Chair for Computer Architecture and Parallel Systems
School of Computation, Information and Technology
Technical University of Munich

Lab 3: Fast Convolution

Practical Course: Accelerating CNNs using PL

12.05.2023

1 Exercise

This last lab will cover optimization of convolution using the FFT and the Winograd transform. As part of the Software report you have to time the optimization schemes for different input sizes and kernel sizes and select the best scheme for your inference implementation. You can test and benchmark the different implementations using the provided `lab3.cpp` file. Verify your implementations for kernel sizes 3×3, 5×5,7×7 and 11×11 and benchmark them for different input sizes and analyze the precision of the different schemes.

2 Winograd

The Winograd transformation is much simpler to implement than the FFT convolution and is performed in a tiled way. You can either pre-compute your weight transformations before inference or transform them once per convolution. The Winograd $F(m \times m, r \times r)$ depends on the output tile size $m \times m$ and kernel size $r \times r$. The input tiles have to be overlapped by $r - 1$. The 1-D winograd transform can be written in matrix form as:

$$Y = A^T[(Gg) \otimes (B^T d)] \qquad \otimes \textit{ element-wise multiplication} \tag{1}$$

and the 2-D form using the same Matrices as $F(m, r)$:

$$Y = A^T[(GgG^T) \otimes (B^T dB)]A \qquad g = \textit{weight matrix} \quad d = \textit{input tile} \tag{2}$$

The back transform only has to be performed once per output channel, as the outputs for different input channels can be summed in the transformed form. After transforming the Weight Tensors, you should tile your input and loop the tiles performing a Winograd convolution for each tile. A more sophisticated approach to perform Winograd and FFT convolution as a Matrix-Matrix multiplication can be found in[1], which you can implement instead if desired.
The required Matrices (A^T,B^T and G) can be generated using the file `genWino.py` forked from [2] to create a C-Header declaring the required Matrices for Convolution.
The file `lab3.cpp` expects you to have implemented the function `winoWeights` to convert the Weight kernels and a `convWinograd` function that uses pre-converted weights and the input to perform convolution. The `WINOGRAD_STRUCT` holds the predefined Matrices as well as tile size and stride they can be generated using the `genWino.py` script.

[1] Lavin and Gray, “Fast Algorithms for Convolutional Neural Networks”.
[2] https://github.com/andravin/wincnn

3 FFT Convolution

The second optimization is to use the FFT to perform convolution. The 1-D DFT is defined as:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn} \qquad 0 \leq k < N \tag{3}$$

and the inverse as

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j\frac{2\pi}{N}kn} \qquad n \leq 0 < N \tag{4}$$

the factor $\frac{1}{N}$ can either be used in the forward or inverse transformation, but it has to be multiplied in one of them. The 2-D forward transformation then becomes:

$$X[p,q] = \sum_{m=0}^{M-1} (e^{-j\frac{2\pi}{M}pm} \sum_{n=0}^{N-1} x[m,n] e^{-j\frac{2\pi}{N}qn}) \tag{5}$$

$$X[p,q] = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} e^{-j\frac{2\pi}{N}qn} x[m,n] e^{-j\frac{2\pi}{M}pm} \tag{6}$$

$$X[p,q] = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[m,n] e^{-j2\pi(\frac{qn}{N}+\frac{pm}{M})} \tag{7}$$

and the respective inverse transform

$$x[p,q] = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] e^{j2\pi(\frac{qn}{N}+\frac{pm}{M})} \tag{8}$$

A convolution in the time domain becomes a multiplication in the frequency domain:

$$x[n] * w[n] \rightarrow X[k] \odot W[k] \tag{9}$$

To deal with the effects of circular convolution you should use the overlap-save (or a different overlap) scheme as discussed in the lecture. The `FFT_STRUCT` in the `conv.cpp` file have precomputed optimal values for tile size as described in the lecture but you can experiment with different tile sizes.
The weight kernel only has to be transformed once during convolution and can even be pre transformed for all convolutions. The back-transform only has to be performed once for every output channels as the different input channel convolutions can be added up in the frequency domain before the inverse FFT.

$$\alpha x[n] + \beta w[n] \rightarrow \alpha X[k] + \beta W[k] \tag{10}$$

You have to support fft convolutions for kernels of size 3×3, 5×5,7×7 and 11×11. The `C_Tensor` class is a Tensor of complex values, with less functionality as the full Tensor class but feel free to extend it. **The way `pytorch` and our implementation stores matrices for convolution is the wrong way for convolution using FFT. You thus first have to flip the Weight kernel before using it.** You can use the `flipMatrix()` function for this. We are only using real values in the time domain and can thus save on computation by using the fact that:

$$(g[n] + jf[n]) * w[n] = (g[n] * w[n]) + j(f[n] * w[n]) \qquad \textit{All real signals} \tag{11}$$

Which means we can convolute two tiles in one FFT convolution by storing one of them in the imaginary part.

4 Algorithm

4.1 FFT ($X^{N_x \times N_x}, W^{M \times M}$)

1. Pick tile size N (minimize)
2. Pad Weight matrix to N and perform 2D-FFT (Can be pre-calculated)
3. Loop through your input feature map with Stride $N - (M - 1)$
   a) Convert input tile ($N \times N$) and perform element-wise multiplication with converted weight
   b) Add up tiles from different input channels in frequency domain
   c) Perform inverse 2D-FFT on output tile
   d) Store $N - (M - 1)$ values in the output matrix (Discard first $M - 1$)

4.2 Winograd

1. Pick a output tile size $F(m \times m, r \times r)$
2. Generate the corresponding Matrices A^T, G, B^T
3. Convert the Weight Matrix GgG^T (Can be pre-calculated)
4. Loop through the input feature map with tile stride
   a) Convert input tile of tile size $B^T dB$
   b) Element-wise multiply $m = (B^T db) \odot (GgG^T)$
   c) Add up tiles from different input channels
   d) Convert back and store in appropriate output feature map $A^T mA$