CSC411 A2

Q1.

1) How many weights: F * h * w.

2) back propagation
Cross-Entropy:

$$E = -\sum_n \sum_k t_k{}^{(n)} \log\left(o_k{}^{(n)}\right)$$

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial o_k} * \frac{\partial o_k}{\partial z_k} * \frac{\partial z_k}{\partial w_{kj}}$$

$$\frac{\partial E}{\partial o_k} = -\sum_k t_k * \frac{1}{o_k}$$

For the fully connect output layer, since the activation function is soft-max:

$$\frac{\partial o_j}{\partial z_k} = \left(1 - o_j\right) o_j \qquad (k = j)$$

$$\frac{\partial o_j}{\partial z_k} = -o_j o_k \qquad (k \neq j)$$

$$\frac{\partial E}{\partial w_{kj}} = -\sum_{n=1}^{N} \left(t_k{}^{(n)} * \frac{1}{o_k{}^{(n)}} * \left(1 - o_k{}^{(n)}\right) o_k{}^{(n)} + \sum_{k \neq j} \left(t_k{}^{(n)} * \frac{1}{o_k{}^{(n)}} \right) \left(-o_k{}^{(n)} o_j{}^{(n)}\right) \right) h_j{}^{(n)}$$

$$= -\sum_{n=1}^{N} \left(t_k{}^{(n)} * \left(1 - o_k{}^{(n)}\right) + \sum_{k \neq j} \left(t_k{}^{(n)} * \frac{1}{o_k{}^{(n)}} \right) \left(-o_k{}^{(n)} o_j{}^{(n)}\right) \right) h_j{}^{(n)}$$

$$= -\sum_{n=1}^{N} \left(t_k{}^{(n)} - o_k{}^{(n)} * t_k{}^{(n)} + \sum_{k \neq j} \left(t_k{}^{(n)} * \frac{1}{o_k{}^{(n)}} \right) \left(-o_k{}^{(n)} o_j{}^{(n)}\right) \right) h_j{}^{(n)}$$

$$= -\sum_{n=1}^{N} \left(t_k{}^{(n)} + \sum_{k} \left(t_k{}^{(n)} * \frac{1}{o_k{}^{(n)}} \right) \left(-o_k{}^{(n)} o_j{}^{(n)}\right) \right) h_j{}^{(n)}$$

$$= -\sum_{n=1}^{N}\left(t_k{}^{(n)} - \sum_{k} t_k{}^{(n)} o_j{}^{(n)}\right) h_j{}^{(n)} = -\sum_{n=1}^{N}\left(t_k{}^{(n)} - o_j{}^{(n)} \sum_{k} t_k{}^{(n)}\right) h_j{}^{(n)}$$

$$= -\sum_{n=1}^{N}\left(t_k{}^{(n)} - o_j{}^{(n)}\right) h_j{}^{(n)} = \sum_{n=1}^{N}\left(o_j{}^{(n)} - t_k{}^{(n)}\right) h_j{}^{(n)}$$

$$= \sum_{n=1}^{N}\left(o_j{}^{(n)} - t_j{}^{(n)}\right) h_j{}^{(n)}$$

$$\mathrm{w}_{kj} = \mathrm{w}_{kj} - \eta * \sum_{n=1}^{N}\left(o_j{}^{(n)} - t_j{}^{(n)}\right) h_j{}^{(n)}$$

For the hidden layer:

$$\frac{\partial E}{\partial v_{kj}} = \sum_{n=1}^{N} \frac{\partial E}{\partial h_j{}^{(n)}} * \frac{\partial h_j{}^{(n)}}{\partial u_j{}^{(n)}} * \frac{\partial u_j{}^{(n)}}{\partial v_{kj}}$$

Since the activation function is ReLU:

So, when $u_j{}^{(n)} \leq 0$, $\frac{\partial h_j{}^{(n)}}{\partial u_j{}^{(n)}} = 0$, else, $\frac{\partial h_j{}^{(n)}}{\partial u_j{}^{(n)}} = 1$

$$= \sum_{n=1}^{N}\sum_{j=1}^{J}\left(o_j{}^{(n)} - t_j{}^{(n)}\right) w_{kj} * x_k{}^{(n)}$$

