Pattern Recognition: Homework 9

Due date: 2023.4.25

In this homework we will focus on multilayer perceptron. You need to first draw the computing grapth, deduce the gradient of each parameter recursively, then fill in the blank of the file `train.py`

Problem 1 (20 pt)

Consider a three layer MLP with ReLU activation $g(x) = \max(x, 0)$. For the ℓ_{th} layer, denote the weight matrix as $\boldsymbol{W}_\ell \in \mathbb{R}^{d_\ell \times d_{\ell+1}}$, bias term as $\boldsymbol{b}_\ell \in \mathbb{R}^{d_{\ell+1}}$, where d_ℓ is the dimensionality of the ℓ_{th} layer. For the forward pass of ℓ_{th} layer, the model first takes the last layer's output $\boldsymbol{a}_\ell$ ($\boldsymbol{a}_0 = \boldsymbol{x}_0$ is the input), then send it through a linear function as

$$\boldsymbol{z}_\ell = \boldsymbol{W}_\ell \boldsymbol{a}_\ell + \boldsymbol{b}_\ell. \quad \ell = 0, 1, 2$$

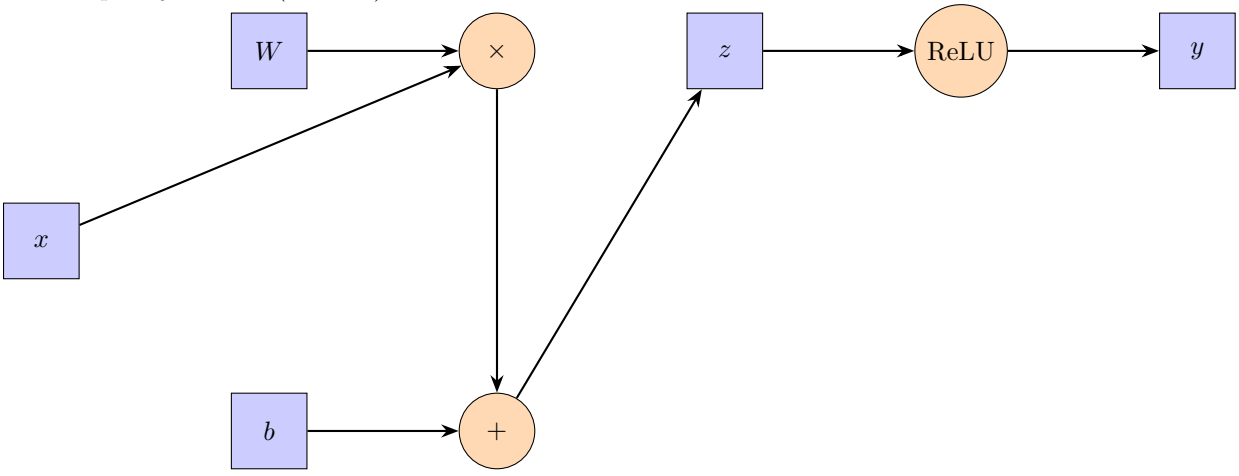
Then apply a non-linear activation g element-wise as

$$\boldsymbol{a}_{\ell+1} = g(\boldsymbol{z}_\ell). \quad \ell = 0, 1$$

Specially, for the last layer $\ell = 2$, we do not need to take activation and directly output $\boldsymbol{a}_2$. In one line, the function parametrized by $\boldsymbol{\theta} = (\boldsymbol{W}_0, \boldsymbol{b}_0, \boldsymbol{W}_1, \boldsymbol{b}_1, \boldsymbol{W}_2, \boldsymbol{b}_2)$ is

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{W}_2 g\left(\boldsymbol{W}_1 g\left(\boldsymbol{W}_0 g\left(\boldsymbol{x}\right) + \boldsymbol{b}_0\right) + \boldsymbol{b}_1\right) + \boldsymbol{b}_2 \tag{1}$$

Give a data point $(\boldsymbol{x}, y)$ and loss function as $J(\boldsymbol{\theta}, \boldsymbol{x}) = \frac{1}{2}\|f(\boldsymbol{x}) - y\|^2$, draw a computing graph of the loss. The graph should contain every parameter $\boldsymbol{w}_\ell, \boldsymbol{b}_\ell$, data $\boldsymbol{x}$ and intermediate output $\boldsymbol{a}_\ell$ as node. Draw an arrow $a \to b$ if computing b depends on a. For example, here is the simplest computational graph of forward pass $y = \mathrm{ReLU}(Wx + b)$



Problem 2 (20 pt)

Based on the computing graph above, recursively derive the gradient $\frac{\partial J}{\partial \boldsymbol{n}}$ for each node $\boldsymbol{n}$ in the graph. Note that given an edge $a \to b$, you can directly use gradient $\frac{\partial J}{\partial b}$ as known in your expression for $\frac{\partial J}{\partial a}$ and do not need to expand it. **hint**: draw it from downstream loss J to upstream $\boldsymbol{x}$.

Problem 3 (60 pt)

Why draw the graph? Because we will use back propagation to compute the gradient for each learnable parameter. You may notice that it is very complicated if we directly compute the gradient $\frac{\partial J}{\partial \boldsymbol{W}_0}$ from equation (1). But we can recursively compute it back from J to $\boldsymbol{W}_0$ with the relationship you derived from problem 2. Fill in the blank in `train.py` to complete the computation of each parameter's gradient, and run the toy regression code to see how neural network fit a function. You should achieve low training and test loss.