EE5904/ME5404 Neural Network Assignment2

Wang Jiangyi
National University of Singapore

1 Question 1: GD and Newton's Method

Consider the Rosenbrock's Valley function:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

Before giving our solution, let's check the figure of this function first:

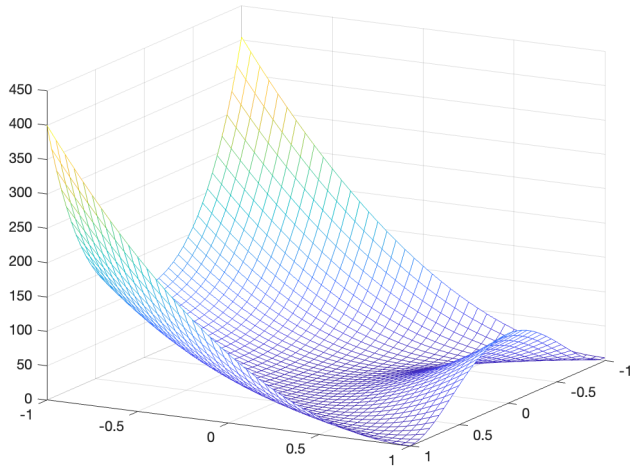

Figure 1: Shape of Rosenbrock's Valley function

From its figure, we can find that, the Rosenbrock's Valley function is non-convex. Therefore, we cannot easily judge the global minimum by calculating the gradient.

1.1 Question a) Global minima

Proof:

Firstly, we must have:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2 \geq 0 \quad \forall (x, y) \in \mathbb{R}^2$$

Since $f(1, 1) = 0$, it must be one of the global minima of Rosenbrock's Valley function.

Then we want to show the uniqueness of $(x, y) = (1, 1)$, i.e.,

$$f(1, 1) < f(x, y) \quad \forall (x, y) \in \mathbb{R}^2 \; and \; (x, y) \neq (1, 1)$$

Let's show this as follows:

$$\begin{aligned} f(x, y) = 0 &\Longleftrightarrow x = 1 \; and \; y = x^2 \\ &\Longleftrightarrow (x, y) = (1, 1) \end{aligned}$$

This implies Rosenbrock's Valley function only has one global minima at $(x, y) = (1, 1)$ where $f(x, y) = 0$.

1.2 Question b) Gradient Descent

Here, we actually give 2 different ways of implementing Steepest Descent Method.:
One is with **fixed step length** (learning rate), which is required in question, and the other is with **adaptive step length** (learning rate).

1.2.1 Fixed learning rate

When learning rate $\eta = 0.001$, after **14001 iterations**, point (x, y) converges to $(0.9992, 0.9988)$, which is very close to the global minima $(x, y) = (1, 1)$. The trajectory of (x, y) is shown as follows:

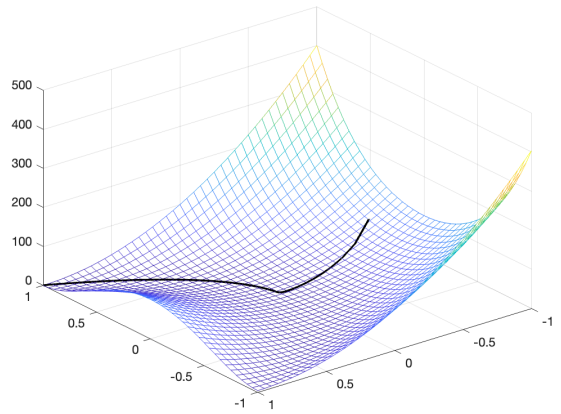

Figure 2: 3-D trajacotry of Steepest Descent

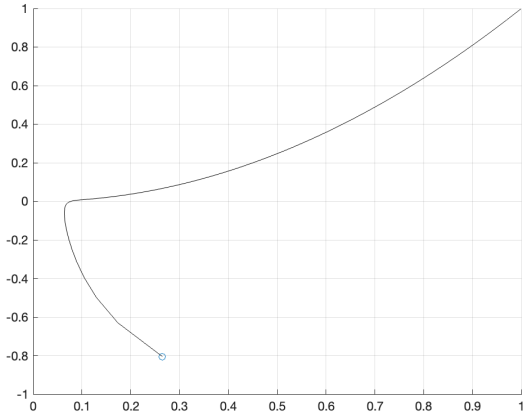

Figure 3: 2-D trajacotry of Steepest Descent

When learning rate $\eta = 1$, after **1 iteration**, the function value is over 10^9, which shows that the algorithm will diverge in this case.

1.2.2 Adaptive learning rate

Here, instead of using fixed learning rate, we search for the appropriate learning rate (step length) for each iteration. Here, we only use **1943 iterations** to achieve $(x, y) = (0.9996, 0.9991)$. The trajectory of (x, y) is shown as follows:

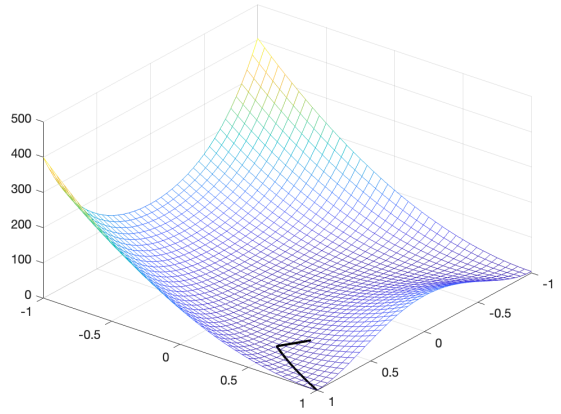

Figure 4: 3-D trajacotry of Steepest Descent

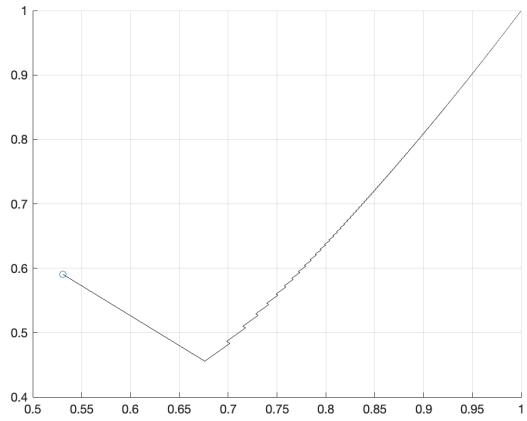

Figure 5: 2-D trajacotry of Steepest Descent

1.3 Question c) Newton's Method

As for Newton's Method, we can only use **5 iterations** to get very close to the global minimum($(x, y) = (1, 1)$), although the initial point is very far away from the minimum. The trajectory of (x, y) is shown as follows:

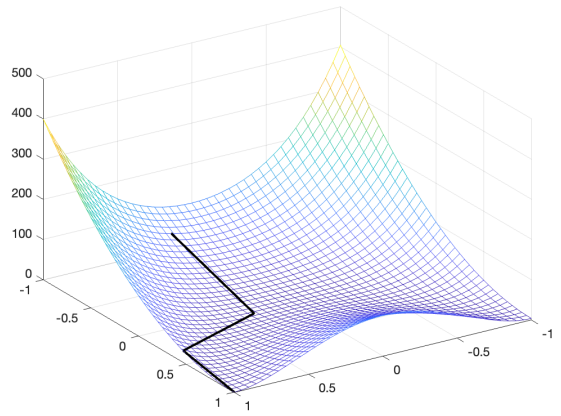

Figure 6: 3-D trajacotry of Newton's Method

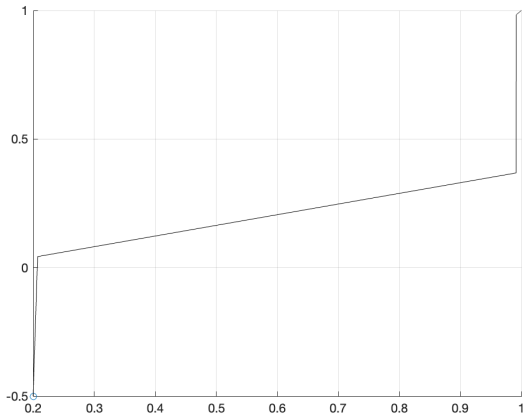

Figure 7: 2-D trajacotry of Newton's Method

The **story** is that, since the Rosenbrock's Valley function is very similar to convex quadratic function locally, the Newton's Method works very well. This is because Newton's Method is just quadratic approximation of original function, and then find the best direction according to the approximation. The more similar to the convex quadratic function locally, the more power Newton's Method will have.

