Overview of DNNs

Chia-Chi Tsai (蔡家齊)
cctsai@gs.ncku.edu.tw
AI System Lab
Department of Electrical Engineering
National Cheng Kung University

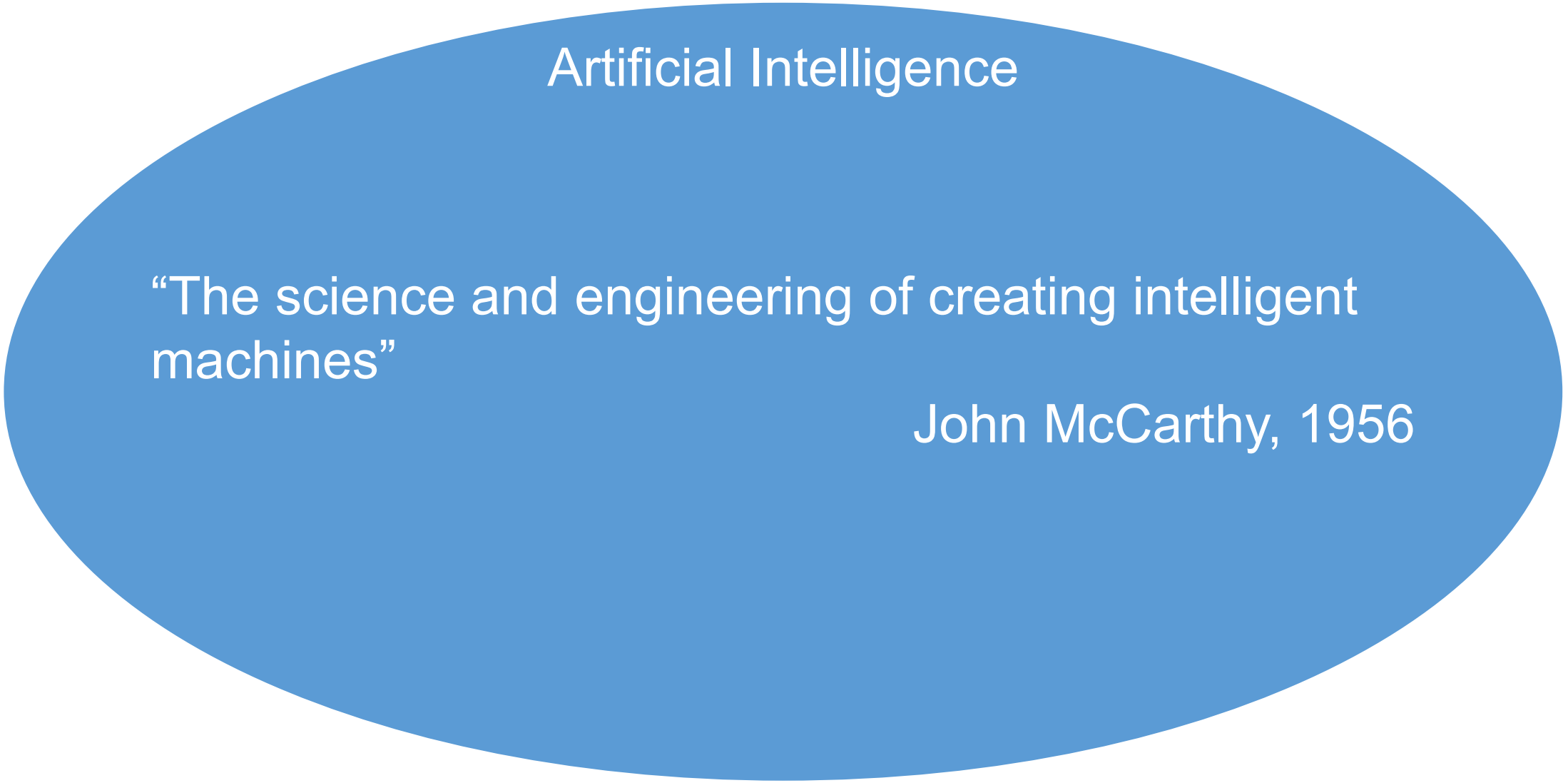
Outline

- AI, ML and DL
- ML Basics
- DL Overview

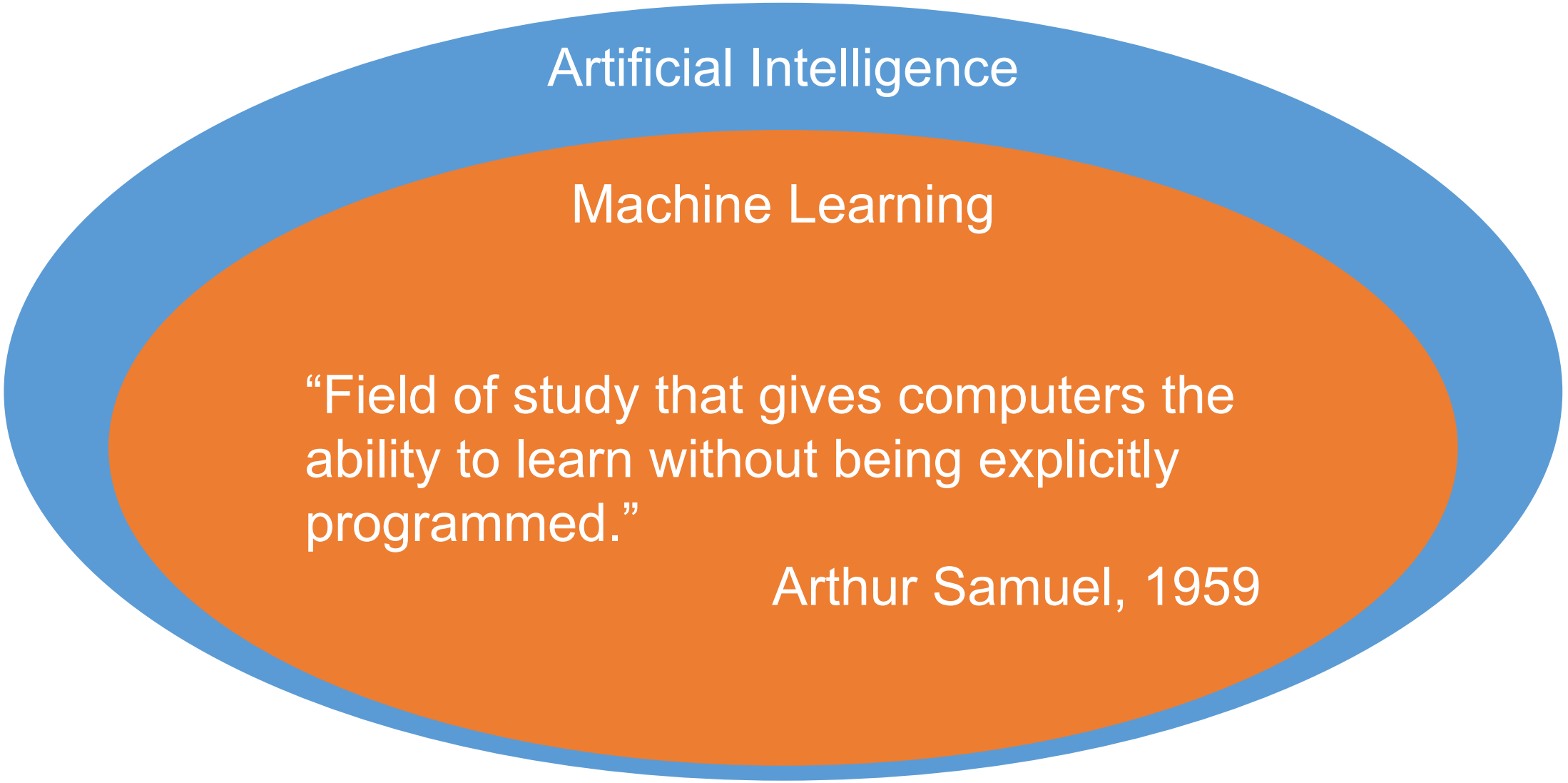
Outline

- AI, ML and DL
- ML Basics
- DL Overview

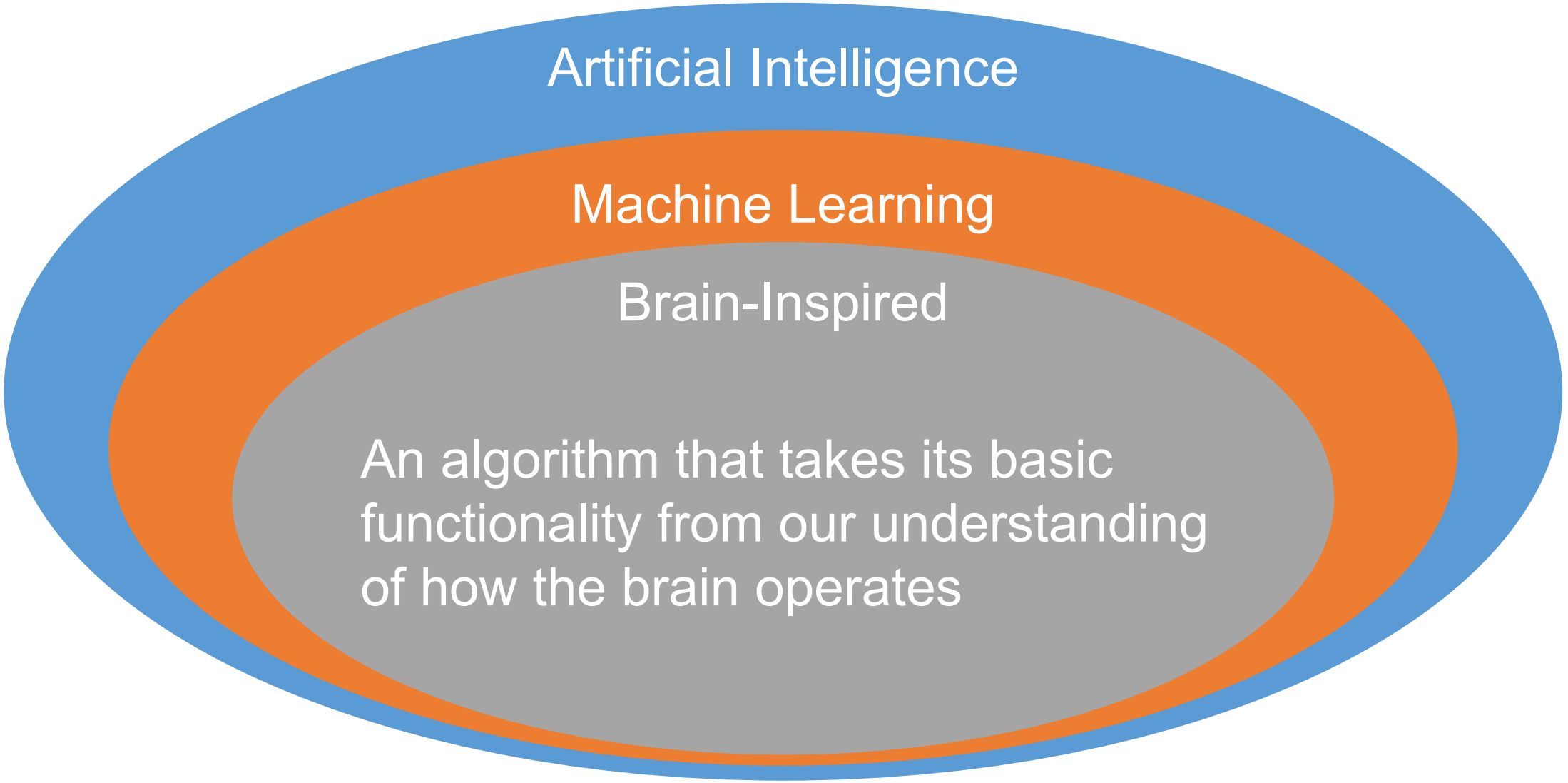
Artificial Intelligence (AI)



Machine Learning (ML)

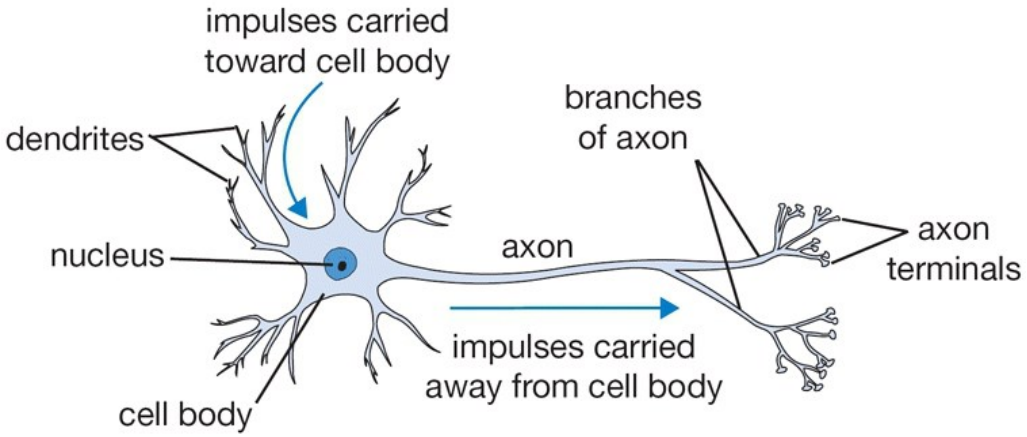


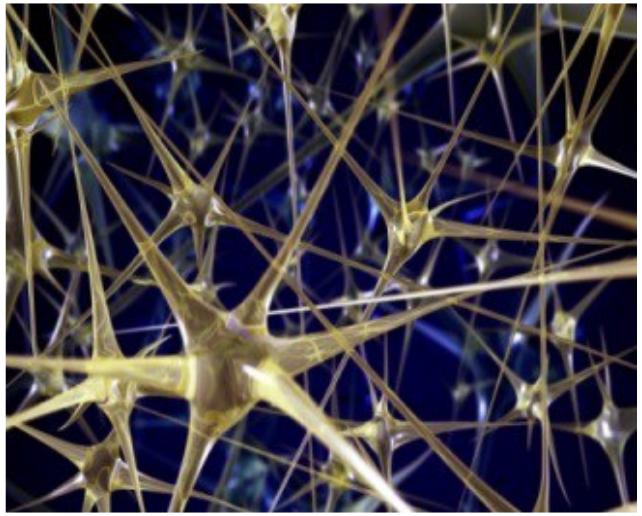
Deep Learning (DL)



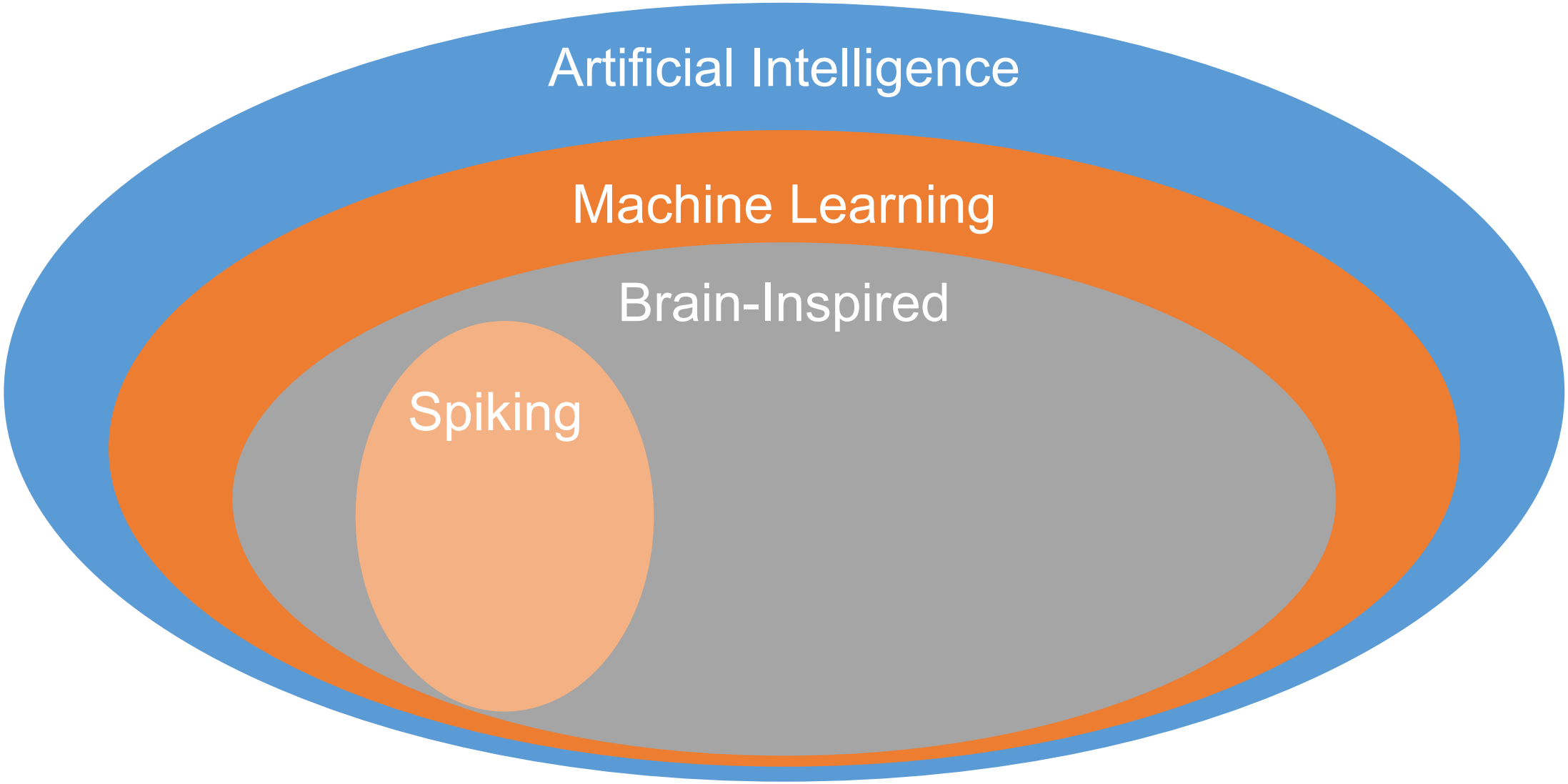
How Does the Brain Work?

- 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



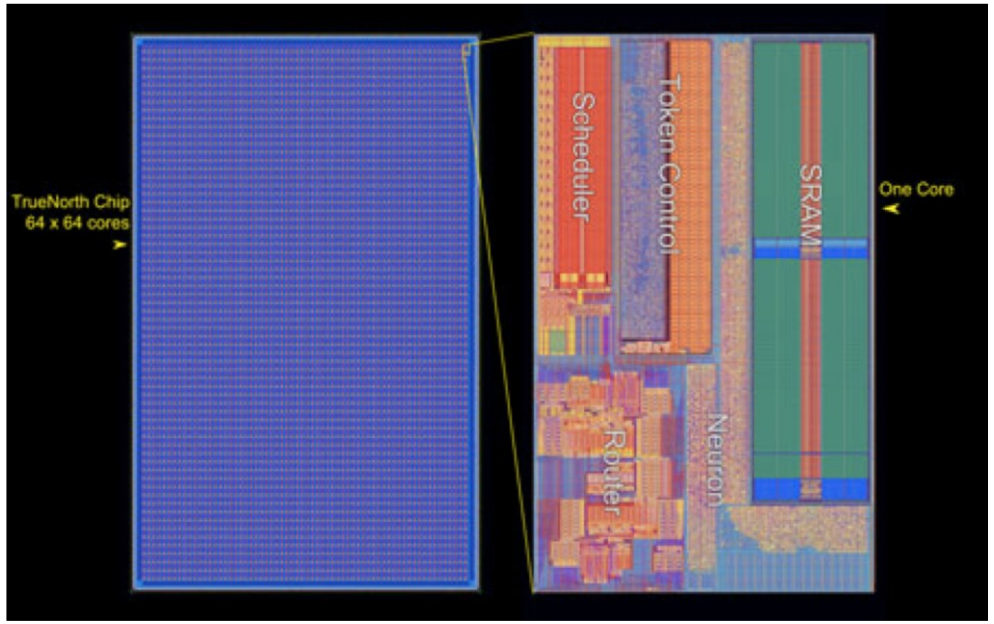


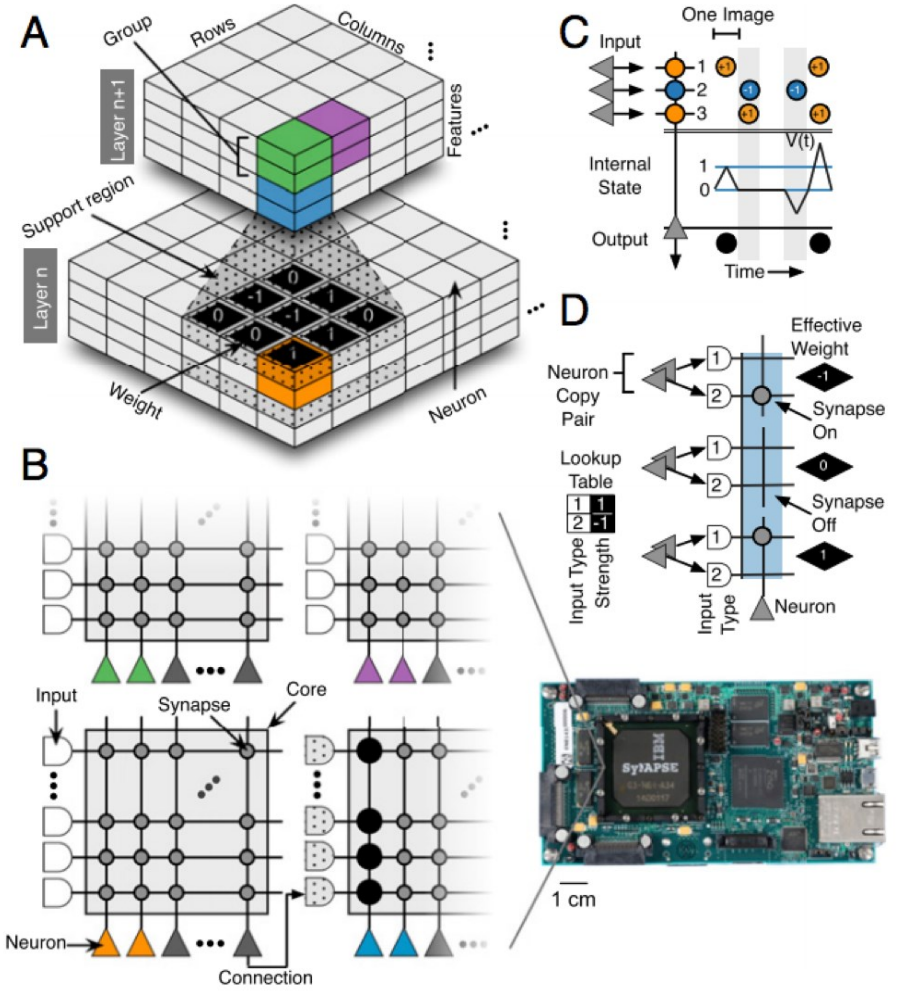
Spiking-based Machine Learning



Spiking Architecture

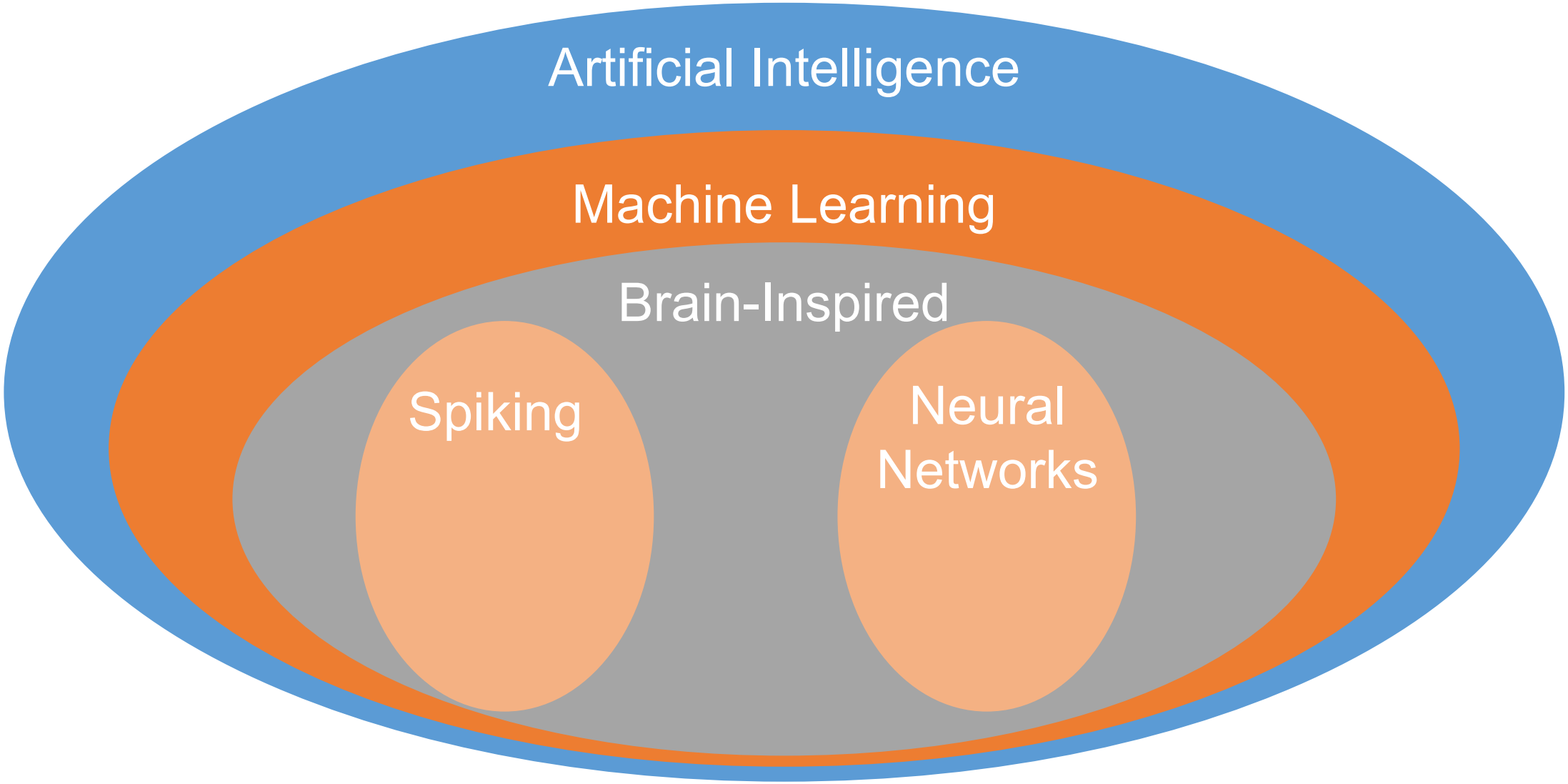
- Brain-inspired
- Integrate and fire
- Example: IBM TrueNorth



