

# CS545 Assignment 8: Time Series Forecasting with Neural Network

He Yan  
yanhe@cs.colostate.edu

December 10, 2009

---

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Background</b>	<b>1</b>
<b>3</b>	<b>Data Preparation</b>	<b>2</b>
<b>4</b>	<b>Use Neural Network to Predict Future RTT</b>	<b>4</b>
<b>5</b>	<b>Experiment with Different Parameters</b>	<b>7</b>
5.1	The number of Inputs . . . . .	8
5.2	The number of Hidden Units . . . . .	9
5.3	Lambda . . . . .	9
<b>6</b>	<b>Discussion</b>	<b>11</b>
<b>7</b>	<b>Conclusion</b>	<b>13</b>

---

## 1 Introduction

In today's content delivery network (CDN), round trip time (RTT) between clients and servers is the most critical performance metric. Most of CDNs assign clients to servers based on RTT. In this assignment, we first employ neural network approach to forecast future RTT between clients and servers based on the past RTTs. Secondly, we experiment with different parameters such as the number of hidden units, the number of inputs and lambda on two different data sets in order to understand how they affect the forecast accuracy. Thirdly, we explain the possible reasons for the observations we get from the second part. At last, we discuss some findings/open issues and present the conclusion.

The remaining sections are organized as follows. Section 2 introduces some background of CDN and how we measure the RRT between servers and clients in a CDN. Section 3 focuses on the RTT data preparation that is a necessary step for RTT time series forecasting. In Section 4, we discuss how neural network is implemented to forecast RTT time series. Section 5 presents the possible explanations of the observations we get from the second part. In Section 6, we present open issues and conclusion.

## 2 Background

In this section, we briefly discuss some important background and how we measure RTTs between clients and CDN servers passively. CDN is a collection of servers containing copies of the same content, placed at various points in the Internet so as to minimize client perceived latency for access to the content from

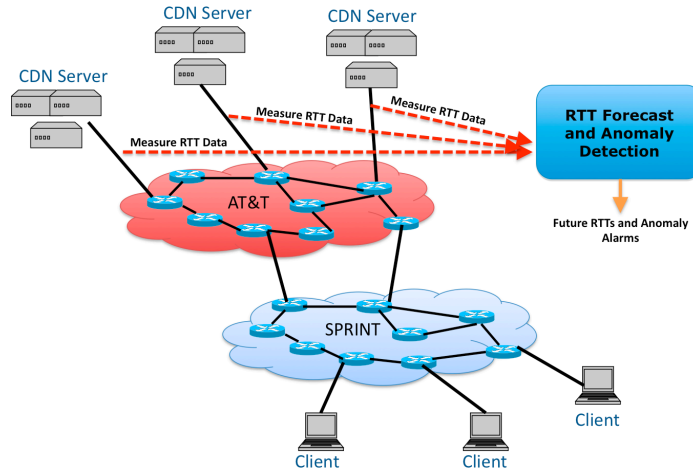


Figure 1: Passive Measurement of RTTs between Clients and Servers

clients. Each client typically is redirected to a server that has a smallest RTT with it, as opposed to all clients accessing the same central server. Predicting future RTT between servers and clients are critical for CDN operation in the following two aspects.

- Assigning servers to clients optimally needs accurate RTT forecast.
- Detecting anomalies in RTT between servers and clients needs accurate RTT forecast.

We measure the RTTs between clients and CDN servers by passively monitoring the hand-shaking packets during the TCP connection setup phase as shown in figure 1. For example, a SYN-ACK packet that CDN server A sends to a client C and the last ACK packet from client C to CDN node A are captured and the time difference of these two packets is used as the estimated RTT between client C and CDN node A. A RTT time series between a client and a CDN server consists of the passive measurements over time. We forecast future RTTs and detect RTT anomalies based on such a time series.

### 3 Data Preparation

In this section, we describe how we pre-process the raw RTT measurement data between each pair of server and client for more effective forecasting. We first try to forecasting RTTs for each pair of server and client. But it turns out to be not practical for two reasons.

- RTT measurements for one pair of server and client is typical too sparse to make any statistical significant forecast. For example, one pair of server and client only has less ten RTT measurements within 5 hours.
- As we have 10 servers and potential  $2^{32}$  IPv4 clients, we can NOT conduct time series forecasting for each pair of them simply due to scalability constrains.

In order to deal with the above issue, we decide to cluster clients that are “close” to each other into a group and conduct time series forecasting for these groups. For example, if there are  $N$  clients that are known to be close to each other, we can simple merge the RTT measurements from them with the same server into a time series based on the arrival time. Figure 2 shows an example how to group the RTT measurements (in ms) from two close clients into a time series. By doing this, we can get enough measurements for statistical significant forecasting in a short period and reduce the number of pairs of client and sever that we need to forecast.

