**Problem 1: Forward vs Backward Differentiation**

Forward-mode differentiation tracks how one input affects every node. Reverse-mode differentiation tracks how every node affects one output. That is, forward-mode differentiation applies the operator ∂/∂x to every node, while reverse mode differentiation applies the operator ∂f/∂ to every node.



a. Apply forward differentiation operator ∂/∂y to each node in the graph i.e., compute ∂x/∂y, ∂y/∂y, ∂p/∂y, ∂q/∂y, ∂f/∂y.

**[Ans]**

∂x/∂y = 0

∂y/∂y = 1

∂p/∂y = ∂x/∂y + ∂y/∂y = 0 + 1 = 1

∂q/∂y = ∂y/∂y + ∂2/∂y = 1 + 0 = 1

∂f/∂y = ∂(p\*q)/∂y = p\*∂q/∂y + q\*∂p/∂y = (x+y)\*1 + (y+2)\*1 = x + 2y + 2

b. Apply forward differentiation operator ∂/∂x to each node in the graph i.e., compute ∂x/∂x, ∂y/∂x, ∂p/∂x, ∂q/∂x, ∂f/∂x.

**[Ans]**

∂x/∂x = 1

∂y/∂x = 0

∂p/∂x = ∂x/∂x + ∂y/∂x = 1 + 0 = 1

∂q/∂x = ∂y/∂x + ∂2/∂x = 0 + 0 = 0

∂f/∂x = ∂(p\*q)/∂x = p\*∂q/∂x + q\*∂p/∂x = (x+y)\*0 + (y+2)\*1 = y + 2

c. Apply backward differentiation operator ∂f/∂ to each node in the graph i.e., compute ∂f/∂f, ∂f/∂p, ∂f/∂q, ∂f/∂y, ∂f/∂x.

**[Ans]**

∂f/∂f = 1

∂f/∂p = ∂(p\*q)/∂p = q = y + 2

∂f/∂q = ∂(p\*q)/∂q = p = x + y

∂f/∂y = (∂f/∂p \* ∂p/∂y) + (∂f/∂q \* ∂q/∂y) = (y+2)\*1 + (x+y)\*1 = x + 2y + 2

∂f/∂x = (∂f/∂p \* ∂p/∂x) + (∂f/∂q \* ∂q/∂x) = (y+2)\*1 + (x+y)\*0 = y + 2

d. Which mode of differentiation is efficient to compute ∂f/∂x, ∂f/∂y.

**[Ans]**

We needed 2 forward differentiations, 1 for ∂f/∂x, and 1 for ∂f/∂y. But we needed only only 1 backward differentiation to compute both of them. Hence, backward differentiation is more efficient.

**Problem 2: Backpropagation in neural network with linear perceptrons**

Given the following neural network with linear perceptrons, do the following:



a. Apply backward differentiation operator ∂E/∂ to each weight in the graph i.e., Compute ∂E/∂w13, ∂E/∂w23, ∂E/∂wx1, ∂E/∂wx2 using back propagation.

**[Ans]**

E = ½ \* (y – o3)2

=> ∂E/∂o3 = o3 – y

o3 = w13\*o1 + w23\*o2

=> ∂o3/∂w13 = o1, ∂o3/∂w23 = o2, ∂o3/∂o1 = w13, ∂o3/∂o2 = w23

o1 = wx1\*x

=> ∂o1/∂wx1 = x

o2 = wx2\*x

=> ∂o2/∂wx2 = x

∂E/∂w13 = ∂E/∂o3 \* ∂o3/∂w13 = (o3 – y) \* o1

∂E/∂w23 = ∂E/∂o3 \* ∂o3/∂w23 = (o3 – y) \* o2

∂E/∂wx1 = ∂E/∂o1 \* ∂o1/∂wx1

= ∂E/∂o3 \* ∂o3/∂o1 \* ∂o1/∂ wx1

= (o3 – y) \* w13 \* x

∂E/∂wx2 = ∂E/∂o2 \* ∂o2/∂wx2

= ∂E/∂o3 \* ∂o3/∂o2 \* ∂o2/∂ wx2

= (o3 – y) \* w23 \* x

b. Given x=1, y = 6, apply gradient descent algorithm to find the values of weights that minimize the squared loss function, E, defined in the class. Assume following equation for gradient updates and also assume the initial weight values as 1.

wij = wij – 0.01 \* ∂E/∂wij

Repeat the algorithm for 3 iterations and observe how the weights gets adjusted.

