E0 294: Systems for Machine Learning
Assignment 1

Rohit Jorige (22166)
Indian Institute of Science, Bangalore

January 29, 2025

Problem 1

Draw a random number between 0 and 1, randomly choose positive/negative sign and assign to each weight value and input fmap value. Assume average pooling, zero padding and stride length of two.

a) Implement the 7-loop naïve implementation of convolution

Listing 1: 7-loop Naïve Convolution Implementation with Stride=2

```
# 7-loop naive implementation with stride length = 2 , average pooling and
    ↪ zero padding
# output will be 8 x 32 x 14 x 14 ( N x M x H x W)
output_fmaps = []
# N = 8 , C = 4 , H = 32 , W = 32 , M = 32 , H = 5 , W = 5
for n in range(8):
    fmap = []
    for m in range(32):
        row = []
        for x in range(14):
            col = []
            for y in range(14):
                mac = 0
                # mac += biases[m] ( not including biases in our calculation)
                for i in range(5):
                    for j in range(5):
                        for c in range(4):
                            mac += input_fmaps[n][c][2*x+i][2*y+j] *
                                ↪ weights[m][c][i][j]
                col.append(mac)
            row.append(col)
        fmap.append(row)
    output_fmaps.append(fmap)
```

We are looping 14 times because of stride length 2 we see that the number of times a kernel passes completley horizontally or vertically is (32-5)//2 +1 = 14.

b) Implement convolution using Toeplitz matrices

Listing 2: Construction of Toeplitz matrices

```
# constructing topelitz matrix for weights
topelitz_weights = []
for m in range(32):
    row = []
    for c in range(4):
        for h in range(5):
            for w in range(5):
                row.append(weights[m][c][h][w])
    topelitz_weights.append(row)

# constructing input matrix for input fmaps
topelitz_input_fmaps = []
for c in range(4):
    for h in range(5):
        for w in range(5):
            row = []
            for n in range(8):
                for x in range(14):
                    for y in range(14):
                        row.append(input_fmaps[n][c][2*x+h][2*y+w])
            topelitz_input_fmaps.append(row)
```

Listing 3: Matrix multiplication and output conversion

```
# multiplying topelitz weights and topelitz input fmaps
topelitz_output_fmaps = []
for x in range(32):
    row = []
    for z in range(1568):
        sum_elements = 0
        for y in range(100):
            sum_elements += topelitz_weights[x][y] * topelitz_input_fmaps[y][z]
        row.append(sum_elements)
    topelitz_output_fmaps.append(row)

# convert output fmaps to topelitz format for comparison
output_fmaps_converted_to_topelitz = []
for m in range(32):
    row = []
    for n in range(8):
        for x in range(14):
            for y in range(14):
                row.append(output_fmaps[n][m][x][y])
    output_fmaps_converted_to_topelitz.append(row)
```

c) Show that both cases produce equal output.

Implemented in code and shown that both give the same equal output. (used numpy to show that both arrays are the same due to rounding off error checking the normal array 1 == array 2)

d) Obtain and report number of instructions and execution time for both a) and b) using perf command.

In the program for part(a) where I run the naive implementation of convolution, the construction of the matrix is also concluded. To get close to the true time taken and number of instructions for only the implementation of the convultion I have implemented the construction of the matrix in a separate program and recorded the time and instructions taken. I have subtracted this from the naive implementation to get close to the true time taken for only the naive conv implementation.

```
   time taken :  2,530.84 - 75.71 = 2455.13 msec
   instructions :  15,524,772,353 - 251,531,762 = 15,273,240,591
```

I have done the same for the Toeplitz implementation of the convolution. I have written a separate program where I recorded the time taken and the instructions in constructing the matrix and then transforming the input and weights matrix to the toeplitz form.

```
   time taken :  1,297.37 - 119.82 = 1177.55 msec
   instructions :  8,831,811,710 - 542,635,354 = 8,289,176,356
```

e) Analyze and comment on the above results

```
Approximate Effective Speedup :  2455.13 ÷ 1177.55 = 2.084947561
Approximate Reduction in Instructions :  15273240591 ÷ 8289176356 = 1.842552256x
```

Reasons for the time speed up and the reduction in number of instructions:
1) **Loop Overhead** : Since we are doing a 7 loop naive implementation, each loop requires index checks, increments, and condition evaluations, these increase execution time and the number of instructions executed significantly.
2) **Memory Access** : We are also doing a stride of length 2, in our naive implementation this causes cache misses and increases the time taken, while in the Toeplitz form since it is just straight matrix multiplications this does not cause any significant cache misses like the naive implementation.
Even if we consider the time taken to make the Teoplitz matrix we can also observe form the results that the time take to construct the toeplitz matrix is outweighed by the efficency it brings about.

Problem 2

Given an input feature X whose dimension is $[h = 3, w = 3, i = 2]$:

$$X = \begin{bmatrix} -8 & 4 & 8 \\ -2 & -9 & -7 \\ 7 & -5 & 7 \end{bmatrix}, \quad \begin{bmatrix} -7 & 6 & 7 \\ -9 & -3 & -8 \\ 1 & -3 & 0 \end{bmatrix}$$

Given a convolution weight whose dimension is $[h = 2, w = 2, i = 2, o = 3]$:

$$\text{Woutchannel1} = \begin{bmatrix} -8 & 1 \\ -2 & 3 \end{bmatrix}, \quad \begin{bmatrix} -6 & -6 \\ 2 & 5 \end{bmatrix}$$

$$\text{Woutchannel2} = \begin{bmatrix} -6 & 9 \\ -7 & 5 \end{bmatrix}, \quad \begin{bmatrix} -4 & 7 \\ -1 & 0 \end{bmatrix}$$

$$\text{Woutchannel3} = \begin{bmatrix} -2 & -3 \\ -2 & 1 \end{bmatrix}, \quad \begin{bmatrix} -3 & -9 \\ 1 & 0 \end{bmatrix}$$

a) What is the dimension of Y = Conv2D(W, X), and what is the total number of elements in the output feature?

The width and height of the Output feature can be calcualted by input width/height - kernel width/height + 1 = 3-2+1 = 2. The number of channels if given by the number of output filters = 3 . Hence the dimension of the output feature is 2x2x3 [h=2 , w=2 , i=3]. Total number of elements in the output feature is 2x2x3 = 12. If we apply a 2x2 average pooling to this we get a 1x1x3 output , then the number of elements in the output feature is 3.

b) Convert Y = Conv2D(W, X) into matrix multiplication $Y' = W' \times X'$ using the Toeplitz matrix generation technique introduced in the class.

Converting to Toeplitz matrix form, we get the following matrices:

$$W' = \begin{bmatrix} -8 & 1 & -2 & 3 & -6 & -6 & 2 & 5 \\ -6 & 9 & -7 & 5 & -4 & 7 & -1 & 0 \\ -2 & -3 & -2 & 1 & -3 & -9 & 1 & 0 \end{bmatrix}$$

$$X' = \begin{bmatrix} -8 & 4 & -2 & -9 \\ 4 & 8 & -9 & -7 \\ -2 & -9 & 7 & -5 \\ -9 & -7 & -5 & 7 \\ -7 & 6 & -9 & -3 \\ 6 & 7 & -3 & -8 \\ -9 & -3 & 1 & -3 \\ -3 & -8 & -3 & 0 \end{bmatrix}$$

$$Y' = W' \times X' = \begin{bmatrix} 18 & -151 & 37 & 156 \\ 132 & 104 & -129 & 20 \\ -43 & -105 & 67 & 134 \end{bmatrix}$$