Spatial Prediction with Long Short-Term Memory Models

Master's Thesis in Statistics

Eirik Grasdal Økland

Supervisor
Hans Karlsen

Department of Mathematics
University of Bergen

January 2022

Abstract

This thesis examines the appliance of deep learning methods within a spatiotemporal statistical structure. The experiment intends to use a predictive model to predict a spatial boundary based on spatially parallel neighbour points, observed and trained on the historical temporal data. We define this as a regression problem on sequential data and approach it with artificial neural networks consisting of Long Short-Term Memory Models. The experiment is facilitated by data simulated from a Spatio-temporal GARCH model. We reshape the data and propose two iterative data structures to provide the prediction model with increased information—one true to the original three dimensions and one restructured to two dimensions. Our experiment does not succeed, as our prediction models cannot deliver satisfactory results. We discuss the reasons for this. Additionally, the results from our experiment show that the restructured two-dimensional data perform better than the original shape. We discuss this as well.

Acknowledgements

I would like to thank my supervisor, Hans Arnfinn Karlsen, for his guidance and the initial idea behind this thesis. I would also like to thank Sondre Hølleland for the helpful communication about his research and code.

Additionally, I would like to show my appreciation to the administration at the Department of Mathematics. I am grateful for their help and service during the trialling circumstances of Covid-19.

Lastly, I wish to thank my family, friends and fellow students.
Without their support, this would not be conceivable.
Thank you.

Contents

List of Figures

1 Introduction

The modelling of spatiotemporal data has been breaking new ground, and mapping data over both space and time is more reliable with a theoretical and empirical background. From the starting point of the time series model Generalized Autoregressive Conditional Heteroskedasticity (GARCH), introduced by Bollerslev (1986), Karlsen first suggested a spatiotemporal extension which later was presented by Hølleland (2016). In later years the extension has been further developed, for example in Hølleland & Karlsen (2020a). His application for considering temperature variability on Svalbard, and related work, shows progress for modelling volatility in time and space.

Recurrent Neural Networks (RNN) was first presented by Rumelhart et al. (1986). They introduced the use of backpropagation in Neural Networks, allowing networks to process sequences of data over time. The RNNs ability to "remember" along the temporal dimension let it learn time-dependent dynamics. Hochreiter & Schmidhuber (1997b) introduced an RNN architecture called Long Short-Term Memory (LSTM) model that improved on RNNs efficiency in remembering long term by letting the layers more effectively drop excess memory. LSTM is recognised for its applicability to speech recognition, text classification and forecasting time series because of its ability to process and learn from sequenced data.

In this thesis, we will consider and experiment with LSTM models on data simulated from a Spatio-temporal GARCH model. We will inspect whether the LSTM models can learn the properties of a spatiotemporal model when trained on this data. Our approach considers two formats of the data when feeding it into the model; the original three-dimensional data structure and a simplified two-dimensional data structure. We compare different variants of LSTM networks, trained on both data structures, and examine how they perform.

To best conduct this study, we will inspect a boundary problem, which was presented with the ST-GARCH in Hølleland (2016). It was handled then by modelling the spatial aspect of the process as circular (one spatial dimension) or on a torus (two spatial dimensions). There was alternatively suggested solving the problem by conditioning on the boundary but was disregarded as it caused considerable data loss. We will return to this alternative method and try to counter the problem of data loss with predicted data.

2 Spatial Temporal Models

The scientific field of Time Series has been a progressive field within modern statistics and data analysis. Over the last 40 years, there have been brought on a wide range of models and extensions in regard to the Generalized Autoregressive Conditional Heteroskedasticity (GARCH) process. Since Engle proposed the Autoregressive Conditional Heteroskedasticity (ARCH) in Engle (1982) and Bollerslev expanded the process to the Generalized ARCH, GARCH, in Bollerslev (1986), there have been proposed many new versions and extensions. Most of which are listed by Bollerslev in his "Glossary to ARCH (GARCH)" Bollerslev et al. (2008). This Glossary gives an impression of the model's popularity and applicability.

The spatiotemporal extension of the GARCH model (ST-GARCH) was first suggested by Hans Karlsen and then presented by Hølleland (2016). In later years the extension has been further developed, for example in Hølleland & Karlsen (2020a). His application for considering temperature variability on Svalbard, and related work, shows progress for modelling volatility in time and space.

This chapter serves as an introduction to the elementary concepts for the spatiotemporal models. We will review the foundational theory, before we start approaching the Spatio-temporal GARCH Model, its properties and the boundary problem.

2.1 Time Series

The properties of Time Series are foundational in the Spatio-temporal GARCH model. Therefore, we will approach the fundamentals in this section before going forward. We cite Brockwell et al. (2016) as our primary source on this theory, as we reference the following definitions to it.

Time series is the collective concept of a series of data ordered temporally. Commonly, it is denoted as $\{X_t\}$, a stochastic process in discrete time, $t \in \mathbb{Z}$. It appears in all disciplines where time is a controlled variate, most recognizable in examples like finance and weather forecast. We proceed by displaying some important definitions.

Definition 2.1 (Mean and Covariance Functions)**.**
Let $\{X_t : t \in \mathbb{Z}\}$, be a time series with $\mathbb{E}(X_t^2) < \infty$. The mean function of $\{X_t\}$ is $\mu_t = \mu_X(t) = \mathbb{E}(X_t)$. The covariance function of $\{X_t\}$ is

$$\gamma_X(r, s) = \text{Cov}(X_r, X_s) = \mathbb{E}(X_r - \mu_r)(X_s - \mu_s), \tag{1}$$

for all integers r and s.

Definition 2.2 (Stationarity and Weak Stationarity)**.**
A time series $\{X_t, \quad t = 0, \pm 1, ...\}$ is said to be stationary if the mean $\mathbb{E}(X_t) = \mu_X(t)$ is independent of t, and the covariance $\text{Cov}(X_{t+h}, X_t) = \gamma_X(t + h, t)$ is independent of t for any integer h.This means that for a time series to be stationary $\{X_t\}$ has to have the same second-order properties as $\{X_{t+h}\}$ for any h.

$\{X_t\}$ is weakly stationary if

(a) $\mathbb{E}X_t^2 < \infty$,

(b) μ_t is independent of t,

(c) $\gamma_X(t + h, t)$ is independent of t for each h.

Definition 2.3 (White Noise Process)**.**
A stationary time series $\{Z_t\}$ of uncorrelated variables with zero mean is called a white noise process, denoted as $\{Z_t\}$ WN$(0, \sigma^2)$. The variables $\{Z_t\}$ are also independent, if they have a Gaussian distribution. The process has $\gamma_Z(h) = \delta_{0,h}\sigma^2$, where δ is the Kronecker-delta symbol and $h =\mid r - s \mid$ is the lag. The mean function is constant and the covariance function is only dependent on h, thus the white noise process is weakly stationary.

With these definitions described, we proceed to describe the fundamental models and processes.

2.1.1 ARMA

Definition 2.4 (Autoregressive Moving Average model)**.**
The time series $\{X_t\}$ is an Autoregressive Moving Average (ARMA) process of order (p, q) if it is stationary and

$$X_t - \phi_1 X_{t-1} - \cdots - \phi_p X_{t-p} = Z_t + \theta_1 Z_{t-1} + \cdots + \theta_q Z_{t-q}, \quad \text{for } t \in \mathbb{Z}, \tag{2}$$

where $\{Z_t\} \sim$ WN$(0, \sigma^2)$. The polynomial $\phi(z) = 1 - \theta_1 - \cdots - \theta_p z^p$ and $\theta(z) = 1+\theta_1 z+\cdots+\theta_q z^q$ are the autoregressive and the moving-average polynomial, respectively.

An assumption that $\{X_t\}$ is an ARMA(p, q)-process further assumes that $\phi(z)$ and $\theta(z)$ has no roots on the unit circle. Following, if the white noise process $\{Z_t\}$ and the polynomials $\phi(z)$ and $\theta(z)$ are given, then 2 defines an ARMA(p, q) model with $\{X_t\}$ as a possible stationary solution. The model's invertibility and causality depend on the roots of $\phi(z)$ and $\theta(z)$. A causal model comes from $\phi(z)$ having all roots strictly outside the unit circle, while the model is invertible if $\theta(z)$ has similar roots. The causal model let

us represent X_t in terms of past and present values of $\{Z_t\}$. Invertibility let us represent Z_t in terms of past and present values of $\{X_t\}$.

Definition 2.5 (Autoregressive Model)**.**
The time series $\{X_t\}$ is a Autoregressive (AR) process of order p if

$$X_t = \phi_1 X_{t-1} + \cdots + \phi_p X_{t-p} + Z_t \quad \text{for } t \in \mathbb{Z} \tag{3}$$

where $\{Z_t\} \sim \text{WN}(0, \sigma^2)$ and $\phi_p \neq 0$.

The AR model is equivalent to an ARMA(p, q) with $q = 0$. Therefore the conditions for stationarity and causality persist for an AR(p). From equation 3, an AR(p) is always invertible.

Definition 2.6 (Moving Average Model)**.**
The time series $\{X_t\}$ is a Moving Average-process (MA) of order q if

$$X_t = Z_t + \theta_1 Z_{t-1} + \cdots + \theta_q Z_{t-q} \quad \text{for } t \in \mathbb{Z}, \tag{4}$$

where $\{Z_t\} \sim \text{WN}(0, \sigma^2)$ and $\theta_q \neq 0$.

The MA model is equivalent to an ARMA(p, q) with $p = 0$. Equation 4 also states that MA(q) always is causal and stationary. The conditions for invertibility for ARMA(p, q) persist to MA(q) as well.

2.1.2 GARCH

The Autoregressive Conditional Heteroskedasticity process (ARCH) expands further to the aforementioned models. With its own expansion, Generalized ARCH (GARCH), it addresses processes of varying volatility. We will introduce both of their definitions and essential properties.

Definition 2.7 (ARCH(p))**.**
Let Z_t be iid WN(0,1) and σ_t^2 be a positive function of $X_s, s < t$. Then $\{X_t\}$ is an ARCH(p) process if for each $t \in \mathbb{Z}$,

$$\begin{aligned} X_t &= \sigma_t Z_t \\ \sigma_t^2 &= \alpha_0 + \sum_{i=1}^{p} \alpha_i X_{t-i}^2, \end{aligned} \tag{5}$$

where p determine the order of the process.

Definition 2.8 (GARCH(p, q)).
Let Z_t be iid WN(0,1). Then $\{X_t\}$ is a GARCH(p, q)-processes if for each $t \in \mathbb{Z}$,

$$\begin{aligned} X_t &= \sigma_t Z_t \\ \sigma_t^2 &= \alpha_0 + \sum_{i=1}^{p} \alpha_i X_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2, \end{aligned} \tag{6}$$

where p and q determine the order of the process.

A natural property of GARCH is that GARCH$(p, 0)$ is the same as an ARCH(p) process.

Moments
We derive the two central moments of the GARCH(p, q) process $\{X_t\}$. The first moment is calculated on the following properties. σ_t is only dependent on past values of $\{X_s\}$ and σ_s, and thus, σ_t and Z_t are independent. In result,

$$\mathbb{E}(X_t) = \mathbb{E}(\sigma_t Z_t) = \mathbb{E}(\sigma_t)\mathbb{E}(Z_t) = 0. \tag{7}$$

The expectation for X_t shown in the first moment is important in the second moment as it facilitates the calculation. Thus,

$$\mathrm{Var}(X_t) = \mathbb{E}X_t^2 = \mathbb{E}(\sigma_t^2 Z_t^2) = \mathbb{E}(\sigma_t^2)\mathbb{E}(Z_t^2) = \mathbb{E}\sigma_t^2, \tag{8}$$

following from the independence of σ_t^2 and Z_t^2, and $\mathbb{E}(Z_t^2) = \mathrm{Var}(Z_t) = 1$. The variance of X_t is finite if and only if $\mathbb{E}\sigma_t^2$ is finite.

Stationarity
Under the assumption that $\{X_t\}$ is a weakly stationary process, we derive a necessary condition for weak stationarity of GARCH processes. The second moment must be finite and $\mu_X = \mathbb{E}(X_t) = 0$. As $\{X_t\}$ is weakly stationary by assumption, point (c) of definition 2.2 also holds. The covariance function only depend on the lag h and we have that $\gamma_X(0) = \mathrm{Var}(X_t) = \mathbb{E}(\sigma_t^2)$ for all $t \in \mathbb{Z}$. Thus,

$$\begin{aligned} \gamma_X(0) = \mathrm{Var}(X_t) = \mathbb{E}(\sigma_t^2) &= \alpha_0 + \sum_{i=1}^{p} \alpha_i \mathbb{E}(X_t^2 - i) + \sum_{j=1}^{q} \beta_j \mathbb{E}(\sigma_{t-j}^2) \\ &= \alpha_0 + \sum_{i=1}^{p} \alpha_i \gamma_X(0) + \sum_{j=1}^{q} \beta_j \gamma_X(0), \end{aligned} \tag{9}$$

which leads to

$$\sigma_t^2 = \mathrm{Var}(X_t) = \frac{\alpha_0}{1 - \sum_{i=1}^{p} \alpha_i - \sum_{j=1}^{q} \beta_j} \tag{10}$$

In order for $\text{Var}(X_t) < \infty$, its necessary that

$$\sum_{i=1}^{p} \alpha_i + \sum_{j=1}^{q} \beta_j < 1. \tag{11}$$

X_t is weakly stationary under these conditions, and subsequently, (10) gives the asymptotic variance of the process.

2.2 Spatial Statistics

We will present the fundamentals from spatial statistics before proceeding to the spatiotemporal models. We have relied on Cressie (2015) for the theory presented here.

Time series data have a rigid relation to order, as it relies on the progression of time. Yesterday will not come after today, and tomorrow will not precede yesterday. Time moves in a decided direction, and we can rely on that. Spatial data do not adhere to any rigid structure like this. The upside on a map is usually the direction north but could quickly be oriented around something else. For example, the blueprint of a building might be oriented according to the side of the building. In this case, the direction of the data is changed. Compared to time series data, we could be moving backwards. Spatial order can also be different in a different data set. What constitutes a nearer neighbour might be different in each set.

Figure 1: One-dimensional Spatial grid.

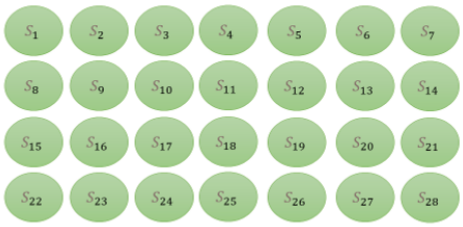


Figure 2: Two-dimensional Spatial grid.

There are several ways of organising spatial data, but we will focus on the continuous spatial process in our case. We structure the process on a grid, which can be one-dimensional, like figure 1, or two-dimensional, like figure 2. Figure 1 exemplify a grid suitable for a spatial model for a bus route, while figure 2 would be suitable for a map. Onwards we will engage with the two-dimensional grid.

We denoted such a grid according to its spatial points, $\{s_1, ..., s_n\}$ where $s_i \in \mathbb{Z}^d, i = 1, ..., n$. These spatial points are accompanied with a neighbourhood structure that defines the points neighbouring relations. Thus, regulating the spatial correlation. We organise this neighbourhood structure in a neighbourhood matrix.

