Machine Learning Systems and Hardware

L1: Algorithm, Operator and Analysis

Hongxiang Fan

IMPERIAL

CATALOG

01
Algorithm Basis

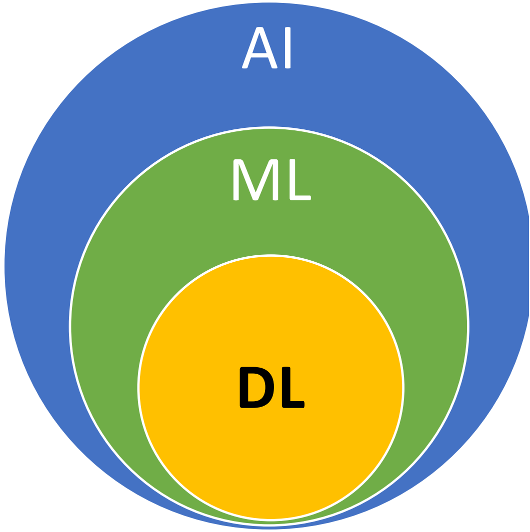
02
Convolutional Neural Networks

03
Attention-based Neural Networks

04
Roofline Model

DL Algorithm Basis: AI, ML and DL

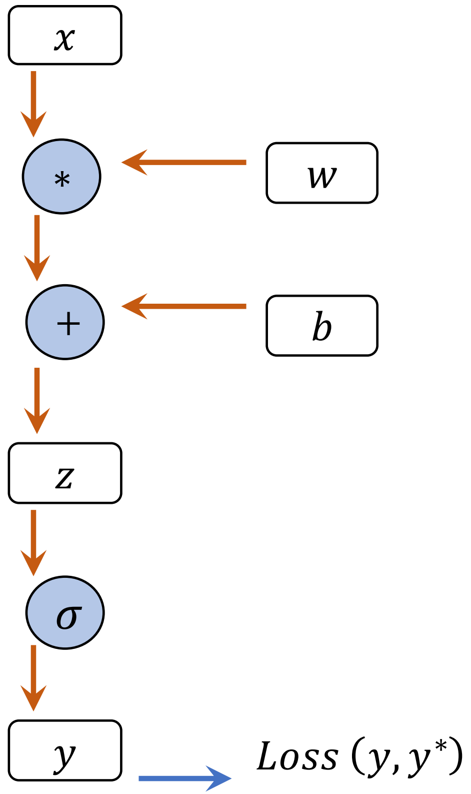
- Artificial Intelligence (AI)
  - ➢ Capability of computational systems to perform tasks associated with intelligence
  - ➢ Learning, reasoning, problem-solving, perception, and decision-making
- Machine Learning (ML)
  - ➢ Core AI approaches using data-driven methods
- Deep Learning (DL)
  - ➢ A subset of ML that employs Deep Neural Networks



DL Algorithm Basis

- Start from a simple (fundamental) example
  - Input: (x) with weights (w) and bias (b)
  - Compute: $z = w \ * x + b$
  - Activation: $y \ = \sigma \ (z)$
  - Loss function $Loss()$ with label y^*
- Forward pass
  - Compute output through linear combination and activation
  - Core AI approaches using data-driven methods
- Compute loss
  - Quantifies how far predictions are from ground truth.
  - Regression: Mean Squared Error (MSE)

$$\frac{1}{N}\sum(y \ - \ y^*)^2$$



DL Algorithm Basis

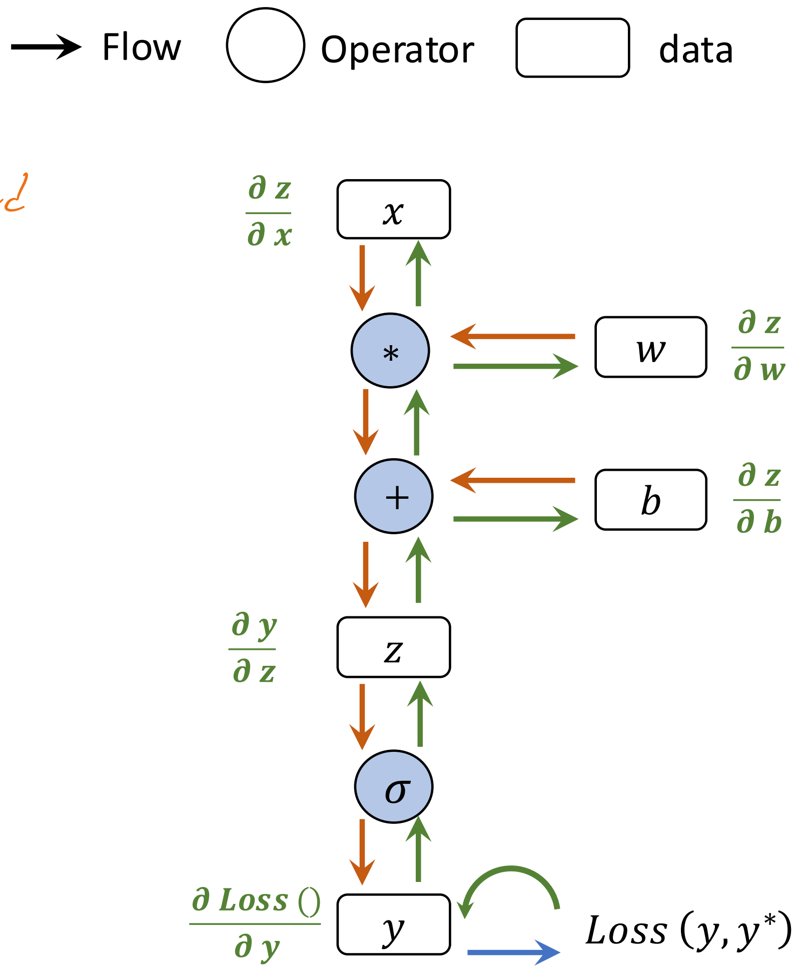
- Compute loss
  - Multi-Class Classification with output logits: $z \in \mathbb{R}^K$
  - Logits (z) to probabilities (p): softmax *this is not required*

$$p_i = softmax(z_i) = \frac{e^{z_i - m}}{\sum_{i=1}^{k} e^{z_i - m}},\ m = max_j(z_j)$$

  - Categorical Cross-Entropy (one-hot target y^*):

$$-\sum y_i^* \log(p_i)$$

- Backward pass
  - Calculate gradients via chain rule
  - Get local derivatives, e.g. $\frac{\partial\, Loss\,()}{\partial\, y}, \frac{\partial\, z}{\partial\, x}, \frac{\partial\, z}{\partial\, b}, \frac{\partial\, z}{\partial\, w}$
  - Multiplying local derivatives
  - $\frac{\partial\, Loss\,(.)}{\partial\, w} = \frac{\partial\, Loss\,(.)}{\partial\, y}\frac{\partial\, y}{\partial\, z}\frac{\partial\, z}{\partial\, w}$
  - $\frac{\partial\, Loss\,(.)}{\partial\, x} = \frac{\partial\, Loss\,(.)}{\partial\, y}\frac{\partial\, y}{\partial\, z}\frac{\partial\, z}{\partial\, x}$

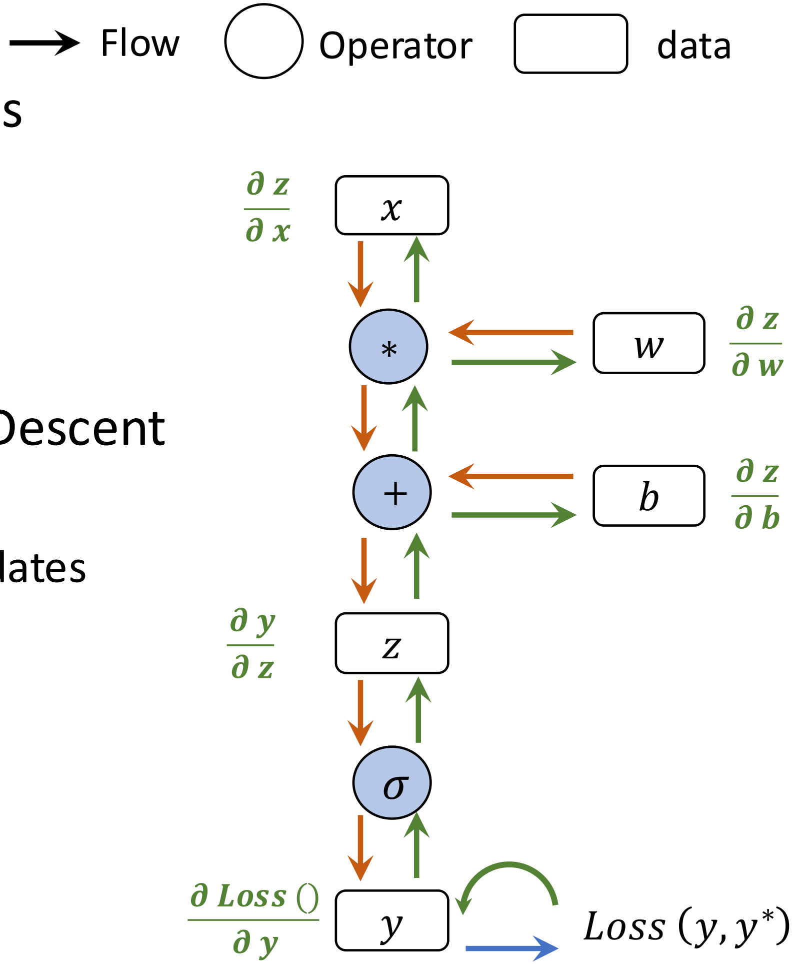


DL Algorithm Basis

- Gradient update
  - Adjust parameters in the direction that reduces loss
  - Assume learning rate is $\boldsymbol{\eta}$

