998082159

CSC411 A2 Q1

Leo Woiceshyn

E-Step: Find Responsibilities

$$p(X) = \sum_{k=1}^{K} \pi_k \mathcal{N}(X \mid \mu_k, \Sigma_k) \rightarrow \text{Given: } \Sigma \text{ is shared for all } k$$

Conditional Probability w/ Bayes' Rule

$$\gamma(z_k) = p(z_k = 1 \mid X) = \frac{p(z_k=1)\, p(X \mid z_k = 1)}{\sum_{j=1}^{K} p(z_j = 1)\, p(X \mid z_j = 1)}$$

$$\gamma(z_k) = \frac{\pi_k \mathcal{N}(X \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(X \mid \mu_j, \Sigma_j)} \quad (1)$$

Likelihood is therefore defined as:

$$L = p(X \mid \pi_k, \mu_k, \Sigma) = \prod_{n=1}^{N} \left[\sum_{k=1}^{K} \pi_k \mathcal{N}(\vec{x}^{(n)} \mid \mu_k, \Sigma) \right]$$

Ln likelihood for monotonically increasing function

$$\ln L = \sum_{n=1}^{N} \ln \left[\sum_{k=1}^{K} \pi_k \mathcal{N}(\vec{x}^{(n)} \mid \mu_k, \Sigma) \right]$$

$$\mathcal{N}(\vec{x} \mid \mu_k, \Sigma) = \frac{1}{\sqrt{(2\pi)^D |\Sigma|}} e^{\left(-\frac{1}{2}(\vec{x} - \mu_k)^T \Sigma^{-1} (\vec{x} - \mu_k)\right)}$$

For a similarity transform, $\frac{\partial (X^T A X)}{\partial X} = X^T (A + A^T)$

M-Step: Parameter Estimation

$$\frac{\partial \ln L}{\partial \mu_k} = \sum_{n=1}^{N} \frac{1}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\vec{x} \mid \mu_j, \Sigma)} \cdot \frac{\partial}{\partial \mu_k} \pi_k \mathcal{N}(\vec{x}^{(n)} \mid \mu_k, \Sigma)$$

$$\frac{\partial}{\partial \mu_k} \pi_k \mathcal{N}(\vec{x}^{(n)} \mid \mu_k, \Sigma) = \pi_k \frac{\partial}{\partial \mu_k} \frac{1}{\sqrt{(2\pi)^D |\Sigma|}} \exp\left(-\frac{1}{2}(\vec{x}^{(n)} - \mu_k)^T \Sigma^{-1} (\vec{x}^{(n)} - \mu_k)\right)$$

$$\text{So } \frac{\partial \ln L}{\partial \mu_k} = \sum_{n=1}^{N} \left[-\frac{1}{2} \frac{\pi_k / \sqrt{(2\pi)^D |\Sigma|}}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\vec{x} \mid \mu_j, \Sigma)} \exp\left(-\frac{1}{2}(\vec{x}^{(n)} - \mu_k)^T \Sigma^{-1} (\vec{x}^{(n)} - \mu_k)\right) \cdot \frac{d}{d\mu_k} \left[(\vec{x}^{(n)} - \mu_k)^T \Sigma^{-1} (\vec{x}^{(n)} - \mu_k) \right] \right]$$

$2\Sigma^{-1}$ since Σ is symmetric

$$\frac{\partial}{\partial \mu_k}\left[(\vec{x}^{(n)}-\mu_k)^T\Sigma^{-1}(\vec{x}^{(n)}-\mu_k)\right] = (\vec{x}^{(n)}-\mu_k)^T\left(\Sigma^{-1}+(\Sigma^{-1})^T\right)\cdot(-1)$$

Substituting back into eqn: (-2 and $-\frac{1}{2}$ cancel out)

$$\frac{\partial \ln l}{\partial \mu_k} = \sum_{n=1}^{N}\left[\underbrace{\frac{\pi_k \mathcal{N}(\vec{x}^{(n)}\mid \mu_k,\Sigma)}{\sum_{j=1}^{k}\pi_j \mathcal{N}(\vec{x}^{(n)}\mid \mu_j,\Sigma)}}_{\gamma(z_k^{(n)})}\cdot(\vec{x}^{(n)}-\mu_k)^T\Sigma^{-1}\right]$$

$$= \sum_{n=1}^{N}\left[\gamma(z_k^{(n)})\underbrace{(\vec{x}^{(n)}-\mu_k)^T\Sigma^{-1}}\right]$$

Equivalent to $(\Sigma^{-1})^T(\vec{x}^{(n)}-\mu_k) = \Sigma^{-1}(\vec{x}^{(n)}-\mu_k)$ since Σ^{-1} symmetric

$$= \sum_{n=1}^{N}\left[\gamma(z_k^{(n)})\Sigma^{-1}(\vec{x}^{(n)}-\mu_k)\right]$$

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

Set to 0:

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

$$\text{Ls } \mu_k = \frac{\sum_{n=1}^{N}\gamma(z_k^{(n)})\vec{x}^{(n)}}{\sum_{n=1}^{N}\gamma_k^{(n)}}$$

→ Let $N_k = \sum_{n=1}^{N}\gamma_k^{(n)}$, the number of data points from Gaussian k.

$$\boxed{\mu_k = \frac{1}{N_k}\sum_{n=1}^{N}\gamma(z_k^{(n)})\vec{x}^{(n)}}$$

→ See full version of $\gamma(z_k^{(n)})$ on previous page

Covariance estimation for M-Step:

$$\frac{\partial \ln L}{\partial \Sigma} = \frac{\partial}{\partial \Sigma} \sum_{n=1}^{N} \ln\left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\vec{x}^{(n)} | \mu_k, \Sigma) \right)$$

$$= \sum_{n=1}^{N} \frac{\pi_k}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\vec{x}^{(n)} | \mu_j, \Sigma)} \cdot \frac{\partial}{\partial \Sigma} \mathcal{N}(\vec{x}^{(n)} | \mu_k, \Sigma)$$

$$\frac{\partial}{\partial \Sigma} \mathcal{N}(\vec{x}^{(n)} | \mu_k, \Sigma) = \frac{\partial}{\partial \Sigma} \frac{1}{2\pi\sqrt{|\Sigma|}} \exp\left(-\frac{1}{2} (\vec{x}^{(n)} - \mu_k)^T \Sigma^{-1} (\vec{x}^{(n)} - \mu_k) \right)$$

Product Rule:

$$= \frac{\partial}{\partial \Sigma} \left(\frac{1}{2\pi\sqrt{|\Sigma|}} \right) \cdot \exp\left(-\frac{1}{2} (\vec{x}^{(n)} - \mu_k)^T \Sigma^{-1} (\vec{x}^{(n)} - \mu_k) \right)$$

$$+ \frac{\exp\left(-\frac{1}{2} (\vec{x}^{(n)} - \mu_k)^T \Sigma^{-1} (\vec{x}^{(n)} - \mu_k) \right)}{2\pi\sqrt{|\Sigma|}} \cdot \frac{\partial}{\partial \Sigma} \left(-\frac{1}{2} (\vec{x}^{(n)} - \mu_k)^T \Sigma^{-1} (\vec{x}^{(n)} - \mu_k) \right)$$

$$= \frac{1}{2\pi} \cdot \left(-\frac{1}{2} \right) \cdot \left(|\Sigma| \right)^{-1/2} \left(\Sigma^{-1} \right)^T \exp\left(-\frac{1}{2} (\vec{x}^{(n)} - \mu_k)^T \Sigma^{-1} (\vec{x}^{(n)} - \mu_k) \right)$$

$$+ \mathcal{N}(\vec{x}^{(n)} | \mu_k, \Sigma) \cdot \frac{1}{2} \left(\Sigma^{-1} (\vec{x}^{(n)} - \mu_k)(\vec{x}^{(n)} - \mu_k)^T \Sigma^{-1} \right)^T$$

$$= \frac{\left(-\frac{1}{2} \right) \left(\Sigma^{-1} \right)^T}{2\pi\sqrt{|\Sigma|}} \exp\left(-\frac{1}{2} (\vec{x}^{(n)} - \mu_k)^T \Sigma^{-1} (\vec{x}^{(n)} - \mu_k) \right)$$

$$+ \mathcal{N}(\vec{x}^{(n)} | \mu_k, \Sigma) \cdot \frac{1}{2} \left(\Sigma^{-1} (\vec{x}^{(n)} - \mu_k)(\vec{x}^{(n)} - \mu_k)^T \Sigma^{-1} \right)^T$$

Factoring out $-\frac{1}{2}\mathcal{N}(\vec{x}^{(n)} | \mu_k, \Sigma)$:

$$= -\frac{1}{2} \mathcal{N}(\vec{x}^{(n)} | \mu_k, \Sigma) \left[\left(\Sigma^{-1} \right)^T - \left(\Sigma^{-1} (\vec{x}^{(n)} - \mu_k)(\vec{x}^{(n)} - \mu_k)^T \Sigma^{-1} \right)^T \right]$$

Subbing back into original expression:

$$\frac{\partial \ln L}{\partial \Sigma} = \sum_{n=1}^{N} \underbrace{\frac{\pi_k \mathcal{N}(\vec{x}^{(n)} | \mu_k, \Sigma)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\vec{x}^{(n)} | \mu_j, \Sigma)}}_{\gamma(z_k^{(n)})} \cdot \left(\frac{-1}{2}\right)\left[(\Sigma^{-1})^T - \left(\Sigma^{-1}(\vec{x}^{(n)} - \mu_k)(\vec{x}^{(n)} - \mu_k)^T \Sigma^{-1}\right)^T\right]$$

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

Set to zero:

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

Transpose both sides: ($(\Sigma^{-1})^T = \Sigma^{-1}$ due to symmetry)

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

Pre-multiply both sides by Σ:

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

$$\boxed{\Sigma = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_k^{(n)})(\vec{x}^{(n)} - \mu_k)(\vec{x}^{(n)} - \mu_k)^T}$$

where $N_k = \sum_{n=1}^{N} \gamma(z_k^{(n)})$

Mixture Parameter Estimation for M-Step: (Using Lagrange Multiplier)

$$\frac{\partial \ln L}{\partial \pi_k} = \frac{\partial}{\partial \pi_k} \sum_{n=1}^{N} \ln \sum_{k=1}^{K} \pi_k \mathcal{N}(\vec{x}^{(n)} | \mu_k, \Sigma) + \lambda \left(1 - \sum_k \pi_k\right)$$

