Homework 1: Backpropagation

CSCI-GA 2572 Deep Learning

Fall 2024

The goal of homework 1 is to help you understand the common techniques used in Deep Learning and how to update network parameters by the using backpropagation algorithm.

Part 1 has two sub-parts, 1.1, 1.2, 1.3 majorly deal with the theory of backpropagation algorithm whereas 1.4 is to test conceptual knowledge on deep learning. For part 1.2 and 1.3, you need to answer the questions with mathematical 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 in 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 EST` of `09/22`. Submit the following files in a zip file `your_net_id.zip` through Gradescope (Gradescope can be either accessed through Brightspace course page –> Content or you should've received an email from Gradescope for log-in instruction):

- `theory.pdf`
- `mlp.py`
- `gd.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 provided vector is treated as column vector.
2. IMPORTANT: You need to use the numerator-layout notation for matrix calculus. Please refer to Wikipedia about the notation. Specifically, $\frac{\partial y}{\partial \mathbf{x}}$ is a row-vector whereas $\frac{\partial \mathbf{y}}{\partial x}$ is a column-vector
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.

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 $f(\cdot) = 5(\cdot)^+ = 5\texttt{ReLU}(\cdot)$ 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) (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.

(a) First compute a prediction from the model (the forward pass); $\tilde{y} = Predictor(x)$

(b) Second, compute the loss through computation of the energy

(c) Zero the gradient parameters (choosing no gradient accumulation from previous epoch) with $optimiser.zero\_grad()$

(d) Compute and accumulate gradient parameters; $\mathcal{L}.backward()$

(e) Step in the opposite direction of the gradient; $optimiser.step()$

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

Layer	**Input**	**Output**
$Linear_1$	$\boldsymbol{x}$	$\boldsymbol{z_1} = \boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}$
f	$\boldsymbol{z_1}$	$\boldsymbol{z_2} = 5(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})^+$
$Linear_2$	$\boldsymbol{z_2}$	$\boldsymbol{z_3} = 5\boldsymbol{W}^{(2)}(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})^+ + \boldsymbol{b}^{(2)}$
g	$\boldsymbol{z_3}$	$\boldsymbol{z_3}$ **(*or* $\hat{\boldsymbol{y}}$)**
$Loss$	$\hat{\boldsymbol{y}}, \boldsymbol{y}$	$\lvert\lvert\hat{\boldsymbol{y}} - \boldsymbol{y}\rvert\rvert^2 = \lvert\lvert 5\boldsymbol{W}^{(2)}(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})^+ + \boldsymbol{b}^{(2)} - \boldsymbol{y}\rvert\rvert^2$

(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 $\texttt{Linear}_1, f, \texttt{Linear}_2, g$.

Dimensions shown are used throughout ($z_1, z_2 \in \mathbb{R}^h$):

