimental design that has been particularly useful for studying how response surfaces change with input parameters in large-scale simulation models (Nannini and Wan 2011). This design is a Nearly Orthogonal Latin Hypercube (NOLH). According to Cioppa and Lucas (2007), NOLH designs spread the design points across the factor space so that each of the factors is examined at many different levels, and at the same time, the design columns (governing each factor) are orthogonal or nearly orthogonal to avoid simple confounding effects.

A NOLH design does not guarantee the ability to detect multiway interactions. Given the constraints we face, however, such design is a reasonable compromise. Furthermore, even if the actual response surface is complex, we can still learn a lot with low degree approximations from the NOLH designs.

The initial design includes most of the parameters over 129 experiments. Since many of the design points are extreme, we train these models for only two epochs. Using the results of these initial experiments, we rule out the non-productive factor ranges and focus the search effort. We summarize the ranges of hyper-parameters and architectural choices in Table 3.

Table 3. Productive ranges of hyper-parameters

Parameter	Initial range	Productive range
Input Choices		
Static Data (Ship type, area and call sign)	Use / Don't Use	Use
Speed and Bearing data	Use / Don't Use	Use
Window Size	[1,100]	[1,30]
Architecture Choices		
LSTM layer size (number of units in each layer)	[1,1000]	[250,500]
Number of LSTM Layers	[1,10]	[1,3]
Dense layer size (number of units in each layer)	[1,1000]	[150,300]
Number of Dense Layers	[1,10]	[1,2]
Learning Parameters Choices		
Optimizer	SGD / RMSprop / Adam / Nadam	RMSprop
Dense layers activation	None / Rectified Linear	Rectified Linear
Dropout Rate	[0,0.5]	[0.1,0.2]
Regularization (activity and kernel)	$[0, 10^{-1}]$	$[0, 10^{-4}]$
Learning rate	$[10^{-6}$,1]	$[10^{-3}, 10^{-2}]$
Learning momentum	[0, 0.99]	[0.8, 0.95]
Learning momentum decay	$[10^{-5}, 0.5]$	$[0.005, 0.1]$

The best performance is for window sizes in the range of [1,30]. Very deep networks, with more than three LSTM layers and two dense layers do not perform well, perhaps due to the "vanishing gradient" effect common in large networks.

An example for a second phase search, with the productive ranges taken from Table 3, where we alter ten factors and run 33 experiments, is shown in Table 4.

Table 4. Second phase NOLH design

Window Size	LSTM Size	LSTM layers	Dense Size	Dense Layers	Dropout Rate	Regularization	Learning Rate	Momentum	Decay Rate
30	273	2	178	2	0.16	0.00002	0.0029	0.95	0.07
27	500	1	206	1	0.12	0.00003	0.0021	0.94	0.05
26	359	3	173	1	0.16	0.00003	0.0011	0.85	0.10
17	469	3	211	2	0.12	0.00004	0.0012	0.87	0.01
28	258	2	183	2	0.17	0.00001	0.0037	0.81	0.02
29	484	2	192	1	0.12	0.00000	0.0075	0.80	0.06
21	367	3	188	1	0.17	0.00001	0.0081	0.93	0.01
16	422	3	202	2	0.13	0.00000	0.0100	0.88	0.09
20	313	1	230	2	0.13	0.00000	0.0015	0.89	0.06
23	414	2	253	1	0.15	0.00000	0.0024	0.93	0.02
22	305	3	295	1	0.11	0.00000	0.0014	0.86	0.06
24	430	2	291	2	0.2	0.00001	0.0025	0.83	0.02
18	289	1	234	2	0.11	0.00009	0.0060	0.85	0.03
25	398	2	281	1	0.16	0.00007	0.0052	0.84	0.08
19	297	3	286	1	0.1	0.00002	0.0056	0.91	0.04
25	406	2	300	2	0.19	0.00001	0.0045	0.92	0.08
16	375	2	225	2	0.15	0.00001	0.0032	0.88	0.05
1	477	2	272	1	0.14	0.00000	0.0034	0.80	0.03
4	250	3	244	2	0.18	0.00000	0.0049	0.81	0.06
5	391	1	277	2	0.14	0.00000	0.0093	0.90	0.01
14	281	1	239	1	0.18	0.00000	0.0087	0.88	0.09
3	492	2	267	1	0.13	0.00002	0.0027	0.94	0.09
2	266	2	258	2	0.18	0.00005	0.0013	0.95	0.05
10	383	1	263	2	0.13	0.00002	0.0012	0.82	0.10
15	328	1	248	1	0.18	0.00004	0.0010	0.87	0.01
11	438	3	220	1	0.17	0.00010	0.0065	0.86	0.04
8	336	2	197	2	0.15	0.00006	0.0042	0.82	0.08
9	445	2	155	2	0.19	0.00006	0.0070	0.89	0.04
7	320	2	159	1	0.1	0.00001	0.0039	0.92	0.09
13	461	3	216	1	0.19	0.00000	0.0017	0.90	0.07
6	352	2	169	2	0.14	0.00000	0.0019	0.91	0.03
12	453	1	164	2	0.2	0.00000	0.0018	0.84	0.07
6	344	2	150	1	0.11	0.00001	0.0022	0.83	0.03

The design factor scatterplots displayed in Figure 12 illustrate the space filling properties of NOLH.

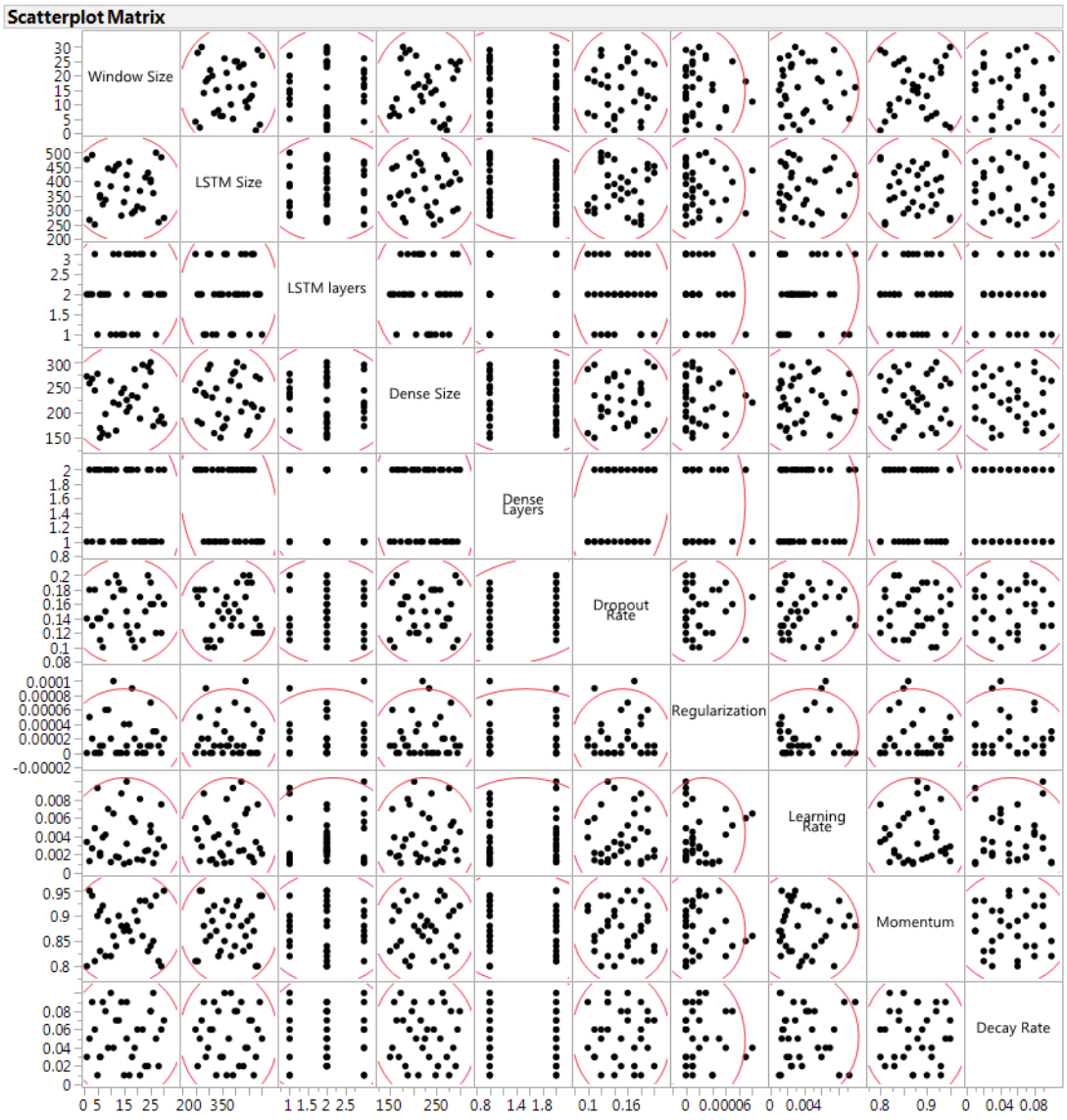

Figure 12. Scatterplot matrix of the design variables

Since many of the factors are categorical with very few levels (two to three) the orthogonality of the design is somewhat compromised but correlation between factors is still relatively low, as seen in Table 5.

Table 5. Correlations of phase two design columns

	Window Size	LSTM Size	LSTM layers	Dense Size	Dense Layers	Dropout Rate	Regularization	Learning Rate	Momentum	Decay Rate
Window Size	1.0000	-0.0030	0.0615	0.0117	-0.0972	0.0184	-0.0262	-0.0076	0.0313	0.0017
LSTM Size	-0.0030	1.0000	0.0312	0.0011	-0.2156	0.0250	-0.0096	0.0144	-0.0104	-0.0128
LSTM layers	0.0615	0.0312	1.0000	-0.0483	-0.2127	-0.0146	-0.0028	0.0803	0.0045	0.0109
Dense Size	0.0117	0.0011	-0.0483	1.0000	-0.0605	0.0239	-0.0176	-0.0093	0.0167	-0.0370
Dense Layers	-0.0972	-0.2156	-0.2127	-0.0605	1.0000	0.2337	0.0203	-0.0042	-0.0769	-0.0565
Dropout Rate	0.0184	0.0250	-0.0146	0.0239	0.2337	1.0000	0.0368	0.0006	-0.0280	-0.0225
Regularization	-0.0262	-0.0096	-0.0028	-0.0176	0.0203	0.0368	1.0000	0.0664	-0.0445	-0.0656
Learning Rate	-0.0076	0.0144	0.0803	-0.0093	-0.0042	0.0006	0.0664	1.0000	-0.0256	-0.0141
Momentum	0.0313	-0.0104	0.0045	0.0167	-0.0769	-0.0280	-0.0445	-0.0256	1.0000	0.0009
Decay Rate	0.0017	-0.0128	0.0109	-0.0370	-0.0565	-0.0225	-0.0656	-0.0141	0.0009	1.0000

b. Parallel Computing

As described by Goodfellow et al. (2016), NNs train much faster on graphics processing units (GPU) than central processing units (CPU). The majority of work in this research is done with a server, hosting the data and computation in a 24-core Intel Xeon E5-2650 with 512 GB RAM and two Nvidia Titan XP 12 GB GPUs using Nvidia CUDA 9.2 libraries.

