Lecture 6: Neural Network Model Compression

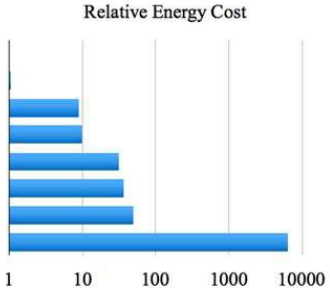
Puneet Gupta

Logistics

- No lecture next Wednesday
  - No presentations
- TA will schedule a project office hour at that time over zoom. Link upcoming.

Why Neural Network Compression?

Operation	Energy [pJ]	Relative Cost
32 bit int ADD	0.1	1
32 bit float ADD	0.9	9
32 bit Register File	1	10
32 bit int MULT	3.1	31
32 bit float MULT	3.7	37
32 bit SRAM Cache	5	50
32 bit DRAM Memory	**640**	**6400**



Energy table for 45nm CMOS process.
[Horowitz, Stanford VLSI Wiki]

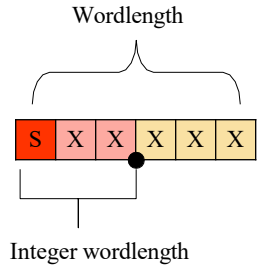
- What is model compression ?
  - Reduce the size of the model (often measured in total weight storage needed)
- Why ?
  - Modern neural networks are very large → Limited Resources on most platforms: memory, CPU, energy, bandwidth
    - may not fit on the device, especially edge/mobile
      - AlexNet
      - 60 million parameters with 5 conv layers and 3 FC layers
      - 240MB of memory to store the parameters
  - Expensive to run → energy cost, especially for battery-operated devices

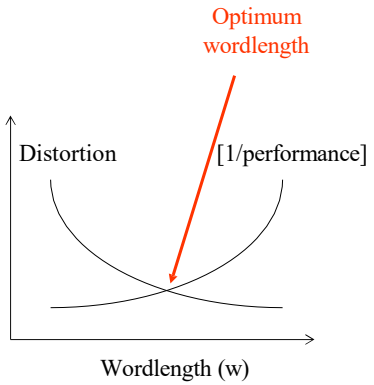
How is a model compressed ?

- Note: this is NOT memory compression with fixed model!
  - You may be able to compress weights or reduce memory traffic by conventional data compression approaches (e.g., Huffman Coding)
- Two categories of approaches
  - Quantization
    - fp32->fp16/bfloat16->fp8 (Training/inference)
    - int16->int8->int4->Ternary->Binary (Inference)
    - Reduce storage and compute
  - Pruning
    - Delete edges, neurons, channels, filters from the network
    - *Can* save storage and compute if done properly

Quantization

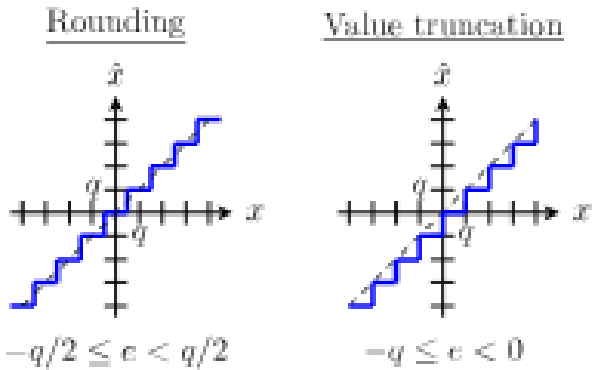
- Float-to-fixed point conversion required to target
  - ASIC and fixed-point digital signal processor core
  - FPGA and fixed-point microprocessor core
- Fixed point format
  - Wordlength (WL); Integer wordlength (IL); Fraction wordlength (FL)
  - FL limits the precision
  - <IL,FL> sets the range to -2^{IL-1},$2^{IL-1}-2^{-FL}$
- Minimize quantization effects
- Avoid overflow
- Find optimum wordlength
  - Longer wordlength
    - May improve application performance
    - Increases hardware cost
  - Shorter wordlength
    - May increase quantization errors and overflows
    - Reduces hardware cost

