



Lecture 0: Introduction and Course Overview

CSCI-GA 3033 / ECE-GY 9483
Efficient AI Computing: Algorithm and Implementation

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

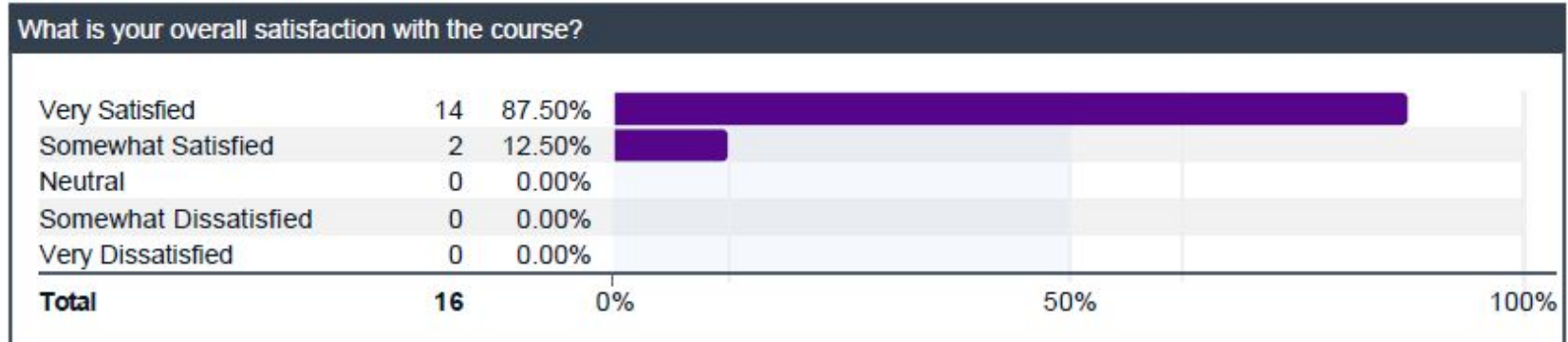
Course Information

- 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 (10%)
 - Assignments (30%): total three of them, each counts 10%
 - Midterm (30%)
 - Final project (30%)
 - Project Proposal (5%) (1 page)
 - Final Presentation (15%)
 - Final Report (10%)
- Readings:
 - Course notes and papers (optional)
 - (reference) Goodfellow, Ian. "Deep learning." (2016). <https://www.deeplearningbook.org/>
- Lecture time:
 - Wednesday: 7:10pm-9:10pm
- Office hour:
 - Friday: 1:30pm-2:30pm, or by appointment ([Zoom](#))

Course Feedback from Previous Semesters



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.

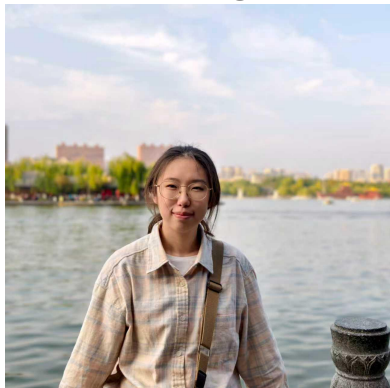
Amazing course. Professor Sai is doing a great job. Course structure, quizzes, midterm, assignments, project--All were super helpful and great.

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

Yifei Feng (CA)



Office hour: Tuesday
11:00am-12:00pm
([Zoom](#))

Raj Appana (CA)



Office hour: Wednesday
5:00pm-6:00pm
([Zoom](#))

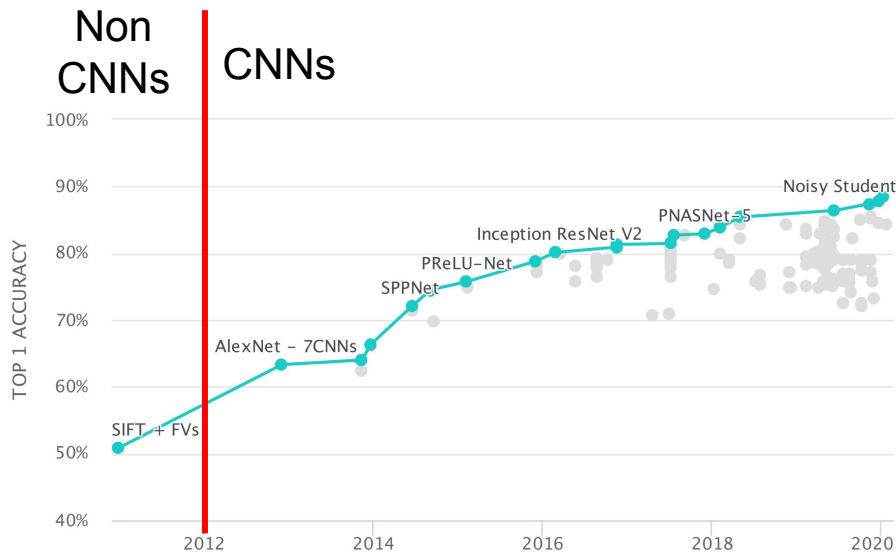
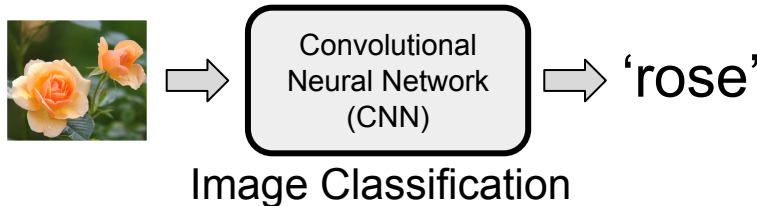
Handong Ji (Grader)



Life is Powered by Deep Learning

- Deep Neural Networks (DNNs) have achieved state-of-the-art performance across a variety of domains

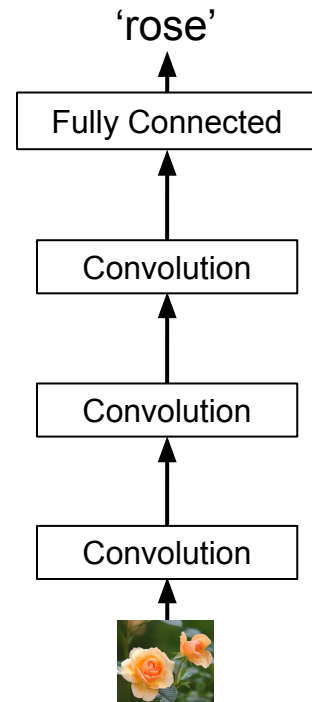
- Image Recognition 
- Video Processing 
- Natural Language Processing 
- Autonomous Driving 



- More desirable modern services are enabled by DNN

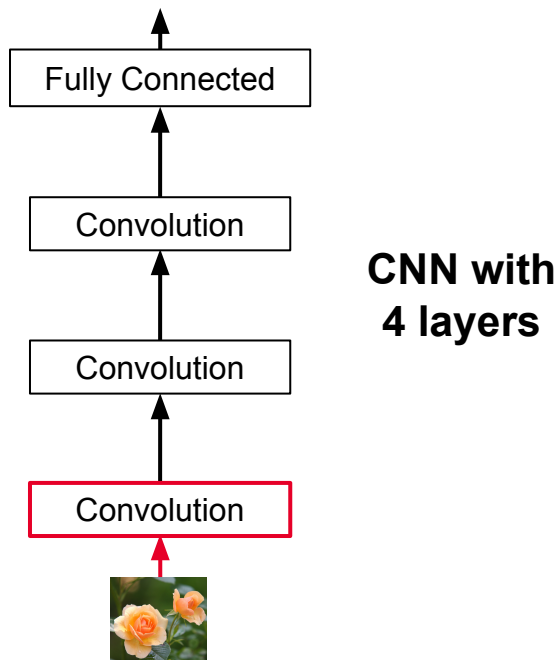
How Deep Neural Network is Executed?

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

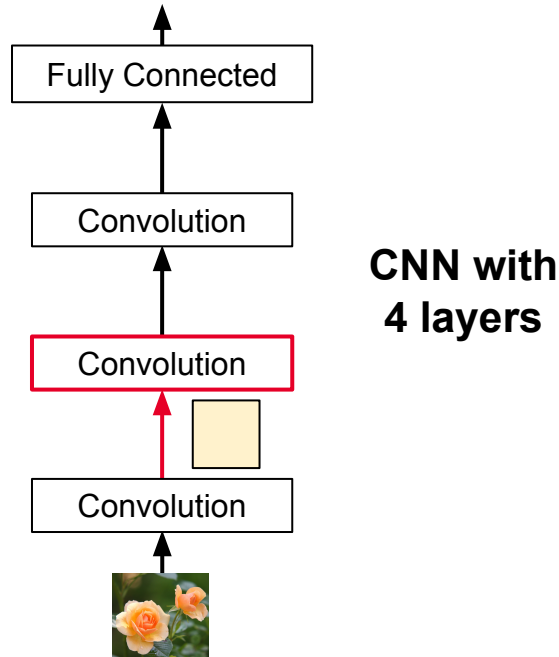


**CNN with
4 layers**

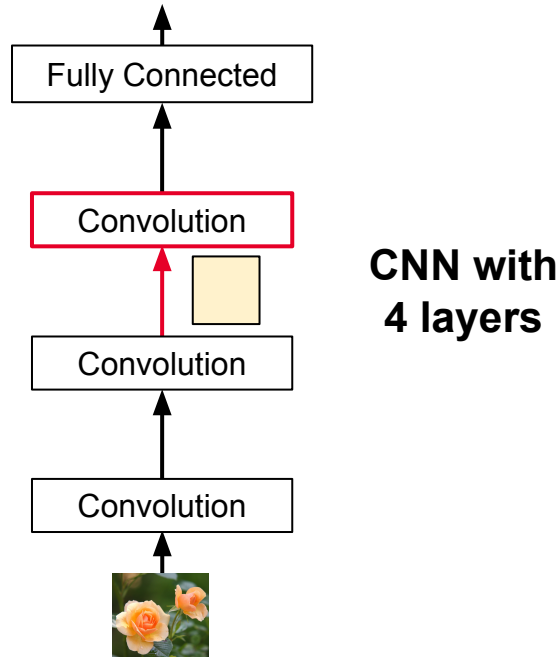
How Deep Neural Network is Executed?



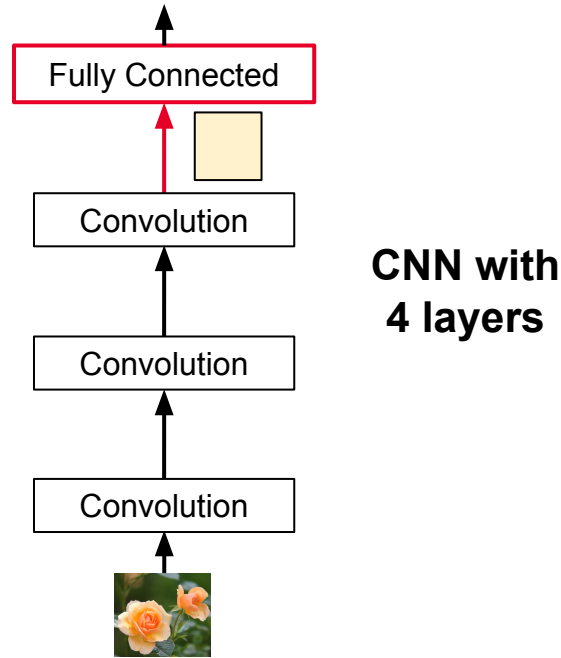
How Deep Neural Network is Executed?



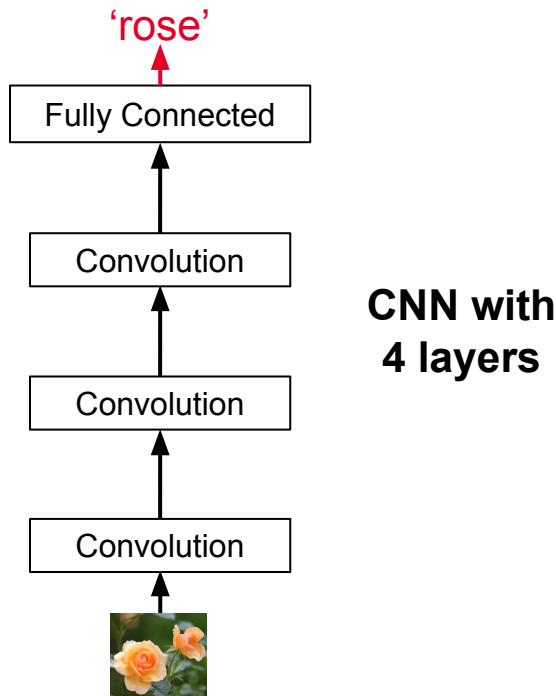
How Deep Neural Network is Executed?



How Deep Neural Network is Executed?

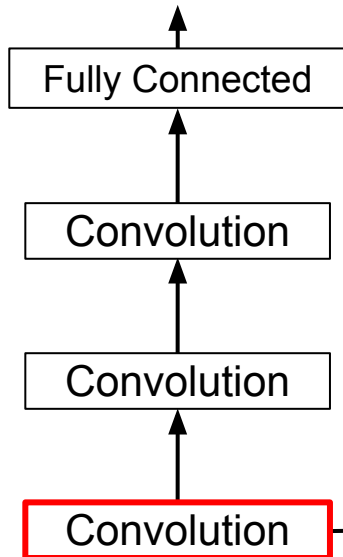


How Deep Neural Network is Executed?

