EECS 498/598 Deep Learning - Homework 1

February 27, 2019

1. **Fully-connected layer**

   Let $X \in \mathbb{R}^{D_{\text{in}}}$. Consider a dense layer with parameters $W \in \mathbb{R}^{D_{\text{in}} \times D_{\text{out}}}, b \in \mathbb{R}^{D_{\text{out}}}$. The layer outputs a vector $Y \in \mathbb{R}^{D_{\text{out}}}$ where Y is given by $W^T X + b$.

   Compute the partial derivatives $\frac{\partial L}{\partial W}, \frac{\partial L}{\partial b}, \frac{\partial L}{\partial X}$ in terms of $\frac{\partial L}{\partial Y}$.

$$
\begin{aligned}
&Y = W^T X + b \\
&\Rightarrow Y_i = \sum_j W_{ji} X_j + b_i
\end{aligned}
$$

Computing $\frac{\partial L}{\partial W}$:

$$
\begin{aligned}
\frac{\partial L}{\partial W_{mn}} &= \sum_i \frac{\partial L}{\partial Y_i} \frac{\partial Y_i}{\partial W_{mn}} \text{ (Chain rule)} \\
&= \sum_i \frac{\partial L}{\partial Y_i} \frac{\partial Y_i}{\partial W_{mn}} \delta[i = n] \\
&= \frac{\partial L}{\partial Y_n} \frac{\partial Y_n}{\partial W_{mn}} \\
&= \frac{\partial L}{\partial Y_n} X_m \\
\Rightarrow \frac{\partial L}{\partial W} &= X \frac{\partial L}{\partial Y}^T \text{ (Outer product)}
\end{aligned}
$$

Computing $\frac{\partial L}{\partial b}$:

$$
\begin{aligned}
\frac{\partial L}{\partial b_p} &= \frac{\partial L}{\partial Y_p} \frac{\partial Y_p}{\partial b_p} \\
&= \frac{\partial L}{\partial Y_p} \cdot 1 \\
\Rightarrow \frac{\partial Y}{\partial b} &= \frac{\partial L}{\partial Y}
\end{aligned}
$$

Computing $\frac{\partial L}{\partial X}$:

$$\begin{aligned}
\frac{\partial L}{\partial X_p} &= \sum_i \frac{\partial L}{\partial Y_i}\frac{\partial Y_i}{\partial X_p} \\
&= \sum_i \frac{\partial L}{\partial Y_i} W_{pi} \\
\Rightarrow \frac{\partial L}{\partial X} &= W\frac{\partial L}{\partial Y}
\end{aligned}$$

If we have a batch of instances $X^1, ..., X^N$ and the corresponding outputs $Y^1, ..., Y^N$,

$$\begin{aligned}
\frac{\partial L}{\partial W_{mn}} &= \sum_n \sum_i \frac{\partial L}{\partial Y_i^n}\frac{\partial Y_i^n}{\partial W_{mn}} \Rightarrow \frac{\partial L}{\partial W} = \sum_n X^n \left(\frac{\partial L}{\partial Y^n}\right)^T \\
\frac{\partial L}{\partial b_p} &= \sum_n \sum_i \frac{\partial L}{\partial Y_i^n}\frac{\partial Y_i^n}{\partial b_p} \Rightarrow \frac{\partial L}{\partial b} = \sum_n \frac{\partial L}{\partial Y^n} \\
\frac{\partial L}{\partial X^n} &= W\frac{\partial L}{\partial Y^n}
\end{aligned}$$

2. **ReLU**

   Let X be a tensor and $Y = \mathrm{ReLU}(X)$. Express $\frac{\partial L}{\partial X}$ in terms of $\frac{\partial L}{\partial Y}$.

   For a scalar x,

$$y = \mathrm{ReLU}(x) = \begin{cases} 0 \text{ if } x < 0 \\ x \text{ if } x \geq 0 \end{cases}$$

$$\Rightarrow \frac{\partial y}{\partial x} = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x > 0 \end{cases} = \delta[x > 0]$$

$$\Rightarrow \frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x} = \frac{\partial L}{\partial y}\delta[x > 0]$$

   Since the ReLU operation is element-wise, we can generalize this to tensors X and Y,

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \odot \delta[X > 0]$$

   where $\odot$ represents elementwise product.

3. **Dropout**

   Given a dropout mask M, let $Y = X \odot M$, where $\odot$ represents element-wise multiplication. Express $\frac{\partial L}{\partial X}$ in terms of $\frac{\partial L}{\partial Y}$.

