Q3

September 26, 2018

0.1 Question 3: Implementation of Deep neural netwrok with arbitary hidden layers and units.

(a) Test your network with the same training data that you used in Problem 2, using both 1D and higher dimensional data. Experiment with using 3 and 5 hidden layers. Evaluate the accuracy of your solutions in the same way as Problem 2.

Sol: Here we implement designing a deep neural network which can handle arbitary inputs,dimensions, hidden layer and hidden units.
Loss Function Regression loss function as given in the question.

Input Inputs are generated randomly by setting the number of training inputs and the dimensions.

Outputs The oputputs are limited to a single output. Here I have used **y=sin(x)** function to produce the ouputs. If the there is a n dimensional input, output is a function of sin() for the first dimension and an summation of all the other subsequent dimensions.

Hyper Paramters For this function there are four hyper paramters, that is the **Learning Rate** (Step Size), the number of **epochs** , the number of **hidden layers** and the number of **hidden units** in each hidden layer.

The Netwrok has 6 main Functions

1) Network Initialization
2) nonlin
3) feed_forward
4) back_prop
5) update_network
6) train

Now lets got thorugh each function

```
In [1]: ## Initialize the Network
        def initialize_network(n_inputs,dimension, hidden_in_each_layer):
            if (dimension==1):
                x= np.random.randn(n_inputs,1) ## Randomly Genrated Inputs
                y= np.sin(x)                   ## Generating Outputs
```

```
    else:
## Breaking the inputs such that, the first input goes to the sin curve and the rest, ad
        x= np.random.randn(n_inputs,dimension) ## Randomly Genrated Inputs
        x1=np.reshape(x[:,0],(n_inputs,1))
        x2 = np.sum(x[:,1:],axis=1,keepdims= True)
        y= np.sin(x1) + x2
## Initializing Weights and Biases by using dictionary
    n_hidden_layers = len(hidden_in_each_layer)
    wts={'w1':np.random.randn(dimension,hidden_in_each_layer[0]) }
    b = {'b1':np.random.randn(hidden_in_each_layer[0],1)}
    for i in range(n_hidden_layers-1):
        wts['w'+str(i+2)]= np.random.randn(hidden_in_each_layer[i],hidden_in_each_layer[
        b['b'+str(i+2)] = np.random.randn(hidden_in_each_layer[i+1],1)
    wts['w'+str(n_hidden_layers+1)] = np.random.randn(hidden_in_each_layer[-1],1)
    b['b'+str(n_hidden_layers+1)] = np.random.randn(1)
    return x,y,wts,b

## Non Linear Function RELU
def nonlin(x, deriv = False):
    if (deriv == True):
        return 1*(x>0)
    return np.maximum(0,x)

## Feed Forward Function
def feed_forward(x,wts,b):
    a = {'a1':(x.dot(wts['w1']).transpose() + b['b1']).transpose()}
    for i in range(len(wts)-2):
        a['a'+str(i+2)] = nonlin((a['a'+str(i+1)].dot(wts['w'+str(i+2)]).transpose() + b
    a['al'] = a['a'+str(len(wts)-1)].dot(wts['w'+str(len(wts))])+ b['b'+str(len(wts))]
    return a

## Back Propogation by appling the four main formulas of back propogation
def back_prop(x,y,a,wts):
    error = (((y-a['al'])**2).sum())                    ## Calculating the error
    d = {'d'+str(len(wts)): a['al'] - y}                ## Delta for final layer
    wt_err = {'wr'+str(len(wts)):a['a'+str(len(wts)-1)].T.dot(d['d'+str(len(wts))])} ##
    b_err = {'br'+str(len(wts)):d['d'+str(len(wts))].sum()}                          ##
    for i in reversed(range(1,len(wts))):
        d ['d'+str(i)] = (wts['w'+ str(i+1)].dot(d['d'+str(i+1)].transpose())).transpose
        if (i==1):
            wt_err['wr'+str(i)] = x.T.dot(d['d'+str(i)])
            b_err['br'+str(i)] = np.sum(d['d'+str(i)].transpose(),axis=1,keepdims= True)
        else:
            wt_err['wr'+str(i)] = a['a'+str(i-1)].T.dot(d['d'+str(i)])
            b_err['br'+str(i)] = np.sum(d['d'+str(i)].transpose(),axis=1,keepdims= True)
#     print(error)
    return wt_err, b_err, d, error
```

```
## Updating the Network
def update_network(wts,b,wt_err,b_err,lr):
    for i in range(0,len(wts)):
        wts['w'+str(i+1)] -= lr * wt_err['wr'+str(i+1)]
        b['b'+str(i+1)] -= lr * b_err['br'+str(i+1)]
    return wts,b

## Training the Network
def train(net,epochs,lr):
    error = []
    (wts,b) = (net[2],net[3])
    for i in range(epochs):
        a = feed_forward(net[0],wts,b)
        err = back_prop(net[0], net[1],a,wts)
        error.append(err[3])
        update = update_network(wts,b,err[0],err[1],lr)
        (wts,b)= (update[0],update[1])
    return error, update[0], update[1]
```

