Apurva Mandalika
334003575

CSCE 636: Deep Learning (Spring 2024)

Assignment #2: Written Part

1)

1. $\frac{\partial \tanh(x)}{\partial x} = \frac{\partial}{\partial x}\left(\frac{e^x - e^{-x}}{e^x + e^{-x}}\right)$

$= \frac{\partial (e^x - e^{-x})}{\partial x} \cdot \frac{1}{e^x + e^{-x}} + (e^x - e^{-x}) \frac{\partial}{\partial x}\left(\frac{1}{e^x + e^{-x}}\right)$

$= (e^x + e^{-x}) \frac{1}{e^x + e^{-x}} + (e^x - e^{-x})\left(\frac{-1}{(e^x + e^{-x})^2}\right) \frac{\partial (e^x + e^{-x})}{\partial x}$

$= 1 + (e^x - e^{-x}) \cdot \left(\frac{-1}{(e^x + e^{-x})^2}\right)(e^x - e^{-x})$

$= 1 - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2}$

$= 1 - \left(\frac{e^x - e^{-x}}{e^x + e^{-x}}\right)^2$

$= 1 - (\tanh(x))^2$

Using this,

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

$= \frac{1}{N} \sum_{n=1}^{N} \frac{\partial}{\partial w} (\tanh(w^T x_n) - y_n)^2$

$= \frac{1}{N} \sum_{n=1}^{N} 2(\tanh(w^T x_n) - y_n) \frac{\partial}{\partial w} (\tanh(w^T x_n))$

$= \frac{2}{N} \sum_{n=1}^{N} (\tanh(w^T x_n) - y_n)(1 - (\tanh(w^T x_n))^2) \cdot \frac{\partial w^T x_n}{\partial w}$

$= \frac{2}{N} \sum_{n=1}^{N} (\tanh(w^T x_n) - y_n)(1 - \tanh^2(w^T x_n)) x_n$

If w -> ∞, tanh(wTx) -> 1 and consequently ∇Ein(w) -> 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)

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

For $x=2, y=1$ forward propagation gives:

$x^{(0)}: \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ $\quad s^{(1)}: (w^{(1)})^T x^{(0)} = \begin{bmatrix} 0.7 \\ 1 \end{bmatrix}$

$x^{(1)}: \begin{bmatrix} 1 \\ 0.6 \\ 0.76 \end{bmatrix}$ $\quad s^{(2)}: (w^{(2)})^T x^{(1)} = [-1.48]$

$x^{(2)}: \begin{bmatrix} 1 \\ -0.9 \end{bmatrix}$ $\quad s^{(3)}: (w^{(3)})^T x^{(2)} = [-0.8]$

$x^{(3)} = -0.8$

$\delta^{(L)} = 2\left(x^{(L)} - y\right)\theta'(s^{(L)})$

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 back propagate using equation

$$\delta^{(l)} = \theta'(s^{(l)}) \otimes \left[w^{(l+1)} \delta^{(l+1)} \right]_1^{d(l)}$$

where $\theta'(s^{(l)}) = \left[1 - x^{(l)} \otimes x^{(l)} \right]_1^{d(l)}$

because of tanh activation.

$\theta'(s^{(2)}) = 1 - (0.9)^2 = 0.19$

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

$= -1.368$

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

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

$$= \begin{bmatrix} 0.64 \\ 0.43 \end{bmatrix} \otimes \begin{bmatrix} -1.368 \\ 4.104 \end{bmatrix} = \begin{bmatrix} -0.87 \\ 1.76 \end{bmatrix}$$

sensitivity values are,

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

Compute gradients with the equation,

$$\frac{\partial e}{\partial w^{(l)}} = x^{(l-1)} (\delta^{(l)})^T$$

$$\frac{\partial e}{\partial w^{(1)}} : x^{(0)} (\delta^{(1)})^T = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \begin{bmatrix} -0.87 & 1.76 \end{bmatrix}$$

$$= \begin{bmatrix} -0.87 & 1.76 \\ -1.74 & 3.52 \end{bmatrix}$$

$$\frac{\partial e}{\partial w^{(2)}} : x^{(1)} (\delta^{(2)})^T = \begin{bmatrix} 1 \\ 0.6 \\ 0.76 \end{bmatrix} \begin{bmatrix} -1.368 \end{bmatrix}$$

$$= \begin{bmatrix} -1.368 \\ -0.82 \\ -1.039 \end{bmatrix}$$

$$\frac{\partial e}{\partial w^{(3)}} : x^{(2)} (\delta^{(3)})^T = \begin{bmatrix} 1 \\ -0.9 \end{bmatrix} \begin{bmatrix} -3.6 \end{bmatrix}$$

$$= \begin{bmatrix} -3.6 \\ 3.24 \end{bmatrix}$$

3) For the Standard residual block, the quantity of trainable parameters is,
((3 x 3 x 32 x 32) + 32) **[C]** + (2 x 32) **[BN]** + ((3 x 3 x 32 x 32) + 32) **[C]** + (2 x 32) **[BN]** = 18,624

