Lab Assignment 3

Notebook:	Machine Learning Lab		
Created:	17/11/2018 20:28	**Updated:**	20/11/2018 06:27
Author:	Prateek Chanda		

Classification of MNIST Data using Radial Basis Function

In this assignment, we used the RBF for classifying MNIST data and perform prediction on the test data. Also we trained a linear classifier to check the performance of prediction.

Dataset

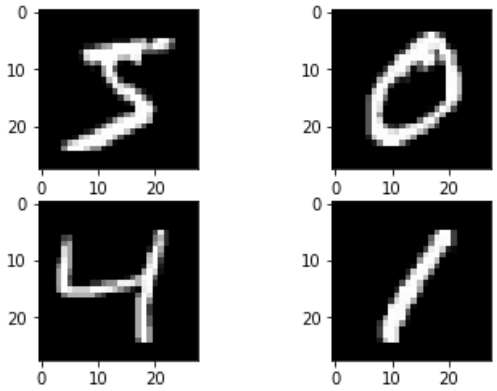
First we loaded the mnist data which contains class labels for digits 0-9 using

```
data = tf.contrib.learn.datasets.mnist.load_mnist()
```

The preceding method loads the entire MNIST dataset (containing 70K samples) and splits it into train, validation, and test data with 55K, 5K, and 10K samples respectively.

Each split contains one numpy array for images (with shape [sample_size, 784]) and one for labels (with shape [sample_size, 1]).

After plotting few of the images from the data sample, we get a plot something like this.



We can now train a linear model over the MNIST dataset. We will use the SVM Classifier which is one type of linear classifier for our purpose. For two-class classification problems, finding a classifier is equivalent to finding a hyperplane that separates the data as well as possible.

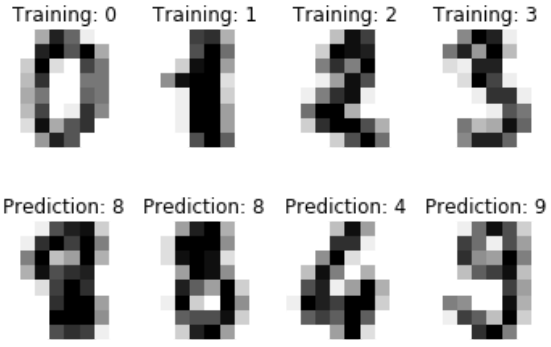
We achieve at the following results.

```
             precision    recall  f1-score   support

          0       1.00      0.99      0.99        88
          1       0.99      0.97      0.98        91
          2       0.99      0.99      0.99        86
          3       0.98      0.87      0.92        91
          4       0.99      0.96      0.97        92
          5       0.95      0.97      0.96        91
          6       0.99      0.99      0.99        91
          7       0.96      0.99      0.97        89
          8       0.94      1.00      0.97        88
          9       0.93      0.98      0.95        92

avg / total       0.97      0.97      0.97       899
```

Corresponding confusion matrix

```
Confusion matrix:
[[87  0  0  0  1  0  0  0  0  0]
 [ 0 88  1  0  0  0  0  0  1  1]
 [ 0  0 85  1  0  0  0  0  0  0]
 [ 0  0  0 79  0  3  0  4  5  0]
 [ 0  0  0  0 88  0  0  0  0  4]
 [ 0  0  0  0  0 88  1  0  0  2]
 [ 0  1  0  0  0  0 90  0  0  0]
 [ 0  0  0  0  0  1  0 88  0  0]
 [ 0  0  0  0  0  0  0  0 88  0]
 [ 0  0  0  1  0  1  0  0  0 90]]
```

After our classification results, we simply try to predict MNIST images from the test data set.

