Introduction to Machine Learning

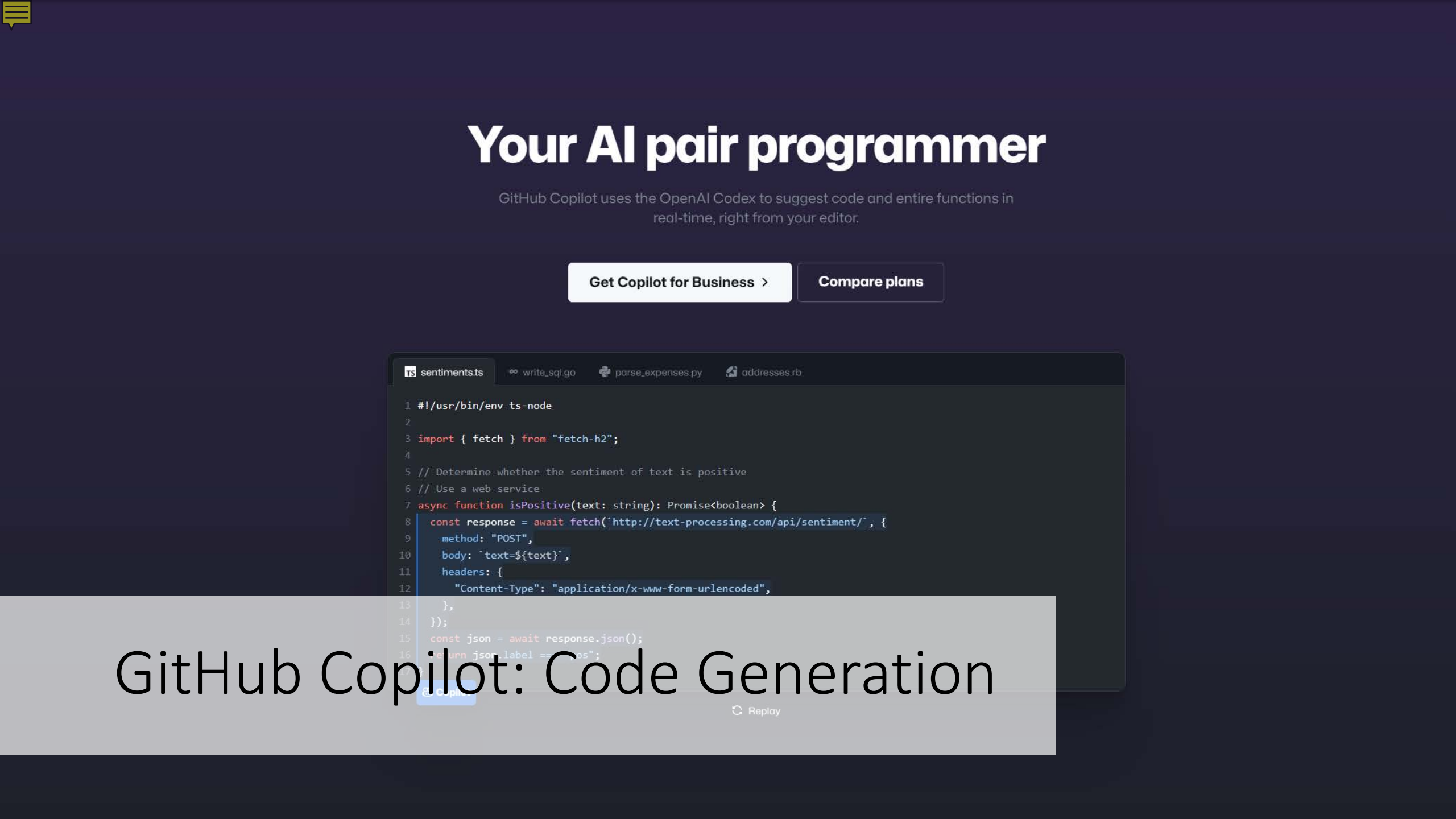
Jinjun Huang@ ZJUSCT

2024.7.11

TOC

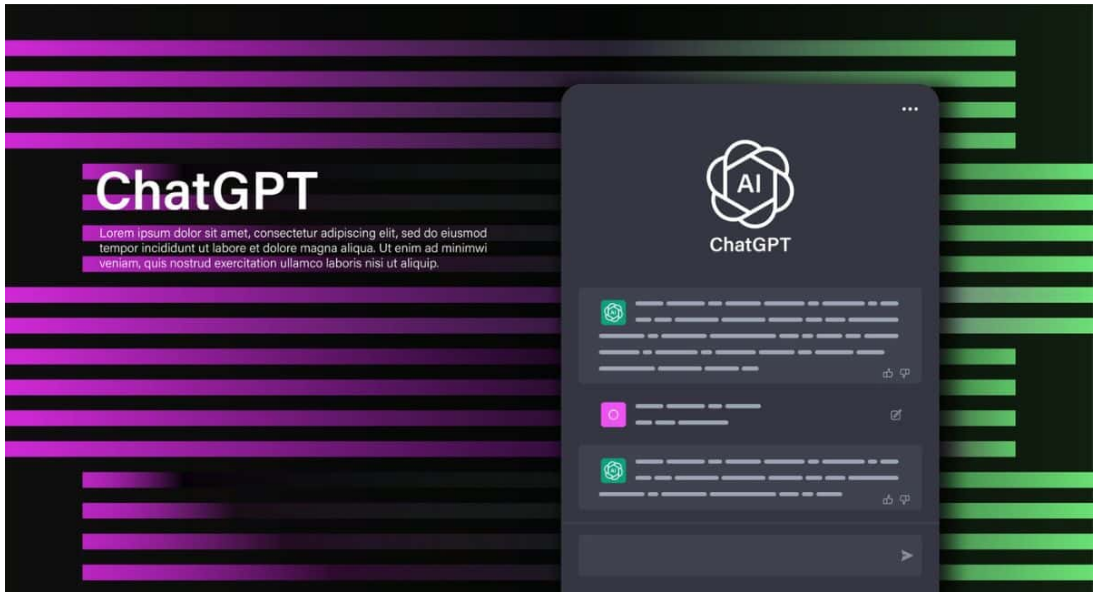
- Introduction
- Basic of Machine Learning
- Typical Network Architectures
  - CNN
  - RNN
  - Attention
- DL Hardware & Software
- MLSys

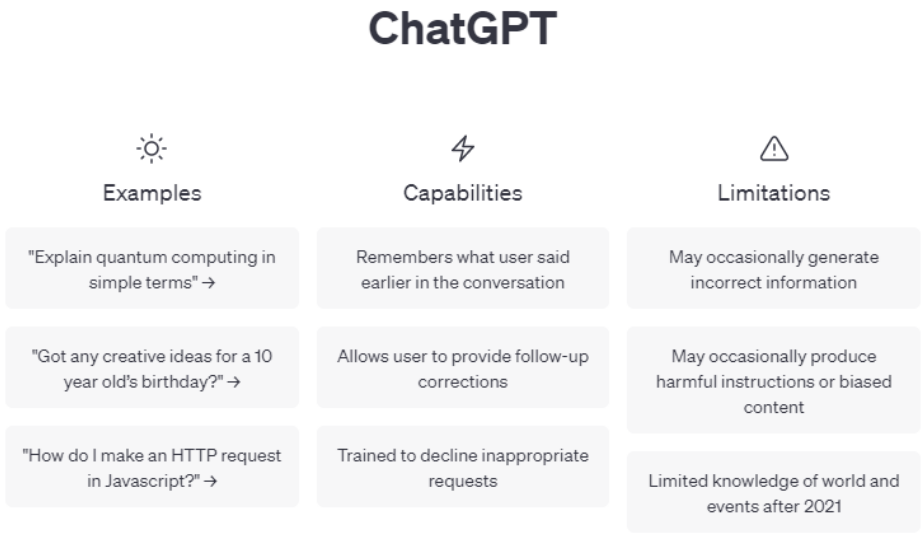
What is machine learning?



GitHub Copilot: Code Generation

GPT-3.5 & GPT-4: ChatGPT

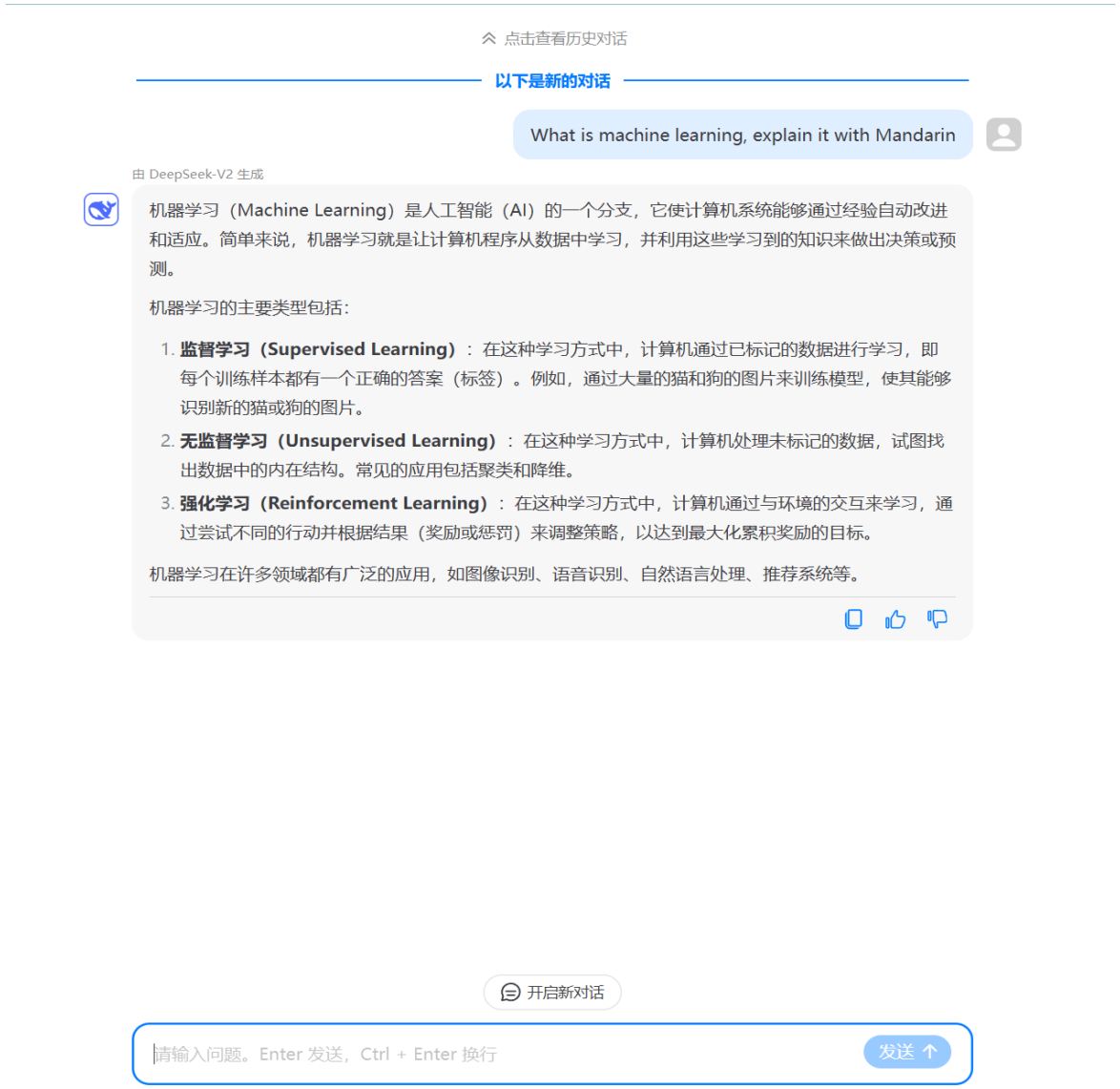




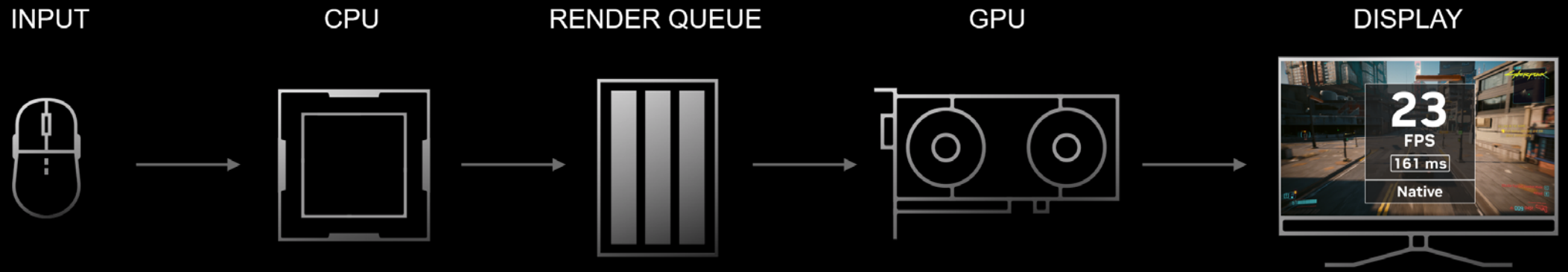
通义千问


通义
对话
效率
智能体
新建对话
通义千问 2.5
你好，我是通义
通情，达义。你的全能AI助手
换一批
AI搜索
通义千问2.5有哪些升级
最新Qwen 2成全球开源大模型第一名
2024世界人工智能大会
嫦娥六号取回了多少月壤
绝区零预下载多个国家和地区登顶
扩写助手
扩写文本、丰富内容：今天天气不错
AI 健身教练
请帮我制定详细的健身计划，我的目标是：...
AI市场分析师
你是专业市场分析师，请给出该行业的市场...
登录后可使用完整功能
登录
输入"/"唤起指令中心，Shift+Enter换行，支持拖拽或粘贴上传文件

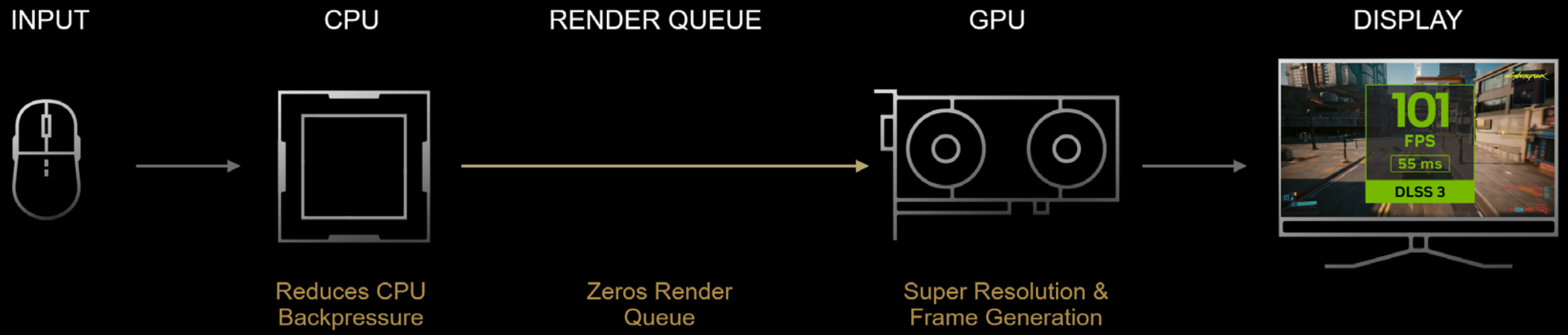
DeepSeek


点击查看历史对话
以下是新的对话
What is machine learning, explain it with Mandarin
由 DeepSeek-V2 生成
机器学习（Machine Learning）是人工智能（AI）的一个分支，它使计算机系统能够通过经验自动改进和适应。简单来说，机器学习就是让计算机程序从数据中学习，并利用这些学习到的知识来做出决策或预测。
机器学习的主要类型包括：
1. 监督学习（Supervised Learning）：在这种学习方式中，计算机通过已标记的数据进行学习，即每个训练样本都有一个正确的答案（标签）。例如，通过大量的猫和狗的图片来训练模型，使其能够识别新的猫或狗的图片。
2. 无监督学习（Unsupervised Learning）：在这种学习方式中，计算机处理未标记的数据，试图找出数据中的内在结构。常见的应用包括聚类和降维。
3. 强化学习（Reinforcement Learning）：在这种学习方式中，计算机通过与环境的交互来学习，通过尝试不同的行动并根据结果（奖励或惩罚）来调整策略，以达到最大化累积奖励的目标。
机器学习在许多领域都有广泛的应用，如图像识别、语音识别、自然语言处理、推荐系统等。
开启新对话
请输入问题。Enter 发送，Ctrl + Enter 换行
发送

SYSTEM LATENCY PIPELINE



DLSS 3 WITH NVIDIA REFLEX BOOSTS PERFORMANCE & RESPONSIVENESS



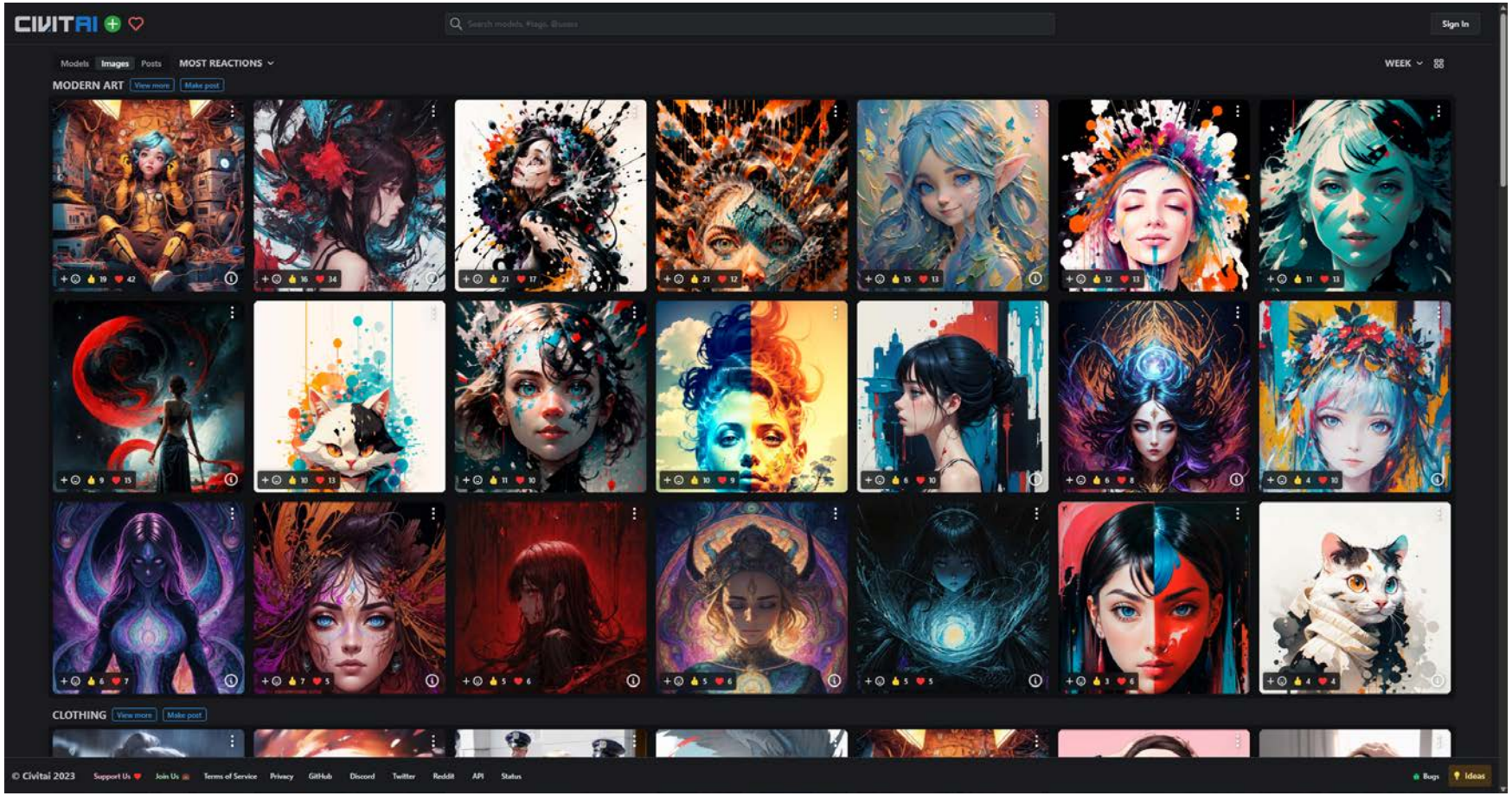
AI Art

- DALL·E 3

- Midjourney

AI Art

- Stable Diffusion + LoRA



Traditional programming (software 1.0)

(does not require 2.0)

vs.

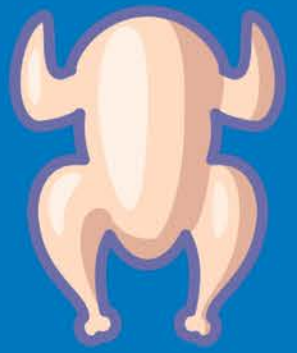
(requires 1.0)

Machine learning (software 2.0)

Ref: https://github.com/mrdbourke/machine-learning-roadmap

Andrei Karpathy post: Software 2.0

Traditional programming

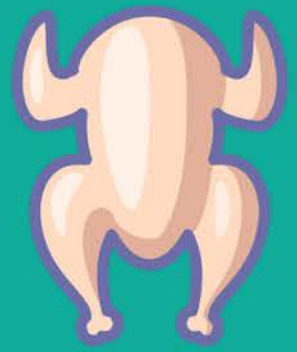


1. Cut vegetables
2. Season chicken
3. Preheat oven
4. Cook chicken for 30-minutes
5. Add vegetables

Starts with → **Makes**

Machine learning algorithm

Inputs



Output

1. Cut vegetables
2. Season chicken
3. Preheat oven
4. Cook chicken for 30-minutes
5. Add vegetables

Starts with → **Figures out**

Ref: https://github.com/mrdbourke/machine-learning-roadmap

Machine learning is turning things (data) into numbers and finding patterns in those numbers.

The computer does this part.

How?

Math.

(we'll cover a little on this later)

Ref: https://github.com/mrdbourke/machine-learning-roadmap

“Why use machine learning?”

Another curious (perhaps even more curious than before) internet dweller

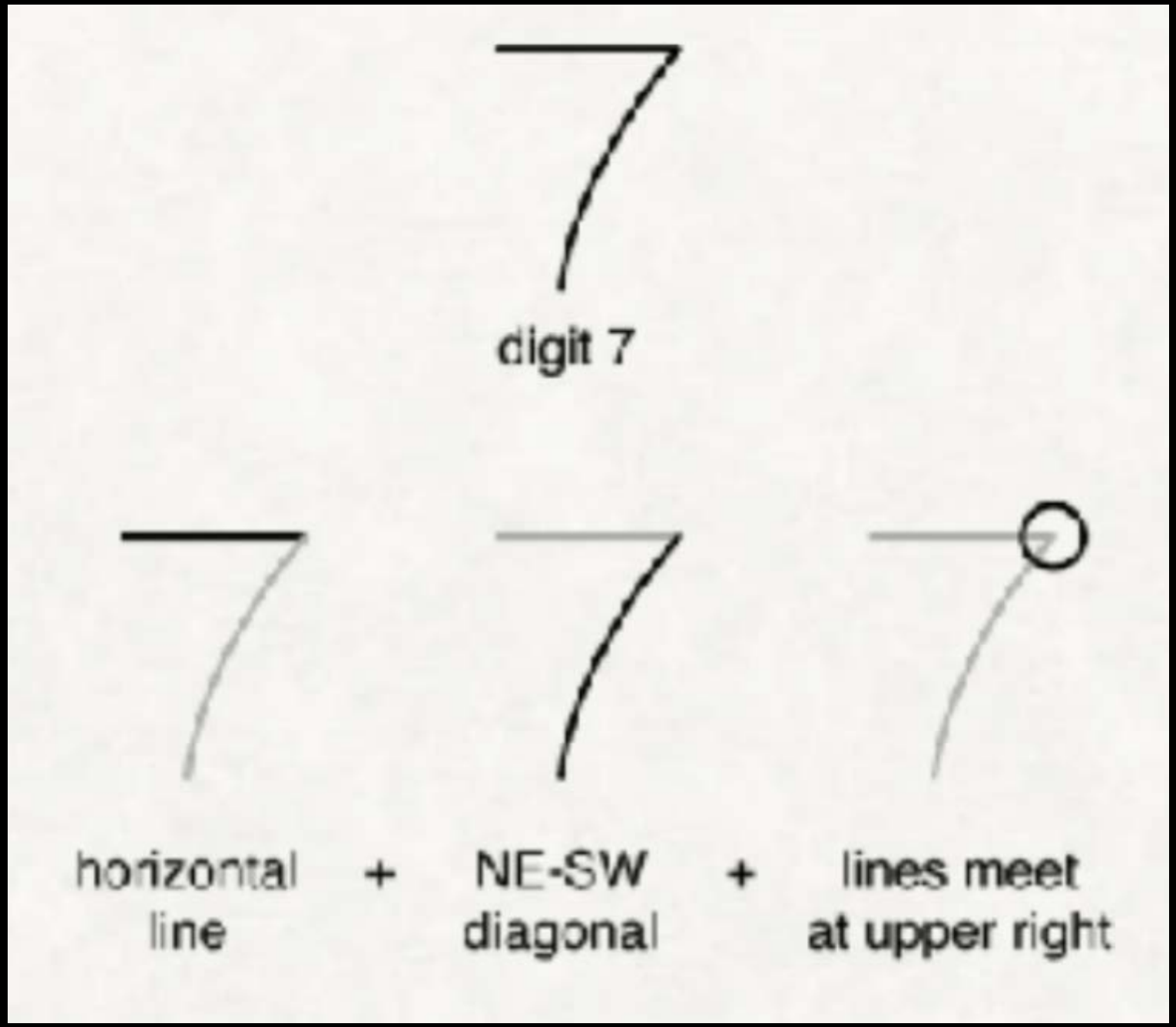
Ref: https://github.com/mrdbourke/machine-learning-roadmap

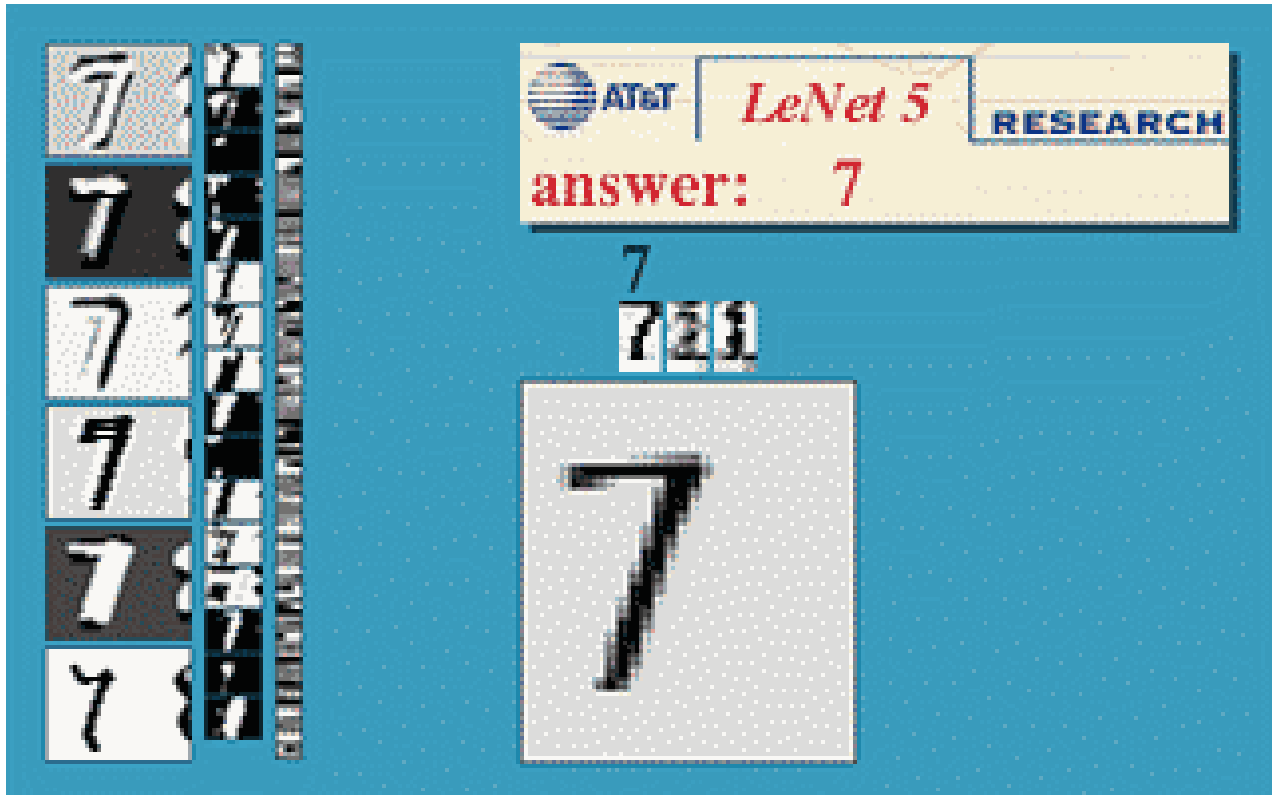
Good reason: ~~Why not?~~

Better reason: Can you think of all the rules?

(probably not)

Ref: https://github.com/mrdbourke/machine-learning-roadmap


digit 7
horizontal line
+
NE-SW diagonal
+
lines meet at upper right


AT&T
LeNet 5
RESEARCH
answer: 7
7
Layer–1
Layer–3
Layer–5
Input

Traditional Approach vs Machine Learning

Traditional

- Hand-crafted rules / features
- Explainable
- Less tested on data
- Insufficient capacity

Machine Learning

- Automatically-learned rules / features
- Less explainable
- Data-driven
- High capacity

Focus on formulating the problem, and then derive the solution more automatically

(maybe not very simple...)

"If you can build a **simple rule-based** system that doesn't require machine learning, do that."

A wise software engineer... (actually rule 1 of Google's Machine Learning Handbook)

Ref: https://github.com/mrdbourke/machine-learning-roadmap

What machine learning is good for 

- **Problems with long lists of rules**—when the traditional approach fails, machine learning may help.
- **Continually changing environments**—machine learning can adapt ('learn') to new scenarios.
- **Discovering insights within large collections of data**—can you imagine trying to go through every transaction your (large) company has ever had by hand?

Ref: https://github.com/mrdbourke/machine-learning-roadmap

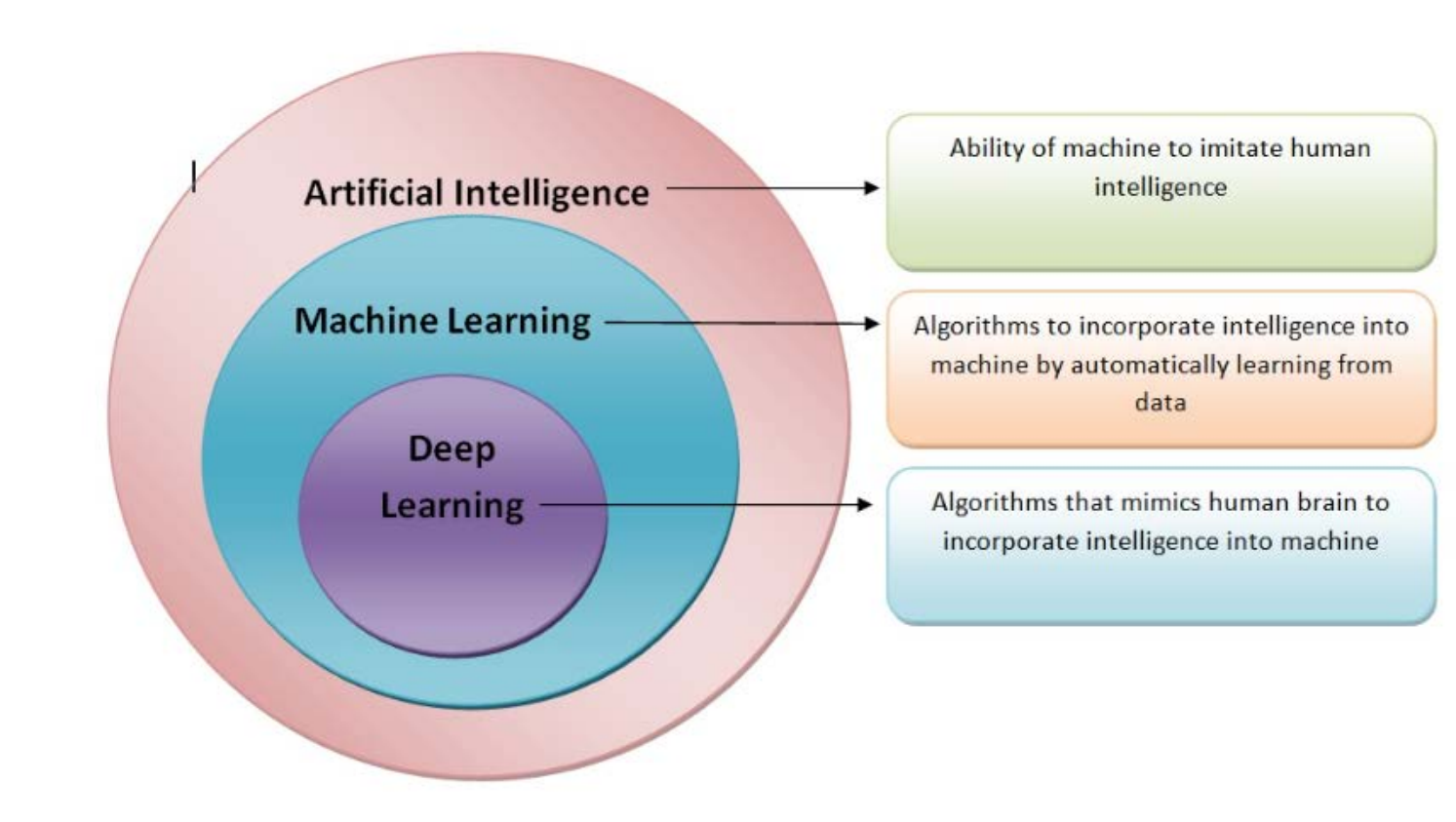
Why ML in an HPC course?

- ML (especially DL) is a special serious of application requiring large computing power
  - Just like other domain-specific applications
  - Can use techniques in traditional optimization
- ML can be used to guide system optimization
- Emerging research topic: MLSys
  - Distributed Learning
  - Inference Acceleration
  - Domain-Specific Language & Architecture
  - AutoSys
  - …

What does this lesson focusing on?

- (Very) Basic ML & DL (What to do)
  - Model, Loss and Optimizer
  - CNN (CV)
  - Attention (NLP)
- MLSys
  - Framework (e.g. PyTorch)
  - Training/Inferencing Systems
  - LLM
- Doesn’t
  - Modern Deep Learning Technology (Many complex networks)
  - Theory (Why do this? Why it works?)
  - Application (How to use/depoly ChatGPT/Stable Diffusion?)

What is the difference between ML/DL/AI?



Stay with the Times

课程号	课程名称	学分	周学时	总学时	建议学年学期
CS1006G	Python程序设计	3.0	2.0-2.0	64	一(秋冬)
CS1241G	人工智能基础（A）	2.0	2.0-0.0	32	一(春夏)

课程号	课程名称	学分	周学时	建议学年学期
211G0290	计算机科学基础（A）	2.0	2.0-0.0	一(秋冬)
211G0200	Python程序设计	3.0	2.0-2.0	一(春夏)

Does it require math?

Basic of machine learning

1. 🤔 Machine Learning Problems

Ref: https://github.com/mrdbourke/machine-learning-roadmap

1. 🤔 Machine Learning Problems

(categories of learning)

Supervised Learning

Unsupervised Learning

Transfer Learning

Reinforcement Learning

Ref: https://github.com/mrdbourke/machine-learning-roadmap

1. 🤔 Machine Learning Problems

(problem domains)

Classification

Regression

Clustering

(dropped) (most important)

ID	Weight	Heartrate	Age	Heart Disease?
0	76	54	55	0
1	81	42	34	0
2	90	70	47	0
3	67	100	79	1

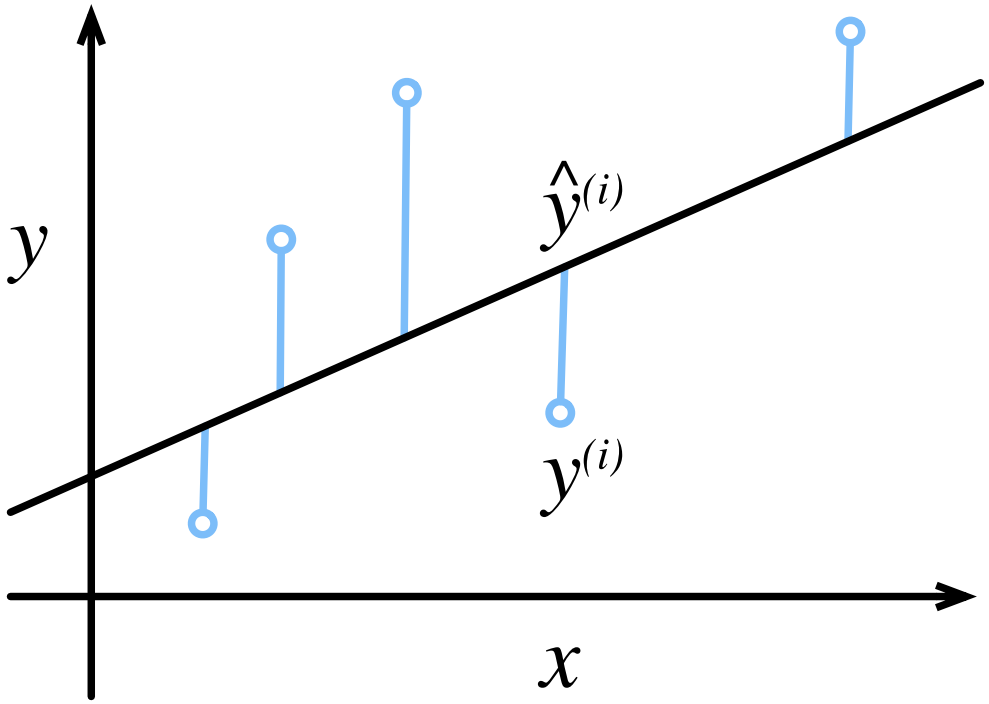
Dimensionality reduction

Ref: https://github.com/mrdbourke/machine-learning-roadmap

Formulation of machine learning

- $y = f(x)$
  - x: input numeric data
  - y: expected result
- What to use for f?
  - E.g. $y = wx + b$
- How to solve params in f?
  - E.g. w & b
- How to get x from real-world data?
- How to judge the quality of f?

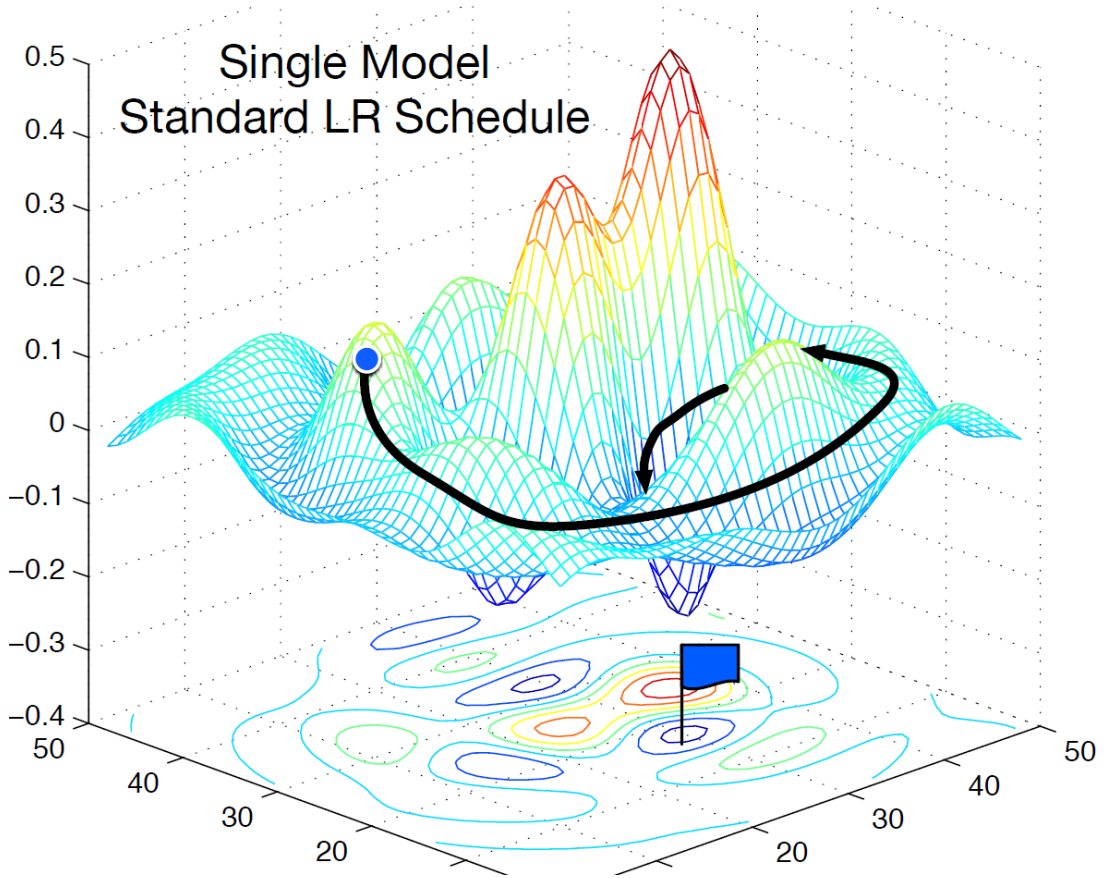
Easiest f : Linear Regression



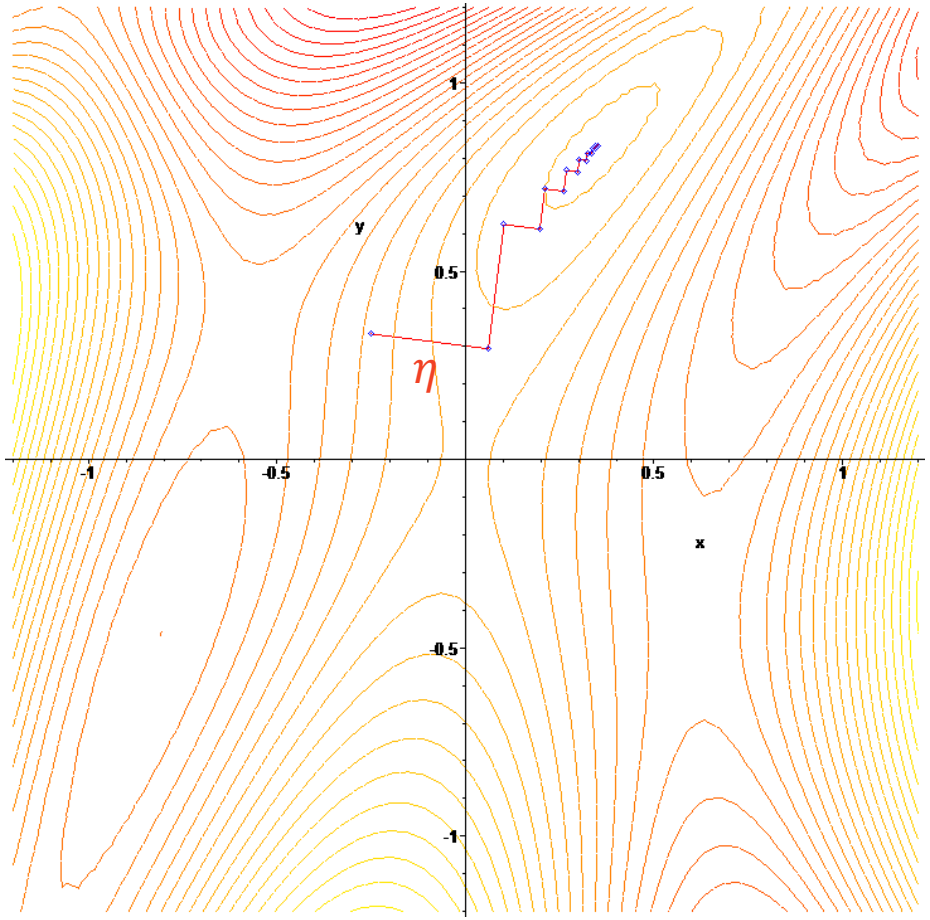
- 1-D: $\hat{y} = wx + b$
- N-D: $\hat{y} = \mathbf{w}^\top \mathbf{x} + b$
- Analytic solution: least square regression
- Fitting (training): a data-driven learning process
- Cost function (loss): $l^{(i)}(\mathbf{w}, b) = \frac{1}{2}\left(\hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)}\right)^2$

How to solve params in f: Gradient Descent

- Gradient: the direction in which the function increases most quickly.



