NYU

Lecture 11:
Hardware Accelerator for DNN Training

Notes

- Project meeting, please sign up!
- Final project presentation
  - I will send out a form for you to select the date and time options.

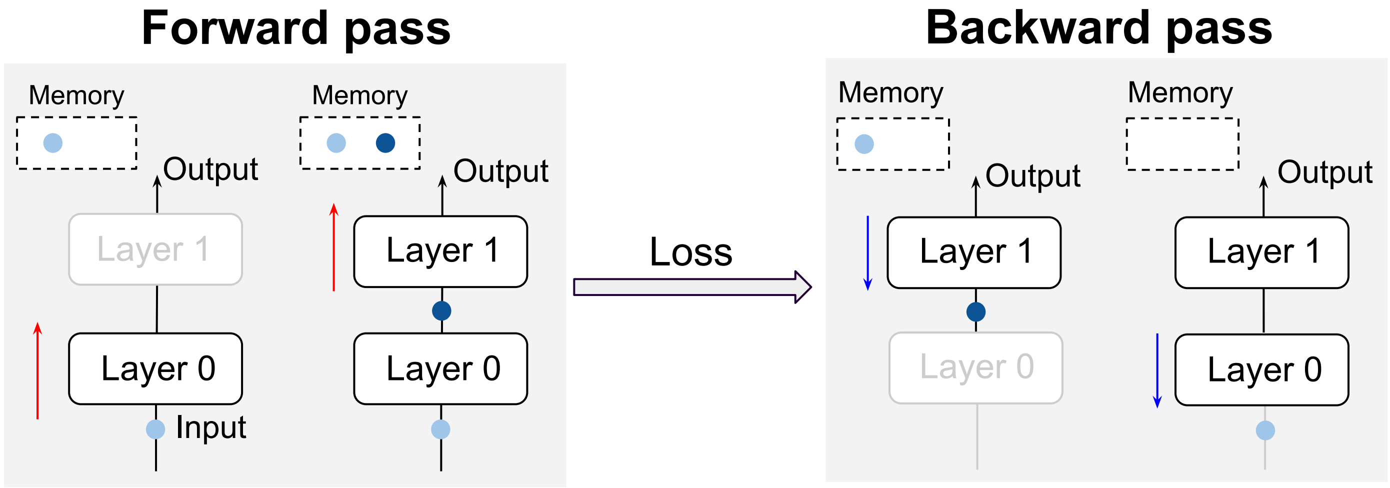
Recap

- Matrix Multiplication with Transposition
- Hardware design for Nonlinear Blocks
- System optimization of LLMs
- Popular transformer accelerator design
  - SpAtten
  - EdgeBert
  - Olive

Topics

- Computation during backward propagation
- Hardware architecture for backward propagation design
- Popular DNN training accelerator design
    - FAST
    - CAMEL
    - ADA-GP
    - Procrustes

Neural Network Training

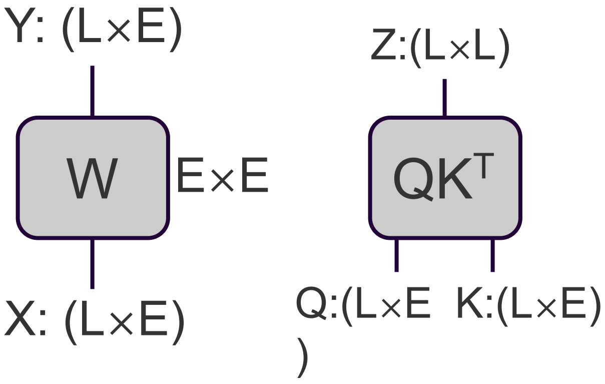


Forward pass

Backward pass

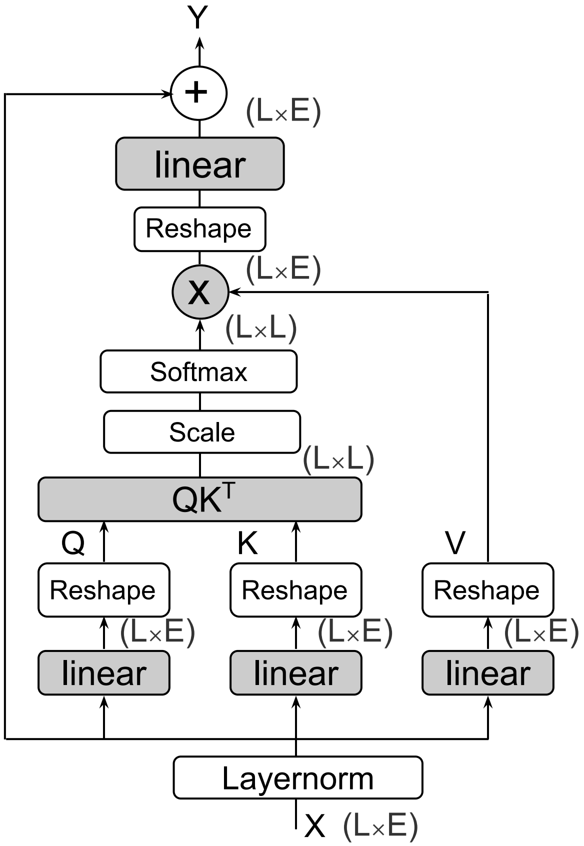
- The peak memory grow linearly as the layer depth increases.
- The backward propagation involves more computations

Training Flow

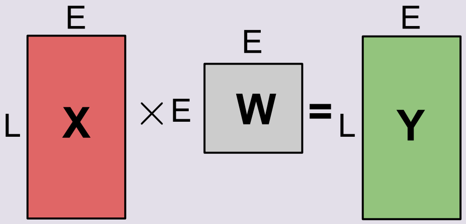


$Y = XW$

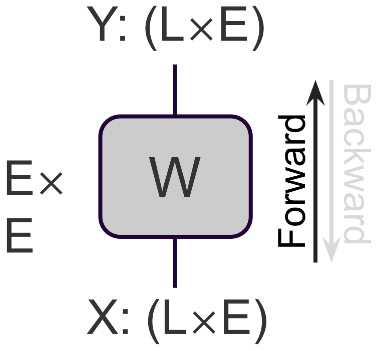
L: number of tokens
E: embedding dimension



