Hardware Architectures for Deep Neural Networks

ISCA Tutorial

June 22, 2019

Website: http://eyeriss.mit.edu/tutorial.html

Massachusetts Institute of Technology

NVIDIA®

Speakers and Contributors

Joel Emer

Senior Distinguished Research Scientist

NVIDIA

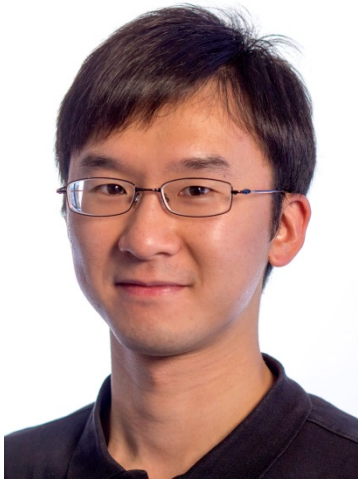
Professor

MIT

Vivienne Sze

Professor

MIT



Yu-Hsin Chen

PhD Graduate

MIT

Research Scientist

NVIDIA

Tien-Ju Yang

PhD Candidate

MIT

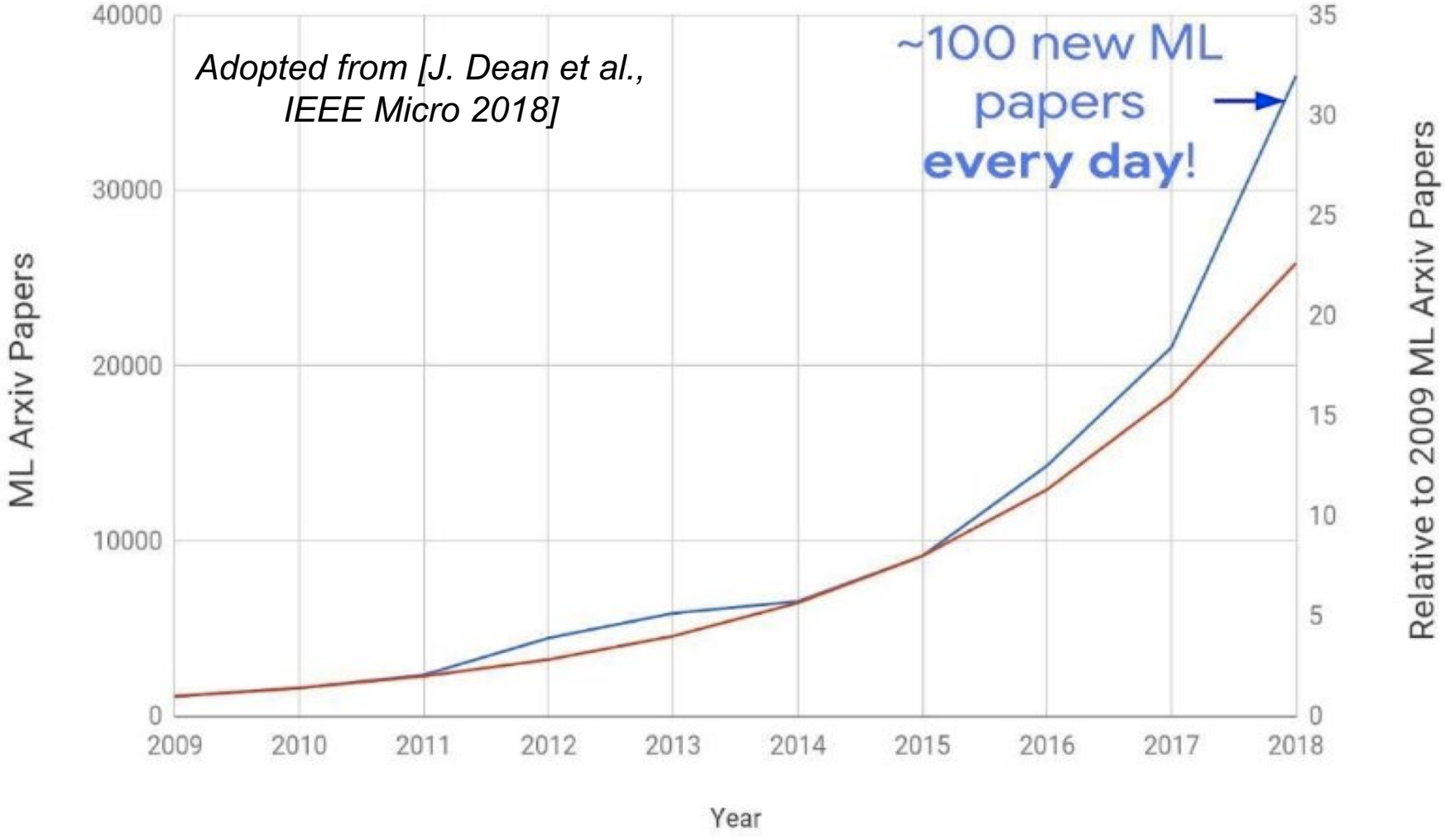
Outline (9AM-5PM)

- Overview of Deep Neural Networks
- DNN Kernel Computation
- DNN Accelerators
- DNN Model and Hardware Co-Design
- DNN Processing In/Near Memory
- Sparse DNN Accelerators

Participant Takeaways

- 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

Growth of Machine Learning Workloads

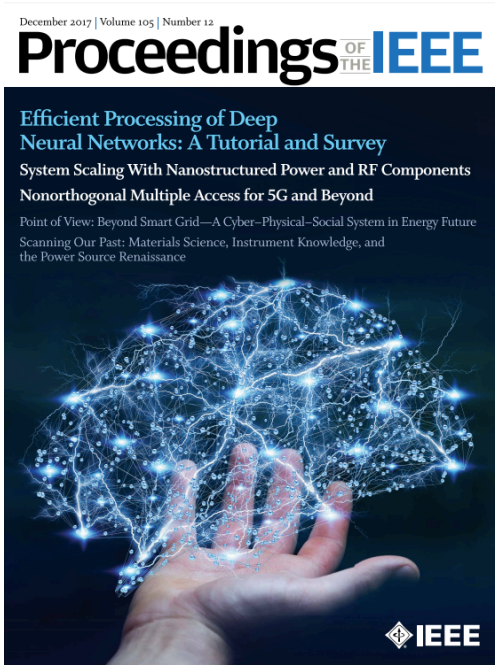


Tutorial aims to cover **key concepts, provide insights and highlight trends** rather than be a comprehensive survey all work

Resources

- **Eyeriss Project: http://eyeriss.mit.edu**
  - Tutorial Slides
  - Energy modeling

V. Sze, Y.-H. Chen, T-J. Yang, J. Emer,
"***Efficient Processing of Deep Neural Networks: A Tutorial and Survey***,"
Proceedings of the IEEE, Dec. 2017

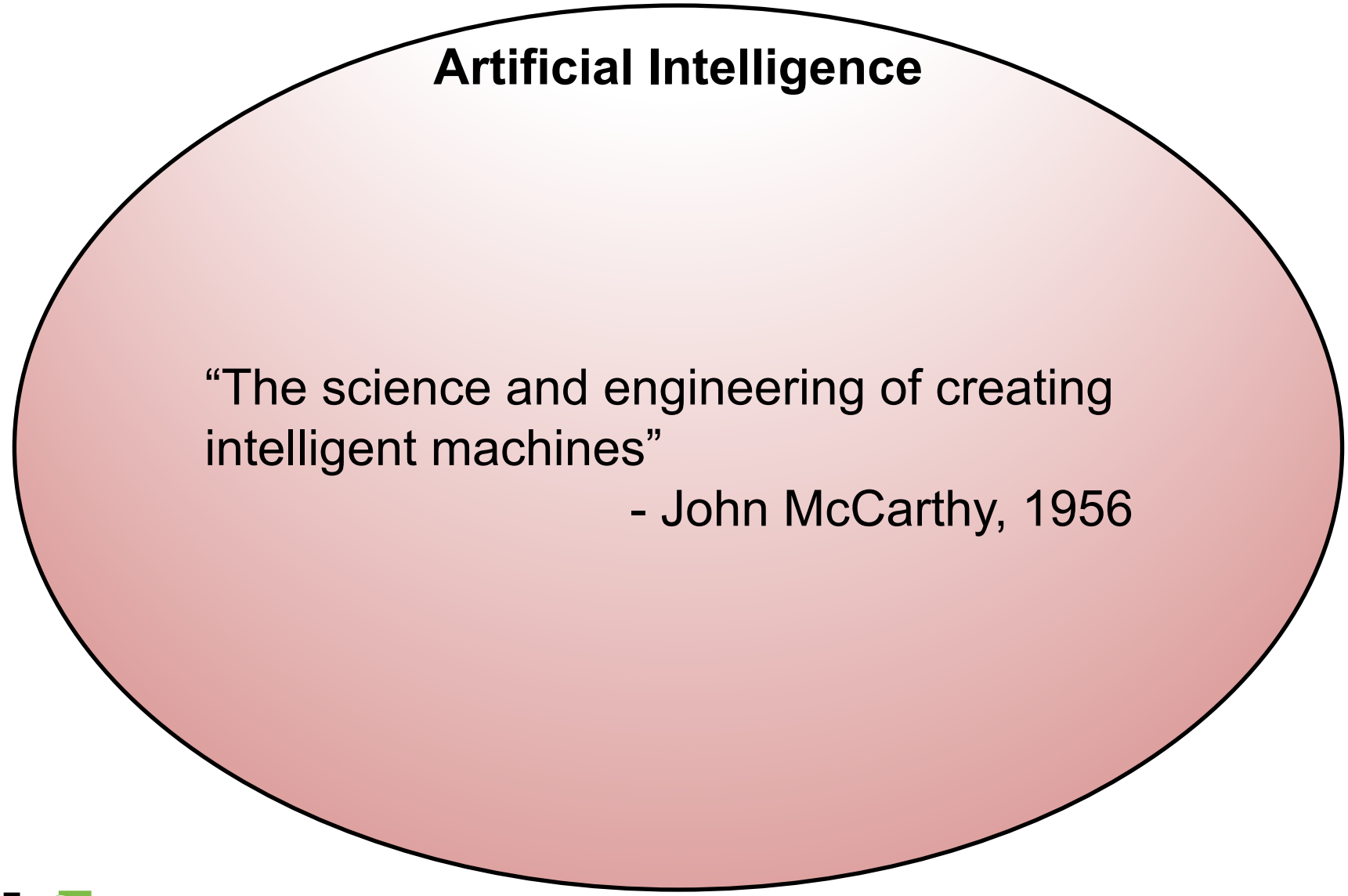


Synthesis Lecture Book coming soon! (Estimate end of summer)

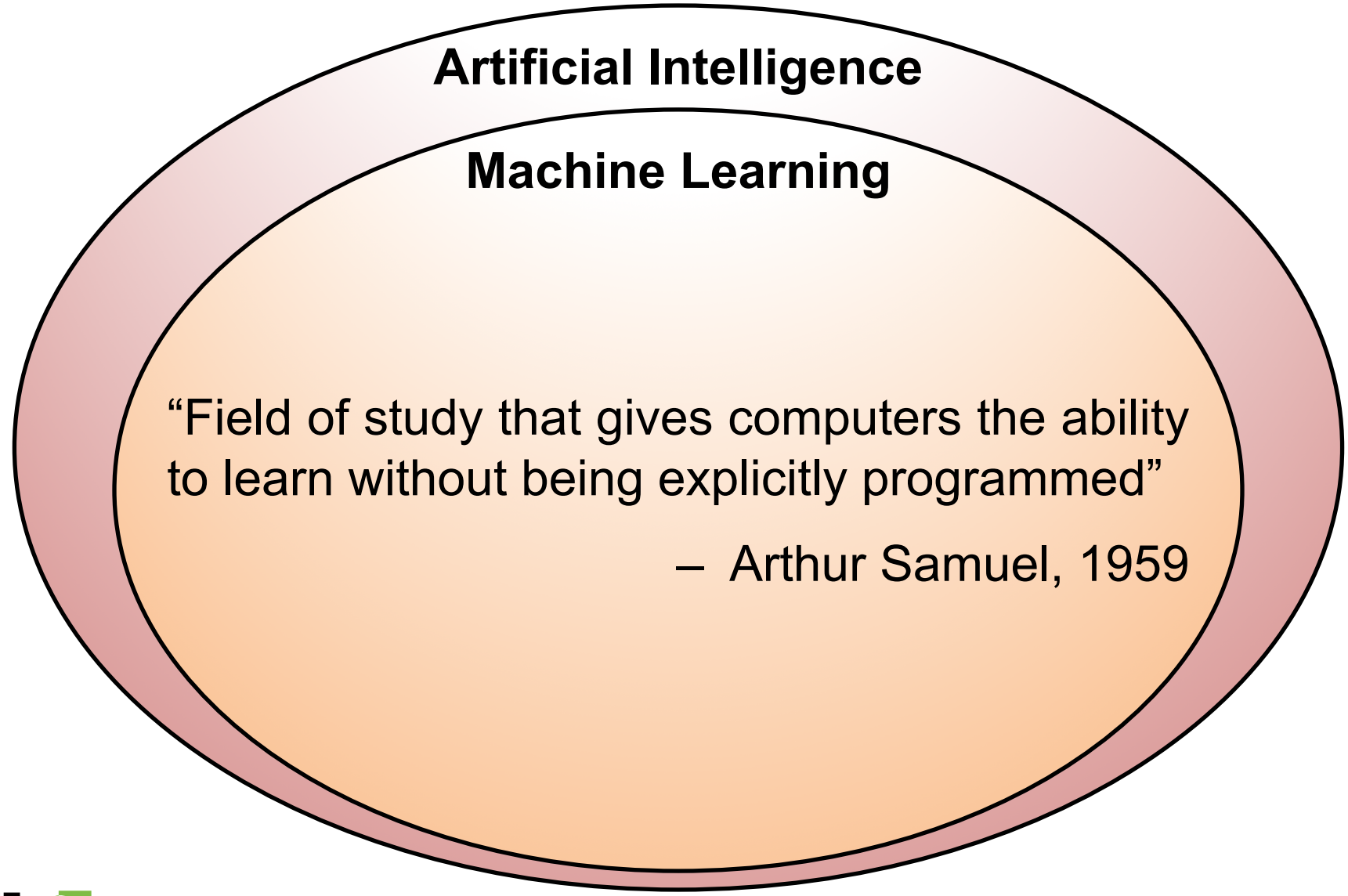
  - Mailing List for updates Follow @eems_mit
    - http://mailman.mit.edu/mailman/listinfo/eems-news