Obviously, the next question we need to answer is how to define clients are close to each other. Geographical distance is natural definition but it doesn’t work well for Internet as two clients are in the same city may far away from each other in the Internet. Luckily, we can leverage on routing information that tells us

Group RTT measures from 2 Clients (against the same server) into a time series

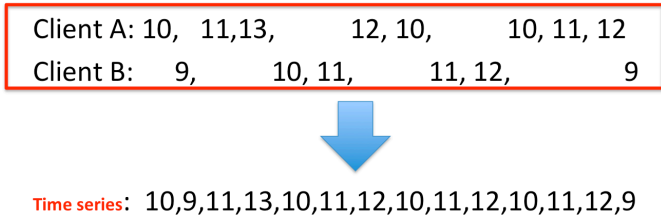


Figure 2: Group RTT Measurements from Clients “Close” to Each Other into a Time Series

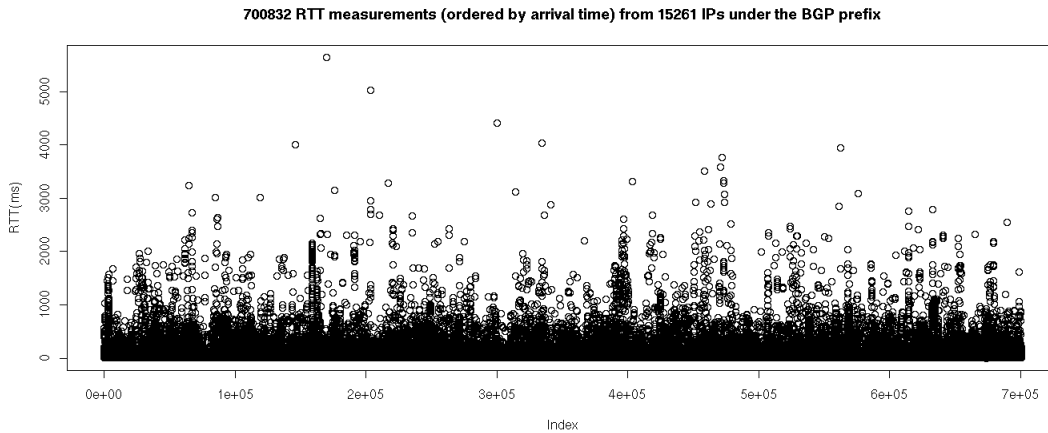


Figure 3: 700832 RTT measurements (ordered by arrival time) from 15261 IPs under the BGP prefix 98.166/16 during 15 days against the same server

what clients are close to each other on the Internet. Specifically, we consider that the IPs of clients that are announced under the same prefix in Border Gateway Protocol, the de facto inter-domain routing protocol of Internet, are close to each other. For example, all IPs of computers on CSU campus are under the same BGP prefix 129.82/16. Note there are some rare cases that the IPs under the same prefix are located at in different networks.

Each RTT actually can be further divided into propagation delay and queuing delay assuming the SYN-ACK and last ACK is too small to cause any transmission delay and processing delay is ignored. Propagation delay is mainly determined by routing and queuing delay is decided by the queuing behaviors on the routers along the path. The premise of grouping IPs under the same BGP prefix into a time series is that these IPs are mostly routed along the same path and share the same queuing behaviors on the routers along the path. As a result, most of IPs under the same BGP prefix should have the similar RTTs.

After grouping the RTT measurements from from all IPs under the same BGP prefix into a time series, how does the time series look like? Is it noisy with many outliers or relative smoothed? In order to conduct time series forecasting effectively, we need to form a relative smoothed time series. Figure 3 shows us a time series that is formed by 700832 RTT measurements from 15261 IPs under the BGP prefix 98.166/16 during 15 days against the same server. As you can see from this figure, there are many outliers and the formed time series are too noisy to do any statistical time series analysis. In order to quantify the outliers, we also plot the CDF of 700832 RTT measurements. As we can see from figure 4, the distribution is very concentrated on the small RTTs ( $\leq 50ms$ ). Specifically, 95% RTTs are small than 47ms.

