Edge-AI (Theory)

Luis Piñuel

ArTeCS - UCM

Section 1

Intro

Goals

- Complement the practical part of the course
- Give an overview of the foundational concepts of Edge AI
- Pay special attention to DNN inference processing and acceleration

Outline

- Intro: What is Edge AI?
- Background AI, ML, DL (aka DNN)
- Overview of DNN
- Reduce storage/compute
- DNN Hardware Specialization
- DNN Accelerator Architectures
- Benchmarking
- Edge AI HW case studies:
  - Mobile
  - Embedded devices
  - Autonomous vehicles

Prior Knowledge

- Computer Organization (necessary)
- Computer Architecture (recommended)

Schedule

- **21/04:** theory lecture
- **28/04:** theory lecture
- **05/05:** group presentations
  - 15 min presentation + 5 min Q&A

Section 2

Edge AI

What is Edge AI?

- Running AI algorithms locally on a hardware device.

Why process data locally?

According to Gatner:

> *"As the volume and velocity of data increases, so too does the inefficiency of streaming all this information to a cloud or data center for processing."*

> *"Around 10% of enterprise-generated data is created and processed outside a traditional centralized data center or cloud. By 2025, this figure will reach 75%"*

What are the main advantages of EAI?

- Latency reduction
- Reduced costs
  - communication, bandwidth, power, ...
- Security
- Privacy

What can EAI be used for?

- Surveillance and Monitoring
- Autonomous vehicles / Control Systems
- Smart speakers / Voice Assistants
- Point of Sale
- AI applied to IoT (aka AIoT)

What kind of hardware device are employed?

- Mostly an embedded device
- Very diverse characteristics (performance, power consumption, costs, etc.) depending on the target application
  - High performance/power for autonomous vehicles
  - Medium to low performance/power for the vast majority of applications
- Most of them based on a SoC with some kind AI hardware
  - Accelerator, coprocessor, ISA extensions, ...

What kind of algorithms are employed?

- Mostly Artificial Neural Networks inference
  - CNN
  - RNN
- But not only . . .
  - SVM
  - KNN
  - DT

What are the general steps to design an AI system?

- Identify the problem.
- Prepare the data.
- Choose the algorithms.
- Train the algorithms.
- Choose a programming language.
- Run on a selected platform.

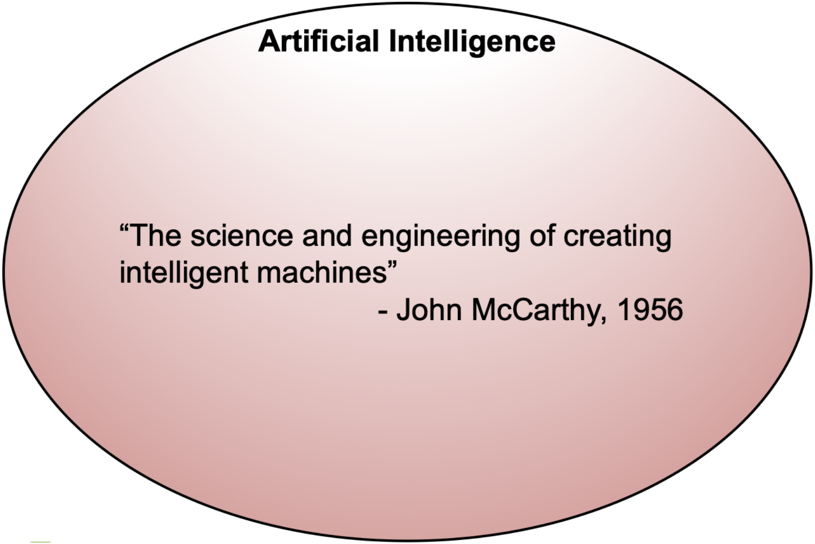
What are specific/critical steps to design an Edge AI system?

- Carefully select the hardware device and the algorithms to meet the application constraints (latency, cost, power, cooling, . . .)
- Map the algorithm to the selected platform
  - Inference framework
  - Libraries
- Fine-tune the system

