# Seminar: Topics in Machine Learning and Applied Econometrics

Kevin Stefan Meyer, B.Sc. Economics
Advisor: Dr. Arne Warnke

Tuesday 9th February, 2021

**Abstract**

This paper will display the methodology and approach applied in mastering the forecasting challenge, exhibiting an examination of a time series of the water level of the German river Rhine. Standard techniques like the econometric standard approach of Box-Jenkins and ARIMA models shall function as a baseline and be evaluated against more sophisticated machine learning methods. In this paper we're focusing on the CNN and LTSM models, where we're going to show that they don't outperform other regression models on this problem. Only when taken into account multiple variables, we can achieve an tremendous increase in accuracy. We always turn to machine learning methods when the classical methods show the possibility to be improved upon. The ability to model unknown nonlinear relationships in time series data and some methods may result in better performance when working with non-stationary observations or some mixture of stationary and non-stationary views of the problem. Furthermore several additional predictors, the general approach towards model selection, their data gathering and processing parts will be explained.

## 1 First-step analysis

This section starts by looking at the plot of the Rhine river level time series. First of all one can already see, through movements around the mean and approximately constant variance, that the underlying stochastic process might be mean-reverting and thus be stationary. This will later be tested with the Augmented-Dickey-Fuller test [1].
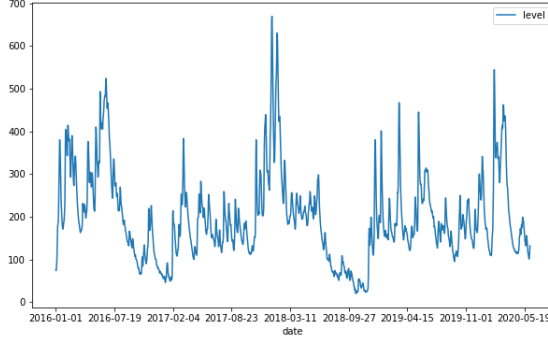
Figure 1: Our basic dataset: Level of the Rhine river from 2016 to mid-2020.

When looking at the bigger machine learning picture, time series analysis has special properties that make it different from traditional classification and regression perspectives. The temporal structure adds an order to the observations, where classical models have to be adjusted to accurately reflect that fact. Aforementioned difference will become especially clear in Section 4, where we're applying methods like MLPs, CNNs as well as LSTMs. While MLP's and CNN's do not handle a specific internal state to account for the temporal structure of time series, LSTMs have been explicitly designed for that matter and thus present a state-of-the-art methodology to handle these specific problems. Nevertheless, all of the aforementioned methodologies depend on the specifics of the time series under investigation and it shall be proven that in our case, the multivariate LSTM model described in Chapter 5 clearly outperforms all other approaches. Another important general approach to mention is the fact that our goal in mind is to outperform on a seven day forecast, which in practice comes down to us testing models with smaller test sets than in a setting with a different goal in mind (i.e. forecasting two weeks, three weeks or even larger time periods). It shall be brought up that all the models perform equally on larger test sets than the ones discussed here, while for ease of reproducability and fitting the correct model for the forecasting challenge this amount has been narrowed down to seven days.

## 1.1   Check for Autocorrelation

In a first step, we look at possible autocorrelation amongst past values of the time series itself. Achieving that, we can plot the correlation coeffi-

cient for each lag variable. This is done in order to get a first grasp at the later configuration of ARIMA models and which lag variables might be most suitable to capture the relationship between the observation and its historic values over time. For this, pandas built-in plot function called the autocorrelation_plot() [2] was used.
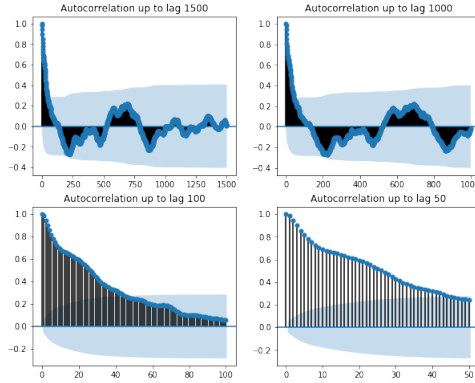


Figure 2: Plots of autocorrelations up to different lag values.