Background of Deep Neural Networks

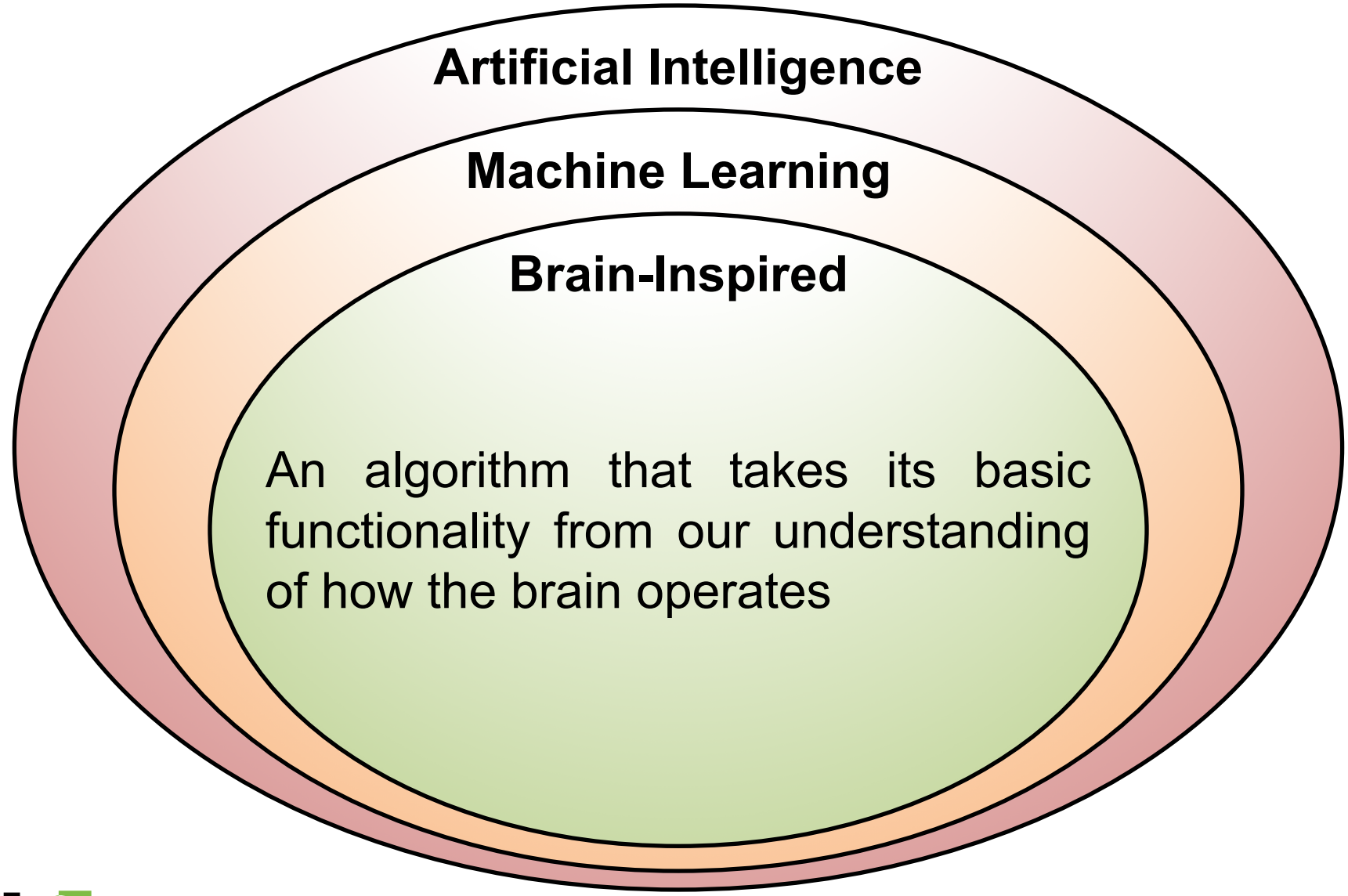
Artificial Intelligence



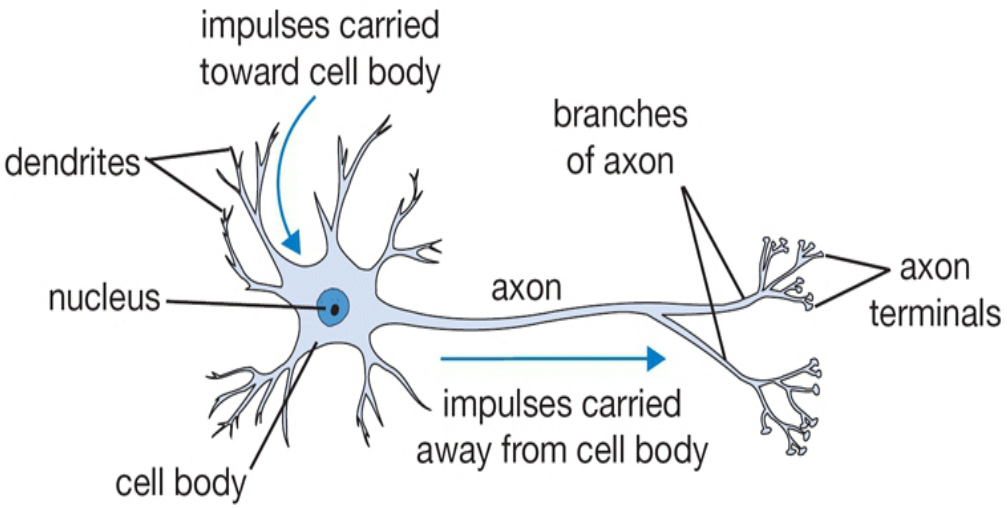
AI and Machine Learning

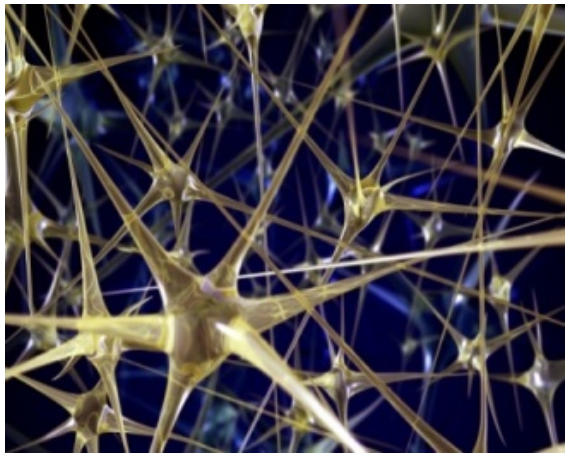


Brain-Inspired Machine Learning



How Does the Brain Work?

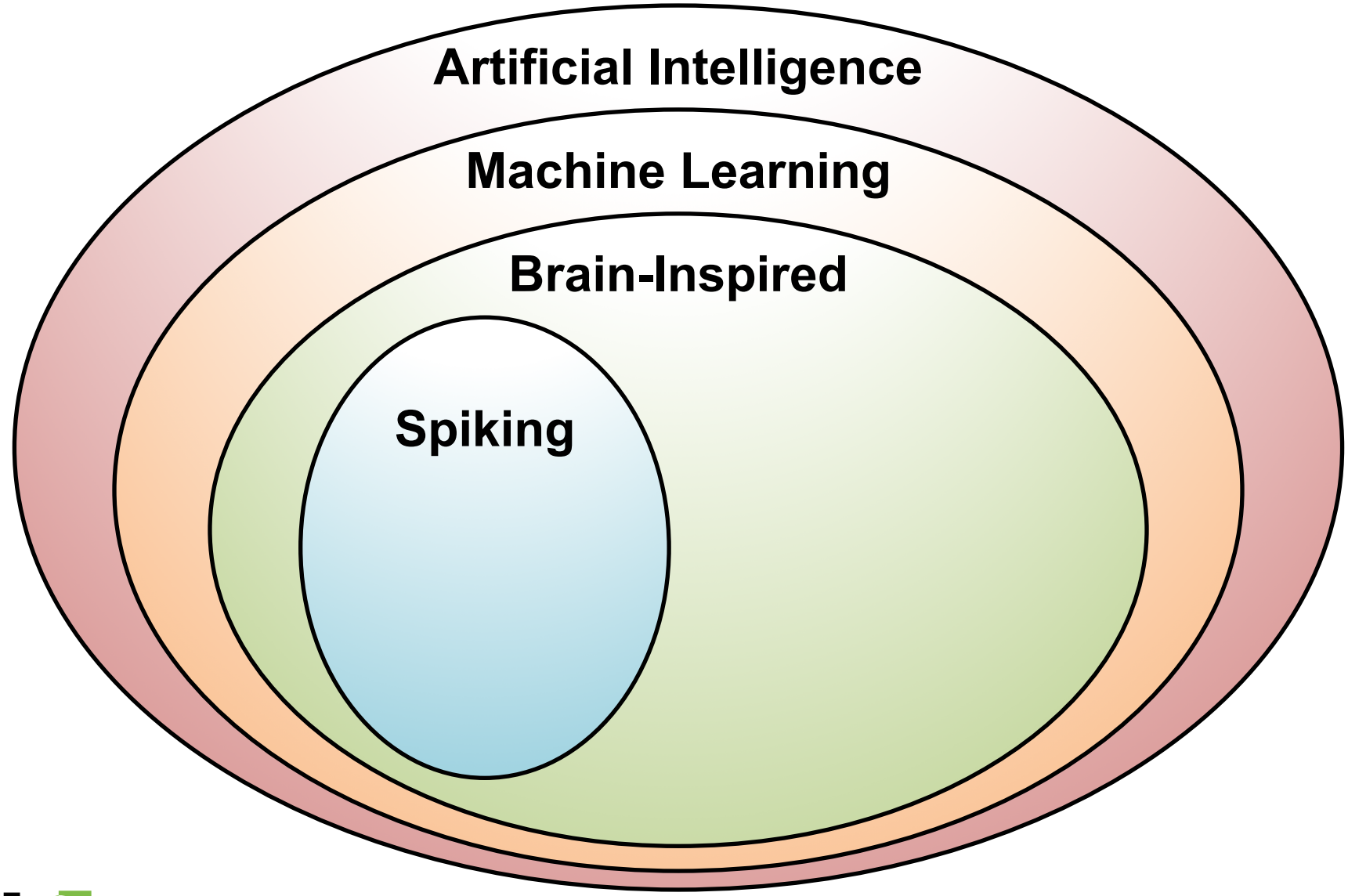




- The basic computational unit of the brain is a **neuron**
  → 86B neurons in the brain
- Neurons are connected with nearly **10^{14} – 10^{15} synapses**
- Neurons receive input signal from **dendrites** and produce output signal along **axon**, which interact with the dendrites of other neurons via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

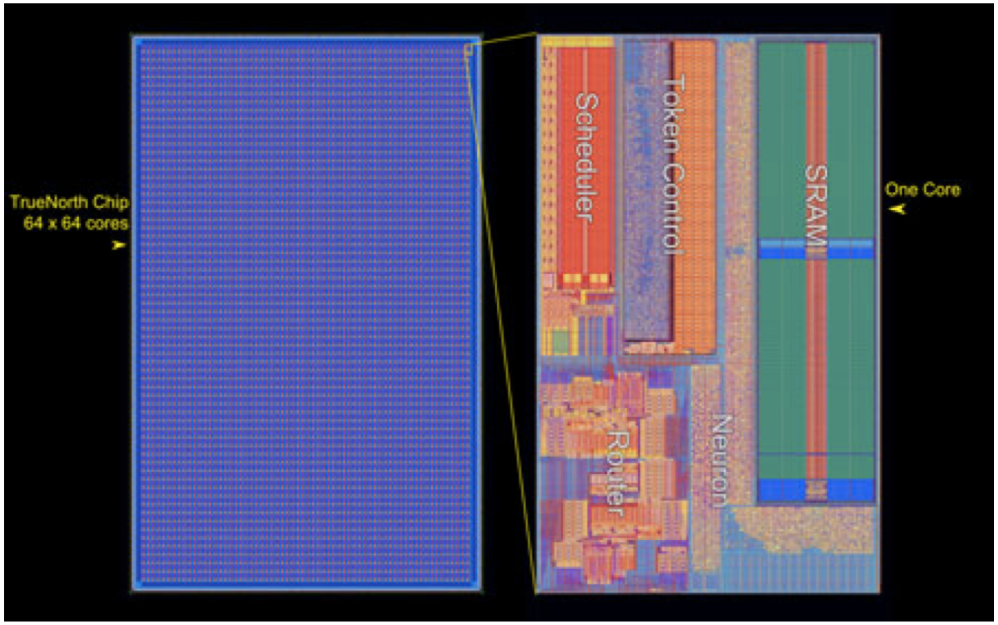
Image Source: Stanford

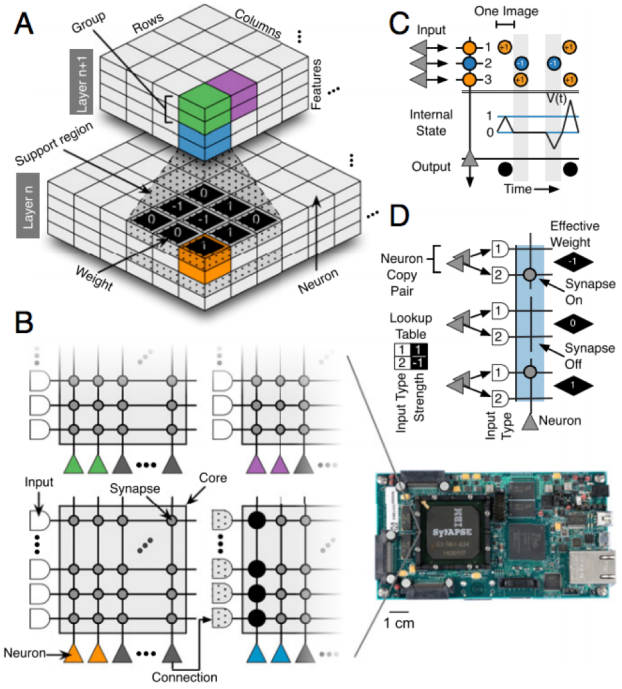
Spiking-based Machine Learning



Spiking Architecture

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

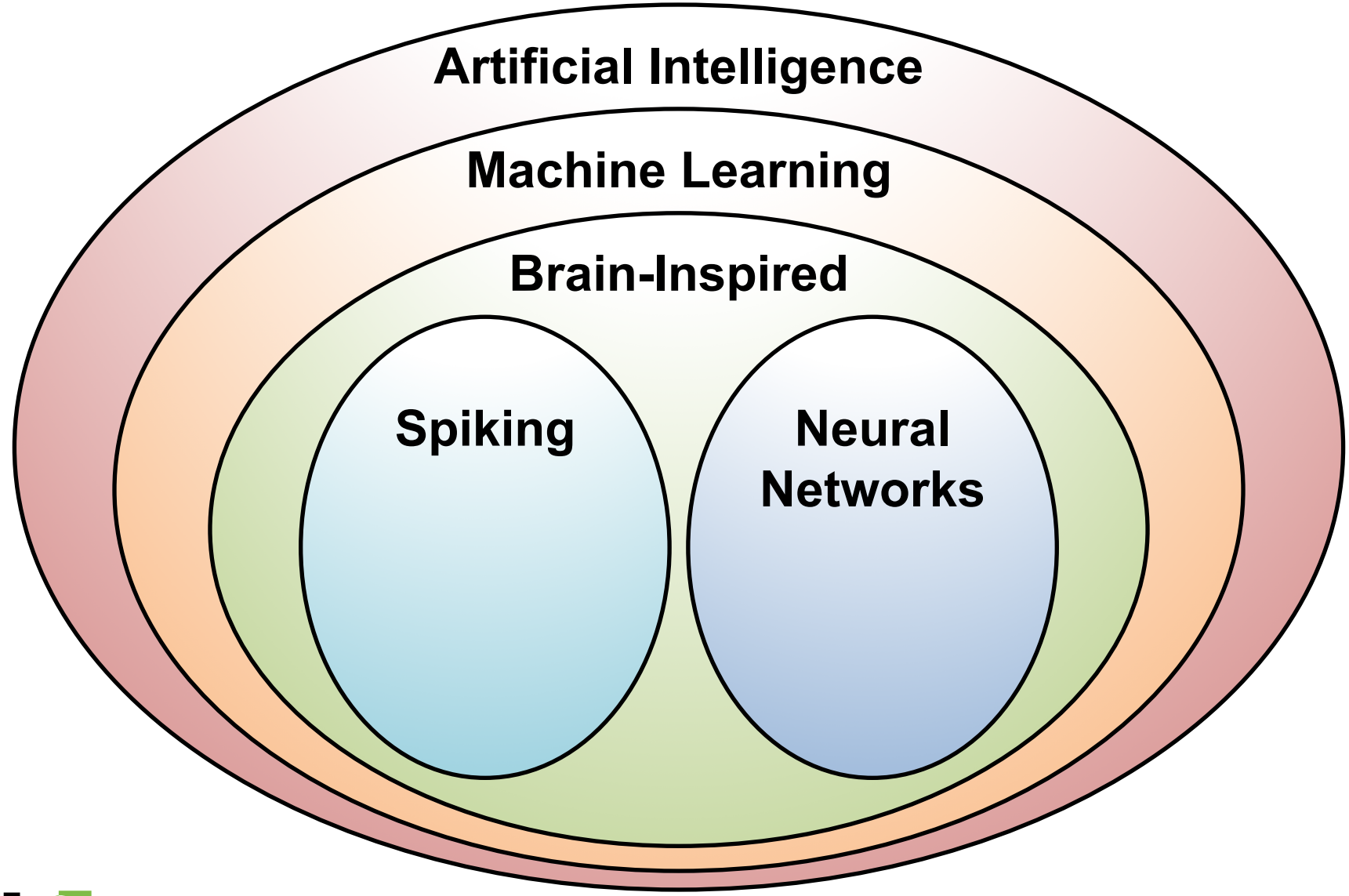




[Merolla et al., Science 2014; Esser et al., PNAS 2016]

