KU LEUVEN

Artificial Neural Networks

Exercise Sessions

Meng YUAN (r0601195)

Master of Artificial Intelligence

Academic year 2017-2018

Session 1 Function Approximation

In this session, we compared different learning algorithms regarding the function approximation problem. We generated the data set using $y = sin(x^2)$ for $x = 0 : 0.05 : 3\pi$ as a simple nonlinear function example and tried to approximate it using a neural network with one hidden layer. The hidden units number for this problem was chosen to be 50 since the neural network with 50 hidden neurons can provide good fit.

Different training algorithms were compared in terms of their performance, speed and generalization capability. Mean squared errors(MSEs) on test sets was used to measure the performance and generalization capability. Each time 10 neural networks were trained and we took the average of MSEs. The speed is characterized by the total running time of 10 iterations. Also, regression analysis between the network outputs and the corresponding targets was performed. If there were a perfect fit, the slope would be 1, and the y-intercept would be 0. R-value is the correlation coefficient between the outputs and targets. The more R-value is closer to 1, the better fit.

1 noiseless case

In Table 1.1, it provides MSE and the total running time of different methods. It can be concluded that Levenberg-Marquardt algorithm provides good performance and the fastest convergence. Bayesian Regularization algorithm provides best performance yet uses quite long time. Quasi-Newton algorithm and conjugate gradient algorithm give less good fit and use more time than Levenberg-Marquardt algorithm. Quasi-Newton algorithm performs better but use slightly longer time than conjugate gradient algorithm. Similar conclusions of model fit can be made according to the results of regression analysis given in Figure 1.1.

	Levenberg-Marquardt	quasi Newton	Conjugate Gradient	Bayesian Regularization
MSE	0.0731	0.1539	0.3257	1.1267e-04
Time	1.9776	4.1873	3.0029	65.4030

Table 1.1: MSE and Total running time

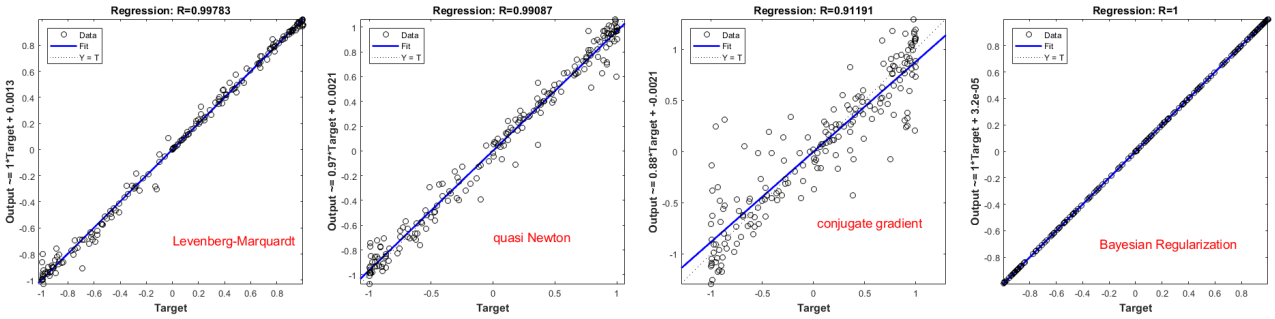

Figure 1.1: Regression analysis on trained networks

2 noisy case

When noisy was added, all methods provides less good performance than their corresponding noiseless cases. However, the conclusions about performance and speed of different are the same. Bayesian Regularization algorithm provides best performance, Levenberg-Marquardt algorithm ranks second, quasi-Newton algorithm performs slightly less good and conjugate gradient algorithm gives least good result. Bayesian Regularization algorithm runs for quite long time, Levenberg-Marquardt algorithm converges fastest, conjugate gradient algorithm and quasi-Newton algorithm run slightly longer than Levenberg-Marquardt algorithm.

	Levenberg-Marquardt	quasi Newton	Conjugate Gradient	Bayesian Regularization
MSE	0.1050	0.1695	0.3894	0.0380
Time	2.0712	4.6345	2.6667	90.0604

Table 2.1: MSE and Total running time

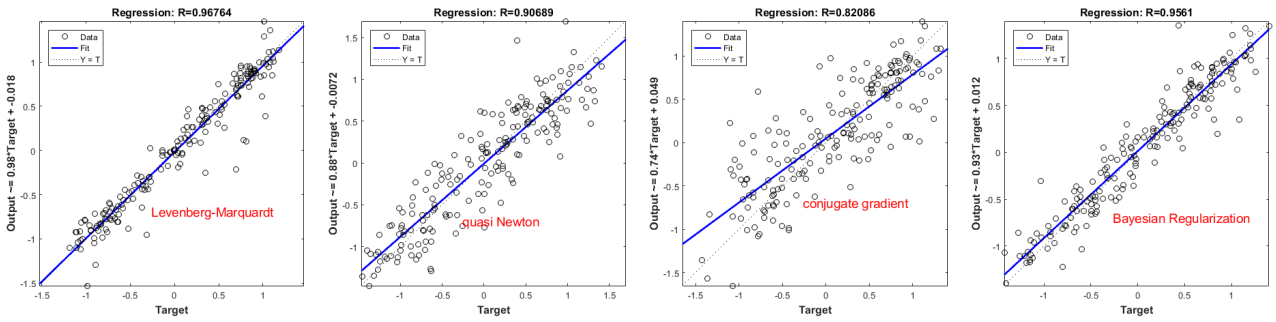

Figure 2.1: Regression analysis on trained networks

To investigate the generalization capability, we considered overparametrized networks (150 neurons). According to Table2.2, Bayesian Regularization algorithm can control the complexity of the model and generalize well when overfitting occurs.