Therefore, as for this problem, Hessian matrix can guide the point to the global minimum only in few steps.

2 Question 2: Function Approximation

Consider the following function:

$$y = 1.2sin(\pi x) - cos(2.4\pi x) \quad for\ x \in [-1.6, 1.6]$$

The figure of the function is: Before doing the numerical experiment, we just use the guideline in lecutre.

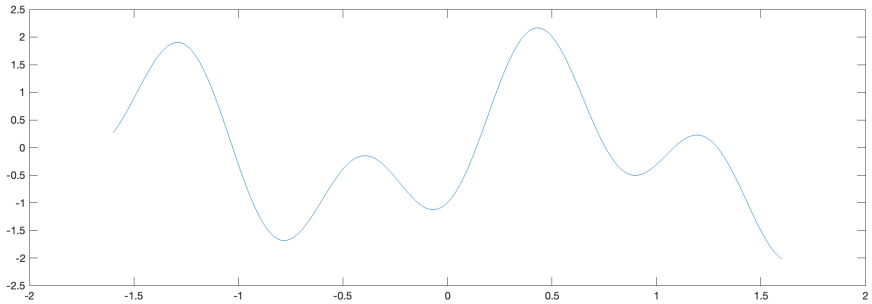

Figure 8: Shape of the function

Since there are **8 line segments** in this figure, we guess appropriate number of hidden neurons is around **8**. Then we train the neural network to check the guideline in lecuture.

2.1 Question a) Sequential Mode MLP

2.1.1 Inside the interval $[-1.6, 1.6]$

In the following discussion, we focus on the MLP: 1-n-1 (where n = 1, 2, ..., 10, 20, 50, 100). In the training procedure, we apply the parameters that `net.trainFcn = 'traingd'` and `net.performParam.regularization = 0`. The maximum number of training epochss is 200. Here is the result of training:

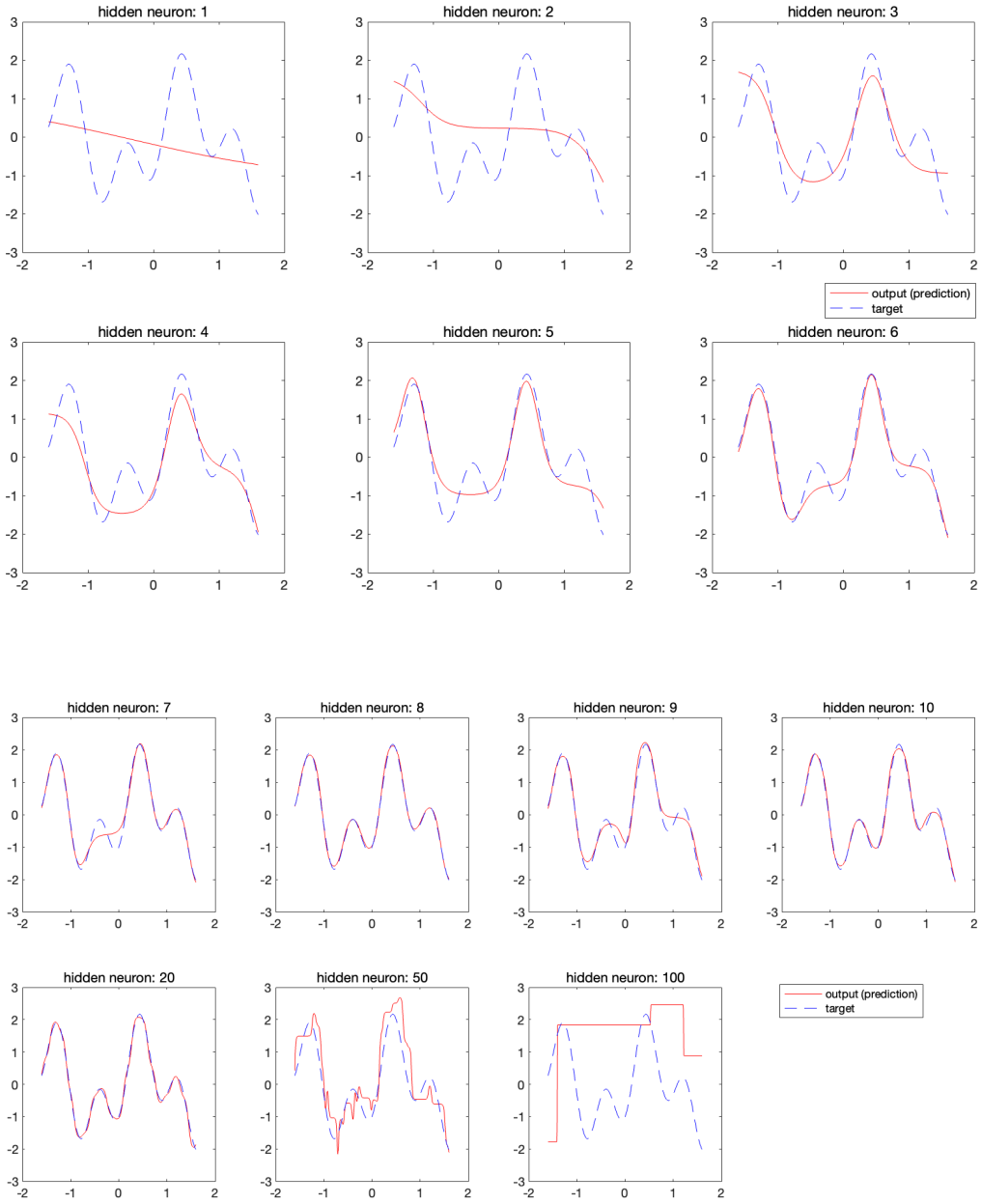

Figure 9: fitting results of different neurons, epoch=200

From Figure 9, it can be easily seen that when number of neurons = 50 and 100, the shape of fitting curve is quite different from the true function. Even when we increase the epoch to 1000, we cannot get good fitting results in these two cases.

In order to determine whether it is proper fitting, we still need to check the **training error and testing**

error of different models. Since our problem is regression problem, we apply **Mean Square Error** here to measure the performance of the model. The result is shown in Figure 10:

Figure 10: MSE of different neurons

Combine Figure 9 and 10 together, we can make the following statements:

1. When the number of hidden neurons is 1, 2, 3, 4, 5, 6, 7, 9, 50, 100, it comes from **under-fitting**. This is because the shape their curves is different from the true function. Moreover, their performance measure, MSE, is all larger than 0.05.

2. When the number of hidden neurons is 8, 10, it **fits properly**. This is obvious because the shape of curve is almost the same as the true function, and the MSE (for both training set and testing set) of model is very close to 0, which is 0.002 and 0.004.

3. When the number of hidden neurons is 20, I think it has the tendency to be **over-fitting** (not totally over-fitting). Although the MSE of this model is still close to 0, for both training set and testing set. However, if we check its shape carefully, we can find this fitting curve **looks not so smooth** and has the tendency to **deviate** from the true function from time to time.

Besides that, I also find that, the parity of the number of hidden neurons also matters.

Therefore, as our discussion above, the minimal number of hidden neurons from the experiments is **8**, which agrees to the guideline in lecture.

