Predicting the Dutch Weather Using Recurrent Neural Networks

Gerben Meijer
University of Twente
Enschede
The Netherlands
g.meijer@student.utwente.nl

ABSTRACT

Accurately predicting the weather has been an ongoing challenge for the scientific community. The recent advancements on neural networks raise the question: can neural networks be used to predict the weather? In this paper, a neural network approach to predict the temperature for the Netherlands is evaluated. Furthermore, different hyperparameter configurations are compared to each other on this complex problem. The results provide a useful overview of different hyperparameters and their optimal settings and show that the created solution is not competitive with conventional methods.

1. INTRODUCTION

The weather influences the life of billions of people every day. Snowfall hinders traffic, hot days cause ice cream sales to rise and rainy weather keeps many people inside. The weather has a substantial influence on businesses, as many businesses have to buy more stock and hire more personnel for certain weather types. Therefore an accurate prediction of the weather in the upcoming days can be vital to many businesses. Furthermore, many people depend on weather forecasts to plan their days. Wrong forecasts could lead to great inconvenience, therefore a more accurate forecast could help many individuals.

This paper focuses on weather forecasts in the Netherlands. The official meteorological institute in the Netherlands is called the *Royal Netherlands Meteorological Institute* (*KNMI*). The KNMI currently uses physics simulations to forecast the weather [2]. These models simulate a large array of variables and dynamics to make an accurate prediction.

Due to recent advancements in machine learning, neural networks have become a powerful tool to solve many data-rich problems. Some examples are text translation [13], classifying images [6] and self-driving cars [5].

In this paper, the power of neural networks is applied to the weather. These new innovative techniques are used to generate weather forecasts. These forecasts consist of the mean temperature for the coming days. The complexity of the model has been increased over time, from short-term predictions using some data from weather stations, to predictions for temperature and weather type for the coming week based on weather maps. To be able to process the sequential gridded data format used for weather maps, a recurrent convolutional neural network is used. Different hyperparameters such as the layer sizes and L2 β are explored to find the optimal settings for this problem. The results show that the final model is able to predict the general trend in temperature, but fails to predict extremes and is not competitive to the existing techniques. The evaluation of hyperparameters provides a close to optimal solution and gives an interesting view on the impact of these parameters on a real world problem.

27^{th} *Twente Student Conference on IT* July 7^{th}, 2017, Enschede, The Netherlands.

1.1 Research Questions

1. To what extent can a convolutional recurrent neural be trained to predict the daily average temperature for the coming week based on European temperature maps by using linear regression?

2. How do different hyperparameters affect the performance of the model?
   (a) How do different hyperparameters affect the mean error and standard deviation of the output generated by the model?
   (b) What are the optimal values for these hyperparameters, considering all other parameters stay the same?

2. METHODS

2.1 Model Layout

Processing sequential grid-based data with a neural network can be done in multiple ways. In this section, the final neural network design, hereafter referred to as *final model*, is explained in detail.

2.1.1 Processing Sequences

While sequential data can be processed using conventional neural networks, a better solution is available. When reading a text, a human will not look at the whole text and instantly know the meaning of that text. Instead, we read the words and remember the context created by the previous words. This is way easier to do than reading a whole text at once. In a similar way, processing this weather data can better be done by making the network remember the important information instead of making it look at everything at once.

A Long Short-Term Memory (LSTM) [11] cell is designed to accomplish such a feat. An LSTM cell has a hidden state that stores the context. The state and output are generated based on the input and the previous state.

The LSTM cell is an improvement over a normal Recurrent Neural Network. Normal RNNs have problems with the vanishing gradient problem [10] which makes it hard to train them. Therefore an LSTM is used instead of a normal RNN.

In the final model, the LSTM cell interprets the input and generates the output. To provide the model with at least one non-zero input and to have one feature distinguish input from output, the output flag was added. There are no concrete results in this paper that prove or disprove the effectiveness of this method.

2.1.2 Convolutional Layers

In order to process the two-dimensional temperature grids, *convolutional neural networks* have been used. This technique is often used with images as input and excels at interpreting patterns in two-dimensional data. Instead of connecting every input pixel to every neuron in the next layer, a kernel is moved around the image that activates on certain input patterns. Often, multiple kernels are used, each reacting to different patterns. Each kernel outputs to a different channel. These channels can be compared to the *RGB* channels in normal images. In most models, the number of channels increases as the image resolution decreases and the features become more abstract.

To decrease the resolution further, maxpooling is used after every convolutional layer. Maxpooling halves both image width and height. Maxpooling does this by using the highest value of every four pixel square in the input as the new value.

After these convolutions, the output is flattened into one large vector and used as input for a fully connected layer.

