Arjun Srinivasan Ambalam ENPM 808 Y HW-1

100 % Test and Train Data

MLP network with 2 hidden layers

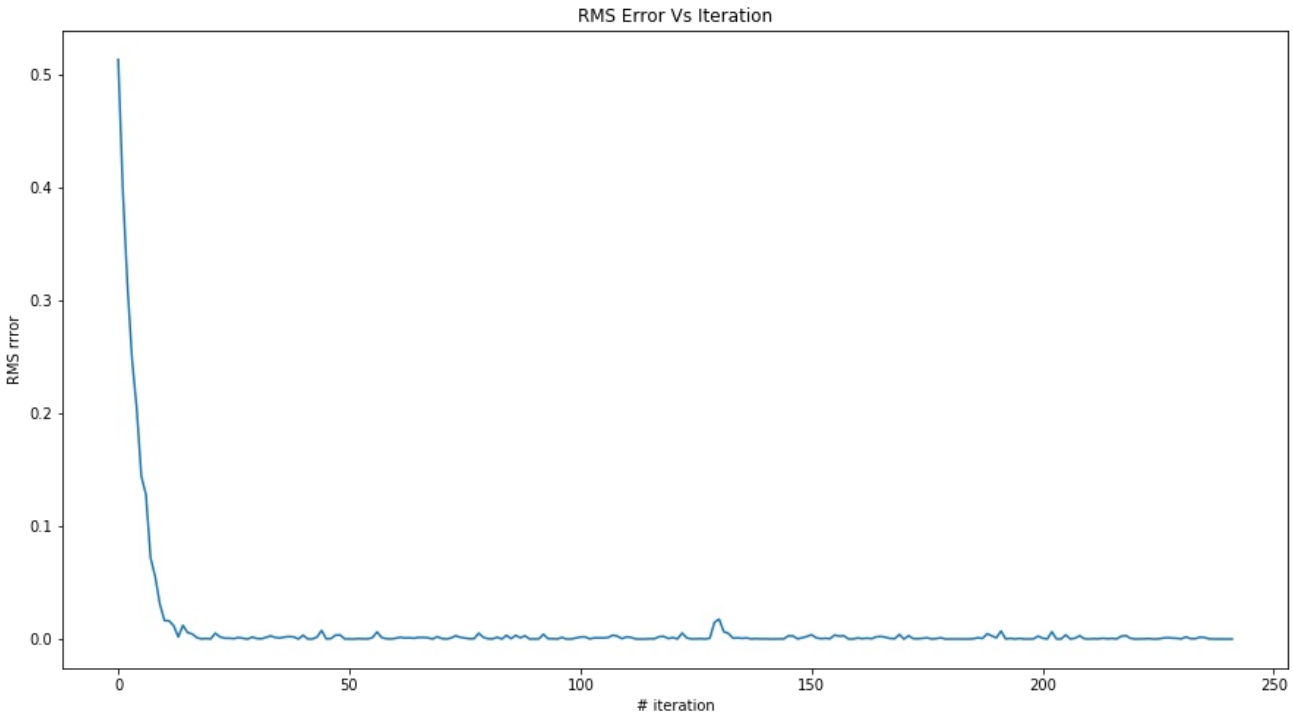
2 Neurons in each layer

Output has 1 neuron

In [18]:

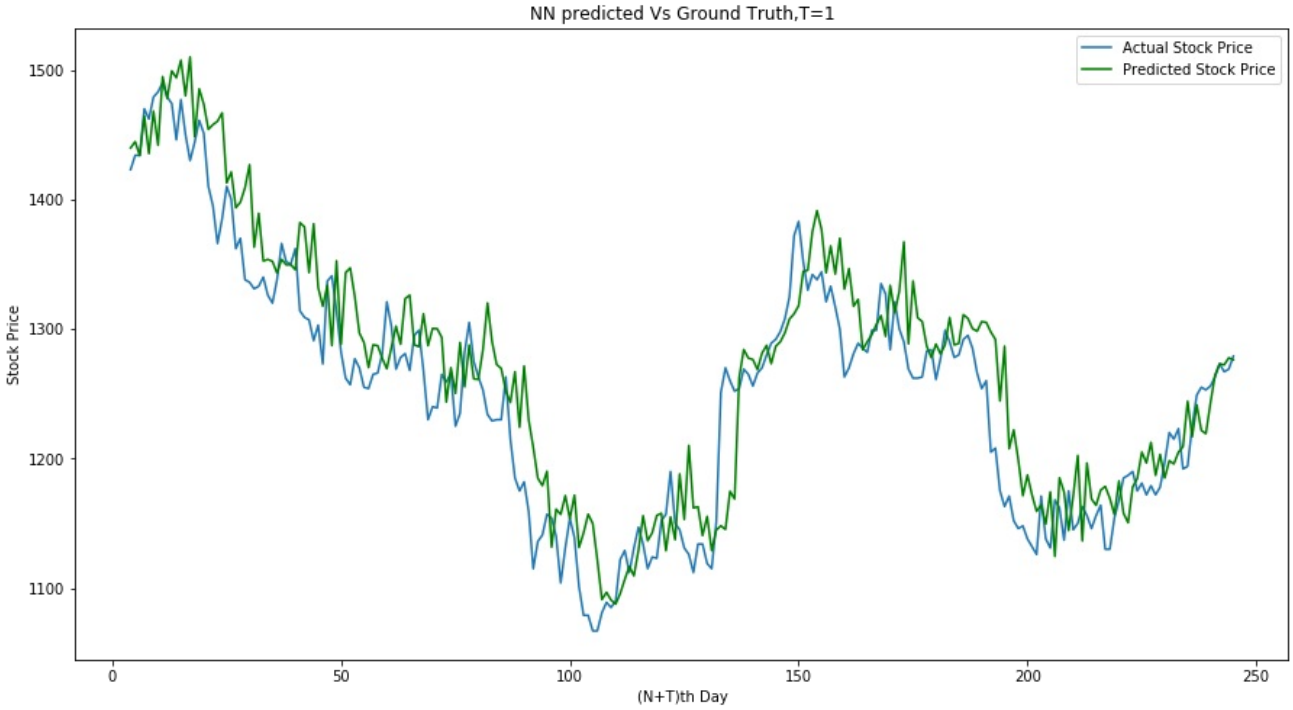
```
# 100 test and train
T = 1 # To test 1,10,30,80
N = 4 # take input from user here.
x_train, y_train,ix = Preprocess(price_arr, N, T)
x_train=np.array(x_train)/1000
y_train=np.array(y_train)/1000
batch_size=1#Used for training Mini batch Gradient Descent if batch_size=1 Stochastic gradient descent
alpha=0.001#Learning Rate
w1, w2, w3, b1, b2, b3=model_fit(x_train, y_train, N, alpha,batch_size, epochs=1)
a1, a2, a3, z1, z2, z3=front_propogation(x_train.T,w1, w2, w3, b1, b2, b3)
y_pred=np.squeeze(a3*1000)
```

```
Mean Error after 242 iterations is 9.97088112803498e-06
```



In [19]:

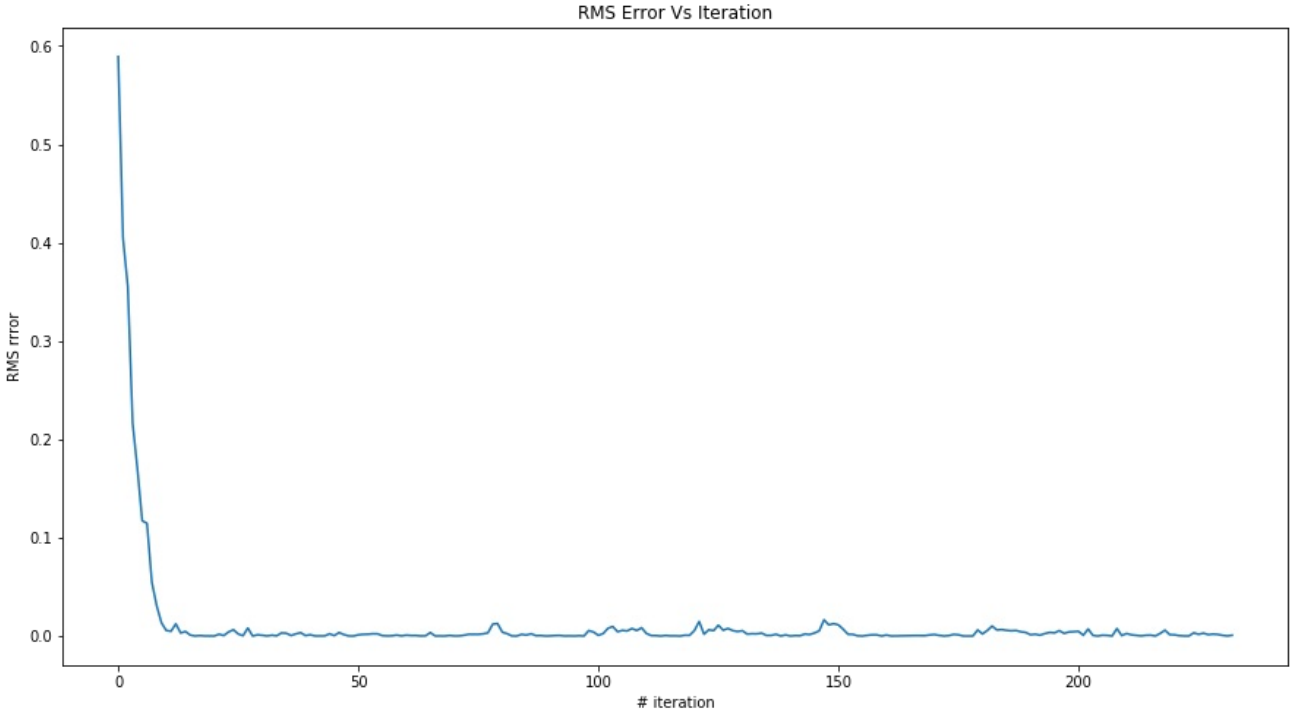
```
plt.figure(figsize=(15,8))
plt.plot(ix,price_arr[N+T-1::], label='Actual Stock Price')
plt.plot(ix,y_pred,color='green', label='Predicted Stock Price')
plt.legend()
plt.title("NN predicted Vs Ground Truth,T=1")
plt.xlabel('(N+T)th Day')
plt.ylabel('Stock Price')
plt.show()
```



In [4]:

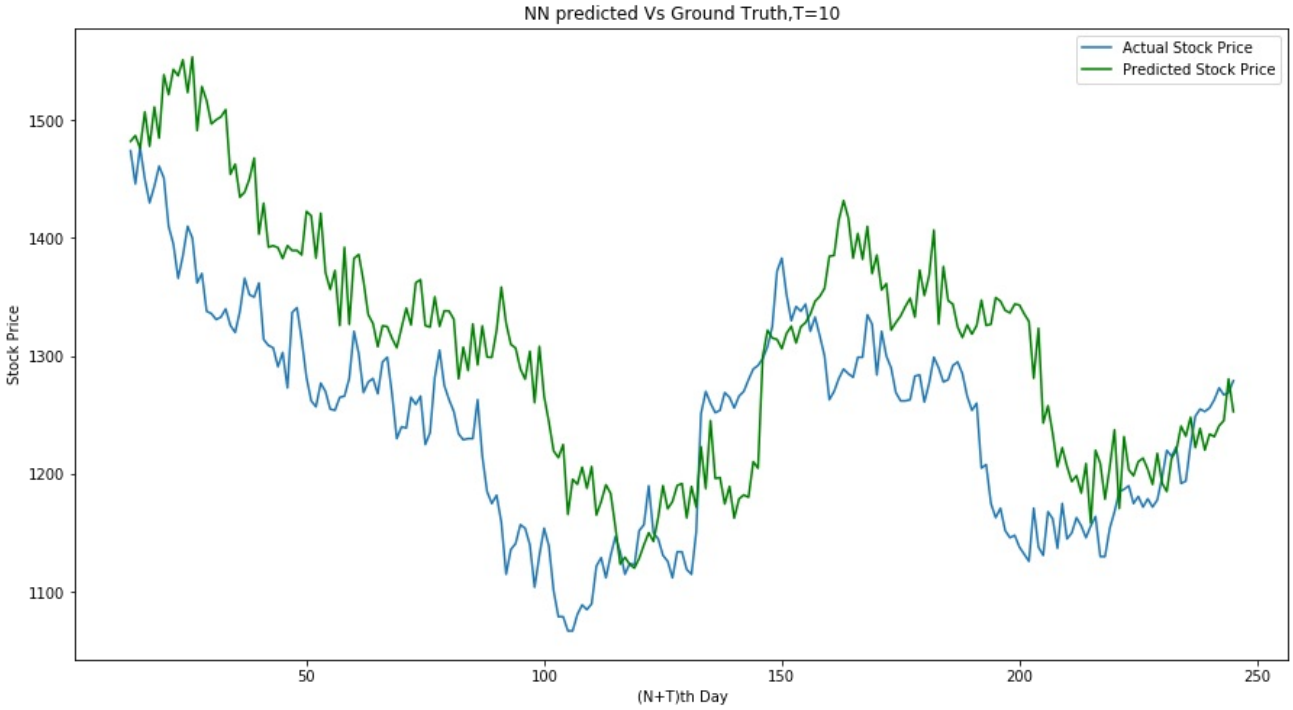
```
T = 10 # To test 1,10,30,80
N = 4 # take input from user here.
x_train, y_train,ix = Preprocess(price_arr, N, T)
x_train=np.array(x_train)/1000
y_train=np.array(y_train)/1000
batch_size=1#Used for training Mini batch Gradient Descent if batch_size=1 Stochastic gradient descent
alpha=0.001#Learning Rate
w1, w2, w3, b1, b2, b3=model_fit(x_train, y_train, N, alpha,batch_size, epochs=1)
a1, a2, a3, z1, z2, z3=front_propogation(x_train.T,w1, w2, w3, b1, b2, b3)
y_pred=np.squeeze(a3*1000)
```

```
Mean Error after 233 iterations is 0.0008402619294150 13
```



In [5]:

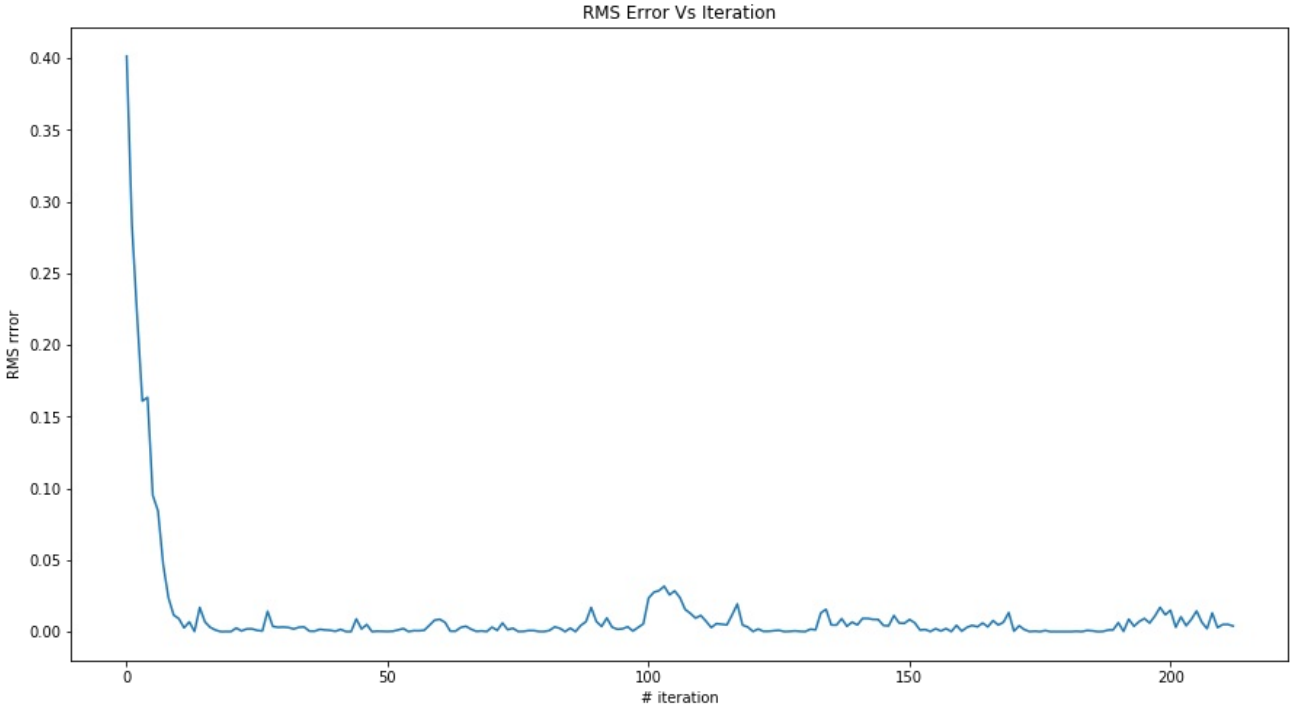
```
plt.figure(figsize=(15,8))
plt.plot(ix,price_arr[N+T-1::], label='Actual Stock Price')
plt.plot(ix,y_pred,color='green', label='Predicted Stock Price')
plt.legend()
plt.title("NN predicted Vs Ground Truth,T=10")
plt.xlabel('(N+T)th Day')
plt.ylabel('Stock Price')
plt.show()
```