On its x-axis the plot depicts the lag numbers, whereas the respective correlation coefficient value is depicted on the y-axis. Similarly, 95% confidence interval cones are provided in order to estimate statistical significance. In this plot, through its very nature of including correlations with large lag values, we can see interesting patterns. Namely we can see that there is substantial negative correlation to the 200 day lag, which from a domain expertise standpoint tells us that this can express the cyclical, seasonal relationship between summer/winter. Furthermore, in the lower row plots we can see that there is significant correlation until up to a lag of about 50.

## 1.2 Check for stationarity

Next, we examine our time series on stationarity. The rationale for this step lies in the required differencing steps to enhance our modeling capabilities in the ARIMA models. Observations in a stationary time series are not dependent on time in the sense of a constant variance and mean over different periods. Furthermore, stationary time series don't exhibit trends or seasonal effects, which still has to be proven for our time series. We're going to test this initial guess of non-stationarity by the Augmented Dickey-Fuller Test [3]. This is a type of unit root test, which determines how strong a time series is influenced by trends and a shift of mean and variance over

time [4]. It achieves that examination by constructing an autoregressive model and optimizing an information criterion (like AIC or BIC) across multiple different lag values. Formally, the null hypothesis of the test is that the time series can be represented by a unit root, which is equivalent to a non-stationary, time-dependent structure. The alternate hypothesis (rejecting the null hypothesis) claims that the time series is stationary.

To formalize the following statistical tests, clear definitions shall be provided [5]:

- Null Hypothesis (H0): Time-dependent structure. If one fails to reject, it suggests the time series has a unit root, meaning it is non-stationary and it exhibits a time dependent structure.

  vs.

- Alternate Hypothesis (H1): If the null hypothesis is rejected, the test suggests the time series does not have a unit root, meaning it is stationary and it does exhibit have time-dependent structure.

Below the ADF-statistic for our time series is depicted.

```
ADF Statistic: -4.199761
p-value: 0.000660
Critical Values:
        1%: -3.434
        5%: -2.863
        10%: -2.568
```

Figure 3: Augmented Dickey-Fuller Test Statistic with corresponding critical values.

Running the example prints the test statistic value of around -4. The intuition for the comparative statics of the statistic is that the more negative it is, the more profound is the rejection of the null hypothesis, implying stationarity. The look-up table of the test statistic as depicted in Dickey's seminal paper [3] implies statistic value of -4.199761 is less than the value of -3.434 at the level of 1%. As a conclusion we can reject the null hypothesis with a significance level of less than 1%. Going back to the enumeration of the null and alternate hypothesis, the test statistic suggests that the process has no unit root, meaning that the time series lacks time-dependent structure and is stationary.

Our first glimpse and naive tests indicated the time series to be non-stationary, the ADF-test claims the contrary - we're going to stick to the technical analysis, but keep our intuition in mind when applying ARIMA models to the time series.

# 2 Box-Jenkins Methodology and naive baseline models.

The Box-Jenkins [6] framework denotes a systematic approach to build predictive forecasting models. Before following this approach throughout the next chapters, a naive baseline model will be established first in order to gain a metric to subsequently outperform. Furthermore, the book by Brownlee [5] provides an overview of a practical approach towards the problem at hand.

