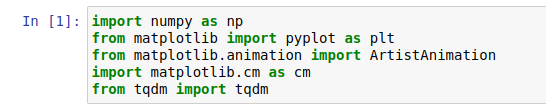
**HOPFIELD NETWORK:**

This assignment looks for a possible neural mechanism underlying such high-level brain functioning as association and concept-formation. Primitive neural models of association and concept-formation are presented, which will elucidate the distributed and multiply superposed manner of retaining knowledge in the brain. These models are subject to two rules of self-organization of synaptic weights, orthogonal and covariance learning. The convergence of self-organization is proved, and the characteristics of these learning rules are shown. The performances, especially the noise immunity, of the association net and concept-formation net are analyzed.

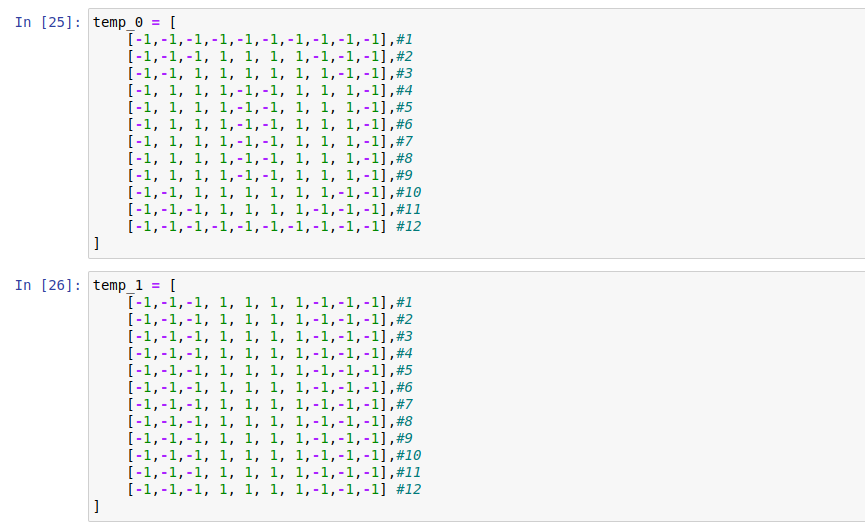
**CODE EXPLAINATION:**

* **Importing Libraries:**



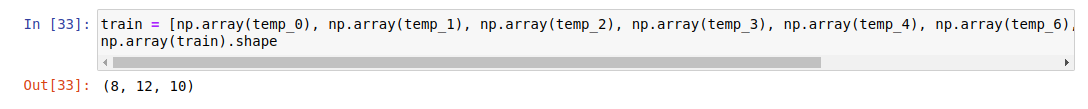
The first set was to import all the important libraries and use them for making our model. We do not have a fancy library that can help us to make a Hopfield network, but we can build it from scratch using "NUMPY.” We can use matplot to plot certain things and sklearn threshold for the pre-processing of the Training and Testing Data. We can use “TDQM” for making the loading lines for Training and Testing purposes.

* **Creating DATATSET:**



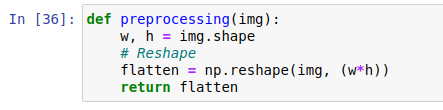
For this step first we created our own dataset so we can do what we are required for the future. The dimensions are as requested 12\*10.

* **Loading DATASET:**

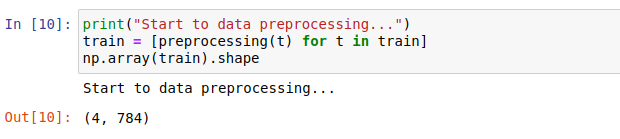


So now combine all the required patterns in an array for later use in the “HOPFIELD” neural network. My model is trained in all the required patterns like “0, 1, 2, 3, 4, 6, 9 and square”.

