Akhil Chilumuru
UIN: 633002192
Sep 28, 2022

CSCE 636: Deep Learning
Assignment 2

1 Answers to the non-programming part

1. **Exercise 7.7 (e-Chap:7-11) in the book "Learning from Data".**
First of all, let's compute the derivative of tanh function:

$$\begin{aligned}
\frac{\partial tanh(x)}{\partial x} &= \frac{\partial}{\partial x}\left(\frac{e^x - e^{-x}}{e^x + e^{-x}}\right) \\
&= \frac{\partial(e^x - e^{-x})}{\partial x}\frac{1}{e^x + e^{-x}} + \left(e^x - e^{-x}\right)\frac{\partial}{\partial x}(\frac{1}{e^x + e^{-x}}) \\
&= \left(e^x + e^{-x}\right)\frac{1}{e^x + e^{-x}} + \left(e^x - e^{-x}\right)\frac{-1}{\left(e^x + e^{-x}\right)^2}\frac{\partial}{\partial x}\left(e^x + e^{-x}\right) \\
&= 1 + \left(e^x - e^{-x}\right)\frac{-1}{\left(e^x + e^{-x}\right)^2}\left(e^x - e^{-x}\right) \\
&= 1 - \frac{\left(e^x - e^{-x}\right)^2}{\left(e^x + e^{-x}\right)^2} \\
&= 1 - \left(\frac{e^x - e^{-x}}{e^x + e^{-x}}\right)^2 \\
&= 1 - \left(tanh(x)\right)^2
\end{aligned}$$

Using this result,

$$\begin{aligned}
\boldsymbol{\nabla} E_{in}(w) &= \frac{\partial}{\partial \mathbf{w}}\left(\frac{1}{N}\sum_{n=1}^{N}\left(tanh(\mathbf{w^T x_n}) - y_n\right)^2\right) \\
&= \frac{1}{N}\sum_{n=1}^{N}\frac{\partial}{\partial \mathbf{w}}\left(tanh(\mathbf{w^T x_n}) - y_n\right)^2 \\
&= \frac{1}{N}\sum_{n=1}^{N}2\left(tanh(\mathbf{w^T x_n}) - y_n\right)\frac{\partial}{\partial \mathbf{w}}\left(tanh(\mathbf{w^T x_n})\right) \\
&= \frac{2}{N}\sum_{n=1}^{N}\left(tanh(\mathbf{w^T x_n}) - y_n\right)\left(1 - (tanh(\mathbf{w^T x_n}))^2\right)\frac{\partial \mathbf{w^T x_n}}{\partial \mathbf{w}} \\
&= \frac{2}{N}\sum_{n=1}^{N}\left(tanh(\mathbf{w^T x_n}) - y_n\right)\left(1 - tanh^2(\mathbf{w^T x_n})\right)\mathbf{x_n}
\end{aligned}$$

Therefore, when $\mathbf{w} \to \infty$, $tanh(\mathbf{w^T x}) \to 1$ and consequently $\boldsymbol{\nabla} E_{in}(w) \to 0$

This means the gradient is not propagated backward. This is why it is difficult to optimize the perceptron because the weights would not get updated.

2. **Repeat the computations in Example 7.1 for the case when the output transformation is the identity.**

Given input and weight matrices are:

$$W^{(1)} = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}; W^{(2)} = \begin{bmatrix} 0.2 \\ 1 \\ -3 \end{bmatrix}; W^{(3)} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

For the data point x=2, y=1 forward propagation gives:

$x^{(0)}$	$s^{(1)}$	$x^{(1)}$	$s^{(2)}$	$x^{(2)}$	$s^{(3)}$	$x^{(3)}$
$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$	$(W^{(1)})^T x^{(0)} = \begin{bmatrix} 0.7 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0.6 \\ 0.76 \end{bmatrix}$	$(W^{(2)})^T x^{(1)} = \begin{bmatrix} -1.48 \end{bmatrix}$	$\begin{bmatrix} 1 \\ -0.90 \end{bmatrix}$	$(W^{(3)})^T x^{(2)} = \begin{bmatrix} -0.8 \end{bmatrix}$	-0.8

