Discussion paper

Deep learning for time series forecasting and nowcasting

Pim Ouwehand,
Sabine Krieg and
Rob Willems

August 2024

Content

Summary

This study is an exploration of where we can expect added value for forecasting and nowcasting time series in official statistics by using deep learning techniques, as an alternative to classic time series models. We will compare several neural network algorithms, identify the key differences with classic time series methods and determine in what situations we can expect to benefit from these algorithms. In an empirical study, the methods are applied to several time series. We find that neural network algorithms can yield similar forecast and nowcast accuracy as classic methods for univariate time series, but it requires some effort to achieve this. When applied to a more challenging problem with several auxiliary variables and a more volatile series, in our case the Long Short-Term Memory model gave accurate results. This leads to the conclusion that deep learning can offer added value compared to classical methods for specific problems, but more research is needed.

Keywords

nowcasting, neural networks, deep learning, time series

1. Introduction

Time series analysis is an important part of the statistical process in official statistics. There are different purposes of such an analysis, for example:

- Seasonal adjustment. The aim is to separate the seasonal pattern from the trend-cycle and the noise. By adjusting for seasonal effects, short-term developments are easier to interpret. At Statistics Netherlands, the package JDemetra+ is used (Eurostat, 2024).

- Improvement of accuracy. The accuracy of current figures is improved by using estimates from the past about the same variable. At Statistics Netherlands, this approach is applied for figures about the labour market, and for consumer confidence (van den Brakel and Krieg, 2009, van den Brakel e.a., 2021). The method is based on structural times series models (Durbin and Koopman, 2012). This can be seen as a kind of small area estimation (Rao and Molina, 2015).

- Nowcasting. This can be applied when there is no data available for the reference period yet. In that case, historical information from the target series is combined with historical and current information from auxiliary series, in order to make an estimate for the reference period. In Zult e.a. (2023) a method is developed which combines information from a monthly series and a quarterly series about tax-turnover of selective subpopulations to a monthly series about the entire population. Part of the method is nowcasting of the quarterly series, as the last value of this series is not available on time for the publication.

- Analysis of published figures. One example is the business cycle dashboard of Statistics Netherlands (2024). There, the business cycle of different economic indicators are visualized together. For this, time series methods are needed in order to estimate the cyclical component of each series.

For part of the applications, it is necessary to estimate the time series components such as the trend and the seasonal. For other applications only the prediction of the next values is needed.

With the increased interest in machine learning (ML) approaches in official statistics, the question arises whether these techniques have added value compared to classic time series methods used in the applications above. ML is often used for forecasting purposes. In order to obtain good predictions, many algorithms use supervised learning. In this subclass of ML, the objective is to find a mapping function from the input data to the output variable of interest. The algorithm then learns the mapping function by feeding it with a large number of examples. Supervised learning can be further grouped into classification and regression problems. Some applications are aimed at classification problems, where the objective is to predict the correct (discrete) class label of an object.

Time series applications are an example of a regression problem. In time series applications, we are interested in a continuous quantity, for which also many supervised ML algorithms exist. These methods might especially be beneficial in forecasting and nowcasting, where the objective is to produce out-of-sample estimates of a time series at an early stage. Forecasting and nowcasting is an important tool in producing timely official statistics (EU, 2017). For nowcasting, commonly time series models are used that model the historical data of the target variable and combine this with early available data sources related to the indicator to be published. In some cases, there is a large dataset of auxiliary information available, and data reduction or variable selection has to be applied to properly model the problem. In this paper, we take some first steps in order to establish whether ML techniques can serve as a suitable alternative to deal with time series forecasting and nowcasting problems. While the main focus of the paper is on predicting new values, we will also discuss ML methods that compute the decomposition of a series into trend, seasonal and other components, albeit more briefly.

In recent years, there has been some evidence that ML methodologies can produce accurate time series forecasts. In some cases, they can outperform classic methods, either as a hybrid approach that combines ML with statistical methods (Makridakis et al., 2020) or as a pure ML approach (Makridakis et al., 2022). Also in official statistics, where focus is more on nowcasting than complete extrapolation (EU, 2017), there are examples (Hopp, 2022) where a ML method can give more accurate predictions than a classic method.

In Section 2, ML methods for time series analysis are described. In the literature, many ML methods are available. In this paper, we test several of them. In Section 3 the setup of the empirical study is described, and results follow in Section 4. The conclusions can be found in Section 5.

The authors thank Tim de Jong and Marco Puts from Statistics Netherlands for useful discussions and sharing their experience with ML and Tim de Jong for reviewing this paper.

2. Methods

In this section, we give an overview of ML methods that are suitable for time series analysis. We found that especially deep learning (DL) methods, a subclass of ML, are appropriate for our objectives. We first discuss the general principles behind DL and how a time series can be modeled in a neural network. After that, we give an overview of DL methods that are available. The overview is not complete as on the internet, many methods and applications of ML for time series analysis can be found. We have attempted to cover the most important methods.

Some of the methods described in this section are applied to our own series, the results are discussed in Section 4. Other methods are only mentioned here, as they look interesting for further research, but are not applied to a dataset.

2.1 Traditional time series analysis

In traditional time series analysis, models such as ARIMA (Box and Jenkins, 1976) or structural time series models (Durbin and Koopman, 2012) are used. These models assume that a time series consists of several components, the most important ones being the trend, the seasonal pattern and the noise or irregular component. For some applications, it is necessary to decompose the series into these components (such as seasonal adjustment), while in other applications, it is useful to make assumptions about the components in order to predict the next values. As we will see below, most of the here considered DL approaches do not attempt to describe the time series in terms of its components explicitly, but rather try to estimate a mapping from the input values (the historical data) to the output (the current or future values) by setting weights appropriately and thereby learning the patterns present in the series.

2.2 Neural networks

Artificial neural networks (ANN) are inspired by the human brain. The output of an ANN can be considered a function of the input: $y = f(x)$. The neural network connects the input values to the output via a collection of units (or nodes) which are connected by edges. Typically, the units are organized in layers. The first layer is the input layer where the input x enters the network. x can be a single number or a vector of numbers. In traditional statistics, this is often called auxiliary or explanatory information, while in data science, the term is features. Each edge has a weight, which determines how the value of the next unit is to be computed. This next unit combines weighted values of previous nodes, where the combination function is possibly non-linear. In this way, the input is processed through the network until the output layer is reached, where y is found. y can be a single number or a vector of numbers.

There is a vast array of possible ANN algorithms that can be used, and in this paper, we will consider only a small subset of the available options. We will start by describing the basic feed-forward ANN, and then move to RNN and LSTM networks, and Neural Prophet.

A feed-forward ANN consists of an input layer, one or more hidden layers, and an output layer. The hidden layer(s) consist of one or more nodes (called 'neurons') that process the data from the previous layer. The data flow is thus unidirectional, from input layer to output layer. All layers are fully connected, i.e. all neurons in one layer are connected to all neurons in the next layer. A relatively simple ANN is shown in Figure 2.1. In order to compute the value of a node (for example the first node of the second layer), first a linear function of the values of the nodes of the previous layer is computed. Then an activation function g(z) is applied to this linear combination. This is described in more detail below.

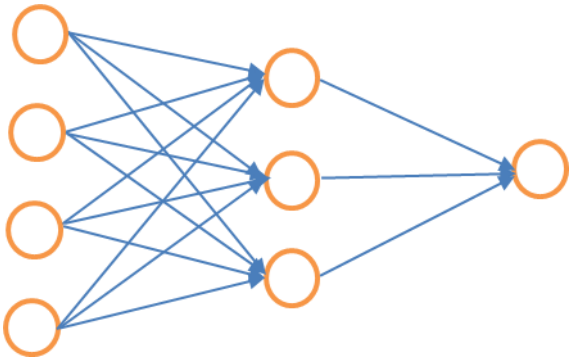


Figure 2.1: feed forward neural network

The user of an ANN has to choose the structure of the ANN. This can be from scratch, but it is also possible to use a structure which is proven to be useful from the literature. The weights of the edges are computed in a training process. This training starts with randomly chosen weights. The model is shown a collection of input values (the so called training set) and computes the output values, based on the weights. Then these computed output is compared with the true values, which are known for the training set. Depending on the difference between output of the model and true value the weights are adopted through a procedure called backpropagation. This is repeated until the model converges based on some criterion. Since the true values from the training set are used in the training process, this process is called supervised learning.

Modeling time series in an ANN

Feed-forward ANN's are not specifically designed for handling time series. Time dependency of the input data is not supported by default. The network connects the input layer with the output layer, and the supervised learning algorithm learns the mapping between these two layers. Any time dependency in a feed-forward network thus has to be addressed in the input layer and the choice of hidden layers.

First, we focus on preparing the input data. Let y_t be our target variable at time t, with t=1,…,T, and X_t the auxiliary variables at time t. The matrix X_t consists of m auxiliary variables, all given for t=1,…,T as well. A nowcasting time series model uses both the historical time series of the target variable and the auxiliary variables to make a prediction, where the latter can be added to the model as regressors. In order to translate this to a supervised learning problem, we use the fact that any time series model is basically a function of historical observations and auxiliary data:

$$y_t = f(y_{t-1}, y_{t-2}, \ldots, y_{t-n}, x_{1,t}, \ldots, x_{m,t}) + \varepsilon_t \quad \text{for t=1,..,T}$$

In a classic time series model, we make assumptions about function f, such as that it describes a trending or seasonal pattern in the series, and the goal is to find an appropriate form of f such that it has good explanatory and predictive value. In neural networks, we do not make any explicit assumptions, and rather let the learning algorithm discover for itself which patterns are present in the data. The algorithm only trains the network in order to give good predictions for the output variable, and explanatory value of the model is not explicitly considered. We therefore use as inputs the variables in the function above: the first n lags of y_t : $y_{t-1}, y_{t-2}, \ldots, y_{t-n}$, and the auxiliary variables for time t: X_t. Figure 2.2 gives an example of these inputs in a feed-forward neural network with a single hidden layer.

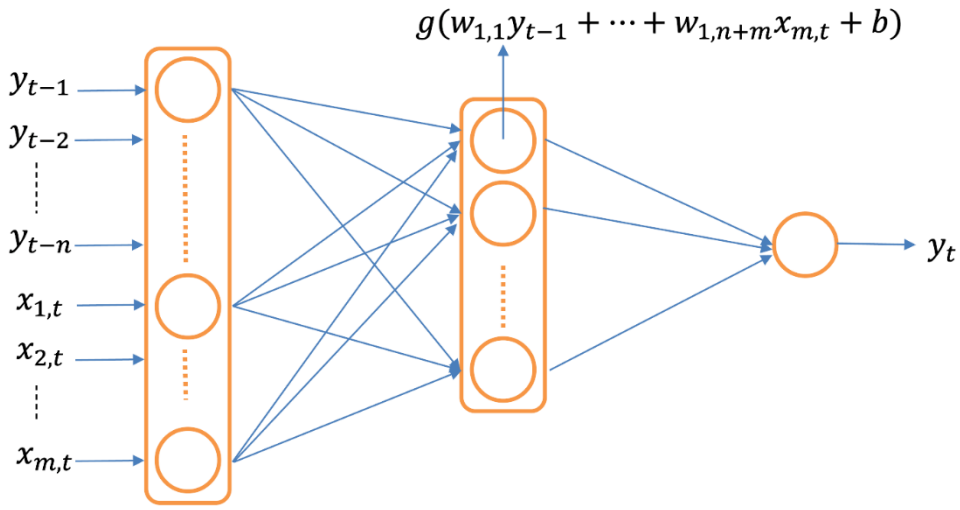


Figure 2.2: time series nowcasting model in a feed-forward neural network

After preparing the data and setting up the input layer, there are a couple of aspects to further determine how the learning algorithm finds a good mapping from input to output layer. The most important choice is the number of hidden layers and the number of neurons in each layer. Although guidelines exist, this is largely a matter of trial and error and depends on the application at hand. Although not all neurons need to be connected with all neurons in the next layer, we will assume this is the case. This is called a fully connected network (also dense

neural network or DNN). Below, when we discuss the training of the network, we will discuss how to gradually find an appropriate network design as we repeatedly let the algorithm learn a mapping function. The way the algorithm converges gives information about how to proceed.

Activation functions

In order to learn a mapping from inputs to output, some inputs should be given lower weights than other inputs. This is done via the neurons in each layer. Each neuron receives a number of inputs from the previous layer, suppose these are $x = x_1, \dots, x_k$. The neuron then first computes a weighted sum of these inputs:

$$z = wx + b = w_1 x_1, \dots, w_k x_k + b$$

After computing this linear combination, an activation function g() is applied. The activation function determines the output of the neuron and thus how the network learns the training data. Many activation functions exist, and the most appropriate one depends on the application at hand. The most important ones are the linear activation function (which is just the weighted sum above), tanh (which results in an output between -1 and 1), the ReLU (rectified linear unit) and the sigmoid function. The latter two are applied in this paper, with their corresponding functions given by:

$$\text{ReLU: } g(z) = \max(0, z)$$
$$\text{Sigmoid: } g(z) = \frac{1}{1 + e^{-z}}$$

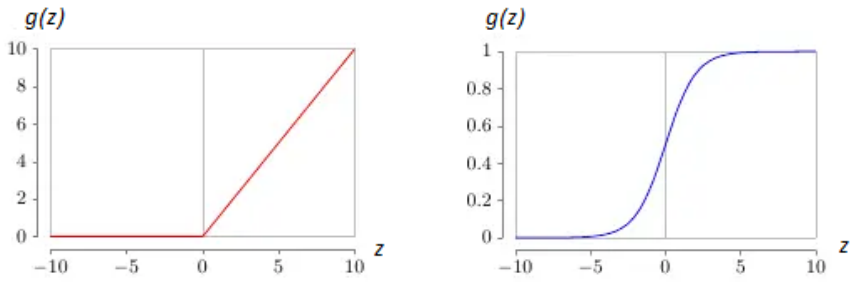


Figure 2.3: ReLU and sigmoid activation functions

The figures show how the weighted sum z is transformed by the activation function. The sigmoid, or logistic, function, for example, always returns a number between 0 and 1, and therefore is often applied for classification problems. It can also be applied to continuous variables, as long as data is scaled to lie in the interval [0,1]. The ReLU always returns a nonnegative number, and therefore can be useful for many time series applications. It is also used in order to obtain the results in this paper. These activation functions are in principle only used for the intermediate layers. For the output layer, the activation function is often chosen to be the identity function so that the operation at the output stage remains linear.