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

$$= \sum_{n=1}^{N} \frac{\mathcal{N}(\vec{x}^{(n)} | \mu_k, \Sigma)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\vec{x}^{(n)} | \mu_j, \Sigma)} + \frac{\partial}{\partial \pi_k} \lambda \left(1 - \sum_k \pi_k\right)$$

Multiply both sides by π_k

$$= \sum_{n=1}^{N} \frac{\mathcal{N}(\vec{x}^{(n)} | \mu_k, \Sigma)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\vec{x}^{(n)} | \mu_j, \Sigma)} + \lambda = 0 \quad \times \pi_k$$

$$= \sum_{n=1}^{N} \underbrace{\frac{\pi_k \mathcal{N}(\vec{x}^{(n)} | \mu_k, \Sigma)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\vec{x}^{(n)} | \mu_j, \Sigma)}}_{\gamma(z_k^{(n)})} + \pi_k \lambda$$

Set to zero:

$$\pi_k \lambda = -\sum_{n=1}^{N} \gamma(z_k^{(n)})$$

$$\pi_k \lambda = -N_k$$

Sum over k's:

$$\sum_{k=1}^{K} \pi_k \lambda = -\sum_{k=1}^{K} \sum_{n=1}^{N} \gamma(z_k^{(n)})$$

$$\lambda = -N$$

$$\boxed{\pi_k = \frac{-N_k}{N} = \frac{N_k}{N}}$$

CSC2515 A2Q2

998082159
Leo Woiceshyn

Assume $n=1, c=1, k=1$

For equation Chain rule:

(6)

$$\left(\frac{\partial E}{\partial f}\right)_{i,j} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial f}$$

In this case:

$$y_{h',w'} = x * f = \sum_{i=1}^{I} \sum_{j=1}^{J} x_{h'+i-1, w'+j-1} \cdot f_{i,j}$$

$$\frac{\partial y}{\partial f} = \sum_{i=1}^{I} \sum_{j=1}^{J} x_{h'+i-1, w'+j-1}$$

Given the padding bounds as $1 \le h' \le H-I+1$, and $1 \le w' \le W-J+1$ can rewrite $\frac{\partial y}{\partial f}$ as follows:

$$\frac{\partial y}{\partial f} = \sum_{a=1}^{H-I+1} \sum_{b=1}^{W-J+1} x_{i+a-1, j+b-1}$$

Therefore, $$\frac{\partial E}{\partial f} = \frac{\partial E}{\partial y} \left[\sum_{a=1}^{H-I+1} \sum_{b=1}^{W-J+1} x_{i+a-1, j+b-1} \right]$$

$$\frac{\partial E}{\partial f} = \sum_{a=1}^{H-I+1} \sum_{b=1}^{W-J+1} \left[x_{i+a-1, j+b-1} \frac{\partial E}{\partial y} \right]$$

∴ By definition of convolution operator:

$$\boxed{\frac{\partial E}{\partial f} = x_{h,w}^{(I-1, J-1)} * \frac{\partial E}{\partial y}}$$

for equation (7)

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial x}$$ can be represented as $x^T f$

$$y_{h',w'} = \sum_{i=1}^{I} \sum_{j=1}^{J} x_{i,h'+i-1,w'+j-1} * f_{i,j}$$

Partials with respect to x^T must be of same dim

$$\frac{\partial y_{h',w'}}{\partial x^T} = \sum_{i=1}^{I} \sum_{j=1}^{J} f^T_{i,j}$$

Applying the padding bounds: $1 \le h' \le H-I+1$, $1 \le w' \le W-J+1$

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} \sum_{a=1}^{H-I+1} \sum_{b=1}^{W-J+1} f^T_{i,j}$$

$$\frac{\partial E}{\partial x} = \sum_{a=1}^{H-I+1} \sum_{b=1}^{W-J+1} \frac{\partial E}{\partial y} f^T_{i,j}$$

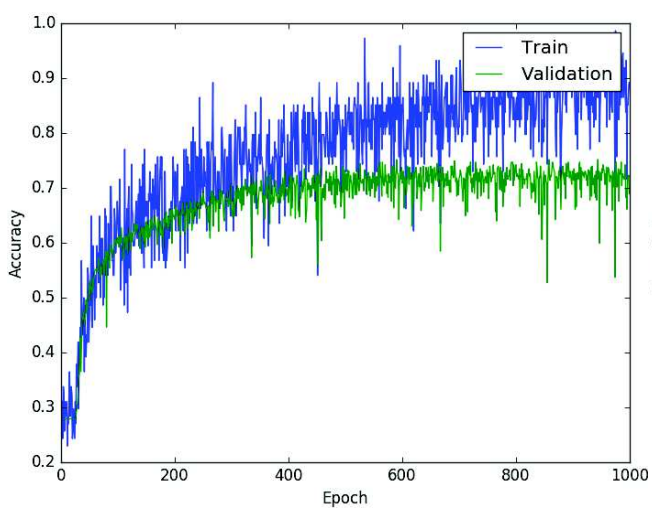
From convolution definition:

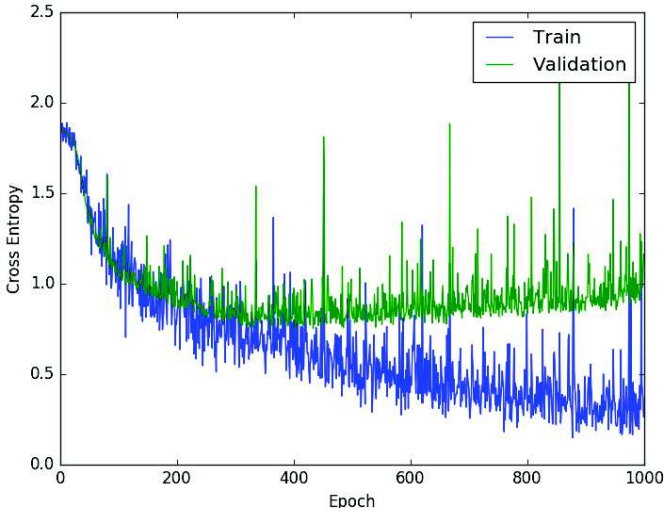
$$\boxed{\frac{\partial E}{\partial x} = \left(\frac{\partial E}{\partial y}\right)^{(I-1,J-1)} * f^T_{i,j}}$$

CSC2515 Assignment 2 Question 3 -Leo Woiceshyn (998082159)

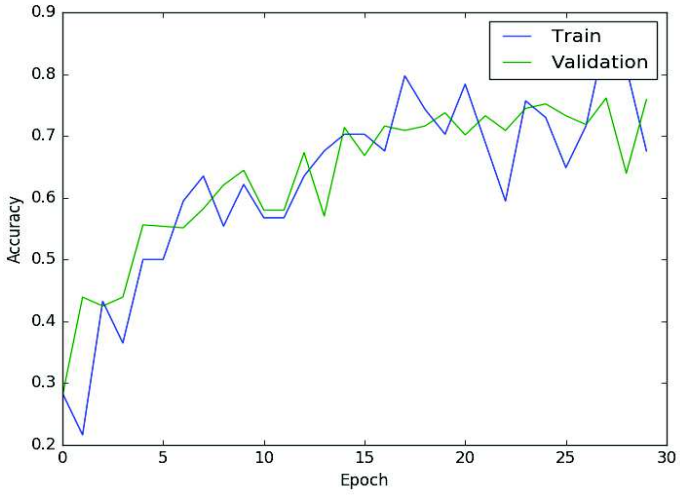
3.1 -Basic Generalization

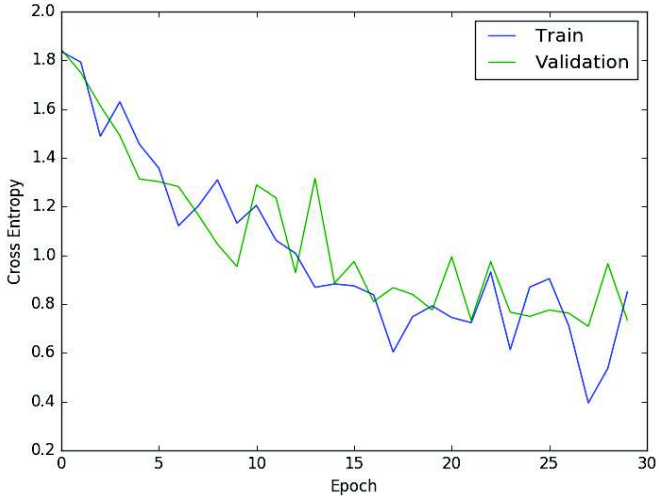
Neural Network (1000 Epochs)





Convolutional Neural Network (30 Epochs)





For the regular neural network, based on the cross-entropy and accuracy charts for the default hyperparameters, the validation accuracy stays consistently lower on average (and the cross entropy consistently higher), during the training process.

For the convolutional neural network, since the number of epochs is considerably lower, it's harder to see this trend. The validation cross entropy seems to be consistently higher after the first 10 epochs, but for the accuracy, the training accuracy is quite sporadic and the expected trend between the two is harder to distinguish.

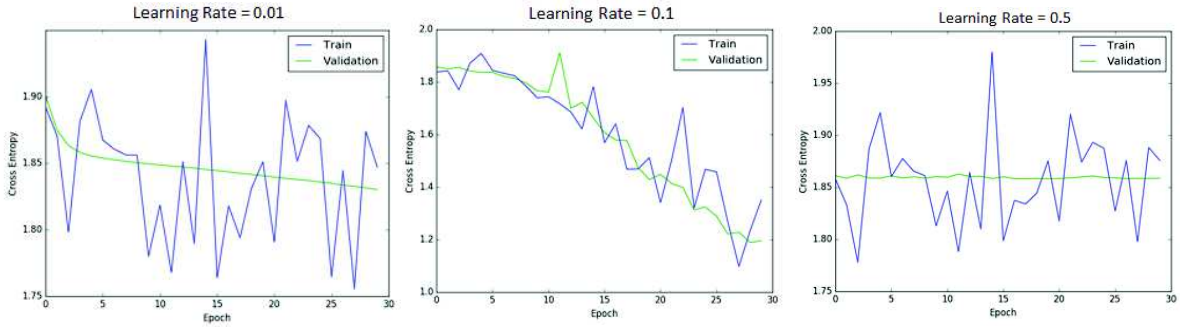
For a significantly lower number of epochs, the CNN resulted in a higher accuracy on the validation and test sets, appearing to reach convergence at an order of magnitude less number of epochs needed.