In [6]:

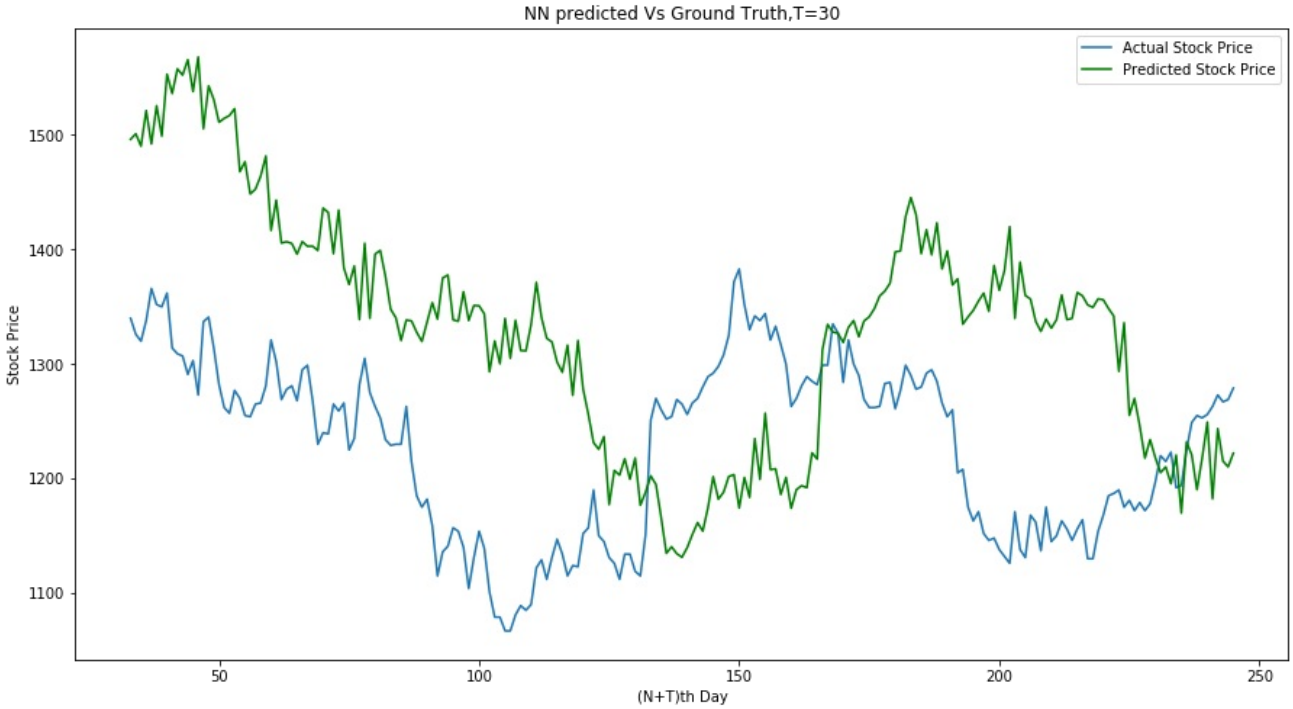
```
T = 30 # To test 1,10,30,80
N = 4 # take input from user here.
x_train, y_train,ix = Preprocess(price_arr, N, T)
x_train=np.array(x_train)/1000
y_train=np.array(y_train)/1000
batch_size=1#Used for training Mini batch Gradient Descent if batch_size=1 Stochastic gradient descent
alpha=0.001#Learning Rate
w1, w2, w3, b1, b2, b3=model_fit(x_train, y_train, N, alpha,batch_size, epochs=1)
a1, a2, a3, z1, z2, z3=front_propogation(x_train.T,w1, w2, w3, b1, b2, b3)
y_pred=np.squeeze(a3*1000)
```

```
Mean Error after 213 iterations is 0.003912175786693948
```



In [7]:

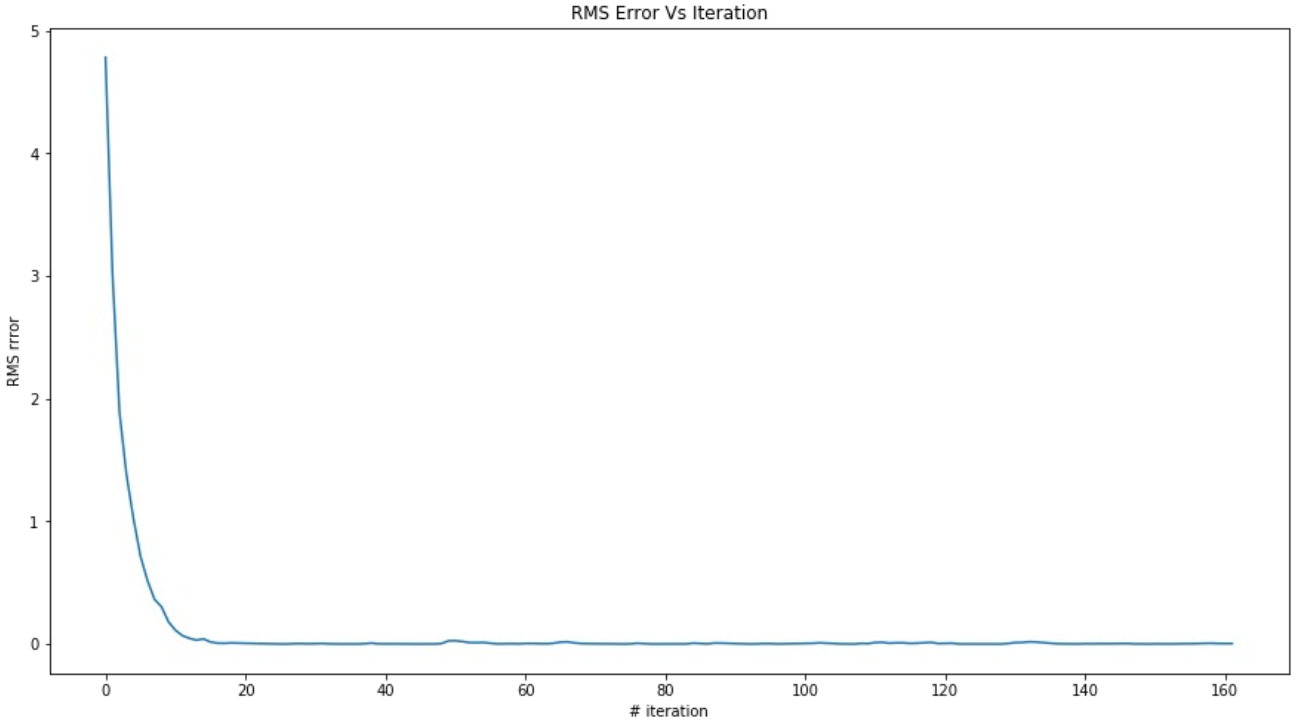
```
plt.figure(figsize=(15,8))
plt.plot(ix,price_arr[N+T-1::], label='Actual Stock Price')
plt.plot(ix,y_pred,color='green', label='Predicted Stock Price')
plt.legend()
plt.title("NN predicted Vs Ground Truth,T=30")
plt.xlabel('(N+T)th Day')
plt.ylabel('Stock Price')
plt.show()
```



In [67]:

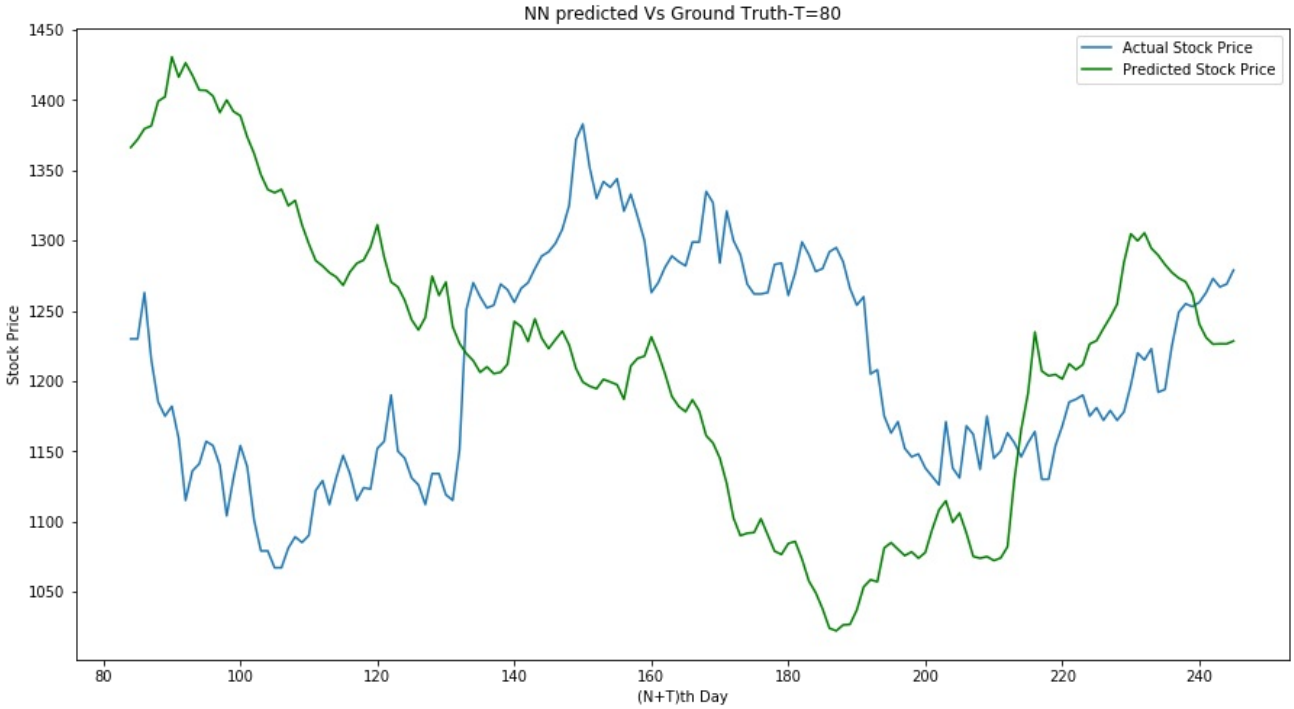
```
T = 80 # To test 1,10,30,80
N = 5
x_train, y_train,ix = Preprocess(price_arr, N, T)
x_train=np.array(x_train)/1000
y_train=np.array(y_train)/1000
batch_size=1#Used for training Mini batch Gradient Descent if batch_size=1 Stochastic gradient descent
alpha=0.001#Learning Rate
w1, w2, w3, b1, b2, b3=model_fit(x_train, y_train, N, alpha,batch_size, epochs=1)
a1, a2, a3, z1, z2, z3=front_propogation(x_train.T,w1, w2, w3, b1, b2, b3)
y_pred=np.squeeze(a3*1000)
```

```
Mean Error after 162 iterations is 0.0033572221353917243
```



In [68]:

```
plt.figure(figsize=(15,8))
plt.plot(ix,price_arr[N+T-1::], label='Actual Stock Price')
plt.plot(ix,y_pred,color='green', label='Predicted Stock Price')
plt.legend()
plt.title("NN predicted Vs Ground Truth-T=80")
plt.xlabel('(N+T)th Day')
plt.ylabel('Stock Price')
plt.show()
```

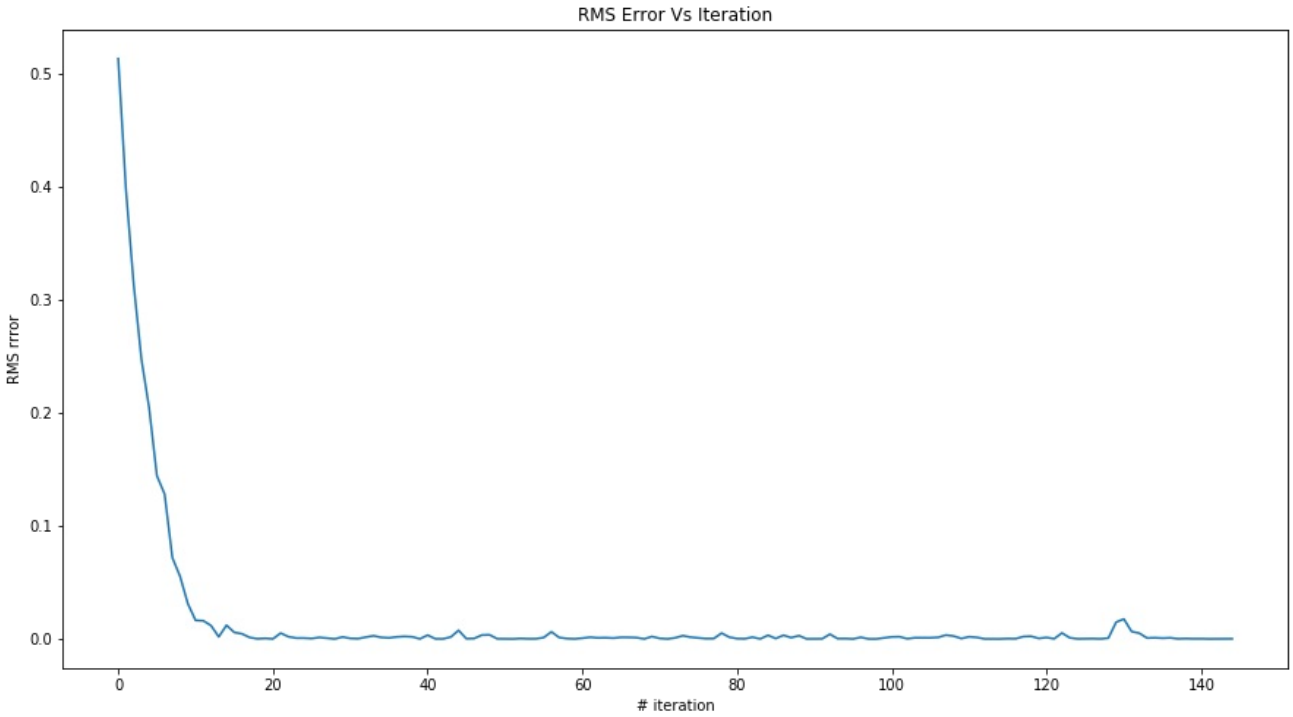


60%-40% Train-Test split

In [46]:

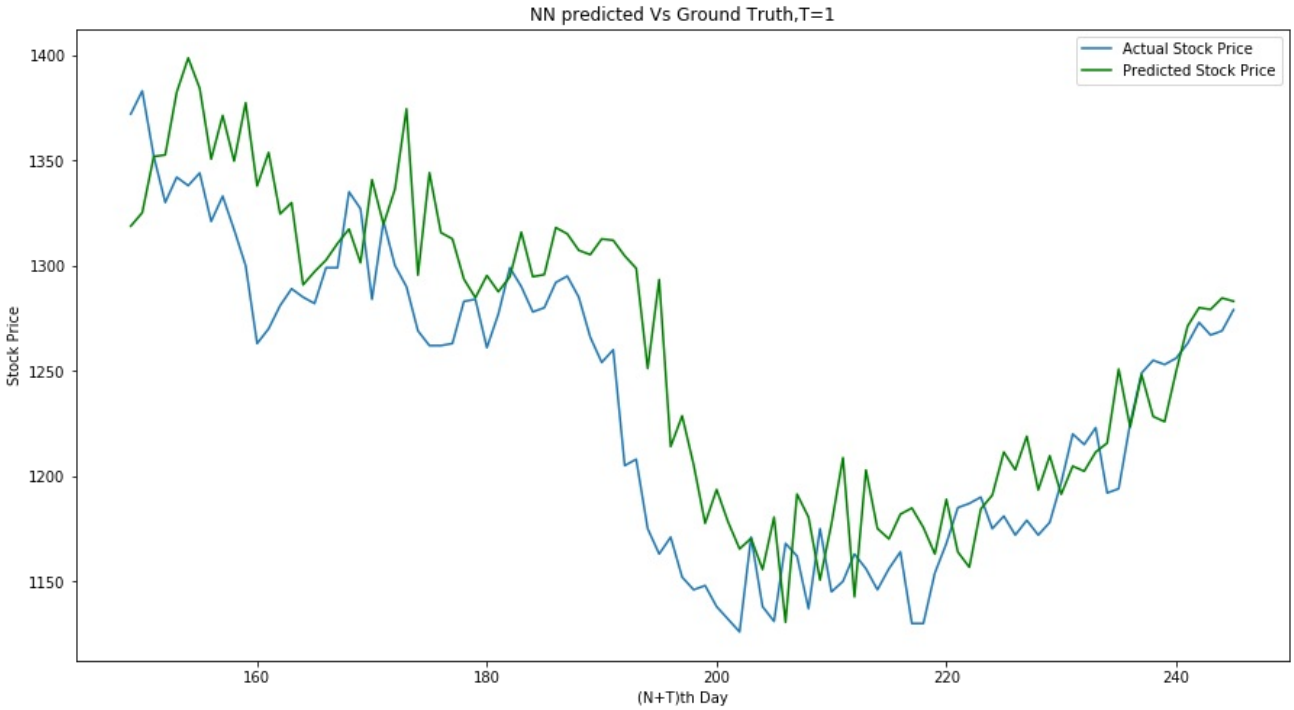
```
T = 1
N = 4
x_train1, y_train1,ix = Preprocess(price_arr, N, T)
x_train=x_train1[0:np.round((len(price_arr) - (N+T) + 1)*0.6).astype(int)]
y_train=y_train1[0:np.round((len(price_arr) - (N+T) + 1)*0.6).astype(int)]
x_test=x_train1[np.round((len(price_arr) - (N+T) + 1)*0.6).astype(int):]
y_test=y_train1[np.round((len(price_arr) - (N+T) + 1)*0.6).astype(int):]

x_train=np.array(x_train)/1000
y_train=np.array(y_train)/1000
x_test=np.array(x_test)/1000
y_test=np.array(y_test)/1000

batch_size=1#Used for training Mini batch Gradient Descent if batch_size=1 Stochastic gradient descent
alpha=0.001#Learning Rate
w1, w2, w3, b1, b2, b3=model_fit(x_train, y_train, N, alpha,batch_size, epochs=1)
a1, a2, a3, z1, z2, z3=front_propogation(x_test.T,w1, w2, w3, b1, b2, b3)
y_pred=np.squeeze(a3*1000)
y_test=y_test*1000
```

```
Mean Error after 145 iterations is 0.00013373082249869422
```



In [47]:

```
plt.figure(figsize=(15,8))
plt.plot(ix[np.round((len(price_arr) - (N+T) + 1)*0.6).astype(int):],y_test, label='Actual Stock Price')
plt.plot(ix[np.round((len(price_arr) - (N+T) + 1)*0.6).astype(int):],y_pred,color='green', label='Predicted Stock
Price')
plt.legend()
plt.title("NN predicted Vs Ground Truth,T=1")
plt.xlabel('(N+T)th Day')
plt.ylabel('Stock Price')
plt.show()
```

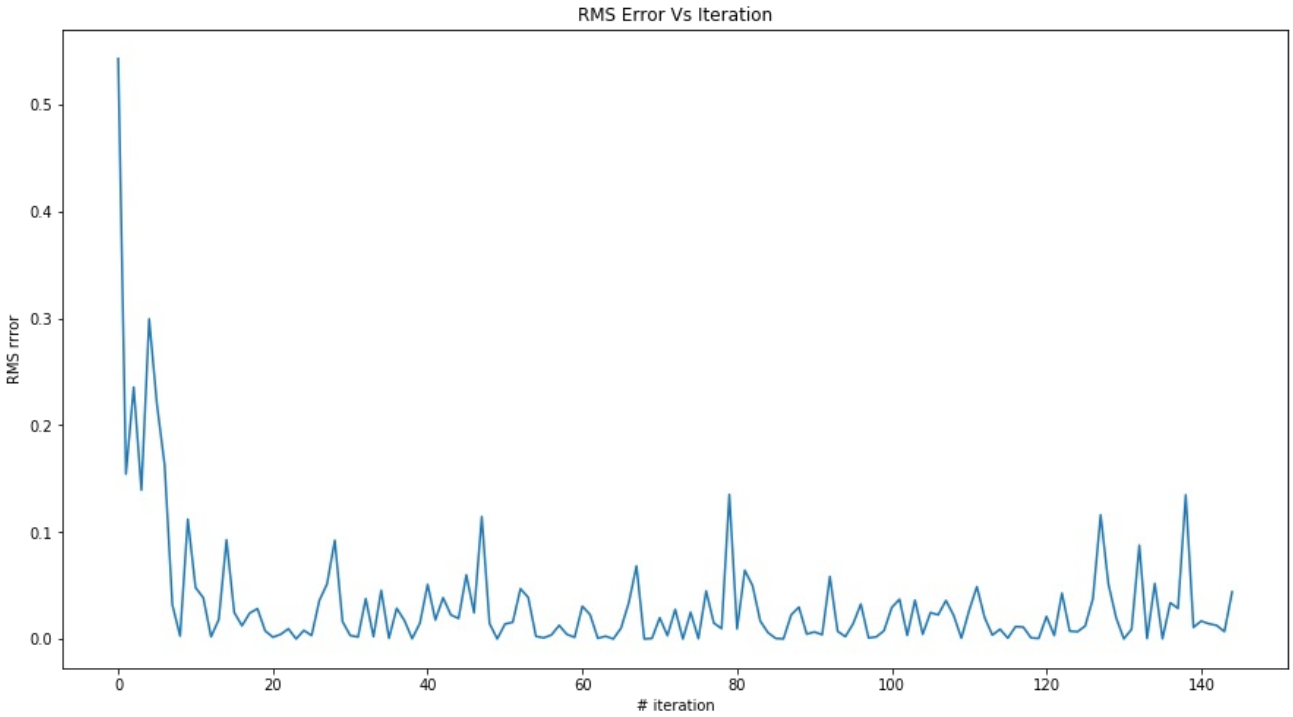


60 -40 random shuffle

In [50]:

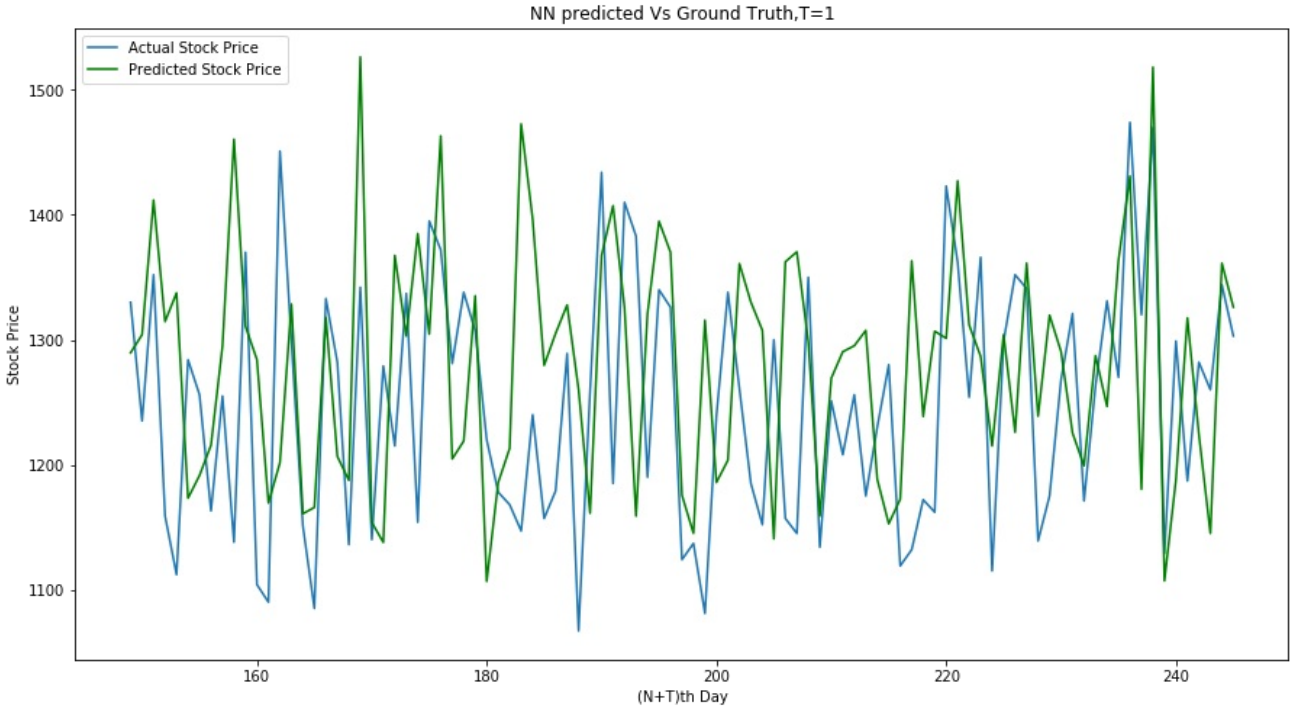
```
T = 1
N = 4
x_train1, y_train1,ix = Preprocess(price_arr, N, T)
x_train1=np.array(x_train1)
y_train1=np.array(y_train1)
np.random.seed(0)
np.random.shuffle(x_train1)
np.random.shuffle(y_train1)
x_train=x_train1[0:np.round((x_train1.shape[0]*0.6)).astype(int)]
x_test=x_train1[np.round((x_train1.shape[0]*0.6)).astype(int):]
y_train=y_train1[0:np.round((y_train1.shape[0]*0.6)).astype(int)]
y_test=y_train1[np.round((y_train1.shape[0]*0.6)).astype(int):]
x_train=np.array(x_train)/1000
y_train=np.array(y_train)/1000
x_test=np.array(x_test)/1000
y_test=np.array(y_test)/1000
batch_size=1#Used for training Mini batch Gradient Descent if batch_size=1 Stochastic gradient descent
alpha=0.001#Learning Rate
w1, w2, w3, b1, b2, b3=model_fit(x_train, y_train, N, alpha,batch_size, epochs=1)
a1, a2, a3, z1, z2, z3=front_propogation(x_test.T,w1, w2, w3, b1, b2, b3)
y_pred=np.squeeze(a3*1000)
y_test=y_test*1000
```

```
Mean Error after 145 iterations is 0.04405534649590057
```



