HW1: Implement a Neural Network

In this homework, you will implement a single layer neural network from scratch for a binary classification problem, then validate it using a PyTorch implementation. The HW zip folder includes the following:

1. **nn.py**: You need to implement a single layer neural network from scratch. Specifically, you need to implement the forward and backward functions for ReLU, LinearMap, SoftmaxCrossEntropyLoss. Then use these functions to construct a single layer NN SingleLayerMLP. You will train a model with your implementation on the provided dataset. Please report your loss and accuracy plot for the training and testing process. *You must not use the autograd functionality of PyTorch in this HW*.
2. **reference.py**: You need to implement a single layer neural network with PyTorch to validate your previous implementation. You can use whatever modules in PyTorch as you like. Please report your loss and accuracy plot for the training and testing process.
3. **data**: Folder containing the training and testing datasets, primarily consisting of RDKit descriptors and their corresponding Kow/logP labels (reduced to binary 0 or 1).

Mathematical Background

The following image exemplifies the single layer neural network that needs to be implemented. Let's denote $x \in \mathbb{R}^D$ as the input column vector, W and b are the weight and bias terms for the linear transformation, where the superscript denotes the transformation index. There are M classes, so $y \in \mathbb{R}^M$. Assuming there are H hidden neurons, so $W^1 \in \mathbb{R}^{H \times D}$, $b^1 \in \mathbb{R}^H$, $W^2 \in \mathbb{R}^{M \times H}$ and $b^2 \in \mathbb{R}^M$.

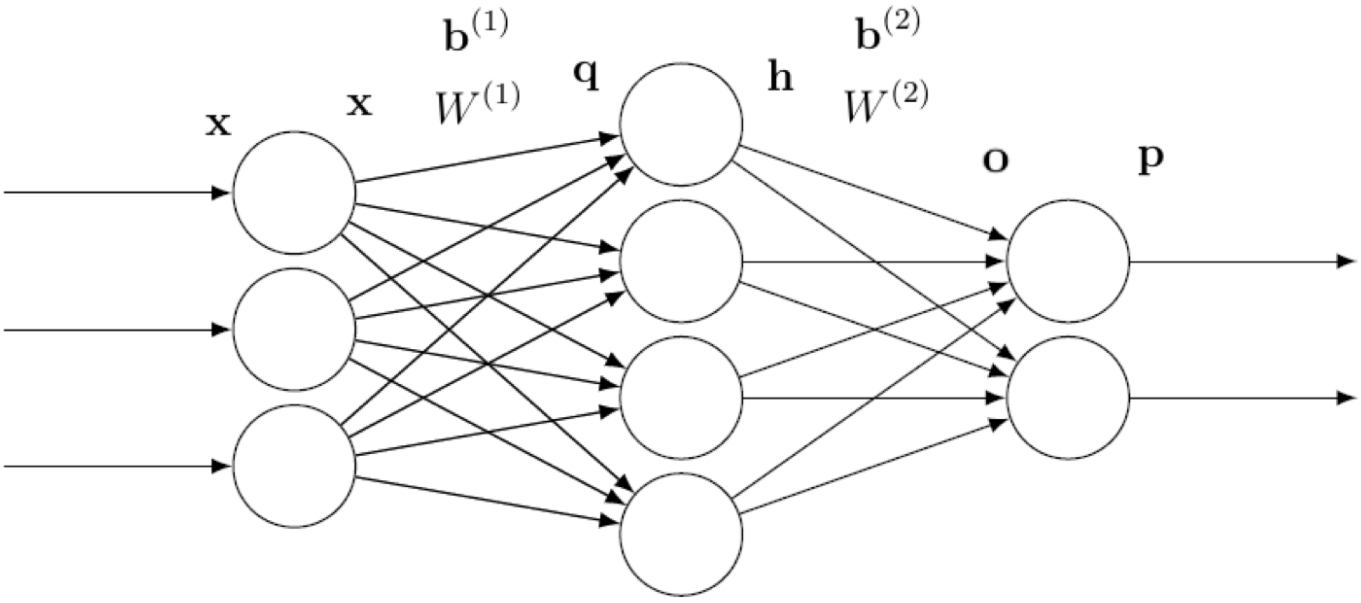

Figure 1: Single layer Multilayer Perceptron (MLP).

For the single layer MLP, the input undergoes the following transformations. Firstly, the column vector x goes through a linear transformation:

$$q = W^1 x + b^1$$

Then non-linearity is added using an element-wise ReLU activation function:

$$h = ReLU(q) = max(0, q)$$

This output is then feeded to the next linear transformation to get the logits o:

$$o = W^2 h + b^2$$

Lastly, applying softmax on the logits yields the probability distribution over the classes p:

$$p = softmax(o)$$

The cross entropy loss function is defined as:

$$L(p, y) = -\sum_{i=1}^{M} y_i log(p_i)$$

where M is the total number of classes.

You need to derive the forward and backward functions for implementing **nn.py**.