
# Advanced Topics in Artificial Intelligence Chipset: Algorithm & Hardware Co-design

**Kaisheng Ma**

**Spring, 2025**

# Course Info

- Kaisheng Ma, IIIS, THU kaisheng@tsinghua.edu.cn
- Q&A through RainPlatform, Wechat Group

# Homeworks and Exams

- **Homeworks 25%**
- **Final Project 80% (individual 40%, together 40%)**
- **Target： Get familiar with some new optimization methods, and implement it.**
- **Find the best algorithm and try it on target LLM, and we will check the overall performance acceleration.**

- **Q&A time： One Hour Before Class**

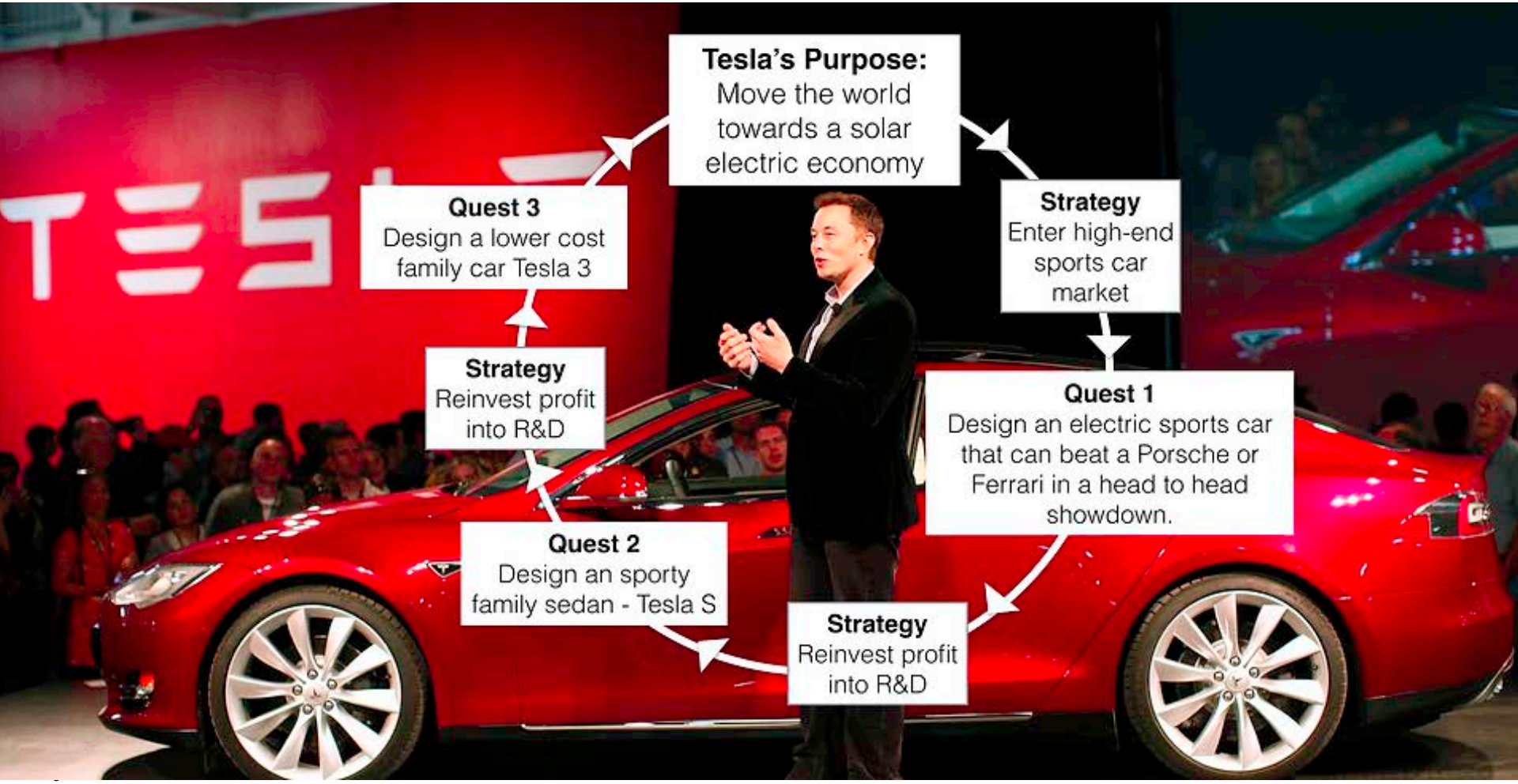
# You will learn from this course

- How to ready AI papers
- History and basic knowledge of Pruning\Quantization\Low-rank\Distillation
  - Basically most of the work Deepseek has done.

- How to make your own MASTER PLAN

# Elon Musk's Master Plan

- **How to make your own MASTER PLAN**




Source: https://www.tesla.com/secret-master-plan

# You will learn from today's class

- **Appetizer: How to read AI Papers**
- **Background of Chips & Self-intro**

- **CNN roadmap**

# Materials Source and Contributors

**Song Han**

**MIT**

**Vivienne Sze**

**MIT**

**Yanzhi Wang**

**Northeastern University**

Ref:https://hanlab.mit.edu/courses/2024-fall-65940
Ref: http://eyeriss.mit.edu/tutorial.html
Ref: https://web.northeastern.edu/yanzhiwang/

# Outline

- **Lesson 1&2**
  - **Overview of Deep Neural Networks**
  - **Software Level - Popular DNN Algorithms and Datasets**
  - **Hardware Level - DNN Kernel Computation**

- **Lesson 3&4**
  - **Pruning Optimizations Directions:**
    **- Compact Model Design**
    **- Pruning: Special Topic**
    **- Low-rank Matrix/Dictionary**
    **- Distillation: Special Topic**

# Outline

- **Lesson 5**
  - **Quantization： 8Bit, Tannery, Binary, XOR**

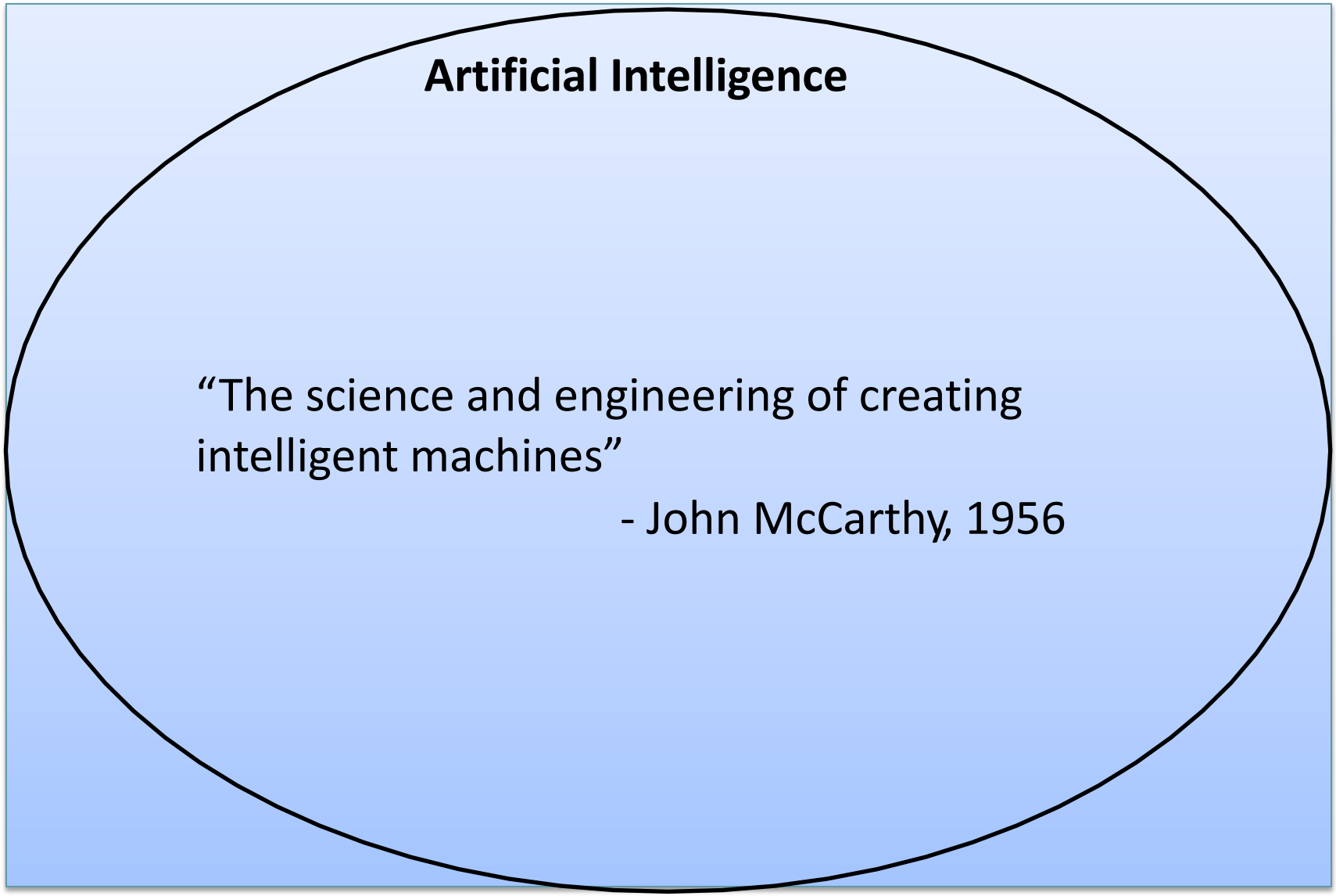
- **Lesson 6, 7&8**
  - **DNN Accelerators - General Architecture**
    **- Case Study: TPU V1, V2, Huawei Qilin Arch, DianNao Series, SCNN, etc.**
  - **Sparse DNN Accelerators**
    **- Case Study: Pattern Pruning and MUSE Arch.**
    **- Case Study: Block-circuit Algorithms and Hardware**
- **Lesson 9 (if time allowed)**
  - **Future Directions: Processing In/Near Memory**

# Goals

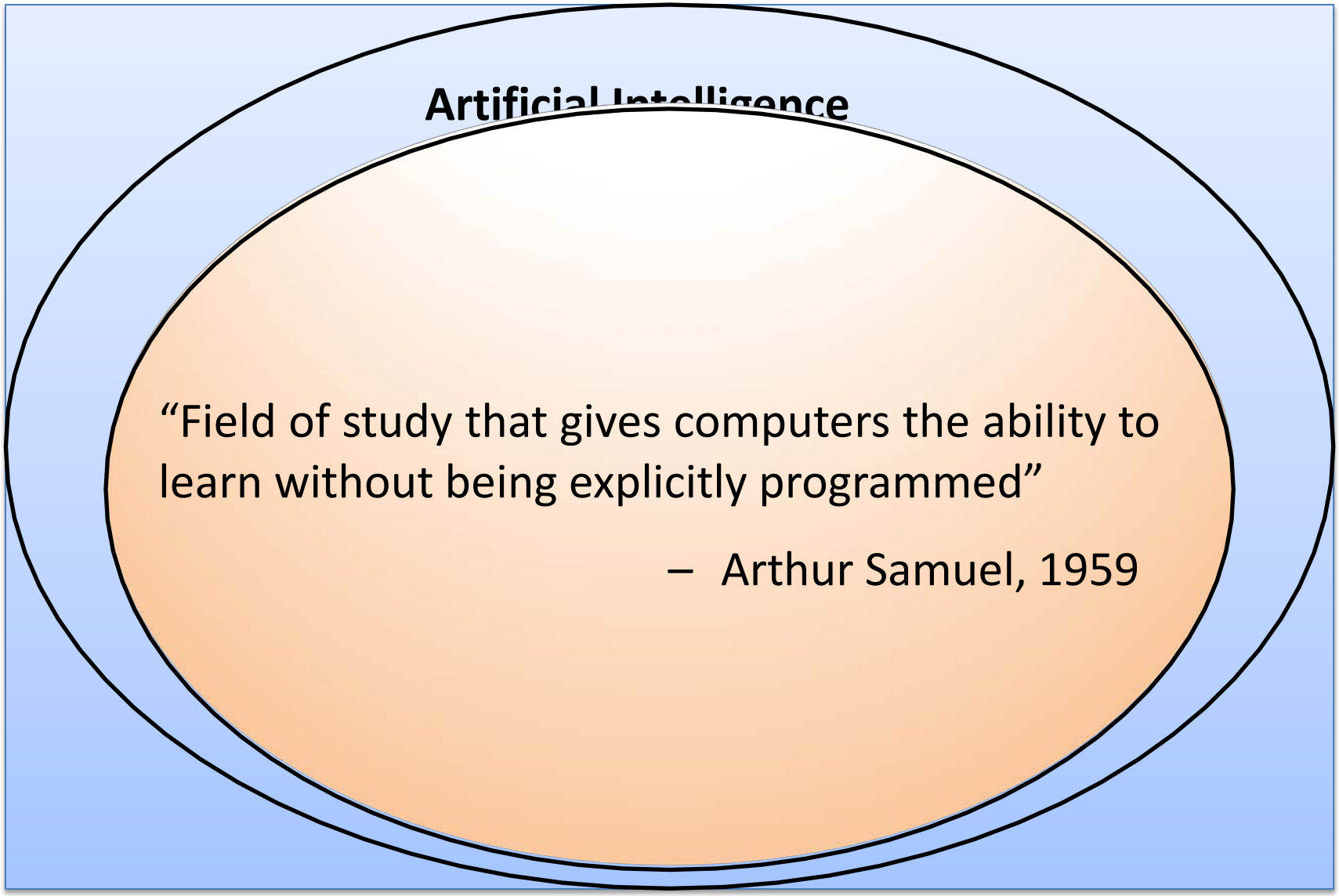
- **Understand the key design considerations for DNNs**
- **Be able to evaluate different implementations of DNN with benchmarks and comparison metrics**
- **Understand the tradeoffs between various architectures and platforms**
- **Assess the utility of various optimization approaches**
- **Understand recent implementation trends and opportunities, especially the Hardware&Software Co-design domain**