2.1.3 Fully Connected Layers

In order to allow for a more complex behavior, the model has both pre- and post-processing layers that interpret the input and LSTM output. Each of the days is passed through these layers separately. This is done for two reasons. Firstly, using the same network for the same kind of input and output makes sense. There should not be any difference between the way the first and second day are interpreted, as interpreting the sequence is a task for the LSTM. Using different network for each input day would most likely force the model to interpret days in the sequence differently while they are effectively the same. Secondly, this would make training a lot harder. This point follows from the earlier point. The model has to learn the weights and biases seven times in such a way that they can all communicate with the LSTM.

In earlier versions of the algorithm, where the output was still provided by 7 output neurons, this lack of generalization was observed. The model would often predict a different day in the sequence in a different way. This would, for instance, cause the third day to be 0.5 degrees warmer than the second day in every prediction. In the final model, this behavior is, while still technically possible, not a solution that the model will usually converge on.

2.1.4 Global Overview

In Figure 1, an overview of the final model is provided. The final model receives temperature data from the past 7 days as input. For each day it receives a map of temperatures all across the European continent, these are labeled m_x in the figure where x denotes the input day. m_7 is a map of the day before the first predicted day. It has to use this to build a seven-day prediction of temperatures for a specific point in the Netherlands. The weather maps are first processed by multiple convolutional layers. Every convolutional layer is followed by max pooling to downsample the output. The final 2D output of these layers is then flattened and used as input for one fully connected layer.

The output of this layer is then used as input for the LSTM. This sequence is extended with 7 more input vectors. These are all zero and are used to generate output after the input sequence has been processed. Furthermore, an *output flag* is added to every vector in the sequence. This flag is 0 for the 7 input elements and 1 for the 7 output elements. The preprocessed inputs for the LSTM are denoted as z_x, where x is the LSTM iteration. The LSTM generates an output sequence of 14 long. This output sequence still contains 7 outputs that have been generated while processing the input and have not been flagged as output. Therefore, these outputs will be discarded.

The 7 output vectors are then individually processed by the post-processing layers. The last layer uses a large number of inputs and no activation function in order to be able to represent many different temperature values. The output temperatures are denoted as t_x, where t_1 is the first day after m_7 and t_7 is exactly one week after m_7.

The model uses linear regression to make the predictions. The training cost or loss is determined using the mean squared error.

2.1.5 Activation Functions and Weight Initialization

The choice of activation functions and initial weights has large influence on the performance of the model. During the design of the model, many configurations have been tested in order to find an optimal configuration. For the convolutional layers the *Rectified Linear Unit* or *ReLU* activation function was used. This activation function is used often and performed better than the *Exponential Linear Unit* or *ELU* activation function. The *ELU* activation function did perform well as activation function for the dense layers but caused issues with the LSTM. The LSTM uses *sigmoid* and *tanh* activation functions, both of which tend to learn slow when the input activations are large. Because the positive output of both the *ReLU* and *ELU* activation functions are not clipped in any way, they require careful tweaking in order to not oversaturate the LSTM layers. The *tanh* activation function did perform similarly to the *ELU* function but did not require the same amount of effort to perform well. In the final model, *ELU* is still used in the first dense layer after the convolutional layers. The *tanh* activation function is used in the rest of the network.

In order to gain optimal performance, the weights have to be sampled from specific distributions. These initialization distributions have been based on Xavier initialization [7]. For the convolutional layer the weights sing the following standard deviation and distribution.

