ECE 657

Assignment 2

(Spring 2022)

Group Members:

Dakota Wang (d82wong@uwaterloo.ca) (20985951)

Himalaya Sharma (himalaya.sharma@uwaterloo.ca) (20971031)

Uditya Laad (ulaad@uwaterloo.ca) (20986041)

UNIVERSITY OF WATERLOO

Question 1

Given:

$$e(n) = y_d(n) - y(n) \quad \text{---(1)}$$

Where: y(n) -> Actual Output at iteration 'n'

$Y_d(n)$ -> Desired Output at iteration 'n'

Actual Output:

$$y(n) = \sum_{k=1}^{N} w_k(n) \cdot \phi\{x(n), c_k, \sigma_k\} \quad \text{---(2)}$$

where :- ϕ ⟶ RBF seed Function

Cost function for Gaussian Kernel:

$$J(n) = \frac{1}{2}|e(n)|^2 = \frac{1}{2}[y_d(n) - y(n)]^2 \quad \text{---(3)}$$

$$= \frac{1}{2}|e(n)|^2 = \frac{1}{2}\left[y_d(n) - \sum_{k=1}^{N} w_k(n) \cdot \phi\{x(n), c_k, \sigma_k\}\right]^2 \quad \text{---(4)}$$

$$= \frac{1}{2}\left[y_d(n) - \sum_{k=1}^{N} w_k(n) \cdot \exp\left(\frac{-||x(n) - c_k(n)||^2}{2\sigma_k^2(n)}\right)\right]^2 \quad \text{---(5)}$$

$$\psi(n) = [\phi\{x(n), c_1, \sigma_1\}, \phi\{x(n), c_2, \sigma_2\}, \ldots, \phi\{x(n), c_N, \sigma_N\}]^T \quad \text{---(6)}$$

Pre-set:

From equation 6, we can say that:

$$\psi(n) = \begin{bmatrix} \phi\{x(n), c_1, \sigma_1\} \\ \phi\{x(n), c_2, \sigma_2\} \\ \vdots \\ \phi\{x(n), c_N, \sigma_N\} \end{bmatrix}_{N*1}$$

From equation 5, we can say that:

$$w(n) = \begin{bmatrix} w_1(n) \\ w_2(n) \\ \vdots \\ w_N(n) \end{bmatrix}_{N\times 1}$$

Using equation 2, and the above forms, we can say that:

$$y(n) = \sum_{k=1}^{N} w_k(n) \cdot \phi\{x(n), c_k, \sigma_k\} = w(n)^T \cdot \psi(n) \quad \text{---(9)}$$

Using Equations '3' and 'a', we can write:

$$J(n) = \frac{1}{2}\left[y_d(n) - w(n)^T \cdot \psi(n)\right]^2 \quad \text{---(b)}$$

Derivation 1:

Taking the derivative of Equation 'b' w.r.t. w(n):

$$\frac{\partial J(n)}{\partial w(n)} = \frac{2}{2}\left[y_d(n) - w(n)^T.\psi(n)\right].\left[-\psi(n)\right]$$

$$\Rightarrow \frac{\partial J(n)}{\partial w(n)} = -e_n.\psi(n) \longrightarrow \text{using eq. (1) \& (a)}$$

$$\Rightarrow \boxed{\left.\frac{\partial J(n)}{\partial w}\right|_{w=w(n)} = -e_n\psi(n)} \quad \text{(c)}$$

We know the update equation:

$$w(n+1) = w(n) - \mu_w.\left.\frac{\partial}{\partial w}J(n)\right|_{w=w(n)}$$

$$\Rightarrow \boxed{w(n+1) = w(n) + \mu_w.e_n.\psi(n)} \quad \text{- using eq. (c)}$$

Hence proved //

Derivation 2:

Taking the derivative of Equation '5' w.r.t. $c_k(n)$:

$$\frac{\partial}{\partial c_k(n)}J(n) = \frac{2}{2}\left[y_d(n) - \sum_{k=1}^{N} w_k(n).\exp\left(\frac{-\|x(n)-c_k(n)\|^2}{2\sigma_k^2(n)}\right)\right] \cdot \frac{\partial}{\partial c_k(n)}\left[-\sum_{k=1}^{N} w_k(n).\exp\left(-\frac{\|x(n)-c_k(n)\|^2}{2\sigma_k^2(n)}\right)\right]$$

$$= -e_n.\frac{\partial}{\partial c_k(n)}\left[\sum_{k=1}^{N} w_k(n).\exp\left(-\frac{\|x(n)-c_k(n)\|^2}{2\sigma_k^2(n)}\right)\right]$$