The above observations suggests we needs a way to prune out the outlier before forecasting time series in order to improve the forecast accuracy. Specifically, we simple group RTT measurements from IPs during a fixed interval into a bin and take the median as the bin RTT. As shown in figure 5, with this simple approach

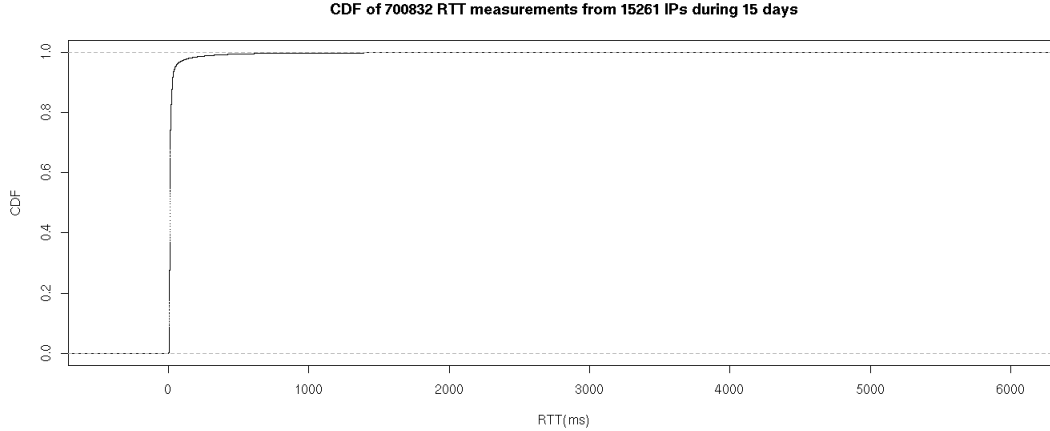


Figure 4: CDF of 700832 RTT measurements from 15261 IPs under the BGP prefix 98.166/16 during 15 days

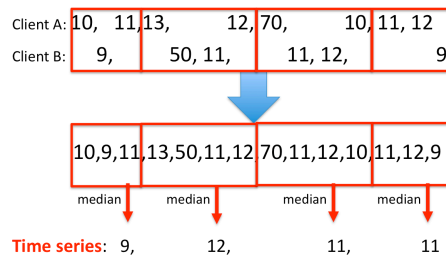


Figure 5: Use Time Bin to Remove Outliers

the outliers such as 70 ms from client A and 50 ms from client don't appear in the formed time series. Using this time bin approach, 700832 RTT measurements from 15261 IPs under the BGP prefix 98.166/16 forms a time series with 2527 data points (bins). Figure 6 shows that the time series is much more smoothed than the time series shown in figure 3. Note in figure 6 the range of y axis is from 0 to 25 while in figure 3 the range of y axis is from 0 to 5000. In the remaining of this report, the time series is formed using this time bin approach with bin size as 10 minutes.

## 4 Use Neural Network to Predict Future RTT

In this section, we discuss how to use neural network to predict future RTTs based on RTT time series. We define a RTT time series as a sequence of RTTs,  $RTT(t)$ ,  $t = 0, 1, \dots$ , where  $t$  represents the index of 10 minutes time bins. Neural networks is used to forecast the future of the time series from values of past RTTs up to the current time. Figure 7 shows an example that 3 past RTT values ( $RTT(t)$ ,  $RTT(t-1)$  and  $RTT(t-2)$ ) are used as the input to neural network to forecast 2 future RTT values ( $RTT(t+1)$  and  $RTT(t+2)$ ) that are the outputs of neural network. Note in this report, we focus on the case of forecasting one future RTT. Our study can be easily extended to the case of forecasting multiple future RTT values.

In our implementation, we first read a RTT time series from a data file and partition it into train set (80%) and test set (20%). The code of doing this is as follows.

```
rft_data=scan("rtts.data")
train = rft_data[1:(length(rft_data)*0.8)]
test = rft_data[-(1:(length(rft_data)*0.8))]
```

After the first step, train set and test set are actually 2 time series (vectors).

Secondly, we need to convert the train set and test set into 4 matrixes, namely Xtrain, Ttrain, Xtest and Ttest. Xtrain and Ttrain are passed to neural network training "makeNN" and Xtest is passed to

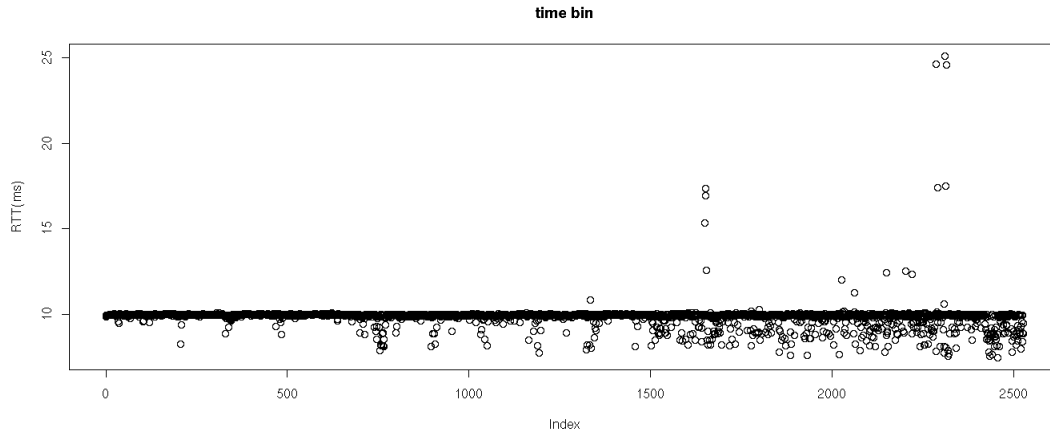


Figure 6: a time series of 2527 data points using time bin approach based on 700832 RTT measurements from 15261 IPs under the BGP prefix 98.166/16