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

Machine Learning with Neural Networks



Neural Networks: Weighted Sum

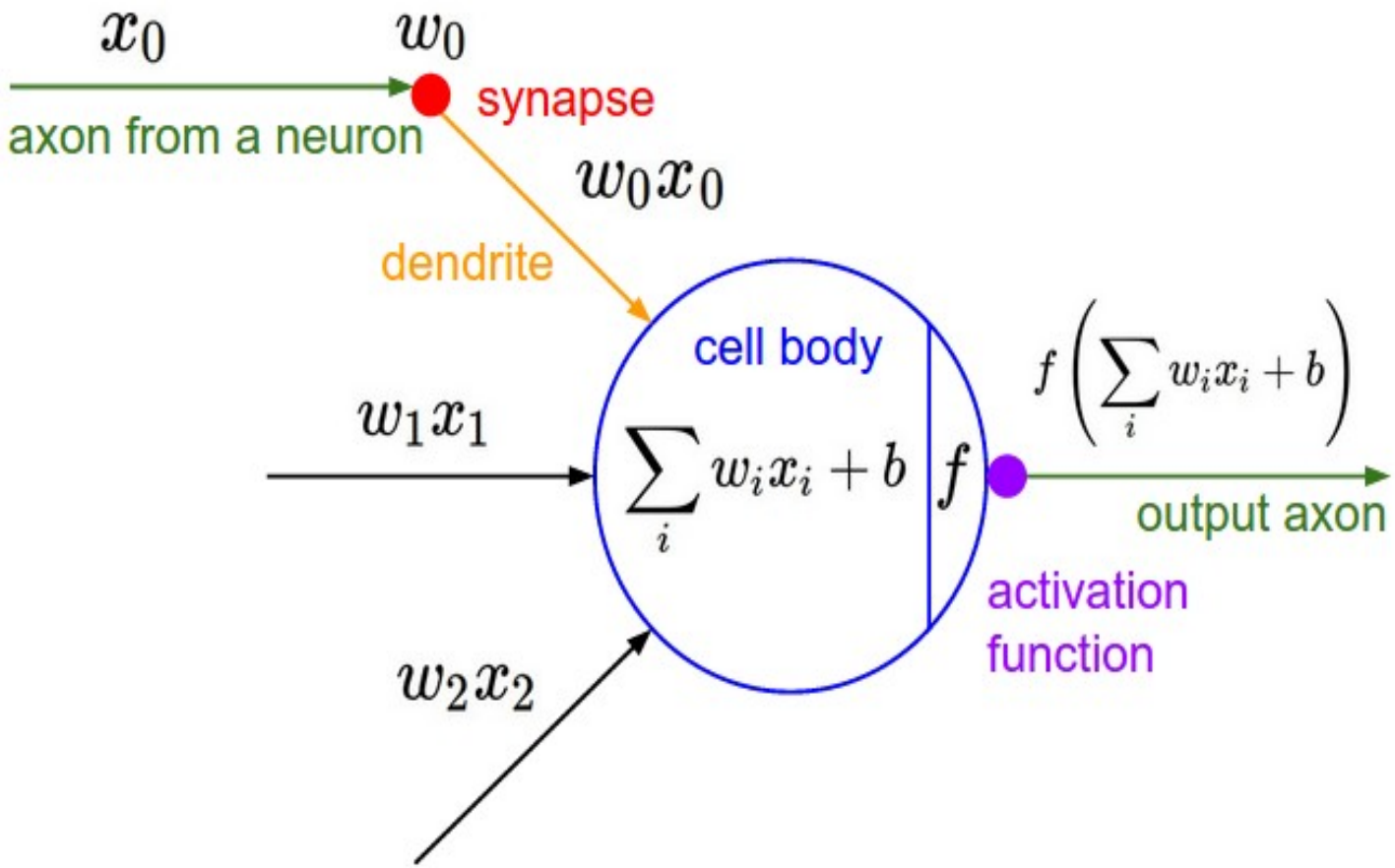
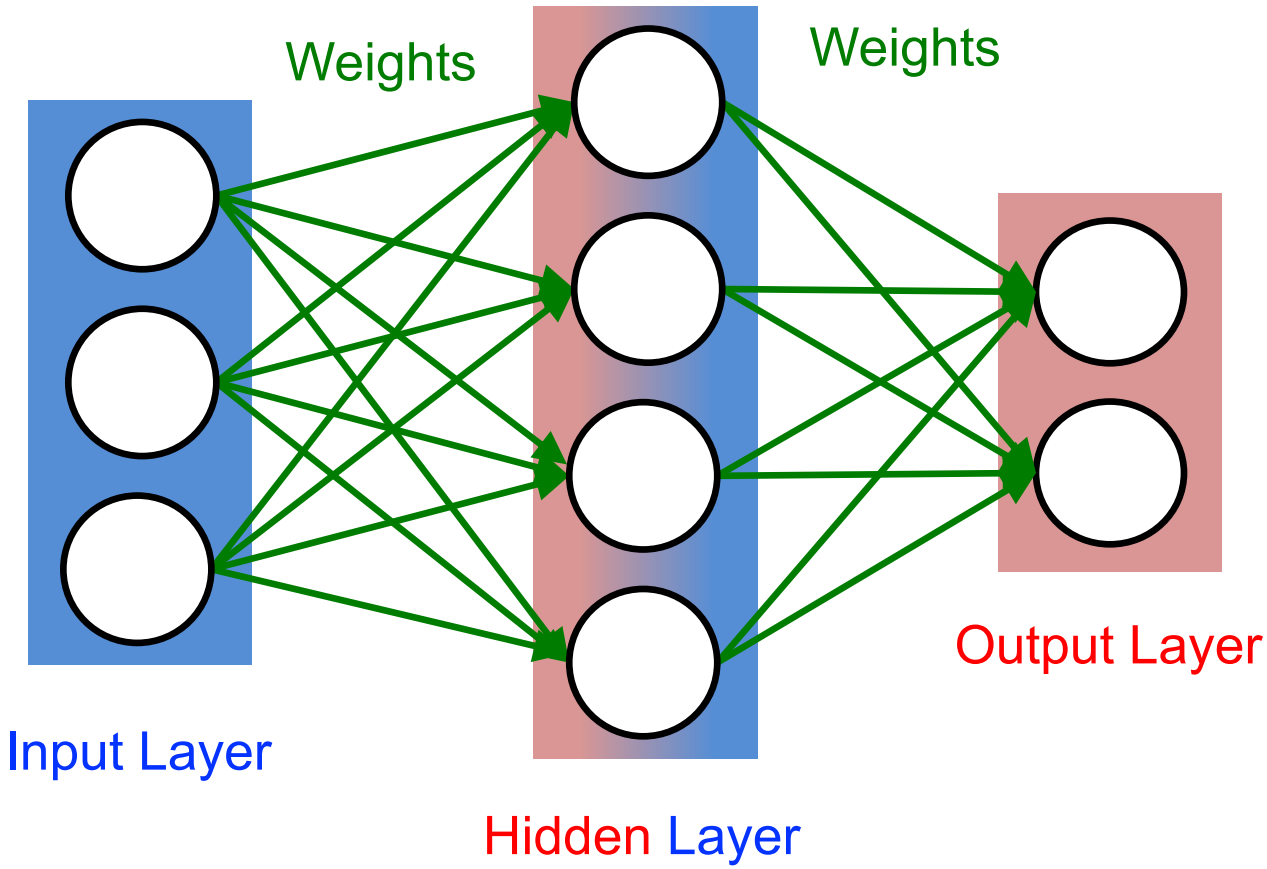
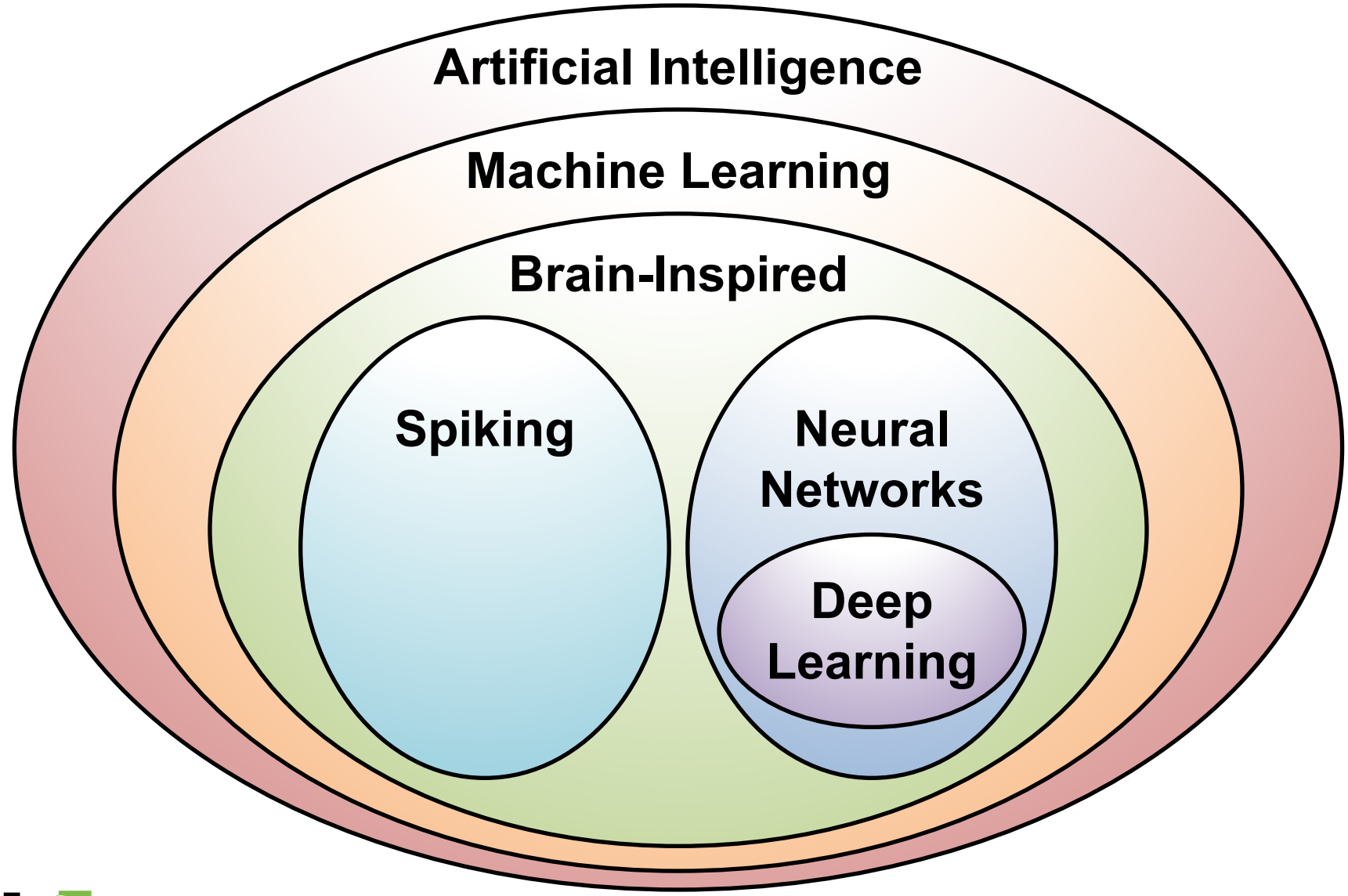


Image Source: Stanford

Many Weighted Sums



Deep Learning



What is Deep Learning?

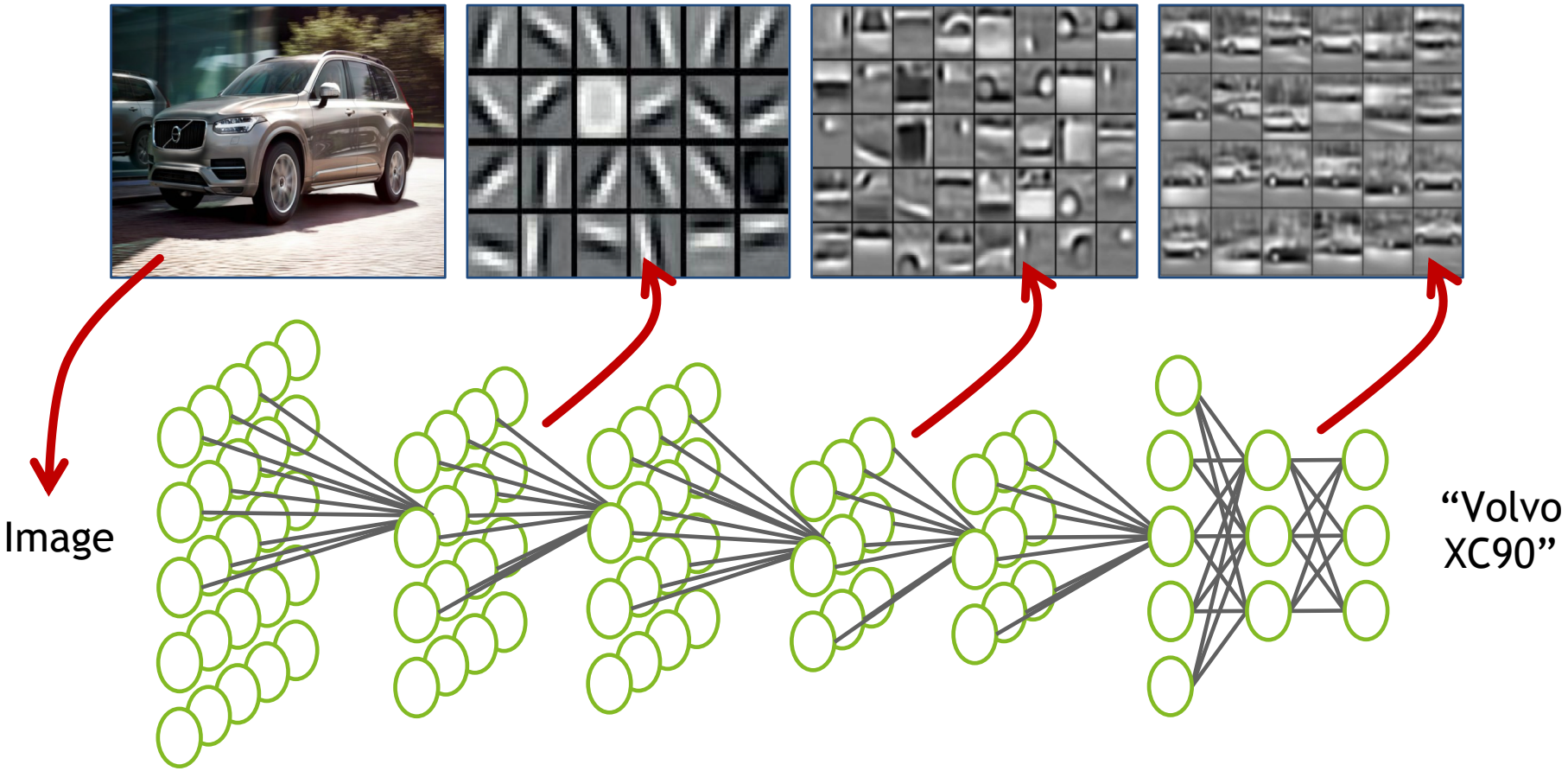
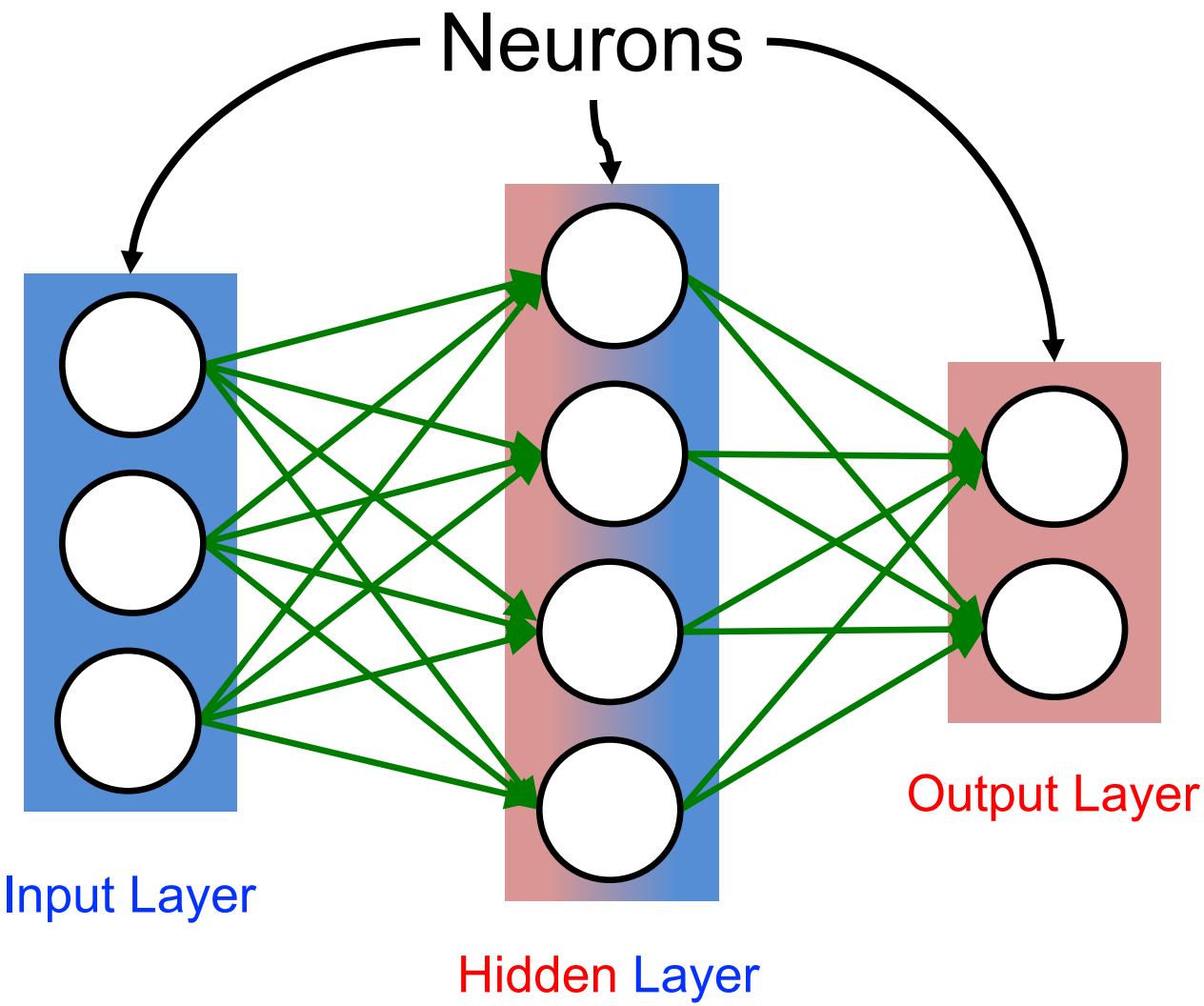


