COMP6211I: Trustworthy Machine Learning

Lecture 2

Minhao Cheng

Theory of Generalization

Formal definition

- Assume training and test data are both sampled from D
- The ideal function (for generating labels) is $f : f(x) \to y$
- Training error: Sample $x_1, \ldots, x_N$ from D and
  - $E_{tr}(h) = \frac{1}{N} \sum_{n=1}^{N} e(h(x_n), f(x_n))$
  - h is determined by $x_1, \ldots, x_n$
- Test error: Sample $x_1, \ldots, x_N$ from D and
  - $E_{te}(h) = \frac{1}{M} \sum_{m=1}^{M} e(h(x_m), f(x_m))$
  - h is independent to $x_1, \ldots, x_n$

Theory of Generalization

Formal definition

- Assume training and test data are both sampled from D
- The ideal function (for generating labels) is $f : f(x) \rightarrow y$
- Training error: Sample $x_1, \ldots, x_N$ from D and
  - $E_{tr}(h) = \frac{1}{N} \sum_{n=1}^{N} e(h(x_n), f(x_n))$
  - h is determined by $x_1, \ldots, x_n$
- Test error: Sample $x_1, \ldots, x_M$ from D and
  - $E_{te}(h) = \frac{1}{M} \sum_{m=1}^{M} e(h(x_m), f(x_m))$
  - h is independent to $x_1, \ldots, x_n$
- Generalization error = Test error = Expected performance on D:
  - $E(h) = \mathbb{E}_{x \sim D}[e(h(x), f(x))] = E_{te}(h)$

Theory of Generalization

The 2 questions of learning

- $E(h) \approx 0$ is achieved through:
  - $E(h) \approx E_{tr}(h)$ and $E_{tr}(h) \approx 0$

Theory of Generalization

The 2 questions of learning

- $E(h) \approx 0$ is achieved through:
  - $E(h) \approx E_{tr}(h)$ and $E_{tr}(h) \approx 0$
- Learning is split into 2 questions:
  - Can we make sure that $E(h) \approx E_{tr}(h)$?
    - Generalization
  - Can we make $E_{tr}(h)$ small?
    - Optimization

Theory of Generalization

Connection to Learning

- Given a function h
- If we randomly draw $x_1, \ldots, x_n$ (independent to h):
  - $E(h) = \mathbb{E}_{x \sim D}[h(x) \neq f(x)] \Leftrightarrow \mu$ (generalization error, unknown)
  - $\frac{1}{N}\sum_{n=1}^{N}[h(x_n) \neq y_n] \Leftrightarrow \nu$ (error on sampled data, known)
- Based on Hoeffding's inequality:
  - $p[\,|\nu - \mu| > \epsilon] \leq 2e^{-2\epsilon^2 N}$
- "$\mu = \nu$" Is probably approximately correct (PAC)
- However, this can only "verify" the error of a hypothesis:
  - h and $x_1, \ldots, x_N$ must be independent

Theory of Generalization

A simple solution

- For each particular h,
  - $P[\,|E_{tr}(h) - E(h)| > \epsilon] \leq 2e^{-2\epsilon^2 N}$
- If we have a hypothesis set $\mathscr{H}$,we want to derive the bound for $P[\text{sup}_{h\in\mathscr{H}}|E_{tr}(h) - E(h)| > \epsilon]$
  - $P[\,|E_{tr}(h_1) - E(h_1)| > \epsilon]$ or … or $P[\,|E_{tr}(h_{|\mathscr{H}|}) - E(h_{|\mathscr{H}|})| > \epsilon]$
  - $\leq \sum_{m=1}^{\mathscr{H}} P[\,|E_{tr}(h_m) - E(h_m)|\,] \leq 2|\mathscr{H}|e^{-2\epsilon^2 N}$
    - Because of union bound inequality $P(\bigcup_{i=1}^{\infty} A_i) \leq \sum_{i=1}^{\infty} P(A_i)$

Theory of generalization

When is learning successful?

- When our learning algorithm $\mathscr{A}$ picks the hypothesis g:
  - $P[\text{SUP}_{h \in \mathscr{H}} | E_{tr}(h) - E(h) | > \epsilon] \leq 2 | \mathscr{H} | e^{-2\epsilon^2 N}$
- If $|\mathscr{H}|$ is small and N is large enough:
  - If $\mathscr{A}$ finds $E_{tr}(g) \approx 0 \Rightarrow E(g) \approx 0$ (Learning is successful!)

Theory of Generalization

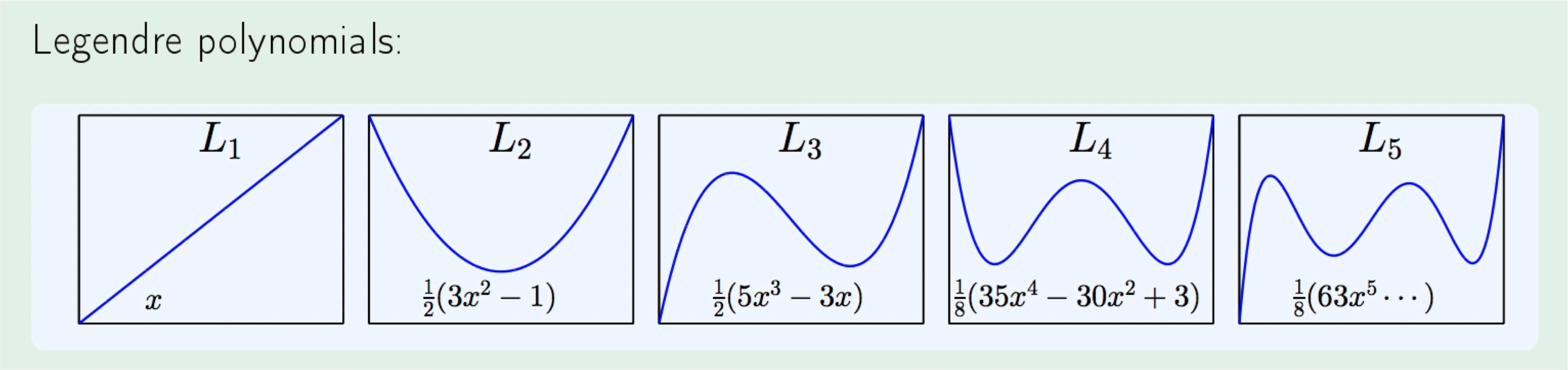
Feasibility of Learning

- $P[\,|E_{tr}(g) - E(g)| > \epsilon] \leq 2|\mathscr{H}|e^{-2\epsilon^2 N}$
  - Two questions:
    - 1. Can we make sure $E(g) \approx E_{tr}(g)$?
    - 2. Can we make sure $E_{tr}(g) \approx 0$?
- $|\mathscr{H}|$: complexity of model
  - Small $|\mathscr{H}|$: 1 holds, but 2 may not hold (too few choices) (under-fitting)
  - Large $|\mathscr{H}|$: 1 doesn't hold, but 2 may hold (over-fitting)

Regularization

The polynomial model

- $\mathcal{H}_Q$: polynomials of order Q
  - $\mathcal{H}_Q = \{ \sum_{q=0}^{Q} w_q L_q(x) \}$
- Linear regression in the $\mathcal{Z}$ space with
  - $z = [1, L_1(x), \ldots, L_Q(x)]$



Regularization

Unconstrained solution

- Input $(x_1, y_1), \ldots, (x_N, y_N) \to (z_1, y_1), \ldots, (z_N, y_N)$
- Linear regression:
  - Minimize: $E_{\text{tr}}(w) = \dfrac{1}{N}\sum_{n=1}^{N}(w^T z_n - y_n)^2$
  - Minimize: $\dfrac{1}{N}(Zw - y)^T(Zw - y)$
- Solution $w_{\text{tr}} = (Z^T Z)^{-1} Z^T y$

Regularization

Constraining the weights

- Hard constraint: $\mathscr{H}_2$ is constrained version of $\mathscr{H}_{10}$ (with $w_q = 0$ for $q > 2$)

Regularization

Constraining the weights

- Hard constraint: $\mathscr{H}_2$ is constrained version of $\mathscr{H}_{10}$ (with $w_q = 0$ for $q > 2$)
- Soft-order constraint: $\sum_{q=0}^{Q} w_q^2 \leq C$

Regularization

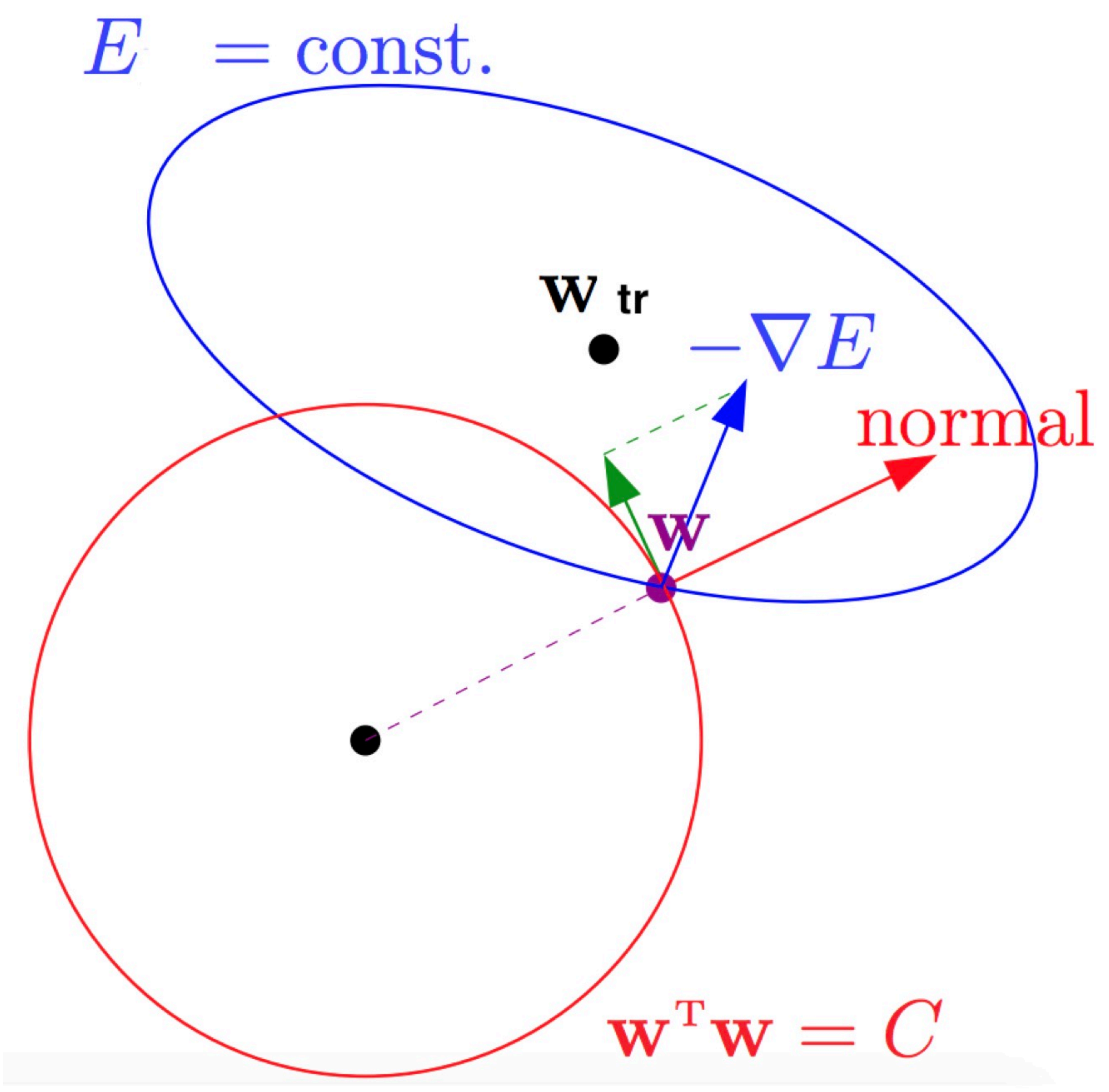
Constraining the weights

- Hard constraint: $\mathcal{H}_2$ is constrained version of $\mathcal{H}_{10}$ (with $w_q = 0$ for $q > 2$)
- Soft-order constraint: $\sum_{q=0}^{Q} w_q^2 \leq C$
- The problem given soft-order constraint:
  - Minimize $\frac{1}{N}(Zw - y)^T(Zw - y)$ s.t. $\underbrace{w^T w \leq C}_{\text{smaller hypothesis space}}$
- Solution w_{reg} instead of w_{tr}

Regularization

Equivalent to the unconstrained version

- Constrained version:
  - $\min_{w} E_{\text{tr}}(w) = \frac{1}{N}(Zw - y)^T(Zw - y)$
    - s.t. $w^T w \leq C$



- Optimal when
  - $\nabla E_{\text{tr}}(w_{\text{reg}}) \propto -w_{\text{reg}}$
  - Why? If $-\nabla E_{\text{tr}}(w_{\text{reg}})$ and w are not parallel, can decrease $E_{\text{tr}}(w)$ without violating the constraint

Regularization

Equivalent to the unconstrained version

- Constrained version:
  - $\min_{w} E_{\text{tr}}(w) = \frac{1}{N}(Zw - y)^T(Zw - y)$ s.t. $w^T w \leq C$
- Optimal when
  - $\nabla E_{\text{tr}}(w_{\text{reg}}) \propto -w_{\text{reg}}$
- Assume $\nabla E_{\text{tr}}(w_{\text{reg}}) = -2\frac{\lambda}{N}w_{\text{reg}} \Rightarrow \nabla E_{\text{tr}}(w_{\text{reg}}) + 2\frac{\lambda}{N}w_{\text{reg}} = 0$

Regularization

Equivalent to the unconstrained version

- Constrained version:
  - $\min_w E_{\text{tr}}(w) = \frac{1}{N}(Zw - y)^T(Zw - y)$ s.t. $w^T w \le C$
- Optimal when
  - $\nabla E_{\text{tr}}(w_{\text{reg}}) \propto - w_{\text{reg}}$
- Assume $\nabla E_{\text{tr}}(w_{\text{reg}}) = -2\frac{\lambda}{N}w_{\text{reg}} \Rightarrow \nabla E_{\text{tr}}(w_{\text{reg}}) + 2\frac{\lambda}{N}w_{\text{reg}} = 0$
- w_{reg} is also the solution of unconstrained problem
  - $\min_w E_{\text{tr}}(w) + \frac{\lambda}{N}w^T w$ (Ridge regression!)

Regularization

Equivalent to the unconstrained version

- Constrained version:
  - $\min_{w} E_{\text{tr}}(w) = \frac{1}{N}(Zw - y)^T(Zw - y)$ s.t. $w^T w \le C$
- Optimal when
  - $\nabla E_{\text{tr}}(w_{\text{reg}}) \propto -w_{\text{reg}}$
- Assume $\nabla E_{\text{tr}}(w_{\text{reg}}) = -2\frac{\lambda}{N} w_{\text{reg}} \Rightarrow \nabla E_{\text{tr}}(w_{\text{reg}}) + 2\frac{\lambda}{N} w_{\text{reg}} = 0$
- w_{reg} is also the solution of unconstrained problem
  - $\min_{w} E_{\text{tr}}(w) + \frac{\lambda}{N} w^T w$ (Ridge regression!) $C \uparrow$ $\lambda \downarrow$

Regularization

Ridge regression solution

- $\min_{w} E_{\text{reg}}(w) = \frac{1}{N}\left((Zw - y)^T(Zw - y) + \lambda w^T w \right)$
- $\nabla E_{\text{reg}}(w) = 0 \Rightarrow Z^T Z(w - y) + \lambda w = 0$

