Lecture 0: Introduction and Course Overview

ECE-GY 9483 / CSCI-GA 3033
Special Topics: Efficient AI and Hardware Accelerator Design

Self Introduction

- Assistant Professor, NYU, ECE & CS, lead **System & AI (SAI) lab**.
- A senior research scientist at Meta, 2022-2024.
- Academic trajectory
  - University of Toronto
    - Bachelor and Master in ECE
    - Master in Statistics
  - Harvard University
    - PhD in CS
- Research Interest:
  - Efficient AI Algorithm
  - AI Hardware Accelerator
  - AR/VR System

NYU SAI LAB

Course Information

- Course website: https://www.saiqianzhang.com/COURSE/
- I use Brightspace to post announcements and grades
- I provide an online zoom meeting option for people interested in auditing the class. However, enrolled students are required to attend in person unless special condition.
- Discussion groups has been created in the Brightspace
- Course email: efficientaiaccelerator@gmail.com

Course Information

- The course will involve 13 lectures, 3 coding assignments, 1 final project, 1 midterm exam and in-class quiz.
  - In-class quiz (15%)
  - In-course presentation (5%)
  - Assignments (30%): total three of them, each counts 10%
  - Midterm (25%)
  - Final project (25%)
    - Project Proposal (5%) (1 page)
    - Final Presentation (10%)
    - Final Report (10%)
- Readings:
  - Course notes and papers (optional)
  - (reference) Goodfellow, Ian. "Deep learning." (2016). https://www.deeplearningbook.org/
- Lecture time:
  - Friday: 5:00pm-7:30pm
- Office hour:
  - Friday: 1:30pm-2:30pm, or by appointment (Zoom)

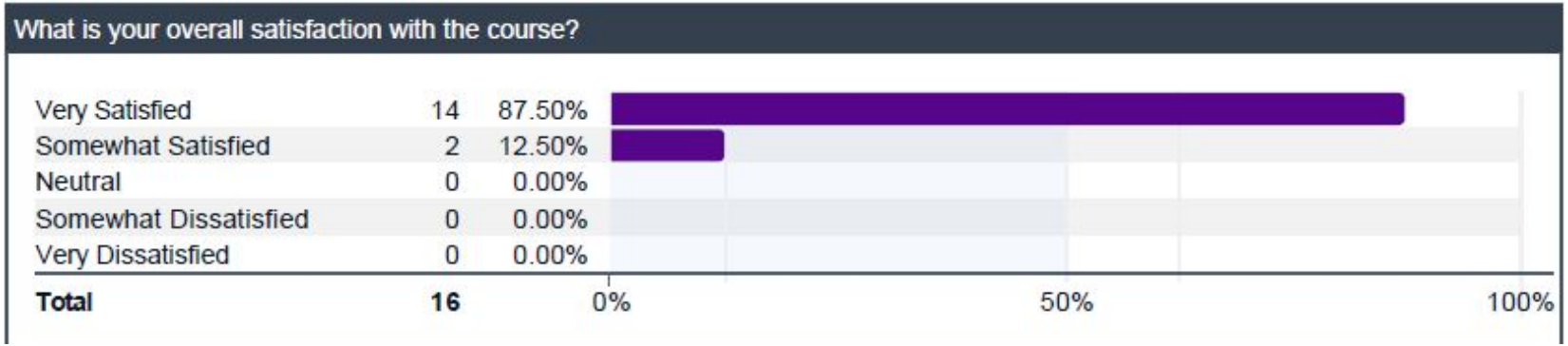
In-Course Presentation General Policy

- Please use Google Slides to create your presentation slides.
- Sign up here: https://docs.google.com/spreadsheets/d/1QL7qBQnMluk-uTniPeH0H6i4ij40hVbuOo0ACXBCMV4/edit?usp=sharing
- There is limit on the number of slides (10 pages for algorithmic paper, 14 pages for architectural paper), make sure to stay within the presentation time limit (15 mins for algorithm paper, 25 mins for arch paper).
- Submission Deadline: Please send the link to your Google Slides presentation by Friday before 2:00 PM each week to Shawn Yin (xy2053@nyu.edu)
- Please ensure that the Google Slides link is set to 'Anyone with the link can view' so that it is accessible to us.

In-Course Presentation Format

- Paper Presentation (3 persons, 15 mins for Algorithm paper, 25 mins for Arch paper):
  - Content
    - Introduction
    - Background
    - Methodology
    - Evaluation
    - Your thoughts & Discussion
  - Evaluation criteria
    - Clarity (3): Did the presenter articulate the main goals of their research or analysis?
    - Structure and Flow (2): Was the presentation logically organized?
    - Depth of Analysis (3): Did the presenter demonstrate a good understanding of the paper?
    - Discussion with audience (2)
    - The grade may differ based on the individual performance

Course Feedback from Previous Semesters



NYU SAI LAB

Course Feedback from Previous Semesters

Comments
Amazingly built course on all fronts, format/ teaching style/ resources etc. Very engaging and pleasurable experience
The course is very practical and cutting–edge. Is a good introduction course for student who want to take part in related research. It would be better taken in the afternoon or earlier. Cause the content is a lot and need high attention to catch up with the pace. In my opinion, I'm feeling a little bit tired during the left half of the lecture.
Amazing course. Professor Sai is doing a great job. Course structure, quizzes, midterm, assignments, project–All were super helpful and great. Also, very nice, polite, understanding and approachable professor.
Professor is really nice and helpful in explaining any topic. Best professor he is

Course Assistant/Grader

Shawn Yin (CA)

Office hour: Monday
1:00pm-2:00pm
(Zoom)

Yifei Feng (CA)

Office hour: Wednesday
11:00am-12:00pm
(Zoom)

Handong Ji (Grader)

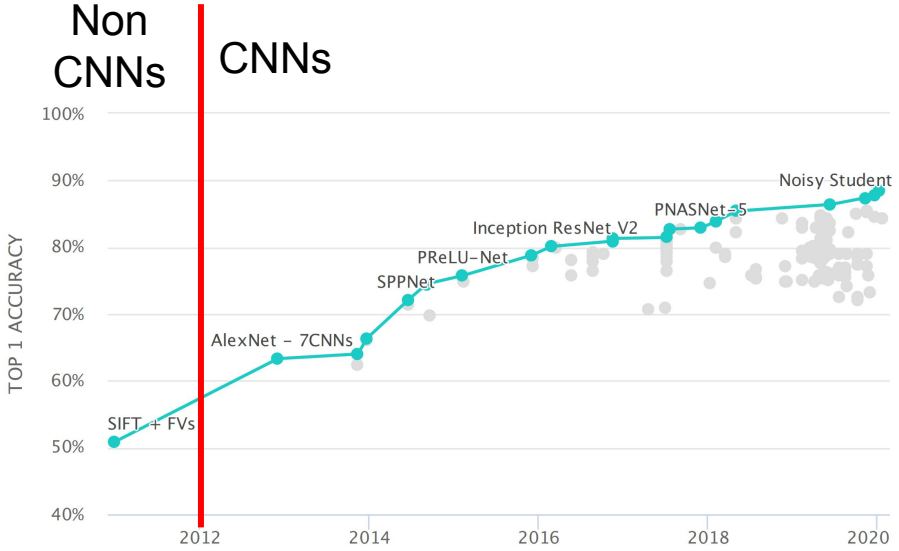
NYU SAI LAB

Life is Powered by Deep Learning

- Deep Neural Networks (DNNs) have achieved state-of-the-art performance across a variety of domains
  - Image Recognition
  - Video Processing
  - Natural Language Processing
  - Autonomous Driving

Convolutional Neural Network (CNN) → ‘rose’

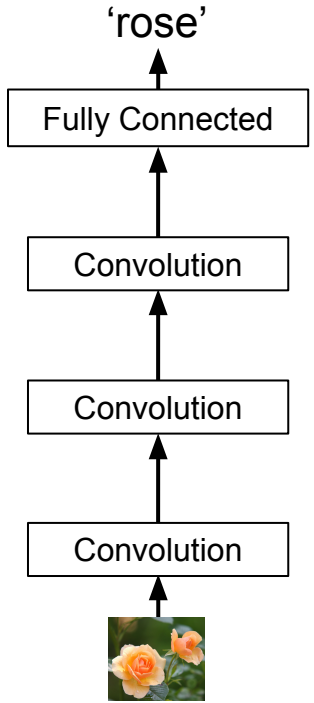
Image Classification



- More desirable modern services are enabled by DNN

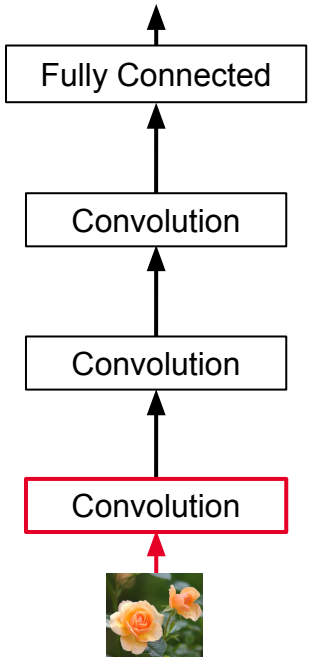
How Deep Neural Network is Executed?

- Use a Convolutional Neural Network (CNN) as an example
- This CNN contains four layers
  - 3 convolutional layers
  - 1 fully connected layer



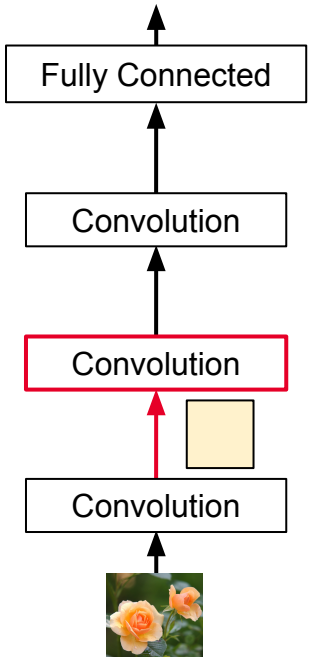
CNN with 4 layers

How Deep Neural Network is Executed?



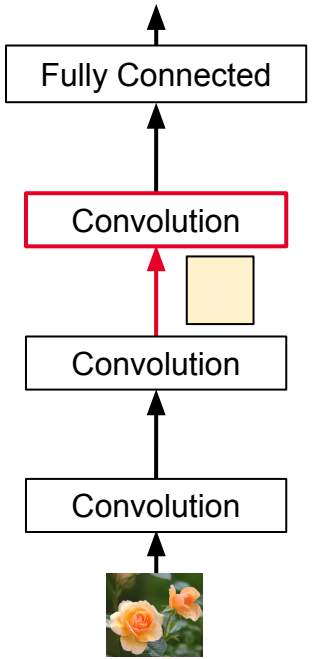
CNN with 4 layers

How Deep Neural Network is Executed?



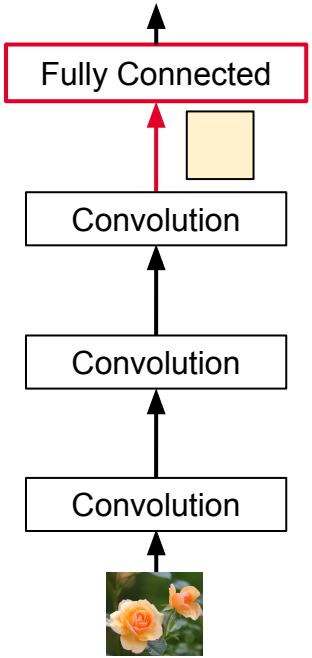
CNN with 4 layers

How Deep Neural Network is Executed?



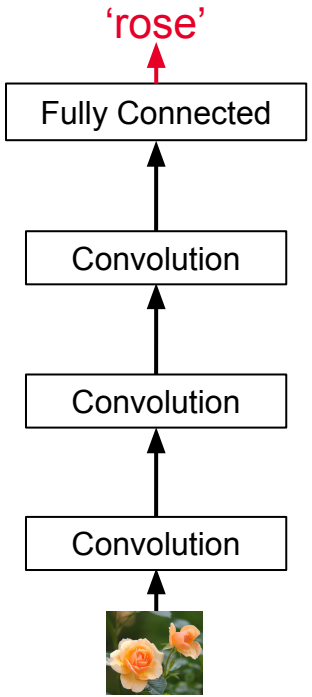
CNN with 4 layers

How Deep Neural Network is Executed?



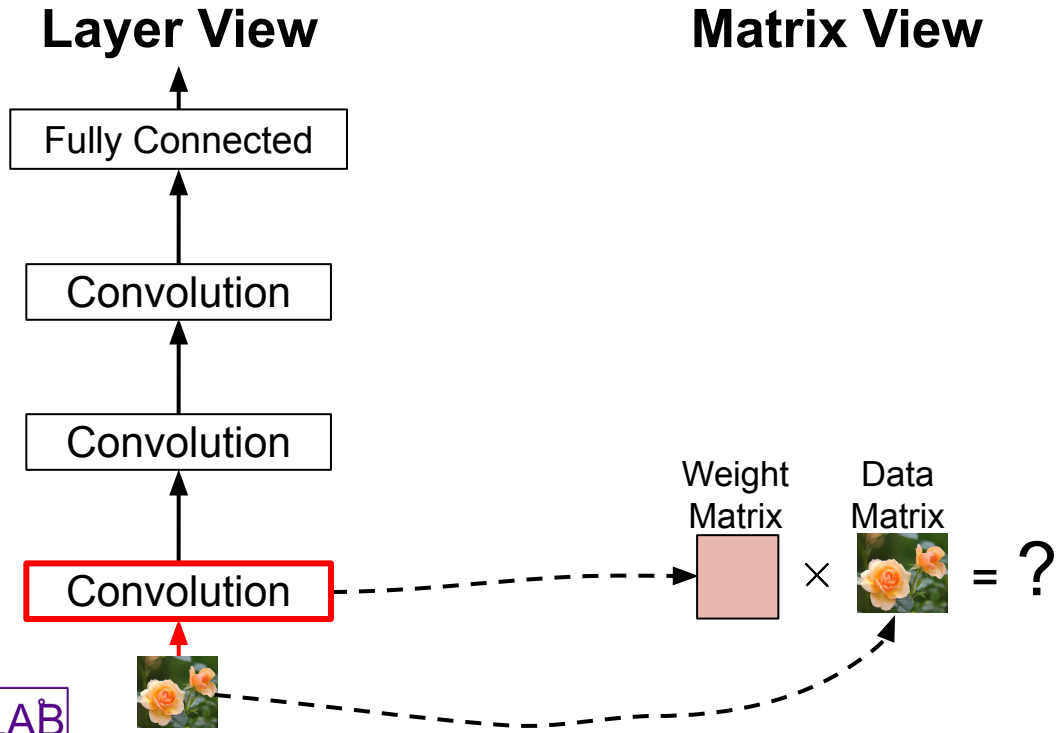
CNN with 4 layers

How Deep Neural Network is Executed?



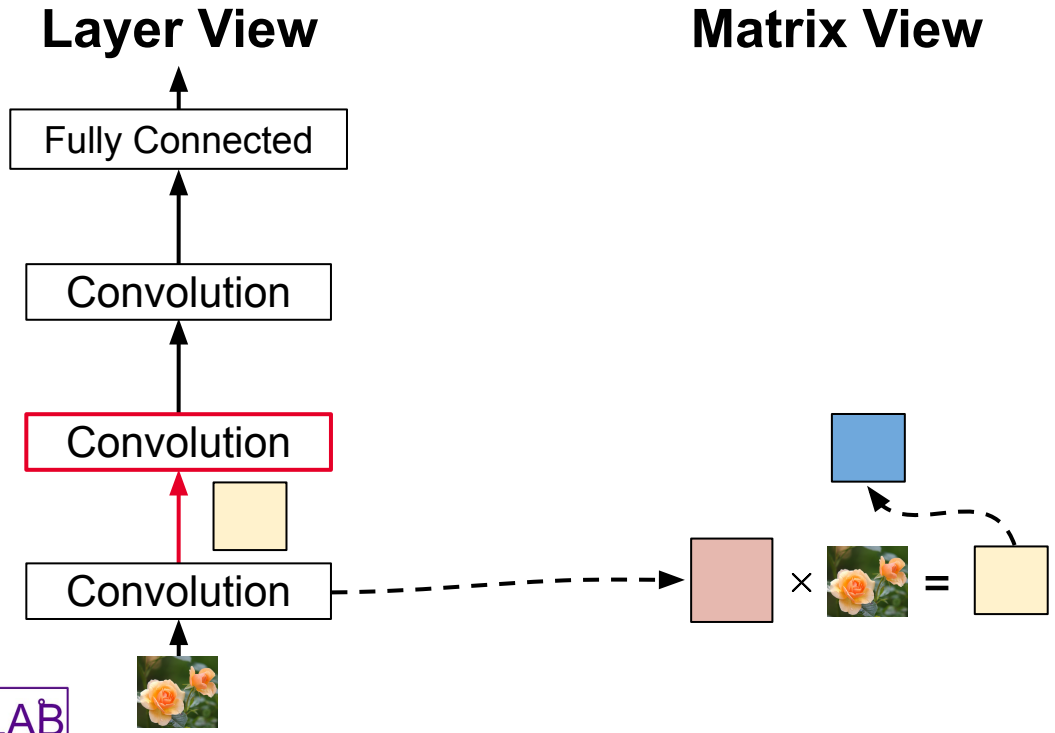
CNN with 4 layers

DNN Execution: A Matrix View



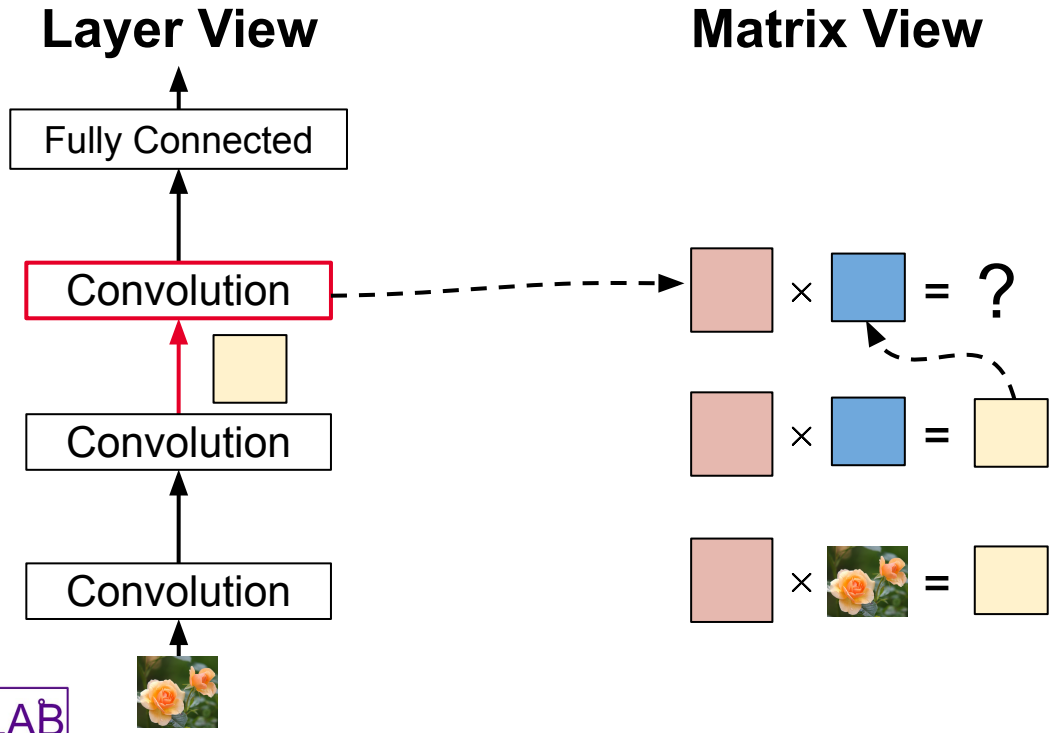
- Weight matrices are **learned** during training

