CSC411 Fall 2015 Assignment 2

Mengdi Xu c2xumeng 999507939

1 EM for Mixture of Gaussians (15 pts)

$$p(x) = \sum_{k=1}^{K} \pi_k N(x|\mu_k, \Sigma_k)$$

Conditional probability (using Bayes Rule) of z given x:

$$\begin{aligned}\gamma(z_k^n) &= p(z_k = 1|x) \\ &= \frac{p(z_k = 1)p(x|z_k = 1)}{p(x)} \\ &= \frac{p(z_k = 1)p(x|z_k = 1)}{\sum_{j=1}^{K} p(z_j = 1)p(x|z_j = 1)} \\ &= \frac{\pi_k N(x|\mu_k, \Sigma)}{\sum_{j=1}^{K} \pi_j N(x|\mu_j, \Sigma)}\end{aligned}$$

Step E:

$$\text{Let } m = \dim(x), \text{ then } \mu_k = \begin{pmatrix} \mu_1 \\ \vdots \\ \mu_m \end{pmatrix}_{m*1}, \Sigma = \begin{pmatrix} \sigma_{11} & \dots & \sigma_{1m} \\ \vdots & \ddots & \vdots \\ \sigma_{m1} & \dots & \sigma_{mm} \end{pmatrix}$$

$$\begin{aligned}p(z_k = 1|x) &= \frac{\pi_k N(x|\mu_k, \Sigma)}{\sum_{j=1}^{K} \pi_j N(x|\mu_j, \Sigma)} \\ &= \frac{\pi_k \frac{1}{\sqrt{(2\pi)^m |\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu_k)^T \Sigma^{-1}(x-\mu_k)\right)}{\sum_{j=1}^{K} \pi_j \frac{1}{\sqrt{(2\pi)^m |\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu_j)^T \Sigma^{-1}(x-\mu_j)\right)} \\ &= \frac{\pi_k \exp\left(-\frac{1}{2}(x-\mu_k)^T \Sigma^{-1}(x-\mu_k)\right)}{\sum_{j=1}^{K} \pi_j \exp\left(-\frac{1}{2}(x-\mu_j)^T \Sigma^{-1}(x-\mu_j)\right)}\end{aligned}$$

Step M:

$$L = \sum_{n=1}^{N} \log\left(\sum_{k=1}^{K} \pi_k N(x^n|\mu_k, \Sigma)\right), subject\ to\ \sum \pi_i = 1$$

Want to find:

1. $\frac{\partial}{\partial \mu_k} L = 0$
2. $\frac{\partial}{\partial \Sigma} L = 0$

3. $\frac{\partial}{\partial \pi_k}\left(L + \lambda(1 - \sum \pi_i)\right) = 0$ (using Lagrange multiplier)

1. $0 = \frac{\partial}{\partial \mu_k} L$

$$= \sum_{n=1}^{N} \frac{1}{\sum_{k=1}^{K} \pi_k N(x^n|\mu_k, \Sigma)} \pi_k \frac{\partial N(x^n|\mu_k, \Sigma)}{\partial \mu_k}$$

$$= \sum_{n=1}^{N} \frac{\pi_k}{\sum_{k=1}^{K} \pi_k N(x^n|\mu_k, \Sigma)} N(x^n|\mu_k, \Sigma)\Sigma^{-1}(x^n - \mu_k)$$

$$= \sum_{n=1}^{N} \gamma(z_k^n)\Sigma^{-1}(x^n - \mu_k)$$

$\Rightarrow \sum_{n=1}^{N} \gamma(z_k^n)\Sigma^{-1}(x^n) = \sum_{n=1}^{N} \gamma(z_k^n)\Sigma^{-1}(\mu_k)$

$\Rightarrow \sum_{n=1}^{N} \gamma(z_k^n)x^n = \sum_{n=1}^{N} \gamma(z_k^n)\mu_k$

$\Rightarrow \mu_k = \sum_{n=1}^{N} \gamma(z_k^n)x^n \,/\, \sum_{n=1}^{N} \gamma(z_k^n)$

2. $0 = \frac{\partial}{\partial \Sigma} L$

$$= \sum_{n=1}^{N} \frac{1}{\sum_{k=1}^{K} \pi_k N(x^n|\mu_k, \Sigma)} \sum_{k=1}^{K} \pi_k \, (-\Sigma^{-1}) N(x^n|\mu_k, \Sigma) \left(\frac{1}{2} - \frac{1}{2}(x - \mu_k)(x - \mu_k)^T\Sigma^{-1}\right)$$

$$= \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_k^n)(-\Sigma^{-1}) \left(\frac{1}{2} - \frac{1}{2}(x - \mu_k)(x - \mu_k)^T\Sigma^{-1}\right)$$

$\Rightarrow 0 = \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_k^n)(1 - 1(x - \mu_k)(x - \mu_k)^T\Sigma^{-1})$

$\Rightarrow \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_k^n) = \left(\sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_k^n)(x - \mu_k)(x - \mu_k)^T\right)\Sigma^{-1}$

$\Rightarrow \Sigma = \left(\sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_k^n)(x - \mu_k)(x - \mu_k)^T\right) / \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_k^n)$

$\Rightarrow \Sigma = \left(\sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_k^n)(x - \mu_k)(x - \mu_k)^T\right) / N$

3. $0 = \frac{\partial}{\partial \pi_k}\left(L + \lambda(1 - \sum \pi_i)\right)$

$$= \left(\sum_{n=1}^{N} \frac{1}{\sum_{k=1}^{K} \pi_k N(x^n|\mu_k, \Sigma)} N(x^n|\mu_k, \Sigma)\right) - \lambda$$

$\Rightarrow \sum_{n=1}^{N} \frac{1}{\sum_{k=1}^{K} \pi_k N(x^n|\mu_k, \Sigma)} N(x^n|\mu_k, \Sigma) = \sum_{n=1}^{N} \frac{\gamma(z_k^n)}{\pi_k} = \lambda$

$\Rightarrow \pi_k\lambda = \sum_{n=1}^{N} \gamma(z_k^n), \pi_j\lambda = \sum_{n=1}^{N} \gamma(z_j^n)$

$\Rightarrow \frac{\pi_j}{\pi_k} = \frac{\sum_{n=1}^{N} \gamma(z_j^n)}{\sum_{n=1}^{N} \gamma(z_k^n)}$

$\Rightarrow \pi_j = \frac{\pi_k \sum_{n=1}^{N} \gamma(z_j^n)}{\sum_{n=1}^{N} \gamma(z_k^n)}$

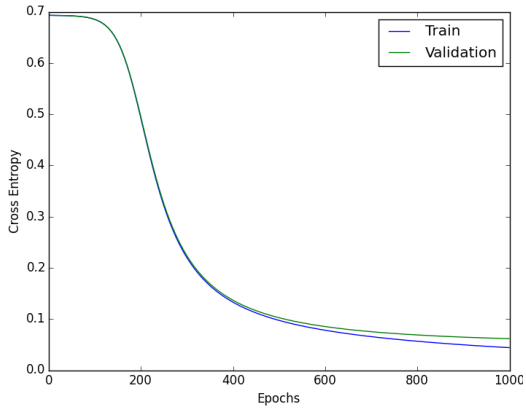
$\Rightarrow 1 = \sum_j \pi_j = \sum_{j=1}^{K} \left(\frac{\pi_k \sum_{n=1}^{N} \gamma(z_j^n)}{\sum_{n=1}^{N} \gamma(z_k^n)}\right) = \pi_k \frac{\sum_{j=1}^{K} \sum_{n=1}^{N} \gamma(z_j^n)}{\sum_{n=1}^{N} \gamma(z_k^n)}$

⇨ $\pi_k = \frac{\sum_{n=1}^{N} \gamma(z_k^n)}{\sum_{j=1}^{K}\sum_{n=1}^{N} \gamma(z_j^n)} = \frac{\sum_{n=1}^{N} \gamma(z_k^n)}{\sum_{n=1}^{N}\sum_{j=1}^{K} \gamma(z_j^n)} = \frac{\sum_{n=1}^{N} \gamma(z_k^n)}{\sum_{n=1}^{N} 1} = \frac{N_k}{N}$,
where $N_k = \sum_{n=1}^{N} \gamma(z_k^n)$.

