TÉCNICO LISBOA

Time Series Forecasting for Mortality and Disease Incidence Data

Diogo Vieira Ferreira

Thesis to obtain the Master of Science Degree in

Biomedical Engineering

Supervisor: Prof. Bruno Emanuel Da Graça Martins

Examination Committee

Chairperson: Prof. Mário Jorge Costa Gaspar da Silva
Supervisor: Prof. Bruno Emanuel Da Graça Martins
Member of the Committee: Prof. Susana de Almeida Mendes Vinga Martins

November 2023

This work was created using LaTeX typesetting language
in the Overleaf environment (www.overleaf.com).

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

Foi chato, duro e por vezes frustrante a elaboração da presente tese. Mas, felizmente, um trabalho deste calibre não se faz completamente sozinho. Para me apoiar tive os meus amigos, muitos deles também a trabalhar em algo semelhantemente chato, duro e às vezes frustrante, a eles e às cervejas que partilhámos em finais de tardes após um dia repleto de chatices durezas e frustrações um grande obrigado. A quem me deu ouvidos enquanto desabafava, em discursos um tanto ou quanto incoerentes, acerca das vicissitudes associadas à tese, entre os quais se destacam a minha mãe e o meu pai, um grande obrigado. Por fim, pelo conhecimento técnico e a experiência prática que me transmitiu, quero agradecer ao meu professor orientador, professor Bruno Martins, obrigado.

Abstract

In the context of mortality forecasting, conventional statistical models like the Auto-Regressive Integrated Moving Average model (ARIMA), and simple machine learning models such as Long Short-Term Memory (LSTM) networks, have been the standard approaches. However, this landscape sharply contrasts with the significant advancements in machine learning, particularly concerning state space models.The focal point of my thesis is to investigate the potential advantages of applying state-of-the-art machine learning models to a mortality forecasting task. To accomplish this, mortality data spanning 252 months was extracted from the United States' underlying mortality database, encompassing 57 different causes of death across five states. This data was used to formulate a mortality forecasting challenge, where models are presented with a 24-month historical context and tasked with forecasting the subsequent 6 months. Six state-of-the-art models were subjected to rigorous testing, which included the S4 and S4 diagonal models. In addition, LSTM and persistence models were utilised as baselines for comparison. The results revealed that, for short-term forecasts, the state-of-the-art models exhibited similar or marginally inferior performance compared to the baselines. However, as the forecasting horizon extended, these advanced models consistently outperformed the baselines. Moreover, it was observed variations in model performance based on the periodicity of the causes of death, with some models excelling at predicting periodic trends, such as the S4, and others being more proficient at forecasting non-periodic patterns, exemplified by the S4 diagonal.

Keywords

Healthcare forecasting; Multivariate mortality forecasting; Mortality forecasting; Time-series forecasting; State Space Models; Machine Learning; United States underlying cause of death database;

Resumo

Na área de previsão em saúde e mortalidade, modelos estatísticos convencionais, como o Auto-Regressive Integrated Moving Average (ARIMA) e modelos simples de aprendizagem de automática, como os Long Short-Term Memory Models (LSTM), têm sido amplamente utilizados. No entanto, este cenário contrasta acentuadamente com os avanços significativos no campo de aprendizagem automática, especialmente no que diz respeito a State Space Models (SSM). O ponto central desta tese é investigar as potenciais vantagens de aplicar os modelos de aprendizagem auomática mais recentes a uma tarefa de previsão de mortalidade. Para tal, extrairam-se dados de mortalidade constituidos por 252 meses da base de dados de mortalidade subjacente dos Estados Unidos, abrangendo 57 diferentes causas de morte em cinco estados. Com esses dados, foi formulada uma tarefa de previsão de mortalidade, no se forneceu aos modelos com um contexto de 24 meses e requisitou-se uma previsão para os próximos 6 meses. Foram testados seis modelos de última geração, incluindo os modelos S4 e S4 diagonal. Além disso, os modelos LSTM e Persistência serviram como bases de comparação. Os resultados revelaram que, para previsões de curto prazo, os modelos de última geração apresentaram desempenho semelhante ou marginalmente inferior às bases de comparação. No entanto, à medida que o horizonte de previsão aumenta, os modelos avançados superam consistentemente as bases de comparação. Além disso, houve variações no desempenho dos modelos com base na periodicidade das causas de morte, alguns modelos destacaram-se na previsão de tendências periódicas, como o S4, e outros mostraram-se mais proficientes na previsão de padrões não periódicos, caso do S4 diagonal.

Palavras Chave

Previsão de Cuidados de Saúde; Previsão de Mortalidade Multivariada; Previsão de Mortalidade; Previsão de Séries Temporais; Modelos de Espaço de Estado; Aprendizagem Automática;

Contents

List of Figures

List of Tables

Acronyms

ARIMA	Autoregressive Integrated Moving Average
AHD	Atherosclerotic Heart Disease
AMI	Acute Miocardial Infarction
BLMN	Bronchus or Lung Malignant Neoplasm
CIHD	Chronic Ischaemic Heart Disease
CNN	Convolutional Neural Network
COD	Causes of Death
CRF	Chronic Renal Failure
DPLR	Diagonal Plus Low Rank
DSS	Diagonal State Space
FCN	Fully Connected Neural Network
FFT	Fast Fourier Transform
GH	Gastrointestinal Haemorrhage
HMD	Human Mortality Database
ICD	International Classification of Diseases
ICU	Intensive Care Unit
iFFT	inverse Fast Fourier Transform
LGB	Light Gradient Boosting
LRA	Long Range Arena
LSSL	Linear State Space Layer
LSTM	Long Short-Term Memory model
LTC	Liquid Time-Constant network
MAE	Mean Absolute Error

MAPE Mean Absolute Percentage Error

MES-LSTM Multivariate Exponential Smoothing Long Short Term Memory

MLP Multilayered Perceptron

MLR Multiple Linear Regression

MSE Mean Squared Error

NPLR Normal Plus Low Rank

OWID Our World in Data

PCA Principal Component Analysis

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

RMSE Root Mean Squared Error

RRSE Root Relative Square Error

S4D S4 Diagonal

SARIMAX Seasonal Auto-Regressive Integrated Moving Average with eXogenous regressors)

SGconv Structured Global Convolution

sMAPE symetric Mean Absolute Percentage Error

SSM State Space Model

SVR Support Vector Regression

VARMAX Vector Autoregressive Moving Average with eXogenous regressors

1

Introduction

Contents

This first chapter will begin by providing the motivation behind this thesis. Then, based on this, it presents the thesis, its goals and research questions, followed by the contributions made and the results obtained from it. Finally, it presents the structure of this document.

1.1 Motivation

In recent times, a significant breakthrough has occurred in connection with modeling sequential data with State Space Models (SSMs), leading to remarkable outcomes, particularly in the Long Range Arena (LRA) [1]. This remarkable advancement can be attributed to the development of the S4 model [2], which integrates a distinctive parameterization of the state space with an efficient algorithm for computing the state space kernel. As a versatile sequence-to-sequence model, the S4 model has demonstrated its adaptability across various domains, including image classification, language modeling, and time series forecasting, consistently achieving competitive results in these diverse tasks. Building upon the success of the S4 model, several variants have emerged, including the S4 Diagonal (S4D) [3] and Diagonal State Space (DSS) [4] diagonal versions, and even a convolutional model inspired by the S4 principles, known as the structured global convolution model [5], among others. These state space models exhibit noteworthy competitiveness among themselves, exhibiting comparable performance within the LRA benchmark. This renders them intriguing candidates for applications that have yet to harness the potential of this development, particularly in healthcare-related forecasting problems, such as multivariate mortality forecasting.

Forecasting plays a pivotal role in healthcare decision-making, equipping stakeholders with the ability to proactively prepare for future circumstances. Time-series forecasting in healthcare finds diverse applications, including predicting medicine expenditures [6], medical bookings [7], and disease incidence, such as forecasting COVID-19 infections [8]. Forecasting results can optimize resource allocation by providing insights into the prevalence of diseases or causes of death within specific periods. Mortality forecasting, in particular, holds vital importance in identifying patterns and trends in mortality rates, thereby guiding the development of effective public health policies and interventions. Moreover, monitoring changes in mortality rates over time is essential in evaluating the effectiveness of medical treatments and interventions. For actuaries involved in life and pension insurance, mortality and disease incidence data assume great significance as they underpin the financial viability of these businesses. However, forecasts in this domain are commonly conducted using Recurrent Neural Networks (RNNs), such as Long Short-Term Memory models (LSTMs), or simpler statistical models like the Autoregressive Integrated Moving Average (ARIMA) or its variations.

These models, although capable in their different applications related to modeling sequential data, are far from the state of the art. The original LSTM [9] is from 1997 and the ARIMA model [10] was

popularised by Box and Jenkins in 1970. On the other hand, the most recent models like the S4 have not yet seen their potential explored in more diverse areas, as is the case with forecasting in healthcare, or mortality forecasting, in particular. This gap between the models currently used in healthcare and state of the art model is the driver behind this research project.

1.2 Thesis Proposal and Research Questions

As hinted in the previous section, my M.Sc. thesis aimed to compare the performance of novel state space models, including the aforementioned variants, with classical and presently employed models in the context of multivariate mortality forecasting. The objective is to generate a 6-month forecast based on a 2-year contextual framework, with a monthly granularity. The dataset used for experiments originates from the United States underlying cause of death database [11], focusing specifically on the most common Causes of Death (COD) in the five most populated states: California, Texas, Florida, New York, and Pennsylvania. The data has been meticulously extracted and processed to facilitate the training and testing of all models. Initially, the hyperparameters of each model were fine-tuned to achieve optimal performance. Subsequently, the models were evaluated using metrics such as the Mean Squared Error (MSE) and the Mean Absolute Error (MAE), to assess their respective forecasting accuracy for 1, 3, and 6-month intervals. The performance of each model was analised according to the state of origin of the data, and according to the causes of death, to see if there are differences in performance given this items. Additionally, the impact of the addition of extra input features, such as State features that encode the state of origin of a given training example was explored. The state of the art models were contrasted against an LSTM and a Persistence model that served as baselines.

1.3 Contributions

The main goal of this thesis was to compare the performance of state of the art models with classical methods in a healthcare forecasting task.

When considering all the features and all the states, the state of the art models performed similarly among themselves across the 3 forecasting windows, with the S4D model performing slightly better. In these same conditions, the baselines were very competitive in the 1 month forecast, with the Persistence model achieving the best performance over all the other models, however this advantage is lost when we increase the forecasting window to 3 and 6 months, since the performance of the baselines deteriorates much more than that from the rest of the models.

It is also interesting to note that results change when we look more closely at the features, by dividing the features into periodical and non-periodical and observing the models performance on these groups.

By doing this, we see differences in the relative performance of the models. For instance we observe that the S4D model is the best at forecasting non-periodical features, while the S4 model is the best at forecasting periodical features.

Finally, the evaluation protocol alongside the respective dataset can be used as a benchmark for additional experiments. All the data and source code was published on GitHub[1], and this resource can be used to test the aptitude of newer models in the healthcare forecasting field.

1.4 Organisation of the Document

This thesis is organised into 5 chapters, excluding the introduction:

- **Chapter 2** discusses **Fundamental Concepts**, introducing the relevant theoretical foundations necessary to understand this work. In here we discuss classical methods used in time series forecasting, along with the basic concepts that are behind machine learning and neural networks, and also how a state space model works.
- **Chapter 3** covers **Related Work**, being divided in two main sections, Section 3.1 discusses recent developments in the healthcare and mortality forecasting areas. In turn, Section 3.2 focuses on the developments around the latest machine learning models, with an emphasis on state space models.
- **Chapter 4** presents the proposed **Methods** discussing the datasets and methodology that was used. The chapter goes over the data processing that was performed along with the training protocol that was adopted and finally it presents the experiments that were performed.
- **Chapter 5** presents **Results** and discusses the conclusions that were obtained in the experiments described in Chapter 4. This chapter is organised in three sections, namely, hyper-parameter adjustment, model comparison, and feature variation, according to the experiments presented in the previous chapter.
- **Chapter 6** presents **Conclusions**, discussing the contributions of this thesis and possible paths to follow in future work.

[1] https://github.com/DiogoVF/Mortality-Forecasting

2

Fundamental Concepts

Contents

This chapter presents fundamental concepts related to time series forecasting, surveying different approaches and models. First, we look at classical methods, such as the ARIMA and the exponential smoothing models that still have practical applications to date. Secondly, we discuss machine learning with neural networks, from the simple perceptron to the more complex LSTM, exploring how these models work and what makes them suitable for time series forecasting. Finally, this section presents a different approach, namely state space models, that has been shown to be a good candidate to model long term dependencies.

2.1 Classical Methods for Time Series Forecasting

Despite the recent trend in applying machine learning to forecasting, there is not yet conclusive evidence that machine learning methods are able to surpass classic statistical methods of forecasting. As evidence to this, we may look at the recent M5 forecasting competition, where one of the baselines was a relatively simple and *off the shelf* solution using exponential smoothing, and only 7.5% of the participants were able to improve upon this baseline [12]. This indicates that classical forecasting methods are not yet obsolete and, therefore, we introduce some that may be useful for comparison.

2.1.1 Persistence Model

The persistence model is very simple and commonly used as a baseline for machine learning and statistical forecasting. The model consists on forecasting the next time step by using the value of the previous time step [13],

$$\hat{y}_{t+1} = y_t. \tag{2.1}$$

2.1.2 Auto-Regressive Integrated Moving Average

ARIMA models [10] correspond to a combination of three operations: an auto-regression, a moving average, and a differencing operation, each controlled by one parameter, p, q and d [13]. Let us now see each of these components and how they combine together.

Starting with the auto-regressive component, in solely auto-regressive models the next time step is estimated as a linear combination of p previous time steps,

$$\hat{y}_{t+1} = l + \alpha_1 y_{t-1} + \alpha_2 y_{t-2} + ... + \alpha_p y_{t-p}. \tag{2.2}$$

In here, l defines the level of the time series and parameter p determines how many of the past values

we will be using, named the lag window. For example, if $p = 3$, the prediction will be a linear combination of the last three values in this series. The p parameter can be chosen based on the auto-correlation function, that measures the correlation between a time series and a lagged version of itself. The lag value with the best correlation is the best choice for p [6]. Additionally, we can also use the partial auto-correlation function, that consists in the correlation between lagged versions of the same signal that are not influenced by previous lags, to estimate this parameter.

The moving average component of the model corresponds to a weighed moving average of the previous q forecasting errors, according to

$$\hat{y}_{t-1} = l + \theta_1 e_{t-1} + \theta_2 e_{t-2} + ... + \theta_q e_{t-q}. \tag{2.3}$$

The q parameter can be estimated using the full or partial auto-correlation function, similar to p from Equation (2.2). Additionally, one can also perform a grid search with the p and q parameters to determine the combination that yields the best results [13].

Finally, we have the differencing component, where the goal is to obtain a stationary time series. First degree differencing consists on measuring the change between consecutive data points, according to the expression

$$y' = y_t - y_{t-1}. \tag{2.4}$$

If the resulting time series is not stationary, one can perform second order differencing where we measure the difference between consecutive data points from a differenced time series. In general, we can have a d order of differencing until the resulting time series is stationary, according to

$$y^d = y_t^{d-1} - y_{t-1}^{d-1}. \tag{2.5}$$

The stationarity of a time series is rather subjective. However, one might use a unit root test, i.e a statistical test that checks if the time series can be divided into three parts, the deterministic, the stochastic and the error and if so if the stochastic component contains a unit root, in order to confirm stationarity.

In conclusion, the ARIMA model takes a non-stationary time series that is converted to a stationary serires via differencing of order d, and the resulting series is modeled using a combination of Equation (2.2) and Equation (2.3), resulting in

$$\hat{y}_{t+1} = l + \alpha_1 y_{t-1} + \alpha_2 y_{t-2} + ... + \alpha_p y_{t-p} + \theta_1 e_{t-1} + \theta_2 e_{t-2} + ... + \theta_q e_{t-q}. \tag{2.6}$$

The final step is to train the model, i.e. finding the best values for θ and α. The most common approach is using the maximum likelihood estimator [13].

2.1.3 Exponential Smoothing

An exponential smoothing model corresponds to a weighted average of the past data points, with more recent points being given more importance than earlier points,

$$\hat{y}_{t+1} = \alpha y_t + \alpha(1-\alpha)y_{t-1} + \alpha(1-\alpha)^2 y_{t-2} + ... = \alpha y_t + (1-\alpha)\hat{y}_t. \tag{2.7}$$

The weights of the previous data points are given by the parameter α, and the higher its value the less relevant are past events. In the limit case, if $\alpha = 1$, then we have the naive/percistence model, corresponding to Equation (2.1). This simple version of exponential smoothing can only be used to describe data without a trend or seasonality, and can only be used to make the forecast one time step ahead. If we want to forecast y_{t+2}, we require the previous y_{t-1} values, which we do not generally have [13].

The previous model can be taken as a basis to develop a family of exponential smoothing models that are able to tackle trends and seasonality. Seasonality can be non-existent, additive, or multiplicative, whereas trend can also be damped. This means that there are 12 different exponential smoothing models in this family, one for each combination of trend and seasonality [14]. Let us look into one of the most used models, namely the Holt-Winters' Three-Parameter Exponential Smoothing model, that tackles an additive trend and multiplicative seasonality:

$$F_{t+h} = (L_t + hT_t)S_{t+h-p}; \tag{2.8}$$

$$L_t = \alpha y_t / S_{t-h} + (1-\alpha)(L_{t-1} + T_{t-1}); \tag{2.9}$$

$$T_t = \beta(L_t - L_{t-1}) + (1-\beta)T_{t-1}; \tag{2.10}$$

$$S_t = \gamma(y_t / L_t) + (1-\gamma)S_{t-p}. \tag{2.11}$$

In this model, the level L, the trend T, and the seasonality S are forecasted separately and then combined to give the final forecast F [13]. The β and γ constants have a similar function to the α parameter in Equation (2.7), but for the trend and seasonal forecasts. The p value corresponds to the periodicity of the data, and h corresponds to the forecasting horizon [13].

The different variations of this family are more adequate for certain types of data than others, depending on the characteristics of such data. In Hyndman et al. [14], a method based on state space models and the Akaike's Information Criterion is presented in order to choose the most adequate exponential smoothing model for the inputted data. This more complex model of exponential smoothing is one of the benchmarks in the M5 competition, managing to outperform 92.5% of the competitors [12].

2.2 Learning with Neural Networks

RNNs are a special kind of neural network designed for time series data. Neural networks are, in turn, a combination of neurons that are organised in layers. As for neurons, these are basically a function that takes the values of the precious neuron layer and computes its own value based on a set of weights and biases. In order to better understand neural networks and how they work we are going to follow a bottom up approach, beginning with basic concepts like the perceptron (a simple kind of neuron), and progressively increasing the complexity until we reach RNNs.

2.2.1 The Perceptron Model

The perceptron was developed in the 1950s and 1960s by Frank Rosenblatt, inspired on earlier work by Warren McCulloch and Walter Pitts [15, 16]. It was based on the way biological neurons work, by integrating the signal coming from the previous neurons. If this input surpasses a threshold, the neuron will fire. In practice, a single perceptron receives a series of real inputs $\mathbf{x} = [x_1, x_2, ..., x_N]$ and it produces one binary output, 0 or 1. The way this works is by multiplying the inputs by weights $\mathbf{w} = [w_1, w_2, ..., w_N]$ that are a characteristic of each perceptron, and compare the value of $\sum_{i=1}^{N} x_i w_i$ with a given threshold, which can also be referred as w_0, resulting in:

$$\text{Output} = \text{g}(\mathbf{x}) = \begin{cases} 1, & \text{if } (\sum_{i=1}^{N} x_i w_i) - w_0 > 0 \\ 0, & \text{otherwise} \end{cases}. \tag{2.12}$$

The main difference between the original perceptron and modern day neurons is the choice of the activation function, g, and the ability to generate continuous outputs. Some popular choices for the activation function include sigmoid functions, namely $e^s/(e^s + 1)$ and $\arctan(s)$, or rectified linear units, such as $\max(0, s)$. The activation function should be continuous and differentiable to allow for the computation of derivatives.

