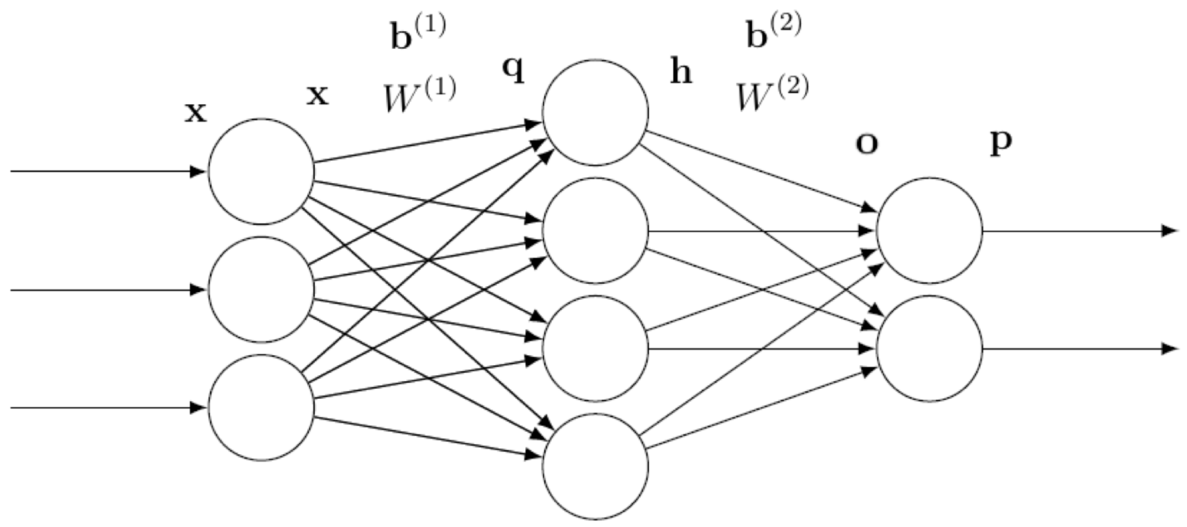
HW1: Implement a Neural Network

In this homework, you will implement a single layer neural network from scratch, then validate it using a PyTorch implementation. The folder include:

- `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 you will 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 cannot use the autograd functionality of PyTorch in this file.
- `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.
- `data` folder contains the training and testing datasets.

Mathematical Background

The following image examplifies the single layer neural network we are going to implement. Let's denote $x \in \mathbb{R}^D$ as the input column vector, b and W are the bias and weights 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 $W^1 \in \mathbb{R}^{H \times D}, b^1 \in \mathbb{R}^H, W^2 \in \mathbb{R}^{M \times H}, b^2 \in \mathbb{R}^M$.



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

$$q = W^1 X + B$$

Then non-linearty is added by using an element-wise ReLU function:

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

The output h is then feed to the next linear transfromation 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`.