DNN Execution: A Matrix View



- Weight matrices are **learned** during training

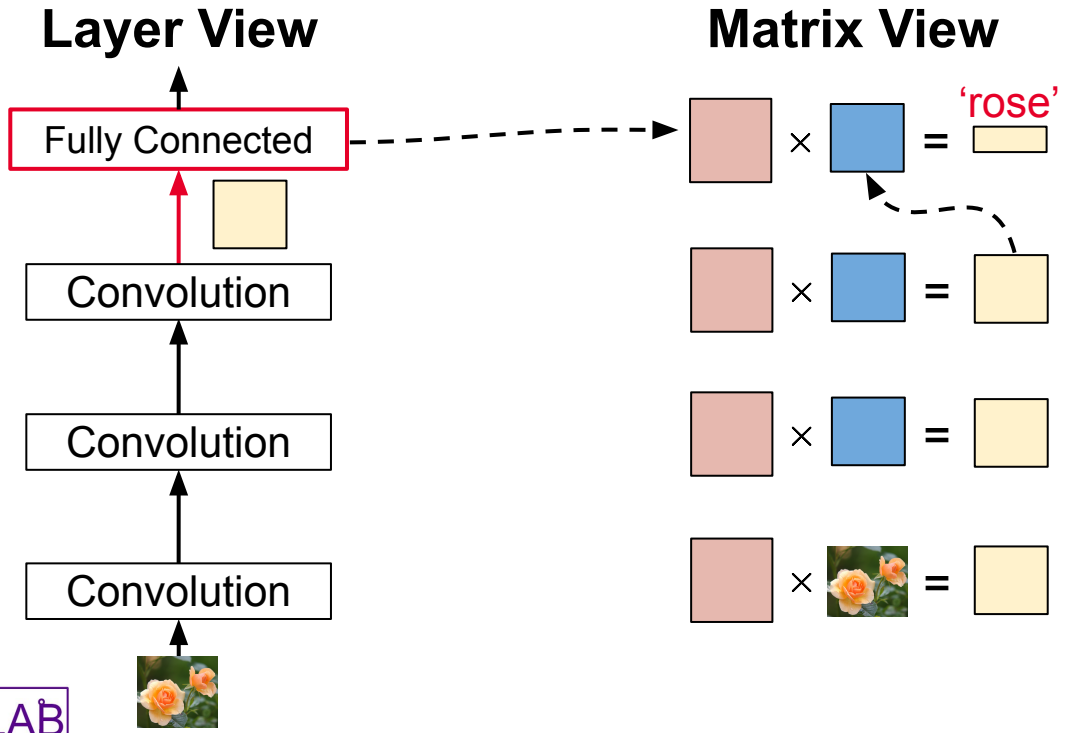
DNN Execution: A Matrix View



- Remaining layers follow this pattern.

NYU SAI LAB

DNN Execution: A Matrix View

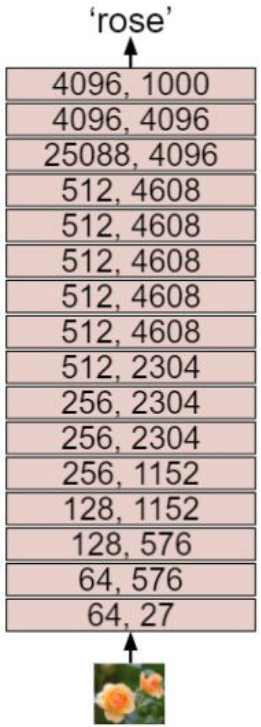


- Remaining layers follow this pattern

NYU SAI LAB

Deployment of DNN: Problems

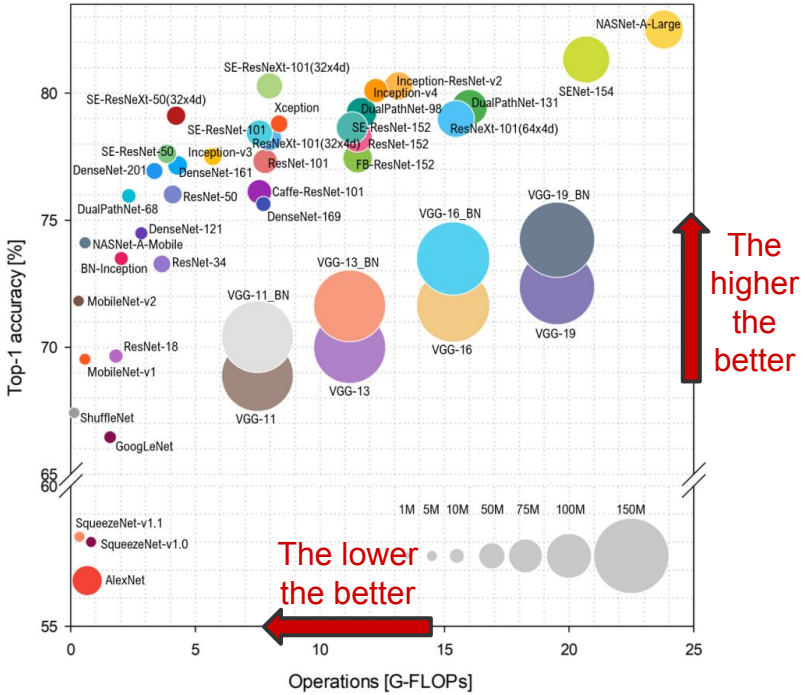
- The majority of computation workloads for DNN inference involves a series of **matrix multiplications.**



VGG-16 is a CNN with over 150M weights across 16 matrices

Deployment of DNN: Problems

- DNN suffers due to:
  - High energy consumption
  - High processing latency
  - High storage cost
- DNN needs to maintain high accuracy

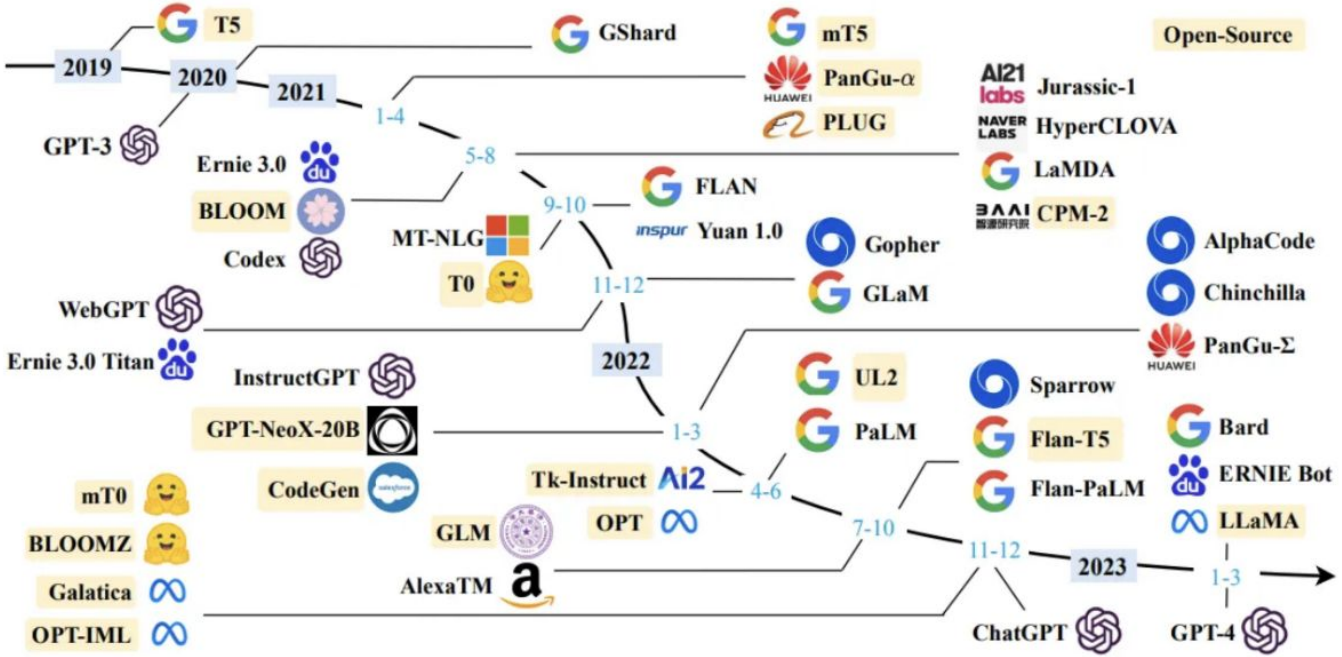


20B multiply/adds per image

NYU SAI LAB

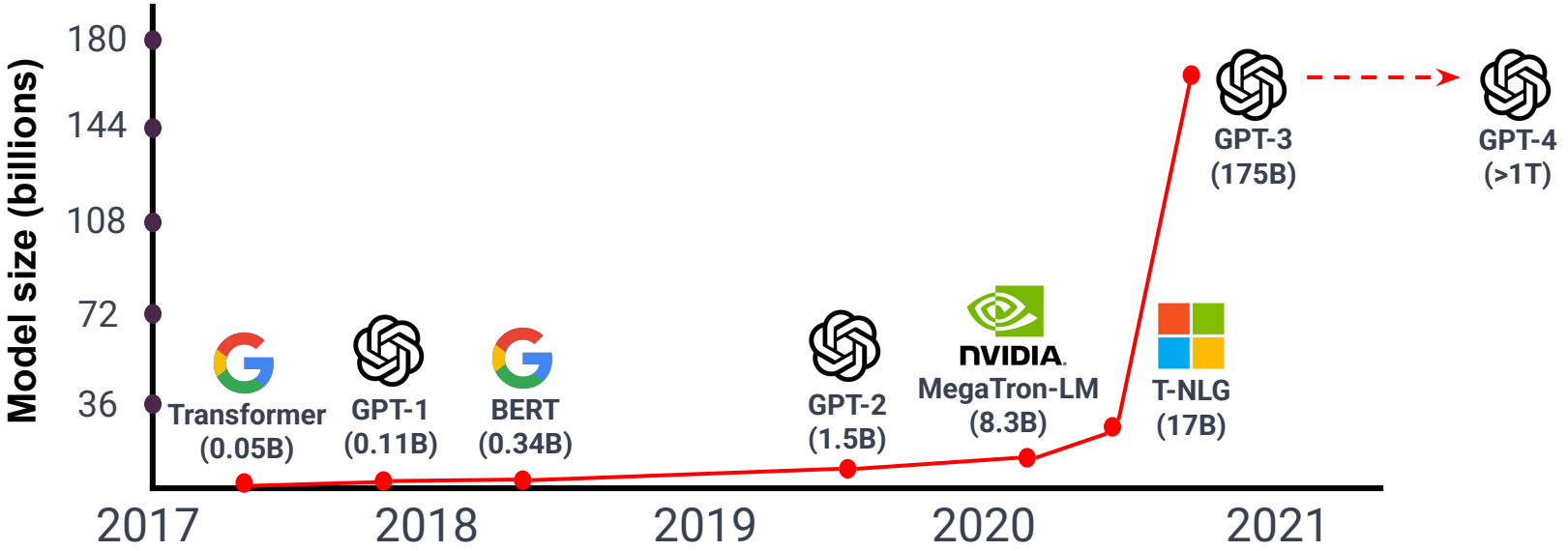
Bianco, Simone, et al. "Benchmark analysis of representative deep neural network architectures." *IEEE Access* 6 (2018): 64270-64277.

The Era of Large Models (LMs)



NYU SAI LAB

Cost of Large Models



- $1.4e^{12}$ FLOPs to execute GPT-2.

NYU SAI LAB

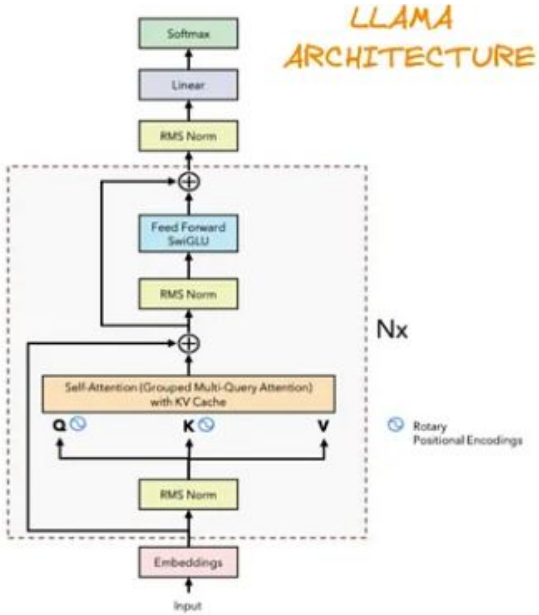
The Cost of Large Models

- Training GPT-4 required **25,000 A100 GPUs** over several weeks.
- **Cost**: Renting a single high-end GPU on cloud services like AWS can cost **$3–$5 per hour.** Training GPT-4 is estimated to cost **$63-100 million** on cloud computing resources.

Efficient AI: An Emerging Area

Model Size	FP16	FP8	INT4
8B	16 GB	8 GB	4 GB
70B	140 GB	70 GB	35 GB
405B LLaMA 3.1	810 GB	405 GB	203 GB

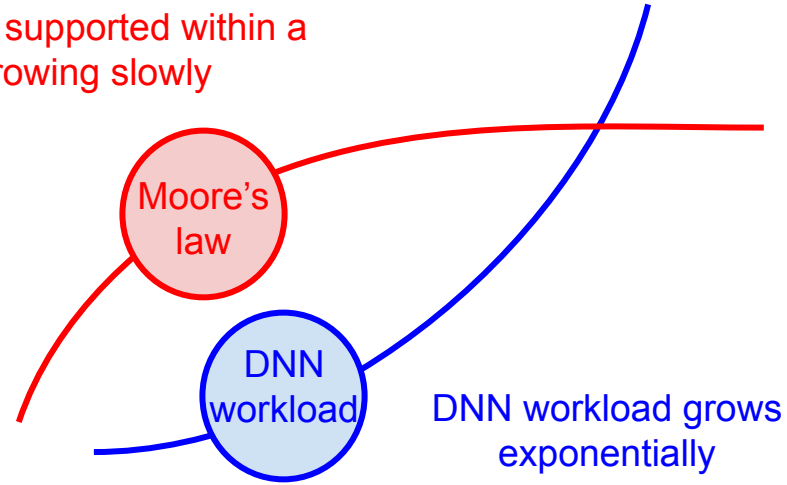
Design more aggressive and efficient AI model is of paramount importance



NYU SAI LAB

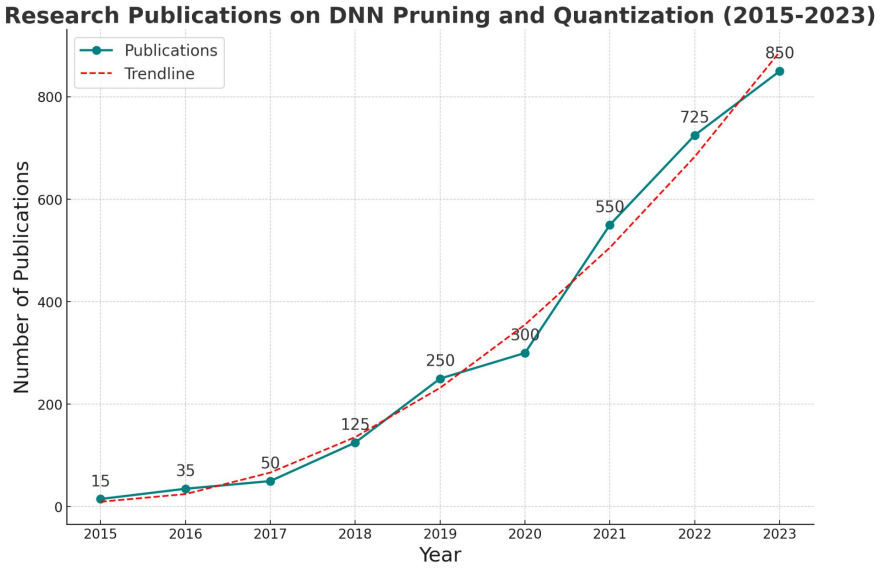
Efficient AI: An Emerging Area

The amount of compute supported within a hardware unit is growing slowly



How to reduce the compute while maintaining a good DNN accuracy?