In [51]:

```
plt.figure(figsize=(15,8))
plt.plot(ix[np.round((y_train1.shape[0]*0.6)).astype(int):],y_test, label='Actual Stock Price')
plt.plot(ix[np.round((y_train1.shape[0]*0.6)).astype(int):],y_pred,color='green', label='Predicted Stock Price')
plt.legend()
plt.title("NN predicted Vs Ground Truth,T=1")
plt.xlabel('(N+T)th Day')
plt.ylabel('Stock Price')
plt.show()
```

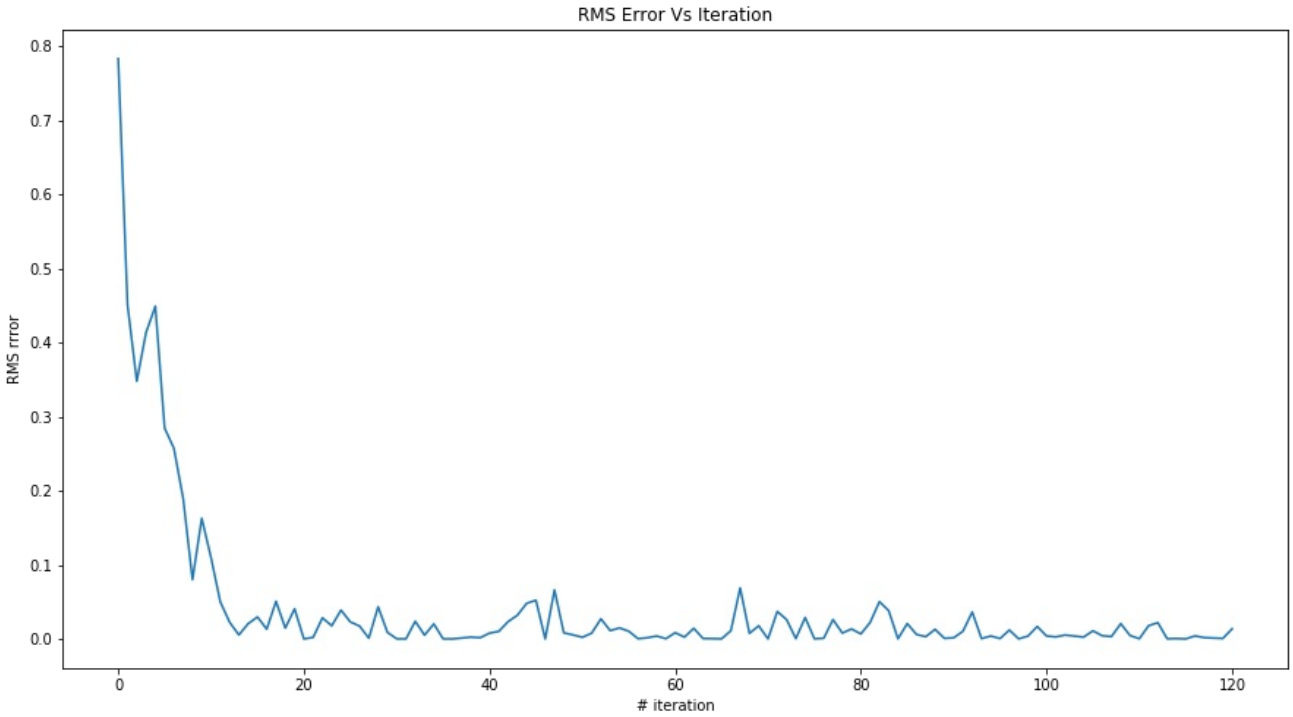


50%-10%-40% Random Shuffle of Data

In [55]:

```
T = 1
N = 4
x_train1, y_train1,ix = Preprocess(price_arr, N, T)
x_train1=np.array(x_train1)
y_train1=np.array(y_train1)
np.random.seed(0)
np.random.shuffle(x_train1)
x_train=x_train1[0:np.round((x_train1.shape[0]*0.5)).astype(int)]
x_val=x_train1[np.round((x_train1.shape[0]*0.5)).astype(int):np.round((x_train1.shape[0]*0.6)).astype(int)]
x_test=x_train1[np.round((x_train1.shape[0]*0.6)).astype(int):]
y_train=y_train1[0:np.round((y_train1.shape[0]*0.5)).astype(int)]
y_val=y_train1[np.round((y_train1.shape[0]*0.5)).astype(int):np.round((x_train1.shape[0]*0.6)).astype(int)]
y_test=y_train1[np.round((y_train1.shape[0]*0.6)).astype(int):]
x_train=np.array(x_train)/1000
y_train=np.array(y_train)/1000
x_test=np.array(x_test)/1000
y_test=np.array(y_test)/1000
x_val=np.array(x_val)/1000
y_val=np.array(y_val)/1000
batch_size=1#Used for training Mini batch Gradient Descent if batch_size=1 Stochastic gradient descent
alpha=0.001#Learning Rate
w1, w2, w3, b1, b2, b3=model_fit(x_train, y_train, N, alpha,batch_size, epochs=1)
```

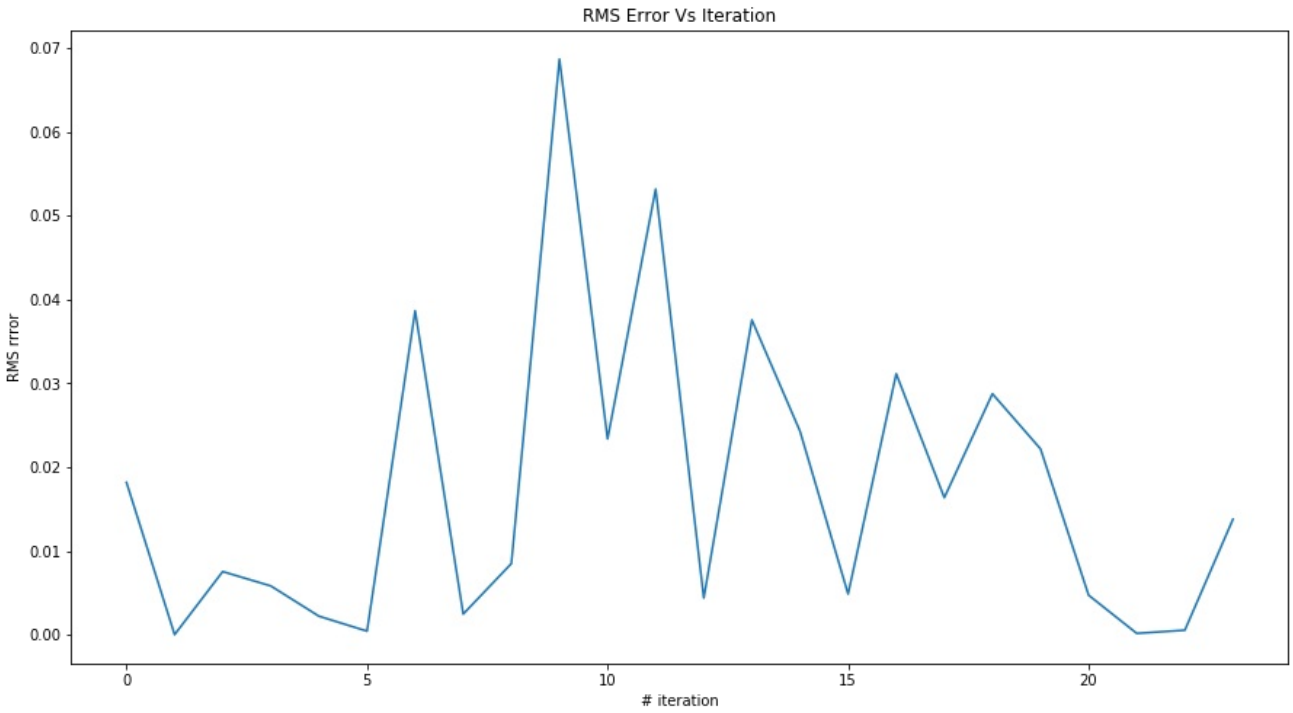
```
Mean Error after 121 iterations is 0.013702892527956189
```