   For scalars x, y,

$$y = xm \Rightarrow \frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x} = m\frac{\partial L}{\partial y} \tag{1}$$

   Since dropout is an elementwise operation, we can generalize the above to multi-dimensional tensors X, Y

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \odot M \tag{2}$$

4. **Batch Normalization** Let X be a 2D tensor of input instances where X_i represents the ith example vector. The output of the layer Y is given by $Y_i = \gamma(\frac{X_i-\mu}{\sigma}) + \beta$ where $\mu = \frac{1}{n}\sum_j X_j$ and $\sigma = \sqrt{\frac{1}{n}\sum_j (X_j-\mu)^2 + \epsilon}$. β and γ are constants that represent a running average of the sample mean and variance, respectively. Note that $\mu, \sigma, \gamma, \beta$ are all vectors.
Derive expressions for $\frac{\partial \mu}{\partial X_i}$ and $\frac{\partial \sigma}{\partial X_i}$.
Based on this, derive an expression for $\frac{\partial L}{\partial X_i}$ in terms of $\frac{\partial L}{\partial Y}$.

Since Batch Normalization normalizes every feature independently, in the following derivation we assume that $X_i, Y_i, \mu, \sigma, \beta, \gamma$ are all scalars.

Compute $\frac{\partial \mu}{\partial X_i}$:

$$
\begin{aligned}
\mu &= \frac{1}{n}\sum_j X_j \\
\frac{\partial \mu}{\partial X_i} &= \frac{1}{n}
\end{aligned}
$$

Compute $\frac{\partial \sigma}{\partial X_i}$:

$$
\begin{aligned}
\sigma &= \sqrt{\frac{1}{n}\sum_j (X_j-\mu)^2 + \epsilon} \\
\frac{\partial \sigma}{\partial X_i} &= \frac{1}{2\sigma}\frac{\partial[\frac{1}{n}\sum_j (X_j-\mu)^2 + \epsilon]}{\partial X_i} \\
&= \frac{1}{2\sigma n}\sum_j 2(X_j-\mu)\frac{\partial (X_j-\mu)}{\partial X_i} \\
&= \frac{1}{\sigma n}\sum_j (X_j-\mu)\left(\delta[j=i] - \frac{1}{n}\right) \\
&= \frac{1}{\sigma n}((X_i-\mu) - \frac{1}{n}\sum_j (X_j-\mu)) \\
&= \frac{X_i-\mu}{\sigma n}
\end{aligned}
$$

$$
\begin{aligned}
Y_i &= \gamma\left(\frac{X_i-\mu}{\sigma}\right) + \beta \\
\Rightarrow \frac{\partial L}{\partial X_i} &= \sum_j \frac{\partial L}{\partial Y_j}\frac{\partial Y_j}{\partial X_i} \\
&= \frac{\gamma}{\sigma^2}\sum_j \frac{\partial L}{\partial Y_j}\left(\sigma\frac{\partial (X_j-\mu)}{\partial X_i} - (X_j-\mu)\frac{\partial \sigma}{\partial X_i}\right) \\
&= \frac{\gamma}{\sigma^2}\sum_j \frac{\partial L}{\partial Y_j}\left(\sigma\left(\delta[j=i] - \frac{1}{n}\right) - \frac{(X_j-\mu)(X_i-\mu)}{n\sigma}\right)
\end{aligned}
$$

Going back to the original setting where X_i, Y_i are vectors, the above result is applicable to each feature independently. $\frac{\partial L}{\partial X_i}$ can hence be computed by performing element-wise arithmetic operations in the above expression.

5. **Convolution**

You will implement a valid convolution layer for this question. Assume the input to the layer is given by $X \in \mathbb{R}^{N\times C\times H\times W}$. Consider a convolutional kernel $W \in \mathbb{R}^{F\times C\times H'\times W'}$. The output of the valid convolution layer is given by $Y \in \mathbb{R}^{N\times F\times H''\times W''}$. The layer produces F output feature maps defined by $Y_{n,f} = \sum_c X_{n,c} *_{\text{valid}} \overline{W_{f,c}}$, where $\overline{W_{f,c}}$ represents the flipped kernel (i.e., $\overline{K}_{i,j}$ is defined as $\overline{K}_{i,j} = K_{H'+1-i,W'+1-j}$). Note that $Y_{n,f} = \sum_c X_{n,c} *_{\text{valid}} \overline{W_{f,c}} = \sum_c X_{n,c} *_{\text{filt}} W_{f,c}$.

Show that

$$\frac{\partial L}{\partial X_{n,c}} = \sum_f W_{f,c} *_{\text{full}} \left(\frac{\partial L}{\partial Y_{n,f}}\right)$$

and

$$\frac{\partial L}{\partial W_{f,c}} = \sum_n X_{n,c} *_{\text{filt}} \left(\frac{\partial L}{\partial Y_{n,f}}\right)$$