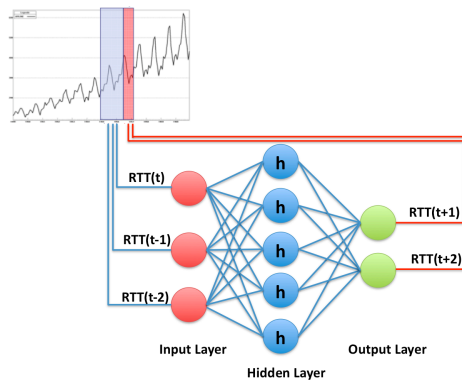


Figure 7: Neural Network Architectures for Time Series Forecasting

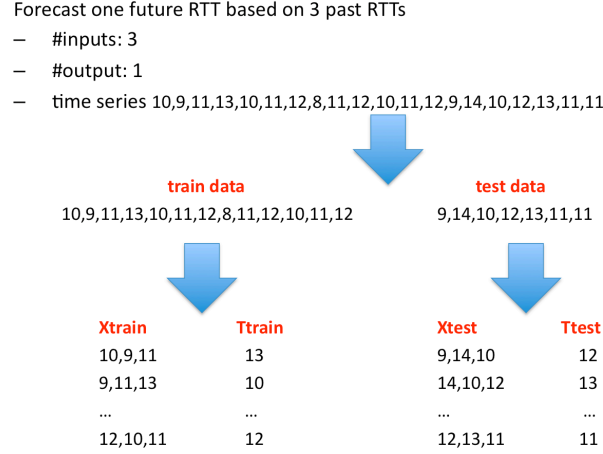


Figure 8: How to Construct Xtrain, Ttrain, Xtest and Ttest

neural network predict function “useNN”. The output of predict function “useNN” will be the forecasted future RTTs, which will be compared with Ttest to verify the forecast accuracy. The shapes of these matrix depends on the number of inputs and outputs in neural networks. In other word, it depends on how many future RTTs (outputs) one wants to forecast based on how many past RTTs (inputs). Suppose a time series has a length  $N$ , the number of inputs is  $I$  and the number of outputs is  $O$ , the shape of Xtrain and Xtest is  $(N - I) \times I$ . The shape of Ttrain and Test is  $(N - I) \times O$ . Figure 8 shows that how to split a time series into train/test sets and how to construct the four matrixes Xtrain, Ttrain, Xtest and Ttest.

Code for this part is listed as follows. Here the number of outputs is always 1 while the number of inputs is specified b the variable “input”.

```

Xtrain = NULL
Ttrain = NULL
for(i in 1:(length(train)-input)){
  for(j in 0: input){
    if(j== input){
      Ttrain = c(Ttrain , train [i+j])
    }
    else{
      Xtrain = c(Xtrain , train [i+j])
    }
  }
}
Ttrain = as.matrix(Ttrain)
Xtrain = matrix(Xtrain , length(Xtrain)/input , input)

Xtest = NULL
Ttest = NULL
for(i in 1:(length(test)-input)){
  for(j in 0: input){
    if(j==lag){
      Ttest = c(Ttest , test [i+j])
    }
    else{
      Xtest = c(Xtest , test [i+j])
    }
  }
}
Ttest = as.matrix(Ttest)
Xtest = matrix(Xtest , length(Xtest)/input , input)

```

Thirdly, function “makeNN” is called to train a two-layer neural network with a specified number of hidden units and lambda based on training data (Xtrain and Train). Lambda controls the penalty on all hidden layer weights except the constant 1 input weights. The hidden layer uses tanh as the nonlinear function and the output layer is linear. Function “makeNN” has seven arguments:

- Xtrain: past RTTs
- Ttrain: target variables of training data
- nhidden: number of hidden units
- lambda: the penalty on all hidden layer weights except the constant 1 input weights
- nIterations, xPrecision, fPrecision: the stop criteria of scaled-conjugate gradient algorithm

The last four arguments are optional and default values of them are 0, 100000, 0.00000001 and 0.00000001 respectively. The code is listed as follows.

```
nn = makeNN(Xtrain , Ttrain , nh , lambda)
```

Fourthly, function “useNN” is called to use the neural network trained by function “makeNN” to predict future RRTs based on the past RTTs based on test data (Xtest). Function “useNN” is quite straightforward, which has two arguments.

