CS 559 - Fall 2021
Homework #5

Krishna Garg
kgarg8@uic.edu

1 Problem Formulation & Gradient Calculations

$E = \frac{1}{2}(D_i - y_i)^2$

$\phi_1(v) = tanh(v)$

$\phi_2(v) = v$

$$\begin{aligned} y_i &= \phi_2(W_2 \phi_1(W_1 X_i + B_1) + B_2) \\ &= W_2 \cdot tanh(W_1 X_i + B_1) + B_2 \end{aligned} \tag{1}$$

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial y_i} * \frac{\partial y_i}{\partial w} \tag{2}$$

CALCULATING $\frac{\partial E}{\partial y_i}$:

$$\frac{\partial E}{\partial y_i} = -(D_i - y_i) \tag{3}$$

CALCULATING $\frac{\partial y_i}{\partial w}$:

$$\begin{aligned} \frac{\partial y_i}{\partial W_2} &= tanh(W_1 X_i + B_1) \\ \frac{\partial y_i}{\partial W_1} &= W_2 \cdot (1 - tanh^2(W_1 X_i + B_1)) \cdot X_i \\ \frac{\partial y_i}{\partial B_2} &= 1 \\ \frac{\partial y_i}{\partial B_1} &= W_2 \cdot (1 - tanh^2(W_1 X_i + B_1)) \end{aligned} \tag{4}$$

2 Pseudo code for Training Algorithm

Initialize weights & biases W_1, W_2, B_1, B_2
Initialize X, V, τ, η
D = sin(20X) + 3X + V
while loss $> \tau$ **do**
    loss = 0
    for i in range(len(X)) **do**
        Calculate y_i using eqn. (1)
        loss +=$(D_i - y_i)^2$
        Calculate gradients $\frac{\partial E}{\partial w}$ using eqns. (2, 3, 4) where $w = \{W_1, W_2, B_1, B_2\}$
        $w \leftarrow w - \eta * \frac{\partial E}{\partial w}$

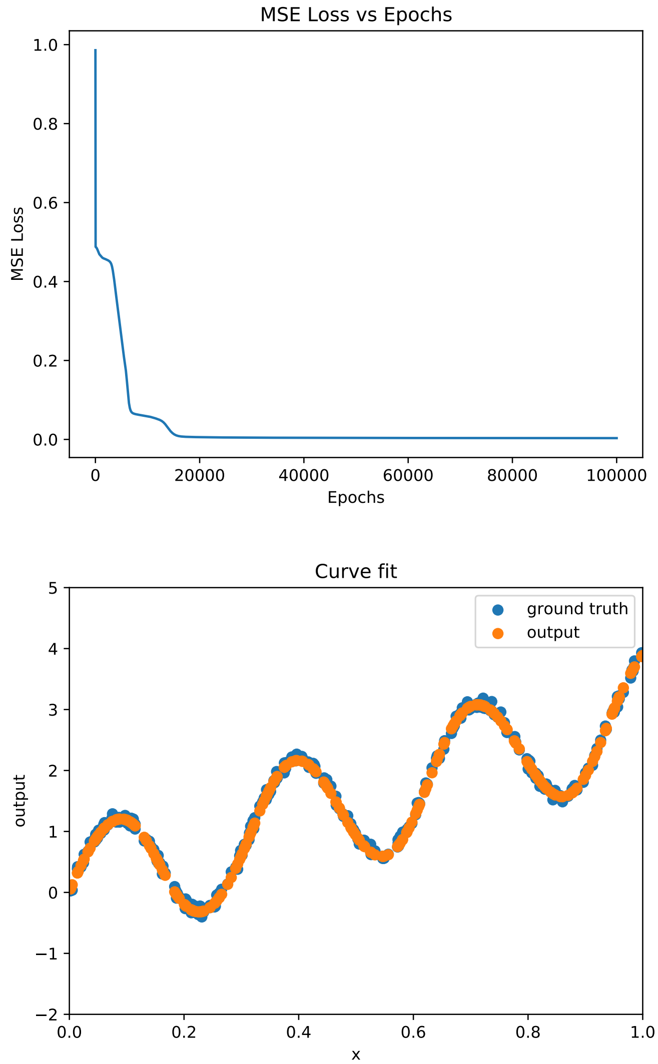

Figure 1: Gradient Descent Method

```
    end for
    loss = loss / len(X)
end while
```

Code

```
import math, numpy as np, matplotlib.pyplot as plt, pdb

seed = 112
np.random.seed(seed)
n = 300  # number of inputs
N = 24   # hidden neurons
eta = 0.001 # learning rate

X = np.random.uniform(0,1,n) # inputs
V = np.random.uniform(-0.1,0.1,n)
D = np.sin(20*X) + 3*X + V # desired values

# weights & biases
W1 = np.random.uniform(-0.1,0.1,N) # layer 1
B1 = np.random.uniform(-0.1,0.1,N) # layer 1
W2 = np.random.uniform(-0.5,0.5,N) # layer 2
b2 = np.random.uniform(0,1,1) # layer 2

epochs = 0
losses = []
while(epochs < 100000): # per epoch
   loss = 0
   for i in range(n):
      # forward pass
      y1 = W1 * X[i] + B1
      y2 = np.sum(W2 * np.tanh(y1)) + b2

      # calculate loss
      loss += (D[i] - y2)**2

      # weight update (backward pass)
      # w <- w - eta * (dE/ dw) # Gradient Descent equation
      W2 = W2 + eta * (D[i] - y2) * np.tanh(y1)
      W1 = W1 + (eta*5) * (D[i] - y2) * (1 - np.tanh(y1)**2) * X[i] * W2
          # observe the higher lr used for the initial layer
      B1 = B1 + (eta*5) * (D[i] - y2) * (1 - np.tanh(y1)**2) * W2
      b2 = b2 + eta * (D[i] - y2) * 1

   losses.append(loss/n)
   if epochs % 1000 == 0:
      print(loss/n)
   epochs += 1

epoch_arr = list(range(1, epochs+1))
plt.figure()
plt.xlabel('Epochs')
plt.ylabel('MSE Loss')
plt.plot(epoch_arr, losses)
plt.title('MSE Loss vs Epochs')
plt.savefig('MSE_Loss_during_BackPropagation_eta_{}.pdf'.format(eta))
plt.show()

output =[]
for i in range(n):
   y1 = W1 * X[i] + B1
   y2 = np.sum(W2 * np.tanh(y1)) + b2
   output.append(y2)

plt.figure()
plt.xlabel('x')
plt.ylabel('output')
plt.axis([0, 1, -2, 5])
```

```
plt.scatter(X, D, label='ground truth')
plt.scatter(X, output, label='output')
plt.legend()
plt.title('Curve fit')
plt.savefig('Curve Fit.pdf')
plt.show()
```