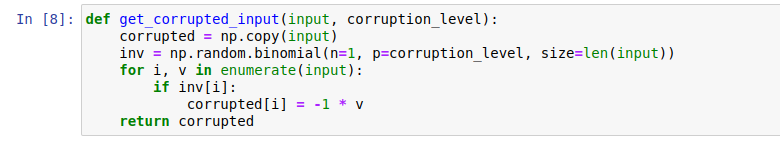
* **PREPROCESSING DATA:**



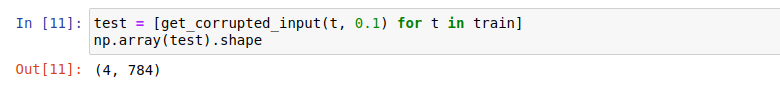
Now to train the model we must preprocess the data. This function does the working of the pre-processing and for that purpose we will convert our whole patterns into a flat “1D” vectors so that they can be processed easily.



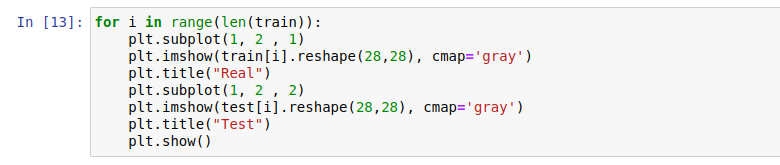
* **CORRUPTING DATA:**



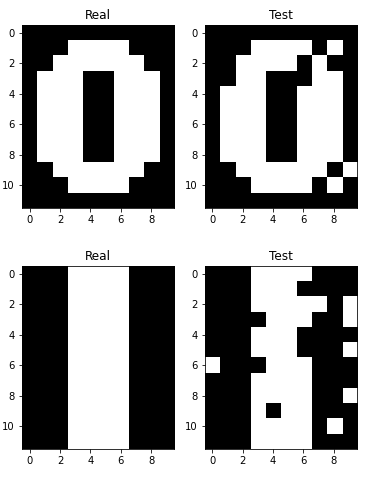
We have the “Real Data (Training DATA)” but where is our “Testing Data”? For that we will have to add “Noice” and irregular values so that we can see the working of the “HOPFILED Model.” This function will generate a Noice for each pixel depending on its “corruption level.”

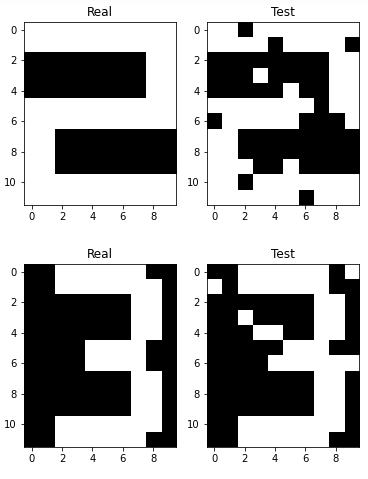


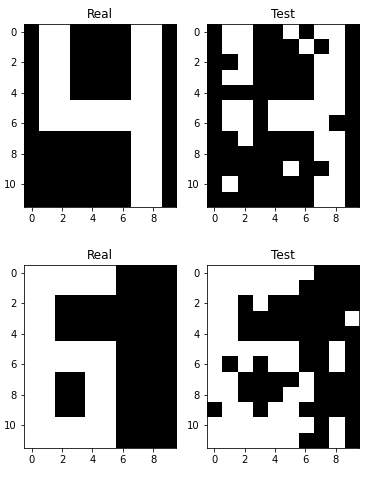
* **RESULT OF CORRUPTING DATA:**

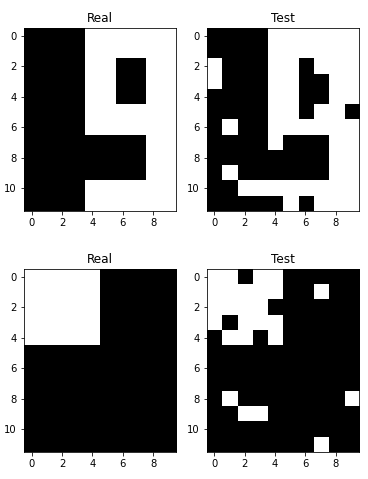


These lines help us plot the real and corrupted values so we can see what is happening.