Regularization

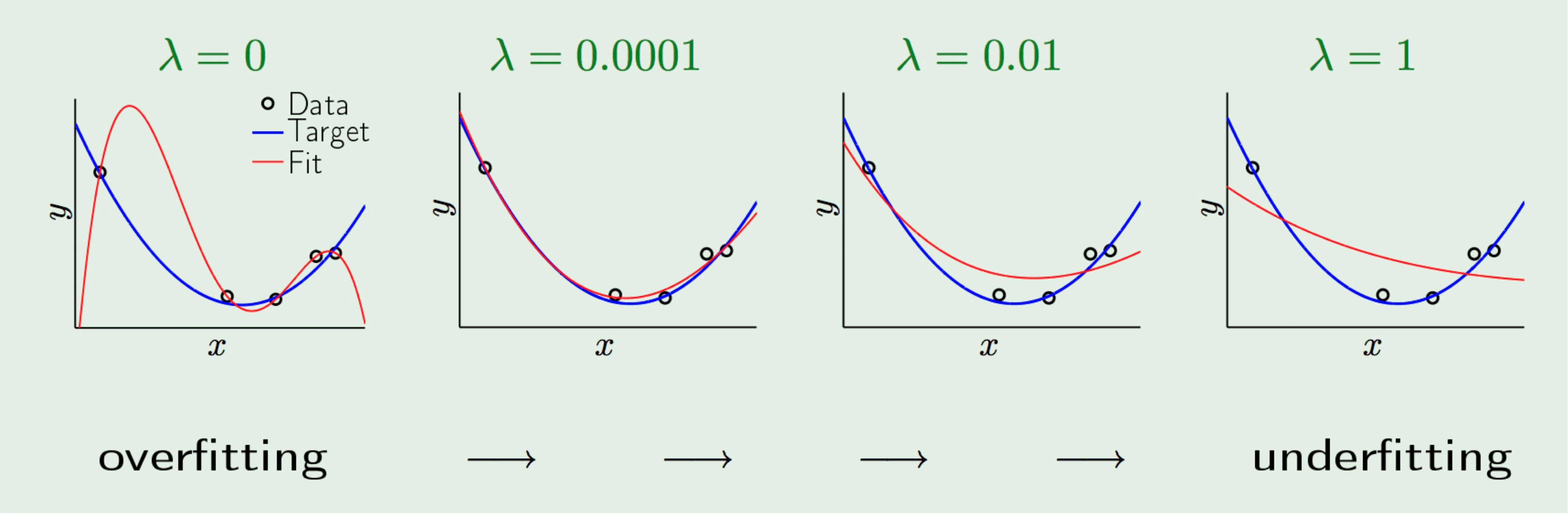
Ridge regression solution

- $\min_{w} E_{\text{reg}}(w) = \frac{1}{N}\left((Zw - y)^T(Zw - y) + \lambda w^T w\right)$
- $\nabla E_{\text{reg}}(w) = 0 \Rightarrow Z^T Z(w - y) + \lambda w = 0$
- So, $w_{\text{reg}} = (Z^T Z + \lambda I)^{-1} Z^T y$ (with regularization) as opposed to $w_{\text{tr}} = (Z^T Z)^{-1} Z^T y$ (without regularization)

Regularization

The result

- $\min_{w} E_{\text{tr}}(w) + \frac{\lambda}{N} w^T w$



Regularization

Equivalent to “weight decay”

- Consider the general case
  - $\min_{w} E_{\text{tr}}(w) + \frac{\lambda}{N} w^T w$

Regularization

Equivalent to "weight decay"

- Consider the general case
   - $\min_{w} E_{\text{tr}}(w) + \frac{\lambda}{N} w^T w$
- Gradient descent:

$$w_{t+1} = w_t - \eta(\nabla E_{\text{tr}}(w_t) + 2\frac{\lambda}{N} w_t)$$

   - $$= w_t \underbrace{(1 - 2\eta\frac{\lambda}{N})}_{\text{weight decay}} - \eta \nabla E_{\text{tr}}(w_t)$$

Regularization

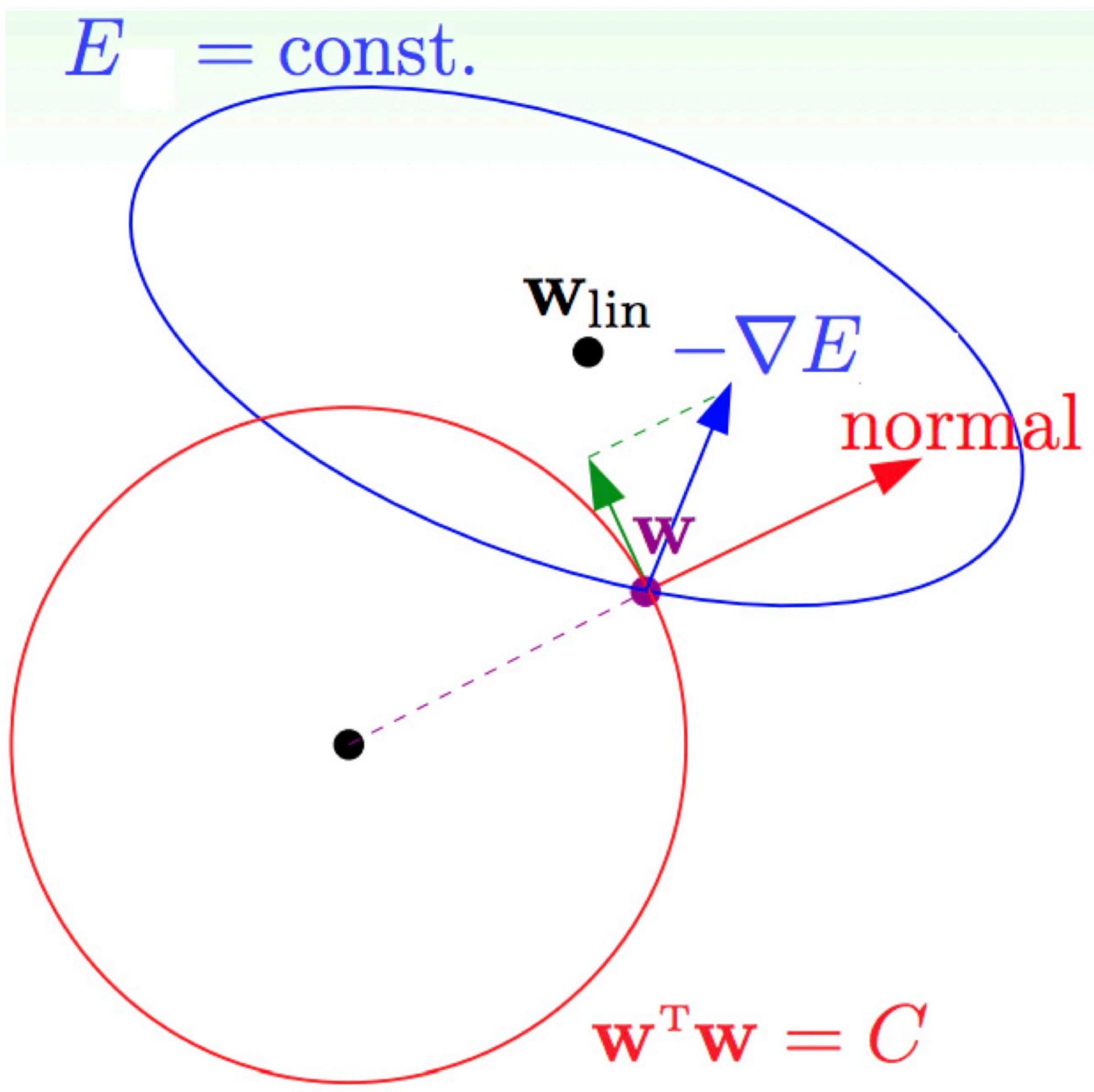
Variations of weight decay

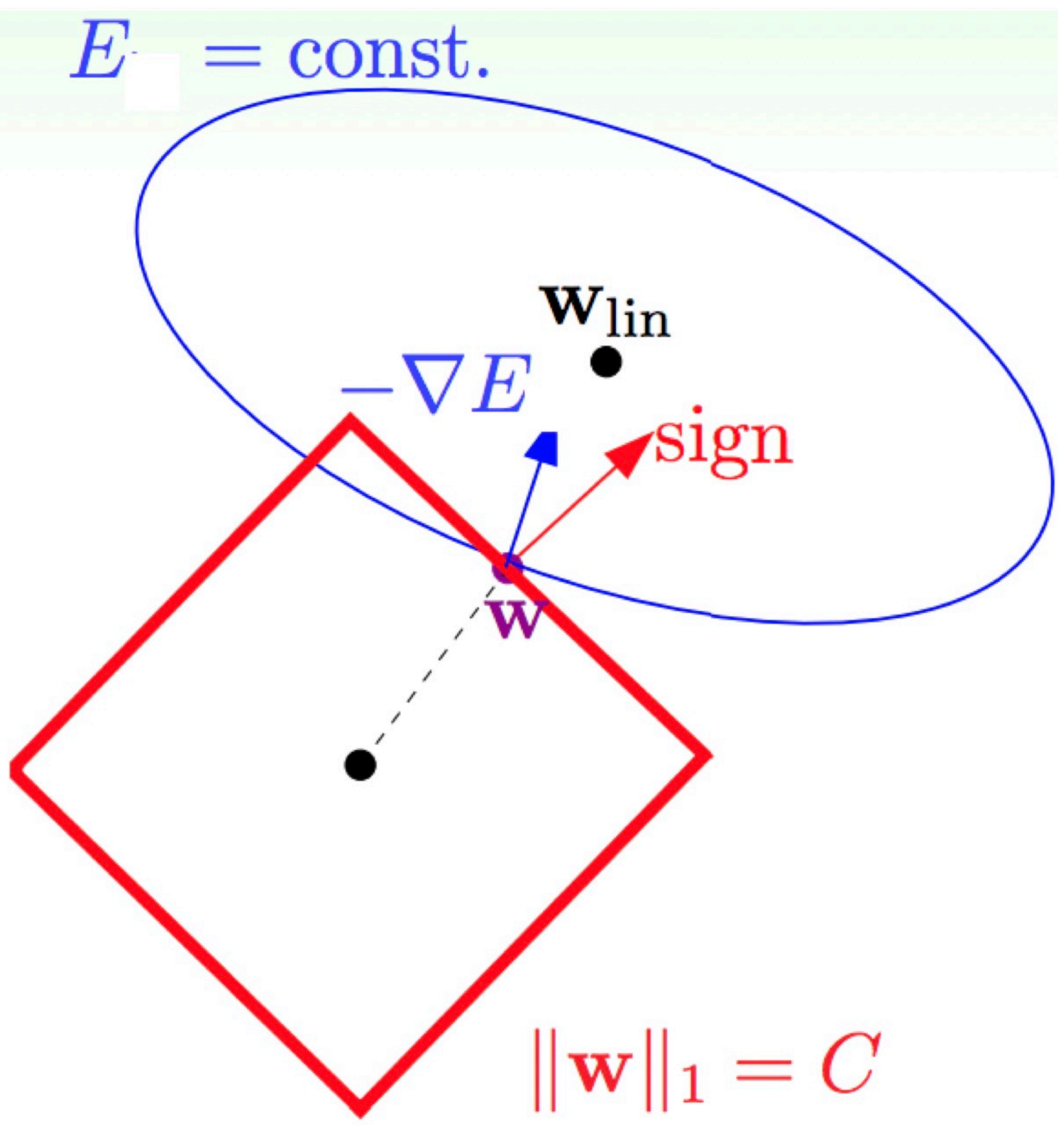
- Calling the regularizer $\Omega = \Omega(h)$, we minimize
  - $E_{\text{reg}}(h) = E_{\text{tr}}(h) + \frac{\lambda}{N}\Omega(h)$
- In general, $\Omega(h)$ can be any measurement for the "size" of h

Regularization

L2 vs L1 regularizer

- L1-regularizer: $\Omega(w) = \|w\|_1 = \sum_q |w_q|$
- Usually leads to a sparse solution (only few w_q will be nonzero)

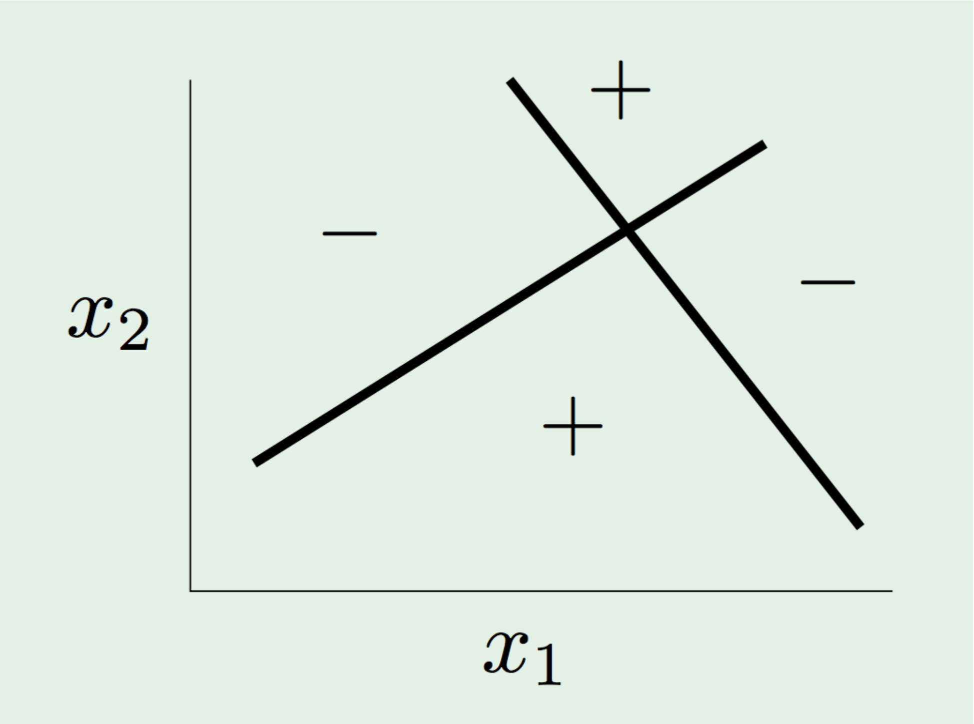




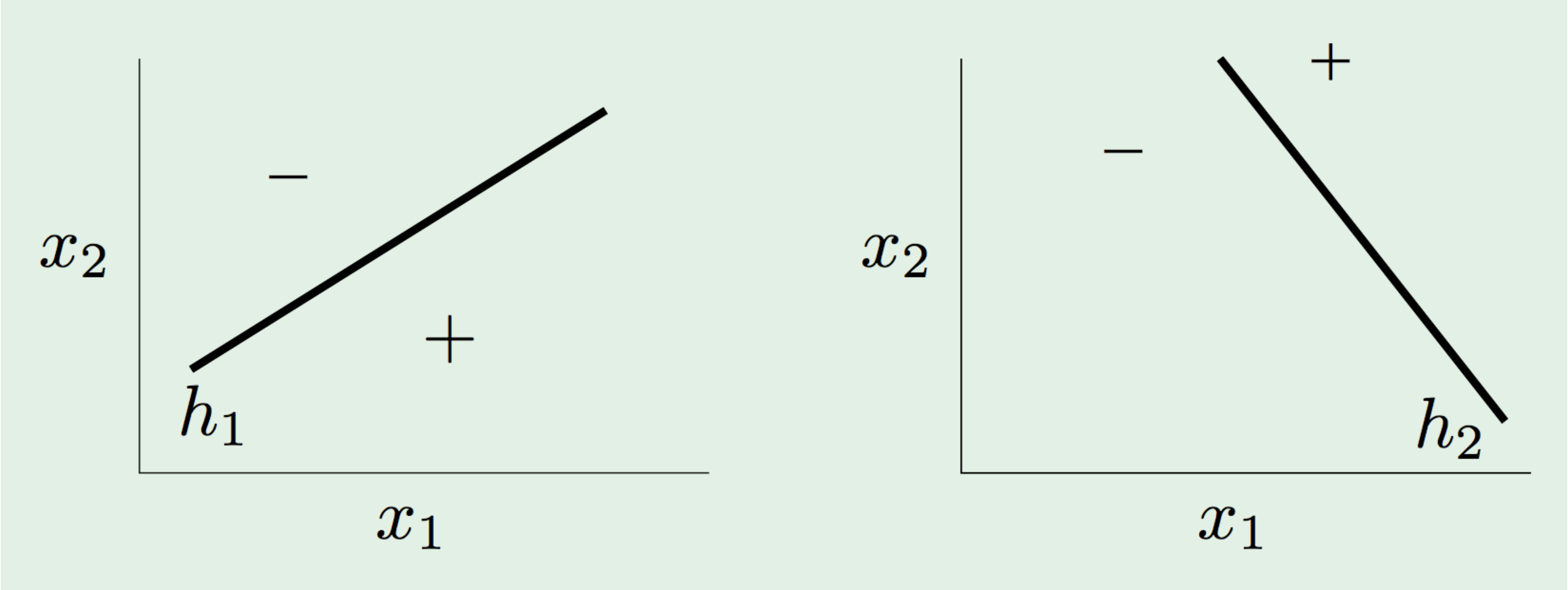
Neural network

Another way to introduce nonlinearity

- How to generate this nonlinear hypothesis?