$$x \in \mathbb{R}^n$$

$$\hat{y} \in \mathbb{R}^K$$

$$\ell \in \mathbb{R}$$

$$W^{(1)} \in \mathbb{R}^{h \times n}$$

$$b^{(1)} \in \mathbb{R}^h$$

$$W^{(2)} \in \mathbb{R}^{K \times h}$$

$$b^{(2)} \in \mathbb{R}^K$$

$$\frac{\partial \ell}{\partial W^{(1)}} \in \mathbb{R}^{n \times h}$$

$$\frac{\partial \ell}{\partial b^{(1)}} \in \mathbb{R}^{1 \times h}$$

$$\frac{\partial \ell}{\partial W^{(2)}} \in \mathbb{R}^{h \times K}$$

$$\frac{\partial \ell}{\partial b^{(2)}} \in \mathbb{R}^{1 \times K}$$

$$\frac{\partial \ell}{\partial \hat{y}} \in \mathbb{R}^{1 \times K}$$

$$\frac{\partial \hat{y}}{\partial z_3} = I \in \mathbb{R}^{K \times K}$$

$$\frac{\partial z_3}{\partial z_2} \in \mathbb{R}^{K \times h}$$

$$\frac{\partial z_2}{\partial z_1} \in \mathbb{R}^{h \times h}$$

$$\frac{\partial z_1}{\partial b_1} \in \mathbb{R}^{h \times h}$$

$$\frac{\partial z_1}{\partial W^{(1)}} \in \mathbb{R}^{h \times (h \times n)}$$

$$\frac{\partial z_3}{\partial b^{(2)}} \in \mathbb{R}^{K \times K}$$

$$\frac{\partial z_3}{\partial W^{(2)}} \in \mathbb{R}^{K \times (K \times h)}$$

$$
\begin{aligned}
\frac{\partial \ell}{\partial z_3} &= \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \\
\frac{\partial \ell}{\partial z_1} &= \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial z_2} \frac{\partial z_2}{\partial z_1} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} W^{(2)} \frac{\partial z_2}{\partial z_1} \\
\frac{\partial \boldsymbol{\ell}}{\partial \boldsymbol{b^{(2)}}} &= \frac{\partial \boldsymbol{\ell}}{\partial \boldsymbol{\hat{y}}} \frac{\partial \boldsymbol{\hat{y}}}{\partial \boldsymbol{z_3}} \frac{\partial \boldsymbol{z_3}}{\partial \boldsymbol{b^{(2)}}} \\
&= \frac{\partial \boldsymbol{\ell}}{\partial \boldsymbol{\hat{y}}} \frac{\partial \boldsymbol{\hat{y}}}{\partial \boldsymbol{z_3}} \\
\frac{\partial \ell}{\partial b^{(1)}} &= \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} W^{(2)} \frac{\partial z_2}{\partial z_1} \\
\frac{\partial \ell}{\partial W^{(1)}} = \frac{\partial \ell}{\partial z_1} \frac{\partial z_1}{\partial W^{(1)}} &= \sum_i \frac{\partial \ell}{\partial (z_1)_i} \frac{\partial (z_1)_i}{\partial W^{(1)}} = \frac{\partial \ell}{\partial z_1} x^T = x \frac{\partial \ell}{\partial z_1} \\
&= x \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} W^{(2)} \frac{\partial z_2}{\partial z_1} \\
\frac{\partial \ell}{\partial W^{(2)}} = \frac{\partial \ell}{\partial z_3} \frac{\partial z_3}{\partial W^{(2)}} &= \sum_i \frac{\partial \ell}{\partial (z_3)_i} \frac{\partial (z_3)_i}{\partial W^{(2)}} = \frac{\partial \ell}{\partial z_3} {z_2}^T = z_2 \frac{\partial \ell}{\partial z_3} \\
&= 5(\boldsymbol{W}^{(1)} \boldsymbol{x} + \boldsymbol{b}^{(1)})^+ \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3}
\end{aligned}
$$

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

$$
\left(\frac{\partial \boldsymbol{z_2}}{\partial \boldsymbol{z_1}}\right)_{ii} = \begin{cases} 0, & z_{1i} < 0 \\ 5, & z_{1i} > 0 \\ \text{undefined (or assigned a value 0)}, & z_{1i} = 0 \end{cases}
$$

Note: the above is a diagonal matrix

$$
\left(\frac{\partial \boldsymbol{\hat{y}}}{\partial \boldsymbol{z_3}}\right)_{ii} = 1 \text{ (and 0 elsewhere, off of diagonal; an identity matrix)}
$$

$$
\frac{\partial \ell}{\partial \boldsymbol{\hat{y}}} = \frac{\partial (||\boldsymbol{\hat{y}} - \boldsymbol{y}||^{\boldsymbol{2}})}{\partial \boldsymbol{\hat{y}}} = 2(\boldsymbol{\hat{y}} - \boldsymbol{y})^{\boldsymbol{T}} \text{ (A vector)}
$$

$$
\implies 2(\hat{y} - y)_i \text{ are the elements}
$$

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.

Layer	Input	Output
$Linear_1$	$\boldsymbol{x}$	$\boldsymbol{z_1} = \boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}$
f	$\boldsymbol{z_1}$	$\boldsymbol{z_2} = \tanh(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})$
$Linear_2$	$\boldsymbol{z_2}$	$\boldsymbol{z_3} = \boldsymbol{W}^{(2)}\tanh(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}) + \boldsymbol{b}^{(2)}$
g	$\boldsymbol{z_3}$	$\sigma(z_3) = \frac{\mathbf{1}}{\mathbf{1+exp(-\boldsymbol{z_3})}}$
$Loss$	$\boldsymbol{\hat{y}} = \boldsymbol{\sigma(z_3), y}$	$\vert\vert\boldsymbol{\hat{y}} - \boldsymbol{y}\vert\vert^2 = \vert\vert\boldsymbol{\sigma(z_3)} - \boldsymbol{y}\vert\vert^2$