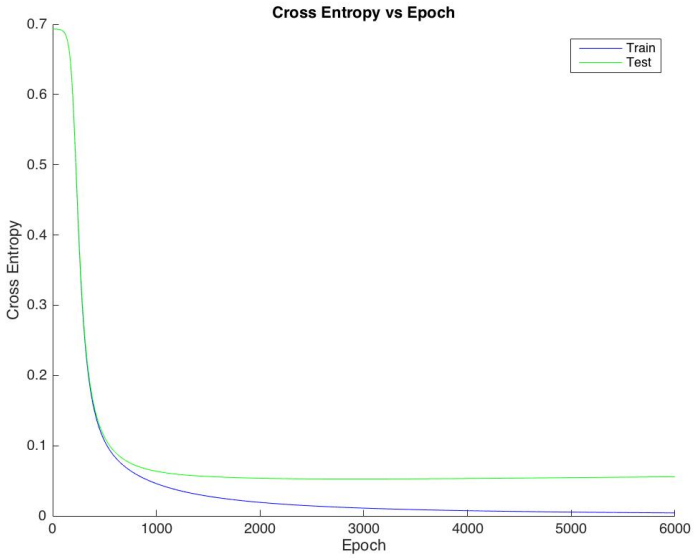
for $u_j{}^{(n)} > 1$

$$v_{kj} = v_{kj} - \eta * \sum_{n=1}^{N}\sum_{j=1}^{J}\left(o_j{}^{(n)} - t_j{}^{(n)}\right) w_{kj} * x_k{}^{(n)}$$

3) How many operations:

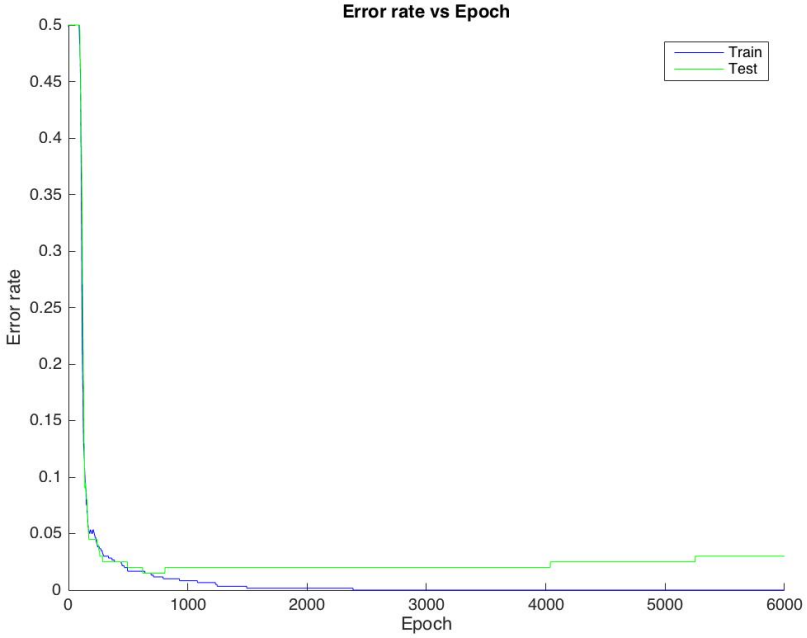
Let us set the stride be S.
((32– w)/S + 1) * (32– h)/S + 1) * F * 3 * w * h

Q2.1



When we keep working on the training set, we can find that the cross entropy of training set will keep reducing and will become 0 at some point. At the same time, the cross entropy of the validation set will also keep reducing but will stop at one point (near 3000 Epoch), and start to goes up. This was caused by “over-fitting”, our model starts to fit the noise in the training set.

