Time-Series Forecasting with Neural Networks

Amin Bavand
bavand@ualberta.ca

1 Introduction

Time series are sets of observation that are recorded through time. A time series follows a predefined model and the change in time series through time follows some patterns. It is possible to extract the behaviour of a time series based on the values of the time series that are changing through time. If we would be able to design a model that can predict the time series future samples based on the previous samples, we can say that we have succeeded in finding the behaviour of that time series. There are so many phenomena that can be modeled with time series. If we are able to do so, then we actually achieved prediction of natural phenomena.

So, we have to do the prediction of future samples of time series just by having the previous samples of that time series. We will design two methods and compare the results of these methods with the existing method on time series prediction. So our problem would be finding the best network that can perform the prediction of time series.

In this project, we use neural networks to do the time series prediction job for us. We use keras package in python to design a feedforward and a recurrent neural network. These neural network have to be trained with the samples of time series. A number of samples should be used as the input of our networks and one other sample that happen after those samples should be used as the target (output) of the network for those inputs. In the training process we find the proper weights for our networks and also the number of layers and neurons that give us the best prediction results. The evaluation of the predictions is based on three measures namely RMSE, NMSE, and NDEI. These measures tell us how close we are to the ideal prediction of time series.

In section 2, we will have a review of the network architectures that we use. In section 3, we comprehensively explain about our methods, including details of the datasets that we use and preprocessing of dataset, design procedures for the neural networks and parameter exploration procedures and the performance measure RMSE, NMSE, and NDEI. In section 4, the results of exploring the parameters are brought. Section 5 includes the final result of our method performing on test data. Finally, section 6 is conclusion.

2 Review

In this project, we perform two neural network architectures on time series: feedforward and recurrent. In sections 2.1 and 2.2, we bring a brief explanation of architectures of feedforward and recurrent neural network.

2.1 Feedforward architecture

For feedforward architecture, we used multilayer perceptron (MLP). A multilayer perceptron consists of an input layer (in which the input is applied to the network), one or several hidden layers, and an output layer (which gives the result of the network on the applied input). The structure of a multilayer perceptron with two hidden layers is shown in *Figure 1*:

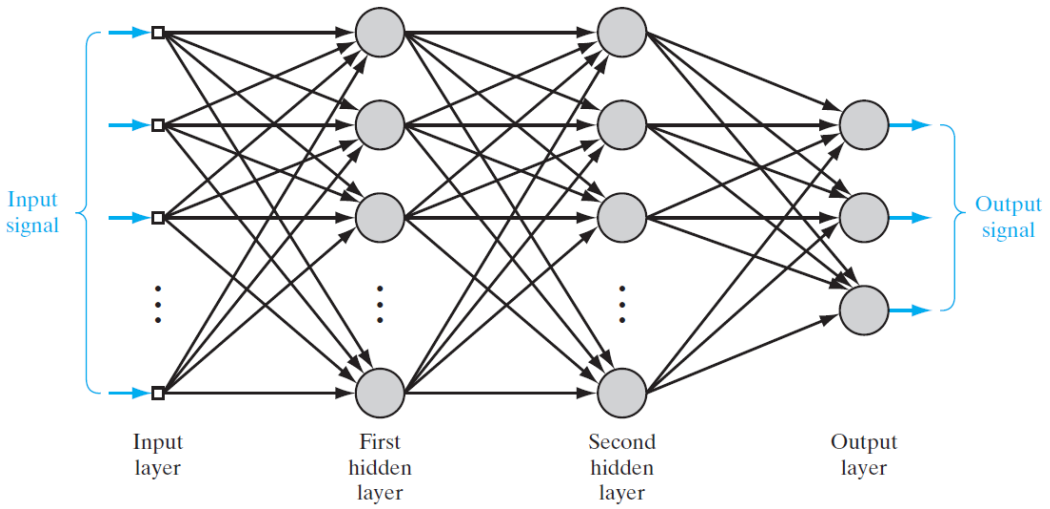

Figure 1: Architecture of a multilayer perceptron with two hidden layers *[1]*

As it can be seen in *Figure 1*, the MLP structure is fully connected (there is a connection between all neurons of a layer to the next layer).
The output, input and weight vector are represented with Y, X, and W. In forward pass we have:

$$v_j(n) = \sum_{i=0}^{m} w_{ij}(n) y_i(n)$$
$$y_j(n) = \varphi\left(v_j(n)\right)$$

For updating weights, if the total error is E, in backward pass we would have:

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

After writing down the equations, finally we have:

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi_j'\left(v_j(n)\right) y_i(n)$$

So, the update term would be:

$$\Delta w_{ji}(n) = \eta e_j(n)\varphi_j'\left(v_j(n)\right) y_i(n)$$

Which η is the learning rate parameter.

