6.5930/1

Hardware Architectures for Deep Learning

Popular DNN Models (Transformers), DNN Evaluation and Training

February 14, 2024

Joel Emer and Vivienne Sze

Massachusetts Institute of Technology
Electrical Engineering & Computer Science

Goals of Today's Lecture

- Popular DNN Models (cont'd)
  - Transformers
- Evaluate and compared DNN Models and Hardware
  - Key Metrics and Design Objectives
  - Datasets and benchmarks
- Training
- Recommended Reading: Chapter 2 & 3
  - https://doi.org/10.1007/978-3-031-01766-7

Popular DNN Models

Important Computational Properties

- Latency and energy affected by a variety of properties including
  - Number of operations that need to done sequentially
    - How many operations can be done in parallel?
  - Amount of data storage (memory capacity)
    - How much data is needed to perform compute?
  - Number of memory accesses (memory bandwidth)
    - How often does data change? How often can data be reused for multiple operations?

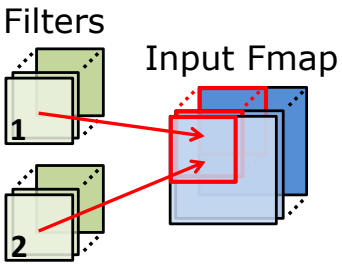
- More details on latency and energy in "**Key Metrics and Design Objectives"** section of today's lecture

Computation Properties of CONV Layers

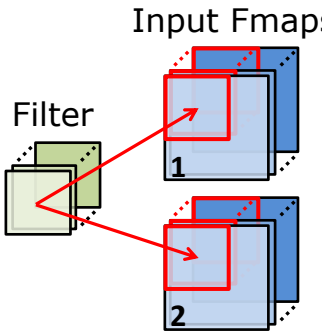
- Do operands change during inference?
  - Inputs **change** (dynamic)
  - Usually, weights **do not change** with input (static)
    - There are exceptions (e.g., Squeeze-and-Excitation)
- Complexity based on shape of layer
  - Number of MACs scales *linearly* with size of inputs
  - Storage:
    - Number of weights RSCM
    - Number of input activations HWC
    - Number of output activations PQM
- Sparse connection – no support outside of spatial dimensions of filter (R x S)
  - Fewer weights reduces amount of data storage required
  - Fewer MAC operations reduces number of operations
- Shared weights across different spatial locations (input)
  - Data reuse reduces number of memory accesses

Computation Properties of CONV Layers

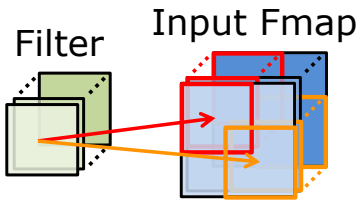
- Multiple forms of parallelism possible
  - Across M (apply multiple filters at the same time – reuse input feature maps)
  - Across N (filter multiple input feature maps at the same time – reuse filters)
  - Across R, S, C (compute MACs at same time, but need to accumulate – reuse both input activations and filter weights)



Fmap Reuse
(Activations)



Filter Reuse
(Weights)



Convolutional Reuse
(Activations, Weights)

- Shape can change per layer (R, S, C, M, H, W) - Flexible hardware required!

Convolution versus Attention Mechanism

- **Convolution**
  - Only models dependencies between spatial neighbors
  - Use sparsely connected layer to spatial neighbors; no support for dependencies outside of spatial dimensions of filter (R x S)
  - **Fixed** region of interest in input for given output
- **Attention**
  - "Allows modeling of [global] dependencies **without regard to their distance**" [**Vaswani**, *NeurIPS* 2017]
  - However, fully connected layer is too expensive; develop mechanism to bias "the allocation of available **computational resources towards the most informative** components of a signal" [**Hu**, *CVPR* 2018]
  - **Dynamically** select region of interest in input for given output

Transformers

- Built from Attention Mechanism [**Vaswani**, *NeurIPS* 2017]
- Widely used for natural language processing (e.g., GPT-3 [**Brown**, *NeurIPS* 2020]), since long dependencies between words exist
- Also used for other forms of data including
  - audio (e.g., AST [**Gong**, *Interspeech* 2021])
  - vision (e.g., ViT [**Dosovitskiy**, *ICLR* 2021])

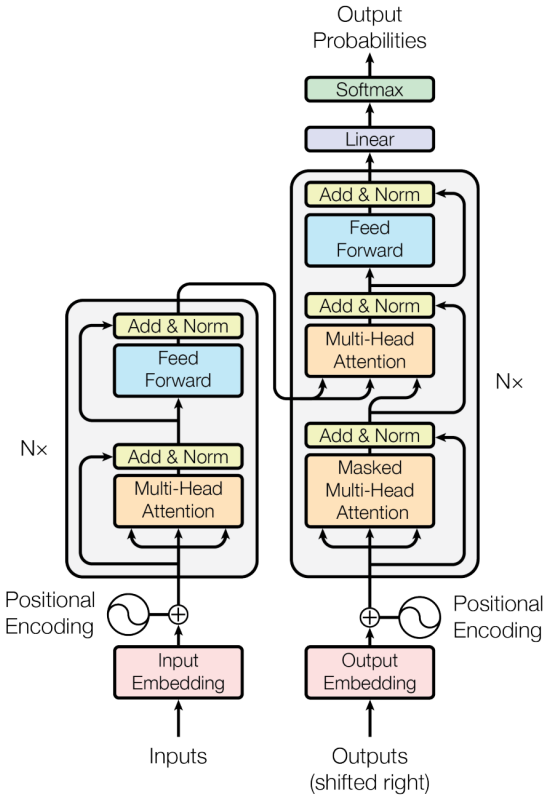
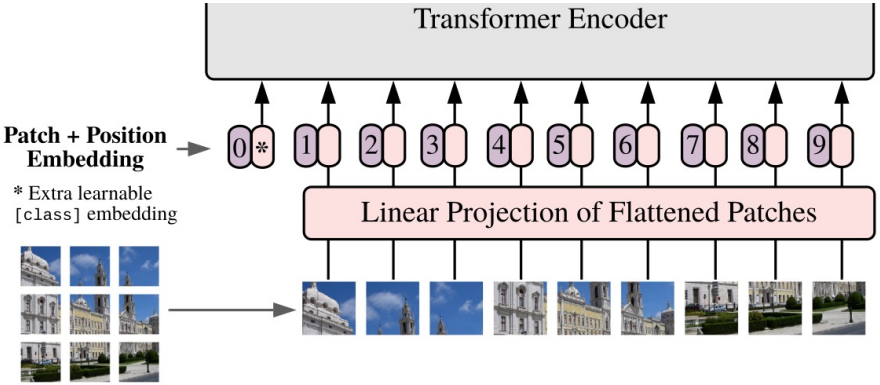


Image Source: [**Vaswani**, *NeurIPS* 2017]

Format of Input into Attention Mechanism

- Break input into chunks referred to as **tokens**
  - For a sentence, each word is a token
  - For an image, each patch of pixels is a token
  - For audio, each patch of spectrogram is a token
- Support variable sized input by processing a sequence of "tokens" one at a time



ViT [**Dosovitskiy**, *ICLR* 2021]

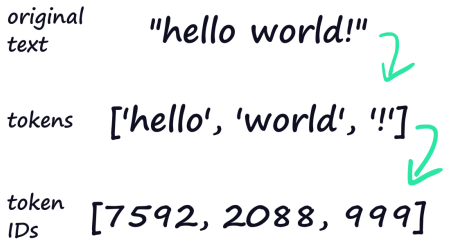
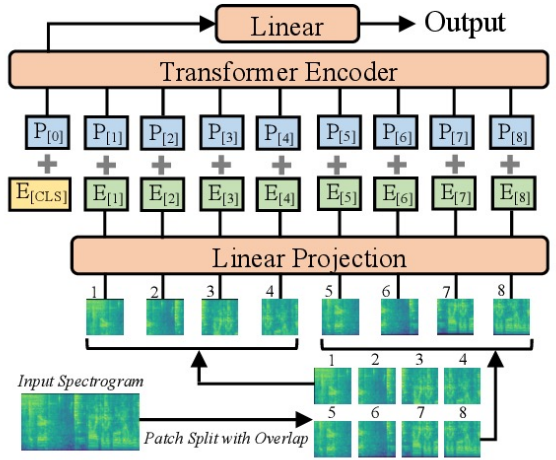


Image Source: https://towardsdatascience.com/why-are-there-so-many-tokenization-methods-for-transformers-a340e493b3a8



AST [**Gong**, *Interspeech* 2021]

Attention Mechanism: Overview

- Submit query (**q**) based on input token
- Assign **attention weights** for query (**q**) against each keys ($\mathbf{k_1}$, $\mathbf{k_2}$...$\mathbf{k_m}$) based on a function $\alpha(\mathbf{q}, \mathbf{k_i})$ that captures their dependencies
- Scale values ($\mathbf{v_1}$, $\mathbf{v_2}$...$\mathbf{v_m}$) associated with each key ($\mathbf{k_1}$, $\mathbf{k_2}$...$\mathbf{k_m}$) using **attention weights**
- Output sum of scaled values

$$\text{Attention}(\mathbf{q}, \mathcal{D}) \stackrel{\text{def}}{=} \sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i$$

(Note: **q**, $\mathbf{k_i}$, and $\mathbf{v_i}$ are vectors)

Can be viewed as database of key, value pairs

$\mathcal{D} \stackrel{\text{def}}{=} \{(\mathbf{k}_1, \mathbf{v}_1), \ldots (\mathbf{k}_m, \mathbf{v}_m)\}$ a database of m tuples of *keys* and *values*.