## 2.1 Baseline models: naive and average methods.

To provide a means of comparison of more advanced forecasting models, naive, i.e. persisting the last observation and average, i.e. persisting an average of different magnitudes of past observations were applied. For this, a test harness was developed by splitting the observations into training and test sets, with only the last 7 observations in the dataset assigned to the test set. Simultaneously, a walk-forward validation model was applied to gradually include the newest observations, meaning that we are not making a 7-day forecast, but seven 1-day forecasts. This previous approach was tested for all natural numbers in the range up to 365 days (which corresponds to the initial hypothesis of a yearly cycle).
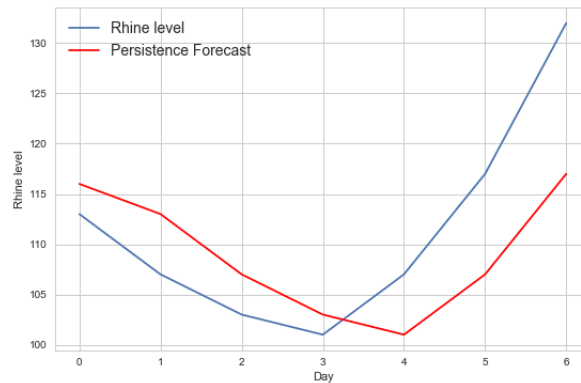


Figure 4: Plot of the persistence forecast (red) vs. actual values (blue).

The best performing persistence model has a RMSE of 7.801, which will in the following be the metric to outperform.

For comparison, if we extend our training set to 10% of the overall data[1], we get an RMSE of 20.535. This will also be evaluated against more sophisticated methods, especially regarding the general fit and not running the danger of overfitting our models by only testing it on the last seven days, for which we want to maximize performance in order to increase the probability of mastering the forecasting challenge.
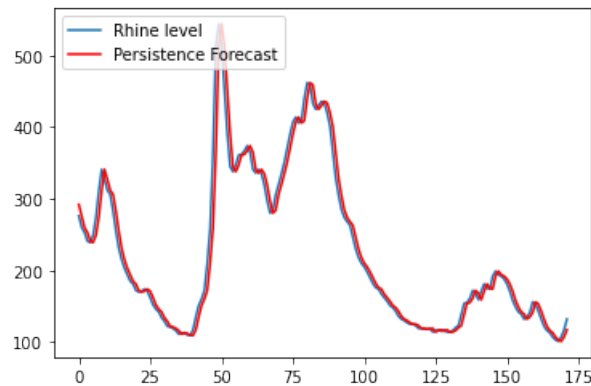


Figure 5: Plot of the persistence forecast (red) vs. actual values (blue) at a larger time frame (172 values, which corresponds to 10% of the dataset.

The next step of the Box-Jenkins approach consists of looking at Autocorrelation functions (ACF) and Partial autocorrelation functions (PACF) in order to get ahold of suitable values for the ARIMA model parameters.

## 2.2 Autocorrelation

Like already mentioned in the introductory chapter, autocorrelation plots indicate suspected correlations of a time series with its lagged values. For our method of grid searching the most suitable model by brute-forcing different parameter configurations, this step is extremly important to limit our search space for hyperparameters to a considerable space.

```
from pandas import read_csv
from matplotlib import pyplot
```

[1]We won't follow the path of doing a two-third-one-third train-test split, because we only have 1720 values available. With larger datasets, that would be the better choice.

```
from statsmodels.graphics.tsaplots import plot_acf
series = read_csv('basicDataset.csv', header=0, index_col=0)
plot_acf(series)
pyplot.show()
```

Running the code yields a finer granulated picture of the results already discussed in the first section (Fig. 6 is configured with a 95% confidence interval).
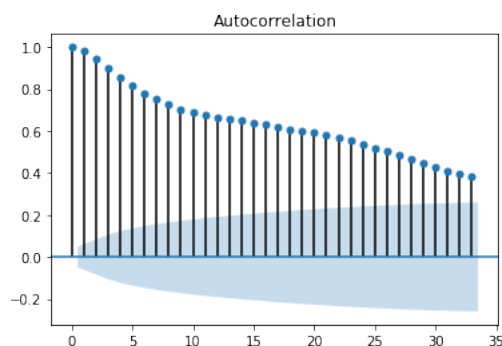


Figure 6: Autocorrelation of the Rhine river dataset up to a lag of 35 days.