2.2 Recurrent architecture

"Recurrent neural networks (RNNs) are a connectionist model containing a self-connected hidden layer. One benefit of the recurrent connection is that a 'memory' of previous inputs remains in the network's internal state, allowing it to make use of past context" [2]. There are many kinds of recurrent architectures that can be used. We tested some of them on our dataset and observed that long short-term memory (LSTM) had better results than the others, so we selected this architecture as the one we use in this project.
There is a flaw in the standard RNN named *vanishing gradient problem*. LSTM was designed to overcome this problem. "An LSTM hidden layer consists of recurrently connected subnets, called memory blocks. Each block contains a set of internal units, or cells, whose activation is controlled

by three multiplicative gates: the input gate, forget gate and output gate" [2]. Figure 2 shows the internal structure of an LSTM:

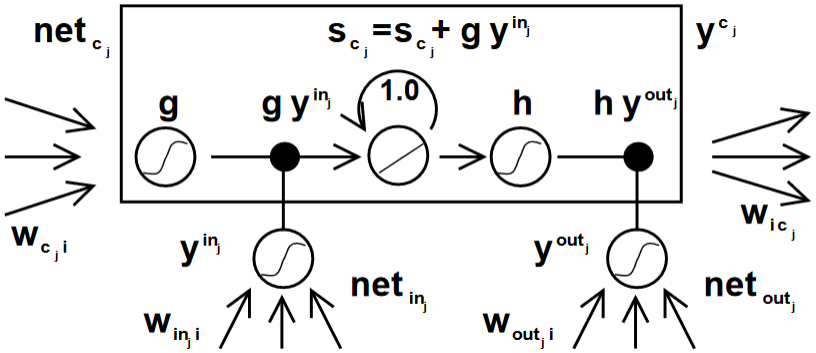

Figure 2: Architecture of a memory cell for LSTM *[3]*

In general if we have *j* memory cells, each denoted by c_j , for forward pass, we have [3]:

$$y^{out_j}(t) = f_{out_j}\left(net_{out_j}(t)\right) \quad \& \quad y^{in_j}(t) = f_{in_j}\left(net_{in_j}(t)\right).$$

Where:

$$net_{out_j}(t) = \sum_u w_{out_j u} y^u(t-1)$$

$$net_{in_j}(t) = \sum_u w_{in_j u} y^u(t-1)$$

$$net_{c_j}(t) = \sum_u w_{c_j u} y^u(t-1)$$

The output of the cell at time *t* would be:

$$y^{c_j}(t) = y^{out_j}(t) h\left(s_{c_j}(t)\right)$$

And internal state is defined as follows:

$$s_{c_k}(0) = 0, s_{c_j}(t) = s_{c_j}(t-1) + y^{in_j}(t) g(net_{c_j}(t))$$

The update for the weight w_{lm} at time *t* are as follows (backward pass) [4]:

$$E(t) = \frac{1}{2}\sum_k e_k(t)^2$$

$$\Delta w_{lm}(t) = -\alpha \frac{\partial E(t)}{\partial w_{lm}} = -\alpha \frac{\partial E(t)}{\partial y^k(t)} \frac{\partial y^k(t)}{\partial w_{lm}} = \alpha \sum_k e_k(t) \frac{\partial y^k(t)}{\partial w_{lm}} =$$

$$\alpha \sum_k e_k(t) \frac{\partial y^k(t)}{\partial y^l(t)} \frac{\partial y^l(t)}{\partial net_l(t)} \frac{\partial net_l(t)}{\partial w_{lm}}$$

$$\Delta w_{lm}(t) = \alpha \frac{\partial y^l(t)}{\partial net_l(t)} \left(\sum_k \frac{\partial y^k(t)}{\partial y^l(t)} e_k(t) \right) y^m(t-1)$$

3 Methodology

As explained in section 1, our goal in this project is time series prediction by using neural networks. We performed feed-forward and recurrent neural networks on Mackey-Glass [5] and Santa Fe Laser [6] time series datasets. In the following section, we will explain about details of these datasets, preprocessing, design procedures for the neural networks, parameter exploration procedures, and the performance measures to be used.