$$\sigma_{conv} = \frac{1}{n_{input}}$$

$$W_{conv} \sim N(0, \sigma_{conv})$$

In the model n_{input} is defined as

$$n_{input} = Width_{kernel} \cdot Height_{kernel} \cdot n_{channels}$$

For the dense layers the weights are initialized using

$$r_{dense} = \sqrt{\frac{6}{n_{input} + n_{output}}}$$

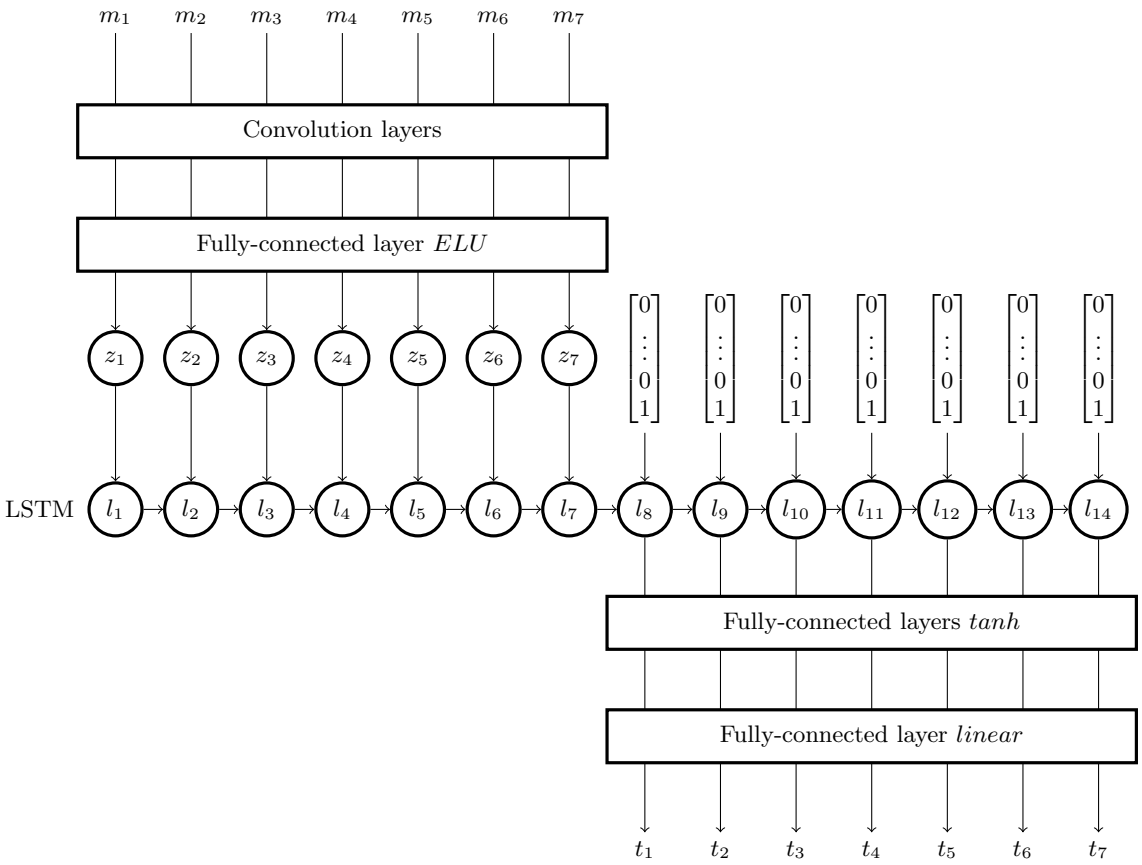

Figure 1. An overview of the model. Each map m_x is individually passed through the pre-processing layers. z_x adds a 0 to the output of these layers. l_x is a pass through the LSTM. At the output, the t_x represents the predicted temperature for that day.

$$W_{dense} \sim U(-r_{dense}, r_{dense})$$

This method of initialization is often used in combination with the *tanh* activation function. This weight initialization is also used in the *ELU* layer, because it proved to be an effective way of reducing the large activations from the convolutional layers.

The output layer has a different initial weight distribution. This distribution was set up mostly by trial and error and makes sure that after initialization the network outputs temperature in the right range. Not doing this made the final performance significantly worse in some situations. The weights in the output layer are sampled using

$$\sigma_{out} = \frac{50}{\sqrt{n_{input}}}$$

$$W_{out} \sim N(0, \sigma_{out})$$

2.2 Data

In order to train and evaluate the model, two data sources have been used. Both data sources contain data about mean temperature for every day between 1951 and 2016 and are provided in netCDF [3] format. Data from 50 weather stations across the Netherlands has been acquired from the KNMI [1]. This data is classified as *open data*, which allows use of the dataset in this research. The temperature maps of Europe have been acquired at the European Climate Assessment & Dataset or ECA&D. This dataset is called E-OBS [9]. The use of this E-OBS dataset is not completely open but is permitted for this research. The E-OBS data provides mean temperatures for the whole European continent.

The data has been normalized using $T_{norm} = \frac{T_{data}}{30} - 0.5$. This ensures that the activations in the convolutional layers don't become too large.

The E-OBS data also contains many empty grid points in the sea, there are no measurements in these locations. By default, these data points have a very large negative value. This highly decreased the performance of the neural network. This problem has been solved by replacing these values with 0 in the normalized data.

A part of the data has been kept apart in order to validate the model.

2.3 Evaluated Factors

During the research, many methods have been applied in an attempt to maximize performance. The most significant methods will be evaluated in this section.

2.3.1 Network Size

The size of a neural network affects the complexity of functions it can fit. A larger network can often be more accurate and powerful. Intuitively, a larger network can also overfit easier because it can express more complex functions. Therefore it is not always better to add more neurons or layers to a network. The size of the preprocessing and LSTM hidden layer, as well as the sizes of the post processing layers, will be scaled to evaluate the effects of layer sizes. The normal network size can be found in Table 1. In the experiment, some of these values will be scaled, as can be seen in the table. When multiplication with the scaling factor results in a non-integer number, the result will be floored to make it a proper integer.

2.3.2 Methods Against Overfitting

Overfitting [8] is one of the biggest dangers to consider when applying deep learning to a problem. When a model overfits, it will perform amazingly on the training data but it will fail as soon as it encounters new data. Instead of generalizing, the model fits all insignificant details and noise in the training data. Due to the commonality of this problem, many techniques have been proposed to overcome this problem.

To overcome this problem, one first has to detect the problem. During the research, the model performance has been cross-validated with testing data. This dataset has been kept aside from the training dataset, ensuring that the data is unknown to the model. Using this data, it became clear that the model did overfit the test data.