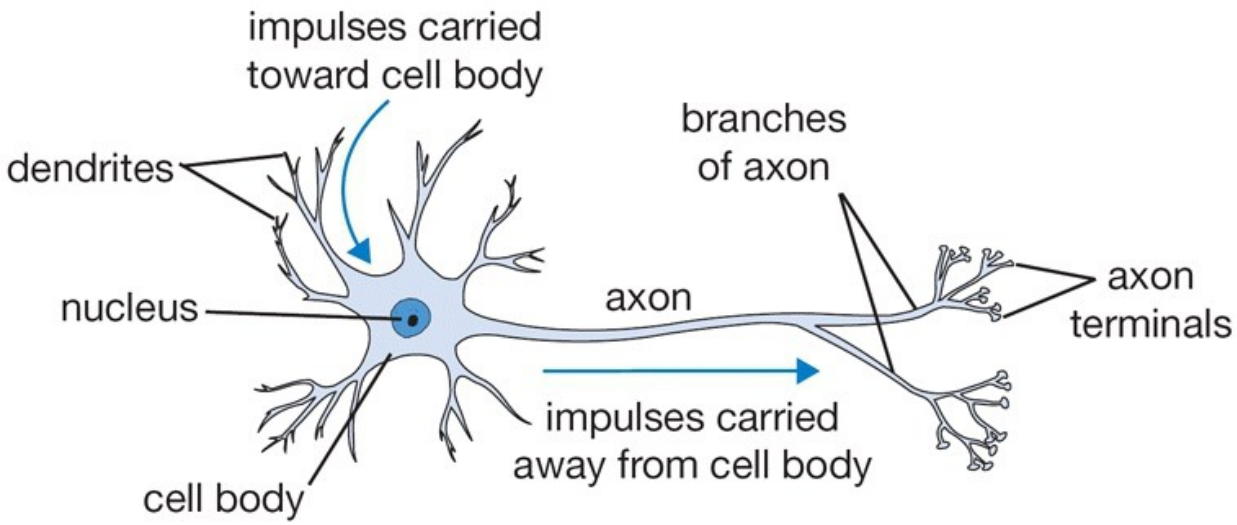


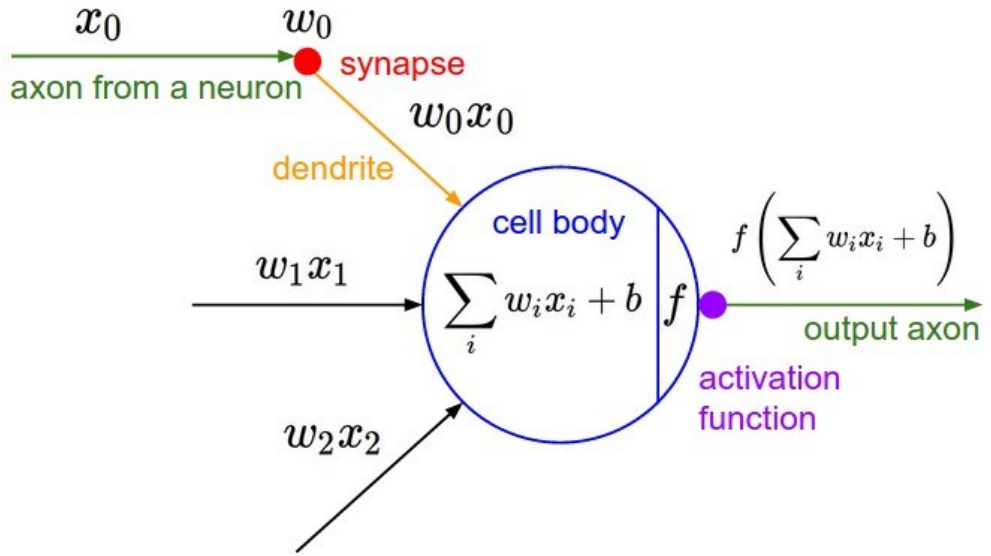
Merolla et al., Science 2014; Esser et al., PNAS 2016

Machine Learning with Neural Networks

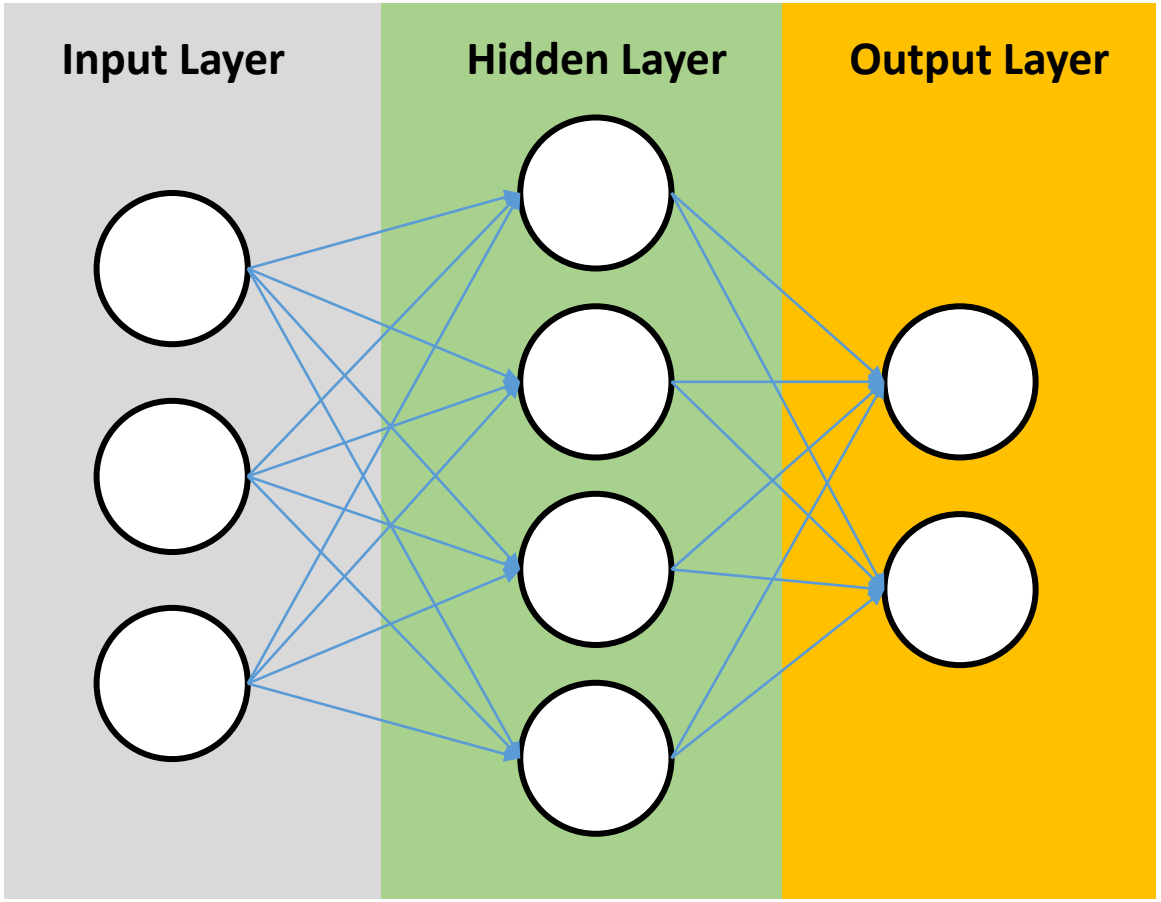


Neural - Weighted Sum

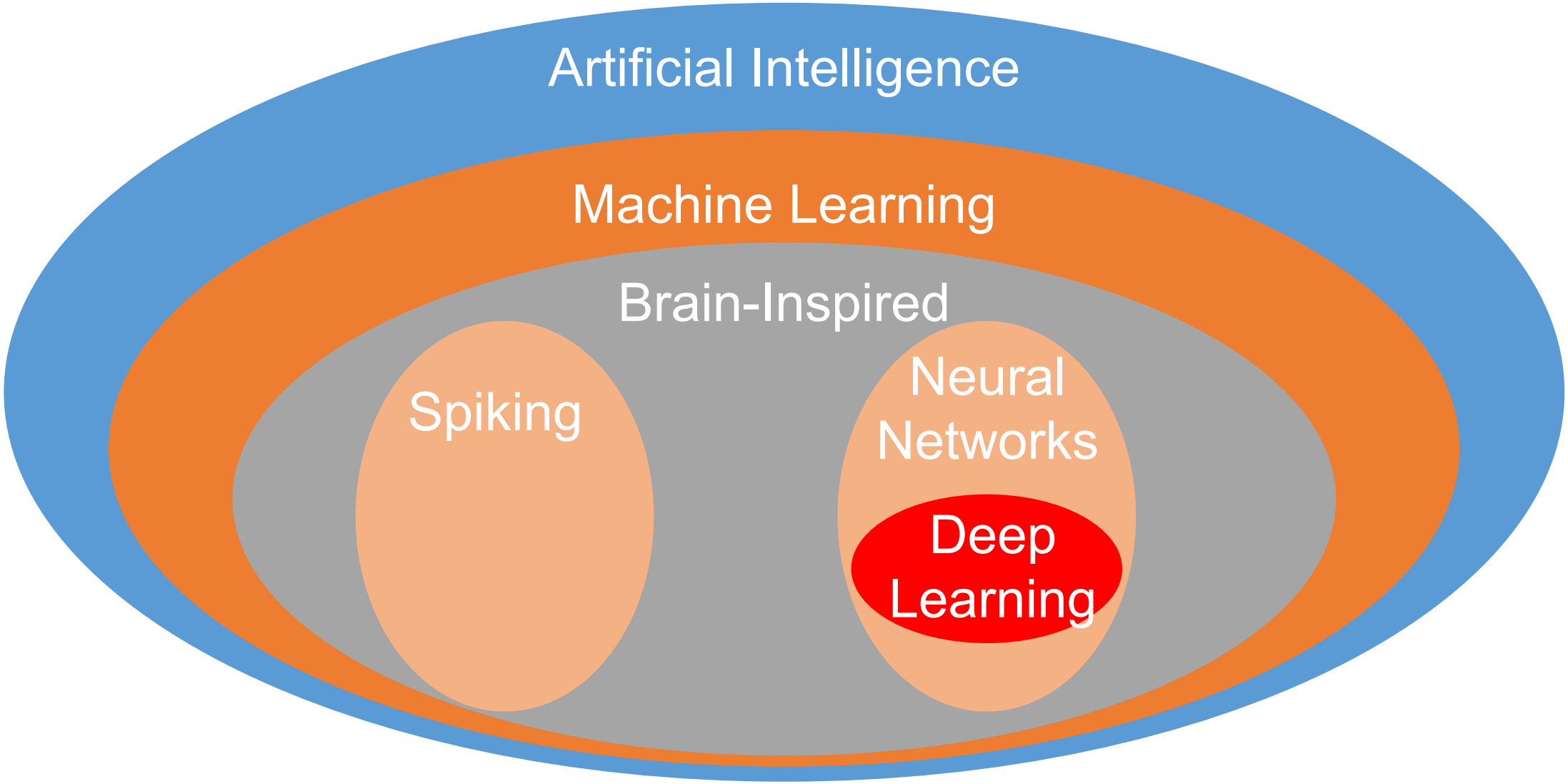




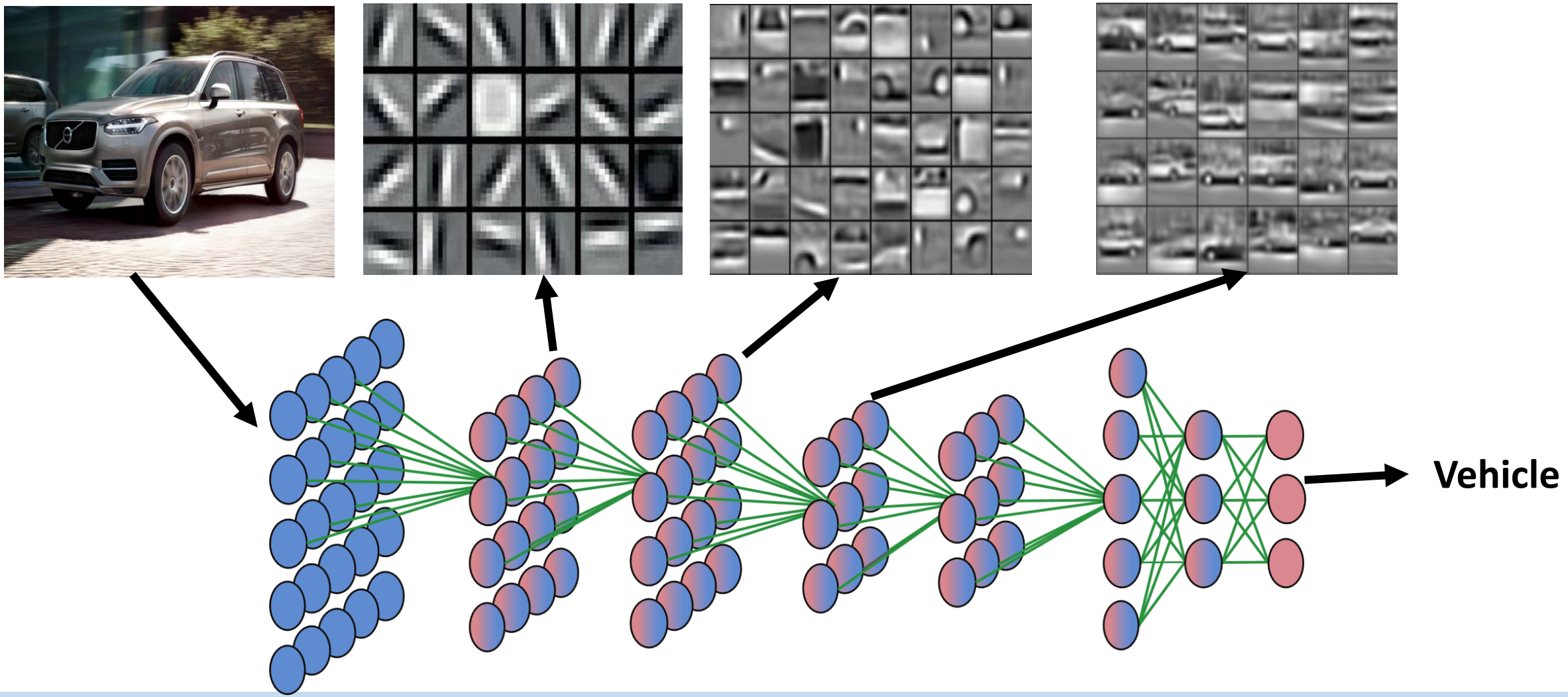
Neural Network - Many Weighted Sums



Machine Learning with Neural Networks

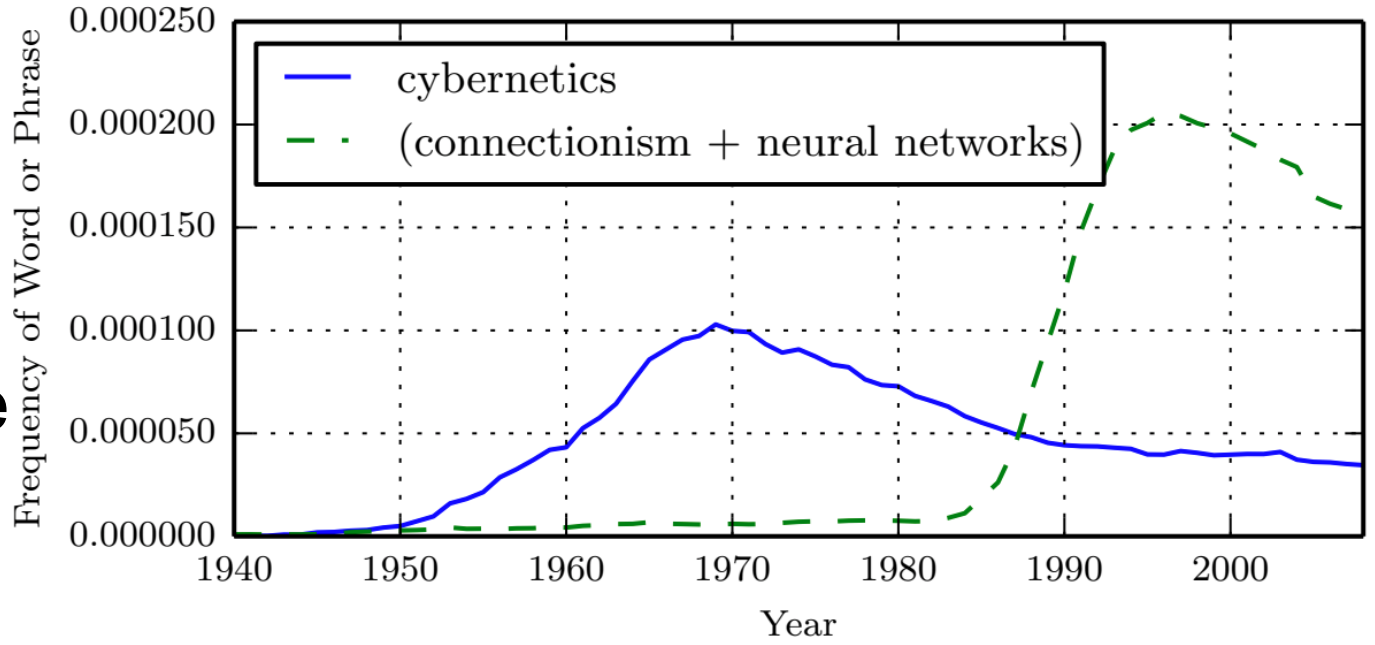


Deep Learning Example



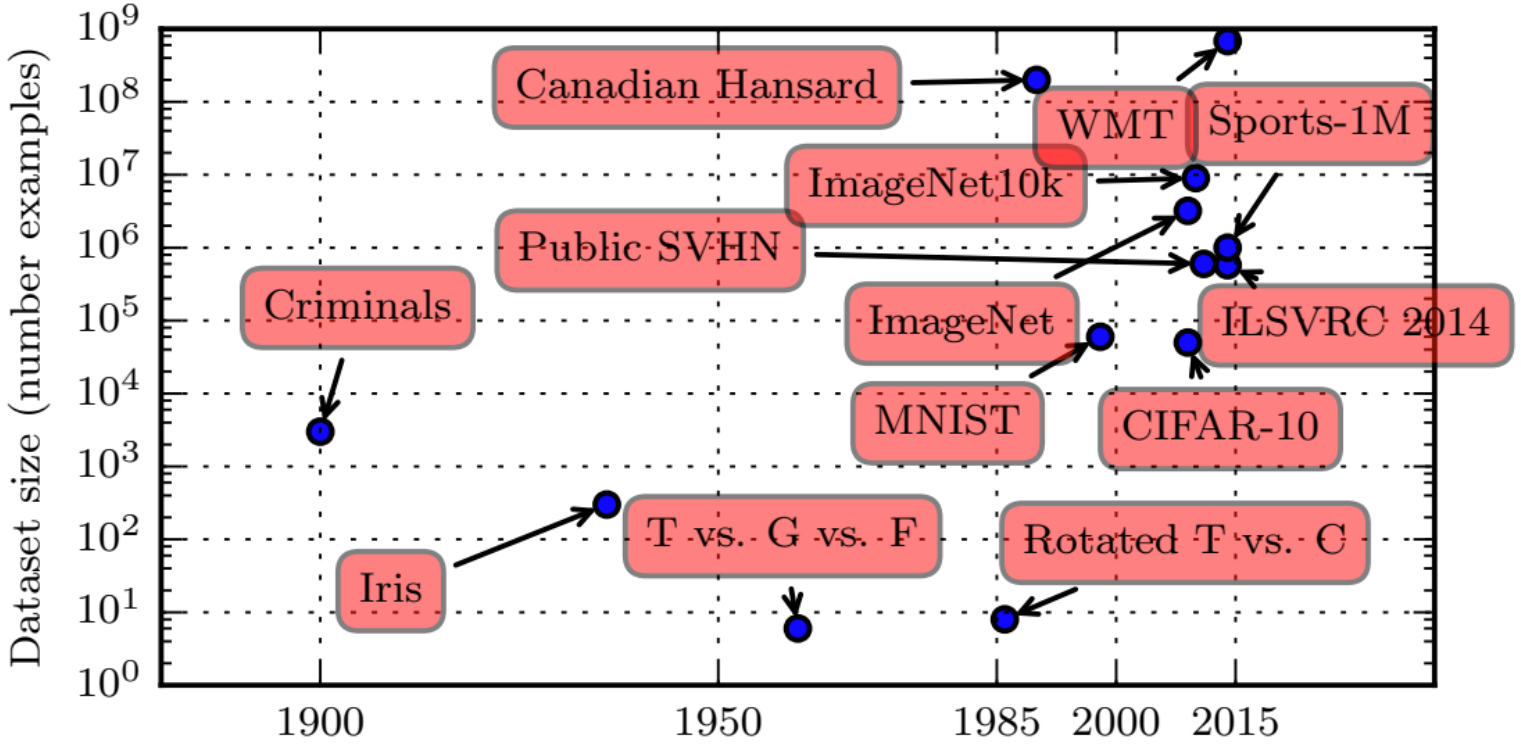
Many Names of Deep Learning

- Three waves of neural networks
  - Cybernetics
    - (1940s – 1960s)
  - Connectionism
    - (1980s – 1990s)
  - Deep learning
    - (2006s – now)
- Diminished role of neuroscience
  - Simply do not have enough information about the brain



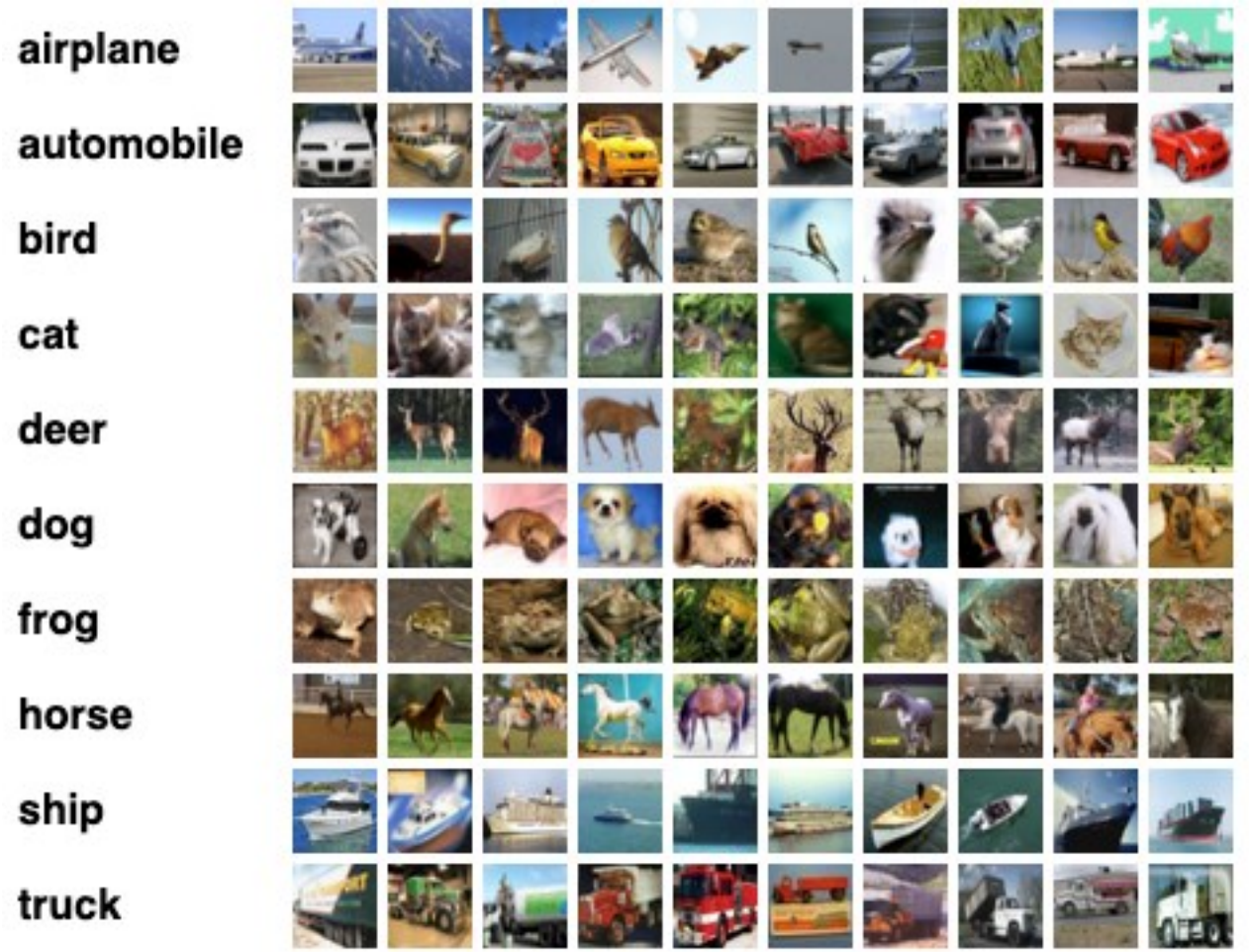
Increasing Dataset Sizes

- The size of datasets has expanded remarkably over time.
- The age of “Big Data” has made machine learning easier.