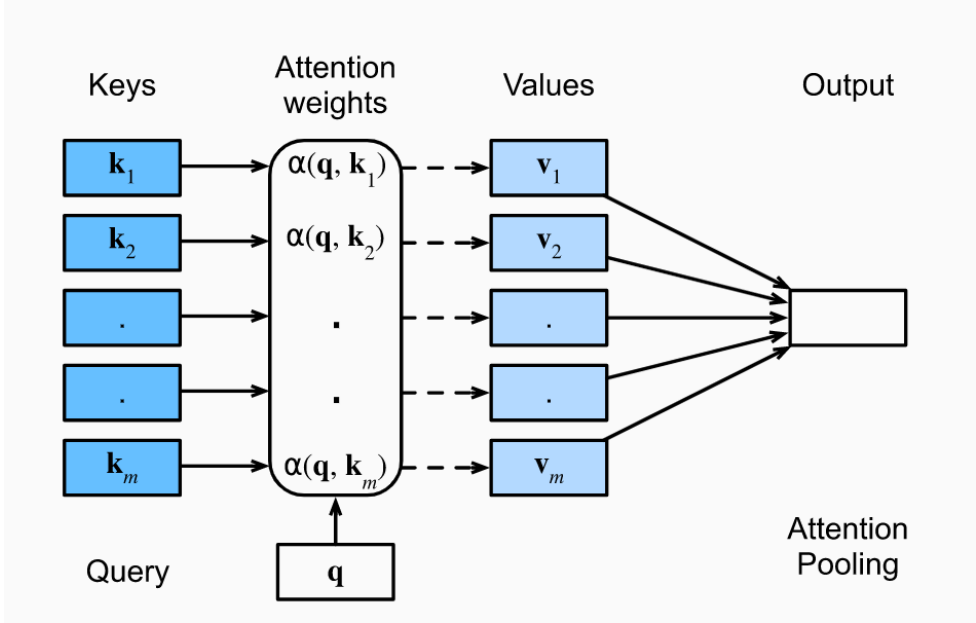


Image Source: https://d2l.ai/

Attention Mechanism: Attention Weights

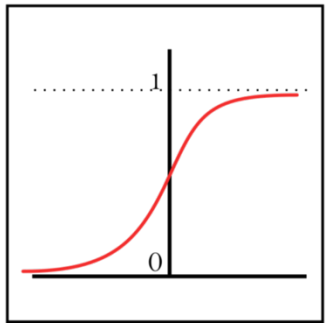
The attention weights are typically computed as follows

1. Compute the dot product between the query (**q**) and key (**k**) vectors, and scale by the length of the vectors (E). This is referred to as the **scaled-dot product attention** scoring function. $\frac{\boldsymbol{q}^T \boldsymbol{k}_i}{\sqrt{E}}$

2. Use **softmax** to scale the weight to be between 0 and 1

$$\alpha(\boldsymbol{q}, \boldsymbol{k}_i) = \text{softmax}\big(a(\boldsymbol{q}, \boldsymbol{k}_i)\big) = \frac{e^{\left(\frac{\boldsymbol{q}^T \boldsymbol{k}_i}{\sqrt{E}}\right)}}{\sum_{j=1} e^{\left(\frac{\boldsymbol{q}^T \boldsymbol{k}_i}{\sqrt{E}}\right)}}$$

Softmax



Modified from source: https://d2l.ai/ (in class we use 'E' rather than 'd' for length of vector)

Image: https://towardsdatascience.com/sigmoid-and-softmax-functions-in-5-minutes-f516c80ea1f9

Attention Mechanism: Multiple Tokens

- To process **multiple tokens** at a time within same sequences, vectors **q**, $\mathbf{k_i}$, and $\mathbf{v_i}$ are combined to form matrices **Q**, **K**, and **V**
- Therefore, the **scaled-dot product attention** becomes

$$\text{softmax}\left(\frac{\boldsymbol{QK}^T}{\sqrt{E}}\right)\boldsymbol{V} \in \mathbb{R}^{P\times F} \quad ,$$

where
- P is the number of queries (tokens) processed at time,
- E is the length of the queries and key vectors,
- F is the length of the values vector, and
- M is the number of key-value pairs in the database such that

queries $\mathbf{Q} \in \mathbb{R}^{P\times E}$ keys $\mathbf{K} \in \mathbb{R}^{M\times E}$ values $\mathbf{V} \in \mathbb{R}^{M\times F}$

- Key operation in the attention mechanism is **matrix multiplication**

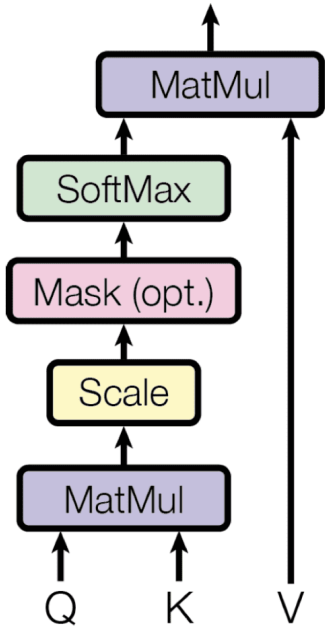
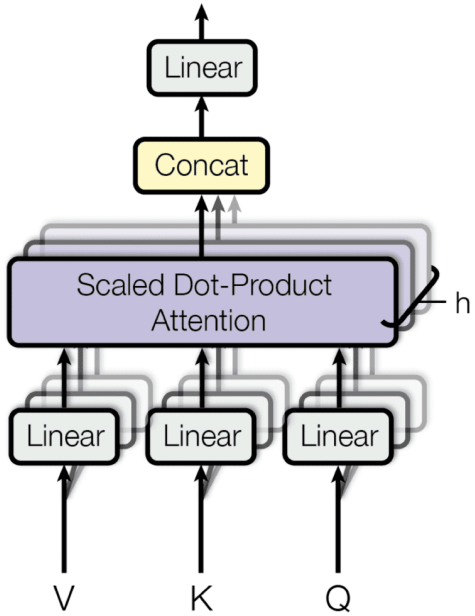


Image Source:
[**Vaswani**, *NeurIPS* 2017]

Modified from source: https://d2l.ai/ (in class we use 'P' rather than 'n', and 'F' rather than 'v')

Attention Mechanism: Multi-Head Attention

- Desirable to capture different behaviors
  - e.g., shorter range versus longer range
- Allow for different transforms of **Q**, **K**, and **V**.
  - This is referred to as **multi-head attention**, where h is the number of heads.
- Transforms are performed with linear projections
  - Three matrix multiplications ($\mathbf{W^Q}$, $\mathbf{W^K}$, $\mathbf{W^V}$) per head
- Outputs are concatenated and undergoes a linear projection
  - Another matrix multiplication ($\mathbf{W^Z}$)
- The weights of these projections are learned



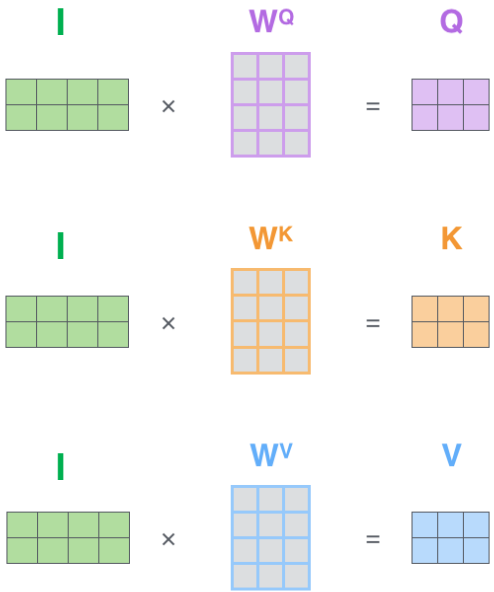
$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1, ...,\text{head}_\text{h})W^Z$$
$$\text{where } \text{head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

[**Vaswani**, *NeurIPS* 2017]

Source: https://d2l.ai/

Attention Mechanism: Generating Q, K, and V

Example of compute per head, where **I** is an "embedding" of an input token.
This is referred to as "**self-attention**" since **Q**, **K**, and **V** are derived from **I**

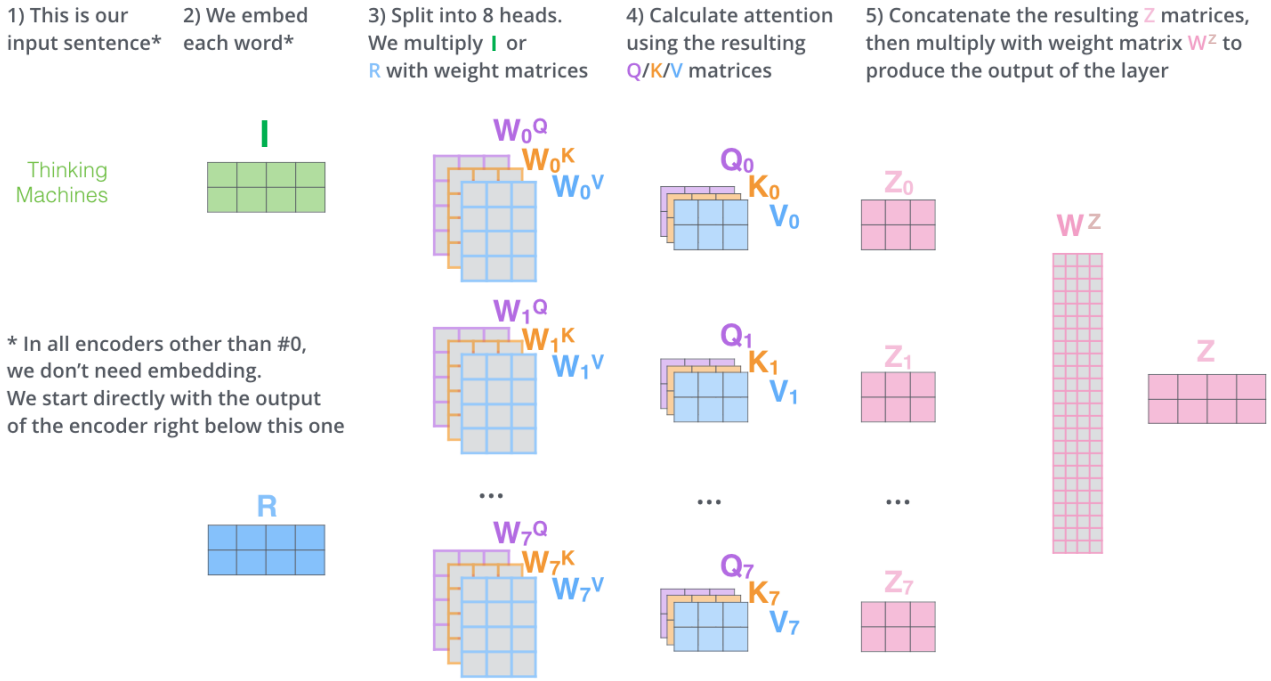


$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{E_k}}\right) V = Z$$

Image Source (modified): https://jalammar.github.io/illustrated-transformer/

Attention Mechanism: Generating Q, K, and V

Example of compute for multiple heads (Note: concatenate results Z_i before multiplying with W^Z)



The output **Z** is processed by a FC layer to generate input **R** to next attention layer.

I is only the input to first layer; **R** is input to subsequent attention layers.