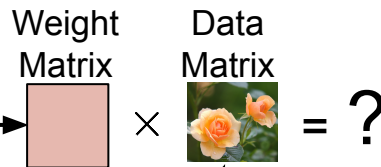


DNN Execution: A Matrix View

Layer View



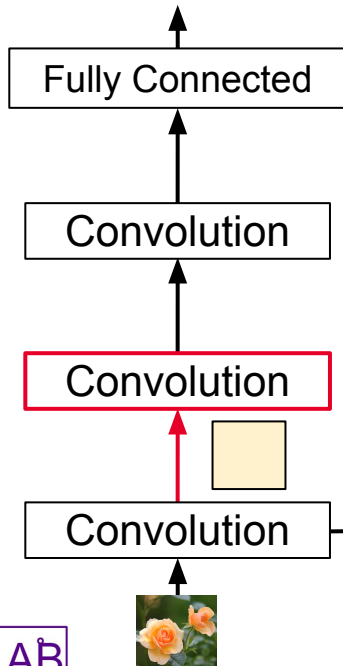
Matrix View



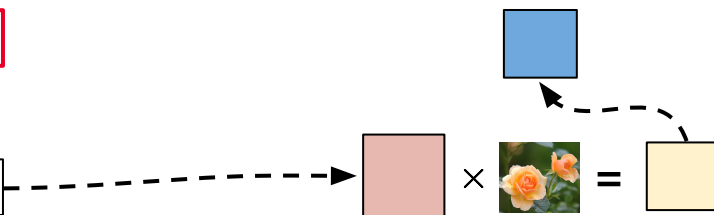
- Weight matrices are **learned** during training

DNN Execution: A Matrix View

Layer View



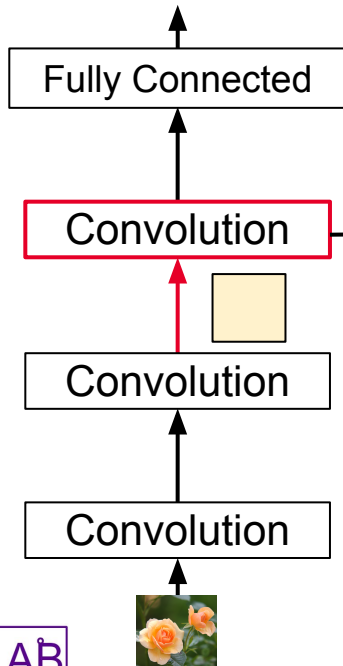
Matrix View



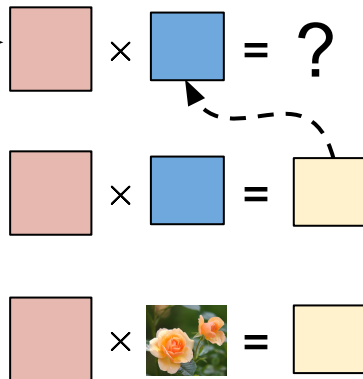
- Weight matrices are **learned** during training

DNN Execution: A Matrix View

Layer View



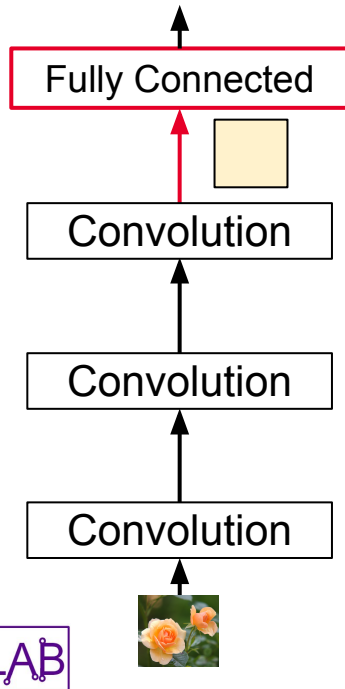
Matrix View



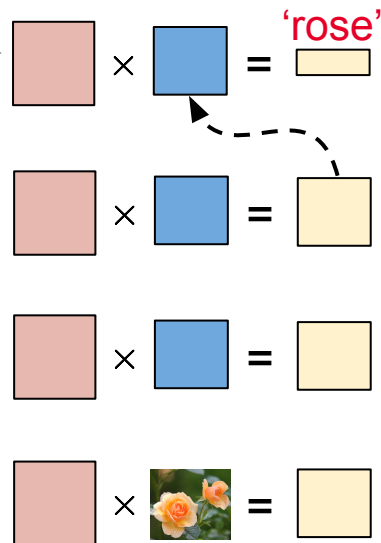
- Remaining layers follow this pattern.

DNN Execution: A Matrix View

Layer View



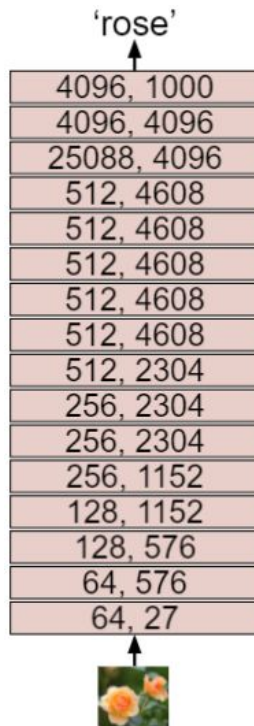
Matrix View



- Remaining layers follow this pattern

Deployment of DNN: Problems

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

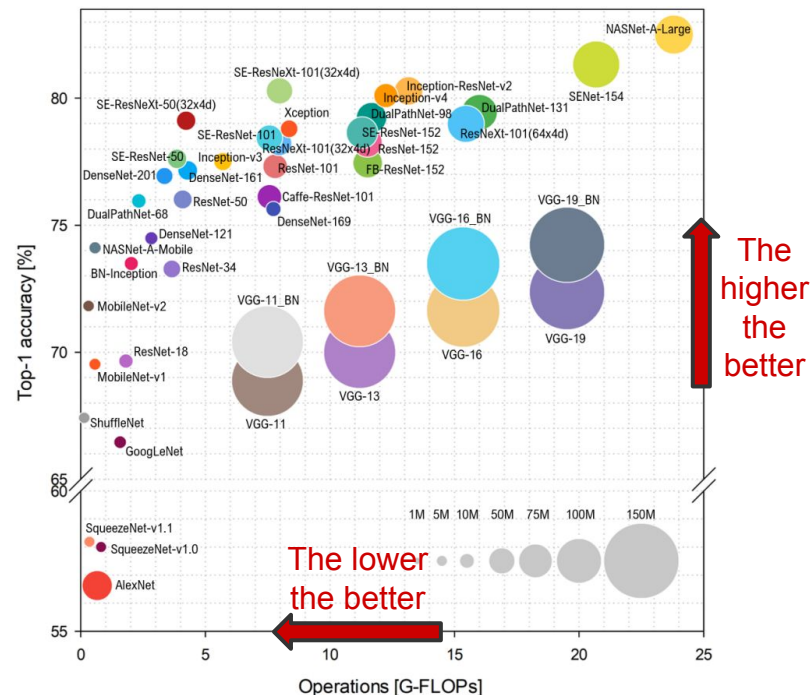


VGG-16 is a CNN with over 150M weights across 16 matrices

Deployment of DNN: Problems

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

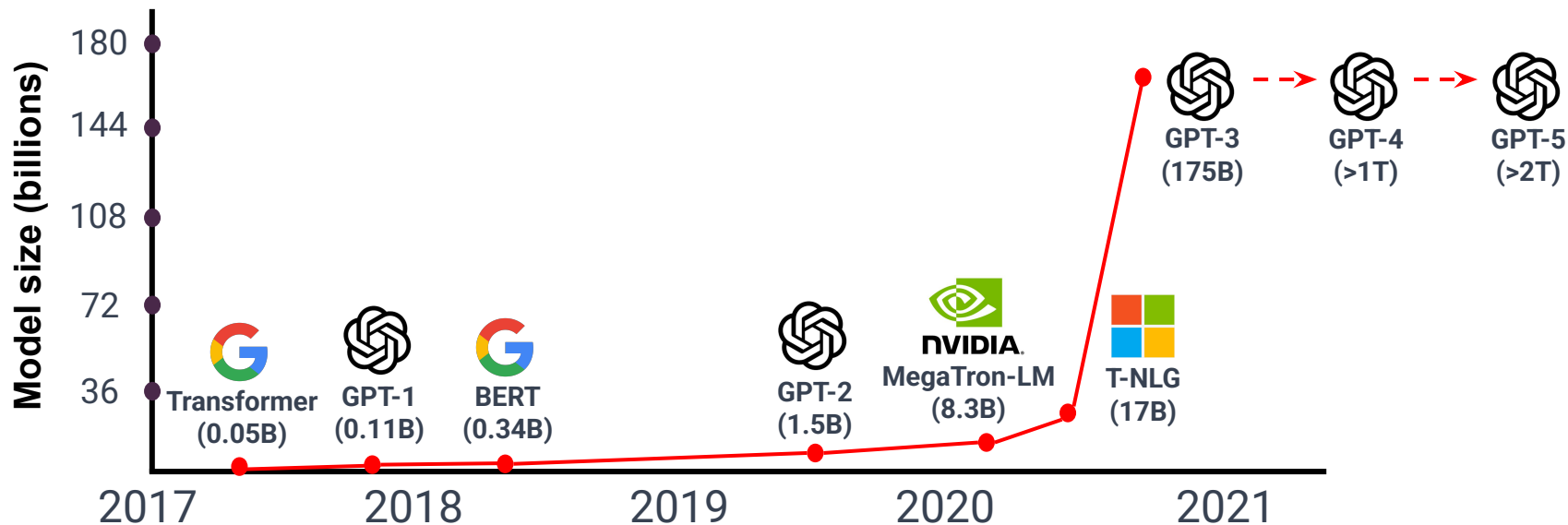
20B multiply/adds
per image



NYU SAI LAB



Cost of Large Models



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

The Cost of Large Models

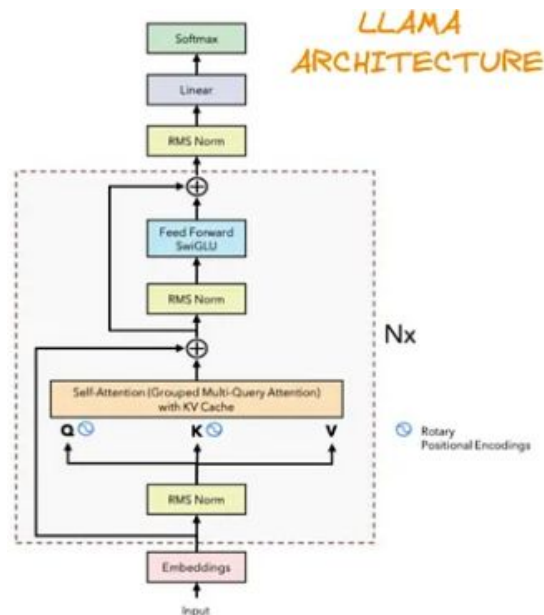


- Training GPT-4 required **25,000 A100 GPUs** over several weeks.
- **Cost:** Renting a single high-end GPU on cloud services like AWS can cost **\$3–\$5 per hour**. Training GPT-4 is estimated to cost **\$63-100 million** on cloud computing resources.

Efficient AI: An Emerging Area

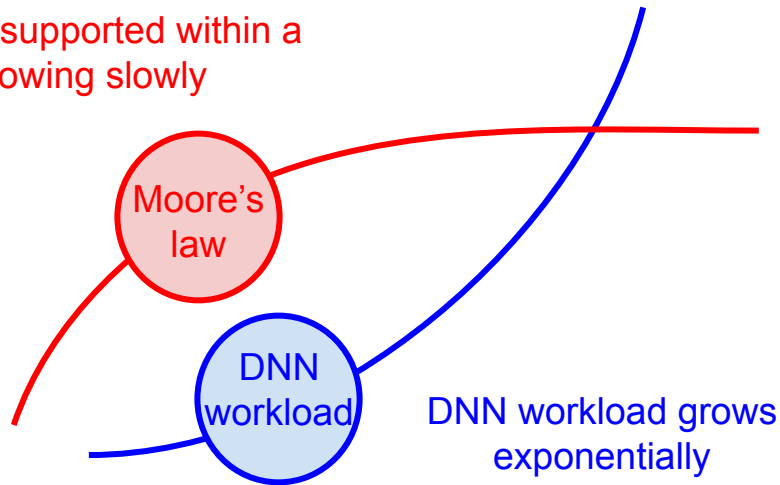
Model Size	FP16	FP8	INT4
8B	16 GB	8 GB	4 GB
70B	140 GB	70 GB	35 GB
405B LLaMA 3.1	810 GB	405 GB	203 GB

Design more aggressive and efficient AI model is of paramount importance



Efficient AI: An Emerging Area

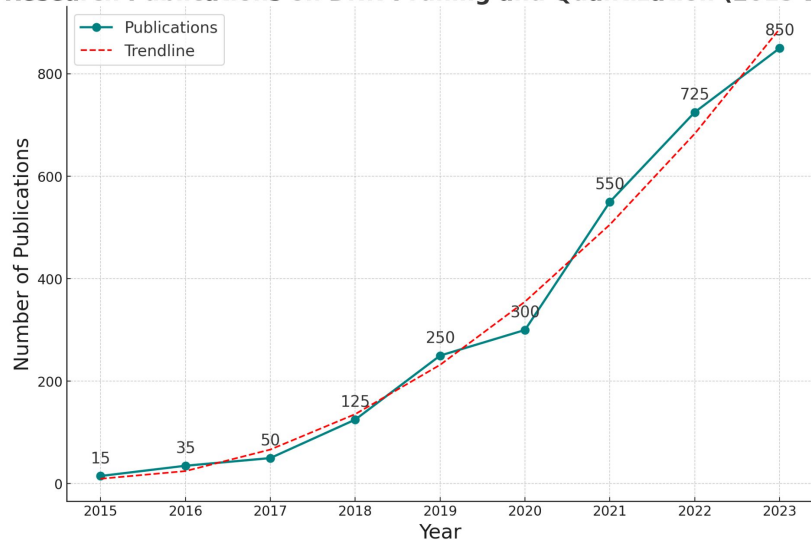
The amount of compute supported within a hardware unit is growing slowly



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

Efficient AI: An Emerging Area

Research Publications on DNN Pruning and Quantization (2015-2023)



- 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.