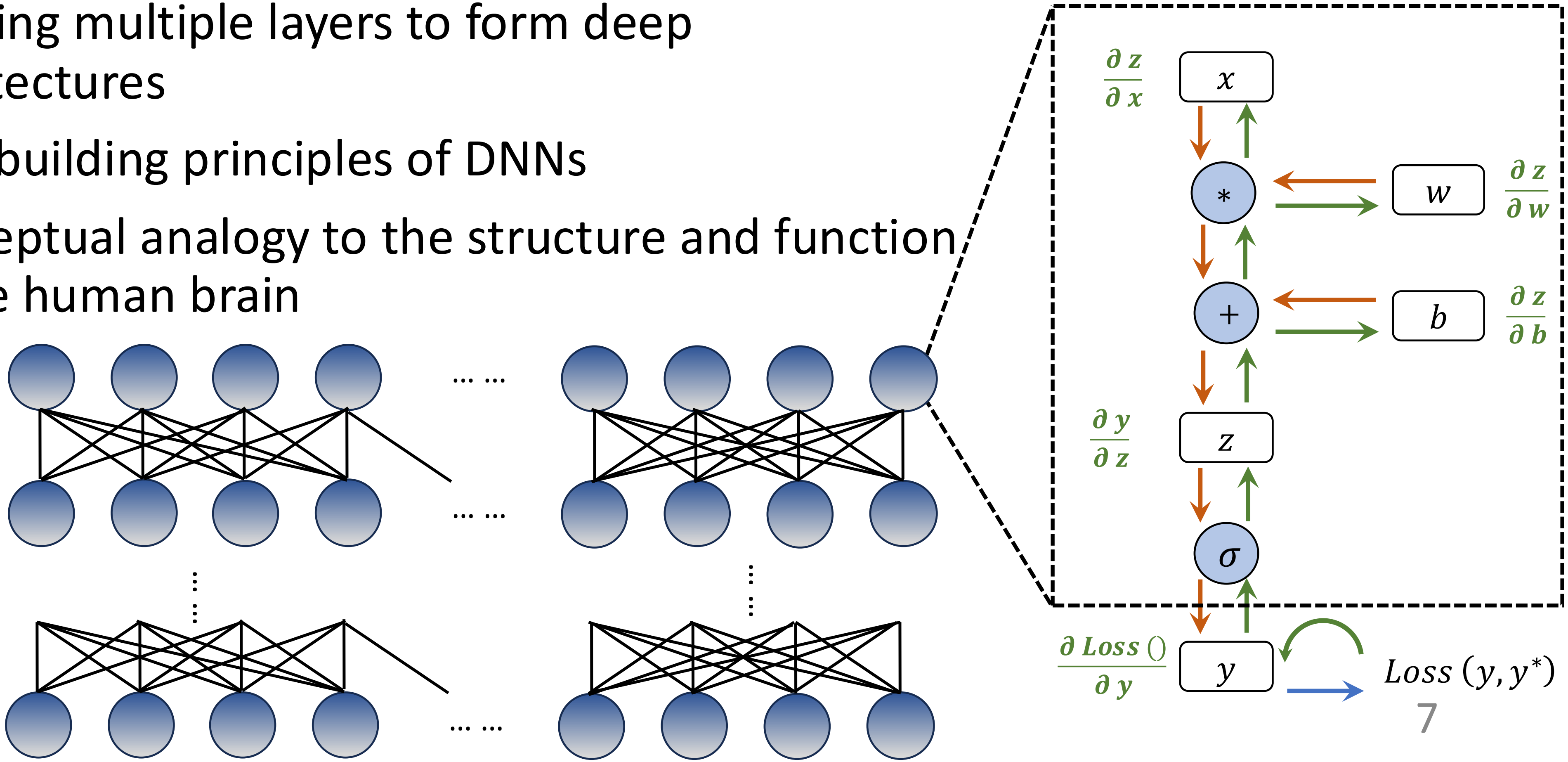
$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} - \boldsymbol{\eta} \frac{\partial \, \boldsymbol{Loss}\,(.)}{\partial \, \boldsymbol{w}}$$

  - Different variants to fit real scenarios constraints
  - Stochastic Gradient Descent, Mini-batch Gradient Descent
  - Adam (Adaptive Moment Estimation)
    - Momentum: moving average of gradients to smooth updates
    - Adaptive Learning Rate



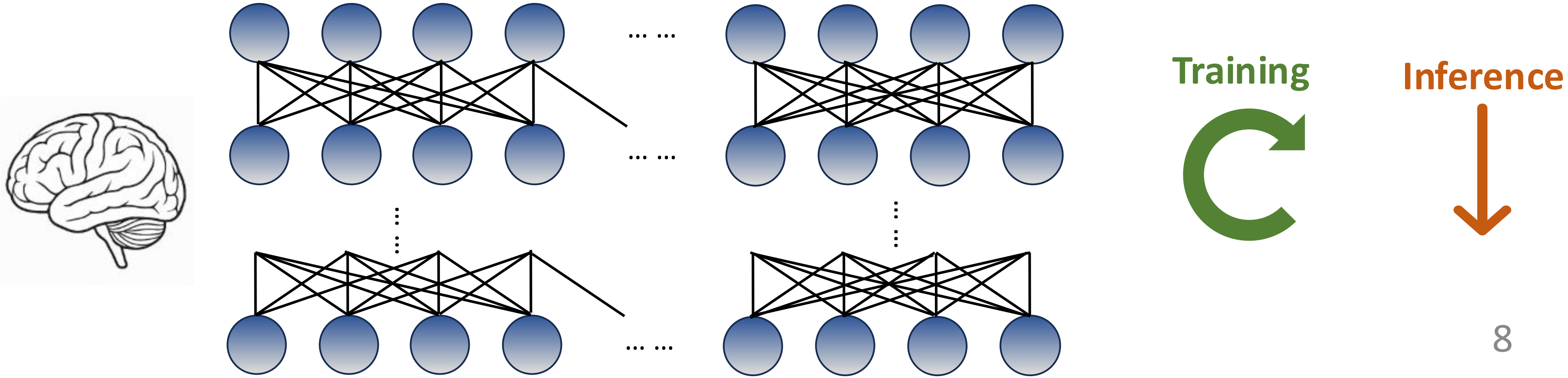
DL Algorithm Basis

- From a single computational unit (neuron) to large-scale parallel processing within a layer
- Stacking multiple layers to form deep architectures
- Core building principles of DNNs
- Conceptual analogy to the structure and function of the human brain



DL Algorithm Basis

- Two primary stages: Training and Inference
- Training stage: DNNs learn parameters from training data
  - ➢Iteratively performs forward pass, backward pass, and gradient updates
  - ➢Modern NLP: pre-training, RLHF, fine-tuning
- Inference stage: DNNs generates predictions for unseen inputs
  - ➢Executes forward pass only on real-world data
  - ➢Also referred to as test-time execution (closely related to test-time scaling)



CATALOG

01
Algorithm Basis

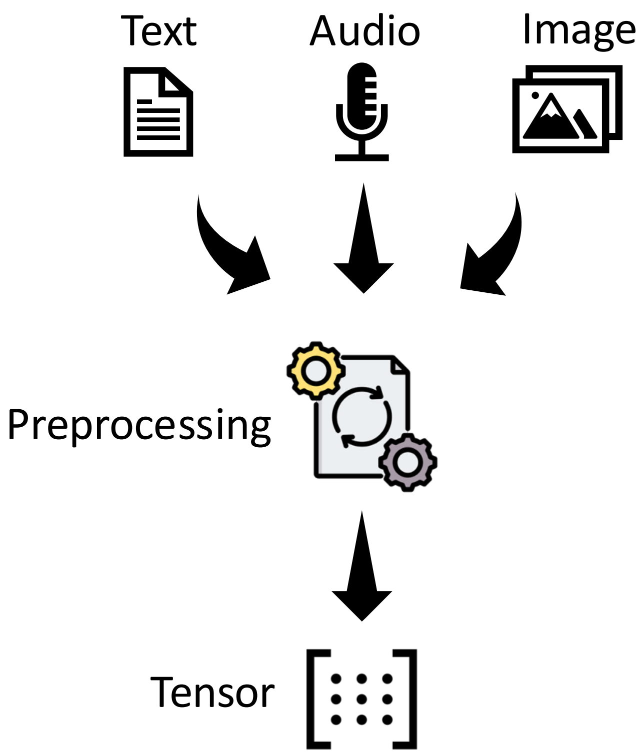
02
Convolutional Neural Networks

03
Attention-based Neural Networks

04
Roofline Model

Deep Neural Networks: Input Data

- Inputs can be different modalities:
  - Image: object recognition, segmentation, detection
  - Audio: speech recognition, audio event detection
  - Text: machine translation, summarization
- Can be single modality or multi-modal inputs
- After preprocessing, inputs are represented as tensors
  - Text: tokenized into numerical sequences
  - Audio: converted to spectrograms or waveform
  - Image: converted to pixel (RGB or grayscale)
- Tensor: multi-dimensional array
  - Scalar (0D), Vector (1D)
  - Matrix (2D) to higher dimensions (3D, 4D, …)