Image Source (modified): https://jalammar.github.io/illustrated-transformer/

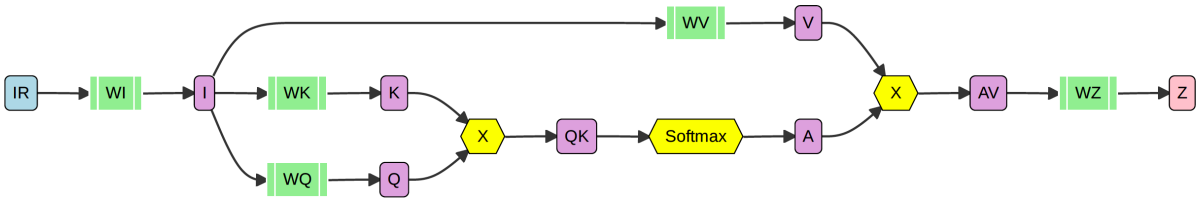
Computation Properties of Attention Mechanism

- Many matrix multiplications
  - Three matrix multiplications for input projections per head for multi-head attention
  - One matrix multiplication for output projections for multi-head attention
  - Two matrix multiplications for computing scaled-dot product attention
  - A total of five matrix multiplications per head plus one for output projection
- Parallel processing across matrix multiplications
  - Across projections of Q, K, and V
  - Across heads
- Sequential dependency between matrix multiplications
  - Within attention: QK^T then multiply by V
  - Between projection and attention: Input projections → Attention → Output projection

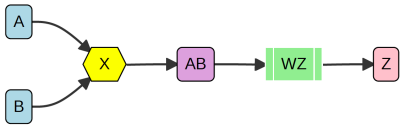
Computation Properties of Attention Mechanism

- Do operands change?
  - Matrices Q, K, and V **change** with input (dynamic)
  - Matrices W^Q, W^K, W^V, and W^Z **does not change** with input (static)
- Reuse W^Q, W^K, W^V, and W^Z across input tokens
- Complexity based on size of matrices and number of tokens
  - Number of MACs scales *quadratically* with number of input tokens
  - Storage: Number of weights in W^Q, W^K, W^V and W^Z (multiplied by number of heads), Intermediate matrices (Q, K, V), Input token matrix I, Output token matrix Z
- Matrix multiplications can be different sizes (design choices: P, D, F, M, H, …)
  - Flexible hardware required!

Summary of Steps in Attention Mechanism



Legend:



- Rounded box –> Tensor
  - Examples: A, B, AB, Z
- Hexagonal box –> Operation
  - Examples: $\times$, $softmax$
- Vertical lined box –> Projection
  - Examples: $\times WZ$

For more details, please refer to
http://csg.csail.mit.edu/6.5930/Lectures/attention.pdf

Einsum Notation

$$I_{m,d} = IR_{m,c} \times WI_{c,d}$$

$$K_{m,e} = I_{m,d} \times WK_{d,e}$$

$$Q_{m,e} = I_{m,d} \times WQ_{d,e}$$

$$QK_{m,p}^{M,P=M} = Q_{p,e}^{M,E} \times K_{m,e}$$

$$SN_{m,p} = exp(QK_{m,p})$$

$$SD_p = SN_{m,p}$$

$$A_{m,p} = SN_{m,p}/SD_p$$

$$V_{m,f} = I_{m,d} \times WV_{d,f}$$

$$AV_{p,f}^{P=M,F} = A_{m,p} \times V_{m,f}$$

$$Z_{p,g} = AV_{p,f} \times WZ_{f,g}$$

Design Choices

- M – number of key-value pairs in the database (sequence length for the query, key and value in self-attention)
- P – number of queries (tokens) processed at a time
- E – vector length (local space embedding) of the queries and keys
- F – vector length (local space embedding) of the values
- H – number of heads in multi-head attention

- C - dictionary size (words in vocabulary)
- D - input global space embedding

- G - output embedding

Examples of Large Language Models (LLMs)

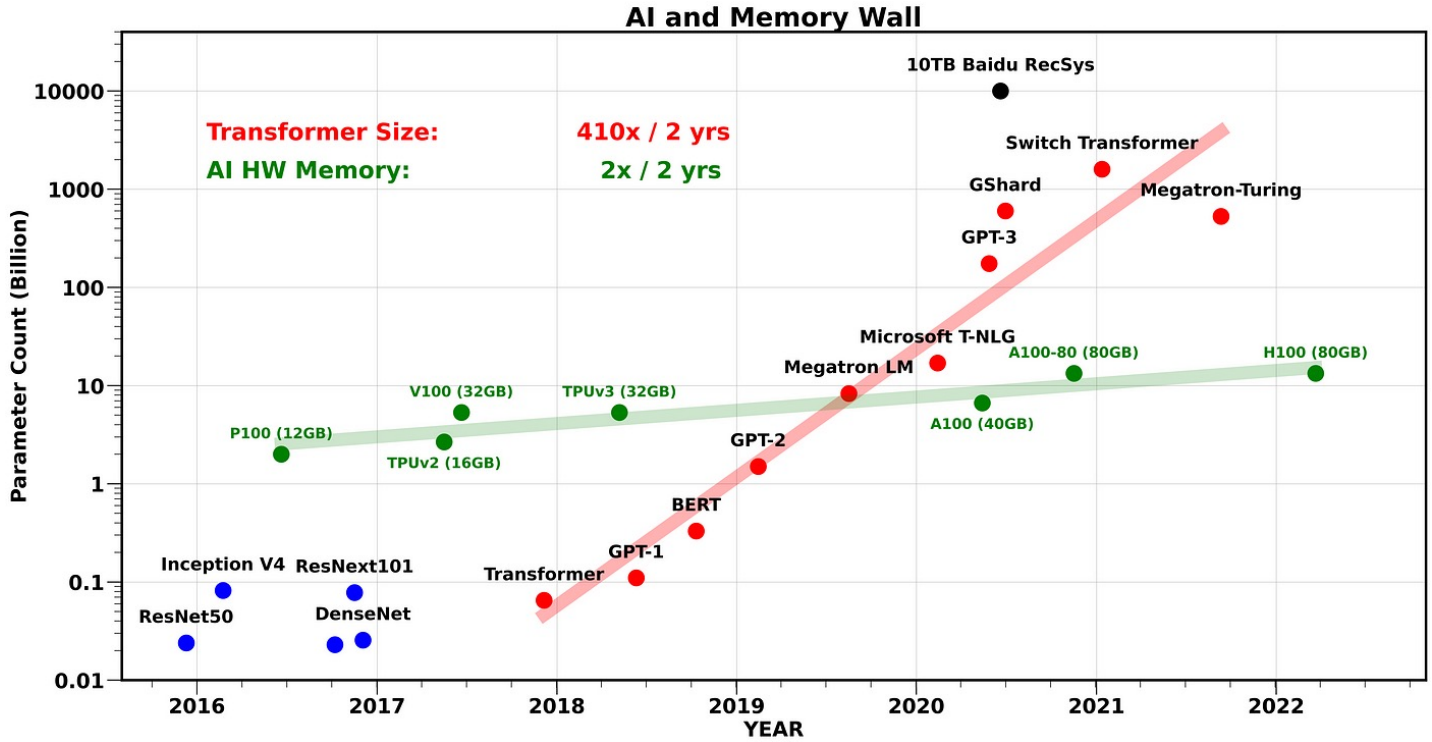
	BERT	BERT	LLAMA	GPT-3
Number of weights (parameters)	**110M**	**340M**	**6.7B**	**175B**
Total MACs per inference (batch size=1)	**53G**	**131G**	**12.8T**	**397T**
Number of layers	12	24	32	96
M	512	512	2048	2048
P	512	512	2048	2048
C	30522	30522	32000	50257
D	768	768	4096	12288
E	32	64	128	128
F	32	64	128	128
G	768	768	4096	12288
H	12	16	32	96
Feed Forward dimension	3072	3072	11008	49152

Storage Requirements for LLMs

- Static (Weights)
  - Linear projection (increase linearly with number of heads)
    - W^Q : DxExH, W^K : DxExH, W^V : DxFxH, W^Z : FxGxH
  - Feed forward layers: 2* Feed Forward dimension x G

- Dynamic
  - Input token matrix I: MxD
  - Intermediate matrices
    - Q: PxE, K: MxE, V: MxF
    - QK^T: MxP, $\text{softmax}\left(\frac{\boldsymbol{QK^T}}{\sqrt{\boldsymbol{E}}}\right)\boldsymbol{V}$: PxF
  - Output token matrix Z: PxG

LLMs Require a Large Amount of Memory



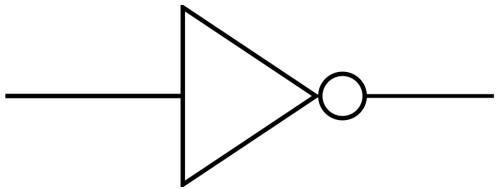
Source: https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8

Key Metrics and Design Objectives

How can we compare designs?

GOPS/W or TOPS/W?

- GOPS = giga (10^9) operations per second
- TOPS = tera (10^{12}) operations per second
- GOPS/Watt or TOPS/Watt commonly reported in hardware literature to show **efficiency** of design
- However, does not provide sufficient insights on hardware capabilities and limitations (especially if based on peak throughput/performance)



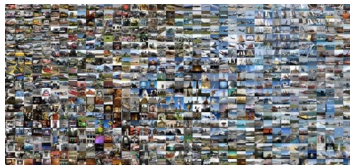
Example: high TOPS per watt can be achieved with inverter (ring oscillator)

Key Metrics: Much more than OPS/W!

- **Accuracy**
  - Quality of result
- **Throughput**
  - Analytics on high volume data
  - Real-time performance (e.g., video at 30 fps)
- **Latency**
  - For interactive applications (e.g., autonomous navigation)
- **Energy and Power**
  - Embedded devices have limited battery capacity
  - Data centers have a power ceiling due to cooling cost
- **Hardware Cost**
  - $$$
- **Flexibility**
  - Range of DNN models and tasks
- **Scalability**
  - Scaling of performance with amount of resources

MNIST

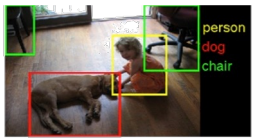
ImageNet



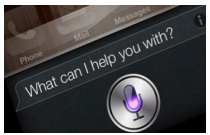
Embedded Device

Data Center

Computer Vision



Speech Recognition



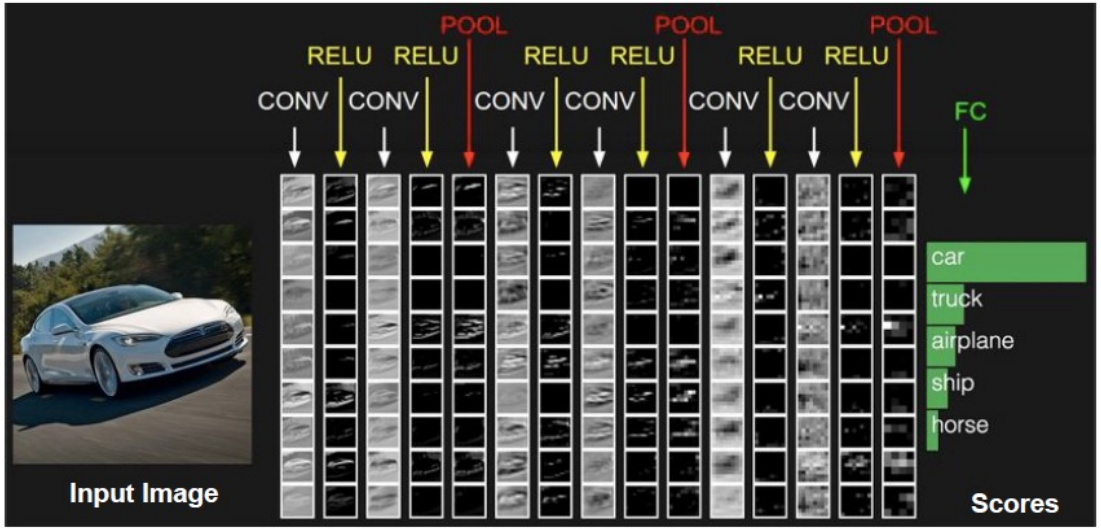
[**Sze**, *CICC* 2017]