For the bottleneck block, the quantity of trainable parameters is,
((1 x 1 x 128 x 32) + 32) **[C]** + (32 x 2) **[BN]** + ((3 x 3 x 32 x 32) + 32) **[C]** + (32 x 2) **[BN]** + ((1 x 1 x 32 x 128) + 128) **[C]** + (128 x 2) **[BN]** = 17,984

where, **[C]** means conv and **[BN]** means Batch Norm

Advantage of the bottleneck:

- For the same number of input feature maps, the bottleneck block will have a significantly low number of parameters
- Since we have lower parameters, we may avoid overfitting and increase the computational efficiency
- Since we have more layers, it could fit more complex data.

Disadvantage of the bottleneck:

- In bottleneck design, 1x1 is stacked with 3x3. So, the receptive field might be low. Therefore, there could be a loss of information.
- Outputs from the bottleneck block has more channels, resulting the increasing of the number of parameters in the subsequent layers.
- We must have identity shortcut in bottleneck design, otherwise, the parametric and computational efficiency won't hold.

4)

a) When x is the output of a fully-connected layer with shape N * C, where:

- N is the batch size, and
- C is the number of output nodes.

The mean and variance are computed as:

Mean: The mean is computed along the batch dimension N for each output node. Since there is only one spatial dimension (no height or width), we only calculate the mean across the batch dimension. Therefore, the shape of the **mean** tensor would be **(1 * C)**. Each element of the mean tensor represents the mean value of each output node across the batch.

Variance: Similarly, the variance is computed along the batch dimension N for each output node. The shape of the **variance** tensor would also be **(1 * C)**, where each element corresponds to the variance of each output node across the batch.

So, for the output of a fully-connected layer, the **mean and variance** tensors computed in batch normalization **have shapes (1 * C),** where C is the number of output nodes. These tensors capture the statistics of each output node across the entire batch, enabling batch normalization to normalize each output node independently based on its mean and variance.

b) When applying batch normalization to the output of a 2D convolutional layer, the input tensor x typically has the shape N * H * W * C, where:
N is the batch size, H is the height of the feature map, W is the width of the feature map, and C is the number of channels.

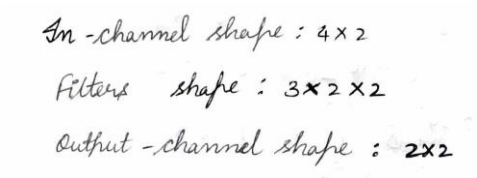
Now, let us consider the computation of mean and variance in batch normalization:

Mean: To compute the mean, we take the average value of each channel across the batch and spatial dimensions. In other words, we calculate the mean for each channel independently across all samples and spatial positions. Therefore, the **mean** is computed along the batch dimension N and spatial dimensions H and W, resulting in the shape **(1 * 1 * 1 * C).** Each element of the mean tensor corresponds to the mean value of the corresponding channel across all samples and spatial locations.

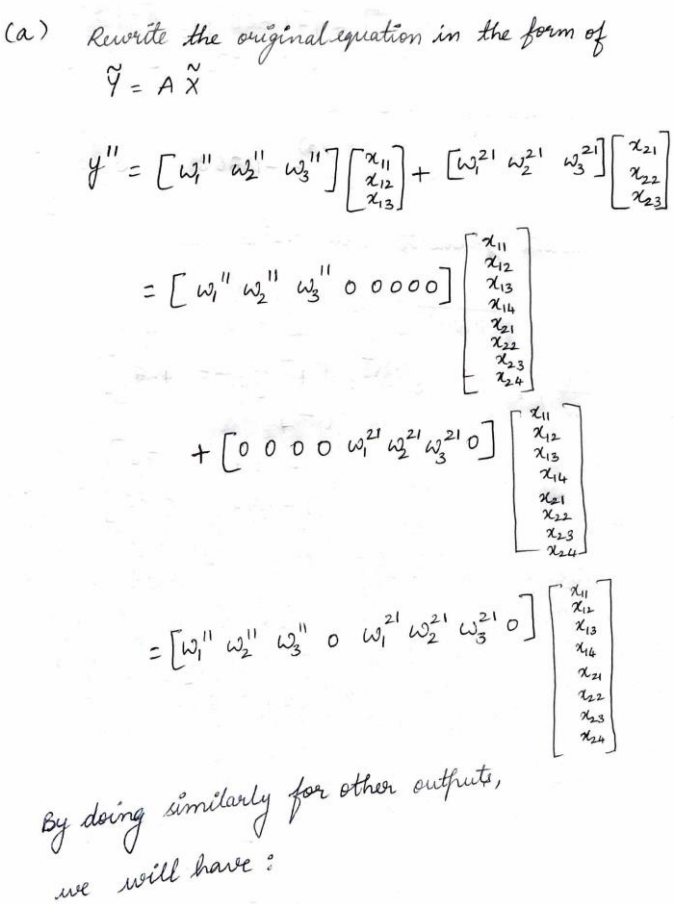
Variance: Similar to the mean, the variance is computed along the batch dimension N and spatial dimensions H and W for each channel independently. Thus, the shape of the **variance** tensor is also **(1 * 1 * 1 * C),** with each element representing the variance of the corresponding channel across the batch and spatial dimensions.