$$\Rightarrow \left.\frac{\partial}{\partial c_k}J(n)\right|_{c=c_k(n)} = -e_n.w_k(n).\frac{\partial}{\partial c_k}\left[\exp\left(-\frac{[x(n)-c_k(n)]^T.[x(n)-c_k(n)]}{2\sigma_k^2(n)}\right)\right]$$

$$= -e_n.w_k(n).\exp\left(\frac{-\|x(n)-c_k(n)\|^2}{2\sigma_k^2(n)}\right) \cdot \frac{-1}{2\sigma_k^2}\frac{\partial}{\partial c_k}\left(x(n).x(n)^T - x(n)^T.c_k(n) - c_k(n)^T.x(n) + c_k(n)^T.c_k(n)\right)$$

$$= e_n.w_k(n).\psi\{x(n), c_k(n), \sigma_k(n)\}.\left[\frac{1}{2\sigma_k^2}\frac{\partial}{\partial c_k}\left(x(n).x(n)^T - 2x_k(n).x(n) + c_k(n)^T.c_k(n)\right)\right]$$

$$= e_n \cdot w_k(n) \cdot \phi\{x(n), c_k(n), \sigma_k(n)\} \cdot \left[\frac{1}{2\sigma_k^2(n)}\left(0 - 2(x(n)) + 2c_k(n)\right)\right]$$

$$\Rightarrow \boxed{\frac{\partial}{\partial c_k} J(n)\Big|_{c_k = c_k(n)} = -e_n \cdot w_k(n) \cdot \phi\{x(n), c_k(n), \sigma_k(n)\} \cdot \left[\frac{x(n) - c_k(n)}{\sigma_k^2(n)}\right]} \quad \text{— (d)}$$

We know the update equation:

$$c_k(n+1) = c_k(n) - \mu_c \frac{\partial}{\partial c_k} J(n)\Big|_{c_k = c_k(n)}$$

$$\Rightarrow \boxed{c_k(n+1) = c_k(n) + \mu_c \frac{e_n \cdot w_k(n)}{\sigma_k^2(n)} \cdot \phi\{x(n), c_k(n), \sigma_k(n)\} \cdot [x(n) - c_k(n)]}$$

↳ using eq. (d)

Hence proved //

Derivation 3:

Taking the derivative of Equation '5' w.r.t. $\sigma_k(n)$:

$$\frac{\partial}{\partial \sigma_k(n)}(J(n)) = \frac{2}{2}\left[y_d(n) - \sum_{k=1}^{N} w_k(n) \cdot \exp\left(\frac{-\|x(n) - c_k(n)\|^2}{2\sigma_k^2(n)}\right)\right] \cdot \frac{\partial}{\partial \sigma_k}\left[-\sum_{k=1}^{N} w_k(n) \cdot \exp\left(\frac{-\|x(n) - c_k(n)\|^2}{2\sigma_k^2(n)}\right)\right]$$

$$= -e_n \cdot \frac{\partial}{\partial \sigma_k(n)}\left[\sum_{k=1}^{N} w_k(n) \cdot \exp\left(\frac{-\|x(n) - c_k(n)\|^2}{2\sigma_k^2(n)}\right)\right]$$

$$\Rightarrow \frac{\partial}{\partial \sigma_k} J(n)\Big|_{\sigma_k = \sigma_k(n)} = -e_n \cdot w_k(n) \cdot \frac{\partial}{\partial \sigma_k(n)}\left[\exp\left(\frac{-\|x(n) - c_k(n)\|^2}{2\sigma_k^2(n)}\right)\right]$$

$$= -e_n \cdot w_k(n) \cdot \exp\left(\frac{-\|x(n) - c_k(n)\|^2}{2\sigma_k^2(n)}\right) \cdot \left[\frac{-\|x(n) - c_k(n)\|^2}{2} \cdot \left(\frac{-2}{\sigma_k^3(n)}\right)\right]$$

$$\boxed{\frac{\partial}{\partial \sigma_k} J(n)\Big|_{\sigma_k = \sigma_k(n)} = -\frac{e_n \cdot w_k(n)}{\sigma_k^3(n)} \cdot \phi\{x(n), c_k(n), \sigma_k(n)\} \cdot \|x(n) - c_k(n)\|^2} \quad \text{— (e)}$$

We know the update equation:

$$\sigma_k(n+1) = \sigma_k(n) - \mu_\sigma \frac{\partial}{\partial \sigma_k} \cdot J(n) \Big|_{\sigma_k = \sigma_k(n)}$$

$$\Rightarrow \boxed{\sigma_k(n+1) = \sigma_k(n) + \mu_\sigma \frac{e_n \cdot w_k(n)}{\sigma_k^3(n)} \cdot \phi\{x(n), c_k(n), \sigma_k(n)\} \cdot ||x(n) - c_k(n)||^2}$$

