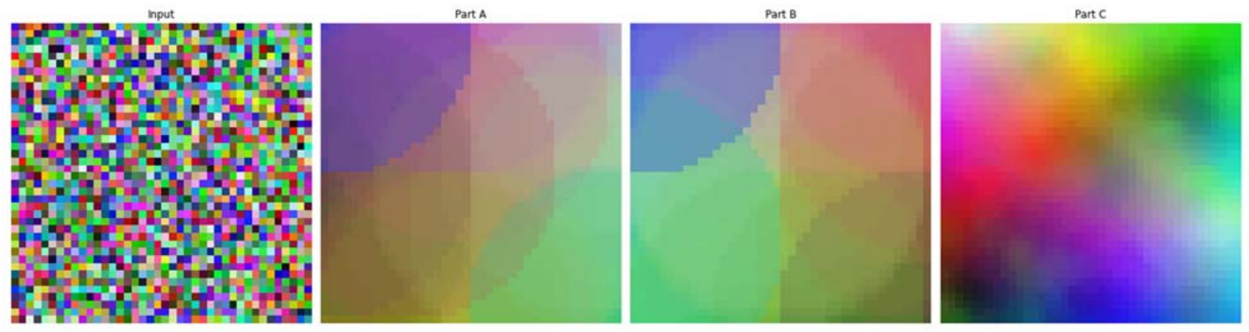
Computational Intelligence Assignment 2 Sara Kodeiri 96521443

Q1 – Kohonen SOFM

I have coded this question in a way that different dimensions can be tested. In the third cell, by setting the “dim” variable, different inputs can be checked. First, a random input is generated which is a 40 * 40 * 3 matrix, shown as an image. After normalizing the values and showing the input, the training starts for each subsection of this question. We keep the initial values of the learning rate (alpha) and radius because we’ll be needing them later. In each epoch, we do the following steps:

1- Update the learning rate or radius if needed.
2- Select an RGB vector (each vector is selected three times overall).
3- Find the best matching unit (neuron) for it.
4- Update the Kohonen Map according to the BMU coordinates we’ve found.

For finding the BMU, we iterate the whole map to find the neuron that has the least distance from the input vector. We then return its coordinates. Using these coordinates, the map is then updated. For updating the map, we choose the neurons that are within the radius of the BMU neuron. These are the only ones that get updated. According to the influence function and learning rate, the neuron weights get updated for the next epoch. The influence is taken into account because the neurons closer to BMU must change more than the ones that are far from it. All the calculations and formulas of this question have been derived from the slides.
The main variable that influences the output, is the radius not being a constant. If we don’t update it, whether the learning rate is constant or not, we are still only taking some fixed neurons into account and it causes **distortion** in our kohonen map (Parts A and B). But if the radius changes over time and decays, no distortion will happen. First the general organizing happens, and then it gets more specific after each epoch.
When the learning rate is constant, there is always a possibility of **not converging**. This problem can be fixed by choosing a learning rate that is “just right”, which will take a lot of experiments and trials. Another way is to just update it after each epoch. Part B has converged better than Part A, but because the radiuses are constants in both situations, the final result isn’t satisfactory. The constant radius can be seen in the results.



Q2 – RBF

For training, I have used 5000 points ranging from -50 to 50. To keep the same ratio, I have used 400 points from -4 to 4. The MLP model I've used has 4 layers and I chose ReLU as their activation function because it also supports non-positive values (for example, sigmoid wouldn't work for this function. Check the third link). Mean Squared Error is also a better loss function when it comes to dealing with continuous functions like sin(x). The epoch count and batch size has been obtained using trial and error.

MLPs are advantageous over RBFs when the underlying characteristic feature of data is embedded deeply inside very **high dimensional** data sets. For example, in image recognition, features depicting the key information about the image is hidden inside tens of thousands of pixels. For such training examples, the redundant features are filtered as the information progress through the stack of hidden layers in MLPs, and as a result, better performance is achieved.

Having only one hidden layer, RBFs have much faster convergence rate as compared to MLP. For low dimensional data where deep feature extraction is not required (such as sin(x)) and results are directly correlated with the component of input vectors, then RBF network is usually preferred over MLP. RBFs are universal approximators, and unlike most machine learning models RBF is a robust learning model.

Links Used

AI-Junkie SOM Tutorial
Super Data Science: The Ultimate Guide to SOM's
MLP for sin(x)
Flatten List