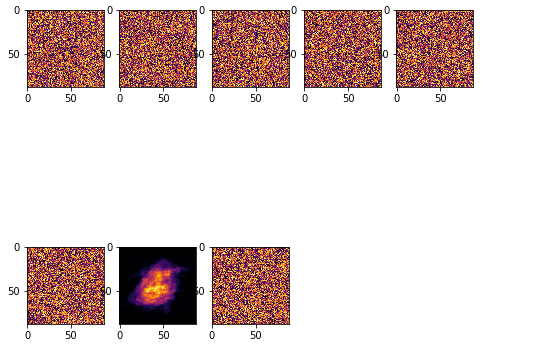
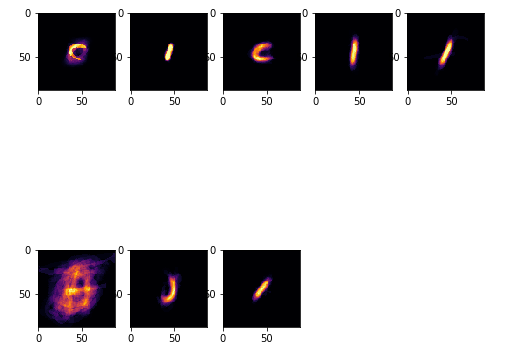
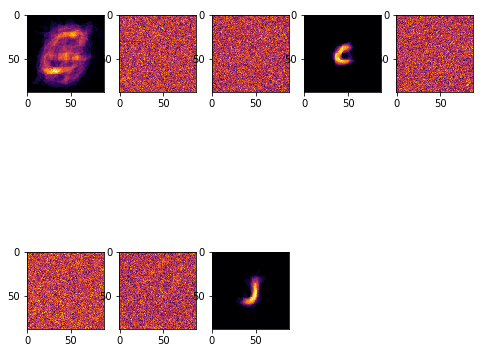
**Competitive Learning Assignment notes:**

Steps to take:

1. Normalised input or initial weights.
2. Noise addition on the weights.
3. Decaying learning rate.
4. Leaky learning: update the weights of the losers as well as winners but with a much smaller learning rate.
5. Update the winners and neighbouring losers.