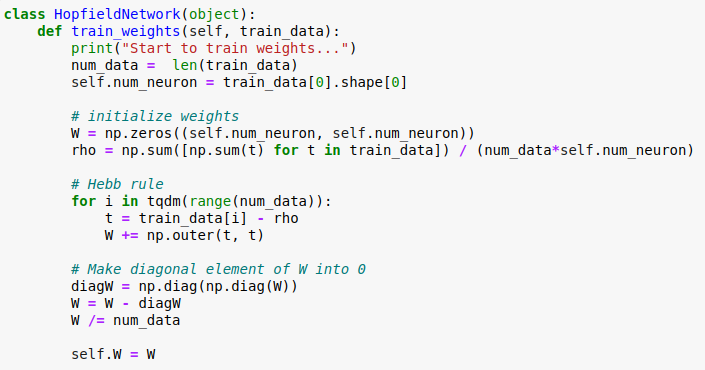






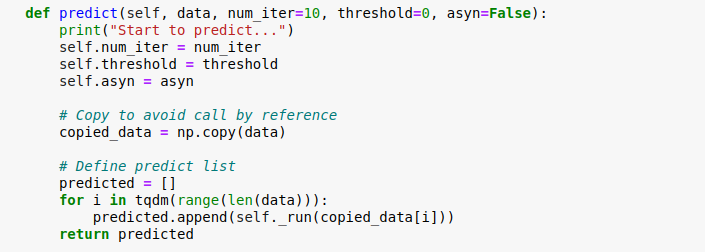


* **HOPFIELD NETWORK:**



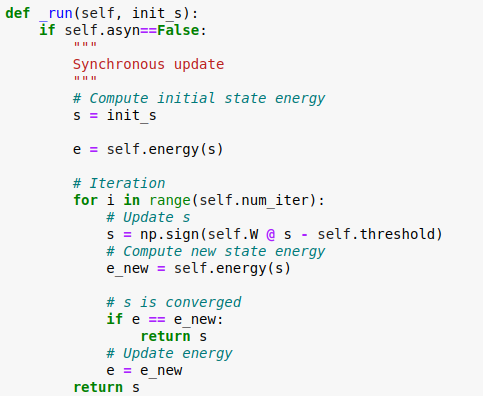
This class is now going to help us by initializing the model. Firstly, we must create an array of dimensions equal to the Training data because only after then will we be able to use this network. Secondly, we will initialize initial weights by Hebb rule then after that we will remove connection the “Diagonals” because a neuron is not connected to itself.

* **PREDICTION:**

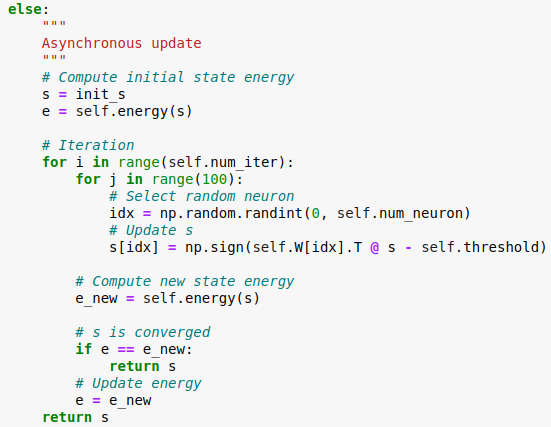


So, for prediction the model will predict the data one by one. The main code is in “\_run” this is just the initial step to set values before running the code.

* **SYNCHRONOUS and ASYNCHRONOUS UPDATE:**

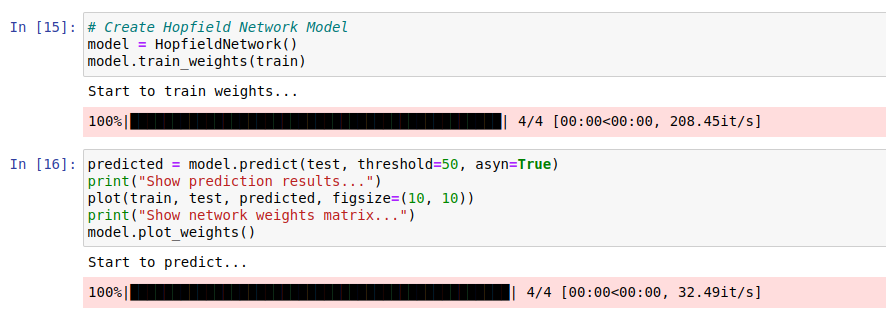


This part is running the model in “Synchronous update” in this all the neurons in the model will update to a new to a new level in a single loop.



