Weather forecast with Deep Learning

Paul MARTIN
ENPC
paul.martin@eleves.enpc.fr

Samuel DIAI
ENPC
samuel.diai@eleves.enpc.fr

Abstract

Despite there exists several statistical models that have been developed to tackle multivariate time series forecasting, deep Learning and Recurrent Neural Network seems more adequate when it comes to capture long-term dependencies. We will focus on how to predict the temperature by comparing state of the art model : a dual attention-based RNN with simpler models in a robust benchmark.

1. Introduction

Times series may be encountered in large range of area, such as finance, biology or physical systems. Forecasting appears as one of the main challenge when it comes to study temporal phenomenons, in this project, we investigate and compare several methods used for forecasting by applying them to temperature prediction. Indeed, weather forecast is one of the most considered estimation since we all watch weather reports. Having an accurate prediction is not only useful for choosing clothes but also to prevent meteorological risks. For example, airline companies rely on weather report to decide if planes can take off, or electricity providers use temperature forecast so as to know if the electricity consumption may increase because of a cold snap. Time series forecast is a challenging topic because of the temporal structure of data which makes the experiments more complex especially if we face multivariate time series. Beside time series are very domain specific, that is to say the expert knowledge can not be easily transferred to another field of application because of different patterns, seasonality or trends.

Weather has a wide range of key indicators such as sunshine, rainfall, wind, pressure or temperature, in this work we will focus on temperature forecasting with multivariate data, we consider the feature time series as 'driving series' namely time series influencing the temperature. We use the dataset *jena climate 2009-2016*, containing 14 features such as pressure or humidity sample every 10 minutes from 2009 to 2016 by Max Planck Institute of bio-geochemistry at only one place. We aim at providing a robust benchmark of the most recent deep learning method such as Dual-Attention LSTM with more traditional architecture such as LSTM or 1D convolutions. We compare this method with several metrics adapted to time series such as MASE or MSE and incorporate baseline methods to put in perspective the different models. We also investigate the effect of some hyper-parameters and test different prediction horizons.

2. Related Work

Time series forecasting remains a challenging topic with several approaches more or less successful. Time series have been studied through the prism of econometrics with models like ARMA or ARIMA studied in [5]. These quite old models may be used either to analyze or forecast a univariate time series, and are still widely used for forecasting. With the recent success of Deep Learning on various tasks (computer vision, natural language processing), more complex methods have emerged and try to use the wealth of information contained in multivariate data. In particular, nonlinear autoregressive exogenous model (NARX) models [12] can be used to deal with nonlinear relationships within the data and can use current and past information to predict future outcomes.

Moreover, neural networks architecture are improved very frequently and allow some models to be very suited for specific tasks. Some specific architectures such as Recurrent neural network (RNN) seems to be well suited to handle temporal structure [3]. Specifics RNN called long short term memory (LSTM) or gated recurrent unit (GRU) are widely used to benefit from the temporal structure of the data but solving the issue of long term dependencies, a problem that traditional RNNs suffer from. They use gated mechanism to avoid vanishing gradient problems [4]. Finally, more recent discoveries highlighted the power of a mechanism in neural networks called ”Attention” [1]. These mechanisms are widely used in NLP [11] and have showed very significant improvements in these fields. They allow the model to learn contextual information and the relationship between some features. The relationships can be temporal, spatial, positional, and attention mechanisms can be used in encoder-decoder structure like proposed by [8] for time series forecasting.

3. Problem Statement and models

3.1. Notations

In our case, we have a multivariate time series. Let's denote $X \in \mathbb{R}^{T \times n}$ with T the length of time series and n the number of features (21 in the experiments). $X_t = (X_t^1, ..., X_t^n)$ is the vector of features at time t and $X^i = (X_1^i, ..., X_T^i)$ is univariate time series of the feature i. We denote $y = (y_1, ..., y_T)$ the target time series, the temperature in the experiments. We define the horizon of prediction τ. The goal of the problem is to learn a function F parametrized by weights θ :

$$\hat{y_{T+\tau}} = F_\theta(X_1, ..., X_T, y_1, ..., y_T)$$

The function F_θ is modeled by a neural network allowing a non linear and recurrent interactions. This function is learned by gradient back propagation on samples minibatch.