Training Flow: Linear Layer

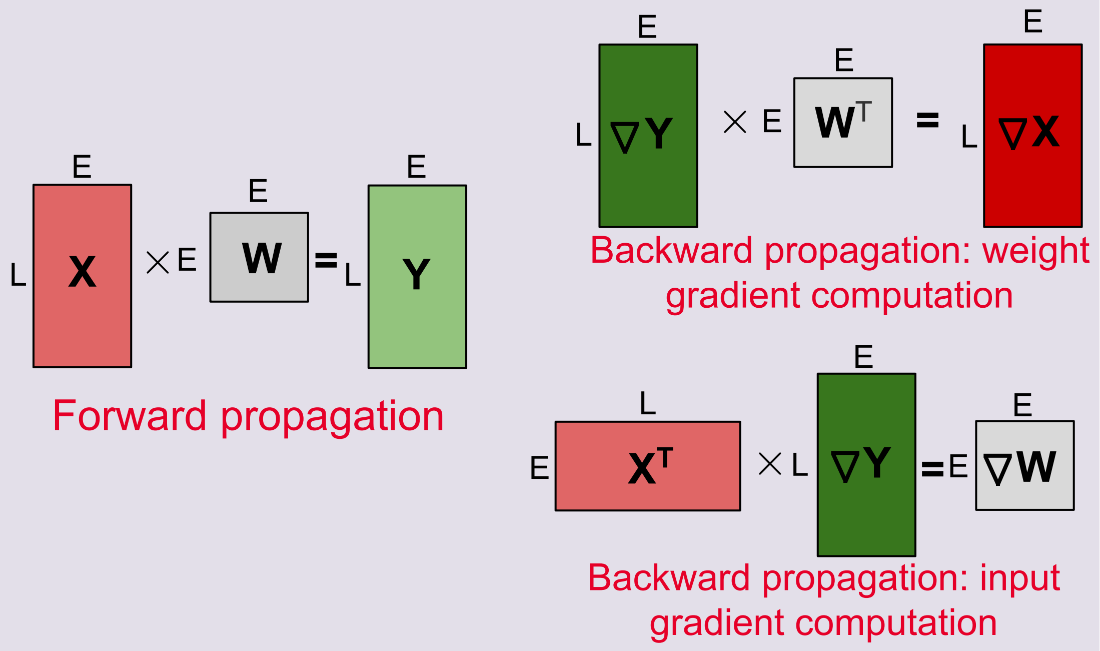


Forward propagation



$Y = XW$

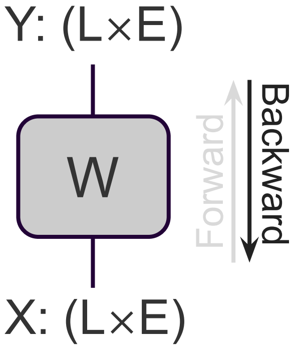
Training Flow: Linear Layer



Forward propagation

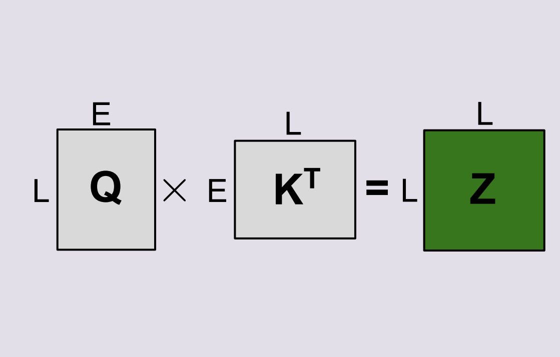
Backward propagation: weight gradient computation

Backward propagation: input gradient computation

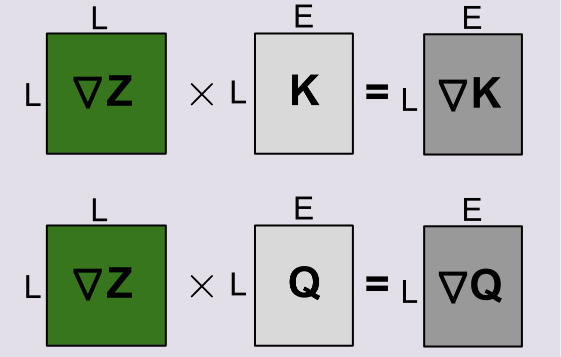


$Y = XW$

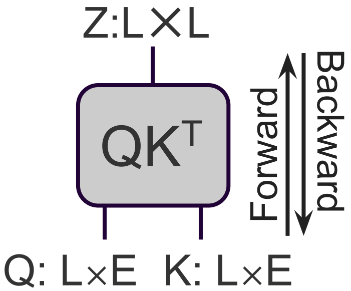
Training Flow: Self-Attention



Forward propagation



Backward propagation



Training Flow: Softmax

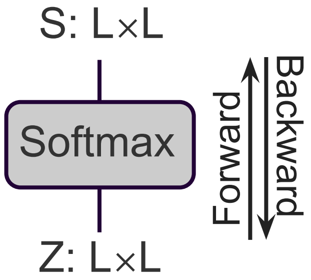
$$s_i = \frac{e^{z_i}}{\Sigma_{j=0}^{N-1} e^{z_j}} \quad For\ i = 1, 2, \cdots, N$$

Forward propagation

$$\frac{\mathrm{ds}}{\mathrm{dz}} = diag(s) - ss^T$$

$$\frac{\mathrm{ds}}{\mathrm{dz}} = \begin{bmatrix} s_1 - s_1^2 & -s_1 \cdot s_2 & -s_1 \cdot s_3 \\ -s_2 \cdot s_1 & s_2 - s_2^2 & -s_2 \cdot s_3 \\ -s_3 \cdot s_1 & -s_3 \cdot s_2 & s_3 - s_3^2 \end{bmatrix}$$

Backward propagation



Training Flow: Normalization

$$\boldsymbol{s} = \alpha \frac{\boldsymbol{z} - \mu_z}{\sigma_z} + \beta$$

$$\boldsymbol{s}_i = \boldsymbol{\alpha} \bar{\boldsymbol{z}}_i + \boldsymbol{\beta}$$

$$\mu_z \quad = \quad \frac{\sum_i z_i}{N}$$

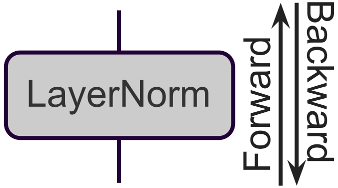
$$\sigma_z \quad = \sqrt{\frac{\sum_i (z_i - \mu_z)^2}{N}}$$

Forward propagation

$$\frac{dL}{d\beta} = \sum_i \frac{dL}{d\boldsymbol{s}_i}$$

$$\frac{dL}{d\alpha} = \sum_i \frac{dL}{d\boldsymbol{s}_i} \bar{z}_i$$

Backward propagation

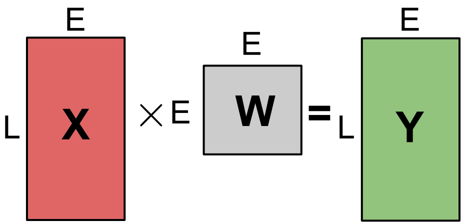


NYU SAI LAB

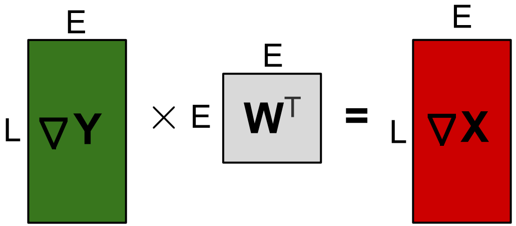
Topics

- Computation during backward propagation
- Hardware architecture for backward propagation design
- Popular DNN training accelerator design
    - FAST
    - CAMEL
    - ADA-GP
    - Procrustes

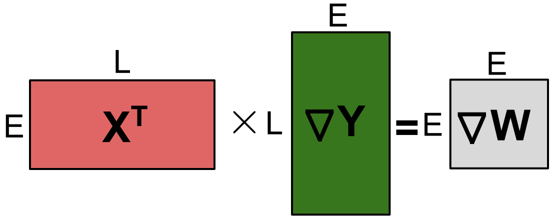
In-place Transposed Matrix Multiplication



