COMP4107 - Assignment 3

Student Name: Yunkai Wang
Student Number: 100968473

Student Name: Jules Kuehn
Student Number: 100661464

Fall 2018

1. Question 1 a) See q1.py for code. To see a comparison of input and output to the network, uncomment printVector() statements.

   In general, the fewer training images were used, the better the Hopfield network performed. These results suggests that a Hopfield network is a very poor choice for classification.
   A major factor in the accuracy was the selection of training images. Since fewer training images performed better, a random selection of a representative 1 or 5 leaves much to chance. To address this, we selected the training images with k-means, taking the images closest to the 2 cluster centers. This results in a selection of training images in which all 1's are very close to the center 1, and all 5's are very close to the center 5. This worked better than random selection, and didn't show the same sharp decline in classification accuracy as more training images were added.
   Alternately, we tried taking as many cluster centers as training images, finding the closest image for each cluster center. This should result in selecting training images that maximally differ from each other. This worked quite poorly as the quite different images caused further degeneration.
   The graphs below compare classification accuracy against the number of images used to train the network (between 1 and 5 of each digit). Each graph shows the average classification accuracy over 5 trials, with 50 test images.

   b) The code to implement the Storkey improvements are also included in our q1.py code.

   As shown below, the Storkey algorithm does not substantially improve the results for the classification, even for the best case shown above (where)

2. Question 2 Our own implementation of K-means was used to generate centers and beta values for a Radial Basis Function neural network (implemented in q2.py). Training on a large proportion of the samples (63000 / 70000) yielded reasonable results (around 95% classification accuracy) but was very slow. It was not practical use such a large training set for all the comparitive tests below, so the accuracy suffers.

   How it works: First, the centers are initialized to a random training point. (Although the code allows for the initialization method suggested in class - creating a random point within

the min/max of each dimension - this method wasn't as effective.) After running K-means, the centers and betas are used to compute RBF activations for the training and test data. This reduces the dimensionality of the input to the number of centers chosen. The RBF activations were then fed into a feed-forward neural net with no hidden layer, which was trained with the Adam optimizer. 3-fold cross-validation was used to produce the results below.

a) Investigating a number of different sizes of hidden layer showed that the best value was around 160 centers. The graph below represents training for 100 epochs with no dropout, at a learning rate of 0.05, and with a sample size of 1000.

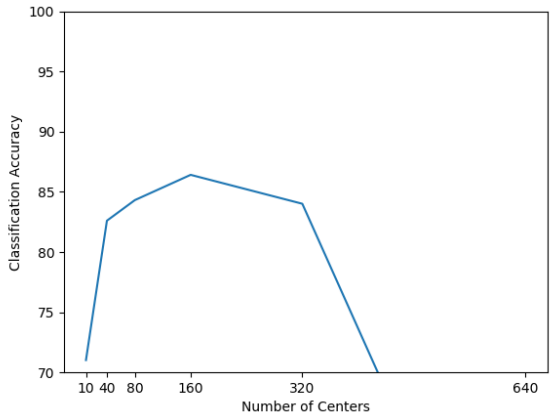


Figure 1: Classification accuracy vs. hidden layer size

Centers	Accuracy
10	71.02
40	82.60
80	84.31
160	86.41
320	84.01
640	42.57

b) Applying dropout before the output layer did not increase performance. We tested with the best case seen above (160 centers, 1000 samples, 100 epochs, lr=0.05) and also an "expanded" case (320 centers, 5000 samples, 500 epochs, lr=0.01). In both cases increasing the dropout level decreased the accuracy.

```
for 160 centers, 1000 samples, and 100 epochs, lr=0.05:
{
    0: 86.49364913138595,
    0.1: 86.49095127939911,
    0.2: 84.80501058292278,
    0.4: 83.68496464162325,
    0.6: 80.80279256419999,
    0.8: 77.10251759336823
}
```