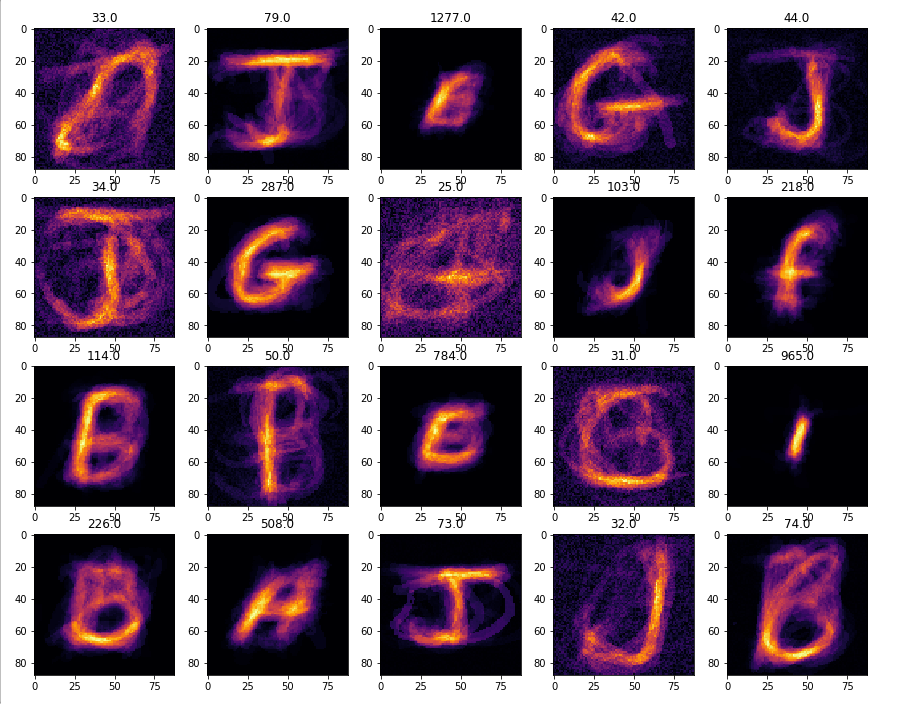
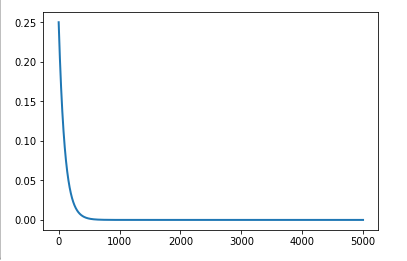
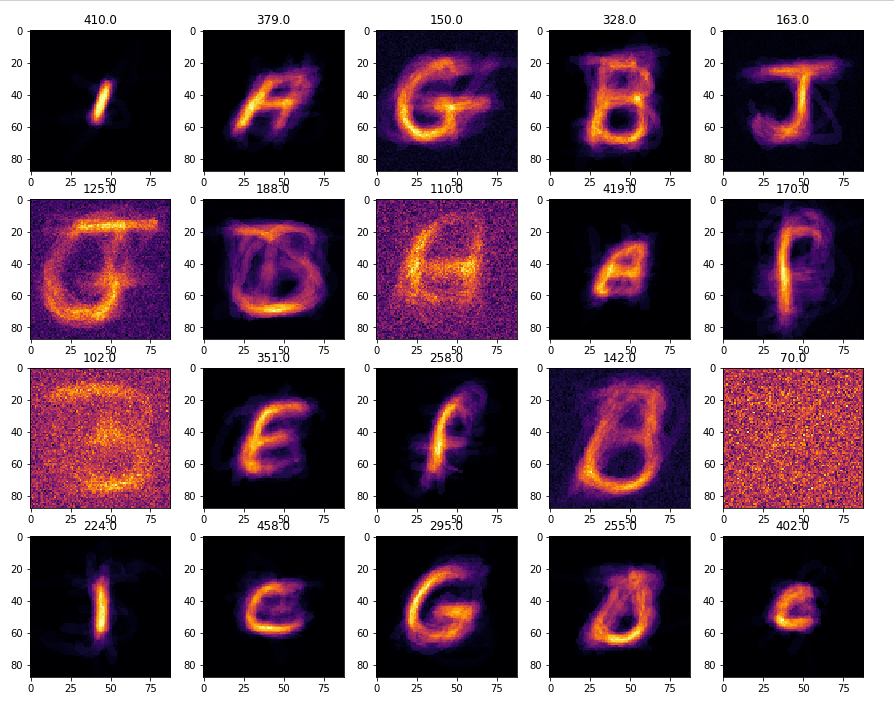
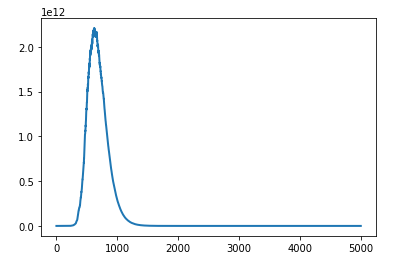
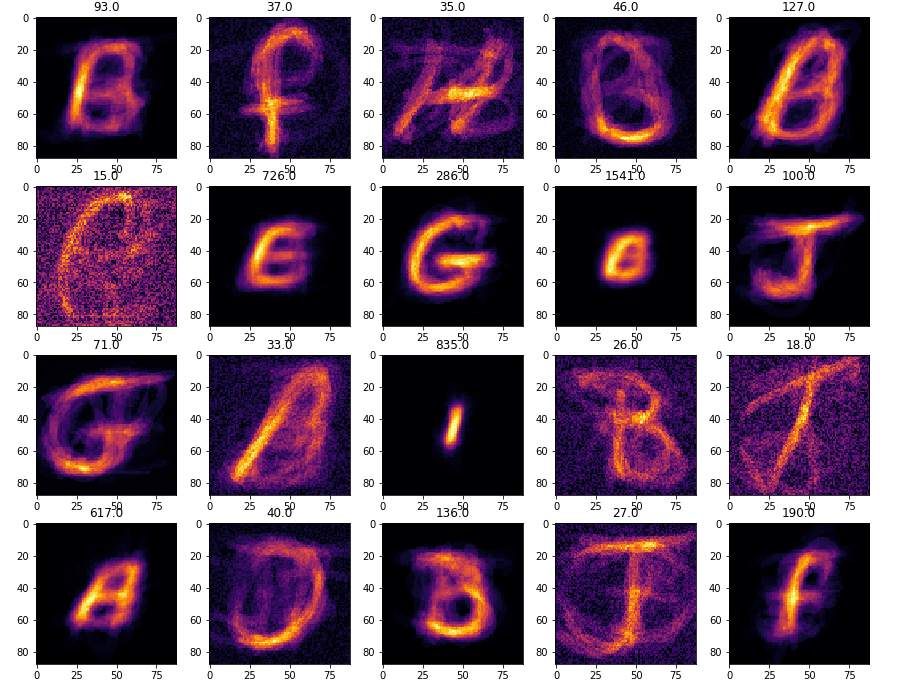
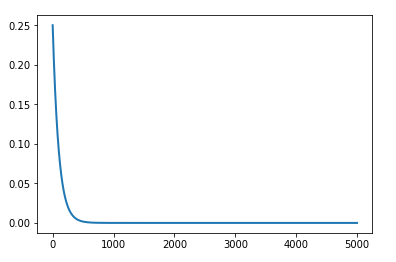
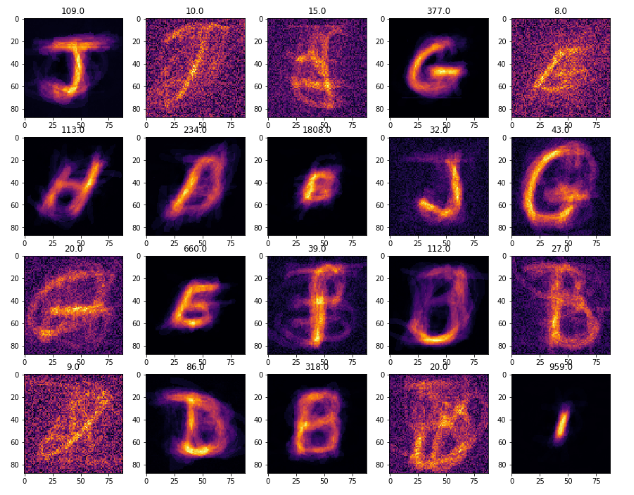
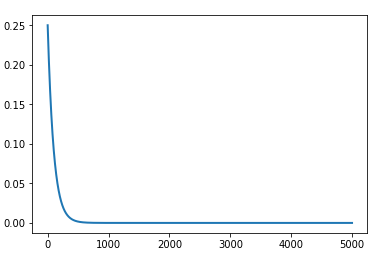
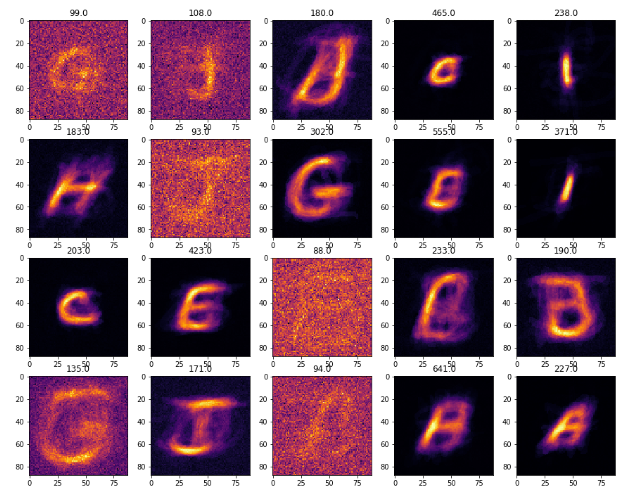
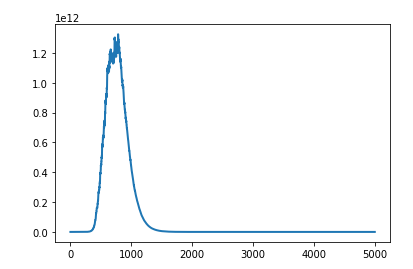
* Original prototype graphs looked like this:
* 
  + This is when using only the lab5 solutions with no changes
* When theres no normalization, this is how it looks like:
* 
  + Normalization eliminates the repetition of data (https://stackoverflow.com/questions/1102590/what-exactly-does-database-normalization-do), so in this case normalization removes the repeated C, I and J letters
* Added noise, which has improved the result of the prototypes. Less dead cells, and more letters are forming. It seems the noise has also darkened the dead cells
  + Code:
    - noise = np.random.normal(0,1,(letters,n)) # add noise
    - W = W + noise
* 
  + The 3 letters I get are ‘C’, ‘I’ and ‘J’ and rarely do I get one letter repeated.
* So I found out that in the counter that records how many times a particular output neuron won, theres one neuron that’s increasing too much, which is the one with all the letters mixed up in it, while theres 5 dead ones with 0 as the count the entire time.
  + Implementing a decay to the weight change could fix this and as a result, distribute the winning neuron counts to the other dead neurons