# Background of Deep Neural Networks

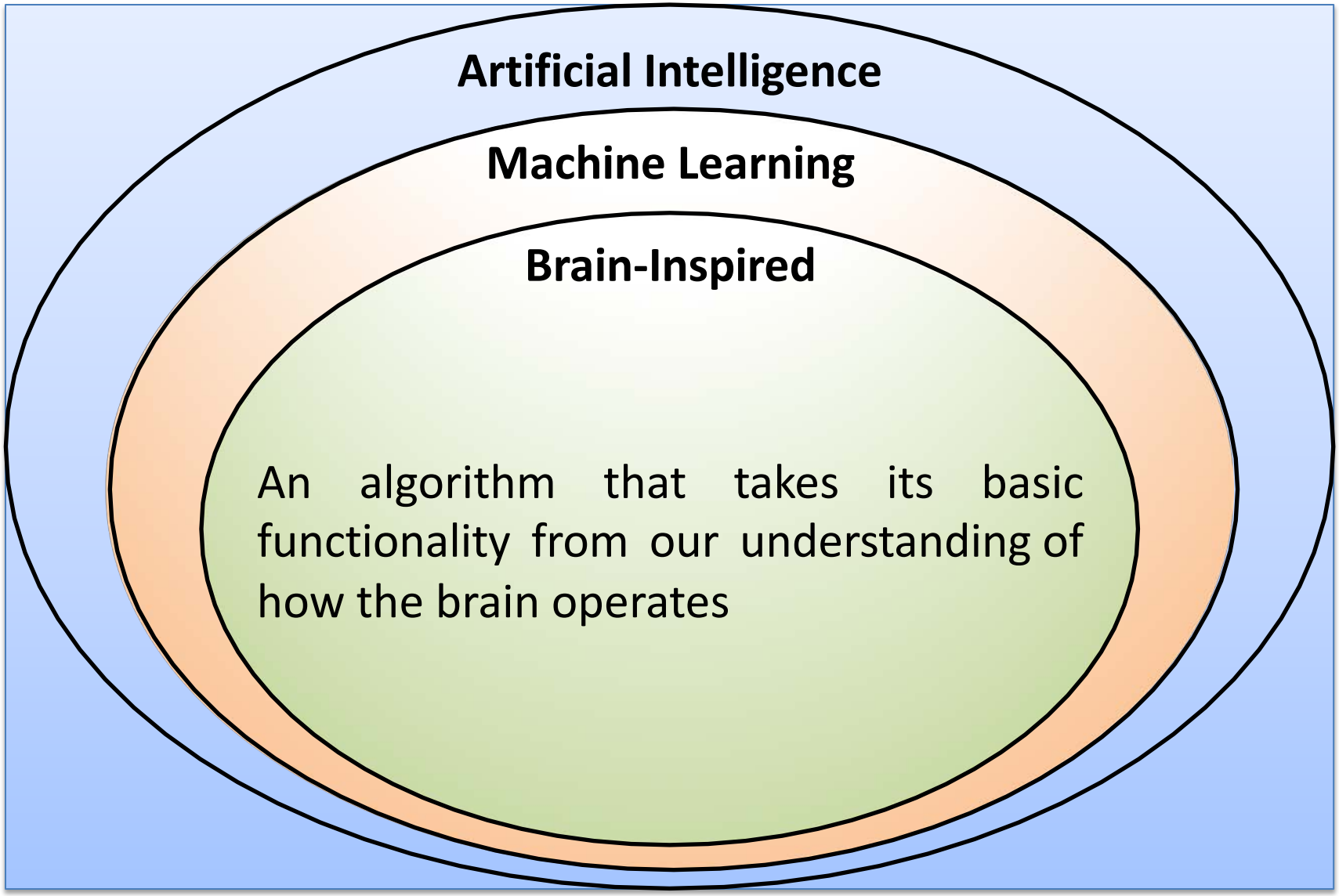
# Artificial Intelligence



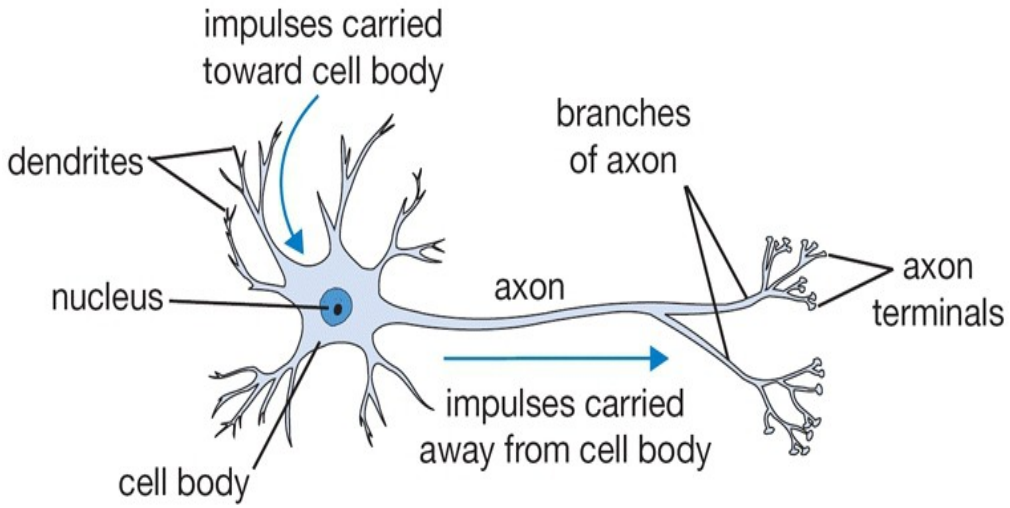
# AI and Machine Learning

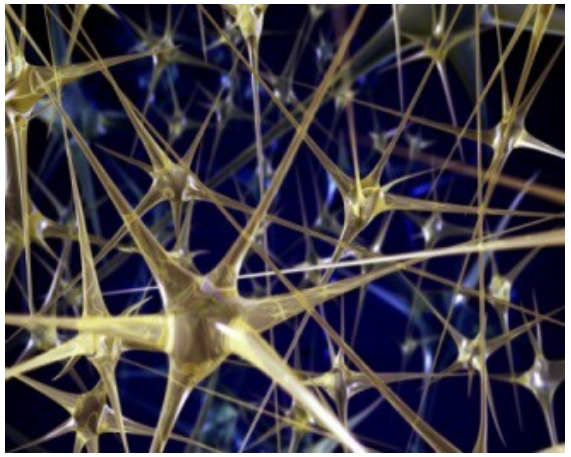


# Brain-Inspired Machine Learning



# How Does the Brain Work?



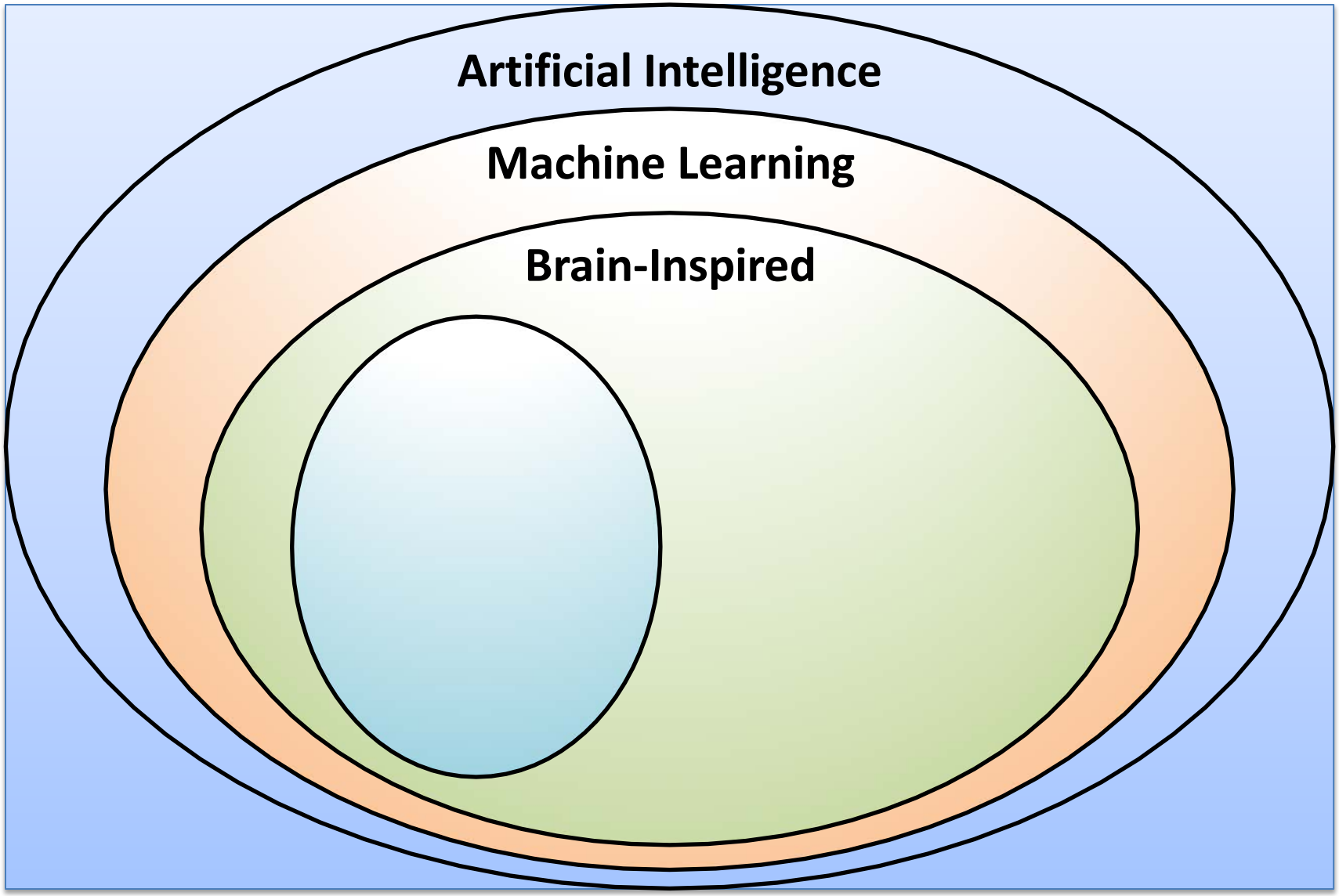



- The basic computational unit of the brain is a **neuron**
  à 86B neurons in the brain
- Neurons are connected with nearly **$10^{14} – 10^{15}$ synapses**
- Neurons receive input signal from **dendrites** and produce output signal along **axon**, which interact with the dendrites of other neurons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



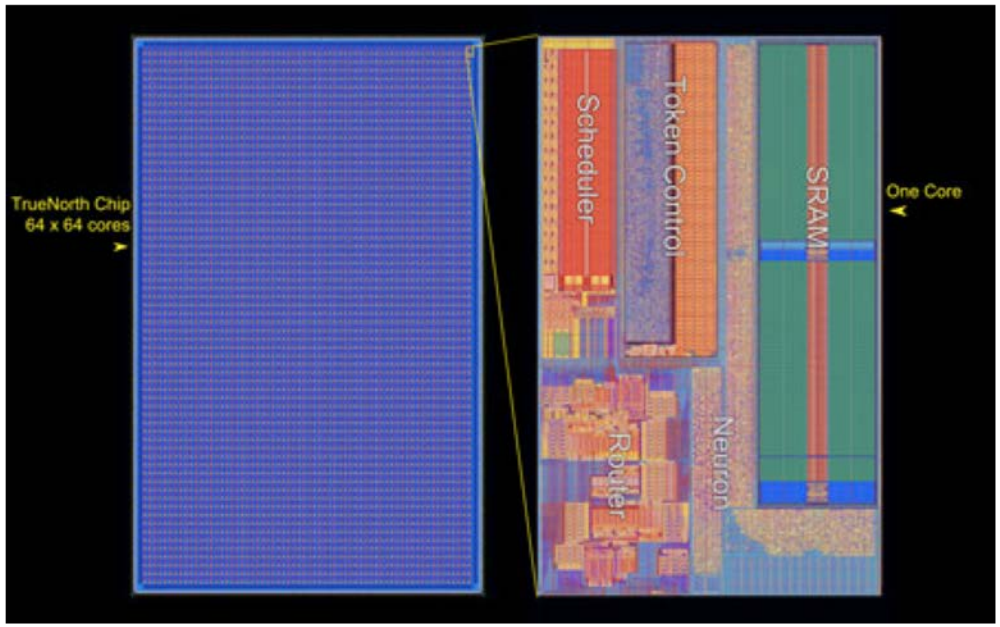
Image Source: Stanford

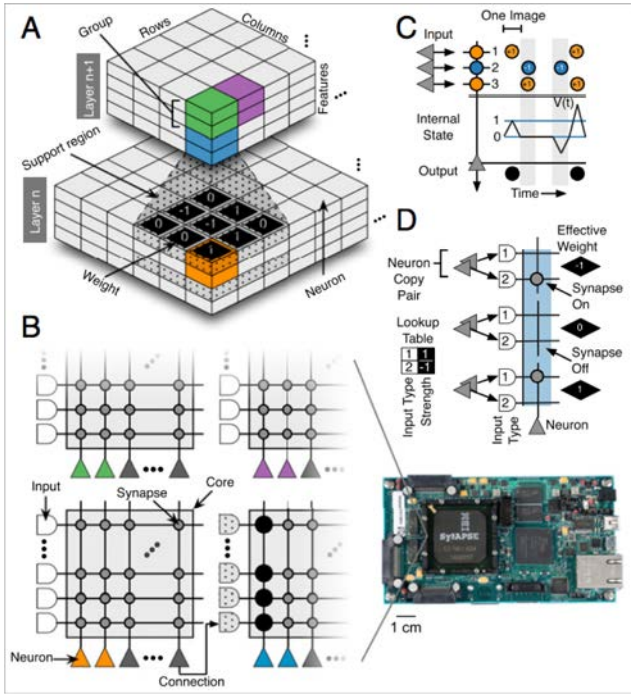
# Spiking-based Machine Learning



# Spiking Architecture

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

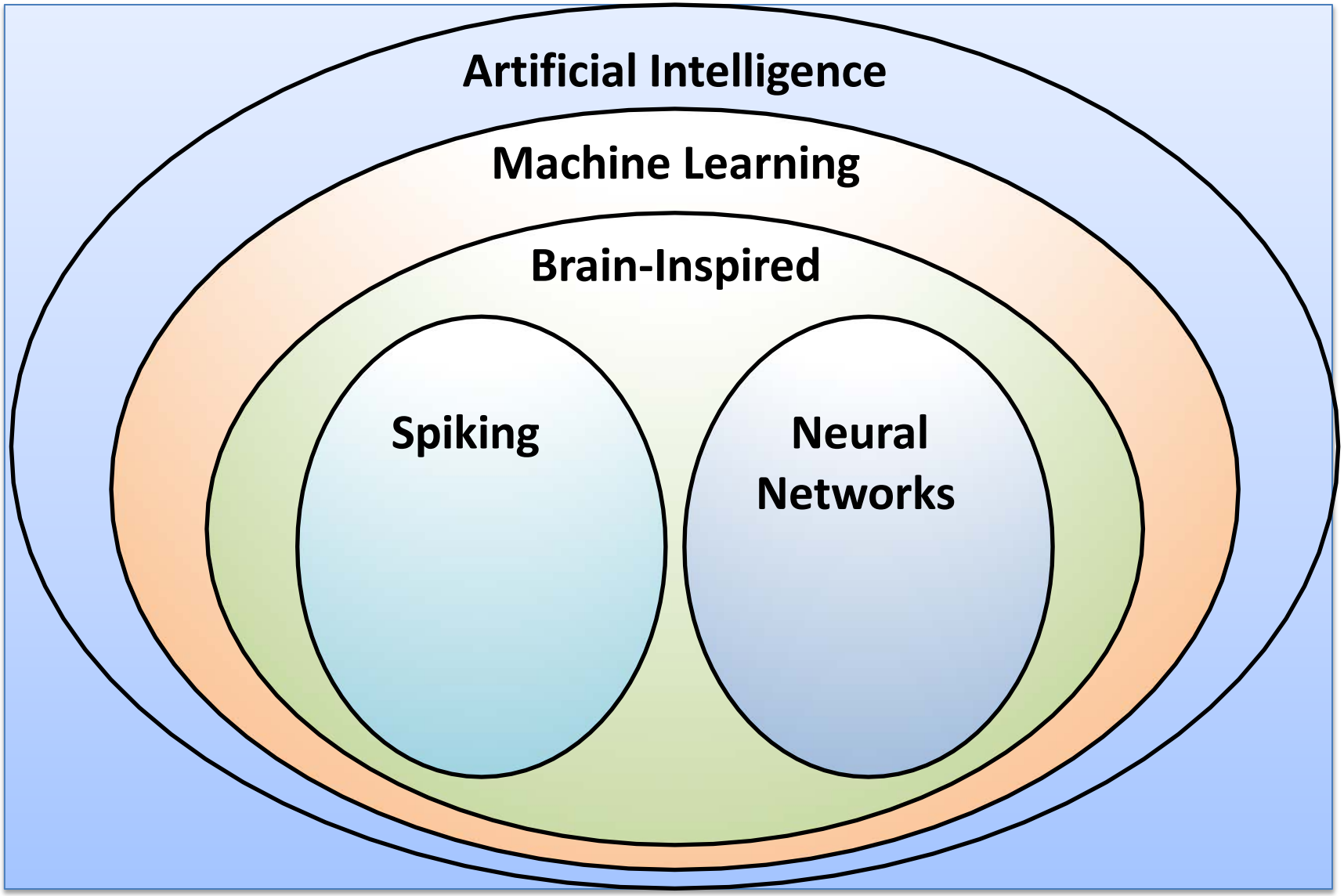



[Merolla et al., Science 2014; Esser et al., PNAS 2016]