2 Neural networks (40 points)

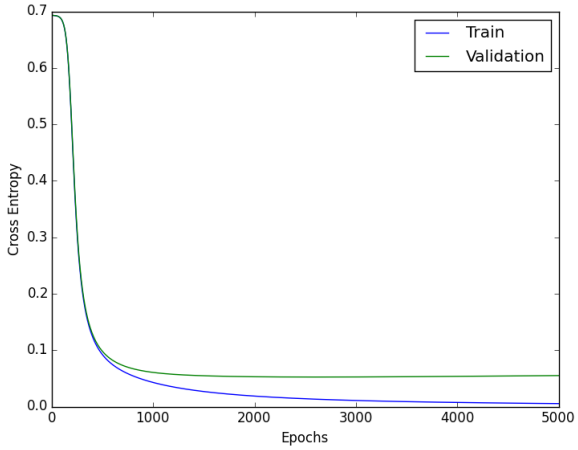
2.1 Basic generalization [8 points]

- **Statistics:**
  - Mean Classification Error and Cross Entropy plots with epochs=1000:



Error: Train 0.04399 Validation 0.06159 Test 0.07212

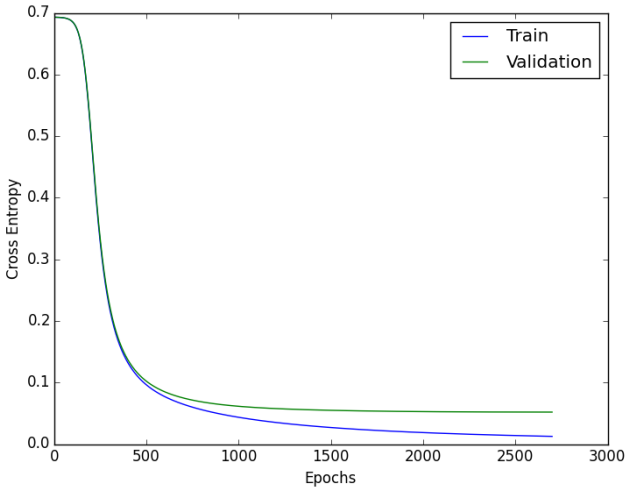
  - Mean Classification Error and Cross Entropy plots with epochs=5000:



Error: Train 0.00536 Validation 0.05496 Test 0.07202

(By examining the output data from the script, the minimum cross entropy of validation set occurs around the 2550th as 0.05236.)

- Mean Classification Error and Cross Entropy plots with epochs=3000:



Error: Train 0.05881 Validation 0.06988 Test 0.07742

- **How does the network's performance differ on the training set versus the validation set during learning?**

  Both cross entropy curves have a dramatic decrease before the first 500 epochs, indicating the models are updating fast. Each iteration can gain a lot of information.

  The rates of decrease slow down after approximately 500 epochs for both curves. The cross entropy of training set slowly converges to 0, while the cross entropy curve for validation set hits its minimum around the 2250th epochs before it increases.

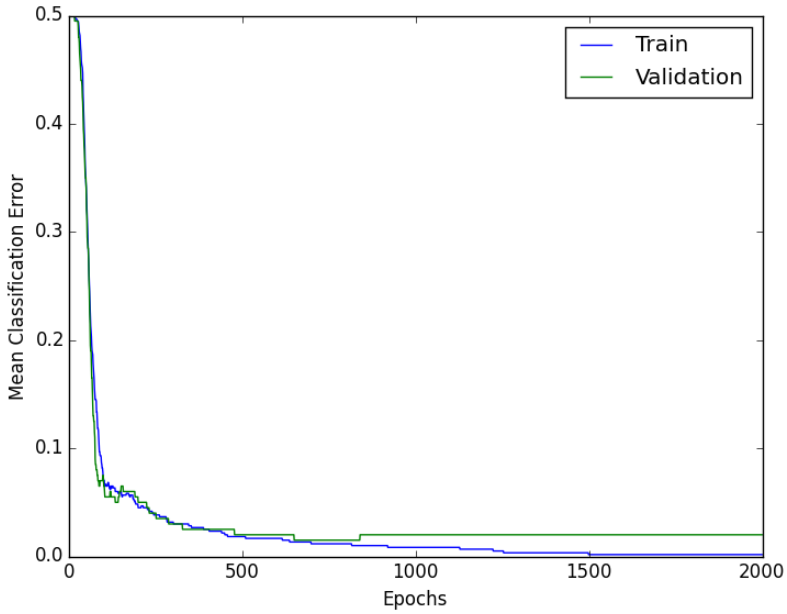
  The reason why CE of validation set starts to increase after some point is because while the model is learning more and more from the training set, it has learnt the noise terms as well. Therefore, the model will not perform as well on the other data sets.

2.2 Classification error [8 points]

- **Implementation of an alternative performance measure to the cross entropy, the mean classification error.**

  Please refer to nn.py.

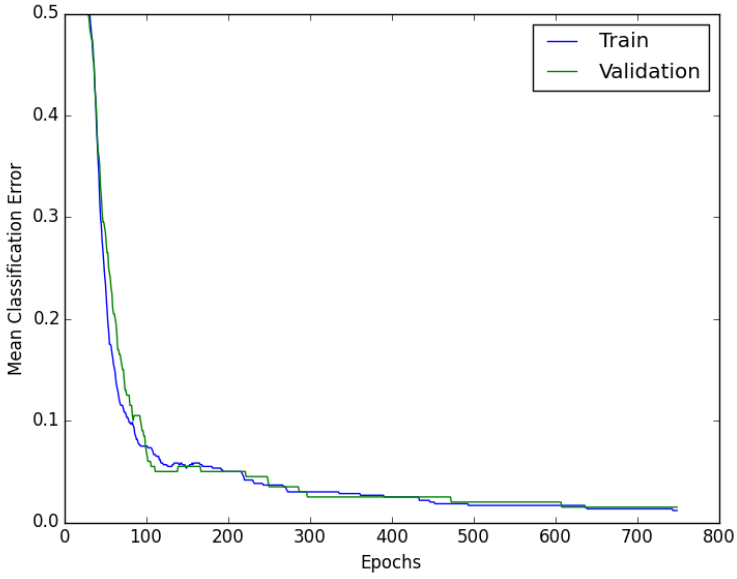
- **Plot the classification error vs. number of epochs, for both training and validation sets.**
  - Mean Classification Error and Cross Entropy plots with epochs=2000:



MCE: Train 0.00167 Validation 0.02000 Test 0.02500

(The minimum of MCE of validation set occurs around the 750th epochs.)

