Labs
Machine Learning Course
Fall 2022

EPFL
School of Computer and Communication Sciences
Nicolas Flammarion & Martin Jaggi
www.epfl.ch/labs/mlo/machine-learning-cs-433

Problem Set 08, Nov 10, 2022
(Solution to Theory Question)

Problem 2 (Variance Preserving Weight Initialization for ReLUs):

When training neural networks it is desirable to keep the variance of activations roughly constant across layers. Let's assume we have $y = \mathbf{x}^\top \mathbf{w}$ where $\mathbf{x} = \text{ReLU}(\mathbf{z})$, $\mathbf{z} \in \mathbb{R}^d$ and $\mathbf{w} \in \mathbb{R}^d$. Further assume that all elements $\{w_i\}_{i=1}^d$ and $\{z_i\}_{i=1}^d$ are independent with $w_i \sim \mathcal{N}(0, \sigma)$ and $z_i \sim \mathcal{N}(0, 1)$.

Derive $\text{Var}[y]$ as a function of d and σ i.e. how should we set σ to have $\text{Var}[y] = 1$.

Hint: Remember the law of total expecation i.e. $\text{E}_A[A] = \text{E}_B[\text{E}_A[A|B]]$

Solution:

$$
\begin{aligned}
\text{Var}[y] &= \text{Var}\left[\sum_{i=1}^{d} w_i x_i\right] \\
&= \sum_{i=1}^{d} \text{Var}\left[w_i x_i\right] && \text{independent terms} \\
&= \sum_{i=1}^{d} \text{E}[(w_i x_i)^2] - \text{E}[w_i x_i]^2 && \text{formula for variance} \\
&= \sum_{i=1}^{d} \text{E}[w_i^2]\text{E}[x_i^2] - \text{E}[w_i]^2\text{E}[x_i]^2 && \text{independence} \\
&= \sum_{i=1}^{d} \text{E}[w_i^2]\text{E}[x_i^2] && \text{since } \text{E}[w_i] = 0 \\
&= d\sigma^2 \text{E}[x_i^2] \\
&= d\sigma^2 \text{E}[\text{ReLU}(z_i)^2 | z_i \geq 0] p(z_i \geq 0) + \text{E}[\text{ReLU}(z_i)^2 | z_i < 0] p(z_i < 0) && \text{law of total expectation} \\
&= \frac{1}{2} d\sigma^2 \text{E}[z_i^2] && \text{ReLU is identity or zero} \\
&= \frac{1}{2} d\sigma^2
\end{aligned}
$$

We should thus set $\sigma = \sqrt{2/d}$ to preserve variance (like we did in the notebook exercise).

Problem 3 (Softmax Cross Entropy):

In the notebook exercises we performed multiclass classification using softmax-cross-entropy as our loss. The softmax of a vector $\mathbf{x} = [x_1, ..., x_d]^\top$ is a vector $\mathbf{z} = [z_1, ..., z_d]^\top$ with:

$$z_k = \frac{\exp(x_k)}{\sum_{i=1}^{d} \exp(x_i)} \tag{1}$$

The label y is an integer denoting the target class. To turn y into a probability distribution for use with cross-entropy, we use one-hot encoding:

$$\text{onehot}(y) = \mathbf{y} = [y_1, ..., y_d]^\top \text{ where } y_k = \begin{cases} 1, & \text{if } k = y \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

The cross-entropy is given by:

$$H(\mathbf{y}, \mathbf{z}) = -\sum_{i=1}^{d} y_i \ln(z_i) \tag{3}$$

We ask you to do the following:

1. Equation 1 potentially computes $\exp$ of large positive numbers which is numerically unstable. Modify Eq. 1 to avoid positive numbers in $\exp$. Hint: Use $\max_j(x_j)$.
2. Derive $\frac{\partial H(\mathbf{y},\mathbf{z})}{\partial x_j}$. You may assume that $\mathbf{y}$ is a one-hot vector.
3. What values of x_i minimize the softmax-cross-entropy loss? To avoid complications, practitioners sometimes use a trick called label smoothing where $\mathbf{y}$ is replaced by $\hat{\mathbf{y}} = (1 - \epsilon)\mathbf{y} + \frac{\epsilon}{d}\mathbf{1}$ for some small value e.g. $\epsilon = 0.1$.