Evaluating Accuracy

- Important to measure accuracy when considering co-design of algorithm and hardware
- Datasets help provide a way to evaluate and compare different DNN models and training algorithms
- All accuracy is not the same
  - Must consider difficulty of task and dataset to get fair comparison

Image Classification Datasets

- Image Classification/Recognition
  - Given an entire image → Select 1 of N classes
  - No localization (detection)



Datasets affect difficulty of task

Image Source:
Stanford cs231n

MNIST

Digit Classification
28x28 pixels (B&W)
10 Classes
60,000 Training
10,000 Testing

LeNet in 1998
(0.95% error)

ICML 2013
(0.21% error)

http://yann.lecun.com/exdb/mnist/

CIFAR-10/CIFAR-100

Image Classification
32x32 pixels (color)
10 or 100 Classes
50,000 Training
10,000 Testing

CIFAR-10
Two-layer network in 2009
(35.16% error)

arXiv 2015
(3.47% error)

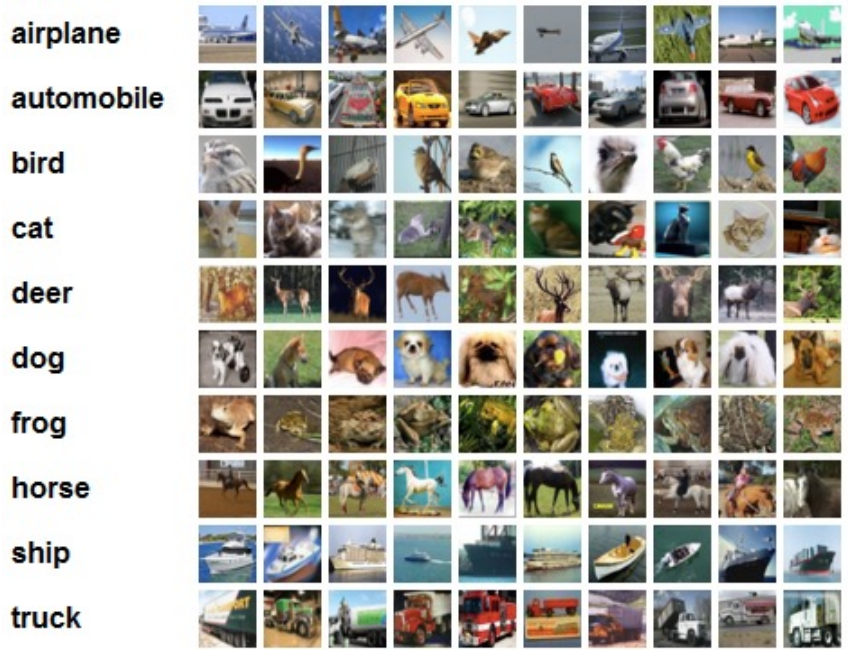


Image Source: http://karpathy.github.io/

Subset of 80M Tiny Images Dataset (Torralba)

https://www.cs.toronto.edu/~kriz/cifar.html

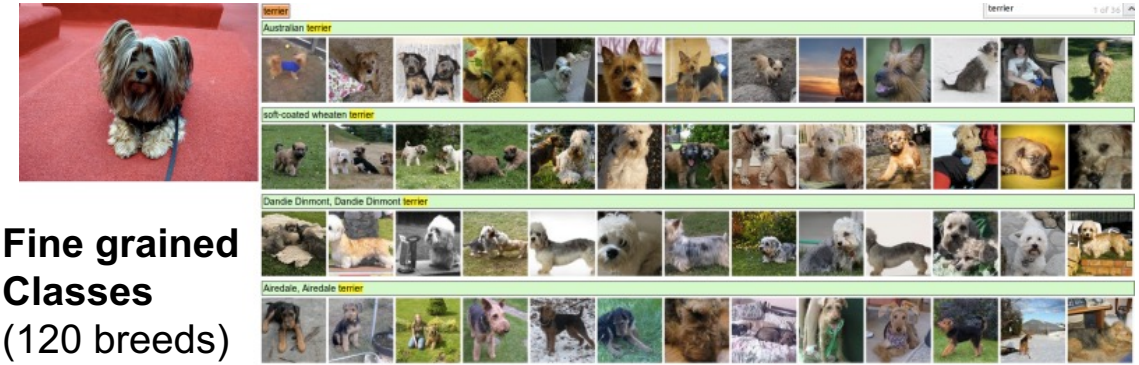
IMAGENET

Image Classification
~256x256 pixels (color)
1000 Classes
1.3M Training
100,000 Testing (50,000 Validation)

Image Source: http://karpathy.github.io/

http://www.image-net.org/challenges/LSVRC/

IMAGENET



Fine grained Classes
(120 breeds)

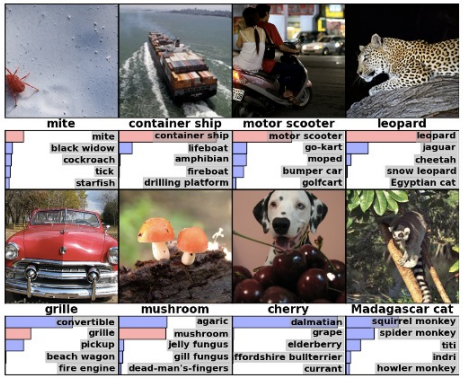
Image Source: http://karpathy.github.io/

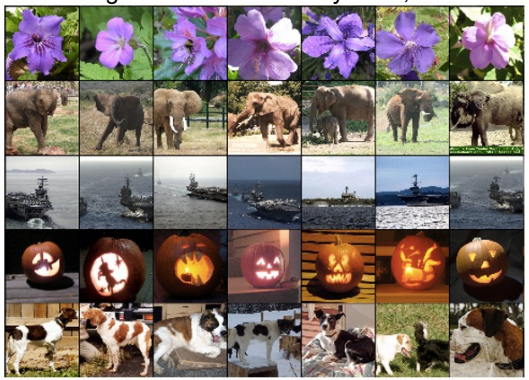
Image Source: Krizhevsky et al., NIPS 2012

Top-5 Error

Winner 2012
(16.42% error)

Winner 2017
(2.25% error)





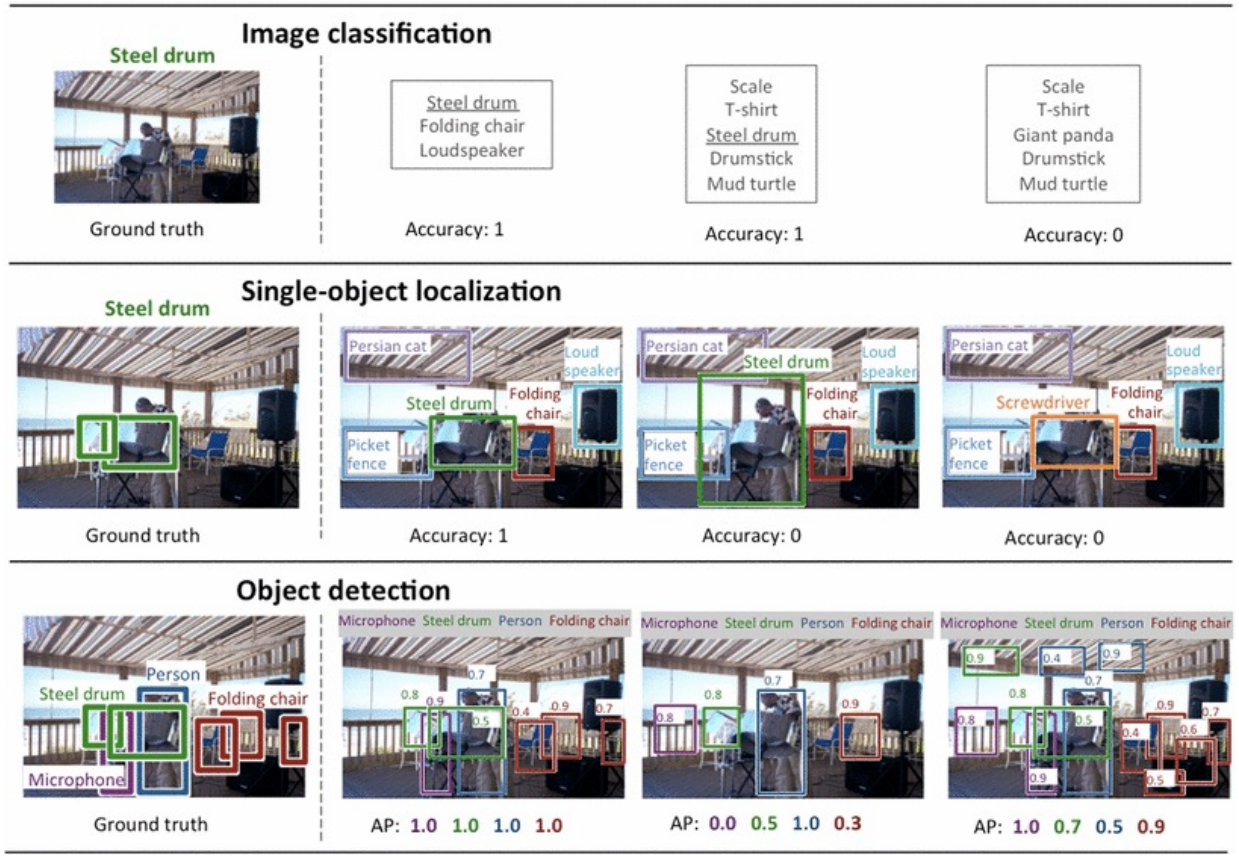
http://www.image-net.org/challenges/LSVRC/

