COMP6229 (2017/18): Machine Learning Assignment

Student Name: Richa Ranjan
Student ID: 29757282

NEURAL NETWORK APPROXIMATION

Exercise 1: BAYESIAN DECISION THEORY

This exercise deals with Bayesian decision rule for pattern classification. Here, we will be considering a two-class pattern classification problem in two dimensions. In this example, we will be demonstrating both the Bayesian rule, and the Neural networks to compute the posterior probabilities and the decision boundaries. For two Gaussian distributed classes, with distinct means $m_1 = \begin{bmatrix} 0 \\ 3 \end{bmatrix}$, $m_2 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$, and distinct covariance matrices $C_1 = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ and $C_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, Baye's formulae for calculating the posterior probabilities are stated below:

$$P[w_1|x] = \frac{p(x|w_1)P[w_1]}{p(x|w_1)P[w_1] + p(x|w_2)P[w_2]} \quad (1)$$

$$P[w_2|x] = \frac{p(x|w_2)P[w_2]}{p(x|w_1)P[w_1] + p(x|w_2)P[w_2]} \quad (2)$$

The decision rule states that we decide w_1 if P(w_1) $>$ P(w_2); otherwise we decide w_2. Further, the probability densities of Multivariate Gaussians are calculated using the below formulae:

$$p(x) = \frac{1}{(2\pi)^{p/2}(detC_1)^{1/2}} exp[\frac{-1}{2}(x - m_1)^T (C_1)^{-1}(x - m_1)]P[w_1] \quad (3)$$

$$p(x) = \frac{1}{(2\pi)^{p/2}(detC_2)^{1/2}} exp[\frac{-1}{2}(x - m_2)^T (C_2)^{-1}(x - m_2)]P[w_2] \quad (4)$$

Bayes formula shows that by observing the value of x we can convert the prior probability to the posterior probability. The product of the likelihood and the prior probability is the most important in determining the posterior probability. The probability density $p(x)$ can be viewed as a scale factor that guarantees that the posterior probabilities sum to one. For the multivariate normal density formula, the density is constant on surfaces where the squared distance (*Mahalanobis distance*) $(x - m_1)^T C_1^{-1}(x - m_1)$ is constant. These paths are called contours (hyperellipsoids). The principle axes of these contours are given by the eigenvectors of the covariance matrix, where the eigenvalues determine the lengths of these axes. When the covariance matrices are different for the two classes, the resulting discriminant functions are quadratic. The decision surfaces are therefore, hyperquadratics. A quadratic decision boundary is shown in figure 1.

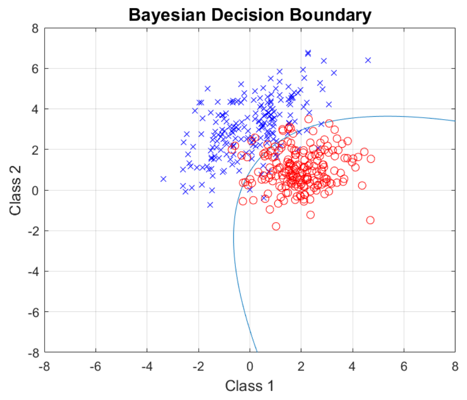


Fig. 1: Bayesian decision boundary in two dimensions

For given posterior probability of one class $P[w_1|x] = 0.5$, the curve is plotted using the Baye's formula. This curve, as plotted in a three dimensional graph resembles the shape of a sigmoid function, as shown in figure 2. When this plot is rotated, the quadratic decision boundary is very clearly visible, as in figure 3.

Exercise 2: FEEDFORWARD NEURAL NETWORKS

Feedforward networks are Artificial Neural Networks, where the units are not connected in a cyclic form. The first layer is the input layer, the last layer is the output

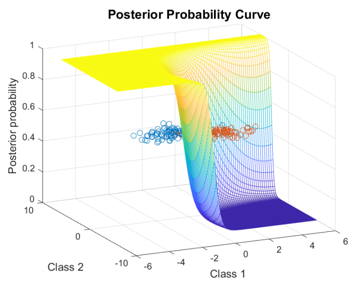


Fig. 2: Posterior Probability curve

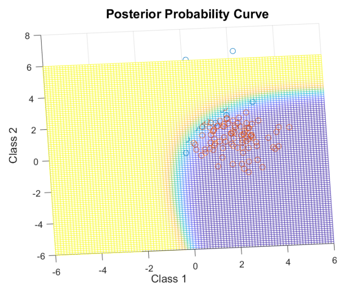


Fig. 3: Quadratic Decision Boundary

layer and as there are no loops, the information only traverses in the forward direction, from input to output layers, via one or more hidden layers. The goal of a feedforward network is to approximate some function f for a classifier, $y = f(x)$ maps an input x to a category y. On training the neural network in a feed forward fashion for pattern classification, the output obtained is presented in figure 4.