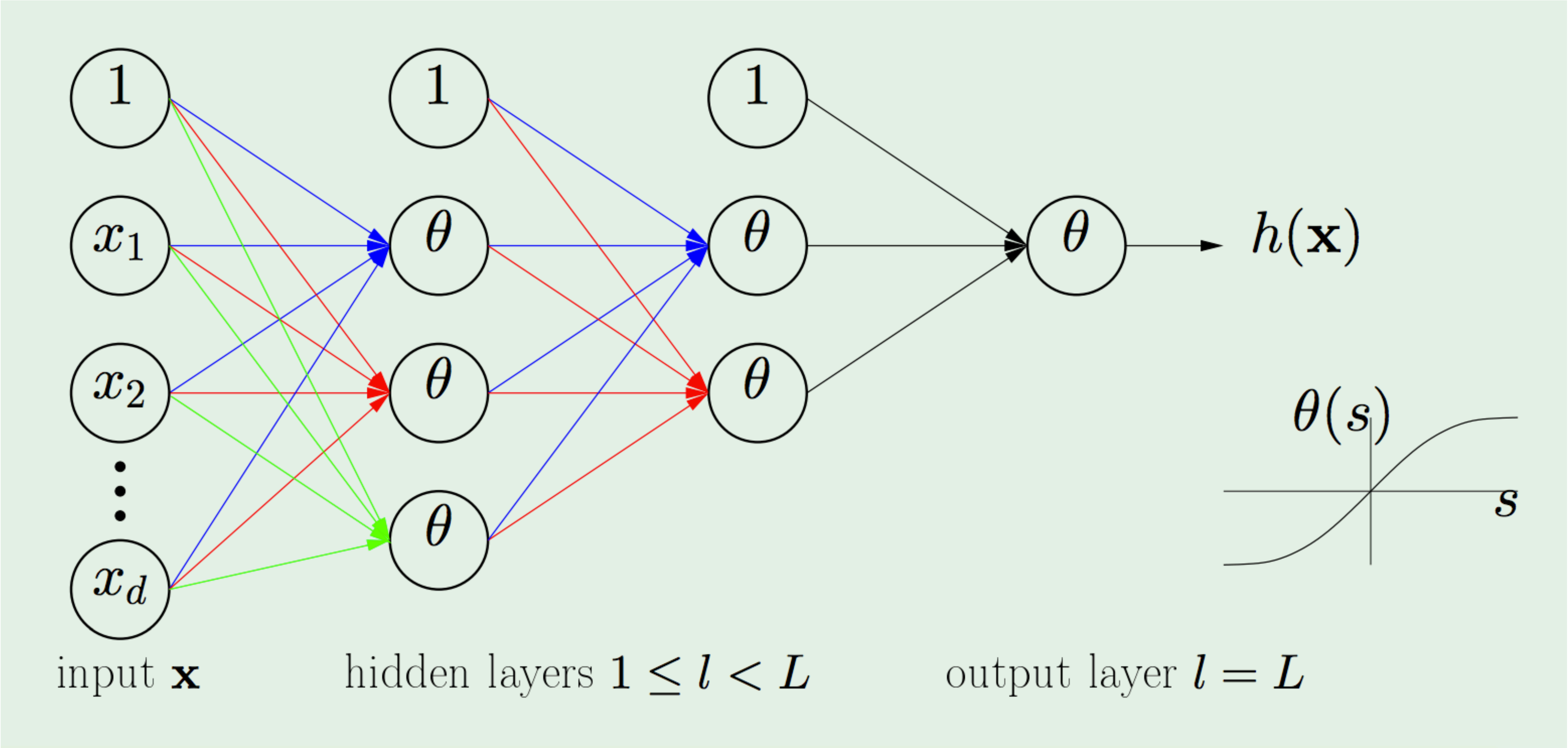
- Combining multiple linear hyperplanes to construct nonlinear hypothesis



Neural Network

Definition

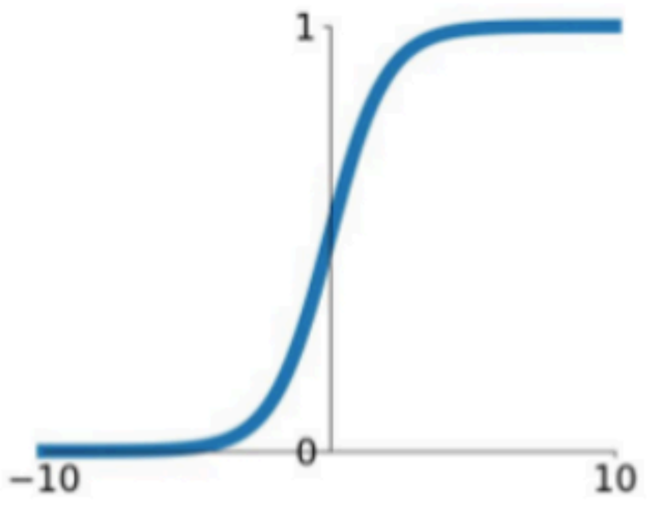
- Input layer: d neurons (input features)
- Neurons from layer 1 to L: Linear combination of previous layers + activation function
  - $\theta(w^T x)$, $\quad \theta$: activation function
- Final layer: one neuron $\Rightarrow$ prediction by sign$(h(x))$



Neural network
Activation Function

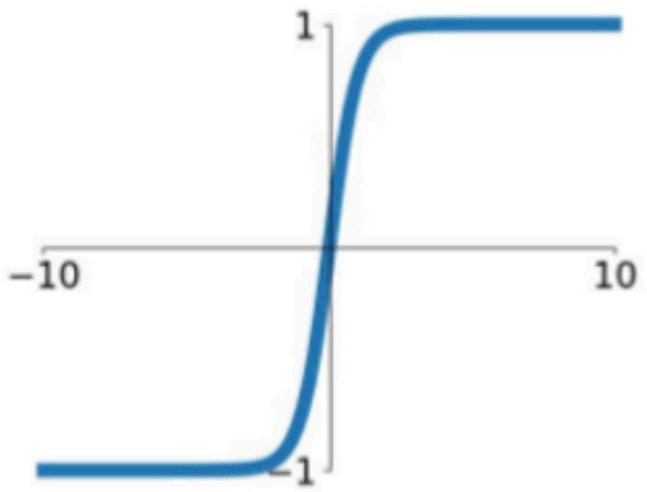
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



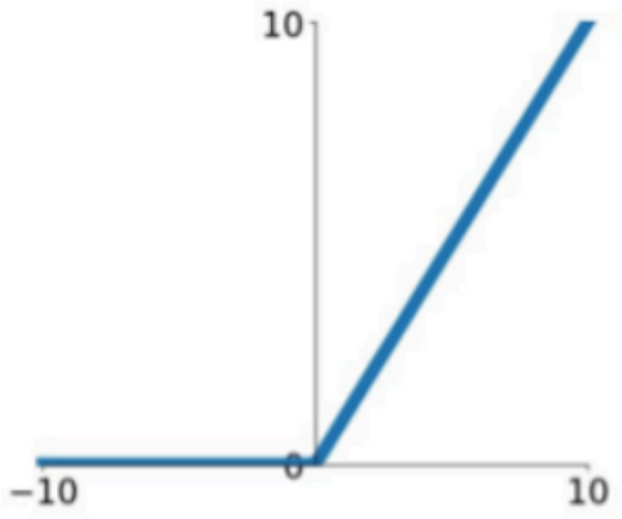
tanh

$\tanh(x)$



ReLU

$\max(0, x)$



Neural Network

Activation: Formal Definitions

- Weight: $w_{ij}^{(l)}$ $\begin{cases} 1 \le l \le L & \text{: layers} \\ 0 \le i \le d^{(l-1)} & \text{: inputs} \\ 1 \le j \le d^{(l)} & \text{: outputs} \end{cases}$

  bias: $b_j^{(l)}$: added to the j-th neuron in the l-th layer

Neural Network

Formal Definitions

- Weight: $w_{ij}^{(l)}$ $\begin{cases} 1 \le l \le L & \text{: layers} \\ 0 \le i \le d^{(l-1)} & \text{: inputs} \\ 1 \le j \le d^{(l)} & \text{: outputs} \end{cases}$

  bias: $b_j^{(l)}$: added to the j-th neuron in the l-th layer
- j-th neuron in the l-the layer:
- $$x_j^{(l)} = \theta(s_j^{(l)}) = \theta\left(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)} + b_j^{(l)}\right)$$

Neural Network

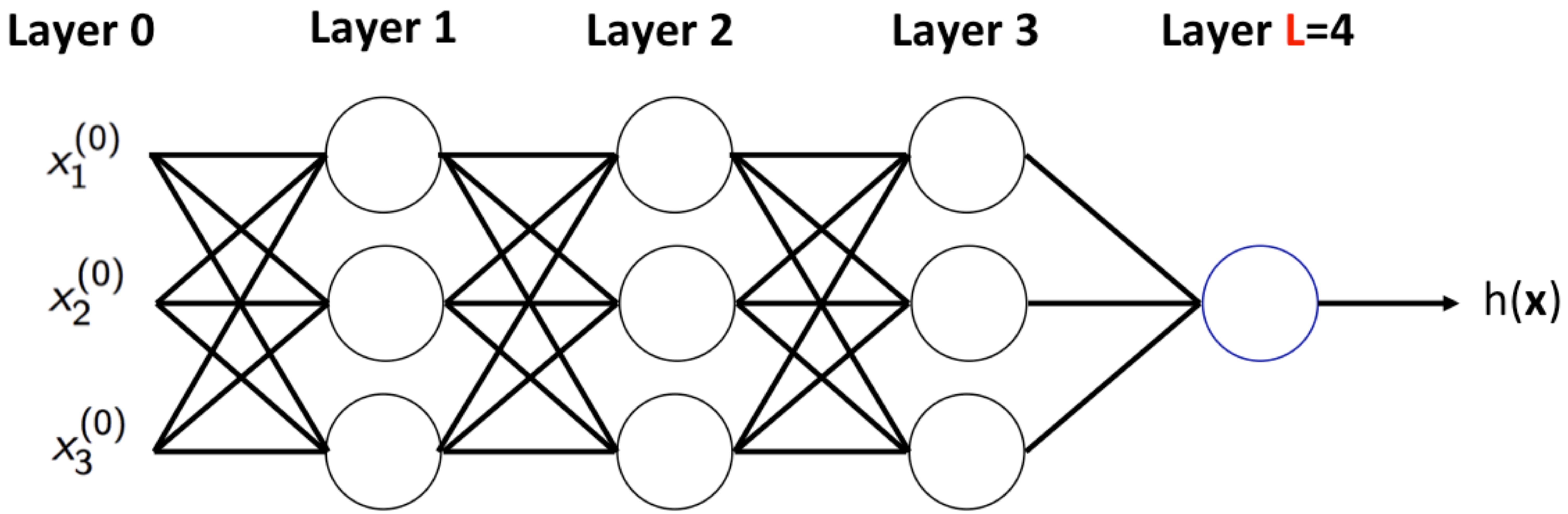
Formal Definitions

- Weight: $w_{ij}^{(l)}$ $\begin{cases} 1 \leq l \leq L & : \text{layers} \\ 0 \leq i \leq d^{(l-1)} & : \text{inputs} \\ 1 \leq j \leq d^{(l)} & : \text{outputs} \end{cases}$

  bias: $b_j^{(l)}$: added to the j-th neuron in the l-th layer
- j-th neuron in the l-the layer:
  - $x_j^{(l)} = \theta(s_j^{(l)}) = \theta(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)} + b_j^{(l)})$
- Output:
  - $h(x) = x_1^{(L)}$

