Time Series Forecasting with Long Short Term Memory and Recurrent Neural Networks

Saga Abdul Amir, Walid Abdul Jalil, Shadman Ahmed, Michael Luxemburg

Royal Institute of Technology
Deep Learning in Data Science DD2424

Abstract. This paper implements LSTM recurrent neural networks on different time series datasets. The performance of the LSTMs are compared with the performance of ARIMA models which are common in time series forecasting. Results show that the root mean square error (RMSE) is significantly lower for the LTSM models for both simple time series as well as the more complex datasets.

Keywords: LSTM, RNN, Sunspots, ARIMA, Time series, Stock market

1 Introduction

Short-term memory is a vital ability of our brain. Watching a movie or reading a text, we base our understanding of the next information on the previous relevant information and context. Traditional neural networks do not implement this trait. Instead of letting information persist, traditional approaches calculate everything from scratch over and over. And instead of being general, earlier methods for learning problems related to sequential data has been targeting only specific problems [1]. As the design of neural networks takes inspiration from the brain, it is reasonable to strive for designing a neural network that imitates the ability of short term memory.

Recurrent neural networks (RNNs) is a solution to this problem. They implement loops, enabling information to persist. The loops allow the network to use the same information over again, until a certain condition tells it to forget the information. Standard RNNs are however not able to store information for long. They do not scale to long term dependencies. Consider a text where we want to predict the next word using our short term memory of the previous words in the text. If only the words in the very same sentence is needed, RNNs perform well. However, there may be situations where information much further back in the text is needed. There are fundamental reasons to why RNNs do not perform well in these situations [3]. This is a major reason for the development of long short-term memory (LSTM) RNNs. LSTMs are furthermore more easily optimized than standard RNNs [2].

LSTMs have been implemented to advance the research in many fields, such

as speech synthesis, language modeling and translation, handwriting recognition and protein structure prediction [1]. They were introduced by Hochreiter and Schmidhuber in 1997 [4], who designed it with the main purpose of handling long-term dependencies. Today there exists a plethora of variants of this network.

LSTMs consist of a chain of repeating sections of the neural network. This is what we call the loops, even though they are unraveled. In traditional RNNs, the structure of the repeating section is simple, and may consist of only one neural network layer. LSTMs allows for a much more complex structures. There may be several layers interacting in different ways. Central aspects are the cell state and information gates. [5] The exact functionality of the LSTM used is described later in the report. Figure 1 visualizes the principle of a LSTM.

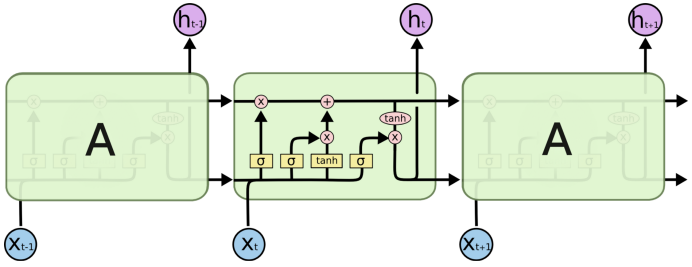


Fig. 1. Principle of LSTM [5]. Each line is a vector, from the output of one node to the inputs of others. The pink circles are point-wise operations, e.g. addition, and the yellow boxes are learned neural network layers.

In this paper, we apply LSTM to three different time series datasets, two of which are naturally occurring phenomena and are thus easier to predict due to a well defined auto-correlation. The third dataset is the daily closing average of the S&P 500 for the last decade, a considerably more complex time series dataset. For each of the datasets, we slice the data and apply the LSTM with Keras, a high-level neural networks API in Python and capable running on top of TensorFlow. We then fit an autoregressive integrated moving average (ARIMA) model for each datasets in the programming language R. Finally, we calculate the average root mean square error (RMSE) for both the neural networks and the ARIMA models through back-testing.

2 Background

2.1 Time series, autocorrelation and ARIMA models

As stated in the introduction, recursive neural networks and LSTMs are intimately related to sequences and lists. Especially sequences involving autocorrelation. Autocorrelation is the presence of correlation between a sequence and lagged (prior) versions of itself. Time series is precisely such a sequence. A time series is a sequence taken at successive equally spaced points in time. It is therefore an autocorrelated sequence in discrete-time yet can still be seen as a random variable. In a time-series, an output at a certain time step is dependent on the output of the previous time-step. Before exploring how great LSTMs are at forecasting such sequences, it is important to highlight other relatively successful approaches to time series prediction and among these approaches, one of the most well-known methods is the autoregressive integrated moving average (ARIMA) models.

