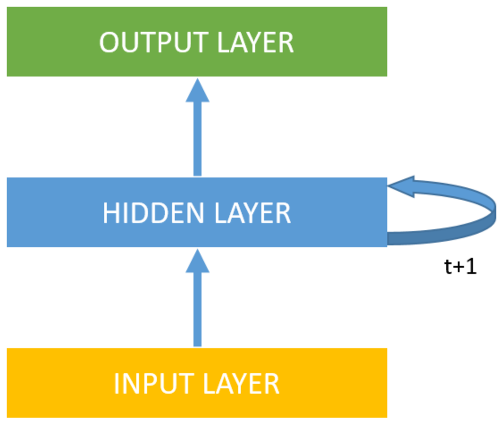

Phase II Report: Using Recurrent Network for Data Regression and Prediction

Independent Study

MENG CHEN
August 4th 2016
CSCI 8991

Objective

The objective of the phase II independent study project is to study the theory, implementation and the practical usage of a recurrent neural network. The theory covers the characteristics of various recurrent neural network model, mathematical modeling and error propagation back in time. The implementation focuses on the construction of a recurrent neural network using Torch 7. The practical usage covers application aspect of the recurrent neural network.

Problem Definition

The phase I focused on classification using feedforward network, so, the phase II will focus on regression instead. In this study, a recurrent neural network will be trained and tested using an artificial dataset. The artificial dataset is generated using lua script and it will contain signal and noise. The signal is the pattern that the recurrent neural network will need to learn; the noise is added to mimic real dataset and check potential overfitting. Ideally, we would want the network to learn only the pattern from a noisy set of data. For dataset used, only one input is fed into the network at each time step, and the network has to use previous "memories" to predict the future signal values and trends.

Theory and Methodology

The recurrent neural network shares many similarities with the feedforward neural network. They are constructed of the same elements (neuron, connection, weights, bias) and they all use backpropagation method in training. The key difference between a feedforward and recurrent neural network is that the recurrent neural network can not only learn multi-dimensional patterns, but also temporal pattern. This very useful to process information encoded in sequences, such as video and audio data, where a single frame or frequency has no meaning, but a structured sequence of frames and frequencies could have meaning. The recurrent neural network achieves this by connecting layers between different time steps, usually back to itself in the next time step. Following schematics shows the differences in the connection of a feedforward and recurrent network (feedforward on the left and recurrent on the right):

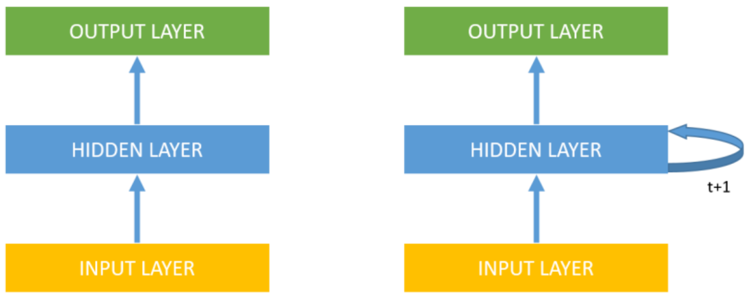

Figure 1 Feedforward and Recurrent Network Comparison

Another way to graphically represent a recurrent neural network is to make the network at each time step its own separate network:

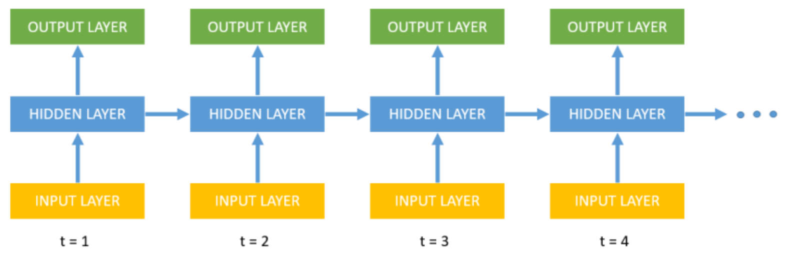

Figure 2 Alternative Representation of Recurrent Neural Network

A vanilla recurrent structure will just add the previous hidden layer activations to itself in the next layer. As the previous graph has illustrated, an input at t=1, not only affects the output at that time step, but also all the subsequent time steps. This is problematic during the backpropagation because we must propagate back deeply into the time, which causes the vanishing gradient problem [1]. The following derivation shows the two steps backpropagation of following simple recurrent network:

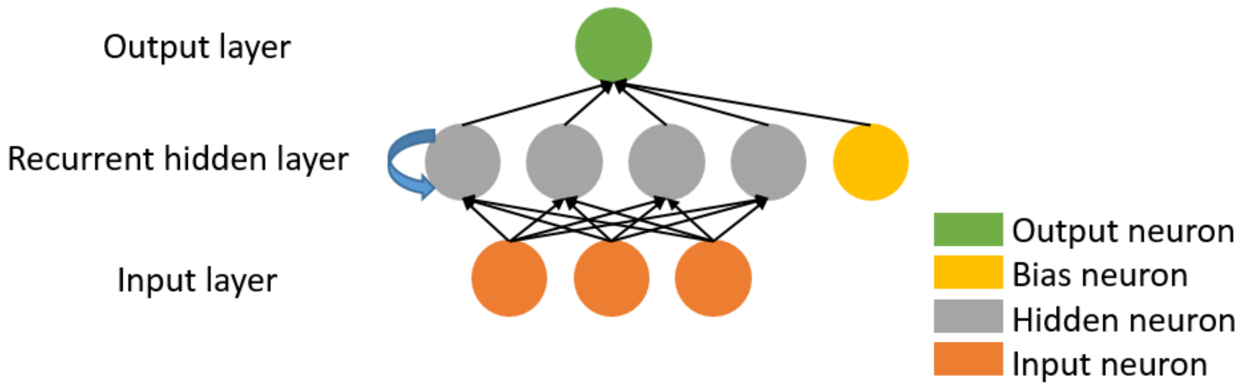

Figure 3 Visualization of the Sample Recurrent Network for Analysis

The activation of input layer:

$$input = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ 1 \end{bmatrix}$$

The connection weights of the feedforward connection and recurrent connection (with subscript for source, superscript for target and comma separated value for time step):

$$w_v^h = \begin{bmatrix} w_{v1}^{h1} & w_{v2}^{h1} & w_{v3}^{h1} & w_b^{h1} \\ w_{v1}^{h2} & w_{v2}^{h2} & w_{v3}^{h2} & w_b^{h2} \\ w_{v1}^{h3} & w_{v2}^{h3} & w_{v3}^{h3} & w_b^{h3} \\ w_{v1}^{h4} & w_{v2}^{h4} & w_{v3}^{h4} & w_b^{h4} \end{bmatrix} \qquad w_{h,t-1}^{h,t} = \begin{bmatrix} w_{h1,t-1}^{h1,t} & w_{h2,t-1}^{h1,t} & w_{h3,t-1}^{h1,t} & w_{h4,t-1}^{h1,t} & w_b^{h1,t} \\ w_{h1,t-1}^{h2,t} & w_{h2,t-1}^{h2,t} & w_{h3,t-1}^{h2,t} & w_{h4,t-1}^{h2,t} & w_b^{h2,t} \\ w_{h1,t-1}^{h3,t} & w_{h2,t-1}^{h3,t} & w_{h3,t-1}^{h3,t} & w_{h4,t-1}^{h3,t} & w_b^{h3,t} \\ w_{h1,t-1}^{h4,t} & w_{h2,t-1}^{h4,t} & w_{h3,t-1}^{h4,t} & w_{h4,t-1}^{h4,t} & w_b^{h4.t} \end{bmatrix}$$