This last property is useful due to the way we train the model. To train a model means to minimize a cost function $\mathcal{L}$ that measures how poorly our perceptron is doing at a given task, given a training set $\mathcal{T} = \{(x^{(k)}, y^{(k)}), k = 1, ..., m\}$ with m instances. The way we do this is through the gradient descent method. Given model parameters $\mathbf{w}$ at a given instant t, we have that:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \Delta\mathbf{w}_t, \ \Delta\mathbf{w}_t = -\eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}}|_{\mathbf{w}_t}, \tag{2.13}$$

where $\Delta\mathbf{w}$ is the step we take after computing the loss at each training example [17]. The step is equal to the negative of the gradient of the cost function, and it is balanced by the parameter η that defines how fast we go down the gradient. If this step size is too big we might overshoot the minimum, but if it is

too small it might take too much time for training to converge

Training also involves choosing a cost function. A common choice for continuous output problems is the quadratic error, given by $\mathcal{L} = \sum_{i=1}^{m} ||y^{(i)} - \hat{y}^{(i)}||^2$ where $\hat{y}^{(i)}$ is the output of the network for input $x^{(i)}$, and $y^{(i)}$ is the desired output. The cost can also be tailored to include other things that we might want to minimize. Two typical methods are the L1 and L2 regularizations, that add to the cost the terms $\lambda||\mathbf{w}||$ and $\lambda||\mathbf{w}||^2$, respectively, preventing the weights from being too large and making it harder for the model to overfit [17, 18]. L1 regularization drives some of the weights to zero, also allowing for feature selection.

2.2.2 Neural Networks

To define a neural network, we organise neurons in an architecture where we choose the number of layers, *N*, and the number of neurons per layer. Let $W^{(n)} = \{w_{ij}\}$, $\mathbf{z}^{(n)} = \{\mathrm{g}(s_j)\}$ and $\mathbf{s}^{(n)} = \mathbf{w}_0^{(n)} + W^{(n)}\mathbf{z}^{(n-1)}$ represent, respectively, the weight matrix, the activations, and the pre-activations for the layer n, with indices i and j referring to the neurons in the next layer and in the previous layer, respectively. Joining it all together we obtain,

$$\mathbf{z}^{(n+1)} = \mathrm{g}(\mathbf{s}^{(n+1)}) = \mathrm{g}(\mathbf{w}_0^{(n+1)} + W^{(n+1)}\mathbf{z}^{(n)}). \tag{2.14}$$

The activations of the neurons are obtained iteratively, with the activations of the next layer, $\mathbf{z}^{(n+1)}$, being obtained from the product of the weight matrix with the activations of the previous layer, $\mathbf{z}^{(n)}$, plus the biases $\mathbf{w}_0^{(n+1)}$. A diagram illustrating this structure can seen in Figure 2.1 [17]. The first activations, $\mathbf{z}^{(0)} = \mathbf{x}$, correspond to the input of the network. Applying Equation (2.14) through all the layers yields the output of the network, $\mathbf{z}^{(N)} = \hat{\mathbf{y}}$.

Training a network is very similar to training a perceptron, and we still use the gradient descent algorithm corresponding to Equation (2.13). However, computing the gradient of the cost function is a little more complicated, since we now have several layers to go through. The solution for this is the backpropagation algorithm, corresponding to:

$$\begin{aligned}
\partial\mathcal{L}/\partial W^{(N)} &= \partial\mathcal{L}/\partial\mathbf{s}^{(N)} \cdot \partial\mathbf{s}^{(N)}/\partial W^{(N)} = (\nabla_{\mathbf{z}}\mathcal{L} \odot \mathrm{g}'(\mathbf{s}^{(N)})) \cdot (\mathbf{z}^{(N-1)})^T, \\
\partial\mathcal{L}/\partial W^{(n)} &= (\nabla_{\mathbf{z}^{(n)}}\mathcal{L} \odot \mathrm{g}'(\mathbf{s}^{(n)})) \cdot (\mathbf{z}^{(n-1)})^T, \\
\nabla_{\mathbf{z}^{(n)}}\mathcal{L} &= ((W^{(n+1)})^T \cdot (\nabla_{\mathbf{z}^{(n+1)}}\mathcal{L} \odot \mathrm{g}'(\mathbf{s}^{(n+1)})) \odot \mathrm{g}'(\mathbf{s}^{n}).
\end{aligned} \tag{2.15}$$

The first elements describe the computation of the derivative of $\mathcal{L}$ in relation to W. In this expression, $\odot$ corresponds to the Hadamard product and g′ corresponds to the first derivative of g. The algorithm begins by computing this derivative for the last layer, where we can immediately compute $\nabla_{\mathbf{z}}\mathcal{L}$. Then,

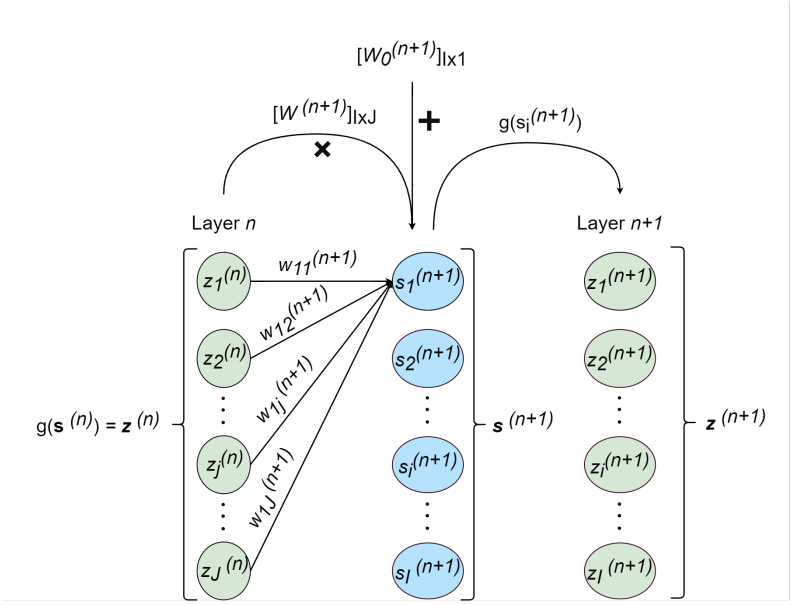

Figure 2.1: Diagram of a neural network consisting of two layers, namely layer n and layer $n+1$ of sizes J and I, respectively. The activations are shown in green and the pre-activations are shown in blue. In the transition between the green and the blue values we have discriminated which elements of matrix W^{n+1} will interact with the neurons of layer n to yield the first pre-activation of the next layer, s_1^{n+1}.

we backpropagate this result to obtain $\nabla_{\mathbf{z}}\mathcal{L}$ for the remaining layers [17].

2.2.3 Recurrent Neural Networks

In feedforward neural networks, like the ones discussed earlier, a single input determines the activation of all the neurons in the network. In RNNs this is not the case, and we instead give the network a series of inputs. The output of the hidden layers for the first input will influence the output of the next input in the series. This process is useful because it gives the network a sense of what has happened before, and how that might influence later outputs. RNNs are adequate for applications that rely on sequential data, where the past may influence future data, such as in the case of speech signals, natural language text, or time series of real values.

RNNs introduce a new kind of input not present in normal neural networks, namely the hidden layer values for the previous input, which we will call memory layer, $\mathbf{m}_t$. The index t represents the number of inputs that have been passed and T is the length of the input series. For instance, $\mathbf{m}_1$ represents the memory layer after the first input and $\mathbf{m}_T$ represents the memory layer after all the inputs. The memory layer will have its own weight matrix, W_m, that will decide how much and what parts from the previous time step do we want to use. By applying the matrix W_m to the memory layer and summing this result with the next input, $\mathbf{x}_{t+1}$, times its own normal weight matrix, W_{input}, we obtain the next iteration of the

memory layer, $\mathbf{m}_{t+1}$. To get the final output of the layer, $\mathbf{z}_T$, we multiply the resulting memory layer from all the inputs, M_T, with the output weights, W_{output}, as described in Equation (2.16).

$$\begin{aligned} \mathbf{m}_{t+1} &= \mathrm{g}(W_M \mathbf{m}_t + W_{input} \mathbf{x}_{t+1}), \\ \mathbf{z}_T &= W_{output} \mathbf{m}_T. \end{aligned} \tag{2.16}$$

Note that the number recursions is the same as the number of inputs in the input series, which corresponds to the number of past data points that we want to consider.

The goal in training a RNN is the same as training a normal neural network: we want to find the optimal values for W_m, W_{input} and W_{output} that minimize a given loss function. For this can we use an adaptation of the backpropagation algorithm, commonly referred to as the backpropagation through time algorithm. If we look at the last part of Equation (2.15), the difference is that we also backpropagate using the weight matrices not only from the different layers but also from the memory layers, so we backpropagate between layers and between recursive layers, i.e., according to time [19].

Standard RNNs, as described until now, have, nonetheless, problems in capturing long term dependencies. There are two main reasons for this: first, because after each recursion the memory layer is a function of the current data point and all the previous data points. This means there is no criteria for what is relevant for the network to remember. Thus, in long sequences, the hidden layer is unable to capture potentially useful information. Secondly, we have the issue of the vanishing gradient. In the normal backpropagation algorithm, we multiply the gradient by the weight matrices that connect the layers, and if there are not too many layers, the final gradient is enough to significantly change the weights of the first layers, leading the network closer to the a minimum of the loss function. However, in backpropagation through time, when we have long sequences we need to backpropagate through all the sequence, this means that, by the end of the sequence, we have multiplied the gradient with the weights of the memory layer many times. If the weights are initialised between 0 and 1, by the end of the sequence we have virtually no gradient to update the weights, which corresponds to a vanishing gradient. If they are initialised above 1, we have an exploding gradient that goes to infinity and is also not useful for training [19]. LSTMs were proposed to address this issue, as they have a way of selecting which information should be kept in the memory layer and which should be forgotten [19, 20].

2.2.4 Long Short-Term Memory Models

In standard RNN models, the recursion consists of one simple layer, namely the memory layer. In LSTMs [9], the recursive part consists of four layers or gates, namely the forget gate, **f**, the input gate, **i**, the output gate, **o**, and the $\tanh$ gate, $\tilde{\mathbf{c}}$. These gates serve the purpose of selecting the information to add, remove, and use at each recursion. The information is stored in the cell state denoted as **c**, that

will be the equivalent of the memory layer and will be changed through element-wise summations and multiplications with the gates. Finally, LSTM's activations, $\mathbf{h}$, will be controlled by the output gate and the cell state [20], as shown in the next equations:

$$\begin{aligned}
\mathbf{f}_t &= \sigma(W_{\mathbf{f}} \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_{\mathbf{f}}), \\
\mathbf{i}_t &= \sigma(W_{\mathbf{i}} \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_{\mathbf{i}}), \\
\tilde{\mathbf{c}}_t &= \tanh(W_{\mathbf{c}} \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_{\mathbf{c}}) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t, \\
\mathbf{o}_t &= \sigma(W_o \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_{\mathbf{o}}), \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t).
\end{aligned} \tag{2.17}$$

A visual representation of these equations can be found in Figure 2.2. Note that in Equation (2.17) the gates are indexed by t, representing their state after input $\mathbf{x}_t$. Note also that the layer is dependent on both the previous activations, $\mathbf{h}_{t-1}$, and the previous cell state, $\mathbf{c}_{t-1}$. The forget gate will decide which elements of the cell state to keep and which to eliminate and to what degree, due to the sigmoid gate that returns values from 0 to 1. The input gate and the $\tanh$ gate select which values should be updated and and what update to give them. Finally, the output gate, will decide which values present in the cell state are relevant, given the current input, and will produce the final activation. Note that this is only one example of LSTMs, since we can vary the connections between the different gates [20].

The dependence of the RNNs memory layer on its previous version, i.e. $\frac{\partial \mathbf{m}_t}{\partial \mathbf{m}_{t-1}}$, was governed by a single weight matrix, $W_{\mathbf{m}}$, leading to successive multiplications by $W_{\mathbf{m}}$ and causing the vanishing gradient problem. However, the LSTM cell state dependence on the previous cell state is now a sum of four elements, including the activations of the forget gate, i.e. $\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} = \frac{\partial \mathbf{f}_t}{\partial \mathbf{c}_{t-1}} \cdot \mathbf{f}_t + \mathbf{f}_t + \frac{\partial \mathbf{i}_t}{\partial \mathbf{c}_{t-1}} \cdot \tilde{\mathbf{c}}_t + \frac{\partial \tilde{\mathbf{c}}_t}{\partial \mathbf{c}_{t-1}} \cdot \mathbf{i}_t$. For the vanishing gradient to occur all the components of this sum would have to go to 0, and since they perform very different roles within the LSTM this is more unlikely [20].

2.3 State-Space Models

SSMs were originally developed to represent physical systems with a set of 1-D inputs $\mathsf{x}(t)$, outputs $\mathsf{y}(t)$ and state variables $\mathsf{u}(t)$ that were related via first order ordinary differential equations, according to:

$$\begin{aligned}
\mathsf{u}'(t) &= A \cdot \mathsf{u}(t) + B \cdot \mathsf{x}(t), \\
\mathsf{y}(t) &= C \cdot \mathsf{u}(t) + D \cdot \mathsf{x}(t).
\end{aligned} \tag{2.18}$$

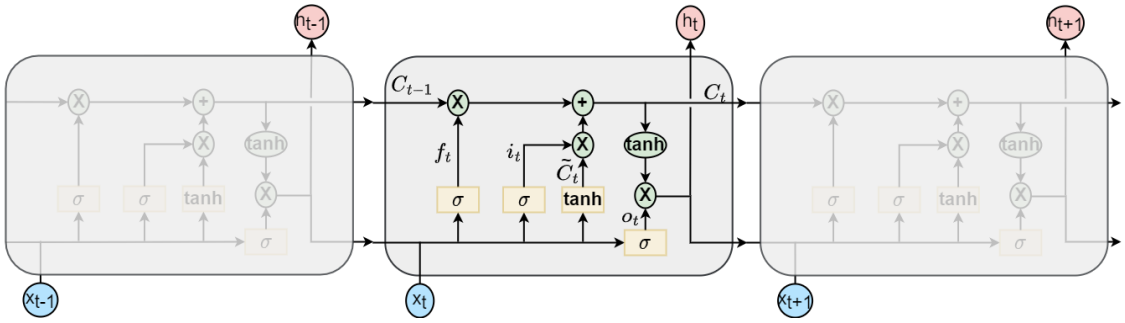

Figure 2.2: LSTM architecture. The divergence of lines represents vector duplication, the joining of lines represents concatenation, the green circles are element-wise operations, the red circles are the layers activations, the blue circles are the inputs of the layer and the yellow boxes represent the different gates and their activation functions.

Equation (2.18), defines a linear system with P inputs, Q outputs and N state variables. As such, matrix A, named the state matrix, has dimensions $N \times N$, matrix B, named the input matrix, has dimensions $N \times P$, matrix C, named the output matrix, has dimensions $Q \times N$, and matrix D, named the feedforward matrix, has dimensions $Q \times P$. The state matrix defines how to update the state space based on the current state space. The input matrix defines how to update the state space based on the current inputs. The output matrix defines how the state space influences the output. Finally, the feedforward matrix bypasses the state space and defines how the current input influences the current output. Recent research [4, 21–23] has shown that SSMs can also be useful as layers in neural networks providing sequence to sequence mapping and being able to solve a variety of problems such as time series forecasting. In this approach, the machine learning problem is to discover the matrices A, B, C, and D.

A time series is a sequence of data points measured at successive and equally spaced points in times. Thus, in order to make forecasts on time series, we must adjust the model in order to accommodate for a discrete input. This is done by selecting a step size Δ such that the derivative in equation Equation (2.18) becomes the next iteration, and $u(t)$ becomes the previous iteration:

$$\begin{aligned} u_k &= \bar{A} \cdot u_{k-1} + \bar{B} \cdot x_k, \\ y_k &= \bar{C} \cdot x_k + \bar{D} \cdot u_k. \end{aligned} \tag{2.19}$$

Note that the matrices also needed to be changed and, assuming a zero order:

$$\begin{aligned} \bar{A} &= e^{(A\Delta)}, \\ \bar{B} &= A^{-1}(\bar{A} - I)B, \\ \bar{C} &= C, \\ \bar{D} &= D. \end{aligned} \tag{2.20}$$

When A is not a diagonal, computing its exponential is computationally demanding, and thus we often resort to approximations such as the bilinear approximation, where $e^{(A\Delta)} \approx (I - \Delta/2 \cdot A)^{-1}(I + \Delta/2 \cdot A)$.

With this transformation the matrices are now as follows:

$$\begin{aligned} \bar{A} &= (I - \Delta/2 \cdot A)^{-1}(I + \Delta/2 \cdot A), \\ \bar{B} &= (I - \Delta/2 \cdot A)^{-1}\Delta B, \\ \bar{C} &= C, \\ \bar{D} &= D. \end{aligned} \tag{2.21}$$

Equation (2.19) expresses a sequence to sequence mapping from $x_k \rightarrow y_k$. We can also see that it is recurrent in u_k, and somewhat similar to a RNN, with memory layer u_k and $\bar{A}$ as the W_m.

Another interesting property of state space models is their ability to be represented as a convolution as well, which facilitates training. This is done by unrolling the recursion in Equation (2.19), so that

$$\begin{aligned} y_k &= \overline{CA^kB} \cdot x_0 + \overline{CA^{k-1}B} \cdot x_1 + ... + \overline{CA^1B} \cdot x_{k-1} + \overline{CB} \cdot x_k, \\ y &= \bar{K} * \mathbf{x}, \\ \bar{K} &\in \mathbb{R}^L = (\overline{CB}, \overline{CA^1B}, ..., \overline{CA^{L-1}B}, \overline{CA^LB}). \end{aligned} \tag{2.22}$$

In the previous expression, $\bar{K}$ is the convolution kernel or filter, and $*$ denotes a convolution operation.

State-space models provide sequence to sequence modeling, where the size of the input is the same as the size of the output. This characteristic is not inherently useful for forecasting, since this task requires going beyond the input sequence. One way to get around this issue is to include padding in the input sequence, meaning that we provide the model with the training sequence padded with a sequence with the size of the intended forecast window. Then we train the model to output a sequence where the forecast is present in place of the padding, this is illustrated in Figure 2.3.

2.4 Overview

This chapter, presents classical methods like the ARIMA, which consists of the combination of three operations: a differencing, a moving average and an auto-regression, governed by hyper-parameter d, i.e. the number of differencing operations, and also q and p, that are the lag values used in the moving average and the auto-regression, respectively. The auto-regression component is simply a linear combination of the previous p time instances regulated by the $\boldsymbol{\alpha} = \alpha_{[1,...,p]}$ parameters. The same goes for the moving average, but now we have only q parameters and the linear combination is not of past values but rather of past errors. Additionally, we have looked at the Exponential Smoothing family of models, the simplest of which makes the predictions as a weighed average of the past values, controlled by $0 < \alpha < 1$. We also explored the Holt-Winters' Three-Parameter Exponential Smoothing model, that supports an additive trend, multiplicative seasonality, and a level as independent exponential smoothings with their respective α values.

