LSTMs and Mean Global Temperature Anomaly Prediction

COMP815 Nature Inspired Computing - Assignment 2

Eirik Folkestad - Student no: 17966805

I. Introduction

When Recurrent Neural Networks (RNN) were introduced they were able to solve many of the problems other machine learning algorithms could not solve. However, certain problems with the vanishing gradient problem and difficulties with remembering longer sequences showed that standard RNNs also had limitations. LSTMs was a successful attempt in accommodating these problems and proved to be able to solve problems that even RNNs would have trouble solving.

In this project we will first have a look at some achievements that others have made with the use of LSTMs, then we will see how the standard RNN network and the LSTM network works, and then attempt to apply an LSTM network on a time-series prediction problem with data consisting of mean global temperature anomalies.

II. Related Work

Graves [1] shows how LSTM networks can be used to generate complex sequences with long-range structure, by predicting one point of data at a time. The approach is demonstrated for text (data is discrete) and online handwriting (data is real-valued). It is extended to handwriting synthesis by letting the network to condition its predictions on a sequence of text. The resulting system is able to generate different styles of realistic cursive handwriting.

Sundermeyer, Schlter, and Ney [2] show that LSTM networks are useful in language modeling. They are aware that the standard recurrent neural networks improve on language modeling by being able to take into account all of the preceding words in a sentence whereas an FFNN can only exploit a fixed context length to predict the next word. However, they use LSTMs over standard RNNs due to being easier to train and better at learning long sequences. In their work, they analyze an LSTM network on an English and French language modeling task where the experiments show improvement over simpler RNN models.

Gregor, Danihelka, Graves, Rezende and Wiestra [3] introduces the Deep Recurrent Attentive Writer (DRAW) neural network architecture for image generation. DRAW networks combine a novel spatial attention mechanism that imitates the foveation of the human eye and a variational auto-encoder which make use of LSTMs. The system improves on the state of the art for generative models on MNIST, and, when trained on the Street View House Numbers dataset, it generates images that look very real.

Eck and Shmidhuber [4] creates an adaptive signal processing device that learns by example how to generate new instances of a given musical style. LSTMs are favoured over RNNs due being able to keep track of distant events in a sequence. LSTM networks have been successful in similar domains where simpler RNNs have failed. If given blues music, the results show that LSTMs can learn the music form and is able to compose novel blues melodies and play it with good timing. The LSTM network also does not drift from the wanted music form once the relevant structure is found.

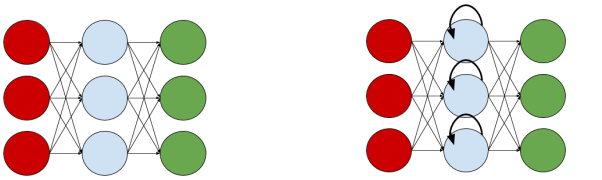
Lipton, Kale, Elkan and Wetzel [5] investigate the use of LSTMs due to their sequence learning abilities in the aid of diagnosing patients based on multivariate time-series from clinical medical data, especially in the intensive care unit. Patients sensor data is recorded in their Electronic Health Record, but it is difficult to mine effectively due to inconsistent data and other irregularities. The LSTM is used to perform multilabel classification of diagnoses and trained only on raw time series, the model outperform many strong baselines including a multilayer perceptron trained on hand-engineered features.

III. Theory

A. *Recurrent Neural Network*

A recurrent neural network (RNN) is an artificial neural network (ANN) which is capable of processing sequential data. It can do this by 'remembering' previous states of computation and use this information to aid in the prediction task. A feed-forward neural network (FFNN), on the other hand, makes the Markov assumption [6] which means that it computes the next state based on only the present state. A visualization of this can be seen in Figure 1a where we can see that information is passed only from a unit, also called a neuron in FFNNs, to a unit of the next layer in the computational graph. An RNN adds a recurrent edge to the units as we see in Figure 1b. This recurrent edge allows the RNN to use what it has already learned in previous states in addition to the present state compute the next state. The ability of recalling previously learned information can resemble a 'memory'.

