Karlsruhe Institute of Technology

Department of Economics and Management

Institute of Economics (ECON)

Statistical Methods and Econometrics

Prof. Dr. Melanie Schienle

Bachelor's Thesis

DeepAR for Infectious Disease Forecasting

Written by **Kai Reffert**

Matr. No. **2279216**

Industrial Engineering and Management

First Supervisor: **Prof. Dr. Melanie Schienle**

Second Supervisor: **Prof. Dr. Wolf-Dieter Heller**

15.June 2023

I hereby confirm truthfully that I have authored this bachelor's thesis independently and without the use of source material and aids other than those stated, that I have marked all passages literally or textually adapted from other sources as such and that I have respected the statutes of the Karlsruhe Institute of Technology (KIT) for ensuring good scientific practice.

Karlsruhe, 14.06.2023

.....

(Kai Reffert)

Abstract

The annual threat of the influenza disease is prominent to this day. Thus, epidemic forecasts are of greater importance to public health organisations, as they increase situational awareness and provide time to set preventive actions in place. Probabilistic forecasts further quantify the uncertainty, which is especially important in decision-making and risk management. Since epidemiological forecasting often produces many related time series, we propose that global models which estimate model-parameters jointly based on every time series could be beneficial for this task. In addition, deep learning-based models have displayed great performances on general time series tasks, thus we introduce the application of the global DeepAR model to probabilistic influenza disease prediction in German districts. Furthermore, we summarise the theoretical framework of the DeepAR and rivalling models, i.e. an endemic-epidemic modelling approach and a feedforward neural network. By comparing the DeepAR model to the two other models, we try to assess the suitability of it to this task. In summary, we conclude that the DeepAR model is beneficial for our task and that it outperforms its rivals.

Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviations

COVID-19	Coronavirus Disease 2019
LK	Landkreis (rural district)
SK	Stadtkreis (urban district)
SIR	Susceptible-infected-recovered
CRPS	Continuous ranked probability score
QS	Quantile score
IS	Interval score
PI	Prediction interval
WIS	Weighted interval score
DL	Deep learning
NN	Neural network
ANN	Artificial neural network
DNN	Deep neural network
FNN	Feedforward neural network
MLP	Multilayer perceptron
CNN	Convolutional neural network
RNN	Recurrent neural network
ERNN	Elman recurrent neural network
LSTM	Long-short-term memory
GRU	Gated recurrent unit
GluonTS	Gluon Time Series Toolkit
DeepAR	Deep autoregressive (network)

1 Introduction

For many years the infectious disease of influenza has immensely endangered humans. In fact, in 1918 the Spanish flu, corresponding to one of the most devastating epidemic outbreaks in modern history, evolved due to the spread of the A(H1N1) influenza virus (Trilla et al., 2008). Furthermore, the estimated number of deaths due to the Spanish flu range from 20 to over 50 million (Johnson and Mueller, 2002; Patterson and Pyle, 1991; Taubenberger and Morens, 2006). Almost a century later a descendant of the initial A(H1N1) influenza disease caused the swine flu pandemic during the years of 2009 and 2010 (Scalera and Mossad, 2009). Today, the World Health Organisation estimates that annually occurring outbreaks of influenza are responsible for approximately 3 to 5 million cases of severe illness and 290000 to 650000 deaths (WHO, 2023). Overall, the impact reduction of seasonal epidemics and pandemics is of greater importance to public health authorities (Achrekar et al., 2012). Furthermore, Longini et al. (2005) and Ferguson et al. (2005) showed that epidemics may be preventively contained if an early detection system is in place. Thus, one of the key challenges in public health is to continuously monitor and predict the development of influenza in the population, as this presents the public health authorities with time to set necessary policies and resources in place (Jung et al., 2022).

In the past mechanistic models were often implemented to model the continuous development of disease infections over time. Moreover, models of this class, e.g. the chain-binomial or the susceptible-infected-recovered (SIR) model (Daley and Gani, 2005; Andersson and Britton, 2012), try to express the spread on an individual level (Paul et al., 2008). Thus, mechanistic models require the availability of the number of infected and susceptible individuals at every time point, however they can be approximated by a branching process if information about the susceptible population is missing (Andersson and Britton, 2012). In a surveillance setting, this is often the case, as data of the susceptible population is rarely provided. Therefore, Held et al. (2005) have introduced a model that is based on a multivariate branching process with immigration. Furthermore, this model was often applied to human influenza surveillance data, for example by Paul et al. (2008), Paul and Held (2011) as well as Geilhufe et al. (2014). Hence, we decided to incorporate a version of this model as our baseline model.

Following the general increase in popularity of deep learning (DL) models, growing numbers of DL models were incorporated for epidemiological prediction tasks, e.g. long-short-term memory (LSTM) networks were implemented to forecast COVID-19 outbreaks in Saudi Arabia (Elsheikh et al., 2021) as well as in Russia, Peru and Iran (Wang et al., 2020b). Initially, DL originated from Hinton et al. (2006), who first illustrated that deep neural

networks (DNNs), i.e. neural networks (NNs) with many layers, can be trained with the correct weight initialization. This breakthrough popularized DNNs, as they are capable of handling many complex non-linear problems. Simultaneously, many factors have further benefited the rise of DNNs, most notably the increase in computational capacity and the beginning of the Big Data era. Furthermore, higher computational capacities enable a faster implementation of even deeper models, thus further enhancing the NNs overall learning capacity (Lara-Benítez et al., 2021). In addition, the Big Data era characterizes the increasing data availability. In the context of time series, Big Data usually corresponds to a plethora of associated time series that come from a similar origin (Hewamalage et al., 2021b). Thus, more complex models, and especially complex data driven models like DNNs, benefit the most from the large amounts of data. Consequently, many novel DL models and DL architectures were introduced during the last years. In this thesis, we mainly focus on the application of the DeepAR model, introduced by Salinas et al. (2020), to the forecasting problem of influenza surveillance time series data in German districts. In addition, we compare it to a regular feedforward neural network (FNN) and the previously mentioned baseline approach.

There have been several approaches that assess a variety of models on the task of predicting influenza time series. At first mainly mechanistic and statistical models were implemented, e.g. an SIR model variation to explore epidemiological effects of influenza A viruses (Casagrandi et al., 2006) or autoregressive models fit on influenza surveillance data from China (Song et al., 2016) or the United States (Achrekar et al., 2012). More recently DL models were also implemented to forecast influenza surveillance time series. Soliman et al. (2019) compared traditional statistical models, such as the auto-regressive integrated moving average, least absolute shrinkage and selection operator regression, beta regression as well as non-parametric multivariate adaptive regression splines, with a deep learning model, namely the FNN, on seasonal influenza data of Dallas County. In a similar study Tapak et al. (2019) compared support vector machines, random forests and FNNs on the task of time series modelling and outbreak detection of influenza disease cases in Iran. Yang et al. (2020) implemented a LSTM model to deal with influenza time series data in Taiwan. Wu et al. (2018) developed a deep learning approach by combining a recurrent neural network (RNN) with a convolutional neural network (CNN). They compared their approach with traditional statistical models on regional influenza surveillance time series of the United States and Japan. In fact, Jung et al. (2022) incorporated the approach of Wu et al. (2018) into their analysis. Moreover, they additionally introduce a deep learning based self-attention model and compare it with other models on regional influenza data sets from the United States and prefectures in Japan.

As demonstrated above, previous studies also compare and assess various models on influenza data, but to the best of our knowledge the DeepAR model was not yet implemented in this context. Many open-source packages are designed to give an intuitive access to the latest DL models, e.g. GluonTS[1] (Alexandrov et al., 2019) or TensorFlow[2] (TensorFlow Developers, 2023; Abadi et al., 2016). Nevertheless, traditional statistical models are still considered simpler, more robust, efficient, and automatic, therefore they are typically implemented by non-experts (Hewamalage et al., 2021b). Thus, we first want to answer the question whether it is reasonable for non-experts to implement a basic variant of the DeepAR model to this epidemiological time series prediction task. Subsequently we also want to reflect the expert side by answering the question if it is beneficial to set up an optimized DeepAR model for the influenza time series prediction task. This requires more knowledge about the DeepAR model, as the model is adjusted in several steps to suit the task at hand. We want to answer both questions by evaluating and comparing the performances of the associated models with the baseline approach.

In subsequent sections, scalars and vectors are expressed as lowercase letters x, while matrices are written as uppercase letters X. Furthermore, a collection of N univariate time series is denoted by $\{z_{i,1:T_i}\}_{i=1}^{N} = \{z_{i,1}, \dots, z_{i,T_i}\}_{i=1}^{N}$, where $z_{i,t}$ is the value of the i-th time series at time point t. Additionally, the set corresponding to the values of all N time series during the time interval of $[t_1, t_2]$ is denoted as $Z_{t_1:t_2}$. Often the collection of time series is accompanied by covariates $\{X_{i,1:T_i}\}_{i=1}^{N}$, where the vector $x_{i,t}$ can be time-dependent or static.

This thesis is organised as follows. The next two sections cover the theoretical framework of probabilistic forecasting and their evaluation as well as local and global models respectively. Chapter 4 then describes the theoretical foundation of the models we are going to analyse. Here, we especially focus on NNs, RNNs and architectures that are relevant to understand the theory behind the DeepAR model. After that, Chapter 5 outlines the data set and Chapter 6 summarises our results. Lastly, Chapter 7 concludes this thesis.

[1] https://gluon-ts.mxnet.io
[2] https://www.tensorflow.org/

2 Probabilistic forecasts and their evaluation

This chapter briefly motivates and introduces the concept of probabilistic forecasts and typical evaluation metrics for probabilistic forecasts. After that we focus on interval and quantile forecasts and suited scoring measures for their predictions.

2.1 Probabilistic forecasting

Traditionally, point or single-valued forecasting methods have been among the most common forecasting techniques. However, these methods lack information about uncertainties of their predictions, which can be a major disadvantage when the forecasts are used in decision-making. Hence, Gneiting and Katzfuss (2014) state that forecasts should take on a probabilistic form, as this enables the modelling of uncertainties in forecasts. Additionally, Bracher et al. (2021) propose that this probabilistic behaviour particularly holds for epidemic forecasts. To this day, probabilistic forecasting commonly follows Gneiting et al. (2007) by training a model on the available data to maximize the sharpness of its predictive distribution subject to calibration. Furthermore, sharpness describes the concentration of predictive distributions, i.e. the sharper a forecast, the more informative it is. Calibration refers to the consistency between the predictions and the actual observations, as we ideally want the observations to be interchangeable with random draws from the predictive distribution. According to Gneiting and Katzfuss (2014), both concepts can be summarised into a single numerical score by proper scoring rules, thus enabling a collective assessment of calibration and sharpness. Generally, scoring rules $s(F, x)$ are defined as measures that summarise the predictive quality of forecasts by assigning a numeral score, when given the predictive distribution F and the materialization x (Gneiting et al. (2007)). In the following chapters, we consider scoring rules that we want to minimize, i.e. lower numerical scores represent better predictive performance. There is a plethora of scoring rules that lead to desirable assessments, nevertheless proper scoring rules, developed to be most common to evaluate probabilistic forecasts today (Machete, 2013). Proper scoring rules encourage forecasters to express their true beliefs, as there is no enticement for forecasters to report deviating forecasts from their honest opinion about the future (Winkler, 1996). One popular proper score for full predictive distributions is the logarithmic score, which originates from Good (1952). In a discrete setting, it can be described as

$$logS(F, x) = \log p(x), \tag{1}$$

where F is the forecast and x is the materialization, so that $p(x)$ represents the probability assigned to x by F (Bracher et al., 2021). The logarithmic score is a local score, therefore it relies on the predictive distribution exclusively through the observed outcome x, as opposed

to accounting for probabilities of other potential events that did not materialize (Bernardo, 1979; Gneiting and Raftery, 2007; Tyralis and Papacharalampous, 2022). In contrast to succeeding scoring rules, larger values of the logarithmic score are more desirable, hence a common modification is to select the negatively oriented $-\log p(x)$ instead. The positively oriented logarithmic score can deteriorate towards $-\infty$ if $p(x) = 0$ (Bracher et al., 2021), thus it heavily penalizes forecasts that assign (near) zero probability to the observed outcome. Hence the logarithmic score lacks robustness, whereas the continuous ranked probability score (CRPS) is often considered as a more robust proper scoring rule alternative (Gneiting et al., 2007). The CRPS can be defined as

$$CRPS(F, x) = \int_{-\infty}^{\infty} \{F(y) - I(y \geq x)\}^2 dy\,, \tag{2}$$

where F is a cumulative distribution function and $I(y \geq x)$ is the indicator function that is one if $y \geq x$ and zero otherwise (Bracher et al., 2021; Matheson and Winkler, 1976). In the setting of point forecasts the CRPS generalizes to the absolute error, thus the CRPS can be used to directly compare the performance of point and probabilistic forecasts (Gneiting and Raftery, 2007). Intuitively, the relation to the absolute error shows that the CRPS is a measure of distance between the predictive distribution and is on the same scale as the outcome variable. The following chapter deals with scoring rules for quantile and interval forecasts as the previously introduced scores cannot be applied to interval or quantile forecasts.

2.2 Scoring rules for quantile and interval forecasts

Although it is common in probabilistic forecasting to predict and output full probability distributions, they are difficult to store in full detail unless they are parametric. Furthermore, Bracher et al. (2021) argue that in short notice epidemic situations, such as the COVID-19 pandemic, the solution of defining a general binning scheme is difficult, as the outcome is highly variable over space and time. Thus, they suggest evaluating probability distribution forecasts reported in an interval or quantile format, as this format solely requires the definition of nominal levels. In fact, forecasts issued in a quantile or interval format were adopted during the COVID-19 pandemic by the COVID-19 Forecast Hub (Ray et al., 2020). The previously mentioned principle of training a model to maximize the sharpness of its predictive distribution subject to calibration remains prominent for quantile and interval forecasts. In fact, the calibration of quantile forecasts can commonly be assessed with the empirical coverage metric, often referred to as coverage, which corresponds to the share of observed values that fall under a given prediction quantile, e.g. the percentage of observations that is lower or equal to the value of the 50% quantile (Gneiting et al., 2023). In contrast to the coverage, the previously covered scoring rules are not applicable to the

quantile or interval format. Thus, we are going to present more adequate scoring rules, which were also portrayed by Bracher et al. (2021).

Initially, Cervera and Muñoz (1996) defined a set of proper scoring rules for quantiles. In addition, Gneiting and Raftery (2007) expanded this set to a more general formulation of propriety for quantile scoring functions:

$$S(r_1, \dots, r_k; x) = \sum_{i=1}^{k} \left[\alpha_i s_i(r_i) + \left(s_i(x) - s_i(r_i)\right) I(x \leq r_i)\right] + h(x), \tag{3}$$

such scoring rules are proper for forecasts at the quantiles at levels $\alpha_1, \dots, \alpha_k \in (0,1)$, for s_i with $i = 1, \dots, k$ nondecreasing and an arbitrary h. Here r_1, ..., r_k are the quantiles reported by the forecaster, while *x* is the materialization and *I* is the indicator function. Throughout this thesis we consider standard central prediction intervals (PIs), so that $(1-\alpha)$ is the nominal coverage rate and the lower and upper bound correspond to the level $(\alpha/2)$ and $(1-\alpha/2)$ predictive quantiles respectively. Gneiting and Raftery (2007) propose that when one initializes the above formula with the following parameters $\alpha_1 = \frac{\alpha}{2}$, $\alpha_2 = 1 - \frac{\alpha}{2}$, $s_1(x) = s_2(x) = 2\frac{x}{\alpha}$ and $h(x) = -2\frac{x}{\alpha}$, one receives the interval score (IS):

$$IS_\alpha(l, u; x) = (u - l) + \frac{2}{\alpha} * (l - x) * I(x < l) + \frac{2}{\alpha} * (x - u) * I(x > u). \tag{4}$$

Note that due to the negative orientation of the interval score, in which lower values are better, it is additionally necessary to multiply Equation (4) by minus one to receive a positively oriented version. According to Bracher et al. (2021) the IS includes three simple yet important concepts. First, the IS uses the length of the $(1-\alpha)$ prediction interval, thus the sharpness of the predictive distribution is included. Second, through the implementation of the indicator function $I(x < l)$ the observations that are below the lower bound will be penalized. Furthermore, the effective penalty value is dependent upon the magnitude of error $(l - x)$ as well as the level of α, where the penalty increases in value for high $(1-\alpha)$ values. Third, a similar term penalizes observations that are above the upper bound of the prediction interval.

Alternatively, the IS may also be computed from the standard piecewise linear quantile score (QS) (Gneiting and Raftery, 2007; Bracher et al., 2021) with the following equations:

$$QS_\tau(F, x) = 2 \cdot (I(x \leq q_\tau) - \tau) \cdot (q_\tau - x), \tag{5}$$

$$IS_\alpha(F, x) = \frac{QS_{\alpha/2}(F, x) + QS_{1-\alpha/2}(F, x)}{\alpha}. \tag{6}$$

The QS is calculated for a specified quantile level τ, where q_τ is the corresponding quantile of forecast F, *x* is the materialization and *I* is the indicator function. Given the quantile scores at the $(\alpha/2)$ and $(1-\alpha/2)$ level we can compute the IS for the central $(1-\alpha)$ prediction interval as in Equation (6).

Although the interval score is an adequate proper scoring rule for quantile and interval forecasts, Bracher et al. (2021) describe that the general task is to provide forecasts for multiple distinct levels of PIs $(1-\alpha_1) < (1-\alpha_2) < \ldots < (1-\alpha_K)$. In addition, one is also often tasked to report the predictive median m, which can be seen as the $(1-\alpha_0)$ prediction interval. Thus, Bracher et al. (2021) suggest using the weighted interval score (WIS):

$$WIS_{\alpha_{0:K}}(l,u;x) = \frac{1}{\mathrm{K}+0.5} \cdot \left(\mathrm{w}_0 \cdot |x-m| + \sum_{k=1}^{K} \{ \mathrm{w}_k \cdot IS_\alpha(l,u;x) \} \right), \tag{7}$$

with weights w_i for $i = 1, \ldots, \mathrm{K}$. If all weights are nonnegative and unnormalized, the WIS is proper. Moreover, Bracher et al. (2021) propose that this selection of weights:

$$w_k = \begin{cases} \frac{1}{2} \text{ for } k = 0 \\ \frac{\alpha_k}{2} \text{ for } k > 1 \end{cases} \tag{8}$$

is natural and leads to an approximation of the CRPS by the WIS, when given a wealth of equally spaced levels $\alpha_1, \alpha_2, \ldots, \alpha_K$. This score has been used in numerous applications in epidemiology, e.g. Cramer et al. (2022), Bracher et al. (2022), Bosse et al. (2022) and Taylor and Taylor (2022). Based on the alternative formulation of the IS in Equation (6), the WIS with weights of Equation (8) can also be expressed through the QS:

$$WIS_{\alpha_{0:K}}(F;x) = \frac{1}{2\mathrm{K}+1} \cdot \left(QS_{\tau_{K+1}}(F,x) + \sum_{k=1}^{K} \left\{ QS_{\tau_k}(F,x) + QS_{\tau_{2K+2-k}}(F,x) \right\} \right), \tag{9}$$

$$= \frac{1}{2\mathrm{K}+1} \cdot \left(\sum_{k=1}^{2K+1} \{ 2 \cdot \left(I\left(x \le q_{\tau_k}\right) - \tau_k \right) \cdot (q_{\tau_k} - x) \} \right), \tag{10}$$

with quantiles $q_{\tau_1}, \ldots, q_{\tau_{2K+1}}$ at corresponding levels $0 < \tau_1 < \ldots < \tau_{K+1} = 0.5 < \ldots < \tau_{2K+1} < 1$. The median is represented by the $q_{\tau_{K+1}}$ quantile, whereas the remaining quantiles correspond to the lower and upper bounds of the central prediction intervals $(1-\alpha_1), \ldots, (1-\alpha_K)$ (Bracher et al., 2021).

3 Local and global forecasting methods

Previously we distinguished between the prediction methods of probabilistic and point forecasts and established rules to evaluate them. Aside from that, we can differentiate between local and global models for tasks involving a multitude of similar time series. Traditionally, these tasks were addressed by local models which estimate their model parameters individually for each time series. However, the age of Big Data introduced soaring amounts of related time series, where each series might contain relevant information for the prediction of the others. Therefore, global models which estimate their model parameters jointly from all available time series through cross-learning are specifically useful in these situations.

Generally, the goal in univariate time series forecasting of multiple available time series $z_{i,t_0:T}$ is to estimate the conditional distribution:

$$P\left(z_{i,t_0:T}\middle|Z_{1:t_0-1}, X_{1:T}; \theta_i\right), \tag{11}$$

in which $z_{i,t_0:T}$ denote the values of time series i on the prediction range $[t_0, T]$, while $Z_{1:t_0-1}$ are the values of all available time series $i = 1, \dots, N$ on the conditioning range $[1, t_0 - 1]$, and $X_{1:T}$ are the covariates which have to be present throughout the time points $[1, T]$ (Salinas et al., 2020). Since we are within the univariate setting, we predict each time series i separately, therefore the parameters of our model θ_i are also time-series dependent for each time series i. Note that in a multivariate setting the forecasts would predict the collective of available time series $Z_{t_0:T}$ jointly, while the model parameters θ would be time series independent. Hence, the division into local and global models is considered complementary to the distinction between univariate and multivariate models (Januschowski et al. 2022).

Local univariate models independently estimate the parameters for each time series with the following predictive distribution:

$$P\left(z_{i,t_0:T}\middle|z_{i,1:t_0-1}, X_{i,1:T}; \theta_i\right), \text{with } \theta_i = \Psi\left(z_{i,1:t_0-1}, X_{i,1:T}\right), \tag{12}$$

where Ψ is a function that outputs the (local) distribution parameters of time series i (Benidis et al., 2023). Furthermore, the input to this function Ψ consists of the past values of time series i, as well as the observed covariates $X_{i,1:T}$, which can be multi-dimensional. Since local models scale with the size of the data set, Montero-Manso and Hyndman (2021) conclude that they may have a higher model complexity compared to global approaches, which have constant complexity. Additionally, Benidis et al. (2023) note that local models are unable to forecast cold start problems, that is the target time series at hand has few or no historical

values for the training process. Whereas global models are capable of handling cold start problems. Furthermore, a univariate global model estimates the conditional distribution:

$$P\left(z_{i,t_0:T}\middle|Z_{1:t_0-1}, X_{1:T}; \theta_i\right), \text{with } \theta_i = \Psi\left(z_{i,1:t_0-1}, X_{i,1:T}, \Phi\right), \tag{13}$$

where the learnable parameters Φ are shared by all time series during the training process of the model Ψ, which is usually represented by neural networks (Benidis et al., 2023). Due to the addition of the learnable parameters Φ, individual time series may depend upon every other time series $Z_{1:t_0-1}$ and their associated covariates $X_{1:T}$. Although the training process is done collectively for the time series at hand, the predicted parameters θ_i remain individual for each time series, i.e. each time series is forecasted separately. Despite the fact that global models have been regarded almost restrictively for problems with homogeneous groups of time series, Montero-Manso and Hyndman (2021) show that they can be applied successfully to problems with heterogeneous groups of time series as well. Nevertheless, Hewamalage et al. (2021a) argue that the outperformance conditions of global over local models remain uncertain. Furthermore, they constructed multiple experiments on this behalf, where they discovered that in uncertain situations non-linear, non-parametric models may be favourable overall.

4 Methods

This chapter explains the models and methods implemented in Chapter 6. First, we go over the theoretical concepts of artificial and recurrent neural networks. Second, we summarise the DeepAR model introduced by Salinas et al. (2020). Lastly, we discuss the baseline model in Section 4.4.

4.1 Artificial neural networks

In Chapter 3, we have mentioned that global models are usually represented by artificial neural networks (ANNs), in this section we briefly introduce ANNs and feedforward neural networks (FNNs). However, we further refer to Goodfellow et al. (2016) for a more detailed description of ANNs, as well as Benidis et al. (2023), who provide a comprehensive overview of common modern deep forecasting models.

To begin with, ANNs, also referred to as neural networks (NNs), represent an information-processing method, inspired by the information processing in biologic nervous systems (Dongare et al., 2012). In a more practical view, they can be regarded as non-linear statistical data modelling tools used to find data patterns or to model connections between inputs and outputs (Singh and Chauhan, 2009). NNs are built from building blocks, also called neurons or units, which are combined into connected layers. The neurons work together to learn to solve a problem by example, similar to the learning process of humans (Maind and Wankar, 2014).

One of the first mathematically modelled neurons was introduced by McCulloch and Pitts (1943). In particular, a neuron of this kind can be interpreted as a non-linear function, which produces a scalar output based on its input (Bishop, 1994). Figure 1a displays a neuron that receives an input x_i and consists of an affine transformation with trainable weights w_i and bias b as well as an optional nonlinear activation function f, i.e. $f(\sum x_i w_i + b)$. This nonlinear activation function f makes a network more expressive, as it enables the NN to model nonlinearity, whereas the bias b gives the NN the possibility to move the activation function horizontally (Kriegeskorte and Golan, 2019). Despite this simple neuron structure, it has been proven that any function that satisfies certain weaker constraints can be approximated by a sufficiently large network, provided it implements an appropriate structure and properly chosen weights (Osman Kurban, 2004).

Over the years, different types of NNs have evolved to suit certain tasks, e.g. recurrent neural networks (RNNs) for sequential tasks or convolutional neural networks (CNNs) for processing and analysing structured image and video data. A common and simple type of NN is the feedforward neural network (FNN), which is built by stacking layers of neurons on top of each other. Note that the often interchangeably used term multilayer perceptron (MLP)

corresponds to a subclass of fully connected FNNs. A MLP is depicted in Figure 1b as an acyclic graph with nodes and edges, corresponding to neurons and trainable weighted connections respectively. It is visible that every edge moves the neuron's output as an input to a neuron of the next layer (Wang et al., 2020a). Thus, FNNs are called feedforward, as the information flows from the first to the last layer without feedback connections (Goodfellow et al., 2016). The first and last layer correspond to the input and output layer respectively, whereas the remaining layers are called hidden layers. Moreover, the input layer receives the inputs of the problem, the hidden layer models the relationship between inputs and outputs by adjusting the parameters, i.e. weights and biases, and the output layer returns the output of the network (Dongare et al., 2012). In fact, the overall number of layers corresponds to the network's depth. Generally, we categorize a NN as deep when it includes more than one hidden layer (Kriegeskorte and Golan, 2019), these NNs then fall under the category of deep learning (DL) (Schmidhuber, 2015). Because deep NNs (DNNs) combine several hidden layers, higher hidden layers can work with the results of lower hidden layers, this enables them to achieve good performances with comparably fewer weights and biases (Kriegeskorte and Golan, 2019).

