Gas Flow Prediction using Long Short-Term Memory Networks

Regression analysis of gas flow time series in a nationwide gas transport system using LSTM networks

Maikel Withagen
s1867733
1 December 2017

Master's Thesis

Department of Artificial Intelligence
University of Groningen
The Netherlands

Internal Supervisor
Prof. dr. L. R. B. Schomaker

External Supervisor
ir. S. Aaftink

Abstract

The organization of natural gas flows through an international gas transport network is a very complex and abstract process. Due to the slow, flowing aspect of compressed gas, operations on the network's organization take some time to show their effects, and an intricate balance has to be struck between multiple factors. The aim of this study, was to find if LSTM networks were able to model such time series, and can therefore be used as predictive models for the gas flows in the network. The LSTM networks were able to sufficiently model all the possible components of a time series, if certain conditions were met. When modelling the national usage-based network, LSTM networks were able to achieve better results than the default time series regression technique ARIMA, and the heuristic currently used by the Gasunie. An important factor in achieving these results, was the use of an autoencoder layer, which allowed us to input the intrinsic network state to the LSTM model, while negating the naivety problem. A model of the complete international model gave imperfect results however, where the non-existent influence of model parameters on the performance indicated that the network most likely was lacking information. Further research is recommended to extend the model with information-rich features such as international natural gas prices on the stock markets, or expanding the complexity of the network's topology.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Gasunie is a European gas infrastructure company that provides the transport of natural gas in the Netherlands and the northern part of Germany. Operational decisions are made by so-called dispatchers, who use a variety of network information combined with their expert knowledge to optimize the flow of gas through the network, in order to meet everyone's needs. Regulation of this gas transport system consists, amongst other things, of properly organizing and distributing the supply and demand orders of several shippers over the network. The shippers are gas transporting or trading parties in the gas economy, transporting and/or trading large volumes of natural gas internationally. A number of these shippers have to acquire natural gas in order to supply their customers, for example the inhabitants of their region. This allows people to heat their homes, cook their food, etc. using natural gas. There also exist a number of shippers who have no desire to put their acquired gas to practical use, but are purely trading for profit. They do however, need transport of their orders.

The gas transport network can therefore be viewed as two-sided. On one hand, the International part of it has to accommodate large volumes of gas flowing through the network, with in- and outputs at interconnection points to foreign gas transport networks, placed along the border. On the other hand there is a National part, which has to ensure that all the local gas distribution networks receive their share of gas, so that households and local industry can continue to function.

The Gasunie itself does not occupy itself with any sort of trading, but is purely a market facilitating party. A useful piece of information in the operating process is an accurate prediction of the expected gas flows in the network. Dispatchers have to ensure certain levels of pressure, gas quality, and flow in the network to keep it operating, and can alter these values with the use of, for example, compressing stations, mixing points and more general with the opening and closing of pipelines. The gas transport network itself is a very dynamic system, where operating decisions do not yield immediate results, but influence the network over time. Increasing the pressure in a pipeline to increase flow does not happen instantaneously, but takes time to build up. The effects of such an action can take hours to propagate through the network, and actually have an effect in the desired part of the system. Next to the temporal delay of actions in the network, one can imagine that starting up a large compressor unit, or adjusting a gas mixing facility does not happen in an instant. Proper predictions of gas flow can therefore allow dispatcher to proac-

tively manage the network, and possibly increase the network efficiency in terms of operating costs, compared to a more reactive managing position. Each of the compression/mixing stations has its own optimal curve in terms of usage, load and cost efficiency. By being able to accurately predict the gas flow for the next few hours, the use of these subsystems could be optimized. An accurate prediction of shipper behaviour also allows dispatchers to deal with possible near-future problems.

People have a natural tendency to heat their homes when it's cold, and stop heating when it is warm enough. The majority of Dutch households has a connection to a regional gas network, and about 75% of their gas usage is used for heating. The gas flow needs of the national market is therefore mostly influenced by current meteorological variables such as temperature, cloud coverage, season of the year and so on. Other parties such as local industries also have their influence on the national part of the gas network, but are usually more constant. On the other hand, predicting shippers' behaviour requires not only knowledge of the meteorological variables of that day, but also knowledge about the shippers' earlier behaviour. The flexible organization of the gas network allows for shippers to compensate for possible errors made in previous days or adjusting their behaviour to decisions made by other shippers. Furthermore, as shippers' behaviour is very much money-driven, the price of natural gas on the stock markets also has an influence on shippers' operating methods and therefore their shipping orders.

Modelling this type of behaviour requires a system that can include decisions made in the past in its predictions. A standard approach to such time-dependent problems is the use of statistical methods such as Autoregressive Integrated Moving Average (ARIMA)[1]. However, as the names suggests, ARIMA uses a moving average to interpret time-dependent sequential data, and is therefore unsuited to model trends such as seasonality. Extensions exist for this problem, such as the seasonal ARIMA which has added input values for seasonality terms, but using them requires additional knowledge about the input data. Long Short-Term Memory networks (LSTMs) are proven to be effective in modelling and predicting data in time-driven or sequential data, and could therefore be suitable for this problem. Therefore, an advantage of LSTMs in this case (and in general) is the fact that they can handle trends/seasonality by design.

In the first part of this thesis, the theoretical background for the used neural network algorithms, recurrent neural networks and LSTMs in particular, is discussed. Following the theoretical background, the intrinsic modelling capabilities of LSTM will be researched to judge if and how they can be used for regression of time series. Subsequently, findings from the first part will be used to design a general LSTM regression model that is suitable for the prediction of gas flow in the national gas transport network of the Gasunie. This model will be developed in two use cases; on national meteorological-driven network points known as RNBs; and on the more behaviour driven International part of the gas transport network.

Chapter 2

Neural Networks

This chapter gives the theoretical background to recurrent neural networks. In particular, attention is given to the Long Short-Term Memory networks, which are used extensively in this thesis.

2.1 Perceptrons

Artificial neural networks were originally inspired by how higher organisms were thought to be able to contain and remember information. The discovery that information is stored in connections or associations, i.e. it is contained in the *preference for a particular response*, led to the development of a basal building block of most the neural networks today; the perceptron[2].

A perceptron maps several binary inputs $[x_1, x_2, \ldots, x_i]$, or an input vector $\mathbf{x}$ to a single binary output $f(x)$ (see Figure 2.1). $f(x)$ is computed by calculating the dot product between the input vector $\mathbf{x}$ and the corresponding weights vector $\mathbf{w}$ and comparing it to a threshold value. If the outcome of the dot product exceeds the threshold value, the perceptron wil output '1', otherwise it will output '0' (Figure 2.1a). Because the input vector is binary, the information, or knowledge in this system is stored in the weights vector. The importance of the presence (or absence) of an input node can be adjusted by varying its corresponding weight-value. The perceptron is capable of learning linearly separable patterns, and can approach every possible step function[3].

Nowadays, the binary input restriction of the perceptron has been removed, allowing for more subtle modelling. Also, the threshold has been implemented as an additional input with value -1 and a variable weight, turning it into a so-called bias. Its functionality remains the same, as a threshold of 0 still has to be met, but by implementing it as another neuron, calculations are made easier. To allow the network to compute continuous an possible non-linear output, the binary output is usually replaced with a so called activation function (Figure 2.1c). A common activation function is the sigmoid function (Figure 2.1d), which makes this single-layer perceptron model identical to the logistic regression model.

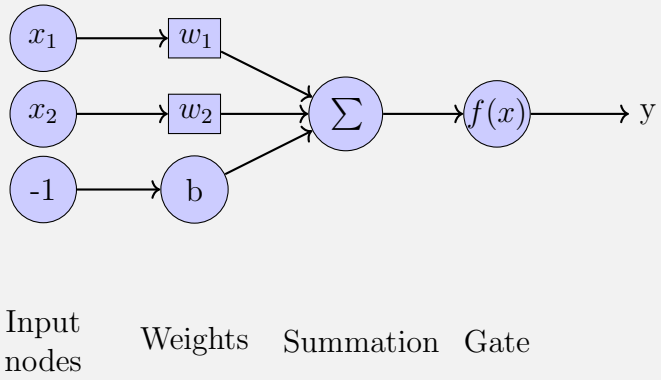
For the training of a perceptron, a set of labelled training data is needed.
Let D be this training set, where $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$, n is the number of training samples, where each sample consists of an input vector $\mathbf{x}$ and corresponding label y. The model is trained by adjusting its weights in order to minimize the error of its output, given by a so-called *cost* or *loss function*, which

Figure 2.1: A Standard perceptron with three input nodes

2.1a: The original gate formula

$$f(x) = \begin{cases} 0 & \textbf{if } \sum x_i w_i \leq \textbf{Threshold value} \\ 1 & \textbf{if } \sum x_i w_i > \textbf{Threshold value} \end{cases} \tag{2.1}$$

2.1b: A graphical representation of the perceptron



2.1c: The Updated gate formula where h(x) is an activation function

$$f(x) = \begin{cases} 0 & \textbf{if } x \cdot w + b \leq 0 \\ g(f(x)) & \textbf{if } x \cdot w + b > 0 \end{cases} \tag{2.2}$$

2.1d: Logistic activation function

$$g(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

can be as simple as the difference between the targeted output and the actual output.

2.1.1 Delta rule

For single-layer neural networks, the weight adjustments for each network connection are given by the delta rule(Equation 2.4)

The delta rule is designed to minimize the error in the network's output through gradient descent. Gradient descent is a first-order iterative optimization algorithm, which is designed to find a local minimum in the error space. The algorithm takes

Figure 2.2: Delta Rule

$$\Delta w_{ij} = \alpha(t_j - y_j)g'(h_j)x_i \tag{2.4}$$
$$h(j) = \Sigma x_i w_{ij} \tag{2.5}$$

where:

Δw_{ij}: the weight adjustment of weight j of node i

$g(x)$: neuron's activation function

$h(j)$: the weighted sum of j's inputs

α: the learning rate

t_j: the target output

y_j: the actual output

x_i: the i^{th} input

proportional steps towards the negative of the gradient as shown in Equation 2.4, i.e. it assigns a value to the influence of a single neuron on the resulting cost function, and alters the neurons weight proportional to its influence, and the resulting output mismatch.

In a perceptron with linear activation, the first order derivative of the activation function becomes a constant, and the delta rule can be simplified to:

$$\Delta w_{ij} = \alpha(t_j - y_j)x_i \tag{2.6}$$

2.2 Multi-Layered Perceptron

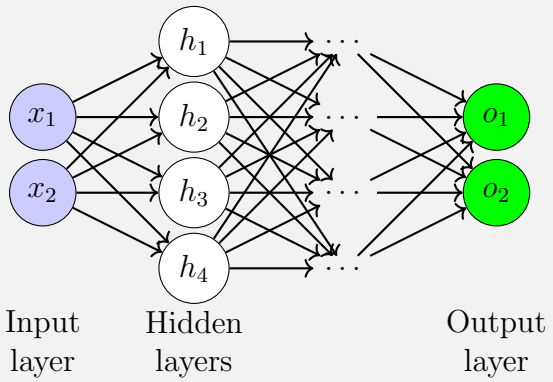
A logical extension of the single-layer neural networks as seen in the previous section, is the addition of more layers. A so-called multi-layered perceptron (MLP) consists of three or more layers of which the first layer consists of input nodes, followed by a number of hidden layers, of which the last layer is known as the output layer (Figure 2.3). Default MLPs are fully connected, meaning that every single node of layer n is connected to all nodes of the previous *n-1* and the next layer *n+1*.

Unlike the simple two-layer perceptron networks (who have been shown incapable of approximating functions outside a select group of functions[3]), multi-layer perceptrons with as few as one hidden layer and using any arbitrary non-linear activation function, are capable of approximating any measurable function with any desired measure of accuracy if provided with sufficiently many hidden units[4].

2.2.1 Backpropagation

A common method of training a multi-layer perceptron, or any type of feed forward neural network, is called backpropagation[5]. This algorithm consists of two consecutive steps that can be repeated indefinitely.

Figure 2.3: A graphical representation of a multi-layer perceptron network



The first step of the algorithm consists of a forward propagation part, where the network is presented with an input vector. This vector is then propagated through the network, until the output layer is reached. The output of the network is evaluated in comparison to the target output, via a *loss-function*, such as the mean squared error (Equation 2.8) .

The resulting error value(s) is/are then propagated backwards through the network, with respect to the weights in the networks. Each neuron now has an associated error value which reflects its contribution to the given output. Along with the error value, the gradient of each neuron with respect to the loss function is calculated. This pair is then fed to an optimization method, which gives a weight change for each neuron trying to minimize the loss function. Using gradient descent, this gives us a weight update rule of:

$$\Delta w_j = -\alpha \frac{\delta E}{\delta w_j} \tag{2.7}$$

Notice that the weight adjustment is the negative of the gradient, hence the name gradient descent.

It can be seen that backpropagation is a generalization of the delta rule, so it can be applied to multi-layered networks. Backpropagation does require the activation function of the individual neurons to be differentiable, in order to calculate gradients for each layer's neurons via the chain-rule. Also, since $\frac{\delta E}{\delta w_j} \neq 0$ for learning to occur, the step-function from the perceptron is now unsuitable.

2.2.2 Backpropagation alternatives

Of course, as with most machine learning techniques, alternatives exist for the default and somewhat primitive standard Gradient Descent Backpropagation (GD). The learning rate parameter has perhaps the most influence on converging on a stable minimum in the error space. Several variations on GD hope to benefit from this influence by using a variable learning rate, or by adding a certain 'momentum' to the weight updates, e.g. by using RMSProp[6]. The weight change is then scaled by the 'velocity' of previous gradients, where subsequent consistent gradients can

Figure 2.4: The Mean Squared Error (MSE)

$$E = \sum_j \frac{1}{2}(t_j - y_j)^2 \tag{2.8}$$

where:

E: is the total network error

t_j: the targeted output for output neuron j

y_j: the actual output for output neuron j

build up momentum and larger weight adjustments are possible. As a bonus, small oscillations in the error space are dampened, which reduces the chance to end up in a local optimum.

Another possible improvement over GD, is the use of algorithms based on the works of Broyden, Fletcher, Goldfarb, and Shanno (BFGS)[7], which are based on the Newton-Raphson method for seeking stationary points of a differentiate function but which don't require intensive calculations[8].

However these methods rely on good starting points and parameter settings to achieve good results. The more random, brute-force approach of GD can be surmounted by increased computational power these days, making it the favoured approach for neural networks, often coupled with a minibatch setup, to somewhat reduce the randomness.

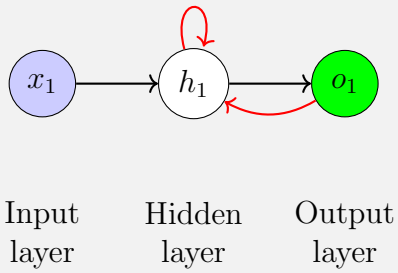
2.3 LSTM

2.3.1 Recurrent networks

In MLPs and other feed forward networks, the assumption is made that each input vector is independent of each other. Every new input generates an output that is based solely on information from the input vector, no knowledge about previously given input vectors or any residual activation of the neurons is saved in the network. This allows for fairly easy computations and regression models, but this assumption cannot be made on the data of a large number of problems. For example, predicting the next most likely occurring word in a sentence requires knowledge about previous words in that sentence. This can be solved by giving the previous words of the sentence, but how much words do you give to the network? Do you give it words of the previous sentence if you are at the start of the sentence? What if you are at the start of a new paragraph? A solution for this problem are recurrent neural networks (RNNs), who have connections between their units to allow information previously put into the network to flow back to the current calculation (Figure 2.5). Due to their recurring connections, RNNs are able to keep track of previous states and implement an internal state. This allows them to process arbitrary sequences of inputs, and makes them applicable for dynamic temporal problems.

A specific type of RNN, that has gained a lot of scientific attention lately is the

Figure 2.5: A graphical representation of a recurrent neural network (recurrent connections in red)



Long Short-Term Memory[9](LSTM) network. The reason for this interest is due to the ability of LSTM networks to remember occurrences over large time lags, without having the vanishing gradient problem.

2.3.2 Vanishing Gradient

The standard training method of recurrent neural networks is an extension of the previously discussed Backpropagation method. This approach, called backpropagation through time[10](BPTT) is able to perform the gradient descent method in a continuous-time network, such as RNNs. A very crude explanation of BPTT is that the recurrent network is unrolled for several time steps (Figure 2.6), after which backpropagation can be applied over the resulting multi-layered network.

Since the error is propagated backwards through a network, the gradient of a cell's weights depends on the gradients of all the connected cells in previous layers (when viewed backwards), as the gradient is calculated via the product rule. Every gradient calculation contains a multiplication with a cell's activation function, which usually is a squashing function such as the logistic function $g(x) = (1 + e^{-x})^{-}1$, or the hyperbolic tangent $g(x) = \tanh(x)$. Therefore, with each consecutive layer, the gradient decreases exponentially. This results in very slow learning front layers, but more importantly, in the case of an unrolled network the influence of the first networks' nodes (Figure 2.6: s_1) on the resulting output error (Figure 2.6: E_3) has vanished. This effectively means that standard RNNs fail to learn with time lags greater than 5–10 discrete steps between the relevant input and corresponding target values[11].

When using activation functions whose derivatives produce large values, an opposite effect can occur, namely that the gradients start to explode, resulting in other unwanted behaviour.

2.3.3 LSTM

An LSTM is a type of recurrent neural network that contains specific neurons called LSTM units. These LSTM units are not affected by the vanishing gradient problem, since by design they apply no activation function within its recurrent components.