Training an ANN

To describe the organization of the training process we need some terms: epoch, batch size, learning rate, loss function, optimizer. The training set is processed in the model in relatively small portions which are called batches. After a batch is processed, the weights are adjusted. Stochastic gradient descent is an iterative method to optimize the loss of a predictive model for the training set. It uses the first order derivative of the loss function. At each batch, it calculates in which way the weights should be adjusted so that the loss can reach a minimum. The learning rate parameter determines the size of this adjustment step at each iteration. Through backpropagation, which calculates gradients for the weights, the loss is transferred from one layer to another, and the weights are modified depending on the loss. In our applications below, we will use the 'Adam' optimizer to perform this task and vary the learning rate. Processing the entire training set once is called an epoch. The loss-function determines how the difference between output of the model and true value is computed. When the output is a real number, the mean squared error is a an obvious choice.

A simple approach is to train a model with a predetermined number of epochs. Another possibility is to apply 'early stopping', in which after every epoch the model is applied on a validation set (this validation set is no part of the training set and therefore not involved in the estimation of the weights) and compute the validation loss. Then the training can stop when this validation loss does not improve anymore for several epochs, and the weights with the smallest validation loss can be used.

Types of ANN

In this paper we consider four types of ANN: Dense neural networks (DNN), Recurrent neural network (RNN), Long short-term memory (LSTM) and Neural Prophet. The first three ANN are based on the same principles, the only difference is the structure of the units and edges. Neural Prophet is different from this, as it also uses methods from the classic time series analysis. Here we describe some general properties of DNN, RNN and LSTM. For information about Neural Prophet see Section 2.2.4. In this section we shortly describe two other methods (deep state space and N-Beats, see section 2.2.5 and 2.2.6). The application of these methods is left for further research.

The methods DNN, RNN, LSTM discover for themselves whether there is a seasonal pattern, this is not given in the model specification. The user can add this knowledge about the seasonality take into account in the choice of the right values from the past as features.

The DL-methods DNN, RNN, LSTM considered in this paper only predict future values of the series without computing the components. When this decomposition is necessary in a particular application, it could be possible to develop a method which combines DL prediction with a traditional method. For example for seasonal adjustment it is necessary to predict the series before the actual seasonal adjustment can be carried out. Normally, ARIMA is used to compute the predictions, this could be replaced by DL.

2.2.1 DNN

There are many tutorials on the internet which demonstrate applications of DL for time series. Yang (2020), is one of them which shows some relatively simple code using the package Keras in Python to forecast time series. The figures in this tutorial looks quite promising, so it is worth trying to reproduce this on our own series. The first method of this tutorial is DNN. In the tutorial, the model consists of the input layer, two hidden layers and the output layer. In our experiments, we tried models with one, two, or three hidden layers and different numbers of units in the layers.

2.2.2 RNN

RNN (Recurrent neural network) is the second method from Yang (2020). Whereas the data flow in a DNN is unidirectional, in a RNN feedback loops are possible.

2.2.3 LSTM

LSTM (Long short-term memory) is the third method from Yang (2020). It is a special kind of a RNN, consisting of quite complex so called LSTM-cells. See also Hochreiter and Schmidhuber (1997). These cells are constructed especially for sequence data such as time series. They makes it easier to remember important information from periods in the past, and forget other information. Apart from an input value, they take as input a long-term memory component (known as the cell state) and a hidden state. The latter two are output of the previous LSTM cell. Within the LSTM cell, these inputs are processed through filters which are each their own neural network, updated by activation functions. These filters determine which part of the long-term memory should be forgotten (given less weight), and which part of the new input or hidden state should be added to the memory and how the hidden state is altered.

2.2.4 Prophet and Neural Prophet

Prophet and Neural Prophet are two time series methods developed by Facebook and published in 2017 and 2020 respectively. Prophet is the predecessor to Neural Prophet, where the latter includes DL. Although not a ML method, we therefore first discuss Prophet in this section.
Whereas the previous methods in this section (DNN, RNN and LSTM) only predict the next value (or next values) in the series, Prophet and Neural Prophet are able to decompose a series into different unobservable components.

The Prophet forecasting model

Prophet is a procedure for forecasting time series data developed by Facebook, see Taylor and Letham (2017). Prophet allows for non-linear trends. The model is fit based on traditional methods. The time series model that Prophet uses is the following additive model

$$y_t = T_t + S_t + H_t + e_t \qquad (1)$$

where

- y_t is the observable
- T_t is the trend

- S_t is the seasonal
- H_t models the effects of holidays
- e_t is the irregular term

Each summand of the right hand side of (1) is said to be a component and since it cannot be directly observed (as opposed to y_t) it is an unobservable component. The deviating parts of this model (as opposed to a standard component model, see e.g. Durbin and Koopman, 2012) are therefore the absence of a business cycle and the inclusion of a special holiday effect. The aim of Prophet now is to view the forecasting problem as a curve-fitting problem and consequently solve the latter. This is a fundamentally different approach from the standard approach of e.g. Durbin and Koopman (2012).

We will now describe the three individual unobservable components in some more detail. For more information, see Taylor and Letham (2017).

Prophet accepts two possible functions as trend: a continuous piecewise series consisting of either non-linear saturating growth functions or an affine function. In both cases turning points or change points must be specified. These are time stamps between which the same particular function is valid. Prophet lets the user either preselect these time stamps $t_1, \dots, t_l$ or give an upperbound for l (the number of time stamps or change points) and let Prophet automatically select them. The continuous piecewise nonlinear saturating growth takes growth in natural ecosystems as an example (Hutchinson, 1978) and uses the following function

$$g_t = \frac{C}{1 + \exp(-k(t - m))} \qquad (2)$$

with C the carrying capacity, k the growth rate and m an offset parameter. It is subsequently modified to better suit growth in economic activities by allowing both C and k to be time dependent. They can then be defined by the user. If not, Prophet assumes all variables are constant and calculates fitting values, although documentation does not give explicit details.

The continuous piecewise affine function has the simplicity of affine functions, yet it is also adjustable enough to meet real-life problems due to having this property not globally but piecewise. At each timestamp $t_1, \dots, t_l$ the constituent functions change slope. After fitting a trend based on a given past, the trend is extrapolated for forecasting using the last function.

The seasonal function s_t is modelled using standard Fourier series:

$$s(t) = a_0 + \sum_{n=1}^{N} \left(a_n \cos\frac{n\pi t}{L} + b_n \sin\frac{n\pi t}{L} \right)$$

Where L has to be chosen by the user according to the periodicity of the series. There are $2N + 1$ parameters to be estimated. The truncation number N itself may be chosen arbitrarily, but not too big (risk of overfitting) and not too small (risk of mis-fitting) either. The preprogrammed values are N = 10 for monthly seasonality

(12 months within a year) and N = 3 for daily seasonality (7 days within a week). According to Taylor and Letham (2017) these "work well for most problems". For other periodicities, other values for N should be chosen.

The user may identify a list of holidays per year. For each holiday i there are D_i corresponding dates. These too must be identified by the user. Prophet has a predefined set of the holidays for a number of countries. Prophet estimates a κ_i for each holiday i, assuming an additive change κ_i for each element of D_i on the trend. This component is interesting for series of daily figures, where for each day it can be decided whether it is a holiday or not. In the case of monthly figures, it should be defined that an entire month is a holiday, which is generally not appropriate. In this part, Prophet is less flexible than for example seasonal adjustment with X13-ARIMA. The latter method can take into account that for example the figure for March or April is influenced by the Easter holiday.

The fitting procedure for model parameters uses Limited-memory BFGS.

Neural Prophet

The time series model that Neural Prophet uses is the following additive model

$$y_t = T_t + S_t + H_t + F_t + A_t + L_t + e_t \quad (3)$$

where

- y_t, T_t, S_t, H_t and e_t are defined in the same way as for Prophet.
- F_t models regression effects for future known exogenous variables (future regressors)
- A_t models auto-regression effects
- L_t models regression effects for lagged observations of exogenous variables (lagged regressors)

As for Prophet, the summands of the right hand side of (3) are unobservable components.

The deviating parts of this model (as opposed to Prophet proper) are

- F_t , A_t and L_t all use the AR-Net. AR-Net is a simple autoregressive neural network for time series developed by Meta (Facebook)

We see some differences of NeuralProphet compared to Prophet. First, there are some more components. Second, NeuralProphet is trained with DL methods (based on PyTorch). The makers of NeuralProphet claim that NeuralProphet outperforms Prophet. Neural Prophet is furthermore based on PyTorch where Prophet was based on STAN (a state-of-the-art platform for statistical modeling and high-performance statistical computation).

The fitting procedure for model parameters of NeuralProphet uses mini-batch Stochastic Gradient Descent.

- The loss function is the Huber loss, a loss function less sensitive to outliers in data than the standard squared error loss.

- Regularization. The weight values are scaled in a predefined way, for details we refer to Triebe et al (2021).
- Optimizer. The optimizer can be set by the user but the default value is the AdamW optimizer
- The learning rate can be set by the user but a default value is calculated as well.
- The batch size can be set by the user but a default value is calculated as well.
- The number of epochs can be set by the user but a default value is calculated as well.
- The learning rate scheduler is “Icycle”.

Further features:
- Automatic filling in of missing data
- Normalization of data is possible using a handful of preprogrammed options.
- Hyperparameters have defaults proven in practice

Pro’s and con’s for Prophet and Neural Prophet

Pro:
- They offer a free and (once correctly installed) relatively easy to use python package with which a number of standard approaches to modeling and forecasting are offered.
- Can handle large datasets
- Automatic selection of training related hyperparameters.
- Fourier term seasonality at different periods such as monthly, daily, weekly, hourly.
- Plotting for forecast components, model coefficients and final predictions.
- (Only Neural Prophet) supports auto-regression, covariates, lagged and future regressors

Con’s:
- as with many larger python packages, the interdependencies underlying the main package may cause problems when installing
- the package is specifically designed for certain business problems and does not address other issues (e.g. time dependent margins)
- the definition of trend is not based on a distribution of residuals but instead is treated as a curve-fitting problem
- the number of default change points is so large that trends are approximated unrealistically

2.2.5 Deep state space

The traditional structural time series models are sometimes also called state space models. More precisely, a structural time series model has to be written in state-space representation in order to be estimated.

A structural time series model assumes that the series consists of components like trend and seasonal. These components are called state variables. Then a structural time series model consists of two linear equations. The first equation, the transition equation, describes how the state variables evolve over time. The second, so called measurement equation, describes how the observed series

depends on the state variables. When these two linear equations are replaced by an ANN, the result is a deep state space model (Gedon e.a., 2021).

2.2.6 N-BEATS

Kafritsas (2021) describes relatively recent developments of deep time series analysis. In this tutorial, four methods are mentioned. We restrict ourselves here to the first one, N-BEATS (see also Peixeiro, 2022). This is a ready to use neural network for time series forecasting, more sophisticated than the ones from Yang (2020), see Section 2.2.1 – 2.2.3. In Oreshkin e.a. (2020), N-BEATS is applied on a large set of time series of different length and different characteristics, with, according to the paper, promising results.

3. Setup of empirical and simulation studies

In the next section, we present the results of a simulation and an empirical study of the performance of deep learning methods. Before presenting these results, in the current section, we describe the setup of these experiments.

3.1 Software

For the empirical and simulation experiments we used Python and R (R Core Team, 2018). The deep learning models DNN, RNN and LSTM were programmed in Python 3.9 with use of the packages Tensorflow, Keras and Scikit-learn. For the experiments with Prophet and Neural Prophet the corresponding packages were used. For modelling the ARIMA-models Python 3.9 with package Statsmodels was used, while Structural times series models (STM) were computed in R with the use of the KFAS package (Helske, 2017), which implements the models from Durbin and Koopman (2012).

3.2 Datasets

In order to study the possible value of applying DL to time series problems, we consider several datasets with different characteristics.

The first dataset consists of simulated data, and contains three artificial series. This allows us to systematically vary the properties of the series, most importantly the length. Empirical series are mostly relatively short for ML algorithms, which are typically applied to large datasets. This simulation study also allows us to vary the amount of noise present in the series, relative to the seasonal pattern, and the

complexity of the neural network that is used. This study gives us some first ideas on the situations in which DL may be beneficial for times series applications.

The three artificial series have a length of 5300 periods. As we will see, with this length quite accurate predictions are possible. The last 300 periods are used as test set, the other periods are available as training set. By selecting a larger or smaller part of this set, the influence of the length of the training set is investigated. We have the following three series:

- A trend (computed as moving average of a random walk) plus a large seasonal pattern (of length 12, not changing over time) plus noise
- The same trend plus a similar but smaller seasonal pattern plus noise
- The same trend plus noise

The series are created using basic functions of Excel. These series are quite regular. See the figures in Section 4 for an impression.

The first empirical dataset is the well-known airline series (Box et al., 1994). This monthly time series was chosen since it shows a clear and 'predictable' seasonal pattern and a smooth upward trend. Also, there are no outliers or other effects that make fitting a model more difficult. This series can therefore be easily modelled with classical methods, such as with the well-known airline ARIMA-model (Box et al., 1994), and predicting future values should not be difficult. It is therefore the first real data example to study how we can achieve plausible results with ML, that have similar out-of-sample performance compared to classical methods.

As a second empirical series, we use consumption of households. The monthly series is available since 2000. We include the series until December 2019 (due to the corona crisis). The seasonal pattern in this series is relatively regular, compared with other real series. We test two auxiliary series for this real series. First, we use the original series, slightly changed by adding some noise. Second, we use consumption of clothes. In both cases, we use the value of period t of the auxiliary series to predict the target series for period t. This is in this situation not realistic, it is meant as a demonstration of the method.

