

COMP4107 - Assignment 3

Student Name: Yunkai Wang
Student Number: 100968473

Student Name: Jules Kuehn
Student Number: 100661464

Fall 2018

1. a) See q1.py for code.

As shown in the diagram and table below, the fewer training images were used, the better the Hopfield network performed in general. This suggests that a Hopfield network is a very poor choice for classifying even two classes.

A major factor in the accuracy was the selection of training images. Since fewer training images performs better, with a random selection the performance of the network is basically a matter of chance. To address this, we selected the training images with k-means, taking the images closest to the cluster centers, and as many cluster centers as training images. This would (in general) make for training images that are more different from each other, which would limit the degeneration.

Figure 1: Function Contours

Figure 2: Cost (MSE) at epoch

Using 8 neurons for the hidden layer is appropriate for this function. The MSE for the test data with 8 neurons is better than is required, and the training converges after fewer epochs than the network using 50 neurons for the hidden layer.

The table below corresponds to the experiment shown in the graph above.

- b) By setting "storkey = True" in q1.py, you can inspect our results for the bonus.

As shown in the table below, the Storkey algorithm does not substantially improve the results for the classification.

Figure 3: Convergence vs optimizer

Figure 4: CPU time vs optimizer

The reduced number of required training epochs for RMSPropOptimizer (seen in the table below) more than compensates for the increased CPU time for each epoch, seen in Figure 4.

- c) We can further show that 8 neurons is a good choice by noting that early stopping has no effect. Our experiments have not been able to demonstrate an overtraining phenomenon with

even 50 hidden neurons, but a premature early stop can be seen in Figure 6, and the table below.

Figure 5: Slow convergence

Figure 6: Premature early stop

The network with 50 hidden neurons also is very slow to converge to the target MSE - even using the RMSPropOptimizer - compared to when using 8 hidden neurons (Figure 5).

Figure 7: Reaching training goal

Figure 8: Early stopping not triggered

Figure 7 shows the MSE at each epoch near where the training goal is reached. This captures the expected trend - that the training error will decrease consistently, while the test error (9x9 grid) has more variation. The validation data is most different from the training data (as it is randomly sampled from the function), so it is expected that its error would also have the highest variance.

Figure 8 shows that having an early stopping mechanism for this function makes no difference, as the early stopping mechanism is never triggered (8 hidden neurons, 4000 epochs, 0.02 LR). Thus, the two contours are virtually identical, differing only by the stochastic elements of the experiments (initial weights, order of training data).

2. Instructions on how to run q2.py:

Running q2.py without changing anything will run all three parts. This may take a while as it runs 10 experiments for each number of hidden neurons, and after one is finished, a new neural net is created to run the second one, etc. Therefore, it will take about 1 min for it to finish and plot the graphs. To run a single part only, comment out the other lines in the `experiment()` function.

We started by using the default learning rate for AdamOptimizer (0.001). This took 400-700 epochs to converge to the result. However, if we use 0.01 as the learning rate, it converges faster – within 100 epochs every time. Therefore, we chose 0.01 as the learning rate for this question.

- (a) We took the averages of 10 experiments for each of 5, 10, 15, 20, and 25 hidden neurons. The results are shown below.

Figure 9: Percentage of Recognition Errors

The table below corresponds to the experiment shown in the graph above.

- (b) 15 is chosen as the number of hidden neurons, since 15 is more accurate than 10, but using more than 15 neurons is not much better (if at all). When we used 15 neurons in the hidden layer, the number of epochs to reach zero errors is much less than is shown in the assignment question, which we think is fine. Perhaps this is due to a different learning rate.

Figure 10: Training on perfect data

Figure 11: Re-training after noisy data

- (c) The network trained with both perfect and noisy data performed just as well on perfect data, and better on noisy data. The chart below is for the network with 15 neurons.

Figure 12: Percentage of Recognition Errors

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

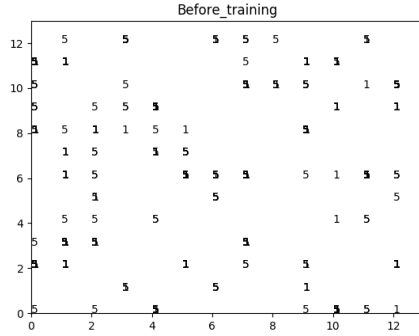


Figure 13: Training on perfect data

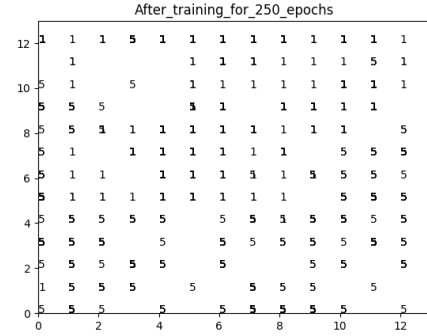


Figure 14: Re-training after noisy data

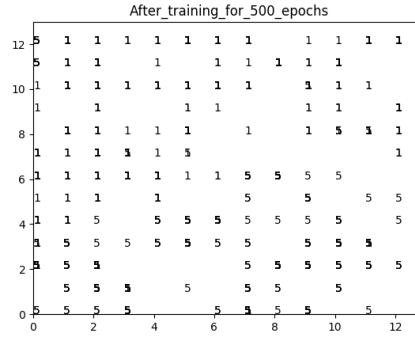


Figure 15: Re-training after noisy data

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.

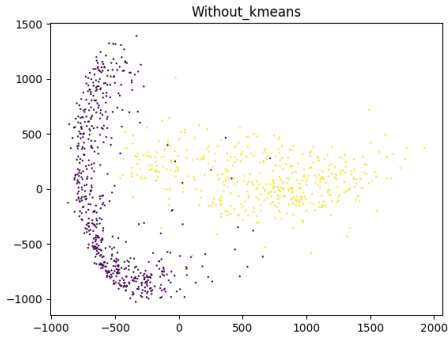


Figure 16: Training on perfect data

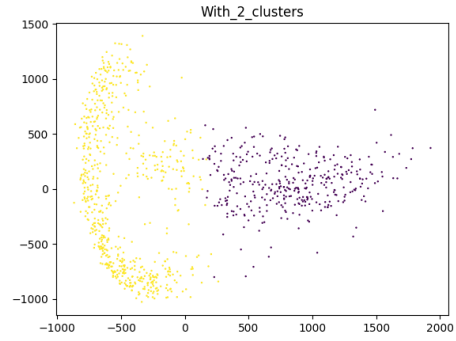


Figure 17: Re-training after noisy data



Figure 18: Re-training after noisy data

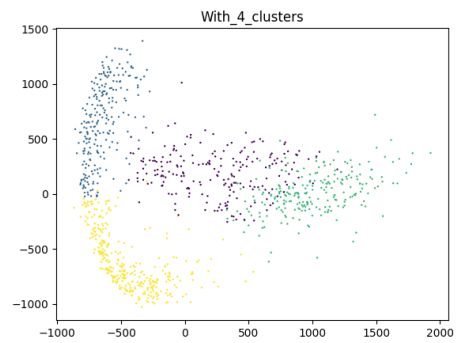


Figure 19: Re-training after noisy data



Figure 20: 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
```

Figure 21: Training on perfect data