Figure 2.6: A graphical representation of a single cell recurrent neural network unrolled over 3 steps

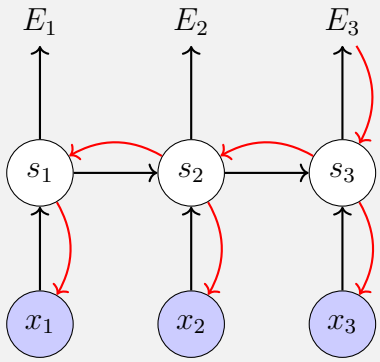
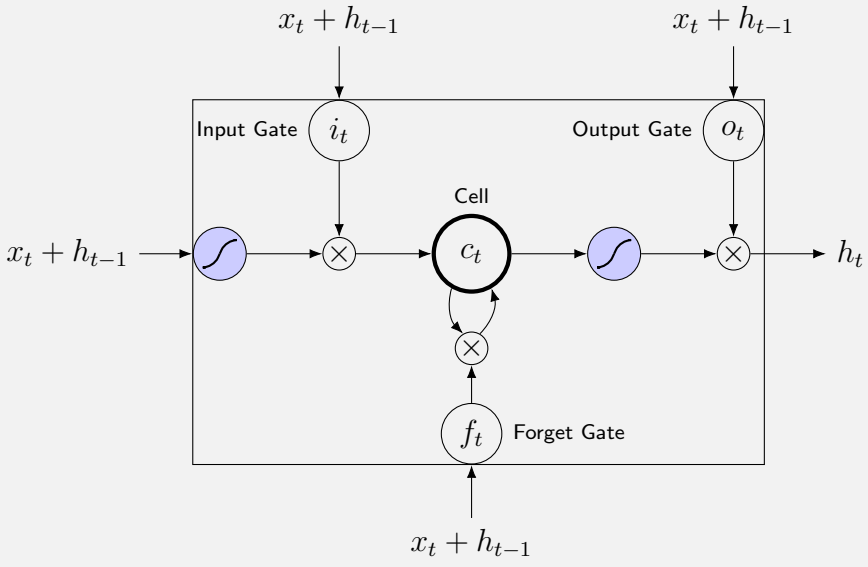


Figure 2.7: A standard LSTM cell



Therefore, the gradient will not vanish nor explode with the use of backpropagation through time. Instead, LSTM cells are capable of remembering gradients over time. This is done by keeping track of an internal cell state, and the flow of information in and out of the cell is regulated with several gated inputs.

A graphic overview of an LSTM cell can be seen in Figure 2.7. The cell state is modified by the three input gates; the *input* gate which regulates the extent to which new data flows into the cell, the *forget* gate which regulates the extent to which the previous state is remembered, and the *output* gate which regulates the extent to which the internal state is used to calculate the cell's output activation. Each of the gates are implemented with the sigmoid function (2.1d), to generate

Figure 2.8: Standard LSTM Equations

$$i_t = \sigma(W_i \cdot [x_t, h_{t-1}]) \tag{2.9}$$
$$f_t = \sigma(W_f \cdot [x_t, h_{t-1}]) \tag{2.10}$$
$$c_t = f_t * c_{t-1} + i_t * \tanh(W_c \cdot [x_t, h_{t-1}]) \tag{2.11}$$
$$o_t = \sigma(W_o \cdot [x_t, h_{t-1}]) \tag{2.12}$$
$$h_t = o_t * \sigma(c_t) \tag{2.13}$$

where:

σ: is the sigmoid function

W: is the matrix corresponding to the relevant gate

a value between 0 and 1. These functions are pointwise multiplied with the input information (and previous activation) to regulate the information flow in the cell. The actual information flowing into the cell is also scaled with an activation function, usually the non-linear hyperbolic tangent function. This results in the standard LSTM equations as shown in Figure 2.8.

2.4 Predicting time series

LSTMs are shown to be able to capture long term time-dependant patterns in time series data. For example, in [12] an LSTM network was trained that was able to differentiate between a neuronal spike pattern of a spike occurring every 49 time steps, versus a pattern that spiked every 50 time steps.

Where normal recurrent networks would have forgotten the occurrence of a spike after about 5 time steps, due to the vanishing gradient, the LSTM was able to retain the spiking information over time, and was able to correctly separate a 49-period from a 50-period spiking neuron.

Chapter 3

Predicting univariate gas flow time series

The global interest of this thesis is the ability of Long Short-Term Memory networks to interpret and predict time series data. In particular, its ability to predict the flows of gas in a nationwide distribution network. The following chapter will give an introduction to the standard analysis methods of time series, and experiments are performed to test if LSTM networks can match the needed efficiency.

3.1 Time series analysis

In order to predict a univariate time series such as sales over time, outside temperature, or gas flows; one should be able to capture all the characteristics of such a series. To further understand these characteristics, a standard univariate time series ($\mathbf{X} = [X_0 \dots X_n]$), can be decomposed in a combination of one or several types of components as shown in Figure 3.1[13].

Many statistical regression methods, such as the AR(I)MA models, try to estimate the trend-cycle and seasonal components of a time series to forecast future values. As the name suggests, the trend-cycle component can be described as the overall trend or long-term movement of a time series. A visualization of such a trend line is shown in Figure 3.2 The blue line displays the original time data series, which is the number of international airline passengers from 1949 to 1960, as used in [14]. The orange line is an approximation of the trend of the data, made by taking

Figure 3.1: Classic Time Series Decomposition

$$X_t = T_t + S_t + Y_t \tag{3.1}$$

where:

T_t: is a slowly changing function, known as a **trend-cycle** component

S_t: is a periodic function, known as a **seasonal** component

Y_t: is a **random noise** component

Figure 3.2: Airline Passenger data

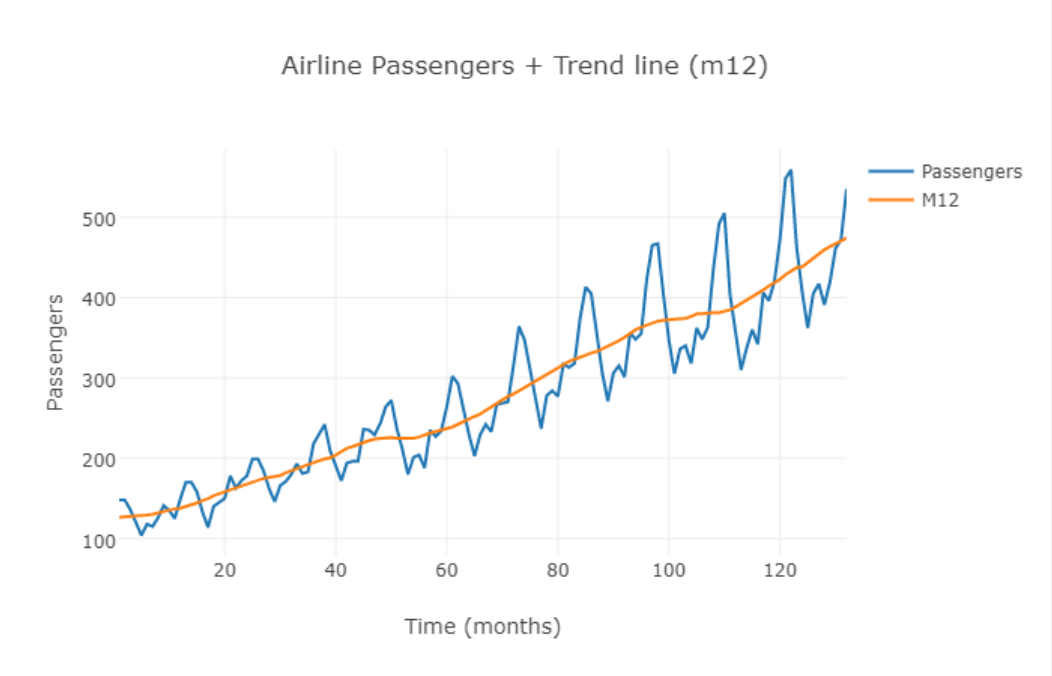
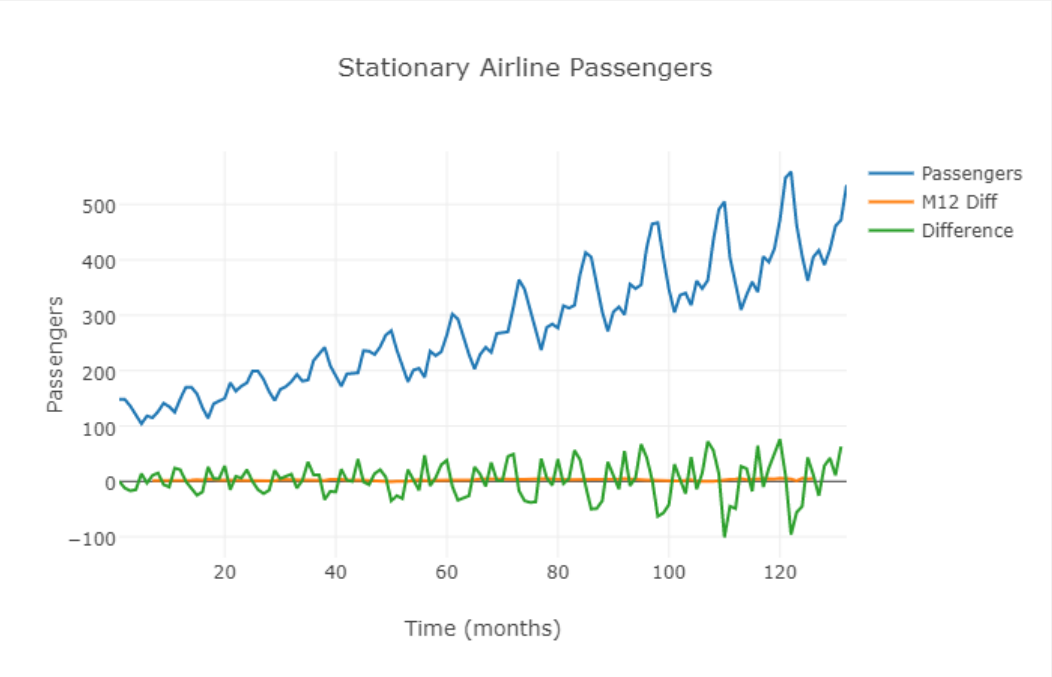


Figure 3.3: Stationary Airline Passenger data



a 12-MA (Moving Average) calculation, meaning that for each twelve subsequent values, the average is calculated. The window size was set at 12 time steps because the time series contains monthly data, and we expect that any reoccurring patterns or seasonal trends occur inside a year. By averaging over 12 months, all the smaller fluctuations inside a year are compressed to their influence on the larger trend.

Time series that do not contain a trend-cycle component are called *stationary time series*. This means that the time series is a process of which the mean and variance do not change over time. Most statistical methods are fitted on a stationary time series, since the only component to model is a seasonal component. A time series can be made stationary by either estimating the existing trend and subtracting it, or by differencing the data. Estimation of the trend can be done by fitting a polynomial to the time series (with a least squares estimator), and subsequently subtracting the fitted function from the time series. Predictions are then constructed by combining the output of the fitted trend function, with the output of a model fitted on the stationary time series.

A less complex solution for removing the trend is differencing the data. Differencing is done by applying a 1-lag difference operator on the data. Consequently, a model fitted on this time series is only able to predict further differences. In order to predict actual regression values, each time step up until the desired t, has to be generated.

An illustration of differencing is shown in Figure 3.3, where the blue line again shows the Airline passenger data. The green line displays the result of applying the 1-lag difference function on the Airline passenger data, and the orange line displays an approximation of any trend on the differenced data, by again applying a 12-MA rolling-mean. The differenced time series clearly no longer displays an ascending trend along the 'Time' axis. This is also illustrated by the rolling mean, 'M12 Diff', which is constant over time. To truly create a stationary time series, the variance also has to remain constant. An easy fix for this dataset is applying the log function on the differenced time series (which is a general trick for creating stationary data sets), but other datasets could require other transformation in order to obtain good results.

The seasonal component in this dataset is illustrated by the regular interval of peaks and troughs over the months. It displays an oscillatory pattern over a certain time, and repeats after each pattern.

Finally, after extracting both the trend-cycle and the seasonal/periodic components from the data, the so-called 'random noise' remains. Looking at the differenced Airline passenger data, it seems that there is a repeating pattern every 10 time steps or so. A closer inspection reveals that these patterns are not all the same, for instance at the start of the dataset the pattern is much less pronounced compared to the tail of the data. These small fluctuations can be appointed to 'random' or unpredictable influences, comparable to statistical residuals.

3.1.1 Gas flow as a univariate time series

By applying the just seen characteristics to a regular gas flow time series, we ascertain that such a gas flow can be analysed as a univariate time series.

The trend-cycle characteristic describes the overall trend of a time series. Regarding a gas flow, this could possibly translate to the number of households/industrial

clients on the network, which can grow or diminish over time. This would cause a gradually increasing of decreasing gas flow over time, and can be considered a trend-cycle component in the time series.

The seasonal component of a gas flow time series is also clearly apparent. If the main usage group of a network point are households, gas usage will spike during morning and evening hours when the people are at home, and will dip during the night or daytime, when people are sleeping and working. Likewise, gas usage also varies during the course of a year, as the average temperature in summer is higher then in winter. This function, with predetermined periodicity can be considered a seasonal component.

The final component category might the most interesting one, the random noise component. Fast fluctuations of the gas flow, such as temperature differences (a hot day in a colder week), or a large increase in gas sales due to a lower gas price on an foreign gas market, cannot be explained with seasonal or trend components. However, this component may not be as random as the name suggests. Since all gas shippers perform their actions with a system of business rules or Business Intelligence (BI), their actions should display some sort of pattern. In the case of a univariate gas flow series, this may be hard to notice, but if, for example, a gas shipper always buys actively at the start of the gas day and sells his surplus at the end of the day, then the co-occurrence of these two actions should be visible in the gas flow. Most statistical methods could have trouble with these kinds of patterns, but an LSTM network should be able to pick up these time-lagged behaviours.ns of the gas flow, such as temperature differences (a hot day in a colder week), or a large increase in gas sales due to a lower gas price on an foreign gas market, cannot be explained with seasonal or trend components. However, this component may not be as random as the name suggests. Since all gas shippers perform their actions with a system of business rules or Business Intelligence (BI), their actions should display some sort of pattern. In the case of a univariate gas flow series, this may be hard to notice, but if, for example, a gas shipper always buys actively at the start of the gas day and sells his surplus at the end of the day, then the co-occurrence of these two actions should be visible in the gas flow. Most statistical methods could have trouble with these kinds of patterns, but an LSTM network should be able to pick up these time-lagged behaviours.

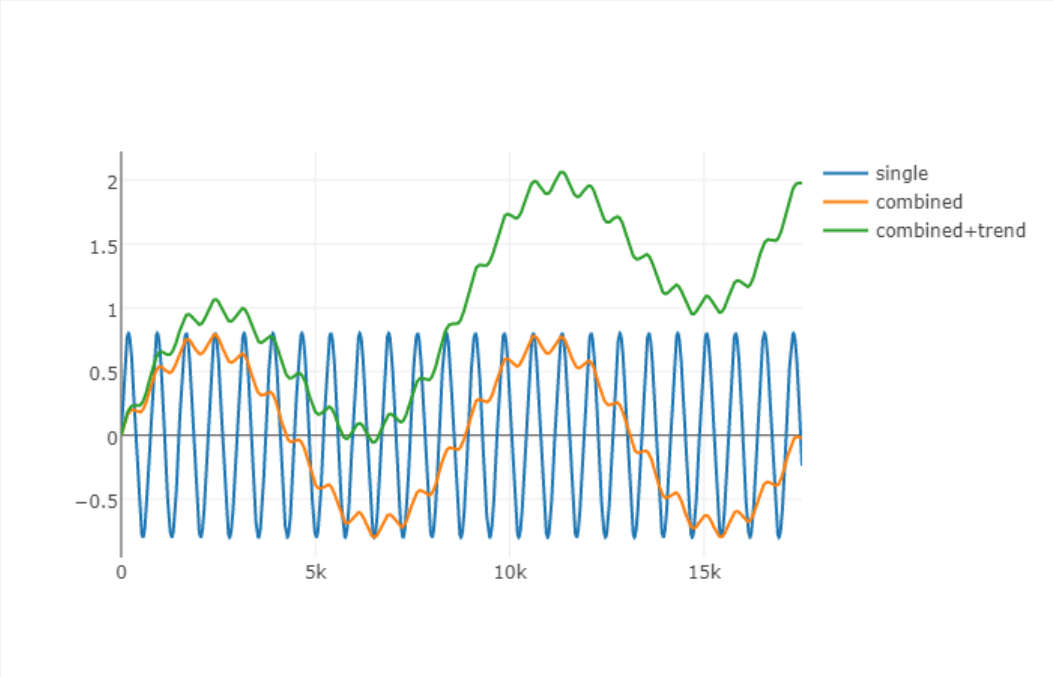
3.2 LSTM on basic time series

3.2.1 Learning long-term sine predictions

Previous research has shown that the performance of conventional Artificial Neural Networks (ANNs), such as Multi-layer Perceptrons; Nonlinear Autoregressive Neural Networks with exogenous input (NARX); Support Vector Regression (SVR); and so on, on electric load forecasting, is on par with statistical forecasting methods such as ARIMA[15]–[17].

However, the effectiveness of LSTM networks on univariate time series still proves to be a challenge. LSTM networks can learn temporal sequences and long term dependencies better than any conventional neural network[18], but its oscillatory, seasonal abilities still require a more thorough analysis, especially in long-term predictions. Most regression research focuses on predicting on generally one, or up to

Figure 3.4: The different sine time series



a few time steps in advance, whereas it could be more practical to predict longer time sequences. The gas transport operator, for example, could benefit from longer prediction windows, as the gas-network is a dynamic, but slow process, where every action and its consequences unfold over several time steps, in contrast to e.g. regression of an electric grid.

There are some caveats to learning long-term oscillatory patterns with LSTMs. as [19], who trained LSTM networks on a simple sine wave, stated;

> [...] some configurations of LSTM were able to model the signal, accurately predicting up to 400 steps forward. However, we also found that similar architectures did not perform properly [...] probably due to the fact that LSTM architectures got over trained and the learning algorithm got trapped in a local minimum. [...]