	Levenberg-Marquardt	quasi Newton	Conjugate Gradient	Bayesian Regularization
MSE	0.5518	0.6361	0.9103	0.2777
Time	2.5946	18.0199	3.8279	334.8017

Table 2.2: MSE and Total running time

Session 2 Recurrent Neural Networks

1 Hopfield Networks

In this session, we investigate the key feature of Hopfield network - its dynamics tend to converge to attractor states - by checking its ability to correctly retrieve noisy handwritten digits 0;...; 9. The dataset includes 198 examples of 15 x 16 pixel digits where each element is represented by binary data. Since Hopfield network is fully connected recurrent neural network, the resulting network contains 240 neurons and a 240 x 240 weight matrix. 10 instances are selected as equilibrium points and some noise are introduced to them. The level of noise and the number of iterations are modified during different trials of training Hopfield network. The idea is to test whether the trained Hopfield network can return to the equilibrium points.

Mostly the reconstructed images converge to the correct attractors when relatively high number of iterations are applied for less noisy images. However, sometimes the network converges to undesireable equilibrium points, i.e. spurious patterns, which could be 1.the negation of stored pattern, 2. linear combination of an odd number of stored patterns, 3. local minima that are not correlated to any linear combination of stored patterns.[1] In Figure 1.1(limited number of iterations for very noisy data), it can be seen that digits 0,4 and 7 converged perfectly to the attractors; digits 2,3,5,6 and 9 converged to a stable state that differs a bit from the stored memories; digits 1 and 8 not converging to the correct states.

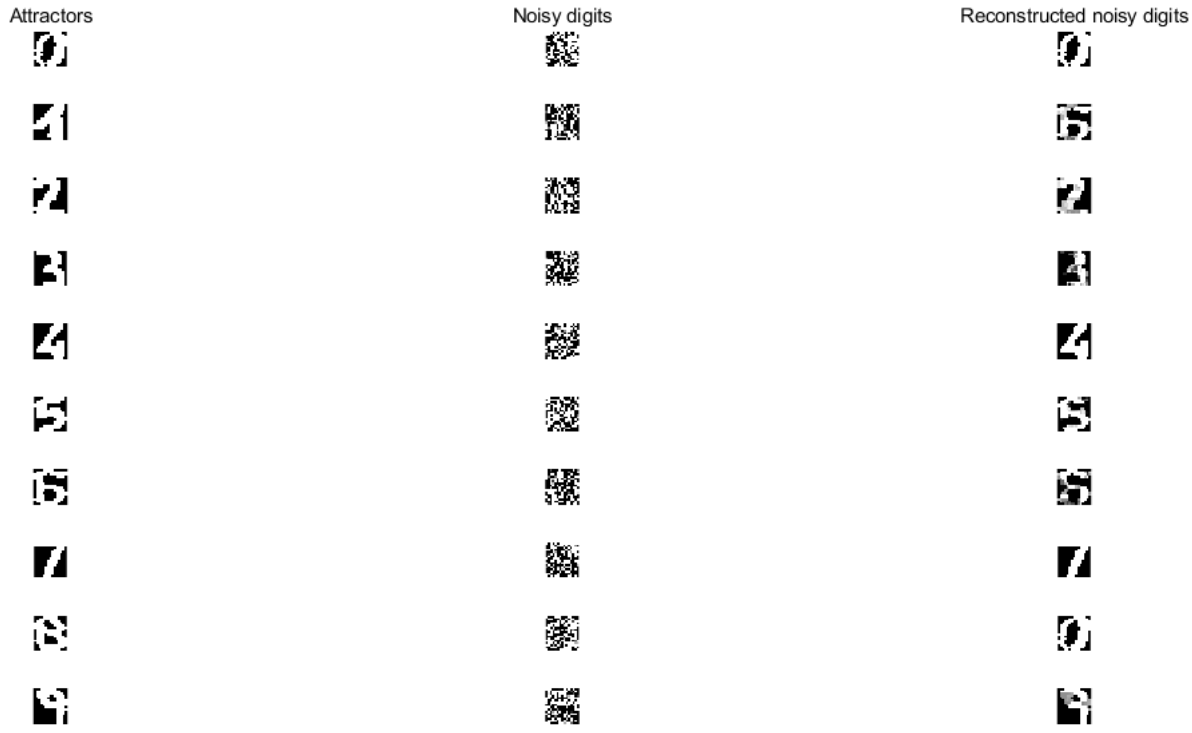

Figure 1.1: Results of Hopfield NN for handwritten digit patterns
(noiselevel = 7, number of iterations = 15)

[1]http://www.doc.ic.ac.uk/ ae/papers/Hopfield-networks-15.pdf

2 Elman Networks

Elman network differs from conventional recurrent neural networks in that each subsequent hidden layer has not only weights from previous layer but also recurrent weights. The output values of hidden layer neurons are saved and fed back fully connected to hidden layer neurons. However, the output layer values are not fed back to network. This recurrent connection allows Elman network to both detect and generate time-varying patterns.[2]

In this session, Elman network is applied to model the behavior of Hammerstein system. Different training settings were compared in terms of their performance, which are measured by the average of 10 runs' Mean square error(MSE) on the test set.

According to Table 2.1, as the number of epochs increases, the performance becomes better. The architecture of the network was adjusted by increasing the number of hidden neurons to 100 or adding one additional hidden layer with 50 neurons. The results show that using more complex networks doesn't necessarily give better performance duo to overfitting. The original dataset has 1000 points (300 for training, 200 for test), and we tried to model with more examples, i.e. 5000 points (1500 for training, 1000 for test). By increasing the number of inputs, the performance becomes better. Different structures of Hammerstein system are also compared. Hammerstein systems with the random terms following sine and cosine function behave more or less the same. However, system with small parameter value gives better performance. Because larger parameter represents less stable time series, which are more difficult to model.