If we include lags bigger than the ones depicted in the figure above (like in Fig. 2), we can deduce other relevant patterns (see discussion there).

## 2.3   Partial Autocorrelation

Partial autocorrelations in contrast distill the influence of lagged values by removing the effects of intervening observations, thus providing patterns for configuring the ARIMA parameters.

```
from pandas import read_csv
from matplotlib import pyplot
from statsmodels.graphics.tsaplots import plot_pacf
series = read_csv('basicDataset.csv', header=0, index_col=0)
plot_pacf(series, lags=50)
pyplot.show()
```

Running the example creates a 2D plot of the partial autocorrelation for the first 35 lags.
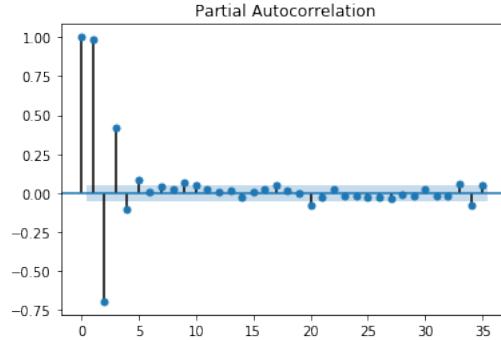
7

Figure 7: Partial Autocorrelation of the Rhine river dataset up to a lag of 35 days.

Fig. 8 depicts a taxonomy (see [7] for a more detailed description) for model selection based on different behaviours of the problem-dependent ACF and PACF plots. The classification based on this taxonomy shall be shortly summarized in order to transfer it to our problem:

1. If the ACF plot decreases gradually and the PACF plot has a sharp drop after p lags, this indicated the underlying process being an AR model.

2. If the ACF plot has a sharp drop after q lags while the PACF exhibits a gradual decreasing trend, this is an indication of the underlying process being a MA model.

3. If both the ACF as well as the PACF plot gradually decrease, an ARMA process should be used to model the problem at hand.

| Terms | ACF | PACF |
|-------|-----|------|
| AR | Geometric | p significant lags |
| MA | q significant lags | Geometric |
| ARMA | Geometric | Geometric |

Figure 8: Taxonomy of modelling decisions based on ACF & PACF plots [7].

Following the taxonomy we have a strong inclination to assume an AR(4) process (ACF plot showing a geometric decrease while PACF has four significant lags). This will define our baseline model against which we're going to evaluate our sophisticated machine learning methods, i.e. the LSTM and CNN model approaches.

# 3 ARIMA models

Recapitulating and building the intuition behind autoregressive models in general, next there shall be depicted the formal mathematical model behind ARMA models. An ARMA model [1] is specified as

$$y_i = c + \sum_{i=1}^{p} \varphi_i y_{t-i} + \sum_{i=1}^{p} \theta_i \varepsilon_{t-i}, \tag{1}$$

which depicts the fact that it takes both lag values of the variable itself as well as previous error terms into account, weighing them according to the estimated influence on the current value. Following our intuition of examining the ACF and PACF plots, we fit an AR(4) model to our dataset and examine the results. When once again doing a 90-10 split for the respective train and test percentage, we get an RMSE of 12.815. This outperforms the naive approach of persisting values and can thus be considered skillful.

Below is the plot using only a seven day test window, yielding an error rate of 4.258.
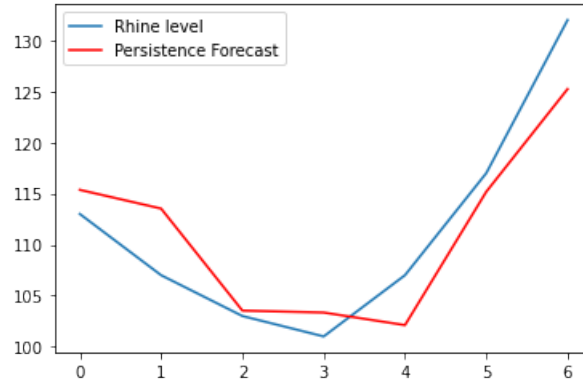


Figure 9: AR(4) model, plotted against the realized Rhine water level.