Now, we know that $\delta^{(L)} = 2(x^{(L)} - y)\theta'(s^{(L)})$
However $\theta'(s^{(L)}) = 1$ here because output transformation is linear
Therefore, $\delta^{(3)} = 2(x^{(3)} - y) = 2(-0.8 - 1) = -3.6$
Now we just back propagate using the equation $\delta^{(l)} = \theta'(s^{(l)}) \otimes [W^{(l+1)}\delta^{(l+1)}]_1^{d(l)}$
where $\theta'(s^{(l)}) = [1 - x^{(l)} \otimes x^{(l)}]_1^{d^{(l)}}$ because of the tanh activation
$\theta'(s^{(2)}) = 1 - 0.9^2 = 0.19$

$$\delta^{(2)} = [0.19] \otimes [\begin{bmatrix} 1 \\ 2 \end{bmatrix} * -3.6]_1^1 = 0.19 * 2 * -3.6 = -1.368$$

$$\theta'(s^{(1)}) = 1 - \begin{bmatrix} 0.6^2 \\ 0.76^2 \end{bmatrix} = \begin{bmatrix} 0.64 \\ 0.43 \end{bmatrix}$$

$$\delta^{(1)} = \begin{bmatrix} 0.64 \\ 0.43 \end{bmatrix} \otimes \begin{bmatrix} 0.2 \\ 1 \\ -3 \end{bmatrix}_1^2 * -1.368 = \begin{bmatrix} 0.64 \\ 0.43 \end{bmatrix} \otimes \begin{bmatrix} -1.368 \\ 4.104 \end{bmatrix} = \begin{bmatrix} -0.87 \\ 1.76 \end{bmatrix}$$

In summary, these are the sensitivity δ values:

$\delta^{(3)}$	$\delta^{(2)}$	$\delta^{(1)}$
-3.6	-1.368	$\begin{bmatrix} -0.87 \\ 1.76 \end{bmatrix}$

Now, we compute gradients with the equation $\frac{\partial e}{\partial W^{(l)}} = x^{(l-1)}(\delta^{(l)})^T$

$\frac{\partial e}{\partial W^{(1)}}$	$\frac{\partial e}{\partial W^{(2)}}$	$\frac{\partial e}{\partial W^{(3)}}$
$x^{(0)}(\delta^{(1)})^T = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \begin{bmatrix} -0.87 & 1.76 \end{bmatrix} = \begin{bmatrix} -0.87 & 1.76 \\ -1.74 & 3.52 \end{bmatrix}$	$x^{(1)}(\delta^{(2)})^T = \begin{bmatrix} 1 \\ 0.6 \\ 0.76 \end{bmatrix} \begin{bmatrix} -1.368 \end{bmatrix} = \begin{bmatrix} -1.368 \\ -0.82 \\ -1.039 \end{bmatrix}$	$x^{(2)}(\delta^{(3)})^T = \begin{bmatrix} 1 \\ -0.9 \end{bmatrix} \begin{bmatrix} -3.6 \end{bmatrix} = \begin{bmatrix} -3.6 \\ 3.24 \end{bmatrix}$

3. **Consider the standard residual block and the bottleneck block in the case where inputs and outputs have the same dimension. Compute the number of training parameters when the dimension of inputs and outputs is 128 × 16 × 16 × 32. For the bottleneck block, compute the number of training parameters when the dimension of inputs and outputs is 128 × 16 × 16 × 128. Compare the two results and explain the advantages and disadvantages of the bottleneck block.**

   While calculating parameters, we can ignore the batch size, so let's drop 128 while writing dimensions.