3.2. Models

3.2.1 Baseline Models

In order to provide good insights about the results, we decided to include very simple models which allow us to compare the results. Indeed the metric of a state of the art model may be hard to interpret, thus having a baseline permits to analyze more accurately the performance of the models.

The first baseline model is a very simple but efficient heuristic : we can use the last known value. If the prediction horizon is quite short, the approach may be quite challenging to overcome.

The second method we use is based on rolling average, we base the prediction on the average of the last known temperatures on a specific window. Because of the seasonality, the performance of this procedure may be poor because it would de-seasonalized the time series if the window is too large.

We also include a simpler recurrent architecture which is a simple LSTM architecture with different number of hidden size or different number of layers. This model is interesting because we can see if we obtain an improvement by adding an attention mechanism in a recurrent network.

3.2.2 Convolutional Network

Convolutional networks are mainly used to handle images but [10] proposed a causal convolution models to generate audio wave forms. The architecture is adapted to time series and especially to capture very long term dependencies which are often needed in audio because of the high sampling rate. We include a model inspired by the latter to see if it could perform well in forecasting.

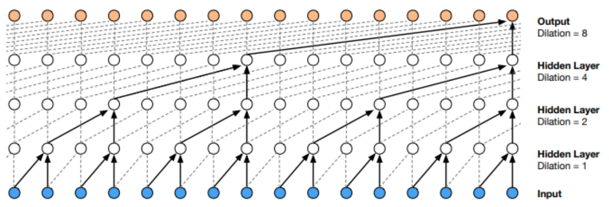

Figure 1. Stack of dilated causal convolutions

Because of the temporal structure, we have to make the convolutions causal i.e. make sure only past data are used to predict the future. Besides, we use dilated convolution which apply the filter over a larger area of the signal by skipping some inputs with a given step. By using a network as presented by 1, the kernel size is equal to 2, but the dilation is increasing at each layer therefore the length of the receptive field (the number of past time step used to compute the value of a neuron on the last layer) grows exponentially with the number of layer. By using convolution, the network can be highly parallelized reducing the training time but also using much more past data as input. To forecast we keep only the last neurons of each channel then doing the prediction thanks to a multi-layer perceptron. In our experiments, the receptive field of the model is fixed to 32.

3.2.3 Dual-Stage Attention-Based RNN

The main goal of this project is to analyze a recurrent neural network with a dual attention mechanism (DA-RNN) by including the results in a benchmark. The DA-RNN has an encoder-decoder architecture. The first attention layer selects relevant features in order to encode the time series and the second attention layer focus on temporal importance in the decoder part. This architecture proposed by [8] enables to capture long term dependencies and to learn both the relevance of the driving series and the past time step. The recurrent neural network is a LSTM proposed by [6], defines a cell state c_t and a hidden state h_t such that :

$$h_t = g_1(x_t, c_t, h_{t-1})$$
$$c_t = g_2(x_t, c_{t-1}, h_{t-1})$$

With g_1 and g_2 defined in [6]. We encode only the driving series x_t^k for $t \in [\![1; T]\!]$ with a multi-layer perceptron defining the unormalized weights e_t^k that can be trained jointly with the RNN.

$$e_t^k = V_e^T \tanh(W_e[h_{t-1}, c_{t-1}] + U_e x_k)$$

With $W_e \in \mathbb{R}^{T \times 2\text{n\_hidden}}$, $U_e \in \mathbb{R}^{T \times T}$ and $V_e \in \mathbb{R}^T$. These linear layers also have a bias which is not precised for

