Submission for Homework 1: Backpropagation

Jay Nitin Kaoshik
jnk7726@nyu.edu
N13143537

CSCI-GA 2572 Deep Learning

Fall 2024

1 Theory (50pt)

1.1 Two-Layer Neural Nets

We are given the following neural net architecture:

$$\texttt{Linear}_1 \to f \to \texttt{Linear}_2 \to g$$

where $\texttt{Linear}_i(x) = \boldsymbol{W}^{(i)}\boldsymbol{x} + \boldsymbol{b}^{(i)}$ is the i-th affine transformation, and f, g are element-wise nonlinear activation functions. When an input $\boldsymbol{x} \in \mathbb{R}^n$ is fed to the network, $\hat{\boldsymbol{y}} \in \mathbb{R}^K$ is obtained as the output.

1.2 Regression Task

We would like to perform regression task. We choose $f(\cdot) = 5(\cdot)^+ = 5\texttt{ReLU}(\cdot)$ and g to be the identity function. To train this network, we choose MSE loss function $\ell_{\mathrm{MSE}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \|\hat{\boldsymbol{y}} - \boldsymbol{y}\|^2$, where y is the target output.

Answer for 1.2 (a):

(a) (1pt) Name and mathematically describe the 5 programming steps you would take to train this model with `PyTorch` using SGD on a single batch of data.

(1) **Performing Forward Pass** : We pass the Input $\boldsymbol{x}$ through the two-layer neural network, applying the transformations $\hat{\boldsymbol{y}} = g(\texttt{Linear}_2(f(\texttt{Linear}_1(x))))$ to get the model's prediction $\hat{\boldsymbol{y}}$.

(2) **Calculating the Loss** : We then calculate the MSE Loss: $\ell_{\mathrm{MSE}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \|\hat{\boldsymbol{y}} - \boldsymbol{y}\|^2$ where $\boldsymbol{y}$ is the target output.

(3) **Clearing Gradients from Past Iteration** : In PyTorch, we can use `optimizer.zero_grad()` to reset all previously stored gradients to zero, ensuring they don't accumulate from earlier iterations.

(4) **Calculating Gradients for Current Iteration (Backprop)** : Perform backpropagation with `loss.backward()`, which computes the gradients $\frac{\partial \ell}{\partial \boldsymbol{W}}$ and $\frac{\partial \ell}{\partial \boldsymbol{b}}$ for the weights and biases.

(5) **Updating Weights & Biases using Optimizer** : We then perform the optimization step using SGD. We first define our optimizer in the following way: `optimizer = SGD(model.parameters(), learning_rate, momentum)` and then call the step function on the SGD object i.e., `optimizer.step()`

(b) (4pt) For a single data point (x, y), write down all inputs and outputs for forward pass of each layer. You can only use variable $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{W}^{(2)}, \boldsymbol{b}^{(2)}$ in your answer. (note that $\texttt{Linear}_i(\boldsymbol{x}) = \boldsymbol{W}^{(i)}\boldsymbol{x} + \boldsymbol{b}^{(i)}$).

Answer for 1.2 (b):

$\texttt{Linear}_1$ Layer I/O:

- Input: $\boldsymbol{x}$
- Output: $\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}$

f Layer I/O:

- Input: $\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}$
- Output: $5(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})^+$

$\texttt{Linear}_2$ Layer I/O:

- Input: $5(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})^+$
- Output: $\boldsymbol{W}^{(2)}(5(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})^+) + \boldsymbol{b}^{(2)}$

g Layer I/O:

- Input: $\boldsymbol{W}^{(2)}(5(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})^+) + \boldsymbol{b}^{(2)}$
- Output: $\boldsymbol{W}^{(2)}(5(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})^+) + \boldsymbol{b}^{(2)}$

`Loss Function` I/O:

- Input: $\boldsymbol{y}$, $\boldsymbol{W}^{(2)}(5(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})^+) + \boldsymbol{b}^{(2)}$
- Output: $\|\boldsymbol{W}^{(2)}(5(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})^+) + \boldsymbol{b}^{(2)} - \boldsymbol{y}\|^2$