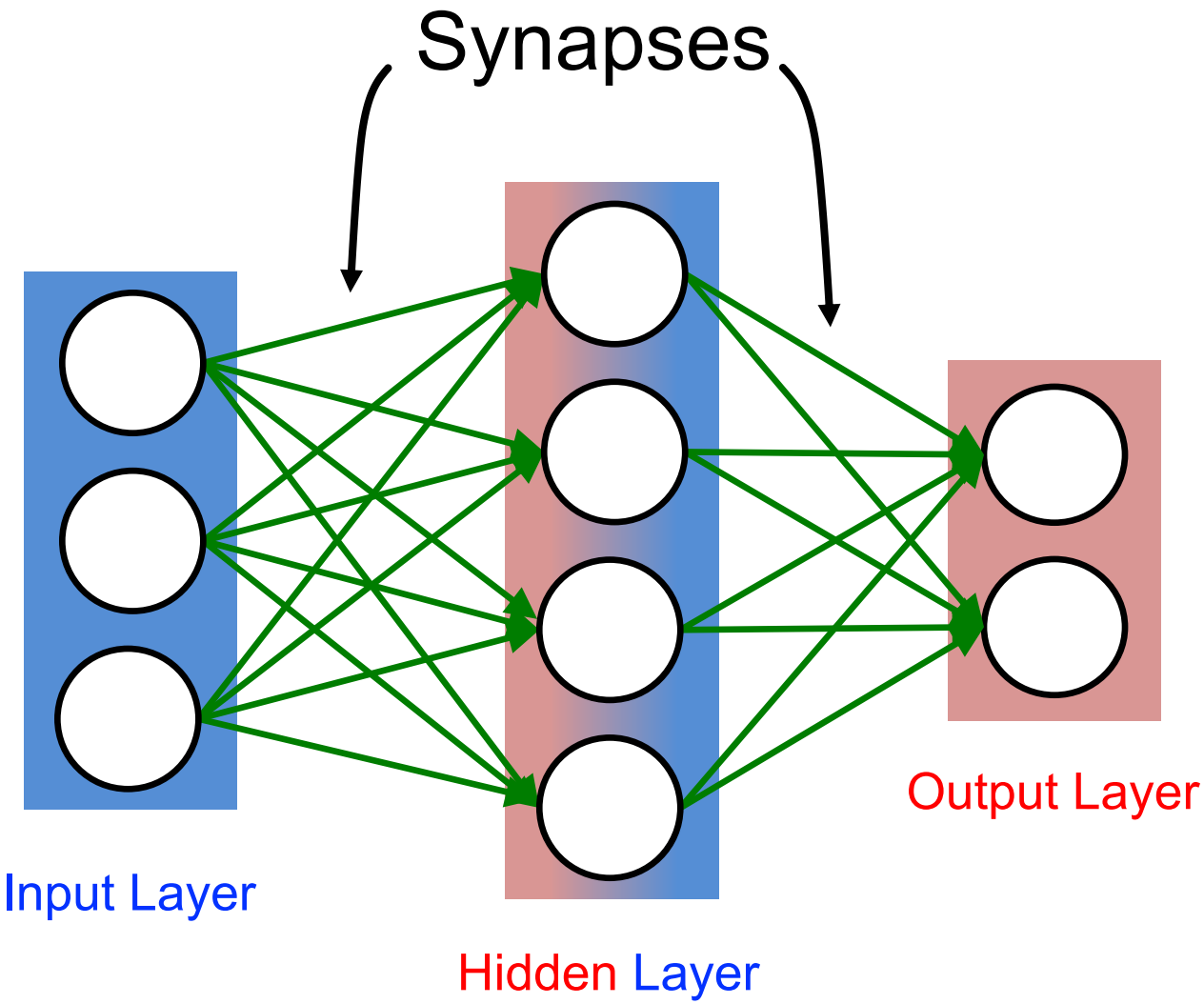
Image Source: [Lee et al., Comm. ACM 2011]

Overview of Deep Neural Networks

DNN Terminology 101



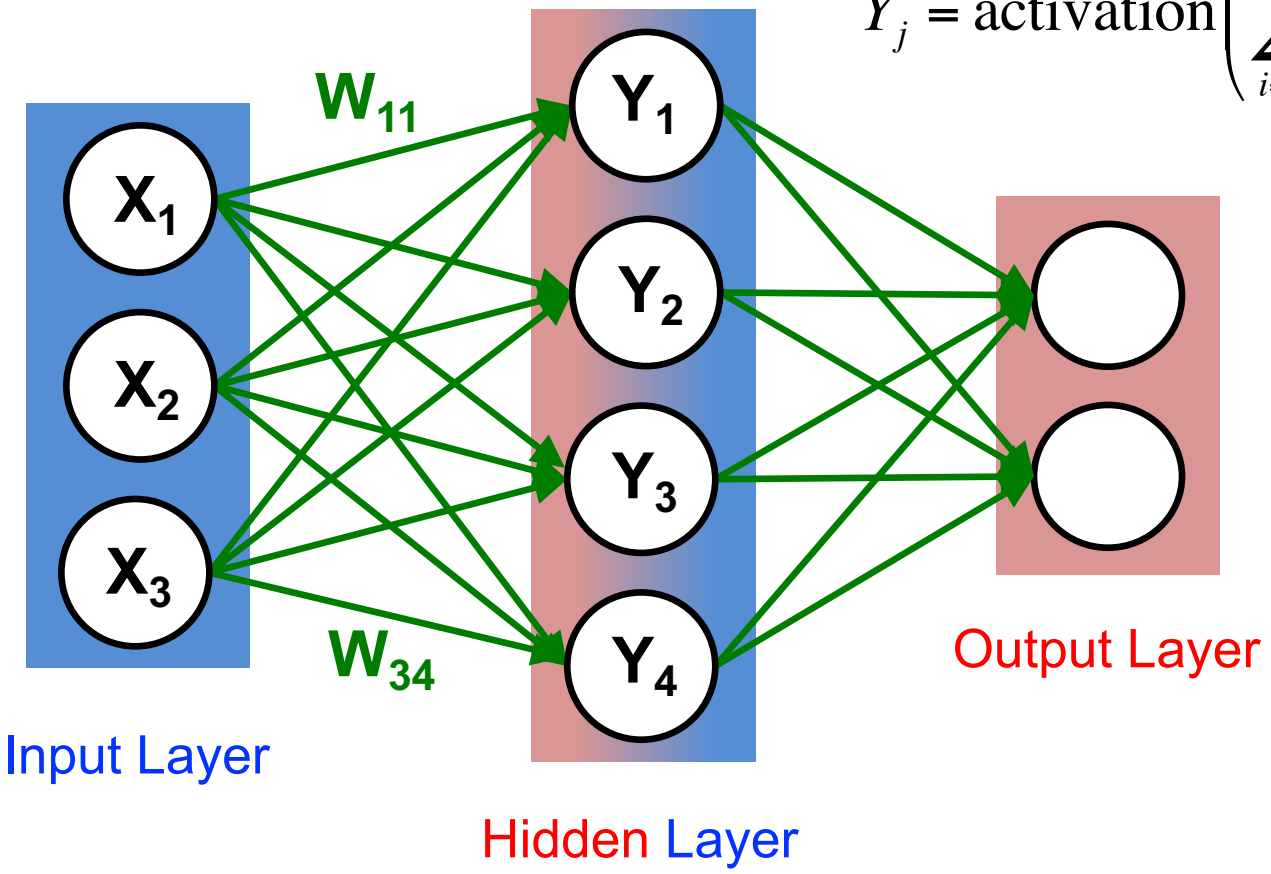
DNN Terminology 101



DNN Terminology 101

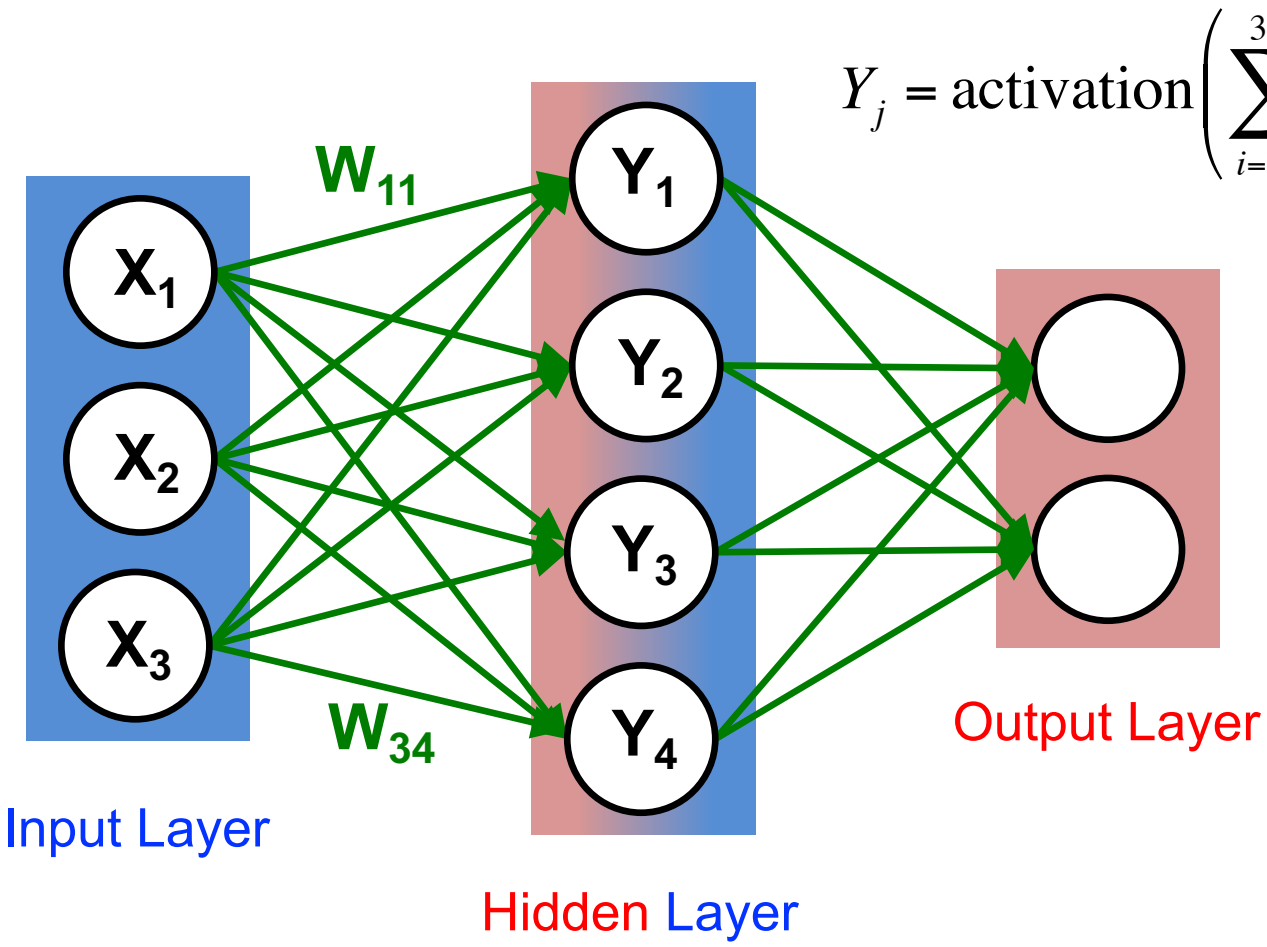
Each synapse has a **weight** for neuron **activation**