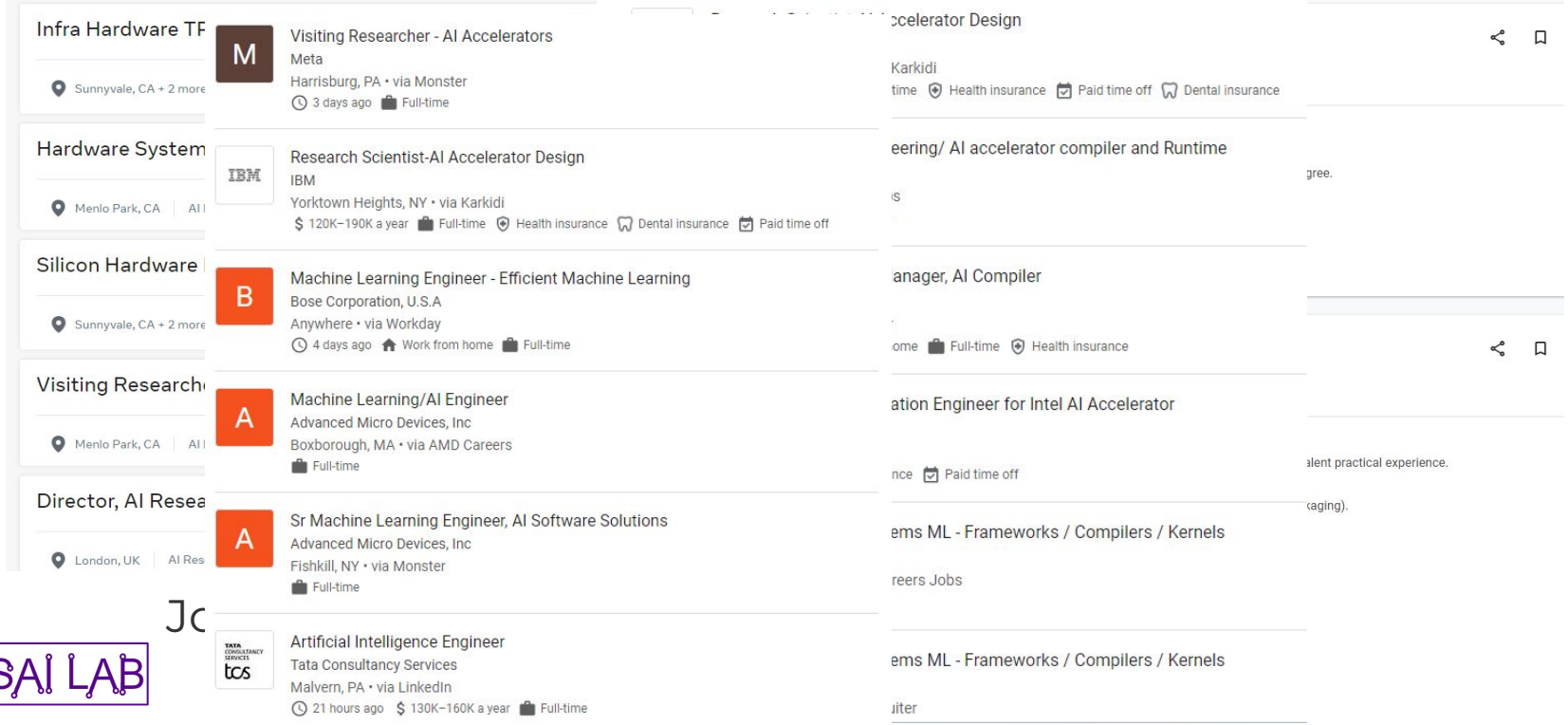
Efficient AI: An Emerging Area



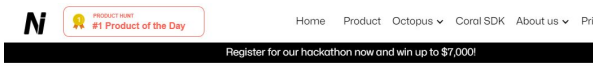
- Efficient AI has become one of the most popular areas in AI community.
- The recent emergence of large models has further heightened the need for efficient AI.