However, the search described in this section uses the Naval Postgraduate School (NPS) High Performance Computing center (HPC), sending the computing jobs in batches to a heterogeneous computing cluster managed with the Simple Linux Utility for Resource Management (SLURM) job scheduler, and running them in parallel over dozens of nodes, sharing 16 GPUs with 120 GB.

We implement the models using the Python language, the Keras library (Chollet 2015), and a TensorFlow (Abadi et al. 2016) backend.

c. Additional Challenges

Another issue with the training process regards the exact method of feeding the data into the networks. The model "learns" on mini-batches of a set of 20–100 tracks each time,

updates its internal weights and parameters according to the prediction errors, and moves to the next batch of tracks. However, with Keras all tracks in a batch must have the same length.

One solution to such limitation is to choose a maximum length for all tracks in the data set, truncate the longer ones, and pad the shorter ones with zeros (or a relevant symbol that signals the model that this is not "real" data) or pre-pad these tracks with the first "real" input repeated. Initially we implemented this approach, paying a double fine; we lost massive parts of the data in the long tracks, and we forced the models to learn irrelevant data in the pre-padded short tracks.

A better solution is to create a data generator that iterates through the training set. The data generator sorts the training set from shortest to longest track, then iteratively takes a batch size of tracks, truncates them to the length of the shortest track in the batch (avoiding pre-padding), and feeds them as input to the model. Then the generator jumps forward a fixed number of tracks and repeats the process. This approach is illustrated in Figure 13.

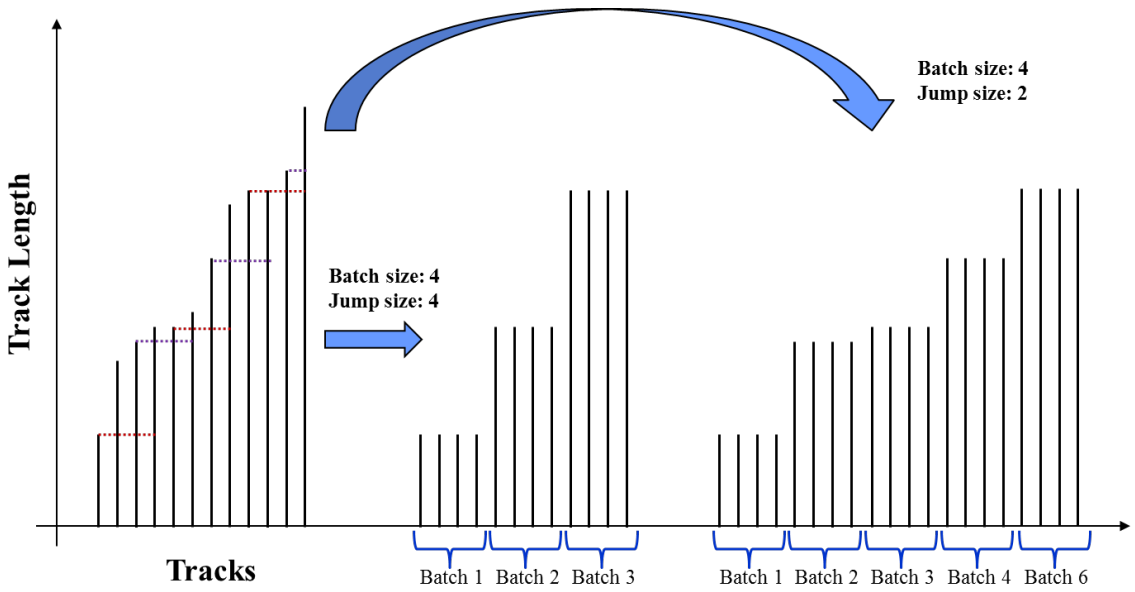

Figure 13. Data generator iterative scanning procedure

In this example, the entire training set has 12 tracks, and the batch size is four tracks; therefore, standing at track (*i*), we truncate tracks (*i+1*), (*i+2*), and (*i+3*) according to tracks'

(*i*) length (they are sorted by length). If we choose a jump size of 4, we iterate through the training set in three batches. If we choose jump size of 2, we iterate through the training set in six batches, feeding most (all apart from the first and last half batch) of the tracks into the network twice. This is desirable since in training NNs the data is usually fed to the model many times anyway, and by using small jump sizes the model gets to "see" more of the data.

B. RESULTS

We train two types of NNs: regression and classification models. Since they use different loss functions, we use the classification model's output to compute a miss distance (the distance between the predicted and actual ship location) to be comparable to the regression model.

We use the following metrics of accuracy:

- median miss distance
- average miss distance
- distribution of the miss distances

Since the prediction is based on time series data, and we do not expect the model to give accurate predictions during the very beginning of the track. Thus, we give the model a "grace period" of 100 warm-up time steps and only from that point onward do we measure the losses.

All models share some hyper-parameters found to give good performance. L2-Norm regularization with a coefficient of 0.0001 is used to regularize the recurrent weights in the LSTM layers and the dense layers. Whenever we use dropout layers, they are set to randomly dropout 0.1 of the connections between the layers from both their sides. The regularization and dropouts are only active during the training of the models.

Furthermore, the activation function of the dense layers in all models is ReLu. All models are trained with the RMSprop optimizer, using an initial learning rate of 0.005, a momentum of 0.9, and decay of 0.01. We use a learning rate scheduler that drops the learning rate by 50% if the loss over the validation set do not improve in the previous five epochs.

The results are presented according to their prediction horizon. First, we present models predicting one time step into the future, then 30 time steps, and finally 100 time steps into the future.

1. Prediction Models Using One Time Step (1 Minute)

We begin by presenting the results of short-term predictions of one time step (1 minute) into the future. Table 6 presents the mean and median miss distance for the best models, calculated over a test set that includes 2,647 tracks of various lengths (sampled at random from the initial database and separated from the training and validation data sets). The actual miss distance is calculated using the Equirectangular distance on all predictions made after a warm-up period of 100 time steps (100 minutes), leaving a total of 2,843,539 predictions to evaluate.

Table 6. Accuracy of 1-minute prediction models

#	Model description					Mean miss distance ± 95% CI [meters]	Median miss distance [meters]
	Type	Inputs	Window size	LSTM layers	Dense layers		
1	Classification 20 meters	All Dynamic Data	20	1X250	1X150	8	0
2	Classification 20 meters	All types	20	1X250	1X150	123±1	12
3	Classification 100 meters	All types	20	1X250	1X150	135±1	10
4	Classification 100 meters	All types	20	2X400	1X300	137±1	14
5	Classification 100 meters	Coordinates only	20	1X250	1X150	105±1	100
6	Classification 100 meters	All types	1	1X250	1X150	287±1	148
7	Regression Clusters	All Dynamic Data	20	1X250	1X150	810±2	506
8	Regression	All types	20	1X250	1X150	808±2	606
9	Regression	All types	20	2X400	1X300	865±2	637
10	Regression	All types	5	1X250	1X150	1380±4	1168

#	Model description					Mean miss distance ± 95% CI [meters]	Median miss distance [meters]
	Type	Inputs	Window size	LSTM layers	Dense layers		
11	Regression	All Dynamic Data	20	1X250	1X150	881±2	703
12	Regression	Coordinates only	20	1X250	1X150	1025±3	826

One of the architectural design choices in classification NNs is the resolution of classes available. While classifying the bearing is done with 180 classes, of two degrees each, we experiment with different distance classification resolutions. The first two models, #1 and #2 use a fine grid of distance categories of 20 meters; thus, the possible categories are: {0, 10, 30, 50, … ,1500}. The rest of the classification models use a resolution of 100 meters with categories of {0, 50, 150, … ,1500}.

The classification models appear to outperform the regression models in one time-step predictions. The best model, #1, is the slim version with one LSTM layer with 250 units, one dense layer with 150 units, coordinates input window of 20 time steps that takes as input the dynamic data (coordinates, speed, and bearing) and does not take the static data. Its predictions, in a resolution of 20 meters, have a mean miss distance of 8 meters and median of zero meters. Here, the prediction horizon is one time step, which is 1 minute, so the ship can travel up to 1,000 meters in this time (given the speed limit we use to filter the data of 60 km/hour).

The next best model has the same architecture, but with static data as additional input. Surprisingly, the extra information supplied not only do not improve the predictions, it makes them worse. This result is consistent with the other models results, which leads us to suspect the embedding mechanism we use does not fit the problem or that the training set is too small. In contrast, using the speed and the bearing as input, in addition to the coordinates, improves the predictive performance, as can be seen in comparing models #1 and #5 and regression models #11 through #12. We conclude that richer input has the potential to improve performance, but choosing the right representation design is important for that goal.

Further, it seems that larger models perform slightly worse that the smaller ones, as can be seen when comparing models #3 and #4 and #8 and #9. On the other hand, a longer coordinate input window seems to improve the results for both classification and regression as can be seen by comparing models #4 and #6, and with models #8 and #10. The best regression model is #7 that uses the cluster centers approach. This model assigns weights to 339 cluster centers and the prediction is therefore the weighted center of these points.

The superiority of classification models for this time horizon prediction can be partially explained by the way they work. Classification models use a set of categorical units as an output layer and so their predictions are discretized to a set of distances and heading angles. In the one time-step models, the maximum distance is category 1,500 meters, and thus the potential miss distances are bounded, which focuses the network's prediction and gives it a relative advantage over regression models.