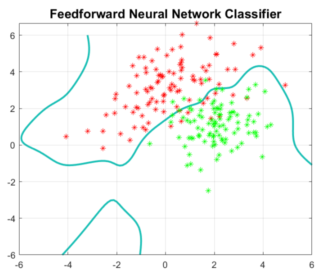


Fig. 4: Feedforward Neural Network decision boundary

In a feedforward neural network, training inputs are applied to the input layer of the network, and desired outputs are compared at the output layer. During the learning process, a forward sweep is made through the network, and the output of each element is compared layer by layer. The difference between the output of the final layer and the desired output is back-propagated to the previous layer, usually modified by the derivative of the transfer function, and the connection weights are normally adjusted using the *Delta Rule*. Also, after the training is over, the neural network behavior can be observed, as in figures 5 and 6.

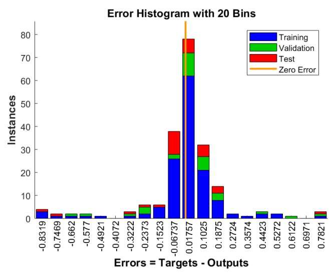


Fig. 5: Error Histogram

In figure 5, we see the error histogram, which shows that the error instances have a normal distribution-like structure.

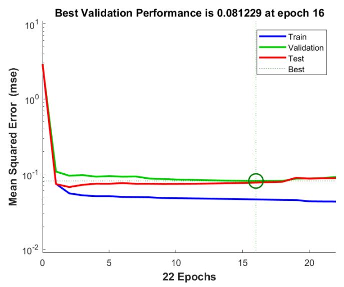


Fig. 6: Validation Performance

The performance graph in figure 6 shows the Mean squared errors for Training, Test and Validation data sets. **Inference:** From the above observations, it is clearly evident that the Baye's decision boundary is a better classifier when compared to the neural network output. Estimating classifier accuracy is important to evaluate how accurately a given classifier will classify unknown samples on which it has not been trained. The Naive Baye's classifier performs surprisingly well with small amounts of training data that is significantly insufficient for Neural Networks. This is why the Baye's classifier produces a better result with this amount of data.

TIME SERIES PREDICTION

CHAOTIC TIME SERIES: MACKEY GLASS MODEL

Machine Learning techniques are instrumental in Time series prediction applications. The *Mackey-Glass* model is a popular chaotic time series. It is obtained by integrating the nonlinear differential equation,

$$\frac{dx}{dt} = \frac{ax(t-\tau)}{1+x(t-\tau)^{10}} - bx(t) \tag{5}$$

It can be numerically solved using, for example, the 4th order **Runge-Kutta** method, at discrete, equally spaced time steps: $x(t+\Delta t) = mackeyglass\_rk4(x(t), x(t-\tau), \Delta t, a, b)$ where the function *mackeyglass-rk4* numerically solves the Mackey-Glass delayed differential equation using the 4th order Runge Kutta. This is the RK4 method:

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

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

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

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

$$x(t+\Delta t) = x(t) + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{6} + \frac{k_4}{6}$$

where *mackeyglass-eq* is the function which returns the value of the Mackey-Glass delayed differential equation in (5) once its inputs and its parameters (a,b) are provided. A typical Mackey Glass time series is shown in the figure 7. Using Mathwork's pre-defined functions, a Time Series of sample size 2000 is generated.

A Neural Network architecture uses Finite Impulse Response linear filters to provide dynamic interconnectivity between processing units. The network is applied to the chaotic time series prediction task. The neuron receives the inputs and then passes the sum through a nonlinear squashing function. Neurons are arranged in layers to form a network. Training the network is accomplished through a modification of the backpropagation algorithm called temporal backpropagation in which error terms are symmetrically filtered backward through the network.