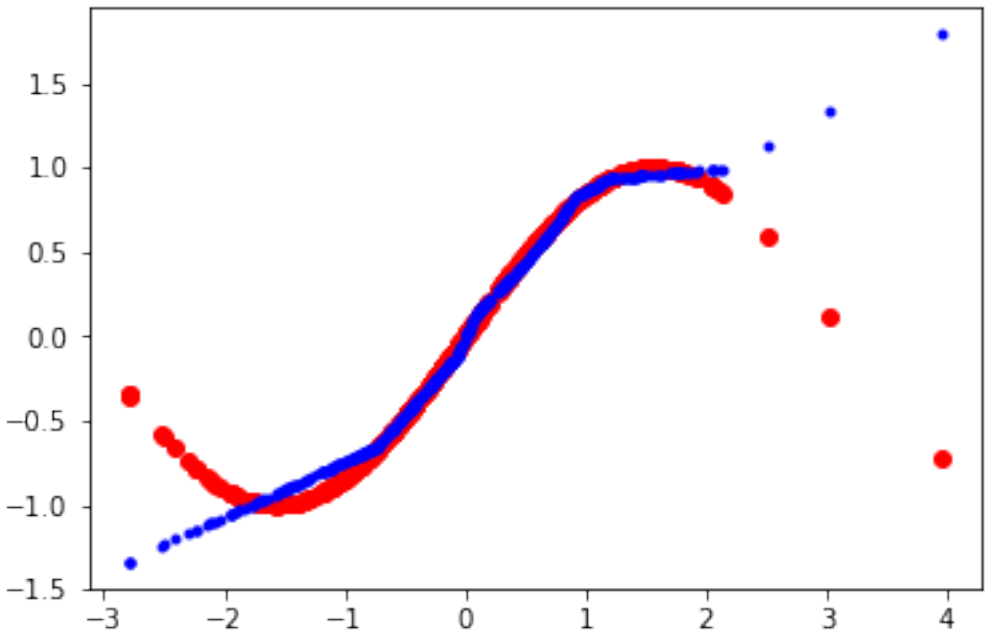
0.2 (a)Defining the Main Function and Implementing for 1D input

