CPEN455: Deep Learning
Homework 2

Seiya Nozawa-Temchenko

Finished: Feb. 23, 2025

1 XOR Function [21pts]

Let us consider the XOR function. Given two binary digits, if they differ from each other, then XOR returns 1, otherwise 0. The visualization of the XOR function is shown in Figure 1. Now let us use neural networks to learn this function, *i.e.*, to find a neural network that can exactly implement this function.

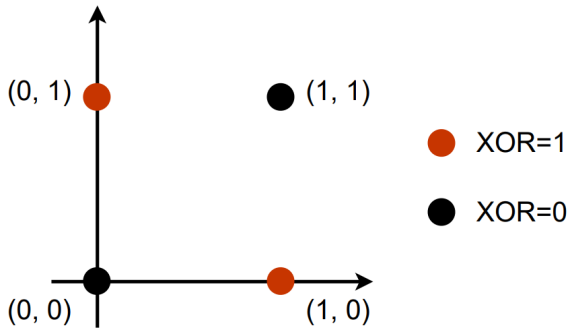

Figure 1: XOR Function

1.1 [5pts] Explain why a single linear-layer neural network can not learn the XOR function. In particular, in our context, a linear-layer neural network is $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$, where $\mathbf{x} \in \{0, 1\}^2$, $\mathbf{w} \in \mathbb{R}^2$, $b \in \mathbb{R}$.

A single linear-layer neural network can't learn XOR as it is not linearly separable. XOR's plotted output 1 lies on one diagonal, while output 0 lies on the opposite, meaning that there is no straight line that can group the outcomes. Since $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ can only produce a linear decision boundary, it can't represent XOR. To learn XOR, at least one hidden layer with nonlinear activation is needed (like ReLU).

1.2 [8pts] Construct a two-layer neural network with the activation function $\sigma(x) = x^2$ that can exactly implement the XOR function. In other words, the architecture is of a "Linear-Activation-

Linear" style. You can use as many hidden units (*i.e.*, the number of nonlinearly activated units) as you like.

We can solve XOR by creating a representation such that for $\mathbf{x} = [x_1, x_2]^\top$:

$$h_1 = \sigma(x_1 - x_2),\ h_2 = \sigma(x_1 + x_2 - 1) \tag{1}$$

$$\mathbf{h}(\mathbf{x}) = \sigma(\mathbf{W}^\top \mathbf{x} + \mathbf{c}),\ \mathbf{W}^\top = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix},\ \mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \tag{2}$$

Since $\sigma(x) = x^2$, this creates the following behavior:

- For $[0, 0]^\top$, we have $h_1 = (0 - 0)^2 = 0,\ h_2 = (0 + 0 - 1)^2 = 1$
- For $[0, 1]^\top$, we have $h_1 = (0 - 1)^2 = 1,\ h_2 = (0 + 1 - 1)^2 = 0$
- For $[1, 0]^\top$, we have $h_1 = (1 - 0)^2 = 1,\ h_2 = (1 + 0 - 1)^2 = 0$
- For $[1, 1]^\top$, we have $h_1 = (1 - 1)^2 = 0,\ h_2 = (1 + 1 - 1)^2 = 1$

We can now choose the output layer to combine the hidden activations layer:

$$f(\mathbf{x}) = \frac{1}{2}(h_1 - h_2 + 1) \tag{3}$$

$$f(\mathbf{x}; \mathbf{W}, \mathbf{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \mathbf{h} + b,\ \boldsymbol{w} = \begin{bmatrix} \frac{1}{2} \\ -\frac{1}{2} \end{bmatrix},\ \mathbf{h} = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ -1 \end{bmatrix},\ b = \frac{1}{2} \tag{4}$$

We can test this solution to ensure it aligns with XOR:

- For $[0, 0]^\top$, we have $f(\mathbf{x}) = \frac{1}{2}(0 - 1 + 1) = 0$
- For $[0, 1]^\top$, we have $f(\mathbf{x}) = \frac{1}{2}(1 - 0 + 1) = 1$
- For $[1, 0]^\top$, we have $f(\mathbf{x}) = \frac{1}{2}(1 - 0 + 1) = 1$
- For $[1, 1]^\top$, we have $f(\mathbf{x}) = \frac{1}{2}(0 - 1 + 1) = 0$

1.3 [8pts] Construct a two-layer neural network with the ReLU activation function $\sigma(x) = \max(x, 0)$ that can exactly implement the XOR function. In other words, the architecture is of a "Linear-Activation-Linear" style. You can use as many hidden units (*i.e.*, the number of nonlinearly activated units) as you like.