The hidden recurrent layer activation at time step t is computed by the sum of feedforward and feedback connections:

$$h_t = \begin{bmatrix} w_{h1,t-1}^{h1,t} & w_{h2,t-1}^{h1,t} & w_{h3,t-1}^{h1,t} & w_{h4,t-1}^{h1,t} & w_b^{h1,t} \\ w_{h1,t-1}^{h2,t} & w_{h2,t-1}^{h2,t} & w_{h3,t-1}^{h2,t} & w_{h4,t-1}^{h2,t} & w_b^{h2,t} \\ w_{h1,t-1}^{h3,t} & w_{h2,t-1}^{h3,t} & w_{h3,t-1}^{h3,t} & w_{h4,t-1}^{h3,t} & w_b^{h3,t} \\ w_{h1,t-1}^{h4,t} & w_{h2,t-1}^{h4,t} & w_{h3,t-1}^{h4,t} & w_{h4,t-1}^{h4,t} & w_b^{h4.t} \end{bmatrix} \begin{bmatrix} h_{1,t-1} \\ h_{2,t-1} \\ h_{3,t-1} \\ h_{4,t-1} \\ 1 \end{bmatrix} + \begin{bmatrix} w_{v1}^{h1} & w_{v2}^{h1} & w_{v3}^{h1} & w_b^{h1} \\ w_{v1}^{h2} & w_{v2}^{h2} & w_{v3}^{h2} & w_b^{h2} \\ w_{v1}^{h3} & w_{v2}^{h3} & w_{v3}^{h3} & w_b^{h3} \\ w_{v1}^{h4} & w_{v2}^{h4} & w_{v3}^{h4} & w_b^{h4} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ 1 \end{bmatrix}$$

Finally, the hidden layer is connected to the output layer:

$$w_{h2}^{o} = [w_{h2,1}^{o} \quad w_{h2,2}^{o} \quad w_{h2,3}^{o} \quad w_{h2,4}^{o} \quad w_b^{o}] \qquad output = [o]$$

The error using MSE (Mean Square Error) measure can be computed as following:

$$E = (o - t)^2$$

And the error gradient with respect to the output neuron can be expressed as:

$$\delta^o = 2(o - t)$$

, where t is the target output activation. The gradient with respect to the previous connection weights can be expressed as following:

$$\frac{\partial E}{\partial w_h^o} = h\delta^o = \begin{bmatrix} 2h_1(o-t) \\ 2h_2(o-t) \\ 2h_3(o-t) \\ 2h_4(o-t) \\ 2(o-t) \end{bmatrix}$$

Then, the gradient with respect to the hidden layer activation can be expressed as following:

$$\delta_h^o = \frac{\partial E}{\partial h} = [w_{h1}^o\delta^o \quad w_{h2}^o\delta^o \quad w_{h3}^o\delta^o \quad w_{h4}^o\delta^o \quad w_b^o\delta^o]$$

So far, the backpropagation steps have been the same as the one for the feedforward included in the phase I report. The different for the backpropagation starts here, since a recurrent layer not only needs to propagate back in the network, but also back in time. We will start with the propagation back in the network, and the gradient with respect to the connection weights can be expressed as following:

$$\frac{\partial E}{\partial w_v^{h,t}} = \begin{bmatrix} v_1\delta_{h1}^o & v_1\delta_{h2}^o & v_1\delta_{h3}^o & v_1\delta_{h4}^o \\ v_2\delta_{h1}^o & v_2\delta_{h2}^o & v_2\delta_{h3}^o & v_2\delta_{h4}^o \\ v_3\delta_{h1}^o & v_3\delta_{h2}^o & v_3\delta_{h3}^o & v_3\delta_{h4}^o \\ \delta_{h1}^o & \delta_{h2}^o & \delta_{h3}^o & \delta_{h4}^o \end{bmatrix}$$

Then, we can do the propagation back in time and express the gradient with respect to the feedback connection weights as following:

$$\frac{\partial E}{\partial w_{h,t-1}^{h,t}} = \begin{bmatrix} h_{1,t-1}\delta_{h1}^o & h_{1,t-1}\delta_{h2}^o & h_{1,t-1}\delta_{h3}^o & h_{1,t-1}\delta_{h4}^o \\ h_{2,t-1}\delta_{h1}^o & h_{2,t-1}\delta_{h2}^o & h_{2,t-1}\delta_{h3}^o & h_{2,t-1}\delta_{h4}^o \\ h_{3,t-1}\delta_{h1}^o & h_{3,t-1}\delta_{h2}^o & h_{3,t-1}\delta_{h3}^o & h_{3,t-1}\delta_{h4}^o \\ h_{4,t-1}\delta_{h1}^o & h_{4,t-1}\delta_{h2}^o & h_{4,t-1}\delta_{h3}^o & h_{4,t-1}\delta_{h4}^o \\ \delta_{h1}^o & \delta_{h2}^o & \delta_{h3}^o & \delta_{h4}^o \end{bmatrix}$$

Then, the gradient with respect to the previous hidden layer activation is:

$$\delta^o_{h_{t-1}} = \frac{\partial E}{\partial h_{t-1}} = \left[\sum_{k=1}^{4} w^{t,k}_{t-1,1}\delta^0_{h,k} \quad \sum_{k=1}^{4} w^{t,k}_{t-1,2}\delta^0_{h,k} \quad \sum_{k=1}^{4} w^{t,k}_{t-1,3}\delta^0_{h,k} \quad \sum_{k=1}^{4} w^{t,k}_{t-1,4}\delta^0_{h,k} \quad \sum_{k=1}^{4} w^{t,k}_{t-1,b}\delta^0_{h,k}\right]$$

If we were to propagate the error back in the network in the previous time step, we will get following gradient:

$$\frac{\partial E}{\partial w_v^{h,t-1}} = \begin{bmatrix} v_1\delta^o_{h_{1,t-1}} & v_1\delta^o_{h_{2,t-1}} & v_1\delta^o_{h_{3,t-1}} & v_1\delta^o_{h_{4,t-1}} \\ v_2\delta^o_{h_{1,t-1}} & v_2\delta^o_{h_{2,t-1}} & v_2\delta^o_{h_{3,t-1}} & v_2\delta^o_{h_{4,t-1}} \\ v_3\delta^o_{h_{1,t-1}} & v_3\delta^o_{h_{2,t-1}} & v_3\delta^o_{h_{3,t-1}} & v_3\delta^o_{h_{4,t-1}} \\ \delta^o_{h_{1,t-1}} & \delta^o_{h_{2,t-1}} & \delta^o_{h_{3,t-1}} & \delta^o_{h_{4,t-1}} \end{bmatrix}$$

$$= \begin{bmatrix} v_1\sum_{k=1}^{4} w^{t,k}_{t-1,1}\delta^0_{h,k} & v_1\sum_{k=1}^{4} w^{t,k}_{t-1,2}\delta^0_{h,k} & v_1\sum_{k=1}^{4} w^{t,k}_{t-1,3}\delta^0_{h,k} & v_1\sum_{k=1}^{4} w^{t,k}_{t-1,4}\delta^0_{h,k} \\ v_2\sum_{k=1}^{4} w^{t,k}_{t-1,1}\delta^0_{h,k} & v_2\sum_{k=1}^{4} w^{t,k}_{t-1,2}\delta^0_{h,k} & v_2\sum_{k=1}^{4} w^{t,k}_{t-1,3}\delta^0_{h,k} & v_2\sum_{k=1}^{4} w^{t,k}_{t-1,4}\delta^0_{h,k} \\ v_3\sum_{k=1}^{4} w^{t,k}_{t-1,1}\delta^0_{h,k} & v_3\sum_{k=1}^{4} w^{t,k}_{t-1,2}\delta^0_{h,k} & v_3\sum_{k=1}^{4} w^{t,k}_{t-1,3}\delta^0_{h,k} & v_3\sum_{k=1}^{4} w^{t,k}_{t-1,4}\delta^0_{h,k} \\ \sum_{k=1}^{4} w^{t,k}_{t-1,1}\delta^0_{h,k} & \sum_{k=1}^{4} w^{t,k}_{t-1,2}\delta^0_{h,k} & \sum_{k=1}^{4} w^{t,k}_{t-1,3}\delta^0_{h,k} & \sum_{k=1}^{4} w^{t,k}_{t-1,4}\delta^0_{h,k} \end{bmatrix}$$

The overall recurrent layer weights are updated by the combined gradient in all time steps:

$$\frac{\partial E}{\partial w_v^{h}} = \frac{\partial E}{\partial w_v^{h,t}} + \frac{\partial E}{\partial w_v^{h,t-1}}\frac{\partial E}{\partial w_v^{h,t-2}} + \cdots$$

Thought, the weights are affected by the errors propagated back in time, the terms quickly become very small as we go back further. This is what the vanishing gradient is referring to. The vanishing gradient makes the network almost impossible to “remember” long term correlations because the error gradient with respect to the historic inputs does not play any significant role in updating the weights. Instead, the network gets updated from the new inputs each time.