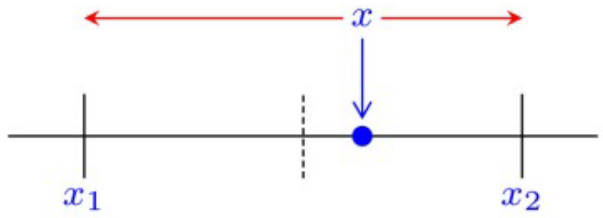


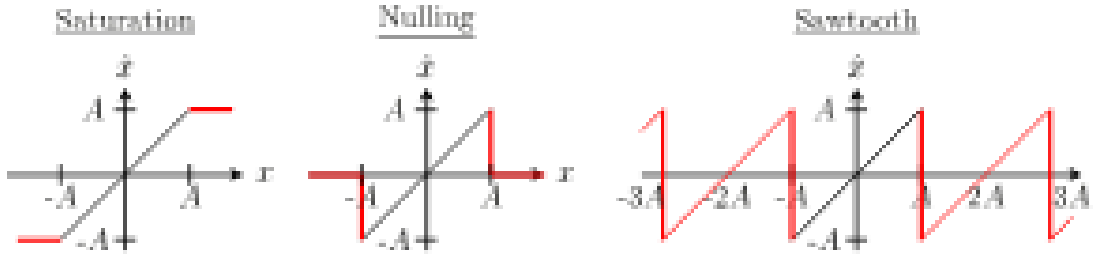


Quantization Effects

- Quantization modes
  - Round
  - Truncation
  - Stochastic Rounding
    - x→x1 with probability (x-x1)/(x2-x1)
    - On average rounded value is x unlike deterministic rounding
- Overflow modes
  - Saturation
  - Saturation to zero (Nulling)
  - Wrap-around (Sawtooth)
    - Preserves difference of two rounded numbers!
- Hardware may not support all modes

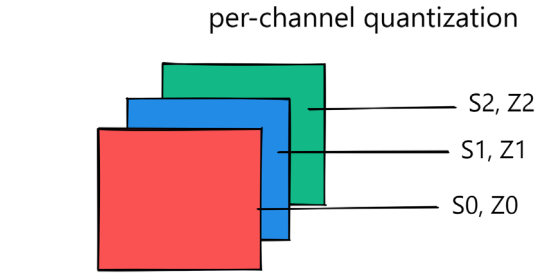


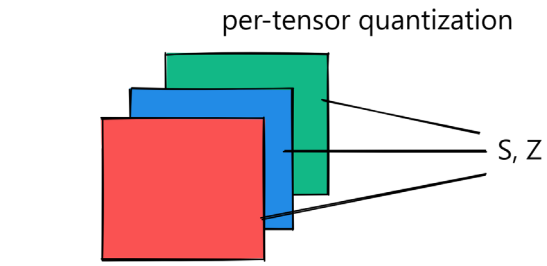


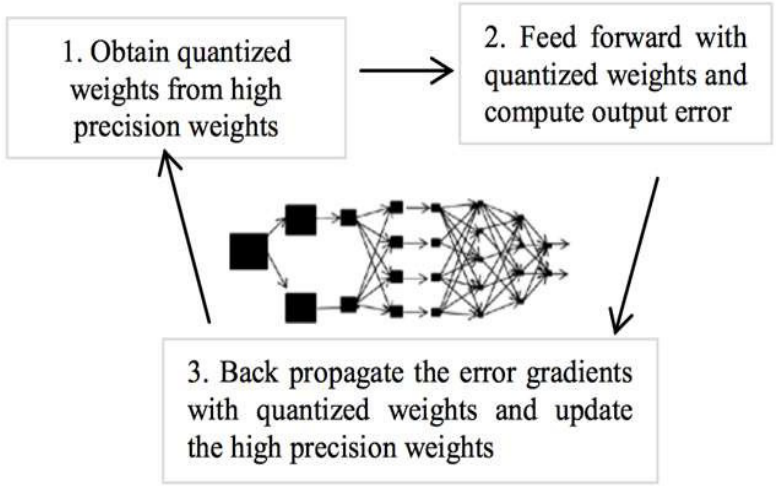


Quantization Contexts

- Typical quantization: $Q(r) = round(r/S + Z)$
  - Scale and bias to select the quantization range
    - S: input_range/output_range
    - Z: to map 0 in input space to 0 in output space
- Post Training Static Quantization
  - Quantize weights (and activations) to required wordlength/bitwidth
- Quantization Aware Training
  - fake quantization to all the weights and activations during the model training
  - higher inference accuracy than the post-training quantization methods