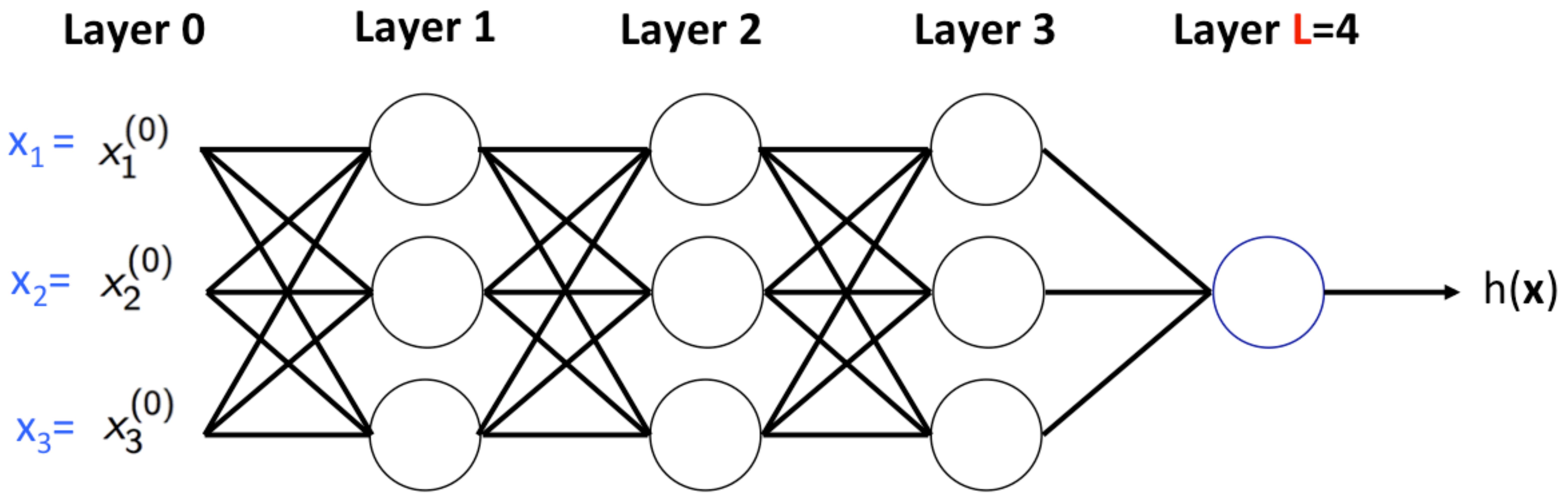
Neural Network

Forward propagation



Neural Network

Forward propagation

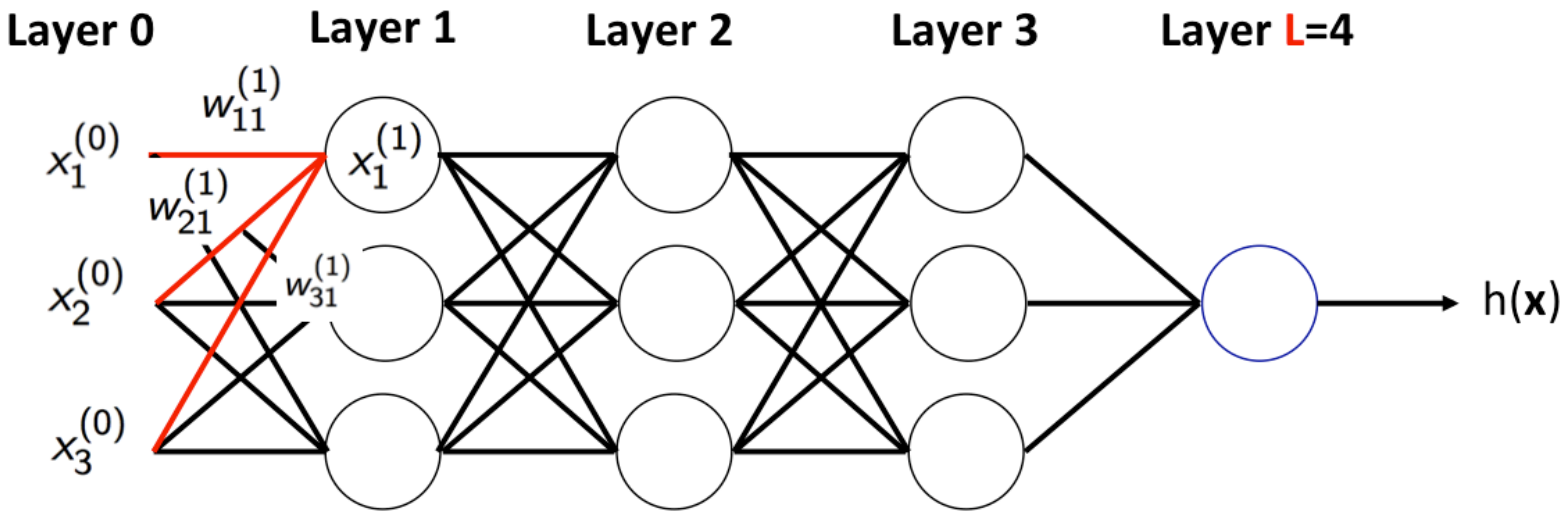


features for one data point

$\mathbf{x} = [x_1, x_2, x_3]$

Neural Network

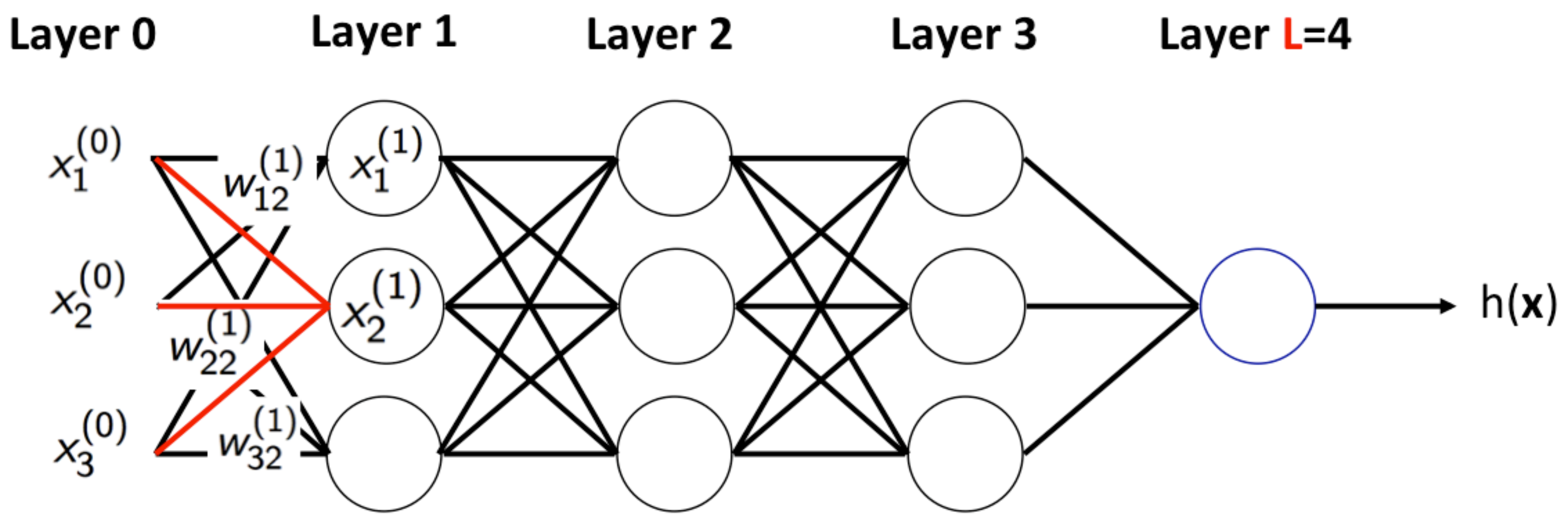
Forward propagation



$$x_1^{(1)} = \theta(\sum_{i=1}^{3} w_{i1}^{(1)} x_i^{(0)})$$

Neural Network

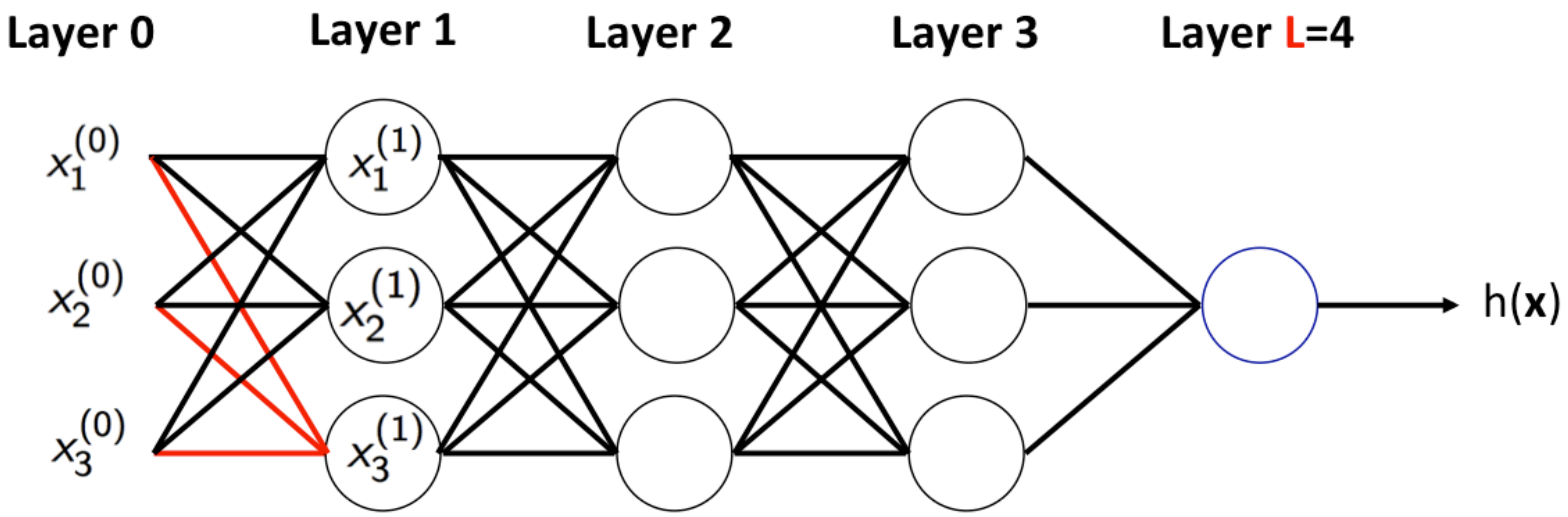
Forward propagation



$$x_2^{(1)} = \theta(\sum_{i=1}^{3} w_{i2}^{(1)} x_i^{(0)})$$

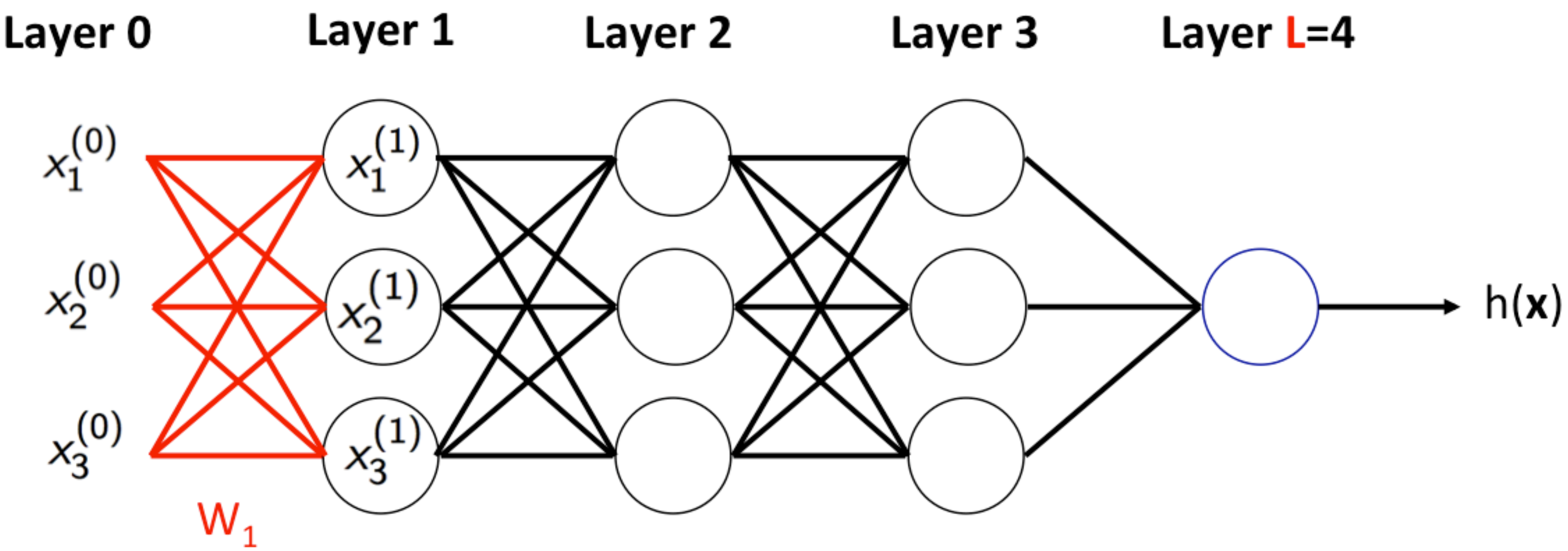
Neural Network

Forward propagation



Neural Network

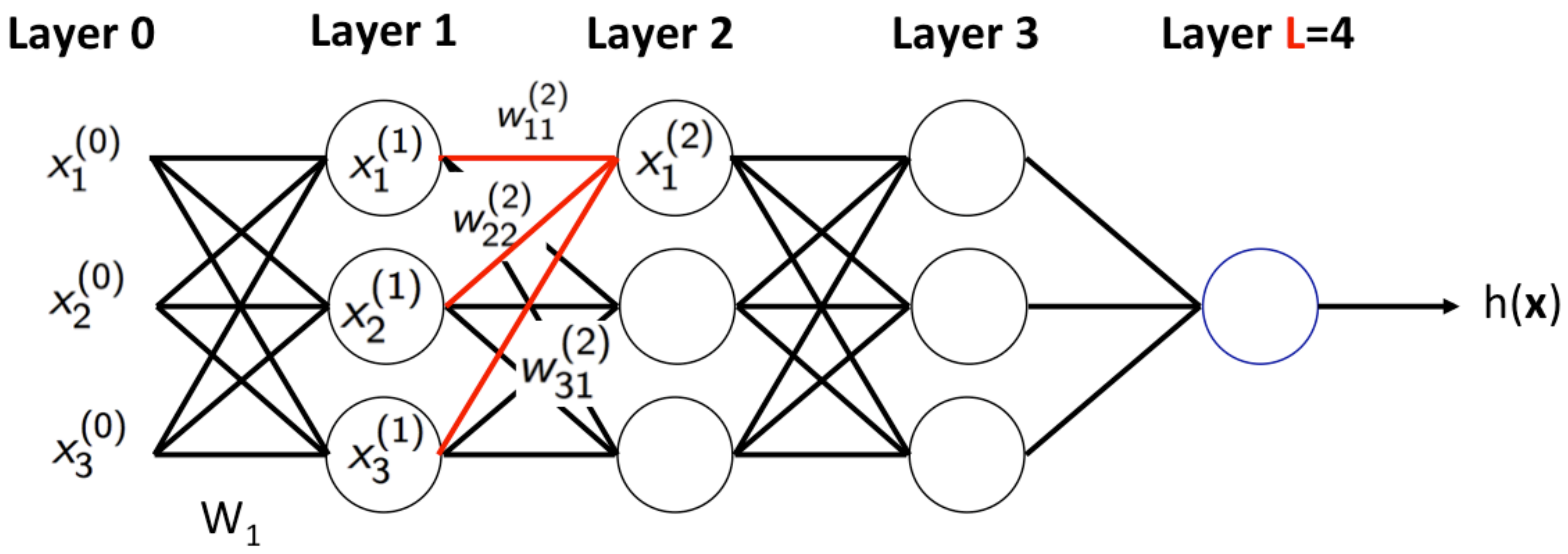
Forward propagation



$$\boldsymbol{x}^{(1)} = \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix} = \theta\left(\begin{bmatrix} w_{11}^{(1)} & w_{21}^{(1)} & w_{31}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & w_{32}^{(1)} \\ w_{13}^{(1)} & w_{23}^{(1)} & w_{33}^{(1)} \end{bmatrix} \times \begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \\ x_3^{(0)} \end{bmatrix} \right) = \theta(W_1 \boldsymbol{x}^{(0)})$$

Neural Network

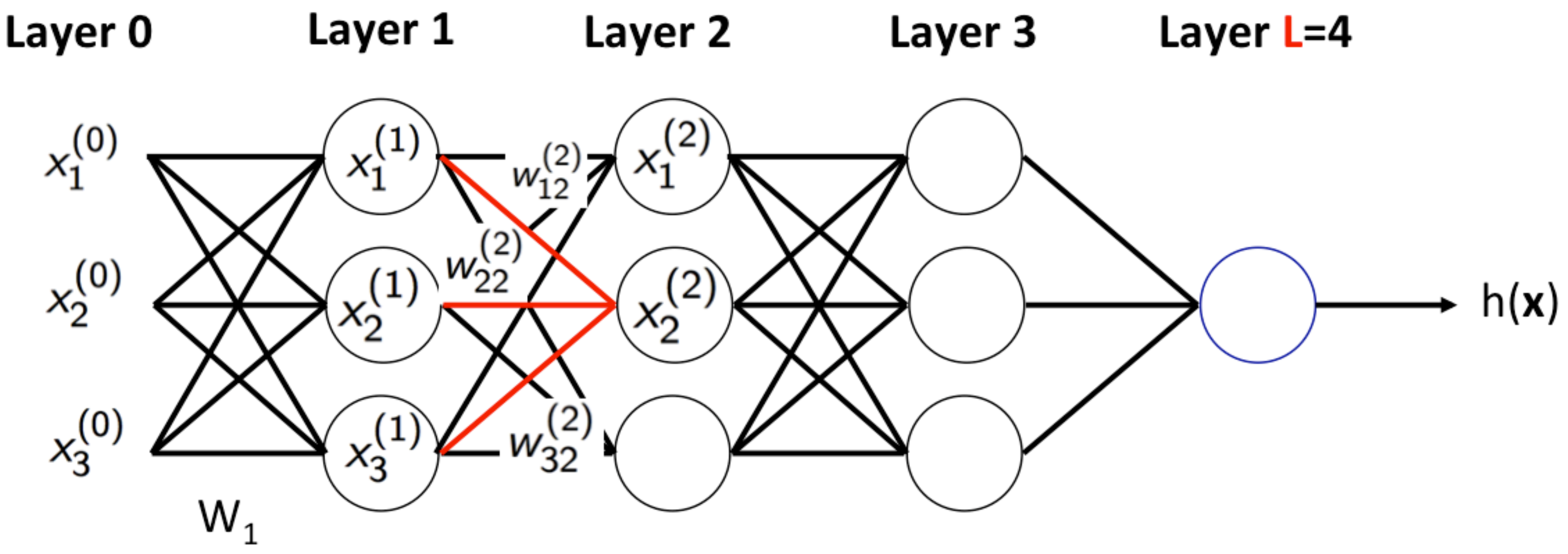
Forward propagation



$$x_1^{(2)} = \theta(\sum_{i=1}^{3} w_{i1}^{(2)} x_i^{(1)})$$

Neural Network

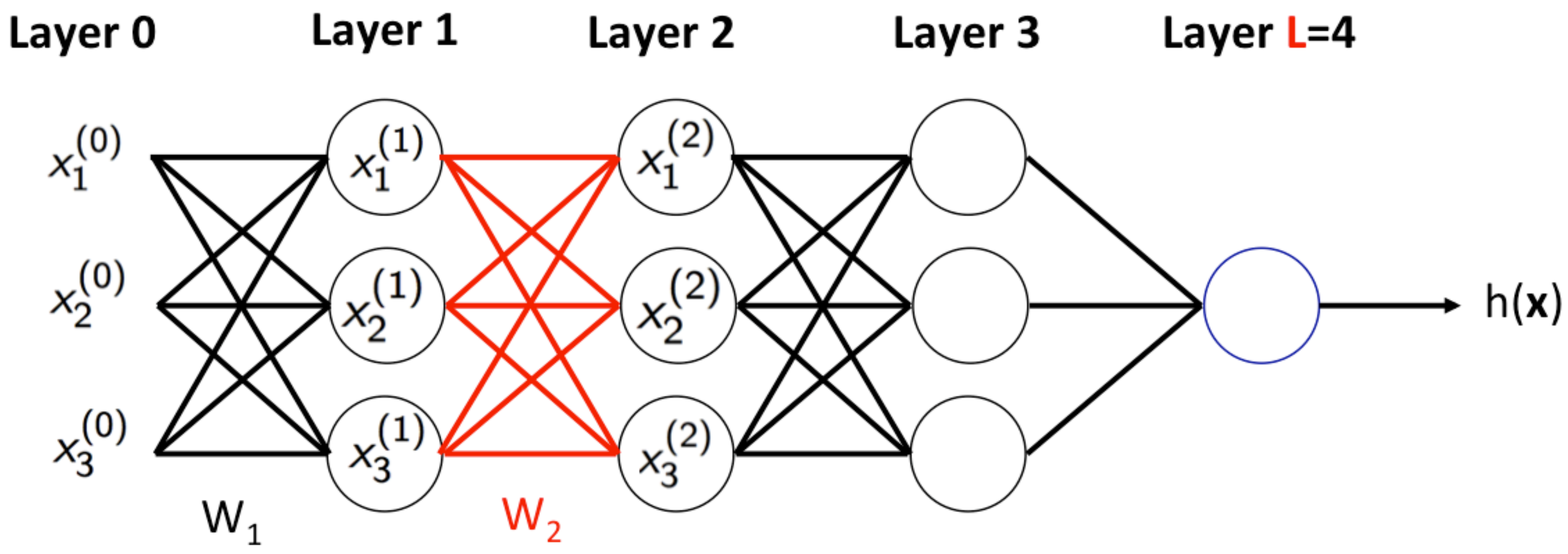
Forward propagation



$$x_2^{(2)} = \theta(\sum_{i=1}^{3} w_{i2}^{(2)} x_i^{(1)})$$

Neural Network

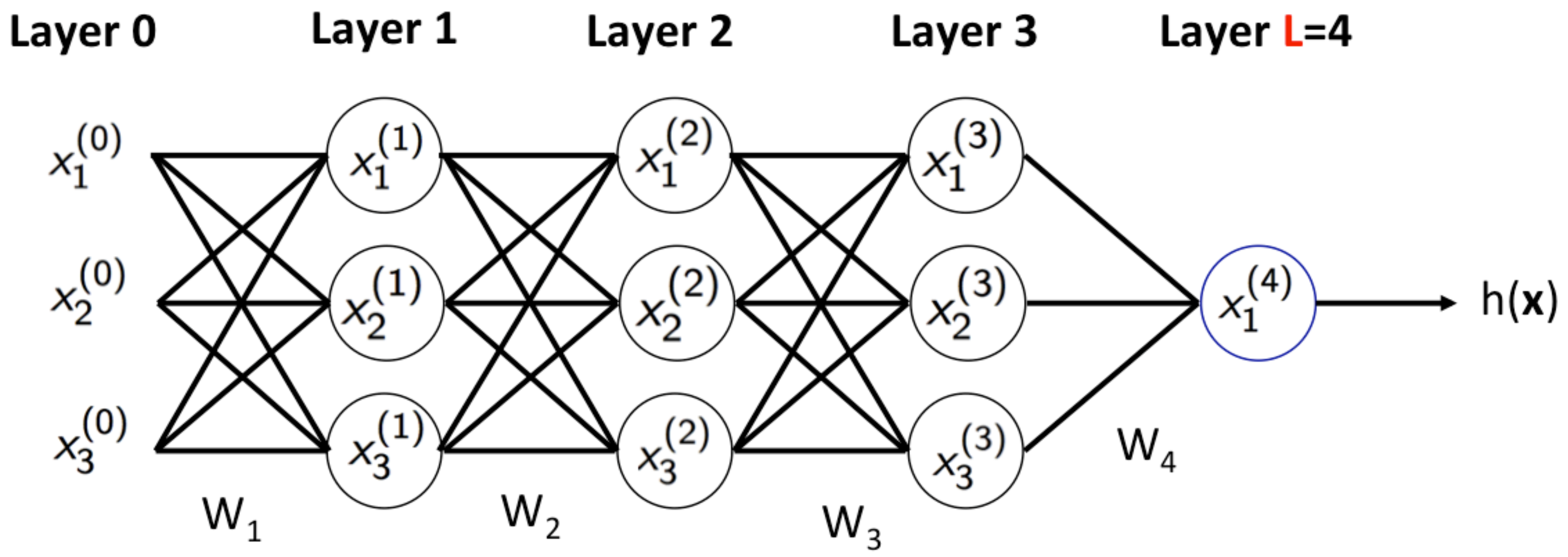
Forward propagation



$$\boldsymbol{x}^{(2)} = \begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \end{bmatrix} = \theta\left(\begin{bmatrix} w_{11}^{(2)} & w_{21}^{(2)} & w_{31}^{(2)} \\ w_{12}^{(2)} & w_{22}^{(2)} & w_{32}^{(2)} \\ w_{13}^{(2)} & w_{23}^{(2)} & w_{33}^{(2)} \end{bmatrix} \times \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix} \right) = \theta(W_2 \boldsymbol{x}^{(1)})$$