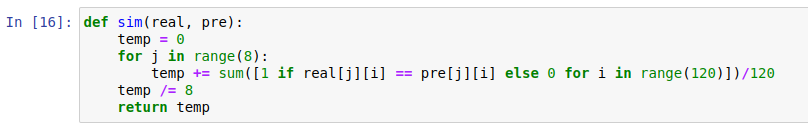
Whereas this part is running the model in “Asynchronous update” in this only one of the neurons in the model will update to a new to a new level and we will deal with others one by one.

* **MNIST AND HOPFIELD:**



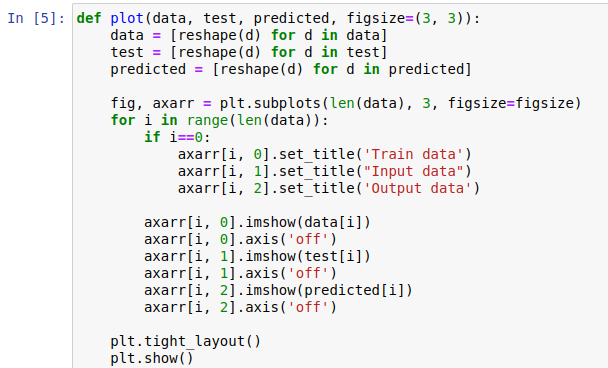
Just create an instant of the HOPFIELD class and call training and prediction function.