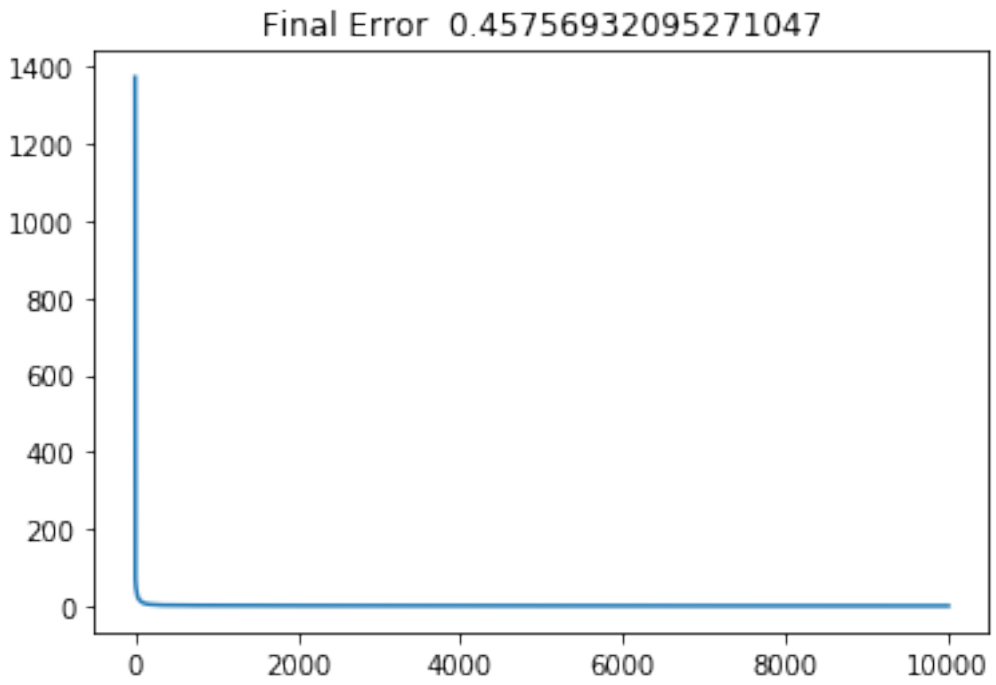
In [2]:
```
# Importing the Libraries
import numpy as np
from matplotlib import pyplot as plt
from math import exp

## Defining the inputs
## hidden_in_each_layer is such that: the number of entries
# define the number of hidden layers and each entered value is the number of hidden unit
(n_inputs,dimension,hidden_in_each_layer) =(100,1,[3,6,6,6,2]) ## Use [3,6,5] for good a
np.random.seed(1)                                              ## Use [3,6,6,6,2] for good a

# Initializing the Network
net = initialize_network(n_inputs,dimension, hidden_in_each_layer)

## Hyper Parameter Initialization
epochs= 10000
lr = 0.0001
# net[0]
# print(net[1])

## Train
tr = train (net,epochs,lr)
```

0.2.1 Plotting the Graphs

Red: Ground Truth Values
Blue: Predicted Values

In [3]:
```
n_test = 500
dimension=1
## Plotting the Outputs vs Inputs
if (dimension==1):
        test_input= np.random.randn(n_test,1) ## Randomly Genrated Inputs
        gtruth = np.sin(test_input)
else:
## Breaking the inputs such that, the first input goes to the sin curve and the rest add
        x= np.random.randn(n_inputs,dimension) ## Randomly Genrated Inputs
        x1=np.reshape(x[:,0],(n_inputs,1))
        x2 = np.sum(x[:,1:],axis=1,keepdims= True)
        y= np.sin(x1) + x2
plt.plot(test_input,gtruth,'r.',markersize=12)

# Caluclating Predictions from the network
predict= feed_forward(test_input,tr[1],tr[2])['al']

plt.plot(test_input,predict ,'b.')
plt.show()

# Plotting the Error
plt.title('Final Error  ' + str(tr[0][-1]))
plt.plot(tr[0])
plt.show()
```





0.2.2 Important Inferences

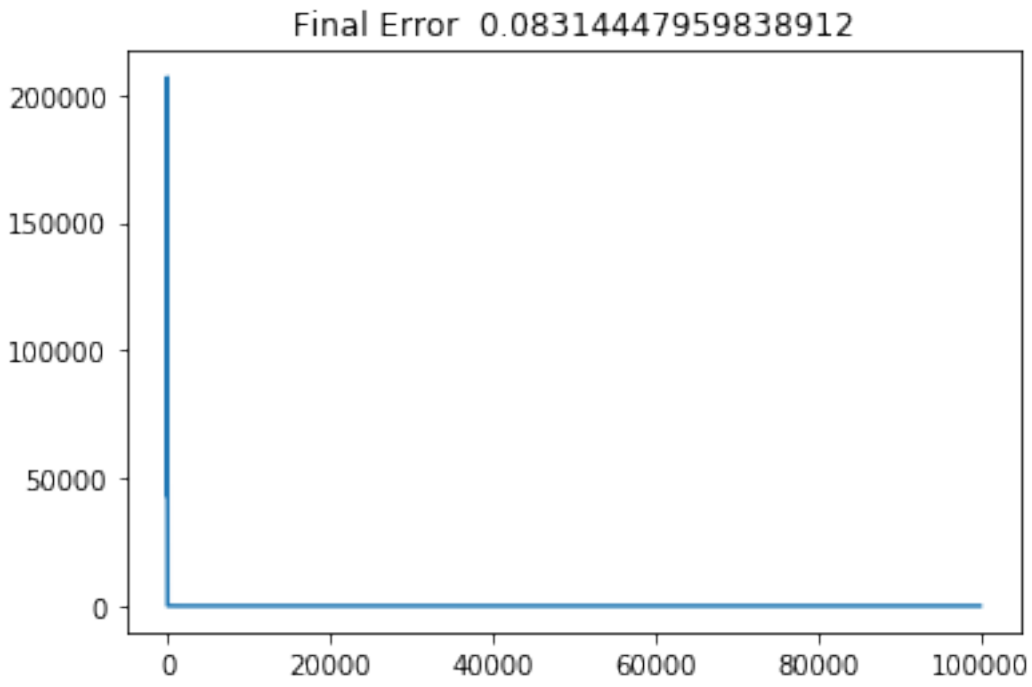
1. Comaparing to the previous questions, the number of epochs increased as the number of hidden layers increased.
2. Varying the hidden units has a significant effect on the output. From experiments, the hidden units should be small in the intial layer, increase in the middle layers and again decrease in the final layers.
3. As the number of parameters are more, decreasing the learning rate and increasing the epochs helped.

