Long Short Term Memory Multi-Step Time Series Forecasting

Michael Malto   
Faculty of Computing  
Ulster UniverstityJordanstown, Northern Ireland  
malto-m@ulster.ac.uk

*Abstract*—The system designed within is an exploration of how changing variables and hyper parameters will affect the learning rate and effectiveness of a Long Short Term Memory network. It is to showcase how training an effective model is a fine balance between many variables and there is no ‘best’ variable. The data set used is Belgium’s Power Consumption.

Keywords—Recurrent, Neural, Networks, Forecasting, Long, Short, Term, Memory

# Overview of neural networks

## Traditional Neural Networks (NN)

Traditional neural networks for image analysis like classifying apples or bananas through deep learning used convolutional filters. Various features are extracted and are used as inputs to the fully connected layer, also known as feed forward layer. Usually a probability output such as: 80 percent of being apples and 20 percent chance of being bananas. Information is only fed forward, hence this is known as a feed forward network. This type of network does not have any information from the past.

## Recurrent Neural Netowrk (RNN)

A recurrent neural network is similar to a feed forward network but it can remember the past. It has connections that point backwards. Cases where we model x to predict y, feed forward networks work fine. However, cases where we model y to predict y from previous data points, this is known as autocorrelation and feed forward networks are not enough. Recurrent neural networks work well for these types of sequential information

Fig 1 Recurrent Neural Network and Feed Forward Neural Network

There are many classification of recurrent neural networks:

* One to one which takes one input and produces a single output such as image classification.
* One to many which takes one input and produces many outputs such as image captioning.
* Many to one which takes many inputs and produces one output such as sentiment analysis.
* Many to many which takes in many inputs and produces many outputs such as language translation.

## Limitations of Recurrent Neural Networks

While recurrent neural networks work well for shorter sequences, longer sequences cannot be processed by recurrent neural networks. They can suffer from Exploding Gradient where there is an exponential increase the model weights from accumulated errors. Similarly, they can suffer from Vanishing Gradient where the gradients become too small and the weights are not changed significantly.

## Long Short Term Memory

Long short term memory is designed to overcome the limitations outlined above and is a type of recurrent neural networks. They are able to process longer sequences because they have internal state which can be used to store information from the past. They have a forget gate which controls what information from the past is discarded. The input gate controls what information is stored in the cell state. Sigmoid function is used to remember or forget information and tanh is used to overcome the vanishing gradient problem.

Diagram, schematic

Description automatically generated

Fig 2 RNN cell compared with LSTM cell

# Problem domain

Given the amount of data available online, the author has chosen to use the power consumption [1]. of Belgium from 2015 to 2020.

Chart, histogram

Description automatically generated

Fig 3 Belgium Power Consumtion in Megawatts, from 2015 to 2020.

The dataset is a univariate time series of the power consumption taken every 15 minutes. The problem can be framed as, given recent power consumption, can we predict the expected power consumption in the future. For example, given the last week of data, can the next week be predicted. This is referred to as a multi step time series forecasting problem.

# Design and development

## Network Design

The LSTM network will be a many to many type mentioned above. There will be multiple inputs and it will have multiple outputs. This type of model would be useful in planning for future power consumption. The author has chosen to model the following, given the last 8 hours of data, predict the next 8 hours. Meaning there will be 8 inputs and 8 outputs. The model will have an Encoder Decoder LSTM architecture. The encoder will be a hidden LSTM layer that takes the input and outputs a vector. The decoder will then be another hidden LSTM layer connected to a Dense layer before the output.

Icon

Description automatically generated

Fig 4 Many to many LSTM [2].

## Data Preparation

As mentioned, the dataset has samples every 15 minutes. Therefore to fit the model described in the Network Design, every 4th sample is taken to take every hour. The n\_input variable refers to the timestep. This means that there will be 8 hours of input, and 8 hours of output.

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

Fig 5 Taking the samples every hour.

After X is reshaped as required, the data was split into 3 sets. 70% of the total data is used to train the model, 15% is used to validate the data, and the remaining 15% is used to test the data. Testing the model on never before seen data is important to verify that the model can generalize and predict effectively.

Text

Description automatically generated

Fig 6 Splitting the data into training, validation and testing data.

The resulting shape of the dataset split is as follows Each shape is in the format of (samples, timesteps, features).

Text

Description automatically generated

Fig 7 Resulting shape after splitting dataset.

## Developing the model

The LSTM network described in the Network Design section is implemented as Fig 8 illustrated. An LSTM layer as the encoder is defined with the ReLU activation function and the desired input shape. The repeat vector is the input sequence repeated, once for each timestep in the output sequence.

Text

Description automatically generated

Fig 8 Defining the model

The decoder is implemented as an LSTM layer with ReLU function again, this time with return sequences set as True. The TimeDistributed wrapper allows the layer to be used for each time step from the decoder. The dropout will randomly set input units to 0 which helps prevent overfitting during the training of the model.

Text

Description automatically generated



Fig 9 Setting the learning rate and compiling the model.

The optimizer Adam was chosen as an optimization algorithm and is an extension of the stochastic gradient descent. It combines both Adaptive Gradient Algorithm (AdaGrad) and Root Mean Squared Propagation (RMSProp). [3]. Mean Squared Error, or MSE was chosen as the loss function as it is most common for regression. The model is then compiled in Fig 9.

Text

Description automatically generated

Fig 9 Summary of the model

# Training the model

The model is then trained with the training set of data. The validation set was passed in the validation data parameter. The model is only trained on the training set, the validation set is used to judge how well the model can predict on data that it has not seen before while it is being trained.

