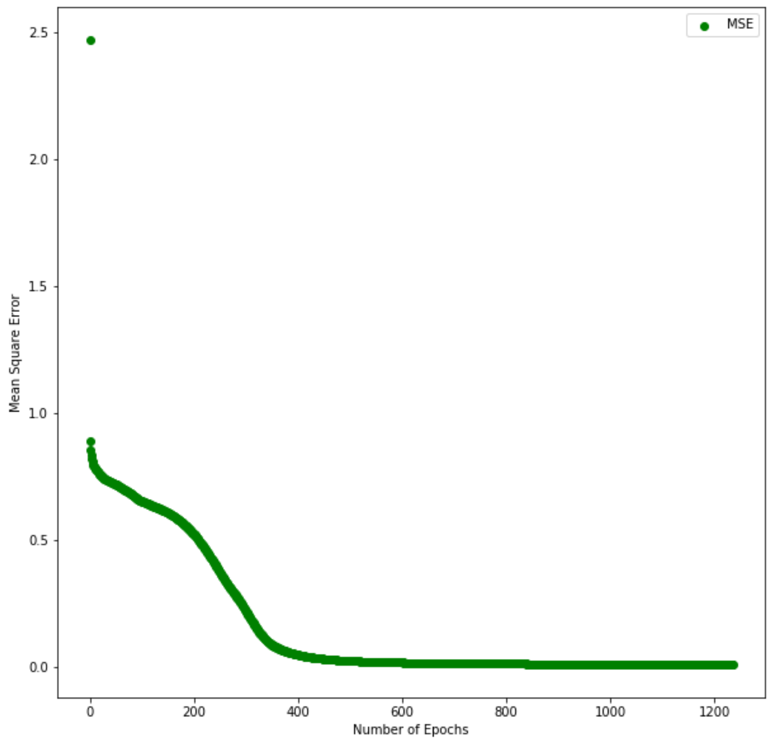
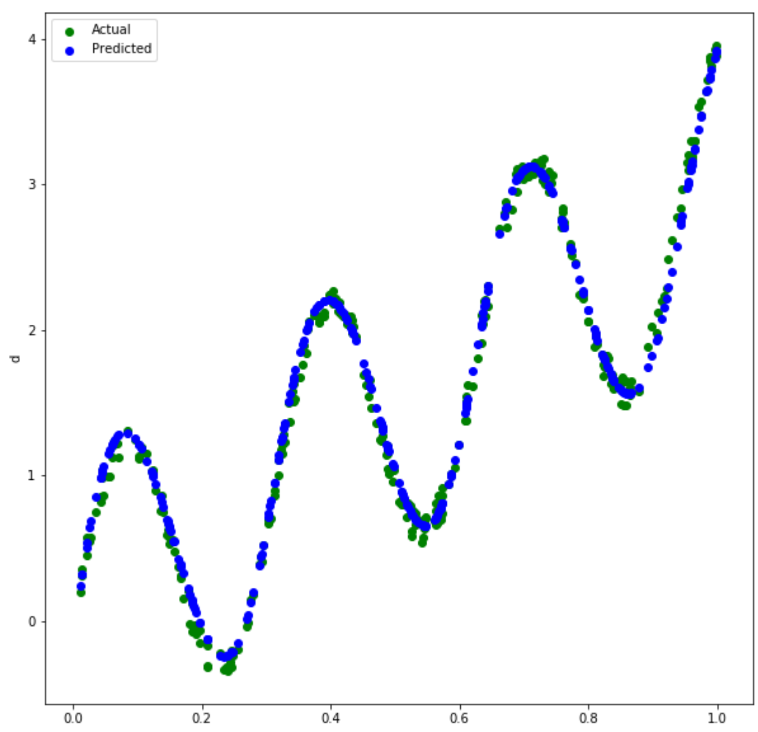