Time-series	**No of inputs**	**Architectures**	**No of epochs**	**MSE**
$\begin{cases} x(t+1) = 0.8x(t) + sin(u(t+1)) \\ y(t+1) = x(t+1) \end{cases}$	1000	1 Hidden Layer(L1:50)	1000	0.4243
			3000	0.2330
			6000	0.1885
	1000	1 Hidden Layer(L1:100)	1000	0.3643
			3000	0.3385
			6000	0.2120
	1000	2 Hidden Layers(L1:50, L2:50)	1000	0.2847
			3000	0.2501
			6000	0.2457
	5000	1 Hidden Layer(L1:50)	1000	0.2198
			3000	0.1984
			6000	0.1682
$\begin{cases} x(t+1) = 0.8x(t) + cos(u(t+1)) \\ y(t+1) = x(t+1) \end{cases}$	1000	1 Hidden Layer(L1:50)	1000	0.3138
			3000	0.2832
			6000	0.2123
$\begin{cases} x(t+1) = 0.3x(t) + sin(u(t+1)) \\ y(t+1) = x(t+1) \end{cases}$	1000	1 Hidden Layer(L1:50)	1000	0.1002
			3000	0.0914
			6000	0.0175

Table 2.1: MSE for different settings

[2]Time Series Prediction with Multilayer Perceptron, FIR and Elman Neural Networks

Session 3 Unsupervised learning

1 Handwritten Digits PCA and reconstruction

Principal Component Analysis(PCA) reduces the dimensionality of original data by projecting it onto lower-dimensional space. It only retains principal components(PCs) that can capture the important structure of data, thus eliminates redundancy. In this session, PCA is applied on handwritten images of the digit 3.

Figure 1.2 shows the eigenvalues of the covariances matrix of the whole dataset in a descending order. It can be observed that eigenvalues decrease quickly, first 20 eigenvalues are larger than 0.5 and eigenvalues after 50 are close to 0. We compassed the dataset by projecting it onto k PCs and then reconstructed it. In figure 1.1, mean, original and reconstructed images are displayed. It shows that the more PCs used the better reconstruction quality.

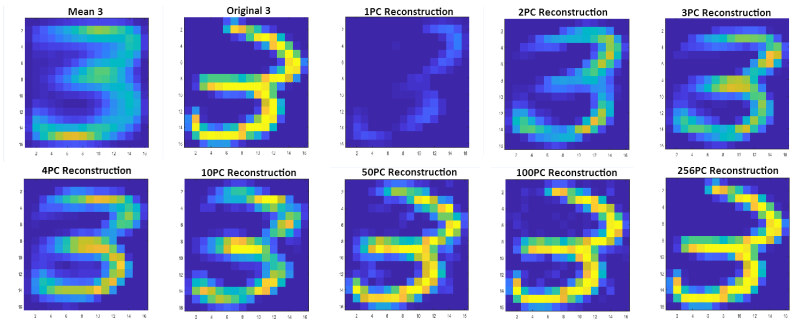

Figure 1.1: Mean, original and reconstruction images

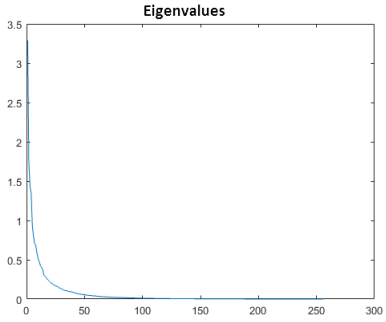

Figure 1.2: Eigenvalues

We measured the reconstruction error by calculating the root mean square difference between the reconstructed and the original data, as shown in figure 1.3. It can be concluded that as more PCs are used in the reconstruction, the RMS error decreases until it approaches zero (when k=256, all information are used). In figure 1.4, the sum of all but the i largest eigenvalues are plotted. The sum of unused eigenvalues declines quickly. As the reconstruction error caused by unused PCs declines, the sum of unused eigenvalues declines proportionally. That is why we can capture the important information of the original data by only retaining first few PCs because non-participating eigenvalues are not very large.

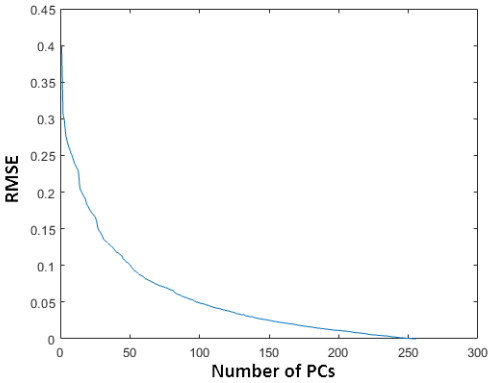

Figure 1.3: Reconstruction error

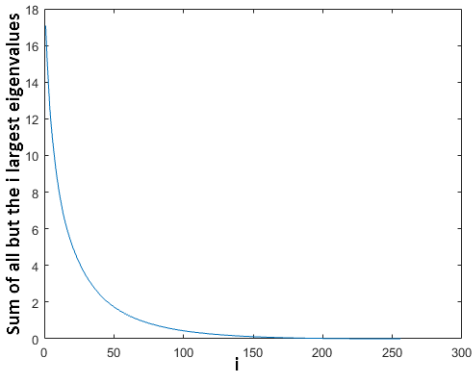

Figure 1.4: Sum of unused eigenvalues

2 SOM applied to the Iris datasets

This part shows how self-organizing maps(SOMs) cluster iris flowers into natural classes topologically according to four flower attributes. We investigated how different choice of the grid size, topology and epochs number can influence the performance. The mean Adjusted Rand Index(ARI) of 10 iterations is used to evaluate the performance. Visualization of the position evolvement of prototypes are displayed in Figures. The SOM weight positions figure plots the input vectors in three colors according to their true classes and shows how the SOM classifies the input space by showing red dots for each neuron's weight vector and connecting neighboring neurons with blue lines.[1]