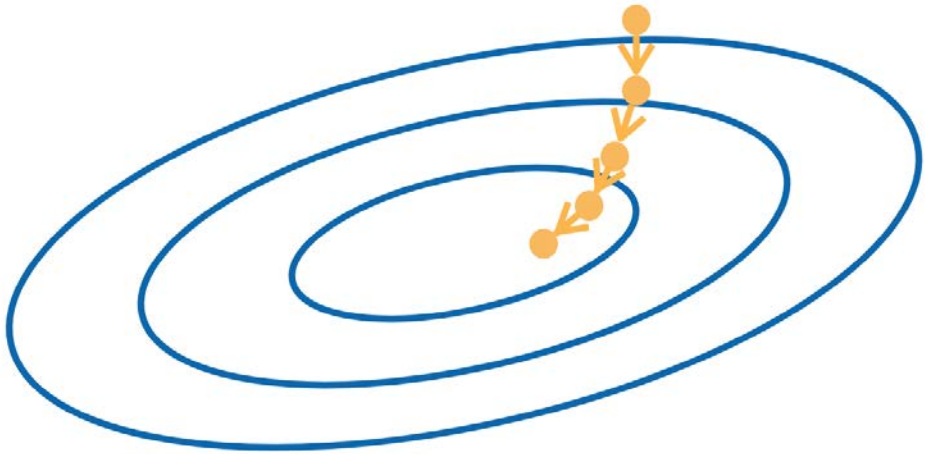
Gradient Descent



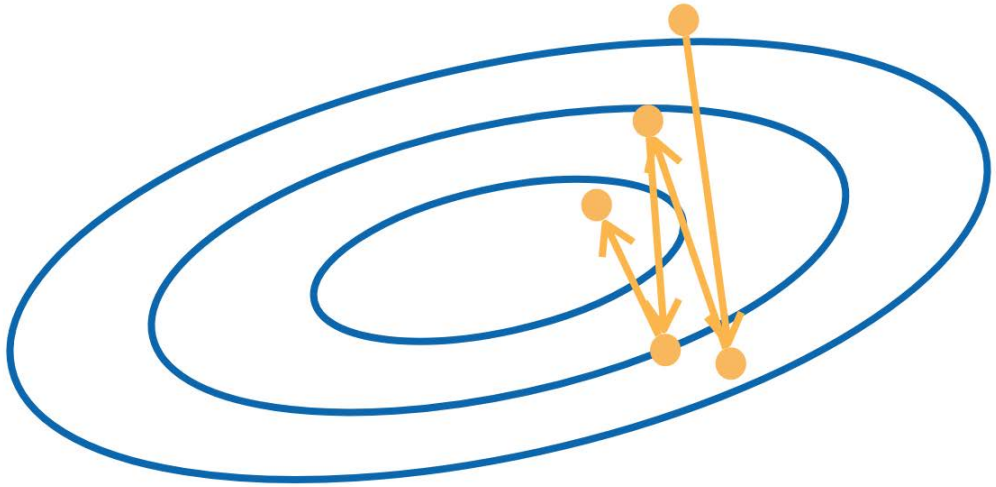
- $L(\mathbf{w}, b) = \sum_i l^{(i)}(\mathbf{w}, b)$
- $(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \eta \nabla L(\mathbf{w}, b)$
- Learning rate: η, length of one step
- Hyper-parameter: parameters whose values are used to control the learning process.
- Q: How to choose a learning rate?

Choose a Learning Rate

Too small

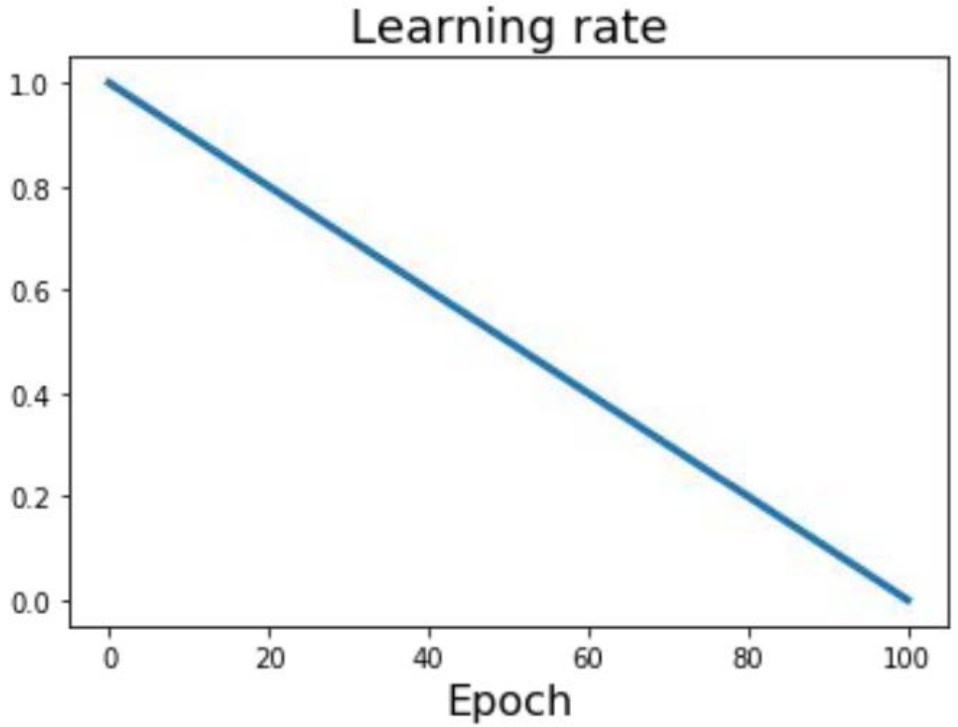


Too big

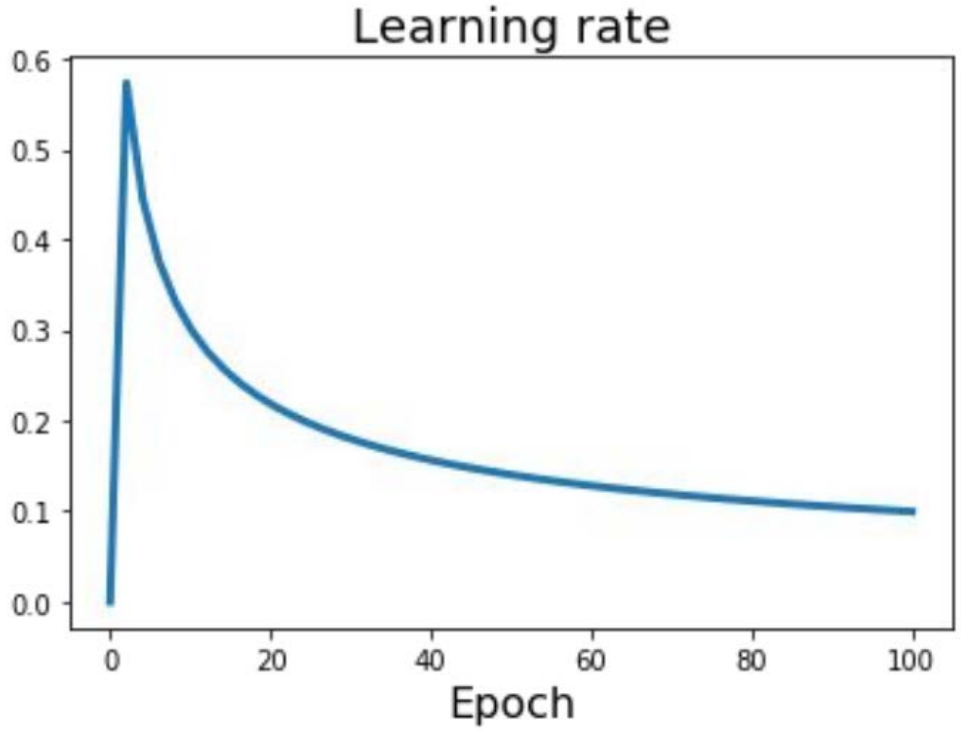


Learning Rate Scheduler

Linear Decay



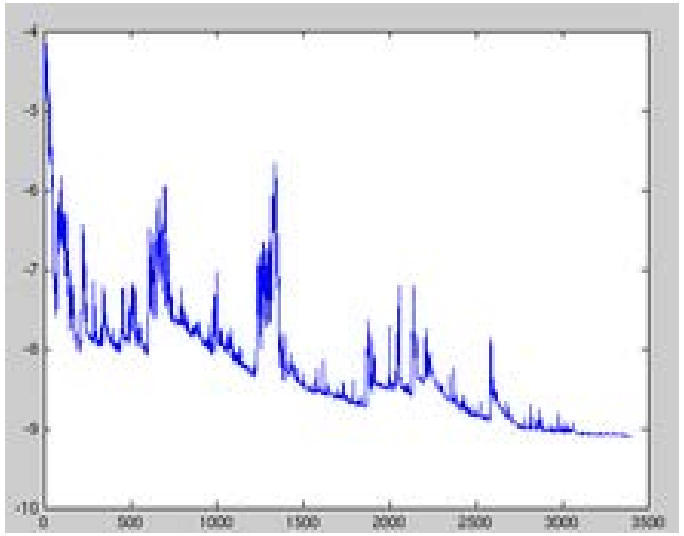
Linear Warmup



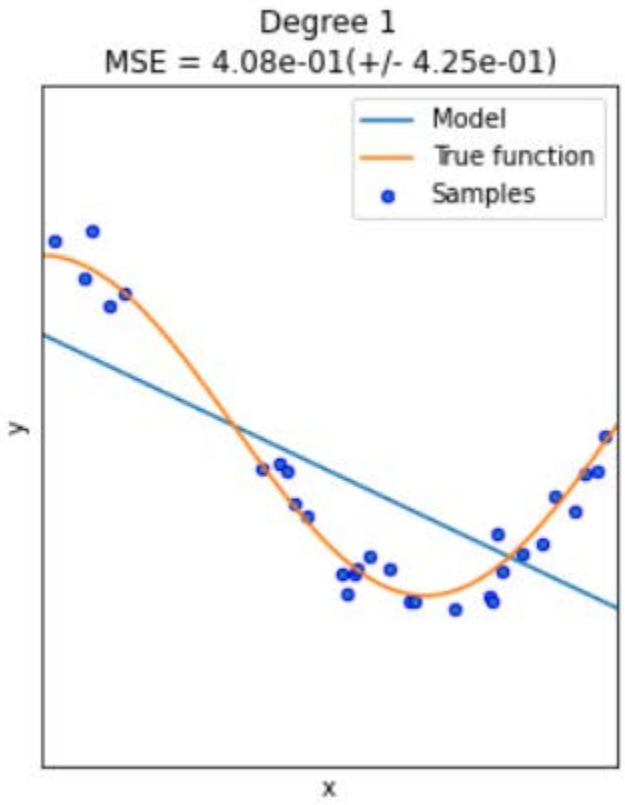
LR can also be searched using AutoML

Minibatch Stochastic Gradient Descent

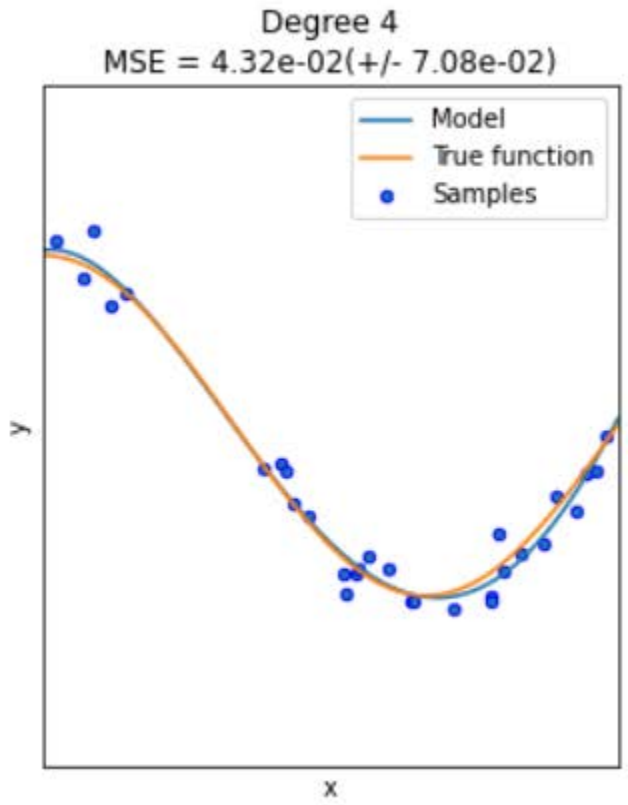
- Randomly sample several points B to approximate the loss.
- $L(\mathbf{w}, b) = \sum_{i \in B} l^{(i)}(\mathbf{w}, b)$
- Batch size $s = |B|$, another important hyper-parameters.
  - Too small: Workload is too small, hard to fully utilize computation resources; bad approximation.
  - Too big: high memory consumption.



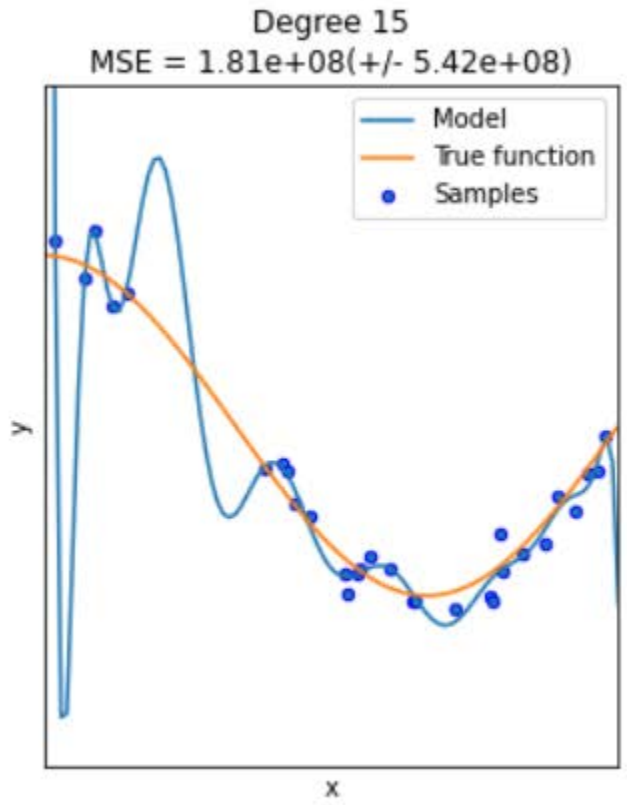
Overfitting and Underfitting



Underfitting 🤔

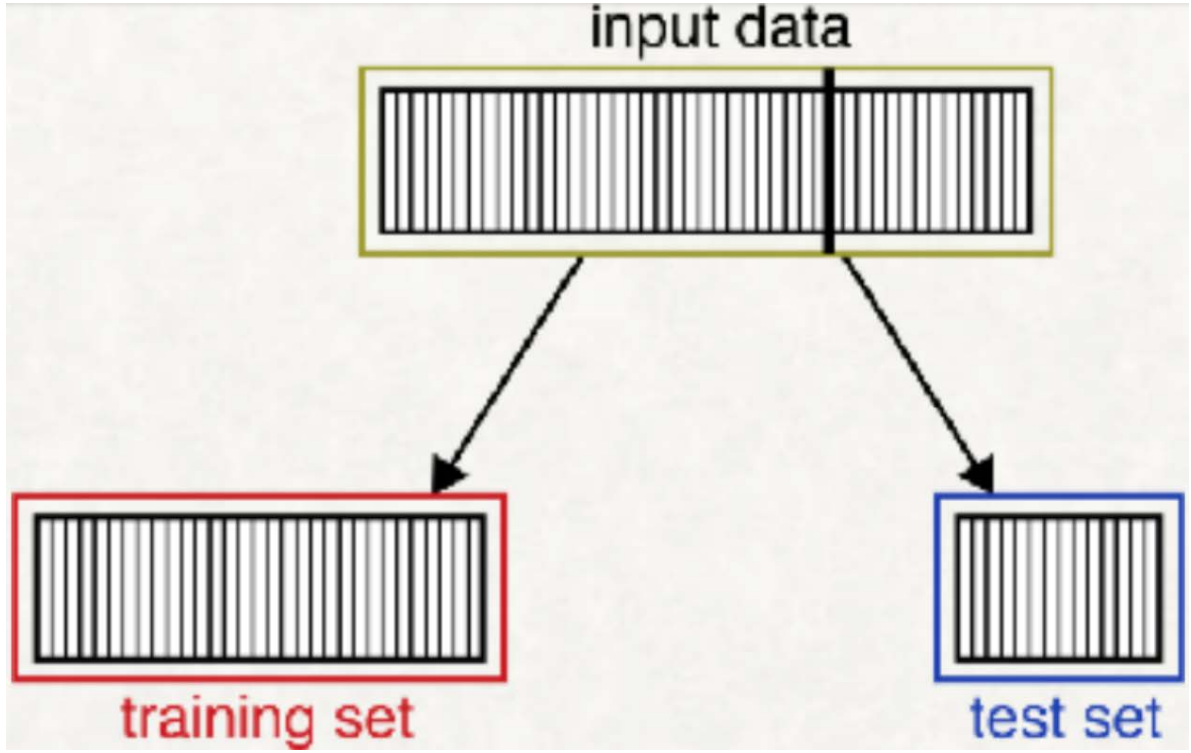


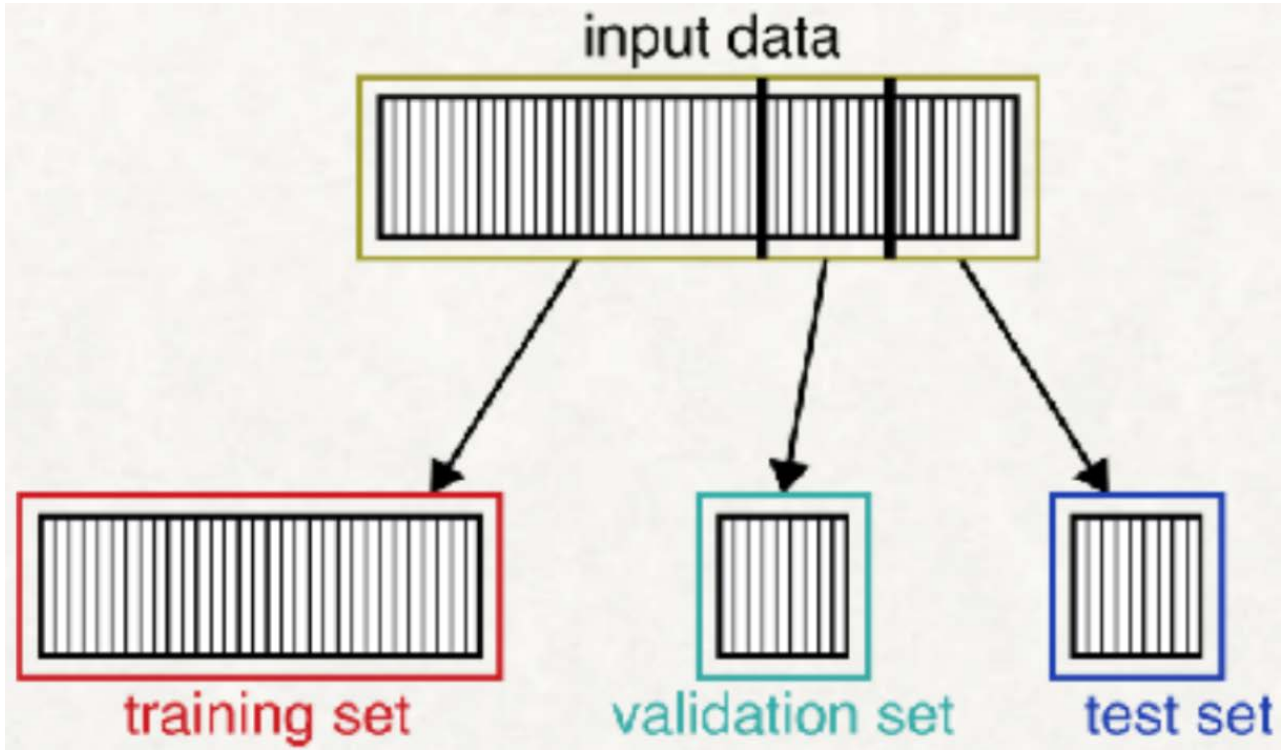
Optimum 😎



Overfitting 🤪

Training, validation, and test data sets





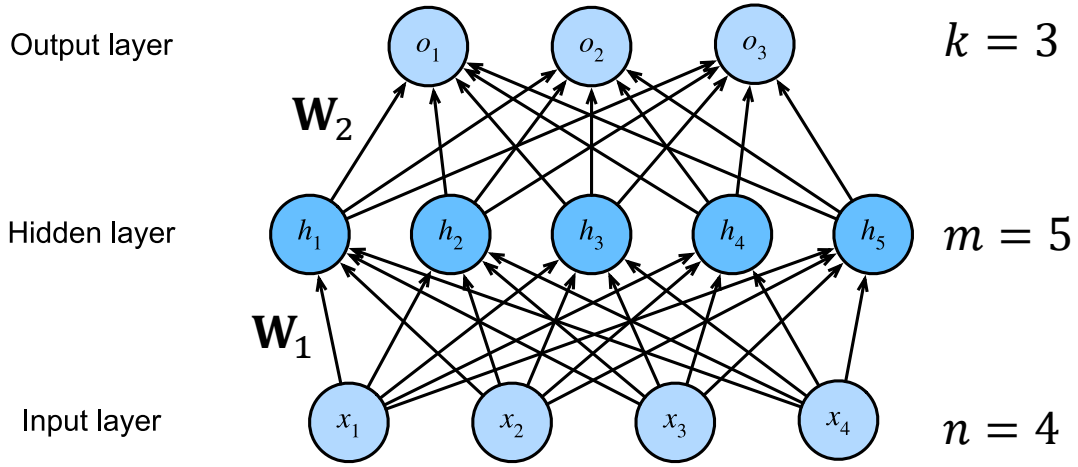
Training, validation, and test data sets

Any Questions?

Basic of deep learning

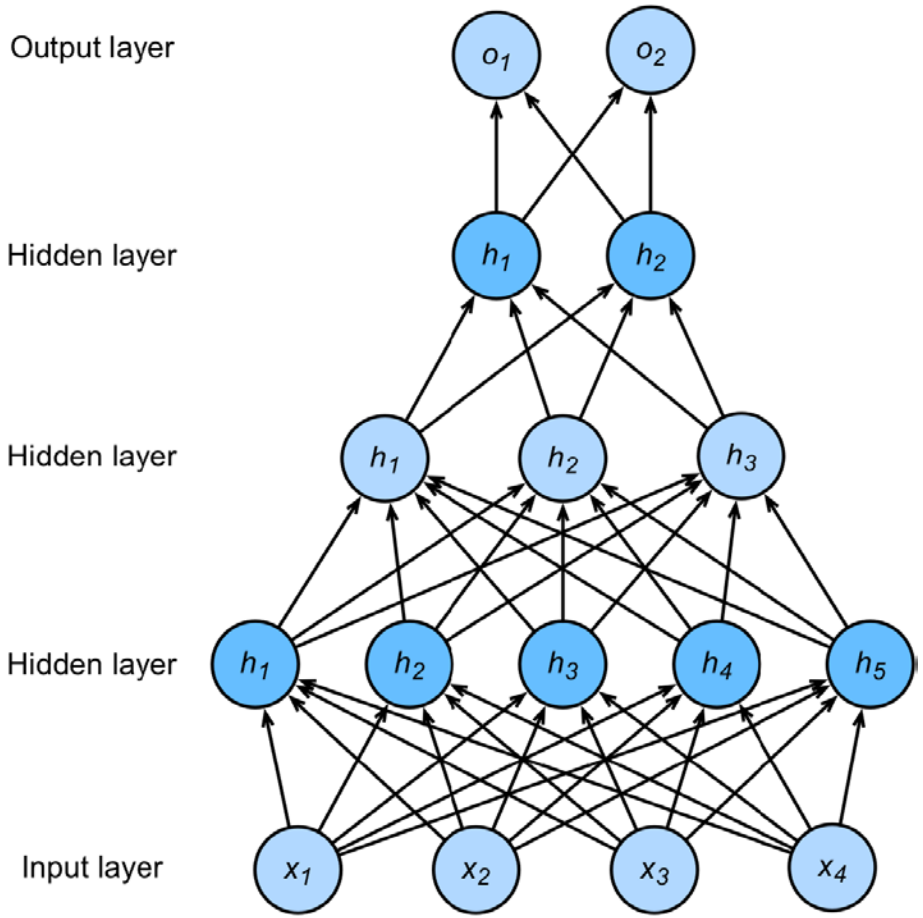
Multilayer Perceptrons

- Input: $\mathbf{x} \in \mathbb{R}^n$
- Hidden: $\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$
  - $\mathbf{W}_1 \in \mathbb{R}^{m \times n}, \mathbf{b}_1 \in \mathbb{R}^m, \mathbf{h} \in \mathbb{R}^m$
  - σ: activation function
- Output: $\mathbf{o} = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$
  - $\mathbf{W}_1 \in \mathbb{R}^{k \times m}, \mathbf{b}_1 \in \mathbb{R}^k, \mathbf{o} \in \mathbb{R}^k$
- $\hat{\mathbf{y}} = \sigma_2(\mathbf{o})$



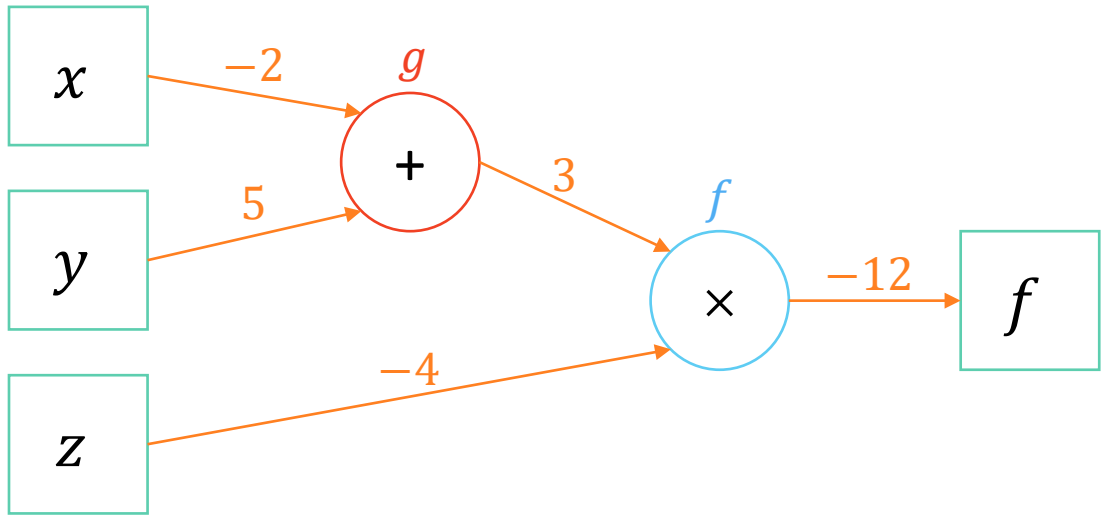
Multiple Layers

- $\mathbf{h}_1 = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$
- $\mathbf{h}_2 = \sigma(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$
- $\mathbf{h}_3 = \sigma(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3)$
- $\mathbf{o} = \mathbf{W}_4\mathbf{h}_3 + \mathbf{b}_4$
- Q: How to calculate gradient as network gets deeper?



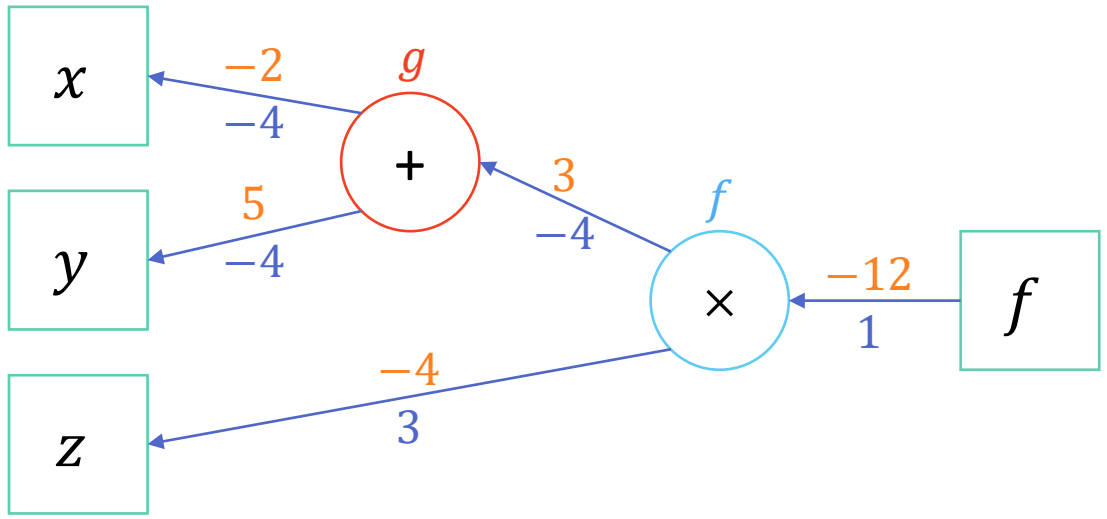
Computation Graph: Forward Propagation

- $f = f(x, y, z) = (x + y)z$
  - $g = x + y$
  - $f = gz$
- e.g. $x = -2, y = 5, z = -4$

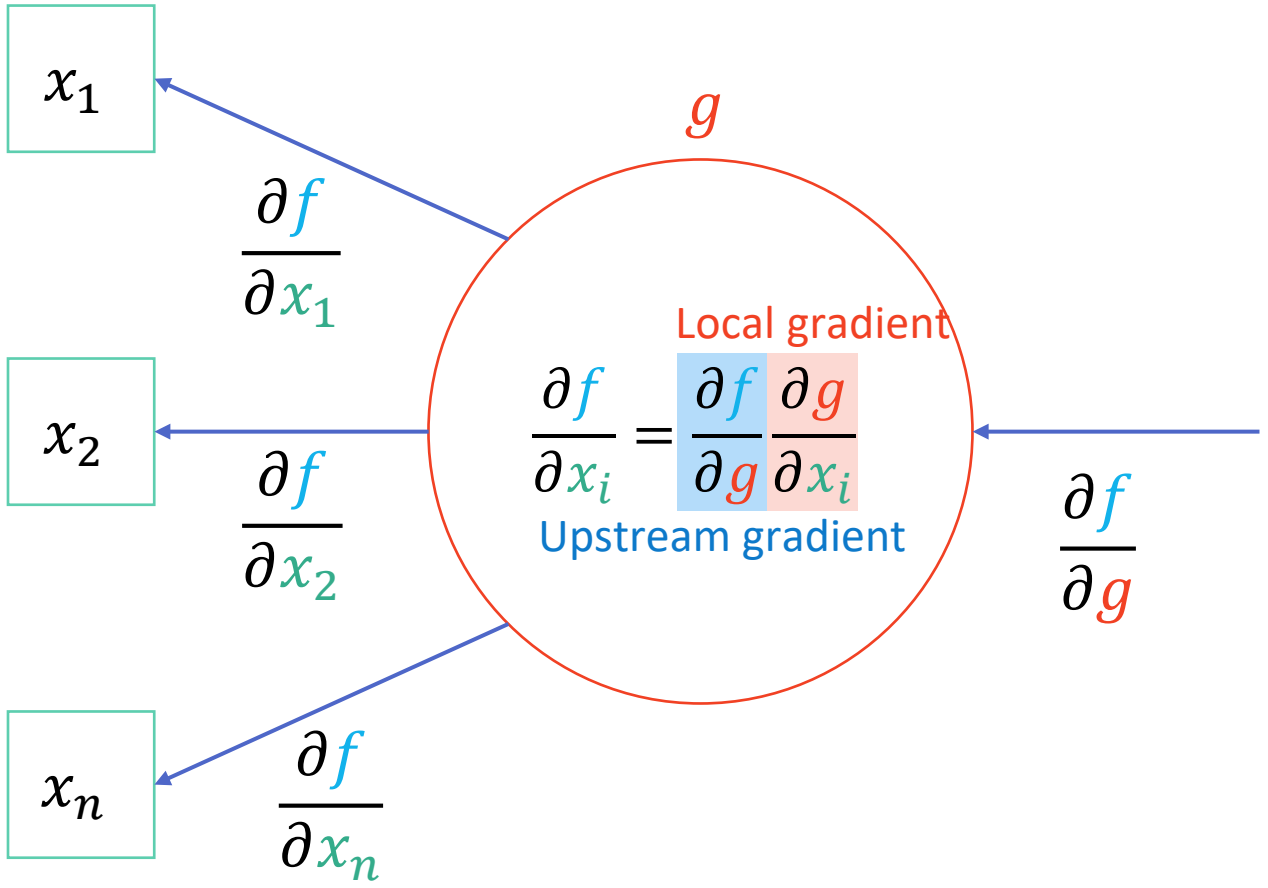


Computation Graph: Back Propagation

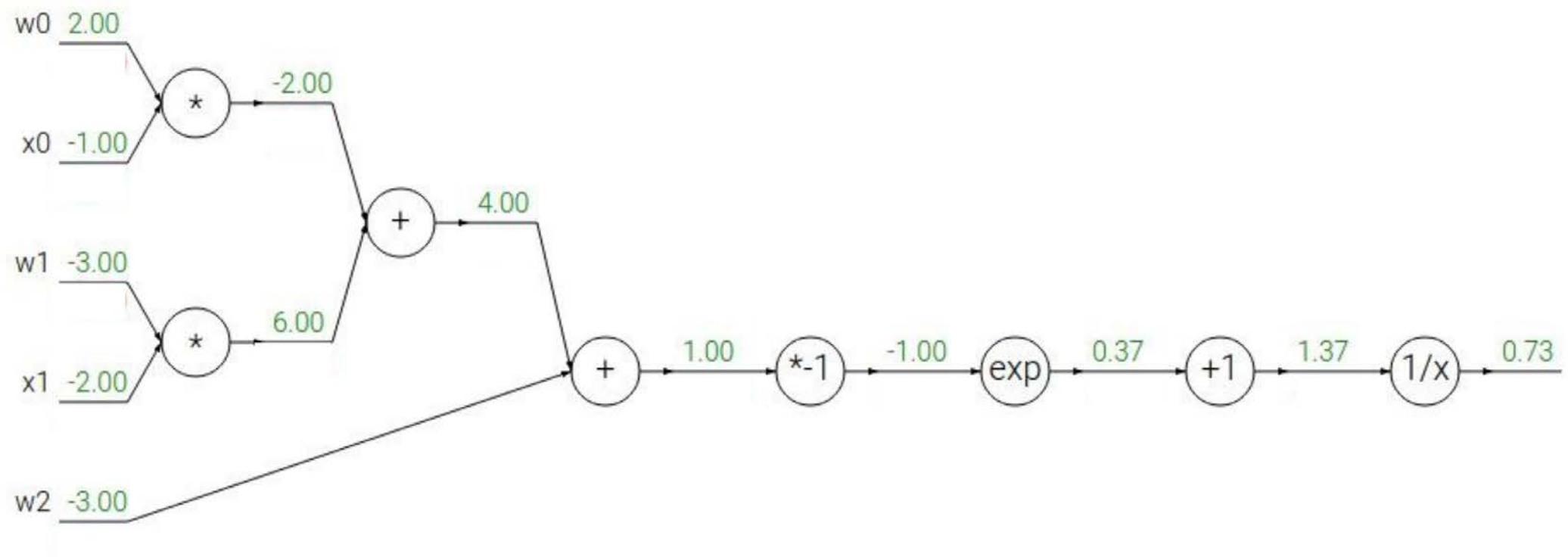
- $f = f(x, y, z) = (x + y)z$
  - $g = x + y$
  - $f = gz$
- e.g. $x = -2, y = 5, z = -4$
- Find $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$
- $\frac{\partial f}{\partial f} = 1$
- $\frac{\partial f}{\partial g} = z = -4, \frac{\partial f}{\partial z} = g = 3$
- $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x} = -4, \frac{\partial f}{\partial y} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial y} = -4$



Back Propagation: Chain Rule

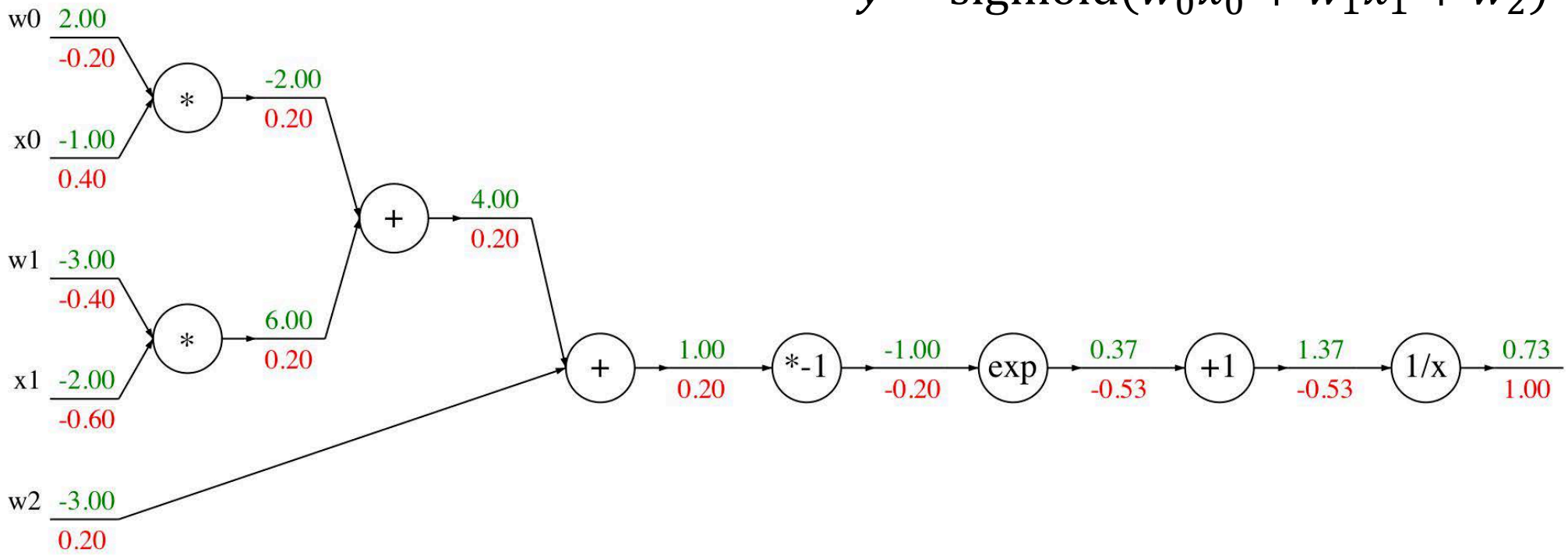


Another Example



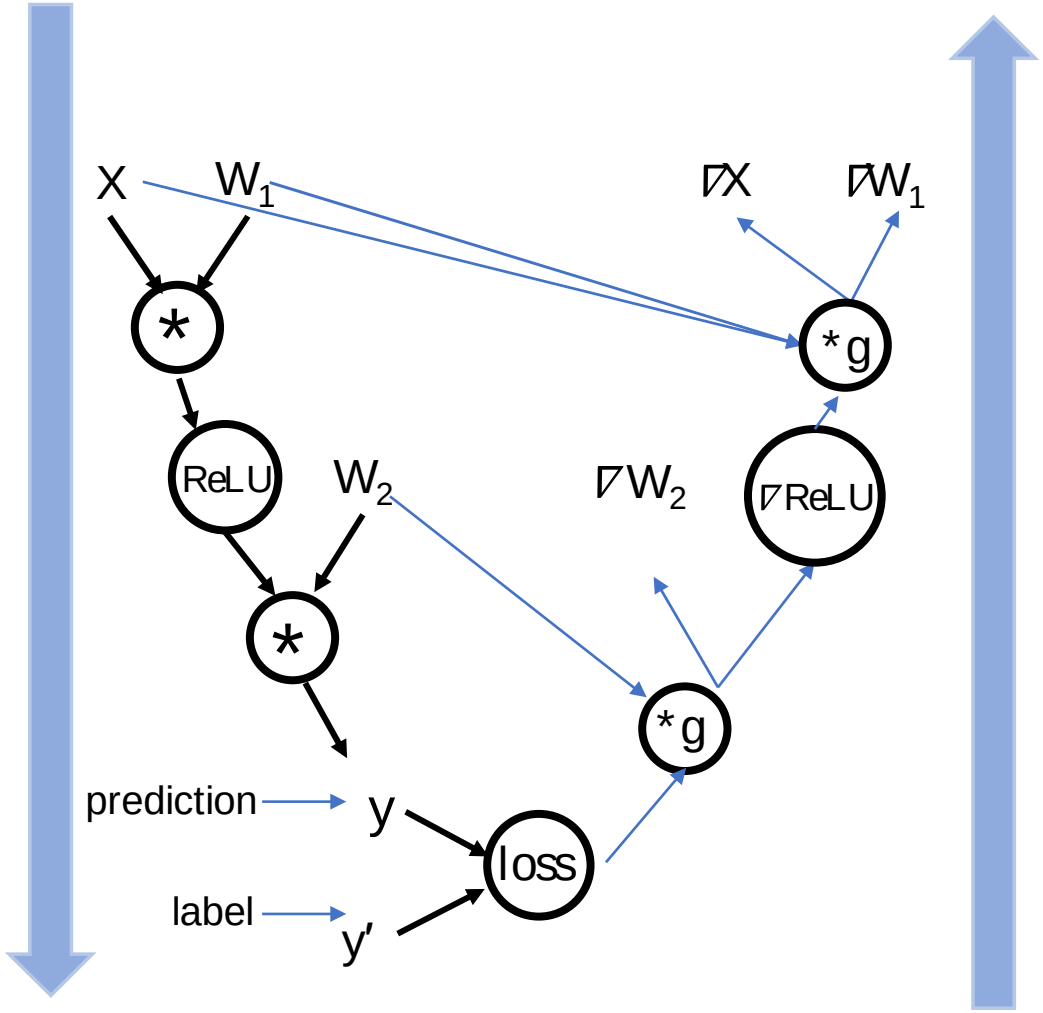
Another Example

$$y = \text{sigmoid}(w_0 x_0 + w_1 x_1 + w_2)$$



Computation Graph

- 🔽 : calculate $f(x)$
- 🔼 : calculate $f'(x)$
- Usually, each node corresponds to an operator in learning framework



Other components in DL

- Layers
  - Activation Function
  - Loss Function
  - Regularization
  - Dropout
  - Normalization
- Optimizers

Problems in DL

- Fitting Non-linear Functions
- Gradient Vanishment
- Overfitting

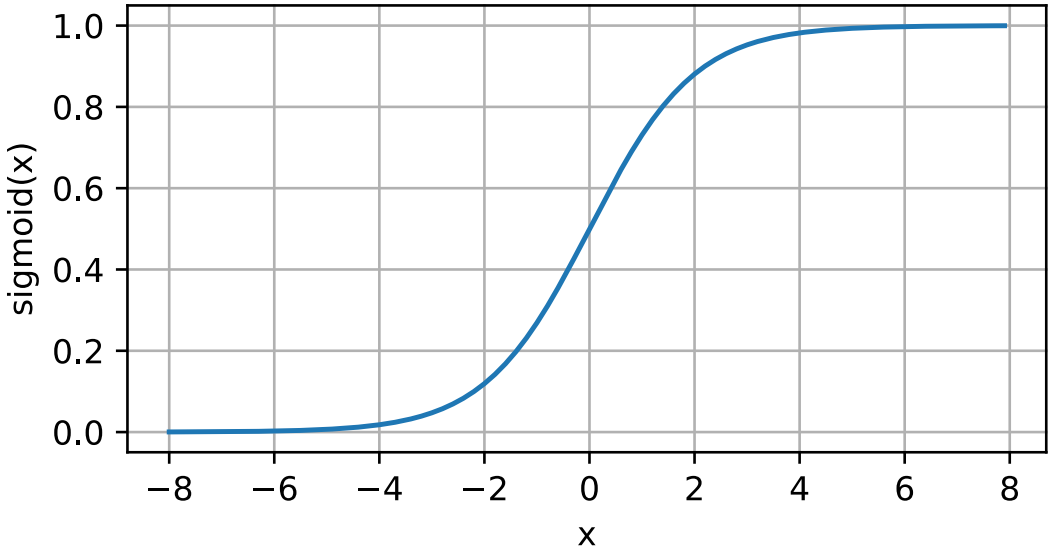
Activation Function

- Activation functions make MLP nonlinear
  - w/o activation function:
    - $\mathbf{h} = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1$
    - $\mathbf{o} = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2 = \mathbf{W}_2(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 = \mathbf{W}_2\mathbf{W}_1\mathbf{x} + \mathbf{b}'$ -> linear…
  - w/ activation function:
    - $\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$
- What to use for σ?