**[Ans]**

x = 1, y = 6

*Iteration 1:*

w13 = w23 = wx1 = wx2 = 1

o1 = wx1 \* x = 1 \* 1 = 1

o2 = wx2 \* x = 1 \* 1 = 1

o3 = (w13 \* o1) + (w23 \* o2) = (1 \* 1) + (1 \* 1) = 1 + 1 = 2

∂E/∂w13 = (o3 – y) \* o1 = (2 – 6) \* 1 = -4

∂E/∂w23 = (o3 – y) \* o2 = (2 – 6) \* 1 = -4

∂E/∂wx1 = (o3 – y) \* w13 \* x = (2 – 6) \* 1 \* 1 = -4

∂E/∂wx2 = (o3 – y) \* w23 \* x = (2 – 6) \* 1 \* 1 = -4

w13 = w13 – 0.01 \* ∂E/∂w13 = 1 – 0.01 \* (-4) = 1.04

w23 = w23 – 0.01 \* ∂E/∂w23 = 1 – 0.01 \* (-4) = 1.04

wx1 = wx1 – 0.01 \* ∂E/∂wx1 = 1 – 0.01 \* (-4) = 1.04

wx2 = wx2 – 0.01 \* ∂E/∂wx2 = 1 – 0.01 \* (-4) = 1.04

*Iteration 2:*

w13 = w23 = wx1 = wx2 = 1.04

o1 = wx1 \* x = 1.04 \* 1 = 1.04

o2 = wx2 \* x = 1.04 \* 1 = 1.04

o3 = (w13 \* o1) + (w23 \* o2) = (1.04 \* 1.04) + (1.04 \* 1.04) = 2.1632

∂E/∂w13 = (o3 – y) \* o1 = (2.1632 – 6) \* 1.04 = -3.990272

∂E/∂w23 = (o3 – y) \* o2 = (2.1632 – 6) \* 1.04 = -3.990272

∂E/∂wx1 = (o3 – y) \* w13 \* x = (2.1632 – 6) \* 1.04 \* 1 = -3.990272

∂E/∂wx2 = (o3 – y) \* w23 \* x = (2.1632 – 6) \* 1.04 \* 1 = -3.990272

w13 = w13 – 0.01 \* ∂E/∂w13 = 1.04 – 0.01 \* (-3.990272) = 1.0799

w23 = w23 – 0.01 \* ∂E/∂w23 = 1.04 – 0.01 \* (-3.990272) = 1.0799

wx1 = wx1 – 0.01 \* ∂E/∂wx1 = 1.04 – 0.01 \* (-3.990272) = 1.0799

wx2 = wx2 – 0.01 \* ∂E/∂wx2 = 1.04 – 0.01 \* (-3.990272) = 1.0799

*Iteration 3:*

w13 = w23 = wx1 = wx2 = 1.0799

o1 = wx1 \* x = 1.0799 \* 1 = 1.0799

o2 = wx2 \* x = 1.0799 \* 1 = 1.0799

o3 = (w13 \* o1) + (w23 \* o2) = (1.0799 \* 1.0799) + (1.0799 \* 1.0799) = 2.33238

∂E/∂w13 = (o3 – y) \* o1 = (2.33238 – 6) \* 1.0799 = -3.96067

∂E/∂w23 = (o3 – y) \* o2 = (2.33238 – 6) \* 1.0799 = -3.96067

∂E/∂wx1 = (o3 – y) \* w13 \* x = (2.33238 – 6) \* 1.0799 \* 1 = -3.96067

∂E/∂wx2 = (o3 – y) \* w23 \* x = (2.33238 – 6) \* 1.0799 \* 1 = -3.96067

w13 = w13 – 0.01 \* ∂E/∂w13 = 1.0799 – 0.01 \* (-3.96067) = 1.11951

w23 = w23 – 0.01 \* ∂E/∂w23 = 1.0799 – 0.01 \* (-3.96067) = 1.11951

wx1 = wx1 – 0.01 \* ∂E/∂wx1 = 1.0799 – 0.01 \* (-3.96067) = 1.11951

wx2 = wx2 – 0.01 \* ∂E/∂wx2 = 1.0799 – 0.01 \* (-3.96067) = 1.11951

**Problem 3: Backpropagation in neural network with sigmoid perceptrons**

Given the following neural network with linear perceptrons, do the following:



a. Apply backward differentiation operator ∂E/∂ to each weight in the graph i.e., Compute ∂E/∂w13, ∂E/∂w23, ∂E/∂wx1, ∂E/∂wx2 using back propagation.

**[Ans]**

E = ½ \* (y – o3)2

=> ∂E/∂o3 = o3 – y

a3 = w13\*o1 + w23\*o2

o3 = sigmoid(a3)

= sigmoid(w13\*o1 + w23\*o2)

=>

∂o3/∂w13 = o1\*sigmoid’(a3)

∂o3/∂w23 = o2\*sigmoid’(a3)

∂o3/∂o1 = w13\*sigmoid’(a3)

∂o3/∂o2 = w23\*sigmoid’(a3)

*Note: sigmoid’(x) = sigmoid(x)\*[1 – sigmoid(x)]*

a1 = wx1\*x

o1 = sigmoid(a1) = sigmoid(wx1\*x)

=>

∂o1/∂wx1 = x\*sigmoid’(a1)

a2 = wx2\*x

o2 = sigmoid(a2) = sigmoid(wx2\*x)

=>

∂o2/∂wx2 = x\*sigmoid’(a2)

*Error gradients calculation*

∂E/∂w13 = ∂E/∂o3 \* ∂o3/∂w13

= (o3 – y)\*o1\*sigmoid’(a3)