After gaining some insights from these datasets, we move to an actual nowcasting problem. In this case, we will make a nowcast for Dutch GDP, which is a quarterly series at a high level of aggregation, with no assumed working day pattern or outliers. The GDP data itself is the series in constant prices (2015=100), running from 1995Q1 to 2021Q1. In order to avoid estimation problems arising around the observations affected by the Covid crisis, we will first truncate our dataset at the first quarter of 2020, and later also study the series including that period. In order to make nowcasts for a certain reference period, we use the historical data of the GDP series and combine this with current information from auxiliary data. In our case, the latter consists of a number of macro-economic series that show high correlation with GDP and should help to improve the out-of-sample estimates of the target series.

Preprocessing is an important step in time series analysis. It can consist of steps such as outlier detection (and removal), correcting for structural breaks, seasonal adjustment, calendar adjustment, and detrending or transforming the data (in order to make the series stationary). Whether all these steps are necessary, depends on the applied method and on the time series. All of these steps involve an explanatory time series model, or at least try to explicitly deal with a certain effect. In ML, at least some preprocessing should be done in order help the numerical optimisation, such as normalising the inputs. Preprocessing is a very important step to obtain accurate results, as this determines the algorithm's ability to learn. We can also apply the techniques from classic time series analysis, and effectively create a combined classic and ML approach. In this first study, the amount of preprocessing was limited, in order to discover how robust the algorithms are to the abovementioned problems.

3.3 Evaluation

The quality of the models is evaluated by looking at figures where the predictions are compared with the true values. Furthermore, different indicators are computed to compare different models. These are

- the Mean Error (ME), which measures the average difference between the actual values and the predicted values. Ideally, on average positive and negative errors should cancel out, and this measure should be close to zero.
- the Mean Absolute Error (MAE), which measures the average absolute difference between the actual values and the predicted values.
- The Root Mean Squared Error (RMSE) is the square root of the MSE. The MSE is the sum of the squared differences between the actual values and the predicted values. The squaring (besides getting rid of negative numbers) also "punishes" large errors, i.e. their impact on the mean is enlarged.
- The Mean Absolute Percent Error (MAPE) is the average absolute percent difference between the actual values and the predicted values.

These indicators are computed as follows:

$$\mathrm{ME} = \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)$$

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i|$$

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2}$$

$$\mathrm{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i}{x_i} - 1\right| * 100$$

With n = total numbers of observation, y_i = actual value for the ith observation and x_i = predicted value for the ith observation.

MAE and (R)MSE depend on the order of magnitude of the series and can only be used to compare different models for the same data. Because of its relative nature, The MAPE can also be used to compare results of different data sets.

4. Results

In this section we discuss our experiences with applying different methods. In the first part of this section, we show the initial challenges to achieve acceptable results. In Section 4.3 we show that by stepwise improvement of the parameters the results can be improved substantially. Also in Section 4.5, the predictions are quite accurate. These results are based on the experiences of the first part of this section.

4.1 Instable estimation

When a neural network is trained, the program tries to find the optimal set of weights. Generally, a local optimum is found. The training of the model starts with randomly chosen weights, which are improved by the optimization procedure. When the model is initialized again and the training is repeated, often another optimum is found, due to the randomly chosen starting points and the complexity of the models. In Figure 4.1, we show the predictions which are based on different local optimums of exactly the same models (details about the figures will be explained later). We see that the second local optimum is much better than the first one. Note that there are different methods to choose the starting weights. Comparing these methods is left for further research.

The fact that there are different local optima is not always a problem. The neural networks are complex, and it is possible that the same (or a similar) accuracy can be achieved with very different weights. But Figure 4.1 shows two solutions with different accuracy. This problem could be a sign that the model is somehow miss-specified. One kind of miss-specification is a too complex model given the size of the training set. This is definitely the case in some of the cases which are described later. However, the problem of different local optima also occurs when the training set is larger or when the model is less complex, and is happens with RNN and LSTM as well. All models are trained several times. In the continuation of this section, we present the results based on the best training result, given the RMSE. This problems with the local optimums are a serious problem when we want to find out which model specification is to be preferred in a specific situation.

There are some other reasons for problems with local optimums, like a non-optimal learning rate or not applying regularization methods for the weights. First

steps in exploring these possibilities will be shown in the continuation of this section. These possibilities can be investigated in more detail in further research.

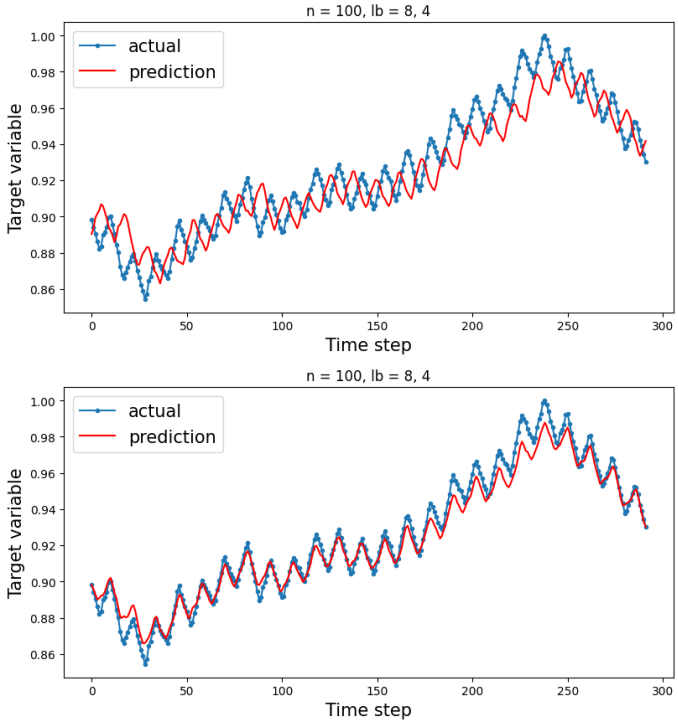


Figure 4.1: result for DNN-model for artificial series with small seasonal pattern, for upper panel the RMSE = 0.014, for the lower panel RMSE = 0.0055

One of the local optimums is found to be a constant, the mean of the training set. Figure 4.2 shows such a result. We found that this happened quite often under RNN, but sometimes also under DNN and LSTM. RNN's are known to be difficult to be trained, due to the so-called vanishing gradient problem, where the gradient goes to 0 and the weights are no longer updated and thus the model stops learning. Initializing and training the model again can sometimes (but not always) help to find a better solution.

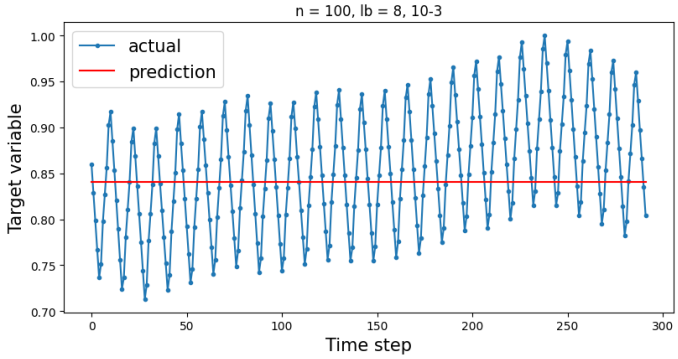


Figure 4.2: result for RNN-model for artificial series with large seasonal pattern

4.2 Artificial series

4.2.1 Results for DNN, RNN and LSTM

Table 4.1 shows which parameters are tested on the DNN model for the three artificial series, as well as the corresponding RMSE computed on the test series. Column 2 of the table shows the length of the training series which is selected from one of the three artificial series. This is the part of the series which is used to create the training set. The number of elements in the training set is slightly smaller since there is some "loss" as some periods are used as lag. The number of lags is shown in column 3 of the table. Early stopping (where the algorithm is stopped if it does not further improve the validation loss for a while, in this case 5 epochs) is used in all cases. The number of epochs before the training stops varies between a few dozen and almost a thousand. A learning rate of 0.0005 is used after some experiments. It is quite possible that this parameter can be improved.

The models consist of the input layer, one or two hidden layers and the output layer. The number of units in the hidden layers is given in column 4 of the table. When there are two values in this column, there are two hidden layers, otherwise, there is only one hidden layer. In Figure 4.3, 4.4, and 4.5, the predictions for all of the specifications in Table 4.1 are compared with the true values of the series. The specifications of the models are given in the head of the figure (length of training series n, lb for number of lags, and the number of units in one or two hidden layers). The abbreviation lb stands for look-back which is another term for lags. For the computations for the series with a large seasonal, we can conclude that with a longer training series more accurate model predictions are possible. But even with 100 periods in the training series, the predictions follow the trend and the seasonal pattern quite well. Including more lags (of 8 of 16) in the model seems to let the model recognize the seasonal pattern better, but even with 2 lags, the seasonal pattern is quite well-predicted (but too small). The estimates for the seasonal pattern are also too small with 100 periods in the training series. There, the development of the trend is also slightly underestimated.

When we look at the examples for the series with small seasonal, there seems to be a large difference between 8 or 16 lags, combined with a training series of length 1000 and 16 and 8 units in the hidden layers. This is, however, probably the result of many attempts to train the model. Also for the model with 8 lags, mostly a local optimum with an accuracy around the one of the other model (with 16 lags) is found. Only once, the very accurate optimum is found which is shown in the table and the figure. It is possible that there is a similarly accurate local optimum also for the model with 16 lags, which is not found during the training attempts.

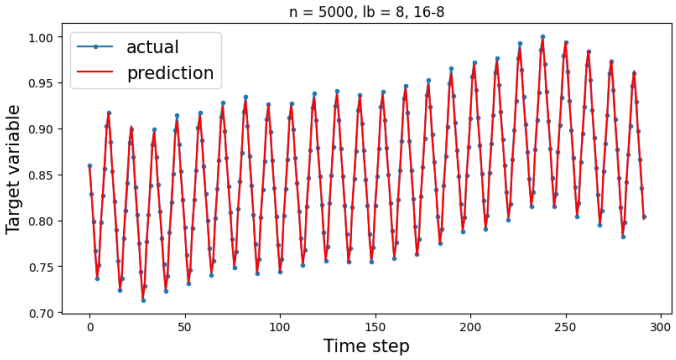
For a series without a seasonal pattern, there is a simple idea for prediction: just use the value of $t-1$ to predict t. This should be quite accurate and can be found without using an ANN. When the length of the training series is 5000, the model predictions of the DNN are close to this simple idea. This changes when the length of the training series is shorter. It seems plausible that this simple solution is easier to find (among many other local optimums) when the model is parsimonious, i.e. a few lags and a few units. For the length of 1000, this seems to be true, for the

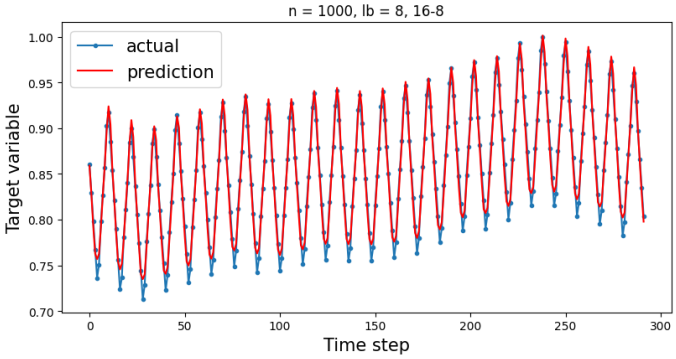
length of 100, it is not. This is, again, probably a result of many local optimums, and which solutions are coincidently found.

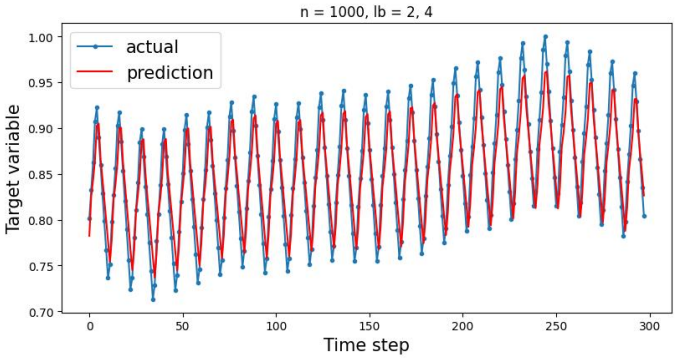
It seems that a model with 2 lags could be appropriate for a series without a seasonal pattern. For a series with a seasonal (monthly) pattern, a model with more lags seems to be better. It is remarkable that it does not seem necessary to have 12 lags (or at least the 12th lag) in this application with a monthly seasonal pattern. The problem with local optimums makes that we cannot be sure about this conclusion. Nevertheless, the figures show that the seasonal pattern is found quite well, even with 2 lags.

