Project 3 – Directions and Sample Output

- Fill out the notebook like normal.
    - The only concept you might not be familiar with is classes, but you should be able to figure it out. Within the NeuralNet class, you can access instance variables with the "self" prefix. The __init__ function is the constructor, that is called whenever you use syntax like:

      nn1 = NeuralNet([2, 1])

      This indicates you are calling the __init__ constructor, passing the the list [2,1] as the sizes parameter.

    - Unlike Java, that has access specifiers like public and private, Python does not use these keywords. You can access the self.whatever variables outside of the class by using the name of the object instead of "self." So for instance, you can access the "Z" variable inside the class by using "self.Z" and outside of the class by using "nn1.Z" (or whatever the variable is called.

- The end of the notebook asks you to repeat the neural network experiment with a second hidden layer (a 3-layer network). Make sure you complete all the steps to train this.

Sample Output/Sanity Checks

```
Forward prop, predict

dim of W1 is (2, 1)
dim of b1 is (1, 1)
[[ 3.58877839]
 [-3.63440561]
 [ 4.54259045]]

[[0.97311094]
 [0.02572061]
 [0.98946635]]

0.9731109351054051

Backprop

dim of W1 is (2, 1)
dim of b1 is (1, 1)
dim of Z1 is (3, 1)
dim of A1 is (3, 1)
[[-0.02688906]
 [-0.97427939]
 [-0.01053365]]

[[ 0.86176106]
 [-0.14478146]]

[-0.33723404]
```

Compute cost

```
dim of W1 is (2, 1)
dim of b1 is (1, 1)
dim of Z1 is (3, 1)
dim of A1 is (3, 1)
dim of delta1 is (3, 1)
dim of deriv_W1 is (2, 1)
dim of deriv_b1 is (1,)
np.float64(2.1593668344765207)
```

For gradient descent, try to get to a cost less than 0.7

2 layer using sizes of [2,4,1]

Forward prop, predict

```
dim of W1 is (2, 3)
dim of b1 is (1, 3)
dim of W2 is (3, 1)
dim of b2 is (1, 1)
[None, array([[-0.3764814 , -3.35914354,  3.90971267],
       [-2.71848575, -0.28604774, -3.27811532],
       [ 3.09853382, -0.70916775,  2.79627088]]), array([[6.44682928],
       [0.76103773],
       [6.09983372]])]

[array([[ 1.96469186, -2.13860665],
       [-2.73148546,  0.51314769],
       [ 2.1946897 , -0.7689354 ]]), array([[0.        , 0.        , 3.90971267],
       [0.        , 0.        , 0.        ],
       [3.09853382, 0.        , 2.79627088]]), array([[0.99841697],
       [0.68157899],
       [0.99776178]])]

0.9984169684439286
```

Backprop

```
dim of W1 is (2, 3)
dim of b1 is (1, 3)
dim of W2 is (3, 1)
dim of b2 is (1, 1)
dim of Z1 is (3, 3)
dim of A1 is (3, 3)
dim of Z2 is (3, 1)
dim of A2 is (3, 1)
[None, array([[-0.        , -0.        , -0.00230216],
       [-0.        , -0.        , -0.        ],
       [-0.00091901, -0.        , -0.00325498]]), array([[-0.00158303],
       [-0.31842101],
       [-0.00223822]])]

[None, array([[-0.00067231,  0.        , -0.00388891],
       [ 0.00023555,  0.        ,  0.00247543]]), array([[-0.00231173],
       [ 0.        ],
       [-0.00414929]])]

[None, array([-0.00030634,  0.        , -0.00185238]), array([-0.10741409])]
```

Compute cost

```
dim of W1 is (2, 3)
dim of b1 is (1, 3)
dim of W2 is (3, 1)
dim of b2 is (1, 1)
dim of Z1 is (3, 3)
dim of A1 is (3, 3)
dim of Z2 is (3, 1)
dim of A2 is (3, 1)
dim of delta1 is (3, 3)
dim of deriv_W1 is (2, 3)
dim of deriv_b1 is (3,)
dim of delta2 is (3, 1)
dim of deriv_W2 is (3, 1)
dim of deriv_b2 is (1,)
np.float64(2.9689832054980694)
```

For gradient descent, try to get to a cost of less than 0.07