When it comes to machine learning methods, this chapter described the LSTM networks and the

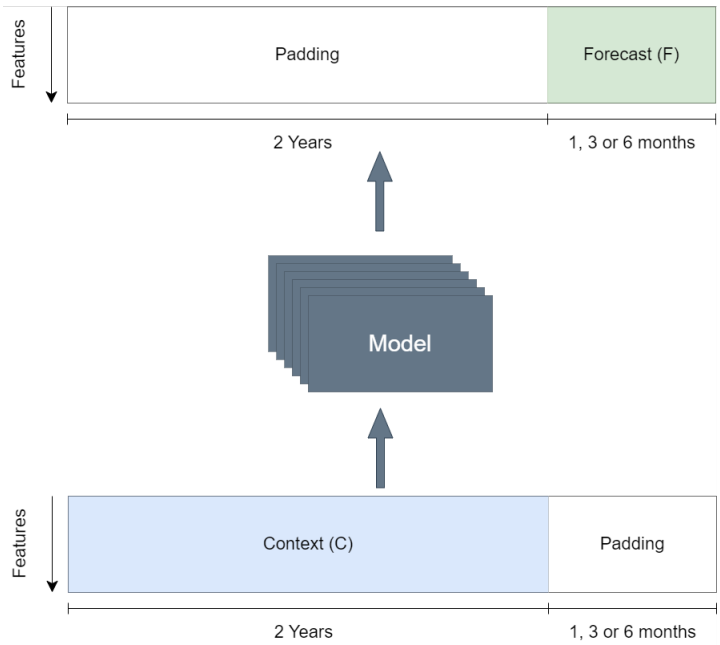

Figure 2.3: This figure illustrates the use of padding to perform forecasts in sequence-to-sequence models. We feed the model the context concatenated with a padded sequence with the size of the intended forecast. The model, in turn, returns the forecast in were previously was the padding.

state space models. The LSTM is a modified recurrent neural network. As such, the activations of the model change as we go through the input sequence. Furthermore, this model considers a representation of the previous inputs and outputs, $\mathbf{h}_t$, that will serve as input along with the next input in the series, $\mathbf{x}_{t+1}$. The representation $\mathbf{h}_t$ is dependent on 4 other layers or gates, namely the forget gate, $\mathbf{f}$, the input gate, $\mathbf{i}$, the output gate, $\mathbf{o}$, and the tanh gate, $\tilde{\mathbf{c}}$, that decide which information to keep or discard as we go along the sequence. Finally, state space models, take a different approach, relying on discretised first order differential equations to relate an input $\mathbf{x}$ with a state space representation $\mathbf{u}$ and the output $\mathbf{y}$, according to Equation (2.19). The state space representation plays a similar role to the $\mathbf{h}$ layer in the LSTM, allowing the SSM to keep track of the inputs that it has seen. Another characteristic of SSMs is their ability to unroll the recursion that comes with the discretisation into a convolution, meaning that the problem of training is reduced to finding the best convolution kernel, $\hat{K}$, for a given task.

3

Related Work

Contents

This chapter takes a look at previous work addressing forecasting tasks in the healthcare domain. It first looks at current applications of forecasting in healthcare, the models that are being used, and to what extent they succeed. Secondly, the chapter analyses recent work in the field of time series forecasting, detached from the healthcare domain. We do this to understand the most recent models, their advantages and disadvantages, and their potential applicability to healthcare forecasting.

3.1 Time Series Forecasting

Forecasting in healthcare is an extremely valueable and versatile tool. It can, for instance, support predictions not only of disease incidence in a public health perspective, but also at the hospital level, allowing hospitals to strategise and use their resources more efficiently. Some applications that will be briefly covered in this section include forecasting medical expenditures [6], disease incidence [8], medical bookings [7] and mortality rates [24, 25].

3.1.1 Healthcare Forecasting

Kaushik et al. [6] compares several forecasting techniques to predict the weekly average expenditure per patient of two medicines, A and B, which are both in the top 10 of the most prescribed pain medications in the U.S.. They selected 5 models to compare, namely the persistence model (Section 2.1.1), the ARIMA (Section 2.1.2), a Multilayered Perceptron (MLP) (Section 2.2.2), an LSTM (Section 2.2.4), and an ensemble. The ensemble model combines the best predictions from multiple models. In this case the authors performed a weighted average of the predictions from ARIMA, MLP and the LSTM, and the weights were computed based on the individual performance of each model. The authors used the Root Mean Squared Error (RMSE) and the R-squared (R^2) to evaluate the models' performance, and these scores were combined into an objective function as $[\text{RMSE}/10 + (1 - R^2)]$, that was used during training. All the models forecasted one step ahead with walk forward validation, meaning that the model uses the training data to make a single prediction which is compared with the real value, that in turn is added to the training set for the next iteration. A similar technique will be used in Chapter 4. To adjust the hyper-parameters of each model, the authors used the autocorrelation and partial autocorrelation functions, followed by a grid search for the p and q parameters of the ARIMA model, with the d parameter being selected based on the amount of differencing operations required to make the time series stationary. For the other models, namely the MLP and LSTM, the hyper-parameters were the number of layers, the number of neurons per layer or hidden size, the batch size, and the number of epochs, which were fine tuned using a genetic algorithm [26]. As for the ensemble, the weights of the average were selected based on the best predictions of 30 runs of each model. With the hyper-parameters tuned, the authors performed 30 runs of each model and selected the best runs of each to compare. The ensemble model

performed the best, closely followed by the MLP and the LSTM models, all of which clearly outperformed both the ARIMA and the persistence models.

A similar comparison between time series forecasting models was performed by Kumar et al. [8], where the goal was to forecast confirmed, active, recovered, and death cases caused by COVID-19, using ARIMA models and the Prophet time series forecasting models [27] developed by Facebook. The data stemmed from 10 of the most affected countries, namely the USA, Spain, Italy, France, Germany, Russia, Iran, United Kingdom, Turkey, and India. The forecasts were performed at the country and world-wide level. The authors used as metrics the RMSE, the MAE, the Root Relative Square Error (RRSE) and the Mean Absolute Percentage Error (MAPE). The experimental results showed the ARIMA models outperformed the Prophet in essentially every taks. Unfortunately, the authors did not test machine learning models.

On another note, Piccialli et al. [7] proposed a forecasting framework in the context of predicting medical bookings using a multi-source approach. Unlike Kaushik et al. [6] and Kumar et al. [8], here the time series of medical bookings is used in conjunction with multi-variate weather and air-quality time series. First, the authors performed a correlation and causality analysis among the considered time series, in order to validate the assumption that there are mutual relations between them. Using techniques such as the Granger causality [28], convergent cross mapping [29], and PCMCI+ [30], they found Granger-causality between the features of both the air-quality and weather time series and the medical bookings time-series, as well as causalities between regressors (for instance, a bidirectional causality between wind speed and carbon monoxide concentrations). These tests also showed correlations between the booking time series and lagged versions of the weather and air-quality time series, leading to the addition of lagged variables into the features space. This addition significantly increases the feature space, which is tackled by using an LSTM-autoencoder to reduce the dimentionality without losing information. The final step is the predictive stage itself, which can be divided into two parts: First, 4 regressors (i.e., a random forest regressor, a gradient boosted decision tree regressor, a lasso and a ridge linear regressor) are used to produce predictions. The second phase consists of using a Convolutional Neural Network (CNN) on the predictions, and a LSTM on the concatenation of the predictions and the original features, to form two different branches that are then combined using a feedforward neural network. The rationale behind this is to have the CNN identify local patterns and the LSTM recognise mid-to-long-term temporal dependencies between the features. This novel ensemble architecture was compared with classical methods, such as the ARIMA, as well as with machine learning methods such as standalone versions of LSTM, MLP, Support Vector Regression (SVR), and other ensemble architectures. The proposed model performed the best.

3.1.2 Forecasting Mortality Rates

The subject of mortality rate forecasting is somewhat distinct from general healthcare forecasting, being used not only as a tool for healthcare decision making but also in actuarial work, particularly in the context of life and pension insurance. Classically, modeling mortality was performed using statistical models such as the Lee-Carter mortality model [31], that decomposes mortality into two main components, namely an age-related component and a time-related component using Principal Component Analysis (PCA). However, with machine learning methods being increasingly more popular, there have been several attempts at extending the Lee-Carter model and other statistical models using machine learning, namely by Perla et al. [24] and Mathonsi et al. [25].

Perla et al. [24] considered the goal of forecasting mortality rates with data from the Human Mortality Database (HMD) [11]. Mortality data can be organised by population groups, with the assumption being that each population group has different mortality behaviours. The authors defined population has the combination of two static categorical variables, namely region, r, and gender, g, with $r \in \mathcal{R} = \{r_1 ... r_N\}$ and $g \in G = \{\text{male, female}\}$. Each population has a mortality matrix, $U^{(r,g)} \in \mathbb{R}^{10\times100}$, comprising the mortality rates for the last 10 years for people of age in range 0-99. The model receives as input the categorical features r and g along with their respective mortality matrix U. Comprised of 3 embedding layers one of which a CNN, responsible for embedding the 3 features. These embeddings are them concatenated and fed through a Fully Connected Neural Network (FCN) layer to decode the embeddings and make the predictions. The most important part of the model are the embeddings, and these are done separately to each feature. The region and gender features are both embedded into $\mathbb{R}^5$, although we should note that these embedding dimensions represent two hyper-parameters of the model,n_r, n_g. For U the authors use a CNN with 32 filters, each of dimension 3, meaning that U will be embedded into $\mathbb{R}^{10\times32}$. This is then passed through a max pooling layer of size 2 and a batch normalisation layer, so that the final embedding is in $\mathbb{R}^{5\times32}$. The authors also tested some variations using an LSTM instead of the CNN, with the rationale being that the LSTM may better capture the temporal dependencies of the data. Besides testing on the HMD data, the authors also tested the models in the United States Mortality Database, in order to confirm if the models were capable of generalisation. The model with the CNN was the best performing method across both the HMD and the United States Mortality Database, being able not only to outperform the LSTM version but also the previous state of the art model, namely the DEEP model [32], therefore paving the way for other machine learning models in mortality forecasting.

On the other hand, Mathonsi et al. [25] considered a different approach to mortality forecasting, the authors introduced a hybrid deep learning model, the Multivariate Exponential Smoothing Long Short Term Memory (MES-LSTM), combining multivariate exponential smoothing with a LSTM. This model is an extension of the hybrid method named ES-RNN developed by Smyl [33] that was able to outperform both statistical and machine learning methods in the M4 competition [34]. The MES-LSTM can be

divided into three main parts, namely the pre-processing, the deep learning, and the post-processing. The pre-processing consists on applying a exponential smoothing to each variable, and the authors consider both cases of additive and multiplicative seasonality. The results of the pre-processing are de-trended and de-seasonalised, using the estimates from the exponential smoothing, and fed through a LSTM layer followed by a Rectified Linear Unit (ReLU) FCN. Finally, the post-processing re-trends and re-seasonalises the data and performs the inverse of the exponential smoothing process to obtain the intended predictions. This model was tested using the Our World in Data (OWID) COVID-19 dataset, where 46 variables were selected, these included total cases, Intensive Care Unit (ICU) patients, hospitalised patients, and total tests, among others. The model was designed to forecast and present the uncertainty of the forecast for two attributes of interest, namely the total cases and total deaths. The MES-LSTM model was tested against Seasonal Auto-Regressive Integrated Moving Average with eXogenous regressors) (SARIMAX), Vector Autoregressive Moving Average with eXogenous regressors (VARMAX), LSTM, DeepAR [35], Multiple Linear Regression (MLR), and Light Gradient Boosting (LGB) models and was able to outperform the baselines in most contexts according to symetric Mean Absolute Percentage Error (sMAPE) and RMSE metrics.

3.2 Recent Methods for Time Series Forecasting

This section discusses the most recent sequence models, their rational, and performance in real datasets. The goal of this section is to allow the readers to gain a general understanding on how and why these models work, and their potential application in time series forecasting.

3.2.1 The S4 Sequence Model

SSMs have two main issues in terms their application to time series analysis. The first issue concerns dealing with long range dependencies. Gu et al. [22] found that basic SSMs perform very poorly in practice when provided with a random A matrix. This can be intuitively understood by noting that linear first-order ODEs solve to an exponential function, meaning that they may suffer from gradient scaling issues, either vanishing or exploding gradients. To solve this, the authors developed a family of matrices $A \in \mathbb{R}^{N \times N}$, named the HiPPO matrices, that allow the state u to memorise the history of inputs x, and can be described as:

$$\text{HiPPO Matrix} = A_{nk} = \begin{cases} (2n+1)^{\frac{1}{2}}(2k+1)^{\frac{1}{2}} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k \end{cases}. \tag{3.1}$$

The idea is to have A save the coefficients of the Legendre series, whose weighted sum approximates the input. These coefficients are updated at each training step. By using this matrix as the initialisation of a basic SSM model, creating the Linear State Space Layer (LSSL), Gu et al. [21] were able to significantly improve the performance of the model.

The second issue with SSMs, which was not tackled by Gu et al. [21], concerns is the computational and memory requirements for the state space representation. This is clear when we look at Equation (2.22), where we can see that, for sequence length L and state space dimension N^2, we have successive multiplications by A, requiring $O(N^2L)$ operations and $O(NL)$ space. A more recent paper on this issue [2] suggests, besides the use of a Hippo class matrix, the use a special representation and algorithm that is able to circumvent the repetitive multiplications of A. This is done in four major steps that will be briefly described next.

The first step is to avoid computing $\bar{K}$ directly, as in Equation (2.22), instead computing its spectrum by evaluating its truncated generating function:

$$\hat{K}_L(z; \overline{A}, \overline{B}, \overline{C}) \in \mathbb{C} := \sum_{i=0}^{L-1} \overline{C}\overline{A}^i\overline{B}z^i. \tag{3.2}$$

The procedure from Equation (3.2) is also referred to as a z-transform, and it essentially converts the convolution filter from the time domain into the frequency domain. We are now in the position to obtain the filter's discrete Fourier transform, by evaluating Equation (3.2) at the roots of unity, $\Omega = \{\exp(2\pi\frac{k}{L} : k \in [L])\}$. Then we can apply an inverse Fourier transformation in $O(L\log(L))$ by using an inverse Fast Fourier Transform (iFFT) to recover the filter in the time domain. The importance of this operation is that we can replace matrix power with an inverse, according to:

$$\hat{K}_L(z) = \overline{C}(I - \overline{A}^L z^L)(I - \overline{A}z)^{-1}\overline{B} = \tilde{C}((I - \overline{A}z)^{-1}\overline{B}). \tag{3.3}$$

Since for all $z \in \Omega_L$, we have that $z^L = 1$, we can insert this term into the constant $\tilde{C}$. Now we have a way to compute $\bar{K}$ using the inverse of a matrix instead of the power.

In the second step, let us look at an ideal scenario, where $A = \Lambda$, i.e. a diagonal matrix. By playing around with Equation (3.3) and making the equations dependent on the original SSM matrices, as in Equation (2.21), we get:

$$\hat{K}_\Lambda(z) = \tfrac{2\Delta}{1+z}\tilde{C}\left[2\tfrac{1-z}{1+z} - \Delta\Lambda\right]^{-1} B = c(z)\textstyle\sum_i \frac{\tilde{C}_i B_i}{\mathsf{g}(z)-\Lambda_i} = c(z)\cdot k_{z,\Lambda}(\tilde{C}, B) \tag{3.4}$$

where $\mathsf{c}(z) = \frac{2\Delta}{1+z}$ and $\mathsf{g}(z) = 2\frac{1-z}{1+z}$. Note that $k_{z,\Lambda}(\tilde{C}, B)$ is a Cauchy Kernel, which has several fast implementations, such as the fast multipole method that can be done in $O(N)$ [36].

The third step consists in relaxing the diagonal assumption to allow a low rank component with

$P, Q \in \mathbb{C}^{N\times 1}$, such that $A = \Lambda - PQ^*$, i.e. a Diagonal Plus Low Rank (DPLR) matrix. Using the Woodbury identity we can compute the inverse as a function of the diagonal matrix and the low rank terms according to,

$$(\Lambda + PQ^*)^{-1} = \Lambda^{-1} - \Lambda^{-1}P(1 + Q^*\Lambda^{-1}P)^{-1}Q^*\Lambda^{-1}. \tag{3.5}$$

Substituting this diagonal A in Equation (3.3), applying Equation (3.5) and distributing terms, we obtain:

$$\hat{K}_{DPLR} = c(z)\left[k_{z,\Lambda}(\tilde{C}, B) - k_{z,\Lambda}(\tilde{C}, P)(1 + k_{z,\Lambda}(Q^*, P))^{-1}k_{z,\Lambda}(Q^*, B)\right]. \tag{3.6}$$

This is now the final version of the convolution kernel, basically consisting of four Cauchy kernels, that are computationally efficient.

Notice that this approach is only applicable to diagonal plus low-rank matrices, which is not the case for the HiPPO matrix. Nevertheless, the HiPPO does have a special structure, namely it is Normal Plus Low Rank (NPLR). Since normal matrices are unitarily diagonalizable they are essentially equivalent to diagonal plus low-rank matrices from the perspective of SSMs, because we are not training the unitary matrices, only the diagonal part. This means that:

$$A = V\Lambda V^* - PQ^* = V(\Lambda - V^*P(V^*Q)^*)V^*, \tag{3.7}$$

for unitary $V \in \mathbb{C}^{N\times N}$, diagonal Λ, and low-rank factorization $P, Q \in \mathbb{R}^{N\times r}$, where r is the rank. Thus, we just need to extract from HiPPO the NPLR representation, and from that representation extract Λ followed by the application of Equation (3.6).

To sum up, the S4 layer is parameterized by first initialising the A matrix as the HiPPO matrix, which we now know is equivalent to $\Lambda - PQ^*$. The trainable parameters for each SSM are the diagonal Λ and vectors $P, Q, B, \tilde{C} \in \mathbb{C}^{N\times 1}$. These are the $5N$ trainable parameters. The S4 defines a 1-D sequence map from $\mathbb{R}^L \to \mathbb{R}^L$, but for problems with multiple features we can define a map of size H, i.e. the number of features, by simply defining H independent copies of the same procedure, basically creating a 1-D sequence map for each feature. Then, in order to capture dependencies between features, a mixing layer composed of two fully connected feed-forward networks is added to obtain the final output.

To finalize, let us now look at the performance of the S4 model in practice. The S4 was designed in order to be able to model long range dependencies in the analysis of long sequences. This problem was formalised by the LRA benchmark [1], which features a series of 6 benchmark problems involving long range dependencies:

- **pathfinder**: a classification task where the goal is two determine if two points are connected by a dashed line in a 32×32 image;
- **pathfinder-X**: an extreme version of pathfinder, with a 128×128 input image;

- **image classification**: consisting on the classification task of flattened images from the CIFAR-10 dataset;
- **byte-level text classification**: this task is set up as a binary classification using the IMDB reviews dataset;
- **long list operations**: this consists of giving the model a long list (length up to $2K$) with operators and numbers organised hierarchically and see if it can perform the operations in the adequate order to produce the right output;
- **byte-level document retrieval**: in this case the model is asked to compare between two large sequences as check their similarity.

The S4 model outperformed every previous model on all the tasks, and it was the first model to be able to solve the pathfinder-x benchmark with a performance better than random guessing [2]. On average, the S4 model achieved a score of 86.09%. Although none of this benchmarks tasks correspond to a forecasting problem, they all test the ability of the models to deal with long range dependencies, which is a very desirable property for forecasting.

Despite being a robust method with very relevant results, the S4 model and its associated algorithm are very complex and not very intuitive. Due to this, several simplifications of the S4 model arose in the literature, including the diagonal state space model [4], the S4 diagonal model [3], and the liquid state space model [37]. The first two models focus their attention in simplifying the A matrix, and instead of using a NPLR matrix stemming from HiPPO they use a diagonal matrix that stems from the diagonalisation of the normal part of the HiPPO matrix, therefore creating a diagonal state space. The liquid SSM variant changes the usual state space model corresponding to Equation (2.18) by adding a liquid time constant, while maintaining the S4 NPLR methodology.

3.2.2 Diagonal State Space Models

Let us begin by analysing the DSS model from Gupta et al. [4]. The S4 model uses a complex method in order to deal with the consecutive matrix multiplications, that are inefficient for non-diagonal matrices. The motivation behind this model is to have precisely a diagonal matrix, or at least a diagonalizable matrix A, to facilitate the computation of the kernel from Equation (2.18). Since A is diagonalizable, we do not require the bilinear approximation from Equation (2.21) and can use a zero hold assumption as in Equation (2.20) to discretize A and obtain $\bar{A}$. So, for a given step size $\Delta \in \mathbb{R}_{>0}$, we now have that,

$$\overline{K}_{\Delta,L} = (\overline{CB}, \overline{CA^1B}, ..., \overline{CA^{L-1}B}, \overline{CA^LB}) = (Ce^{A\cdot j\Delta}(e^{A\Delta} - I)A^{-1}B)_{0\leq j<L}. \tag{3.8}$$

The main difference between the S4 and the DSS models is the way Gupta et al. [4] found to compute $\overline{K}$. For a kernel $K \in \mathbb{R}^{1\times L}$ of state space (A, B, C), where $A \in \mathbb{C}^{N\times N}$ is diagonalizable over $\mathbb{C}$ with eigenvalues $\lambda_i \neq 0\ \forall i \in [1, N]$ and $e^{L\lambda_i\Delta} \neq 1$, there exists $\tilde{w}, w \in \mathbb{C}^{1\times N}$ such that,

$$\begin{aligned} (a) \quad & K = \overline{K}_{\Delta,L}(\Lambda, (1)_{1\leq i\leq N}, \tilde{w}) = \tilde{w}\cdot\Lambda^{-1}(e^{\Lambda\Delta} - I)\cdot \text{elementwise-exp}(P), \\ (b) \quad & K = \overline{K}_{\Delta,L}(\Lambda, ((e^{L\lambda_i\Delta} - 1)^{-1})_{1\leq i\leq N}, w) = w\cdot\Lambda^{-1}\cdot \text{row-softmax}_{\epsilon}(P). \end{aligned} \tag{3.9}$$

In the previous expression, $P \in \mathbb{C}^{N\times L}$ such that $P_{ik} = \lambda_i k\Delta$, and Λ is the diagonal matrix with the eigenvalues λ_i. Either (a) or (b) allow for the computation of K without the use of the z-transform, and consequently the Fourier analysis, and without the Woodbury identity.