Definition 2.9 (Neighbourhood Matrix)**.**
Let s_i and s_j be two grid and define $\mathcal{N}(s_j) = \{s_k : s_k \text{ and } s_j \text{ are neighbours}\}$. Then s_i

and s_j are neighbours if $s_i \in \mathcal{N}(s_j)$.

We denote $\boldsymbol{\mathcal{W}}$ as the neighbourhood matrix and let $\boldsymbol{\mathcal{W}} = \{\omega_{ij} : 1 \leq i, j \leq n\}$, and then define

$$\omega_{ij} = \mathbf{1}(s_j \in \mathcal{N}(s_i)),$$

where $\mathbf{1}(\cdot)$ is the indicator function and $\mathcal{N}(s_i)$ is the neighbourhood of s_i.

Accordingly, we have a neighbourhood matrix of zeros and ones, where one indicates a neighbour relation. $\mathcal{N}$ sources a common neighbouring pattern. It enforces mutual neighbouring relation, as a point can not be a point's neighbour without it being the same back. Thus, $s_i \in \mathcal{N}(s_j) \Leftrightarrow s_j \in \mathcal{N}(s_i)$. It also denies points from being their own neighbour, thus, $s_i \notin \mathcal{N}(s_i)$. This leads the neighbourhood matrix, $\boldsymbol{\mathcal{W}}$, to be a symmetric matrix with zeros along the diagonal.

2.3 Spatio-temporal GARCH

This section builds on the previous topics and describes the Spatio-temporal GARCH model (ST-GARCH). We rely on the works of Hølleland and Karlsen, and mainly, the theory published in Hølleland & Karlsen (2020b). Accordingly, we ensure sufficient knowledge of the model behind the data we aim to simulate and the task we set out to solve. Foremost, we define the ST-GARCH according to Hølleland & Karlsen (2020b).

Definition 2.10 (Spatio-Temporal GARCH)**.**
Let $\alpha : \mathbb{Z} \times \mathbb{Z}^d \rightsquigarrow \mathbb{R}_0$ be a function with finite support. For fixed $i, \alpha_i : \mathbb{Z}^d \rightsquigarrow \mathbb{R}_0$. The function β is defined in the same way. We refer to α, β as the parameter functions. The ST-GARCH model is given by

$$\begin{aligned} X_t(u) &= \sigma_{(t)}(u) Z_t(u), \quad u \in \mathbb{Z}^d \\ \sigma_t^2(u) &= \omega + \sum_i \sum_v \alpha_i(v) X_{t-i}^2(u-v) + \sum_i \sum_v \beta_i(v) \sigma_{t-i}^2(u-v), \quad u \in \mathbb{Z}^d, \end{aligned} \tag{12}$$

for $t \in \mathbb{Z}$.

The modelled process is $\{X_t(u)\}$, while $\{Z_t(u)\}$ is a residual process and $\{\sigma_t(u)\}$ is the volatility process.

From the definition it follows, $\mathbb{A}_{1i} = \{v \in \mathbb{Z}^d : \alpha_i > 0\}$ and $\mathbb{A}_{2i} = \{v \in \mathbb{Z}^d : \beta_i > 0\}$ for $i \geq 1$. For $i < p$, the model allows for some zero-valued $\alpha_i(v)$ for $v \in \mathbb{A}_{1i}$ and likewise for the β's.

According to the largest (p, q), the order of the model is defined. Thus, $\mathbb{A}_{1p}$ and $\mathbb{A}_{2q}$ are non-empty. For a defined order, the second part of 12 can be expressed more as

$$\sigma_t^2(u) = \omega + \sum_{i=1}^{p} \sum_{v \in \mathbb{A}_{1i}} \alpha_i(v) X_{t-i}^2(u-v) + \sum_{i=1}^{q} \sum_v \beta_i(v) \sigma_{t-i}^2(u-v). \tag{13}$$

Thus, we state that $\theta \overset{\text{def}}{=} (\omega, \boldsymbol{\alpha}, \boldsymbol{\beta})$ for the parameter vector contained in the parameter space Θ. Here $\boldsymbol{\alpha} \overset{\text{def}}{=} \{\alpha_i(v), v \in \mathbb{A}_{1i}, i = 1, ..., p\} and \boldsymbol{\beta} \overset{\text{def}}{=} \{\beta_i(v), v \in \mathbb{A}_{2i}, i = 1, ..., q\}$. This uphold with the restriction that $\omega > 0$.

2.3.1 Model Properties

The ST-GARCH model brings elements from an ordinary GARCH model into its properties, appropriate for the added spatial dimension. We will describe the properties

according to the new terms. But we will keep the subject elementary to serve the purpose of the thesis. For further understanding and proof, we recommend Hølleland & Karlsen (2020b).

Moments

As for the GARCH model, we derive the two central moments of the ST-GARCH(p, q) process $\{X_t\}$. Similar to GARCH, $\sigma_t(\boldsymbol{u})$ and $Z_t(\boldsymbol{u})$ are independent. So,

$$\mathbb{E}(X_t(\boldsymbol{u})) = \mathbb{E}(\sigma_t(\boldsymbol{u})Z_t(\boldsymbol{u})) = \mathbb{E}(\sigma_t(\boldsymbol{u}))\mathbb{E}(Z_t(\boldsymbol{u})) = 0. \tag{14}$$

Thus, the second moment will define the variance of $X_t(\boldsymbol{u})$. This brings the calculation down to

$$\text{Var}(X_t(\boldsymbol{u})) = \mathbb{E}X_t^2(\boldsymbol{u}) = \mathbb{E}(\sigma_t^2(\boldsymbol{u})Z_t^2(\boldsymbol{u})) = \mathbb{E}(\sigma_t^2(\boldsymbol{u}))\mathbb{E}(Z_t^2(\boldsymbol{u})) = \mathbb{E}\sigma_t^2(\boldsymbol{u}). \tag{15}$$

Stationarity

We will follow the derivation of weakly stationarity of the ST-GARCH from Hølleland (2016). When doing this, we do acknowledge that derives and proves stationarity for a Circular ST-GARCH under certain conditions.

We assume $\{X_t(\boldsymbol{u})\}$ to be a weakly stationary process, as we did for GARCH, and follow a similar derivation. We stated that $\mathbb{E}(X_t(\boldsymbol{u})) = 0$, so we fulfill one condition allready. The remaining conditions require; $\mathbb{E}X_t^2(\boldsymbol{u}) = \mathbb{E}(\sigma_t^2(\boldsymbol{u})) < \infty$ and $\gamma_X(s,t) = \gamma_X(h)$. The derivation follows

$$\begin{aligned}\sigma^2 &= \text{Var}(X_t(\boldsymbol{u})) = \mathbb{E}(\sigma_t^2(\boldsymbol{u})) \\ &= \alpha_0 + \sum_{s=1}^{p}\sum_{\boldsymbol{v}\in\mathbb{\Delta}_{1i}} \alpha_s((\boldsymbol{v}))\mathbb{E}(X_t^2 - s)(\boldsymbol{u}-\boldsymbol{v}) + \sum_{s=1}^{q}\sum_{\boldsymbol{v}\in\mathbb{\Delta}_{2i}} \beta_s(\boldsymbol{v})\mathbb{E}(\sigma_{t-s}^2)(\boldsymbol{u}-\boldsymbol{v}).\end{aligned} \tag{16}$$

Under our assumptions $\sigma^2 = \mathbb{E}(\sigma_t^2(\boldsymbol{u})) = \mathbb{E}(X_t^2(\boldsymbol{u}))$ for all $t \in \mathbb{Z}$. So,

$$\sigma_t^2 = \alpha_0 + \sum_{s=1}^{p}\sum_{\boldsymbol{v}\in\mathbb{\Delta}_{1i}} \alpha_s((\boldsymbol{v}))\sigma^2 + \sum_{s=1}^{q}\sum_{\boldsymbol{v}\in\mathbb{\Delta}_{2i}} \beta_s(\boldsymbol{v})\sigma^2. \tag{17}$$

Then solving for σ^2,

$$\sigma_t^2 = \alpha_0 \left(1 - \sum_{s=1}^{p}\sum_{\boldsymbol{v}\in\mathbb{\Delta}_{1i}} \alpha_s((\boldsymbol{v})) - \sum_{s=1}^{q}\sum_{\boldsymbol{v}\in\mathbb{\Delta}_{2i}} \beta_s(\boldsymbol{v})\right)^{-1}. \tag{18}$$

As a result, for the assumption for $\{X_t(\boldsymbol{u})\}$ to uphold, we must have $\mathbb{E}X_t^2(\boldsymbol{u}) = \sigma^2 < \infty$. This then causes

$$\sum_{s=1}^{p} \sum_{\boldsymbol{v} \in \mathbb{\Delta}_{1i}} \alpha_s((\boldsymbol{v})) + \sum_{s=1}^{q} \sum_{\boldsymbol{v} \in \mathbb{\Delta}_{2i}} \beta_s(\boldsymbol{v}) < 1. \tag{19}$$

For these assumptions the ST-GARCH $\{X_t(\boldsymbol{u})\}$ is considered weakly stationary.

2.3.2 The Boundary Problem

The development/definition of the model for the ST-GARCH process occurs without limitations regarding the spatial aspect. This assumption is theoretically sound, relying the model on an infinite spatial structure, but is practically non-applicable/existent. A real-life application would be limited to the range of which is observed and measured, as we cannot observe infinite amounts, nor assume unobserved data as assume. Thus we have a boundary between an area which is observed/accounted for and the unobserved.

We can call the observed area our area of interest. Within this area all spatial points are based on the same spatial structure. Each point have either four neighbouring points, in the case of a rook contiguity neighbourhood, or eight neighbouring points, for queen contiguity neighbourhood, except points at the boundary, or close to it. We consider the queen contiguity from here on, and in that case a point on the boundary would lack three of eight neighbours, only five would be observed. While the corner points would only have three out of eight neighbouring points observed. Considering more than first-order neighbours, more points inside the boundary are affected, in terms of second- and third-order neighbours. Consequently, lack of information on the boundary propagates inwards in the area of interest.

As mentioned earlier in this chapter, Hølleland (2016) proposes the circular model to counter this problem. By doing so, linking boundary point as neighbours to opposing edge, any truncation effect is avoided. However it creates a closed environment, and implies a dependency between opposing edges. These has been shown to be solvable issues, making the circular model proficient. Yet we want to inspect the alternative solution proposed.

The alternative approach is to reduce the area of interest, and as a result we assume a new boundary, now encircled by the actual former boundary. That way we have a boundary with observed neighbours. We use the former boundary for calculating $\sigma_t(\boldsymbol{u})$, but leave it out out of the parameter estimation. That way we ensure better performance within the reduced area, by avoiding unobserved values. Granted this causes loss of data. Which may be of lesser significance on larger areas. For example a grid of 30×30 would have an encircling boundary of $2(30 + 30 - 2) = 116$ which would decrease the data amount from 900 to 784. This means we still have 87,11% of the data maintained for

the parameter estimation. While if the area supposedly would be of dimensions 10×10 and the boundary would contain $2(10 + 10 - 2) = 36$ points of the total 100 in the area. Leaving only 64%, less than two thirds, of the observed data to full use.

All observed data should be considered vital as observations may be sparse. Therefore we propose the use of a predicted outer boundary. Instead of a reduction of the area of interest, we would make use of a prediction model to construct buffer data to be used in calculation of $\sigma_t(\boldsymbol{u})$ on the boundary. Thus all observations can be used in parameter estimation. This predicted boundary will not be as complete as observed data, but would pose as an approximate mapping around our observations. The goal is that accurate predictions of the underlying model in this outer boundary acts as realistic dependencies for the inner boundary, and, conceivably, we will have an area of interest which is less affected by this edge effect. In this way we still avoid a truncation effect, but keep boundaries intact.

A paradoxical issue is where the equilibrium between provided data for prediction model and benefit of avoiding truncating effect lies.As discussed earlier, smaller spatial areas are far more exposed to truncating effect, but leaves little spatial data to train a prediction model. Constructing such a prediction model on solely spatial terms of a smaller area would not be effective. Consequently, we will construct a prediction model that learns over time.

3 Deep Learning

Deep learning is an expanding field of research where new methods and appliances are constantly being developed and presented. New standards are set continuously, and it can be hard to keep track. The source material in this field is vast, so there is a need to pick out the most relevant theory. Therefore, we will describe some general basics of the field and some more specific subjects for our experiment. Consequently, we will leave out some concepts. For more extensive literature on the theoretical aspect of deep learning, we refer to Goodfellow et al. (2016), Marsland (2011) and Efron & Hastie (2016). Brownlee (2018) and Chollet (2018) are useful resources for practical implementation.

This chapter will serve as a theoretical entry point for the method we have decided to apply to the boundary problem. We aim to predict numerical values based on sequential input of numerical values. The values are continuously distributed, so we will have to phrase it as a regression problem. There are many ways to go about predictive models, but not all are suitable for regression. We have decided to apply a Long Short Term Memory Model (LSTM) to our problem. LSTM is a type of recurrent neural network (RNN) often used on time series data (Sezer et al. (2020)). Its ability to process sequences and interpret temporal data makes it suitable for our task.

The overall approach we have to solving the task is called Supervised Learning. This means we train our prediction model by supplying it with input values x and expected output values y. The model's task is to learn from this relationship and estimate a function describing it. So, when fully trained and fed values with the same form as x, it should predict a $\hat{y}$, which is equivalent to y, or at least approximate. This is appropriate for our experiment as we simulate the data ourselves, supplying input and output values to train our model. The goal is for our model to estimate the relationship between our boundaries to a function.

3.1 Deep Learning Preliminaries

We will explain some fundamental machine learning concepts ahead of describing the relevant models. Much of this is general knowledge within machine learning, but we find it helpful to establish them before proceeding. The definitions are all based on the literature at the chapter's start.

Artificial Neural Networks

We introduce the basic ideas and structure behind the most common method within deep learning. Artificial Neural Network is a collective designation to computing systems with several connected nodes. Each node processes the input and transmits the result

as a signal to all nodes connected to it. The nodes and the connections are assigned with weights and distributed in layers. These layers are organised as an input layer, processing the input, one or several hidden layers, for learning, and an output layer, to process the learned "function" to viable output. An neural network used for machine learning purposes adapts to the problem and data by adjusting these weights. The structure of the network remains the same while the weights are adjusted. This way, the neural network regulates which processes should be most influential for the output. So, a classification neural network will adjust its weights each time it puts the wrong label on input to correct its mistake for next time. A regression neural network will do the same when suggesting an inaccurate value to come closer the next time. However, rather than based on right or wrong value, it should assess how far off the suggestion was.

Data Splitting

Ahead of the training process for a prediction model, the data intended for use, both target and feature data, is split into three. We need a set of data to train the model, a set to validate the progression under training and a set to evaluate the model. We do not want these sets to interfere with each other, as it would compromise the metrics. So, we distribute the data into separate sets; training, validation and test set. The distribution usually follows 50:25:25 for large amounts of data, but if it is sparse, one would instead follow 60:20:20. During training, the model learns from the training set, while the validation set is used as a reference to see if the model's learning rate is reasonable. When the training process is done, the test set is used to derive performance metrics for the model to evaluate its performance.