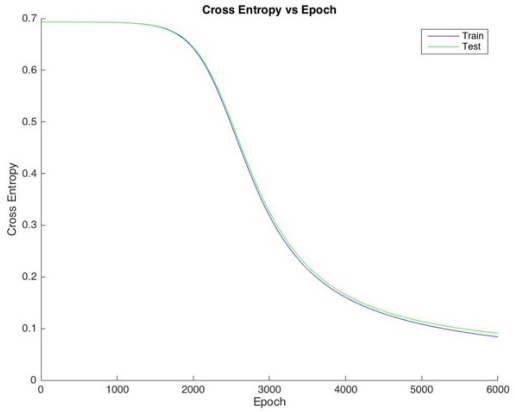
Q2.2



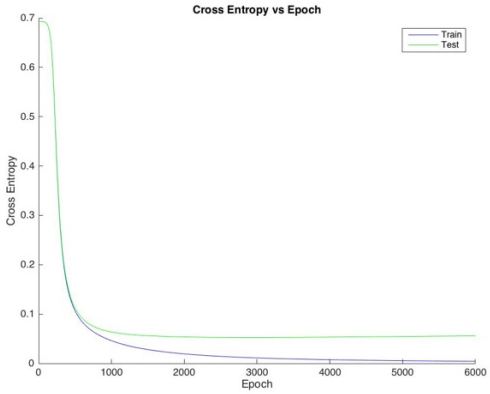
Q2.3

Different Learning rate, Cross Entropy

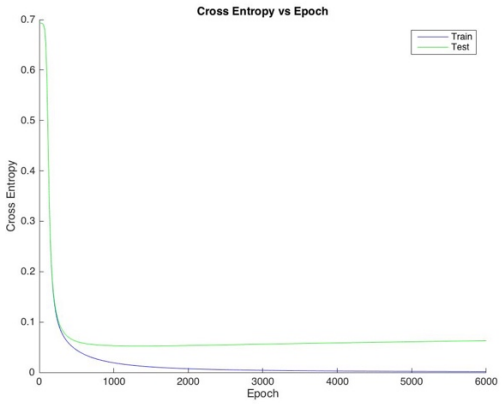
eps = 0.01:



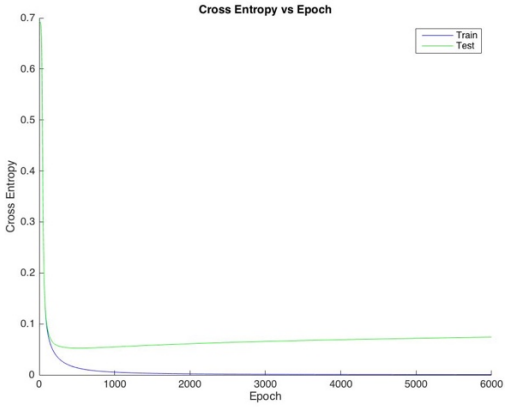
eps = 0.1:



eps = 0.2:



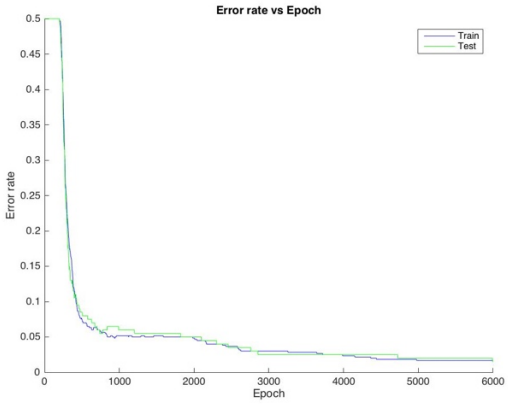
eps = 0.5:

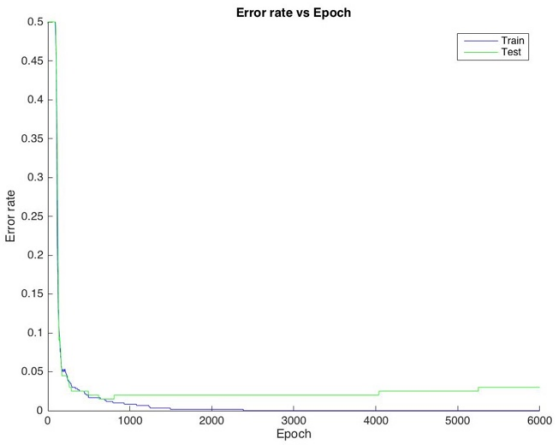


Different Learning rate, Error Rate

eps = 0.01:

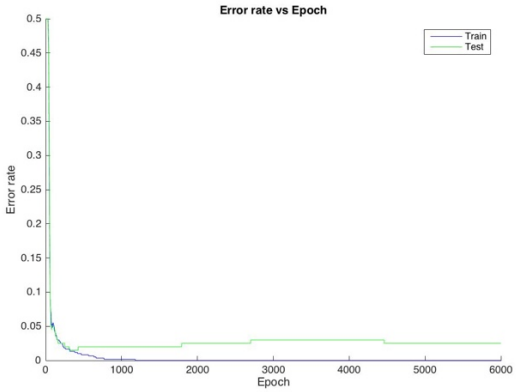
eps = 0.1:

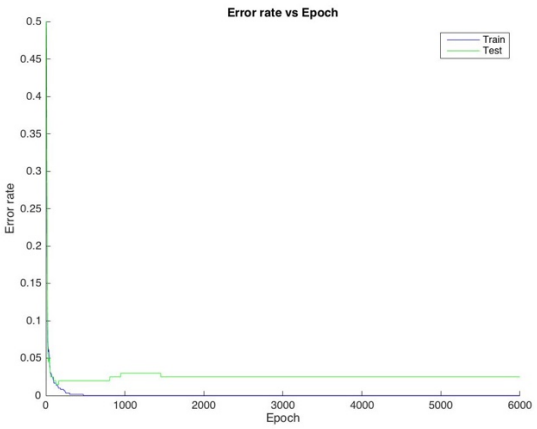




eps = 0.2:

eps = 0.5:



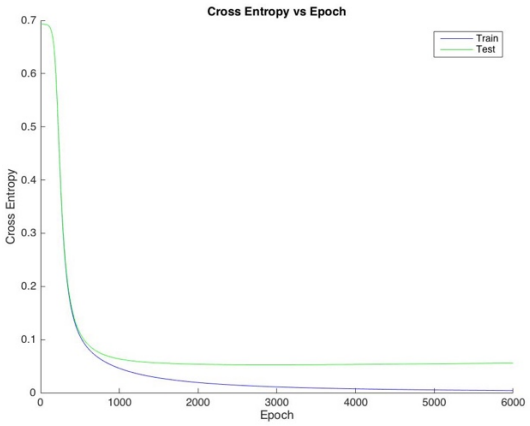


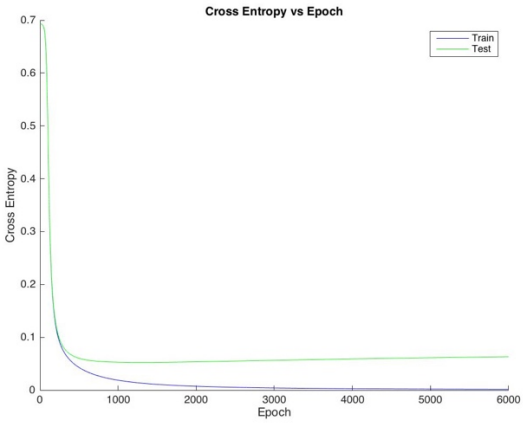
From the picture above, we can find that the with the increase of the learning rate, the convergence rate will also become larger. This phenomenon shows that the model will convergence faster when we apply a larger learning rate.

Different Momentum, Cross Entropy

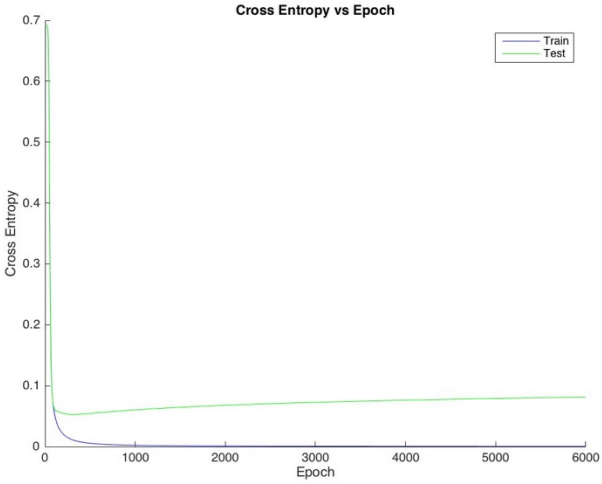
Momentum = 0.0:

Momentum = 0.5:



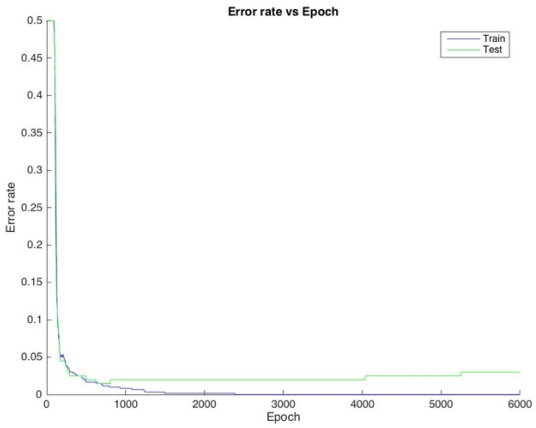


Momentum = 0.9

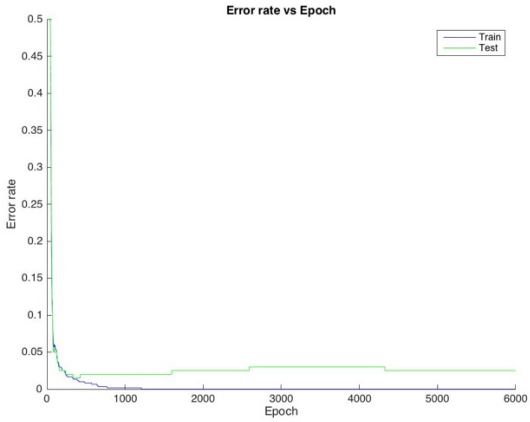


Different Momentum, Error Rate

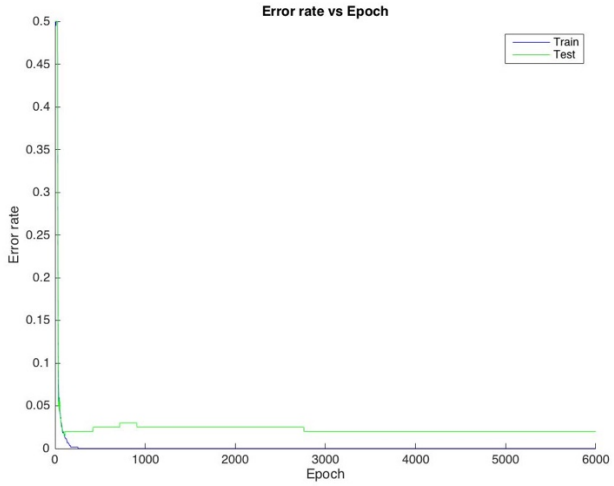
momentum = 0.0:



momentum = 0.5:



momentum = 0.9

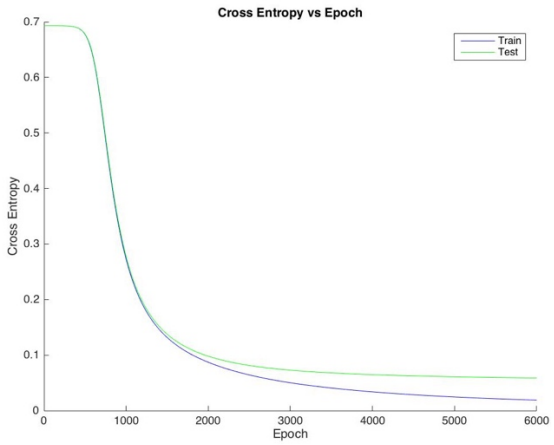


From the picture above, we can find that the with the increase of the momentum, the convergence rate will also become larger. This phenomenon shows that the model will convergence faster when we apply a larger momentum to learning rate. However, if the momentum is very large, it will also cause a huge difference between validation data's results and train data's result.

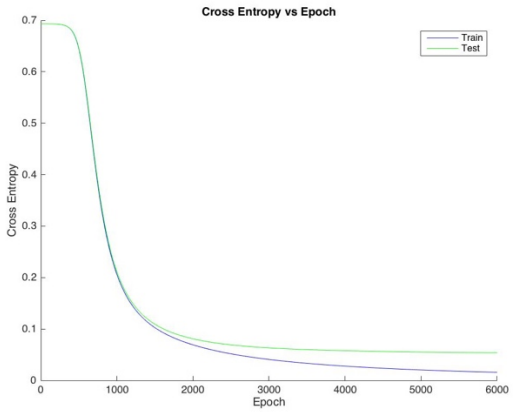
From the data above, we can find that both small learning rate (also momentum) and large learning rate (also momentum) cannot be considered as good choices. Lower value of parameters will make the model converge slowly, and larger value will cause a huge difference between validation data's results and train data's result. As a result, we should choose the value like 0.2 (learning rate) and 0.5(momentum).

Q.2.4

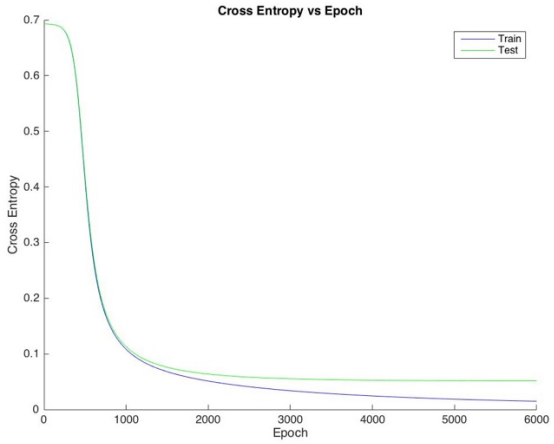
Cross Entropy:
Hidden units = 2: Hidden units = 5:

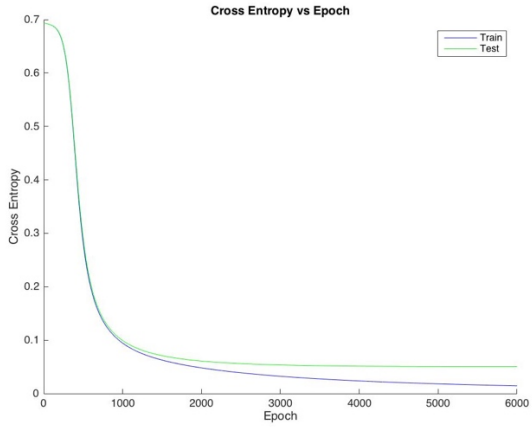


Hidden units = 30:



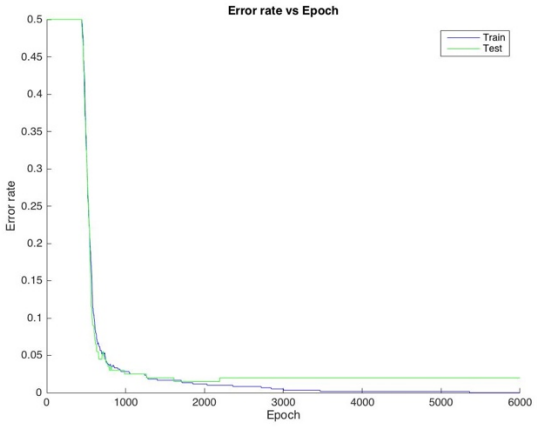
Hidden units = 100:



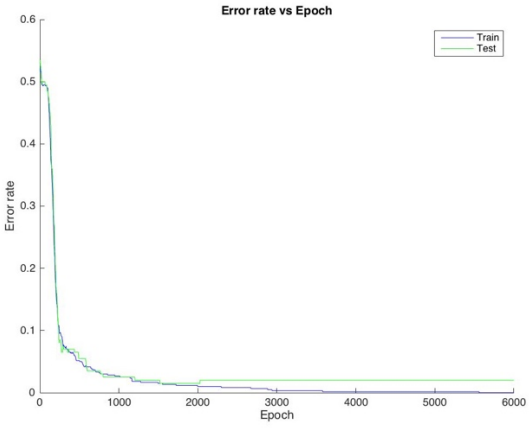


Error Rate:

Hidden units = 2:

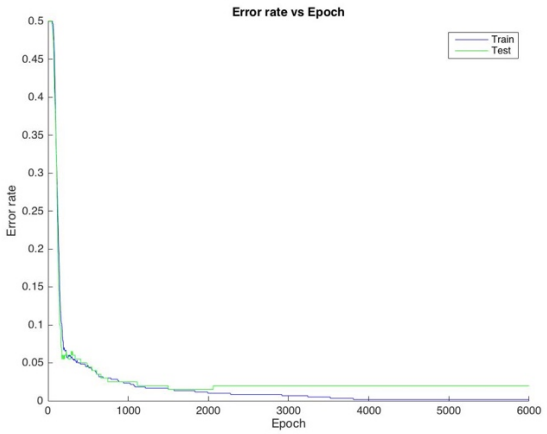


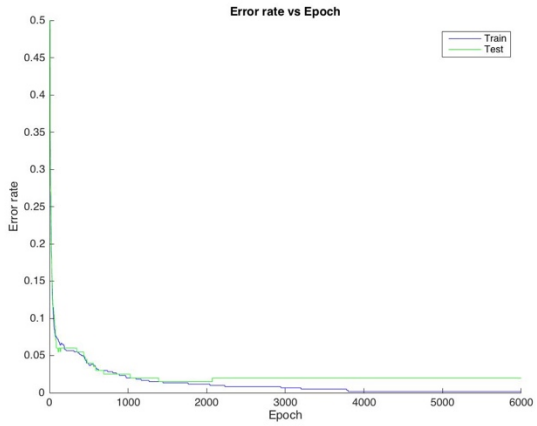
Hidden units = 5:



Hidden units = 30:

Hidden units = 100:





Generalization:
With the increase of the number of hidden units, we can find that the model is less generalized, more easily to fit the noise in the data (over fitting). When hidden units = 30 and 100, models are easier to fit the training data than the model which hidden units = 2 and 5. As mentioned in the class, when a model is simpler, the better generalized it does. So, when the hidden units equal to 2, the model is more generalized.

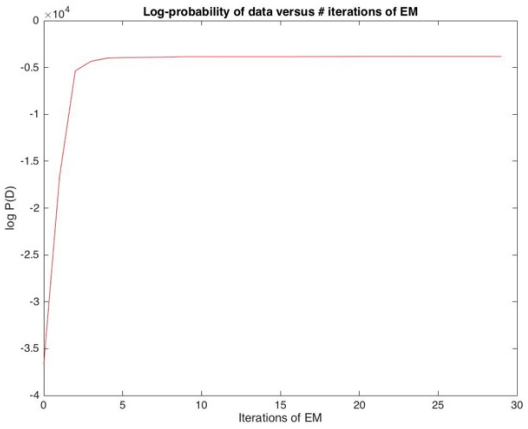
Convergence properties:
With the increase of the number of hidden units, we can find that the convergence rate will also become faster. For the cross entropy, the result of data become different near 1500 Epoch when hidden units = 2 or 5. But, when hidden units = 30 or 100, the result of data become different near 1000 Epoch. The Error Rate part is similar with the cross entropy part.