↳ Using equation (e)

Hence proved //

Question 2

The article discusses and builds upon a famous associative memory model called the Hopfield Network. The network is energy-based and uses an associative memory model in attempt to mimic the human brain. The Hopfield Network is comprised of neurons and synaptic conduits. The neurons represent states that are either ON or OFF ($+1$ or -1) and the synaptic conduits are each assigned a weight value. Since this is a neural network, it inherently has a high degree of parallelism and is capable of distributed storage of information. The model is also robust, consisting of many basic elements which are of low computational complexity. As this is a neural network it does require significant computational power. The goal of the network is to give it a probe (a starting state) in hopes that it will reconstruct the appropriate fundamental memory. The states can be changed either asynchronously or synchronously. To effectively implement a Hopfield network, we want the fundamental memories to be attractors, ensuring that probes are being mapped to fundamental memories of similar states. Fundamental memories are fixed points (usually) in a networks state space and should "pull in" similar states. For the fixed points to be attractors we must strategically design our network. In the case of the Hopfield Network our strategic design must be implemented through the fixed connection weights.

To strategically select connection weights Hopfield had a method for memory encoding. Assume we have set of m memories $\boldsymbol{x} = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(m)}\}$ with each memory having n components. We construct a $n \times n$ matrix for each memory ($\boldsymbol{T}_{\alpha}$) and ensure 0 diagonal by subtracting the identity matrix (neurons are not connected to themselves). Then we summate each memory matrix to get a final connection matrix ($\boldsymbol{T}$). This allows us to rely on the connection matrix to reconstruct memories from a probe. This technique works best when the number of memories (m) is significantly smaller than the number of neurons (n).

$$\boldsymbol{T}_{\alpha} = \boldsymbol{x}^{(\alpha)}\left(\boldsymbol{x}^{(\alpha)}\right)^{T} - \boldsymbol{I}_{n}$$

$$\boldsymbol{T} = \sum_{\alpha=1}^{m} \boldsymbol{T}_{\alpha}$$

There are other possible techniques that can be used for constructing connection matrices. One way is to have fundamental memories be exactly ordinary eigenvectors of the connection matrix with positive eigen values. Both Abu-Mostafa and St. Jacques have shown that the maximum capacity for a memory of length n is n. In their case the memories don't require a radius of attraction of more than 0.

Previously mentioned was the notion of asynchronously updating states of a Hopfield Network. These adjustments are made using the connection matrix $\boldsymbol{T}$ and the current state vector $\boldsymbol{x}^{(n)}$. The next state is calculated as $\boldsymbol{T}\boldsymbol{x}^{(n)}$ which will give us some vector $\boldsymbol{x}'$. The term asynchronous refers to the fact we will only allow one component to change when moving from the current state vector $\boldsymbol{x}^{(n)}$ to the next state vector $\boldsymbol{x}^{(n+1)}$. For example, if $\boldsymbol{x}^{(1)} = (+,+,+,+,+)^{T}$ and $\boldsymbol{x}' = (-,-,+,+,+)^{T}$ we will only allow one component to change which will result in a next state vector of $\boldsymbol{x}^{(n+1)} = (-,+,+,+,+)^{T}$ or $\boldsymbol{x}^{(n+1)} = (+,-,+,+,+)^{T}$. This process will be repeated until the there are no sign changes, meaning the network has converged on a fundamental memory.

$$\boldsymbol{x}_{i}^{(k+1)} = \mathbf{sgn}\left(\sum_{j=1}^{n} \boldsymbol{T}_{ij}\boldsymbol{x}_{j}^{(k)}\right) = \begin{cases} +\mathbf{1}, & \mathbf{if} \sum \boldsymbol{T}_{ij}\boldsymbol{x}_{j}^{(k)} \geq \mathbf{0} \\ -\mathbf{1}, & \mathbf{if} \sum \boldsymbol{T}_{ij}\boldsymbol{x}_{j}^{(k)} < \mathbf{0} \end{cases}$$

An appealing characteristic of the Hopfield network is that it is stable. Historically Recurrent Neural Networks (RNNs) have had poor stability, but Hopfield has used constraints to make his Network very stable. If we want to retrieve memories it is ideal to have them be fixed points under the $\mathrm{sgn}(\boldsymbol{Tx})$ mapping. We also want these fixed points to have "pull-in", in the sense that probes will converge on similar fundamental memories. Say there is a probe and that $(1-\rho)n$ of its components are known. This implies that ρn of its components are incorrect, the maximum ρn value is called the *radius of attraction*. The maximum ratio of incorrect values is denoted as ρ, where $0 \leq \rho \leq 1$. Around each fixed point there will be a ρn sphere where probes within the bounds of the sphere will be pulled in to converge on a spheres center point, also known as a fundamental memory. Another testament to the network's stability is that it always converges. This has been proven by Hopfield as he has shown that each asynchronous update to the state always decreases the networks energy. The network has converged once the minimum energy has been reached. If the energy always decreases for every state update, it will always reach the minimum value, guaranteeing it will converge.