Neural Network

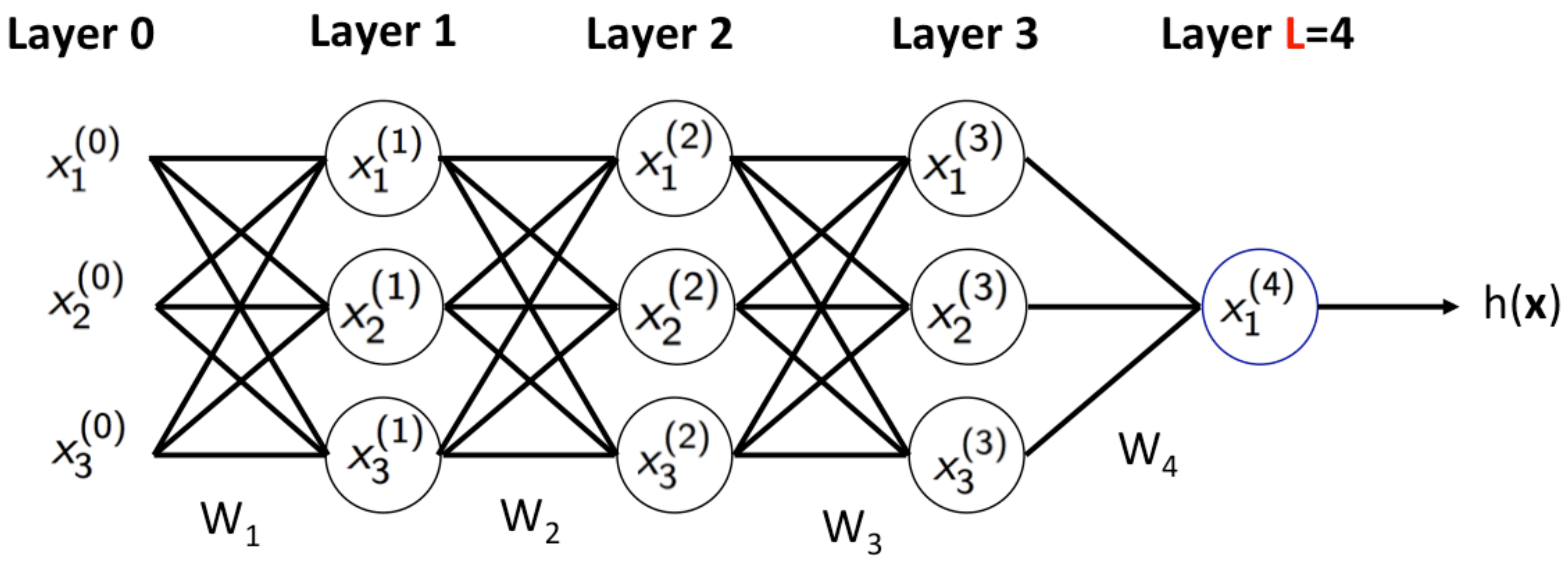
Forward propagation



$$
\begin{aligned}
h(\boldsymbol{x}) = x_1^{(4)} &= \theta(W_4\boldsymbol{x}^{(3)}) = \theta(W_4\theta(W_3\boldsymbol{x}^{(2)})) \\
&= \cdots = \theta(W_4\theta(W_3\theta(W_2\theta(W_1\boldsymbol{x}))))
\end{aligned}
$$

Neural Network
Forward propagation

- With the bias term:
  $h(x) = \theta(W_4\theta(W_3\theta(W_2\theta(W_1x + b_1) + b_2) + b_3) + b_4)$



$$h(\boldsymbol{x}) = x_1^{(4)} = \theta(W_4\boldsymbol{x}^{(3)}) = \theta(W_4\theta(W_3\boldsymbol{x}^{(2)}))$$
$$= \cdots = \theta(W_4\theta(W_3\theta(W_2\theta(W_1\boldsymbol{x}))))$$

Neural Network

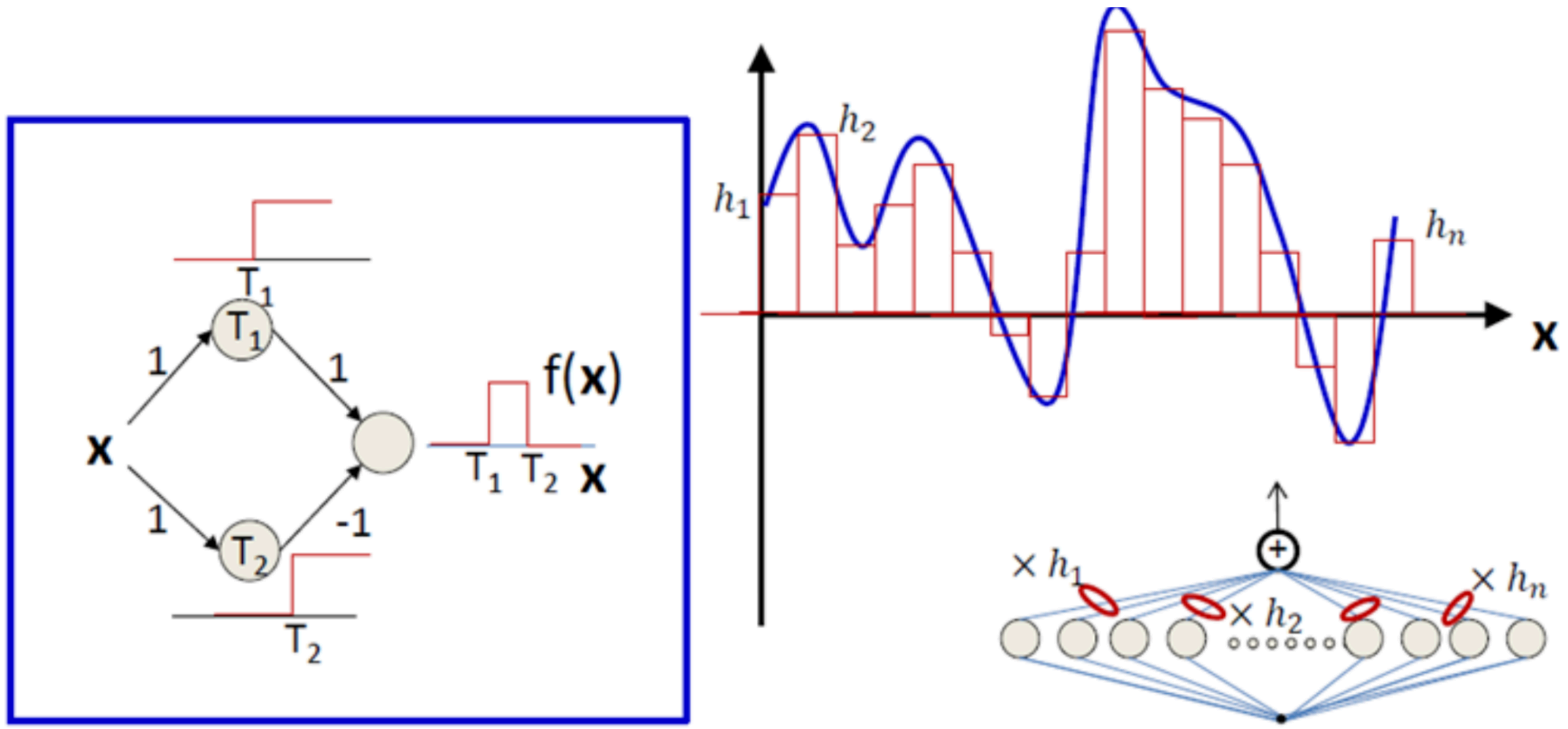
Capacity of neural networks

- Universal approximation theorem (Horink, 1991):
    - "A neural network with single hidden layer can approximate any continuous function arbitrarily well, given enough hidden units"
- True for commonly used activations (ReLU, sigmoid, …)

Neural Network

Universal approximation for step activation

- How to approximate an arbitrary function by single-layer NN with step function as activation:



2 hidden units to form a "rectangle" **any function can be approximated by rectangles**

(figure from https://medium.com/analytics-vidhya)

Neural Network

Training

- Weights $W = \{W_1, \ldots, W_L\}$ and bias $\{b_1, \cdots, b_L\}$ determine $h(x)$
- Learning the weights: solve ERM with SGD
- Loss on example (x_n, y_n) is
  - $e(h(x_n), y_n) = e(W)$

Neural Network

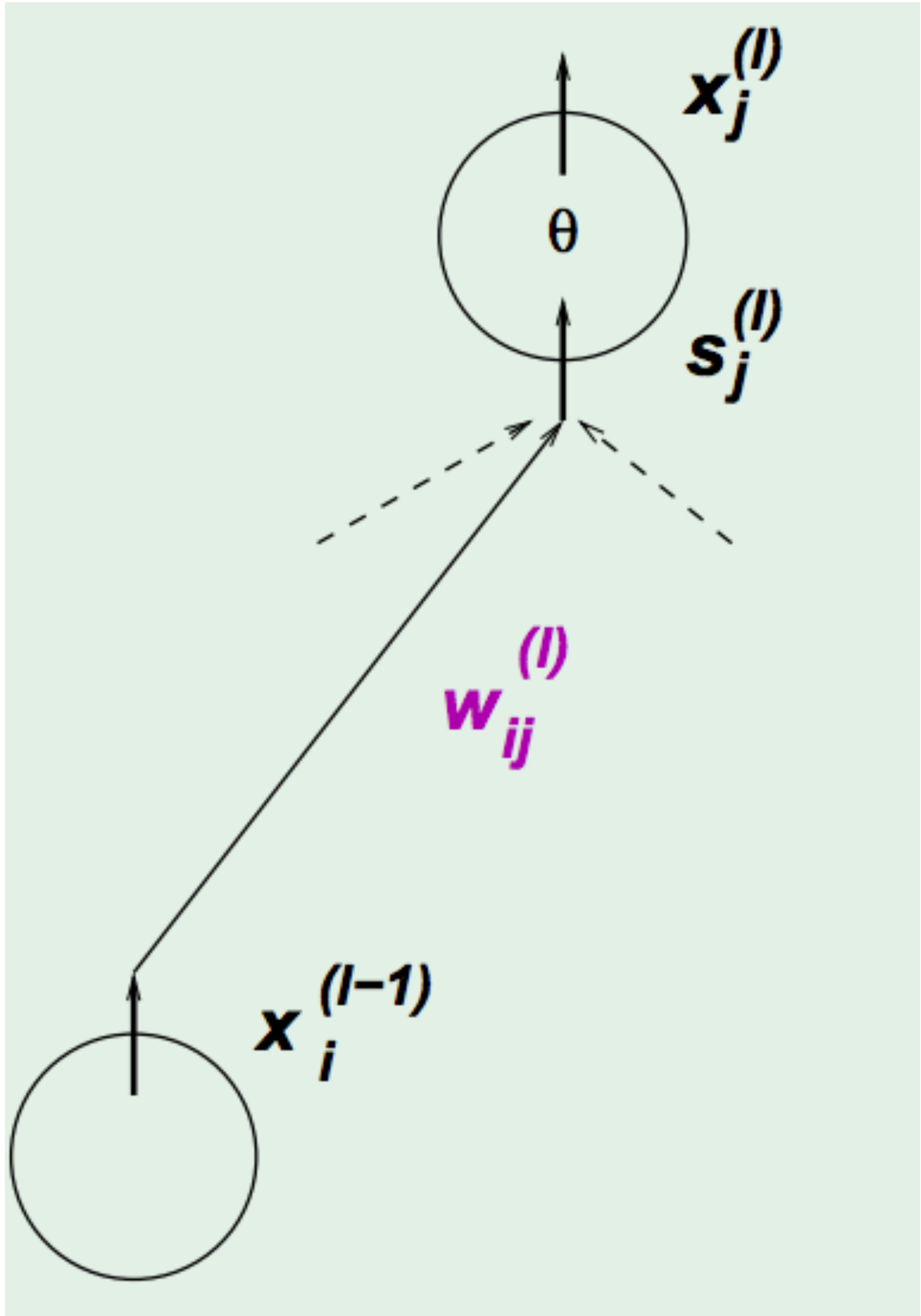
Training

- Weights $W = \{W_1, \ldots, W_L\}$ and bias $\{b_1, \cdots, b_L\}$ determine $h(x)$
- Learning the weights: solve ERM with SGD
- Loss on example (x_n, y_n) is
  - $e(h(x_n), y_n) = e(W)$
- To implement SGD, we need the gradient
  - $\nabla e(W) : \{\frac{\partial e(W)}{\partial w_{ij}^{(l)}}\}$ for all i, j, l (for simplicity we ignore bias in the derivations)

Neural Network

Computing Gradient $\frac{\partial e(W)}{\partial w_{ij}^{(l)}}$

- Use chain rule:
  - $$\frac{\partial e(W)}{\partial w_{ij}^{(l)}} = \frac{\partial e(W)}{\partial s_j^{(l)}} \times \frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}}$$
  - $$s_j^{(l)} = \sum_{i=1}^{d} x_i^{(l-1)} w_{ij}^{(l)}$$
- We have $\frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}} = x_i^{(l-1)}$