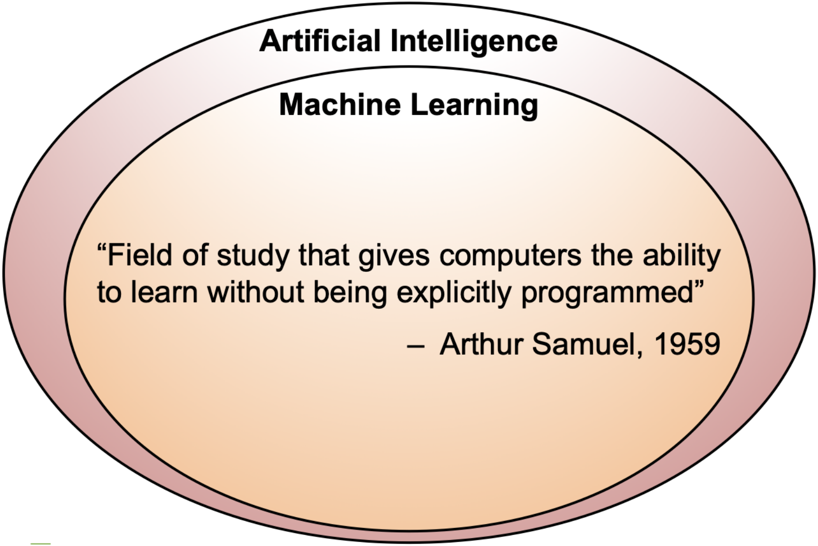
Section 3

Background: AI, ML, DL

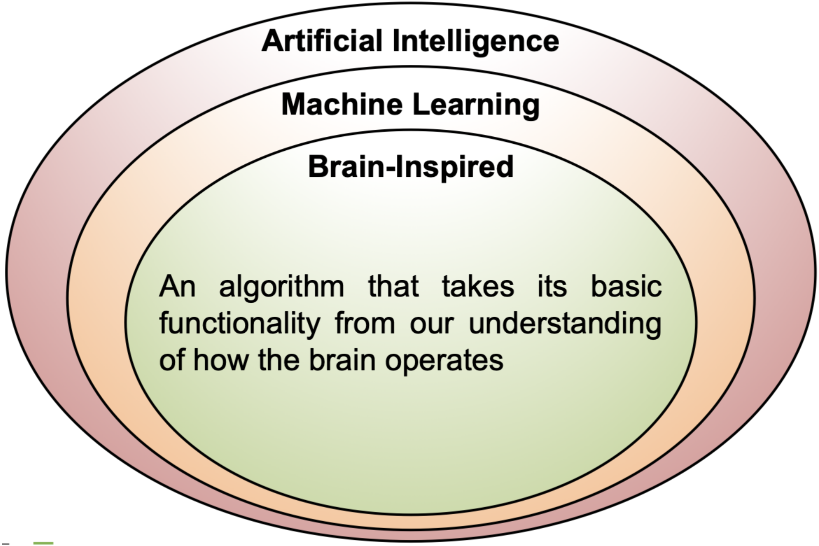
Artificial Intelligence



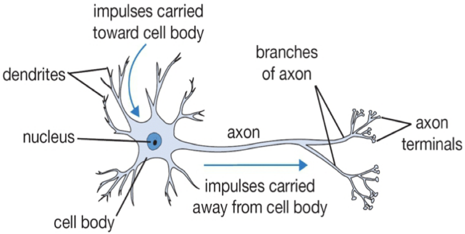
AI and Machine Learning

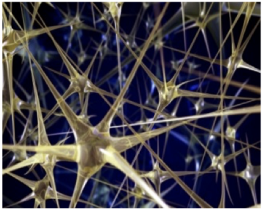


Brain-Inspired Machine Learning



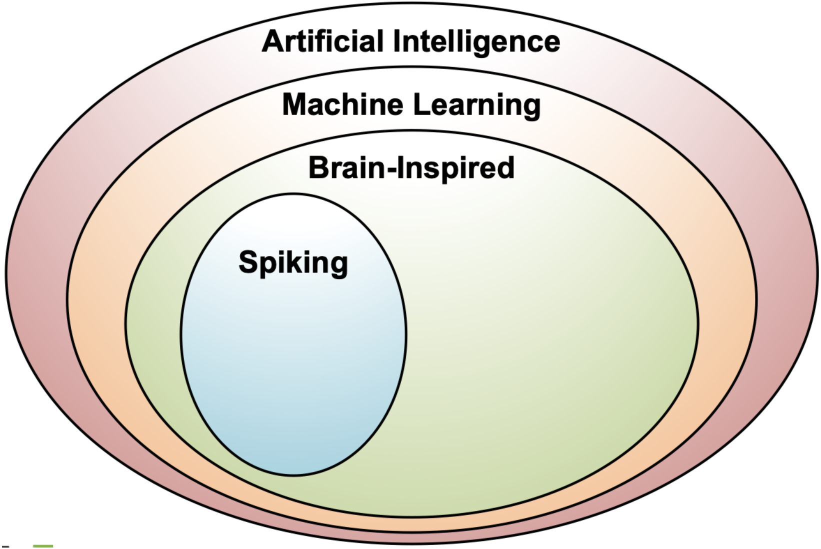
How Does the Brain Work?





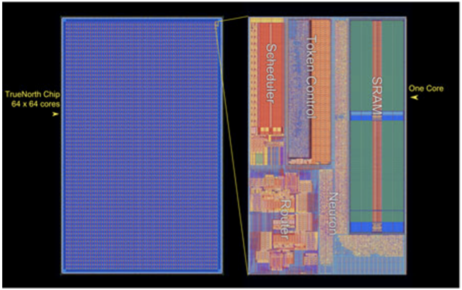
- The basic computational unit of the brain is a **neuron**
  → 86B neurons in the brain
- Neurons are connected with nearly **10^{14} – 10^{15} synapses**
- Neurons receive input signal from **dendrites** and produce output signal along **axon**, which interact with the dendrites of other neurons via **synaptic weights**
- Synaptic weights – learnable & control influence strength