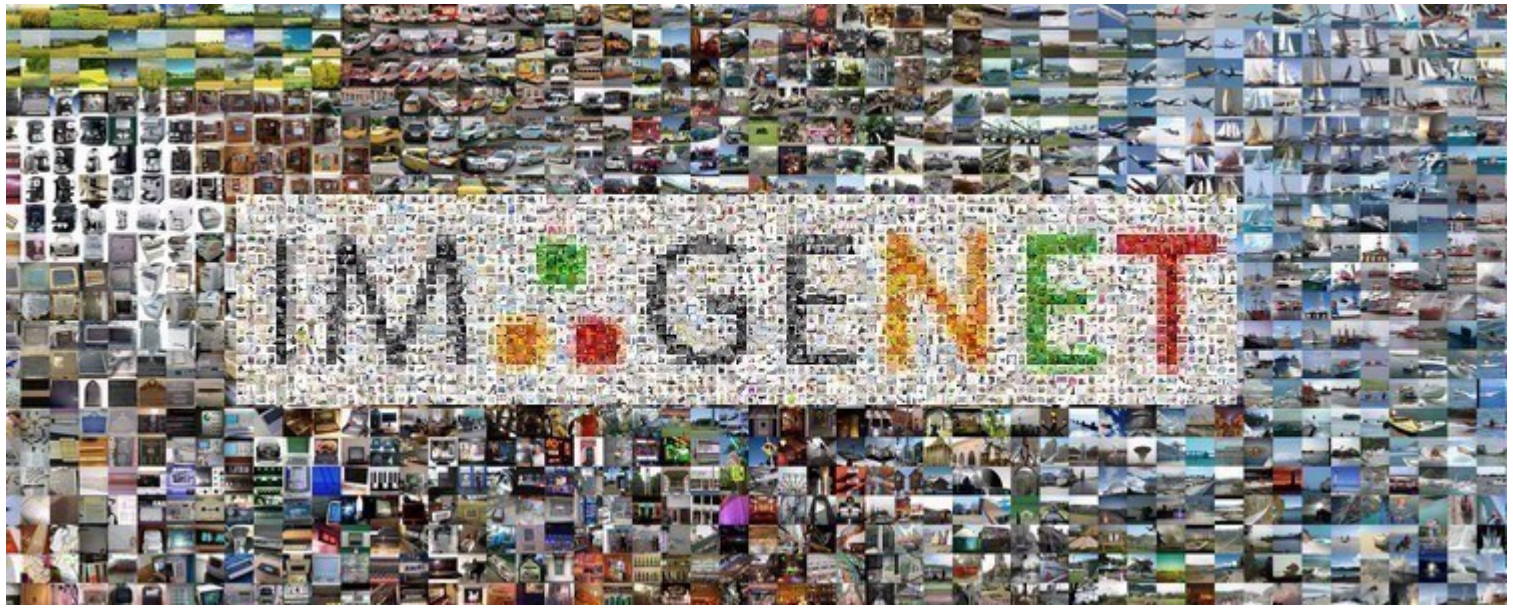
Increasing Dataset Sizes: CIFAR10

- 60,000 32x32 images
  - 50,000 training
  - 10,000 test
- 10 classes
  - 6,000 images/class



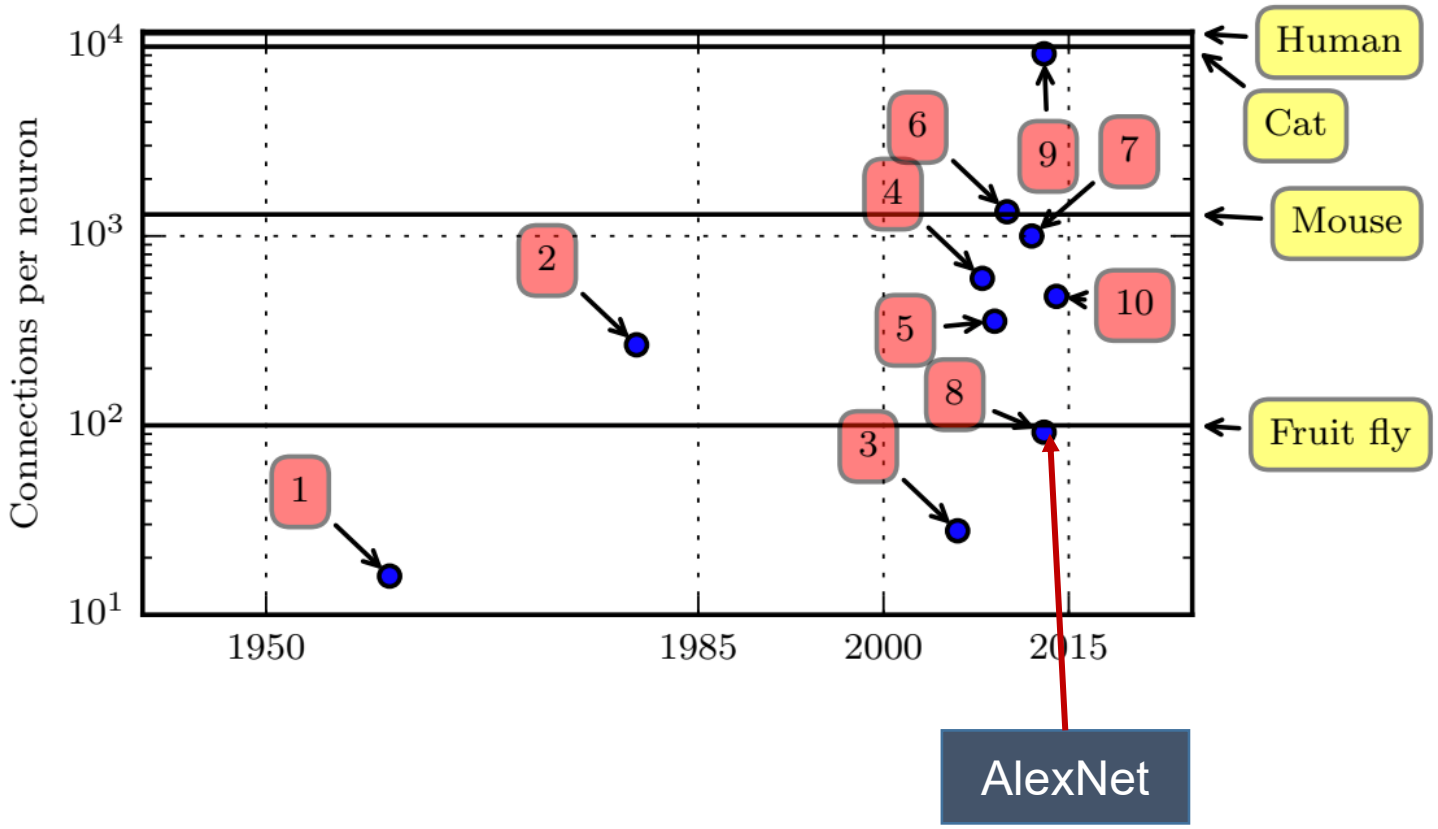
Increasing Dataset Sizes: ImageNet

- Varied in dimensions and resolution images
  - 1,281,167 training
  - 50,000 validation
  - 100,000 test
- 1000 classes



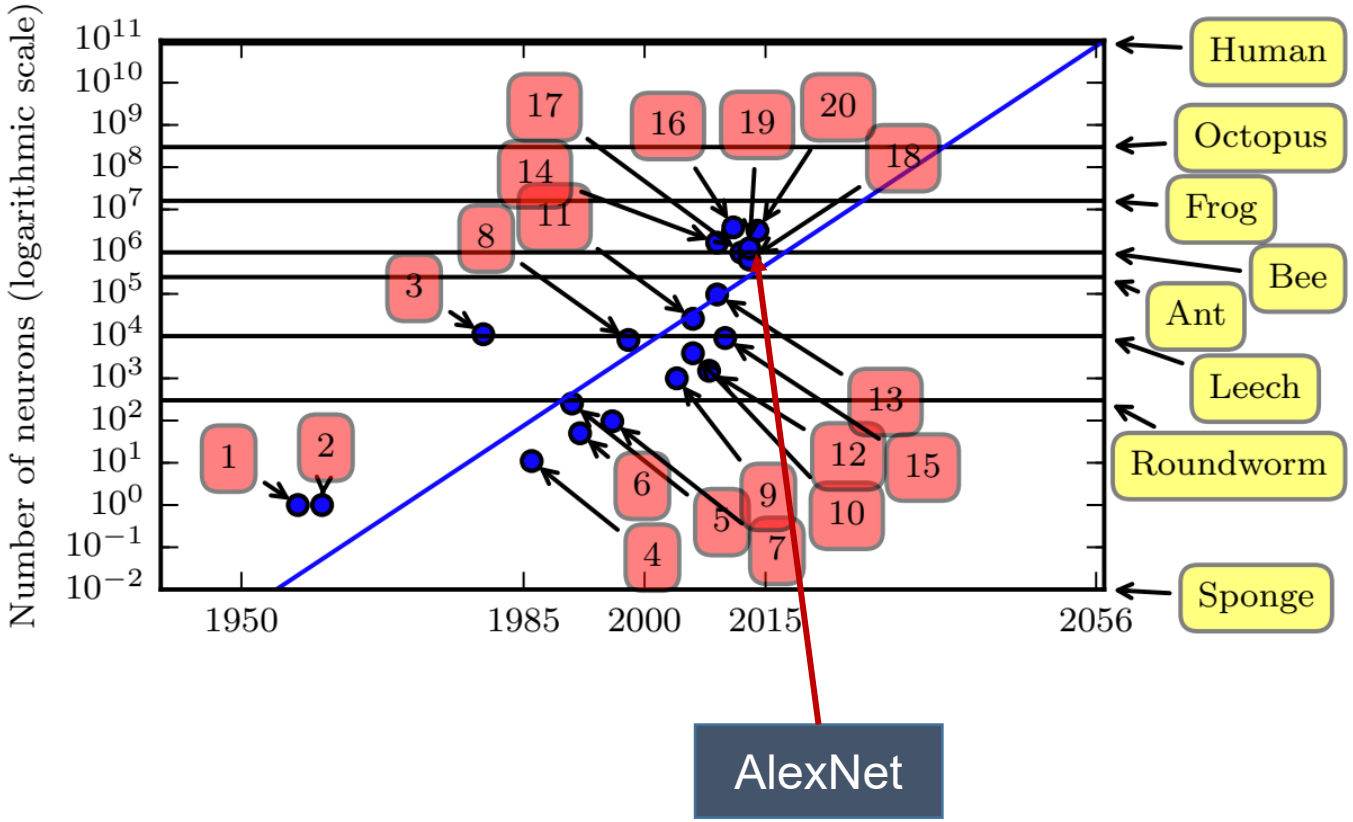
Increasing Model Sizes

- Model size:
  - # of connections / neuron
  - # of neurons
- Largely due to the availability of faster hardware (CPUs and GPUs)
- Expect to continue



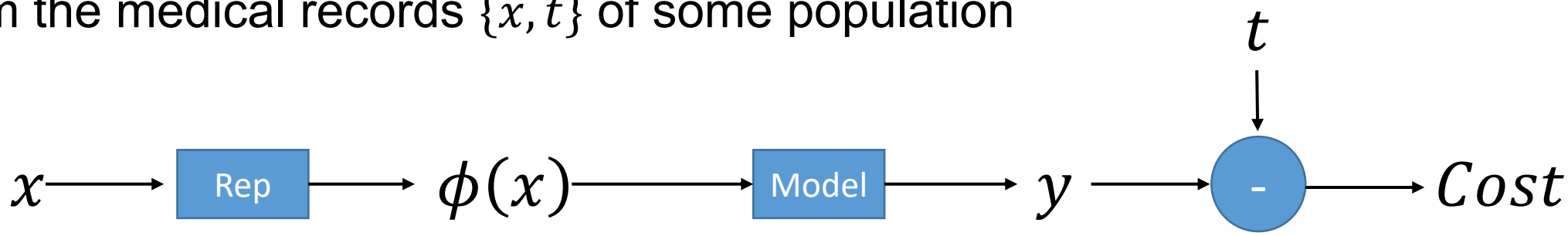
Increasing Model Sizes

- Model size:
  - # of connections / neuron
  - # of neurons
- Largely due to the availability of faster hardware (CPUs and GPUs)
- Expect to continue



Machine Learning

- Acquiring knowledge by extracting **patterns** from **raw data**
- Example: To predict a person's wellness *t* from their MRI scan x by learning patterns from the medical records $\{x, t\}$ of some population



  - x: *MRI scan*
  - $\phi(x)$: *data representation of MRI* scan
  - $y \in (0,1)$: model prediction with parameter w

$$y = f_w(\phi(x)) \triangleq \sigma(w^T\phi(x)), \text{ where } \sigma(x) = \frac{1}{1+e^{-x}}$$

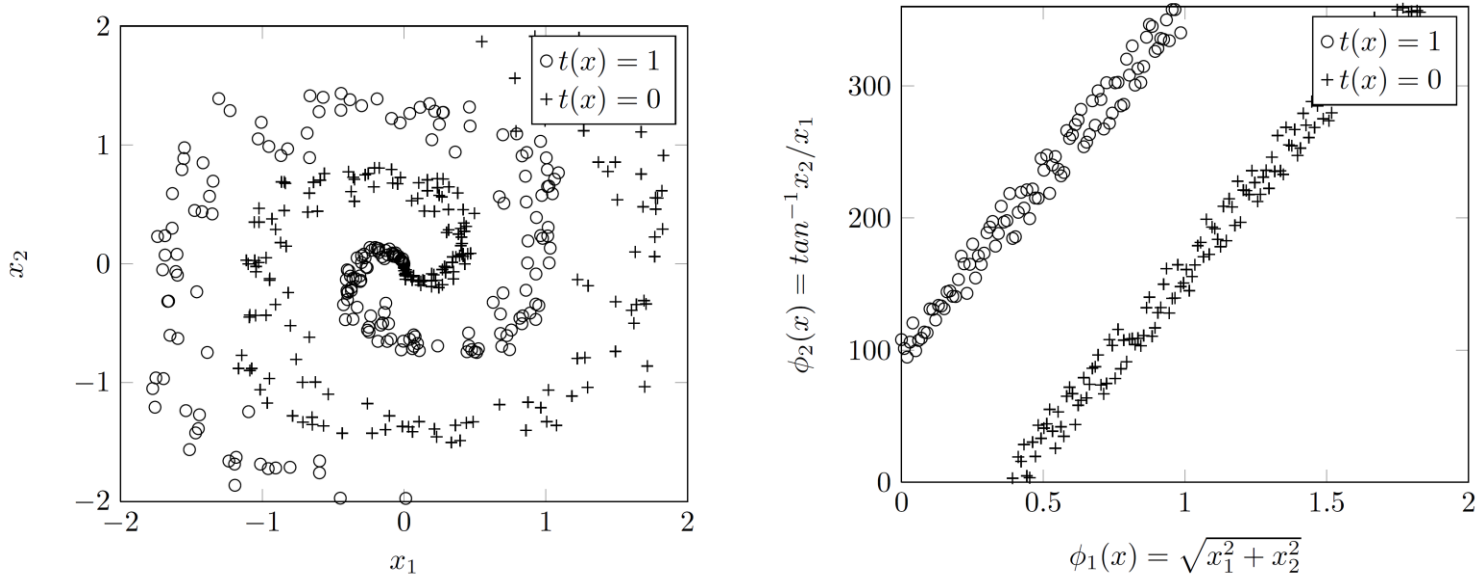
  - $t \in \{0,1\}$: ground-truth result associated with input x
- Cost: some distance between y and t (e.g.$\|y - t\|_2^2$), which is to be minimized w.r.t. w over the $\{x, t\}$ pairs

Machine Learning

- Essentially, we want to find a function $f_w(\phi(x))$to approximate $t(x)$
- In the present example, $f_w(\phi(x))$ bears a probabilistic interpretation of $p(t = 1|x; w)$
- The setting here is termed supervised learning as the ground-truth result t is given for each x

Data Representation - $\phi(x)$

- Data representation can critically determine the prediction performance



- In classic machine learning, hand-designed features are usually used
- For many tasks, it is however difficult to know what features should be used

Deep Learning

- A machine learning approach whose data representation is based on building up a **hierarchy of concepts**, with each concept defined through its relation to simpler concepts
- Using the previous example, this amounts to learning a function of the following form

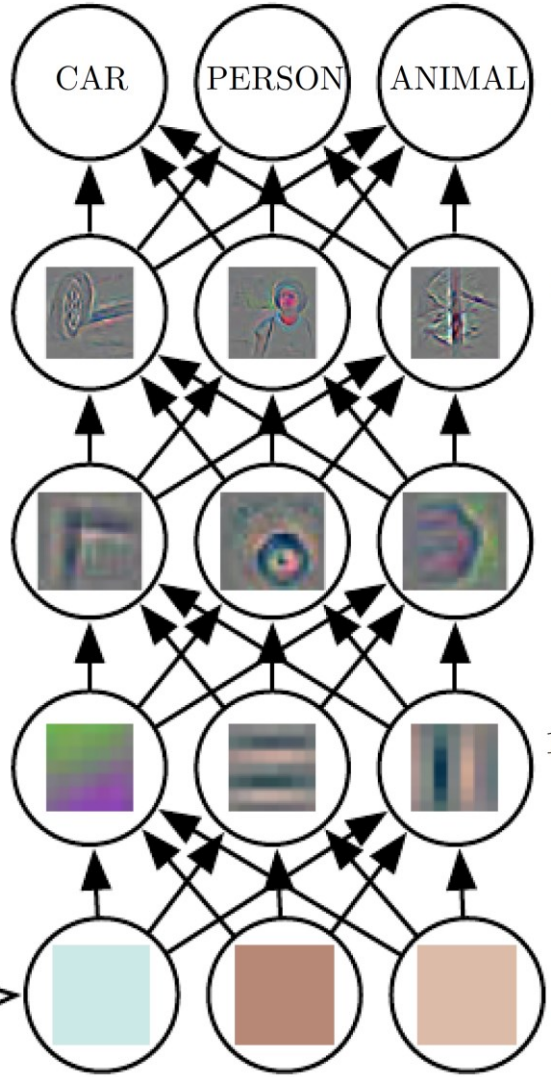
$$f_{w,\theta_n,\theta_{n-1},\ldots,\theta_1}(x) = \sigma(w^T \phi_{\theta_n}(\phi_{\theta_{n-1}}(\ldots \phi_{\theta_1}(x))))$$

where $w, \theta_n, \theta_{n-1}, \ldots, \theta_1$ are model parameters

Hierarchy of concepts/features

- $\phi_\theta(\cdot)'s$ are generally vector-valued functions, e.g. $\phi_\theta(x) = \sigma(\theta_x)$
- Such a deep model allows to construct a complicated function $f(x)$ from nested composition of simpler functions $\phi_\theta(\cdot)'s$

Example: Feedforward Deep Networks



$$\sigma(\phi_{\theta_3}(\phi_{\theta_2}(\phi_{\theta_1}(x))))$$

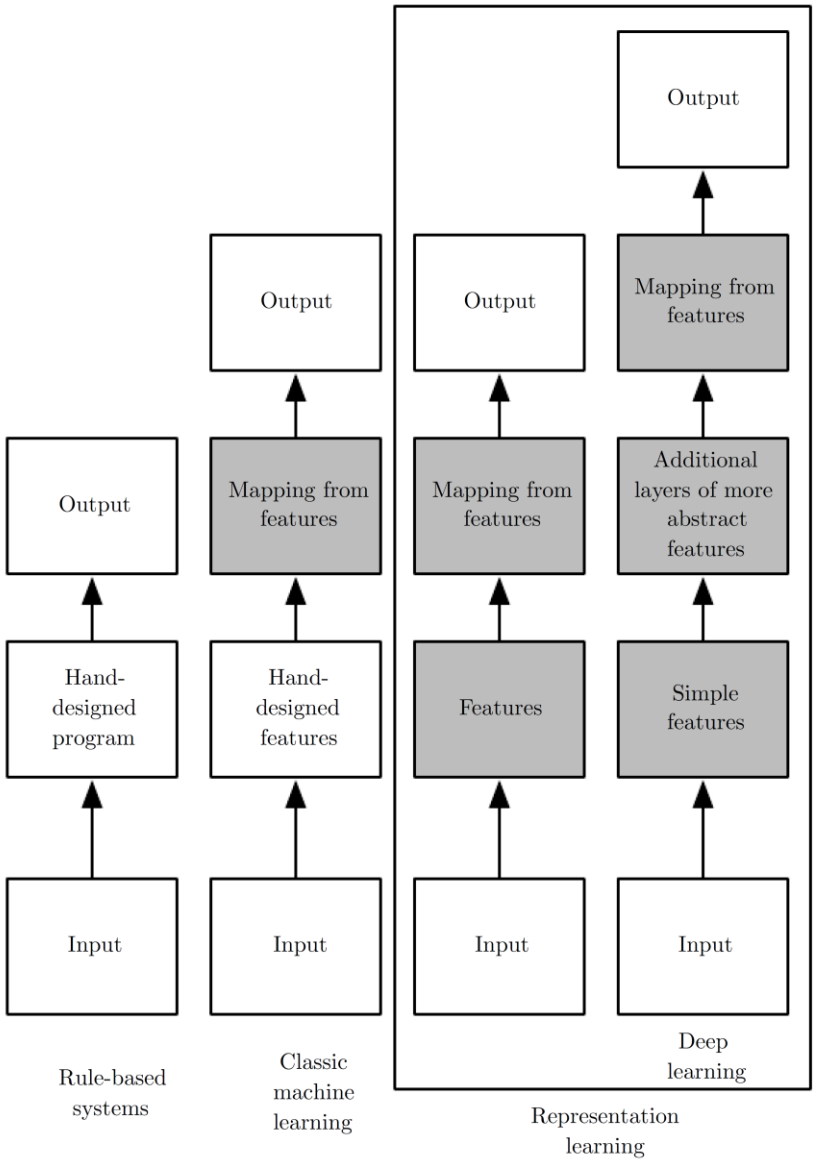
$$\phi_{\theta_3}(\phi_{\theta_2}(\phi_{\theta_1}(x)))$$

$$\phi_{\theta_2}(\phi_{\theta_1}(x))$$

$$\phi_{\theta_1}(x)$$

$$x$$

Different AI Systems



Outline

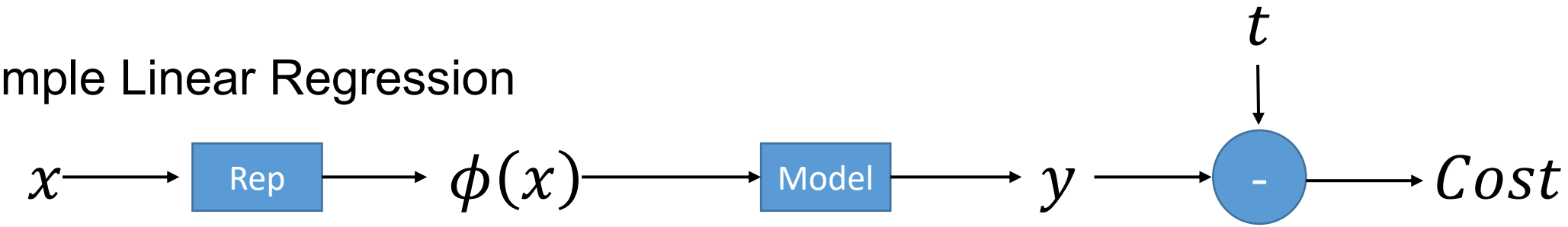
- AI, ML and DL
- ML Basics
- DL Overview