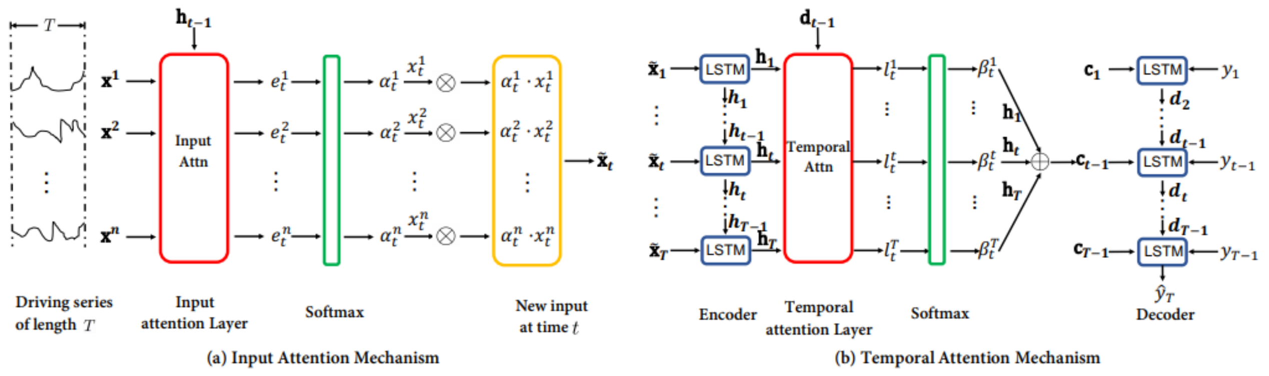

Figure 2. Dual-Stage Attention-Based RNN

clarity. Then by using a softmax, we obtain the normalized attention weights :

$$\alpha_t^k = \frac{\exp(e_t^k)}{\sum_{j=1}^K \exp(e_t^j)}$$

Then the input is encoded :

$$\tilde{x_t} = (\alpha_t^1 x_t^1, ..., \alpha_t^K x_t^K)$$

We allow the model to learn which feature is relevant by weighting them them at each time step. This context input is then used as input of the recurrent layer of the encoder :

$$h_t = g_1(\tilde{x_t}, c_t, h_{t-1})$$
$$c_t = g_2(\tilde{x_t}, c_{t-1}, h_{t-1})$$

The decoder part of the network which takes the hidden states of the encoder as input allow to decode the learned representation and its attention mechanism focused rather on the temporal aspect contrary to the encoder layers. The decoder is also a LSTM which has a hidden states h_t^d and a cell state c_t^d. The temporal attention weights are also learned through a multi-layer perceptron :

$$l_t^i = V_d^T \tanh(W_d[h_{t-1}^d, c_{t-1}^d] + U_d h_i)$$

With $V_d \in \mathbb{R}^{n\_hidden}$, $W_d \in \mathbb{R}^{n\_hidden \times n\_hidden}$, these linear layers also have a bias which is not mentioned in the equations. We always assume that the encoder and decoder have the same number of hidden size. Then we define the normalized weights and the weighted input of the decoder :

$$\beta_t^i = \frac{\exp(l_t^i)}{\sum_{j=1}^K \exp(l_t^j)}$$

$$c_t = \sum_{i=1}^T \beta_t^i h_i$$

We only encode the driving time series, but the past of the target time series (the temperature) also contains useful information therefore we concatenate the output of the encoder-decoder with the target time series to have an input of the decoder with all the information required.

$$\tilde{y_{t-1}} = \tilde{w}^T[y_{t-1}, c_{t-1}]$$

We use $\tilde{y_{t-1}}$ to update the cell state and hidden state of the decoder's LSTM then we do the forecasting with a multilayer perceptron taking as input the last hidden and cell state of the decoder. These last weights contain all the previous information.

$$\hat{y_{T+\tau}} = W_f[h_T^d, c_T] + b$$

The global architecture of the dual-stage attention based RNN is represented in 2. The attention layers allow to interpret the model which is useful when we study multivariate time series with a large number of features which may be correlated between them. Indeed the attention weights quantify how much a feature is important and how a much a time step is useful in order to forecast the temperature.

4. Methodology

4.1. Dataset

For this project, we used the dataset Jena climate 2009-2016 from Max Planck Institute of bio-geochemistry. It contains 14 features such as pressure or humidity sampled every 10 minutes from 2009 to 2016 totaling more than 450000 samples.

4.2. Feature engineering

Before feeding our network with the dataset, we did some preprocessing.

- We converted the wind vector : wind speed (m/s) and wind orientation (deg) to its cartesian coordinated in a cartesian system so we have two new features V_x and V_y (m/s). Indeed, polar coordinates are not very suited to be inputs : angles 0° and 360° should be numerically closed but they are not. Coordinates in cartesian coordinates are smoother and continuous.