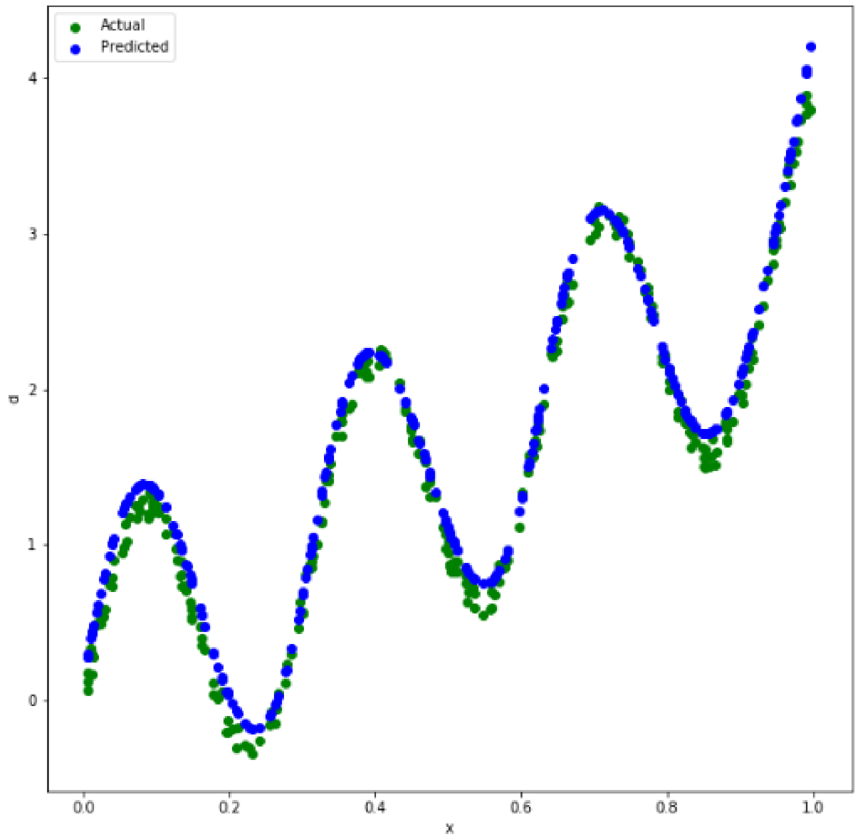
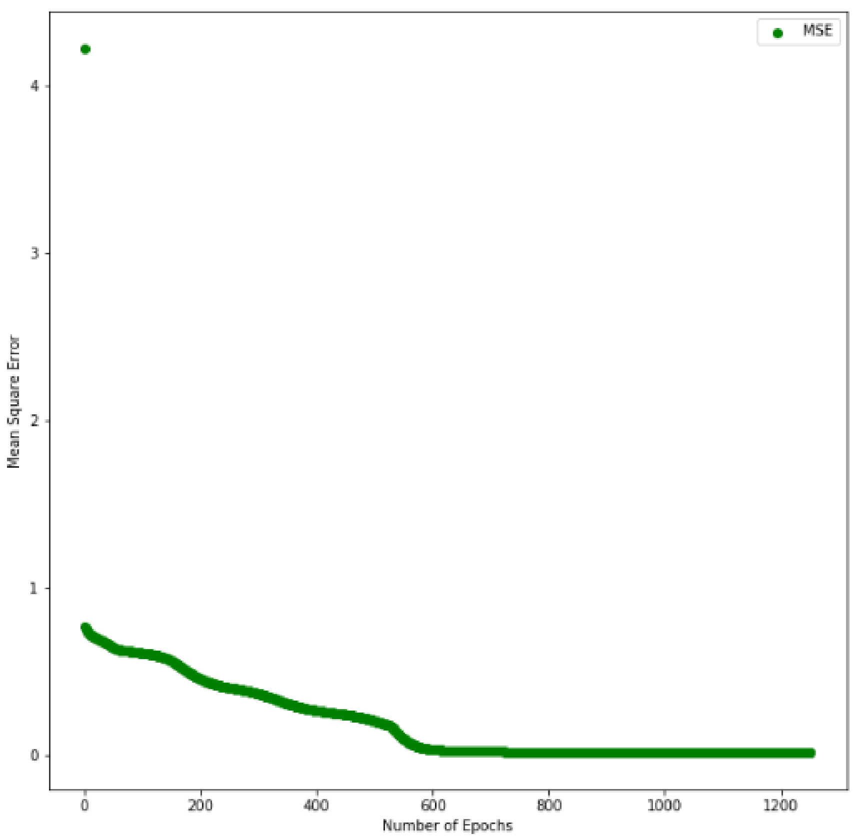
HW4