Learning Algorithms

(Mitchell, 1997) A computer program is said to learn from **experience E** with respect to some class of **tasks T** and **performance measure P**, if its performance at tasks in T, as measured by P, improves with experience E

Example - Linear Regression

- Example Linear Regression



  - Task T: To predict y from x by outputting

$$\hat{y} = w^T \phi(x) = \phi(x)^T w$$

  - Experience E: To learn w by minimizing, over a training set $\left(X^{(train)}, y^{(train)}\right)$,

$$MSE^{(train)} = \frac{1}{m^{(train)}} \left\| \hat{y}^{(train)} - y^{(train)} \right\|_2^2$$

where

$$\hat{y}^{(train)} = \Phi^{(train)} w, \Phi^{(train)} = \begin{bmatrix} \phi(x_0^{(train)})^T \\ \phi(x_1^{(train)})^T \\ \vdots \\ \phi(x_{m-1}^{(train)})^T \end{bmatrix}$$

$$y^{(train)} = (y_0^{(train)}, y_1^{(train)}, \ldots, y_{m-1}^{(train)})$$

Example - Linear Regression

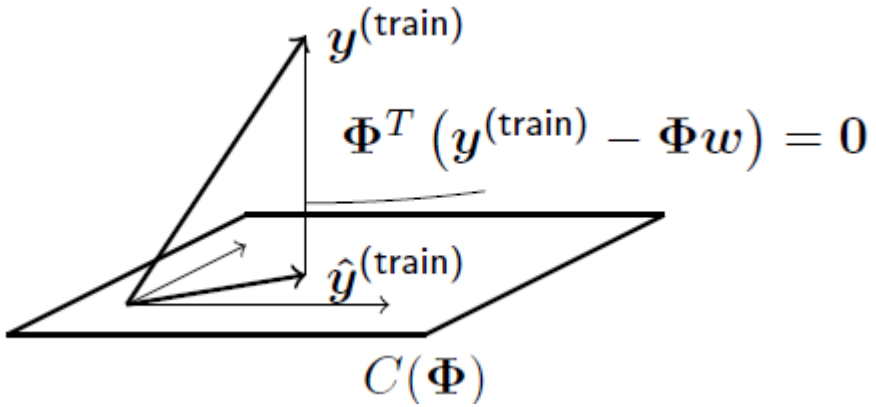
- Performance P: To measure mean squared error on a test set $\left(X^{(test)}, y^{(test)}\right)$, i.e.,

$$MSE^{(test)} = \frac{1}{m^{(test)}} \left\| \hat{y}^{(test)} - y^{(test)} \right\|_2^2$$

- To minimize $MSE^{(train)}$, w can be solved by setting

$$\nabla_w MSE^{(train)} = 0$$

- A geometrical view is to solve $\hat{y}^{(train)}$ as the projection of $y^{(train)}$ onto the column space of $\Phi^{(train)}$



- We then have

$$w = (\Phi^{(train)^T} \Phi^{(train)})^{-1} \Phi^{(train)^T} y^{train}$$

Example - Linear Regression

- The present model can be extended to include a bias term b

$$\hat{y} = w^t \phi(x) + b = \tilde{w}^T \tilde{\phi}(x),$$

with

$$\tilde{w} = \begin{bmatrix} w \\ b \end{bmatrix}, \tilde{\phi}(x) = \begin{bmatrix} \phi(x) \\ 1 \end{bmatrix}$$

- Prediction with a polynomial of degree 2

$$\tilde{y} = w_2 x^2 + w_1 x^1 + b = \tilde{w}^T \tilde{\phi}(x)$$

where

$$\tilde{w} = \begin{bmatrix} w_2 \\ w_1 \\ b \end{bmatrix}, \tilde{\phi}(x) = \begin{bmatrix} \phi_2(x) \\ \phi_1(x) \\ 1 \end{bmatrix} = \begin{bmatrix} x^2 \\ x^1 \\ 1 \end{bmatrix}$$

Dataset

- ML tasks are usually described in terms of how the ML system should process an example.
- A dataset is a collection of many examples.
- ML algorithms can be broadly categorized as **supervised** and **unsupervised** by what kind of dataset they process.
  - Supervised: each example of the dataset is associated with a label or target
    - E.g., Classification, regression
  - Unsupervised: experience dataset without labels
    - E.g., Clustering
  - Reinforcement: Not a fixed dataset, interact with an environment
- Partition the dataset into:
  - Training set: where the model was trained
  - Test set: where the trained model was tested

Generalization

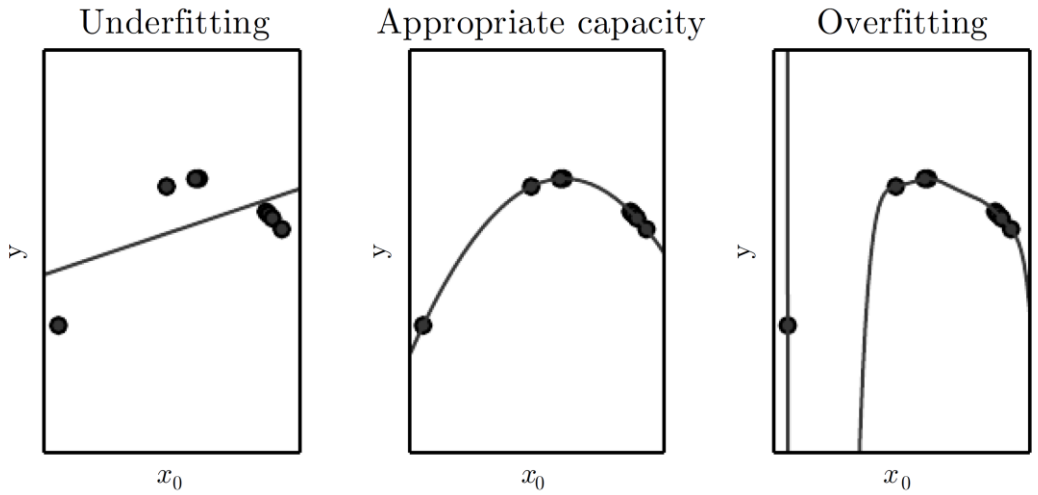
- The model's ability to perform well on **new**, previously **unseen** inputs
- Generalization error is defined to be the expected **value** of the **error** on a **new input** and is typically estimated by measuring the performance on a **test set** collected separately from the **training set**
- Examples (x, y) in the training and test sets are assumed to be drawn independently from the same distribution, $p_{data}(x, y)$
- **Bayes error**
  - Minimum generalization error achieved by an oracle model having knowledge of $p_{data}(x, y)$
  - E.g.: if $y = w^T \phi(x) + \varepsilon$ and ε is Gaussian noise independent of x, Bayes error = $Var(\varepsilon)$ with MSE measure

Training Error and Test Error

- Pitfall
  - At first glance, the expected test error should be the same as the expected training error for a given model, because the data in these sets are drawn from the same distribution
- In practice, we sample the training set, use it to train the model, and then sample the test set to measure test error
- Generally, test error ≥ training error

Underfitting vs. Overfitting

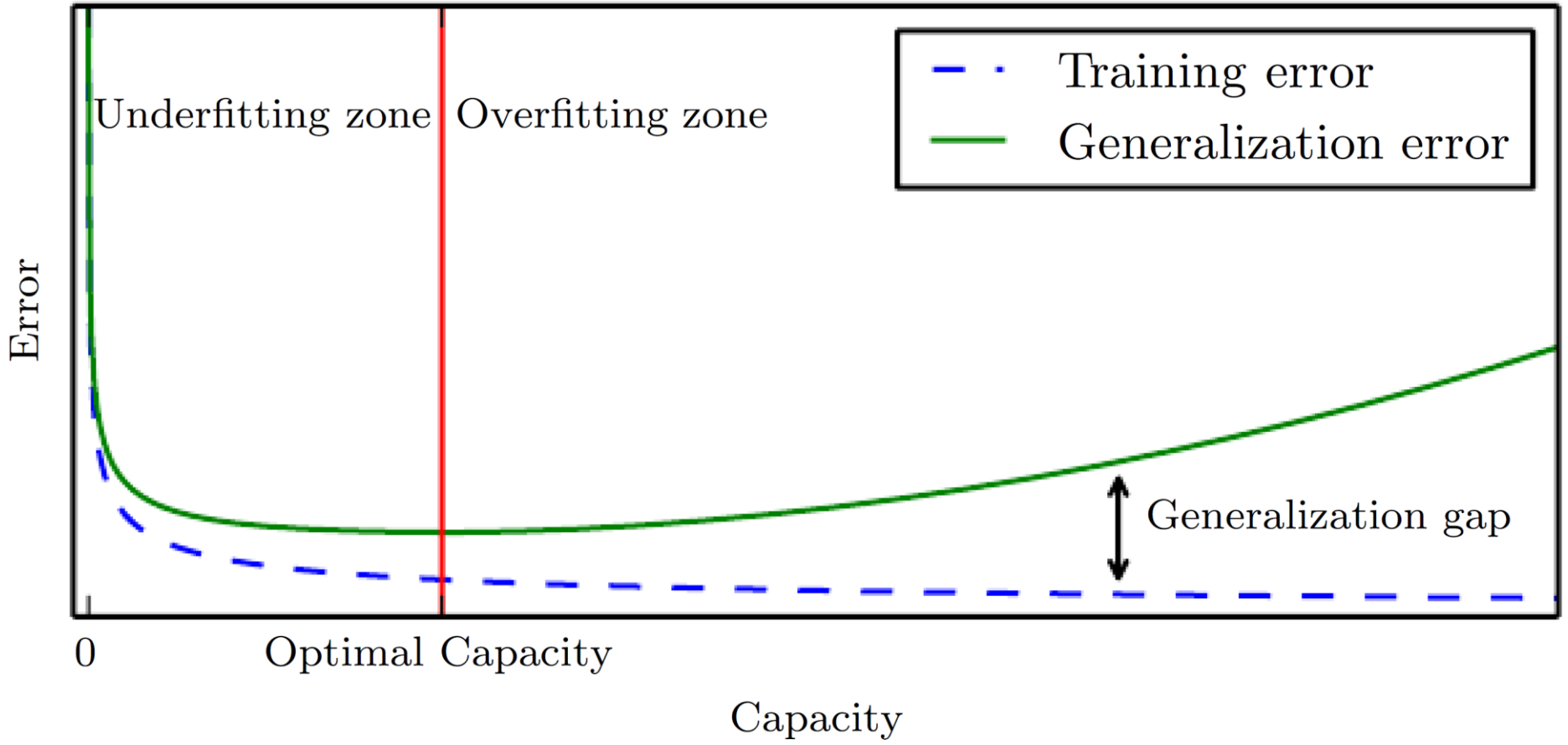
- Two objectives to achieve in designing a model
  1. Make training error small to avoid underfitting
  2. Make gap between training and test error small to avoid overfitting
- Trade-off can be made by altering the **model capacity**, which refers broadly to a model's ability to fit a wide variety of functions
- Example: Fitting a polynomial model to quadratic data



$$\hat{y} = \sum_{i=1}^{d} w_i x_0^i + b$$

Capacity and Error

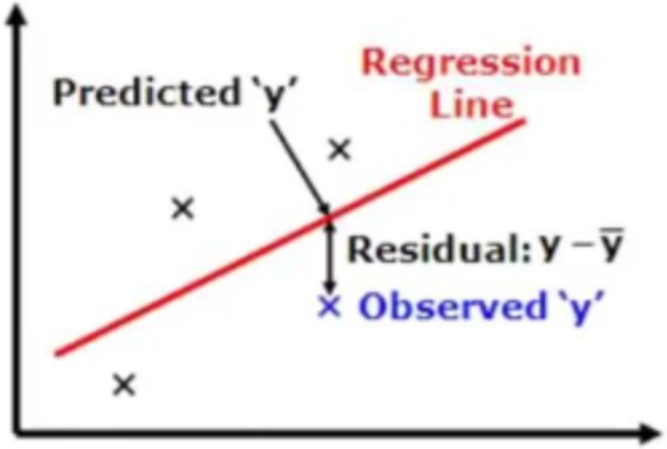
- Typical relation between capacity and error



Cost Function

- Measures the performance of a ML model for given data
- Quantifies the error between predicted and expected values in the **training** set.
- Example

$$Cost\ Function = MSE_{train} = \frac{1}{m}\sum_{i=0}^{m}(\hat{y}_i - y_i)^2$$

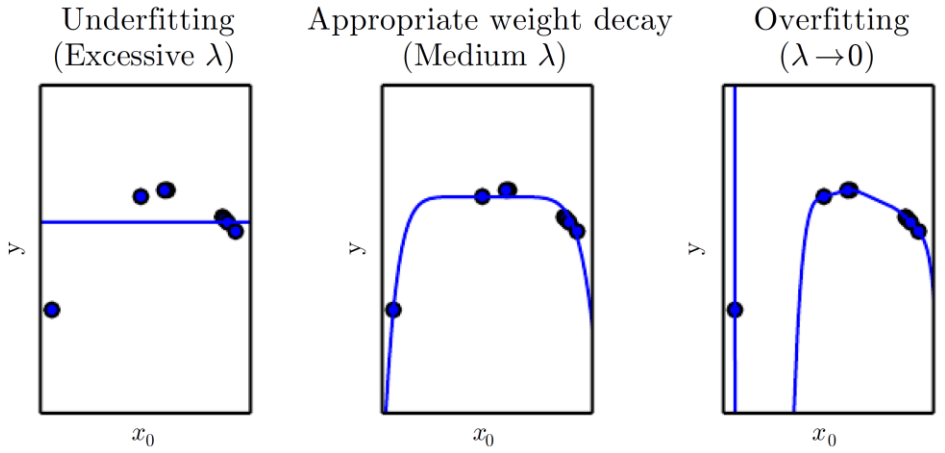


Regularization

- Modification made to the learning algorithm to reduce generalization error (usually at the cost of higher training error)
  - Not reducing the training error
- Example: To include **weight decay** in the training criterion

$$J(w) = MSE^{(train)} + \lambda w^T w$$

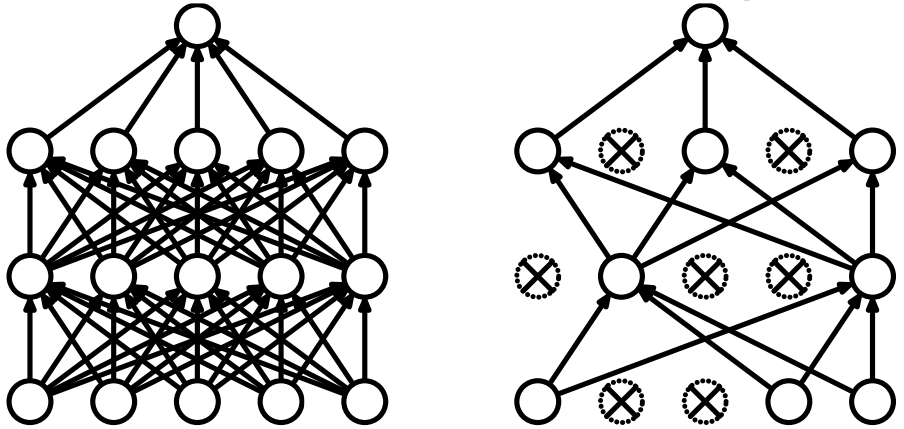
where λ controls preference for small w and is determined a priori



A degree-9 polynomial model fitted to quadratic data

Regularization - Examples

- Weight decay (L2/1 regularization)
  - Expressing preferences of smaller weights
  - $Cost\ Function = MSE^{(train)} + \lambda \sum_i w_i^2\ (L2)$
  - $Cost\ Function = MSE^{(train)} + \lambda \sum_i |w_i|\ (L1)$
- Dropout
  - Temporally remove nodes from network
  - Train a large ensemble of models that share parameters



Standard Neural Net

After applying dropout.

Estimators

- Point estimation: To provide a single estimate of some quantity from observing independent and identically distributed (i.i.d.) samples
  - Consider m i.i.d samples $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ drawn from a Bernoulli distribution with mean θ

$$P\left(x^{(i)}; \theta\right) = \theta^{x^i}(1-\theta)^{1-x^{(i)}}, x^{(i)} = \{1,0\}$$

  - The **sample mean** can be used to give a point estimate of θ

$$\hat{\theta}_m = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

Estimators

- A point estimator $\hat{\theta}_m$ of a parameter θ is any function of the observed samples $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

$$\hat{\theta}_m = g(x^{(1)}, x^{(2)}, \dots, x^{(m)})$$

  - θ is fixed but unknown from the **frequentist** viewpoint
  - $x^{(1)}, x^{(2)}, \dots, x^{(m)}$ are seen samples of a random variable
  - As a result, $\hat{\theta}_m$ is a random variable

Bias

- The bias of the estimator $\hat{\theta}_m$ is defined as

$$bias(\hat{\theta}_m) = E(\hat{\theta}_m) - \theta$$

  where the expectation E(·) is taken w.r.t. $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$
  - $\hat{\theta}_m$ is **unbiased** if $bias(\hat{\theta}_m) = 0$
  - $\hat{\theta}_m$is **asymptotically unbiased** $if \lim_{m \to \infty} bias(\hat{\theta}_m) = 0$
- In the Bernoulli example, the sample mean is an unbiased estimator

$$E\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right] = \frac{1}{m}\sum_{i=1}^{m} E\left[x^{(i)}\right] = \theta$$

Consistency

- An estimator $\hat{\theta}_m$ is said to be consistent in probability if

$$\lim_{m \to \infty} P(\left|\hat{\theta}_m - \theta\right| > \varepsilon) = 0, \varepsilon > 0$$

- Consistency ensures that the bias of the estimator diminishes as the number of data samples grows
- In the Bernoulli example, the sample mean is consistent

Estimators for Gaussian Distribution

- Gaussian probability density function

$$N\left(x^{(i)};\mu,\sigma^2\right)=\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{1}{2}\frac{(x^{(i)}-\mu)^2}{\sigma^2}\right)$$

- Sample mean (unbiased)

$$\hat{\mu}_m=\frac{1}{m}\sum_{i=1}^{m}x^{(i)}$$

- Sample variance (asymptotically unbiased)

$$\hat{\sigma}_m^2=\frac{1}{m}\sum_{i=1}^{m}(x^{(i)}-\hat{\mu}_m)^2$$

  - It can be shown that $E[\hat{\sigma}_m^2]=(m-1)\sigma^2/m$
  - Unbiased sample variance $\hat{\sigma}_m^2=m\hat{\sigma}_m^2/(m-1)$

Variance of the Estimator

- Variance of the estimator indicates how much the estimator varies as a function of the samples
  - Its squared root is called standard error
- Example: Standard error of the sample mean $\hat{\mu}_m$

$$SE(\hat{\mu}_m) = \sqrt{Var\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right]} = \frac{\sigma}{\sqrt{m}}$$

  where σ is usually estimated by $\sqrt{\hat{\sigma}_m^2}$
- By the central limit theorem,

$$\frac{\hat{\mu}_m - 0}{SE(\hat{\mu}_m)} \sim N(0,1)$$

- The 95 percent confidence interval can thus be derived as

$$(\hat{\mu}_m - 1.96SE(\hat{\mu}_m), \hat{\mu}_m + 1.96SE(\hat{\mu}_m))$$

Variance of the Estimator

In experiments, it is common to say algorithm A performs better than B if its 95 percent upper bound of the test error is smaller than the lower bound of B's test error

From Linear to Nonlinear

- To extend linear models to represent nonlinear function of x:

$$y = w^T x + b \rightarrow y = w^T \phi(x) + b$$

  - To use kernel functions such as radial basis functions (RBFs), e.g.:

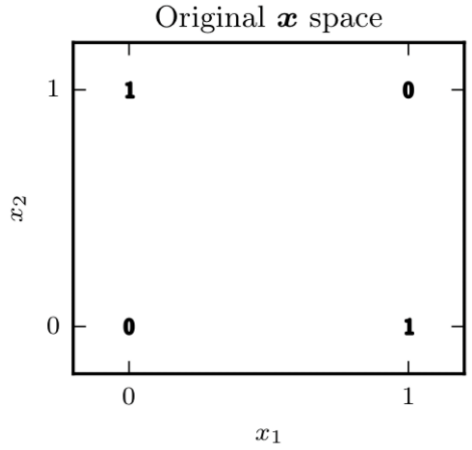
$$\phi(x) = e^{-(\epsilon \|x - x_i\|)^2}$$

  - Manually engineer ϕ, that is, features in computer vision, speech recognition, etc.
  - To learn ϕ from data:

$$y = f(x; \theta, w) = \phi(x; \theta)^T w$$

A Simple Example: Learning XOR

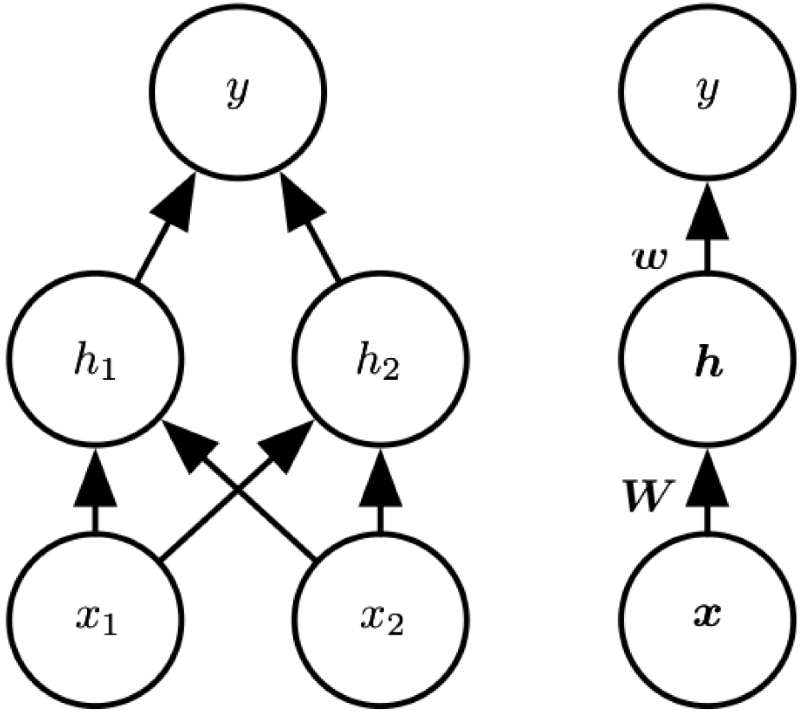
- Data: $\chi = \{[0,0]^T, [0,1]^T, [1,0]^T, [1,1]^T\}$
- Target function: $\mathrm{y} = f^*(\chi) = \{0,1,1,0\}$
- Linear model: $y = f(x; \theta = \{w, b\}) = x^T w + b$
- MSE loss function: $J(\theta) = \frac{1}{4}\sum_{x \in \chi}(f^*(x) - f(x,\theta))^2$
- Linear model is **NOT** able to represent XOR function