Sigmoid

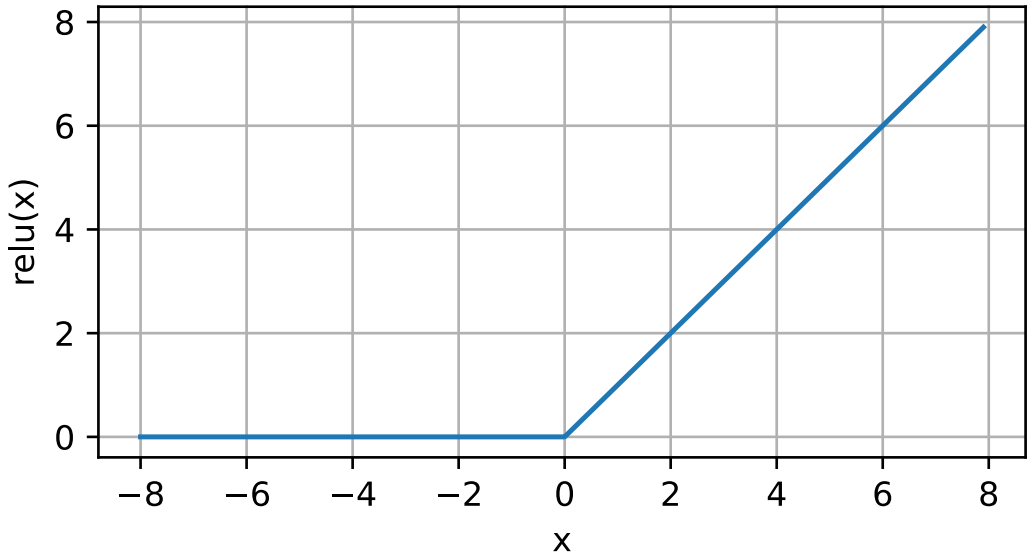
- $\text{sigmoid}(x) = \frac{1}{1+\exp(-x)}$
- $\frac{d}{dx}\text{sigmoid}(x)$

$$= \frac{\exp(-x)}{(1+\exp(-x))^2}$$
$$= \text{sigmoid}(x)\,(1-\text{sigmoid}(x))$$



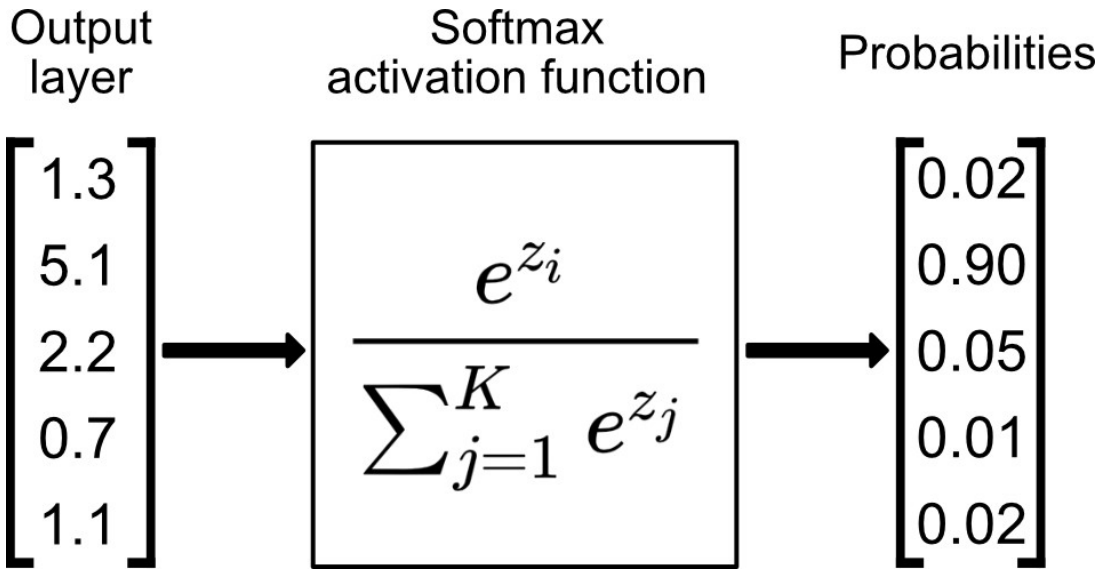
Rectified Linear Unit (ReLU)

$$\mathrm{ReLU}(x) = \max(x, 0)$$



Softmax

- $\text{softmax}(\mathbf{o})_i = \frac{\exp(o_i)}{\sum_{j=1}^{k} \exp(o_j)}$
  - $i = 1, \ldots, k$
  - $\mathbf{o} = (o_1, \ldots, o_k) \in \mathbb{R}^k$



Loss Function

- Cross Entropy
- Mean Square Error

True probability distribution (one-hot)

$$H(p, q) = -\sum_{x \in \text{classes}} p(x) \log q(x)$$

Your model's predicted probability distribution

Mean

Error

Squared

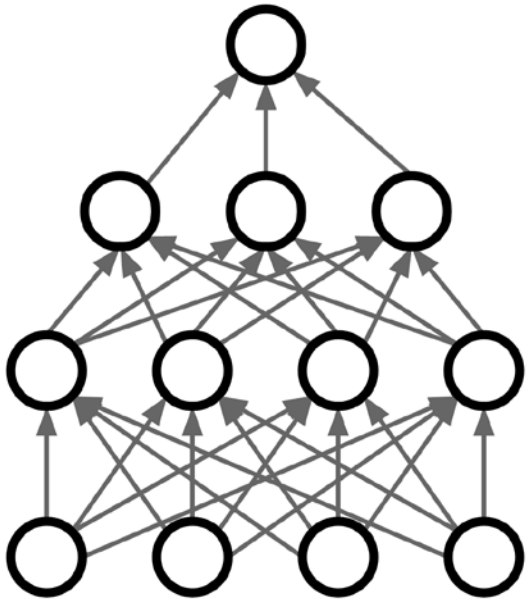
$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

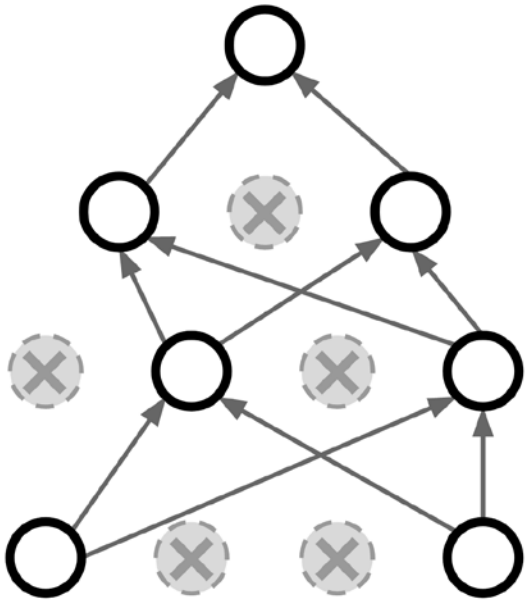
Regularization

- Regularization term
  - $L'(W) = L(W) + \lambda R(W)$
  - L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$
  - L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

Dropout (Srivastava et al., 2014)

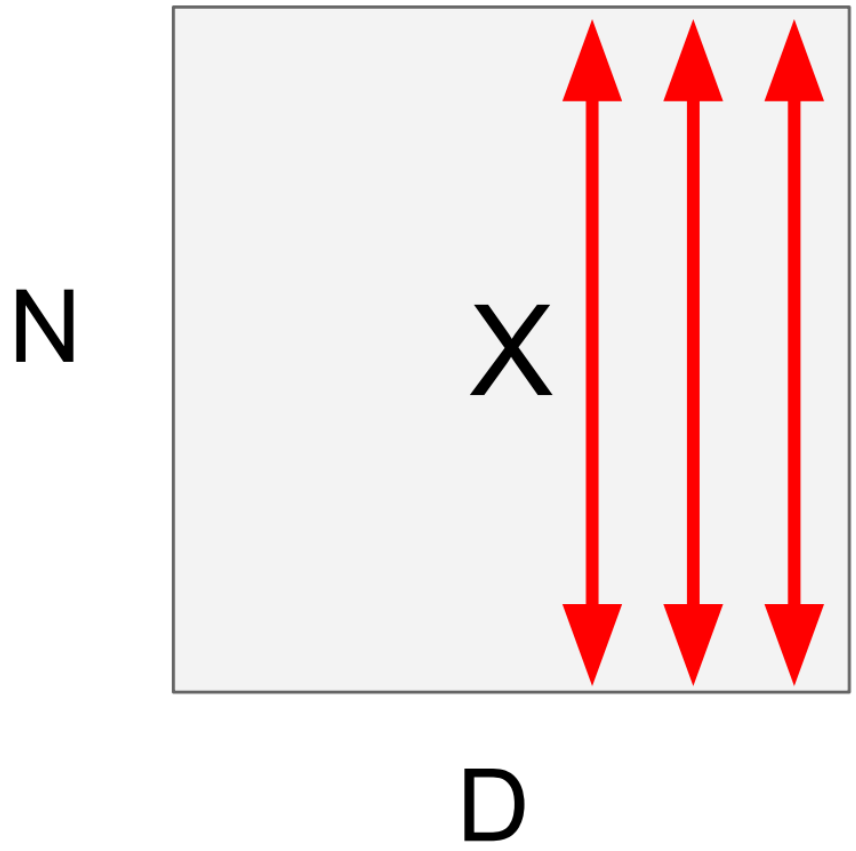
- In each forward pass, randomly set some neurons to zero
- Probability of dropping is a hyperparameter



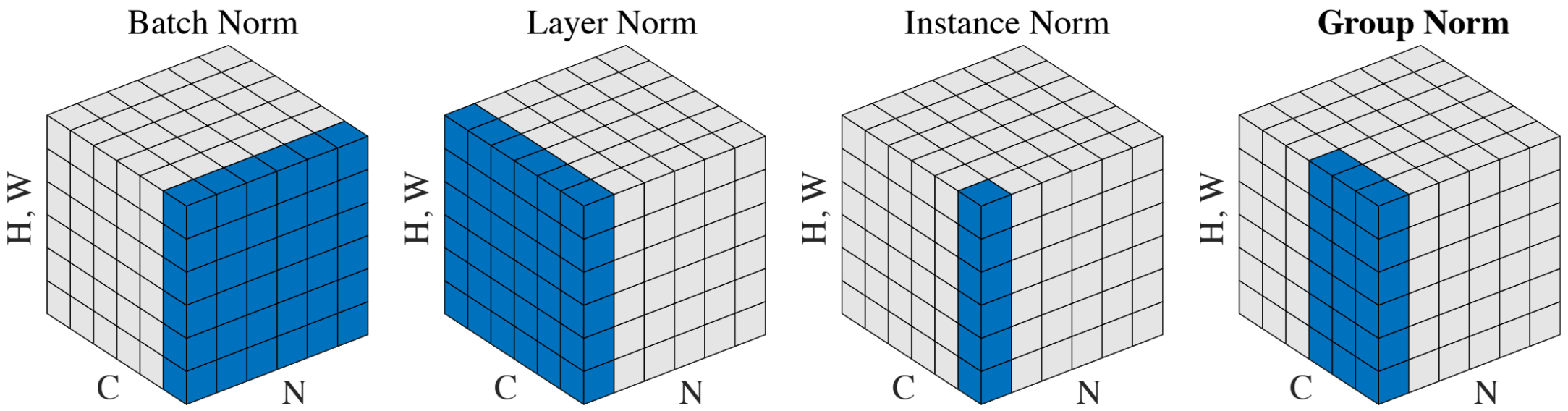


Batch Normalization

- Motivation: input x can be bad in distribution
  - Not centered around 0
  - Different scaling at each element
- Input: x NxD
- Mean: $\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$ D
- Var: $\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N} \left(x_{i,j} - \mu_j\right)^2$ D
- Output: $\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$ NxD
- Q: any information loss?

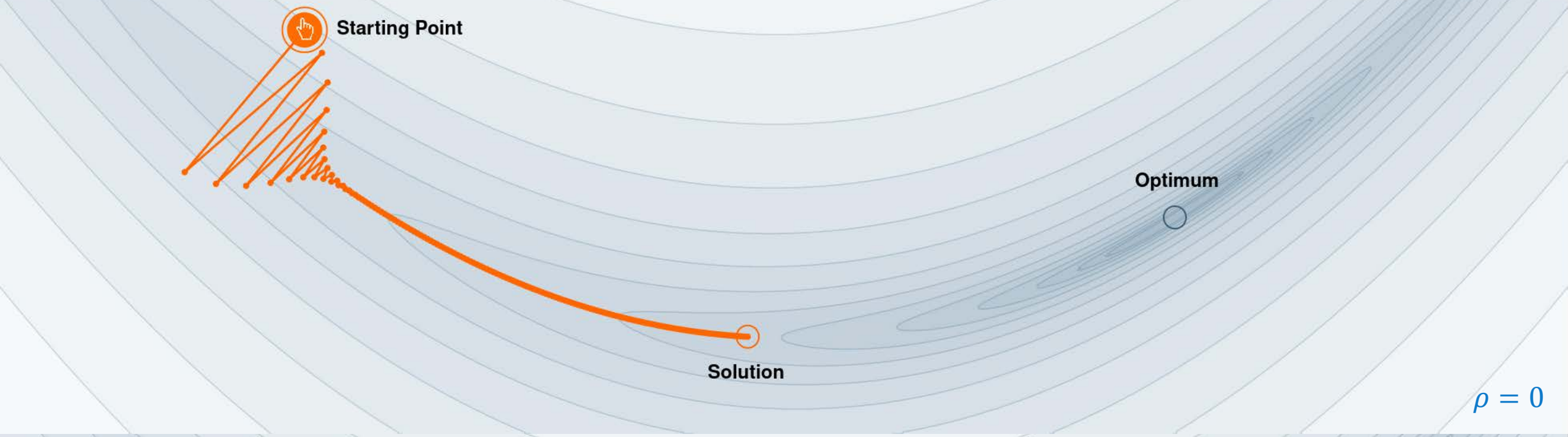


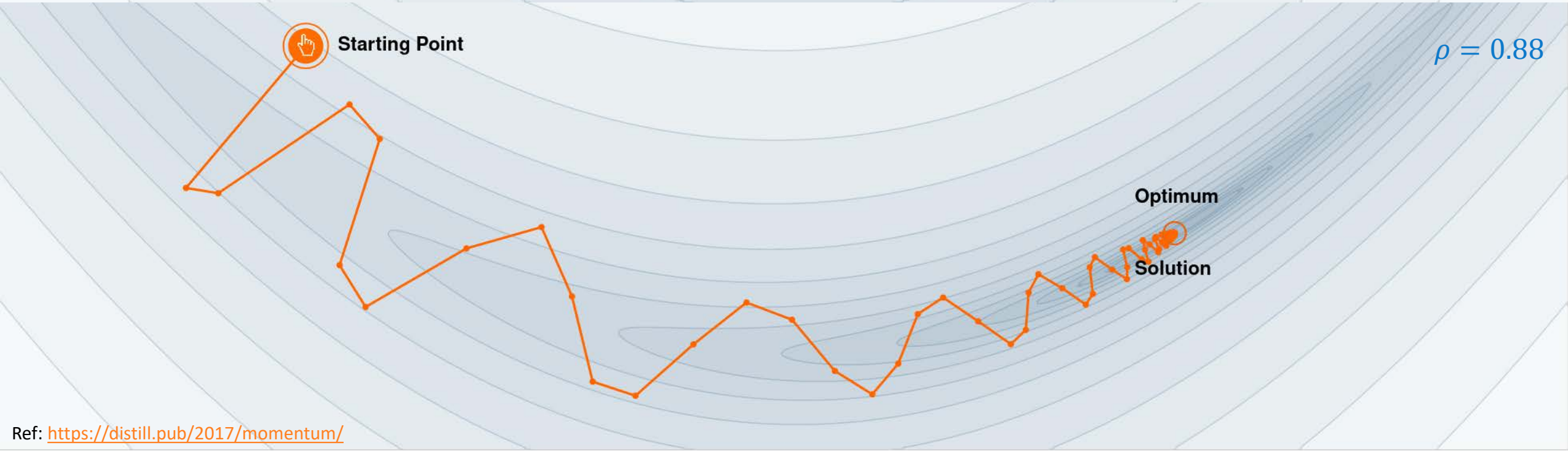
Normalization Methods



(Fancier) Optimizers

- SGD: $W \leftarrow W - \eta \nabla L(W)$
- SGD + Momentum (Sutskever et al., 2013)
  - $v \leftarrow \rho v + \nabla L(W)$
  - $W \leftarrow W - \eta v$
  - v: "velocity" as a running mean of gradients
  - ρ: "friction", typically $\rho = 0.9$ or $\rho = 0.99$
- Q: Why momentum?





Ref: https://distill.pub/2017/momentum/

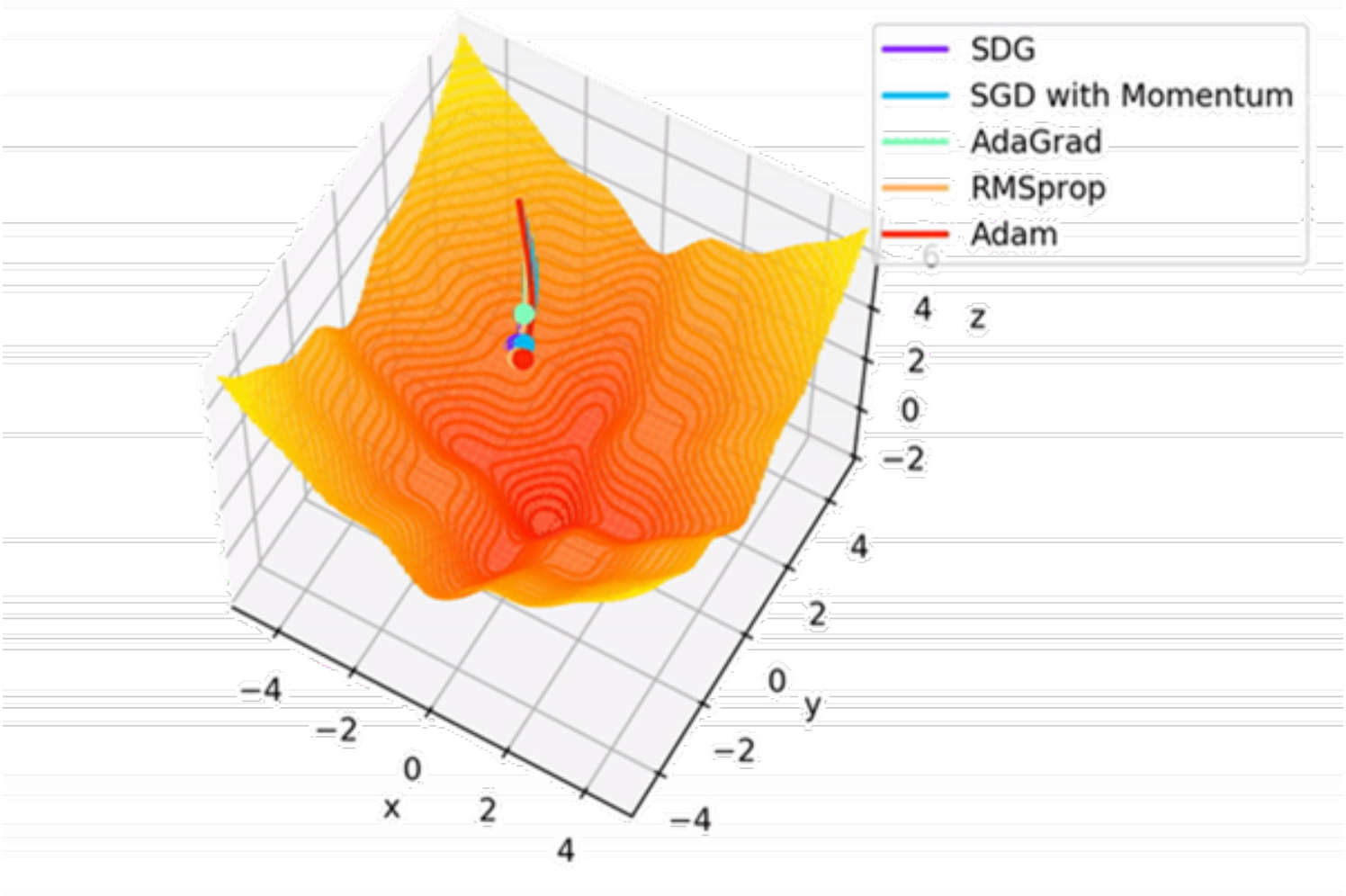
(Fancier) Optimizers

- SGD: $W \leftarrow W - \eta \nabla L(W)$
- AdaGrad (Duchi et al., 2011)
  - $s \leftarrow s + \left(\nabla L(W)\right)^2$
  - $W \leftarrow W - \eta \frac{\nabla L(W)}{\sqrt{s}+\epsilon}$
  - Intrinsic: $\eta \leftarrow \eta \,/\, \sqrt{\sum_0^{t-1}(gi^2)}$
  - Progress along “steep” directions is damped
  - Progress along “flat” directions is accelerated
  - Step size decays to zero over long time 🤔

(Fancier) Optimizers

- SGD: $W \leftarrow W - \eta \nabla L(W)$
- 🚀 Adam (Kingma et al., 2015)
  - At each iteration $i = 1, \ldots, n$:
  - $v \leftarrow \beta_1 v + (1 - \beta_1) \nabla L(W)$
  - $s \leftarrow \beta_2 s + (1 - \beta_2) \left(\nabla L(W)\right)^2$
  - Bias correction: $v' = \frac{v}{1-\beta_1^i}$, $s' = \frac{s}{1-\beta_2^i}$
  - $W \leftarrow W - \eta \frac{v'}{\sqrt{s'} + \epsilon}$
  - Adam with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\eta = 10^{-3}$ or 5×10^{-4} is a great starting point for many models! 😎

Optimizer Comparison



Any Questions?

CNN

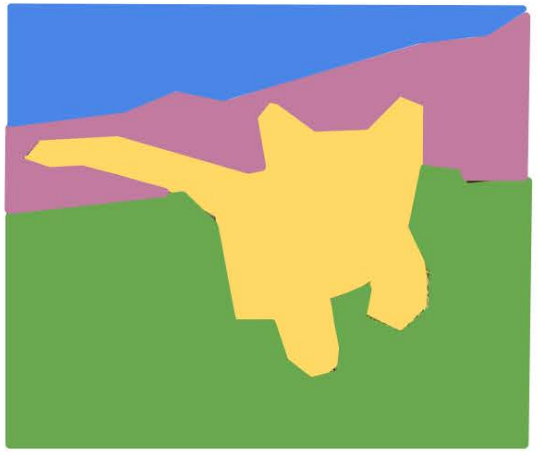
Computer Vision Tasks

Classification

CAT

No spatial extent

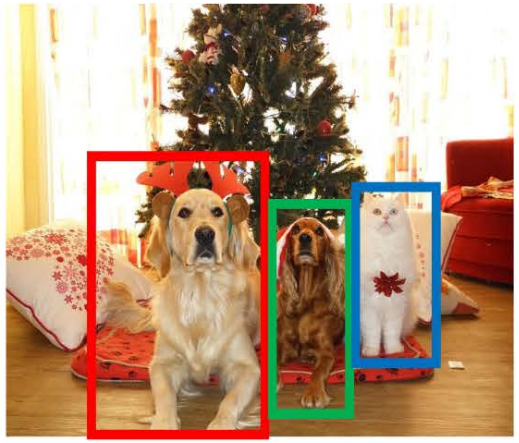
Semantic Segmentation



GRASS, **CAT**, **TREE**, **SKY**

No objects, just pixels

Object Detection



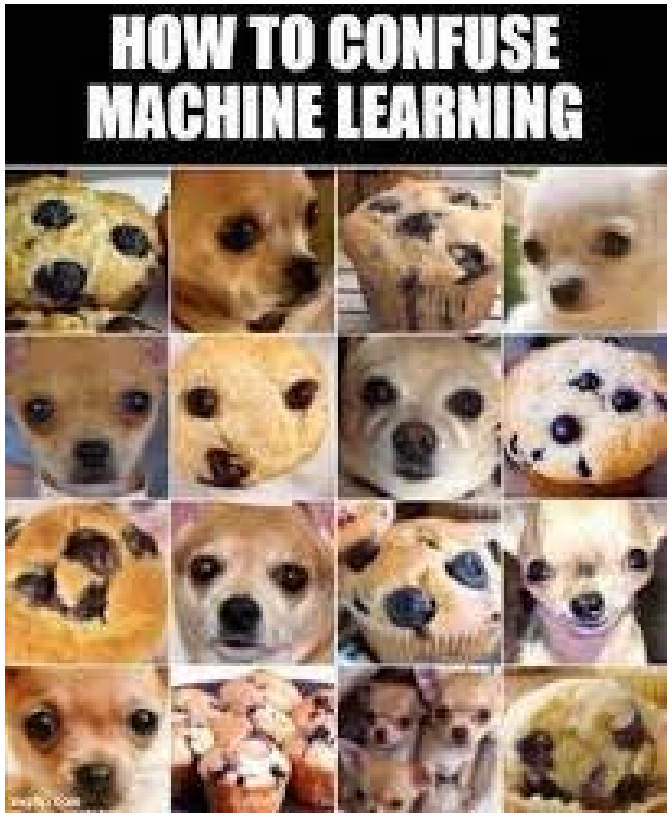
DOG, **DOG**, **CAT**

Instance Segmentation

DOG, **DOG**, **CAT**

Multiple Object

Computer Vision Tasks

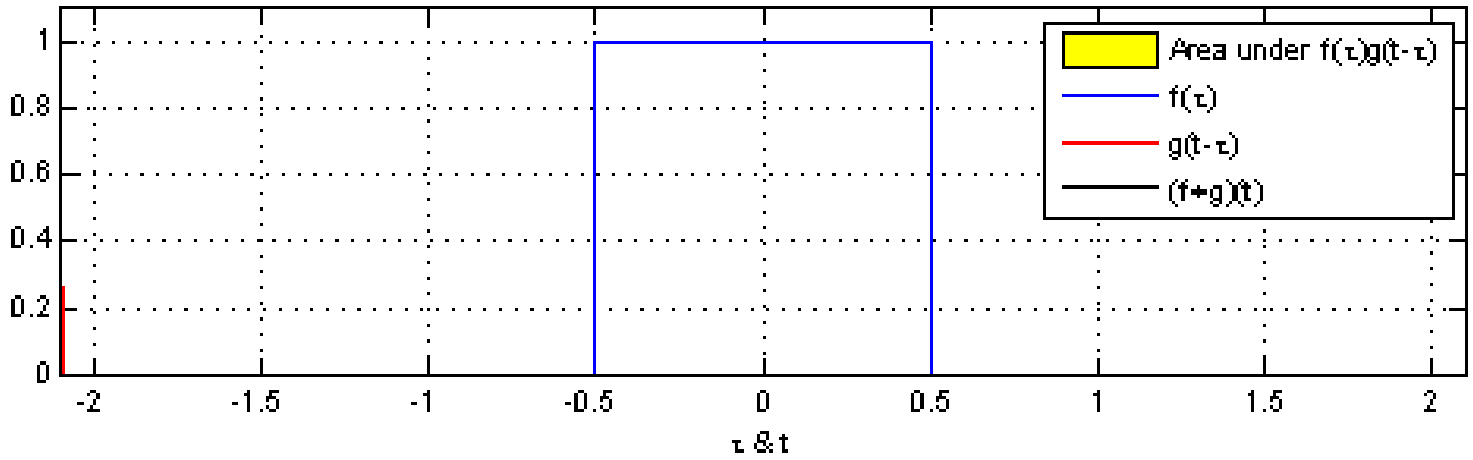


Computer Vision Tasks

Mask R-CNN, He et al., 2017

Convolution

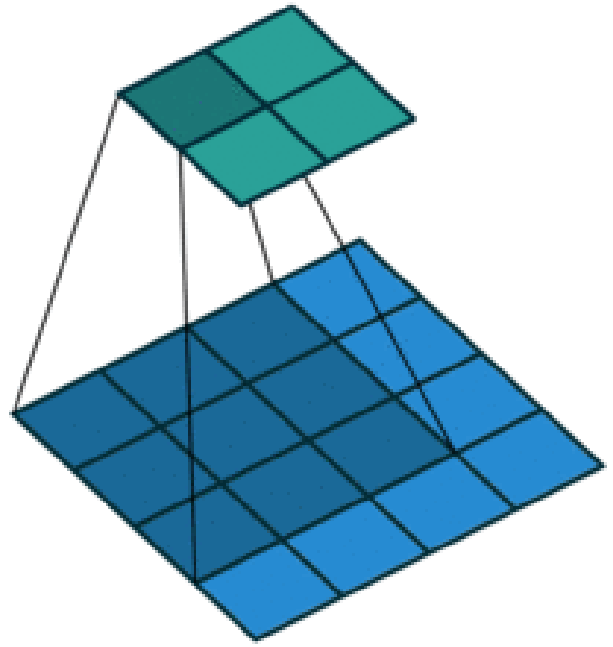
- $(f * g)(t) \coloneqq \int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau$
  - Discrete: $(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] g[n-m]$



2D Discrete Convolution

$$(f * g)(i,j) = \sum_a \sum_a f(a,b)g(i-a, j-b)$$

Input				Kernel			Output	
0	1	2	*	0	1	=	19	25
3	4	5		2	3		37	43
6	7	8						



Padding

- Avoid losing pixels
  - Padding with 0
  - Padding with nearest neighbor

Input

0	0	0	0	0
0	0	1	2	0
0	3	4	5	0
0	6	7	8	0
0	0	0	0	0

(Padding = 1)

*

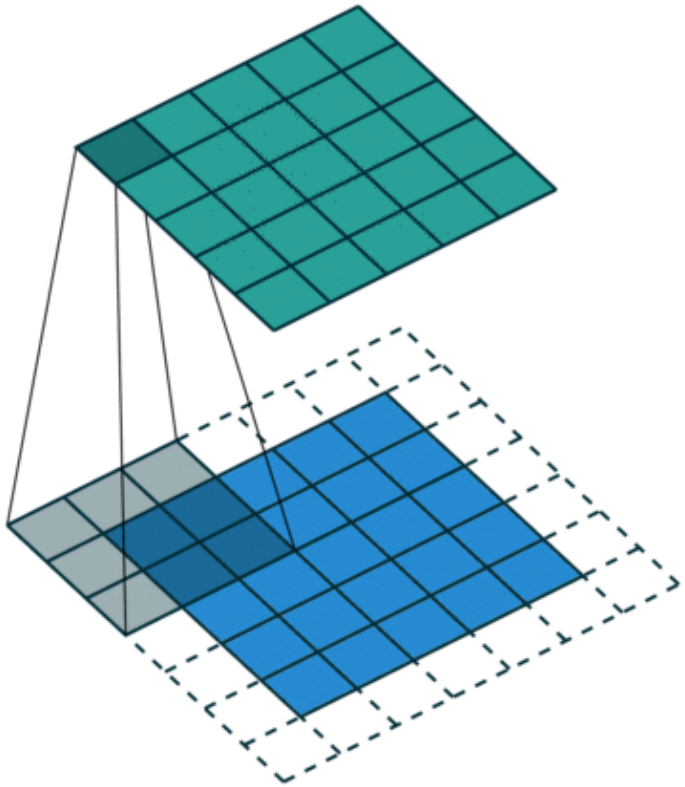
Kernel

0	1
2	3

=

Output

0	3	8	4
9	19	25	10
21	37	43	16
6	7	8	0



Stride

- Step length of movement of kernel

Input

0	0	0	0	0
0	0	1	2	0
0	3	4	5	0
0	6	7	8	0
0	0	0	0	0

(Stride x, y = 2, 3)

*

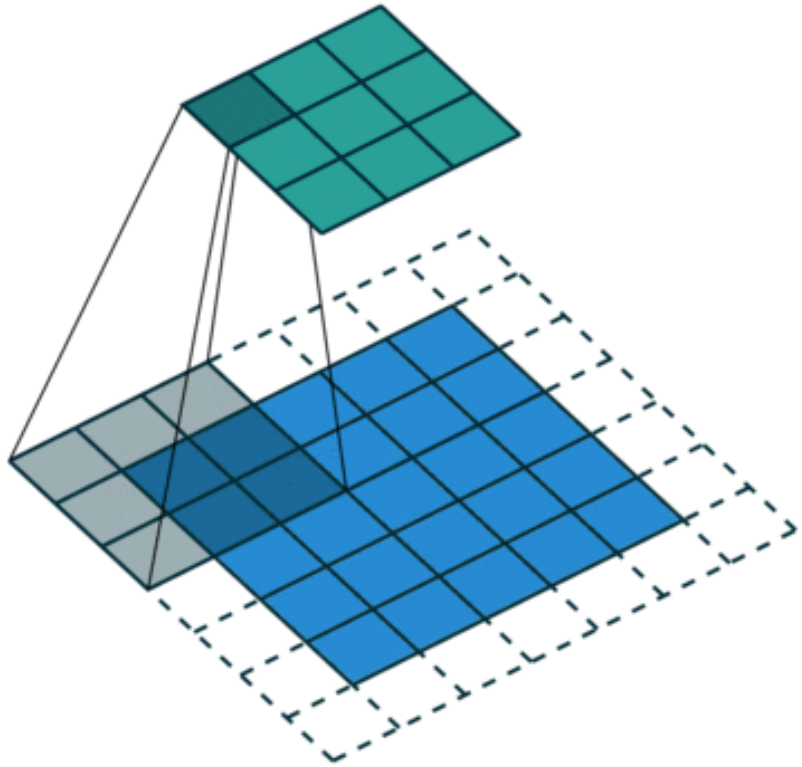
Kernel

0	1
2	3

=

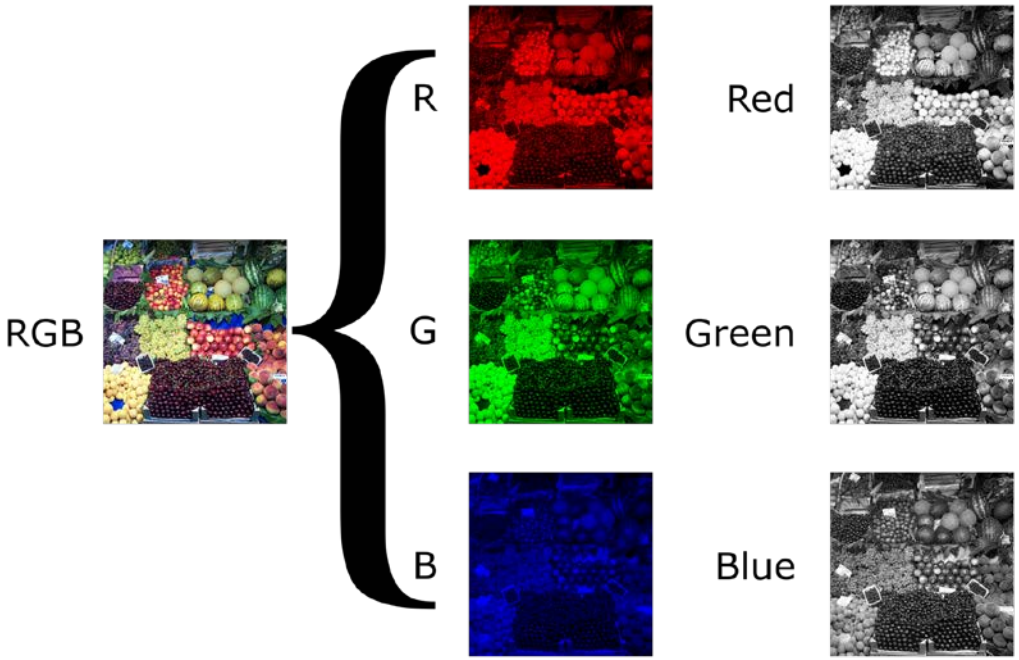
Output

0	8
6	8

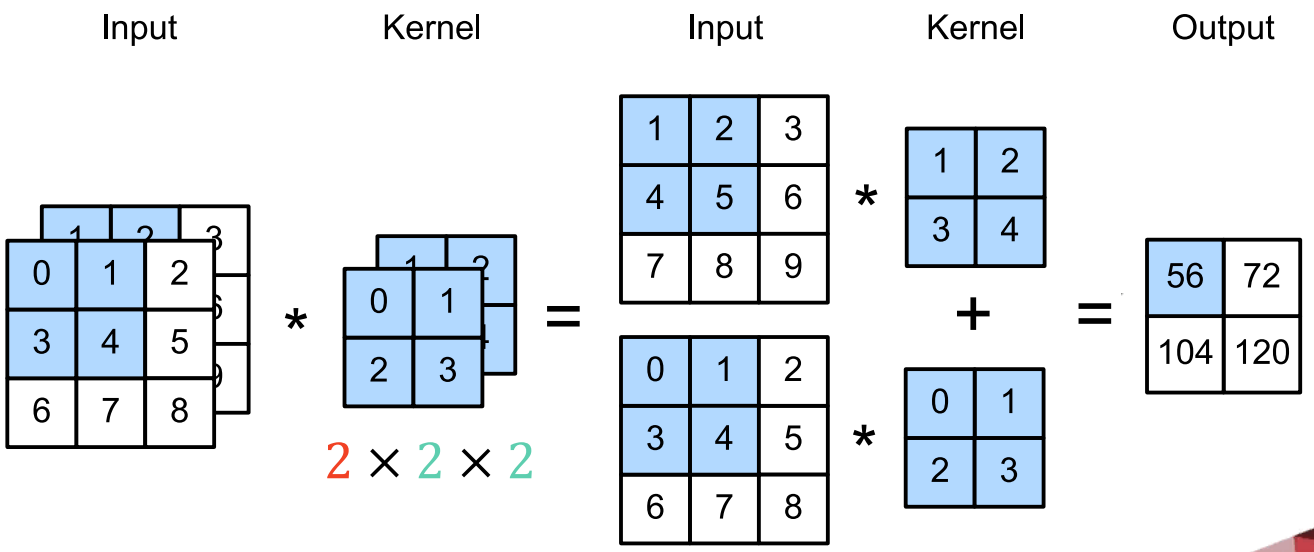


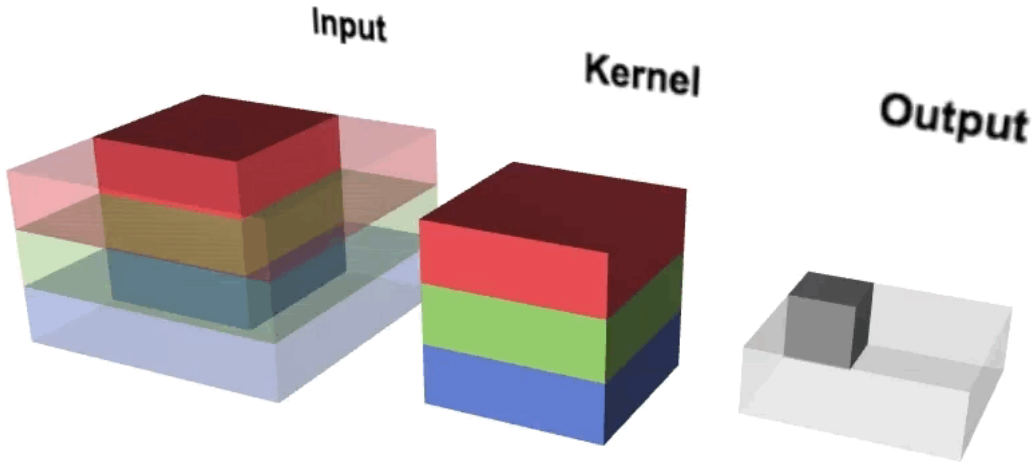
Multiple Input Channels

- Input data may contain multiple channels c_i.
- Kernel contains a tensor of shape $k_h \times k_w$ for every input channel.
- Concatenating these c_i tensors together yields a convolution kernel of shape $c_i \times k_h \times k_w$.