Forward propagation



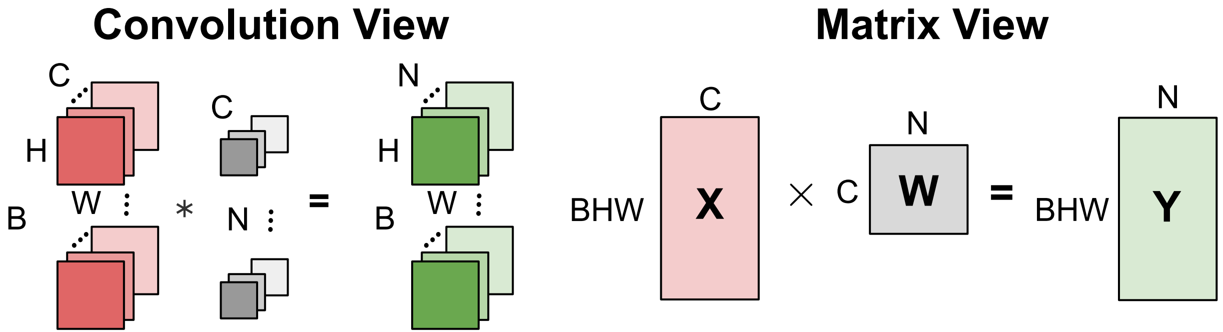
Backward propagation: weight gradient computation



Backward propagation: input gradient computation

Forward Pass for Convolutional Layer

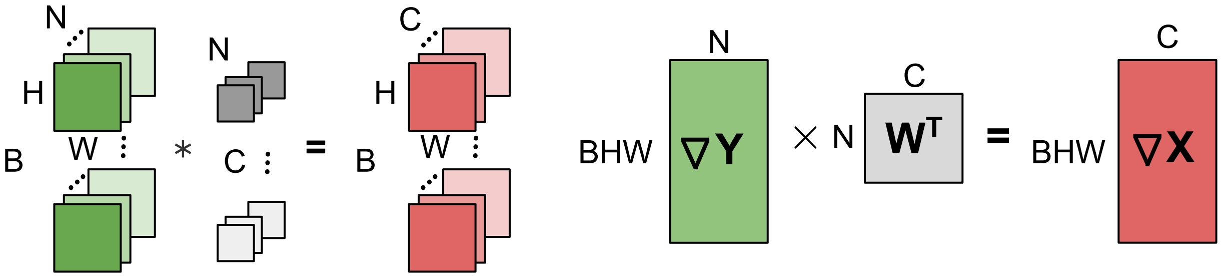
Forward Pass
Compute output **Y**



- Assume a weight kernel size of 1✖1.

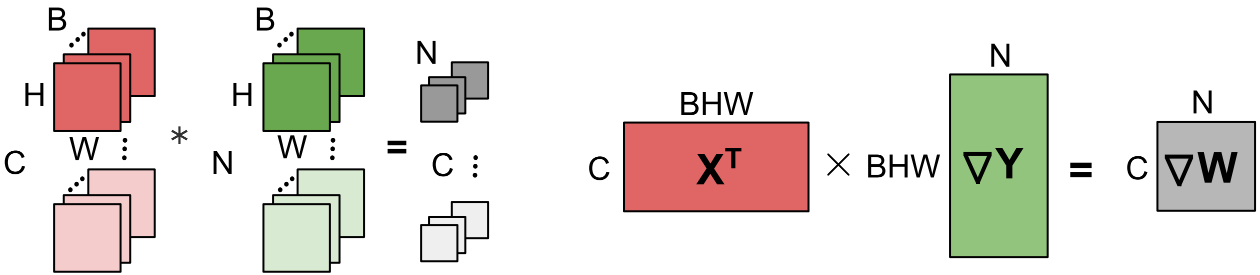
Backward Pass for Convolutional Layer

Backward Pass
Compute Activation gradients **∇X**



Backward Pass for Convolutional Layer

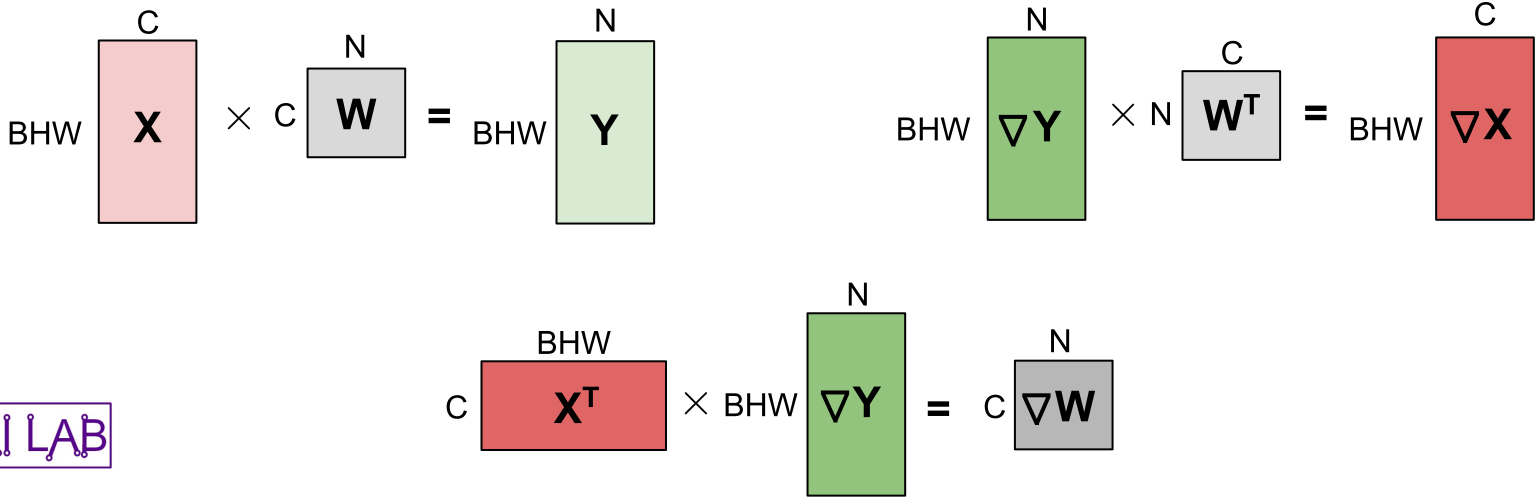
Backward Pass
Compute weight gradients **∇W**



NYU SAI LAB

In-place Transposed Matrix Multiplication

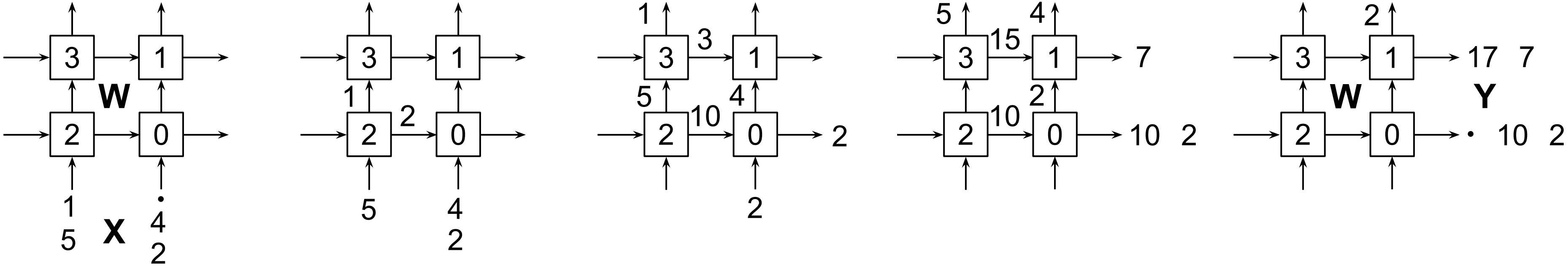
- In the training of neural networks, we need to perform transposed matrix multiplication
- Instead of using a separate hardware for matrix transposition, transposed matrix multiplication can be performed using a systolic array.