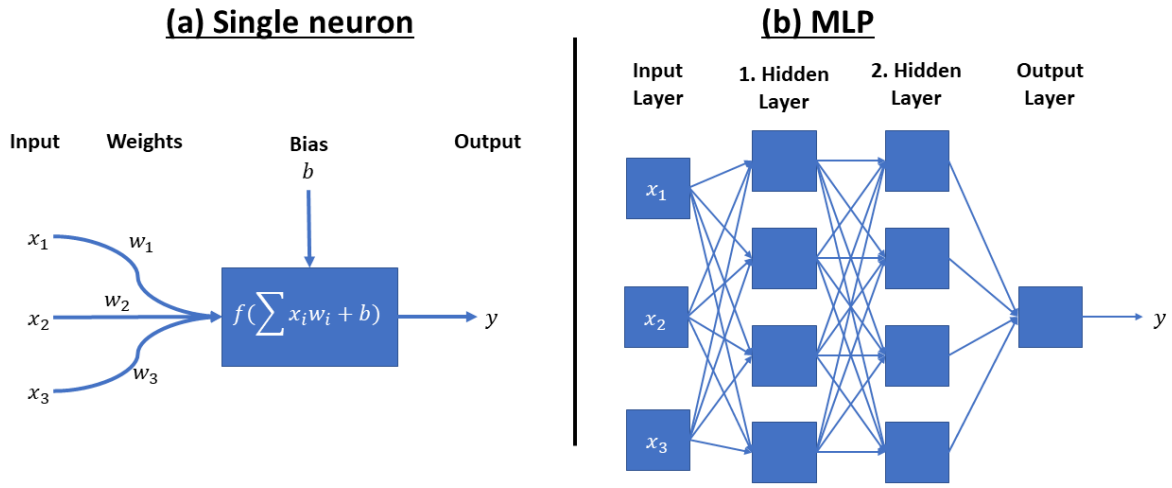

Figure 1: (a) Blueprint of a single neuron or unit. Each input is multiplied with a corresponding weight. Then they are added together with a bias followed by an activation function. Lastly, the output y is returned. Note that the bias and weights are trainable parameters. (b) Depiction of the network of a multilayer perceptron (MLP) with four layers. Every square corresponds to a single neuron, except the squares of the input layer.

Throughout the years, DNNs were responsible for breakthroughs in many important topics, e.g. speech recognition, speech synthesis, knowledge discovery and image recognition (Van Den Oord et al., 2016; Krizhevsky et al., 2017; Schmidhuber, 2015; LeCun et al., 2015). Although FNNs have been applied successfully in some time series applications as well (Raeesi et al., 2014; Abdulkarim and Engelbrecht, 2021), they were often outperformed by traditional statistical models (Farajzadeh et al., 2014; Ho et al., 2002; Oancea and Ciucu,

2014). This claim is supported by several time series prediction contests, e.g. the M3 (Makridakis & Hibon, 2000) and NN3 (Crone, et al., 2011) competitions, which further displayed poor performances of NNs compared to rivalling models. However, Hewamalage et al. (2021b), Hyndman (2020) as well as Kriegeskorte and Golan (2019) suspect that this is due to formerly insufficient computing resources and sparse data. Nevertheless, RNNs are generally better suited for time series problems, as this type of NN has been specifically designed for sequential tasks. Hence, the next chapter first introduces RNNs, before covering characteristic elements and network architectures of RNNs.

4.2 Recurrent neural networks

Previously, we have introduced feedforward neural networks (FNNs), which have proven successful in a multitude of applications, however they have a limited memory capacity and a poor fit on sequential modelling tasks (Zeroual et al., 2020). Thus, recurrent neural networks (RNNs) are often chosen in their favour, as RNNs are specialised to work with sequential data (Goodfellow et al., 2016). Moreover, the effectiveness of RNNs on sequential tasks is mainly due to cyclic connections between consecutive hidden states, which allow the network to incorporate information from past hidden states and current inputs into the calculation of the current hidden state (Yu et al., 2019). Hence, RNNs can remember previous inputs through their hidden state, enabling them to produce outputs that are based on past and current inputs. In addition, RNNs share weight parameters between hidden layers, whereas regular FNNs incorporate different weights for every layer (Benidis et al., 2023). In general, Jordan (1986) regards NNs as recurrent if they include at least one cycle from a unit back to itself.

RNNs have been applied successfully to many sequential tasks often connected to natural language topics, e.g. machine translation (Sutskever et al., 2014; Bahdanau et al., 2016; Kalchbrenner and Blunsom, 2013; Wu et al., 2016), language modelling (Pascanu et al., 2014; Sutskever et al., 2011; Graves, 2014) and speech recognition (Graves et al., 2013; Sak et al., 2014). In the last years, RNNs have received more attention addressing time series problems, as they started to succeed in related forecasting competitions. In fact, a RNN-based model developed by Smyl (2020) won the M4 competition (Makridakis et al., 2020). Similarly, a RNN won the Kaggle Web traffic[3] competition sponsored by Google Inc. and Voleon in 2017. In 2022, the M5 competition was hosted by Makridakis et al. (2022). Although this competition was dominated by tree-based models, a DeepAR modelling approach, which itself further incorporates RNNs, ranked among the top three submissions.

[3] https://www.kaggle.com/c/web-traffic-time-series-forecasting

In general, different RNNs are often distinguished by their implemented type of cell and the arrangement of their cells, i.e. their architecture. Thus, in the next section we will present commonly implemented recurrent cells, namely the Elman cell, the long-short-term memory (LSTM) cell and the gated recurrent unit (GRU) introduced by Elman (1990), Hochreiter and Schmidhuber (1997) as well as Cho et al. (2014) respectively. After that, we will summarise the stacked recurrent architecture (Schmidhuber, 1992; Hihi and Bengio, 1995) and the Encoder-Decoder RNN (Cho et al., 2014) in Section 4.2.2.

4.2.1 Recurrent cells

A first successful instance of a recurrent neural network was implemented by Elman (1990). He expanded the research of Jordan (1986) and introduced the Elman network, as depicted in Figure 2a. Elman's network introduces a context unit that stores past information and feeds it back to the hidden unit, enabling the network to incorporate past information when updating the current cell state (Hewamalage et al., 2021b). Furthermore, the layers of the Elman network are separated into input x, output y, hidden h and context c, they are additionally connected by trainable connections (solid lines). In contrast, the hidden-to-context connection (dotted line) has a fixed weight of one, therefore it cannot be trained. In fact, it simply clones the output of the hidden unit at time point t, known as its hidden state h_t, into the context layer. Thus, the context layer can be replaced by a trainable connection from the preceding hidden state h_t to its following hidden state h_{t+1}, see Figure 2b. Furthermore, we can distinguish between multiple networks, depending on the combination of trainable connections. Yu et al. (2019) discuss a network that implements the standard recurrent cell, a type of Elman cell visible in Figure 3a, where solely the input-to-hidden and hidden-to-hidden layer connections are trainable and insert them into an affine linear transformation, as such:

$$h_t = \sigma(W_h \cdot h_{t-1} + V_x \cdot x_t + b_i), \tag{14}$$

where W_h and V_x are trainable weights of hidden and input layer, while b_i is the bias. Additionally, $\sigma(\cdot)$ represents the sigmoid function and h_t and x_t denote the hidden state and the input vectors of the cell at time t respectively. Lastly, the output vector y_t is simply set to the hidden state value h_t, as follows:

$$y_t = h_t. \tag{15}$$

In a different approach Hewamalage et al. (2021b) model the basic Elman RNN (ERNN) cell, see Figure 3b, which also derives its hidden state from Equation (14), however it adds an additional trainable hidden-to-output connection:

$$y_t = tanh(W_o \cdot h_t + b_o), \tag{16}$$

with the weight matrix W_o and the bias b_o, as well as the hyperbolic tangent function $tanh(\cdot)$ used for the activation of the output.

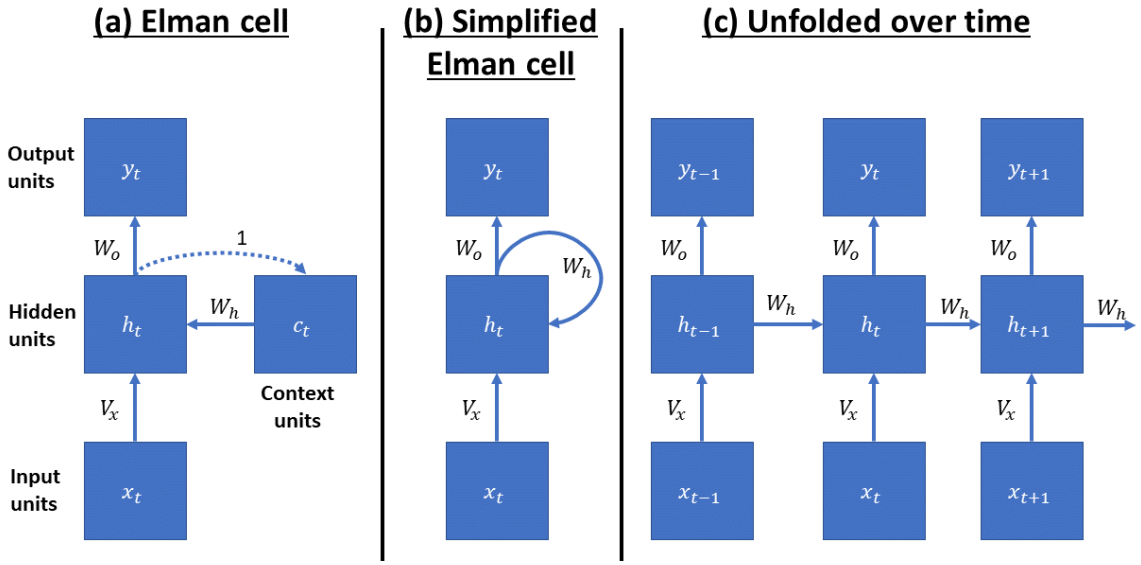

Figure 2: (a) Visualization of the recurrent network introduced by Elman (1990). It contains four different unit types, i.e. input, hidden, context and output units. Trainable weighted connections are labelled with V_x, W_h, and W_o, whereas the unweighted connection between hidden and context units is denoted with 1. (b) Depiction of an equivalent simplified version of the original Elman network. It replaces the unnecessary context units of the original Elman network with weighted connections. (c) An illustration of the simplified Elman network unfolded over multiple time steps.

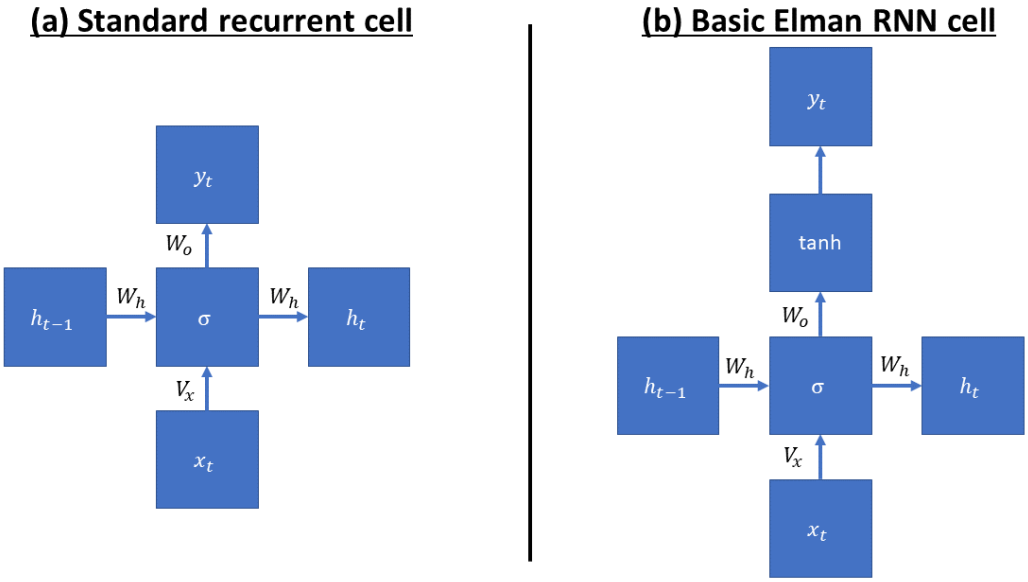

Figure 3: Comparison of (a) the standard recurrent cell (Yu et al., 2019) and (b) the basic ERNN cell (Hewamalage et al., 2021b). Both cells implement a sigmoid function $\sigma(\cdot)$ to model the hidden state h_t given the input x_t and the previous hidden state h_{t-1}. However, the basic ERNN cell additionally applies the hyperbolic tangent function $tanh(\cdot)$ before returning the output.

Upon implementation to a sequential XOR task, Elman (1990) showed that his network is capable of learning some of the input's inherent temporal structure. Although the Elman cell and its variations are still commonly implemented in some projects today, they are often succeeded by improved cell designs. In fact, Bengio et al. (1994) have shown that Elman's recurrent cells exhibit the problem of vanishing and exploding gradients when handling long sequences. Thus, they are unable of transferring long-term effects into the future (Hewamalage et al., 2021b). Therefore, multiple research teams have proposed methods to solve this issue like Bengio et al. (1994), Lang et al. (1990), Ring (1992), Schmidhuber (1992) or Mozer (1991) among others. However, the long-short-term memory network (LSTM), introduced by Hochreiter and Schmidhuber (1997), amounted to a state-of-the-art, as for example Yu et al. (2019) conclude that there is no other recurrent cell type which universally dominates the LSTM cell in terms of performance.

The first LSTM cell introduced by Hochreiter and Schmidhuber (1997) is a complex memory unit derived from a recurrent self-connected linear unit called the constant error carousel. Additionally, the LSTM cell introduces a cell state c_t that stores information over variable time periods. Hence the cell state c_t represents the long-term memory, while the hidden state h_t represents the short-term memory of the LSTM cell (Hewamalage et al., 2021b). Besides that, Hochreiter and Schmidhuber (1997) also innovate a gating mechanism that consists of two gates, the input gate, which determines what new information is going to be stored in the cell, and the output gate, which decides what information will be included into the final output. Both gates depend upon the current cell state c_t and the input x_t. The modern LSTM cell, as depicted in Figure 4, can be represented with mathematical equations as follows:

$$i_t = \sigma(W_i \cdot h_{t-1} + V_i \cdot x_t + b_i), \tag{17}$$

$$o_t = \sigma(W_o \cdot h_{t-1} + V_o \cdot x_t + b_o), \tag{18}$$

$$\tilde{c}_t = \tanh(W_{\tilde{c}} \cdot h_{t-1} + V_{\tilde{c}} \cdot x_t + b_{\tilde{c}}), \tag{19}$$

$$c_t = c_{t-1} + i_t \odot \tilde{c}_t, \tag{20}$$

$$y_t = h_t = o_t \odot tanh(c_t), \tag{21}$$

where W_i, W_o and $W_{\tilde{c}}$ represent the weight matrices of input gate, output gate and candidate cell state respectively. Similarly, V_i, V_o, $V_{\tilde{c}}$ and b_i, b_o, $b_{\tilde{c}}$ denote the weight matrices related to the current input x_t and the biases respectively. The values for the input and output gate are derived from affine linear transformations surrounded by the sigmoid function. Furthermore, the cell state is obtained from the sum of the previous cell state c_{t-1} as well as the product of the element wise multiplication $\odot$ of input gate value i_t and candidate cell state $\tilde{c}_t$. Likewise, the hidden state h_t, which equals the final output value y_t, is represented by the elementwise

multiplication product of output gate value o_t and hyperbolic tangent value of the current cell state c_t. Because the sigmoid function produces values in the range of [0,1], the input and output gate value can set the elementwise multiplication product to zero if they are themselves zero. Alternatively, a value of one maintains the value of their respective multiplication counterpart.

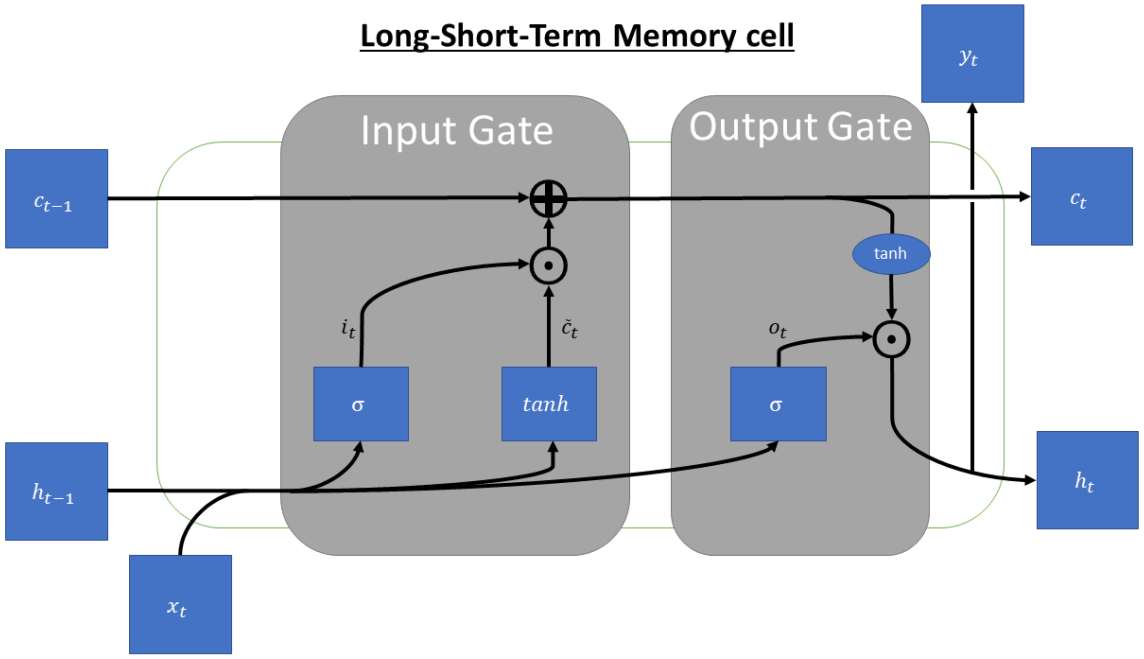

Figure 4: Visualization of the first LSTM cell proposed by Hochreiter and Schmidhuber (1997). It introduces a cell state c_t and a gating mechanism, which consists of an input and output gate. Note that the ⊙ symbol represents elementwise multiplication, whereas ⊕ symbolizes regular addition.

Hochreiter and Schmidhuber (1997) conclude that the LSTM cell can solve tasks previously regarded unsolvable for RNNs, as they show that a LSTM network can solve long-time lag problems involving distributed, continuous-valued representations. Upon further research Gers et al. (2000) express a severe limitation of this unit, they detect linear growth during the presentation of a time series within the cell states c_t. Hence, the cell states could grow unbounded for a continuous input stream, note that this persists for tasks where the cell state is assumed resettable. Furthermore, Goodfellow et al. (2016) explain that some tasks may contain sequences consisting of sub-sequences, where the network builds up information within every sub-sequence, while being required to forget the old state by setting it to zero. Ideally, the RNN handles such tasks by learning when to clear the state itself. Therefore, Gers et al. (2000) propose an addition to the LSTM cell. They introduce a forget gate that enables the model to decide which information will be cut from the previous cell state during calculation of the current cell state. This addition to the cell is shown in Figure 5.

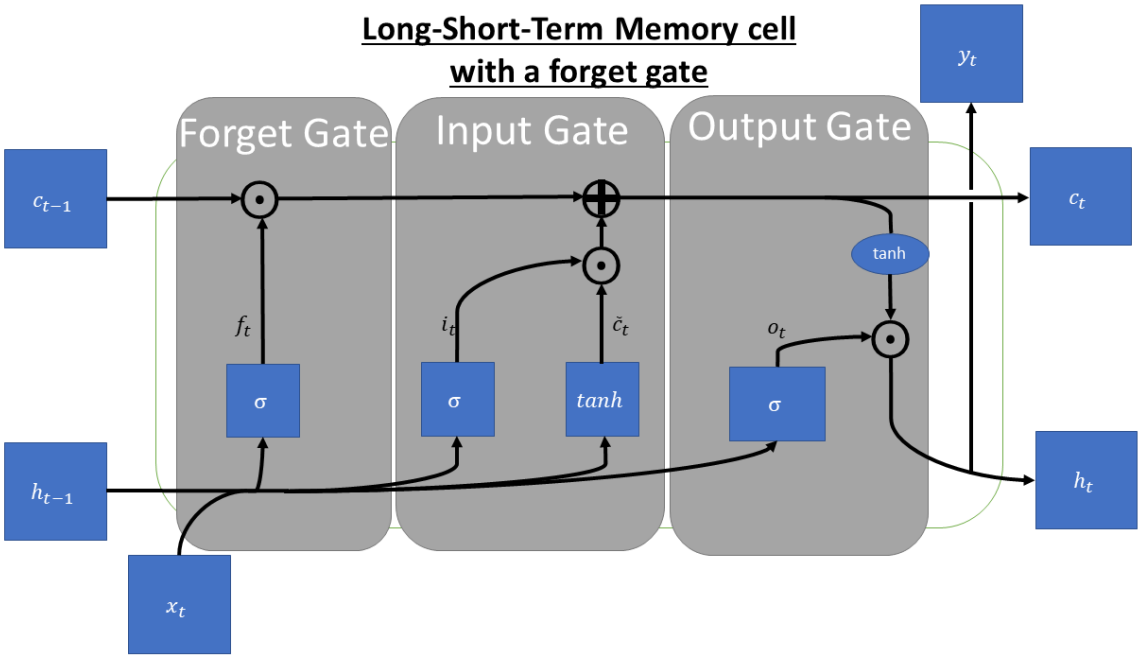

Figure 5: Illustration of the LSTM cell with a forget gate, which enables the RNN to learn to cut information from the previous cell state during the calculation of the current cell state.

The addition of the forget gate as well as its impact on the cell state calculation can be mathematically expressed as follows:

$$f_t = \sigma\big(W_f \cdot h_{t-1} + V_f \cdot x_t + b_f\big), \tag{22}$$

$$c_t = c_{t-1} \odot f_t + i_t \odot \tilde{c}_t, \tag{23}$$

while the remainder of the previously introduced equations stay identical. In fact, if the forget gate value, which lies in the range of [0,1], was equal to one for each time point, this cell would correspond to the LSTM cell without a forget gate. Conclusively, a forget gate value of one implies that the previous cell state c_{t-1} is included entirely during the determination of the current cell state, while it is removed entirely if the forget gate value equals zero. In further research Gers and Schmidhuber (2000) conclude that the extension of forget gates is mandatory for problems with a continuous input. However, they further suggest that LSTM cells benefit from direct connections between cell state and each gate, as this allows the gates to access information stored within the current internal state. Thus, they introduce the LSTM cell with peephole connections, also known as the vanilla LSTM cell. It is depicted in Figure 6. The implementation of peephole connections exclusively influences the gate equations, which are modified as such:

$$i_t = \sigma(W_i \cdot h_{t-1} + V_i \cdot x_t + P_i \cdot c_{t-1} + b_i), \tag{24}$$

$$o_t = \sigma(W_o \cdot h_{t-1} + V_o \cdot x_t + P_o \cdot c_t + b_o), \tag{25}$$

$$f_t = \sigma\big(W_f \cdot h_{t-1} + V_f \cdot x_t + P_f \cdot c_{t-1} + b_f\big), \tag{26}$$

where P_i, P_o and P_f represent the cell state weight matrices of each gate. Note that the input and forget gate both receive the previous cell state c_{t-1} as an input, while the output gate receives the current cell state c_t.

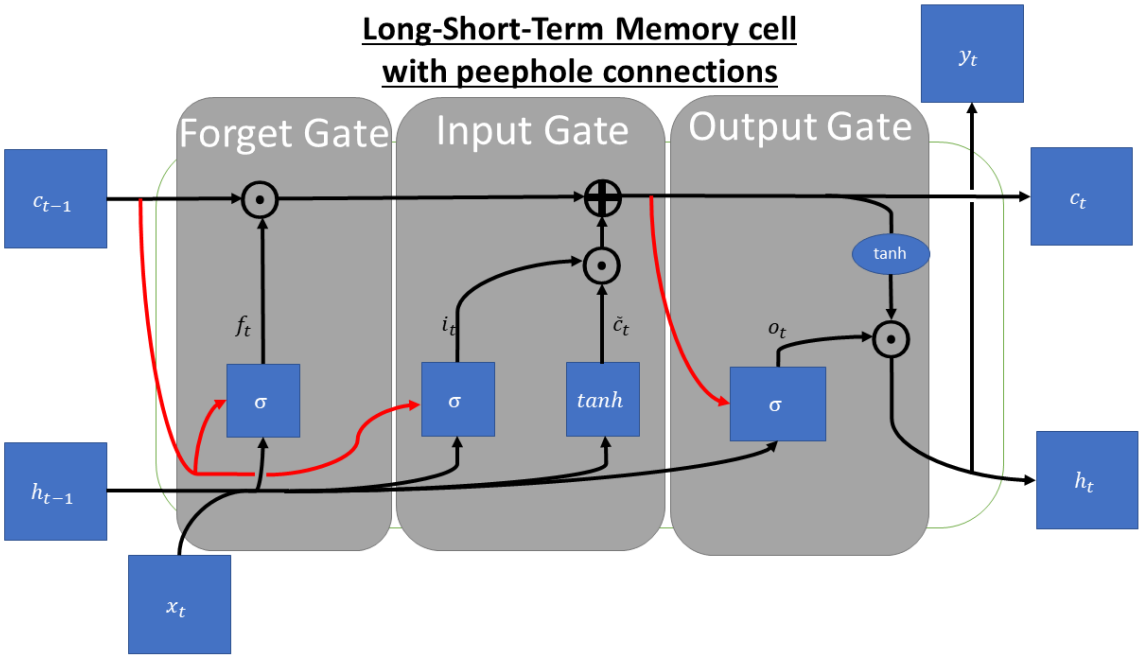

Figure 6: Depiction of the LSTM cell with a forget gate and peephole connections. The peephole connections enable the RNN to implement information stored in the previous cell state into the forget and input gate, whereas information stored in the current cell state is utilised in the output gate.