http://www.research.ibm.com/articles/brain-chip.shtml

# Machine Learning with Neural Networks



# Neural Networks: Weighted Sum

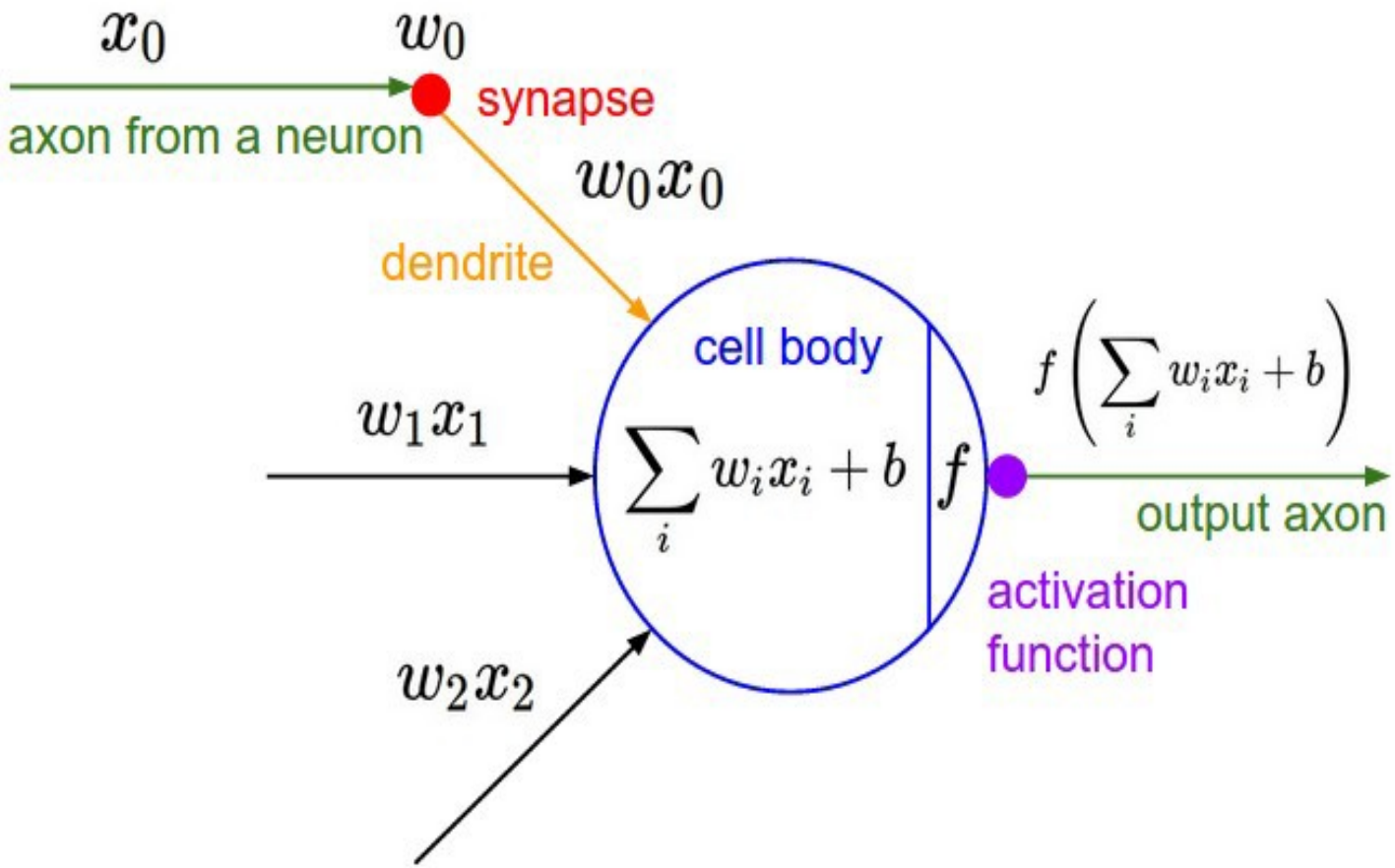
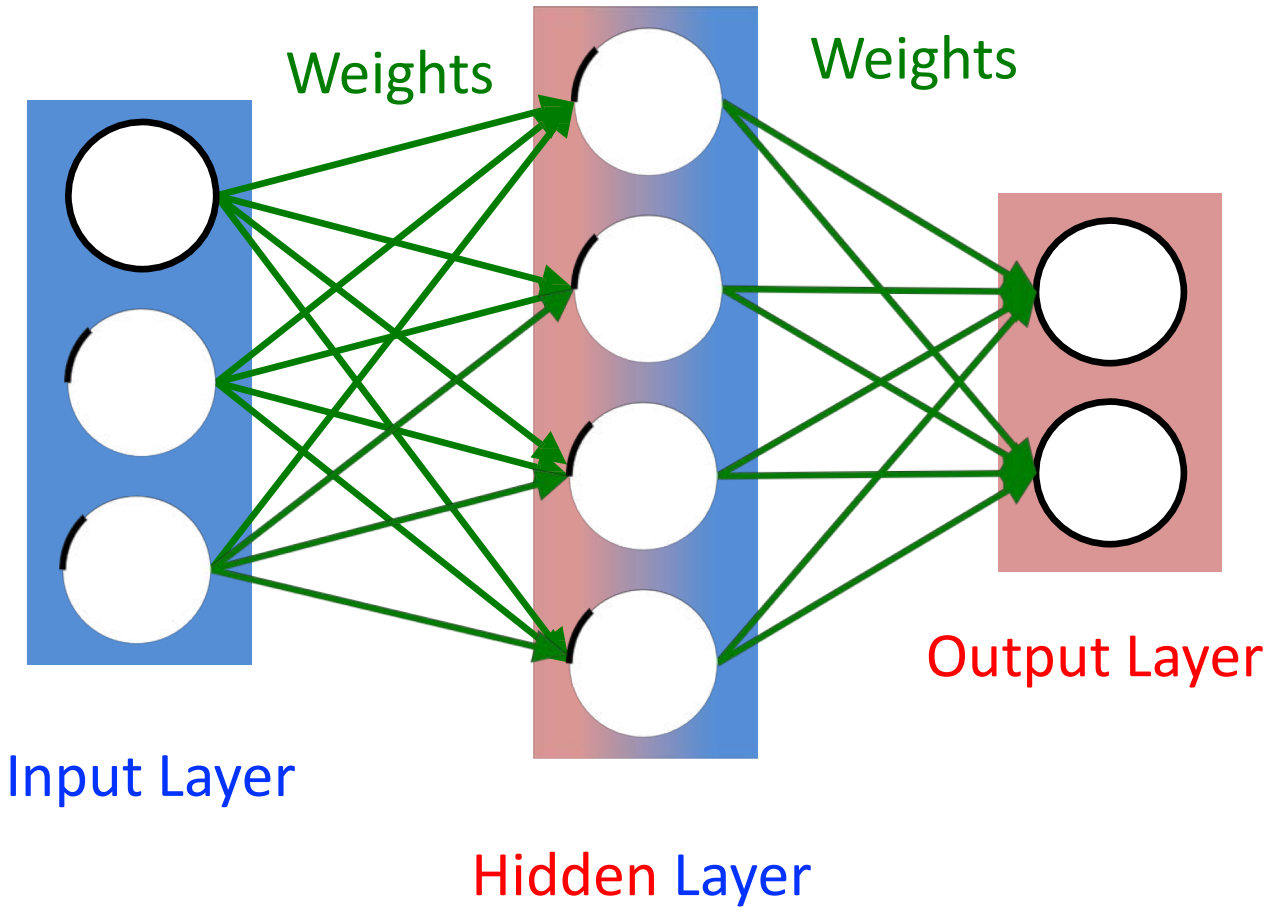
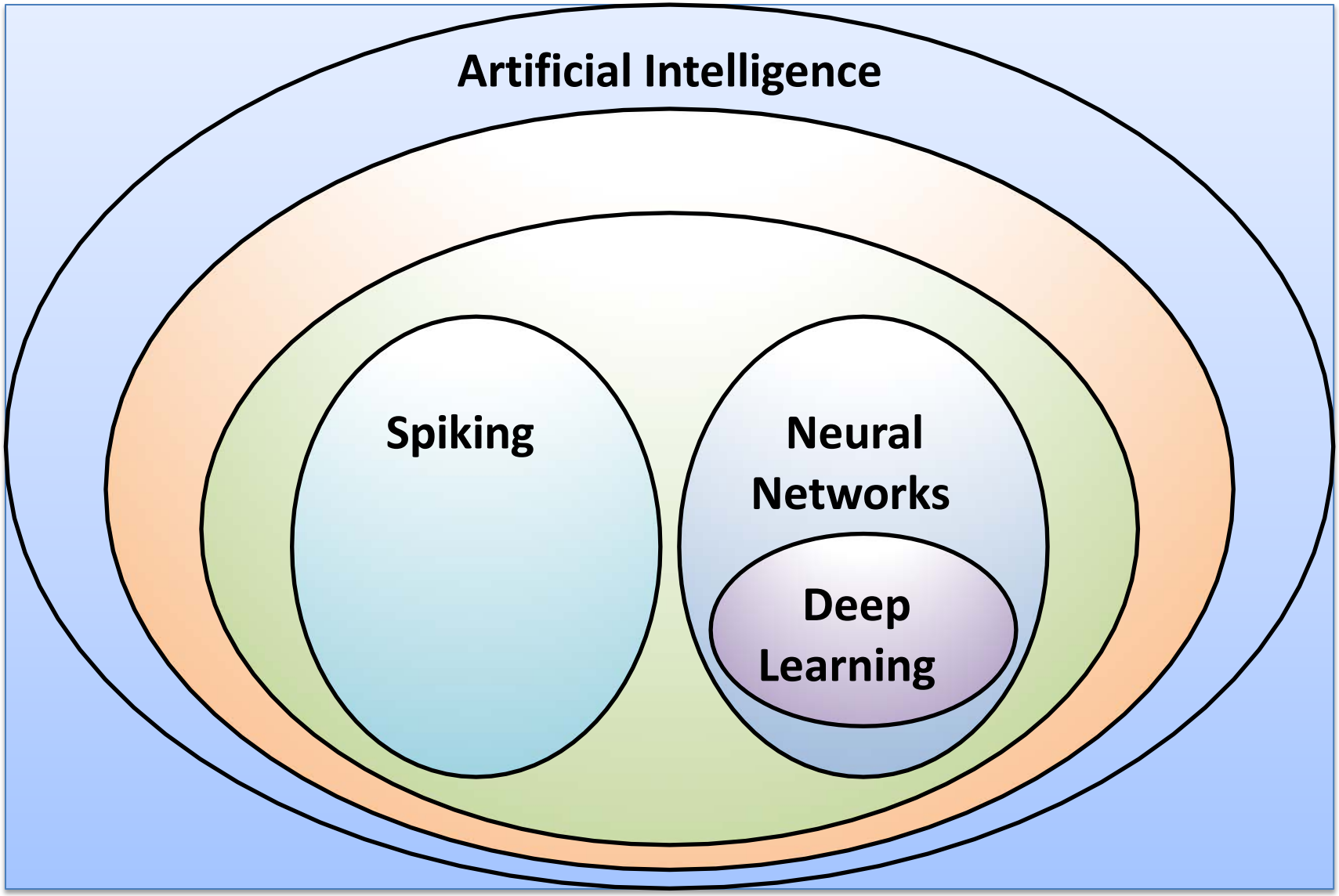



Image Source: Stanford

# Many Weighted Sums



# Deep Learning



# What is Deep Learning?

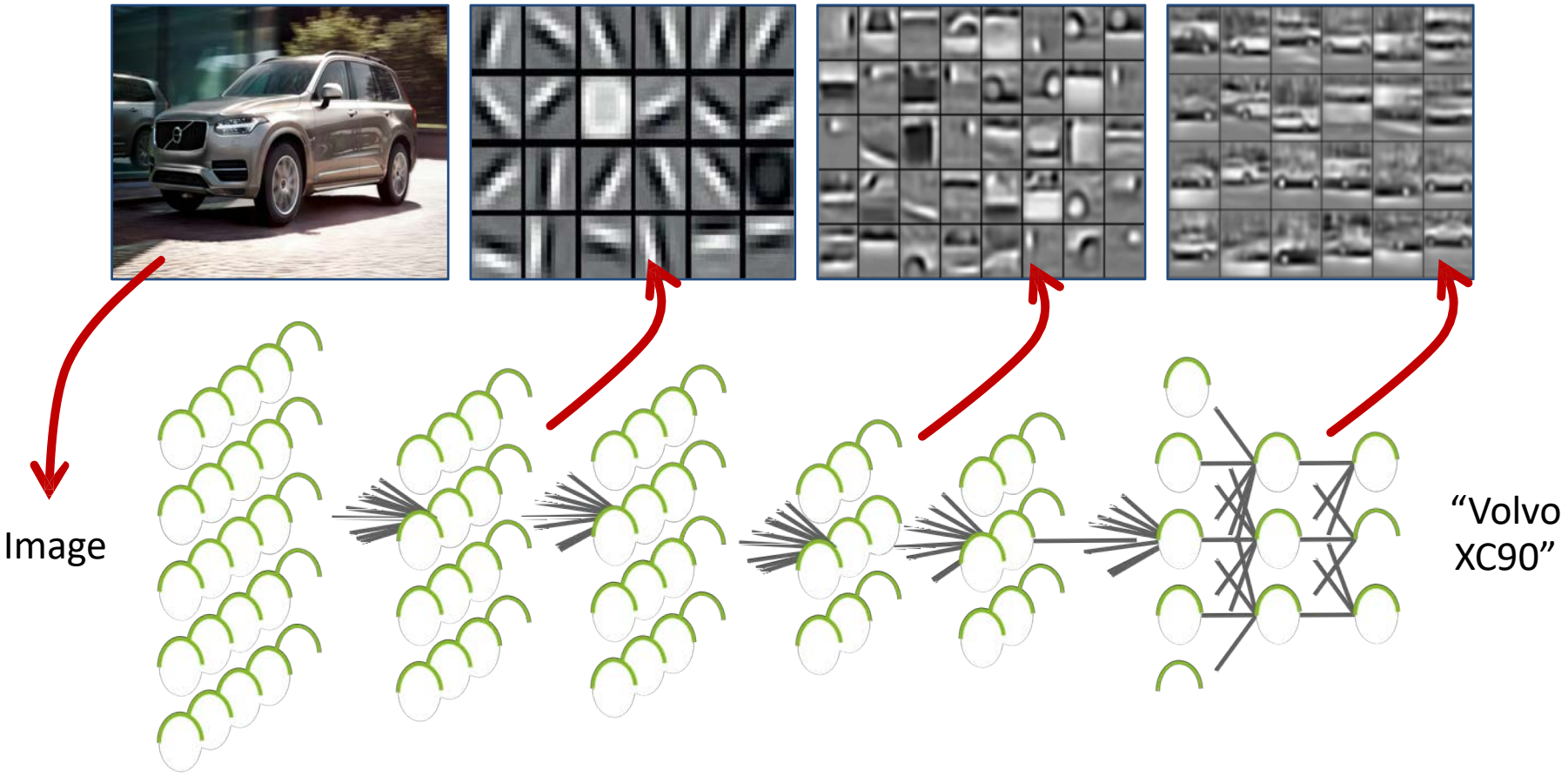
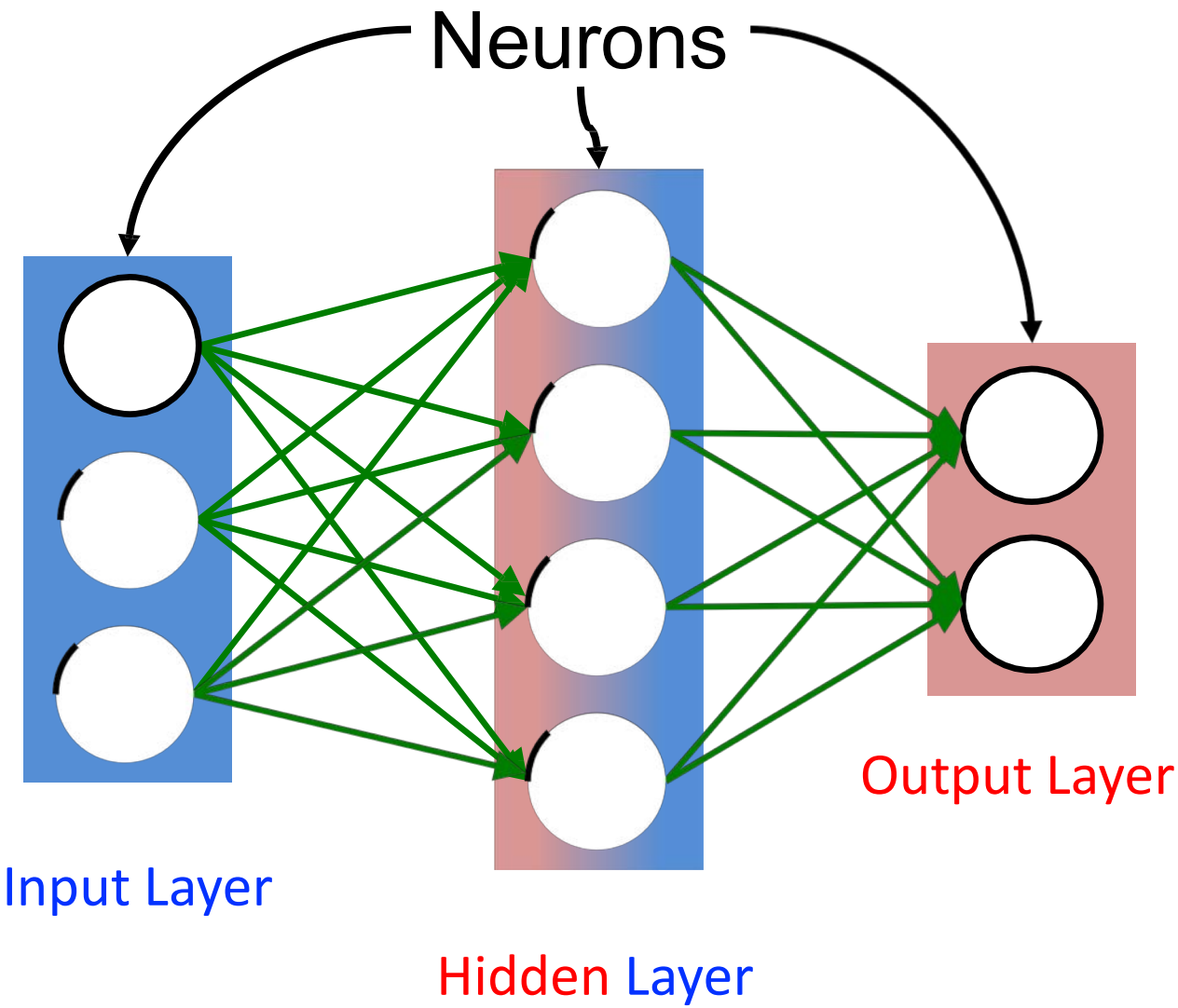