If a time series is a random variable of the form described above, it can be viewed as a combination of signal and noise. An ARIMA model can then be viewed as a filter that tries to separate the signal from the noise. ARIMA models are, in theory the most general class of models for forecasting time series. The autoregressive (AR) part of ARIMA means that the variable that is evolving with time is regressed on it's own prior values. The moving average (MA) part, means that the regression error is a linear combination of error terms whose values occurs at present and previous time in the past. Finally the integrated (I) part means you subtract current values from past ones to make sure that process is stationary.

Given a time series of data X_t where t is an integer index and X_t are real numbers, an ARIMA model is given by:

$$\left(1 - \sum_{i=1}^{p'} \alpha_i L^i\right) X_t = \left(1 + \sum_{i=1}^{q} \theta_i L^i\right) \epsilon_t \tag{1}$$

where L is the lag operator, the α_i are the parameters of the autoregressive part of the model, θ_i are the parameters of the moving average part and ϵ_t are the error terms. All in all, the AR part is called the long term filter and the MA part is called the short term filter.

2.2 LSTMs for speech recognition and text generation.

Whilst LSTMs are good at time series forecasting, they have been used to solve a wide range of tasks. These include speech recognition, text generation and text sentiment analysis. In text generation and sentiment analysis, raw text is tokenized, meaning that each word is assigned an integer. Next, those integer tokens are converted into real-valued vectors. Those real-valued vectors are then

used as input in the LSTM cells. Depending on the input, say a few words, the output could be seen as "suggesting" the next word. While in reality is it simply the most probable real-valued vector that comes next in the sequence. In the case of sentiment analysis, such as classifying whether a text review is positive or not, the output is put through a sigmoid function that outputs a number between 0 and 1.

In the domain of speech recognition, Graves et al (2013), published a paper called "Speech recognition with deep recurrent neural networks". They trained deep Long Short-term Memory RNNs and achieved (at the time) the lowest recorded test set error on the TIMIT phoneme recognition benchmark. The TIMIT corpus is a famous data set that contains speech data used for development and evaluation of automatic speech recognition systems. RNNs are inherently deep in time, since their hidden states is a function of all previous hidden states. What Graves et al did, was to investigate whether RNNs would benefit from depth in space instead. They combined deep and bidirectional LSTM RNNs with end-to-end training and achieved state-of-the-art results.

3 Datasets

In this study, we have three datasets of the type time-series, that will be used in the LSTM neural networks and in the ARIMA models. The first dataset is the famous Sunspots dataset. A monthly time series dataset taken from the U.S. National Oceanic & Atmospheric Administration (NOAA). The dataset contains monthly observations from 1749 - 2013. It's complexity is described as moderate. The second dataset contains the daily average minimum temperature of Australia over 10 years. Finally the third dataset is the most complex dataset. It contains the daily closing average of the S&P 500 over a decade (downloaded from Yahoo Trading). Table 1 contains descriptions for each dataset.

Data sets			
	Name	Time Interval	Complexity
Dataset 1	Sunspots	Monthly-3177 steps	Moderate
Dataset 2	Daily Min Temp. Australia	Daily-3650 steps	Simple/Moderate
Dataset 3	S&P 500 Closing Average	Daily-2616 steps	Very volatile

Table 1. Descriptions of each of the data sets.

4 Approach

4.1 The problem of long term dependencies

A normal recurrent neural network (RNN), takes in some input x_t and outputs a value h_t. Then a loop passes from one step of the network to the next. In

reality, it is easier to think of an unrolled neural network where the components can be seen as multiple copies of themselves. Fig 2 shows how an RNN would look when "unrolled".

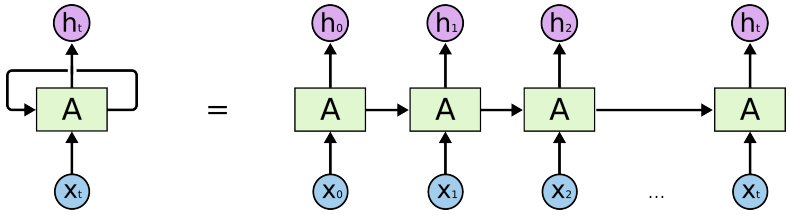
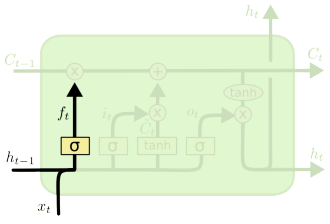


Fig. 2. Depiction of an unrolled RNN.