- model: the trained neural network returned by function “makeNN”
- Xtest: input variables of test data

The code is listed as follows.

```
predict = useNN( nn , Xtest )
```

We won’t describe the details of the two functions “makeNN” and “useNN” as they were discussed in detail in assignment 5’s report.

At last in order to quantify the accuracy of forecasting we calculate the RMS errors between the forecasted RTTs and the actual RTTs in Ttest. The code is as follows.

```
testerror <- predict - Ttest  
RMSE = sqrt(mean(testerror^2))
```

## 5 Experiment with Different Parameters

In this section, we experiment with several key parameters in order to understand how they affect the accuracy of neural network time series forecasting. Specifically, we focus on the following 3 parameters.

- the number of inputs of neural network
- the number of hidden units in neural network
- lambda: the penalty on all hidden layer weights

The accuracy is measured in RMS error or RMSE between the forecasted RTTs and actual RTTs as mentioned above. Note here we focus on the accuracy of forecast test data. We use two data sets for the experiments in this section.

- In the first data set, we have 700832 RTT measurements from 15261 IPs under the BGP prefix 98.166/16 and construct a time series of 2527 data points using time bin approach.
- In the second data set, we have 304822 RTT measurements from 8307 IPs under the BGP prefix 24.2.128/17 and construct a time series of 2517 data points using time bin approach.

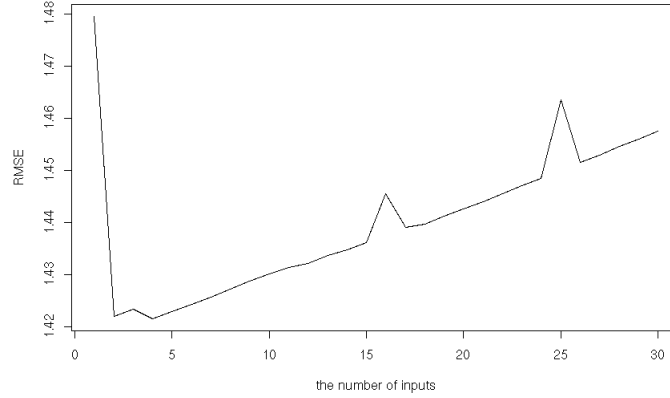


Figure 9: The Number of Inputs VS. RMSE

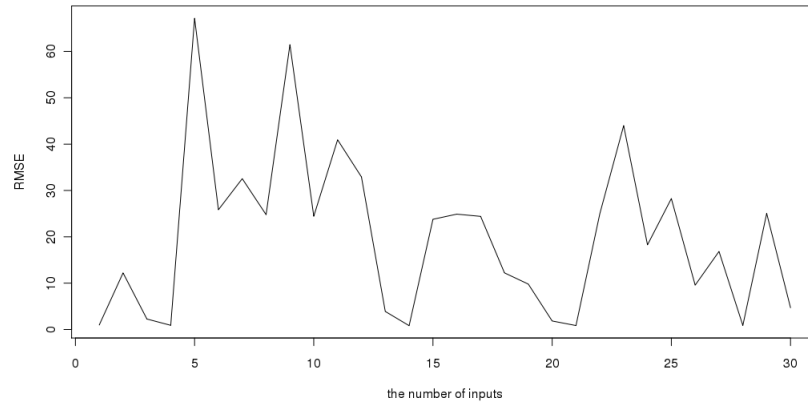


Figure 10: The Number of Inputs VS. RMSE

## 5.1 The number of Inputs

In this subsection, we vary the number of inputs from 1 to 30 while we fix the number of hidden units as 5 and lambda as 0. First we experiment with the first data set and collect the RMSEs for different numbers of inputs. Figure 9 shows the relation between the number of inputs and RMSE. We can clearly see that for the first data set 4 inputs will generate the smallest RMSE and overall more inputs give bigger RMSE. In the context of time series forecasting, forecasting the future RTT based on 4 past RTTs gives the best results. We also notice for the first data set that the impact on RMSE by the number of inputs is not significant as the range of y axis of figure 9 is only between 1.42 and 1.48.

In order to verify if the above observations still hold for other time series, we conduct the same experiment on the second data set. Figure 10 shows the relation between the number of inputs and RMSE for the second data set. We can see that the pattern in figure 10 is very different from figure 9. Specifically, the RMSEs for the number of inputs varying from 1 to 30 have a large variance from 0.8233493 to 67.1580812. When the number of inputs is 14, the min RMSE 0.8233493 is achieved. When the number of inputs is 5 the max RMSE 67.1580812 is achieved. Although overall trend is not clear for the second data set, we can confirm that for different data sets we need different parameters.