(c) (6pt) Write down the gradients calculated from the backward pass. You can only use the following variables: $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{W}^{(2)}, \boldsymbol{b}^{(2)}, \frac{\partial \ell}{\partial \hat{\boldsymbol{y}}}, \frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{z}_1}, \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z}_3}$ in your answer, where $\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3, \hat{\boldsymbol{y}}$ are the outputs of $\texttt{Linear}_1, f, \texttt{Linear}_2, g$.

Answer for 1.2 (c):

Let's assume that the dimension H refers to the size of the $\texttt{Linear}_1$ layer, which is the number of neurons (or units) that process the input data in this layer. This means that, $\boldsymbol{z}_1, \boldsymbol{z}_2 \in \mathbb{R}^H$. We already know that $\boldsymbol{x} \in \mathbb{R}^n$ and $\hat{\boldsymbol{y}} \in \mathbb{R}^K$. This means that $\boldsymbol{z}_3 \in \mathbb{R}^K$.

We also now know that, $\boldsymbol{W}^{(1)} \in \mathbb{R}^{H \times n}$, $\boldsymbol{b}^{(1)} \in \mathbb{R}^H$, $\boldsymbol{W}^{(2)} \in \mathbb{R}^{K \times H}$ and $\boldsymbol{b}^{(2)} \in \mathbb{R}^K$.

Using Chain Rule,

$$\frac{\partial \ell}{\partial \boldsymbol{z_3}} = \frac{\partial \ell}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}}$$

$$\frac{\partial \ell}{\partial \boldsymbol{z_2}} = \frac{\partial \ell}{\partial \boldsymbol{z_3}} \frac{\partial \boldsymbol{z_3}}{\partial \boldsymbol{z_2}}$$

$$\text{Thus, } \frac{\partial \ell}{\partial \boldsymbol{z_2}} = \frac{\partial \ell}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}} \boldsymbol{W^{(2)}}$$

$$\frac{\partial \ell}{\partial \boldsymbol{z_1}} = \frac{\partial \ell}{\partial \boldsymbol{z_2}} \frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}}$$

$$\text{Thus, } \frac{\partial \ell}{\partial \boldsymbol{z_1}} = \frac{\partial \ell}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}} \boldsymbol{W^{(2)}} \frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}}$$

Now, we Calculate Gradients w.r.t Biases,

$$\frac{\partial \ell}{\partial \boldsymbol{b^{(1)}}} = \frac{\partial \ell}{\partial \boldsymbol{z_1}} \frac{\partial \boldsymbol{z_1}}{\partial \boldsymbol{b^{(1)}}}$$

$$\text{Thus, } \frac{\partial \ell}{\partial \boldsymbol{b^{(1)}}} = \frac{\partial \ell}{\partial \boldsymbol{z}_1}$$

$$\frac{\partial \ell}{\partial \boldsymbol{b^{(2)}}} = \frac{\partial \ell}{\partial \boldsymbol{z_3}} \frac{\partial \boldsymbol{z_3}}{\partial \boldsymbol{b^{(2)}}}$$

$$\text{Thus, } \frac{\partial \ell}{\partial \boldsymbol{b^{(2)}}} = \frac{\partial \ell}{\partial \boldsymbol{z_3}}$$

Calculating Gradients w.r.t Weights, We Have,

$$\frac{\partial \ell}{\partial \boldsymbol{W^{(1)}}} = \frac{\partial \ell}{\partial \boldsymbol{z_1}} \frac{\partial z_1}{\partial \boldsymbol{W^{(1)}}}$$

When we compute the gradient with respect to a matrix, we're essentially calculating how each element of the matrix affects the loss. While $\frac{\partial \ell}{\partial \boldsymbol{z_1}}$ is a vector, $\frac{\partial z_1}{\partial \boldsymbol{W^{(1)}}}$ is a **Tensor** because it represents the change of each element of the output vector z_1 w.r.t. each element of matrix $W^{(1)}$. Since each element of z_1 contributes on a linear combination of the corresponding row of $W^{(1)}$ and the input x, the gradient will involve the input vector x.