A Simple Example: Learning XOR

- Use one hidden layer containing two hidden units to learn ϕ

$$y = f(x; W, c, w, b) = f^{(2)}(f^{(1)}(x))$$



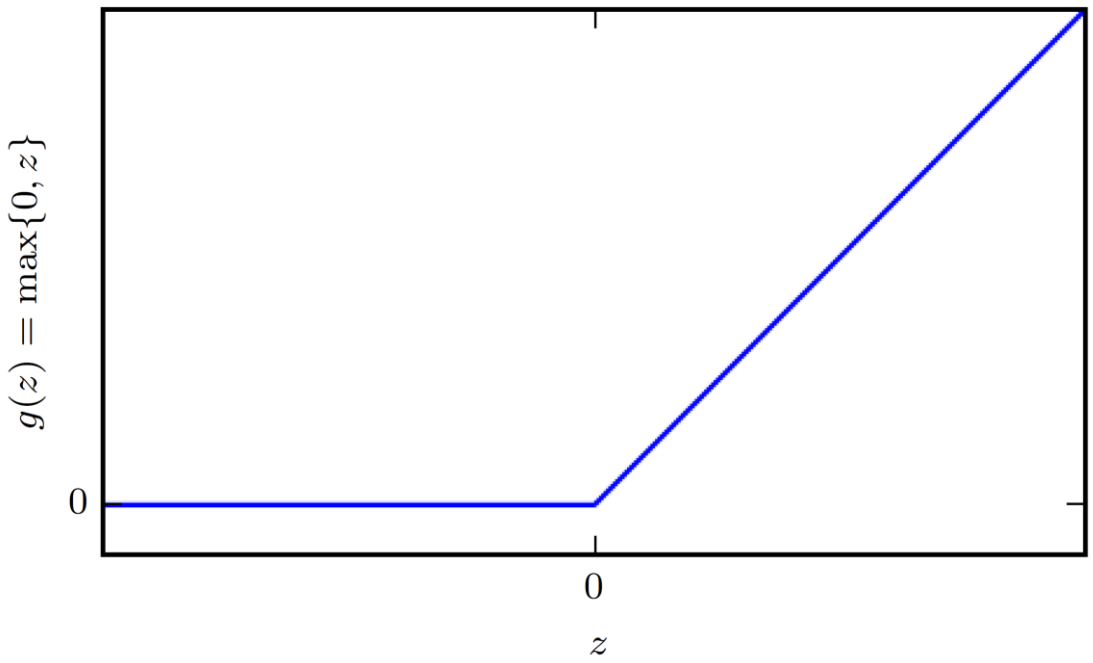
$$y = f^{(2)}(h; w, b)$$

$$\phi(x) = h = f^{(1)}(x; W, c)$$

A Simple Example: Learning XOR

- **Let there be nonlinearity!**
- ReLU: rectified linear unit: $g(x) = \max\{0, z\}$
- ReLU is applied element-wise to **h**:

$$h_i = g(x^T W_{:i} + c_i)$$



A Simple Example: Learning XOR

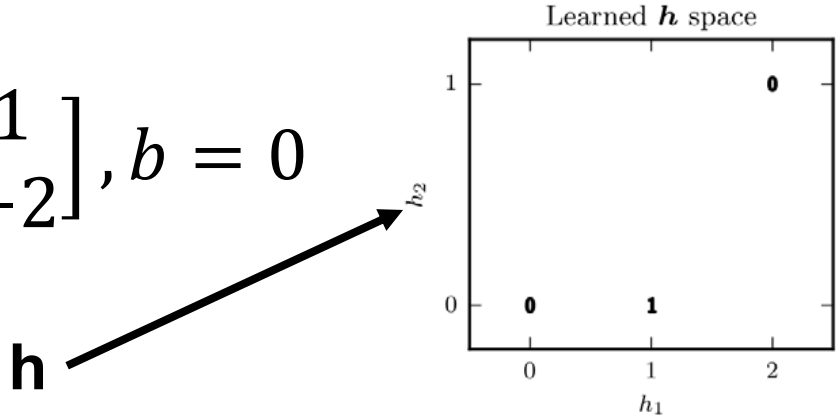
- Complete neural network model:

$$y = f(x; W, c, w, b) = f^{(2)}\left(f^{(1)}(x)\right) = w^T \max\{0, W^T x + c\} + b$$

- Obtain model parameters after training

$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, c = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, w = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$$

- Run the network



$$x = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \xrightarrow{W^T x + c} \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix} \xrightarrow{ReLU} \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix} \xrightarrow{w^T h + b} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Optimization

- Learning algorithms often seek to minimize/maximize some objective function w.r.t. the model parameter, e.g.

$$arg\min_{w} J(w) = -E_{x,y\sim\hat{p}_{data}}[p_{model}(y|x)]$$

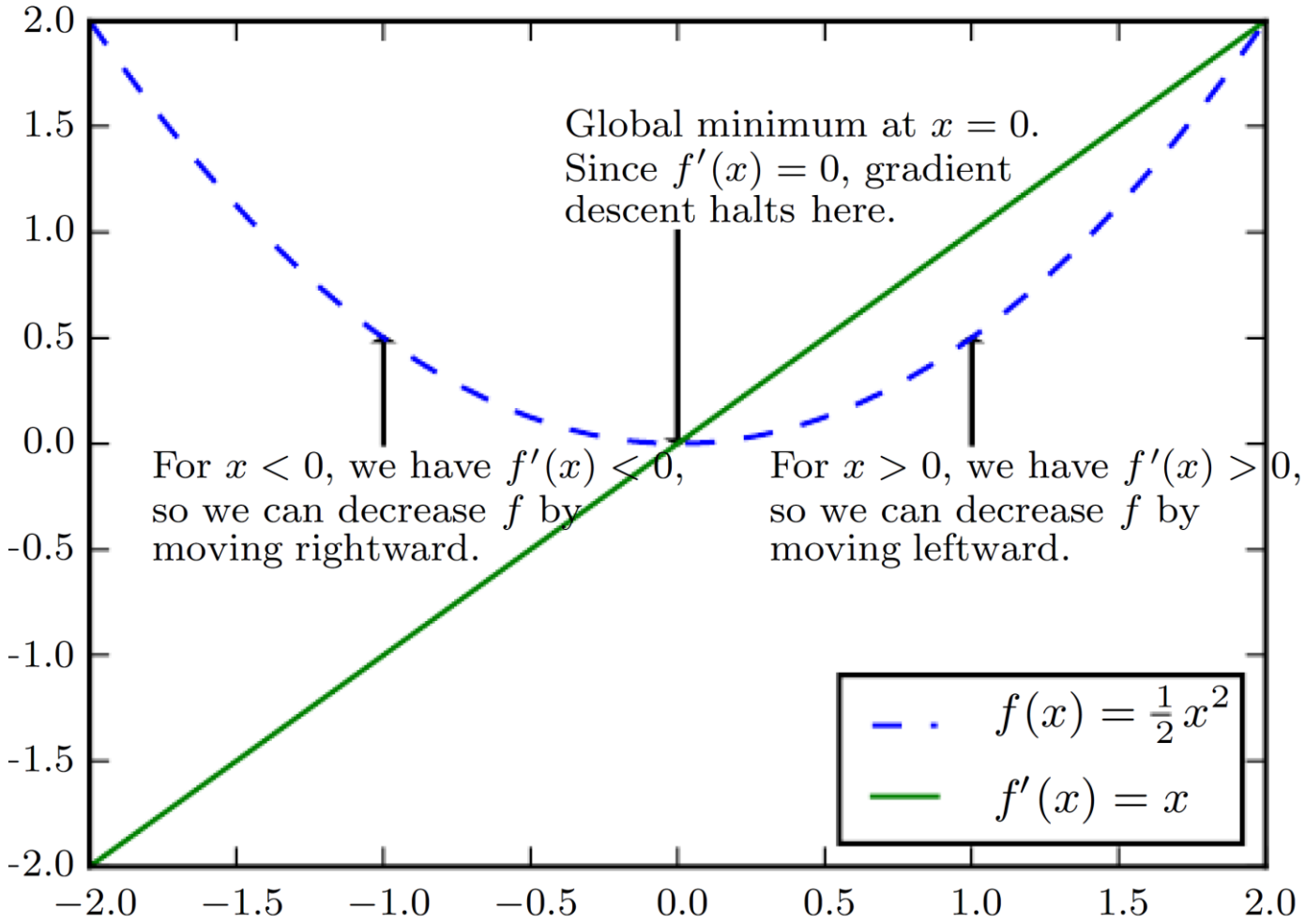
- Very often, there is no closed-form solution
- Gradient-based learning algorithms are thus called for to update estimates of the solution via an iterative procedure

Gradient-based Optimization

- Nonlinearity of NN causes non-convex loss functions
- Method to minimize the **cost function** by updating weights
- **No convergence guarantee**
- Sensitive to initial values of parameters
  - Weight - small random values
  - Bias - zero of small positive values
- Gradient descent:
  - Iteratively moving in the direction of steepest descent as defined by the negative of the gradient
- Stochastic gradient descent (SGD)
  - To handle large training sets
  - Only run a subset of the training sets (i.e., batch/minibatch) for each update
  - Easier to converge

Gradient Descent

- Follow the slope



Gradient Descent (Steepest Descent)

- To decrease $J(w)$ in the direction in which it decreases the fastest

$$w^{(n+1)} = w^{(n)} - \epsilon \nabla_w J\left(w^{(n)}\right)$$

  where ϵ controls the step size for each update

- The negative gradient $-\nabla_w J\left(w^{(n)}\right)$ points to the direction in which $J(w)$ decreases the fastest at $w^{(n)}$
  - To see this, we define the directional directive at w_0 with respect to α to be

$$\frac{\partial}{\partial \alpha} J(w_o + \alpha\mu)$$

  - Using the chain rule, it can then be evaluated as

$$\frac{\partial}{\partial \alpha} J(w_0 + \alpha\mu) = \mu^T \nabla_w J(w_0), \qquad when\ \alpha = 0.$$

  - Using Tayper-1 approximation at w_0

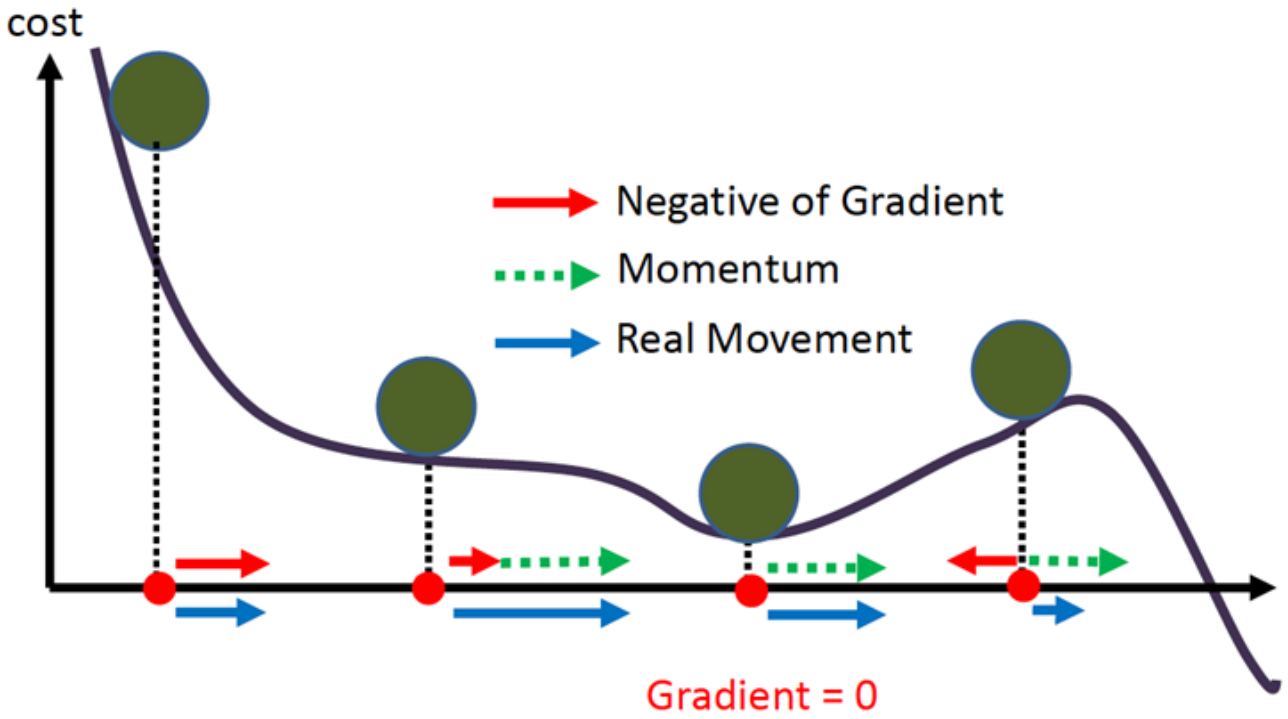
$$J(w) \approx J(w_0) + (w - w_0)^T \nabla_w J(\omega_0)$$

  - The unit vector μ that points in the direction $-\nabla_w J\left(w^{(n)}\right)$ yields a minimal directive among other unit vectors

Momentum

- Prefers to go in a similar direction as before

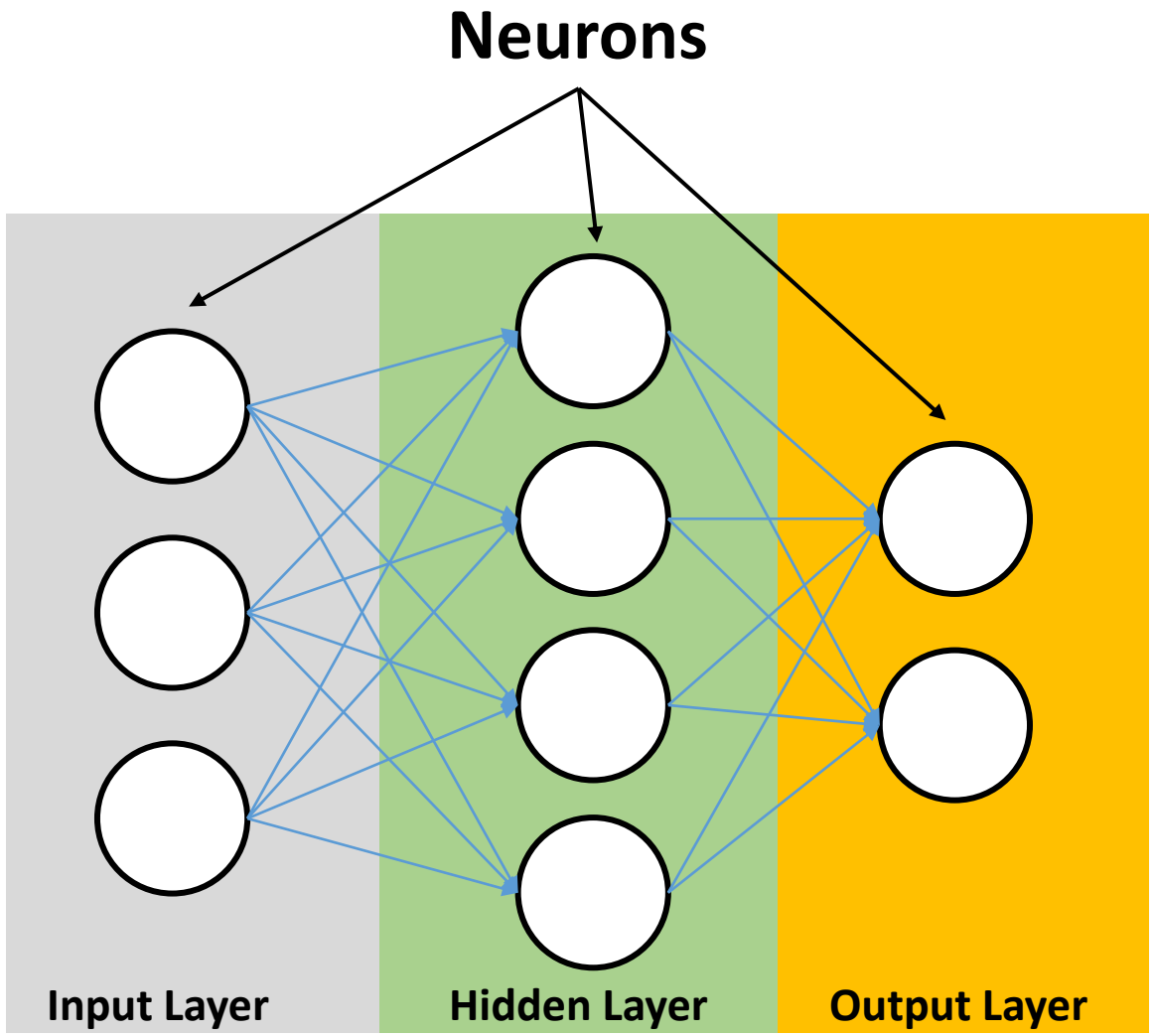
Movement = Negative of Gradient + **Momentum**



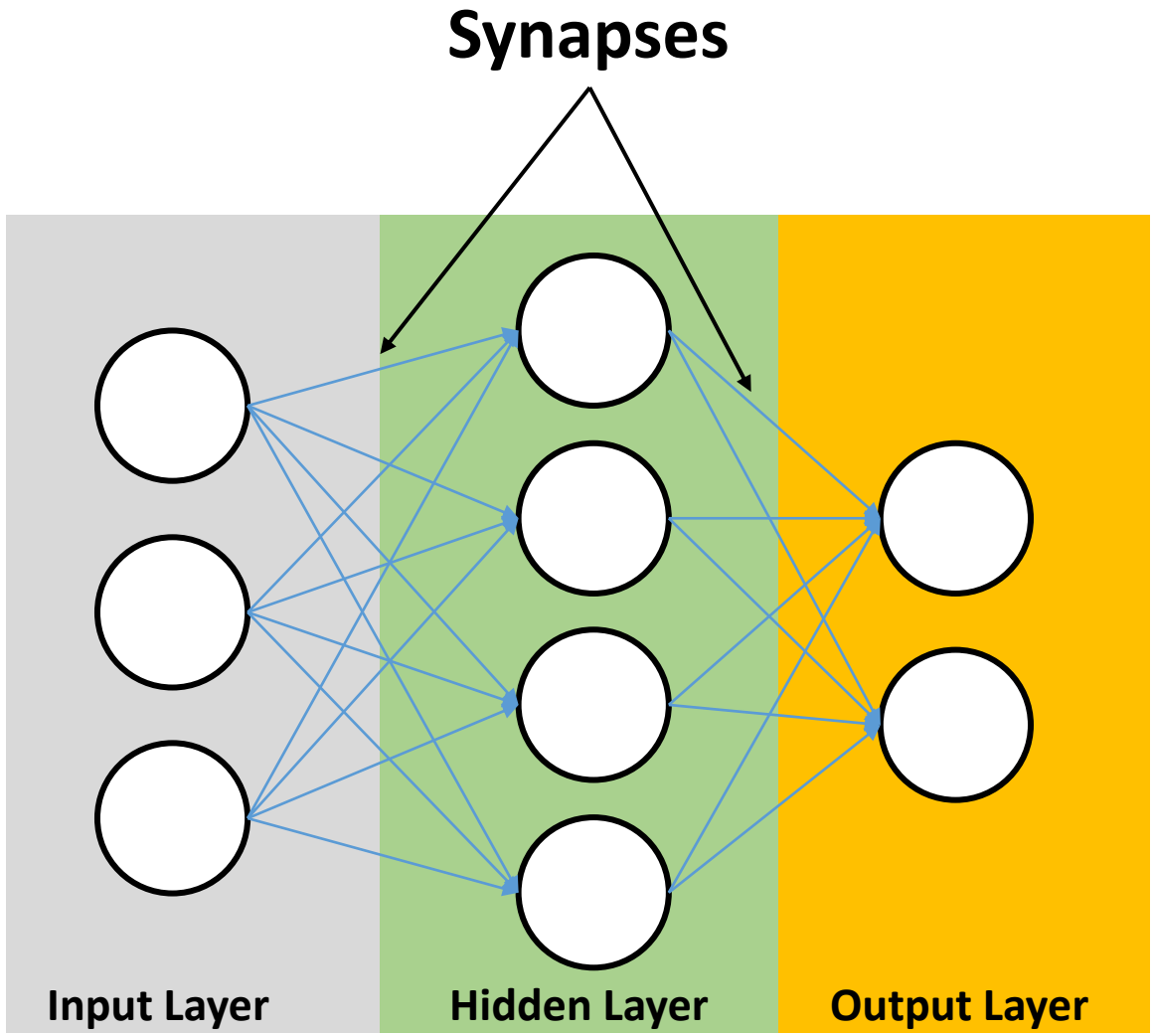
Outline

- AI, ML and DL
- ML Basics
- DL Overview

DNN Terminology

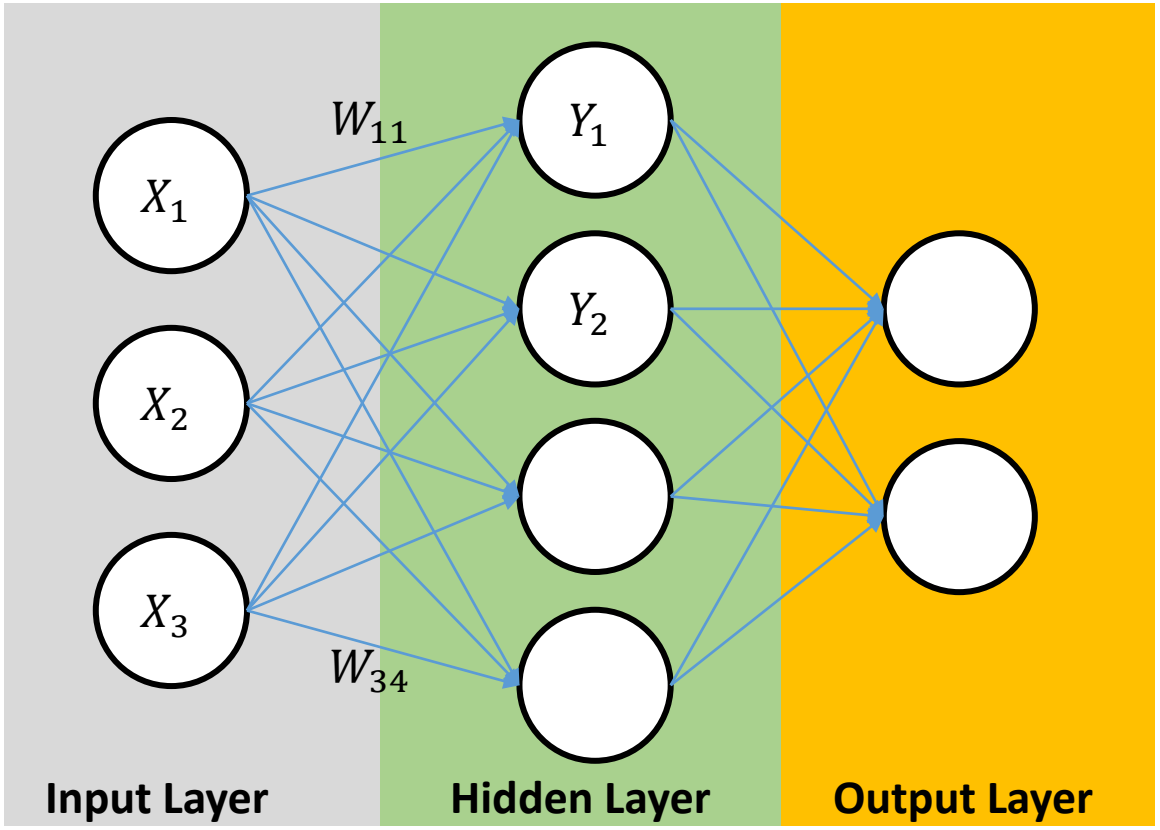


DNN Terminology



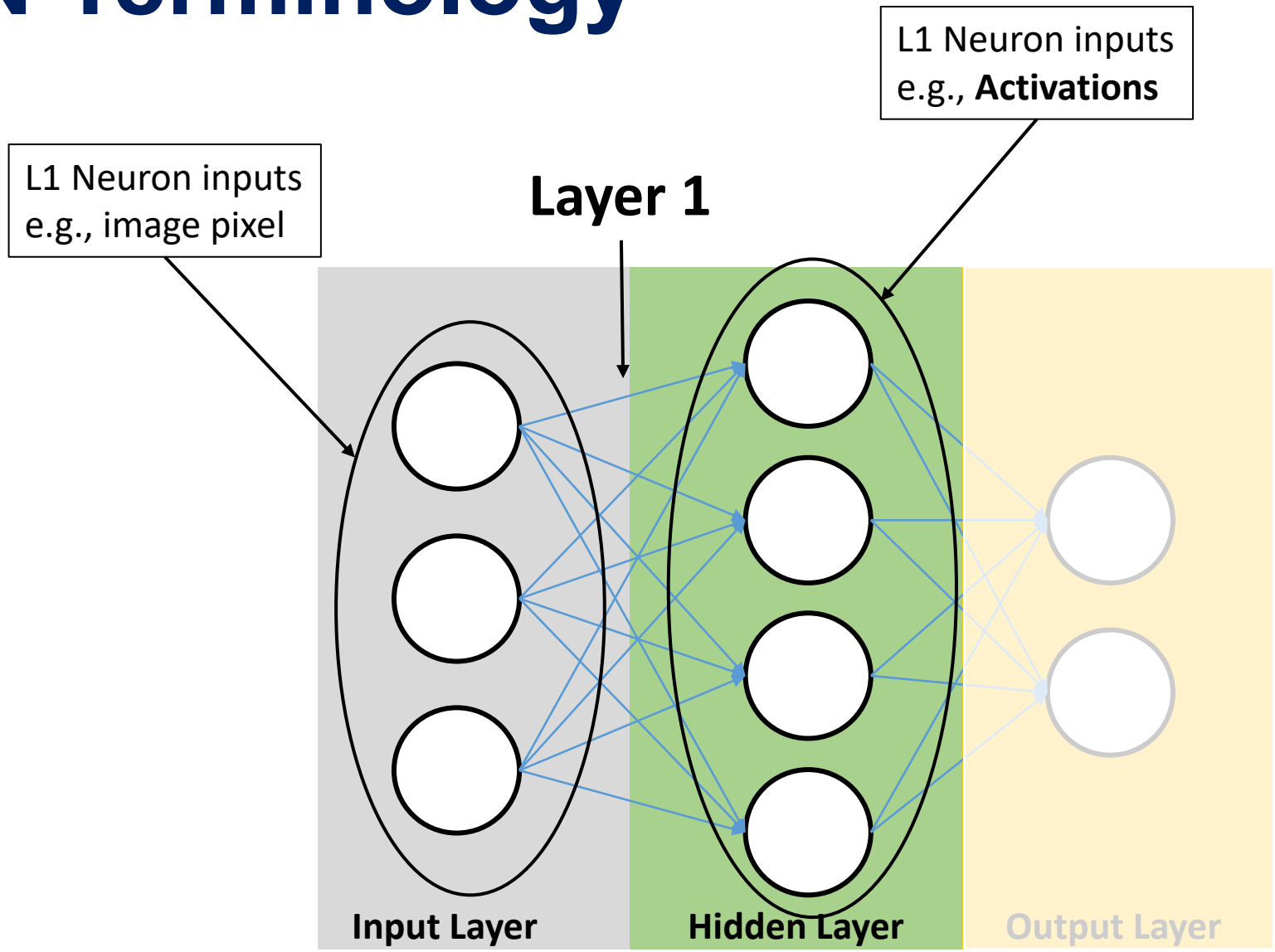
DNN Terminology

- Each synapse has a weight for neuron activation
- **Weight Sharing**: multiple synapses use the **same weight value**