In extension to this research, an experiment was conducted where LSTM networks were modelled on simple sine-based time series, in order to understand the oscillatory properties of LSTM networks.

Experiment setup

For this first experiment, several model and training parameters were tested. Next to a simple sine-wave, a combination of sine waves (two summed sines) with and without an increasing trend were modelled by the LSTMs. The sine generation was loosely based on the gas flows; The single sine had a period of a month in 1-hour time steps (24 * 31), the combination sine was composed of a sine with a day period (24 time steps) and a sine with a period of a year (365 * 24 time steps). The trend-given

sine consisted of the combination sine, with the addition of $^4/_{(2*365*24)}$ every time step. A graphical impression of the given sine time series is shown in Figure 3.4.

The topology of the LSTM network consisted of a single input node, followed by an LSTM layer, and a final Dense output node. This Dense node is a regular densely connected neuron, with connections to each of the LSTM cells in the LSTM layer. The number of LSTM cells was either 2, 5, or 50. The number of input time steps was either 1, 2, or 26. The batch-size was either 1, or 128.

Batch-size, Sample, Epoch

Batch-size in this setting corresponds to the number of samples that are propagated through the network per time step.

A **sample** corresponds to a single input/output data couple, and both the in- and output can consist of several time steps of a number of features.

For example, say that one would want to predict the next 10 values of two time series, with the previous 24 time-steps as input. A single sample would then consist of an input of 24 time steps of two values, and an output of 10 time steps of two values. With a batch-size of 128, every learning 'step' of the network consists of propagating 128 such samples through the network, collecting the model predictions, and updating the model's weights on the (average) error between the predictions and the target output.

Training the network with gradiënt descent and a batch-size of 1, is known as Stochastic Gradiënt Descent.
Training the network with the entire collection of samples in one batch, is known as Batch Gradiënt Descent, and a batch-size between these two is known as Mini-Batch Gradiënt Descent.
In addition, propagating a complete set of samples through a network is called an **epoch**.

A larger batch size means less time-steps needed for one epoch, and since propagating the samples of one batch can occur in parallel (since the network is not updated during the batch), the time needed for a single epoch can be greatly reduced. Because the weights of the network are updated after each batch, this also means less updates of the model, and each update can be seen as an average update over the batch results. Usually, the learning rate increases with larger batch sizes, as overfitting is somewhat confined by using an average result as gradient input, instead of a single sample. By training a network in batches, overall estimation of the gradient direction can be improved, by removing the noise of individual samples. However, this also means that possible information given by single samples could be lost.

In addition to the normal random sample-like training behaviour, a special practice of training called 'stateful' was also tested.

With the repeated oscillatory nature of the sines in mind, the data was split 50/50 into a training, and a test set. Next to measuring the performance of a network, the test set was used as a validation set, to prevent overfitting and allow

Stateful networks

Normally, a neural network is trained by propagating batches of random samples through the network. After each propagation, the network is 'reset' (the internal activations are reset), and a new sample can be propagated. Because of this resetting of activations in the network, any time-delayed pattern that needs to be learned by the network should occur within the supplied time steps of a single sample.
For example, in [11] LSTM networks were trained to differentiate between spike pattern sequences of either 49, or 50 time steps. The networks were trained by supplying them with samples with an input length of either 49 or 50 time steps. If the spike pattern sequences were divided over multiple samples, the network would never be able to learn to differentiate, since by the time the sample containing the end of the pattern was propagated throughout the network, all activation of previous samples would have been lost.

A possible solution for this problem in terms of sine-learning would be to ensure that entire periods of the time series are present in the models' sample input series. However, in the case of the combined sine, this would mean that a single sample would require an enormous input size. This leads to a very generalized learning of the pattern.
A better solution is the so called 'stateful' network, where the activation is not reset after each batch. In such a network, the input data has to be propagated sequentially, usually with a batch size of 1. This allows for a possibly better modelling of the time-dependant factors in a time series, as the training occurs in a more natural way.
A downside to stateful networks, is the fact that they cannot benefit from the performance upgrade of batch-training.

for early stopping in case of stagnating learning behaviour.

Results

The global results of this experiment are shown in Table 3.1. The overall performance of the networks is shown in this table, where all scores are obtained from averaging over all the tested networks.

Figure 3.5 shows a scatterplot of all the tested networks' performance. Looking at this scatterplot, it becomes clear that some networks were unable to convergence in the given number of epochs. Further inspection of these networks showed that their Root Mean Squared Error (RMSE) was not particularly high, but was still decreasing. This indicates that these networks have reached a very shallow declining platform in the error space, where they slowly but surely converge to a (local) optimum. It is however not the case that these networks have reached a local, sub-optimal convergence point, as is the case with the networks with a very high final

Table 3.1: Sines: Global results

Table 3.1 The average train and test results (mean and one standard deviation of the RMSE) of stateful and stateless LSTM networks on the different sine time series. Similar results on both train and test data indicate that no overtraining has occurred.

Stateless

	Train		Test	
	$\mu(RMSE)$	σ	$\mu(RMSE)$	σ
Single	0.000712	0.000307	0.000721	0.000312
Comb	0.000851	0.000239	0.000859	0.000242
Comb + Trend	0.161982	0.292796	0.163449	0.295214

Stateful

	Train		Test	
	$\mu(RMSE)$	σ	$\mu(RMSE)$	σ
Single	0.001261	0.000302	0.000927	0.000122
Comb	0.004150	0.003818	0.004200	0.003760
Comb + Trend	0.002062	0.000915	0.000998	0.295214

RMSE.

Remarkably, most of the networks that got stuck a a local optimum were stateless networks with only 2 or 5 LSTM cells and tried to model the trend sine wave. This leads to the early conclusion that in order to properly model a complex combination of sine-waves and trends such as the trend sine wave, a network must have a sufficient number of LSTM cells in order to capture the complexity of the time series. This conclusion is strengthened by Figure 3.6. It clearly shows that the variance in performance with networks with only 2 or 5 LSTM cells is huge, and on overage is not able to properly model the trend sine wave. Remarkably, only having two LSTM cells provides the network with enough complexity to properly model both the single, and the combination sine wave.

In order to not skew the following results, and contaminate any possible conclusions, the outliers named above were removed in further analyses.

Figure 3.5: Sines - Scatterplot of results

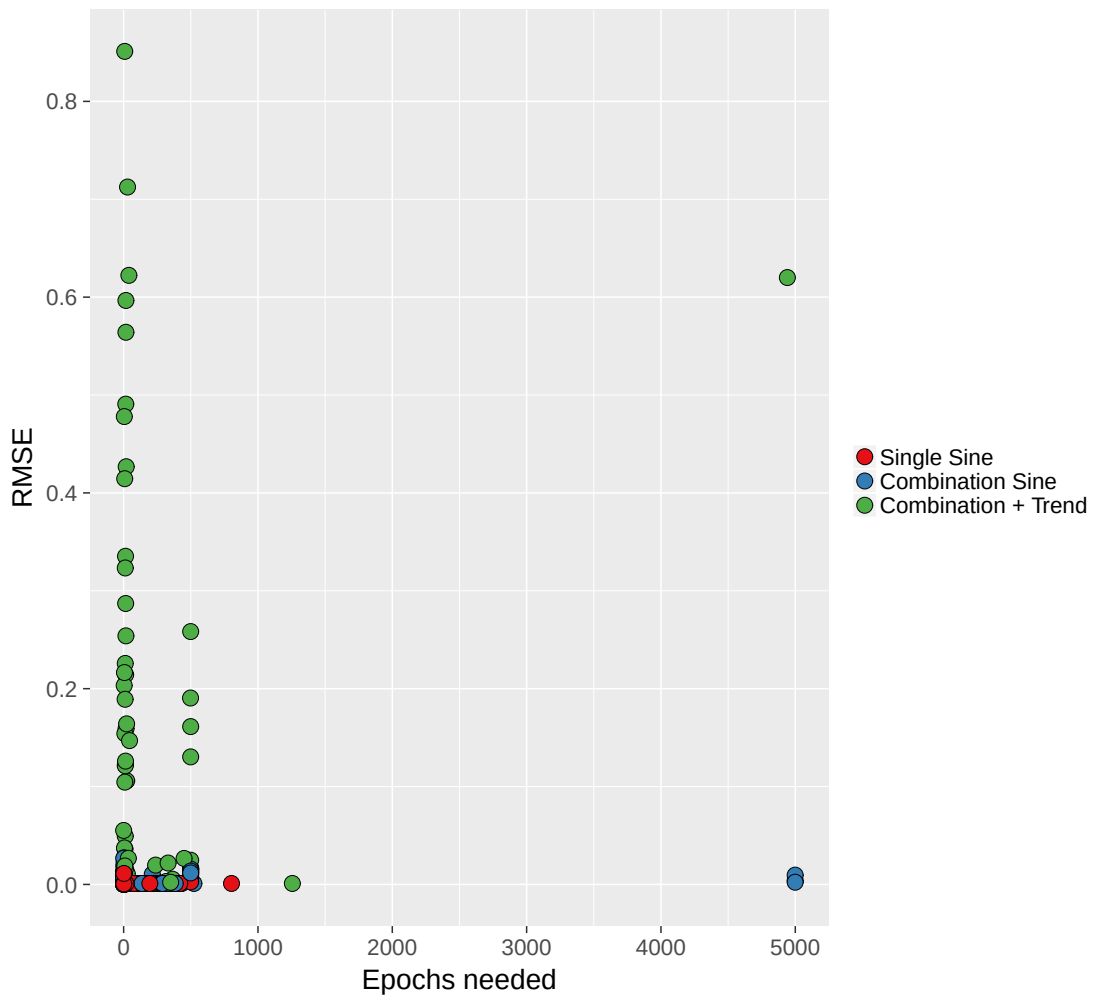


Figure 3.5 The number epochs needed before conversion, against the resulting RMSE score for each individual experiment. Each dot corresponds to a separate experiment, with the possible sine-wave models indicated by colour. High RMSE value indicates convergence on a local minima, high number of epochs indicates trouble to converge.

Figure 3.7, Figure 3.8, and Figure 3.9 further show the influence of different parameters on the network's performance.

As can be seen in Table 3.1, for the single and the combination sine, a stateful networks seems to perform better. It receives a lower average Root Mean-Squared Error, and shows less variance in its results.

However, for the trend sinewave, the stateless network seems unable to model the data. In this case, the stateful network severely outperforms the stateless variant. This effect was to be expected as the trend sinewave has a strong sequential dependant attribute; i.e. the increasing trend over time. Since the stateful network is trained sequentially, it is able to capture this property, while the stateless network - trained randomly - is not.

The stateful network does tend to be a more difficult network to train. Figure 3.7 displays that a stateful network requires a longer period of training on average.

Figure 3.6: Sines - Number of LSTM cells against RMSE

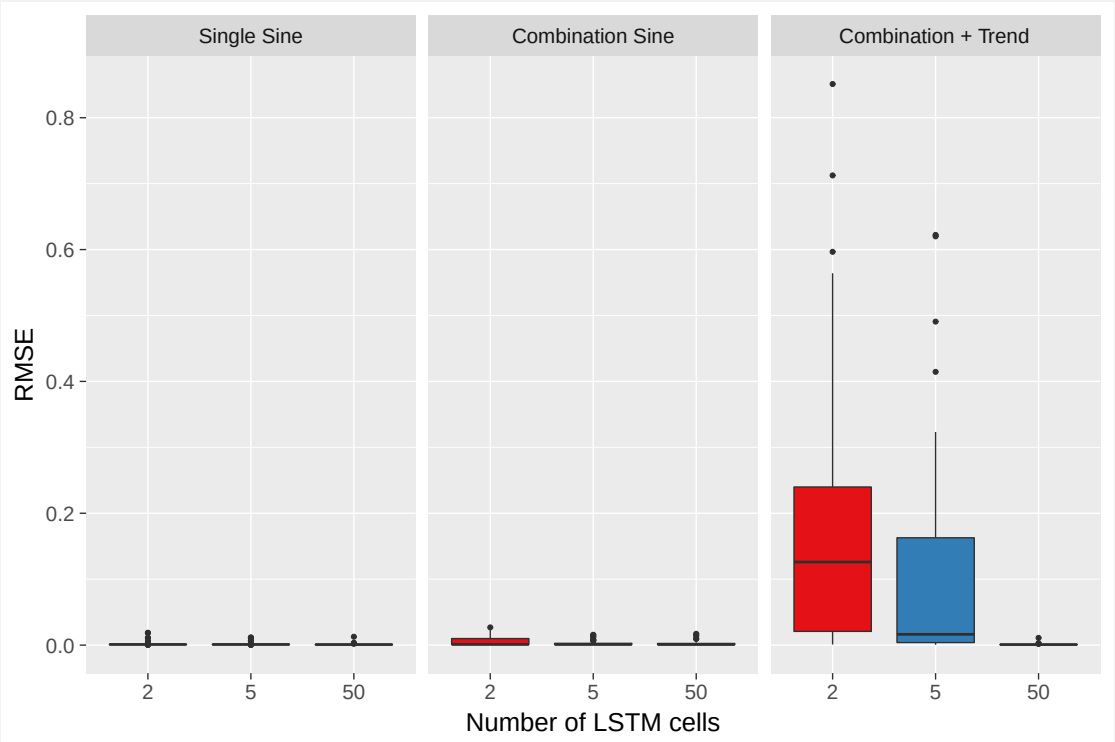


Figure 3.6 The influence of the number of hidden LSTM cells on the performance is shown by the average error (RMSE) per sine type.

Figure 3.8 shows, that the extra time needed, can not be accounted for by increased time per epoch, as the number of epochs before convergence also increases.

Figure 3.6 displays the influence of the number of LSTM cells in the hidden layer. It shows that for the easy, single sine case, two LSTM cells is sufficient to accurately model the sine wave. Increasing the number of LSTM cells does decrease the resulting RMSE but only slightly, while other factors such as training time increase.

As said before, training with a larger batch size allows for faster convergence and can overcome individual sample noise, but can also lead to a more 'general' model, resulting in higher RMSE scores, as illustrated by Figure 3.9. For example, in the case of a batch size of 128, the first batch will consists of the first 128 samples, the second batch contains samples 129–256, and so on. This greatly reduces the number of batches propagated through the network, and therefore the number of weight-updates per batch. Thus, the usage of a batch-size of 1 in the case of stateful networks is a valid choice, and the increase in runtime is hopefully countered by the increase in performance.

Figure 3.7: Sines - Statefulness against duration

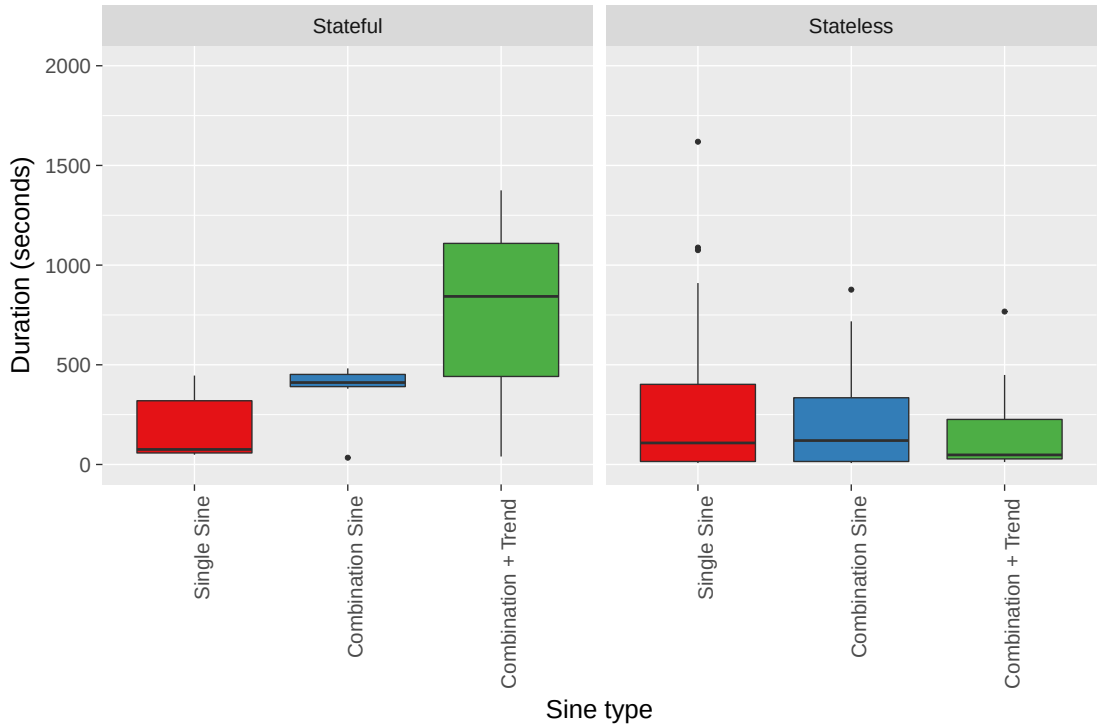


Figure 3.7 The differences in experiment duration, or computation time needed is compared per sine type for both stateful and stateless networks. A lower duration indicates a faster convergence of the network.