An example for a prediction is given in Figure 14. The blue line represents the track itself (the target for the prediction) and the red line represents the model's prediction. The blue and purple bubble icons mark every tenth time step. Since the predictions are accurate, it is hard to see the difference between the lines (since they are partially transparent they blend to purple).

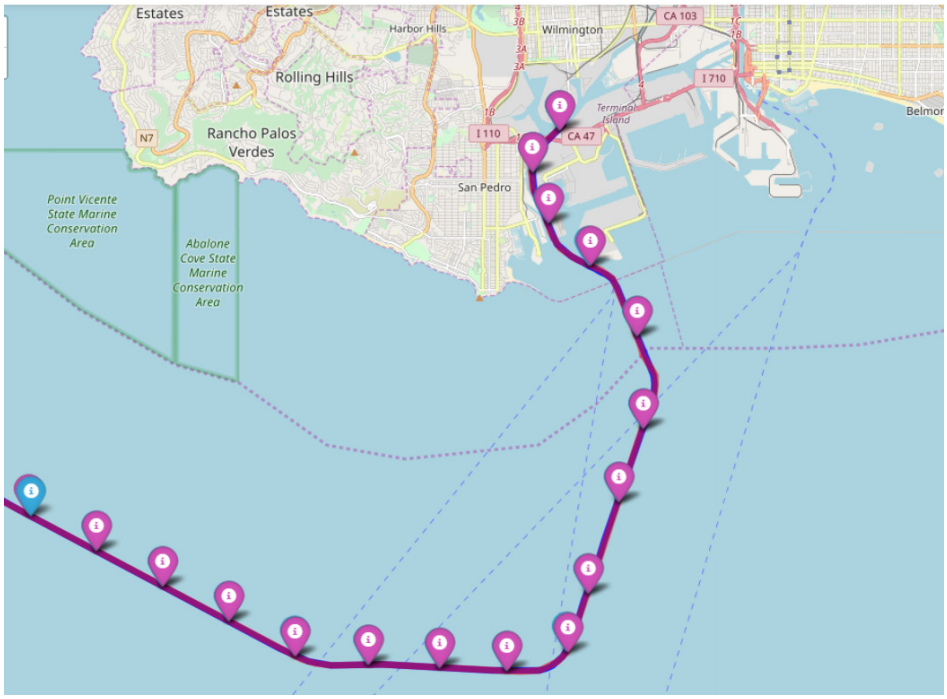

Figure 14. One-minute predictions (red) of classification model #1 and actual track (blue)

As can be seen in Figure 14, the RNN learns the motion pattern of the ship and successfully predicts the next time-step location of the ship. This result is achieved both in classification RNNs and in regression RNNs (though less outstanding).

2. Prediction Models Using Thirty Time Steps (30 Minutes)

Next, we train a set of models to predict 30 time steps (30 minutes) into the future. The best results are given in Table 7 . Next to the median miss distance scaled to 1-minute misses, so that it can be compared to the results in Table 6. All classification models use a distance resolution of 215 meters.

Table 7. Accuracy of 30-minute prediction models

#	Model description					Mean miss distance ± 95% CI [meters]	Median miss distance [meters] (miss per minute)
	Type	Inputs	Window size	LSTM layers	Dense layers		
1	Classification	All Dynamic Data	20	1X250	1X150	670±2	411 (14)
2	Classification	All Dynamic Data	40	1X250	1X150	673±2	400 (13)
3	Classification	All types	20	1X250	1X150	1367±4	473 (16)
4	Classification	All types	20	2X400	1X300	1508±4	501 (17)
5	Classification	Coordinates only	20	1X250	1X150	2862±9	2143 (71)
6	Classification	All types	1	1X250	1X150	3845±15	929 (31)
7	Regression Clusters	All Dynamic Data	20	1X250	1X150	1453±5	838 (28)
8	Regression	All types	20	1X250	1X150	1541±5	1102 (37)
9	Regression	All types	40	1X250	1X150	1623±5	1154 (38)
10	Regression	All Dynamic Data	20	1X250	1X150	1445±5	1063 (35)
11	Regression	All types	10	2X400	1X300	2325±8	1222 (41)
12	Regression	Coordinates only	20	1X250	1X150	2481±8	1694 (56)

The results of these models are similar to the short-term prediction models. Classification models seem to perform better and avoiding the use of the static data improves the prediction substantially. Yet, richer dynamic data is better than using coordinates alone, as reflected in models #1 and #5 and regression models #7 and #12.

Experimenting with a larger coordinate input window size of 40 yields roughly the same results. On the other hand, decreasing its size to 10 or 1 significantly reduces model accuracy, as seen when comparing models #3 and #6. The best regression model, #7, uses the clustering approach with 339 centers.

In the case of 30 time-step prediction models the overall predictions are close to the actual track. Given the average speed of the ships in the database, 26 km/hour or 14 knots, in 30 time steps (30 minutes) a ship travels on average 13 km. Therefore, a median miss distance of about one kilometer for the regression model and less than 500 meters for the classification model is quite accurate. Figure 15 shows the miss distances distributions for the best classification and regression models (#1 and #6, respectively).

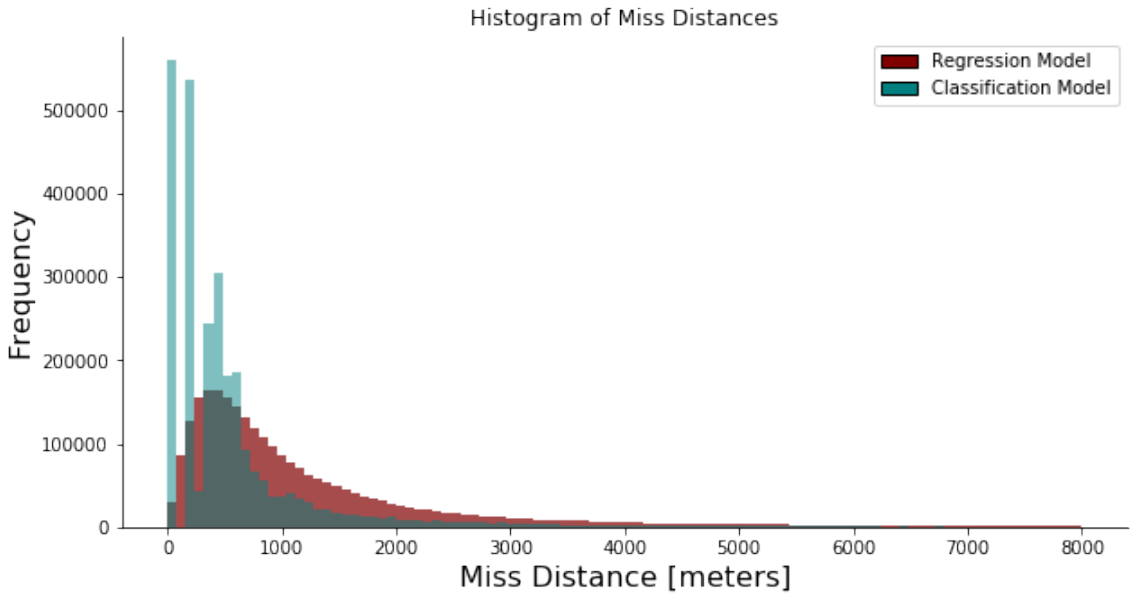

Figure 15. Histogram of miss distances for models #1 and #7

The classification model miss distance distribution is centered around lower distances than the regression model. Also, some discretization of the miss distances can be seen in the lower range. This is due to the discrete categories of distances, which are in intervals of 215 meters for this model. In Figure 16, we present an example to predictions made by model #1, using the same color-coding as in Figure 14.

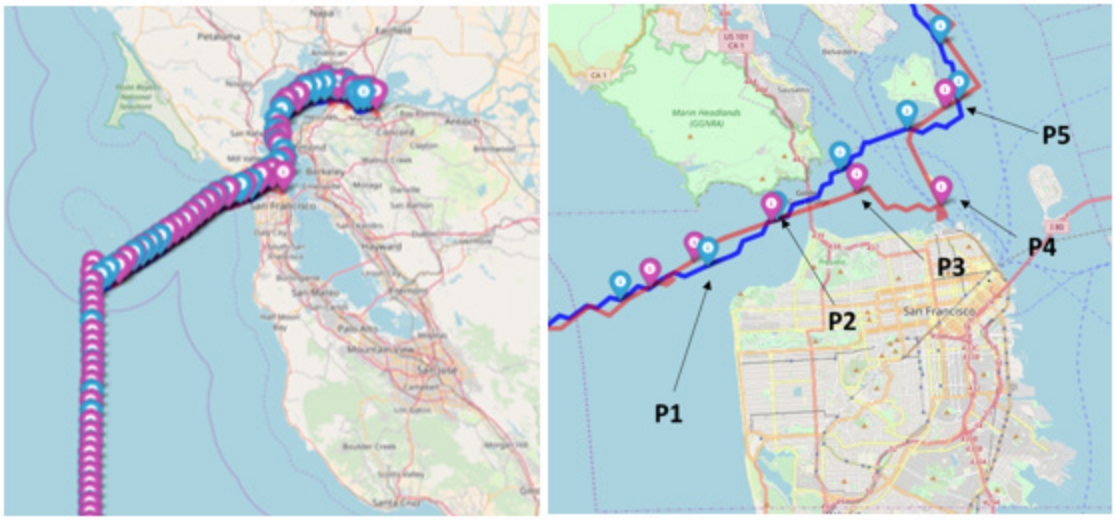

Figure 16. 30-minute predictions (red line and bubbles) of classification model #1 and actual track (blue line and bubbles)

In the left plot of Figure 16, we show a track from September 17, 2016, of ship with MMSI 367455580 that is sailing to Concord. The right plot of Figure 16 takes a closer look at the prediction process near San Francisco (SF). Since the model predicts 30 time steps ahead and every bubble on the map marks ten time steps, when a ship is at P1, the model predicts that in 30 minutes, the ship will be at P4, heading to SF port. Yet, at ten minutes later, when the ship is at P2, the model acknowledges the slight change in bearing and corrects its prediction to P5. At P3 the model is already confident in the ships direction and predicts the sharp left turn that will happen in 20 minutes' time.

In Figure 17, we present an example of predictions made by the best regression model, #7. The green dots represent the cluster centers the model uses to build its predictions.