3.2 – Optimization

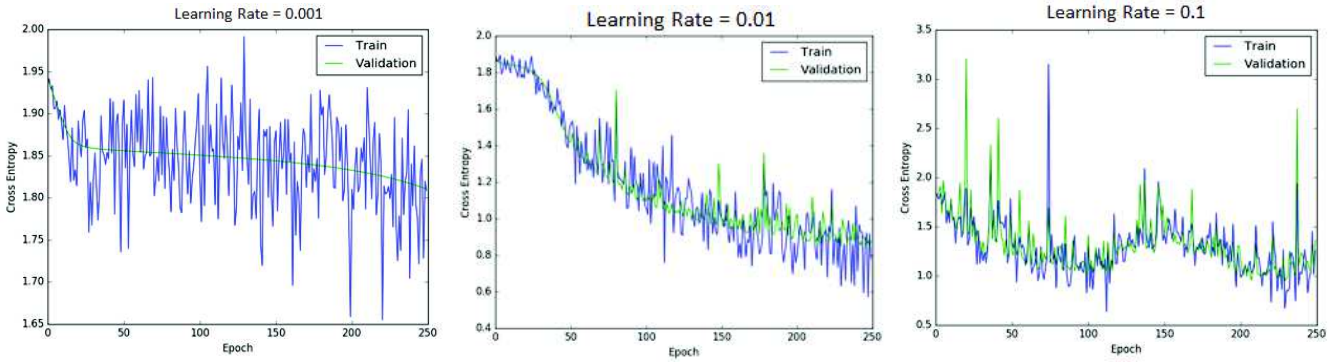
Note: I decided to use 250 epochs to compare my fully-connected NNs, as I found that these graphs showed the expected trends while taking significantly less time for me to generate than using 1000 epochs.

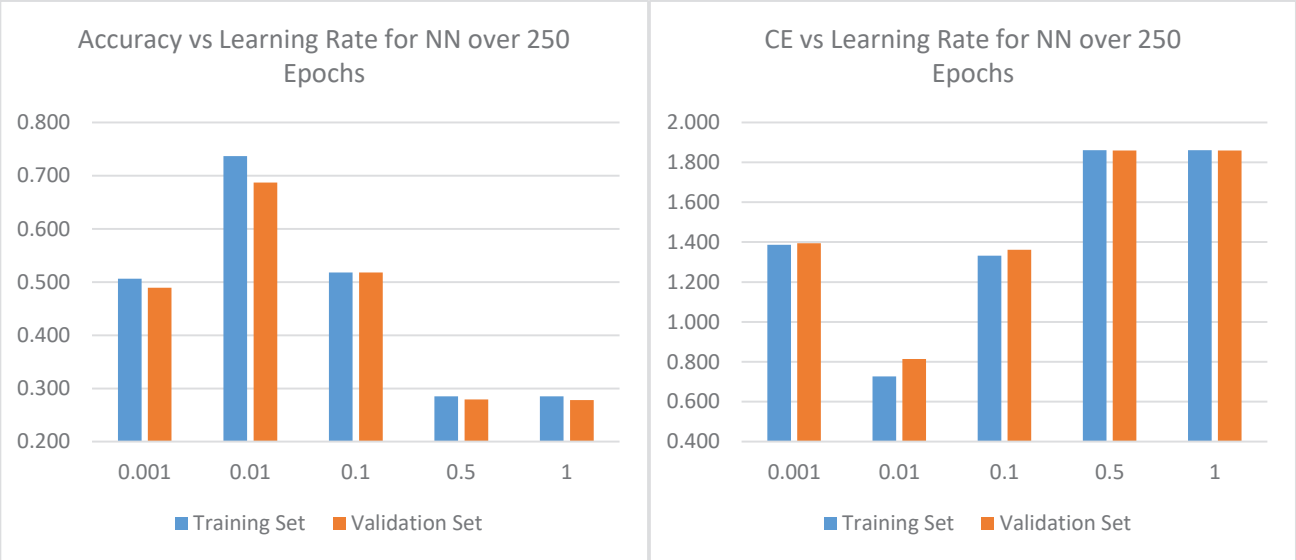
3.2.1 - Varying the Learning Rate (values I used were 0.001, 0.01, 0.1, 0.5, and 1)

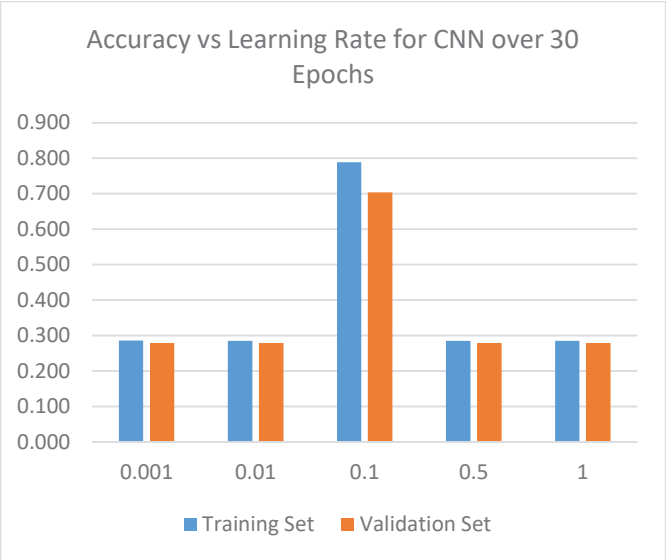
CNN:

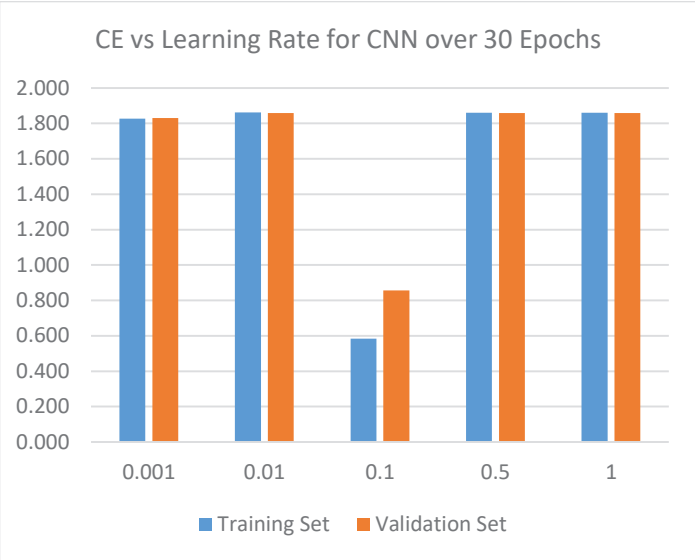


NN:







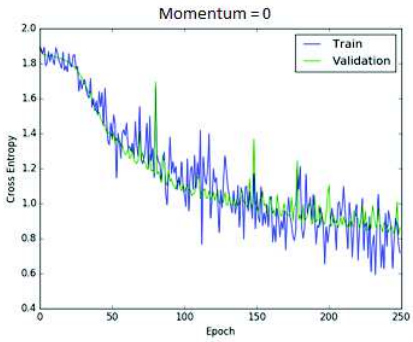


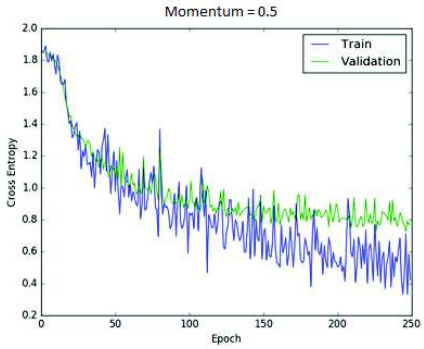
For the regular neural network, varying the learning rate to values other than the default parameter of 0.01 causes the network to converge slower, meaning that the cross-entropy values for the other parameters is higher, and the accuracies are lower. For this network, the learning rates that were closer to the default (0.001 and 0.1) gave better results than the further two learning rates (0.5 and 1).

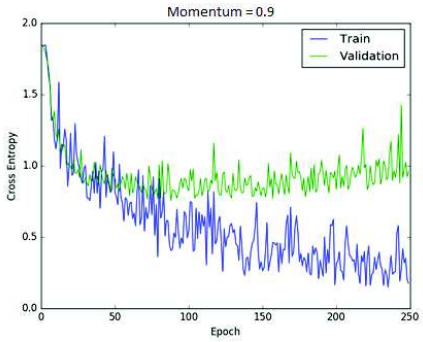
For the convolutional neural network, each of the other learning rates other than the default had very poor accuracies and high cross-entropies when compared to the 0.1 learning rate. Unlike the regular neural network, the learning rates that were closer to 0.1 (0.01 and 0.5) did not have noticeably better results than the learning rates which were further away (0.001 and 1).