Efficient AI: An Emerging Area

Infra Hardware TF

Sunnyvale, CA + 2 more

Visiting Researcher - AI Accelerators

Meta

Harrisburg, PA • via Monster

3 days ago Full-time

Hardware System

Menlo Park, CA AI

Research Scientist-AI Accelerator Design

IBM

Yorktown Heights, NY • via Karkidi

\$ 120K-190K a year Full-time Health insurance Dental insurance Paid time off

Silicon Hardware

Sunnyvale, CA + 2 more

Machine Learning Engineer - Efficient Machine Learning

Bose Corporation, U.S.A

Anywhere • via Workday

4 days ago Work from home Full-time

Visiting Research

Menlo Park, CA AI

Machine Learning/AI Engineer

Advanced Micro Devices, Inc

Boxborough, MA • via AMD Careers

Full-time

Director, AI Resea

London, UK AI Res

Sr Machine Learning Engineer, AI Software Solutions

Advanced Micro Devices, Inc

Fishkill, NY • via Monster

Full-time

Artificial Intelligence Engineer

Tata Consultancy Services

Malvern, PA • via LinkedIn

21 hours ago \$ 130K-160K a year Full-time

Accelerator Design

Karkidi

time Health insurance Paid time off Dental insurance

engineering/ AI accelerator compiler and Runtime

gree.

anager, AI Compiler

ome Full-time Health insurance

ation Engineer for Intel AI Accelerator

nce Paid time off

alent practical experience.

aging).

ems ML - Frameworks / Compilers / Kernels

reers Jobs

ems ML - Frameworks / Compilers / Kernels

ilter

Jc



AI Tech Startups/Unicorns



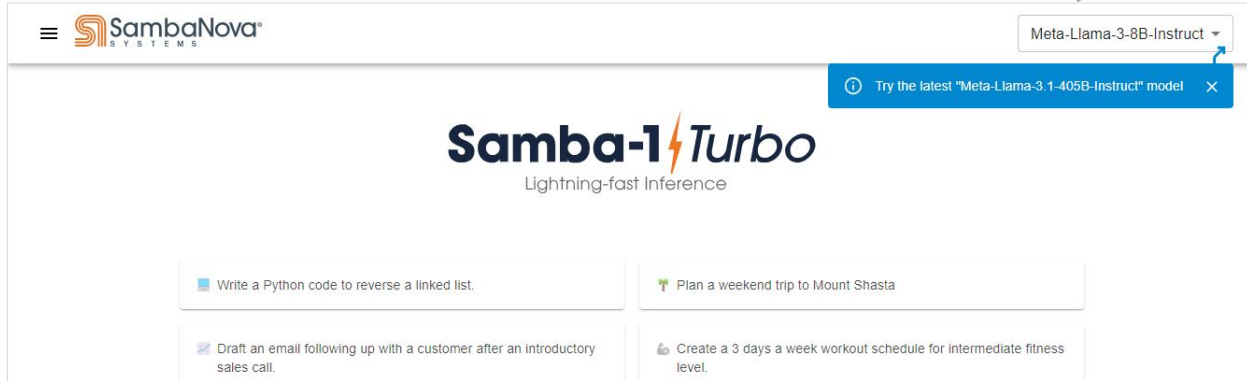
Products ▾ Technology ▾ Resources ▾ Solutions ▾ Community ▾ About ▾

Contact Us

Unleash the Fastest Private Enterprise GPT

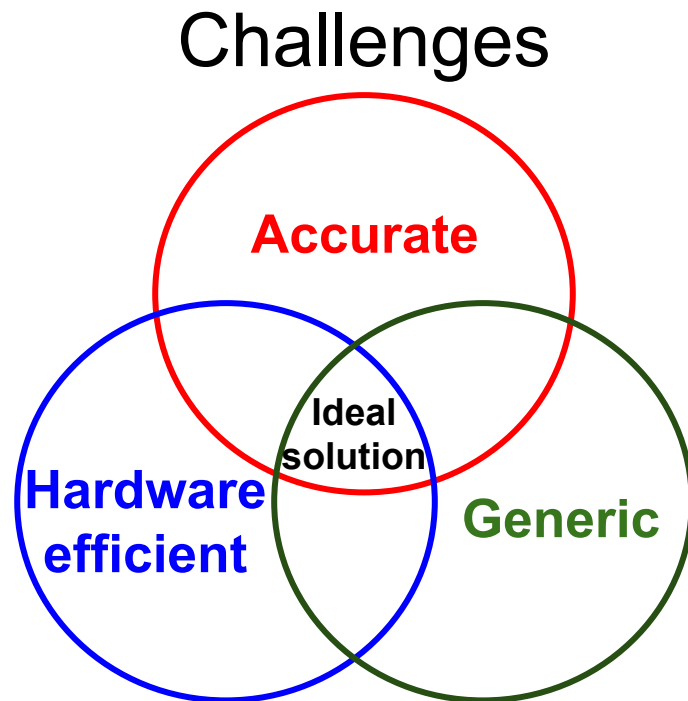
World record performance with Llama 3.1 405b on the only generative AI platform with full accuracy.

Try it now!