Comparing 3 and 5 layer hidden networks

1. I was able to get a good prediction of the sin() function by tuning the params to lr= 0.001; epochs = 1000 and with 3 hidden layers with 3, 6 and 5 hidden units respectively.
2. We can see that the predictions in the initial points and final points is not accurate as, the training data here is also very less, therefore poor predictions.
3. For 5 layers: epochs = 10,000; lr= 0.0001; hidden units and layers: [3,6,6,6,2].
4. When using 5 hidden layers, increasing the epochs to 10,000 and decreasing the learning rate to 0.0001 helped. This is due to the fact that the number of weights significantly increase.
5. keeping the number of hidden units smaller helps getting a better accuracy fot both 3 and 5 layer hidden network.

0.3 Defining the Main Function and Implementing for nD input

In [4]:
```
# Importing the Libraries
import numpy as np
from matplotlib import pyplot as plt
from math import exp

## Defining the inputs
## hidden_in_each_layer is such that: the number of entries
# define the number of hidden layers and each entered value is the number of hidden unit
(n_inputs,dimension,hidden_in_each_layer) =(100,2,[25,15])
np.random.seed(1)

# Initializing the Network
net = initialize_network(n_inputs,dimension, hidden_in_each_layer)

## Hyper Parameter Initialization
epochs= 100000
lr = 0.0001
# net[0]
# print(net[1])

## Train
tr = train (net,epochs,lr)

# Plotting the Error
plt.title('Final Error  ' + str(tr[0][-1]))
plt.plot(tr[0])
plt.show()
```