Figure 3.8: Sines - Statefulness against epochs

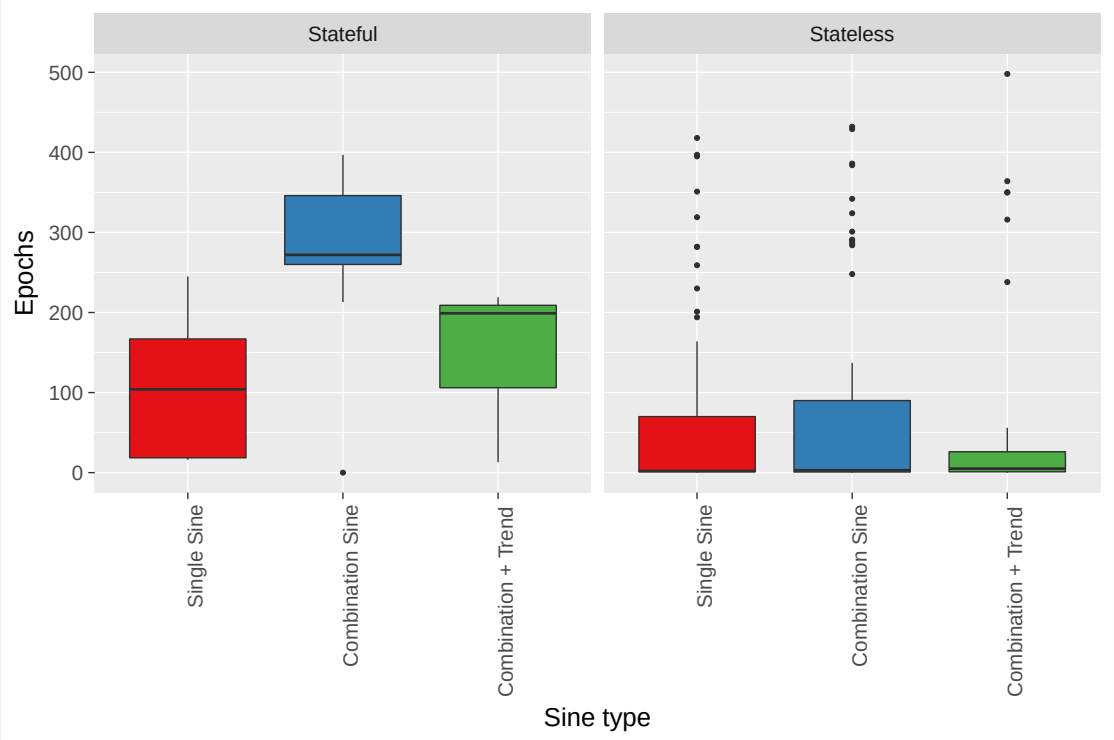


Figure 3.8 The differences in number of epochs needed for convergence is compared per sine type for both stateful and stateless networks. A lower number indicates less steps, or weight updates needed for convergence.

Figure 3.9: Sines - Batch size against RMSE

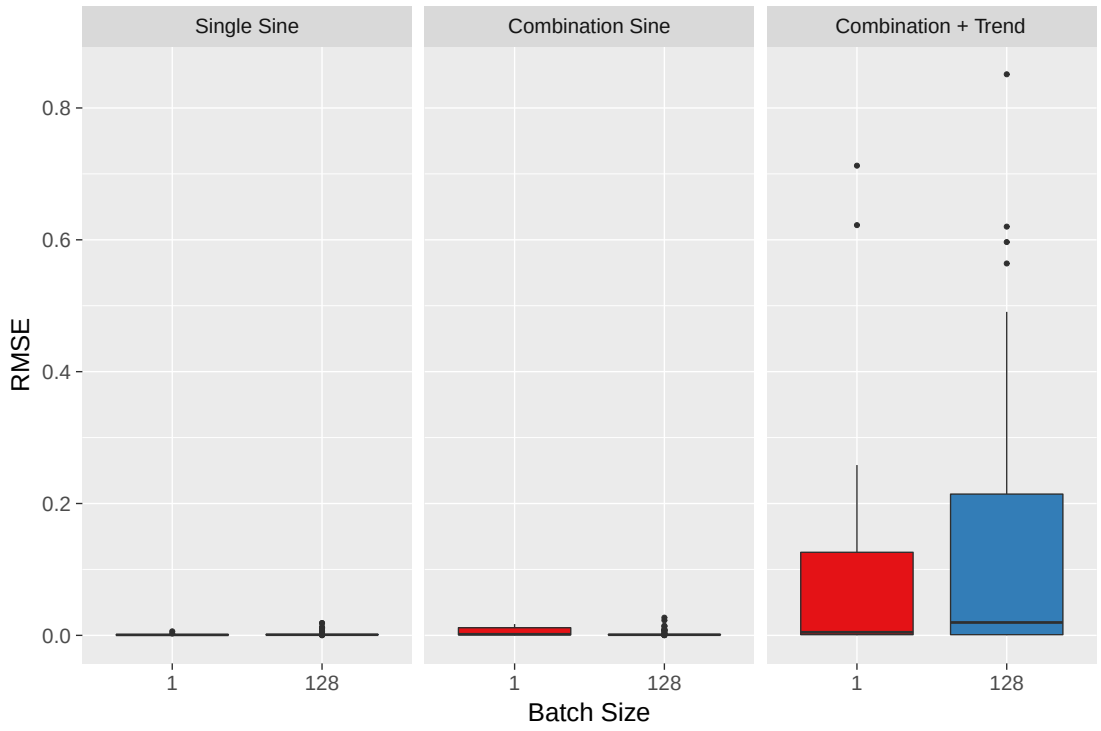


Figure 3.9 The influence of different batch sizes on the resulting error (RMSE), shown for each of the sine types.

3.2.2 Long-term prediction strategies

Regard the univariate time series as stated before: ($\mathbf{X} = [X_0 \dots X_t]$). In the default regression task, the challenge is to predict the next step in the time series given N previous steps:

$$X_{t+1} = f(X_t, x_{t-1}, \dots, X_{t-N})$$

In the case of long-term predictions, or multi-step ahead, tasks, the challenge is to predict the next H time steps, where H is called the prediction horizon.

$$X_{t+1}, X_{t+2}, \dots, X_{t+H}] = f(X_t, \dots, X_{t-N})$$

Prediction with $H > 1$ is generally done by using one of three strategies:

1. **Continuous prediction**
   Continuous, or Iterative prediction[20], is done by forecasting one time step at a time, after which the predicted value is used as input for the next prediction step. This process runs recursive until the complete prediction horizon is reached.

2. **Direct prediction**
   With Direct prediction[21], a separate model for each time step in the prediction horizon is made. The advantage of this strategy is that each network is trained for a specific delay between input and prediction. This can result is fairly small and simple networks, who are less likely to overfit and such. It does however require the training of multiple networks. Also, if there is no causal relation between the input and the current prediction step, then such a model could never achieve good performance Therefore, if the time series consists of data with a strong sequential influence, on of the other prediction strategies is likely better suited.

3. **Multivariate prediction**
   The previous prediction strategies both proposed networks with a single output value. With Multivariate prediction, the size of the output layer corresponds to H, thereby predicting the entire horizon in a single step. By producing the entire prediction horizon simultaneously, the prediction error does not compound over time. All the predictions are made from the same model (hidden layer), thus the possible temporal dependencies in the time series are preserved.

   However, this model is more complex, compared to the two other prediction strategies, and therefore more difficult to train; and more prone to overfitting; and has less flexibility.

Each of the strategies was tested, with varying results. Continuous prediction is an intuitive method, but was prone to under-/overfitting, especially on the periodicity of the time series. When continuing on previous predictions, it must be ensured that the network is properly able to model the oscillatory component of a time series, as the prediction error stacks with each consecutive prediction[22]. It took

great effort for iterative prediction to precisely match the oscillations of the sine-time series, since the prediction horizon (48 time steps) exceeded the smallest oscillatory period, and errors in intermediate predictions propagated forward through further predictions. Therefore, a small mismatch in periodicity resulted in large errors, and multiple times the predicted signal ended in antiphase with its target values.

Direct prediction was immune for this accumulation of prediction errors. However, the independent nature of the separate models lead to so called 'broken' predictions. For example, consider the prediction of the function $y = x$, a linear trend-line. The complete prediction should correspond to a straight line segment, but individual differences between the models, might result in spikes in the line, as a single model might have failed to correctly model its own prediction pattern, but also has no information about surrounding predictions, and is therefore unable to correct its results. Another drawback of direct prediction was the increased computation time in training the models, as a separate model was trained for each time step in the prediction horizon.

Multivariate proved to be best suited for the continuous prediction of the sine time series, and by extension, the gas flow time series. Because of the dynamic nature of gas transportation, the temporal influence of certain variations can be spread out over several time steps. A sudden surge in the morning, might result in a drop in flow in the evening, 10 hours later. Direct prediction was unable to adequately model this behaviour, due to the 'uncooperative' manner of predicting[22]. Continuous prediction also was unable to precisely follow such a surge, resulting in deviant subsequent predictions. However, multivariate prediction did capture this kind of behaviour, benefiting from the fact that individual predictions were not individually modelled, giving it the ability to capture temporal dependencies in the complete prediction horizon.

Chapter 4

Use-Case: Predicting RNB Stations

As said in the Introduction, the Gasunie is an international gas infrastructure company, providing the transport of natural gas in the Netherlands. A large part of their infrastructure is used for the transport of natural gas to houses and industries, to be used for heating; cooking; manufacturing; and so on.

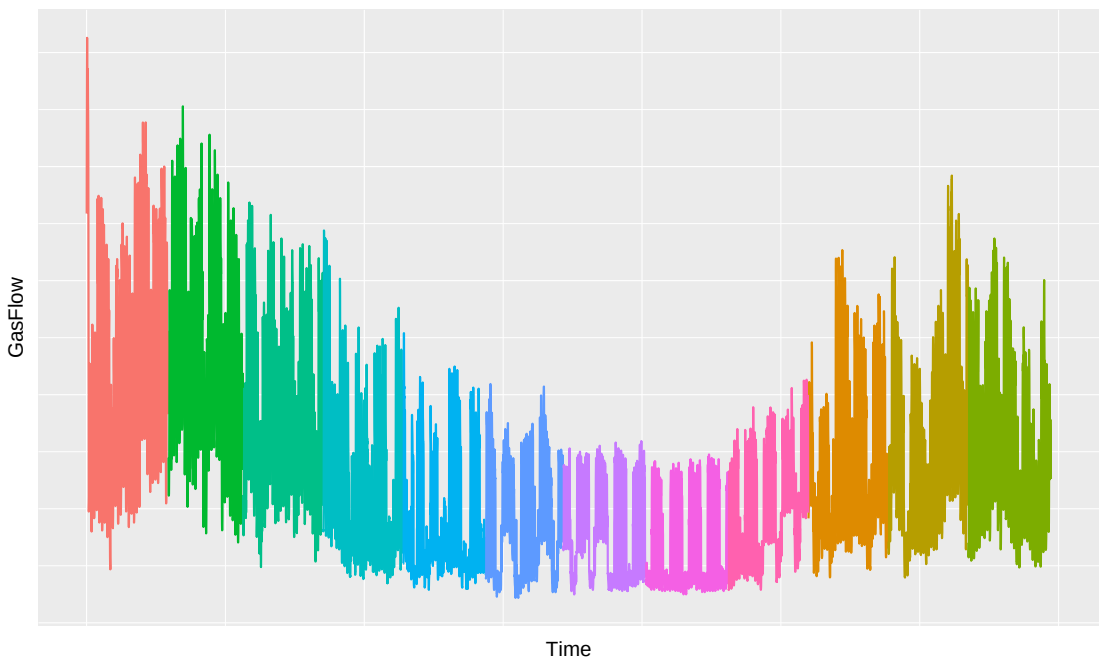
The transport of natural gas requires a complex and diversified network, consisting of numerous types of regulator stations, that allow the use of e.g. storage buffers in order to manage the large increase in transportation demand on cold day, peak-shavers for more sudden anomalies, and a more basic type of station, named regional net administrator points (Regionaal Net Beheerder in Dutch, or RNB abbreviated)

RNBs are points where the gas is delivered to a regional distribution net, to be used by regular households and industries. In contrast to the international network points, where gas flows are influenced by shippers who aim to make a profit in the stock-market like trading of gas, the gasflow of RNBs is almost exclusively influenced by actual gas demand.

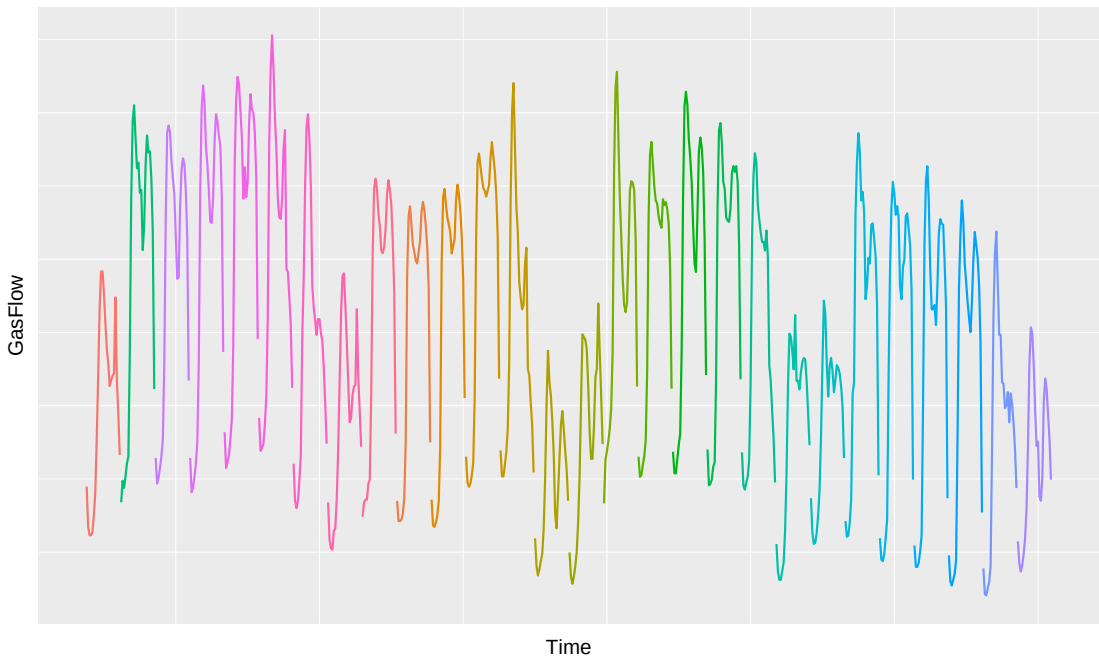
This leads to relatively easy to model times series, similar to the sine wave time series mentioned in the previous chapter, since there are not a lot of of factors influencing the the time series. An example of a gas flow time series of an RNB station is given in Figure 4.1. The year of data clearly shows a slow oscillation over the entire year, due to the fact that natural gas is predominantly used for heating. Therefore the gas demand will reduce during warm months, and increase during the cold months. Secondly, the month of data shows a characteristic pattern of flow for a normal day. This pattern is known as a 'camel hump', and relates to the fact that the highest demand for heat is during the morning (people wake up, offices are warmed), and during the evening (people arrive back home), with a through in between. The weekend effect is also clearly visible, as after 5 consecutive normal days, the flow dips for two days before rising back to normal. This dip is caused due to the fact that most people are at home during the weekend, so there is no need for the heating of large office buildings and the like.

Figure 4.1: Default gas flow of an RNB Station

A year in data - months coloured



A month in data - days coloured



4.1 The Model

4.1.1 Naivety problem

Initial networks were fitted on such a default gas flow, and achieving a low error score on small horizon predictions proved to be an easy task. However, performance quickly diminished on longer prediction horizons. The network topology consisted of a number of basic 'Dense' cells, all fully connected to the hidden layer with linear activation, who served as input layer; a (varied) number of LSTM cells in the hidden layer; followed by another set of Dense cells, serving as output layer. The number of cells in the output layer was set to the prediction horizon, the number of cells in the input and hidden layer was varied, but proved to make little difference in terms of performance.

The RNB gas demand is largely weather-based, and it was therefore not surprising that the networks converged on a well known heuristic in weather prediction. Namely that a good prediction for the weather of today, is copying the weather of yesterday. This is because consequent measures of such time series are very similar, and simply repeating the last input sample as prediction results in an easily obtained low error score.

Effectively, the neural networks started acting as naïve predictors by simply propagating the current input to the output. This also explains the lack of influence of added hidden layer cells, or additional historical values.

A possible explanation for this behaviour, is that the networks were unable to extract an adequate model of the data, and settled on the 'next best thing'. However, additional information, such as weather forecast, resulted in similar behaviour.

While the nïve predictor heuristic gives fairly good results, it is undesirable behaviour, as no actual model of the data is learned, thereby making it impossible to properly respond to unseen behaviour.

Two possible solutions were tested to solve this problem. First a lag was introduced between the input and prediction time steps. Effectively, this lead to a behaviour which was similar to Direct Prediction[21], where a separate model was trained for each forecast time step. It proved very difficult to find a suitable lag, due to the strong oscillatory properties of the time series, and long term predictions still yielded bad results.

The other approach to inhibiting the naïve predictor heuristic, was to use a Sequence to Sequence(S2S)[23] network topology which yielded better results.

4.1.2 Autoencoder

In the S2S approach, one LSTM layer is used to read the input sequence, one time step at a time, to obtain a fixed size encoding of the time series (Equation 4.1). A second LSTM layer is then trained to predict the time steps from the prediction horizon from the fixed encoding, similar to the hidden layer of the naïve networks (Equation 4.2). The encoding layer is trained following the autoencoder principle.

Autoencoders are unsupervised learning models, where the number of output nodes is equal to the number on input nodes. Its purpose is not to predict a target value Y, given input X, but rather to reconstruct its input X as its output. They consist of two parts, an encoder Φ, which transforms the input X to a fixed size

encoding, and a decoder Ψ, which transforms the encoding back to its original values.

$$\Phi : X \to E \tag{4.1}$$
$$\Psi : E \to X \tag{4.2}$$

Autoencoders are primarily used as a dimensionality reduction tool. By posing a bottleneck on the flow of information; i.e. E is usually smaller than X; an autoencoder is forced to learn a useful representation of the data, similar to the convolutional layers of Convolution Neural Networks. Therefore, after successfully training an autoencoder on the desired data, the encoder output E can be used a dimensionality-reduced representation of X. This allows an autoencoder to be used as a feature extractor[24]. In contrast to complicated, hand-engineered feature extraction algorithms such as SIFT[25], autoencoders prove to be an efficient manner of learning model representations.