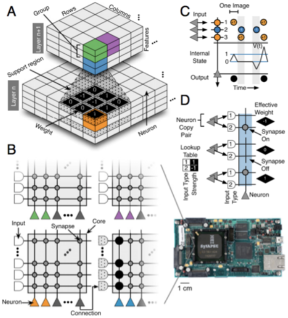
Spiking-based Machine Learning



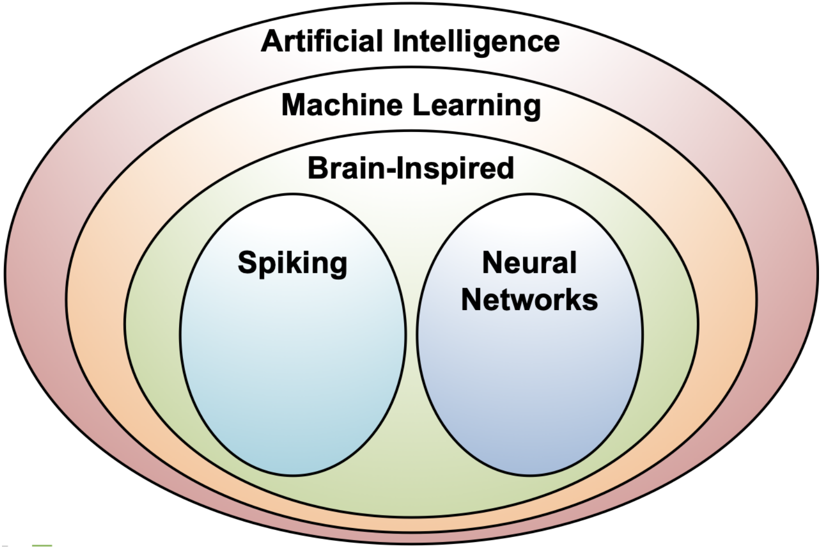
Spiking Architecture

- Brain-inspired
- Integrate and fire
- Example: IBM TrueNorth

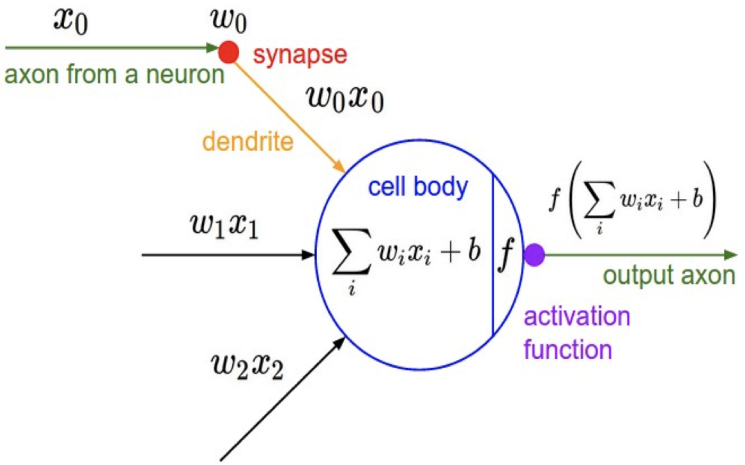




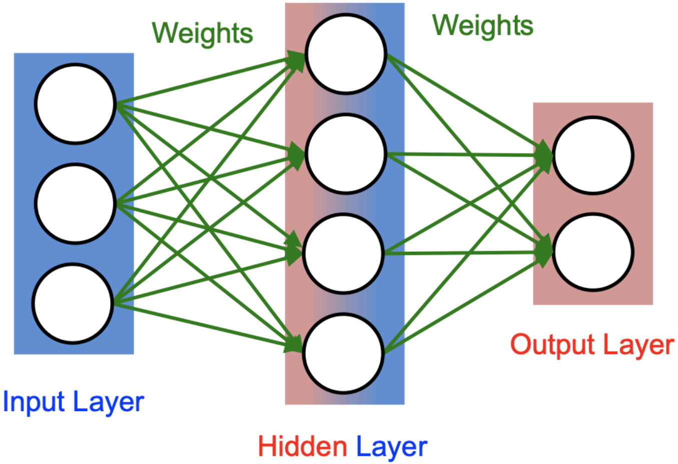
Machine Learning with Neural Networks



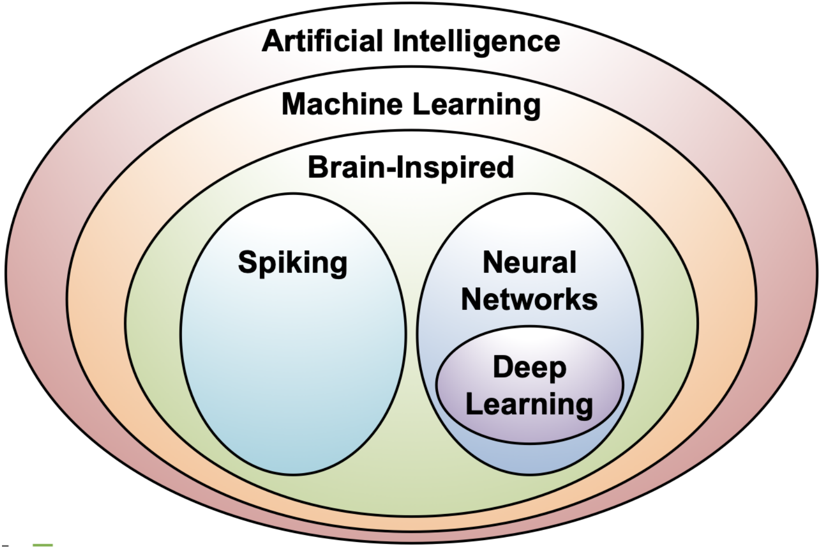
Neural Networks: Weighted Sum



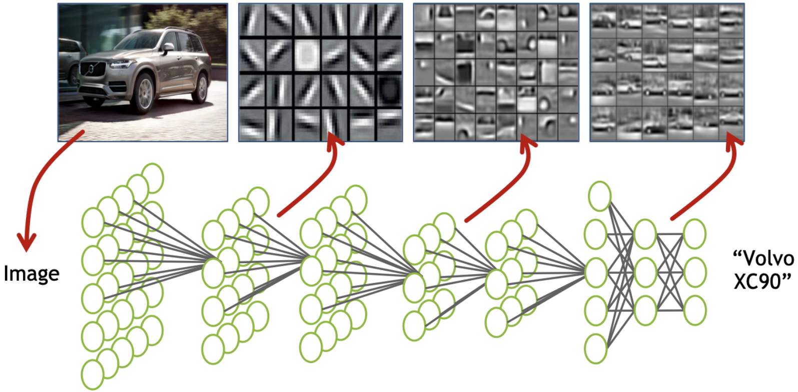
Many Weighted Sums



Deep Learning



What is Deep Learning?



Why DL is so popular?

Big Data Availability

GPU Acceleration

New ML Techniques

facebook — **350M images** uploaded per day

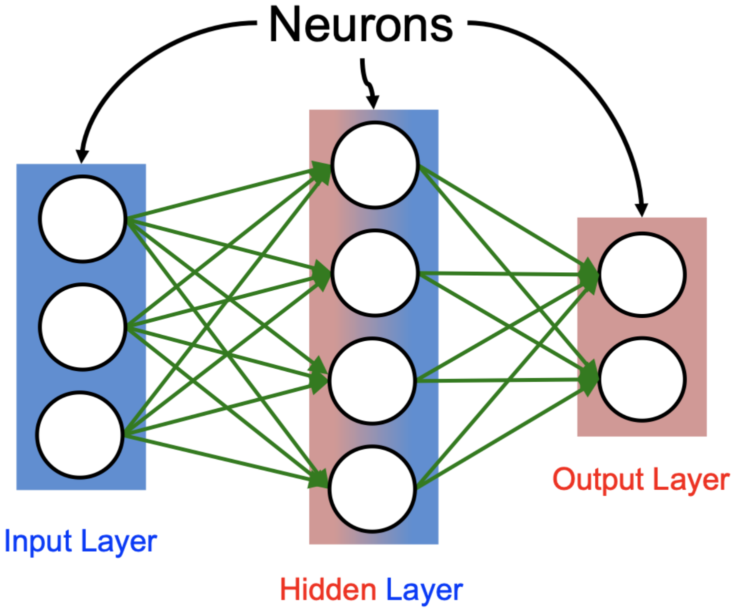
Walmart — **2.5 Petabytes** of customer data hourly