We can solve XOR by creating a representation such that for $\mathbf{x} = [x_1, x_2]^\top$:

$$h_1 = \sigma(x_1 + x_2),\ h_2 = \sigma(x_1 + x_2 - 1) \tag{5}$$

$$\mathbf{h}(\mathbf{x}) = \sigma(\mathbf{W}^\top \mathbf{x} + \mathbf{c}),\ \mathbf{W}^\top = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},\ \mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \tag{6}$$

Since $\sigma(x) = x^2$, this creates the following behavior:

- For $[0, 0]^\top$, we have $h_1 = \max\{0, 0 + 0\} = 0,\ h_2 = \max\{0, 0 + 0 - 1\} = 0$

- For $[0,1]^\top$, we have $h_1 = \max\{0, 0+1\} = 1,\ h_2 = \max\{0, 0+1-1\} = 0$
- For $[1,0]^\top$, we have $h_1 = \max\{0, 1+0\} = 1,\ h_2 = \max\{0, 1+0-1\} = 0$
- For $[1,1]^\top$, we have $h_1 = \max\{0, 1+1\} = 2,\ h_2 = \max\{0, 1+1-1\} = 1$

We can now choose the output layer to combine the hidden activations layer:

$$f(\mathbf{x}) = h_1 - 2h_2 \tag{7}$$

$$f(\mathbf{x}; \mathbf{W}, \mathbf{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \mathbf{h} + b,\ \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix},\ \mathbf{h} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ -1 \end{bmatrix},\ b = 0 \tag{8}$$

We can test this solution to ensure it aligns with XOR:

- For $[0,0]^\top$, we have $f(\mathbf{x}) = 0 - 2(0) = 0$
- For $[0,1]^\top$, we have $f(\mathbf{x}) = 1 - 2(0) = 1$
- For $[1,0]^\top$, we have $f(\mathbf{x}) = 1 - 2(0) = 1$
- For $[1,1]^\top$, we have $f(\mathbf{x}) = 2 - 2(1) = 0$

2 Convolution [34pts]

Suppose we have a mini-batch of images X with size (batch $\times$ channel $\times$ height $\times$ width) $100 \times 3 \times 512 \times 512$. We would like to classify these images into 10 categories. Let us build a convolutional neural network with the following specifications per layer:

- 1st layer: $H_1 = \text{ReLU}(\text{Conv}(X))$, where we have 128 convolutional filters (filter/kernel size $5 \times 5 \times 3$) with stride 2 and padding 2.
- 2nd layer $H_2 = \text{ReLU}(\text{Conv}(X))$, where we have 64 convolutional filters (filter/kernel size $5 \times 5 \times ?$) with stride 2 and padding 2.
- 3rd layer: $H_3 = \text{ReLU}(\text{Conv}(X))$, where we have 32 convolutional filters (filter/kernel size $3 \times 3 \times ?$) with stride 2 and padding 1.
- 4th layer: $H_4 = \text{ReLU}(\text{Conv}(X))$, where we have 16 convolutional filters (filter/kernel size $1 \times 1 \times ?$) with stride 1 and padding 0.
- 5th layer: $H_5 = \text{AvgPool}(X)$, where we apply a 2D average pooling (kernel size 2×2) with stride 2 and padding 0.
- 6th layer $Y = \text{Linear}(\text{Flatten}(H_5))$, where we flatten the tensor and apply a linear layer to compute the logits $\mathbf{Y}$.

2.1 **[3pts]** Compute the numbers of channels of the convolutional kernels at 2nd, 3rd, and 4th layers (*i.e.*, those marked with ?)

The depth of the convolutional kernel is the total number of channels and must always have the same number of channels as the previous layers' filters. This means that the 2nd layer's kernel has 128 channels, the 3rd layer's kernel has 64 channels, and the 4th layer's kernel has 32 channels.

2.2 **[6pts]** Compute the shapes of representations at each layer, *i.e.*, H_1, H_2, H_3, H_4, H_5, and Y.

We can find the shapes of representation at each layer by referencing the convolution formula:

$$\text{size}[y] = \left\lfloor \frac{n + 2p - m}{s} + 1 \right\rfloor, \; n = \text{size}[x], \; m = \text{size}[h] \tag{9}$$

This means that the output shapes are:

- $H_1 \rightarrow 100 \times 128 \times 256 \times 256$
- $H_2 \rightarrow 100 \times 64 \times 128 \times 128$
- $H_3 \rightarrow 100 \times 32 \times 64 \times 64$
- $H_4 \rightarrow 100 \times 16 \times 64 \times 64$
- Pooling is similar to convolution except that it averages neighbors rather than taking the weighted sum. This means that at H_5, shape is $100 \times 16 \times 32 \times 32$.
- When we flatten H_5, the shape is 100×16384. After we apply a linear layer (multiplication using a weight matrix size 16384×10) our logits $\mathbf{Y}$ have shape 100×10.

2.3 **[6pts]** Suppose all convolutional layers and the last linear layer **do not** have the bias parameters. Compute the number of learnable parameters per layer and the total number of learnable parameters.

The learnable parameter comes from the weights in the filters. So the number of learnable parameters would be the product of the filter size and the number of filters.