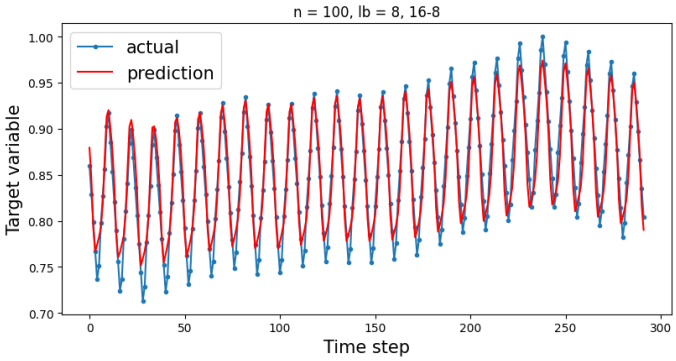
Table 4.1: Results for DNN with artificial series

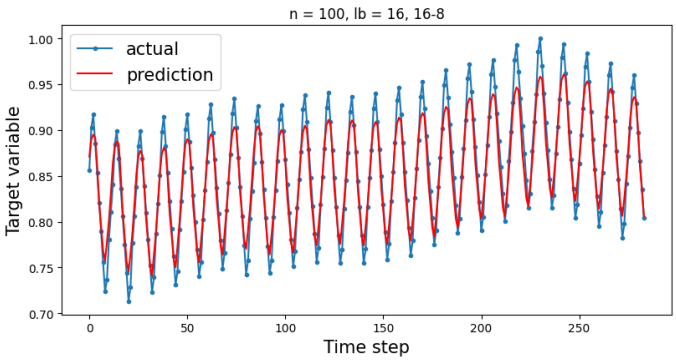
Series	Length training series	lags	units	RMSE
Large seasonal	5000	8	16-8	0.0037
Large seasonal	1000	8	16-8	0.0091
Large seasonal	1000	2	4	0.017
Large seasonal	100	8	16-8	0.017
Large seasonal	100	16	16-8	0.016
Large seasonal	100	8	4	0.011
Small seasonal	1000	8	16-8	0.0031
Small seasonal	1000	16	16-8	0.013
Small seasonal	100	16	16-8	0.0100
Small seasonal	100	8	4	0.0055
No seasonal	5000	8	16-8	0.0012
No seasonal	1000	8	16-8	0.0061
No seasonal	1000	2	4	0.0013
No seasonal	100	8	16-8	0.0041
No seasonal	100	2	4	0.0068


n = 5000, lb = 8, 16-8
actual
prediction
Target variable
Time step


n = 1000, lb = 8, 16-8
actual
prediction
Target variable
Time step


n = 1000, lb = 2, 4
actual
prediction
Target variable
Time step


n = 100, lb = 8, 16-8
actual
prediction
Target variable
Time step


n = 100, lb = 16, 16-8
actual
prediction
Target variable
Time step

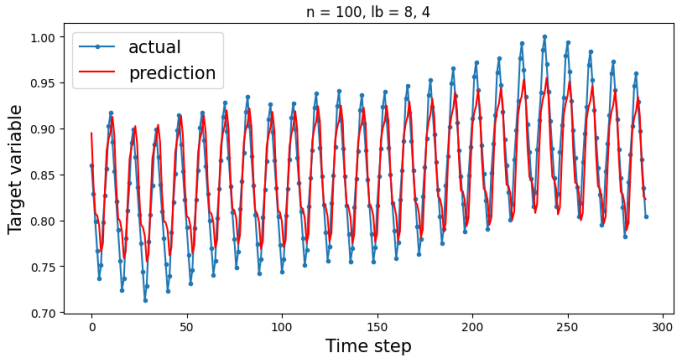
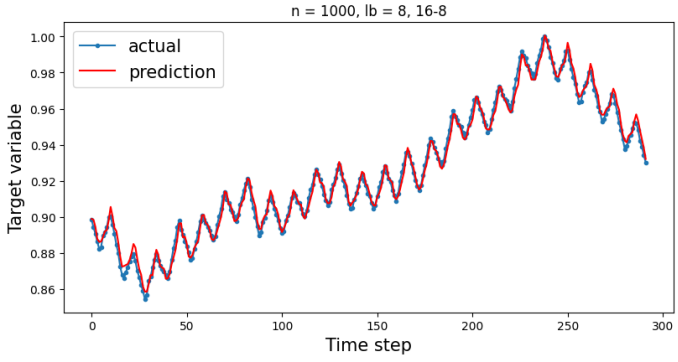
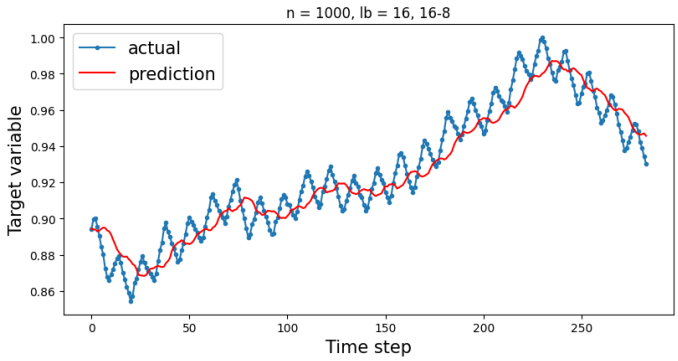


Figure 4.3: results with DNN-model for series with large seasonal pattern





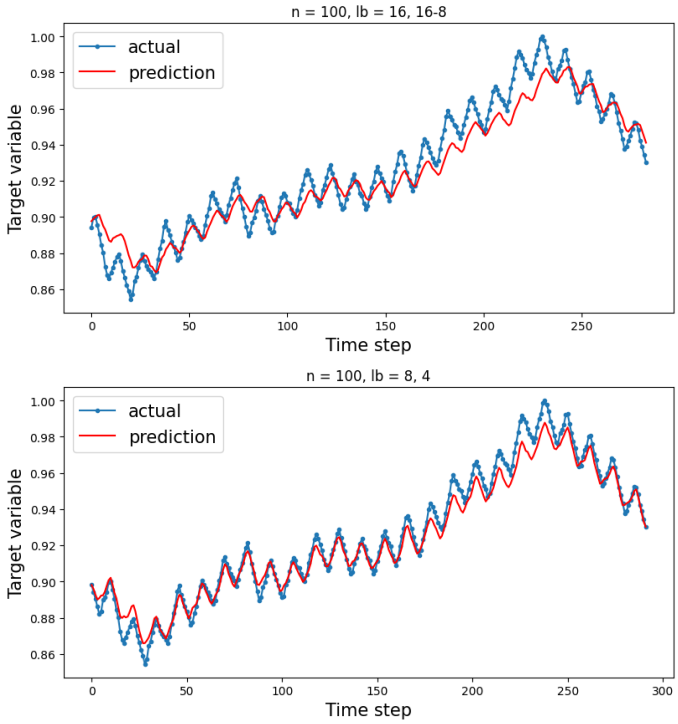
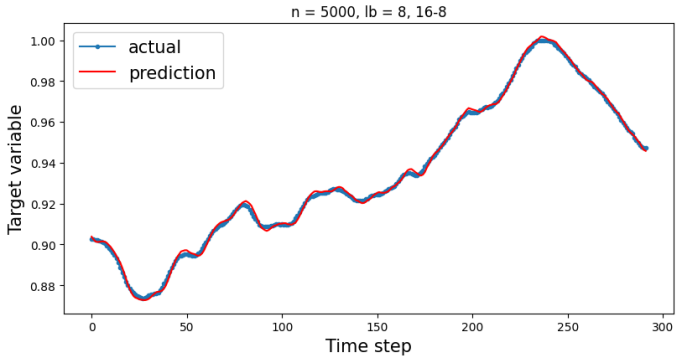
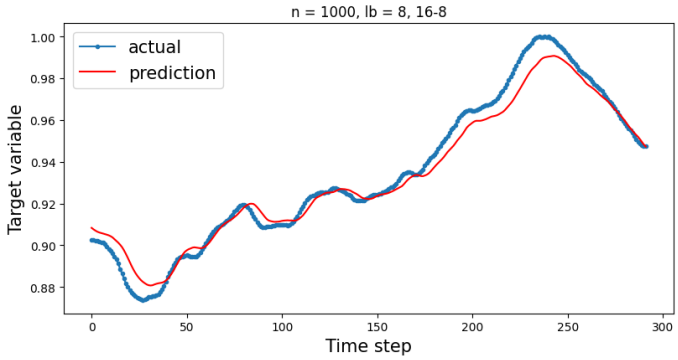
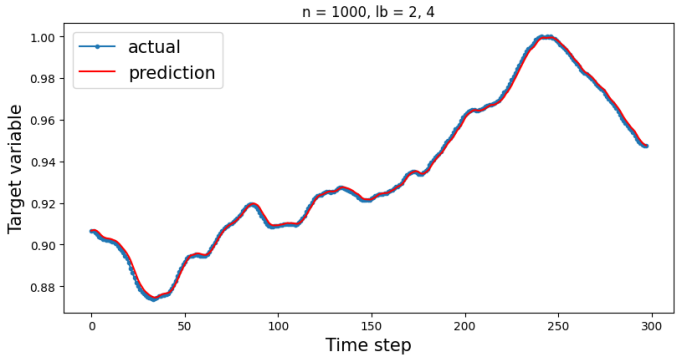
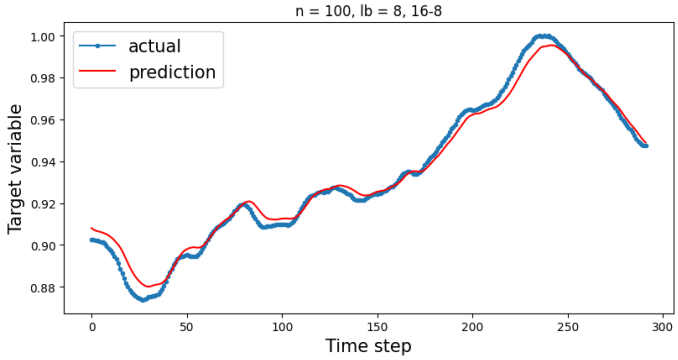


Figure 4.4: results with DNN-model for series with small seasonal pattern


n = 5000, lb = 8, 16-8
actual
prediction
Target variable
Time step


n = 1000, lb = 8, 16-8
actual
prediction
Target variable
Time step


n = 1000, lb = 2, 4
actual
prediction
Target variable
Time step


n = 100, lb = 8, 16-8
actual
prediction
Target variable
Time step

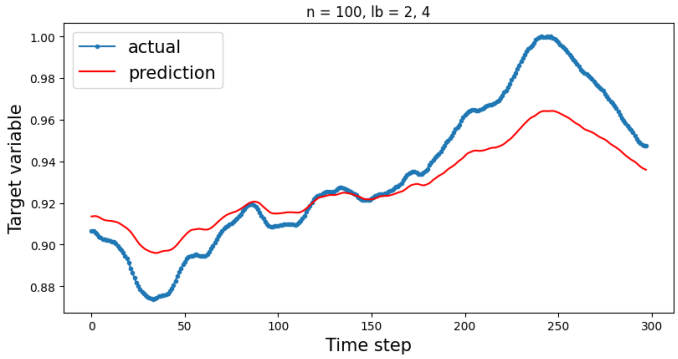
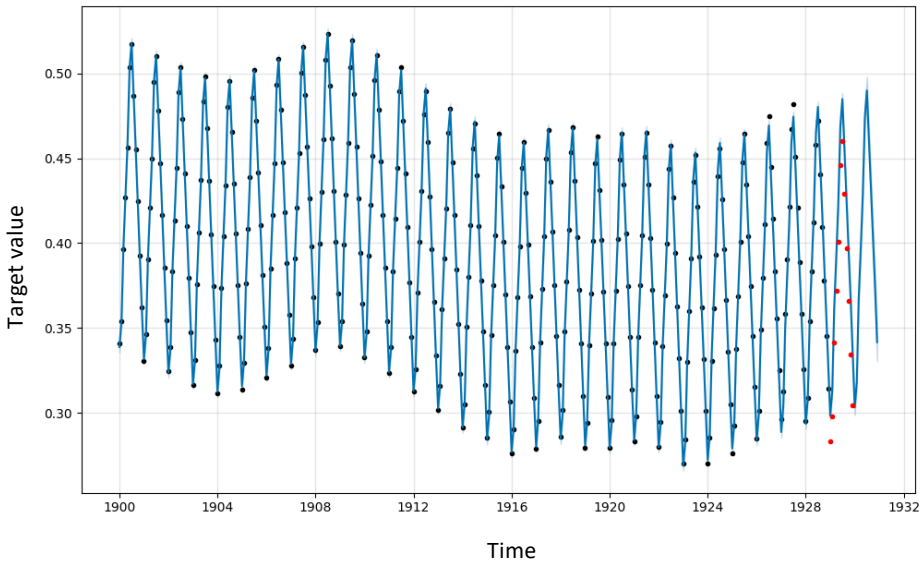


Figure 4.5: results with DNN-model for series without seasonal pattern

RNN and LSTM models are also applied to these artificial series. Detailed results for RNN and LSTM can be found in the Appendix. We see that RNN is not promising at all. The accuracy of LSTM is sometimes comparable with the one of DNN, but not clearly better. In the continuation of the project, RNN is not used anymore.

4.2.2 Results for Prophet

The series with large seasonal effect was also tested with Prophet and Neural Prophet. For this, the first 3000 points of the total series were selected. Figure 4.6 (top) shows the original series (black), the model estimates for the series (trend and seasonal) (blue) and predictions (red). This means that the part of the series where black points are added is used as input of the model. The last 12 points are predicted. This part of the originals series was not used by the model. The predictions meet the original series quite well. In the lower figure the change points (vertical red lines) and the estimated trend are visible. Here, we find an MAE = 0.002189 and RMSE = 0.04679. This is, however, not entirely comparable with the results in the previous section, since the predictions here are computed given the original series with the last 12 points excluded, whereas in the previous sections, one-step-ahead-predictions are computed.



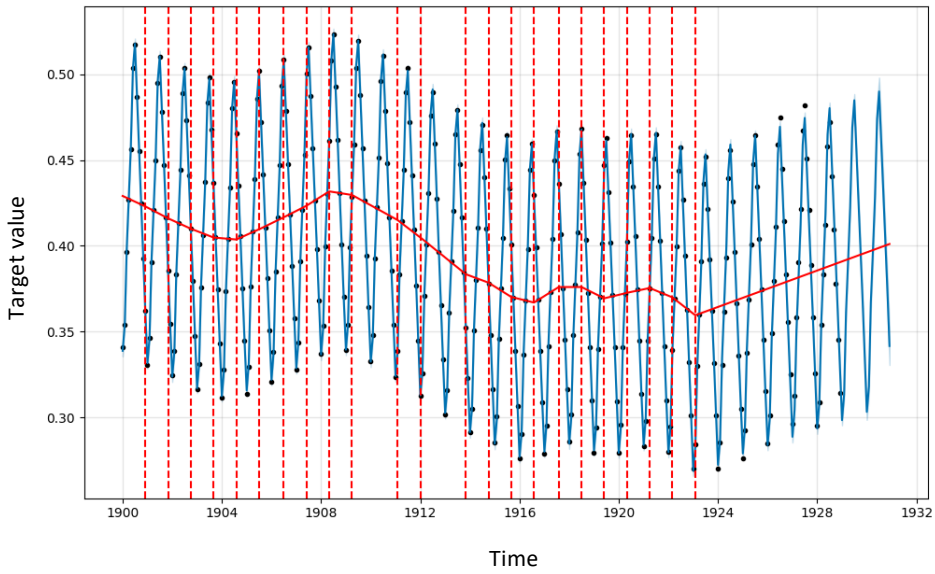


Figure 4.6: Above: the original series in black and calculated signal in blue. Predicted values (n=12) in red. Below: trend in solid red, change points at red dashed vertical lines

4.2.3 Results for Neural Prophet

The same series with large seasonal was also tested with Neural Prophet.