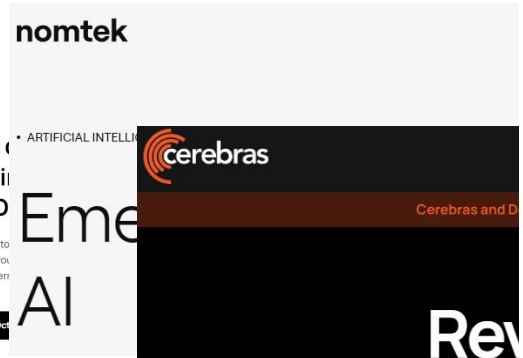
NYU SAI LAB

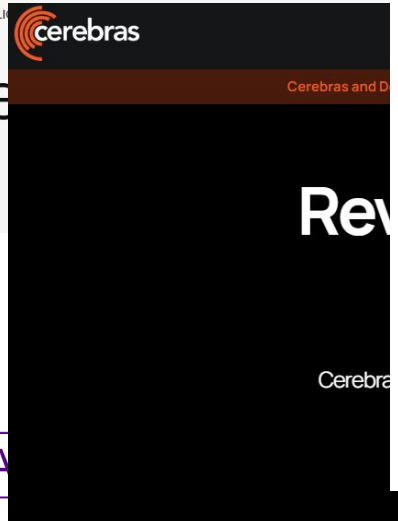
Efficient AI: An Emerging Area

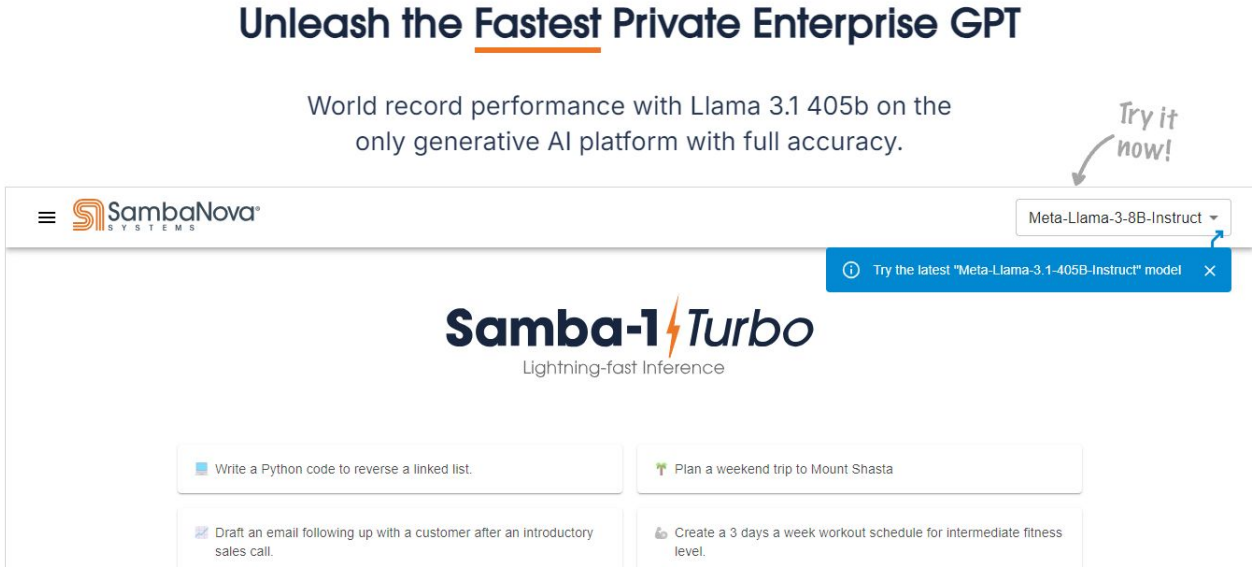


AI Tech Startups/Unicorns



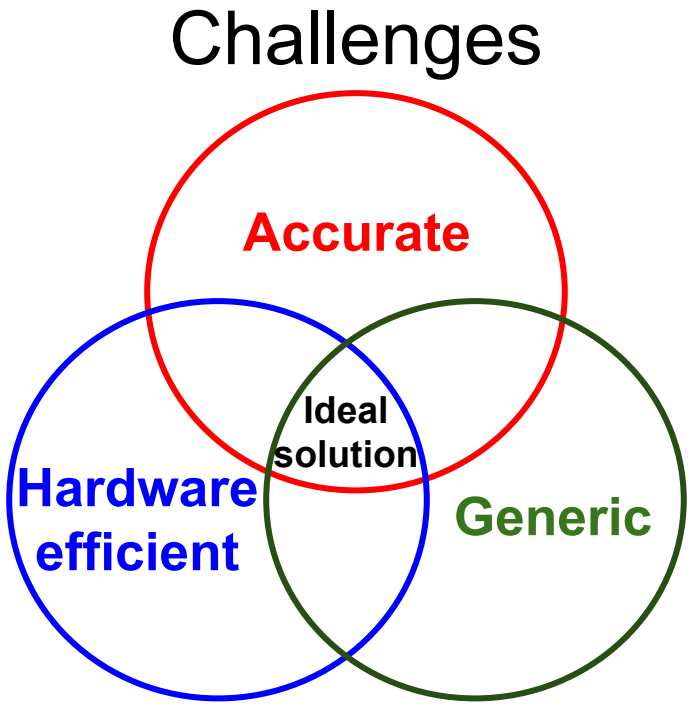






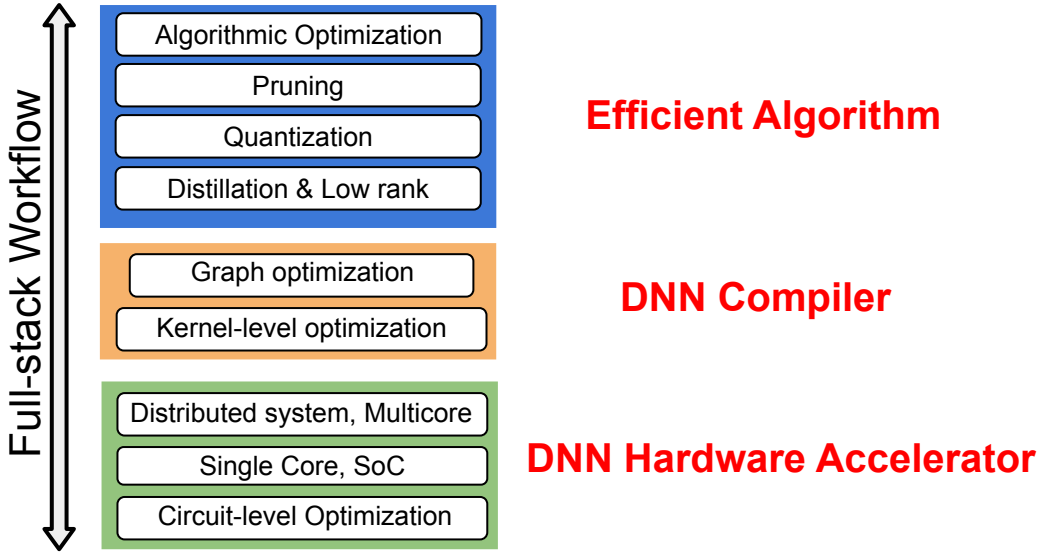
NYU SAI LA

Efficient AI: An Emerging Area

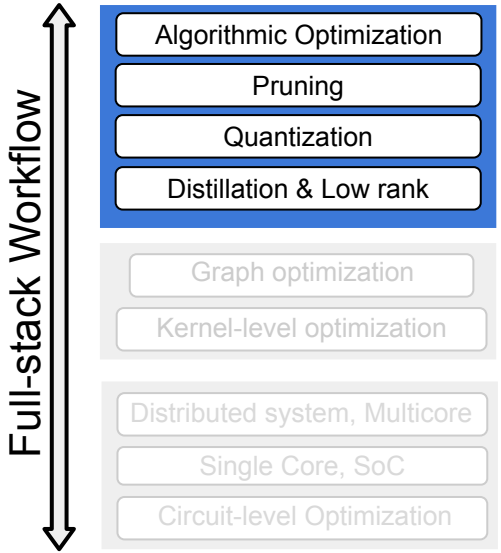


NYU SAI LAB

Efficient AI: Full-stack Workflow



Efficient AI: Full-stack Workflow



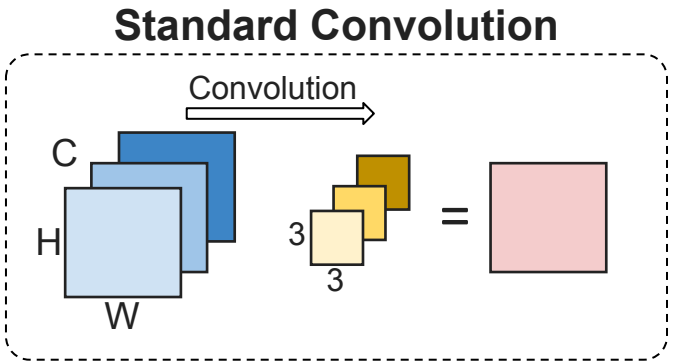
Efficient Algorithm

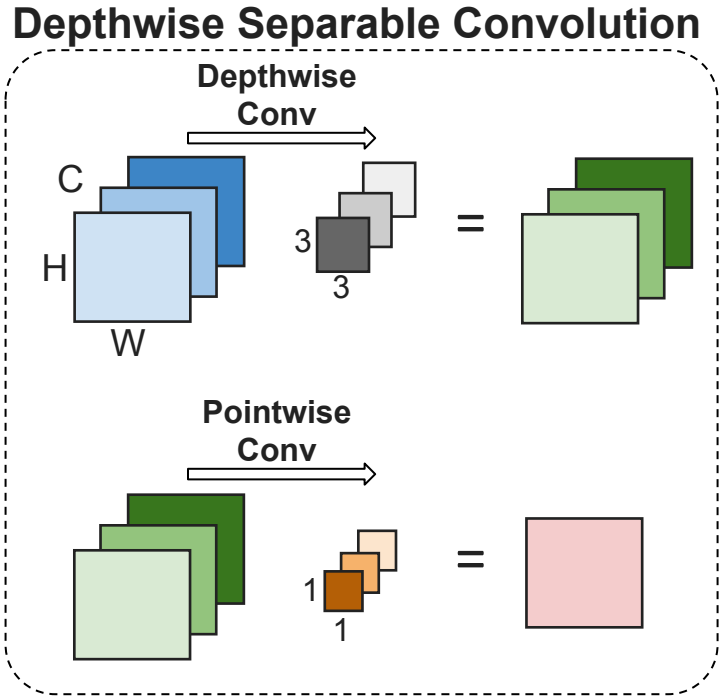
DNN Compiler

DNN Hardware Accelerator

NYU SAI LAB

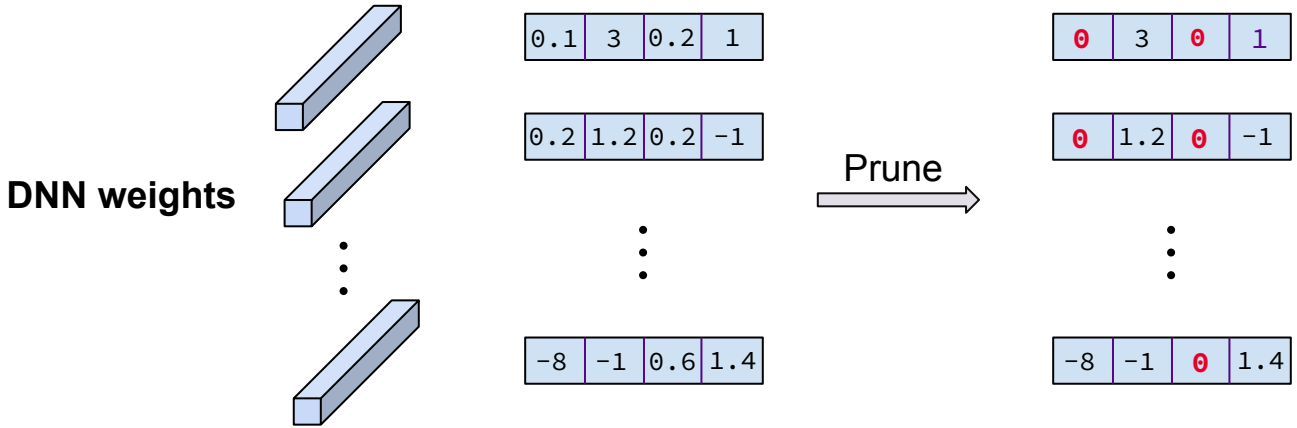
Algorithmic Optimization



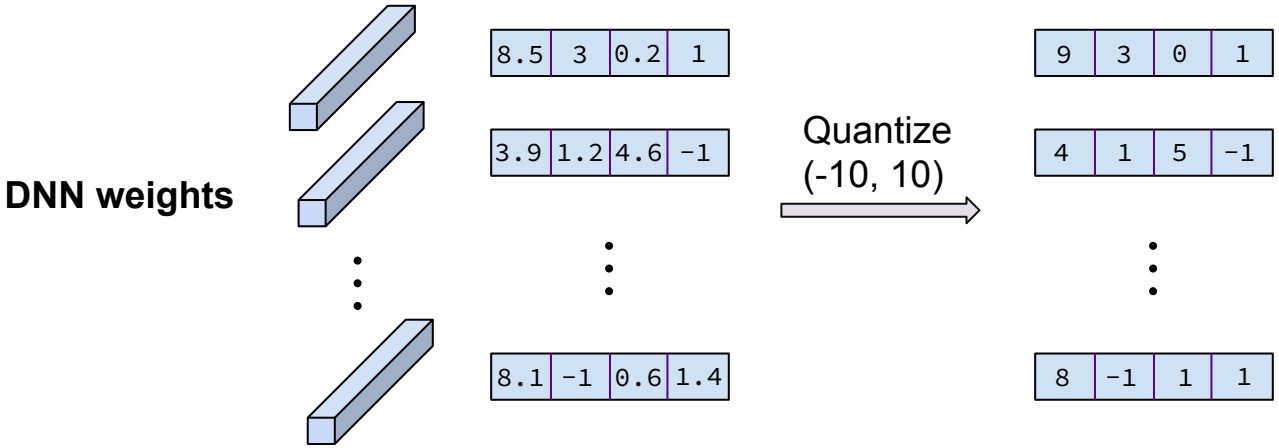


NYU SAI LAB

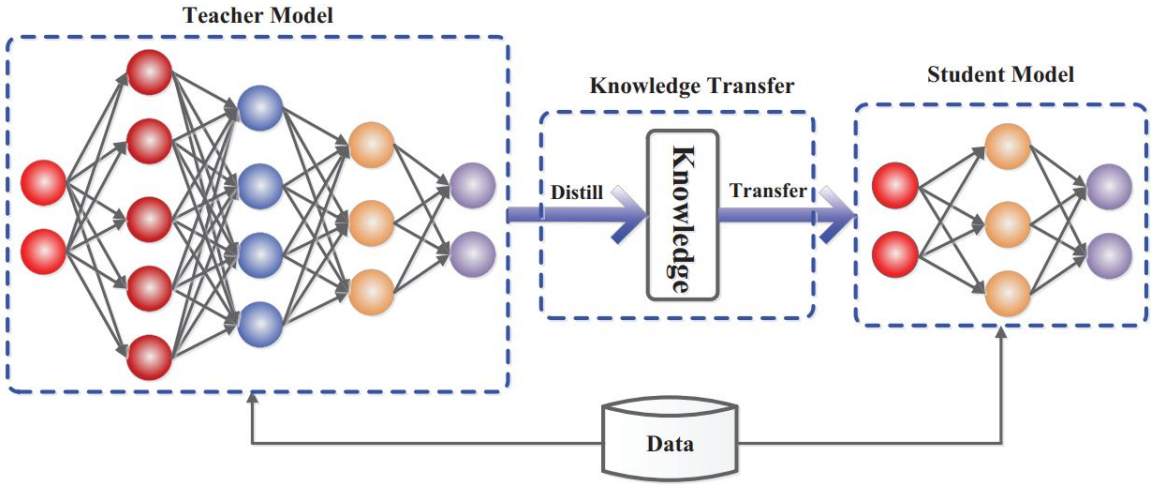
Efficient DNN Algorithm: Pruning



Efficient DNN Algorithm: Quantization

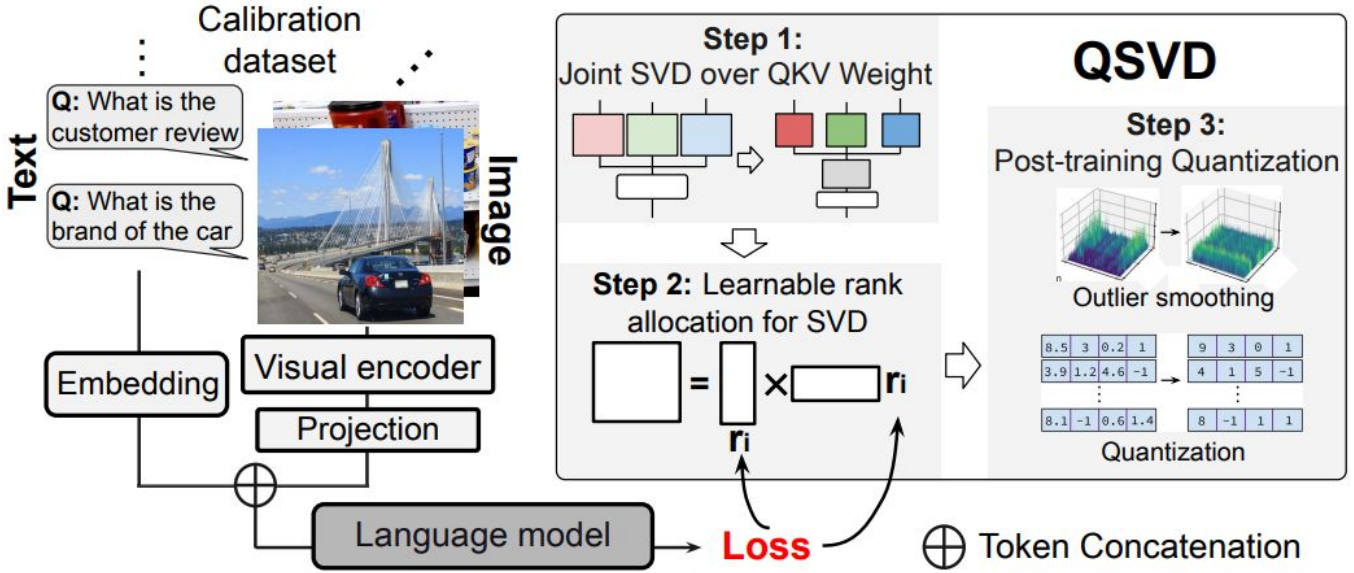


Knowledge Distillation



NYU SAI LAB

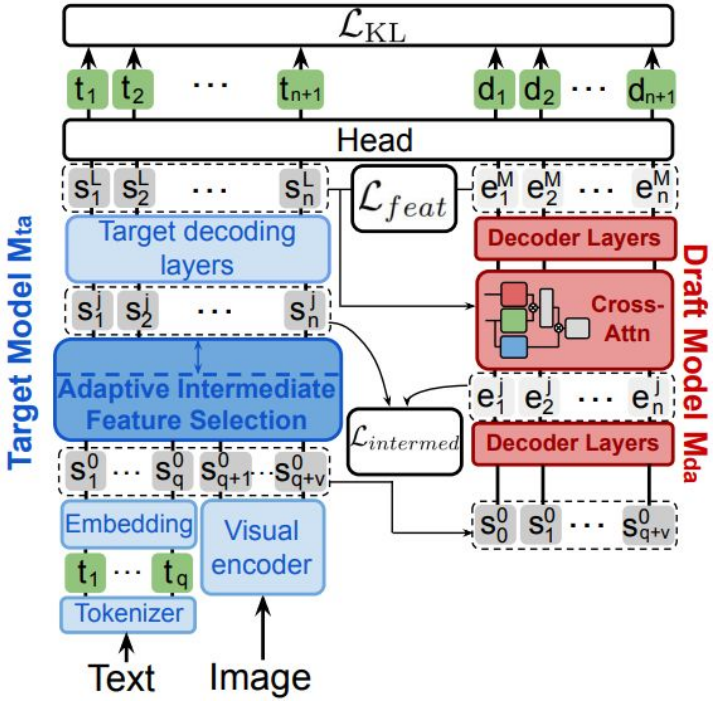
QSVD

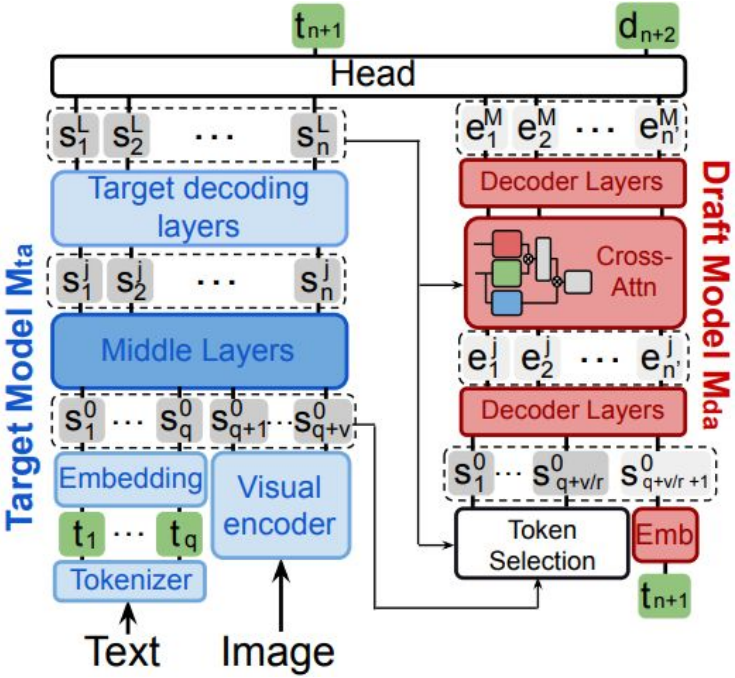


- We propose leveraging Singular-Value Decomposition over the joint query (Q), key (K), and value (V) weight matrices to reduce KV cache size and computational overhead.

NYU SAI LAB

Speculative Decoding with DREAM

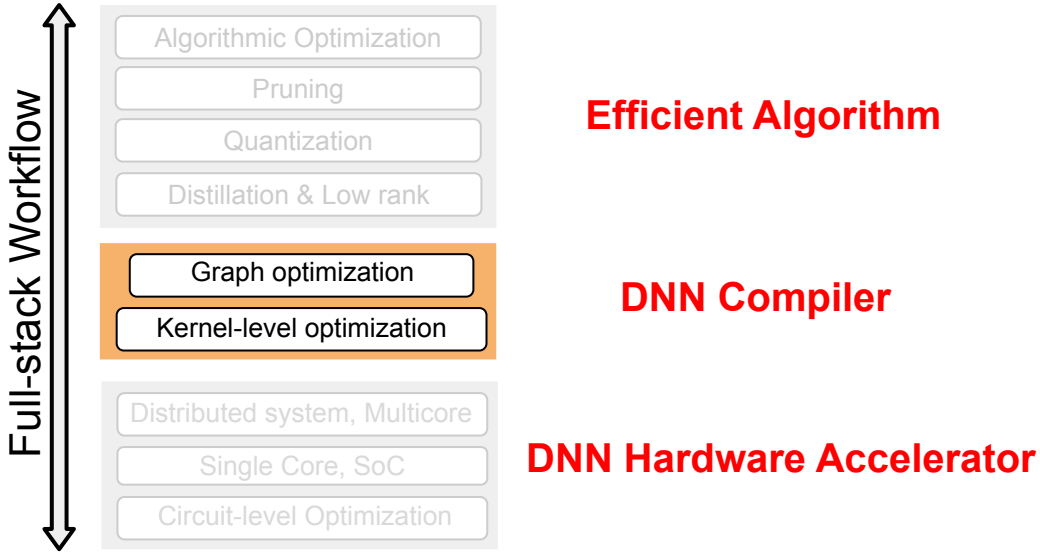




NYU SAI LAB

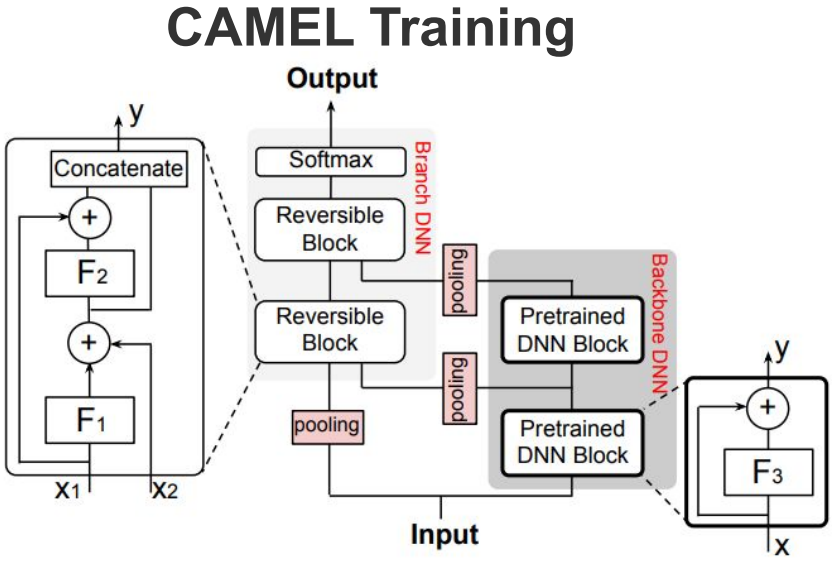
- We introduce DREAM, a novel speculative decoding framework tailored for VLMs.

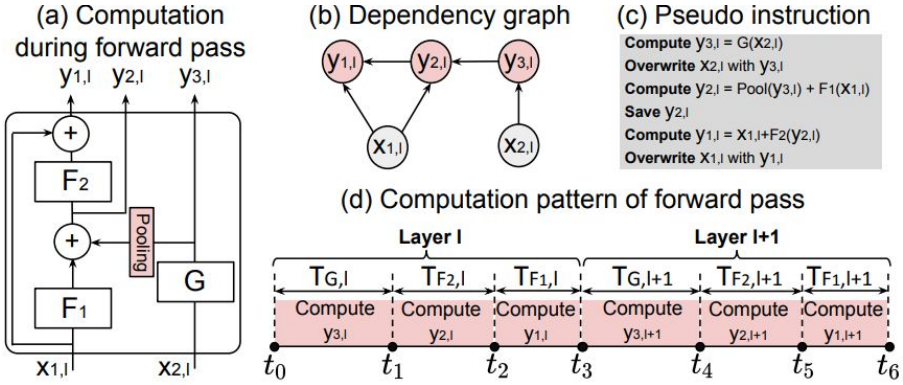
Efficient AI: Full-stack Workflow



NYU SAI LAB

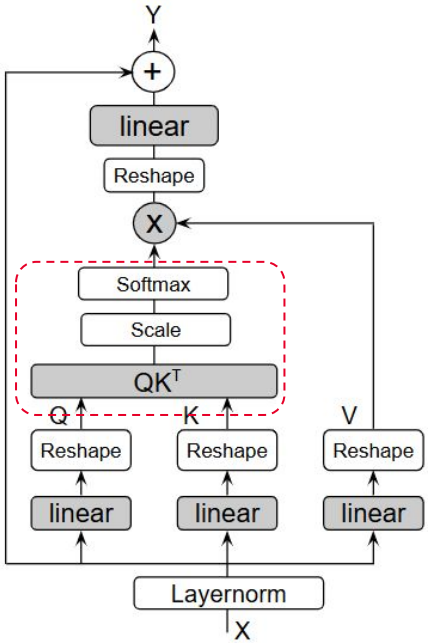
Graph Level Optimization

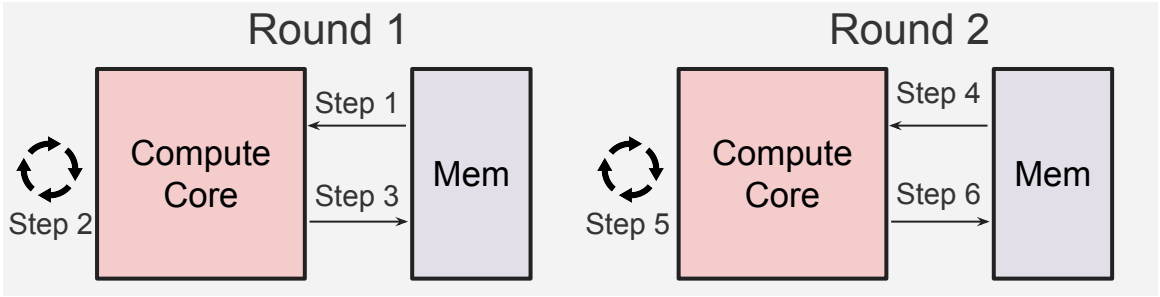


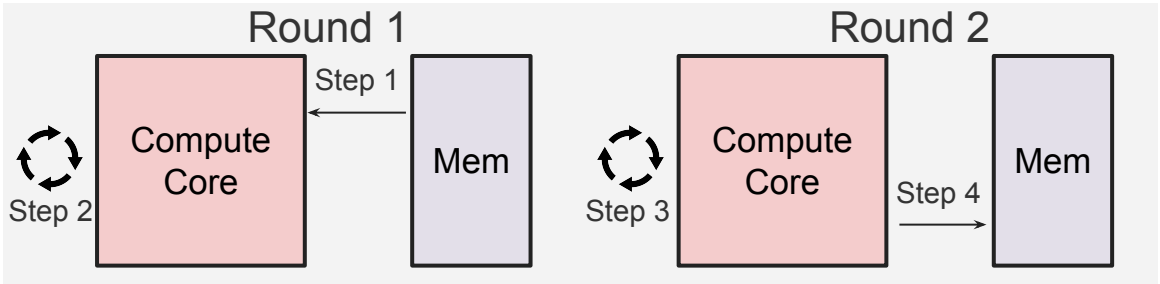


Zhang, Sai Qian, et al. "Camel: Co-designing ai models and edrams for efficient on-device learning." *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024.