In-place Transposed Matrix Multiplication

$$\underset{\mathbf{X}}{\begin{bmatrix}1 & 4\\5 & 2\end{bmatrix}}\underset{\mathbf{W}}{\begin{bmatrix}2 & 3\\0 & 1\end{bmatrix}} = \underset{\mathbf{Y}}{\begin{bmatrix}2 & 7\\10 & 17\end{bmatrix}}$$

- Weight stationary, input from bottom, accumulation from left to right

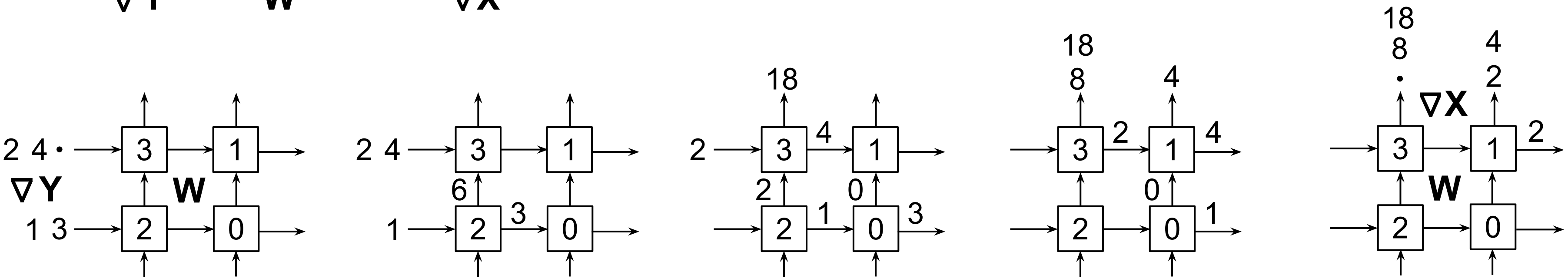


- The weights are preloaded into the systolic array, while the input matrix is streamed into the array from bottom to top.
- The output is produced at the right.

In-place Transposed Matrix Multiplication

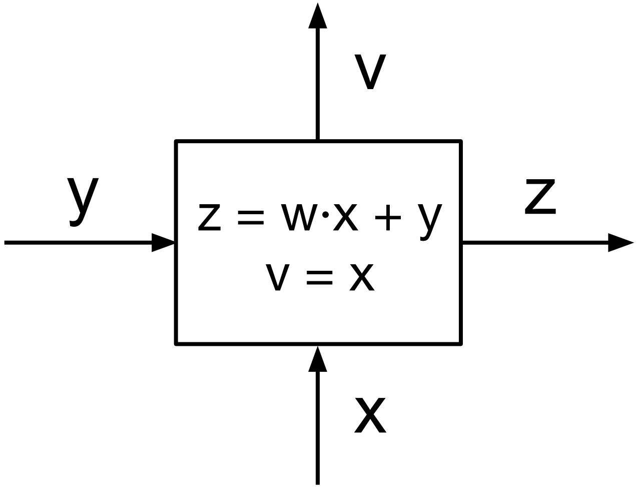
$$\underset{\nabla \mathbf{Y}}{\begin{bmatrix}3 & 4\\ 1 & 2\end{bmatrix}}\underset{\mathbf{W}^{T}}{\begin{bmatrix}2 & 0\\ 3 & 1\end{bmatrix}} = \underset{\nabla \mathbf{X}}{\begin{bmatrix}18 & 4\\ 8 & 2\end{bmatrix}}$$

- Weight stationary, input from left, accumulation upwards



- To compute the input gradient, the data gradient is fed into the systolic array from the left, and the output is produced at the top.

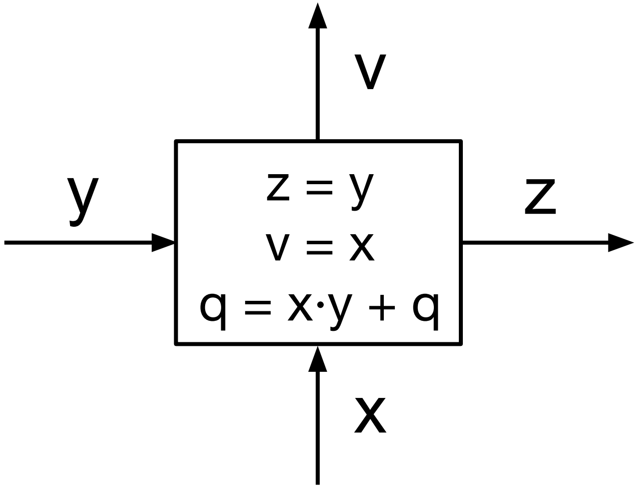
Systolic Array: Weight-Stationary Version



- Takes data (x and y) as input
- w stays in the systolic cell
- Performs a multiply-accumulate operation

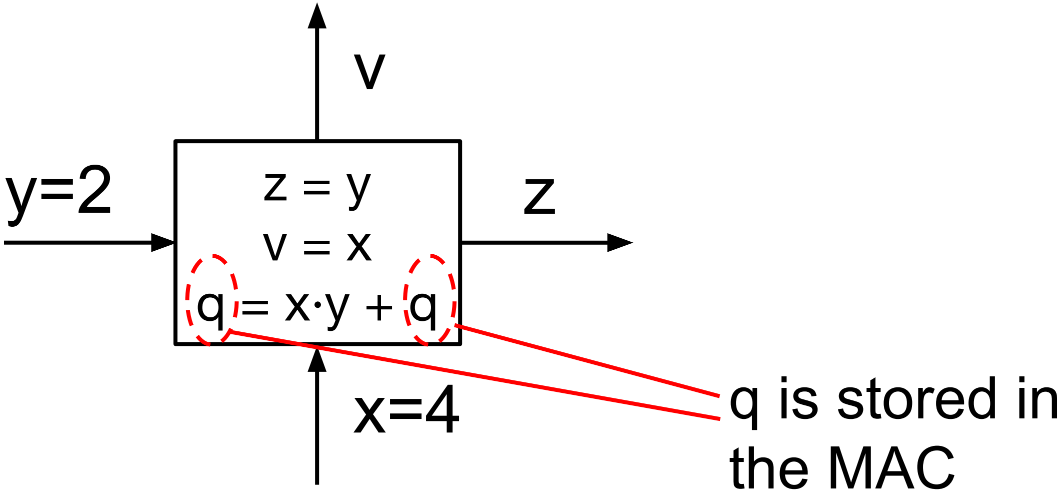
NYU SAI LAB

Systolic Array: Accumulation-Stationary Version

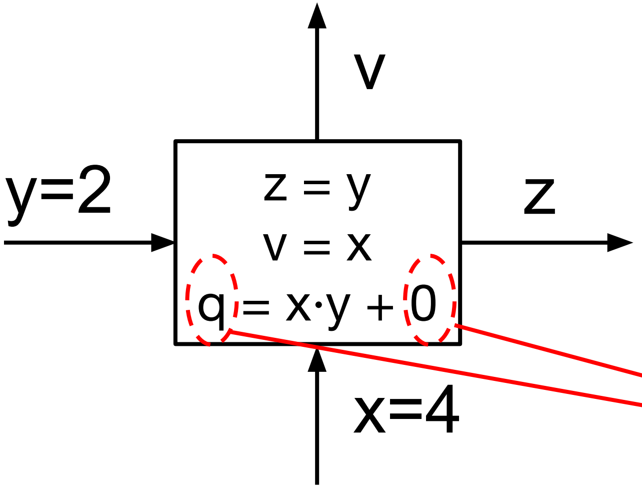


- Takes data (x and y) as input
- Accumulated result q stays in the systolic cell
- Performs a multiply-accumulate operation