YouTube — **300 hours** of video uploaded every minute

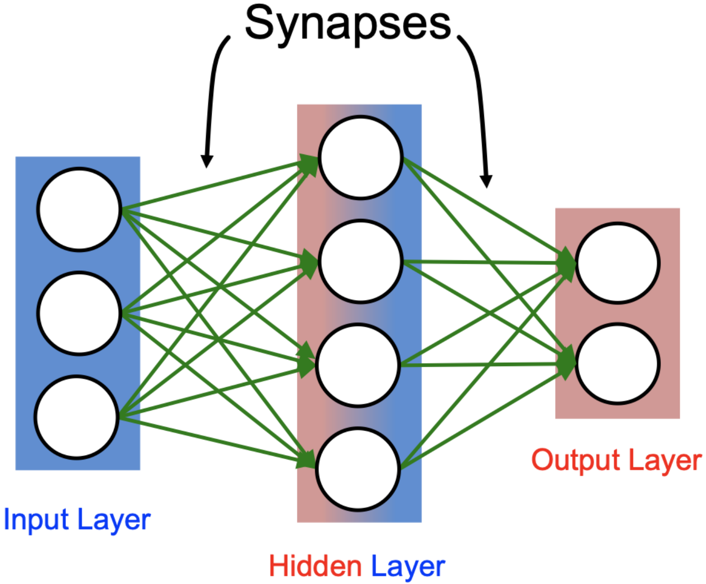
Section 4

Overview of DNN

Terminology - Neurons



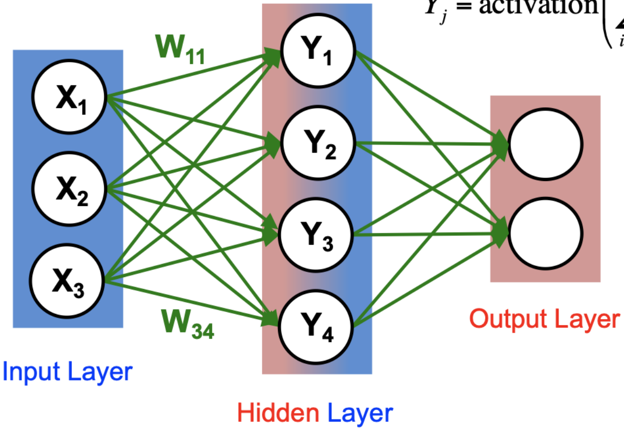
Terminology - Synapses



Terminology - Synapses

Each synapse has a weight for neuron activation

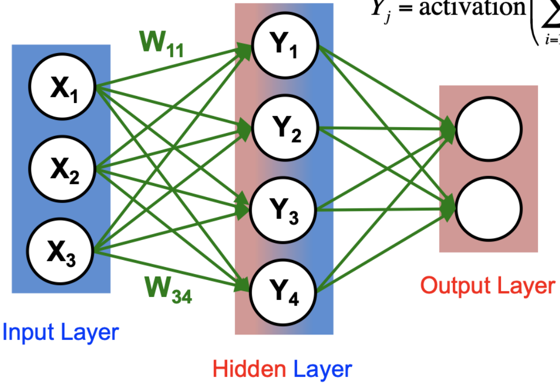
$$Y_j = \text{activation}\left(\sum_{i=1}^{3} W_{ij} \times X_i\right)$$



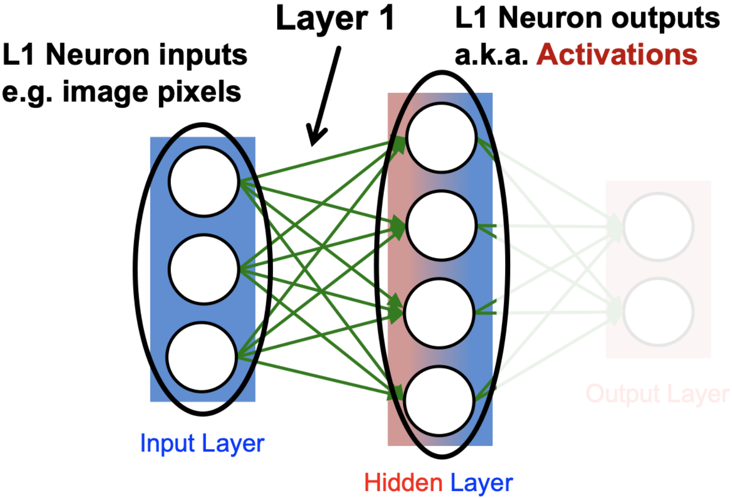
Terminology - Weight Sharing

Weight Sharing: multiple synapses use the **same weight value**

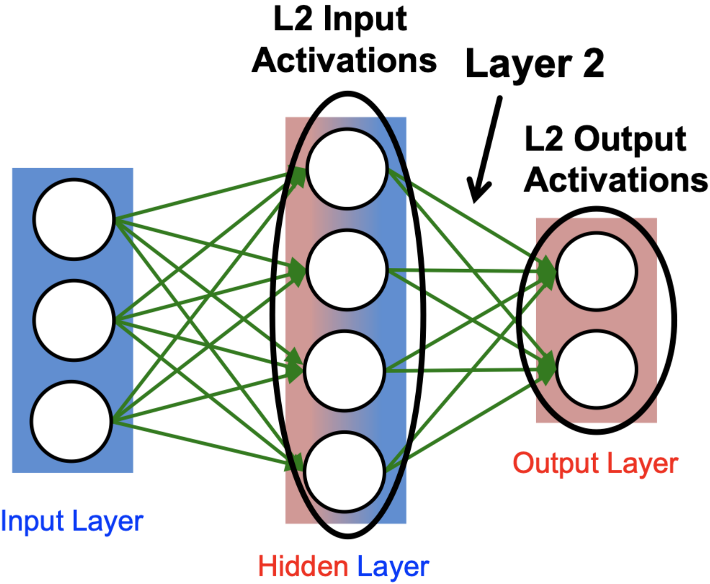
$$Y_j = \text{activation}\left(\sum_{i=1}^{3} W_{ij} \times X_i\right)$$



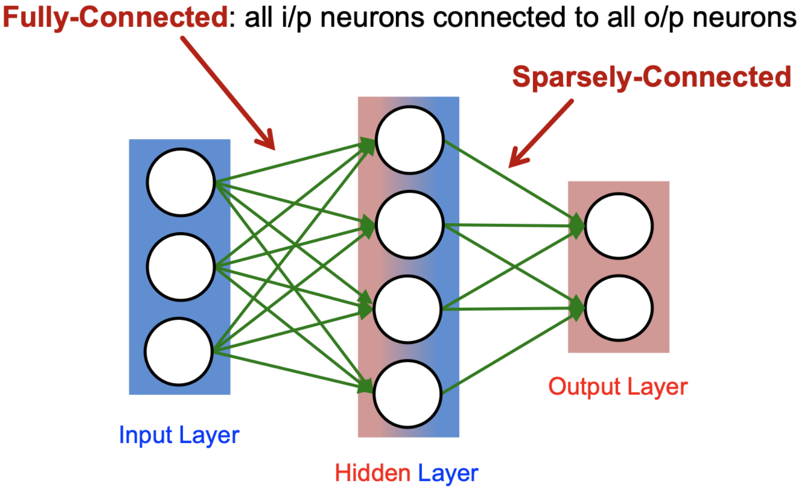
Terminology - Layers



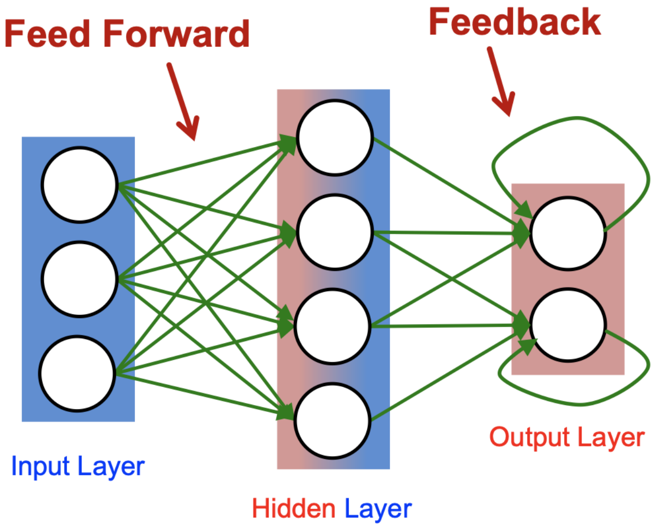
Terminology - Layers



Terminology - Connection pattern



Terminology - Connection pattern



Popular Types of DNNs

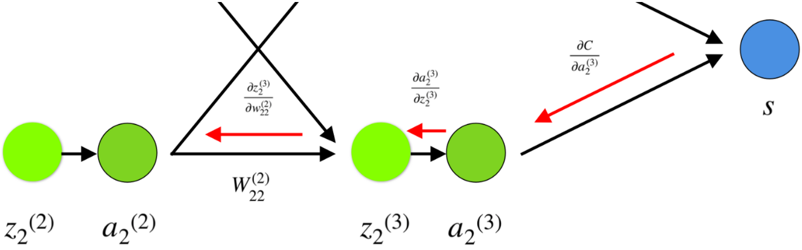
- Fully-ConnectedNN
  - feed forward, a.k.a. multilayer perceptron (MLP)
- ConvolutionalNN(CNN)
  - feed forward, sparsely-connected w/ weight sharing
- RecurrentNN(RNN)
  - feedback
- LongShort-TermMemory(LSTM)
  - feedback + storage

