Homework 1: Backpropagation

The goal of homework 1 is to help you understand how to update network parameters by the using backpropagation algorithm.

For part 1, you need to answer the questions with mathematics equations. You should put all your answers in a PDF file and we will not accept any scanned hand-written answers. It is recommended to use LaTeX.

For part 2, you need to program with Python. It requires you to implement your own forward and backward pass without using autograd. You need to submit your `mlp.py` file for this part.

The due date of homework 1 is `23:55` of `02/25`. Submit the following files in a zip file `your_net_id.zip` through NYU classes:

- `theory.pdf`
- `mlp.py`

The following behaviors will result in penalty of your final score:

1. 5% penalty for submitting your files without using the correct format. (including naming the zip file, PDF file or python file wrong, or adding extra files in the zip folder, like the testing scripts from part 2).
2. 20% penalty for late submission within the first 24 hours. We will not accept any late submission after the first 24 hours.
3. 20% penalty for code submission that cannot be executed using the steps we mentioned in part 2. So please test your code before submit it.

1 Theory (50pt)

To answer questions in this part, you need some basic knowledge of linear algebra and matrix calculus. Also, you need to follow the instructions:

1. Every vector is treated as column vector.
2. You need to use the numerator-layout notation for matrix calculus. Please refer to Wikipedia about the notation. *
3. You are only allowed to use vector and matrix. You cannot use tensor in any of your answer.
4. Missing transpose are considered as wrong answer.

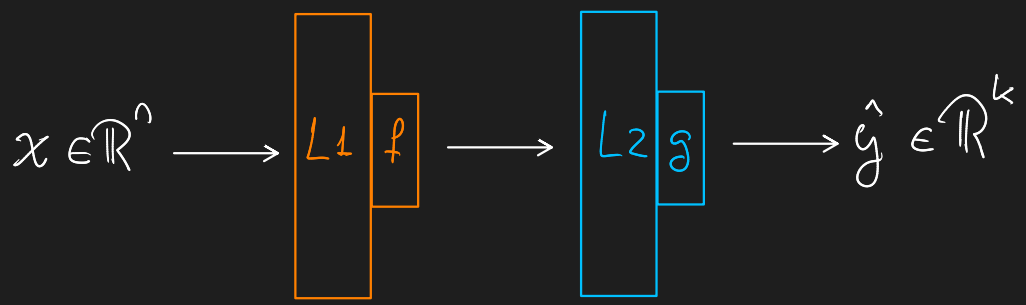
* https://en.wikipedia.org/wiki/Matrix_calculus#Numerator-layout_notation

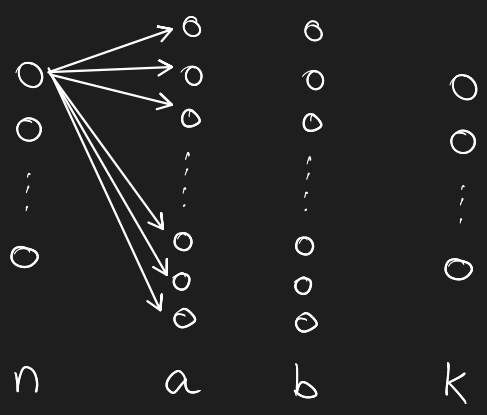
1.1 Two-Layer Neural Nets

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

(a) 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. Feed forward pass
2. Compute Loss (or free energy)
3. Delete/Zero past gradients
4. Backward pass: Compute and accumulate gradients
5. Update parameters

```
# Full-batch training loop
for t in range(1_000):
    # Feed forward to get the logits
    I = model(X)

    # Compute the free energy F
    F = C(I, y)
    L = F.mean()

    # Zero the gradients
    optimiser.zero_grad()

    # Backward pass to compute and accumulate the gradient
    # of the free energy w.r.t our learnable params
    L.backward()

    # Update params
    optimiser.step()

    # Display epoch, L, and accuracy
    overwrite(f'[EPOCH]: {t}, [LOSS]: {L.item():.6f}, [ACCURACY]: {acc(I, y):.3f}')
```

$$\hat{y} := g(L_2(f(L_1(x))))$$

$$\ell := \| \hat{y} - y \|$$

(b) 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 $\texttt{Linear}_i(\boldsymbol{x}) = \boldsymbol{W}^{(i)}\boldsymbol{x} + \boldsymbol{b}^{(i)}$).

Layer	Input	Output
$\texttt{Linear}_1$		
f		
$\texttt{Linear}_2$		
g		
$\texttt{Loss}$		

(c) Write down the gradient 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 $\texttt{Linear}_1, f, \texttt{Linear}_2, g$.

Parameter	Gradient
$\boldsymbol{W}^{(1)}$	
$\boldsymbol{b}^{(1)}$	
$\boldsymbol{W}^{(2)}$	
$\boldsymbol{b}^{(2)}$	

(d) 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)?

1.3 Classification Task

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

(a) 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.

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

(c) 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 σ. Explain why this choice of f can be beneficial for training a (deeper) network.