M7-L1 Problem 1

In this problem, you will implement a perceptron function that can take in multiple inputs at once as a matrix and output the result of multiplying by a weight matrix and adding a bias vector. Then you will use this function in a loop to implement a multilayer perceptron.

```
import numpy as np
np.set_printoptions(precision=3)
```

Function: `perceptron_layer()`

Complete the function definition for `perceptron_layer(x, weight, bias)`. Inputs:

- `x`: An $N \times n$ matrix of N inputs, each with n features.
- `weight`: An $m \times n$ weight matrix, to be multiplied by the input `x`
- `bias`: A 1-D array of m biases, to be added to the m outputs

Return:

- $N \times m$ output a

a can be obtained by multiplying the weight matrix by the inputs, then adding bias. You must figure out how to make the dimensions work out (e.g. by transposing as necessary) to give the correct size result.

A nonlinear activation would be applied after this function in the context of an MLP, so don't include it in the function. A test case is included for you to check for correctness.

```
def perceptron_layer(x, weight, bias):
    a = np.dot(x, weight.T) + bias
    return a

# Example: N = 3, n = 2, m = 4
x = np.array([[1,2],[3,4],[5,6]])
weight = np.array([[-1.5, -3], [0.5, 1], [1, 1.5], [2, -2]])
bias = np.array([3, -2, .5, -1])
a = perceptron_layer(x, weight, bias)
result = np.array(np.array([[ -4.5,   0.5,   4.5,  -3. ],[-13.5,
3.5,   9.5,  -3. ],[-22.5,   6.5,  14.5,  -3. ]]))

print("Your result", a, sep="\n")
print("Correct result:", result, sep="\n")

Your result
[[ -4.5   0.5   4.5  -3. ]
 [-13.5   3.5   9.5  -3. ]
 [-22.5   6.5  14.5  -3. ]]
Correct result:
[[ -4.5   0.5   4.5  -3. ]
```

```
 [-13.5   3.5   9.5  -3. ]
 [-22.5   6.5  14.5  -3. ]]
```

Function: `MLP()`

Now by looping through several perceptron layers, you can create a multilayer perceptron (AKA a Neural Network!). Complete the function below to do this. Inputs:

- `x`: An $N \times n$ matrix of N inputs, each with n features.
- `weights`: A list of weight matrices
- `biases`: A list of bias vectors

Return:

- Result of applying each perceptron layer with activation, to the input one-by-one

Apply sigmoid activation (a sigmoid function is given) on all layers EXCEPT the final layer.

A test case is provided for you to check your function.

```python
def sigmoid(x):
    return 1./(1.+np.exp(-x))

def MLP(x, weights, biases):
    # YOUR CODE GOES HERE
    for i in range(len(weights)-1):
        x = sigmoid(perceptron_layer(x, weights[i], biases[i]))
    # not applying sigmoid to the last layer
    x = perceptron_layer(x, weights[-1], biases[-1])
    return x

# Example
np.random.seed(0)
dims = [2,6,8,3,1]
weights = []
biases = []
for i,_ in enumerate(dims[:-1]):
    weights.append(np.random.standard_normal([dims[i+1],dims[i]]))
    biases.append(np.random.rand(dims[i+1]))
x = np.random.uniform(-10,10,size=[10,2])

result = np.array([[0.029],[0.267],[0.314],[0.027],[0.319],[0.297],
[0.331],[0.343],[0.187],[0.335]])
y = MLP(x, weights, biases)

print("   Your result: ", y.T, ".T",sep="")
print("Correct result: ", result.T, ".T", sep="")
```

```
   Your result: [[0.029 0.267 0.314 0.027 0.319 0.297 0.331 0.343
0.187 0.335]].T
```

```
Correct result: [[0.029 0.267 0.314 0.027 0.319 0.297 0.331 0.343
0.187 0.335]].T
```