Inference vs. Training

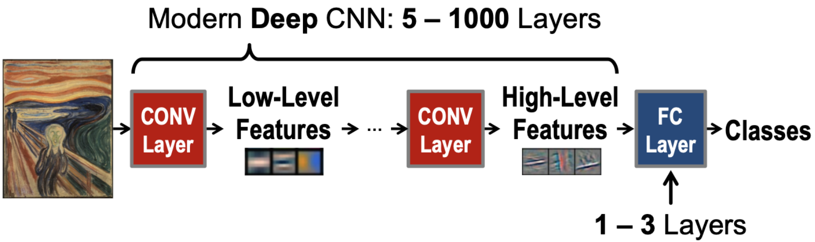
- Training: Determine weights (i.e. learn)
  - Supervised:
    - Training set has inputs and outputs, i.e., labeled
  - Unsupervised / Self-Supervised:
    - Training set is unlabeled
  - Semi-supervised:
    - Training set is partially labeled
  - Reinforcement:
    - Output assessed via rewards and punishments
- Inference: Apply weights to determine output

Backpropagation

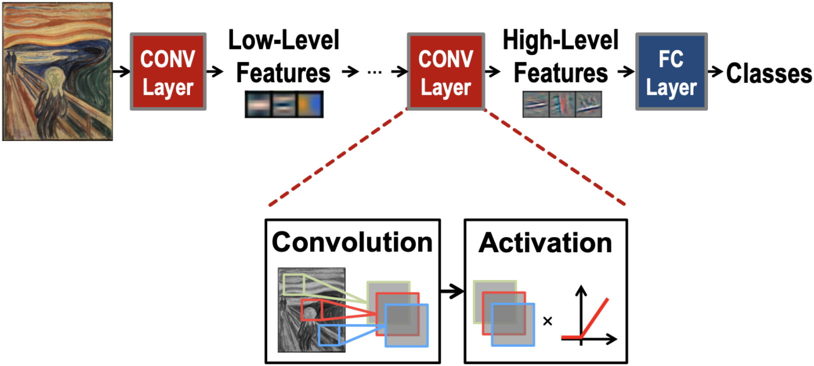
- Training consist on 2 phases:
  - Forward propagation: i.e. weighted sum
  - Back-propagation: algorithm that computes the gradient in weight space with respect to a loss function.



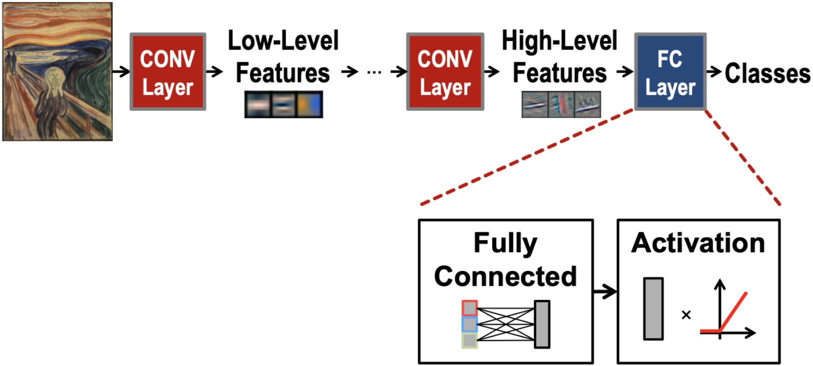
Deep Convolutional Neural Networks



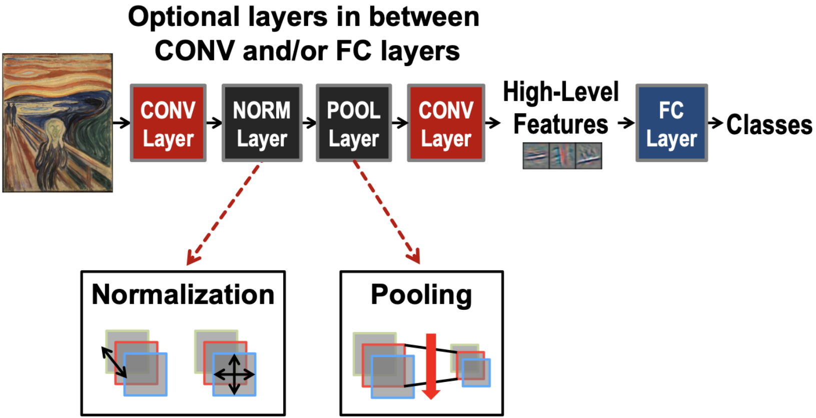
Deep Convolutional Neural Networks



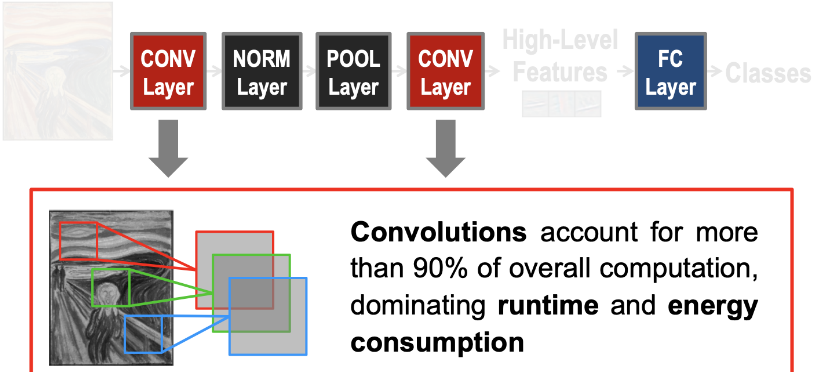
Deep Convolutional Neural Networks



Deep Convolutional Neural Networks



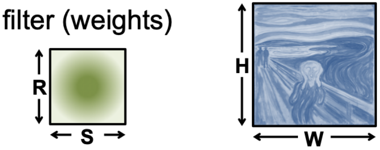
Deep Convolutional Neural Networks



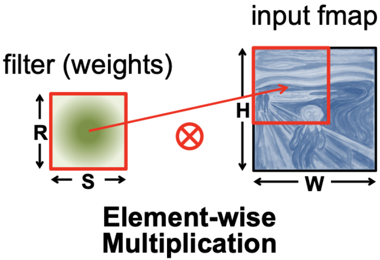
Convolutions account for more than 90% of overall computation, dominating **runtime** and **energy consumption**