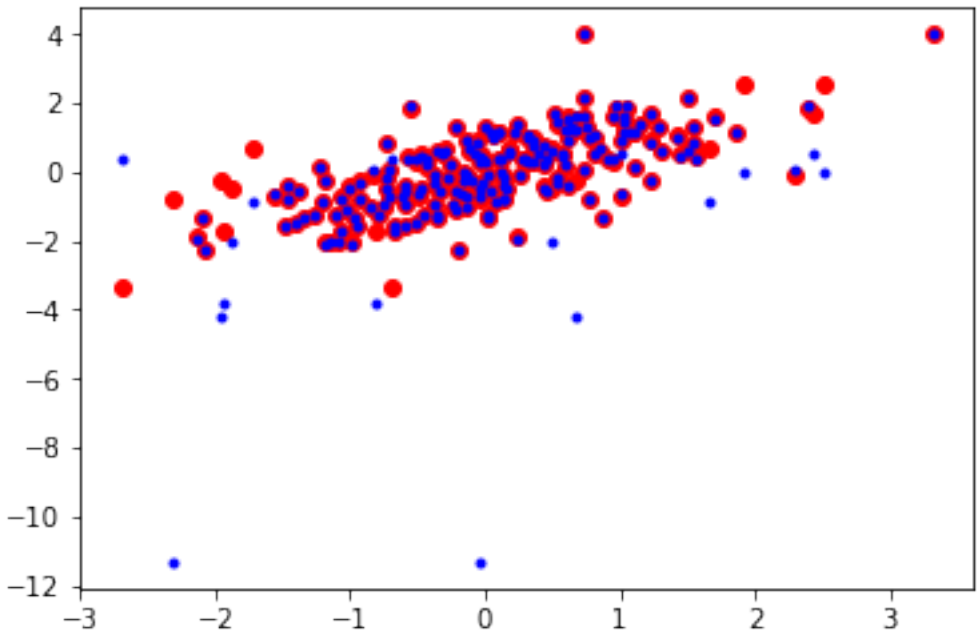


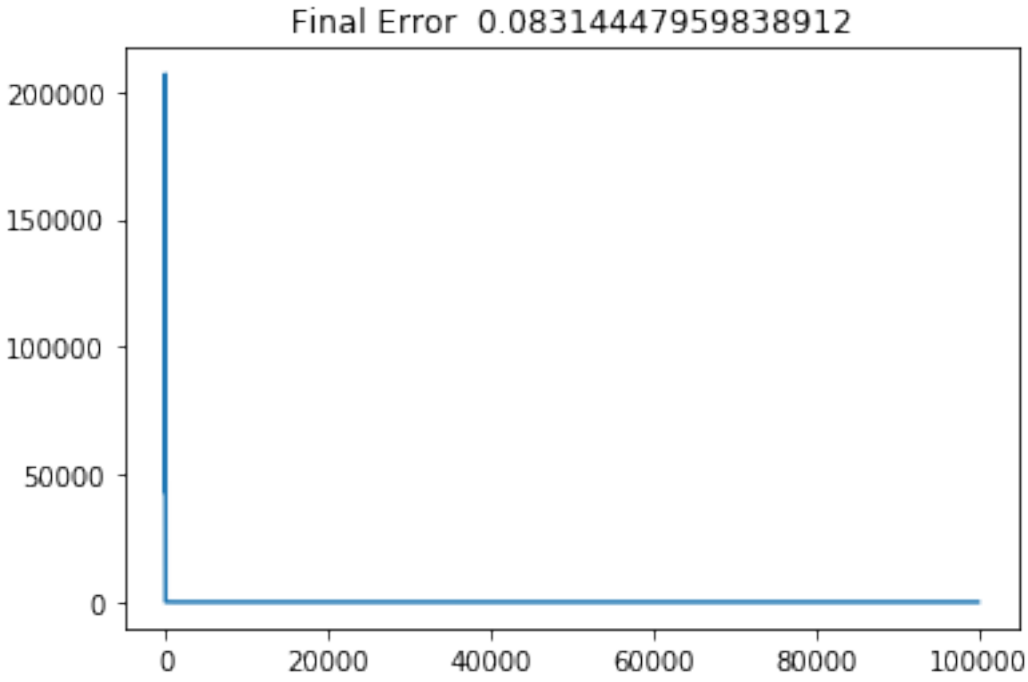
0.3.1 Plotting the Graphs

Red: Ground Truth Values
Blue: Predicted Values

```
In [5]: n_test = 500
        dimension=2
        ## Plotting the Outputs vs Inputs
        if (dimension==1):
                test_input= np.random.randn(n_test,1) ## Randomly Genrated Inputs
                gtruth = np.sin(test_input)
        else:
        ## Breaking the inputs such that, the first input goes to the sin curve and the rest ad
                x= np.random.randn(n_inputs,dimension) ## Randomly Genrated Inputs
                x1=np.reshape(x[:,0],(n_inputs,1))
                x2 = np.sum(x[:,1:],axis=1,keepdims= True)
                gtruth= np.sin(x1) + x2
        # plt.plot(test_input1,gtruth,'r.')
        plt.plot(x,gtruth,'r.',markersize=12)

        # Caluclating Predictions from the network
        predict= feed_forward(x,tr[1],tr[2])['al']

        plt.plot(x,predict ,'b.')
        plt.show()
```

```
# Plotting the Error
plt.title('Final Error  ' + str(tr[0][-1]))
plt.plot(tr[0])
plt.show()
```





0.3.2 Important Inferences

1. For nDimensions find the right hyperparamters becomes equally difficlut.
2. Increasing the hidden units or hidden layers did not help get better accuracy.
3. Although, by keeping 25 and 15 hidden units in 2 hidden layers, I was able to get a moderatley accurate result, which is better than using a single hiiden layer as in the previous question.
4. Changing the dimension, will require to tune the hyperparamters again, but I could not find any significant connection between dimension and paramter tuning.

0.4 (B) Comments on HyperParamter Tuning

- From my experience, I would say tuning the shallow network is much easier than deep netwrok.
  - This is because of very unexpected behavior of deep netwroks with change in any hyper parameter.
  - For example, changing the hidden unit in a layer by a single value can give an accurate output or an entirely faulty output.
- For the linear functions we used in the above questions, I think using a shallow network is better, as it is able to predict an accurate function. Deep metwroks can be used for more complex functions.