Solution:

Part 1:

$$z_k = \frac{\exp(x_k)}{\sum_{i=1}^{d} \exp(x_i)} = \frac{\exp(-\max_j(x_j))}{\exp(-\max_j(x_j))} \frac{\exp(x_k)}{\sum_{i=1}^{d} \exp(x_i)} = \frac{\exp(x_k - \max_j(x_j))}{\sum_{i=1}^{d} \exp(x_i - \max_j(x_j))} \tag{4}$$

Part 2:

$$\begin{aligned} \frac{\partial H(\mathbf{y}, \mathbf{z})}{\partial x_j} &= \frac{\partial H(\mathbf{y}, \mathbf{z})}{\partial z_y} \frac{\partial z_y}{\partial x_j} \\ &= \frac{-1}{z_y} \frac{\partial}{\partial x_j} \frac{\exp(x_y)}{\sum_{i=1}^{d} \exp(x_i)} \end{aligned}$$

For $j = y$ we have:

$$\frac{-1}{z_y} \frac{\partial}{\partial x_j} \frac{\exp(x_j)}{\sum_{i=1}^{d} \exp(x_i)} = -\frac{\sum_{i=1}^{d} \exp(x_i)}{\exp(x_j)} \cdot \frac{\exp(x_j) \sum_{i=1}^{d} \exp(x_i) - \exp(x_j)^2}{(\sum_{i=1}^{d} \exp(x_i))^2} = -\frac{\sum_{i=1}^{d} \exp(x_i) - \exp(x_j)}{\sum_{i=1}^{d} \exp(x_i)} = z_j - 1$$

For $j \neq y$ we have:

$$\frac{-1}{z_y} \frac{\partial}{\partial x_j} \frac{\exp(x_y)}{\sum_{i=1}^{d} \exp(x_i)} = -\frac{\sum_{i=1}^{d} \exp(x_i)}{\exp(x_y)} \cdot \frac{-\exp(x_j) \exp(x_y)}{(\sum_{i=1}^{d} \exp(x_i))^2} = \frac{\exp(x_j)}{\sum_{i=1}^{d} \exp(x_i)} = z_j$$

We can concisely write:

$$\frac{\partial H(\mathbf{y}, \mathbf{z})}{\partial \mathbf{x}} = \mathbf{z} - \mathbf{y} \tag{5}$$

Part 3: The loss is minimized when $x_j \to \begin{cases} \infty \text{ for } j = y \\ -\infty \text{ else} \end{cases}$, label smoothing makes the minimum finite.

Problem 4 (Computation and Memory Requirements of Neural Networks):

Let's consider a fully connected neural network with L layers in total, all of width K. The input is a mini-batch of size $n \times K$. For this exercise we will only consider the matrix multiplications, ignoring biases and activation functions.

- How many multiplications are needed in a forward pass (inference)?
- How many multiplications are needed in a forward + backward pass (training)?
- How much memory is needed for an inference forward pass? The memory needed is the sum of the memory needed for activations and weights. Assume activations are deleted as soon as they are no longer required.
- How much memory is needed for a training forward + backward pass? Note that during training we additionally use the forward activations for the computation of the derivatives.

Solution:

We have a sequence of L matrix multiplications $Y = XW$ where $X \in \mathbb{R}^{n \times K}$ and $W \in \mathbb{R}^{K \times K}$. In the backward pass we compute derivatives w.r.t. the input $\delta_X = \delta_Y W^\top$ and the weights $\delta_W = X^\top \delta_Y$ where δ_Y are the derivatives w.r.t. the output Y. The derivatives have the same shapes as the original variables. The number of multiplications for each of the tree potential matrix multiplications is nK^2. The number of parameters in W is K^2.

During inference we compute L matrix multiplications for a total of nK^2L multiplications. We need to store all parameter matrices and X and Y for a single layer at a time. The memory requirements are therefore $LK^2 + 2nK$ values.

During training we additionally compute the matrix multiplications in the backward pass for a total of $3nK^2L$ multiplications. We store the same parameters but now we also need to store their gradients. Since we need X for the computation of δ_W we also need to store this for each layer. In total our memory requirements are roughly $2LK^2 + LnK$ values.

Note that if n is large, training can require significantly more memory than inference. The optimizer might also keep more values for each weight than the gradient, such as the first and second moments of the gradients. This further increases the memory requirements. The computation cost of training is roughly $3\times$ higher than inference per batch.