- $H_1 \rightarrow 5 \times 5 \times 3 \times 128 = 9600$
- $H_2 \rightarrow 5 \times 5 \times 128 \times 64 = 204800$
- $H_3 \rightarrow 3 \times 3 \times 64 \times 32 = 18432$
- $H_4 \rightarrow 1 \times 1 \times 32 \times 16 = 512$
- $H_5 \rightarrow 0$ as AvgPool has no weights or biases, but rather just performs an operation
- $\mathbf{Y} \rightarrow 16384 \times 10 = 163840$

2.4 **[6pts]** Suppose all convolutional layers and the last linear layer have the bias parameters. Compute the number of bias parameters per layer and the total number of learnable parameters.

With the bias parameters included, each filter gets an extra bias parameter, meaning we can sum this value to our earlier results.

- $H_1 \rightarrow 128 = 9600 + 128 = 9728$
- $H_2 \rightarrow 204800 + 64 = 204864$
- $H_3 \rightarrow 18432 + 32 = 18464$
- $H_4 \rightarrow 512 + 16 = 528$
- $H_5 \rightarrow 0$ as AvgPool has no weights or biases, but rather just performs an operation
- $\mathbf{Y} \rightarrow 163840 + 10 = 163850$

2.5 [13pts] We learn batch normalization (BN) for linear layers. Now let us generalize it to convolutional layers! The key idea of BN is to *compute the mean and standard deviation (std) of activations so that one can normalize them by subtracting the mean and dividing them by the standard deviation.*

In convolutional layers, denoting the activation as $X \in \mathbb{R}^{B \times C \times H \times W}$ (batch $\times$ channel $\times$ height $\times$ width), there are multiple ways to compute the mean and std.

1. Suppose we would like to compute one mean and one std for the entire mini-batch. Write the normalization equations (*i.e.*, computing mean and std, and subtracting the mean and dividing by the std) of BN for convolutional layers. [2pts]
2. Suppose we would like to compute C mean and C std for the entire mini-batch. Write the normalization equations of BN for convolutional layers. [2pts]
3. Suppose we would like to compute $H \times W$ mean and $H \times W$ std for the entire mini-batch. Write the normalization equations of BN for convolutional layers. [2pts]
4. Suppose we would like to compute $C \times H \times W$ mean and $C \times H \times W$ std for the entire mini-batch. Write the normalization equations of BN for convolutional layers. [2pts]

Suppose we want the normalization layer to behave like a convolution layer, *i.e.*, we use the same operations at different spatial locations but different operations across input channels. Which one of the above four ways should you choose to compute the mean and std? [2pts] How would you add the learnable linear transformation as what BN does for MLPs? Write the equations. [3pts]

To compute one mean and one std for a mini-batch we have:

$$m = \frac{1}{BCHW} \sum_{b=1}^{B} \sum_{c=1}^{C} \sum_{h=1}^{H} \sum_{w=1}^{W} X_{b,c,h,w} \tag{10}$$

$$v = \frac{1}{BCHW} \sum_{b=1}^{B} \sum_{c=1}^{C} \sum_{h=1}^{H} \sum_{w=1}^{W} (X_{b,c,h,w} - m)^2 \tag{11}$$

$$\hat{X}_{b,c,h,w} = \frac{X_{b,c,h,w} - m}{\sqrt{v + \epsilon}} \tag{12}$$

To compute one mean and one std per channel we have:

$$m_c = \frac{1}{BHW} \sum_{b=1}^{B} \sum_{h=1}^{H} \sum_{w=1}^{W} X_{b,c,h,w} \tag{13}$$

$$v_c = \frac{1}{BHW}\sum_{b=1}^{B}\sum_{h=1}^{H}\sum_{w=1}^{W}(X_{b,c,h,w} - m_c)^2 \tag{14}$$

$$\hat{X}_{b,c,h,w} = \frac{X_{b,c,h,w} - m_c}{\sqrt{v_c + \epsilon}} \tag{15}$$

To compute one mean and one std per location we have:

$$m_{h,w} = \frac{1}{BC}\sum_{b=1}^{B}\sum_{c=1}^{C}X_{b,c,h,w} \tag{16}$$

$$v_{h,w} = \frac{1}{BC}\sum_{b=1}^{B}\sum_{c=1}^{C}(X_{b,c,h,w} - m_{h,w})^2 \tag{17}$$

$$\hat{X}_{b,c,h,w} = \frac{X_{b,c,h,w} - m_{h,w}}{\sqrt{v_{h,w} + \epsilon}} \tag{18}$$

To compute one mean and one std per channel and location we have:

$$m_{c,h,w} = \frac{1}{B}\sum_{b=1}^{B}X_{b,c,h,w} \tag{19}$$

$$v_{c,h,w} = \frac{1}{B}\sum_{b=1}^{B}(X_{b,c,h,w} - m_{c,h,w})^2 \tag{20}$$

$$\hat{X}_{b,c,h,w} = \frac{X_{b,c,h,w} - m_{c,h,w}}{\sqrt{v_{c,h,w} + \epsilon}} \tag{21}$$

Convolutional layers apply the same filter across spatial locations, meaning that, for each channel, the same weights are used. Computing C mean and std for the entire mini-batch ensures we have the same operation at each spatial location, which varies per channel.