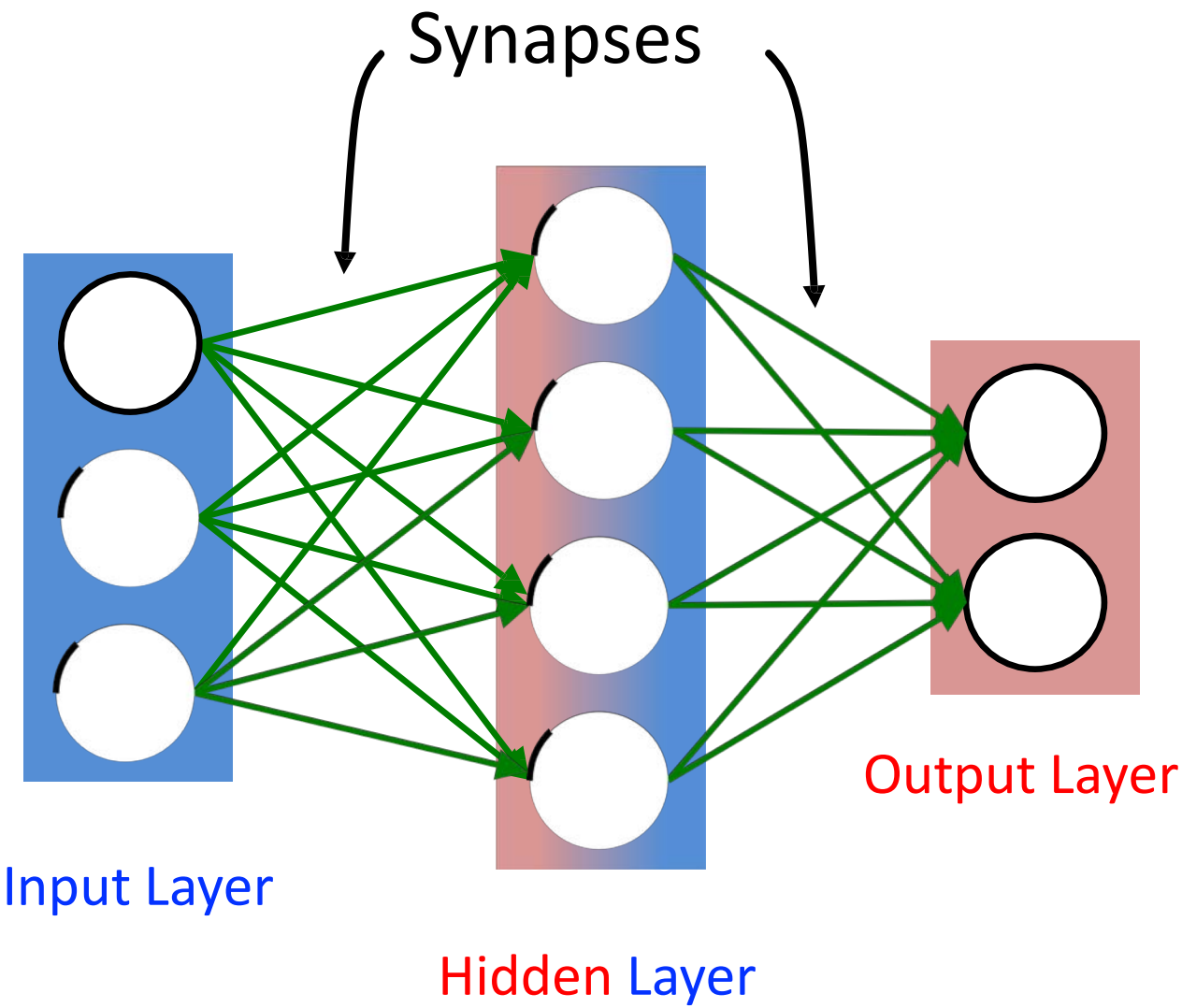

Image Source: [Lee et al., Comm. ACM 2011]

# Overview of Deep Neural Networks

# DNN Terminology 101



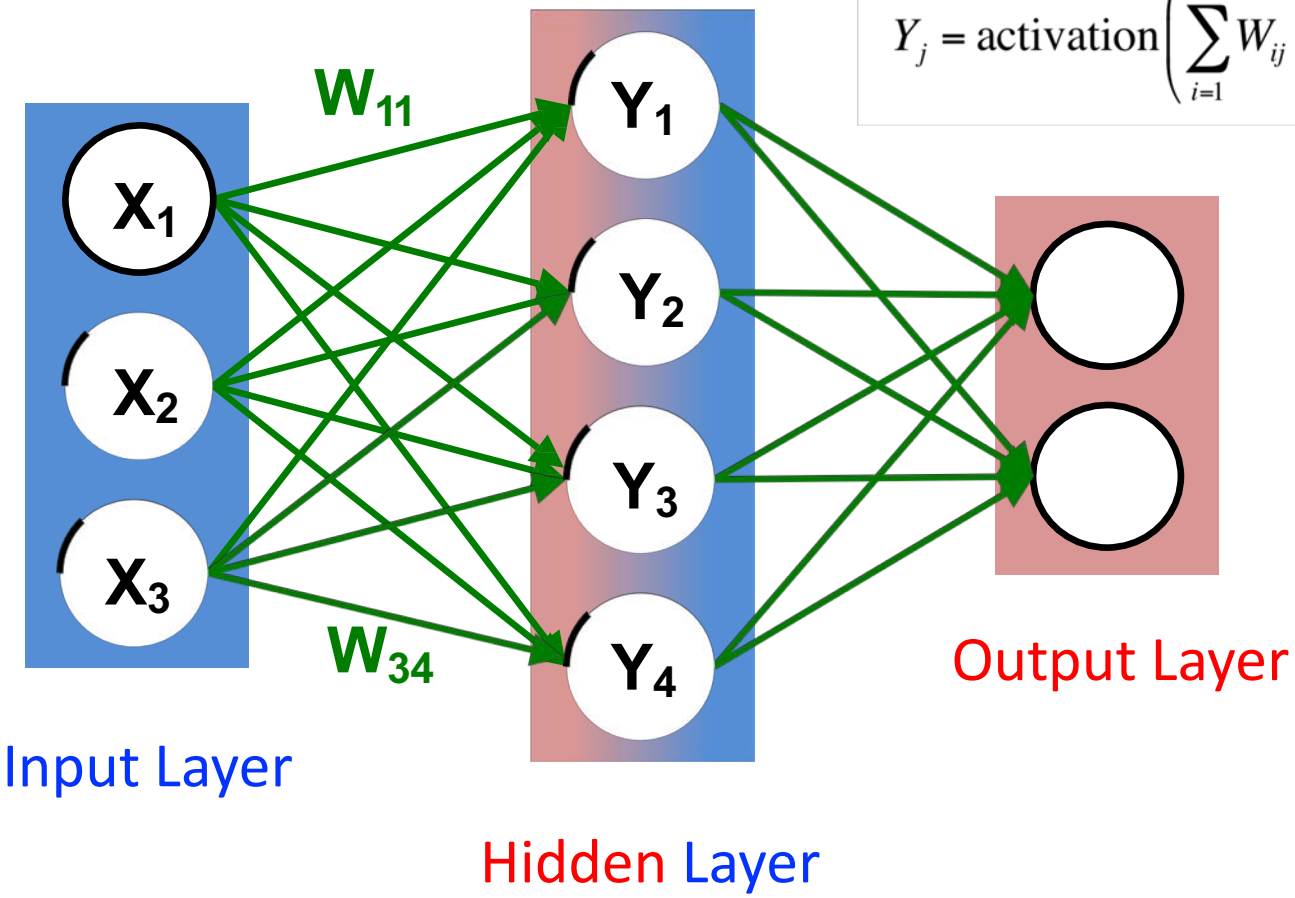
# DNN Terminology 101



# DNN Terminology 101

Each synapse has a **weight** for neuron **activation**

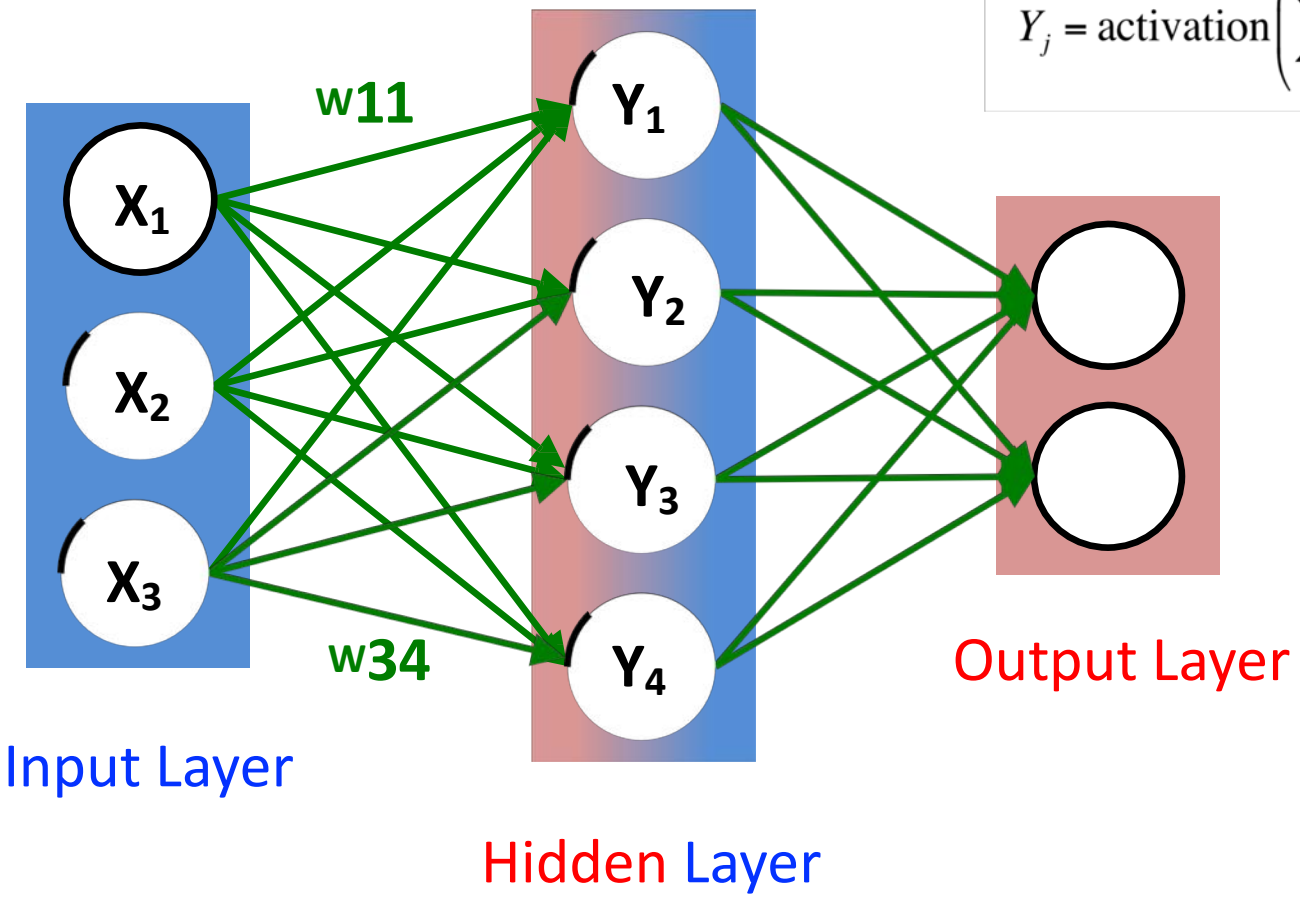
$$Y_j = \text{activation}\left(\sum_{i=1}^{3} W_{ij} \times X_i\right)$$



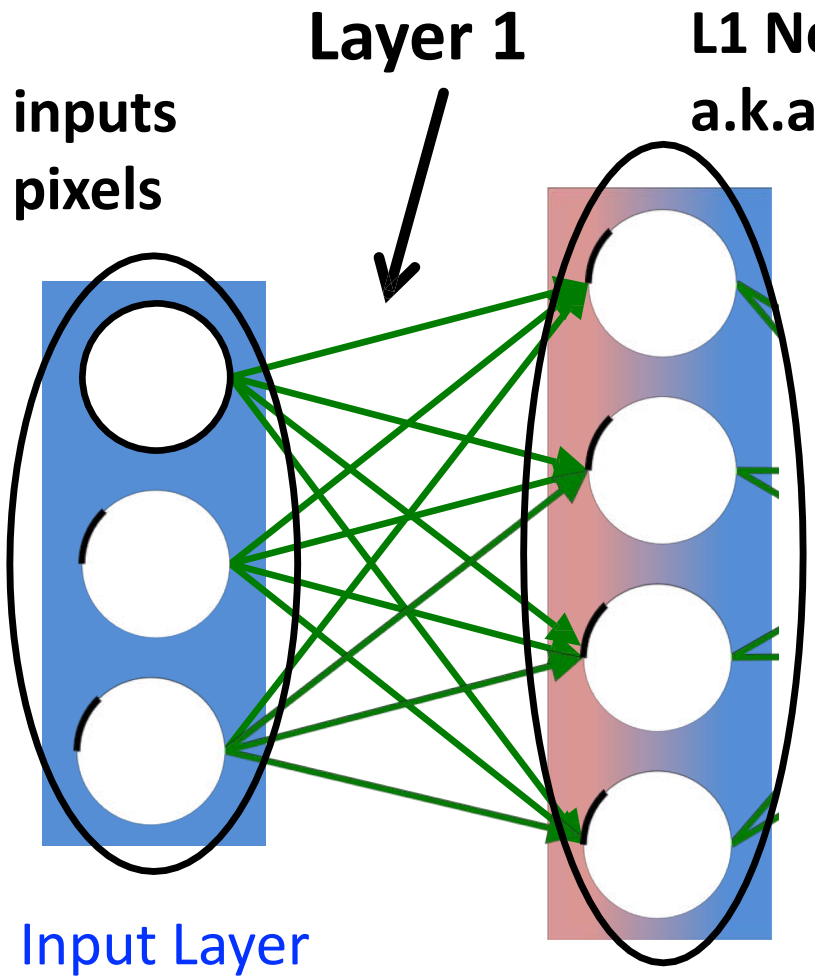
# DNN Terminology 101

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

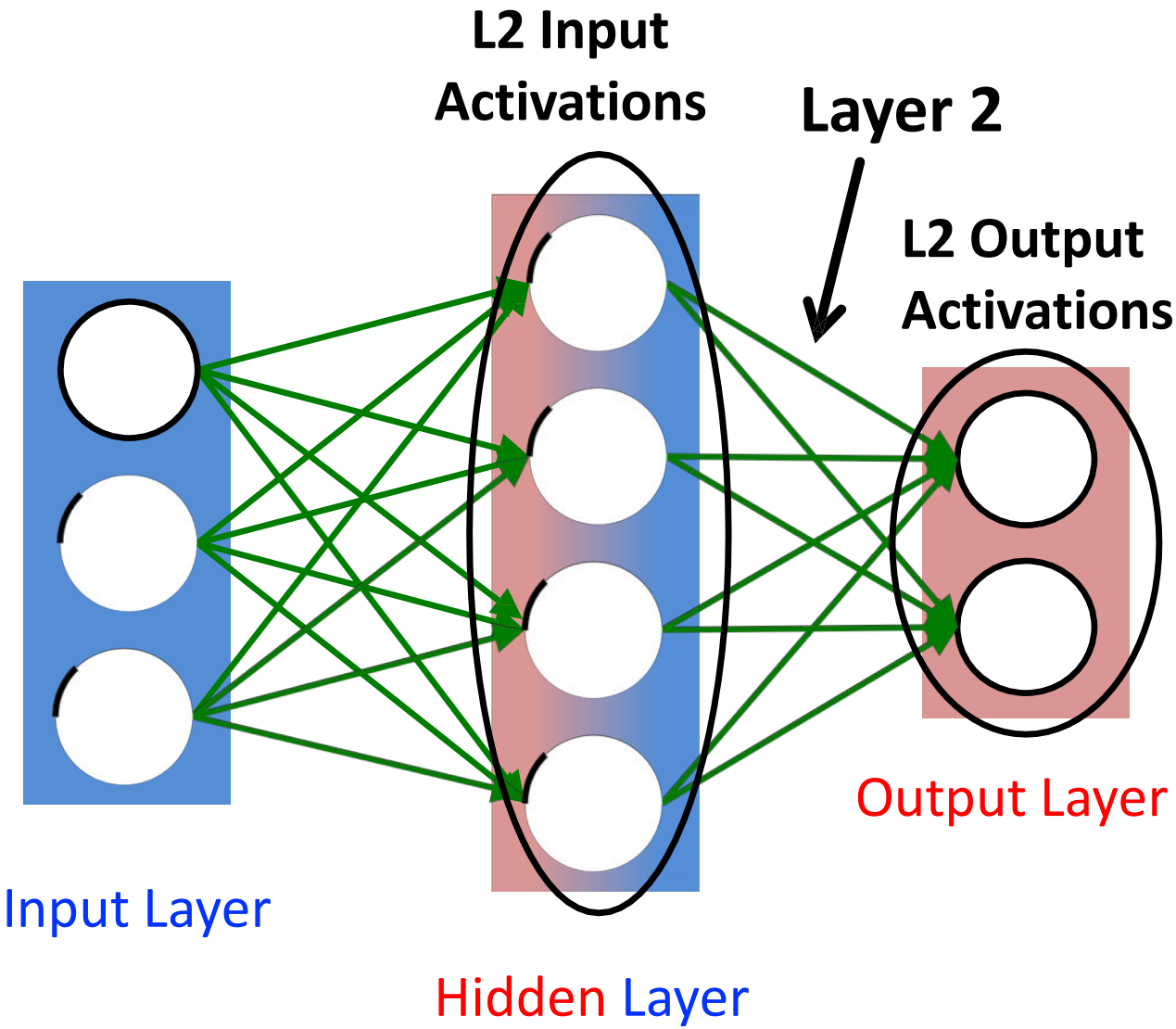
$$Y_j = \text{activation}\left(\sum_{i=1}^{3} W_{ij} \times X_i\right)$$