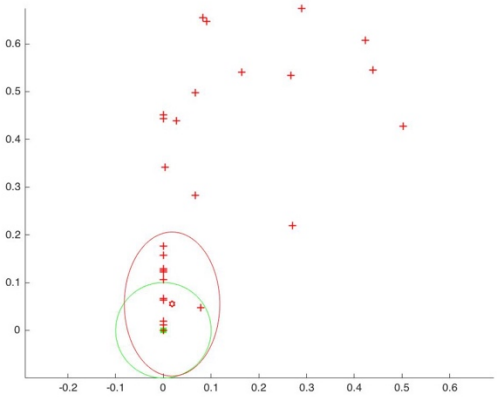
Q2.5
K-nn:
Train data: 0.9950
Test data: 0.9875
Validation data: 0.9922

NN:
Train data: 0.9877
Test data: 0.9413
Validation data: 0.9477

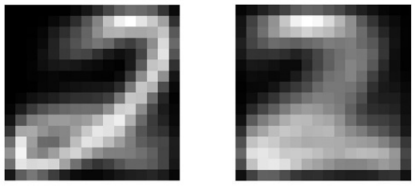
From the data above, we can find that K-nn did little better than Neural network in this case, this is because that neural network is more robust than the k-nn, and data complexity is small.

Q3.2
For 2's:
mixing proportions: (0.4900, 0.5100)
log P(train2): -3.820e+03

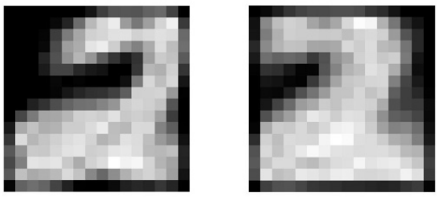




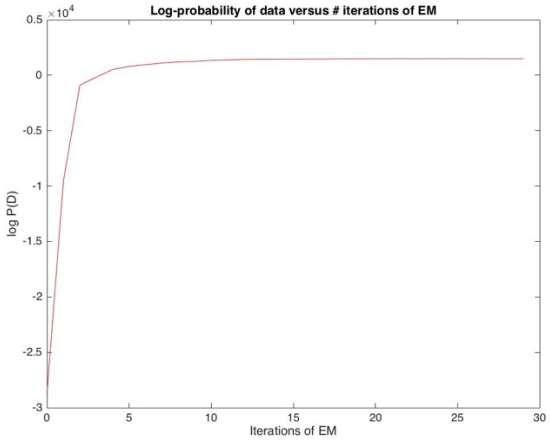
mean:

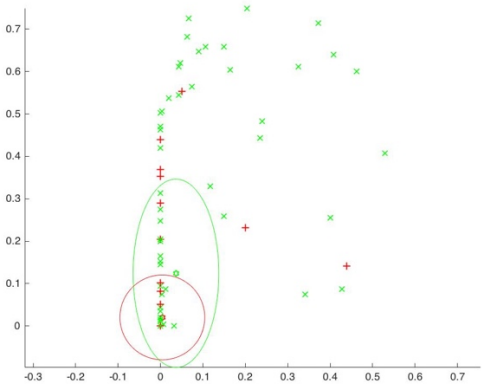


variance:

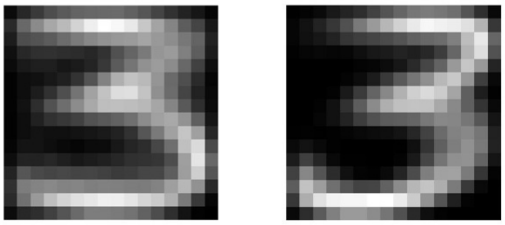


For 3’s:
mixing proportions: (0.4768, 0.5232)
log P(train2): 1.485e+03

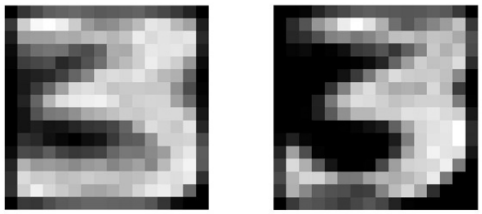




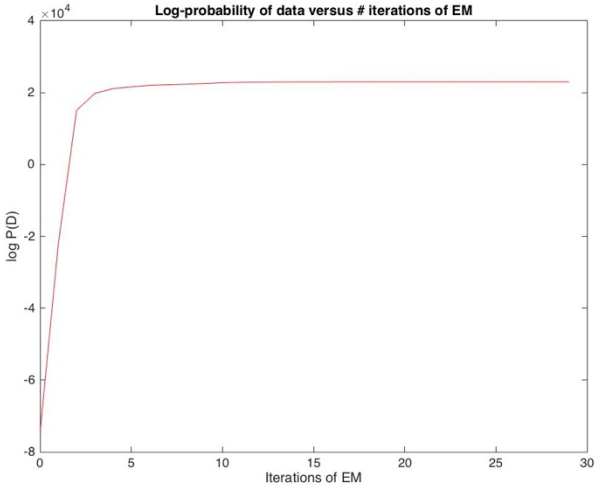
mean:



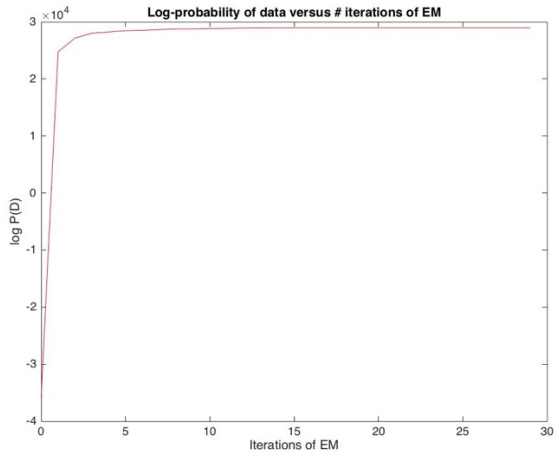
variance:



Q3.3
Random:
Log-prob: 2.5495e+04

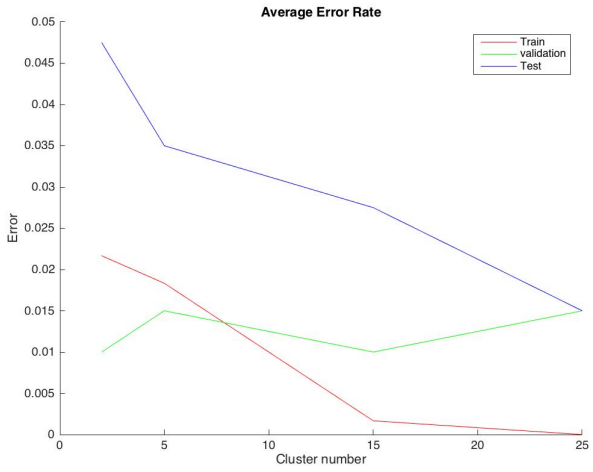


K-means:
Log-prob:2.8348e+04



From the information above, we can find that the convergence speed of k-means algorithm is greater than the random one. The final log-prob result of k-means is also larger than the random one. We can conclude that k-means does better than randomly initialized in this case.

Q3.4



1) Since the model is constructed by the training data. When the number of clusters increase, the model will start to over fitting the train data, and the error rate will keep decreasing.
2) The error rate of the test data will decrease at the first, and the rate will keep decreasing. Finally, the decrease rate will become 0, and the error rate will start to increase. The reason why this happen is that the model is over fitting the training data, it is less generalized.

3) I will choose a model whose accuracy is good and more generalized. In this question, I will choose 15 as the number of clusters. This is because the error rate of validation data start to increase at this point.

Q3.5
MOG:
Training data: (1 – 0.0017) = 0.9983
Validation data: (1 – 0.0100) = 0.9900
Test data: (1-0.0225) = 0.9775

NN:
Training data: 1 - 0.001667 = 0.9983
Validation data: 1 – 0.02 = 0.98
Test data: 1 – 0.0275 = 0.9725

Performance:
From the data, above, we can find that MOG did better than NN in this case. In NN, we choose the number of mixture components as the hidden unit number, this number is not very big, which means the model is not very Complicated. If the data we need to deal with is more complicated than this one, I think it's hard to say that MOG did better than NN. If the data is complicated enough, NN will do better than MOG.

Hidden unit, mixture components:
In this case, we choose the number of mixture components as the hidden unit number, which means we treat hidden unit as mixture components in this case, both of them take the information from the input source, do some calculation and transmit the result to the final output. It is hard to say which one is better in this case. But, I think if we need to train a high dimension data, hidden units will do better than mixture components, we can increase the number of hidden units to deal with the high dimension data, which is more efficient than the mixture components.