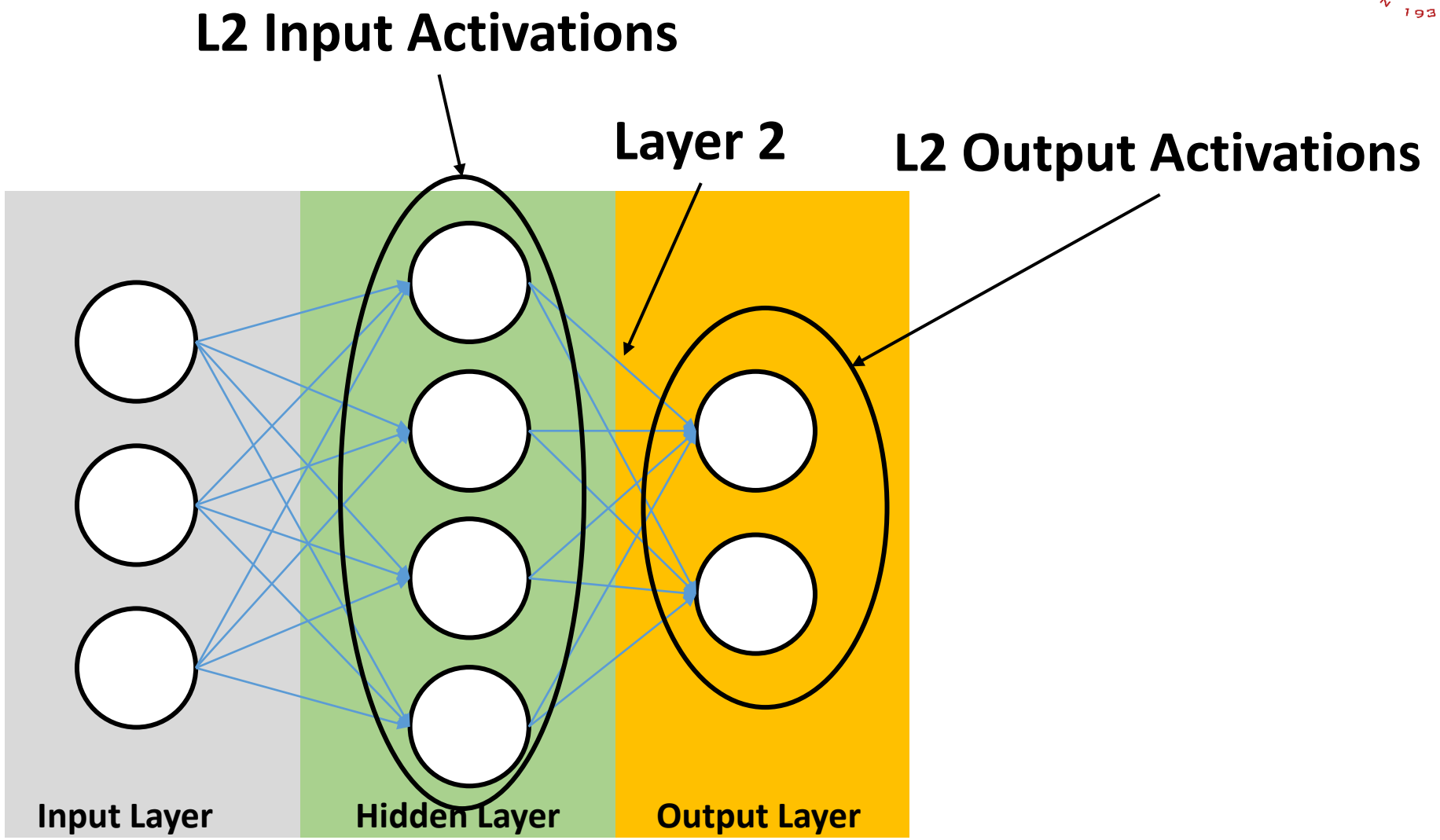


$$Y_j = activation(\sum_{i=1}^{3} W_{ij} \times X_i)$$

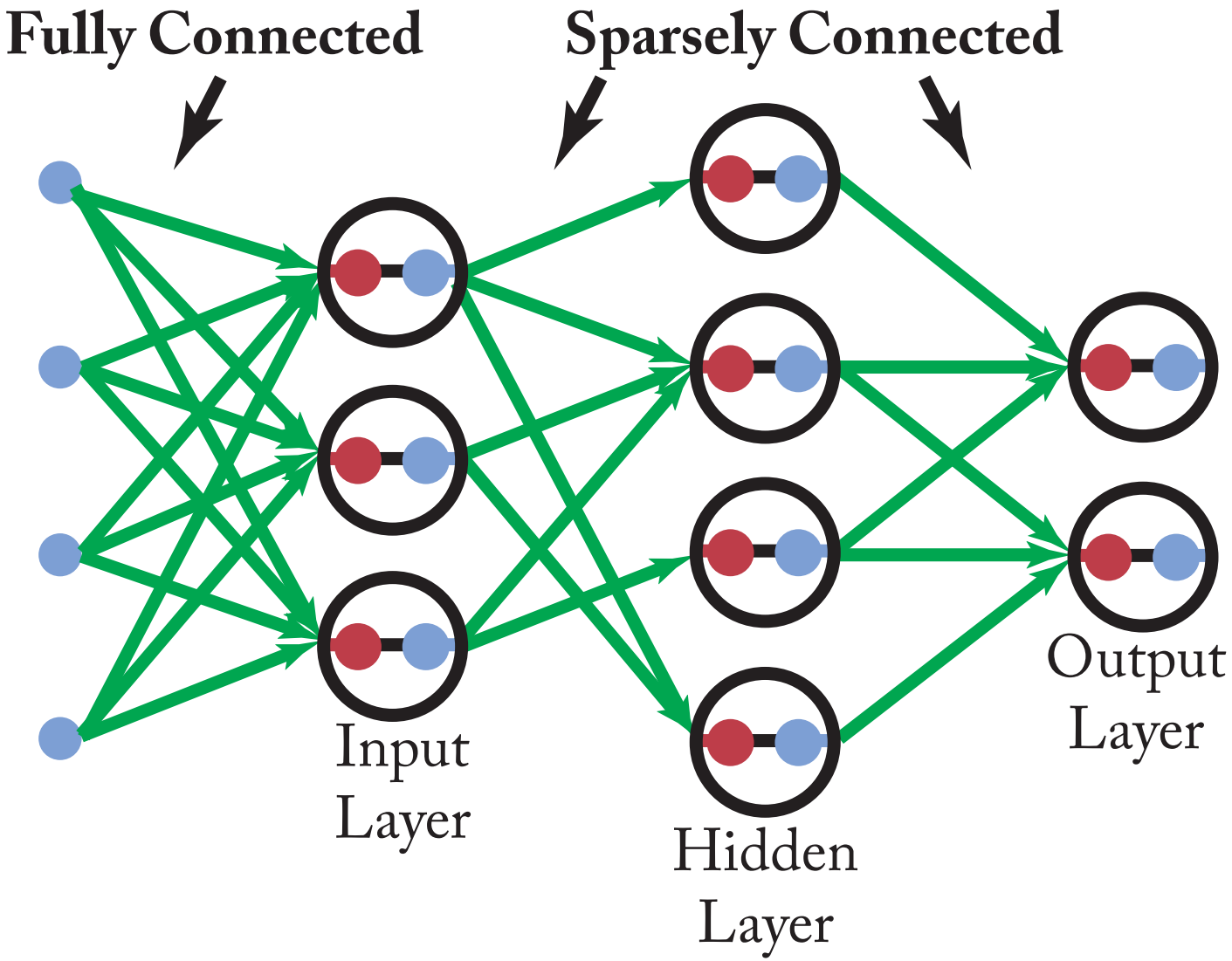
DNN Terminology



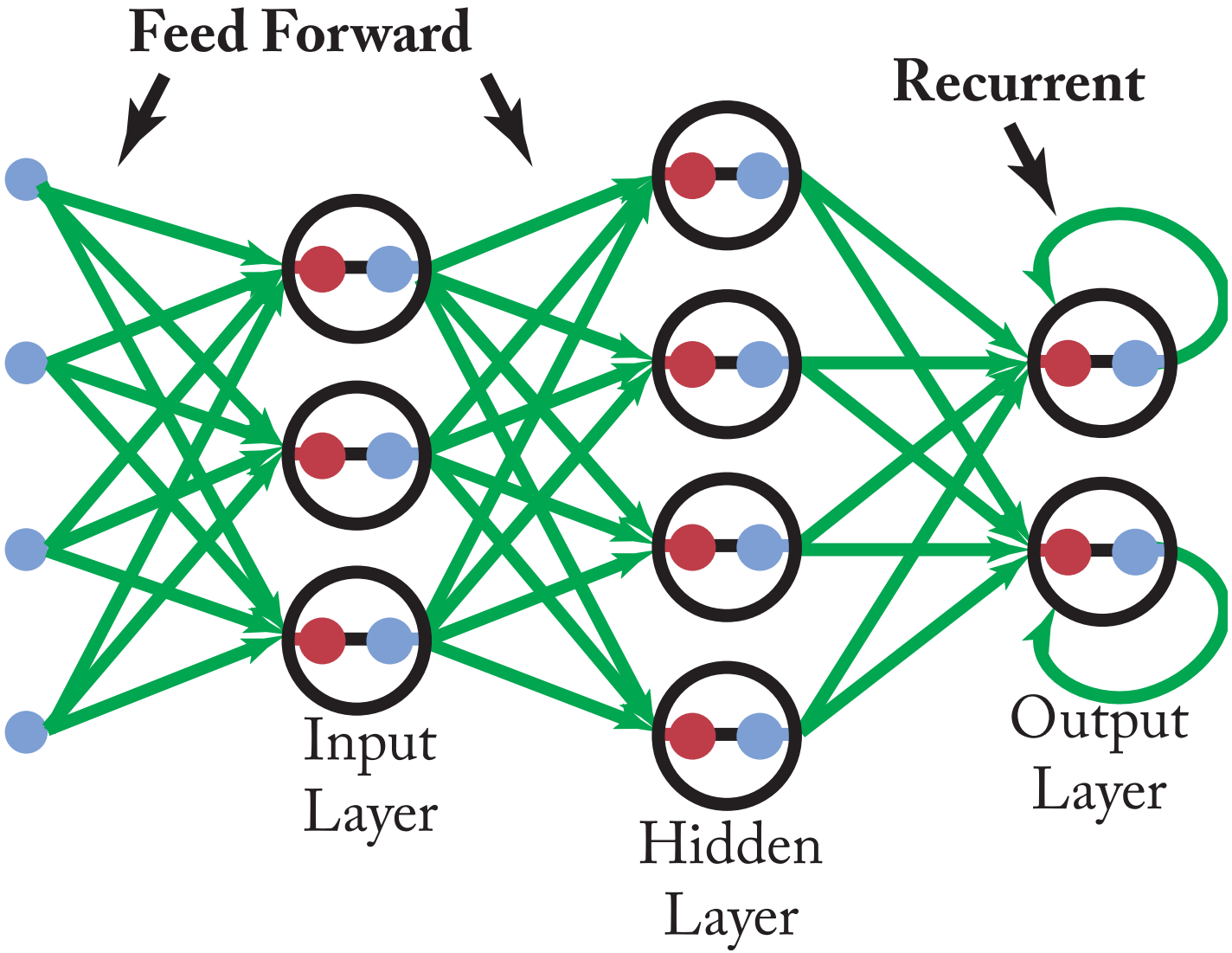
DNN Terminology



Fully-connected v.s. Sparsely Connected



Feed-forward v.s. Feed-back



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, Auto Encoder, General Adversarial Networks(GAN)

Multi-layer Perceptron

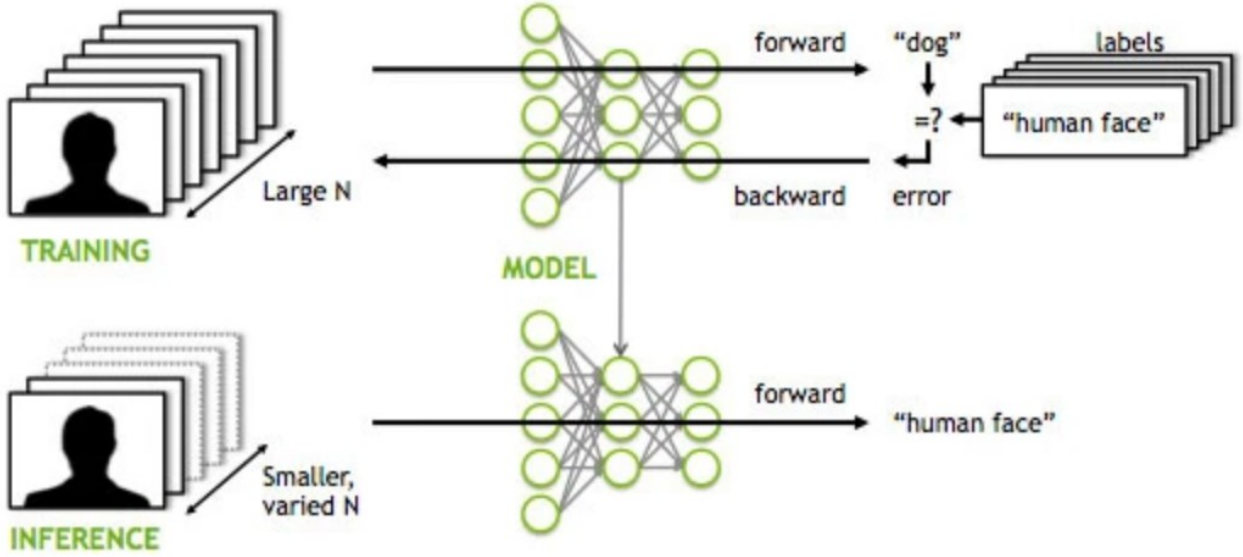
- A non-linear classifier
- Training: find network weighs w to minimize the error between true training labels y_i and estimated labels $f_w(x_i)$

$$E(w) = \sum_{i=1}^{N} (y_i - f_w(x_i))^2$$

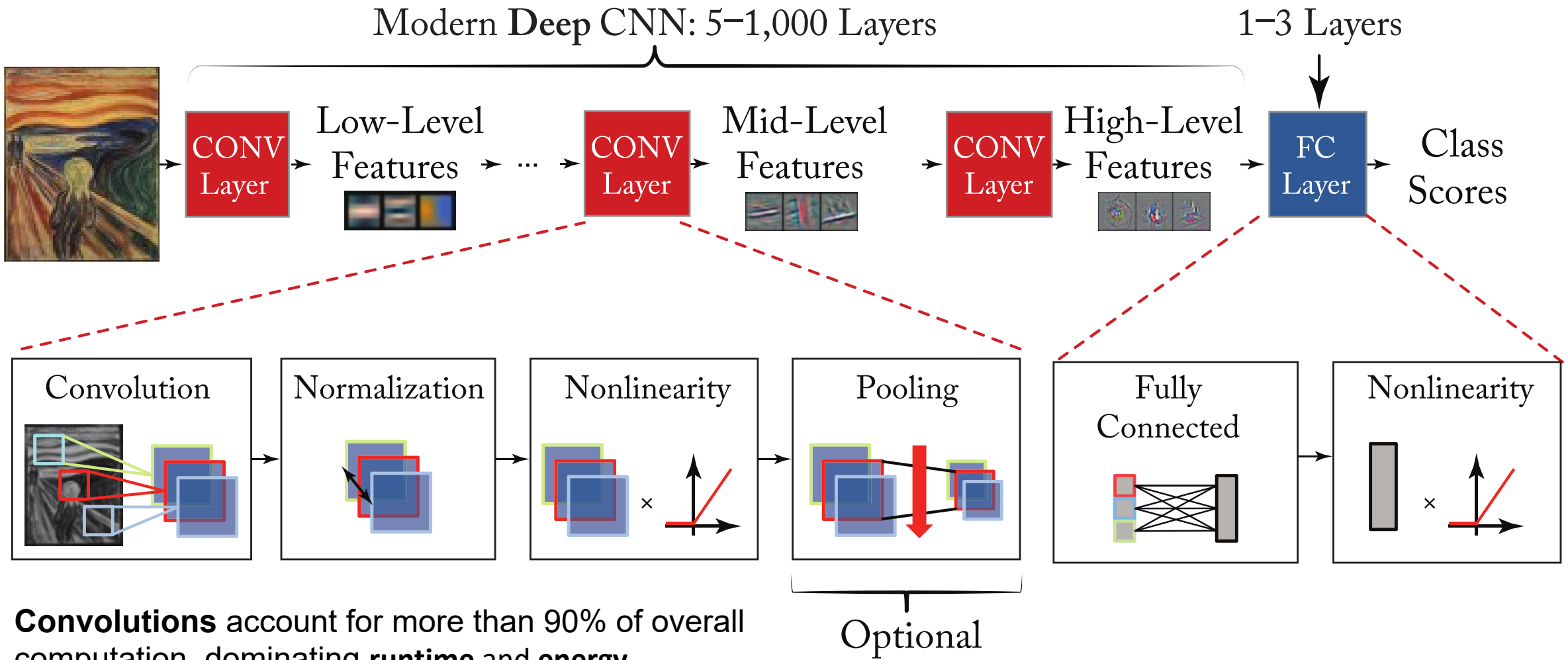
- Minimization can be done by gradient descent provided f is differentiable
- This training method is called **back-propagation**

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

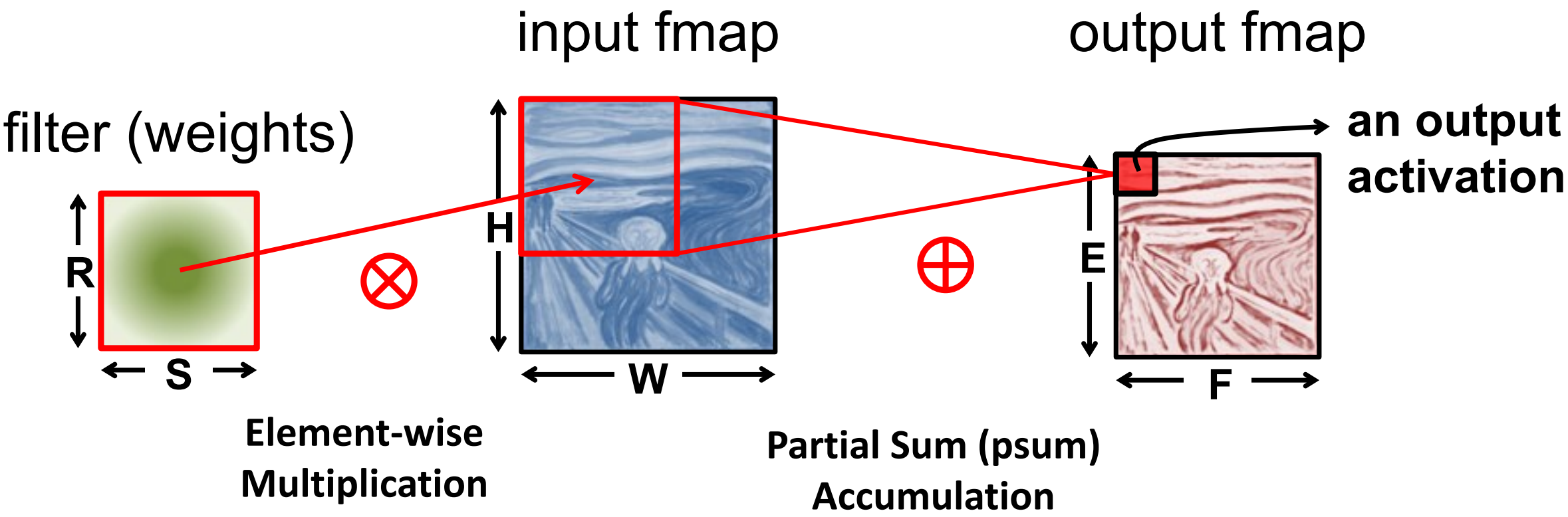


Deep Convolutional Neural Network

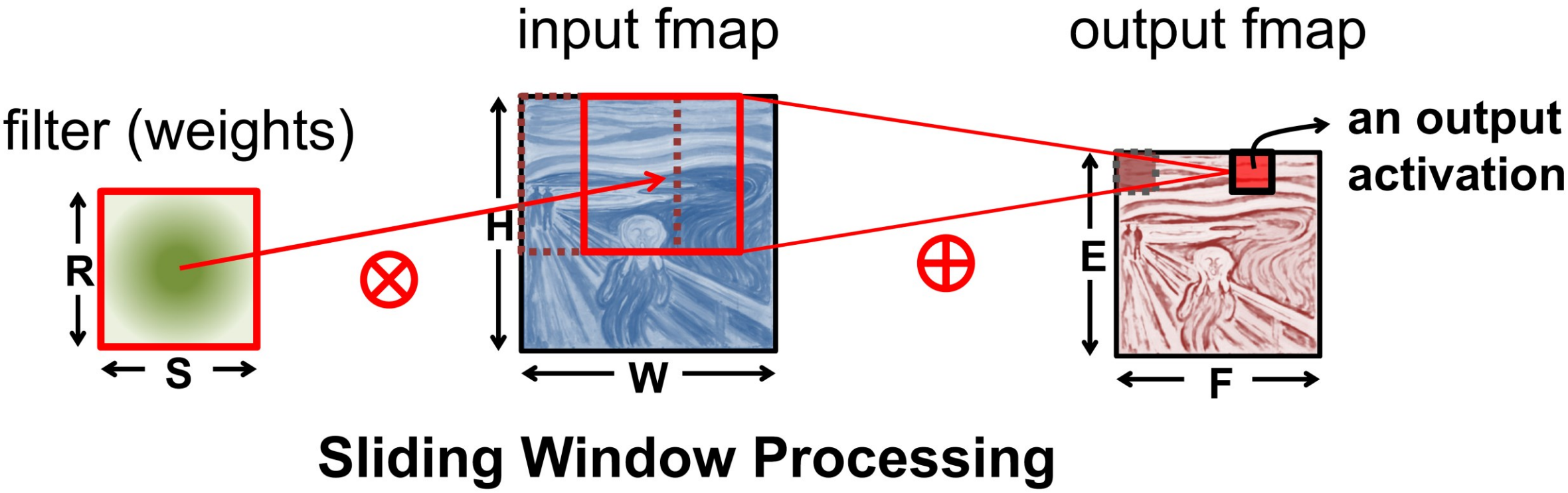


Convolutions account for more than 90% of overall computation, dominating **runtime** and **energy consumption**