Batch size

The batch size is the regulation metric that controls how much of the training set is processed in the model at a time. It regulates the number of samples the model considers before computing the loss function and updating the weights. So, letting in one sample would update the weights by each input. This might cause the model to be oblivious to patterns in the data and find a suboptimal solution—for example, a local minimum in gradient descent. By setting the batch size equal to the size of the training set, we would only have one update of weights, which seldom would be the best solution. Therefore, we generally choose a size in between.

Number of epochs

While the batch size decides how we divide the training set each time we pass it through the model, the number of epochs is the number of times we pass them through the model in total. Simply put, it is the recurrence of the training process. This allows the model to adapt to complex models iteratively. If the batch size is $1/Q$ of the data and epochs equal R, the weights will be updated $Q \times R$ times.

Learning Rate

A learning rate is an amount we ask our model to adjust for when getting an answer wrong. This means that a rate of 1, or close to, makes the model update its weight drastically often. Which could make the model learn fast but also risk it never stabilising. We do not want our model to over-correct, so usually, we choose a conservative learning rate closer to 0.1 or less. The choice depends on the model. Since we use a loss function as the measure to find the optimal solution, it is logical to approach it iteratively to minimise the loss. As the model is initialised with random weights, significant initial adjustments might lead the model awry. Still, setting a learning rate that is too small could be excessive and hamper the learning by taking too long.

Overfitting

A model overfits if it indicates good accuracy under training, but it only shows on the training data and not the validation data. It suggests that the model has learned specific features in the training set that do not occur in the validation set or other data in general. This failure to generalise makes the model limited. Smaller data sets are more at risk for this. To avoid this, we can limit the size of the neural network. Overfitting is often the case of the data adapting to the model when it is supposed to be the other way around. The data must be the primary influence for the development.

Parameter norm penalties

Regularisation is a possible way to avoid overfitting. Goodfellow et al. (2016) suggest parameter norm penalties as a strategy for this. This means we add a parameter norm penalty $\Omega(\mathbb{W})$ to the loss function, so that we now want to minimise

$$\tilde{C}(\boldsymbol{Y}, \boldsymbol{X}, \mathbb{W}) = C(\boldsymbol{Y}, f(\boldsymbol{X}, \mathbb{W})) + \lambda\Omega(\mathbb{W}),$$

where $\lambda \geq 0$ is a hyperparameter that controls the degree of regularization.

We continue with the explanations from Goodfellow et al. (2016). Usually, $\Omega(\mathbb{W})$ is the L^2-parameter norm penalty. This is known as ridge regression.

$$\Omega(\mathbb{W}) = \frac{1}{2}\|\mathbb{W}\|_2^2 = \frac{1}{2}\sum_{k=1}^{K-1}\left\|\boldsymbol{W}^{(k)}\right\|_2^2 = \frac{1}{2}\sum_{k=1}^{K-1}\sum_{i}\sum_{j}(w_{ij}^{(k)})^2$$

Adding the L^2-regularisation to the loss function give us

$$\tilde{C}(\boldsymbol{Y}, \boldsymbol{X}, \mathbb{W}) = C(\boldsymbol{Y}, f(\boldsymbol{X}, \mathbb{W})) + \frac{\lambda}{2}\sum_{k=1}^{K-1}\sum_{i}\sum_{j}(w_{ij}^{(k)})^2.$$

Taking the partial derivative with respect to each weight,

$$\frac{\partial \tilde{C}(\boldsymbol{Y}, \boldsymbol{X}, \mathbb{W})}{\partial w_{ij}^{(k)}} = \frac{\partial C(\boldsymbol{Y}, f(\boldsymbol{X}, \mathbb{W}))}{\partial w_{ij}^{(k)}} + \lambda w_{ij}^{(k)}. \tag{20}$$

Then, in terms of each weight matrix $\boldsymbol{W}^{(k)}$, we can write the derivative as

$$\nabla_{\boldsymbol{W}^{(k)}} \tilde{C} = \frac{\partial \tilde{C}(\boldsymbol{Y}, \boldsymbol{X}, \mathbb{W})}{\partial \boldsymbol{W}^{(k)}} = \nabla_{\boldsymbol{W}^{(k)}} C + \lambda \boldsymbol{W}^{(k)}. \tag{21}$$

The gradient update for each weight matrix is

$$\boldsymbol{W}^{*(k)} = \boldsymbol{W}^{(k)} - \alpha \left(\nabla_{\boldsymbol{W}^{(k)}} C + \lambda \boldsymbol{W}^{(k)} \right).$$

We can also write this as

$$\boldsymbol{W}^{*(k)} = \underbrace{\boldsymbol{W}^{(k)} - \alpha \nabla_{\boldsymbol{W}^{(k)}} C}_{\text{regular gradient update}} - \alpha \lambda \boldsymbol{W}^{(k)}.$$

Consequently, compared to a regular gradient update, the L^2-regularization will shrink the weights by an extra factor of $\alpha\lambda$ in every gradient update.

Lasso regression is the alternative parameter norm penalty, the L^1-regularisation. This is derived,

$$\begin{aligned} \Omega(\mathbb{W}) = \|\mathbb{W}\|_1 = \sum_{k=1}^{K-1} \left\| \boldsymbol{W}^{(k)} \right\|_1 &= \sum_{k=1}^{K-1} \sum_{i} \sum_{j} \mid w_{ij}^{(k)} \mid \\ \tilde{C}(\boldsymbol{Y}, \boldsymbol{X}, \mathbb{W}) = C(\boldsymbol{Y}, f(\boldsymbol{X}, \mathbb{W})) + \lambda &\sum_{k=1}^{K-1} \sum_{i} \sum_{j} \mid w_{ij}^{(k)} \mid . \end{aligned} \tag{22}$$

Taking the partial derivative with respect to each weight, this becomes

$$\frac{\partial \tilde{C}(\boldsymbol{Y}, \boldsymbol{X}, \mathbb{W})}{\partial w_{ij}^{(k)}} = \frac{\partial C(\boldsymbol{Y}, f(\boldsymbol{X}, \mathbb{W}))}{\partial w_{ij}^{(k)}} + \lambda \text{sgn}(w_{ij}^{(k)}), \tag{23}$$

where $\text{sgn}(w) = \frac{w}{|w|}$. In terms of each weight matrix $\boldsymbol{W}^{(k)}$, we can write

$$\nabla_{\boldsymbol{W}^{(k)}} \tilde{C} = \frac{\partial \tilde{C}(\boldsymbol{Y}, \boldsymbol{X}, \mathbb{W})}{\partial \boldsymbol{W}^{(k)}} = \nabla_{\boldsymbol{W}^{(k)}} C + \lambda \text{sgn}(\boldsymbol{W}^{(k)}). \tag{24}$$

Here we define $\text{sgn}(\boldsymbol{W}^{(k)})$ as the signum function applied to each element $w_{ij}^{(k)}$ in $\boldsymbol{W}^{(k)}$.

Thereby,

$$\boldsymbol{W}^{*(k)} = \boldsymbol{W}^{(k)} - \alpha \left(\nabla_{\boldsymbol{W}^{(k)}} C + \lambda \text{sgn}(\boldsymbol{W}^{(k)}) \right).$$

Alternatively,

$$\boldsymbol{W}^{*(k)} = \underbrace{\boldsymbol{W}^{(k)} - \alpha \nabla_{\boldsymbol{W}^{(k)}} C}_{\text{regular gradient update}} - \alpha\lambda \text{sgn}(\boldsymbol{W}^{(k)}).$$

We observe that the L^1-regularisation is sign-dependent, as we subtract the constant $\alpha\lambda$ for a positive weight and add for a negative weight. Also differing between L^1- and L^2-regularization is that L^1 move the weights to zero. This may cause sparser weight matrices. Combining both regression, we get an "elastic net", which was first proposed in Zou & Hastie (2005).

Early Stopping

We can usually spot overfitting during training. The loss function of the training data continues to decrease, while the one belonging to the validation data stagnates or starts to increase. By applying Early Stopping, we program in a condition that stops the training when the loss function no longer indicates betterment for several epochs. It then reverts to the last minimum. This does not only hinder overfitting but can also save computational time by stopping the training when sufficient. We have to decide how many epochs the loss function is allowed not to decrease before stopping. It is denoted as patience. Some epochs of patience could be needed, as fluctuation can be inevitable sometimes.

Variants of Gradient descent

We use the Adam optimiser in our experiments and will describe it here. The optimiser was developed by Kingma & Ba (2014). It combines Adagrad (Duchi et al. (2011)) and RMSProp (Tieleman et al. (2012)). These are both variants of gradient descent, like many other optimisers. The Adam optimiser is widely recognised and often chosen as the default. The algorithm uses the gradient's first and second moment estimates to update the weights. Exponential moving averages estimate the moments and then correct for bias.

We set

$$\nabla_k C_t = \nabla_{\boldsymbol{W}^{(k)}} C(\boldsymbol{Y}, f(\boldsymbol{X}, \boldsymbol{W})) \mid_{\boldsymbol{Y}=\boldsymbol{Y}_{(t)}, \boldsymbol{X}=\boldsymbol{X}_{(t)}, \boldsymbol{W}=\boldsymbol{W}_{(t-1)}} \tag{25}$$

where $\boldsymbol{Y}_{(t)}$ and $\boldsymbol{X}_{(t)}$ are matrices containing the vectors for the inputs and targets in batch t, and $\boldsymbol{W}_{(t-1)}$ is the updated weights from the previous batch.

The optimiser initialise the first and second moment estimator matrices as $\boldsymbol{M}_0^{(k)} = \boldsymbol{0}$ and $\boldsymbol{V}_0^{(k)} = \boldsymbol{0}$, for $k = 1, ..., K-1$. Also we let $\beta_1, \beta_2 \in [0, 1)$ and $\boldsymbol{\epsilon} = \epsilon \boldsymbol{1}$ where $\epsilon \in \mathbb{R}$. Kingma & Ba (2014) suggests $\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$ as reasonable default values.

For the t-th batch, the algorithm is derived as,

$$\begin{aligned}
\boldsymbol{M}_{(t)}^{(k)} &= \beta_1 \boldsymbol{M}_{(t-1)}^{(k)} + (1-\beta_1)\nabla_k C_t, \\
\boldsymbol{V}_{(t)}^{(k)} &= \beta_2 \boldsymbol{V}_{(t-1)}^{(k)} + (1-\beta_2)[\nabla_k C_t]^2, \\
\widehat{\boldsymbol{M}}_{(t)}^{(k)} &= \frac{1}{1-\beta_1^t}\boldsymbol{M}_{(t)}^{(k)} \\
\widehat{\boldsymbol{V}}_{(t)}^{(k)} &= \frac{1}{1-\beta_2^t}\boldsymbol{V}_{(t)}^{(k)} \\
\boldsymbol{W}*_{(t)}^{(k)} &= \boldsymbol{W}_{(t-1)}^{(k)} - \alpha \frac{\widehat{\boldsymbol{M}}_{(t)}^{(k)}}{\sqrt{\widehat{\boldsymbol{V}}_{(t)}^{(k)}} + \boldsymbol{\epsilon}}, \quad k = 1, ..., K-1.
\end{aligned} \tag{26}$$

After t batches, the bias corrected first moment estimator for the gradient is

$$\widehat{\boldsymbol{M}}_{(t)}^{(k)} = \frac{1-\beta_1}{1-\beta_1^t}\sum_{i=1}^{t}\beta_1^{t-i}\nabla_k C_i. \tag{27}$$

The expected value of this estimator follows

$$\mathbb{E}[\widehat{\boldsymbol{M}}_{(t)}^{(k)}] = \frac{1-\beta_1}{1-\beta_1^t}\sum_{i=1}^{t}\beta_1^{t-i}\mathbb{E}[\nabla_k C_i]. \tag{28}$$

If we assume that $\mathbb{E}[\nabla_k C_i] = \mathbb{E}[\nabla_k C_t]$, we get that

$$\mathbb{E}[\widehat{\boldsymbol{M}}_{(t)}^{(k)}] = \mathbb{E}[\nabla_k C_t]\frac{1-\beta_1}{1-\beta_1^t}\sum_{i=1}^{t}\beta_1^{t-i} = \mathbb{E}[\nabla_k C_t]. \tag{29}$$

As a result, there is an unbiased estimator for the first moment of the gradient at batch t. The same can be shown for the bias corrected second moment estimator $\widehat{\boldsymbol{V}}_{(t)}^{(k)}$.

Equation 27 shows the unbiased first moment estimator for the gradient at batch t, gets a more significant contribution from the last batch gradient than the first ones. Similarly, the unbiased second moment estimator for the gradient at batch t.

Feed-Forward Pass

To better understand the processing in a neural network, we follow the introduction of the feed-forward pass in Efron & Hastie (2016). We consider a network consisting of K layers, denoted as L_k for $k = 1, 2, ..., K$. The first layer gives the input vector $\mathbf{x} = \{x_j, j = 1, 2, ..., p\}$ and $w_l j$ denotes the weights for the variable j in the node l. $w_{lo}^{(1)}$

is the bias weight. Then the transition from first to secon layer

$$\begin{aligned} z_l^{(2)} &= w_{l0}^{(1)} + \sum_{j=1}^{p} w_{lj}^{(1)} x_j \quad \text{for } l = 1, ..., n_2, \\ a_l^{(2)} &= g^{(2)}(z_l^{(2)}) \qquad \text{for } l = 1, ..., n_2, \end{aligned} \tag{30}$$

$a_l^{(2)}$ are the values for the nodes in this layer.

This translates generally to the transition $k-1$ to k

$$\begin{aligned} z_l^{(k)} &= w_{l0}^{(k-1)} + \sum_{j=1}^{p_{k-1}} w_{lj}^{(k-1)} a_j^{(k-1)}, \quad \text{for } l = 1, ..., p_k, \\ a_l^{(k)} &= g^{(k)}(z_l^{(k)}). \end{aligned} \tag{31}$$

The These general equations are also true for $k = 2$ if we let $a_j^{(1)} = x_j$. This can also be presented in vector notation

$$\begin{aligned} z^{(k)} &= \boldsymbol{W}^{(k-1)} a^{(k-1)} \\ a^{(k)} &= g^{(k)}(z^{(k)}), \end{aligned} \tag{32}$$

where $\boldsymbol{W}^{(k-1)}$ represent the weight matrix, storing the weights from layer L_{k-1} to layer L_k, with the bias weight as well.

3.2 Recurrent Neural Networks

We build on our understanding of neural networks to introduce the Recurrent Neural Network (RNN). RNNs further develop the neural network provided to handle sequential data. Thus, opposing the originator that is meant for single, independent data points. The RNN process sequences and learns dependencies within them. By adding a concept of "memory" in the network, the model applies experience from previous inputs to its current output. The RNN considers the sequences samples as a whole and individual values, creating a context of input and applying and adjusting it for the following input sample. This makes the RNN processing time series and text and phrases. It has the added ability to recognise recurrent patterns.