The model required 97 epochs to run in a few minutes. After the final epoch the relevant performance numbers are MAE: 0.0424, RMSE: 0.0537, Loss: 0.00145. This seems slightly worse than the results with Prophet. One example is not sufficient to draw general conclusions, but at least we cannot confirm that Neural Prophet always outperforms Prophet.

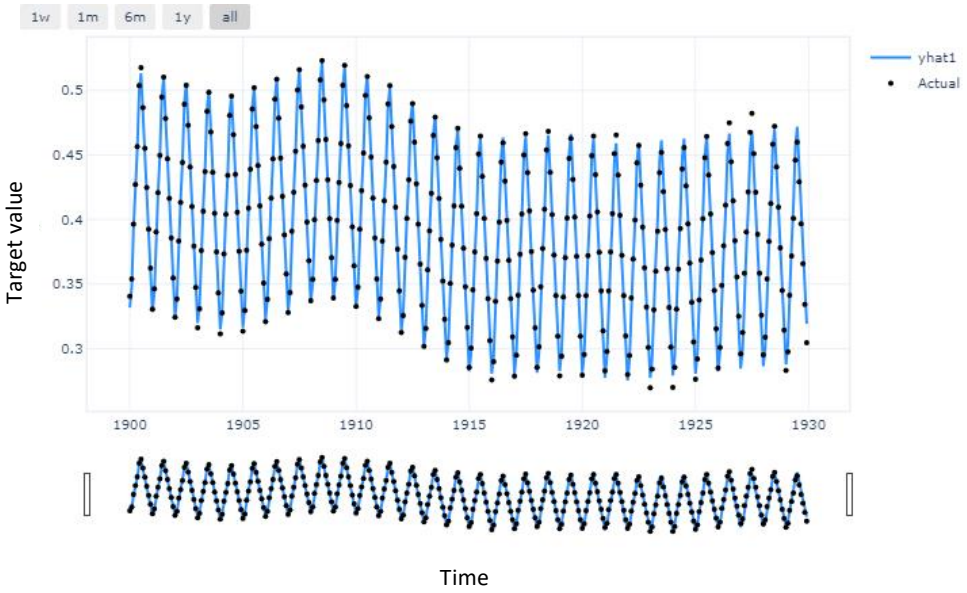


Figure 4.7: Original series in black, predicted values in blue

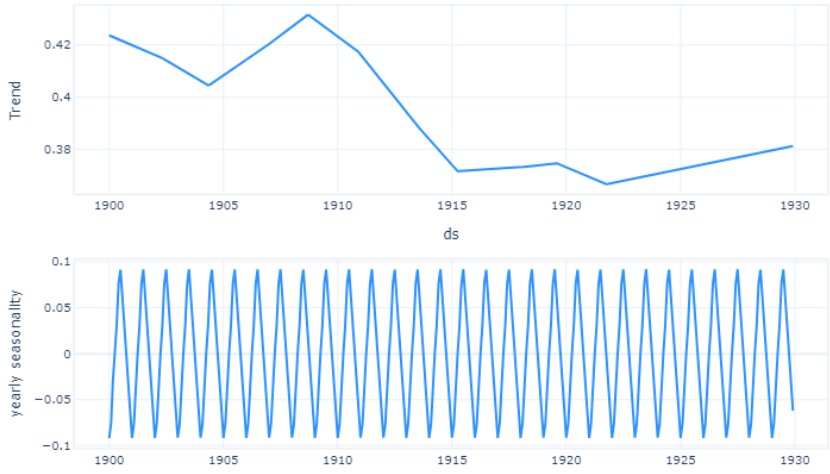


Figure 4.8: Decomposition of series with large seasonal into trend and seasonal by neural prophet

4.3 Airline data

Since this time series behaves quite regular, with a slight upward trend and a significant seasonal pattern, it should not be difficult for an ANN to discover these patterns and make extrapolations. We will start with a simple DNN and increase complexity and refine the model and parameters based on several accuracy measures, which are measured over the entire training and test set. An important tool in assessing the quality of the model is the learning curve. The learning curve is a graphical representation of the loss function from epoch to epoch during training. It informs us whether the algorithm has converged, and about the performance (loss) of training and test set. Note that the learning curve was also used in the previous section but in a more implicit way.

We start with a simple network architecture, namely a network with one or two hidden layers, and one output layer, where the hidden layers consist of only a few neurons. Our first attempt to model the airline data is to take as input only lags 1 to 5 of the target series, so $X = (y_{t-1}, .., y_{t-5})$. For now, we ignore the seasonal component and do not include any seasonal lags. The outcomes of the training procedure should then inform us how we can improve the model. In order to model this simple input, our network consists of one hidden layer with 2 neurons, about half of the number of input variables, and a final output layer. We denote this as an DNN(2,1) network. Further settings are: learning rate 0.001, epochs=200, batch size=2, training set: 108 periods, test set: 24 periods. We did not apply early stopping in this case.

The results of training the model are shown in Figure 4.9. If we study the learning curves for the training and test set, we can see that there is no convergence yet, since both curves are still declining. Results will improve if we increase the number of epochs. Also, it is apparent that there is a clear gap between the learning curve of the training and test set. The gap is called the generalisation gap, since it

informs us about how well the fitted model will perform on new data. At the end of training procedure, the learning curve of the test set is therefore a bit higher than the curve for the training set. Ideally, the gap should be small, so that we can expect accurate out-of-sample predictions. In this case, the gap is quite large, indicating that the model is (not surprisingly) probably underfit. If we look at the resulting predictions in the graph on the right, the underfit of the model is confirmed. The fitted values and predictions are slightly lagged, as can be expected from a simple autoregressive setup with only the first couple of lags of the target series.

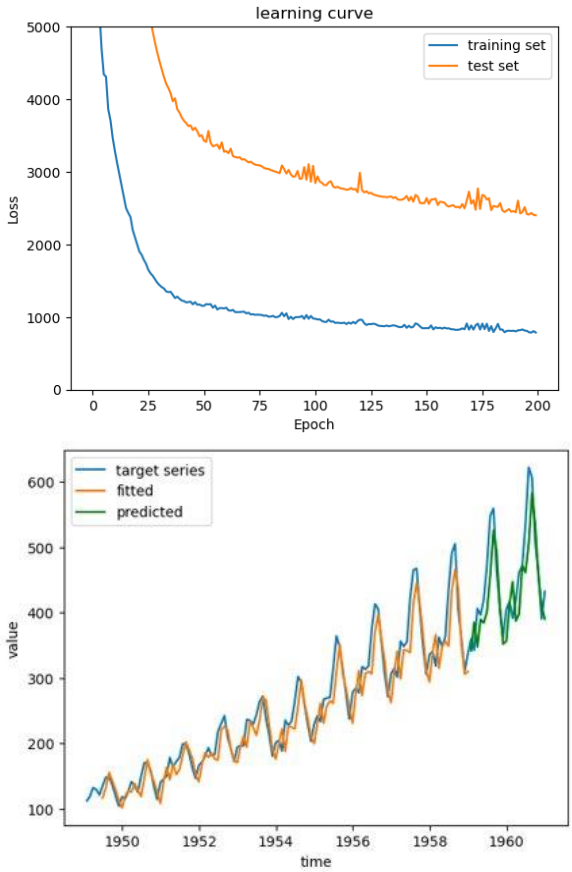


Figure 4.9: learning curve (above) and fitted and predicted values (below) for a DNN(2,1) network with as input lag 1-5 of target series.

In order to improve the model, we could try to add more informative data to the model or increase the complexity of the network, i.e. add hidden layers and/or neurons. In order to capture the seasonal pattern, we will use as input data lags 1 and 12 of the target series, so that $X = (y_{t-1}, y_{t-12})$. In Figure 4.10, the learning curves are shown for two DNN's. In the left panel, the learning curves for the same DNN(2,1) as above are shown. We can see that the gap between the curves is smaller already (Please note the y-axis is different than in the figure above). However, the learning curve is still declining and the generalisation gap is relatively large. In the panel on the right a slightly more complex network with two hidden layers and 4 and 2 neurons respectively is shown. We denote this as an DNN(4,2,1) network. This network architecture further improves the results, since the learning curves are now very close after 200 epochs of training.

The improvement over the first model with lags 1-5 is confirmed by the respective RMSE's in Table 4.2.

Table 4.2: Accuracy measures for the test set of 24 periods, network with 2 or 3 layers and 5 or 2 lags of the target variable

	lags	RMSE Training set	RMSE Test set
DNN(2,1)	1-5	27.98	49.00
DNN(2,1)	1,12	17.96	27.61
DNN(4,2,1)	1,12	15.27	18.53

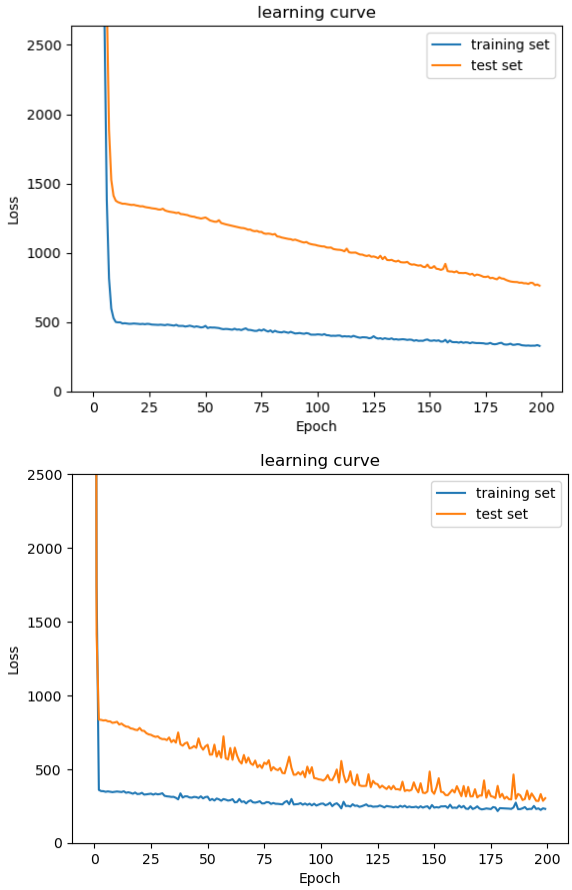


Figure 4.10: learning curves for fitted and predicted values for a DNN(2,1) (above) and DNN(4,2,1) network (below) with as inputs lag 1 and 12 of the target series.

After some further refinements, we found a neural network that gives accurate and stable results every time the training is repeated. In this case, we first rescale the data (by simply dividing by 100) to lie around 1 for numerical stability. We take as inputs lags 1-12 of the target variable and use a slightly more complex DNN(16,8,3,1) network. Also, the learning rate was decreased in order to reach a lower final value for the loss function, see Figure 4.11.

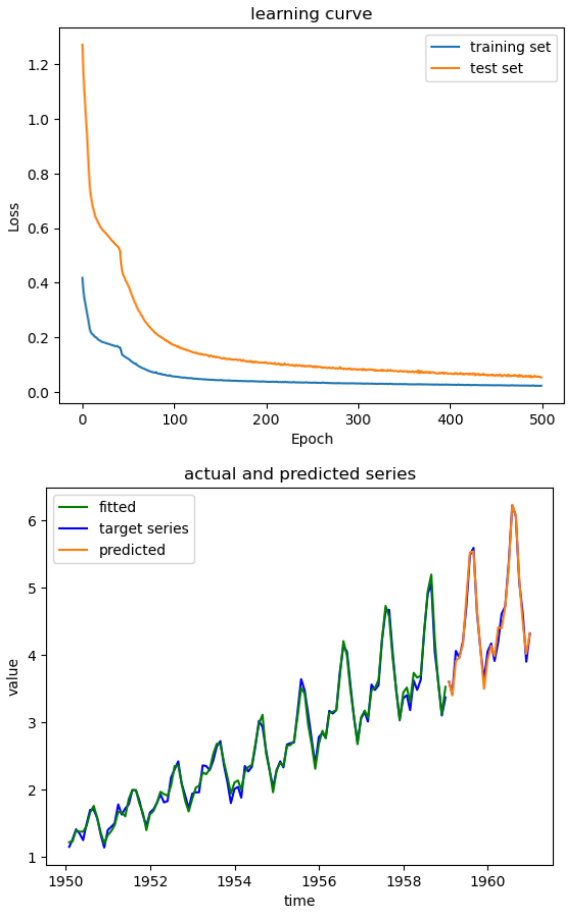


Figure 4.11: learning curve (left) and fitted and predicted values (right) for an DNN(16,8,3,1) network with as input lag 1-12 of target series.

We can now also compare the predictions to those of the classic time series methods. As can be seen in Figure 4.12 and Table 4.3, the predictions at first glance look similar. This is also reflected by the accuracy measures in the table below, which have the same order of magnitude. The exact value of the accuracy measures should only be seen as an indication that similar results can be obtained from each method. The exact figures of course depend on specific choices made for each type of model, and therefore the accuracy of individual methods could probably be improved if we optimise further.

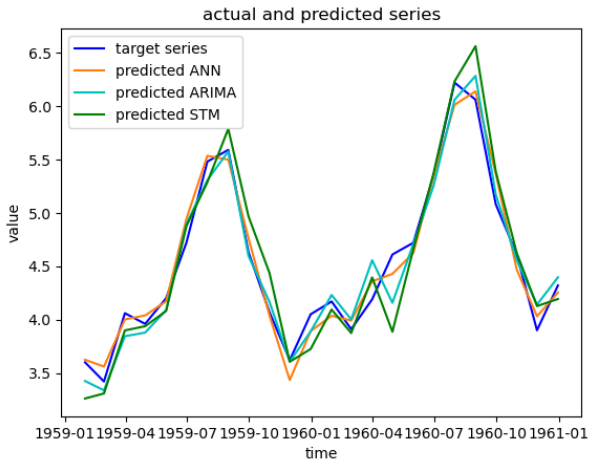


Figure 4.12: one step ahead predictions for DNN, ARIMA and STM models.

