ECE657 Assignment 2, Hammad Ahmed Rehman (20293340, harehman)

2) The paper begins by colloquially defining the Hopfield network as a network of densely connected neurons, with each neuron being allowed to have a value of +1 or -1 (sign function). The weights in between them are fixed. Computation takes place by each neuron evaluating the weighted sum of the states of all the neurons in the system, and applying the sign function to the sum. The paper discusses two modes of changing the state of an "input probe" (a vector used to "retrieve" a memory); asynchronous and synchronous.

The paper indicates that such a network is useful as it allows for a "high degree of parallelism, distributed storage of information, robustness" and the ability to perform tasks of low complexity. Each network will have "fixed points", ie memories in the system. The paper attempts to specify what the maximum memory capacity of a Hopfield network would be. The paper hypothesises that each stored memory is an "attractor" such that any input probe which is sufficiently similar to the stored memory is "attracted" to the memory by a series of operations performed by the network, ie the probe's state is modified to match the stored memory. It defines the "Hamming distance" as a measure of the similarity between a probe and a stored state; this distance is defined as the number of positions at which the two input vectors are different.

The paper then goes on to talk about how to define rules for encoding memories into the network such that the process described in the previous paragraph works, and how that encoding affects the capacity of the network. The encoding rule can be thought of as the algorithm to define the weights between the neurons. The algorithm roughly goes as follows: define a matrix T which is the Hopfield connection matrix, a function of the vectors to be encoded. Then, when given a probe x, you use T to find the stored memory with the smallest Hamming distance to the probe. You do this by finding T dot x and then replacing the ith component of x (i being random) with the sign of the ith component of Tx. This process is convergent according to Hopfield. This process could also be done synchronously. The paper also goes on to show alternative methods of construction the Hopfield matrix capable of storing a similar amount of memories.

The next part of the paper argues intuitively about the capacity. If you require a "fixed" value of each stored memory of size n, then the upper bound on the capacity of the network is n memories. But if you let the requirements for those stored memories be probabilistic (i.e. you settle for a certain probability of accuracy rather than 100%), the upper bound on capacity is 2n.

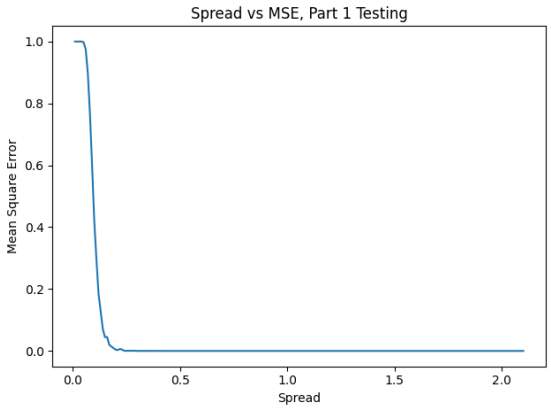
The next section (IV) shows examples of how the network works. One of the points it highlights is that it is possible for a probe to land at a fixed point which is either not a memory or not the closest memory. Section V, stability, mentions the attractor feature of neurons again. It mentions that one approach to this is to fix the signs of known components of the probe to their values, regardless of the value in Tx, but they argue that it doesn't make a difference in the end. This section then makes a probabilistic argument where if you know (1-p)n

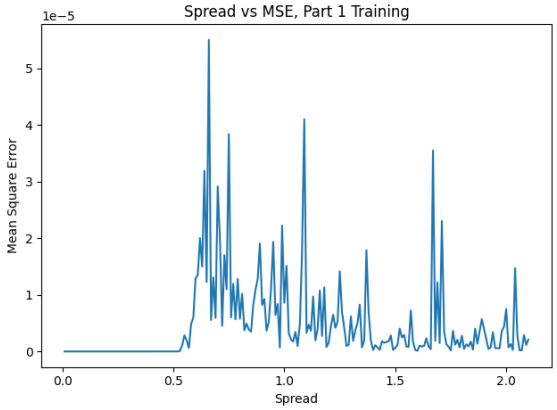
components, then you have a sphere of radius pn around each memory. If any probe lands in those spheres, then it converges to the centre of that sphere (in one or multiple steps, depending on whether it's synchronous or asynchronous). The last part of Section V derives the concept of energy mentioned in class.

Section VI informally presents the capacity heuristics, but in summary, the asymptomatic capacity depends on your accuracy. If you probe the network with a probe where the values are at most pn away from a fundamental memory, and you want exactly 100% accuracy, your capacity is: [(1-2p)^2/4][n/logn]. If you want "essentially" 100% accuracy but can allow for a miniscule amount of errors, then that capacity doubles. If you allow for recall of fundamental memories without the requirement that any of them be exact, then your capacity is n/2logn. Finally, if your p is <1/2, and you require a small fraction of the memories to be recalled exactly, your capacity is n/4logn.

