Homework 1: Backpropagation

CSCI-GA 2572 Deep Learning

Spring 2022

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 `02/17`. Submit the following files in a zip file `your_net_id.zip` through NYU Brightspace:

- `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.

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) (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.

(b) (5pt) 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		
Loss		

(c) (8pt) 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
$W^{(1)}$	
$b^{(1)}$	
$W^{(2)}$	
$b^{(2)}$	

(d) (3pt) 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)?

Solution:

(a)

1. Clear the gradients of all parameters. (model.zero_grad())
2. Forward the Data (output = model.forward(batch))
3. Compute the loss function (loss = criterion(output, target))
4. Backpropagation to compute the gradients of the loss function with respect to the parameters. (loss.backward())
5. Update the parameters and update the optimizer status. (optimizer.step())

(b)

$\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{W}^{(2)}, \boldsymbol{b}^{(2)}$

For a single data point (x, y), we have:

Layer	Input	Output
$\texttt{Linear}_1$	$\boldsymbol{x}$	$\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)}$
f	$\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)}$	$f(\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)})$
$\texttt{Linear}_2$	$f(\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)})$	$\boldsymbol{W}^{(2)}f(\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)})+\boldsymbol{b}^{(2)}$
g	$\boldsymbol{W}^{(2)}f(\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)})+\boldsymbol{b}^{(2)}$	$\boldsymbol{W}^{(2)}f(\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)})+\boldsymbol{b}^{(2)}$
$\texttt{Loss}$	$\boldsymbol{W}^{(2)}f(\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)})+\boldsymbol{b}^{(2)}$	$\|\boldsymbol{W}^{(2)}f(\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)})+\boldsymbol{b}^{(2)}-\boldsymbol{y}\|^2$

(c)

Parameter	Gradient
$W^{(1)}$	
$b^{(1)}$	
$W^{(2)}$	$2(\boldsymbol{W}^{(2)}f(\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)})+\boldsymbol{b}^{(2)}-\boldsymbol{y})f(\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)})$
$b^{(2)}$	

(d)

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) (2pt + 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.

(b) (2pt + 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)?

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

Solution: (a)

Layer	Input	Output
`Linear`$_1$		
f		
`Linear`$_2$		
g		
`Loss`		

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

(b)

(c)

(d)

1.4 Conceptual Questions

(a) (1pt) Why is softmax actually soft(*arg*)max?

(b) (1pt) In what situations, soft(*arg*)max can become unstable?

(c) (1pt) Should we have two consecutive linear layers in a neural network? Why or why not?

(d) (4pt) Can you draw the graph of the derivative for the following functions?

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

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

(f) (1pt) How should we adjust the learning rate as we increase or decrease the batch size?

Solution:

(a)

Soft-{Function-Name} is the name of the softened version of the function.

The argmax function takes a vector as input and returns a one-hot vector of the maximum value.

The softmax (softargmax) function is a softened version of this function, which return a distribution but not the value of the maximum value.

(b)

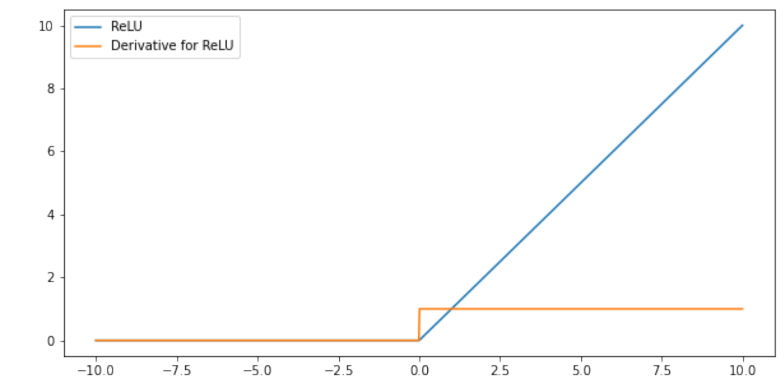
If the maximum value and the minimum value are in quite different scales, the softmax function will become unstable (numerically unstable for values with a very large range).