Table 4.3: Accuracy measures for the test set of 24 periods

	RMSE	MAPE	ME
ARIMA(0,1,1)(0,1,1)	0.1722	3.06	-0.0173
STM (smooth trend + seasonal)	0.2592	4.32	0.1577
DNN(16,8,3,1) with lags 1-12	0.1503	2.71	0.0077

As we have seen in Section 4.1, one important aspect of ANN's is that the results from the training algorithms are stochastic in nature. This means that in order to optimise the weights for the neural network the algorithm needs starting values of these weights. If no prior information is available, it is common to use random starting values. The consequence of this is that after training the final weights do not reach a global optimum, and the predictions from the network are not accurate. This, however, is not different from for example estimating a Structural time series model, where starting conditions for the Maximum Likelihood estimation of the hyperparameters are needed. The difference is that for Neural networks the starting weights are usually drawn randomly, leading to different outcomes each time the training process is repeated. It is too early to conclude whether the problem is more or less substantial for Neural networks compared to structural time series models, as we have not sufficient experience with Neural networks yet.

In order to check to what extent this applies to our model, we repeated the training procedure 25 times, and plotted the bandwidth of the predictions in Figure 4.13.

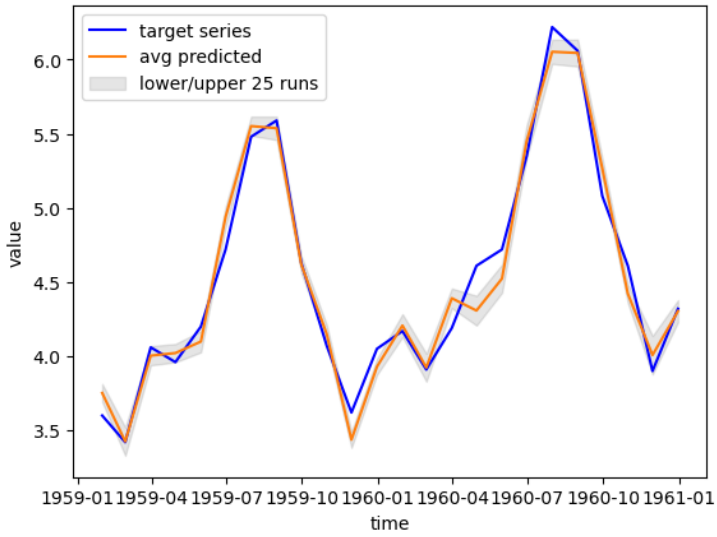


Figure 4.13: predictions for the test set based on 25 runs

For this final model, the predictions did not vary much (at least visually). In earlier attempts (Section 4.1 and 4.2), there was much more variation. In this case the following steps helped us get stable results: i) rescale the data in order to lie around 1, ii) include several lags of the target variable, especially the seasonal lag, while iii) increasing complexity of the network, iv) try several values for learning rate, number of epochs and batch size.

Now that we know that our model yields relatively stable results if we repeat training 25 times, how will the model perform if we repeatedly estimate the model in a real-life setting. For this, we perform cross validation over the test set, in which we make consecutive one-step ahead nowcasts, for which we re-estimate the weights every time new data becomes available. The result is shown in Figure 4.14 and Table 4.4. Accuracy measures are slightly better than above, because weights are now optimised every time step.

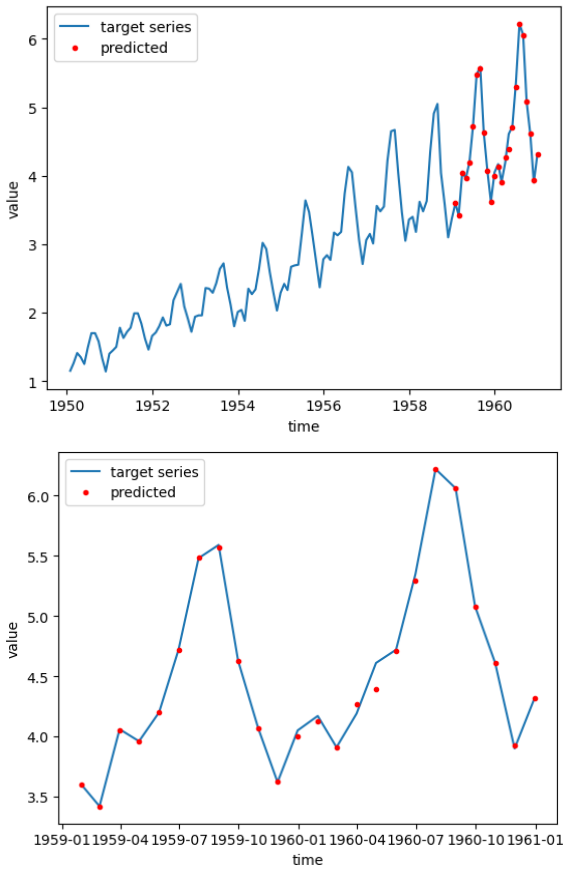


Figure 4.14: Cross validation of the 24 one-step ahead forecasts.

Table 4.4: Accuracy measures for a cross validation over the test set of 24 periods

	RMSE	MAPE	ME
DNN(16,8,3,1) with lags 1-12	0.067	0.80	-0.0149

The Airline data is also investigated with Neural prophet. The model required 434 epochs to run in a few minutes. After the final epoch the relevant performance numbers are MAE: 17.294933, RMSE: 22.160141, Loss: 0.001261. This is clearly less accurate than the results with the DNN. Figure 4.15 shows that Neural Prophet does not “see” that the seasonal effect is increasing.

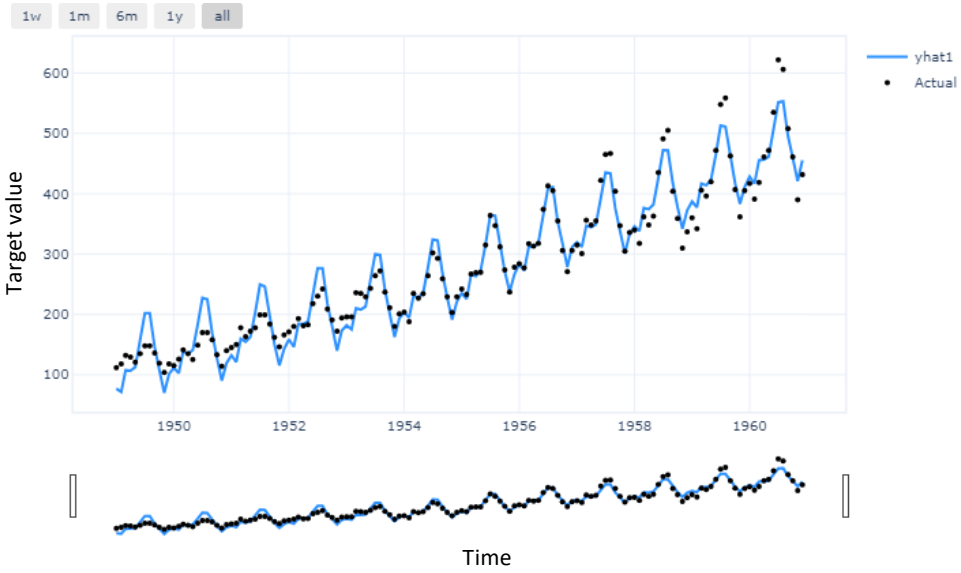


Figure 4.15: Prediction and actual values

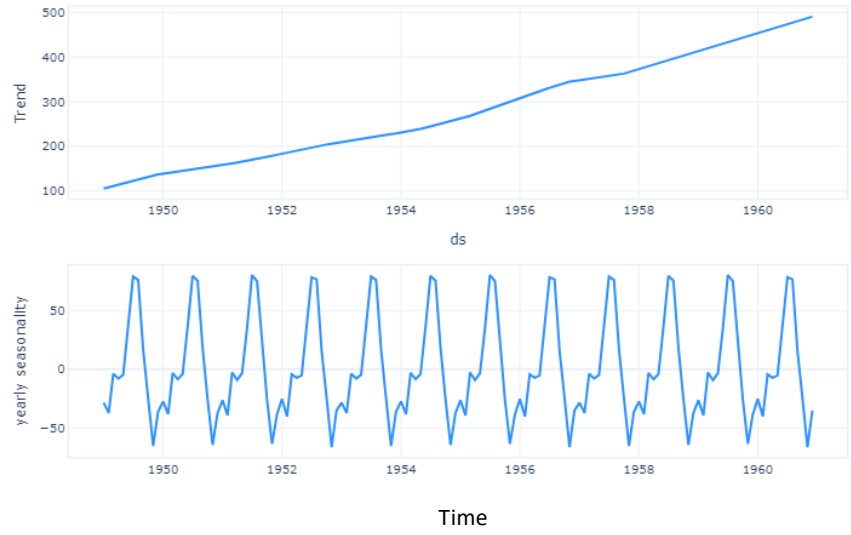


Figure 4.16 Above trend, below seasonality computed by Neural Prophet for airline data

4.4 Consumption households

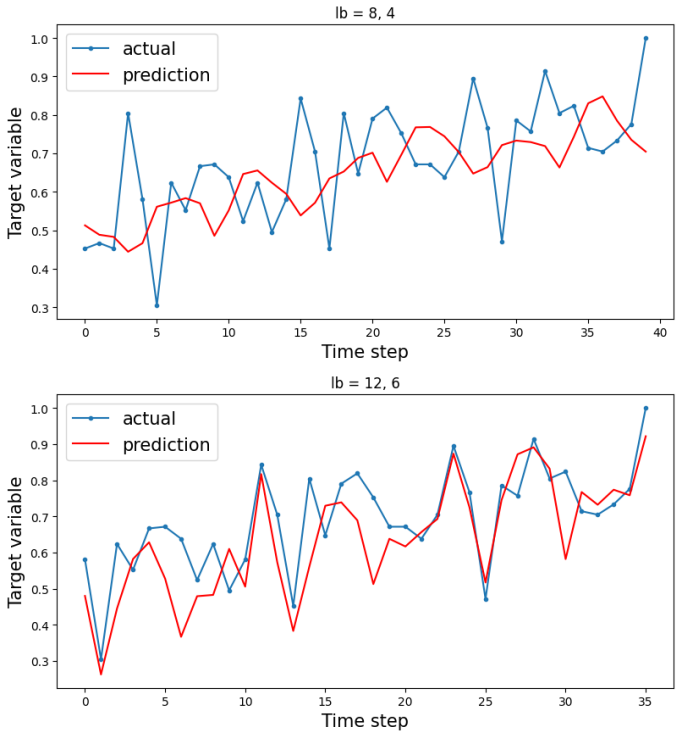
The next series we investigate is about consumption of households. Here, the length of the training series is always around 192, it only varies due to the choice a more or fewer lags). We only apply DNN-models and vary the specification of the model (lags, number of units, mostly 1, sometimes 2 hidden layers), see Table 4.5, Figure 4.17. Early stopping is used in all cases. The number of epochs before the training stops vary between a few dozen and almost a thousand. The learning rate of 0.0005 is used.

The smallest RMSE is found with 16 lags and one hidden layer with 6 units (as long as no auxiliary series is involved). There, the model predictions follow the trend of the true values quite well. Also the most pronounced part of the seasonal pattern is recognized.

The artificial auxiliary series (the original series with noise) clearly improves the accuracy of the model, see Figure 4.18. With real data, this is not the case (Table 4.5 and Figure 4.17, 4.18).

Table 4.5: results for real data with DNN

Auxiliary series	Lags	units	RMSE
No	8	4	0.148054
No	12	6	0.110815
No	12	4	0.108904
No	12	6-2	0.146765
No	12	4-4	0.124539
No	16	6	0.077974
Clothes	15	4	0.145499
Original series with noise	15	4	0.056125



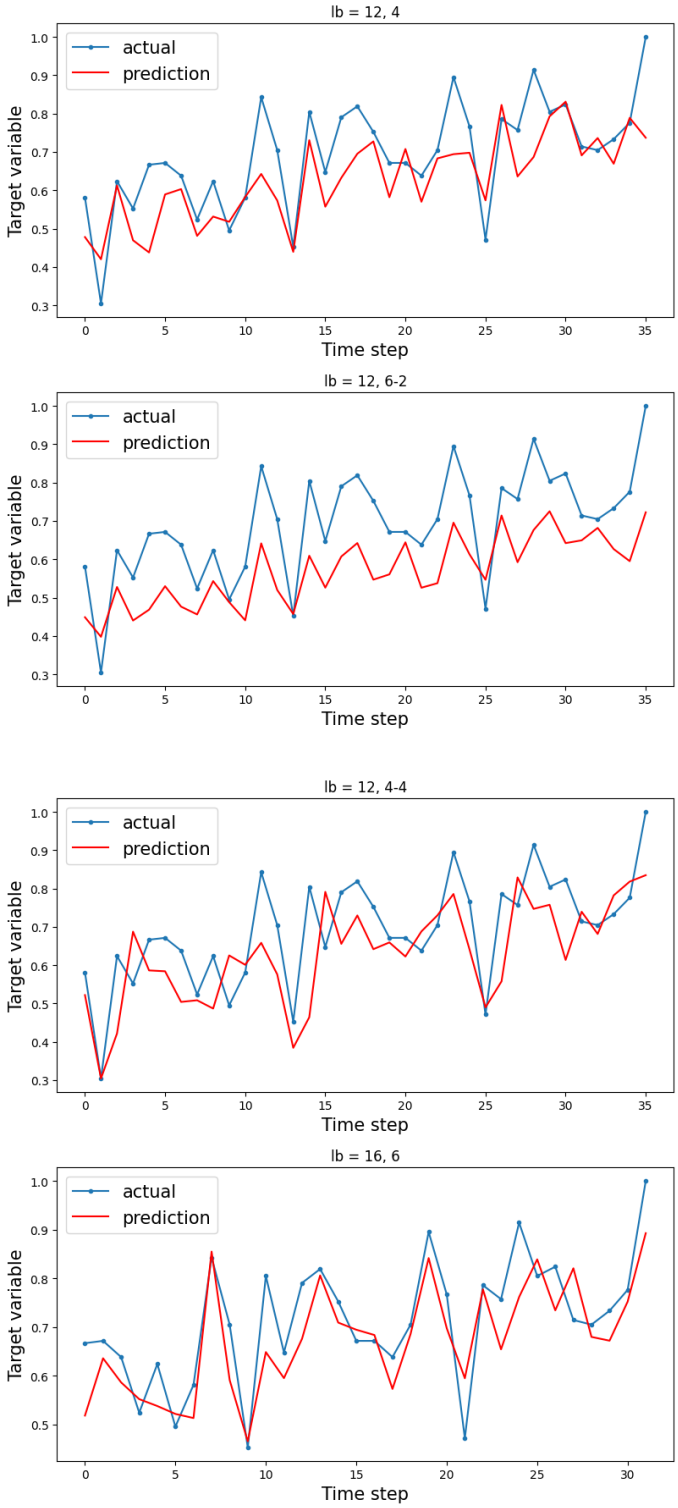


Figure 4.17: results for prediction of consumption of households with different DNN models without auxiliary series