$$\text{Thus, } \frac{\partial \ell}{\partial \boldsymbol{W^{(1)}}} = \boldsymbol{x} \frac{\partial \ell}{\partial \boldsymbol{z_1}}$$

$$\text{Similarly, } \frac{\partial \ell}{\partial \boldsymbol{W^{(2)}}} = \frac{\partial \ell}{\partial \boldsymbol{z_3}} \frac{\partial z_3}{\partial \boldsymbol{W^{(2)}}}$$

$$\text{And, } \frac{\partial \ell}{\partial \boldsymbol{W^{(2)}}} = \boldsymbol{z_2} \frac{\partial \ell}{\partial \boldsymbol{z_3}}$$

Final Answer for 1.2 (c):

$$\frac{\partial \ell}{\partial \boldsymbol{W^{(1)}}} = \boldsymbol{x} \frac{\partial \ell}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}} \boldsymbol{W^{(2)}} \frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}} \text{ and } \frac{\partial \ell}{\partial \boldsymbol{b^{(1)}}} = \frac{\partial \ell}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}} \boldsymbol{W^{(2)}} \frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}}$$

$$\frac{\partial \ell}{\partial \boldsymbol{W^{(2)}}} = 5(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})^{+} \frac{\partial \ell}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}} \text{ and } \frac{\partial \ell}{\partial \boldsymbol{b^{(2)}}} = \frac{\partial \ell}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}}$$

Dimensions for the Parameter Gradients:

$$\frac{\partial \ell}{\partial \boldsymbol{W}^{(1)}} \in \mathbb{R}^{n\times H},\ \frac{\partial \ell}{\partial \boldsymbol{b}^{(1)}} \in \mathbb{R}^{1\times H},\ \frac{\partial \ell}{\partial \boldsymbol{W}^{(2)}} \in \mathbb{R}^{H\times K},\ \frac{\partial \ell}{\partial \boldsymbol{b}^{(2)}} \in \mathbb{R}^{1\times K}$$

(d) (2pt) Show us the elements of $\frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}}$, $\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}}$ and $\frac{\partial \ell}{\partial \hat{\boldsymbol{y}}}$ (be careful about the dimensionality)?

Answer for 1.2 (d):

Dimensions for Question Gradients:

$$\frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}} \in \mathbb{R}^{H\times H},\ \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}} \in \mathbb{R}^{K\times K},\ \frac{\partial \ell}{\partial \hat{\boldsymbol{y}}} \in \mathbb{R}^{1\times K}$$

Types: $\frac{\partial \ell}{\partial \hat{\boldsymbol{y}}}$ is a Row Vector whereas $\frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}}$ and $\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}}$ are Matrices.

Elements of $\frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}}$

$$\left(\frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}}\right)_{ij} = \begin{cases} 5, & \text{if } i = j \text{ and } (z_{1i} > 0) \\ 0, & \text{if } i \neq j \text{ or } (z_{1i} \leq 0) \end{cases}$$

Elements of $\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}}$

$$\left(\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}}\right)_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}$$

Elements of $\frac{\partial \ell}{\partial \hat{\boldsymbol{y}}}$

$$\left(\frac{\partial \ell}{\partial \hat{\boldsymbol{y}}}\right)_i = 2 \times (\hat{y} - y)_i$$

Final Answer for 1.2 (d):

All Elements **NOT** Present on Diagonal of $\frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}}$ and $\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}}$ are 0.

Elements Present on Diagonal of $\frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}}$ where $z_1 > 0$ are 5.

Elements Present on Diagonal of $\frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}}$ where $z_1 \leq 0$ are 0.

Elements Present on Diagonal of $\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}}$ are 1.