Image Classification Summary

	MNIST	CIFAR-10	IMAGENET
Year	1998	2009	2012
Resolution	28x28	32x32	256x256
Classes	10	10	1000
Training	60k	50k	1.3M
Testing	10k	10k	100k
Accuracy	0.21% error (ICML 2013)	3.47% error (arXiv 2015)	2.25% top-5 error (2017 winner)

http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

Next Tasks: Localization and Detection



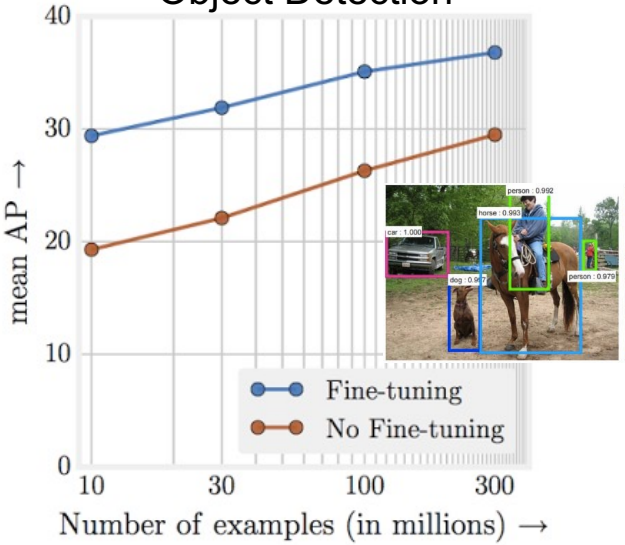
[**Russakovsky**, *IJCV* 2015]

Effectiveness of More Data

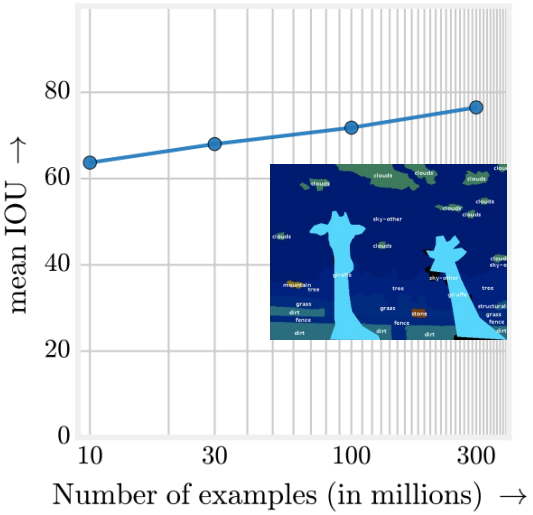
Accuracy increases logarithmically based on amount training data

Results from Google Internal Dataset
JFT-300M (300M images, 18291 categories)
Orders of magnitude larger than ImageNet

Object Detection



Semantic Segmentation



"Disclaimer – Large scale learning: We would like to highlight that the training regime, learning schedules and parameters used in this paper are based on our understanding of training ConvNets with 1M images. Searching the right set of hyper-parameters requires significant more effort: **even training a JFT model for 4 epochs needed 2 months on 50 K-80 GPUs.** Therefore, in some sense the quantitative performance reported in this paper underestimates the impact of data for all reported image volumes."

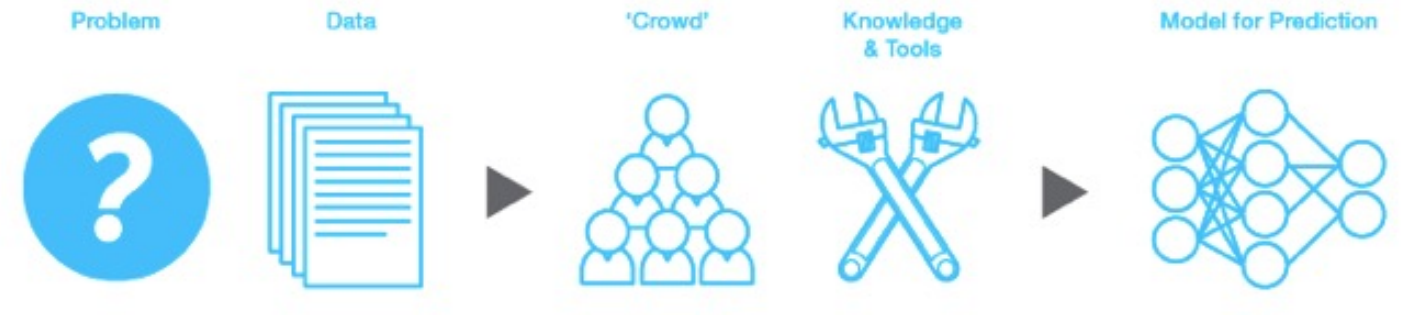
[**Sun**, *ICCV* 2017]

Recently Introduced Datasets

- Google Open Images (~9M images)
  - https://github.com/openimages/dataset
- Youtube-8M (8M videos)
  - https://research.google.com/youtube8m/
- AudioSet (2M sound clips)
  - https://research.google.com/audioset/index.html

Kaggle

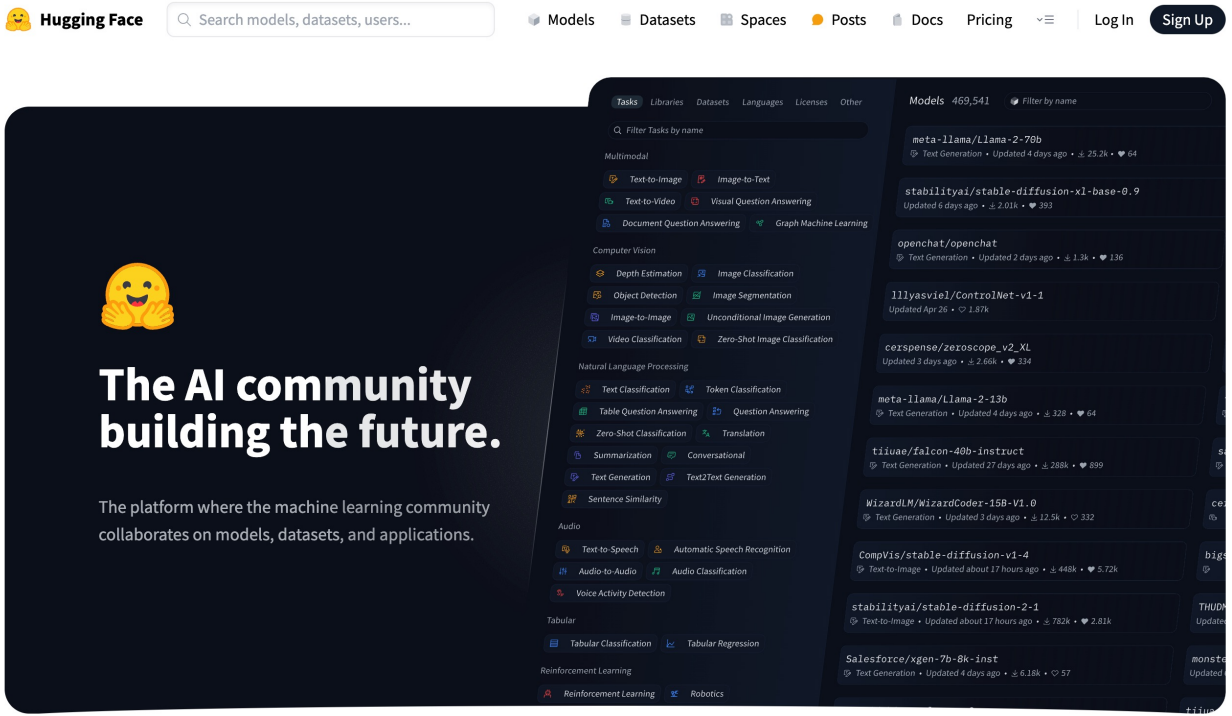
A platform for predictive modeling competitions



Over 3,500 competition submissions per day
Over 2000+ datasets!

Starting 2018, ImageNet Challenge hosted by Kaggle
https://www.kaggle.com/c/imagenet-object-localization-challenge

Hugging Face



https://huggingface.co/

Key Design Objectives of DNN Processor

- **Increase Throughput and Reduce Latency**
  - Reduce time per MAC
    - Reduce critical path → increase clock frequency
    - Reduce instruction overhead
  - Avoid unnecessary MACs (save cycles)
  - Increase number of processing elements (PE) → more MACs in parallel
    - Increase area density of PE or area cost of system
  - Increase PE utilization* → keep PEs busy
    - Distribute workload to as many PEs as possible
    - Balance the workload across PEs
    - Sufficient memory bandwidth to deliver workload to PEs (reduce idle cycles)
- Low latency has an additional constraint of **small batch size**

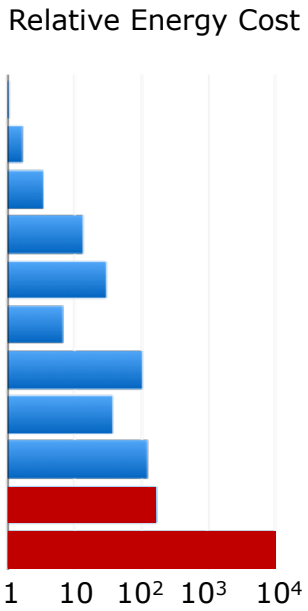
*(100% = peak performance)

Key Design Objectives of DNN Processor

- **Reduce Energy and Power Consumption**
  - Reduce data movement as it dominates energy consumption
    - Exploit data reuse
  - Reduce energy per MAC
    - Reduce switching activity and/or capacitance
    - Reduce instruction overhead
  - Avoid unnecessary MACs
- Power consumption is limited by heat dissipation, which limits the maximum # of MACs in parallel (i.e., throughput)

Operation:	Energy (pJ)
8b Add	0.03
16b Add	0.05
32b Add	0.1
16b FP Add	0.4
32b FP Add	0.9
8b Multiply	0.2
32b Multiply	3.1
16b FP Multiply	1.1
32b FP Multiply	3.7
32b SRAM Read (8KB)	5
32b DRAM Read	640

[**Horowitz**, *ISSCC* 2014]



Key Design Objectives of DNN Processor

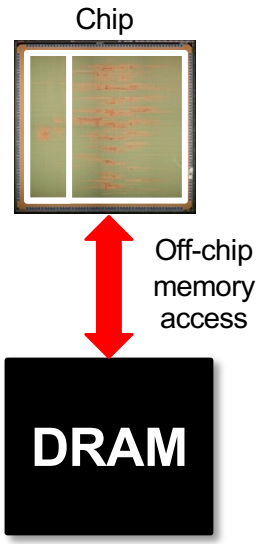
- **Flexibility**
  - Reduce overhead of supporting flexibility
  - Maintain efficiency across wide range of DNN models
    - Different layer shapes impact the amount of
      - Required storage and compute
      - Available data reuse that can be exploited
    - Different precision across layers & data types (weight, activation, partial sum)
    - Different degrees of sparsity (number of zeros in weights or activations)
    - Types of DNN layers and computation beyond MACs (e.g., activation functions)

Key Design Objectives of DNN Processor

- **Scalability**
  - Increase how performance (i.e., throughput, latency, energy, power) scales with increase in amount of resources (e.g., number of PEs, amount of memory, etc.)