0.5 (c) Experiments to see if deep netwroks take more time to converge?

0.5.1 Using Shallow Network

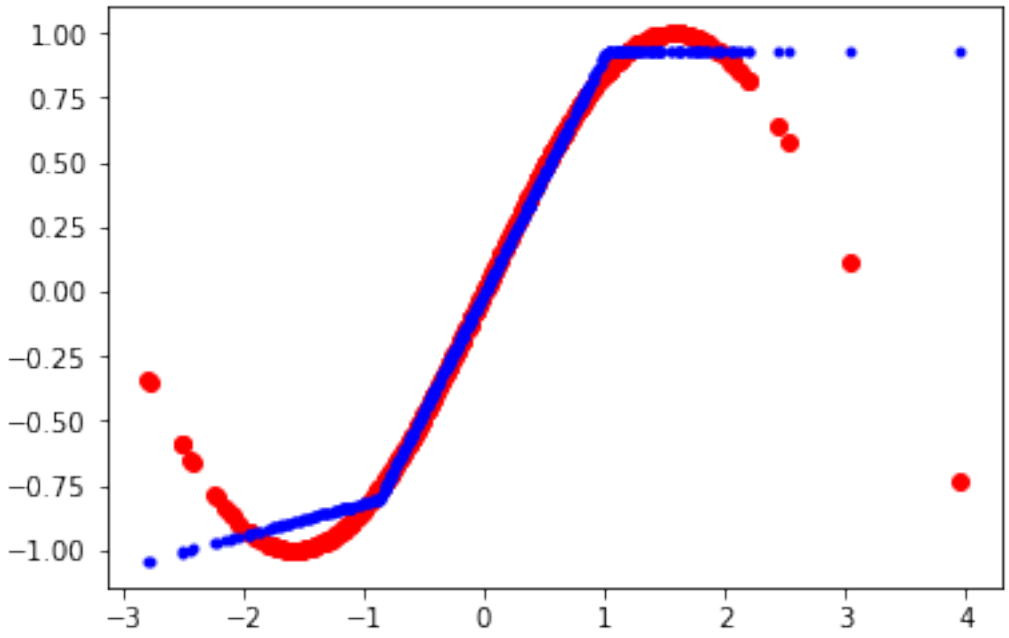
In [19]:
```
# Importing the Libraries
import numpy as np
from matplotlib import pyplot as plt
from math import exp

## Defining the inputs
## hidden_in_each_layer is such that: the number of entries
# define the number of hidden layers and each entered value is the number of hidden uni
(n_inputs,dimension,hidden_in_each_layer) =(100,1,[1,2,2]) ## Use [3,6,5] for good accu
np.random.seed(1)                                          ## Use [3,6,6,6,2] for good

# Initializing the Network
net = initialize_network(n_inputs,dimension, hidden_in_each_layer)

## Hyper Parameter Initialization
epochs= 5000
lr = 0.0001
# net[0]
# print(net[1])
```

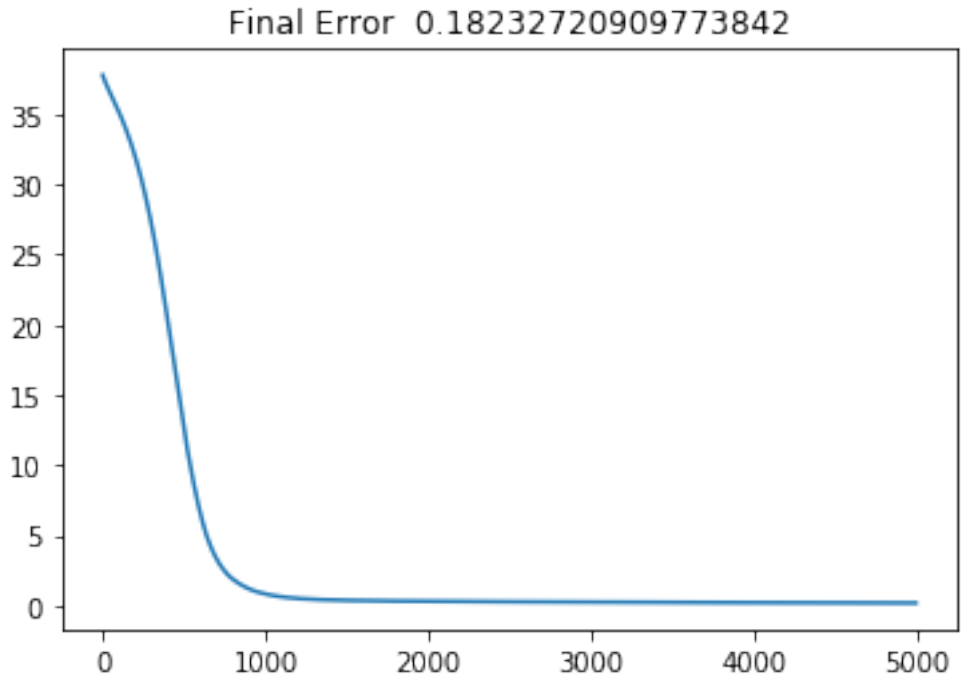
```
        ## Train
        tr = train (net,epochs,lr)
```