Neural Network

Computing Gradient $\frac{\partial e(W)}{\partial w_{ij}^{(l)}}$

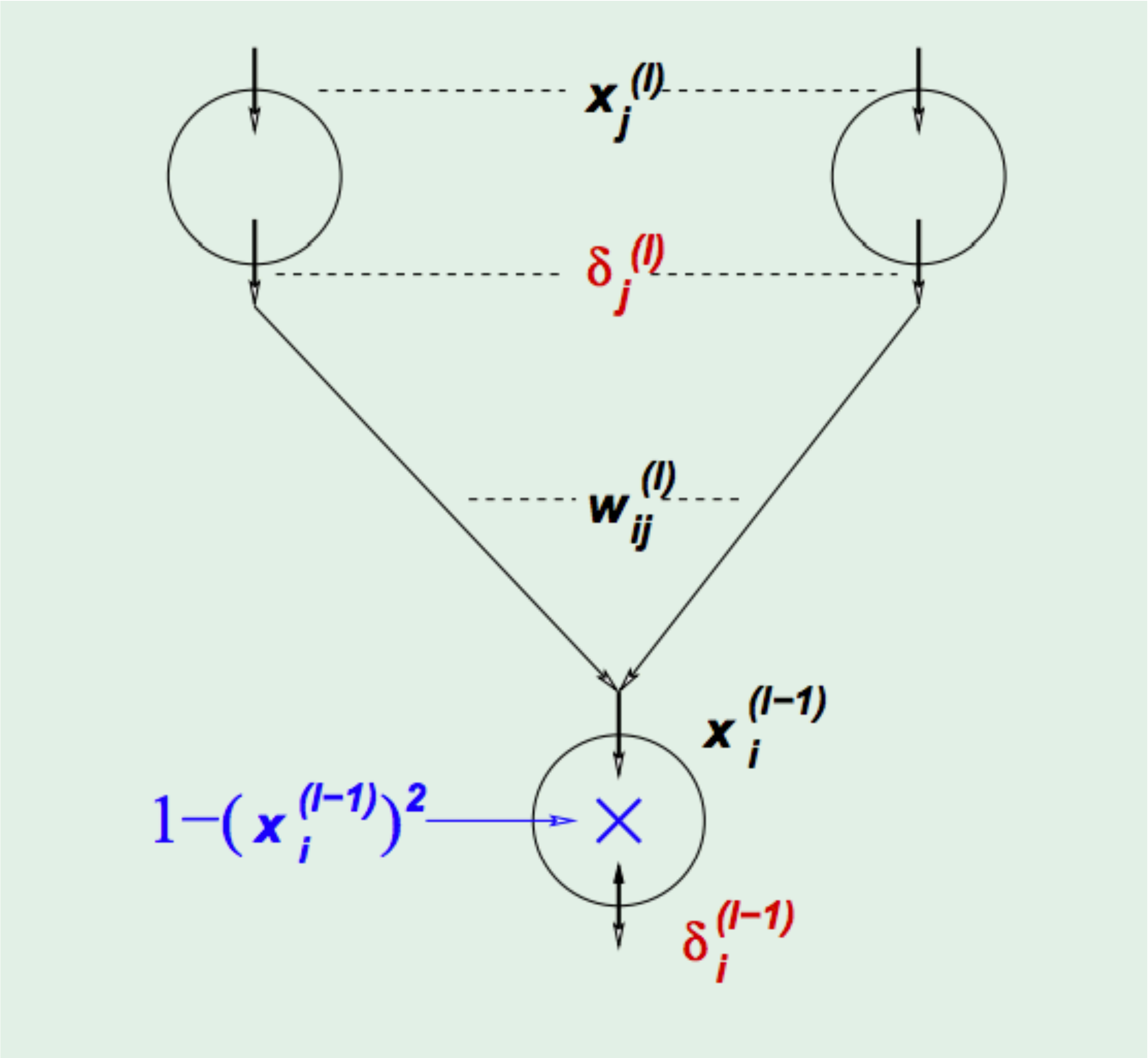
- Define $\delta_j^{(l)} := \frac{\partial e(W)}{\partial s_j^{(l)}}$
- Compute by layer-by-layer:

$$\delta_i^{(l-1)} = \frac{\partial e(W)}{\partial s_i^{(l-1)}}$$

$$= \sum_{j=1}^{d} \frac{\partial e(W)}{\partial s_j^{(l)}} \times \frac{\partial s_j^{(l)}}{\partial x_i^{(l-1)}} \times \frac{\partial x_i^{(l-1)}}{\partial s_i^{l-1}}$$

- $$= \sum_{j=1}^{d} \delta_j^{(l)} \times w_{ij}^{(l)} \times \theta'(s_i^{(l-1)}),$$
- where $\theta'(s) = 1 - \theta^2(s)$ for tan

- $\delta_i^{(l-1)} = (1 - (x_i^{(l-1)})^2) \sum_{j=1}^{d} w_{ij}^{(l)} \delta_j^{(l)}$



Neural Network

Final layer

- (Assume square loss)
  - $e(W) = (x_1^{(L)} - y_n)^2$
  - $x_1^{(L)} = \theta(s_1^{(L)})$
- So,

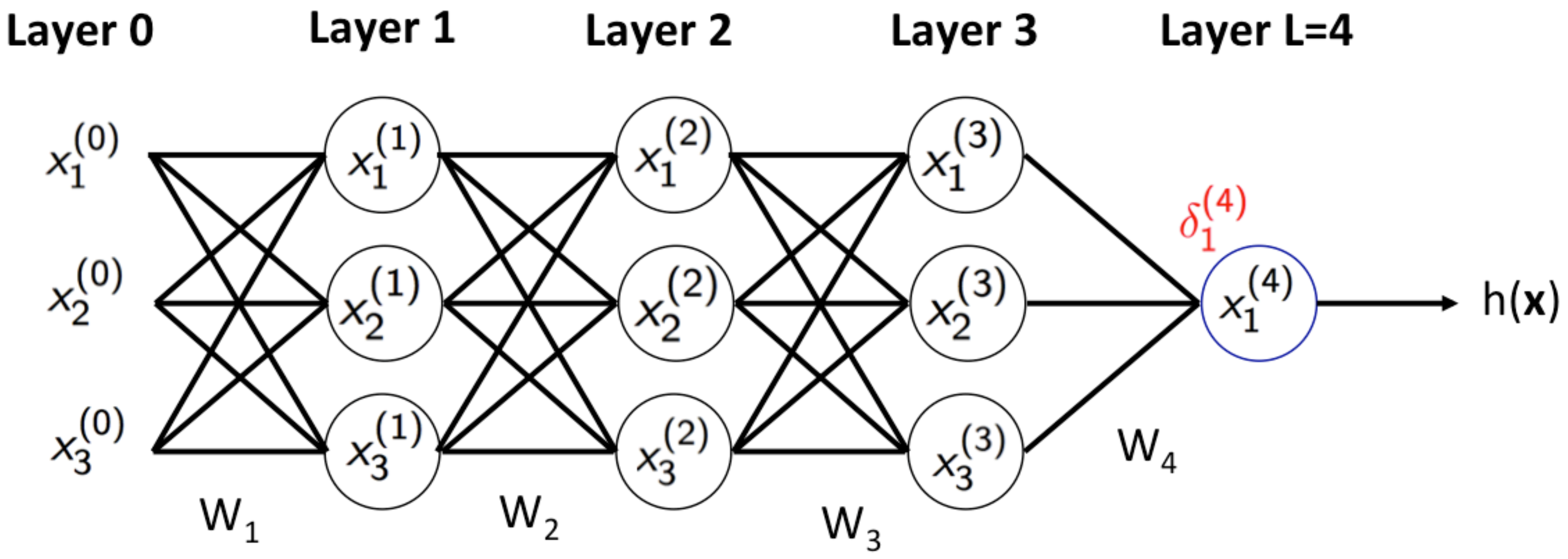
$$\delta_1^{(L)} = \frac{\partial e(W)}{\partial s_1^{(L)}}$$

- $$= \frac{\partial e(W)}{\partial x_1^{(L)}} \times \frac{\partial x_1^{(L)}}{\partial s_1^{(L)}}$$

$$= 2(x_1^{(L)} - y_n) \times \theta'(s_1^{(L)})$$

Neural Network

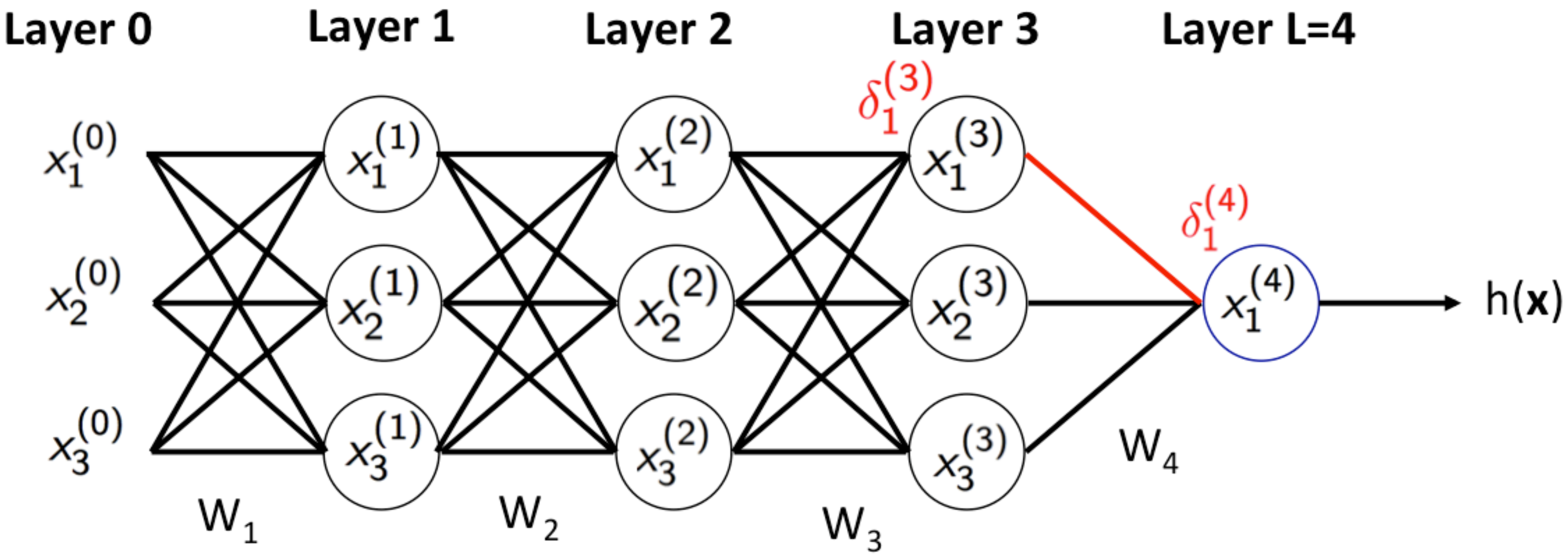
Backward propagation



$$\delta_1^{(4)} = 2(x_1^{(4)} - y_n) \times (1 - (x_1^{(4)})^2)$$

Neural Network

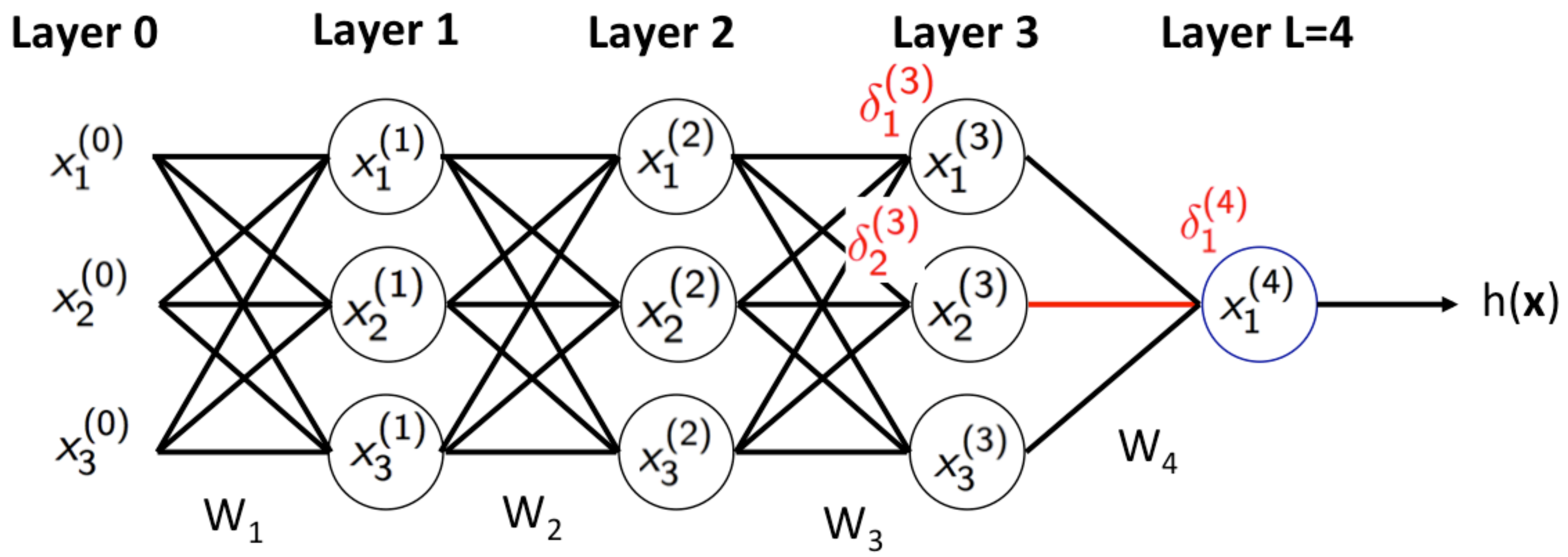
Backward propagation



$$\delta_1^{(3)} = (1 - (x_1^{(3)})^2) \times \delta_1^{(4)} \times w_{11}^{(4)}$$

Neural Network

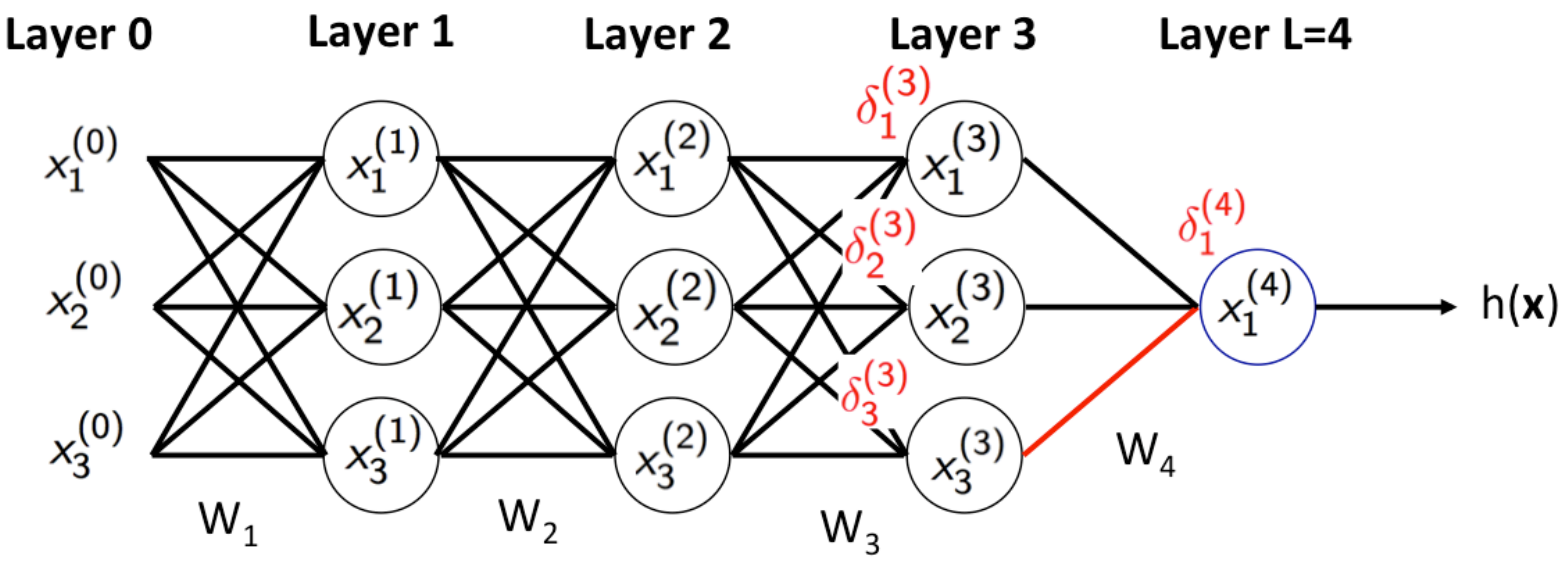
Backward propagation



$$\delta_2^{(3)} = (1 - (x_2^{(3)})^2) \times \delta_1^{(4)} \times w_{21}^{(4)}$$

Neural Network

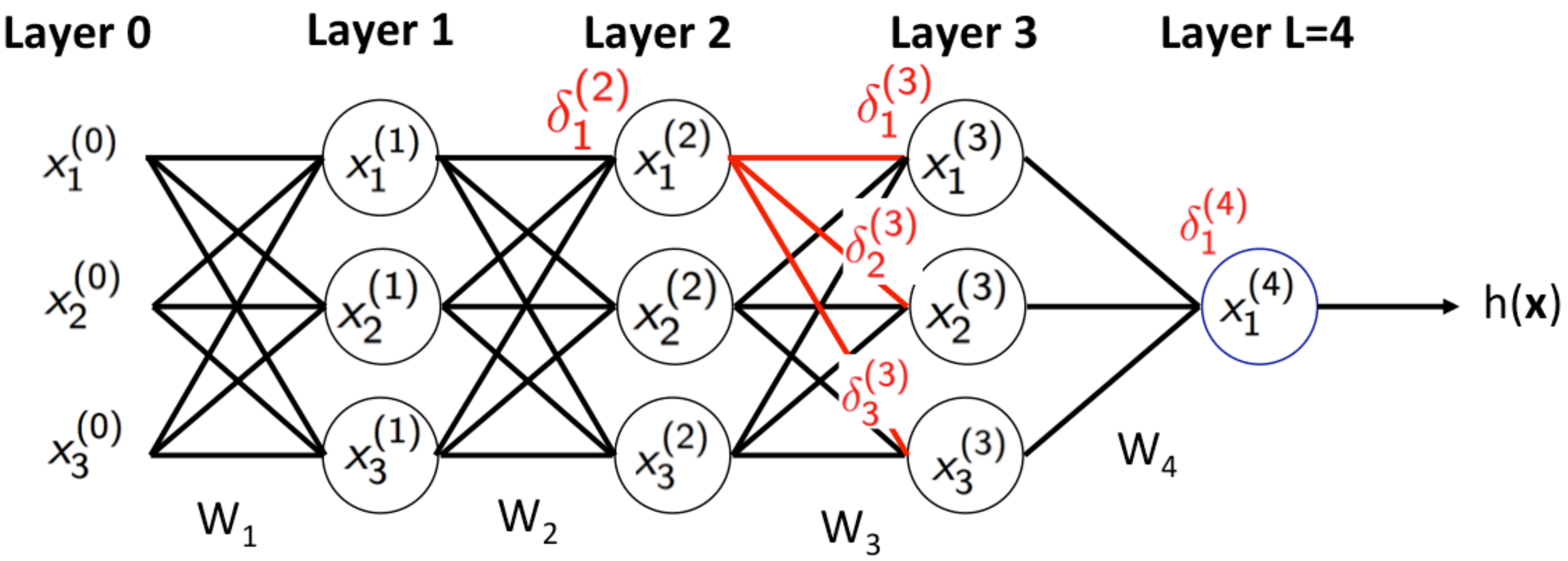
Backward propagation



$$\delta_3^{(3)} = (1 - (x_3^{(3)})^2) \times \delta_1^{(4)} \times w_{31}^{(4)}$$

