**Eric He**

**eh1885**

**Deep Learning**

# Homework 1: Backpropagation

# 1. Two-layer neural network

## 1.1 Regression task

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

Suppose we are provided `model`, a Torch model object, a loss function given by `l(y_pred, y_true)`, and a learning rate given by `lr`.

```
import torch
import torch.optim as optim

optimizer = optim.SGD(model.parameters(), lr=lr)

y_pred = model.forward() # 1. forward pass
loss = l(y_pred, y_true) # 2. compute loss
optimizer.zero_grad() # 3. zero the grad-parameters
loss.backward() # 4. accumulate the grad-parameters
optimizer.step() # 5. step in the opposite direction of the grad-parameters
```

### 1.1.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.

**Linear 1**

Input: $x$

Output: $W^1 x + b^1$

**f**

Input: $W^1 x + b^1$

Output: $f(W^1 x + b^1)$

**Linear 2**

Input: $f(W^1 x + b^1)$

Output: $W^2(f(W^1 x + b^1)) + b^2$

**g**

Input: $W^2(f(W^1 x + b^1)) + b^2$

Output: $g(W^2(f(W^1 x + b^1)) + b^2)$

**Loss**

Input: $g(W^2(f(W^1x+b^1))+b^2)$

Output: $\frac{1}{2}(g(W^2(f(W^1x+b^1))+b^2)-y)^2$

## 1.1.c Write down the gradient calculated from the backward pass. You can only use $x, y, W^1, b^1, W^2, b^2, \frac{\delta l}{\delta y}, \frac{\delta z_2}{\delta z_1}, \frac{\delta \hat{y}}{\delta z_3}$ in your answer, where $z_1, z_2, z_3, \hat{y}$ are the outputs of `Linear_1, f, Linear_2, g`.

For convenience, assume $W^1$ is a matrix of shape $(m_1, n_1)$ and assume $W^2$ is a matrix of shape $(m_2, n_2)$. Denote $I_{m_1}$ to be the identity matrix of shape $(m_1, m_1)$ and $I_{m_2}$ to be the identity matrix of shape $(m_2, m_2)$.

**W1**

We first expand the derivative:

$$\frac{\delta l}{\delta W^1} = \frac{\delta l}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta z_3} \frac{\delta z_3}{\delta z_2} \frac{\delta z_2}{\delta z_1} \frac{\delta z_1}{\delta W^1}$$

We now have to fill in the gradients $\frac{\delta z_3}{\delta z_2}$ and $\frac{\delta z_1}{\delta W^1}$.

As $z_3 = W^2 z_2 + b^2$, we have $\frac{\delta z_3}{\delta z_2} = W^2$.

As $z_1 = W^1 x + b^1$, we have $\frac{\delta z_1}{\delta W^1} = I_{m_1} x^T$. Notice that I have written this gradient as a matrix where $x^T$ is repeated several $m_1$ times; strictly speaking, because $z_1$ is a vector of shape $m_1$ and $W_1$ is a matrix of shape $(m_1, m_2)$, the Jacobian is of shape $(m_1, m_1, m_2)$. However, because $W_1$ is linear, the Jacobian is diagonal along one of the $m_1$ dimensions and I was able to squash it to a matrix with no repercussions.

Then the entire computation is

$$\frac{\delta l}{\delta W^1} = \frac{\delta l}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta z_3} W^2 \frac{\delta z_2}{\delta z_1} I_{m_1} x^T$$

## 1.1.d Show us the elements of $\frac{\delta z_2}{\delta z_1}$, $\frac{\delta \hat{y}}{\delta z_3}$ and $\frac{\delta l}{\delta \hat{y}}$?

$\frac{\delta z_1}{\delta z_1}$ is the derivative of the ReLU operation applied elementwise to the vector $z_1$. The subgradient of ReLU(x) is $1$ if $x > 0$ and $0$ otherwise, so we can write $\frac{\delta z_2}{\delta z_1} = 1_{z_1 > 0}$, the indicator function elementwise for if $z_{1i} > 0$.

$\frac{\delta \hat{y}}{\delta z_3}$ is the derivative of the identity function of $z_3$. The identity function has derivative $1$, the identity matrix.

$\frac{\delta l}{\delta \hat{y}}$ is the derivative of the square loss, and is given by $\hat{y} - y$.

# 1.2 Classification Task

### 1.2.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 MLE loss function.

The forward pass would look the same, of course as I retained the $f$ and $g$ notations in the original answer. However, instead of $f(W^1 x + b^1) = \max(0, W^1 x + b^1)$, we now have $f(W^1 x + b^1) = \sigma(W^1 x + b^1)$. And instead of $g(W^2(f(W^1 x + b^1)) + b^2) = W^2(f(W^1 x + b^1)) + b^2$, $g_2$ is now $\sigma(W^2(f(W^1 x + b^1)) + b^2)$

The backward pass now involves the deriving the logistic sigmoid function, $\sigma(z) = (1 + \exp(-z))^{-1}$. We have

$$\begin{aligned}\frac{\delta \sigma}{\delta z} &= -(1 + \exp(-z))^{-2})(-\exp(-z)) \\ &= \frac{\exp(-z)}{(1 + \exp(-z))^2}\end{aligned}$$

For $f(z_1)$, this would correspond to $z_1 = W^1 x + b^1$. We would substitute in the above for $\frac{\delta z_2}{\delta z_1}$.

For $g(z_3)$, this would correspond to $z_3 = W^2(f(W^1 x + b^1)) + b^2$. We would substitute in the above for $\frac{\delta \hat{y}}{\delta z_3}$.

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

In the forward pass, we would only have to change the final loss function.

In the backward pass, we would have to change the $\frac{\delta l}{\delta \hat{y}}$ multiple in each of the gradient calculations. The gradient of the cross-entropy loss function with respect to $\hat{y}$ is given by $-(\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}})$

## 1.2.c

Setting $f$ to be the ReLU function is better for training deeper networks because the ReLU function does not have gradient saturation in the same way the sigmoid function does. The ReLU function's derivative, as claimed earlier, is an indicator function, while the sigmoid function's derivative goes to 0 as the sigmoid's input takes extreme value.