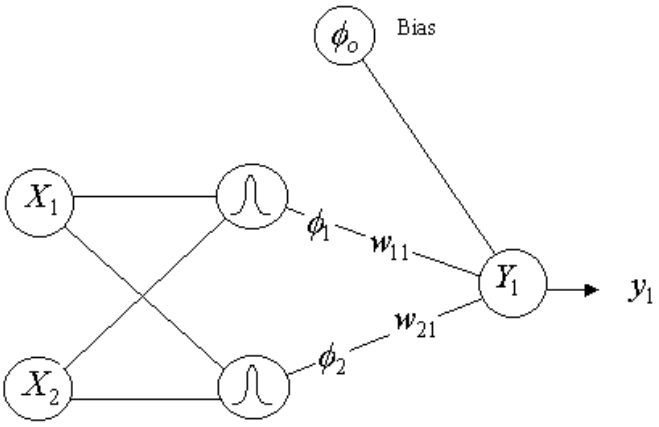


Training Using Radial Basis Function

Radial Basis Neural Network

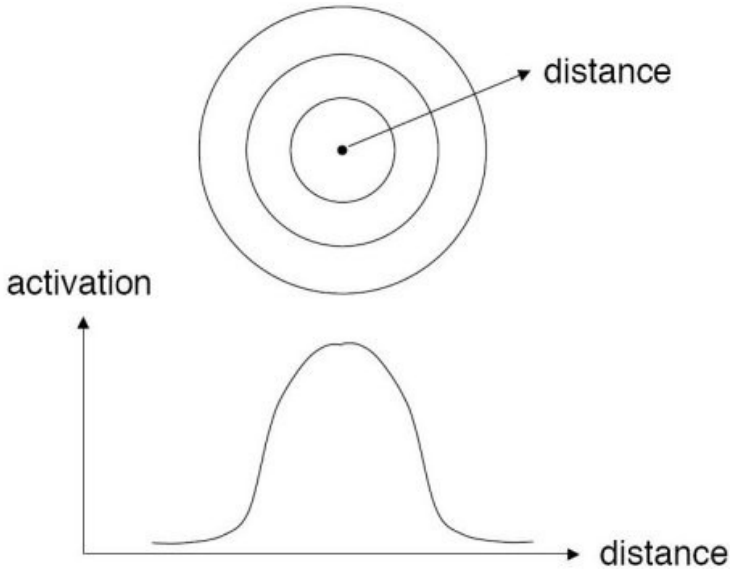
RBNN is composed of input, hidden, and output layer. RBNN is strictly limited to have exactly one hidden layer. We call this hidden layer as feature vector.

- RBNN increases dimenion of feature vector.



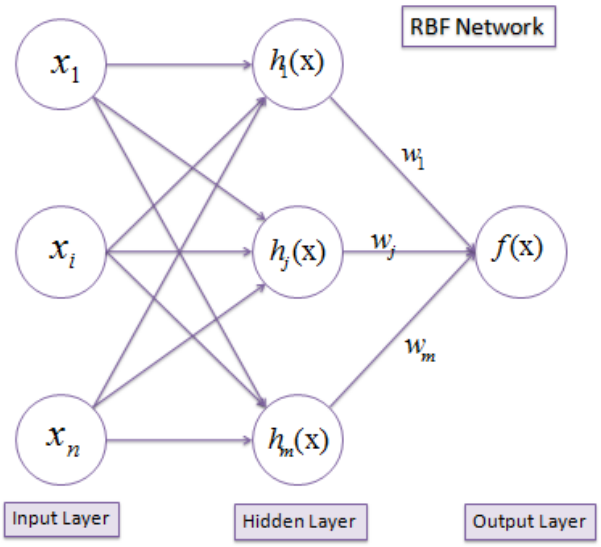
In this, we apply a **non-linear transfer function** to the MNIST feature vector before we go for classification problem.
Hence, we use the Radial Basis Function as the non-linear function.

- *we define a receptor = t*
- *we draw confrontal maps around the receptor.*
- *Gaussian Functions are generally used for Radian Basis Function(confrontal mapping).*
  So we define the radial distance r = ||x- t||.



Gaussian Radial Function :=

$$\phi(r) = exp\ (-\ r^2/2\sigma^2) \qquad where\ \sigma > 0$$



$$f(x) = \sum_{j=1}^{m} w_j h_j(x)$$

$$h(x) = \exp\left(-\frac{(x-c)^2}{r^2}\right)$$

Hyper - parameters used

- batch_size for stochastic gradient descent
- number of centroids (hidden neurons)
- variance (σ)

Training the data

```
We have 60000 observations with 784 dimensions
Train with approximately 83 epochs
```

We calculate the centroids using Kmeans with the number of centers set to the hyperparameter *number of centroids (no of hidden neurons)*.