(c)

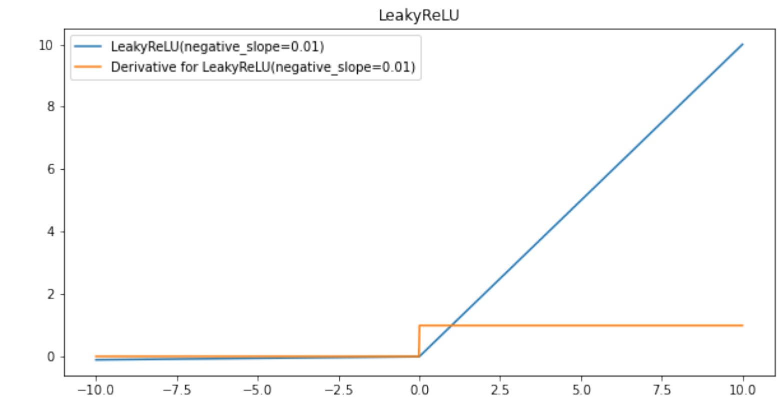
We should not have two consecutive linear layers in a neural network. Two consecutive linear layers can be transfered to a single linear layer.

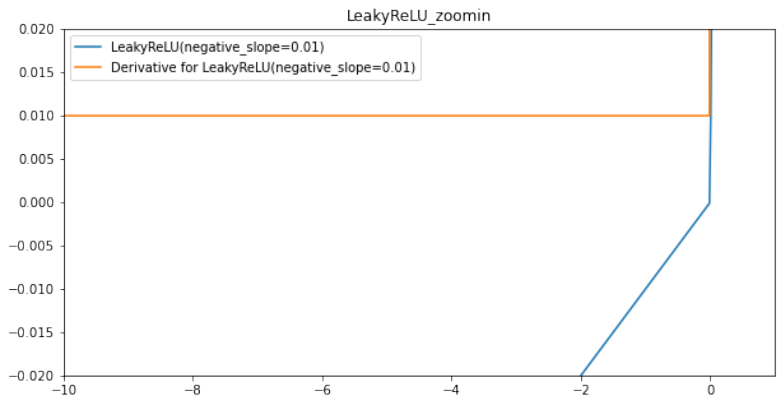
(d)

ReLU

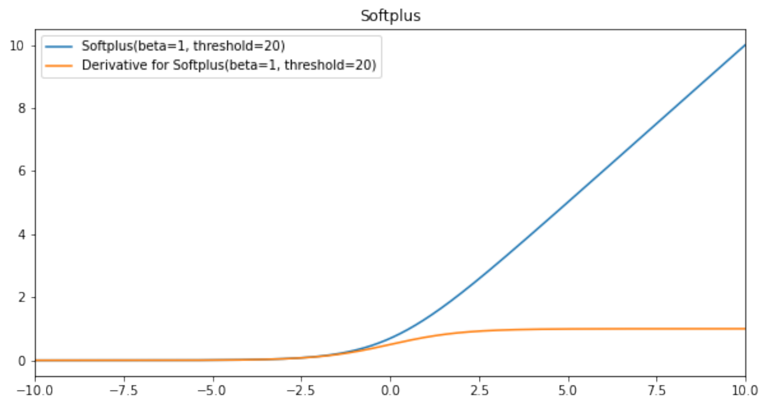

ReLU
Derivative for ReLU

LeakyReLU

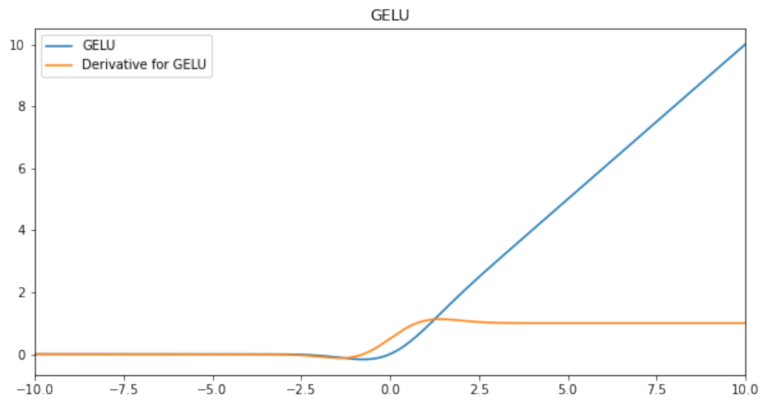

LeakyReLU
LeakyReLU(negative_slope=0.01)
Derivative for LeakyReLU(negative_slope=0.01)


LeakyReLU_zoomin
LeakyReLU(negative_slope=0.01)
Derivative for LeakyReLU(negative_slope=0.01)

Softplus(beta=1)



GELU



(d) The four types of linear transformations are: dilations, shears, rotations, reflections and projections.

What is the role of linear transformation and non linear transformation in a neural network?

(e)

When we increasing the batch size, the learning rate should be decreased. When we decrease the batch size, the learning rate should be increased.

2 Implementation (50pt)

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. 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.