# DNN Terminology 101

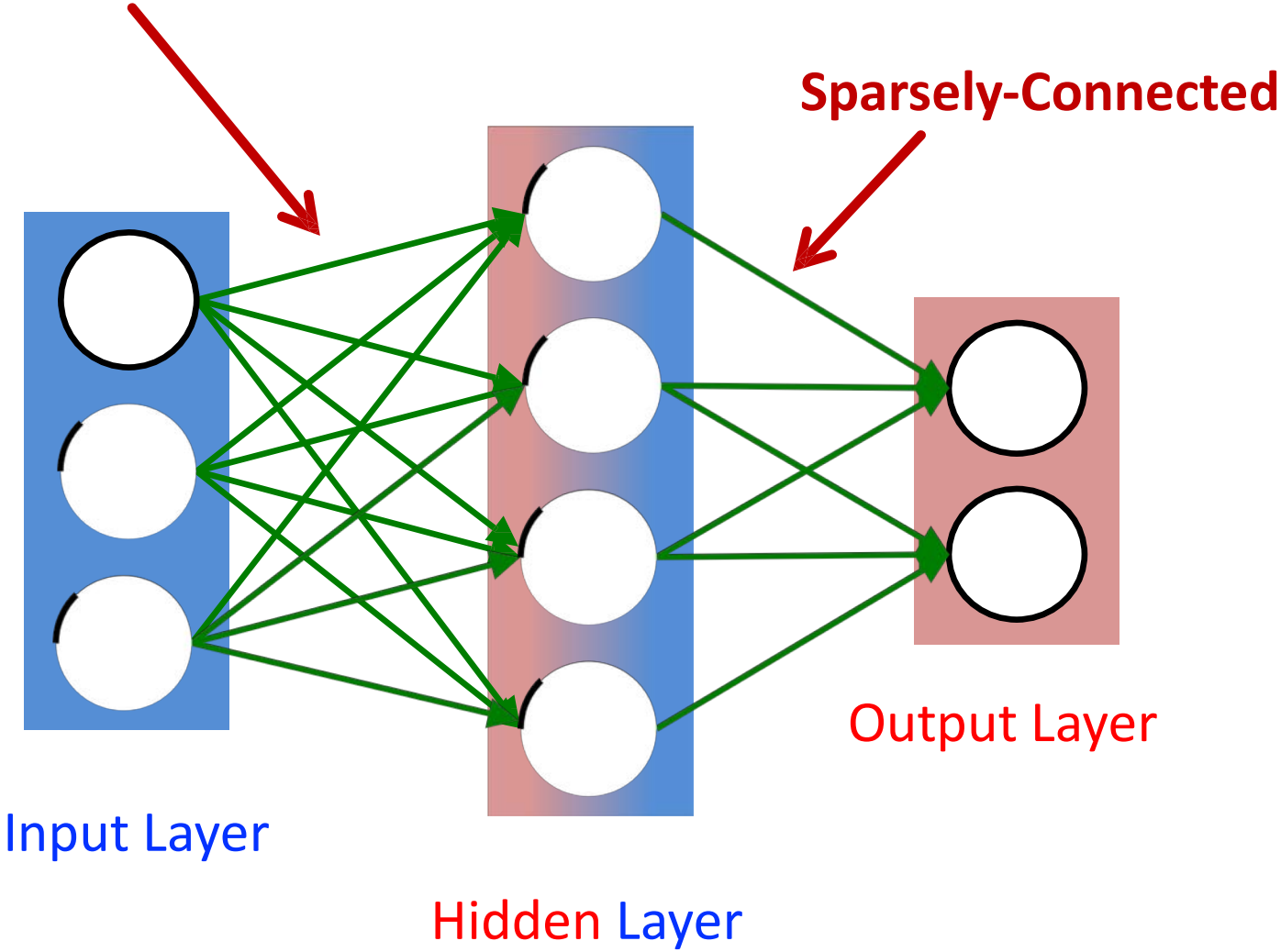


# DNN Terminology 101

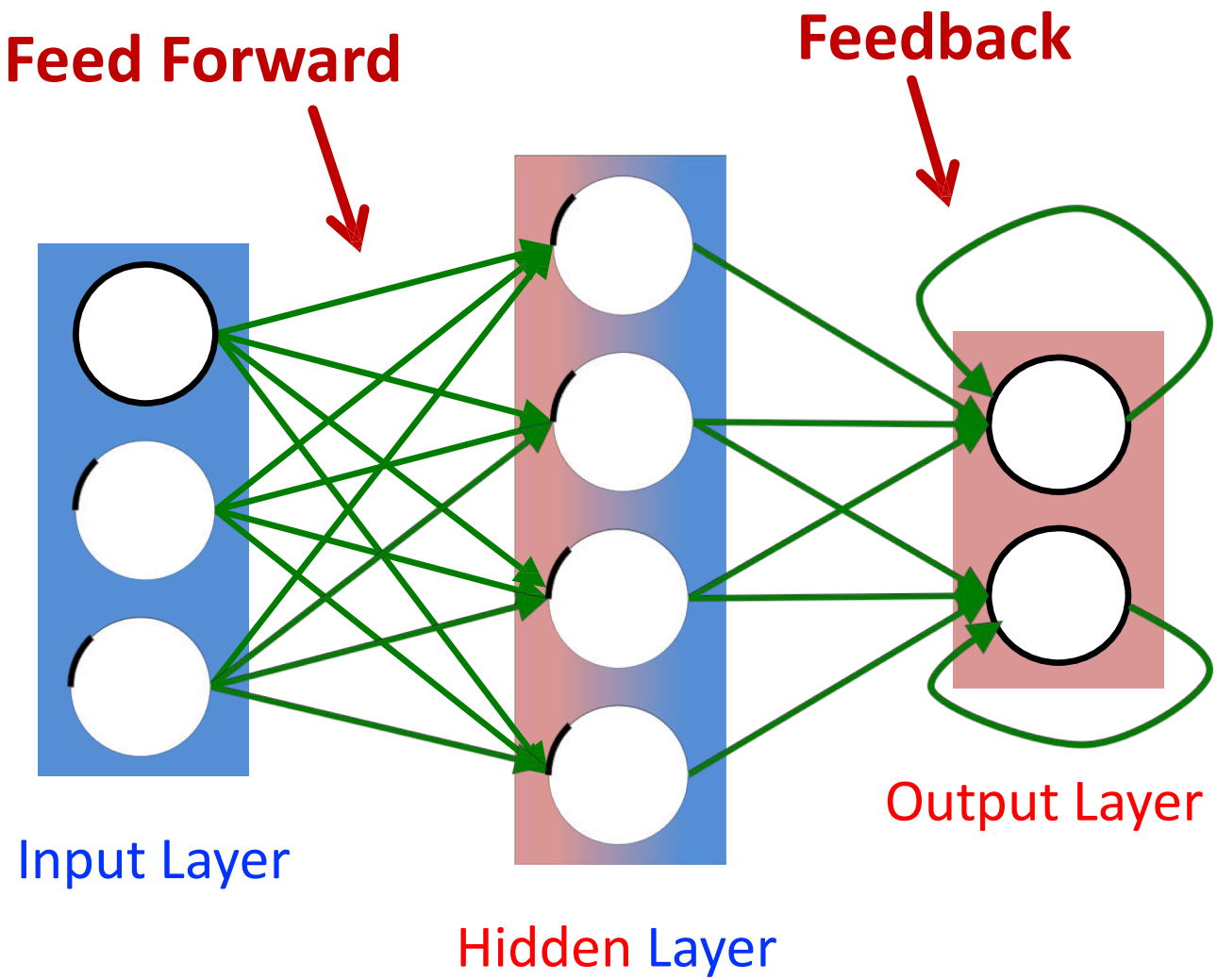


# DNN Terminology 101

**Fully-Connected**: all i/p neurons connected to all o/p neurons



# DNN Terminology 101



# Popular Types of DNNs

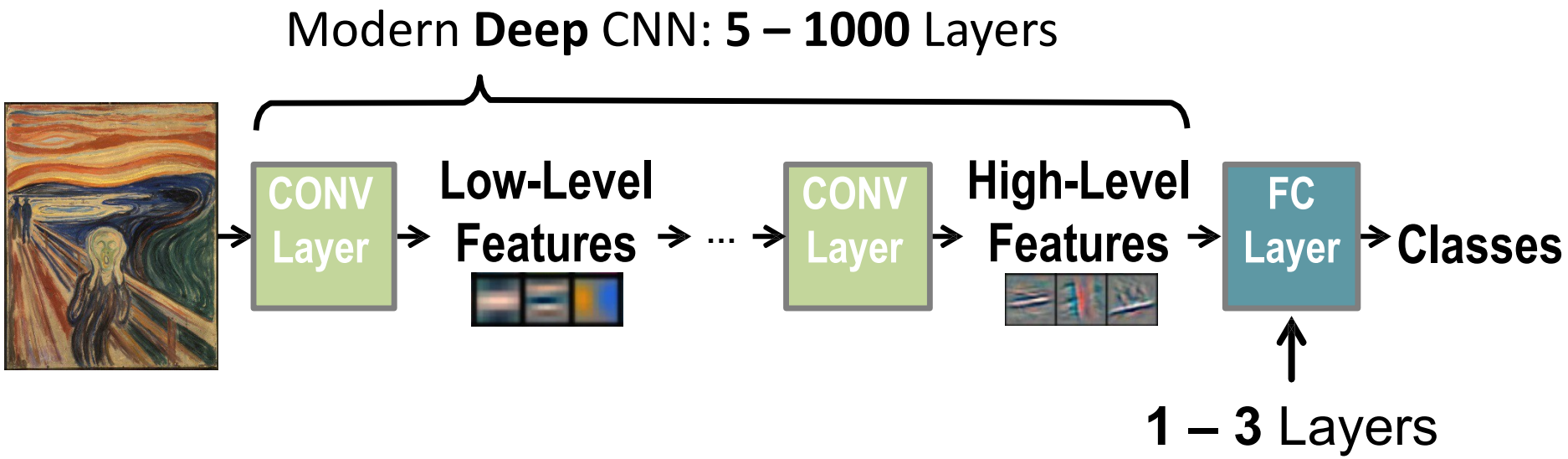
- **Fully-Connected NN**
  - feed forward, a.k.a. multilayer perceptron (MLP)
- **Convolutional NN (CNN)**
  - feed forward, sparsely-connected w/ weight sharing
- **Recurrent NN (RNN)**
  - feedback
- **Long Short-Term Memory (LSTM)**
  - feedback + storage
- **Transformer**
  - Self attention + feed forward

# Inference vs. Training

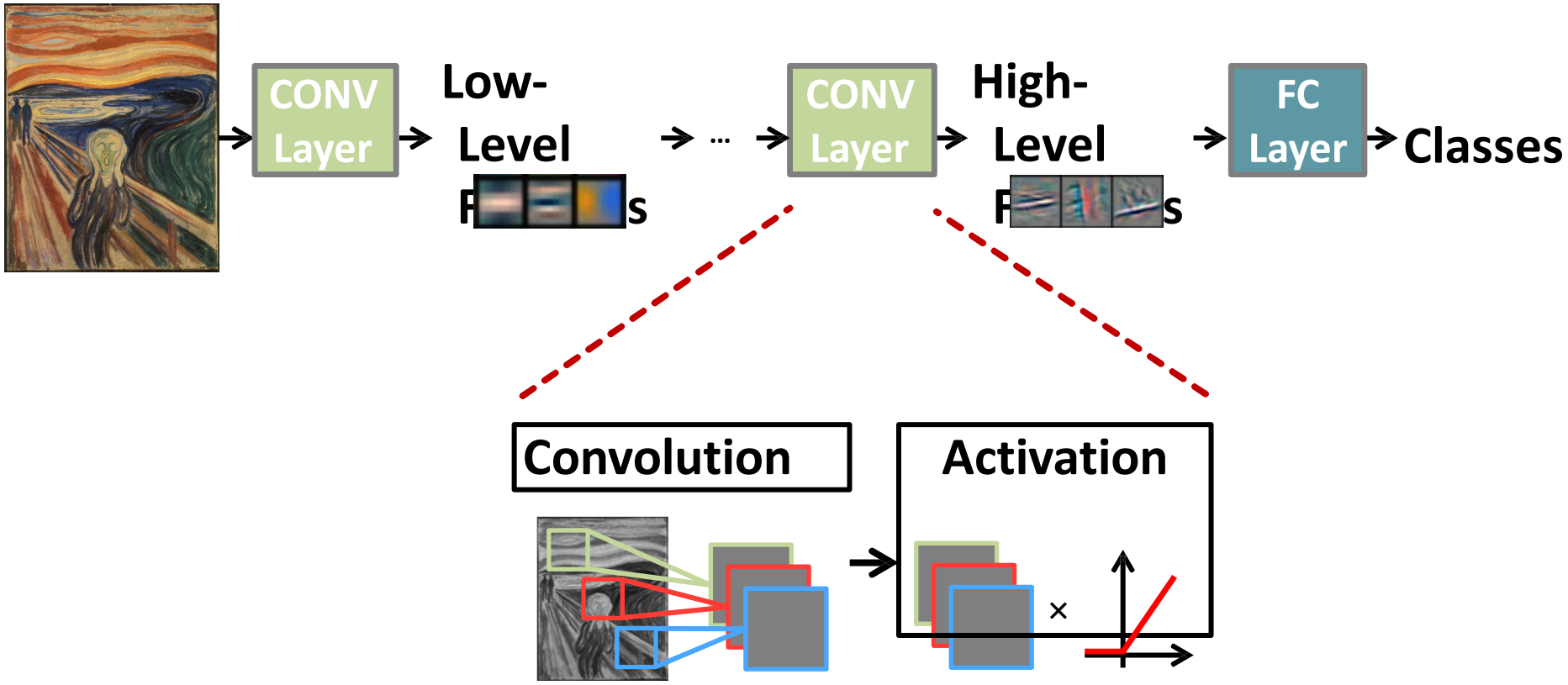
- **Training:** Determine weights
  - **Supervised:**
    - Training set has inputs and outputs, i.e., labeled
  - **Unsupervised / Self-Supervised:**
    - Training set is unlabeled
  - **Semi-supervised:**
    - Training set is partially labeled
  - **Reinforcement:**
    - Output assessed via rewards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