We will introduce the process in an RNN relating to the general feed-forward pass presented earlier. To do so, we rely on Goodfellow et al. (2016). The feed-forward equation,

from the first layer to the second, for a RNN is

$$\begin{aligned} z^j_{h,}(2) &= u^{(1)}_h x_j + \sum_{i=1}^{H} w^{(1)}_{h,i} a^{(2)}_{i,j-1} \\ a^{(2)}_{h,j} &= g(z^{(j)}_{h,j}) \quad \text{for } j = 1, ..., p \text{ and } h = 1, ..., H. \end{aligned} \tag{33}$$

Compared to 30, there are differences. In 30, $z^{(2)}_{h,j}$ is not only dependent on node, h, but also the step within the sequential input, j for x_j. $a^{(2)}_{i,j-1}$ is the activation of node i at step $j-1$ and $b^{(1)}_h$ is the bias for the h-th node. $u^{(1)}_h$ is the weight connecting the input x_j and node h and $w^{(1)}_{h,i}$ is the weight connecting node i and h in steps $(j-1)$ and j. The weights remain the same for every step j of the sequence.

The addition of $u^{(1)}_h$, which is responsible for the input weights, influence the input directly. While, the set of weights from the input layer $w^{(1)}_{h,i}$ are influenced by the set of activation functions from the step before. The set of activation functions is often referred to as the current context and brings along the "memory".

This shows some of the innovations RNN does to a neural network. We have not discussed backpropagation, which is an important feature. It is the update of weights' influence according to the calculated gradient of the loss function. This is to regulate the change of weights in hindsight for calculated loss. RNN's property of "memory" is linked to backpropagation through time (BPTT), which make the update of gradients more extensive. This comes from the constant update of the current context. The extensive updates of the gradients may cause the vanishing gradient problem, which means calculated gradients may become so small that the weights do not get updated. This problem is solved in the LSTM model. For further reading on BPTT, we refer to Goodfellow et al. (2016).

3.3 Long Short-Term Memory

We introduce the Long Short-Term Memory model in this section, according to Brownlee (2017). Hochreiter & Schmidhuber (1997a) introduced an innovation on the RNN to counter the problem of vanishing gradients. This is what is known as the Long Short-Term Memory model. The hidden layers are replaced with what is known as memory cells. Furthermore, these blocks have an added self-loop, which let the gradient flow unchanged and not gradually vanish by updates.

The gates are also an essential feature of the memory cells. These are also weighted functions that regulate the information flow in the cell. A memory cell has three gates; Forget gate, Input gate and Output gate. The forget gate decides what information to discard from the cell. The input gate decides which values from the input to update the

memory state. Lastly, the output gate decides what to put out based on input and the memory of the cell. The memory cell includes an internal state as well. The internal state is built up through exposure to input over time steps and influences the output in each step. This figures as "memory" for the memory cell.

We examine the memory cells further by following Goodfellow et al. (2016). We consider a LSTM network. A forget gate unit for time step t in cell i is explained followingly

$$\boldsymbol{f}^{(t)} = \sigma(\boldsymbol{b}^{\boldsymbol{f}}\boldsymbol{u}^{\boldsymbol{f}}x^{(t)} + \boldsymbol{W}^{\boldsymbol{f}}\boldsymbol{a}^{(t-1)}). \tag{34}$$

The σ denotes the sigmoid activation function, which is works as the admission function. Value of 1 permits information flow, while value 0 prevents flow. It decides on the input of $x^{(t)}$ and $\boldsymbol{a}^{(t-1)}$, the output from the memory block at time step $(t-1)$. $\boldsymbol{b}^{\boldsymbol{f}}$ denotes the bias weights. While $\boldsymbol{u}^{\boldsymbol{f}}$ and $\boldsymbol{W}^{\boldsymbol{f}}$ are the input weights and the recurrent weights between time steps. The forget gate is element-wise multiplied with the internal state in the previous time step. The input gate is described as follows

$$\boldsymbol{i}^{(t)} = \sigma(\boldsymbol{b}^{\boldsymbol{i}}\boldsymbol{u}^{\boldsymbol{i}}x^{(t)} + \boldsymbol{W}^{\boldsymbol{i}}\boldsymbol{a}^{(t-1)}). \tag{35}$$

It regulates the amount of new information from $x^{(t)}$ and $\boldsymbol{a}^{(t)}$ that is added to the internal state at t. The input combines with the self-loop internal state to update the internal state. Self-loop is given as

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh(\boldsymbol{b}^{\boldsymbol{c}}\boldsymbol{u}^{\boldsymbol{c}}x^{(t)} + \boldsymbol{W}^{\boldsymbol{c}}\boldsymbol{a}^{(t-1)}). \tag{36}$$

The internal state is updated like,

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \odot \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \odot \tilde{\boldsymbol{c}}^{(t)}. \tag{37}$$

Now the internal state contains the information it decided to remember and new input in a weighted combination. This leads to following output gate

$$\boldsymbol{o}^{(t)} = \sigma(\boldsymbol{b}^{\boldsymbol{o}}\boldsymbol{u}^{\boldsymbol{o}}x^{(t)} + \boldsymbol{W}^{\boldsymbol{o}}\boldsymbol{a}^{(t-1)}). \tag{38}$$

Which is multiplied element-wise with the internal state of tanh,

$$\boldsymbol{a}^{(t)} = \boldsymbol{o}^{(t)} \odot \tanh(\boldsymbol{c}_t). \tag{39}$$

The memory cell's output is then $\boldsymbol{a}^{(t)}$. This output should then be passed through new memory cell for further training, or a basic node, for extraction of results.

After some experimental testing of different models, we concluded to go further with

LSTM. It intuitively fits our problem, as we ask to process sequences and wish to uncover dependencies in them. The capacity to learn sequence in total would be, and the memory component instils optimism for our solution.

4 Data Simulation and Preparation

To facilitate our experiment, we simulate a Spatio-temporal GARCH model. We will describe the process in this chapter and present the data we will proceed with. Furthermore, we will present the measures to prepare these data sets for neural networks.

Based on the theory from the 2, we will practically simulate from and estimate an ST-GARCH model, as we need sample data to run the experiments we want to do empirically.

The general procedure follows: We will first simulate data from an ST-GARCH model based on initial parameter values. Then estimate parameters based on an area of interest. Finally, simulate data from an ST-GARCH based on the estimated parameter. The procedure takes place in an R-environment, using the STARMAGARCH package, developed and presented by Sondre Hølleland in Hølleland (2020b).

Afterwards, we prepare the data for deep learning. We discuss what measures have to be made for it to be compatible with training a machine learning prediction model while also ensuring that the underlying model dynamics are not compromised.

4.1 Method of Simulation

To correctly achieve what we have set out to do, we follow the vignette presented on Hølleland (2020a). We start by creating a neighbourhood array for our spatial grid structure. It is defined by spatial dimension, neighbourhood type and circularity. We set the following values for our experiment: The spatial dimension is set to $(20, 20)$, which set the model to be defined two-dimensional over a $[20 \times 20]$ area. We choose the neighbour type to be "queen", so the model is defined with queen contiguity, and we define the model as non-circular.Next, we define the initial parameters for the model and set a temporal dimension. We create an ST-GARCH model from which we simulate data, setting time steps to 100, the output is of shape $[100 \times 20 \times 20]$.

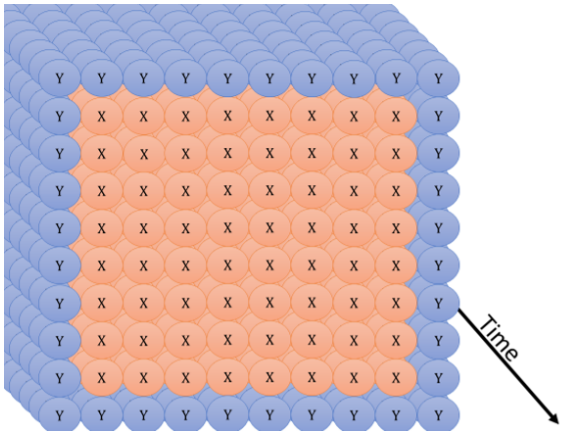


Figure 3: The three-dimensional perception of the boundary model.

Next, we update our parameters, where we set μ to be the calculated mean of our simulated data. We use the new parameters and our simulated data and neighbourhood array to create a circular likelihood function. Next, we withdraw a smaller

area within the simulated data, $[12 \times 12]$, in the spatial centre and withdraw between steps 20 to 80 to avoid an edge effect. We use this together with our newly created circular likelihood function and estimate an ST-GARCH model based on this. We pull out the parameter values from our estimation and simulate new data. Following this method with varying initial parameter values ensure we get data sets that are different but with the same properties. This variety is of importance when evaluating the models presented later.

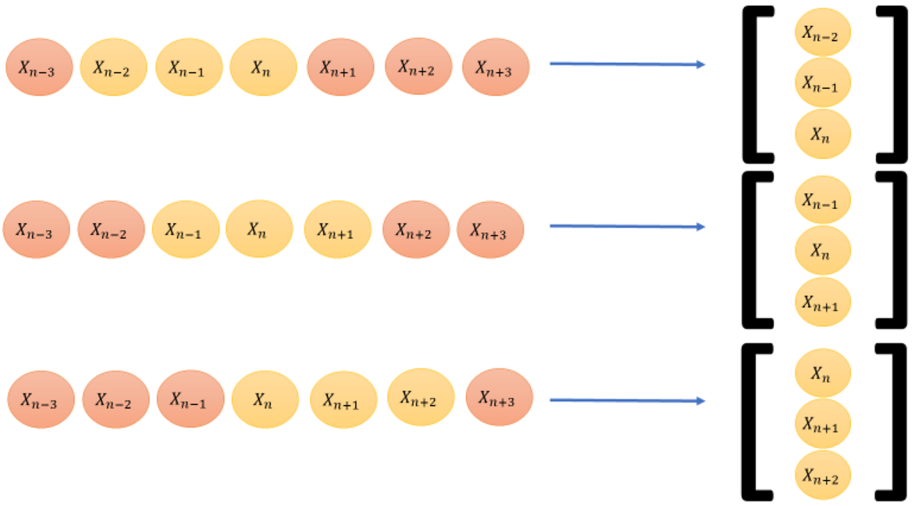
Remark 4.1.
In the process of simulating our data sets, we applied an estimation of a circular ST-GARCH to our simulated data before simulating new data based on the estimated parameters. This was originally part of an idea to consider if a prediction model trained on data from this "estimate"-simulation would approximate the original simulation. The idea did not come past this stage, but we kept the data sets for our current experiment. When extracting an area of interest from this simulation, we are provided with applicable data. The data shows the same properties as data simulated from the true parameters. We will not go further into this, but the likelihood-estimation applied is a good estimator for a circular stationary ST-GARCH. For further reading, we refer to Hølleland & Karlsen (2020b).

Data Sets
Two different procedures have been used to create data sets for training our prediction model. We want a large amount of data, as this gives us the freedom to explore different aspects of making a prediction model. For instance, we want to adjust the size of our training data for the model. The size may affect the model to overly generalise or not generalise sufficiently. An extensive simulation also ensure enough data to be used to test and evaluate the model.

However, to simulate spatiotemporal models with belonging neighbourhood arrays is computational demanding. We have limited ourselves to data of $[20 \times 20]$-spatial dimensions, as it suffices for our use. Accordingly, we proceed with three different data sets with varying temporal dimensions.

Data set 1: *Size:* $[10000 \times 20 \times 20]$. *Simulated over 100 processes of 100 time steps, based on the same estimation.*
Data set 2: *Size:* $[100000 \times 20 \times 20]$. *Simulated in one process.*
Data set 3: *Size:* $[500 \times 20 \times 20]$. *Simulated in one process.*

Figure 4: Illustration of the sequencing of observed data.



4.2 Data Preparation

We withdraw a spatial area of size $[14 \times 14]$ from the data sets. This way, we avoid the simulation's edge effect and leave room for experimentation. We denote the inner $[12 \times 12]$-area as our area of interest, where the edge is the inner boundary X, and we denote the circumference as the outer boundary Y, our prediction targets. Figure 3 gives a figurative impression of this. We want four pairs of sequences per timestep, each representing the inner-outer boundary relationship for each side. So, the inner boundary is divided into four sequences of 12, where each corner occurs on both sides. We do the same to the neighbouring points on the outer boundary, leaving out its corners. Subsequently, the sequences are sorted according to time steps in a data frame and exported from the R-environment.

In the Python environment, we reconstruct the spatial dimension of the data. Then, we transform the sequences of Xs into two-dimensional arrays, where each row is an iterative sequence of the three Xs that are considered the nearest neighbour of each Y. The sequencing is illustrated in figure 4. We do this to facilitate features for the prediction models. Following this setup of relations, each inner boundary is then left to ten iterative sequences of three Xs. At the same time, the outer boundaries are compromised in both ends by one Y because of the lack of paired sequence near the corners. Thus, we satisfy the relational pairing we want to base our prediction model on, which is shown in figure 5.

Now we have feature data X and target data Y in three-dimensional and two-dimensional arrays respectively; $[n \times 10 \times 3]$ and $[n \times 10]$. n is the amount of time step dependent on the data set. Both are synchronically sorted according to temporal and spatial dimensions. Figure 6 show the organising of the three-dimensional array of feature data.

We propose an alternative way to organise the data as well. We reduce the data by one dimension by combining the spatial and the temporal dimensions. This way, we let the spatial boundary points occur sorted according to time steps, seeing periodical patterns of intervals of ten appear. Thus, we retain arrays for X and Y that are more manageable, in the shape of $[n \cdot 10 \times 3]$ and $[n10 \times 3]$ respectively.

Before we start developing prediction models, we need to scale our data to get a better effect of the model fitting and avoid sign confusion in the models. We use Scikit-learn's Pedregosa et al. (2011) *MinMaxScaler*, which scales the values to the range of zero to one. Thus the properties of the data are maintained without sign confusion.

Now we presume the data sets as ready for the modelling process.

4.3 Computational Challenges

Some assessments have been made in the simulation process based on restrictions linked to computational processes. The STARMAGARCH package used in R exports calculation tasks to a different programming language, C++, to more efficiently do them. We had trouble with the memory allocation in our C++ compilator in our computational setup. This hindered us from doing more extensive simulations.

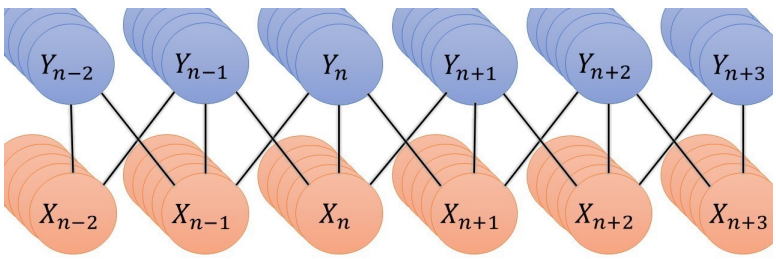


Figure 5: Relationship between inner, X, and outer, Y, boundary in Spatial point-based.