- We analyzed the seasonalities of the model. This task is essential in Time Series forecasting, we need to give the model some information about seasonality (or periodicity) such that the model can learn them easily (with a few parameters/layers). We expect the data to have two main seasonalities : a yearly one (corresponding to the season) and a daily one (corresponding to night and day).

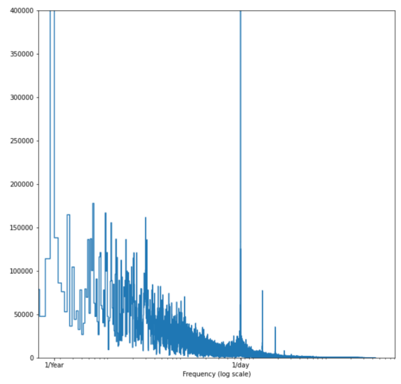

Figure 3. FFT of the signal

This intuition is confirmed by doing the Fourier transform of the temperature signal, leading to 2 peaks corresponding to the supposed frequencies. In order to handle these seasonalities, we add periodic function (cosinus and sinus) with a one-year and one-day period so that the model can capture these effects as proposed by [2].

- Similarly for the stationarity, we added some additional information such that the model can handle without too much effort linear trends. We just added the information about the day, month, year. With this method, the model can learn simple features such as the daily warming.

- We also normalized the inputs using a standard normalization : we subtracted the mean and divided by the standard deviation. We need to be careful though, we fit the normalization only on the training dataset and we use the same parameters (mean, variance) for the testing and validation sets.

Finally, we have 21 pre-processed features instead of 18. We can see the Pearson correlation between these features with the Temperature (Target).

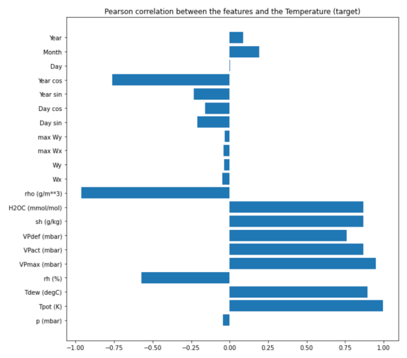

Figure 4. Person correlation between the features and the target

As we can see in the correlation plot, the features added for the seasonality (cosinus and sinus) seem to be significantly correlated to the temperature. We can also see that the correlation is less notable between the temperature and the trend relative features (day, month, year).

4.3. Rolling Mechanism

We couldn't use the naive k-fold decomposition because we would need to train the models with future data. Indeed, in the classical k-fold cross validation, we split the dataset in k parts, use k - 1 part for training and the reminding part for testing. We do the same process for every part by permuting the folds. The issue is that for some permutations, we will train the model on data posterior to the testing set. To avoid this problem we use a temporal cross validation, that is to say, the train-validation-test decomposition is rolled over the whole dataset, thus preserving the temporal structure.

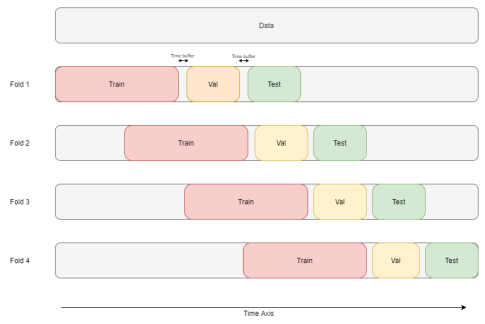

Figure 5. Temporal cross validation

Each fold use the same amount of data for the decomposition leading to a training set much more compact than the whole dataset. Finally, we added some "time buffers" these are small time intervals where the corresponding time-series sub-trajectories are not taken to construct the testing, validation, training sets. We remove voluntarily these samples so that the trajectories of the different sets are independent.