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

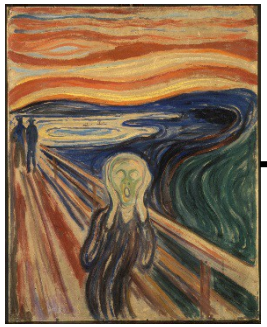
# Deep Convolutional Neural Networks



# Deep Convolutional Neural Networks



# Deep Convolutional Neural Networks



CONV Layer

Low-Level Features

…

CONV Layer

High-Level Features

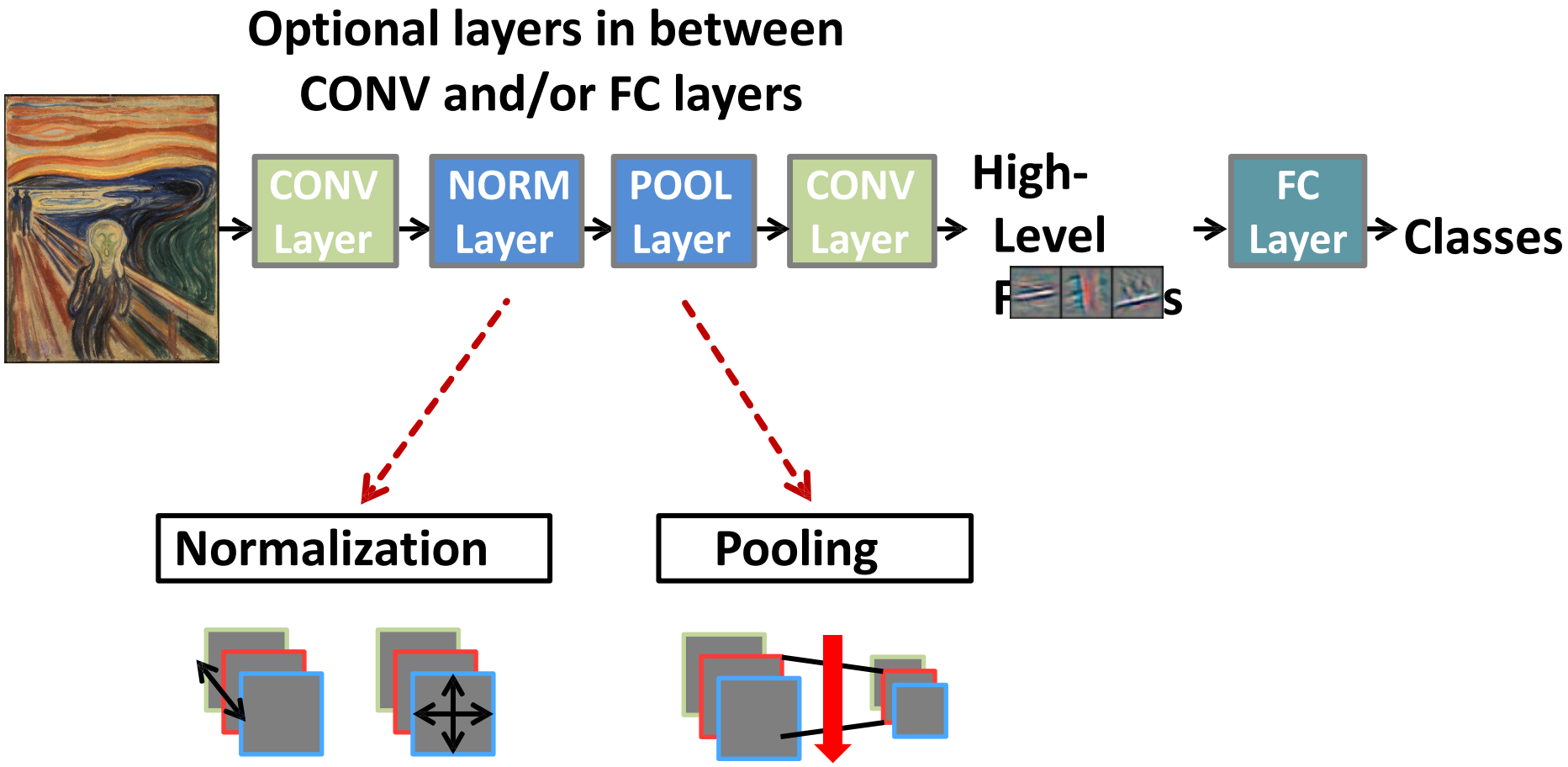
FC Layer

Classes

Fully Connected

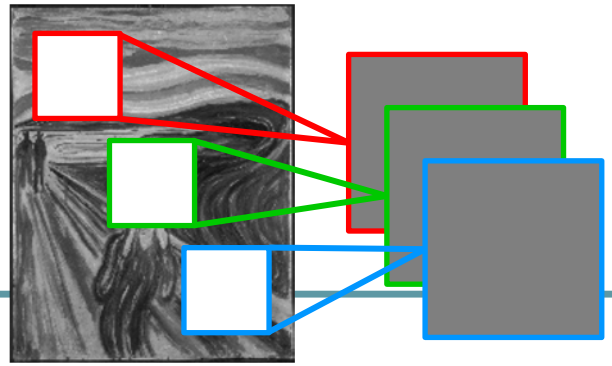
Activation

# Deep Convolutional Neural Networks



# Deep Convolutional Neural Networks

CONV Layer | NORM Layer | POOL Layer | CONV Layer | FC Layer

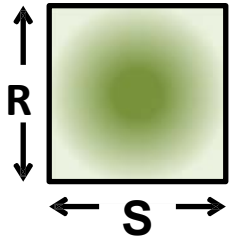


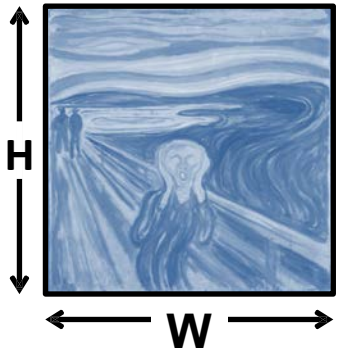
**Convolutions** account for more than 90% of overall computation, dominating **runtime** and **energy consumption**

# Convolution (CONV) Layer

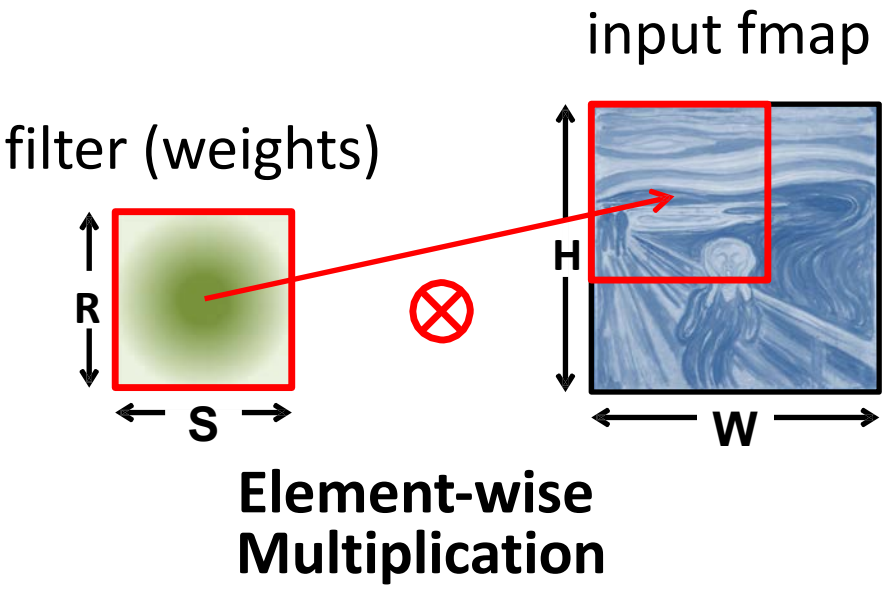
a plane of input activations
a.k.a. **input feature map (fmap)**

filter (weights)