1.3 Classification Task

We would like to perform multi-class classification task, so we set $f = \tanh$ and $g = \sigma$, the logistic sigmoid function $\sigma(z) \doteq (1 + \exp(-x))^{-1}$.

(a) (4pt + 6pt + 2pt) If you want to train this network, what do you need to change in the equations of (b), (c) and (d), assuming we are using the same MSE loss function.

Answer for 1.3 (a) - (b):

`Linear`$_1$ Layer I/O:

- Input: $\boldsymbol{x}$
- Output: $\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)}$

f Layer I/O:

- Input: $\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)}$
- Output: $\tanh(\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)})$

`Linear`$_2$ Layer I/O:

- Input: $\tanh(\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)})$
- Output: $\boldsymbol{W}^{(2)}(\tanh(\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)}))+\boldsymbol{b}^{(2)}$

g Layer I/O:

- Input: $\boldsymbol{W}^{(2)}(\tanh(\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)}))+\boldsymbol{b}^{(2)}$
- Output: $\sigma(\boldsymbol{W}^{(2)}(\tanh(\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)}))+\boldsymbol{b}^{(2)})$

`Loss Function` I/O:

- Input: $\boldsymbol{y}$, $\sigma(\boldsymbol{W}^{(2)}(\tanh(\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)}))+\boldsymbol{b}^{(2)})$
- Output: $\|\sigma(\boldsymbol{W}^{(2)}(\tanh(\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)}))+\boldsymbol{b}^{(2)})-\boldsymbol{y}\|^2$

Answer for 1.3 (a) - (c):

We Know,

$$\frac{\partial \ell}{\partial \boldsymbol{z_3}} = \frac{\partial \ell}{\partial \hat{\boldsymbol{y}}}\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}}$$
$$\frac{\partial \ell}{\partial \boldsymbol{z_2}} = \frac{\partial \ell}{\partial \hat{\boldsymbol{y}}}\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}}\boldsymbol{W^{(2)}}$$
$$\frac{\partial \ell}{\partial \boldsymbol{z_1}} = \frac{\partial \ell}{\partial \hat{\boldsymbol{y}}}\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}}\boldsymbol{W^{(2)}}\frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}}$$

Also,

$$\frac{\partial \ell}{\partial \boldsymbol{b^{(1)}}} = \frac{\partial \ell}{\partial \hat{\boldsymbol{y}}}\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}}\boldsymbol{W^{(2)}}\frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}}$$
$$\frac{\partial \ell}{\partial \boldsymbol{b^{(2)}}} = \frac{\partial \ell}{\partial \hat{\boldsymbol{y}}}\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}}$$
$$\frac{\partial \ell}{\partial \mathbf{W^{(1)}}} = \boldsymbol{x}\frac{\partial \ell}{\partial \boldsymbol{z_1}}$$
$$\frac{\partial \ell}{\partial \mathbf{W^{(2)}}} = \boldsymbol{z_2}\frac{\partial \ell}{\partial \boldsymbol{z_3}}$$

Final Answer for 1.3 (a) - (c):

$$\frac{\partial \ell}{\partial \boldsymbol{W}^{(1)}} = \boldsymbol{x} \frac{\partial \ell}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}} \boldsymbol{W^{(2)}} \frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}} \text{ and } \frac{\partial \ell}{\partial \boldsymbol{b}^{(1)}} = \frac{\partial \ell}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}} \boldsymbol{W^{(2)}} \frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}}$$

$$\frac{\partial \ell}{\partial \boldsymbol{W}^{(2)}} = \tanh(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}) \frac{\partial \ell}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}} \text{ and } \frac{\partial \ell}{\partial \boldsymbol{b}^{(2)}} = \frac{\partial \ell}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}}$$

Answer for 1.3 (a) - (d):

Elements of $\frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}}$

$$\left(\frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}}\right)_{ij} = \begin{cases} 1 - \tanh^2((z_1)_i) \textbf{ OR } \operatorname{sech}^2((z_1)_i), & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}$$