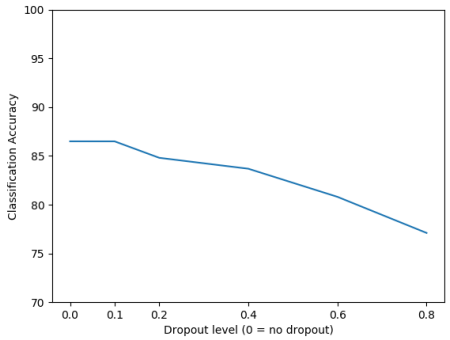


Figure 2: Dropout with 160 centers

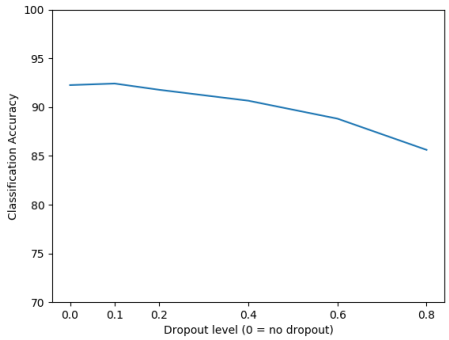


Figure 3: Dropout with 320 centers

```
for 320 centers, 5000 samples, and 500 epochs, lr=0.01:
{
    0: 92.25957215181523,
    0.1: 92.41908419990857,
    0.2: 91.77876615314769,
    0.4: 90.65975656015401,
    0.6: 88.81949583464971,
    0.8: 85.61887915837953
}
```

3. Question 3

   (a) For part a, here are the parameters and dimension of the SOM computed. We used only 1024 samples to train the SOM, the reason for doing this is just to shorted the running time. Otherwise the SOM takes too long to converge. Since we are using 1024 samples, so we used a SOM of size 12×12. The σ value is 1, and the learning rate is 0.1. Here is the graphs generated by using these parameters:

   It's clear that as time proceed, the SOM starts to group all the 1's together, and all the 5's together, where initially the SOM was initialized randomly.

   (b) Here are the plots for the K-means solutions with different number of clusters.

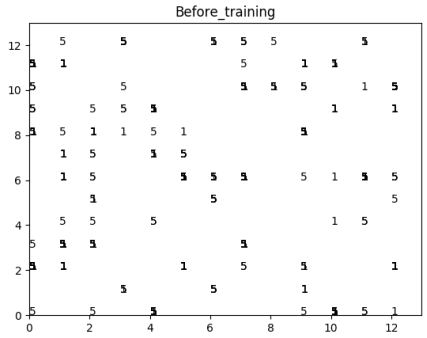


Figure 4: Training on perfect data

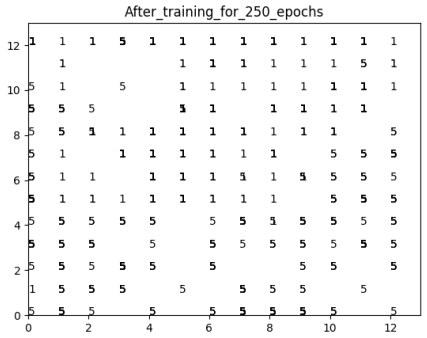


Figure 5: Re-training after noisy data

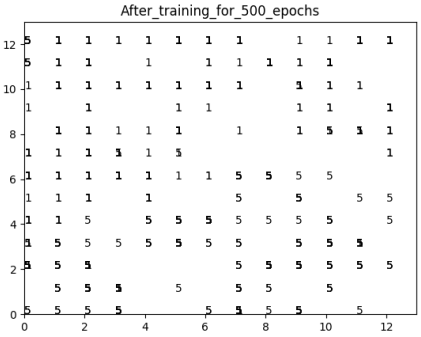
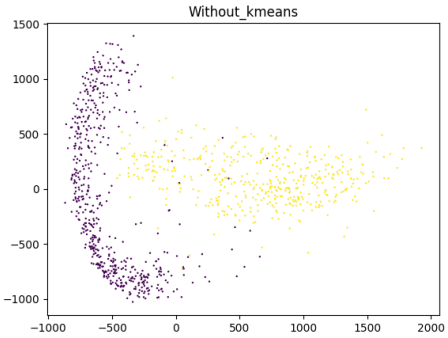
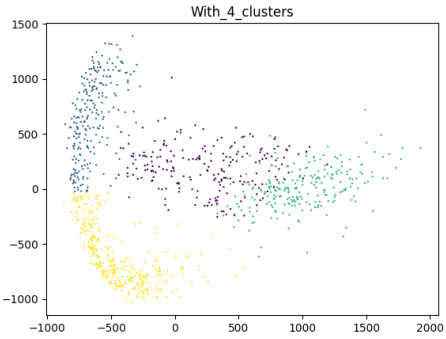


Figure 6: Re-training after noisy data

4. Question 4

   (a) We tried this question with two hidden layers, and we tried a different number of hidden layers for both layers. But the accuracy doesn't change for the different number of layers that we tried. The accuracy is always 9% for all values that we tried. There may be some problem with the implementation we had.






```
>>> print(experiment(h1_size=200, h2_size=50))
0.09003436426116838
>>> print(experiment(h1_size=200, h2_size=50, learning_rate=0.01))
0.09003436426116838
>>> print(experiment(pca=True, h1_size=200, h2_size=50, learning_rate=0.01))
0.09003436426116838
```