3.1 Dataset

Before specifically talking about the dataset, we first explain what time series are and what the meaning of time series prediction is.

"A time series $\{Y_t\}$ or $\{y_1, y_2, ..., y_T\}$ is a discrete time, continuous state process where time $t = 1, 2, ..., T$ are certain discrete time points spaced at uniform time intervals" [7]. Time series prediction (or forecasting) means finding a model that could predict the future values of the process based on values that observed so far.

Now that we know preliminary information about time series forecasting problem, we can jump to the details of our two datasets.

3.1.1 Mackey-Glass dataset

Mackey-Glass dataset comes from Mackey-Glass equation:

$$\text{mackeyglass\_eq:} \quad \frac{dx(t)}{dt} = \frac{ax(t-\tau)}{1+x(t-\tau)^{10}} - bx(t)$$

It can be numerically solved using the 4th order Runge-Kutta method [8]:

$$k_1 = \Delta t.mackeyglass\_eq\left(x(t), x(t-\tau), a, b\right)$$

$$k_2 = \Delta t.mackeyglass\_eq\left(x(t+\frac{1}{2}k_1), x(t-\tau), a, b\right)$$

$$k_3 = \Delta t.mackeyglass\_eq\left(x(t+\frac{1}{2}k_2), x(t-\tau), a, b\right)$$

$$k_4 = \Delta t.mackeyglass\_eq\left(x(t+k_3), x(t-\tau), a, b\right)$$

$$x(t+\Delta t) = x(t) + \frac{k_1}{6} + \frac{k_2}{6} + \frac{k_3}{6} + \frac{k_4}{6}$$

In the dataset that we used for this project, there are 1201 generated samples with Mackey-Glass dataset [5]. The time series is illustrated in Figure 3:

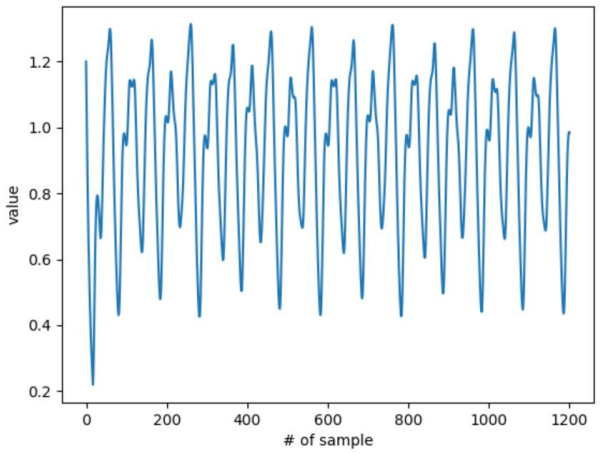

Figure 3: Mackey-Glass time series representation

3.1.2 Santa Fe Laser dataset

"The Santa Fe laser data is a standard benchmark data set in system identification. It serves as good starting point to start exploring time series models. It records only one observable (laser intensity), has little noise" [6] and is generated from Lorenz sets of equations [9]:

$$\frac{dx}{xt} = \sigma(y - x)$$

$$\frac{dy}{dt} = rx - y - xz$$

$$\frac{dz}{dt} = xy - bz$$

The exact explanation of how this dataset is generated is given in source [10]:

"The measurements were made on an 81.5-micron 14NH3 cw (FIR) laser, pumped optically by the P(13) line of an N2O laser via the vibrational aQ(8,7) NH3 transition. The intensity data was recorded by a LeCroy oscilloscope. No further processing happened. The experimental signal to noise ratio was about 300 which means slightly under the half bit uncertainty of the analog to digital conversion. The data is a cross-cut through periodic to chaotic intensity pulsations of the laser. Chaotic pulsations more or less follow the theoretical Lorenz model of a two level system." There are 1000 samples in the time series and it is shown in Figure 4:

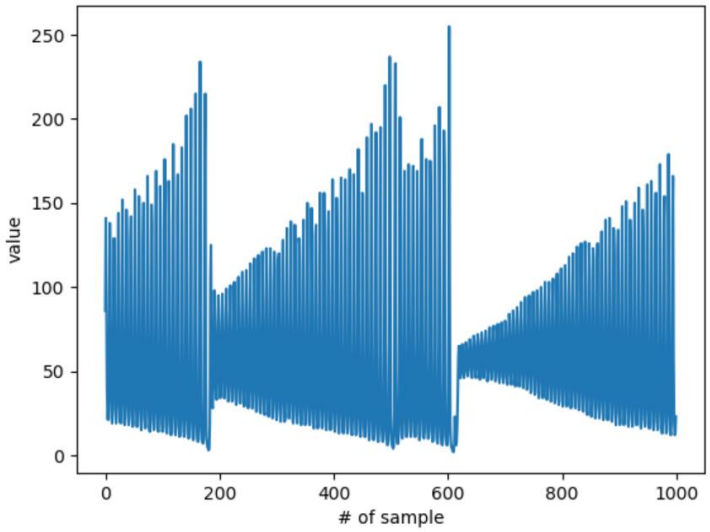

Figure 4: Santa Fe Laser time series

3.2 Preprocessing

As we explained in the previous section, our dataset is of size $n \times 1$ in which n is the number of samples in the time series. But our goal is to train a neural network in which the inputs are some samples of time series and the output is the prediction of future samples. So we have to convert our datasets to the right format.

An important question that comes to mind is that how many previous sample are needed in order for the model to be able to predict the next sample accurately. Another question is that do the previous samples that are being fed to the model need to be consecutive or can they have some distances between them. In fact, in almost every time series forecasting problem, we choose our samples with some distances with each other. Assume we are at time T. If we have *m* previous samples and the distance between two consecutive samples would be *d*, then the forecasting model input is:

$$\text{input} = [y_T, y_{T-d}, y_{T-2d}, \ldots y_{T-(m-1)d}]$$

Therefore, the first problem needed to be solved is finding proper choices for *m* and *d* so that they best describe the relationship between samples and the number of samples that are needed to be known for predicting the future sample. We refer this problem as the delay embedding problem. Fortunately, there is a toolbox named TISEAN [11] that includes functions that allow us to explore what values of *m* and *d* are likely best for a given dataset. Note that we should choose the delay *d* large enough so that y_T and y_{T-d} be essentially independent of each other so they can serve as coordinates of the reconstruction space, but not so independent as to have no correlation with each other [1].

After using TISEAN, we found the best values *m*, *d* for two datasets. For Mackey-Glass these values are found to be $m = 7, d = 11$ and for Santa Fe Laser the value are *m=8* and *d=2*.

Figures below show how we found these values.

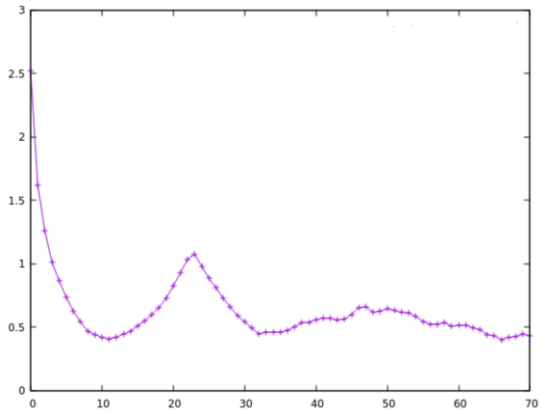

Figure 5: Mackey-Glass delay is equivalent to the first minimum, so d=11

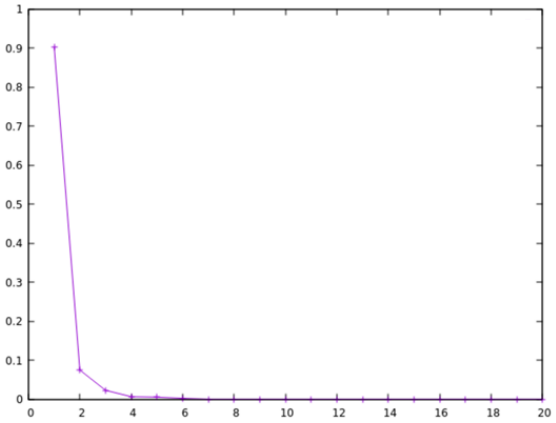

Figure 6: Mackey-Glass dimension is equal to the point which plot reaches zero, so m=7