As we can see, we can indeed improve upon our naive forecast model. Grid search regarding other lag and differencing parameters led to no improvement of performance, so in this model class (with only considering our univariate time series) we can consider it the most skillful model.

Next, we're going to go deeper on the seasonal structure of our data. The intuition that the Rhine level follows a periodical pattern leads directly to the application of the SARIMA model (which has proven not to yield

better results than the ARIMA model; thus, following the approach of better readibility, a grid search of various SARIMA hyperparameters has been outsourced to the Jupyter notebook file). As just stated, we're focusing on the ARIMA model in this paper instead. The results for Grid Searching the ARIMA parameters is depicted in Fig. 11. As we see, the only model that slightly outperforms the AR(4) model is the ARIMA(4, 0, 1) configuration, again convincing us, analogously to our findings from the Augmented Dickey-Fuller test, that we don't have to take differencing into account[2].

```
ARIMA(0, 0, 0) RMSE=83.056
ARIMA(0, 0, 1) RMSE=43.204
ARIMA(0, 1, 0) RMSE=20.580
ARIMA(0, 1, 1) RMSE=14.318
ARIMA(0, 1, 2) RMSE=13.152
ARIMA(0, 2, 0) RMSE=16.210
ARIMA(0, 2, 1) RMSE=15.250
ARIMA(0, 2, 2) RMSE=14.773
ARIMA(1, 0, 0) RMSE=20.475
ARIMA(1, 0, 2) RMSE=12.980
ARIMA(1, 1, 0) RMSE=14.915
ARIMA(1, 1, 1) RMSE=13.215
ARIMA(1, 1, 2) RMSE=13.089
ARIMA(1, 2, 0) RMSE=15.747
ARIMA(1, 2, 1) RMSE=14.936
ARIMA(2, 0, 0) RMSE=14.470
ARIMA(2, 0, 1) RMSE=12.940
ARIMA(2, 0, 2) RMSE=12.850
ARIMA(2, 1, 0) RMSE=13.052
ARIMA(2, 1, 1) RMSE=12.996
ARIMA(2, 1, 2) RMSE=12.909
ARIMA(2, 2, 0) RMSE=14.677
ARIMA(2, 2, 1) RMSE=13.069
ARIMA(4, 0, 0) RMSE=12.837
ARIMA(4, 0, 1) RMSE=12.801
ARIMA(4, 1, 0) RMSE=12.944
ARIMA(4, 1, 1) RMSE=12.913
ARIMA(4, 2, 0) RMSE=14.326
```

Figure 10: RMSE's of different configurations of the ARIMA model.

---

[2]For model selection purposes, here we tried a 2/3-1/3 split, yielding the same ranking as a 90-10 split.

# 4  MLP, CNN and LSTM approaches

Now we're going to focus on different "pure"[3] machine learning methods, which present the state-of-the-art approach and, due to there mathematical properties of taking non-linearity into account, often outperform standard econometric models. This section makes use of the methodology of Jason Brownlee, especially his treatise "Deep Learning for Time Series Forecasting" [9].
We begin our investigation by configuring Multilayer Perceptron (MLP) models [8]. An MLP is a feedforward neural network model that can be, similarly to the ARIMA methods used before, take multiple lag observations and using them as input features to be able to predict one or more time steps from those observations. The input parameters for our model are (this methodology follows the state-of-the-art in the industry [9], also see the official keras specification [10]):

- n_input: The number of lag observations to use as input.

- n_nodes: The number of nodes to use in the hidden layer.

- n_epochs: The number of times to train the model on the entire training dataset.

- n_batch: The number of samples within an epoch after which weights are updated.