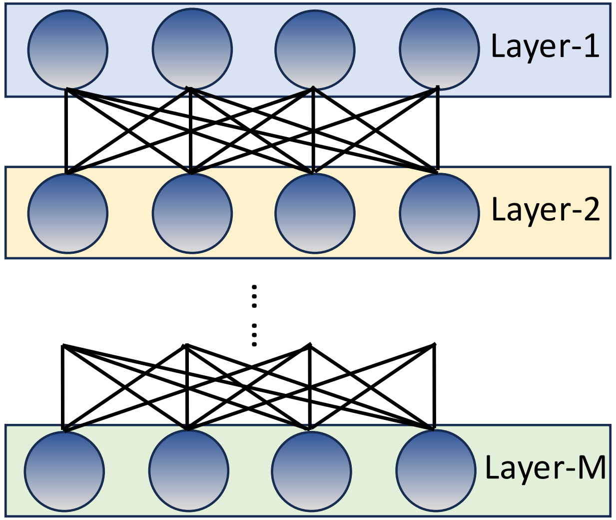


Deep Neural Networks: Fully Connected Layer

- Structure: each neuron in a layer connects to all neurons in the next layer
- Computation: matrix–vector/matrix multiplication
  - Batch size: B, input size: N_{in}, output size: N_{out}
  - $\boldsymbol{y} \in \mathbb{R}^{B * N_{in}}$, $\mathbf{w} \in \mathbb{R}^{N_{out} * N_{in}}$, $\boldsymbol{x} \in \mathbb{R}^{B * N_{in}}$, $\boldsymbol{b} \in \mathbb{R}^{N_{out}}$

$$\boldsymbol{y} = \boldsymbol{w} * \boldsymbol{x} + \boldsymbol{b}$$

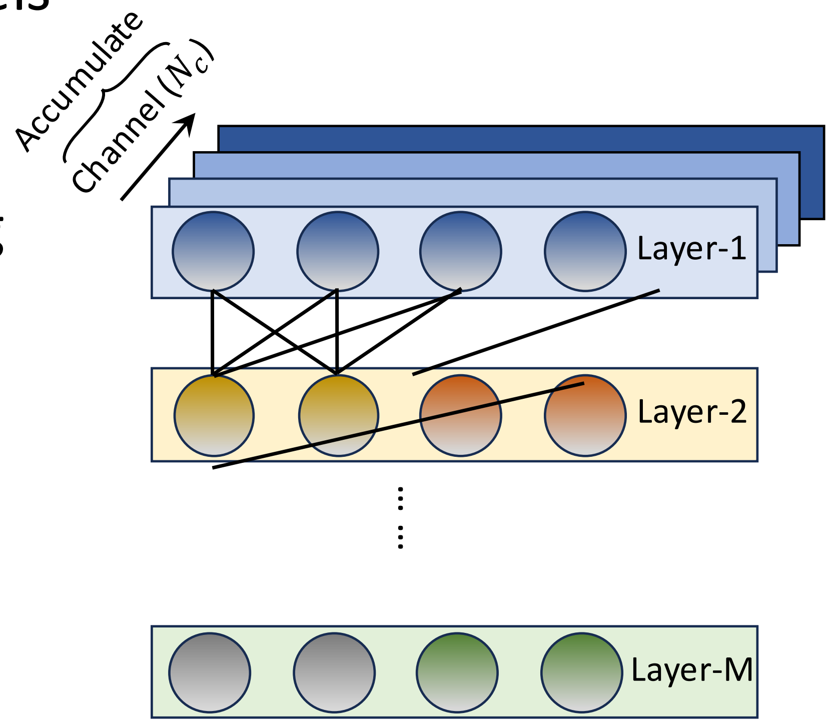
  - Floating point operations (FLOPs):
    - Multiplication counts: $\mathrm{B} * N_{out} * N_{in}$
    - Addition counts: $\mathrm{B} * N_{out} * (N_{in} - 1)$
    - Total FLOPs $\approx 2 * \mathrm{B} * N_{out} * N_{in}$



- Application: Classification layers for image/audio/text…
- Limitations:
  - Poor spatial locality
  - Large parameter count and high computation

Deep Neural Networks: Convolutional Layer

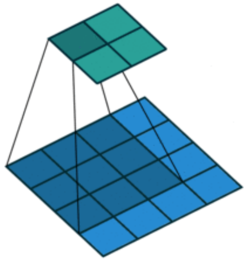
- Structure: Applies small learnable filters/kernels (e.g. 3x3 or 5x5) across local regions of input feature maps
- Computation: matrix multiplication in a sliding window manner
  - Input height: H_{in}, input width: W_{in}
  - Kernel size: $K_h * K_w$
  - Channel number: N_c, Filter number: N_f
  - Output height (**no padding, no strides**):
    $$H_{out} = H_{in} - K_h + 1$$
  - Output width (**no padding, no strides**):
    $$W_{out} = W_{in} - K_w + 1$$
  - Total FLOPs count:
    $$\approx H_{out} * W_{out} * N_f * (K_h * K_w * N_c)$$



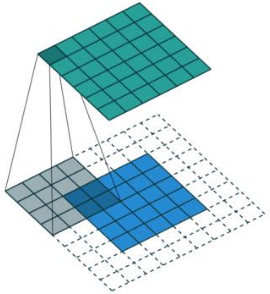
Deep Neural Networks: Convolutional Layer

- Blue matrices are inputs, and green matrices are outputs

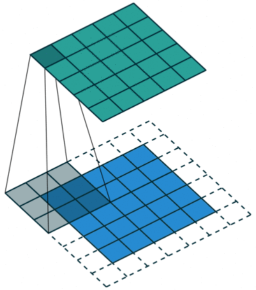
No padding, no strides



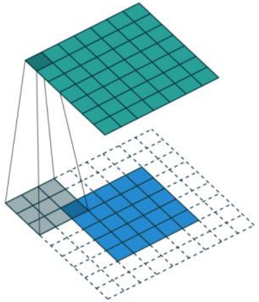
Arbitrary padding, no strides



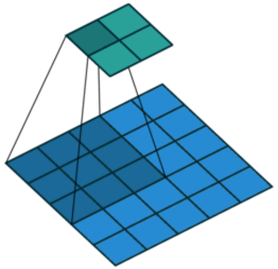
Same padding, no strides



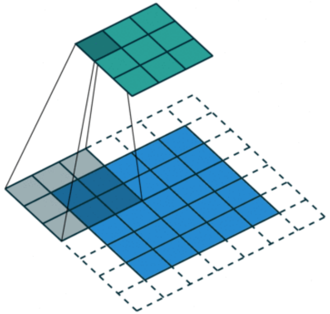
Full padding, no strides



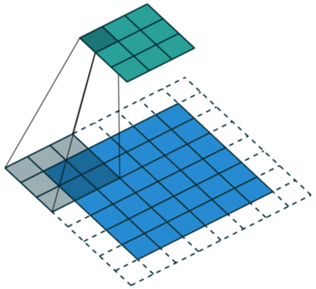
No padding, strides



No padding, strides



Padding, strides



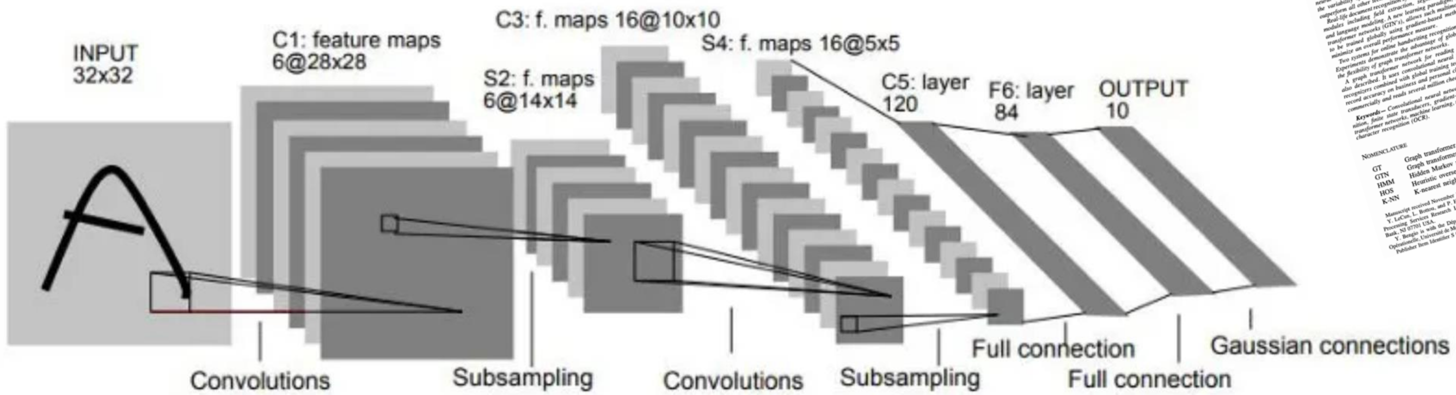
Source: https://github.com/vdumoulin/conv_arithmetic

Deep Neural Networks: Convolutional Layer

- Parameter efficiency: exploits spatial locality
- Translation invariance: kernels are applied across all spatial positions.
- Application: computer vision
  - ➢Image recognition: LeNet, VGG
  - ➢Object detection: YOLO

➢LeNet-5: Yann LeCun in 1998

LeNet-5 Architecture




Gradient-Based Learning Applied to Document Recognition

YANN LECUN, MEMBER, IEEE, LÉON BOTTOU, YOSHUA BENGIO, AND PATRICK HAFFNER

Invited Paper

NN Neural network.
OCR Optical character recognition.
PCA Principal component analysis.
RBF Radial basis function.
RS-SVM Reduced-set support vector method.
SDNN Space displacement neural network.
SVM Support vector method.
TDNN Time delay neural network.
V-SVM Virtual support vector method.

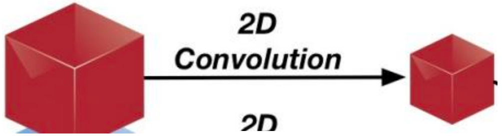
I. Introduction

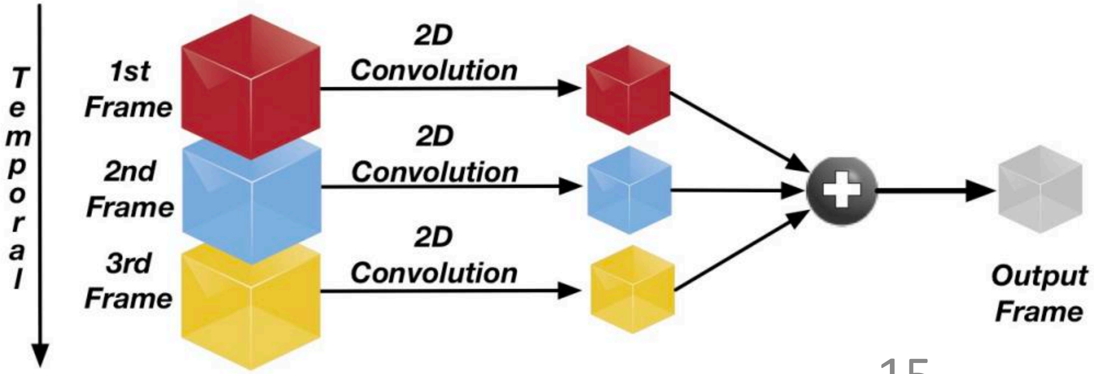
Source: LeCun, Y. et al. (1998). Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE, 86(11), 2278–2324.