Specifications to Evaluate Metrics

- **Accuracy**
  - Difficulty of dataset and/or task should be considered
  - Difficult tasks typically require more complex DNN models
- **Throughput**
  - Number of PEs with utilization (not just peak performance)
  - Runtime for running specific DNN models
- **Latency**
  - Batch size used in evaluation
- **Energy and Power**
  - Power consumption for running specific DNN models
  - Off-chip memory access (e.g., DRAM)
- **Hardware Cost**
  - On-chip storage, # of PEs, chip area + process technology
- **Flexibility**
  - Report performance across a wide range of DNN models
  - Define range of DNN models that are efficiently supported



Evaluation Process

The evaluation process for whether a DNN system is a viable solution for a given application might go as follows:

1. **Accuracy** determines if it can perform the given task
2. **Latency and throughput** determine if it can run fast enough and in real-time
3. **Energy and power consumption** will primarily dictate the form factor of the device where the processing can operate
4. **Cost**, which is primarily dictated by the chip area and external interfaces, determines how much one would pay for this solution
5. **Flexibility** determines the range of tasks it can support

Example: Metrics of Eyeriss Chip

ASIC Specs	Input
Process Technology	65nm LP TSMC (1.0V)
Total Core Area (mm^2)	12.25
Total On-Chip Memory (kB)	192
Number of Multipliers	168
Clock Frequency (MHz)	200
Core area (mm^2) /multiplier	0.073
On-Chip memory (kB) / multiplier	1.14
Measured or Simulated	Measured

Metric	Units	Input
Name of CNN Model	Text	AlexNet
Top-5 error classification on ImageNet	#	19.8
Supported Layers		All CONV
Bits per weight	#	16
Bits per input activation	#	16
Batch Size	#	4
Runtime	ms	115.3
Power	mW	278
Off-chip Access per Image Inference	MBytes	3.85
Number of Images Tested	#	100

Comprehensive Coverage for Evaluation

- **All metrics** should be reported for fair evaluation of design tradeoffs
- Examples of what can happen if certain metric is omitted:
  - **Without the accuracy given for a specific dataset and task**, one could run a simple DNN and claim low power, high throughput, and low cost – however, the processor might not be usable for a meaningful task
  - **Without reporting the off-chip bandwidth**, one could build a processor with only multipliers and claim low cost, high throughput, high accuracy, and low chip power – however, when evaluating system power, the off-chip memory access would be substantial
- Are results measured or simulated? On what test data?
- Hardware should be evaluated on a wide range of DNNs
  - No guarantee that DNN algorithm designer will use a given DNN model or given reduce complexity approach. **Need flexible hardware!**

MLPerf: Workloads for Benchmarking

23 Companies
7 Institutions

First results in Dec 2018

https://mlperf.org/

A broad ML benchmark suite for measuring performance of ML software frameworks, ML hardware accelerators, and ML cloud platforms.

- A broad suite of DNN models to serve as a common set of benchmarks to measure the performance and enable fair comparison of various software frameworks, hardware accelerators, and cloud platforms for both training and inference of DNNs. **(edge compute in the works!)**
- The suite includes a wide range of DNNs (e.g., CNN, RNN, etc.) for a variety of tasks include image classification, object identification, translation, speech-to-text, recommendation, sentiment analysis and reinforcement learning.
- Categories: cloud/edge; training/inference; closed/open

Specifications of DNN Models

- **Accuracy**
  - Define task and dataset
- **Shape of DNN Model** ("Network Architecture")
  - # of layers, filter size (R, S), # of channels (C), # of filters (M)
- **# of Weights & Activations (storage capacity)**
  - Number of non-zero (NZ) weights and activations
- **# of MACs (operations)**
  - Number of non-zero (NZ) MACS

Specifications of DNN Models

Metrics	AlexNet	VGG-16	GoogLeNet (v1)	ResNet-50
Accuracy (top-5 error)*	19.8	8.80	10.7	7.02
# of CONV Layers	**5**	**16**	**21**	**49**
# of Weights	2.3M	14.7M	6.0M	23.5M
# of MACs	666M	15.3G	1.43G	3.86G
# of NZ MACs**	**394M**	**7.3G**	**806M**	**1.5G**
# of FC layers	**3**	**3**	**1**	**1**
# of Weights	58.6M	124M	1M	2M
# of MACs	58.6M	124M	1M	2M
# of NZ MACs**	**14.4M**	**17.7M**	**639k**	**1.8M**
Total Weights	**61M**	**138M**	**7M**	**25.5M**
Total MACs	**724M**	**15.5G**	**1.43G**	**3.9G**
# of NZ MACs**	**409M**	**7.3G**	**806M**	**1.5G**

*Single crop results: https://github.com/jcjohnson/cnn-benchmarks

**# of NZ MACs computed based on 50,000 validation images

Weights & MACs → Energy & Latency

- ***Warning:*** *Fewer weights and MACs (indirect metrics) do not necessarily result in lower energy consumption or latency (direct metrics). Other factors also important such as filter shape, batch size and hardware mapping.*

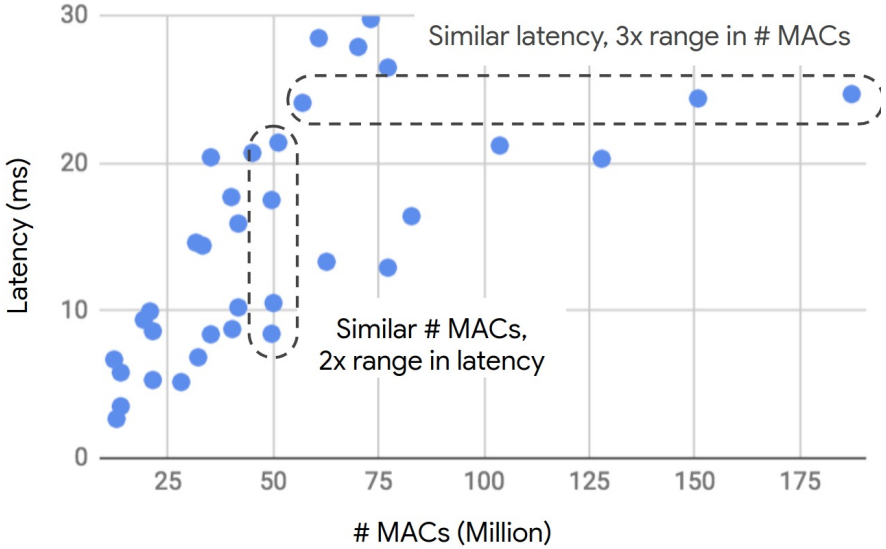
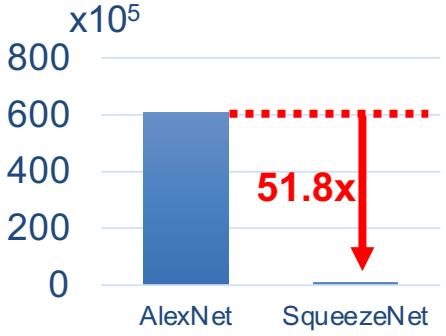


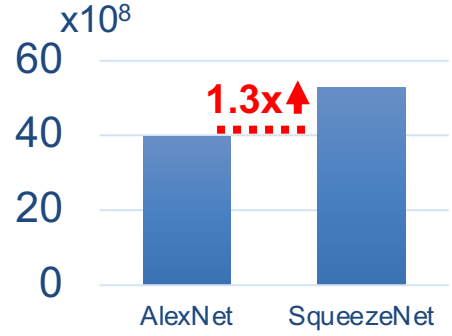
Image Source:
Google AI Blog

[**Yang**, *CVPR* 2017], [**Chen**, *SysML*, 2018], [**Yang**, *ECCV* 2018]

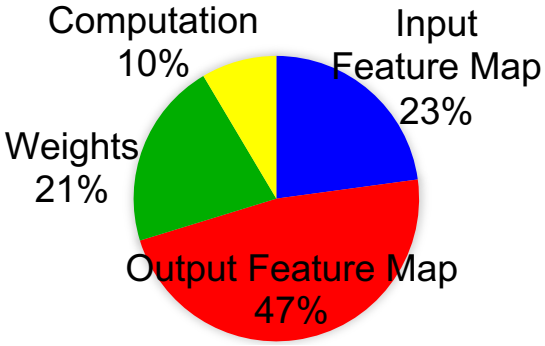
Example: AlexNet vs. SqueezeNet



# of Weights



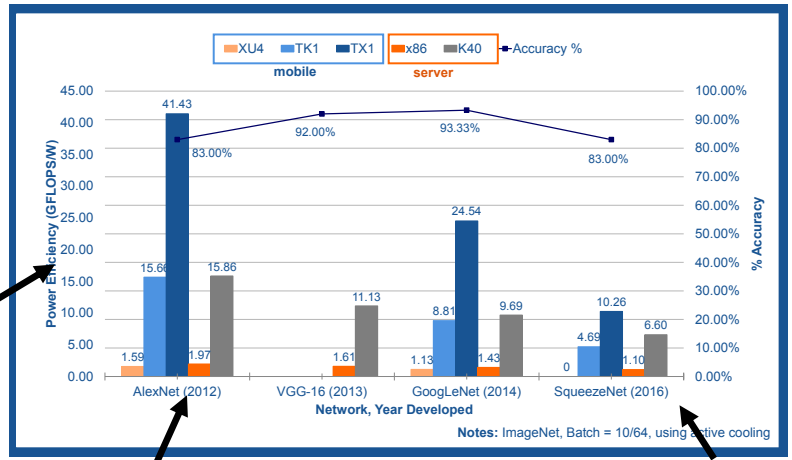
Normalized Energy



Energy Breakdown (SqueezeNet)

Results for SqueezeNetv1.0
Batch size=48

Power Efficiency (GFLOPS/W)

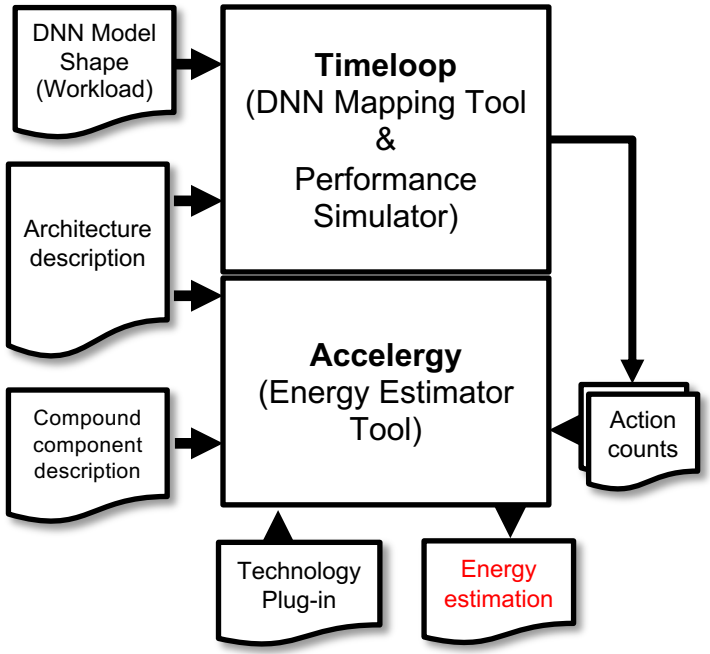


AlexNet **SqueezeNet**

[**Movidius**, *Hot Chips* 2016]