After trying different combinations for the number of inputs, the specification of the tuple of [36, 50, 100, 100] proved to be the most suitable one. Unintuitively, after running the algorithm for several times, 36 lags seemed to be more accurate than simpler configurations. After fitting the model and repeatedly evaluating it to get in parts rid of the inherent randomness, the algorithm yielded the following result.

---

[3]Depending on the way of looking at the terminology; "pure" here means models that make use of more sophisticated internal structures, like neural networks or internal state machines.
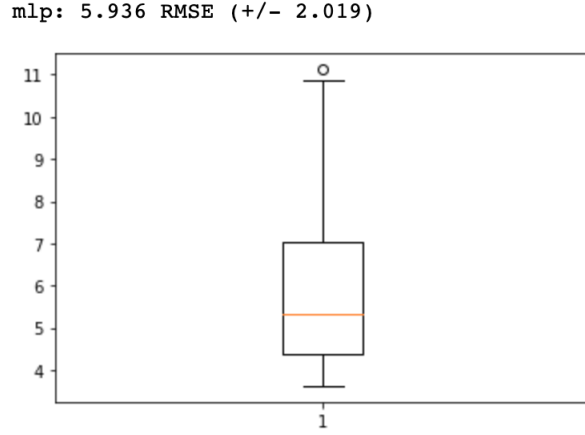
mlp: 5.936 RMSE (+/- 2.019)



Figure 11: Box and Whisker Plot of Multilayer Perceptron RMSE Forecasting the Rhine Level.

Next, we're going to explore CNNs and LTSMs, as well as combinations of the two. CNNs have traditionally been developed for two-dimensional image data, although its usage can be expanded on time series forecasting poblems. In its simplest form, i.e. the one-dimensional CNN, the network is configured with a convolutional hidden layer that operates over a 1D sequence [11]. To stack another neural network layer on top of it and narrow down the most important elements of the output of the previous layer, it is conventional to take a pooling layer as the subsequent step. The last layer consists of a dense fully connected layer that interprets the features extracted by the convolutional part of the model ( [11] and [5]). One has to carefully construct the input to the network, because of the fact that the specification requires a specific input format, i.e. [samples, timesteps, features]. Since we only have one feature (namely the Rhine water level itself) this is straightforward using the built-in numpy functionality. For our problem, I chose a convolutional layer with 256 filter maps and a kernel size of 4, followed by a max pooling layer and a dense layer to interpret the input features. The model was fit using the efficient Adam version of stochastic gradient descent and optimized using the mean squared error loss function, for the sake of comparability to the models tested above. The performance is depicted in Fig. 13, where we can see no clear improvement to the existing ARIMA model.
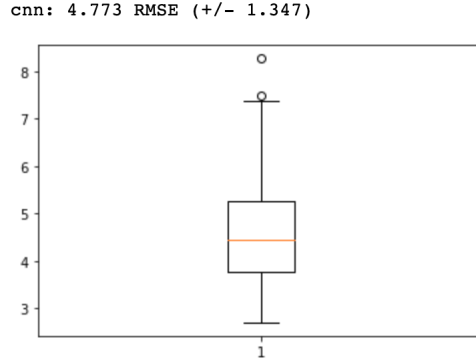
12

cnn: 4.773 RMSE (+/- 1.347)



Figure 12: Box and Whisker Plot of the CNN RMSE Forecasting the Rhine Level.

A CNN model can also be used in a hybrid model with an LSTM backend. In a first step the CNN inteprets subsequences of the input that are then passed on as a sequence to an LSTM model to further process the latter. This hybrid model is called a CNN-LSTM. The CNN-LSTM approach yielded an RMSE of 5.097, with a standard deviation of +/- 1.366. Compared to our AR-approach earlier we didn't improve with that approach.