- Mean Classification Error and Cross Entropy plots with epochs=750:



MCE: Train 0.01167 Validation 0.01500 Test 0.02250

- **How does the network's performance differ on the training set versus the validation set during learning?**

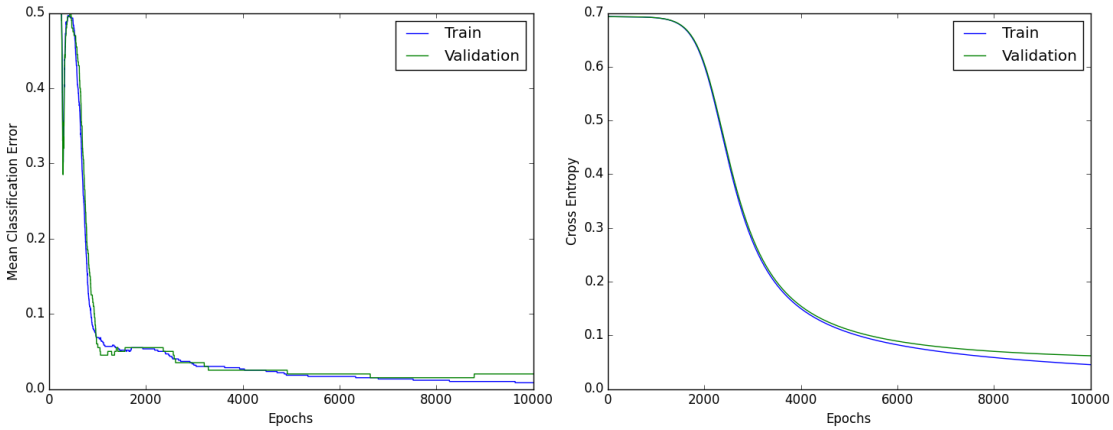
  Both MCE curves have a dramatic decrease before the first 100 epochs, indicating the models are updating fast. Each iteration can gain a lot of information.

  The rates of decrease slow down after approximately 100 epochs for both curves. The cross entropy of training set slowly converges to 0, while the cross entropy curve for validation set hits its minimum around the 750th epochs.

  The reason why MCE of validation set starts to increase after some point is because while the model is learning more and more from the training set, it has learnt the noise terms as well. Therefore, the model will not perform as well on the other data sets.

2.3 Learning rate [8 points]

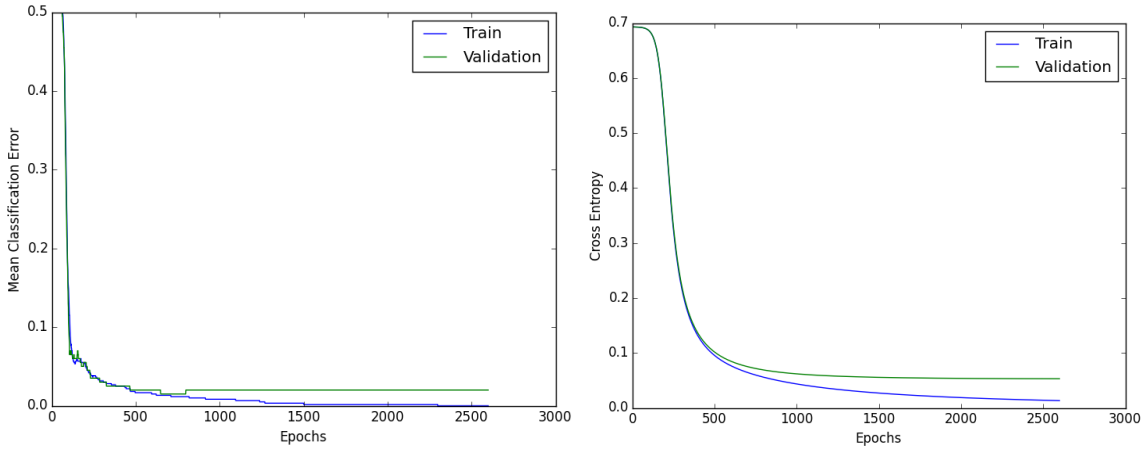
- **Try different values of the learning rate ϵ:**
  - $\boldsymbol{\epsilon = 0.01, epochs = 10000}$



Error: Train 0.04529 Validation 0.06193 Test 0.07256

MCE: Train 0.00833 Validation 0.02000 Test 0.02750

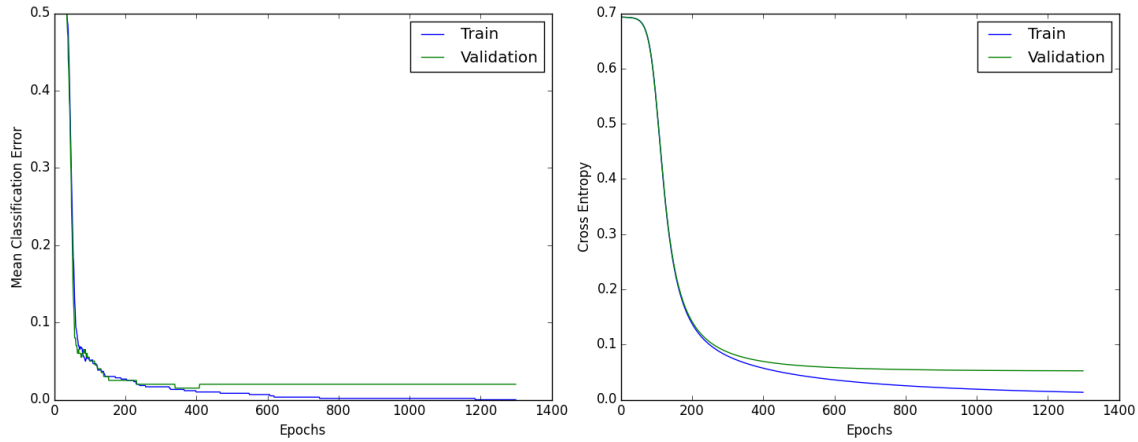
- $\epsilon = 0.1, epochs = 2600$



Error: Train 0.01320 Validation 0.05305 Test 0.06745

MCE: Train 0.00000 Validation 0.02000 Test 0.03000

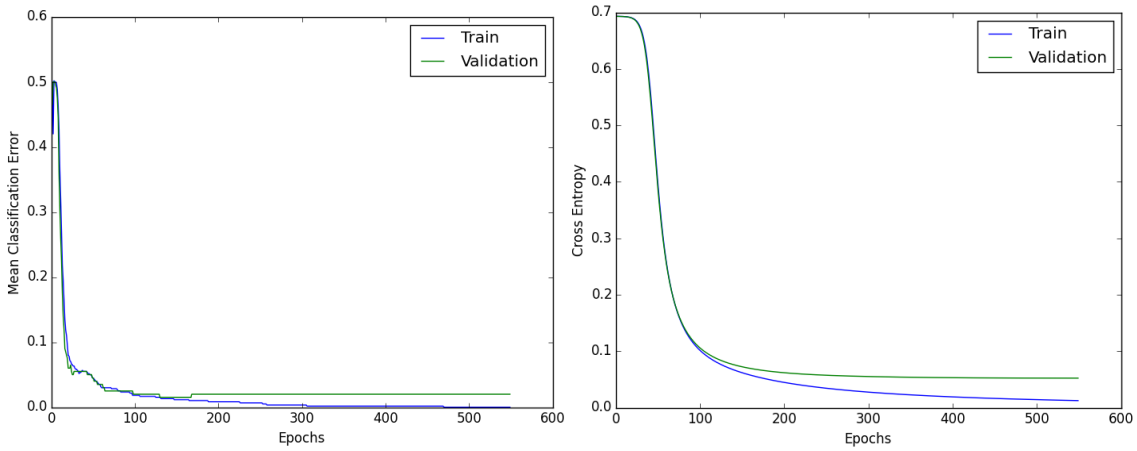
- $\epsilon = 0.2, epochs = 1300$



Error: Train 0.01340 Validation 0.05232 Test 0.06787

MCE: Train 0.00000 Validation 0.02000 Test 0.03000

- $\epsilon = 0.5, epochs = 550$



Error: Train 0.01236 Validation 0.05219 Test 0.06795

MCE: Train 0.00000 Validation 0.02000 Test 0.03000

- **What happens to the convergence properties of the algorithm as the learning rate changes (looking at both cross entropy and %Correct)?**