Q1.


MSE
Mean Square Error
Number of Epochs
2.5
2.0
1.5
1.0
0.5
0.0
0
200
400
600
800
1000
1200


Actual
Predicted
d
x
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0


Actual
Predicted
0
1
2
3
4
d
0.0
0.2
0.4
0.6
0.8
1.0
x


MSE
0
1
2
3
4
Mean Square Error
0
200
400
600
800
1000
1200
Number of Epochs

Psuedo code

0) Given n, x, v
1) Initialize $d_i = \sin(20x_i) + 3x_i + v_i$
2) Initialize feed forward activation functions.
3) Initialize feed back activation functions.
   3.1) tanh(v) – feedforward
   3.2) $1 - \tanh^2(v)$ – feedback (Derivative)
4) Initialize weights.
   4.1) Random uniform distribution for W_input & N = 24 (-5,5)
   4.2) " " " W_bias & N = 24 (-1,1)
   4.3) " " " W_output N = 24 (-5,5)
   4.4) " " " W_final N = 1 (-1,1)
5) Initialize output neuron
   5.1) v – function
   5.2) 1 – derivative.
6) Feed forward network.
   6.1) while(True)
      Do → for i < n=300
         alpha = (X[i] * W_input [j]) + W_bias [j]
         beta = matrix multiplication of alpha & w_output to final
*) Backpropagation.
   do e = –((d[i] – y[i]) * eta*2)/n
   ↳ for i < n=300.
      for j < N=24
         w_output (weight) = e * alpha.
         w_input (weight) = e * X[i] * w_output * derivative (tanh()).
         w_output – e * w_output[j] * derivative (tanh()).

8) Update the weight.
9) Find mean square error.
   for i < n=300.
      mse += $(d[i] - y[i])^2$
10) plot graph for mean square error & number of epochs.
11) plot Scatter plot of feed forward network output & desired output

Source Code

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

# input data
n = 300
x = np.random.uniform(low=0.0, high=1.0, size=n)
v = np.random.uniform(low=-0.1, high=0.1, size=n)

# desired output
d = []
for i in range(n):
    d.append(np.sin(20*x[i]) + (3*x[i]) + v[i])

fig, ax = plt.subplots(figsize=(10,10))
plt.xlabel('x')
plt.ylabel('d')
plt.scatter(x,d, c = 'green', label = 'Actual')
plt.legend(loc = 'best')
plt.show()

# feed-forward activation functions
def act_fun(v):
    return np.tanh(v)

def act_op(v):
    return v

# feedback activation functions
def derv_act_fun(v):
    return (1 - np.tanh(v)**2)

def derv_act_op(v):
    return 1

# weight initialization
N = 24
w_input = np.random.uniform(low=-5, high=5, size=N)
w_bias = np.random.uniform(low=-1, high=1, size=N)
w_output = np.random.uniform(low=-5, high=5, size=N)
w_final = np.random.uniform(low=-1, high=1, size=1)
eta = 6
```

```
list_mse = []
z = 0
while(True):
    # feed-forward network
    u = []
    y = []
    alphas = []
    betas = []
    for i in range(n):
        v = []
        temp = []
        for j in range(N):
            alpha = (x[i]*w_input[j]) + w_bias[j]
            temp.append(alpha)
            v.append(act_fun(alpha))
        alphas.append(temp)
        u.append(v)
        beta = np.matmul(np.array(u[i]),w_output) + w_final
        betas.append(beta[0])
        y.append(act_op(beta[0]))

        # backpropagation
        e = -((d[i] - y[i])*eta*2)/n
        w_output_grad = []
        w_input_grad = []
        w_bias_grad = []
        w_final_grad = []
        delta_final = - e
        w_final_grad.append(delta_final)
        for j in range(N):
            delta_u = e * u[i][j]
            w_output_grad.append(delta_u)
            delta_w = e  * x[i] * w_output[j] * derv_act_fun(alphas[i][j])
            w_input_grad.append(delta_w)
            delta_bias = e * w_output[j] * derv_act_fun(alphas[i][j])
            w_bias_grad.append(delta_bias)
        # weight update
        w_input = np.subtract(w_input, np.asarray(w_input_grad))
        w_output = np.subtract(w_output, np.asarray(w_output_grad))
        w_bias = np.subtract(w_bias, np.asarray(w_bias_grad))
        w_final = np.subtract(w_final, np.asarray(w_final_grad))
    # mean square error
    mse = 0
    for i in range(n):
        mse += (d[i] - y[i])**2
    mse = mse/n
    list_mse.append(mse)

    print (mse, eta, z)

    if list_mse[z] > list_mse[z-1]:
        eta = 0.9*eta
    if list_mse[-1]<0.01:
        break
    z += 1
```

```
fig, ax = plt.subplots(figsize=(10,10))
plt.ylabel('Mean Square Error')
plt.xlabel('Number of Epochs')
plt.scatter(range(len(list_mse)), list_mse, c = 'green', label = 'MSE')
plt.legend(loc = 'best')
plt.show()

fig, ax = plt.subplots(figsize=(10,10))
plt.ylabel('d')
plt.xlabel('x')
plt.scatter(x,d, c = 'green', label = 'Actual')
plt.scatter(x,y, c = 'blue', label = 'Predicted')
plt.legend(loc = 'best')
plt.show()
```