Convolution (CONV) Layer

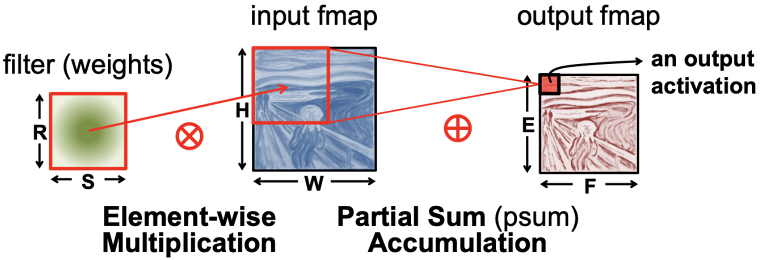
a plane of input activations
a.k.a. **input feature map (fmap)**



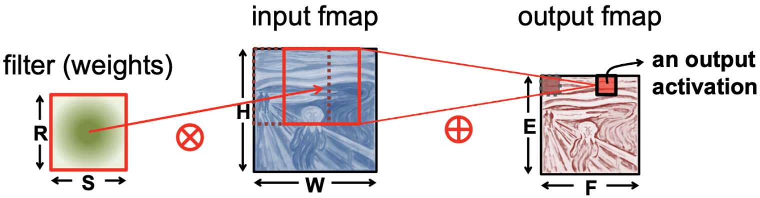
Convolution (CONV) Layer



Convolution (CONV) Layer

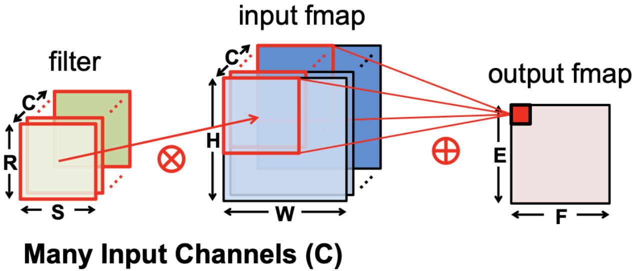


Convolution (CONV) Layer

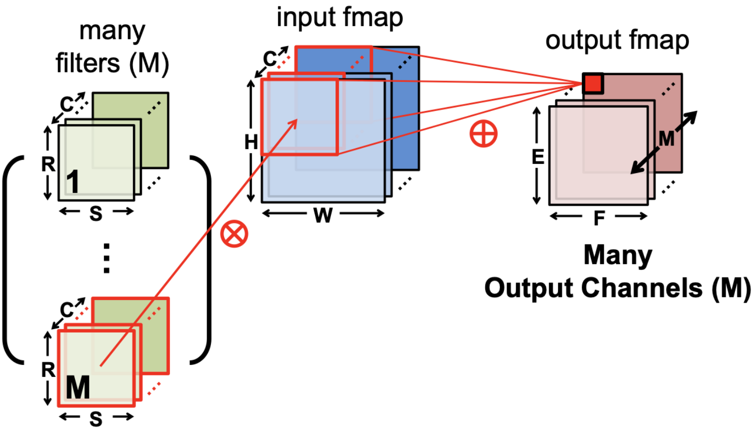


Sliding Window Processing

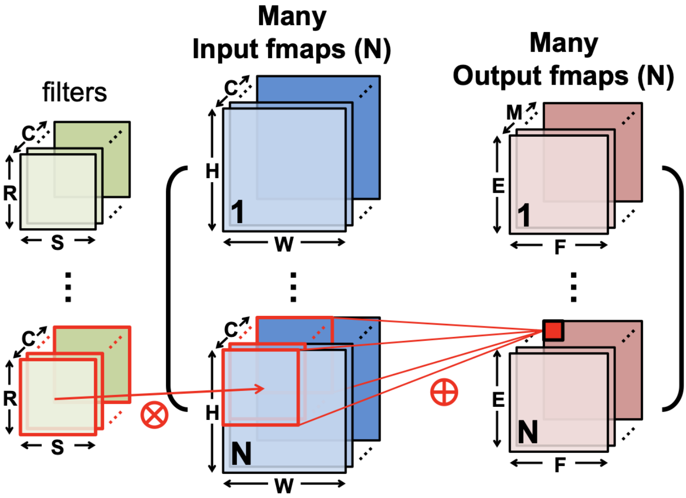
Convolution (CONV) Layer



Convolution (CONV) Layer



Convolution (CONV) Layer



CNN Decoder Ring

- **N – Number of input fmaps/output fmaps (batch size)**
- **C – Number of 2-D input fmaps /filters (channels)**
- **H – Height of input fmap (activations)**
- **W – Width of input fmap (activations)**
- **R – Height of 2-D filter (weights)**
- **S – Width of 2-D filter (weights)**
- **M – Number of 2-D output fmaps (channels)**
- **E – Height of output fmap (activations)**
- **F – Width of output fmap (activations)**

CONV Layer Tensor Computation

Output fmaps (O) Biases (B) Input fmaps (I) Filter weights (W)

$$\underline{\mathbf{O}[n][m][x][y]} = \text{Activation}(\underline{\mathbf{B}[m]} + \sum_{i=0}^{R-1}\sum_{j=0}^{S-1}\sum_{k=0}^{C-1} \underline{\mathbf{I}[n][k][Ux+i][Uy+j]} \times \underline{\mathbf{W}[m][k][i][j]}),$$

$$0 \le n < N, 0 \le m < M, 0 \le y < E, 0 \le x < F,$$

$$E = (H - R + U)/U, F = (W - S + U)/U.$$

Shape Parameter	Description
N	fmap batch size
M	# of filters / # of output fmap channels
C	# of input fmap/filter channels
H/W	input fmap height/width
R/S	filter height/width
E/F	output fmap height/width
U	convolution stride

CONV Layer Implementation

Naïve 7-layer for-loop implementation:

for each output fmap value

convolve a window and apply activation

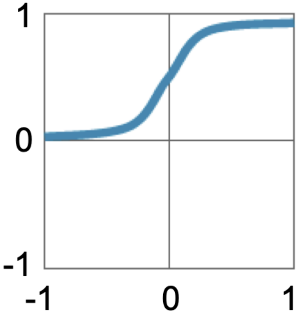
```
for (n=0; n<N; n++) {
    for (m=0; m<M; m++) {
        for (x=0; x<F; x++) {
            for (y=0; y<E; y++) {

                O[n][m][x][y] = B[m];
                for (i=0; i<R; i++) {
                    for (j=0; j<S; j++) {
                        for (k=0; k<C; k++) {
                            O[n][m][x][y] += I[n][k][Ux+i][Uy+j] × W[m][k][i][j];
                        }
                    }
                }

                O[n][m][x][y] = Activation(O[n][m][x][y]);
            }
        }
    }
}
```