So, for the output of a 2D convolutional layer, the **mean and variance** tensors computed in batch normalization have shapes **(1 * 1 * 1 * C)**, where C is the number of channels. These tensors capture the statistics of each channel across the entire batch and spatial dimensions, allowing batch normalization to normalize each channel independently based on its mean and variance.

5)



a)



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

b)

(b) Gradient for x_{11}

$$\frac{\partial L}{\partial x_{11}} = \frac{\partial L}{\partial y_{11}} \cdot \frac{\partial y_{11}}{\partial x_{11}} + \frac{\partial L}{\partial y_{12}} \cdot \frac{\partial y_{12}}{\partial x_{11}} + \frac{\partial L}{\partial y_{21}} \cdot \frac{\partial y_{21}}{\partial x_{11}} + \frac{\partial L}{\partial y_{22}} \cdot \frac{\partial y_{22}}{\partial x_{11}}$$

$$= \begin{bmatrix} \frac{\partial y_{11}}{\partial x_{11}} & \frac{\partial y_{12}}{\partial x_{11}} & \frac{\partial y_{21}}{\partial x_{11}} & \frac{\partial y_{22}}{\partial x_{11}} \end{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial y_{11}} \\ \frac{\partial L}{\partial y_{12}} \\ \frac{\partial L}{\partial y_{21}} \\ \frac{\partial L}{\partial y_{22}} \end{bmatrix}$$

$$= \begin{bmatrix} w_1^{11} & 0 & w_1^{12} & 0 \end{bmatrix} \frac{\partial L}{\partial Y}.$$

Similarly, if we do for others too, we get.

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

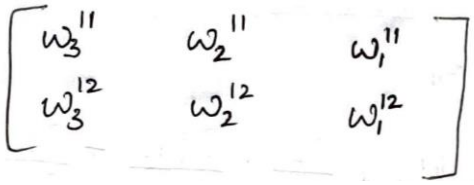
So, $B = A^T$

c) Yes, it is possible to imagine (∂L/∂~X) = B (∂L/∂~Y) as a convolution on (∂L/∂Y) to obtain (∂L/∂X)

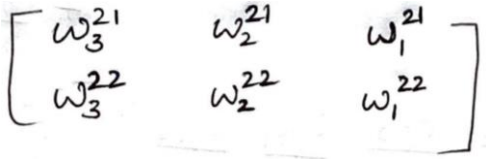
Since, it is kind of a up sampling operation (2x2 to 4x2), we need to pad (∂L/∂Y) with zeros first. For simplicity, let us denote ∂L/∂Y as ∇Y:

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

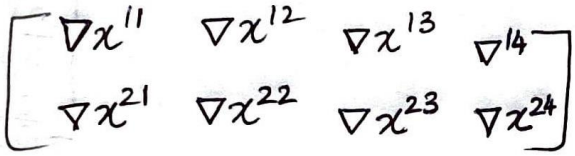
Now, the 1st kernel would be



And, the 2nd kernel would be

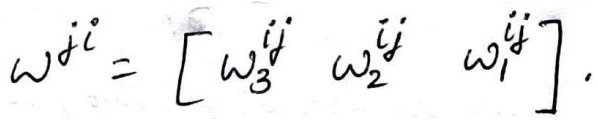


The output would be



Which is the same as (∂L/∂X)

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



where Wji scans the jth channel in a padded (∂L/∂Y) and produces ith channel in (∂L/∂X)
Therefore, it is possible to view the gradient computation as a convolution operation.