The vanishing gradient problem is alleviated by the Long-Short-Term-Memory (LSTM) layer, which uses gates to control the updates of the stored information. Following is an interior view of LSTM recurrent layer:

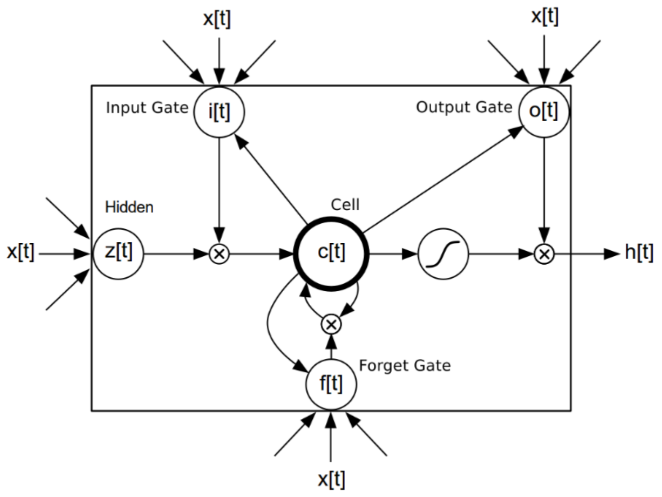

Figure 4 Schematics for the interior of a LSTM cell [2]

A LSTM cell contains three gates: input gate, forget gate and output gate. Input gate controls the amount of information to update to the cell and forget gate controls the amount of memory to contribute to the next state. Many variations of LSTM exist and following schematic is an illustration of an alternative LSTM design, in which, it uses the output as the recurrent feedback instead of the cell state. Regardless those minor difference, the essence of the LSTM is still the same, which is using gate mechanism to control the information update, thus keeping long-term memory.

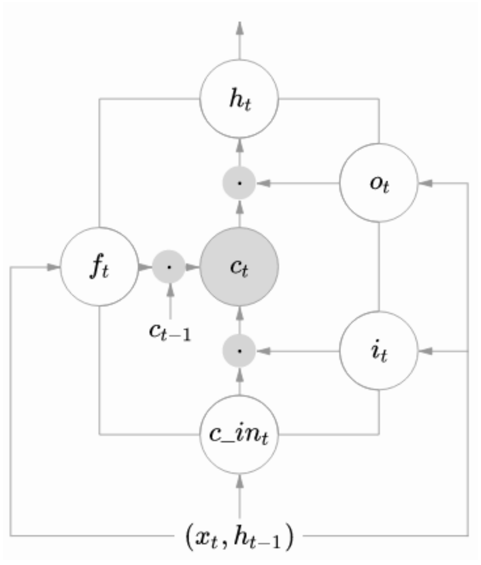

Figure 5 Schematics for an alternative LSTM Updated with Output Values [3]

Detailed cell updating calculation for this LSTM can be found reference [3].

Implementation

This project used Torch7 to implement all the recurrent neural network. A torch rnn library [4] from Element-Research, provided various modules for constructing a recurrent network. The implementation of the LSTM modules is based on the 2013 paper by Alex Graves, Abdel-rahman Mohamed and Geoffrey Hinton. [5]

Dataset

For learning purpose, a small artificial dataset was used to train and test the recurrent network. The data are generated using a Lua script. The script simulates a composite wave function with normal random noise to simulate the real signal. Following are the equations used to generate the artificial dataset:

$$Index = 0.5\sin\left(2\pi\frac{t}{50}\right) + 0.3\sin\left(2\pi\frac{t}{100}\right) + 0.4\sin\left(\frac{45}{180}\pi + 2\pi\frac{t}{125}\right) + 0.5\sin\left(\frac{75}{180}\pi + 2\pi\frac{t}{125}\right) + k \times Noise$$

The noise is assumed to have a normal distribution and is computed using Box-Muller transform:

$$Noise = \sqrt{-2\ln(x)}\cos(2\pi rx)$$

Where x is uniformly distributed between 0 and 1. The variable k is used to control the relative size of the noise, or the signal to noise ratio. In the LSTM script, the dataset will be normalized to center around zero and have the standard deviation of 1. The signal curve in the following graph shows the normalized script generated training series with noise level of 0.4 and the pattern curve represents the series with noise level of 0:

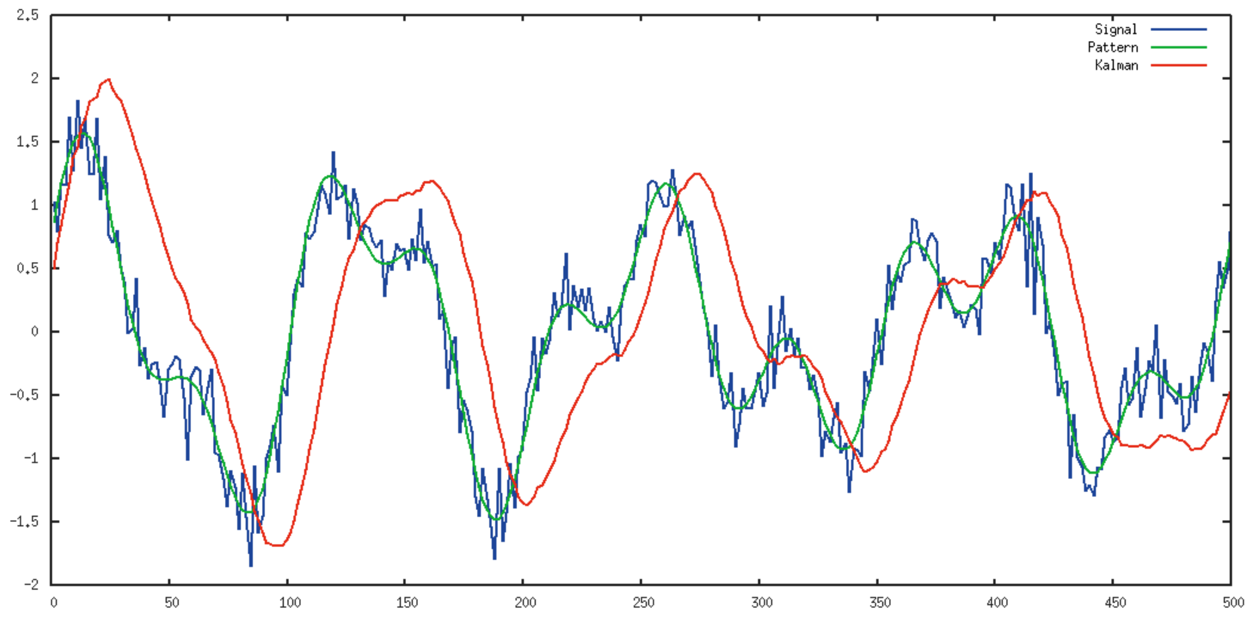

Figure 6 Example of the Artificial Data Series

The Kalman filter works well in many tracking situations to correct the measurements. But when it is used for prediction (using the output of the prediction phase of Kalman filter), the predictor lags behind the actual trend too much to be useful. This is because Kalman filter is agnostic to the underlying pattern and history, thus, it cannot predict the trend ahead of time. We can use a recurrent neural network to extract patterns from the historic data, and use the extracted pattern to predict the future change in a timely manner.