Deep Neural Networks: 3D Convolutional Layer

- Structure: Applies 3D learnable filters/kernels (e.g. 3×3×3) across spatial + temporal (or volumetric) dimensions of input feature maps
- Computation: Extension of 2D convolution with an extra depth dimension
  - ➢Input Dimension: $H_{in} * W_{in} * D_{in}$
  - ➢Kernel size: $K_h * K_w * K_d$
  - ➢Channel number: N_c, Filter number: N_f
  - ➢Total FLOPs count:

$$H_{out} * W_{out} * D_{out} * N_f * (K_h * K_w * K_d * N_c)$$

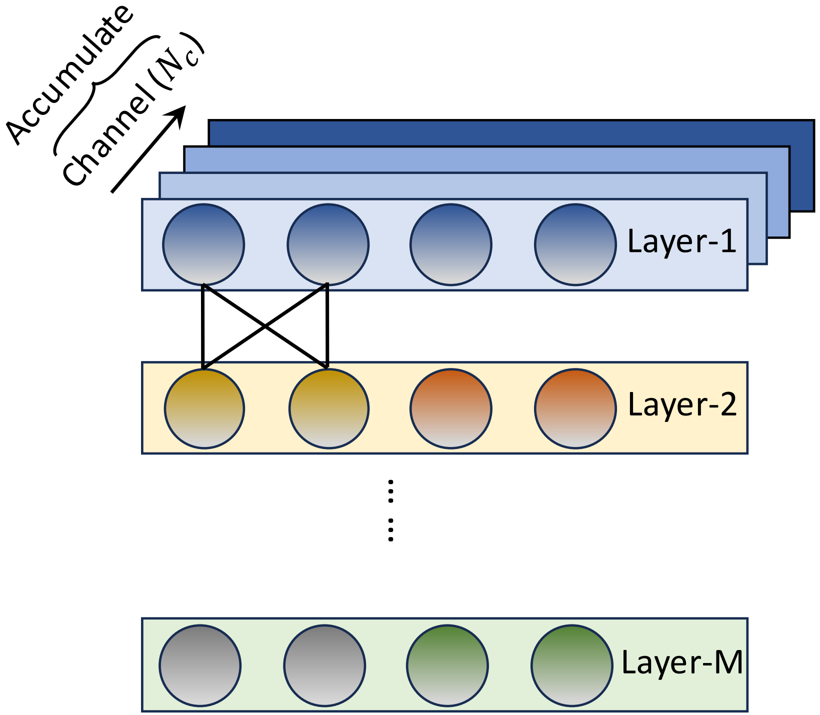




Source: Hongxiang, Fan. et al. F-E3D: FPGA-based Acceleration of an Efficient 3D Convolutional Neural Network for Human Action Recognition

Deep Neural Networks: Efficient Convolution Variants

- Depth-wise convolution:
  - ➢Kernel is not shared along the channel dimension
  - ➢Each input channel is convolved separately with its own filter
  - ➢Input channel number equals to output channel number

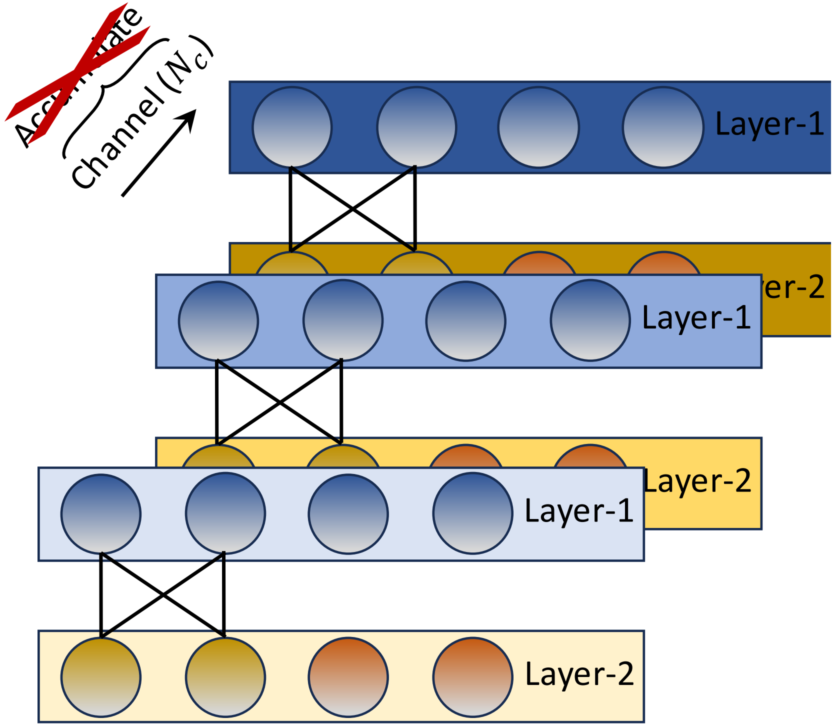


Deep Neural Networks: Efficient Convolution Variants

- Depth-wise convolution:
  - ➢ Kernel is not shared along the channel dimension
  - ➢ Each input channel is convolved separately with its own filter
  - ➢ Input channel number equals to output channel number
- Computation:
  - ➢ Total FLOPs count:

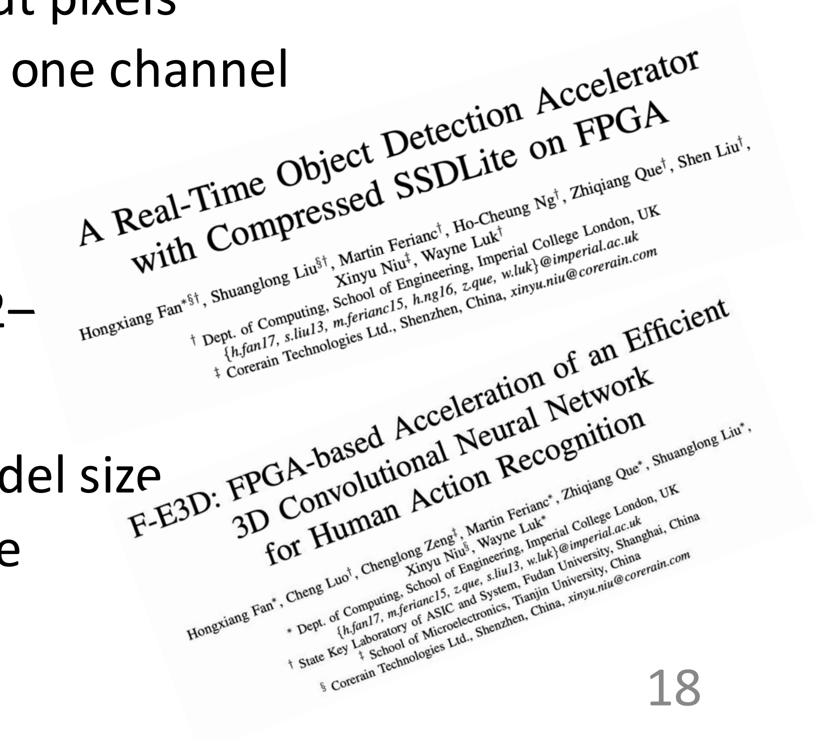
$$\approx H_{out} * H_{in} * N_f * (K_h * K_w \cancel{* N_c})$$

- Drawbacks:
  - ➢ Accuracy degradation
    - ○ Compensate by point-wise convolution (kernel 1*1)
    - ○ Depth-wise separable convolution (MobileNets from Google)
  - ➢ Hardware Inefficiency on GPU



Deep Neural Networks: Efficient Convolution Variants

- Limited Parallelism
  - ➢Standard Conv: Channels processed together → High thread occupancy
  - ➢Depth-Wise Conv: Independent channel → Small workloads per kernel
- Poor Weight Reuse
  - ➢Standard Conv: Weights shared across many input pixels
  - ➢Depth-Wise Conv: Each kernel weight is used for one channel
- Reduced FLOPs ≠ Proportional Speedup
  - ➢Depth-wise Conv on MobileNetV2 (edge GPU)
  - ➢FLOPs reduction: ~9×. Latency reduction: only ~2–
- Better for Mobile/Edge Devices
  - ➢Memory Constrained: Fewer weights reduce model size
  - ➢Compute Constrained: Parallelism gap less severe


A Real-Time Object Detection Accelerator with Compressed SSDLite on FPGA

Hongxiang Fan[*§†], Shuanglong Liu[§†], Martin Ferianc[†], Ho-Cheung Ng[†], Zhiqiang Que[†], Shen Liu[†], Xinyu Niu[‡], Wayne Luk[†]

[†] Dept. of Computing, School of Engineering, Imperial College London, UK
{h.fan17, s.liu13, m.ferianc15, h.ng16, z.que, w.luk}@imperial.ac.uk
[‡] Corerain Technologies Ltd., Shenzhen, China, xinyu.niu@corerain.com

F-E3D: FPGA-based Acceleration of an Efficient 3D Convolutional Neural Network for Human Action Recognition

Hongxiang Fan[*], Cheng Luo[†], Chenglong Zeng[‡], Martin Ferianc[*], Zhiqiang Que[*], Shuanglong Liu[*], Xinyu Niu[§], Wayne Luk[*]

[*] Dept. of Computing, School of Engineering, Imperial College London, UK
{h.fan17, m.ferianc15, z.que, s.liu13, w.luk}@imperial.ac.uk
[†] State Key Laboratory of ASIC and System, Fudan University, Shanghai, China
[‡] School of Microelectronics, Tianjin University, China
[§] Corerain Technologies Ltd., Shenzhen, China, xinyu.niu@corerain.com