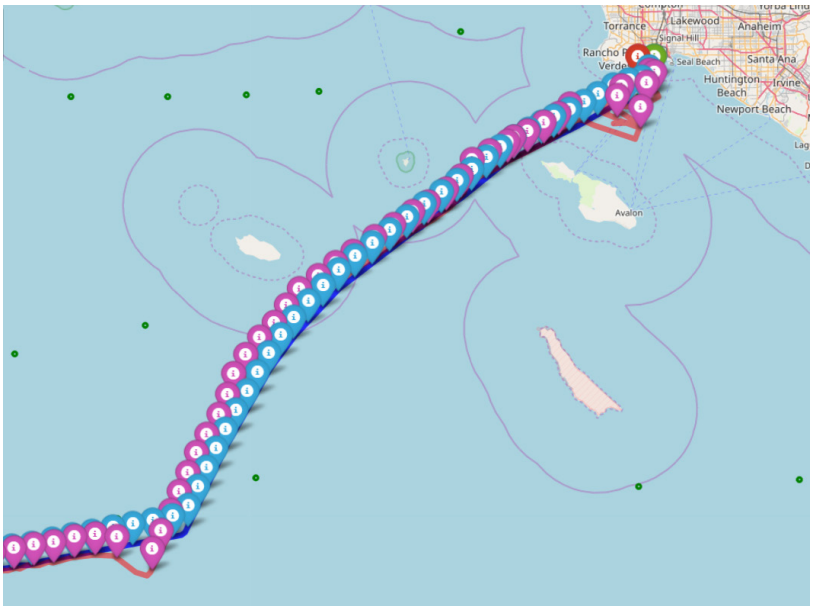

Figure 17. 30-minute predictions (red line and bubbles) of #7 regression model and actual track (blue line and bubbles) with cluster centers (green dots)

At the beginning of the track where the green and red markers are located, the prediction is not good, as the model is in its "warm-up" period. After the warm-up period, the model predicts the track quite accurately. Sudden turns are not always predicted, and the model corrects itself to the actual track after a short while. This is especially encouraging in the regression models, since we are predicting a set of coordinates that are only bounded by the convex hull of landmark cluster centers, covering the entire region.

The predicted track itself is less fragmented than the one produced by the classification approach since the possible predicted locations are not discretized. Using the weighted mean of the cluster points also contributes to a smoother prediction. It is worth noting the cluster centers that can be seen in Figure 17 are relatively sparse, which might negatively affect the performance.

3. Prediction Models Using One Hundred Time Steps (100 Minutes)

Forcing the models to predict "far" into the future makes the RNNs learn different patterns. As the information on which the predictions are made is less current, the models must now give larger weight to common ship behaviors in the region and extract more from

the static input data. In Table 8 the best models' performance can be observed. The classification models use a distance resolution of 500 meters. The angular resolution becomes a potential issue at these prediction horizons, as the prediction will be typically around 45 km ahead. Two degrees difference at such distance a will translate to a full mile.

Table 8. Accuracy of 100-minute prediction models

#	Model description					Mean miss distance ± 95% CI [meters]	Median miss distance [meters] (miss per minute)
	Type	Inputs	Window size	LSTM layers	Dense layers		
1	Classification	All Dynamic Data	20	1X250	1X150	2795±4	1673 (17)
2	Classification	All types	20	1X250	1X150	3024±4	1728 (17)
3	Classification	All types	20	2X400	1X300	3222±4	1891 (19)
4	Classification	All Dynamic Data	20	2X400	1X300	2974±4	1502 (50)
5	Regression Clusters (1023)	All Dynamic Data	20	1X250	1X150	3763±11	2020 (20)
6	Regression Clusters (339)	All Dynamic Data	20	1X250	1X150	3950±11	2087 (21)
7	Regression	All types	20	1X250	1X150	4427±13	2642 (26)
8	Regression	All types	10	2X400	1X300	4435±13	3006 (30)
9	Regression	All Dynamic Data	20	1X250	1X150	4200±12	2595 (26)

As in the results shown in Table 6 and Table 7, the classification models seem to be better, but the difference is smaller. There might be a threshold effect where regression has some limiting error that is very prominent when trying to predict a short time into the future, but less so when predicting 100 time-steps forward. The best regression models are the ones using the clustering approach. Increasing the number of cluster centers is challenging computationally, as the computer has to hold in memory an array with one of its dimensions

being the number of clusters. The largest we experiment with is a 1,023 cluster centers model, #5, which seems to perform better than model #6, which uses the same architecture but with a smaller number of cluster centers. This makes sense, as it should be easier for the model to be accurate with the added centers. In Figure 18 we display the difference in the cluster centers point densities.

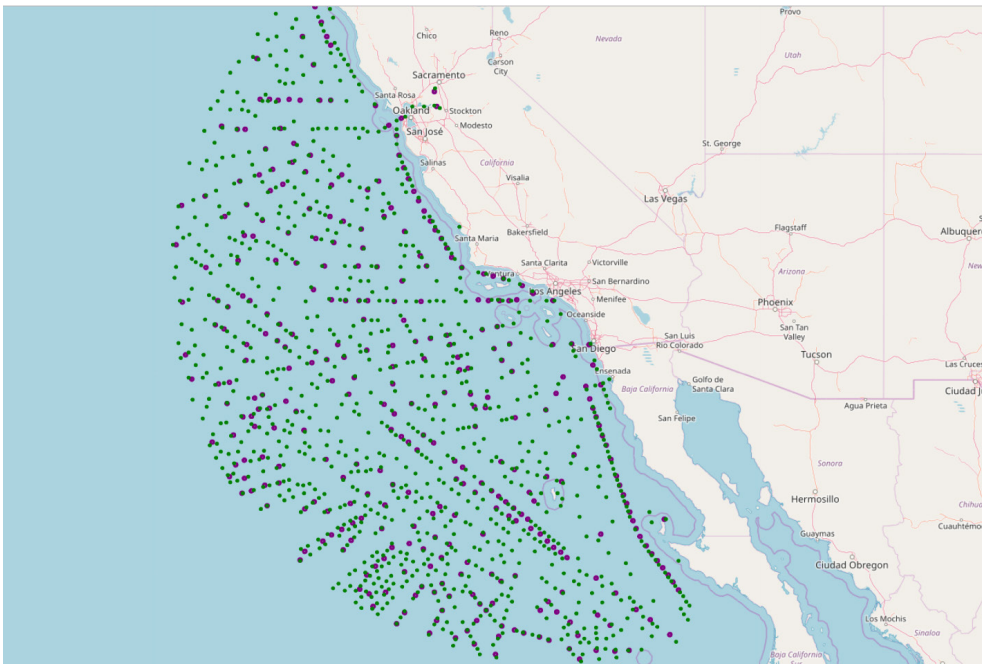

Figure 18. Cluster center points using 339 clusters (purple) and 1,023 clusters (green)

The miss distance distributions of models #1 and #5 in Figure 19 show the discretization effect for the classification model and the longer heavier tail of larger miss distances for the regression model.

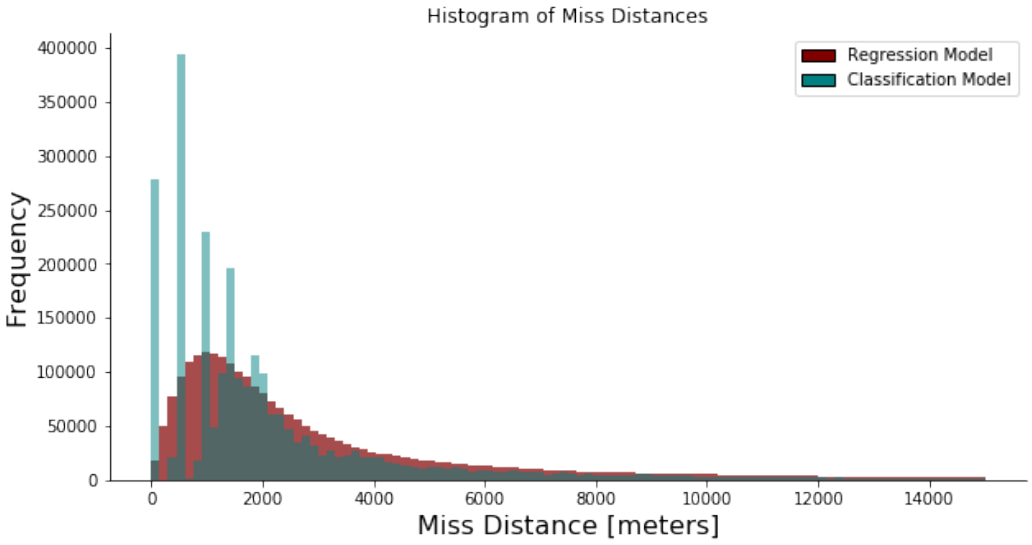

Figure 19. Miss distance histograms of models #1 and #5

In Figure 20 we give an example of predictions made by the best classification model, #1, on a ship heading to Los Angeles port. We thinned the frequency of markers along the track to one every 50 time steps.

Figure 20. 100-minute predictions (red) of classification model #1 and actual track (blue)

At P1 the model predicts that the ship will reach P3 in 100 time steps (an hour and forty minutes from the current point). It predicts the entrance to the port, but aims for a different location to harbor. At P2, the model predicts that the ship will reach the central part of LA port. The fragmentation of the track is due to the discretization discussed earlier.

4. Detection of Abnormal Ship Behaviors

A model with good predictive ability and relatively low miss distances can be used to detect anomalous behaviors. Most tracks in the training database exhibit normal behavior. We expect abnormal tracks to deviate from the norm, and therefore to be harder to predict with our model, which is trained on primarily normal behavior. Abnormal tracks and potentially nonlegitimate ones should have larger prediction errors. Tracks with high miss distance have something in them that is abnormal or not frequently witnessed during the training of the prediction model.

In the next passages we show an initial exploration of anomalies detection using the prediction loss, which is far from a developed methodology. We evaluate the models shown in the previous tables over every track in the test set and sort the resulting mean miss distances from smallest to largest. The tracks with the smallest mean miss distances are simple tracks of straight lines and smooth sails along the coast and into harbor. On the other hand, the tracks with the largest mean miss distance are seen to exhibit erratic behavior with unexpected turns at different locations. In Figure 21 we give the distribution of mean miss distance across each track of the best 30 time-steps classification model (#1 in Table 7). The right tail of the distribution includes the tracks where the prediction, on average along the track, is less accurate.