There are two ways we can test the performance of a trained recurrent neural network in this study. The first is to simply compare the prediction to the pattern curve. Because this is an artificially generated dataset, we actually know the true pattern of the data, which is not usually the case for real dataset. The second as well as the more realistic way to test the network is to run it using a set of data which has not been used for training. In this case, it will require us to re-run the lua script to create a new set of data with different noise.

For this study, the recurrent neural network will only take one input, which is the current index value, and it needs to predict future index value. The length of memories (number of time steps backpropagation can go back) allowed for the network will be one of the hyper parameters for optimization.

Results

There are four main areas of recurrent neural network design this study aims to explore. However, given limited time and resources, those explorations were not conducted to the maximum possible depth; instead, the aim is to quick develop all the necessary knowledges in all four areas, and develop a sense of intuition on recurrent neural network designs. With that been said, the following four areas of investigation are listed below:

- How does memory length affect the performance of the network?
- How does dataset size affect the performance of the network?
- How does the network complexity affect the performance of the network?
- How far ahead can we use the network to forecast?

Design

The following schematics shows the design of the recurrent neural network, which has been used as the benchmark for later hyper-parameters comparison studies:

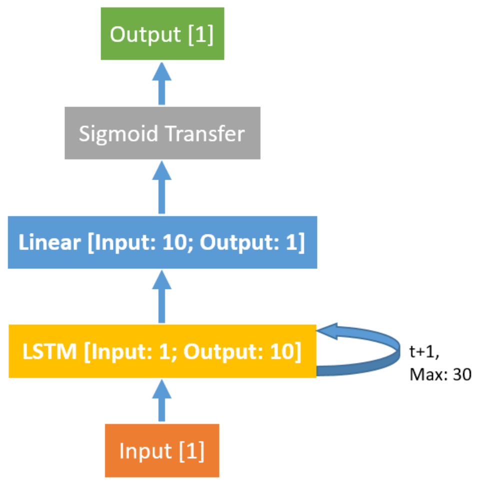

Figure 7 Benchmark Network Design

For each of the later comparison study, a changed will be introduced to the above design. The output differences would help us to understand how the network performance is affect by the change.

Noise

Before starting on any of the specific investigation, it is helpful to look into the effect of noise level on the network performance, because it helps us to understand the results in context. When a noise free series is used to train the network, we get following prediction series, which is named as RNN in the legend:

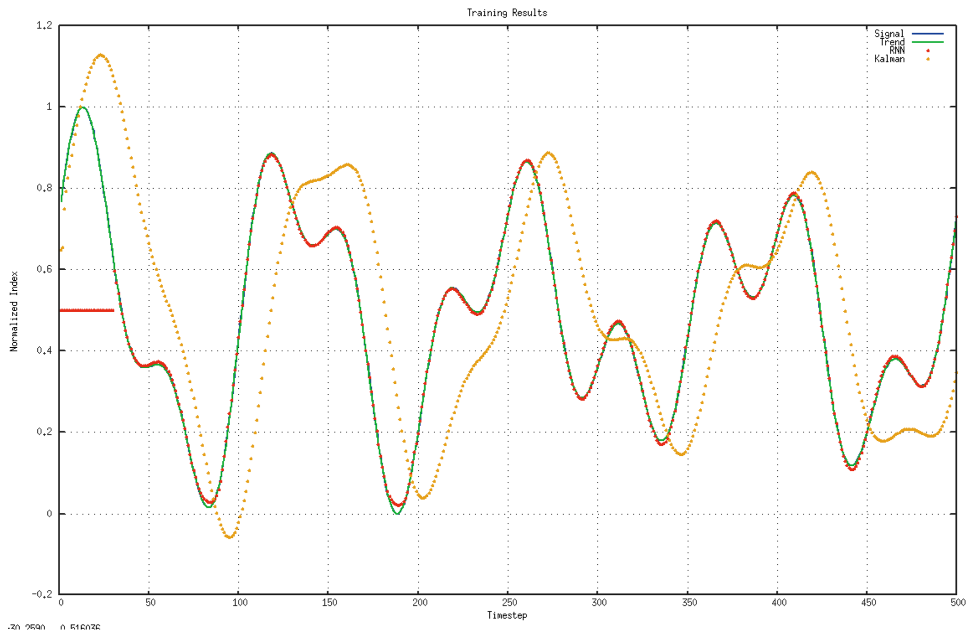

Figure 8 Results of Network Trained using Noise-free Data

In the graph above, the Signal and Trend curves are overlapping, because there is no noise in the dataset. Each of the RNN data point is generated using the inputs from previous 30 historic data point, which is the length of the flat line in the beginning of the RNN series. In the beginning, the network does not have enough historic information to make a precise prediction, thus the prediction is not updated to the actual curve. Nevertheless, we can see the predictions are very accurate (nearly overlapping the actual signal curve) when the noise is turned off.

Now, we turn on the noise in the signal series and train the network again using the noisy dataset. First, the noise level is set to be 0.1 and lead, the number of steps to predict ahead, to be 1:

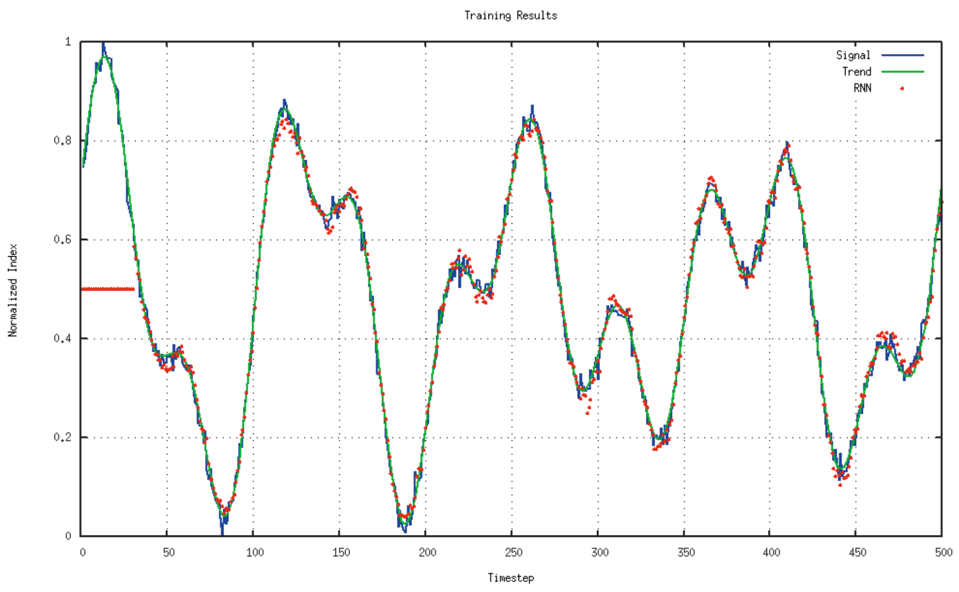

Figure 9 Results of Network Trained using Data with Noise Level 0.1

We can see that the prediction deviates away from the trend curve more than the one trained using a noise free dataset. Nevertheless, the deviation follows the trend closely. In order to measure the performance of the network at predicting the trend, two metrics are considered:

- Mean deviation
- Earning percentage

Mean deviation measures the mean distance of the predicted value from the trend value. The smaller the value is, the better the network is at finding and following the actual trend. Earning percentage is a metrics that emphasizes the direction, and a prediction results are binary, correct or incorrect. A correct prediction means the predicted growth sign matches the actual growth sign and an incorrect prediction means the predicted growth sign mismatches the actual growth sign. For example, if the predicted value is high than the current value and the next trend value is also higher than the current trend value, then, the prediction is correct. The earning percentage is computed as the difference between the correct and incorrect prediction. A 100% earning means the network, predicted the growth sign 100% correctly and 50% earning means that the network predicted the growth sign 75% correctly and 25% incorrectly. Thus, we can plot those metrics the above the training results:

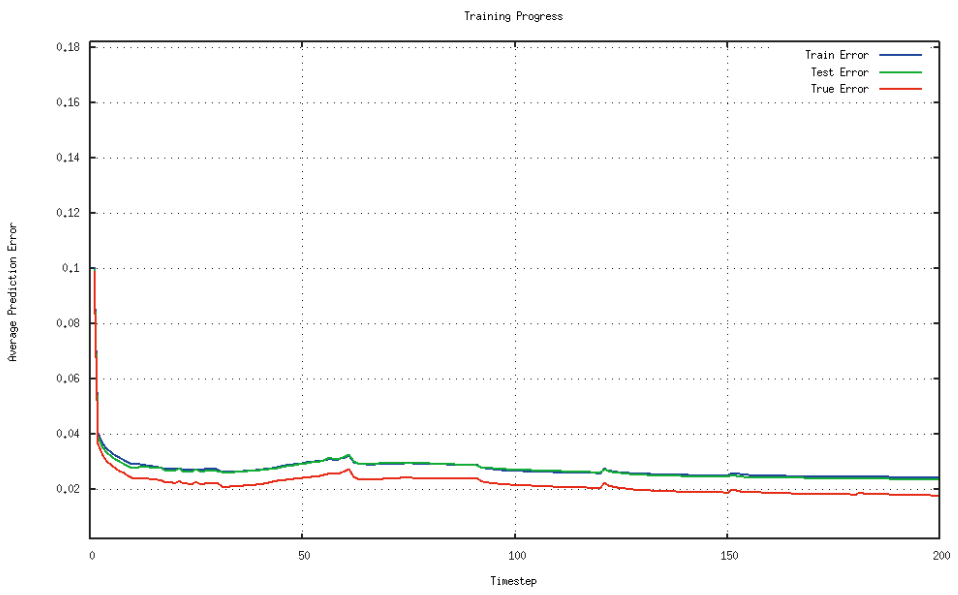

Figure 10 Average Error for Network Trained using Data with Noise Level 0.1

After 200 iterations, the average prediction deviation from the trend is about 0.02.

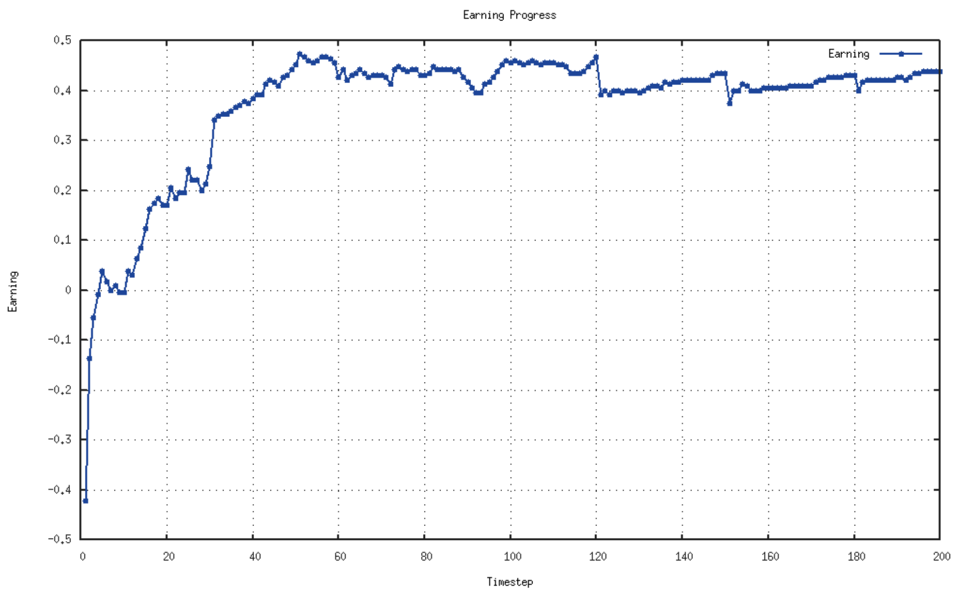

Figure 11 Earning for Network Trained using Data with Noise Level 0.1

The earning starts out low, then, stabilizes around 0.45, meaning the network can predict the growth sign about 72.5% of the time.

Now, we increase the noise level from 0.1 to 0.5 and repeat the training process:

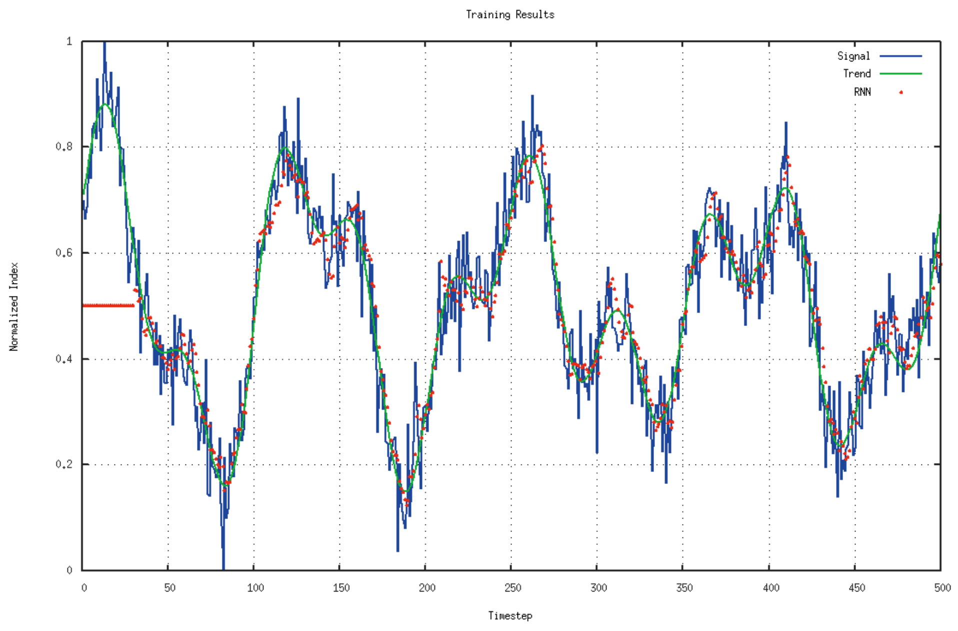

Figure 12 Results of Network Trained using Data with Noise Level 0.5

With more noise, the prediction deviation from the trend curve becomes larger. Nevertheless, the prediction points distribution still follows the general trend curve. The predicted value deviation from the trend curve has also increased as the following graph shows:

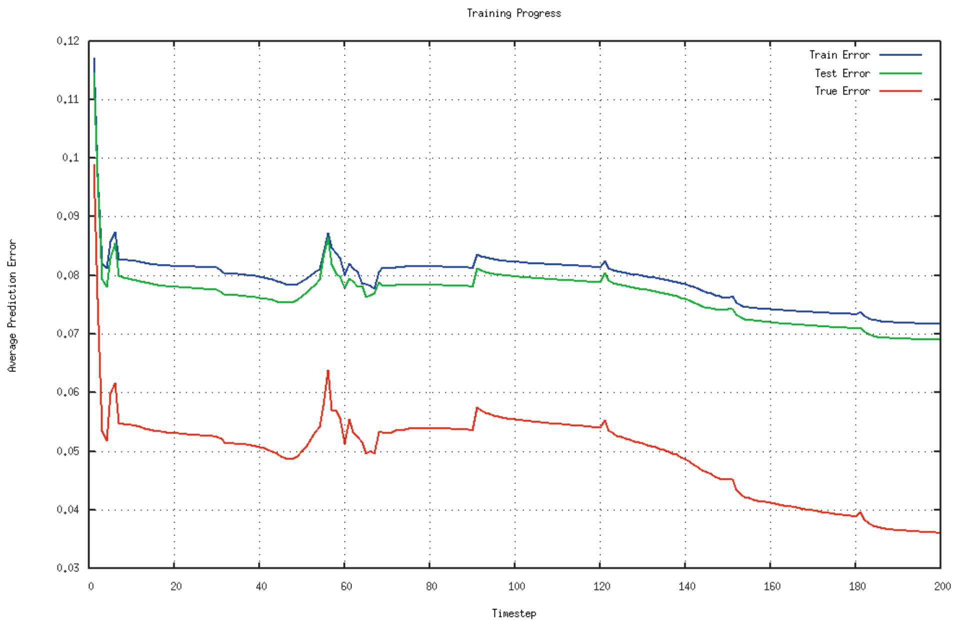

Figure 13 Average Error for Network Trained using Data with Noise Level 0.5

The following earning curve suggests that the network has difficulties learning the trend from the noisy dataset. This is understandable, because as the noise gets stronger, the random noise fluctuation can overwhelm the changes in the trend values, thus making the trend seems random.

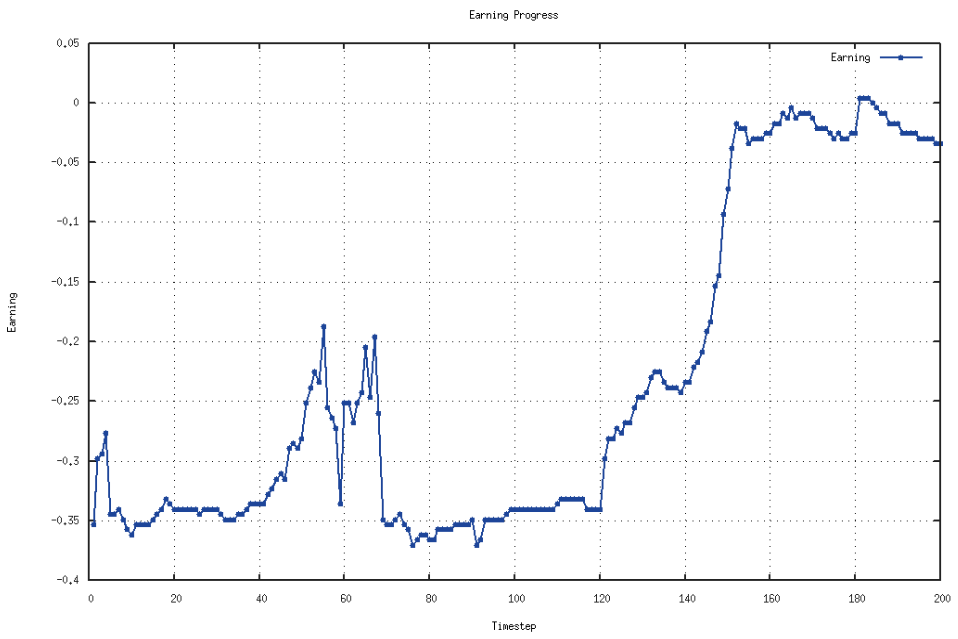

Figure 14 Earning for Network Trained using Data with Noise Level 0.5

So, in general, we can say that the noise in the dataset will make extracting pattern or trend more difficult, and the predictions we can draw from a noisy dataset is less accurate. This is important, because while there are many techniques to configure and optimize a recurrent neural network for certain task, the first step should always be to access how good the dataset is.

Memory Length

Previously, we have used memory length of 30 time steps for the benchmark design. The longer the memory period we allow the network to access in order to make a prediction, the more accurately it can predict the outcome, at least that is the intuition. So, now, we drop the allowed memory access length from 30 to only 10 and observe the effects.

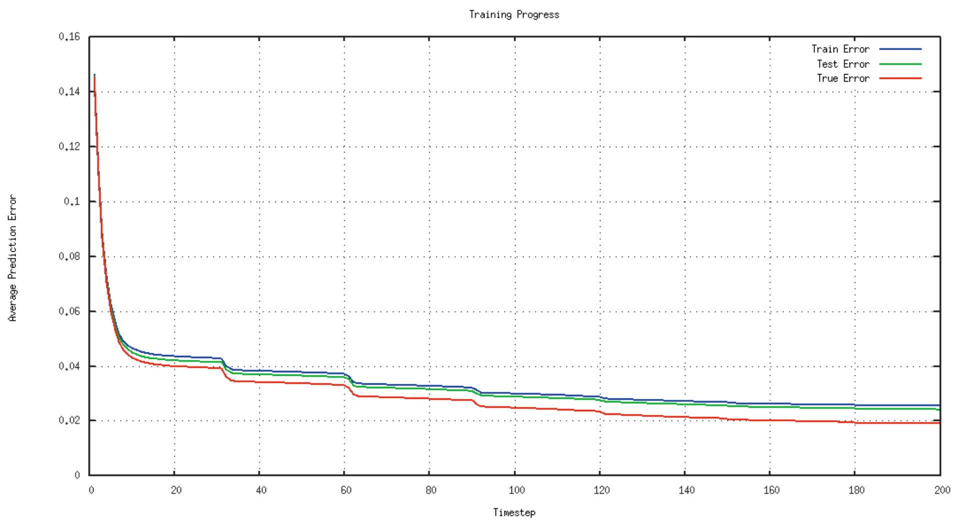

Figure 15 Average Error for Network Trained using Memory Length 10

The average error hovers at about the same value as the benchmark.

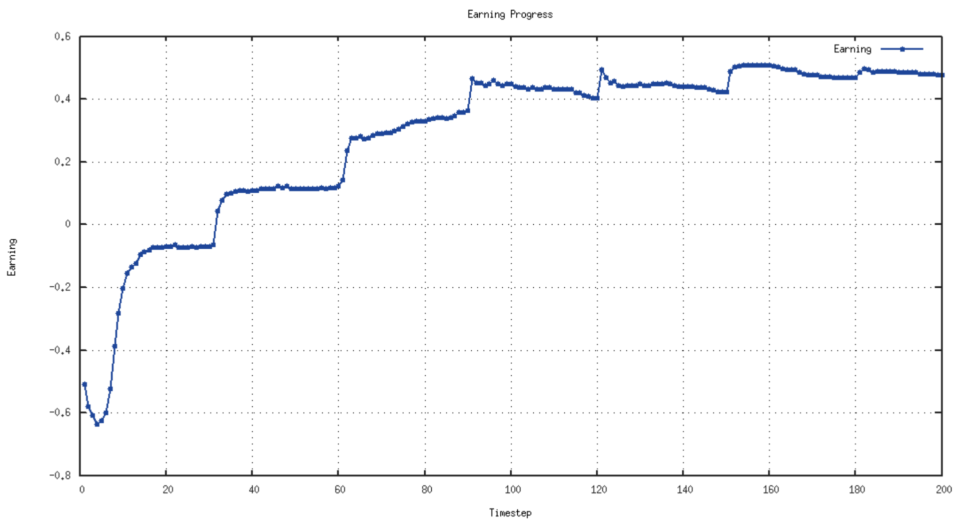

Figure 16 Earning for Network Trained using Memory Length 10

The earning percentage stops at about 0.5, which is slightly higher than the 0.45 in the benchmark. The result seems counter-intuitive at first. Because, we expect the network to make less accurate predictions with less historic data. One potential explanation is that while both memory length 10 and 30 suffices the job, the network training using length of 10 simply has more gradient updates, since the shorter series leads to 20 extra updates after sweeping through the entire series. So, the network is trained slightly. Another explanation is that even though increasing the memory length would increase the theoretical performance, it does not guarantee the optimization will end up in a minimum location superior than the network trained using less memory information. This is the same as a more complex convolutional neural network does not always work better than a simpler convolutional neural network because of potential overfitting and local minimum issue.