If we focus on (a), the equation suggests we can parameterize the state spaces using $\lambda, \tilde{w} \in \mathbb{C}^N$. However, there is an issue with the fact that the real part of the λ_i parameters may become positive during training, which may lead to instability on long inputs, due to the term elementwise-exp(P) that contains terms as large as $e^{\lambda_i\Delta(L-1)}$. This issue can be worked around by restricting the real part of Λ to be positive, which can be done by using the parameters $\Lambda_{real}, \Lambda_{im} \in \mathbb{R}^N$ and computing Λ as $-$elementwise-exp$(\Lambda_{real}) + i\cdot\Lambda_{im}$, where $i = \sqrt{-1}$.

if we look at (b), we see a different way to solve the instability. Instead of restricting the values of Λ we can normalise each row of elementwise-exp(P) by the sum of its elements, ensuring there is no exponential explosion. This is done by again parameterizing the problem as $\Lambda_{real}, \Lambda_{im} \in \mathbb{R}^N$ but computing Λ with no restrictions, i.e. as $\Lambda_{real} + i\cdot\Lambda_{im}$, and using the softmax as we see in Equation (3.9). However, since the softmax can have singularities over $\mathbb{C}$, a special version of the softmax transformation is used, described in detail in [4].

The next step is to decide how to initialize the parameters, for the matrix Λ. We still want to take advantage of the characteristics of the HiPPO matrix, but this is not a diagonalizable matrix, since it is NPLR. In [4], the authors followed a very simple approach by dropping the low-rank components and using only the normal part of the HiPPO matrix, creating the *Skew-Hippo*, that is described as

$$\textit{Skew-Hippo}_{i,j} = \begin{cases} (2i+1)^{\frac{1}{2}}(2j+1)^{\frac{1}{2}}/2 & i < j \\ -\frac{1}{2} & i = j \\ -(2i+1)^{\frac{1}{2}}(2j+1)^{\frac{1}{2}}/2 & i > j \end{cases}. \tag{3.10}$$

Note that by using Equation (3.9) we are concentrating both the B and C matrices into the parameter w or $\tilde{w}$, essentially keeping B and C independent and simply freezing $B = 1$ while training C. Another issue with this model, as noted by Gu et al. [3], is that despite being more simple and intuitive when compared to S4, it uses more memory in the kernel construction, and the row-normalization makes the model dependent on the sequence length L. Moreover, although (b) in Equation (3.9) has comparable results to the S4, option (a) falls somewhat short of having the same performance.

Another diagonal state space approach is the S4D model presented by Gu et al. [3]. This model extends on the results of the diagonal state space model from Gupta et al. [4], where the initialisation of A as the diagonalisation of the normal part of the HiPPO matrix yielded good results. Again, in this model, the main difference originates from the computation of the the convolution kernel,

$$\begin{aligned} \overline{K} &= (\overline{B}^T \odot \overline{C}) \cdot V_L(\overline{A}), \\ V_L(\overline{A})_{n,l} &= \overline{A}_n^l. \end{aligned} \tag{3.11}$$

In the previous expression, $\odot$ is the Hadamard product and V_L is known as the Vandermonde matrix, which can be computed efficiently in $\tilde{O}(N + L)$ operations and $O(N + L)$ space [3]. Note that the computation of the kernel is not dependent on the discretisation approach, as it is in Equation (3.9), so we can use any discretisation we find suitable. The exploding exponential problem occurs again. The authors note that the exponential parameterisation can be used, as in the DSS, but also other functions such as the ReLU or softplus can be used, as long as they are bonded to one side. This model does not constraint the parameterisation of B and C, allowing them to be trained separately. Finally, this model also performs adequately with multiple initialisations, such as having Λ, corresponding to the diagonalised normal part of the HiPPO matrix, but also other approximations of this matrix, described in great detail in [3].

Both diagonal models, i.e. DSS and S4D, outperformed the S4 model in the list operations, text classification and document retrieval tasks. However, on average, across all tasks, the DSS approach using the softmax achieved a score of 81.88% and the S4D approach achieved a score of 84.89%, falling short of the S4's performance.

3.2.3 Liquid State Space Model

A Liquid Time-Constant network (LTC) introduces an input dependent state transition mechanism. In other words, while Equation (2.18) uses a constant A matrix to do the state space transition, we now have a function parameterised by A and B of the input x. In Hasani et al. [37], the authors define a linear LTC state space model as follows:

$$\begin{aligned} u'(t) &= [A + Bx(t)]u(t) + Bx(t). \\ y(t) &= Cu(t), \end{aligned} \tag{3.12}$$

This formalisation can be discretised using the bilinear approximation as:

$$\begin{aligned} u_k &= [\overline{A} + \overline{B}x_k]u_{k-1} + \overline{B}x_k, \\ y(t) &= \overline{C}u_k, \end{aligned} \tag{3.13}$$

where the $\overline{A}, \overline{B}, \overline{C}$ matrices are the same as in Equation (2.21). This also has a convolutional representation as follows:

$$
\begin{gathered}
y_k = \overline{K} * x + \overline{K}_{liquid} * x_{correlation}, \\
\overline{K}_{liquid} \in \mathbb{R}^{\tilde{L}} := K_{\tilde{L}}(\overline{C}, \overline{A}, \overline{B}) = (\overline{C}\overline{A}^{(\tilde{L}-i-p)}\overline{B}^{p})_{i\in[0,\tilde{L}],p\in[2,P]} = \\
= (\overline{C}\overline{A}^{\tilde{L}-2}\overline{B}^{2}, ..., \overline{C}\overline{B}^{p}), \\
x_{correlation} = [\text{all the combinations of the } 2^{nd}\text{,}3^{rd}\text{,...until } P^{th} \text{ order correlation signals}],
\end{gathered}
\tag{3.14}
$$

where $\tilde{L}$ is the length of $x_{correlation}$. There are a couple of things to note from this formulation. Firstly, we have a new hyperparameter P, which represents the maximum order of the correlation terms to be taken into account when computing the output decision. Secondly, notice that the computations of the $\overline{K}$ and $\overline{K}_{liquid}$ parameters are independent, so we can use the S4 method in order to compute $\overline{K}$ efficiently, leaving only the computation of $\overline{K}_{liquid}$.

The efficient computation of $\overline{K}_{liquid}$ composes the second contribution of Hasani et al. [37]. They found that $\overline{K}_{liquid}$ for a given order p, i.e. $\overline{K}_{liquid=p}$, could be computed from the pre-computed S4 kernel, $\overline{K}$, by using the Hadamard product of that kernel with $\overline{B}$ powered to the chosen liquid order:

$$
\overline{K}_{liquid=p} = [\overline{K} \odot \overline{B}^{p-1}] \cdot J_L, \tag{3.15}
$$

where J_L is the backwards identity matrix. Therefore, by applying Equation (3.15) for every $p \in \{2, ..., P\}$ and iteratively appending the results, we obtain the entire $\overline{K}_{liquid}$. The authors named this mode as KB, that stands for Kernel $\times$ B. Another mode emerged when the authors introduced a simplification, where $\overline{A}$ is an identity matrix in the liquid kernel expression, as shown in Equation (3.14). This originates a simpler model, since now the computation of $\overline{K}_{liquid=p}$ can be done as

$$
\overline{K}_{liquid=p} = (\overline{C}\overline{B}^{p}), \tag{3.16}
$$

and again one now just needs to compute this for every p value and we get $\overline{K}_{liquid}$. This mode is called the PB mode, standing for Powers of B.

Note that the parameterisations, the layer composition, and the training methodology is completely identical to the S4 model. When it comes to results, the authors only reported on the liquid-S4 with the PB mode and this model was able to outperform all the previously discussed models in the image classification and list operation tasks of the LRA benchmark, achieving an average performance of 87.32% on all tasks, even with small P values ranging from 2 to 6.

3.2.4 Structural Global Convolution Kernels

Li et al. [5] noted that the S4 model uses a global convolution allied to a DPLR parameterisation and a HiPPO matrix representation to achieve a strong performance. Inspired by this, the authors set out to find basic principles that can be applied to a global convolution in order to make the model perform adequately. They arrived at two intuitive basic principles. First, parameterisation should be efficient in sequence length, meaning that the number of parameters should not be heavily dependent on the size of the sequence. This principle is found both on classic CNNs, who have a fixed kernel, and the S4. Secondly, the convolution kernel should have a decaying structure, meaning that the weights for convolving with close neighbours should be larger than those for distant neighbours. This is very intuitive, since the closer the neighbour the more information it should have, specially when we are talking about forecasting. The S4 model satisfies this due to the consecutive matrix multiplications, but this is not necessarily the case for regular CNNs.

The Structured Global Convolution (SGconv) model, created by Li et al. [5] using multi-scaled sub-kernels and weighted combinations, is based on these principles. For each feature they define the set $S = \{w_i \in \mathbb{R}^d : 0 < i < \log_2(\frac{L}{d})\}$ where each w_i is the parameter for the i^{th} sub-Kernel, k_i, L is the sequence length, and d is the size of each kernel. Note that d will also determine the maximum amount of scaling applied, and the total number of sub-kernels $N = \log_2(\frac{L}{d}) + 1$. Additionally, since the data is discrete, they used an upsampling operation implemented as linear interpolation, denoted as $\text{Upsample}_l(X)$, meaning upsampling x to length l. There are another two parameters to be introduced, namely one to ensure that the convolution does not change the scale of the input, named Z, usually set so that the kernel norm is one, and another to control and induce the decay, α, usually set to 0.5. The final kernel is given by the weighted concatenation of all the sub-kernels,

$$\begin{gathered} K = \frac{1}{Z}[k_0, ..., k_{N-1}], \\ \text{where } k_i = \alpha^i \text{Upsample}_{d2^{\max(i-1,0)}}(w_i). \end{gathered} \tag{3.17}$$

Each successive sub-kernel will have successivly lower weights to account for the second principle, and the decay is controlled by α, see Figure 3.1. In the implementation, the authors use the Fast Fourier Transform (FFT) to efficiently compute the convolution in $O(L \log L)$ time, which is more efficient than the S4.

In terms of experimental results, this model outperformed the S4 in every challenge of the LRA [1], except for the image classification tasks. On average, across all tasks, the SGconv model achieved a score of 87.17%. This means that this model is simpler, more efficient, and more effective at modeling long-range dependencies than the S4.

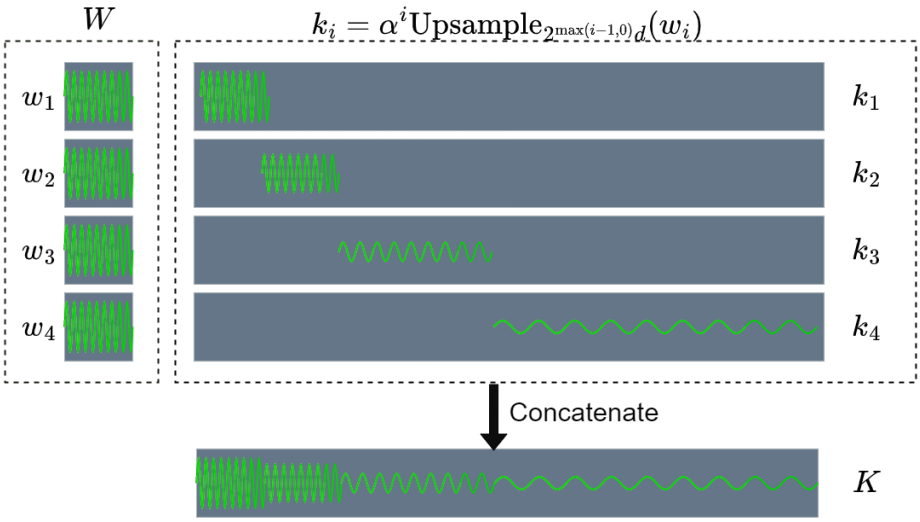

Figure 3.1: Graphical representation of the kernel for the structured global convolution model. Note that despite the image not being to scale, the size of each sub-kernel is double the size of the previous sub-kernel, while having the same number of parameters. Additionally, the weights w_i are represented as being all the same, which is not necessarily the case. The illustration was adapted from [5].

3.3 Overview

This chapter presented some of the current methods being applied in healthcare forecasting. For instance, Kaushik et al. [6] performed a comparison between ARIMA, MLP, LSTM, and ensemble models in forecasting the average expenditure of two of the top ten most prescribed pain medications in the U.S., in a simple time series forecasting task. Piccialli et al. [7] reported on a model that combines LSTM, CNN and FCN layers in order to predict medical bookings with a multi-source approach, using not only the medical bookings time series but also multi-variate weather and air-quality time series. The chapter also explored recent developments in mortality forecasting, with Perla et al. [24] extending the classical Lee-Carter mortality model [31] using CNN and LSTM layers, one to embed the features into a lower dimension, i.e. the CNN, and the other to make the predictions by taking advantage of the temporal dependencies, the LSTM. Mathonsi et al. [25] that took a different approach by introducing a hybrid model for mortality forecasting; i.e. MES-LSTM, that combines exponential smoothing with LSTM layers in order to produce a multivariate forecast on COVID-19 data such as total cases and deaths.

A second part of this chapter focused on recent models in the context of time series forecasting, with a great emphasis in SSMs. I presented 6 models, namely the S4, the DSS, the S4D, two variants of the liquid-S4, namely the PB and KB, and the SGconv. The S4 introduced to the standard SSM a special parameterisation of the A matrix, namely the HiPPO matrix, that minimises gradient scaling issues. Additionally, it introduced a special algorithm to improve the computational and memory requirements of the state space representation. The DSS and S4D models are diagonal versions of the S4, which

allows them to circumvent the S4's more complex algorthim and replace it with simpler computations of the K kernel. The liquid variants of the S4 take a different approach in that, instead of simplifying they add a layer of complexity. In these models, the state transition is performed not by a constant A matrix, but rather by a function, parameterised by A and B, of the input, **x**, with this change makes it so that we now require the computation of two kernels, namely the *S4 kernel* that can be computed with the S4 algorithm, and the *liquid kernel* that is computable by either the PB or the KB methods. Finally, the SGconv corresponds to a convolutional model that takes advantage of some principles of the S4 and translates them into a convolutional approach. This model defines a series of sub-kernels each responsible for a portion of the sequence length. Each successive sub-kernel doubles in length and is halved in size, so that more recent data is given more importance over older data.

4

Methodology

Contents

As we have seen in Chapter 3, there have been several major advancements in machine learning in recent years, namely in the development of deep state space models for sequential data, spearheaded by the S4 model [2]. At the same time, developments in healthcare and mortality forecasting have been focused on the use of statistical approaches in combination with relatively simple machine learning layers, such as the LSTM. This thesis assesses the possible benefits of applying the latest deep learning methods to the field of healthcare forecasting, specially focusing on mortality forecasting. This chapter presents the methodology used to achieve this goal.

The sate space models discussed in Chapter 3 will be tested, namely the S4 model [2], the diagonal state space models S4D [3] and DSS, the variations of the liquid-S4 model with the KB and the PB kernels [37] and the model that uses the structured global convolution kernel [5]. Additionally, The LSTM and Persistence models will be used as baselines.

The different models will be trained and tested on a multivariate mortality forecasting task, which consists of forecasting the mortality rates of several Causes Of Death (COD) when given a two year context. More specifically, the task will consist of three sub-tasks, namely predicting the mortality rates for the next month, the next three months, and the next six months.

The chapter is divided into four main sections datasets (Section 4.1), Methodological approach (Section 4.2), hyper-parameter adjustment (Section 4.3), and experiments (Section 4.4).Section 4.1 takes a look at the United States underlying cause of death database [11], from which the data was collected, explaining the data and the data pre-possessing. Section 4.2 describes the training protocol, Section 4.3 addresses the methodology used to optimize the hyper-parameters of each model. Finally Section 4.4 describes the experiments performed to compare the models and their variations.

4.1 Datasets

As stated previously, the data used in the experiments stems from the United States underlying cause of death database of the United States Center for Disease Control [11]. This database contains mortality counts by underlying cause of death from every U.S. state since 1999 to 2020. However, since the COVID-19 pandemic drastically changed normal habits, making it so that data from 2020 may not be representative, we therefore decided to exclude this year. Additionally, we may choose the temporal granularity of the data: finer data is more desirable, since it allows both for a better understanding of trends and seasonality, at the same time it gives us more datapoints to work with. On the other hand, very fine data will be more susceptible to noise and variations that do not follow the trend and are not useful for generalisations. For these reasons a monthly granularity was chosen as a middle ground.

When considering the geographical origin and granularity of the data, again, very fine data is more susceptible to noise than broader data. However, there may be differences in the behaviour of certain

COD from place to place. With this in mind, the geographical granularity was selected to be state-wise. As such, the 5 most populous states in the United States were selected for data collection: California, Texas, Florida, New York, and Pennsylvania (from most to least populated).

Another issue to consider is that there are 8147 CODs contemplated by the database and a big majority of which do not display significantly high death counts to be present in the database query. This happens because the Center for Disease Control suppresses data that represents fewer than 10 people, due to privacy concerns. Therefore, since we do not want any missing data to be fed to our models, we remove all the CODs that have missing values from 1999 to 2019 and across all the five states. This leaves us with 57 CODs, which is a big reduction. However, this is not a major issue, since the removed CODs do not have significantly high counts to be modelled without significant noise, and they are also the least significant ones in practical applications, since their impact in the healthcare systems is negligible when compared to the other, more prevalent, CODs.

From the remainder 57 CODs we decided to also explore which are the most and least common causes, so we average the death rates across the 21 years and the 5 states. The top three from each category are displayed next alongside their International Classification of Diseases (ICD) codes:

- **Most common**:
    1. Atherosclerotic heart disease - I25.1;
    2. Bronchus or lung, unspecified - Malignant neoplasms - C34.9;
    3. Acute myocardial infarction, unspecified - I21.9;
- **Least common**:
    1. Motor neuron disease - G12.2;
    2. Malignant neoplasm of rectum - C20;
    3. Peripheral vascular disease, unspecified - I73.9;

Some data analysis was also performed with the goal of determining which CODs were more periodical and which were less periodical, so that later, in Chapter 5, we can analyse the differences in model performance based on these features. In order to do this, we used a two step method. First, we detrended every COD by doing a linear regression as a means to estimate the trend and then subtracting the regression coefficient from the data. Second, we computed a Welch periodogram [38] on each feature and determined if it had any peaks with periods below 24 months, since the context will be this size, with a power spectral density above 0.2 across all the five states selected. This yielded a list of 11 periodical CODs and 32 non-periodical CODs. Figure 4.1 displays 4 examples of the features in question, featuring two periodical and two non-periodical COD. Interestingly, the top 10 most common causes of death are included in the 11 periodical features, which is not surprising because least common features are more susceptible to noise.

To sum up, from the database it was extracted a data matrix, $DM \in \mathbf{R}^{252\times57\times5}$, where each value represents the number of people that passed away in a given month from a given COD in a given state

of the U.S.. The final step is to obtain the mortality rates. For this, since the state populations are only updated yearly, we must divide each year by their corresponding state population and multiply by 100 000 to obtain the number of deaths per 100 000 inhabitants. This allows us to easily compare the results between states with different populations. These 57 CODs when organised into a 24 month context will be the features for our forecasting task.