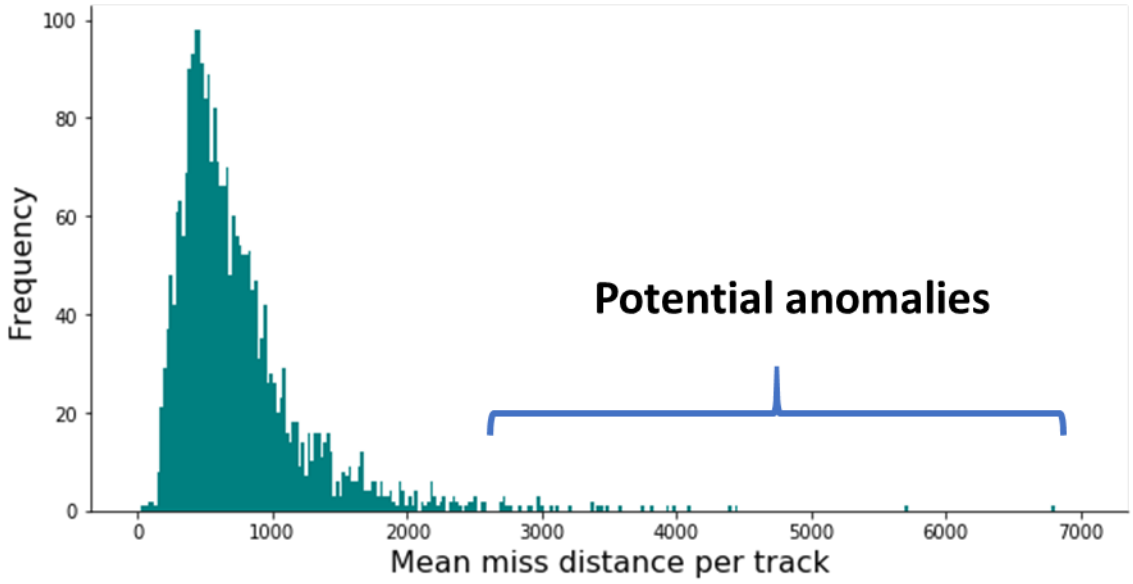

Figure 21. Histogram of mean miss distance per time step for anomaly detection

Figure 22 and Figure 23 show examples of four cargo and passenger ship tracks with the highest mean miss distance. To make a clearer display we do not plot the bubbles that mark every tenth time step as in the previous tracks' figures. We do mark the first time step with a green bubble and the first prediction with a red one.

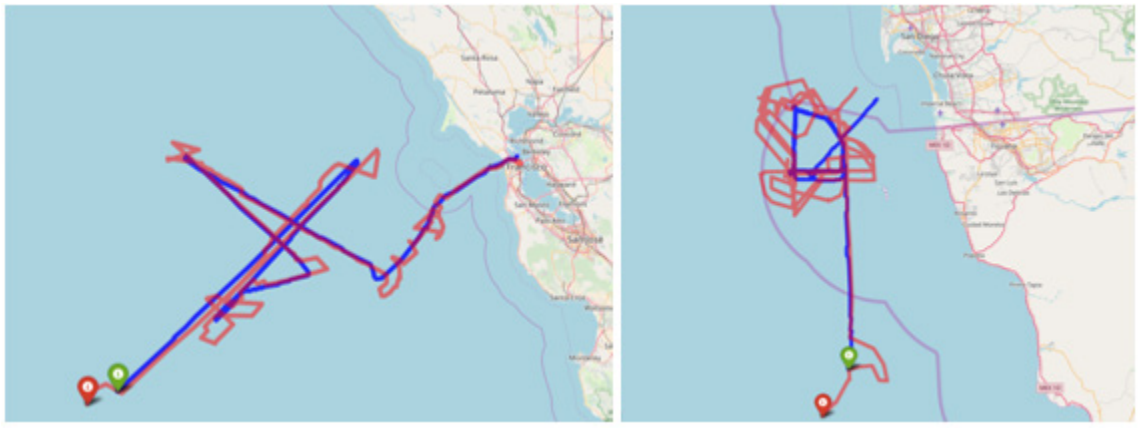

Figure 22. Examples of anomalous tracks (blue) with their predicted tracks (red)

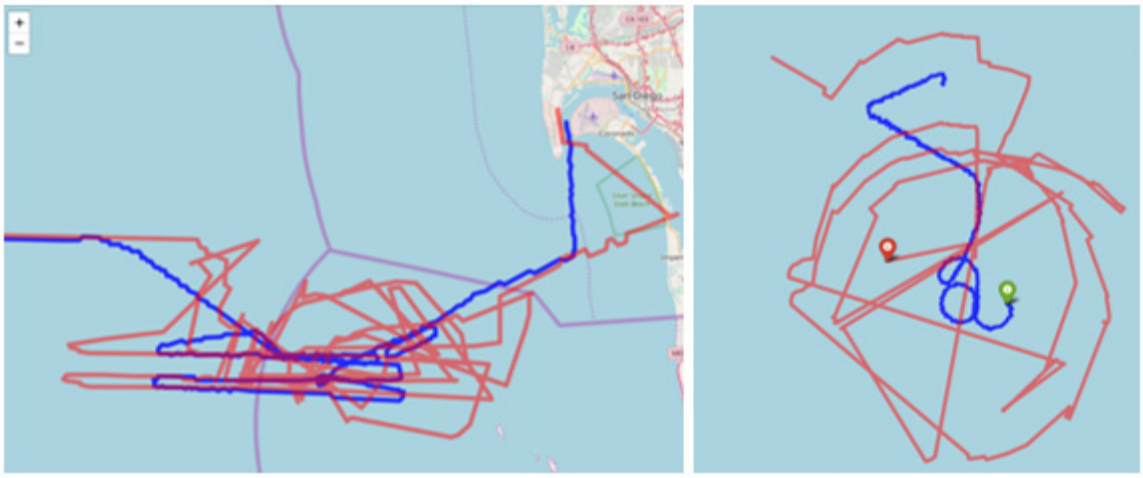

Figure 23. Examples of anomalous tracks (blue) with their predicted tracks (red)

The ability to detect such behaviors using the mean miss distance, without the need for a human operator to watch screens filled with ship tracks, is a valuable tool in today's maritime environment. A system that takes as input these tracks and additional information can potentially identify anomalies from their very beginning, alert an operator, or execute other sets of orders. This can be done by following the online miss distance of the track, searching for sudden spikes. Figure 24 shows an anomalous track of the Japanese cargo ship *Texas Highway* (MMSI 432440000) passing by San Diego with bubble markers at every 50th time step (annotated). Unexpectedly, starting at around time-step 200, the ship makes a few sharp turns and only gets back to track after the 400th time step.

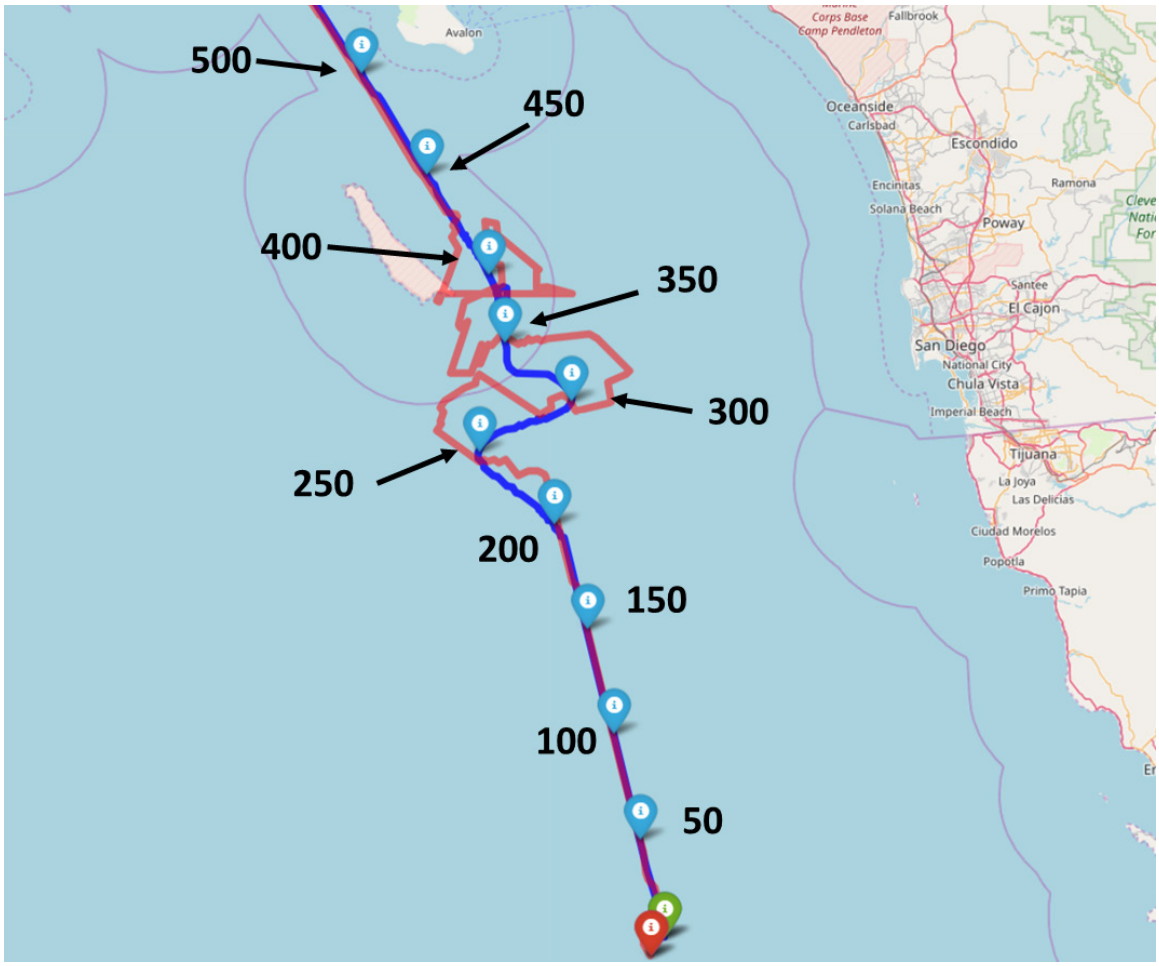

Figure 24. Anomalous ship track of *Texas Highway* (blue) with its predicted track (red)

Following the "online" prediction miss distance plotted in Figure 25, it is easy to detect the beginning of the anomalous behavior of the ship by the rising spikes, as pointed to by the red arrow.