4.2 Experiments

An exploratory experiment was set up to find an optimal set of parameter settings, for the use of autoencoders in the case of gas flow prediction.

Since an autoencoder is a certain type of neural network, all the caveats of neural network training also apply to autoencoders. As seen in chapter 3, a neural network needs enough complexity-modelling capabilities, through the organization and sheer number of network units, to be able to accurately model the data's characteristics. In this case, the autoencoder needed to extract the predominant components of any 48 time steps of the time series.

Usually, a regular Multi Layer Perceptron is used as autoencoder, where the number of outputs is set to be equal to the input. This type of network would neglect the possible temporal characteristics present in the data, since all time steps are offered as a 1-dimensional vector. These Spatial Autoencoders (since the datapoint's position is a dominant distinguishing factor, its interaction with its neighbours is neglected) have performed quite well on 2-dimensional visual tasks[26] or convolutional tasks[27]. However, if there is temporal information between the time-steps, the use of a temporal autoencoder might be better suited. Temporal autoencoders are build out of temporal neural networks such as LSTMs, and consider their input rather as a sequence.

In the classic spatial autoencoder, the encoder learns a simple mapping, where the high-dimensional data is decomposed in a lower-dimensional representation. Likewise, reconstructing the original input transpires via another direct mapping of hidden layer to output layer. With the temporal autoencoder, the input is fed sequentially and the model state after propagating the last time step is considered to be the decomposed representation of the original time series. Reconstructing the original time series with the decoder then consists of repeatedly feeding the decomposition to the decoder layer, allowing the temporal components of the LSTM cells (such as forget-rate, memory consolidation) to reconstruct the entire time series.

Because the input time series for the autoencoder now consists of small patterns of 48 time steps, there is no more need for the LSTM layers to be stateful. Internal activation can be reset after each presented pattern, and the train-/test-dataset can be shuffled. This also allows for some optimization of the batch-size.

The batch size, or number of samples propagated through the network each training step (propagate, compute error, adjust weights, reset), was previously set to 1 so that each time step of the entire time-series could be presented sequentially. As mentioned before, a larger batch size allows multiple sequences of time steps, or patterns, to be propagated through the network, and subsequent update steps are then adjusted to the mean error on the patterns. This allows for faster training due to parallelization possibilities and inter-sample noise is reduced. However, if the individual samples contain specific information, details might be lost in the averaging.

Initial tests revealed that larger batch-sizes converged significantly quicker, but also produce worse results. It seems that there is a trade-off between training accuracy and training speed (not training time, as large batch trained networks would need more epochs to reach similar performance, if even possible). In order to both gain a decrease in training time, and a good performance, the batch size can be decreased during training. By stepwise decreasing the batch-size initial training might be sped up, while the reductions in batch size allow for more detailed learning of the model.

Originally, a batch size of 1 was the default modus operandi for Gradient Descent, and is called **Stochastic Gradient Descent** or **SGD**. If the gradient descent was applied to the entire batch, it is called **Batch Gradient Descent**, and with a batch-size between 1 and the total data set, it is named **Mini-batch Gradient Descent**. We named the decreasing batch-size approach **Declining Batch Gradient Descent** or **DBGD**.

4.2.1 Training the Autoencoder

The experiments were performed on two years of hourly data of a regular RNB network point. Due to the periodic similarity, the first year of data was used as training set, while the second year was used as validation data.

Figure 4.2 displays a scatter plot of the RMSE of every run against its encoding dimension, Figure 4.3 shows a similar image, with the RMSE of the validation dataset. It is clear that the Temporal autoencoders performs considerably better than the Spatial autoencoders, regardless of the encoding dimension, or the batch-size approach (not indicated in the figures). However, as can be seen in Figure 4.4, Spatial autoencoders require less computation time to converge on an optimum configuration, and show less spread in computation time needed. This is analogous to the findings in chapter 3, where MLPs proved to be less accurate but more stable with regards to local optima.

It is clear that Temporal autoencoders outperform Spatial ones on this particular dataset. This leads to the conclusion that there exists a strong temporal connection between the data points, and that the Temporal autoencoder is able to pick up on this connection and uses it to improve its performance. Therefore, further analysis is done solely on experimental data from Temporal autoencoder.

Figure 4.5 and Figure 4.6, show the mean validation loss and mean computation time per encoding dimension of the Temporal autoencoders, respectively. Note that in both figures a smoothing function is applied on the data points, using a linear regression model with two degrees of freedom, so that a crude elbow point can be identified. Both the decreasing, as the stationary batch-size approach achieve similar

Figure 4.2: Autoencoder: Encoding dimension against loss

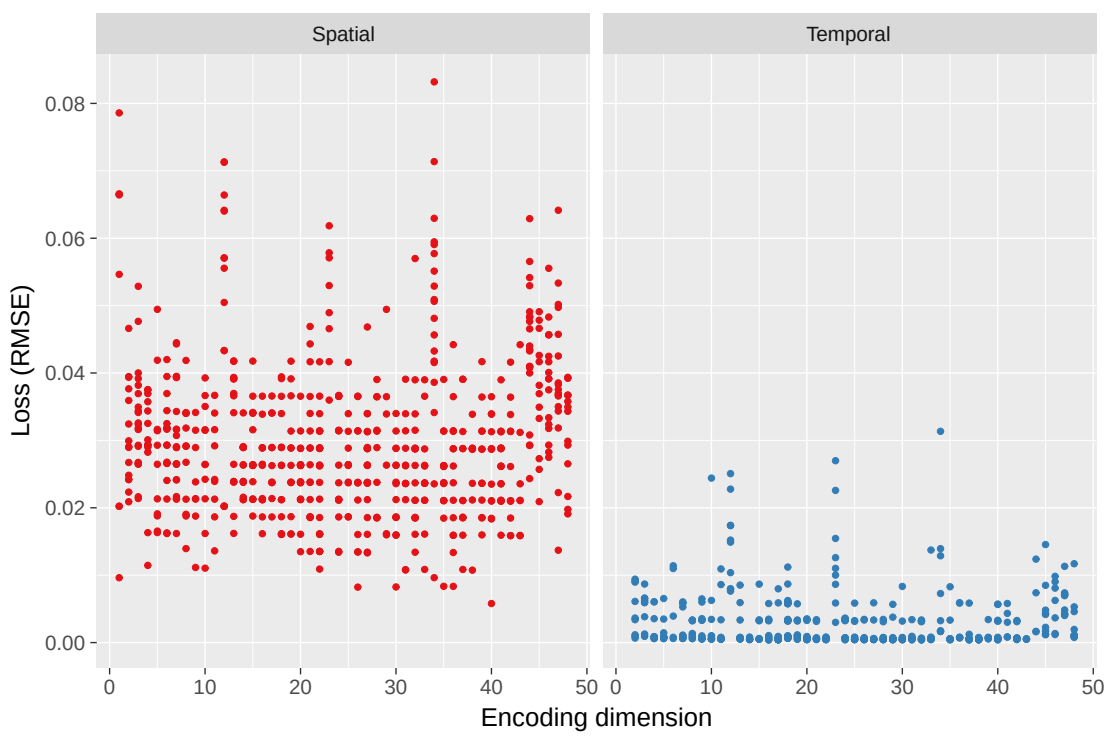


Figure 4.2 The encoding dimension set against the resulting **train** error (RMSE) for both spatial en temporal autoencoders. Each dot indicates an individual experiment.

Figure 4.3: Autoencoder: Encoding dimension against validation loss

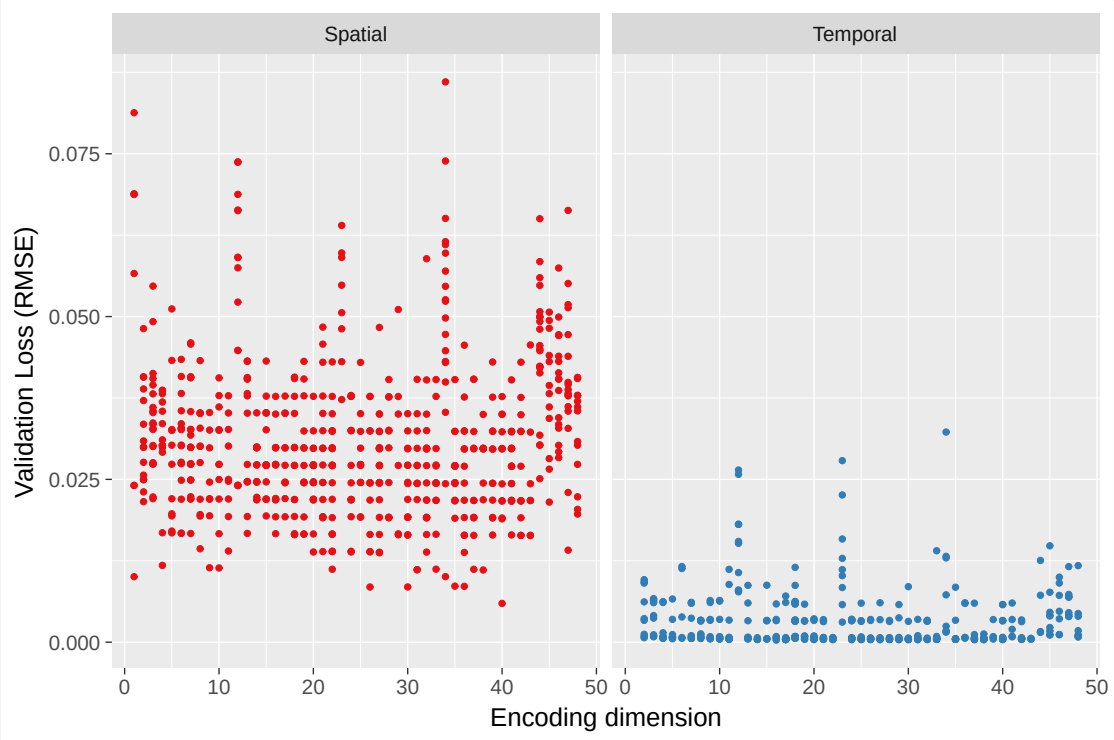


Figure 4.3 The encoding dimension set against the resulting **test** error (RMSE) for both spatial en temporal autoencoders. Each dot indicates an individual experiment.

Figure 4.4: Autoencoder: Encoding dimension against Computation Time

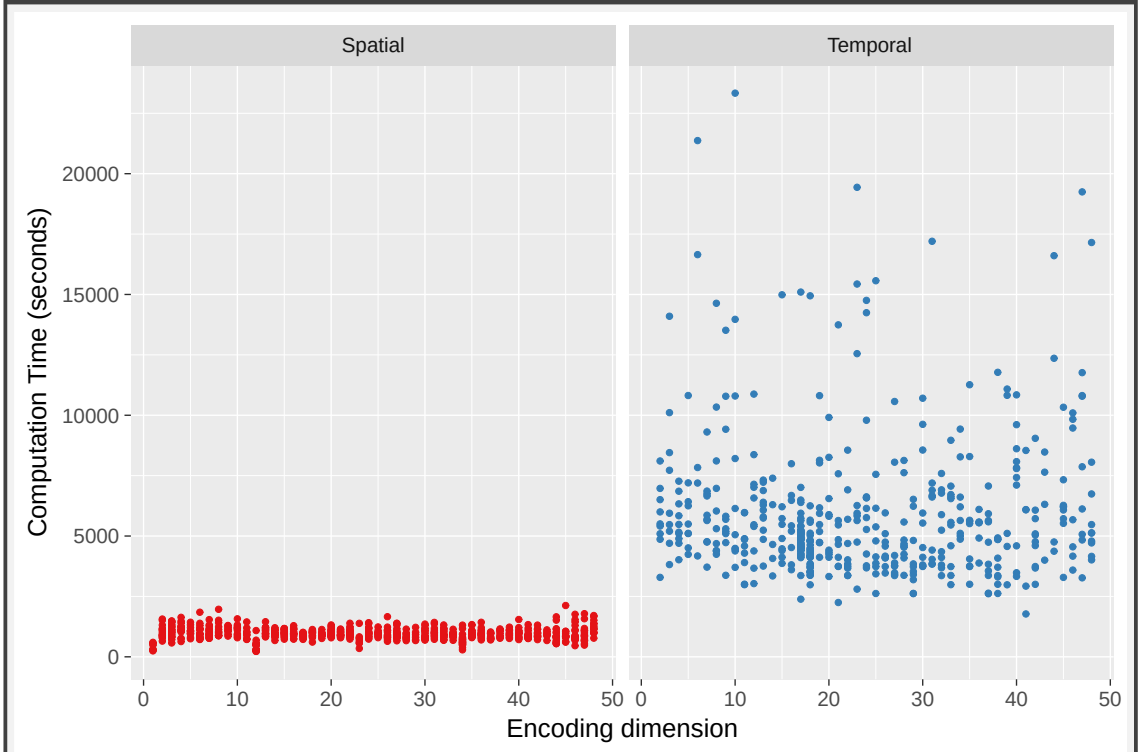


Figure 4.4 The encoding dimension set against the computation time needed for convergence. Each dot indicates an individual experiment.

Figure 4.5: Autoencoder: Mean encoding dimension against validation loss

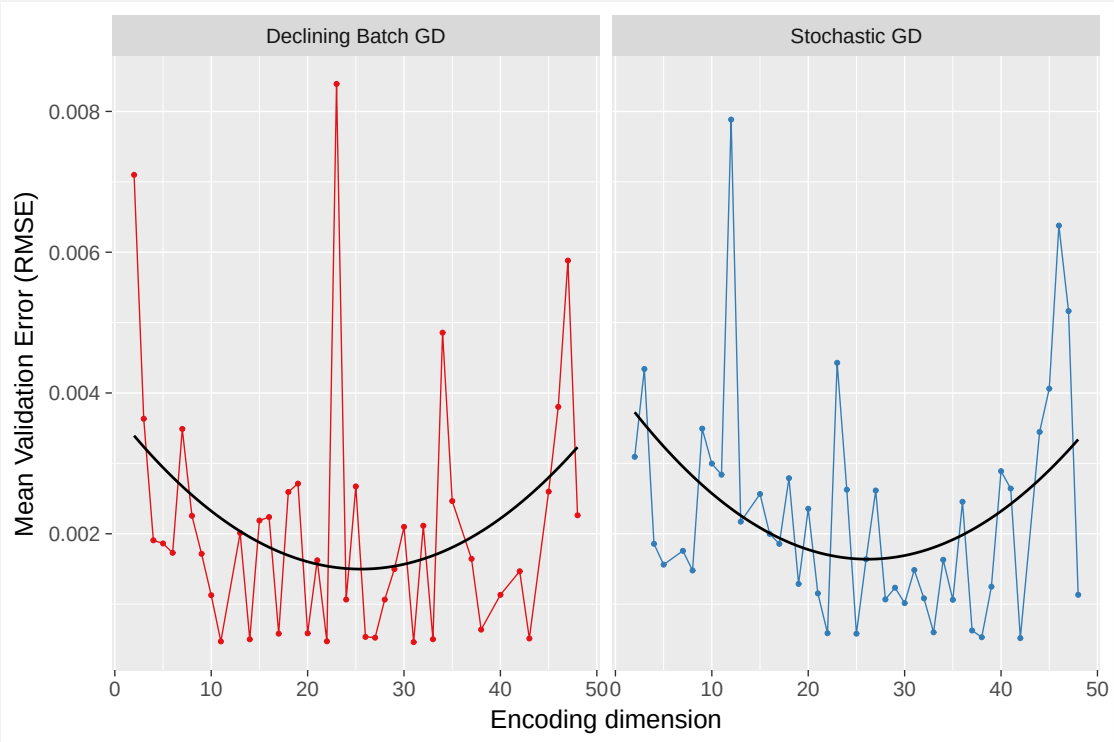


Figure 4.5 The mean validation error (RMSE) per gradient descent type, averaged per encoding dimension. An additional linear regression model is fitted on these averaged values. Each dot indicates an individual experiment.

optimal performance in terms of mean RMSE. However, the decreasing batch approach performs best using ~ 25 encoding dimensions, while the stochastic approach needs more encoding dimensions for optimal performance. Remarkably, Figure 4.6 shows that the optimal encoding dimension in terms of performance, corresponds to the optimal encoding dimension in terms of computation time. Both approaches achieve similar optimal performance in similar computation times. Figure 4.8 shows that the decreasing batch size generally takes somewhat longer to train, but, as Figure 4.7 shows, is more stable in achieving optimal performance.

Figure 4.6: Autoencoder: Mean encoding dimension against Computation Time

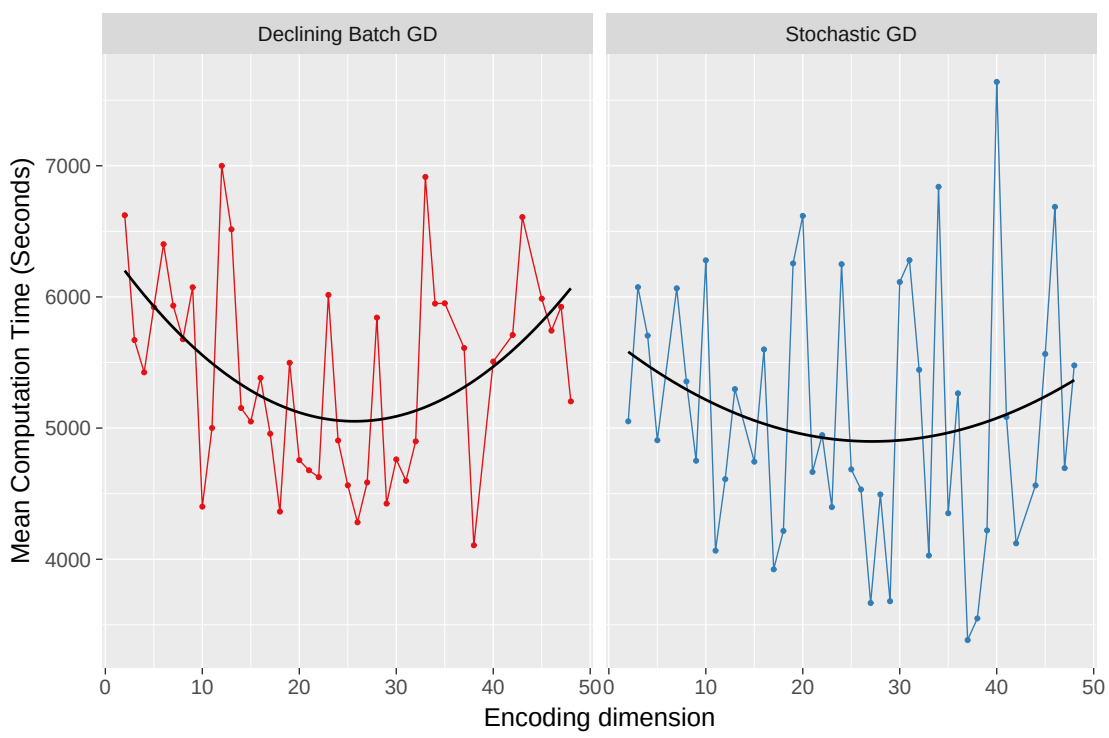


Figure 4.6 The computation time per gradient descent type on the test set, averaged per encoding dimension. An additional linear regression model is fitted on these averaged values.

