# 1. XOR problem in Tensorflow Playground

1. Ksbv d
2. Dfjvbsd
3. D vjzb

# Feedforward multilayer perceptron

1. Vbdsk
2. Vdjv
3. Vdf
4. Sjvnf
5. Jkdvn
6. Multiclass neural network classifier
7. **Implementation and convergence criterion:**

The multiclass neural network classifier with one hidden layer is implemented using python. The implementation has two parts –

(1) a class named MulticlassNN.py is implemented that contains the definition of training model and

(2) a train-test implementation named TrainTestNN.ipynb that imports training model MulticlassNN.py to run the training and test data (dev data) with different configurations.

In Neural network model,

* no. of layers (input, hidden and output), no. of neurons per hidden layer, activation function and cost function are initialized.
* An input layer to hidden layer and a hidden layer amongst themselves are connected using activation functions which implies, output of layer is input to the next layer after activation function is acted upon each set of inputs until it reaches final output layer. I have used three activation functions – sigmoid, tanh and softmax, implemented using NumPy package’s inbuilt functions – tanh, exp, sum and divide.
* The training model has three steps, viz. forward pass, error computation and backward pass. Each instance will go through activation function while forward pass and error is computed at output. Based on error, it will be back propagated by taking the derivative deltas w.r.t activation and multiplying it with learning rate to decide on size of learning step, i.e., calculating the gradient descent. Weights are then updated by subtracting the gradients from current weight and again passed forward to check for error.
* Error is computed based on two cost functions, viz. cross entropy or mean square error.
* Accuracy is then computed based on total correctly classified predicted labels divided by total labels in the input data multiplied by 100.

In train-test implementation,

* intermediary

**Convergence criteria:**

until the classifier converges and a decent accuracy is attained.

1. S db
2. Csjbd
3. Evaluation

# Code







# Accuracy of the classifier on the training set

98.56 %

# Accuracy of the classifier on the test set

88.1 %

# Implementation and convergence criterion

The digit classification problem is implemented using python. The two packages numpy and itertools are used. The dataset ,X containing the matrix of 784 integers that represent the grayscale values of a 28x28 handwritten digit is prepended with 1 in each row and the resulting updated dataset and its corresponding labels, Y are taken into two separate numpy arrays, respectively. Another numpy array, w is created to store weight vectors from w0 to w9, of rows containing 785 zeros each as bias elements.

The dataset is then split into half by taking all odd numbered rows as parts of training data and even numbered rows as part of test data. Similarly, label data is also split into half by taking odd numbered rows as training output and even numbered rows as test output.

The classifier is defined using perceptron algorithm as follows:

For each weight w[i] in the weight vector w, activation for each training instance is calculated as dot product of weight and training data and stored in activation list. The maximum weighted value’s index, index\_max is taken and compared with the corresponding y label for that instance. If index\_max is equal to its corresponding y label, then the algorithm does nothing, but if the index\_max is not equal to its corresponding y label, then the weight of the index\_max is updated by subtracting the data X from the weight of the index\_max, and the weight of the corresponding y label is updated by adding the data X to the weight of the corresponding y label.

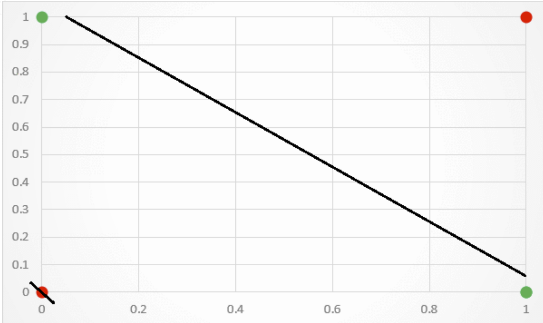
Convergence criterion: The above steps when repeated over 100-500 times increases the weight of true labels and decreases the weight of incorrect prediction of labels and eventually converges to correct classification.

