Homework 1: Backpropagation

DS-GA 1008 Deep Learning

1 Two-Layer Neural Network

1.1 Regression Task

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

Step 1. *prediction = model(batch)* - put the single batch of data into the model

Step 2. *loss = criterion(prediction, batch.label)* - calculate the loss between predicted label and ground-truth label.

Step 3. *optimizer.zero_grad()* - initialize the gradient to be zero

Step 4. *loss.backward()* - compute the backpropagation

Step 5. *optimizer.step()* - update the weights

(b)

Layer	Input	Output
Linear_1	x	$W^{(1)}x + b^{(1)}$
f	$W^{(1)}x + b^{(1)}$	$(W^{(1)}x + b^{(1)})^+$
Linear_2	$(W^{(1)}x + b^{(1)})^+$	$W^{(2)}(W^{(1)}x + b^{(1)})^+ + b^{(2)}$
g	$W^{(2)}(W^{(1)}x + b^{(1)})^+ + b^{(2)}$	$W^{(2)}(W^{(1)}x + b^{(1)})^+ + b^{(2)}$
Loss	$W^{(2)}(W^{(1)}x + b^{(1)})^+ + b^{(2)}$, y	$\frac{1}{2}(W^{(2)}\left(W^{(1)}x + b^{(1)}\right)^+ + b^{(2)} - y)^2$

(c)

Parameter	Gradient
$W^{(1)}$	$x \times \frac{\partial l}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_3} \times W^{(2)} \times \frac{\partial z_2}{\partial z_1}$
$b^{(1)}$	$\frac{\partial l}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_3} \times W^{(2)} \times \frac{\partial z_2}{\partial z_1}$
$W^{(2)}$	$(W^{(1)}x + b^{(1)})^+ \times \frac{\partial l}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_3}$
$b^{(2)}$	$\frac{\partial l}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_3}$

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

$\frac{\partial z_2}{\partial z_1}$ is a diagonal matrix with the element $(\frac{\partial z_2}{\partial z_1})_{ii} = \begin{cases} 0, & z_{1i} < 0 \\ 1, & z_{1i} \geq 0 \end{cases}$ on the diagonal, and 0 elsewhere.

$\frac{\partial z_2}{\partial z_1}$ is a diagonal matrix with the element $(\frac{\partial \hat{y}}{\partial z_3})_{ii} = 1$ on the diagonal, and 0 elsewhere.

$\frac{\partial l}{\partial \hat{y}}$ is a vector that $\frac{\partial l}{\partial \hat{y}} = (\hat{y} - y)^T$

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 $\sigma(z) = (1 + \exp(-z))^{-1}$.

(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
Linear_1	x	$W^{(1)}x + b^{(1)}$
f	$W^{(1)}x + b^{(1)}$	$\sigma(W^{(1)}x + b^{(1)})$
Linear_2	$\sigma(W^{(1)}x + b^{(1)})$	$W^{(2)}\sigma(W^{(1)}x + b^{(1)}) + b^{(2)}$
g	$W^{(2)}\sigma(W^{(1)}x + b^{(1)}) + b^{(2)}$	$\sigma(W^{(2)}\sigma(W^{(1)}x + b^{(1)}) + b^{(2)})$
Loss	$\sigma(W^{(2)}\sigma(W^{(1)}x + b^{(1)}) + b^{(2)})$, y	$\frac{1}{2}(\sigma(W^{(2)}\sigma(W^{(1)}x + b^{(1)}) + b^{(2)}) - y)^2$

Parameter	Gradient
$W^{(1)}$	$x \times \frac{\partial l}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_3} \times W^{(2)} \times \frac{\partial z_2}{\partial z_1}$
$b^{(1)}$	$\frac{\partial l}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_3} \times W^{(2)} \times \frac{\partial z_2}{\partial z_1}$
$W^{(2)}$	$\sigma(W^{(1)}x + b^{(1)}) \times \frac{\partial l}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_3}$
$b^{(2)}$	$\frac{\partial l}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_3}$

$\frac{\partial z_2}{\partial z_1}$ is a diagonal matrix with the element $(\frac{\partial z_2}{\partial z_1})_{ii} = [\sigma(z_1)(1 - \sigma(z_1))]_i$,
$\frac{\partial \hat{y}}{\partial z_3}$ is a diagonal matrix with the element $(\frac{\partial \hat{y}}{\partial z_3})_{ii} = [\sigma(z_3)(1 - \sigma(z_3))]_i$,

$\frac{\partial l}{\partial \hat{y}} = (\hat{y} - y)^T$.

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

Layer	Input	Output
Linear_1	x	$W^{(1)}x + b^{(1)}$
f	$W^{(1)}x + b^{(1)}$	$\sigma(W^{(1)}x + b^{(1)})$
Linear_2	$\sigma(W^{(1)}x + b^{(1)})$	$W^{(2)}\sigma(W^{(1)}x + b^{(1)}) + b^{(2)}$
g	$W^{(2)}\sigma(W^{(1)}x + b^{(1)}) + b^{(2)}$	$\sigma(W^{(2)}\sigma(W^{(1)}x + b^{(1)}) + b^{(2)})$
Loss	$\sigma(W^{(2)}\sigma(W^{(1)}x + b^{(1)}) + b^{(2)})$, y	$-[ylog\left(\sigma\left(W^{(2)}\sigma\left(W^{(1)}x + b^{(1)}\right) + b^{(2)}\right)\right)$ $+(1 - y)\log\left(1 - \sigma\left(W^{(2)}\sigma\left(W^{(1)}x + b^{(1)}\right) + b^{(2)}\right)\right)]$

Parameter	Gradient
$W^{(1)}$	$x \times \frac{\partial l}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_3} \times W^{(2)} \times \frac{\partial z_2}{\partial z_1}$
$b^{(1)}$	$\frac{\partial l}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_3} \times W^{(2)} \times \frac{\partial z_2}{\partial z_1}$
$W^{(2)}$	$\sigma(W^{(1)}x + b^{(1)}) \times \frac{\partial l}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_3}$
$b^{(2)}$	$\frac{\partial l}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_3}$

$\frac{\partial z_2}{\partial z_1}$ is a diagonal matrix with the element $(\frac{\partial z_2}{\partial z_1})_{ii} = [\sigma(z_1)(1 - \sigma(z_1))]_i$,

$\frac{\partial \hat{y}}{\partial z_3}$ is a diagonal matrix with the element $(\frac{\partial \hat{y}}{\partial z_3})_{ii} = [\sigma(z_3)(1 - \sigma(z_3))]_i$,

$\frac{\partial l}{\partial \hat{y}} = (-\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}})^T$.

(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.

The advantage of using ReLU as f is that it can reduce the likelihood of gradient vanishing. The gradients of sigmoid is always smaller than 1, so when the network goes deeper, the gradient, we multiply these gradients according to chain rule, and the results would go to zero quickly. But the gradient of ReLU is constant, so it helps to reduce the likelihood of gradient vanishing.

Another benefit would be improving computational efficiency. While sigmoid needs some operation of exponential computation, ReLU needs only recognizing whether the input is positive or negative, and generating sparsity in representation.