We can add learnable shift parameter γ and scale parameter β per channel:

$$m_c = \frac{1}{BHW}\sum_{b=1}^{B}\sum_{h=1}^{H}\sum_{w=1}^{W}X_{b,c,h,w} \tag{22}$$

$$v_c = \frac{1}{BHW}\sum_{b=1}^{B}\sum_{h=1}^{H}\sum_{w=1}^{W}(X_{b,c,h,w} - m_c)^2 \tag{23}$$

$$\hat{X}_{b,c,h,w} = \gamma_c\frac{X_{b,c,h,w} - m_c}{\sqrt{v_c + \epsilon}} + \beta_c \tag{24}$$

2.6 [Bonus 6pts] Compute the sizes of receptive fields of any unit at each layer, *i.e.*, any element of i.e., H_1, H_2, H_3, H_4, H_5, and Y.

A receptive field is the region of input that affects the output (based on filter size). In general, the relationship, with kernel size k, stride s, and receptive field r, can be seen as:

$$r_0 = 1 + \sum_{l=1}^{L}(h_l - 1)\prod_{j=1}^{l-1} s_j \tag{25}$$

- $H_1 \rightarrow r_0 = 1 + (k_1 - 1)\prod_{j=1}^{0} s_j = 1 + (5-1)(1) = 5 \rightarrow 5 \times 5$
- $H_2 \rightarrow r_0 = 1 + \sum_{l=1}^{2}(h_l - 1)\prod_{j=1}^{l-1} s_j = 1 + 4 + (5-1)(2) = 13 \rightarrow 13 \times 13$
- $H_3 \rightarrow r_0 = 1 + \sum_{l=1}^{3}(h_l - 1)\prod_{j=1}^{l-1} s_j = 1 + 4 + 8 + (3-1)(2)(2) = 21 \rightarrow 21 \times 21$
- $H_4 \rightarrow r_0 = 1 + \sum_{l=1}^{4}(h_l - 1)\prod_{j=1}^{l-1} s_j = 1 + 4 + 8 + 8 + (1-1)(2)(2)(2) = 21 \rightarrow 21 \times 21$
- $H_5 \rightarrow r_0 = 1 + \sum_{l=1}^{5}(h_l - 1)\prod_{j=1}^{l-1} s_j = 1 + 4 + 8 + 8 + 0 + (2-1)(2)(2)(2)(1) = 29 \rightarrow 29 \times 29$
- $\mathbf{Y} \rightarrow 29 \times 29$ since it is just the flattened output of H_5

3 Transposed Convolution [20pts]

3.1 [4pts] Suppose we have a single-channel image of size $X \in \mathbb{R}^{H \times W}$. We perform convolution with a kernel of size $K \times K$, stride of S, and padding of P. Derive the expressions of the size of the output $Y = \text{Conv}(X)$. In particular, suppose $Y \in \mathbb{R}^{H' \times W'}$, write the expressions of H' and W'.

With padding, the image is effectively size $X' \in \mathbb{R}^{(H+2P)\times(W+2P)}$. We can reference the convolution formula to find the output size:

$$H' = \left\lfloor \frac{H + 2P - K}{S} + 1 \right\rfloor, \; W' = \left\lfloor \frac{W + 2P - K}{S} + 1 \right\rfloor, \; Y \in \mathbb{R}^{H' \times W'} \tag{26}$$

3.2 [8pts] Following the question 3.1, suppose we have $Y \in \mathbb{R}^{H' \times W'}$. If we further apply the transposed convolution with a kernel of size $K \times K$, stride of S, input padding of P, and output padding of P' to previous output Y, we should be able to produce an image $\hat{X} = \text{TransposedConv}(Y)$ that has the same shape as the input image X, *i.e.*, $\hat{X} \in \mathbb{R}^{H \times W}$. Derive the expression of the output padding P'. Note that the output padding P' is different from the input padding P and both of them will be used in transposed convolution.

$$\begin{aligned} W' = \left\lfloor \frac{W + 2P - K}{S} + 1 \right\rfloor &\rightarrow W = S(W' - 1) - 2P + K \\ \hat{W} = W &\rightarrow \hat{W} = S(W' - 1) - 2P + K + P' \\ &\rightarrow P' = \hat{W} - (S(W' - 1) - 2P + K) \\ \hat{W} = W &\rightarrow P' = W - (S(W' - 1) - 2P + K) \end{aligned} \tag{27}$$

Note that dimensions W and H are interchangeable since $W = H$.

3.3 [8pts] Following the above context, let us look at concrete examples of transposed convolution.

(1) Suppose we apply the 2D transposed convolution (stride $S = 1$, padding $P = 0$, output padding $P' = 0$) with kernel W to Y , where

$$Y = \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix} \quad W = \begin{bmatrix} -1 & 2 \\ 3 & -4 \end{bmatrix} \tag{28}$$

Compute the output $\hat{X}$ and show the intermediate computation results.

