COMP4434 Big Data Analytics

Assignment #2

The variables and functions are defined as follows:

```
In [1]: import numpy as np
        x = np.array([0.9, 0.1, -1])
        y = np.array([0, 1])
        lr = 0.01
        relu = lambda x: max(x, 0)
```

The weights w_1, w_2 are initialized as follows:

```
In [2]: w_1 = np.array([
            [0.3, 0.9, 1, 0.4],
            [0.6, 0.8, -0.3, -0.6],
            [-1.0, 0.1, -0.4, -0.2]
        ])
        w_2 = np.array([
            [0.3, 0.8, 0.2, 0],
            [-0.1, 0.0, -0.6, 0.1]
        ])
```

1 Forward propagation

1.1 First layer

The first layer's input would be ip_2. Multiply $w_1 \times ip_2$, sum and apply activation function (ReLU) to each row:

```
In [3]: ip_2 = np.insert(x, 0, [1])
        a_2 = np.array([relu(sum(row)) for row in np.multiply(w_1, ip_2)])
```

Therefore, values of $a_1^{(2)}, a_2^{(2)}, a_3^{(2)}$ would be:

```
In [4]: print(a_2)
```

```
[0.81 1.89 0.  ]
```

1.2 Second layer

The second layer's input would be ip_3. Multiply $w_2 \times ip_3$, sum and apply activation function (ReLU) to each row:

```
In [5]: ip_3 = np.insert(a_2, 0, [1])
        a_3 = np.array([relu(sum(row)) for row in np.multiply(w_2, ip_3)])
```

Therefore, values of $a_1^{(3)}, a_2^{(3)}$ would be:

```
In [6]: print(a_3)
```

```
[1.326 0.   ]
```

1.3 Lost function

For the lost function $J(\theta)$:

```
In [7]: j = lambda a3_y: 0.5 * ((a3_y[0] - a3_y[1]) ** 2)
        j_theta = sum(map(j, np.column_stack((a_3, y))))
        print(j_theta)
```

```
1.3791380000000004
```

2 Backpropagation

2.1 Calculate gradients with respect to w_2 which is from the second layer. E_{total} would be same as $J(\theta)$:

```
In [8]: sum_b = lambda i: sum(np.multiply(w_2[i], ip_3))

        d_et_a3i = lambda i: a_3[i] - y[i]
        d_a3i_sumb = lambda i: 1 if sum_b(i) >= 0 else 0
        d_sumb_w2ij = lambda i, j: ip_3[j]

        d_et_w2ij = lambda i, j: d_et_a3i(i) * d_a3i_sumb(i) * d_sumb_w2ij(i, j)
```

For $w_2[0]$ case, the gradients are:

```
In [9]: arr_0 = np.array([d_et_w2ij(0, j) for j in range(ip_3.shape[0])])
        print(arr_0)
```

```
[1.326   1.07406 2.50614 0.     ]
```

Which the new weights for $w_2[0]$ will be:

```
In [10]: nw_2_0 = np.array([w_2[0][j] - (lr * arr_0[j]) for j in range(arr_0.shape[0])])
         print(nw_2_0)
```

```
[0.28674   0.7892594 0.1749386 0.       ]
```

For $w_2[1]$ case, the gradients are:

```
In [11]: arr_1 = np.array([d_et_w2ij(1, j) for j in range(ip_3.shape[0])])
         print(arr_1)
```

```
[-0. -0. -0. -0.]
```

Which the new weights for $w_2[1]$ will be:

```
In [12]: nw_2_1 = np.array([w_2[1][j] - (lr * arr_1[j]) for j in range(arr_1.shape[0])])
         print(nw_2_1)
```

```
[-0.1  0.  -0.6  0.1]
```

2.2 Calculate gradients with respect to w_1 which is from the first layer. $E_{\text{total}} = E_{\text{out1}} + E_{\text{out2}}$:

```
In [13]: sum_a = lambda i: sum(np.multiply(w_1[i], ip_2))

         d_eo1_a2j = lambda j: a_3[0] * 1 * w_2[0][j]
         d_eo2_a2j = lambda j: a_3[1] * 1 * w_2[1][j]
         d_et_a2j = lambda j: d_eo1_a2j(j) + d_eo2_a2j(j)
         d_a2i_suma = lambda i: 1 if sum_a(i) >= 0 else 0
         d_suma_w1ij = lambda i, j: ip_2[j]

         d_et_w1ij = lambda i, j: d_et_a2j(j) * d_a2i_suma(i) * d_suma_w1ij(i, j)
```

For $w_1[0]$ case, the gradients are:

```
In [14]: arr_0 = np.array([d_et_w1ij(0, j) for j in range(ip_2.shape[0])])
         print(arr_0)
```

```
[ 0.3978   0.95472  0.02652 -0.     ]
```

Which the new weights for $w_1[0]$ will be:

```
In [15]: nw_1_0 = np.array([w_1[0][j] - (lr * arr_0[j]) for j in range(arr_0.shape[0])])
         print(nw_1_0)
```

```
[0.296022  0.8904528 0.9997348 0.4      ]
```

For $w_1[1]$ case, the gradients are:

```
In [16]: arr_1 = np.array([d_et_w1ij(1, j) for j in range(ip_2.shape[0])])
         print(arr_1)
```

```
[ 0.3978   0.95472  0.02652 -0.     ]
```

Which the new weights for $w_1[1]$ will be:

```
In [17]: nw_1_1 = np.array([w_1[1][j] - (lr * arr_1[j]) for j in range(arr_1.shape[0])])
         print(nw_1_1)
```

```
[ 0.596022   0.7904528 -0.3002652 -0.6      ]
```

For $w_1[2]$ case, the gradients are:

```
In [18]: arr_2 = np.array([d_et_w1ij(2, j) for j in range(ip_2.shape[0])])
         print(arr_2)
```

```
[ 0.  0.  0. -0.]
```

Which the new weights for $w_1[2]$ will be:

```
In [19]: nw_1_2 = np.array([w_1[2][j] - (lr * arr_2[j]) for j in range(arr_2.shape[0])])
         print(nw_1_2)
```

```
[-1.   0.1 -0.4 -0.2]
```

Therefore, w_1 ($\theta^{(1)}$) after first iteration will be:

```
In [20]: print(np.array([nw_1_0, nw_1_1, nw_1_2]))
```

```
[[ 0.296022   0.8904528  0.9997348  0.4      ]
 [ 0.596022   0.7904528 -0.3002652 -0.6      ]
 [-1.         0.1       -0.4       -0.2      ]]
```

And w_2 ($\theta^{(2)}$) after first iteration will be:

```
In [21]: print(np.array([nw_2_0, nw_2_1]))
```

```
[[ 0.28674    0.7892594  0.1749386  0.       ]
 [-0.1        0.        -0.6        0.1      ]]
```