System Level Optimization



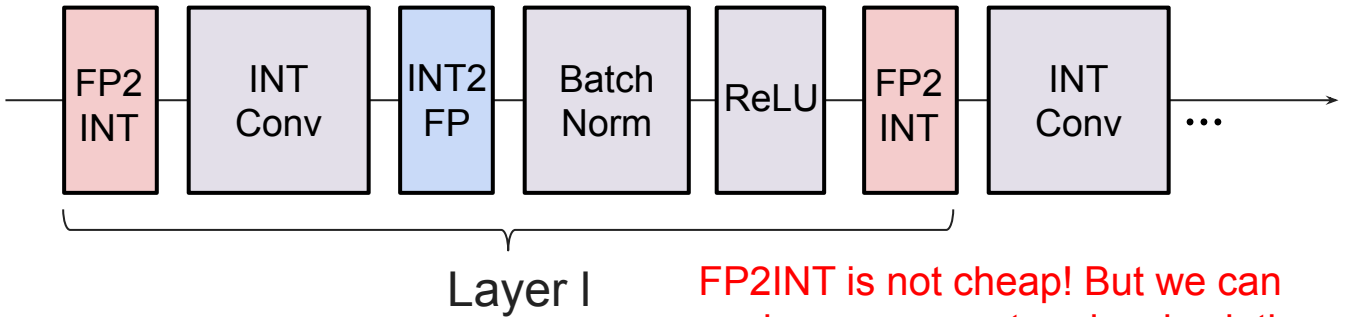




NYU SAI LAB

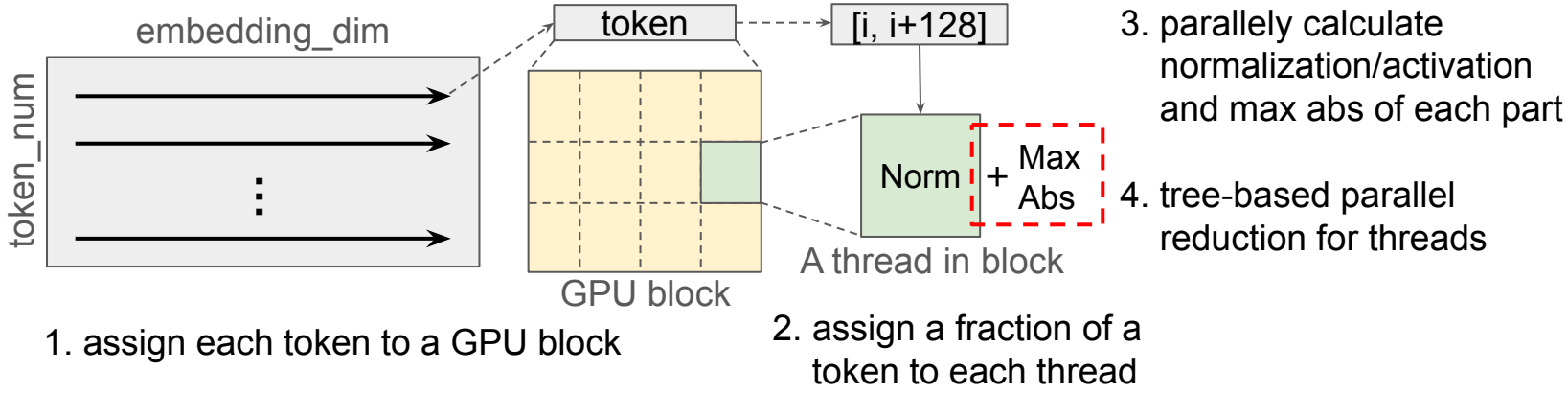
System Level Optimization

- How to convert a number x to INT representation?
  - Set the clipping range: (-L, L), bitwidth: b
  - Compute the scale: $s = 2L/(2^b - 2)$
  - Clip the input x: $x_c = Clip(x, L, -L)$
  - Calculate the INT representation: $x_{int} = round(x_c/s)$
  - Rescale: $x_q = sx_{int}$



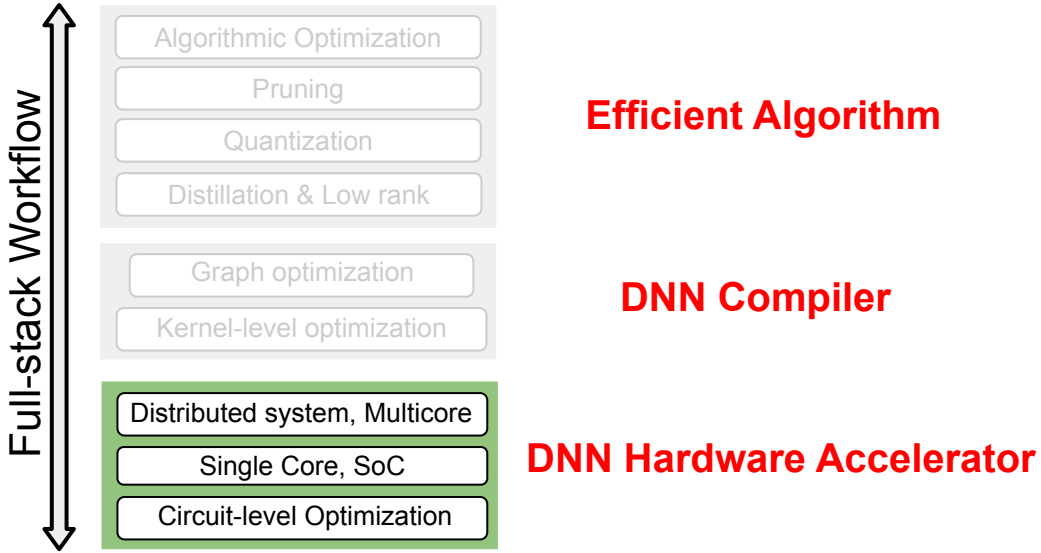
FP2INT is not cheap! But we can explore some system-level solution

Kernel Fusion



- For example, we can fuse the max searching operation to the batch normalization operation within LLM.

Efficient AI: Full-stack Workflow



Hardware Support for DNN

- GPU is better than CPU in terms of throughput for both Neural Network training and inference.
  - GPU leverages the highly parallelized architecture of its computing units to handle computational intensive operations.
- However, GPU:
  - General purpose, although much more specific than CPU.
  - Still not fast and power-efficient enough.
  - Does not support advanced efficient DNN algorithm.

NYU SAI LAB

NVIDIA

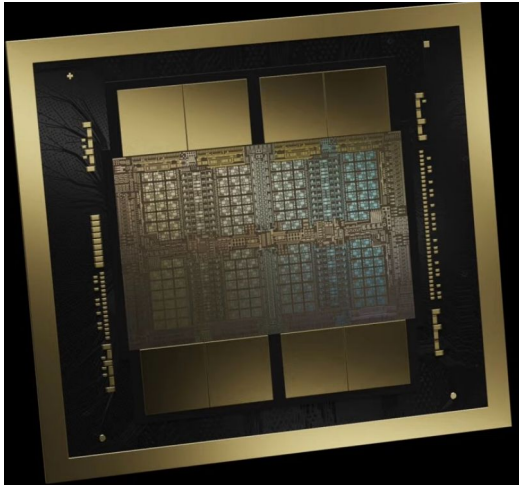
NVIDIA H100

Chip size	814 mm^2
On-chip memory	~50MB
Total memory	~96GB HBM
Cores	16,896 FP32 + 528 Tensor
Precision	FP16/FP8/INT8
Memory bandwidth	0.003 Petabytes/sec

https://www.techpowerup.com/gpu-specs/h100-sxm5-96-gb.c3974

NVIDIA

Chip size	-
On-chip memory	-
Total memory	192GB HBM
Cores	-
Precision	FP16/FP8/FP4/INT8
Memory bandwidth	8 Terabytes/sec



NVIDIA Blackwell

NYU SAI LAB

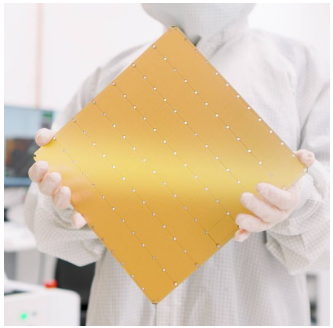
https://wccftech.com/nvidia-blackwell-gpu-architecture-official-208-billion-transistors-5x-ai-performance-192-gb-hbm3e-memory/

Hardware Support for DNN

- ASIC-based implementations have been recently explored to accelerate the DNN inference.
  - Google's TPU, Apple's Neural Engine, Cerebras AI chip, …
- FPGA-based accelerators for DNN inference have been recently developed.
  - Has good programmability and flexibility
  - Short development cycles
  - Can be used as a benchmark before implementing on ASIC

Tensor Processing Unit (Google)

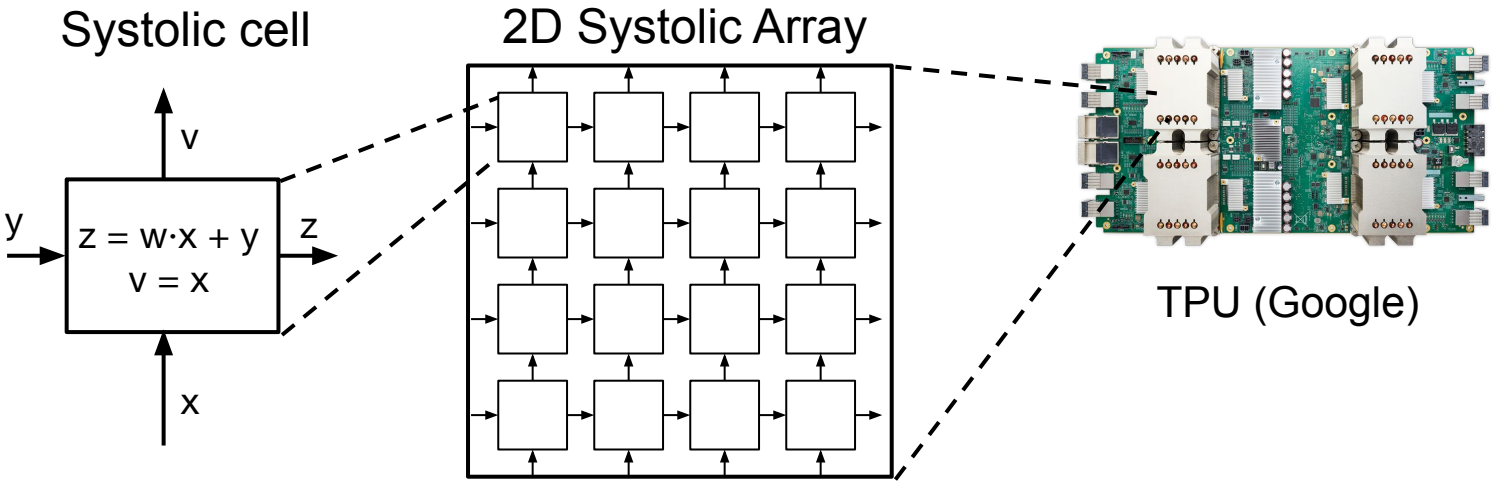
Alveo Accelerator Card (Xilinx)



Cerebras CS-3

Systolic Array

- Kung and Leiserson, "Systolic Arrays for VLSI," 1978 and Kung, "Why systolic architectures?' 1982
- 2D grid of multiplier-accumulators (MACs) for matrix multiplication
- Used by Google TPU for deep learning (2017), etc



Bit-serial Low-precision Multiplier

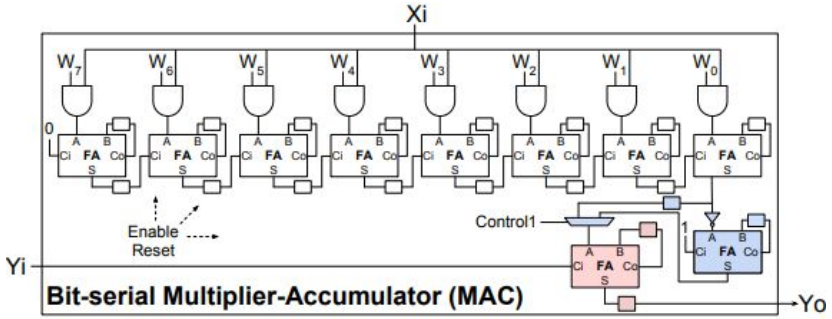
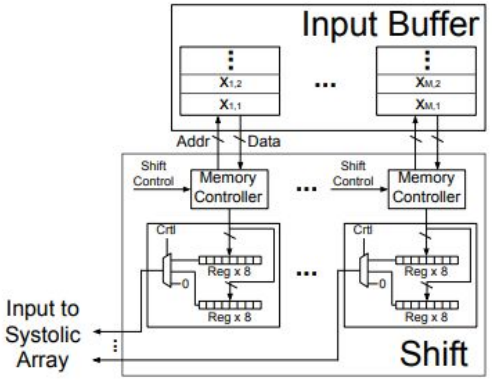
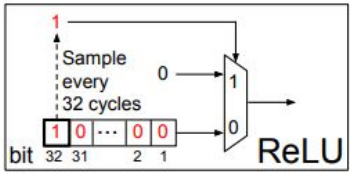


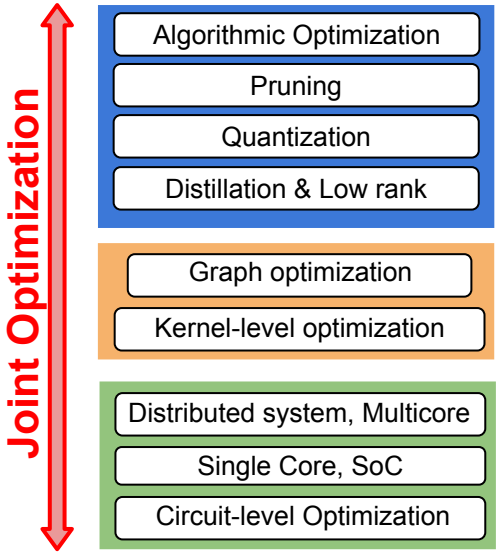
Figure 7: Bit-serial multiplier-accumulator (MAC).



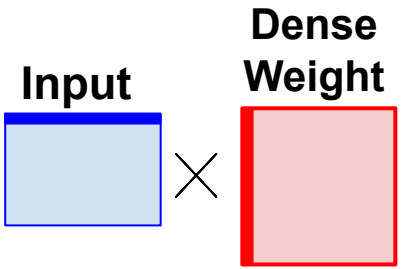


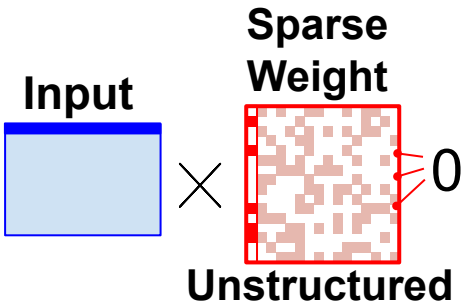
NYU SAI LAB

Why We Need Codesign?

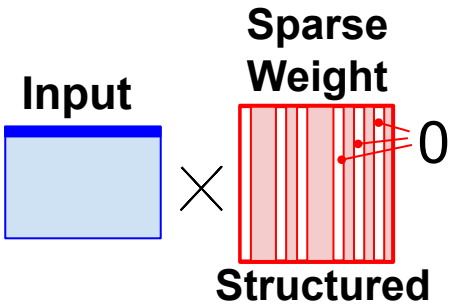


Why We Need Codesign?





High accuracy
low hardware efficiency

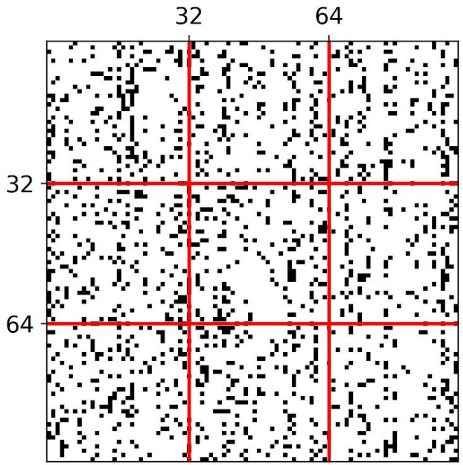


Low accuracy
high hardware efficiency

Hardware architecture needs to be considered when designing efficient DNN.

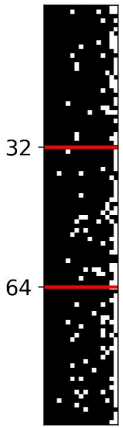
Column Combining

Sparse
Weight Matrix



Column Combining
8x reduction in size

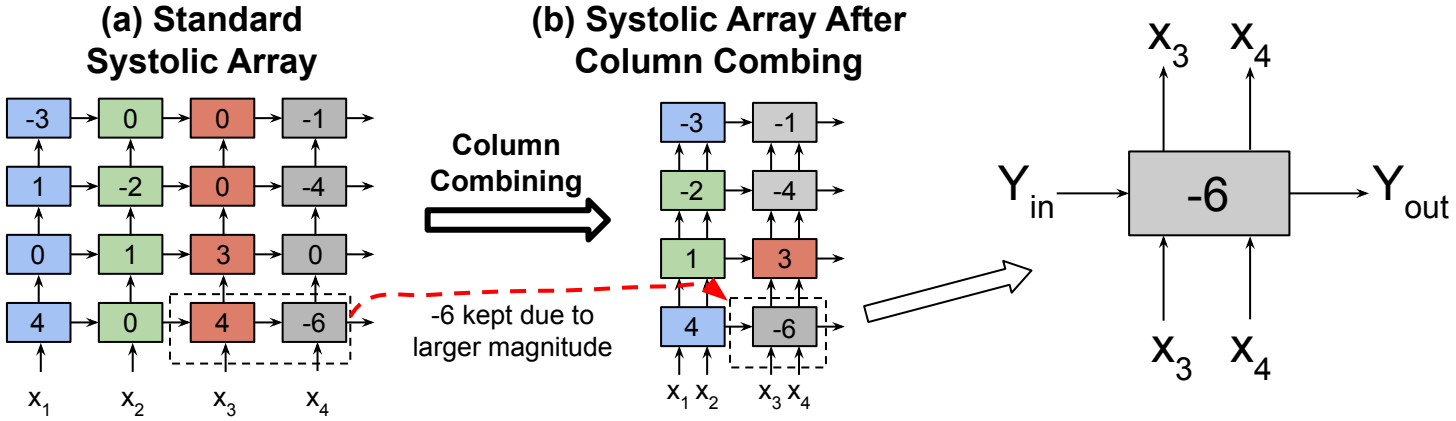
Packed Format in
Systolic Array



NYU SAI LAB

Kung, H. T., Bradley McDanel, and Sai Qian Zhang. "Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization." *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2019.