After conducting several experiments, in which the network has been tasked to learn to time and count explicitly or implicitly, Gers and Schmidhuber (2000) showed that the LSTM network without peephole connections fails to produce a solution or performes worse than the peephole LSTM. Hence, they conclude that the addition of peephole connections provides LSTM networks with necessary resources to fulfill these kinds of problems. Furthermore, Gers and Schmidhuber (2001) conclude that the LSTM with peephole connections is the first RNN capable of learning context sensitive languages. However, Gers et al. (2001) compared an auto-regressive LSTM network with a time window based MLP on time series data and discovered that the MLP outperformed the LSTM network. Therefore, they suggest that LSTM networks should be used on problems, where the traditional time window approaches fail. Although LSTM networks are considered a state-of-the-art for machine learning problems by Greff et al. (2017), they appear ad-hoc to Jozefowicz et al. (2015), as the significance of individual parameters is not fully understood and thus their assumed optimality is not certain. As a consequence, Greff et al. (2017) compared the

performance of eight LSTM variants, derived by changing a single property of the vanilla LSTM cell. They come to the conclusion that no cell can improve significantly upon the vanilla LSTM cell. In addition, they discover that the output activation function and the forget gate are of greater importance, as their removal significantly reduces the performance. Khaldi et al. (2022) also experimented with LSTM cell variations and discovered that the output activation function is a critical component to capture long-term dependencies. However, they additionally conclude that the LSTM cell without an output activation function is best for chaotic behaving time series. In a more general RNN cell comparison, Jozefowicz et al. (2015) identify the input and forget gate as most important to the LSTM cell, while the output gate is rather unimportant.

Despite their notable results in multiple applications, Cho et al. (2014) wanted to improve the LSTM units by reducing their computational complexity. Thus, they created the gated recurrent unit (GRU), depicted in Figure 7, which is closely related to the LSTM cell and reduces the number of gates to two. Namely, the reset gate r_t and the update gate u_t, the latter combines the roles of input and forget gate used in LSTM cells (Hewamalage et al., 2021b). Because the single GRU cell has fewer associated parameters due to the reduction of gates, it is less powerful than the single LSTM cell (Yu et al., 2019).

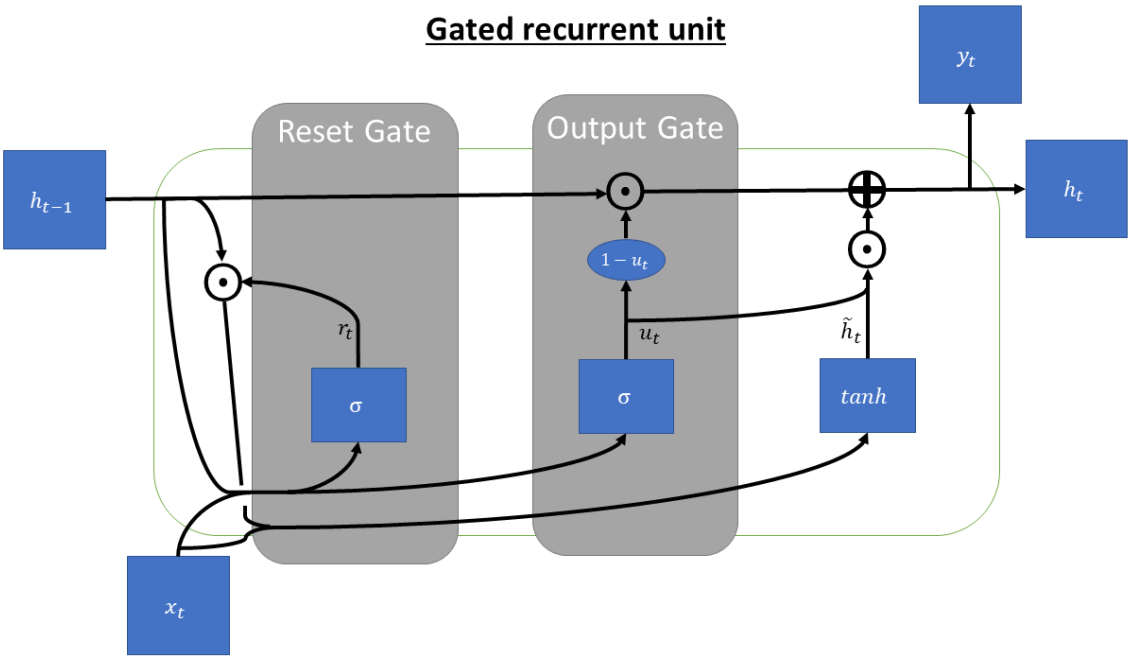

Figure 7: Image of the GRU cell introduced by Cho et al. (2014). This cell consists of a reset and output gate. In addition $\tilde{h}_t$ represents the current candidate hidden state.

The GRU cell can be expressed mathematically as follows:

$$r_t = \sigma(W_r \cdot h_{t-1} + V_r \cdot x_t + b_r), \tag{27}$$

$$u_t = \sigma(W_u \cdot h_{t-1} + V_u \cdot x_t + b_u), \tag{28}$$

$$\tilde{h}_t = \tanh(W_{\tilde{h}} \cdot (r_t \cdot h_{t-1}) + V_{\tilde{h}} \cdot x_t + b_{\tilde{h}}), \tag{29}$$

$$y_t = h_t = (1 - u_t) \odot h_{t-1} + u_t \odot \tilde{h}_t, \tag{30}$$

where the reset gate r_t determines how much information of the previous hidden state h_{t-1} is implemented into the current candidate state $\tilde{h}_t$. Moreover, a reset gate value near zero causes the candidate state to ignore the previous hidden state and reset it based on the input x_t. This enables the cell to remove unnecessary information (Cho et al., 2014). The update gate decides on the new information that will be incorporated into the current hidden state h_t from the candidate state $\tilde{h}_t$ while the remainder is kept unchanged, i.e. is simply copied from the previous hidden state h_{t-1}. Similar to the internal cell state c_t in LSTMs, the GRU's update gate is responsible for the storage of long-term dependencies (Cho et al., 2014). Generally, Cho et al. (2014) intended the implementation of separate reset and update gates for each hidden unit, hence every hidden unit is able to capture information over varying time scales. Furthermore, they distinguish between short-term and long-term dependency units, consisting of frequently active reset and update gates respectively. In their conclusion Chung et al. (2014) state that the GRU is similar to the LSTM in terms of performance, however the conditions of outperformance of either cell compared to its competitors remain uncertain. Hence, Jozefowicz et al. (2015) perform an extensive evolutionary RNN cell architecture search, where they compare over ten thousand different RNN architectures with three outperforming the LSTM and GRU cells on most of their tasks. Although they discover cells that outperform the LSTM and GRU on some tasks, they have similar cell architecture types to the LSTM or GRU cell. Thus, they propose that optimal RNN cell architectures, vastly different from the LSTM cell, are at least not found upon a trivial way. Lastly, Khaldi et al. (2022) perform an extensive comparison on synthetically generated time series with a single specified behaviour, split up into deterministic, random-walk, nonlinear, long-memory and chaotic behaviour. Besides the previously mentioned LSTM cell comparison, they experiment on general types of RNN cells, in which the overall best performing cells are variations of the minimal gate unit (Zhou et al., 2016), which further reduces the number of parameters compared to the LSTM or GRU cell. Furthermore, their comparison suggests that the Elman cell has no significant benefit on any data set compared to its rivals.

4.2.2 Architectures

So far, we have talked about the different types of recurrent cells, but RNNs are also distinguished by their architecture. One advantage of implementing advanced architectures

is that this increases the RNNs depth, leading to networks which are capable of efficiently modelling a broader range of functions (Bianchini and Scarselli, 2014). Moreover, Bengio (2009) explains the "depth of an architecture" as the number of layers of non-linear operations that were used to learn a function. Due to this definition, Pascanu et al. (2014) state that the depth of RNNs is ambiguous since time unfolded RNNs theoretically already implement a deep neural network without the addition of artificial depth. However, increasing the number of layers by using different architectures may nevertheless be beneficial. First this chapter presents the simple stacked RNN architecture, followed by the RNN Encoder-Decoder architecture. Additionally, the cells of the following architectures are replaceable by LSTM cells, e.g. a stacked RNN can be turned into a stacked LSTM by replacing the cells with LSTM cells.

The most straightforward implementation of depth into a RNN is achieved by stacking multiple recurrent layers on top of each other, see Figure 8. Moreover, the stacked RNN repeats the same structure for each layer, while the output of a layer is used as the input for the following layer (Hewamalage et al., 2021b). Furthermore, the idea behind the innovation of the stacked RNN by Schmidhuber (1992) and Hihi and Bengio (1995) is that long-term dependencies do not depend upon a specific time scale, instead they estimate that sequential data is structured hierarchically. Therefore, the goal of the stacked RNN is that each layer operates at a different time scale, enabling the RNN of learning long-term dependencies. Although the stacked RNN's ability to handle multiple time scales in an input sequence is advantageous, its connection between consecutive hidden states at every level remain shallow in a classical sense, thus restricting the network in functions it can represent (Pascanu et al., 2014). The stacked RNN with k layers can be mathematically expressed as follows:

$$h_t^{(l)} = \begin{cases} f\left(V^{(1)}x_t + W^{(1)}h_{t-1}^{(1)} + b^{(1)}\right) \text{for } l = 1 \\ f\left(V^{(l)}h_t^{(l-1)} + W^{(l)}h_{t-1}^{(l)} + b^{(l)}\right) \text{for } l = 2, \dots, k \end{cases} \tag{31}$$

$$y_t = g\left(W_o h_t^{(k)} + b_o\right), \tag{32}$$

where $V^{(l)}$, $W^{(l)}$ and $b^{(l)}$ represent the weight matrices and the bias of layer l respectively, thus indicating that each layer implements separate weights and biases. Furthermore, each hidden state is calculated using the hidden state of the previous layer and the previous time point, except for the hidden state of the first layer, which replaces the non-existent previous hidden state with the input x_t instead. Eventually the output y_t is determined in the output layer using the hidden state of the last hidden layer $h_t^{(k)}$ as well as the corresponding weight

matrix W_o and bias b_o. Lastly, $f(\cdot)$ and $g(\cdot)$ represent element-wise activation functions, such as the sigmoid or the hyperbolic tangent function (Turek et al., 2020).

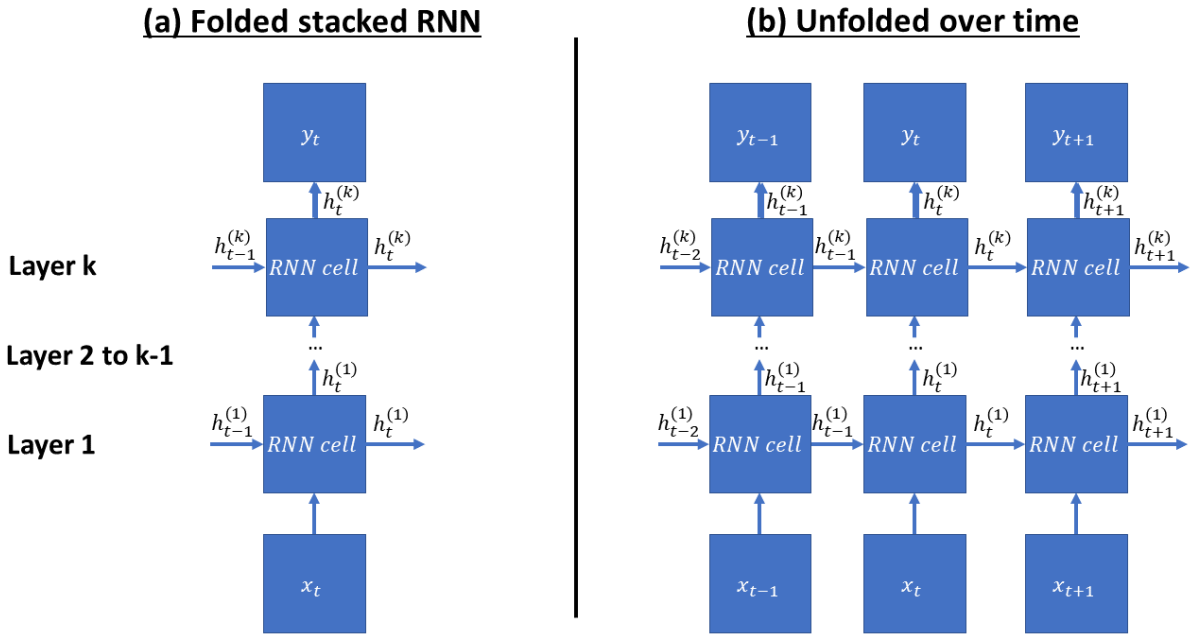

Figure 8: Visualization of a stacked RNN, where (a) shows the folded and (b) the time unfolded representation.

Regular RNNs can be applied straightforwardly to problems, in which we must map sequences to sequences while the lengths of the sequences are known a-priori. However, they are difficult to implement for tasks such as neural machine translation, where the input and output sequences have different lengths and are correlated in a complicated non-monotonic fashion (Sutskever et al., 2014). Thus, the Encoder-Decoder RNN architecture, introduced by Cho et al. (2014), is typically applied to these tasks. This RNN architecture, visible in Figure 9, is composed of two RNNs. First, the Encoder RNN, which learns a mapping of a variable-length input sequence to a fixed-length vector representation $\mathbf{c}$. Moreover, its vector output is equal to the final hidden state, which summarises the whole input sequence. Second, the Decoder RNN learns a mapping of the fixed-length vector $\mathbf{c}$ to a variable-length output sequence. It produces the output sequence by iteratively forecasting the next output y_t, which depends upon the hidden state h_t, the previous output y_{t-1} and the summary vector of the input sequence $\mathbf{c}$. Furthermore, the conditional distribution of the next output point is determined as such:

$$P(y_t|y_{t-1},\dots,y_1,c) = g(h_t,c,y_{t-1}), \tag{33}$$

where the hidden state is further calculated as follows:

$$h_t = f(h_{t-1},c,y_{t-1}), \tag{34}$$

with the activation functions $f(\cdot)$ and $g(\cdot)$, note that $g(\cdot)$ is required to produce valid probabilities in the interval of $[0,1]$ (Cho et al., 2014). Conclusively, the Encoder-Decoder RNN can model variable length sequences to variable length sequences.

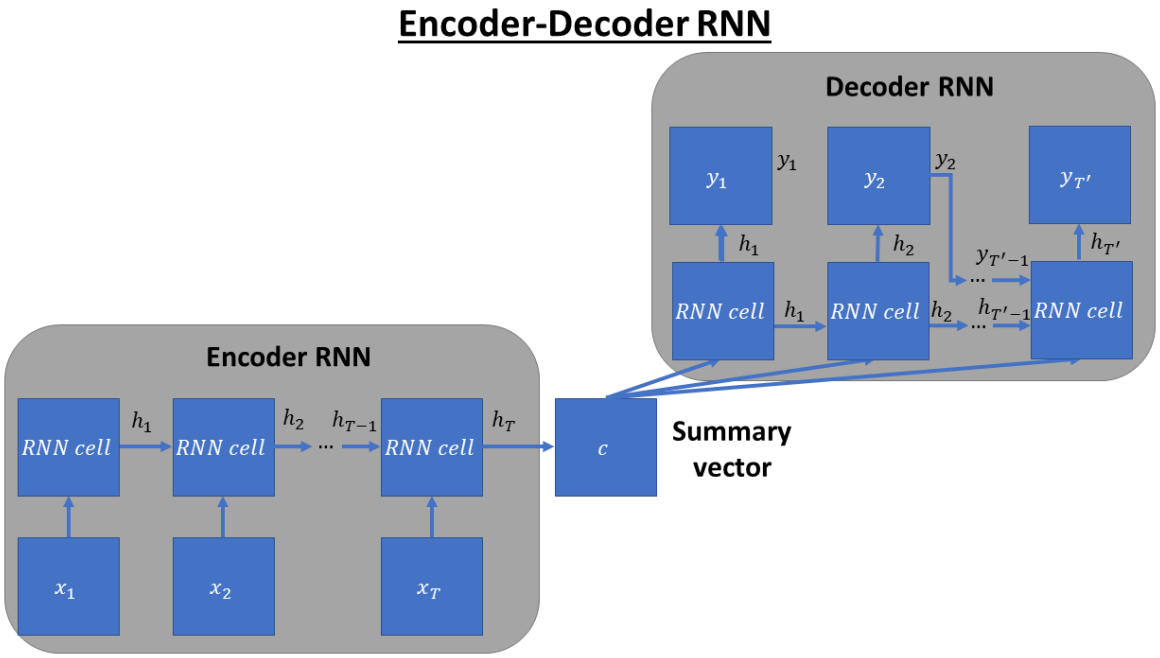

Figure 9: The Encoder-Decoder RNN architecture introduced by Cho et al. (2014). The encoder RNN encodes the information stored in the input sequence into a summary vector c. While the decoder decodes the summary vector and generates sequences.

Generally, the two RNNs are trained together to maximize the conditional log-likelihood:

$$\max_{\theta} \frac{1}{N} \sum_{n=1}^{N} \log p_{\theta}(y_n|x_n) \tag{35}$$

where θ are the model parameters and (y_n, x_n) represent an output input pair. Because language translation tasks prevalently deal with variable length input and output sequences, the Encoder-Decoder RNN was applied successfully to many related tasks, e.g. English to French translation (Cho et al., 2014; Sutskever et al., 2014) and natural language to structured query language translation (Cai et al., 2018). Moreover, Sutskever et al. (2014) suggest that it is important to discover problem encoding sequences that maximize the amount of short-term dependencies. Furthermore, they propose that RNN cells that are capable of handling long-term dependencies, for example the LSTM or the GRU cell, are beneficial. Besides language translation tasks, the Encoder-Decoder RNN was established on a variety of tasks, e.g. software reliability prediction (Li et al., 2022) or multivariate time series forecasting (Zhang et al., 2022). Lastly, the Encoder-Decoder architecture is also incorporated within the DeepAR model, which is the topic of the following section.

4.3 DeepAR

The DeepAR model was implemented by a team of Amazon's researchers (Salinas et al., 2020) to tackle forecasting problems of thousands or millions of related time series by implementing a global univariate modelling approach. Moreover, DeepAR is based around an RNN backbone, while its input consists of past lagged target values $z_{i,t}$ and related covariates $x_{i,t}$ (Benidis et al., 2023). Furthermore, its output is either a point forecast or a probabilistic forecast, by estimating the parameters of a distribution. DeepAR is a global univariate model. Thus, Salinas et al. (2020) assume that the model distribution $Q_\Theta\left(z_{i,t_0:T}\middle|z_{i,1:t_0-1}, X_{i,1:T}\right)$ follows the composition of likelihood factor multiplication as such:

$$Q_\Theta\left(z_{i,t_0:T}\middle|z_{i,1:t_0-1}, X_{i,1:T}\right) = \prod_{t=t_0}^{T} Q_\Theta\left(z_{i,t}\middle|z_{i,1:t-1}, X_{i,1:T}\right) = \prod_{t=t_0}^{T} p\left(z_{i,t}\middle|\theta\left(h_{i,t}, \Theta\right)\right), \tag{36}$$

where the likelihood $p\left(z_{i,t}\middle|\theta\left(h_{i,t}, \Theta\right)\right)$ is a fixed distribution. Furthermore, this distribution depends upon a function $\theta\left(h_{i,t}, \Theta\right)$ that determines the distribution parameters based on model parameters Θ and the output of the underlying recurrent network $h_{i,t}$, expressed as:

$$h_{i,t} = h\left(h_{i,t-1}, z_{i,t-1}, x_{i,t}, \Theta\right), \tag{37}$$

with $h(\cdot)$, a function established by a multi-layer recurrent neural network usually implemented with LSTM or GRU cells (Salinas et al., 2020). In addition, the collective of RNN parameters $h(\cdot)$ and distribution parameters $\theta(\cdot)$ is embodied by the model parameters Θ. Because the model's input in Equation (37) consists of the previous output of the network $h_{i,t-1}$, it is recurrent. Additionally, the model is autoregressive, as the preceding observation $z_{i,t-1}$ is also part of the input.

Ideally, the aforementioned likelihood distribution is selected to complement the statistical characteristics of the data. Thus, we chose to incorporate the provided negative binomial distribution in our analysis in Chapter 6, as it is commonly used to represent positive count data. Furthermore, the negative binomial distribution is implemented into the DeepAR model as follows:

$$p_{NB}(z|\mu, \alpha) = \frac{\Gamma\left(\mathrm{z} + \frac{1}{\alpha}\right)}{\Gamma(z+1)\Gamma\left(\frac{1}{\alpha}\right)} \left(\frac{1}{1+\alpha\mu}\right)^{\frac{1}{\alpha}} \left(\frac{\alpha\mu}{1+\alpha\mu}\right)^{z}, \tag{38}$$

where Γ is the gamma function ($\Gamma(\mathrm{n}) = (\mathrm{n}+1)!$) and the parameters are the mean $\mu \in \mathbb{R}^+$ and the shape parameter $\alpha \in \mathbb{R}^+$, which adjusts the variance proportionally to the mean, i.e. $Var[z] = \mu + \mu^2\alpha$. The DeepAR model directly produces all parameters $\theta(\cdot)$ of the (negative

binomial) distribution, in this case mean μ as well as shape parameter α, for the next time step by applying the following operations:

$$\mu(h_{i,t}) = \log\left(1 + \exp\left(W_\mu^T h_{i,t} + b_\mu\right)\right), \tag{39}$$

$$\alpha(h_{i,t}) = \log\left(1 + \exp\left(W_\alpha^T h_{i,t} + b_\alpha\right)\right), \tag{40}$$

where both parameters implement the network output $h_{i,t}$, which is input into a fully connected layer with trainable weights W_j and biases b_j for j equal to μ and α respectively. In addition, the result of the fully connected layer is then surrounded by a soft-plus activation function to guarantee positively oriented parameters. Figure 10 visualizes the DeepAR model during the training phase. Here, the model's similarity to the Encoder-Decoder becomes clear, as it encodes the inputs of the conditioning range into an initial hidden state h_{i,t_0-1}, which is used as input for the decoder in the prediction range. Although, Salinas et al. (2020) mention that the structure of encoder and decoder can differ from each other, they used the same architecture and shared the weights across both entities, as this mimics the regular encoder-decoder architecture.

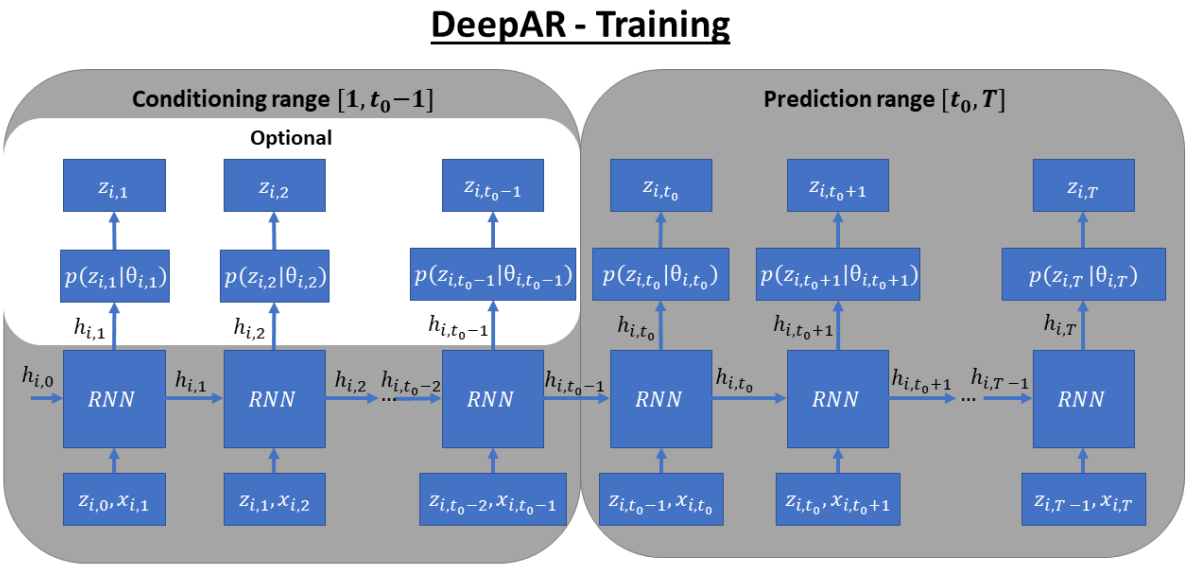

Figure 10: Illustration of the training process of the DeepAR model proposed by Salinas et al. (2020). For each time step t, the network receives the covariates $x_{i,t}$, the previous target value $z_{i,t-1}$ and the previous hidden state $h_{i,t-1}$ as input. Note that the hidden state and target value at time step 0 are initialized with zero. Then the current hidden state, calculated with Equation (37), is used to determine the distribution parameters θ. They are further inserted into the likelihood $p(z|\theta)$, which is used to train the model parameters. Here, the white box represents the possibility to include the conditioning range observations into the learning process.

During the training process, see Figure 10, all observations $\{z_{i,1:T}\}_{i=1,\dots,N}$ and their related covariates $\{X_{i,1:T}\}_{i=1,\dots,N}$ are realized so that the model parameters Θ can be obtained by maximizing the log-likelihood

$$\mathcal{L} = \sum_{i=1}^{N} \sum_{t=t_0}^{T} log\, p\left(z_{i,t} \middle| \theta\left(h_{i,t}, \Theta\right)\right), \tag{41}$$

which can be optimized by the stochastic gradient descent algorithm with respect to Θ. Furthermore, Salinas et al. (2020) state that in cases, where the encoder and decoder are identical, the likelihood terms of the encoder during the conditioning range ($t = 0, \ldots, t_0 - 1$) can be included as well, as they regard the differences to the approach in Equation (41) as negligible.

To prevent the model from learning absolute time through the position of $z_{i,t}$, Salinas et al. (2020) implemented a training window approach, which constrains the model to learn time information exclusively through covariates $x_{i,t}$. Moreover, in this approach the model creates several training windows with separate starting points for every time series, while the total length T and the relative length of conditioning and prediction phase are kept final. Although, the selection of starting points is prohibited to include observations that lie beyond the final training time point, it is possible for starting points to begin before the first observation. However, the values of these earlier time points are set to zero, which enables the model to learn representations of time series with a marginally small or no history of observed values.