$$
\begin{aligned}
\frac{\partial \ell}{\partial z_3} &= \frac{\partial \ell}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_3} \\
\frac{\partial \ell}{\partial z_1} &= \frac{\partial \ell}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_3}\frac{\partial z_3}{\partial z_2}\frac{\partial z_2}{\partial z_1} = \frac{\partial \ell}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_3}W^{(2)}\frac{\partial z_2}{\partial z_1} \\
\frac{\partial \boldsymbol{\ell}}{\partial \boldsymbol{b}^{(2)}} &= \frac{\partial \boldsymbol{\ell}}{\partial \boldsymbol{\hat{y}}}\frac{\partial \boldsymbol{\hat{y}}}{\partial \boldsymbol{z_3}}\frac{\partial \boldsymbol{z_3}}{\partial \boldsymbol{b}^{(2)}} \\
&= \frac{\partial \boldsymbol{\ell}}{\partial \boldsymbol{\hat{y}}}\frac{\partial \boldsymbol{\hat{y}}}{\partial \boldsymbol{z_3}} \\
\frac{\partial \ell}{\partial b^{(1)}} &= \frac{\partial \ell}{\partial z_1}\frac{\partial z_1}{\partial b^{(1)}} = \frac{\partial \ell}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_3}W^{(2)}\frac{\partial z_2}{\partial z_1} \\
\frac{\partial \ell}{\partial W^{(1)}} &= \frac{\partial \ell}{\partial z_1}\frac{\partial z_1}{\partial W^{(1)}} = \sum_i \frac{\partial \ell}{\partial (z_1)_i}\frac{\partial (z_1)_i}{\partial W^{(1)}} = \frac{\partial \ell}{\partial z_1}x^T \\
&= x\frac{\partial \ell}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_3}W^{(2)}\frac{\partial z_2}{\partial z_1} \\
\frac{\partial \ell}{\partial W^{(2)}} &= \frac{\partial \ell}{\partial z_3}\frac{\partial z_3}{\partial W^{(2)}} = \sum_i \frac{\partial \ell}{\partial (z_3)_i}\frac{\partial (z_3)_i}{\partial W^{(2)}} = \frac{\partial \ell}{\partial z_3}{z_2}^T \\
&= \tanh(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})\frac{\partial \ell}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_3} \\
\left(\frac{\partial z_2}{\partial z_1}\right)_{ii} &= [(\text{sech}(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}))]^2_{1i} \quad \text{and off diagonal elements are 0} \\
\left(\frac{\partial \hat{y}}{\partial z_3}\right)_{ii} &= \sigma((z_3)_i)(1 - \sigma(z_3)_i) \quad \text{and off diagonal elements are 0} \\
\frac{\partial \ell}{\partial \boldsymbol{\hat{y}}} &= \frac{\partial(\vert\vert\boldsymbol{\hat{y}} - \boldsymbol{y}\vert\vert^2)}{\partial \boldsymbol{\hat{y}}} = 2(\boldsymbol{\hat{y}} - \boldsymbol{y})^T \\
&\implies 2(\hat{y} - y)_i \text{ are the elements}
\end{aligned}
$$

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

| Loss |
Input: $\hat{\boldsymbol{y}}$
Output: $\ell_{BCE}(\hat{\boldsymbol{y}},\boldsymbol{y}) = \frac{1}{K}\sum_{i=1}^{K} -[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$,
$\hat{y}_i = \frac{1}{1+\exp(-z_{3i})}$
$\frac{\partial \ell}{\partial \hat{y}} = \frac{1}{K}(\frac{-y}{\hat{y}} + \frac{1-y}{1-\hat{y}})^T \implies \frac{1}{K}\frac{\hat{y}_i - y_i}{\hat{y}_i(1-\hat{y}_i)}$ are elements

(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.

Sigmoid function is more computationally intensive to compute compared to ReLU due to exponential operation; the latter produces sparsity in matrices which encourages numerical optimization techniques taking advantage of this.

ReLU also avoids the vanishing gradient problem, whereby the number of parameters receive very small updates such that the nodes deviate greatly from their optimal value. As the gradient is constant for ReLU compared to the sigmoid gradient always being smaller than 1, successive operations will start to prohibit learning.