Grid size	epochs	ARI
1x3	50	0.5482
1x3	100	0.6615
1x3	200	0.7163
1x3	300	0.7246

Grid size	ARI
1x3	0.7163
2x2	0.6162
3x3	0.6159
4x4	0.2845

Table 2.1: ARI for different epochs(left) and grid sizes(right) with gridtop topology

In Table 2.1, it provides ARI for different training settings. According to left table, the performance becomes better as the number of epochs increase and after 200 epochs the values of ARI keep slight fluctuations around 0.72. Figure 2.1 also shows that the discriminate ability increases with the increase of number of epochs.

According to right table in Table 2.1, the performances of 2x2 and 3x3 rectangular grid SOM are similar and worse than 1x3 rectangular grid SOM. The 4x4 rectangular grid SOM gives the worst result because of extreme overfiting with 16-neurons. Figure 2.2 also shows how complex SOM neuron networks overfit the data.

Different topologies for the original neuron locations were specified with the functions gridtop, hextop or randtop. The gridtop topology starts with neurons in a rectangular grid; the hextop function creates neurons in a hexagonal pattern; randtop generates a random pattern of neurons.[2] Although the prototypes distribute differently before training, the final results are similar.

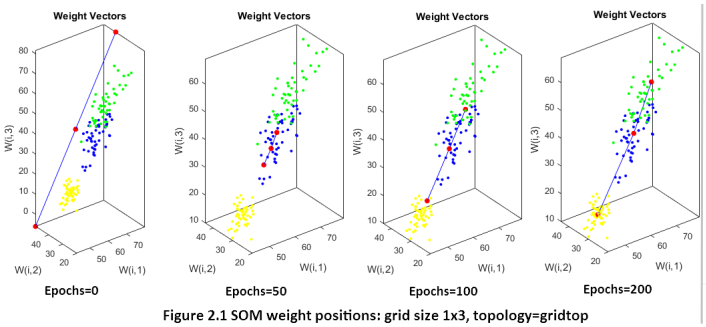

Figure 2.1 SOM weight positions: grid size 1x3, topology=gridtop

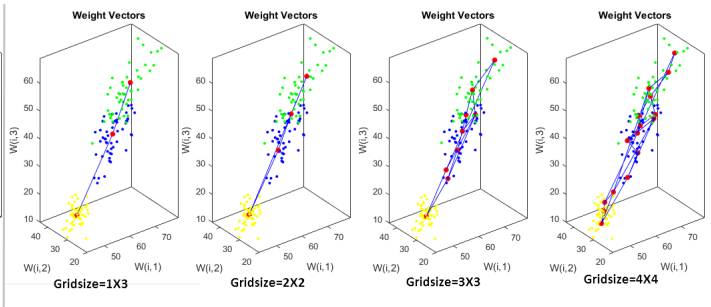

Figure2.2 SOM weight positions: epochs=300, topology=gridtop

[1] https://nl.mathworks.com/help/nnet/ref/plotsompos.html
[2] http://matlab.izmiran.ru/help/toolbox/nnet/selfor11.html

Session 4

1 Stacked Autoencoders

Training multilayer neural networks for complex dataset can be difficult in practice. To train more effectively, we can apply Stacked Autoencoder. It firstly trains each layer individually and then fine-tuning using backprorogation in the end.

In this session, we compared the performance of Stacked Autoencoders and normal neuron networks with multiple hidden units based on the classification problem of handwritten digits. The default architecture built of stacked autoencoders can be summarized as a 784-100-50-10 deep network. First, a sparse autoencoder is trained on 784 input data points to learn primary features. Then, the features generated from the first autoencoder's 100 node hidden layer are used as inputs for the second sparse autoencoder to learn secondary features. Next, a softmax layer with 10 neurons is trained to classify the 50 secondary features in a supervised fashion. Finally, fine-tuning is applied for the combined stacked autoencoder with 2 hidden layers and a softmax classifier layer.

According to Table 1.1, we can investigate the influence of the different model parameters on the model performance and compare the performance of Stacked Autoencoders to normal multilayer neural networks. The results were obtained by averaging over 10 runs. It can be concluded that the accuracy improved a lot by performing backpropagation on the whole multilayer network (i.e. fine-tuning). By increasing the MaxEpoches, the pre-tuning accuracy increased but the fine-tuning accuracy was not necessarily improved. More complex model with a 784-125-80-10 architectural and simpler model with a 784-85-40-10 architectural were compared while keeping MaxEpoches constant. A better result with fine-tuning accuracy equal to 99.75 was obtained with simpler model. However, more complex model provided sightly worse result than default setting. This could possibly be explained by overfitting. Normal 1-hidden layer and 2-hidden layer normal neuron networks provided similar results and were always worse than the Stacked Autoencoders.

Stacked Autoencoders							**1 Layer NN**		**2 Layer NN**		
Layer1	Max Epochs1	Layer2	Max Epochs2	Max Epochs3	PreTuning Accuracy	FineTuning Accuracy	Layer1	Accuracy	Layer1	Layer2	Accuracy
100	400	50	100	400	86.08	99.68	100	96.83	100	50	96.81
100	600	50	300	600	94.74	98.80	100	96.83	100	50	96.81
125	400	80	100	400	92.76	99.66	125	97.27	125	80	96.40
85	400	40	100	400	74.44	99.75	85	95.51	85	40	96.42

Table 1.1: Performance of different neuron networks

2 CNN

In this part, we investigated the architecture of a widely used convolutional neural network, AlexNet. It contains 5 convolutional layers and 3 fully connected layers.