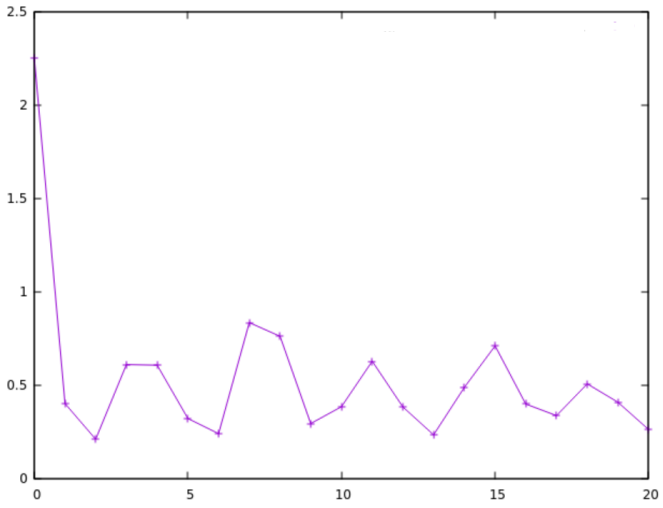

Figure 7: Santa Fe delay is equivalent to the first minimum, so d=2

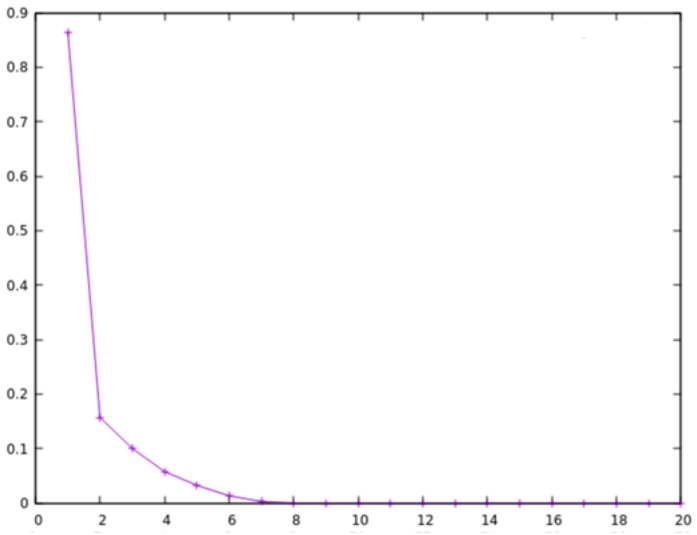

Figure 8: Santa Fe dimension is equal to the point which plot reaches zero, so m=8

3.3 Design Procedures for Neural Networks

For designing our two neural networks (feedforward and recurrent), we used Keras package in Python. Both networks have an input layer with size *m* (dimension found using TISEAN) and an output layer with size 1 (because there is one predicted sample for each set of *m* observed samples). There can be one or more hidden layers and the number of neurons in each layer can be found by performing cross validation. For recurrent network, there is at least one LSTM layer and maybe some dense layers.

3.4 Parameter exploration procedures

For parameter exploration, we used 5-fold cross validation. As we said, we use the first 2/3rd of the dataset as training data and the rest as test data. Obviously, the k-fold cross validation is performed on the training set. Note that we cannot divide the data after preprocessing to k parts and perform cross validation on that. What we have to do is to first divide the original dataset into k parts and for each part, preprocess them and convert them into sequences with length *m* as our inputs and an output. Then each of the preprocessed parts can be used as one fold in our cross validation.

K-fold cross validation means we separate our data into K parts and train the network with the specified parameters K times, each time taking one of those K parts as our validation and the rest as our train data and finally report the result based on average of results on all of K folds.
The parameters that we want to explore are the number of hidden layers and also the number of neurons in each hidden layer. Also, the type of activation function can be found in the parameter exploration process. For recurrent architecture, some hidden layers are LSTM, so we have to decide how many LSTM hidden layers and how many dense (fully connected) layers should be in our network. So, in each k-fold cross validation run, we construct a network with a specific number of hidden layers and a specific number of neurons in each layer.
For recurrent network, we used one hidden LSTM layer. We changed the number of neurons in this hidden layer from 2 to 15. We set the number of neurons in LSTM layer equal to the number that gives us the best results in cross validation. Now we add another fully connected layer and change the number of neurons in that layer and perform cross validation again to find the best parameter. If the best result is better than the best result for one layer, then we use two layers.

For feedforward network, we do exactly the same as we did for recurrent network, but instead of the first hidden layer being LSTM, we use a fully connected layer.
To sum up, we performed cross validation for networks with one and two hidden layers and with each layer has a number between 2-15 neurons. The network that gives best evaluation metrics for k-fold cross validation will be our selected network for performing on test data.