The forward propagation performed in an RNN starts with inputting the data into the input layer of the network. The information is processed in the recurrent layer by recurrent units for each time-step t denoted as a_t as we see in Equation 1. a_t is denoted by the sum of the input, x_t, and the output of the previous state, h_{t-1} which represents the information from the recurrent edge. The input and output of the previous time-step are weighted by an input-to-hidden layer weight matrix U and W and these weight matrices are shared over every



(a) FFNN Graph Structure (b) RNN Graph Structure

Fig. 1: **(a)** FFNN Example Graph Structure and **(b)** RNN Example Graph Structure. Red Denotes Units in Input Layer, Blue in the Hidden Layer(s) and Green in the Output Layer

time-step. The bias, b, is added to increase expressiveness of the RNN.

$$a_t = U * x_t + W * h_{t-1} + b \tag{1}$$

Before we can calculate an output of the RNN, an activation function is first applied on a_t to induce non-linearity into the network as seen in Equation 2. A typical activation can be variants of $sigmoid$, $tanh$ or $ReLU$ functions which maps a value to a fixed interval.

$$h_t = activation(a_t) \tag{2}$$

The output of the RNN is calculated by applying a hidden-to-output weight matrix which is shared by every time-step, V, on the output of the time-step t along with a bias, c, for expressiveness as we see in Equation 3.

$$o_t = c + V * h_t \tag{3}$$

In Equation 4 we see that the output for a time-step t, $\hat{y}_t$ is then decided by a $softmax$ function which gives out the probability of the input x_t belonging to a class.

$$\hat{y}_t = softmax(o_t) \tag{4}$$

The RNN uses the Back-Propagation Through Time (BPTT) algorithm to train the network after each iteration which is almost similar to the Back-Propagation (BP) algorithm. The difference is that it does not only update the weights in each layer in the direction that reduces the prediction error, it also updates the weights for each time-step. This introduces a problem into the RNN called the vanishing/exploding gradient problem. The essence of this problem is that if the weights approach zero or a large number this would further decrease or increase the training with every time-step due to the use of shared weights and BPTT [7]. Another problem will also be that what happened recently might get overwritten fairly easy since we do not have a scheme that takes into account that recent information might be more important than past information if we encounter a sub-sequence.

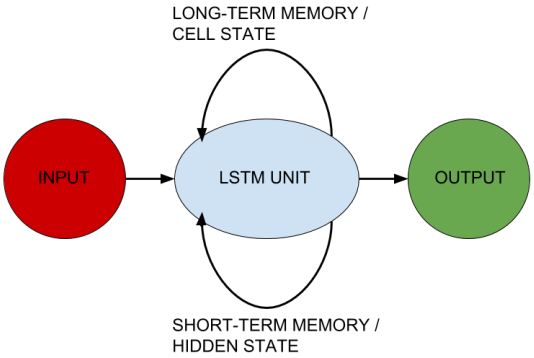


Fig. 2: An Illustration of An LSTM Network with 1 Hidden Layer and 1 LSTM Unit

B. *LSTM Network*

An effective method for sequence modeling is a gated RNN which utilize gated recurrent units instead of the recurrent units. An LSTM network, or Long Short-Term Memory Network, is one such gated RNN. The LSTM unit is a gated unit and contains three gates of which each brings new functionality into the RNN. The general idea is to create paths through time that makes the gradients not vanish/explode. The alterations introduced in the LSTM unit is done to accommodate the vanishing gradient problem and more systematic memory management. The gates introduced are:

- **A Forget Gate** which controls what information we should forget.
- **An Input Gate** which decides what we should remember of the input data.
- **An Output Gate** which decides what comes out of the LSTM cell based on what the input was and what resides in the memory.

These gates control what is now not only a hidden state, h_t, but also a cell state, $cell_t$. These are as follows:

- **The Hidden State** functions as a short-term memory of the previous states when input and is updated with the current state information before being output.
- **The Cell State** functions as a long-term memory of the previous states when input and is updated with the current state information before being output.

The hidden state and the cell state function respectively as a short-term memory and a long-term memory of which makes for better control of the memory flow in the neural network with more predictable and useful outcomes. These states are, as in the RNN units, updated based on the previous state h_{t-1} and now also $cell_{t-1}$ as we see in Figure 2 of a LSTM network with 1 hidden layer and 1 LSTM unit. The unit is receiving information from the previous state from the two recurrent edges.