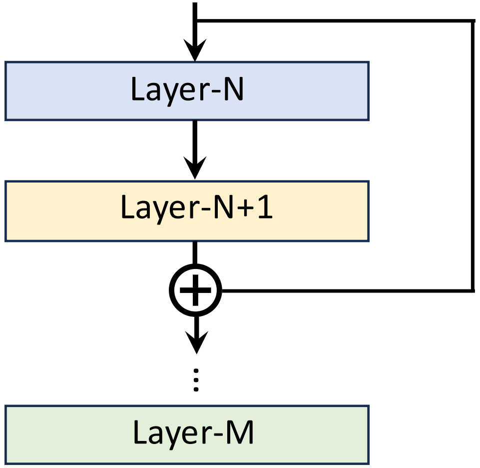
Deep Neural Networks: Residual Block

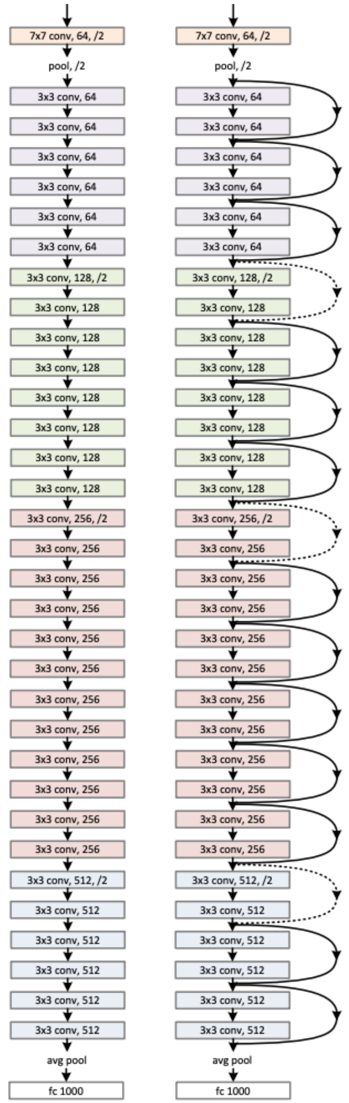
Source: Kaiming, He. et al. Deep Residual Learning for Image Recognition

- DNNs suffer from vanishing gradients:
  - ➢Difficult to train as depth increases
- Idea: Add a shortcut (skip connection)

$$y = \sigma(w * x + b) + \boldsymbol{x}$$

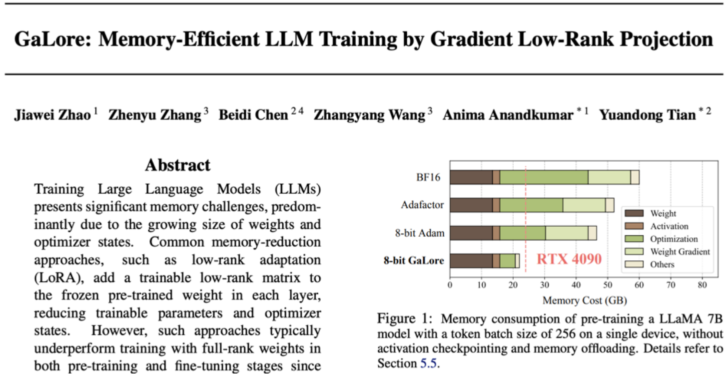
- Mitigates vanishing gradients:
  - ➢Enables very deep architectures
- Foundation of ResNet (50/101/152)
  - ➢"Deep Residual Learning for Image Recognition", Kaiming He et. al
  - ➢CVPR best paper in 2016
  - ➢Nearly 30k citations





Deep Neural Networks: Memory Perspective

- Key components of memory footprint
  - ➢ Model parameter
    - o Weights: # $Parameter \ * Size(data_{type})$: FP32, FP16, INT8, INT4
    - o Bias: Often negligible but still counted N_f
  - ➢ Intermediate activations
    - o Input & output feature maps for each layer, Batch size: B
    - o 2D → $B * H_{in} * W_{in} * \ N_c$
    - o 3D → $B * H_{in} * W_{in} * \ N_c * D_{in}$
  - ➢ Optimizer States (for training)
    - o Adam keeps 2× extra memory for momentum & variance terms



GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection

Jiawei Zhao[1] Zhenyu Zhang[3] Beidi Chen[2,4] Zhangyang Wang[3] Anima Anandkumar[*1] Yuandong Tian[*2]

Abstract

Training Large Language Models (LLMs) presents significant memory challenges, predominantly due to the growing size of weights and optimizer states. Common memory-reduction approaches, such as low-rank adaptation (LoRA), add a trainable low-rank matrix to the frozen pre-trained weight in each layer, reducing trainable parameters and optimizer states. However, such approaches typically underperform training with full-rank weights in both pre-training and fine-tuning stages since

Figure 1: Memory consumption of pre-training a LLaMA 7B model with a token batch size of 256 on a single device, without activation checkpointing and memory offloading. Details refer to Section 5.5.

Source: Jiawei, Zhao. et al. GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection

CATALOG

01 Algorithm Basis

02 Convolutional Neural Networks

03 Attention-based Neural Networks

04 Roofline Model

Deep Neural Networks: DNN Workload Trend

- Model workload composition is **evolving rapidly**
  - ➢Google TPU usage statistics (% by model type) provide real-world insights
  - ➢Reflects actual production deployment demands
- Observation-1: CNN/MLP share steadily declines over TPU generations
- Observation-2: Transformer models grow rapidly, becoming dominating

DNN Model	*TPU v1 7/2016 (Inference)*	*TPU v3 4/2019 (Training & Inference)*	*TPU v4 Lite 2/2020 (Inference)*	*TPU v4 10/2022 (Training)*
MLP/DLRM	61%	27%	25%	24%
RNN	29%	21%	29%	2%
CNN	5%	24%	18%	12%
Transformer	--	21%	28%	57%
(BERT)	--	--	*(28%)*	*(26%)*
(LLM)	--	--	--	*(31%)*

Source: TPU-v4

Deep Neural Networks: Recurrent Neural Networks

- Sequential data challenges:
  - In tasks like language, speech, or time series, the meaning of current input depends on past context
  - Can 3D CNN mitigate this? Yes, but not very efficient
  - Feedforward NN: no mechanism to remember previous information
- Key idea of Recurrent Neural Networks (RNNs):
  - A **hidden state** that propagates information over time steps
  - Enabling the model to “remember” previous inputs (“memory”)
- Example: Predicting the next word
  - Context : “<u>ML Systems and Hardware module is amazing, I am going to</u>”
  - Generate: “**spend more time on it this term**”

Deep Neural Networks: Recurrent Neural Networks

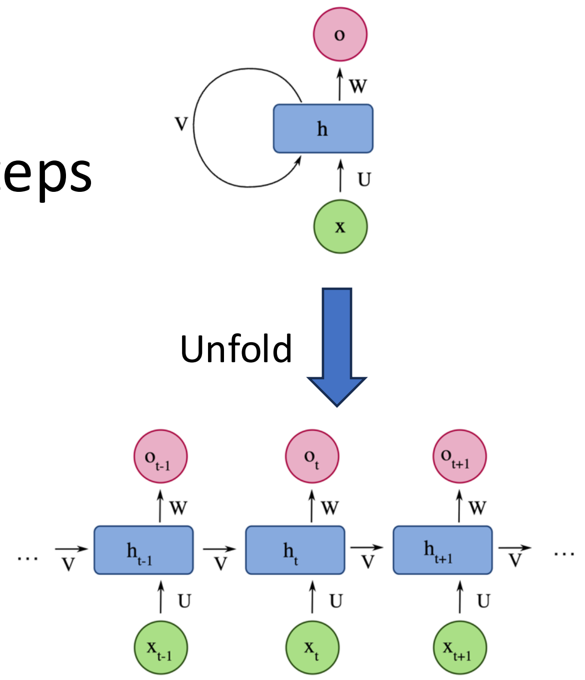
- Structure: Processes sequential data by maintaining a hidden state
  - ➢Updated at each time step
  - ➢Time-step: t, input: x_t, hidden state: h_t, output: o_t
- Computation: Matrix operations repeated across time steps
  - ➢Hidden state update:

$$h_t = \sigma(U * x_t + V h_{t-1})$$

  - ➢Output generation:

$$o_t = Softmax(W * h_t)$$

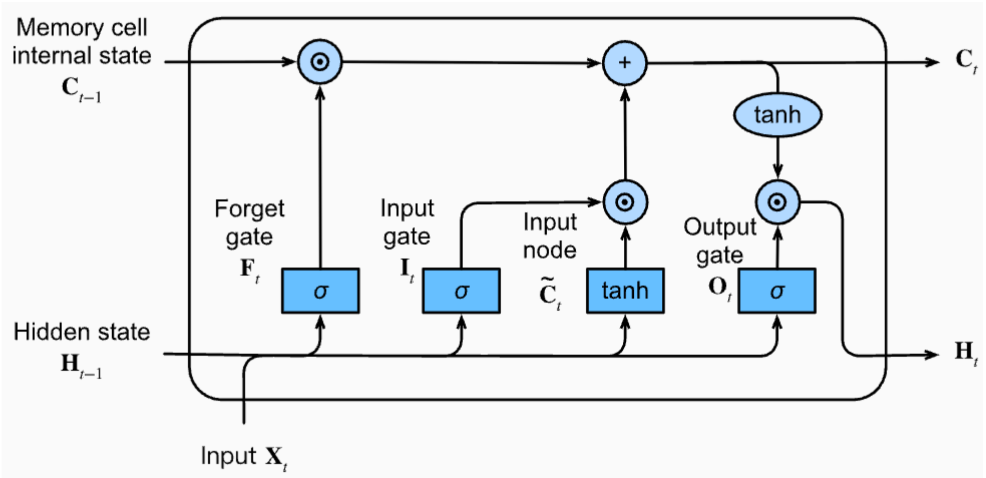
- Total FLOPs? (Tutorial Question)
- Limitations
  - ➢Short-term memory: Gradients vanish for long sequence
  - ➢Difficult to learn long-term patterns in language



Source: Wikipedia

Deep Neural Networks: Long Short-Term Memory (LSTM)

- Motivation: Addresses vanishing gradient problem in standard RNNs
- Structure: Maintains a cell state c_t that carries long-term memory
  - Uses gates to control information flow
- Computation: gates and states update
  - Forget gate: $f_t = \sigma(U_t * x_t + V_t h_{t-1})$
  - Input gate: $i_t = \sigma(U_i * x_t + V_i h_{t-1})$
  - Output gate: $y_t = tanh(U_y * x_t + V_y h_{t-1})$
  - Input node: $\hat{c}_t = \sigma(U_c * x_t + V_c h_{t-1})$
  - Cell state: $c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t$
  - Hidden state: $h_t = y_t \odot \tanh(c_t)$



Source: Dive into Deep Learning

- Total FLOPs? (Tutorial Question)
- Limitations: 1. Sequential compute 2. Limited long-range memory

