Problem 1. Let $\mathbf{z}$ denote a vector $\mathbf{z} = [z_1, z_2, \ldots, z_n]^T$. Let $\mathbf{p} = [p_1, p_2, \ldots, p_n]^T$, such that

$$p_i = \frac{e^{z_i}}{\sum_{i=1}^{n} e^{z_i}}. \tag{1}$$

That is $\mathbf{p}$ is the output when the softmax function is applied to $\mathbf{z}$. Derive the Jacobian matrix

$$\frac{\partial \mathbf{p}}{\partial \mathbf{z}} = \left[\frac{\partial p_j}{\partial z_i}\right]_{i,j=1,1}^{n,n}. \tag{2}$$

Problem 2. Let $\mathbf{z}$ denote a "logit" vector $\mathbf{z} = [z_1, z_2, \ldots, z_n]^T$. Let $\mathbf{p} = [p_1, p_2, \ldots, p_n]^T$, such that

$$p_i = \frac{e^{z_i}}{\sum_{i=1}^{n} e^{z_i}}. \tag{3}$$

Let $\mathbf{y}$ denote a probability vector $\mathbf{y} = [y_1, y_2, \ldots, y_n]^T$ such that $y_i \geq 0, \forall i \in [1, n]$, and $\sum_{i=1}^{n} y_i = 1$. Let J denote the cross entropy between $\mathbf{p}$ and $\mathbf{y}$:

$$J(\mathbf{z}) = -\sum_{i=1}^{n} y_i \log p_i, \tag{4}$$

where log is natural logarithm.

(a) Derive the gradient vector

$$\frac{\partial J}{\partial \mathbf{z}} = \begin{bmatrix} \frac{\partial J}{\partial z_1} \\ \vdots \\ \frac{\partial J}{\partial z_n} \end{bmatrix} \tag{5}$$

Hint: You can either use the Jacobian, or directly take the gradient. If you take the derivative directly, without using the Jacobian matrix, then it is helpful to write $\log(p_i) = z_i - \log\left(\sum_{i=1}^{n} e^{z_i}\right)$.

(b) Write a `Python` class called `crossEntropyLogit` that implements the mapping from $\mathbf{z}$ to J. It should include two methods: `doForward(self, z, y)`, which returns J, and `doBackward(self, y)`, which returns $\frac{\partial J}{\partial \mathbf{z}}$. You can assume that `doBackward` is always called after `forward`. The implementation should assume that $\mathbf{z}$ and $\mathbf{y}$ are matrices, where each column represents one data vector. The code should produce the desired output for the following code snippet:

```
import numpy as np
CE=crossEntropyLogit()
np.random.seed(1)
z=np.random.rand(3,2)
```

```
y=np.eye(3)[:,:2]
J1=CE.doForward(z,y)       # should be 1.04376...
J2=CE.doForward(z+1000,y)  # should be 1.04376...
dz=CE.doBackward(y)

# dz= [[-0.29358105  0.22809874]
       [ 0.13604698 -0.34982719]
       [ 0.15753407  0.12172845]]
```

Hint: Note that adding a constant to all entries of $\mathbf{z}$ does not change $\mathbf{p}$. So one can subtract the maximum of entries of $\mathbf{z}$ from every entry without affecting the result.

Problem 3. Let a neural network be such that it has two neurons in one single layer. The neurons has two common inputs. The model can be described as

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{6}$$

We are given two training points:

(a) When $\mathbf{x} = [1, 0]^T$, $\mathbf{y} = [1, 0]^T$.

(b) When $\mathbf{x} = [0, 1]^T$, $\mathbf{y} = [0, 1]^T$.

Note that $\mathbf{y}$, the output has been one-hot encoded. Let $\mathbf{X}$ and $\mathbf{Y}$ both be the 2×2 identity matrix, denoting the input and output for the training data. Use softmax on $\mathbf{z}$ and use cross-entropy as the cost function, run the forward and backward propagation manually to update $\mathbf{W}$ and $\mathbf{b}$ for two iterations, with the following conditions:

(a) Both initialized to all zeros.

(b) Learning rate is $\eta = 1$.

(c) Use gradient descent.

That is, make two updates of $\mathbf{W}$ and $\mathbf{b}$ manually. You need to show the intermediate steps (values of $\mathbf{z}$, $d\mathbf{z}$, $\mathbf{p}$, $d\mathbf{W}$, $d\mathbf{b}$, etc).

Problem 4. Download the provided programs, `Data.py`, `NeuralNetwork.py`, and `test_MNIST.py`. Fill in the missing code in `NeuralNetwork.py` at places marked by `....`, so that the program `test_MNIST.py` can run correctly. Experiment with doing classification with the MNIST data set, using the following settings:

(a) Single layer, 10 neurons, softmax + cross entropy objective function.

(b) Three layers, [(50, ReLU), (50, ReLU), (10, Linear)], softmax + cross entropy objective function.

(c) Experiment with other neuron settings, with no more than 3 layers, and no more than 150 neurons in total.

Submit your final codes (both `NeuralNetwork.py` and `test_MNIST.py`. Report both training and testing errors for the different settings. To compute training error, you will need to modify the `main()` program code.

`END OF ASSIGNMENT`