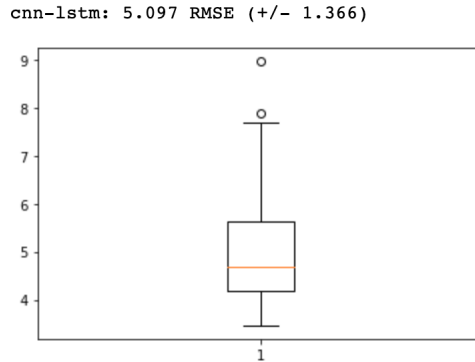cnn-lstm: 5.097 RMSE (+/- 1.366)



Figure 13: Box and Whisker Plot of the CNN-LSTM RMSE Forecasting the Rhine Level.

In our final model of this section, we're going to test a standalone LSTM approach. The LSTM, initially proposed by [12], can be seen as an alternative to standard neural networks. Due to its nature it can be applied to time series in a straightforward way. The hyperparameters for the LSTM model are the same as in the MLP model; they are:

- n_input: The number of prior inputs to use as input for the model (e.g. 7 days).

- n_nodes: The number of nodes to use in the hidden layer (e.g. 50).

- n_epochs: The number of training epochs (e.g. 1000).

- n_batch: The number of samples to include in each minibatch (e.g. 32).

- n_diff: The difference order (e.g. 0 or 180).

The average RMSE of the LSTM model is rather disappointing, leading to the conclusion that an additional predictor is necessary.
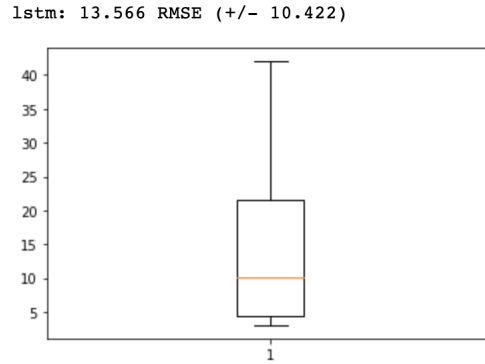


Figure 14: Box and Whisker Plot of the LSTM RMSE Forecasting the Rhine Level.

## 5    Multivariate Forecasting

Our approach now is to use other data than the plain vanilla time series of the Rhine level to test if further enhancement of influential variables increases our capability of predicting. The decision process involved taking into account the Rhine level of stations tracking the precipitation level along the Rhine river. To accomplish this, the first step was in marking over 400 coordinates along the Rhine via the Google Maps API and fetch stations located near said coordinates (to be exact, the distance had to be strictly less than 10km). To calculate the distance, the Haversine formula [13] was used. The haversine formula determines the great-circle distance between two points on a sphere given their longitudes and latitudes, which makes

it perfect for our purpose. The next step was in getting data from the
"Deutsche Wetterdienst" regarding historical and recent precepitation data
[14]. We achieved this by using the Keras/TensorFlow Libraries and their
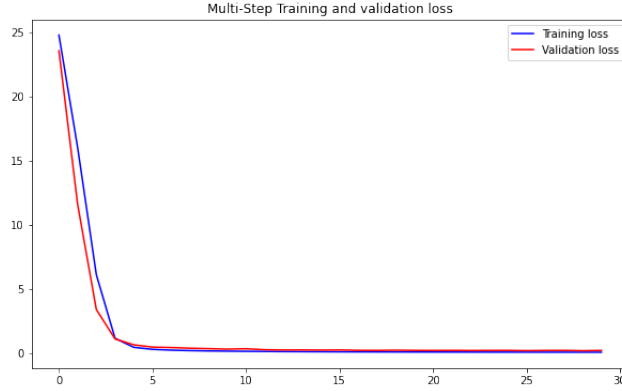implementation of LSTM networks.



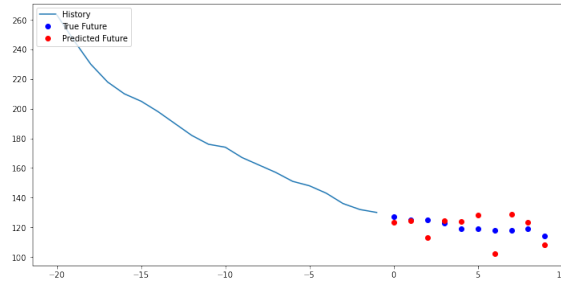Figure 15: Training and validation loss for our training and validation set.