2.3.3 Dropout

Dropout [14] is a technique that has achieved great success in overcoming the overfitting problem. Dropout effectively removes a percentage of the activations in the network during training. This enforces the network to become more robust since it cannot rely on individual neurons. Therefore, using dropout enforces a more generic solution.

While dropout performs well in certain cases, there are some situations where applying dropout will have a negative impact on the performance of the model. Use of dropout in regression tasks like the one in this paper is said to have negative impact on performance. Use of dropout in convolutional might also reduce performance, although this is still debatable. Using a large dropout keep chance, $\geqslant 80\%$, might act as noise and improve generalization. Multiple dropout configurations will be evaluated in the results.

2.3.4 L2 Regularization

L2 Regularization is another method against overfitting. It adds a cost to the squared value of the weights. This cost multiplied with a factor β and added to the loss function. The impact of regularization can be regulated by adjusting this β. In the results, some values for β will be evaluated.

2.3.5 Other Extensions

Most of the runs will have two extra modifications. The first modification is an alternate cost function. On top of the normal cost, an extra cost is added for the difference in standard deviation of the output. The original model did often generate forecasts that were very linear. Their standard deviation was quite low. While these predictions were technically more correct, it delivered very safe predictions. While the added cost does force the network to perform worse on the mean error measurement, it greatly improves the standard deviation of the output.

The other added extension is input noise. This noise is used to make the network more resilient to unknown inputs and thus also reduces overfitting. This noise matrix is generated from $N(0, \sigma = 0.05)$.

2.4 Implementation

The model and experiments are programmed in Python 3 using TensorFlow [4]. Data is converted from netCDF to a binary format that can be natively read by Python for better access across different systems.

The model is trained in epochs of 49 minibatches and then evaluated using the test data. The minibatch size is 22. Every epoch, a summary of the performance on the test data is added to TensorBoard, which is a tool used to evaluate the performance of the model. The data from TensorBoard is used to compare different the evaluation parameters discussed earlier.

The model is trained using the Adam optimizer [12]. The

Table 1. An overview of the layers in the model

Layer	Activation function	Input and Output vector sizes	Output size scaled
Convolutional	*ReLU*	25 by 25 kernel, 1 chan. in, 4 out	No
Convolutional	*ReLU*	5 by 5 kernel, 4 in, 5 out	No
Convolutional	*ReLU*	5 by 5 kernel, 5 in, 6 out	No
Convolutional	*ReLU*	5 by 5 kernel, 6 in, 7 out	No
Dense	*ELU*	$14 * 17 * 7 = 1666$ in, 20 out	Yes
LSTM cell	*sigmoid* and *tanh*	Sequence of 14 vectors, each 21 in. 10 out/hidden layer	Yes
Dense	*tanh*	10 in, 10 out	Yes
Dense	*tanh*	10 in, 100 out	Yes
Dense	*linear*	100 in, 1 temperature out	Yes

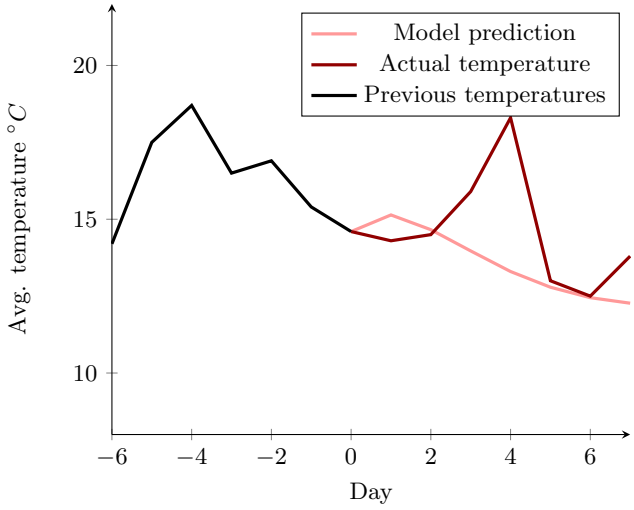

Figure 2. An average prediction made by the model. It failed to predict the warmer days before the temperature went down

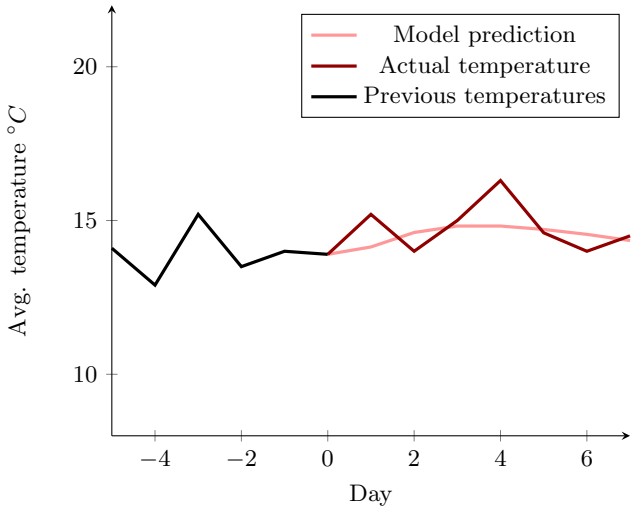

Figure 3. The model performs above average on a quite consistent temperature

learning rate starts at 0.001 and decays exponentially with a decay rate of 0.1 over 7100 minibatch passes.