When talking about the capacity of a Hopfield Network it is defined as a growth. If we want all memory to be recalled correctly then we have a capacity of $\frac{(1-2\rho)^2}{4} n/\log(n)$. A trick to increase the capacity is to allow a small number of the fundamental memories to be exceptional, meaning they don't have ρn spheres around them. In doing this the capacity can be increased to $\frac{(1-2\rho)^2}{2} n/\log(n)$. It is also known that if we store more than $\frac{n}{2\log n}$ memories that they will most certainly not all be fixed points. As it is ideal for our fundamental memories to be fixed points with radiuses of attraction it will restrict the capacity of the network.

Another way to increase capacity, if ρ is given, is to allow some steps in the wrong direction. This would increase the capacity to be $\frac{n}{4\log n}$. In addition, we could further increase capacity by allowing a small fraction of fundamental memories to be exceptional. This doubles the capacity to be $\frac{n}{2\log n}$.

The capacity can also be increased by allowing a small number of components to be incorrect. If we denote the error as ϵ we effectively create spheres of radius ϵn. The boundary of these spheres become new stable points for the memory. In this scenario the only fixed points of concern are the extraneous points comprised of the ϵn sphere boundaries. Introducing this error threshold allows the capacity to become linear.

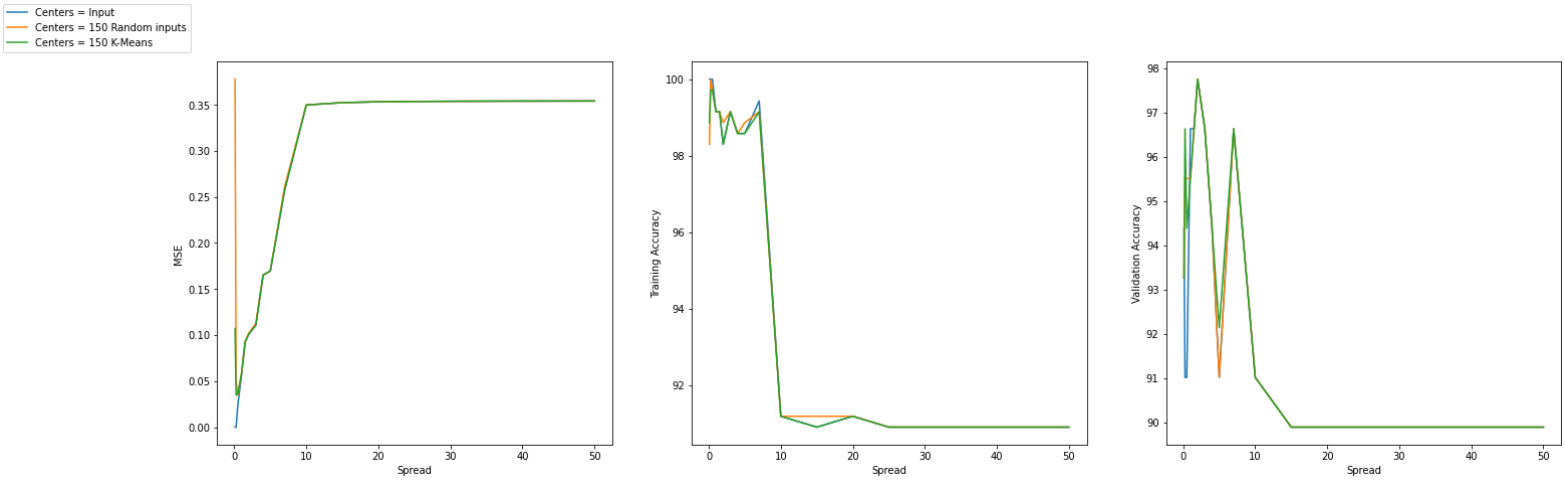
The article concludes by presenting the equation $F_n \sim (1.0505)2^{0.2874\pi}$ where the number of fixed points of a model is asymptotic to F_n. This assumes that connection matrix is symmetrical, has zeros down the diagonal, and has entries that are independent identically distributed zero-mean gaussian random variables. This asymptotic result is found to have a contradiction when m is equivalent to a constant times n. The article leaves off with the ongoing problem of finding an actual constant x in our asymptotic expression $F_n \sim (1.0505)2^{x\pi}$.