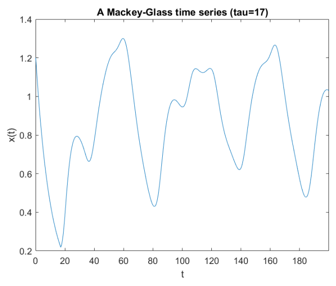


Fig. 7: Mackey Glass Time Series

IMPLEMENTATION

For this process, we use the first *N=1500* samples to train a prediction model and the remaining *500* as our test data. Now, we construct a design matrix with *p=20*, where after every 20 iterations, the output of the last row gets shifted by one step each. Our input matrix has *N-p+1* rows and p columns, with each row being a time shifted version of the previous one. A linear predictor is estimated from the training data and a one step ahead prediction is carried out on the test data. This prediction graph is shown in figure 8.

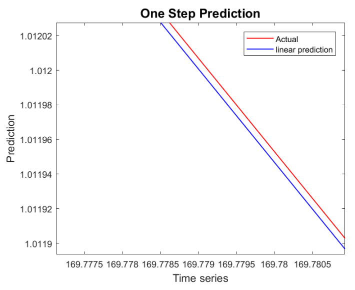


Fig. 8: Linear Predictor for One Step Prediction

Observation: As evident from the above figure, the actual and the predicted values are closely placed with each other. There is very minimal gap between these curves (visible only when the figure is zoomed in).Thus it can be inferred that the linear predictor performs really well with one step prediction.

A feed forward Neural Network is also trained for this prediction. The Neural Network output and the training values are shown in figures 9 and 10 respectively.

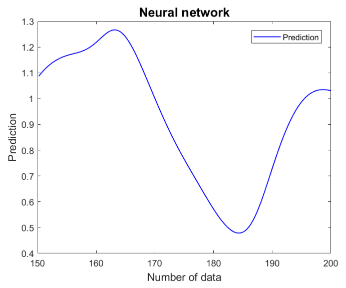
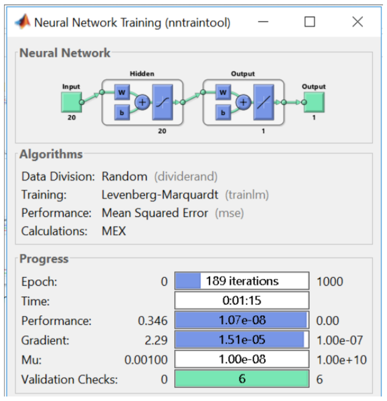


Fig. 9: Neural Network One Step Prediction


Neural Network Training (nntraintool)

Neural Network

Algorithms

```
Data Division:  Random  (dividerand)
Training:       Levenberg-Marquardt  (trainlm)
Performance:    Mean Squared Error  (mse)
Calculations:   MEX
```

Progress

```
Epoch:                 0   189 iterations   1000
Time:                      0:01:15
Performance:       0.346   1.07e-08         0.00
Gradient:           2.29   1.51e-05         1.00e-07
Mu:              0.00100   1.00e-08         1.00e+10
Validation Checks:     0   6                6
```

Fig. 10: Neural Network Architecture

As we can see from the graphs, the Neural Network performs really well with the one step prediction problem, as good as the linear predictor. This is because the sample size is really small, i.e. significantly insufficient for a Neural Network model. For such a small sample size, the result of a one step prediction is remarkably accurate. Also, the error histogram and performance graphs are shown in figures 11 and 12.

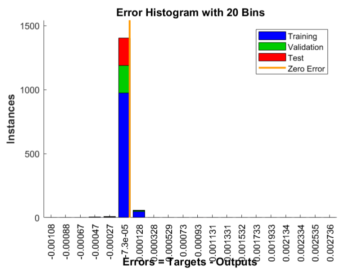


Fig. 11: Error Histogram for Time Series Prediction

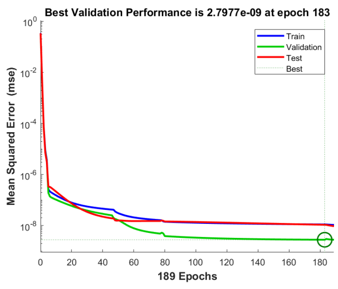


Fig. 12: Performance for Time Series Prediction

The mean squared error has the best performance at 2.7977e-09 for the validation data. During the training, the squared error is minimized by using the temporal backpropagation algorithm to adapt the network. Once the network is trained, long-term iterated prediction is achieved by taking the estimate and feeding it back as input to the network. The system can be iterated forward in time to achieve predictions as far into the future as desired. The most important part of the network is the hidden layer. Hidden layer is composed of nodes which are connected to both output and input layers. The output nodes are used for prediction of the future value of the time-series. In the feedforward neural network, the information flow is uni-directional. This kind of model assumes that there is a relationship between future value and the past observation and neural networks are used for identifying this relationship.