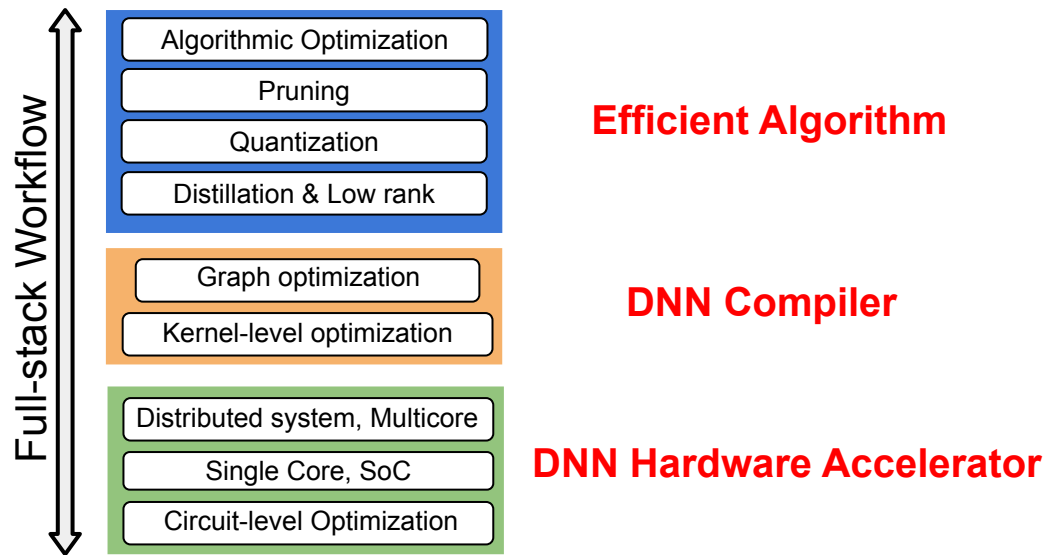


NYU SAI LA

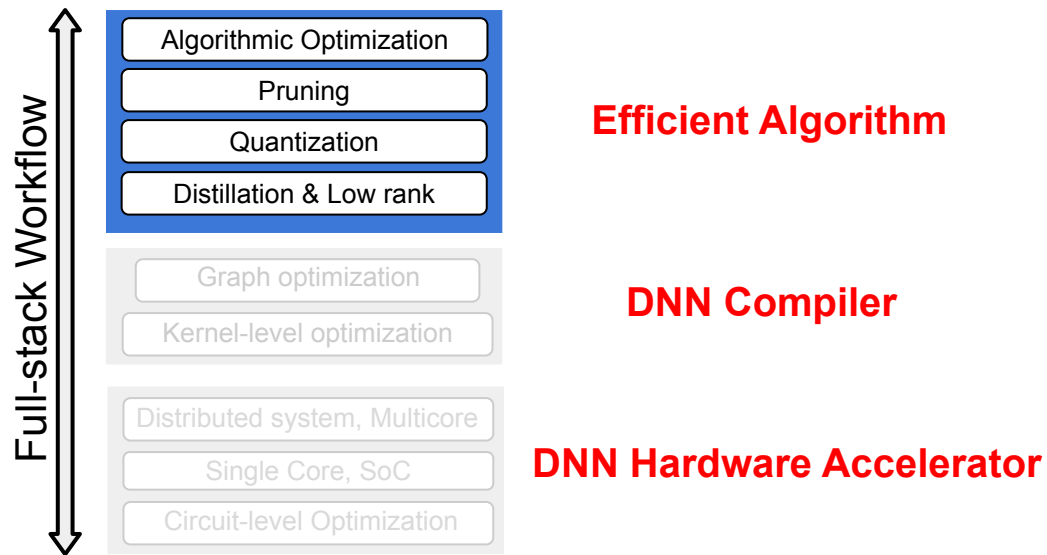
Efficient AI: An Emerging Area



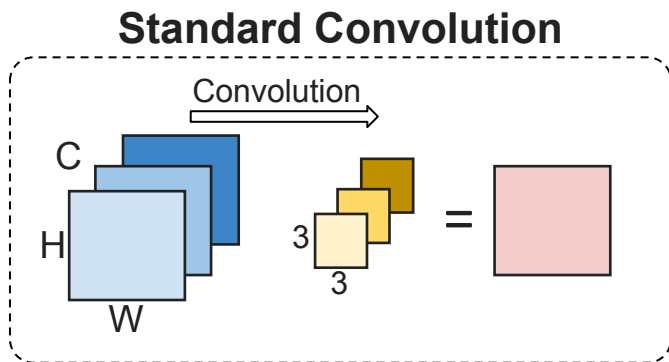
Efficient AI: Full-stack Workflow



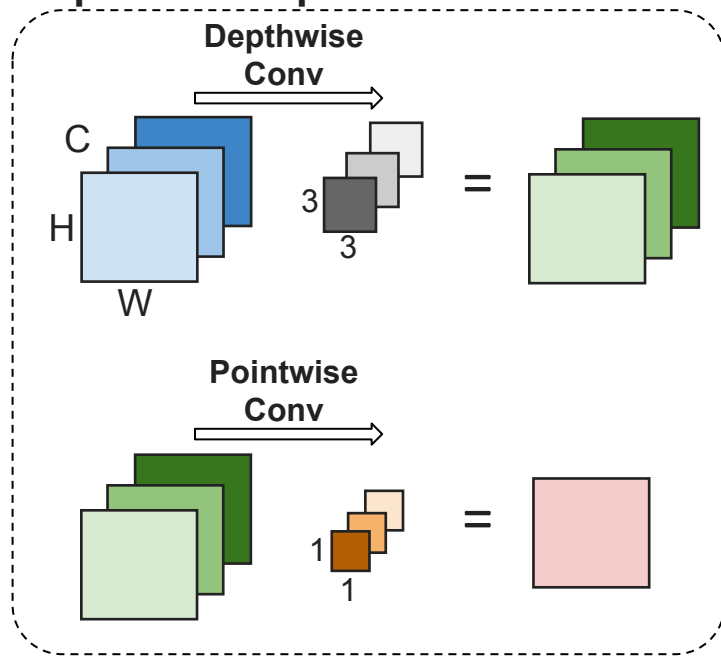
Efficient AI: Full-stack Workflow



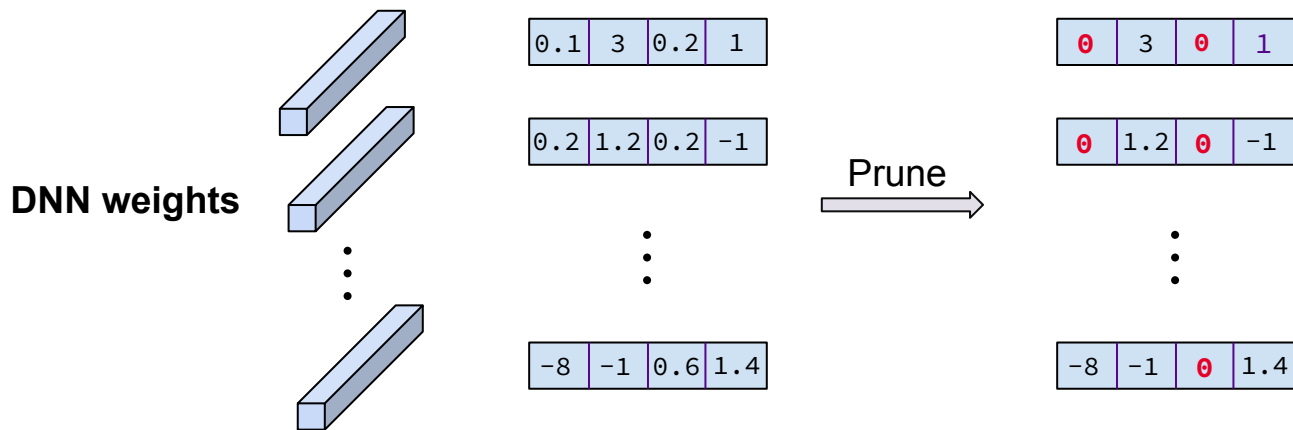
Algorithmic Optimization



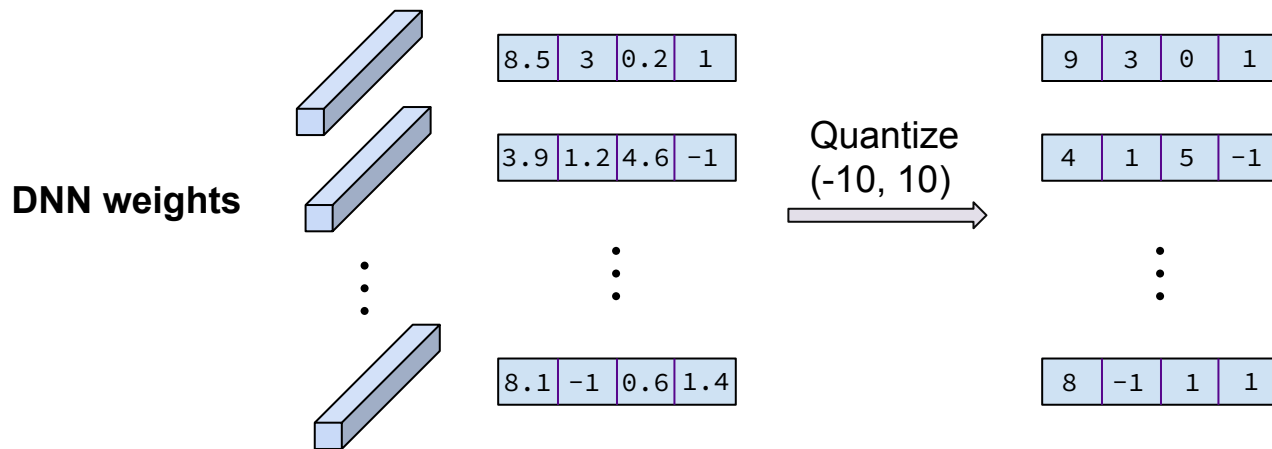
Depthwise Separable Convolution



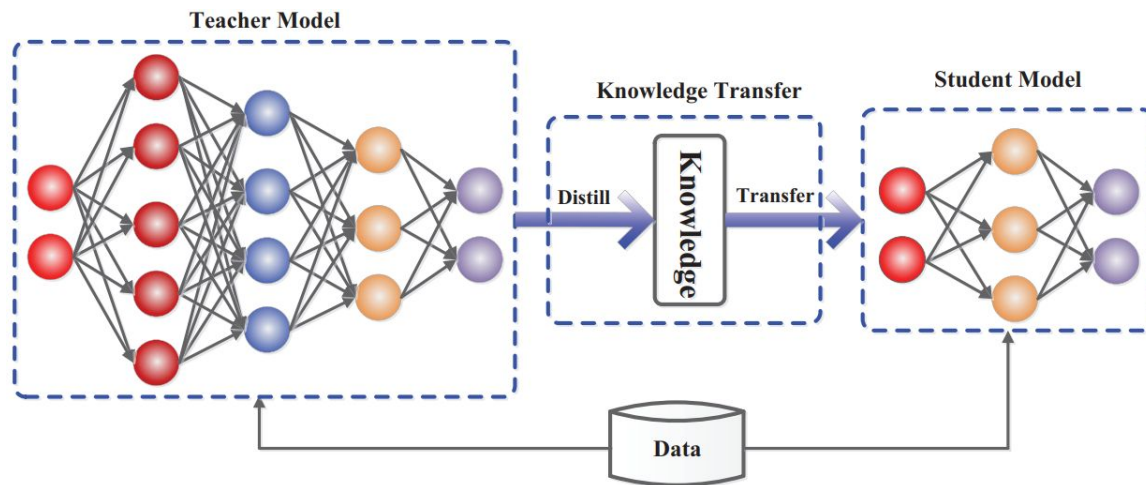
Efficient DNN Algorithm: Pruning



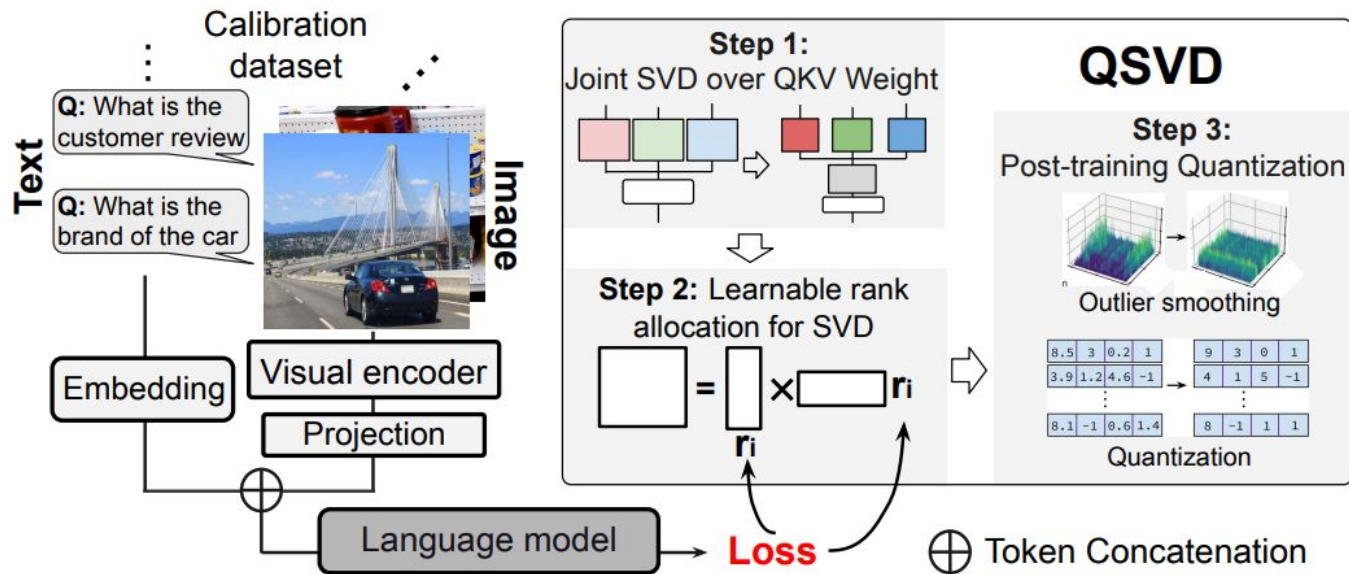
Efficient DNN Algorithm: Quantization



Knowledge Distillation

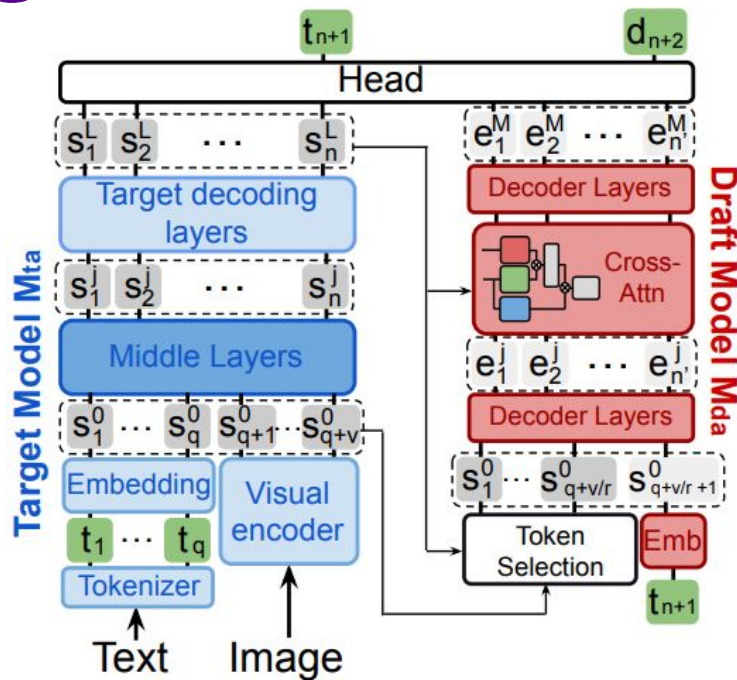
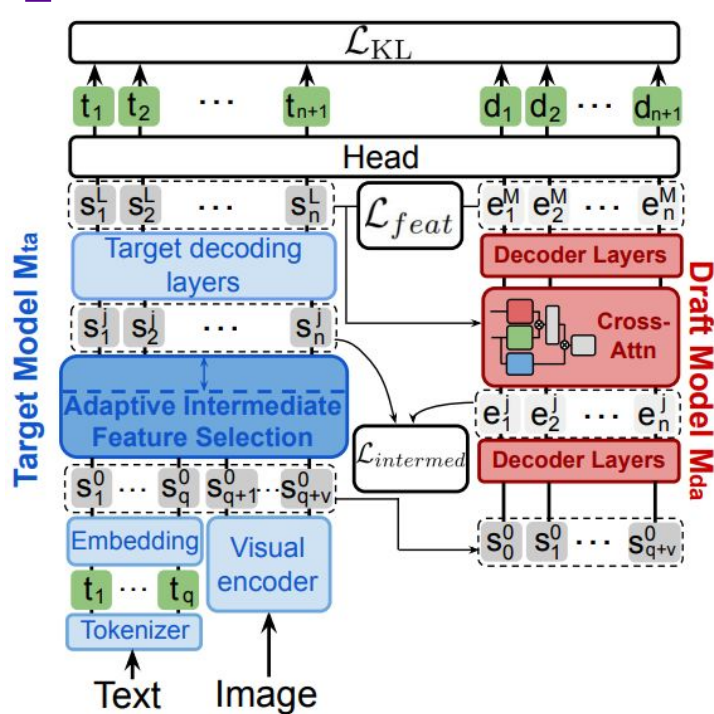


QSVD



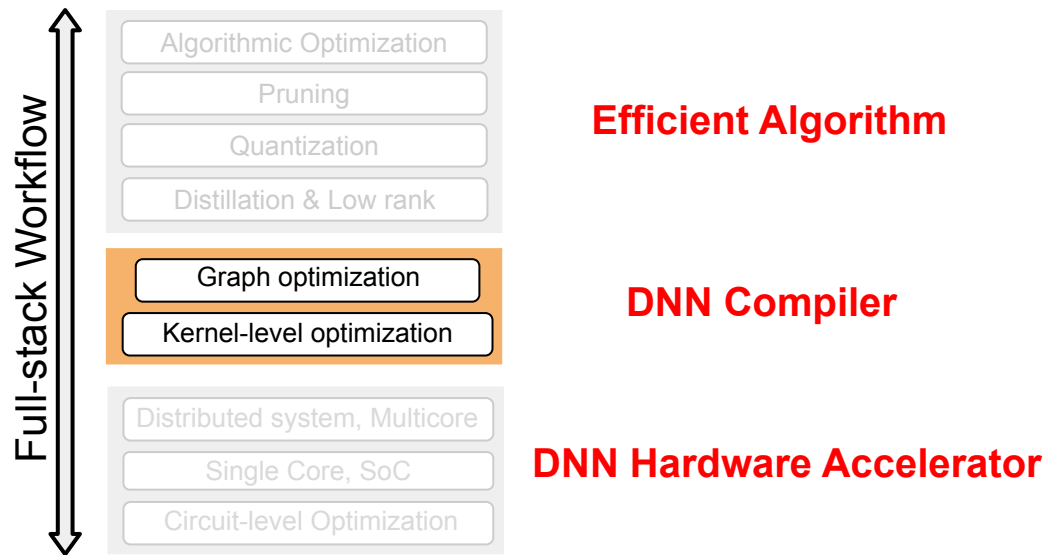
- We propose leveraging Singular-Value Decomposition over the joint query (Q), key (K), and value (V) weight matrices to reduce KV cache size and computational overhead.

Speculative Decoding with DREAM



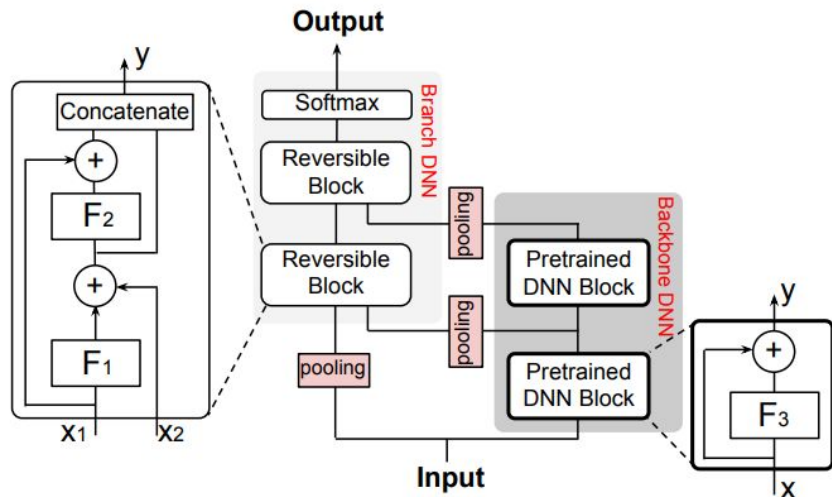
- We introduce DREAM, a novel speculative decoding framework tailored for VLMs.

Efficient AI: Full-stack Workflow

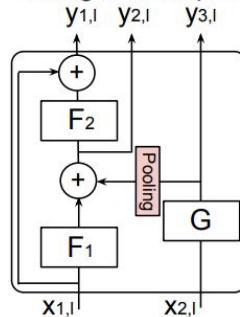


Graph Level Optimization

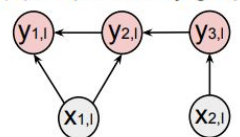
CAMEL Training



(a) Computation during forward pass



(b) Dependency graph

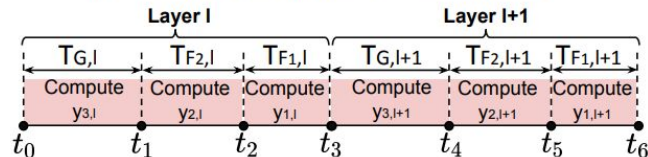


(c) Pseudo instruction

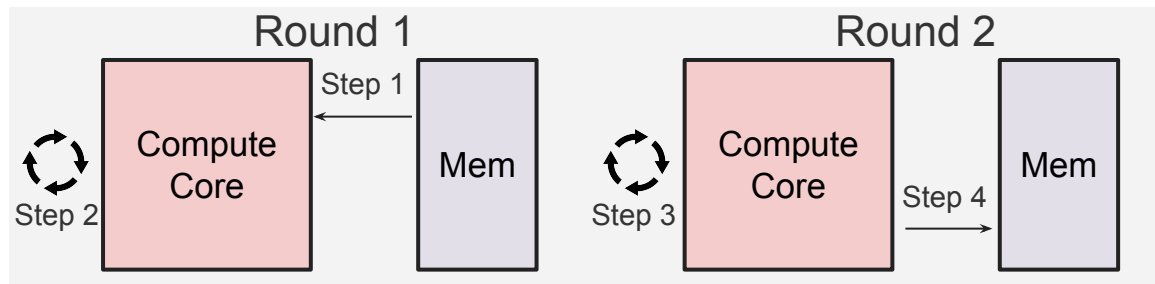
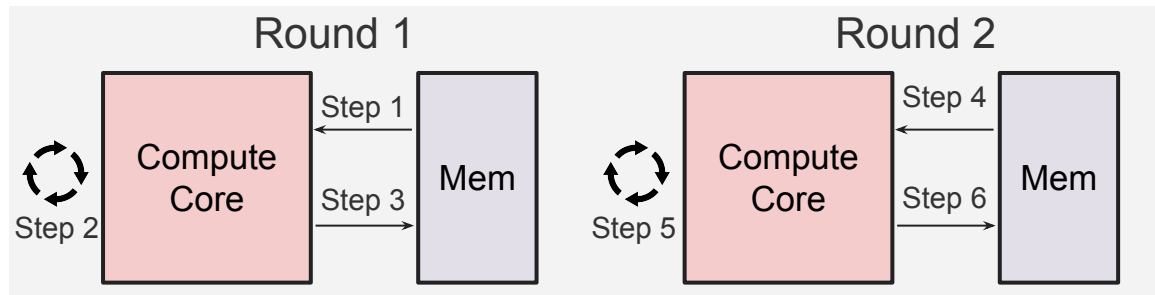
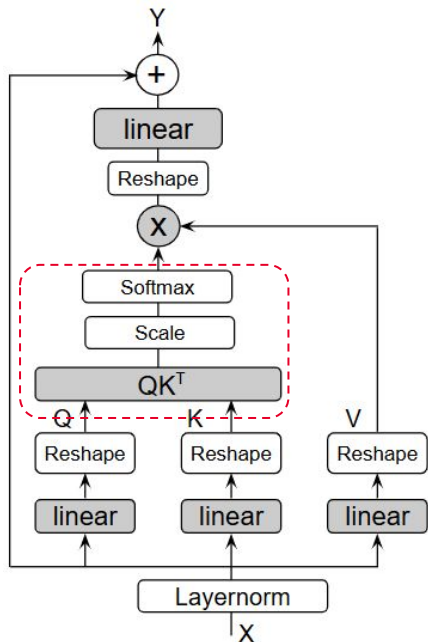
```

Compute  $y_{3,i} = G(x_{2,i})$ 
Overwrite  $x_{2,i}$  with  $y_{3,i}$ 
Compute  $y_{2,i} = \text{Pool}(y_{3,i}) + F_1(x_{1,i})$ 
Save  $y_{2,i}$ 
Compute  $y_{1,i} = x_{1,i} + F_2(y_{2,i})$ 
Overwrite  $x_{1,i}$  with  $y_{1,i}$ 
    
```