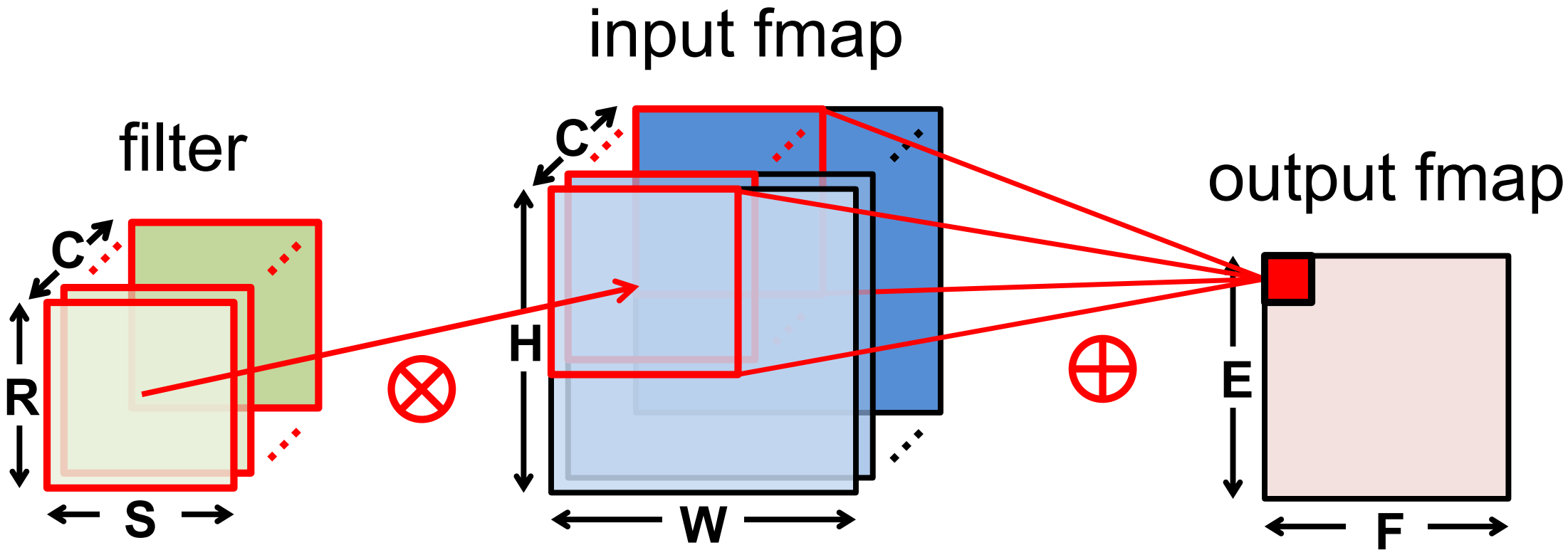
Convolution (CONV) Layer



Convolution (CONV) Layer

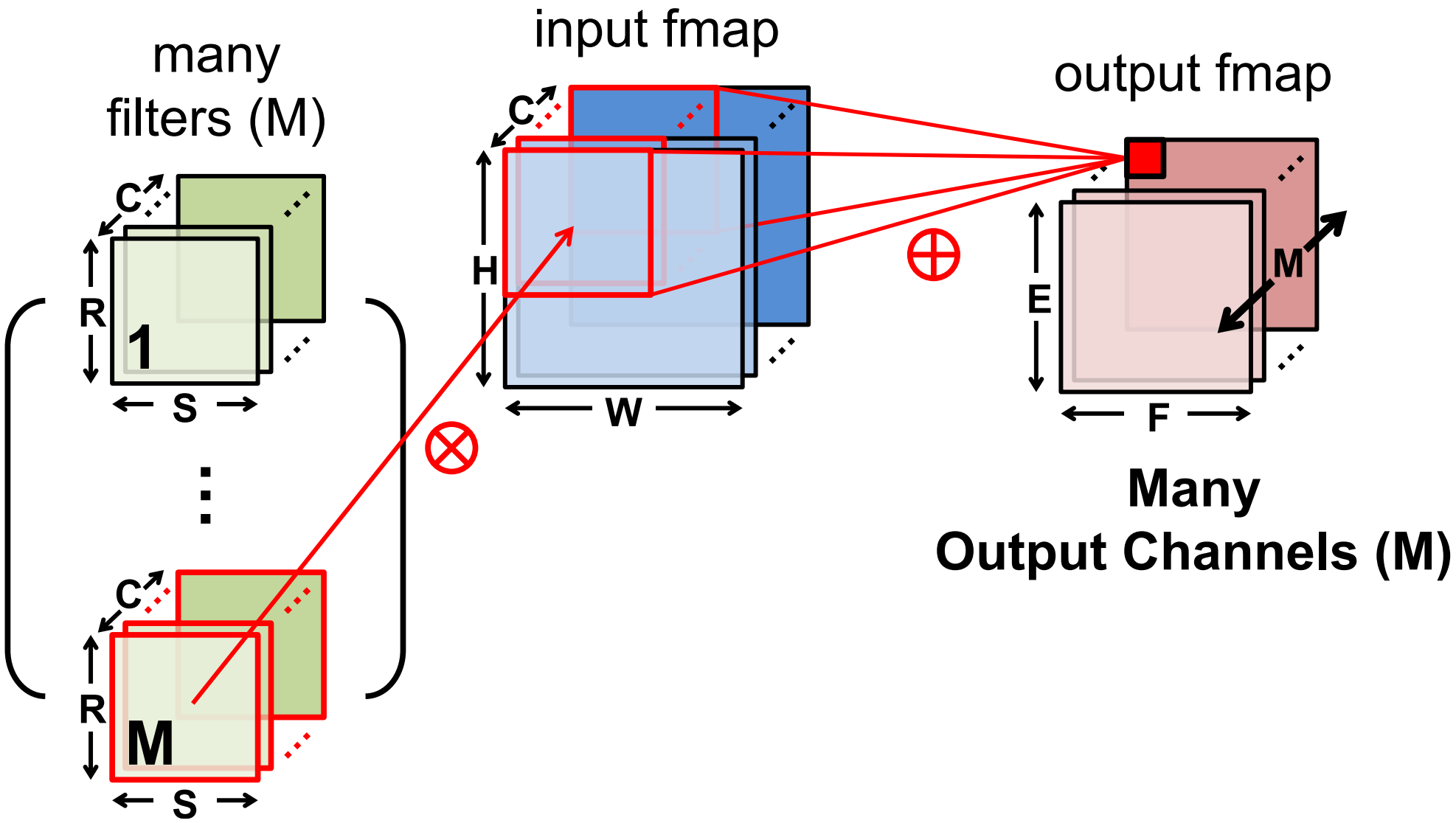


Convolution (CONV) Layer

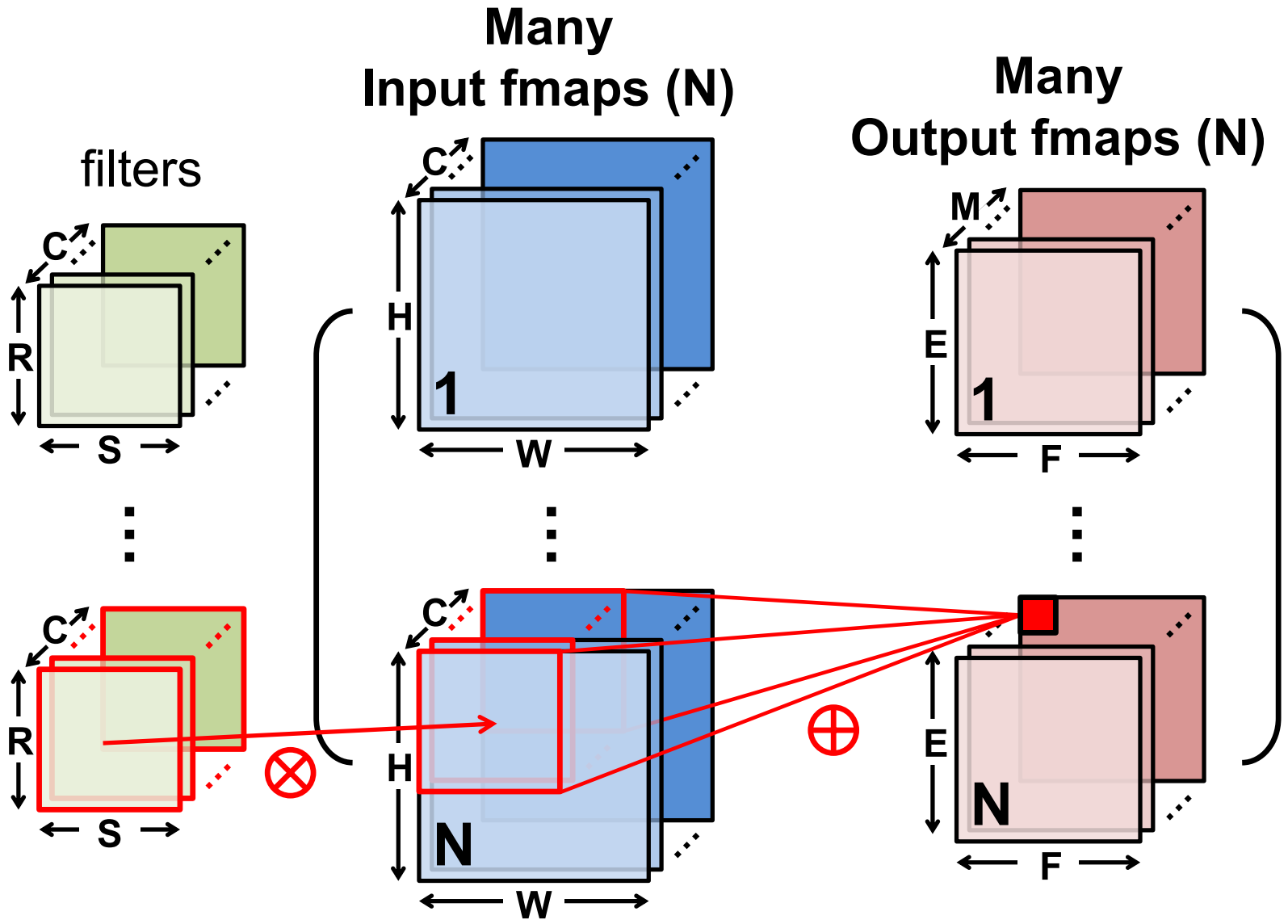


Many Input Channels (C)

Convolution (CONV) Layer



Convolution (CONV) Layer



CNN Decoder Ring

Shape Parameter	Description
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)

$$O[n][m][x][y] = Activation\left(B[m] + \sum_{i=0}^{R-1}\sum_{j=0}^{S-1}\sum_{k=0}^{C-1} I[n][k][U_{x+1}][U_{y+j}] \times W[m][k][i][j]\right)$$

$$0 \le n \le N, 0 \le m \le \mathrm{M}, 0 \le y \le E, 0 \le x \le F$$

$$E = \frac{H - R + U}{U}, F = \frac{W - S + U}{U},$$

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

CONV Layer Implementation

- Naïve 7-layer for-loop implementation:

```
1  for (n=0; n<N; n++) {
2      for (m=0; m<M; m++) {
3          for (x=0; x<F; x++) {
4              for (y=0; y<E; y++) {
5                  O[n][m][x][y] = B[m];
6                  for (i=0; i<R; i++) {
7                      for (j=0; j<S; j++) {
8                          for (k=0; k<C; k++) {
9                              O[n][m][x][y] += I[n][k][Ux+i][Uy+j] × W[m][k][i][j];
10                         }
11                     }
12                 }
13                 O[n][m][x][y] = Activation(O[n][m][x][y]);
14             }
15         }
16     }
17 }
```

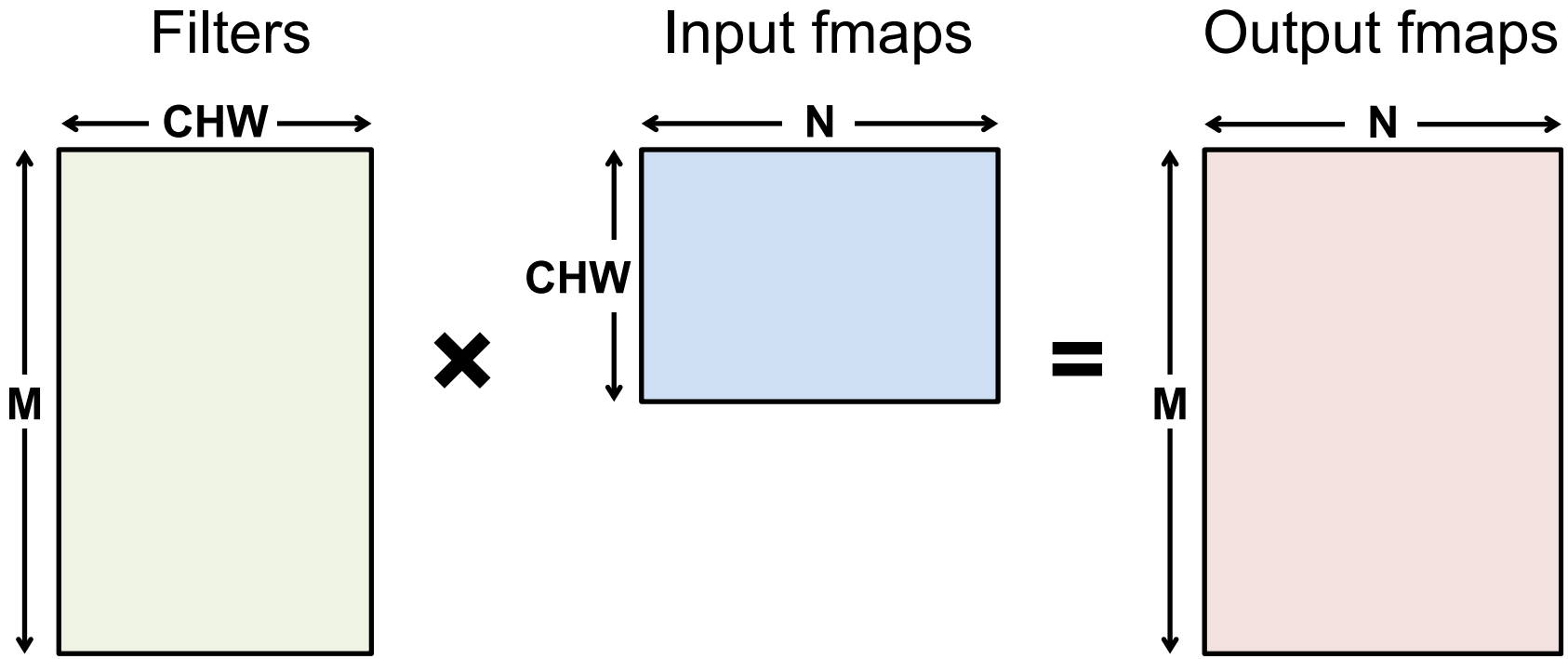
for each output fmap value

Convolve a window and apply activation

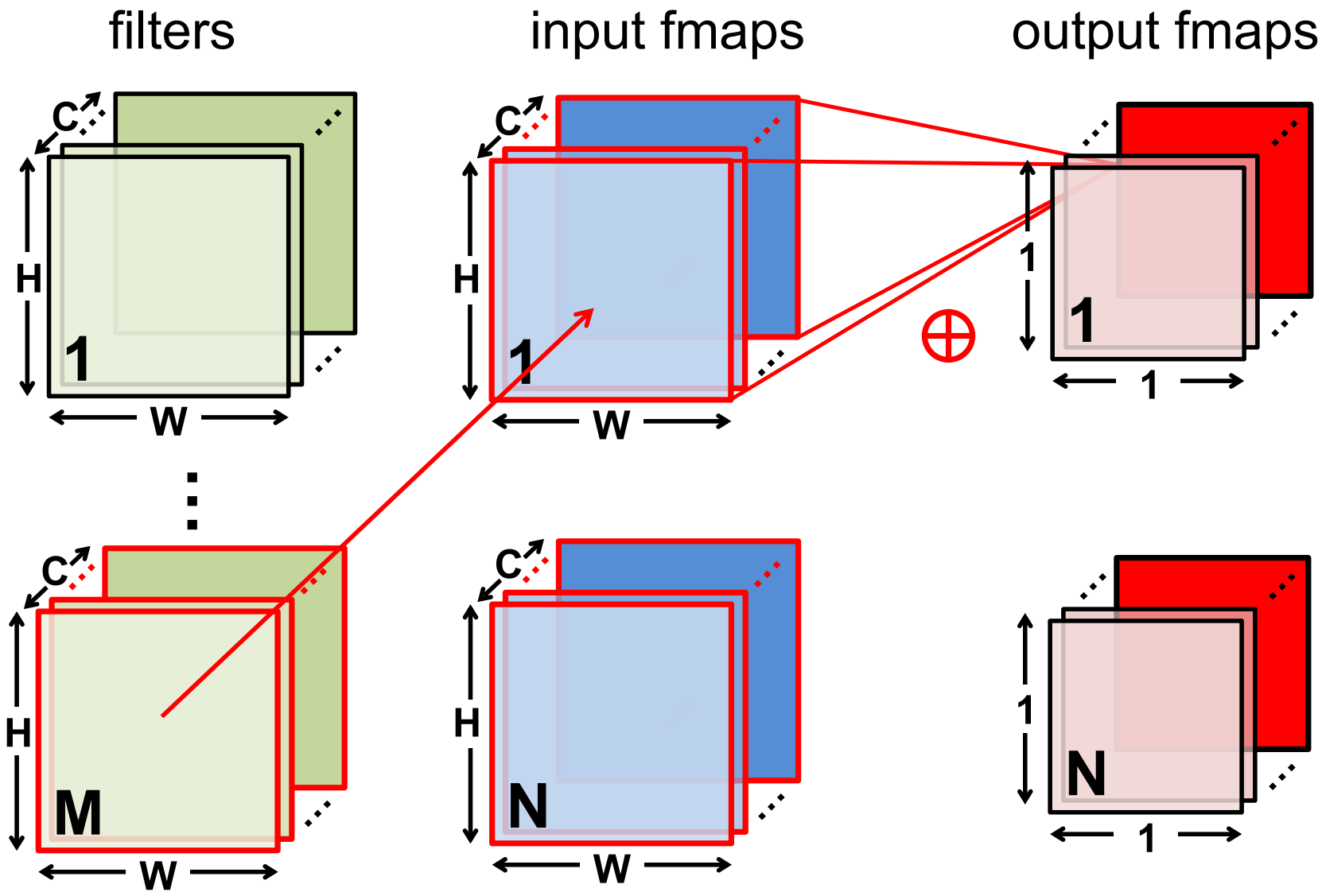
Shape Parameter	Description
N	Number of **input fmaps/output fmaps** (batch size)
C	Number of 2-D **input fmaps /filters** (channels)
H/W	Height/Width of **input fmap** (activations)
R/S	Height/Width of 2-D **filter** (weights)
M	Number of 2-D **output fmaps** (channels)
E/F	Height/Width of **output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

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**

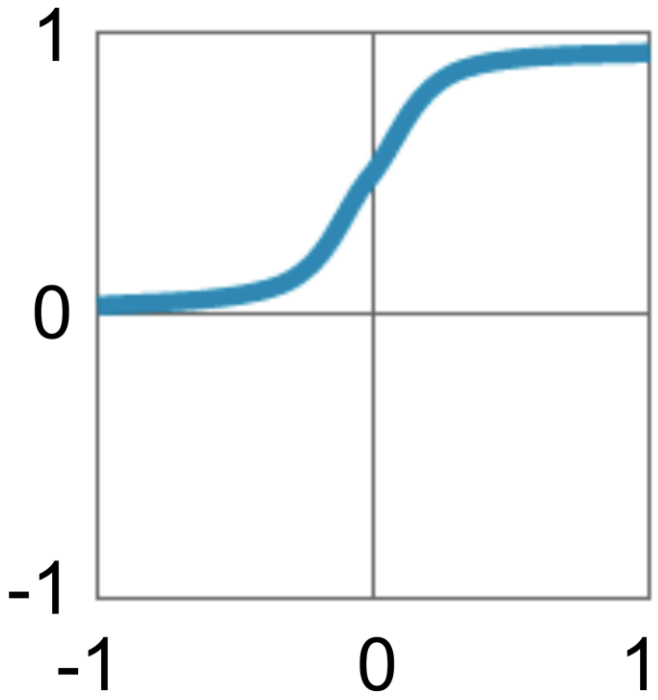


FC Layer – From CONV Layer POV



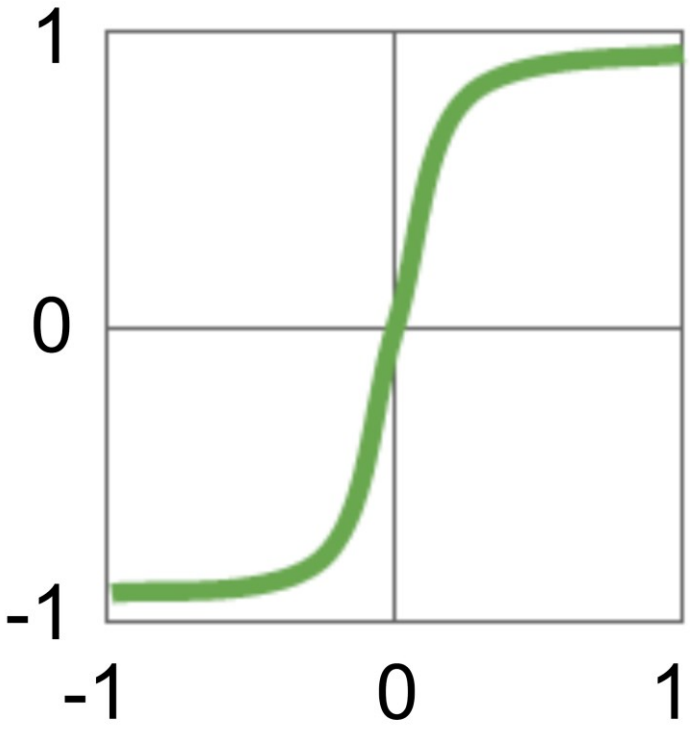
Traditional Activation Functions

Sigmoid



$$y = \frac{1}{1 + e^{-x}}$$

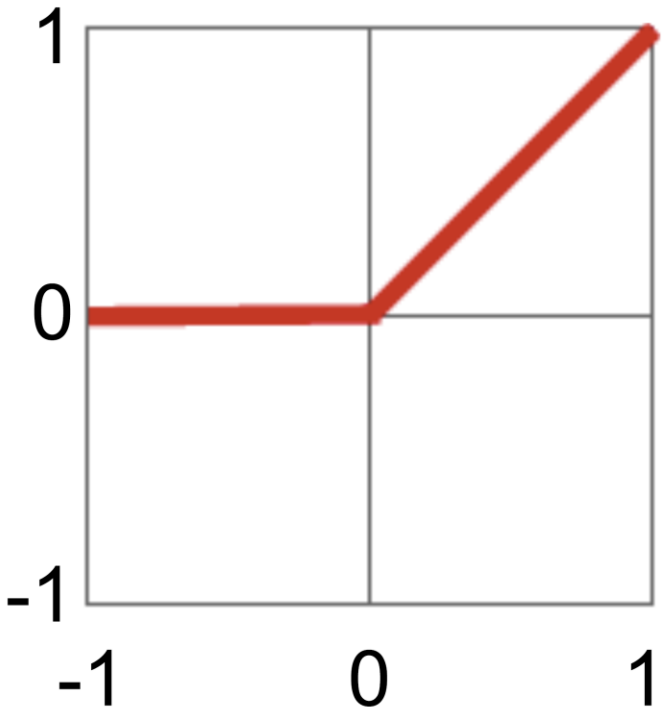
Hyperbolic Tangent



$$y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

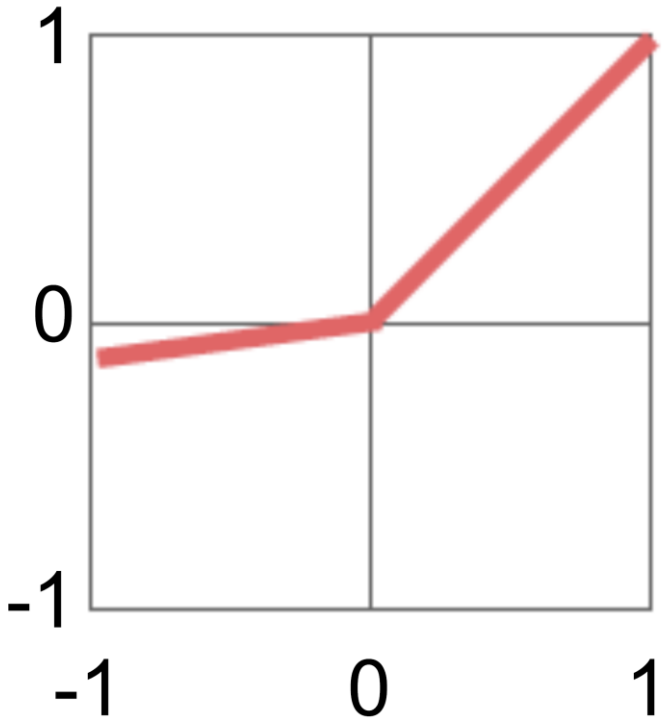
Modern Activation Functions

Rectified Linear Unit (ReLU)