FREE RUNNING MODE

For this exercise, we train the Neural Network, it then predicts the next sample, we use this prediction into the input for the next iteration to predict the following sample. In these steps, we proceed without using the truth values. If we train the Neural Network on the Time Series data, the network learns to approximate the underlying generating function so well that if we let it run in a free running mode, up to a few cycles, it generates the whole time series! For the free running mode, the graph looks like the one shown in figure 13.

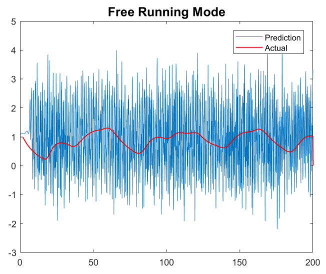


Fig. 13: Free Running Mode for Time Series Prediction

In the equation 5, as mentioned, 'τ' is the delay constant. As its value is increased, an initially stable equilibrium becomes unstable and stable periodic solutions appear. As 'τ' is further increased, we observe cycles with periods apparently chaotic or aperiodic regime.

Observation: As we see from the graph, stable but complex oscillations appear in the prediction values. For the actual data, cycles with periodic oscillations are observed. If our model is reasonably good, in a free running mode, it generates a couple of iterations of those cycles, and then it loses track. This happens because the prediction will always have an error, and when we feed this error back, the error can accumulate and we may lose track. If it is trained well, it goes well for few cycles.

FINANCIAL TIME SERIES

The Neural Network training methodology used in the above exercise is used for the prediction of a Financial Time Series as well. The training process is same as used in case of the Time Series prediction. Here, our aim is to predict the FTSE index value for the next day.

IMPLEMENTATION (WITH PRICE ONLY)

For this prediction, the daily `FTSE100` data is obtained for the past five years from the finance data provider called *Investing.com UK*. The **Price** values are used for making the predictions. Similar to the above prediction problem, a design matrix with *p=20* is created. The input matrix has each row as a time shifted version of the previous one. In this case, the value of p being 20 means the Price index values are taken for past 20 trading days, and the prediction is made for tomorrow. The Neural Network is trained to make predictions with the Price index values, and the resulting graph looks like the one shown in figure 14.

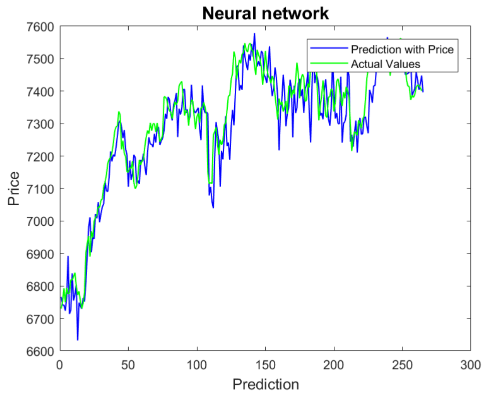


Fig. 14: Neural Network Prediction for Price index

As can be seen, the prediction is nearly accurate at the start, but then it diverges after few rounds, and then again it's in line with the actual values, and diverges slightly again. This divergence is unavoidable and reveals one of the important tenets of chaos theory.

Free Running Mode: For the free running mode, we train the Neural Network in a manner similar to that in Time Series prediction. The model predicts the next sample, we use this prediction into the input for the next iteration to predict the following sample. For the free running mode, the performance graph looks like the one shown in figure 15. The mean squared error at 4426.6933 has the best performance for validation set.

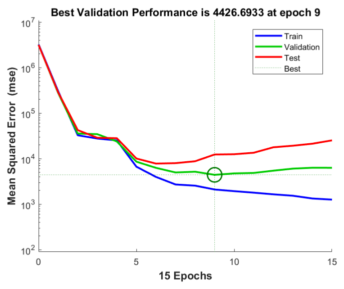


Fig. 15: Neural Network Prediction performance

For this mode, the prediction graph looks like the one shown in figure 16. As we see in the figure, the prediction values oscillate in the beginning and later, the graph becomes steady.