3.5 Performance Measures

As we know, time series prediction is different than classification. In classification problems, the outputs are discrete, which means there are only a few possible values for the output which each are equivalent to the data belonging to one of the classes). But in time series prediction, the output is not discrete and can be any real number. So, the difference between the predicted output and the real one matters here. There are many performance measures defined in the literature for time series prediction. We use NMSE, RMSE and NDEI in this project. Assume the number of samples are n and y_i, $\hat{y}_i$ are the real values and predicted values for those samples, then these measures are defined as follows:

$$NMSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sigma_x^2 n}$$

$$RMSE = \sqrt{\sum_{i=1}^{n}\frac{(y_i - \hat{y}_i)^2}{n}}$$

$$NDEI = \frac{1}{\sigma_x}\sqrt{\sum_{i=1}^{n}\frac{(y_i - \hat{y}_i)^2}{n}}$$

The first and third equations are from [6]. In almost every paper, at least one of RMSE and MSE is used. But RMSE is more preferred, because it is on the same scale as the data [12]. NMSE is the normalized version of MSE. The reason that we chose this value as one of our measures is that since it divides MSE by the variance of data, it can give us a fair comparison between different datasets. The last measure, NDEI, is a combination of these two measure. It first compute RMSE and then divides it by the variance of data, so it can interpreted as a measure that maps a measure in the same scale as data to an interval which it can be compared to other datasets. Also, lots of papers used this measure and also [13] which worked on both of datasets that we used in this project, used this measure as well. All on all, it seems that NDEI is a good choice for using as performance measure in our project.

4 Parameter exploration results

As we explained in the previous section, we performed cross validation for each set of parameters and Tables 1, 2, 3, 4 show the mean and variance of our 3 measures across 5 folds for our two datasets and two algorithms. Note that the range of parameters that we explored are much larger than that we brought in these tables. These tables are some sample of the parameters and obviously, the best parameters are brought in the table. In the tables, one layer and two layer architectures are examined. For recurrent, one layer means one LSTM layer and two layer means LSTM and one dense layer after that. For increasing speed, we had only 100 epochs in each fold of k-fold cross validation, because the comparison is important. For final training, we can have more epochs, so the network would be trained best.

Table 1: The result of parameter exploration with feedforward architecture for Mackey-Glass dataset

	Mean NMSE	Std NMSE	Mean RMSE	Std RMSE	Mean NDEI	Std NDEI
1 layer	1.13	0.73	0.15	0.01	6.69	3.32

with 10 neuron						
1 layer with 100 neuron	1.26	0.85	0.15	0.01	6.95	3.54
2 layers both have 10 neurons	1.11	0.71	0.15	0.015	6.65	3.29
1 layer with 200 neurons	1.18	0.83	0.15	0.016	6.7	3.56

Table 2: The result of parameter exploration with recurrent architecture for Mackey-Glass dataset

	Mean NMSE	Std NMSE	Mean RMSE	Std RMSE	Mean NDEI	Std NDEI
1 layer with 10 neuron	0.78	0.46	0.14	0.014	5.54	2.62
1 layer with 100 neuron	1.00	0.8	0.13	0.019	6.00	3.57
2 layers both have 10 neurons	0.91	0.68	0.12	0.021	5.64	3.3

Table 3: The result of parameter exploration with feedforward architecture for Santa Fe Laser dataset

	Mean NMSE	Std NMSE	Mean RMSE	Std RMSE	Mean NDEI	Std NDEI
1 layer with 10 neuron	0.95	1.07	25.01	8.24	0.23	0.22
1 layer with 100 neuron	1.12	1.37	25.7	9.2	0.024	0.024
2 layers both have 10 neurons	1.09	1.33	25	9.04	0.023	0.024
1 layer with 200 neurons	1.13	0.28	25.9	9.1	0.024	0.024

Table 4: The result of parameter exploration with recurrent architecture for Santa Fe Laser dataset

	Mean NMSE	Std NMSE	Mean RMSE	Std RMSE	Mean NDEI	Std NDEI
1 layer with 10 neuron	2.47	1.24	63.1	6.5	0.049	0.025
1 layer with 100 neuron	0.4	0.16	28.2	7.9	0.016	0.009
2 layers both have 10 neurons	0.65	0.41	32.9	8.13	0.024	0.02
1 layer with 200 neurons	0.29	0.28	17.1	9.54	0.012	0.011

We can see that as the number of neurons increases, the performance would be better. Also, as the variance of Santa Fe Laser dataset is large, so the measure value RMSE which has the same scale as data, would be large. Also, it can be seen that the results of recurrent networks are better than feedforward.

