MFDNN HW6

MinGyu Shin

April 22, 2024

Problem 1 : *Dropout-ReLU = ReLU-Dropout.*

Claim : Positive(non-negative) homogeneous function can commute with dropout. (Commute means that they can switch their composition order.)

Proof: Let f be dropout with p and σ be positive(non-negative) homogeneous activation function. Note that every homogeneous functions at 0 equals zero. Then

$$\sigma(f(x)) = \begin{cases} \sigma(\frac{x}{1-p}) & w.p.\ 1-p \\ \sigma(0) & w.p.\ p \end{cases} = \begin{cases} \frac{1}{1-p}\sigma(x) & w.p.\ 1-p \\ 0 & w.p.\ p \end{cases} = f(\sigma(x))$$

Since for nonnegative α,

$$ReLU(\alpha x) = \begin{cases} \alpha x & x \geq 0 \\ 0 & x < 0 \end{cases} = \alpha \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} = \alpha ReLU(x),$$

and for negative part slope of Leaky ReLU $a > 0$,

$$LeakyReLU(\alpha x) = \begin{cases} \alpha x & x \geq 0 \\ a\alpha x & x < 0 \end{cases} = \alpha \begin{cases} x & x \geq 0 \\ ax & x < 0 \end{cases} = \alpha LeakyReLU(x),$$

that is, ReLU and LeakyReLu are positive homogeneous, (a) and (c) is true.

For Sigmoid, $\rho(x) = 1/(1+e^{-x})$, with $x = 1$,

$$\rho(f(1)) = \begin{cases} \rho(\frac{1}{1-p}) & w.p.\ 1-p \\ \rho(0) & w.p.\ p \end{cases} = \begin{cases} \frac{1}{1+e^{1/(p-1)}} & w.p.\ 1-p \\ \frac{1}{2} & w.p.\ p \end{cases} \neq \begin{cases} \frac{1}{(1-p)(1+e^{-1})} & w.p.\ 1-p \\ 0 & w.p.\ p \end{cases} = f(\rho(1)),$$

thus (b) is false.

Problem 2 : *Default weight initialization.*

Pytorch initializes weight and bias of a linear layer based on He initialization. It means the weight and bias with input $x \in \mathbb{R}^k$ are drawn from $\mathcal{U}[-\frac{1}{\sqrt{k}}, \frac{1}{\sqrt{k}}]$. Since A and b does not depends on x,

$$\mathbb{E}[y_i] = \mathbb{E}[\mathbb{E}[A_i y_{i-1} + b_i | y_{i-1}]] = \mathbb{E}[y_{i-1}\mathbb{E}[A_i] + \mathbb{E}[b_i]] = 0 \quad i = 1, \dots, L.$$

$$\begin{aligned}
Var(y_\ell)_{ij} &= \mathbb{E}[(y_{\ell i} - \mathbb{E}[y_{\ell i}])(y_{\ell j} - \mathbb{E}[y_{\ell j}])] \\
&= \mathbb{E}[y_{\ell i} y_{\ell j}] \\
&= \mathbb{E}[A_{\ell i} y_{\ell-1} A_{\ell j} y_{\ell-1}] + \mathbb{E}[A_{\ell i} y_{\ell-1} b_{\ell j}] + \mathbb{E}[A_{\ell j} y_{\ell-1} b_{\ell i}] + \mathbb{E}[b_{\ell i} b_{\ell j}] \\
&= \mathbb{E}[y_{\ell-1}^\intercal A_{\ell i}^\intercal A_{\ell j} y_{\ell-1}] + \mathbb{E}[A_{\ell i} y_{\ell-1} b_{\ell j}] + \mathbb{E}[A_{\ell j} y_{\ell-1} b_{\ell i}] + \mathbb{E}[b_{\ell i} b_{\ell j}] \\
&= \mathbb{E}[y_{\ell-1}^\intercal \mathbb{E}[A_{\ell i}^\intercal A_{\ell j} | y_{\ell-1}] y_{\ell-1}] + \mathbb{E}[\mathbb{E}[A_{\ell i} | y_{\ell-1}] y_{\ell-1} \mathbb{E}[b_{\ell j} | y_{\ell-1}]] + \mathbb{E}[\mathbb{E}[A_{\ell j} | y_{\ell-1}] y_{\ell-1} \mathbb{E}[b_{\ell i} | y_{\ell-1}]] + \mathbb{E}[b_{\ell i} b_{\ell j}] \\
&= \delta_{ij} \mathbb{E}[y_{\ell-1}^\intercal \mathrm{diag}(\frac{1}{3n_{\ell-1}}) y_{\ell-1}] + \delta_{ij} \frac{1}{3n_{\ell-1}} \\
&= \delta_{ij} \mathrm{Tr}(\mathrm{diag}(\frac{1}{3n_{\ell-1}}) Var(y_{\ell-1})) + \delta_{ij} \frac{1}{3n_{\ell-1}} \quad \ell = 1, \dots, L.
\end{aligned}$$

By recursive calculation,

$$Var(y_L) = \frac{1}{3^L} + \sum_{i=1}^{L} \frac{1}{3^i n_{L-i}}$$

Problem 3 : *Backprop for MLP with residual connections.*

(i) For $\ell = L$,

$$\frac{\partial y_L}{\partial y_{L-1}} = A_L.$$

For $\ell = 2, \dots, L-1$,

$$\frac{\partial y_\ell}{\partial y_{\ell-1}} = \frac{\partial(\sigma(A_\ell y_{\ell-1} + b_\ell) + y_{\ell-1})}{\partial y_{\ell-1}} = \mathrm{diag}(\sigma'(A_\ell y_{\ell-1} + b_\ell))A_\ell + I_m,$$

as p6, hw4.

