Yukai Yang DS-GA 1008 HW1

1.1 Two-Layer Neural Nets

$Linear_1 \to f \to Linear_2 \to g$

1.2 Regression Task

(a) 5 steps to train the model with PyTorch using SGD

1. Feed forward to get the logtis
2. Compute parameters in forward pass, and then compute the cost (or accuracy).
3. Clear previous gradients.
4. Compute the backward pass parameters and accumulate the gradient back to the first layer.
5. Update the parameters with the partial derivative we got from previous step.

(b)Write all inputs and outputs for each layer.

Answers

Layer	Input	Output
Linear_1	x	$W^{(1)}x + b^{(1)}$
f(ReLU)	$W^{(1)}(x) + b^{(1)}$	$\max\{0, W^{(1)}(x) + b^{(1)}\}$
Linear_2	$\max\{0, W^{(1)}(x) + b^{(1)}\}$	$W^{(2)}(\max\{0, W^{(1)}(x) + b^{(1)}\}) + b^{(2)}$
g(Identity)	$W^{(2)}(\max\{0, W^{(1)}(x) + b^{(1)}\}) + b^{(2)}$	$W^{(2)}(\max\{0, W^{(1)}(x) + b^{(1)}\}) + b^{(2)}$
Loss(MSE)	$W^{(2)}(\max\{0, W^{(1)}(x) + b^{(1)}\}) + b^{(2)}, y$	$\|\|(W^{(2)} \cdot (\max\{0, W^{(1)}(x) + b^{(1)}\}) + b^{(2)} - y)^2\|\|$

Answers

Parameter	Gradient
W^{(1)}	$(\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_3} \cdot W^{(2)} \cdot \frac{\partial z_2}{\partial z_1})^T \cdot x^T$
b^{(1)}	$\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_3} \cdot W^{(2)} \cdot \frac{\partial z_2}{\partial z_1}$
W^{(2)}	$\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_3} \cdot (\max\{0, W^{(1)}x + b^{(1)}\})^T$
b^{(2)}	$\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_3}$

$$\text{gradient of } W^{(1)} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_3} \cdot \frac{\partial z_3}{\partial z_2} \cdot \frac{\partial z_2}{\partial z_1} \cdot \frac{\partial z_1}{\partial W^{(1)}}$$
$$\text{gradient of } b^{(1)} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_3} \cdot \frac{\partial z_3}{\partial z_2} \cdot \frac{\partial z_2}{\partial z_1} \cdot \frac{\partial z_1}{\partial b}$$
$$\text{gradient of } W^{(2)} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_3} \cdot \frac{\partial z_3}{\partial W^{(2)}}$$
$$\text{gradient of } b^{(2)} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_3} \cdot \frac{\partial z_3}{\partial b^{(2)}}$$

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

$\frac{\partial z_2}{\partial z_1} = 1 \text{ if } z_1 > 0, = 0 \text{ otherwise.}$

$(\text{More formally it's actually } 0 \text{ only when } z_1 < 0, \text{ but DNE when } z_1 = 0)$

$$(\frac{\partial \hat{y}}{\partial z_3})_{ij} = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ if } i \neq j \end{cases}.$$

$\frac{\partial l}{\partial \hat{y}} = 2\hat{y} - 2y. \text{ (Here } \hat{y} \text{ is the same output of } g \text{ in part (b) above.)}$

1.3 Classification Task

$f, g = \sigma$, the sigmoid function. Others remain unchanged.

(a) What to change in the 1.2 (b), (c) and (d)

For input and output, just change everything with $ReLU$ into a sigmoid function. Meanwhile, the output of g is also going to become the sigmoid of its input, instead of unchanged like above. The rest are the same.

For gradient, the chain rule still works the same, but specific terms of the activation functions need to be changed. Namely, if we still take $\frac{\partial z_2}{\partial z_1}, \frac{\partial \hat{y}}{\partial z_3}, \text{ and } \frac{\partial l}{\partial \hat{y}}$ as granted without expanding them all, then we just need to change the gradient of $W^{(2)}$ above into $\frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z_3} \cdot \sigma(W^{(1)}x + b^{(1)})$.
For the elements of the 3 granted items, $\frac{\partial z_2}{\partial z_1} = \sigma(z_1) \cdot (1 - \sigma(z_1)) \text{ since } z_2 = \sigma(z_1)$. Similarly,
$\frac{\partial \hat{y}}{\partial z_3} = \sigma(z_3) \cdot (1 - \sigma(z_3))$.
$\frac{\partial l}{\partial \hat{y}}$ remains unchanged. Also same definition for $\hat{y}$ as above).

(b) Change f = ReLU(), g = Sigmoid(). Explain the choice of f.

Like explained in 1.4 (d) below, $ReLU$ is very helpful in preventing gradient vanishing, which would strongly undermines the learning of NN if happened in the first layer.
It also computes faster then $Sigmoid$, considering the structure of these two functions.

1.4 Conceptual Questions

(a) Why is softmax actually soft(arg)max?

It does not directly return the max value. Instead, it will trasnform the input into a probability distribution that maintains the relative magnitiude difference. It only shows us which original element was the largest, and that's how the arg comes from.

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

Overflow and Underflow may be problematic for softmax. Numbers that are too large will be approximated as ∞, while those that are too small will be approximated as 0. If a lot of such approximation exists, the final result may become unreliable. [2]

(c) Y/Y not 2 consecutive linear layers?

Of course no. A normal sandwich always has one piece of ham between two pieces of bread. Unless u went for double cheeseburger or something.

For neural network, say we have $z_1 = w_1 x + b_1, z_2 = w_2 z_1 + b_2$. It's obvious that we can combine these two together and get $z_2 = w_2 \cdot w_1 \cdot x + w_2 \cdot b_1 + b_2$. This is still in $z = Wx + b$ format. Thus 2 consecutive linear layers are just unnecessary at all.

(d) One pro one con for Sigmoid, Tanh, ReLU and LeakyReLU

Aa Activation Function	Pros	Cons
Sigmoid	Sensitive to changes in X when X is close to 0	Values always positive, can be confusing in some cases
Tanh	Y values symmetric to 0, often more convenient for calculations	Same with sigmoid, may become useless when X is too small/large (as it goes to the tail of the graph)
ReLU	The 0 in negative part helps to learn much faster	No learning at all in the negative area, but sometimes may need it
LeakyReLU	Even in negative X region can still learn new things	Need more computations than ReLU

(e) 4 Different types of Linear Transformation + Roles of Linear and Non-linear Transformation

(I thought in class we said 5, so maybe just take one of these off and... it should be fine 😂)
Dilation, Rotation, Reflection, Shearing, Translation.

Linear Transformation in NN: affect the input data in the 5(or 4) ways above. This will be used for later calculations.

Non-linear Transformation: Without this, according to part (c) above, we won't learn anything new from the network, as everything can be just reproduced via addition and multiplcation of the input. Theoretically, a NN should be able to take input from $-\infty$ to ∞, and convert it to $0 \sim 1$ or $-1 \sim 1$. That will require non-linearity to be

introduced.
Another more advanced(but essentially I think this would be reversing the logic) is that Universal Approximation Theorem indicates that non-linearity must be included in a useful neural network.

(f) How to adjust learning rate as batch size changes?

Usually a larger batch size requires a larger learning rate. When batch size increases or becomes too large, the generalization to the whole dataset may become more difficult. Therefore we don't want to remain a rather low learning rate as that may lead to a higher training accuracy but will certainly undermine the testing accuracy. Vice versa.

Reference

[1] Numerically Stable Softmax
https://stackoverflow.com/questions/42599498/numercially-stable-softmax