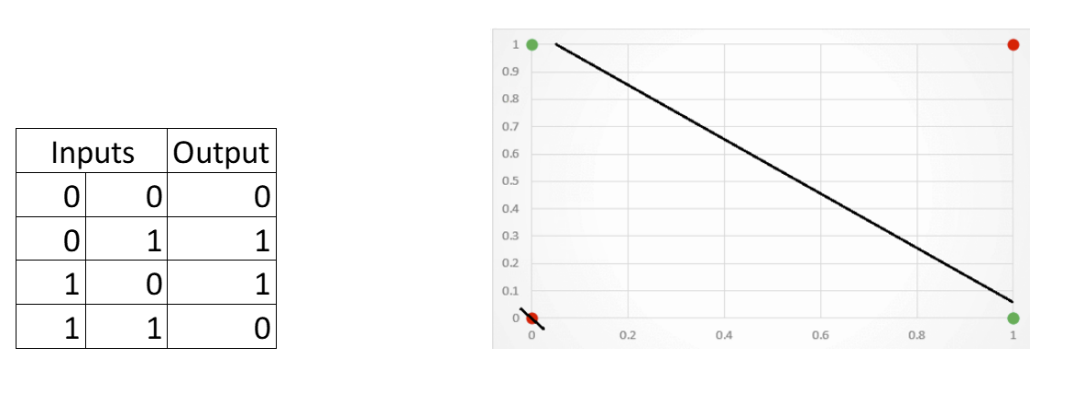
After the data is trained, the accuracy of the classifier for training data is calculated as follows:

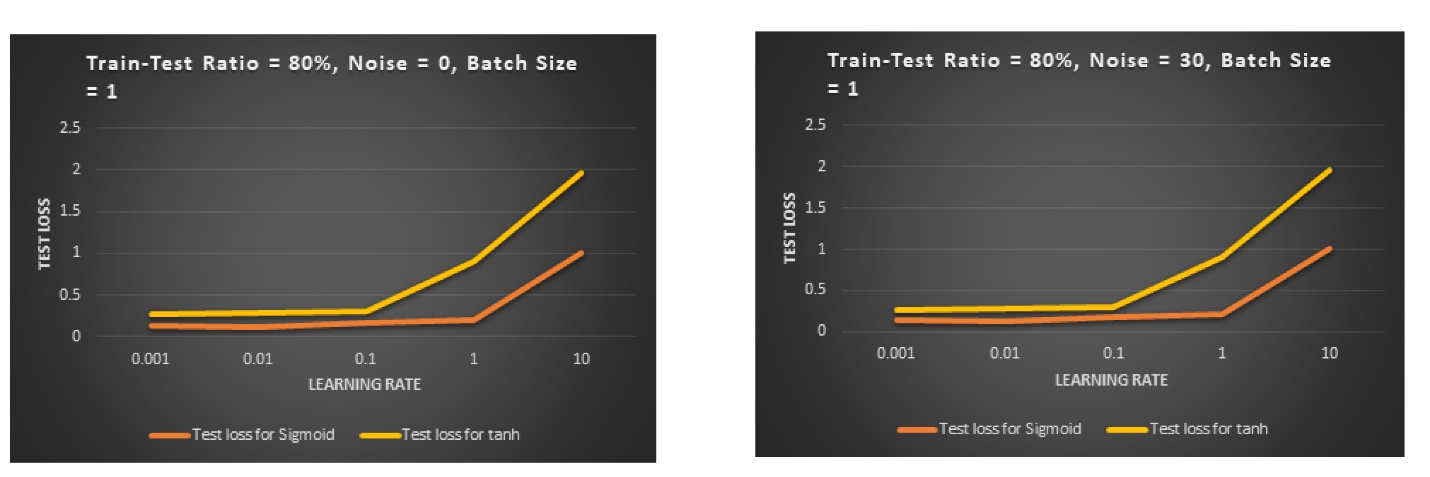
For all 5000 rows of training dataset, the corresponding y training label is subtracted from maximum weighted value’s index ‘index\_max‘ for that data X, and stored in a list called TrainingResult. This gives 0 for all the correctly classified data and a non-zero value for all incorrectly classified data. The accuracy is then calculated by counting all no zero values and subtracting it with total data divided by total data which is 5000 multiplied by 100 to give accuracy in percentage.

Similary, for test data, the accuracy of the classifier is calculated as follows:

For all 5000 rows of test dataset, the corresponding y test label is subtracted from maximum weighted value’s index ‘index\_max‘ for that data X, and stored in a list called TestResult. This gives 0 for all the correctly classified data and a non-zero value for all incorrectly classified data. The accuracy is then calculated by counting all no zero values and subtracting it with total data divided by total data which is 5000 multiplied by 100 to give accuracy in percentage.

 ****





References:

* <https://stackoverflow.com/questions/40346892/how-can-i-import-python-custom-class-in-anaconda-jupyter-notebook>
* <https://www.renom.jp/notebooks/tutorial/neuralnetwork/download_mnist/notebook.html>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Train to Test data Ratio = 80%** | | | **Batch size = 1** | |
| **Epochs = 5** | | | **No. of hidden layers = 1** | |
| **No. of neurons per hidden layer** | **Learning Rate** | **Activation Function** | **Cost function** | **Test loss** |
| 20 | 0.01 | tanh | Cross Entropy | 0% |