(2) Suppose we apply the 2D transposed convolution (stride $S = 2$, padding $P = 1$, output padding $P' = 1$) with kernel W to Y, where

$$Y = \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix} \quad W = \begin{bmatrix} -1 & 2 & -3 \\ 1 & -2 & 3 \\ 3 & 2 & -1 \end{bmatrix} \tag{29}$$

Compute the output $\hat{X}$ and show the intermediate computation results.

Using our previous equation, we can identify that size is $W_1 = 1(2-1) - 2(0) + 2 = 3$:

$$\begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix} \text{ with } \begin{bmatrix} -1 & 2 \\ 3 & -4 \end{bmatrix} = \begin{bmatrix} -4 & 8 & \\ 12 & -16 & \\ & & \end{bmatrix} + \begin{bmatrix} & -3 & 6 \\ & 9 & -12 \\ & & \end{bmatrix} + \begin{bmatrix} & & \\ -2 & 4 & \\ 6 & -8 & \end{bmatrix} + \begin{bmatrix} & & \\ & -1 & 2 \\ & 3 & -4 \end{bmatrix}$$
$$\hat{X} = \begin{bmatrix} -4 & 5 & 6 \\ 10 & -4 & -10 \\ 6 & -5 & -4 \end{bmatrix} \tag{30}$$

Since this is a little more tedious, I implemented the transposed convolution function from PML1 p.487. The idea is equivalent to padding the input image with $(h-1, w-1)$ 0s (on the bottom right), where (h, w) is the kernel size, then placing a weighted copy of the kernel on each one of the input locations, where the weight is the corresponding pixel value, and then adding up. The process is illustrated with the easier example in part 1 above.

```
import numpy as np

K = np.array([[-1, 2, -3],
              [ 1, -2,  3],
              [ 3,  2, -1]])
X = np.array([[9, 8, 7],
              [6, 5, 4],
              [3, 2, 1]])

h,w = K.shape
Y = np.zeros((X.shape[0] + h - 1, X.shape[1] + w - 1), dtype=float)
```

```
for i in range(X.shape[0]):
    for j in range(X.shape[1]):
        Y[i:i + h, j:j + w] += X[i, j] * K
        print(Y)
        print("-----------------------------------------")
```

The terminal output with the $\hat{X}$ at the end:

```
PS C:\Users\me\Desktop\CPEN-455> py trans_conv.py
[[ -9.  18. -27.   0.   0.]
 [  9. -18.  27.   0.   0.]
 [ 27.  18.  -9.   0.   0.]
 [  0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.]]
-----------------------------------------
[[ -9.  10. -11. -24.   0.]
 [  9. -10.  11.  24.   0.]
 [ 27.  42.   7.  -8.   0.]
 [  0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.]]
-----------------------------------------
[[ -9.  10. -18. -10. -21.]
 [  9. -10.  18.  10.  21.]
 [ 27.  42.  28.   6.  -7.]
 [  0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.]]
-----------------------------------------
[[ -9.  10. -18. -10. -21.]
 [  3.   2.   0.  10.  21.]
 [ 33.  30.  46.   6.  -7.]
 [ 18.  12.  -6.   0.   0.]
 [  0.   0.   0.   0.   0.]]
-----------------------------------------
[[ -9.  10. -18. -10. -21.]
 [  3.  -3.  10.  -5.  21.]
 [ 33.  35.  36.  21.  -7.]
 [ 18.  27.   4.  -5.   0.]
 [  0.   0.   0.   0.   0.]]
-----------------------------------------
[[ -9.  10. -18. -10. -21.]
 [  3.  -3.   6.   3.   9.]
 [ 33.  35.  40.  13.   5.]
 [ 18.  27.  16.   3.  -4.]
 [  0.   0.   0.   0.   0.]]
-----------------------------------------
[[ -9.  10. -18. -10. -21.]
```

```
 [  3.  -3.   6.   3.   9.]
 [ 30.  41.  31.  13.   5.]
 [ 21.  21.  25.   3.  -4.]
 [  9.   6.  -3.   0.   0.]]
-----------------------------------------
[[ -9.  10. -18. -10. -21.]
 [  3.  -3.   6.   3.   9.]
 [ 30.  39.  35.   7.   5.]
 [ 21.  23.  21.   9.  -4.]
 [  9.  12.   1.  -2.   0.]]
-----------------------------------------
[[ -9.  10. -18. -10. -21.]
 [  3.  -3.   6.   3.   9.]
 [ 30.  39.  34.   9.   2.]
 [ 21.  23.  22.   7.  -1.]
 [  9.  12.   4.   0.  -1.]]
-----------------------------------------
```

4 Dilated (*a.k.a* Atrous) Convolution [14pts]

4.1 [**4pts**] Suppose we apply the 2D dilated convolution with kernel W, dilation (spacing between neighboring elements within the convolutional kernel) $D = 2$, stride $S = 2$, and padding $P = 2$ to input X, where

$$X = \begin{bmatrix} 3 & 1 & 2 & 6 & 5 \\ 6 & 8 & 1 & 7 & 9 \\ 2 & 7 & 4 & 2 & 3 \\ 8 & 3 & 5 & 4 & 1 \\ 1 & 5 & 2 & 7 & 6 \end{bmatrix} \quad W = \begin{bmatrix} -1 & 2 & -3 \\ 1 & -2 & 3 \\ 3 & 2 & -1 \end{bmatrix} \tag{31}$$

Compute the output $Y = \text{DilatedConv}(X)$ and show the intermediate computation results.