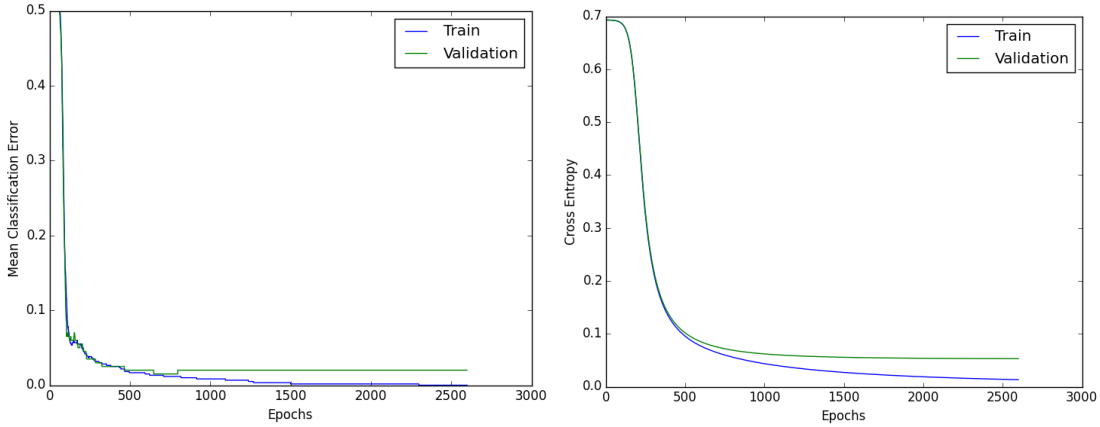
The best number of epochs occurs while the MCE (i.e. 1 - %Correct) or CE curves of validation set is at their minimum. The observation to the above information can be concluded into the table below:

Learning rate	Best epochs from MCE	Best epochs from CE
0.01	8000	12000
0.1	650	2500
0.2	380	1250
0.5	150	550

Therefore, we can conclude that it takes longer to converge for smaller learning rates.

➢ **Try different values of momentum, fix learning rate to be 0.1:**

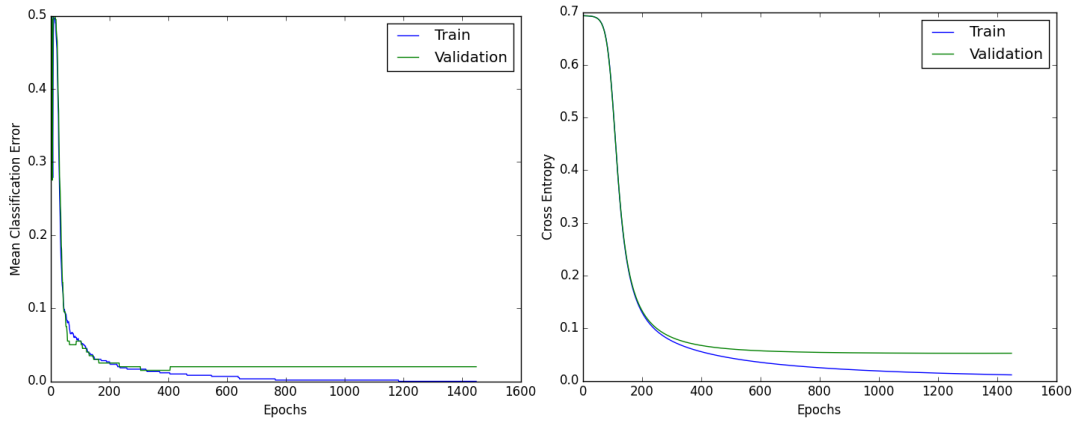
- $\boldsymbol{Momentum = 0, epochs = 2600}$



Error: Train 0.01320 Validation 0.05305 Test 0.06745

MCE: Train 0.00000 Validation 0.02000 Test 0.03000

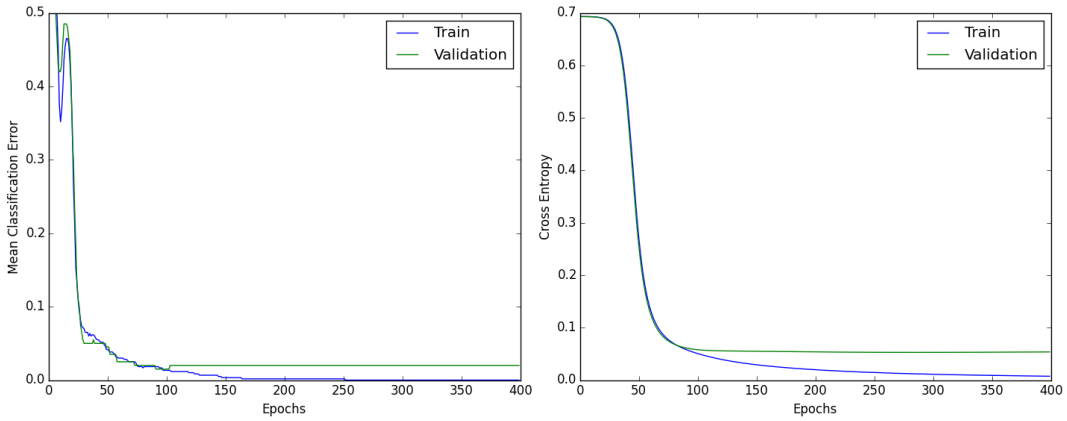
- $\boldsymbol{Momentum = 0.5, epochs = 1450}$



Error: Train 0.01137 Validation 0.05239 Test 0.06837

MCE: Train 0.00000 Validation 0.02000 Test 0.03000

- $\boldsymbol{Momentum = 0.9, epochs = 400}$



Error: Train 0.00735 Validation 0.05369 Test 0.07244

MCE: Train 0.00000 Validation 0.02000 Test 0.03000

➢ **How does the momentum affect convergence rate?**

The best number of epochs occurs while the MCE (i.e. 1 - %Correct) or CE curves of validation set is at their minimum. The observation to the above information can be concluded into the table below:

Momentum	Best epochs from MCE	Best epochs from CE
0.0	650	2500
0.5	380	1400
0.9	70	400

Therefore, we can conclude that it takes longer to converge for smaller momentums.

➢ **How would you choose the best value of these parameters?**

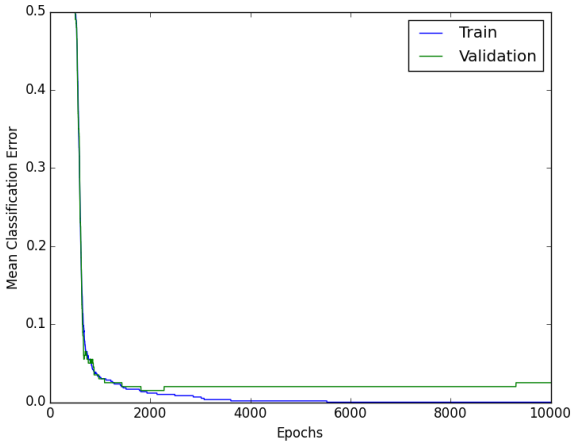
Learning rate: controls the size of "steps" in each iteration. Too small of a learning rate will result a very slow converging rate of the model.

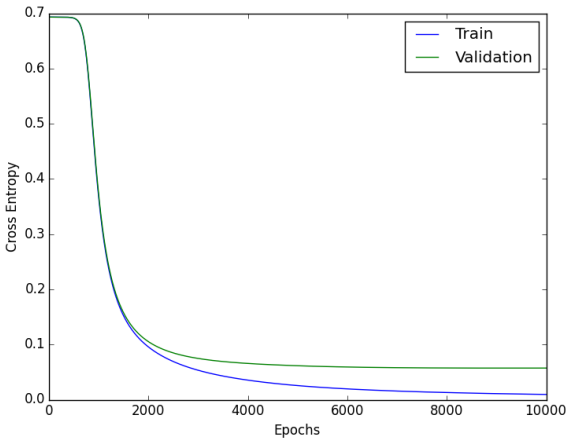
Momentum: regularization term, allows the attenuation of oscillations in the gradient descent process. Larger momentum helps in converging faster.