2.1.2 Outside the interval

Let's calculate the function value (of our MLP) and compare with the true one:

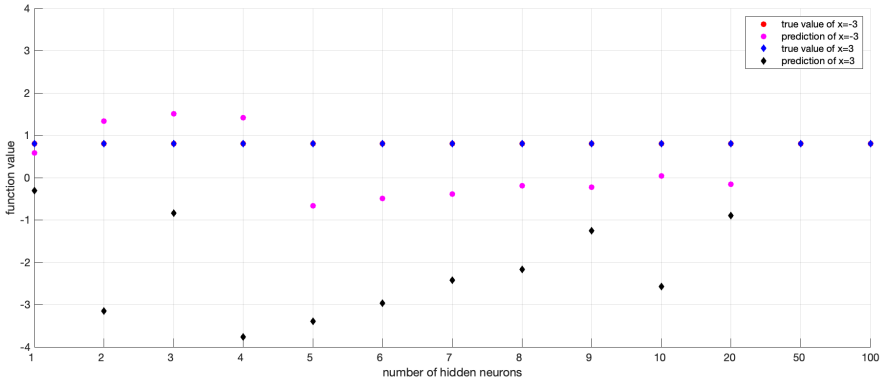

Figure 11: $x = +3$ and $x = -3$, Sequential Mode MLP

It is obvious that our MLP **cannot** give reasonable prediction outside the interval. It is predictable because

MLP is just function approximation. If we do not feed it the correct target, i.e., the function value at $x = 3$ or $x = -3$, then it will learn nothing about them.

2.2 Question b) Batch Mode MLP with 'trainlm'

2.2.1 Inside the interval $[-1.6, 1.6]$

Here, we apply the parameters that `net.trainFcn = 'trainlm'` and `net.performParam.regularization = 0`. The maximum number of training epochss is 200. Here is the result of training:

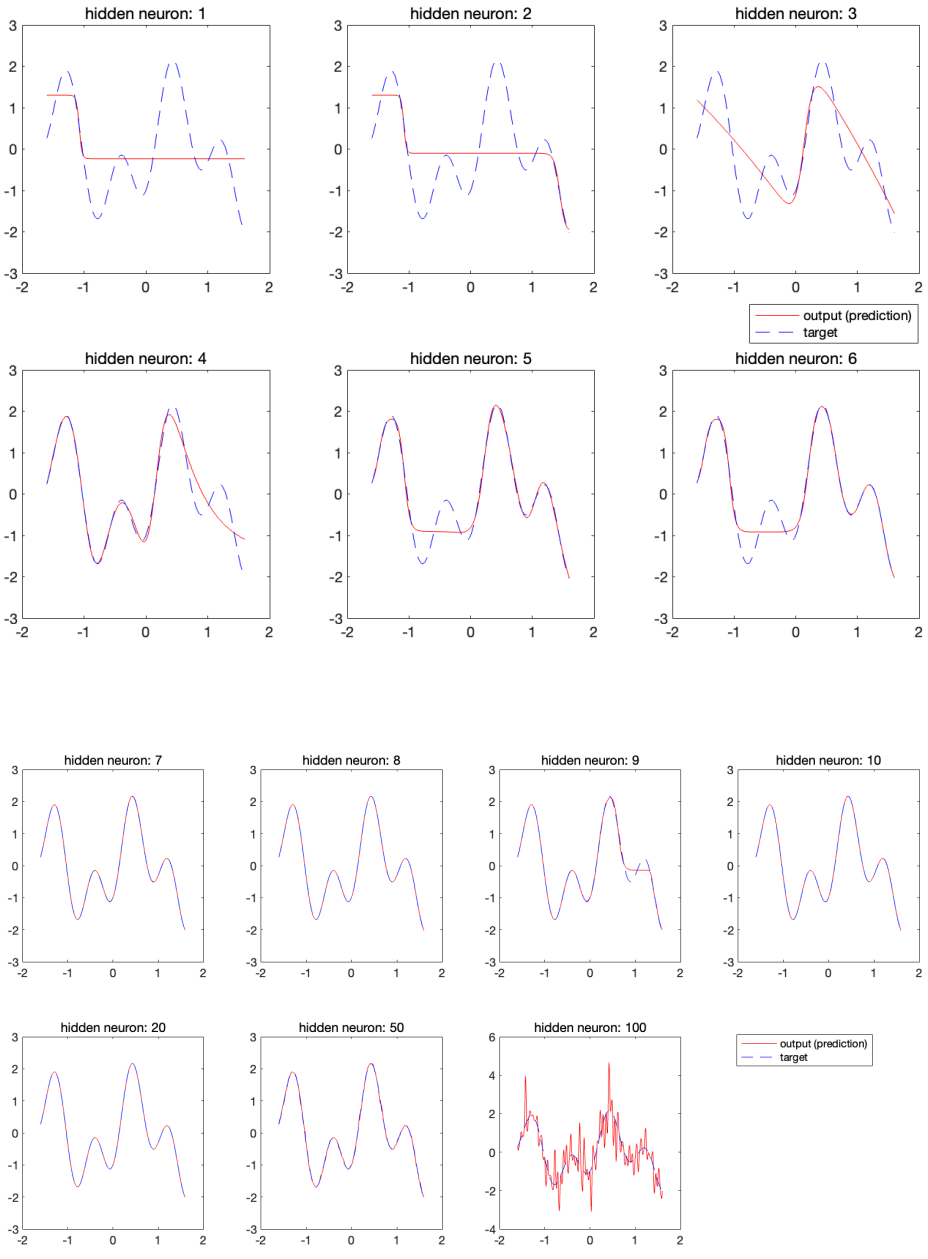

Figure 12: fitting results of different neurons, epoch=200

It is worthwhile to mention that, using Batch Mode with `net.trainFcn = 'trainlm'`, the learning algorithm will quickly terminates as the gradient will quickly converge to 0.

The performace of the model can be seen in Figure 13:

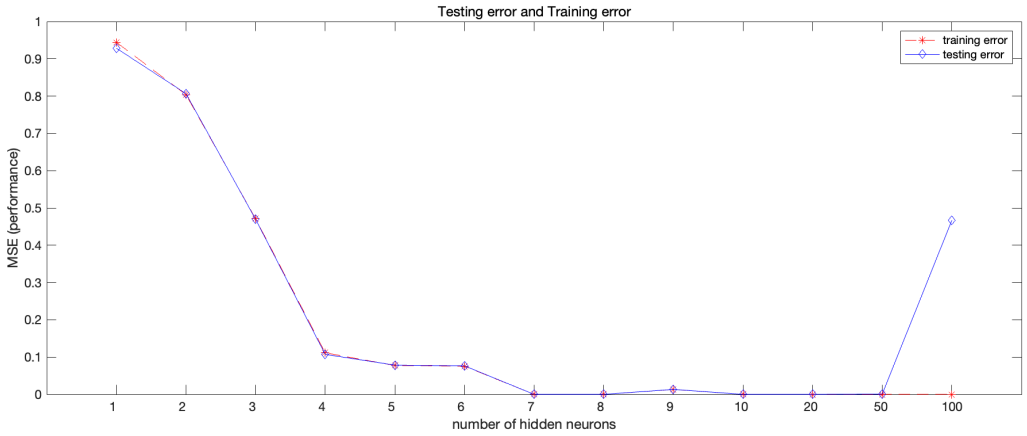

Figure 13: MSE of different neurons

Combine Figure 12 and 13 together, we can make the following statements:

1. When the number of hidden neurons is 1, 2, 3, 4, 5, 6, 9, it comes from **under-fitting**. This is because the shape their curves is different from the true function. Moreover, their performance measure, MSE, is all larger than 0.1.