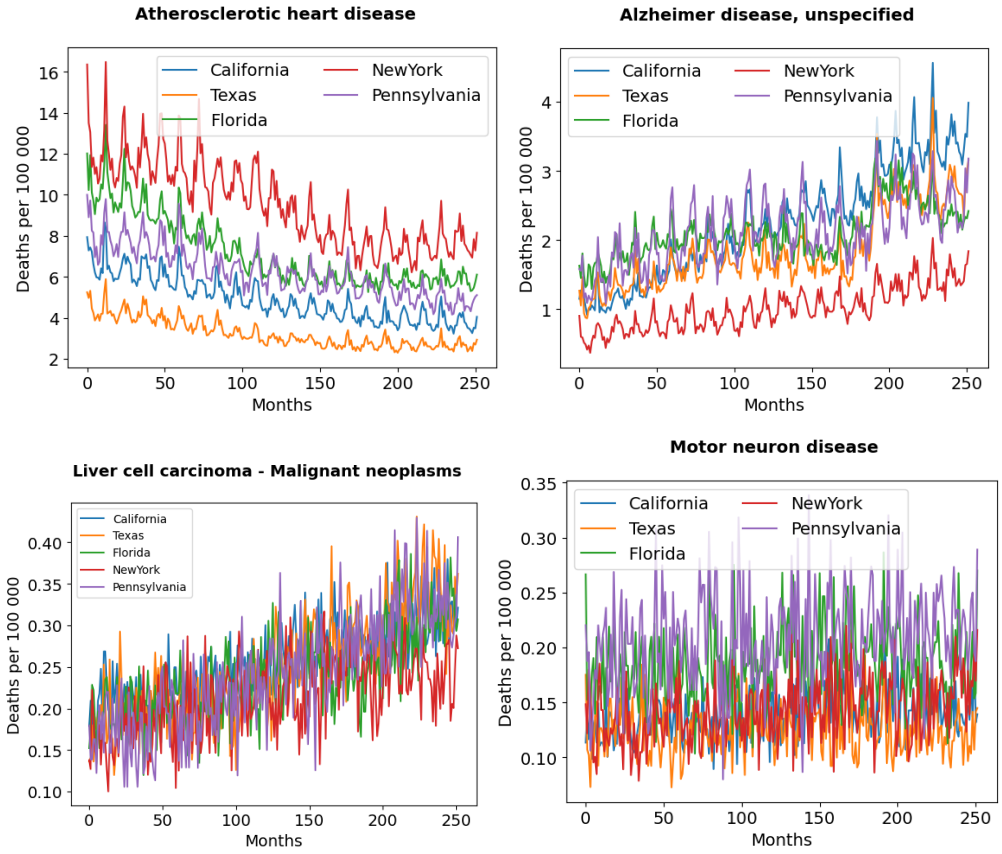

Figure 4.1: This figure presents four graphs representing four causes of death from 1999 to 2019, measured in deaths per 100 000 inhabitants. On the top row we have atherosclerotic heart disease and Alzheimer disease, that are included in both the top 10 most common causes of death and are also among the ones that show the most periodicity. On the bottom, there are other two graphs, namely one from liver cell carcinoma and another from motor neuron disease, which are both, in turn, in the top 10 least common causes of death and are the ones who present the least amount of periodicity.

4.2 Methodological Approach

Being familiar with the data, we are now in a position to discuss the training protocol. If we take any datapoint from the data that was collected, we have only information about the mortality rates and no information relating to the time of the year that datapoint stems from. Therefore, it is very difficult for the models to have a sense of seasonality. For this reason, we introduce two sinusoidal features with a

period of 6 and 12 months

$$S_6 = \sin(x\tfrac{\pi}{3}), \quad S_{12} = \sin(x\tfrac{\pi}{6}), \quad x \in [1, ..., 252] \tag{4.1}$$

, respectively. The two sinusoidal features provide this temporal information.

We also divided the data into training and test sets, consisting of 75% and 25% of the total data, or 189 and 63 months without overlap, respectively, for each of the five states. We will be providing the models with a 24 month context and receive a 6-month forecast, where we will evaluate the 1, 3, and 6 month marks. Hence, we divided the two sets into 30 month parcels, containing the context and the ground truth, that will be used to evaluate the predictions. This process was performed for all five states and the results were merged into two big training and test sets, which had a grand total of 795 training examples and 165 test examples. Finally, in both sets, the parcels were randomly organised into batches of size 64.

The S4, S4Diagonal, DSS, the KB and PB liquid-S4 models, and the convolutional SGconv model, are sequence-to-sequence models in which the length of the input is the same as the length of the output. For this reason, we introduce a mask with the size of the maximum forecasting window, i.e. 6 months, meaning that for these models the sequence length is 30 months. On the other hand, the LSTM does not have a problem receiving an input of length 24 and returning an output of length 30, adding the 6 months of the predictions, Figure 2.3.

The models were all implemented within a simple architecture that is common between them, consisting of two linear fully connected layers, one between the input and the model and another between the output of the model and the actual output of the prediction task, that we will call encoder and decoder, respectively. The encoder allows us to project our variables into a higher or lower dimension feature space, in case that there is information that is better explored in these dimensions. For instance, if there are features that convey the same information perhaps exploring them in a lower dimension space is more beneficial, since we do not have redundant information. On the other hand, the decoder may help the model when it comes to the order of magnitude of the predictions, for instance, allowing the main sequential layer to better capture the dependencies between variables, without *needing to worry* about the order of magnitude.

On what regards the training itself, all models made forecasts for the next 6 months, and the training loss was computed from the MSE, defined as

$$\text{MSE}(Y) = 1/N \sum_{i=1}^{N} ||y_i - \hat{y}_i||^2 \tag{4.2}$$

of all the outputs, including the forecast and the model adjustments to the known values. This was done because preliminary tests showed that the models had a lot of difficulty learning solely based on the

errors of their forecasts. Additionally, the sinusoidal features are removed during the computation of the loss, since we are not interested in predicting the time of the year that the data belongs to. However, these features are present in every other step, namely, their ground truth values are provided to the models when making the predictions. This means that they are present in the mask of the state space and convolutional models and, for the LSTM, we always provide the model with the ground truth of the sinusoidal features rather than the predicted values.

The main goal of the training protocol is find the best possible parameters for the models, that minimize a given cost function. In this case, we want to minimize the MSE of the 6-month forecast. This minimisation process can be performed by several different algorithms, including the gradient descent algorithm, briefly described in Section 2.2, that essentially updates the weights by subtracting the gradient of the cost function, as shown in Equation (2.13). Another more sophisticated method is the AdamW [39] optimizer, that unlike the standard gradient descent also takes into account the variations of the gradient, so we can take a bigger step when the gradient is changing very little and smaller steps when the gradient is changing a lot and the direction of the minimum is not clear. AdamW does this by keeping track of the first and second moments of the gradient, which consist on the exponential moving averages of the gradient and the square of the gradient, respectively. Furthermore, AdamW also has a weigth decay component, similar to the way L2 regularization works, in order to prevent the updates from being too large, reducing overfitting. So, in a similat way to Equation (2.13), we now have

$$\begin{aligned}
\mathbf{w}_{t+1} &= \mathbf{w}_t + \Delta\mathbf{w}_t, \\
\Delta\mathbf{w}_t &= -\eta[\alpha\tfrac{\mathbf{m}_t}{\sqrt{\mathbf{v}_t}+\epsilon} + \lambda\mathbf{w}_{t-1}], \\
\mathbf{m}_t &= [\beta_1\mathbf{m}_{t-1} + (1-\beta_1)\tfrac{\partial\mathcal{L}}{\partial\mathbf{w}}|_{\mathbf{w}_t}]/(1-\beta_1^t), \\
\mathbf{v}_t &= [\beta_2\mathbf{v}_{t-1} + (1-\beta_1)(\tfrac{\partial\mathcal{L}}{\partial\mathbf{w}}|_{\mathbf{w}_t})^2]/(1-\beta_2^t),
\end{aligned} \tag{4.3}$$

where $\mathbf{m}_t$ and $\mathbf{v}_t$ represent the first and second moment and where β_1 and β_2 control the exponential decay, meaning how far back should we consider the average gradient and the average square gradient, respectively. The segment λw_{t-1} represents the weight decay, parameterised by λ, and the parameter ϵ is used to improve numerical stability so that we don't divide by 0. In this work we utilised the following values for the AdamW optimizer: $\eta = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-08,$ and$\lambda = 0.01$.

A key question when training any neural network model is to know when to stop the training in order to avoid overfitting the model to the training data, losing the capacity to generalize to data outside the training set. To address this issue we can take advantage of the test set. Using this set of data, at each epoch, we evaluate the model using the MSE between the 6-month forecast provided by the model and the ground truth, and save the parameters of the model with the lowest 6-month MSE. If the lowest MSE value does not change for 500 epochs I stop training and return the best performing model so far. The value of 500 epochs might seem a lot, but the dataset in use is relatively small, and therefore it does not

take long for the training to stop.

During testing, we collect 6 main metrics, namely the 1-month, 3-month and 6-month errors both in terms of the MSE and the MAE metrics,

$$\begin{aligned} \text{MSE}(Y) &= 1/N \sum_{i=1}^{N} ||y_i - \hat{y}_i||^2, \\ \text{MAE}(Y) &= 1/N \sum_{i=1}^{N} ||y_i - \hat{y}_i||. \end{aligned} \tag{4.4}$$

4.3 Hyper-Parameter Adjustment

The models being tested have different sets of hyper-parameters, and in order to optimise their performance on the task we must select the most fitting hyper-parameters. The general rationale was to select some interval of values for each hyper-parameter, train the models for each combination of hyper-parameters and, based on the MSE of the 6-month forecast of the test set, choose the best combination of hyper-parameters. However, training the models has some variability, i.e. when the same model is trained successively with the same hyper-parameters it does not necessarily reach the same results.This is due to some randomness that is inherent to the models, namely in their initializations. Hence, for each combination of hyper-parameters, we train each model a total of 10 times, which enables not only the analyses of the mean MSE values across these iterations but also the standard deviations of the errors. Which, in turn, allows a more in depth comparison between hyper-parameter combinations, so as to determine which combination consistently outperforms the others.

The state space models, namely the S4, S4Diagonal, DSS, and the Liquid-S4 model with both the KB and PB strategies, have two main hyper-parameters to consider, the state space dimension and the number of features, referred to as N and H, respectively, in Section 3.2. For the state space dimension we used the values $\{16, 32, 64, 128\}$. For the features we tried out the values $\{30, 59, 118\}$. The rationale behind this was to try a number of features, equaling the number of COD plus the two sinusoidal features, and then half and double the number of features to see which dimensionality the models prefer. An additional parameter that will be common across all models is the number of stacked working layers, which varied between the values $\{1, 2, 3\}$. The Liquid-S4 models have an additional hyperparameter the P value that represents the maximum order of the correlation terms to be taken into account, as described in Section 3.2.3. Hasani et al. [37] found that small P values are sufficient to tackle very complex problems like the ones in the LRA. For this reason, and to reduce the computational work, this hyper-parameter will only be varied after finding the optimal values for the other hyper-parameters, and P will take values in the set $\{2, 3, 4, 5\}$.

For the SGconv model, the features and number of working layers are also applicable hyper-parameters. However, instead of the dimensionality of the state space, we have the dimension of each kernel, d. There is, nevertheless, a nuance. As we have seen in Chapter 3, the number of sub-kernels is de-

pendent on their size, with $N = \log_2(\frac{L}{d}) + 1$, and since our sequence length is $L = 30$, and since N needs to be at least 1, we have that the maximum size of each sub-kernel is $d_{\text{max}} = \frac{L}{2^{N-1}} = 30$. So this hyper-parameter varied between $d \in \{4, 8, 16, 30\}$.

Finally, for the LSTM model the number of working layers remain the same. Additionally, we have as an hyper-parameter the hidden size or the size of the activaions, $\mathbf{h}$ as referenced in chapter 2, this size was varied between $\{16, 32, 64, 128\}$. In the other models, namely the S4 model, the feature size, H, would influence the amount of state space, since there would be one state space for each feature. This is not the case in the LSTM. In this model, the size of the input is always converted into the hidden size, so there is no point in changing the number of features, since this is done already by the model, according to Equation (2.17). Additionally, there are two ways through which we can train it to perform long range predictions, with and without teacher forcing, meaning that we either provide the model with its own previous prediction to make the following prediction, or we provide the ground truth value to perform each subsequent prediction. Teacher forcing is already the case when the LSTM models the input context, but let us test if it is useful when making the predictions. The rationale behind this methodology is that, in the beginning of training, the model is very bad at making forecasts, specially when it has to iterate on its own output to produce a longer forecast. The model would benefit from first using teacher forcing, since it makes it easier to train and then progress to not using teacher forcing. To implement this, we introduce another hyper-parameter, namely the probability of a given batch using teacher forcing, P_{tf}, that takes a value in $\{0.0, 0.25, 0.5, 0.75, 1\}$.

4.4 Experiments

With the optimal hyper-parameters selected, we are now in the position to define which experiments will be performed with the different models as well as test out some variations on both the task and the models.

In terms of model comparison the intent is to compare the S4, S4D, DSS, LS4PB, LS4KB and SGconv among themselves, and with the LSTM and Persistence baselines. The goal is to understand if there is any model that stands out overall, in a particular state of the U.S. or set of features for each forecasting window. To achieve this, each model will be trained according to the protocol described in Section 4.2, using the optimal hyper-parameters. For each model, we perform 20 training runs and select, for comparison, the best iterations according to the MSE of the 6-month long forecast. Afterwards, the best candidate of each model will be tested on the 1-month, 3-month, and 6-month forecasts using both MSE and MAE metrics. This test will be performed for each state individually. This is done to see if the models have any geographical preference, since one model may be performing very well in a given state and very poorly in another. Additionally, we will also observe the results through the

features perspective, meaning that instead of averaging the CODs across each state we average the different states across each COD. This way we can understand which CODs the models forecast better or worse.

Notice that during the training process the model has no information about the state of the U.S. where the data originates from. This comes from an assumption that the features behave in a similar enough way between states. To assess this intuition, we will also introduce another set of features, namely the state features, that consist of a one-hot-encoding of the state of origin of the data. The aim of this experiment will be to test the impact that the additional features have on the performance of the models, namely the sinusoidal and state features. Specifically, each model will be trained with each combination of features, meaning that every model will be trained in 4 different setups: without the sinusoidal and state features, with only the sinusoidal features, with only the state features, and with both the sinusoidal and state features. Each of these combinations will be trained a total of 10 times, to account for variability, and using the optimal hyper-parameters. Then, the average MSE values on the test set across 10 iteration of the 1, 3 and 6 month forecasts will be used to evaluate their performance.

5

Experimental Results and Discussion

Contents

This chapter presents and discusses the results from the experimental protocols described in Chapter 4. This chapter is organised in 3 sections, corresponding to the experiments that address hyper-parameter adjustment, model comparisons, and assesment of feature variations.

5.1 Hyper-Parameter Adjustment

This section is focused on the tests used for selecting the optimal hyper-parameters for each model. Each model was assigned an interval of values for each of their hyper-parameters and was trained on every combination of said hyper-parameters. To account for variability in training, each combination of hyper-parameters was trained 10 times. All the models had the number of working layers, n_{wl}, as an hyper-parameter that varied between the values $\{1, 2, 3\}$. The state space models (i.e. the S4, S4Diagonal, DSS, and Liquid-S4 both KB and PB variants) had the number of features, H, and the state space dimension, N, varying between $\{30, 59, 118\}$ and $\{16, 32, 64, 128\}$, respectively. The Liquid-S4s also had their P value varied between $\{2, 3, 4, 5\}$. As for the convolutional SGconv model, the same interval for the number of features was used, but instead of the state space dimension we have the number of sub-kernels, d, that varied between $\{4, 8, 16, 30\}$. The LSTM model had its hidden size change between $\{16, 32, 64, 128\}$ and the likelyhood of teacher forcing, P_{tf} varying between $\{0.0, 0.25, 0.5, 0.75, 1\}$.

Since the models behaved rather similarly, the optimal set of hyper-parameters will be introduced for all of them and then we will use the the S4 and the LSTM model to better understand the training process. The results were as follows:

- **S4**: H=118; N=32; n_{wl}=1.
- **S4D**: H=118; N=16; n_{wl}=1.
- **DSS**: H=118; N=32; n_{wl}=1.
- **LS4PB**: H=118; N=128; n_{wl}=1, P=3.
- **LS4KB**: H=118; N=16; n_{wl}=1, P=4.
- **SGconv**: H=118; d=16; n_{wl}=1.
- **LSTM**: h=118; P_{tf}=0.75; n_{wl}=1.

Figure 5.1 illustrates the hyper-parameter adjustment process for the S4 model. This figure presents three plots, one for each number of features tested. In each plot we have three lines (i.e. one green, one orange, and one blue) that represent the number of layers tested. On the x-axis we have the variation of the SSM size. There are a total of 36 combinations of these hyper-parameters, with each being trained 10 times. The graphs present the average of these 10 iterations with a solid line and the standard deviation above and below the average as a shaded area. In the y axis there is the MSE of the 6 month long forecasts.

The first conclusion from Figure 5.1 is that, regardless of the number of features and the SSM size, a single working layer is always more beneficial. This may be the case because the task at hand is sufficiently simple to be handled by a single layer. To put things into perspective, a single layer of the S4

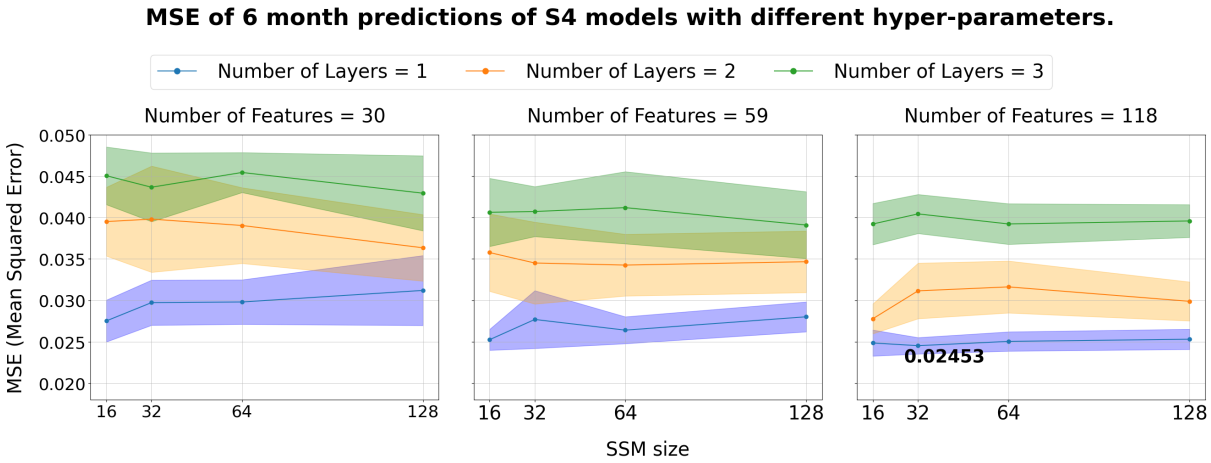

Figure 5.1: Hyper-parameter adjustment of the S4 model. In the x-axis we have the variation of the SSM size. The 3 charts present the variations on the number of features, and the different colored lines encode the number of layers. Each combination of these hyper-parameters was trained 10 times and the average of these results is displayed in solid line and dots, while the standard deviation above and below the average is shaded in. The performance of each alternative is measured in the y-axis using MSE as a metric. The MSE was computed from 6-month long forecasts.

model has $5N$ trainable parameters per feature. With $N = 32$, this totals to 160 parameters per features for a task where each feature has only 24 past values, it may be more than sufficient. A further increase in the number of layers may only lead to an excess of parameters and make the model more prone to overfitting.

Secondly, we can see from Figure 5.1 that increasing the number of features also has a beneficial impact on performance. This is also the case for all the other models that were tested. Increasing the number of features also increases the number of parameters, but it does not directly increase the number of parameters per feature. This behaviour may indicate that the features are more easily interpretable in higher dimensions. This may be due to having the models decomposing the feature into several components that are easier to analyse. An example of this decomposition may be decomposing the time series into trend and seasonality and forecast them separately.

Finally, when we observe the SSM size in Figure 5.1, we can see that it is the least significant hyper-parameter, as its impact on MSE is much less than the number of layers or the number of features. When the number of features is 118, there is no discernible difference in performance when we change the SSM size, since all the MSE values are within the standard deviation. Nevertheless, the SSM size that presented the lowest average MSE was selected.