In [58]:

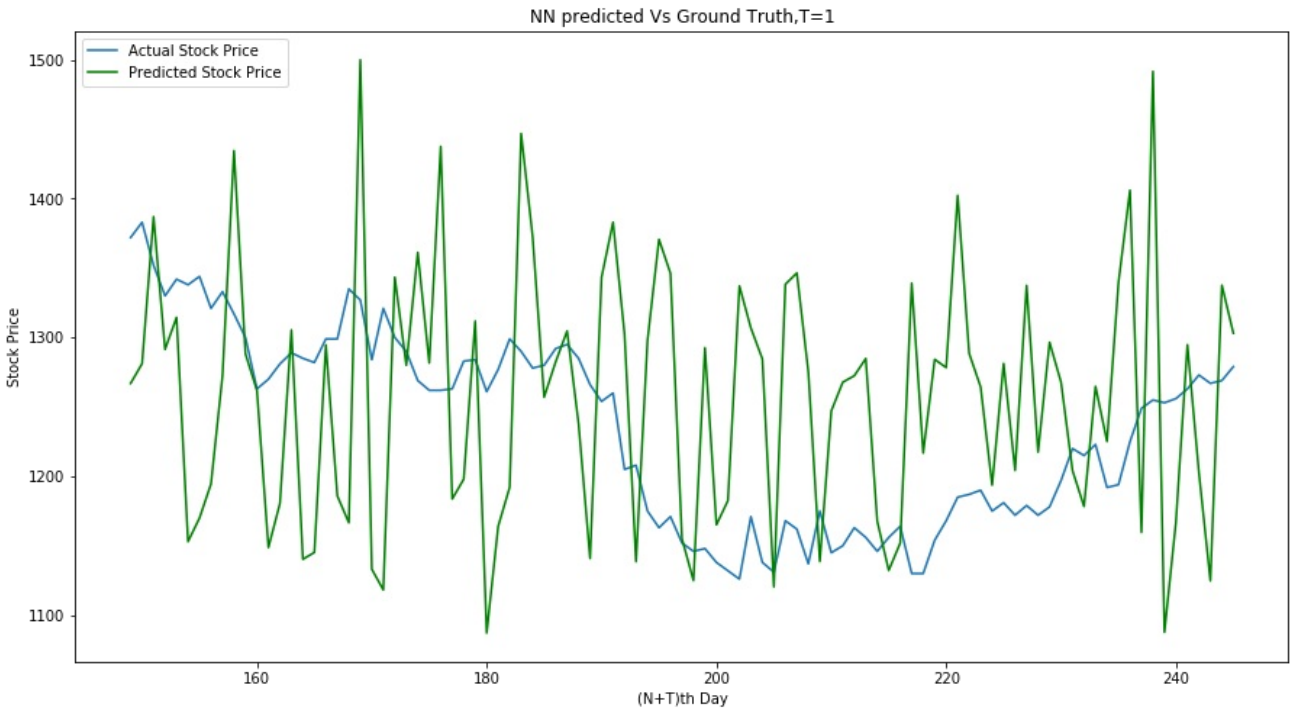
```
w1, w2, w3, b1, b2, b3=val_fit(x_val, y_val, N, alpha,batch_size,1,w1, w2, w3, b1, b2, b3)
a1, a2, a3, z1, z2, z3=front_propogation(x_test.T,w1, w2, w3, b1, b2, b3)
y_pred=np.squeeze(a3*1000)
y_test=y_test*1000
```

```
Mean Error after 24 iterations is 0.01378372426527778
```



In [60]:

```
plt.figure(figsize=(15,8))
plt.plot(ix[np.round((y_train1.shape[0]*0.6)).astype(int):],y_test, label='Actual Stock Price')
plt.plot(ix[np.round((y_train1.shape[0]*0.6)).astype(int):],y_pred,color='green', label='Predicted Stock Price')
plt.legend()
plt.title("NN predicted Vs Ground Truth,T=1")
plt.xlabel('(N+T)th Day')
plt.ylabel('Stock Price')
plt.show()
```



Two Moon -Softmax

Shallow NeuralNetwork for classification using SoftMax activation

Input layer has 2 neurons xposition,yposition

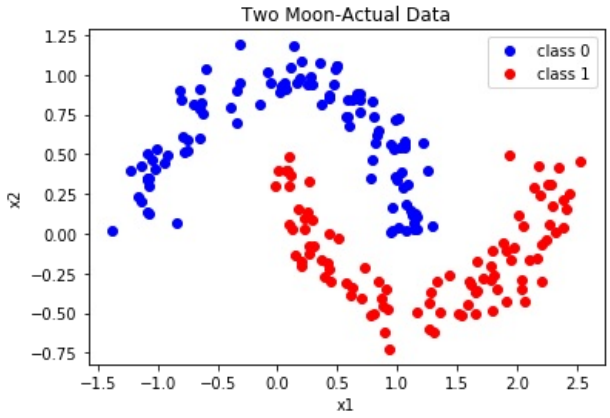
1 Hidden layer has 1 neuron

2 output layer neurons corresponding to class 0 and 1

Actual Two Moon Data Plot

In [4]:

```
plt.plot(x1[y==1],x2[y==1], 'bo',label='class 0')#data corresponding to 0
plt.plot(x1[y==0],x2[y==0], 'ro',label='class 1')#data corresponding to 1
plt.title("Two Moon-Actual Data")
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.show()
```



In [3]:

```
import numpy as np
import matplotlib.pyplot as plt

## Parsing Data from Text file
with open('two_moon.txt', 'r') as f:
    lines=f.readlines()

xpos=[]
ypos=[]
clas=[]
for i in range(4,len(lines)):
        j=lines[i].strip().find('\t')
        k=(lines[i].strip())[0]
        for l in range(1,j):
                k+=(lines[i].strip())[l]
        xpos.append(k)
        j2=lines[i].strip().find('\t',j+2)
        q=(lines[i].strip())[j+1]
        for l in range(j+2,j2):
                q+=(lines[i].strip())[l]
        ypos.append(q)
        clas.append((lines[i].strip())[j2+1])
#Reshaping Data to be used by the Sofmax model
x=np.array(xpos)
x1= x.astype(np.float64)
y=np.array(ypos)
x2= y.astype(np.float64)
c=np.array(clas)
y=c.astype(np.int)
X=np.vstack((x1,x2))
X=X.T
a=np.array([0,1])#if 0
b=np.array([1,0])#if 1
if y[0]==1:
    Y=np.copy(a)

if y[0]==0:
    Y=np.copy(b)
for i in range(1,y.shape[0]):
    if y[i]==1:
        Y=np.vstack((Y,a))

    if y[i]==0:
        Y=np.vstack((Y,b))
```