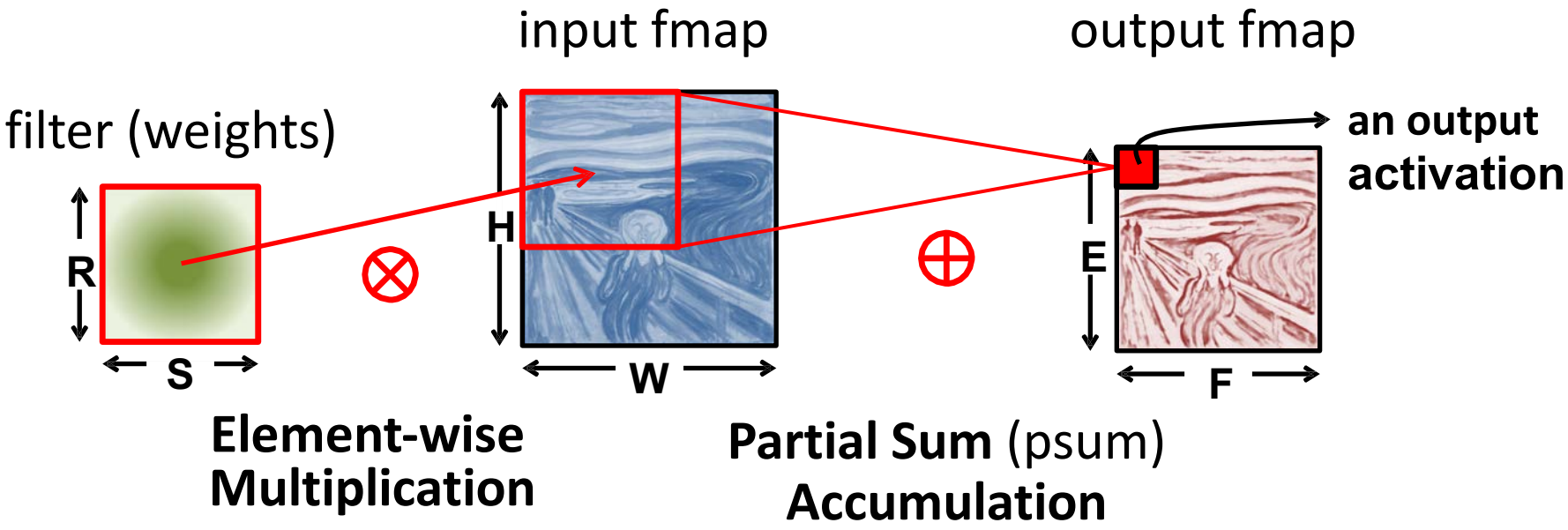


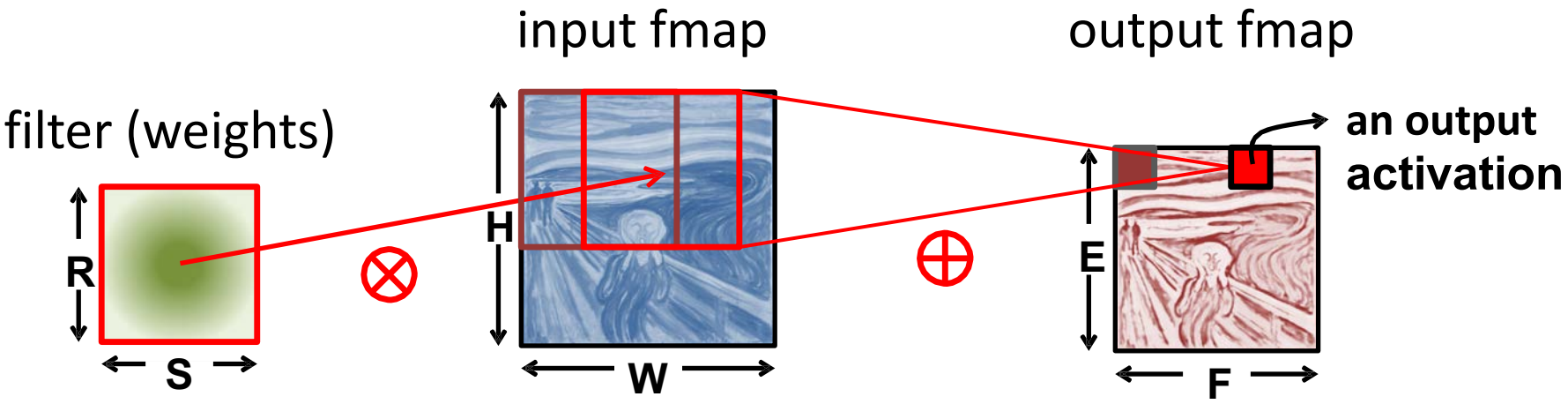
# Convolution (CONV) Layer



# Convolution (CONV) Layer

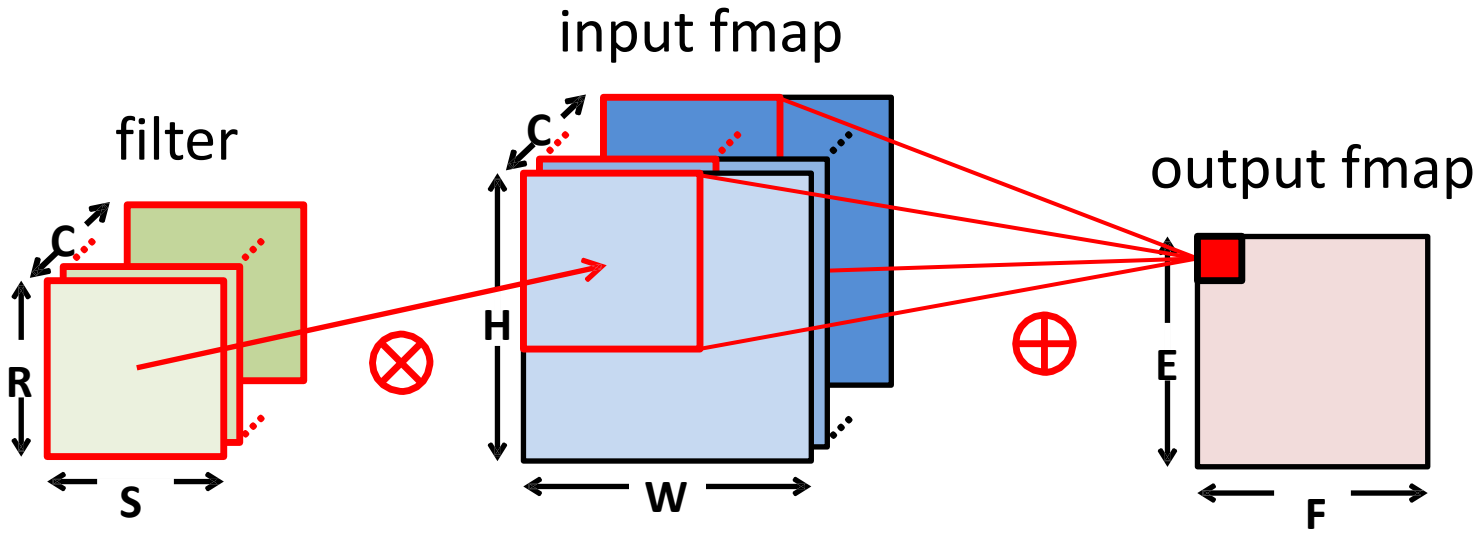


# Convolution (CONV) Layer



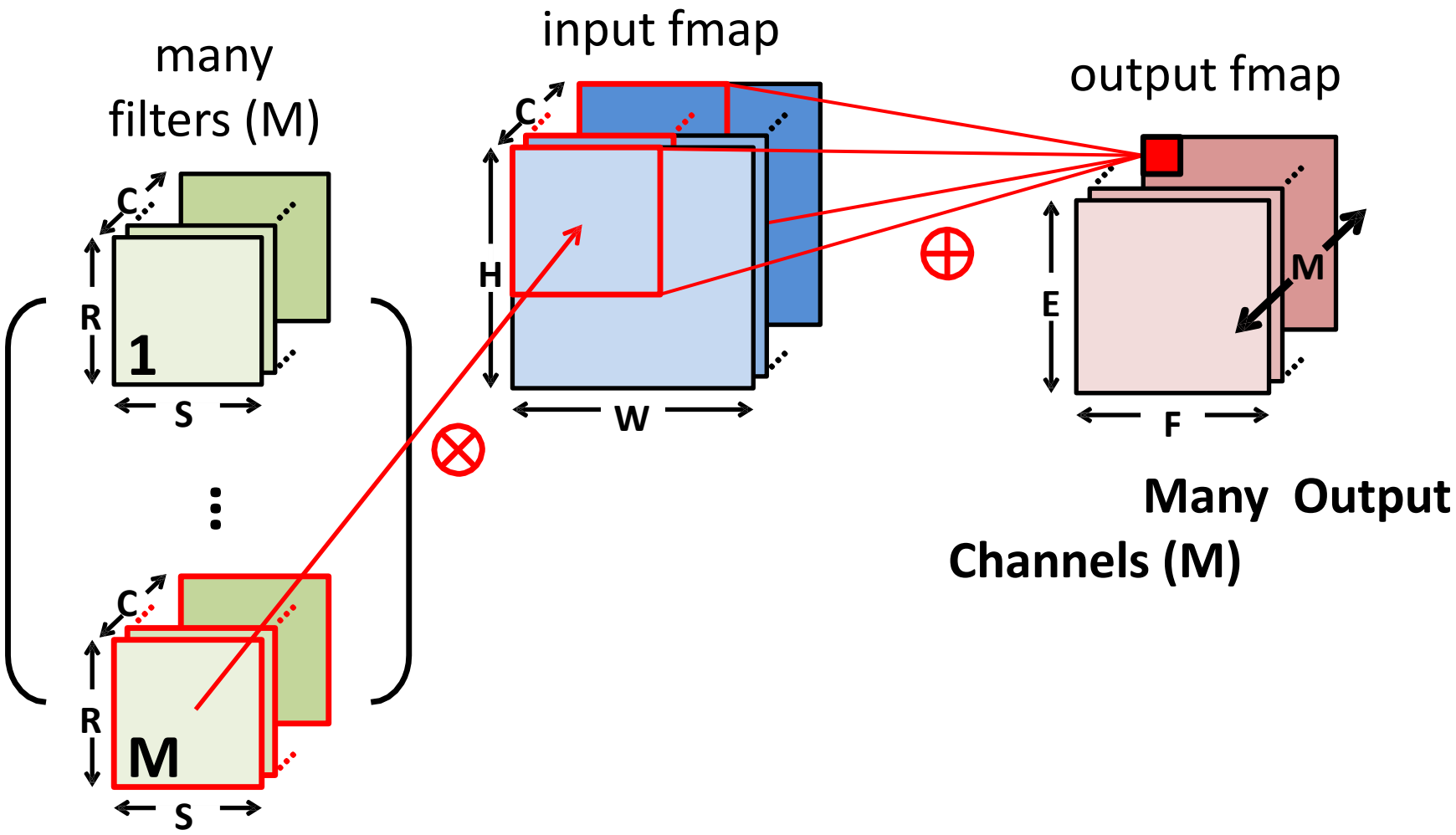

**Sliding Window Processing**

# Convolution (CONV) Layer

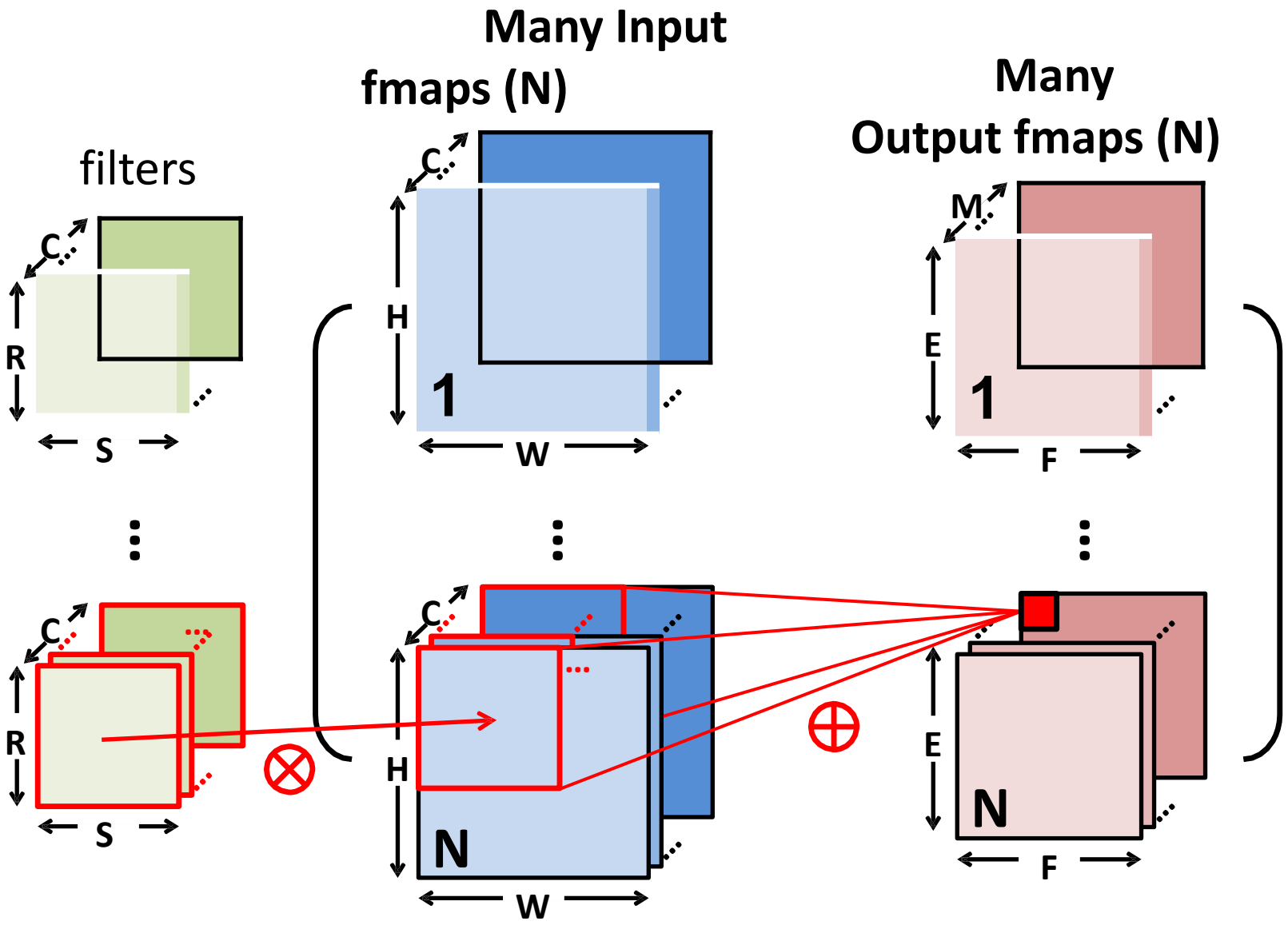



**Many Input Channels (C)**

# 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)

Input fmaps (I)

Biases (B)

Filter weights (W)

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

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

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

| 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:

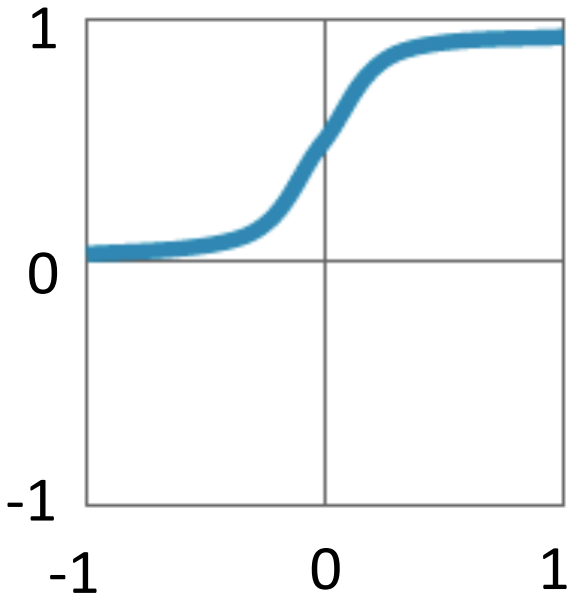
```
for (n=0; n<N; n++) {  for (m=0;
        m<M; m++) {
              for (x=0; x<F; x++) {  for (y=0; y<E;
                    y++) {
                          O[n][m][x][y] = B[m];
                          for (i=0; i<R; i++) {  for (j=0; j<S;
                                j++) {
                                      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]);
                    }
              }
        }
}
```

for each output fmap value

convolve a window and apply activation

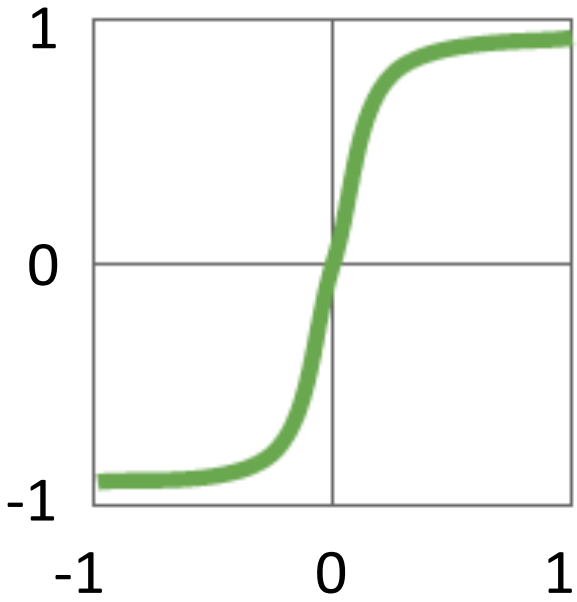
# Traditional Activation Functions

## Sigmoid




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

## Hyperbolic Tangent



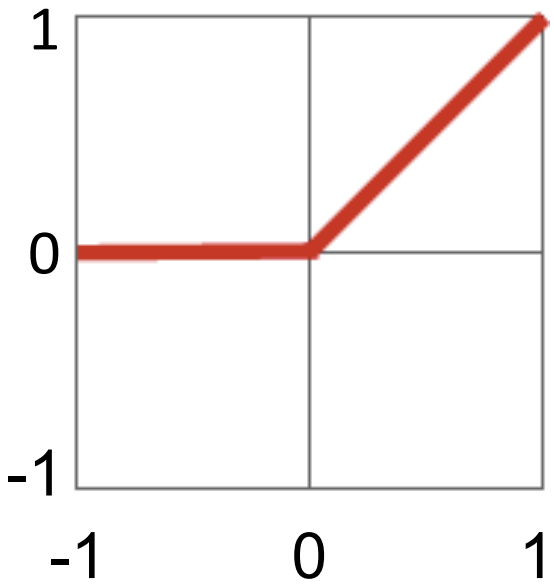

$$y=(e^{x}-e^{-x})/(e^{x}+e^{-x})$$



Image Source: Caffe Tutorial

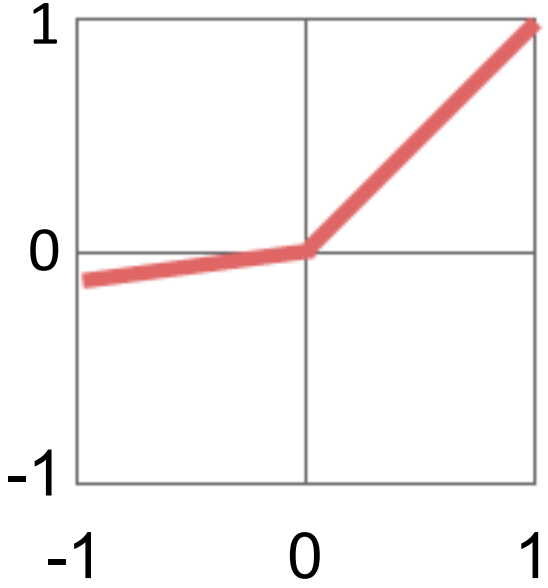
# Modern Activation Functions

**Rectified Linear Unit (ReLU)**




$y=\max(0,x)$

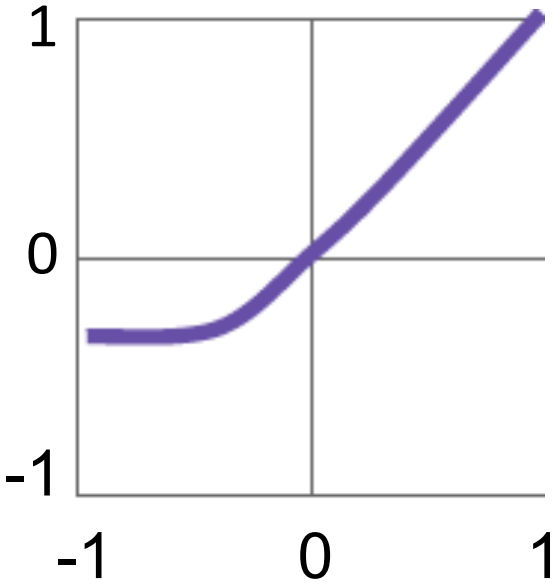
**Leaky ReLU**




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

$\alpha$ = small const. (e.g. 0.1)

