Homework 1: Backpropagation

DS-GA 1008 Deep Learning

Fall 2020

The goal of homework 1 is to help you understand how to update network parameters by using backpropagation algorithm. There are two parts in this homework.

For part 1, you need to answer the questions with mathematics equations and some explanations. 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. You need to submit your `mlp.py` file for this part.

The due date of homework 1 is `10/02`. Submit the following files in a zip file `your_net_id.zip` through NYU classes:

- `queston_1.pdf`
- `mlp.py`

1 Two-Layer Neural Network

You are given the following neural net architecture:

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

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

1.1 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 $\ell_{\text{MSE}}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$, where y is the target output.

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

(b) For a single data point (x, y), write down all inputs and outputs for forward pass of each layer. You can only use $x, y, W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}$ in your answer.

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

(c) Write down the gradient calculated from the backward pass. You can only use $x, y, W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}, \frac{\partial \ell}{\partial \hat{y}}, \frac{\partial z_2}{\partial z_1}, \frac{\partial \hat{y}}{\partial z_3}$ in your answer, where $z_1, z_2, z_3, \hat{y}$ are the outputs of $\texttt{Linear}_1, f, \texttt{Linear}_2, g$.

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

(d) Show us the elements of $\frac{\partial z_2}{\partial z_1}$, $\frac{\partial \hat{y}}{\partial z_3}$ and $\frac{\partial \ell}{\partial \hat{y}}$?

1.2 Classification Task

We would like to perform binary classification (*i.e.* $K = 1, y \in \{0, 1\}$) by using a "binary network", 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.

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

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

(b) Now you think you can do a better job by using a *binary cross-entropy* (BCE) loss function $\ell_{\text{BCE}}(\hat{y}, y) = -\left[y \log(\hat{y}) + (1-y)\log(1-\hat{y})\right]$. what do you need to change in the equations of (b), (c) and (d)?

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

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

(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.

2 Implementation

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. You are not allowed to use the autograd functions in `PyTorch`. We will test your results with different test cases. We provide one example test case `test.py`. Please create your own test cases and make sure your implementation is correct.