Multiple Input Channels

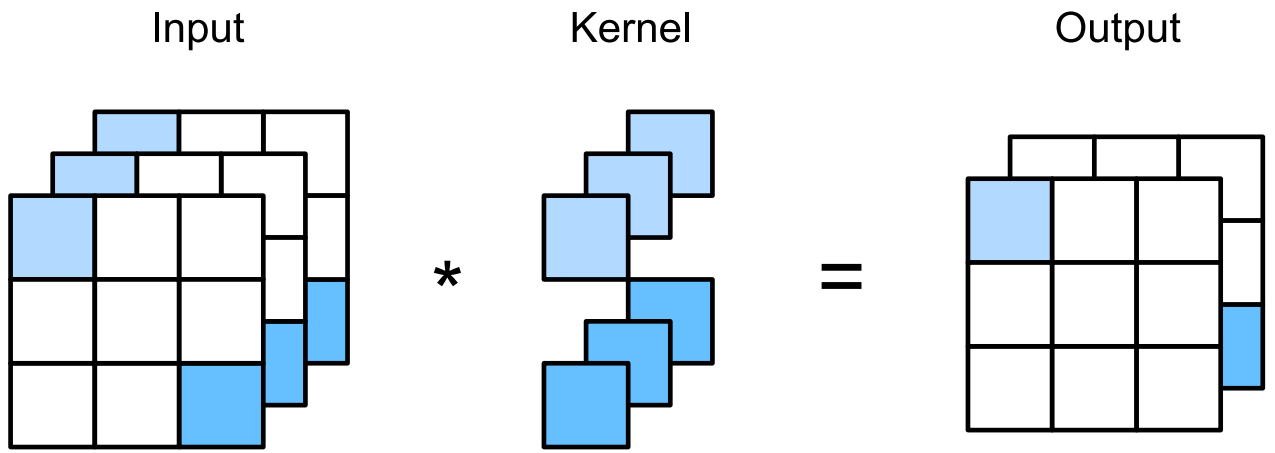




Multiple Output Channels

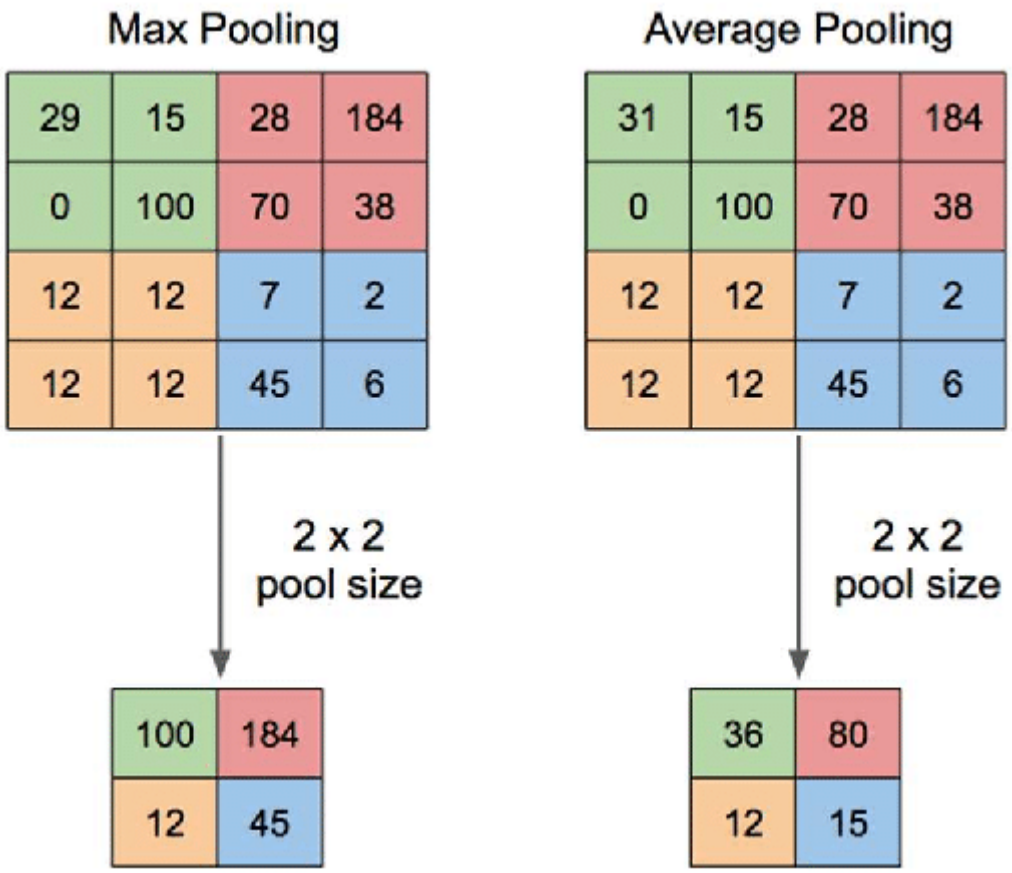
- Use c_o output different channels, each of shape $c_i \times k_h \times k_w$.
- Resulting shape of kernel:

$$c_o \times c_i \times k_h \times k_w$$

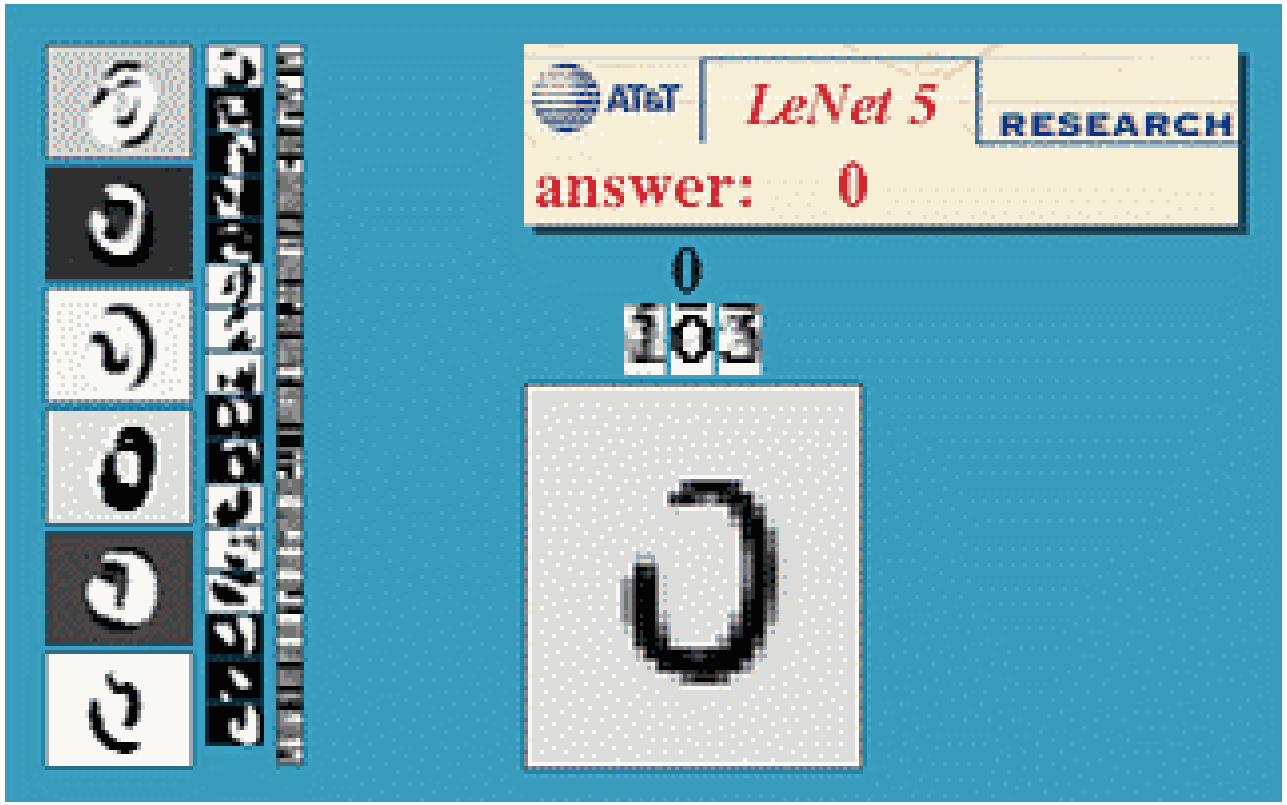


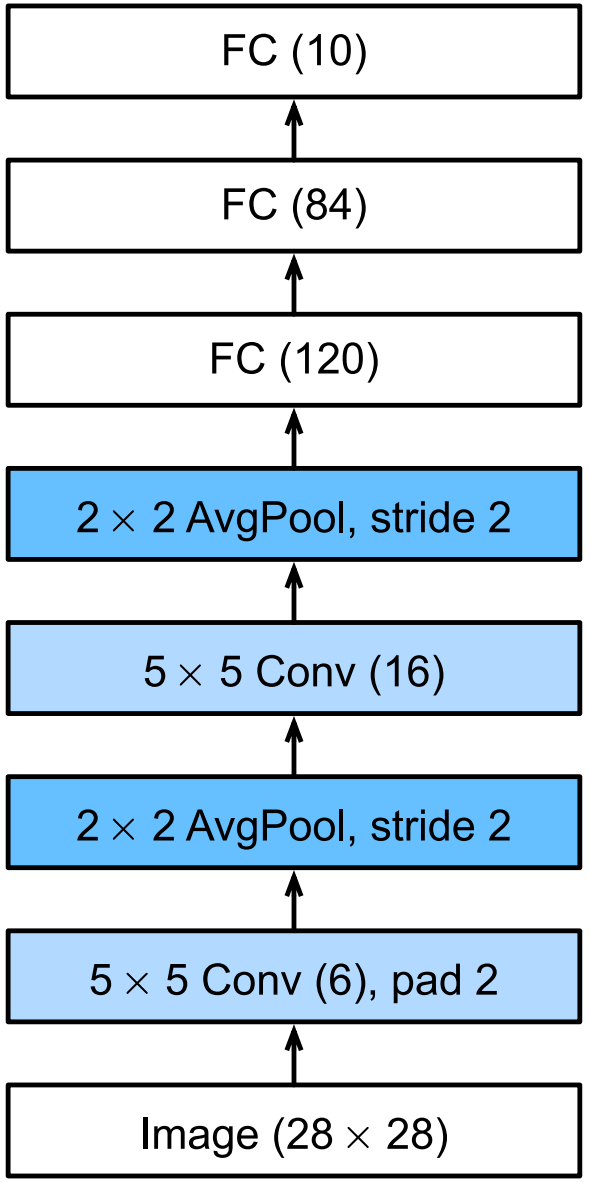
Pooling

- Again, a fixed-shape window sliding over the input matrix.
- Padding and strides also apply.
- No learnable parameters.
- Determine a value for each window.
  - Max pooling: max value in the window;
  - Avg. pooling: avg. value of the window.

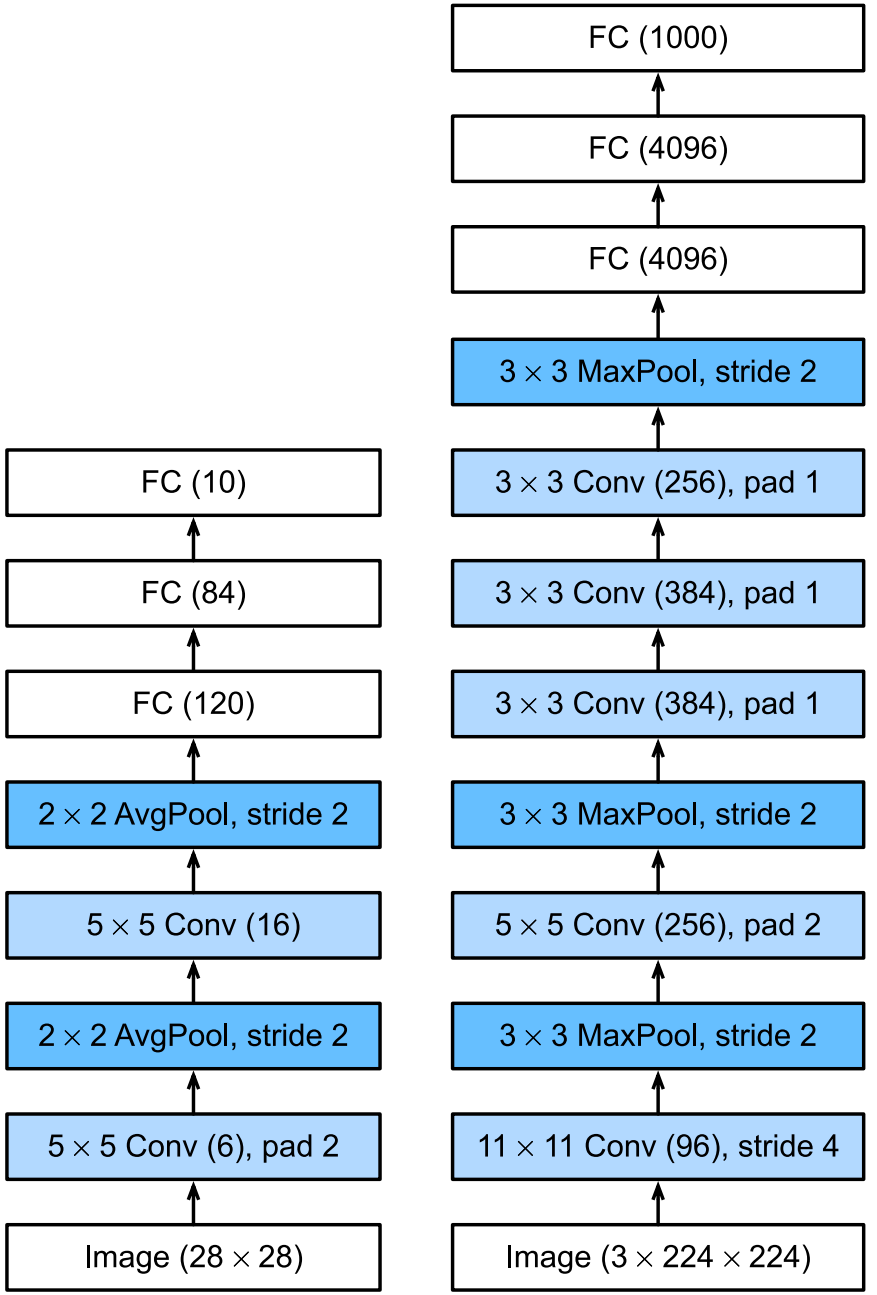


LeNet (LeCun et al., 1998)



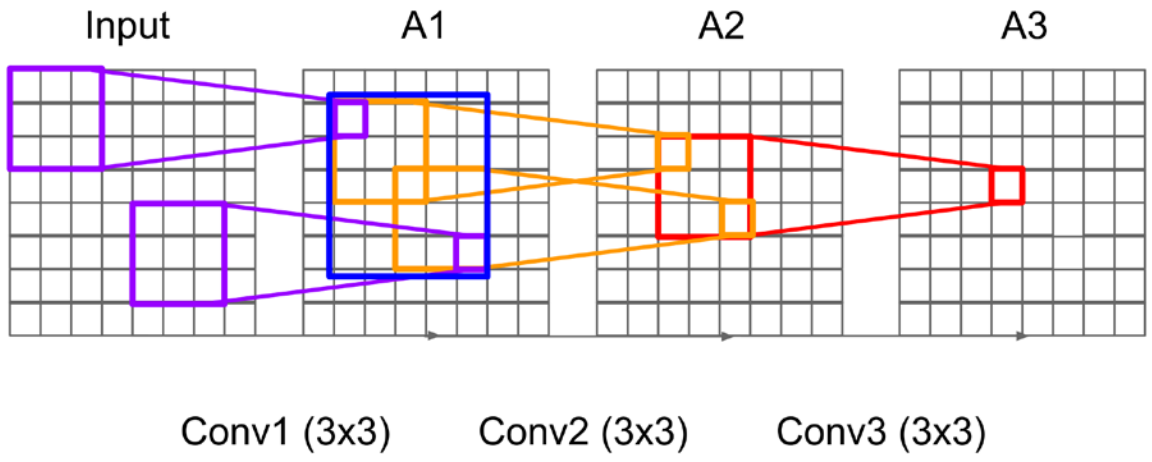


AlexNet (2012)



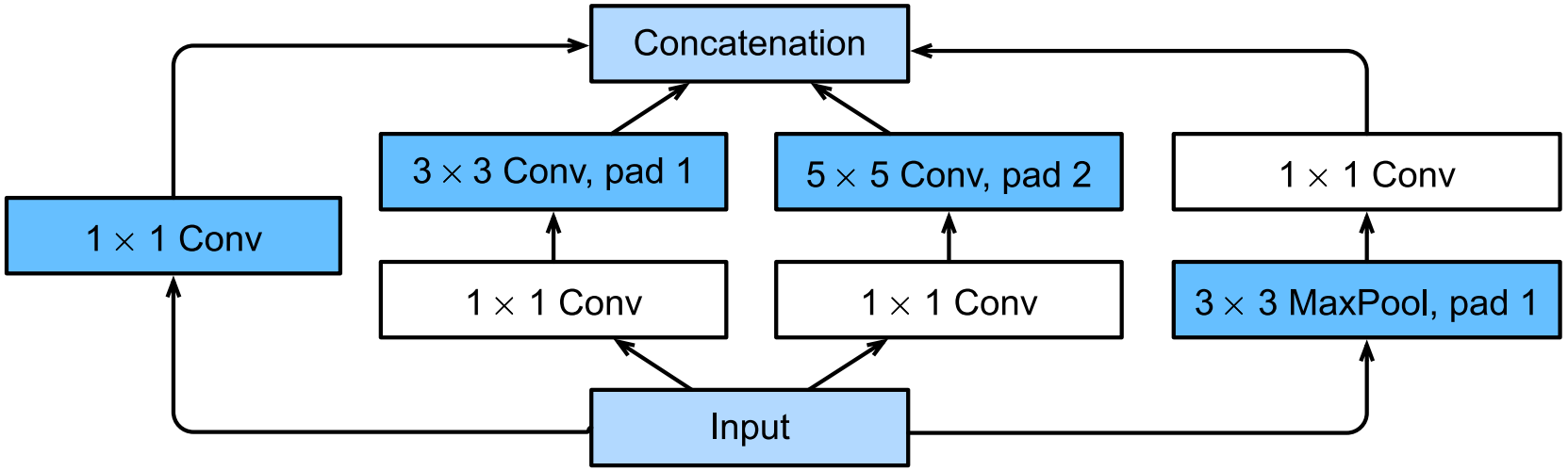
VGGNet (Simonyan et al., 2014)

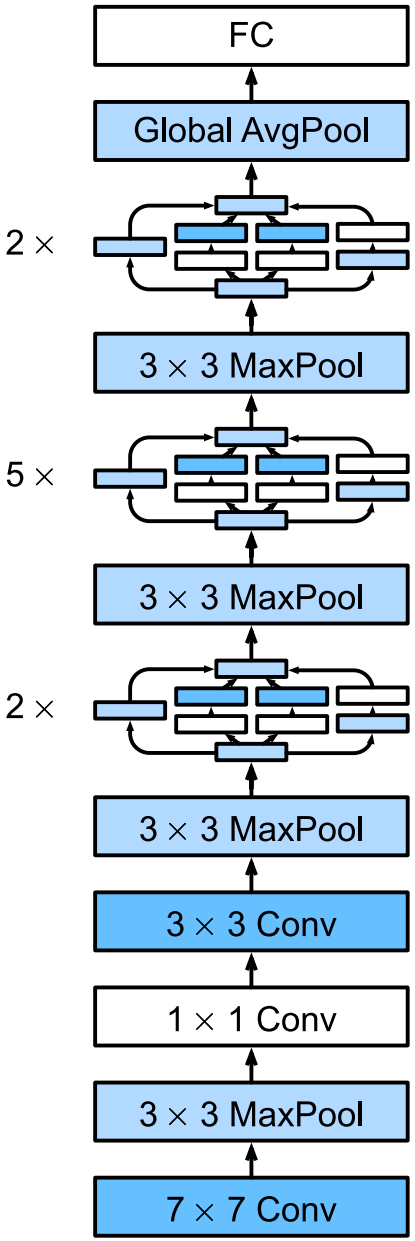
- Small filters, deeper networks
  - 8 layers -> 16 – 19 layers
  - 3 x 3 conv
  - 2 x 2 max pooling



GoogLeNet (Szegedy et al., 2014)

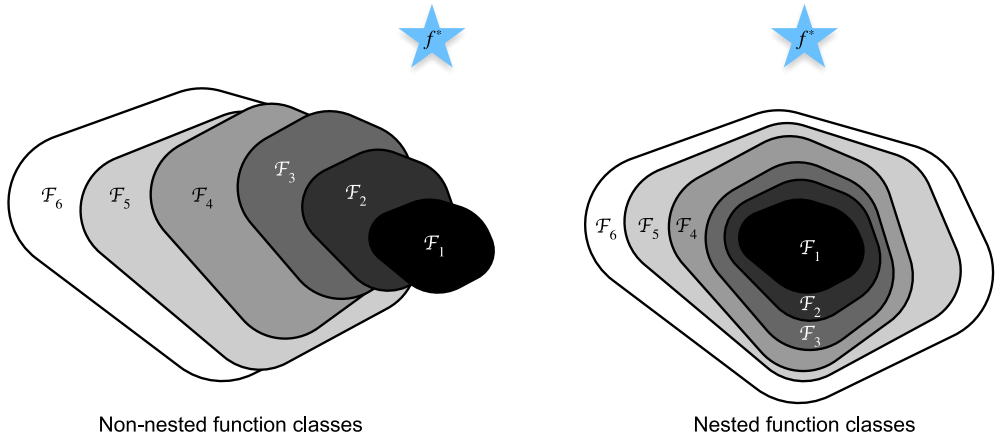
- Inception Blocks
  - 1 x 1 conv + 3 x 3 conv + 5 x 5 conv to extract information from different spatial sizes.
  - 1 x 1 conv on the input to reduce the number of channels.
  - Concatenation along the channel dimension.

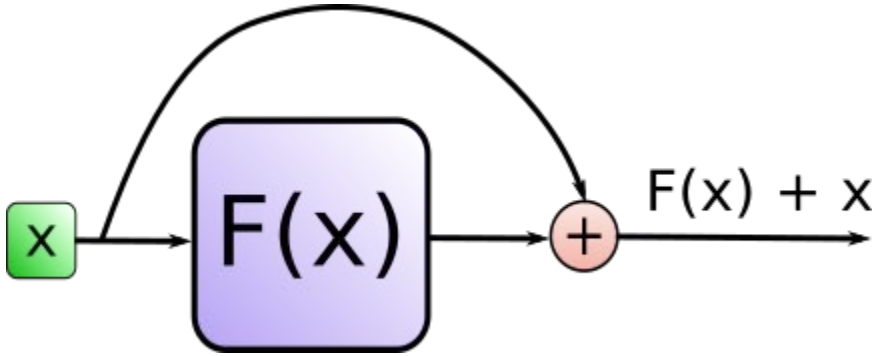




ResNet (He et al., 2016)

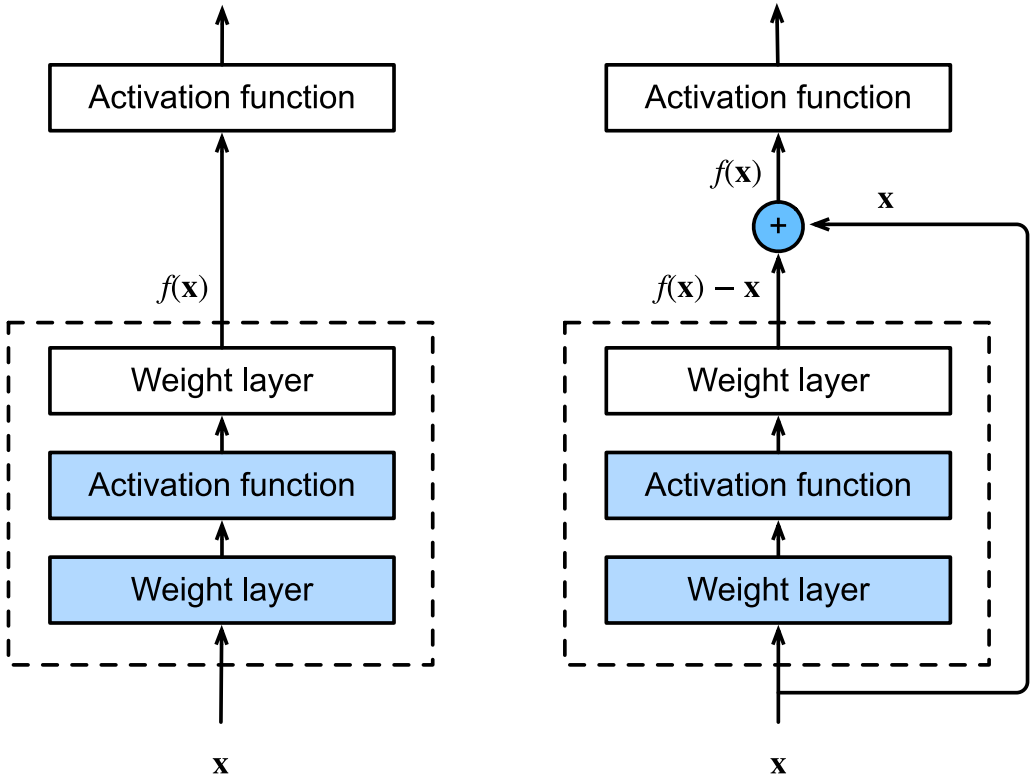
- Class of function $\mathcal{F} = \{f(\mathbf{x}; w)\}$, specify a network architecture.
  - e.g. $\mathcal{F}_1$ = VGG16, $\mathcal{F}_2$ = VGG19
- Assume that f^* is the "truth" function we want.
- We expect $\mathcal{F}_i \to \mathcal{F}_{i+1}$ is getting closer to f^*.
- If $\mathcal{F}_i \not\subseteq \mathcal{F}_{i+1}$ there is no guarantee that this should even happen.

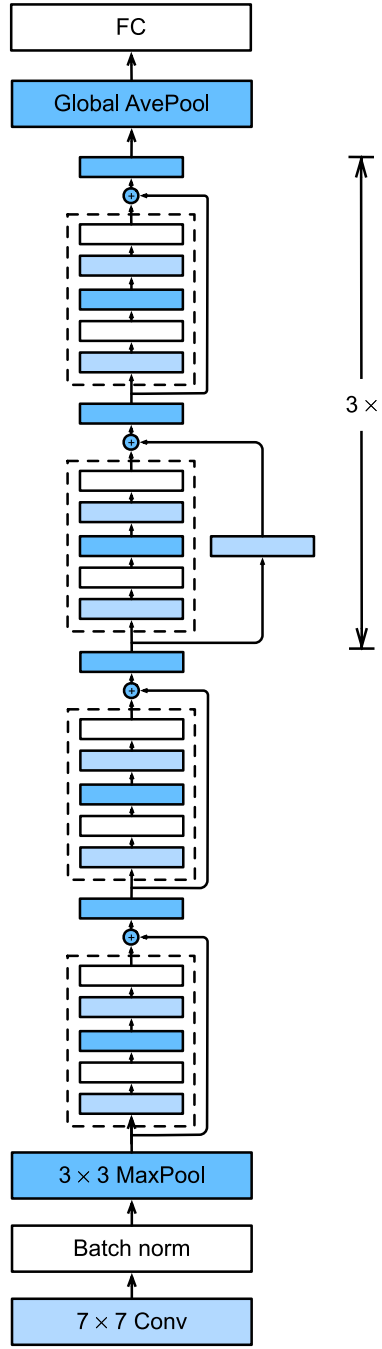




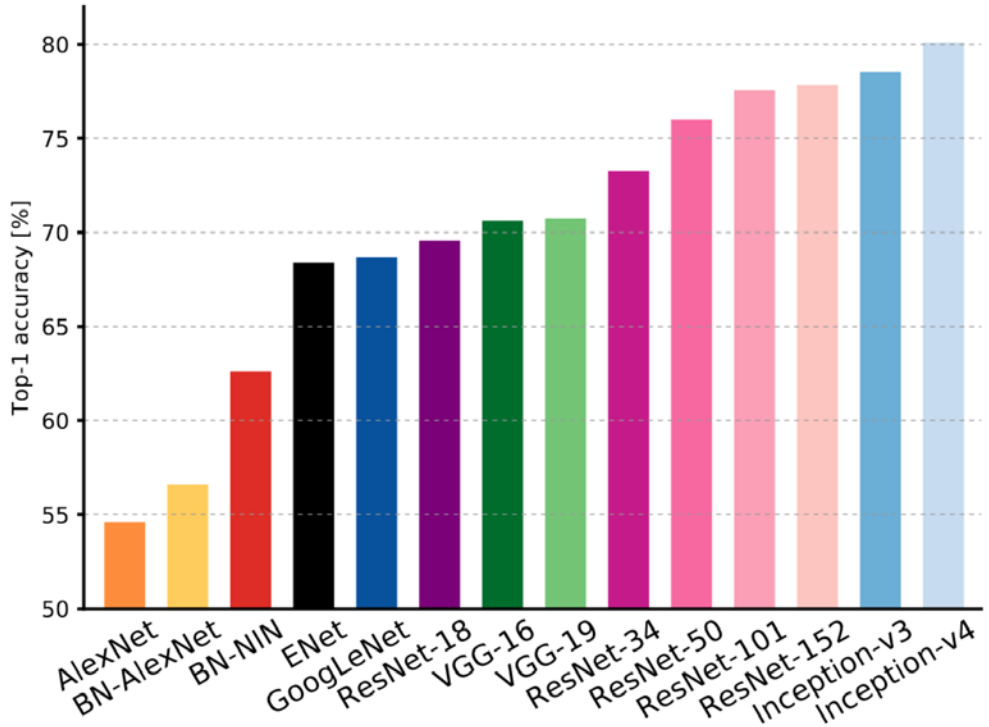
ResNet (He et al., 2016)

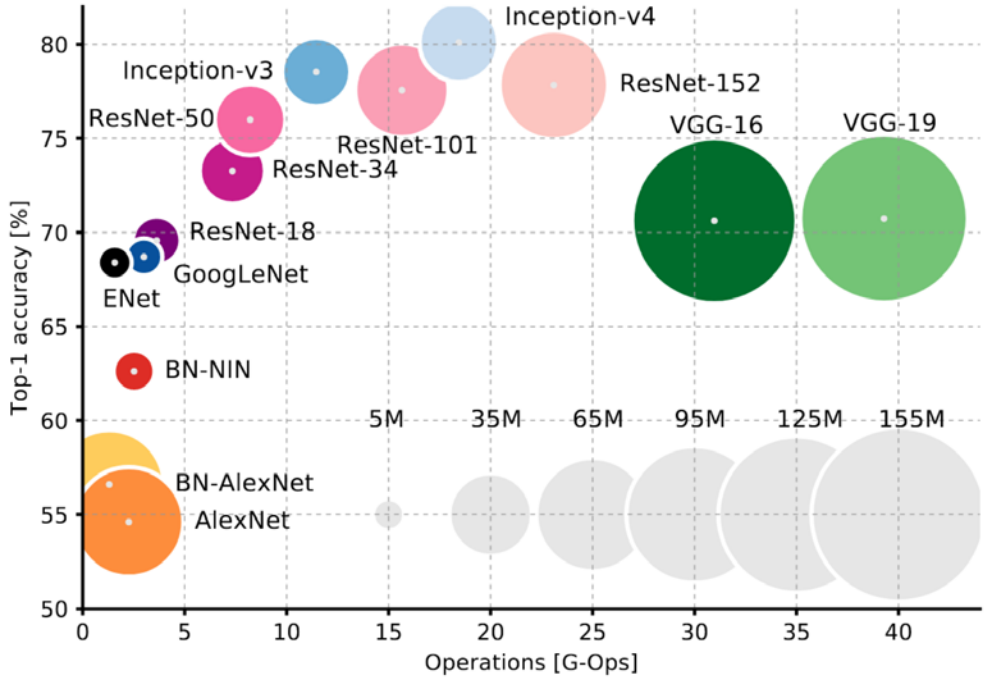
- Learn the residual mapping $f(\mathbf{x}) - \mathbf{x}$ instead.
- Ensure $\mathcal{F}_i \subseteq \mathcal{F}_{i+1}$ (let residual mapping = 0)





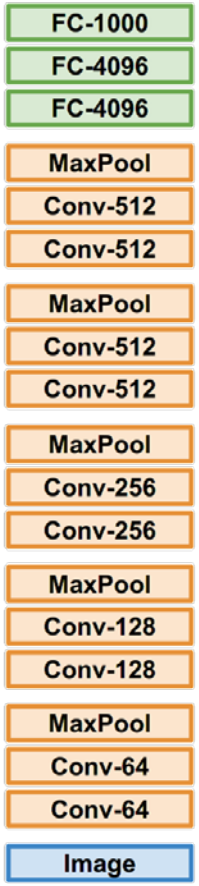
Models Comparison



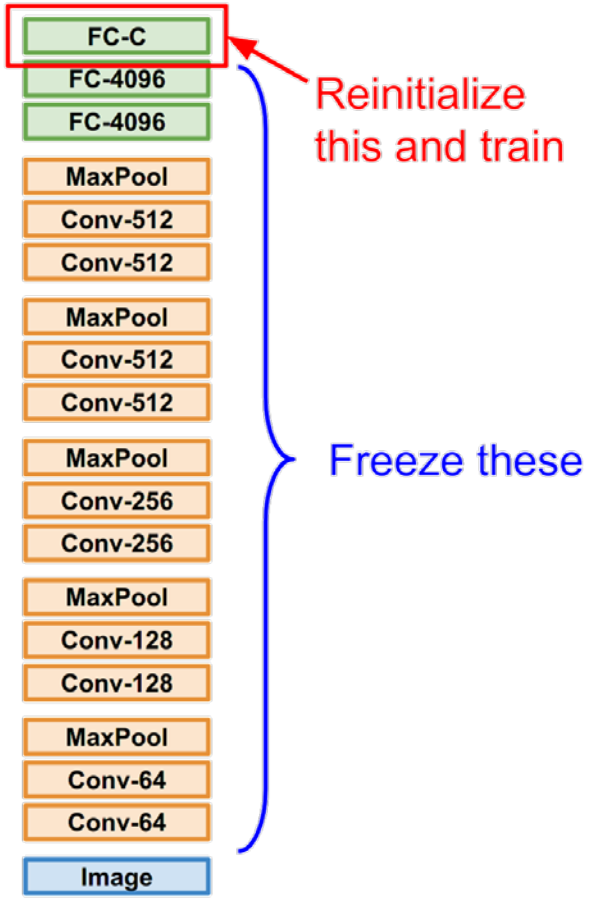


Transfer Learning

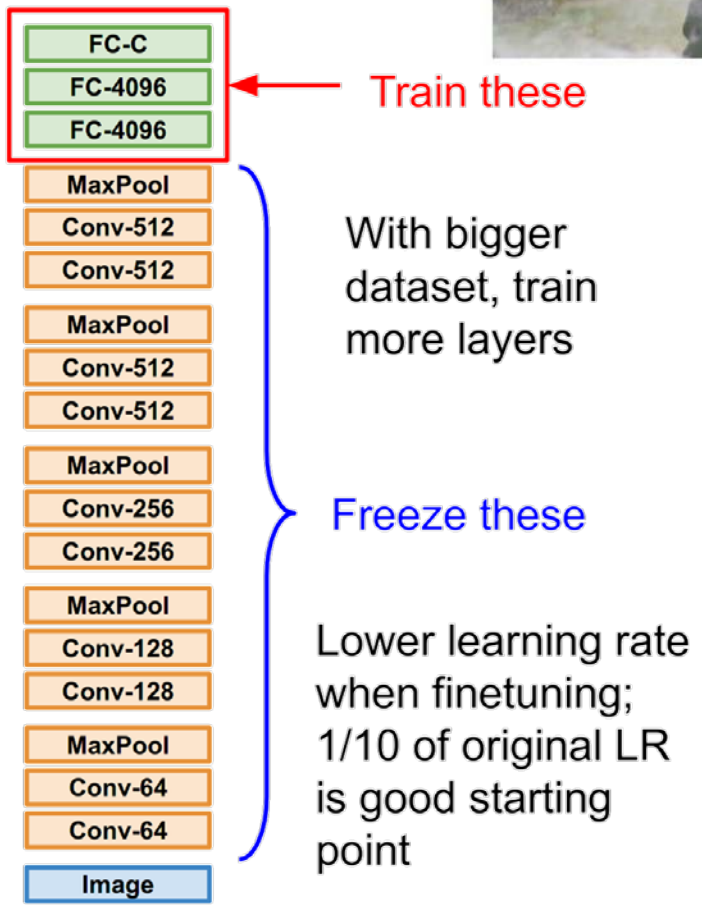
1. Train on Imagenet



2. Small Dataset (C classes)



3. Bigger dataset

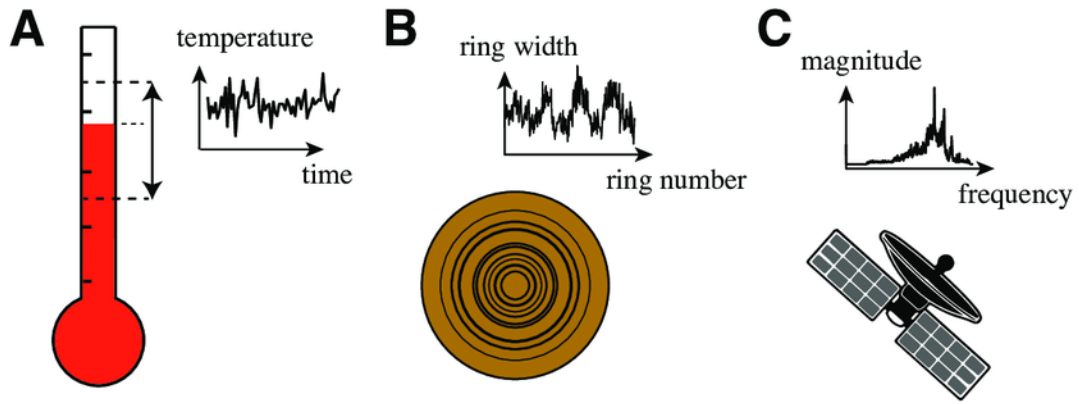


Any Questions?

RNN

Sequential Data

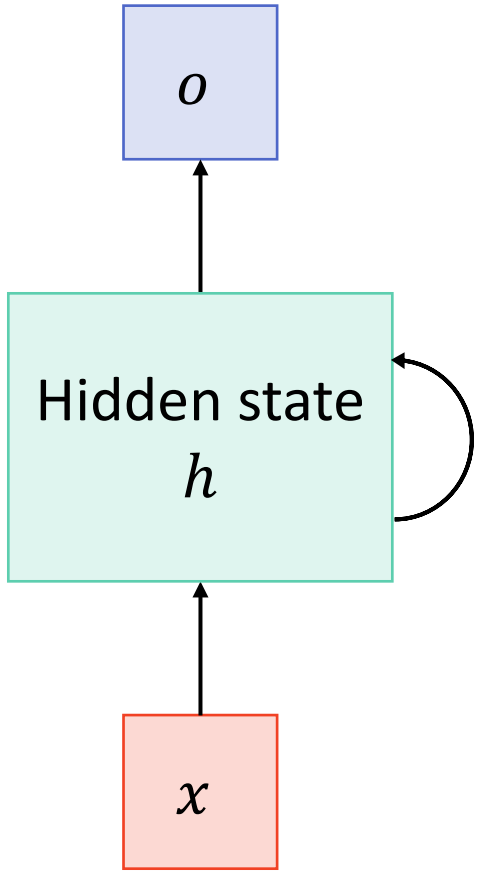
- Text
- Audio
- Stock quotes
- Video frames
- Motions and gestures
- Image as a sequence of pixels



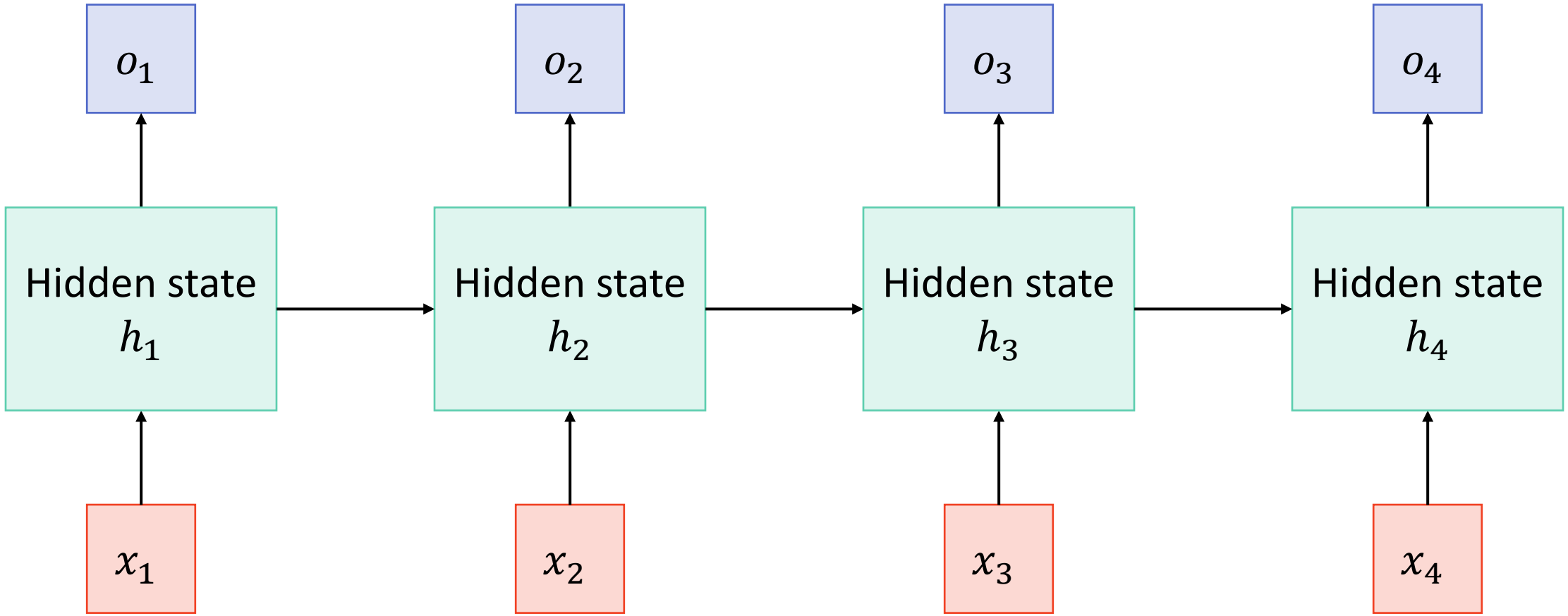
Recurrent Neural Network

- Hidden state “remembers” sequential input.
- For each step, update hidden state and determine output using recurrence formula:

$$h_t = f(h_{t-1}, x_t; w_h)$$
$$o_t = g(h_t; w_o)$$



Unfolding Calculation Graph



Input Output Relationship

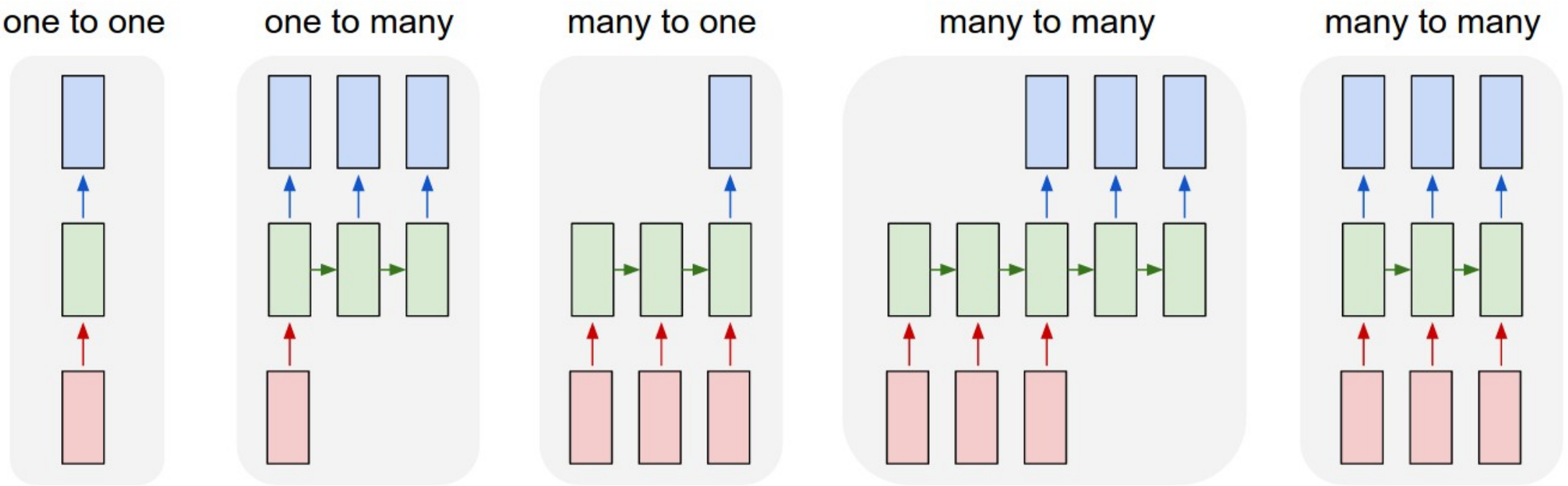


Image Captioning

Video Classification

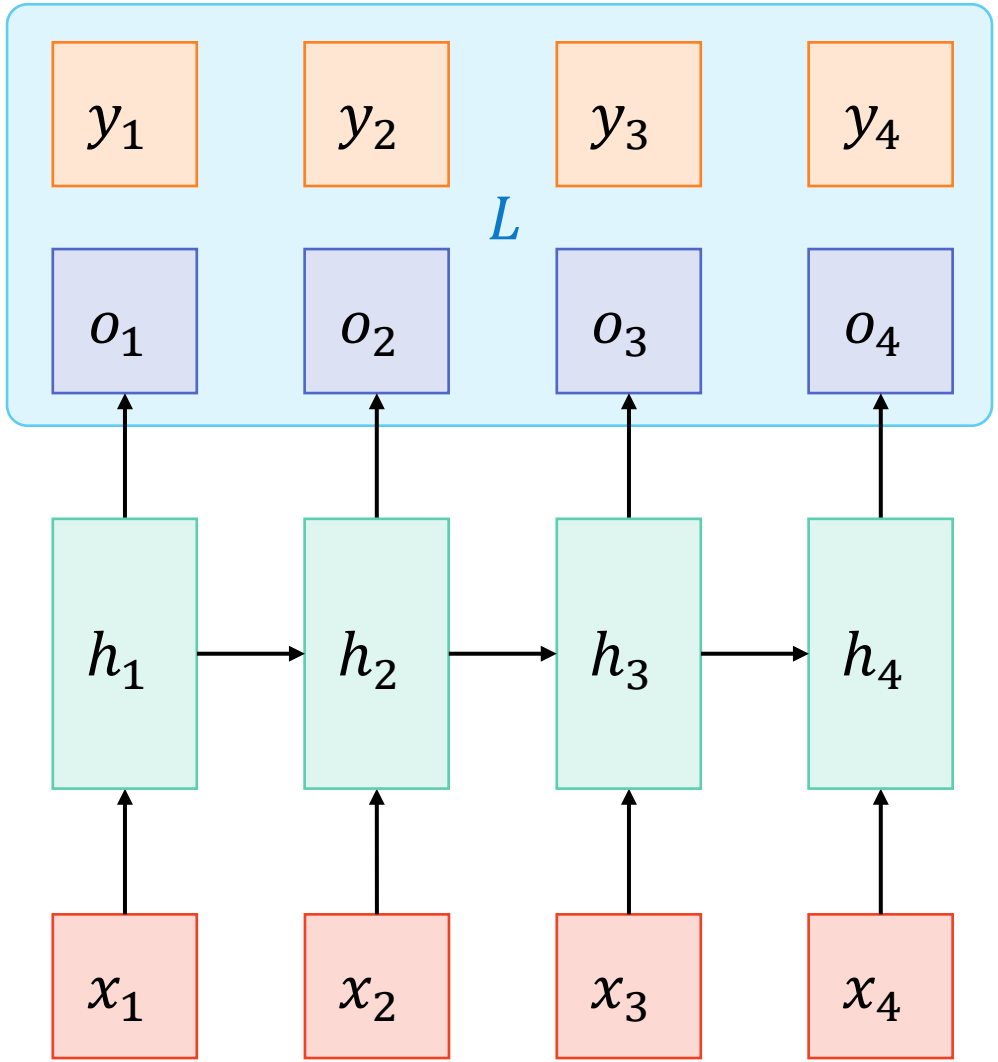
Video Captioning

Frame-level
Video Classification

Backpropagation Through Time

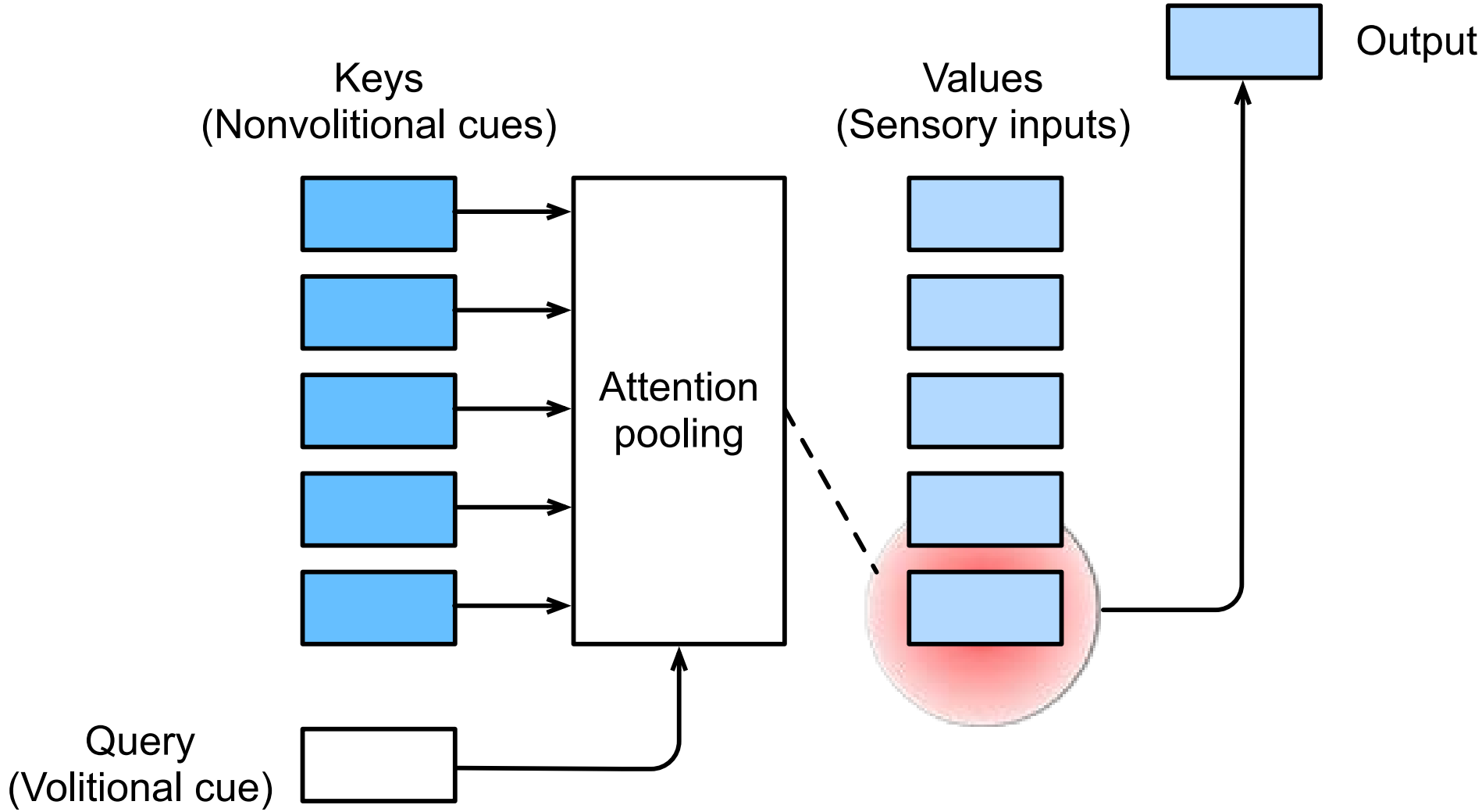
- $h_t = f(h_{t-1}, x_t; w_h)$
- $o_t = g(h_t; w_o)$