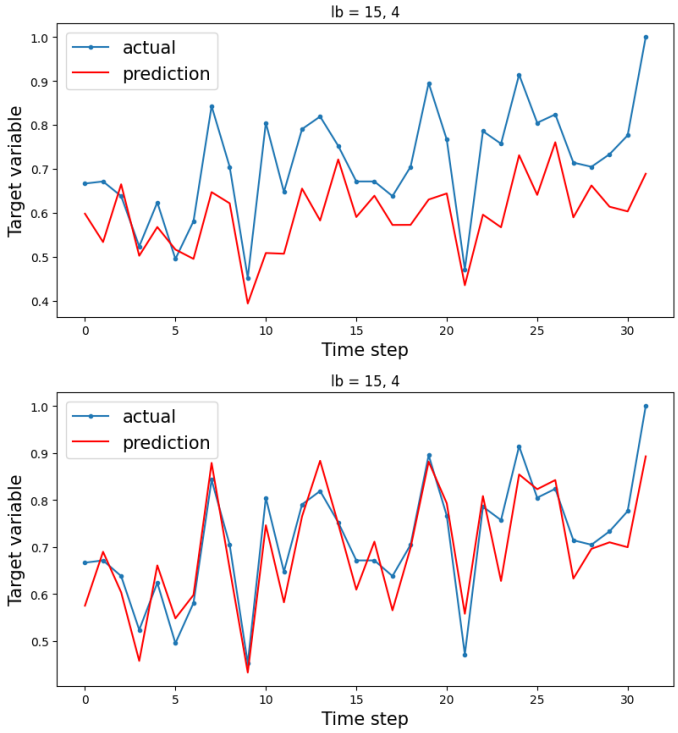


Figure 4.18: results for prediction of consumption of households with different DNN models with auxiliary series (upper figure: Clothes, lower figure: original series with noise).

4.5 Nowcast of GDP

We now move to a more challenging problem, in which we want to nowcast a target variable using several auxiliary variables. In this section we will also consider LSTM models.

Although a large set of auxiliary variables is typically used for nowcasting GDP, e.g. when applying Dynamic Factor models (e.g., Banbura et al., 2010), we restrict ourselves to a smaller set of 9 variables. In this study the objective is to assess the potential value of DNN and LSTM algorithms, and compare with classic time series models. This relatively small subset will streamline the analysis process and avoid potential complications. Dynamic factor models involve more steps to reduce the amount of input data and deal with mixed frequencies. Typically, also, the series are seasonally and calendar adjusted. In our case all auxiliary variables are published figures with no further preprocessing, facilitating a direct comparison with traditional time series models. Most of these variables are published at a quarterly level already, a couple of them (consumer confidence and unemployment) were aggregated to a quarterly frequency. All variables are expected to be highly correlated with GDP and are released earlier than GDP itself, making them suitable for including them in a nowcasting model. This setup will enable us to circumvent issues associated with mixed frequency and ragged edges, contributing to the comparison of the models under consideration.

For our neural network models, we use the first 4 lags as input variables. In addition, we use auxiliary data from 9 indicators: Number of businesses, number

of persons self-employed, labour participation, energy consumption, consumer confidence, services price index, hospitality turnover, government expenditure, job vacancies. These series have a starting date between 1995Q1 and 2007Q1. Since our models require all variables to be available, this means the starting data for our models is 2007Q1.

The same neural network as above was found to be suitable for this dataset, i.e. an DNN(16,8,3,1) network. For this dataset we now compare this DNN model with a network with 2 hidden LSTM layers and one output layer. After normalising the data, the networks were trained with the following parameters, which were established after some manual optimisation: learning rate 0.001, epochs=1000, batch size=4 or 8, training set: 45 periods, test set: 8 periods (for series up to 2020Q1, see below) or training set: 41 periods, test set: 16 periods (for series up to 2021Q1, see below). We also applied early stopping in this case, with 'patience' parameter set to 200 epochs. This is more than usual, but in this application did not cost a lot of computation time, and in some cases this improved the results. In order to improve the smoothness of the learning rate, we optimised the batch size and at the same time reduced the learning rate (while increasing the number of epochs). Since the test set is relatively small, the learning curve for the test set showed some fluctuations. Increasing the size of the test set to 16 periods (see below) improved this and gave more stable results.
We first consider the target series up to the covid crisis, which clearly affected many macroeconomic statistics. We therefore take a subsample of our dataset up to 2020Q1, where we used the first 45 periods for training and the last 8 periods as a test set. As we can see in Figure 4.19, the algorithm picks up the pattern in the training set, and manages to extrapolate this to the test set.

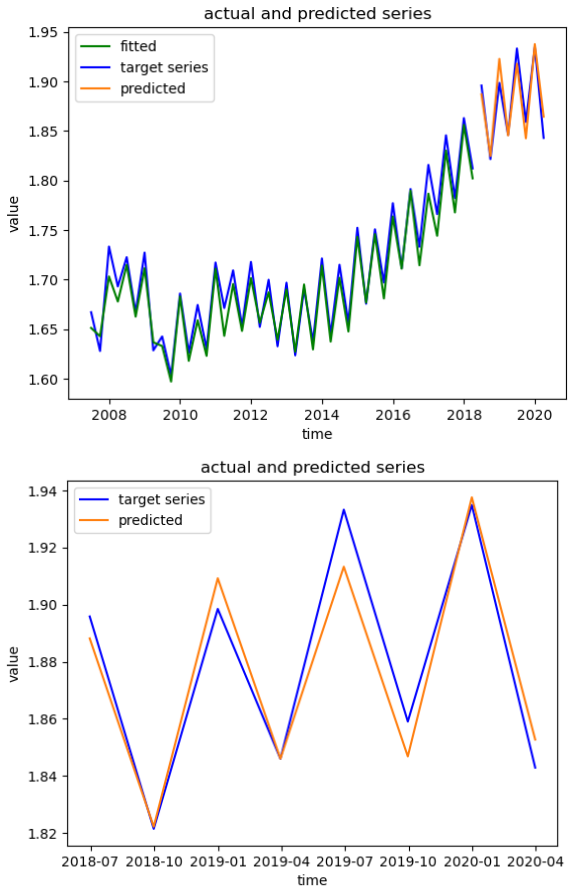


Figure 4.19: fitted and predicted values for an DNN(16,8,3,1) network with a test set of 8 periods.

Since the gradient descent algorithm has random starting weights, we repeat the optimisation procedure 25 times, in order to get an indication of model stability. As shown in Figure 4.20, model predictions for the test set did not differ much under the given settings when we repeat estimation.

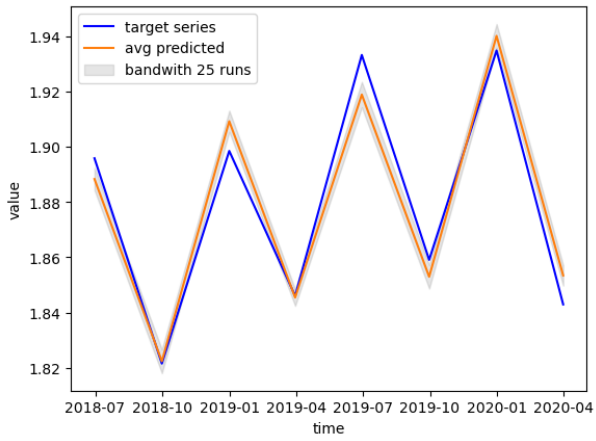


Figure 4.20: predictions for the test set based on 25 runs

After calculating predictions with classic time series models as well as an LSTM model, we see that all methods manage to pick up the seasonal pattern in a similar way if we look at the graph below. When comparing accuracy measures, however, we see that the DNN and the STM with smooth trend plus seasonal have the best accuracy measures. The DNN also performs a bit better than the LSTM, which should be more suitable for time series data. However, as with the airline series, we did not optimise all model settings, and the test set is only 8 periods. Based on this small experiment, we cannot clearly favour one method over the other.

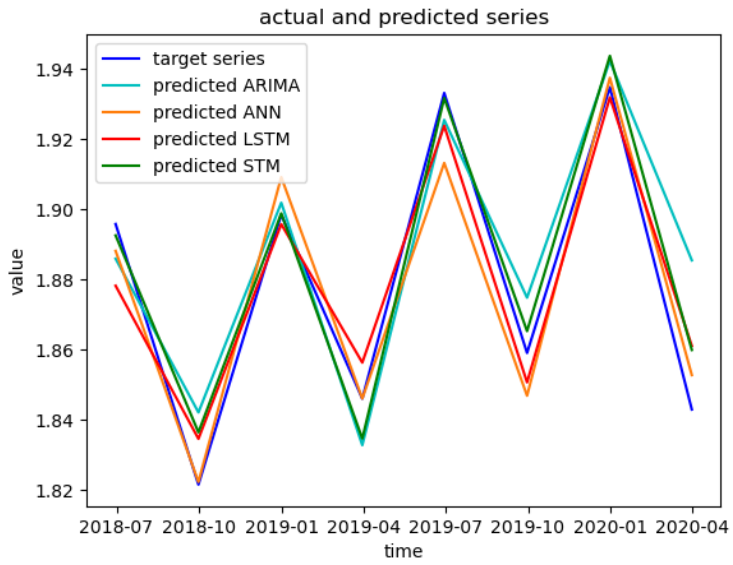


Figure 4.21: one step ahead nowcasts for DNN, LSTM, ARIMA and STM models.

Table 4.6: Accuracy measures for the test set of 8 periods

	RMSE	MAPE	ME
ARIMA(0,1,1)(0,1,1)	0.0188	0.81	-0.0069
STM (smooth trend + seasonal)	0.0098	0.43	0.0039
DNN(16,8,3,1)	0.0097	0.43	-0.0007
LSTM	0.0117	0.55	-0.0001

In order see the potential differences of the methods a bit better, we now include the observations that were affected by the covid pandemic, and increase the test set to 16 periods. Our dataset now runs from 2007Q1 to 2021Q1. Since it is not possible to predict the downturn in GDP, as a result of Covid, based on historical data alone, we should now see the effect of adding the auxiliary data. Only if the auxiliary series contain information on such a significant decline, we can expect a nowcast to show this effect as well.

We perform a cross validation over the test set, where we gradually expand the dataset, and the models are re-estimated every time as new data is added to the series. Both the DNN and LSTM network give accurate predictions in the period before the pandemic. Here, adequately modelling the historical time series pattern is perhaps enough to obtain a reasonably accurate nowcast. Once the crisis has started, the DNN fails to give accurate nowcasts, and the LSTM is significantly

better. This also becomes apparent from the accuracy measures (Table 4.7), which clearly favour the LSTM.

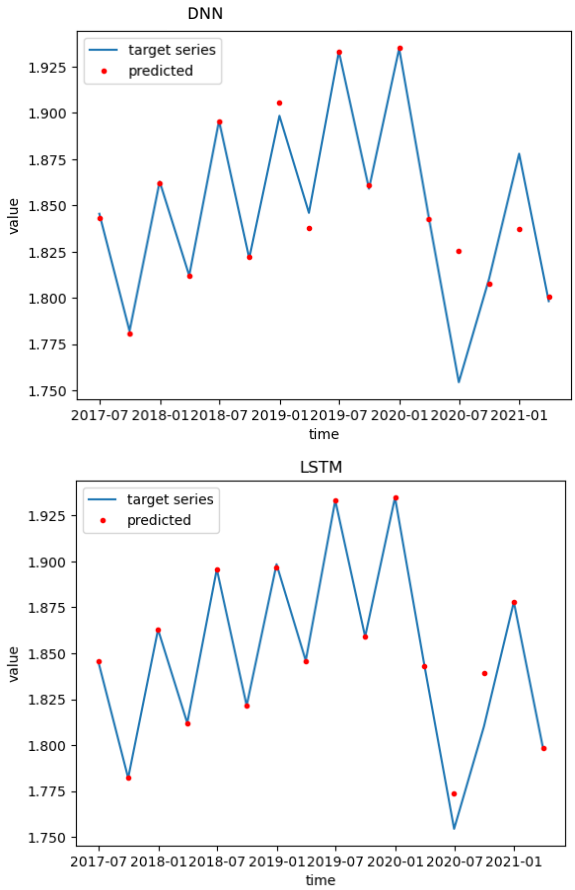


Figure 4.22: Cross validation of 16 one-step ahead nowcasts for DNN and LSTM models

Table 4.7: Accuracy measures for a cross validation over a test set of 16 periods

	RMSE	MAPE	ME
DNN(16,8,3,1)	0.0207	0.49	0.0016
LSTM	0.0087	0.18	0.0029

Since, when GDP is published, most attention goes to the growth rate, we repeat estimation for the GDP quarter on quarter growth rate. All models and settings are kept the same, except for the learning rate, which is now a bit smaller because of the scale of the data. Again, we see that the DNN is not always accurate, and the LSTM is very close to actual published figures (Figure 4.23). This is confirmed by the accuracy measures (Table 4.8), which are better for the LSTM model than for DNN.