Figure 4.7: Autoencoder: Mean Validation Loss

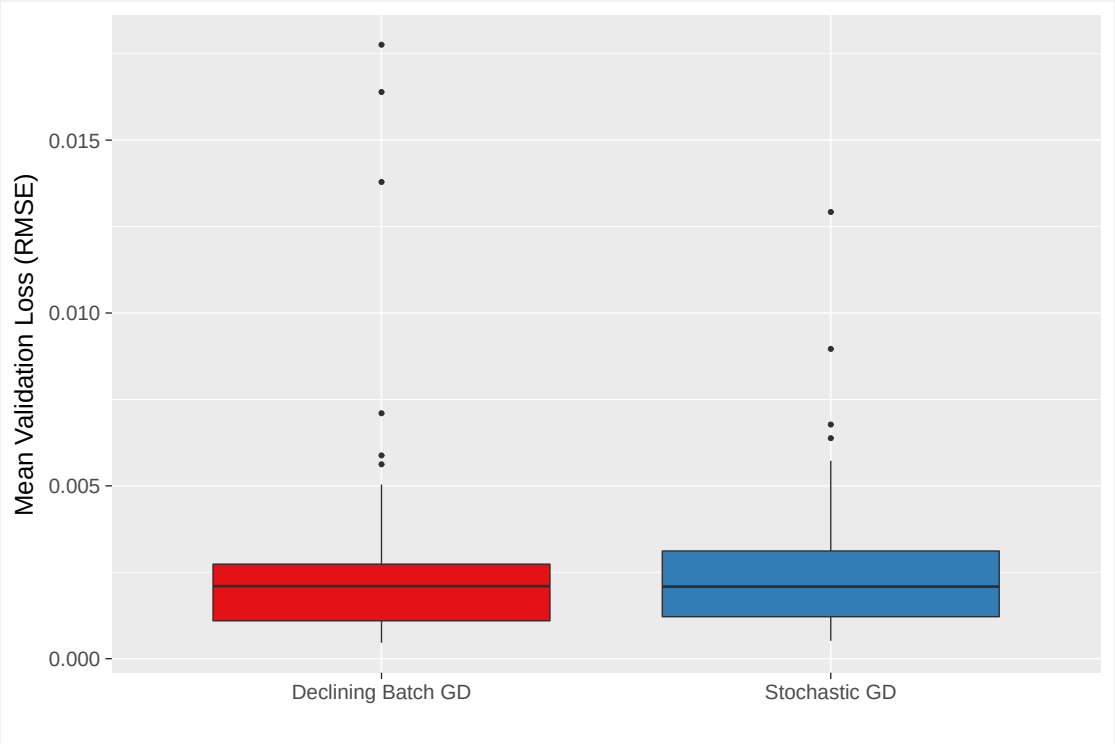


Figure 4.7 A boxplot of the mean validation error (RMSE) per gradient descent type.

Figure 4.8: Autoencoder: Mean Computation Time

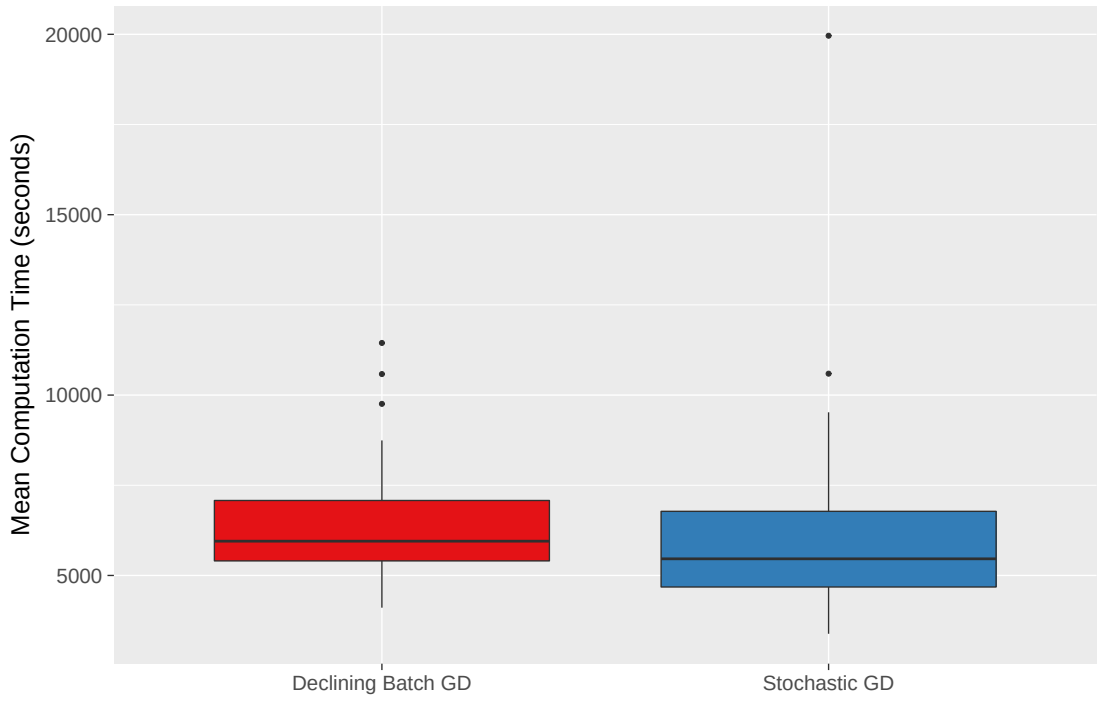


Figure 4.8 A boxplot of the average computation time needed per gradient descent type.

4.2.2 Training the complete model

Using the explored autoencoder topology, we can now form a complete neural network structure that should be able to model the gas flow time series of an RNB network point.

Our base model consists of a four layered neural network, with the first layer being the input made of Dense cells; the second and third layer consist of LSTM cells; and the final layer displays the output, again using Dense cells.

The second layer, consisting of LSTM cells, is separately trained from the third layer, following the autoencoder principle, and using the Declining Batch Gradient Descent approach. Both its in- and output consist of the time-series to model. Due to long computational times, three types of diligence were tested. At first, the model was trained until convergence, with a maximum of 100 epochs per batch size. This **Full** type of diligence needed more then 12 hours to converge. Therefore, an optimized training schedule was designed, where the threshold of convergence (difference in validation loss over the epochs) was increased, resulting in a reduction in computation time of factor 3, called **Optimized**. Finally, a **Fast** training schedule was tested to analyse the influence of thorough encoder training on the overall performance, where the encoder stopped training if a validation RMSE of 0.1 was reached, and with a maximum of 10 epochs per batch size.

The third layer, consisting of 100 LSTM cells was used for the regression part of the task. Its input consisted of the time series encoding given by the previous layer with the possible addition of the weather predictions for the time steps to predict. This regression layer was trained using standard Stochastic Gradient Descent, training was stopped when 5 consecutive epochs did not yield an improvement in the validation error. Results of these topologies are presented in the next section.

4.3 Results

Table 4.1: RNB: Global results - Autoencoder

Table 4.1 An overview of the first, autoencoder part of the network's results in terms of train and test error, and computation time needed.

	Train Error (RMSE)		Test Error (RMSE)		Computation Time (s)	
	Mean	SD	Mean	SD	Mean	SD
Fast	0.0860	0.0069	0.0930	0.0060	450.64	142.56
Full	0.0019	0.0004	0.0023	0.0005	86527.67	46450.23
Optimized	0.0053	0.0026	0.0063	0.0031	34369.75	16866.34

Table 4.2: RNB: Global results - Regression

Table 4.2 An overview of the total regression network's results in terms of train and test error, and computation time needed.

Only Flow used

	Train Error (RMSE)		Test Error (RMSE)		Computation Time (s)	
	Mean	SD	Mean	SD	Mean	SD
Fast	0.0196	0.0043	0.0535	0.0113	946.7895	349.4365
Full	0.0079	0.0006	0.0475	0.0117	2261.0304	913.8239
Optimized	0.0122	0.0043	0.0399	0.0063	1144.0061	428.9412

Weather and Flow used

	Train Error (RMSE)		Test Error (RMSE)		Computation Time (s)	
	Mean	SD	Mean	SD	Mean	SD
Fast	0.0193	0.0049	0.0432	0.0170	1123.1286	350.4972
Full	0.0046	0.0007	0.0257	0.0127	6586.3261	1804.8984
Optimized	0.0105	0.0037	0.0266	0.0065	1443.9870	1092.6564

Table 4.1 presents the results of the encoder layer of the model, Table 4.2 displays the results of the second, 'regression' layer. As could be expected, the Full diligence gives the best performance in terms of RMSE scores, and the worst in computation time, followed by the Optimized, and Fast diligence. The hyperbolic decline in error, which is characteristic for these type of networks, is visible in the resulting RMSE scores. For achieving a factor 10 better performance, a factor 100 more computation time is needed. While the general assumption that a better performing encoder yields better overall accuracy, perhaps a sufficiently trained encoder already contains enough information for the overall system to achieve acceptable performance.

Similar observations are shown by the regression results (Table 4.2), where the actual regression performance is displayed. Again, the Fast diligence networks achieve the highest RMSE scores. Both the Full, and the Optimized networks benefit from the addition of weather predictions. Remarkably, the Fast networks perform the same on training, regardless of the weather predictions. The Optimized networks actually surpass the Fully trained networks, if weather information is used in the model.

Figure 4.9 and Figure 4.10 display the average loss during training and testing, respectively. The combined use of flow and weather information allows the Full diligence to achieve a very low training loss. This also shows in the test loss. When only using the flow information, the Full diligence achieves the lowest training error, but this is not reflected in the test results. Here, the optimized diligence has the best performance, indicating that the exhaustive training of the full diligence might

Figure 4.9: RNB: LSTM boxplot Training Loss

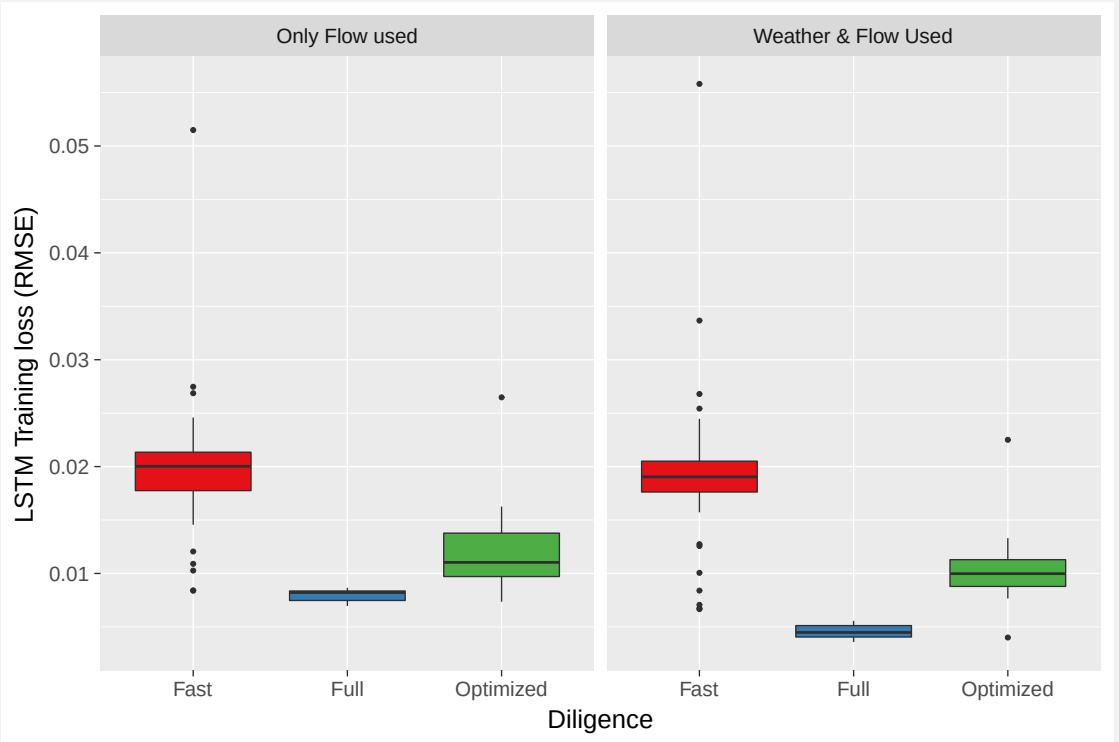


Figure 4.9 shows the influence of the different types of diligence on the resulting error (RMSE) on the **train** data, for both flow-only and weather-flow models.

lead to some overtraining.

Figure 4.11 shows a boxplot of the computational time needed to train the final LSTM regression layer. Note that this does not include the training time for the first, encoder part of the network, which is were the different diligences were applied. If all encoding training diligences resulted in proper encodings, where all the critical information was contained in the encoding without much additional noise, then regression performance and computation time needed would have been similar. This, however, is not the case. The addition of weather information to the regression layer leads to significantly increased computation time for the Fully trained encoding layer. Compared to the other two diligences, where the addition of the weather information only slightly increases the computation time, this might indicate that the fully trained encoder does not contain the principal information components of the time series (most likely due to overfitting), or is not able to properly extract these components from the encoding. Both the Fast and the Optimized encodings only have slightly increased computation times, while test performance increases significantly.

Figure 4.10: RNB: LSTM boxplot Test Loss

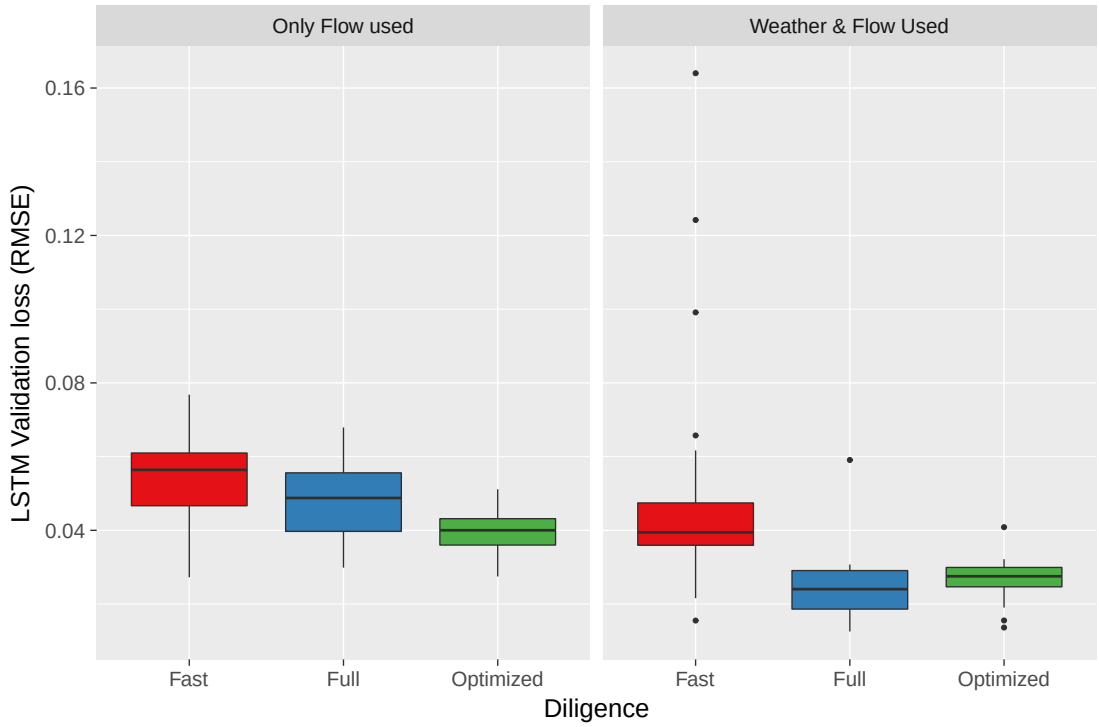


Figure 4.10 shows the influence of the different types of diligence on the resulting error (RMSE) on the **test** data, for both flow-only and weather-flow models.