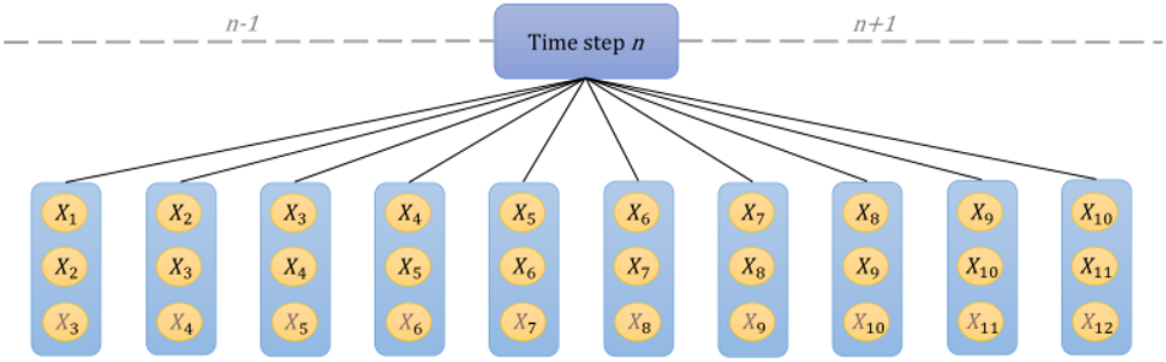


Figure 6: Shape of three-dimensional feature data per time step.

5 Prediction Model

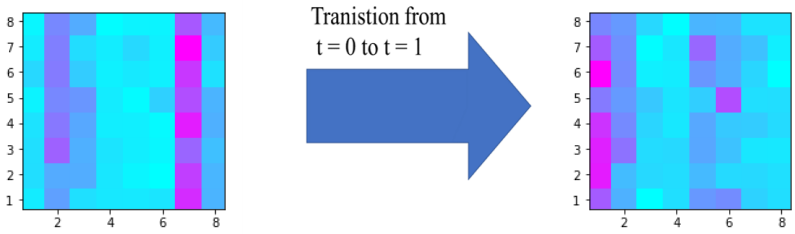
Through this chapter we will give a figurative explanation of the prediction models, we will present the challenges and solutions related to the structure of our data, and we will discuss the technical optimization of the network.

We want our prediction model to be versatile and robust, thus making it applicable anywhere in the model, to predict spatially. But, specifically, is to predict an outer boundary based on an observed boundary, for only one step in time. In this case the data amount is sparse, especially for models of smaller spatial areas, and our prediction model should be able to show consistency in performance, even in several different smaller boundaries. In this case, spatial dependency is what is mainly considered

5.1 Figurative Explanation

The environment in which we intend to operate our prediction model is one of three dimensions: two spatial dimensions and the dimension of time. A known comparable visual understanding of this is a weather map showing temperature variability. Figure 7 is comparable to this. It shows an 8×8-area of pixels, which in our case will fit well, while a normal weather map would be more detailed. We could assume this to show the transition day-by-day of temperature distributed spatially within this 8×8-

Figure 7: Example of change per time step in a 8×8 Spatio Temporal Model.



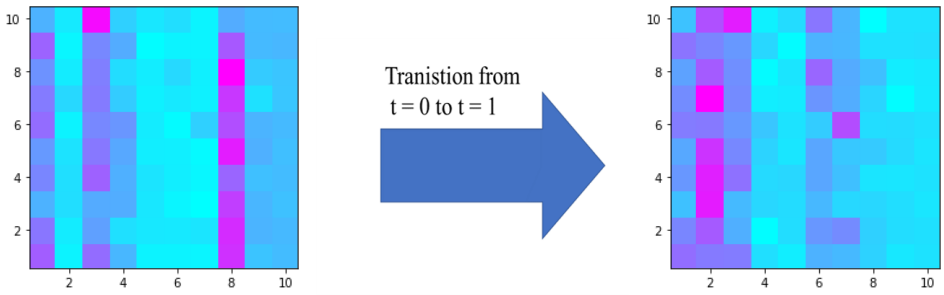
area of interest in contexts of weather maps. We assume the shades of pink to be high temperatures, while shades of blue are low and purple is the intermediate temperature. We assume dependence between the neighbouring pixels within this defined area of pixels. In forecasting weather, this would translate to a correlation between the temperature in a city and the temperatures in the surrounding areas. Likewise, we assume dependence between the shade of a pixel in figure 7 at $t = 0$ and $t = 1$. Relating to the same city would translate to the correlation between today's and yesterday's temperature.

This works as the basic environment for our model to adapt to and predict within. Furthermore, we would like the model to predict, still following our comparison, the temperature for the neighbouring areas to the boundaries of our weather map. The result would be an expansion on figure 7 to figure 8. Making the 8×8-area expand to 10×10-by adding a new predicted boundary, called the outer boundary. So, we shall construct a method that predicts the values on such boundary based on the information in the 8×8-area.

To do so, we approach this as a regression problem, where the value on the predicted boundary, denoted as Y_n where n is the spatial placement, is dependent on its three spatially nearest neighbours on the observed boundary, denoted as X_{n-1}, X_n and X_{n+1}. This relationship is illustrated in figure 5. Setting it up like this leaves us relatively few features to our model, but assuming the spatial dependencies of neighbouring points in our data, we know each feature has a significant relation to the target value and in between themselves.

This setup is also crucial for the prediction model to apply to smaller areas of interest. The demand of three observations per prediction limits the prediction model to provide two fewer values than the provided input. Considering the 8×8-area mentioned above, and denoting the values on the upper side boundary by $X_1, X_2, ..., X_8$, our prediction model would only provide an outer boundary of six predicted values. This is because of the ratio of three features to one prediction. See figure 9 for an illustrative presentation of this type of relation. This "iterative recurrent" sequencing, also shown in figure 4,

Figure 8: Example in figure 7 expanded in all spatial directions to a 10×10 spatial model.



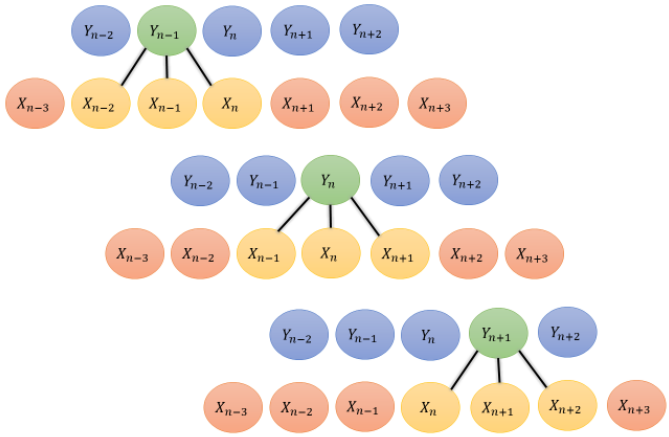


Figure 9: Illustration of the Sequencing for our prediction model.

do ensure that we get as much information from each X we can. However, we see that on the ends of the upper boundary of eight observations from our area of interest, we do not have the data to predict an outer boundary point parallel to X_1. This Y_1 would need to be provided with the nonexistent X_0.

We recognise this as a corner problem and propose a solution to this later. However, first, we ensure to limit the scope of the problem. The consequence of increasing the number of features in the ratio between features and predictions is less predicted data. Where the 3:1-ratio provides six predicted values from eight, the 5:1-ratio only provides four

predicted values, which is insufficient, as over half of the intended outer boundary would have to be provided in other ways. Granted, this would be tolerable for more extensive spatial areas of interest as the data loss would not be as significant.

There is an argument for using second order-neighbouring points a step further from the boundary and including them in the model to add features. This could offer more data per prediction and provide a more "well-informed" prediction model. We opt out of this setup as the degree of spatial dependency (second-order neighbouring) would have to be communicated to the model, which would cause challenges in the formatting of the data. This would demand a more complex model, which may be more rigid when applied. There should also be concerns about whether the significance of the first degree spatial relations would also run the risk of being levelled out.

As touched upon earlier, our method for prediction poses a problem when approaching the corners of our outer boundary. We decided on a prediction model which minimises this problem, yet we need to propose a solution.

The dynamics of the corner problem is illustrated in figure 10.

The $X's$ are the observed values, Y_L and Y_U denotes the predicted boundary values on respectively Left and Upper side.

We calculate the mean of the predicted values on the boundaries and add this value on both ends for the mentioned extension of our predicted boundary. This is denoted in figure 10 as $\bar{Y_L}$ and $\bar{Y_U}$. For the extension to also include the corners of this boundary we calculate the combined mean of the intersecting sides, $(\bar{Y_L} + \bar{Y_U})/2 = \bar{Y_{L,U}}$, giving it its notation as $\bar{Y_{L,U}}$ in figure 10.

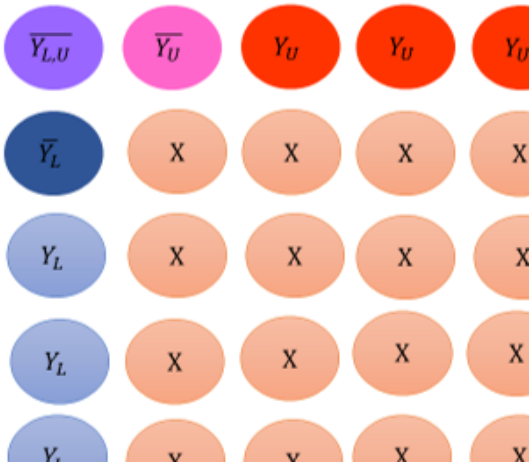
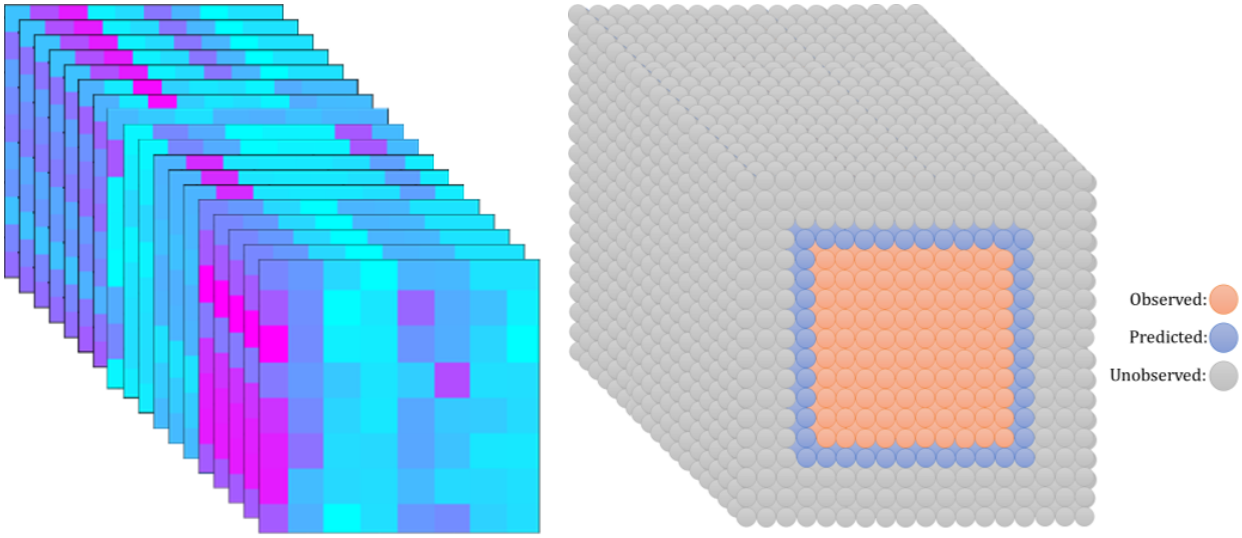


Figure 10: Corner dynamic of predicted boundary, with example Upper-Left corner.

This practical approach to this problem will complement the prediction model presented in this thesis. There are other ways to address this issue. One could apply the same method as above on the observed boundary, X, to supply the prediction model with synthetic data and then achieve a coherent circumferential boundary. Moreover, one could predict the boundary as a whole, overlapping the corners. This would offer predicted values for what is denoted as $\bar{Y_L}$ and $\bar{Y_U}$ in figure 10, but omit $\bar{Y_{L,U}}$. In addition, the spatial dependencies between X's and Y's will not be considered for whether

We started this chapter by presenting the weather map comparison for our data, where the map changed over the transition from $t = 0$ to $t = 1$. When considering this applied in a weather report, we can envision an animation where each frame represents a time step and the gradual temperature change within the area. Illustration of this type of animation decomposed, frame-by-frame, is seen in figure 11 (a). Even though changes in colours signify a change in values, we know the structure, or grid, to remain the same. Figure 11 (b) offer a point-based understanding of this structure. There is an assumption for the unobserved to adhere to the same structure. This makes it predictable to operate in, and thus suitable for fitting deep learning model.

For our prediction model to perform well, we expose it to the spatial relations illustrated in figure 5 extensively. Therefore, we feed the model samples per time step. To make the model learn both temporal and spatial dependencies, we strictly keep the temporal order in the input.



(a) Frame-by-frame representation of gradual change in area of interest

(b) Spatiotemporal point-based Representation

Figure 11: Different representations of Spatiotemporal data.

5.2 Technical Execution

As concurred in chapter 3, we have decided to model our prediction model as a Long Short-Term Memory Network. For this purpose, we utilise the open-source neural network library Keras Chollet et al. (2015). This library functions as an application programming interface for the Tensorflow library Abadi et al. (2015) through the programming language Python. The accessibility Keras offers to powerful machine learning tools and its convenience in use has proven to be favourable for us in the comprehensive testing of algorithms and later when experimenting with several adjustments for optimization. Other important software being used in relation to the programs we have run are Pandas pandas development team (2020), NumPy Harris et al. (2020) and Scikit-learn Pedregosa et al. (2011).

LSTM networks are often utilised for classification tasks, for example Text classification Zhou et al. (2015) and Time Series Classification Karim et al. (2017), whereas our task requires a regression-based solution. The premise of applying the LSTM algorithm for our problem was influenced by Brownlee (2018). It serves as a useful resource in the intersection of the Keras library and Time Series. Accounting for the LSTM algorithm aptness for identifying long term correlations in sequences, Brownlee introduces several experiments involving variants of Deep Learning algorithms through Keras. We expand on these onto spatiotemporal data, and as a result, construct a prediction model for spatial and temporal dependencies.

We have searched for an optimal model by experimenting with the different data. Accordingly, we have arrived at two solutions for feeding data to the prediction model. As

discussed in 4, we have prepared the data in two formats. One format is true to the original structure, where the X-sequences are stored in $[10 \times 3]$-arrays, and similarly, the Y's are stored in $[10 \times 1]$-arrays. Thus, the data is distributed according to time steps n in samples of aforementioned arrays, structured to input X: $[n \times 10 \times 3]$ and output Y: $[n \times 10 \times 1]$. In this way, the network will process each boundary in assembled form and provide predictions per sample of the form $[10 \times 1]$.

The second format is purposefully "flattened" to ensure more manageable data. Rather than assembling arrays per time step, we store the $[10 \times 3]$- and $[10 \times 1]$-arrays in, respectively, $[(n \cdot 10) \times 3]$- and $[(n \cdot 10) \times 1]$-arrays, in temporal order. Thus, each sample is the individual pairing of X-sequence and Y, rather than complete boundaries. This way, the boundaries appear as a period, making each spatial point periodic within the array. Comparatively, this format is more manageable than the three-dimensional and allows the model to be more flexible. However, there should be concerns about whether the properties of the underlying structure are compromised and whether the potential lack of information in the data will hurt the prediction model's performance.