Elements of $\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}}$

$$\left(\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z_3}}\right)_{ij} = \begin{cases} \sigma((z_3)_i)(1 - \sigma((z_3)_i)), & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}$$

Elements of $\frac{\partial \ell}{\partial \hat{\boldsymbol{y}}}$

$$\left(\frac{\partial \ell}{\partial \hat{\boldsymbol{y}}}\right)_i = 2 \times (\hat{y} - y)_i$$

(b) (4pt + 6pt + 2pt) Now you think you can do a better job by using a *Binary Cross Entropy* (BCE) loss function $\ell_{\text{BCE}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{K}\sum_{i=1}^{K} -\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]$. What do you need to change in the equations of (b), (c) and (d)?

Answer for 1.3 (b) - (b):

$\texttt{Linear}_1$ Layer I/O:

- Input: $\boldsymbol{x}$
- Output: $\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}$

f Layer I/O:

- Input: $\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}$
- Output: $\tanh(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})$

$\texttt{Linear}_2$ Layer I/O:

- Input: $\tanh(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})$
- Output: $\boldsymbol{W}^{(2)}(\tanh(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})) + \boldsymbol{b}^{(2)}$

g Layer I/O:

- Input: $\boldsymbol{W}^{(2)}(\tanh(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})) + \boldsymbol{b}^{(2)}$
- Output: $\sigma(\boldsymbol{W}^{(2)}(\tanh(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})) + \boldsymbol{b}^{(2)})$

`Loss Function` I/O:

- Input: $\boldsymbol{y}$, $\sigma(\boldsymbol{W}^{(2)}(\tanh(\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)}))+\boldsymbol{b}^{(2)})$
- Output: $-\frac{1}{K}[y^T\log(\sigma(\boldsymbol{W}^{(2)}(\tanh(\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)}))+\boldsymbol{b}^{(2)})+$ $(1-y)^T\log(1-\sigma(\boldsymbol{W}^{(2)}(\tanh(\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)}))+\boldsymbol{b}^{(2)})]$

Answer for 1.3 (b) - (c):

$$\frac{\partial \ell}{\partial \boldsymbol{W^{(1)}}} = \boldsymbol{x}\frac{\partial \ell}{\partial \boldsymbol{\hat{y}}}\frac{\partial \boldsymbol{\hat{y}}}{\partial \boldsymbol{z_3}}\boldsymbol{W^{(2)}}\frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}} \text{ and } \frac{\partial \ell}{\partial \boldsymbol{b^{(1)}}} = \frac{\partial \ell}{\partial \boldsymbol{\hat{y}}}\frac{\partial \boldsymbol{\hat{y}}}{\partial \boldsymbol{z_3}}\boldsymbol{W^{(2)}}\frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}}$$

$$\frac{\partial \ell}{\partial \boldsymbol{W^{(2)}}} = \tanh(\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)})\frac{\partial \ell}{\partial \boldsymbol{\hat{y}}}\frac{\partial \boldsymbol{\hat{y}}}{\partial \boldsymbol{z_3}} \text{ and } \frac{\partial \ell}{\partial \boldsymbol{b^{(2)}}} = \frac{\partial \ell}{\partial \boldsymbol{\hat{y}}}\frac{\partial \boldsymbol{\hat{y}}}{\partial \boldsymbol{z_3}}$$

Answer for 1.3 (b) - (d):

Elements of $\frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}}$

$$(\frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}})_{ij} = \begin{cases} 1-\tanh^2((z_1)_i) \textbf{ OR } \operatorname{sech}^2((z_1)_i), & \text{if } i=j \\ 0, & \text{if } i \neq j \end{cases}$$

Elements of $\frac{\partial \boldsymbol{\hat{y}}}{\partial \boldsymbol{z_3}}$

$$(\frac{\partial \boldsymbol{\hat{y}}}{\partial \boldsymbol{z_3}})_{ij} = \begin{cases} \sigma((z_3)_i)(1-\sigma((z_3)_i)), & \text{if } i=j \\ 0, & \text{if } i \neq j \end{cases}$$