The main idea of dilated convolution comes from the following relation:

$$Y_{i,j} = \sum_{w=1}^{K} \sum_{h=1}^{K} x_{i+dw,j+dh} w_{w,h} \tag{32}$$

If dilation rate $d = 1$, the equation is just a standard convolution. We can begin by padding the input by $P = 2$:

$$X_{pad} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 1 & 2 & 6 & 5 & 0 & 0 \\ 0 & 0 & 6 & 8 & 1 & 7 & 9 & 0 & 0 \\ 0 & 0 & 2 & 7 & 4 & 2 & 3 & 0 & 0 \\ 0 & 0 & 8 & 3 & 5 & 4 & 1 & 0 & 0 \\ 0 & 0 & 1 & 5 & 2 & 7 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{33}$$

$$Y_{0,0} \rightarrow \begin{bmatrix} 0 & 0 & 0 \\ 0 & 3 & 2 \\ 0 & 2 & 4 \end{bmatrix} \cdot \begin{bmatrix} -1 & 2 & -3 \\ 1 & -2 & 3 \\ 3 & 2 & -1 \end{bmatrix} \rightarrow 0+0+0+0-6+6+0+4-4=0 \tag{34}$$

$$Y_{1,0} \rightarrow \begin{bmatrix} 0 & 3 & 2 \\ 0 & 2 & 4 \\ 0 & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} -1 & 2 & -3 \\ 1 & -2 & 3 \\ 3 & 2 & -1 \end{bmatrix} \rightarrow 0+6-6+0-4+12+0+2-2=8 \tag{35}$$

$$Y_{2,0} \rightarrow \begin{bmatrix} 0 & 2 & 4 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} -1 & 2 & -3 \\ 1 & -2 & 3 \\ 3 & 2 & -1 \end{bmatrix} \rightarrow 0+4-12+0-2+6+0+0+0=-4 \tag{36}$$

$$Y_{0,1} \rightarrow \begin{bmatrix} 0 & 0 & 0 \\ 3 & 2 & 5 \\ 2 & 4 & 3 \end{bmatrix} \cdot \begin{bmatrix} -1 & 2 & -3 \\ 1 & -2 & 3 \\ 3 & 2 & -1 \end{bmatrix} \rightarrow 0+0+0+3-4+15+6+8-3=25 \tag{37}$$

$$Y_{0,2} \rightarrow \begin{bmatrix} 0 & 0 & 0 \\ 2 & 5 & 0 \\ 4 & 3 & 0 \end{bmatrix} \cdot \begin{bmatrix} -1 & 2 & -3 \\ 1 & -2 & 3 \\ 3 & 2 & -1 \end{bmatrix} \rightarrow 0+0+0+2-10+0+12+6+0=10 \tag{38}$$

$$Y_{1,1} \rightarrow \begin{bmatrix} 3 & 2 & 5 \\ 2 & 4 & 3 \\ 1 & 2 & 6 \end{bmatrix} \cdot \begin{bmatrix} -1 & 2 & -3 \\ 1 & -2 & 3 \\ 3 & 2 & -1 \end{bmatrix} \rightarrow -3+4-15+2-8+9+3+4-6=-10 \tag{39}$$

$$Y_{1,2} \rightarrow \begin{bmatrix} 2 & 5 & 0 \\ 4 & 3 & 0 \\ 2 & 6 & 0 \end{bmatrix} \cdot \begin{bmatrix} -1 & 2 & -3 \\ 1 & -2 & 3 \\ 3 & 2 & -1 \end{bmatrix} \rightarrow -2+10+0+4-6+0+6+12+0=24 \tag{40}$$

$$Y_{2,1} \rightarrow \begin{bmatrix} 2 & 4 & 3 \\ 1 & 2 & 6 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} -1 & 2 & -3 \\ 1 & -2 & 3 \\ 3 & 2 & -1 \end{bmatrix} \rightarrow -2+8-9+1-4+18+0+0+0=12 \tag{41}$$

$$Y_{2,2} \rightarrow \begin{bmatrix} 4 & 3 & 0 \\ 2 & 6 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} -1 & 2 & -3 \\ 1 & -2 & 3 \\ 3 & 2 & -1 \end{bmatrix} \rightarrow -4+6+0+2-12+0+0+0+0+0=-8 \tag{42}$$

$$\therefore Y = \begin{bmatrix} 0 & 25 & 10 \\ 8 & -10 & 24 \\ -4 & 12 & -8 \end{bmatrix} \tag{43}$$

4.2 [4pts] Suppose the input $X \in \mathbb{R}^{H \times W}$. If we apply the dilated convolution with a kernel of size $K \times K$, stride of S, and padding of P to the input X, we get the output $Y = \text{DilatedConv}(X)$ where $Y \in \mathbb{R}^{H' \times W'}$. Derive the expression of the output shapes, *i.e.*, H' and W'.