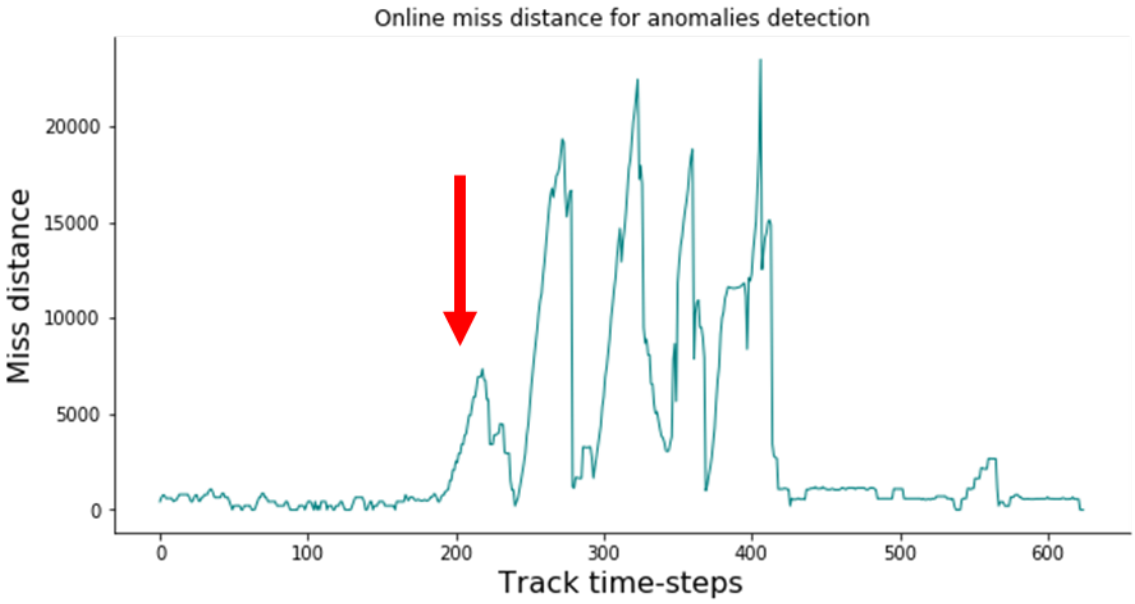

Figure 25. Online miss distance of *Texas Highway*'s track

V. DISCUSSION

In this chapter we discuss potential improvements to the models and suggest relevant future work. We also discuss some of the challenges we encounter and their solution.

A. MODEL IMPROVEMENTS AND FUTURE WORK

Our models' performance falls short of giving a satisfactory and universal solution to the problem of predicting ship trajectories and detecting anomalies from those predictions. There are numerous ways in which the models can be improved and their performance potentially enhanced. We describe some of these here.

1. Data

Although we use a large data set, the actual number of tracks is small. For example, a recent work by De Brébisson et al. (2015) using NNs to predict taxi ride destination in the city of Porto, Portugal, based on a Kaggle challenge, use 1.7 million tracks (or parts of tracks) to predict the final ride's destination. We use about 1% (~17,500 tracks) of their training set size. We also use only two years' worth of AIS data, out of at least eight years of available records.

Further, in the process of cleaning the data we exclude a tremendous number of training examples. In addition, there are methods to artificially enlarge the data set. One could "play" tracks in both directions, thus doubling the size of the training set. Experimenting with partitioning long tracks into shorter ones might also prove useful. We believe that expanding the data set to include these types of additional data can improve the models' predictive ability.

We see that, in general, enriching the data improves the prediction. Nonetheless, this must be done in an appropriate manner. We believe the method of embedding the static data can be improved so that it will make a positive contribution to the predictive ability. This might be done by experimenting with the embedding size, which we do not do. Embedding more input types such as the time and date and the weather might prove useful as well.

While the static data do not prove to have a useful impact, the additional dynamic data of speed and bearing certainly do. Adding a window scheme as we use for the coordinates might enhance their contribution in a similar manner. Further, adding "environmental" data

such as the location of nearby ships and Meteorological and Oceanographic (METOC) information seems like a promising enhancement. These nearby locations can be extracted from the existing AIS data.

2. Model Architecture

To use more geographical context features, one might implement a combination of convolutional layers to "read the map" as an image. Convolutional LSTM networks have been successfully used by Xingjian et al. (2015) for rain precipitation nowcasting. If the input will be rich enough, such a scheme might prove efficient.

Batch normalization developed by Ioffe and Szegedy (2015) dramatically improves the performance of deep NNs. This technique has been adapted by Ba et al. (2016) and others to recurrent architectures using layer normalization. We did not experiment with such architectural variants, but they might prove beneficial to future work.

Although the clustering approach is the most successful out of the regression models, we believe the results do not represent its full potential. We use a relatively small number of clusters and do not thoroughly tile the area of interest with artificial cluster centers. For most models we use 339 cluster centers to predict coordinates in an area the size of approximately 2 million km^2. In comparison, De Brébisson et al. (2015) use approximately 3300 cluster centers to predict coordinates in an area the size of approximately 250 km^2.

Another approach that we experiment with, but do not report in the main chapters of this thesis, is the absolute positioning approach for classification models. We describe this approach in Appendix A. It might be worth examining again with larger data sets.

3. Training

In most cases we stop the network training before reaching convergence. This is especially relevant when using a learning rate scheduler. Since the training sessions are relatively short, there is no chance for the learning rate to decrease significantly, thus preventing potential "leaps forward" in performance as described by Goodfellow et al. (2016).

Using model evaluation over the test set we identify anomalous tracks, as we explain in Chapter IV. This approach can be incorporated into the training procedure by creating a

separate model that tries only to classify whether a track is normal or anomalous. To enhance such a classifier, supervised learning can be used as well by enriching the data set with known incidents of criminal activities, piracy, ships with technical issues that affected their track, etc.

B. CHALLENGES AND LESSONS LEARNED IN WORKING WITH NEURAL NETWORKS

In this part we describe some of the challenges and lessons learned during the research, meant as guidance for future projects of this sort.

- In calculating distances on a map, we find that using the Equirectangular distance to approximate distance is sufficient and is fast. However, when projecting the next destination given origin coordinates, distance and bearing, we find that one must use the Haversine distance formula to be accurate.

- In some instances, the NN might use division operations on very small numbers. Working in a digital environment, one might come across floating point problems, a very small number "becomes" zero, and division by this number results in NaN (Not a Number) values. For this, we use two solutions:

- Keras has a software option to clip the norm and values of the weights' updates so that NaN values do not (frequently) occur.

- We use numeric data type of large memory such as float32 or float64.

- When working with serial data in NNs, truncation or padding might be needed. We propose being careful with that, as the NN might "understand" the padded values to be what they are meant to be. At an early stage of this work we discover that one of the models is only learning to predict the (0,0) padded values, which are the geometric center of all the coordinates in the data set.

- At the beginning of this work we only used tracks with a much larger time interval between consecutive points, 15 minutes instead of 1 minute. As a ship can travel a good number of miles in 15 minutes, the prediction resolution is poor in the LA region and leads to impossible tracks that cross land and so forth. We recommend working with the most frequent data possible, and if needed thin it to 1-minute intervals. An example of an impossible track is shown in Figure 26, where a ship crosses through San Francisco on its way to Oakland port.

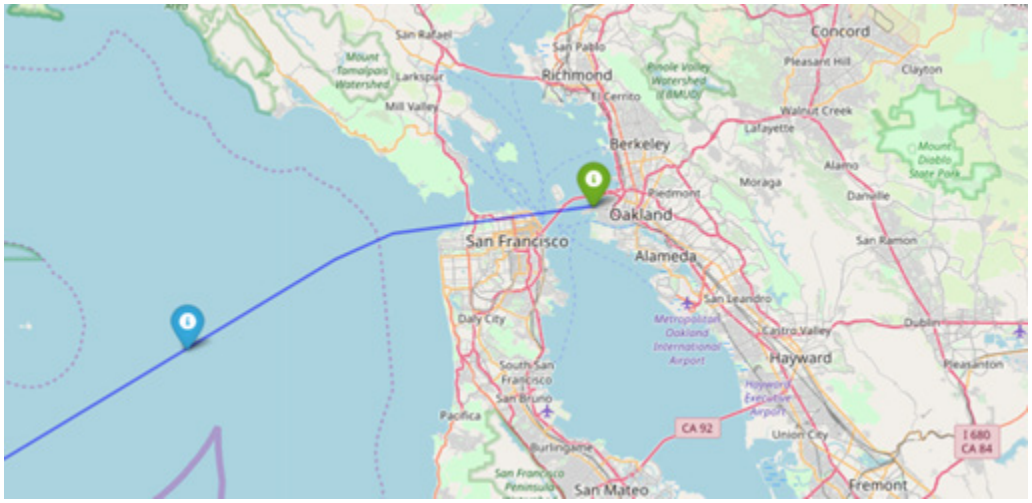

Figure 26. Example of bad resolution time-steps interval causing the interpolated track to cross land

VI. CONCLUSIONS

In this work, we use RNNs to predict vessel movements based on recent travel history in the form of geographic time series data. Using AIS data we construct models that make accurate short, medium, and long-term predictions of future vessel location given recent travel history. We implement two primary approaches for model construction, posing the predictive task first as a classification problem and second as a regression problem. After implementing multiple variations of a deep RNN, we find that a classification approach, predicting discretized bearing and distance classes, works best, achieving the most accurate predictions of future position. Nevertheless, we still find that a successful regression approach can be formulated based on a clustering scheme, where regression targets are limited to a convex combination of engineered landmark points. By nature of the regression problem, the predicted vessel tracks are much smoother than those made by the discretized classification approach.

For neural network design, we use NOLH for hyper-parameter selection using a two-phase search process. We implement the models using the Python language and the Keras library (Chollet 2015) with a TensorFlow (Abadi et al. 2016) backend. Training the models is done using the NPS High Performance Computing center facilities, including a computing cluster running the training jobs in parallel over dozens of nodes, using 16 Nvidia GPUs with 120GB.