First the forget gate is invoked and $cell_t$ is changed to accommodate what we should continue to remember or forget. As seen in Equation 5, this is done by by producing a number between 1 and 0 where 1 means remember everything from

Year	Land+Ocean
1995-12	0.28
1996-01	0.25
1996-02	0.48
1996-03	0.32

TABLE I: Example of Time-Series Data Records 1995-2017

the previous state $t-1$ and 0 means forget everything from previous state $t-1$.

$$f_t = sig(W_f * x_t + U_f * h_{t-1} + b_f) \tag{5}$$

The second step is to decide what we are going to update the cell state with. The input gate i_t, as seen in Equation 6, decides which values of the cell state we will update and the candidate values, $cand_t$ as seen in Equation 7, are values we want to add to the cell state. By combining these two with element-wise multiplication, we can update the cell state with only specific values.

$$i_t = sig(W_i * x_t + U_i * h_{t-1} + b_i) \tag{6}$$

$$cand_t = tanh(W_{cand} * x_t + U_{cand} * h_{t-1} + b_{cand}) \tag{7}$$

The cell state is now updated as seen in Equation 8.

$$cell_t = f_t \circ cell_{t-1} + i_t \circ cand_t \tag{8}$$

Lastly we want to find the new hidden state h_t. This is done by finding which values we want to output first, o_t, as seen in Equation 9. As we see in Equation 10, h_t can be found by combining the output with $tanh(cell_t)$ with element-wise multiplication so we only output the parts of the cell state that we want.

$$o_t = sig(W_o * x_t + U_o * h_{t-1} + b_o) \tag{9}$$

$$h_t = o_t \circ tanh(cell_t) \tag{10}$$

IV. Data & Preprocessing

LSTMs have been prove useful for predicting future elements from a sequence [1] and we are going to apply it on a univariate time-series. The dataset that we will use in this project is a time-series which contains **1652** data records of monthly global mean temperature from January **1880** to August **2017** measured by NASA [8]. We will, however, use a subset consisting of the last **249** months which is the period December **1995** to August **2017** due to computational complexity. The values are labeled **Station** and **Land+Ocean** and they are both anomalies from the mean global temperature over the 30-year period 1951-1980 only measured differently. In this project we are going to reduce the data from a multivariate time-series to a univariate time-series by only using the **Land+Ocean** value. The time-series can be seen in Figure 3 and four data records can be seen in Table I.

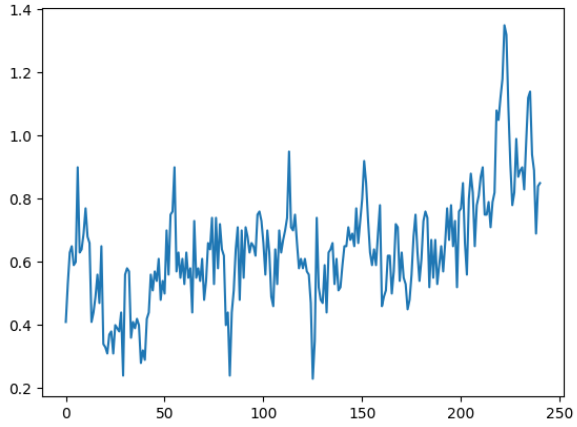


Fig. 3: Mean Global Temperature Anomalies 1995-2017

Year	Water Level
1996-01	-0.03
1996-02	0.23
1996-03	-0.16

TABLE II: Example of Stationary Time-Series Data Record 1996-2017

Before we can use the dataset in the LSTM network model we need to do a couple of transformations to it so that we can produce better results when predicting water levels.

One important step is to make the dataset stationary. This is desirable since we would like the predictions to be independent of the real temperature anomaly and instead be about the increase or decrease in the temperature anomaly. To do so we transform the **Land+Ocean** field in every record to represent the difference from the previous data-point in the time-series. The example in Table II represent one such transform for 1996-2017. Notice that we only kept December in the dataset since the difference is defined by the current value minus the previous value, which now makes the dataset one value smaller, hence January 1996 to August 2017.

After this we transform the time-series to a supervised learning problem where the target of each temperature anomaly increase value is the next temperature anomaly increase value in the time-series. An example can be seen in Table III. The anomaly for 1996-04 is 0,36.

The dataset is now split into a training set and a test set

Year	Land+Ocean Value	Land+Ocean Target
1996-01	-0.03	0.23
1996-02	0.23	-0.16
1996-03	-0.16	0.04

TABLE III: Example of Stationary Time-Series Data Record 1996-2017 as a Supervised Learning Problem

depending on how many of the last months we want to predict. This will vary with the conducted experiments.