From the above experiments, we know 4 inputs give best forecasting accuracy for the first data set and 14 inputs give best forecasting accuracy for the second data set. But why 4 and 14 are the magic numbers for the first data set and second data set? We try to answer this question by drawing autocorrelation plots that is a commonly-used tool for checking patterns in a data set. Patterns are ascertained by computing autocorrelations for data values at varying lags. Figure 11 shows the autocorrelation plot for the first data set. It clearly shows that the autocorrelation is significant for lags smaller than 5. This may explain why we

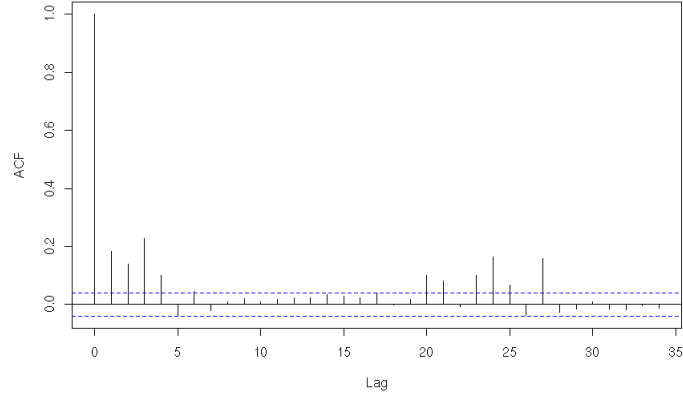


Figure 11: Autocorrelation plot for the first data set

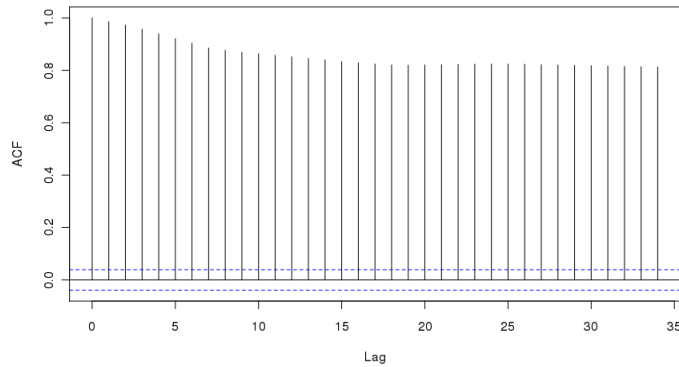


Figure 12: Autocorrelation plot for the second data set

get the best result with the number of input as 4. Figure 12 shows the autocorrelation plot for the second data set. Unlike the first data set, we can hardly see anything significant around lag 14 there.

## 5.2 The number of Hidden Units

In this subsection, we vary the number of hidden units from 1 to 10 while we fix the number of inputs as 4 for the first data set and 14 for the second data set. Lambda is fixed as 0 for both data sets.

First we experiment with the first data set and collect the RMSEs for different numbers of hidden units. Figure 13 shows the relation between the number of hidden units and RMSE. We can see that for the first data set the number of hidden units doesn't impact the RMSE much as the range of y axis is only from 1.42154 to 1.42160. That suggests adding more hidden units doesn't help much.

Similar to what we did for the number of inputs, here in order to verify if the above observations still hold for other time series, we conduct the same experiment on the second data set. Figure 14 shows the relation between the number of inputs and RMSE for the second data set. We can see that the curve in figure 14 is kind of different from the one in figure 13. Specifically, the RMSEs for the number of hidden units varying from 1 to 10 have a much larger variance from 0.8233493 to 10.9859927. When the number of hidden units is 5, the min RMSE 0.8233493 is achieved. When the number of inputs is 9 the max RMSE 10.9859927 is achieved. It suggests that for the second data set using the number of hidden units between 3 and 6 will give better results.

## 5.3 Lambda

In this subsection, we vary lambda from 0 to 2 while we fix the number of inputs as 4 for the first data set and 14 for the second data set. And the number of hidden units is fixed as 7 for the first data set and 5 for