- $L(x_1, \dots, x_T, y_1, \dots, y_T, w_h, w_o) = \frac{1}{T}\sum_{t=1}^{T} l(y_t, o_t)$.
- $\frac{\partial L}{\partial w_h} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial w_h} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial o_t} \frac{\partial g(h_t, w_o)}{\partial h_t} \frac{\partial h_t}{\partial w_h}$



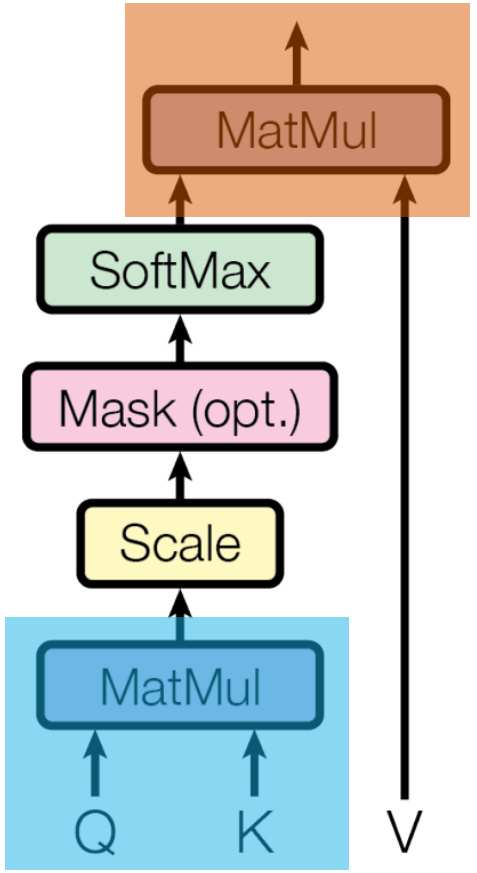
Attention

Attention Cues



Scaled Dot-Product Attention

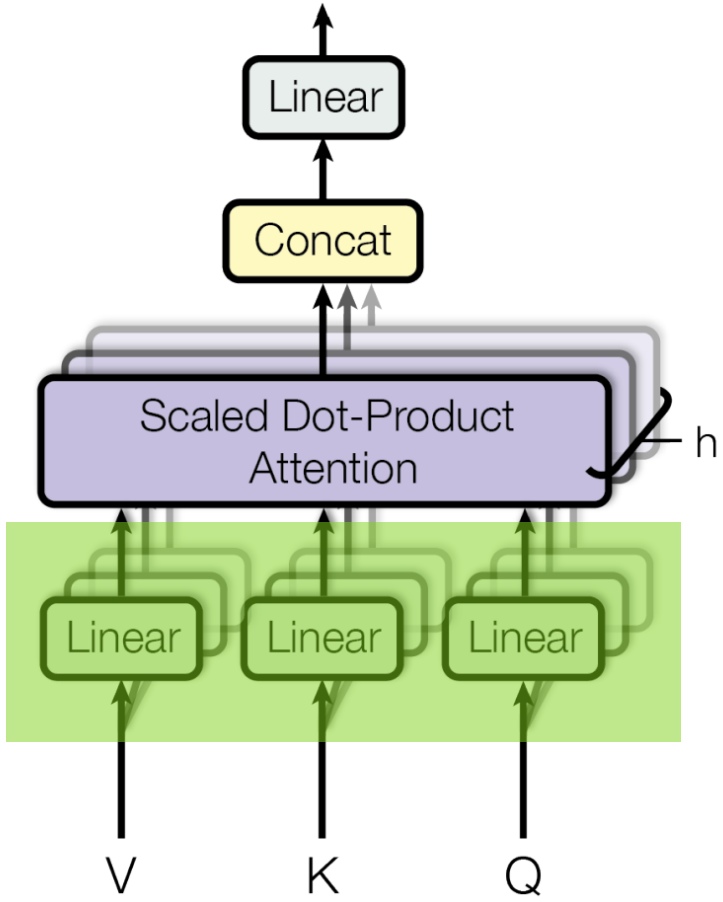
- $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$



Multi-Head Attention

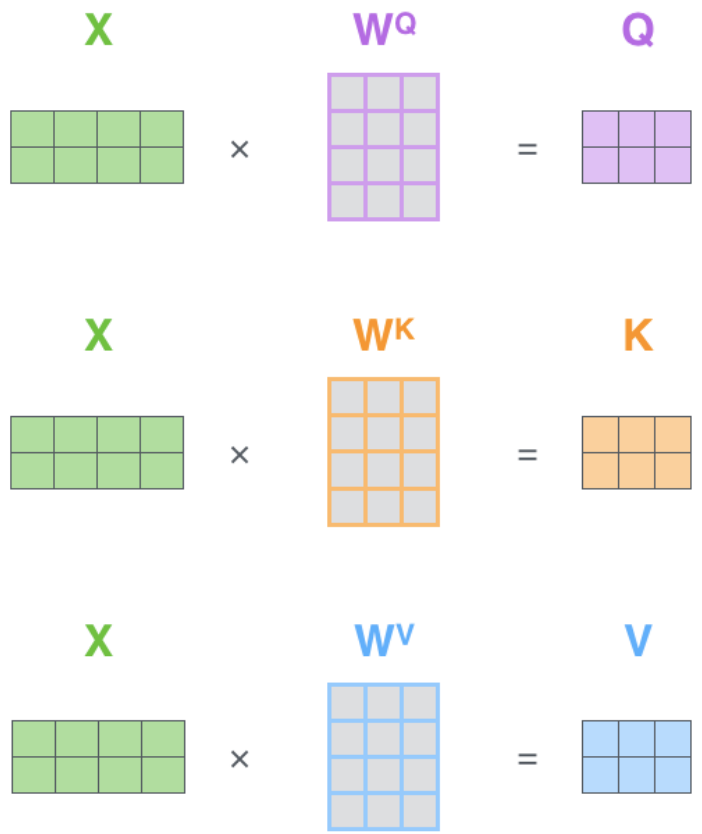
- $\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$
- Where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

- Linear projection for Q, K, V
  - Add more expressivity
- Different linear layer for different head
  - Let heads care about different things

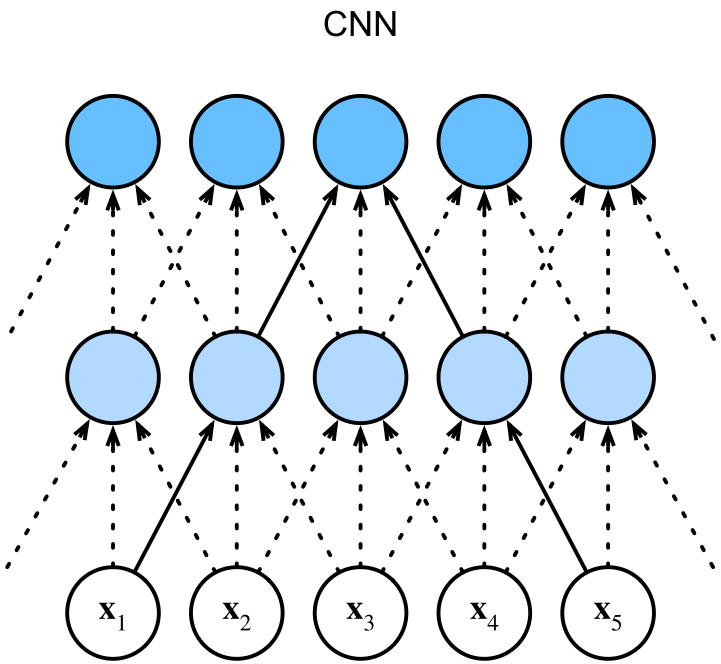


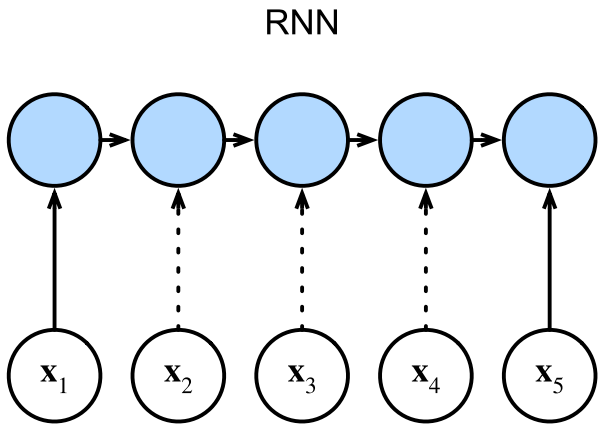
Self-Attention

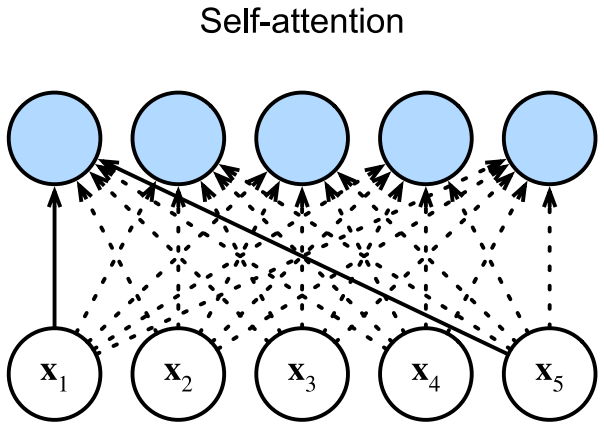
- V = K = Q = X
- $\text{Attention}(X, X, X)$
- Q: meaning behind?



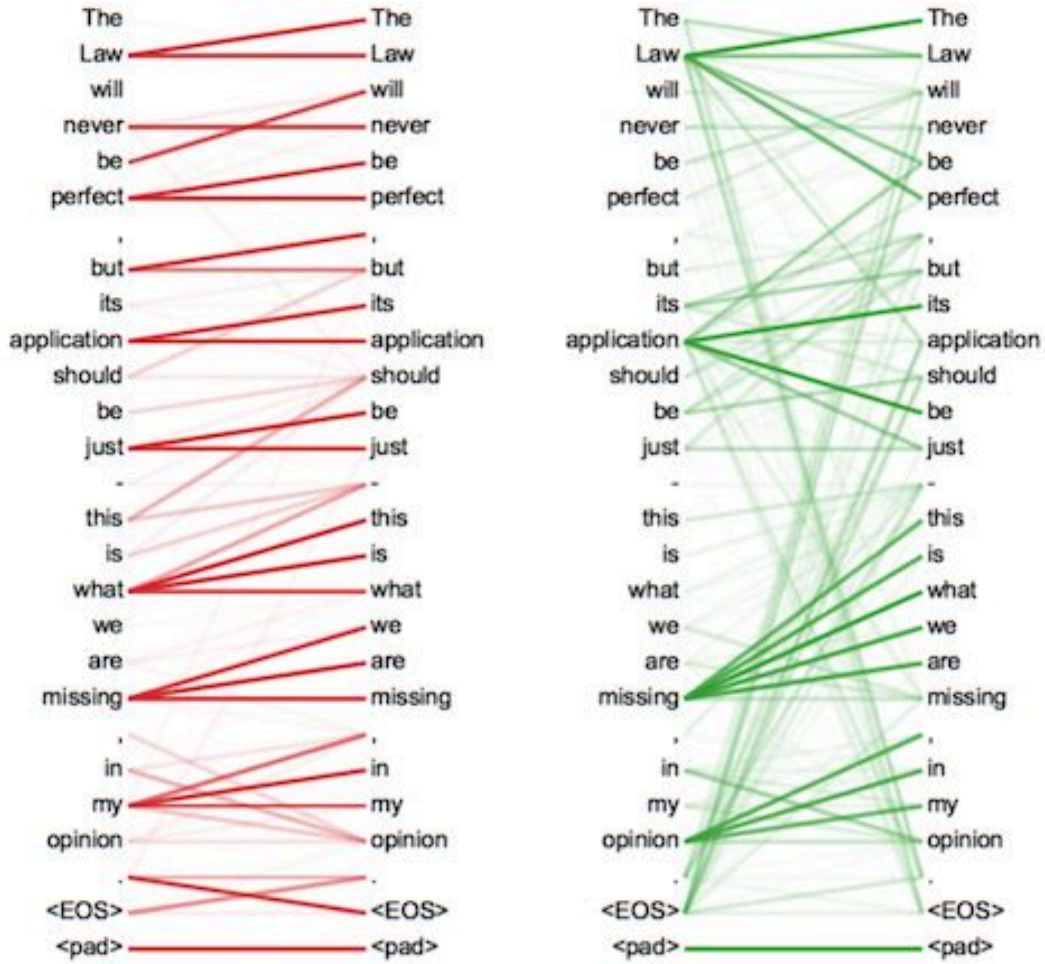
Comparing CNNs, RNNs, and Self-Attention



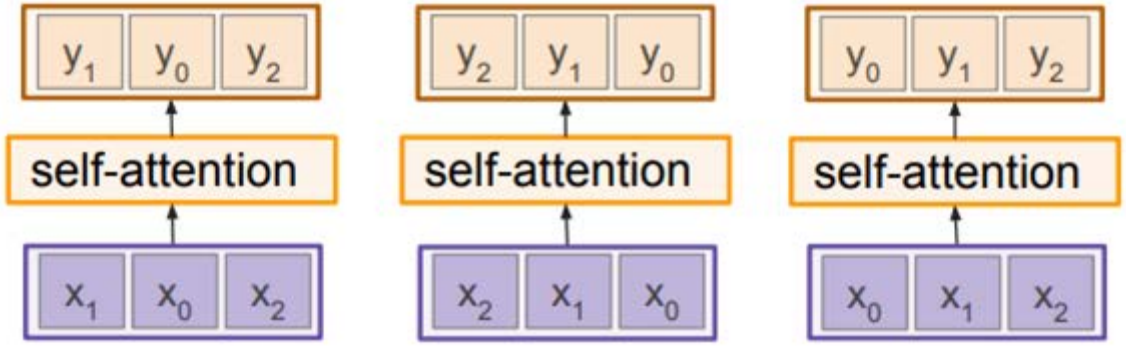




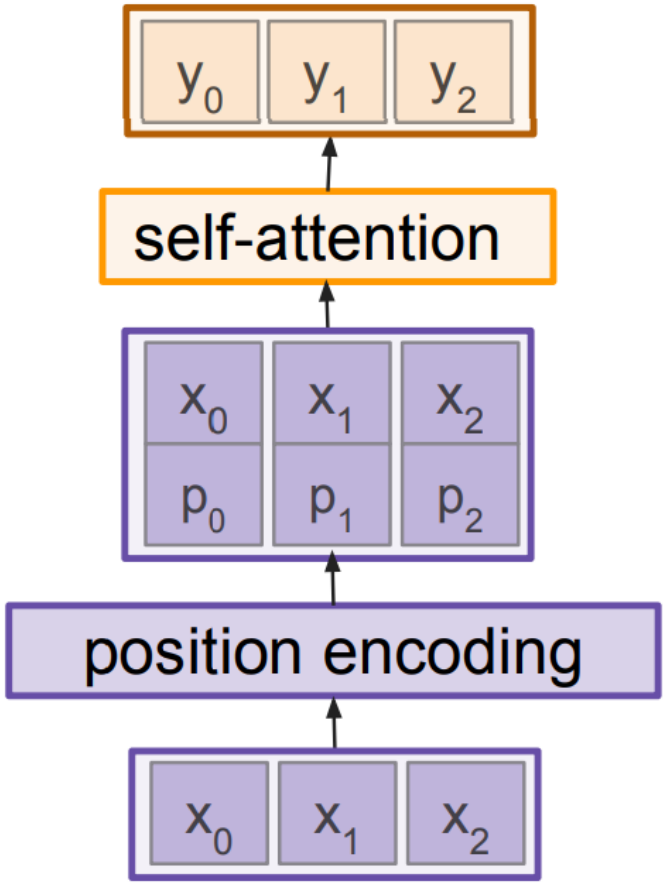
Visualize Self-Attention



Permutation Equivariant Problem in Self-Attention

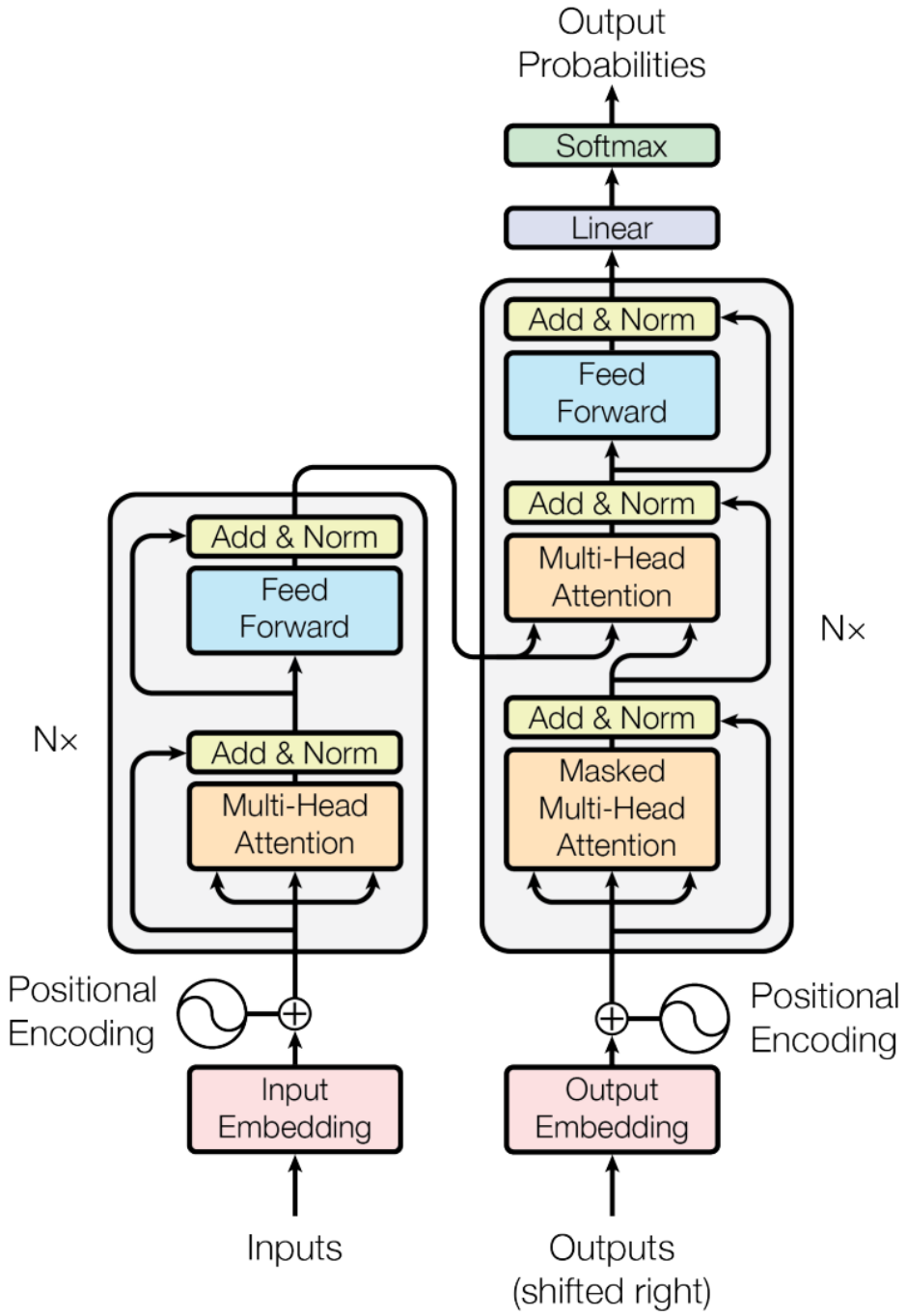


- Solution: Add position actively
- Position encoding
  - sin-cos
  - Learned encoding

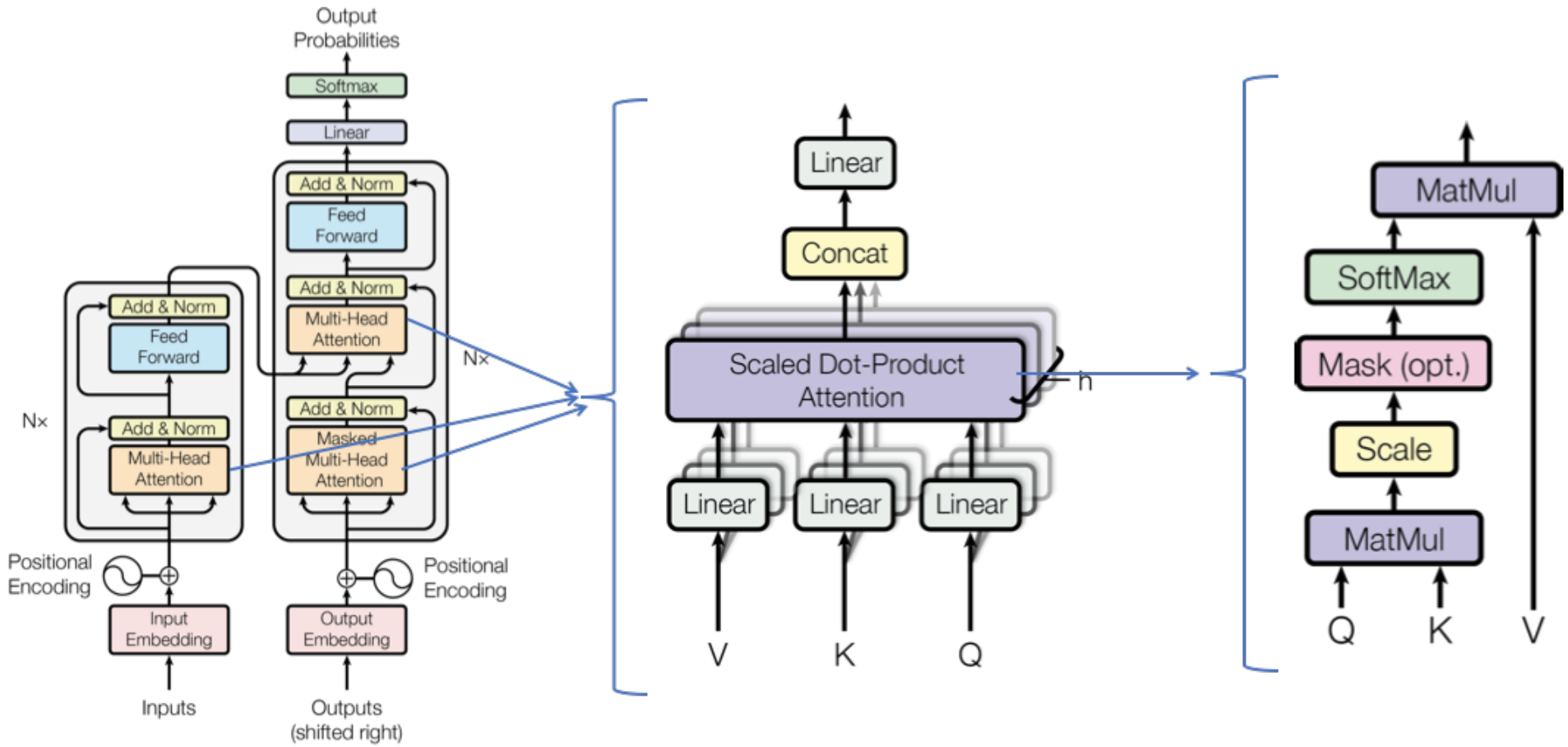


Transformer (Vaswani et al., 2017)

- Encoder-Decoder
- Masked MHA
  - Prevent vectors from looking at future vectors when training

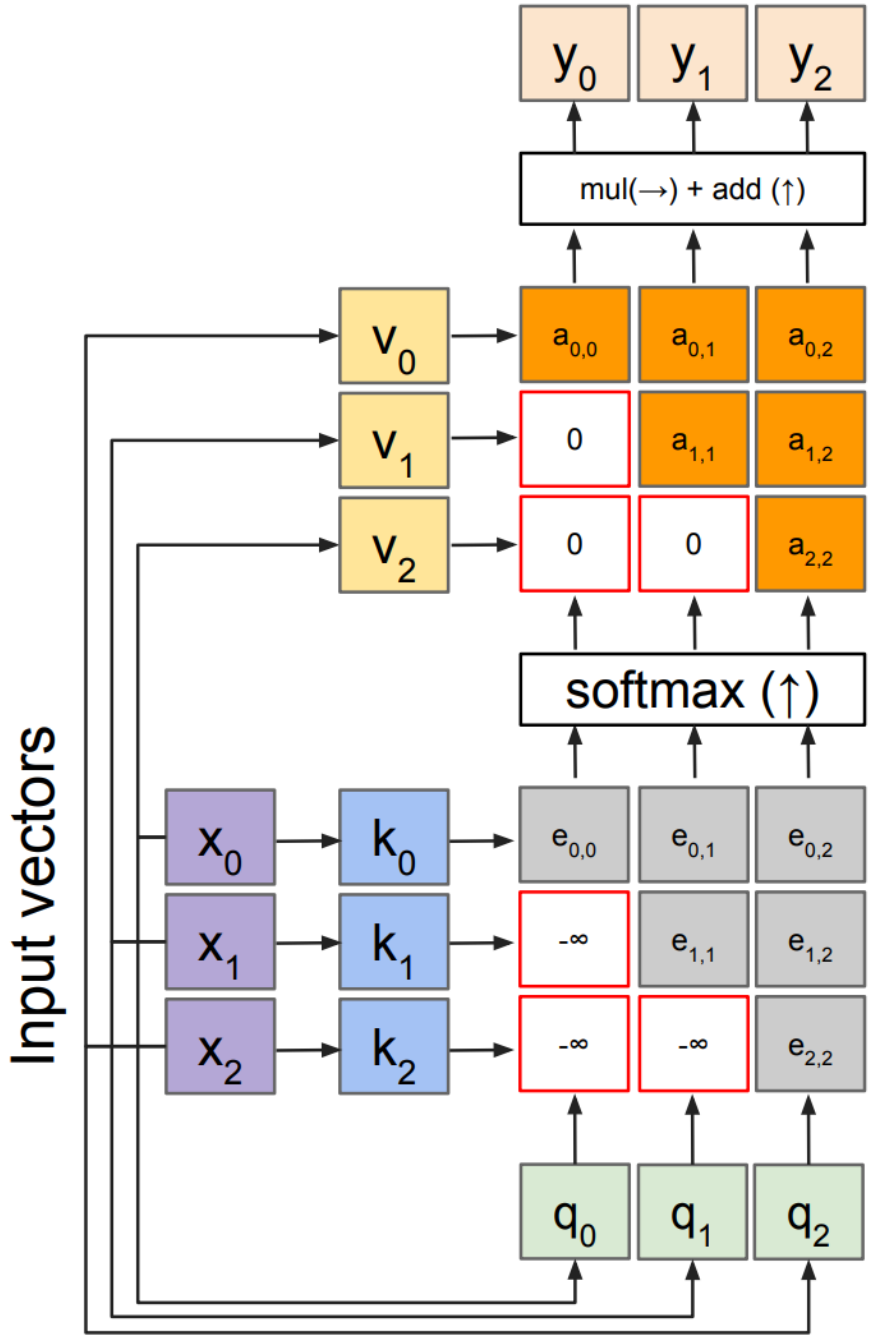


Transformer (Vaswani et al., 2017)



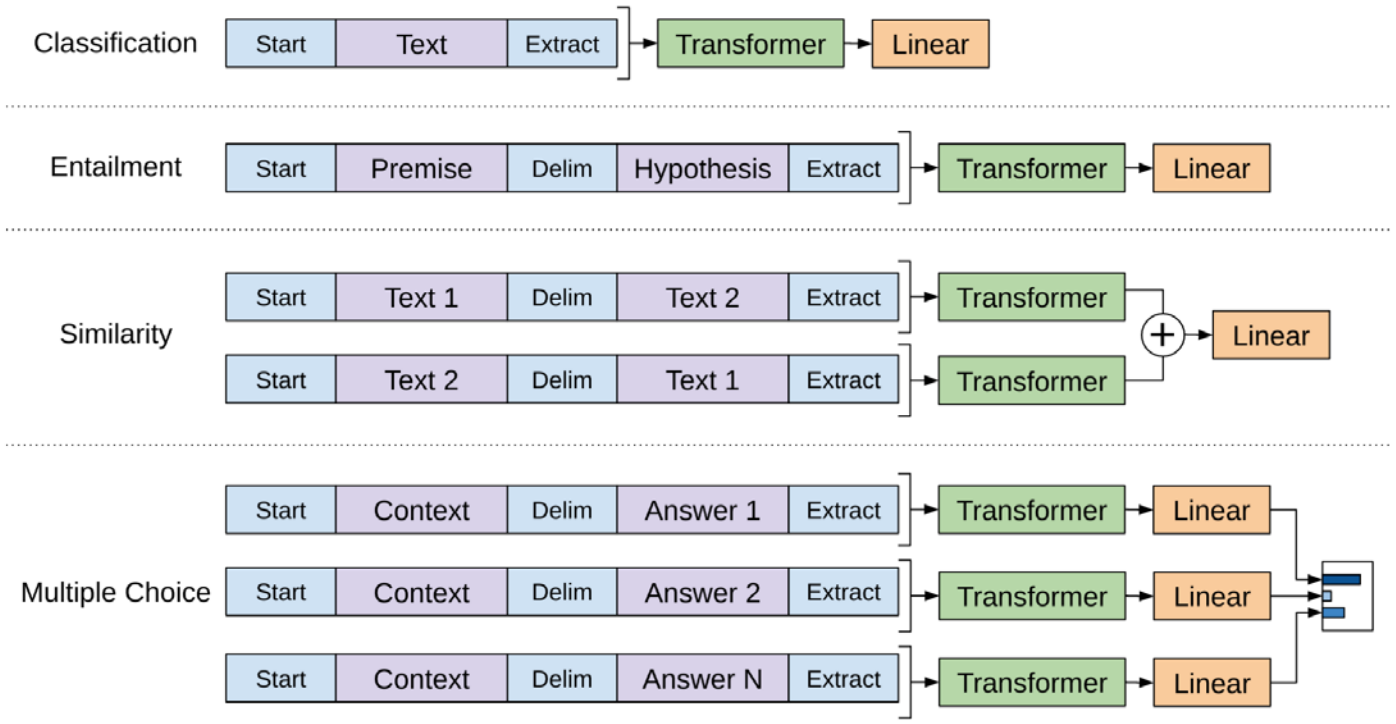
Transformer (Vaswani et al., 2017)

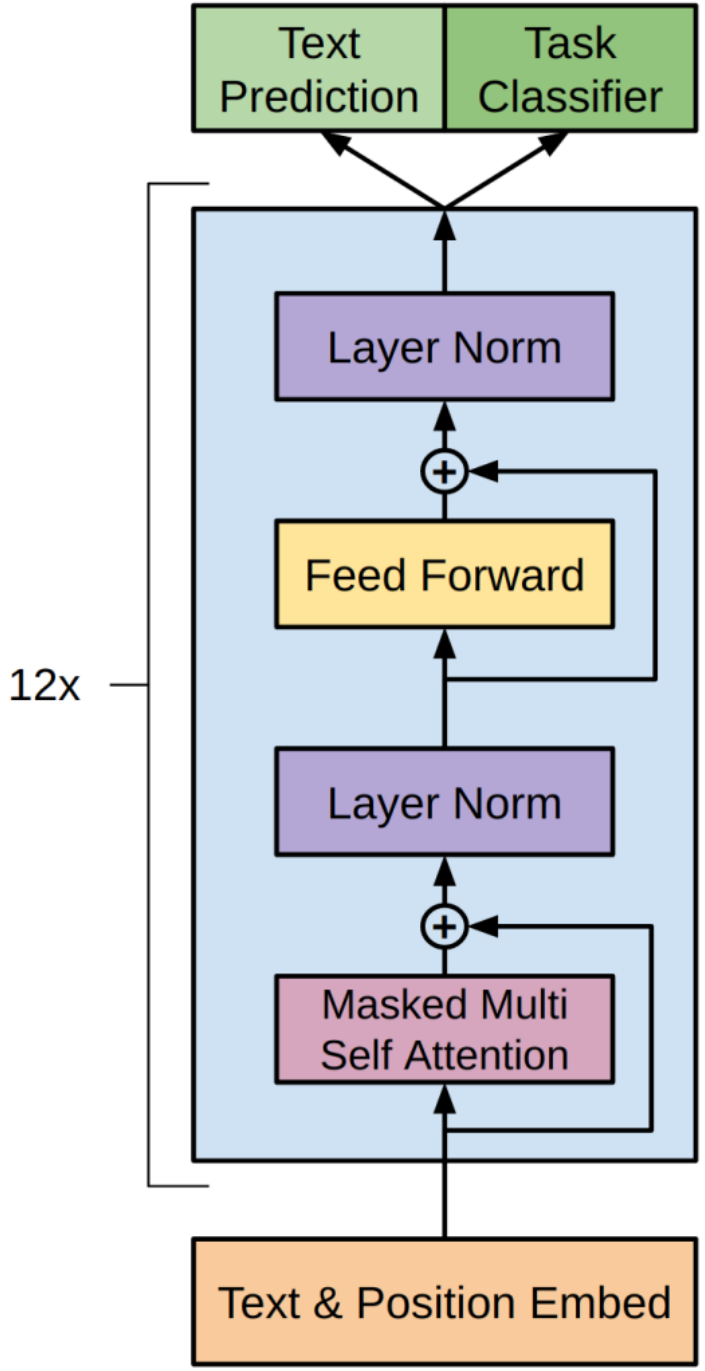
Masked MHA


Input vectors
x_0
x_1
x_2
k_0
k_1
k_2
v_0
v_1
v_2
q_0
q_1
q_2
$e_{0,0}$ $e_{0,1}$ $e_{0,2}$
-∞ $e_{1,1}$ $e_{1,2}$
-∞ -∞ $e_{2,2}$
softmax (↑)
0 0 $a_{2,2}$
0 $a_{1,1}$ $a_{1,2}$
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$
mul(→) + add (↑)
y_0 y_1 y_2

GPT (Radford et al., 2018)

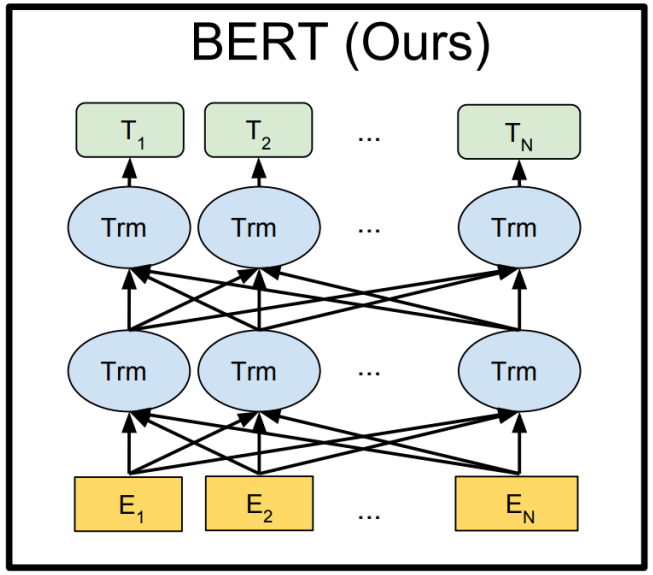
- Decoder only
- Generative Language Model
  - Pretrain the model by learning to predict the next token
- Fine-tune for downstream tasks

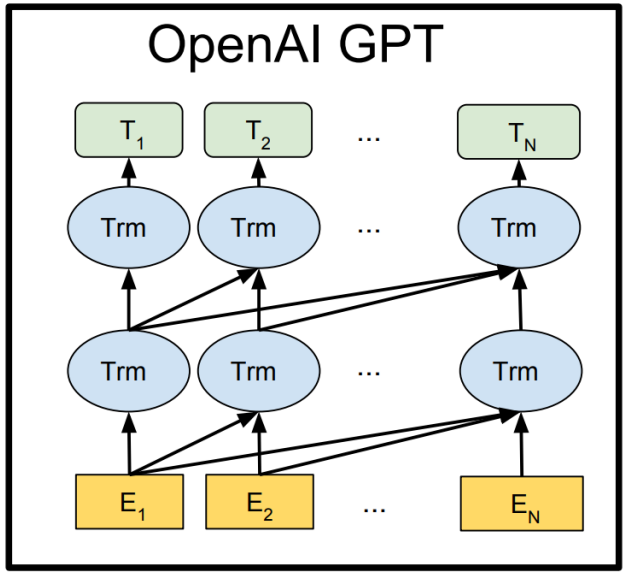


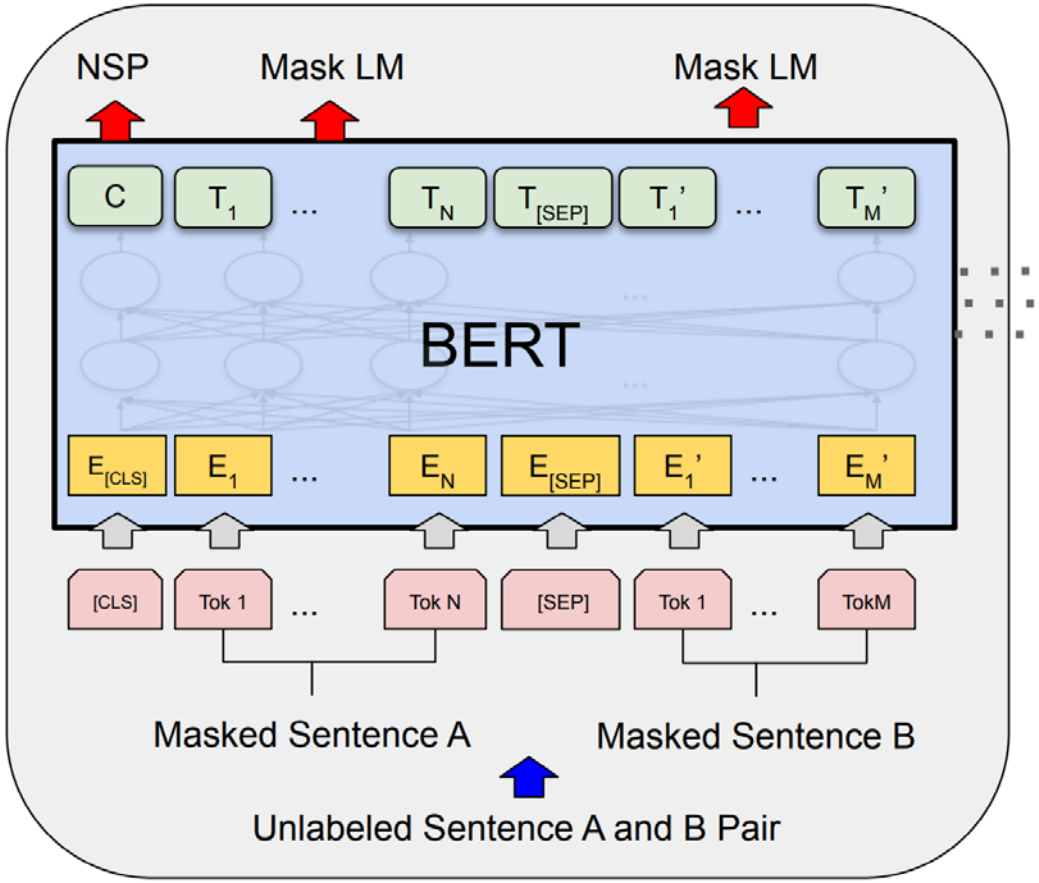


BERT (Devlin et al., 2018)

- Encoder only
- Masked Language Model
  - Pretrain by predicting the masked token
- No need for mask in attention







Llama (Hugo et al., 2023)

- Decoder only
- QK encoded with RoPE
- RMSNorm vs LayerNorm
- SwiGLU vs ReLU

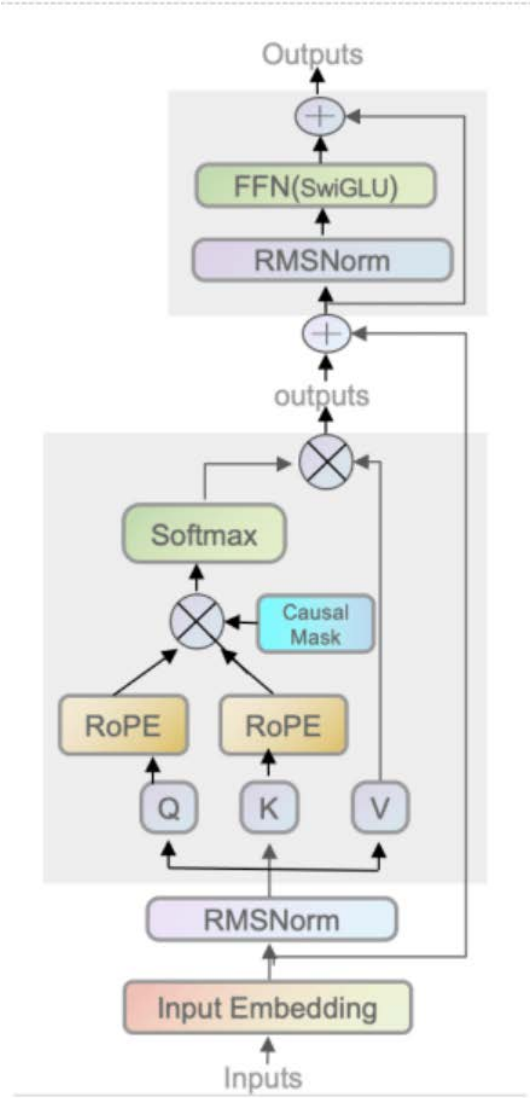


图 4: Llama 模型架构

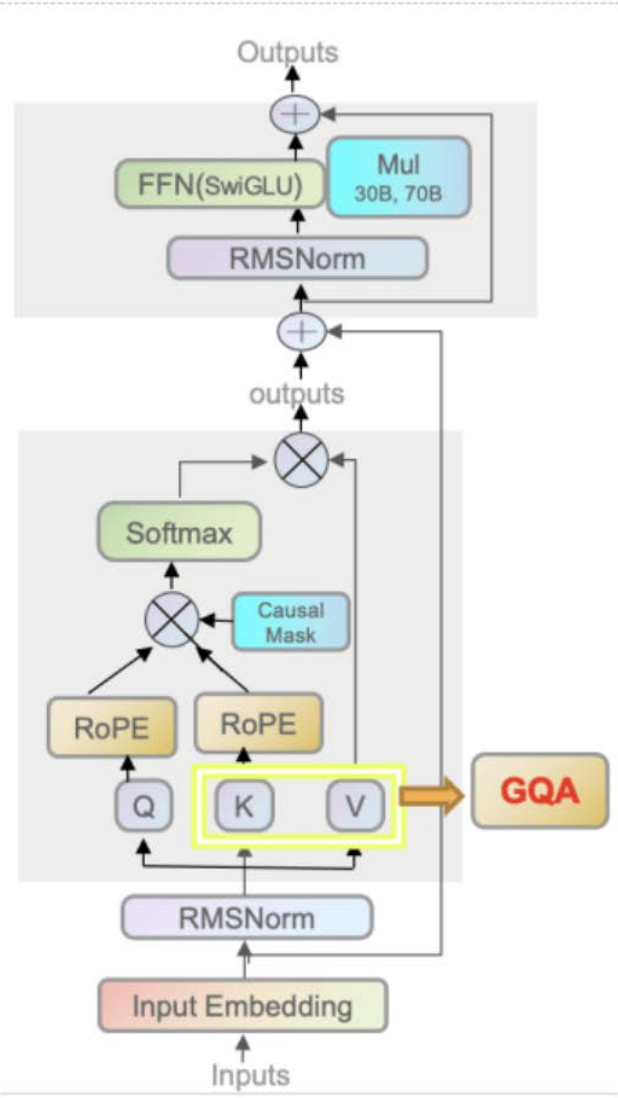
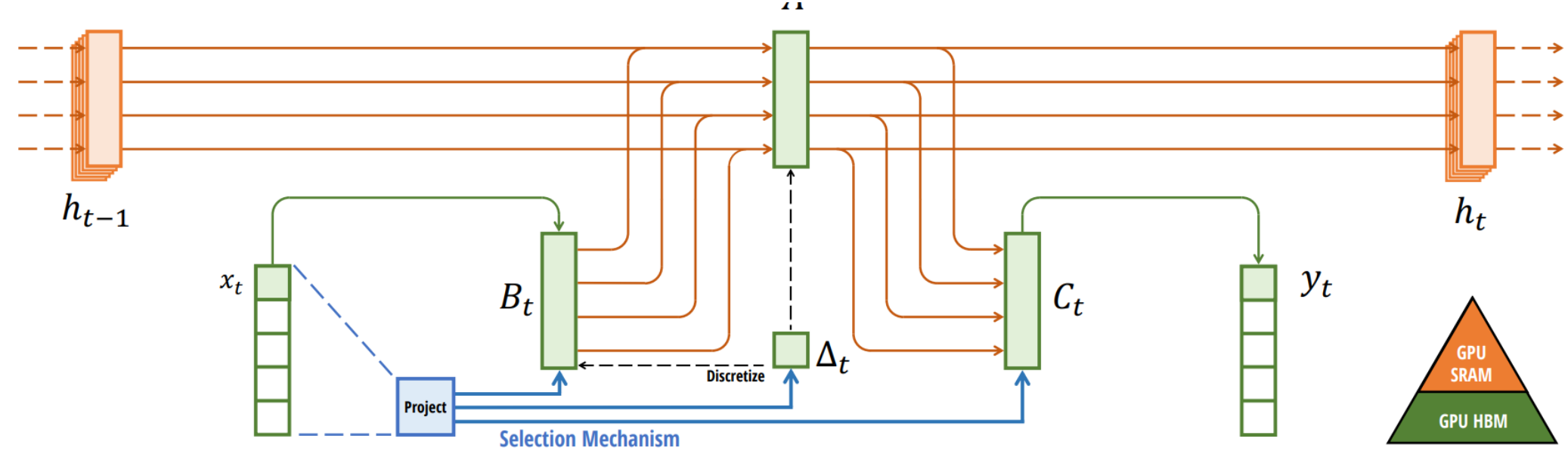


图 5: Llama 2 模型架构

Mamba (Albert et al., 2023)

- More Efficient Model Architecture
- Based on SSM: x(t) -> y(t) with h(t)



Any Questions?