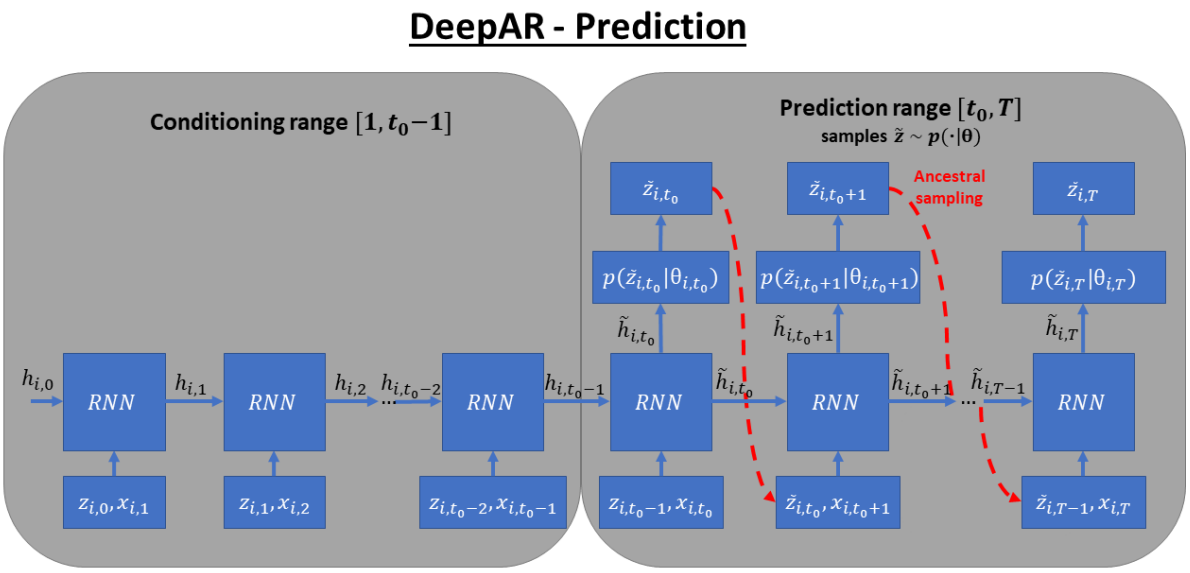

Figure 11: Display of the prediction process of the DeepAR model. Initially, the observations $z_{i,1:t_0-1}$ and covariates $x_{i,1:t_0-1}$ are inserted into the trained model to derive the hidden state h_{i,t_0-1} by evaluating Equation (37). Then ancestral sampling is performed, where samples are drawn $\tilde{z} \sim p(\cdot|\theta)$ and fed back as an input for the next time step. This is repeated until T is reached and a sample trace is created.

After the model parameters Θ have been determined during the training process, samples $\tilde{z}_{i,t_0:T} \sim Q_\Theta\left(z_{i,t_0:T} \middle| z_{i,1:t_0-1}, X_{i,1:T}\right)$ can be drawn through ancestral sampling, see Figure 11. Here, the first step is to derive the initial hidden state for the decoder h_{i,t_0-1} by calculating

Equation (37) for $t = 1, \ldots, t_0 - 1$. After that the initial values for the hidden state and the samples are assigned: $\tilde{h}_{i,t_0-1} = h_{i,t_0-1}$ and $\tilde{z}_{i,t_0-1} = z_{i,t_0-1}$. Finally, the hidden states are determined $\tilde{h}_{i,t} = h\left(h_{i,t-1}, \tilde{z}_{i,t-1}, x_{i,t}, \Theta\right)$ and the samples are drawn $\tilde{z}_{i,t} \sim p\left(\cdot \middle| \theta\left(\tilde{h}_{i,t}, \Theta\right)\right)$ iteratively for $t = t_0, t_0 + 1, \ldots, T$.

Ultimately, large collections of time series are likely to exhibit a power-law of scales, that is the time series differ vastly in size. Thus, Salinas et al. (2020) explain that there are two major challenges. The first problem is that the autoregressive input $z_{i,t-1}$ and some of the network's output parameters scale equivalently with the observations $z_{i,t}$, while the non-linear network operations do not. Hence, the network would have to learn the scaling of the input and the inverted scaling of the output itself. Although this problem can be avoided beforehand for real-valued data by scaling the input in a pre-processing step, this is simply not possible for count distributions. Therefore, Salinas et al. (2020) implement a time-series dependent scale factor v_i, which divides the autoregressive inputs $z_{i,t-1}$ and is applied to the network parameters of the negative binomial distribution as follows:

$$\mu = v_i log\left(1 + exp\left(o_\mu\right)\right), \tag{42}$$

$$\alpha = log\left(1 + exp(o_\alpha)\right)/\sqrt{v_i}, \tag{43}$$

where o_μ, o_α are the preceding output of the network for parameter μ and α respectively. Because it can be difficult to agree upon a suitable representation for this parameter, Salinas et al. (2020) suggest the implementation of the heuristic $v_i = 1 + \frac{1}{t_0}\sum_{t=1}^{t_0} z_{i,t}$.

Besides that, the second problem arises for imbalanced data sets, where uniform-randomly selecting stochastic optimization techniques are likely to choose frequent time series with small magnitudes in favour of rare time series with large magnitudes. Thus, the predictions for rare high magnitude time series would turn out too low. In addition, there is a plethora of scenarios, in which rare time series with high magnitudes are more important for the overall prediction outcome, for example the prediction of total influenza disease cases is expected to be more dependent upon rare larger districts compared to frequent smaller ones. Therefore, Salinas et al. (2020) decided to sample the training instances non-uniformly with a weighted scheme, where the probability to select a window of a certain training instance depends upon the previously introduced scale factor v_i.

Lastly, Salinas et al. (2020) apply the DeepAR model on five different datasets, each of them includes at least hundreds of time series. Additionally, they compare it to four baseline models, that are considered state-of-the-art on demand integer datasets, as well as two baseline RNN models, one with gaussian likelihood, uniform sampling and a simpler scaling

method and the other with negative binomial likelihood, uniform sampling and no scaling. Their results suggest that DeepAR outperforms these models significantly.

In recent times several researchers applied the DeepAR model successfully to varying research topics, for example ground subsidence prediction (Li et al., 2023), network traffic prediction of base station cells (Zhang et al., 2021) or atmospheric PM2.5 prediction (Jiang et al., 2021). Clark et al. (2022) implemented the DeepAR model for the prediction of 165 similar groundwater time series derived from different environmental stations across Australia. After a comparison with other models, they concluded that DeepAR is advantageous in handling time series with few historical data points, however it is not straightforward to implement. Furthermore, they specifically criticize the autoregressive nature of DeepAR as they suspect the model might deliver better long-term predictions without them.

4.4 Baseline model

Held et al. (2005) have introduced an endemic-epidemic multivariate spatio-temporal model designed for disease count data $y_{i,t}$ stratified into units $i = 1,\dots,I$ and time points $t = 1,\dots,T$. In our case, a spatial-temporal application, the unit i corresponds to a geographical region, while it may represent an age group or different pathogens in other cases (Paul et al., 2008). The initial model of Held et al. (2005) is based on a Poisson branching process with immigration. However, in cases where overdispersion is likely, as it is for predominant diseases like influenza, the Poisson distribution is considered disadvantageous to the negative binomial distribution (Paul and Held, 2011). Thus, the general hhh4-model assumes that the infectious disease counts $y_{i,t}$ follow a negative binomial distribution $y_{i,t}|y_{i,t-1} \sim NegBin(\mu_{i,t}, \psi)$ for $i = 1,\dots,I$ with mean

$$\mu_{i,t} = e_{i,t}\upsilon_{i,t} + \lambda_{i,t}y_{i,t-1} + \phi_{i,t}\sum_{j\neq i} w_{ji}y_{j,t-1}\,, \tag{44}$$

and conditional variance $\mu_{i,t}(1 + \psi\mu_{i,t})$. Furthermore, the variance is dependent upon a strictly positive overdispersion parameter $\psi > 0$, which, if set to zero, causes the modelled distribution to reduce to the Poisson distribution. Optionally, this parameter can be implemented unit specific ψ_i, this is especially beneficial when units correspond to different disease types (Paul et al., 2008). The mean is additively partitioned into endemic ($e_{i,t}\upsilon_{i,t}$) and epidemic ($\lambda_{i,t}y_{i,t-1} + \phi_{i,t}\sum_{j\neq i} w_{ji}y_{j,t-1}$) parts. The endemic component $\upsilon_{i,t}$ is responsible for the representation of a default rate of cases determined through external factors, which are often of temporal nature. In addition, it is scaled proportional to an offset of expected counts $e_{i,t}$ which usually corresponds to the population size of region i (Meyer and Held, 2014). The epidemic component is tasked of modelling sudden target deviations based on previous

disease count realizations. It is further decomposed into autoregressive and neighbourhood effects, where the former is established on the previous disease cases of the unit $y_{i,t-1}$. In contrast, the neighbourhood component, also known as the spatiotemporal component, strictly implements previous disease cases from other regions $y_{j,t-1}$ with $j \neq i$. Moreover, its weights w_{ji} correspond to the transmission strength from unit j to unit i, traditionally the weights were chosen as:

$$w_{ji} = \begin{cases} 1/n_j & \text{if } j \text{ and } i \text{ are adjacent} \\ 0 & \text{else} \end{cases}, \tag{45}$$

with the total number of directly adjacent neighbours n_j of unit j. However, this choice strictly limits the number of epidemic cases imported from other regions $y_{j,t-1}$ $(with\, j \neq i)$ during consecutive time points to directly adjacent neighbours. Thus, ignoring the spread through long distance traffic of humans. Therefore, Meyer and Held (2014) incorporate a power-law distance decay approach for the weights, as follows:

$$w_{ji} = \begin{cases} o_{ji}^{-d} & \text{for } j \neq i \\ 0 & \text{for } j = i \end{cases} \tag{46}$$

where o_{ji} corresponds to the adjacency order of j and i, while $d > 0$ is a decay parameter. Additionally, the weights are normalized so that $\sum_{k=1}^{I} w_{jk} = 1$ for all rows j of the weight matrix. When the decay parameter moves towards ∞, the weights mimic the previous first-order weight choice. Whereas a decay parameter of zero corresponds to equal weights for every region. The endemic, autoregressive and neighbourhood mean components are each parametrized by a parameter $(\upsilon_{i,t}, \lambda_{i,t}, \phi_{i,t})$, which can be modelled as a log-linear predictor. Moreover, this creates a rich regression model, with the following mathematical expressions:

$$log(\upsilon_{i,t}) = \alpha_i^{(\upsilon)} + b_i^{(\upsilon)} + \beta^{(\upsilon)^T} z_{it}^{(\upsilon)}, \tag{47}$$

$$log(\lambda_{i,t}) = \alpha_i^{(\lambda)} + b_i^{(\lambda)} + \beta^{(\lambda)^T} z_{it}^{(\lambda)}, \tag{48}$$

$$log(\phi_{i,t}) = \alpha_i^{(\phi)} + b_i^{(\phi)} + \beta^{(\phi)^T} z_{it}^{(\phi)}, \tag{49}$$

where each intercept $\alpha_i^{(\cdot)}$ can be modelled fixed, unit-specific or random (Meyer et al., 2017). Although fixed unit-specific intercepts account for heterogeneous disease transmission and incidence levels not explained by covariates, for example due to underreporting (Bernard et al., 2014), they become infeasible for moderate to high numbers of different regions (Paul and Held, 2011). Thus, Paul and Held (2011) introduce random effects $b_i = (b_i^{(\upsilon)}, b_i^{(\lambda)}, b_i^{(\phi)})^T$, which are assumed to follow a trivariate normal distribution with mean zero and a positive definite covariance matrix $\Sigma = diag(\sigma_\upsilon^2 I, \sigma_\lambda^2 I, \sigma_\phi^2 I)$. Furthermore, the covariance matrix is

parameterized by unknown variance parameters $\sigma^2_{(\cdot)}$ that are further multiplied by the identity matrix I (Paul and Held, 2011). Even though the preceding covariance matrix assumes that the three components are uncorrelated, it is possible to implement a differently specified covariance matrix, which includes correlations between the three random effects. Lastly, the log-linear predictors are modelled using parameters $\beta^{(\cdot)}$ which are multiplied by exogenous covariate vectors $z_{it}^{(\cdot)}$, both are component specific (Meyer and Held, 2016). Besides additional explanatory variables, the covariate vectors often incorporate temporal sine-cosine effects, as they are intended to represent seasonal variations of incidence levels (Meyer and Held, 2014). Moreover, Held and Paul (2012) suggest that it may be favourable to incorporate seasonal effects in each of the three components, whereas it was previously only implemented in the endemic component.

5 Data

This chapter summarises the data set used for our evaluation study. First, we portray the target time series before we introduce the associated covariates. Lastly, we display selected regions that were chosen to compare, evaluate and visualize model performances in the following results chapter.

First, we obtained our target time series $y_{i,t}$ from the SurvStat@RKI database[4], which is administrated by the Robert Koch Institute in Germany. These series represent counts of influenza cases of different regions $i = 1, \ldots, 411$ aggregated over weekly time points $t = 1, \ldots, 1148$, leading to a total of 471.828 data elements. While each region i corresponds to one of 411 German districts, also known as “Landkreise” (LK) and “Stadtkreise” (SK), the time points t are in a weekly format, i.e. counts are aggregated over the week and reported on Sundays. The reporting period starts with the week of the 7th of January 2001 and ends with the week of the 1st of January 2023. The target time series are continuously reported due to the German reporting obligation of direct influenza virus detection. Figure 12 depicts the time series aggregated over all regions, this and some of the following plots were created with the “matplotlib”[5] python package (Hunter, 2007).

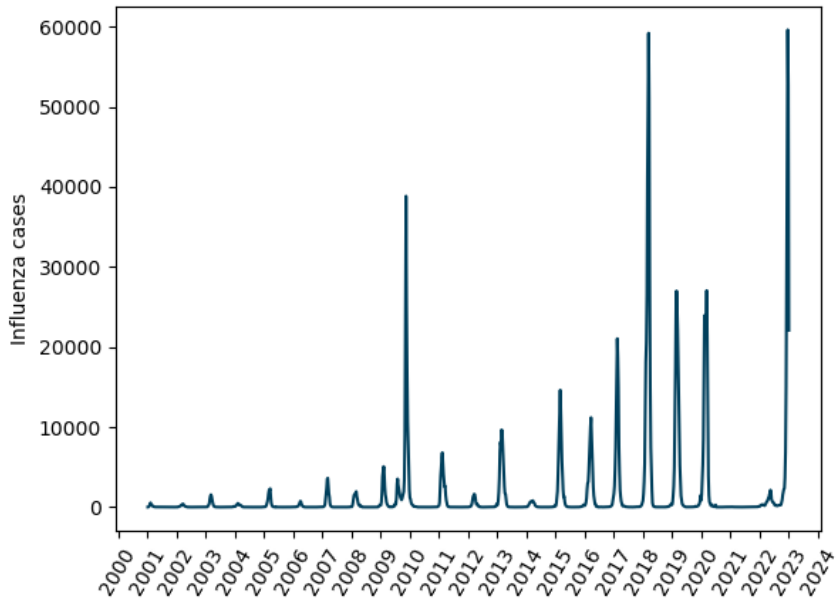

Figure 12: Visualization of the time series aggregated over all regions.

In Figure 12 it is visible that COVID-19 sanctions, which started in the year 2020, also influenced the spread of influenza, as we can observe a clear stagnation after the annual peak in early 2020. Thus, we decided to restrict our analysis to weeks before the 30th of

[4] https://survstat.rki.de/
[5] https://matplotlib.org/

September 2020, as this includes the peak of 2020 and leaves out the periods of strong regulations. Additionally, we can observe that the influenza counts of years before 2009 are comparably lower, since no year has case counts over ten thousand, whereas after 2008 peaks are almost exclusively over ten thousand cases. An exception to this rule are the years of 2012 and 2014. Lastly, it is relevant to point out that the year of 2009 exhibits a double peak, i.e. a peak during its beginning and end respectively.

Next, we implemented a binary neighbourhood matrix $W_{i,j}$ with $i = 1,\ldots, 411$ and $j = 1,\ldots, 411$ describing the direct connections between German districts. Moreover, a value of one corresponds to adjacent districts, whereas a value of zero represents districts that are not connected. The values on the diagonal of the matrix are also zero, i.e. a district is regarded as not adjacent to itself. We obtained the matrix by determining each district's neighbours within a shapefile from the year 2011 derived from the "Bundesamt für Kartographie und Geodäsie"[6]. Figure 13b illustrates that the neighbours have been determined correctly, shown by the example of "Landkreis Karlsruhe". In addition, Figure 13a depicts the general shapefile that contains all German districts. Here, it is visible that Berlin is regarded as a single district, in contrast to the influenza time series, which incorporate a division of this district into multiple smaller regions. Thus, we manually inserted the direct connections of Berlin's smaller districts to themselves and to their neighbours outside of Berlin into the neighbourhood matrix, while padding remaining matrix entries with zero.

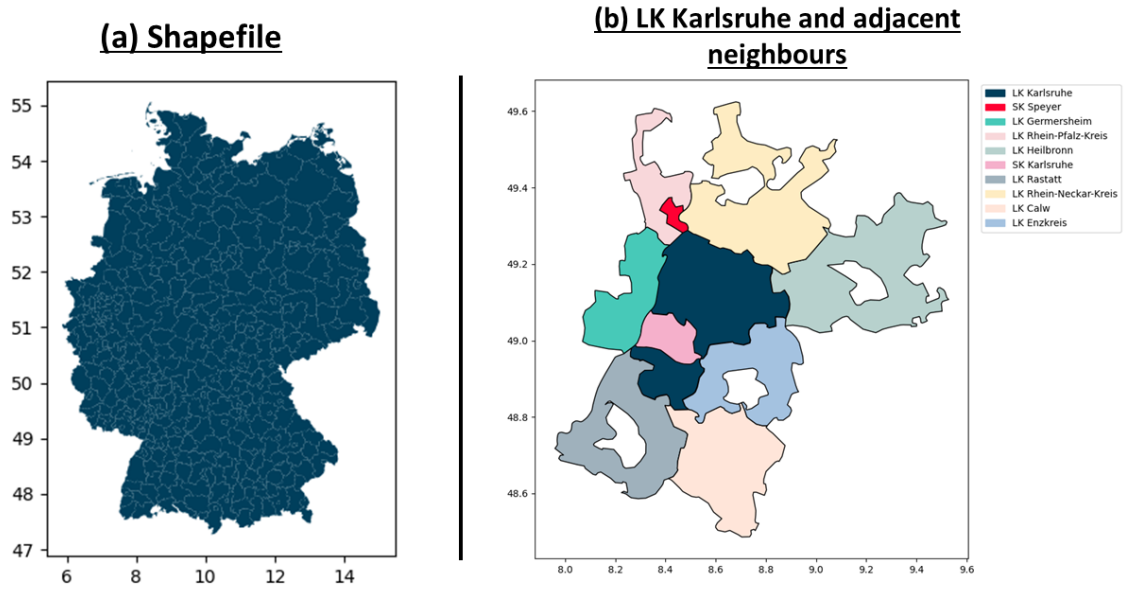

Figure 13: Illustration of (a) the overall shapefile regarding all German districts and (b) the shapefile covering "LK Karlsruhe" and its directly adjacent neighbouring regions.

[6] https://gdz.bkg.bund.de/

Lastly, we incorporated population data in form of a vector e_i with $i = 1, \ldots, 411$ representing each German district. We first obtained population data from the "Statistische Bundesamt (Destatis)"[7], this included population data from the year 2000 up to the year 2021 for all German districts except Berlin's districts, which were again reported as one singular region. Thus, we further incorporated population data from the "Amt für Statistik Berlin-Brandenburg"[8]. However, the available population data for Berlin's districts is limited to the years of 2005 to 2013. Therefore, we decided to incorporate a population vector of a single year instead of a population matrix. Moreover, we chose a population vector based on data from the year 2011. Since 2011 lies central on our observation spectrum and we expect smaller differences to years before and after 2011 as the differences in time are also smaller.

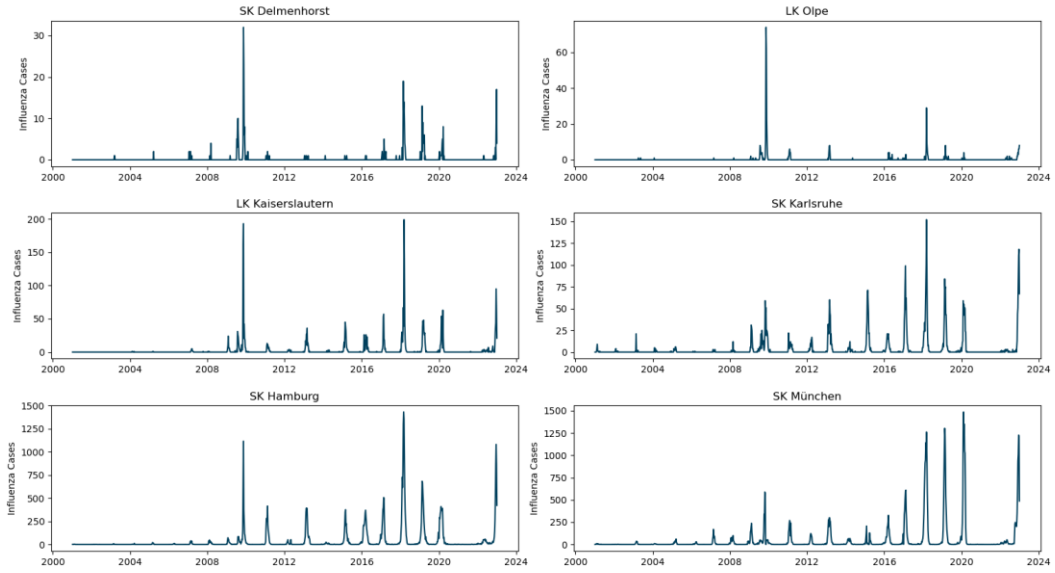

Figure 14: Depiction of the influenza count time series of six different regions. The first two plots, titled "SK Delmenhorst" and "LK Olpe", represent time series with the lowest mean influenza cases. In contrast, the last two time series, namely "SK München" and "SK Hamburg", correspond to time series with the highest number of average influenza cases.

To present, analyse and visualize individual predictions of different models in the next chapter, we selected six districts based on the mean of the corresponding influenza time series, they are depicted in Figure 14. Moreover, "SK Delmenhorst" and "LK Olpe" represent the two regions with the lowest mean influenza cases, whereas "SK München" and "SK Hamburg" correspond to the two districts with the highest average influenza counts. In addition, we included "SK Karlsruhe" and "LK Kaiserslautern" since their values are close to the mean and median of the time series averages over all regions respectively. In the next section, we will introduce the model setup and present our results.

[7] https://www.destatis.de/
[8] https://statis.statistik-berlin-brandenburg.de/webapi/

6 Results

At the beginning, Section 6.1 summarises our setup and methodology. Then, we attempt to answer the question if it is reasonable to implement the DeepAR and FNN model without prior knowledge as an “off the shelf” implementation, hence we investigate the performance of their default model and compare it to the baseline approach in Section 6.2. Furthermore, default models are fit with hyperparameters at default values, thus they have not been fine-tuned for the specific task at all and are likely to perform worse compared to fine-tuned models. After the evaluation of default models, we are going over models that have been optimized on the test set in Section 6.3. Here, we want to answer the question whether exhaustive tuning processes are beneficial for our models on this task by evaluating and comparing each model on the validation set.

6.1 Setup

To begin with, we have decided to truncate the available data as mentioned in Chapter 5. To be specific, Figure 15 visualizes the truncated total influenza count time series separated into train, test and validation set. Here, we train our model on data up to the week of the 25th of September 2016. After that, we evaluate, compare and optimize the performances of individual models on the test data starting with the week of the 2nd of October 2016 and ending with the week of the 30th of September 2018. Lastly, we compare and evaluate models on the validation period, which starts with the week of the 7th of October 2018 and ends with the week of the 27th of September 2020. It is obviously important that the validation data is not incorporated into model adjustment or selection processes, as model performances would not be independent from the validation data anymore.

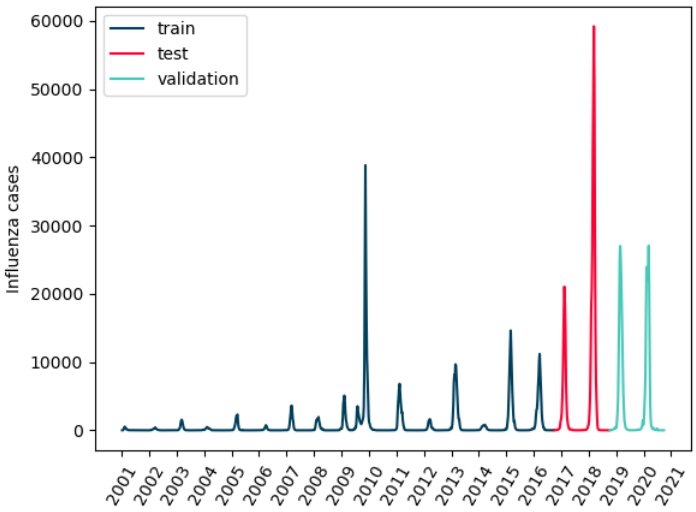

Figure 15: Image of the total influenza count time series divided into the train (blue), test (red) and validation (green) set. The series is further truncated to end with the validation set.

The forecasts for the testing and validation period are determined through a rolling window approach, where we first determine forecasts for the next four weeks before we shift the prediction window by one week into the future and repeat this prediction process for the whole forecasting period. After obtaining the forecasts, we report them in a quantile format with the selected quantile levels $0.025,\ 0.1,\ 0.25,\ 0.5,\ 0.75,\ 0.9$ and 0.975.

We incorporated the Gluon Time Series Toolkit (GluonTS) python library to implement the NNs introduced in Chapter 4. The GluonTS toolkit was specifically designed to quickly build complex deep learning-based time series models in the context of probabilistic modelling (Alexandrov et al., 2019). However, before we implement the NN models, we have to process the data into the required format. First, for each target time series, GluonTS requires a dictionary that contains a “target” and a “start” field, corresponding to the values of the time series and the start point of the time series respectively. Next, the dictionaries are combined into a list. Since it is assumed that the time series share the same spacing in between time steps, i.e. an identical frequency, the “freq” parameter is specified outside of the dictionary. Lastly, some optional fields may be included such as “feat_dynamic_real”, “feat_static_real” and “feat_static_cat”, which represent fields for time varying real valued, time independent real valued and time independent categorical covariates respectively. In our application, we regularly insert the week of the year into “feat_dynamic_real” and the population vector into “feat_static_real”. Besides that, we also incorporate the neighbourhood matrix by creating a binary vector out of every column of the neighbourhood matrix, e.g. we create a vector labelled “SK Karlsruhe” that contains all 411 matrix entries of the corresponding neighbourhood column while preserving the order of the entries, which we insert into the “feat_static_cat” field. Note that GluonTS also provides resources to directly transform DataFrame-objects from the “pandas”[9] python library (The Pandas Development Team, 2022) into the right format. After constructing the data sets for the training and testing period, we determine a rolling data set version of the testing set. Then we fit an estimator object, for example the DeepAREstimator, to the training data. The fit estimator is then inserted into a function that returns iterator objects for a variable number of separate predictions and corresponding true values for the testing period, we generally set this number to 100. After that we must unpack the iterators into lists, which is very time expensive, note that we were not able to decrease the computation times of this step. In the end, the predictions are evaluated and visualized. Different models are implemented through different estimators, while the remaining steps are kept the same.