We have adapted our LSTM networks to handle both data formats and subsequently experimented with variations on both models. We will present the variations that consistently performed the best from here on.

5.2.1 Model setup

We declare the standard features of our model variations first. The common optimisation algorithm for all variations is "Adam", which is presented in 3. Our initial trials have shown "Adam" outperformed other optimisers suitable for regression models. The optimiser "RMSprop" came closest in performance to "Adam", but seeing that "Adam" had the consistent edge, we chose it going forward. This comes as no surprise as "Adam" builds on "RMSprop" and is a very popular optimiser.

Similarly, Mean Squared Error operates as the standard loss function. The potentially suitable loss functions for a regression model are limited to Mean Absolute Error, Mean Squared Error, and Mean Squared Logarithmic Error. We could early count out the MSLE because we work with scaled data, and hence there is a lack of logarithmic effect. MAE and MSE have more similar performance results, although MSE shows itself to be more consistent. The MSE does punish more significant errors than MAE, which is advantageous for our data. We know our data is volatile, which means Mean Directional Accuracy is an important measure. Mean directional accuracy measures the proportion of signs (negative or positive change) between predicted sequences that are true to the actual data. Thus we choose a loss function that inflicts harder punishments on more significant errors, as it will discourage predictions that moves in the opposite direction

from the actual data.

Otherwise, all variants have at least one LSTM-layer and one Dense(1)-layer, often considered by Vanilla LSTM. The Dense(1)-layer acts as the output layer of the model and, by default, is assigned with a 'linear' activation function. The 'linear' activation function is preferred for regression as our intended predictions are unbounded quantities. For the output layer of the three-dimensional models, we add the "wrapper" TimeDistributed to the Dense(1)-layer. As the samples consist of $[10 \times 3]$-arrays and are trained to predict $[10 \times 1]$-arrays, the "wrapper" allows for the predictive outputs to be produced as single values. This means the output is processed one by one as steps themselves, although the input is processed as $[10 \times 3]$ per sample, and the model learns accordingly. As a result, we ensure that all points are individually but still in the circumstances of a sequence of ten.

Thus, we have a base model. The variants we have gone forward with are different regarding depth and width and the amount of data used in training. We set the standard to be shallow models after experiencing good results, but we experiment with two "stacked LSTM "networks, inspecting the effect of stacking. We also experiment with the width of our model. Our standard LSTM-layers are composed of a modest ten memory cells. We also run a narrower model with three memory cells and a broader one with 1000 memory cells, thereby seeing its effect on our problem. Lastly, we experiment with the amount of data used in training. Recognizing that the data is sampled from a spatiotemporal structure, we must consider that some data features may become less prominent in larger sets of data. If these features are of interest for our model to learn, we might be better off constricting the amount of data.

Next, we list the models with their specifications:

Model I: *1 LSTM-layer of 10 units. Trained on 5000 data entries.*
Model II: *1 LSTM-layer of 3 units. Trained on 5000 data entries.*
Model III: *1 LSTM-layer of 1000 units. Trained on 5000 data entries.*
Model IV: *1 LSTM-layer of 10 units. Trained on 500 data entries.*
Model V: *3 LSTM-layers of 10 units. Trained on 500 data entries.*
Model VI: *6 LSTM-layers of 10 units. Trained on 500 data entries.*

Hence, we proceed further with these models.

The number of data entries mentioned above translates to total "row"-input for each model, not the number of samples. So, in the case of three-dimensional input, 5000 data entries are distributed over 500 samples and 500 data entries over 50 samples. While in the "flattened" case, 5000 data entries equal the same amount of samples, likewise for 500.

5.2.2 Training

The procedures of training are similar for all models. The validation sets per model are adjusted for an 80:20-ratio concerning the training set of the model. After some trials, we conclude on a basis for hyperparametric values. We find a batch size of 10 for the two-dimensional model and one for the three-dimensional one shows consistent results. Following, a number of epochs equivalent to 50 are effective for both.

Setting the principal batch size to 10 is generally relatively small. We push the model to consider the data in sequences of 10, the spatial size of the boundaries we intend to predict. This way, the weights in the network will only be updated as ten data samples are shown to the network. Subsequently, this leads to a prediction model primed for predicting ten values at the time. The hypothesis is that this will emphasise the spatial and temporal dependencies within the data, and the model will learn to estimate the underlying model. We wish to have the same effect for the three-dimensional model and choose a batch size of one. Then the weights are updated for each sample. This is especially necessary for the models based on 500 data entries. A batch size of 10 would only let the model update its weights five times per epoch, which is less than we would like.

Usually, larger batch sizes will be preferred in machine learning problems because problems with many defined features will need more data to sort out the defining ones. Our model is based on three features; three X's to one Y, that also is recurring. So, it is important to have the model learn the "hidden features" of spatial dependencies within each boundary sequence and then learn temporal dependencies and generalise over time. The risk in letting the model process larger quantities in each batch, for example 100 equivalent ten boundary sequences, is that it will potentially disregard the order of the data in favour of extreme values, stretching across boundary sequences. Consequently, this would be unfavourable for predictions of 10. Seeing we are more interested in the volatility within and in between each boundary sequence than the larger volatility pattern of the data samples concatenated.

The choice of the number of epochs follows the decision of batch size. Models with smaller batch sizes tend to converge faster than models with large batch sizes. This is a consequence of a smaller batch size letting the model update the weights earlier, while the other model must process more data before updating. Therefore, given our determined batch size, we can assume that 50 epochs will suffice for our loss function to converge.

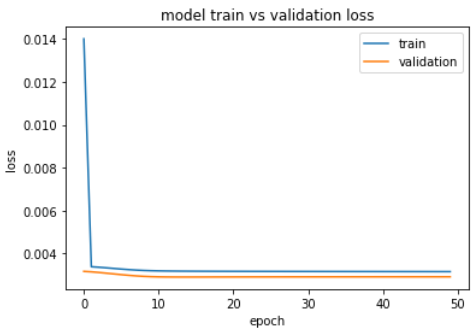


Figure 12: The learning from training of 2D Model II on data set 1.

We train our models on these assumptions and observe the learning rates. From what we observe, most models show convergence in learning, but some deviations appear of the total 36 models. In figure 12, we observe the learning rate of the two-dimensional model II when trained on data set 1. The curve is immediately very steep, flattens by epoch number two, and is horizontal from epoch number ten. This indicates that the model does most of its learning in epoch one and does not learn from epoch ten. The model was expected to learn fast, as it only has one LSTM-layer of 3 units, but this curve suggests that the model has concluded on a suboptimal solution. We consider the consequences of this in 6.

6 Results

In this section, we present the results following the evaluation of our finalised models. We consider several performance measures to examine our different approaches to the problem.

The initial impression is that our experiment has not led to a well-performing prediction model. According to the performance measures, we can achieve similar or better results by calculating the mean of the target values from the training data set. However, the trained model shows it outperforms a mean calculated of the observed values in the test set. Thus, the model has learned something about the spatial relationship between the inner and outer boundary but has not been able to predict the outer boundary values accurately. We take a closer look at the performance measures.

RMSE and MAE

We must evaluate our models using the root mean squared errors (RMSE) as we chose to train them using the mean squared error (MSE) as the loss function. We also use the mean absolute error (MAE) to supply. These performance metrics are defined as follows:

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N}}$$

$$\text{MAE} = \frac{\sum_{i=1}^{N} \mid y_i - \hat{y}_i \mid}{N}$$

where $\hat{y}_i$ denotes the predicted values and y_i denotes the observed values.

Given these definitions, there is no surprise that the relational result between different models and data sets seems to be the same on both metrics. The table, 1, illustrate the results given by the metrics, aided with gradual color coding. The values marked in green are the lowest errors, while red signifies the opposite. We observe that the mean of the target values, Y, consistently performs on par with the best models.This RMSE-value is approximate to the standard deviation of the observed data, as their means are almost indistinguishable. This works as our benchmark for metrics of error. Similarly, the mean of the features used for predictions, X, consistently performs worse.

The performance metrics suggest some results of comparing the models. The models with more layers performed best for the data set simulated recursively. These were also

		RMSE:			MAE:		
Data:		**1**	**2**	**3**	**1**	**2**	**3**
Model I	**2D:**	0.05884	0.05229	0.12192	0.04298	0.03976	0.09529
	3D:	0.05899	0.05916	0.13463	0.04314	0.04496	0.10501
Model II	**2D:**	0.05888	0.05229	0.12187	0.04302	0.03975	0.09528
	3D:	0.05904	0.05378	0.12409	0.04317	0.04108	0.09747
Model III	**2D:**	0.0605	0.05234	0.12187	0.04484	0.03978	0.09527
	3D:	0.05887	0.05279	0.12269	0.043	0.04025	0.09599
Model IV	**2D:**	0.05957	0.05477	0.12238	0.04347	0.04184	0.09585
	3D:	0.0599	0.05354	0.12275	0.04366	0.0407	0.09626
Model V	**2D:**	0.05841	0.0532	0.12182	0.04248	0.04052	0.09523
	3D:	0.05862	0.05289	0.12225	0.04271	0.04019	0.09563
Model VI	**2D:**	0.05822	0.05256	0.12256	0.04227	0.03995	0.09598
	3D:	0.05864	0.0527	0.12249	0.04274	0.04001	0.09581
Mean(X(test))		0.096963	0.064364	0.132399	0.083244	0.051135	0.104323
Mean(Y(train))		0.058226	0.052308	0.121781	0.042266	0.039792	0.095173

Table 1

the models trained on fewer data. The result is reasonable as these models have learned with more recurrence than the others, over fewer data points. And practically, only been informed by data from a single simulation. In this case, spending more time with fewer data proved better. This may suggest that the models trained on more data from this data set, and thereby from several simulations, have generalised the predictions according to the larger amount. Even though the learning rate did not suggest it in the training process, the more extensive data seems to have caused the model to overfit. We did opt out of the models of several layers trained on more extensive data in the training process for this reason. Seemingly, more attentive use of data set 1 in the modelling process would yield better results.

We consider the result from data set 2 and 3 together. Comparatively, their results show similar relations between the models. Both data sets consist of data from a single simulation, although the simulation behind data set 2 extend over more time steps.

We observe that the models of only one layer perform best for these data sets when fed with two-dimensional data. And without much variation in performance based on the number of units. An increase in units should affect the learning capabilities of the model. The model should learn the implicit relationships more strictly, with the risk of overfitting. The reading of our results suggests that there are marginal differences in learning given the number of units. This indicates that the model learns a limited amount of information based on its input. Compared to the same models adapted for and fed with three-dimensional data, the performances are better. This suggests that the two-dimensional models make the right decisions but cannot optimise on its available information. Neither of the models with several layers performs better than the models

		MDA:		
Data:		**1**	**2**	**3**
Model I	**2D:**	0.73836316	0.74766	0.74789
	3D:	0.70994199	0.71814	0.71371
Model II	**2D:**	0.73796119	0.74831	0.74838
	3D:	0.72474495	0.73855	0.73674
Model III	**2D:**	0.72048817	0.74792	0.74956
	3D:	0.72134427	0.74795	0.74775
Model IV	**2D:**	0.73203771	0.73339	0.74432
	3D:	0.73534707	0.73875	0.73473
Model V	**2D:**	0.7440323	0.74225	0.74819
	3D:	0.74174835	0.74495	0.74575
Model VI	**2D:**	0.74933074	0.74736	0.74152
	3D:	0.74454891	0.74855	0.74074
Mean(X(train))		0.570714	0.665333	0.718719
Mean(Y(test))		0.74855	0.74855	0.745746

Table 2

		Variance:		
Data:		**1**	**2**	**3**
Model I	**2D:**	0.00000168	0.00000143	0.00001031
	3D:	0.00000505	0.0007853	0.00314724
Model II	**2D:**	0.00000169	0.00000133	0.00000511
	3D:	0.00000638	0.00017099	0.00049381
Model III	**2D:**	0.00000029	0.00000076	0.0000015
	3D:	0.0000048	0.00001995	0.00022331
Model IV	**2D:**	0.00013583	0.00020452	0.00004784
	3D:	0.00017841	0.00010646	0.00017466
Model V	**2D:**	0.00001612	0.00003078	0.00000802
	3D:	0.0000382	0.00003888	0.00003649
Model VI	**2D:**	0.00000027	0.00000092	0.0000477
	3D:	0.00002747	0.00002216	0.00000011
X(train)		0.00211332	0.00181145	0.0071064
Y(test)		0.00338674	0.002733374	0.01482867

Table 3

of one layer, but they do better than model IV. This suggests that more layers will improve the model when trained on fewer samples. We can back up this assumption by looking at the results from data set 1, as it seemed to show similarly.

We have considered the results of the two-dimensional models, as they performed best, but we will compare them with the three-dimensional model. The three-dimensional approach performs consistently worse than most of our two-dimensional models when we look at the tables. And looking at the figures, we see why. For example figure 13, where we can see model I on each boundary of our area of interest for both data set 2 and 3. We can see an apparent periodicity in the graph, where the model predicts a low value every tenth prediction, the start of each boundary, regardless of the value of the observation. Although the predictions are inaccurate, which shows in the performance values, they do far better in predicting the degree of volatility of the observed value. However, combining the ordered nature of the data and the metrics, RMSE and MAE, a prediction model is penalised less by predicting conservatively along with the mean. The penalty of a prediction on the opposite side of the mean from the observation is too significant for the model to not learn from it, especially when using MSE as its loss function.

MDA and Variance

We want to consider further the issue of accuracy and volatility. Therefore we consider the Mean Directional Accuracy (MDA) and the variance of the predictions, listed in table 2 and 3. MDA is a measure of prediction accuracy forecasting often used in economics and finance. It compares the direction of the value between steps in the observed series and in the predicted series, for example whether stock prices falls or rises. Mathematically its explained as follows:

$$\text{MDA} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}_{\text{sign}(y_i - y_{i-1}) == \text{sign}(\hat{y}_i - y_{i-1})}$$

where $\hat{y}_i$ denotes the predicted values and y_i denotes the observed values.

Table 2 shows that the ranking between the models are similar by this measure as

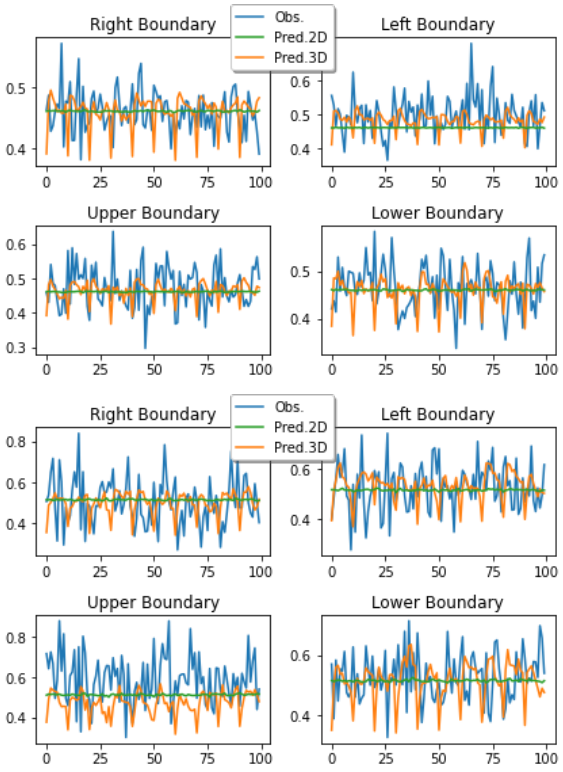


Figure 13: Model I, data sets 2 and 3.