∂E/∂w23 = ∂E/∂o3 \* ∂o3/∂w23

= (o3 – y)\*o2\*sigmoid’(a3)

∂E/∂wx1 = ∂E/∂o1 \* ∂o1/∂wx1

= (∂E/∂o3 \* ∂o3/∂o1)\* ∂o1/∂ wx1

= (o3 – y)\*w13\*sigmoid’(a3)\* x\*sigmoid’(a1)

∂E/∂wx2 = ∂E/∂o2 \* ∂o2/∂wx2

= (∂E/∂o3 \* ∂o3/∂o2)\* ∂o2/∂ wx2

= (o3 – y)\*w23\*sigmoid’(a3)\*x\*sigmoid’(a2)

b. Given x=0.2, y = 1, apply gradient descent algorithm to find the values of weights that minimize the squared loss function, E, defined in the class. Assume following equation for gradient updates and also assume the initial weight values as 1.

wij = wij – 0.01 \* ∂E/∂wij

Repeat the algorithm for 3 iterations and observe how the weights gets adjusted.

**[Ans]**

x = 0.2, y = 1

*Iteration 1:*

w13 = w23 = wx1 = wx2 = 1

a1 = wx1\*x = 1\*0.2 = 0.2

o1 = sigmoid(a1) = sigmoid(0.2) = 0.5498

a2 = wx2\*x = 1\*0.2 = 0.2

o2 = sigmoid(a2) = sigmoid(0.2) = 0.5498

a3 = w13\*o1 + w23\*o2 = 1.0996

o3 = sigmoid(a3) = sigmoid(1.0996) = 0.7502

∂E/∂w13 = (o3 – y)\*o1\*sigmoid’(a3) = -0.0257

∂E/∂w23 = (o3 – y)\*o2\*sigmoid’(a3) = -0.0257

∂E/∂wx1 = (o3 – y)\*w13\*sigmoid’(a3)\* x\*sigmoid’(a1) = -0.0023

∂E/∂wx2 = (o3 – y)\*w23\*sigmoid’(a3)\*x\*sigmoid’(a2) = -0.0023

w13 = w13 – 0.01 \* ∂E/∂w13 = 1 – 0.01 \* (-0.0257) = 1.000257

w23 = w23 – 0.01 \* ∂E/∂w23 = 1 – 0.01 \* (-0.0257) = 1.000257

wx1 = wx1 – 0.01 \* ∂E/∂wx1 = 1 – 0.01 \* (-0.0023) = 1.000023

wx2 = wx2 – 0.01 \* ∂E/∂wx2 = 1 – 0.01 \* (-0.0023) = 1.000023

*Iteration 2:*

a1 = wx1\*x = 0.2000046

o1 = sigmoid(a1) = 0.5498

a2 = wx2\*x = 1\*0.2 = 0.2000046

o2 = sigmoid(a2) = 0.5498

a3 = w13\*o1 + w23\*o2 = 1.09995

o3 = sigmoid(a3) = 0.75025

∂E/∂w13 = (o3 – y)\*o1\*sigmoid’(a3) = -0.02573

∂E/∂w23 = (o3 – y)\*o2\*sigmoid’(a3) = -0.02573

∂E/∂wx1 = (o3 – y)\*w13\*sigmoid’(a3)\* x\*sigmoid’(a1) = -0.0023171

∂E/∂wx2 = (o3 – y)\*w23\*sigmoid’(a3)\*x\*sigmoid’(a2) = -0.0023171

w13 = w13 – 0.01 \* ∂E/∂w13 = 1.00051

w23 = w23 – 0.01 \* ∂E/∂w23 = 1.00051

wx1 = wx1 – 0.01 \* ∂E/∂wx1 = 1.000046

wx2 = wx2 – 0.01 \* ∂E/∂wx2 = 1.000046

*Iteration 3:*

a1 = wx1\*x = 0.2000092

o1 = sigmoid(a1) = 0.5498

a2 = wx2\*x = 1\*0.2 = 0.2000092

o2 = sigmoid(a2) = 0.5498

a3 = w13\*o1 + w23\*o2 = 1.100238

o3 = sigmoid(a3) = 0.75030

∂E/∂w13 = (o3 – y)\*o1\*sigmoid’(a3) = -0.02572

∂E/∂w23 = (o3 – y)\*o2\*sigmoid’(a3) = -0.02572

∂E/∂wx1 = (o3 – y)\*w13\*sigmoid’(a3)\* x\*sigmoid’(a1) = -0.0023169

∂E/∂wx2 = (o3 – y)\*w23\*sigmoid’(a3)\*x\*sigmoid’(a2) = -0.0023169

w13 = w13 – 0.01 \* ∂E/∂w13 = 1.00077

w23 = w23 – 0.01 \* ∂E/∂w23 = 1.00077

wx1 = wx1 – 0.01 \* ∂E/∂wx1 = 1.000069

wx2 = wx2 – 0.01 \* ∂E/∂wx2 = 1.000069