5 Final results and comparison to other works

5.1 Our work's results

From Table 1-4 we can infer that the best set of parameters for both datasets and structures are summarized in Table 5:

Table 5

	Best network type	Number of layers	Number of neurons
Mackey-Glass	Recurrent	1	10
Santa Fe Laiser	Recurrent	1	200

Now that we have found the best set of parameters, we can use them as the ultimate network parameters. So, we can train our networks with these parameters. This time we should use the whole training set (the first $2/3^{rd}$ of dataset) for training our networks.
After we trained our networks with the parameters we found in parameter exploration part, we have to test the network on test data to see the performance of our trained networks on them. As we explained before, for evaluation of time series prediction, we chose three performance measures called NMSRE, RMSE, and NDEI. Table 6 shows these performance measures for the predictions of networks on the test data.

Table 6: The final out of sample test results (all three measures)

	Mackey-Glass	Santa Fe Laser
NMSE	0.06	0.02
RMSE	0.02	2.7
NDEI	0.1	0.005

The large RMSE for Santa Fe Laser dataset is not strange, because the range of data amplitude is large and RMSE is a measure that has the same scale as data.
Figure 5 and Figure 6 are the results on the two datasets.

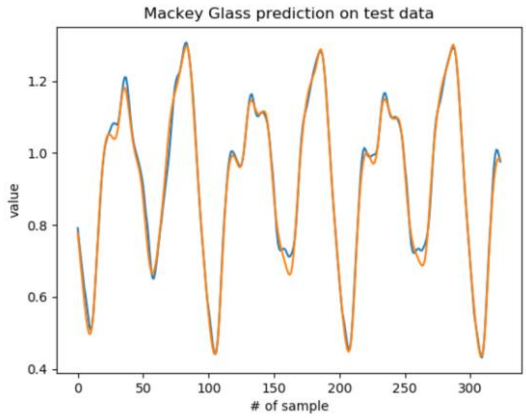

Figure 9: Mackey Glass time series for test data. Blue plots are the prediction time series and the orange plots are the real time series

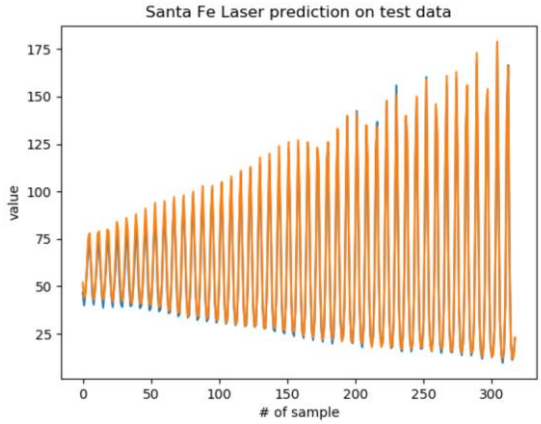

Figure 10: Santa Fe Laser time series for test data. Blue plots are the prediction time series and the orange plots are the real time series

It can be seen that the predictions are very accurate and performance measures are good. So we can conclude that we have succeeded in the prediction of time series using recurrent neural network with LSTM.

5.2 Other Works' Results

There are so many publications that worked with the two time series we used in this project (Mackey-Glass and Santa Fe Laser). They used different approaches for time series predictions. Tables 7 and 8 show some performance measures for different approaches on Mackey-Glass and Santa Fe Laser dataset.

Table 7: Performance measures of different methods on Mackey-Glass dataset. The results are from different works and the references are all references of reference *[13]*

Method	**NDEI**	**MSE**	**NMSE**
ANCFIS	0.0027	3.10e-7	1.37e-6
ANFIS [38]	0.007	-	-
AR model [38]	0.19	-	-
Cascade-corr NN [13]	0.06	-	-
Backprop NN [13]	0.02	-	-
Polynomial [13]	0.04	-	-
Linear predictor [13]	0.55	-	-
TSK-NFIS [23]	0.0406	2.18e-5	-
AR model [23]	0.0492	3.2e-5	-
NAR [23]	0.0466	2.89e-5	-
Neural Network [23]	0.0488	3.21e-5	-
SVR [15]	-	-	4.5e-3
Bagging SVR [15]	-	-	2.0e-3
Boosting SVR 1 [15]	-	-	8.2e-4
Boosting SVR 2 [15]	-	-	8.0e-4