Building the Hidden Layer

```
with tf.name_scope("Hidden_layer") as scope:
    #Centroids and var are the main trainable parameters of the first layer
    centroids = tf.Variable(tf.Variable(cent, dtype = tf.float32),name='centroids')
    #var is the variance used for each of the hidden neurons
    var =
tf.Variable(tf.truncated_normal([num_centr],mean=var_rbf,stddev=10,dtype=tf.float32),name='RBF_variance')
    exp_list = []
    for i in range(0,num_centr):
        #calculate the exponential value for each of the hidden neuron with respect to the corresponding
centroid

 exp_list.append(tf.exp((-1*tf.reduce_sum(tf.square(tf.subtract(x,centroids[i,:])),1))/(2*var[i])))
        phi = tf.transpose(tf.stack(exp_list))
```

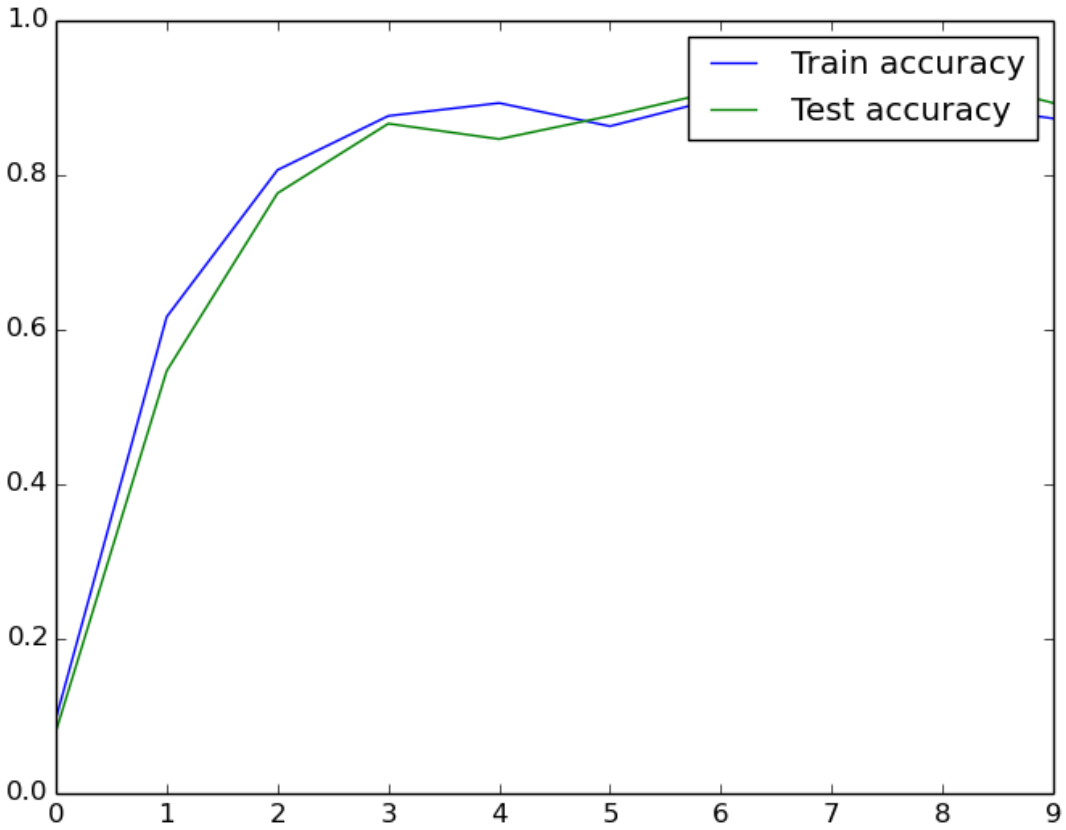
Once we have build the hidden layer we simply compute the output layer as follows

```
with tf.name_scope("Output_layer") as scope:
    w = tf.Variable(tf.truncated_normal([num_centr,num_classes], stddev=0.1,
dtype=tf.float32),name='weight')
    bias = tf.Variable( tf.constant(0.1, shape=[num_classes]),name='bias')
    h = tf.matmul(phi,w)+bias
```