(d) Computation pattern of forward pass

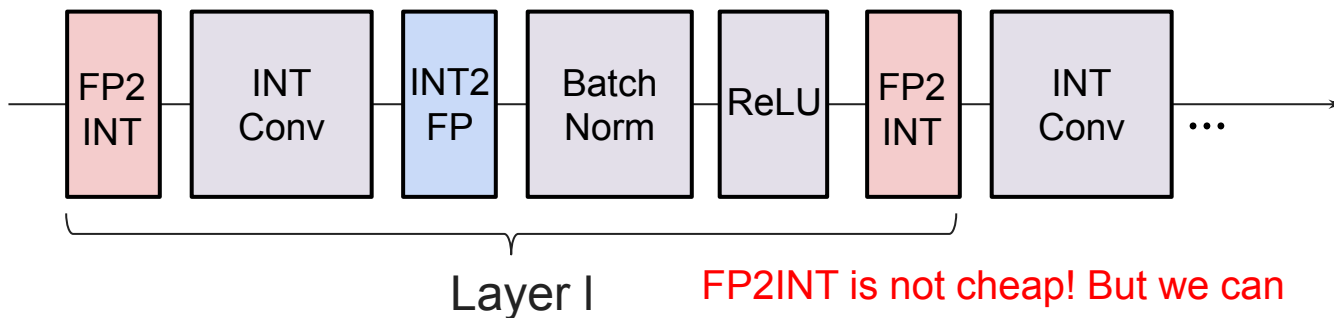


System Level Optimization



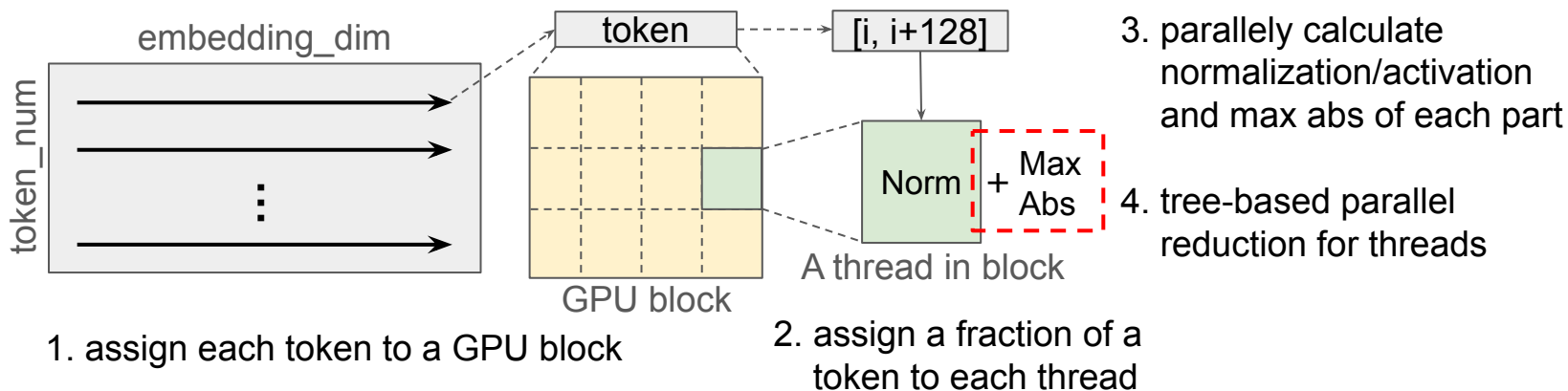
System Level Optimization

- 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 : $x_c = \text{Clip}(x, L, -L)$
 - Calculate the INT representation: $x_{int} = \text{round}(x_c / s)$
 - Rescale: $x_q = s x_{int}$



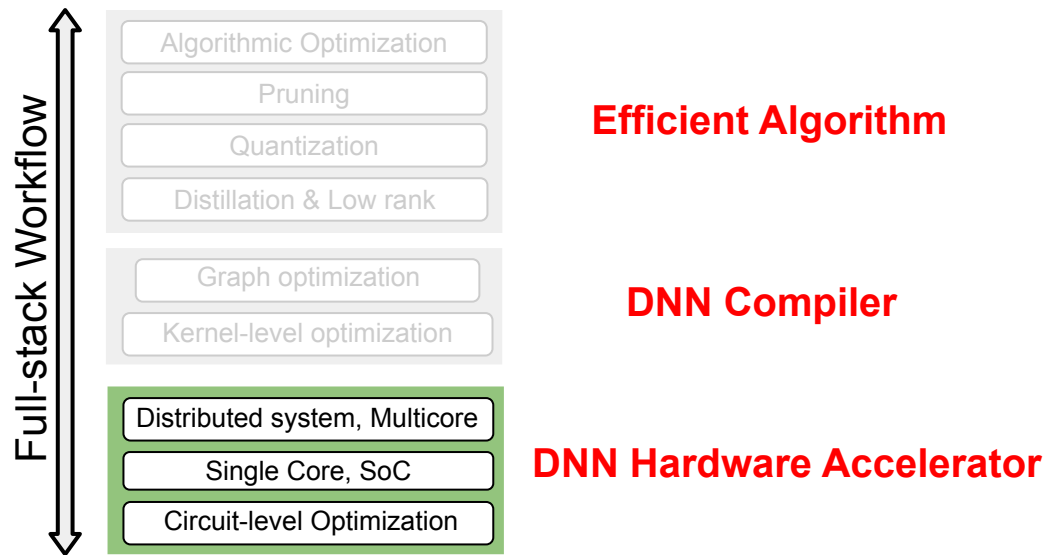
FP2INT is not cheap! But we can explore some system-level solution

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.



NVIDIA

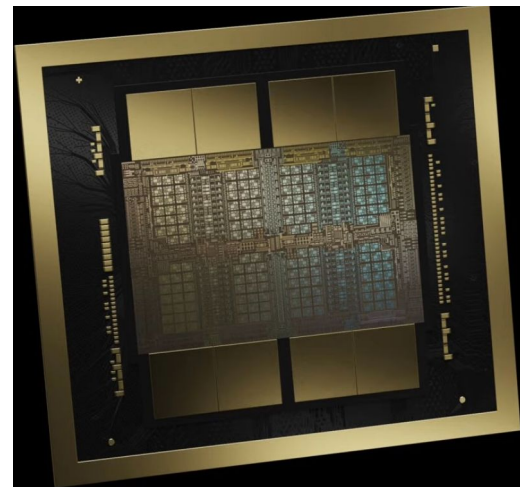


NVIDIA H100

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

NVIDIA

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



NVIDIA Blackwell

Hardware Support for DNN

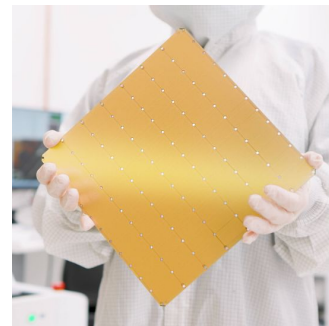
- ASIC-based implementations have been recently explored to accelerate the DNN inference.
 - Google's TPU, Apple's Neural Engine, Cerebras AI chip, ...
- FPGA-based accelerators for DNN inference have been recently developed.
 - Has good programmability and flexibility
 - Short development cycles
 -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



Tensor Processing Unit (Google)



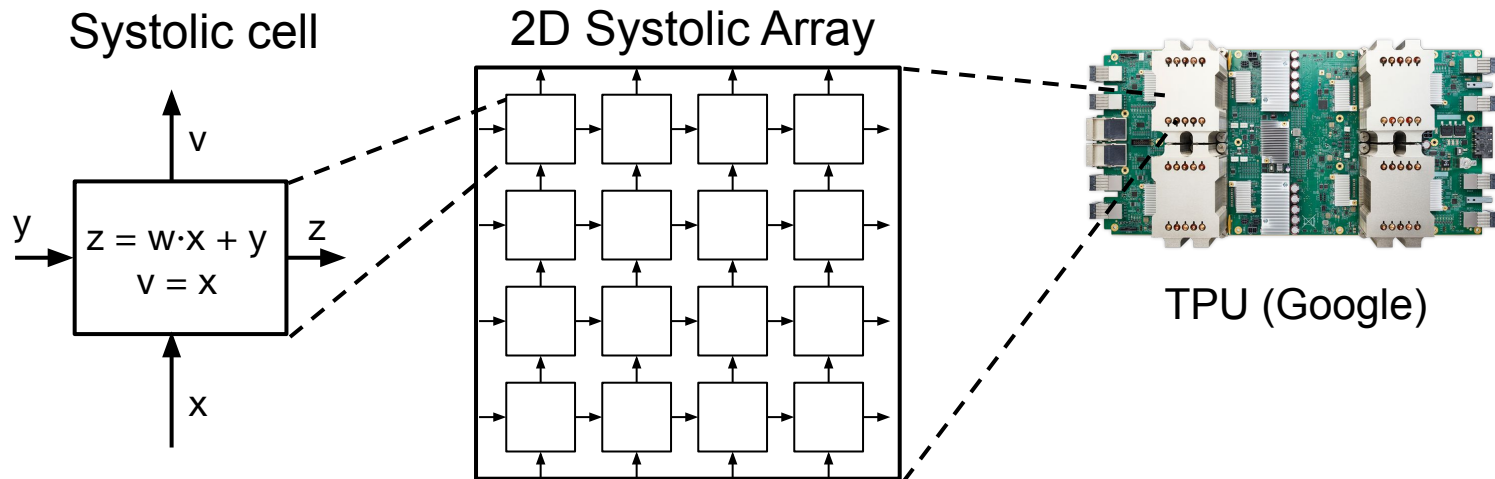
Alveo Accelerator Card (Xilinx)



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

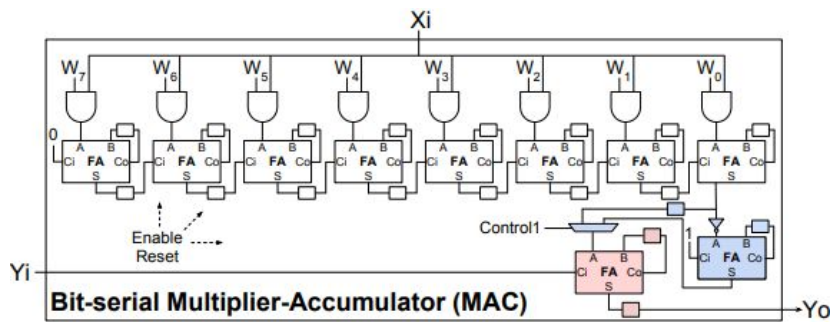
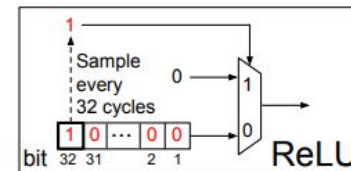
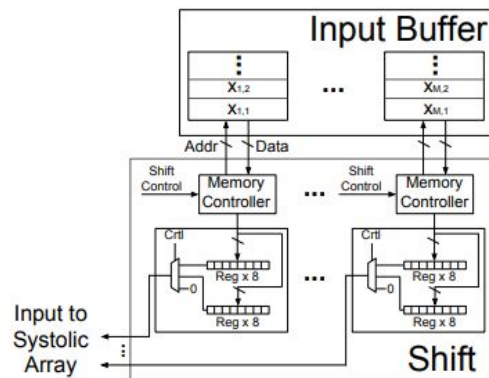
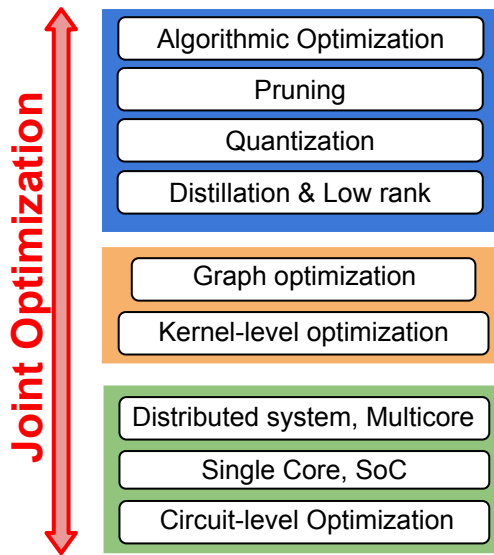


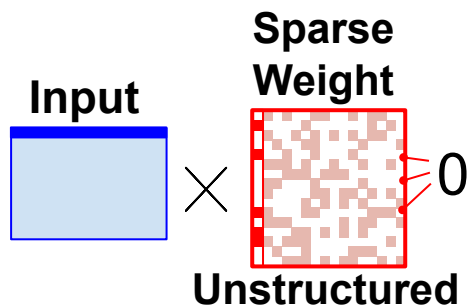
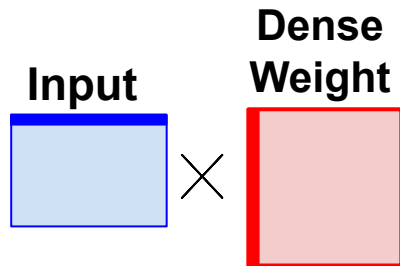
Figure 7: Bit-serial multiplier-accumulator (MAC).



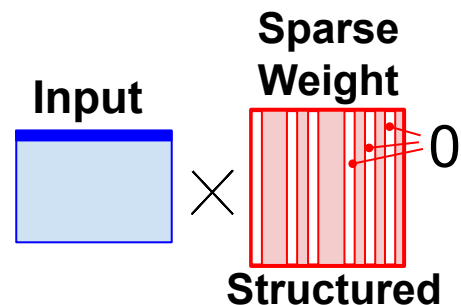
Why We Need Codesign?



Why We Need Codesign?



High accuracy
low hardware efficiency

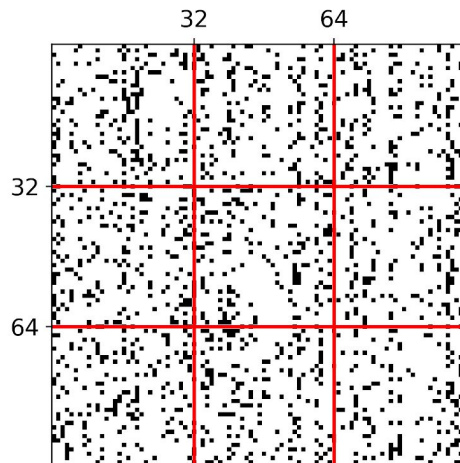


Low accuracy
high hardware efficiency

Hardware architecture needs to be considered when designing efficient DNN.