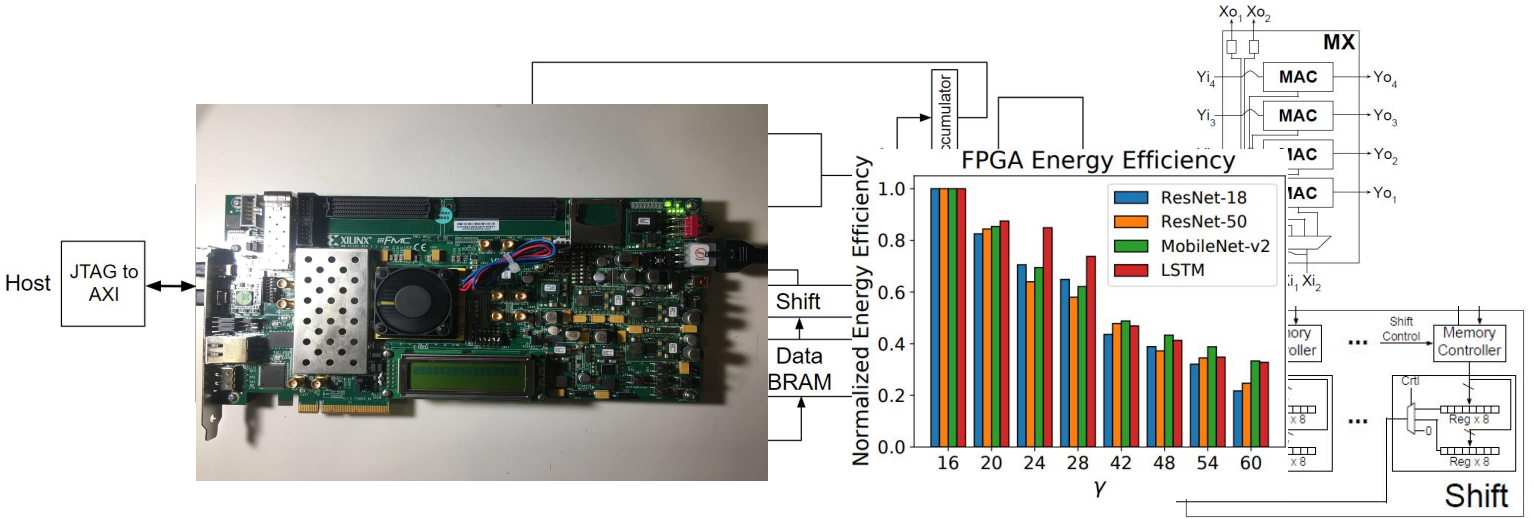
Column Combining



- Column combining can greatly increase the utilization efficiency of the systolic array
- Recently, Nvidia A100 GPU adopts a similar idea to support the balanced structured sparsity on their GPU

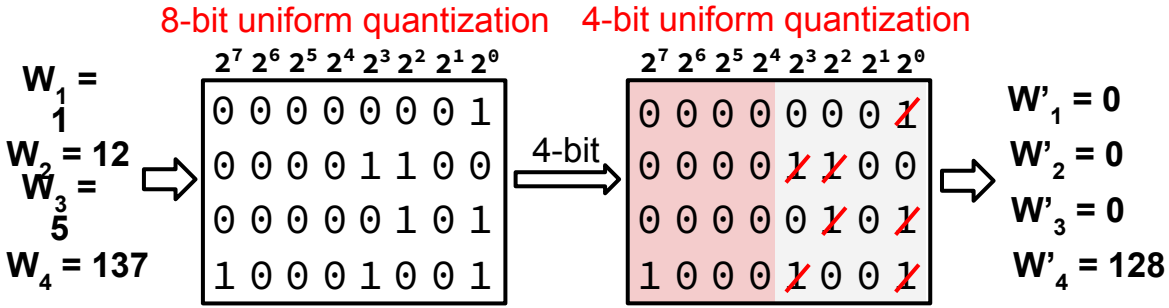
Kung, H. T., Bradley McDanel, and Sai Qian Zhang. "Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization." *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2019.

FPGA Accelerator



Kung, H. T., Bradley McDanel, and Sai Qian Zhang. "Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization." *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2019.

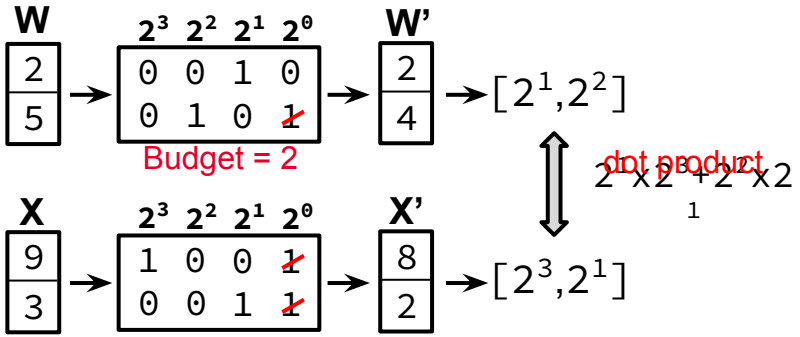
Term Quantization



- Low-precision quantization leads to significant quantization error.
- Both weights and input activation are highly biased in values.

Kung, H. T., Bradley McDanel, and Sai Qian Zhang. "Term revealing: Furthering quantization at run time on quantized dnns." *arXiv preprint arXiv:2007.06389* (2020).

Term Quantization



4-bit uniform quantization

2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0

0	0	0	0	0	0	0	1
0	0	0	0	1	1	0	0
0	0	0	0	0	1	0	1
1	0	0	0	1	0	0	1

$W'_1 = 0$

$W'_2 = 0$

$W'_3 = 0$

$W'_4 = 128$

TQ with a budget = 4

2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0

0	0	0	0	0	0	0	1
0	0	0	0	1	1	0	0
0	0	0	0	0	1	0	1
1	0	0	0	1	0	0	1

$W'_1 = 0$

$W'_2 = 12$

$W'_3 = 0$

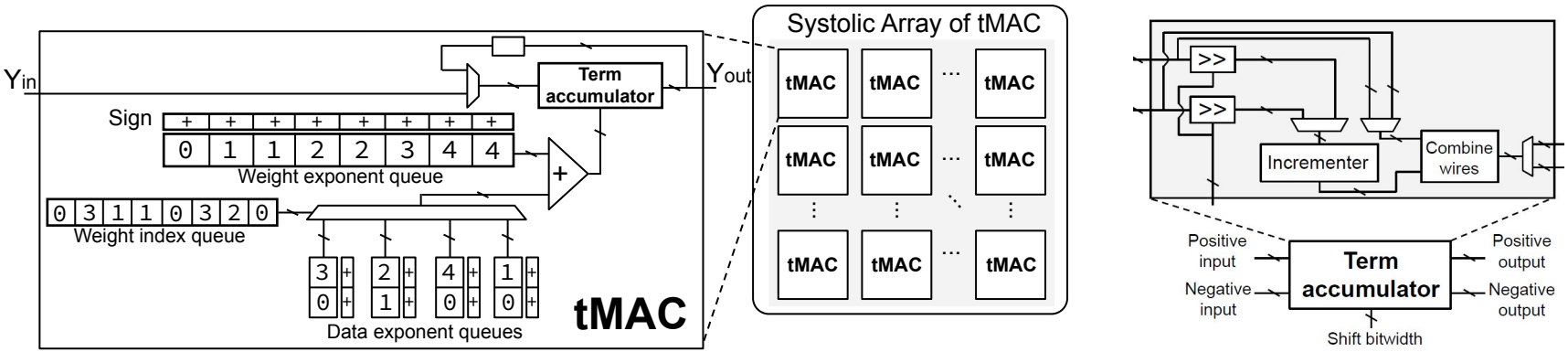
$W'_4 = 136$

- We can control the term-level computations by setting a **group term budget.**
- For a group of values, we rank and remove the small terms based on this budget.

NYU SAI LAB

Kung, H. T., Bradley McDanel, and Sai Qian Zhang. "Term revealing: Furthering quantization at run time on quantized dnns." *arXiv preprint arXiv:2007.06389* (2020).

Term Quantization: Accelerator Design

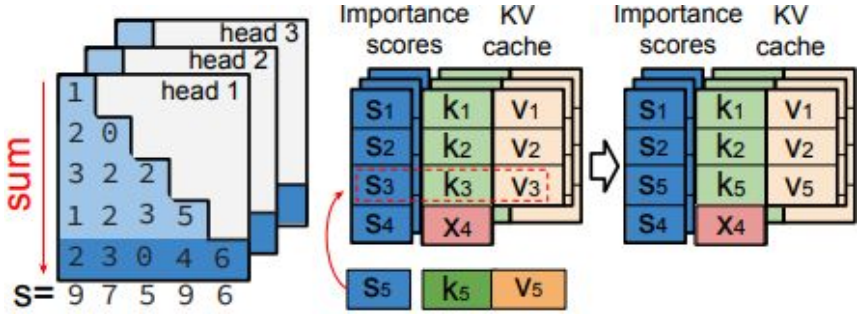


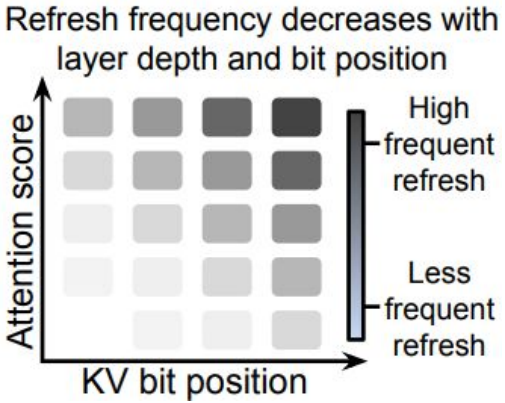
- We propose the term MAC (tMAC) for the efficient implementation of TQ.
- A tMAC processes all term-pair multiplications across a group of weight and data values.
- Each term is represented by their corresponding exponent (2-3 bits).
- The term accumulation can be implemented using half adders.

NYU SAI LAB

Kung, H. T., Bradley McDanel, and Sai Qian Zhang. "Term revealing: Furthering quantization at run time on quantized dnns." *arXiv preprint arXiv:2007.06389* (2020).

Kelle: Co-design KV Caching and eDRAM for Efficient LLM Serving in Edge Computing

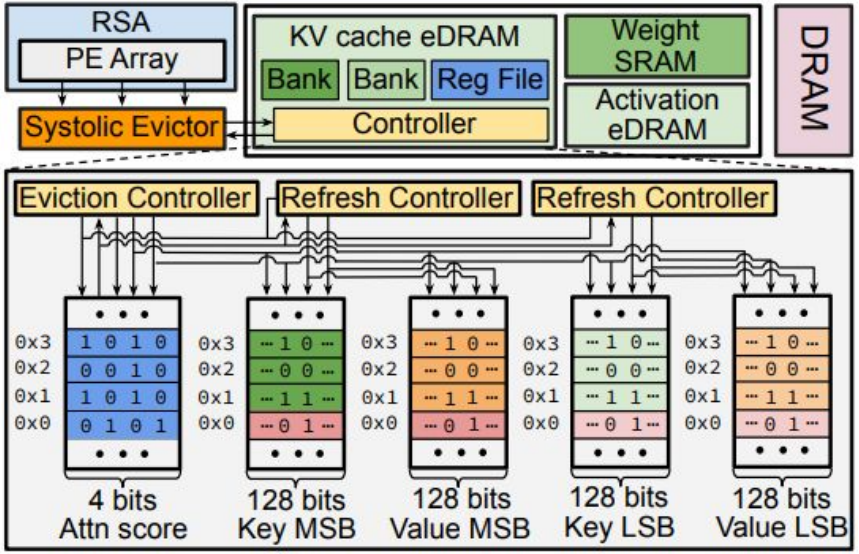




- We propose using embedded DRAM (eDRAM) as the primary storage for LLM serving in edge device, which offers higher storage density compared to SRAM.
- To reduce eDRAM costs and improve overall system performance, we propose Kelle, a software-hardware co-design solution optimized for deploying LLMs on eDRAMbased edge systems.

NYU SAI LAB

Kelle: Co-design KV Caching and eDRAM for Efficient LLM Serving in Edge Computing



- Combined with our fine-grained memory eviction, recomputation, and refresh control algorithms, the Kelle accelerator delivers a 3.9× speedup and 4.5× energy savings compared to existing baseline solutions.

NYU SAI LAB

Lecture Plan (Tentative)

Chapter 1: Basics and Efficient DNN Architectures

- Lecture 1: Review the basics of DNN
- Lecture 2: CNNs, RNNs and Variants
- Lecture 3: Transformer and its Application in AIGC

Lecture Plan (Tentative)

Chapter 2: Efficient DNN Algorithms

- Lecture 4: DNN Pruning
- Lecture 5: DNN Quantization
- Lecture 6: Distillation, Low rank Decomposition and NAS
- Lecture 7: Algorithm for Large Model Efficiency
- Lecture 8: Efficient DNN Training, Distributed Training, Federated Learning

Lecture Plan (Tentative)

Chapter 3: System and Hardware Design for AI

- Lecture 9: Distributed Machine Learning System for Training and Inference
- Lecture 10: CNN Dataflow & Hardware Accelerators
- Lecture 11: Transformer & LLM Accelerators
- Lecture 12: Hardware Accelerator for DNN Training
- Lecture 13: New Computation Paradigms / ARVR Computing

Lecture 1: Neural Network Basics

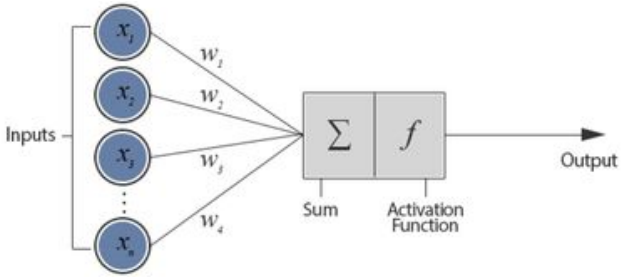
ECE-GY 9483 / CSCI-GA 3033
Special Topics: Efficient AI and Hardware Accelerator Design

Basics of Deep Neural Networks

- Multi-layer Perceptrons (MLPs)
  - Fully-connected layers
  - Activation functions
  - Loss function
  - Backpropagation
- How forward and backward propagation is performed?
- How to compute the gradient?
- How to update the weight?
- How to initialize the weight before training?

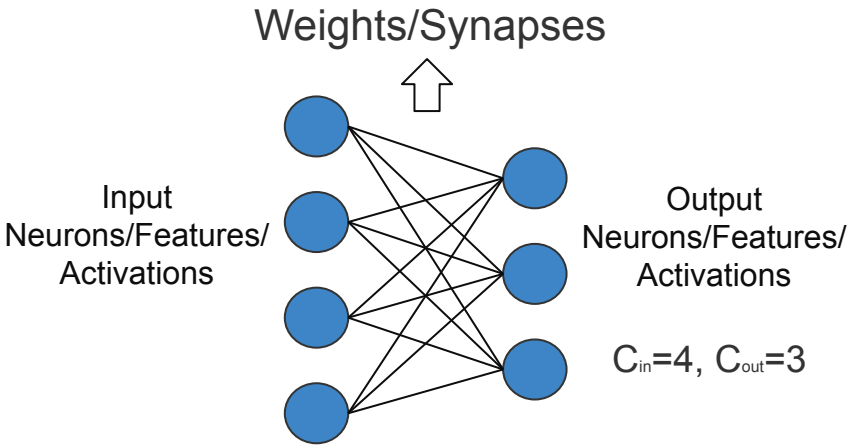
Multi-layer Perceptrons

- Usually consists of fully-connected layers with nonlinear activation functions.



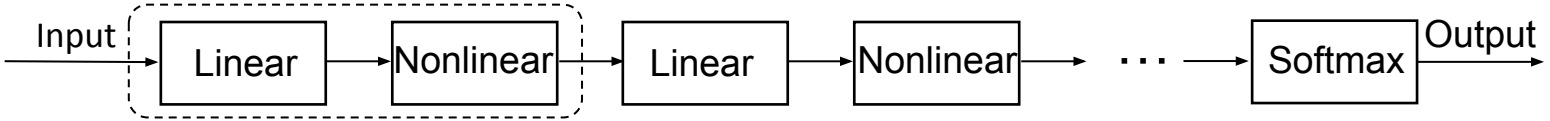
- A neural network consists of interconnected nodes, called neurons, organized into layers.
- Each neuron receives input signals (activations), performs a computation on them, and produces an output signal that may be passed to other neurons in the network.

Fully-connected layers (Linear layers)



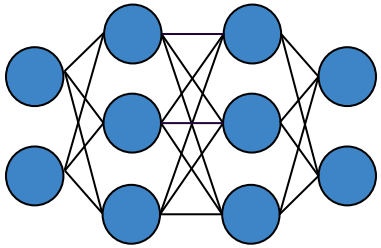
$$Y = XW + b$$

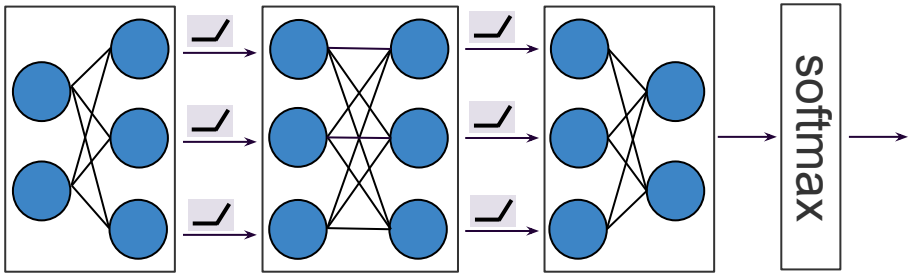
- X (input activations): $B \times C_{in}$
- Y (output activations): $B \times C_{out}$
- W (weights): $C_{in} \times C_{out}$
- b (bias): $1 \times C_{out}$
- C_{in}: Number of input activations
- C_{out}: Number of output activations
- B: batch size

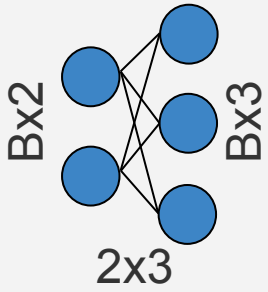


NYU SAI LAB

Computational Cost for MLP

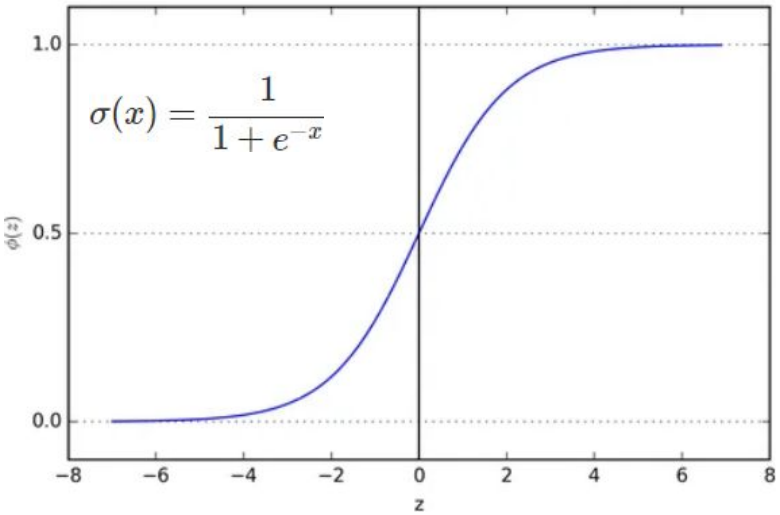






- Number of multiply-accumulate operations (MACs):
  - Bx2x3 = 6B
- Storage cost:
  - 6 x 32 = 192 bits (Weights)
  - (2B + 3B) x 32 bits (Activation)