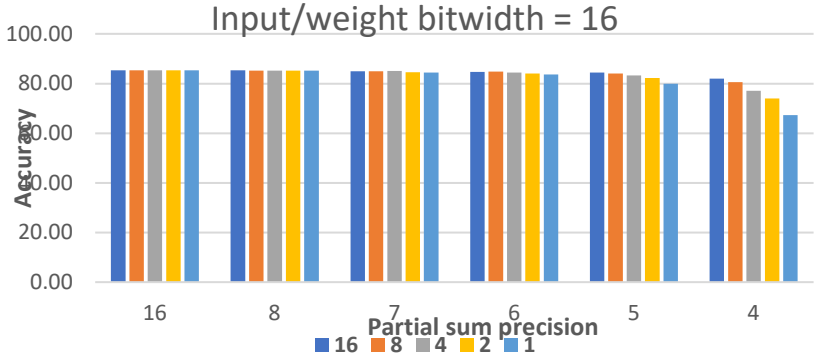


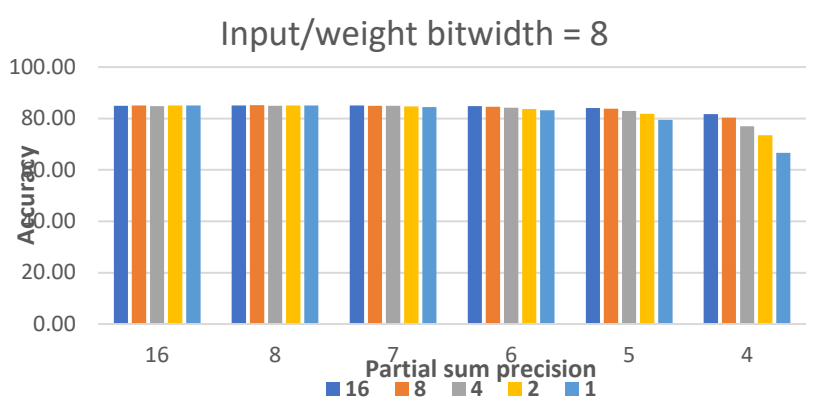


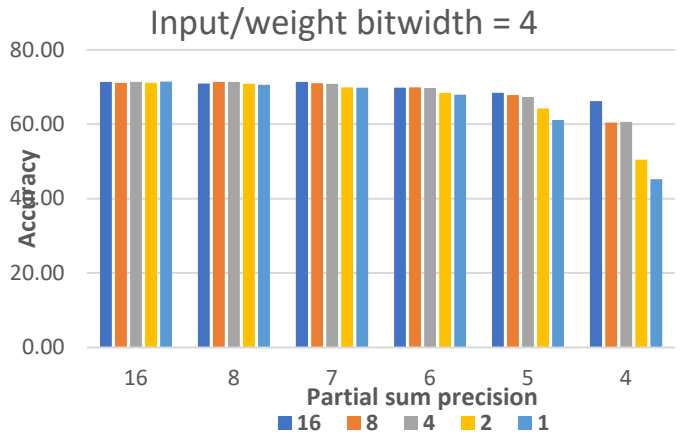


Partial Sum Quantization

- Typically algorithmic dot-product size > hardware MAC size → partial sums are computed
- Digital systems usually do not quantize partial sums
  - E.g. algorithmic filter size: 256; 8b inputs, 8b weights → 24b final output → each MAC in a systolic array would be 24b output →hardware overhead
- Analog systems often quantize partial sums (analog to digital conversion is the quantizer)
  - Larger hardware MAC size → less quantization error
  - E.g., Resnet14 on CIFAR10 (quantization-aware training)
    - Each color is a different hardware dot product size