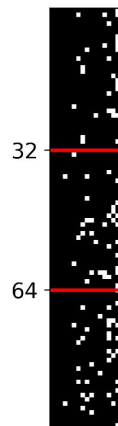
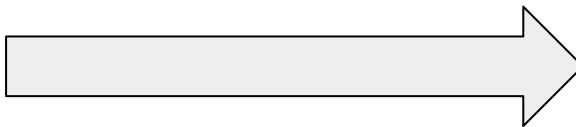
Column Combining

Sparse
Weight Matrix

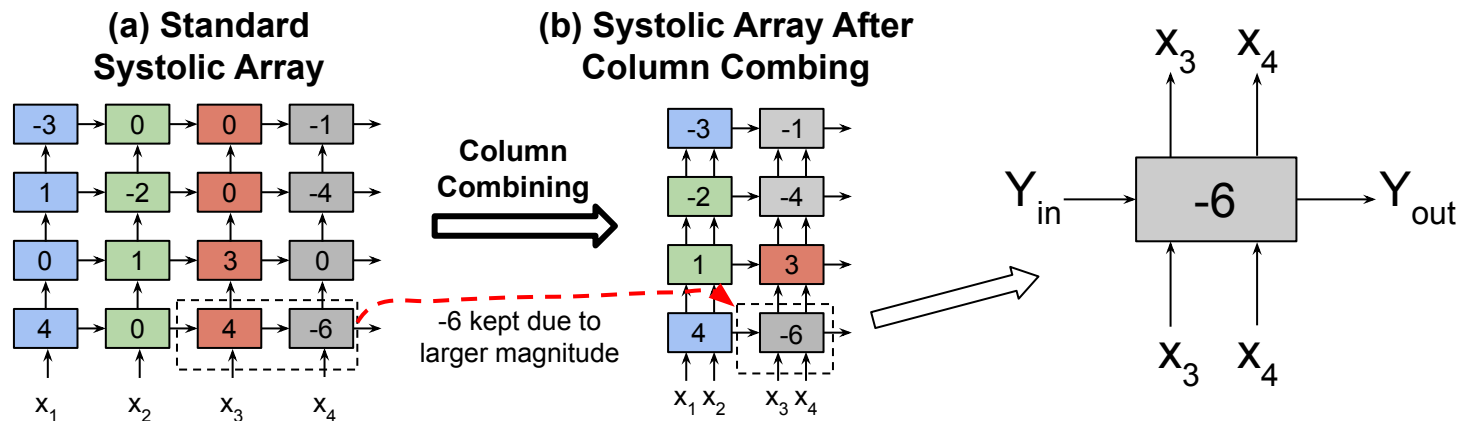


Packed Format in
Systolic Array

Column Combining
8x reduction in size

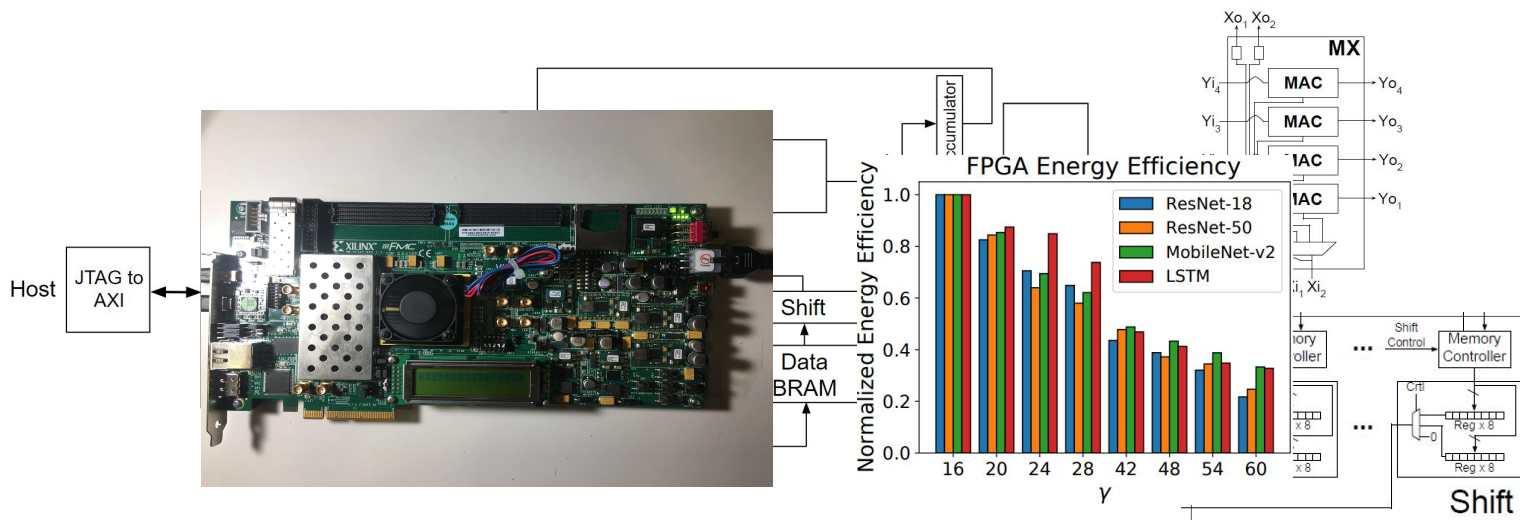


Column Combining

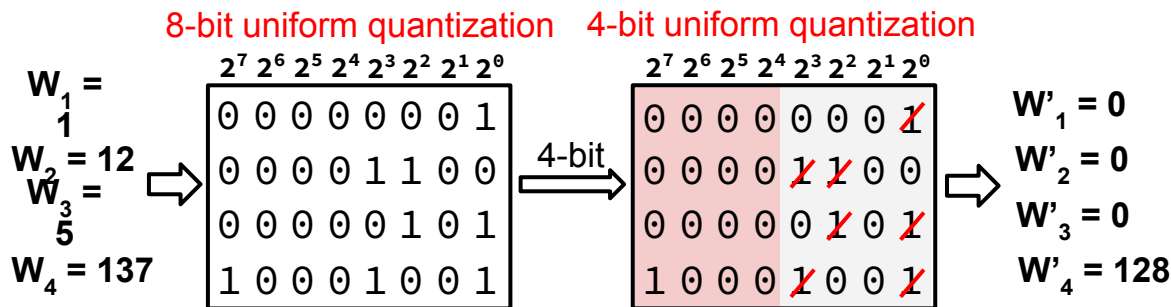


- Column combining can greatly increase the utilization efficiency of the systolic array
- Recently, Nvidia A100 GPU adopts a similar idea to support the balanced structured sparsity on their GPU

FPGA Accelerator

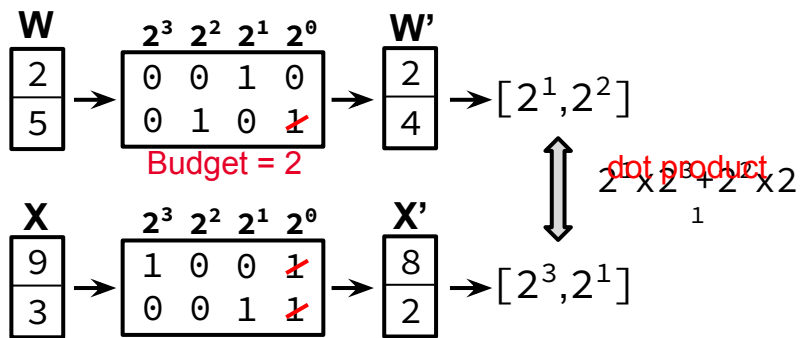


Term Quantization



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

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

$$W'_1 = 0$$

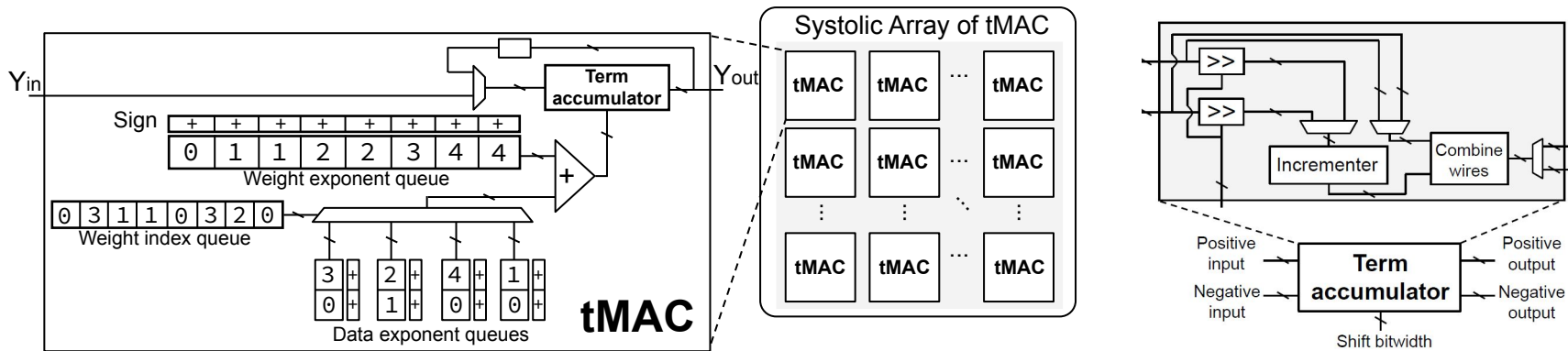
$$W'_2 = 12$$

$$W'_3 = 0$$

$$W'_4 = 136$$

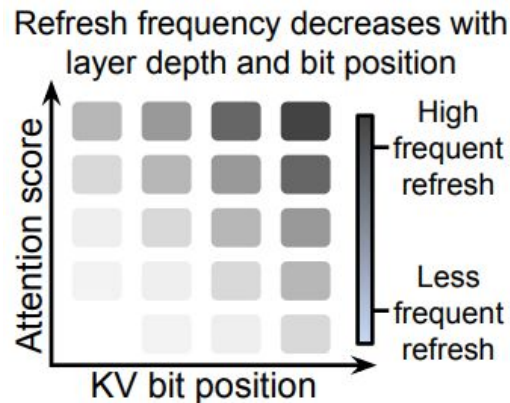
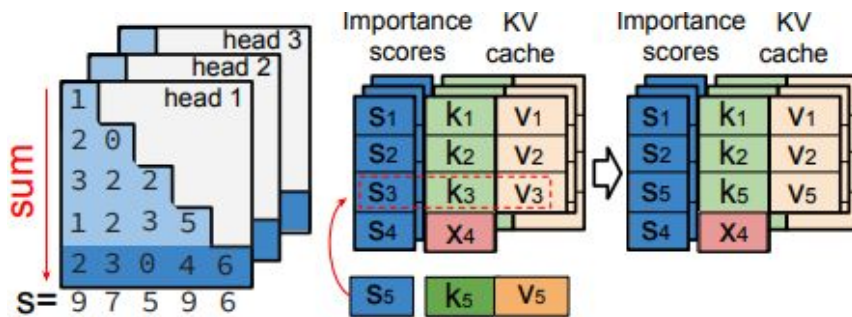
- We can control the term-level computations by setting a **group term budget**.
- For a group of values, we rank and remove the small terms based on this budget.

Term Quantization: Accelerator Design



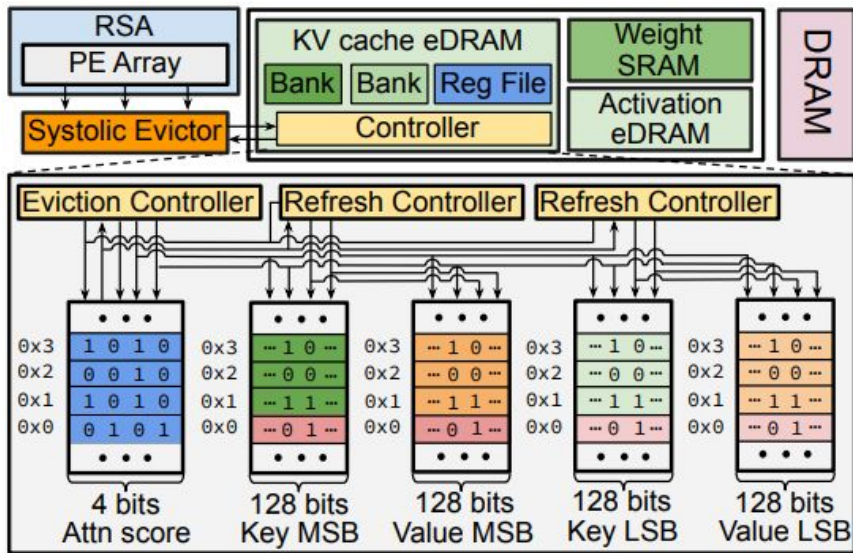
- We propose the term MAC (tMAC) for the efficient implementation of TQ.
- A tMAC processes all term-pair multiplications across a group of weight and data values.
- Each term is represented by their corresponding exponent (2-3 bits).
- The term accumulation can be implemented using half adders.

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.

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 $3.9\times$ speedup and $4.5\times$ energy savings compared to existing baseline solutions.

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: Intro to Transformer and Large Model

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: Machine Learning System for Large Model
- Lecture 11: AI Accelerator Introduction and CNN Accelerators
- Lecture 12: Transformer & LLM Accelerators
- Lecture 13: Guest Lecture



Lecture 1: Neural Network Basics

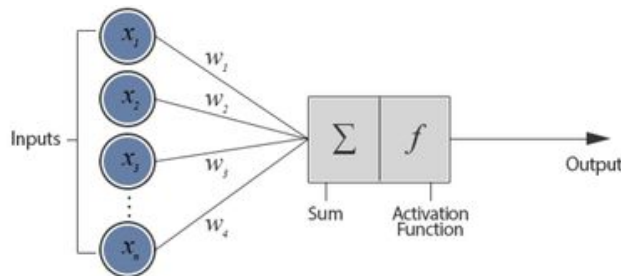
CSCI-GA 3033 / ECE-GY 9483
Efficient AI Computing: Algorithm and Implementation

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?

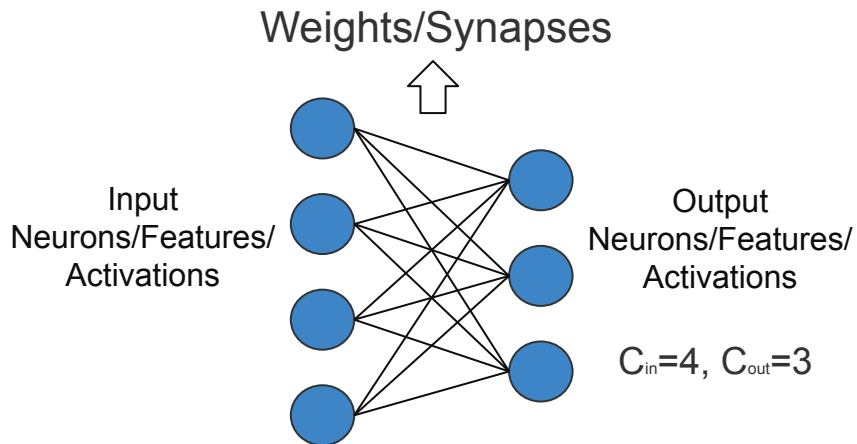
Multi-layer Perceptrons

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



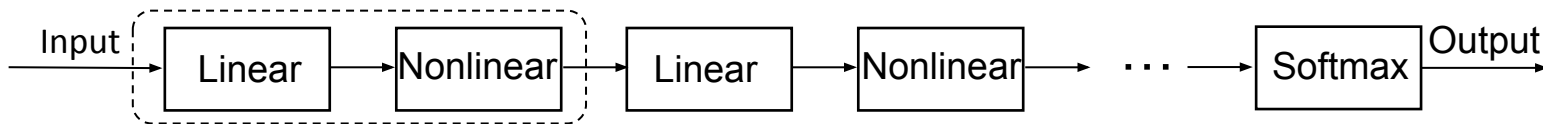
- A neural network consists of interconnected nodes, called neurons, organized into layers.
- Each neuron receives input signals (activations), performs a computation on them, and produces an output signal that may be passed to other neurons in the network.