		MAPE:		
Data:		**1**	**2**	**3**
Model I	**2D:**	9.55263	8.6061	18.63762
	3D:	9.99696	9.73158	20.53864
Model II	**2D:**	9.56222	8.60398	18.63425
	3D:	10.0029	8.89179	19.06441
Model III	**2D:**	10.22758	8.60862	18.63247
	3D:	9.98539	8.71104	18.77309
Model IV	**2D:**	9.66117	9.05615	18.7464
	3D:	10.20238	8.8076	18.8273
Model V	**2D:**	9.44248	8.77023	18.62458
	3D:	10.01253	8.69853	18.70414
Model VI	**2D:**	9.39492	8.64606	18.77262
	3D:	9.99501	8.65845	18.73863
Mean(X(test))		20.454131	11.066858	20.403768
Mean(Y(train))		9.974546	8.612085	18.614194

Table 4

well. MDA measures the prediction models' capability to stay on the right side of the observed model. Given the ST-GARCH models autoregressive properties, it is logically sound that the predictions along the observed model's mean contribute to a good MDA score. By that premise, the non-volatile prediction models are rewarded. Table 3 shows the variance of the predicted values, and here we see more apparent distinctions between the models. The models that performed best on RMSE, MAE and MDA has a minimal variance, while the opposite is true for those performing worse on those metrics. This substantiates the assumption that the models have penalised the models out of volatile predictions.

MAPE

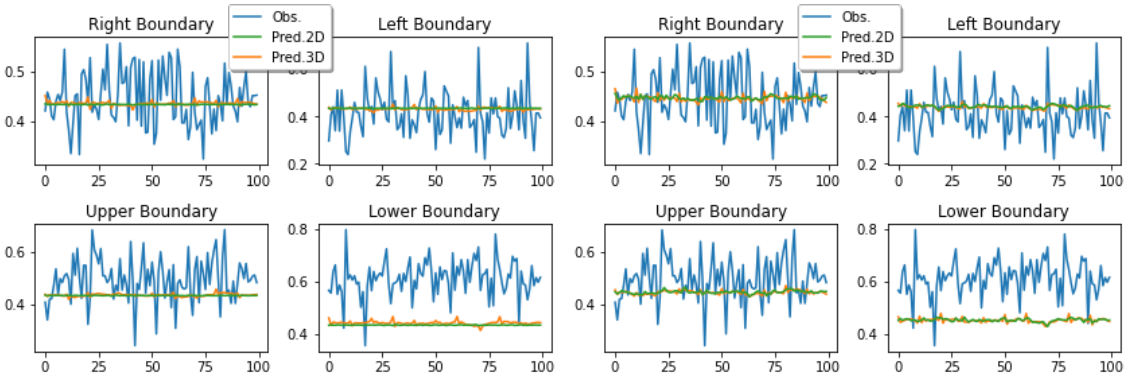
We will also consider the Mean Absolute Percentage Error (MAPE) as a final performance measure. MAPE is calculated followingly:

$$\text{MAPE} = \frac{100\%}{N}\sum_{i=1}^{N} \mid \frac{y_i - \hat{y}_i}{y_i} \mid$$

where $\hat{y}_i$ denotes the predicted values and y_i denotes the observed values.

MAPE give us an intuitive interpretation of the relative error of our prediction models. Table 4 reveals the values and adds to the relational assessment of the models. It states a better than 90% accuracy for several models trained on data set 1 and 2, while those trained on data set 3 show approximately 80% accuracy. If we consider the figures 14, 15 and 16, which shows model III on the different data sets, we observe that the observed and predicted values are generally higher. This raises the issue with MAPE as

a performance metric. MAPE penalises over-forecasting harder than under-forecasting. So, MAPE favours the models fitted on data with a lower mean.

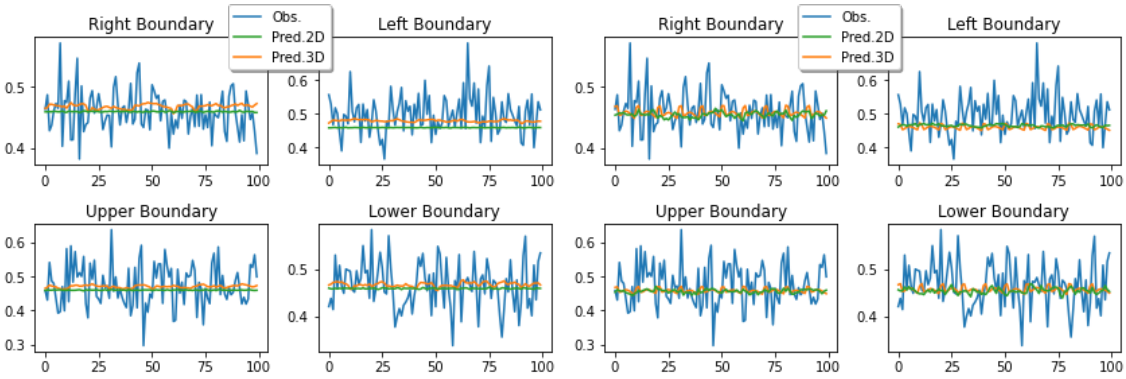


(a) Model III. (b) Model V.

Figure 14: Model III and V on Data set 1.

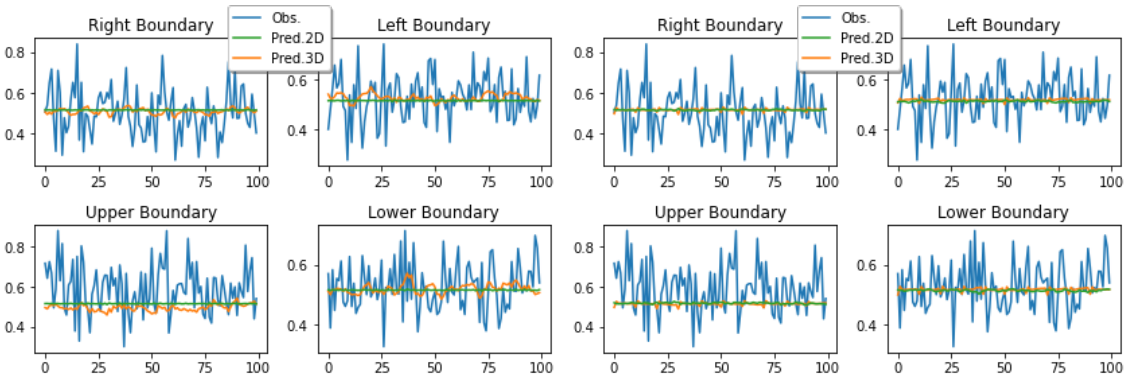
Versatility

We test the prediction models for versatility by applying them to all boundaries of the area of interest. This indicates whether the spatial relationship learned by the prediction models in training is applicable regardless of the relative positioning between the inner and outer boundary. The figures in this chapter show the prediction models' efforts on all sides. Most prediction models seem to adapt well, but several plots indicate biased models. For example the lower boundary of model III and model V on data set 1, in figure 14. This plot indicates an inability in the models to stray away from a mean value it has calculated during training. Thus, it suggests that the prediction models lack the versatility and spatial awareness we want.



(a) Model III. (b) Model V.

Figure 15: Model III and V on Data set 2.



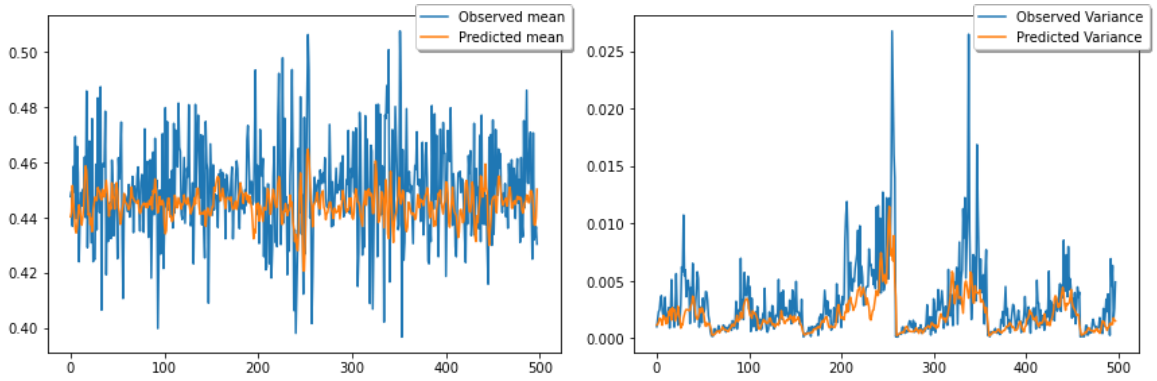
(a) Model III. (b) Model V.

Figure 16: Model III and V on Data set 3.

7 Discussion

We discuss our experimental shortcomings in hindsight. The results of the models did not meet initial expectations, and the values of RMSE have stagnated around the standard deviation of our test data. Different models and adjustments we have tried have not made a positive impact, and thus, we review our experiment process.

The first proposal to an alternative approach is handling the preprocessing of the data another way. We have done trials with non-scaled and squared data without better effect than our approach. Trials with standardised data ($z = ((x - \mu)/\sigma)$) did not prove to perform better either. Considering our feature data is sampled from the same source iteratively, there are limitations to what we can do to differentiate between them. One solution is to strengthen the significance of the nearest neighbour by multiplying the value to give that feature a bigger impact, which would be true to the spatial relations of a model of "rook" contiguity. However, it may also be helpful for a model of "queen" contiguity, like ours. Another solution we propose is introducing the fourth feature. A discrete value denotes whether the mean of the three current features deviates from the feature data mean. For example 1 for negative deviation, 3 for positive, and 2 for approximate mean. This categorisation might emphasise volatility. These solutions are not evident to provide the prediction model with enough information to ensure better results. However, they are examples of how we can enhance and expand on attributes of the existing data.



(a) Model I, trained on boundary means. (b) Model I, trained on boundary variances.

Figure 17: Predictions of mean- and variance-values.

We propose reconsidering the basic setup behind our model for other alternative approaches. Admittedly, it appears to be too complex and our approach too naive. The feature data is convoluted and sparse, making it hard for a model to learn all aspects of the data. We observe from our models that they learn the approximate mean of the target values but struggle to approximate the specific values. So, we can redefine the task accordingly.

We calculate the mean and variance of each boundary, both in target and feature sets, and construct separate sets for boundary means and variances, ordered temporally. We use similar relations between target and feature values as in the original setup, but in this case, the features are not iterated in space but in time. Target value at time step t, Y_t, is accompanied by feature values, X_{t-1}, X_t and X_{t+1}. We used this data format to train two versions of our two-dimensional Model I, one for the sets of means and the other for the set of variances. The prediction results can be seen in figure 17.

The results look better than from our original experiment. On the other hand, we did use two models to predict the mean and the variance of each boundary, while we initially set out to predict all points on these boundaries with one model. It is not what we intended, but it suggests that the model can generate better predictions on better data. Cause, for the mean solution, the variance of both feature and target data is approximately five times smaller than the original data, making it more reasonable for the model to learn something conclusive. Both suggested models have good results but will not be applicable for the boundary problem we set out to solve. The need for data from the previous and next time steps makes the predictor time-dependent and more data demanding.

We point out the potential opportunities that are with more data. It would be possible to simulate and handle an enormous scope of this spatiotemporal data with the proper facilities. This would open up other ways to solve the problems we have faced. For example, one could feed timesteps of spatial areas into Convolutional Neural Networks, assume images, and experiment with boundary filters.

In hindsight, we reconsider the decisions that led the experiment to dead ends.

Upon revision, the outline of the experiment was naive and relied too heavily on the promise of deep learning. The expectation that the method was supposed to work led us to disregard signs to reevaluate the approach and instead improve the existing one. This led to a feedback loop of incremental changes that did not take the experiment past the local minimum. Consequently, we ended up with a technical search for an optimal model that was more demanding than expected.

8 Conclusion

For this thesis, we set out to use deep learning methods to supply spatiotemporal statistic models. We intended to construct a spatial prediction model to propose a predicted boundary based on sparse iterated data, trained over temporal data. The solution we ended up pursuing did not achieve satisfactory results.

We simulated several data sets from a Spatio-Temporal GARCH model to facilitate the experiment. We utilised these for a broad experimental search for a suitable method. After concluding on the Long Short-Term Memory model, an extensive process of adjusting for optimisation was carried out. Varying and insufficient results caused the experiment to halt. We did not acquire an adequate solution to implement back into the spatiotemporal data, which would be the next step—followed by estimation based on the enlarged spatial area.

We have identified and discussed some possible reasons for this outcome. The starting point for our solution seemed to be too demanding and our approach too naive. We were too committed to the initial idea and unsuccessfully tried to modify it to an optimal solution. Admittedly, there has been spent a lot of time on small adjustments to provoke big changes, which has failed. The experiment stagnated at a local minimum as a result.

However, a compelling result was derived during our experiment: the two-dimensional data outperforming the three-dimensional. As it was used for the LSTM model, the data's change in structure did not seem to compromise it. This may indicate which structure of data LSTM models is most suitable for. We believe this is worth further examination.

Although our conclusion does not prompt so, there is reason to believe that an LSTM model can be of better use in conjunction with spatiotemporal GARCH. With a broader foundation of data, a broader approach could deliver a more consistent prediction model.

A Additional Figures

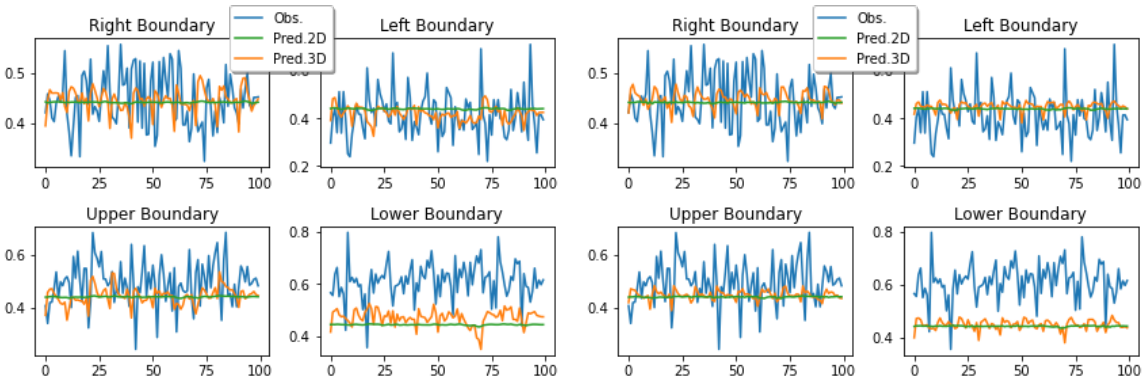


Figure 18: Model I to VI over data 1.