3. RESULTS

In this section, the results of the research are examined. All of the evaluated models have been trained on a high-end GPU for 2 hours. First, the forecasting performance will be evaluated, then the different techniques mentioned in the methods section will be compared.

3.1 Forecasting Performance

After implementation of the model, the results were not deemed competitive with the current weather prediction models. In Figures 2, 3, and 4 some forecasts can be seen. These examples are generated using the model that had the lowest mean error after 2 hours of training (`reg-050`). The chosen examples are one average prediction (chosen by eye) and two extremes, one has a very small mean error while the other one has a large mean error.

3.2 Comparison Between Models

In this section, a number of different configurations will be compared. This comparison will provide information about the influence of different hyperparameters on model performance. In Table 2 an overview is provided of all used configurations.

3.2.1 Using Extensions

As proposed in the methods, the effect of the 2 extensions will be measured. These extensions were an added cost on

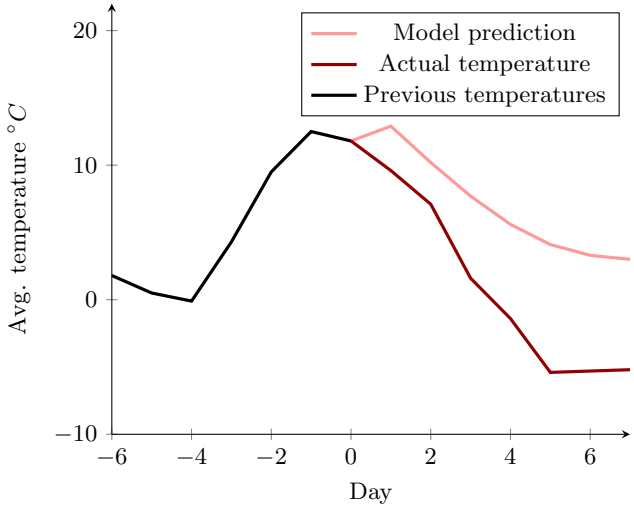

Figure 4. The model does not foresee the freezing temperatures

Table 2. An overview of all tested configurations

Run name	L2 β	Dropout position	Dropout keep factor	Layer scaling	Uses extensions
`base-without`	0	None	1.0	1.0	No
`base-with`	0	None	1.0	1.0	Yes
`drop-all`	0	Pre, post, and LSTM	0.5	1.0	Yes
`drop-conv`	0	Convolutional layers	0.8	1.0	Yes
`drop-prelstm`	0	Pre and LSTM	0.5	1.0	Yes
`lay-0.5`	0	None	1.0	0.5	Yes
`lay-0.25`	0	None	1.0	0.25	Yes
`lay-5.0`	0	None	1.0	5.0	Yes
`reg-0.001`	0.001	None	1.0	1.0	Yes
`reg-0.05`	0.05	None	1.0	1.0	Yes
`reg-0.5`	0.5	None	1.0	1.0	Yes