**Exponential LU**

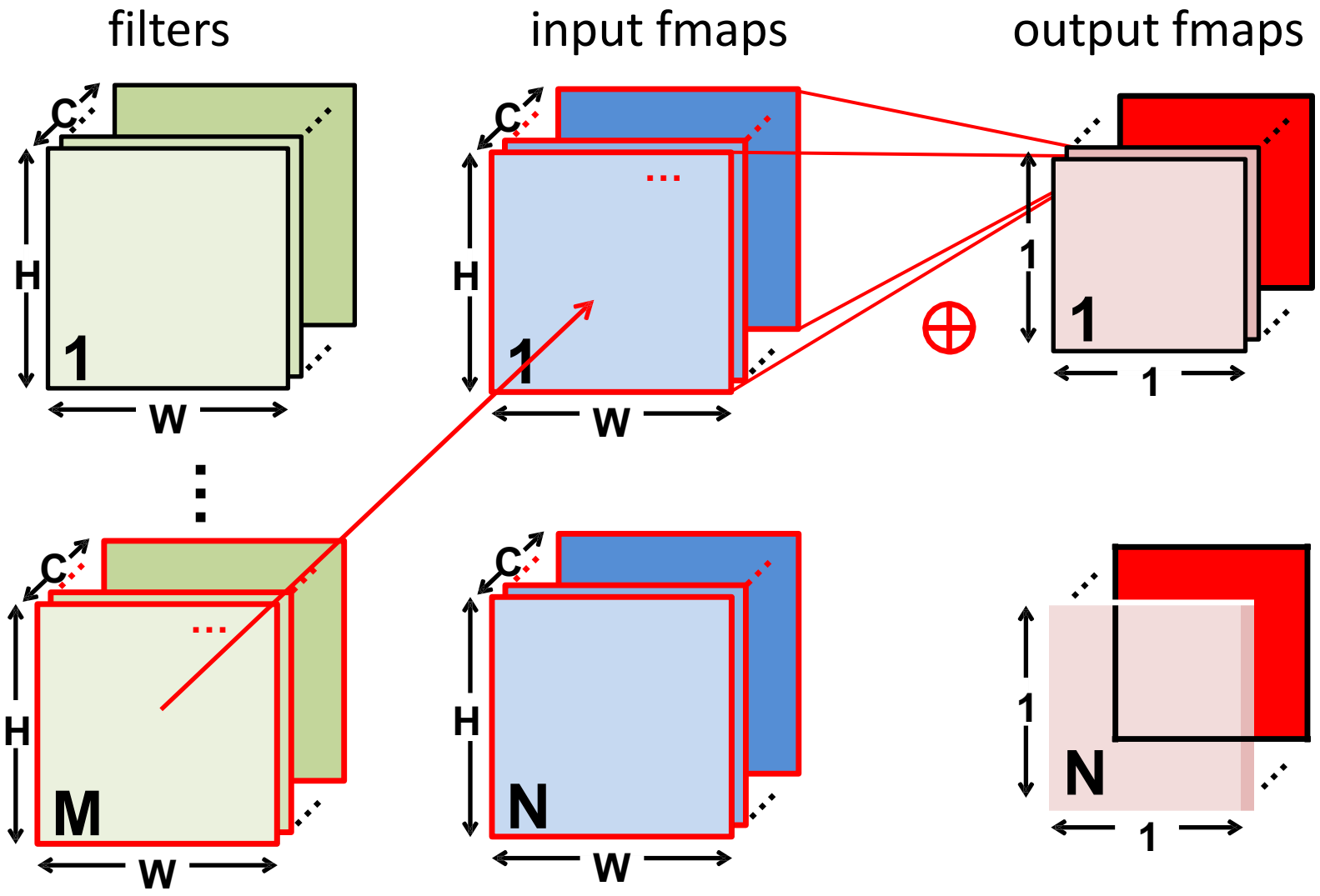



$$y=\begin{cases} x, & x\geq 0 \\ \alpha(e^x-1), & x<0 \end{cases}$$



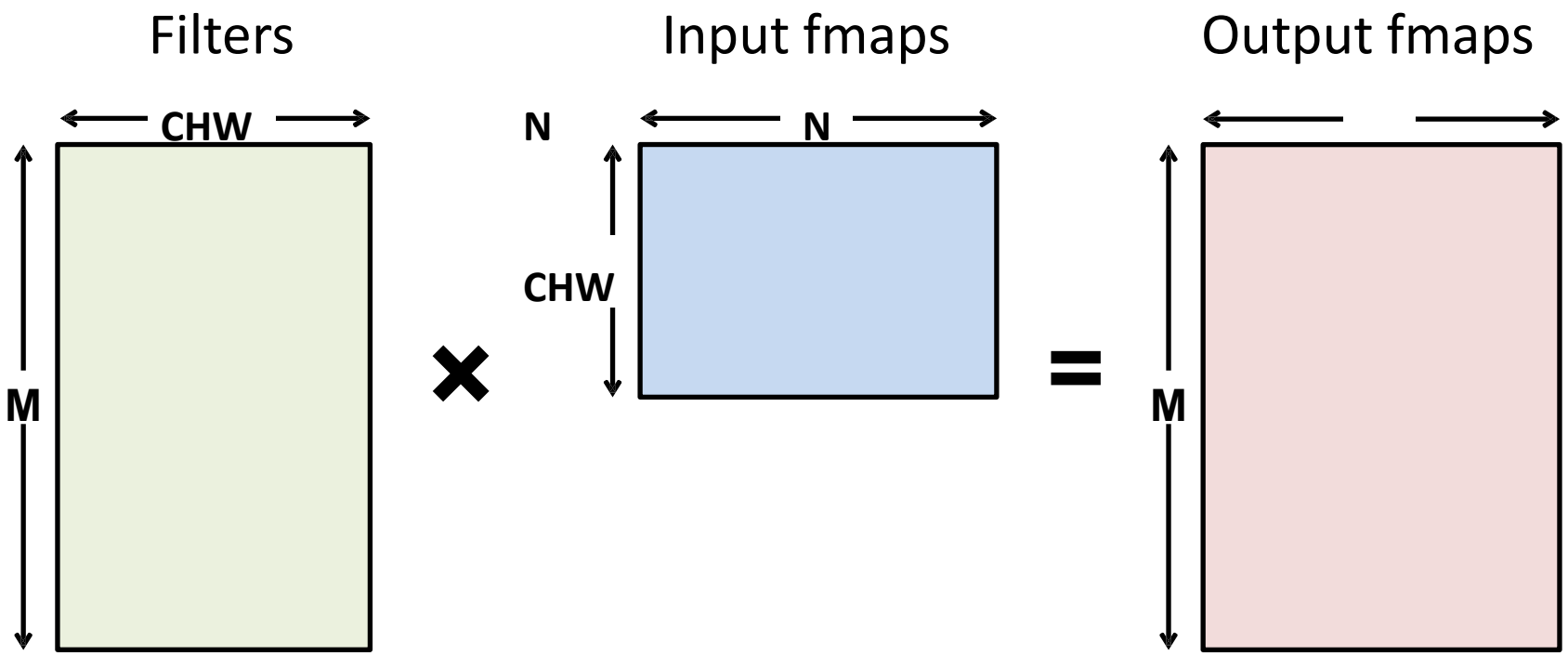
Image Source: Caffe Tutorial

# FC Layer – from CONV Layer POV



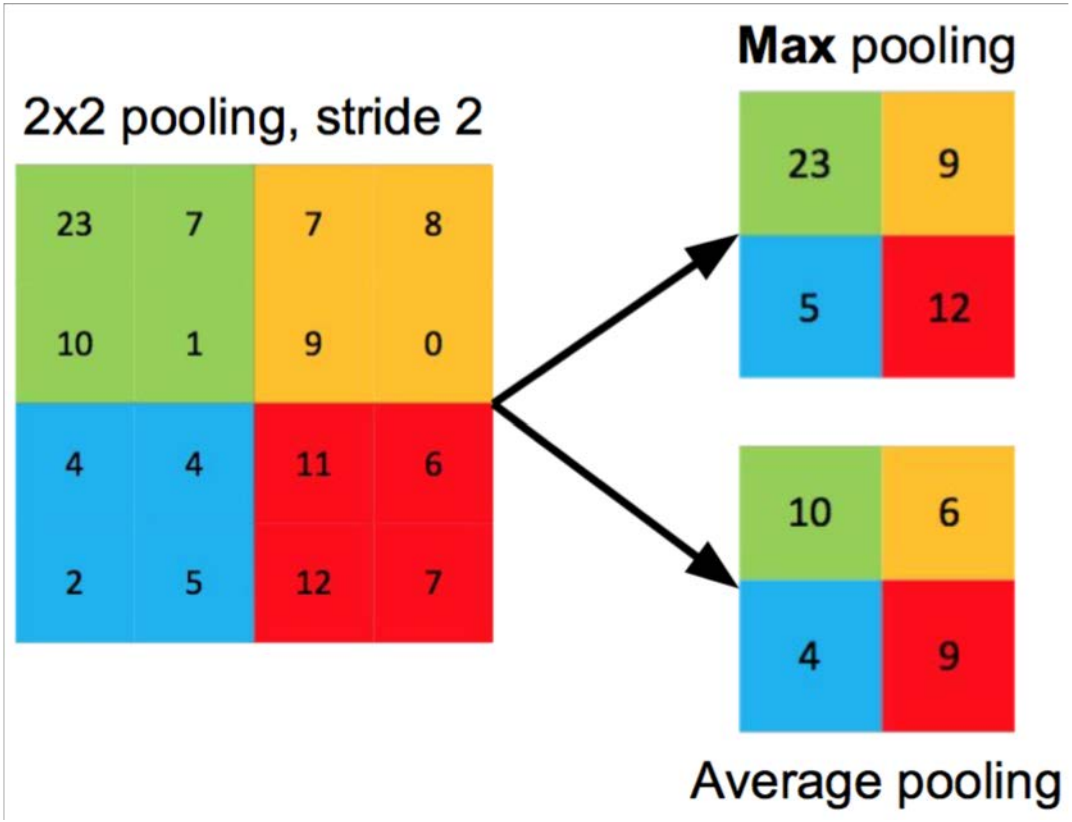
# Fully-Connected (FC) Layer

- 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 à depending on stride




Increases translation-invariance and noise-resilience



Image Source: Caffe Tutorial

# POOL Layer Implementation

## Naïve 6-layer for-loop max-pooling implementation:

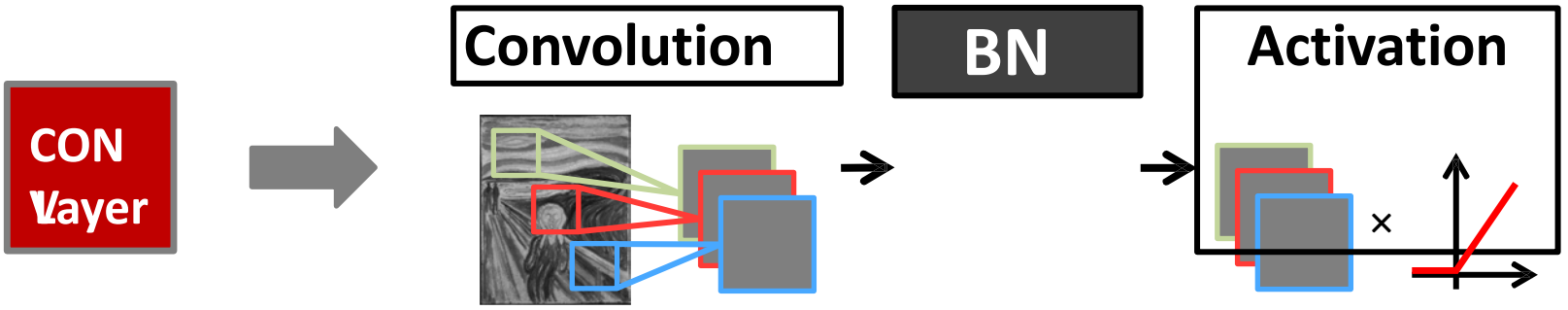
```
for (n=0; n<N; n++) {  for (m=0;
      m<M; m++) {
            for (x=0; x<F; x++) {  for (y=0; y<E;
                  y++) {

                        max = -Inf;
                        for (i=0; i<R; i++) {  for (j=0; j<S;
                              j++) {
                                    if (I[n][m][Ux+i][Uy+j] > max) {
                                          max = I[n][m][Ux+i][Uy+j];
                                    }
                              }
                        }

                        O[n][m][x][y] = max;
                  }
            }
      }
}
```

for each pooled value

find the max with in a window

# 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**

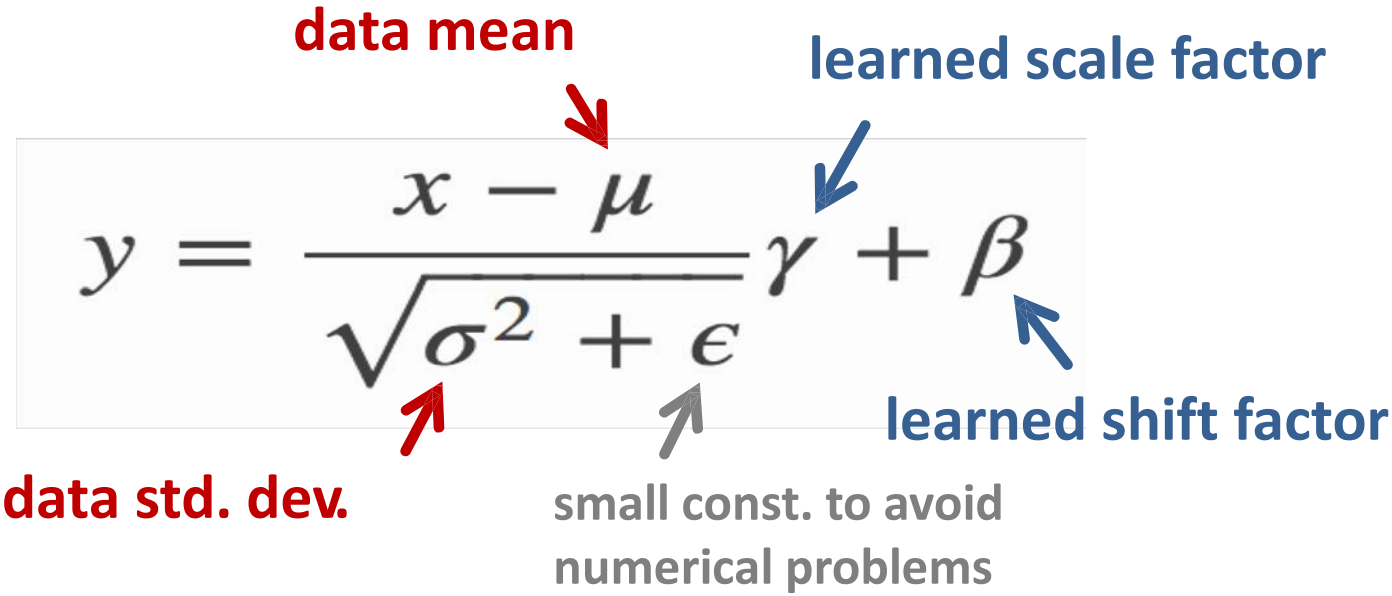



Believed to be key to getting high accuracy and faster training on very deep neural networks.

[Ioffe et al., ICML 2015]

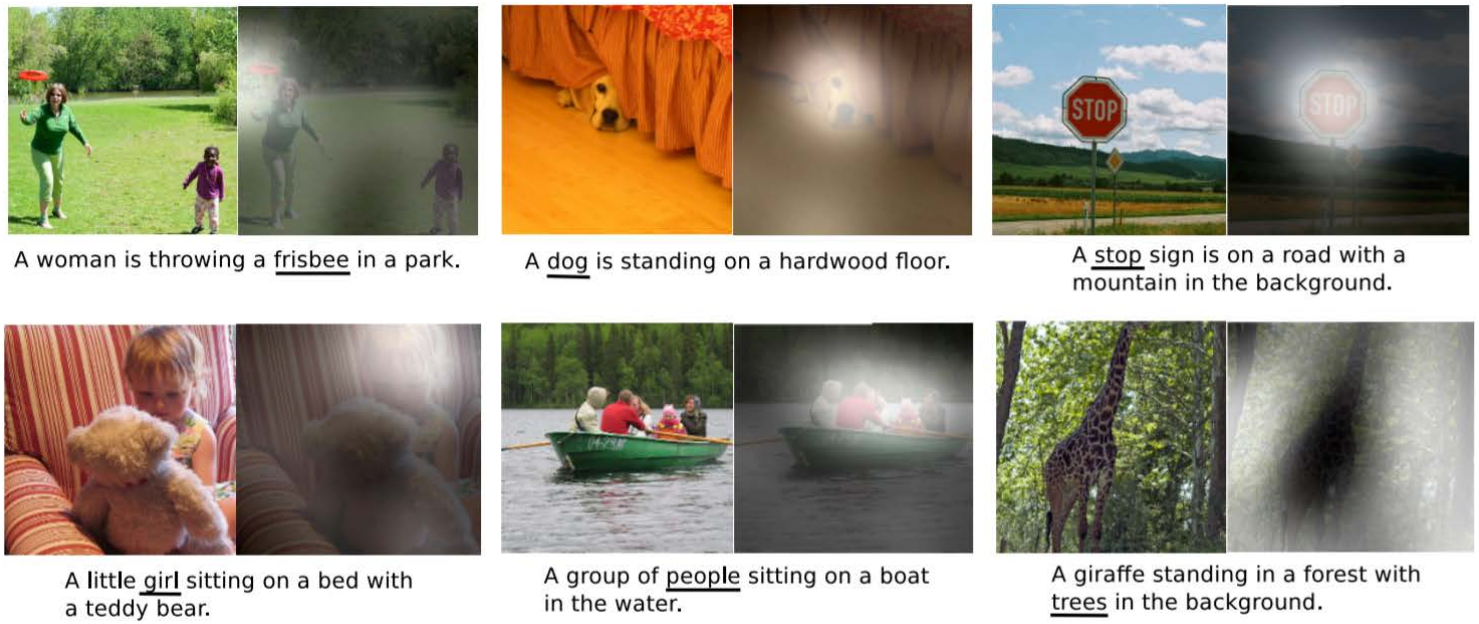
# BN Layer Implementation

- The normalized value is further scaled and shifted, the parameters of which are learned from training



# Attention Modules

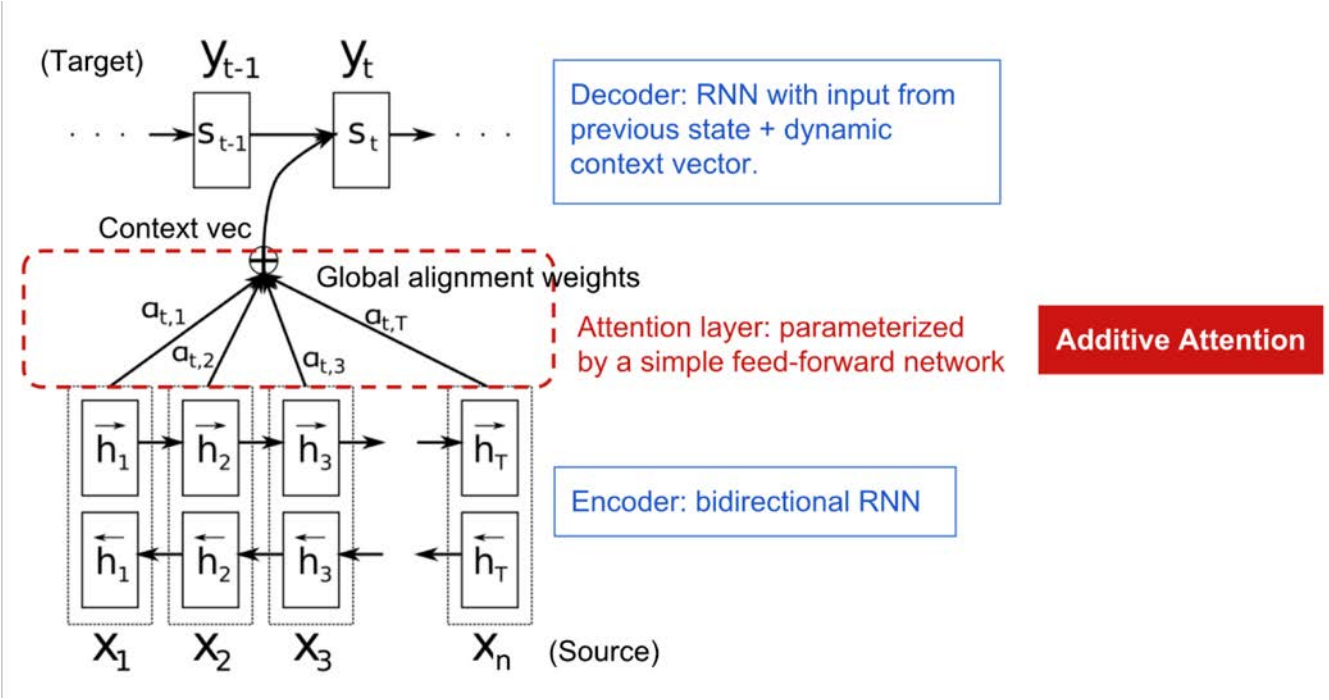
- **Attention** is, to some extent, motivated by how we pay visual attention to different regions of an image or correlate words in one sentence.




An example of vision attention in words-based image retrieval .

# Attention Modules

- **Attention** is, to some extent, motivated by how we pay visual attention to different regions of an image or correlate words in one sentence.




An example of language attention in machine translation.

# Attention Modules

- Hard Attention & Soft Attention

Soft Attention

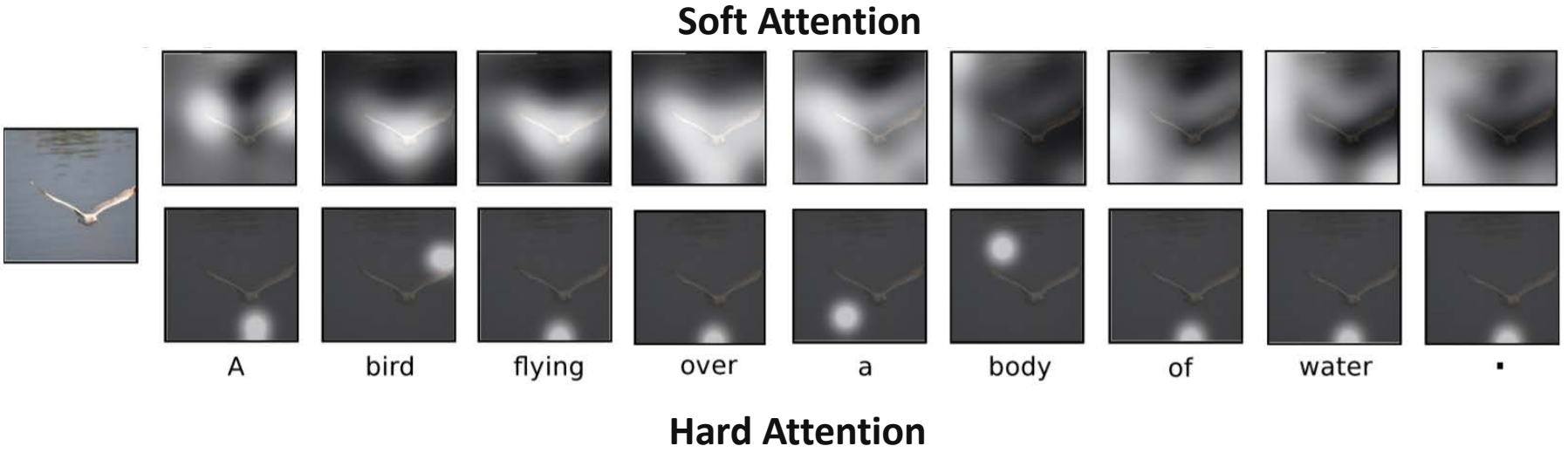
Hard Attention

- Looking through a foggy pane of glass represents soft attention, where the entire image is still being “seen”, but certain areas are being attended to more.

- The binoculars represent hard attention, where we are only seeing a subset of the image, hopefully the part most relevant to our task.

# Attention Modules

- Hard Attention & Soft Attention




- This figure compares the soft attention and hard attention in the a “vision-language” task. Soft attention shows the importance of each patch while hard attention shows the most important patch

# Summary

- Roadmap of AI
- DNN structure
- Popular Types of DNN
- Typical operations
  - Convolution (CONV)
  - Fully-Connected (FC)
  - Max Pooling
  - ReLU
  - Attention