2. When the number of hidden neurons is 7, 8, 10, 20, 50, it **fits properly**. This is obvious because the shape of curve is almost the same as the true function, and the MSE (for both training set and testing set) of model is very close to 0.

3. When the number of hidden neurons is 100, it suffers from **strongly over-fitting**. It has almost 0 training error, while its testing erro is dramatically high. Moreover, the shape of its fitting curve is very sharp.

Therefore, as our discussion above, the minimal number of hidden neurons from the experiments is **7**, which is close to the guideline in lecture.

2.2.2 Outside the interval

Let's calculate the function value (of our MLP) and compare with the true one:

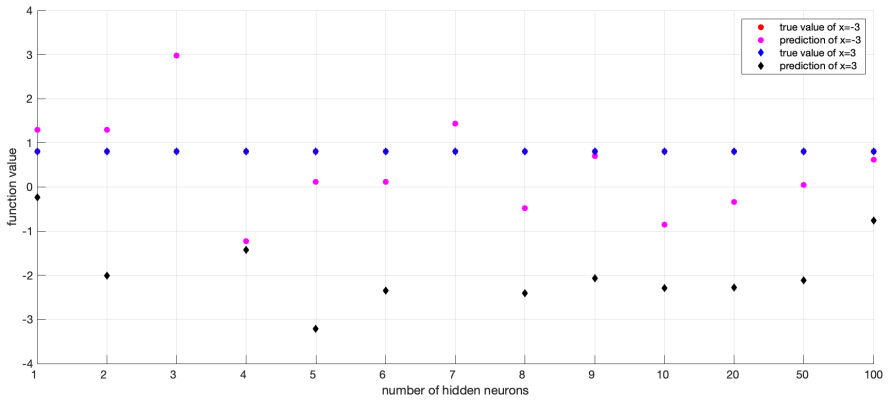

Figure 14: $x = +3$ and $x = -3$, Sequential Mode MLP

It is obvious that our MLP **cannot** give reasonable prediction outside the interval, and the reason is the same as that in Question a).

2.3 Question c) Batch Mode MLP with 'trainbr'

2.3.1 Inside the interval $[-1.6, 1.6]$

Here, we apply the parameters that `net.trainFcn = 'trainbr'` and `net.performParam.regularization = 0`. The maximum number of training epochss is 200. Here is the result of training:

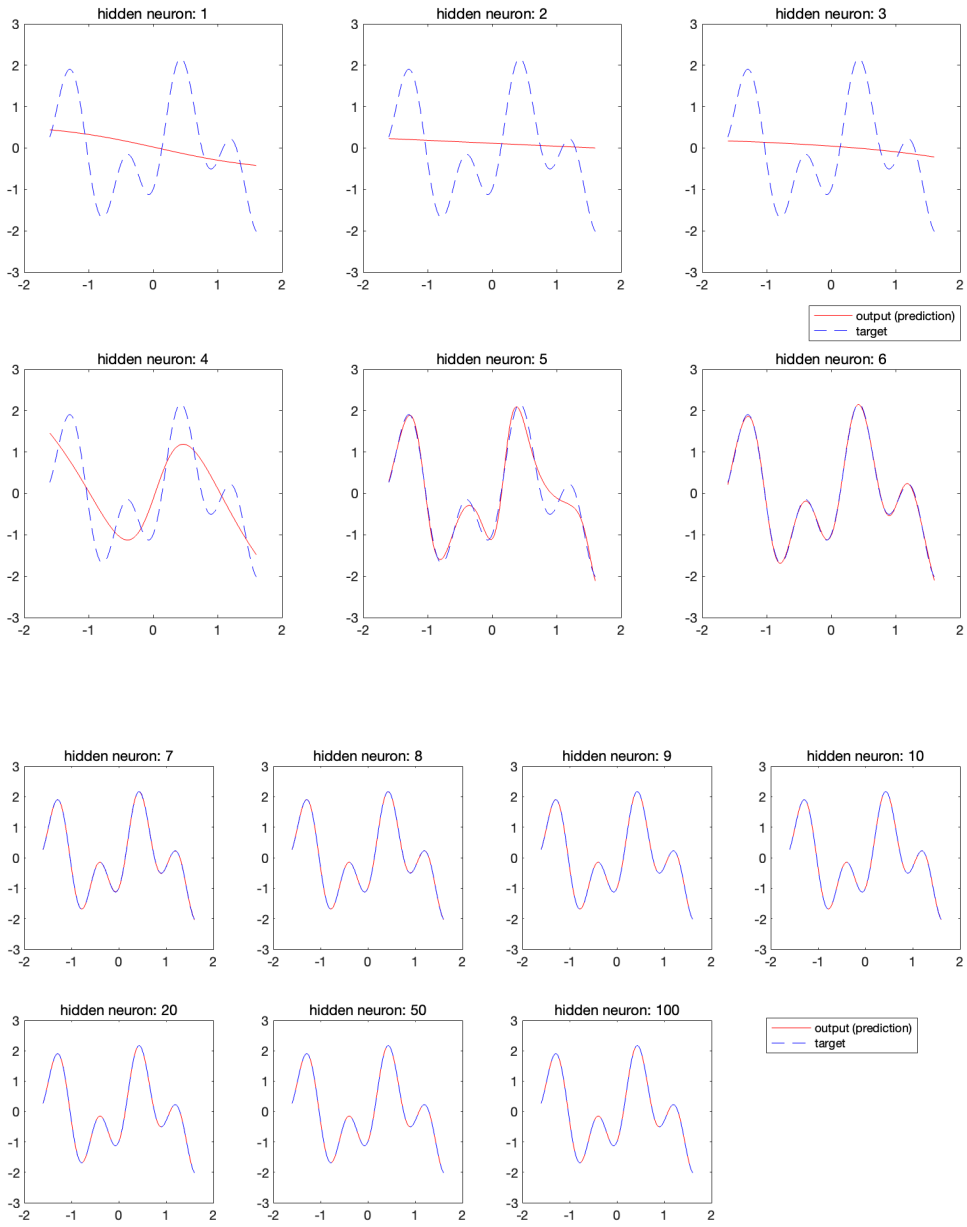

Figure 15: fitting results of different neurons, epoch=200

Different from the Batch Mode with `net.trainFcn = 'trainlm'`, the network with `net.trainFcn = 'trainbr'` **behaves quite well** on both training set and testing set when there are enough hidden neurons.

With this training method, we notice that for each number of hidden neurons, the training process will iterate for 200 epochs **without early stop**. This might be why the network with 100 hidden neurons also generalize well (which generalize very bad with `net.trainFcn = 'trainlm'`).

The performace of the model can be seen in Figure 16:

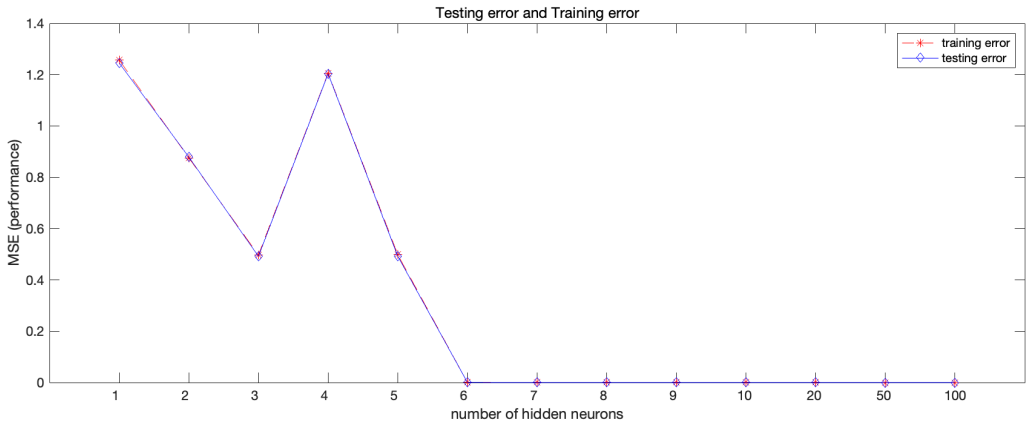

Figure 16: MSE of different neurons

Combine Figure 15 and 16 together, we can make the following statements:

1. When the number of hidden neurons is 1, 2, 3, 4, 5, it comes from **under-fitting**. This is because the shape their curves is different from the true function. Moreover, their performance measure, MSE, is all larger than 0.2.