3.2.2 – Varying the Momentum (Values I used were 0, 0.5, and 0.9)

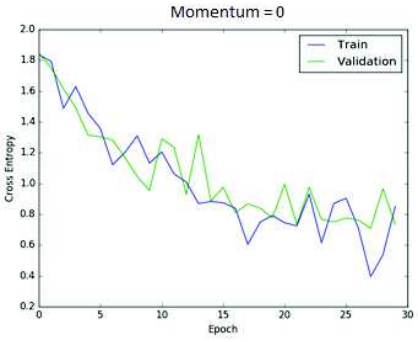
NN:

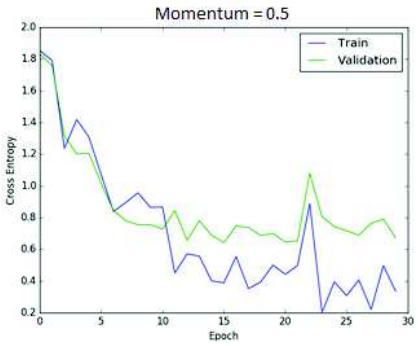


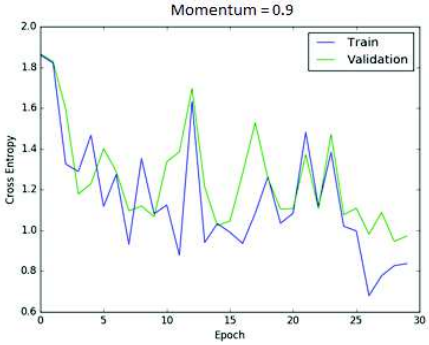


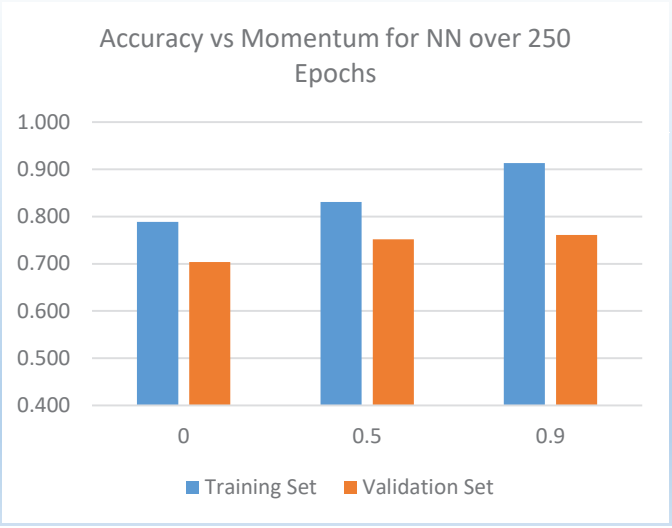


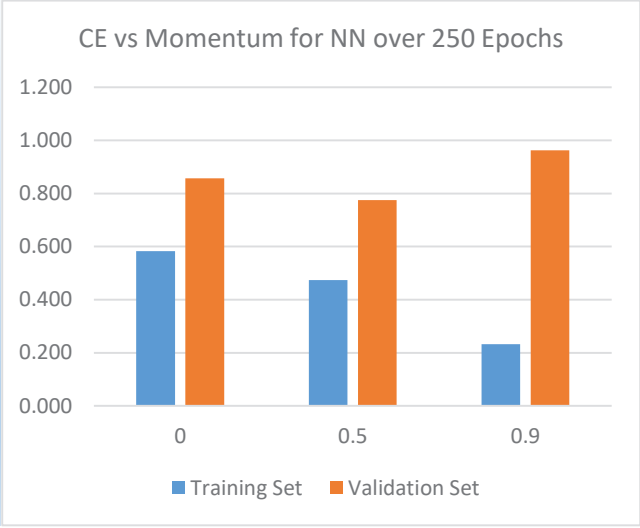
CNN:

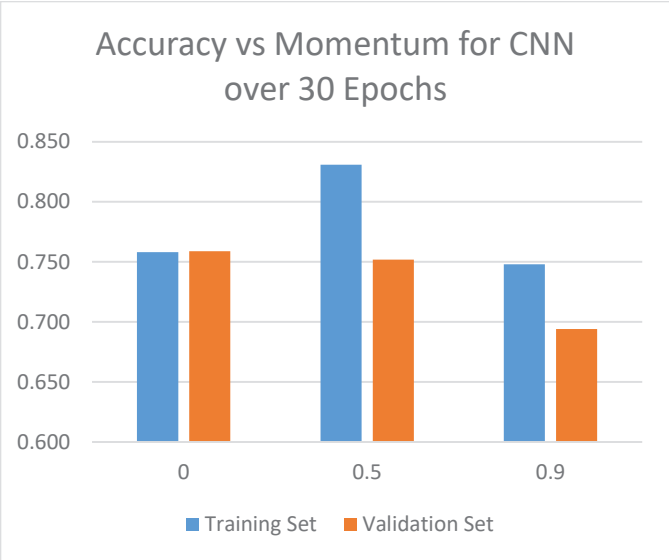


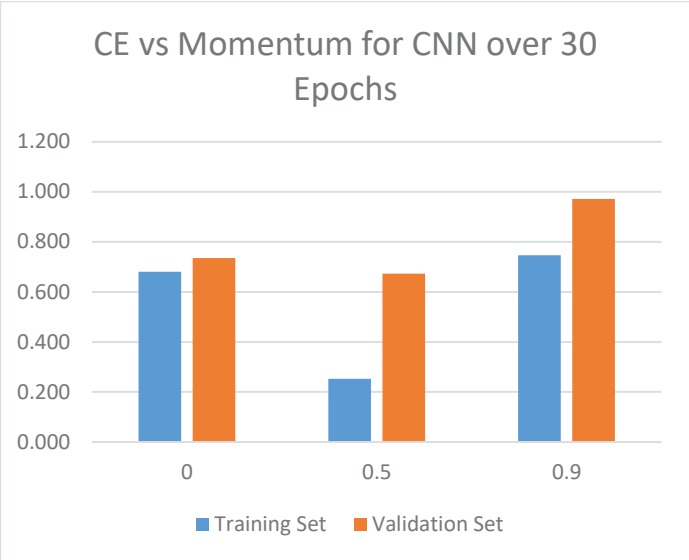




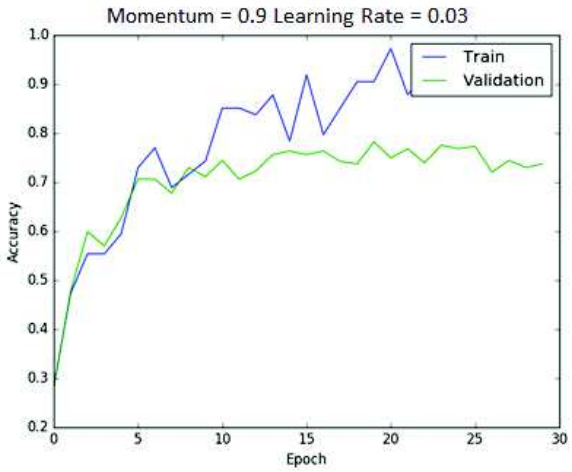


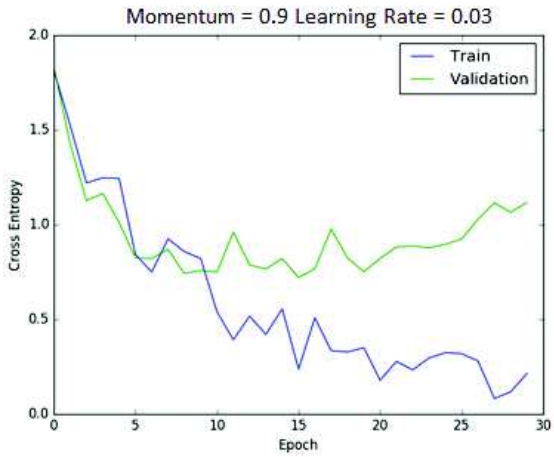






For both the regular and convolutional neural network, increasing the momentum increases how quickly the network converges. The momentum parameter of 0.5 seems to perform the best; although it doesn't begin converging as quickly as for 0.9, it has less oscillation and ends up converging faster than for a value of 0.5. Overall, a 0.5 value for momentum both converges faster than not having any momentum and yields roughly the same accuracy and cross entropy after the same number of epochs. The 0.9 momentum initially converges faster, but does not seem to outperform even the case of not having any momentum, when considering the cross entropy and accuracy over a consistent number of epochs. I suspect that the learning rate would need to be adjusted here to get better results for the momentum trials, as I'm holding it constant at the default values to see the effect of varying it in isolation. To test this intuition, here is the CNN run again for a momentum of 0.9, but with a learning rate of 0.03 instead of 0.1:

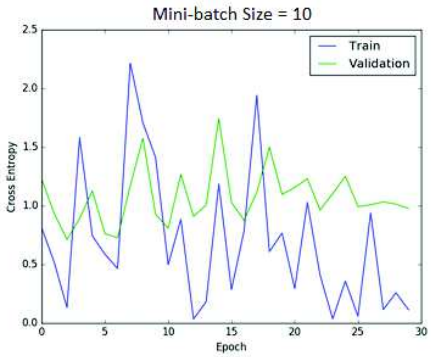


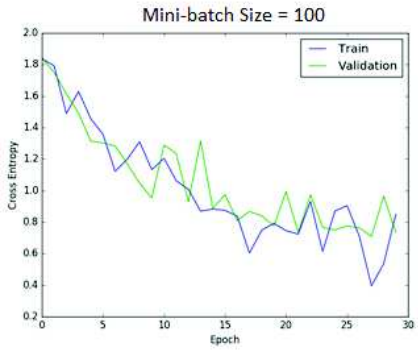


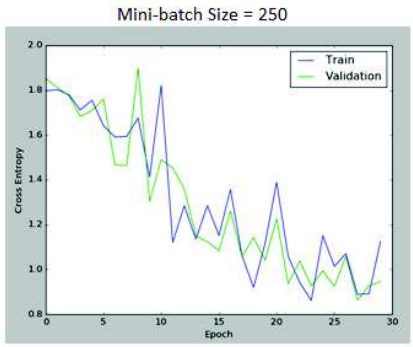
As expected, the lower learning rate was important when using the higher momentum value. Now we can see the reduction in large fluctuations in cross-entropy as we would expect, and a higher final validation accuracy than we got for the default learning rate of 0.1.