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

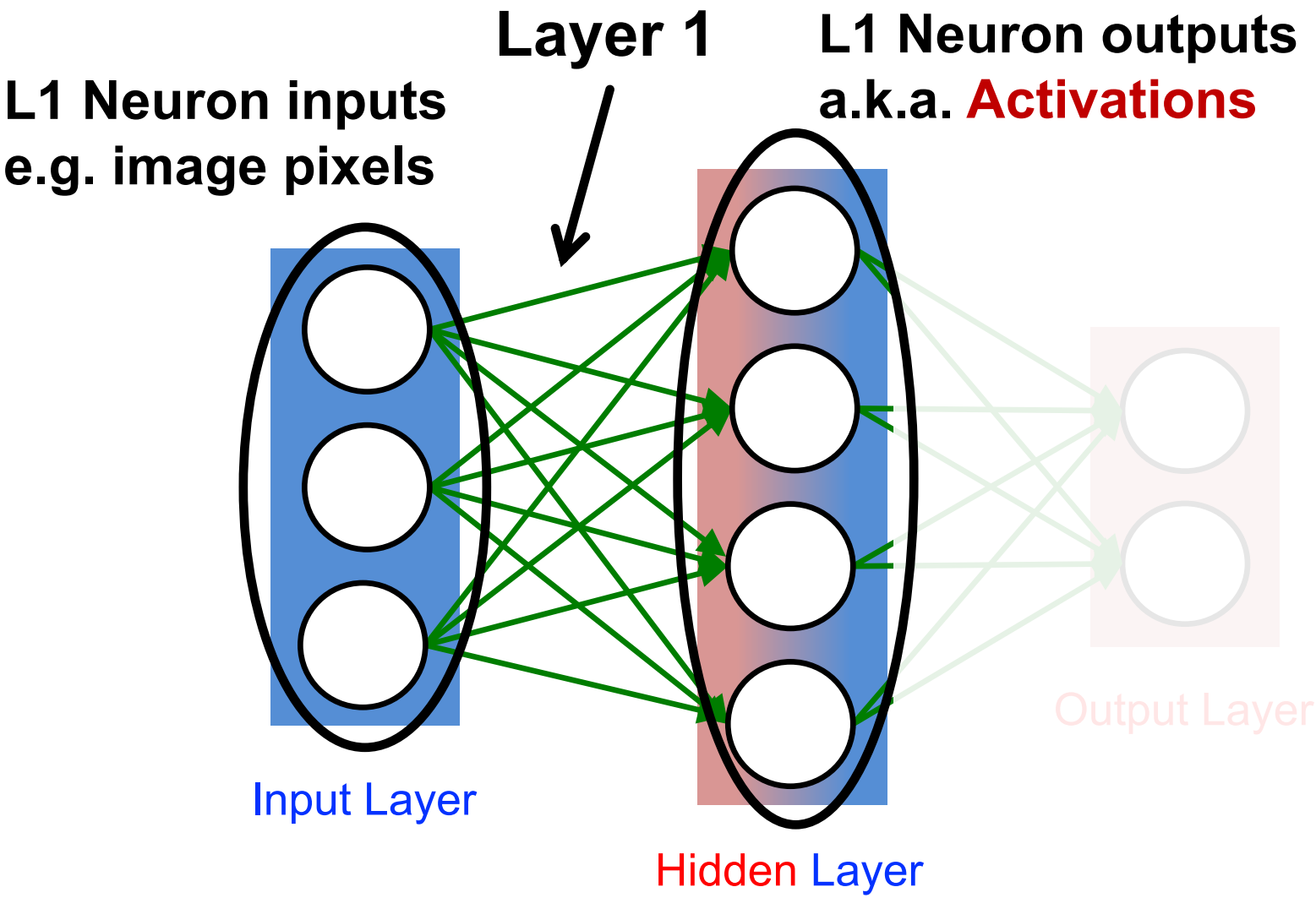


DNN Terminology 101

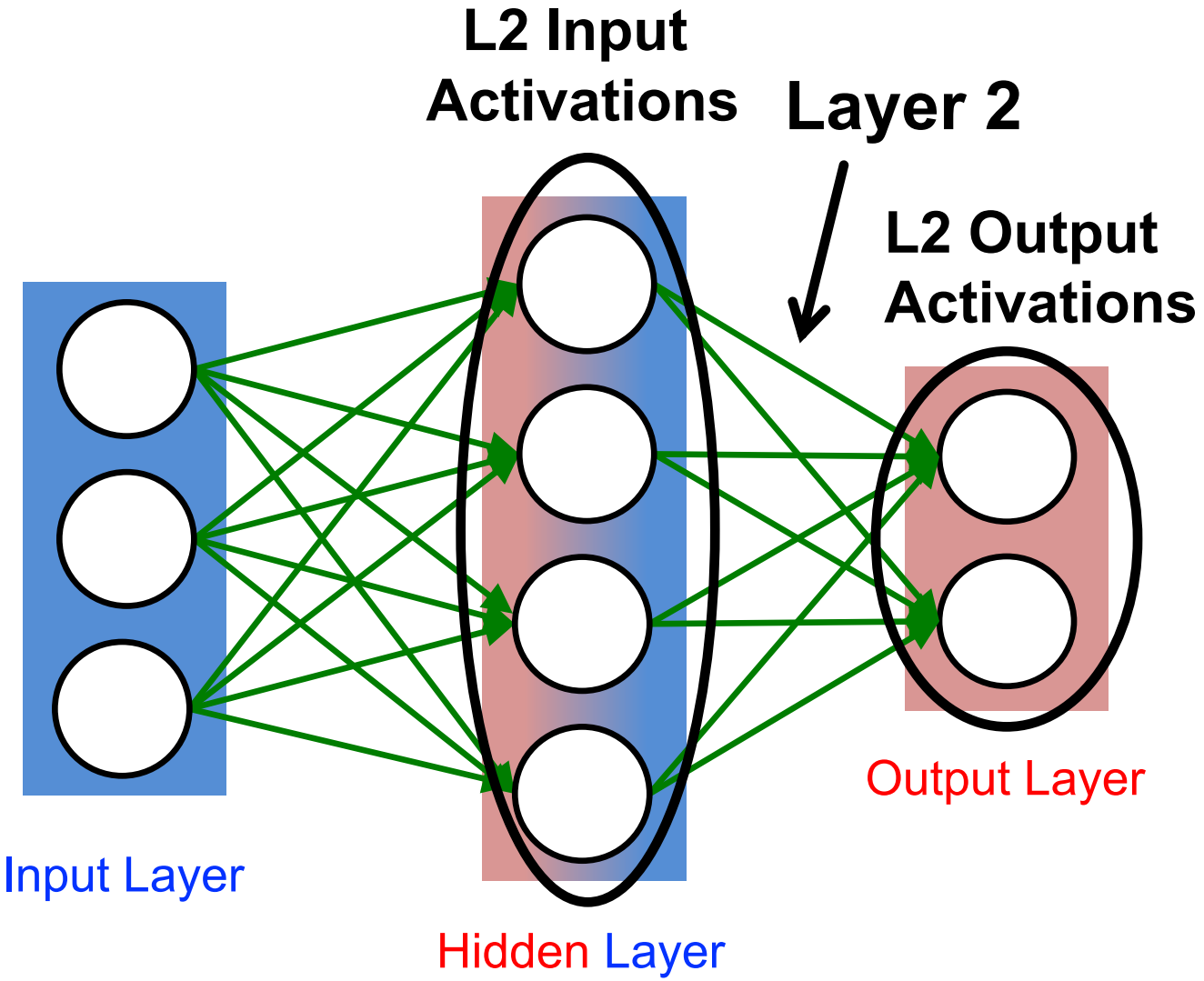
Weight Sharing: multiple synapses use the **same weight value**



DNN Terminology 101

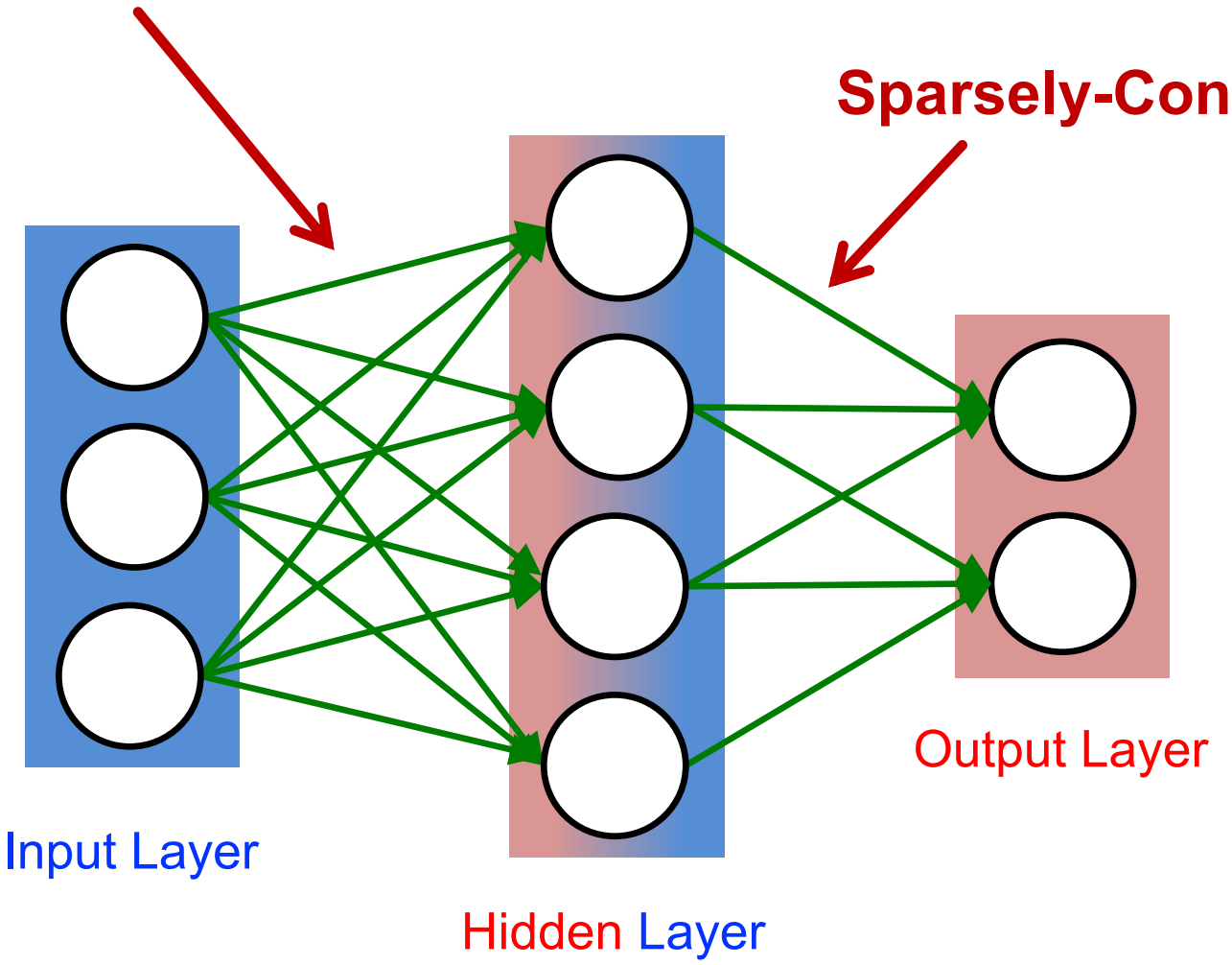


DNN Terminology 101

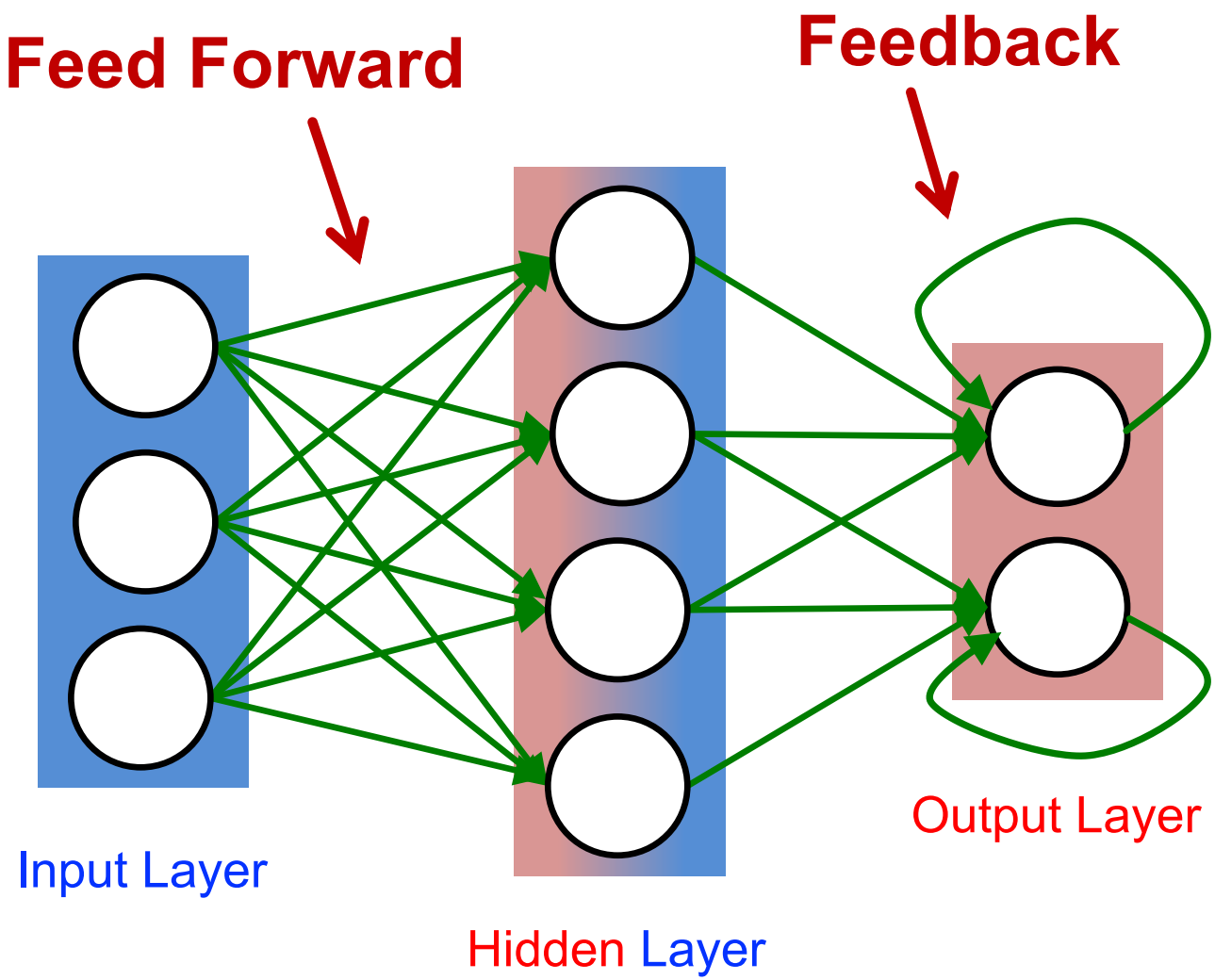


DNN Terminology 101

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



DNN Terminology 101



Popular Types of DNNs

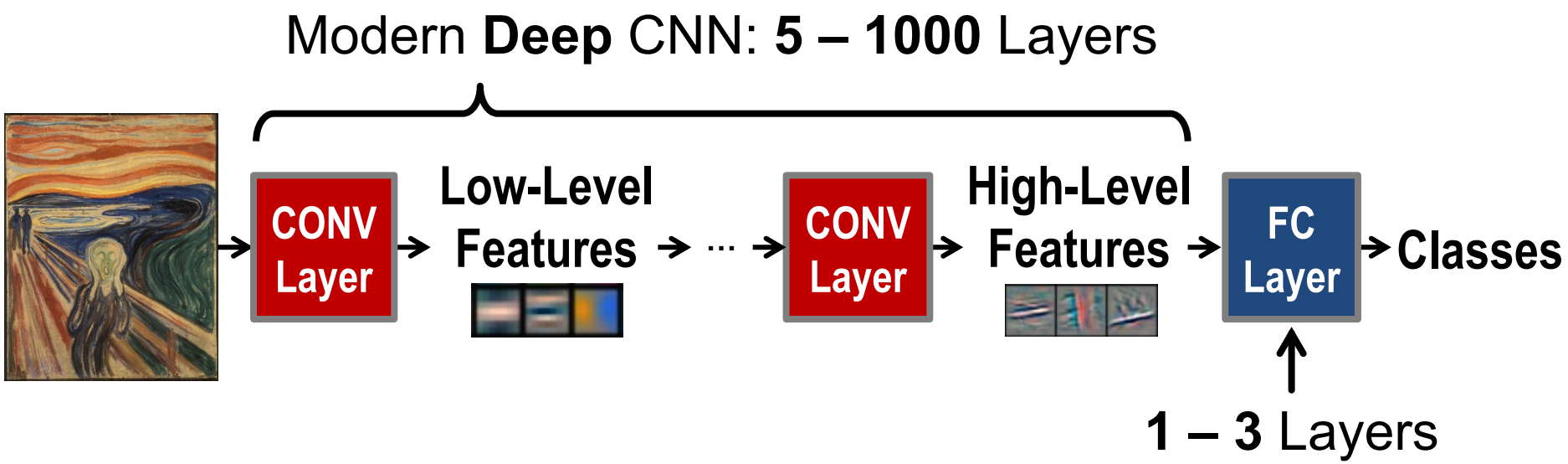
- **Fully-Connected NN**
  - feed forward, a.k.a. multilayer perceptron (MLP)
- **Convolutional NN (CNN)**
  - feed forward, sparsely-connected w/ weight sharing
- **Recurrent NN (RNN)**
  - feedback
- **Long Short-Term Memory (LSTM)**
  - feedback + storage