Lastly, our baseline model is incorporated as a simplified version of the statistical model described in Section 4.4. We implemented it with the open-source R (R Core Team, 2023)

[9] https://pandas.pydata.org/

package “surveillance”[10] (Meyer et al., 2017; Salmon et al., 2016). Here, we first had to format the data into a surveillance time series object of the S4-class, which combines our influenza, population and neighbourhood data. In addition, we also had to specify the start and frequency of the time series as the first week of 2001 and 52 weeks per year respectively. Next, we defined the log-linear predictors contained in the predictive mean of the model, see Equation (44), as:

$$log(\upsilon_{i,t}) = \alpha_i^{\upsilon} + \beta^{\upsilon} sin(\omega t) + \gamma^{\upsilon} cos(\omega t) + \delta t, \tag{50}$$

$$log(\phi_{i,t}) = \alpha_i^{\phi} + \beta^{\phi} sin(\omega t) + \gamma^{\phi} cos(\omega t) + log(e_i), \tag{51}$$

where the random intercept terms $\alpha_i^{(\cdot)} \sim^{iid} \mathcal{N}(0, \sigma_{(\cdot)}^2)$ are implemented via the $ri(\cdot)$ function. $\beta^{(\cdot)}$, $\gamma^{(\cdot)}$ and δ represent the parameters of the model, while $\omega = \frac{2\pi}{freq} = \frac{2\pi}{52}$ corresponds to Fourier frequencies incorporated within the sine and cosine effects. Moreover, we used the function $addSeason2formula(\cdot)$ from the “surveillance” package to add the sine and cosine terms, enabling us to model seasonal variation. Note that we opted to use only one set of sine and cosine effects per predictor, as this keeps the model’s complexity considerably low. In addition to the already prominent population offset in the endemic component, we also introduced a population offset $log(e_i)$ into the neighbourhood component. Another important change compared to the model in 4.4 is that we omitted the autoregressive predictor from the model. Instead, we accounted for previous cases of the same region through the neighbourhood component by adding a constant scalar of one to each entry of the neighbourhood matrix. Moreover, this change can be interpreted as introducing each region as its own direct neighbour, while also increasing the order of previous neighbours by one, e.g. first order neighbours become second order neighbours and so on. This not only reduces the total amount of parameters but also ensures that seasonal terms of the endemic component are identical, as it is no longer split up into two parts. After we fit the model to the training data, we use the “hhh4addon”[11] package to implement a rolling window approach by predicting the next four time steps given the current true underlying value, while iteratively increasing the inserted value. Lastly, the model’s predictions are output as quantile forecasts for our defined quantile levels.

6.2 Default models and their results

In this section we begin by defining the default DeepAR and default FNN model. Then, we present some applied evaluation metrics to the model’s predictions, which are derived by following the previously explained setup of Section 6.1.

[10] https://surveillance.r-forge.r-project.org/
[11] https://github.com/jbracher/hhh4addon

First, the default DeepAR model is implemented by utilising the “DeepAREstimator” from GluonTS. Although most parameters of the default DeepAR model possess default values, some are task specific and need to be determined in advance. One of them is the frequency, corresponding to the frequency of the data. In our application this parameter is set to “W-Sun”, as our target data is reported in a weekly format on Sundays. In addition, the prediction length of the model represents the amount of time steps we want to forecast. Because we want to produce forecasts for the next one to four weeks, we set the prediction length to four. In contrast to the regular default DeepAR model, we additionally set the output distribution to a negative binomial distribution. Moreover, we deemed the default standard gaussian distribution as implausible and inferior to the negative binomial distribution, as we work with count data, which is non-negative.
Secondly, we incorporate the default FNN model with the “SimpleFeedForwardEstimator”. Similar to our default DeepAR model, the default FNN was also determined with a prediction length of four and a negative binomial output distribution, note that the FNN does not have a frequency parameter. Additionally, the FNN works with given covariates without additional model specifications. In contrast, the DeepAR model incorporates the parameters “use_feat_dynamic_real”, “use_feat_static_real” and “use_feat_static_cat”, which determine whether the model is allowed to use the “feat_dynamic_real”, “feat_static_real” and “feat_static_cat” fields from the data respectively. Since these parameters are standardly set to “False”, the default DeepAR model does not include covariates.

After conducting model runs according to the previously explained methodologies, we receive the predictions for the test window. However, although the window size of the one- to four-week ahead forecasts is equal, their start and end point differ from each other, for example the first one-week ahead predictions are for the 2nd of October 2016, whereas the first four-week ahead predictions begin three weeks later. Thus, we have decided to truncate the predictions, so that they simultaneously begin with the four-week ahead predictions and end with the last one-week ahead forecasts. This ensures that the different week ahead predictions are evaluated over a uniform basis.

The mean WIS in Table 1 was determined by computing the quantile loss, see Equation (5), for the defined quantile levels of a unique location and week ahead forecast combination. Then, we calculated the mean over all quantiles and all locations. This results in the mean WIS of the one- to four-week ahead predictions, the overall average is then calculated as the mean of these four WIS. Table 1 suggests that the default DeepAR model performs best, as it has the lowest mean WIS for each of the one- to four-week ahead forecasts. Furthermore, we see that the NN models are performing particularly well on the one-week ahead forecasts compared to the hhh4 model. However, as we move closer to the four-week ahead

predictions, the margins of improvement over the benchmark become smaller for the DeepAR model. Whereas the FNN model already has the worst performance of the three models beginning with the two-week ahead forecast. Interestingly, the FNN's three-week ahead predictions perform worse than the four-week ahead forecasts of the DeepAR and the hhh4 model.

	Mean WIS				
	1-Week Ahead	**2-Week Ahead**	**3-Week Ahead**	**4-Week Ahead**	**Overall Average**
Default DeepAR	**279.03**	**394.80**	**491.59**	**577.03**	**435.61**
Default FNN	295.89	464.19	627.08	783.19	542.59
hhh4	342.83	421.26	512.50	601.90	469.62

Table 1: Summary of the average WIS scores of the DeepAR, FNN and hhh4 model on the test data. The lowest scores are highlighted.

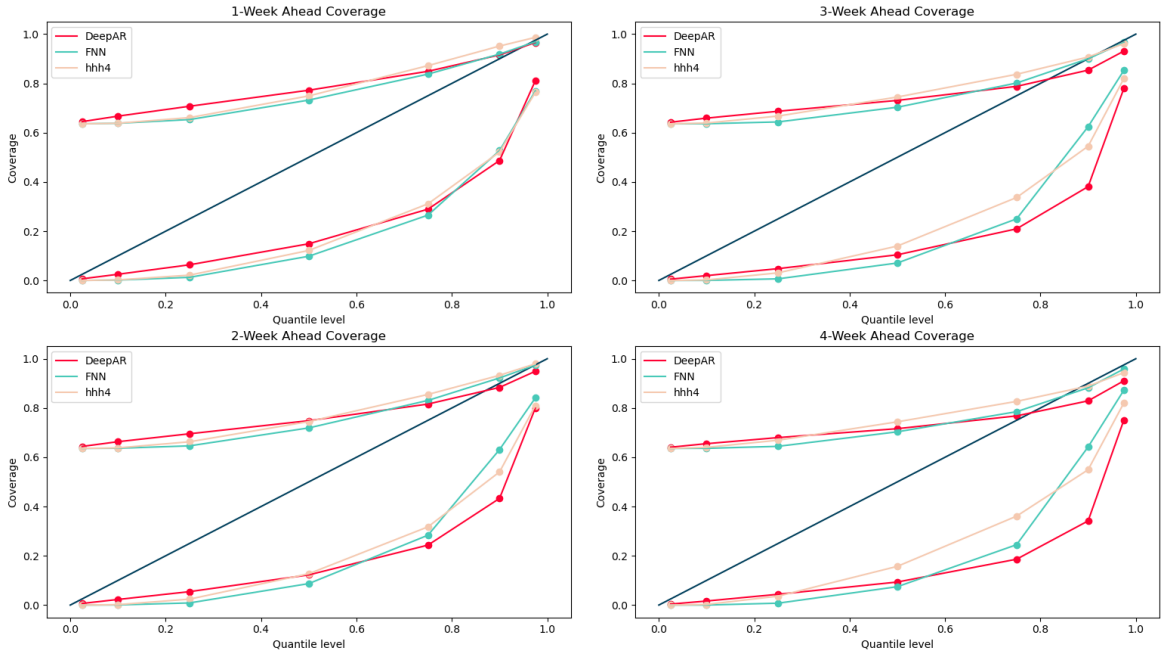

Figure 16: Visualization of the coverage results of the DeepAR, FNN and hhh4 model on the test set. The diagonal represents the optimal coverage, whereas the lower and upper plot correspond to the lower and upper coverage respectively.

To assess calibration, we compare the different week ahead predictions with regard to the coverage of the defined quantiles, see Figure 16. For each week ahead forecast, we display the lower and upper coverage, as proposed for count data in Gneiting et al. (2023). Additionally, the individual coverage values are also provided in Tables 5 and 6 in the appendix. While the upper coverage takes observations into account that are smaller or

equal to the predicted quantile, the lower coverage only accepts observations that are smaller than the predicted quantile. It is visible that the gap between both plots is relatively large for every model but gets smaller as we move towards higher quantile levels. This large discrepancy likely originates from the high number of zeros between influenza seasons within the testing data, see Figure 15. Furthermore, due to the incorporated negative binomial output distributions, we always predict nonnegative values. Thus, a materialized observation of zero can be interpreted as an observation that is always counted by the upper coverage version, as we always predict values greater or equal to zero. However, the lower coverage only takes zeros into account, when the predicted quantile is strictly larger than zero. On a different note, the lower coverage plots suggest that the DeepAR model is best for lower quantile levels, as the values are closest to the diagonal. Whereas the hhh4 model performs better for higher quantile levels and the FNN performs best at the highest quantile levels for the two- to four-week ahead forecast. Furthermore, the intersection between the model's coverages shifts towards lower quantiles for higher week ahead forecasts, for example the DeepAR model still performs best for the first four quantile levels of the one-week ahead forecast before it is overtaken by the hhh4 model, whereas the hhh4 model is already better than the DeepAR model on the 0.5 quantile level of the two-week ahead forecast. This again shows that the performance of the DeepAR model is comparably better for short-term forecasts, while the performance of the hhh4 model seems to deteriorate less quickly for longer forecast horizons. Lastly, we want to add that the upper coverage values of the hhh4 model seem to get better when we move from the one- to the four-week ahead predictions. Similarly, the FNN's lower coverage values for the highest quantile level rise for longer forecast horizons. Table 6 in the appendix supports both points more clearly.

We illustrate the one-week ahead predictions of each model for the selected locations in Figure 17, note that we also provide the associated two- to four-week ahead forecasts in Figures 23, 24 and 25 in the appendix. First, Figure 17 suggests that the prediction intervals of the FNN and hhh4 model are larger than the PIs of the DeepAR model for all locations except "SK Delmenhorst", where it can be argued that the hhh4's PIs are slightly smaller. Thus, it may be proposed that the sharpness of the DeepAR model is the best out of the three, as it also possesses the lowest WIS score, while having similar coverage values to the other two models for the one-week ahead forecasts. Next, the larger peak of 2018 seems to be harder to predict than the peak in 2017, as the median forecast of all three models is almost exclusively lower than the peak of 2018. In addition, the margins between median forecast and true observations are much wider for the 2018 peak than for the peak in 2017. This is particularly evident for the hhh4's median forecast, which is regularly too low and often deviates from the week with the highest cases, i.e. its predicted highest value lies regularly in weeks after the actual highest value. A good example for this matter is the 2018

peak of the “LK Kaiserslautern” time series, where the hhh4 model predicts a gradual ascent of cases, thus failing to point out each of the three sudden peaks. In contrast, the DeepAR and FNN predictions follow the peak’s curve more closely. Aside from that it is visible that the median of all three models follows the periods with low influenza cases equally well. However, the PIs of the DeepAR model are often relatively large for these periods, this becomes clear for the “SK Delmenhorst” time series, but it is also observable for other time series. Thus, it can be argued that the hhh4 and FNN model are advantageous for these periods, as they are less likely to make this error.

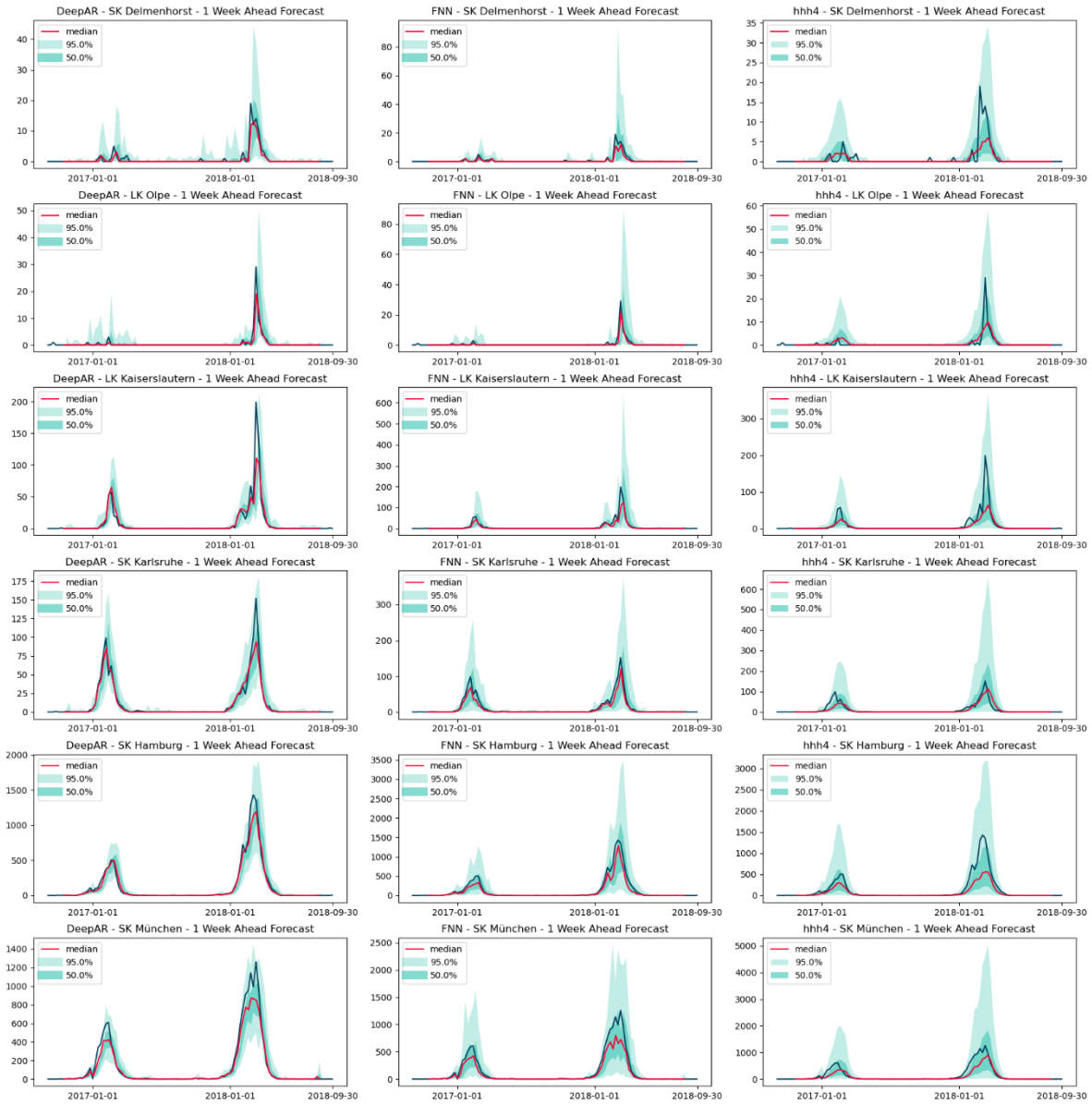

Figure 17: One-week ahead predictions by the default DeepAR, FNN and hhh4 model for selected regions of the test data set. The red line represents the median of the predictions, while the dark and light green surfaces correspond to predictions that lie within the 50th and 95th prediction interval respectively. The blue lines equal the true underlying observations.

On a different note, Figure 18 visualizes the results of 100 model runs of each default model. Here, Figure 18a illustrates the average WIS of each model, i.e. the mean WIS over the one- to four-week ahead forecasts. The previously covered single model run of the default DeepAR model has an average WIS of 435.61 over all forecasts. This corresponds to a slightly better run than the median of the 100 model runs. In contrast, the previous single FNN run has an average WIS of 542.59, which is near the maximum of the FNN's boxplot. This indicates that our previous results are based on a poorly performing default FNN and a good performing default DeepAR model. Nevertheless, we can argue that the default DeepAR model is still clearly superior to the default FNN model in terms of the average WIS of the 100 model runs, but it also exhibits more variability between individual performances. Additionally, Figure 18b shows that in general the default FNN has a faster computation time than the default DeepAR model, as the median of the DeepAR's computation time is almost equal to the maximal model time of the FNN's computation time boxplot.

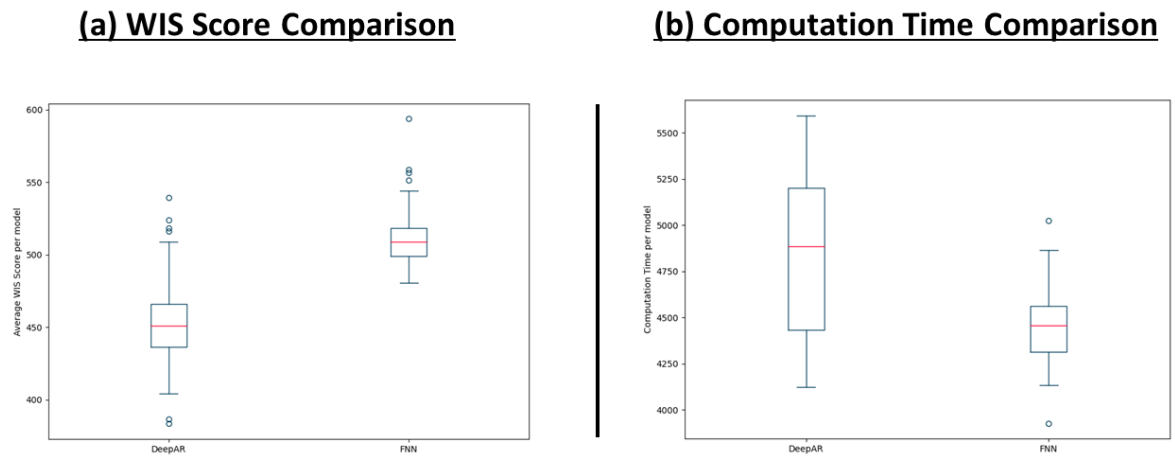

Figure 18: Summary of the (a) average one- to four-week ahead WIS Scores and (b) Computation Times (measured in seconds) of 100 model runs on the test set of the default DeepAR and the default FNN.

Conclusively, we have been able to show that the default DeepAR model produces reasonable results for our task, even without further tuning. However, one may be reluctant to implement the default FNN model, as it can be argued that the default FNN results are worse than those of the baseline model, especially for longer forecasting horizons. Next, we want to compare the performance of tuned and default models while assessing whether the tuning process is worth the effort.

6.3 Tuned models and their results

This chapter finalizes our experiments by firstly going over the tuning process and models that we derived from it. After that, we evaluate and compare the predictive performances of the different models on the validation data.

The tuning process was performed by incorporating the "ray tune"[12] python library (Liaw et al., 2018). Although ray tune allows us to implement and combine various parameter search techniques, we decided to solely implement grid searches of the parameter space. Thus, the first step for every tuning session was to define parameter values, where each unique combination of parameters is successively inserted into the corresponding estimator object. Then, we follow the usual training, prediction and estimation process. However, we only use the average of the one- to four-week ahead WIS to assess the quality of a model run. Since each NN model is also relatively variable, see Figure 18, we run each parameter combination five times and determine the mean WIS over all five model runs.

Table 2 displays the subset of the DeepAR model's hyperparameters with the associated values that were tried out during our tuning process. Here, "num_cells" and "num_layers" define the number of cells per layer and number of layers of the model respectively. The "context_length" parameter represents the length of the unrolled decoder RNN. Besides that, the "cell_type" hyperparameter defines the implemented type of cell, as described in Section 4.3, the recurrent cells of the DeepAR model are typically implemented as either LSTM or GRU cells. The "epochs" parameter determines how many training iterations the model is allowed to go through. Lastly, we can decide if the model has access to provided features via the "use_feat_static_real", "use_feat_dynamic_real" and "use_feat_static_cat" parameters. Moreover, the "cardinality" parameter only needs to be provided if the "use_feat_static_cat" parameter equals "True", as it presents the model with information about the dimensionality of the categorical features.

	DeepAR
Hyperparameters	**Parameter Combinations**
num_cells	10, 20, 40, 60, 80, ***140***
num_layers	1, 2, 3, 4, 5, ***6***, 8, 10, 12
context_length	1, ***2***, 4, 8, 52, 104
cell_type	***LSTM***, GRU
epochs	8, 20, 30, 40, 60, 80, 90, 100, 140, ***200***
use_feat_static_real	True, ***False***
use_feat_dynamic_real	***True***, False
use_feat_static_cat	True, ***False***
cardinality	[2, 2, 2, …, 2, 2, 2], ***None***

Table 2: Summary of the DeepAR's hyperparameters. Underlined values represent the default values, whereas the bold and italic values correspond to the best discovered parameter combination.

[12] https://docs.ray.io/en/latest/tune/index.html

Likewise, we performed grid searches of the subset of the FNN's hyperparameter space, see Table 3. Here, the "epochs" parameter is identical to the associated DeepAR parameter. However, the "context_length" of the FNN corresponds to the amount of time steps that condition the forecasts, which is different from the "context_length" of the DeepAR model. In addition, we adjusted the "num_hidden_dimensions" parameter of the FNN, which defines the number of hidden nodes in each layer. The remaining parameters, i.e. "num_batches_per_epoch", "batch_normalization" and "batch_size", determine the number of batches per iteration, if normalization is applied to batches and the size of batches respectively.

	FNN
Hyperparameters	**Parameter Combinations**
num_hidden_dimensions	[5], [10], [20], [40, 40], [80, 80], [100, 100], [140, 140], [180, 180], [200, 200], ***[40, 40, 40]***, [80, 80, 80], [40, 40, 40, 40], [80, 80, 80, 80], [100, 100, 100, 100, 100, 100, 100]
context_length	2, 4, 26, 32, 52, 64, ***104***, 208
epochs	5, 10, 20, 50, 75, 100, 150, ***200***
num_batches_per_epoch	25, 30, 40, 50, ***60***, 100
batch_normalization	***True***, False
batch_size	16, 20, 32, 64, ***124***

Table 3: Summary of the FNN's hyperparameters.

We decided to solely focus on the parameters of the best model, i.e. the model with the lowest average WIS on the test set, highlighted in italic and bold in Table 2 and 3. Moreover, the best DeepAR model achieved an average WIS of 352.98 during tuning on the test set, note that the median WIS of the default DeepAR in Figure 18 corresponded to about 450. Furthermore, the average WIS of the best FNN model was 381.70, whereas the median of the default FNN corresponded to around 500.

Figure 19a illustrates the WIS improvement of the tuned models on the test set in a more profound fashion, as we further performed 100 model runs of the tuned models and compared them to the 100 default model runs also displayed in Figure 18. Obviously, the tuned models perform better on the test set, as their parameters have been optimized for it. In addition, Figure 19b suggests that both tuned models occupy less time for the fitting process than their default counterpart.

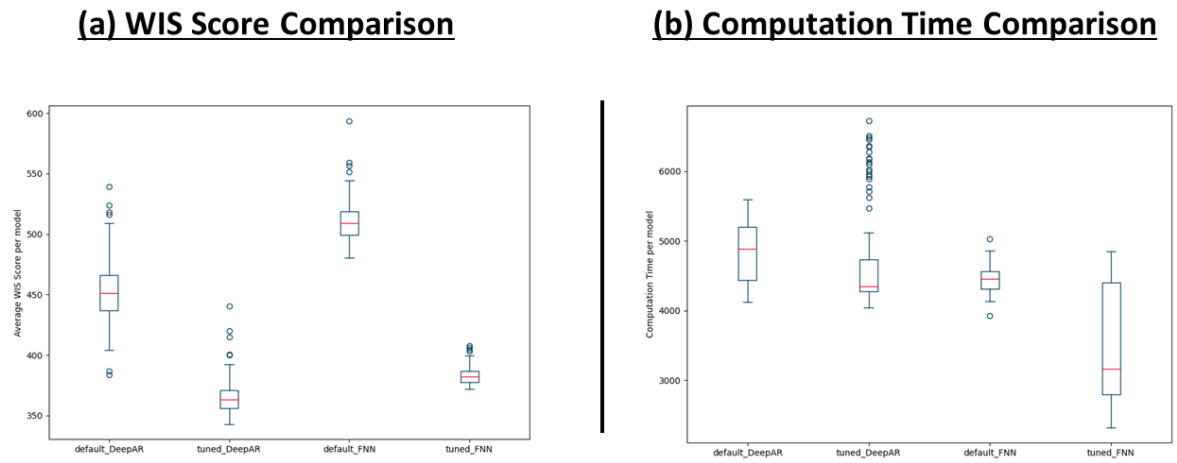

Figure 19: Illustration of the (a) average one- to four-week ahead WIS Scores and (b) Computation Times (measured in seconds) of 100 model runs on the test set of the default DeepAR, tuned DeepAR, default FNN and tuned FNN.

	Mean WIS				
	1-Week Ahead	**2-Week Ahead**	**3-Week Ahead**	**4-Week Ahead**	**Overall Average**
Default DeepAR	238.90	338.59	434.09	495.71	376.82
Default FNN	245.67	369.00	512.40	689.82	454.22
hhh4	278.02	328.18	391.24	456.23	363.42
Tuned DeepAR	**236.76**	**324.91**	**377.16**	**411.60**	**337.61**
Tuned FNN	282.97	375.63	468.51	536.29	415.85

Table 4: Summary of the average WIS scores of the default and tuned DeepAR and FNN models as well as the hhh4 model on the validation data.