Deep Neural Networks: Attention Mechanism

- “Attention Is All You Need” from Google
- Problem with RNN/LSTM
  - ➢Sequential processing → slow
  - ➢Long-range dependencies degrade
- Idea of Attention:
  - ➢Allow model to attend to any part of the sequence directly
  - ➢No need to pass all information through a single bottleneck hidden state
  - ➢First introduced in 2014 for machine translation
  - ➢“Neural machine translation by jointly learning to align and translate” by Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio.
- Foundational building block of generative AI

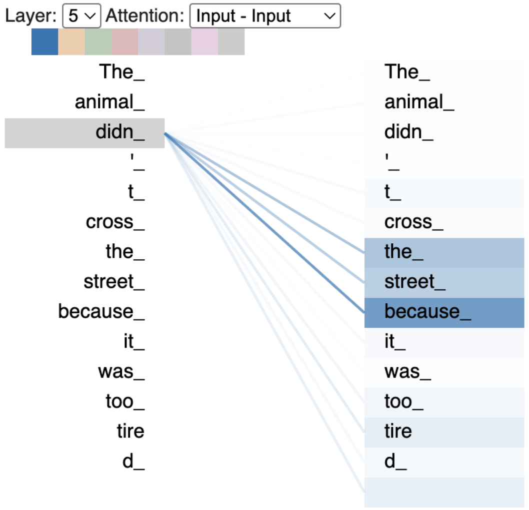
Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

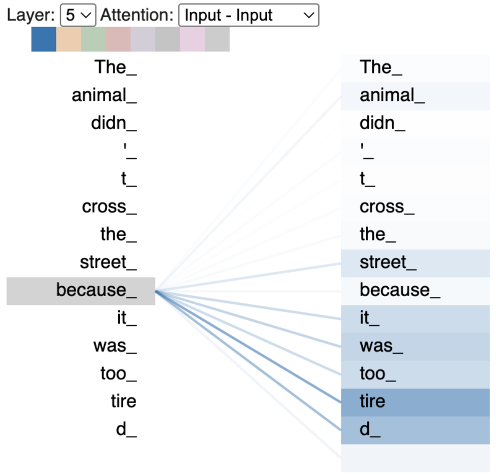
Deep Neural Networks: Attention Mechanism

- Each token to dynamically "focus" on other tokens when generating a representation
  - ➢Captured by Attention Map
  - ➢Code example: [Link1], other visualization: [Link2]
- Example:
  - ➢Word "*didn_*" attends strongly to "*because_*"
  - ➢Shows ability to model long-range dependencies without sequential bottleneck
- Mechanism:
  - ➢Each token has a Query (Q), Key (K), and Value (V).
  - ➢Similarity between Query and Keys determines attention weights
  - ➢Output is a weighted sum of Values

Source: Link1



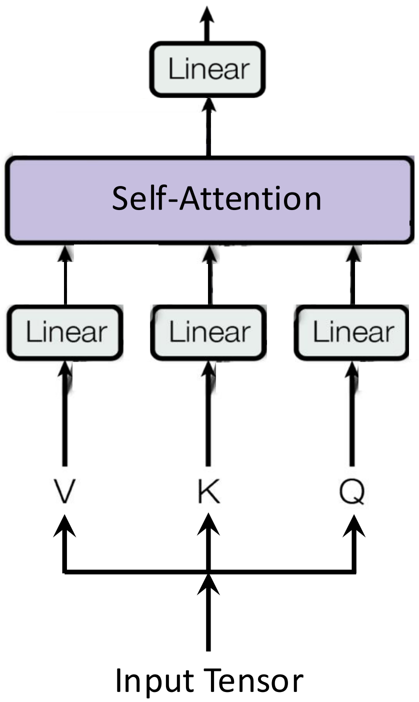
Attention map of "didn_"



Attention map of "because_"

Deep Neural Networks: Attention Mechanism

- Input Linear Projection:
  - Query: $Q = W_q * X$
  - Key: $K = W_k * X$
  - Value: $V = W_v * X$
- Attention Map Computation:
  - Score: $Score(Q, K) = QK^T$
  - Weights: $Score_{norm} = Softmax(Score(Q, K))$
  - Intermediate output: $Out_{interm} = Score_{norm} * V$
  - Each position can "look" at all other positions weighted by relevance
  - Shape of each tensor in Attention? (Tutorial Question)
- Output Linear Projection:
  - Final output: $Out_{final} = W_{out} * Out_{interm}$



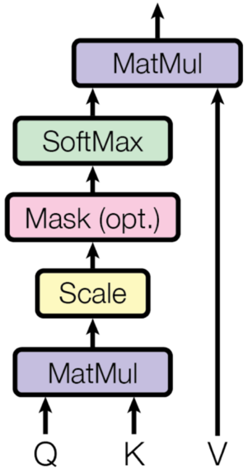
Deep Neural Networks: Scaled Dot-Product Attention

- Why scaling: Stabilizes gradients when $Q\backslash K\backslash V$ dimension becomes large
- Formulation
  - Internal dimension of K: d_k

$$Scaled\_Att(Q, K, V) = Softmax(\frac{Q * K^T}{\sqrt{d_k}}) * V$$

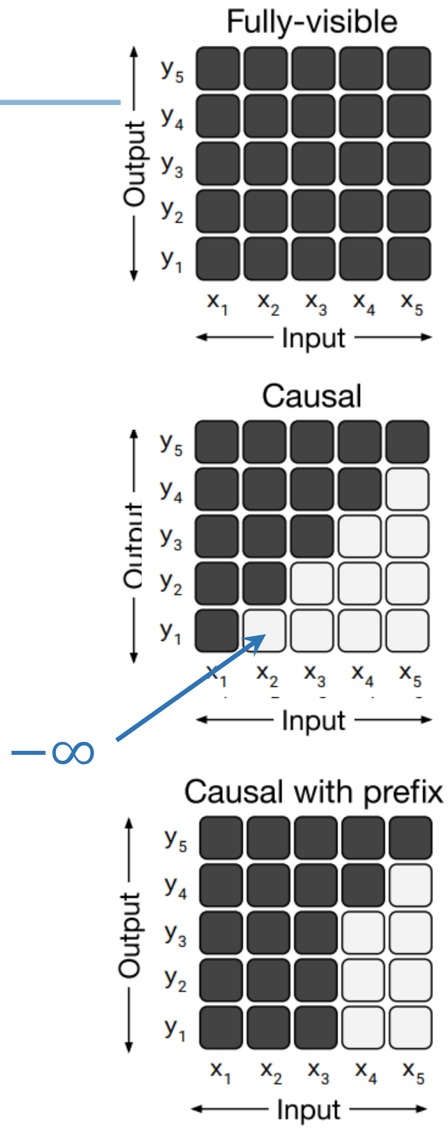
- Attention Mask
  - Prevents certain positions from influencing others
  - Applied before the softmax: set selected attention scores to -∞
  - Making the softmax output zero at those positions
  - Introduce flexibility to control different causality

Source: Attention is All You Need



Deep Neural Networks: Attention Mask

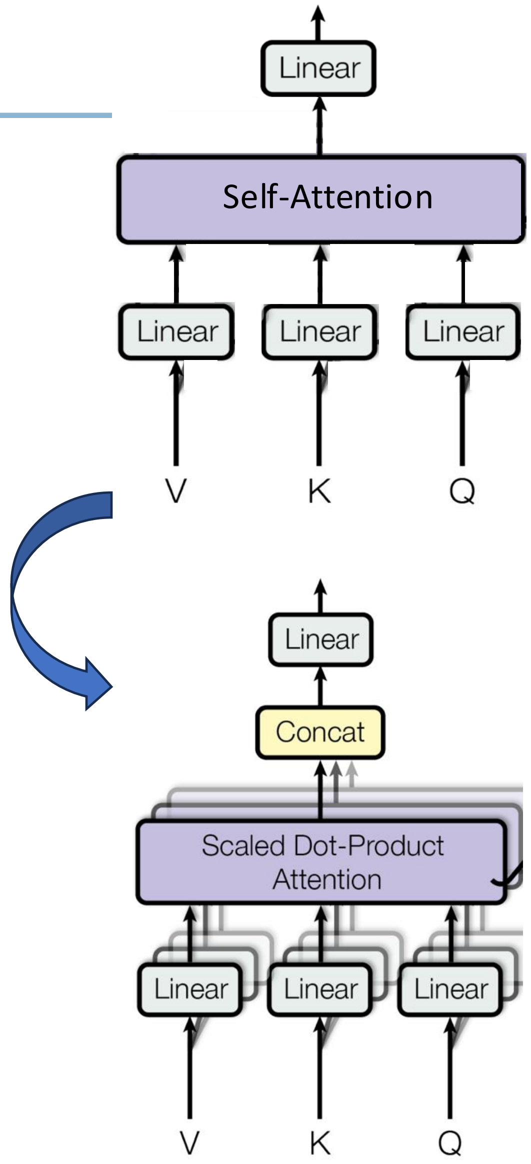
- Fully-visible pattern
  - ➢ Every token can attend to every other token
  - ➢ Used in layers where bidirectional context is available
- Causal pattern
  - ➢ Token at position i can only attend to positions $\leq i$
  - ➢ Prevents information “leak” from the future
  - ➢ Used in autoregressive models for left-to-right generation
- Causal with prefix
  - ➢ Initial tokens (prefix) visible to all positions (prompts/questions)
  - ➢ Subsequent tokens follow causal restriction.
  - ➢ Useful for instruction-following tasks where a shared context is available before generation