Figure 5.2 has the same structure as Figure 5.1, but instead of the number of features we have the likelihood of teacher forcing, and instead of the SSM size we have the hidden size. A single working layer continues to be the preferred, with additional layers presenting more variability and higher averages (so much so that the results for 3 layers are not displayed). Increasing the likelihood of teacher forcing

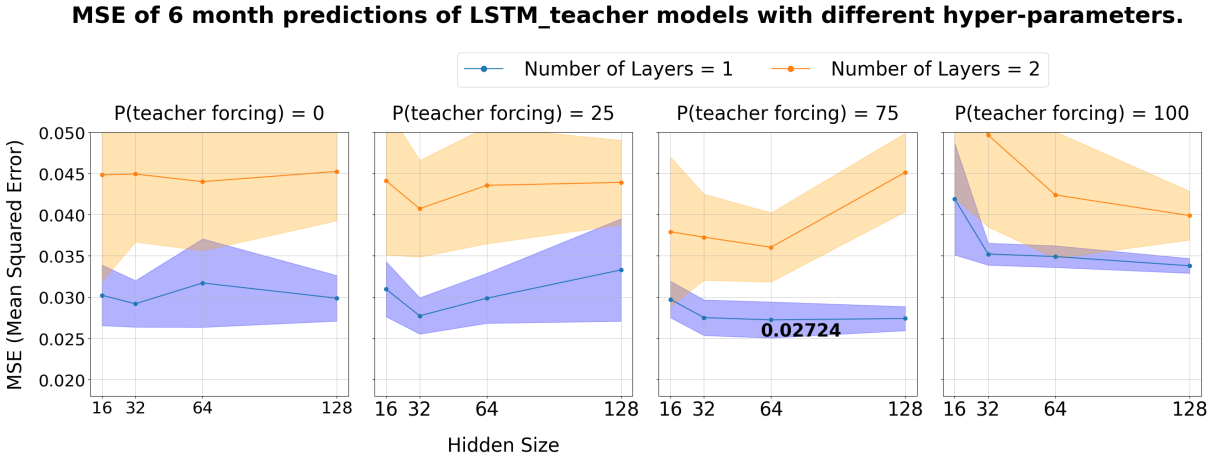

Figure 5.2: Hyper-Parameter Adjustment of the LSTM model. In the x-axis we have the variation of the hidden size, h. Each chart presents a value of the probability of teacher forcing, and the colored plots represent the number of layers being used. A number of layers equal to 3 is not displayed since the MSE for the combinations with this hyper-parameter is too high. Each combination of hyper-parameters was trained through 10 iterations. The average of these results is displayed in solid line and dots, and the standard deviation above and below the average is shaded in. The y-axis measures the performance of each combination of hyper-parameters using the MSE of the 6-month long forecast.

seems to improve the performance up until 75%. In this plot the variation of the hidden size does not seem to have a significant impact on the performance of the model.

5.2 Model Comparison

This section focuses on comparing the performance of each model on the test dataset, considering the 1 month, 3 month and 6 month forecasts, using both MSE and MAE metrics. This was done by training each model 20 times and selecting the iteration that performed the best according to the average MSE of the 6 month long forecasts. With the best performing version of each model selected, we are in a position to make predictions on the test dataset. We will look at these results through two lenses. First, we consider the state lens where the goal is to understand if there are differences in the performance of the models according to the state of origin of the data. Second, we perform a similar analysis but now looking at the performances of the models in each feature, to understand if there are features or classes of features which are better modelled by one model or another. In order to perform both these analysis the predictions were normalised using a min max normalisation of each feature:

$$F_{normalised} = 10 \times \frac{F - F_{min}}{F_{max} - F_{min}}, \tag{5.1}$$

Where F_{max} and F_{min} are the minimum and maximum values of each feature. This allows the comparison of the MSE and MAE values between features, states and models.

Table 5.1 presents the performance of the models in each state of origin. The first conclusion to draw from this table is that, on average, the best performing model was the S4D, having the best results for the 3 month and 6 month forecasts, and the second best in the one month forecast according to both metrics. Interestingly, the persistence model was the best performer in the 1 month forecast, even managing to be the best model in the three month forecast for Texas and Florida, which goes to show that simple models can still be very useful.

Another takeaway from Table 5.1 is that models that present the best performance on a given state of origin do not show the same dominance in other states. For instance, the S4 model was the best performing model across all tasks and both metrics in California, but not in the other states. Similarly, the S4D model was the best in Pennsylvania. Also, when comparing the overall results within models, we can see that the S4, the S4D, and the Persistence models achieved better results in California, whereas the LSTM had better results in Texas. The complete table can be found in Appendix A.

Table 5.1: Comparison of the S4, the S4D, the LSTM, and the Persistence models on the state of origin of the data. The models were trained a total of 20 times with their respective optimal hyper-parameters, and the best performer of these models in the 6 month long forecast was selected. Each best performer was evaluated in three tasks,namely one month, three month and six month forecasts, using both the MSE and theMAE metrics. Highlighted in **bold** and underlined are the best and second best results, respectively, on the given task and metric across all the 8 models tested.

	S4						S4D					
	1		3		6		1		3		6	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
California	**0.9725**	**0.7295**	**0.9813**	**0.7408**	**0.9711**	**0.7393**	1.0074	0.7844	1.0453	0.7988	1.0696	0.8128
Texas	1.5131	0.9518	1.5296	0.9532	**1.4654**	**0.9273**	1.6065	0.9746	1.6579	0.9862	1.6428	0.9864
Florida	2.1984	1.1784	2.2330	1.1897	2.1124	1.1647	2.0992	1.1061	2.0736	1.1065	**1.9617**	**1.0864**
New York	1.7662	0.9965	1.8027	1.0207	1.9762	1.0670	1.8360	1.0201	1.9783	1.0583	2.1017	1.0950
Pennsylvania	2.7268	1.2790	2.7945	1.3052	2.8229	1.2987	2.4120	1.1950	**2.3484**	**1.1855**	**2.3779**	**1.1983**
Average	1.8354	1.0270	1.8682	1.0419	1.8696	1.0394	1.7922	1.0160	**1.8207**	**1.0271**	**1.8307**	**1.0358**

	LSTM						Percistence					
	1		3		6		1		3		6	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
California	1.5264	0.9133	1.8421	1.0103	1.8250	1.0037	0.9752	0.7339	1.3315	0.8723	1.6444	0.9628
Texas	1.3211	0.8619	1.5924	0.9502	1.7115	0.9997	**1.1912**	**0.8130**	**1.5002**	**0.9156**	1.7647	0.9899
Florida	2.0664	1.0967	2.5928	1.2303	3.2290	1.3822	**1.8790**	**1.0205**	**1.8711**	**1.0518**	2.1920	1.1312
New York	1.7889	0.9904	2.0258	1.0700	2.2295	1.1277	**1.6681**	0.9759	1.9887	1.0715	2.3653	1.1899
Pennsylvania	2.7469	1.2698	2.9790	1.3324	3.1140	1.3735	3.0123	1.3217	3.4371	1.4283	4.2717	1.6339
Average	1.8900	1.0264	2.2064	1.1187	2.4218	1.1774	**1.7452**	**0.9730**	2.0257	1.0679	2.4476	1.1815

To better understand these models let us now explore their performance when viewed from the features perspective, results for this are shown in Table 5.2. To make this table the MSEs across all the states of origin, in order to obtain the average MSEs of the features. In Section 4.1 I classified the features into periodical and non-periodical. This table reflects this classification, presenting 3 examples

of periodical features and 3 examples of non-periodical features, along with the average of across both classifications. We can see that, overall, the models present a better performance on the more periodical features. We can also see that the models that perform the best on periodical features, namely the S4 and the LSTM, are not the same as the ones that perform the best on the non-periodical features, namely the S4D and the Persistence. The complete table can be found in Appendix A.

Figure 5.3 presents a more in depth look at the performance of the models on the two groups of features and across all the features. Each boxplot presents six metrics. The bottom and the top of the boxes are the first and third quartiles, while the whisker are the 5 and 95 percentiles. The orange line is the median and the green triangle is the average of the results. Each model has 3 boxplots, one for the periodical features, G, another for the non-periodical features, B, and a final one with all the features, A. The smaller the box, the more consistent the model is, and the closest the box is to 0 the more accurate the model is. For this graph, the S4 and S4D models were selected to be compared with the LSTM and Persistence baselines (the complete version of this graph can be found in Appendix A) because the S4 is the best performer when it comes to the periodical features and the S4D is the best performer in the non-periodical features. We can tell right away that the models have a significantly better performance on the periodical features when compared with the non-periodical features, which is expected.

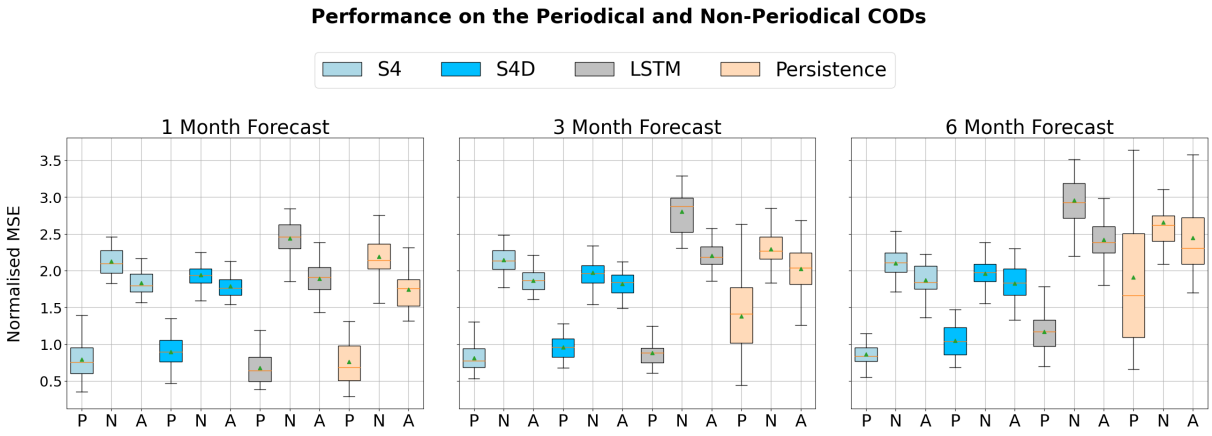

Figure 5.3: Boxplots of the normalised MSE of the 1, 3 and 6 month predictions for the S4, S4D, LSTM and Persistence models, the values plotted are the average of each of the samples in the test set across the states and the features. On the x axis we vary the category of features that were used, the periodical features, P, the non-periodical features, N, and all the features, A. Each boxplot encompasses six measurements, the bottom and top of the box are the first and third quartiles, respectively, the whiskers are the 5 and 95 percentiles, the orange line is the median and the green triangle is the average. Note that the best performing model changes according to both the category of the features and the length of the forecast.

Let us first focus on the periodical features. In the 1 month forecast the LSTM and the Persistence models perform slightly better, and the S4D is somewhat worse. However, if we increase the forecast to 3 months, the Persistence model worsens significantly, becoming the worst performing, and the LSTM becomes a bit worse than the S4. Increasing the forecast to 6 months turns the advantage to the SSMs,

Table 5.2: Comparison of the S4, the S4D, the LSTM and the Persistence models, with their performance grouped by feature. The models were trained a total of 20 times with their respective optimal hyper-parameters and the best performer of these models in the 6 month long forecast was selected. Each best performer was evaluated in three tasks, corresponding to one month, three month and six month forecasts using both the MSE and the MAE metrics. The results were organised by feature, meaning that the results were averaged across the state of origin. The features were organised into periodical features (P) and Non-Periodical features (NP). Three features from each of these categories were selected as examples to be displayed along with the average results from the respective groups. Highlighted in **bold** and underlined are the best and second best results, respectively, on the given task, metric, and features, across all the 8 models tested. The periodical features are Atherosclerotic Heart Disease (AHD), Bronchus or Lung Malignant Neoplasm (BLMN),Acute Miocardial Infarction (AMI). The non-periodical features areGastrointestinal Haemorrhage (GH),Chronic Ischaemic Heart Disease (CIHD),Chronic Renal Failure (CRF).

	S4						S4D					
	1		3		6		1		3		6	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
AHD	0.3602	0.4303	0.3617	0.4386	0.4440	0.4848	0.3655	0.4412	0.3875	0.4690	0.4256	0.4854
BLMN	**0.2825**	**0.4218**	**0.2601**	**0.3950**	0.3069	0.4326	0.5901	0.6618	0.5227	0.6223	0.5221	0.6095
AMI	**0.2969**	**0.4268**	**0.3242**	**0.4485**	**0.3677**	**0.4752**	0.5113	0.5875	0.4939	0.5565	0.5849	0.6132
Avg P	0.7957	0.6530	0.8154	**0.6572**	**0.8613**	**0.6775**	0.8975	0.7267	0.9622	0.7480	1.0473	0.7841
GH	2.1990	1.2372	2.3906	1.3039	1.9460	1.1894	0.7710	0.6390	0.7338	0.6155	0.6973	0.6250
CIHD	2.5838	1.3623	1.9069	1.1376	2.1470	1.2455	1.1084	0.8283	**0.8819**	**0.7256**	**0.8085**	**0.7295**
CRF	**0.8635**	**0.7246**	**0.8813**	**0.7346**	**0.9756**	**0.7708**	0.9302	0.7290	1.0640	0.7758	1.0863	0.7780
Avg N	2.1264	1.1323	2.1497	1.1434	2.1010	1.1269	**1.9433**	**1.0658**	**1.9729**	**1.0809**	**1.9645**	**1.0827**
Avg	1.8354	1.0270	1.8682	1.0419	1.8696	1.0394	1.7922	1.0160	**1.8207**	**1.0271**	**1.8307**	**1.0358**

	LSTM						Percistence					
	1		3		6		1		3		6	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
AHD	**0.2495**	**0.3655**	**0.2070**	**0.3452**	**0.2373**	**0.3756**	0.4203	0.4745	0.8210	0.6787	1.2252	0.8590
BLMN	0.4579	0.5257	0.6137	0.6200	1.0582	0.8339	0.5947	0.6016	0.5344	0.5746	0.6440	0.6278
AMI	0.6192	0.6213	0.9392	0.7960	1.5104	1.0087	0.5664	0.5864	1.1859	0.8963	1.8153	1.0827
Avg P	**0.6794**	**0.6155**	0.8794	0.7154	1.1670	0.8281	0.7639	0.6441	1.3790	0.8773	1.9115	1.0525
GH	**0.7454**	**0.6102**	**0.6738**	**0.5725**	**0.6109**	**0.5457**	0.9738	0.6772	1.0506	0.7415	1.2036	0.7775
CIHD	1.1228	0.8366	1.6752	1.0142	1.9188	1.1184	**1.0211**	**0.7812**	1.3054	0.9125	1.4963	0.9621
CRF	1.4305	0.9612	1.5008	0.9792	1.6632	1.0397	1.2726	0.8903	1.8049	1.0070	2.0237	1.0922
Avg N	2.4395	1.2030	2.8047	1.2912	2.9551	1.3275	2.1933	1.1234	2.2947	1.1480	2.6557	1.2316
Avg	1.8900	1.0264	2.2064	1.1187	2.4218	1.1774	**1.7452**	**0.9730**	2.0257	1.0679	2.4476	1.1815

with the S4 taking the lead. Surprisingly, the distribution of the S4 results seem to narrow as we increase the forecast window.

Now looking at the non-periodical features, the S4 and S4D models clearly perform better than the baselines, especially in the 6 month forecast, with the S4D being slightly better than the S4. Overall better results were achieved with state space models in this context, suggesting that these more complex models are better suited to deal with non-periodical features. When we look at all the features we see a similar behaviour as with the periodical features: in the 1 month forecast the baselines' performance is comparable with the S4 and S4D, but as we increase the forecast window the baselines' results rapidly deteriorate.

Figure 5.4 presents an overview of the results for all the models tested, where each models' performance is plotted in terms of the average performance on each test sample, i.e. the average normalised MSE across the state of origin and features for each test sample. This graph shows that the state space and convolutional models perform very similarly, with the S4, S4D, and SGconv models performing slightly better in the 1 and 3 month forecasts. Another takeaway from this graph is the fact that the more complex models are able to handle the increase in the forecast horizon much better than the simpler LSTM and Persistence models. However, the Persistence model has a very good performance in the 1 month forecast, outperforming the remaining models.

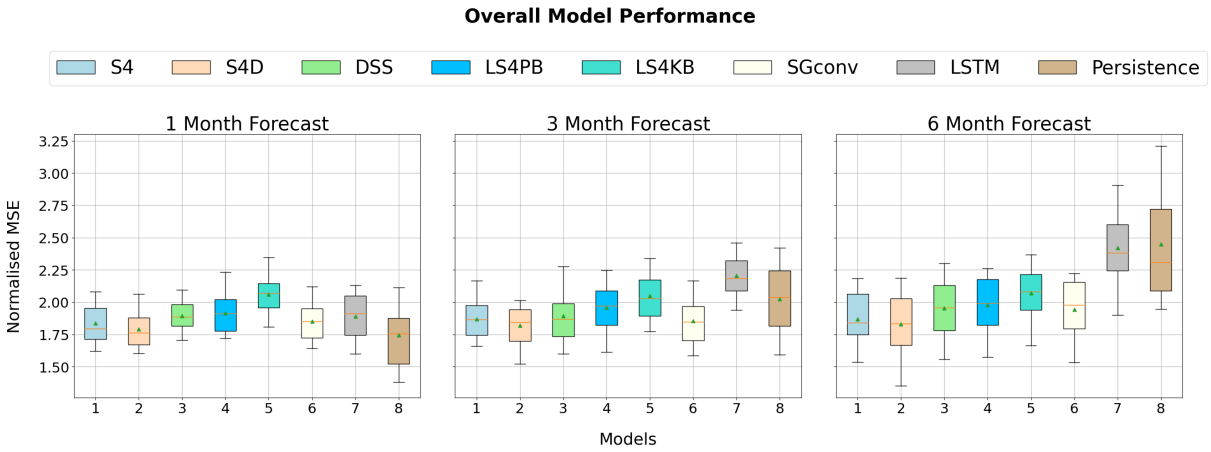

Figure 5.4: Boxplots of the normalised MSE of the 1, 2 and 3 month predictions for all the models. The values plotted are the average performance of each sample in the test set, meaning I averaged across states and features. Each boxplot encompasses six measurements, the bottom and top of the box are the first and third quartiles, respectively, the whiskers are the 5 and 95 percentiles, the orange line is the median and the green triangle is the average.

In sum, the models perform differently according to both the state of origin of the data, the periodicity of the features, and the length of the forecast. More complex models, such as S4 and S4D, tend to outperform the LSTM and Persistence baselines in forecasts over longer temporal windows, whereas these baselines are better in the short 1 month forecast. The S4D seems to be better suited to deal

with less periodic causes of death, and the S4 model is better at more periodical CODs. However, as we have seen from Figure A.1, the performance across the state of the art models is rather similar. Examples of the forecasts can be found in Figure 5.5.