Table 4 shows the mean WIS of the five models and its separation into the different week ahead forecasts. First, we can assess the tuned DeepAR model as the best out of the five models, as it has the lowest mean WIS score for each week ahead forecast. In contrast to the testing period, the default DeepAR model is outperformed by the baseline model for the 2- to 4-week ahead forecast. Similarly, the default FNN is outperformed by the baseline approach, which again suggests that the default models perform well on short-term forecasts but are outperformed by the baseline on longer forecasting horizons. Furthermore, it seems that the tuned models mainly improve the long-term forecasting performance, as the margins of improvement are relatively small for smaller week ahead forecasts but increase for longer forecasting horizons. Interestingly, the tuned FNN performs worse than the default FNN on the 1- and 2-week ahead forecasts, suggesting that the tuned FNN sacrifices on short-term

forecasting performance to increase its long-term predictive ability. Similar to Table 1, the 3-week ahead performance of the FNN is worse than the 4-week ahead performance of all other models except the tuned FNN. Likewise, the tuned DeepAR and the baseline model perform better on the 4-week ahead predictions than the tuned FNN on the 3-week ahead predictions. Lastly, the tuned DeepAR 4-week ahead forecast also outperforms the default DeepAR's 3-week ahead forecast.

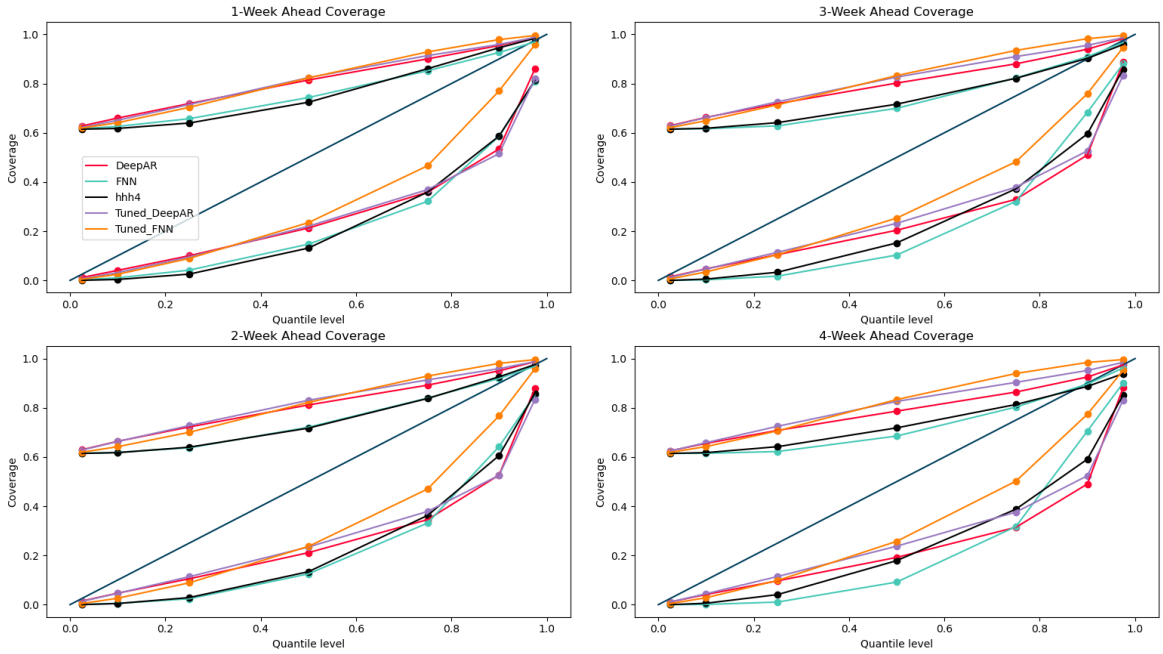

Figure 20: Lower and upper coverage plots of the validation period for all five models.

To evaluate calibration, Figure 20 illustrates the lower and upper coverages of all five models for the validation period, in addition Table 7 and 8 in the appendix provide the underlying coverage values. In Figure 20 it is visible that the tuned FNN performs best for the lower coverage at quantile levels that are higher or equal to 0.5. It is also among the best for lower quantile levels, however the DeepAR models are often slightly better. Besides that, we can argue that both tuned models perform better than the default models, however the margin of improvement is larger for the FNN models than for the DeepAR models. Although the baseline and the default FNN model are the worst models for lower quantiles of the lower coverage, they perform better than both DeepAR models on higher quantile levels. In addition, the hhh4 and the default FNN model are the best models for the upper coverage.

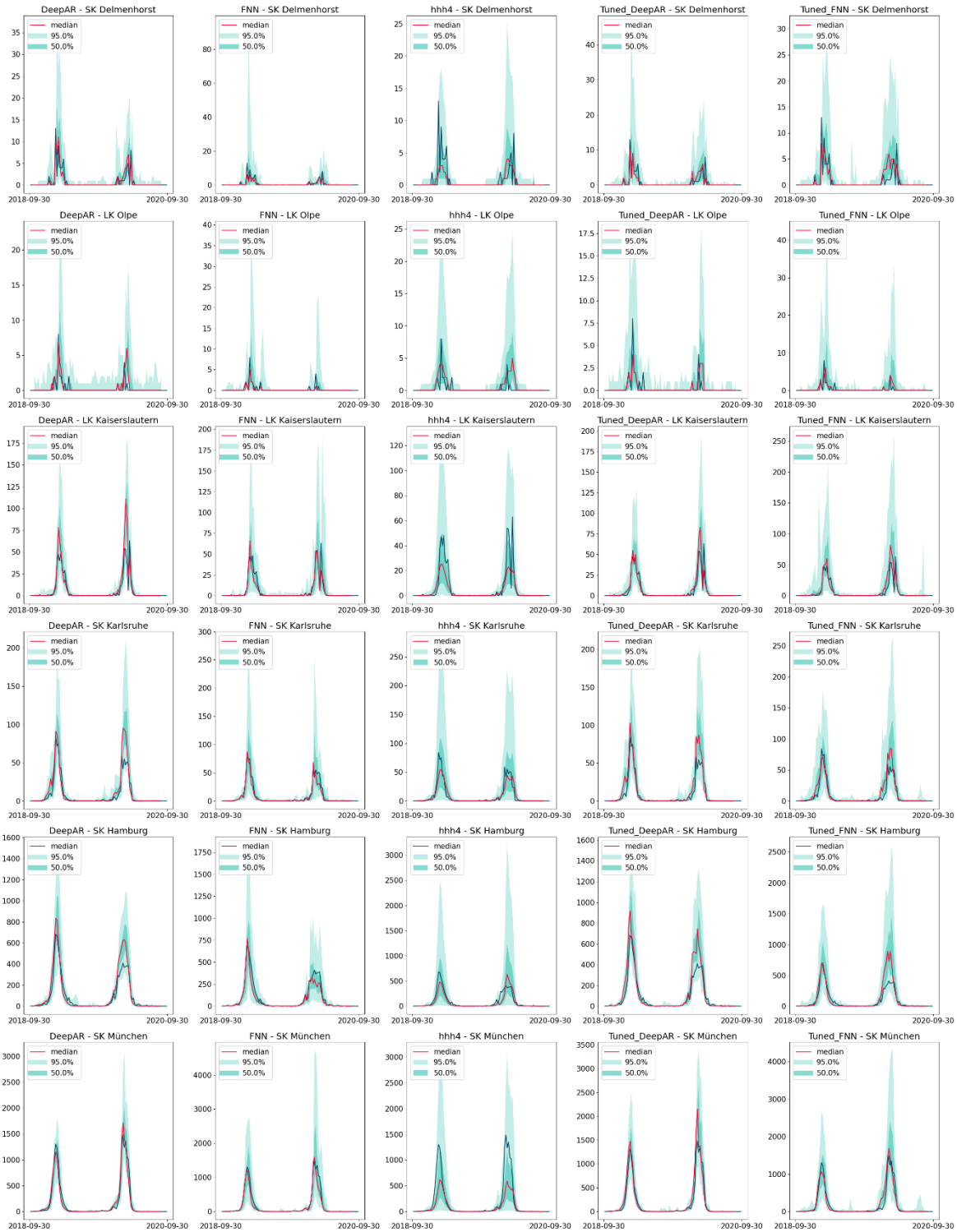

Figure 21: One-week ahead predictions of the default DeepAR, default FNN, hhh4, tuned DeepAR and tuned FNN mode for selected regions of the validation data set.

On a different note, we visualized the regional 1-week ahead predictions of the validation period of all five models in Figure 21, note that the 2-,3- and 4-week ahead predictions are visualized in Figures 26, 27 and 28 in the appendix respectively. In Figure 21, the predictions of the DeepAR models seem narrower than the predictions of the rest again, but it is hard to

assess which of the two DeepAR models produces the sharpest forecast. Another reoccurring theme is that the default DeepAR model produces comparably larger PIs for periods outside peak regions, e.g. the predictions of “LK Olpe” are almost entirely larger than zero for the 95% PI. Furthermore, the hhh4 model best avoids this mistake, as its prediction intervals are almost exclusively zero for periods in between peaks. A flaw of the hhh4 model is that its median predictions are often too low, e.g. the median forecast for “LK Kaiserslautern” or “SK München”. Moreover, for “SK München” the 50% PI is also too low. In contrast, the other four models all produce more suited median forecasts for “SK München”. Although the baseline model does not seem to perform as poorly for “SK Hamburg”, it still produces the largest PIs out of the five models for this region. Overall, this suggest that the hhh4 model is potentially unfavourable for regions with relatively high case numbers. This is also supported by the fact that the lower bound of the baseline model’s 95% PI is regularly close to values of zero during peak periods, hence its 95% PI ends up wider than those of the competing models. Aside from that, the 2020 peak of the time series of “SK Karlsruhe” and “SK Hamburg” seems difficult to forecast for some models, as the same three out of the five models produce median forecasts and 50% PIs that are larger than the true underlying observations. Solely the default FNN and the baseline model avoid this mistake.

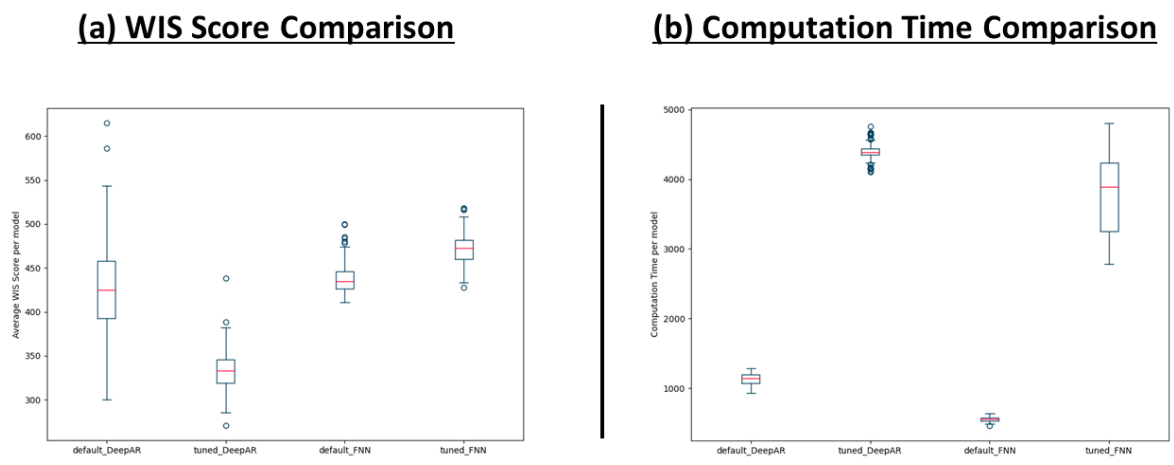

Figure 22: Summary of the (a) average one- to four-week ahead WIS Scores and (b) Computation Times (measured in seconds) of 100 model runs on the validation set of the default DeepAR, tuned DeepAR, default FNN and tuned FNN.

Lastly, Figure 22a and 22b are determined identically to Figure 19a and 19b, except the 100 different models are fit on the combination of the training and testing period and then evaluated on the validation set. Figure 22a also indicates that the tuned DeepAR model is the best model out of the four models, as it has the lowest median, maximal and minimal WIS value out of the four models. In contrast, the tuned FNN model is the worst model. Furthermore, the distinction between the second best and third best model is not as clear.

While the default DeepAR model has the lower median and minimal WIS, the default FNN has the lower maximal WIS. Overall, the validation period shows that the tuned FNN was potentially overfit on the testing period, as its good performances on the test set did not translate well to the validation set. Besides that, we want to note that the default DeepAR model has the highest WIS spread, as it has some of the best and worst model runs out of all model runs. Figure 22b shows that the default models drastically decreased their modelling time per model, whereas the tuned models slightly increased their modelling time per model. Hence, the tuned models were unable to support the claim that they are the faster models, as Figure 19b suggested earlier.

Altogether we conclude that the improvement of the tuning process mostly increases the quality of longer forecasting horizons. Although the performance of the tuned DeepAR model improves for every week ahead forecast, the improvement is more significant for longer forecasting horizons. Similarly, the tuned FNN's performance increases for longer forecasting horizons, however its short-term forecasts are worse than those of default FNN. Nevertheless, we have shown that the DeepAR model benefits from the tuning process and outperforms the other four models it competed with on our task. Thus, we evaluate the tuning process as generally beneficial, advantageous and appropriate, however we further advise to take the general trade-off between modelling time and performance into account for the specific task at hand.

7 Conclusion

In this thesis, we compared the DeepAR model to other models on the task of producing probabilistic forecasts for influenza disease cases of German districts. First, we have introduced the theoretical framework of probabilistic forecasts, especially those reported in an interval or quantile format, and their evaluation. Then, we have presented the theoretical basics of global and local forecasting models before discussing the theoretical foundation of general NNs and RNNs. Furthermore, we have laid our focus on RNNs and have presented common RNN cells and architectures. The theoretical background of this thesis has then been completed by addressing the DeepAR's and the baseline's theoretical framework.

After that, we have explained our incorporated data set and our modelling setup. Then, we have first reviewed the performance of the default DeepAR, default FNN and baseline model on the test set. Here, we have established that the default DeepAR model is generally applicable to this task by non-experts, whereas it may be unreasonable to implement the default FNN for longer forecasting horizons. After explaining the tuning process and our resulting model choice, we have assessed the predictions of the default and tuned models as well as the predictions of the baseline approach on the validation set. In summary, the tuned DeepAR model was best, while the tuned models mainly improved the long-term forecasting performance. Therefore, we conclude that the implementation of the DeepAR model is reasonable for the prediction of influenza disease cases in German districts.

Future Research

In general, our modelling approach offers much room for improvement. First, we could further include additional covariates, similar to many comparable projects, e.g. include Twitter data (Achrekar et al., 2012; Paul et al., 2014), search engine data (Fantazzini, 2020; Polgreen et al., 2008) or air traffic information (Paul et al., 2008; Colizza et al., 2007). Besides that, the global modelling approach of the DeepAR model might also benefit from similar epidemiological time series, e.g. influenza time series from other countries or time series of other diseases through multi-pathogen forecasting. In fact, numerous studies suggest that the meningococcal disease has an impact on influenza infections (Jensen et al., 2003; Cartwright et al., 1991; Jansen et al., 2008).

Often ensemble methods that combine several forecasting methods lead to better performances, for example one conclusion from the M4 competition (Makridakis et al., 2020) was that hybrid and combinatorial methods constitute the future for better forecasts. Later, the M5 competition (Makridakis et al., 2022) further affirmed the benefits of combinatorial models. Hence, an ensemble model derived from all our models or from a subset of our models could lead to better results. Furthermore, the proposed ensemble model could further

benefit from a better adjusted baseline model.
Although the DeepAR model is a global model, it does not incorporate information from other time series during inference. Thus, Sen et al. (2019) introduced the DeepGLO model, which was designed to use all available time series during training and inference. Theoretically this model should translate well to epidemiological time series forecasts, as we work with many related time series which likely also benefit from information of other time series during inference.

Overall, epidemiological predictions pose a complex problem, in which forecasting methods are going to improve and change continuously over time. Thus, it is important to contribute to the development of forecasting methods by qualitatively assessing the performance and applicability of novel models in this domain.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X., 2016. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. http://arxiv.org/abs/1603.04467

Abdulkarim, S.A., Engelbrecht, A.P., 2021. Time series forecasting with feedforward neural networks trained using particle swarm optimizers for dynamic environments. Neural Comput & Applic 33, 2667–2683. https://doi.org/10.1007/s00521-020-05163-4

Achrekar, H., Gandhe, A., Lazarus, R., Yu, S.-H., Liu, B., 2012. Twitter Improves Seasonal Influenza Prediction., in: Healthinf. pp. 61–70.

Alexandrov, A., Benidis, K., Bohlke-Schneider, M., Flunkert, V., Gasthaus, J., Januschowski, T., Maddix, D.C., Rangapuram, S., Salinas, D., Schulz, J., Stella, L., Türkmen, A.C., Wang, Y., 2019. GluonTS: Probabilistic Time Series Models in Python. http://arxiv.org/abs/1906.05264

Amt für Statistik Berlin-Brandenburg, Population data from Berlins districts from 2005 to 2013, https://statis.statistik-berlin-brandenburg.de/webapi/jsf/tableView/tableView.xhtml, last accessed on 16 May 2023.

Andersson, H., Britton, T., 2012. Stochastic epidemic models and their statistical analysis. Springer Science & Business Media.

Bahdanau, D., Cho, K., Bengio, Y., 2016. Neural Machine Translation by Jointly Learning to Align and Translate. http://arxiv.org/abs/1409.0473

Bengio, Y., 2009. Learning Deep Architectures for AI. FNT in Machine Learning 2, 1–127. https://doi.org/10.1561/2200000006

Bengio, Y., Simard, P., Frasconi, P., 1994. Learning long-term dependencies with gradient descent is difficult. IEEE Trans. Neural Netw. 5, 157–166. https://doi.org/10.1109/72.279181

Benidis, K., Rangapuram, S.S., Flunkert, V., Wang, Y., Maddix, D., Turkmen, C., Gasthaus, J., Bohlke-Schneider, M., Salinas, D., Stella, L., Aubet, F.-X., Callot, L., Januschowski, T., 2023. Deep Learning for Time Series Forecasting: Tutorial and Literature Survey. ACM Comput. Surv. 55, 1–36. https://doi.org/10.1145/3533382

Bernard, H., Werber, D., Höhle, M., 2014. Estimating the under-reporting of norovirus illness in Germany utilizing enhanced awareness of diarrhoea during a large outbreak of Shiga toxin-producing E. coli O104:H4 in 2011 – a time series analysis. BMC Infect Dis 14, 116. https://doi.org/10.1186/1471-2334-14-116

Bernardo, J.M., 1979. Expected Information as Expected Utility. Ann. Statist. 7. https://doi.org/10.1214/aos/1176344689

Bianchini, M., Scarselli, F., 2014. On the Complexity of Neural Network Classifiers: A Comparison Between Shallow and Deep Architectures. IEEE Trans. Neural Netw. Learning Syst. 25, 1553–1565. https://doi.org/10.1109/TNNLS.2013.2293637

Bishop, C.M., 1994. Neural networks and their applications. Neural networks 65.

Bosse, N.I., Abbott, S., Bracher, J., Hain, H., Quilty, B.J., Jit, M., Centre for the Mathematical Modelling of Infectious Diseases COVID-19 Working Group, van Leeuwen, E., Cori, A., Funk, S., 2022. Comparing human and model-based forecasts of COVID-19 in Germany and Poland. PLoS Comput Biol 18, e1010405. https://doi.org/10.1371/journal.pcbi.1010405

Bracher, J., Ray, E.L., Gneiting, T., Reich, N.G., 2021. Evaluating epidemic forecasts in an interval format. PLoS Comput Biol 17, e1008618. https://doi.org/10.1371/journal.pcbi.1008618

Bracher, J., Wolffram, D., Deuschel, J., Görgen, K., Ketterer, J.L., Ullrich, A., Abbott, S., Barbarossa, M.V., Bertsimas, D., Bhatia, S., Bodych, M., Bosse, N.I., Burgard, J.P., Castro, L., Fairchild, G., Fiedler, J., Fuhrmann, J., Funk, S., Gambin, A., Gogolewski, K., Heyder, S., Hotz, T., Kheifetz, Y., Kirsten, H., Krueger, T., Krymova, E., Leithäuser, N., Li, M.L., Meinke, J.H., Miasojedow, B., Michaud, I.J., Mohring, J., Nouvellet, P., Nowosielski, J.M., Ozanski, T., Radwan, M., Rakowski, F., Scholz, M., Soni, S., Srivastava, A., Gneiting, T., Schienle, M., 2022. National and subnational short-term forecasting of COVID-19 in Germany and Poland during early 2021. Commun Med 2, 136. https://doi.org/10.1038/s43856-022-00191-8

Bundesamt für Kartographie und Geodäsie, Shapefile of German districts from 2011, https://daten.gdz.bkg.bund.de/produkte/vg/vg2500/, last accessed on 16 May 2023.

Cai, R., Xu, B., Yang, X., Zhang, Z., Li, Z., Liang, Z., 2018. An Encoder-Decoder Framework Translating Natural Language to Database Queries. http://arxiv.org/abs/1711.06061

Cartwright, K.A.V., Jones, D.M., Kaczmarski, E., Smith, A.J., Stuart, J.M., Palmer, S.R., 1991. Influenza A and meningococcal disease. The Lancet 338, 554–557. https://doi.org/10.1016/0140-6736(91)91112-8

Casagrandi, R., Bolzoni, L., Levin, S.A., Andreasen, V., 2006. The SIRC model and influenza A. Mathematical Biosciences 200, 152–169. https://doi.org/10.1016/j.mbs.2005.12.029

Cervera, J., Munoz, J., 1996. Proper scoring rules for fractiles. Bayesian statistics 5, 513–519.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y., 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. http://arxiv.org/abs/1406.1078

Chung, J., Gulcehre, C., Cho, K., Bengio, Y., 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. http://arxiv.org/abs/1412.3555

Clark, S.R., Pagendam, D., Ryan, L., 2022. Forecasting Multiple Groundwater Time Series with Local and Global Deep Learning Networks. IJERPH 19, 5091. https://doi.org/10.3390/ijerph19095091

Colizza, V., Barrat, A., Barthélemy, M., Vespignani, A., 2007. Predictability and epidemic pathways in global outbreaks of infectious diseases: the SARS case study. BMC Med 5, 34. https://doi.org/10.1186/1741-7015-5-34

Cramer, E.Y., Ray, E.L., Lopez, V.K., Bracher, J., Brennen, A., Castro Rivadeneira, A.J., Gerding, A., Gneiting, T., House, K.H., Huang, Y., 2022. Evaluation of individual and

ensemble probabilistic forecasts of COVID-19 mortality in the United States. Proceedings of the National Academy of Sciences 119, e2113561119.

Crone, S.F., Hibon, M., Nikolopoulos, K., 2011. Advances in forecasting with neural networks? Empirical evidence from the NN3 competition on time series prediction. International Journal of Forecasting 27, 635–660. https://doi.org/10.1016/j.ijforecast.2011.04.001

Daley, D.J., Gani, J.M., 2005. Epidemic modelling: an introduction, Repr. ed, Cambridge studies in mathematical biology. Cambridge Univ. Press, Cambridge.

Dongare, A.D., Kharde, R.R., Kachare, A.D., 2012. Introduction to artificial neural network. International Journal of Engineering and Innovative Technology (IJEIT) 2, 189–194.

Elman, J.L., 1990. Finding Structure in Time. Cognitive Science 14, 179–211. https://doi.org/10.1207/s15516709cog1402_1

Elsheikh, A.H., Saba, A.I., Elaziz, M.A., Lu, S., Shanmugan, S., Muthuramalingam, T., Kumar, R., Mosleh, A.O., Essa, F.A., Shehabeldeen, T.A., 2021. Deep learning-based forecasting model for COVID-19 outbreak in Saudi Arabia. Process Safety and Environmental Protection 149, 223–233. https://doi.org/10.1016/j.psep.2020.10.048

Fantazzini, D., 2020. Short-term Forecasting of the COVID-19 Pandemic using Google Trends Data: Evidence from 158 Countries. SSRN Journal. https://doi.org/10.2139/ssrn.3671005

Farajzadeh, J., Fakheri Fard, A., Lotfi, S., 2014. Modeling of monthly rainfall and runoff of Urmia lake basin using “feed-forward neural network” and “time series analysis” model. Water Resources and Industry 7–8, 38–48. https://doi.org/10.1016/j.wri.2014.10.003

Ferguson, N.M., Cummings, D.A.T., Cauchemez, S., Fraser, C., Riley, S., Meeyai, A., Iamsirithaworn, S., Burke, D.S., 2005. Strategies for containing an emerging influenza pandemic in Southeast Asia. Nature 437, 209–214. https://doi.org/10.1038/nature04017

Geilhufe, M., Held, L., Skrøvseth, S.O., Simonsen, G.S., Godtliebsen, F., 2014. Power law approximations of movement network data for modeling infectious disease spread. Biometrical Journal 56, 363–382.

Gers, F.A., Eck, D., Schmidhuber, J., 2001. Applying LSTM to Time Series Predictable through Time-Window Approaches, in: Dorffner, G., Bischof, H., Hornik, K. (Eds.), Artificial Neural Networks — ICANN 2001, Lecture Notes in Computer Science. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 669–676. https://doi.org/10.1007/3-540-44668-0_93

Gers, F.A., Schmidhuber, J., 2001. LSTM recurrent networks learn simple context-free and context-sensitive languages. IEEE Trans. Neural Netw. 12, 1333–1340. https://doi.org/10.1109/72.963769

Gers, F.A., Schmidhuber, J., 2000. Recurrent nets that time and count, in: Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium. Presented at the Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium, IEEE, Como, Italy, pp. 189–194 vol.3. https://doi.org/10.1109/IJCNN.2000.861302

Gers, F.A., Schmidhuber, J., Cummins, F., 2000. Learning to Forget: Continual Prediction with LSTM. Neural Computation 12, 2451–2471. https://doi.org/10.1162/089976600300015015

Gneiting, T., Balabdaoui, F., Raftery, A.E., 2007. Probabilistic Forecasts, Calibration and Sharpness. Journal of the Royal Statistical Society Series B: Statistical Methodology 69, 243–268. https://doi.org/10.1111/j.1467-9868.2007.00587.x

Gneiting, T., Katzfuss, M., 2014. Probabilistic Forecasting. Annu. Rev. Stat. Appl. 1, 125–151. https://doi.org/10.1146/annurev-statistics-062713-085831

Gneiting, T., Raftery, A.E., 2007. Strictly Proper Scoring Rules, Prediction, and Estimation. Journal of the American Statistical Association 102, 359–378. https://doi.org/10.1198/016214506000001437

Gneiting, T., Wolffram, D., Resin, J., Kraus, K., Bracher, J., Dimitriadis, T., Hagenmeyer, V., Jordan, A.I., Lerch, S., Phipps, K., Schienle, M., 2023. Model Diagnostics and Forecast Evaluation for Quantiles. Annu. Rev. Stat. Appl. 10, 597–621. https://doi.org/10.1146/annurev-statistics-032921-020240

Good, I.J., 1952. Rational decisions. Journal of the Royal Statistical Society: Series B (Methodological) 14, 107–114.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep learning, Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts.