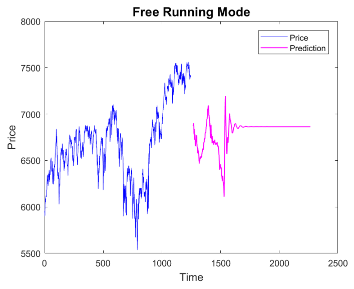


Fig. 16: Price prediction in the Free Running Mode

Inference: In my view, this Neural Network model would give me an opportunity to make money, as it is evident from the graphs and results above, that the network has been trained well. If we see the network prediction plotted with the actual values in figure 14, we can see that the prediction values are close enough when compared to the actual prices.

IMPLEMENTATION (WITH PRICE AND VOLUME)

In the same network architecture as described above, if along with the Price index, the **Volume** Traded information is used as an input, we would see different results and see if an additional input would improve the predictions or not. From the same data set, the Volumes are also taken for the past 20 days and along with the Price values, they are fed into the Neural Network for making predictions for the next day. The Neural Network results for these two inputs are shown in figure 17.

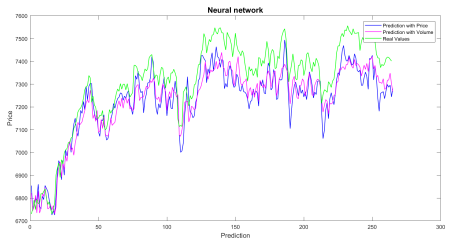


Fig. 17: Neural Network Prediction with Price and Volume

Free Running Mode: For the Free Running Mode, the Network is trained exactly in the same way, and we have a prediction graph that looks like the one shown in figure 18.

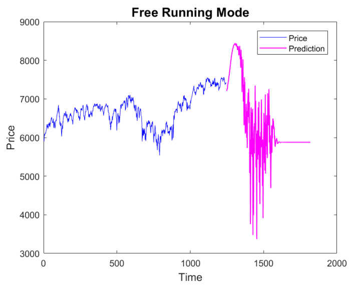


Fig. 18: Prediction in the Free Running Mode with Price and Volume as inputs

As seen in the figure, after oscillating for a few cycles, the graph becomes steady. The performance graph for this training is shown in figure 19.

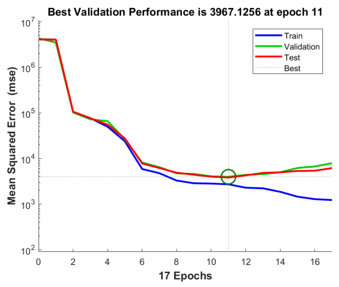


Fig. 19: Neural Network Performance with Price and Volume

The mean squared error value has the best performance for validation at 3967.1256. This value is better than the error value obtained when the network was trained just with the Price value. Therefore, in my view, taking the Volume information as an input to the Neural Network, would be advisable as it produces better results than that with just the Price values. The prediction ability has, in my opinion, improved in this case.

Prediction of Chaotic Time Series With NAR Neural Network

As an addition to the Mackey Glass Time Series discussed above, we can design a Neural Network for the recursive prediction of this series. A **Nonlinear Auto Regressive** Neural Network is defined, and after the network is trained, it is transformed into a closed-loop NAR network. The Matlab functions `narnet` and `preparets` are used for the same. This function automatically shifts input and target time series as many steps as are needed to fill the initial input and layer delay states. If the network has open-loop feedback, then it copies feedback targets into the inputs as needed to define the open-loop inputs. We can then perform a recursive prediction on the validation data. The architecture and results are shown in figures 20 and 21.

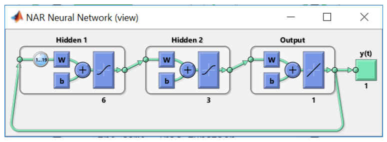


Fig. 20: NAR Neural Network Architecture

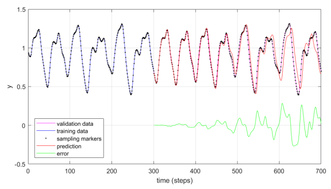


Fig. 21: NAR Neural Network Time Series Prediction

References

[1] https://www.byclb.com/TR/Tutorials/neural_networks/ch4_1.htm

[2] https://arxiv.org/ftp/arxiv/papers/1008/1008.3282.pdf

[3] M. C. Mackey and L. Glass, Oscillation and chaos in physiological control systems, Science, vol. 197, no. 4300, pp. 287289, 1977//

[4] E. A. Wan, Modeling nonlinear dynamics with neural networks: Examples in time series prediction, in Proceedings of the Fifth Workshop on Neural Networks, pp. 327232, 1993.