Moreover, the "*Capacity of the Hopfield Associative Memory*" article covers a wide variety of topics pertaining to Hopfield Networks. Covering the fundamental mechanics as well as discussing numerous techniques for increasing the model's capacity. It is very clear that there are modifications that when made drastically increase the capacity of the Hopfield Network, but this will come at the cost of the model's performance.

References

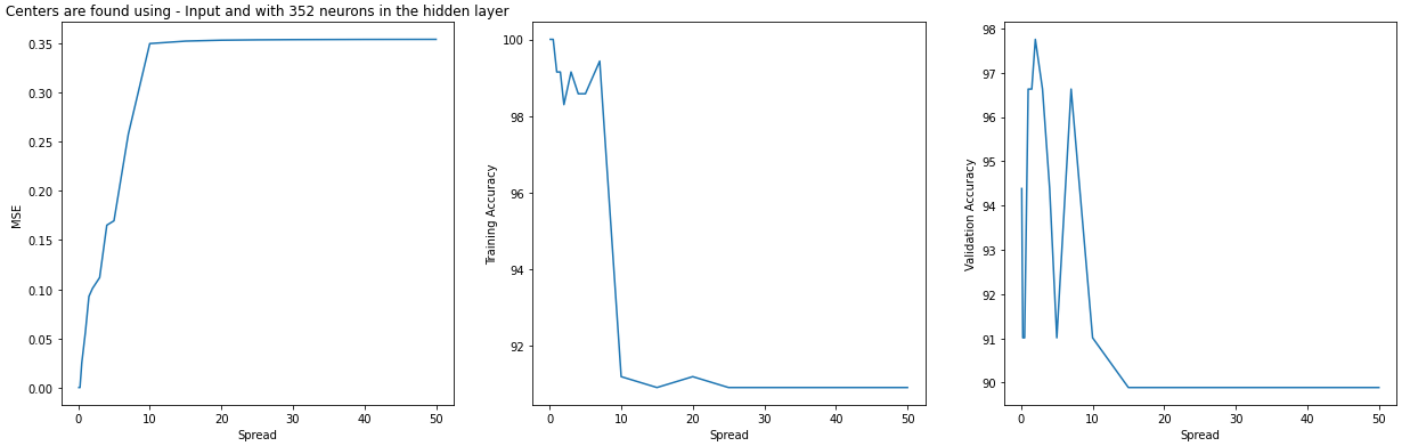
[1] R.McEliece, "The Capacity of the Hopfield Associative Memory", *IEEE Transactions On Information Theory*, July 1987

Question 3



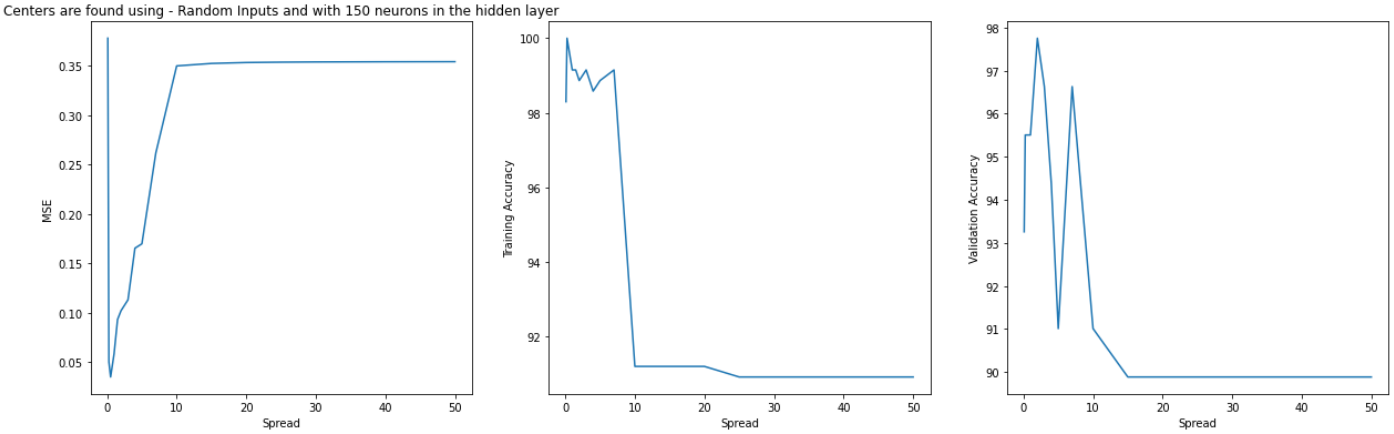
Following are the results achieved for different values of Spread:

Part 1] Centers = 352 Input Points



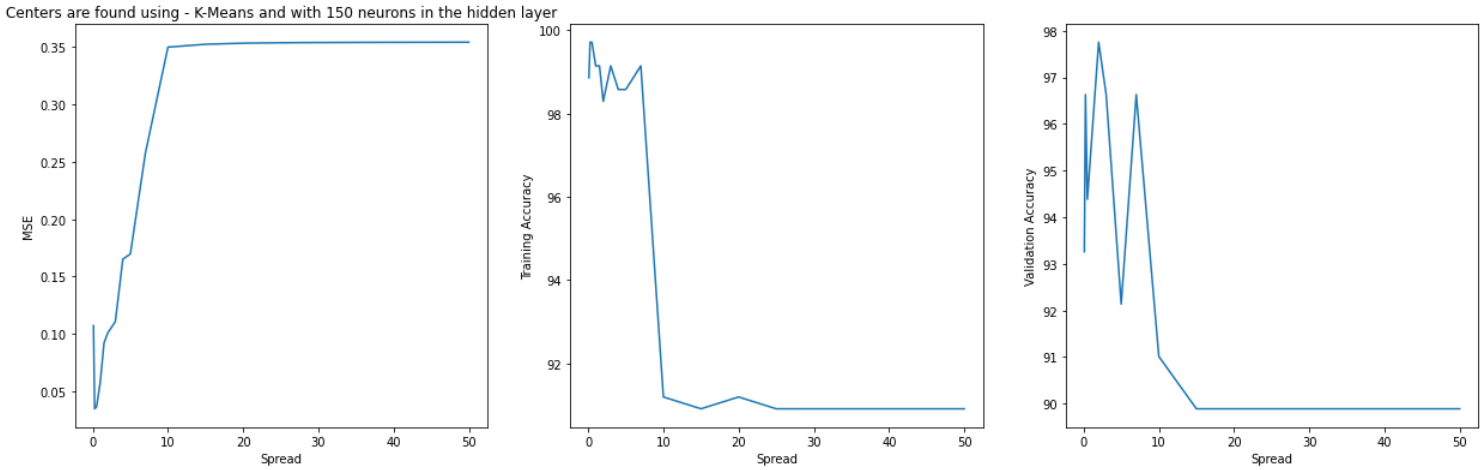
- Mean square Error keeps increasing with increase in 'Spread' value & after 'spread = 10' stays almost constant
- Training Accuracies have very little fluctuations initially, and then suddenly drop to around 90 for spread >= 10.
- Testing (Validation) accuracies follow a trend similar to training, except that the initial fluctuations are large. [The large initial fluctuations are expected, since the validation set may hold new inputs (which were not exposed earlier, during training)]

Part 2].b] Centers = 150 Random Input Points



- Mean square Error keeps increasing with increase in 'Spread' value (With the exception of 'spread = 0.1 to 0.5' where it actually drops between the 2 points) & after 'spread = 10' stays almost constant
- Training Accuracies – Results are again similar to those achieved in Part-1, except for 'spread = 0.1 to 0.5', where it actually goes up by a small margin.
- Testing (Validation) accuracies – Results are similar to those achieved in Part-1, although not exactly the same.

Part 2].c] Centers = 150 Points, Found using K-Means



- The results are similar to those achieved in Part 2.a] and for 'spread = 0.1 to 0.5' the change is relatively smaller in case of all 3 parameters.

Comparison in performance of the network:

- As is evident from the graphs, there is very little difference in performance between the 3 different configurations of centers.
- All the performance parameters (Mean Square Error (MSE), Training Accuracy in %, Validation Accuracy in %) are very similar in each of the configurations.
- A noticeable difference that we can see between the configurations is only for the initial values of spread. Otherwise, the results are very similar.

Comment on Spread Value:

- The above results show the importance of 'Spread' value in Radial Basis Function Neural Networks.
- The performance of the network largely varies depending on the spread value, compared to other parameters.
- A spread value between '1' and '2' appears to provide an efficient result, with good accuracies and low cumulative error.

Following Configuration has been used:

- spreads = [0.1, 0.25, 0.5, 1, 1.5, 2, 3, 4, 5, 7, 10, 15, 20, 25, 30, 40, 50]
- Radial Function = "Gaussian Kernel", Loss Function = "Mean Square Error"

Changing the Spread Values:

The choice of spread values can be changed by changing the value in '**spreads**' parameter inside the constructor call for **assignment** under **'Train and observe the Network'** section.

```
# -------------- Prepare Input Data for the network -----------------
a = assignment(center_methods = ["Input", "Random Inputs", "K-Means"], spreads = [0.1, 0.25, 0.5, 1, 1.5, 2, 3, 4, 5,
7, 10, 15, 20, 25, 30, 40, 50], test_validation_division = 8/10)
a.generate_input_data()
a.train_and_observe()
```