Graves, A., 2014. Generating Sequences With Recurrent Neural Networks. http://arxiv.org/abs/1308.0850

Graves, A., Mohamed, A., Hinton, G., 2013. Speech recognition with deep recurrent neural networks, in: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. Ieee, pp. 6645–6649.

Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R., Schmidhuber, J., 2017. LSTM: A Search Space Odyssey. IEEE Trans. Neural Netw. Learning Syst. 28, 2222–2232. https://doi.org/10.1109/TNNLS.2016.2582924

Held, L., Höhle, M., Hofmann, M., 2005. A statistical framework for the analysis of multivariate infectious disease surveillance counts. Statistical Modelling 5, 187–199. https://doi.org/10.1191/1471082X05st098oa

Held, L., Paul, M., 2012. Modeling seasonality in space-time infectious disease surveillance data: Modeling seasonality in space-time data. Biom. J. 54, 824–843. https://doi.org/10.1002/bimj.201200037

Hewamalage, H., Bergmeir, C., Bandara, K., 2021a. Global Models for Time Series Forecasting: A Simulation Study. http://arxiv.org/abs/2012.12485

Hewamalage, H., Bergmeir, C., Bandara, K., 2021b. Recurrent Neural Networks for Time Series Forecasting: Current Status and Future Directions. International Journal of Forecasting 37, 388–427. https://doi.org/10.1016/j.ijforecast.2020.06.008

Hihi, S., Bengio, Y., 1995. Hierarchical Recurrent Neural Networks for Long-Term Dependencies, in: Touretzky, D., Mozer, M.C., Hasselmo, M. (Eds.), Advances in Neural Information Processing Systems. MIT Press.

Hinton, G.E., Osindero, S., Teh, Y.-W., 2006. A fast learning algorithm for deep belief nets. Neural computation 18, 1527–1554.

Ho, S.L., Xie, M., Goh, T.N., 2002. A comparative study of neural network and Box-Jenkins ARIMA modeling in time series prediction. Computers & Industrial Engineering 42, 371–375. https://doi.org/10.1016/S0360-8352(02)00036-0

Hochreiter, S., Schmidhuber, J., 1997. Long Short-Term Memory. Neural Computation 9, 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hunter, J.D., 2007. Matplotlib: A 2D Graphics Environment. Comput. Sci. Eng. 9, 90–95. https://doi.org/10.1109/MCSE.2007.55

Hyndman, R.J., 2020. A brief history of forecasting competitions. International Journal of Forecasting 36, 7–14. https://doi.org/10.1016/j.ijforecast.2019.03.015

Jansen, A.G.S.C., Sanders, E.A.M., Van Der Ende, A., Van Loon, A.M., Hoes, A.W., Hak, E., 2008. Invasive pneumococcal and meningococcal disease: association with influenza virus and respiratory syncytial virus activity? Epidemiol. Infect. 136, 1448–1454. https://doi.org/10.1017/S0950268807000271

Januschowski, T., Gasthaus, J., Wang, Y., Salinas, D., Flunkert, V., Bohlke-Schneider, M., Callot, L., 2022. Criteria for Classifying Forecasting Methods. http://arxiv.org/abs/2212.03523

Jensen, E.S., Lundbye-Christensen, S., Samuelsson, S., Sørensen, H.T., Carl Schønheyder, H., 2003. A 20-year ecological study of the temporal association between influenza and meningococcal disease. Eur J Epidemiol 19, 181–187. https://doi.org/10.1023/B:EJEP.0000017659.80903.5f

Jiang, F., Han, X., Zhang, W., Chen, G., 2021. Atmospheric PM2.5 Prediction Using DeepAR Optimized by Sparrow Search Algorithm with Opposition-Based and Fitness-Based Learning. Atmosphere 12, 894. https://doi.org/10.3390/atmos12070894

Johnson, N.P.A.S., Mueller, J., 2002. Updating the Accounts: Global Mortality of the 1918-1920 "Spanish" Influenza Pandemic. Bulletin of the History of Medicine 76, 105–115. https://doi.org/10.1353/bhm.2002.0022

Jordan, M.I., 1986. Serial order: a parallel distributed processing approach. technical report, june 1985-march 1986. California Univ., San Diego, La Jolla (USA). Inst. for Cognitive Science.

Jozefowicz, R., Zaremba, W., Sutskever, I., 2015. An empirical exploration of recurrent network architectures, in: International Conference on Machine Learning. PMLR, pp. 2342–2350.

Jung, S., Moon, J., Park, S., Hwang, E., 2022. Self-Attention-Based Deep Learning Network for Regional Influenza Forecasting. IEEE J. Biomed. Health Inform. 26, 922–933. https://doi.org/10.1109/JBHI.2021.3093897

Kalchbrenner, N., Blunsom, P., 2013. Recurrent continuous translation models, in: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing. pp. 1700–1709.

Khaldi, R., Afia, A.E., Chiheb, R., Tabik, S., 2022. What is the best RNN-cell structure for forecasting each time series behavior? http://arxiv.org/abs/2203.07844

Kriegeskorte, N., Golan, T., 2019. Neural network models and deep learning. Current Biology 29, R231–R236. https://doi.org/10.1016/j.cub.2019.02.034

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2017. ImageNet classification with deep convolutional neural networks. Commun. ACM 60, 84–90. https://doi.org/10.1145/3065386

Lang, K.J., Waibel, A.H., Hinton, G.E., 1990. A time-delay neural network architecture for isolated word recognition. Neural Networks 3, 23–43. https://doi.org/10.1016/0893-6080(90)90044-L

Lara-Benítez, P., Carranza-García, M., Riquelme, J.C., 2021. An Experimental Review on Deep Learning Architectures for Time Series Forecasting. Int. J. Neur. Syst. 31, 2130001. https://doi.org/10.1142/S0129065721300011

LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. nature 521, 436–444.

Li, C., Zheng, J., Okamura, H., Dohi, T., 2022. Software Reliability Prediction through Encoder-Decoder Recurrent Neural Networks. Int J Math, Eng, Manag Sci 7, 325–340. https://doi.org/10.33889/IJMEMS.2022.7.3.022

Li, T., Xiao, F., Li, J., Zhang, J., 2023. Deepar-Based Ground Subsidence Prediction Method, in: Zeng, Z., Gaikar, V., Lotfi, R. (Eds.), Proceedings of the 2022 3rd International Conference on E-Commerce and Internet Technology (ECIT 2022). Atlantis Press International BV, Dordrecht, pp. 760–771. https://doi.org/10.2991/978-94-6463-005-3_77

Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J.E., Stoica, I., 2018. Tune: A Research Platform for Distributed Model Selection and Training. http://arxiv.org/abs/1807.05118

Longini, I.M., Nizam, A., Xu, S., Ungchusak, K., Hanshaoworakul, W., Cummings, D.A.T., Halloran, M.E., 2005. Containing Pandemic Influenza at the Source. Science 309, 1083–1087. https://doi.org/10.1126/science.1115717

Machete, R.L., 2013. Contrasting probabilistic scoring rules. Journal of Statistical Planning and Inference 143, 1781–1790.

Maind, S.B., Wankar, P., 2014. Research paper on basic of artificial neural network. International Journal on Recent and Innovation Trends in Computing and Communication 2, 96–100.

Makridakis, S., Hibon, M., 2000. The M3-Competition: results, conclusions and implications. International Journal of Forecasting 16, 451–476. https://doi.org/10.1016/S0169-2070(00)00057-1

Makridakis, S., Spiliotis, E., Assimakopoulos, V., 2022. M5 accuracy competition: Results, findings, and conclusions. International Journal of Forecasting 38, 1346–1364. https://doi.org/10.1016/j.ijforecast.2021.11.013

Makridakis, S., Spiliotis, E., Assimakopoulos, V., 2020. The M4 Competition: 100,000 time series and 61 forecasting methods. International Journal of Forecasting 36, 54–74. https://doi.org/10.1016/j.ijforecast.2019.04.014

Matheson, J.E., Winkler, R.L., 1976. Scoring Rules for Continuous Probability Distributions. Management Science 22, 1087–1096. https://doi.org/10.1287/mnsc.22.10.1087

McCulloch, W.S., Pitts, W., 1943. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics 5, 115–133.

Meyer, S., Held, L., 2016. Incorporating social contact data in spatio-temporal models for infectious disease spread. Biostat kxw051. https://doi.org/10.1093/biostatistics/kxw051

Meyer, S., Held, L., 2014. Power-law models for infectious disease spread. Ann. Appl. Stat. 8. https://doi.org/10.1214/14-AOAS743

Meyer, S., Held, L., Höhle, M., 2017. Spatio-Temporal Analysis of Epidemic Phenomena Using the R Package surveillance. J. Stat. Soft. 77. https://doi.org/10.18637/jss.v077.i11

Montero-Manso, P., Hyndman, R.J., 2021. Principles and algorithms for forecasting groups of time series: Locality and globality. International Journal of Forecasting 37, 1632–1653. https://doi.org/10.1016/j.ijforecast.2021.03.004

Mozer, M.C., 1991. Induction of Multiscale Temporal Structure, in: Moody, J., Hanson, S., Lippmann, R.P. (Eds.), Advances in Neural Information Processing Systems. Morgan-Kaufmann.

Oancea, B., Ciucu, Şt.C., 2014. Time series forecasting using neural networks. http://arxiv.org/abs/1401.1333

Osman Kurban, A., 2004. Analysis of shafts surface pressures using neural networks. Industrial Lubrication and Tribology 56, 217–225. https://doi.org/10.1108/00368790410541561

Pascanu, R., Gulcehre, C., Cho, K., Bengio, Y., 2014. How to Construct Deep Recurrent Neural Networks. http://arxiv.org/abs/1312.6026

Patterson, K.D., Pyle, G.F., 1991. The geography and mortality of the 1918 influenza pandemic. Bulletin of the History of Medicine 65, 4–21.

Paul, M., Held, L., 2011. Predictive assessment of a non-linear random effects model for multivariate time series of infectious disease counts. Statistics in medicine 30, 1118–1136.

Paul, M., Held, L., Toschke, A.M., 2008. Multivariate modelling of infectious disease surveillance data. Statist. Med. 27, 6250–6267. https://doi.org/10.1002/sim.3440

Paul, M.J., Dredze, M., Broniatowski, D., 2014. Twitter Improves Influenza Forecasting. PLoS Curr. https://doi.org/10.1371/currents.outbreaks.90b9ed0f59bae4ccaa683a39865d9117

Polgreen, P.M., Chen, Y., Pennock, D.M., Nelson, F.D., 2008. Using Internet Searches for Influenza Surveillance. CLIN INFECT DIS 47, 1443–1448. https://doi.org/10.1086/593098

Raeesi, M., Mesgari, M.S., Mahmoudi, P., 2014. Traffic time series forecasting by feedforward neural network: a case study based on traffic data of monroe. Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci. XL-2/W3, 219–223. https://doi.org/10.5194/isprsarchives-XL-2-W3-219-2014

R Core Team, 2023. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/

Ray, E.L., Wattanachit, N., Niemi, J., Kanji, A.H., House, K., Cramer, E.Y., Bracher, J., Zheng, A., Yamana, T.K., Xiong, X., Woody, S., Wang, Y., Wang, L., Walraven, R.L., Tomar, V., Sherratt, K., Sheldon, D., Reiner, R.C., Prakash, B.A., Osthus, D., Li, M.L., Lee, E.C., Koyluoglu, U., Keskinocak, P., Gu, Y., Gu, Q., George, G.E., España, G., Corsetti, S., Chhatwal, J., Cavany, S., Biegel, H., Ben-Nun, M., Walker, J., Slayton, R., Lopez, V., Biggerstaff, M., Johansson, M.A., Reich, N.G., 2020. Ensemble Forecasts of Coronavirus Disease 2019 (COVID-19) in the U.S. (preprint). Epidemiology. https://doi.org/10.1101/2020.08.19.20177493

Ring, M., 1992. Learning Sequential Tasks by Incrementally Adding Higher Orders, in: Hanson, S., Cowan, J., Giles, C. (Eds.), Advances in Neural Information Processing Systems. Morgan-Kaufmann.

Robert Koch-Institut (SurvStat@RKI), Weekly influenza data from 2001 to 2023 https://survstat.rki.de/, last accessed on 24 January 2023.

Sak, H., Senior, A., Beaufays, F., 2014. Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition. http://arxiv.org/abs/1402.1128

Salinas, D., Flunkert, V., Gasthaus, J., Januschowski, T., 2020. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. International Journal of Forecasting 36, 1181–1191. https://doi.org/10.1016/j.ijforecast.2019.07.001

Salmon, M., Schumacher, D., Höhle, M., 2016. Monitoring Count Time Series in *R* : Aberration Detection in Public Health Surveillance. J. Stat. Soft. 70. https://doi.org/10.18637/jss.v070.i10

Scalera, N.M., Mossad, S.B., 2009. The First Pandemic of the 21st Century: Review of the 2009 Pandemic Variant Influenza A (H1N1) Virus. Postgraduate Medicine 121, 43–47. https://doi.org/10.3810/pgm.2009.09.2051

Schmidhuber, J., 2015. Deep Learning in Neural Networks: An Overview. Neural Networks 61, 85–117. https://doi.org/10.1016/j.neunet.2014.09.003

Schmidhuber, J., 1992. Learning Complex, Extended Sequences Using the Principle of History Compression. Neural Computation 4, 234–242. https://doi.org/10.1162/neco.1992.4.2.234

Sen, R., Yu, H.-F., Dhillon, I., 2019. Think Globally, Act Locally: A Deep Neural Network Approach to High-Dimensional Time Series Forecasting. https://doi.org/10.48550/ARXIV.1905.03806

Singh, Y., Chauhan, A.S., 2009. Neural networks in data mining. Journal of Theoretical & Applied Information Technology 5.

Smyl, S., 2020. A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. International Journal of Forecasting 36, 75–85. https://doi.org/10.1016/j.ijforecast.2019.03.017

Soliman, M., Lyubchich, V., Gel, Y.R., 2019. Complementing the power of deep learning with statistical model fusion: Probabilistic forecasting of influenza in Dallas County, Texas, USA. Epidemics 28, 100345. https://doi.org/10.1016/j.epidem.2019.05.004

Song, X., Xiao, J., Deng, J., Kang, Q., Zhang, Y., Xu, J., 2016. Time series analysis of influenza incidence in Chinese provinces from 2004 to 2011. Medicine 95, e3929. https://doi.org/10.1097/MD.0000000000003929

Statistisches Bundesamt (Destatis), German population data from 2000 until 2021, https://www-genesis.destatis.de/genesis//online?operation=table&code=12411-0015, last accessed on 15 May 2023.

Sutskever, I., Martens, J., Hinton, G.E., 2011. Generating text with recurrent neural networks, in: Proceedings of the 28th International Conference on Machine Learning (ICML-11). pp. 1017–1024.

Sutskever, I., Vinyals, O., Le, Q.V., 2014. Sequence to Sequence Learning with Neural Networks. http://arxiv.org/abs/1409.3215

Tapak, L., Hamidi, O., Fathian, M., Karami, M., 2019. Comparative evaluation of time series models for predicting influenza outbreaks: application of influenza-like illness data from sentinel sites of healthcare centers in Iran. BMC Res Notes 12, 353. https://doi.org/10.1186/s13104-019-4393-y

Taubenberger, J.K., Morens, D.M., 2006. 1918 Influenza: the mother of all pandemics. Revista Biomedica 17, 69–79.

Taylor, K.S., Taylor, J.W., 2022. Interval forecasts of weekly incident and cumulative COVID-19 mortality in the United States: A comparison of combining methods. PLoS ONE 17, e0266096. https://doi.org/10.1371/journal.pone.0266096

TensorFlow Developers, 2023. TensorFlow. https://doi.org/10.5281/ZENODO.7987192

The pandas development team, 2022. Pandas-dev/pandas: Pandas. https://doi.org/10.5281/zenodo.7979740

Trilla, A., Trilla, G., Daer, C., 2008. The 1918 “Spanish Flu” in Spain. CLIN INFECT DIS 47, 668–673. https://doi.org/10.1086/590567

Turek, J.S., Jain, S., Vo, V., Capota, M., Huth, A.G., Willke, T.L., 2020. Approximating Stacked and Bidirectional Recurrent Architectures with the Delayed Recurrent Neural Network. http://arxiv.org/abs/1909.00021

Tyralis, H., Papacharalampous, G., 2022. A review of probabilistic forecasting and prediction with machine learning. http://arxiv.org/abs/2209.08307

Van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K., 2016. WaveNet: A Generative Model for Raw Audio. http://arxiv.org/abs/1609.03499

Wang, G., Zhang, Y., Ye, X., Mou, X., 2020a. Machine learning for tomographic imaging, IPEM-IOP series in physics and engineering in medicine and biology. IOP Publishing, Bristol, UK.

Wang, P., Zheng, X., Ai, G., Liu, D., Zhu, B., 2020b. Time series prediction for the epidemic trends of COVID-19 using the improved LSTM deep learning method: Case studies in Russia, Peru and Iran. Chaos, Solitons & Fractals 140, 110214. https://doi.org/10.1016/j.chaos.2020.110214

WHO, 2023. Influenza (Seasonal). World Health Organization, https://www.who.int/news-room/fact-sheets/detail/influenza-(seasonal), last accessed on 30 May 2023.

Winkler, R.L., Muñoz, J., Cervera, J.L., Bernardo, J.M., Blattenberger, G., Kadane, J.B., Lindley, D.V., Murphy, A.H., Oliver, R.M., Ríos-Insua, D., 1996. Scoring rules and the evaluation of probabilities. Test 5, 1–60. https://doi.org/10.1007/BF02562681

Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, Ł., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., Dean, J., 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. http://arxiv.org/abs/1609.08144

Wu, Y., Yang, Y., Nishiura, H., Saitoh, M., 2018. Deep Learning for Epidemiological Predictions, in: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval. Presented at the SIGIR '18: The 41st International ACM SIGIR conference on research and development in Information Retrieval, ACM, Ann Arbor MI USA, pp. 1085–1088. https://doi.org/10.1145/3209978.3210077

Yang, C.-T., Chen, Y.-A., Chan, Y.-W., Lee, C.-L., Tsan, Y.-T., Chan, W.-C., Liu, P.-Y., 2020. Influenza-like illness prediction using a long short-term memory deep learning model with multiple open data sources. J Supercomput 76, 9303–9329. https://doi.org/10.1007/s11227-020-03182-5

Yu, Y., Si, X., Hu, C., Zhang, J., 2019. A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. Neural Computation 31, 1235–1270. https://doi.org/10.1162/neco_a_01199

Zeroual, A., Harrou, F., Dairi, A., Sun, Y., 2020. Deep learning methods for forecasting COVID-19 time-Series data: A Comparative study. Chaos, Solitons & Fractals 140, 110121. https://doi.org/10.1016/j.chaos.2020.110121

Zhang, H., Li, S., Chen, Y., Dai, J., Yi, Y., 2022. A Novel Encoder-Decoder Model for Multivariate Time Series Forecasting. Computational Intelligence and Neuroscience 2022, 1–17. https://doi.org/10.1155/2022/5596676

Zhang, J., Zuo, X., Xu, M., Han, J., Zhang, B., 2021. Base Station Network Traffic Prediction Approach Based on LMA -DeepAR, in: 2021 IEEE 6th International Conference on Computer and Communication Systems (ICCCS). Presented at the 2021 IEEE 6th International Conference on Computer and Communication Systems (ICCCS), IEEE, Chengdu, China, pp. 473–479. https://doi.org/10.1109/ICCCS52626.2021.9449212

Zhou, G.-B., Wu, J., Zhang, C.-L., Zhou, Z.-H., 2016. Minimal gated unit for recurrent neural networks. Int. J. Autom. Comput. 13, 226–234. https://doi.org/10.1007/s11633-016-1006-2

A Appendix

A.1 Figures

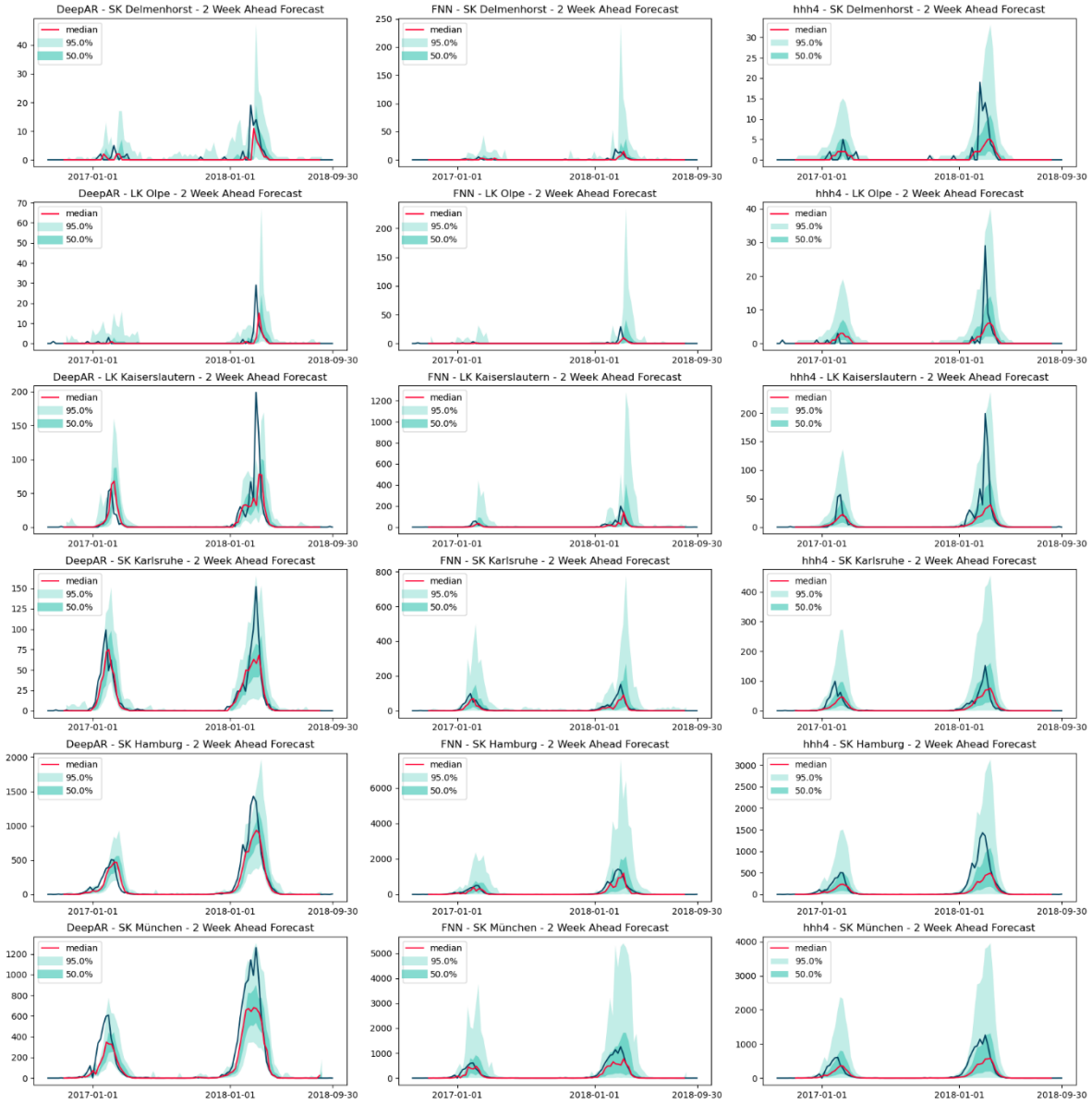

Figure 23: Two-week ahead forecasts of the default DeepAR, default FNN and hhh4 model for the test period.

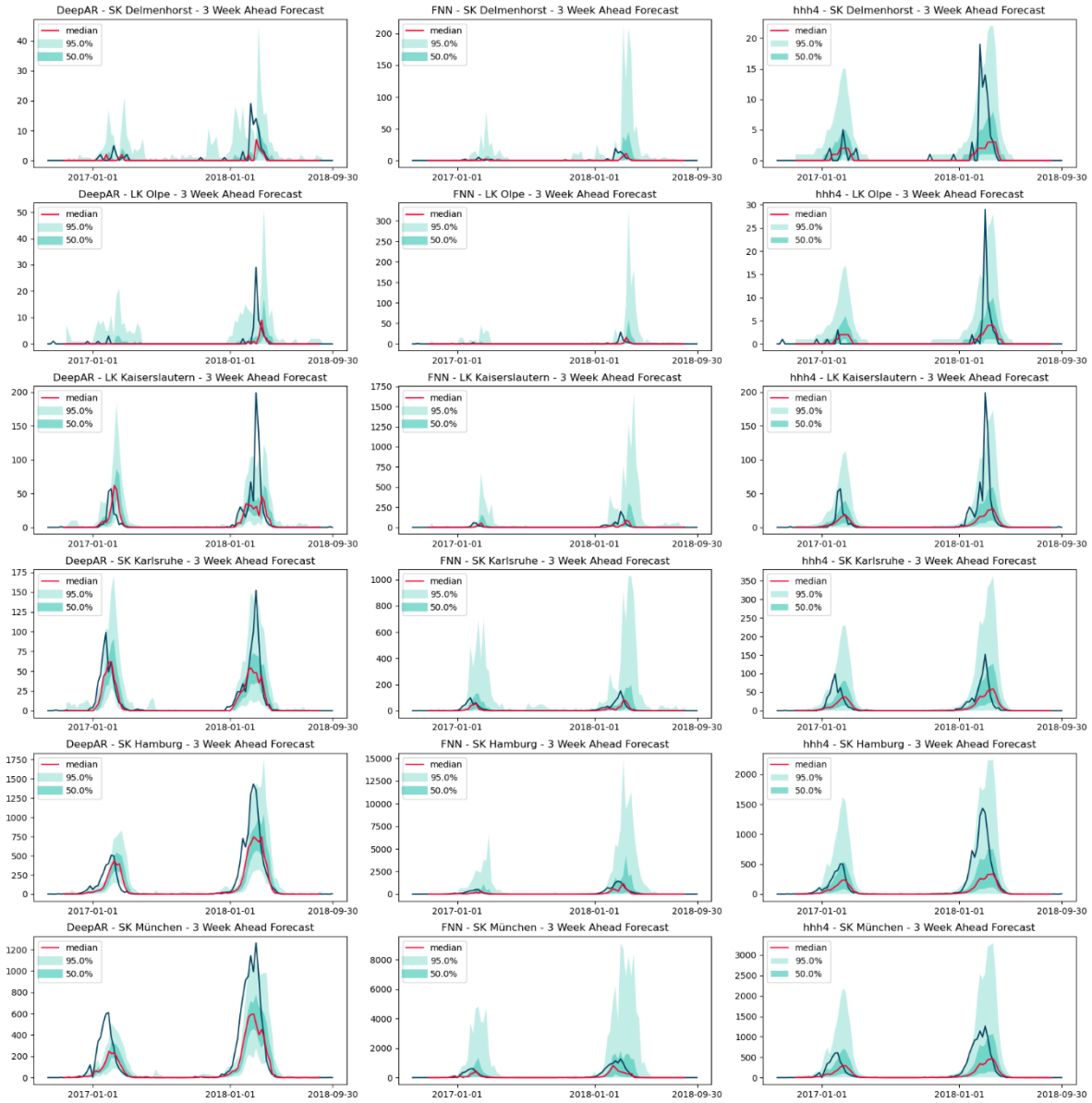

Figure 24: Three-week ahead forecasts of the default DeepAR, default FNN and hhh4 model for the test period.