Inference vs. Training

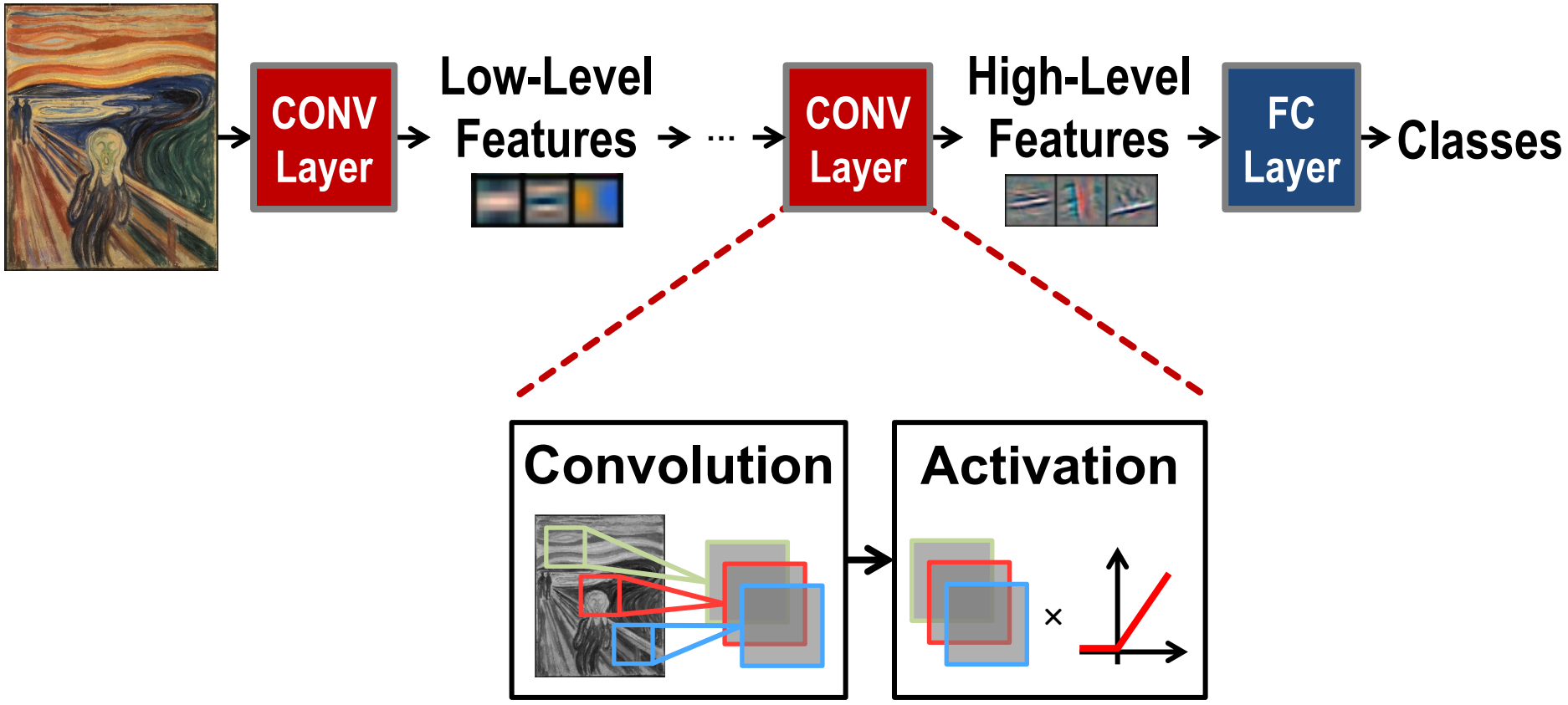
- **Training:** Determine weights
  - **Supervised:**
    - Training set has inputs and outputs, i.e., labeled
  - **Unsupervised / Self-Supervised:**
    - Training set is unlabeled
  - **Semi-supervised:**
    - Training set is partially labeled
  - **Reinforcement:**
    - Output assessed via rewards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

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

Deep Convolutional Neural Networks

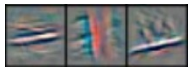


Deep Convolutional Neural Networks

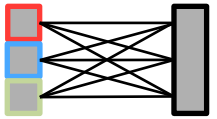


Deep Convolutional Neural Networks

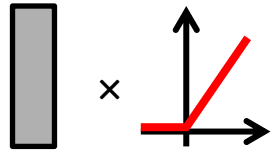
CONV Layer → Low-Level Features → ··· → CONV Layer → High-Level Features → FC Layer → Classes



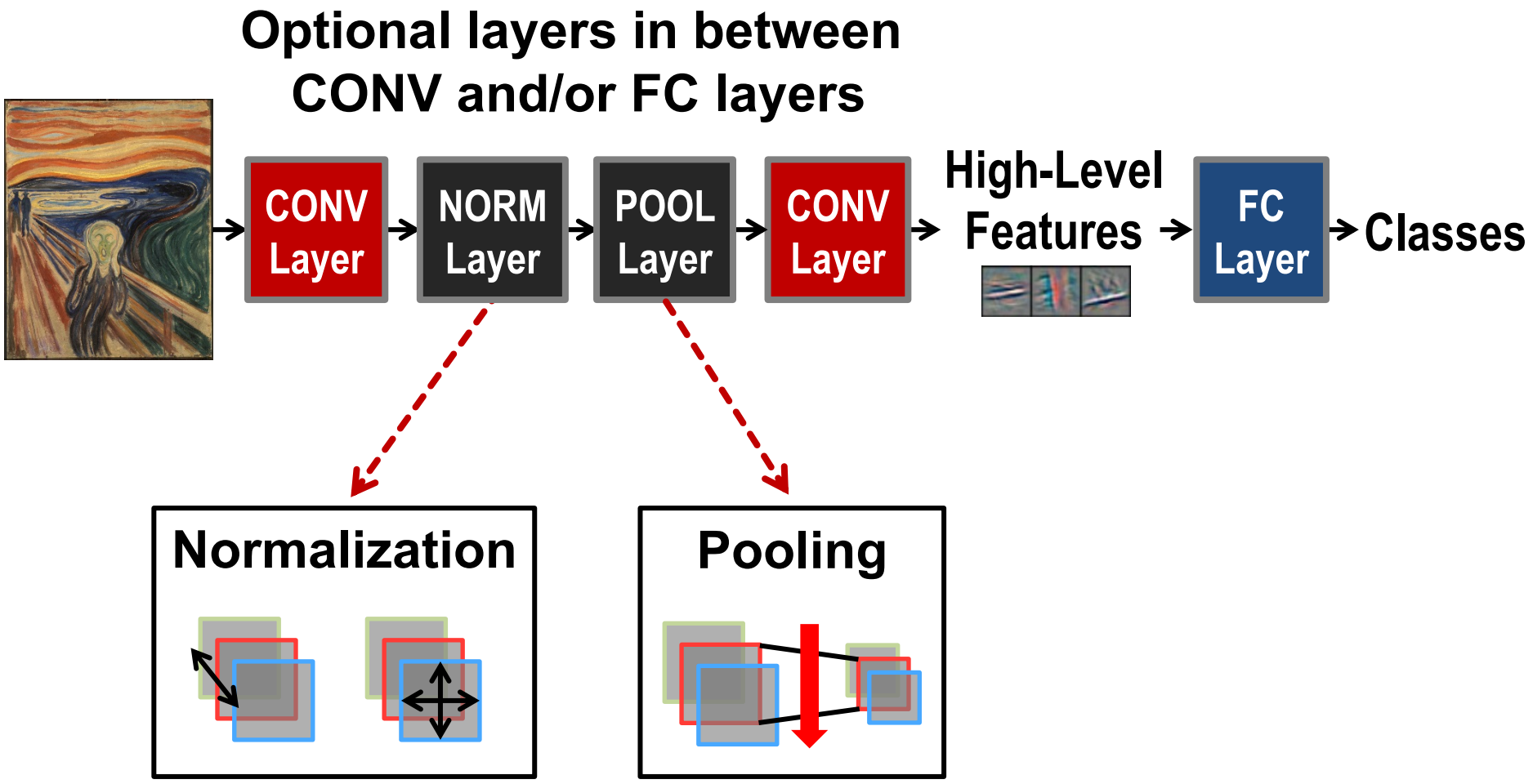
Fully Connected



Activation

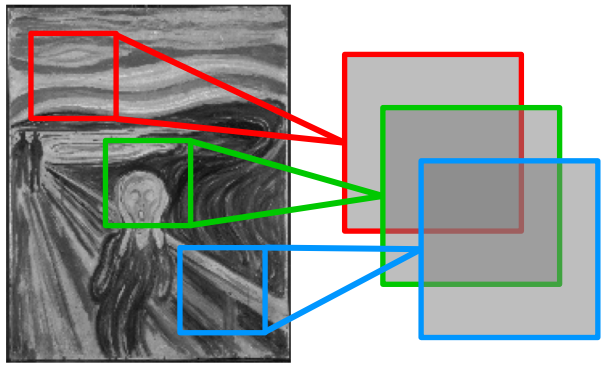


Deep Convolutional Neural Networks



Deep Convolutional Neural Networks

CONV Layer → NORM Layer → POOL Layer → CONV Layer → High-Level Features → FC Layer → Classes



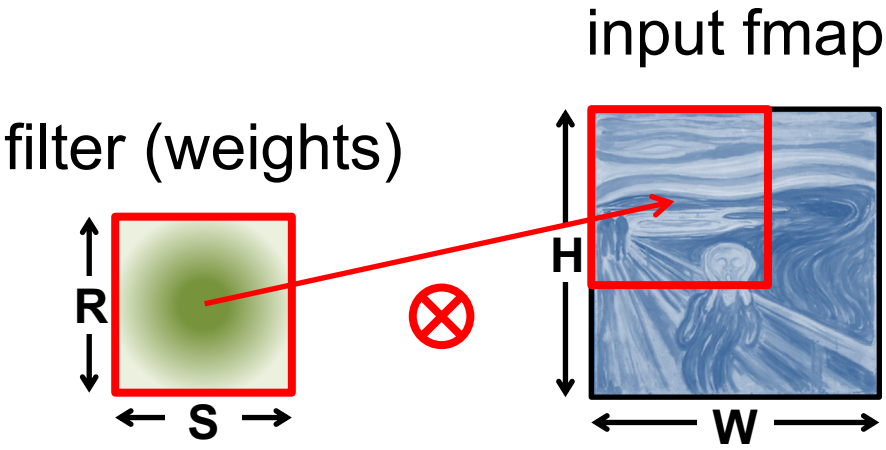
Convolutions account for more than 90% of overall computation, dominating **runtime** and **energy consumption**

Convolution (CONV) Layer

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

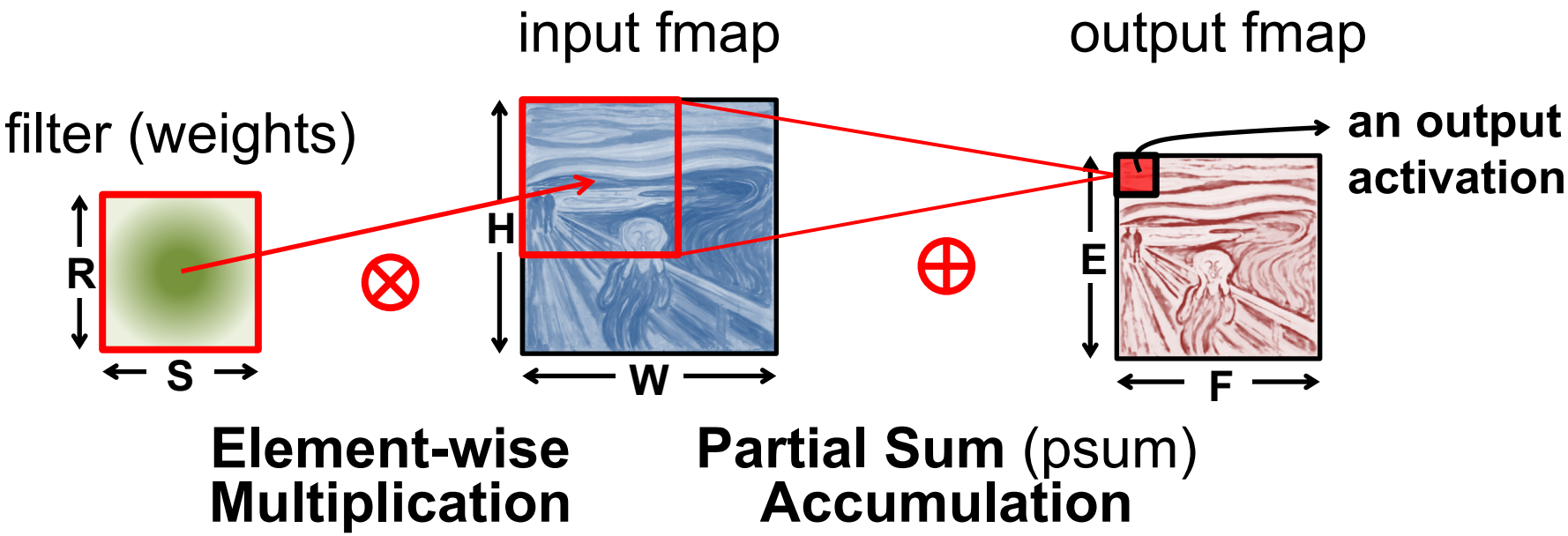
filter (weights)