Data Size

Originally, we have used one complete period of dataset for the training. To compares the differences a larger dataset would be made, and following simulations were conducted using five complete periods:

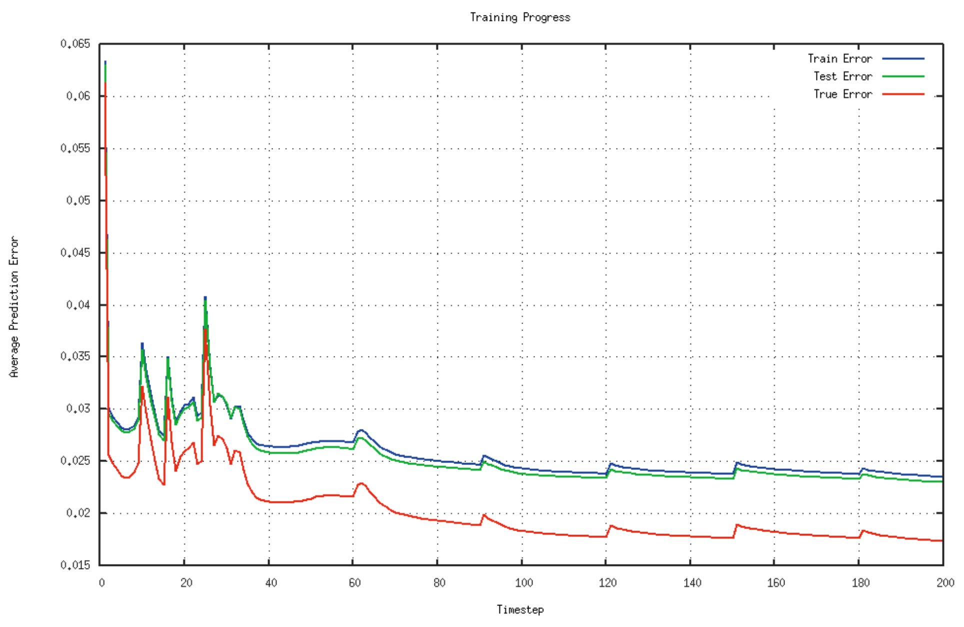

Figure 17 Average Error for Network Trained using Data with 5 Complete Period

From the error plot, we see a significant improvement over the benchmark errors.

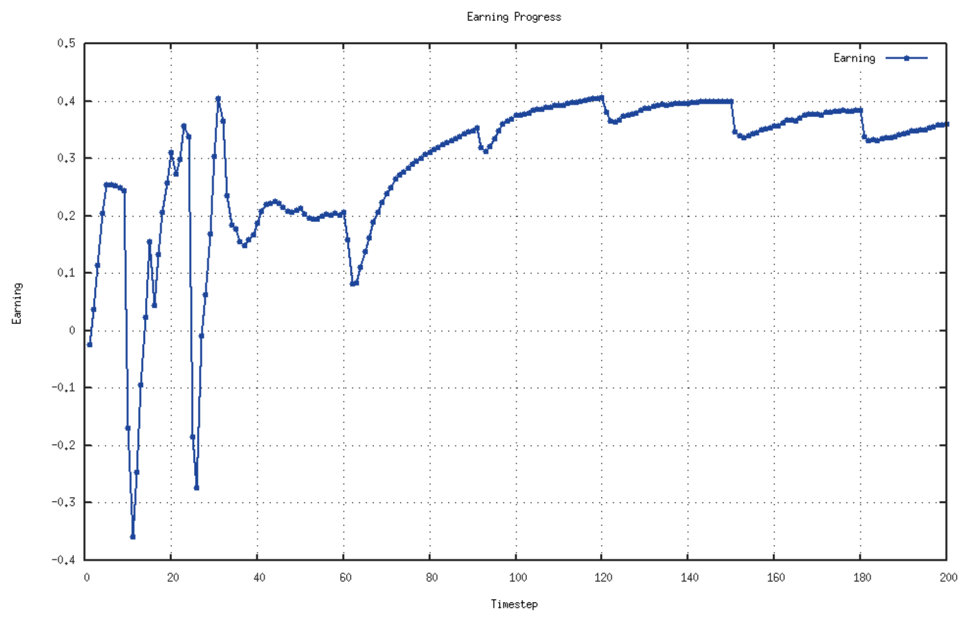

Figure 18 Earning for Network Trained using Data with 5 Complete Period

However, there is a slight decrease in the earning percentage. The explanation for the increase in the prediction accuracy and decrease in trend prediction is that while the longer data effectively reduces the noise in the data, thus lead to more accuracy, it is not guarantee to have an improvement over the earning, because the optimization is based on error measurement not the trend.

Complexity

The complexity refers to the number of the connections between recurrent layers in two adjacent time steps. The higher the number of connection there is, the more complex temporal pattern and transformation the network will be able to learn. For the original benchmark study, the recurrent layer output size is defined to be 10. In comparison, following results are from training with the recurrent layer output size of 3:

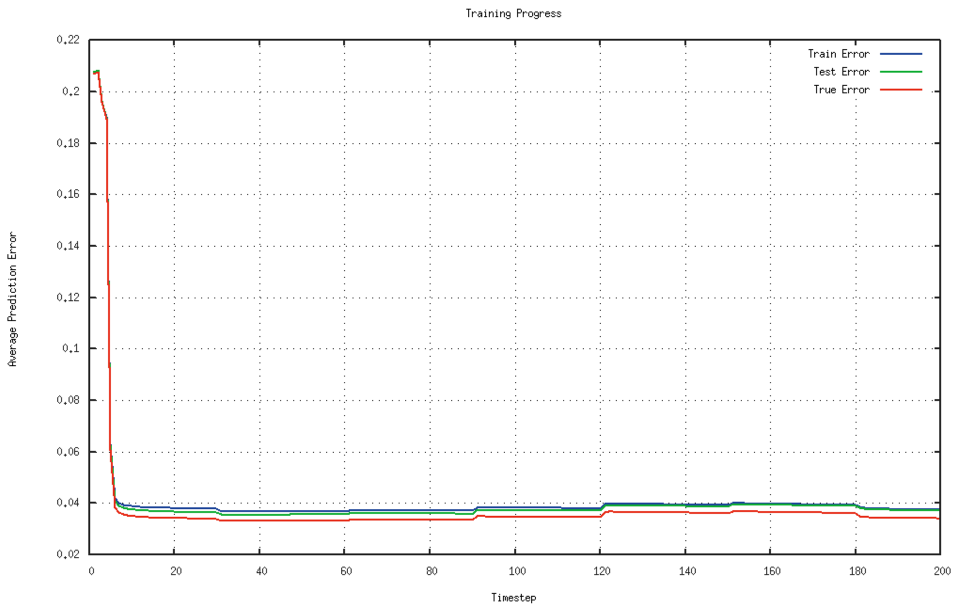

Figure 19 Average Error for Network Trained using Recurrent Layer Output of 3

The error has increase from the benchmark and it does not seem to have further room for improvement.