Fully-connected layers (Linear layers)

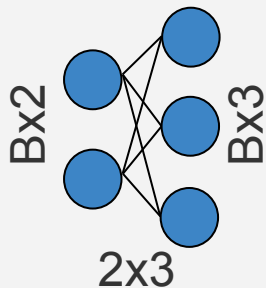
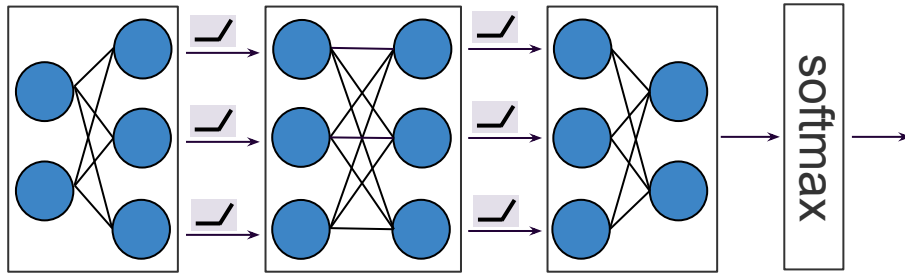
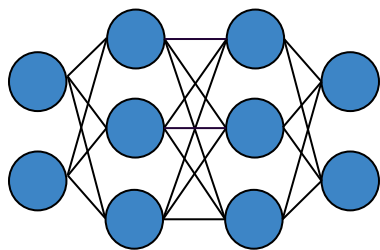


$$Y = XW + b$$

- 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

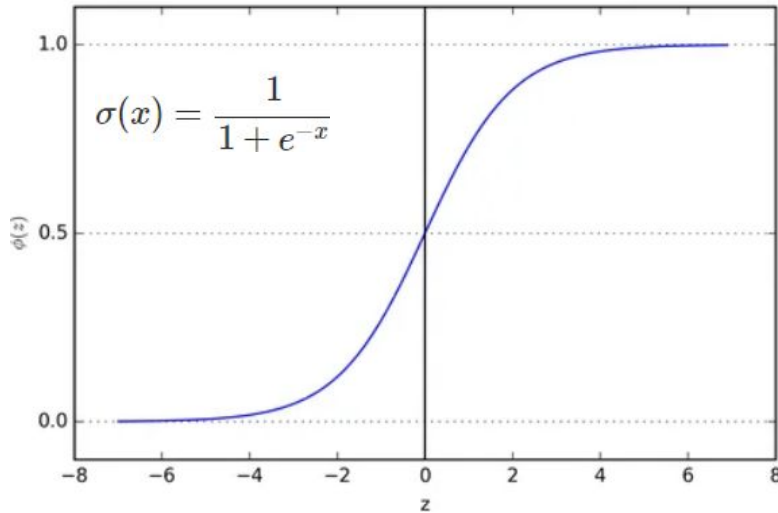


Computational Cost for MLP



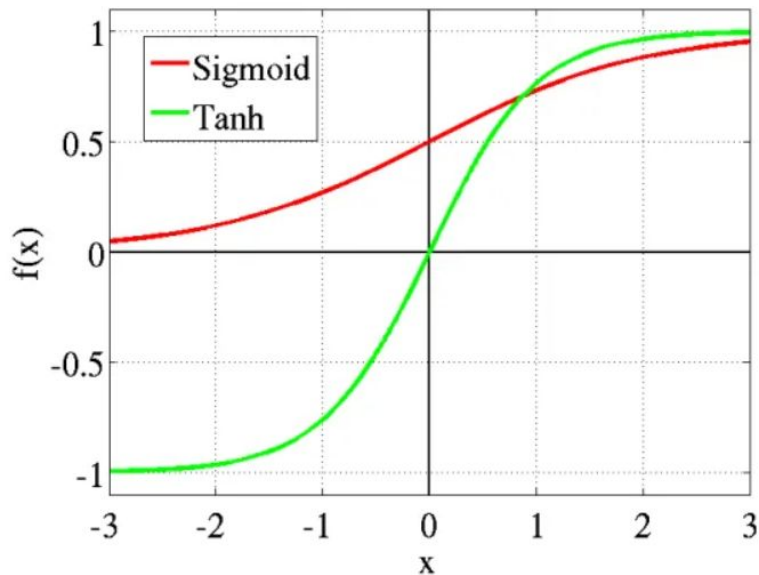
- Number of multiply-accumulate operations (MACs):
 - $B \times 2 \times 3 = 6B$
- Storage cost:
 - $6 \times 32 = 192$ bits (Weights)
 - $(2B + 3B) \times 32$ bits (Activation)

Sigmoid



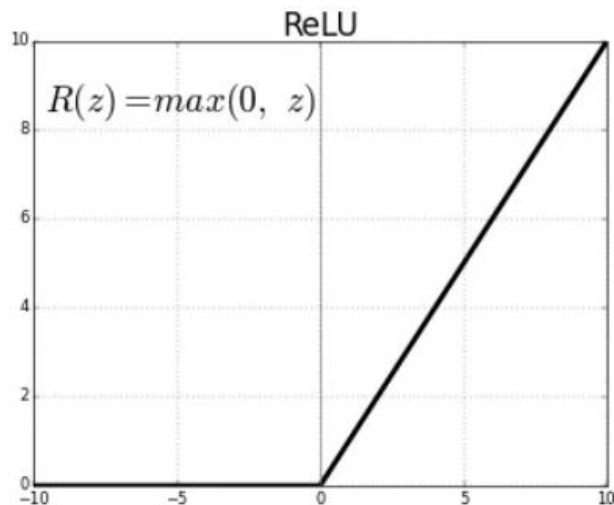
- Function: $\sigma(x) = \frac{1}{1 + e^{-x}}$
- Domain: $(-\infty, \infty)$
- Range: $[0, 1]$
- Differentiable everywhere
- Derivative: $\delta(x)(1 - \delta(x))$

Tanh



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

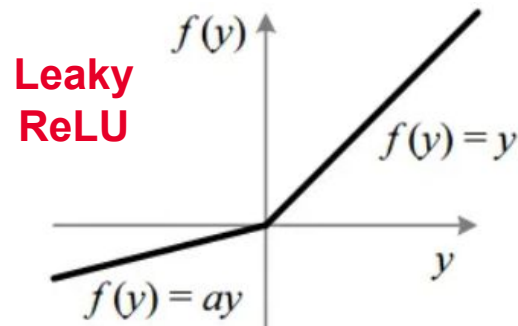
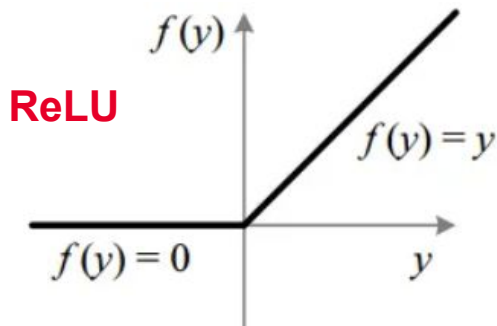
ReLU



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

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

Leaky ReLU



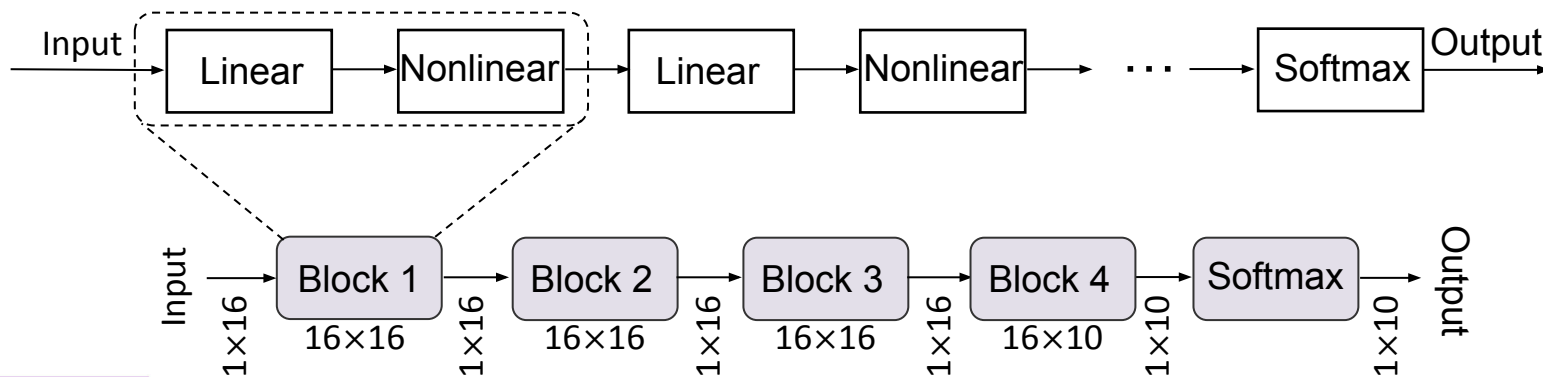
$$\text{Leaky_ReLU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ ax, & \text{otherwise} \end{cases}$$

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

Softmax

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

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



Loss Functions

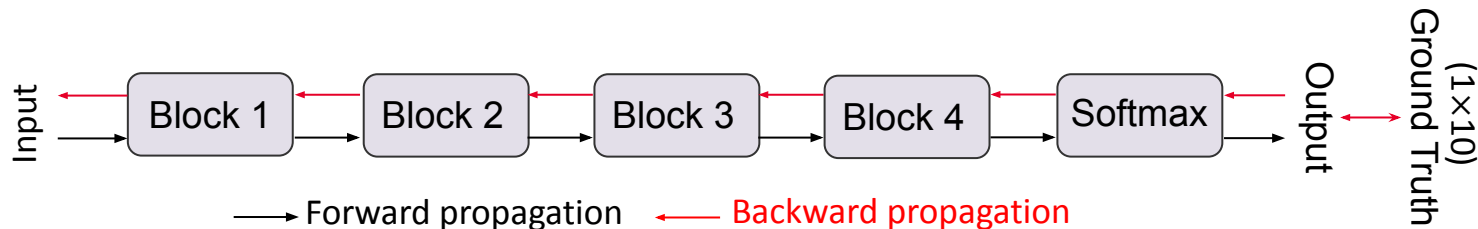
- Loss functions quantify the difference between the DNN output and the ground truth output in the training dataset.

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

L2 loss

$$\mathcal{L}_{\text{CE}} = - \sum_{c=1}^C y_c \log(p_c).$$

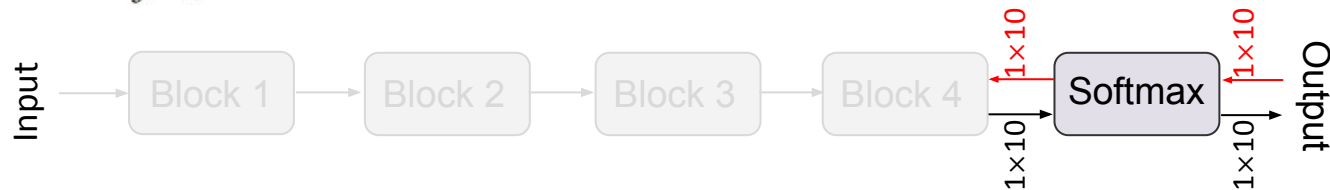
Cross-entropy loss



Softmax

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

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

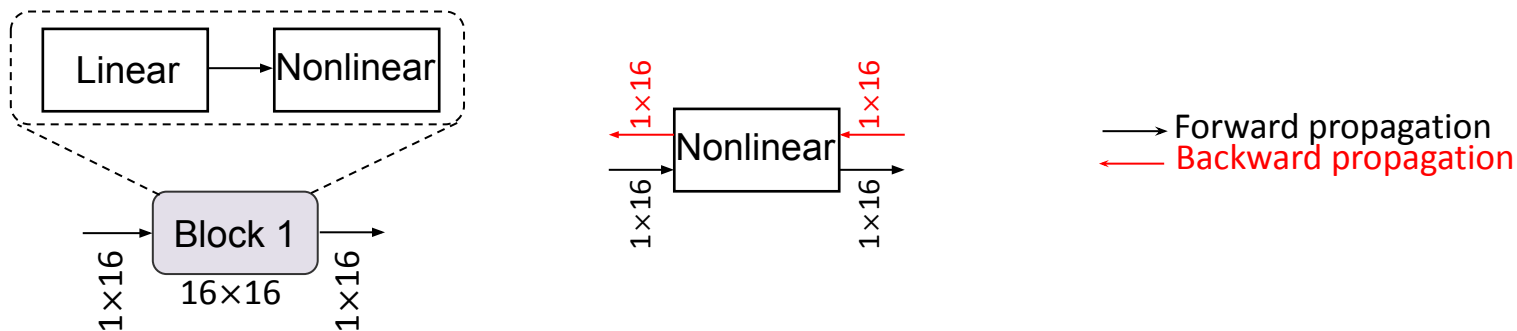


$$\frac{ds}{dz} = \text{diag}(s) - ss^T$$

When s has a dimension of 3 $\longrightarrow$

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

Backpropagation for Nonlinear Layers

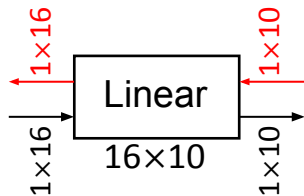
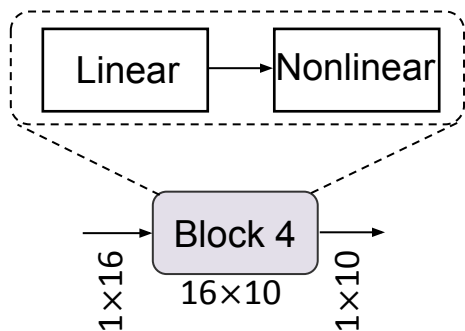


- Due to the elementwise nature, usually the nonlinear layer does not change the input and output shape during both forward and backward passes.

Backpropagation for Nonlinear Layers

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

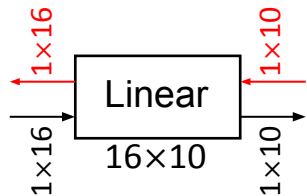
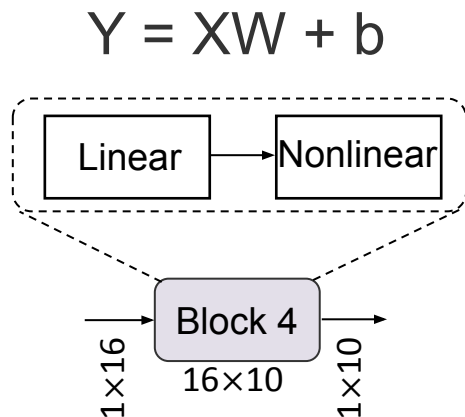
Backpropagation for Nonlinear Layers



→ Forward propagation
← Backward propagation

- 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)



$$\frac{dL}{dX} = \frac{dL}{dY} \frac{dY^\top}{dX} = \frac{dL}{dY} W^\top$$

Derivative wrt data

$$\frac{dL}{db} = \frac{dL}{dY}$$

Derivative wrt bias

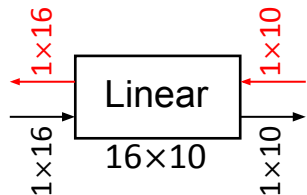
$$\frac{dL}{dW} = X^\top \frac{dL}{dY}$$

Derivative wrt weight

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): $1 \times C_{out}$



$$\frac{dL}{dX} = \frac{dL}{dY} \frac{dY^\top}{dX} = \frac{dL}{dY} W^\top$$

Derivative wrt data

$$\frac{dL}{db} = \frac{dL}{dY}$$

Derivative wrt bias

$$\frac{dL}{dW} = X^\top \frac{dL}{dY}$$

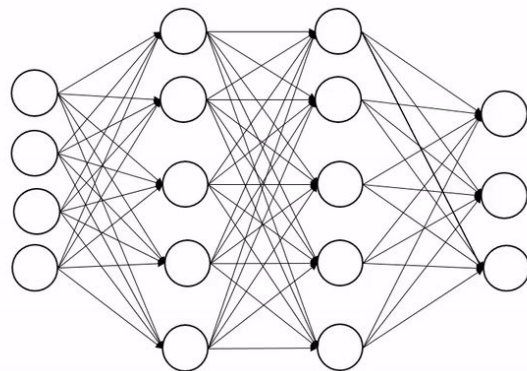
Derivative wrt weight

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.