The first convolutional layer filters the 227x227x3 input image using 96 kernels of size 11x11x3 with stride [4 4] and no padding. According to the visualization of the network filter weights, it can be seen that each learned 55x55 weight matrix captures one of the 96 basic image features, such as edges and blobs.

ReLU and Cross Channel Normalization layer do not have an impact on the dimension of the input. Maxpooling layers summarize the information of neighboring groups of neurons and thus reduce the spatial size of the representation. It partitions the input image into a set of non-overlapping rectangles and retains the maximum value for each rectangle. The input to layer 5 (Maxpooling) is 55x55x96. It performs downsampling by taking the maximum input within the 3x3 subregion and then moving 2 units horizontially to next subregion. This provides 27x27x96 outputs and will be fed to layer 6. [Output dimension=(Input dimension - Filter size + 2*Padding)/Stride + 1 =(55-3)/2+1=27]

The final softmax classifier has 1000 neurons which produces a distribution over the 1000 class labels. Compared to the initial input size, the dimension of the problem is intensively reduced but the important information are retained. That is why we use AlexNet as a feature extractor.

In this part, we compared the performance of different CNN architectures based on the handwritten digits classification problem. The obtained results are shown in Table 2.1. Compared to the original architecture, by adding an additional Maxpooling layer after the second convolutional layer, the accuracy decreased a bit since maxpooling performs downsampling and throws away some information. Though less features was fed to the final classifier, the total running time increased a bit because an extra layer also increased the time complexity and dimensionality reduction was added quite late in the network. Compared to the original architecture, by adding a third convolutional layer, the accuracy increased a bit(from 0.988 to 0.990), yet costed much more time(from 114s to 264s). So it does not worth it to add more convolutional layer. On the other hand, reducing the amount of convolutional layers was also tried. The CNNet with only one convolutional layer resulted in lower accuracy yet consumed much less time. By increasing the filter size and number of filters of the convolutional layer, the accuracy increased.

CNN Architecture	**Time(seconds)**	**Accuracy**
Conv1(5,12)+Relu, Maxpooling(2), Conv2(5,24)+Relu, FC(10)	114	0.9880
Conv1(5,12)+Relu, Maxpooling(2), Conv2(5,24)+Relu, Maxpooling(2), FC(10)	135	0.9812
Conv1(5,8)+Relu, Maxpooling(2), Conv2(5,16)+Relu, Conv3(5,32)+Relu, FC(10)	264	0.9900
Conv1(5,20)+Relu, Maxpooling(2), FC(10)	49	0.9712
Conv1(7,32)+Relu, Maxpooling(2), FC(10)	89	0.9848

Table 2.1: Performance of different CNNs

Final project

1 Nonlinear regression and classification with MLPs

1.1 Regression

In this session, the feedforward neuron network is applied to approximate a nonlinear function. The function was constructed from 5 independent nonlinear functions, namely, $f(X_1, X_2) = 9f_1(X_1, X_2) + 6f_2(X_1, X_2) + 5f_3(X_1, X_2) + f_4(X_1, X_2) + f_5(X_1, X_2)$. To obtain training, validation and test set, 3 samples of 1000 points are selected from the given points. To ensure good generalizability of the model, the 3 samples should be independent and cover the whole domain of the given dataset. Neighboring points and periodic sampling should be avoided. By using randomized list of indicies, representative and independent samples could be generated. Figure 1.1 shows the surface of the generated training set.