To find the best value of these parameters, I would choose a small but not too small learning rate (i.e. 0.01 to 0.1), and a moderate momentum value (i.e. 0.3 to 0.5). With the correctly chosen parameters, the model will learn fast without missing important details or too much oscillations.

2.4 Number of hidden units [8 points]

- **Set the learning rate to 0.02, momentum to 0.5 and try different numbers of hidden units on this problem.**
  - **2 hidden units, epochs = 10000**

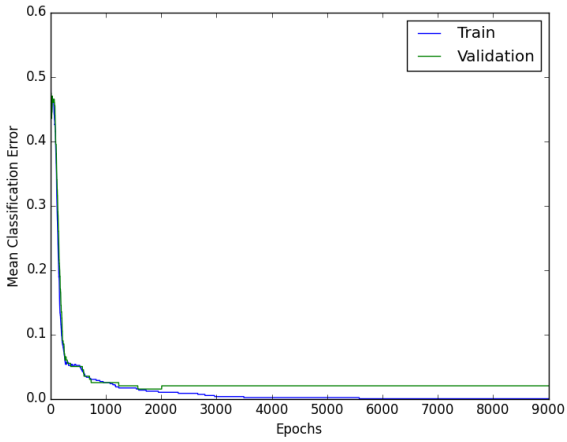


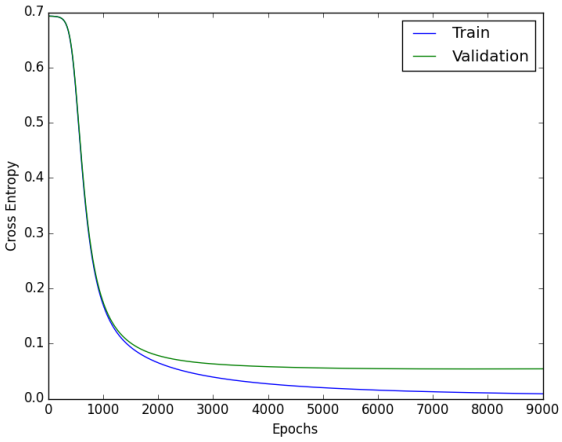


Error: Train 0.00953 Validation 0.05733 Test 0.07395

MCE: Train 0.00000 Validation 0.02500 Test 0.03000

- **5 hidden units, epochs = 9000**

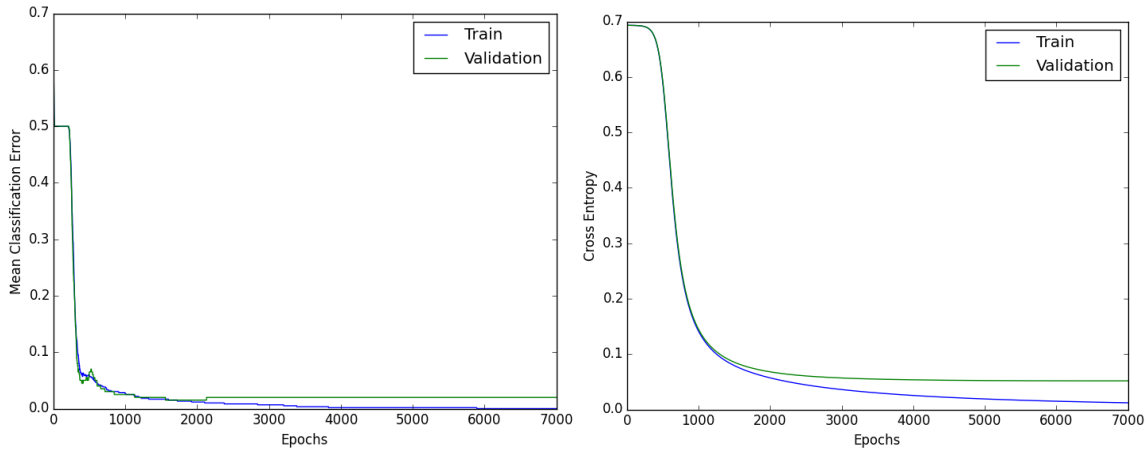




Error: Train 0.00888 Validation 0.05410 Test 0.07106

MCE: Train 0.00000 Validation 0.02000 Test 0.03000

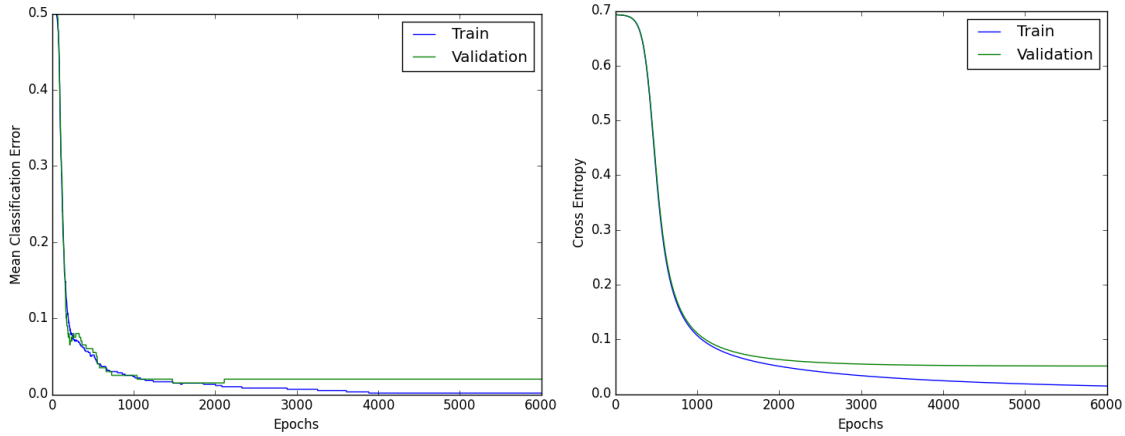
- **10 hidden units (original), epochs = 7000**



Error: Train 0.01209 Validation 0.05170 Test 0.06697

MCE: Train 0.00000 Validation 0.02000 Test 0.02750

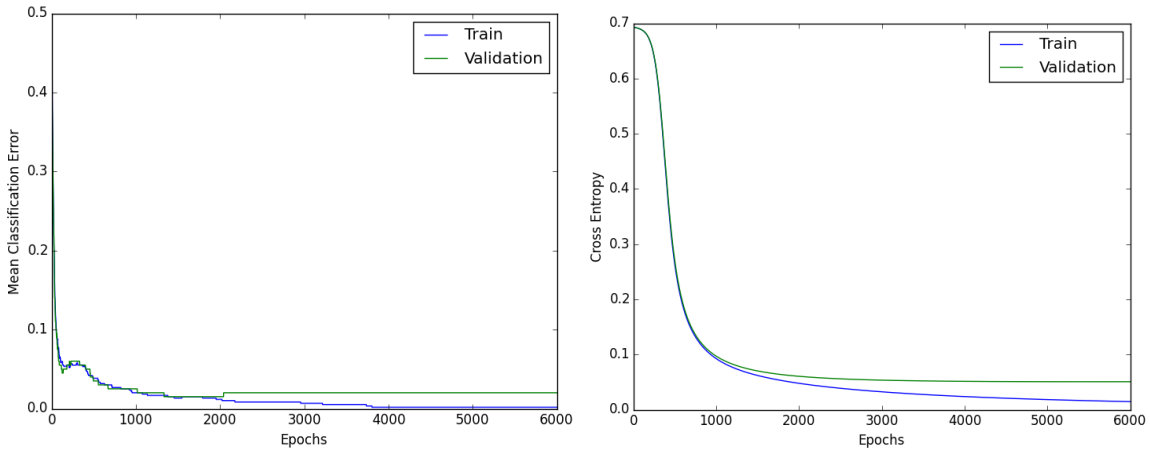
- **30 hidden units, epochs = 6000**



Error: Train 0.01477 Validation 0.05132 Test 0.06562

MCE: Train 0.00167 Validation 0.02000 Test 0.02750

- **100 hidden units**



Error: Train 0.01460 Validation 0.05072 Test 0.06560

MCE: Train 0.00167 Validation 0.02000 Test 0.02750

➢ **Describe the effect of this modification on the convergence properties.**

The best number of epochs occurs while the MCE (i.e. 1 - %Correct) or CE curves of validation set is at their minimum. The observation to the above information can be concluded into the table below:

# hidden units	Best epochs from MCE	Best epochs from CE
2	2050	9000
5	1800	7500
10	1700	6400
30	1700	6000
100	1650	5400

Therefore, we can conclude that it takes longer to converge for fewer hidden units.