DNN

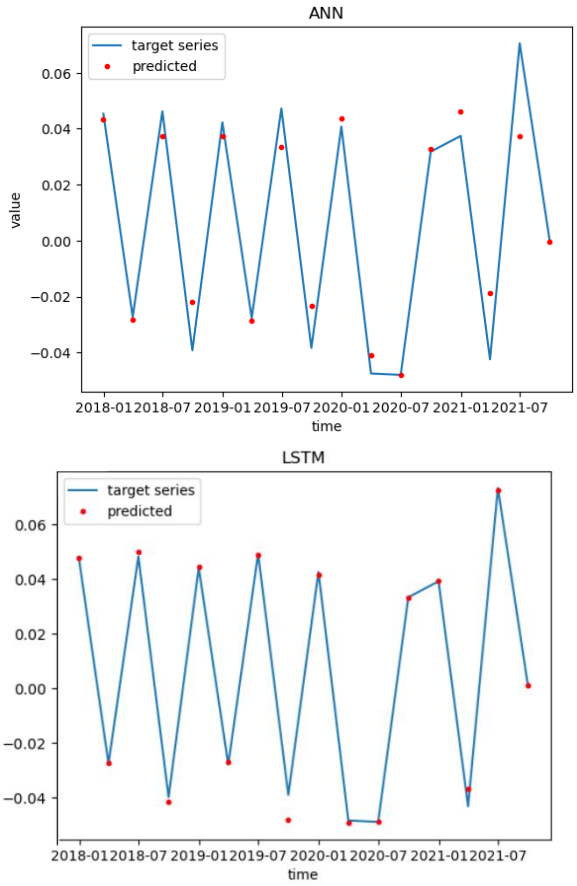


Figure 4.23: Cross validation of 16 one-step ahead nowcasts for DNN and LSTM models

Table 4.8: Accuracy measures for a cross validation over a test set of 16 periods

	RMSE	MAPE	ME	MAE
DNN(16,8,3,1)	0.0132	0.2029	-0.0007	0.0093
LSTM	0.0120	0.1402	-0.0008	0.0065

5. Conclusion

Deep learning consists of powerful algorithms and could be interesting in many fields within official statistics. In this exploratory study we investigated the potential added value of deep learning algorithms when applied in times series analysis. The main focus is the computation of forecasts and nowcasts. With some of the considered methods times series can also be decomposed into components like trend and seasonal.

In our experiments, we used the raw time series as they are published, and did not do a lot of pretreatment of the series as is usually done in time series analysis. We chose to do this in order to see if the neural network algorithms can discover patterns for themselves without leaning on classic time series methods. We saw however, that preprocessing the data can have an impact on the estimates, and in order to benefit from the full potential of ANN's, this is an important step. Although the ANN's studied here managed to discover the most important patterns in the series, many ANN's are not especially designed to model time series. A related issue is the choice of the network design, i.e. the number of hidden layers and the neurons in each layer. There is no direct connection between a classic time series model specification and an equivalent neural network shape. The optimal shape is usually determined by trial and error. A more complex dataset usually requires a more complex neural network, with a larger depth or more nodes per layer. There are no common rules to determine the optimal shape of the neural network, but there are some guidelines to these choices and when gradually optimising the network a lot can be derived from the learning curve (the loss reached after a certain number of epochs). After some trial and error, we found networks that gave acceptable results. A last point concerning the application of the algorithms is their stochastic nature. The algorithms applied use random initialisation of the weights. This means that depending on where the optimisation starts, the estimation routine may not converge or we may reach a local optimum. If the algorithms give a different result each time we run it, this could be an indication of a misspecified model, that the algorithm is not properly parametrised, or needs more training examples.

Clearly, estimation is different from classic time series models. The size of the training set is usually fixed in time series applications, i.e. it is not possible to obtain more training examples. However, there exist techniques to augment the training data that we did not consider yet. Another solution to this problem is transfer learning. The idea is that a model is pretrained with other series, which are somehow similar to the target series. This way, the total training set is larger, which should improve the accuracy of the models. The added value of these techniques for time series applications needs more research. In this paper we restricted ourselves to a quite small set of relatively simple deep learning methods. In the next steps of the project, more methods should be included. Some of the methods are mentioned in Section 2, other interesting methods could be convolutional neural networks of transformers. Another topic in the continuation of the project could to further improve the learning process, and focus on aspects such as regularization, the choice of hyperparameters like the learning rate and batch size, methods for the optimization procedure, and the choice of the random starting values for the weights.

An important characteristic with the algorithms studied in this paper, is the explanatory value they offer and the level of insight they give in how the predictions are influenced by different factors. For example, although we saw that an ANN can adequately capture a seasonal pattern in a time series and make accurate predictions, the seasonal pattern is not estimated explicitly, and we thus cannot study the seasonal component directly. This also applies to other time

series components. Some of the methods considered in this paper are able to decompose time series into their components. Since the main focus of this paper was forecasting, this was not investigated in detail.

The main conclusion is that neural network algorithms can produce similar results (i.e., accuracy of predictions/nowcasts) if applied to univariate time series. In order to achieve this, some effort is required in optimising parameters and further settings of the algorithms, perhaps more than with classic methods. When applied to a more challenging problem with several auxiliary variables and a more volatile series, in our case the LSTM model, which should be especially suitable for time series data, gave accurate results. This leads to the conclusion that deep learning can be perhaps offer added value compared to classical methods for specific problems. Some potential applications may be the following:

- Applications with large datasets. Deep learning algorithms are known to be powerful techniques for large datasets. In time series analysis, we usually only have a limited amount of data per series. However, if applied to time series problems with a large amount of training data available, there may be added value to classic techniques. Possible applications are problems with long time series or a large number of time series, such as in (GDP) nowcasting as an alternative for Dynamic Factor Models (Hopp, 2022a, 2022b). Another possible application would be a situation where we have many subaggregate series available in order to nowcast the aggregate series, such as in business statistics.
- Hybrid ML methods instead of pure ML. As we showed in the empirical study, preprocessing is very important for ML algorithms. We could design a hybrid method where some steps are done by classic methods (such as seasonal adjustment, outlier removal, variable selection) and then combined with ML techniques to make nowcasts.

6. Literature

Anders, U., O. Korn (1999), Model selection in neural networks, Neural Networks, Volume 12, Issue 2, 309-323,

Banbura, M., D. Giannone, and L. Reichlin. 2010. Nowcasting. ECB Working Paper 1275. DOI: http://dx.doi.org/10.2139/ssrn.1717887.

Box, G. E. P. and Jenkins, G. M. (1976). Time Series Analysis: Forecasting and Control (revised edition), Holden Day, San Francisco.

Box, G. E. P., Jenkins, G. M. and Reinsel, G. C. (1994) Time Series Analysis, Forecasting and Control, 3rd edn. Englewood Cliffs: Prentice Hall.

Brownlee, J. (2020), Deep Learning for Time Series Forecasting: Predict the Future with MLPs, CNNs and LSTMs in Python, Machine learning mastery.

Van den Brakel, J.A. and S. Krieg (2009). Estimation of the Monthly Unemployment Rate through Structural Time Series Modelling in Rotating Panel Design. Survey Methodology, 35, 177-190.

Van den Brakel, J.A., S. Krieg and M. Smeets (2021). Estimating monthly indicators for Consumer Confidence using Structural Time Series Models. Discussion paper, Statistics Netherlands. https://www.cbs.nl/en-gb/background/2021/46/estimating-consumer-confidence-using-time-series-models

Durbin, J. and S. J. Koopman (2012). Time Series Analysis by State Space methods, Oxford University Press.

European Union and the United Nations (2017), Handbook on Rapid Estimates

Gedon, D., N. Wahlström, T.B. Schön and L. Ljung (2021). Deep State Space Models for Nonlinear System Identification. IFAC PapersOnLine 54-7, 481 – 486.

Eurostat (2024), JDemetra+, https://github.com/jdemetra

Harvey (1989), Forecasting, structural time series models and the Kalman filter, Cambridge University Press.

Helske, J. (2017), KFAS: Exponential Family State Space Models in R. Journal of Statistical Software, 78(10), 1-39, doi:10.18637/jss.v078.i10

Hochreiter, S., J. Schmidhuber (1997). Long Short-Term Memory. *Neural Comput*, 9 (8), 1735–1780. doi: https://doi.org/10.1162/neco.1997.9.8.1735

Hopp, D. (2022a), Benchmarking Econometric and Machine Learning Methodologies in Nowcasting, UNCTAD Research Paper No. 83. Available at: https://unctad.org/system/files/official-document/ser-rp-2022d3_en.pdf

Hopp, D. (2022b), Economic Nowcasting with Long Short-Term Memory Artificial Neural Networks (LSTM), Journal of Official Statistics, Vol. 38, No. 3, 2022, 847–873, http://dx.doi.org/10.2478/JOS-2022-0037

Hutchinson, G. E. (1978), An introduction to population ecology. Yale University Press. New haven and London.

James, G., D. Witten, T. Hastie, R. Tibshirani, J. Taylor (2023), An introduction to statistical learning: with applications in Python, Springer Texts in Statistics.

Kafritsas, N. (2021). The Best Deep Learning Models for Time Series Forecasting. Internet tutorial on Towards Data Science, https://towardsdatascience.com/the-best-deep-learning-models-for-time-series-forecasting-690767bc63f0

Makridakis, S., E. Spiliotis, V. Assimakopoulos (2020), The M4 Competition: 100,000 time series and 61 forecasting methods, International Journal of Forecasting, Volume 36, Issue 1, 54-74.

Makridakis, S., E. Spiliotis, V. Assimakopoulos (2022), M5 accuracy competition: Results, findings, and conclusions, International Journal of Forecasting, Volume 38, Issue 4, 1346-1364.

Oreshkin, B., D. Carpov, N. Chapados and Y. Bengio (2020). N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. Published as conference paper at ICLR 2020.

Peixeiro, M. (2022). The easiest Way to Forecase Time Series Using NBEATS. Internet tutorial. https://towardsdatascience.com/the-easiest-way-to-forecast-time-series-using-n-beats-d778fcc2ba60

R Core Team (2018). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rao, J.N.K. and I. Molina (2015). Small Area Estimation, 2nd edition. Wiley.
Statistics Netherlands (2024). Business cycle dashboard. https://www.cbs.nl/en-gb/visualisations/economy-dashboard/business-cycle/business-cycle

Taylor and Letham (2017). " Forecasting at Scale," The American Statistician, Taylor & Francis Journals, vol. 72(1), 37-45

Triebe, O., Hewamalage, H., Pilyugina, P., Laptev, N. P., Bergmeir, C. and Rajagopal, R. (2021). NeuralProphet. Explainable Forecasting at Scale, https://api.semanticscholar.org/CorpusID:244729652

Yang, L. (2020). A Quick Deep Learning Recipe: Time Series Forecasting with Keras in Python. Internet tutorial on Towards Data Science, https://towardsdatascience.com/a-quick-deep-learning-recipe-time-series-forecasting-with-keras-in-python-f759923ba64

Zult, D., Krieg, S., Schouten, B., Ouwehand, P. and van den Brakel, J.. (2023). "From Quarterly to Monthly Turnover Figures Using Nowcasting Methods" Journal of Official Statistics, vol.39, no.2, 3923, 253-273. https://doi.org/10.2478/jos-2023-0012

7. Appendix: results for artificial series with RNN and LSTM

Early stopping is used in all cases. The number of epochs before the training stops vary between a few dozen and almost a thousand. The learning rate of 0.0005 is used for the RNN models and 0.001 for the LSTM models.

Table A.1 shows the results for the RNN-model. The table is similar as Table 4.1 in Section 4. Preliminary attempts showed that RNN-models are more difficult to train, therefore we restrict ourselves to the series with the large seasonal pattern and without seasonal pattern. The model specification with two hidden layers with 10 units (RNN layer) and 3 units (Dense layer) is very difficult to train. In most attempts, a straight line is found, similar as in Figure 4.2 above. For the example with length of the training series of 100 and large seasonal, we found (after many attempts) another solution which is shown in the table.

There might be better specifications of this model, but all together, this model does not look promising, compared to the DNN-model discussed before.

Table A.1: Results for RNN with artificial series

Series	Length training series	Lags	units	RMSE
Large seasonal	1000	8	10-3	0.017
Large seasonal	100	16	16-8	0.021
Large seasonal	100	8	10-3	0.064
No seasonal	1000	2	3	0.0015
No seasonal	1000	8	10-3	Straight line
No seasonal	1000	16	16-8	0.0071
No seasonal	100	2	3	0.016
No seasonal	100	8	10-3	Straight line
No seasonal	100	16	16-8	0.012

Table A.2 shows the results for the LSTM model. When the series with large seasonal are predicted, a model with 10 units and l 16 lags seems to be preferred above a model with only 6 units and 8 lags. This is despite the fact that the model with 10 units and 16 lags is very large, especially when the training series has a length of 100. This is remarkable, as in this case with a short series and a large model the risk of overfitting is quite large. As always, we cannot be sure whether this result is caused by the local optima which are coincidentally found.

With the series with a small seasonal pattern, the smaller model seems to be preferable, especially when the training series is short.

For the series without seasonal pattern, all models are far less accurate than the model which uses the value from $t-1$ as prediction for t. For the short training series, a very parsimonious model seems to be preferable.

Table A.2: Results for LSTM with artificial series

Series	Length training series	Lags	units	RMSE
Large seasonal	1000	8	6	0.016
Large seasonal	1000	16	10	0.009
Large seasonal	100	8	6	0.095
Large seasonal	100	16	10	0.019
Large seasonal	100	16	16	0.033
Small seasonal	1000	8	6	0.015
Small seasonal	1000	16	10	0.017
Small seasonal	100	8	6	0.040
Small seasonal	100	16	10	0.076
Small seasonal	100	16	16	0.077
No seasonal	1000	8	6	0.040
No seasonal	1000	16	10	0.016
No seasonal	100	8	6	0.055
No seasonal	100	16	10	0.066
No seasonal	100	2	1	0.039
No seasonal	100	4	6	0.049

Explanation of symbols

Empty cell	Figure not applicable
.	Figure is unknown, insufficiently reliable or confidential
*	Provisional figure
**	Revised provisional figure
2017–2018	2017 to 2018 inclusive
2017/2018	Average for 2017 to 2018 inclusive
2017/’18	Crop year, financial year, school year, etc., beginning in 2017 and ending in 2018
2013/’14–2017/’18	Crop year, financial year, etc., 2015/’16 to 2017/’18 inclusive

Due to rounding, some totals may not correspond to the sum of the separate figures.

Colophon

Publisher
Centraal Bureau voor de Statistiek
Henri Faasdreef 312, 2492 JP Den Haag
www.cbs.nl

Prepress
Statistics Netherlands, CCN Creation and visualisation

Design
Edenspiekermann

Information
Telephone +31 88 570 70 70, fax +31 70 337 59 94
Via contactform: www.cbs.nl/information