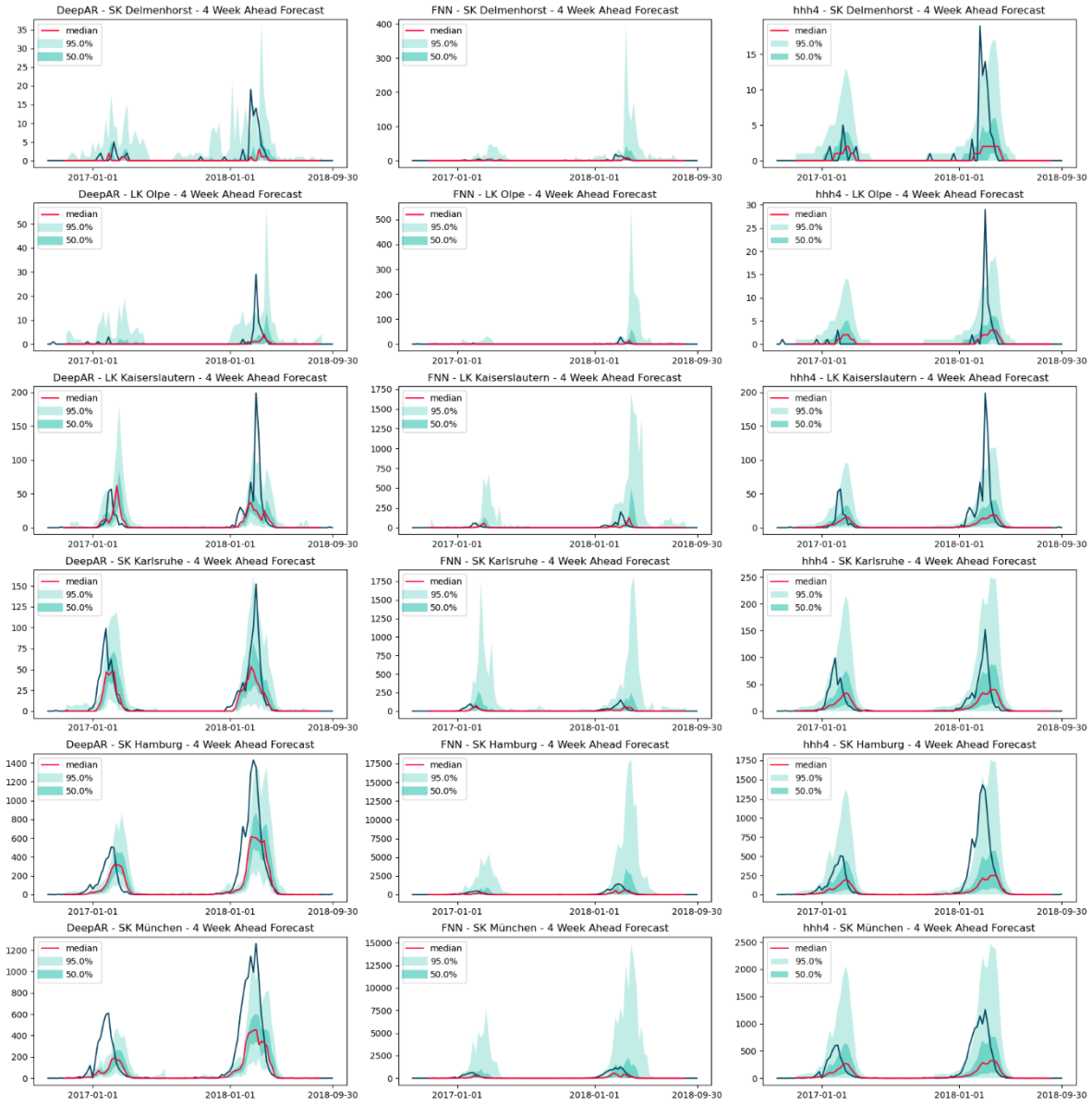

Figure 25: Four-week ahead forecasts of the default DeepAR, default FNN and hhh4 model for the test period.

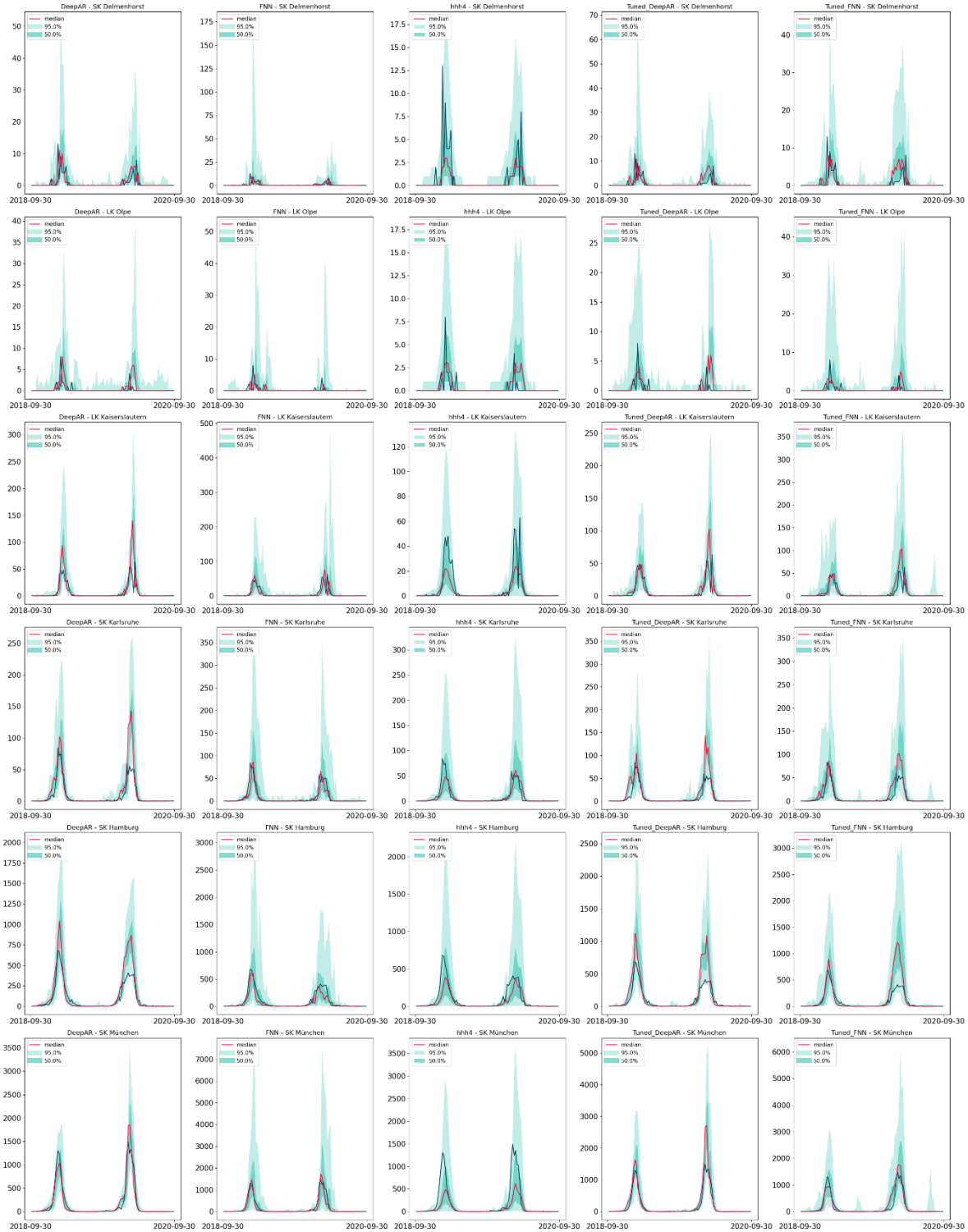

Figure 26: Two-week ahead forecasts of the default DeepAR, default FNN, hhh4, tuned DeepAR and tuned FNN model for the test period.

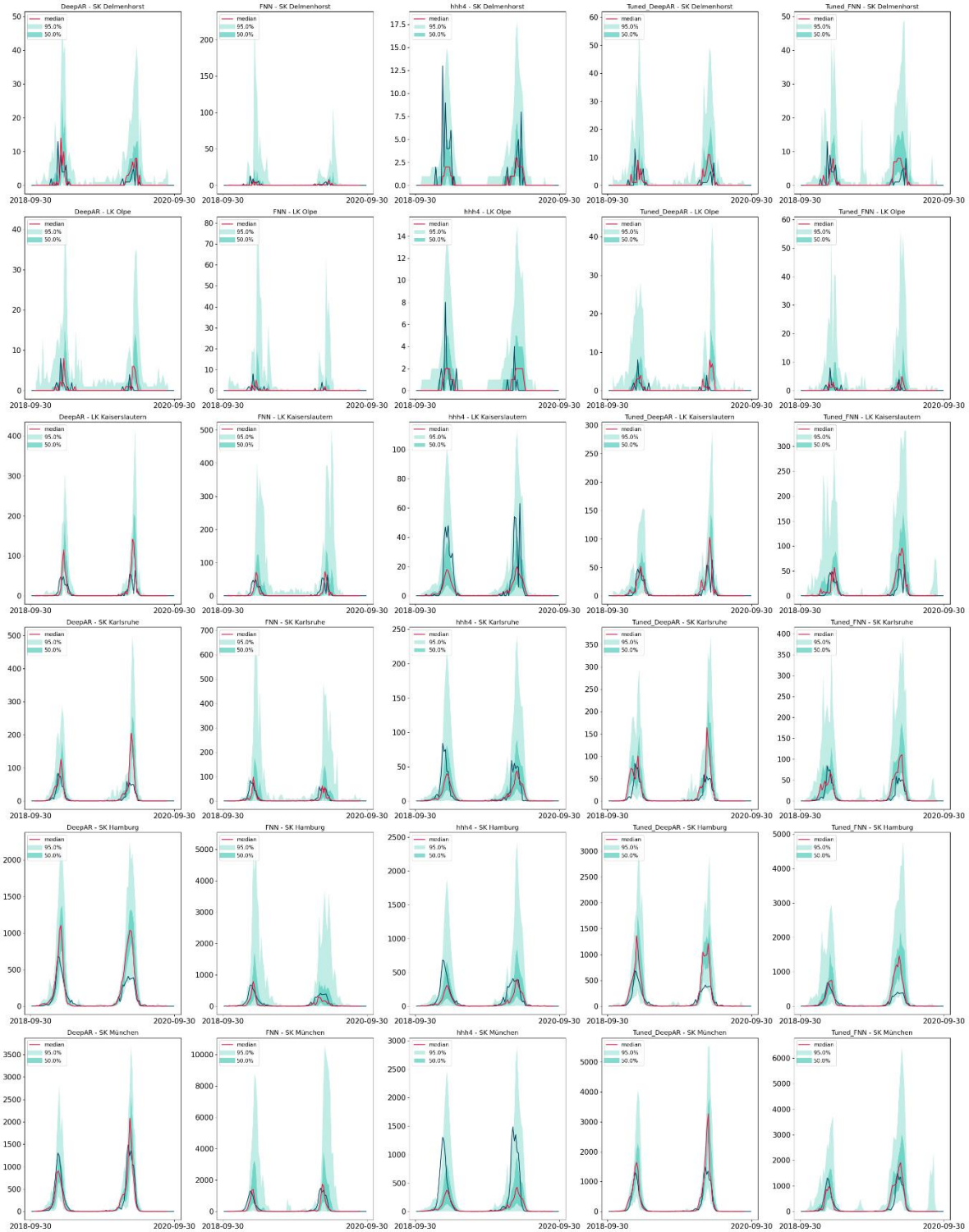

Figure 27: Three-week ahead forecasts of the default DeepAR, default FNN, hhh4, tuned DeepAR and tuned FNN model for the test period.

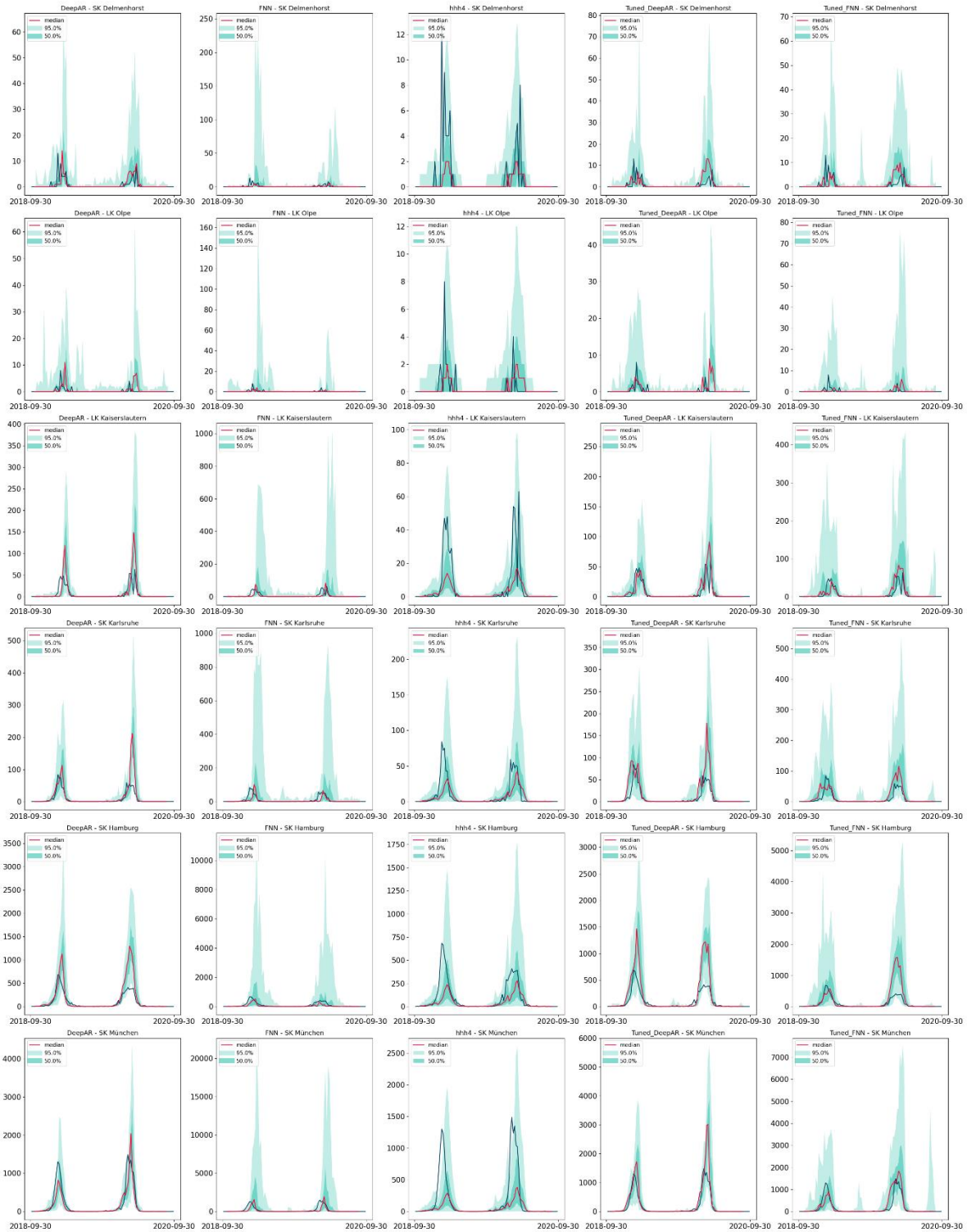

Figure 28: Four-week ahead forecasts of the default DeepAR, default FNN, hhh4, tuned DeepAR and tuned FNN model for the test period.

A.2 Tables

Quantile		Coverage			
		1 Week Ahead	2 Week Ahead	3 Week Ahead	4 Week Ahead
0.025	**DeepAR**	0.6449	0.6440	0.6425	0.6409
	FNN	0.6361	0.6360	**0.6359**	**0.6359**
	hhh4	**0.6360**	**0.6359**	**0.6359**	**0.6359**
0.1	**DeepAR**	0.6671	0.6632	0.6591	0.6552
	FNN	**0.6381**	**0.6366**	**0.6363**	**0.6362**
	hhh4	0.6383	0.6380	0.6392	0.6402
0.25	**DeepAR**	0.7072	0.6953	0.6869	0.6799
	FNN	**0.6532**	**0.6463**	**0.6436**	**0.6444**
	hhh4	0.6611	0.6628	0.6674	0.6692
0.5	**DeepAR**	0.7723	0.7485	0.7310	0.7160
	FNN	**0.7322**	**0.7191**	**0.7033**	**0.7033**
	hhh4	0.7491	0.7448	0.7447	0.7438
0.75	**DeepAR**	0.8491	**0.8160**	**0.7880**	**0.7679**
	FNN	**0.8377**	0.8308	0.8015	0.7850
	hhh4	0.8721	0.8550	0.8369	0.8269
0.9	**DeepAR**	**0.9155**	**0.8835**	**0.8538**	**0.8290**
	FNN	0.9189	0.9220	0.9000	0.8820
	hhh4	0.9516	0.9327	0.9071	0.8888
0.975	**DeepAR**	**0.9660**	**0.9491**	**0.9312**	**0.9105**
	FNN	0.9688	0.9741	0.9661	0.9585
	hhh4	0.9868	0.9793	0.9615	0.9435

Table 5: Upper coverages of the default and the baseline models on the test set rounded to 4 decimal places. We highlighted the coverages that are closest to the associated quantile.

Quantile		Coverage			
		1 Week Ahead	2 Week Ahead	3 Week Ahead	4 Week Ahead
0.025	**DeepAR**	**0.0068**	**0.0066**	**0.0054**	**0.0040**
	FNN	0.0001	0.0000	0.0000	0.0000
	hhh4	0.0001	0.0000	0.0000	0.0000
0.1	**DeepAR**	**0.0252**	**0.0227**	**0.0197**	**0.0164**

	FNN	0.0017	0.0005	0.0003	0.0001
	hhh4	0.0022	0.0018	0.0021	0.0025
0.25	**DeepAR**	**0.0636**	**0.0545**	**0.0481**	**0.0438**
	FNN	0.0127	0.0086	0.0067	0.0076
	hhh4	0.0216	0.0240	0.0310	0.0362
0.5	**DeepAR**	**0.1490**	0.1225	0.1042	0.0935
	FNN	0.0984	0.0871	0.0709	0.0745
	hhh4	0.1218	**0.1271**	**0.1397**	**0.1571**
0.75	**DeepAR**	0.2895	0.2434	0.2096	0.1861
	FNN	0.2653	0.2837	0.2501	0.2450
	hhh4	**0.3112**	**0.3175**	**0.3367**	**0.3610**
0.9	**DeepAR**	0.4869	0.4331	0.3810	0.3416
	FNN	**0.5272**	**0.6313**	**0.6244**	**0.6420**
	hhh4	0.5215	0.5405	0.5448	0.5493
0.975	**DeepAR**	**0.8109**	0.7996	0.7800	0.7515
	FNN	0.7693	**0.8408**	**0.8530**	**0.8724**
	hhh4	0.7637	0.8094	0.8198	0.8206

Table 6: Lower coverages of the default and the baseline models on the test set rounded to 4 decimal places. We highlighted the coverages that are closest to the associated quantile.

		Coverage			
Quantile		**1 Week Ahead**	**2 Week Ahead**	**3 Week Ahead**	**4 Week Ahead**
0.025	**default DeepAR**	0.6278	0.6297	0.6290	0.6250
	default FNN	0.6166	0.6152	0.6147	0.6147
	hhh4	**0.6147**	**0.6146**	**0.6146**	**0.6146**
	tuned DeepAR	0.6223	0.6281	0.6272	0.6245
	tuned FNN	0.6194	0.6193	0.6213	0.6186
0.1	**default DeepAR**	0.6598	0.6637	0.6625	0.6553
	default FNN	0.6258	0.6190	**0.6166**	**0.6151**
	hhh4	**0.6176**	**0.6177**	0.6182	0.6177
	tuned DeepAR	0.6507	0.6632	0.6614	0.6581
	tuned FNN	0.6408	0.6419	0.6488	0.6423
0.25	**default DeepAR**	0.7186	0.7221	0.7184	0.7082
	default FNN	0.6572	**0.6374**	**0.6281**	**0.6221**
	hhh4	**0.6396**	0.6399	0.6411	0.6420
	tuned DeepAR	0.7147	0.7277	0.7259	0.7248

	tuned FNN	0.7033	0.7004	0.7135	0.7062
0.5	**default DeepAR**	0.8143	0.8115	0.8020	0.7864
	default FNN	0.7430	0.7209	**0.6993**	**0.6851**
	hhh4	**0.7241**	**0.7173**	0.7162	0.7184
	tuned DeepAR	0.8241	0.8305	0.8262	0.8260
	tuned FNN	0.8228	0.8218	0.8324	0.8334
0.75	**default DeepAR**	0.9004	0.8920	0.8801	0.8640
	default FNN	**0.8518**	0.8395	0.8228	**0.8030**
	hhh4	0.8604	**0.8379**	**0.8213**	0.8134
	tuned DeepAR	0.9136	0.9134	0.9098	0.9034
	tuned FNN	0.9286	0.9290	0.9347	0.9394
0.9	**default DeepAR**	0.9544	0.9502	0.9396	0.9250
	default FNN	**0.9264**	**0.9191**	0.9093	0.8966
	hhh4	0.9454	0.9259	**0.9025**	**0.8868**
	tuned DeepAR	0.9588	0.9592	0.9552	0.9518
	tuned FNN	0.9788	0.9805	0.9824	0.9841
0.975	**default DeepAR**	0.9874	0.9871	**0.9823**	**0.9744**
	default FNN	**0.9688**	0.9686	0.9663	0.9615
	hhh4	0.9835	**0.9757**	0.9587	0.9387
	tuned DeepAR	0.9879	0.9875	0.9866	0.9846
	tuned FNN	0.9952	0.9957	0.9958	0.9961

Table 7: Upper coverage values of the default and tuned models as well as the baseline model on the validation set rounded to 4 decimal places. We highlighted the coverages that are closest to the associated quantile.

		Coverage			
Quantile		**1 Week Ahead**	**2 Week Ahead**	**3 Week Ahead**	**4 Week Ahead**
0.025	**default DeepAR**	**0.0119**	**0.0148**	**0.0148**	0.0108
	default FNN	0.0022	0.0006	0.0003	0.0001
	hhh4	0.0005	0.0002	0.0001	0.0000
	tuned DeepAR	0.0072	0.0132	0.0125	**0.0110**
	tuned FNN	0.0048	0.0046	0.0062	0.0041
0.1	**default DeepAR**	**0.0407**	**0.0467**	**0.0470**	0.0412
	default FNN	0.0105	0.0052	0.0025	0.0008
	hhh4	0.0039	0.0046	0.0054	0.0057
	tuned DeepAR	0.0315	0.0462	0.0463	**0.0442**
	tuned FNN	0.0248	0.0261	0.0345	0.0280

0.25	**default DeepAR**	**0.1004**	0.1050	0.1047	0.0967
	default FNN	0.0418	0.0240	0.0173	0.0105
	hhh4	0.0255	0.0287	0.0333	0.0408
	tuned DeepAR	0.0951	**0.1135**	**0.1143**	**0.1142**
	tuned FNN	0.0894	0.0888	0.1044	0.0983
0.5	**default DeepAR**	0.2129	0.2112	0.2039	0.1920
	default FNN	0.1473	0.1250	0.1031	0.0917
	hhh4	0.1314	0.1339	0.1523	0.1788
	tuned DeepAR	0.2203	0.2344	0.2321	0.2376
	tuned FNN	**0.2347**	**0.2364**	**0.2532**	**0.2573**
0.75	**default DeepAR**	0.3584	0.3455	0.3294	0.3144
	default FNN	0.3214	0.3314	0.3210	0.3173
	hhh4	0.3589	0.3625	0.3722	0.3880
	tuned DeepAR	0.3690	0.3788	0.3788	0.3761
	tuned FNN	**0.4658**	**0.4700**	**0.4821**	**0.5018**
0.9	**default DeepAR**	0.5343	0.5268	0.5097	0.4903
	default FNN	0.5859	0.6426	0.6833	0.7055
	hhh4	0.5869	0.6059	0.5958	0.5913
	tuned DeepAR	0.5152	0.5259	0.5264	0.5236
	tuned FNN	**0.7700**	**0.7671**	**0.7586**	**0.7739**
0.975	**default DeepAR**	0.8593	0.8790	0.8868	0.8825
	default FNN	0.8089	0.8526	0.8808	0.9007
	hhh4	0.8131	0.8581	0.8585	0.8517
	tuned DeepAR	0.8190	0.8330	0.8337	0.8315
	tuned FNN	**0.9574**	**0.9580**	**0.9471**	**0.9521**

Table 8: Lower coverage values of the default and tuned models as well as the baseline model on the validation set rounded to 4 decimal places. We highlighted the coverages that are closest to the associated quantile.