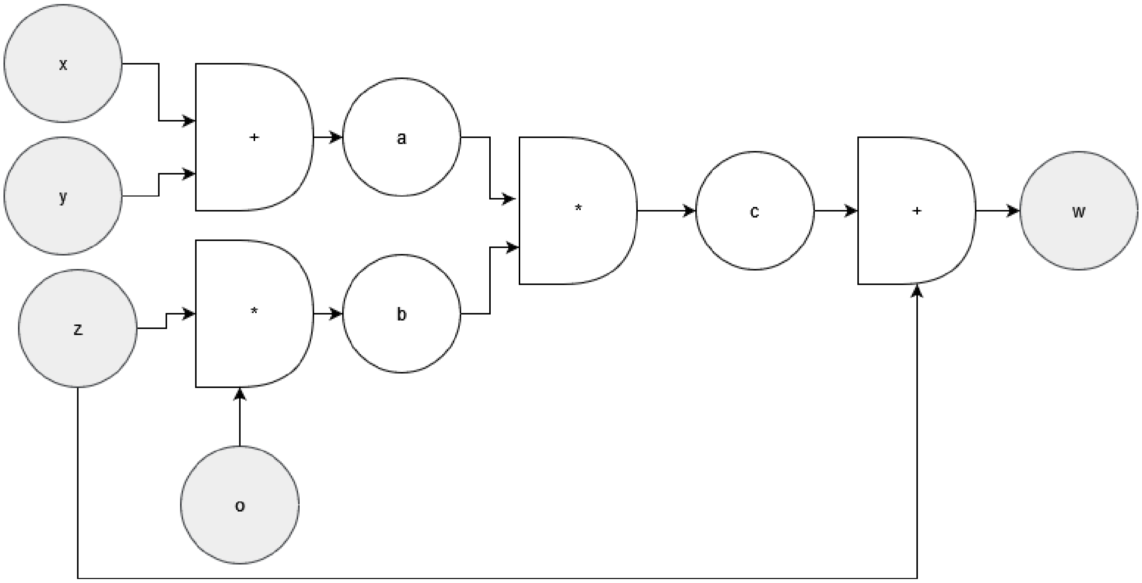
1.4 Conceptual Questions

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

In practice (and considering floating point arithmetic), this will never happen as this function is converting logits to probabilities. Outputs may be very close to 1 and 0 in a “confident classifier” scenario, but will not match them exactly. Note: the “softmax” is not a smooth approximation to the maximum function; it is actually an approximation to the arg max function whose value is the index that has the maximum. Accordingly, some prefer the “softargmax” terminolgy to emphasize this distinction.

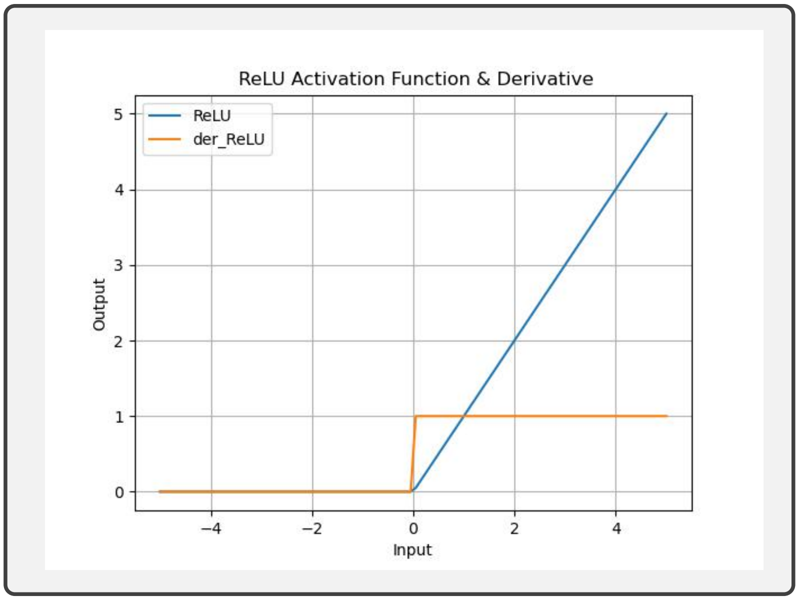
(b) (3pt) Draw the computational graph defined by this function, with inputs $x, y, z \in \mathbb{R}$ and output $w \in \mathbb{R}$. You make use symbols x, y, z, o, and operators $*, +$ in your solution. Be sure to use the correct shape for symbols and operators as shown in class.