2. When the number of hidden neurons is 6, 7, 8, 9, 10, 20, 50, 100, it **fits properly**. This is obvious because the shape of curve is almost the same as the true function, and the MSE (for both training set and testing set) of model is almost 0.

3. **Over-fitting will not happen** in these MLP.

Therefore, as our discussion above, the minimal number of hidden neurons from the experiments is **6**, which is also close to the guideline in lecture.

2.3.2 Outside the interval

Let's calculate the function value (of our MLP) and compare with the true one:

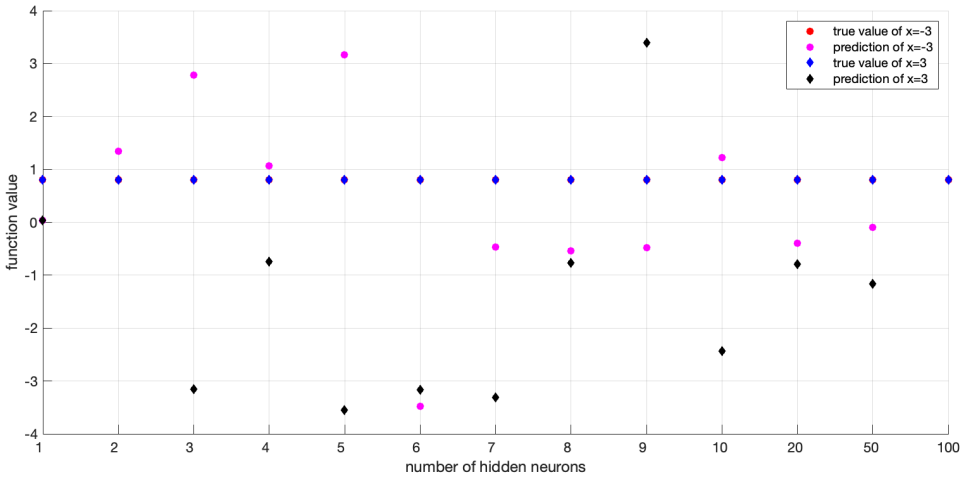

Figure 17: $x = +3$ and $x = -3$, Sequential Mode MLP

It is obvious that our MLP **cannot** give reasonable prediction outside the interval, and the reason is the same as that in Question a).

3 Question 3: Image Classification

Firstly, my student ID is **A0236307J**, which means I should use **Group 1** dataset (dog vs. automobile) to implement this experiment.

Then, before giving the solution to each question, I make some description to the data:

1. Training set contains 900 samples, and the first 450 samples are in Class 0 (manmade objects);
2. Validation set (testing set) contains 100 samples, and the first 50 samples are in Class 0 (manmade objects);
3. Each sample (image) is 1024×1 column vector, which means our MLP should be 1024-n-1 structure.

Moreover, for the whole question, we introduce some performance meausre for classification problem. In our following experiments, we will use *Confusion Matrix* to evalute our models. For binary classification problem, Confusion Matrix is a 2×2 matrix defined as below:

		Actual Values	
		Positive(1)	Negative(0)
Predict Values	Positive[1]	TP	FP
	Negative[0]	FN	TN

Table 1: Confusion Matrix

In Table 1, 'TP' and 'TN' represents those testing points which are classified correctly in positive and nagative class repectively. Similarly, 'FP' and 'FN' represents those which are classified incorrectly in respective classes.

Here, we mainly focus on three quantity, which is **Recall, Precision and Accuracy** and defined as follows:

$$\begin{aligned} Recall\ rate &= \frac{TP}{TP+FP} \\ Precision\ rate &= \frac{TP}{TP+FN} \\ Acccuracy\ &= \frac{TP+TN}{TP+FP+FN+TN} \end{aligned}$$

3.1 Question a) Perceptron

Here, we apply Perceptron algorithm to our dataset. Actually, we firstly consider two cases, one is the original image (whose range is between 0 and 255 in each dimension) and the other is the normalized image (whose range is between 0 and 1 in each dimension). Notice that, to **distinguish from Question b)**, here we actually simply do as follows:

$$\hat{x}(i) = \frac{x(i)}{255} \quad i = 1, 2, ..., 1000 \ and \ x(i) \in \mathbb{R}^{1024\times 1}$$

We want to use these two datasets to train Perceptron and see what will happen. We use all images in training set to train our perceptron sequentially, and we set the number of epoch equal to 100.

The training results for two datasets can be seen in Figure 18 and Figure 19:

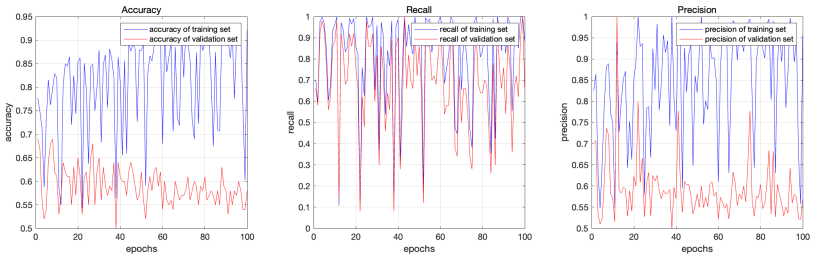

Figure 18: Performance of image whose range is between 0 and 255

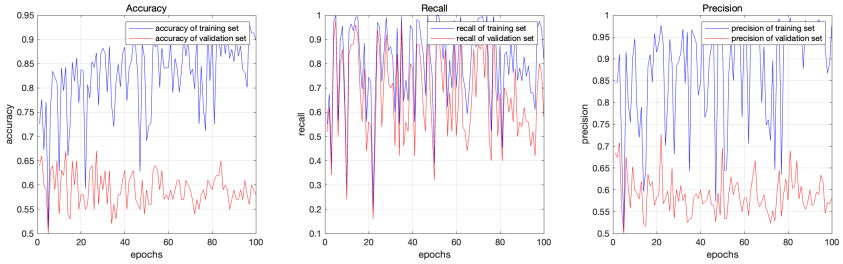

Figure 19: Performance of image whose range is between 0 and 1

From Figure 18 and Figure 19, we can observe that, after 100 epochs, the **classification accuracy** for training set is around **0.92**, and that for validation set is only about **0.6**.

Additionally, we are able to attain further observation as follows:

1. It seems it will **make no help** for the training results after we transform the range of each dimension to $[0, 1]$, simply by dividing 255. We may conclude that, at least for sequential learning in perceptron, enlarge (or shrink) the data according to **the same scale** may make **few** help for the whole training process. What we actually need to do is that, for some features whose value is quite small on average, **we should enlarge them so that they can be easily noticed by our model**. This can be viewed as the **motivation** of normalization.
2. As the iterations (update) go on, although the accuracy of training set fluctuates greatly, it still has a trend of increasing. However, the accuracy of validation set always fluctuates around 0.6. This implies our model does not learn the images in validation set, although images in validation and training set shares the same meaning of patterns (dog and manmade objects). Therefore, the **generalization of our model is poor**, which can be viewed as evidence of over-fitting.

Moreover, after introducing the recall rate and precision rate, we can analyze the performance in a more precise perspective. It can be seen from both 2 figures that the **recall rate** for both training set and validation set are similarly high, while the **precision rate** are very different. This actually shows something and we mainly focus on the validation set. Since the recall rate is realtively high and precision rate is low, we can deduce that, our model focus more on the positive class (dog). That is to say, our model **has a tendency** to give positive-class (dog) prediction, which will lead to high recall rate and low precision rate. This is also can be viewed as evidence of over-fitting.