➢ **Generalization**

If the size of training set is not large enough, Overfitting might happen during neural network training. This is because the network has memorized the training examples, as well as the sampling errors. The network will not generalize well if there is an overfitting. These methods are usually used in preventing overfitting:

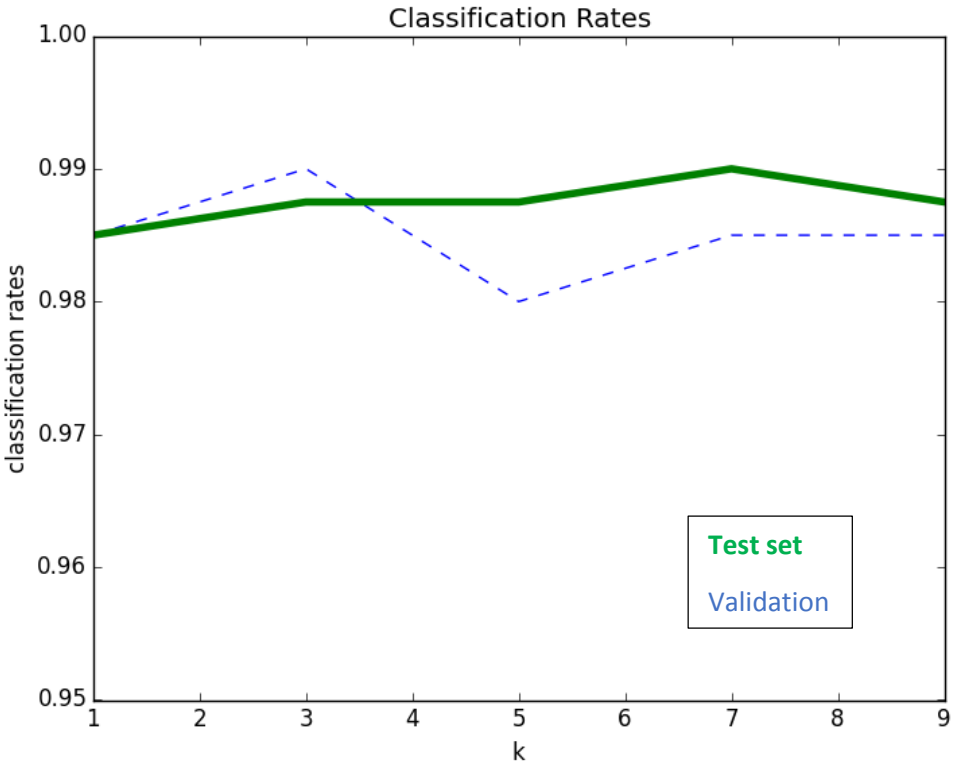
1. Early stopping;

2. Limitation of the number of hidden units;
3. Limiting the size of the weights.

We performed all of the three methods in the previous questions. Early stopping suggests to stop the training before it has time to over fit the training set. In the training process, we introduced a validation data set to determine how many iterations is needed for the model to converge. To limit the number of hidden units, we set the number of hidden units needed and examined on the performance while the number of hidden units changes. We also introduced the momentum term in order to perform weight decay, thus to limit the size of the weights. With the above methods, we can pick a model which performs the best on general datasets.

2.5 Compare k-NN and Neural Networks [8 points]

- **Try k-NN in this digit classification task using the code you developed in the first assignment. Compare the results with those you got using neural networks.**



Classification rates for k-nn classifier:

cr_Validation:

[0.9849999999999999, 0.9899999999999999, 0.9799999999999998, 0.9849999999999999, 0.9849999999999999]

cr_Test:

[0.9849999999999999, 0.9875000000000004, 0.9875000000000004, 0.9899999999999999, 0.9875000000000004]

The classification errors for the validation set is around 0.02, and the classification errors for the testing set is around 0.015, which are lower than the classification errors from neural networks in general. Therefore, K-nn has a better performance than NN on this data set.

- **Briefly comment on the differences between these classifiers.**

k-nn:

In k-Nearest Neighbors classification, we classify a new data point by computing the distances from it to k nearest labeled data points, where k is some odd number. The decision boundary for k-nn model is piecewise linear, thus globally non-linear. A new data point is classified into the class which has the most population among the k labels.

If the training set is too small, the model will include too many mislabeled neighbors for some large k, thus leads to misclassification. However, there is no such problem for large data sets.

Neural networks:

Neural networks uses composite of simple functions (can be linear or non-linear) to create complex non-linear functions. The decision boundary is more flexible than most of the other classifiers.

Comparison:

K-nn is implemented using the distance between the new data and the existing ones. NN includes all data points, including the noise. Therefore, NN is more sensitive to noise while K-nn performs better while there is a few noises. Therefore, as the performance of the output indicates, k-nn is more accurate in this data set comparing with NN.

3 Mixtures of Gaussians (45 points)

3.1 Code

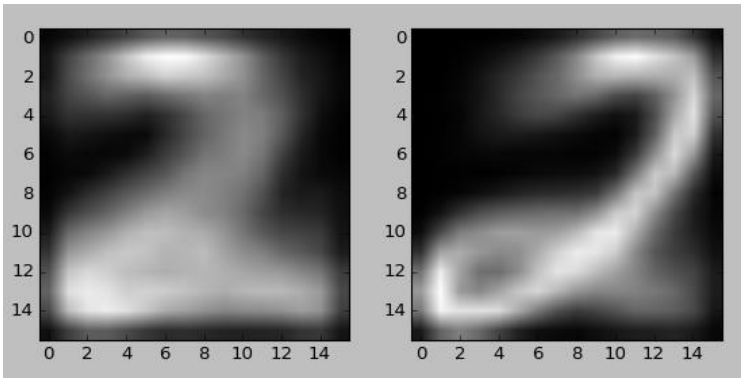
3.2 Training (15 points)

According to the assignment handout, the number of Gaussian clusters is set to 2, and the minimum variance is set to 0.01. As *randconst* goes larger, the probabilities for each Gaussians are closer. Because the *randconst* will dominate the latter term which is randomly selected. The number of iterations should be large enough for the model to be converge.