Elements of $\frac{\partial \ell}{\partial \boldsymbol{\hat{y}}}$

$$(\frac{\partial \ell}{\partial \boldsymbol{\hat{y}}})_i = \frac{1}{K}[\frac{\hat{y}-y}{\hat{y}-\hat{y}^2}]_i$$

(c) (1pt) Things are getting better. You realize that not all intermediate hidden activations need to be binary (or soft version of binary). You decide to use $f(\cdot)=(\cdot)^+$ but keep g as tanh. Explain why this choice of f can be beneficial for training a (deeper) network.

Answer for 1.3 (c):

ReLU is **non-saturating**, meaning that for any input, the output isn't squashed in certain bounds. Also, for positive inputs, the derivative of ReLU is constant (1). This helps in avoiding the **vanishing gradient** problem that can occur with other activation functions like sigmoid or tanh, which tend to saturate and produce very small gradients for large positive or negative values. In a deep network, this issue can prevent gradients from flowing back effectively through the layers, making it hard to train. ReLU also allows **faster convergence**, especially in deep networks, because the gradients remain strong for positive values of x.

ReLU produces **sparse activations**, meaning that for any input, a significant portion of the neurons will output 0 (for negative inputs), which reduces the computational load and introduces sparsity in the model.

1.4 Conceptual Questions

(a) (1pt) Can the output of softmax function be exactly 0 or 1? Why or why not?

Answer for 1.4 (a): The output of the softmax function cannot be exactly 0 or 1. We know the formula for softmax function based on the class probabilities z_i as follows:

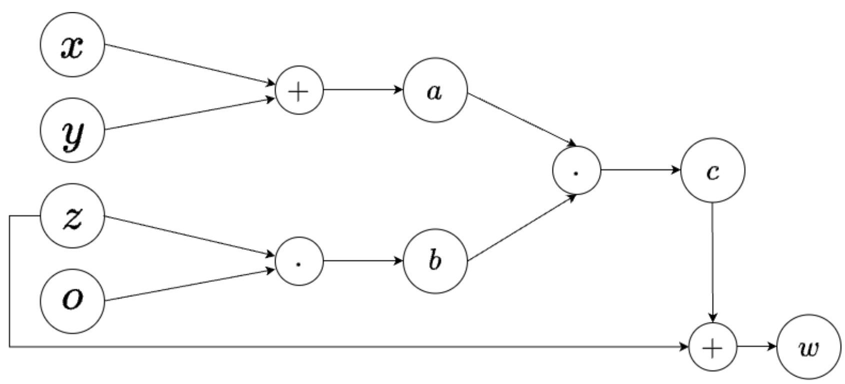
$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

The numerator will always be positive for any real value of z_i making the overall value never exactly 0. The denominator is the sum of all similar positive exponentials, including e^{z_i}. Since the fraction has denominator always greater than the numerator, final value will always be less than 1.

(b) (3pt) Draw the computational graph defined by this function, with inputs $x, y, z \in \mathbb{R}$ and output $w \in \mathbb{R}$. You make use symbols x, y, z, o, and operators $*, +$ in your solution. Be sure to use the correct shape for symbols and operators as shown in class.