Convolution (CONV) Layer

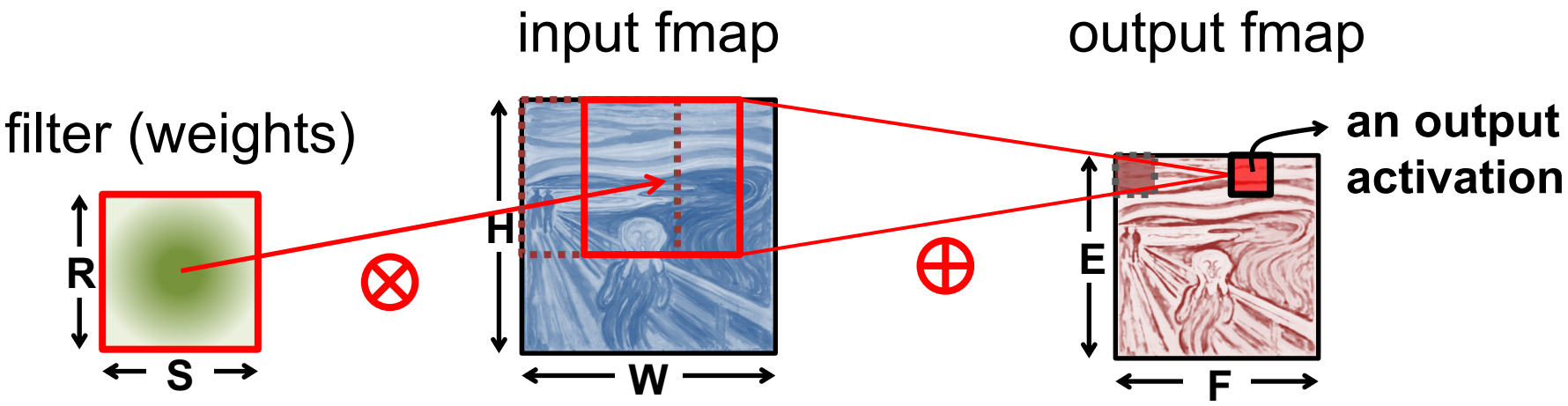


Element-wise Multiplication

Convolution (CONV) Layer

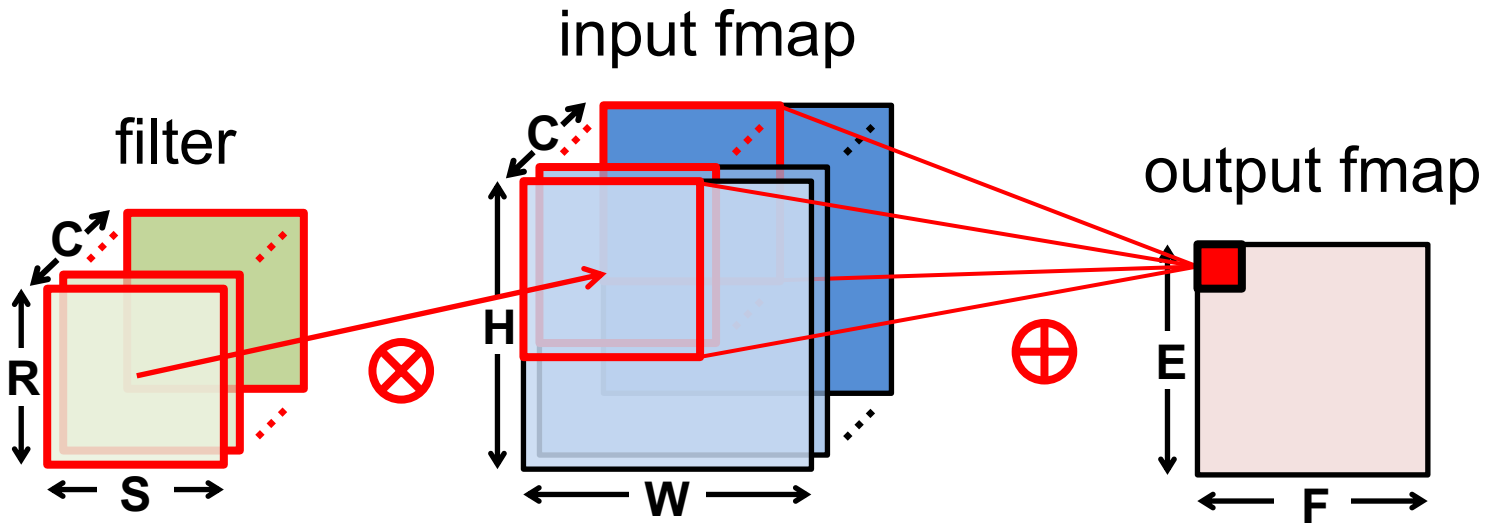


Convolution (CONV) Layer



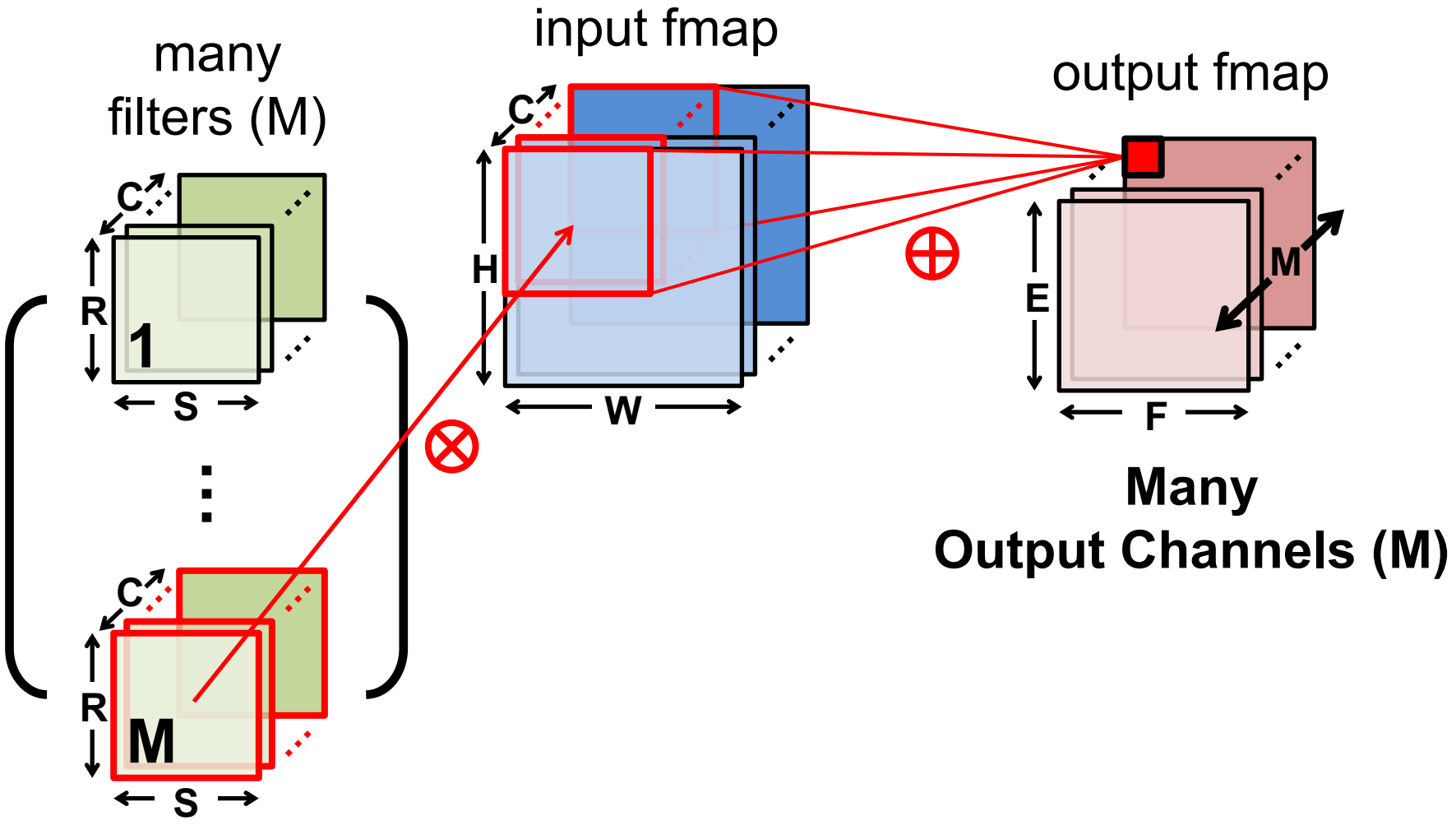
Sliding Window Processing

Convolution (CONV) Layer

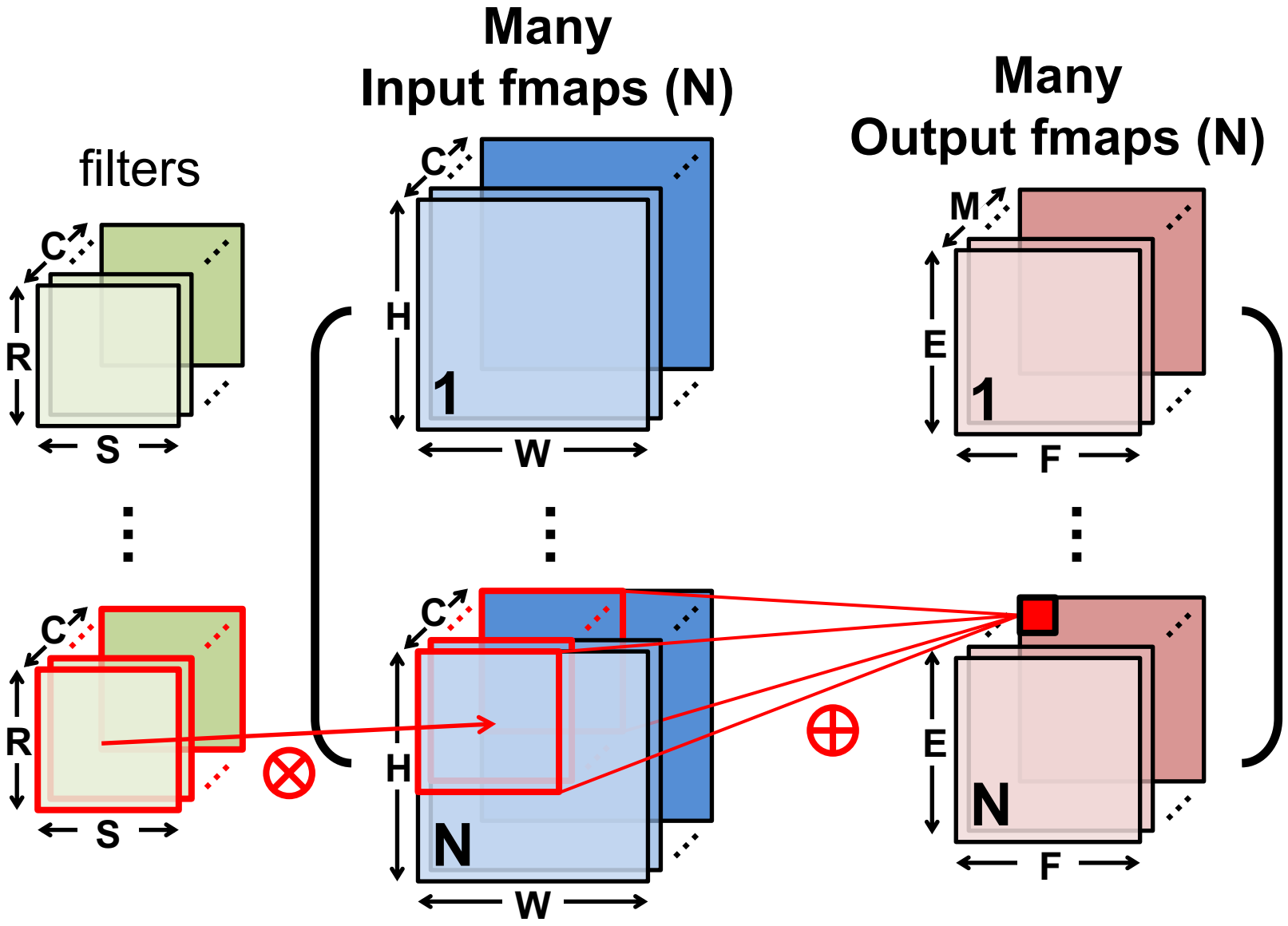


Many Input Channels (C)

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)

Biases (B)

Input fmaps (I)

Filter weights (W)

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

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

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

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

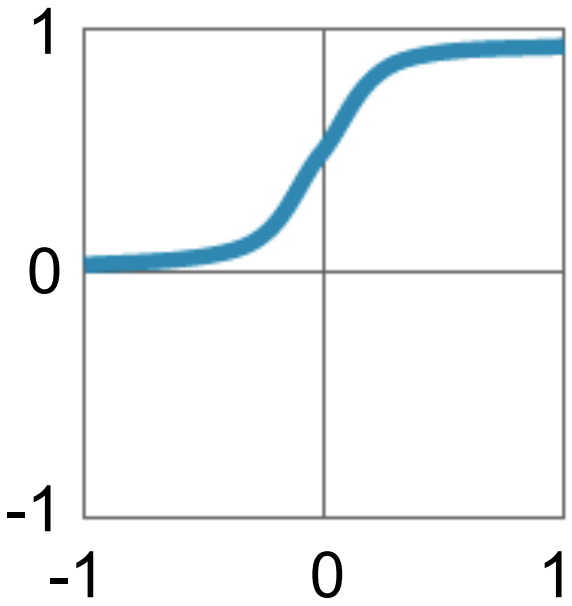
} for each output fmap value

convolve a window and apply activation

```
                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]);
            }
        }
    }
}
```

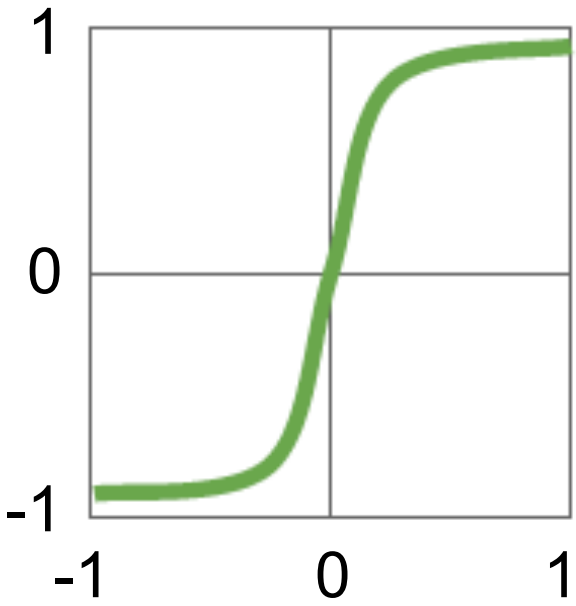
Traditional Activation Functions

Sigmoid



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

Hyperbolic Tangent

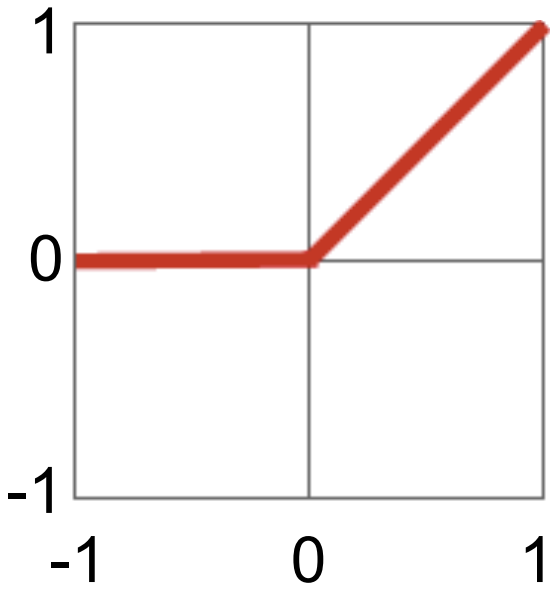


$$y=(e^{x}-e^{-x})/(e^{x}+e^{-x})$$

Image Source: Caffe Tutorial

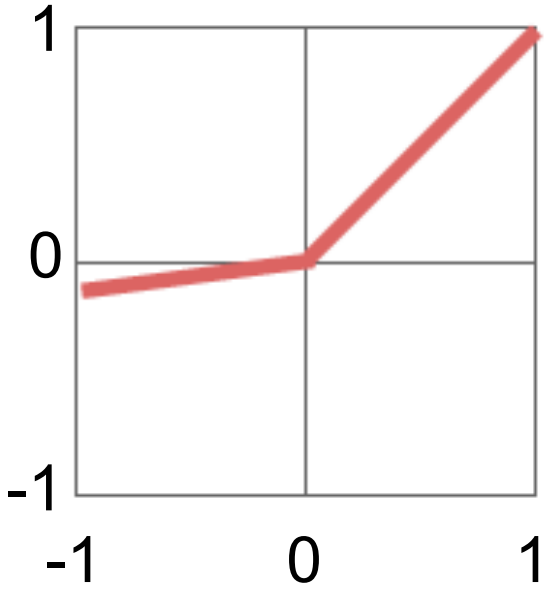
Modern Activation Functions

Rectified Linear Unit (ReLU)