Systolic Cell

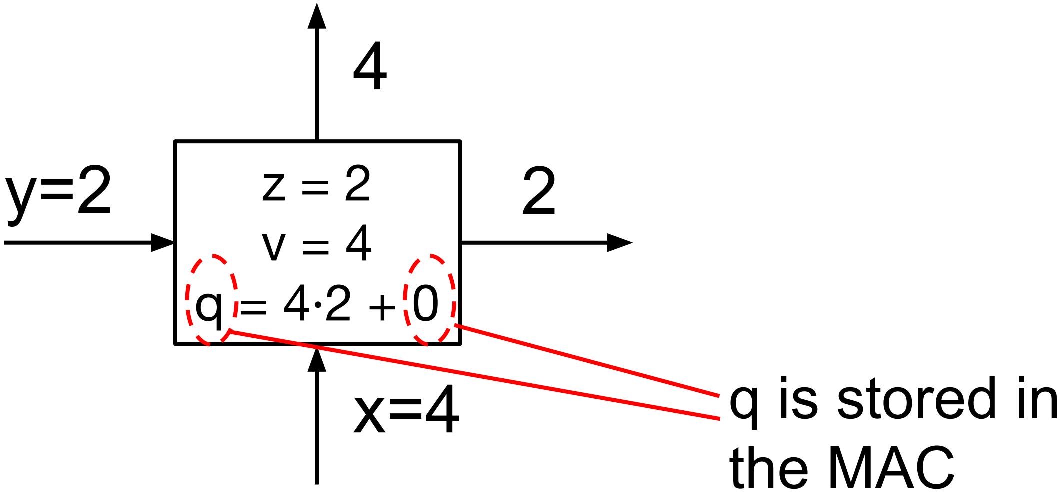


Systolic Cell



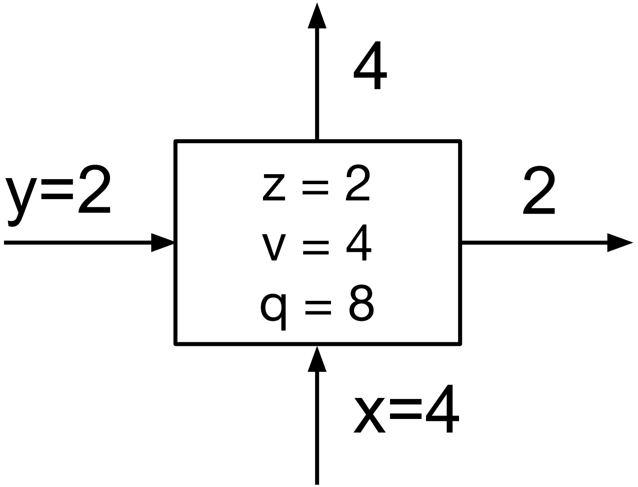
q is stored in the MAC

Systolic Cell



q is stored in the MAC

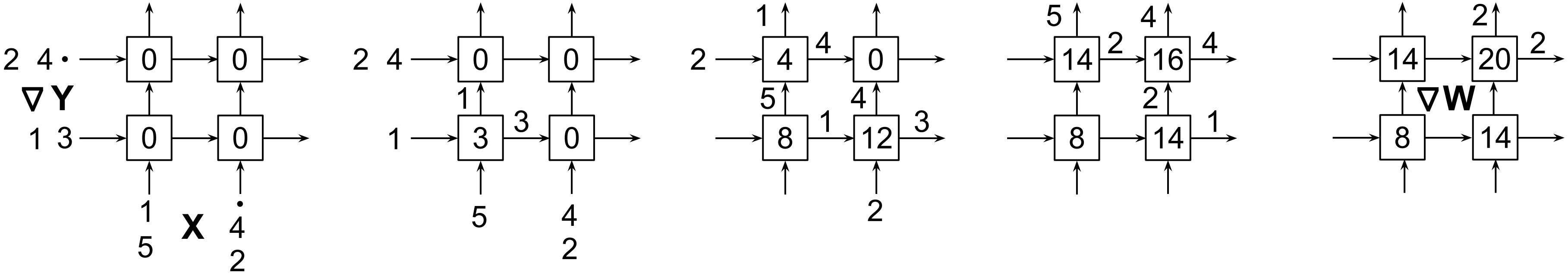
Systolic Cell



In-place Transposed Matrix Multiplication

$$\underset{\mathbf{X^T}}{\begin{bmatrix} 1 & 5 \\ 4 & 2 \end{bmatrix}} \underset{\mathbf{\nabla Y}}{\begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix}} = \underset{\mathbf{\nabla W}}{\begin{bmatrix} 8 & 14 \\ 14 & 20 \end{bmatrix}}$$

- Input from left and bottom, accumulation stationary.

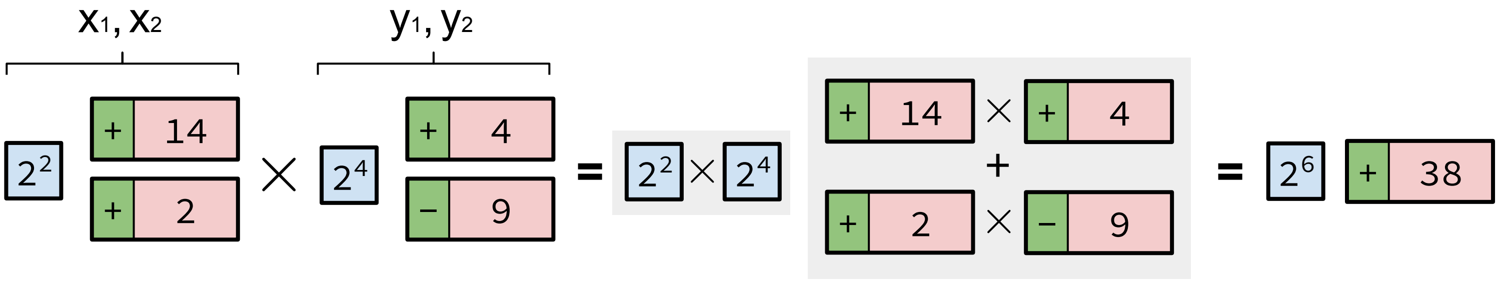


- To compute the weight gradient, the data gradient is input from the left side of the systolic array, while the input activations are fed from the bottom. The resulting weight gradients are accumulated and stored within the systolic cells.

Topics

- Computation during backward propagation
- Hardware architecture for backward propagation design
- Popular dnn training accelerator design
    - FAST
    - CAMEL
    - Tensordash
    - Procrustes

FAST



- The Dot product is performed using BFP format.
- Compare with floating point (FP) format, block floating point (BFP) perform exponent additions and mantissa alignments only at the group level, rather than at the individual elements level.

Fast First, Accurate Second Training

- Previous literature has demonstrated that adding zero-mean Gaussian noise to the weight gradient ∇W can reduce overfitting and improve the convergence of training.