TODO:

* Normalize the train data as well. Check PCA (Lab 4)
  + When normalizing, change x = train[:,i] to x = train[i,:]

24/03/2019

Notes:

* **I’ve started again but this time with the base that the data and weights are normalized.**
* **Initial conditions:**
  + Learning rate (eta) = 0.05
  + Winit = 1
  + Alpha = 0.990
  + Tmax = 5000
  + Prototypes (Letters) = 20
  + Normalizing using Repmat
  + Dead cell is anything below 50 wins
  + Running 10 times to find average of dead cells
* **Results with initial conditions:**
  + 
  + 
  + Counter = [[ 33. 79. 1277. 42. 44. 34. 287. 25. 103. 218. 114. 50. 784. 31. 965. 226. 508. 73. 32. 74.]]
  + Dead cells = 7
  + Average Dead Cells = 7.7
* **1) Adding Noise to weights**
  + Adding noise to weights spreads out the count of winners across the neurons. Meaning more neurons win as a result of noise. But this also means that dead cells have a higher win count than before
  + Therefore it is possible to have a neuron with over 100 wins, but with the visual appearance of a dead cell.
  + The standard for dead cells is therefore increased to 150, as opposed to 50, when dealing with noise.
  + The running average curve also seems to have an interesting change, with a parabolic effect.
* **Results with 1):**
  + 
  + 
  + Counter =[[410. 379. 150. 328. 163. 125. 188. 110. 419. 170. 102. 351. 258. 142. 70. 224. 458. 295. 255. 402.]]
  + Dead cells = 5
  + Average Dead Cells = 7.5
* **2) Decaying learning rate**
  + The learning rate (eta) is multiplied by e^(winning count/1000). So for example if a neuron has won 200 times, the decay will be e^0.2.
* **Results with 2):**
  + 
  + 
  + Counter = [[ 93. 37. 35. 46. 127. 15. 726. 286. 1541. 100. 71. 33. 835. 26. 18. 617. 40. 136. 27. 190.]]
  + Dead Cells = 9
  + Average Dead Cells = 8.4
* **3) Updating loser neurons by 5% of winning neuron weight update**
  + Weights for the loser neurons are updated, but only by 5% of dw, which is what is added to the weight of the winning neuron to update it.
* **Results with 3):**
  + 
  + 
  + Counter = [[ 109. 10. 15. 377. 8. 113. 234. 1808. 32. 43. 20. 660. 39. 112. 27. 9. 86. 318. 20. 959.]]
  + Dead Cells = 10
  + Average Dead Cells = 9.7
* **Results with 1),2) and 3) altogether:**
  + 
  + 
  + Counter = [[ 99. 108. 180. 465. 238. 183. 93. 302. 555. 371. 203. 423. 88. 233. 190. 135. 171. 94. 641. 227.]]
  + Dead Cells = 6
  + Average Dead Cells = 6.8