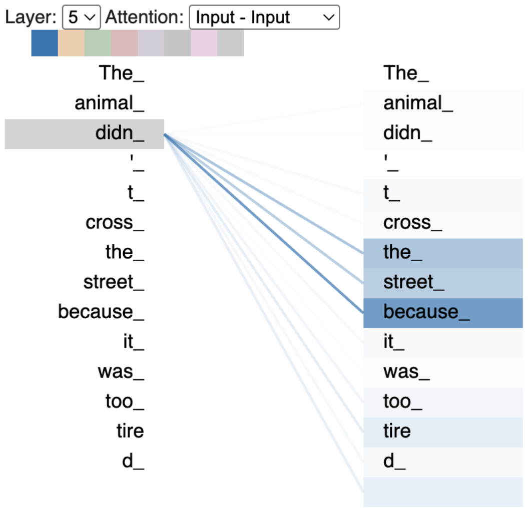
Source: Google T5

Deep Neural Networks: Multi-Head Attention

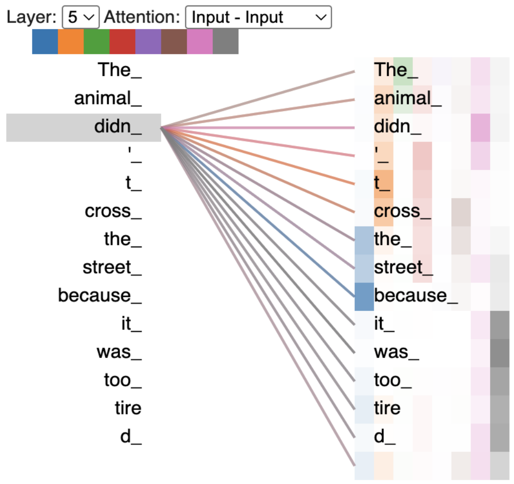
- Multi-Head Attention
  - ➢Multiple attention heads learn different relationships
  - ➢Concatenate outputs from all heads
  - ➢Improves representation capacity
  - ➢FLOPs and tensor shape? (Tutorial Question)
- Building block of Transformer (Attention-based NNs)



Source: Link1



Attention map of “didn_”



Attention map of different heads

Deep Neural Networks: Layer Normalization

- Multi-Head Attention
  - Multiple attention heads learn different relationships
  - Concatenate outputs from all heads
  - Improves representation capacity
  - FLOPs and tensor shape? (Tutorial Question)
- Building block of Transformer (Attention-based NNs)

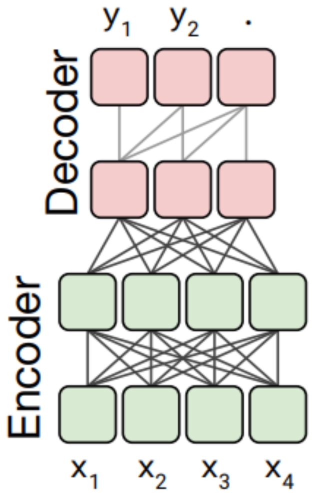
Deep Neural Networks: Encoder and Decoder

- Encoder Block
  - Attention mask: fully-visible (tokens attend to all positions)
  - Purpose: Capture bidirectional relationships and global context
  - Strength: Effective for understanding and representation
- Decoder Block
  - Attention mask: causal (each token attends only to previous tokens)
  - Purpose: Capture autoregressive dependencies
  - Strength: Generates target sequences one token at a time
- Different combinations
  - Different model architectures/variants (e.g., BERT, GPT)
  - Multi-modal architectures with multiple encoders (e.g., vision-language models)

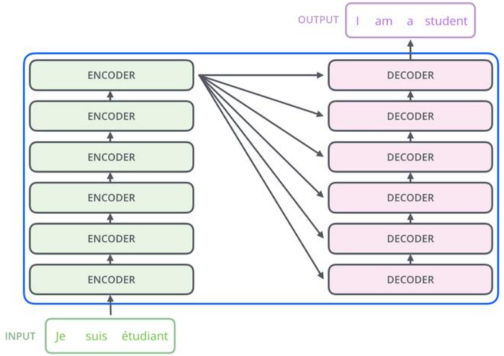
Deep Neural Networks: Encoder-Decoder Architecture

- Encoder for understanding, Decoder for generating
  - Divides work into two specialized modules (Google T5)
- Architecture Flow:
  - Encoder encodes the input → generates K_{enc} and V_{enc}
  - Decoder uses Q_{dec} (its own hidden state) and K_{enc} / V_{enc}
  - Output tokens are generated one at a time
- Cross-Attention:
  - Query: Comes from the decoder's hidden state
  - Keys and Values: Come from the encoder's output representations

$$Cross\_Att(Q_{dec}, K_{enc}, V_{enc}) = Softmax\left(\frac{Q_{dec} * K_{enc}^{T}}{\sqrt{d_k}}\right) * V_{enc}$$



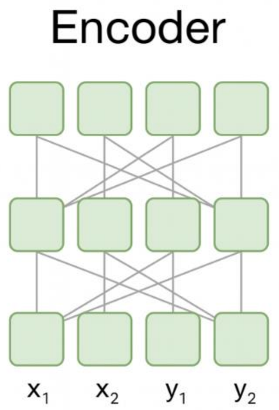
Source: Google T5



Source: "The Illustrated Transformer" from Jay Alammar

Deep Neural Networks: Encoder-Only Architecture

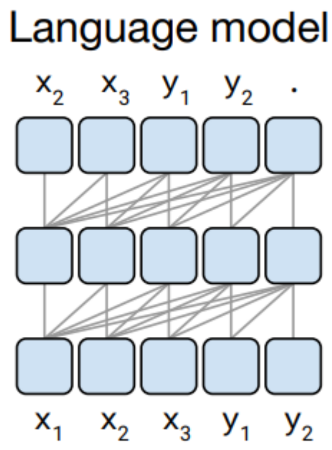
- Encoder-only for bidirectional understanding
  - No decoder: only a stack of encoder blocks
  - Used in masked language models (e.g., BERT)
- Architecture Flow:
  - Each token attends to all other tokens in the sequence
  - Bidirectional attention = deep context embedding
- Limitation
  - Cannot perform autoregressive generation
  - Requires full input sequence to start
  - Computationally expensive for very long sequences



Source: Google T5

Deep Neural Networks: Decoder-Only Architecture

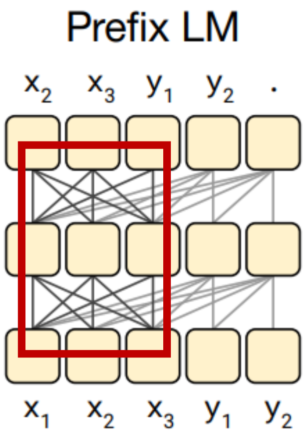
- Decoder-only for autoregressive generation
  - No encoder: a stack of masked self-attention layers
  - Used in autoregressive language models (e.g., GPT family)
- Architecture Flow:
  - All layers use causal masking to prevent access to future tokens
  - Tokens are generated one at a time, based only on prior tokens
- Limitation
  - Cannot use bidirectional context
  - No explicit conditioning on separate input sequence
  - Less effective for understanding tasks



Source: Google T5

Deep Neural Networks: Prefix Decoder-Only Architecture

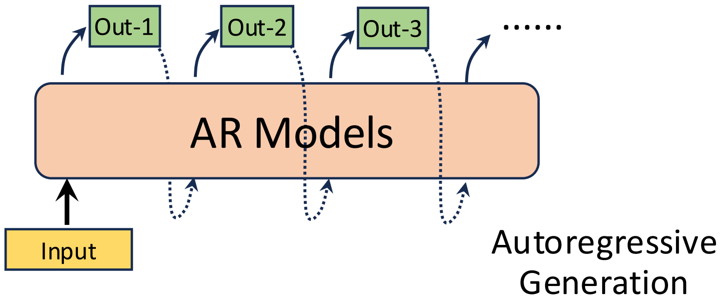
- Key Ideas:
  - A variant of decoder-only model with a prefix section that is fully visible to all tokens
  - Remaining tokens still use causal
- Architecture Flow:
  - Prefix tokens (system prompts, instructions) visible to all positions
  - Target tokens generated autoregressively using causal masking
- Advantages
  - Allows strong conditioning without modifying model weights significantly
  - Commonly used in existing LLMs

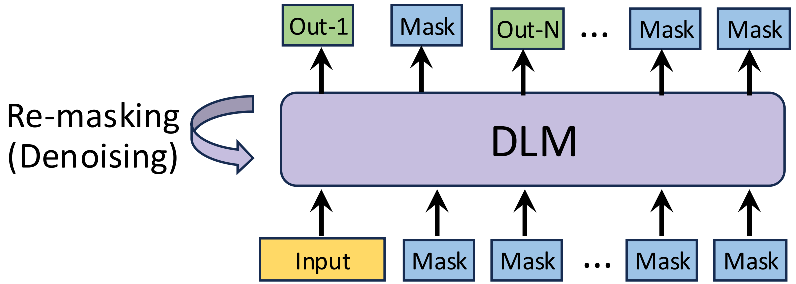


Source: Google T5

Deep Neural Networks: Diffusion Language Model

- Limitations of LLMs (Autoregressive Models):
  - System-level mitigation: batching and scheduling optimization (e.g. PD Split)
  - Algorithm-level mitigation: speculative decoding, parallel decoding
- Diffusion Language Model: Core Idea
  - Inspired by image diffusion models (e.g., Stable Diffusion)
  - Image Diffusion Models: Operate in continuous latent space
  - Diffusion Language Models: denoising process in discrete token space
  - Generate by iteratively **denoising** a sequence of noise tokens (from **mask** tokens)
  - Enables parallel decoding of entire sequence (non-autoregressive)





Deep Neural Networks: Diffusion Language Model

- Key Difference from Autoregressive LLMs
  - LLMs: generate one token at a time (left-to-right)
  - Diffusion LMs: generate all tokens simultaneously, then refine
  - Tradeoff: Better parallelism, but higher number of inference steps
- Industry: Gemini Diffusion achieves 1498 tokens per second
- Current Status
  - Early stage, but gaining traction
  - Requires new system support (e.g., token caching, kernel fusion)
  - Still behind LLMs in generation quality and efficiency
  - Active research by Google, Inception Labs, Bytedance, etc. (as of 2025)
  - Huge potential for multi-modality (e.g., MMADA)

Deep Neural Networks: Other Emerging Architectures

- Mamba (State Space Model)
  - Key idea: Uses selective state space models (SSMs) to model long-range dependencies with linear time complexity
  - Advantages:
    - Efficient for long sequences (1M+ tokens)
    - Lower memory footprint than Transformers
  - Applications: Long-context modeling, time-series, genomics
- Looped Transformer
  - Key idea: Reuses a smaller number of Transformer layers multiple times over the sequence
  - Advantages:
    - Lower parameter count and memory footprint
    - Maintains Transformer-level performance for certain tasks
  - Applications: Edge devices, low-resource inference