The rest of the paper before the conclusion goes on to formally prove all the concepts above. I will assume, given the limits of space on this question, that we are not required to summarise those proofs in any detail, but rather summarise their conclusions as I have done above. What struck me most was that allowing a certain degree of error to come in to the recall feature of the network (ie require that the fundamental memories are not fixed points) allowed a doubling of capacity.

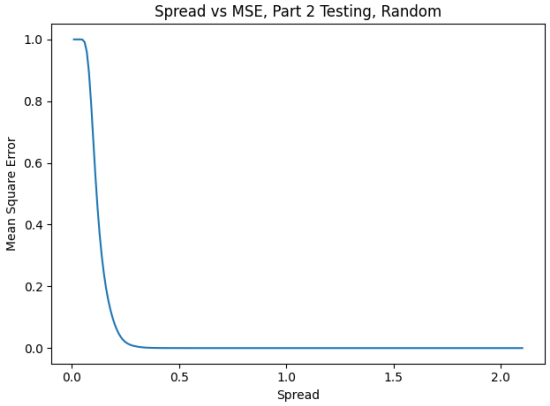
3) For the first part, I setup all 441 points, and did a random split where I used 351 points for training, and 90 for testing. The network was setup with 351 neurons/centres. I then varied the spread from 0.01 to 2.1 in steps of 0.01, and documented the Mean Square Error. Here is a plot of the error vs spread for the first part for training and testing:

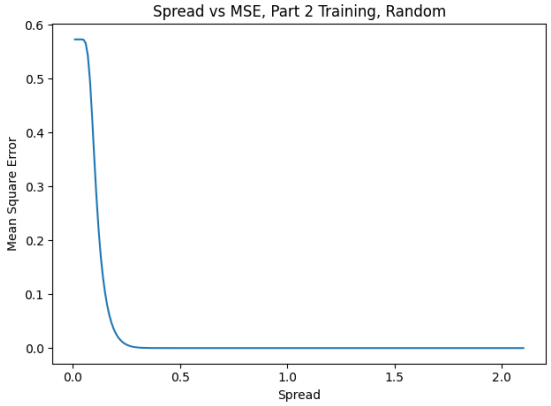




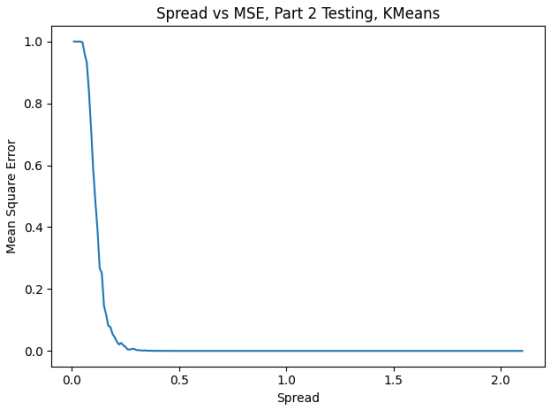
The error settles for testing around a spread of 0.15 – 0.2. For training the error stays low up until a spread of 0.5, and then begins varying wildly. I believe that, because all 351 points has a centre, increasing the spread too much could cause the radius of multiple centres to overlap, resulting in the errors shown.

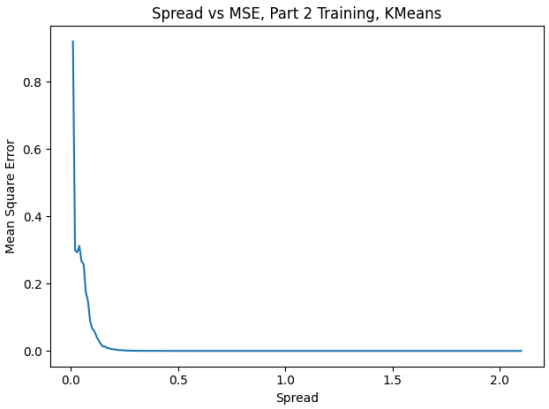
These are the graphs for the second part with random centre selection:





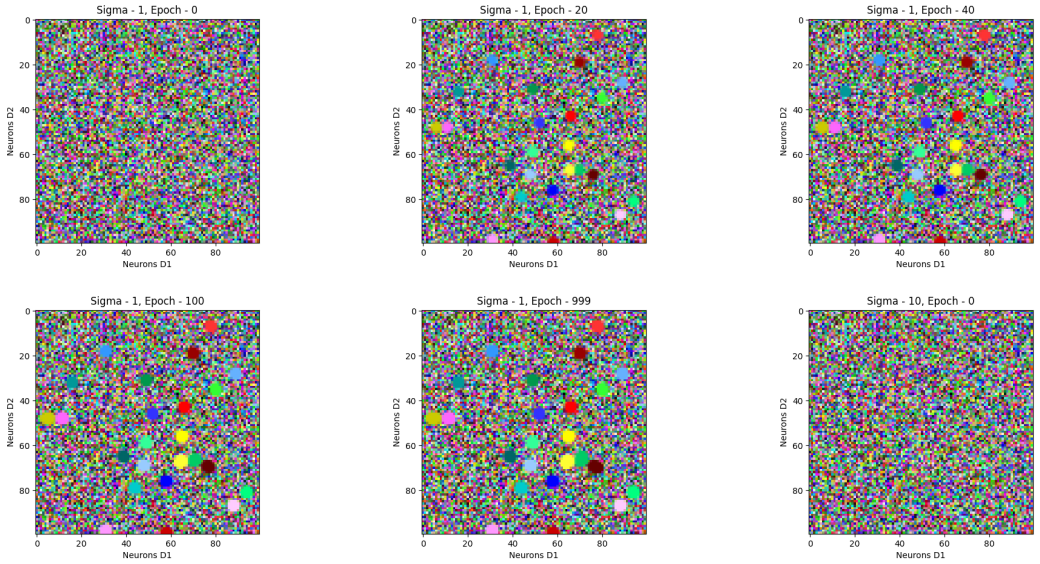
And for kmeans:

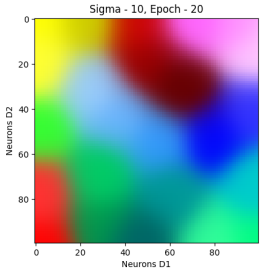


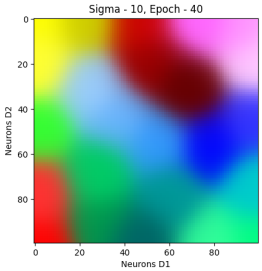


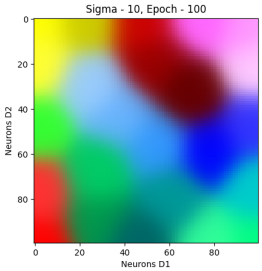
In terms of overall performance, the two networks with 150 neurons performed similarly to the one with 441 neurons. In both cases, minimum error was achieved at an approximate spread of 0.25. In both cases, the minimum error was achieved at a spread value of approximately 0.25, and it then settled there. There were no significant differences between testing and training accuracy in both cases. In all cases the performance seemed to have reached stability in my selected spread range, so I saw no need to extend the range further.

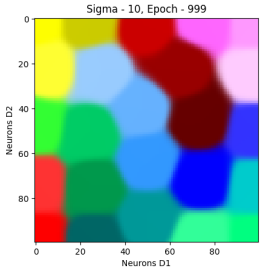
4) Here are the results of the training:

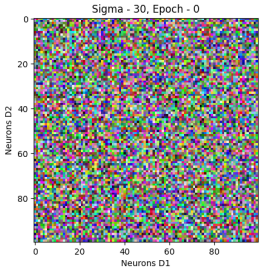


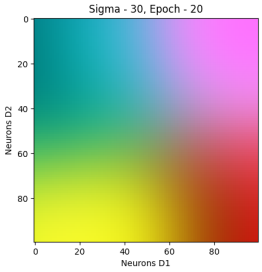

Sigma - 10, Epoch - 20
Neurons D2
Neurons D1

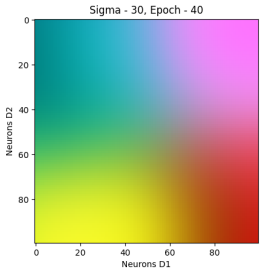

Sigma - 10, Epoch - 40
Neurons D2
Neurons D1

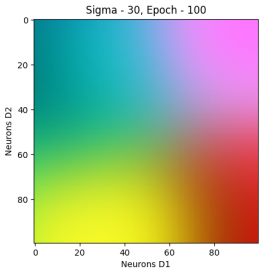

Sigma - 10, Epoch - 100
Neurons D2
Neurons D1

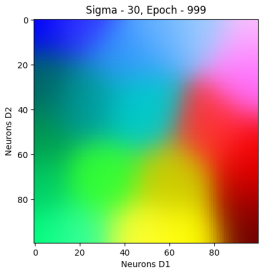

Sigma - 10, Epoch - 999
Neurons D2
Neurons D1

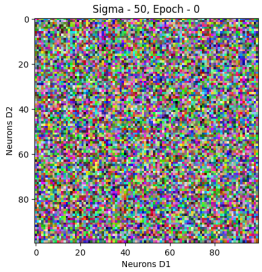

Sigma - 30, Epoch - 0
Neurons D2
Neurons D1

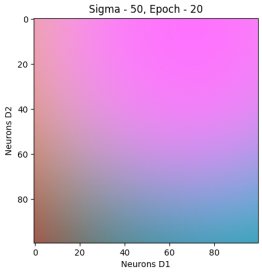

Sigma - 30, Epoch - 20
Neurons D2
Neurons D1

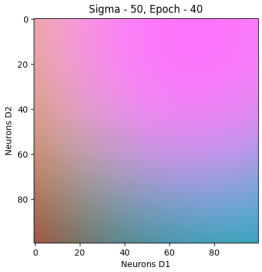

Sigma - 30, Epoch - 40
Neurons D2
Neurons D1

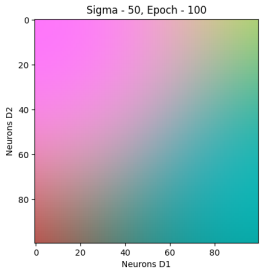

Sigma - 30, Epoch - 100
Neurons D2
Neurons D1

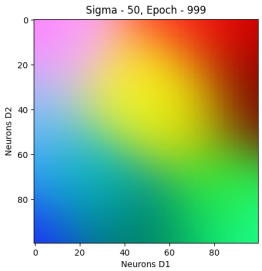

Sigma - 30, Epoch - 999
Neurons D2
Neurons D1

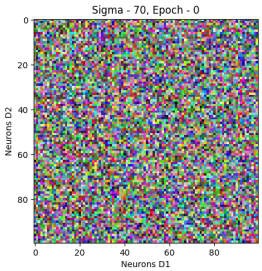

Sigma - 50, Epoch - 0
Neurons D2
Neurons D1

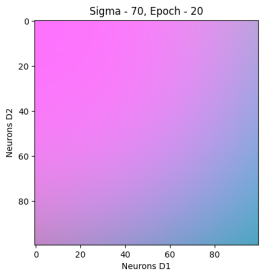

Sigma - 50, Epoch - 20
Neurons D2
Neurons D1

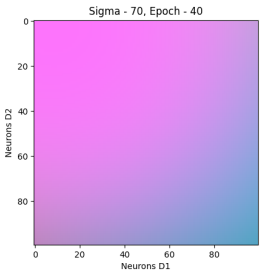

Sigma - 50, Epoch - 40
Neurons D2
Neurons D1

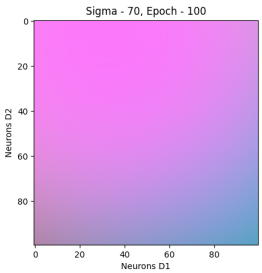

Sigma - 50, Epoch - 100
Neurons D2
Neurons D1

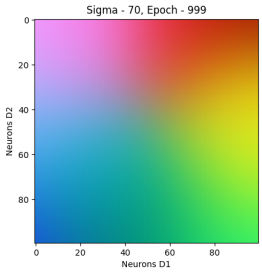

Sigma - 50, Epoch - 999
Neurons D2
Neurons D1


Sigma - 70, Epoch - 0
Neurons D2
Neurons D1









So here are my observations:

- In all cases, the weights being initialised at random led to many clusters prior to training (epoch 0)
- The greater the number of epochs, generally the more the “clustering” effect. However, the greater the number of epochs, the lesser the learning rate, and the lesser the effect of the neighbourhood function. So between epochs 0 – 100, there was a noticeable clustering slowdown effect. However, between 100 – 1000 there was generally a noticeable difference in the number of clusters, and this is probably because of the fact that there were 9 times as many epochs between those two figures than between 0 – 100. For future assignments, I would recommend doing 0, 20, 40, 100, 200, 300… 1000 epochs for greater insight
- The larger the sigma, the greater the effect of the neighbourhood function, resulting in most weights in the layer being updated, rather than the more local effect with lower sigmas. This led to a quick coalescing around the pink/blue gradient shown in sigma 50 and 70.
- It seems like sigma 10 was optimal for displaying clusters closely resembling the original colours I selected. Around the 1000^{th} epoch, it seems like that network was settling around those clusters