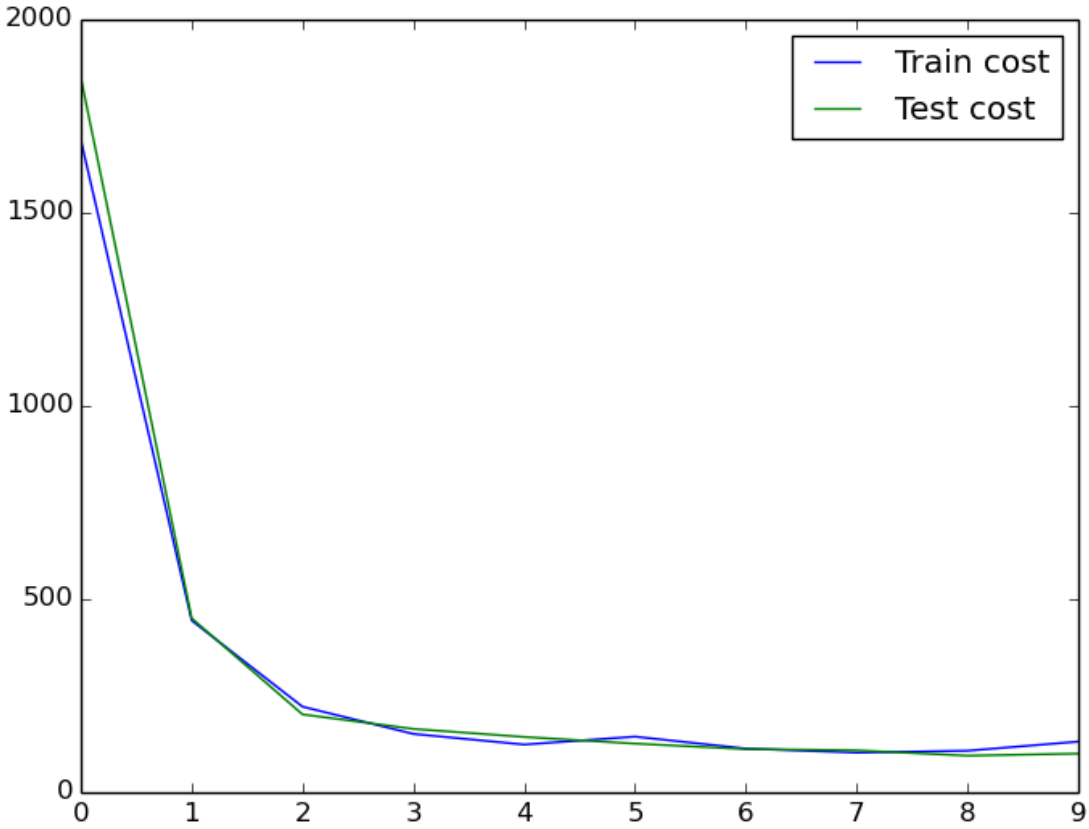
Once we have calculated the output function and added it with a bias, we apply the SoftMax function upon it. The loss was then calculated using cross-entropy function.

We then calculated the accuracy and the loss values for different epoch values.

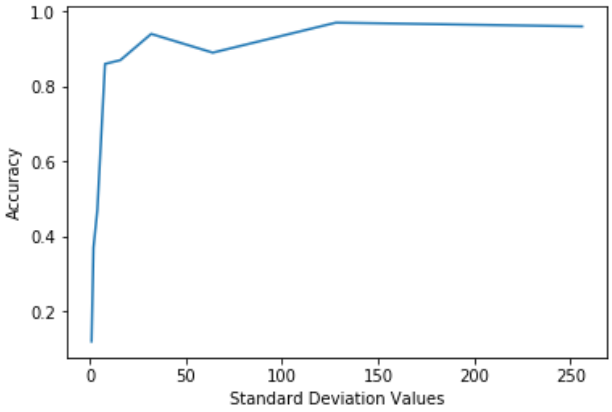
Training and Test Accuracy

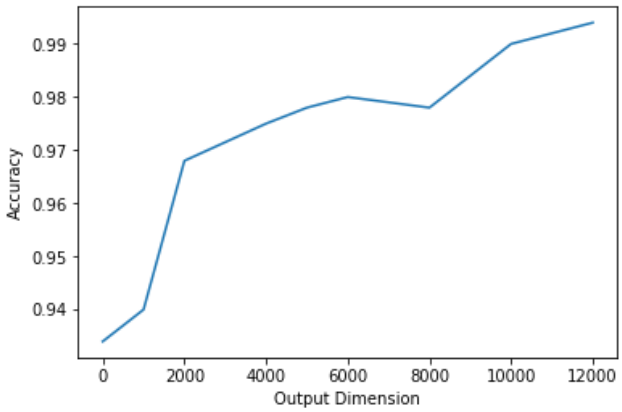


Training and Test Loss



We also compared our results obtained over different hyper-parameters like standard deviation and output dimension.





Finally I used 5 images created by me on pen and paper representing the digits in order to correctly predict the outcome using our classifier. Out of 5, 3 were correctly classified and two wrongly.