RNNs however, have problems with long-term dependencies. If you are trying to predict the next word after the following sentence: 'I am British, therefore I speak ...', an RNN might recognize that a language should be the next output. However, because it has problems with long-term dependencies, it cannot put it in context of the word 'British'. The output might therefore be a language other than English. The gap between British and speak is too large. The larger the gap, the worse the standard RNN performs. LSTMs were specifically designed to avoid this problem to a certain degree.

4.2 LSTM-RNN architecture

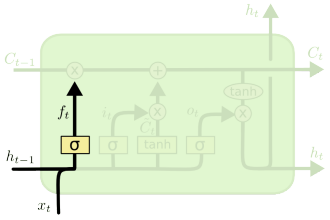
The LSTM contains four interacting layers that together form an LSTM cell. Understanding this cell and it's architecture is crucial to understanding why LSTMs work so well. The first step in an LSTM is to decide what information to throw away and what information to keep. This is done by a sigmoid layer called the forget gate layer. It takes the input h_{t-1} from the previous cell state and a new input value x_t and outputs a number between 0 and 1. 0 means forget completely whilst the value of 1 means remember it all. Figure 3 shows the forget gate layer and the sigmoid function.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

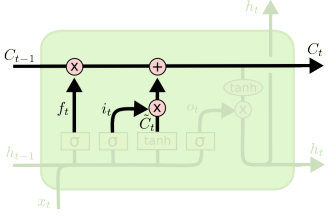
Fig. 3. The forget gate layer

The next step is to decide what information to keep. This is done with another sigmoid function called the input gate, which decides which values to update and stores them in i_t. This is followed by a tanh layer, called the candidate gate layer, which scales the input and creates a vector of the new candidate values, $\tilde{C}_t$. Once both i_t and $\tilde{C}_t$ have been updated, they are point wise multiplied and added to the old cell state C_{t-1} multiplied with the forget gate layer to get the new cell state C_t. This process is depicted in Figures 4 and 5.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

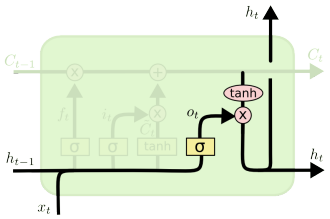
Fig. 4. The input and candidate gate layers.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Fig. 5. Updating the cell state.

The final stage involves deciding what to output into the next cell state. The old output h_{t-1} and the new input x_t is put trough another sigmoid layer, o_t, the updated cell state C_t is then scaled -1 and 1 with a tanh function and multiplied with o_t to create the new output h_t. As shown on Figure 6.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Fig. 6. Deciding the output h_t.

Each of the layers in the cell states have their own weight matrices and biases that are updated iteratively with back propagation using the loss function MAE (Mean Absolute Error). Adam optimizer was used, which is an extension of stochastic gradient descent.

5 Results and Conclusions

Each of the data sets were trained with different number of Epochs as shown in Table 2. The neural network contained two LSTM layers with 50 nodes each. Back testing (which is a form of cross-validation for time series) was done for carried out for each of the datasets. Figure 7 shows the sunspot dataset with both the predicted values and the test values. Figures 8 shows the mean absolute error (MAE) on validation and training, for each slice and how it declined for as the epochs got larger. Figure 9 and 10 shows for the second dataset.

The S&P 500 was considerably harder to predict due to the nature and volatility of a stock market. Even here however, the LSTM did significantly better than the ARIMA models. The predicted vs test data for the stock market is shown in Figure 12. The ARIMA model could not predict the stock market data at all. It is too volatile. Our conclusion is that the LSTMs outperform the ARIMA models at every trial. The ARIMA cannot compete with an LSTM recurrent neural network. This shows the power of recurrent neural networks. You may lose interpret ability but you gain a far higher accuracy. No statistical tests need to be carried out as the RMSE for the LSTM by far outperform those from the ARIMA model. The training time was relatively quick and the neural network was not as complex or deep as could have been. This means that relatively simple LSTM networks can outperform most other time-series models. One does not need powerful computers for this.

LSTM				ARIMA
Sunspots				
Epochs	50	100	150	
RMSE	29.0148	26.0766	27.130643	38.488858
Daily Minimum Temperatures				
Epochs	10	25	100	
RMSE	2.60159	2.40367	2.60707	5.03364
Daily Closing Average S&P 500				
Epochs	100	300	400	
RMSE	1554.20706	1066.63	821.31848	840.7022
Normalized Data/RMSE	0.0047417			0.052462

Table 2. RMSE results from both the LSTM and the ARIMA for each dataset and with different epochs.