Figure 16: Plot of the forecast against the last seven true Rhine level values.

The additional predictor was thus constructed by taking into account
the amount of precipitation in the proximity of the fetching stations, which
should get clear in the additional Jupyter notebook. We tried different
number of neurons for the following network, 32 and 16 are the most suitable
ones, whereas a greater number or neurons has the tendency to overfitting.
The results of this approach were impressive: after a training of 4 epochs, the
MSE was around 1.1407, which clearly outperforms all previous methods.
One can critize this low error by stating the guess of an overfitted model,
but since our challenge was to predict a seven day forecast, this fact was

15

materialized in our thought process. Further increasing the accuracy would definitely lead to the model just memorizing the time series, depicted in the fact that the error rate quickly converges to zero after the fourth epoch. The final forecasts for our challenge are listed below:

|            | Forecasts  |
|------------|------------|
| 2020-06-08 | 132.549759 |
| 2020-06-09 | 127.396751 |
| 2020-06-10 | 118.665489 |
| 2020-06-11 | 122.375198 |
| 2020-06-12 | 128.689163 |
| 2020-06-13 | 120.729950 |
| 2020-06-14 | 108.064194 |
| 2020-06-15 | 137.830338 |
| 2020-06-16 | 116.712555 |
| 2020-06-17 | 124.005730 |

# 6  Conclusion

In this seminar we examined several methods, beginning by standard time series analysis techniques, i.e. the ARIMA model. We established a baseline forecast and proved, that going the path of an univariate forecast, these methods barely outperform naive models. We proceeded to investigate modern architectures like MLP, CNN's, LSTM and hybrid approaches, where our result showed that a multivariate version can significantly improve the model performance. Further investigations regarding more advanced grid searches and the carefully selected inclusion of additional parameters could, given the time and resources, again improve the forecasting performance, especially regarding forecasting intervals longer than seven days and incorporating multiple steps into one forecast.

# References

[1] Stock J, Watson MW (2003). Introduction to Econometrics. New York: Prentice Hall.

[2] http://pandas.pydata.org/pandas-docs/stable/visualization.html#autocorrelation-plot

[3] Cheung, Y.-W., & Lai, K. S. (1995). Lag order and critical values of the augmented Dickey–Fuller test. Journal of Business & Economic Statistics, 13(3), 277–280.

[4] Hamilton, J. D. (1994), Time Series Analysis, Princeton University Press.

[5] Jason Brownlee (2020), Time Series Forecasting with Python.

[6] George Edward Pelham Box and Gwilym Jenkins (1990), Time Series Analysis, Forecasting and Control. Holden-Day, Inc., USA.

[7] https://towardsdatascience.com/identifying-ar-and-ma-terms-using-acf-and-pacf-plots-in-time-series-forecasting-ccb9fd073db8

[8] Almeida, L.B (1997), Multilayer perceptrons. In Handbook of Neural Computation, IOP Publishing Ltd and Oxford University Press.

[9] Jason Brownlee (2020), Deep Learning for Time Series Forecasting.

[10] `https://keras.io/guides/sequential_model/`

[11] LeCun, Y., Bengio, Y. and Hinton, G. (2015). Deep learning. Nature, 521(7553), pp.436-444.

[12] Sepp Hochreiter, Jürgen Schmidhuber (1997), Long short-term memory In: Neural Computation (journal), vol. 9, issue 8, S. 1735–1780.

[13] R. W. Sinnott (1984), Virtues of the Haversine. Sky and Telescope 68 (2), 159.

[14] `https://opendata.dwd.de/climate_environment/CDC/observations_germany/climate/daily/kl/recent/`