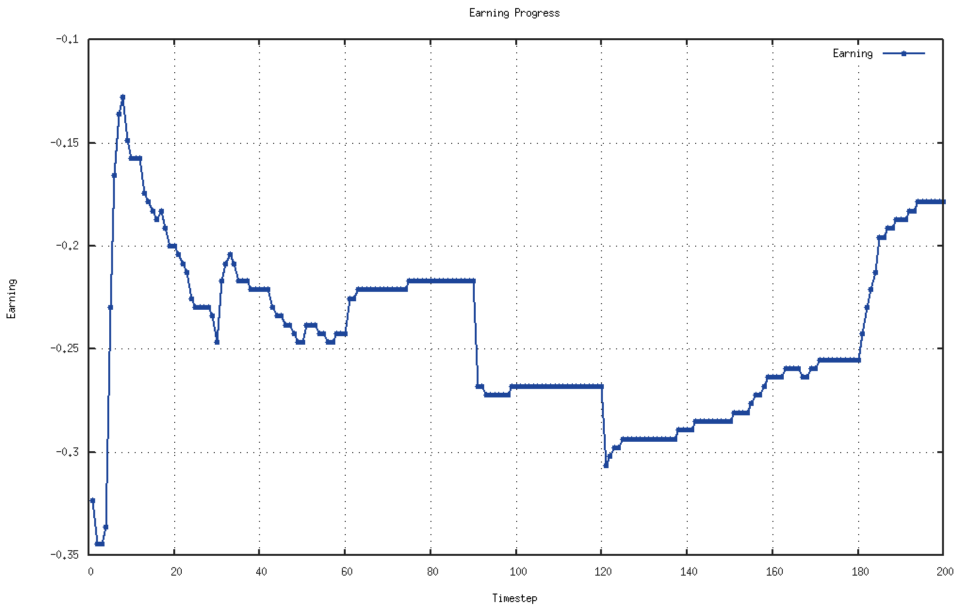

Figure 20 Earning for Network Trained using Recurrent Layer Output of 3

The ability to predict the trend is very weak. Even though the predicted points can fall in the vicinity of the index, the continuity and the trend is not very consistent. Therefore, we can see the capability has reduced significantly after reducing the between-time-step connection complexity.

Lead

Lead refers to the number of time steps ahead we can make a prediction. For the previous studies, we have only tried to predict one time step ahead, and we would like to know how the network performs when we train it to predict further ahead. Following results are from training to predict 5 time steps ahead:

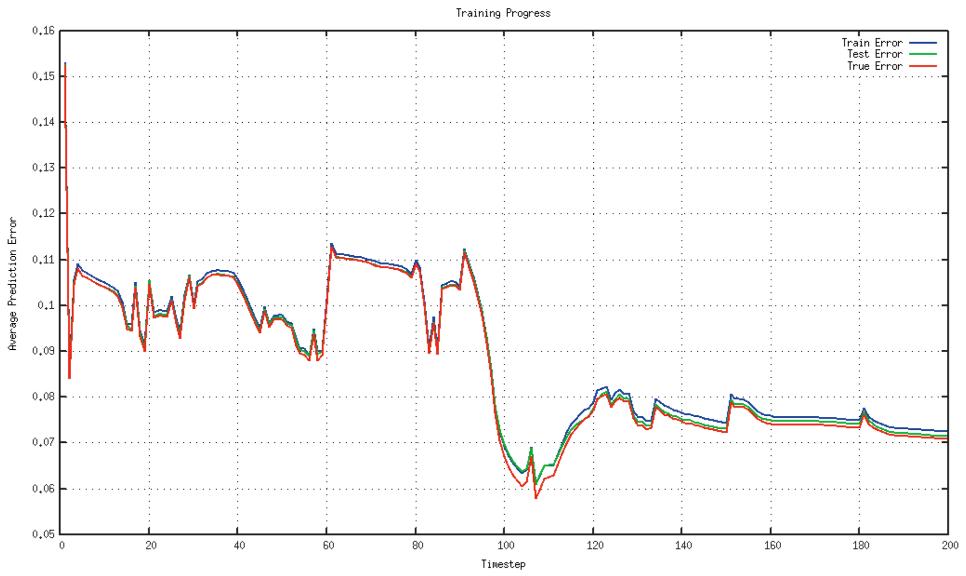

Figure 21 Average Error for Network Trained to Predict 5 Time Steps Ahead

For prediction made five time steps ahead, the accuracy is much lower compared to prediction made one time steps ahead.

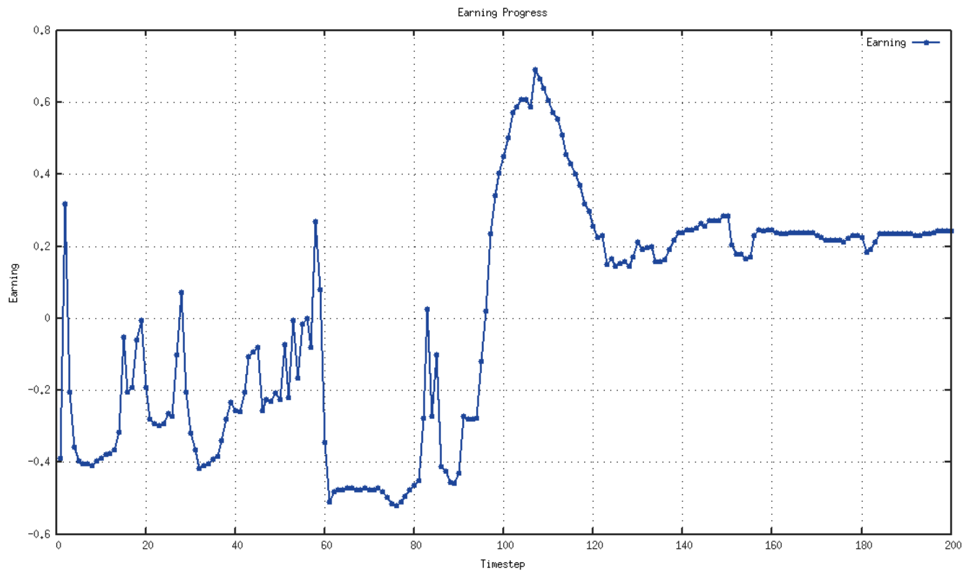

Figure 22 Earning for Network Trained to Predict 5 Time Steps Ahead

Even though the earning is less than that of the benchmark, it still stabilized on a positive value. The network can predict the trend correctly about 62.5% of the time.

Learning

This study has significantly improved my understanding of the recurrent network. The study explicitly showed the problem with a vanilla recurrent network for holding long-term memory and facilitated the deeper understanding of long-short term memory network. The study also practiced the implementation and debugging of a recurrent network using torch 7. The specific topics covered in the study are tabulated below:

Main Area	Key Concept
Regression	• Neural network design for regression task
Training	• Backpropagation in time • Memory Access Length • Reverse order gradient update
Vanilla Recurrent Network	• Recurrent connection • Vanishing gradient
LSTM	• Input, forget and output gates • Gate transfer function • Feedback Connection
Implementation	• Recurrent module (a generic module for recurrent connection) • LSTM module

One of the biggest challenge in dealing with recurrent neural network is debugging. Though, it can be said to all neural networks, it is particularly true for a recurrent neural network, where the output relies on both the input and the order. Nevertheless, the torch library has alleviated much of the difficulties in constructing a function recurrent neural network and it has been a great learning experience.

The coded for this project can be found in following link:

https://github.com/11dtech/gitTorch/tree/master/Recurrent

Reference

[1] Hochreiter, Sepp; Schmidhuber, Jürgen (1997). "Long Short-Term Memory". Neural Computation 9 (8): 1735–1780. doi:10.1162/neco.1997.9.8.1735. PMID 9377276.

[2] Klaus Greff; Rupesh Kumar Srivastava; Jan Koutník; Bas R. Steunebrink; Jürgen Schmidhuber (2015). "LSTM: A Search Space Odyssey". arXiv:1503.04069

[3] Paszke, Adam. "LSTM Implementation Explained." *LSTM Implementation Explained*. N.p., 30 Aug. 2015. Web. 04 Aug. 2016.

[4] "Element-Research/rnn." *GitHub*. Element Research, n.d. Web. 04 Aug. 2016.

[5] Graves, Alex, Abdel-rahman Mohamed, and Geoffrey Hinton. "Speech recognition with deep recurrent neural networks." *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013.