Deep learning hardware

CPU vs GPU

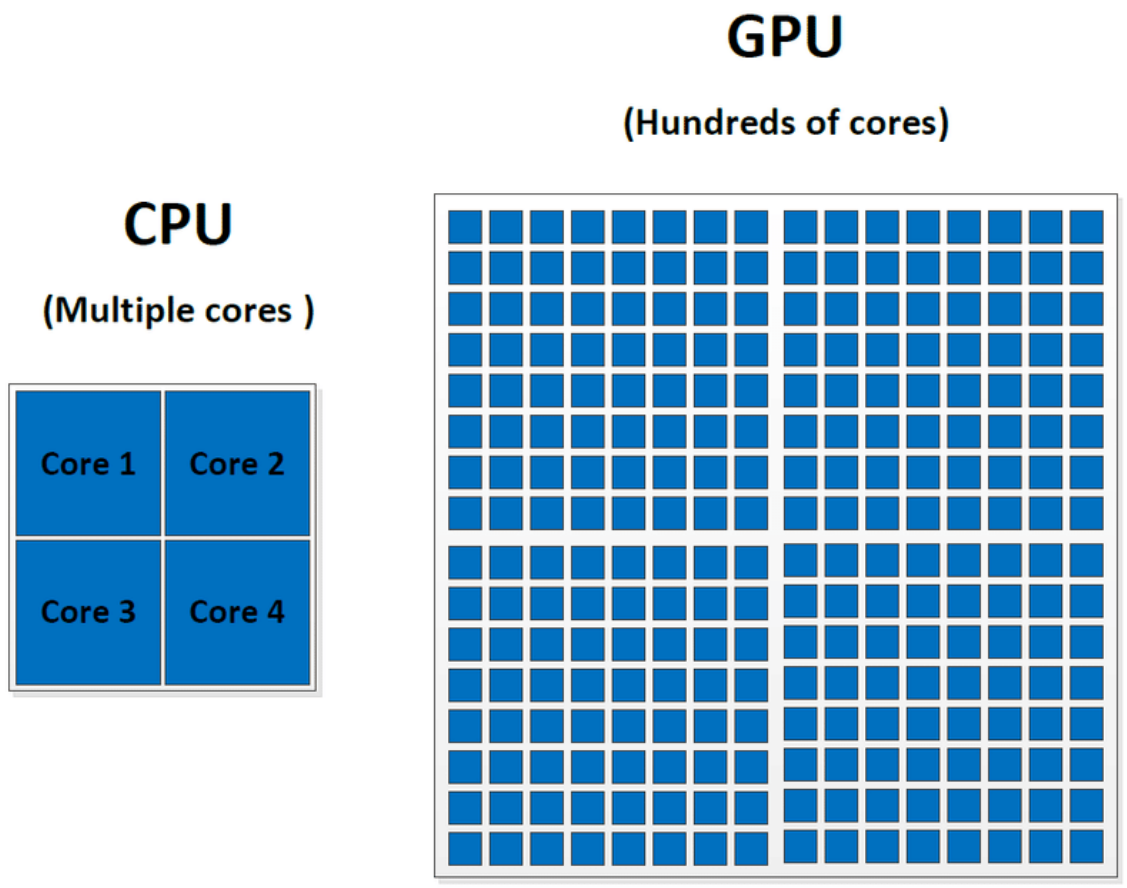
	Cores	Clock Speed	Memory	Price	Compute Capability
CPU Intel® Xeon® Platinum 8558 Processor	48	2.1GHz	System RAM	$4,650	6.45TFlops(FP32)
GPU NVIDIA RTX 4090	16384 Cuda Cores 512 Tensor Cores	2.2GHz	24GB GDDR6X	$1,600	82.5 TFlops(FP32)
GPU NVIDIA Hopper H100 PCIe	14592 Cuda Cores 456 Tensor Cores	1.6GHz	80GB HBM2e	$25,000	756TFlops(FP32 TensorCore)
TPU Google Cloud TPU v5e	4 Matrix Units(MXU) per core	?	16 GBHBM2	$1.2(per hour)	197TFlops(BF16)

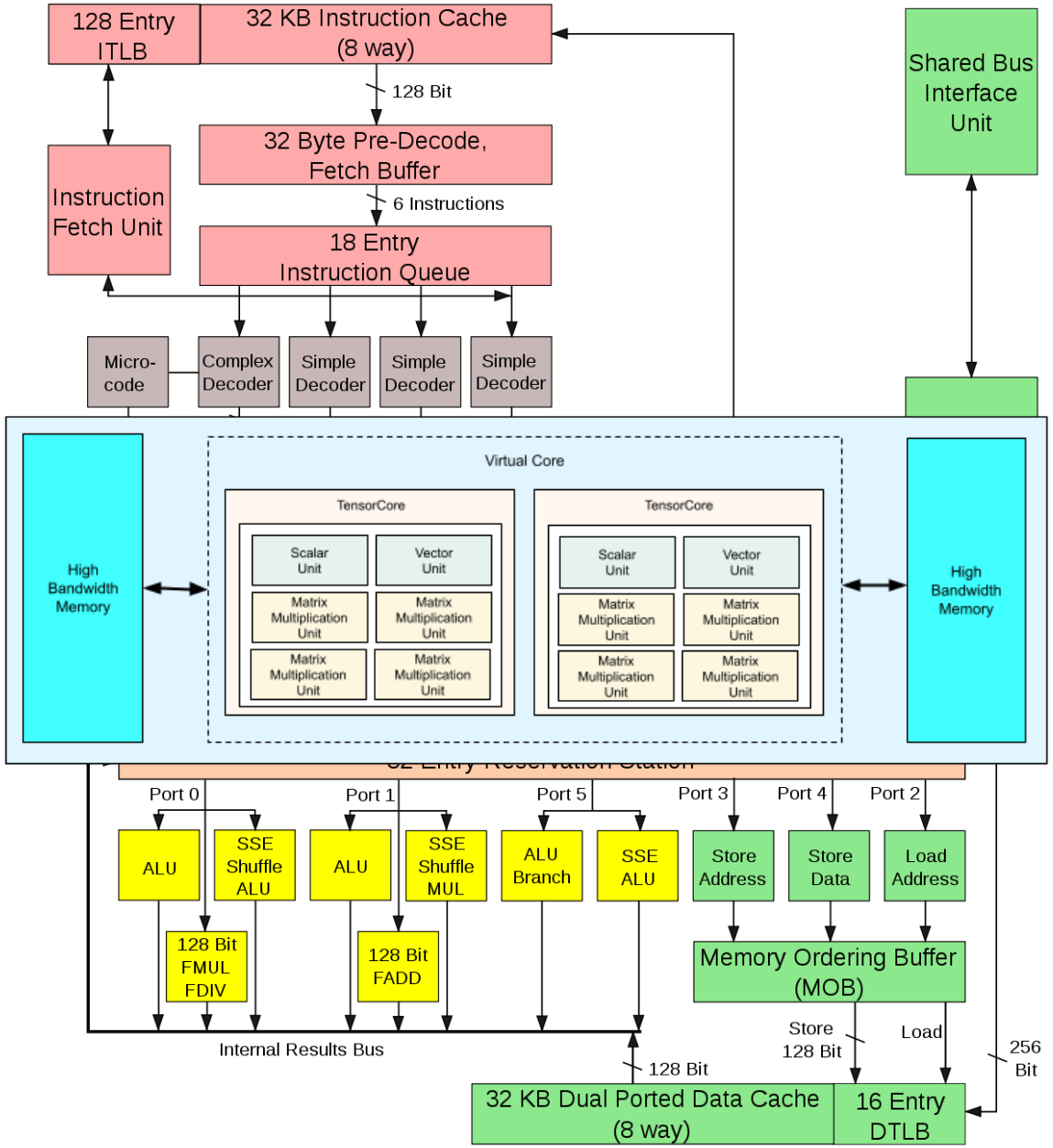
CPU: Fewer cores, but each core is much faster and much more capable; great at sequential tasks

GPU: More cores, but each core is much slower and "dumber"; great for parallel tasks

TPU: Specialized hardware for deep learning

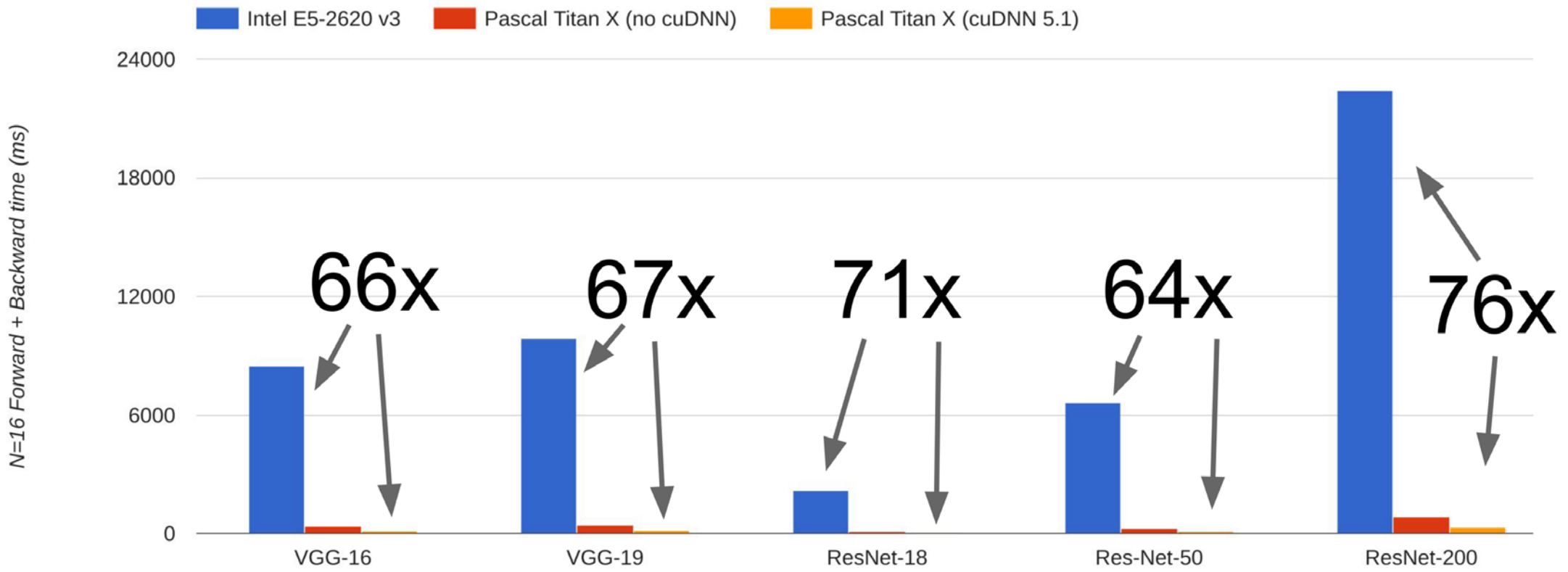
CPU vs GPU


GPU
(Hundreds of cores)
CPU
(Multiple cores)
Core 1
Core 2
Core 3
Core 4

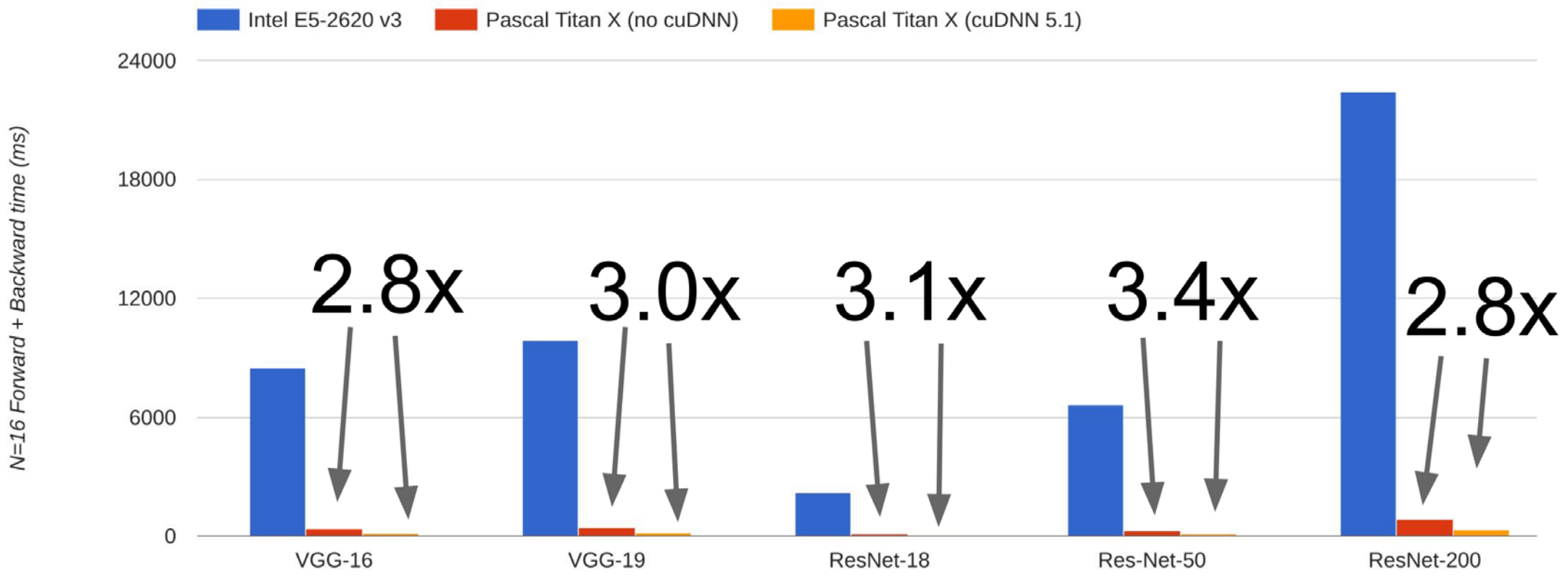


Intel Core 2 Architecture

CPU vs GPU


Intel E5-2620 v3
Pascal Titan X (no cuDNN)
Pascal Titan X (cuDNN 5.1)
N=16 Forward + Backward time (ms)
24000
18000
12000
6000
0
66x
67x
71x
64x
76x
VGG-16
VGG-19
ResNet-18
Res-Net-50
ResNet-200

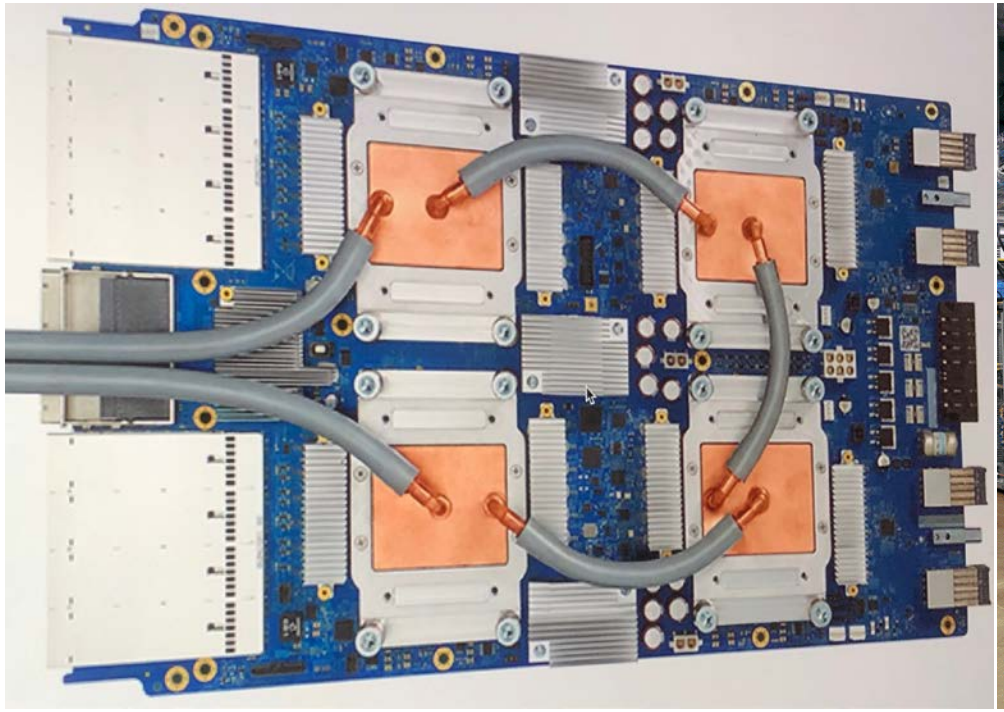
Plain CUDA vs cuDNN



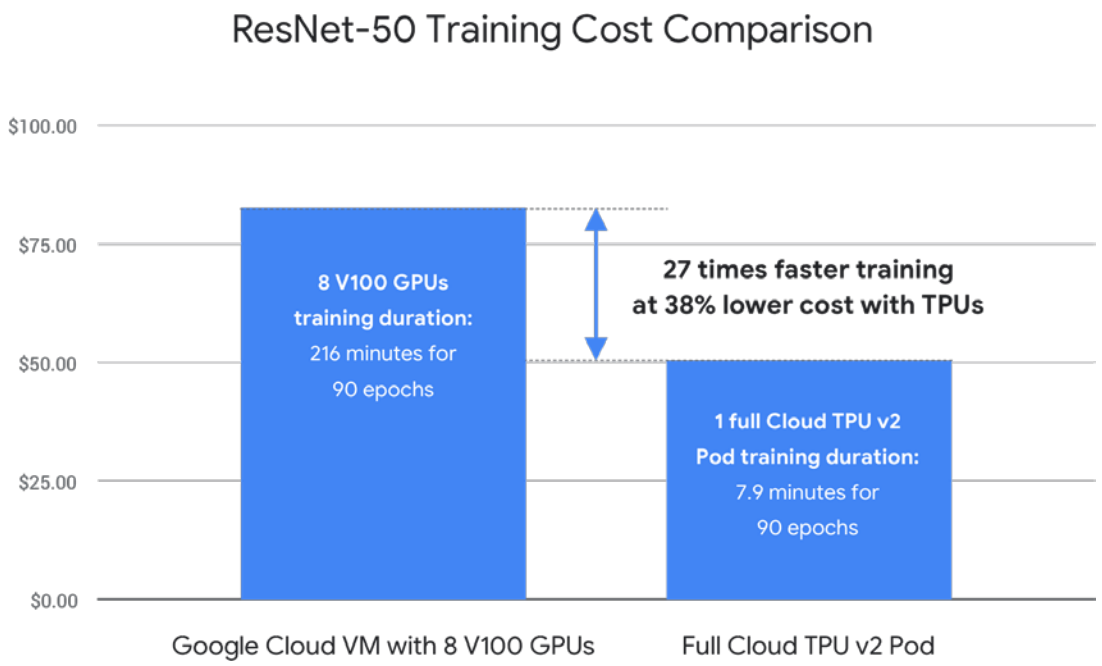
Programming GPUs

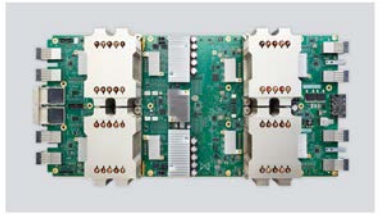
- CUDA
  - Covered in previous sessions
  - Optimized APIs: cuBLAS, cuFFT, cuDNN, etc
- OpenCL
  - Similar to CUDA, but runs on almost anything
  - Usually slower on NVIDIA hardware (30% according to Fang et al.)
- AMD ROCm / HIP
- Compute Shader
  - E.g. HLSL
  - jgbit/vuda

Tensor Processing Unit (TPU)



Tensor Processing Unit (TPU)

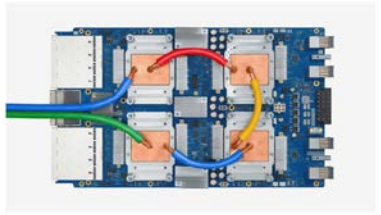




Cloud TPU v2

180 teraflops

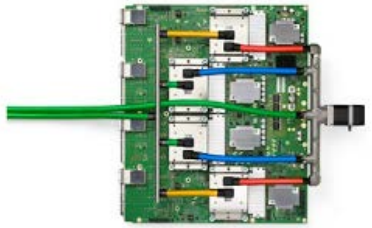
64 GB High Bandwidth Memory (HBM)



Cloud TPU v3

420 teraflops

128 GB HBM



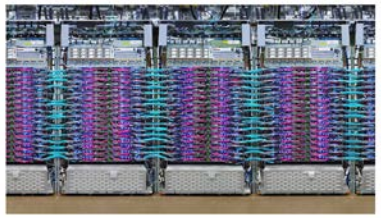
TPU v4

Cloud TPU v2 Pod

11.5 petaflops

4 TB HBM

2-D toroidal mesh network



Cloud TPU v3 Pod

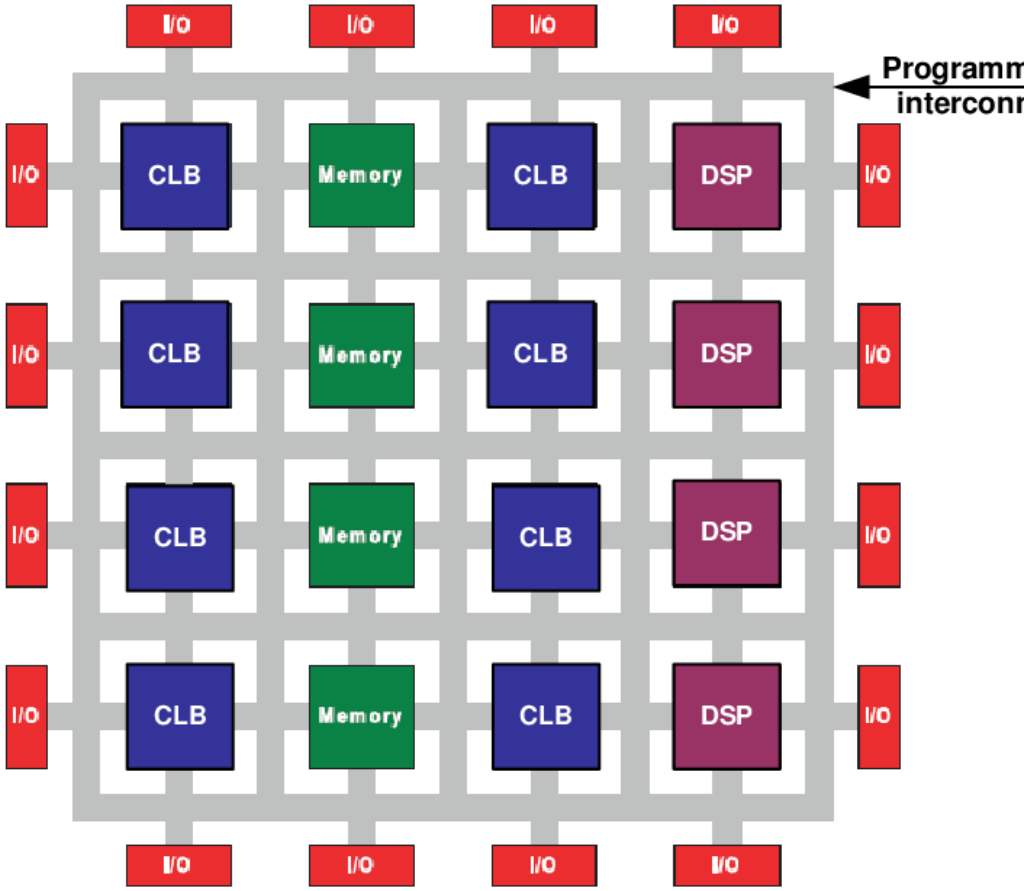
100+ petaflops

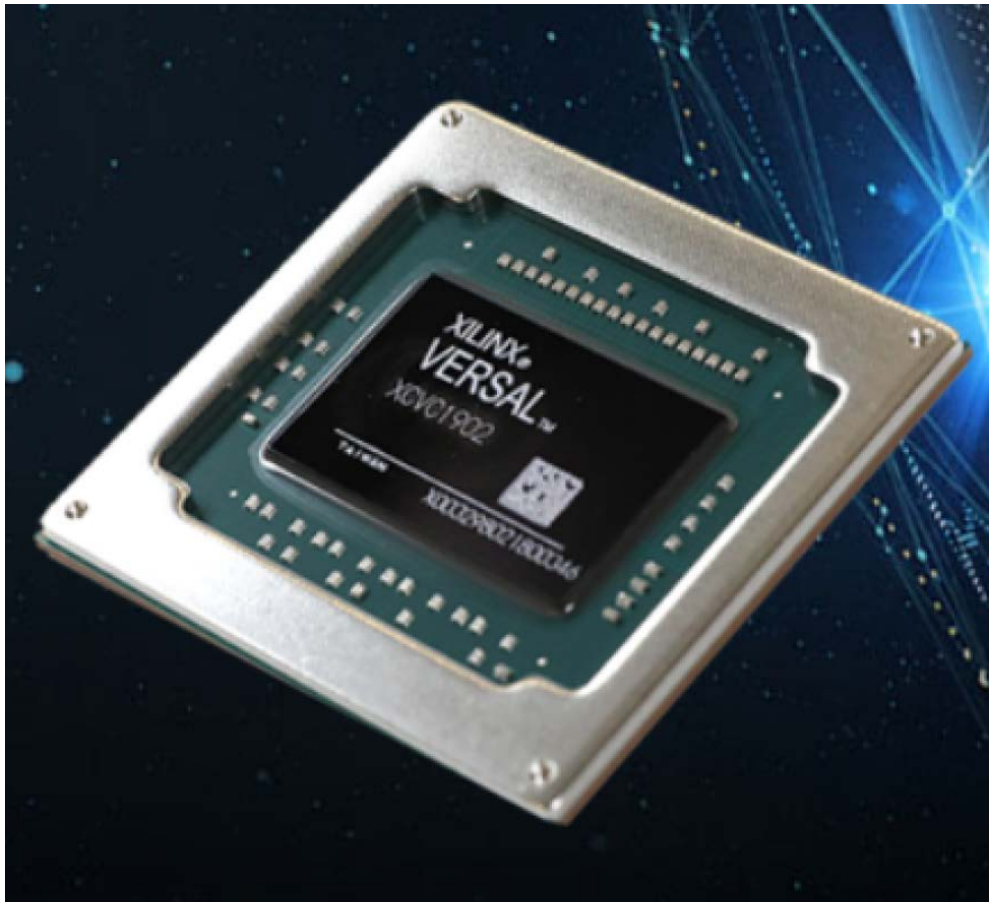
32 TB HBM

2-D toroidal mesh network

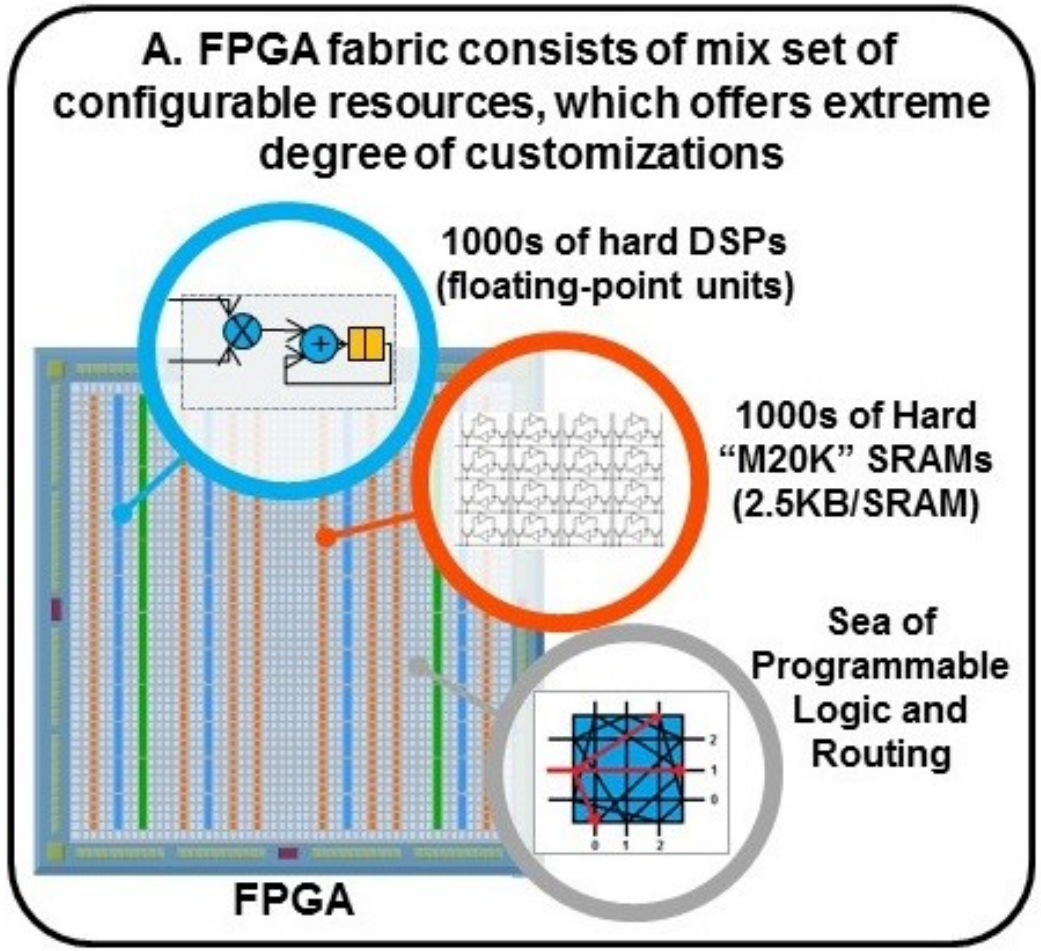
TPU v4 Pod

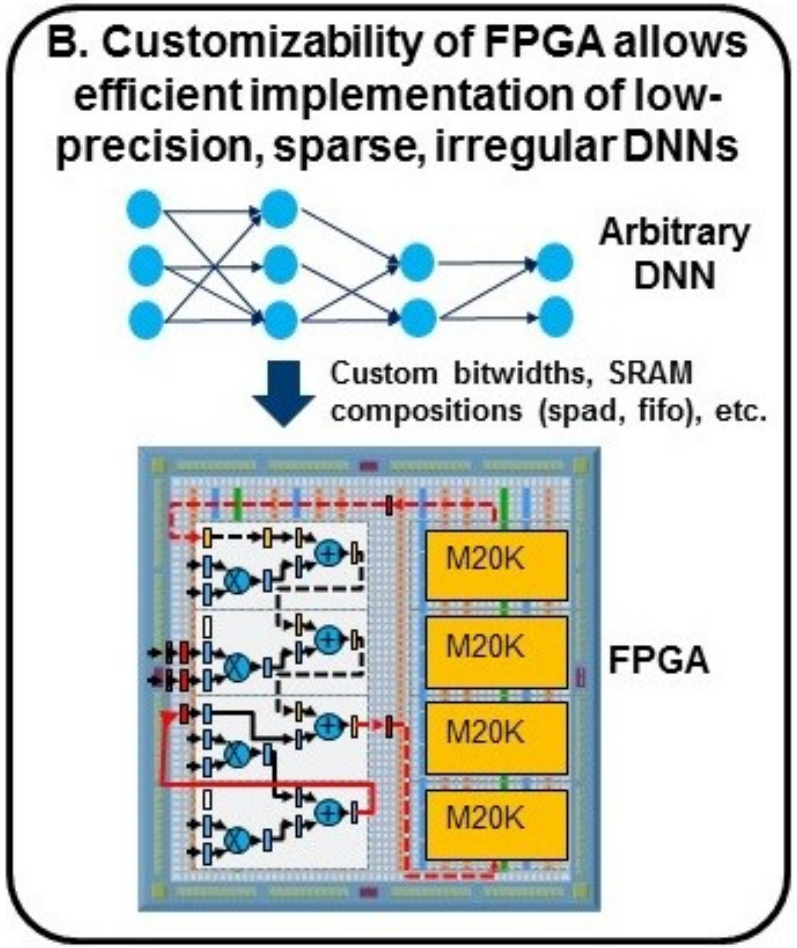
Field-programmable Gate Array (FPGA)



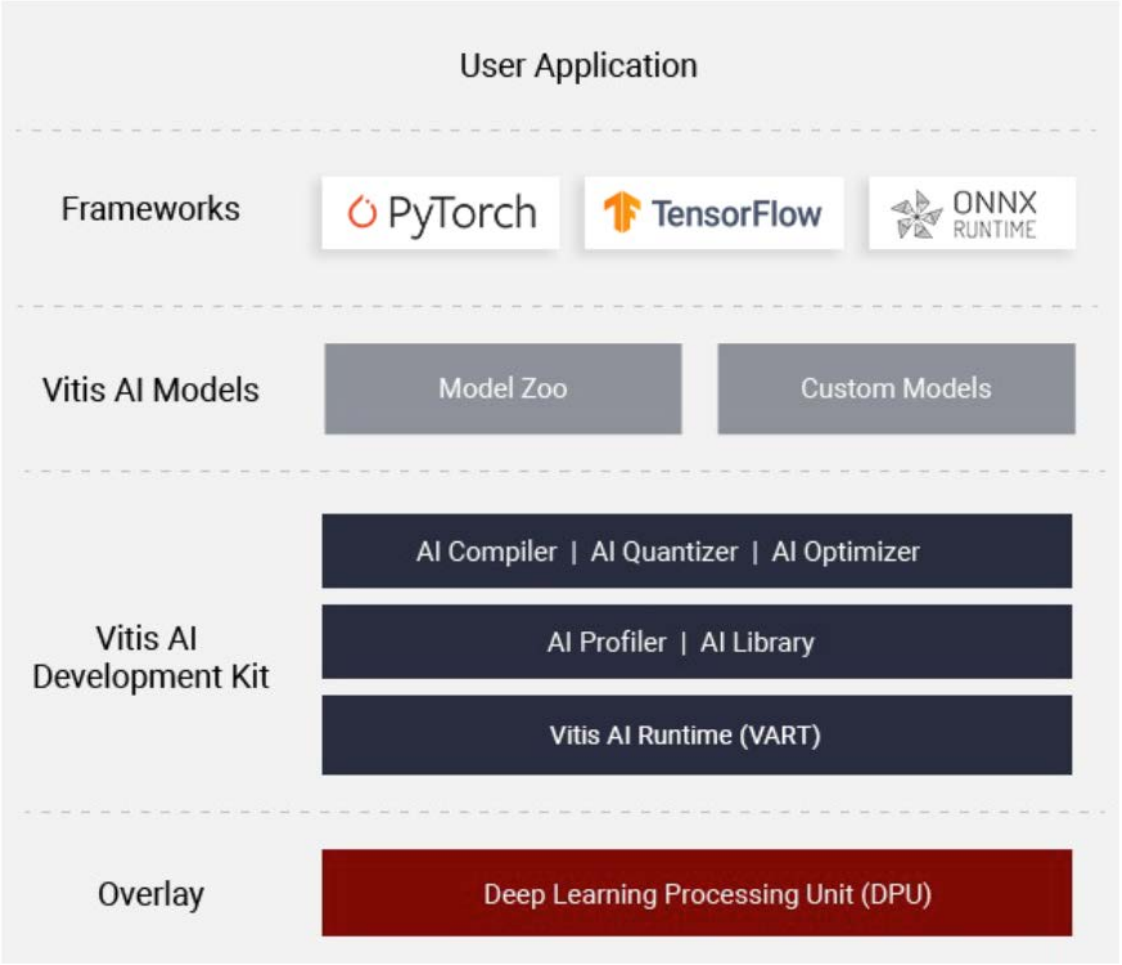


Field-programmable Gate Array (FPGA)