$$a = x + y$$
$$b = z * o$$
$$c = a * b$$
$$w = c + z$$

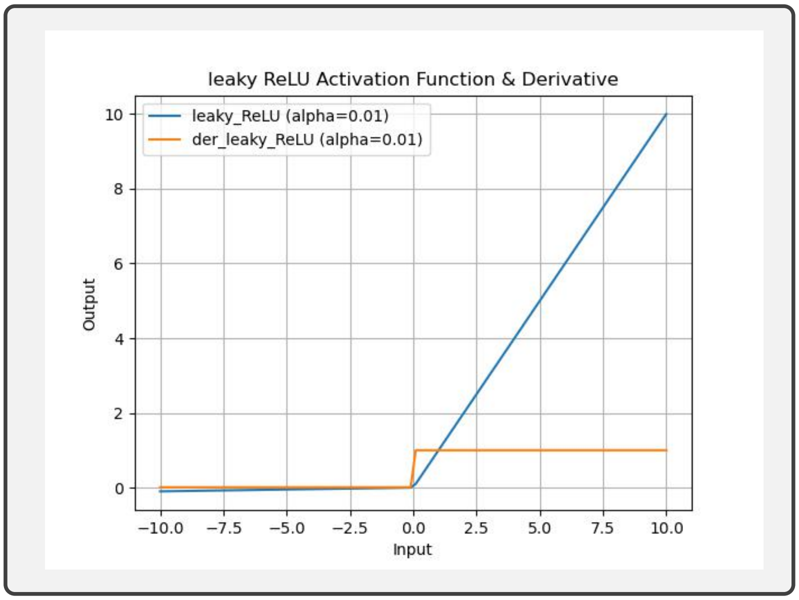


(c) Draw the graph of the following (and its derivative):

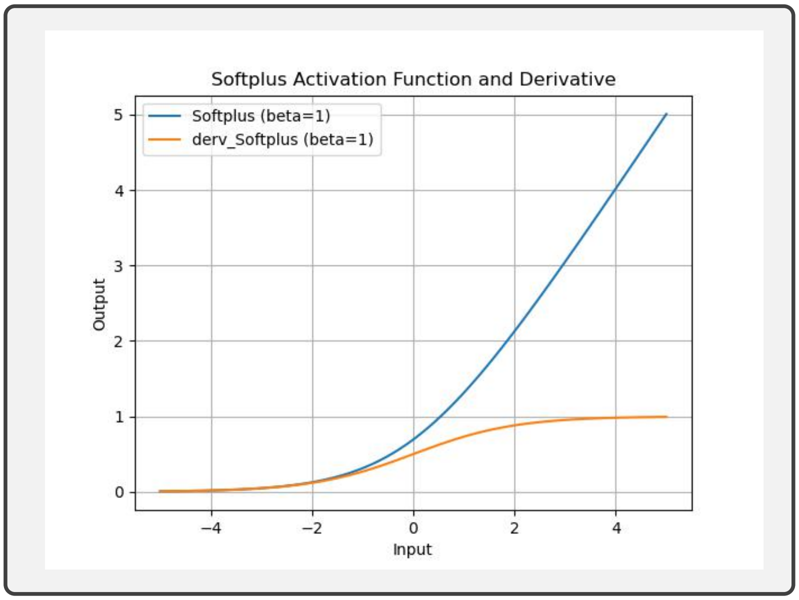
(a) ReLU()