To conclude, there is no doubt that this simplest network is **over-fitting**, and the performance of this perceptron **fluctuates dramatically**.

3.2 Question b) Normalization

Here, it is suggested that we should use normalization on the whole dataset (normalize on each feature). As mentioned in email, here we only use training set to calculate the normalize parameters.

After pre-processing the dataset (normalization), the training result is as Figure 20:

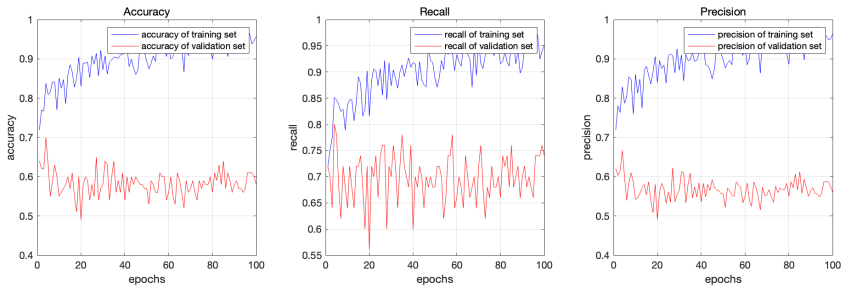

Figure 20: Performance of normalized dataset

Compared with Figure 18 and Figure 19, it can be easily observed that the performance of normalized dataset (both training and validation set) is much more steady. From Figure 20, it also shows that the accuracy of

training set keeps on increasing, while that of validation set seems to go down and fluctuate.

To conclude, normalization may have the benefit that **makes the training process and final performance stabler**. However, it still has the problem that, the accuracy of validation set will keep on fluctuating (or decreasing) while that of training set is still increasing.

Next, I would like to give **2 versions of explanation**:

1. This is a more intuitive version of explanation. Let's imagine the case without normalization. According to the procedure of Perceptron algorithm, the update of weights is **proportional to the input value**. That is to say, if we do not normalize our dataset, some features may have large value, while some features may have small value. This will lead to the update of the weights will be **dominated** by some important features. We can view this kind of update as **sharp update**, which is likely to cause great changes in prediction.

However, if we do normalization, then the scale of each feature can be similar. We can understand like this, we imagine that we have a prior knowledge that each feature is equally important. Then, the update of weights is much **smoother**. This will tend to give slight changes in prediction.

2. This is a more rigorous version of explanation. Actually, for Batch-Mode Perceptron algorithm, we can still translate it as an optimization problem (whose objective is differentiable according to w). Then the Batch-Mode Perceptron Learning algorithm is equivalent to the Steepest Descent Algorithm with fix learning rate (step length).

Proposition: We can use the knowledge of Optimization Theory that, Steepest Descent works well (perfectly) if the function is isotropic (whose contour map is circle in 2-dimensional case). If the semi-major axis is far from semi-minor axis for the contour map, then we say this function is **ill-conditioning**. In some gradient directions, **even slight changes of variable value will lead to dramatic changes of function value**. If we apply exact line search method, '**Zigzag**' will happen, let alone we just use fixed step length.

This gives the motivation of this explanation. When we do without normalization, due to the difference of scales in each feature, the optimization objective function is likely to be greatly ill-conditioning. If we apply Steepest Descent Method with fix step length, it can be imagined that our prediction can be greatly changed within different iterations. If we do with normalization, our problem will tend to be more well-conditioning. This implies that our final result will be stabler.

Furthermore, from this experiment, it seems true that the **accuracy of our perceptron will be the best within first 10 epochs**. After that, the accuracy will tend to go down and fluctuate. To confirm myself, I do some more experiments, which are shown as follows:

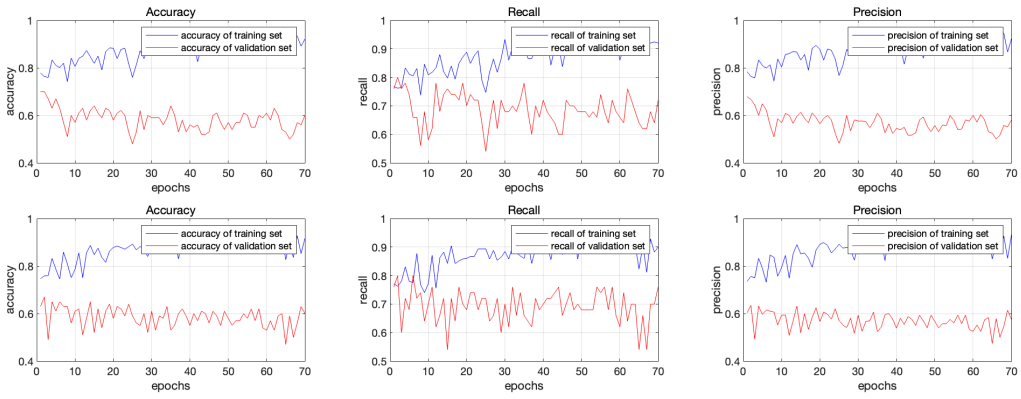

Figure 21: More experiments on normalized dataset

In Figure 21, we can see the accuracy curve will achieve its maxima value in the very early epoch. In the later epochs, the accuracy tend to go down or fluctuate. Therefore, it seems reasonable that we should end our training process within 10 epochs, which may be of help for us to achieve better generalization result.

Lastly, it is worthwhile to mention that, although it seems 10 epoch is quite small number, it actually represents nearly 10000 times of update on weights. Thus, it makes sense that we end our training process in 10 epochs.

3.3 Question c) MLP without regularization

For this question, our candidate MLP is 1-n-1, where $n = [5:5:100,\ 110:10:400,\ 450,\ 500,\ 600,\ 700,\ 800]$. We apply the parameters that `net.performParam.regularization = 0`, `net.trainFcn = 'trainrp'` and `net.divideFcn = 'dividetrain'`. Then we use Batch-Mode to train our network for 150 epochs. In this question, we do not want the whole training process to stop early. Therefore, we use all the training set to avoid early stop. Here, the output activation function is `logsig`. In order to achieve fast convergence, we **introduce target value 0.2 (0.8)** for negative (positive) class.

The results of performance for different number of neurons are as follows:

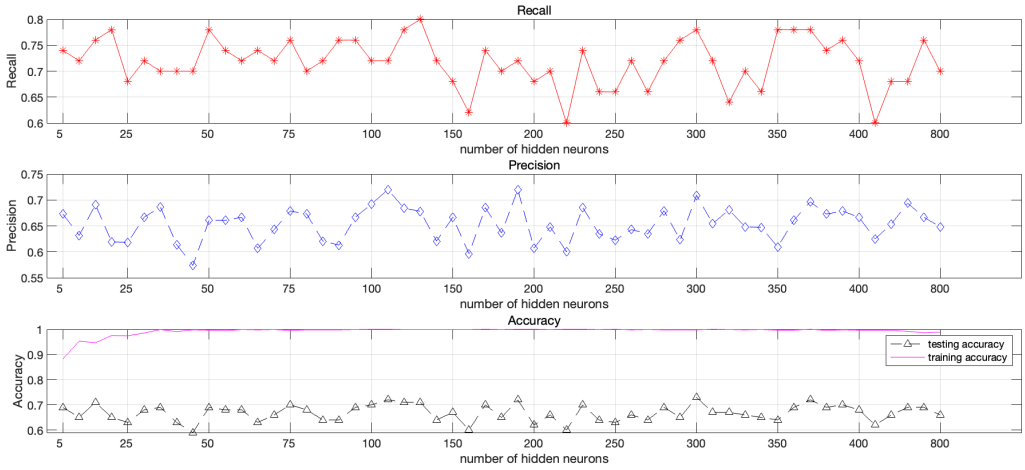

Figure 22: Performance on different number of neurons

It can be seen from Figure 22 that the **classification accuracy** for training set is **almost 1** when the number of hidden neurons is greater than 25. Even when small number of neurons, we can also attain at least 0.9 training accuracy.