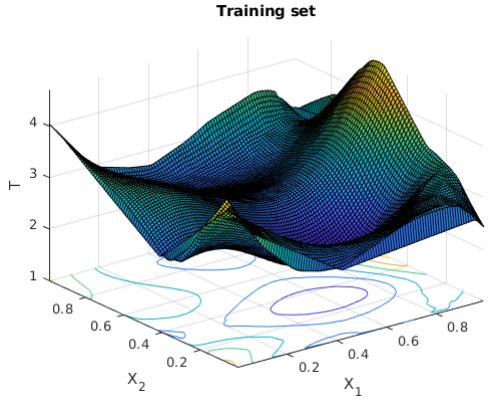

Figure 1.1: Surface of the training set

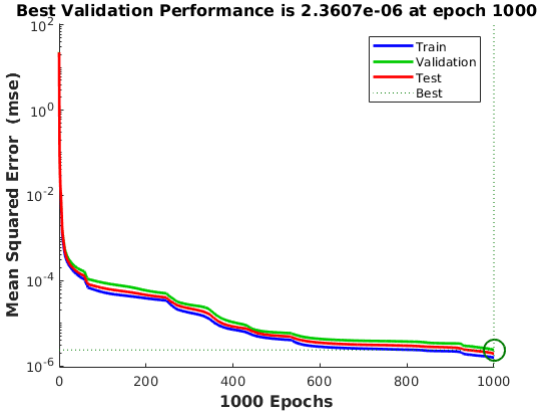

Figure 1.2: Error curve

To search for the best model, the MSEs on validation set and time complexity of different networks were computed (averaging 10 runs) and compared. Number of neurons, transfer function ('logsig', 'tansig') and learning algorithm ('trainbfg', 'trainbr', 'trainlm', 'trainscg') were selected. According to Figure 1.3, the running time increased as the number of neurons increased. This trend is more obvious for Bayesian Regularization algorithm and LM algorithm while for Quasi-Newton algorithm and conjugate gradient algorithm, it increased more slowly. Bayesian Regularization algorithm consumed quite a long time; LM algorithm ranked second place and Quasi-Newton algorithm was third; conjugate gradient algorithm used the least amount of time. According to Figure 1.4, the MSEs on validation set decreased as the number of neurons increased. Conjugate gradient algorithm performed the worst and Quasi-Newton algorithm ranked second place; Bayesian Regularization algorithm and LM algorithm resulted in similar (Bayesian Regularization algorithm slightly better) and smaller MSE values. LM algorithm was selected because it provided small validation MSE without much tradeoff in terms of time. No significant difference of different transfer function was observed, tansig was slightly better in terms of MSE. For both Bayesian Regularization algorithm and LM algorithm, when number of neurons was larger than 20, the validation MSE would not decrease any more as the number of neurons increased. Networks with 2 hidden layers were also tested and no performance improvement was observed. In conclusion, one layer neuron network with 20 neurons using LM algorithm and tansig transfer function was

chosen as final model.

Next, the performance of the final model was assessed on the test set. The surface of the test set and the approximation simulated by the model quite overlapped, indicating good generalizability of the model. The MSE on the test set was further checked. As shown in Figure 1.2, MSEs on training, validation and test set decreased as number of epochs increased and after certain iterations the MSEs stayed at quite small value. The performance on training set always a little bit better than that of test set. But it is okay since the difference is not significant. By increasing the number of neurons or the maximum amount of epochs or switching to Bayesian Regularization algorithm, the MSE could be further reduced. However, the final model already could provide good accuracy (MSE=1.9347e-06). The mentioned methods would not improve the performance much, yet increase computational complexity.

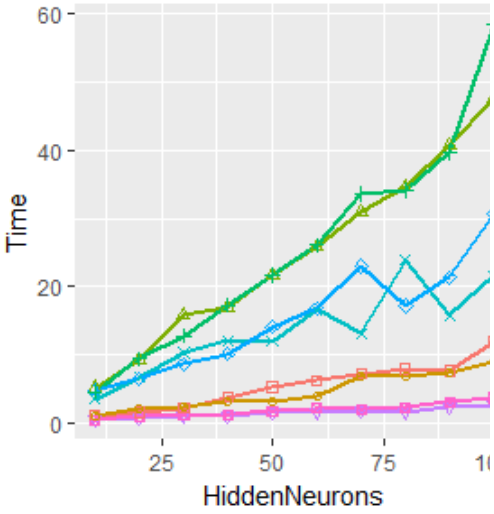

Figure 1.3: Time vs HiddenNeurons

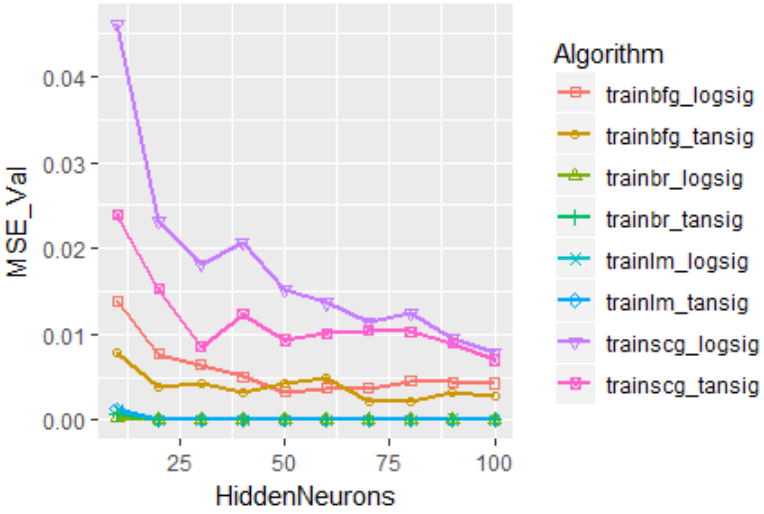

Figure 1.4: MSE vs HiddenNeurons

1.2 Classification

In this session, the feedforward neuron network is applied to classify the red wine quality between class 5 (positive) and class 6 (negative). Correct Classification Ratio (defined as CCR=Number of Correctly classified data x 100/ Total number of data) was used as evaluation criterion.

To search for the best model, the CCRs on validation set and time complexity of different networks were computed (averaging 10 runs) and compared. According to Figure 1.6, the running time increased as the number of neurons increased. This trend is more obvious for Quasi-Newton algorithm and LM algorithm while for conjugate gradient algorithm, it increased more slowly. Quasi-Newton algorithm consumed quite a long time; LM algorithm ranked second place; conjugate gradient algorithm used the least amount of time. According to Figure 1.8, for all algorithms, there did not exist any clear pattern: CCR just randomly fluctuated around 71%. The choice of training algorithm and the number of neurons did not significantly affect the value of CCR. The choice of transfer function did not significantly affect CCR, either. LM algorithm performed slightly better than average and reached highest CCR when the number of neurons equaled 20. In Figure 1.6, using tansig resulted in sightly less time. Networks with more than 2 hidden layers were also tested and no performance

improvement was observed. In conclusion, one layer neuron network with 20 neurons using LM algorithm and tansig transfer function was chosen as final model.