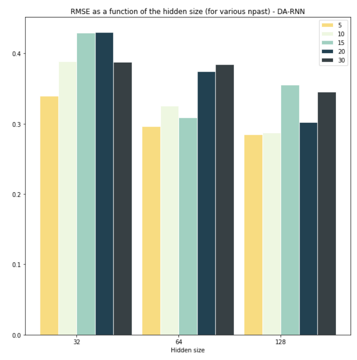

Figure 6. RMSE - DA-RNN

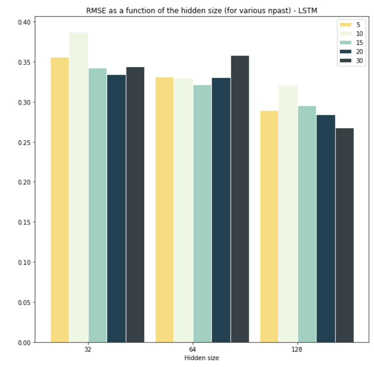

Figure 7. RMSE - LSTM

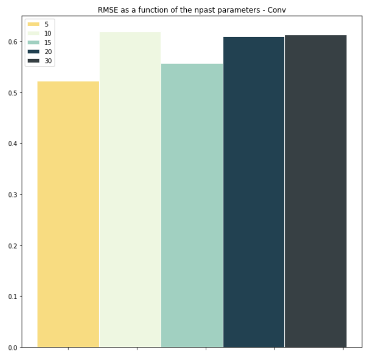

Figure 8. RMSE - Conv

If we don't remove them, the validation set will contain trajectories which are immediately next to training trajectories. This would introduce a temporal bias.

4.4. Metrics

We use several metrics to evaluate the quality of the forecast. By denoting $\hat{y}$ the estimation of the temperature and N the number of forecast made, we compute as usual :

$$\text{RMSE}(\hat{y}, y) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

$$\text{MAE}(\hat{y}, y) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

And we use a more specific metrics [7] : the *mean absolute scaled error* (MASE). For a time series with a seasonality m, a very simple way to estimate the future value is to use the previous known value in the seasonality cycle. The MASE scales the absolute error of the forecast by the average error produced by the simple heuristic presented previously. Thus a MASE score below 1 means that the model outperforms the simple model which predict the future value using the value of the day before. The benefit of this measure is that it's scale free. Sometime, the scale factor is computed on the training set, therefore we compare a MAE to the in-sample prediction, in this work we decide to use the out-of-sample prediction on the same dataset to evaluate the MASE, consequently it leads to greater losses.

$$\text{MASE}(\hat{y}, y) = \frac{1}{N}\frac{\sum_{i=1}^{N}|\hat{y}_i - y_i|}{\frac{1}{N-m}\sum_{i=m}^{N}|y_i - y_{i-m}|}$$

The models are trained using the MSE loss.

5. Experiments

5.1. Results

We fix the forecasting horizon τ in all the experiments at 24 time steps which represent 5 hours. In meteorology it's a rather short horizon since the forecast are made up to a week in general. However we are trying to be precise about the temperature and not a give a range of temperature with large uncertainties. A too small time horizon, for instance 6 time steps (1 hour) would be useless and a very naive model using the last known temperature (last hour temperature) would certainly outperform any complex model, since the temperature doesn't change much between one hour.

We have two main hyperparameters :

- n_{past} : The number of previous time-steps used to forecast the next value $\hat{y}_{t+\tau}$:

$$\hat{y}_{t+\tau} = f(y_{t-n_{\text{past}}}, y_{t-n_{\text{past}}+1}, \ldots, y_t)$$

- n_{hidden} : The hidden size of the RNNs inside each model.

We have run several experiments using DA-RNN, LSTM and the convolutional model. For both LSTM and DA-RNN models, we plot the RMSE on the test set using various n_{past} (length of the trajectories) and various hidden sizes parameters. Each model has been trained during 30 epochs (with early stopping) and over 5-folds.

In particular, we have run one model for each n_{past} in the range of [5, 10, 15, 20, 30] time-steps and for each n_{hidden} in the range of [32, 64, 128].

As we can see on the figures 6, 7, 8, both hidden size and n_{past} have a significant impact on the overall performance. For the LSTM model, a high number of hidden size seems to decrease the RMSE (thus increase the performance), whereas for the DA-RNN it is a bit more unclear. Both 64 and 128 hidden size seem to give the lowest RMSE. Moreover, n_{past} has a different impact according to the model.