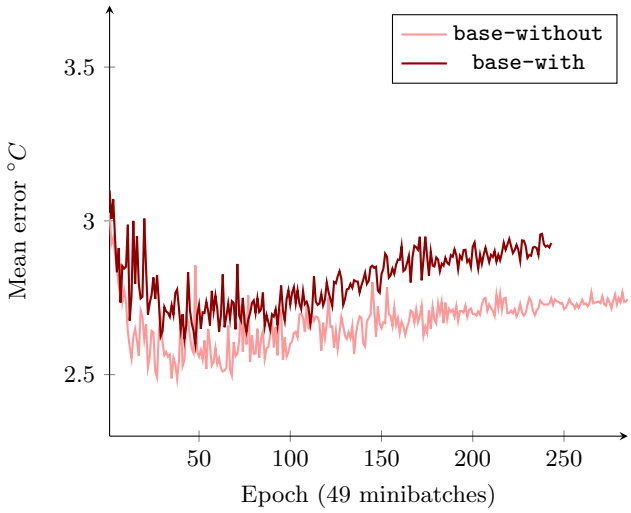

Figure 5. A comparison of mean error between `base-with` and `base-without`

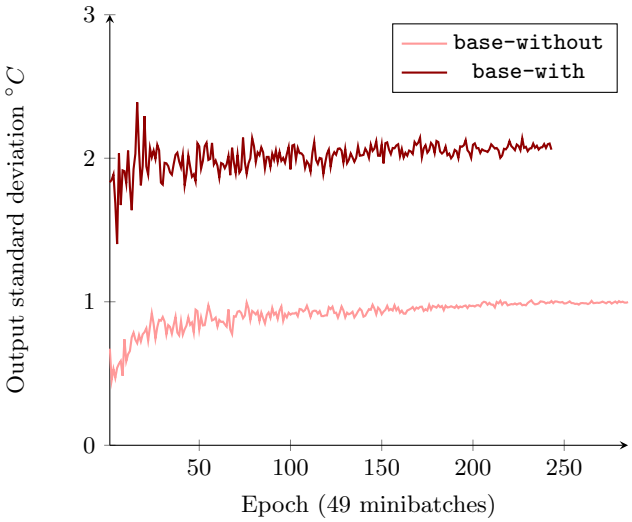

Figure 6. A comparison of standard deviation between `base-with` and `base-without`. The standard deviation of the actual temperatures in this sample is $2.230°C$.

standard deviation and added noise on input. In Figure 5 the mean error of both models is plotted. This is a prediction on a part of the validation data which stays the same throughout the training.

As can be seen in the figure, both configurations overfit. `base-with` ends up around the same performance it had after the first 10 epochs. This has been one of the main issues during the design of the model.

Looking at the mean error, the `base-with` configuration seems to be performing worse. This view changes when comparing the standard deviation of their output. In Figure 6 the standard deviations are plotted over the course of the training. From this graph it becomes obvious that `base-with` is significantly closer to the actual standard deviation, $2.230°C$, than `base-without`.

3.2.2 Dropout

Next, the results for different dropout configurations are listed. The first configuration, `drop-all` applies dropout to all parts of the model except for the convolutional layers and the output layer. `drop-conv` uses a higher dropout keep probability and only applies dropout to the convolutional layers. This functions as a form of noise on all convolutional layers. The last configuration, `drop-prelstm`, applies dropout to the preprocessing layer after the convolutional layers and on the LSTM. In Figure 7 a comparison can be found.

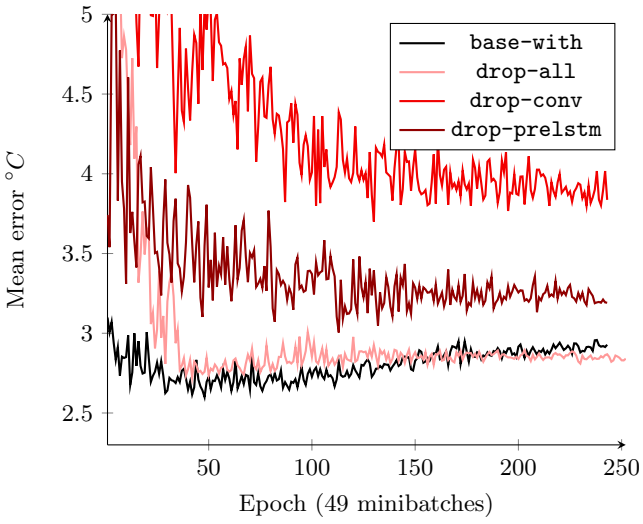

Figure 7. A comparison of mean error between different dropout configurations.

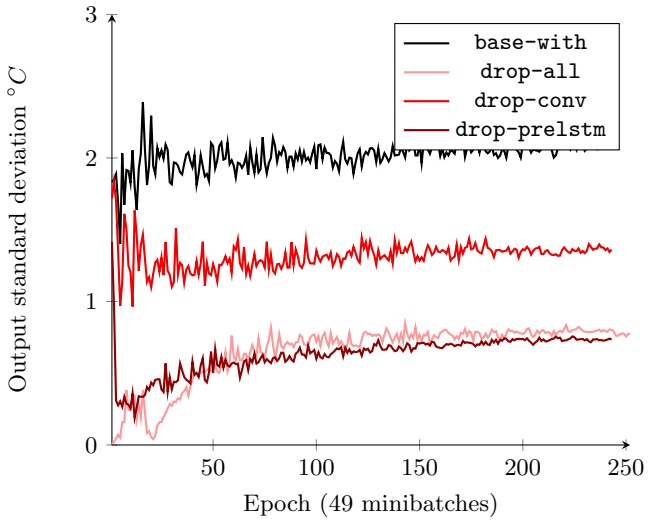

Figure 8. A comparison of the output standard deviation between different dropout configurations.

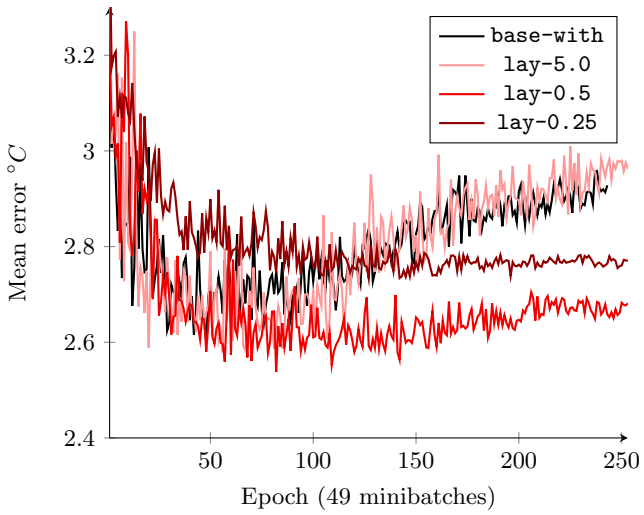

Figure 9. A comparison of mean error between different layer scales.