When it comes to classification accuracy for validation set, it turns out that the results are not so good. When the number of hidden neurons equal to 300 and 370, the **highest accuracy** is achieved, which is **0.74**. **On average**, the accuracy for the validation set is around **0.66**.

To evaluate the performace of MLP network (no matter the hidden neurons number) after 150 epochs without regularization, it is obvious that all these models (for different number of hidden neurons) **behave perfect on training sets**. However, it **cannot generalize well on validation set**. Therefore, we can conclude that all these networks are **over-fitting**.

3.4 Question d) Over-fitting Problem

As we mentioned before, all my trained MLP in c) is **over-fitting**. For simplicity, we want to choose one fixed number of hidden neurons to do further experiments. Our goal is to try to solve the over-fitting problem. As the lecture note says, we have two ways:

1. Determine minimal structure of network;
2. Regularization;

This pattern recognition problem is in high-dimension space, which means we are unable to determine the minimal structure of network just by visualization. Therefore, we try to use SVD to achieve the search of minimal structure to solve over-fitting problem.

In order to implement these 2 ways to solve over-fitting problem, we fix hidden neuron number equal to **300** for our further trials first. Main reason is that, it has the greatest generalization behavior in our previous experiment (Question c)). It is likely to be true that this number of hidden neuron is appropriate for this pattern recognition problem. Therefore, for both regularization and minimal-structure searching, it is reasonable to start our trial with this number of hidden neurons, i.e, **300 hidden neurons**.

Firstly, we do regularization; secondly, we do SVD to find minimal structure of network.

3.4.1 Regularization

First, let's determine when training the MLP with 1024-300-1 (no regularization), after which training epoch it becomes over-fitting. Here, we introduce another performance measure, **cross-entropy**, to determine the epoch. Cross-entropy can be viewed as the perfomance measure in the probability perspective, which treats the output of network as the probability of predicting positive class.

Note that we actually split the 900 training images into 3 different sets, training, validation and testing set with 630, 145 and 145 images respectively. The aim is to check the genralization abillity while training process. The result is in Figure 23:

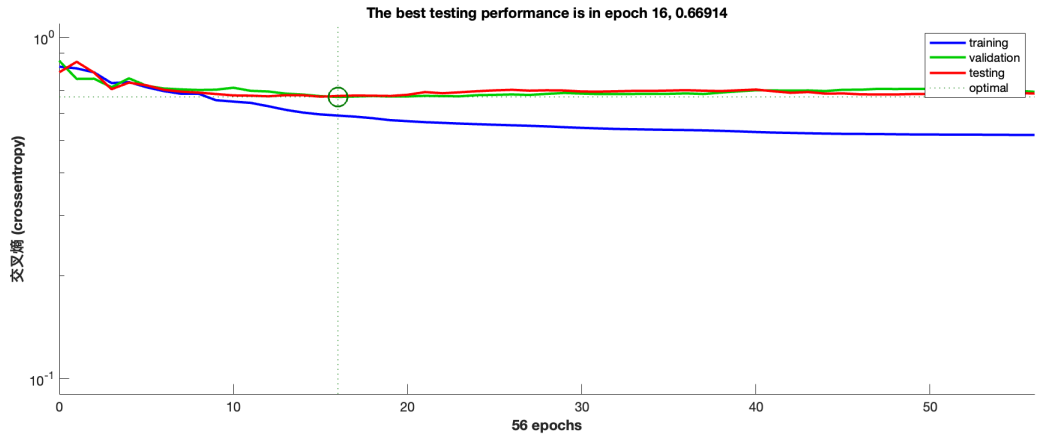

Figure 23: Cross-entropy for 1024-300-1 MLP without regularization

From Figure 23, it can be observed that, in the **first 10 epochs**, testing and validation cross-entropy decreases as training cross-entropy decreases. After that, although training cross-entropy still keeps on decreasing, testing and validation cross-entropy stop decreasing.

Combined with the fact that the perfomance achieved its best in epoch 16, we can deduce that **it becomes over-fitting between 10-th epoch and 16-th epoch.**

Then we will **try regularization** on our MLP. To see whether adding regularization will make some difference to our model, we firstly fix `net.performParam.regularization = 0.3`. The result is shown as Figure 24:

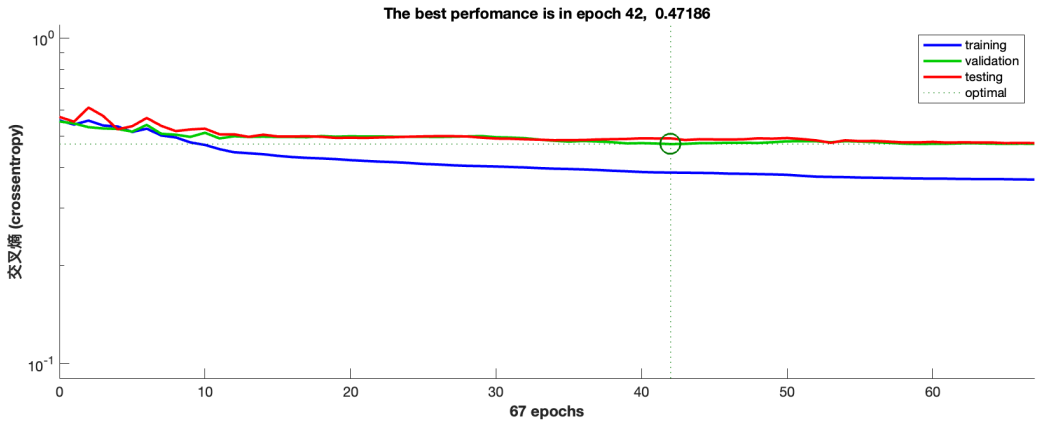

Figure 24: Cross-entropy for 1024-300-1 MLP without regularization

From Figure 24, we can observe that, with the help of regularization, for both validation and testing set, they have more chance to imporve their performance (decrease the cross-entropy). The best performance is in **epoch 42**, which is much better than that in epoch 16 in the previous MLP. That is to say, for the MLP with regularization, it will **be more resistant to over-fitting** since it needs more epochs to suffer from over-fitting.

However, for large number of epochs, there still exists danger for the MLP with regularization to be over-fitting. What we should do is to select appropriate epochs **by trials and errors** to avoid over-fitting problem.

3.4.2 SVD and minimal structure

In this subsection, we aim to choose appropriate number of hidden neurons by SVD. We start our algorithm at **300** hidden neurons, and the threshold of selecting hidden neurons is **95%**.