DNN Processor Evaluation Tools

- **Require systematic way to**
  - Evaluate and compare wide range of DNN processor designs
  - Rapidly explore design space
- **Accelergy [Wu, *ICCAD* 2019]**
  - Early-stage energy estimation tool at the architecture level
    - Estimate energy consumption based on architecture level components (e.g., # of PEs, memory size, on-chip network)
  - Evaluate architecture level energy impact of emerging devices
    - Plug-ins for different technologies
- **Timeloop [Parashar, *ISPASS* 2019]**
  - DNN mapping tool
  - Performance Simulator → Action counts



Labs and final project

Open-source code available at:
http://accelergy.mit.edu

Summary

- Evaluate hardware using the appropriate DNN model and dataset
  - Difficult tasks typically require larger models
  - Different datasets for different tasks
  - Number of datasets growing at a rapid pace
- A comprehensive set of metrics should be considered when comparing DNN hardware to fully understand design tradeoffs

Training

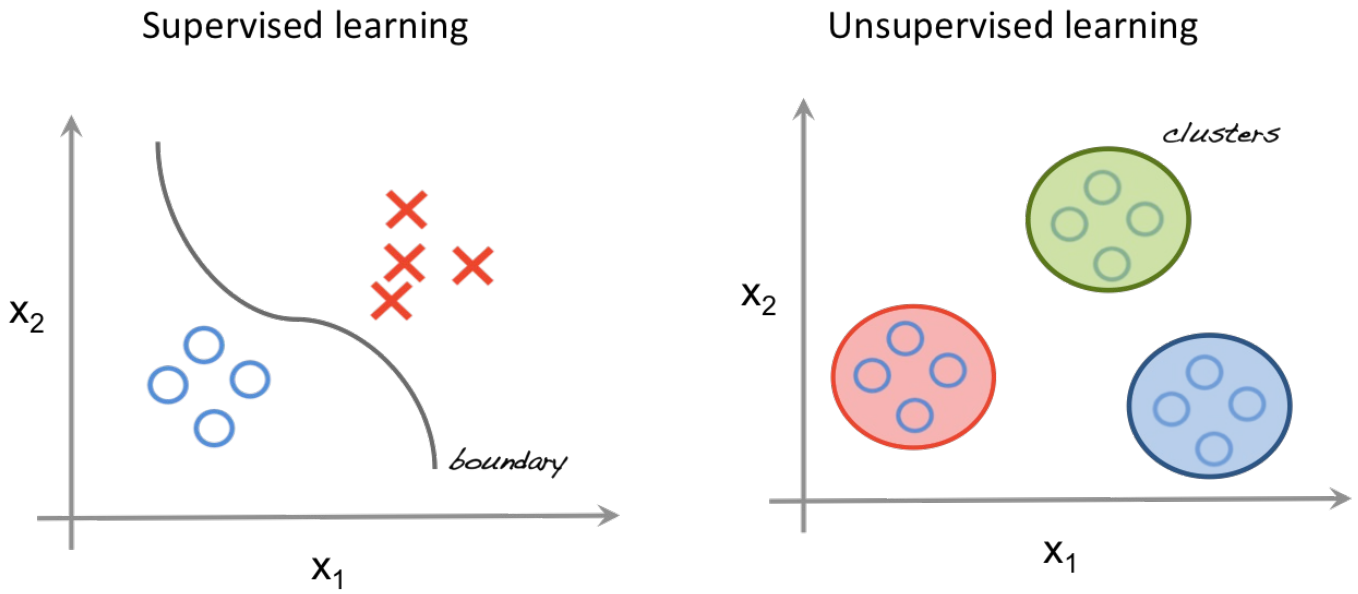
Training vs. Inference

- **Training:** Determine weights
  - **Supervised**
    - Training set has inputs and outputs, i.e., labeled
  - **Unsupervised (Self-Supervised)**
    - Training set is unlabeled
  - **Semi-supervised**
    - Training set is partially labeled
  - **Reinforcement**
    - Output assessed via rewards and punishments

- **Inference:** Apply weights to determine output

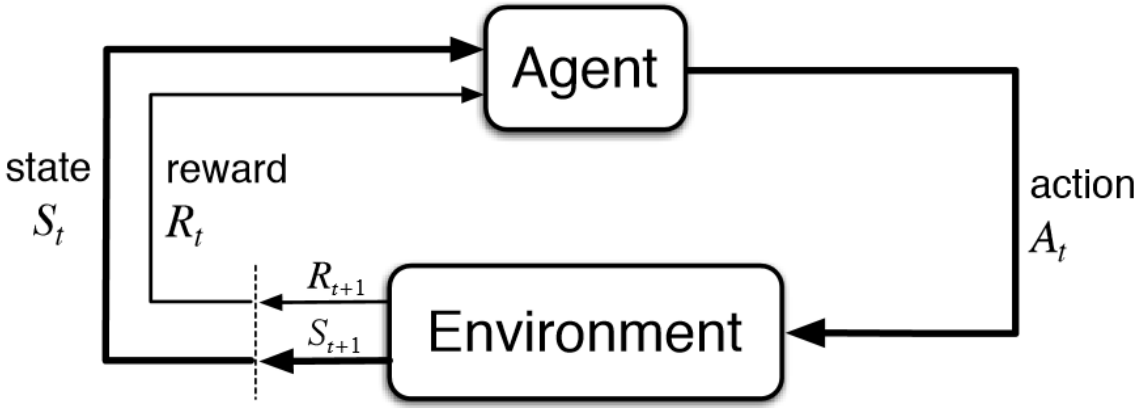
Unsupervised Learning

Finds structure in **unlabeled** data



[image source: cambridgespark.com]

Reinforcement Learning

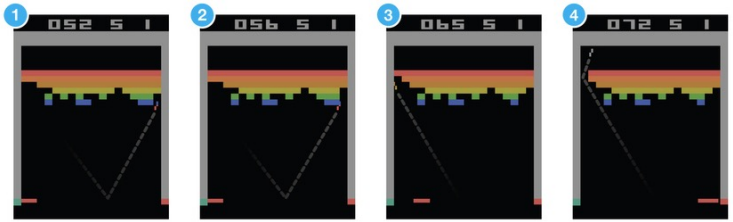


Given the state of the current **environment**, **learn** **a policy** that decides what action the **agent** should take next to maximize expected rewards.

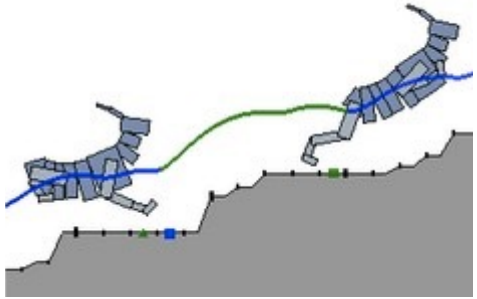
However, the rewards might not be available immediately after an action, but instead only after a series of actions.

Reinforcement Learning Examples

Game Play



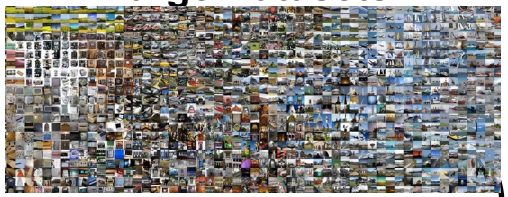
Robotics



Training versus Inference

Training
(determine weights)

Large Datasets



Weights

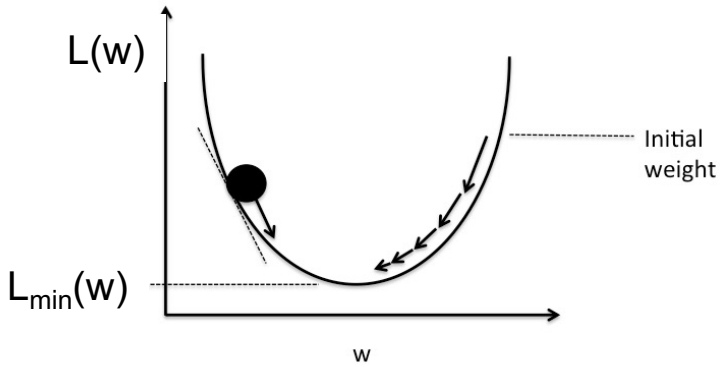
Inference
(use weights)

Gradient Descent

- **Goal:** Determine set of weights to minimize loss
- Use **gradient descent** to incrementally update weights to reduce loss
  - Compute derivative of loss relative to weights to indicate how to change weights (linear approximation of loss function)

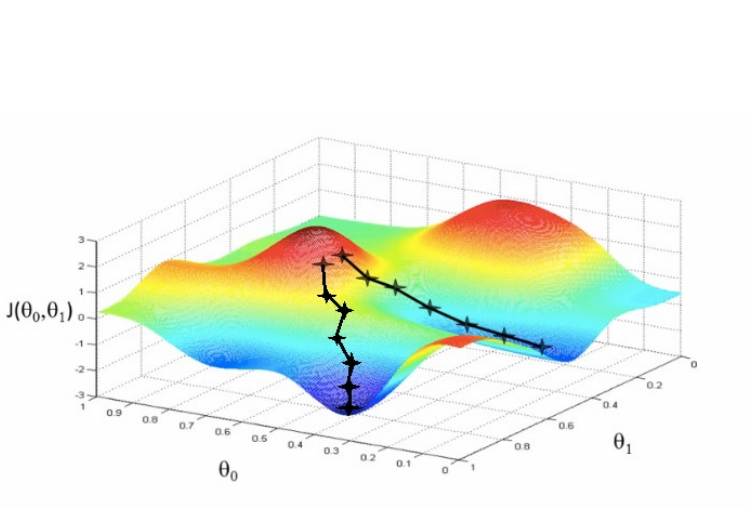
$$w_{ij}^{t+1} = w_{ij}^{t} - \alpha \frac{\partial L}{\partial w_{ij}}$$