Figure 4.11: RNB: LSTM boxplot Computation Time

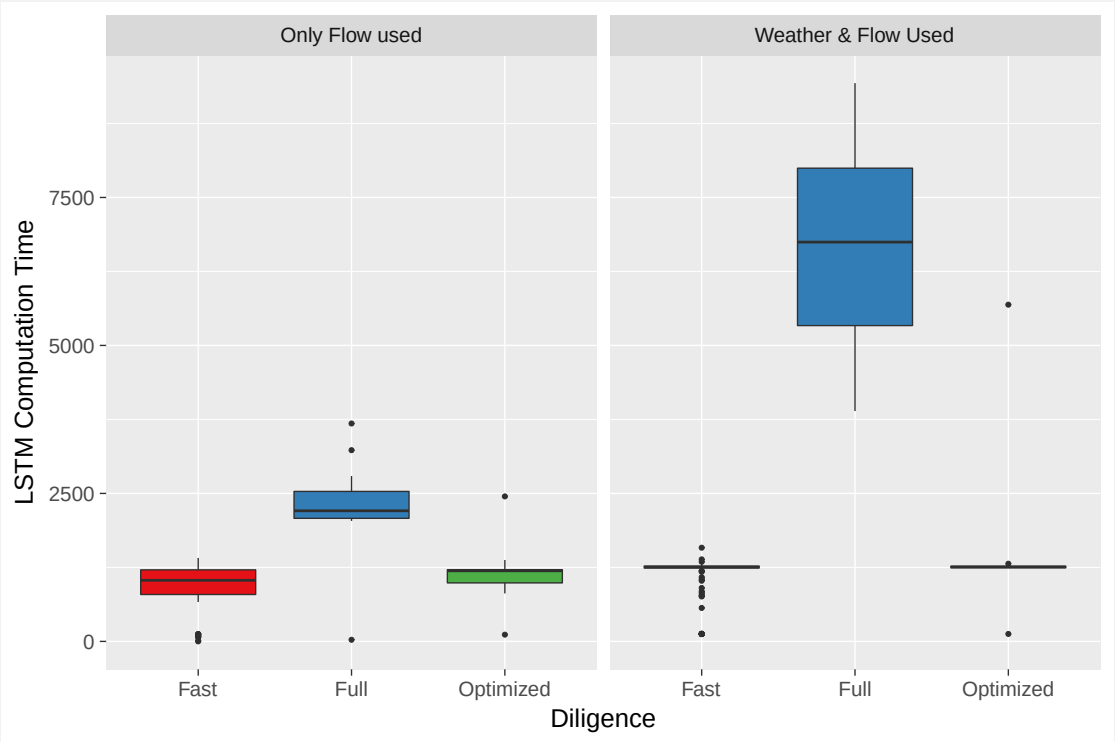


Figure 4.11 shows the influence of the different types of diligence on the computation time needed, for both flow-only and weather-flow models.

4.4 Practical comparison with ARIMA

The previous section showed the results of the application of an LSTM network on the regression task of a gas flow. In order to fully assess the performance of the LSTM on this particular task, a comparison can be made with traditional methods. At the moment, no particular methodology is used to predict gas flows, but they often rely on the expertise and experience of the dispatchers, or on well-known heuristics. Because the RNB stations are very temperature driven, a popular heuristic is to simply copy the flow of yesterday as a prediction for today. Remarkably, at first the LSTM network also learned the use of this heuristic, as explained in subsection 4.1.1. Experiments have been done using a number of statistical regression methods, such as ARIMA. Results of the ARIMA were reported as 85% of the predictions had a relative error distance within 20% of the target value. Applying their model to this particular setup gave it an RMSE score of 0.117 for a specific point in the network. The naivety heuristic results in a RMSE score of 0.0350 on the same network point, whereas the LSTM network scored an average RMSE on the test set of 0.013, averaged over ten separate runs.

So in terms of average performance over time, the LSTM network has a preference over the other methods. However, since these predictions are meant as guidelines to the dispatchers, who in turn operate on a nationwide gas network, average performance is not enough. Especially when anomalies occur in the network, e.g. when certain levels of flow are reached that require a change in network operations, the system needs to have reliable predictions.

Figure 4.12: RNB: RNB: Comparison of Error over Time

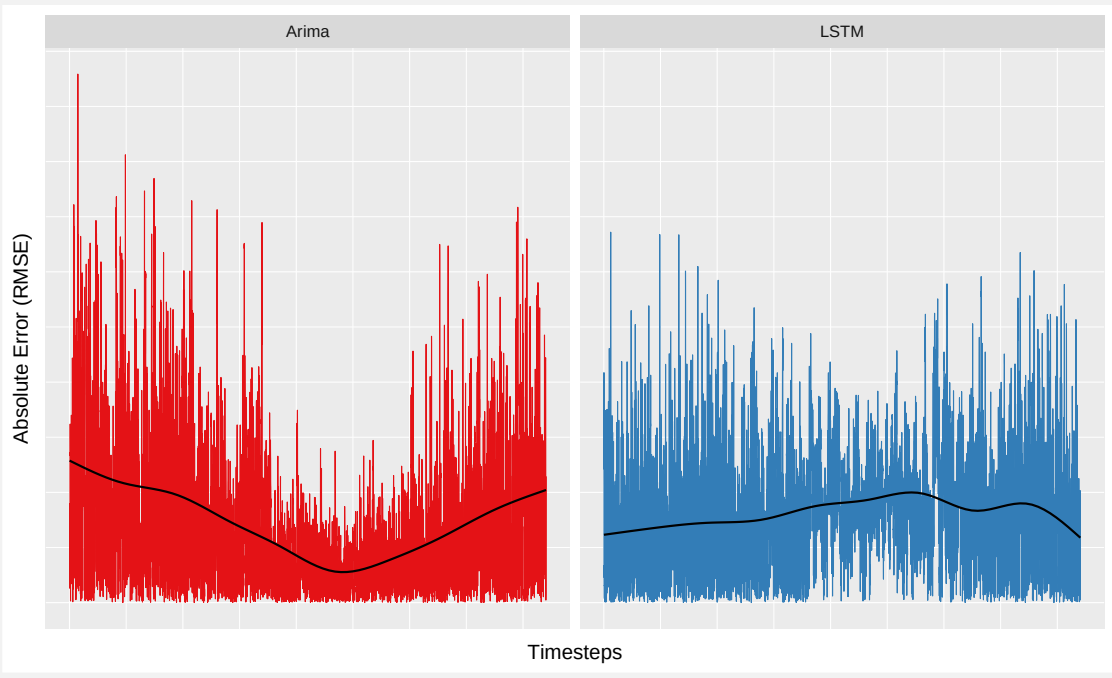


Figure 4.12 compares the absolute RMSE over a year of test data, between the ARIMA and our LSTM model. The LSTM model shows less variation than the ARIMA model, and a lower error (better result) on average.

Figure 4.1 shows that during the summer time, gas flow is fairly regular and predictable, while being more varied in the winter months. Of course, if the temperature outside is sufficient, the temperature driven usage of heating falls to zero, leaving only the basal influences in gas flow, e.g. cooking or continuous factory procedures. Not surprisingly, all the methods perform well during this period. More interesting is performance on the, less predictable, cold months.

Figure 4.12 shows the Absolute Error ($abs(prediction - target)$) over time of the LSTM and the ARIMA, with an added trend line. The pattern of ARIMA's error displays an impediment when using ARIMA. That is, when the time series is highly variable, ARIMA will attempt to model the average behaviour of its given input, and will continue this trend in its prediction. Suppose that tomorrow wil be colder compared to last week. ARIMA's prediction for tomorrow will be the averaged behaviour of the previous days, and will therefore be too low. However, when tomorrow has a similar weather prediction as the previous days, continuing the average trend is actually a good flow prediction for tomorrow.

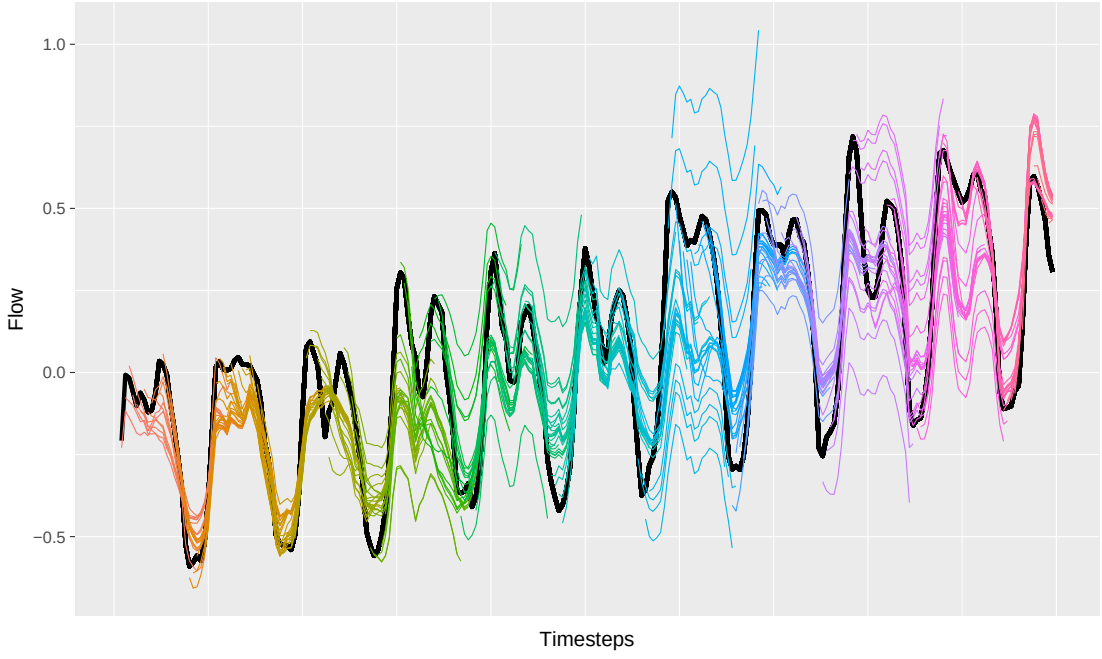
Remarkably, the LSTM shows an inversed pattern. Performance during colder months is good, as the temperature still influences the gas flow amount. Unfortunately for the LSTM, this effect tends to stop in warmer months. This is because around 16 degrees Celsius there exists a so called heating threshold. At this temperature, no more gas is needed for the heating of buildings, and an increase in temperature no longer results in higher gas flows. Because an ARIMA is only trained on the last few days, it is able to pick up on this behaviour fairly easily; the LSTM on the other hand, is trained continuously over the entire time series, and proves unable to account for this structural change.

This behaviour is further illustrated in Figure 4.13 and Figure 4.14. Figure 4.13 shows several predictions (each prediction has its own coloured line) on the first days in January, while Figure 4.14 shows several predictions of both methods around the 16th of July.

Each line in Figure 4.13 is the prediction of a separately trained ARIMA. The variety in flow for close predictions shows that the ARIMA had trouble in estimating the correct starting flow and/or increasing trend in the time series. The LSTM on the other hand, is better able to model the trend of the time series. It also displays a much better resemblance of the characteristic 'camel humps' pattern; the two peaks of gas flow during the morning and evening of a day, whereas the ARIMA sometimes treats the entire day-flow as a single sine peak, ignoring the through in between the peaks. Because of the varying daily behaviour, the naive heuristic also is unsuitable to use as prediction. As expected, the continuous training of the LSTM turns out to be a disadvantage when confronted with structural changes in the time series, such as in the summer(Figure 4.14).

Figure 4.13: Winter Predictions

Arima predictions



LSTM predictions

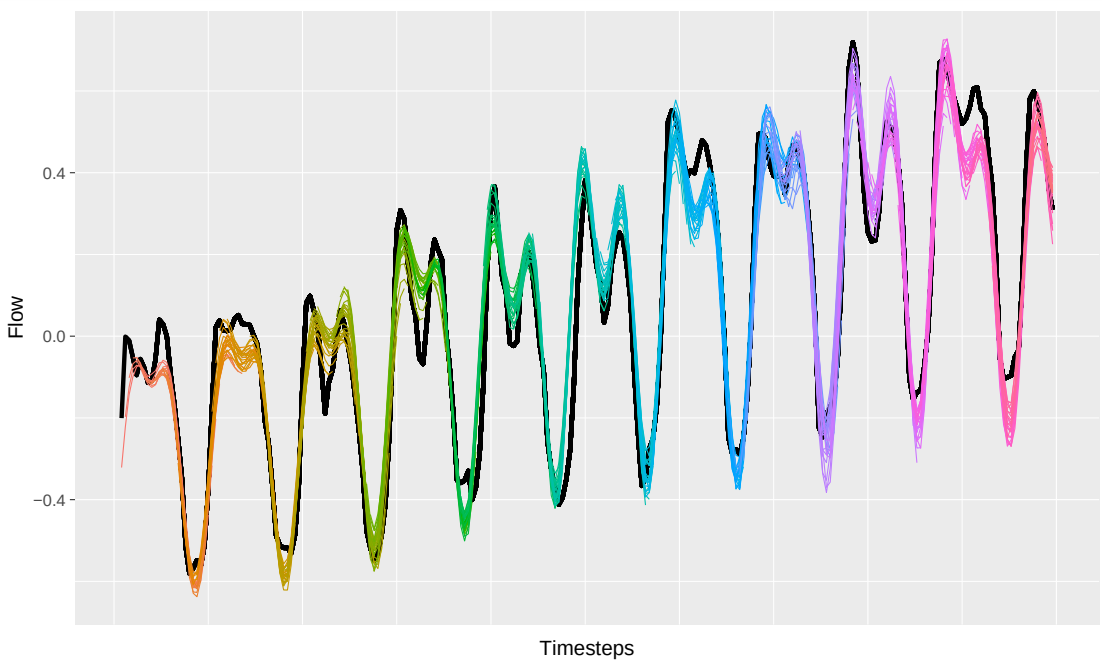
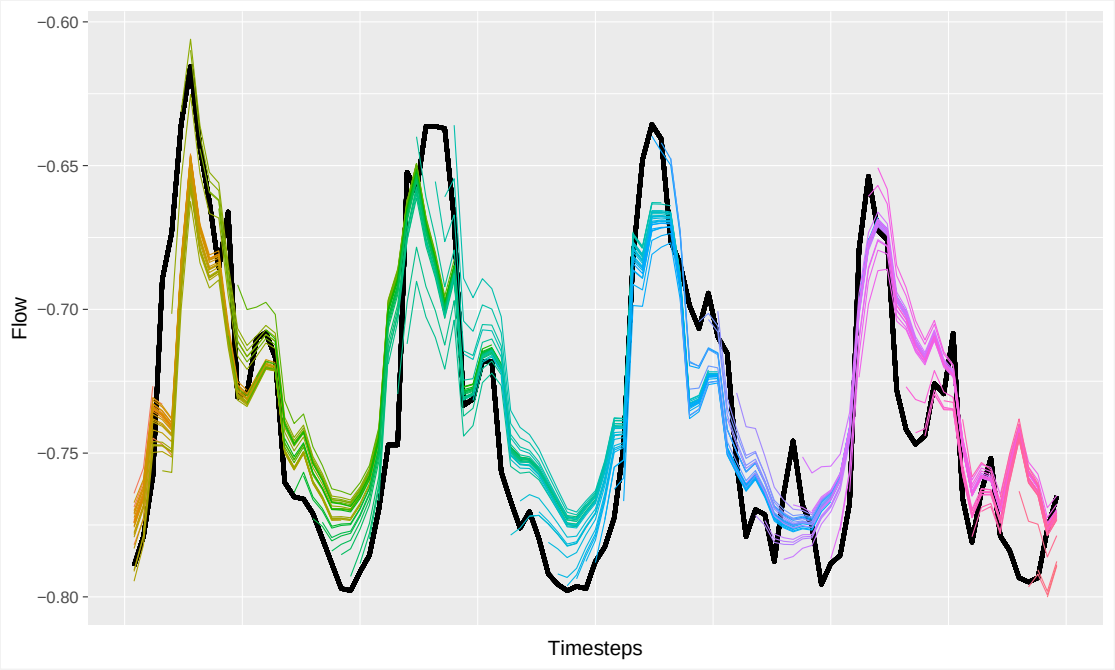


Figure 4.13 shows several predictions of both the LSTM and the ARIMA model on several days of test data in winter time. Each coloured line is an independent prediction of a time interval with its origin one hour after the previous prediction. The black line indicates the prediction's target value. The LSTM models produces much more stable results, closely mimicking the target values, where the ARIMA model is inconsistent in its predictions.

Figure 4.14: Summer Predictions

Arima predictions



LSTM predictions

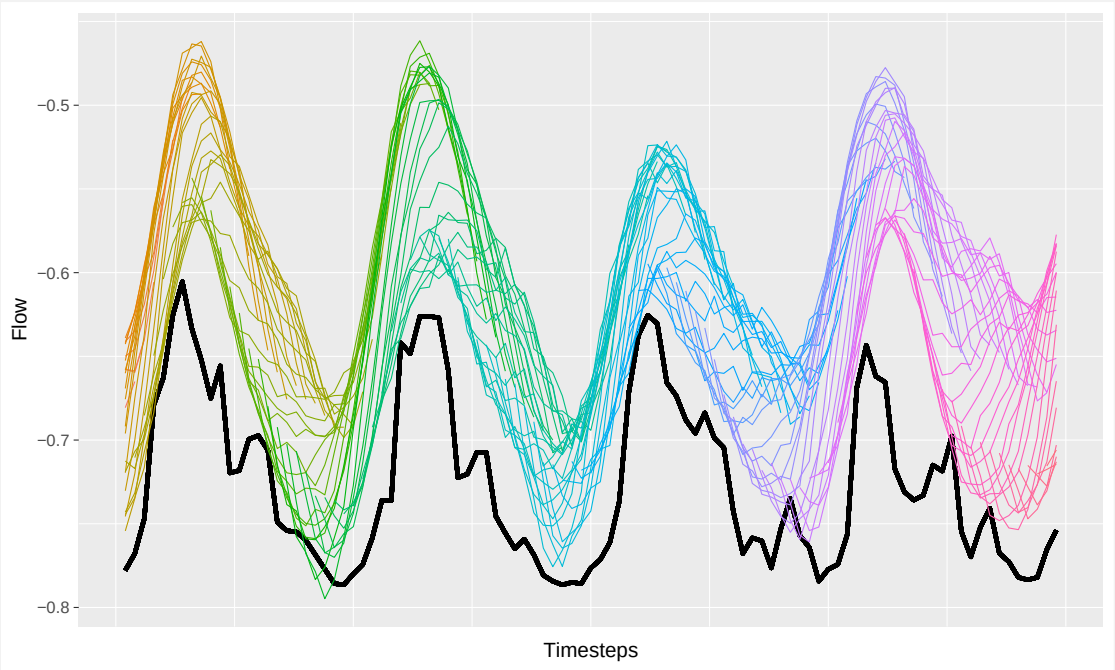


Figure 4.14 shows several predictions of both the LSTM and the ARIMA model on several days of test data in summers time. Each coloured line is an independent prediction of a time interval with its origin one hour after the previous prediction. The black line indicates the prediction's target value. The LSTM model shows consistent predictions, but fails to mimic the changed, underlying target structures.