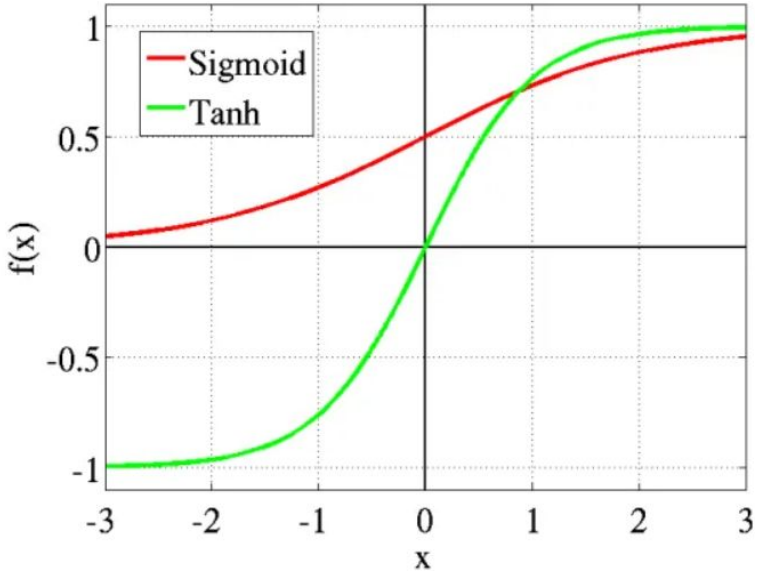
Sigmoid



- Function: $\sigma(x) = \frac{1}{1+e^{-x}}$
- Domain: $(-\infty, \infty)$
- Range: [0,1]
- Differentiable everywhere
- Derivative: δ(x)(1-δ(x))

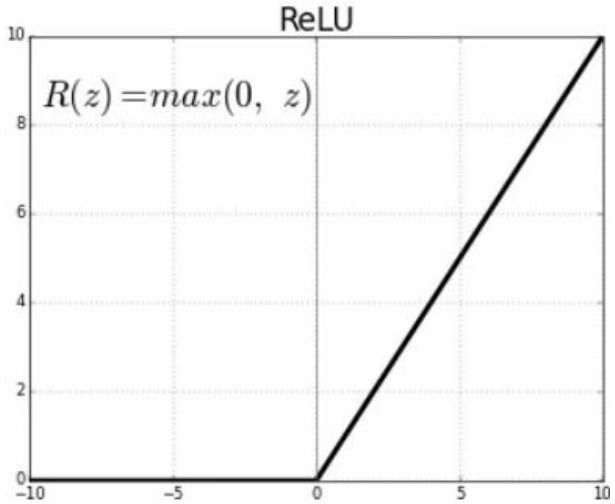
NYU SAI LAB

Tanh



- Function: $tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$
- Domain: $(-\infty, \infty)$
- Range: [-1,1]
- Differentiable everywhere
- Derivative: $1 - tanh^2(x)$

ReLU

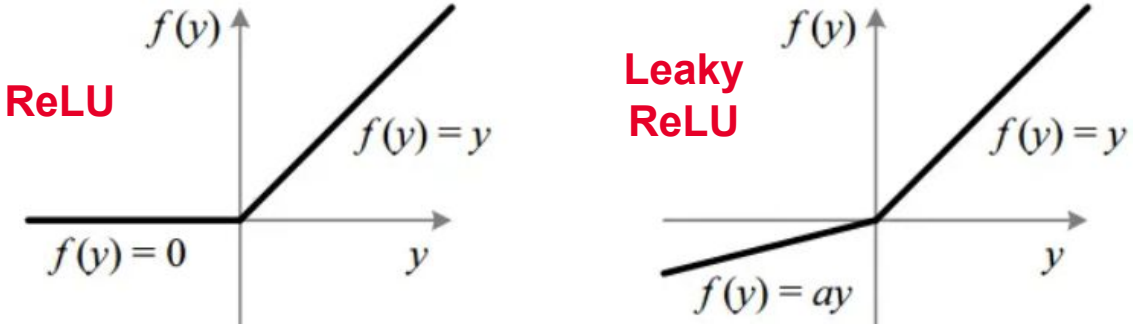


$$ReLU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

- Domain: $(-\infty, \infty)$
- Range: $[0, \infty]$
- Differentiable everywhere $\begin{cases} 1, x > 0 \\ 0, x < 0 \end{cases}$

NYU SAI LAB

Leaky ReLU



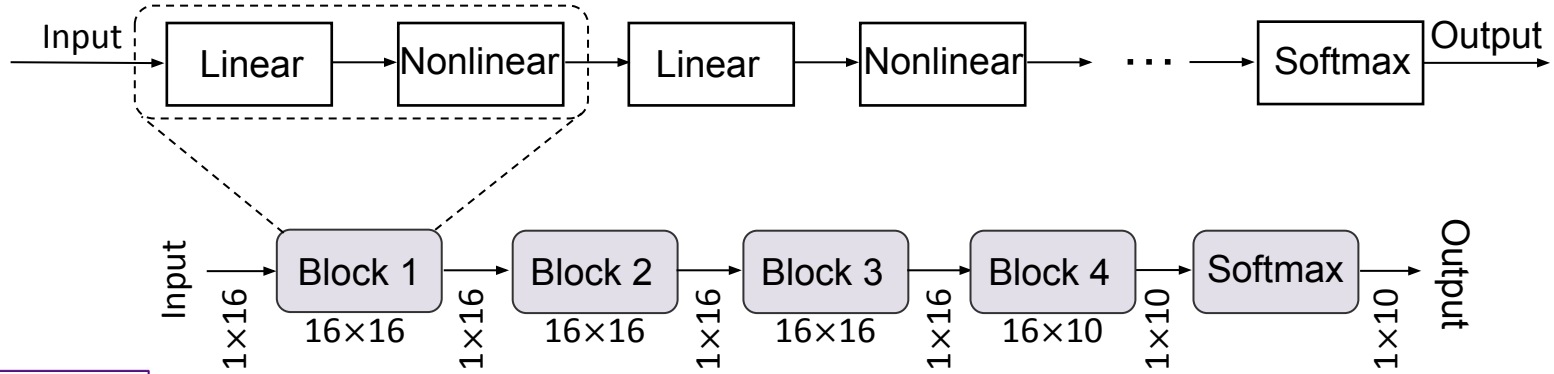
$$Leaky\_ReLU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ ax, & \text{otherwise} \end{cases}$$

- Domain: $(-\infty, \infty)$
- Range: $(-\infty, \infty)$

Softmax

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

- Domain: $[-\infty\ \infty]^N$
- Range: $[0,1]^N$
- It is a multivariate function



NYU SAI LAB

Loss Functions

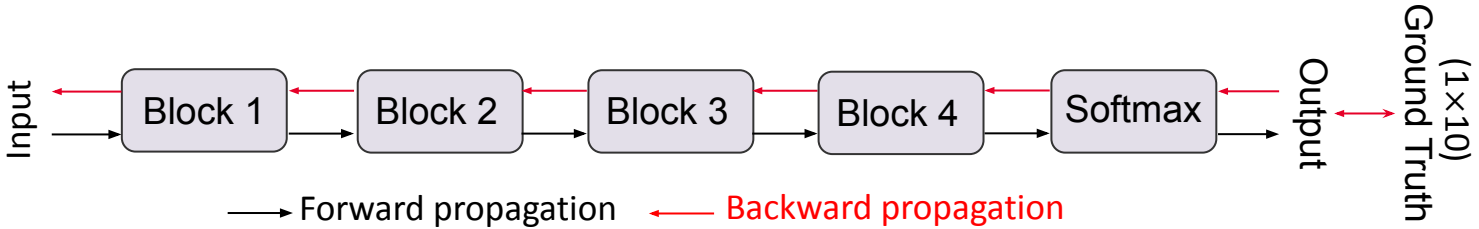
- Loss functions quantify the difference between the DNN output and the ground truth output in the training dataset.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

L2 loss

$$L = -\frac{1}{m}\sum_{i=1}^{m}(y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i))$$

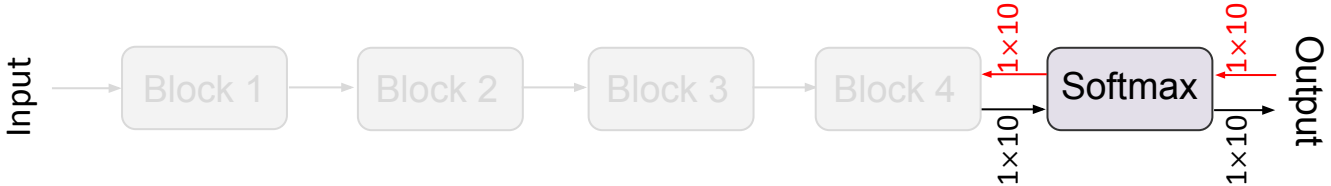
Cross-entropy loss



Softmax

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

- Domain: $(-\infty, \infty)$
- Range: [0,1]

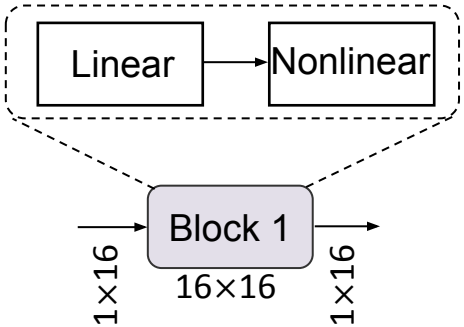


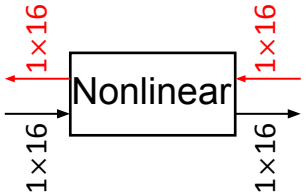
$$\frac{ds}{dz} = diag(s) - ss^T$$

When s has a dimension of 3

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

Backpropagation for Nonlinear Layers



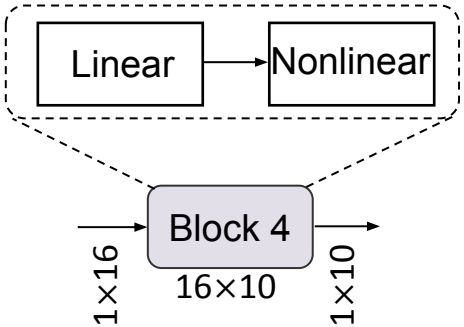


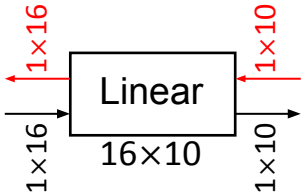
- Due to the elementwise nature, usually the nonlinear layer does not change the input and output shape during both forward and backward passes.

Backpropagation for Nonlinear Layers

- Tanh: $tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$ $\quad 1 - tanh^2(x)$
- ReLU: $ReLU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}$ $\quad \frac{dReLU(x)}{dx} = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}$
- Leaky_ReLU: $Leaky\_ReLU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ ax, & \text{otherwise} \end{cases}$ $\quad \frac{dLeaky\_ReLU(x)}{dx} = \begin{cases} 1, & \text{if } x \geq 0 \\ a, & \text{otherwise} \end{cases}$
- Sigmoid: $\sigma(x) = \frac{1}{1 + e^{-x}}$ $\quad \sigma'(x) = \frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x))$

Backpropagation for Nonlinear Layers

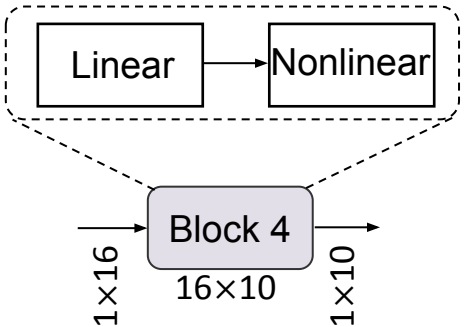


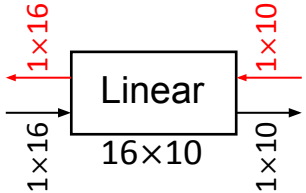


- Due to the elementwise nature, usually the nonlinear layer does not change the input and output shape during both forward and backward passes.

Fully-connected layers (Linear layers)

Y = XW + b





$$\underset{1\times16}{\frac{dL}{dX}} = \frac{dL}{dY}\frac{dY^\top}{dX} = \underset{1\times10}{\frac{dL}{dY}}\underset{10\times16}{W^\top}$$ Derivative wrt data

$$\frac{dL}{db} = \frac{dL}{dY}$$ Derivative wrt bias

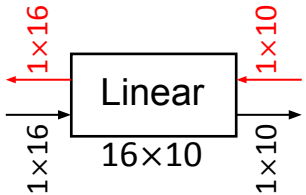
$$\underset{16\times10}{\frac{dL}{dW}} = \underset{16\times1}{X^\top}\underset{1\times10}{\frac{dL}{dY}}$$ Derivative wrt weight

NYU SAI LAB

Fully-connected layers (Linear layers)

$$Y = XW + b$$

- X (input activations): $B \times C_{in}$
- Y (output activations): $B \times C_{out}$
- W (weights): $C_{in} \times C_{out}$
- b (bias): $1 \times C_{out}$



$$\underset{1\times16}{\frac{dL}{dX}} = \frac{dL}{dY}\frac{dY^\top}{dX} = \underset{1\times10\ 10\times16}{\frac{dL}{dY}W^\top}$$ Derivative wrt data

$$\frac{dL}{db} = \frac{dL}{dY}$$ Derivative wrt bias

$$\underset{16\times10}{\frac{dL}{dW}} = \underset{16\times1}{X^\top}\underset{1\times10}{\frac{dL}{dY}}$$ Derivative wrt weight

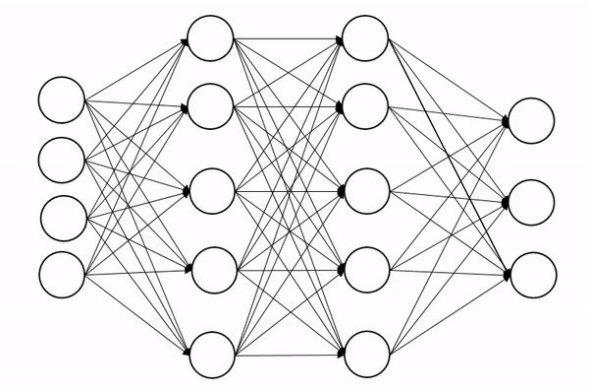
NYU SAI LAB

Weight Decay and Dropout

- The loss function is usually attached with a weight decay loss to penalize the complexity of the function and prevent the overfitting.

$$L = L + \lambda||W||^2$$

- Dropout refers to the practice of disregarding certain nodes in a layer at random during training.
- All the nodes will be there during inference.
- Can be used to prevent overfitting and reduce the dependency on any one of a single node.

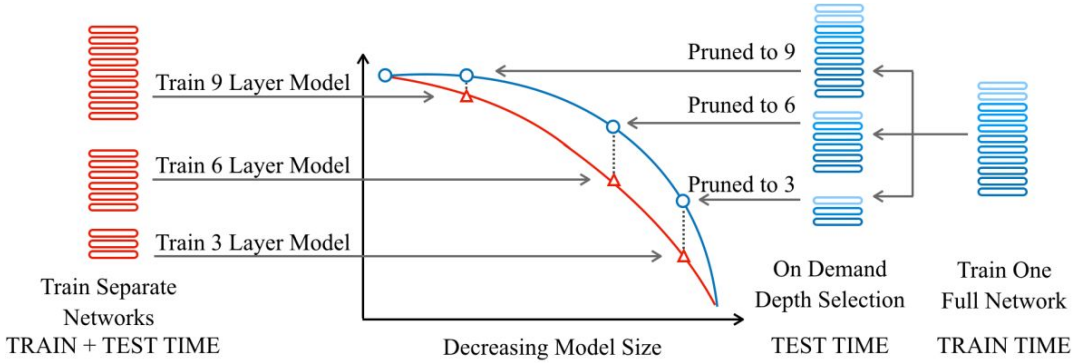


NYU SAI LAB

Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." *The journal of machine learning research* 15.1 (2014): 1929-1958.

Layer Dropout

- LayerDrop, a form of structured dropout, which has a regularization effect during training and allows for efficient skipping at inference time.
- It is possible to select sub-networks of any depth from one large network without having to finetune them and with limited impact on performance.
- Usually used in transformer.

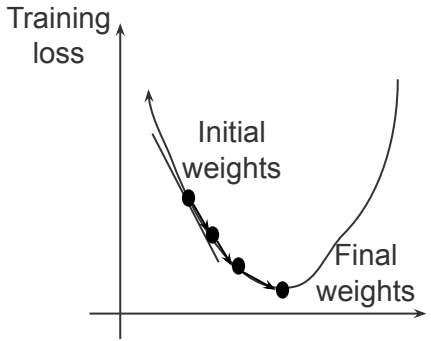


NYU SAI LAB

Fan, Angela, Edouard Grave, and Armand Joulin. "Reducing transformer depth on demand with structured dropout." *arXiv preprint arXiv:1909.11556* (2019).

DNN Training Process

- An optimizer is a crucial element that adjusts DNN parameters during training. Its primary role is to minimize the training loss defined by the loss function.
  - Epoch: The number of times the algorithm runs on the whole training dataset.
  - Batch: The size of block of dataset that is used to update the model weights.
  - Iteration: total_training_data_size/Batch
  - Learning rate: It is a parameter that provides the model a scale of how much model weights should be updated.



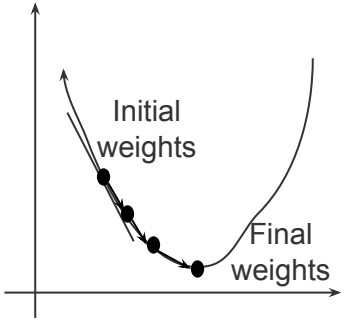
Initialized W for each layer.
For each epoch:
    Shuffle the training data.
    For each batch in training datasize:
        Perform the forward propagation
        Compute the loss and weight gradient via backward propagation.
        Update the weights
    Update the learning rate (if necessary)

Batch, Iteration and Epoch

- A data batch refers to a subset of the entire training dataset used to train the network.
- Iteration refers to a single update of the model's parameters.
- An epoch represents one complete pass through the entire training dataset. Here's what typically happens during an epoch:
  - For example, if you have 1,000 training examples and you use a batch size of 100, it would take 10 iterations to complete one epoch.
- The composition of minibatches typically changes after every epoch during the training of a DNN.