Solution:

$$\begin{aligned}
\frac{\partial L}{\partial X_{n,c,i,j}} &= \sum_f \sum_{m',n'} \frac{\partial Y_{n,f,m',n'}}{\partial X_{n,c,i,j}} \frac{\partial L}{\partial Y_{n,f,m',n'}}, \\
&= \underbrace{\sum_f \sum_{m',n'} \frac{\partial \sum_{c'} \sum_{m'',n''} X_{n,c',m'+m''-1,n'+n''-1} W_{f,c',m'',n''}}{\partial X_{n,c,i,j}} \frac{\partial L}{\partial Y_{n,f,m',n'}},}_{*_{filt} \text{ definition of } Y_{n,f}} \\
&= \underbrace{\sum_f \sum_{m',n'} \frac{\partial \sum_{c'} \sum_{i',j'} X_{n,c',i',j'} W_{f,c',i'-m'+1,j'-n'+1}}{\partial X_{n,c,i,j}} \frac{\partial L}{\partial Y_{n,f,m',n'}},}_{i'=m'+m''-1,j'=n'+n''-1} \\
&= \sum_f \sum_{m',n'} W_{f,c,i-m'+1,j-n'+1} \frac{\partial L}{\partial Y_{n,f,m',n'}}, \\
&= \underbrace{\sum_f \sum_{m''',n'''} W_{f,c,m''',n'''} \frac{\partial L}{\partial Y_{n,f,i-m'''+1,j-n'''+1}},}_{m'''=i-m'+1,n'''=j-n'+1}
\end{aligned}$$

Now, using the definition of full convolution, we have

$$\begin{aligned}
\frac{\partial L}{\partial X_{n,c,i,j}} &= \sum_f \sum_{m''',n'''} W_{f,c,m''',n'''} \frac{\partial L}{\partial Y_{n,f,i-m'''+1,j-n'''+1}} \\
&= \sum_f \left(W_{f,c} *_{\text{full}} \left(\frac{\partial L}{\partial Y_{n,f}}\right)\right)_{i,j}
\end{aligned}$$

Note: You can skip step 5 by noticing that $(\mathbf{a} *_{\text{full}} \mathbf{b})_{i,j} = (\mathbf{b} *_{\text{full}} \mathbf{a})_{i,j}$.

Similarly,

$$\begin{aligned}
\frac{\partial L}{\partial W_{f,c,i,j}} &= \sum_n \sum_{m',n'} \frac{\partial Y_{n,f,m',n'}}{\partial W_{f,c,i,j}} \frac{\partial L}{\partial Y_{n,f,m',n'}}, \\
&= \underbrace{\sum_n \sum_{m',n'} \frac{\partial \sum_{c'} \sum_{m'',n''} X_{n,c',m'+m''-1,n'+n''-1} W_{f,c',m'',n''}}{\partial W_{f,c,i,j}} \frac{\partial L}{\partial Y_{n,f,m',n'}}}_{*_{filt} \text{ definition of } Y_{n,f}}, \\
&= \sum_n \sum_{m',n'} X_{n,c,m'+i-1,n'+j-1} \frac{\partial L}{\partial Y_{n,f,m',n'}}, \qquad (3)
\end{aligned}$$

Now, using the definition of filtering convolution, we have

$$\begin{aligned}
\frac{\partial L}{\partial W_{f,c,i,j}} &= \sum_n \sum_{m',n'} X_{n,c,m'+i-1,n'+j-1} \frac{\partial L}{\partial Y_{n,f,m',n'}} \\
&= \sum_n \left(X_{n,c} *_{\text{filt}} \left(\frac{\partial L}{\partial Y_{n,f}} \right) \right)_{i,j}
\end{aligned}$$

1 [6 points] Short answer questions

1. Describe the main difficulty in training deep neural networks with the sigmoid non-linearity.

   When sigmoid activations saturate, there is little/no gradient, which makes it difficult to escape this region during training. The gradient of the sigmoid activation σ is given by $\sigma(1-\sigma)$, which is small when either the input is too small or too large.

2. Consider a linear layer W. During training, dropout with probability p is applied to the layer input x and the output is given by $y = W(x \odot m)$ where m represents the dropout mask. How would you use this layer during inference to reduce the train/test mismatch (i.e., what is the input/output relationship).

   During training, activations are dropped with a keep probability of p, while all activations are retained during inference. As shown in figure 1, the weights (or the layer input) needs to be scaled by p at test time so that the output has the same expected value during training and inference.

3. If the training loss goes down quickly and then diverges during training, what hyperparameters would you modify ?

   This is an indication that the learning rate is too big. The learning rate should be reduced to prevent gradient steps from overshooting.

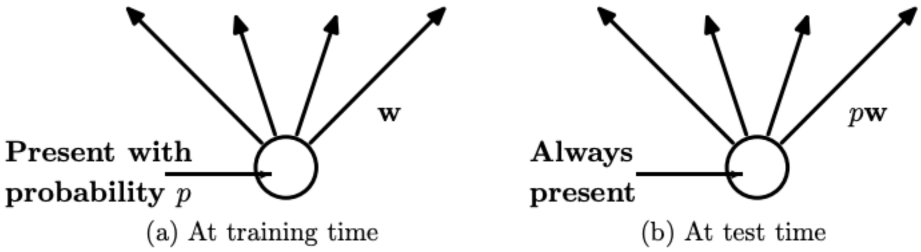

Figure 1: Left: A unit at training time that is present with probability p and is connected to units in the next layer with weights w. Right: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time. Figure source: Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting.”, JMLR 2014.