This type of model, which uses widely available AIS data to predict the future vessel behavior, has applications for enhancement of maritime awareness. For example, collision prevention and assistance with search-and-rescue lost vessel missions are two such applications. We briefly highlight the use of such predictive systems for another purpose, specifically anomaly detection, briefly exploring the use of prediction error as a real-time tracking mechanism for detecting improper vessel activity. Such ability to detect abnormal behavior is useful in automatically classifying suspicious ships that might engage in criminal activity, piracy, or terror, and ships suffering from an emergency that prevents them from following their normal conduct.

Overall, we find RNNs to be a viable method for predicting maritime activity. With NNs proving highly flexible, the potential for extending this work and incorporating new data sources is high, which is expected to improve the RNN's predictive power further, making it a useful and valuable tool for enhancement of maritime domain awareness.

APPENDIX

In this appendix we present an absolute positioning approach for classification models predicting a ship's future track. Although it is not included in the main body of this work, it may prove valuable for future work.

The task at hand is to try to classify the future location of a ship out of a large set of potential locations. Since the core data type is the time stamped coordinates of the ship, we need find a way to express location as a categorical variable.

One option is to use "absolute positioning" of the area of interest so that every class represents an actual region or point on the map. In this approach, it is natural to partition the area of interest into rectangular regions using a grid so that every rectangle is a class that the model can predict. The input in such a case can be either the series of past coordinates of the ship or the series of rectangles in which these coordinates are located. The most convenient method for partitioning a region into a grid is to use "geohashing."

1. Geohashing

Geohashing is a public domain geocoding system that was invented by Gustavo Niemeyer (2008). It encodes a geographic location using letters and digits in the form of a short string. The geohash system is a hierarchical spatial data structure that subdivides the space into sections of grid shape (Geohash 2018), as demonstrated in Figure 27.

Figure 27. Geohash subdivisions example
Source: Movable Type (2018)

Every additional letter or digit (in short, symbol) represents one of the 32 inner cells in the next subdivision (e.g., the letter "g" in Figure 27). Every additional symbol increases the precision of the geohash. For example, an 11-symbol long geohash will have a precision of ±7 centimeters (Niemeyer 2008).

Once the coordinates are transformed into geohashes, every coordinate falls into one of a number of classes. These classes are the set of geohash rectangles that cover our area of interest (Los Angeles region, Figure 3), or more precisely, the rectangles in the area of interest where at least one of the coordinates in the data fall. The next step is to represent the geohashed (categorical) coordinates as NN model inputs.

2. Applying One-Hot Encoding of the Geohashes to Model Inputs

To transform the geohash representation of a coordinate into a valid input for the NN, we use the one-hot encoding method. That is, for *N* unique geohash codes we create an input

vector of length *N*, where all values are zero except for one entry where the value is 1, indicating the "geohash class" of the given coordinate.

We see two advantages in using this method for input and output encoding. First, for every possible location on the map that is present in the one-hot vector, the input can be enriched to include more information such as current weather conditions or the presence of other ships in nearby locations that might affect the behavior of the ship's track. For example, by expanding the input to include another two vectors of the same length, one of them with the value of 1 at the entries that represent the number of other ships in each location and one with values between 1 and 5 to represent the sea-state at each location.

The second advantage has to do with the classification output. Instead of getting only a pair of coordinates (regression approach), the model assigns probabilities to all possible locations at once, which is a richer form of output. The model can then better describe the predictive uncertainty about the multiple potential tracks the ship might take (e.g., if it is nearing a "crossroad" where most ships turn to the right or to the left).

Yet, there are difficulties involved in this input and output representation method. When processing large sets of data as we do in this research, one soon encounters the challenge of memory and computation limits. Transforming coordinates into geohashes and creating the set of all geohashes present in the data resulted in the sets of large size, as seen in Table 9, which provides the number of unique Geohashes present in the data per chosen level of precision.

Table 9. Number of unique geohashes present in the data per precision level

Geohash Precision Level	Geographical Dimension of Geohash	Number of Unique Values in the Data	Number of Unique Values in the Area of Interest
3	160x160 km^2	86	86
4	40x40 km^2	1312	1381
5	5x5 km^2	57,784	88,400
6	1.2x1.2 km^2	645,713	~1,530,000
7	150x150 $meter^2$	2,102,578	~98,200,000

Table 9 shows that using a precision level of 5 results in 57,784 unique geohashes. This implies the model input will be composed of vectors of length 57,784. Choosing a higher level of precision makes the input size orders of magnitude larger. Since the input will be a series of hundreds or thousands of coordinates, with more dimensions for other types of data (speed, heading direction, ship type, etc.) this soon requires a big chunk of memory. More important still, building an NN with this size of input layer will force the model to be of tremendous size, with many millions of parameters, which risks very long to infeasible training time.

A common solution to this sort of problem is called dimensionality reduction. Common methods such as Principal Component Analysis (PCA) (Jolliffe 2011) or Local Linear Embedding (Roweis and Saul 2000) attempt to compress the data into lower dimensional representations, which ideally represent the underlying lower dimensional manifold on which the input data lie. Autoencoding neural networks, developed by Hinton and Salakhutdinov (2006), are one such neural-based dimensionality reduction technique. Furthermore, it is possible to 'embed' such reductions within the network itself, where the learned embedding does not attempt to support the reconstruction error, but to support the classification objective itself.

This is commonly implemented in classification problems with high dimensional data and a large number of classes, such as natural language processing, and is referred to as the use of an embedding layer (Cho et al. 2014; Goodfellow et al. 2016; Tang et al. 2014). The idea behind this is the notion that there can be found much more efficient ways to represent

the data than with sparse one-hot vectors, and that given enough training, the network would be able to learn this representation.

To test whether our data could be embedded into a lower dimensional representation, we first tried creating an autoencoding network that would take a high dimensional input of a one-hot encoded geohash, reduce the dimensionality through a set of layers, and then reconstruct the same geohash one-hot vector at the output layer. Successfully encoding and decoding the same geohash would indicate that there is a good chance we would be able to harness this embedding technique to build a reasonably sized model. Table 10 summarizes the results of a few of these experiments. The autoencoding NN's architecture is of an hourglass shape, starting from a big fully connected dense layer, then smaller layer(s) and growing back toward the output layer. The first and last layers are fully connected to a Keras' embedding layer. Using Geohash precision level of 5, such layer has 57,784 units. Precision level 6 embedding layer has 645,713 units. The connections between the embedding layers and the first and last layers compose most of the NN parameters.

Table 10. Accuracy results of geohash autoencoding networks using one-hot method

Geohash Precision Level	Model Architecture	Number of parameters	Autoencoding Accuracy
5	100-50-100	~11.6 million	72%
	500-200-50-200-500	~58 million	85%
	1000-500-100-500-1000	~116 million	89%
6	100-50-100	~130 million	64%
	500-200-50-200-500	~646 million	Too big to run
	1000-500-100-500-1000	~1292 million	Too big to run

As can be seen from Table 10, the best model is able to accurately reproduce 89% of the geohashes at precision level 5, and 64% of the geohashes at precision level 6. This is not satisfactory, since this will most likely dictate the upper boundary of the model accuracy. Furthermore, even using the embedding technique, there is a big price to be paid in the number of parameters added to the model, especially in the higher precision levels.

Another potential problem with this method is that we only encode the geohashes that appear in the past data. This automatically prevents predicting a future ship location on land, but will also not allow the model to predict a future location in a geohash that is not one of the past ship locations. There are several possible solutions; the most obvious is to increase the size of the ship track data set. Another is to add geohashes. Taking the definition of the area of interest provided in Figure 3, we get almost 90,000 geohashes at a precision level of 5, and more than 1.5 million at a precision level of 6.

3. Multi-Hot Encoding Method

As an alternative to transforming the strings of geohashes to long one-hot vectors, we also use the inherent structure of the geohash itself and create a "multi-hot" vector. A multi-hot vector is a sparse vector composed of the concatenation of multiple one-hot vectors.

In our case, each one-hot vector is used to classify one of the letters in the geohash. For example, if we wanted to represent a precision level 4 geohash, based only on the letters "B," "C," "D," "E," and "F," for example "ECDE," we would use the following multi-hot representation, given in Table 11.

Table 11. Multi-hot vector representation (precision level 4)

Geohash	Multi-hot vector representation
"ECDE"	([0, 0, 0, 1, 0], [0, 1, 0, 0, 0], [0, 0, 1, 0, 0], [0, 0, 0, 1, 0]) Or equivalently: (0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0)
* The internal brackets "[]" are just for demonstration of the way the multi-hot vector is composed of the set of one-hot vectors	

The main advantage of this method is its size and memory efficiency. Continuing the previous example, if there are only five possible letters for a length 4 geohash, we would need a vector of length $4 \times 5 = 20$ to fully represent the space of possible geohashes. However, in the one-hot method, we would need a vector of size $5^4 = 625$ to fully represent the space of possible geohashes.

Representing the entire space of possible precision level 5 geohashes requires an input vector of length $5 \times 32 = 160$ only, significantly shorter than the one-hot version. Another benefit is that in this form the network output can be any possible geohash, not only the ones that are present in the data. This is a desirable outcome, since we would have liked the model to be able to predict the most accurate future location geohash, without being dependent on it showing up earlier.

The disadvantages of this method are that some of the potential richness of the input (adding weather and other ships' locations) is lost and that we are forcing the model to learn a geohash representation that might make sense to humans but is not necessarily suitable for neural networks. Moreover, while the one-hot vector method inherently excludes land-based locations in the area of interest (only geohashes that appeared in the past are represented), the multi-hot vector does not.

It is worth mentioning that this output layer is typical for multi-labeling classification problems (Goodfellow et al., 2016). Classification problems might only try to predict one class out of the set of possible classes using one-hot encoding. In our "multi-hot" approach, we require the model to predict a concatenation of two separate probability distributions, thus "multi-labeling." This requires a customized loss function to allow effective learning.

In this multi-hot approach, we require the model to predict the symbols in the geohash as a concatenation of a number of separate probability distributions. The first 32 values in the output layer (1–32) need to sum up to one and describe the predicted probabilities for the first letter in the geohash. The next 32 values (33–64) will do the same for the second letter and so on.

In order to do that, there is a separate independent 1x32 softmax activated layer for every symbol in the geohash, all receiving the same input from the previous layer. These layers are then concatenated into one output layer of size $32 \cdot \mathrm{P}$, where P stands for the precision level of the geohash. The sum of values over each 1x32 segment is one, and therefore, the sum of the entire output layer will be P.