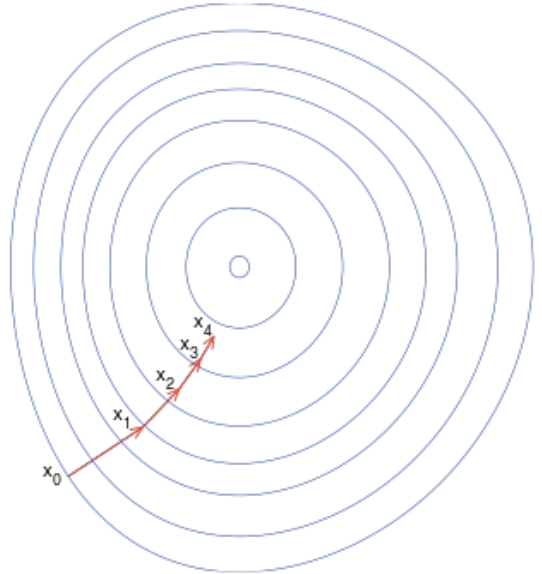
Learning rate



[Image Source: http://sebastianraschka.com/]

Visualization of Gradient Descent





[Image Source: Wikipedia]

Computing Gradients for DNN

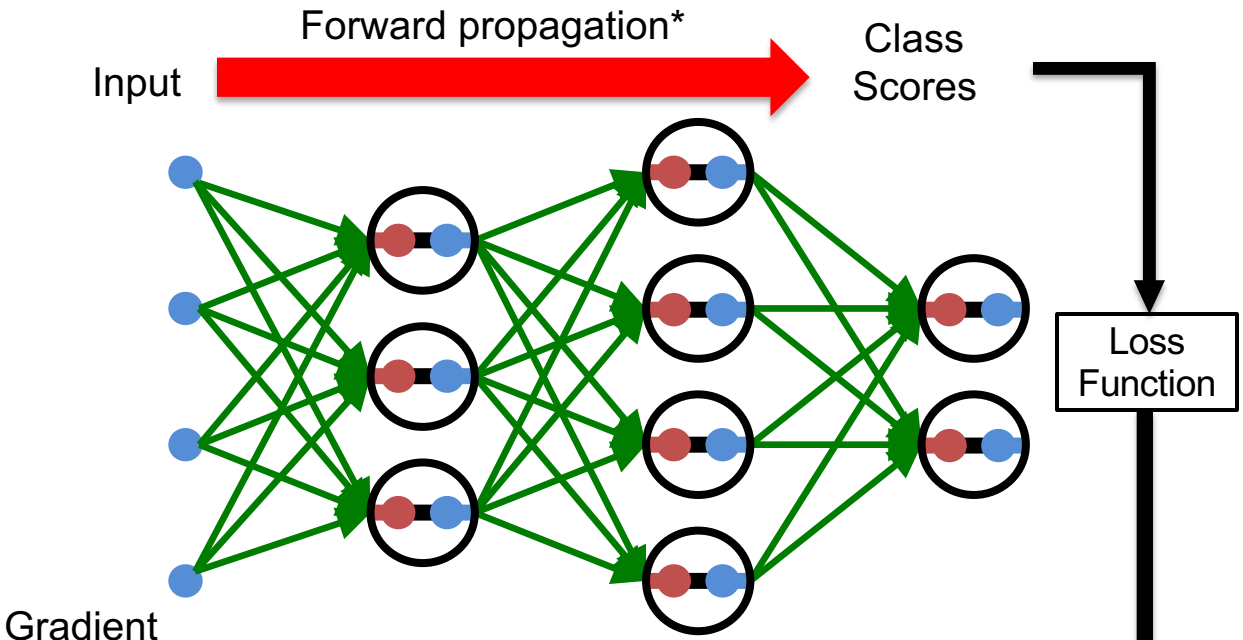
Recall method to **update weights** during **training**:

$$w_{ij}^{t+1} = w_{ij}^{t} - \alpha \frac{\partial L}{\partial w_{ij}}$$

gradient

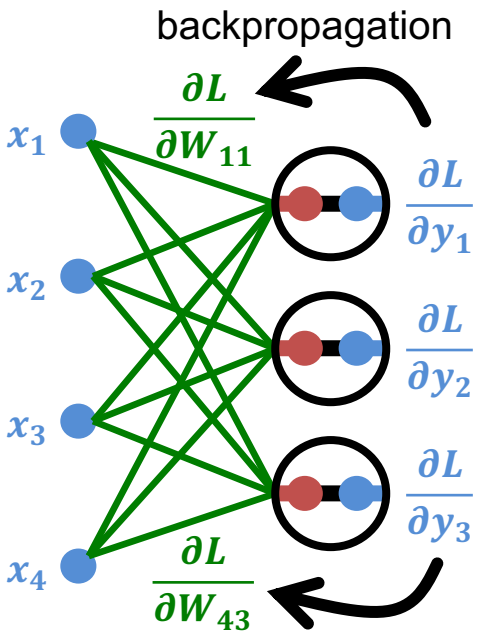
An efficient way to compute gradient (a.k.a. partial derivative) for DNN is using a process called **back propagation.**

Training DNN



Back-Propagation of Weights (per Layer)

Determine how **loss** changes w.r.t. to **weights**



$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial w_{ij}}$$ chain rule

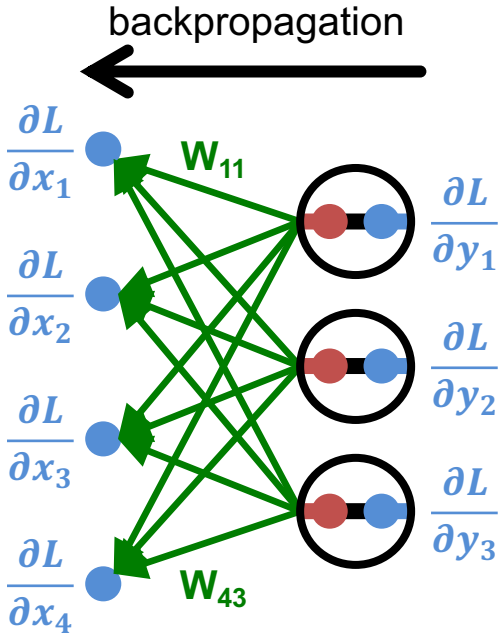
$$y = Wx + b \Rightarrow \frac{\partial y_j}{\partial w_{ij}} = x_i$$

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial y_j} x_i$$

Need to compute $\frac{\partial L}{\partial y_j}$

Back-Propagation of Activations (per Layer)

Determine how **loss** changes w.r.t. to **input activations**



Layer 1 Layer 2

$$\frac{\partial L}{\partial y_j} = \frac{\partial L}{\partial x_j}$$

Layer 2

$$\frac{\partial L}{\partial x_i} = \sum_j w_{ij} \frac{\partial L}{\partial y_j}$$

Similar in form to the computation used for inference

Back Propagation Across All Layers

Gradient w.r.t. **weights**

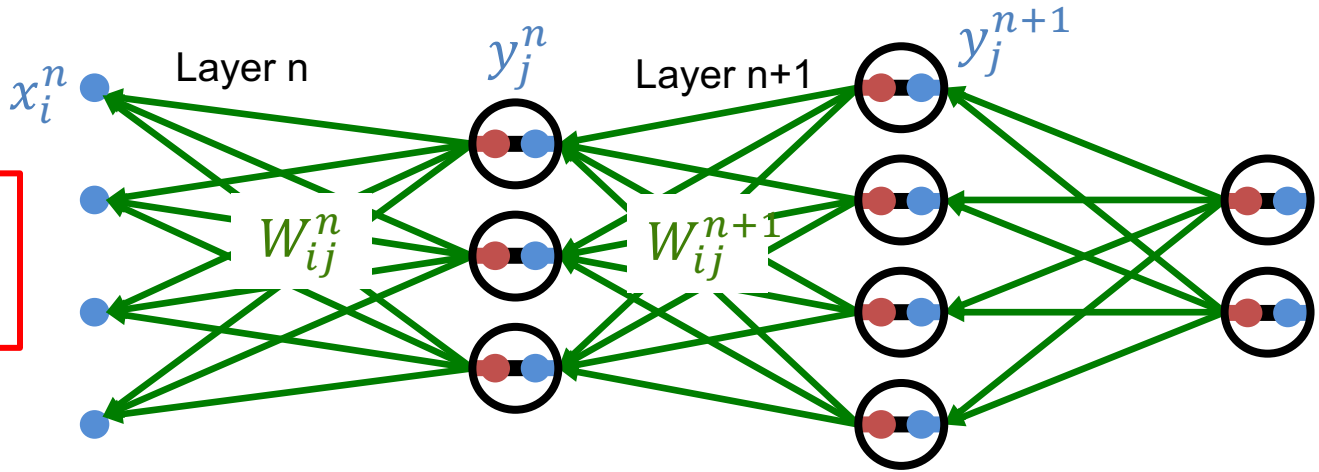
$$\frac{\partial L}{\partial w_{ij}^n} = \frac{\partial L}{\partial y_j^n}\frac{\partial y_j^n}{\partial w_{ij}^n} = \frac{\partial L}{\partial y_j^n} x_i^n$$

Gradient w.r.t. **activations**

$$\frac{\partial L}{\partial y_i^n} = \sum_j w_{ij}^n \frac{\partial L}{\partial y_j^{n+1}}$$

where $y_j^n = \sum_i w_{ij}^n x_i^n + b$

Note: $y_i^n = x_i^{n+1}$



Need to **store activations** from forward propagation!

Demo of CIFAR-10 CNN Training

ConvNetJS CIFAR-10 demo

Description

This demo trains a Convolutional Neural Network on the CIFAR-10 dataset in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used this python script to parse the original files (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and verically.

By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to @karpathy.

http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

References

- Chapter 2 & 3 in Book
  - https://doi.org/10.1007/978-3-031-01766-7
- Other Works Cited in Lecture
  - Russakovsky, Olga, et al. "Imagenet large scale visual recognition challenge." International Journal of Computer Vision 115.3 (2015): 211-252.
  - Sun, Chen, et al. "Revisiting unreasonable effectiveness of data in deep learning era." arXiv preprint arXiv:1707.02968 (2017).
  - Shrivastava, Ashish, et al. "Learning from simulated and unsupervised images through adversarial training." arXiv preprint arXiv:1612.07828 (2016).
  - T.-J. Yang et al., “NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications,” ECCV 2018.
  - Y.-H. Chen*, T.-J. Yang*, J. Emer, V. Sze, “Understanding the Limitations of Existing Energy-Efficient Design Approaches for Deep Neural Networks,” *SysML Conference* 2018.
  - T.-J. Yang et al., “Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning,” *CVPR* 2017.
  - Chen et al., Eyexam, https://arxiv.org/abs/1807.07928
  - Williams et al., “Roofline: An insightful visual performance model for floating-point programs and multicore architectures,” CACM 2009
  - Wu et al., “Accelergy: An architecture-level energy estimation methodology for accelerator designs,” ICCAD 2019
  - Parashar et al., “Timeloop: A systematic approach to dnn accelerator evaluation,” ISPASS 2019