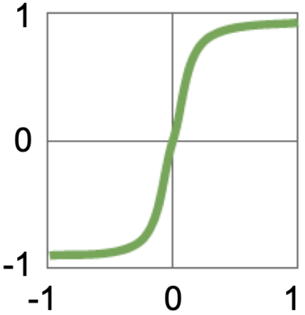
Traditional Activation Functions

Sigmoid



$y=1/(1+e^{-x})$

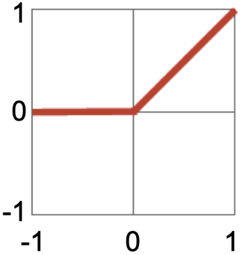
Hyperbolic Tangent



$y=(e^{x}-e^{-x})/(e^{x}+e^{-x})$

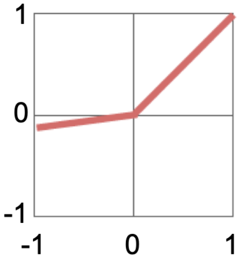
Modern Activation Functions

Rectified Linear Unit (ReLU)



$$y=\max(0,x)$$

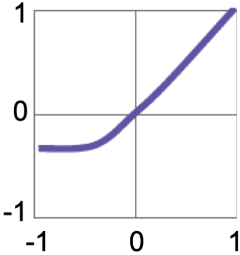
Leaky ReLU



$$y=\max(\alpha x,x)$$

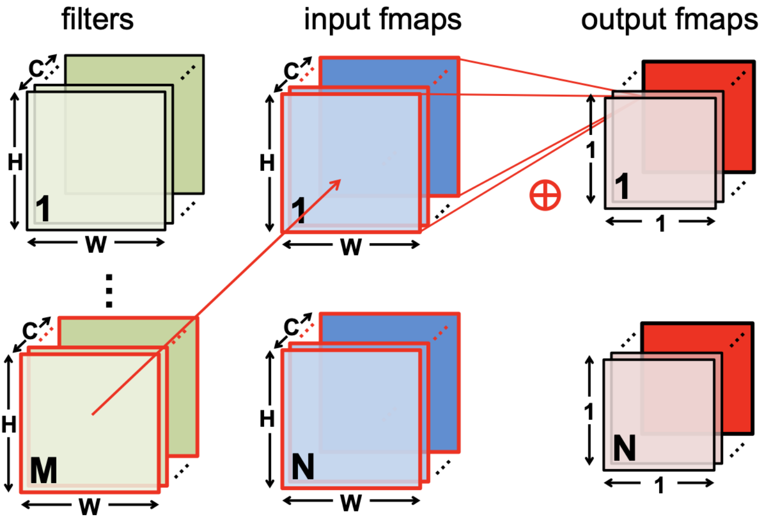
α = small const. (e.g. 0.1)

Exponential LU



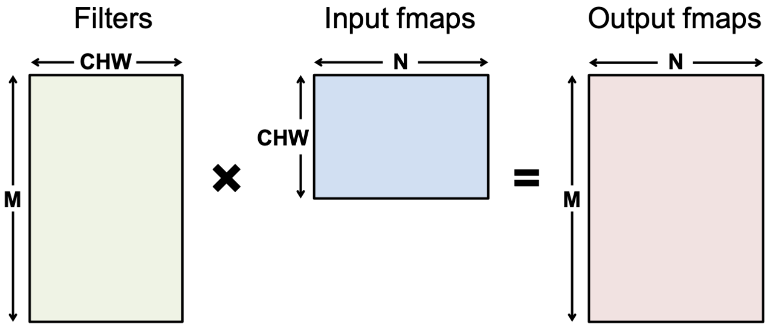
$$y=\begin{cases} x, & x\geq 0 \\ \alpha(e^{x}-1), & x<0 \end{cases}$$

FC Layer – from CONV Layer POV



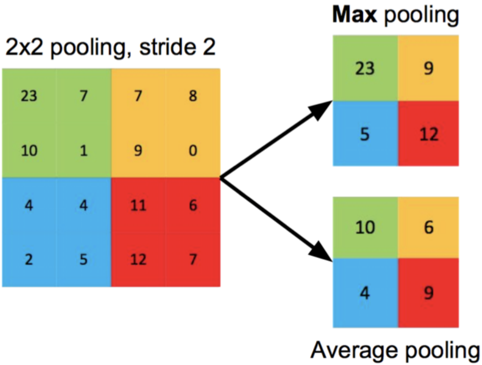
Fully-Connected (FC) Layer

- Height and width of output fmaps are 1 (E = F = 1)
- Filters as large as input fmaps (R = H, S = W)
- Implementation: **Matrix Multiplication**



Pooling (POOL) Layer

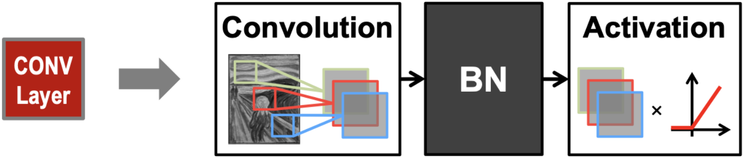
- Reduce resolution of each channel independently
- Overlapping or non-overlapping → depending on stride



Increases translation-invariance and noise-resilience

Normalization (NORM) Layer

- **Batch Normalization (BN)**
  - Normalize activations towards mean=0 and std. dev.=1 based on the statistics of the training dataset
  - put **in between** **CONV/FC** and **Activation function**



Believed to be key to getting high accuracy and faster training on very deep neural networks.

Section 5

Reduce storage/compute

Approaches

- **Reduce size of operands for storage/compute**
  - Floating point → Fixed point
  - Bit-width reduction
  - Non-linear quantization
- **Reduce number of operations for storage/compute**
  - Exploit Activation Statistics (Compression)
  - Network Pruning
  - Compact Network Architectures

What is quantization?