(ii) For $\ell = L$,

$$\frac{\partial y_L}{\partial b_L} = 1, \qquad \frac{\partial y_L}{\partial A_L} = y_{L-1}^\intercal$$

For $\ell = 2, \dots, L-1$, note that

$$\frac{\partial y_\ell}{\partial b_\ell} = \mathrm{diag}(\sigma'(A_\ell y_{\ell-1} + b_\ell)),$$

$$\frac{\partial y_\ell}{\partial (A_\ell)_{ij}} = \begin{bmatrix} 0 \\ \vdots \\ \sigma'((A_\ell)_{i,:} y_{\ell-1} + (b_\ell)_i)(y_{\ell-1})_j \\ \vdots \\ 0 \end{bmatrix},$$

as p6, hw4.

Then,

$$\frac{\partial y_L}{\partial b_\ell} = \frac{\partial y_L}{\partial y_\ell}\frac{\partial y_\ell}{\partial b_\ell} = \frac{\partial y_L}{\partial y_\ell}\text{diag}(\sigma'(A_\ell y_{\ell-1} + b_\ell)),$$

$$(\frac{\partial y_L}{\partial A_\ell})_{ij} = \frac{\partial y_L}{\partial y_\ell}\frac{\partial y_\ell}{\partial (A_\ell)_{ij}} = (\frac{\partial y_L}{\partial y_\ell})_i \sigma'((A_\ell)_{i,:}y_{\ell-1} + (b_\ell)_i)(y_{\ell-1})_j,$$

$$\Rightarrow \frac{\partial y_L}{\partial A_\ell} = \text{diag}(\sigma'(A_\ell y_{\ell-1} + b_\ell))(\frac{\partial y_L}{\partial y_\ell})^\intercal (y_{\ell-1})^\intercal$$

(iii) Since even when $[A_j = 0$ for some $j \in \{\ell+1, \dots, L-1\}]$ or $[\sigma'(A_j y_{j-1} + b_j) = 0$ for some $j \in \{\ell+1, \dots, L-1\}]$

$$\frac{\partial y_j}{\partial y_{j-1}} = \text{diag}(\sigma'(A_j y_{j-1} + b_j))A_j + I_m = I_m,$$

then,

$$\frac{\partial y_L}{\partial y_i} = \frac{\partial y_L}{\partial y_{L-1}}\frac{\partial y_{L-1}}{\partial y_{L-2}} \cdots \frac{\partial y_{i+1}}{\partial y_i}$$

need not vanish.

Problem 4 : *Split-transform-merge convolutions.*

To calculate trainable parameters, use the formula $(in * k^2 + 1) * out$ where in and out are the number of the input and output channels, respectively and k is the kernel size.

(a) First construction : $(256 + 1)128 + (128 * 9 + 1)128 + (128 + 1)256 = 213,504$

Second construction : $((256 + 1)4 + (4 * 9 + 1)4 + (4 + 1)256)32) = 78,592$

(b) Implementing the split-transform-merge convolutions code below.

```
class STMConvLayer(nn.Module):
    new *
    def __init__(self):
        super(STMConvLayer, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d( in_channels: 256, out_channels: 4, kernel_size: 1),
            nn.ReLU(inplace=True),
            nn.Conv2d( in_channels: 4, out_channels: 4, kernel_size: 3,padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d( in_channels: 4, out_channels: 256, kernel_size: 1),
            nn.ReLU(inplace=True)
        )
        self.conv_layer = nn.ModuleList([self.conv for i in range(32)])

    new *
    def forward(self, x):
        out = self.conv_layer[0](x)
        for i in range(1,32):
            out += self.conv_layer[i](out)
        return out
```

Figure 1: The Split-Transform-Merge Convolutions Code

Problem 5 : *Regularization can mitigate double descent.*

Error per number of parameters plot is below.

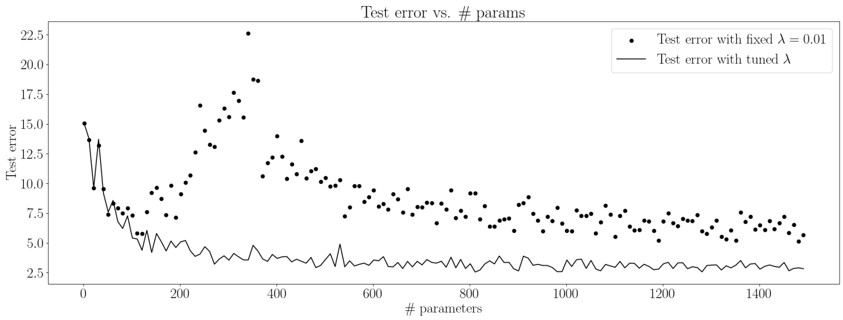

Figure 2: Double Descent