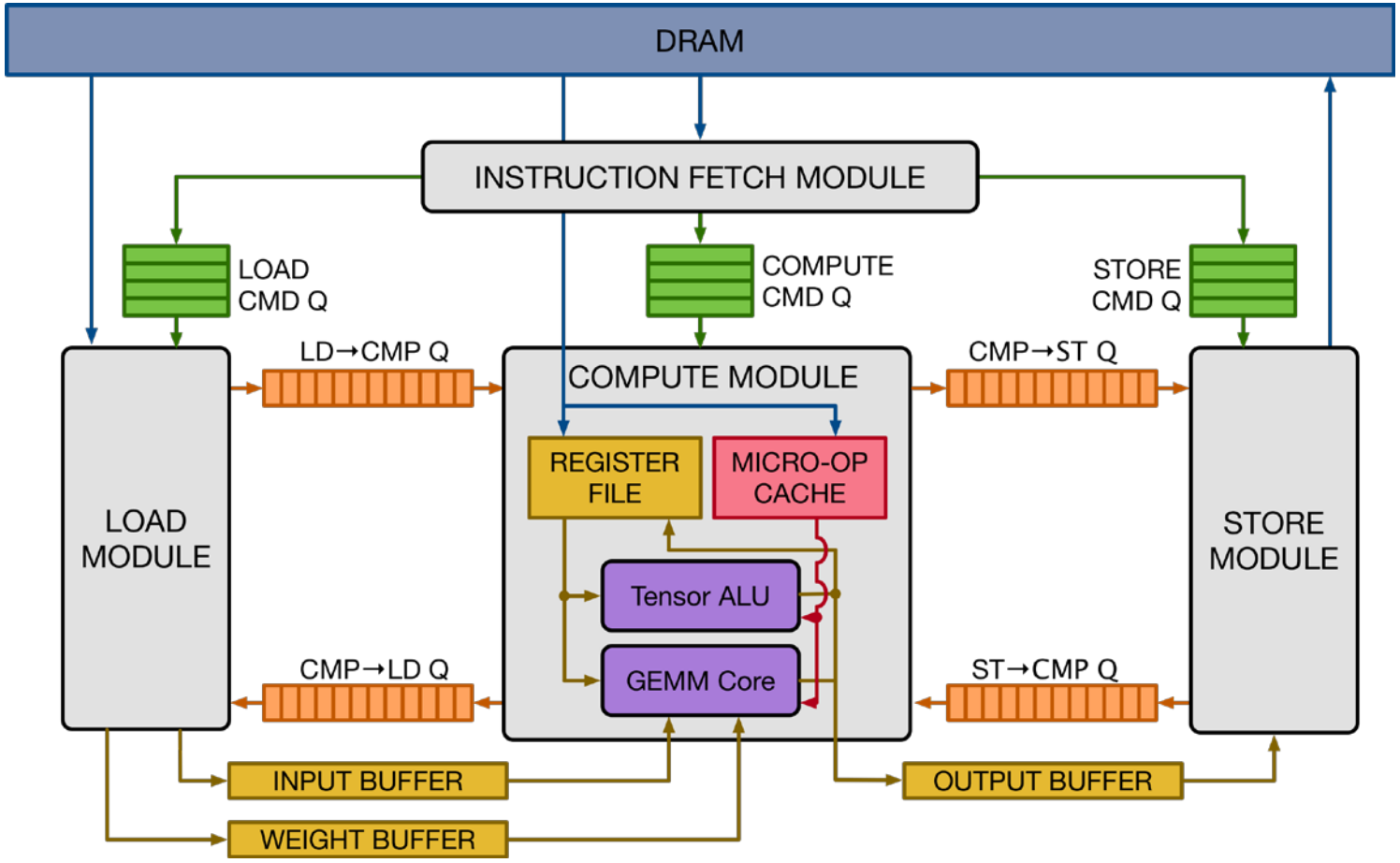




Xilinx Vitis AI

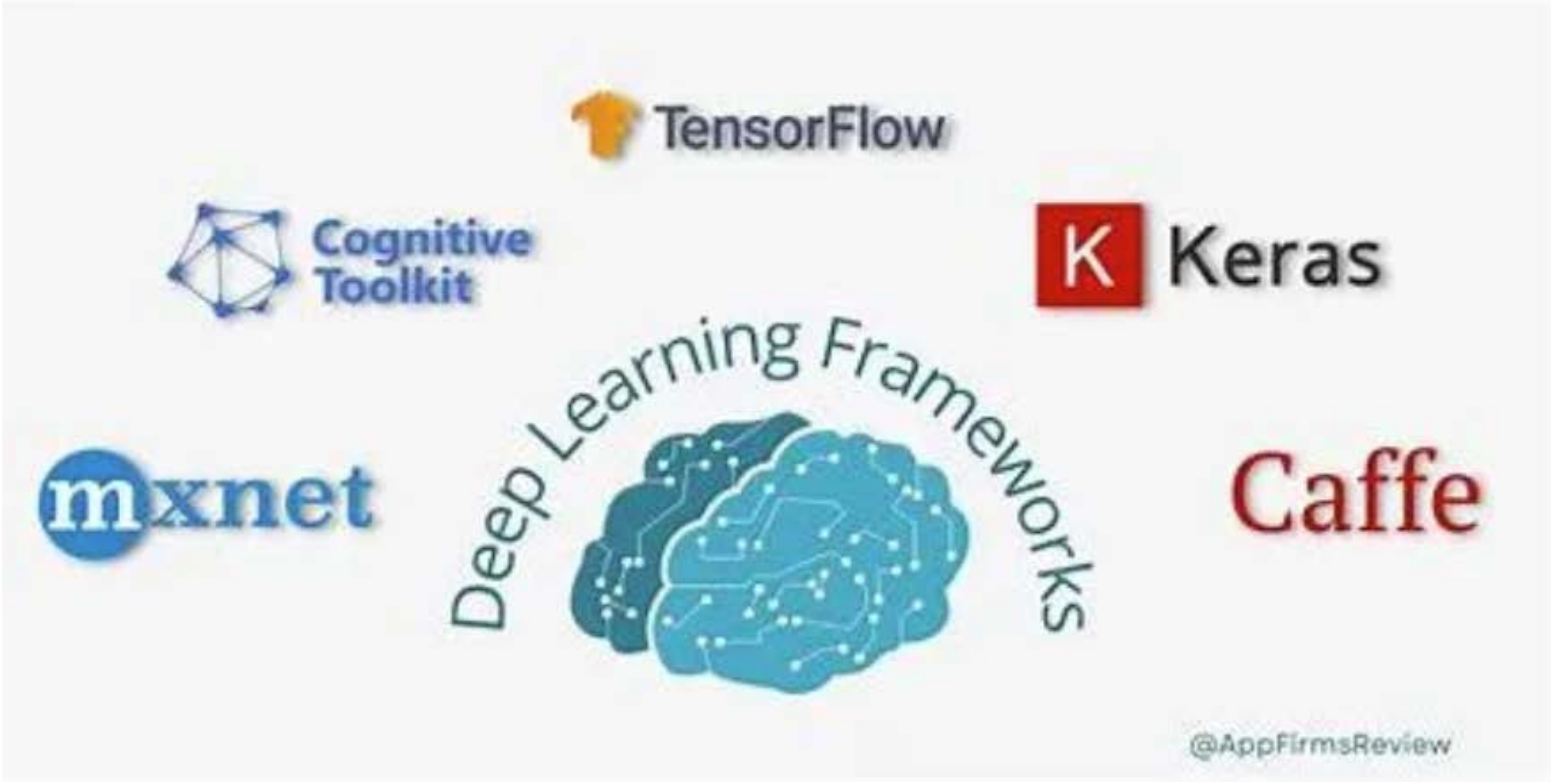


TVM VTA: Versatile Tensor Accelerator



Deep learning software

Deep Learning Frameworks



Best of Frameworks for Deep Learning

PyTorch 101: Tensors

Directly from data

Tensors can be created directly from data. The data type is automatically inferred.

```
data = [[1, 2],[3, 4]]
x_data = torch.tensor(data)
```

From a NumPy array

Tensors can be created from NumPy arrays (and vice versa - see Bridge with NumPy).

```
np_array = np.array(data)
x_np = torch.from_numpy(np_array)
```

With random or constant values:

`shape` is a tuple of tensor dimensions. In the functions below, it determines the dimensionality of the output tensor.

```
shape = (2,3,)
rand_tensor = torch.rand(shape)
ones_tensor = torch.ones(shape)
zeros_tensor = torch.zeros(shape)

print(f"Random Tensor: \n {rand_tensor} \n")
print(f"Ones Tensor: \n {ones_tensor} \n")
print(f"Zeros Tensor: \n {zeros_tensor}")
```

```
# We move our tensor to the GPU if available
if torch.cuda.is_available():
  tensor = tensor.to('cuda')
```

PyTorch 101: Tensors

Arithmetic operations

```
# This computes the matrix multiplication between
two tensors. y1, y2, y3 will have the same value
y1 = tensor @ tensor.T
y2 = tensor.matmul(tensor.T)

y3 = torch.rand_like(tensor)
torch.matmul(tensor, tensor.T, out=y3)


# This computes the element-wise product. z1, z2, z3
will have the same value
z1 = tensor * tensor
z2 = tensor.mul(tensor)

z3 = torch.rand_like(tensor)
torch.mul(tensor, tensor, out=z3)
```

In-place operations Operations that store the result into the operand are called in-place. They are denoted by a `_` suffix. For example: `x.copy_(y)`, `x.t_()`, will change `x`.

```
print(tensor, "\n")
tensor.add_(5)
print(tensor)
```

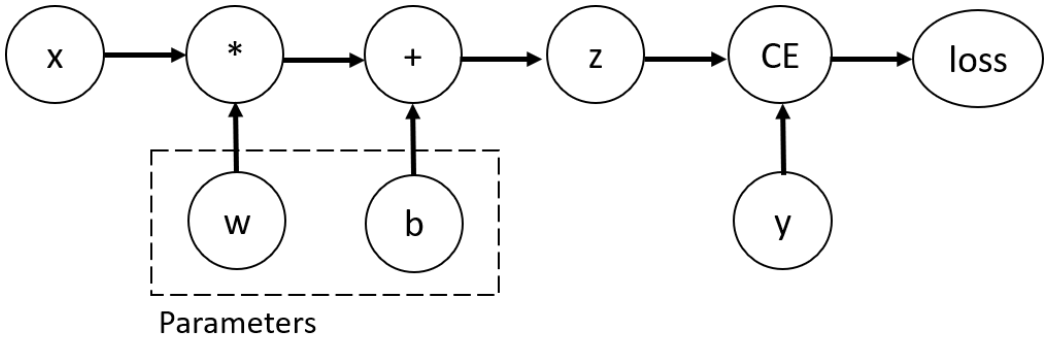
```
n = np.ones(5)
t = torch.from_numpy(n)
```

```
t = torch.ones(5)
print(f"t: {t}")
n = t.numpy()
print(f"n: {n}")
```

PyTorch 101: AutoGrad

```
import torch

x = torch.ones(5)  # input tensor
y = torch.zeros(3)  # expected output
w = torch.randn(5, 3, requires_grad=True)
b = torch.randn(3, requires_grad=True)
z = torch.matmul(x, w)+b
loss =
torch.nn.functional.binary_cross_entropy_with_logits
(z, y)
```




```
loss.backward()
print(w.grad)
print(b.grad)
```

Out:

```
tensor([[0.1496, 0.3099, 0.3019],
        [0.1496, 0.3099, 0.3019],
        [0.1496, 0.3099, 0.3019],
        [0.1496, 0.3099, 0.3019],
        [0.1496, 0.3099, 0.3019]])
tensor([0.1496, 0.3099, 0.3019])
```

```
with torch.no_grad():
    z = torch.matmul(x, w)+b
print(z.requires_grad)
```

PyTorch 101: nn.Module

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10),
            nn.ReLU()
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits
```

```
device = 'cuda' if torch.cuda.is_available() else
'cpu'
print('Using {} device'.format(device))
```

```
model = NeuralNetwork().to(device)
print(model)
```

```
X = torch.rand(1, 28, 28, device=device)
logits = model(X)
pred_probab = nn.Softmax(dim=1)(logits)
y_pred = pred_probab.argmax(1)
print(f"Predicted class: {y_pred}")
```

PyTorch 101: Train

```
def train_loop(dataloader, model, loss_fn,
optimizer):
    size = len(dataloader.dataset)
    for batch, (X, y) in enumerate(dataloader):
        # Compute prediction and loss
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if batch % 100 == 0:
            loss, current = loss.item(), batch *
len(X)
            print(f"loss: {loss:>7f}
[{current:>5d}/{size:>5d}]")
```

```
def test_loop(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    test_loss, correct = 0, 0

    with torch.no_grad():
        for X, y in dataloader:
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) ==
y).type(torch.float).sum().item()

    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy:
{(100*correct):>0.1f}%, Avg loss: {test_loss:>8f}
\n")
```

PyTorch 101: Train

```
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(),
lr=learning_rate)

epochs = 10
for t in range(epochs):
    print(f"Epoch
{t+1}\n-------------------------------")
    train_loop(train_dataloader, model, loss_fn,
optimizer)
    test_loop(test_dataloader, model, loss_fn)
print("Done!")
```

```
Epoch 1
-------------------------------
loss: 2.296610  [    0/60000]
loss: 2.292522  [ 6400/60000]
loss: 2.286929  [12800/60000]
loss: 2.288398  [19200/60000]
loss: 2.276151  [25600/60000]
loss: 2.263381  [32000/60000]
loss: 2.276544  [38400/60000]
loss: 2.262336  [44800/60000]
loss: 2.250045  [51200/60000]
loss: 2.248366  [57600/60000]
Test Error:
 Accuracy: 32.4%, Avg loss: 2.239535
```

PYTORCH

```
# models
encoder = nn.Sequential(nn.Linear(28 * 28, 64), nn.ReLU(), nn.Linear(64, 3))
decoder = nn.Sequential(nn.Linear(3, 64), nn.ReLU(), nn.Linear(64, 28 * 28))

encoder.cuda(0)
decoder.cuda(0)

# download on rank 0 only
if global_rank == 0:
    mnist_train = MNIST(os.getcwd(), train=True, download=True)

# split dataset
transform=transforms.Compose([transforms.ToTensor(),
                              transforms.Normalize(0.5, 0.5)])
mnist_train = MNIST(os.getcwd(), train=True, download=True, transform=transform)

# train (55,000 images), val split (5,000 images)
mnist_train, mnist_val = random_split(mnist_train, [55000, 5000])

# The dataloaders handle shuffling, batching, etc...
mnist_train = DataLoader(mnist_train, batch_size=64)
mnist_val = DataLoader(mnist_val, batch_size=64)

 # optimizer
 params = [encoder.parameters(), decoder.parameters()]
 optimizer = torch.optim.Adam(params, lr=1e-3)

# TRAIN LOOP
model.train()
num_epochs = 1
for epoch in range(num_epochs):
  For train_batch in mnist_train:
    x, y = train_batch
    x = x.cuda(0)
    x = x.view(x.size(0), -1)
    z = encoder(x)
    x_hat = decoder(z)
    loss = F.mse_loss(x_hat, x)
    print('train loss: ', loss.item())

    loss.backward()
    optimizer.step()
    optimizer.zero_grad()