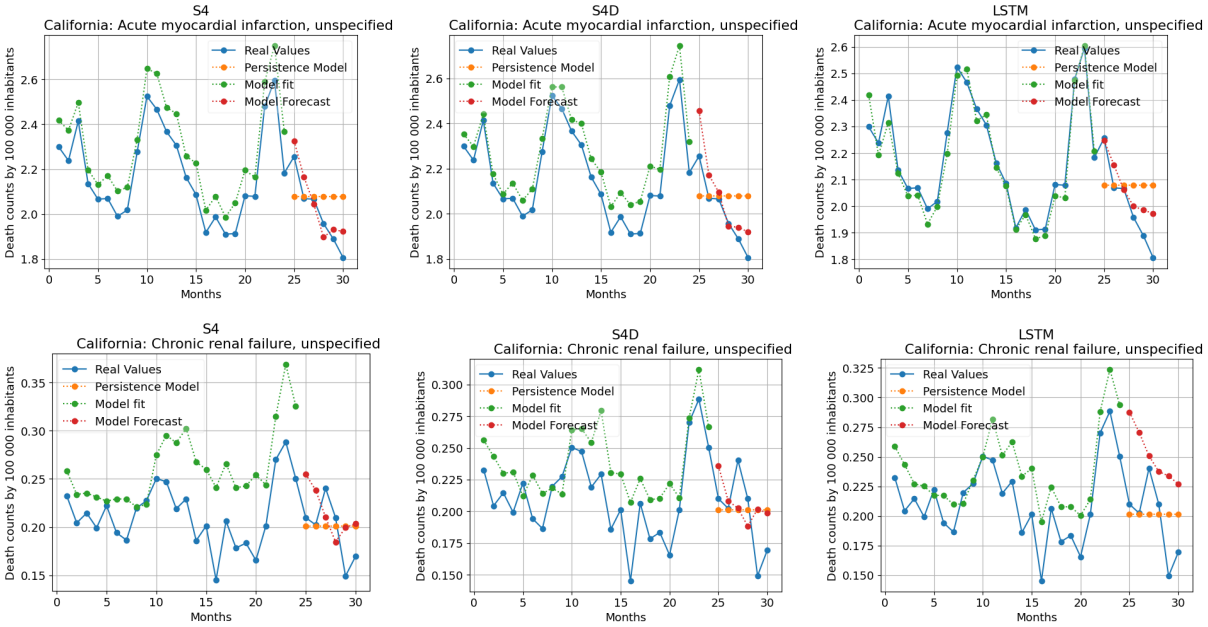

Figure 5.5: Examples of the forecasts by the S4, S4D, and LSTM models in California for the acute myocardial infarction and chronic renal failure CODs, respectively, from the periodical and non-periodical feature groups. The graphs present the real values of of the COD along with the models prediction, using the 24 months context. The graphs also show the persistence model forecasts.

5.3 Feature Variation Experiments

This section aims to understand the impact that additional features besides the death rates have on the performance of the models. Two additional types of features were tested, the sinusoidal features, consisting of two sinusoidal functions with periods of 12 and 6 months, whose goal is to provide the models with a sense of seasonality. These features were used in the other experiments. The second type of features are the state of origin features, that consist of a one-hot encoding for the state of origin of the data, meaning that these represent 5 extra features. The models were trained with all the combinations of these extra features, meaning that the models were tested without any extra feature, with only the sinusoidal features, with only the state features, and with both the state and sinusoidal features. The models were trained 10 times with each combination of features and the MSE values for the 1 month, 3 month, and 6 month forecasts were collected for each iteration.

The results are summarized in Figure A.2, where there are 3 models presented, namely the S4, the S4D, and the LSTM, the complete version of this graph can be found in Appendix A. The S4 showed

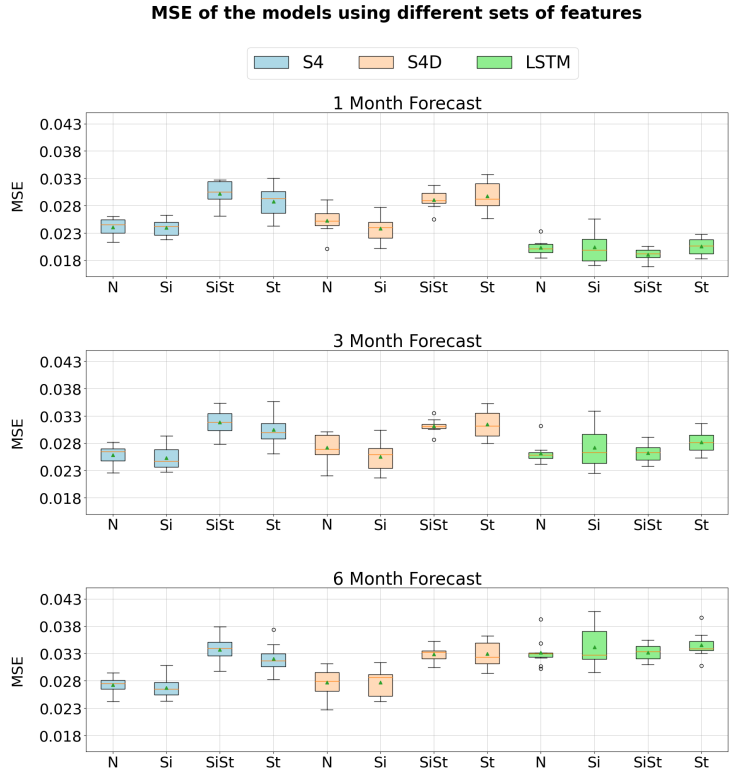

Figure 5.6: Results from the feature variation experiment. Each model was trained 10 times with each combination of features, displayed in the x-axis, no additional features (N) only the sinusoidal features (Si) both the sinusoidal and state features (SiSt), and only the state features (St). For each set of experiments we collected the average MSE of the 6 month forecast, and the results across the experiments are organised into boxplots. The top and bottom of the box represent the first and third quartile, respectively. The orange line is the median, and the whiskers represent the quartiles, plus or minus the inter-quartile range.

no improvement when provided with the sinusoidal features. However, the addition of the state features significantly worsened the performance. This behaviour is shared with the DSS and LS4 PB models. On the other hand, the S4D model showed improvements when provided with the sinusoidal features, but its performance also worsened significantly when provided with the state features, this was also the case for the SGconv and LS4 KB models. The LSTM model was the only approach that somewhat maintained its performance regardless of the features provided. The most discernible change was an increase in the average distribution when it comes to the sinusoidal features.

Overall, the sinusoidal features improved the performance of some of the models and did not impact the performance of the others. The features encoding the state of origin significantly worsened the results of all the models, except for the LSTM. This may be because the models have a greater interest in the seasonality information provided by the sinusoidal features, rather than the location of origin of the data, meaning that the features behave similarly enough across the states in order for the models not to require this information.

6

Conclusions and Future Work

Contents

Current research in healthcare and mortality forecasting has mostly focused on the usage of classic statistical methods, or on the use of simple machine learning models such as the LSTM. However, in the field of machine learning there have been a lot of significant developments when it comes to the creation of novel models, such as state space models, like the S4. The goal of this thesis was to determine if applying these novel models to the field of healthcare forecasting, and specially to mortality forecasting, would be beneficial. For this purpose a forecasting task was created using the United States mortality database, where the goal of the models is to receive a 2 year long context and provide a 6 month long forecast that would be evaluated in the 1, 3 and 6 month points.

This chapter provides the conclusions of the work done in the context of my M.Sc. research project and ponders over future work that may be developed. The chapter is divided into two sections: Section 6.1 presents the main takeaways from this thesis, while Section 6.2 discusses the implications of this thesis and possible research questions that may come of it will be discussed.

6.1 Contributions

This thesis set out to find if applying state of the art machine learning models to a healthcare forecasting task would be an improvement over classical methods. We compared 6 state of the art models with the LSTM and the persistence model as baselines. The results showed that the performance between the state of the art models was rather similar. The baseline models were comparable with the state of the art in the 1 month forecast, and the Persistence model even managed to outperform all the other models in this task. However, these two models were subpar when we extended the forecasting window to 3 and 6 months. Introducing models such as the S4 and the S4D in healthcare forecasting may prove to be beneficial, specially when we intend to do longer forecasts.

We also observed the models performance on two sets of CODs, namely the periodical and the non-periodical. We saw that some models are better suited than others for dealing with the non-periodical features, this is the case with the S4D. The same is true for the periodical features, where the S4 has the advantage. However, every model performed better in the periodical features rather than the non-periodical ones. The models were also tested with two additional sets of features, namely the sinusoidal features and the state features. The sinusoidal features proved to either improve or not significantly affect performance, while the state features mainly worsened the performance. This might indicate that the models value temporally sensitive information more rather than location information.

Finally, the creation of this task, alongside the datasets that were collected is a contribution on its own, since it constitutes an attempt to standardise a forecasting task on healthcare data. The same structure may be used to test new models and become a benchmark, similar to the LRA [1] when it comes to long range dependencies. The source code and data used in the experiments reported on this

dissertation were made available on GitHub[1].

6.2 Future Work

There are several improvements and additional work that can be performed as a continuation of this thesis. We have seen that the models that were tested have different performances according to the length of the forecast and the periodicity of the features. For example, the LSTM was very proficient in the 1 month forecast with the periodic features, while the S4 was better in the longer forecasts also with the periodic features. One way to take advantage of this difference in performance would be to use an ensemble architecture. We could use a weighted average ensemble, like Kaushik et al. [6], where the final prediction is a weighted average over each individual prediction of all models. The weights of each prediction would be dependent on the models performance on that particular feature and forecast length. The problem now would be to find the adequate weights, we could first train each model on a given set, and then, with the models' parameters locked in and using a different validation set, we can adjust the weights. This way, the weights would be adjusted to give the best combined forecast, emphasising a given model according to the features and forecast length where it performs the best.

Another approach towards an overall better performing model could involve the use of a specialised loss function. The loss function that was used in this work, namely the MSE, gives every single prediction the same importance. A specialised loss function could give more importance to features and forecast lengths where the model typically performs worse. For example, the S4D has a hard time with short forecasts, and thus the loss function would give more importance to the short forecast, in an effort to improve the results. Now the problem lies in how much importance we give to each forecast/feature. A way to deal with this would be through a grid search, although this may prove to be very time and resource consuming.

Another possible approach is to introduce additional sets of features, like Piccialli et al. [7] and Mathonsi et al. [25]. These additional features, like air quality and meteorological data, would provide the models with information that might affect the CODs. For example, if we know that low temperatures increase the cases of respiratory illnesses, then if the model sees low temperatures it may predict higher death rates for respiratory illnesses than it would without this feature. These kinds of features present a challenge, in that low temperatures do not cause an immediate increase in the number of deaths by respiratory illnesses, and their effect may be delayed by some time. In order to address this issue, we could perform some feature analysis, perhaps using causality and correlation analysis like a Granger causality test [40] to determine if the additional features Granger-causes any given COD behaviour and with what lag. This way, we can find out which CODs are most affected by these features, as well as the

[1] https://github.com/DiogoVF/Mortality-Forecasting

lag that these features have with those CODs. Having that, we can provide the models with the normal CODs and the lagged versions of the additional features, according to the correlation analysis. There may be the issue where different CODs have different lag values for an additional feature, this may be solved by providing the same additional feature with different lag values. As we are not interested in the prediction of these features, we could remove them from the computation of the training loss, like we did with the sinusoidal features.

This thesis used the publicly available United States mortality database [11] to collect monthly mortality rates at a state level. This corresponds to a high-level forecasting, since the features correspond to relatively large geographical and temporal sets. Having access to other datasets we could explore the models performance in a more local-forecasting, for example, considering hospital level data in a daily basis. This may allow the models to capture trends and behaviours that are more specific to a given hospital, or that depend on smaller time frames, exploring information that may be lost in the state and monthly levels. The forecasting task itself would be the same, and the change would be mainly in the data used. There can be some drawbacks from this approach, as discussed in Chapter 4, like an increase of the signal-to-noise ratio. Since we are using data that represents fewer people, a single outlier has a greater impact on the ability of the data to provide relevant trends and behaviours. Nevertheless, it would be a good opportunity to investigate if models such as the S4D, that performed well on the non-periodical CODs, some of which are characterised by having high noise and being the least prevalent, would also perform well in this new setting. The results from this study could work as a stepping stone for the implementation of a much more complex hierarchical forecast, like the one present in the M5 forecasting competition [12], where we would have to forecast different levels (eg., hospital, regional, and state). This can be dealt with in a bottom-up approach, where we first forecast the bottom levels and aggregate those forecasts in order to generate the larger regional and state level forecasts. This approach may require adapting the current implementations of the models and also the problem of how to aggregate the forecasts would also need to be addressed. Another issue with this approach is obtaining the necessary data, which can be challenging for several reasons, from privacy concerns to data availability.

Bibliography

[1] Y. Tay, M. Dehghani, S. Abnar, Y. Shen, D. Bahri, P. Pham, J. Rao, L. Yang, S. Ruder, and D. Metzler, "Long range arena: A benchmark for efficient transformers," 2020. [Online]. Available: https://doi.org/10.48550/ARXIV.2011.04006

[2] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," 2021. [Online]. Available: https://doi.org/10.48550/ARXIV.2111.00396

[3] A. Gu, A. Gupta, K. Goel, and C. Ré, "On the parameterization and initialization of diagonal state space models," 2022. [Online]. Available: https://doi.org/10.48550/ARXIV.2206.11893

[4] A. Gupta, A. Gu, and J. Berant, "Diagonal state spaces are as effective as structured state spaces," 2022. [Online]. Available: https://doi.org/10.48550/ARXIV.2203.14343

[5] Y. Li, T. Cai, Y. Zhang, D. Chen, and D. Dey, "What makes convolutional models great on long sequence modeling?" 2022. [Online]. Available: https://doi.org/10.48550/ARXIV.2210.09298

[6] S. Kaushik, A. Choudhury, P. K. Sheron, N. Dasgupta, S. Natarajan, L. A. Pickett, and V. Dutt, "Ai in healthcare: Time-series forecasting using statistical, neural, and ensemble architectures," *Frontiers in Big Data*, vol. 3, 2020. [Online]. Available: https://doi.org/10.3389/fdata.2020.00004

[7] F. Piccialli, F. Giampaolo, E. Prezioso, D. Camacho, and G. Acampora, "Artificial intelligence and healthcare: Forecasting of medical bookings through multi-source time-series fusion," *Information Fusion*, vol. 74, pp. 1–16, 2021. [Online]. Available: https://doi.org/10.1016/j.inffus.2021.03.004

[8] N. Kumar and S. Susan, "Covid-19 pandemic prediction using time series forecasting models," in *International Conference on Computing, Communication and Networking Technologies*, 2020, pp. 1–7.

[9] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[10] G. Box and G. Jenkins, *Time Series Analysis: Forecasting and Control*, ser. Holden-Day series in time series analysis and digital processing. Holden-Day, 1970.

[11] Centers for Disease Control and Prevention, National Center for Health Statistics, “Underlying cause of death 1999-2020,” 2021. [Online]. Available: http://wonder.cdc.gov/ucd-icd10.html

[12] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, “M5 accuracy competition: Results, findings, and conclusions,” *International Journal of Forecasting*, 2022. [Online]. Available: https://doi.org/10.1016/j.ijforecast.2021.11.013

[13] V. Kotu and B. Deshpande, “Chapter 12 - time series forecasting,” in *Data Science (Second Edition)*, second edition ed., V. Kotu and B. Deshpande, Eds. Morgan Kaufmann, 2019, pp. 395–445. [Online]. Available: https://doi.org/10.1016/B978-0-12-814761-0.00012-5

[14] R. J. Hyndman, A. B. Koehler, R. D. Snyder, and S. Grose, “A state space framework for automatic forecasting using exponential smoothing methods,” *International Journal of Forecasting*, vol. 18, no. 3, pp. 439–454, 2002. [Online]. Available: https://doi.org/10.1016/S0169-2070(01)00110-8

[15] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain.” *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958. [Online]. Available: http://dx.doi.org/10.1037/h0042519

[16] G. Piccinini, “107C5The First Computational Theory of Cognition: McCulloch and Pitts’s “A Logical Calculus of the Ideas Immanent in Nervous Activity”,” in *Neurocognitive Mechanisms: Explaining Biological Cognition*. Oxford University Press, 11 2020. [Online]. Available: https://doi.org/10.1093/oso/9780198866282.003.0006

[17] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015. [Online]. Available: http://neuralnetworksanddeeplearning.com/

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[19] D. Kalita, “A brief overview of recurrent neural networks (rnn),” Tech. Rep., 2023. [Online]. Available: https://www.analyticsvidhya.com/blog/2022/03/a-brief-overview-of-recurrent-neural-networks-rnn/

[20] C. Olah, “Understanding lstm networks,” Tech. Rep., 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[21] A. Gu, I. Johnson, K. Goel, K. Saab, T. Dao, A. Rudra, and C. Ré, “Combining recurrent, convolutional, and continuous-time models with linear state-space layers,” 2021. [Online]. Available: https://doi.org/10.48550/ARXIV.2110.13985

[22] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Re, “Hippo: Recurrent memory with optimal polynomial projections,” 2020. [Online]. Available: https://doi.org/10.48550/ARXIV.2008.07669

[23] T. Zhou, Z. Ma, X. wang, Q. Wen, L. Sun, T. Yao, W. Yin, and R. Jin, "Film: Frequency improved legendre memory model for long-term time series forecasting," 2022. [Online]. Available: https://doi.org/10.48550/ARXIV.2205.08897

[24] F. Perla, R. Richman, S. Scognamiglio, and M. V. Wüthrich, "Time-series forecasting of mortality rates using deep learning," *Scandinavian Actuarial Journal*, vol. 2021, no. 7, pp. 572–598, 2021. [Online]. Available: https://doi.org/10.1080/03461238.2020.1867232

[25] T. Mathonsi and T. L. van Zyl, "A statistics and deep learning hybrid method for multivariate time series forecasting and mortality modeling," 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2112.08618

[26] D. Whitley, "A genetic algorithm tutorial - statistics and computing." [Online]. Available: https://doi.org/10.1007/BF00175354

[27] S. J. Taylor and B. Letham, "Forecasting at scale," *The American Statistician*, vol. 72, no. 1, pp. 37–45, 2018. [Online]. Available: https://doi.org/10.1080/00031305.2017.1380080

[28] C. W. J. Granger, "Investigating causal relations by econometric models and cross-spectral methods," *Econometrica*, vol. 37, no. 3, pp. 424–438, 1969. [Online]. Available: https://doi.org/10.2307/1912791

[29] G. Sugihara, R. May, H. Ye, C. hao Hsieh, E. Deyle, M. Fogarty, and S. Munch, "Detecting causality in complex ecosystems," *Science*, vol. 338, no. 6106, pp. 496–500, 2012. [Online]. Available: https://doi.org/10.1126/science.1227079

[30] J. Runge, "Discovering contemporaneous and lagged causal relations in autocorrelated nonlinear time series datasets," in *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2020. [Online]. Available: https://doi.org/10.48550/arXiv.2003.03685

[31] R. D. Lee and L. R. Carter, "Modeling and forecasting u. s. mortality," *Journal of the American Statistical Association*, vol. 87, no. 419, pp. 659–671, 1992. [Online]. Available: https://doi.org/10.2307/2290201

[32] D. Salinas, V. Flunkert, and J. Gasthaus, "Deepar: Probabilistic forecasting with autoregressive recurrent networks," 2017. [Online]. Available: https://doi.org/10.48550/ARXIV.1704.04110

[33] S. Smyl, "A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting," *International Journal of Forecasting*, vol. 36, no. 1, pp. 75–85, 2020, m4 Competition. [Online]. Available: https://doi.org/10.1016/j.ijforecast.2019.03.017

[34] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, “The m4 competition: 100,000 time series and 61 forecasting methods,” *International Journal of Forecasting*, vol. 36, no. 1, pp. 54–74, 2020, m4 Competition. [Online]. Available: https://doi.org/10.1016/j.ijforecast.2019.04.014

[35] D. Salinas, V. Flunkert, and J. Gasthaus, “Deepar: Probabilistic forecasting with autoregressive recurrent networks,” 2019. [Online]. Available: https://doi.org/10.48550/arXiv.1704.04110

[36] L. Greengard and V. Rokhlin, “A fast algorithm for particle simulations,” *Journal of Computational Physics*, vol. 73, no. 2, pp. 325–348, 1987. [Online]. Available: https://doi.org/10.1016/0021-9991(87)90140-9

[37] R. Hasani, M. Lechner, T.-H. Wang, M. Chahine, A. Amini, and D. Rus, “Liquid structural state-space models,” 2022. [Online]. Available: https://doi.org/10.48550/ARXIV.2209.12951

[38] P. Welch, “The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms,” *IEEE Transactions on Audio and Electroacoustics*, vol. 15, no. 2, pp. 70–73, 1967. [Online]. Available: https://doi.org/10.1109/TAU.1967.1161901

[39] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” 2019. [Online]. Available: https://doi.org/10.48550/arXiv.1711.0510

[40] A. Shojaie and E. B. Fox, “Granger causality: A review and recent advances,” 2021. [Online]. Available: https://doi.org/10.1146/annurev-statistics-040120-010930

A

Model Comparison: Complete Tables and Graphs

Table A.1: Comparison of all the tested models, with their performance grouped by feature. The models were trained a total of 20 times with their respective optimal hyper-parameters and the best performer of these models in the 6 month long forecast was selected. Each best performer was evaluated in three tasks, corresponding to one month, three month and six month forecasts using both the MSE and the MAE metrics. The results were organised by feature, meaning that the results were averaged across the state of origin. The features were organised into periodical features (P) and Non-Periodical features (NP). Three features from each of these categories were selected as examples to be displayed along with the average results from the respective groups. Highlighted in **bold** and underlined are the best and second best results, respectively, on the given task, metric, and features, across all the 8 models tested. The periodical features are AHD, BLMN,AMI. The non-periodical features areGH,CIHD,CRF.