Model	n_{past}	n_{hidden}	TRAIN			VAL			TEST		
			MAE	RMSE	MASE	MAE	RMSE	MASE	MAE	RMSE	MASE
Naive Last Step	-	-	0.223	0.313	0.656	0.233	0.308	0.691	0.221	0.298	0.677
Naive Rolling Average	5	-	0.238	0.333	0.698	0.249	0.327	0.736	0.235	0.317	0.720
CONV-1D	5	-	0.123	0.165	0.382	0.239	0.315	0.697	0.384	0.522	1.115
LSTM	30	128	0.133	0.181	0.409	0.174	0.231	0.513	0.196	0.268	0.596
DA-RNN	5	128	0.159	0.212	0.481	0.199	0.256	0.577	0.223	0.285	0.684

Table 1. Best results (w.r.t RMSE) for each model

For LSTMs, a high n_{past} seems to be beneficial (lower RMSE), whereas for the DA-RNN and the Convolutional model, a low (5-10-15) n_{past} yields the best results.

We can see on the figure 9 a more detailed view of the impact of the n_{past}. For each broken line (ie for each model), if we have multiple models, we take the best model according to n_{hidden} i.e. the models which has the lowest RMSE. We added too very naive models :

- Naive average : we take the n_{past} features and average them to be the forecast.
- Naive last step : we take the last known value as the forecast.

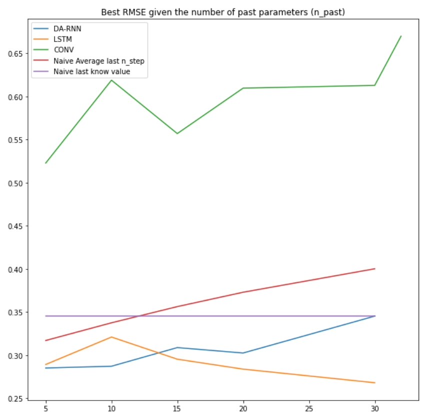

Figure 9. RMSE results - n_{past}

5.2. Discussion

As we can see on the figure 9, the convolutional model does not seem to perform well. On the table 1, we can clearly see that the model overfits : on the train set it has a very low RMSE, but it cannot reproduce it on the test set. Moreover, we can notice that the DA-RNN and LSTM models have really close performance, but the best LSTM model beats the best DA-RNN model. This result does not match the results from [8]. This phenomenon might comes from the high variability of the results. A solution would have been to average many runs of the rolling k-fold training, but this requires an important computational power. The DA-RNN model still have the advantage of being more interpretable with the attention weights whereas the LSTM layers are more of a ”black box”.

Moreover, we can see that the DA-RNN and LSTM methods perform significantly better than the naive last step model and the naive average model. This means that our models are much more accurate than an very simple estimate. This same phenomenon can be observed in a different manner looking at the MASE scores in the table 1. As we have discussed, the MASE score allow us to compare the forecast of our models with the simple heuristic : the forecast is the temperature at the same hour but from the day before. Thus, we can clearly see that our models outperform this heuristic (since the MASE is bellow 1). This heuristic is not so simple since the temperature of the day before (but at the same hour) is usually a good hint about the current temperature. Only the Convolutional model fails to beat this heuristic on the test set (MASE = 1.115 above 1).

5.3. Forecast

We show the effectiveness of the DA-RNN, this model capture well the seasonality of the time series. However we note that the performance are highly dependent of the temperature patterns. Indeed, sometimes we can observe meteorological phenomenon which are not learned by the model.