It is important to note that all dropout configurations start with a significantly worse mean error compared to the base configurations. In Figure 8 one can find the output standard deviations for the configurations. All dropout configurations have a smaller standard deviation compared to the base model. Notice how `drop-all` starts with an almost completely static output. These results will be discussed further in the discussion section.

3.2.3 *Layer Size*

The results of layer scaling can be seen in Figure 9. From this figure, it becomes clear that `lay-0.5` has the best end result. It has the lowest mean error after two hours of runtime. From these runs, as well as from the other runs that have not been plotted, two major trends become visible.

The first trend is that all tested configurations with a scaling factor lower than 0.5 underfit the data. `lay-0.25` performs significantly worse at its best compared to the lowest mean errors of all other runs. `lay-0.15`, a configuration that has not been displayed in the plot, continues this pattern. It performs significantly worse than `lay-0.25`.

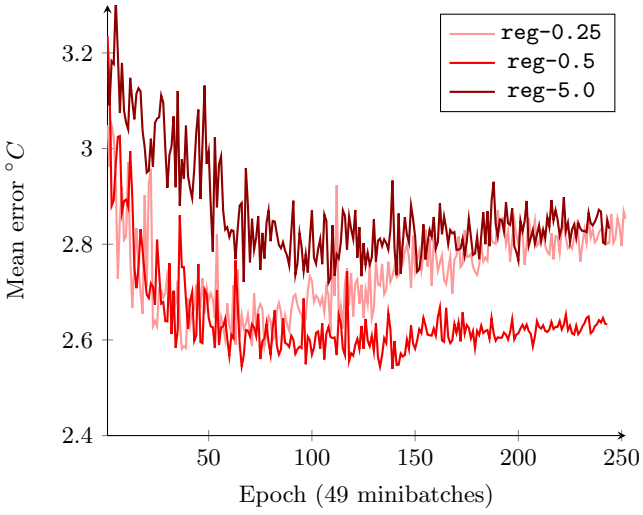

Figure 10. A comparison of mean error between different L2 β factors.

The other trend is that all layer scales above 0.5, maybe even including 0.5, start overfitting. There is most likely an optimal scale slightly under 0.5 because even `lay-0.5` does overfit a small amount. Both `base-with` and `lay-5.0` converge around the same mean error. This case holds with larger scaling factors up until extreme values like 50.0, where training starts to take more epochs and performance decreases.

The standard deviations do differ between different configurations, but not significantly. Smaller layers had a smaller output standard deviation, but still significantly higher than `base-without`.

3.2.4 *L2 Regularization*

L2 regularization has been evaluated as a technique to prevent overfitting. The L2 β value determines the influence of this L2 cost on the total cost function. In Figure 10 the most notable configurations have been plotted.

L2 regularization shows a trend that is similar to the one that was already observed in the layer sizes. The optimal value does lie around $\beta = 0.5$. Lower values tend to overfit and higher values tend to underfit. The standard deviation is not significantly different across different configurations.

4. DISCUSSION

4.1 Forecasting Performance

The forecasting results show that the model is not capable of predicting any extreme weather changes. This is unfortunate since that is the aim of weather forecasts. The model does, however, predict the general trend in the mean temperature.

Another interesting thing to note is the very unnatural bump from previous temperatures to the prediction. In two of the three plots, the temperature on the first day is wrong in an unnatural way. It suddenly bumps up before going down. This might be learned behavior to reach a higher standard deviation. If this is the case, the addition of this extra cost becomes debatable as it forces the network to learn this kind of unwanted behavior. There is no hard proof if this is really the case.

The performance of the model is lacking because of the quality of the data in combination with the expectations. First off, the dataset was too small to train a deep neu-

ral network. This lead to overfitting. Secondly, making a prediction of the temperature up to one week ahead based on maps of land temperatures in Europe not reliably possible. Temperature is dependent on many variables which were not given to the model. This data was not easily available. The current data would probably work better for a country further away from the sea because the sea provides no information. Even then it is estimated that more variables and data would be necessary.

4.2 Comparisons

4.2.1 Extensions

Comparing between the models with and without added noise and standard deviation cost, it becomes clear that the mean error becomes larger when using these extensions. Although it decreases the precision of the predictions, it does arguably make the predictions more useful. A network that predicts the temperature to stay roughly the same when this is clearly no the case is in many cases less useful than one that does try to predict larger temperature changes, even at the cost of some precision.