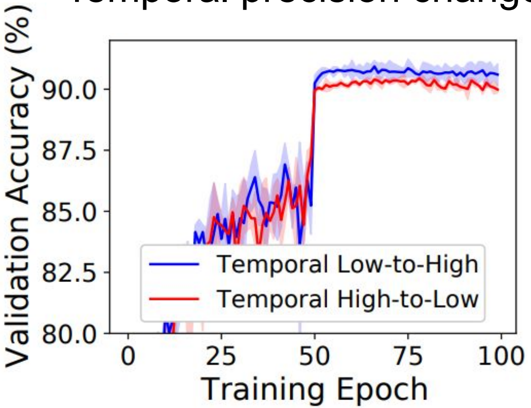
- Decreasing the variance of the noise over iterations achieves better performance than using fixed Gaussian noise throughout training.

- We hypothesize that a similar effect can be achieved by adjusting the BFP precision of weights, activations, and gradients from low to high precisions over training.

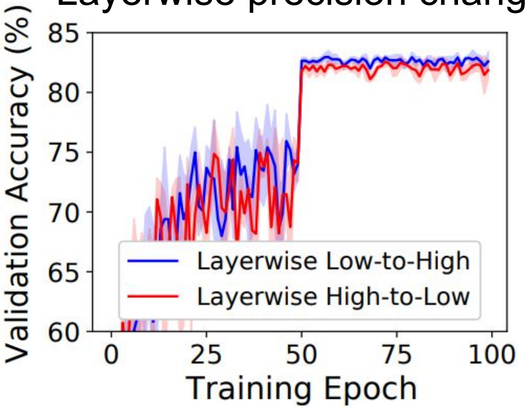
Zhang, Sai Qian, Bradley McDanel, and H. T. Kung. "Fast: Dnn training under variable precision block floating point with stochastic rounding." *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022.

Variable Precision Training

Temporal precision change



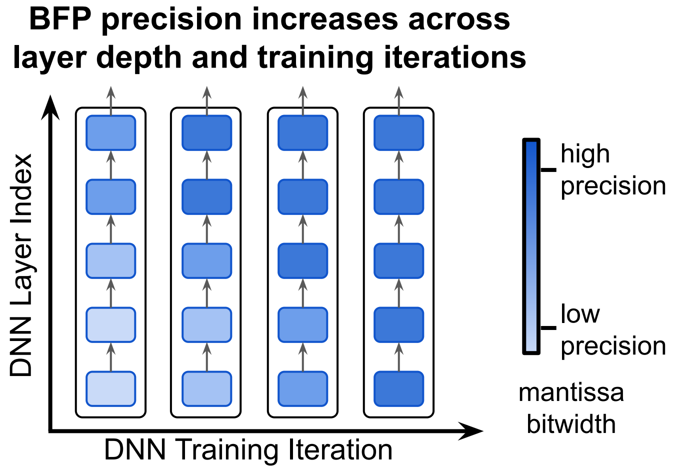
Layerwise precision change



- Under the Temporal High-to-Low scheme, we use FP32 for weights, activations, and gradients for the first part of training, and lower-precision BFP for the second part of training

- Under the Layerwise High-to-Low scheme, we use FP32 precision for the first ten layers, and lower-precision BFP for later layers

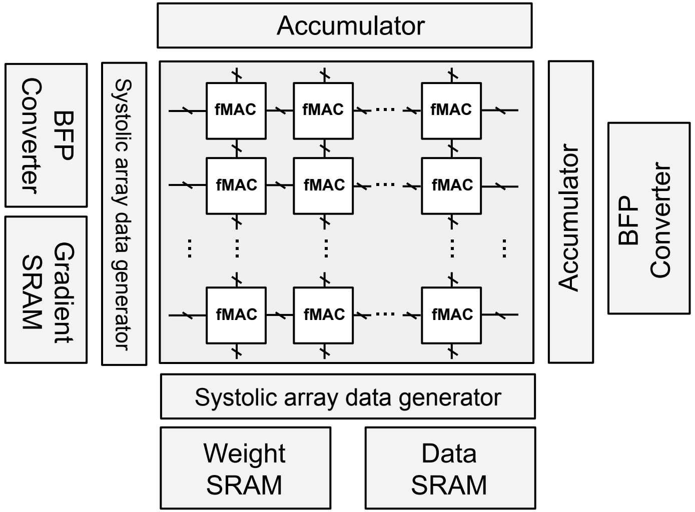
Variable Precision Training



- We progressively increase the BFP precision of weights, activations, and gradients along both layer depth and training iterations
- We name this approach **FAST** (Fast First, Accurate Second Training)

NYU SAI LAB

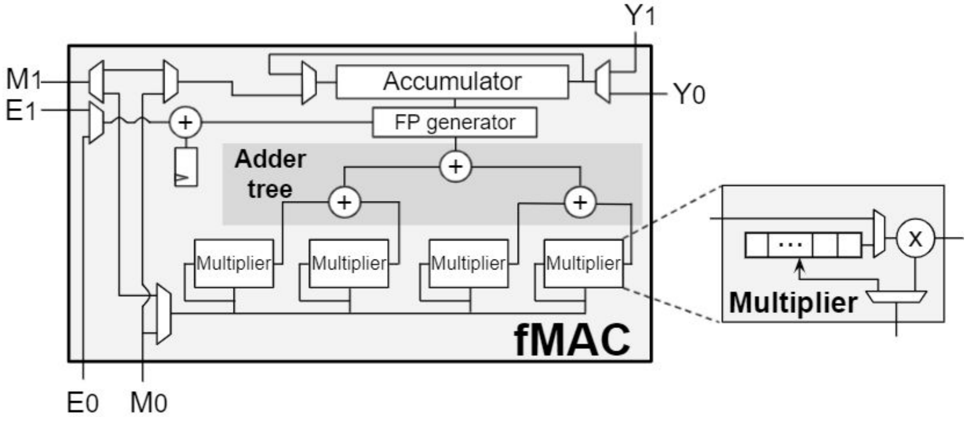
FAST System Design



- Major components of FAST system:
  - Systolic array with FAST multiplier and accumulator (fMAC)
  - BFP converter
  - Accumulator and systolic array input generator
  - Memory subsystem

NYU SAI LAB

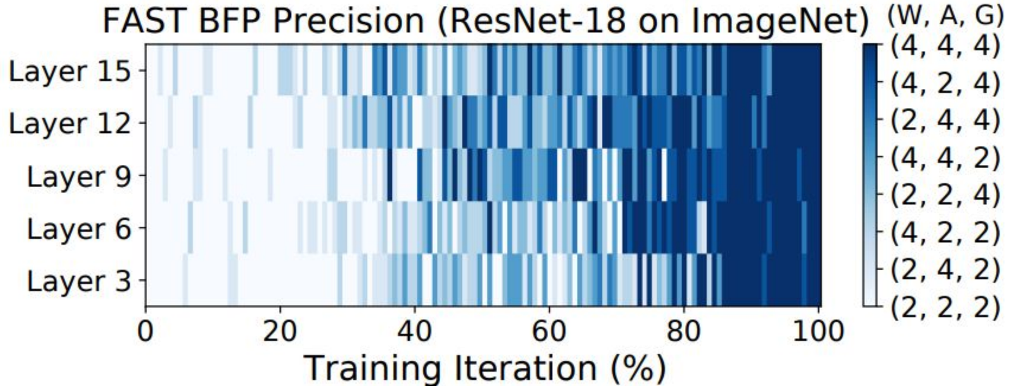
FAST System Design



- fMAC operates on chunks of BFP mantissas (e.g., 2-bit chunks) to support variable-width mantissas in 2-bit increments