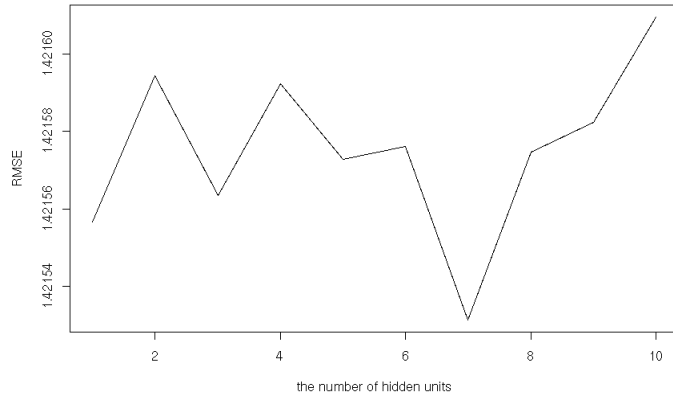


Figure 13: The Number of Hidden Units VS. RMSE for The First Data Set

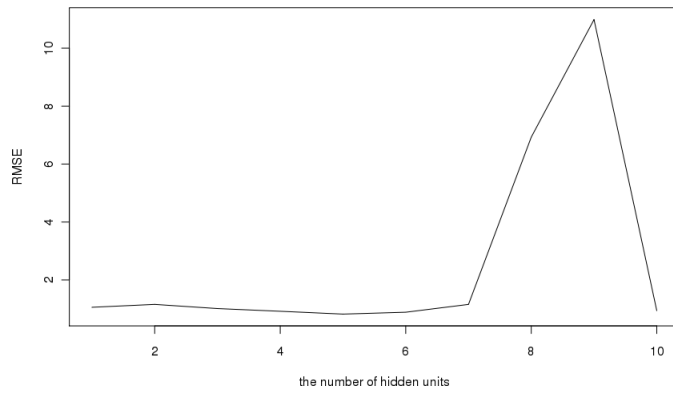


Figure 14: The Number of Hidden Units VS. RMSE for The Second Data Set

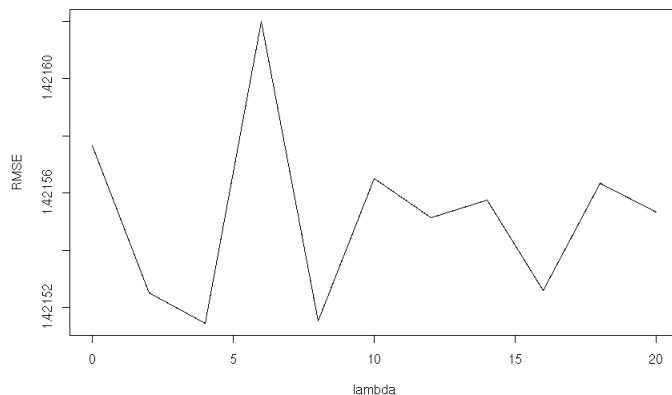


Figure 15: Lambda VS. RMSE for The First Data Set

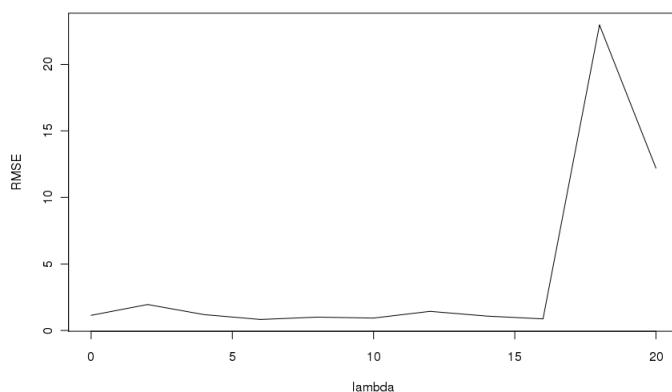


Figure 16: Lambda VS. RMSE for The Second Data Set

the second data set.

First we experiment with the first data set and collect the RMSEs for different lambdas. Figure 15 shows the relation between lambda and RMSE for the first data set. Similar to the number of inputs and hidden units, we can see that for the first data set lambda doesn't impact the RMSE much as RMSE is only ranging from 1.421515 to 1.421620.

Similar to what we have done for the number of inputs and hidden units, here in order to verify if the above observations are similar for other time series, we conduct the same experiment on the second data set. Figure 16 shows the relation between the number of inputs and RMSE for the second data set. Again we can see that the curve in figure 14 is different from the one in figure 13. Specifically, the RMSEs for lambda varying from 0 to 0.2 have a much larger variance from 0.8269092 to 22.9595754. When lambda is 0.6, the min RMSE 0.8269092 is achieved. When lambda is 1.8 the max RMSE 22.9595754 is achieved. It suggests that for the second data set using small lambdas between 0.4 and 1 will give better results.

## 6 Discussion

From the previous section, we have seen that different parameter have very different impacts on different data sets. Specifically, the first data set is not sensitive to the parameters we studied such as the number of inputs, the number of hidden units and lambda. As shown in figure 17, the RMSE doesn't change much regardless the change of the number of inputs and hidden units. Only a few spikes of RMSE appear when the number of inputs and the number of hidden units are relative large. Note here the range of RMSE is only from 1.4 to 2.