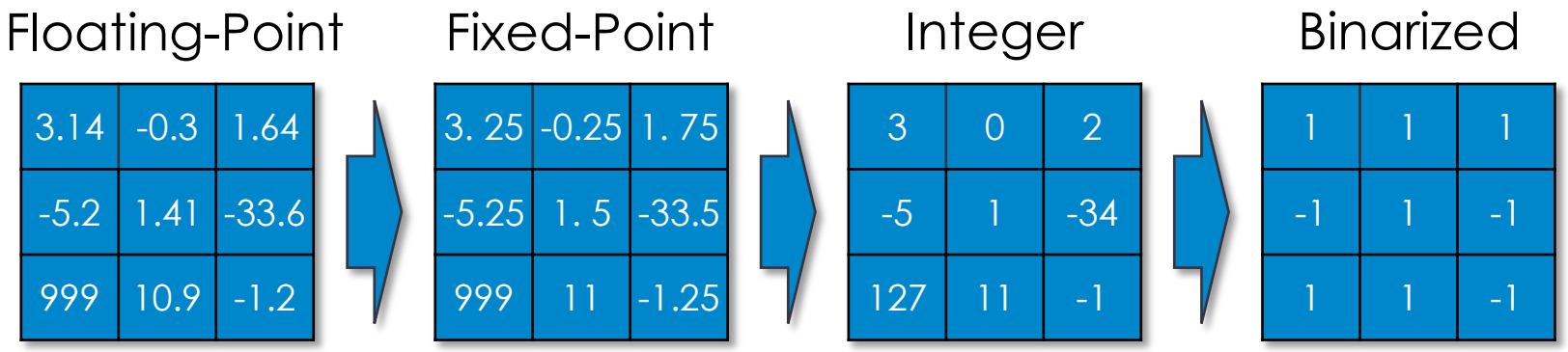






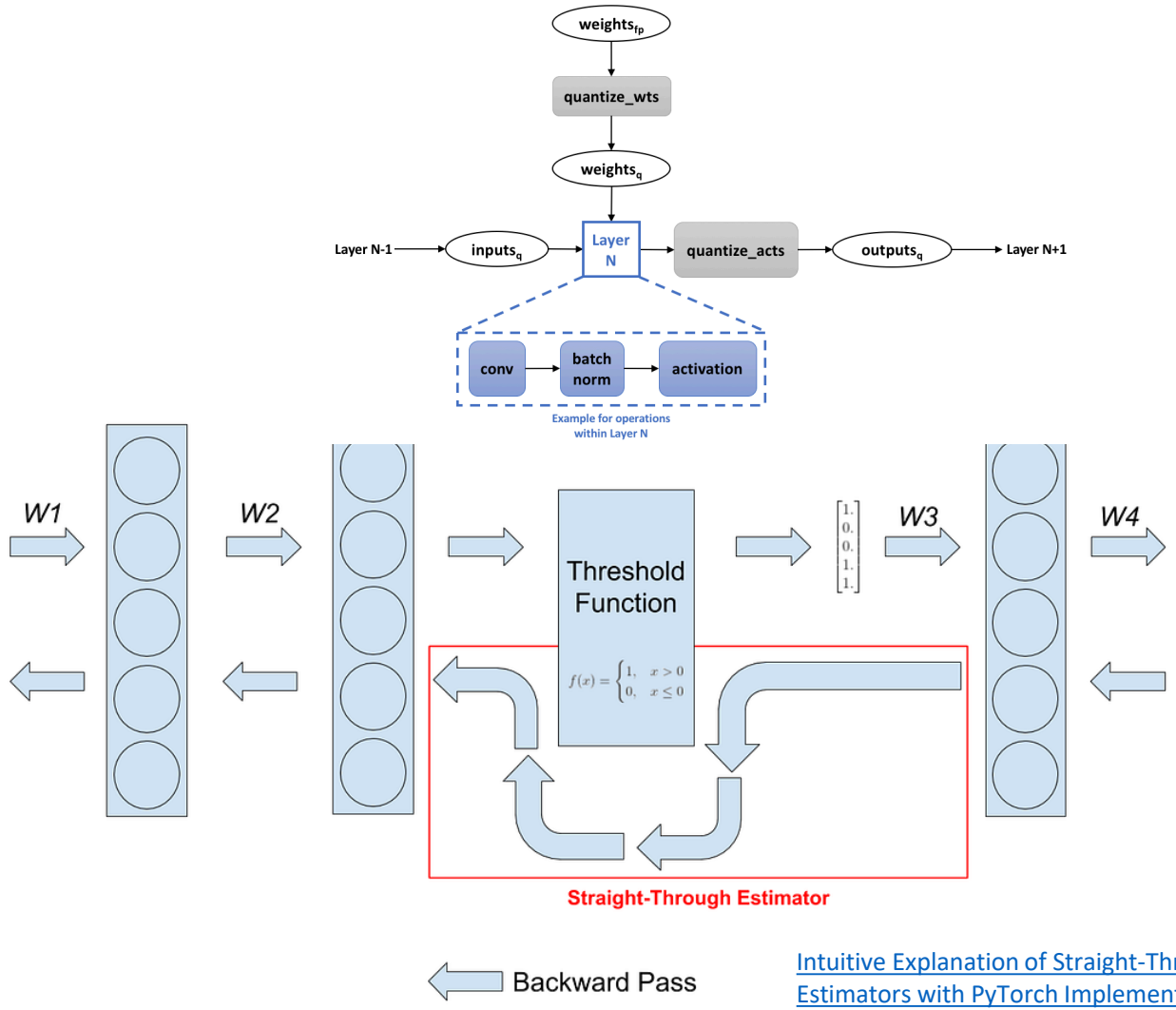
How much quantization ?

- Nvidia T4 example TOPS:
  - 8.1 (FP32), 65 (FP16), 130 (INT8), 260 (INT4)
- Even more extreme quantization
  - Ternary weight networks: weights are -1, 0, +1; activations not quantized
    - MAC replaced by adder/subtractor
  - Binary weight networks: weights are -1, +1
  - XNOR (binarized) networks: weights and activations are -1, +1
    - MULT replaced by bitwise XNOR; Accumulate by popcount (hamming weight)
- Most models have negligible accuracy loss for ~8b *weight* quantization and small loss down to BWNs



Quantization-Aware Training Using STE

- FP weights are preserved during training to accumulate small gradient effects
- STE: Straight Through Estimator
  - Useful when gradient of quantization function is difficult to calculate or not useful (e.g., 0)
  - E.g., for binarization, threshold function gradient is 0
  - In backprop, gradient is just passed along as if quantization function is an identity function



[Intuitive Explanation of Straight-Through Estimators with PyTorch Implementation | by Hassan Askary | Medium]