A reasonable data processing step prior to training of a ANNs is to normalize the input data so that their variability represent their importance. The LSTM network expect the data to be in the range of the activation function. In this case it is the *tanh* function which transforms data into the range **(-1, 1)**. When we scale, we only do so prior to the training data's minimum and maximum values since we do not want to cheat by involving data from the test set in any way when training the LSTM network. If the test set contains values that the training set does not, then our trained model should not be able to know about this from training but rather use its knowledge about the training set to make decisions.

The data is now ready to be used by the LSTM network. After a prediction is made, we calculate the inverse normalization and inverse difference value so that the predicted values represent the predictions in the original format. This means that the predicted values are transformed back to the format seen in Table I.

V. Experiments & Results

We want to find out how accurately we can predict the mean global temperature anomalies for the whole year of **2017** by generating the last *8* anomalies from the previous months and compare the results with the real anomaly values. All the experiments conducted are carried out using an LSTM network with **1** hidden layer similar to the one in Figure 2. It is implemented in *Python* with the *Keras* library [9] and training over 1 epoch is training the network on the full training sequence before resetting the state of the LSTM units cell state. Before prediction we are building up the cell state again by predicting the training data so that the network is ready for predicting the next value of the sequence. The results will be measured in *Root Mean Square Error* (RMSE).

Given the computer power accessible to conduct the experiments, further testing of a higher number of epochs than **20** was not done. However, reasonable results where found within the limitations and no indications was found about that a greater number of epochs would produce better results.

The plots show a window of the last **20** anomalies and their prediction. The true values are plotted as a blue line and the predicted as an orange line. If the blue is not present in the plot then the orange is overlapping it.

In the first experiment, we only want to predict the very last element of the time-series which is August 2017. To do this we train the LSTM network on every temperature anomaly up to the last anomaly. We then predict the last anomaly with high precision as we see in Table IV and we also see that no more than **5** epochs, **1** neuron and **1** hidden layers are needed to perform very well and almost as good with only **2** epochs. The table only contains values that continually improved the result. A visualization of the best result can be seen in Figure 4.

In the second experiment, however, we can see that predicting all the temperature anomalies from **January 2017** to **August 2017**, which is **8** anomalies, decreases performance

TABLE IV: Unistep LSTM Network RMSE Results with Various Hyper-Parameters Averaged over 5 Runs Predicting the Last 1 Month

Epochs	Neurons	Avg. RMSE
1	1	0.013
1	2	0.009
2	1	0.005
5	1	**0.004**

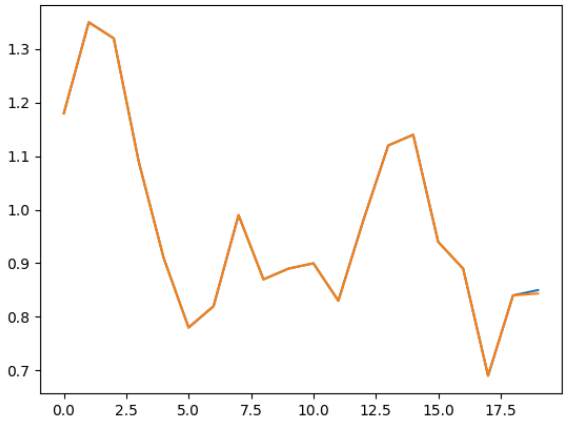


Fig. 4: Visualization of Result Uni-Step LSTM Network with 1 Neuron Trained over 5 Epochs Forecasting the Last Temperature Anomaly

significantly for the extra temperature anomaly we are trying to predict. Each prediction is done with the knowledge of the value of the previous prediction and the error increases with the more anomalies we are trying to predict. This is an expected behaviour since we are predicting new values on already predicted values. The configurations which yielded the best results for ***n*** successive predictions can be seen in Table V and a visualization of the best result can be seen in Figure 5.