	S4						S4D						DSS					
	1		3		6		1		3		6		1		3		6	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Atherosclerotic Heart Disease	0.3602	0.4303	0.3617	0.4386	0.4440	0.4848	0.3655	0.4412	0.3875	0.4690	0.4256	0.4854	0.3724	0.4125	0.4258	0.4464	0.4650	0.4726
Bronchus or Lung Malignant Neoplasm	**0.2825**	**0.4218**	**0.2601**	**0.3950**	0.3069	0.4326	0.5901	0.6618	0.5227	0.6223	0.5221	0.6095	0.4397	0.5406	0.5808	0.6464	0.6122	0.6696
Acute myocardial infarction, unspecified	**0.2969**	**0.4268**	**0.3242**	**0.4485**	**0.3677**	**0.4752**	0.5113	0.5875	0.4939	0.5565	0.5849	0.6132	0.7013	0.7060	0.6104	0.6390	0.6425	0.6482
Average Periodical features	0.7957	0.6530	0.8154	**0.6572**	**0.8613**	**0.6775**	0.8975	0.7267	0.9622	0.7480	1.0473	0.7841	0.7225	0.6409	**0.8124**	0.6857	0.9002	0.7218
Gastrointestinal Haemorrhage, Unspecified	2.1990	1.2372	2.3906	1.3039	1.9460	1.1894	0.7710	0.6390	0.7338	0.6155	0.6973	0.6250	0.7579	0.6617	0.7968	0.6997	0.9086	0.7665
Chronic Ischaemic Heart Disease, Unspecified	2.5838	1.3623	1.9069	1.1376	2.1470	1.2455	1.1084	0.8283	**0.8819**	**0.7256**	**0.8085**	**0.7295**	3.4331	1.6310	2.7464	1.4119	2.8123	1.4416
Chronic Renal Failure, Unspecified	**0.8635**	**0.7246**	**0.8813**	**0.7346**	**0.9756**	**0.7708**	0.9302	0.7290	1.0640	0.7758	1.0863	0.7780	1.2328	0.8755	1.2215	0.8654	1.3462	0.9081
Average Non-Periodical Features	2.1264	1.1323	2.1497	1.1434	2.1010	1.1269	**1.9433**	**1.0658**	**1.9729**	**1.0809**	**1.9645**	**1.0827**	2.3344	1.1786	2.2911	1.1728	2.2967	1.1749
Average All Features	1.8354	1.0270	1.8682	1.0419	1.8696	1.0394	1.7922	1.0160	**1.8207**	**1.0271**	**1.8307**	**1.0358**	1.8941	1.0442	1.8925	1.0513	1.9538	1.0701

	LS4 KB						LS4 PB						SGconv					
	1		3		6		1		3		6		1		3		6	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Atherosclerotic Heart Disease	0.3286	0.4212	0.3488	0.4410	0.3899	0.4689	0.3188	0.4051	0.3328	0.4116	0.3946	0.4495	0.3182	0.4060	0.3850	0.4513	0.4505	0.4770
Bronchus or Lung Malignant Neoplasm	0.4837	0.5685	0.4985	0.5807	0.5365	0.6017	0.3284	0.4524	0.3218	0.4465	**0.2834**	**0.4148**	0.4190	0.5167	0.3215	0.4346	0.4149	0.4829
Acute myocardial infarction, Unspecified	0.6083	0.6285	0.7151	0.6926	0.6882	0.6708	0.4311	0.5420	0.4030	0.5165	0.4402	0.5321	0.6137	0.6352	0.6301	0.6521	0.7027	0.6842
Average Periodical features	0.7967	0.6803	0.8735	0.7145	0.9013	0.7265	0.9520	0.7526	0.9377	0.7462	0.9586	0.7586	0.8929	0.7193	1.0003	0.7624	1.0344	0.7777
Gastrointestinal Haemorrhage, Unspecified	2.1156	1.2687	1.9496	1.1874	1.8842	1.1650	1.2892	0.8871	0.9801	0.7268	1.0157	0.7514	1.2871	0.8948	1.5044	0.9983	1.6310	1.0876
Chronic Ischaemic Heart Disease, Unspecified	2.1269	1.2242	2.3649	1.2992	1.8708	1.1605	1.1992	0.8512	1.0938	0.8354	1.0974	0.8749	1.4350	0.9653	1.2066	0.8789	0.9094	0.7588
Chronic Renal Failure, Unspecified	1.3565	0.9213	1.2853	0.9455	1.4852	1.0092	1.3958	0.9461	1.5636	1.0010	1.5565	1.0016	1.1087	0.7940	1.1720	0.8460	1.2727	0.8473
Average Non-Periodical Features	2.3405	1.1664	2.3744	1.1883	2.3203	1.1703	2.4202	1.1883	2.4524	1.2007	2.4519	1.2020	2.0123	1.0892	2.0087	1.0952	2.0932	1.1154
Average All Features	1.9162	1.0459	1.9562	1.0621	1.9781	1.0659	2.0592	1.0881	2.0478	1.0869	2.0691	1.0964	1.8506	1.0314	1.8536	1.0390	1.9431	1.0637

	LSTM						Percistence					
	1		3		6		1		3		6	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Atherosclerotic Heart Disease	**0.2495**	**0.3655**	**0.2070**	**0.3452**	**0.2373**	**0.3756**	0.4203	0.4745	0.8210	0.6787	1.2252	0.8590
Bronchus or Lung Malignant Neoplasm	0.4579	0.5257	0.6137	0.6200	1.0582	0.8339	0.5947	0.6016	0.5344	0.5746	0.6440	0.6278
Acute myocardial infarction, Unspecified	0.6192	0.6213	0.9392	0.7960	1.5104	1.0087	0.5664	0.5864	1.1859	0.8963	1.8153	1.0827
Average Periodical features	**0.6794**	**0.6155**	0.8794	0.7154	1.1670	0.8281	0.7639	0.6441	1.3790	0.8773	1.9115	1.0525
Gastrointestinal Haemorrhage, Unspecified	**0.7454**	**0.6102**	**0.6738**	**0.5725**	**0.6109**	**0.5457**	0.9738	0.6772	1.0506	0.7415	1.2036	0.7775
Chronic Ischaemic Heart Disease, Unspecified	1.1228	0.8366	1.6752	1.0142	1.9188	1.1184	**1.0211**	**0.7812**	1.3054	0.9125	1.4963	0.9621
Chronic Renal Failure, Unspecified	1.4305	0.9612	1.5008	0.9792	1.6632	1.0397	1.2726	0.8903	1.8049	1.0070	2.0237	1.0922
Average Non-Periodical Features	2.4395	1.2030	2.8047	1.2912	2.9551	1.3275	2.1933	1.1234	2.2947	1.1480	2.6557	1.2316
Average All Features	1.8900	1.0264	2.2064	1.1187	2.4218	1.1774	**1.7452**	**0.9730**	2.0257	1.0679	2.4476	1.1815

Table A.2: Comparison of all the tested models on the state of origin of the data. The models were trained a total of 20 times with their respective optimal hyper-parameters, and the best performer of these models in the 6 month long forecast was selected. Each best performer was evaluated in three tasks,namely one month, three month and six month forecasts, using both the MSE and theMAE metrics. Highlighted in **bold** and underlined are the best and second best results, respectively, on the given task and metric across all the 8 models tested.

	S4						S4D						DSS					
	1		3		6		1		3		6		1		3		6	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
California	**0.9725**	**0.7295**	**0.9813**	**0.7408**	**0.9711**	**0.7393**	1.0074	0.7844	1.0453	0.7988	1.0696	0.8128	1.1697	0.8027	1.2042	0.8202	1.2514	0.8440
Texas	1.5131	0.9518	1.5296	0.9532	**1.4654**	**0.9273**	1.6065	0.9746	1.6579	0.9862	1.6428	0.9864	1.7044	1.0032	1.6625	0.9979	1.6579	1.0031
Florida	2.1984	1.1784	2.2330	1.1897	2.1124	1.1647	2.0992	1.1061	2.0736	1.1065	**1.9617**	**1.0864**	2.2805	1.1660	2.2375	1.1602	2.2197	1.1541
New York	1.7662	0.9965	1.8027	1.0207	1.9762	1.0670	1.8360	1.0201	1.9783	1.0583	2.1017	1.0950	1.7335	1.0090	1.7598	1.0229	1.9589	1.0756
Pennsylvania	2.7268	1.2790	2.7945	1.3052	2.8229	1.2987	2.4120	1.1950	**2.3484**	**1.1855**	**2.3779**	**1.1983**	2.5823	1.2399	2.5988	1.2555	2.6810	1.2738
Average	1.8354	1.0270	1.8682	1.0419	1.8696	1.0394	1.7922	1.0160	**1.8207**	**1.0271**	**1.8307**	**1.0358**	1.8941	1.0442	1.8925	1.0513	1.9538	1.0701

	LS4 KB						LS4 PB						SGconv					
	1		3		6		1		3		6		1		3		6	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
California	1.1237	0.7910	1.1096	0.7994	1.1297	0.7959	1.2476	0.8624	1.1666	0.8362	1.1427	0.8300	1.1067	0.7932	1.0757	0.7869	1.1564	0.8030
Texas	1.9429	1.0544	1.9258	1.0509	1.9031	1.0382	1.8483	1.0362	1.8872	1.0423	1.8841	1.0482	1.4875	0.9603	1.5149	0.9678	1.5597	0.9920
Florida	2.4093	1.1901	2.3931	1.1974	2.3896	1.2018	3.0459	1.3623	2.9016	1.3299	2.7443	1.2993	2.3226	1.1615	2.3353	1.1846	2.4486	1.2137
New York	1.6694	**0.9702**	**1.7440**	**0.9909**	**1.7995**	**1.0075**	1.7277	0.9789	1.8017	1.0099	1.9168	1.0431	1.9829	1.0565	1.9675	1.0520	2.1025	1.0929
Pennsylvania	2.4359	1.2239	2.6083	1.2718	2.6685	1.2862	2.4265	1.2007	2.4821	1.2163	2.6577	1.2613	**2.3534**	**1.1858**	2.3745	1.2037	2.4482	1.2168
Average	1.9162	1.0459	1.9562	1.0621	1.9781	1.0659	2.0592	1.0881	2.0478	1.0869	2.0691	1.0964	1.8506	1.0314	1.8536	1.0390	1.9431	1.0637

	LSTM						Percistence					
	1		3		6		1		3		6	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
California	1.5264	0.9133	1.8421	1.0103	1.8250	1.0037	0.9752	0.7339	1.3315	0.8723	1.6444	0.9628
Texas	1.3211	0.8619	1.5924	0.9502	1.7115	0.9997	**1.1912**	**0.8130**	**1.5002**	**0.9156**	1.7647	0.9899
Florida	2.0664	1.0967	2.5928	1.2303	3.2290	1.3822	**1.8790**	**1.0205**	**1.8711**	**1.0518**	2.1920	1.1312
New York	1.7889	0.9904	2.0258	1.0700	2.2295	1.1277	**1.6681**	0.9759	1.9887	1.0715	2.3653	1.1899
Pennsylvania	2.7469	1.2698	2.9790	1.3324	3.1140	1.3735	3.0123	1.3217	3.4371	1.4283	4.2717	1.6339
Average	1.8900	1.0264	2.2064	1.1187	2.4218	1.1774	**1.7452**	**0.9730**	2.0257	1.0679	2.4476	1.1815

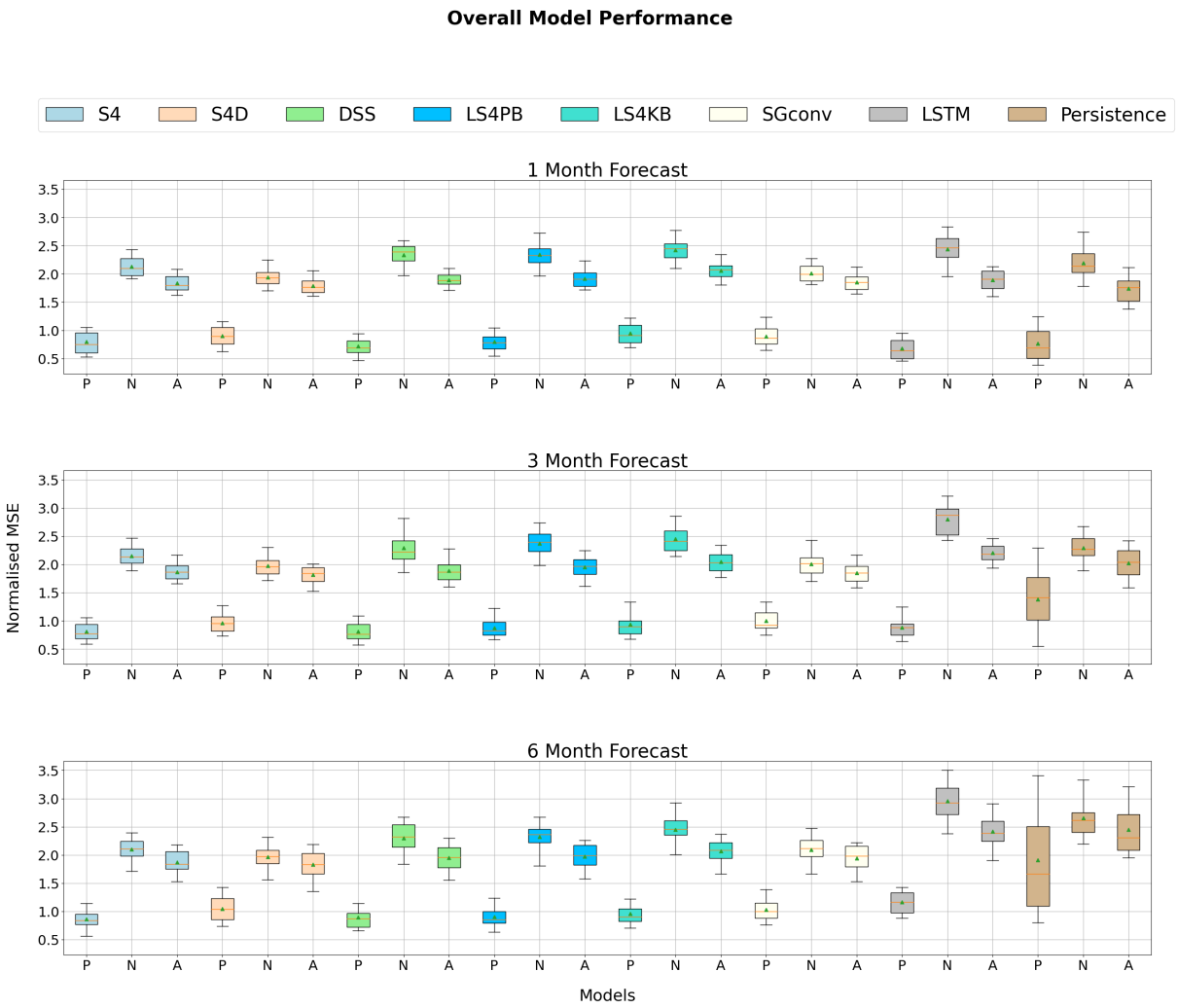

Figure A.1: Boxplots of the normalised MSE of the 1, 2 and 3 month predictions for all the models. The values plotted are the average performance of each sample in the test set, meaning I averaged across States and features. On the x axis is the category of features that were used, the periodical features, P, the non-periodical features, N, and all the features, A. Each boxplot encompasses six measurements, the bottom and top of the box are the first and third quartiles, respectively, the whiskers are the 5 and 95 percentiles, the orange line is the median and the green triangle is the average.

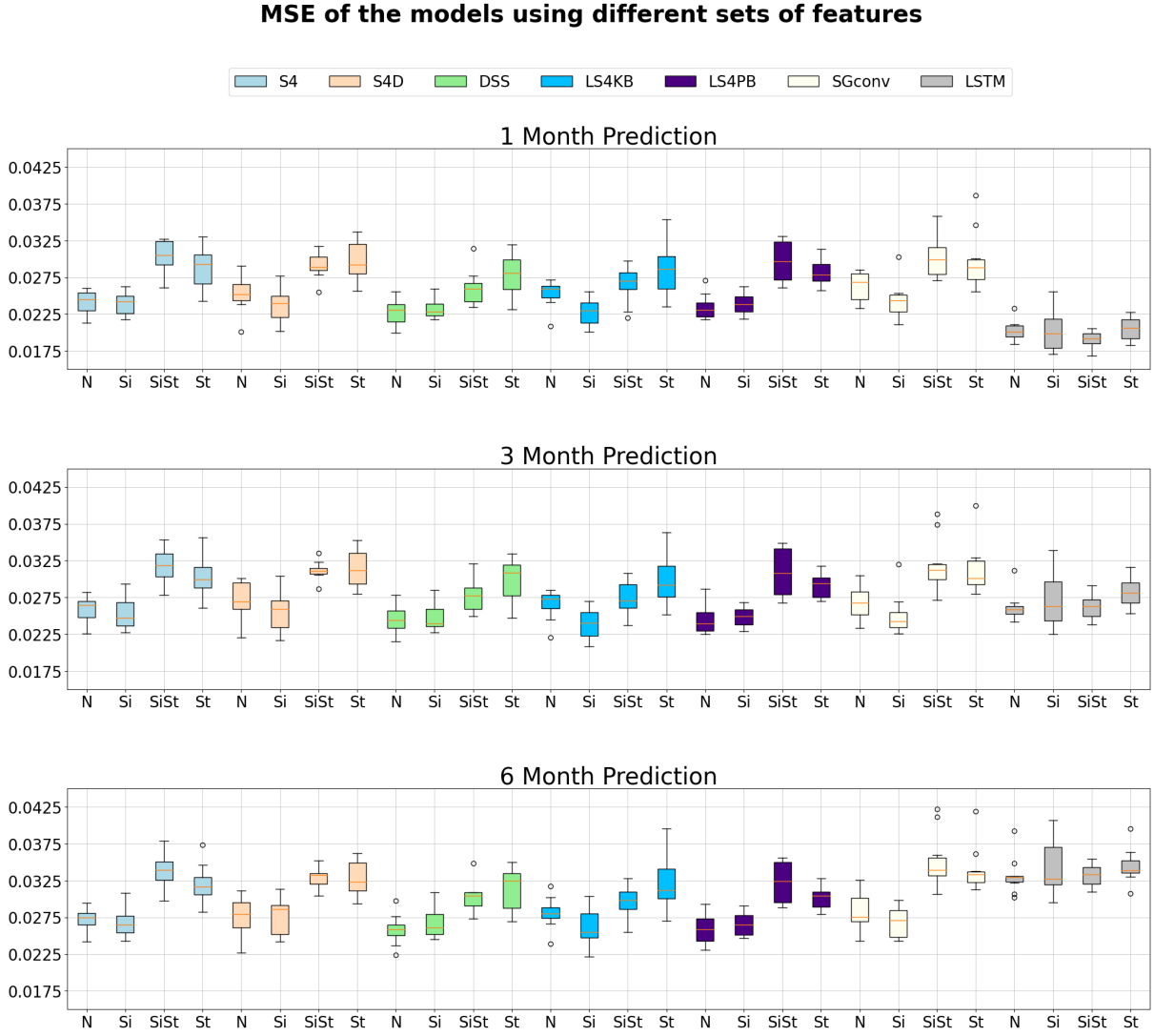

Figure A.2: Results from the feature variation experiment. Each model was trained 10 times with each combination of features, displayed in the x-axis, no additional features, N, only the sinusoidal features, Si, both the sinusoidal and State features, SiSt, and only the State features, St. For each iteration we have the average MSE of the 6 month forecast, the results across the iterations are organised into boxplots, the top and bottom of the box represent the first and third quartile, respectively, the orange line is the median and the whiskers are represent the quartiles plus or minus the inter-quartile range.