$y = \max(0, x)$

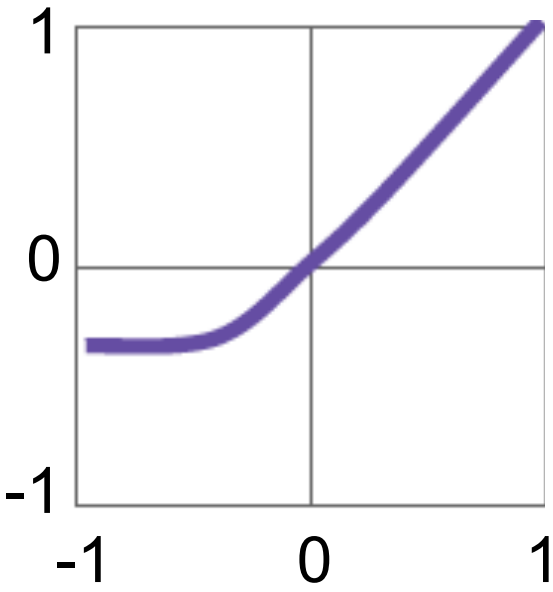
Leaky ReLU



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

α = small const. (e.g. 0.1)

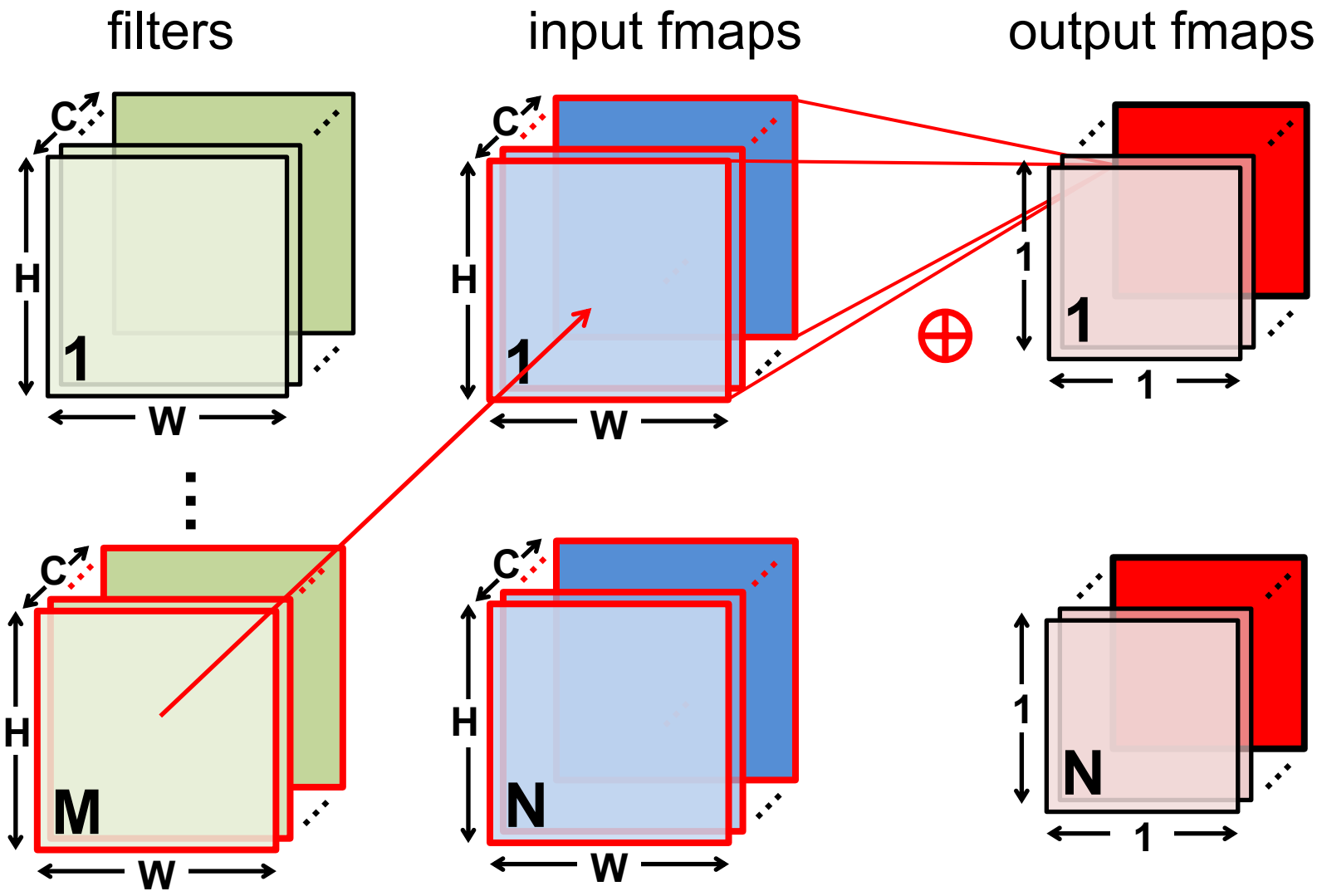
Exponential LU



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

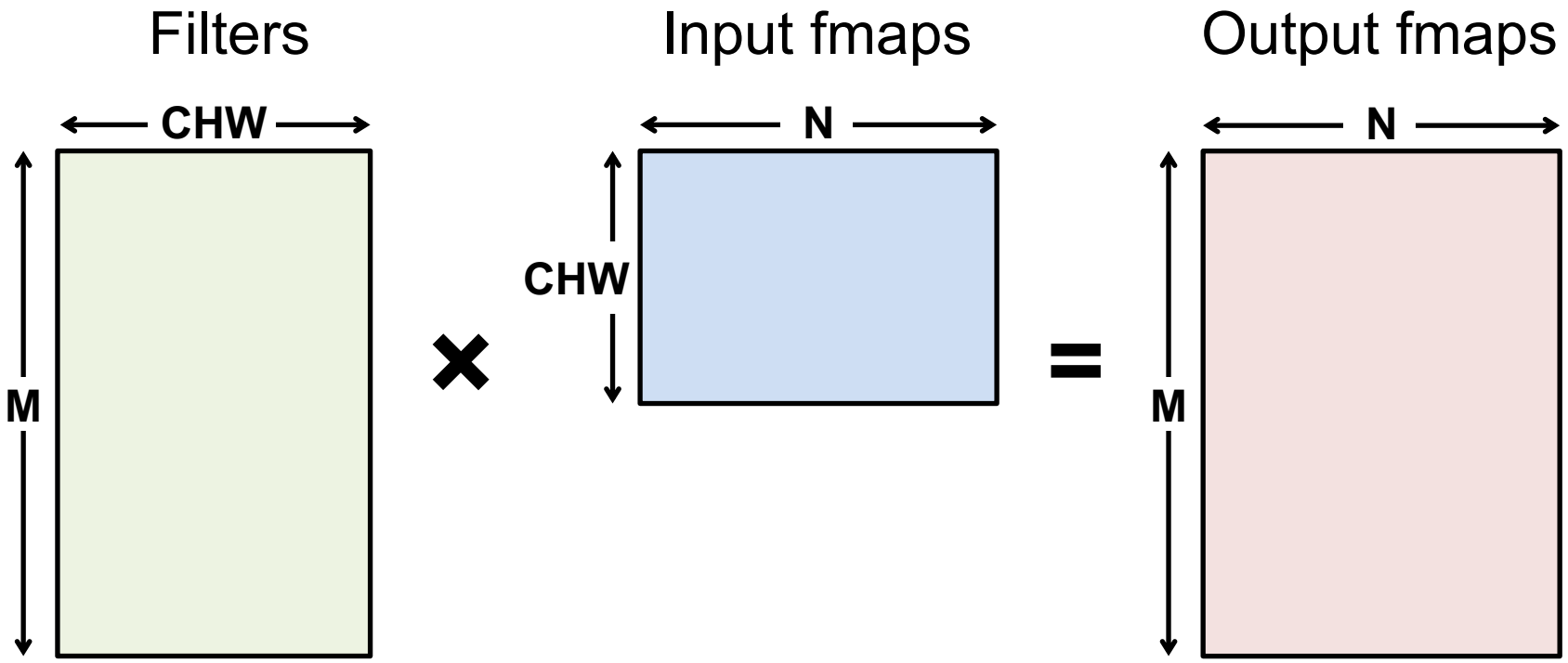
Image Source: Caffe Tutorial

FC Layer – from CONV Layer POV



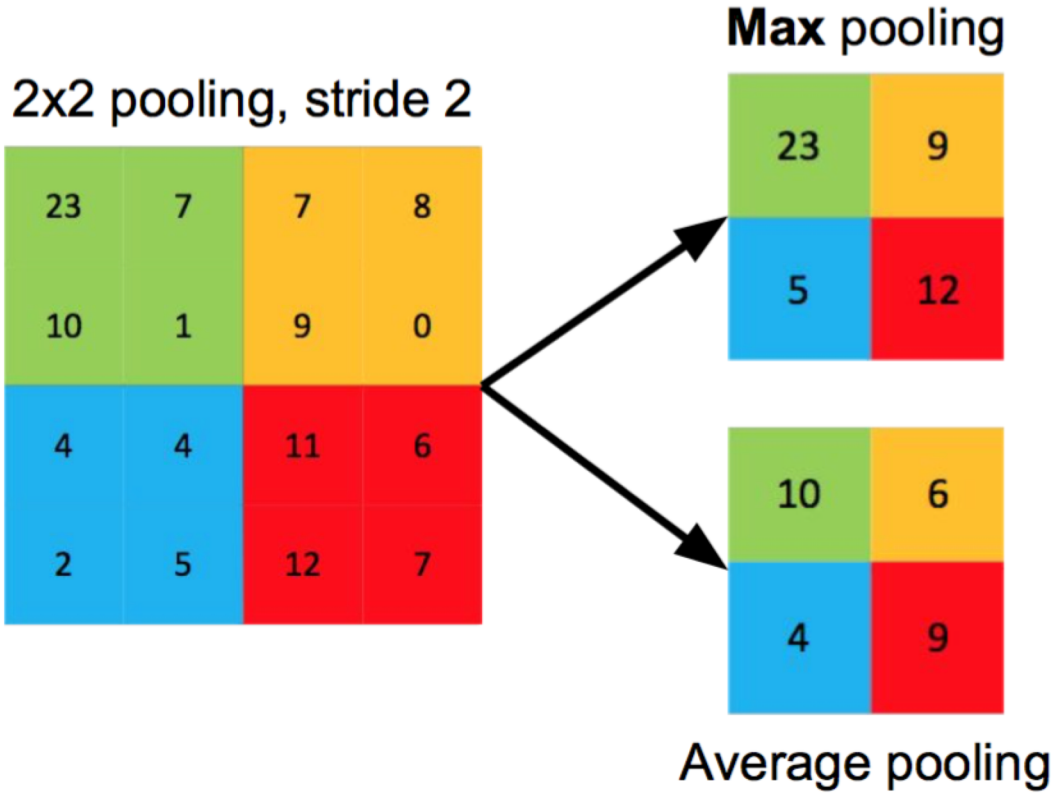
Fully-Connected (FC) Layer

- 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 → depending on stride



Increases translation-invariance and noise-resilience

Image Source: Caffe Tutorial

POOL Layer Implementation

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

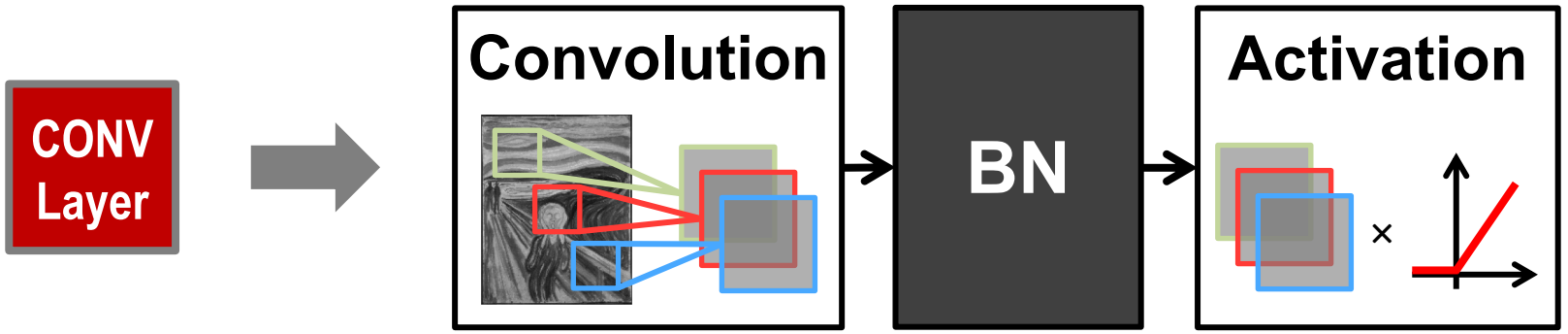
```
for (n=0; n<N; n++) {
    for (m=0; m<M; m++) {
        for (x=0; x<F; x++) {
            for (y=0; y<E; y++) {

                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**

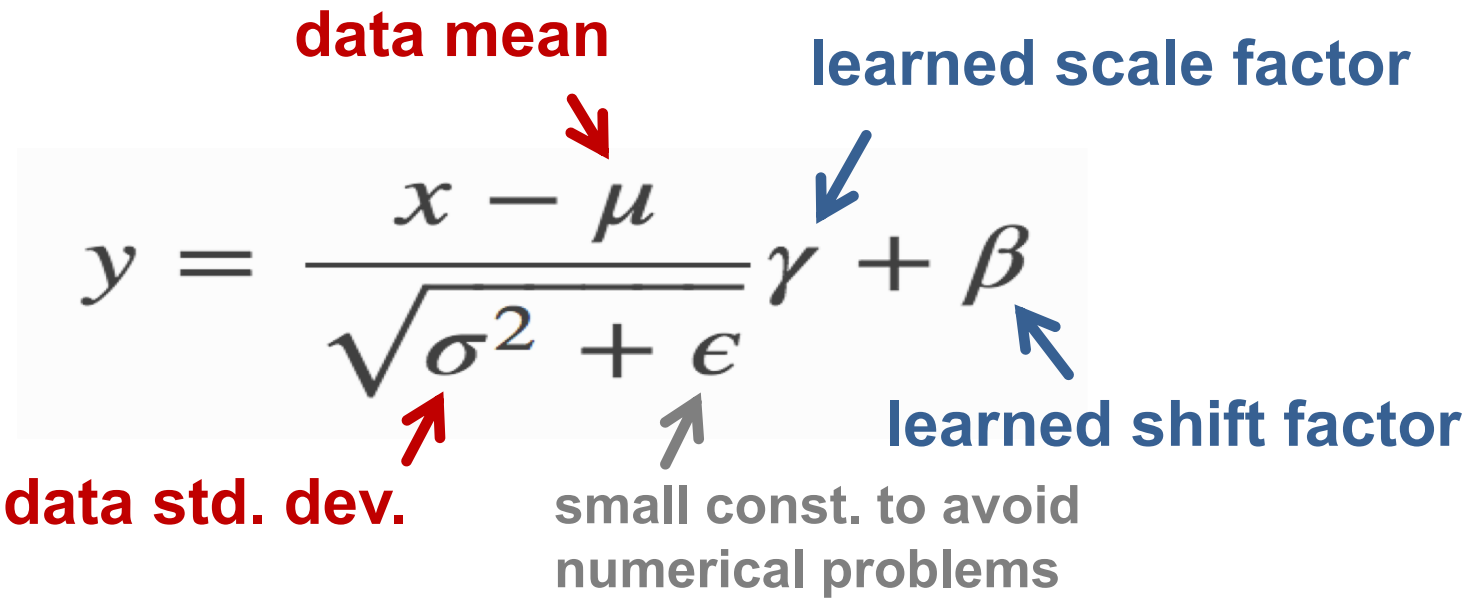


Believed to be key to getting high accuracy and faster training on very deep neural networks.

[Ioffe et al., ICML 2015]

BN Layer Implementation

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



Relevant Components for Tutorial

- Typical operations that we will discuss:
  - Convolution (CONV)
  - Fully-Connected (FC)
  - Max Pooling
  - ReLU