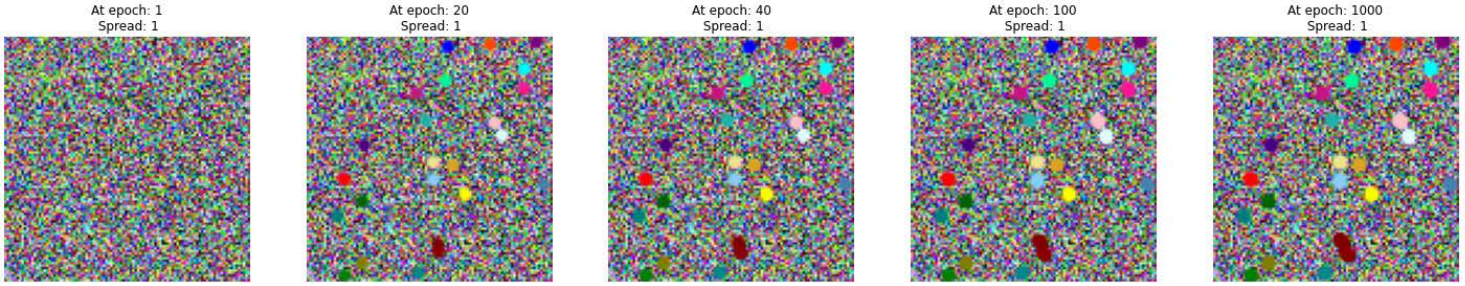
- Simply change the value of '**spreads**'

- **'center_methods'** parameter contains the name of the methods that should be implemented in sequence for the network configuration for Part 1, Part 2a, Part 2b respectively.

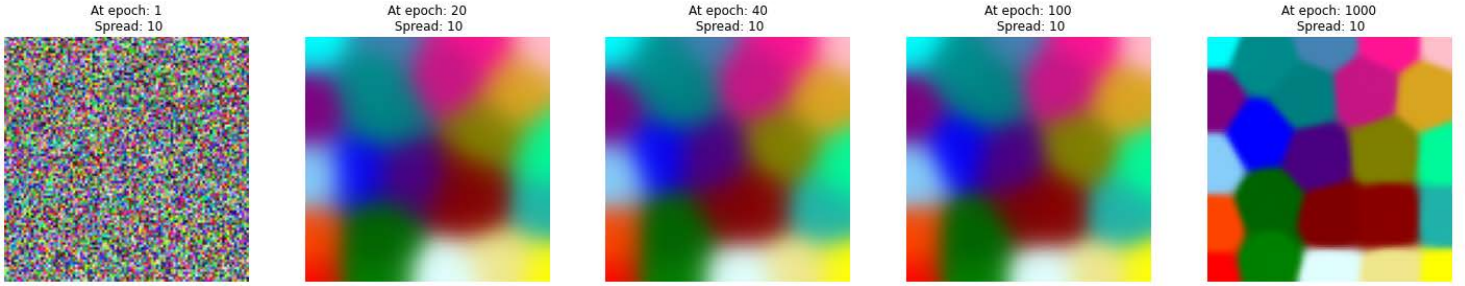
Question 4

Following are the results achieved for different values of Spread:

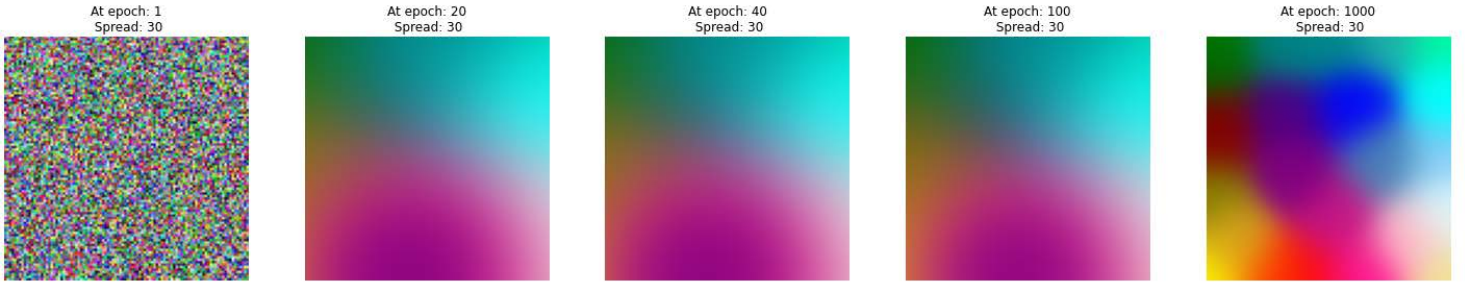
Spread = 1:



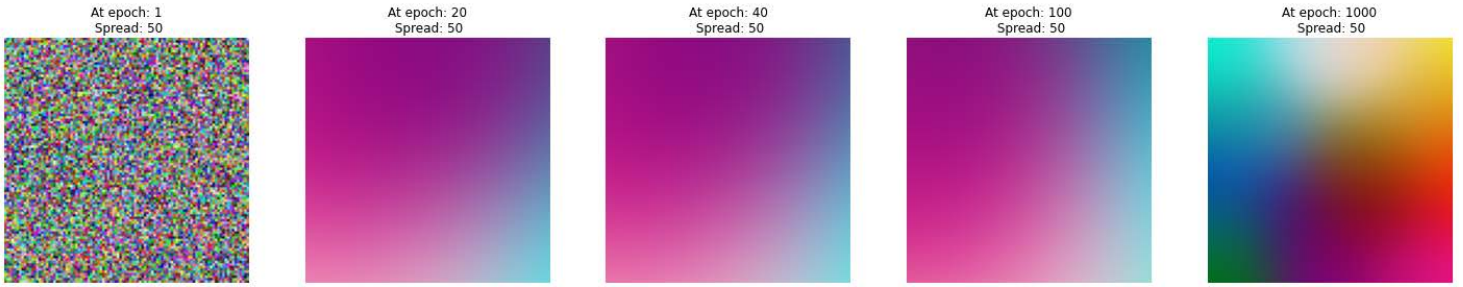
Spread = 10:



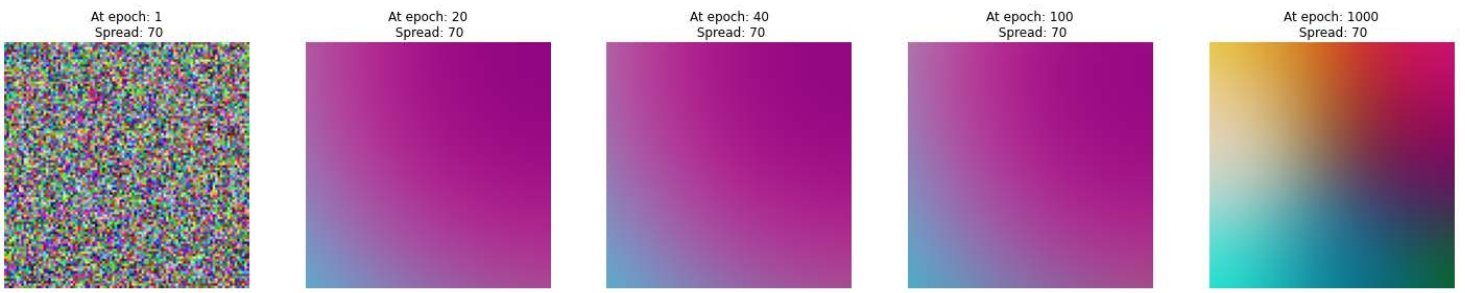
Spread = 30:



Spread = 50:



Spread = 70:



From the above results, we can make the following observations:

- With a low spread value of '1', the network is able to form small clusters with well-formed boundaries in as low as 20 epochs.
  - The clusters are very small, as is expected – due to the smaller neighbourhood size.
  - We can look at it as a case of strong specialization (even at a low number of epochs).
- With Spread = 10, we see good boundaries forming - from 20 epoch onwards, and these boundaries become more clear at epoch=1000.
  - The clusters still maintain a good amount of specialization of colors, at epoch = 1000.
- As we further increase the values of spread, we observe that the network takes longer to form clusters with clear boundaries.
  - The clusters are also larger in size due to a bigger neighbourhood.
  - We can say that the clusters are more generalized, representing a larger sample space.
- A similar trend continues as we keep increasing the value of spread. In fact, at spread values '50' and '70', we can hardly make out any clear boundaries.

From these observations, we can draw the following conclusions:

1] With lower spread values, we achieve clear boundaries in less number of epochs, forming small specialized clusters.

2] With each increase in spread value, we move towards generalization, as we achieve comparatively bigger clusters, representing a larger sample space.
- But note that, if we run the network for relatively more number of epochs (E.g. 2000, 3000, etc), we will move towards more specialized clusters, as the neighbourhood parameter would also go down with each new epoch.

Changing the Spread Values:

The choice of spread values can be changed by changing the value in '`test_sigmas_init`' parameter inside the constructor call for **assignment** under **'Train the network – with RGB inputs'** section.

```
a = assignment(alpha_init = 0.8, test_sigmas_init = [1, 10, 30, 50, 70], max_epochs =
1000, checkpoints_epoch = [20, 40, 100, 1000])
a.generate_input_data()
a.train_and_observe()
```

Simply change the values of '**`test_sigmas_init`**'**.**