Chapter 5

Use-Case: Predicting International Network

5.1 The Model

The Introduction already mentioned that the operating tasks of the dispatchers in the national gas transport network fall into two categories; managing the national market with regional subnetworks and local industries, such as the RNB stations, and overseeing the transport of the international market through managing the orders of gas shippers. Most of the shippers do not operate on a single network point, but instead have portfolios of multiple flows, operating on multiple points in the network. Underlying their behaviours is a set of shipper-dependent business rules, mostly unknown to the Gasunie. The shippers have varying parties depending on them for the supply of natural gas, but the acquisition/transportation orders of natural gas can also be influenced by the market prices in different countries.

Predictability of the gas flows in the international transport is therefore largely dependant on the ability to model individual shipper behaviour. Without having any knowledge of their business rules and agreements, the available information is restricted. Furthermore, due to internal information organization and time limitation it proved infeasible to include stock pricing information in our international model. Instead, the neural network was tested to properly model gas flow and shipper behaviour purely on previous behaviour.

Regarding the conclusions drawn in the previous chapters, an approach similar to the RNB use case was designed. However, where the RNB use case only modelled the flow in a single network point (with usually only 1 or 2 underlying separate flows), the international network needed to fully encode the complete international system state, to be able to model flow transient behaviour. It was therefore decided to attempt to model all the existing gas flows of the international network points simultaneously. Three different methods of encoding were tested, with a varying degree of individuality. The international network is not only fairly complex, but it is also highly mutable, with the addition and subtraction of network points and/or flows on a regular basis. While the LSTM might be able to account for some variations in the overall structure, large trend breaks such as complete networks suddenly having a flow of 0, might result in bad performance. A smaller, but similar effect happened with the RNB use case, where the gas flow stagnated above high temperatures, due to a so called heating threshold.

Therefore, the network was trained on data from 2014–2017, where the first two years were used as training data, the third year as validation data, while the final year was the actual test data. Flows that stopped working before the end of this dataset were removed from the dataset, resulting in a total of 188 flows, spread out over 20 network points. Similar to the RNB use case, the complete model consists of of two parts with separate functions. The first part provides encodings of the actual time series, where the previous 48 time steps of a time series were encoded. The second part had to perform the regression task on the supplied encodings, predicting the next 24 hours **per** gas flow, resulting in a total of 4512 output nodes. This part consisted of 500 LSTM cells, connected to 4512 Dense output nodes.

Three different encodings setups were tested. First, a single encoder was trained per gas flow, with 25 LSTM cells as single flow encoding dimension. This **Individual** approach therefore supplied the second regression layer with 4700 values, or 25 encoding values per 1 of the 188 data flows. In order to have some generalization and aggregation of data in our input, a **Networkpoint** approach of encoders was trained. Here, an encoder was trained per network point, receiving the flow values of each of the flows operating on this network point per time step. This resulted in a total of 20 trained encoder models. As each of these models had an encoding dimension of 25 LSTM cells per flow, again resulting in a total of 4700 encoding values for the second layer per time step. Finally, a **Global** autoencoder was trained. This single encoder received the 188 flow values of all the flows each time step. Due to computational limitations, the encoding dimension per flow had to be reduced to 5 LSTM cells, resulting in 'only' 940 input values for the regression part. Each possible encoder was only trained once until convergence due to time limitations, (each of the 188 individual encoders took 1.8 hours on average, resulting in a computation time of around two weeks).

5.2 Results

Table 5.1 shows the autoencoder training results, whereas Table 5.2 displays the complete regression results. The autoencoders were trained until optimized convergence, as explained in the previous use case. Where in the RNB use case trained autoencoders reached a mean squared error of around 0.006 on the test set, trained **International** autoencoders performed a factor 10 worse, indicating that international gas flows are perhaps less predictable, and harder to capture in a proper encoding. A notable detail is the fact that individual and networkpoint autoencoders performed the same during training and testing, while the Global autoencoder (possibly due to its limited complexity), performed worse during testing, indicating possible overfitting.

Further exploration of the model's hyperparameters is needed, but was not feasible during this thesis, mostly due to the long training time needed as can be seen in the Computation Time table.

Compared to the results achieved in the RNB use case, the total regression results are definitely lacking. Where the RNB models were able to achieve an RMSE score of around 0.025 on the test set, the International models were only able to achieve this level of performance during training. Each of the models ended with a test set RMS error of around 0.17, again a factor 10 worse than during training. Visual inspection confirmed these results, all three models learned a similar approximation

of the *average* flow over time, but failed to learn any interconnections between the different flows. A possible explanation en suggested improvements are given in the next chapter.

Table 5.1: International: Encoder results

Table 5.1 shows the first, autoencoder part of the networks' results on the International usecase in terms of train and test data performance and computation time needed per encoder type.

Encoder - Averages per encoder

	Train Error (RMSE)		Test Error (RMSE)	
	Mean	SD	Mean	SD
Individual	0.0280	0.0568	0.0304	0.1234
Networkpoint	0.0267	0.0245	0.0461	0.0725
Global	0.0362	NA	0.0890	NA

	Computation Time (s)	
	Mean	SD
Individual	6497.24s	4870.96s
Networkpoint	5472.32s	2455.78s
Global	14 021.59s	NA

Encoder - Totals per encoding type

	Computation Time (s)		
	Number of encoders	Total Time	
Individual	188	1 221 482.80s	(2.02 weeks)
Networkpoint	20	109 446.43s	(1.27 days)
Global	1	14 021.59s	(3.89 hours)

Table 5.2: International: Regression results

Table 5.2 shows the complete regression networks' results on the International usecase in terms of train and test data performance and computation time needed per encoder type.

Regression - Averages per encoder type

	Train Error (RMSE)		Test Error (RMSE)	
	Mean	SD	Mean	SD
Individual	0.0237	0.0020	0.1714	0.0027
Networkpoint	0.0269	0.0010	0.1767	0.0033
Global	0.0282	0.0008	0.1701	0.0028

	Computation Time (s)	
	Mean	SD
Individual	46 568.43s	16 100.17s
Networkpoint	61 512.08s	14 008.18s
Global	26 061.75s	4103.23s

Chapter 6

Discussions, Conclusions and Recommendations

In the following chapter, discussion of the results is presented in order. Possible improvements and extensions/future research topics are also mentioned.

6.1 Predicting Univariate Time Series

In this section, the modelling capabilities of LSTM networks were researched to find if and how they could be used for time series regression analysis. Previous research[19] suggested that LSTM networks were easily overtrained but we found this not to be the case. Only a particular set of parameters lead to high error scores, as can be seen in Figure 3.5 and Figure 3.6. i.e. all the Combination+Trend models, with an RMSE deviating from the cluster below 0.1; consist of only 2 of 5 internal LSTM cells. No other form of overtraining was found in the experiment. A few models got stuck in local optima, the 4 models with more than 1000 epochs needed for convergence in Figure 3.5, but these cases are easily identified. The experiment therefore demonstrates that LSTMs are viable for overtraining, but only when unsuited model parameters are chosen. Further analysis shows that most of the tested parameters follow the heuristic of Occam's razor; use the simple variant when possible. The LSTM layer size must be able to capture the time series complexity, but a bigger layer does not yield better results per se. Batch-size training can be used if a more general interpretation of the time series is needed, but if individual samples contain import information, Stochastic Gradient Descent (batch size of 1) should be used. Likewise, if the time series consists of a simple time independent oscillating pattern, a stateless network is suitable. However, in our case where no independent patterns can be extracted from the time series due to constricting temporal behaviour, a stateful model is better suited.

In terms of prediction strategy, our experiments showed that Multivariate prediction, or directly outputting the entire prediction horizon gave the best results of the three methods tested. However, when the prediction horizon becomes larger, accuracy falters on the final predictions. It seems that there is a limit to the size of the prediction horizon when using LSTM networks in our current topology.

It might be the case that a different output layer size allows the network to perform even better. The tested time series has an oscillating pattern that repeats every 24 hours, therefore it is possible that continuously predicting entire days, i.e.

24 hours simultaneously, combines the intuitivity of continuous prediction, while making it less sensitive to exactly fitting the periodicity of the time series. Of course, as mentioned in Taieb(2012)[22], each possible forecast strategy has its pros, cons, and trade-offs. Future research must show if another prediction strategy variant is more appropriate for gas flow regression analysis.

6.2 RNB Stations

The first part of this section was tasked with optimizing the first, encoding part of the final RNB network. With initial networks converging on the naive prediction heuristic of simply repeating the last input as output, we decided to mask the actual values of the input time series. By using an autoencoder, a neural network equivalent of Principal Component Analysis is performed on the time series, which results in an obscured encoding of the time series which still contains its characteristics.

The usage of a temporal autoencoder (using LSTM cells) turned out to be an improvement over the spatial autoencoder. This strengthens the assumption that the gas flow time series has compelling temporal characteristics, and that the temporal information that is present in the time series, can be more effectively modelled with a Temporal, LSTM autoencoder network. Regarding the fact that results further on display that the final error score of the encoder has a serious influence on the model's overall performance, the methodology of using a temporal encoder as primary layer in the regression task seems justified.

With regards to the complete regression model, the behaviour of the absolute error over time is also noticeable, as can be seen in Figure 4.12. The rise in error during the warmer periods in the year coincides with a certain change of gas flow behaviour when a heating threshold is approached. The difference in behaviour is visually displayed in Figure 4.13 and Figure 4.14 Exploratory models where this heating threshold was indicated to the model via an additional binary input feature (0 or 1), already showed a more stable absolute error over time, but also prevented the model from fitting to both behaviours. Instead the model displayed a behaviour that could be characterized as a generalized average between the two behaviours. Therefore, a viable improvement was to create more specific models, each trained on a different period of the year. A crude ensemble learning approach of training a separate model per season already resulted in a Test Error (RMSE) of 0.15, an improvement of 33%, compared tot the best performing RNB model, while also resulting in a more linear error progression over time. Further research is needed to fully explore the possible improvement, e.g. by using a well-tested ensemble learning method as negative correlation[28] or the Bayes Optimal Classifier technique[29].

6.3 International Network

A suitable first remark on this test case is the course removal of a large amount of data from the time series. All flows that ended before the end of 2017 were removed, but one can imagine that they contained a lot of information about certain behaviour while they were still operating. Also, using hourly values per flow might not have been the best data aggregation type. Especially since our interest was in the behaviour of individual shippers, aggregation on their portfolio might have

been better suited. However, by restricting the experiment to individual hourly flow rates, predictions could be directly translated to flows in the physical system, while aggregations would have led to the need for further interpretation.

As noted in the Results section, the three models learned a general approximation of the average gas flow per network point. However, contrary to the gas flow behaviour of RNB stations, the International gas flows behaved very irregular, with large trend breaks and volatility. The fact that our LSTM model design was created to fit flowing, regular and predictable time-series might therefore be an explanation for the results. International shipper behaviour can be very erratic and is based on stock prices and trade opportunities. The LSTM models showed that previous shipper behaviour contains insufficient information to accurately predict the future. A logical extension to this experiment is therefore the addition of extra information to the models. Supplying the models with gas stock prices might aid the models in interpreting sudden changes in gas flow behaviour. Furthermore, the irregularity and volatility might also be an indication that LSTM networks are not the most suitable technique for regression on International gas flows. LSTM networks are suitable for capturing for long-term temporal effects, and are impeded by sudden trend breaks in time series, as was visible in the RNB use case where summer/winter trend breaks had a negative effect on overall performance. Of course there are certain aspects in shipper behaviour that are constant over time, since most of them trade in terms of portfolios and have to satisfy the gas needs of their customers. But the underlying actions that result in satisfying the needs of their customers is clearly not static, whilst this is exactly the behaviour (individual gas flows) that we have tried to predict using our LSTM models.

A future improvement of our regression task can therefore be a different aggregation of the data. By regressing on a less volatile time series, for example on aggregated shipper portfolios, the LSTM models might be able to find the temporal dependencies/influences and result in a better regression model. Contrastingly, regression on individual gas flows might be better suited for neural networks which are less time-sensitive, such as Multi Layer Perceptrons.

6.4 Final Conclusion

In conclusion we can state that LSTMs are suitable for time series regression, as they are able to properly capture the data characteristics. They achieve this performance regardless of possible time series varieties such as differing seasonal trends or multiple oscillating patterns, as long as sensible parameter settings are used. This result is contrary to a general impression that LSTMs are easily overtrained or get trapped in local minima[19]. Such a sub-optimal performance can however occur, when when the model's topology is unable to capture a complex time series, or when the time series itself provides too little information for a proper regression model, as was the case with the International use case.

Furthermore, the usage of a temporal autoencoder setup proved to be significantly better at encoding time series, compared to the 'normal', spatial autoencoder. While more computation time was needed, the resulting losses were significantly lower and more stable.

This improvement persists in the complete regression model, as the autoencoder LSTM model outperforms both the naive prediction and the ARIMA model on the RNB use case. An exploratory collection of specialized autoencoder LSTM models even further improved the root mean squared error with 33%.

Bibliography

[1] S. Makridakis and M. Hibon, "Arma models and the box–jenkins methodology," *Journal of Forecasting*, vol. 16, no. 3, pp. 147–163, 1997.

[2] F. ROSENBLATT, "The perceptron: A probabilistic model for information storage and organization in the brain1," *Psychological Review*, vol. 65, no. 6, p. 1958,

[3] M. Minsky and S. Papert, "Perceptrons: An introduction to computational geometry (expanded edition)," 1969.

[4] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[5] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[6] G. Hinton, N. Srivastava, and K. Swersky, *Neural networks for machine learning lecture 6a overview of mini-batch gradient descent*, 2012.

[7] J.-F. Bonnans, J. C. Gilbert, C. Lemaréchal, and C. A. Sagastizábal, *Numerical optimization: Theoretical and practical aspects*. Springer Science & Business Media, 2006.

[8] M. T. Hagan, H. B. Demuth, M. H. Beale, *et al.*, *Neural network design*. Pws Pub. Boston, 1996, vol. 20.

[9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[10] R. J. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent networks and their computational complexity," *Backpropagation: Theory, architectures, and applications*, vol. 1, pp. 433–486, 1995.

[11] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[12] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with lstm recurrent networks," *Journal of machine learning research*, vol. 3, no. Aug, pp. 115–143, 2002.

[13] P. J. Brockwell and R. A. Davis, *Introduction to time series and forecasting*. springer, 2016.

[14] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: Forecasting and control*. John Wiley & Sons, 2015.

[15] D. C. Park, M. El-Sharkawi, R. Marks, L. Atlas, and M. Damborg, "Electric load forecasting using an artificial neural network," *IEEE transactions on Power Systems*, vol. 6, no. 2, pp. 442–449, 1991.

[16] M. Espinoza, J. A. Suykens, R. Belmans, and B. De Moor, "Electric load forecasting," *IEEE Control Systems*, vol. 27, no. 5, pp. 43–57, 2007.

[17] C. García-Ascanio and C. Maté, "Electric power demand forecasting using interval time series: A comparison between var and imlp," *Energy Policy*, vol. 38, no. 2, pp. 715–725, 2010.

[18] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[19] M. Jiménez-Guarneros, P. Gómez-Gil, R. Fonseca-Delgado, M. Ramírez-Cortés, and V. Alarcón-Aquino, "Long-term prediction of a sine function using a lstm neural network," in *Nature-Inspired Design of Hybrid Intelligent Systems*, Springer, 2017, pp. 159–173.

[20] C. Hamzaçebi, D. Akay, and F. Kutay, "Comparison of direct and iterative artificial neural network forecast approaches in multi-periodic time series forecasting," *Expert Systems with Applications*, vol. 36, no. 2, pp. 3839–3844, 2009.

[21] D. R. Cox, "Prediction by exponentially weighted moving averages and related methods," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 414–422, 1961.

[22] S. B. Taieb, G. Bontempi, A. F. Atiya, and A. Sorjamaa, "A review and comparison of strategies for multi-step ahead time series forecasting based on the nn5 forecasting competition," *Expert systems with applications*, vol. 39, no. 8, pp. 7067–7083, 2012.

[23] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[24] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming auto-encoders," in *International Conference on Artificial Neural Networks*, Springer, 2011, pp. 44–51.

[25] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, Ieee, vol. 2, 1999, pp. 1150–1157.

[26] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, IEEE, 2016, pp. 512–519.

[27] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction," *Artificial Neural Networks and Machine Learning–ICANN 2011*, pp. 52–59, 2011.

[28] Y. Liu and X. Yao, "Ensemble learning via negative correlation," *Neural networks*, vol. 12, no. 10, pp. 1399–1404, 1999.

[29] T. M. Mitchell *et al.*, "Machine learning. 1997," *Burr Ridge, IL: McGraw Hill*, vol. 45, no. 37, pp. 870–877, 1997.

Appendix A

Used soft-/hardware

All programming was done in Python, version 3.5.3. Analysis of the results was done with a combination of the aforementioned Python and R, version 3.4.1. The neural networks were build using a combination of the Theano(0.9.0) and Keras(2.0.4) libraries.

Experiments were done on two desktop computers, the first with an Intel i7-2600 CPU, Quadro 600 GPU and 8GiB of RAM. The second computer consisted of an Intel i7-6700, Quadro K2200 and 16 GiB of RAM.