   (a) For the Standard residual block, input dimensions = 16 x 16 x 32:

Layer	trainable parameters	non-trainable parameters
conv1 3x3,32	3*3*32*32 + 32=9248	0
batchnorm1 32 feature maps	2*32=64	2*32=64
conv2 3x3,32	3*3*32*32 + 32=9248	0
batchnorm2 32 feature maps	2*32=64	2*32=64
Total	18624	128

   (b) For the Bottleneck residual block, input dimensions = 16 x 16 x 128:

Layer	trainable parameters	non-trainable parameters
conv1 1x1,32	1*1*128*32 + 32=4128	0
batchnorm1 32 feature maps	2*32=64	2*32=64
conv2 3x3,32	3*3*32*32 + 32=9248	0
batchnorm2 32 feature maps	2*32=64	2*32=64
conv3 1x1,128	1*1*32*128 + 128=4224	0
batchnorm3 128 feature maps	2*128=256	2*128=256
Total	17984	384

   As we can see bottleneck design has a lower number of parameters despite the bottleneck block having more layers and more input feature maps. This means that if we have the same number of input feature maps, the bottleneck block will have a significantly low number of parameters. This is because we have 1x1 convolutions that reduce dimensions in the bottleneck block. So the advantage of using a bottleneck block is that since we have lower parameters, we may avoid overfitting and increase the computational efficiency. Another advantage is that since we have more layers, generally that means that we have introduced more non-linearity into the model and therefore it could fit more complex data.

   However, to quote the resnet paper *"The parameter-free identity shortcuts are particularly important for the bottleneck architectures. If the identity shortcut in is replaced with projection, one can show that the time complexity and model size are doubled, as the shortcut is connected to the two high-dimensional ends"*. This means we must have identity skip connections in bottleneck design, otherwise, the parametric and computational efficiency that we have shown earlier won't hold. This is a disadvantage of using a bottleneck design. Also, in the standard residual block, we have two 3x3 convolutions stacked together and the receptive field will be equivalent to a 5x5 convolution. But in bottleneck design, 1x1 is stacked with 3x3. So the receptive field might be low. Therefore there could be a loss of information. This is another disadvantage.

4. **Using batch normalization in training requires computing the mean and variance of a tensor.**

   (a) Shape of tensor x is N x C. The shape of both mean and variance in batch normalization is 1 x C.

   (b) If the shape of the output of a convolution layer is N x H x W x C, then the shape of both mean and variance computed in batch normalization will be 1 x 1 x 1 x C.

5. **We investigate the back-propagation of the convolution using a simple example. In this problem, we focus on the convolution operation without any normalization and activation function.**

(a) If we re-write the original equation in the form of $\tilde{Y} = A\tilde{X}$, we have:

$$
\begin{aligned}
y^{11} &= \begin{bmatrix} w_1^{11} & w_2^{11} & w_3^{11} \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \end{bmatrix} + \begin{bmatrix} w_1^{21} & w_2^{21} & w_3^{21} \end{bmatrix} \begin{bmatrix} x_{21} \\ x_{22} \\ x_{23} \end{bmatrix} \\
&= \begin{bmatrix} w_1^{11} & w_2^{11} & w_3^{11} & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & w_1^{21} & w_2^{21} & w_3^{21} & 0 \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \end{bmatrix} \\
&= \begin{bmatrix} w_1^{11} & w_2^{11} & w_3^{11} & 0 & w_1^{21} & w_2^{21} & w_3^{21} & 0 \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \end{bmatrix}
\end{aligned}
$$

if we do similar operations on the other outputs we will have:

$$
\tilde{Y} = \begin{bmatrix} w_1^{11} & w_2^{11} & w_3^{11} & 0 & w_1^{21} & w_2^{21} & w_3^{21} & 0 \\ 0 & w_1^{11} & w_2^{11} & w_3^{11} & 0 & w_1^{21} & w_2^{21} & w_3^{21} \\ w_1^{12} & w_2^{12} & w_3^{12} & 0 & w_1^{22} & w_2^{22} & w_3^{22} & 0 \\ 0 & w_1^{12} & w_2^{12} & w_3^{12} & 0 & w_1^{22} & w_2^{22} & w_3^{22} \end{bmatrix} \tilde{X}
$$