Then, we only give the visualization of fist 9 iterations for the limit of space since there are all 35 iterations. The result is as follows:

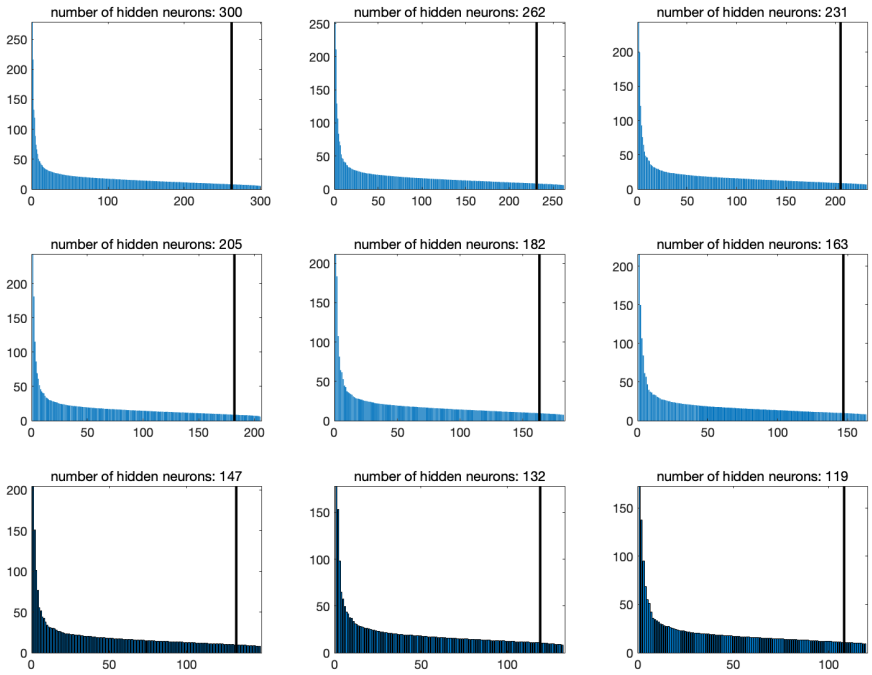

Figure 25: First 9 iterations for SVD

The final result for minimal structure by applying SVD is:

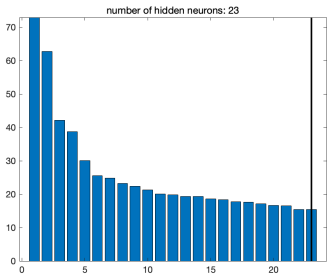

Figure 26: Final result for minimal structure via SVD

Therefore, it is suggested that the **minimal structure of the network is 23**. We should choose hidden neurons around 23 to design our MLP model. After training process, it turns out that for this simple model (only 23 hidden neurons), its testing accuracy can achieve **0.74**, which is a very good result.

3.5 Question e) Sequential Mode of MLP

Here, we give 2 experiments of sequential learning, one is small number of hidden neurons without regularization (use `trainFcn='trainrp'`), and the other is large number of hidden neurons with regularization (use `trainFcn='trainbr'`). To make comparision, we choose the Batch-Mode 1024-23-1 MLP as baseline, whose testing accuracy is 0.74.

Before giving our result, I claim that I modify the sample code in the end of the homework pdf. In that sample code, we actually fix to select first n images as our training set. And our images are in order, which may lead to class-imbalance problem. Therefore, I randomly choose n images to shuffle their order.

3.5.1 23 hidden neurons without regularzation

As for this, we use `trainFcn='trainrp'` to train our MLP sequentially. The result is shown as follows:

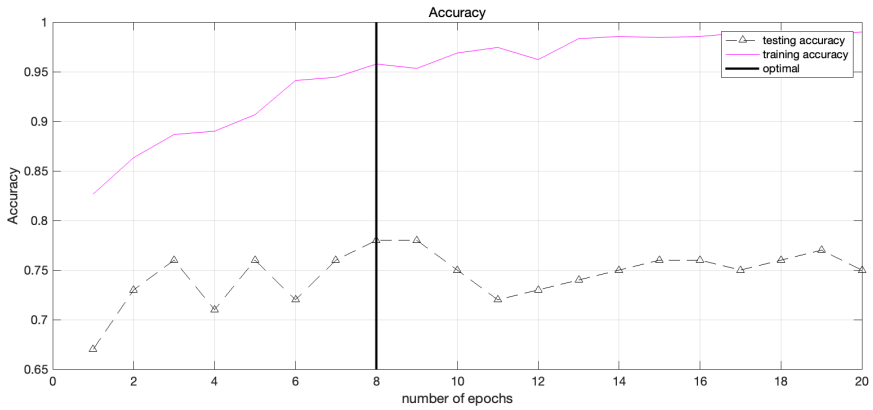

Figure 27: 23 hidden neurons without regularzation

3.5.2 100 hidden neurons with regularization

As for this, we use `trainFcn='trainbr'` and set `trainParam.regularization='0.2'` to train our MLP sequentially. The result is shown as follows:

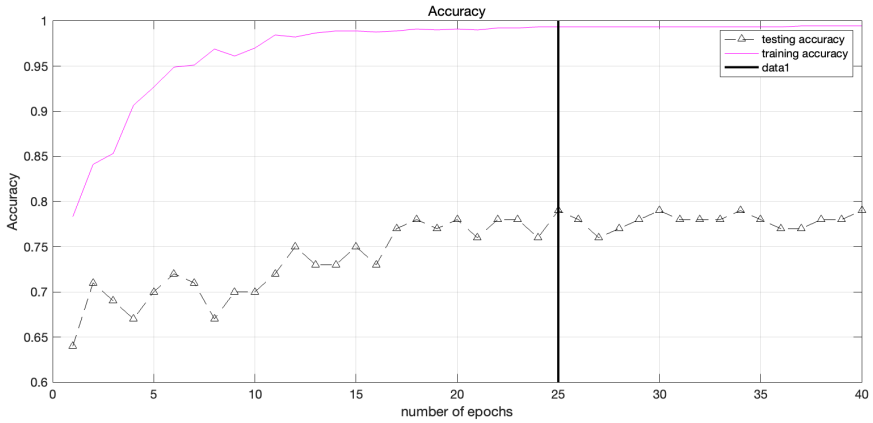

Figure 28: 100 hidden neurons with regularzation

In Figure 27, in epoch 8, we achieve accuracy **0.78 in validation set** and 0.96 in training set. In Figure 28, in epoch 25, we achieve accuracy **0.79 in validation set** and 0.99 in training set. Compared with our baseline (Batch-Mode MLP), there is no ambiguity that **Sequential Mode MLP generalizes much better**, at least for this high-dimensional pattern recognition problem.

Moreover, when we concentrate on Figure 28, we can find that, here the generalization ability of our model keeps on increasing as epochs go on. This is so non-trivial because **all our previous models** seem to share the similarity that, after 10 or 20 epochs, the testing accuracy will start to decrease. However, this model here can **maintain** a high testing accuracy (**around 0.77**), which is a quite good property.

It **never** means that these 2 models are not over-fitting. Instead, we still suffer from the problem of over-fitting. However, what we do in these 2 models is to **release the extent of over-fitting** in the way that we **will not focus too much** on the total training set by training sequentially. Therefore, we will not lose too much ability of generalization, which can be observed from the high testing accuracy (**0.78** and **0.79** respectively).

That finishes my **evaluation** for the **Sequential Mode MLP Model**. Compare with the result we achieved in Question c), i.e., **0.74** testing accuracy, there is no doubt that I will **recommend Sequential Mode MLP** for this pattern recognition problem.

3.6 Question f) Improvements

Here are my schemes which may improve the performance of my MLP, at least in my own perspective:

1. We try to attain more training images.

Explanation: There is no ambiguity that, with training set large enough, we will not suffer from over-fitting problem. I think this is **one of the most powerful** tools to help release over-fitting. However, the cost is very **expensive** for more training samples, especially labelled one.

2. We can make more training images artificially by **rotating, flipping and stretching** on the original training set. Moreover, we even can **add some noise** to achieve more robustness.

Explanation: Let's imagine that, for human beings, our recognition ability is **invariant** under the roatating, flipping and stretching transformation. This is also what our MLP should grasp. Therefore, it is very reasonable to enlarge our training set by these transformation.

3. Try to design the network with **special** structure according to our pattern recognition problem.

Explanation: Actually, we can interpret this in the following way. The special structure of our MLP can be viewed as **clever feature extraction machine**. With feature extraction, we can discard the redundant information and retain those useful in the images. **For one thing**, the reduction in dimensionality will make our algorithm more efficient. **For another thing**, it also will make great help to the generlization behavior since the noice in images is discarded. However, it is difficult to design such clever feature extraction machine.

4 Appendix

All matlab codes (*.m) will be attached in .zip.