3.2.3 – Varying the Mini-batch Size (Values I used were 1, 10, 100, 250, and 1000)

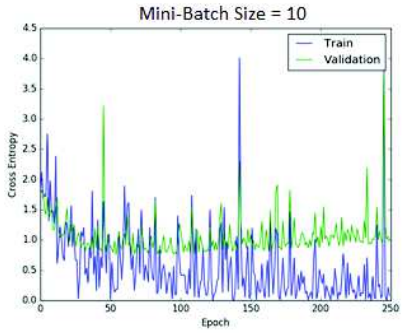
CNN:

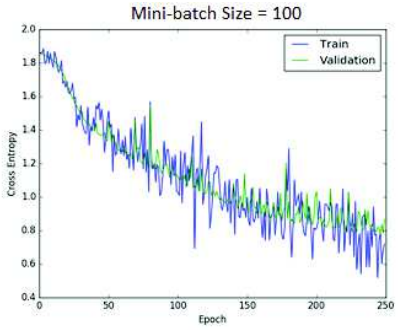


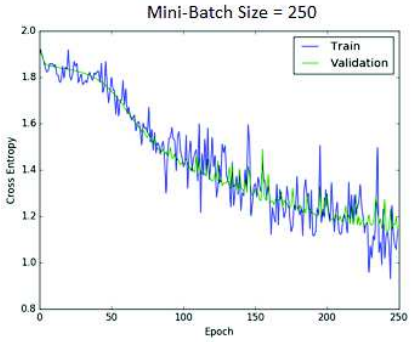


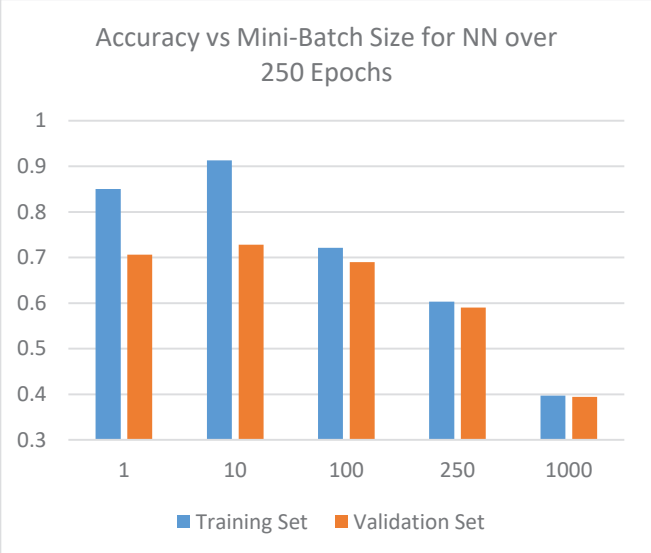


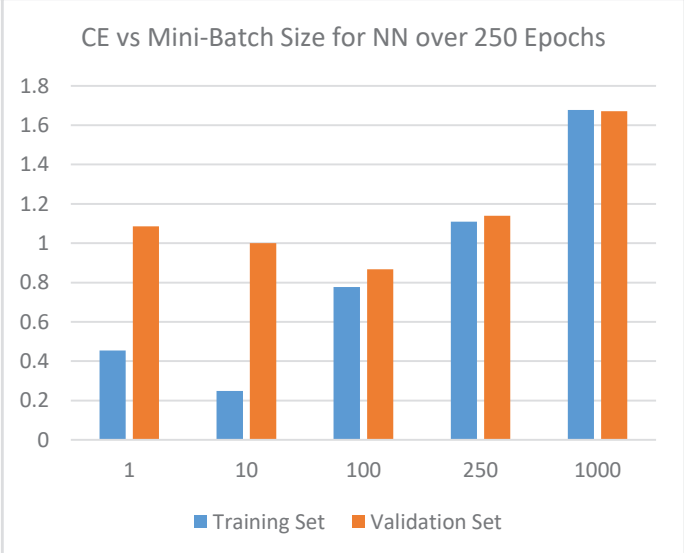
NN:

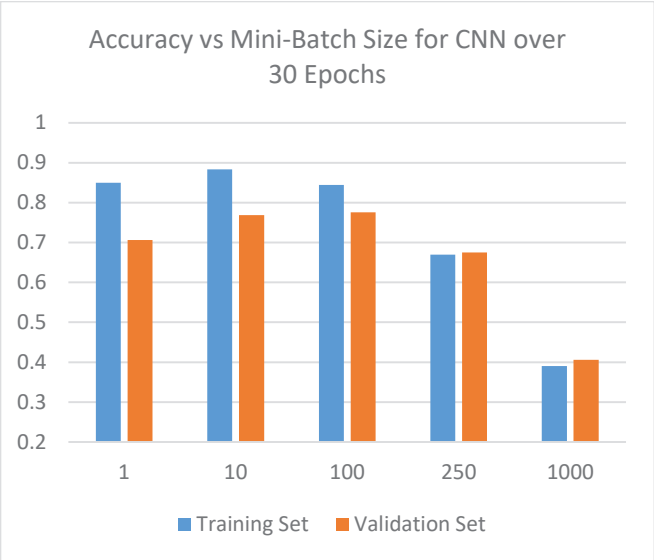


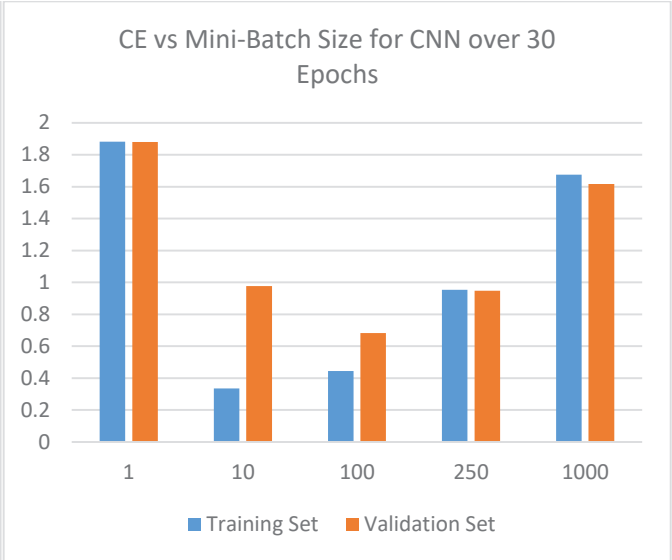












For the both the networks, the default mini-batch parameter size of 100 gives the best results for cross-entropy. However, for the regular neural-network, the validation accuracy is slightly higher for the smaller mini batch sizes. We can see that the smaller amount of mini-batch sizes have a significant margin of difference between the training set and the validation set, meaning that a smaller mini-batch size leads to poorer generalization, whereas the for the larger mini-batch sizes, the validation and training set values are around the same, meaning that increasing the mini-batch size reduces overfitting. The chosen default parameter of 100 seems to be a good choice, as it performs well for both networks in terms of accuracy, convergence speed, and generalization.

Conclusions regarding optimization:

To pick the best values for each of my parameters, I would do something similar to what I did for this assignment; hold the rest of the parameters constant while varying the parameter of interest and choose the value that seems to give the best results during cross-validation. I would also use my intuition about the parameters when I'm getting unexpected results, as I did for the momentum; if I only took the results I got it might seem that momentum does not improve performance, but this was simply due to the learning rate needing to be lower for a highest momentum values.

Once I had some values of the parameters which I thought were good, I would try difference combinations of them on the validation set to further justify my decisions.

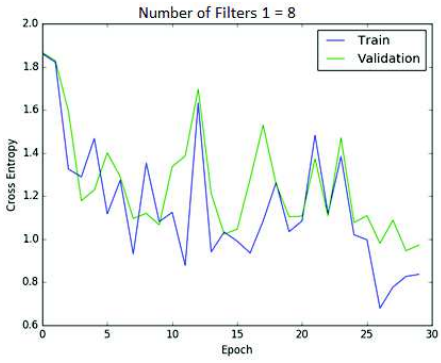
3.3 – Model Architecture

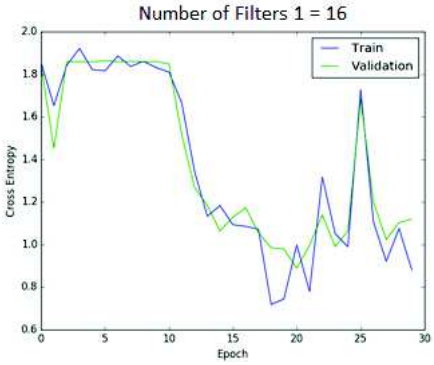
For this section, I kept either the number of nodes in the first layer for the NN, or the number of filters in the second layer, constant, and varied the number of nodes/filters in the second layer. Vice versa, I held the number of nodes/filters in the first layer constant and varied the number of nodes/filters in the second layer for each network. The values I ended up using were the following (First Layer Amount, Second Layer Amount), while keeping the momentum at 0.9 and the rest of the parameters as default:

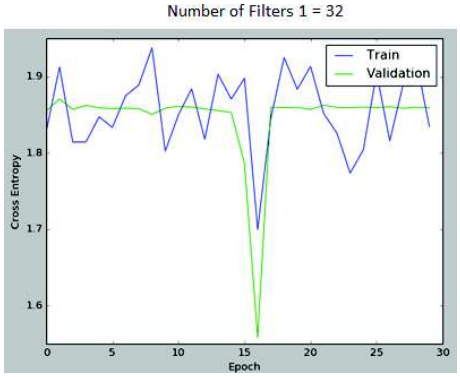
CNN: (8, 16), (16, 16), (32, 16), (8, 8), (8, 16), (8, 32)

NN: (16, 8), (16, 32), (16, 64), (8, 32), (32, 32), (64, 32)

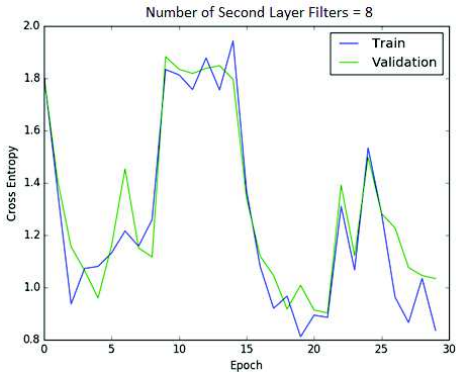
CNN First Layer Filters (Second Layer Filters=16):

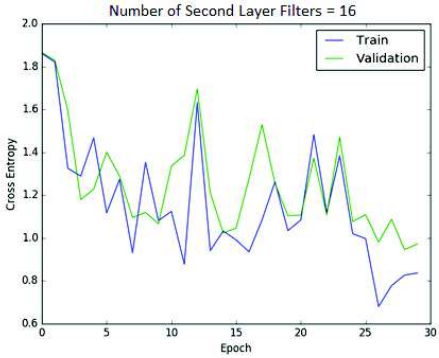


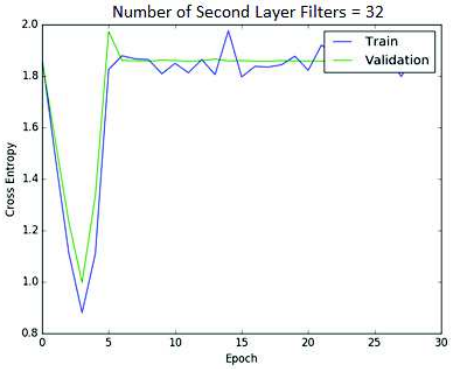




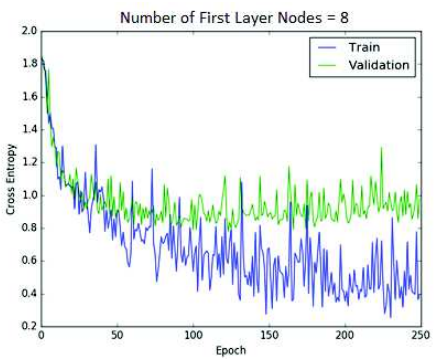
CNN Second Layer Filters (First Layer Filters=8):