In contrast to the first data set, the second data set is sensitive to different parameters. As shown in

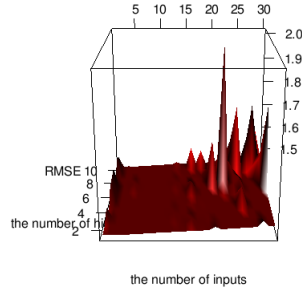


Figure 17: The Number of Inputs and Hidden Units VS. RMSE for The First Data Set

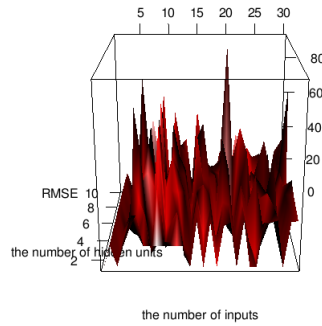


Figure 18: The Number of Inputs and Hidden Units VS. RMSE for The Second Data Set

figure 18, the RMSE varies greatly when the number of inputs or the number of hidden units changes. In addition, the range of RMSE (from 0.8190957 to 91.29912) for the second data set is much larger than the first data set.

What is the difference between these 2 data sets? We try to understand the difference by simple drawing the time series of these 2 data sets. Figure 19 and 20 show the time series plot for both data sets respectively. We can see clear that there is no level shift in the first data set while there is big level shift in the second data set. This may indicate that the neural network forecasting will be more sensitive to the parameters when the time series has larger variance.

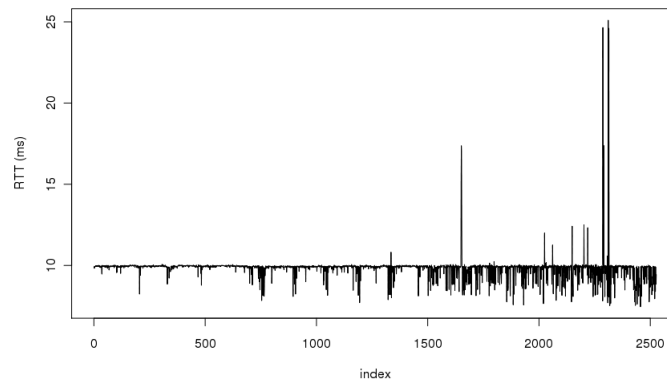


Figure 19: Time Series of The First Data Set

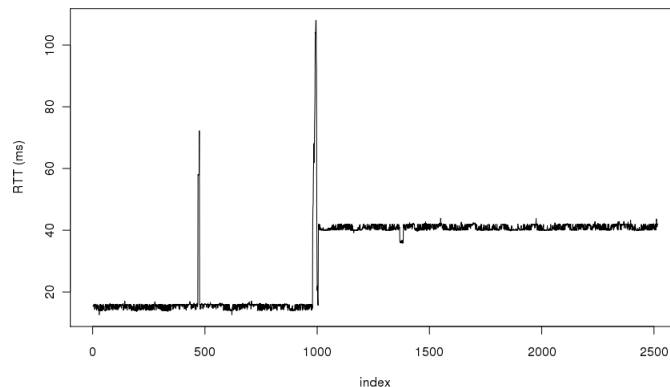


Figure 20: Time Series of The Second Data Set

## 7 Conclusion

In this assignment, we first implemented a neural network for RTT time series forecasting in CDN. Secondly we experimented with different parameters on two different data sets in order to understand how the parameters impact the forecasting accuracy. The parameters that we studied are the number of inputs, the number of hidden units and lambda. Our observations are that for the first data set these parameters have no significant impact on the forecasting accuracy while for the second data set, the impacts of these parameters are quite significant. Thirdly, we found the data set (the RTT time series) with larger variance is more likely to be affected by different parameters in terms of forecasting accuracy.

One open issue we noticed is that we only tried 2 data sets due to time constraints. The 2 data sets are constructed from all IPs under 2 BGP prefix. As there are 30,000 BGP prefixes in the Internet and each of them can construct a data set, we should try more data sets in order to get a deeper understanding on how this neural network forecasting approach works in general. For the future work, we are interested to extend our experiment to more data sets.

## References

- [1] Anderson, C., *Teach Yourself R*, [http://www.cs.colostate.edu/~anderson/cs545/class\\_material/TeachYourselfR.pdf](http://www.cs.colostate.edu/~anderson/cs545/class_material/TeachYourselfR.pdf), 2004.
- [2] *UCI Machine Learning Repository*, <http://archive.ics.uci.edu/ml/datasets/Abalone>, 2008.
- [3] *Assignment 8 Website*, <http://www.cs.colostate.edu/~anderson/cs545/assignments/assignment8.html>, 2009.
- [4] Venables, W.N. and Ripley, B.D., *An Introduction to R—Notes on R: A Programming Environment for Data Analysis and Graphics*, <http://cran.r-project.org/doc/manuals/R-intro.pdf>, 2005.
- [5] *R Online Reference*, <http://sekhon.berkeley.edu/stats/html/optim.html>, 2008.