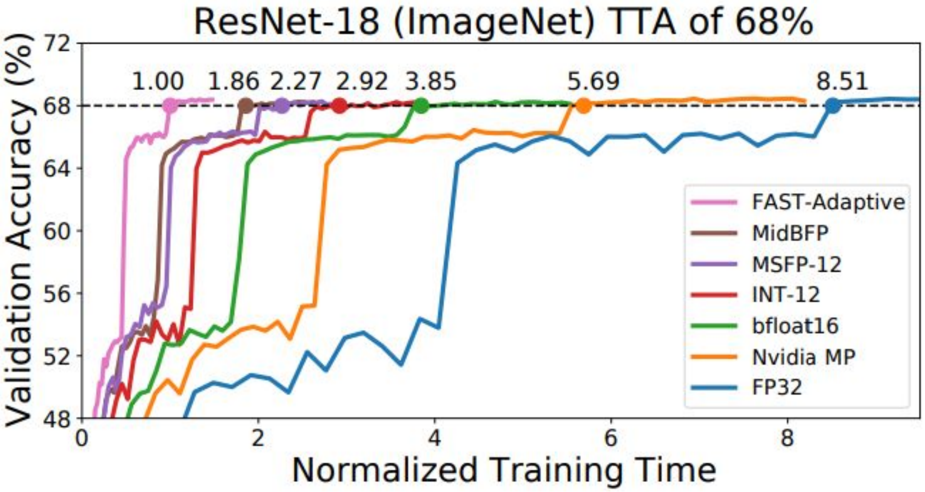
NYU SAI LAB

Evaluation



- FAST progressively increases the BFP precision across both layer depth and iterations during the training process

Evaluation



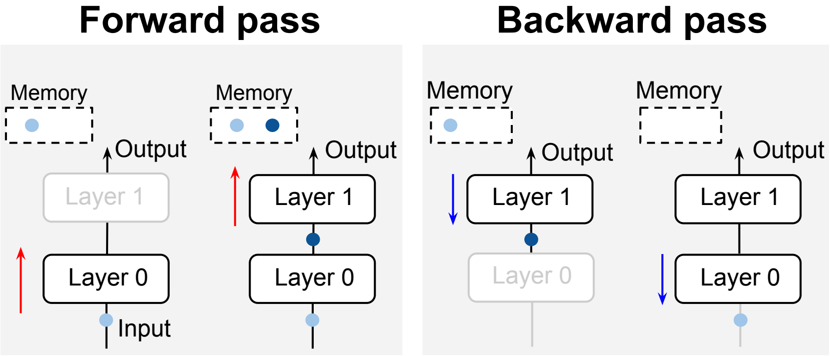
- We use Time-to-Accuracy (TTA) as the evaluation metric to compare different approaches
- Our FAST approach achieves the lowest TTA across all the numeric formats

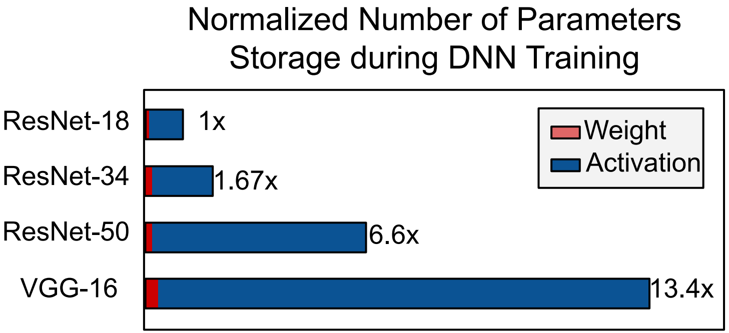
NYU SAI LAB

Topics

- Computation during backward propagation
- Hardware architecture for backward propagation design
- Popular dnn training accelerator design
    - FAST
    - CAMEL
    - Tensordash
    - Procrustes

Memory Efficient Neural Network Training





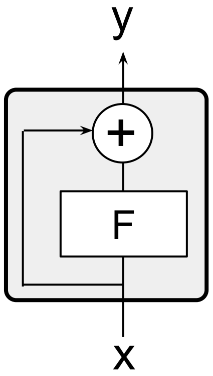
- The memory footprint grows proportional with the layer depth.
- On top of this, small edge devices typically have limited on-chip storage, leading to frequent and costly accesses to off-chip memories.

NYU SAI LAB

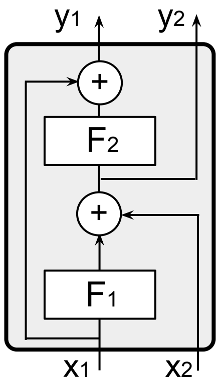
Zhang, Sai Qian, et al. "CAMEL: Co-Designing AI Models and eDRAMs for Efficient On-Device Learning." *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024.

Memory Efficient Neural Network Training

Residual Architecture



Reversible Architecture



Forward pass:

$y_2 = F_1(x_1) + x_2$

$y_1 = F_2(y_2) + x_1$

Backward Pass:

$x_1 = y_1 - F_2(y_2)$

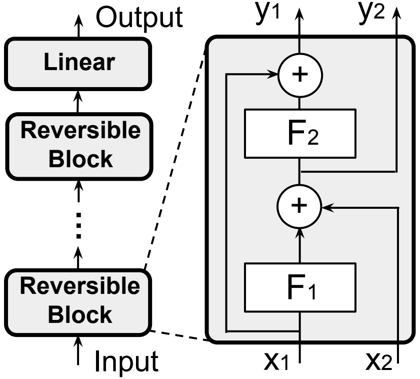
$x_2 = y_2 - F_1(x_1)$

- A reversible residual network (RevNet) is a variant of the canonical residual neural network (ResNet).

Zhang, Sai Qian, et al. "CAMEL: Co-Designing AI Models and eDRAMs for Efficient On-Device Learning." *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024.

Memory Efficient Neural Network Training

Architecture



Operations

Forward pass:

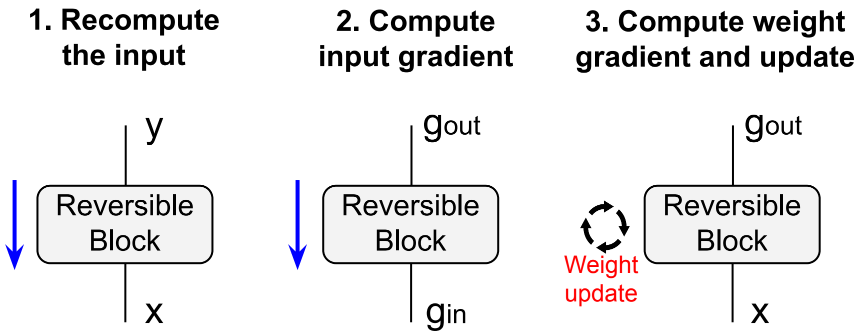
$y_2 = F_1(x_1) + x_2$

$y_1 = F_2(y_2) + x_1$

Backward Pass:

$x_1 = y_1 - F_2(y_2)$

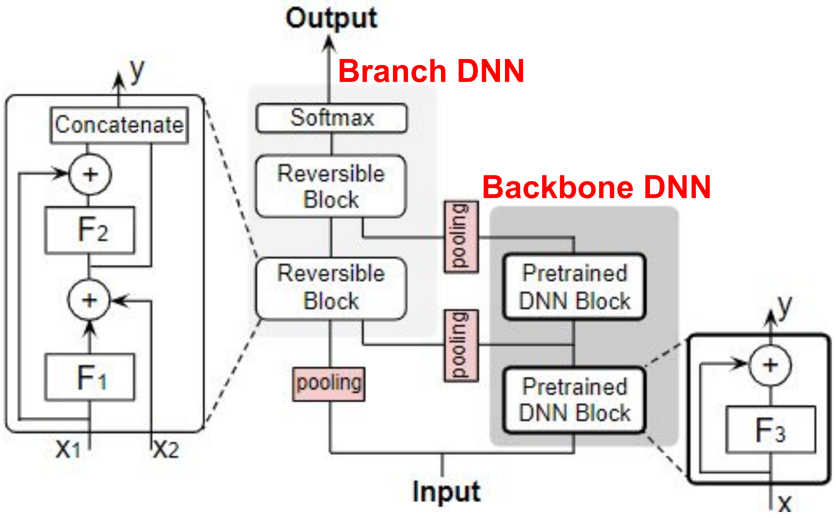
$x_2 = y_2 - F_1(x_1)$



- The reversible architecture enables the backward pass computations to be performed without the need to store the input activations.
- Given the output y, the input activations are first recomputed. Afterwards, the input and weight gradients are computed with standard backward pass operations.

Zhang, Sai Qian, et al. "CAMEL: Co-Designing AI Models and eDRAMs for Efficient On-Device Learning." *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024.

Memory Efficient Neural Network Training



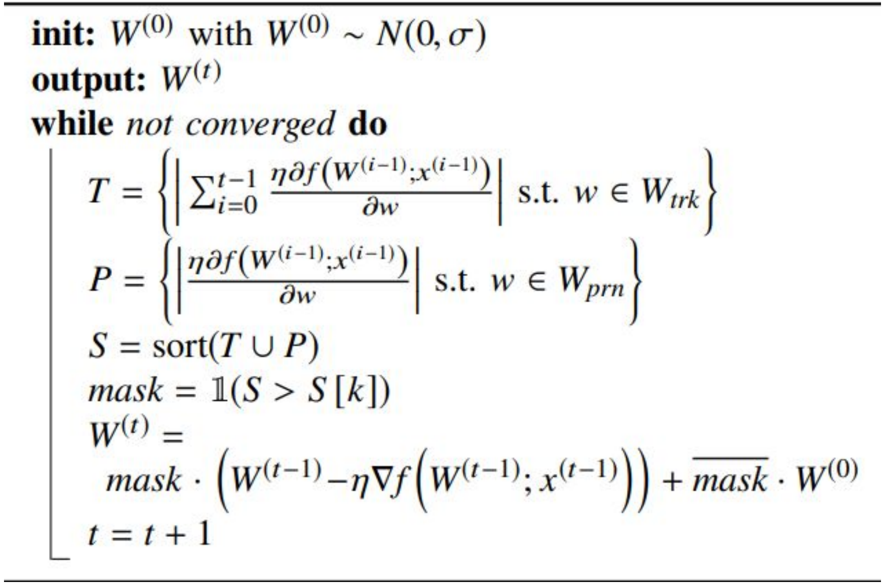
Duplex DNN

- This approach in turn imposes higher compute demands.
- We propose to judiciously train a subset of the model parameters to minimize training.
- The backbone DNN is frozen during the backward pass of the DNN.
- The normalization layers are removed from the branch DNN to facilitate the training process.

NYU SAI LAB

Zhang, Sai Qian, et al. "CAMEL: Co-Designing AI Models and eDRAMs for Efficient On-Device Learning." *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024.

Procrustes

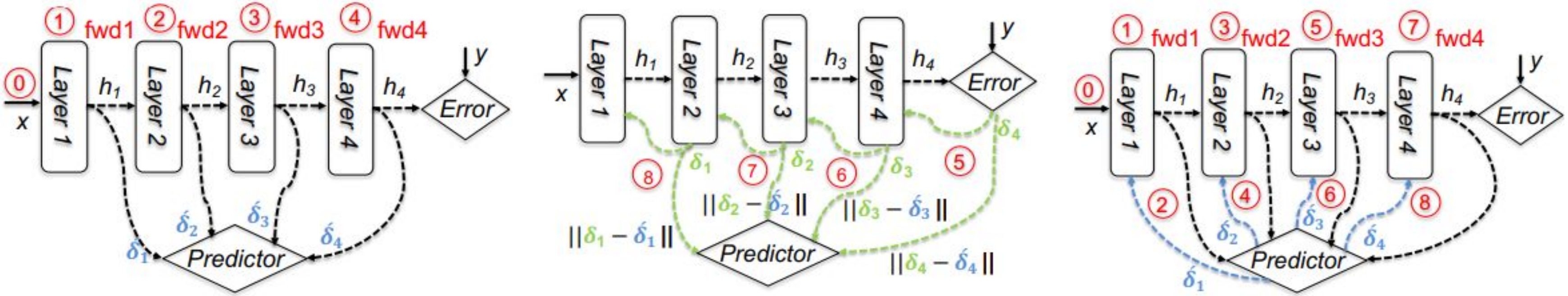


```
init: $W^{(0)}$ with $W^{(0)} \sim N(0, \sigma)$
output: $W^{(t)}$
while not converged do
    $T = \left\{ \left| \sum_{i=0}^{t-1} \frac{\eta \partial f(W^{(i-1)};x^{(i-1)})}{\partial w} \right| \text{ s.t. } w \in W_{trk} \right\}$
    $P = \left\{ \left| \frac{\eta \partial f(W^{(i-1)};x^{(i-1)})}{\partial w} \right| \text{ s.t. } w \in W_{prn} \right\}$
    $S = \text{sort}(T \cup P)$
    $mask = \mathbb{1}(S > S[k])$
    $W^{(t)} =$
      $mask \cdot \left( W^{(t-1)} - \eta \nabla f\left( W^{(t-1)}; x^{(t-1)} \right) \right) + \overline{mask} \cdot W^{(0)}$
    $t = t + 1$
```

- We adapt a sparse training algorithm to be amenable to hardware acceleration; we then develop dataflow, data layout, and load balancing techniques to accelerate it.
- Only a fixed percentage of the parameters (e.g., 10%) are ever allowed to change
- During the each training iteration, the weights with the highest accumulated gradient survive

Yang, Dingqing, et al. "Procrustes: a dataflow and accelerator for sparse deep neural network training." *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020.

ADA-GP



- We propose ADA-GP, which uses gradient prediction adaptively to speed up DNN training while maintaining
  accuracy.
- ADA-GP works by incorporating a small neural network to predict gradients for different layers of a DNN model.

NYU SAI LAB

Janfaza, Vahid, et al. "ADA-GP: Accelerating DNN Training By Adaptive Gradient Prediction." *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 2023.

Presentations

- On-Device Training Under 256KB Memory (Ankit)
- FlashDecoding++: Faster Large Language Model Inference on GPUs (Akshay, Su)
- CAMEL: Co-Designing AI Models and eDRAMs for Efficient On-Device Learning (Jahnavi, Isha)
- Procrustes: a Dataflow and Accelerator for Sparse Deep Neural Network Training (Mohnish, Dae Sung)