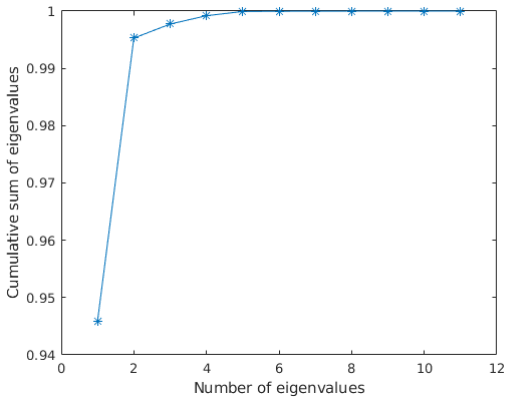

Figure 1.5: Cumsum of eigenvalues

Next, we preprocessed the data first by applying PCA to investigate the benefits of dimensionality reduction. Figure 1.5 plots the normalized cumulative sum of eigenvalues against number of eigenvalues. 99.92% of the variance can be explained by the first 4 principal components. Using the reconstructed dataset, we repeated the model comparison procedure again. Since we threw away some information during PCA, the accuracy was lower than before as expected. In Figure 1.9, for all algorithms, CCR randomly fluctuated around 69%. The choice of training algorithm and the number of neurons did not significantly affect the value of CCR. The choice of transfer function did not significantly affect CCR, either. LM algorithm performed slightly better than average and reached highest CCR when the number of neurons equaled 15. For dataset after PCA, one layer neuron network with 15 neurons using LM algorithm and tansig transfer function was chosen as final model. In Figure 1.7, the running decreased to about half of the time used before and the patterns remained the same. It can be concluded that by applying dimensionality reduction techniques, we could save some time without much tradeoff in terms of accuracy.

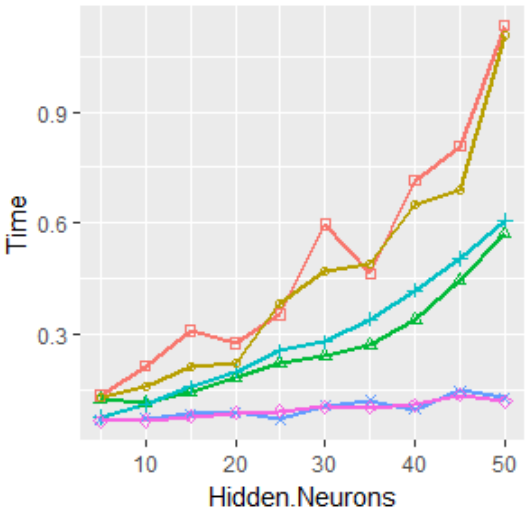

Figure 1.6: Time vs HiddenNeurons

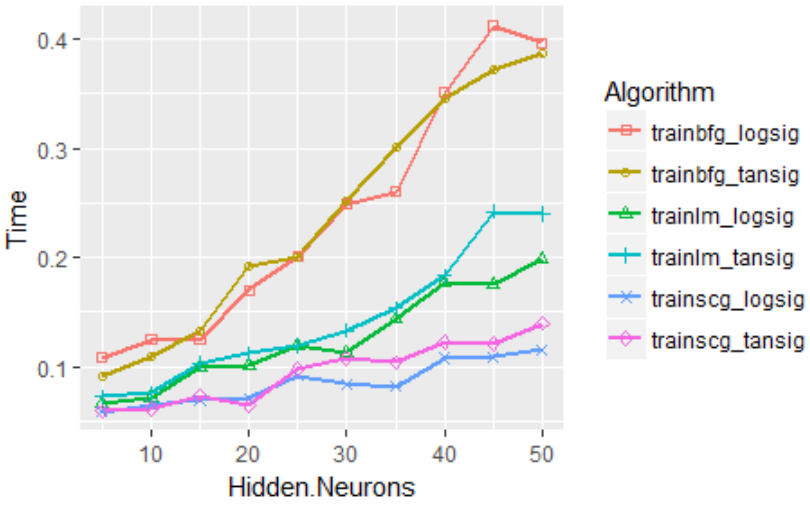

Figure 1.7: Time vs HiddenNeurons(with PCA)

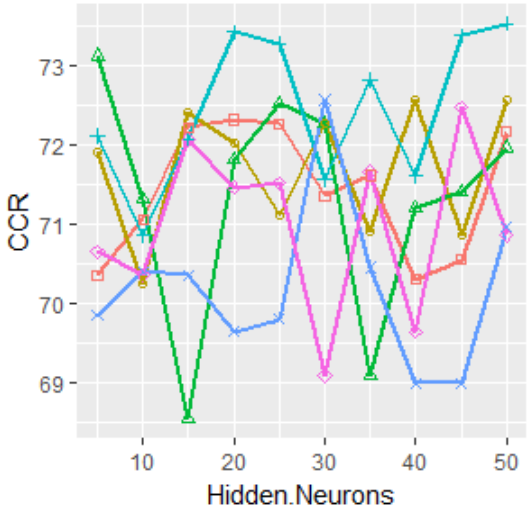

Figure 1.8: CCR vs HiddenNeurons

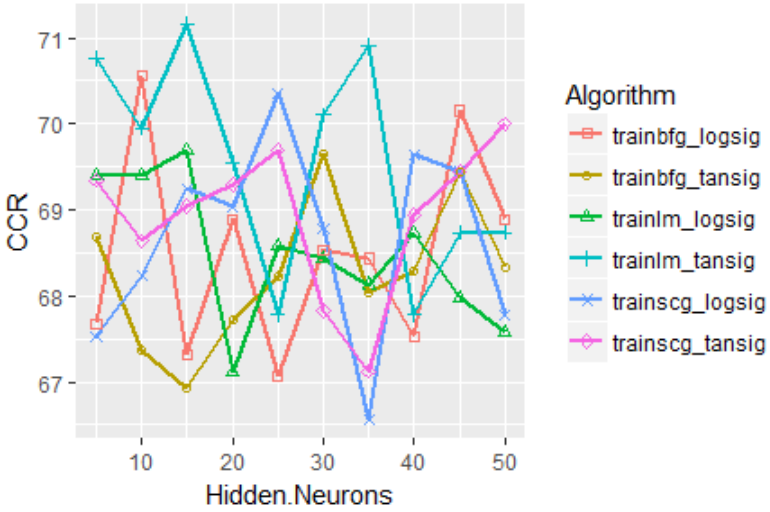

Figure 1.9: CCR vs HiddenNeurons (with PCA)

2 Character recognition with Hopfield networks

In this session, the retrieval capacity of Hopfield network is investigated based on character recognition problem. The individualized collection of characters is *mengyuaABCDEFG...etc.* In total, we have 33 7x5 binary images. In Figure 2.1, the first row shows the first 10 letters of the generated sequence.