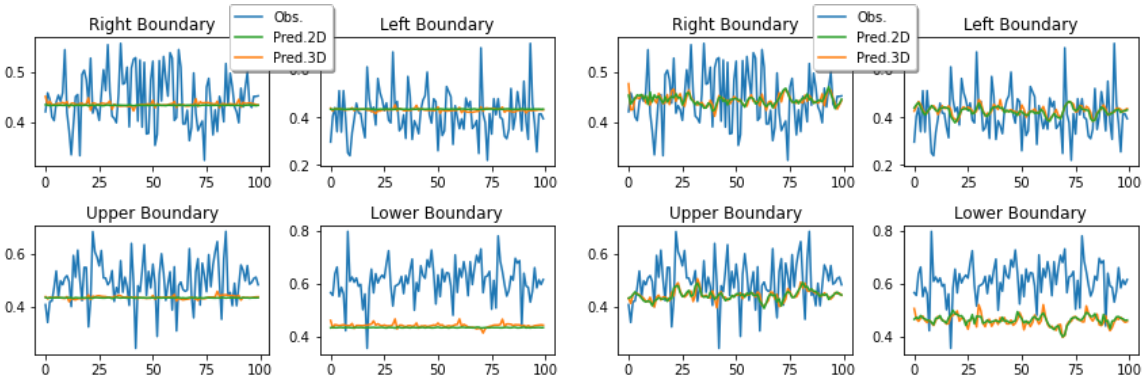


Figure 19: Model III and IV over data 1.

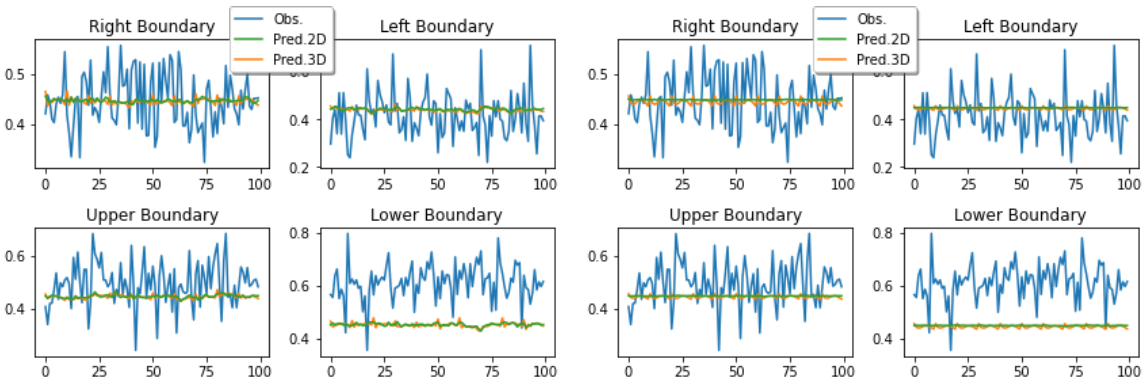


Figure 20: Model V and VI over data 1.

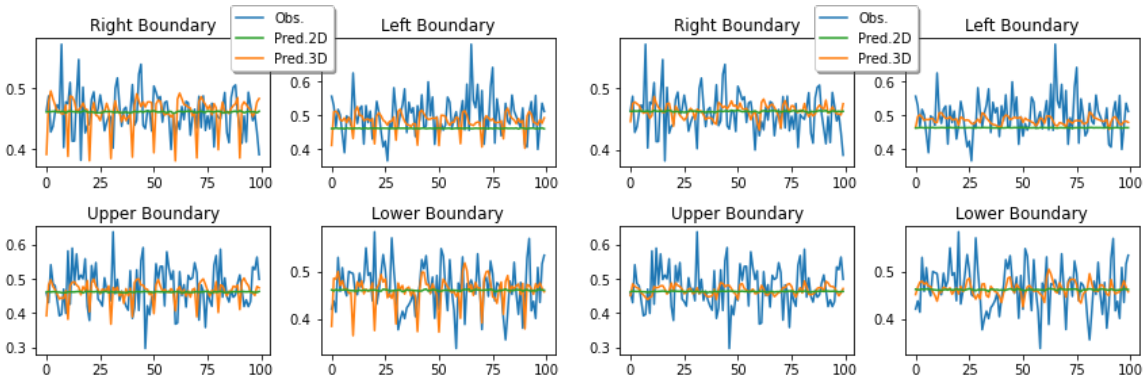


Figure 21: Model I to VI over data 2.

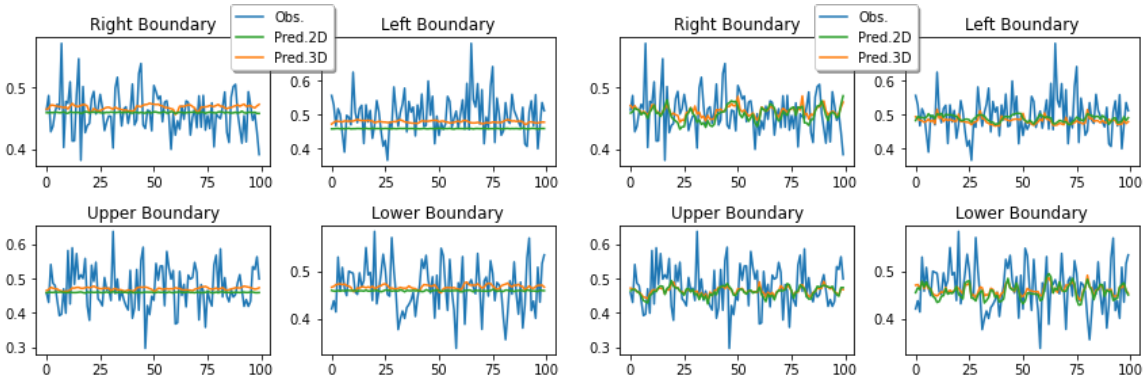


Figure 22: Model III and IV over data 2.

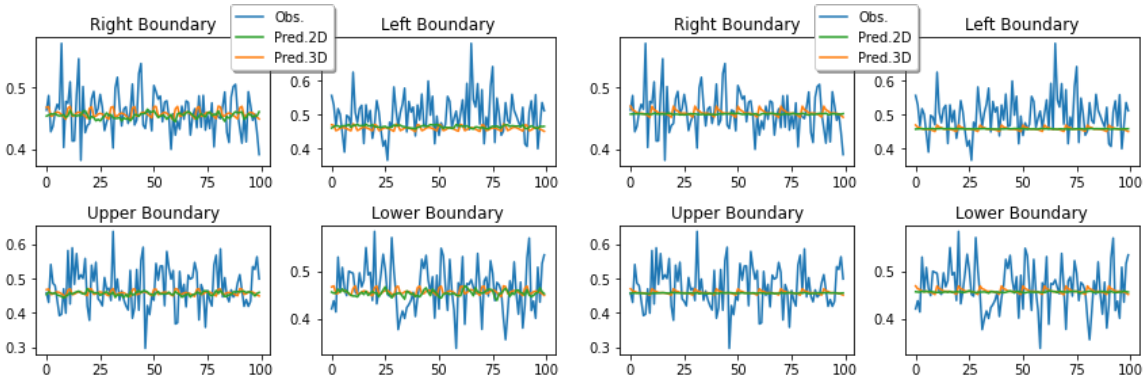


Figure 23: Model V and VI over data 2.

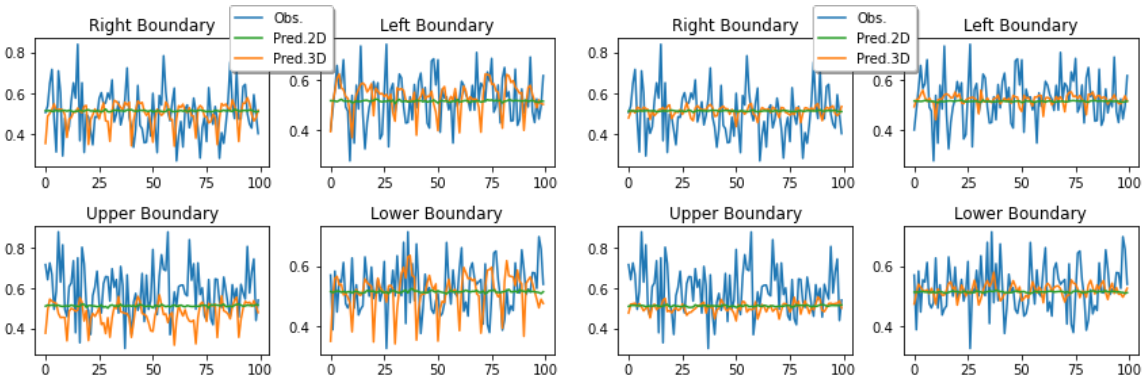


Figure 24: Model I to VI over data 3.

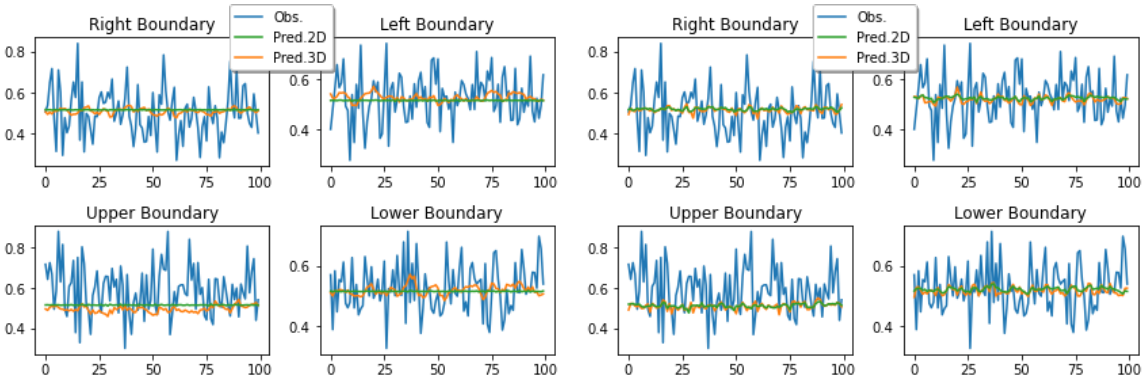


Figure 25: Model III and IV over data 3.

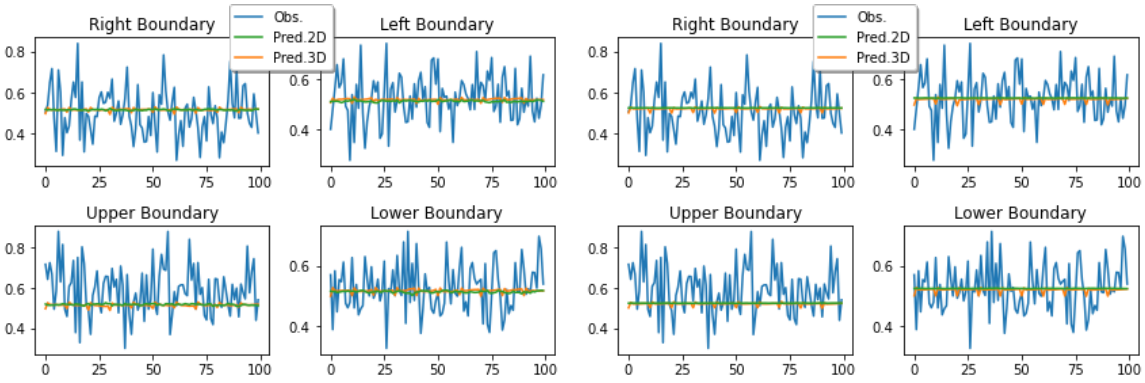


Figure 26: Model V and VI over data 3.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* Retrieved from https://www.tensorflow.org/ (Software available from tensorflow.org)

Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of econometrics*, *31*(3), 307–327.

Bollerslev, T., et al. (2008). Glossary to arch (garch). *CREATES Research paper*, *49*, 1–46.

Brockwell, P. J., Brockwell, P. J., Davis, R. A., & Davis, R. A. (2016). *Introduction to time series and forecasting*. Springer.

Brownlee, J. (2017). *Long short-term memory networks with python: develop sequence prediction models with deep learning*. Machine Learning Mastery.

Brownlee, J. (2018). *Deep learning for time series forecasting: predict the future with mlps, cnns and lstms in python*. Machine Learning Mastery.

Chollet, F. (2018). *Deep learning in python*. Springer-Verlag, New York.

Chollet, F., et al. (2015). *Keras*. GitHub. Retrieved from https://github.com/fchollet/keras

Cressie, N. (2015). *Statistics for spatial data*. John Wiley & Sons.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, *12*(7).

Efron, B., & Hastie, T. (2016). *Computer age statistical inference* (Vol. 5). Cambridge University Press.

Engle, R. F. (1982). Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica: Journal of the econometric society*, 987–1007.

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

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... Oliphant, T. E. (2020, September). Array programming with NumPy. *Nature*, *585*(7825), 357–362. Retrieved from https://doi.org/10.1038/s41586-020-2649-2 doi: 10.1038/s41586-020-2649-2

Hochreiter, S., & Schmidhuber, J. (1997a). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hochreiter, S., & Schmidhuber, J. (1997b, 12). Long short-term memory. *Neural computation*, *9*, 1735-80. doi: 10.1162/neco.1997.9.8.1735

Hølleland, S. (2016). *Spatio-temporal generalized autoregressive conditional heteroskedasticity models* (Unpublished master's thesis). Universitetet i Bergen (UiB).

Hølleland, S. (2020a). *Starmagarch.* https://github.com/holleland/starmagarch. GitHub.

Hølleland, S. (2020b). Volatility modelling in time and space.

Hølleland, S., & Karlsen, H. A. (2020a). Decline in temperature variability on svalbard. *Journal of Climate*, *33*(19), 8475–8486.

Hølleland, S., & Karlsen, H. A. (2020b). A stationary spatio-temporal garch model. *Journal of Time Series Analysis*, *41*(2), 177–209.

Karim, F., Majumdar, S., Darabi, H., & Chen, S. (2017). Lstm fully convolutional networks for time series classification. *IEEE access*, *6*, 1662–1669.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Marsland, S. (2011). *Machine learning: an algorithmic perspective*. Chapman and Hall/CRC.

pandas development team, T. (2020, February). *pandas-dev/pandas: Pandas.* Zenodo. Retrieved from https://doi.org/10.5281/zenodo.3509134 doi: 10.5281/zenodo.3509134

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*, 533-536.

Sezer, O. B., Gudelek, M. U., & Ozbayoglu, A. M. (2020). Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. *Applied Soft Computing*, *90*, 106181.

Tieleman, T., Hinton, G., et al. (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.

Zhou, C., Sun, C., Liu, Z., & Lau, F. (2015). A c-lstm neural network for text classification. *arXiv preprint arXiv:1511.08630*.

Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, *67*(2), 301–320.