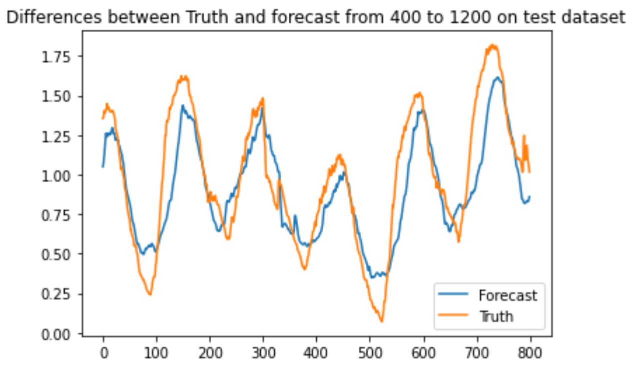

Figure 10. Forecasting vs Ground truth - Standard pattern

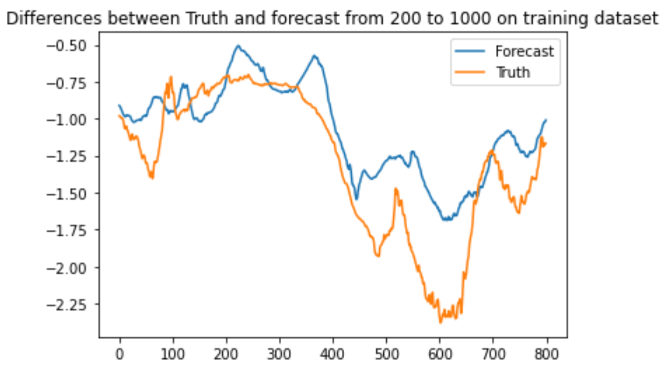

Figure 11. Forecasting vs Ground truth - different pattern

For example in the figure 10 and 11, we represent the forecast made by the model (DA-RNN with 64 hidden size). The figure 10 is the forecast made on an extract of the test set, the cyclical pattern is coherent and well captured by the model. However, on the figure 11 which represent the forecast of the model on the training set, the model does not match well with the ground truth. Even the true trajectory does not follow the seasonal day pattern. This looks like an extreme weather event.

This indicates that a perturbation in the temperature pattern may lead to weak performance.

5.4. Attention Mechanism

The interest of having an attention mechanism in addition to provide better results is to interpret the model. Indeed we can identify for a given input the weights attributed to each features and to each temporal time step. If we detect different climatic periods in the dataset (summer, winter, heat wave etc..), we could analyze the evolution of importance weights.

The figures 12 and 13 have been generated with a DA-RNN with 32 hidden layers, taking as input 12 past time steps and having a MSE of 0.041 on the test set. We observe in 12 the weights given at each feature for the past time step used as input data. In general, the weights don't evolve a lot over time. We notice that some features like *rh (%)*, *rho (g/m**3)* and *Day sin* have often a heavy weight. This is logical since it's also the feature the most correlated with the temperature according to 3.

The temporal weights are more challenging to interpret, it represents the importance of the i-th hidden state of the encoder for the decoder part for each time step. The rows represent the time step while the columns represent each hidden state of the encoder. The number 0 represent the oldest time step and the oldest hidden state used to forecast data. In general, we see that the oldest encoder hidden state has a very low weight meaning that the information present at this time step is not crucial for the forecasting. As we