Neural Network

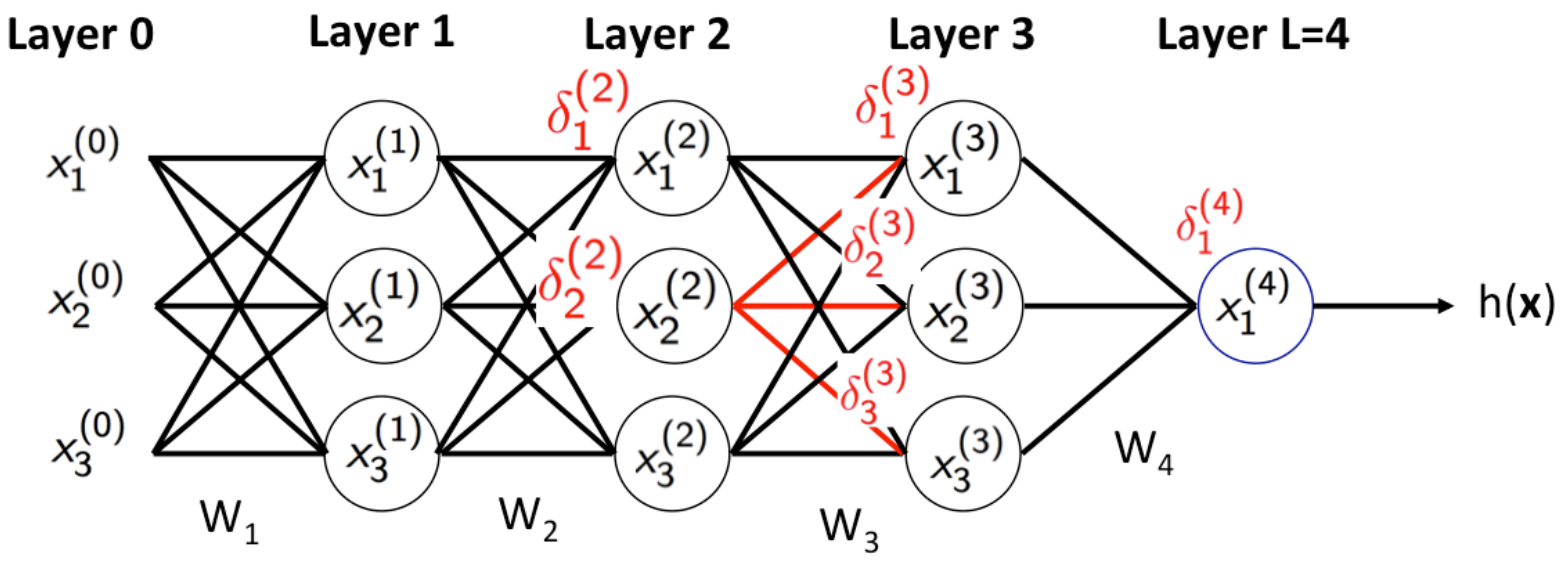
Backward propagation



$$\delta_1^{(2)} = (1 - (x_1^{(2)})^2) \sum_{j=1}^{3} \delta_j^{(3)} w_{1j}^{(3)}$$

Neural Network

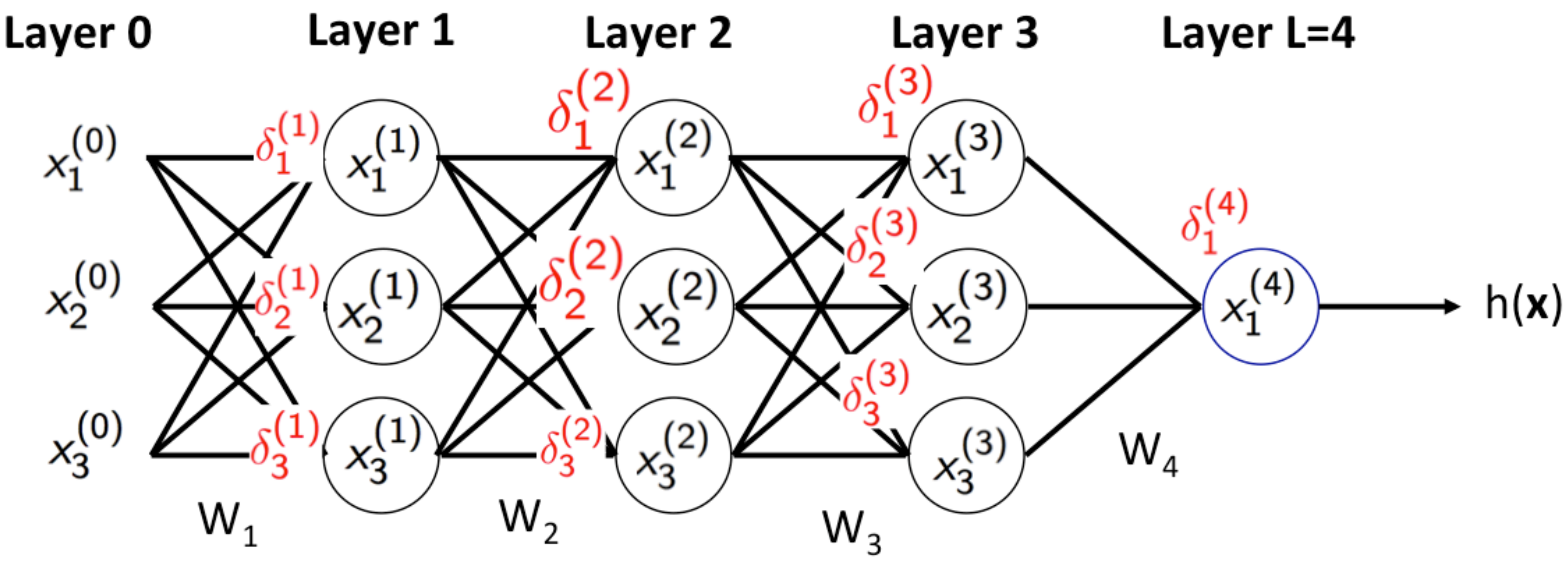
Backward propagation



$$\delta_2^{(2)} = (1 - (x_2^{(2)})^2) \sum_{j=1}^{3} \delta_j^{(3)} w_{2j}^{(3)}$$

Neural Network

Backward propagation

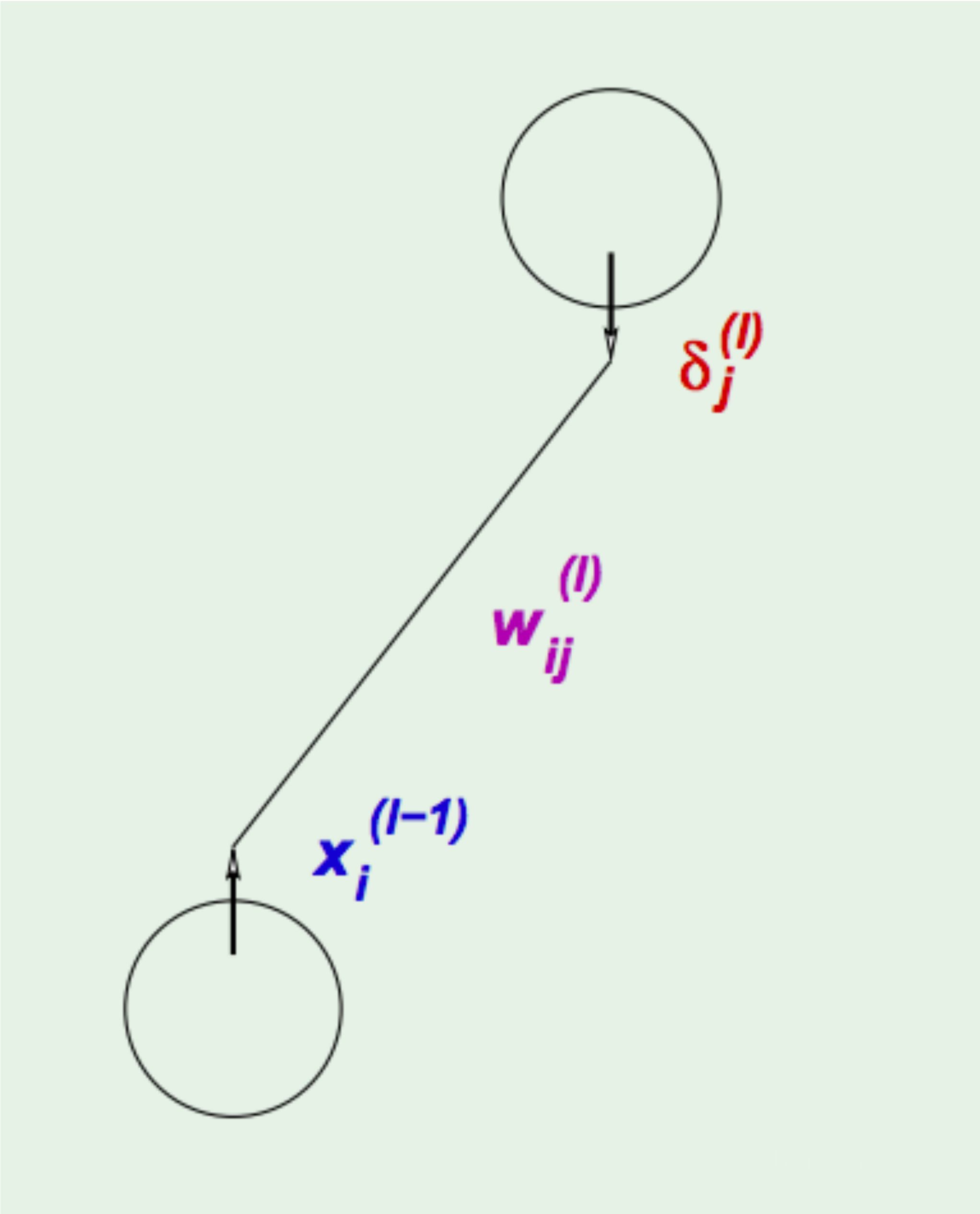


Neural Network

Backpropagation

SGD for neural networks

- Initialize all weights $w_{ij}^{(l)}$ **at random**
- For iter $= 0, 1, 2, \cdots$
  - Forward: Compute all $x_j^{(l)}$ from input to output
  - Backward: Compute all $\delta_j^{(l)}$ from output to input
  - Update all the weights $w_{ij}^{l} \leftarrow w_{ij}^{(l)} - \eta x_i^{(l-1)} \delta_j^{(l)}$



Neural Network

Backpropagation

- Just an automatic way to apply chain rule to compute gradient
- Auto-differentiation (AD) --- as long as we define derivative for each basic function, we can use AD to compute any of their compositions
- Implemented in most deep learning packages (e.g., pytorch, tensorflow)

Neural Network

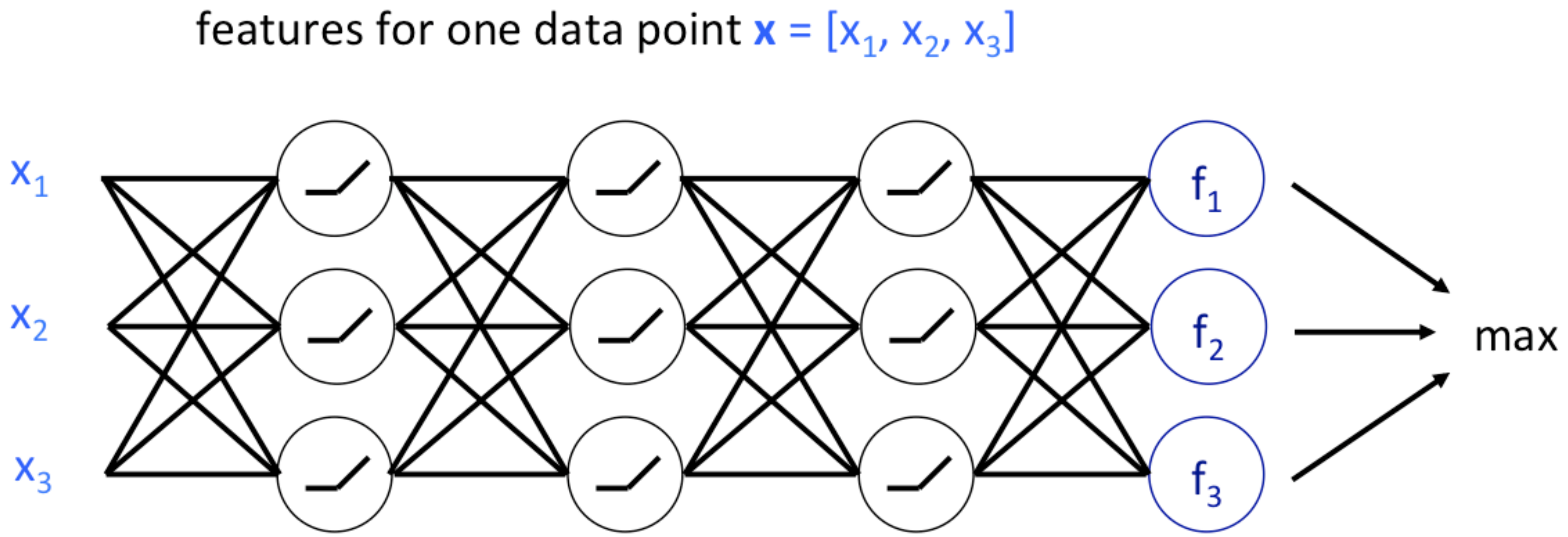
Backpropagation

- Just an automatic way to apply chain rule to compute gradient
- Auto-differentiation (AD) --- as long as we define derivative for each basic function, we can use AD to compute any of their compositions
- Implemented in most deep learning packages (e.g., pytorch, tensorflow)
- Auto-differentiation needs to store all the intermediate nodes of each sample
  - $\Rightarrow$ Memory cost > number of neurons $\times$ batch size
  - $\Rightarrow$ This poses a constraint on the batch size

Neural Network

Multiclass Classification

- K classes: K neurons in the final layer
- Output of each f_i is the score of class i
  - Taking $\arg\max_i f_i(x)$ as the prediction



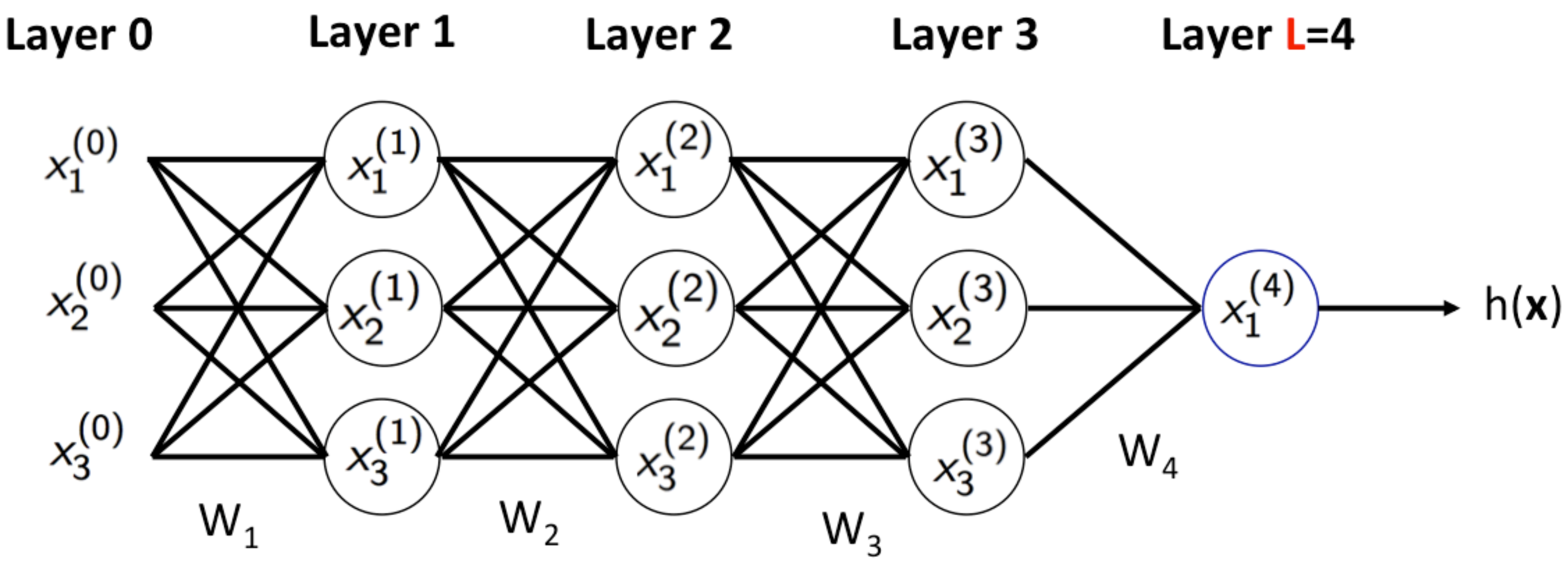
Neural Network

Multiclass loss

- Softmax function: transform output to probability:
  - $[f_1, \cdots, f_K] \rightarrow [p_i, \cdots, p_K]$
  - Where $p_i = \dfrac{e^{f_i}}{\sum_{j=1}^{K} e^{f_j}}$
- Cross-entropy loss:
  - $L = -\sum_{i=1}^{K} y_i \log(p_i)$
  - Where y_i is the i-th label