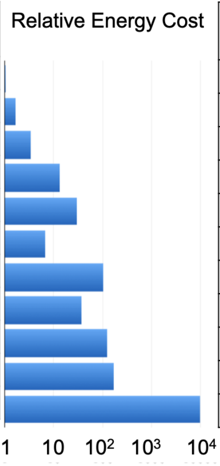
- **Precision** refers to the **number of levels**
  - Number of bits = $\log_2$ (number of levels)

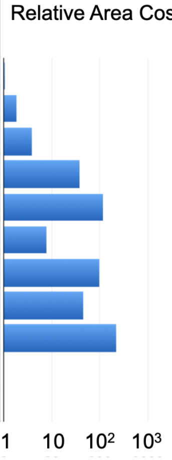
- **Quantization:** mapping data to a smaller set of **levels**
  - Linear, e.g., fixed-point
  - Non-linear
    - Computed (e.g., floating point, log-domain)
    - Table lookup (e.g., learned)

Objective: Reduce size to improve speed and/or reduce energy while preserving accuracy

Cost of Operations

Operation:	Energy (pJ)	Area (μm^2)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FP Add	0.4	1360
32b FP Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A



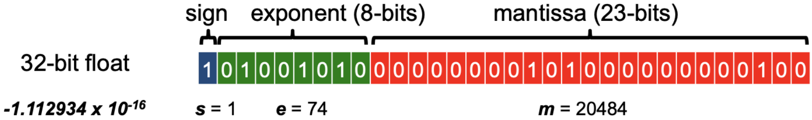


Number representation

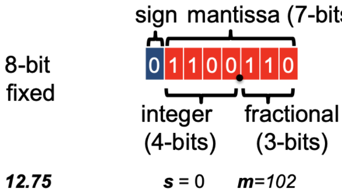
	S	E	M	Range	Accuracy
FP32	1	8	23	$10^{-38} – 10^{38}$	.000006%
FP16	1	5	10	$6x10^{-5}$ - $6x10^{4}$	.05%
Int32	1		31	$0 – 2x10^{9}$	½
Int16	1		15	$0 – 6x10^{4}$	½
Int8	1		7	0 – 127	½

Floating Point -> Fixed Point

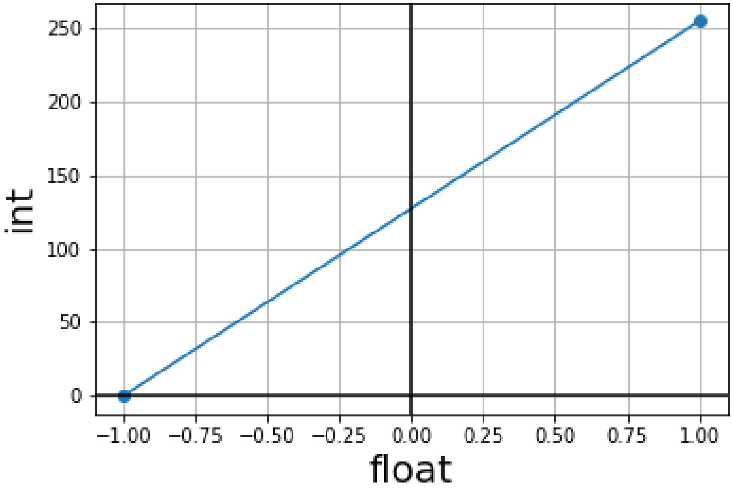
Floating Point



Fixed Point



FP [-1,1) -> INT8 [0,256)



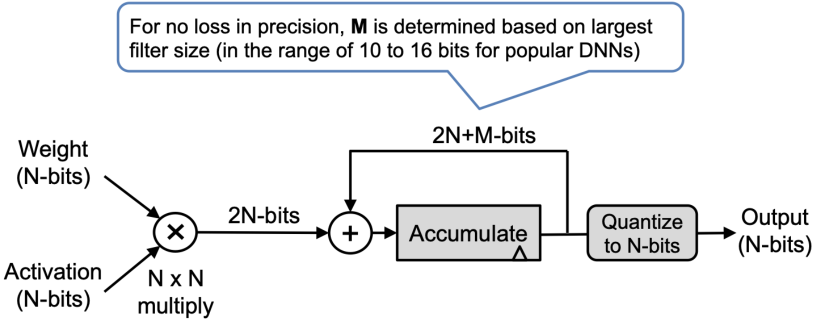
FP [-1,1) -> INT8 [-128,128)

$$\begin{pmatrix} -0.18120981 & -0.29043840 \\ 0.49722983 & 0.22141714 \end{pmatrix} \begin{pmatrix} 0.77412377 \\ 0.49299395 \end{pmatrix} = \begin{pmatrix} -0.28346319 \\ 0.49407474 \end{pmatrix}$$

$$x \mapsto \left\lfloor 128 \frac{x}{a} \right\rfloor \qquad x \mapsto \frac{ax}{16384}$$

$$\begin{pmatrix} -24 & -38 \\ 63 & 28 \end{pmatrix} \begin{pmatrix} 99 \\ 63 \end{pmatrix} = \begin{pmatrix} 4770 \\ 8001 \end{pmatrix} \qquad \begin{pmatrix} -0.2911377 \\ 0.48834229 \end{pmatrix}$$

N-bit precision

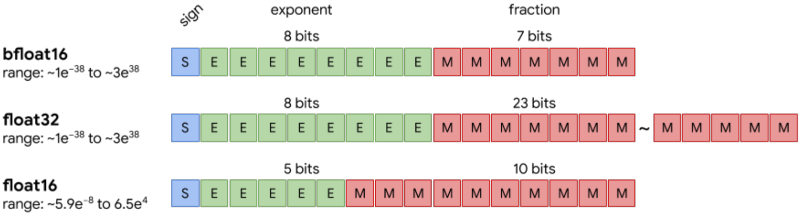


FP formats for DL

- FP32. the standard format for DNN
- FP16: little HW support, only useful for GPUs of for saving storage
- INT8: saves storage/power & improves speedup but significant accuracy loss
- New formats are needed
  - BFLOAT16
  - TF32
  - Posit?

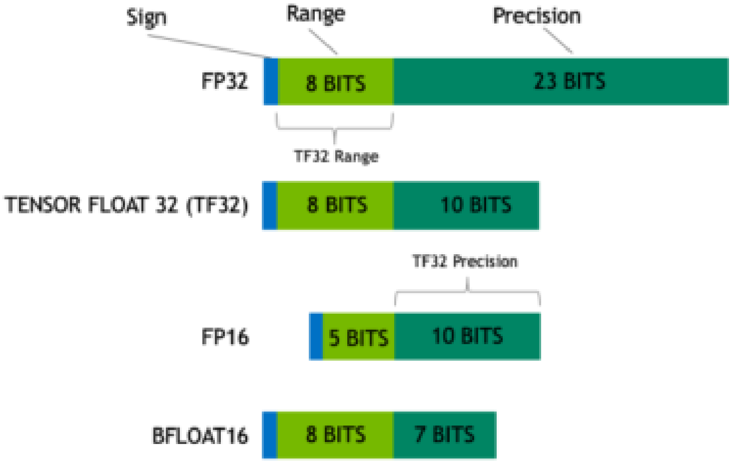
BFLOAT16

- Brain Floating Point Format (Google)



TF32

- Tensor Float 32



Quantization strategies

- **Post-training**
  - Train de model using *float32* weights and inputs
  - Then quantize weights
  - Simple to apply, but higher accuracy loss
- **Quantization-aware training**
  - Quantize the weights (or even activations) during training
  - This has the best result, but it is more involved

Network pruning