0.5.2 Plotting the Graphs

Red: Ground Truth Values
Blue: Predicted Values

```
In [20]: n_test = 500
         dimension=1
         ## Plotting the Outputs vs Inputs
         if (dimension==1):
                 test_input= np.random.randn(n_test,1) ## Randomly Genrated Inputs
                 gtruth = np.sin(test_input)
         else:
         ## Breaking the inputs such that, the first input goes to the sin curve and the rest ad
                 x= np.random.randn(n_inputs,dimension) ## Randomly Genrated Inputs
                 x1=np.reshape(x[:,0],(n_inputs,1))
                 x2 = np.sum(x[:,1:],axis=1,keepdims= True)
                 y= np.sin(x1) + x2
         plt.plot(test_input,gtruth,'r.',markersize=12)

         # Caluclating Predictions from the network
         predict= feed_forward(test_input,tr[1],tr[2])['al']

         plt.plot(test_input,predict ,'b.')
         plt.show()

         # Plotting the Error
         plt.title('Final Error  ' + str(tr[0][-1]))
         plt.plot(tr[0])
         plt.show()
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−3
−2
−1
0
1
2
3
4


Final Error 0.18232720909773842
35
30
25
20
15
10
5
0
0
1000
2000
3000
4000
5000

0.6 Using Deep Network

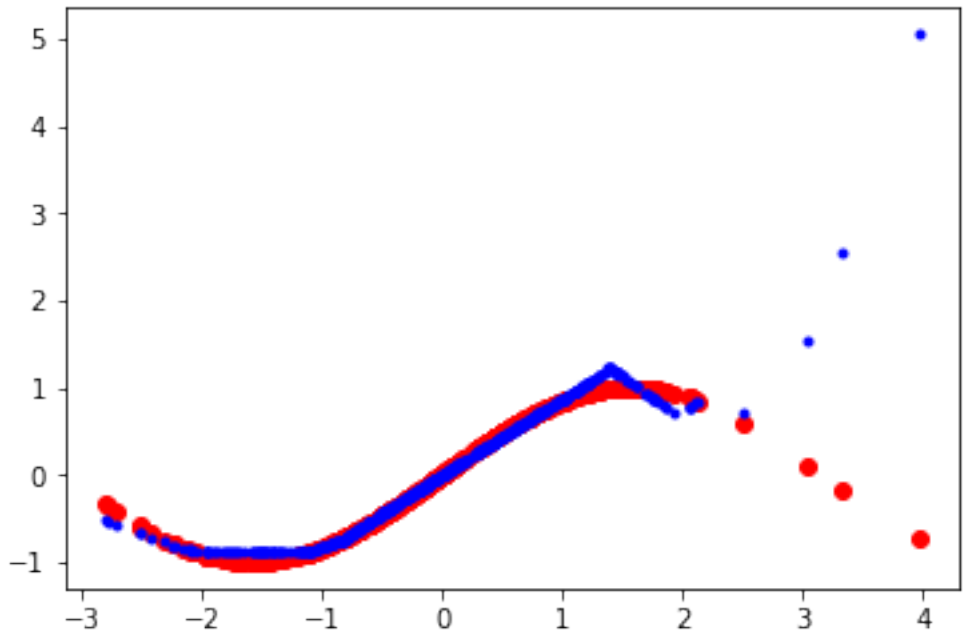
```
In [40]: # Importing the Libraries
         import numpy as np
         from matplotlib import pyplot as plt
         from math import exp