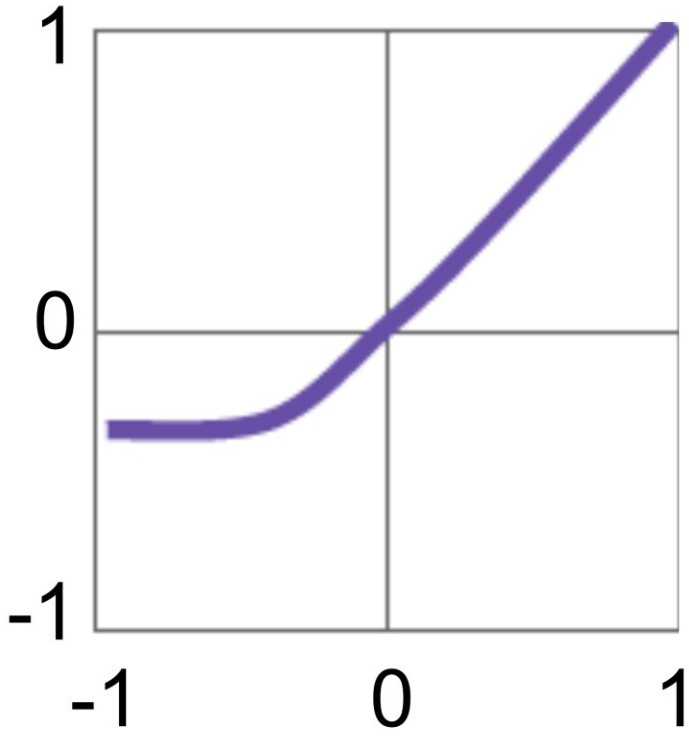
$$y = max(0, x)$$

Leaky ReLU



$$y = max(ax, x)$$
$$a = small\ constant, e.g. 0.1$$

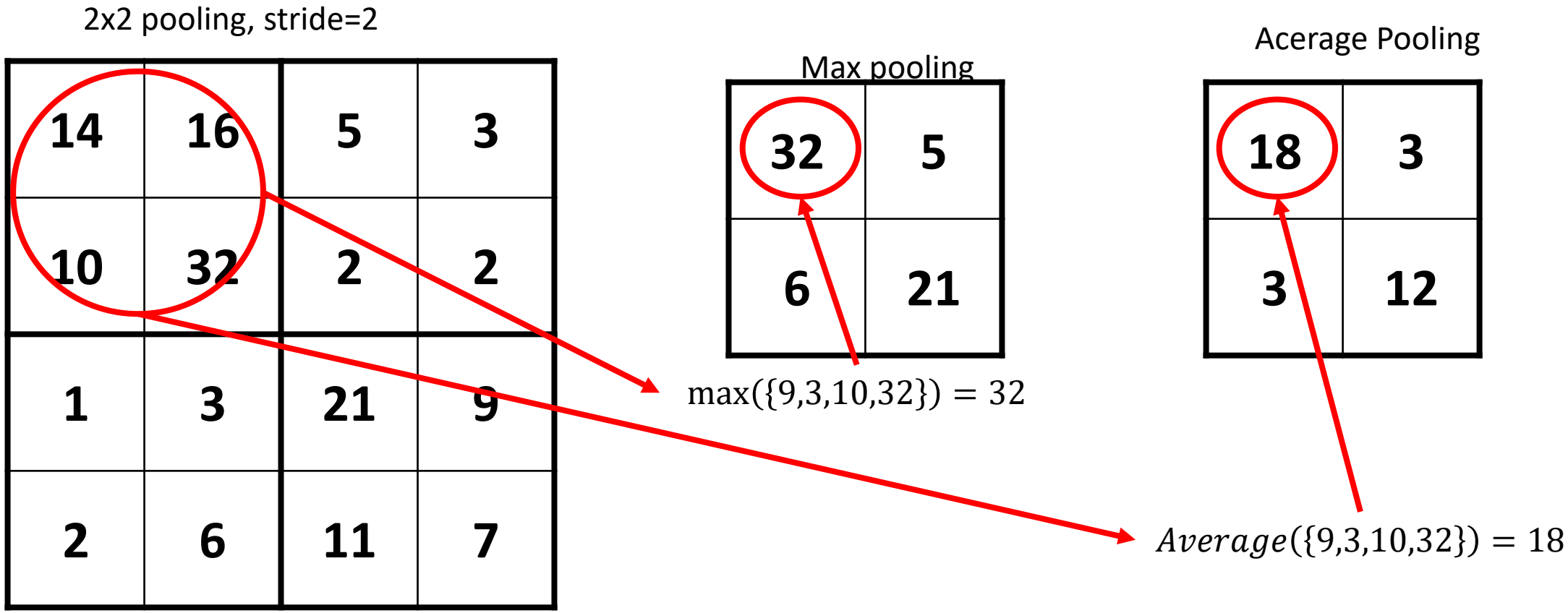
Exponential LU (ELU)



$$y = \begin{cases} x, & x \geq 0 \\ a(e^x - 1), & x < 0 \end{cases}$$
$$a = small\ constant, e.g. 0.1$$

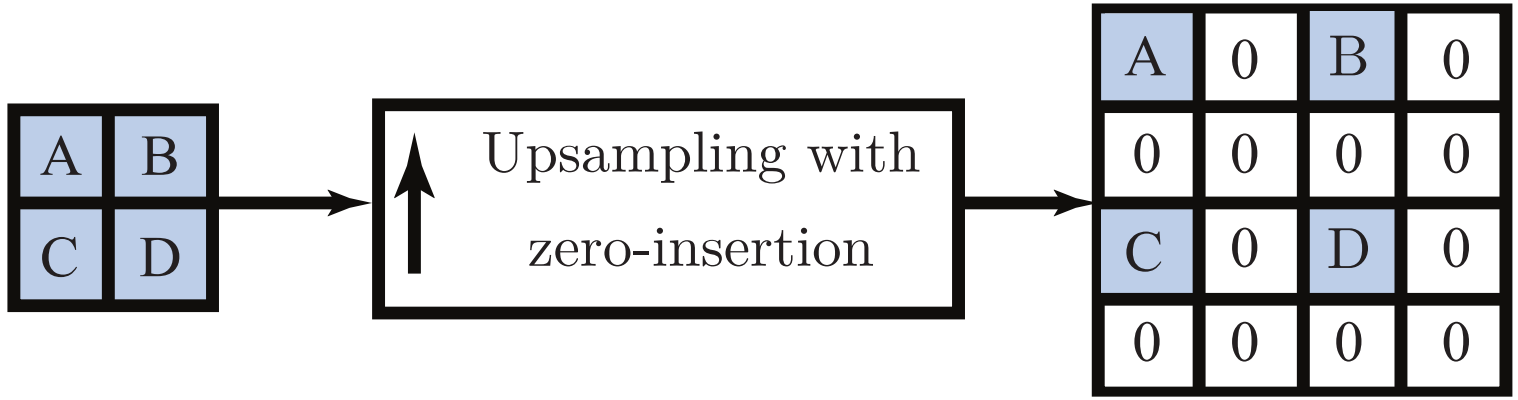
Pooling (POOL) Layer

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



Unpooling and Upsampling

Zero insertion (unpooling)



Nearest neighbor

POOL Layer Implementation

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

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

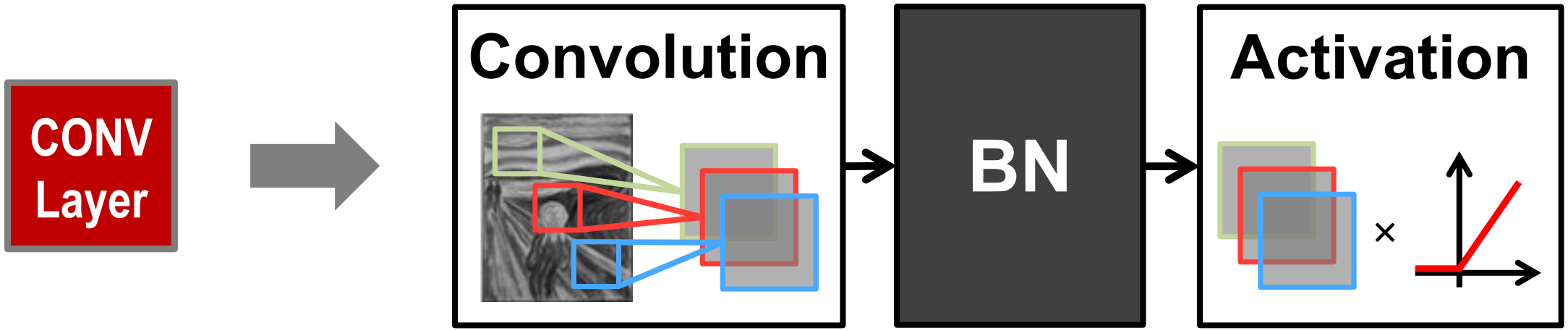
```
1  for (n=0; n<N; n++) {
2      for (m=0; m<M; m++) {
3          for (x=0; x<F; x++) {
4              for (y=0; y<E; y++) {
5                  max = -Inf;
6                  for (i=0; i<R; i++) {
7                      for (j=0; j<S; j++) {
8                          if (I[n][m][Ux+i][Uy+j] > max) {
9                              max = I[n][m][Ux+i][Uy+j];
10                         }
11                     }
12                 }
13                 O[n][m][x][y] = max;
14             }
15         }
16     }
17 }
```

for each output fmap value

Find the max within a window

Normalization (NORM) Layer

- Batch Normalization (BN)
  - Normalize activations towards mean=0 and standard deviation= 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.



Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

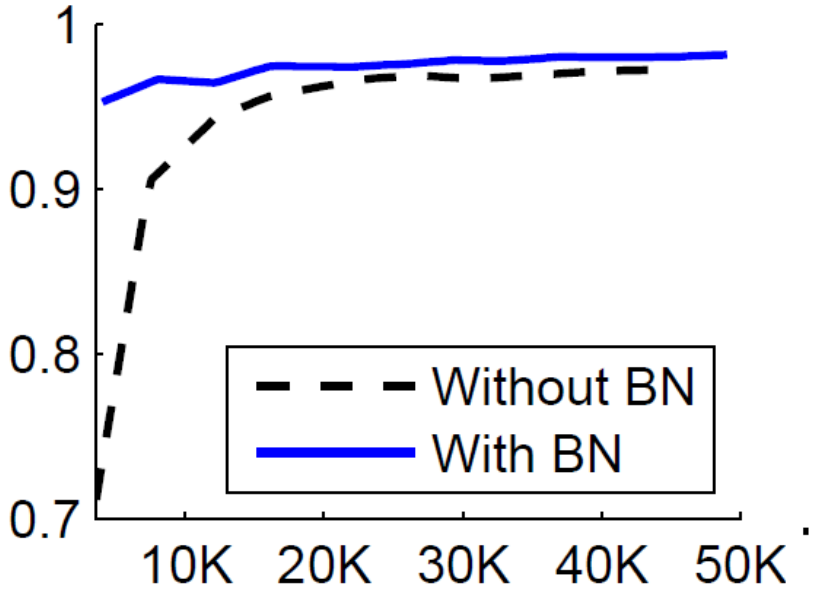
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

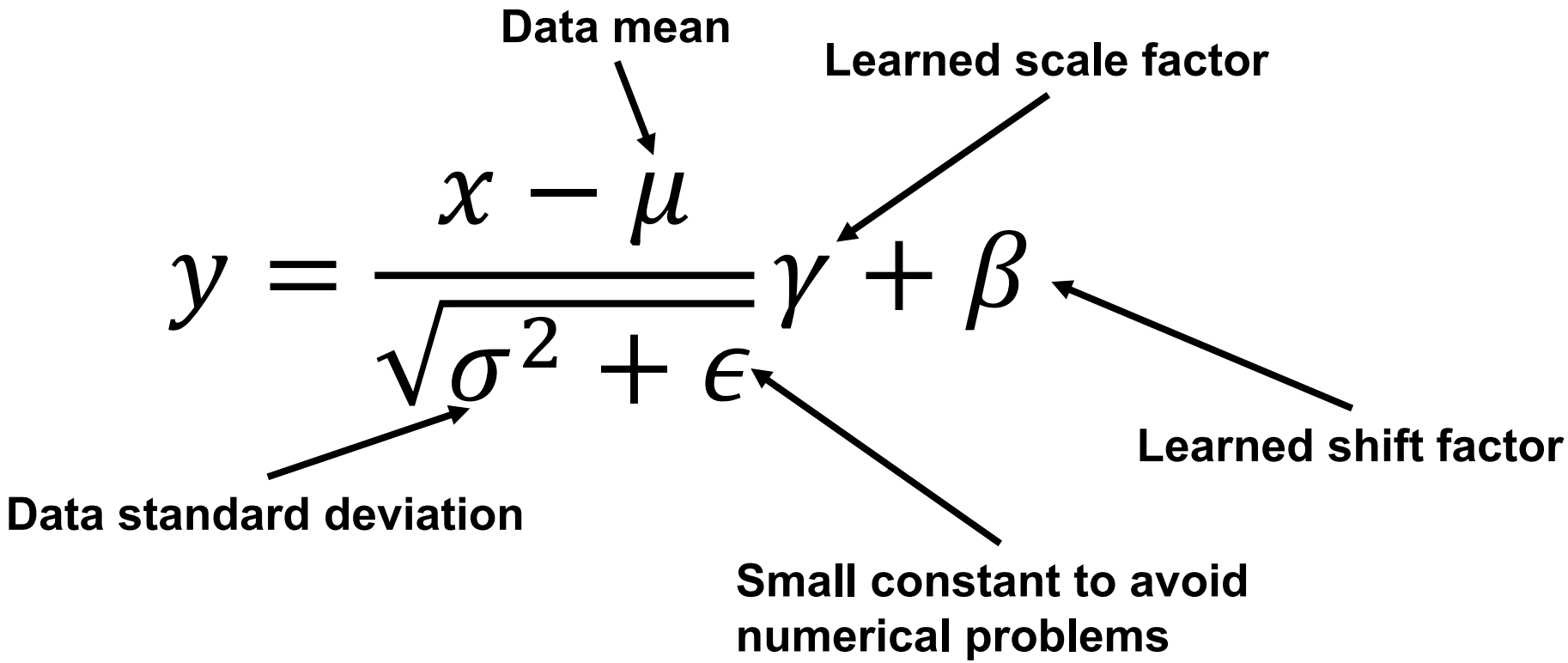
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$



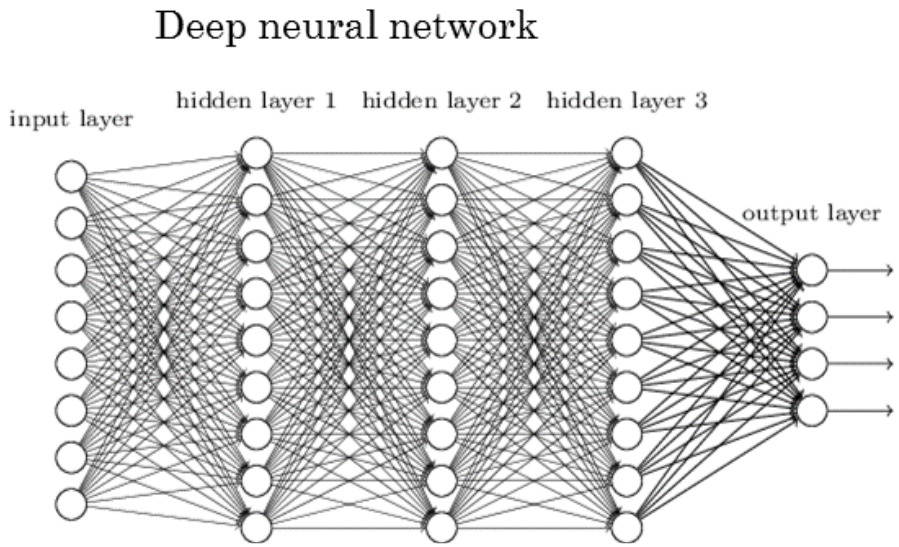
BN Layer Implementation

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



Architecture Design

- Architecture: overall structure of the network
  - How many units it should have
  - How these units should be connected to each other
  - How to choose the depth and width of each layer
- Deeper networks often
  - Use far fewer units per layer and far fewer parameters
  - Generalize to the test set
  - Are harder to optimize

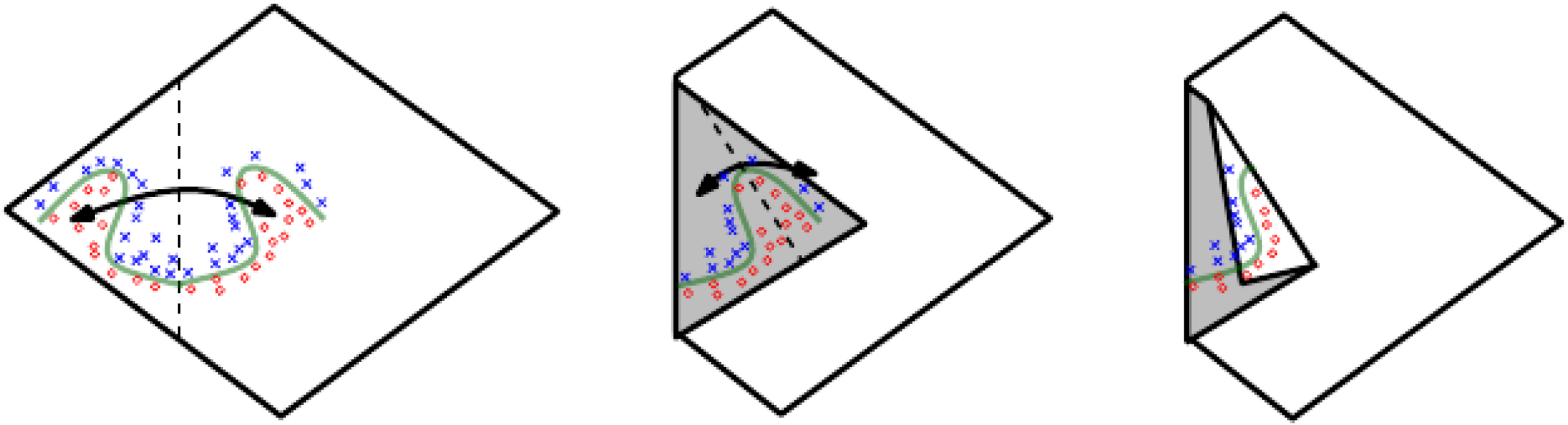


Universal Approximation Properties and Depth

- Given enough hidden units, a feedforward network can approximate any Borel measurable function
- A feedforward network with a single layer is sufficient to represent any function
  - but the layer may be infeasibly large and may fail to lean and generalize correctly
- Using deeper models can
  - Reduce the number of units required to represent the desired function
  - Reduce the generalization error

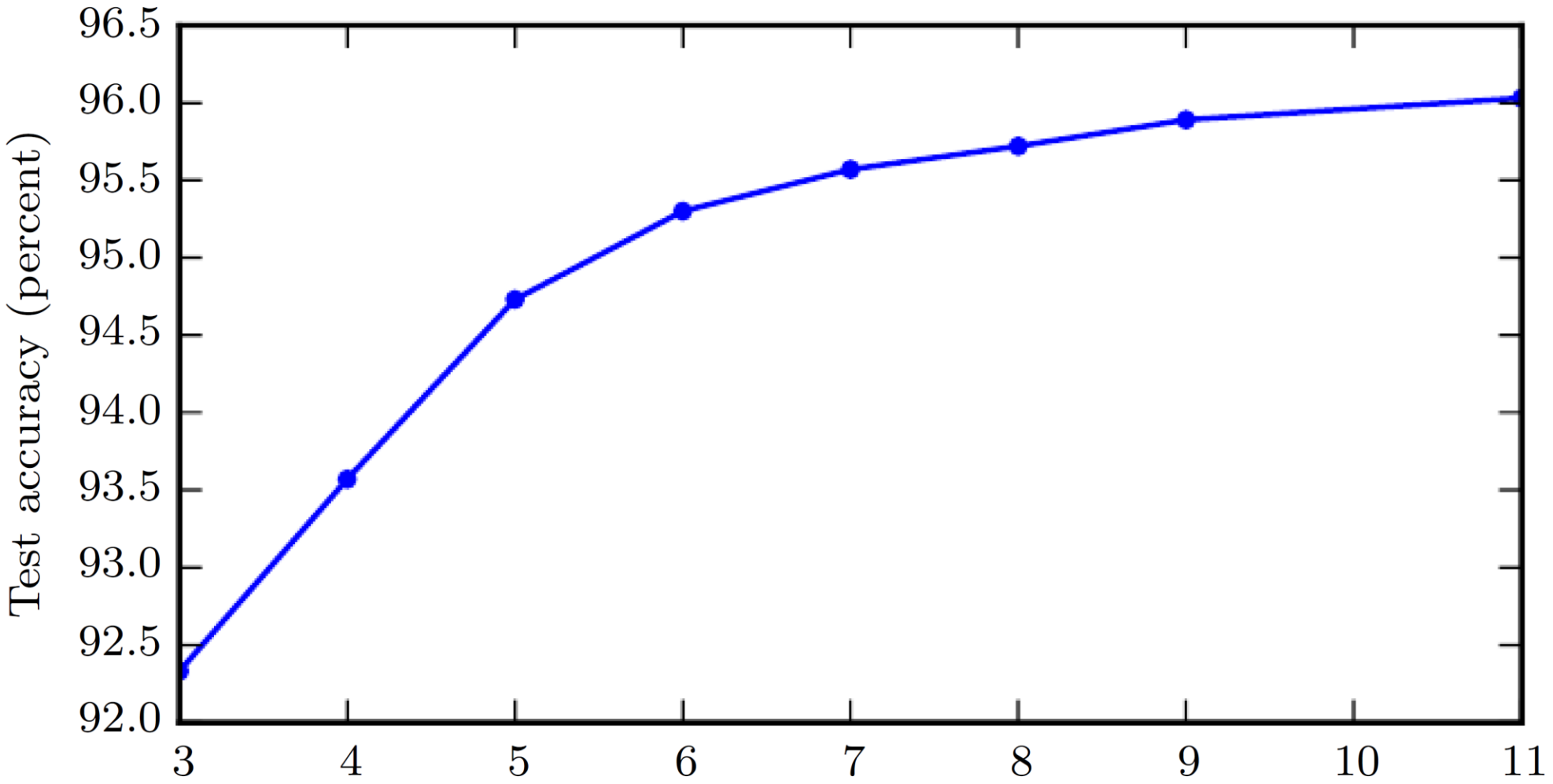
Universal Approximation Properties and Depth

- A network with absolute value rectification creates mirror images of the function computed on top of hidden units



Better Generalization with Greater Depth

- SVHN dataset



Large, Shallow Models Overfit More