In the third experiment, we want to predict the last *n* anomalies by the past anomalies directly. This means that we will en up having *n* predictions of *n* time-steps into the future. This is different from the last experiment since we are not predicting only **1** anomaly at a time, but rather *n* anomalies at a time. This also involves reshaping the data into a value-target paired data set which contains values of size *1* with

TABLE V: Unistep LSTM Network RMSE Results with Best Hyper-Parameters Averaged over 5 Runs Predicting the Last *n* Month(s)

Months	Epochs	Neurons	Avg. RMSE
2	5	16	0.027
4	1	1	0.180
8	1	1	0.178

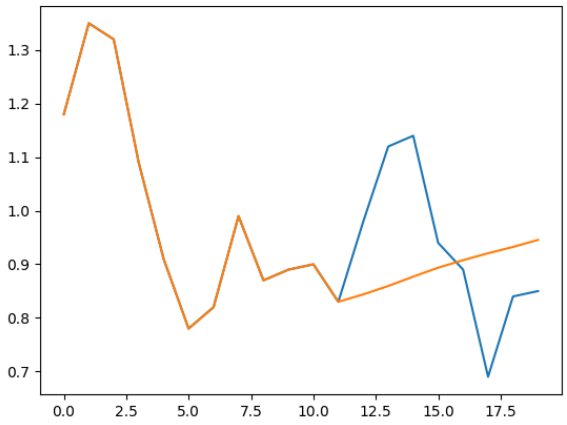


Fig. 5: Visualization of Result Uni-Step LSTM Network with 1 Neuron Trained over 1 Epochs Forecasting 8 Anomalies

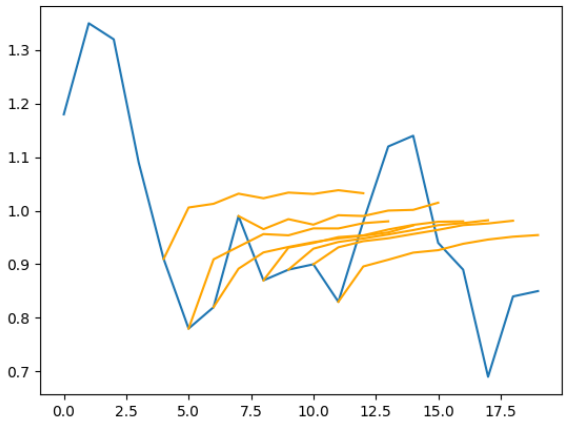


Fig. 6: Visualization of Result for Multi-Step LSTM Network with 16 Neurons Trained over 5 Epochs Forecasting 8 Anomalies

TABLE VI: Multi-Step LSTM Network RMSE Results with Various Hyper-Parameters Averaged over 5 Runs Predicting the Last *8* Month(s)

Months	Epochs	Neurons	Avg. RMSE
8	2	1	0.165
8	5	1	0.139
8	5	2	0.135

corresponding targets of size n. Since we want to predict the **8** last elements, we set $\boldsymbol{n}$**=8**. The method yielded better results than the method in the second experiment and can be seen in Table VI and a visualization of the best result can be seen in Figure 6.

VI. Discussion

A. *Experiment 1*

The results of this experiments were really good and it seems like this was a fairly trivial task for the LSTM network. With only **2** epochs the results were almost as good as they could get, even though the best result was achieved with 5 epochs. The difference was, however, negligible so it could be possible that the number of runs were too few for detecting the near true average for this the value sufficiently. We can see the prediction made was nearly perfect from the visualization in Figure 4 with and RMSE of **0.004** as seen in Table IV. After approximately **5** epochs the network began overfitting to the training data and we would get worse results.

B. *Experiment 2*

The results in this experiment was interesting. Predicting two anomalies was still a relatively easy task for the LSTM model. Over **5** epochs with **16** neurons the network performed relatively good with RMSE of **0.027** as we can see in Table V. Nevertheless, after the second anomaly prediction the network started to struggle with the prediction and reducing the number of epochs and neurons to **1** produced the best results. As we see from Figure 5 the best result for predicting 8 anomalies looks like the network is not fit at all. This makes sense since the weights are not updated too much yet due to the few epochs of training, which, in other words, makes the model underfitted. By conducting a little extra experiment predicting 8 anomalies over **5** epochs with **16** neuron model we get the an RMSE of **0.200** and we can visualize the result as in Figure 7. Comparing the two results, we can conclude that neither results are very good. The LSTM struggles with finding a fit that yield good results which means that the model cannot predict properly based on the training data, which means that we do not have enough good data to train on or that the data is not predictable, or the method we are using is not working very well.