(b) Let us write gradient for x_{11} and we can generalize from it:

$$
\begin{aligned}
\frac{\partial L}{\partial x_{11}} &= \frac{\partial L}{\partial y_{11}} \frac{\partial y_{11}}{\partial x_{11}} + \frac{\partial L}{\partial y_{12}} \frac{\partial y_{12}}{\partial x_{11}} + \frac{\partial L}{\partial y_{21}} \frac{\partial y_{21}}{\partial x_{11}} + \frac{\partial L}{\partial y_{22}} \frac{\partial y_{22}}{\partial x_{11}} \\
&= \begin{bmatrix} \frac{\partial y_{11}}{\partial x_{11}} & \frac{\partial y_{12}}{\partial x_{11}} & \frac{\partial y_{21}}{\partial x_{11}} & \frac{\partial y_{22}}{\partial x_{11}} \end{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial y_{11}} \\ \frac{\partial L}{\partial y_{12}} \\ \frac{\partial L}{\partial y_{21}} \\ \frac{\partial L}{\partial y_{22}} \end{bmatrix} \\
&= \begin{bmatrix} w_1^{11} & 0 & w_1^{12} & 0 \end{bmatrix} \frac{\partial L}{\partial \tilde{Y}}
\end{aligned}
$$

If we write out the full matrix we would get the following:

$$
\frac{\partial L}{\partial \tilde{X}} = \begin{bmatrix} w_1^{11} & 0 & w_1^{12} & 0 \\ w_2^{11} & w_1^{11} & w_2^{12} & w_1^{12} \\ w_3^{11} & w_2^{11} & w_3^{12} & w_2^{12} \\ 0 & w_3^{11} & 0 & w_3^{12} \\ w_1^{21} & 0 & w_1^{22} & 0 \\ w_2^{21} & w_1^{21} & w_2^{22} & w_1^{22} \\ w_3^{21} & w_2^{21} & w_3^{22} & w_2^{22} \\ 0 & w_3^{21} & 0 & w_3^{22} \end{bmatrix} \frac{\partial L}{\partial \tilde{Y}}
$$

We can observe that $B = A^T$

(c) Yes, it is possible to imagine $\frac{\partial L}{\partial X} = B\frac{\partial L}{\partial \tilde{Y}}$ as a convolution on $\frac{\partial L}{\partial Y}$ to obtain $\frac{\partial L}{\partial X}$

Since, it is kind of a upsampling operation (2x2 to 4x2), we need to pad $\frac{\partial L}{\partial Y}$ with zeros first. For simplicity, let us denote $\frac{\partial L}{\partial Y}$ as $\boldsymbol{\nabla}\mathrm{Y}$:

convolution input=$\begin{bmatrix} 0 & 0 & \boldsymbol{\nabla}y^{11} & \boldsymbol{\nabla}y^{12} & 0 & 0 \\ 0 & 0 & \boldsymbol{\nabla}y^{21} & \boldsymbol{\nabla}y^{22} & 0 & 0 \end{bmatrix}$

Now, the 1^{st} kernel would be=$\begin{bmatrix} w_3^{11} & w_2^{11} & w_1^{11} \\ w_3^{12} & w_2^{12} & w_1^{12} \end{bmatrix}$

And, the 2^{nd} kernel would be=$\begin{bmatrix} w_3^{21} & w_2^{21} & w_1^{21} \\ w_3^{22} & w_2^{22} & w_1^{22} \end{bmatrix}$

The output would be= $\begin{bmatrix} \boldsymbol{\nabla}x^{11} & \boldsymbol{\nabla}x^{12} & \boldsymbol{\nabla}x^{13} & \boldsymbol{\nabla}x^{14} \\ \boldsymbol{\nabla}x^{21} & \boldsymbol{\nabla}x^{22} & \boldsymbol{\nabla}x^{23} & \boldsymbol{\nabla}x^{24} \end{bmatrix}$

Which is the same as $\frac{\partial L}{\partial X}$

If we want to write these weights in the notation that the question uses, then

$$W^{ji} = \begin{bmatrix} w_3^{ij} & w_2^{ij} & w_1^{ij} \end{bmatrix}$$

where W^{ji} scans the j^{th} channel in a padded $\frac{\partial L}{\partial Y}$ and produces i^{th} channel in $\frac{\partial L}{\partial X}$

Therefore, it is possible to view the gradient computation as a convolution operation.