Mathematical Foundation of DNN : HW 6

Jeong Min Lee

April 17, 2024

1

Let $D_p(\cdot)$ define as follows:

$$D_p(x) = \begin{cases} 0 \text{ with probability } p \\ x/(1-p) \text{ with probability } 1-p \end{cases} \tag{1}$$

For the case of ReLU and LeakyReLU, the dropout can be commutative. Let f denote the ReLU Leaky ReLU.

$$\begin{aligned} D_p(f(h)) &= \begin{cases} 0 \text{ with probability } p \\ f(h)/(1-p) \text{ with probability } 1-p \end{cases} && (\because \text{the defition of dropout}) \\ &= \begin{cases} 0 \text{ with probability } p \\ f(h/(1-p)) \text{ with probability } 1-p \end{cases} && (\because f(ax) = af(x)\ \forall x, a > 0) \\ &= f(D_p(h)) \end{aligned}$$

However, for the sigmoid, we cannot do same thing to the discussion above since it is not homogeneous function. In addition, there is a trivial counter-example. Consider the hidden layer with element $[0, 0, 1]$. If we do dropout first, the last node can be zero with probability p and it results in $[0, 0, 0]$. Then, after applying the sigmoid function, the all three elements in the output must be same. However, if we do sigmoid activation first, since $Sigmoid(0) \neq 0$, the intermediate layer would be $[\sigma(0), \sigma(0), \sigma(1)]$. Although dropout cannot change their value to be evenly, they are not same to the result of Dropout-Sigmoid.

2

3

I refer to the result of HW4-6 to solve this problem.

i

From the problem 6-(a) in HW4, we saw that $\frac{\partial}{\partial y_{l-1}}\sigma(A_l y_{l-1}) + b_l = \text{diag}(\sigma'(A_l y_{l-1} + b_l))A_l$. Thus,

$$\begin{aligned} y_l &= \sigma(A_l y_{l-1} + b_l) + y_{l-1} \\ \frac{\partial y_l}{\partial y_{l-1}} &= \text{diag}\left(\sigma'(A_l y_{l-1} + b_l)\right) A_l + I_m \end{aligned}$$

Here, I_m denotes the identity with dimension m.

ii

Since b_l and A_l are independent to y_{l-1}, we can directly use the result of problem 6 at HW4(with simple chain rule).

$$\begin{aligned} \frac{\partial y_L}{\partial b_l} &= \frac{\partial y_L}{\partial y_l}\frac{\partial y_l}{\partial b_l} = \frac{\partial y_L}{\partial y_l}\text{diag}(\sigma'(A_l y_{l-1} + b_l)) \\ \frac{\partial y_L}{\partial A_l} &= \text{diag}(\sigma'(A_l y_{l-1}) + b_l)\left(\frac{\partial y_L}{\partial y_l}\right)^T y_{l-1}^T \end{aligned}$$

iii

Both $\frac{\partial y_L}{\partial b_i}$ and $\frac{\partial y_L}{\partial A_i}$ contain $\frac{\partial y_L}{\partial y_i}$ term. According to the chain rule,

$$\frac{\partial y_L}{\partial y_i} = \frac{\partial y_L}{\partial y_{L-1}} \cdot \frac{\partial y_{L-1}}{\partial y_{L-2}} \cdots \frac{\partial y_{i+1}}{\partial y_i} = \prod_{k=i+1}^{L} \frac{\partial y_k}{\partial y_{k-1}} = \prod_{k=i+1}^{L} \operatorname{diag}\left(\sigma'(A_k y_{k-1} + b_k)\right) A_k + I_m$$

As the equations above tell, eventhough $A_j = 0$ for some $j \in \{l+1, \cdots, L-1\}$ or $\sigma'(A_j y_{j-1} + b_j) = 0$ for some $j \in \{l+1, \cdots, L-1\}$, the identity matrices are still alive. Thus, the derivatives do not have to be zero. Note that the other components in both derivatives also are nonzero, in general.

4

a

In this problem, I used the following formula.

$$\text{trainable parameter} = (\text{kernel size})^2 \times C_{out} \times C_{in} + C_{out}$$

The overall calculation of the first convolution layer is as follow.

$$(128 \times 1^2 \times 256 + 128) + (128 \times 3^2 \times 128 + 128) + (256 \times 1^2 \times 128 + 256) = 213,504$$

For the second implementation, considering that each path have the same number of the trainable paramters, I calculated the number of trainable paramters for a single path. The whole calculation process is as follow.

$$(256 \times 1^2 \times 4 + 4) + (4 \times 3^2 \times 4 + 4) + (4 \times 1^2 \times 256 + 256) = 2456$$
$$\therefore 32 \times 2456 = 78,592$$

b

```
class STMConvLayer(nn.Module):
    def __init__(self):
        super(STMConvLayer, self).__init__()
        self.conv = nn.ModuleList()
        for _ in range(32):
            self.conv.append(
                nn.Sequential(
                    nn.Conv2d(256, 4, 1, dtype=torch.float),  # Specify dtype=torch.float
                    nn.ReLU(),
                    nn.Conv2d(4, 4, 3, padding=1, dtype=torch.float),  # Specify dtype=torch.
                                                                       float
                    nn.ReLU(),
                    nn.Conv2d(4, 256, 1, dtype=torch.float)  # Specify dtype=torch.float
                )
            )
    def forward(self,x): # x : bathced data
        output = torch.zeros(x.shape, dtype=torch.float)
        for path in self.conv:
            output += path(x.float())
        return output
```

```
STMConvLayer(
  (conv): ModuleList(
    (0-31): 32 x Sequential(
      (0): Conv2d(256, 4, kernel_size=(1, 1), stride=(1, 1))
      (1): ReLU()
      (2): Conv2d(4, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (3): ReLU()
      (4): Conv2d(4, 256, kernel_size=(1, 1), stride=(1, 1))
    )
  )
)
```