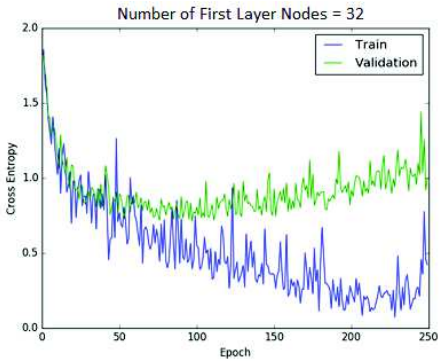


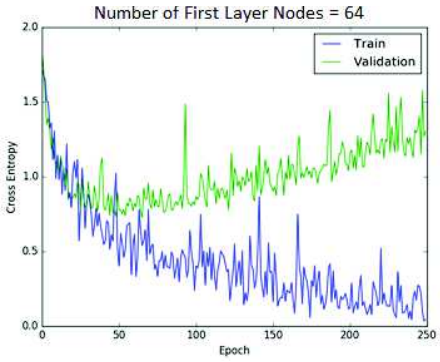




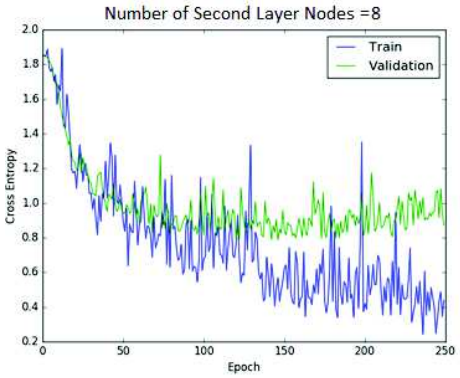
NN First Layer Nodes (Second Layer = 32 Nodes):

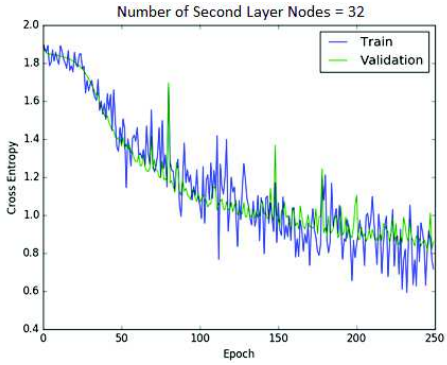


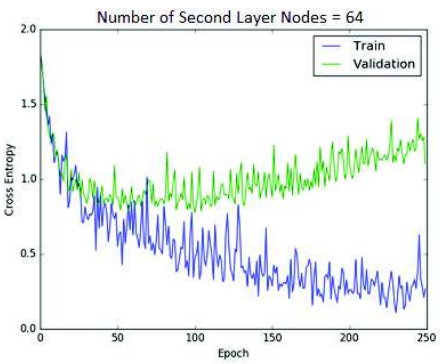




NN Second Layer Nodes (First Layer = 16 Nodes):







Based on the plots for varying each of the hidden layer and number of filters for the two networks, we can clearly see the effects that these have on the convergence and generalization of our network. For the fully-connected network, increasing the size of the first layer does not seem to change the convergence significantly, but we can see that the gap between the training and validation set curves increases, showing that an increased number of nodes leads to worse overall generalization. For the second layer, both having too low number of nodes and too high number of nodes led to both faster initial convergence but poorer generalization. In general, when for a high number of nodes in each respective layer, increasing the number of epochs led to even poorer generalization, as the training cross entropy continually got lower, while the validation cross-entropy kept rising.

For the CNN plots, changing the amounts of filters led to very strange, sporadic, results, where the cross entropy seemed to suddenly, drastically change. For the first layer, the network did not even converge for a high number of filters. For the first layer value of 16, and the second layer value of 8, the network seems to be converging, but with massive fluctuations, and would likely end up converging if more epochs were used. In general, increasing the number of filters seems to prevent the network from converging at all, whereas lower number of filters speed up convergence, but seem more unstable during the optimization process.

3.4 – Comparing CNN to fully-connected NN

Default Parameters:

NN: 3 Layer network, with 2 hidden layers. Number of inputs: 2304, First Hidden Layer: 16, Second Hidden Layer: 8, Number of outputs: 7.

First weight matrix and biases: $2304\ x\ 16\ Weights + 16\ Biases\ = 36880$

Second weight matrix and biases: $16\ x\ 8 + 8\ Biases = \ 136$

Third weight matrix and biases: $8\ x\ 7 + 7\ Biases = \ 63$

Total number of parameters for NN: 37079

CNN: Number of Channels: 1, Filter Size: 5 Number of Filters in First Layer: 8, Number of Filters in Second Layer: 16, Number of outputs: 7.

First weight matrix and biases: $5\ x\ 5\ x\ 1\ x\ 8\ Weights + 8\ Biases = 208$

Second weight matrix and biases: $5\ x\ 5\ x\ 8\ x\ 16\ Weights + 16\ Biases = 3216$

Third weight matrix and biases: $16\ x\ 64\ x\ 7\ Weights + 7\ Biases = \ 7175$

Total Number of Parameters for CNN: 10599

If our objective is to have a similar number of parameters for both networks, since our CNN is already running slowly, reduce the number of nodes in the hidden layers of the fully-completed network to make it close to the number of CNN parameters:

Choose: First Hidden Layer: 4, Second Hidden Layer: 64

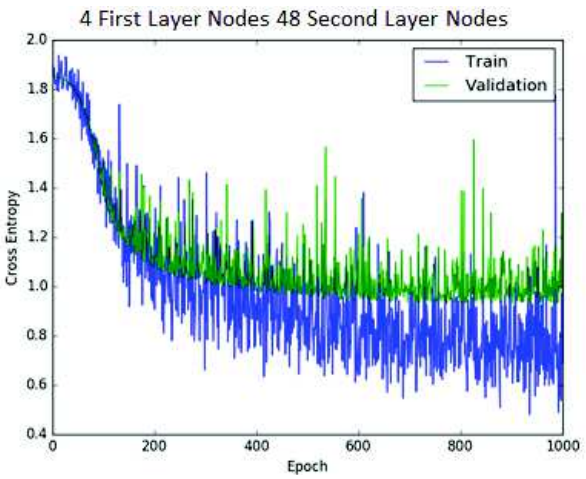
First weight matrix and biases: $2304\ x\ 4\ Weights + 16\ Biases\ = 9232$

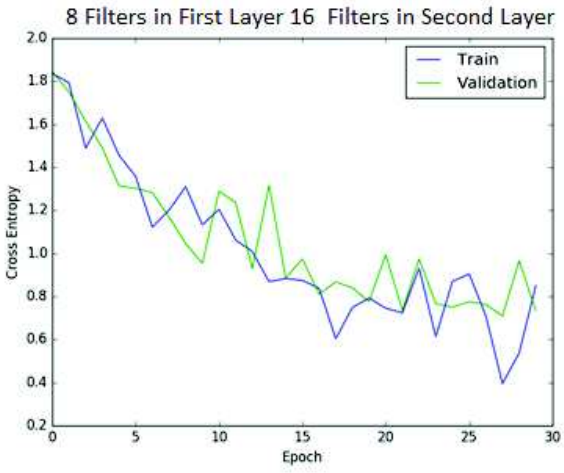
Second weight matrix and biases: $16\ x\ 48 + 8\ Biases = \ 776$

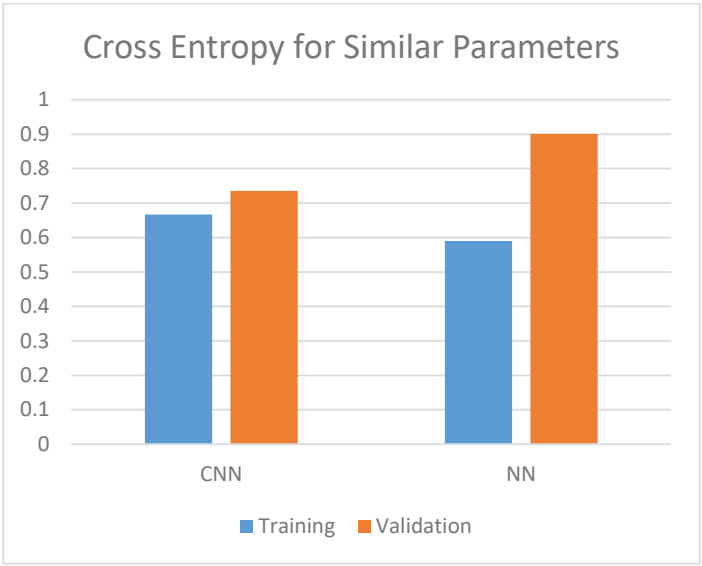
Third weight matrix and biases: $48\ x\ 7 + 7\ Biases = \ 343$

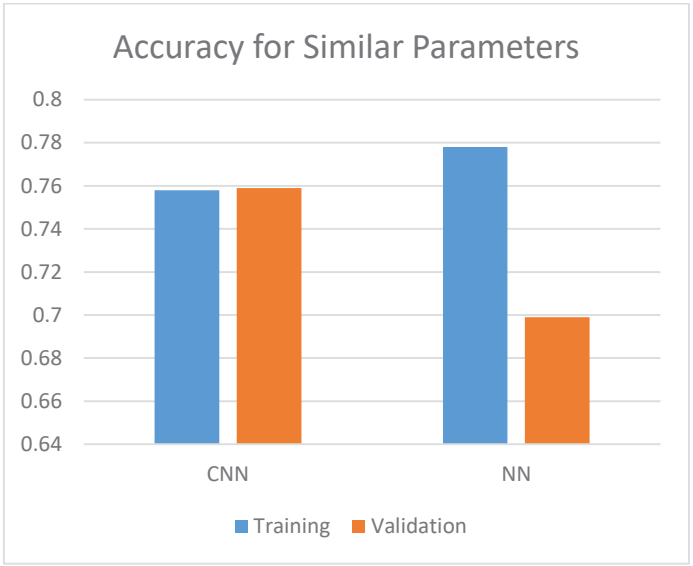
Total number of parameters for NN: 1035

So, our number of parameters is now very similar between the two networks. Performance:



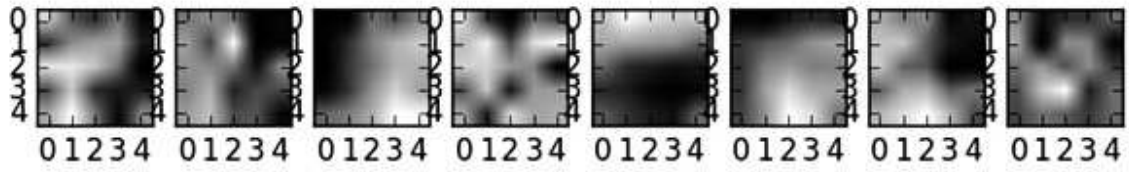




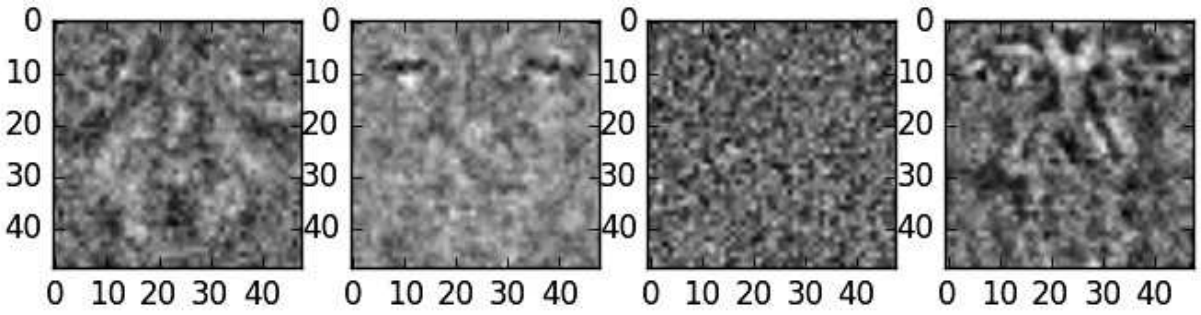


Comparing the performance, the CNN performs much better than the fully-connected network. Not only does it have better accuracy and lower cross-entropy, the NN seems to overfit, unlike the CNN as there is a large discrepancy between the training and validation values. Thus, the CNN leads to better generalization. This is due to the fact that the convolutional neural network is designed to take into account the spatial arrangement of the data, and does not treat input pixels which are far and close apart as on equal footing, as a fully-connected network does. The local connectivity of the convolutional neural network ensures that its filters adapt to the local input patterns. These properties allow it to achieve much better generalization than the fully connected network.

Plots for the first layer filters of CNN:



Plot of the first layer of weights of the NN:



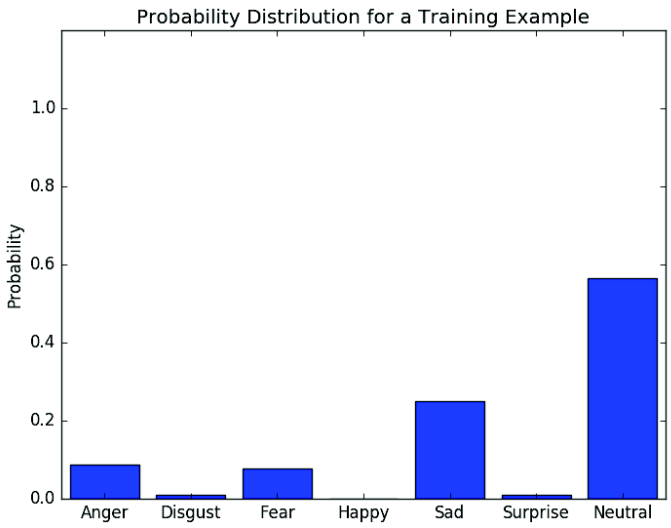
The following plots were generated using my plotmodels.py file, which modified the showmeans function from the kmeans.py file. Based on the images, we can see that there are much stronger image feature patterns present in the convolutional layer than the in the fully-connected network, which appear to be more subtle. The third input layer weights do not seem to have any sort of discernible pattern. Just approximately based on the results, seeing these plots makes sense when considering the superior performance of the CNN for roughly equivalent parameter amounts.

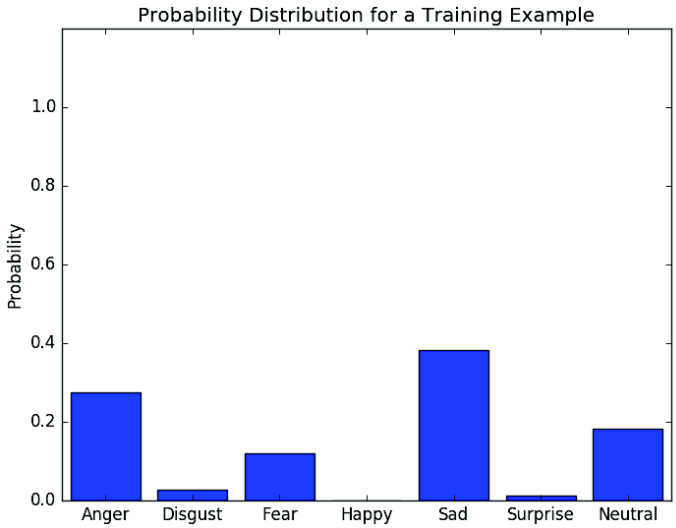
3.5 – Network Uncertainty

We are asked to plot examples of where the network is not confident of the classification output. My interpretation of this definition is examples where the softmax function output for that classification is low. I did this by making a modified version of nn.py, which I called nn_modified.py. Here, in the main section, I check random training examples and analyzed the probability distribution of the outputs from the softmax until I found examples where the network had low confidence in the classification output.

```
def PlotProbabilities(softmax, targets):
     plt.xticks([1, 2, 3, 4, 5, 6, 7], ['Anger', 'Disgust', 'Fear', 'Happy', 'Sad',
'Surprise', 'Neutral'])
     plt.bar([1, 2, 3, 4, 5, 6, 7], softmax,align='center')
     plt.axis([0.5, 7.5, 0, 1.2])
     plt.title('Probability Distribution for a Training Example')
     plt.ylabel('Probability')
     plt.show()
```

Then, using one of my saved models, I did trial and error until I came upon training examples where the highest probability for a class was less than 0.6, which I declared as my threshold. Examples:





For the right case, the model correctly predicts 'Sad' as the emotion; however, for the left case, the model predicts 'Neutral', but the correct emotion is 'Sad'. I use the target_train output from the LoadData function to check for each of these examples. Thus, when the neural network is uncertain in its prediction, it's likely that the model will make an incorrect prediction if it simply chooses the class with the highest probability from the Softmax function. To incorporate this into our model, or in general models where there are high stakes involved in what we do with the classification information, we could set some form of logic which only outputs the classification if the probability is above some threshold, or return the output class along with a value of uncertainty.

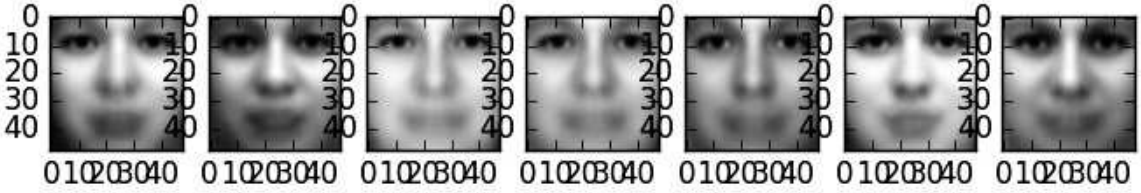
CSC2515 Assignment 2 Question 4 - Leo Woiceshyn (998082159)

4.2 – Training

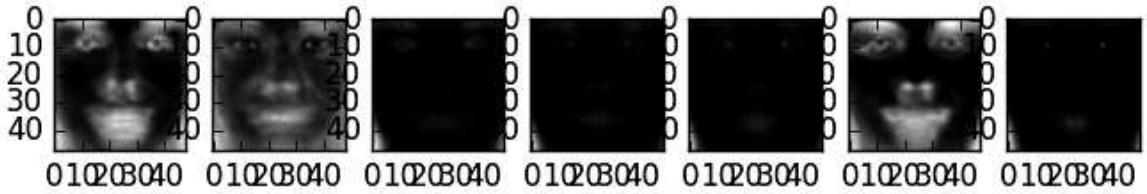
For choice of my randConst parameter, I essentially just used trial and error with a few different values of randConst (0.1,1,5, 10, 100) and chose the parameter which maximized the log-likelihood of my data, which was 10. There are likely better ways to do this, but due to time constraints I chose this method. Changing randConst did not seem to have much of an impact on the results either way. Once I had my randConst value selected, in order to ensure I was not getting poor results due to local optima, I ran the function 10 times for the same parameter values and chose the results for which the log likelihood was the highest, which was: Iter 10 logLikelihood 6360833.88963

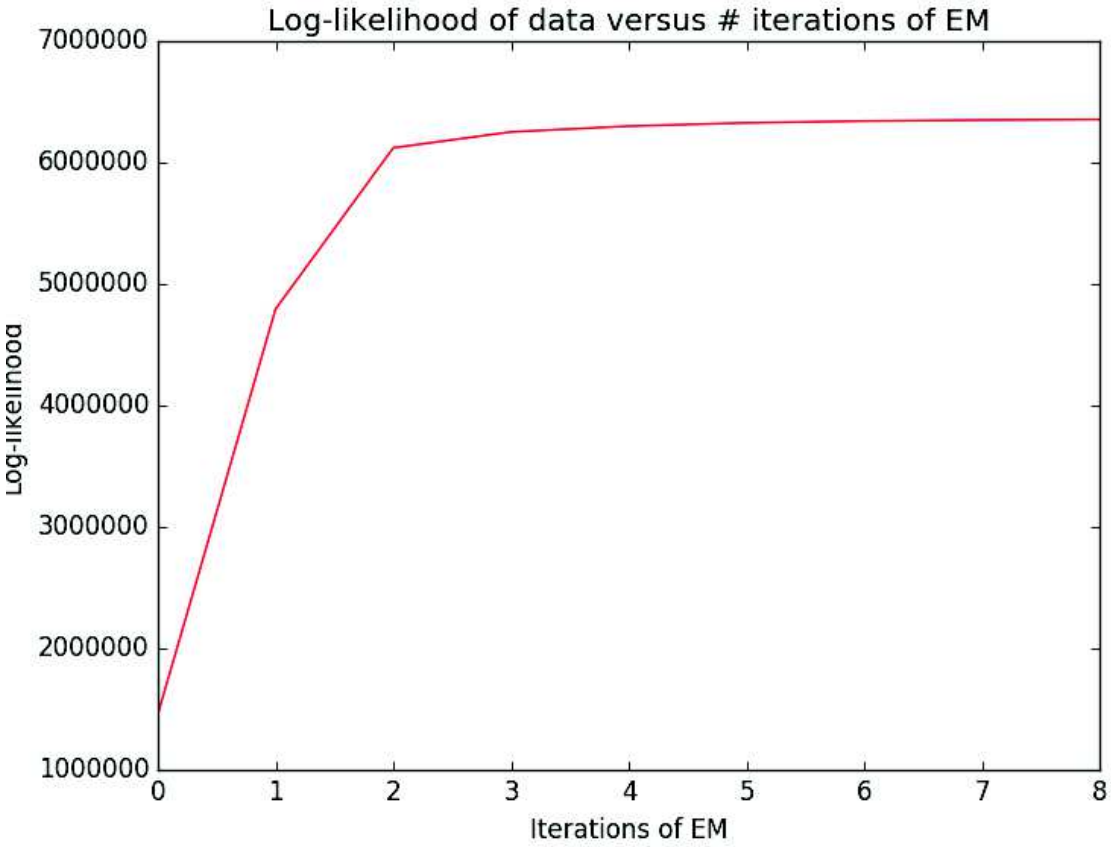
Results for 7 clusters, minimum variance of 0.01, and randConst of 10.0:

Means:



Variances:





mixing coefficients =

[[0.08853854]

[0.05336334]

[0.13518085]

[0.17718169]

[0.20452034]

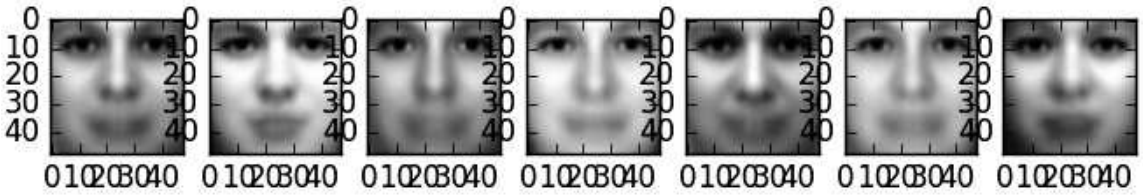
[0.09432989]

[0.24688535]]

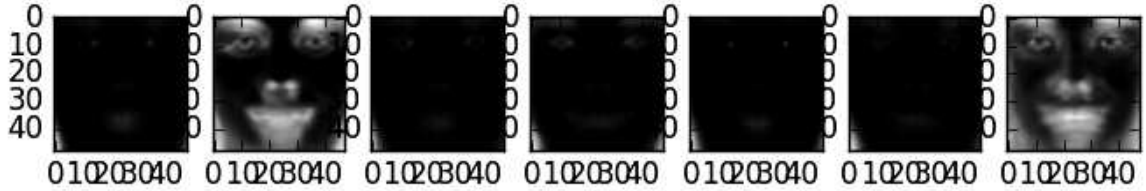
4.3 – Initializing a mixture of Gaussians with K-means

For the K-means initialization, I used the same randConst value. Similarly to the previous section, I ran the program 10 times and took the results that yielded the highest log likelihood. Here are the results for my best run, which gave me Iter 10 logLikelihood 6405141.14065:

Means:



Variances:



mixing coefficients =

[[0.07291034]

[0.19060971]

[0.07613867]

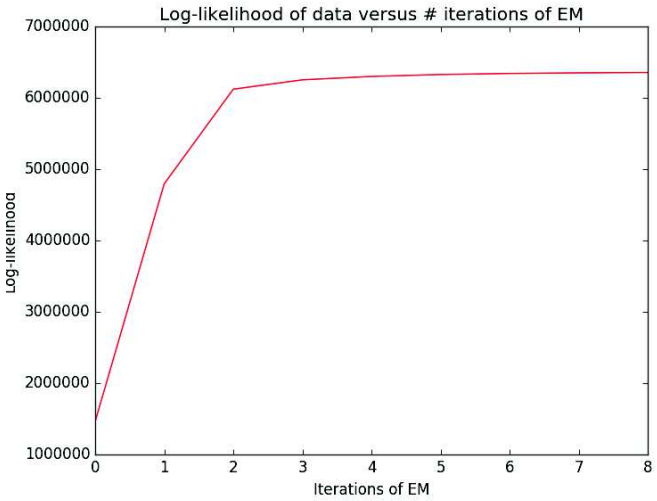
[0.21407571]

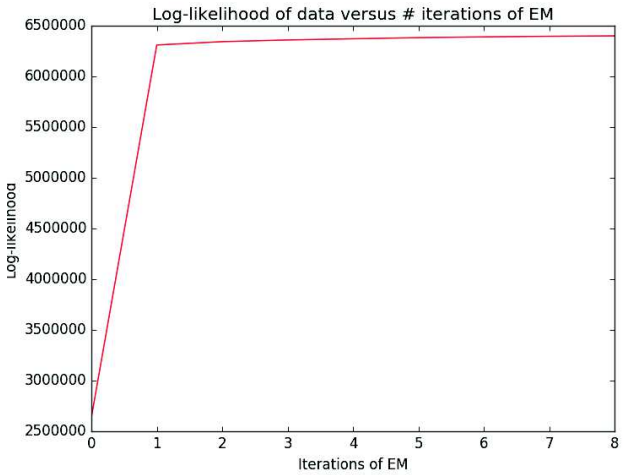
[0.11534503]

[0.23734984]

[0.0935707]]

Log Likelihood Comparison:



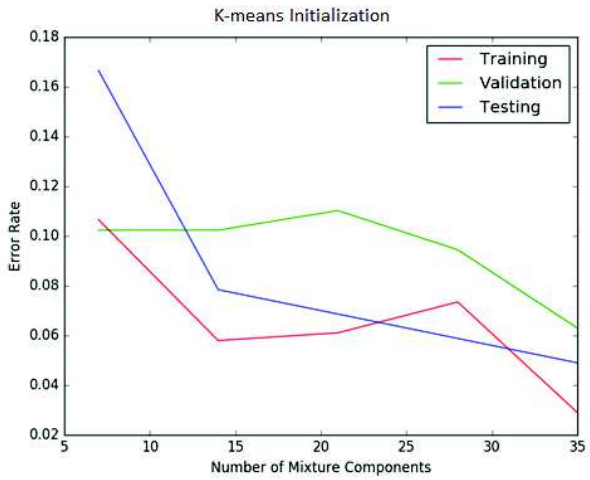


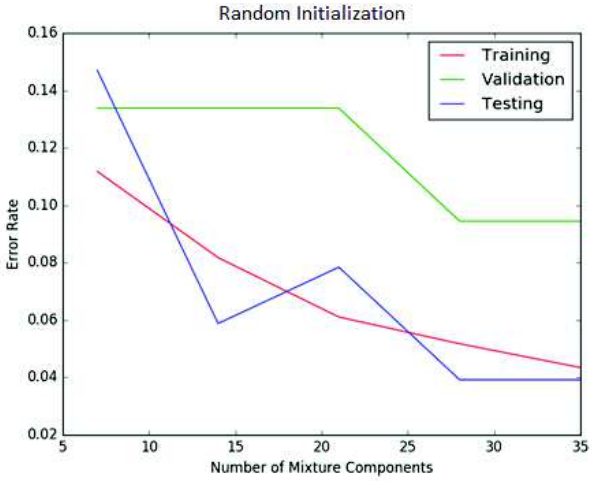
As it should be, the convergence was not only much faster using the K-means initialization method, but the log-likelihood also ended at a higher value. Thus, K-means initialization should be the preferred method.

4.4 – Classification using MoGs

4.4a)

Here are plots of the three error curves for training, test and validation, using a randConst value set to 10, as I did previously. The two different plots are for the two different initialization methods (random and K-means):





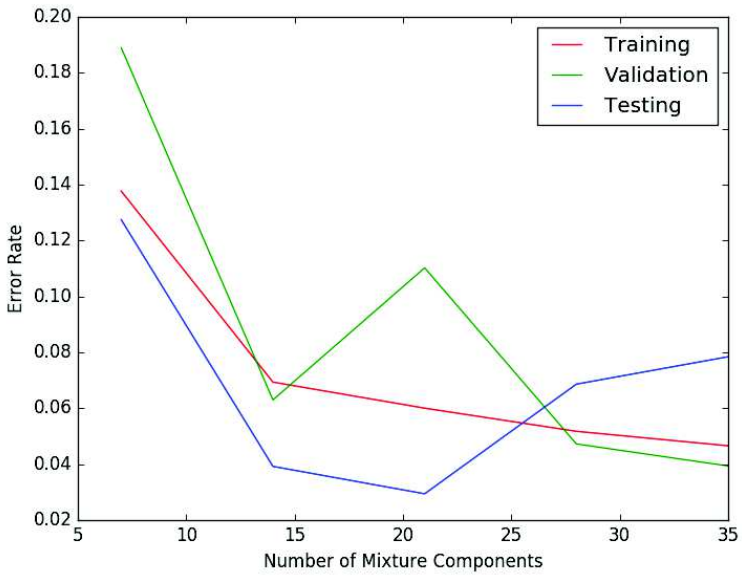
4.4b)

The reason that the error rates on the training sets generally decrease with increasing number of clusters (mixture components in this case) is that having more clusters allows the model to handle outliers better, since they can be assigned to their own cluster, and thus don't result in providing a lot of error to other clusters. Essentially, if every single data point was assigned to its own cluster, there would be 0 error, but the model wouldn't accurately represent the data (overfitting).

4.4c)

In each of the two initialization cases plotted above, the test set error starts higher than the validation or the training error, but reduces quickly to around the training set error rate as the number of mixture components is increased. The reason that the test set error starts high is that initially the model is underfitting the data, and there are not enough clusters to accurately represent the data. The error rapidly decreases until around 15 mixture components, and then slowly begins to decrease. Since the test error never begins to increase, it's unlikely that the number of clusters is too high.

To illustrate a case when the model might be overfitting, I modified some of the parameters in mogEM.py to produce a plot where the test error starts to go up:



In this case, the ideal number of mixture components is 15-20, and further increasing it would lead to overfitting, increasing the test error.