Text

Description automatically generated

Fig 10 Training the model

The model is then evaluated which returns the metric values. The metrics are as follows:

* MSE – Mean Squared Error. It is the difference between the models prediction and the actual values. It is then squared then averaged out across the whole dataset. MSE helps ensure that the trained model has no outlier predictions with huge errors. [4]
* MAE – Mean Absolute Error. It is the difference between the models prediction and actual values. Apply the absolute value to the difference then average it out across the whole dataset.
* RMSE – Root Mean Squared Error

# Analyzing performance

## Base Configuration

The configuration used for the following results:

* Verbose: 2
* Epochs: 20. An entire pass of the dataset.
* Batch size: 128. This defines the number of samples to be propagated through the network. The model is updated every time a batch is processed.
* Learning rate: 0.00008
* N\_input: 8, N\_forecast:
* Dropout: 0.02

With the n\_inputs set to 8 and the n\_forecast set to 8 the results show that using the previous 8 hours of data, the model will predict the next 8 hours of data.

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated



Fig 11 Results of the configuration described.

The model above is promising as the loss curves gradually descend indicating that the model is doing well and the learning rate is optimal.

## Adjusting the n\_input configurations

The difference between base configuration and these configurations is that the n\_input is adjusted, everything else is the same. As seen in Fig 12, by changing the n\_input to 4 the loss metrics are higher. The graphs when changing the n\_input is similar, hence the author will only compare the evaluation scores. The higher loss scores indicates that the model is not as precise as the previous configuration.

Fig 12 Results of adjusting n\_input to 4

 Fig 13 Results of adjusting n\_input to 12

By comparing the RMSE it is observed that the n\_input of 8 is better than an input of 4 or 12.

## Adjusting the learning rate

These results use the base configuration but the learning rate is changed accordingly. By changing the learning rate to 0.l we observe the following.

Graphical user interface, application

Description automatically generated

Fig 13 Results of adjusting the learning rate to 0.1

The loss curves drop off drastically indicating that the learning rate is too high. Next, the learning rate is adjusted to 0.000001. As illustrated in Fig 13, the learning loss curves are do not go down enough. This is a result of the learning rate too low, meaning that the model does not make any meaningful changes to the weights. From the results identified from changing the learning rate, it is observed that finding the optimal learning rate relative to the configuration is important. As a learning rate that is too high will cause the model to converge too quickly. A learning rate that is too low will cause the model to progress very slowly.

Diagram, timeline

Description automatically generated

Fig 14 Results of adjusting learning rate to 0.000001

## Adjusting batch size

Using the base configuration but changing the batch size from 128 to 64 yields the following.

Graphical user interface, application

Description automatically generated

Text

Description automatically generated

Fig 15 Results of adjusting batch size from 128 to 64

The RMSE score is very close to the batch size of 128. However the loss curves drop off more significantly compared to Fig 11. Using a batch size of 32 made the training the time for the model considerably longer than a batch size of 128 and 64.

Graphical user interface, diagram, application

Description automatically generated

Text

Description automatically generated

Fig 16 Results of adjusting batch size from 128 to 32

While the train time of a lower batch size was much longer compared to the higher numbers, the MAE and RMSE do not differ much.

## Adjusting prediction size

Using the base configuration but adjusting the n\_forecast time to 3.

Diagram

Description automatically generated with medium confidence

Fig 17 Results of adjusting n\_input from 8 to 3

Text

Description automatically generated

Using a lower forecast time yielded a lower MAE and RMSE score. From this result we can deduce that having a longer sequence to predict a shorter sequence can be considered better. This is because the model has more historical data to look back on and does not have to predict so far in the future.

## Epoch size of 100

Graphical user interface

Description automatically generated with low confidence

Text

Description automatically generated

Fig 18 Results of adjusting the epoch to 100

The base configuration was ran at 100 epochs. We can observe that the MAE and RMSE score is very similar to the base configuration with epoch size of 20. However, the training time was considerably longer compared to the epoch size of 20. From this observation, the author can conclude that increasing the epoch size may not necessarily aid with the effectiveness of the model. A high er epoch size may even have diminishing returns.

# Analysis Conclusion

From the analysis conducted, we can conclude that by adjusting hyper parameters such as learning rate we can change how effective the model learns. Comparing the scores such as MAE and RMSE the results from different configurations can be compared. From the experiments conducted, the author has a greater appreciation and understanding of neural networks, recurrent neural networks and long short term memory networks. Compared to models that are used in live production it is safe to say that there are much more variables and hyper parameters that must be tuned in order to have an effective model.

# Applications of LSTMs

Extending and improving the base configuration can have many applications in the real world. For example, a model such as the base configuration can be used by electricity providers to predict trends and patterns so they can prepare for future supply and demand.

As briefly mentioned earlier there are many more applications to LSTMs. These include but are not limited to Time series prediction, speech recognition, rhythm learning.

# References

1. Francois Raucent, “Western europe power consumption” in Kaggle: https://www.kaggle.com/datasets/francoisraucent/western-europe-power-consumption.
2. Andrej Karpath, “The Unreasonable Effectiveness of Recurrent Neural Networks” in blog. http://karpathy.github.io/2015/05/21/rnn-effectiveness/
3. Jason Brownlee PhD, “Gentle Introduction to the Adam Optimization Algorithm for Deep Learning” in blog. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
4. George Seif, “Understanding the 3 most common loss functions for Machine Learning Regression” in blog. https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression-23e0ef3e14d