CATALOG

01 Algorithm Basis

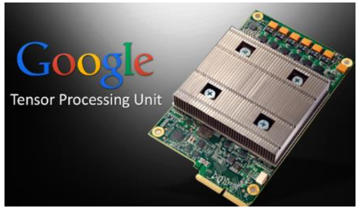
02 Convolutional Neural Networks

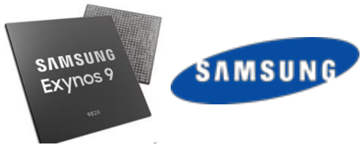
03 Attention-based Neural Networks

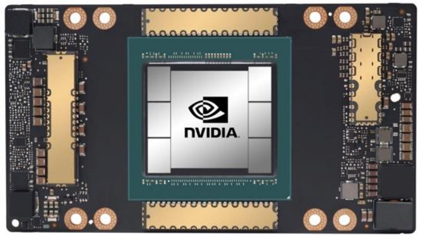
04 Roofline Model

Deep Neural Networks: Roofline Model

- Neural networks vary and keep evolving:
  - Encoder-only (e.g., BERT), decoder-only (e.g., GPT), diffusion models, Mamba, etc.
- Core computations are largely the same:
  - Matrix multiplications (e.g., attention: $QK^{\mathrm{T}}V$)
  - Element-wise ops (ReLU, GeLU, normalization)
  - Batch operations over sequences
- Goals of ML System Developers:
  - Implement and optimize performance (latency/throughput) for target platforms
  - Challenge: hardware diversity (GPU, TPU, NPU) with varied specs (compute/memory)

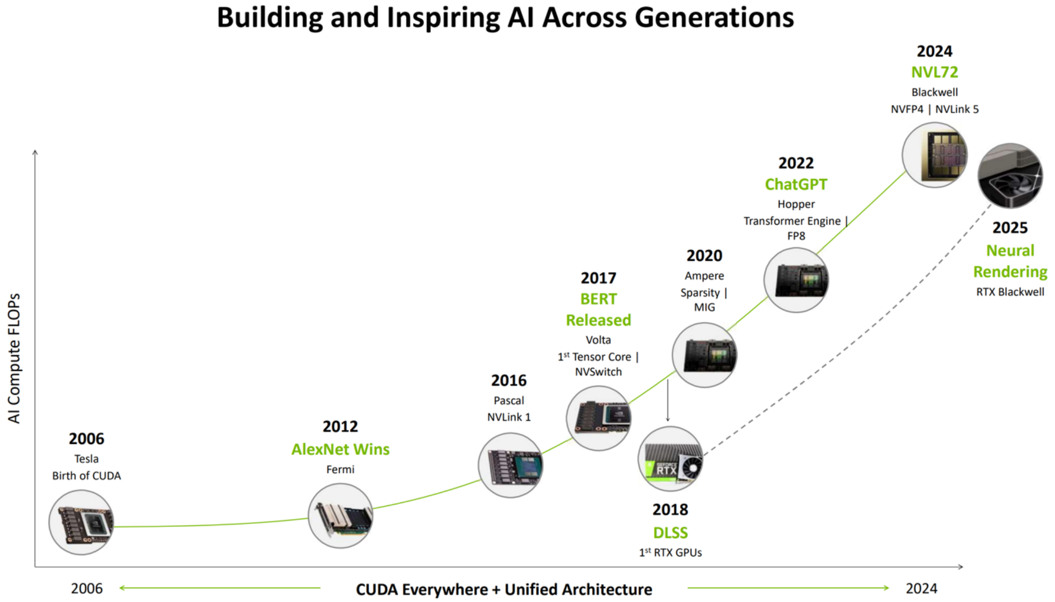






Deep Neural Networks: Roofline Model

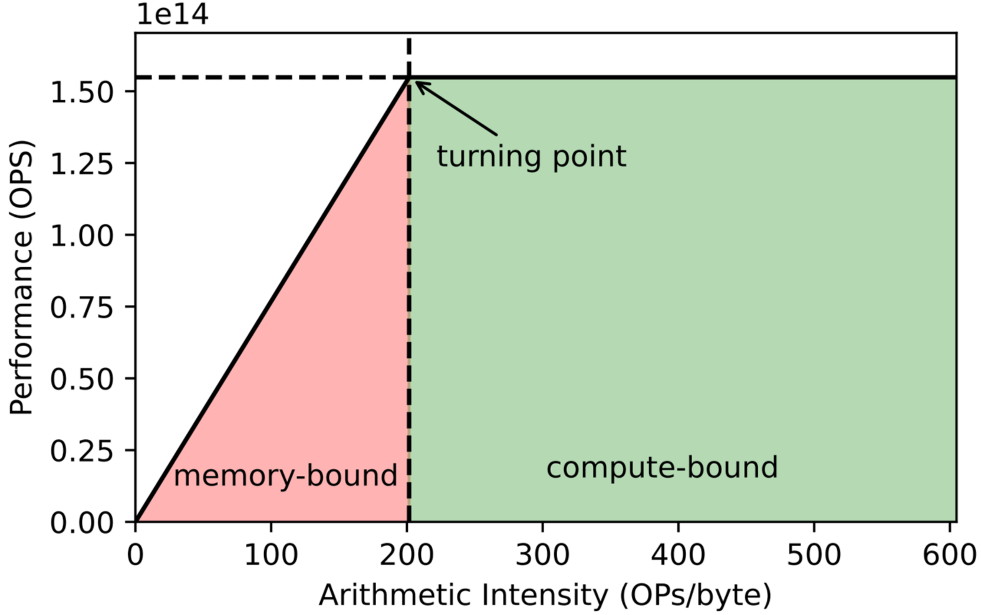
- Even within one vendor: NVIDIA from Tesla (2006) to Blackwell (2024)
  - Each generation improves memory bandwidth, compute FLOPs, and interconnect
- Need tools to:
  - Quantify bottlenecks (compute-bound vs memory-bound)
  - Guide optimization efforts
  - Compare hardware suitability



Source: hotchips'25

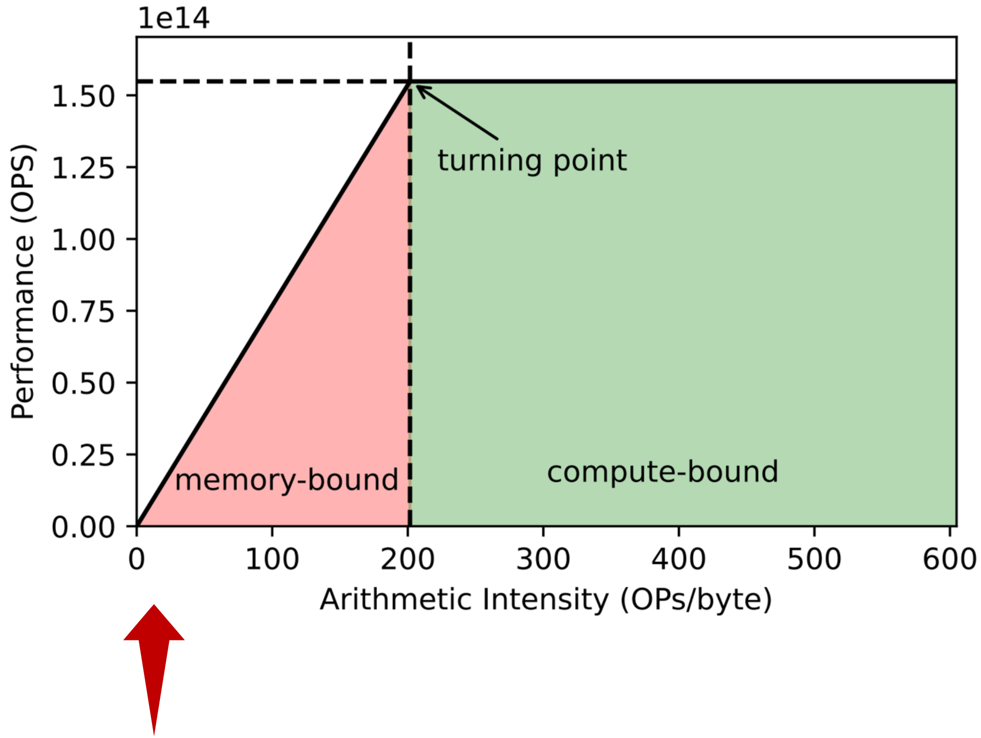
Deep Neural Networks: Roofline Model

- Roofline model:
  - X-axis: Operational/Arithmetic Intensity (OI = FLOPs/byte)
  - Y-axis: Performance (FLOPs)
  - Region 1: Memory-bound (limited by bandwidth)
  - Region 2: Compute-bound (limited by peak FLOPs)
  - Turning point: where bandwidth limit meets compute limit
- Key characteristics:
  - Sloped line: Bandwidth × OI
  - Horizontal line: peak achievable FLOPs



Deep Neural Networks: Roofline Model

- Memory-Bound Example:
  - GEMV (Matrix-Vector Multiplication)
  - Operation: $y = A \cdot x$
    - Matrix $A \in \mathbb{R}^{4096 \times 4096}$
    - Vector $x \in \mathbb{R}^{4096}$
    - Output $y \in \mathbb{R}^{4096}$
  - Operational counts:
    - One multiply-add counts as two operations
    - $FLOPs = 2 \times 4096^2 = 33.6 \times 10^6$
  - Memory access:
    - Weights and activations are 32-bit (4B)
    - $A: 4096 \times 4096 \times 4B = 67.1MB$
    - $x, y: 4096 \times 4B = 16KB$
  - Operational Intensity (OI): $\frac{33.6\times10^6}{67\times10^6} \approx 0.5 FLOPs/$Byte



Recap

- Algorithm Basis
  - Forward → Loss → Backward → Gradient Update
  - Training and Inference
- Convolutional Neural Network
  - Computational Complexity
  - Different Variants
- Attention-based Neural Network
  - RNN/LSTM → Attention
  - Attention Mask, Encoder/Decoder/Encoder-Decoder Architectures
- Roofline Model
  - Operational Intensity/Turning Point
  - Compute-Bound/Memory-Bound