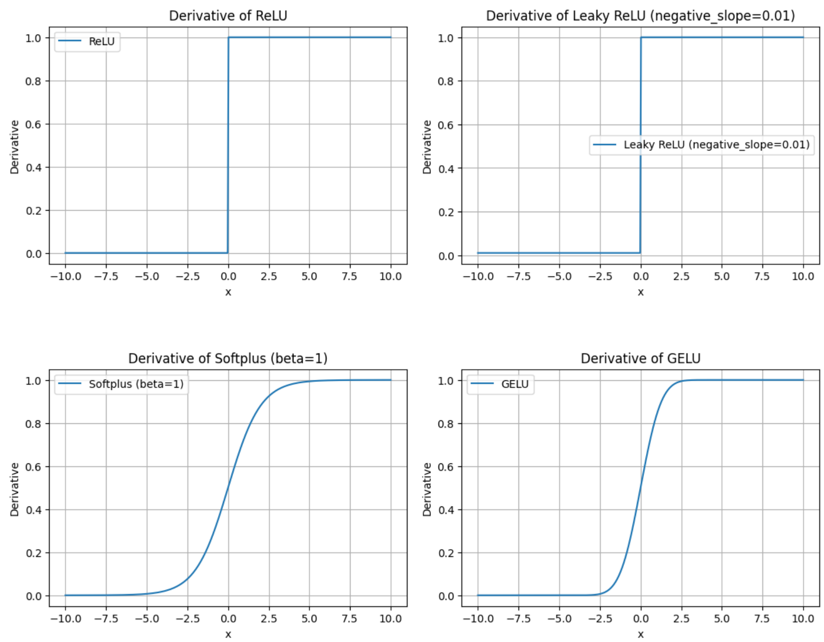
$$\begin{aligned} a &= x + y \\ b &= z * o \\ c &= a * b \\ w &= c + z \end{aligned}$$

Answer for 1.4 (b):



(c) (3pt) Draw the graph of the derivative for the following functions?

- `ReLU()`
- `LeakyReLU(negative_slope=0.01)`
- `Softplus(beta=1)`
- `GELU()`



Answer for 1.4 (c):

(d) (3pt) Explain what are the limitations of the ReLU activation function. How do leaky ReLU and softplus address some of these problems?

Answer for 1.4 (d): The following are some limitations of the ReLU (Rectified Linear Unit) activation function:

- For negative input, ReLU outputs zero. During training this will result in some neurons which receive negative values, to stop updating since their activation values will be 0 consistently in every update cycle as their gradients will become zero. This will cause parts of the network to become inactive.
- For positive input, ReLU can produce large values at times, which may lead to exploding gradients and as a result unstable training.
- Also it is not differentiable at zero, and this could lead to issues with optimization in some cases.

Leaky ReLU and Softplus address some of these limitations as follows:

- **Leaky ReLU:** Leaky ReLU allows a small, non-zero gradient for negative input values. Instead of outputting 0 for negative inputs, it outputs a linearly scaled value (e.g., αz, where α is a small constant). This helps prevent the first problem mentioned above seen with ReLU.

- **Softplus:** Softplus is a smooth approximation of the ReLU function, defined as $\text{Softplus}(z) = \log(1 + e^z)$. Unlike ReLU, Softplus is differentiable everywhere and never gives exactly 0 gradients. It has a smoother curve which reduces abrupt changes in the output, as a result addressing issues related to gradient stability as well.

(e) (2pt) What are 4 different types of linear transformations? What is the role of linear transformation and non-linear transformation in a neural network?

Answer for 1.4 (e): Four Types of **Linear Transformations** include:

- **Scaling:** Multiplying a vector by a scalar, which stretches or shrinks it along its direction.
- **Rotation:** Rotating a vector around an origin by a specific angle without changing its length.
- **Translation:** Shifting a vector by a fixed amount, though this is strictly an affine transformation and not purely linear.
- **Shearing:** Skewing a vector such that it shifts in one direction, while preserving parallelism of lines in the transformation.

Role of linear and non-linear transformations in a neural network:

- **Linear transformations:** Linear transformations form the fundamental operations in a neuron where inputs are multiplied by the weights and added to biases. These operations project the input space to a new space essential for better learning of the underlying pattern behind the input-output set. However, linear transformations alone are insufficient, since a composition of linear transformations results in another linear transformation, limiting the expressiveness.
- **Non-linear transformations:** Non-linear transformations, introduced by activation functions like ReLU, Sigmoid, or Tanh, allow the network to learn complex patterns. They are essential for capturing hierarchical, multi-layered feature representations and for making DNNs.