Splitting the Data int 50% train, 20% validate, 30% test

Trainging using train dataset

In [6]:

```
W,b=Param_init()
Wt,bt,probst=SoftMax(X_train,Y_train,W,b)
print("\n")
print("Training data accuracy")
Accuracy_Softmax(Wt,bt,X_train,Y_train)
```

```
iteration 0: loss 0.691849
iteration 10: loss 0.335310
iteration 20: loss 0.303971
iteration 30: loss 0.288907
iteration 40: loss 0.280170
iteration 50: loss 0.274706
iteration 60: loss 0.271115
iteration 70: loss 0.268664
iteration 80: loss 0.266942
iteration 90: loss 0.265705
iteration 100: loss 0.264799
iteration 110: loss 0.264126
iteration 120: loss 0.263620
iteration 130: loss 0.263235
iteration 140: loss 0.262941
iteration 150: loss 0.262713
iteration 160: loss 0.262537
iteration 170: loss 0.262399
iteration 180: loss 0.262291
iteration 190: loss 0.262206


Training data accuracy
Accuracy for Softmax Prediction is
88.0
```

Using Validation set for generalising

In [7]:

```
Wv,bv,probsv=SoftMax(X_val,Y_val,Wt,bt)
print("\n")
print("Validation Data training accuracy")
Accuracy_Softmax(Wv,bv,X_val,Y_val)
```

```
iteration 0: loss 0.320639
iteration 10: loss 0.286310
iteration 20: loss 0.283641
iteration 30: loss 0.282086
iteration 40: loss 0.281050
iteration 50: loss 0.280287
iteration 60: loss 0.279688
iteration 70: loss 0.279200
iteration 80: loss 0.278792
iteration 90: loss 0.278445
iteration 100: loss 0.278148
iteration 110: loss 0.277892
iteration 120: loss 0.277671
iteration 130: loss 0.277478
iteration 140: loss 0.277310
iteration 150: loss 0.277164
iteration 160: loss 0.277035
iteration 170: loss 0.276923
iteration 180: loss 0.276824
iteration 190: loss 0.276736


Validation Data training accuracy
Accuracy for Softmax Prediction is
85.0
```

In [8]:

```
probsv,p,q=prob_scores(probsv)
Conf_Mat(probsv,Y_val)
```

```
Confusion Matrix :
            Predicted 0      Predicted 1
Actual 0          34       6       40
Actual 1          6        34      40
                 40        40
```

Model on Test Data

In [9]:

```
Wf,bf,probsf=SoftMax(X_test,Y_test,Wv,bv)
print("\n")
print("Accuracy For Test Data")
Accuracy_Softmax(Wf,bf,X_test,Y_test)
```

```
iteration 0: loss 0.350693
iteration 10: loss 0.257974
iteration 20: loss 0.253511
iteration 30: loss 0.252319
iteration 40: loss 0.251852
iteration 50: loss 0.251560
iteration 60: loss 0.251333
iteration 70: loss 0.251146
iteration 80: loss 0.250991
iteration 90: loss 0.250862
iteration 100: loss 0.250756
iteration 110: loss 0.250669
iteration 120: loss 0.250598
iteration 130: loss 0.250539
iteration 140: loss 0.250492
iteration 150: loss 0.250453
iteration 160: loss 0.250421
iteration 170: loss 0.250396
iteration 180: loss 0.250376
iteration 190: loss 0.250359


Accuracy For Test Data
Accuracy for Softmax Prediction is
90.0
```

In [10]:

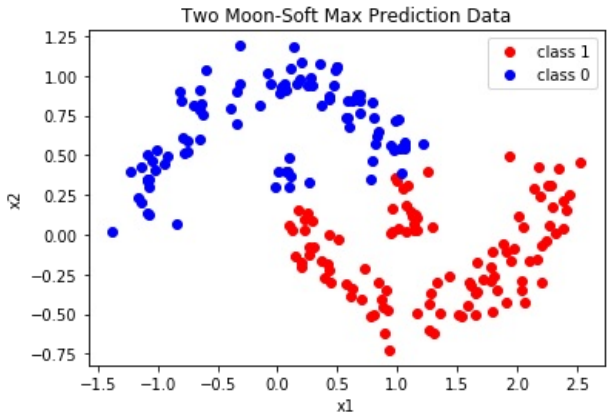
```
probsf,p,q=prob_scores(probsf)
Conf_Mat(probsf,Y_test)
```

```
Confusion Matrix :
            Predicted 0      Predicted 1
Actual 0          54       6       60
Actual 1          6        54      60
                 60        60
```

Soft-Max prediction of Two Moon data

In [11]:

```
prob,p,q=predict_Softmax(Wf,bf,X,Y)
plot_pred(x1,x2,q,p)
```



Logistic Regression

In [13]:

```
# initialize parameters randomly
W = np.random.randn(3,1)
# some hyperparameters
step_size = 1e-0
# gradient descent loop
num_examples = X.shape[0]
for i in range(200):
  # evaluate class scores, [N x K]
  scores = np.dot(X, W)
  sigmoid=1./(1+np.exp(-scores))
  data_loss = -np.matmul(y[y==1].T,np.log(sigmoid[y==1]))-np.matmul((1-y[y==0]).T,np.log(1-sigmoid[y==0]))
  loss = data_loss/num_examples

  if i % 10 == 0:
    print ("iteration %d: loss %f" % (i, loss))

  grad=(1/num_examples)*np.matmul((sigmoid-y).T,X)
  W=W-step_size*(grad.T)
```

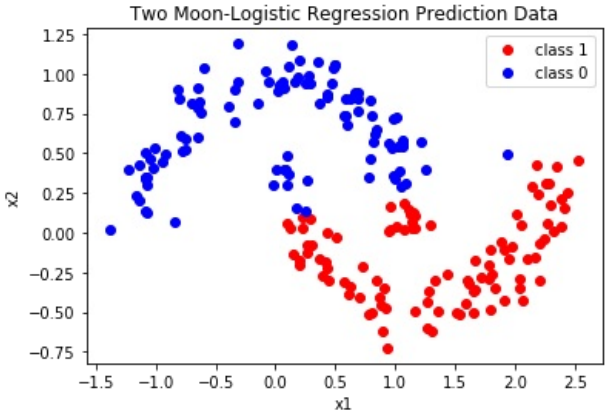
```
iteration 0: loss 1.604810
iteration 10: loss 0.479524
iteration 20: loss 0.376301
iteration 30: loss 0.343254
iteration 40: loss 0.325286
iteration 50: loss 0.313539
iteration 60: loss 0.305209
iteration 70: loss 0.299044
iteration 80: loss 0.294355
iteration 90: loss 0.290717
iteration 100: loss 0.287852
iteration 110: loss 0.285564
iteration 120: loss 0.283718
iteration 130: loss 0.282213
iteration 140: loss 0.280977
iteration 150: loss 0.279953
iteration 160: loss 0.279099
iteration 170: loss 0.278384
iteration 180: loss 0.277781
iteration 190: loss 0.277270
```

In [14]:

```
f_scores = np.dot(X, W)
pred=1./(1+np.exp(-f_scores))
pred[pred>=0.5]=1
pred[pred<0.5]=0
x1=x1.reshape(200,1)
x2=x2.reshape(200,1)
```

In [15]:

```
plt.plot(x1[pred==0],x2[pred==0], 'ro',label='class 1')
plt.plot(x1[pred==1],x2[pred==1], 'bo',label='class 0')
plt.title("Two Moon-Logistic Regression Prediction Data")
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.show()
```



In [16]:

```
acc=y==pred
pos=0
for i in acc:
    if i:
      pos+=1
accuracy=100*(pos/acc.shape[0])
print("Accuracy for Logistic Regression is")
print(accuracy)
```

```
Accuracy for Logistic Regression is
87.5
```

The classification Accuracy using

Logistic Regression is 87.5%

SoftMax Activation is 90%

In [17]:

```
yz=y==0
pz=pred==0
trneg=pz==yz
trueneg=0
falseneg=0
for i in trneg:
    if i:
        trueneg+=1
    else:
        falseneg+=1
yp=y==1
pp=pred==1
trpos=pp==yp
truepos=0
falsepos=0
for i in trpos:
    if i:
        truepos+=1

    else:
        falsepos+=1
first_row_sum = trueneg + falseneg
second_row_sum = falseneg + truepos
first_col_sum = trueneg + falseneg
second_col_sum = truepos + falsepos
print("Confusion Matrix : ")
print("\t    Predicted 0      Predicted 1 \t")
print("Actual 0 \t " + str(trueneg) + "\t  " + str(falsepos) + "\t  " + str(first_row_sum))
print("Actual 1 \t " + str(falseneg) + "\t  " + str(truepos) + "\t  " + str(second_row_sum))
print("\t \t" +str(first_col_sum) + "\t  " + str(second_col_sum))
```

```
Confusion Matrix :
            Predicted 0      Predicted 1
Actual 0          175      25      200
Actual 1          25       175     200
                 200       200
```