         ## Defining the inputs
         ## hidden_in_each_layer is such that: the number of entries
         # define the number of hidden layers and each entered value is the number of hidden uni
         (n_inputs,dimension,hidden_in_each_layer) =(100,1,[3,6,5,6,6,2]) ## Use [3,6,5] for goo
         np.random.seed(1)                                                ## Use [3,6,6,6,2] for good

         # Initializing the Network
         net = initialize_network(n_inputs,dimension, hidden_in_each_layer)

         ## Hyper Parameter Initialization
         epochs= 3500
         lr = 0.0001
         # net[0]
         # print(net[1])

         ## Train
         tr = train (net,epochs,lr)
```

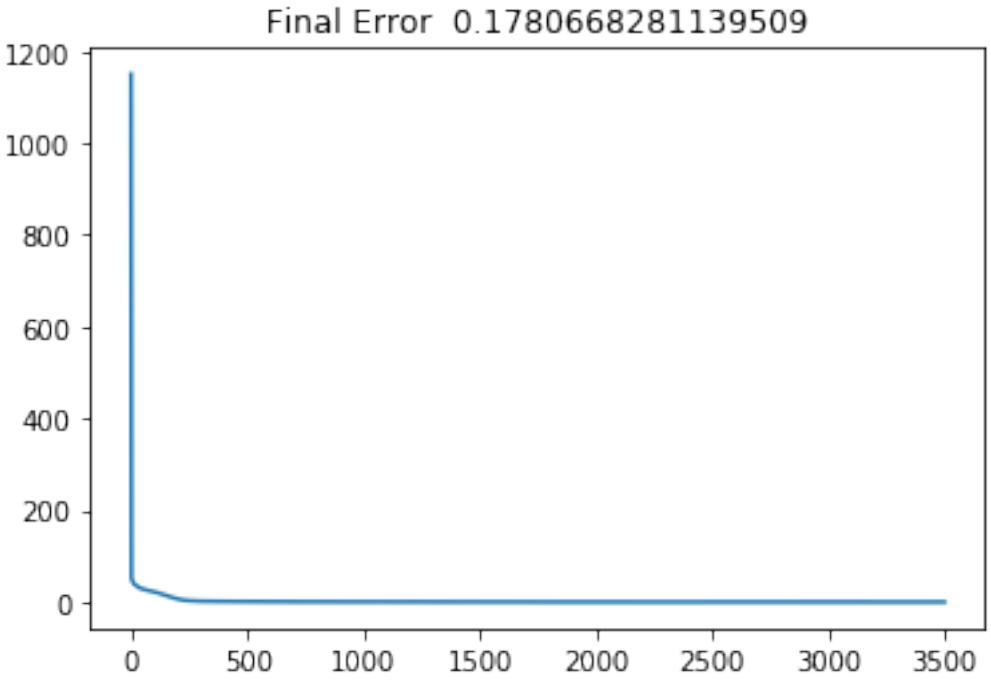
0.6.1 Plotting the Graphs

Red: Ground Truth Values
Blue: Predicted Values

```
In [41]: n_test = 500
         dimension=1
         ## Plotting the Outputs vs Inputs
         if (dimension==1):
                 test_input= np.random.randn(n_test,1) ## Randomly Genrated Inputs
                 gtruth = np.sin(test_input)
         else:
         ## Breaking the inputs such that, the first input goes to the sin curve and the rest ad
                 x= np.random.randn(n_inputs,dimension) ## Randomly Genrated Inputs
                 x1=np.reshape(x[:,0],(n_inputs,1))
                 x2 = np.sum(x[:,1:],axis=1,keepdims= True)
                 y= np.sin(x1) + x2
         plt.plot(test_input,gtruth,'r.',markersize=12)

         # Caluclating Predictions from the network
         predict= feed_forward(test_input,tr[1],tr[2])['al']

         plt.plot(test_input,predict ,'b.')
         plt.show()
```

```
# Plotting the Error
plt.title('Final Error  ' + str(tr[0][-1]))
plt.plot(tr[0])
plt.show()
```





0.7 Inferences for comapring shallow network to deep network.

- Keeping all the hyper paramters constant other then that epochs and and hidden layer (hidden units), the deep network converged to an error of 0.17 in 3500 epochs, whereas the shallow network converge to an error of 0.18 in 5000 epochs.
- Also, the accuracy for deep netwrok is much better than the shallow network.