We can use dilated kernel size $K_d = 1 + (K-1)d$ and eq.26 to find output shapes:

$$H' = \left\lfloor \frac{H + 2P - K_d}{S} + 1 \right\rfloor = \left\lfloor \frac{H + 2P - d(K-1) - 1}{S} + 1 \right\rfloor \tag{44}$$

$$\therefore W' = \left\lfloor \frac{W + 2P - d(K-1) - 1}{S} + 1 \right\rfloor \tag{45}$$

4.3 [6pts] Following the context of 4.2, we apply the transposed dilated convolution with a kernel of size $K \times K$, stride of S, padding of P, and output padding of P' to Y so that we have the same sized output as X. In particular, denoting $\hat{X} = \text{TransposedDilatedConv}(Y)$, we have $\hat{X} \in \mathbb{R}^{H \times W}$. Derive the expression of the output padding P'.

$$\begin{aligned} W' = \left\lfloor \frac{W + 2P - K_d}{S} + 1 \right\rfloor \rightarrow W &= S(W'-1) - 2P + (1 + d(K-1)) \\ \hat{W} = W \rightarrow \hat{W} &= S(W'-1) - 2P + (1 + d(K-1)) + P' \\ P' &= \hat{W} - (S(W'-1) - 2P + d(K-1) + 1) \\ \hat{W} = W \rightarrow P' &= W - (S(W'-1) - 2P + d(K-1) + 1) \end{aligned} \tag{46}$$

Note that dimensions W and H are interchangeable since $W = H$.

5 Grouped Convolution [5pts]

Suppose the input $X \in \mathbb{R}^{B \times C \times H \times W}$. We ignore the bias in this question.

5.1 [1pts] If we apply the convolution with N kernels (each with spatial size $K \times K$), stride of 1, and half padding to the input X, we get the output $Y = \text{Conv}(X)$. Compute the number of learnable parameters in this convolution.

The learnable parameter comes from the weights in the filters. So the number of learnable parameters would be the product of the filter size and the number of filters.

$$K \times K \times C \times N = CK^2N \tag{47}$$

5.2 [4pts] If we apply the grouped convolution with group size M, spatial kernel size $K \times K$, stride of 1, and half padding to the input X, we get the output $Y' = \text{GroupedConv}(X)$, which is of the same shape as Y in 5.1, *i.e.*, $Y' \in \mathbb{R}^{B \times N \times H \times W}$. Assuming M divides C and M divides N, compute the number of learnable parameters in this convolution. Explain the benefits of grouped convolutions vs. the vanilla convolutions in 5.1.

The learnable parameter comes from the weights in the filters. So the number of learnable parameters would be the product of the filter size and the number of filters. Since grouped convolution splits vanilla convolution into groups, we can find the number of learnable parameters per group.

$$K \times K \times \frac{C}{M} \times \frac{N}{M} = \frac{CK^2N}{M^2} \tag{48}$$

Grouped convolution then concatenates the M groups on top of each other.

$$M \times \frac{CK^2N}{M^2} = \frac{CK^2N}{M} \tag{49}$$

Grouped convolution is a way to maintain the same shaped input and output in convolution with a fewer number of parameters. It helps reduce memory storage and can speed up training. It also improves computational efficiency by lowering costs and allowing better parallelization for maximal hardware use. These benefits are a reason that AlexNet, which uses grouped convolution, is so efficient.

6 Depthwise Separable Convolution [6pts]

Suppose the input $X \in \mathbb{R}^{B\times C\times H\times W}$. We ignore the bias in this question. Consider the vanilla convolution with N kernels (each with size $K\times K\times C$), stride of 1, and half padding to the input X. Then consider the depthwise separable convolution which has the same spatial kernel size $K \times K$ and produces the same-size output as the vanilla convolution. Compare these two types of convolutions in terms of the number of learnable parameters and the number of operations (multiplications and additions between scalars).