Table 8: Comparison of NMSE measure for Santa Fe dataset with different methods. The results are from different works and the references are all references of reference *[13]*

Method	**NMSE**
ANCFIS	0.0274
[76]	0.028
[46]	0.026
[14]	0.0701
[71]	0.099
MLP [22]	0.0996
MLP IT [22]	0.1582
LSTM [22]	0.3959
LSTM IT [22]	0.3642
Linear [22]	1.2505
FIRN [74]	0.023
sFIRN [74]	0.0273
MLP [44]	0.0177
RSOM [44]	0.0833
EP-MLP [2]	0.2159
[69]	0.077
[78]	0.016
[3]	0.029
Method	**MSE**
ANCFIS	0.001089
[71]	0.0014

The performance measures for our method is a little bit worse than the best methods in the literature. But as we can see, it is better than so many methods. For both datasets, the performance measures of our method seem acceptable, as the calculated performance measures are in a same range as these performance measures in the literature.

6 Conclusions

In this project, we performed two neural networks (feedforward and recurrent) on Mackey-Glass and Santa Fe Laser datasets. These datasets are time series and the goal was to design a method which can predict the value of samples of time series based on the previous samples. In fact, we wanted to find a model with neural network that can model the behaviour of time series. We trained our neural network with different hyper parameters and performed 5-fold cross validation

to find the best parameters for each network on each dataset. After finding the best parameters, we trained our network on the whole training set with the parameters that we found and tested that trained network on test data and evaluate the performance with three performance measures RMSE, NMSE, and NDEI and compared the performance with the existing literatures. Finally, we concluded that the performance of our model is promising and comparable to other methods in literature. With more powerful computers, we can find even better parameters in parameters exploration for our dataset and it is completely possible that the performance would be improved.

7 References

[1] S. Haykin, Neural Networks and Learning Machines, New Jersey: Pearson Education, Inc., 2008.

[2] A. Graves, M. F. S. Liwicki, R. Bertolami, H. Bunke and J. Schmidhuber, "A Novel Connectionist System for Unconstrained Handwriting Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, pp. 855-868, 2009.

[3] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735-1780, 1997.

[4] F. A. Gres, J. Schmidhuber and F. Cummins, "Learning to Forget: Continual Prediction with LSTM," *Neural Computation*, vol. 12, no. 10, pp. 2451-71, 1999.

[5] W. Waddah, "Mackey-Glass time series," figshare, 15 Nov 2016. [Online]. Available: https://figshare.com/articles/Mackey-Glass_time_series/4233584/1. [Accessed 04 Apr 2019].

[6] M. Chandorkar, "System Identification using Gaussian Processes: Santa Fe Laser Data Set," 7 MArch 2016. [Online]. Available: http://mandar2812.github.io/posts/santa-fe-laser/. [Accessed 05 Apr 2019].

[7] M. I. Ullah, "Time Series Analysis and Forecasting," 27 Dec 2013. [Online]. Available: http://itfeature.com/time-series-analysis-and-forecasting/time-series-analysis-forecasting. [Accessed 04 Apr 2019].

[8] M. Cococcioni, "Mackey-Glass time series generator," MathWorks, 2009. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/24390-mackey-glass-time-series-generator?focused=c3a85d03-f40d-8e3c-441d-062d2d41db8a&tab=example. [Accessed 04 Apr 2019].

[9] E. Weeks, "My Adventures in Chaotic Time Series Analysis," [Online]. Available: http://www.physics.emory.edu/faculty/weeks//research/tseries1.html. [Accessed 05 Apr 2019].

[10] [Online]. Available: http://www-psych.stanford.edu/~andreas/Time-Series/SantaFe.html.

[11] R. Hegger, H. Kantz and T. Schreiber, "Nonlinear Time Series Analysis," 06 Jan 1999. [Online]. Available: https://www.pks.mpg.de/~tisean/Tisean_3.0.1/index.html. [Accessed 04 Apr 2019].

[12] R. K. A. B. Hyndman, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, 2006.

[13] Z. Chen, S. Aghakhani, M. James and S. Dick, "ANCFIS: A Neurofuzzy Architecture Employing Complex Fuzzy Sets," *IEEE TRANSACTIONS ON FUZZY SYSTEMS*, vol. 19, no. 2, pp. 305-322, 2011.

[14] F. A. Gres, J. Schmidhuber and F. Cummins, "Learning to Forget: Continual Prediction with LSTM," *Neural Computation*, vol. 12, no. 10, pp. 2451-71, 1999.