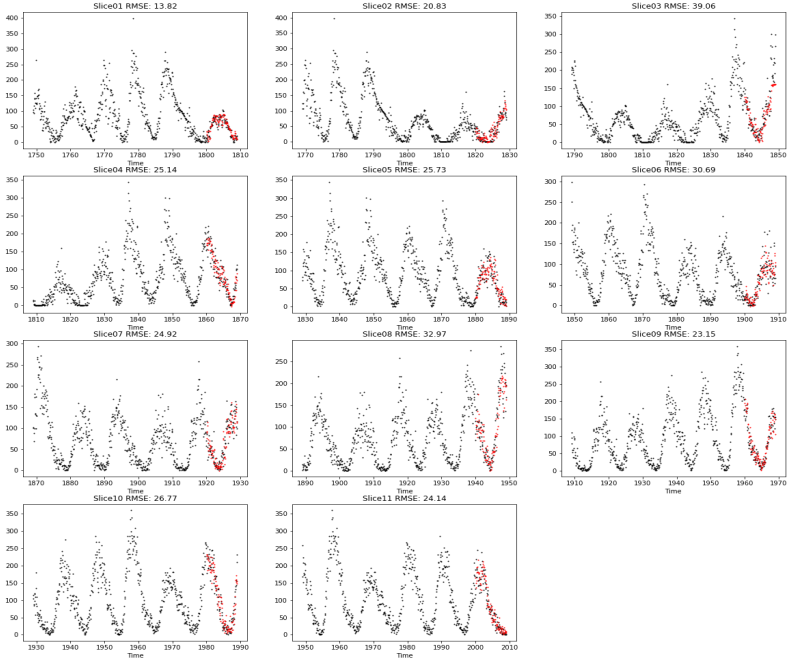


Fig. 7. Backtesting crossvalidation on Sunspots dataset

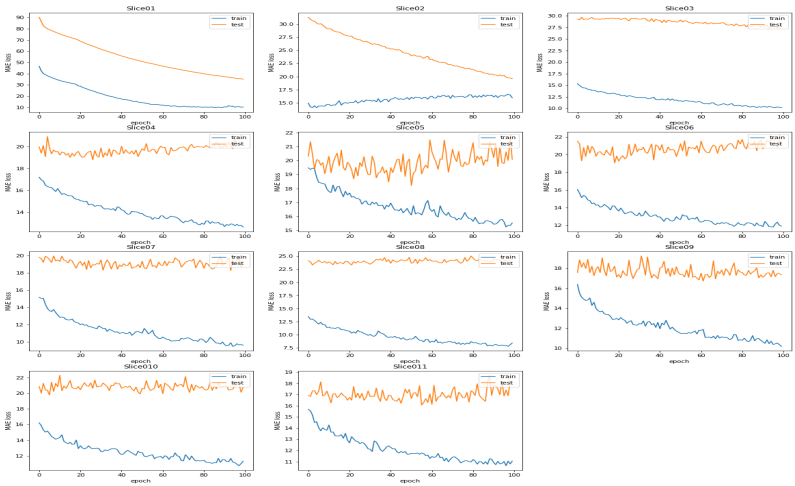


Fig. 8. MAE loss on training and validation Sunspots dataset

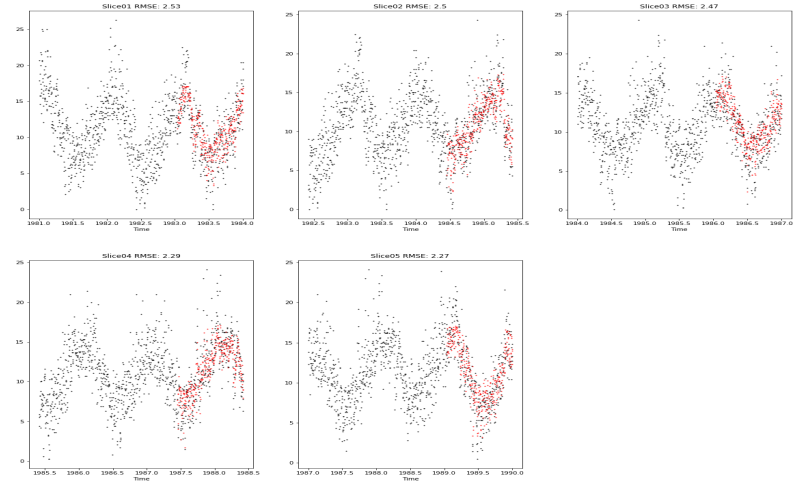


Fig. 9. Backtested Predictions on Daily Min Temp. Australia dataset

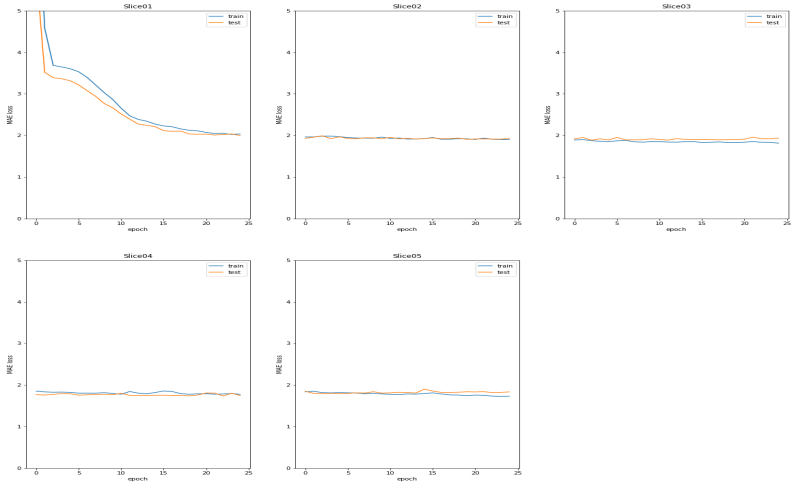


Fig. 10. MAE loss on training and validation Daily Min Temp. Australia dataset

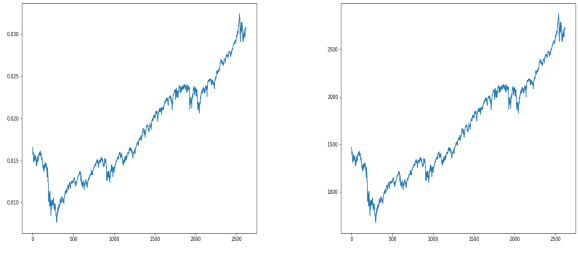


Fig. 11. Normalized and not normalized data, Daily Closing Average S& P 500

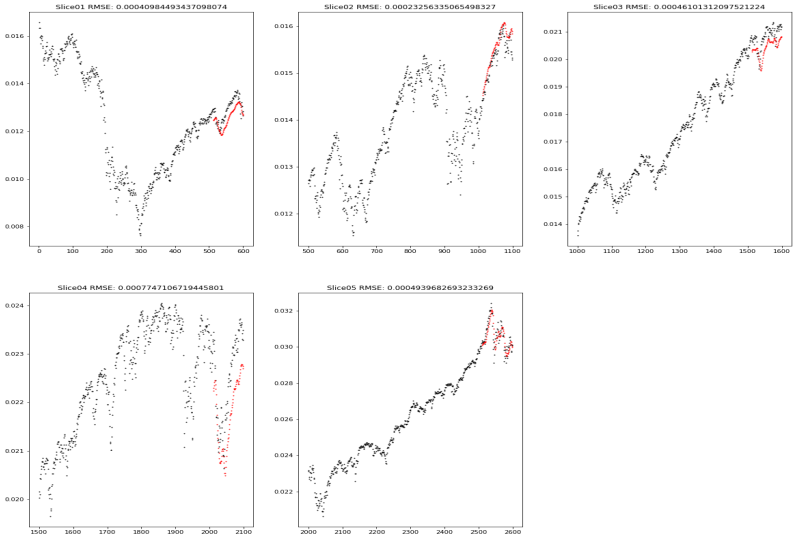


Fig. 12. Backtested Predictions on normalized Daily Closing Average S& P 500

References

1. Klaus Greff, Rupesh K. Srivastava, Jan Koutnk, Bas R. Steunebrink, Jrgen Schmidhuber, 2016, *LSTM: A Search Space Odyssey*, https://ieeexplore.ieee.org/abstract/document/7508408/

2. S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, S. C. Kremer, J. F. Kolen, 2001, *Gradient flow in recurrent nets: The difficulty of learning long-term dependencies* USA:IEEE Press

3. Josef Hochreiter, 1991 *DIPLOMARBEIT IM FACH INFORMATIK, Untersuchungen zu dynamischen neuronalen Netzen*, http://people.idsia.ch/ juergen/SeppHochreiter1991ThesisAdvisorSchmidhuber.pdf

4. Josef Hochreiter and Jurgen Schmidhuber, 1997, *LONG SHORT-TERM MEMORY*, Neural Computation 9(8):1735-1780, http://www.bioinf.jku.at/publications/older/2604.pdf

5. Colah, 2015, *Understanding LSTM Networks*, http://colah.github.io/posts/2015-08-Understanding-LSTMs/