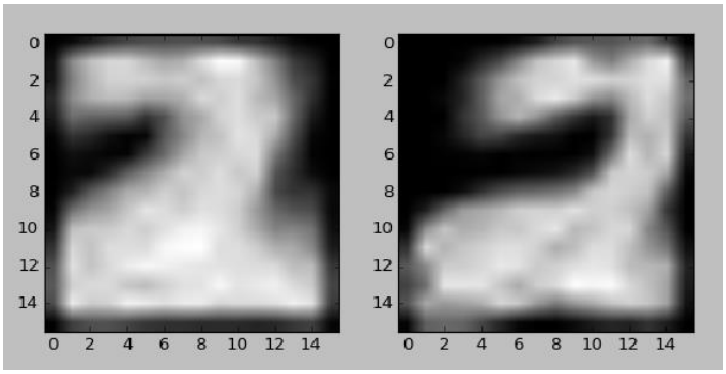
We need to find a model that gives the largest likelihood. Therefore, we pick the model to be:

Number of Gaussian clusters: **2**
Minimum variance: **0.01**
Number of iterations: 15
randConst: 1

- **Digit 2**
  - **Mean vectors:**



  - **Variance vectors:**

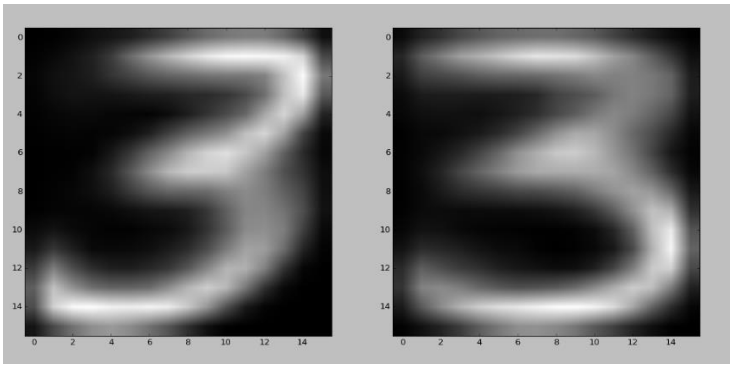


- **Mixing proportions for each clusters at the 15th iteration:**

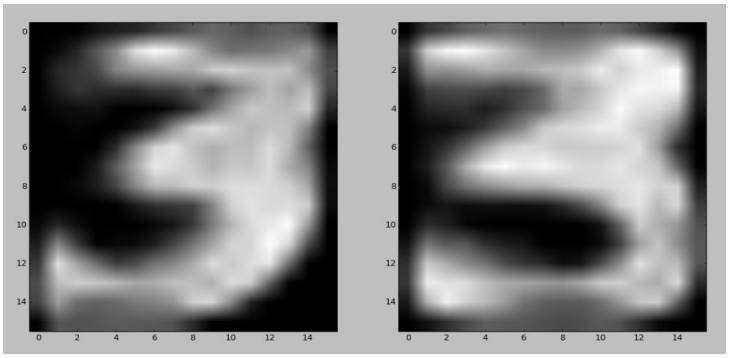
  [[0.50333398] [0.49666602]]

- $\boldsymbol{log\ P(Training\ Data) = -4044.95891}$ **at the 15th iteration.**

- **Digit 3**
  - **Mean vectors:**



  - **Variance vectors:**

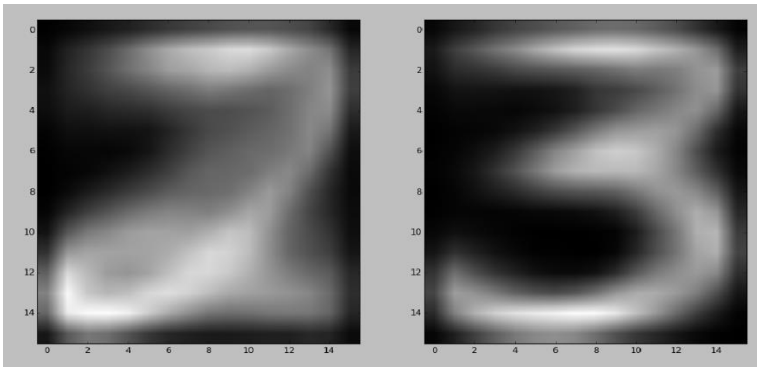


  - **Mixing proportions for each clusters at the 15th iteration:**

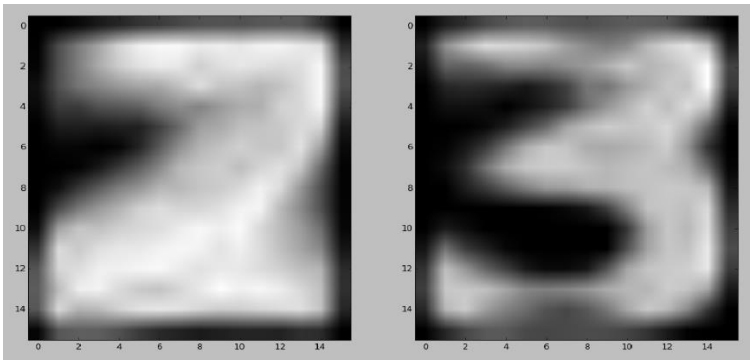
    [[0.45104682] [0.54895318]]

- $log\ P(Training\ Data) = 1989.48160$ **at the 15th iteration.**

- **Digits 2 & 3**
  - **Mean vectors:**



  - **Variance vectors:**



  - **Mixing proportions for each clusters at the 15th iteration:**

    [[0.59143768] [0.40856232]]

  - $log\ P(Training\ Data) = -16348.25814$ **at the 15th iteration.**

3.3 Initializing a mixture of Gaussian with k-means (10 points)

- **Mean vectors:**

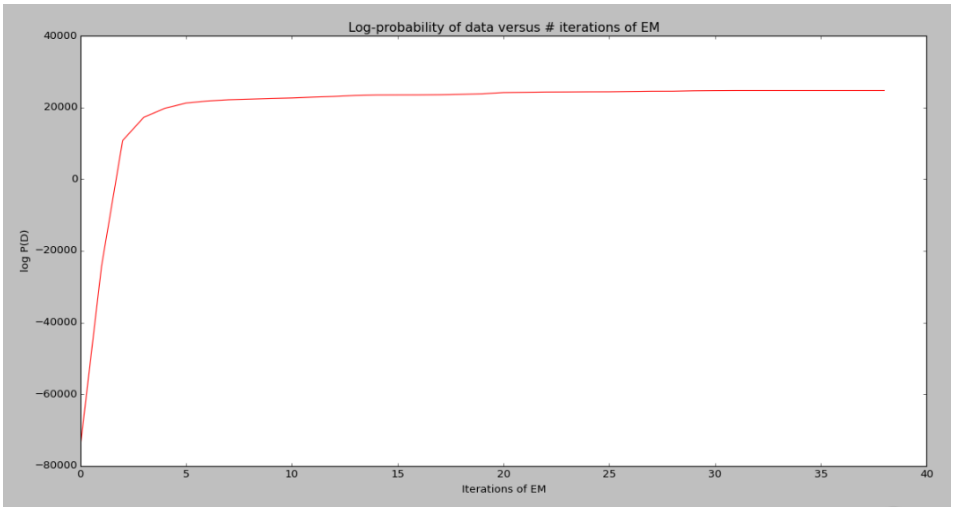
- **Variance vectors:**

- **Proportion of each cluster** (20 clusters) **at the 40th iteration:**

  [[0.02166667] [0.03333333] [0.03833333] [0.05669661] [0.03500184]
  [0.02666671] [0.05501463] [0.05666667] [0.04499971] [0.05333331]
  [0.05166659] [0.05999176] [0.0683334] [0.05330867] [0.09832281]
  [0.025] [0.06499802] [0.0399702] [0.05166667] [0.06502908]]

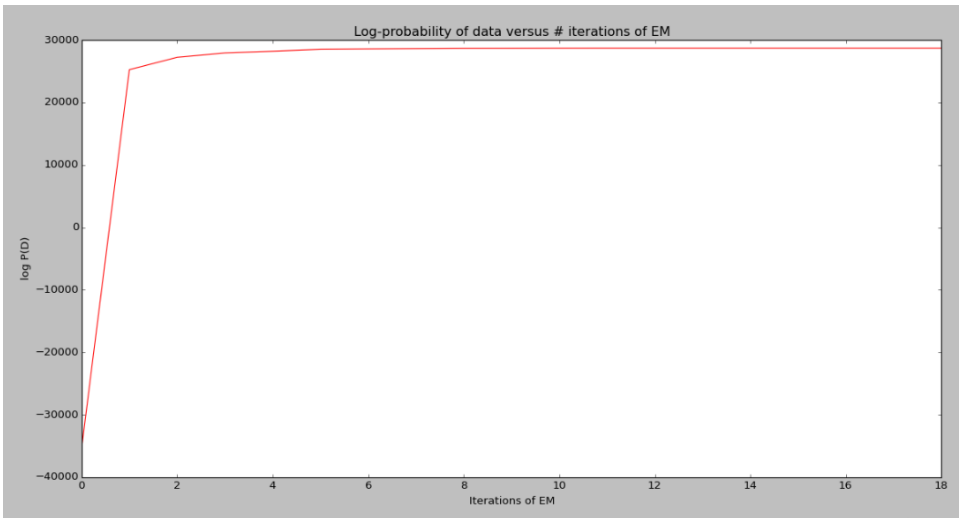
- **Speed of convergence**

  Original initialization method: (with 20 components on all 600 training vectors)



$$\boldsymbol{log\ P(Training\ Data)} = 24755.96477$$

  K-means initialization method:

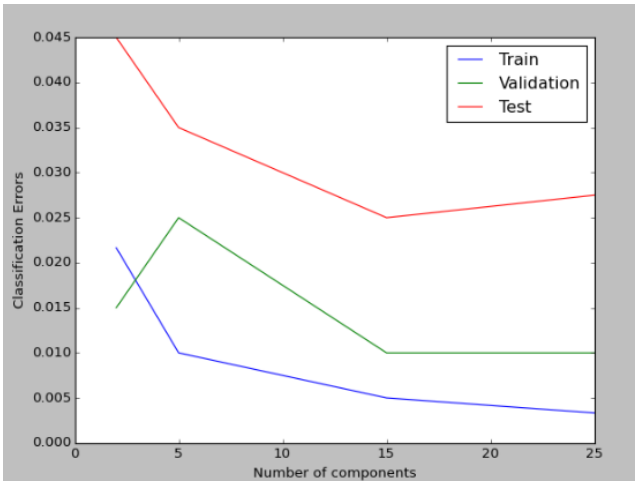


$$\boldsymbol{log\ P(Training\ Data)} = 27107.59756$$

It takes around 40 iterations for the model to converge using the original initialization method. It takes around 15 iterations for the model to converge by introducing K-means. The speed convergence of the initialization based on k-means is a lot faster.

3.4 Classification using MoGs (20 points)

- Classification errors versus number of mixture components:



- The error rates on the training sets generally decrease as the number of clusters increases. As there are more clusters, the data will be classified more accurately. Each cluster will have more specific “properties”, thus there will be less classification error.
- The error rate for the test set decreases initially and then bounce back up when the number of components keeps increasing. This is because the model is starting to over fit the training set. Therefore, it does not perform well on general data sets. Therefore, as we can see, both classification errors on validation set and test set tend to increase while there are more than 15 components.
- To choose a particular model, I would choose a model with the number of components that has a small (non-zero) classification error on training set and the errors on the validation set and testing set should be at their minimum point. The number of iterations should be large enough for the training set to converge and not too large to cause overfitting.

3.5 Bonus:

The best MoG classifier:

Number of iterations: 10

Number of components: 15

Minimum variance: 0.01

randConst: 1

Classification errors:

Training set: 0.0083333333333333332

Validation set: 0.014999999999999999

Testing set: 0.035000000000000003

Neural networks:

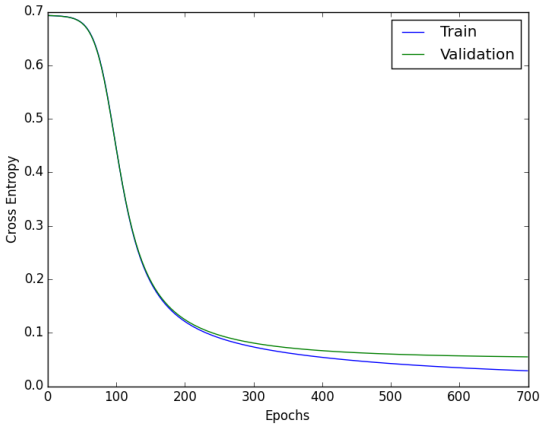
num_hiddens = 15

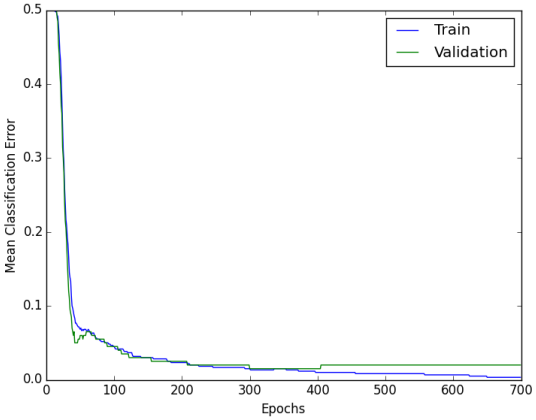
eps = 0.1

momentum = 0.5

num_epochs = 700

Classification errors:





Error: Train 0.02875 Validation 0.05484 Test 0.06771

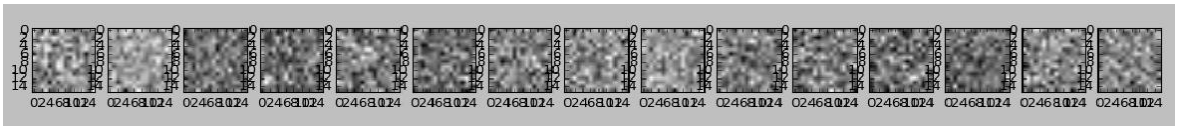
MCE: Train 0.00333 Validation 0.02000 Test 0.03000

Comparisons:

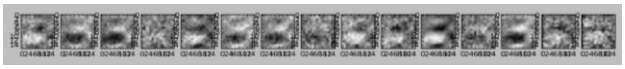
Neural networks and mixture of Gaussians performs equally well on this data set based on the classification errors. Mixture of Gaussian is an unsupervised learning method, which does not need a target set during the training. However, neural networks performs well when the target output is given.

Visualization of the input to hidden weights in neural networks:

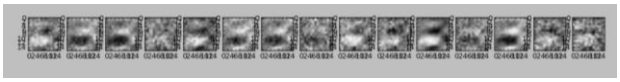
(at the 10th iteration)



(at the 100th iteration)



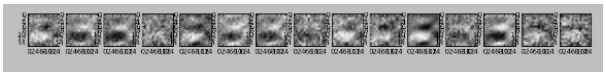
(at the 200th iteration)



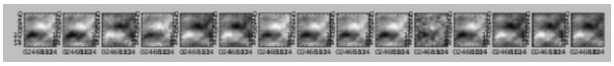
(at the 300th iteration)

(at the 400th iteration)

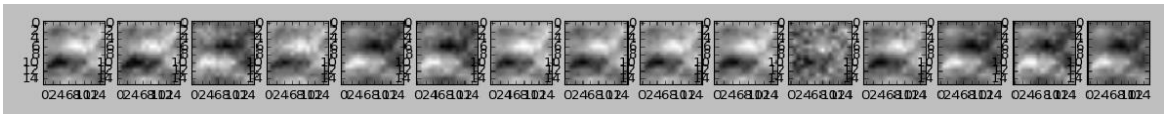
(at the 500th iteration)



(at the 600th iteration)



(at the 700th iteration)



Due to the limitation on uploading file size, the intermediate plots are shown in the program. Please run the program for detailed images.

As the training continues, a clearer partial 3 or 2 can be recognized from the images of hidden units. In the above image (at the 700th iteration), some parts are darker and some others are lighter. This indicates that each weight has being taking information locally from the digits. They will generate a better overall view when the training is complete.

In the mixture of Gaussians, the number of components really means how many Gaussian distributions are in the model. As the number is too large, the model will overfit by including all the noises. In neural network, the number of hidden units will also has the potential to over fit the model.