4. Optimization and Loss Function

Training the model, we discover that the model easily learns the first letters of the geohash, as they seldom change in our area of interest (for example, the first and second letter encode rectangles of size 2500x2500 km^2 and 1200x1200 km^2, respectively) but has a harder time learning the last letters of the geohash, which change frequently. To help the learning process, we change the weights given to each letter prediction in the loss function. For example, an error in the first letter will "cost" 1, while an error in the fifth letter will cost 5.

Let C denote the number of classes to predict ($C = 32$ symbols in this case), $y_{p,i} \in \{0,1\}$ the ground-truth for the i^{th} class of the p^{th} symbol in the geohash and $\hat{y}_{p,i} \in [0,1]$ the model prediction for the i^{th} class of the p^{th} symbol. Let P denote the precision level of the geohash and w_p the weight given to each geohash level. The loss function is:

$$\mathcal{L}(\hat{y}, y) = -\frac{1}{C \cdot P} \sum_{p=1}^{P} \sum_{i=1}^{C=32} w_p \cdot y_{p,i} \cdot log(\hat{y}_{p,i})$$

where

$$\sum_{i=1}^{C} y_{p,i} = \sum_{i=1}^{C} \hat{y}_{p,i} = 1 \; \forall \; p \in \{1,2,3 \dots P\}$$

This modification has proven to be very useful in training the model. However, as stated earlier, the results are not as accurate as the ones obtained using the relative positioning classification and the regression models. For this reason, we do not pursue this method any further and focus our efforts on the models presented in the main chapters of this thesis. We believe that the advantages listed for absolute positioning classification are not to be ignored, and that this approach can be successfully implemented for this problem or others alike.

LIST OF REFERENCES

Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M (2016) Tensorflow: A system for large-scale machine learning. *OSDI*. 265–283.

Al-Molegi A, Jabreel M, Ghaleb B (2016) STF-RNN: Space Time features-based recurrent neural network for predicting people next location. *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*. (IEEE), 1–7.

Ba JL, Kiros JR, Hinton GE (2016) Layer normalization. arXiv preprint arXiv:1607.06450.

Bergstra J, Bengio Y (2012) Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13(Feb):281–305.

Bishop C, Bishop CM (1995) *Neural Networks for Pattern Recognition* (Oxford University Press, Cambridge, UK).

Bomberger NA, Rhodes BJ, Seibert M, Waxman AM (2006) Associative learning of vessel motion patterns for maritime situation awareness. *Information Fusion, 2006 9th International Conference on*. (IEEE), 1–8.

Bridle JS (1990) Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. *Neurocomputing*. (Springer, Berlin, Heidelberg), 227–236.

Cheng Y (1995) Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17(8):790–799.

Cho K, Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y (2014) Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.

Chollet F (2015) *Keras Documentation*. Accessed February 10, 2018, https://keras.io/

Cioppa TM, Lucas TW (2007) Efficient nearly orthogonal and space-filling Latin hypercubes. *Technometrics* 49(1):45–55.

De Brébisson A, Simon É, Auvolat A, Vincent P, Bengio Y (2015) Artificial neural networks applied to taxi destination prediction. arXiv preprint arXiv:1508.00021.

Eck D, Schmidhuber J (2002) Learning the long-term structure of the blues. *International Conference on Artificial Neural Networks*. (Springer, Berlin, Heidelberg), 284–289.

Fukunaga K, Hostetler L (1975) The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory* 21(1):32–40.

Gardiner C (2009) *Stochastic methods* (Springer, Berlin).

Geohash (2018) *Wikipedia*. Accessed March 10, 2018, https://en.wikipedia.org/wiki/Geohash.

Gers FA, Schraudolph NN, Schmidhuber J (2002) Learning precise timing with LSTM recurrent networks. *Journal of Machine Learning Research* 3(Aug):115–143.

Goodfellow I, Bengio Y, Courville A, Bengio Y (2016) *Deep Learning* (MIT Press Cambridge, MA).

Graves A (2012) *Supervised Sequence Labelling with Recurrent Neural Networks* (Springer, Berlin, Heidelberg).

Graves A, Liwicki M, Fernández S, Bertolami R, Bunke H, Schmidhuber J (2009) A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31(5):855–868.

Gunst RF (1996) *Response surface methodology: process and product optimization using designed experiments* (Taylor & Francis, London, UK).

Harati-Mokhtari A, Wall A, Brooks P, Wang J (2007) Automatic Identification System (AIS): Data reliability and human error implications. *Journal of Navigation* 60(3):373–389.

Hastie T, Tibshirani R, Friedman J (2009) Unsupervised learning. *The Elements of Statistical Learning*. (Springer-Verlag, New York), 485–585.

Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. *Science* 313(5786):504–507.

Ioffe S, Szegedy C (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

Jiang X, de Souza EN, Liu X, Soleimani BH, Wang X, Silver DL, Matwin S (2017) Partition-wise recurrent neural networks for point-based AIS trajectory classification. *Computational Intelligence*:6.

Jolliffe I (2011) Principal component analysis. *International Encyclopedia of Statistical Science*. (Springer-Verlag, New York), 1094–1096.

Karpathy (2015) The unreasonable effectiveness of recurrent neural networks. Andrej Karpathy Blog. Accessed August 11, 2018, http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. arXiv:1412.6980 [cs].

Kleijnen JPC, Sanchez SM, Lucas TW, Cioppa TM (2005) State-of-the-art review: A user's guide to the brave new world of designing simulation experiments. *INFORMS Journal on Computing* 17(3):263–289.

Kordmahalleh MM, Sefidmazgi MG, Homaifar A, Liess S (2015) Hurricane trajectory prediction via a sparse recurrent neural network. *Proceedings of the 5th International Workshop on Climate Informatics*.

Kubat M, Holte RC, Matwin S (1998) Machine learning for the detection of oil spills in satellite radar images. *Machine Learning* 30(2–3):195–215.

Leca CL, Nicolaescu I, Rîncu CI (2015) Significant location detection & prediction in cellular networks using artificial neural networks. *Computer Science and Information Technology* 3(3):81–89.

Liu Q, Wu S, Wang L, Tan T (2016) Predicting the next location: A recurrent model with spatial and temporal contexts. *AAAI*. 194–200.

MarineCadastre.gov (2018) Vessel traffic data. MarineCadastre.gov. Accessed May 21, 2018, https://marinecadastre.gov/ais/

Marine Traffic.com (2018) What is the significance of the AIS Shiptype number? *MarineTraffic Help*. Accessed August 3, 2018, http://help.marinetraffic.com/hc/en-us/articles/205579997-What-is-the-significance-of-the-AIS-Shiptype-number-.

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

Mazzarella F, Arguedas VF, Vespe M (2015) Knowledge-based vessel position prediction using historical AIS data. *Sensor Data Fusion: Trends, Solutions, Applications (SDF), 2015*. (IEEE), 1–6.

Movable Type (2018) Geohash. Movable Type. Accessed February 17, 2018, https://www.movable-type.co.uk/scripts/geohash.jpg.

Nannini CJ, Wan H (2011) Designs for large-scale simulation experiments, with applications to defense and homeland security. *Design and Analysis of Experiments, Volume 3: Special Designs and Applications* 810:413.

Niemeyer G (2008) *Geohash*. Geohash. Accessed April 17, 2018, http://geohash.org/.

Pallotta G, Horn S, Braca P, Bryan K (2014) Context-enhanced vessel prediction based on Ornstein-Uhlenbeck processes using historical AIS traffic patterns: Real-world experimental results. *Information fusion (fusion), 2014 17th international conference on*. (IEEE), 1–7.

Pallotta G, Vespe M, Bryan K (2013) Vessel pattern knowledge discovery from AIS data: A framework for anomaly detection and route prediction. *Entropy* 15(6):2218–2245.

Pascanu R, Gulcehre C, Cho K, Bengio Y (2013) How to construct deep recurrent neural networks. *arXiv:1312.6026 [cs, stat]*.

Ristic B, La Scala BF, Morelande MR, Gordon NJ (2008) Statistical analysis of motion patterns in AIS Data: Anomaly detection and motion prediction. *FUSION*. 1–7.

Robbins H, Monro S (1985) A stochastic approximation method. *Herbert Robbins Selected Papers*. (Springer, New York, NY), 102–109.

Roweis ST, Saul LK (2000) Nonlinear dimensionality reduction by locally linear embedding. *Science* 290 (5500): 2323–2326.

Rumelhart DE, Hinton GE, Williams RJ (1985) *Learning internal representations by error propagation*, California Univ San Diego La Jolla Inst for Cognitive Science. (MIT Press, Cambridge, MA). http://www.dtic.mil/get-tr-doc/pdf?AD=ADA164453

Sanchez SM, Wan H (2012) Work smarter, not harder: A tutorial on designing and conducting simulation experiments. *Proceedings of the Winter Simulation Conference*. (Winter Simulation Conference, Piscataway, New Jersey: Institute of Electrical and Electronic Engineers), 170.

Shazeer N, Mirhoseini A, Maziarz K, Davis A, Le Q, Hinton G, Dean J (2017) Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. arXiv:1701.06538 [cs, stat].

Tang D, Wei F, Yang N, Zhou M, Liu T, Qin B (2014) Learning sentiment-specific word embedding for twitter sentiment classification. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1555–1565.

Tieleman T, Hinton G (2012) Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning* 4(2):26–31.

Tu E, Zhang G, Rachmawati L, Rajabally E, Huang GB (2016) Exploiting AIS data for intelligent maritime navigation: A comprehensive survey. arXiv:1606.00981 [cs].

US Coast Guard Navigation Center (2018) Types of automatic identification systems (per ITU-R M.1371 and IEC standards). Navigation Center. Accessed August 3, 2018, https://www.navcen.uscg.gov/?pageName=typesAIS.

Wijaya WM, Nakamura Y (2013) Predicting ship behavior navigating through heavily trafficked fairways by analyzing AIS data on apache HBase. *2013 First International Symposium on Computing and Networking-Across Practical Development and Theoretical Research (CANDAR)*. (IEEE), 220–226.

Werbos PJ (1988) Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks* 1(4):339–356.

Xingjian SHI, Chen Z, Wang H, Yeung DY, Wong WK, Woo WC (2015) Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *Advances in Neural Information Processing Systems*. 802–810.

Young BL (2017) Predicting vessel trajectories from AIS data using R. Master's thesis, Department of Operations Research, Naval Postgraduate School, Monterey, CA.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California