* **EVALUATION FUNCTION:**

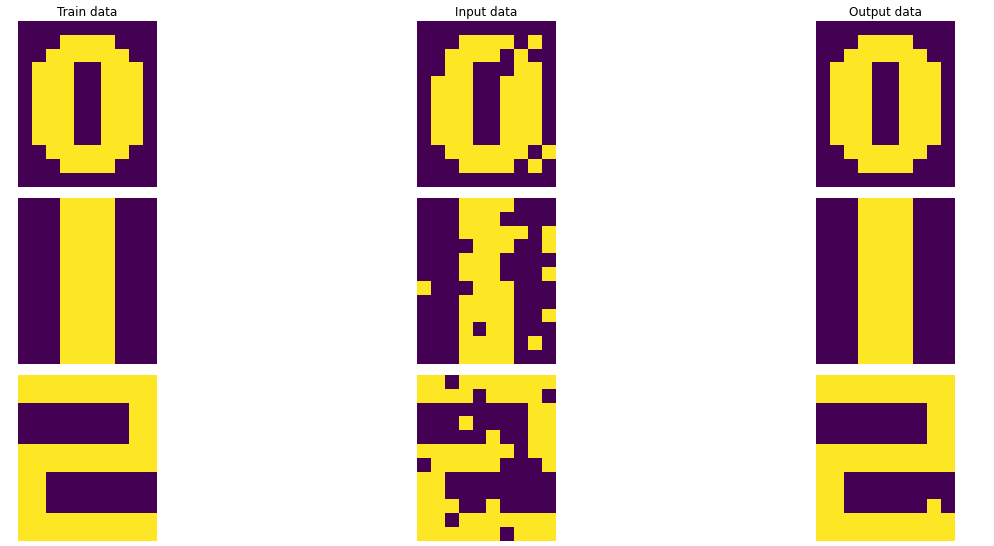


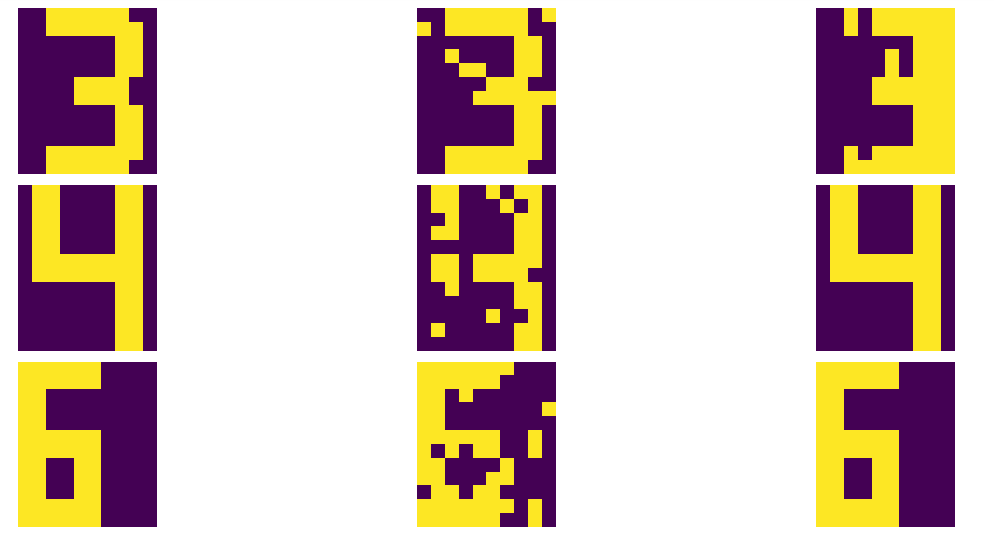
As our evaluation function I choose “SIMILARITY” and “CONFUSION\_MERTICE” function because that way we will be able to check how did our model performed. “SIM” will get **how many of our pixel values are same and will return the similarity value between “Real” and “Predicted” values.** Whereas the “Confusion Mertice” will return **difference between each pixel value and return the Mertice of “Real” and “Predicted” values.**

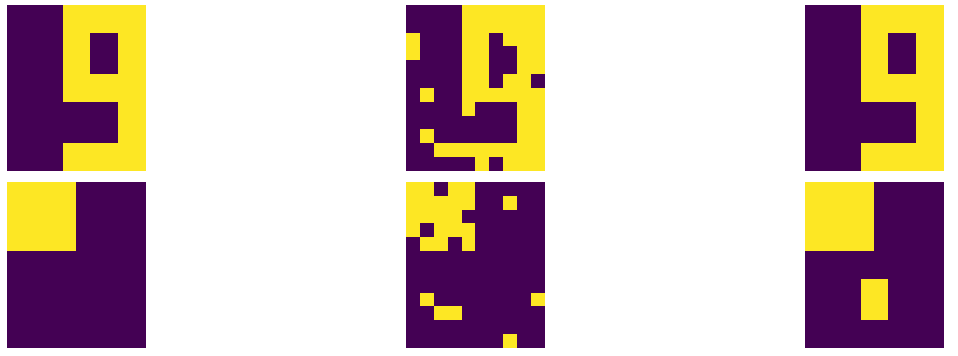
* **RESULT:**



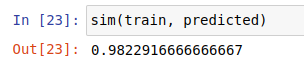
This is to display the result, let us see what we have achieved. This is the result of “**Asynchronous update**”:



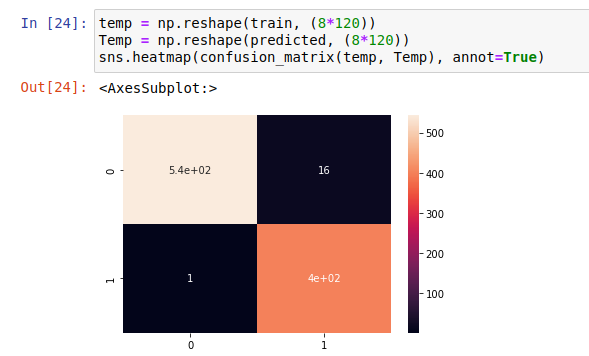




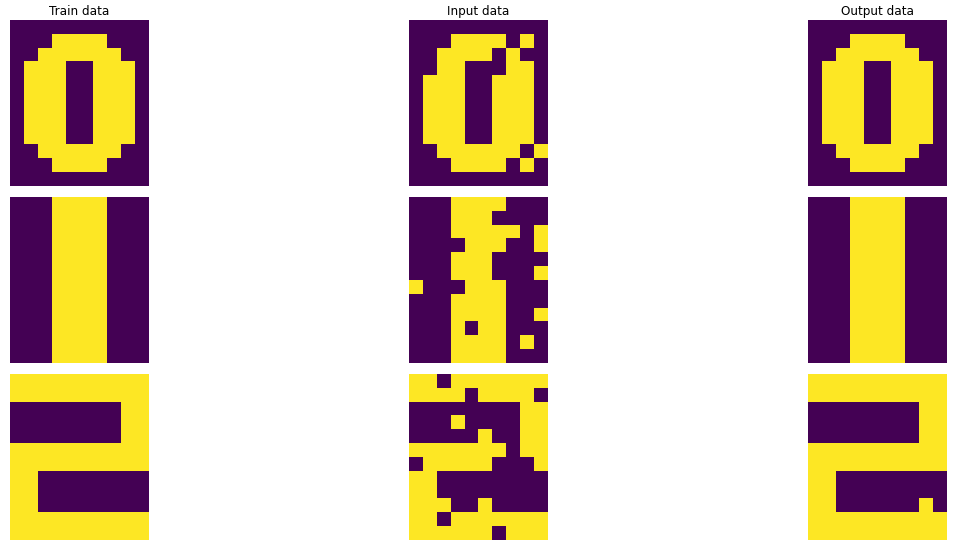
**SIM:**

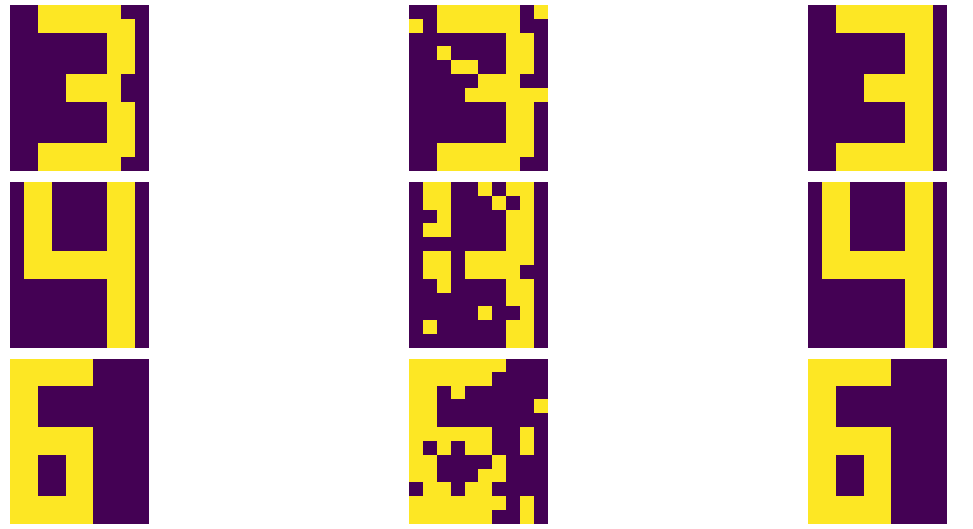


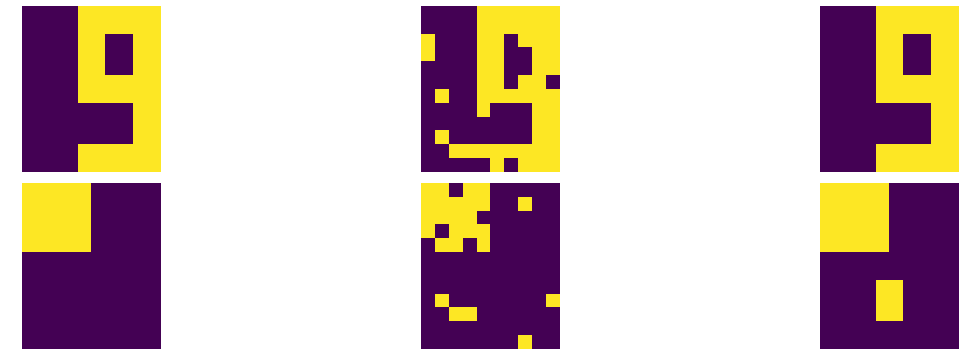
**CONFUSION MERTICE:**



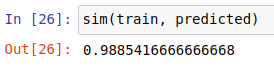
The result of “**Synchronous update**”:







**SIM:**



**CONFUSION MERTICE:**

