Bitwise Convolution for Ternary and Binary Neural Networks

Ternary and binary neural networks (TNNs and BNNs) are widely adopted fast deep learning models. They utilize 2-bit ternary (-1, 0, +1) and 1-bit binary (-1, +1) values to conduct convolution. Taking TNNs as an example, ternary convolution contains quantization, data reshaping, and bitwise direct convolution or bitwise General Matrix Multiplication (GEMM). The detailed algorithm is described in the following paper:

TAB: unified and optimized ternary, binary and mixed-precision neural network inference on the edge

The goal of the project is to optimize one layer of a TNN, consisting of

- data preparation (Algorithm 1 in the paper), followed by
- bitwise GEMM (Algorithm 3 in the paper), followed by
- the activation function (e.g., PReLU https://paperswithcode.com/method/prelu).

You will implement one TNN layer on X86_64 or ARM CPU. Depending on how things go and difficulty, possible extension includes the other three types of ternary and binary convolution (TBN, BTN, and BNN) described in the paper.

Quick explanation & baseline code:

Algorithm 1 has four functionalities:

1. Quantization: It quantizes the input activation X into 2-bit ternary values.
2. Bit-Packing: Then it packs the 2-bit values into 64-bit integers for higher data-level parallelism because existing CPUs don't have native 2-bit data types.
3. Data Reshaping: It also reshapes the output quantized X from NCHW (Batch Size, Channel, Height, Width) to NHWCB (Batch Size, Height, Width, Channel, Bit) data format with zero padding.
4. Img2Row/Col: Then an Image-to-Row/Column function transforms the 5-dim X (4-dim in binary cases as Bit=1) into a flat matrix considering the convolution stride, so that convolution can be done by GEMM.

Algorithm 3 is a plain bitwise matrix multiplication. PReLU is a non-linear function similar to leaky ReLU but it allows parameterized leakage for negative numbers.

The naive ternary convolution in C/C++ (TNN showed in Algo. 1 & 3) is provided as a baseline on GitHub (https://github.com/yiweifengyan/ASL_TAB/tree/main). Algo.1 is realized by Ternarize() and Img2Row() in the baseline code for clearer logic. The Ternarize() has fused the quantization, bit-packing, and data reshaping. Note that it may be possible to fuse/recombine these computation steps, reshape the data format, and explore different SIMD micro-kernel sizes for better performance.