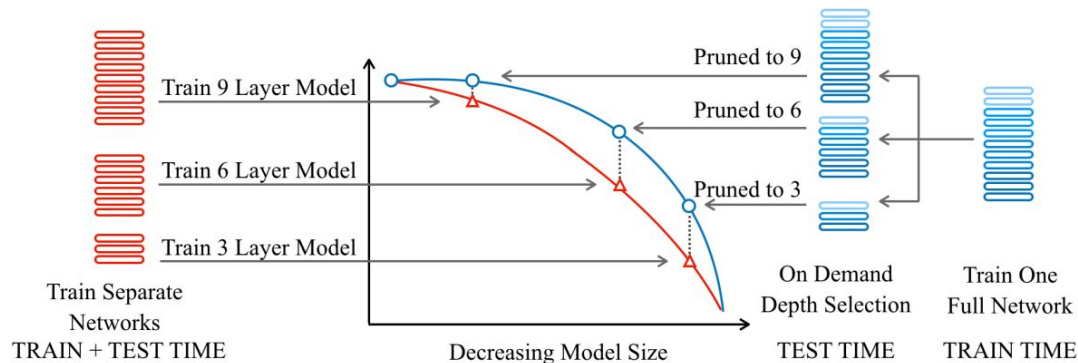
$$L = L + \lambda ||W||^2$$

- 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.



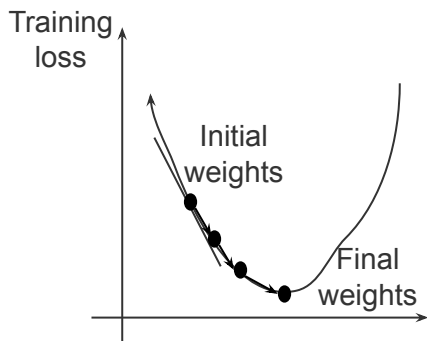
Layer Dropout

- 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.



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: $\text{total_training_data_size} / \text{Batch}$
 - 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 dataset:

Perform the forward propagation

Compute the loss and weight gradient via backward propagation.

Update the weights

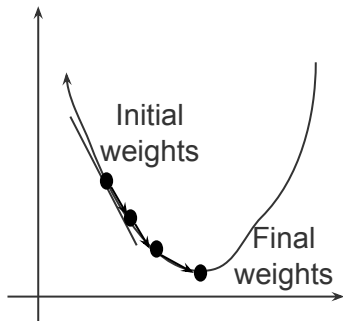
Update the learning rate (if necessary)

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: $\text{total_trainingdata_size}/\text{Batch}$
 - 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

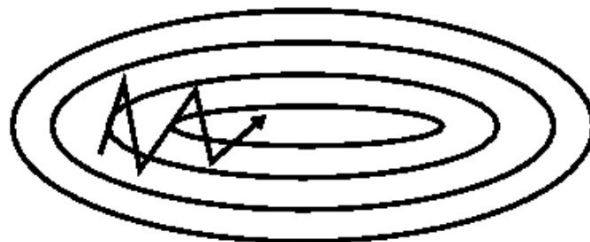
Update the learning rate (if necessary)

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.
- $W' = W - \eta g_t$ $g_t \rightarrow s g_{t-1} + (1-s) dL/dW$, s is a hyperparameter between 0 and 1, close to 1.



SGD without momentum



SGD with momentum

RMSProp

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

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

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

1.6 will be scaled down with RMSProp

- 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.

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)

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)

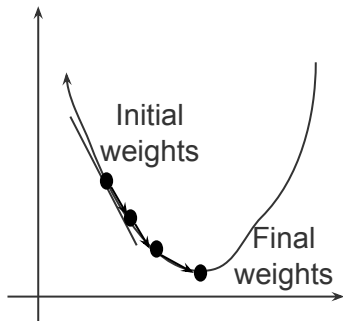
end while

return θ_t (Resulting parameters)

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

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: $\text{total_trainingdata_size}/\text{Batch}$
 - 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 dataset:

Perform the forward propagation

Compute the loss and weight gradient via backward propagation.

Update the weights

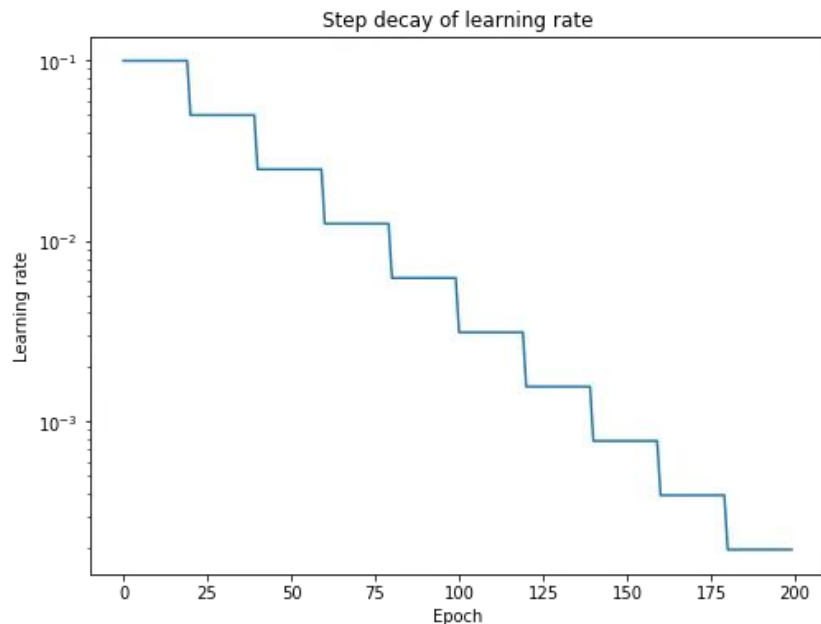
Update the learning rate (if necessary)

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.

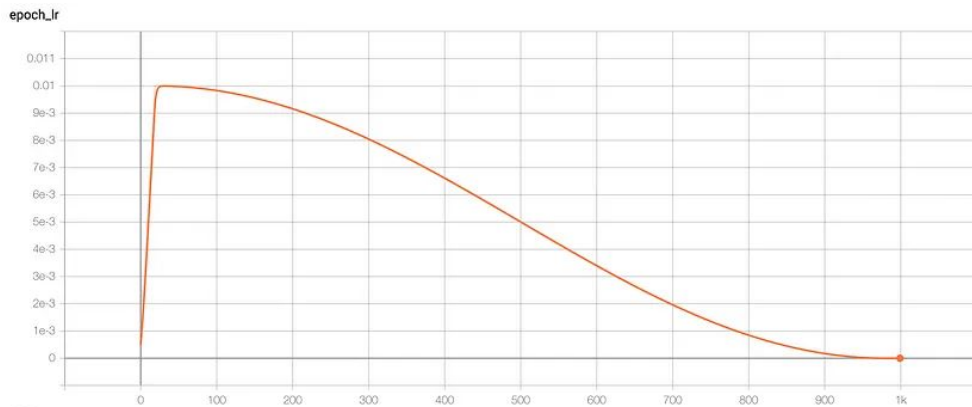
$$W' = W - \eta g_t$$

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.

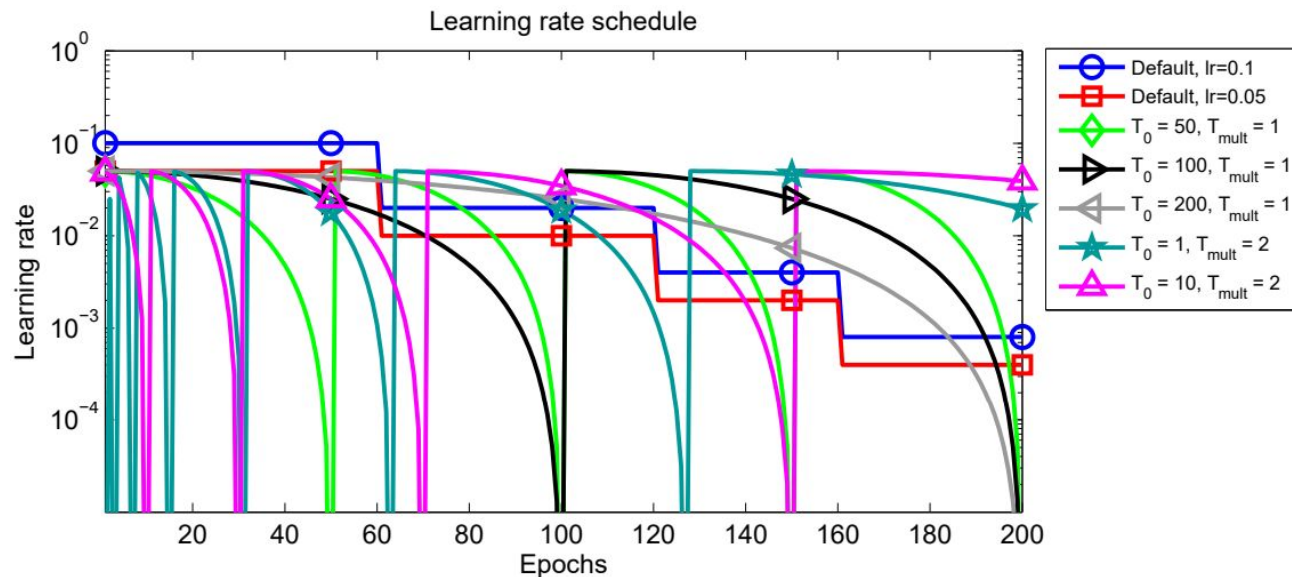
Cosine Learning Rate



- 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.

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

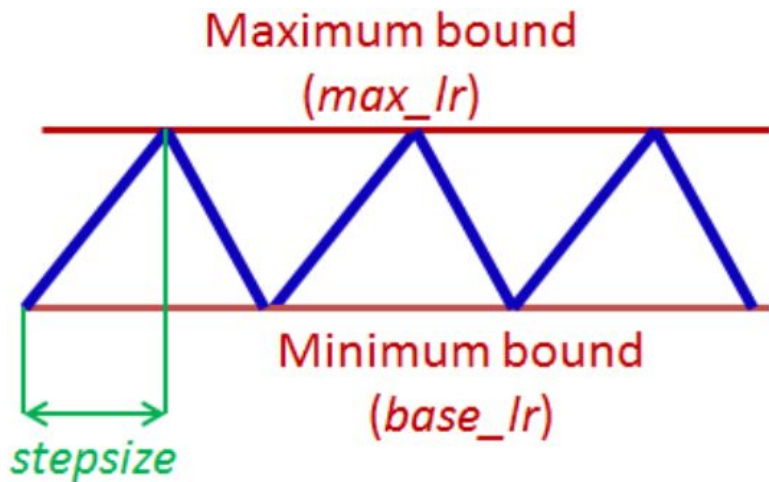
Cosine Learning Rate



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

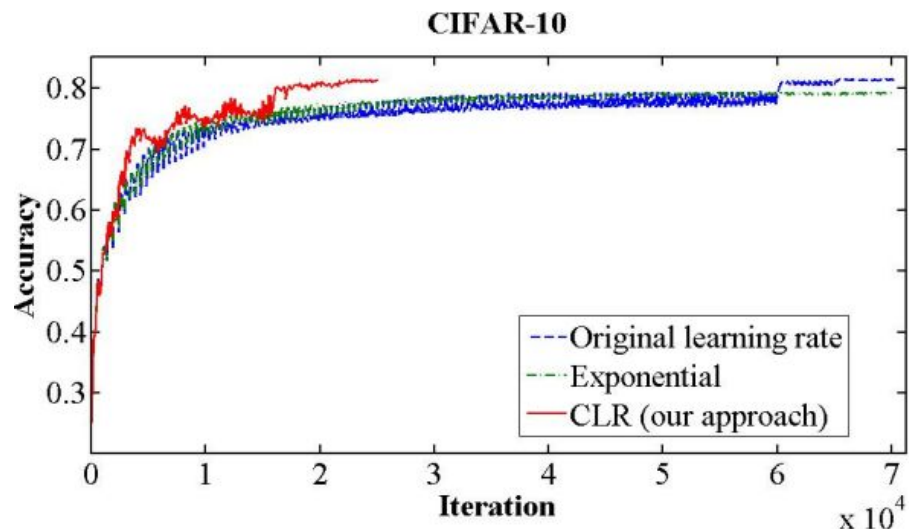
$$\eta_t = \eta_{min}^i + \frac{1}{2}(\eta_{max}^i - \eta_{min}^i)(1 + \cos(\frac{T_{cur}}{T_i}\pi)),$$

Cyclical Learning Rate



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

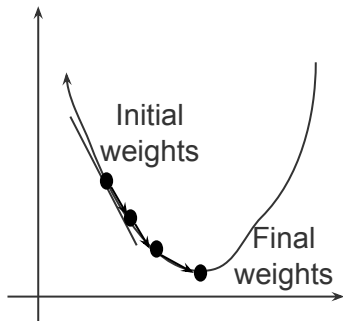
Cyclical Learning Rate



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

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: $\text{total_trainingdata_size}/\text{Batch}$
 - 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

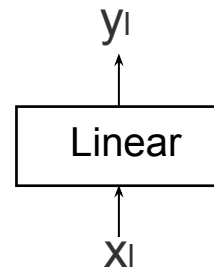
Update the learning rate (if necessary)

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.



Derivation

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

Assume $\mathbf{W}_l$ has a shape of m_l by n_l , and $\mathbf{x}_l$ has a size of $n_l \times 1$, then $\mathbf{y}_l$ has a size of $m_l \times 1$.

For each element $y_{l,i}$ of $\mathbf{y}_l$, 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 $x_{l,i} = \text{ReLU}(y_{l-1,j})$

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

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

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

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

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

Given this, we have:

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

Derivation

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

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

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