DNN Training Process

- An optimizer is a crucial element that fine-tunes DNN parameters during training. Its primary role is to minimize the model's error or loss function, enhancing performance.
  - Epoch: The number of times the algorithm runs on the whole training dataset.
  - Batch: The size of block of dataset that is used to update the model weights dataset.
  - Iteration: total_trainingdata_size/Batch
  - Learning rate: It is a parameter that provides the model a scale of how much model weights should be updated.



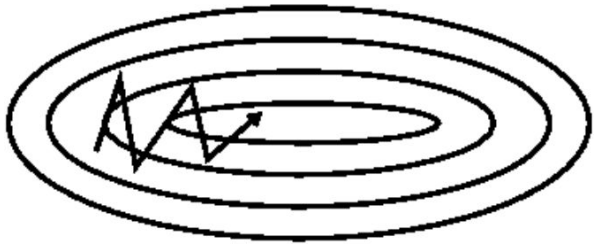
Initialized W for each layer.
For each epoch:
    Shuffle the training data.
    For each batch in training datasize:
        Perform the forward propagation
        Compute the loss and weight gradient via backward propagation.
        Update the weights
    Update the learning rate (if necessary)

NYU SAI LAB

Stochastic Gradient Descent (with Momentum)

- $W' = W - \eta \, dL/dW$
- Due to the significant noise introduced during the SGD process, it is beneficial to stabilize the process.
- $W' = W - \eta g_t \quad g_t \rightarrow s g_{t-1} + (1-s) dL/dW$, s is a hyperparameter between 0 and 1, close to 1.

SGD without momentum



SGD with momentum

RMSProp

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

- All operations are elementwise operations.
- When the variance of gradients is high, we scale down the gradient as we want to be more conservative and vice versa.

g = [0.02, -0.04, 1.6, -0.01]

1.6 will be scaled down with RMSProp

http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

Adam Optimizer

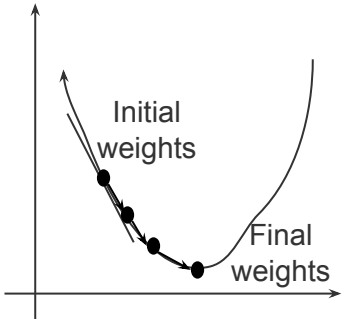
Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

- Combine RMSProp with Momentum SGD.
- By adapting the learning rate during training, Adam converges much more quickly than SGD.

Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014).

DNN Training Process

- An optimizer is a crucial element that fine-tunes DNN parameters during training. Its primary role is to minimize the model's error or loss function, enhancing performance.
  - Epoch: The number of times the algorithm runs on the whole training dataset.
  - Batch: The size of block of dataset that is used to update the model weights dataset.
  - Iteration: total_trainingdata_size/Batch
  - Learning rate: It is a parameter that provides the model a scale of how much model weights should be updated.



Initialized W for each layer.
For each epoch:
    Shuffle the training data.
    For each batch in training datasize:
        Perform the forward propagation
        Compute the loss and weight gradient via backward propagation.
        Update the weights
    Update the learning rate (if necessary)

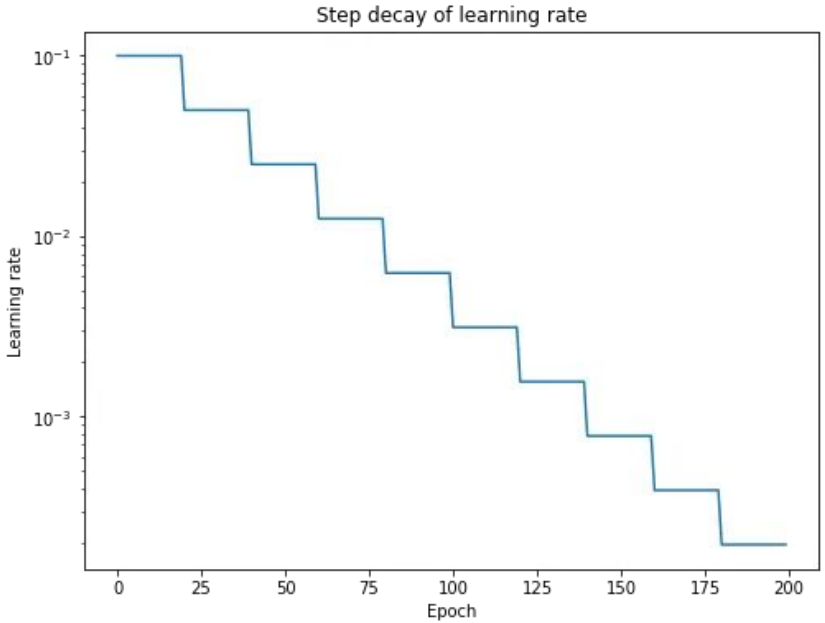
NYU SAI LAB

Learning Rate Scheduler

- Learning rate η is an important hyperparameter for training the DNNs.
- A large learning rate can help the algorithm to converge quickly. But it can also cause the algorithm to bounce around the minimum without reaching it or even jumping over it if it is too large.
- If the learning rate is too small, the optimizer may take too long to converge or get stuck in a plateau if it is too small.

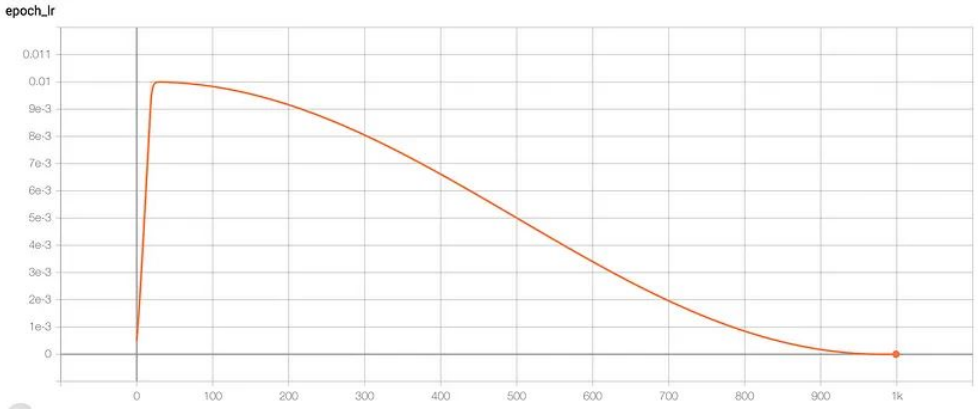
$$W' = W - \eta g_t$$

Multistage Learning Rate



- The learning rate is reduced by a fixed amount after every T epochs.
- Typically, the learning rate is reduced to 10% of its value after every T epochs.
- Widely used in image classification task.

Cosine Learning Rate



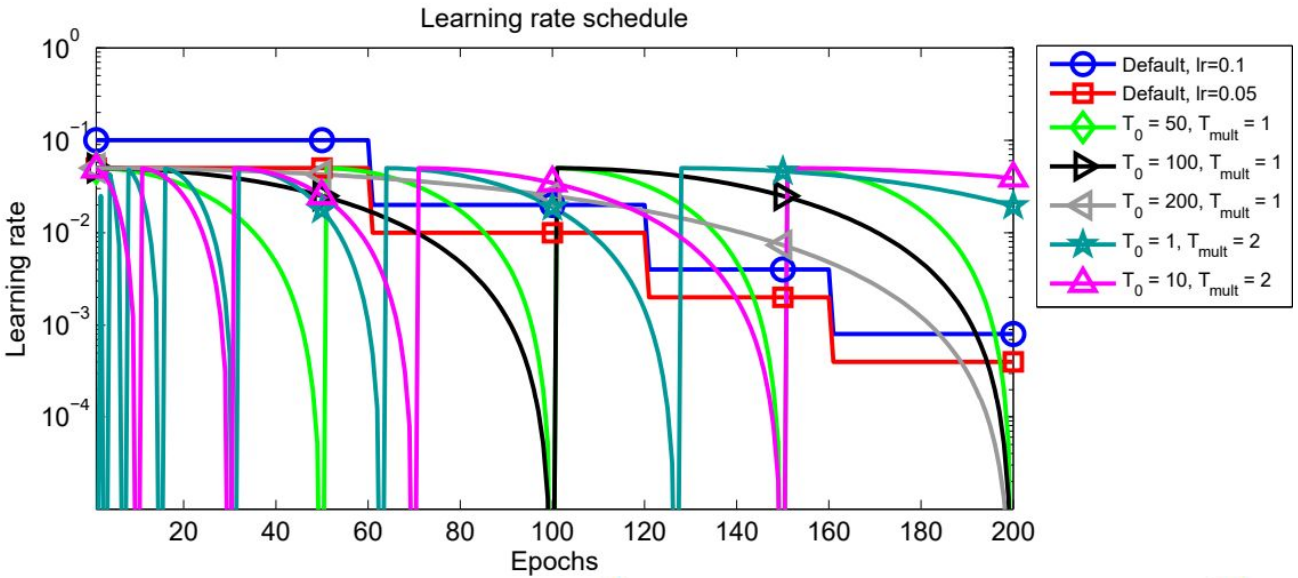
- We propose to periodically simulate warm restarts of SGD, where in each restart the learning rate is initialized to some value and is scheduled to decrease.
- Periodic restart can effectively avoid local minima and saddle points during the training.

$$\eta_t = \eta_{min}^i + \frac{1}{2}(\eta_{max}^i - \eta_{min}^i)(1 + \cos(\frac{T_{cur}}{T_i}\pi)),$$

NYU SAI LAB

Loshchilov, Ilya, and Frank Hutter. "Sgdr: Stochastic gradient descent with warm restarts." *arXiv preprint arXiv:1608.03983* (2016).

Cosine Learning Rate

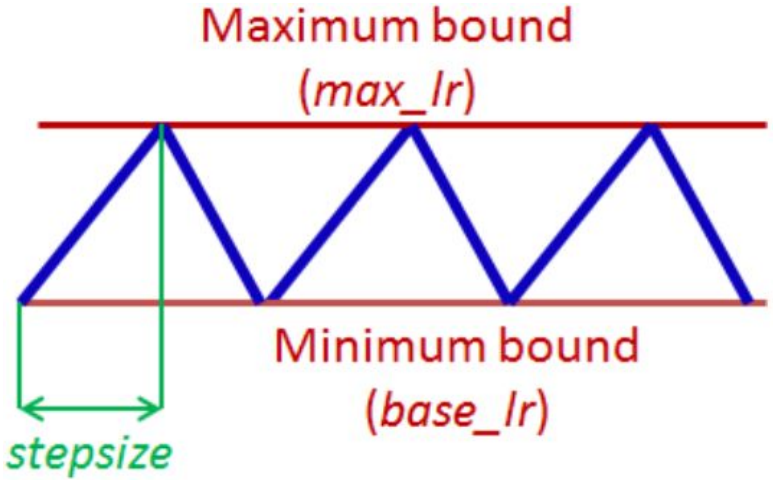


- Tcur accounts for how many iterations have been performed since the last restart.
- Tcur is updated at each iteration t.
- The SGD is restarted once Ti epochs are performed, where i is the index of the run.
- Ti may increase with i.

$$\eta_t = \eta_{min}^i + \frac{1}{2}(\eta_{max}^i - \eta_{min}^i)(1 + \cos(\frac{T_{cur}}{T_i}\pi)),$$

Loshchilov, Ilya, and Frank Hutter. "Sgdr: Stochastic gradient descent with warm restarts." *arXiv preprint arXiv:1608.03983* (2016).

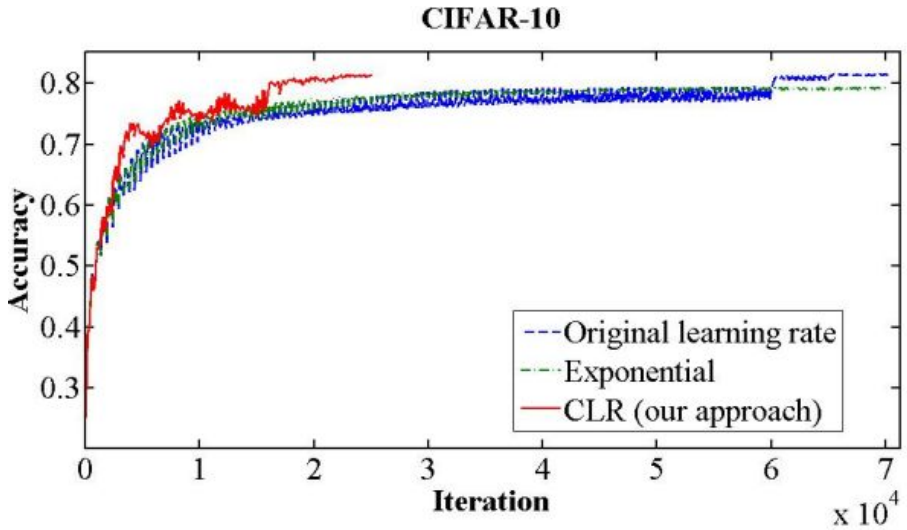
Cyclical Learning Rate



- Increasing the learning rate might have a short term negative effect and yet achieve a longer term beneficial effect.

Smith, Leslie N. "Cyclical learning rates for training neural networks." *2017 IEEE winter conference on applications of computer vision (WACV)*. IEEE, 2017.

Cyclical Learning Rate

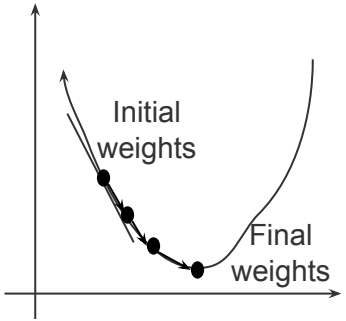


- The red curve shows the result of training with cyclical learning rate achieves the shortest convergence time.

Smith, Leslie N. "Cyclical learning rates for training neural networks." *2017 IEEE winter conference on applications of computer vision (WACV)*. IEEE, 2017.

DNN Training Process

- An optimizer is a crucial element that fine-tunes DNN parameters during training. Its primary role is to minimize the model's error or loss function, enhancing performance.
  - Epoch: The number of times the algorithm runs on the whole training dataset.
  - Batch: The size of block of dataset that is used to update the model weights dataset.
  - Iteration: total_trainingdata_size/Batch
  - Learning rate: It is a parameter that provides the model a scale of how much model weights should be updated.



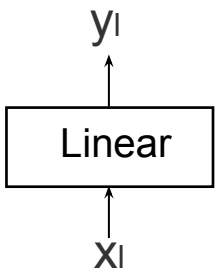
Initialized W for each layer.
For each epoch:
    Shuffle the training data.
    For each batch in training datasize:
        Perform the forward propagation
        Compute the loss and weight gradient via backward propagation.
        Update the weights
    Update the learning rate (if necessary)

NYU SAI LAB

DNN Initialization: Kaiming Initialization

- Kaiming initialization is designed for modern DNN that uses ReLU.

$$W \sim \mathcal{N}\left(0, \frac{2}{n^l}\right)$$



- Target: ensure the activation variance is the same across different layers.
- Assumption:
  - ReLU activation.
  - Weight is normally distributed with mean of zero.
  - Weight and activations are independent.

NYU SAI LAB

He, Kaiming, et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." *Proceedings of the IEEE international conference on computer vision*. 2015.

Derivation

$\mathbf{y}_l = \mathrm{W}_l \mathbf{x}_l + \mathbf{b}_l$

Assume W_l has a shape of m_l by n_l, and x_l has a size of $n_l \times 1$, then y_l has a size of $m_l \times 1$.

For each element $y_{l,i}$ of y_l, its variance $var(y_{l,i}) = var(\sum_{j=1}^{n_l} W_{l,i,j} x_{l,j}) = n_l var(W_{l,i,j} x_{l,j})$

Assume each pair of $W_{l,i,j}$ and x_j are independent random variable, then we have:

$$var(W_{l,i,j} x_{l,j}) = E(W_{l,i,j}^2 x_{l,j}^2) - E^2(W_{l,i,j} x_{l,j}) = E(W_{l,i,j}^2) E(x_{l,j}^2) - E^2(W_{l,i,j}) E^2(x_{l,j})$$

Assume $W_{l,i,j}$ follows a normal distribution with mean of 0, that is $E(W_{l,i,j}) = 0$, then:

$$var(W_{l,i,j} x_{l,j}) = var(W_{l,i,j}) E(x_{l,j}^2)$$

$$var(y_{l,i}) = n_l var(W_{l,i,j} x_{l,j}) = n_l var(W_{l,i,j}) E(x_{l,j}^2)$$

Derivation

Let see how $E(x_{l,i}^2)$ is related to the variance of $y_{l-1,j}$, where $x_{l,j}=ReLU(y_{l-1,j})$

$$E(x_{l,j}^2) = E(ReLU^2(y_{l-1,j}))$$

Then we have: $E(ReLU(y_{l-1,j})^2)$

$$= E(ReLU(y_{l-1,j})^2|y_{l-1,j} > 0)P(y_{l-1,j} > 0) + E(ReLU(y_{l-1,j})^2|y_{l-1,j} < 0)P(y_{l-1,j} < 0)$$
$$= E(ReLU(y_{l-1,j})^2|y_{l-1,j} > 0)P(y_{l-1,j} > 0) = 0.5E(y_{l-1,j}^2) = 0.5var(y_{l-1,j})$$

Therefore, we have: $E(x_{l,j}^2) = 0.5var(y_{l-1,j})$

Given this, we have:

$$var(y_{l,i}) = n_l var(W_{l,i,j})E(x_{l,j}^2) = 0.5n_l var(W_{l,i,j})var(y_{l-1,j})$$

Derivation

$$var(y_{l,i}) = (\prod_{s=2}^{l} 0.5 n_s var(W_{s,i,j}))var(y_{1,j})$$

In order to ensure the variance of y does not change, we have to make sure:

$$var(W_{s,i,j}) = \frac{2}{n_s} \quad W_{s,i,j} \sim \mathcal{N}(0, \sqrt{\frac{2}{n_s}})$$