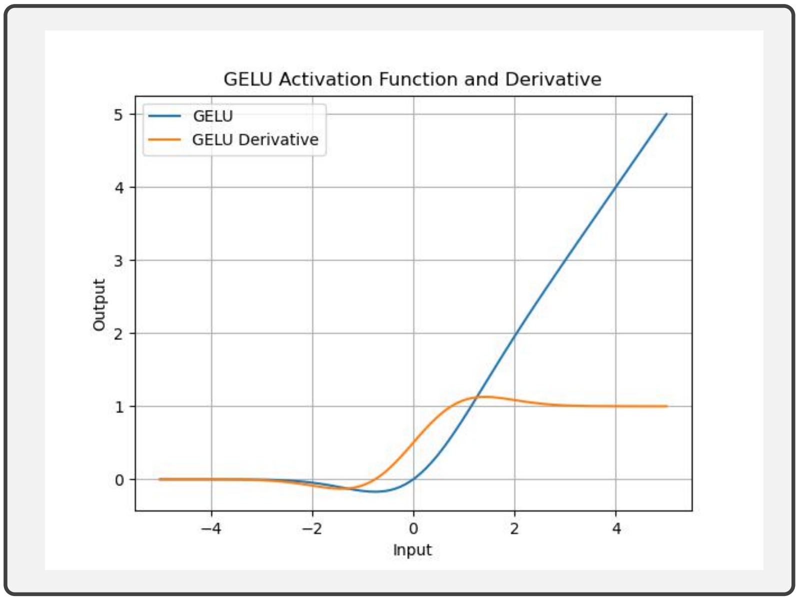
(b) LeakyReLU(0.01)



(c) Softplus(beta is 1)


Softplus Activation Function and Derivative
Softplus (beta=1)
derv_Softplus (beta=1)
Output
0
1
2
3
4
5
−4
−2
0
2
4
Input

(d) GELU


GELU Activation Function and Derivative
GELU
GELU Derivative
Output
0
1
2
3
4
5
−4
−2
0
2
4
Input

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

> There is the issue of dead neurons when using the ReLU activation function. If a large gradient flows through a ReLU neuron, it updates weights such that the neuron will always output zero, which prevents any stops learning as no input will be considered. Note that its output is also unbounded. To address these concerns the leaky ReLU and softplus prevents dying neurons by introducing non-zero, constant gradient for negative inputs in the case of the former and offers a smooth, differentiable alternative that has natural bounds with the latter.

(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?

> Reflection, rotation, scaling, and shear are the main types of linear transformations. In neural networks, each output unit produces the linear combination of the inputs and the connection weights. Once we allow translation, these essentially become affine transformations. To introduce nonlinearity to the model (to capture greater complexity by having a larger hypothesis space), activation functions ingest these transformations through a nonlinear function and treat that as the unit output.

2 Implementation (50pt)

2.1 Backpropagation (35pt)

You need to implement the forward pass and backward pass for `Linear`, `ReLU`, `Sigmoid`, `MSE loss`, and `BCE loss` in the attached `mlp.py` file. We provide three example test cases `test1.py`, `test2.py`, `test3.py`. We will test your implementation with other hidden test cases, so please create your own test cases to make sure your implementation is correct.

Recommendation: Go through this Pytorch tutorial to have a thorough understanding of Tensors.

Extra instructions:

1. Please use Python version ≥ 3.7 and PyTorch version > 1.7.1 (this code is tested on PyTorch version 2.4.0). We recommend you to use Miniconda the

manage your virtual environment.

2. We will put your mlp.py file under the same directory of the hidden test scripts and use the command `python hiddenTestScriptName.py` to check your implementation. So please make sure the file name is mlp.py and it can be executed with the example test scripts we provided.

3. You are not allowed to use PyTorch autograd functionality in your implementation.

4. Be careful about the dimensionality of the vector and matrix in PyTorch. It is not necessarily follow the the Math you got from part 1.

2.2 Gradient Descent (15pt + 5pt)

In DeepDream, the paper claims that you can follow the gradient to maximize an energy with respect to the input in order to visualize the input. We provide some code to do this. Given a image classifier, implement a function that performs optimization on the input (the image), to find the image that most highly represents the class. You will need to implement the gradient_descent function in sgd.py. You will be graded on how well the model optimizes the input with respect to the labels.

Extra hints:

1. We try to *minimize* the energy of the class, e.g. maximize the class logit. Make sure you are following the gradient in the right direction

2. A reasonable starting learning rate to try is 0.01, but depending on your implementation, make sure to sweep across a few magnitudes.

3. Make sure you use normalize_and_jitter, since the neural network expect a normalized input. Jittering produces more visually pleasing results

You may notice that the images that you generate are very messy and full of high frequency noise. Extra credit (5 points) can be had by generating visually pleasing images, and experimenting with visualizing the middle layers of the network. There are some tricks to this:

1. Blur the image at each iteration, which reduces high frequency noise

2. Clamp the pixel values between 0 and 1

3. Implement weight decay

4. Blur the gradients at each iteration

5. Implement gradient descent at multiple scales, scaling up every so often