We started with a simple example by storing the first 5 characters as attractors and training the Hopfield network, in order to recall distorted version of them. The distorted letters were generated by inverting 3 randomly chosen pixels. In Figure 2.1, from second to fourth row, it shows the original, distorted and recalled first 5 characters. The Hopfield network was capable of retrieving them perfectly. The existence of spurious patterns was not observed because of the small number of attractors.

Figure 2.1: Original, distorted, recalled letters

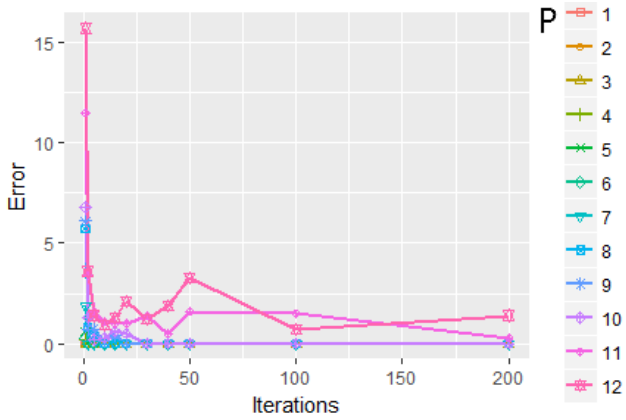

Figure 2.2: Error vs Iterations

With the increase of the number of stored characters (P), the number of spurious patterns increases and the distorted characters tend to converge to these spurious states. Critical Loading Capacity is the maximal number of characters (P_{max}) that can be stored such that the network is able to recall them without error. The error is defined as the percentage of wrongly retrieved characters. For large values of P and N(number of neurons), the theoret-

ical loading capacity of a Hopfield network can be estimated using $P_{max} \approx 0.138N$. In our case, N=7x5=35, so $P_{max} \approx 5$.

To determine the loading capacity in our case, we varied the value of P (1 to 33) and calculated the corresponding errors. For each value of P, the number of iterations (up to 200) were also varied. Each individual test was repeated for 10 times and the mean error value was used for comparison. Figure 2.3 plots the error against P for varying iteration numbers. With a single iteration, the network was already able to recall 7 letters perfectly. As the iteration number increased, the error decreased for same value of P. The curves of several high iteration numbers overlapped, indicating the network converged. For enough number of iterations, after $P = 10$, the error increased above 0, meaning loading capacity is 10. This can also be confirmed as shown in Figure 2.2. For P value smaller than and equal to 10, the error decreased to 0 after about 40 iterations. While for P value larger than 10, the error stayed at some low value after convergence. The result differed from the theoretical value since the formula should be used only for large P and N.

Increasing the number of neurons can increase the loading capacity of the Hopfield network. The number of neurons is determined by the dimensionality of the input data. Exapanding the number of pixels in each character actually means generating higher resolution image. Each character was zoomed 2 times (using repelem()), resulting in 14x10 images and the number of neurons equal 140. Figure 2.4 plots the error as a function of P with enlarged images. Only after 15 iterations, all 33 characters can be perfectly recalled.

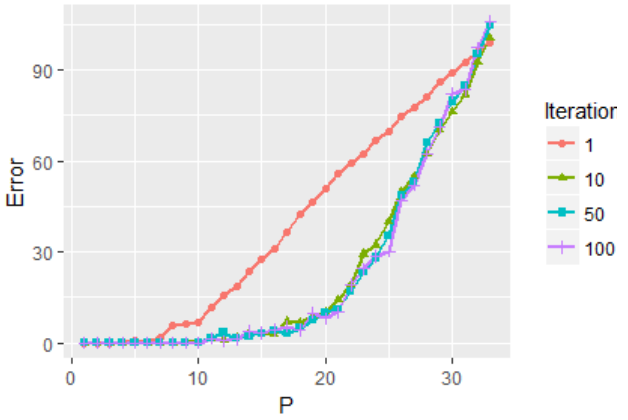

Figure 2.3: Error vs P

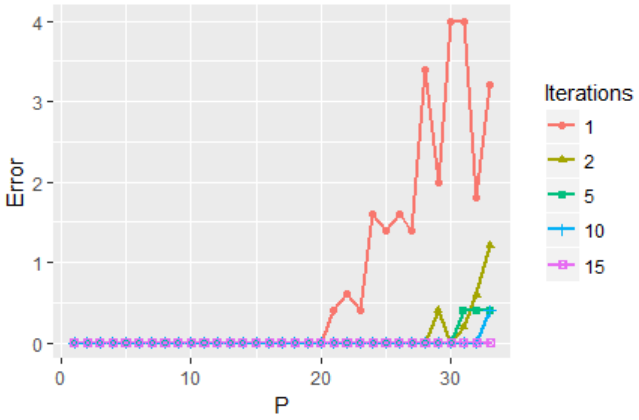

Figure 2.4: Error vs P (higher resolution)