Vanilla convolution has CK^2N parameters as we found earlier. For each output element shape $B \times N \times H \times W$, we compute a dot-product over CK^2 elements (for multiplications but $CK^2 - 1$ for additions).

$$B \times N \times H \times W \times (CK^2 + CK^2 - 1) = BHNW(2CK^2 - 1) \approx 2BCHK^2NW \tag{50}$$

Depthwise separable convolution is broken into two steps, a depthwise convolution and a pointwise convolution. In depthwise, instead of N kernels, we have one $K \times K$ kernel per one of C channels. For each output element shape $B \times C \times H \times W$, we compute a dot-product over K^2 elements (for multiplications but $K^2 - 1$ for additions).

$$B \times C \times H \times W \times (K^2 + K^2 - 1) = BCHW(2K^2 - 1) \approx 2BCHK^2W \tag{51}$$

In pointwise, since we have a 1×1 convolution with C channels for N filters, we have CN parameters. For each output element shape $B \times N \times H \times W$, we compute over C elements (for multiplications but $C - 1$ for additions).

$$B \times H \times W \times (CN + CN - 1) = BHW(2CN - 1) \approx 2BCHNW \tag{52}$$

We can combine pointwise and depthwise for the depthwise separable convolution.

$$\text{Parameters} = C(K^2 + N),\ \text{Operations} = 2BCHW(K^2 + N) \tag{53}$$

For comparison, I will write the vanilla convolution results as well.

$$\text{Parameters} = CK^2N,\ \text{Operations} = 2BCHW(K^2N) \tag{54}$$

We see that depthwise separable convolution is almost always much fewer parameters as soon as $K^2 > 1$. Depthwise separable also reduces (especially for large K) compared to vanilla convolution.

7 Receptive Field Size [Bonus 12pts]

Suppose we have a mini-batch of images X with size (batch $\times$ channel $\times$ height $\times$ width) $100 \times 3 \times 512 \times 512$. We would like to classify these images into 10 categories. Let us build a convolutional neural network with the following specification per layer:

- 1st layer: $H_1 = \text{ReLU}(\text{Conv}(X))$ where we have 32 convolutional filters (filter/kernel size $5 \times 5 \times 3$) with stride 2 and padding 2.
- 2nd layer: $H_2 = \text{ReLU}(\text{DepthwiseConv}(X))$ where we have 32 depthwise convolutional filters (filter/kernel size 5×5) with stride 2 and padding 2.
- 3rd layer: $H_3 = \text{ReLU}(\text{PointConv}(X))$ where we have 64 point convolutional filters (filter/kernel size $1 \times 1 \times 32$) with stride 1 and padding 0.
- 4th layer: $H_4 = \text{ReLU}(\text{TransposedConv}(X))$ where we have 32 transposed convolutional filters (the corresponding convolutional filter/kernel size, the stride, and the padding are $3 \times 3 \times 32$, 2, and 0 respectively). There is no output padding.
- 5th layer: $H_5 = \text{AvgPool}(X)$ where we have an 2D average pooling (kernel size 2×2) with stride 2 and padding 0.
- 6th layer: $Y = \text{Linear}(\text{Flatten}(H5))$ where we flatten the tensor and apply a linear layer to compute the logits Y.

Compute the sizes of receptive fields of any unit at each layer, *i.e.*, any element of *i.e.*, H_1, H_2, H_3, H_4, H_5, and Y.

For an ordinary 2D convolution, the formula for receptive field for layer i is $r_i = r_{i-1} + (k-1)j_{i-1}$ where $j_i = j_{i-1}s$. We can refer to the full equation:

$$r_0 = 1 + \sum_{l=1}^{L}(h_l - 1)\prod_{j=1}^{l-1} s_j \tag{55}$$

- 1st layer: $H_1 = \text{ReLU}(\text{Conv}(X))$
  $r_1 = r_0 + (k-1)j_0 = 1 + (5-1)1 = 5,\ j_1 = j_0 s = 1(2) = 2$
  Receptive field size: 5×5
- 2nd layer: $H_2 = \text{ReLU}(\text{DepthwiseConv}(X))$
  $r_2 = r_1 + (k-1)j_1 = 5 + (5-1)2 = 13,\ j_2 = j_1 s = 2(2) = 4$
  Receptive field size: 13×13
- 3rd layer: $H_3 = \text{ReLU}(\text{PointConv}(X))$
  $r_3 = r_2 + (k-1)j_2 = 13 + (1-1)4 = 13,\ j_3 = j_2 s = 4(1) = 4$
  Receptive field size: 13×13
- 4th layer: $H_4 = \text{ReLU}(\text{TransposedConv}(X))$
  Transposed convolution essentially upsamples spatial dimension. Since stride should expand the output spatial dimension by s, each output location covers more of the input than a vanilla convolution with the same kernel size and stride. We are, in some sense, almost reversing the

formula.
$r_4 = k + (r_3 - 1)j_3 = 3 + (13 - 1)4 = 27,\ j_4 = \frac{j_3}{s} = \frac{4}{2} = 2$
Receptive field size: 27×27

- 5th layer: $H_5 = \text{AvgPool}(X)$
  $r_5 = r_4 + (k - 1)j_4 = 27 + (2 - 1)2 = 29,\ j_5 = j_4 s = 2(2) = 4$
  Receptive field size: 29×29

- 6th layer: $Y = \text{Linear}(\text{Flatten}(H5))$
  Since the final layer is connected all positions of H_5, each output unit depends on the H_5 map. This means that the linear layer does not add a further local kernel.
  Receptive field size: 29×29