C. *Experiment 3*

By looking at the third experiment compared to the second we notice a significant improvement in performance. A reasonable assumption of why the result improved is that we are no longer using predicted data to make further predictions. Instead we are predicting a window of 8 elements at a time from all the previous values. Yet, this method did also not manage to predict the anomalies correctly. We can see in Figure 6, that the orange lines are trying to connect with the blue line in the future which then makes the prediction correct for the **8th** anomaly in the future if they hit it with the tip. They are sometimes close but none of them actually hit. They best result as we see in Table VI, yielded an RMSE of **0.135** which is still not good but it is at least better than the result from the second experiment. After **5** epochs the model started to overfit. We can at least detect the trend in the data since every prediction is higher than the value predicting from. So we can expect a steady increase in the future which seems correct from

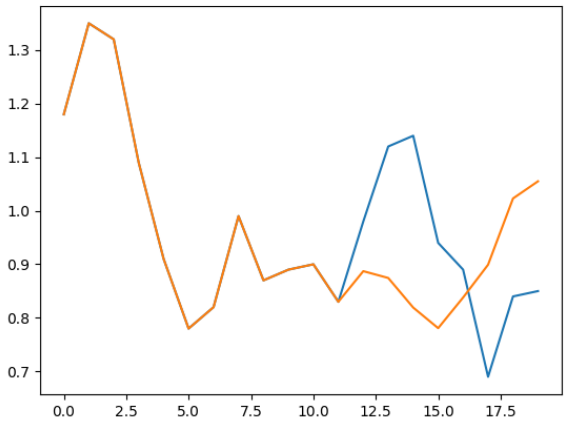


Fig. 7: Visualization of Result Uni-Step LSTM Network with 16 Neuron Trained over 5 Epochs Forecasting 8 Anomalies

the data and also that the mean global temperature is in fact increasing every year.

VII. Conclusion

From the experiments predictions with the LSTM model seems fit for short-term predictions, no more than 2 temperature anomalies in succession, and also fit for detecting trends in the data. We can also conclude that the data we were trying to predict was difficult to recognize a particular sequence pattern from. Mean global temperature anomalies proved to be a real-world data set which was difficult to make predictions from. The data reflect countless of variable as the *Land+Ocean* variable. We can from this already know that the pattern in how the single variable behaves will be very difficult, if not impossible, to derive from a single variable and we cannot expect very good results, or at least not good results for predicting values far ahead in time. We could probably produced better results if we had access to many more variables to predict from and used a multivariate prediction method instead of the univariate method. Nevertheless, LSTMs seem promising if used on data without too much white noise [10], and from what we can see from what others have been capable of producing with them, we can safely say that an LSTM network is a powerful tool in many tasks.

References

[1] A. Graves, "Generating sequences with recurrent neural networks," *CoRR*, vol. abs/1308.0850, 2013. [Online]. Available: http://arxiv.org/abs/1308.0850

[2] M. Sundermeyer, R. Schlüter, and H. Ney, "Lstm neural networks for language modeling," in *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

[3] K. Gregor, I. Danihelka, A. Graves, and D. Wierstra, "DRAW: A recurrent neural network for image generation," *CoRR*, vol. abs/1502.04623, 2015. [Online]. Available: http://arxiv.org/abs/1502.04623

[4] D. Eck and J. Schmidhuber, "Finding temporal structure in music: Blues improvisation with lstm recurrent networks," in *NEURAL NETWORKS FOR SIGNAL PROCESSING XII, PROCEEDINGS OF THE 2002 IEEE WORKSHOP*. IEEE, 2002, pp. 747–756. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.141.2656

[5] Z. C. Lipton, D. C. Kale, C. Elkan, and R. C. Wetzel, "Learning to diagnose with lstm recurrent neural networks." *CoRR*, vol. abs/1511.03677, 2015. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1511.html#LiptonKEW15

[6] Wikipedia, "Markov property," 2017, accessed on 04.10.2017. [Online]. Available: https://en.wikipedia.org/wiki/Markov_property

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[8] G. A. Schmidt and R. B. Schmunk, "Giss surface temperature analysis - monthly mean global surface temperature," 2017, accessed on 15.10.2017. [Online]. Available: https://data.giss.nasa.gov/gistemp/graphs/

[9] Keras, "Keras: The python deep learning library," accessed on 28.09.2017. [Online]. Available: https://keras.io/#keras-the-python-deep-learning-library

[10] Wikipedia, "White noise," 2017, accessed on 27.10.2017. [Online]. Available: https://goo.gl/EXpXpt