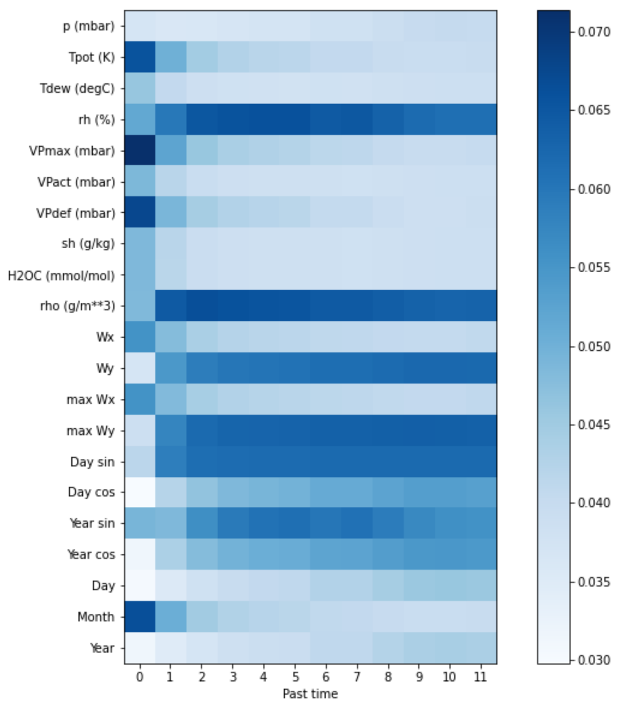

Figure 12. Feature attention weights

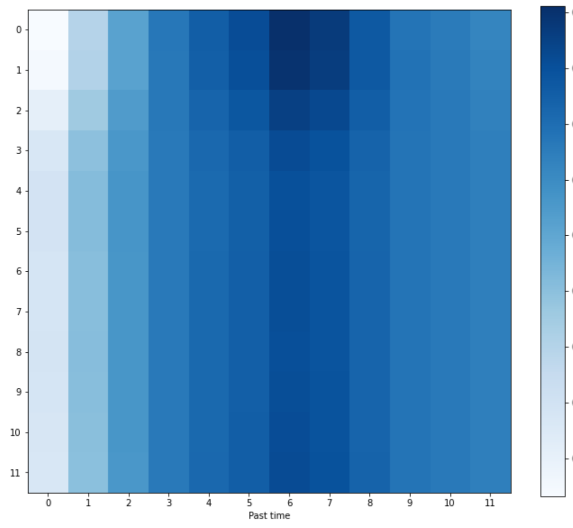

Figure 13. Temporal attention weights

look at the bottom row (the most recent information for the decoder), we observe that the weights tend to be more balanced, that is to say we take into account each past encoder hidden state. In general, by looking at several plots for different moments in the test set we observe that the oldest values are less important than more recent ones, it makes sense since to do the forecasting the recent values are closer to the real values that have to be predicted. A new proficient field of interest in deep learning is based on the transformer architecture that also use self-attention layers. [13] proposed a study based on transformer architecture both to embed time series and to forecast multivariate time series. Adding this model to the benchmark would be a trail for future work.

6. Conclusion

Despite a greater potential, the dual-stage attention-based RNN does not outperform a LSTM model, although the attention mechanism allows more flexibility to learn from data. However the dual-stage attention-based recurrent networks outperforms the benchmarks models consisting of a dilated causal convolutions networks and also simple heuristics. We demonstrate by doing robust experiments that the models based on recurrent neural networks are more suited to forecast multivariate time series, even if they include a double seasonality as shown with the climate dataset. We notice that the model may have a large variance in function of the fold i.e. of the pattern present in the dataset which can be degenerated, to avoid this problem we could have ran several times the same experiments so as to smooth the results. Beside this new architecture can be used to select relevant features thanks to the attention weighting but also provides insights about how the neural network learns from the data.

Doing a probabilistic forecasting would be an interesting approach in order to quantify the uncertainties. Theses techniques characterize the conditional probabilities of the future data knowing the past one as proposed in [9]. By proposing a mixture of density, the model would be flexible enough to fit the true distribution, allowing a user to construct scenarios based on these confidence intervals.

To improve weather forecasting, we could use data coming from a grid of meteorological stations. Indeed, there are strong correlations between the weather at different locations (because weather is a spatio-temporal phenomenon). Thus it would enrich the model but it would get closer to the real method provider use to predict the temperature which is to run physical simulations.

Source code of our implementation can be found here : https:www.github.com/SamuelDiai/weather

References

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate, 2016. 1

[2] F. Chollet. *Deep Learning with Python*. Manning Publications Co., USA, 1st edition, 2017. 4

[3] R.-G. Cirstea, D.-V. Micu, G.-M. Muresan, C. Guo, and B. Yang. Correlated time series forecasting using multi-task deep neural networks. page 1527–1530, 2018. 1

[4] F. A. Gers, D. Eck, and J. Schmidhuber. Applying lstm to time series predictable through time-window approaches. In *ICANN*, 2001. 1

[5] J. D. Hamilton. *Time Series Analysis*. Princeton University Press, 1 edition, Jan. 1994. 1

[6] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997. 2

[7] R. Hyndman and G. Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, Australia, 2nd edition, 2018. 5

[8] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, and G. Cottrell. A dual-stage attention-based recurrent neural network for time series prediction, 2017. 1, 2, 6

[9] J. Toubeau, J. Bottieau, F. Vallée, and Z. De Grève. Deep learning-based multivariate probabilistic forecasting for short-term scheduling in power markets. *IEEE Transactions on Power Systems*, 34(2):1203–1215, 2019. 8

[10] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio, 2016. 2

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017. 1

[12] E. Walter. Nonlinear system identification: Oliver nelles; springer, berlin, 2001, isbn 3-540-67369-5. *Automatica*, 39:564–568, 01 2003. 1

[13] N. Wu, B. Green, X. Ben, and S. O'Banion. Deep transformer models for time series forecasting: The influenza prevalence case, 2020. 8