# EVAL LOOP
model.eval()
with torch.no_grad():
  val_loss = []
  for val_batch in mnist_val:
    x, y = val_batch
    x = x.cuda(0)
    x = x.view(x.size(0), -1)
    z = encoder(x)
    x_hat = decoder(z)
    loss = F.mse_loss(x_hat, x)
    val_loss.append(loss)
  val_loss = torch.mean(torch.tensor(val_loss))
  model.train()
```

PYTORCH LIGHTNING

PyTorch Lightning

Turn PyTorch into Lightning

Lightning is just plain PyTorch.

Any Questions?

Accelerating Training Systems

GPT-2 Model

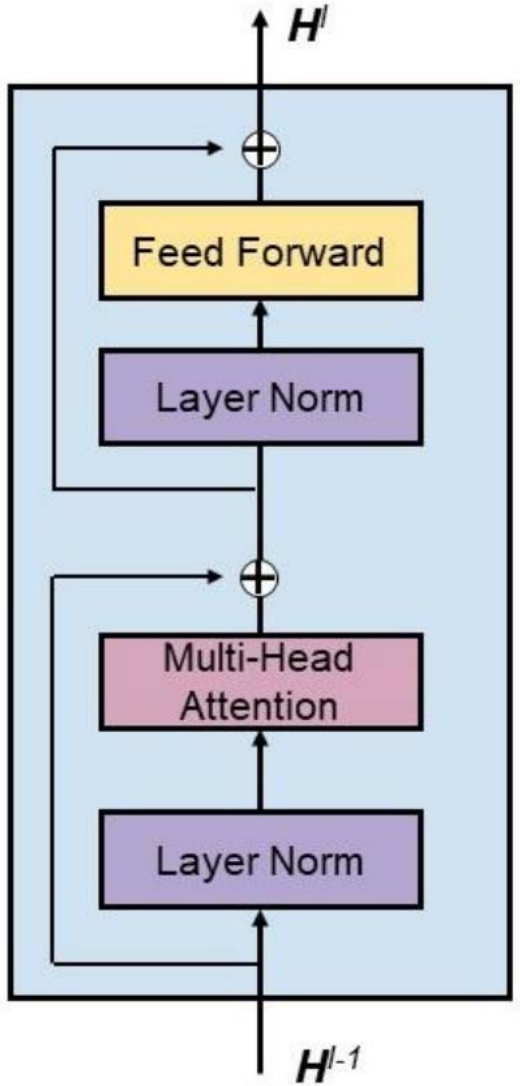
$$H^0 = E + P$$

$$H^l = \text{Transformer\_block}\left(H^{l-1}\right), 1 \ll l \ll L$$

$$\text{FFN}(x) = max(0, xW_1 + b_1)W_2 + b_2$$

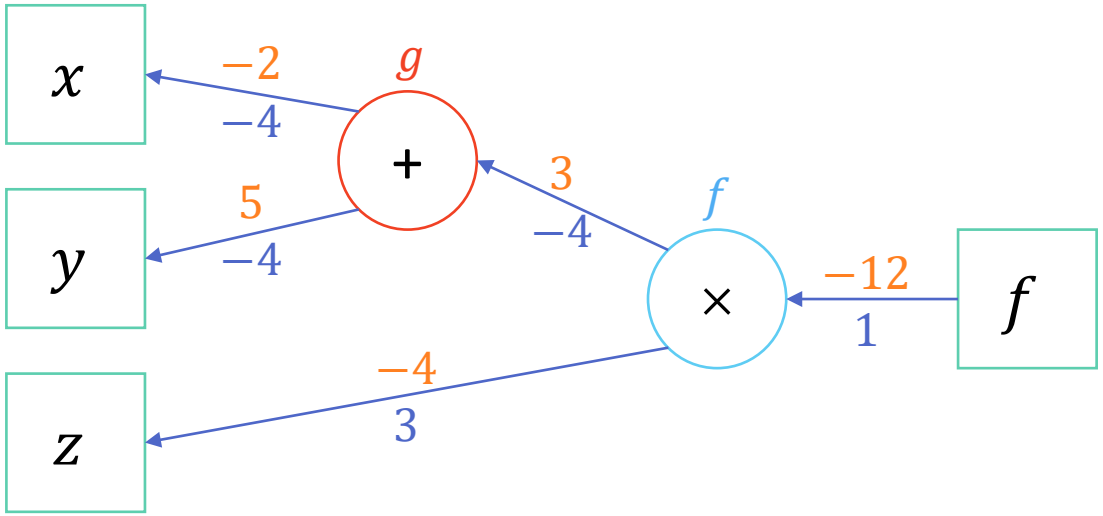
$$\text{P}(\text{x}_\text{k} \mid \text{x}_1, x_2, \ldots, x_{k-1}) = \text{softmax}\left(W_v H_k^L\right)\Big|_{x_k}$$

$$\text{Loss} = \sum_{k=1}^{\text{n}} -\log \text{P}(\text{x}_\text{k} \mid \text{x}_1, x_2, \ldots, x_{k-1})$$



Target

- Faster
  - Train the model in less time
  - Forward and Backward time
- Larger
  - Train larger model in limited memory
- Where does my memory go?
  - Parameter
  - Gradient
  - Optimizer State
  - Activation



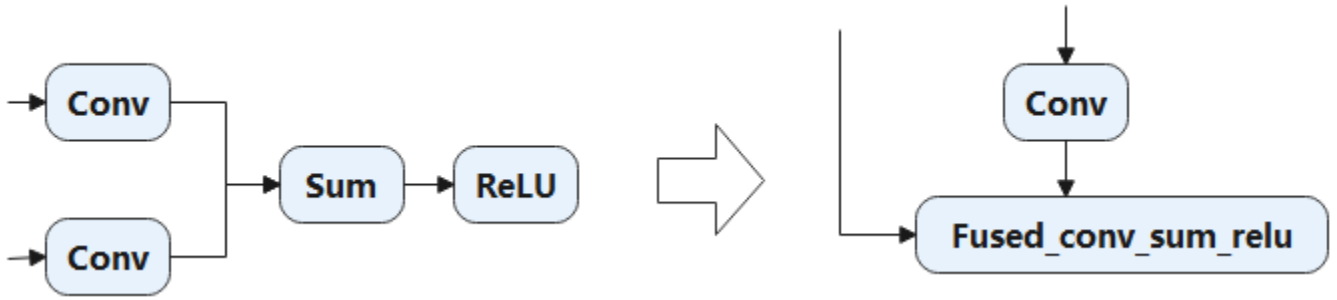
- $v \leftarrow \beta_1 v + (1-\beta_1)\nabla L(W)$
- $s \leftarrow \beta_2 s + (1-\beta_2)\left(\nabla L(W)\right)^2$

How to enhance performance?

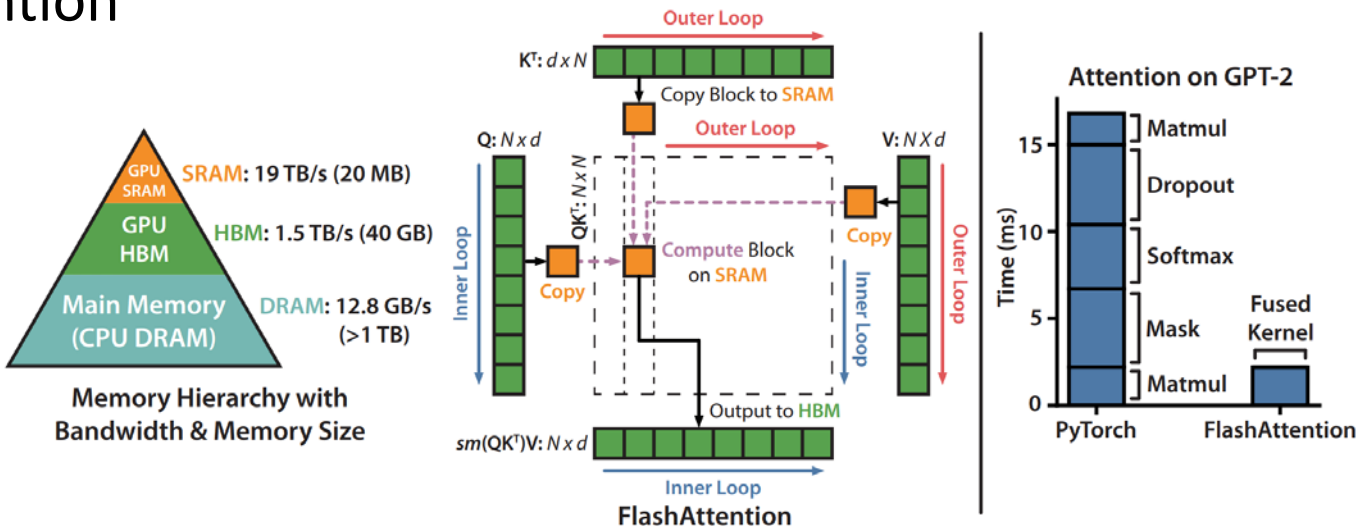
- More efficient operator implementation
- More efficient model design
  - Sparsity
  - PEFT
- Scalability
  - Parallelism
  - ZeRO
  - Heterogeneous training
- More efficient training procedure
  - Gradient Checkpointing
  - Gradient Accumulation
  - AMP
  - New optimizer

More efficient operator implementation

- Operator Fusion

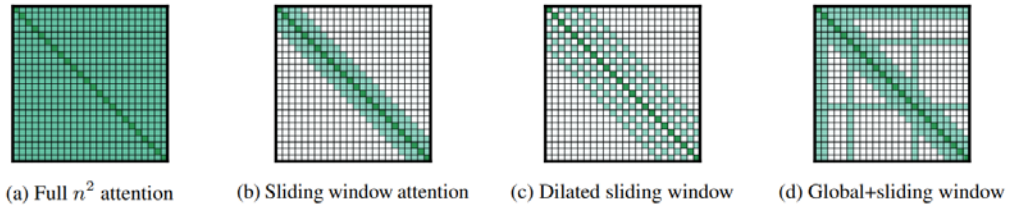


- More efficient operator
  - FlashAttention

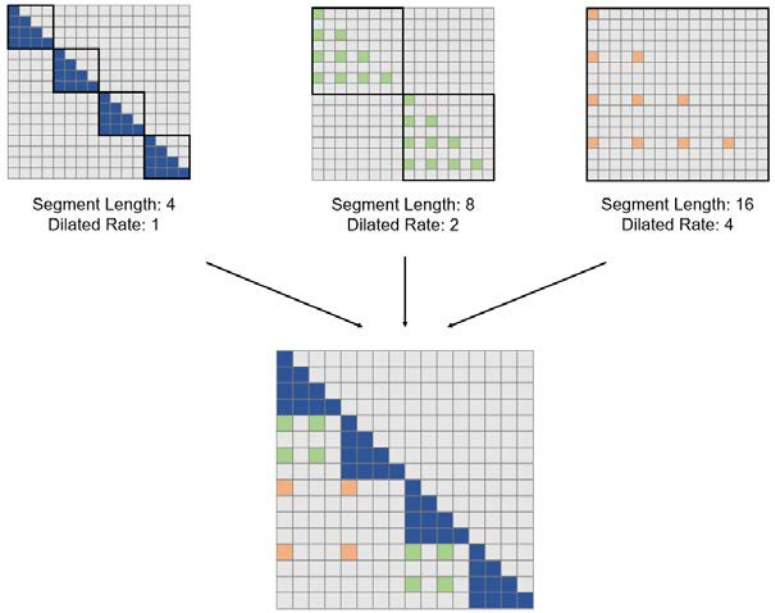


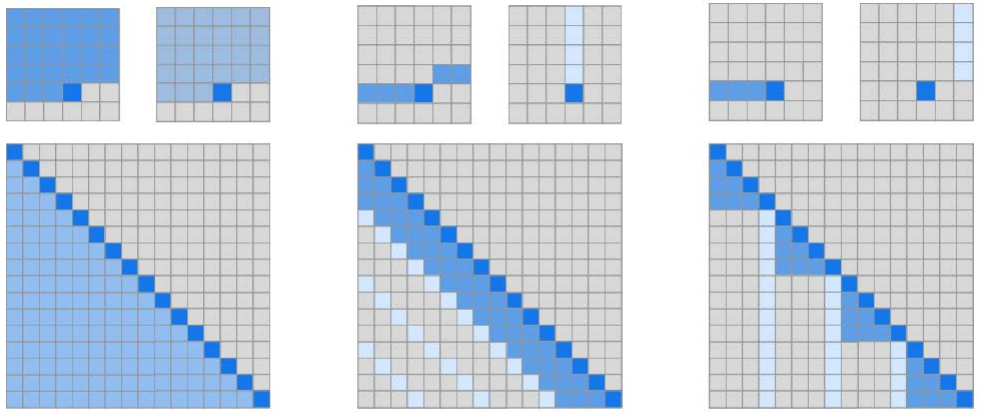
More efficient model design

- Sparsity
  - LongFormer
  - Sparse Transformer
  - LongNet



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

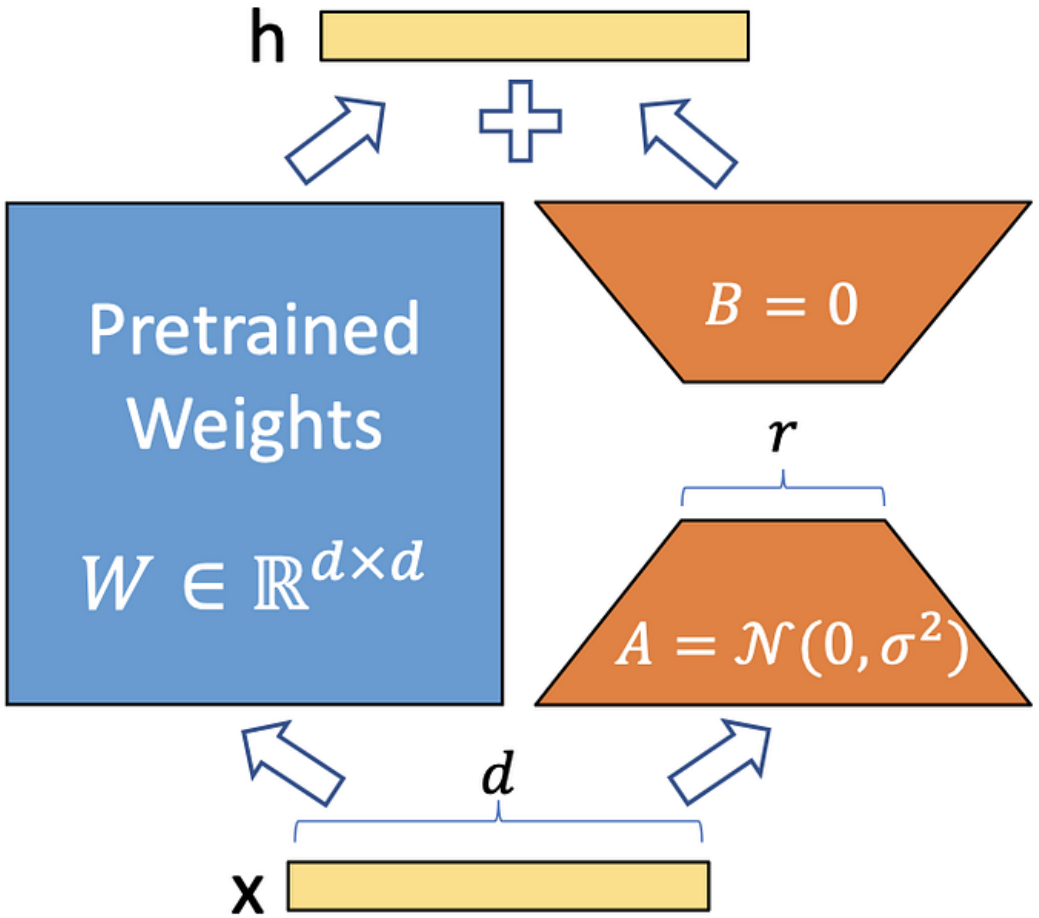




(a) Transformer (b) Sparse Transformer (strided) (c) Sparse Transformer (fixed)

PEFT (Parameter-Efficient Fine-tuning)

- LoRA



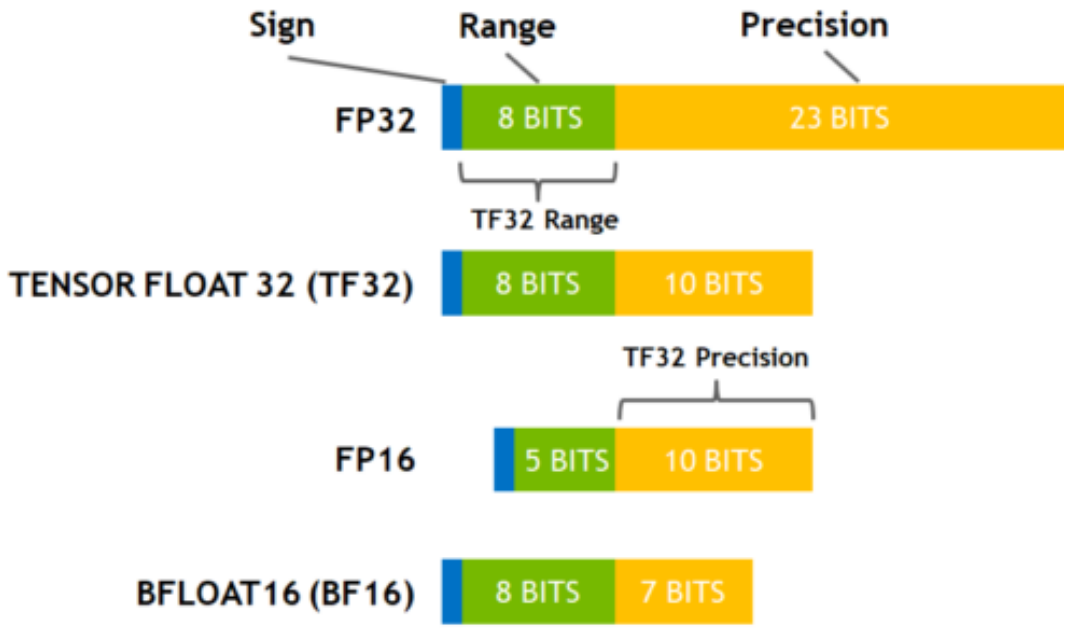
More efficient training procedure

- Gradient Accumulation
  - Increase batch size
    - Reduce relative GPU memory cost
    - Reduce overall optimizer iteration steps
- Gradient Checkpointing
  - Release activation records after forward
  - Slow down 20~30%, but reduce memory footprints

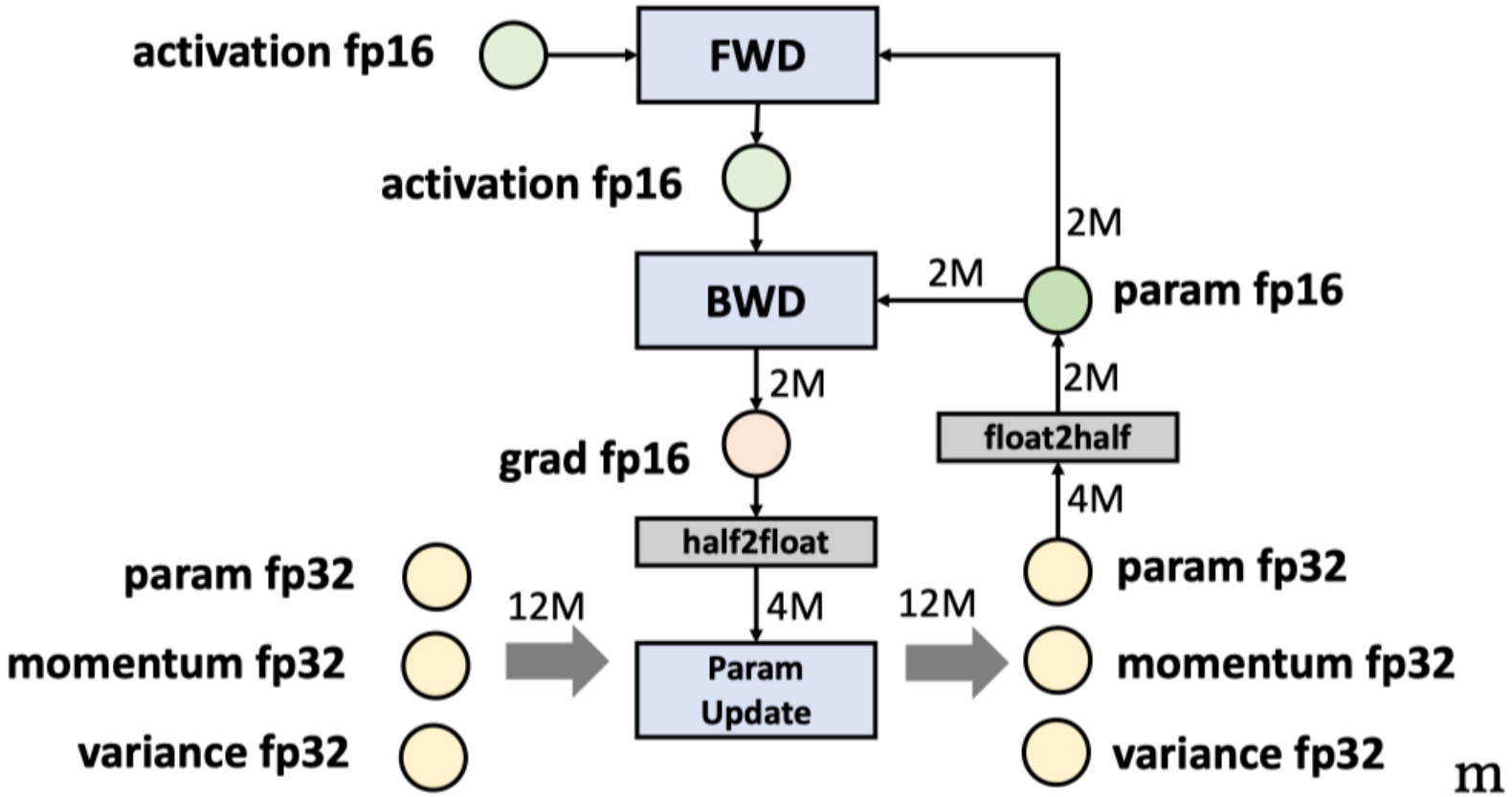
Mixed Precision Training

- FP16/TF32/BF16

	A100 80GB PCIe	A100 80GB SXM
FP64	9.7 TFLOPS	
FP64 Tensor Core	19.5 TFLOPS	
FP32	19.5 TFLOPS	
Tensor Float 32 (TF32)	156 TFLOPS | 312 TFLOPS*	
BFLOAT16 Tensor Core	312 TFLOPS | 624 TFLOPS*	
FP16 Tensor Core	312 TFLOPS | 624 TFLOPS*	
INT8 Tensor Core	624 TOPS | 1248 TOPS*	
GPU Memory	80GB HBM2e	80GB HBM2e
GPU Memory Bandwidth	1,935 GB/s	2,039 GB/s
Max Thermal Design Power (TDP)	300W	400W ***
Multi-Instance GPU	Up to 7 MIGs @ 10GB	Up to 7 MIGs @ 10GB



Typical training procedure (with fp16)

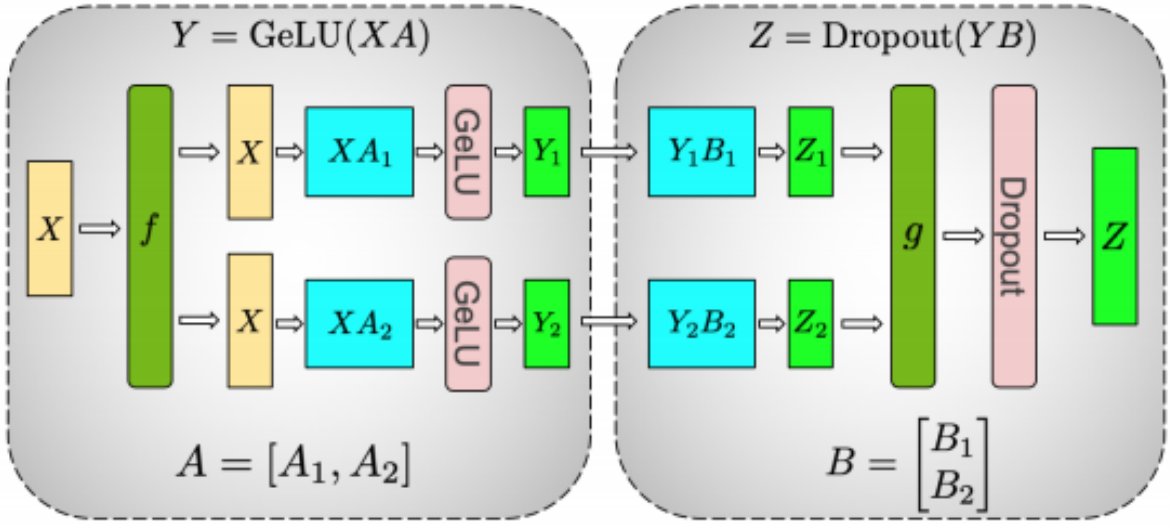


Parallelism

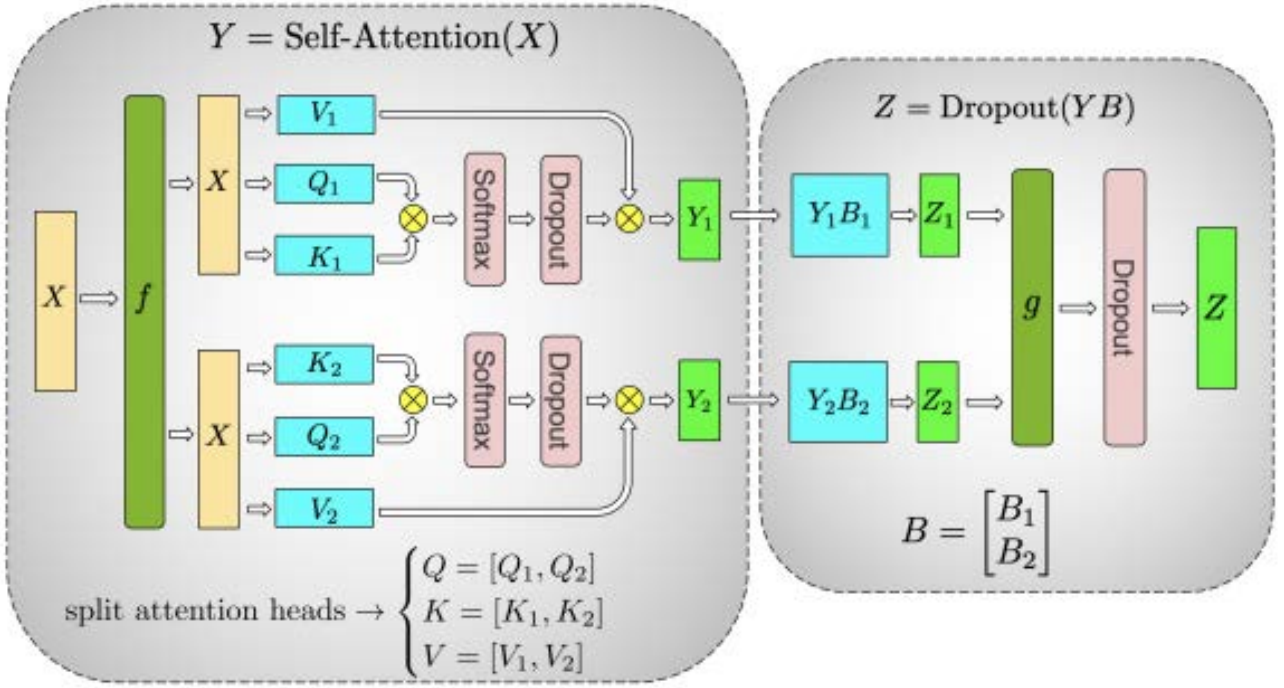
- Model Parallelism
  - Tensor Parallel
  - Pipeline Parallel
- Data Parallelism
  - DP
  - ZeRO

Tensor Parallel

- 1D
  - Proposed first in Megatron



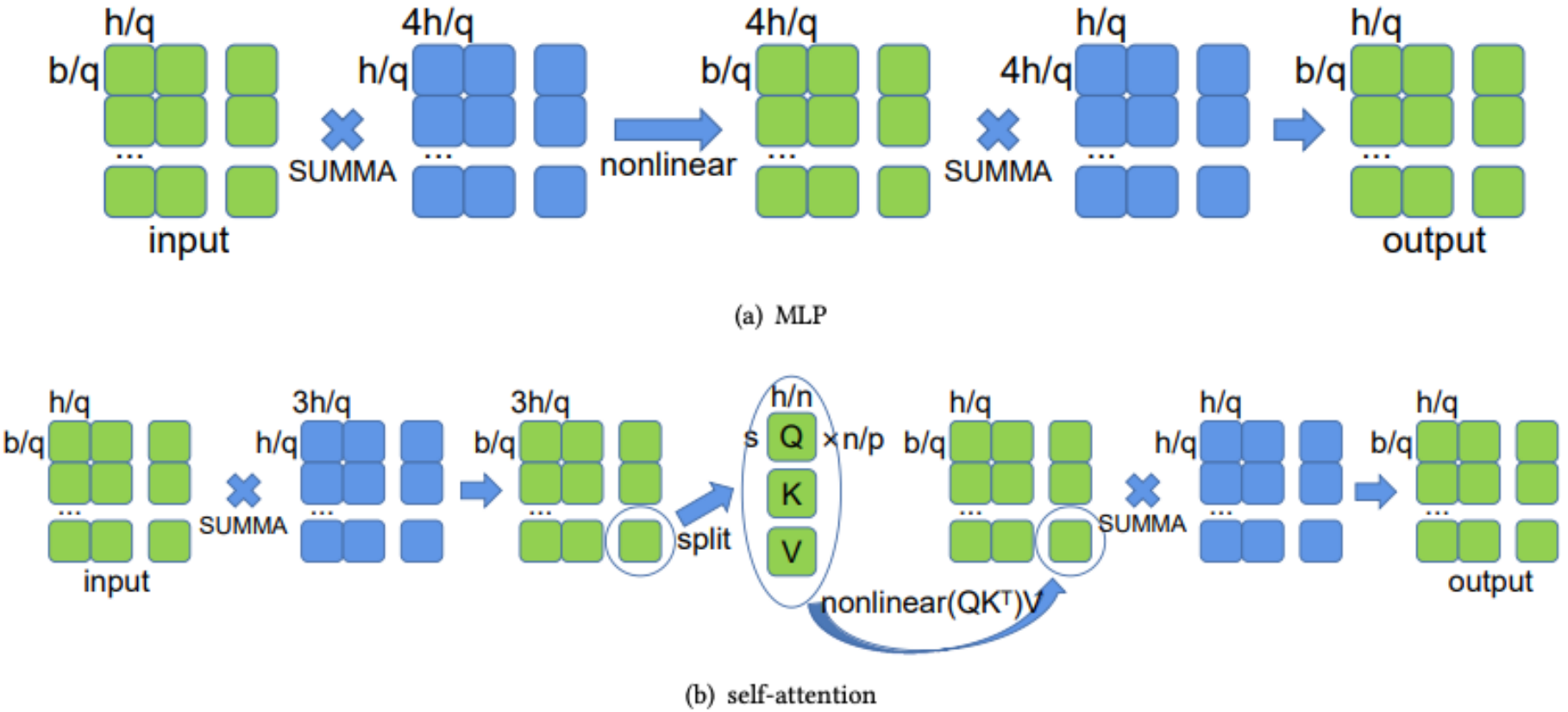
(a) MLP



(b) Self-Attention

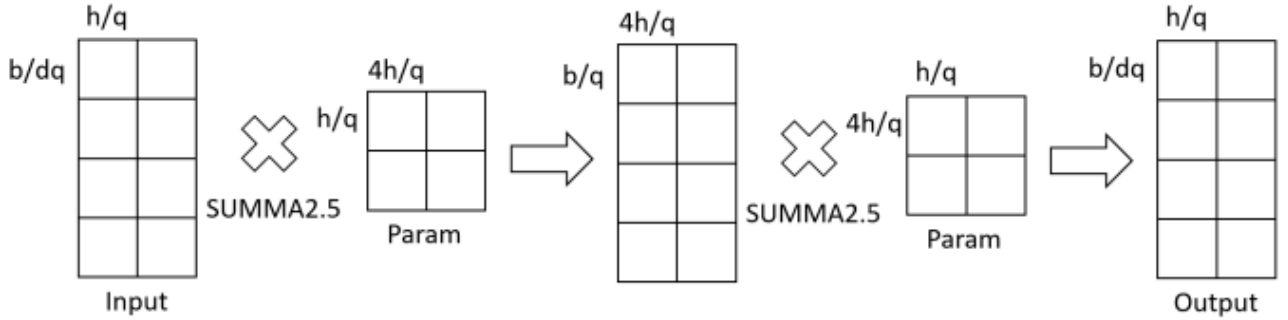
Tensor Parallel

- 2D
  - An Efficient 2D Method for Training Super-Large Deep Learning Models

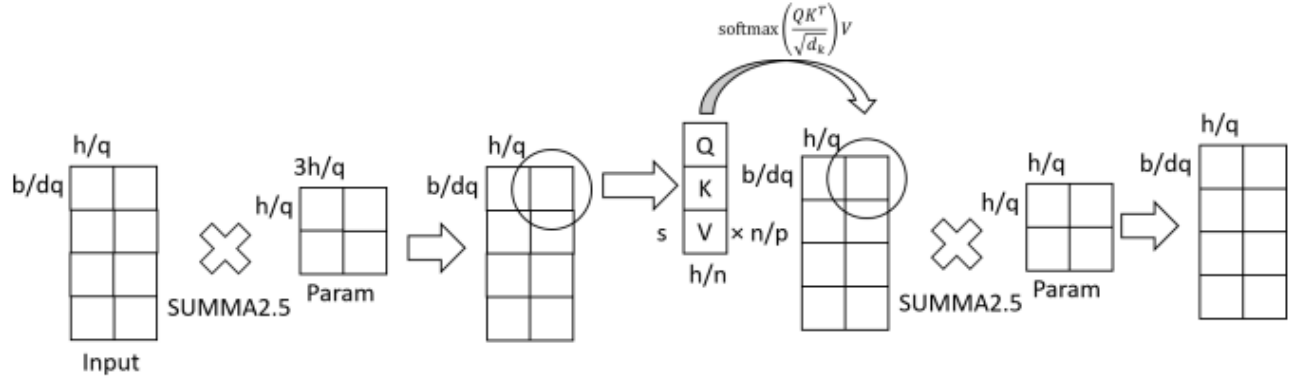


Tensor Parallel

- 2.5D
  - 2.5-dimensional distributed model training



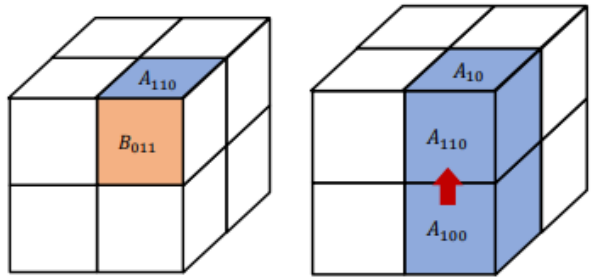
(a) Feed forward



(b) Multi-head attention

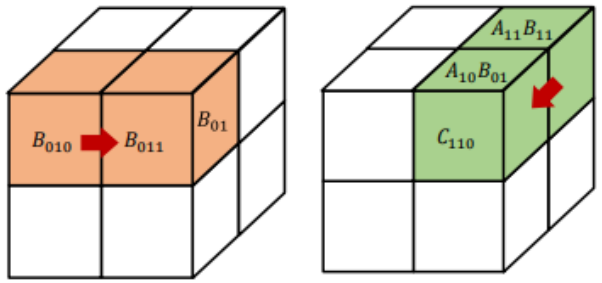
Tensor Parallel

- 3D
  - [Maximizing Parallelism in Distributed Training for Huge Neural Networks]



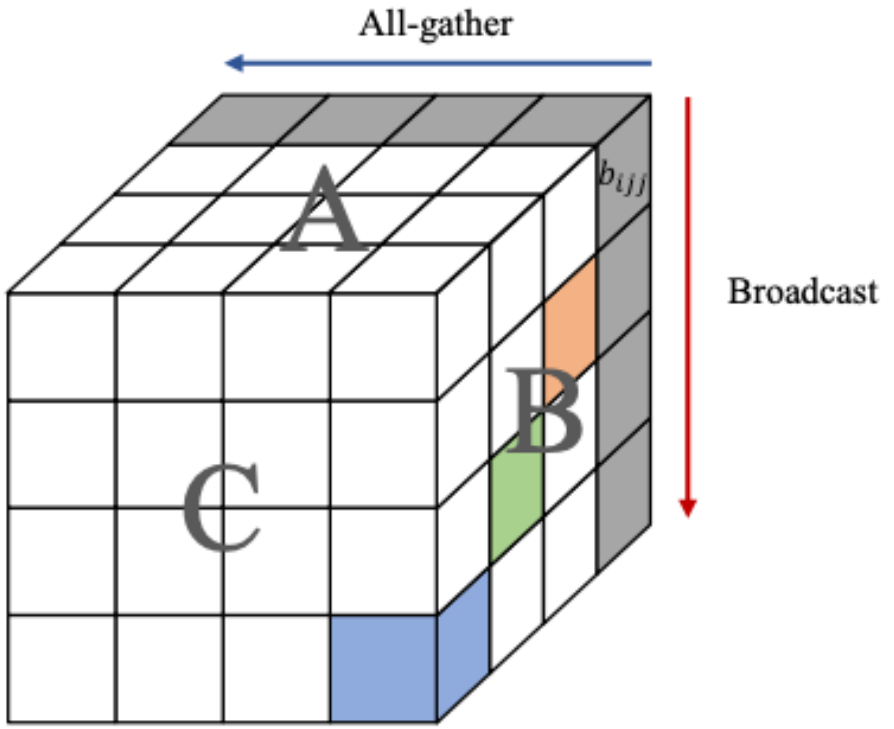
(a) The input and weight submatrices on each processor.

(b) All-gather A_{il} in the y direction.



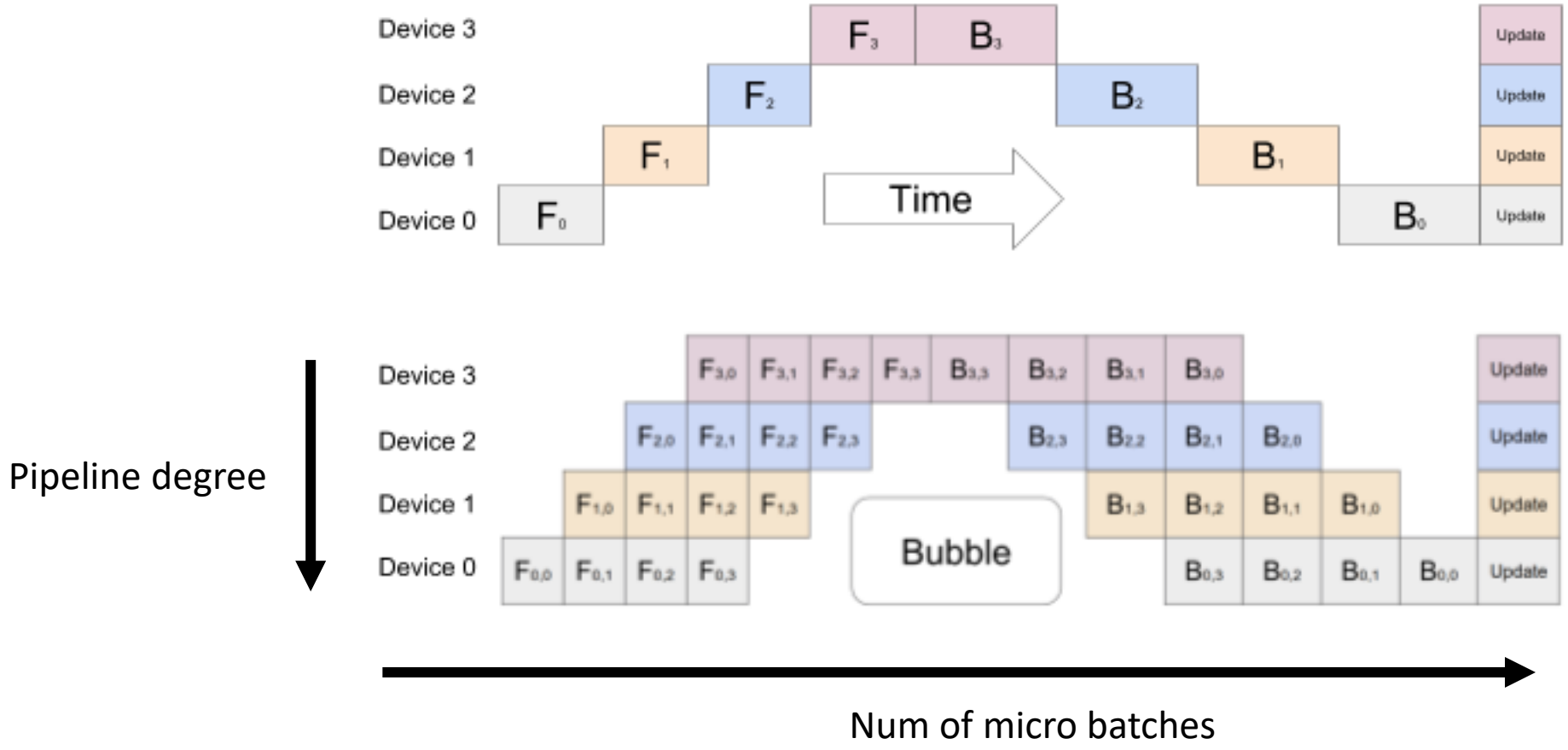
(c) All-gather B_{lj} in the x direction.

(d) Compute each C_{ijl} and reduce-scatter them in the z direction.



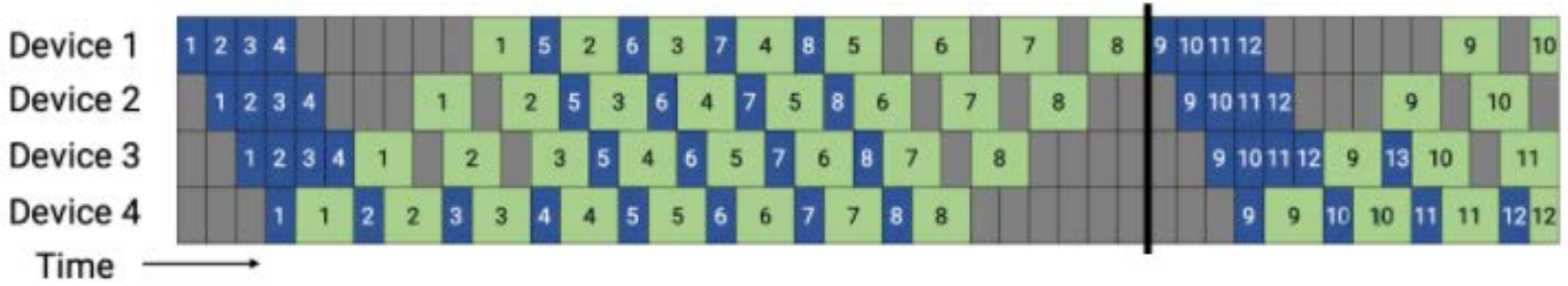
Pipeline Parallel

- GPipe

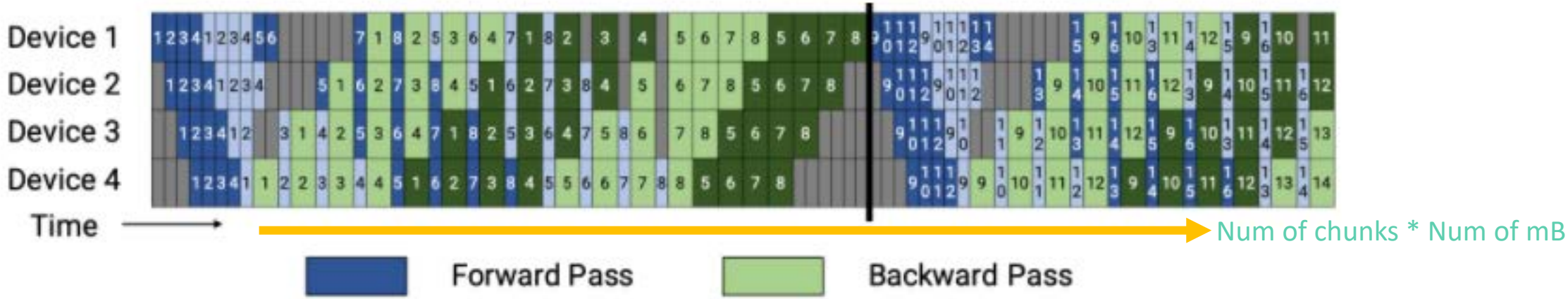


Pipeline Parallel

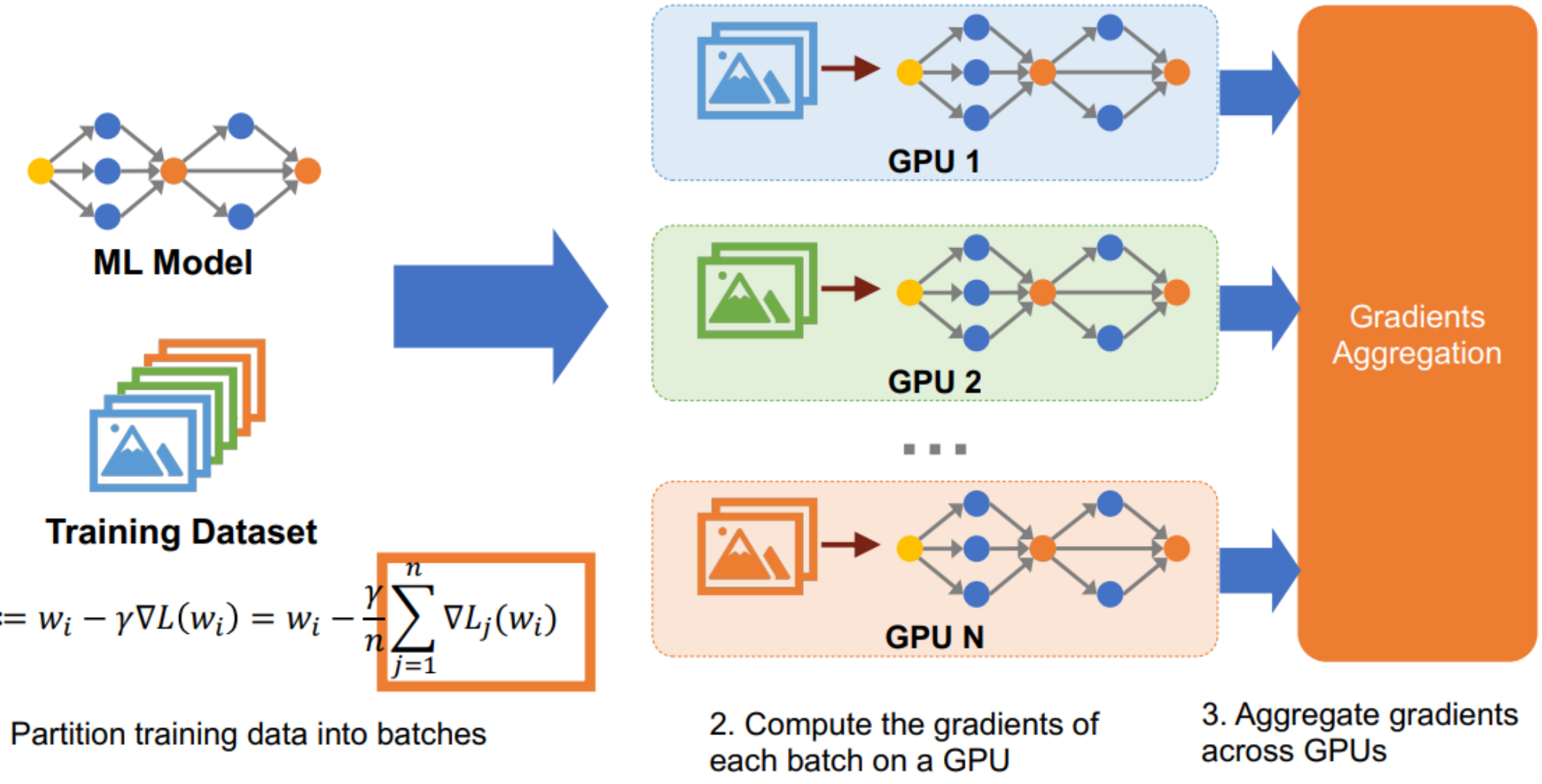
- 1F1B (one forward pass followed by one backward pass)



- Interleaved pipeline (backward execution of the microbatches is prioritized whenever possible)

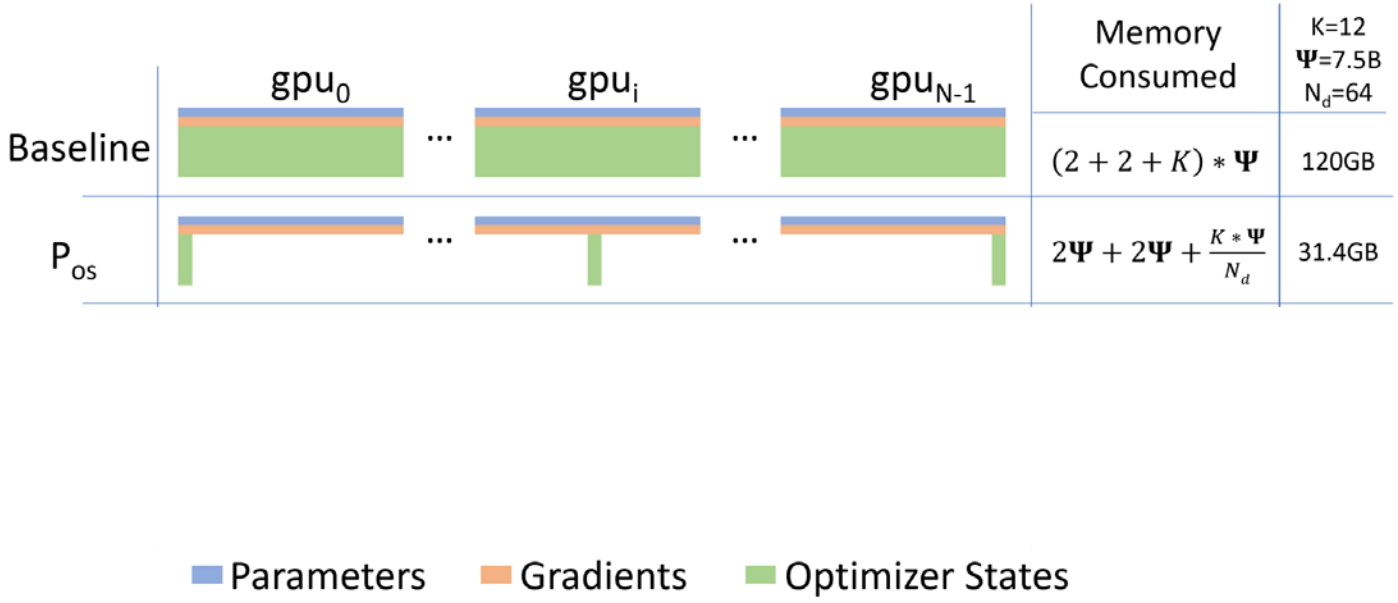


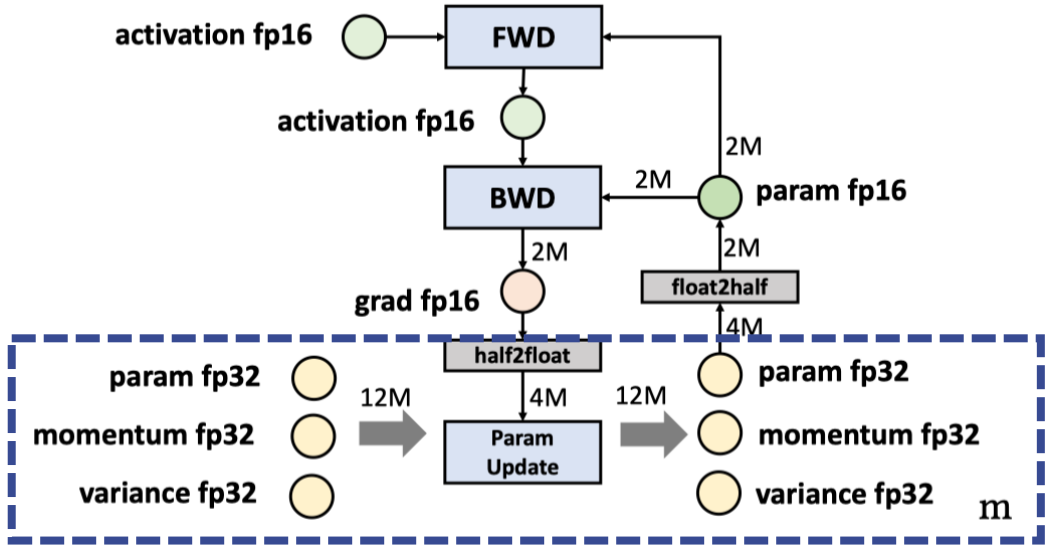
Data Parallel



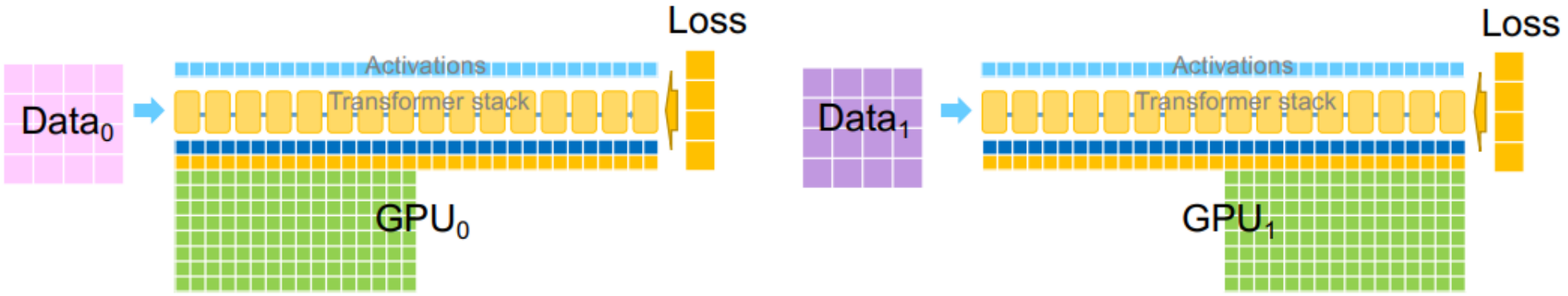
ZeRO

- Stage 1


gpu0
gpui
gpuN-1
Memory Consumed
K=12
Ψ=7.5B
Nd=64
Baseline
(2 + 2 + K) * Ψ
120GB
Pos
2Ψ + 2Ψ + K * Ψ / Nd
31.4GB
Parameters
Gradients
Optimizer States


activation fp16
FWD
activation fp16
2M
BWD
2M
param fp16
2M
2M
grad fp16
float2half
4M
half2float
param fp32
momentum fp32
variance fp32
12M
4M
12M
Param Update
param fp32
momentum fp32
variance fp32
m

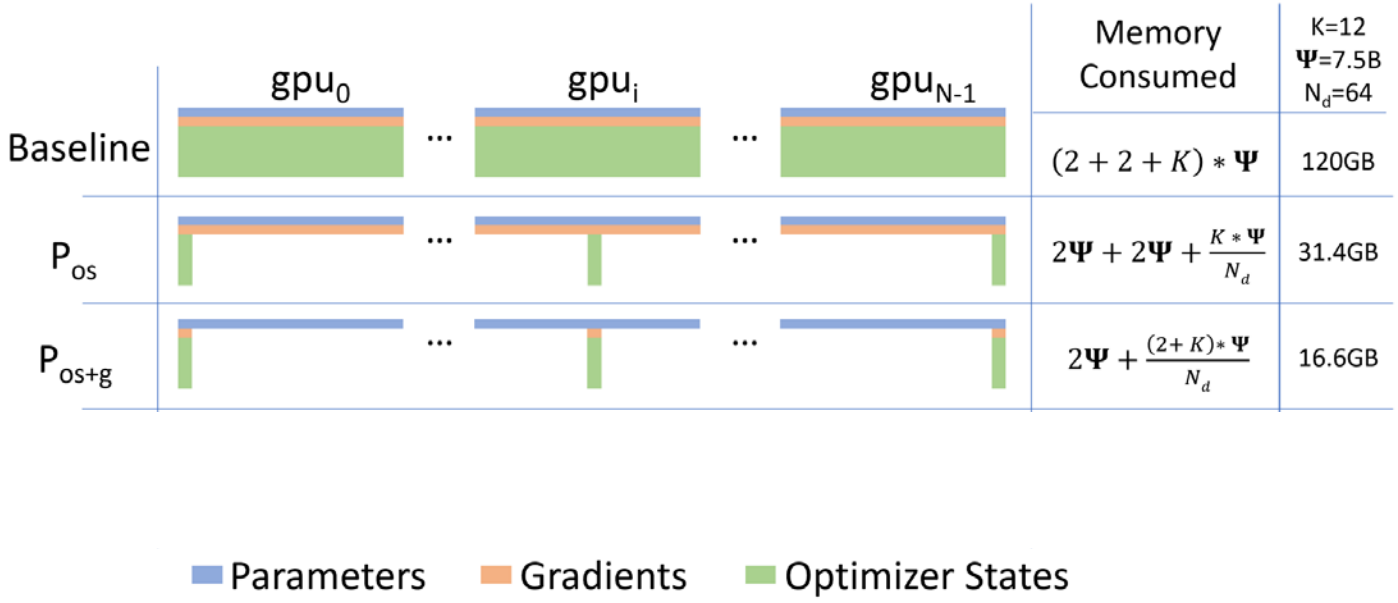
ZeRO

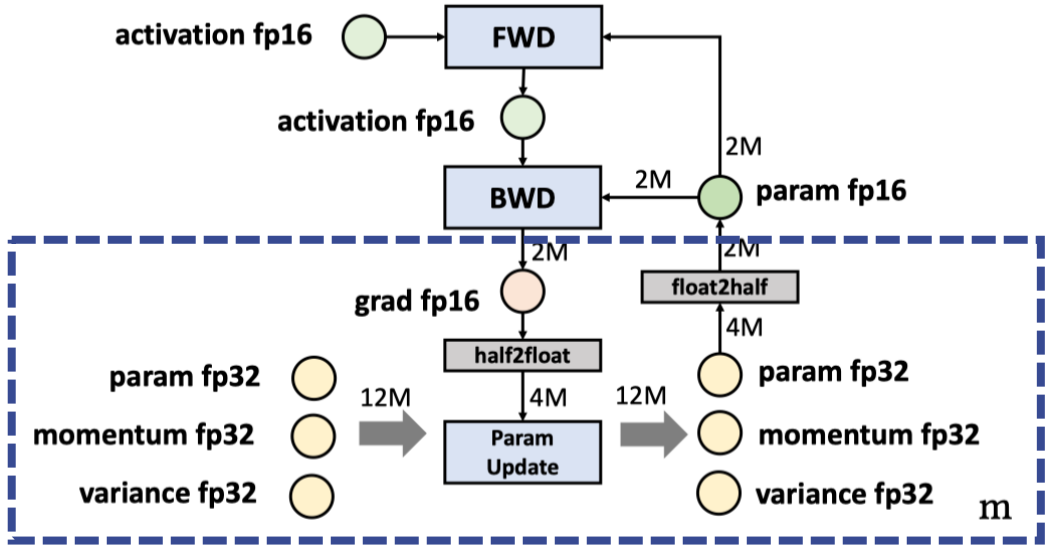


- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients and AllReduce to average
- Update the FP32 weights with ADAM optimizer
- Update the FP16 weights
- All Gather the FP16 weights to complete the iteration

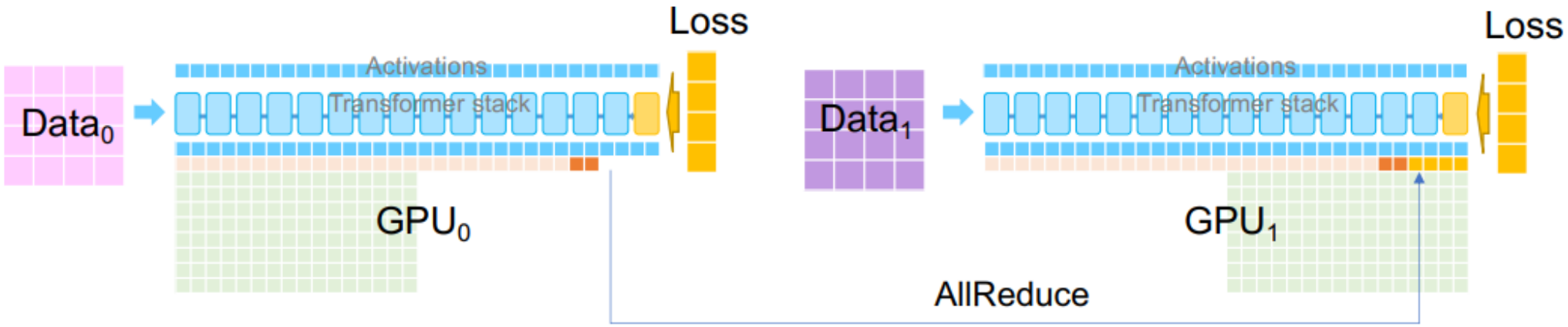
ZeRO

- Stage 2


gpu0
gpui
gpuN-1
Memory Consumed
K=12
Ψ=7.5B
Nd=64
Baseline
(2 + 2 + K) * Ψ
120GB
Pos
2Ψ + 2Ψ + (K * Ψ)/Nd
31.4GB
Pos+g
2Ψ + ((2+K) * Ψ)/Nd
16.6GB
Parameters
Gradients
Optimizer States


activation fp16
FWD
activation fp16
BWD
2M
2M
param fp16
2M
2M
grad fp16
float2half
4M
half2float
param fp32
momentum fp32
variance fp32
12M
4M
12M
Param Update
param fp32
momentum fp32
variance fp32
m

ZeRO

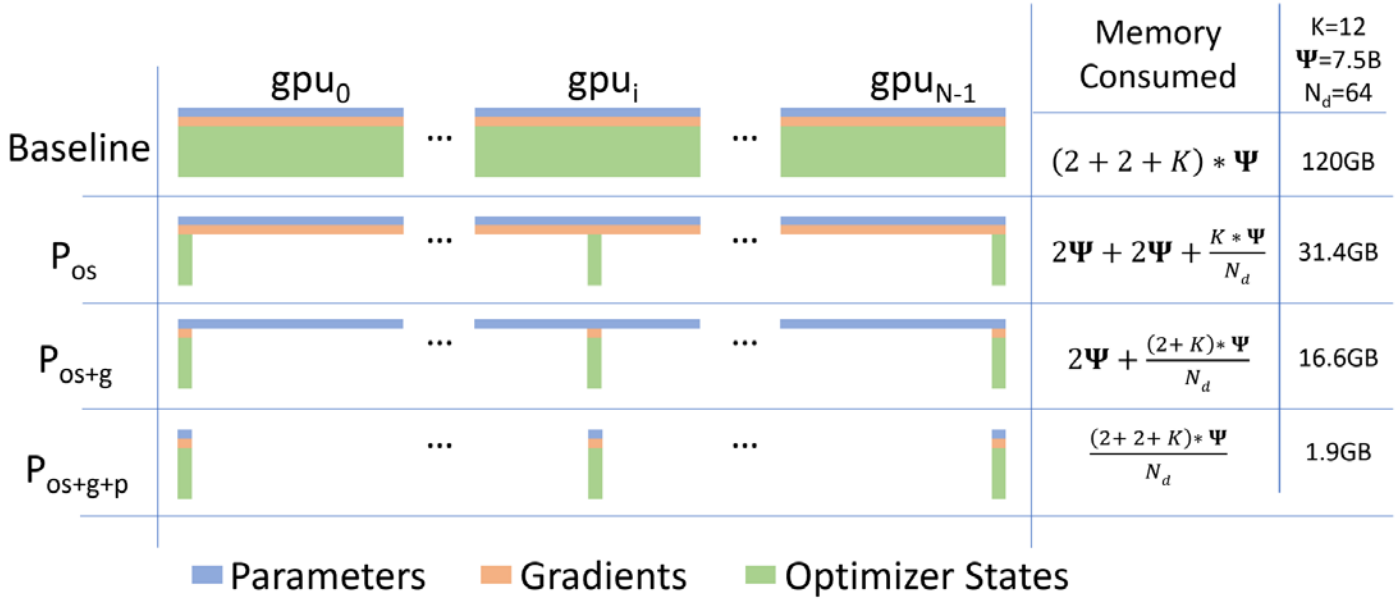


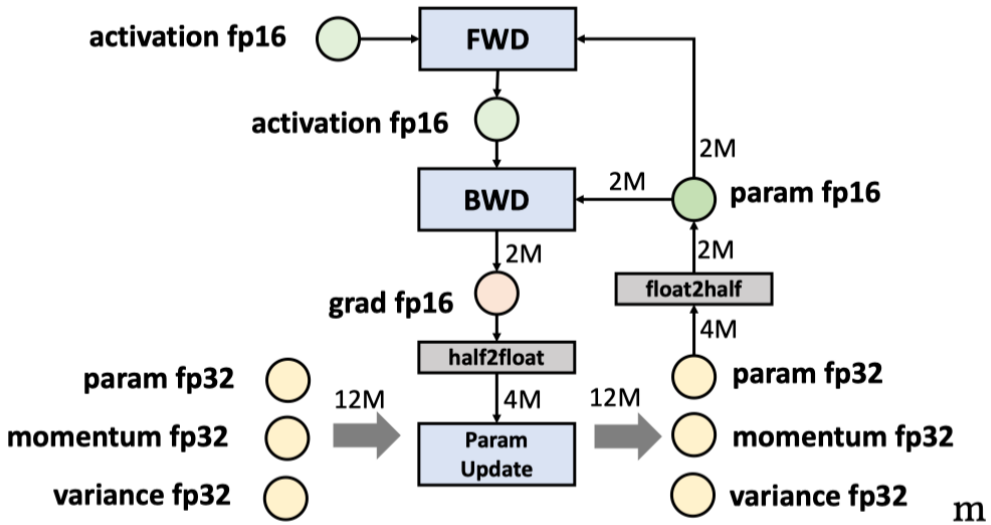
- Partitioning gradients across GPUs

dating parameters

ZeRO

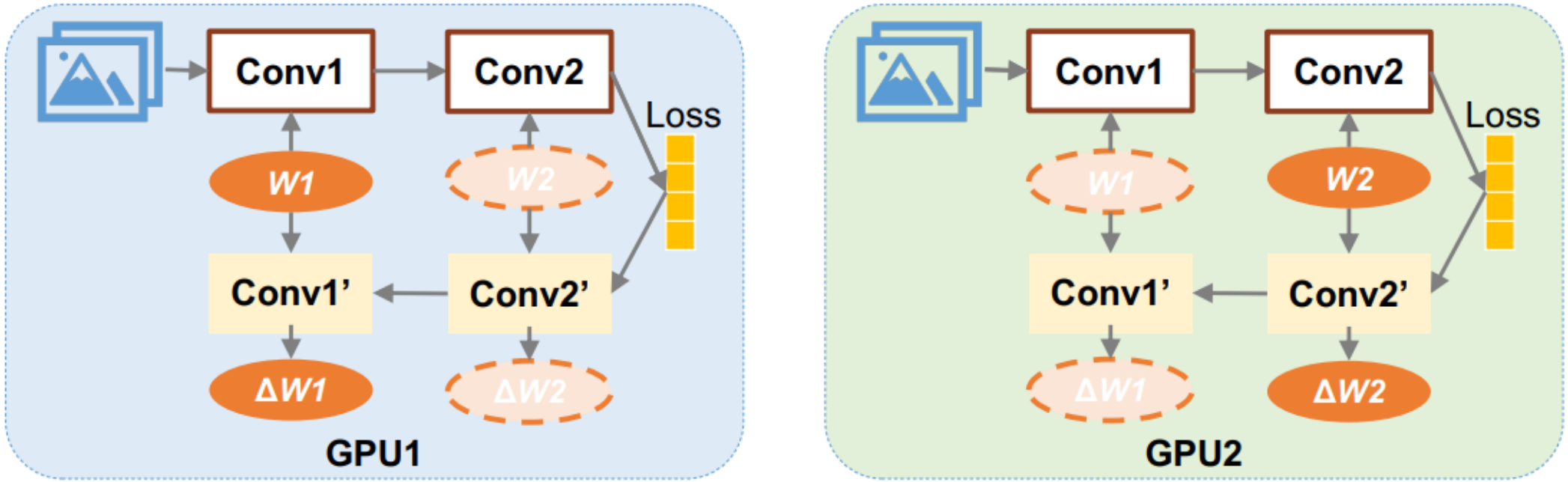
- Stage 3


gpu0
gpui
gpuN-1
Memory Consumed
K=12
Ψ=7.5B
Nd=64
Baseline
(2 + 2 + K) * Ψ
120GB
Pos
2Ψ + 2Ψ + K * Ψ / Nd
31.4GB
Pos+g
2Ψ + (2+ K)* Ψ / Nd
16.6GB
Pos+g+p
(2+ 2+ K)* Ψ / Nd
1.9GB
Parameters
Gradients
Optimizer States

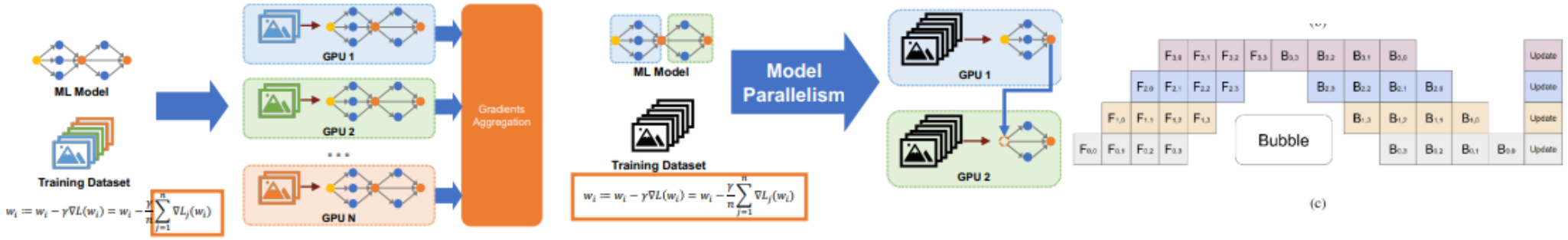

activation fp16
FWD
activation fp16
BWD
2M
2M
param fp16
2M
2M
grad fp16
float2half
4M
half2float
param fp32
momentum fp32
variance fp32
12M
4M
12M
Param Update
param fp32
momentum fp32
variance fp32
m

ZeRO

- In ZeRO, model parameters are partitioned across GPUs
- GPUs broadcast their parameters again during backward



Summary: Pros and Cons



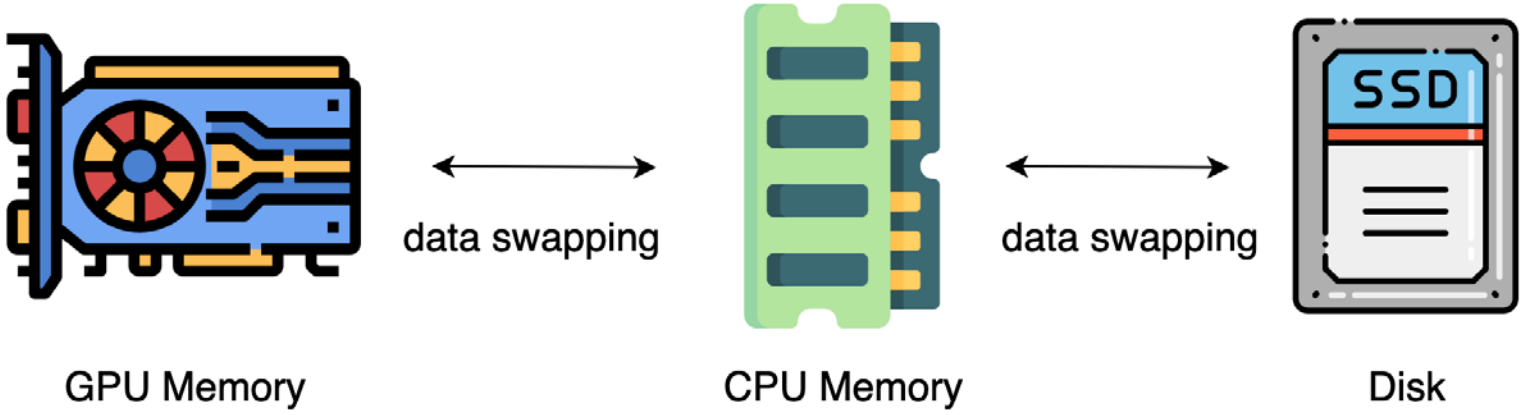
	Data Parallelism	Model Parallelism	Pipeline Parallelism
Pros	✓ Massively parallelizable ✓ Require no communication during forward/backward	✓ Support training large models ✓ Efficient for models with large numbers of parameters	✓ Support large-batch training ✓ Efficient for deep models
Cons	❖ Do not work for models that cannot fit on a GPU ❖ Do not scale for models with large numbers of parameters	❖ Limited parallelizability; cannot scale to large numbers of GPUs ❖ Need to transfer intermediate results in forward/backward	❖ Limited utilization: bubbles in forward/backward

Automatic Parallelism

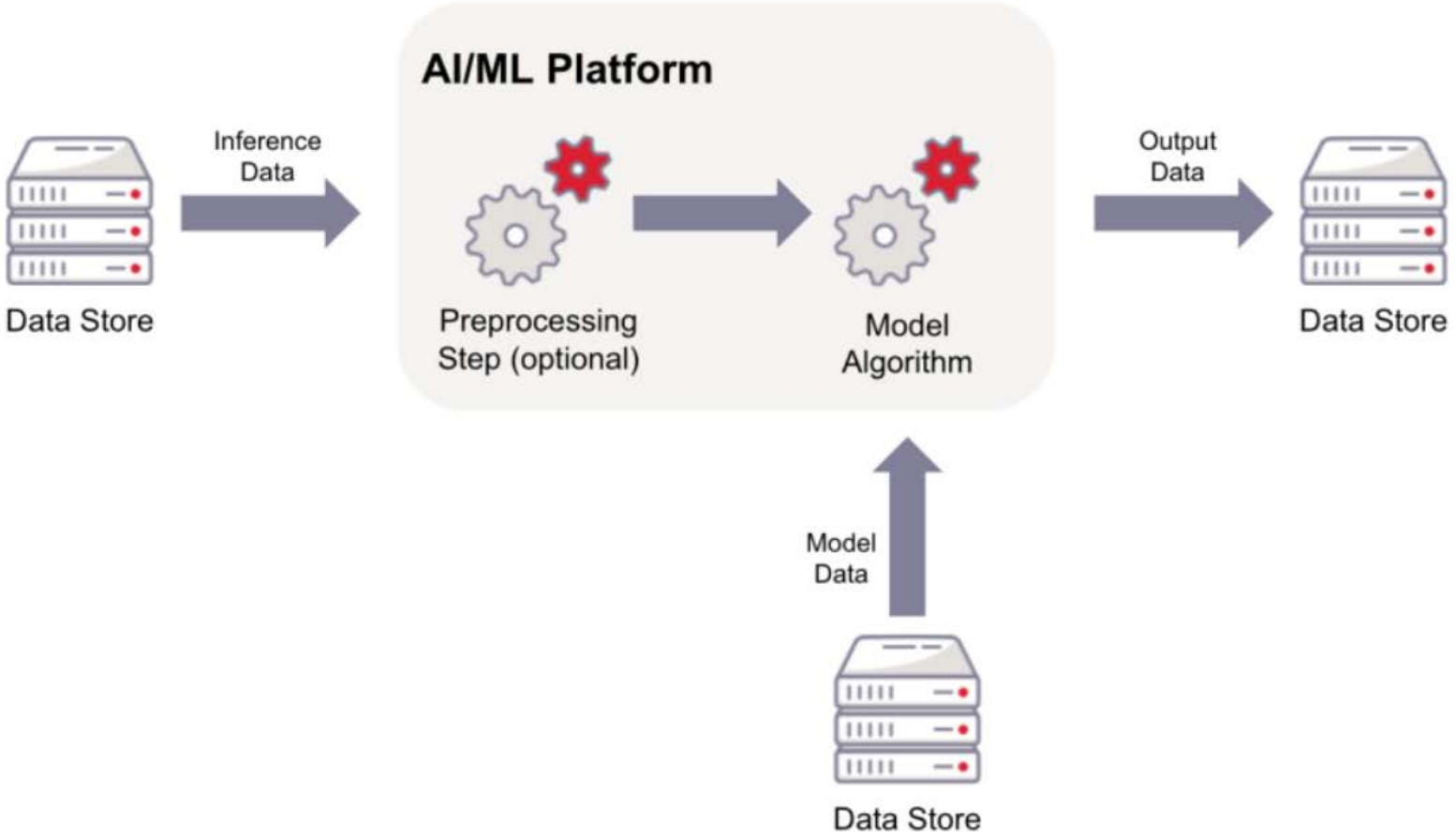
- Alpa: Jax
- Galvatron: Hetu
- Colossal-Auto: PyTorch

Heterogeneous training

- GPU + CPU: ZeRO-Offload
- GPU + NVME: ZeRO-Infinity
- GPU + CPU + NVME: PatrickStar/ColossalAI



Inference vs Training



Lab5

- Attention
- Transformer
- Operator Acceleration

Further Reading

- Stanford CS231n course, available online
- Dive into Deep Learning, https://d2l.ai
- A series of ML courses by Hung-yi Lee, available on YouTube
- Deep Learning by Ian Goodfellow et al.