4.2.2 Dropout

When looking at the dropout results, it is immediately visible that all dropout configurations start with predictions that are way worse than those of the base model. Furthermore, both `drop-conv` and `drop-prelstm` fail to converge at a reasonable mean error. The reason for the bad start of all of these configurations is that the activations tend to be lower due to dropout. When on average half of the inputs are dropped, the output activation tends to be smaller when using the same weights. Because the weight initialization has not changed, the model outputs way smaller values.

Although two of the proposed configurations have failed, `drop-all` has actually converged on a smaller mean error than the overfitted `base-with`. It has also mostly prevented overfitting, although there seems to be a small increase in mean error over the test data. While `drop-all` might perform well when looking solely to the mean error, it does not seem to perform well when examining the standard deviation of the output. Examining the standard deviations, it is visible that `drop-all` starts with an almost static output. Due to this, the network has predominantly trained towards a low cost on the error which was more rewarding than raising the standard deviation.

4.2.3 Layer Size

The results for layer scaling show that a scaling factor of 0.5 is close to optimal. Higher values all tend to overfit in similar fashion, while lower values quickly begin to underfit. It makes sense that there is an optimal layer size. Larger layers can and will fit more complex outputs. When the layers become larger, the model learns the data instead of the underlying pattern. Smaller layers, on the other hand, cannot fit the underlying pattern well enough.

It is debatable where the exact bottleneck is situated in this case. The LSTM hidden layers and a post-processing layer are only 2 neurons in `lay-0.25`. If these layers would have been equal to the preprocessing layer, which has double the neurons, then the model might have been more optimal using a scaling factor of 0.25 compared to `lay-0.5`. This is difficult to prove, as that would require many runs on more detailed configurations. This is out of the scope if this research.

There is some discussion to be had about the choice of the base model, given these results. `lay-0.5` would seem like a better candidate, given its better performance. On the other hand, the overfitting of the current base configuration makes it an excellent benchmark for methods such as dropout or L2 regularization.

4.2.4 L2 Regularization

The results of the L2 regularization configurations have shown that L2 regularization is a strong method to counter overfitting. In the current model, the weights of the last layer have not been included in the L2 cost on purpose. This is most likely the reason why the network is able to keep a consistent standard deviation.

5. CONCLUSION

A convolutional recurrent neural network cannot predict the daily mean temperature for the upcoming week using linear regression with an accuracy that comes close to that of conventional models. The largest issue is the input data, which is not enough to train a deep neural network and does not contain enough information to accurately predict the temperature for the upcoming week. Future work may apply this model to other data like hourly precipitation. This would, in theory, allow for more data because of the shorter time frame. Hourly data, or even shorter time steps, would also be more predictable as weather tends to be chaotic.

In the results, it becomes obvious that dropout does severely impact the performance of the model and care should be taken when using this technique. When applied correctly, it does stop overfitting almost completely while staying competitive with models without dropout. L2 regularization does stop the model from overfitting but will lead to underfitting when the L2 β is too large. It is more stable than dropout in similar conditions. Different layer sizes do influence the performance as expected. Smaller networks tend to underfit while larger networks tend to overfit.

Based on the evaluation of the chosen hyperparameters, an optimal model would have layers with half the number neurons the final model had in the adjusted layers. It would use an L2 β close to 0.5. It would either use no dropout or use the configuration of `drop-all`. Based on the desired output, standard deviation cost might be omitted. This depends on the desired result.

6. ACKNOWLEDGEMENTS

I acknowledge the E-OBS dataset from the EU-FP6 project ENSEMBLES (`http://ensembles-eu.metoffice.com`) and the data providers in the ECA&D project (`http://www.ecad.eu`) [9]. Furthermore, I would like to thank D. Bucur and M. Poel for their assistance during the research. Finally, I would like to thank G.J. Laanstra for providing access to the deep learning cluster.

7. REFERENCES

[1] Knmi data centre. `https://data.knmi.nl/datasets`.

[2] Knmi webpage about weather models. `https://www.knmi.nl/kennis-en-datacentrum/uitleg/weermodellen`, last visited on May 10th 2017.

[3] netcdf website. `http://www.unidata.ucar.edu/software/netcdf/`, last visited on May 11th 2017.

[4] Tensorflow. `https://www.tensorflow.org/`, last visited on May 4th 2017.

[5] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[6] D. Ciregan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3642–3649, June 2012.

[7] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[8] D. M. Hawkins. The problem of overfitting. *Journal of Chemical Information and Computer Sciences*, 44(1):1–12, 2004. PMID: 14741005.

[9] N. H. A. K. T. E. K. P. J. M. N. Haylock, M.R. A european daily high-resolution gridded dataset of surface temperature and precipitation. *J. Geophys. Res (Atmospheres)*, 113, D20119, 2008.

[10] S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions.

[11] S. Hochreiter and J. Schmidhuber. Long short-term memory. 1997.

[12] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[13] Q. V. Le and M. Schuster. A neural network for machine translation, at production scale. `https://research.googleblog.com/2016/09/a-neural-network-for-machine.html`, last visited on May 4th 2017.

[14] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.