Convolutional Neural Network

Neural Networks



$$h(\mathbf{x}) = x_1^{(4)} = \theta(W_4\mathbf{x}^{(3)}) = \theta(W_4\theta(W_3\mathbf{x}^{(2)}))$$
$$= \cdots = \theta(W_4\theta(W_3\theta(W_2\theta(W_1\mathbf{x}))))$$

- Fully connected networks ⇒ doesn't work well for computer vision applications

Convolutional Neural Network

Convolution Layer

- Fully connected layers have too many parameters
  - $\Rightarrow$ poor performance
- Example: VGG first layer
  - Input: $224 \times 224 \times 3$
  - Output: $224 \times 224 \times 64$
  - Number of parameters if we use fully connected net:
    - $(224 \times 224 \times 3) \times (224 \times 224 \times 64) =$ 483 billion
  - Convolution layer leads to:
    - Local connectivity
    - Parameter sharing

Convolutional Neural Network

Convolution

- The convolution of an image x with a kernel k is computed as
  - $(x * k)_{ij} = \sum_{pq} x_{i+p,j+q} k_{p,q}$

1	0.5	20
0.25	0	0
0	0	20

$*$

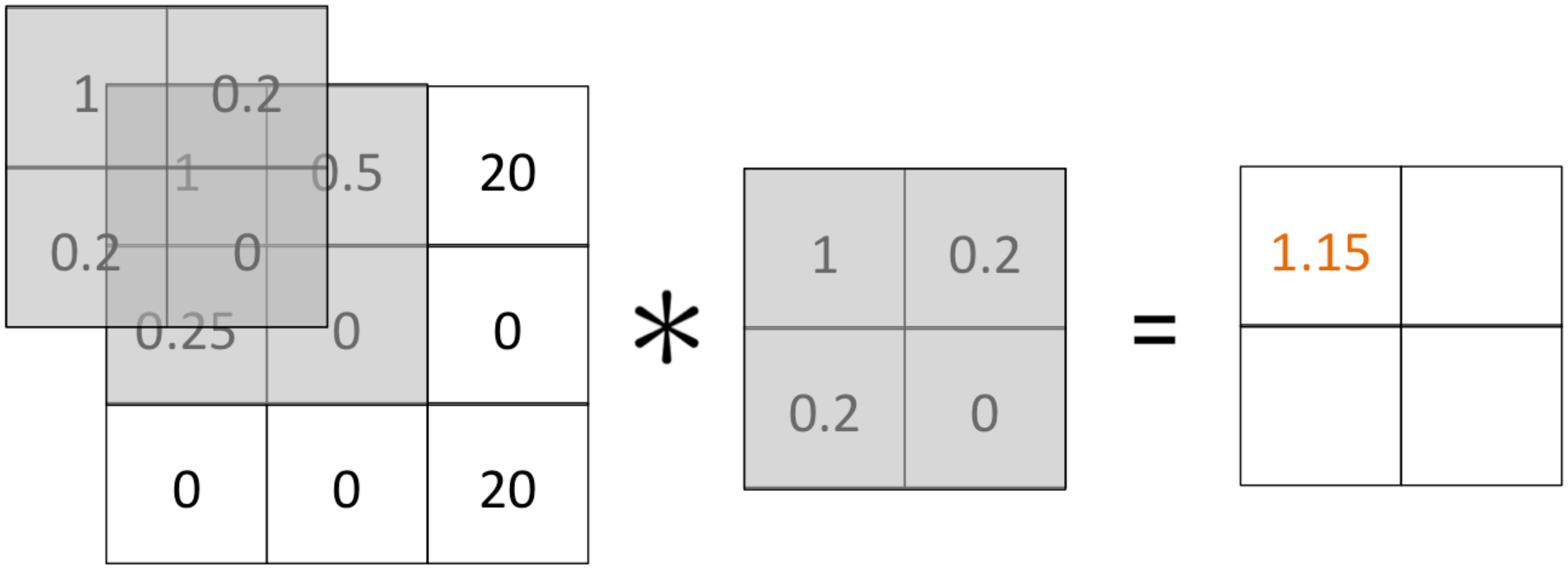
1	0.5
0.25	0

$=$

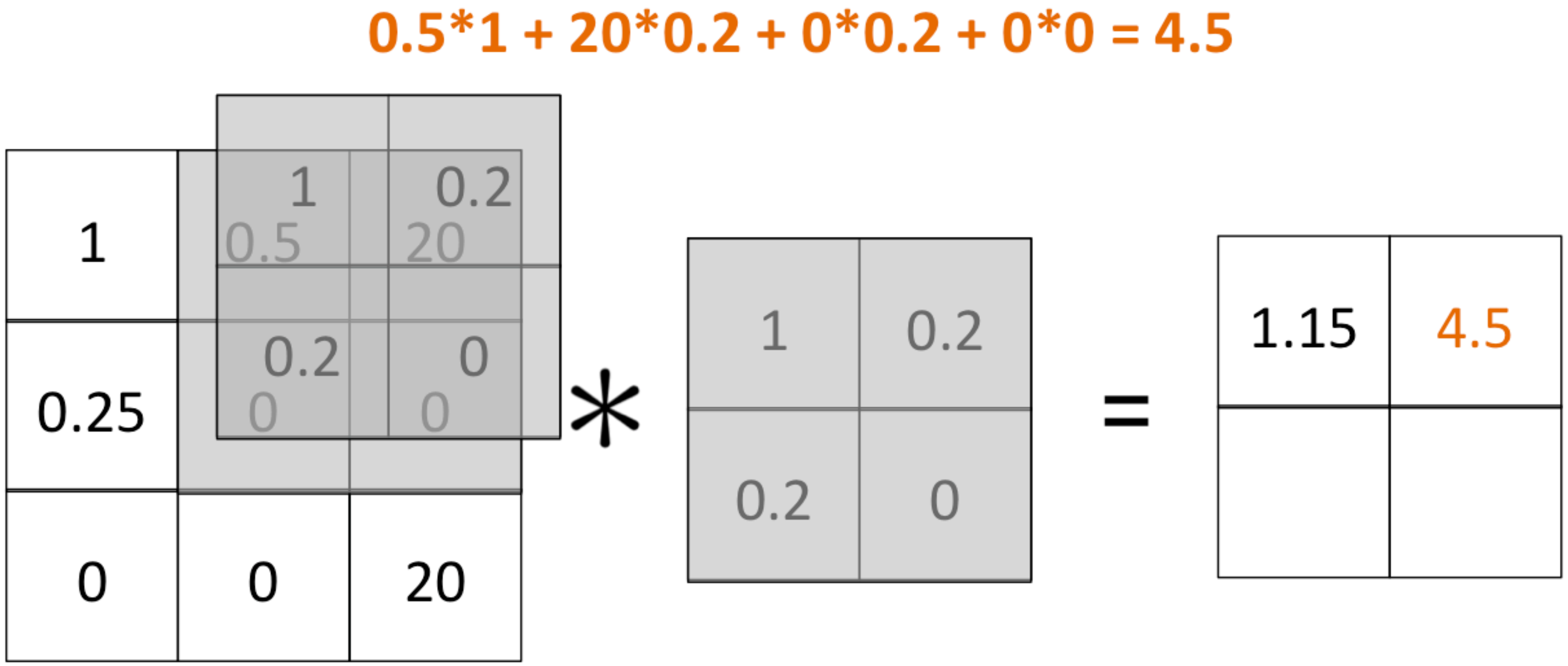
Convolutional Neural Network

Convolution

1*1 + 0.5*0.2 + 0.25*0.2 + 0*0 = 1.15



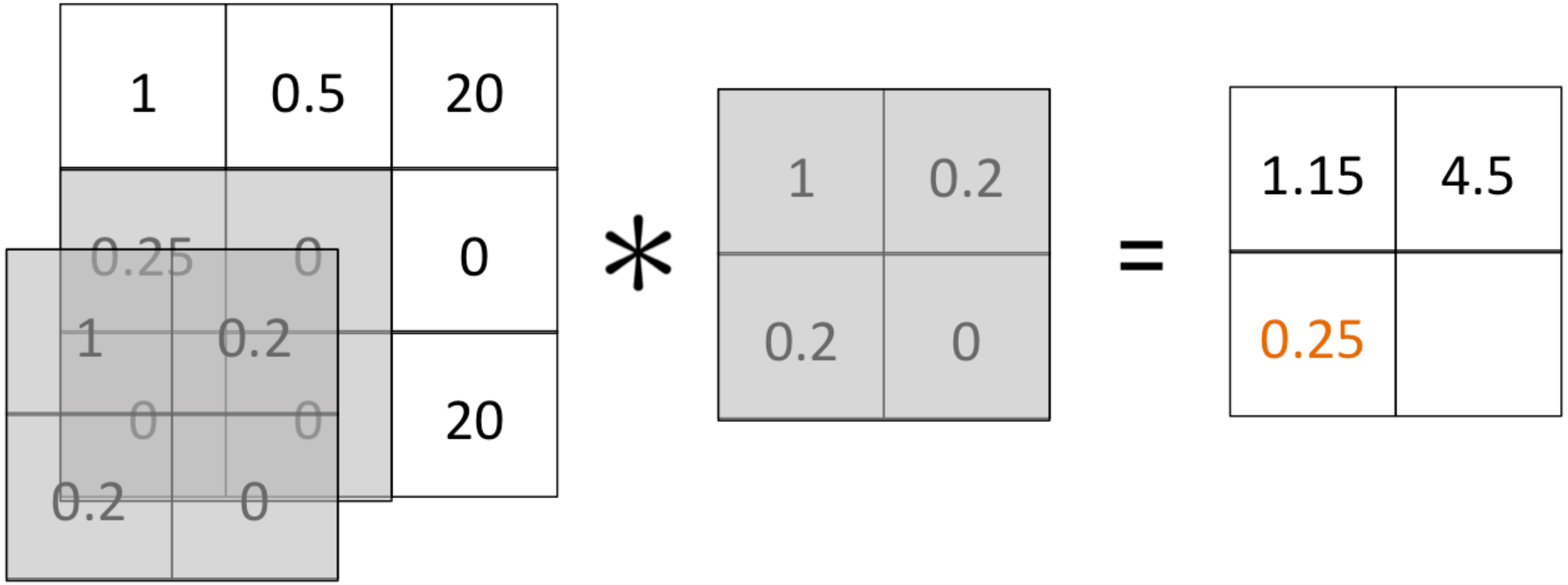
Convolutional Neural Network
Convolution



Convolutional Neural Network

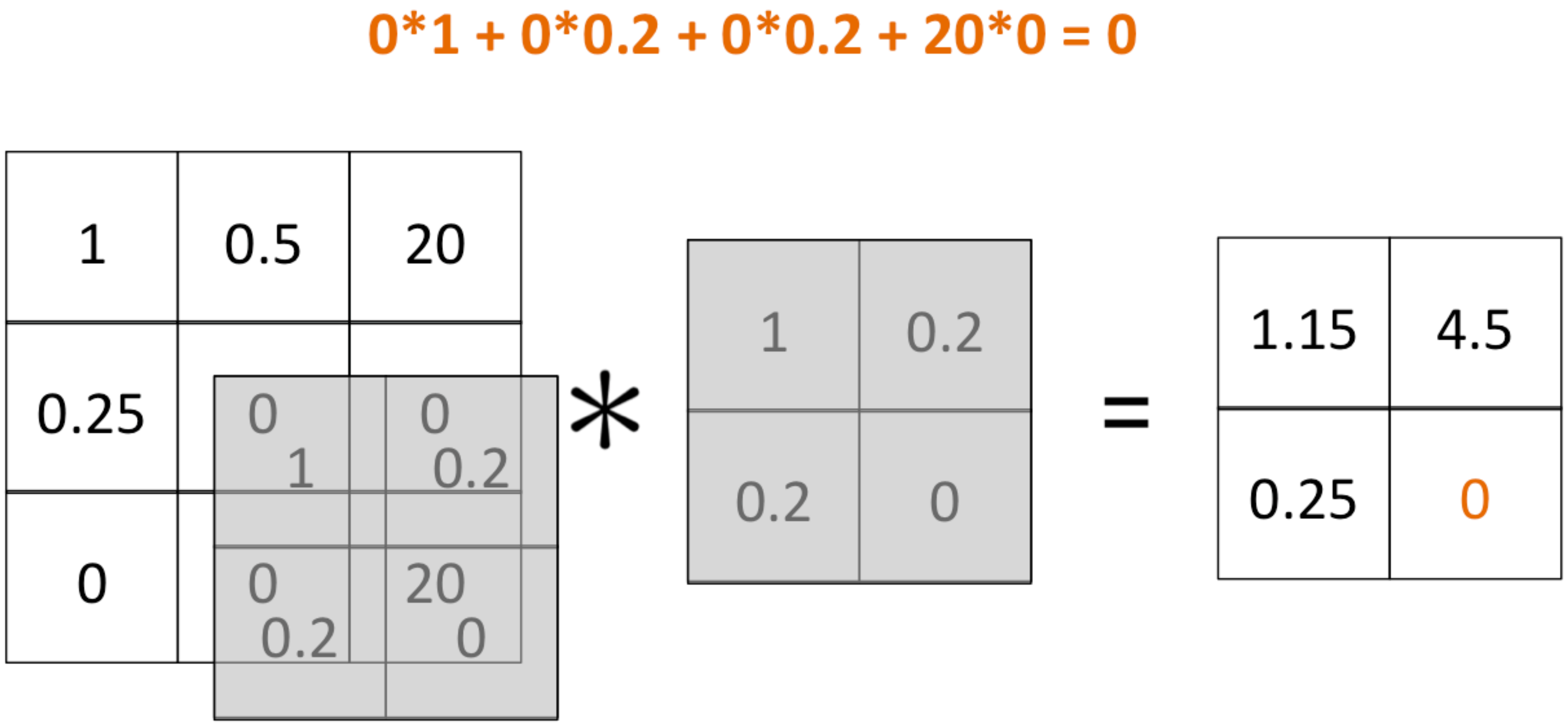
Convolution

0.25*1 + 0*0.2 + 0*0.2 + 0*0 = 0.25



Convolutional Neural Network

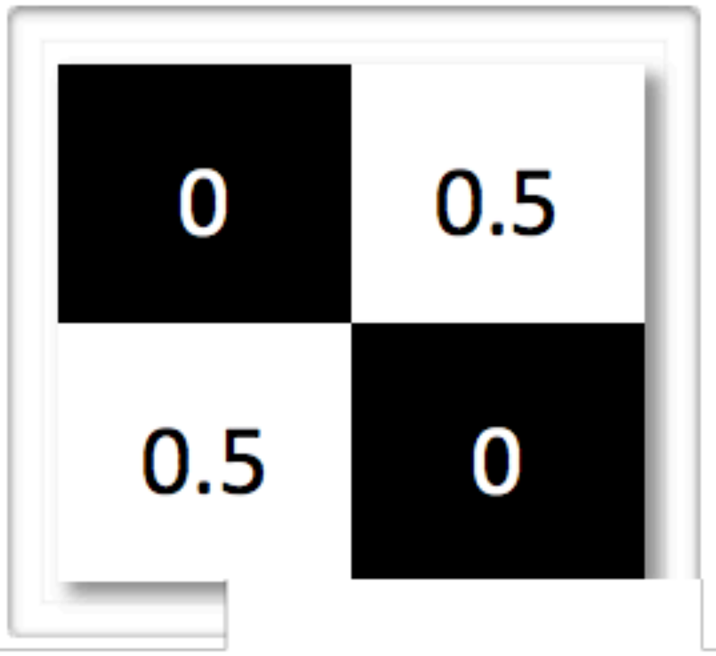
Convolution

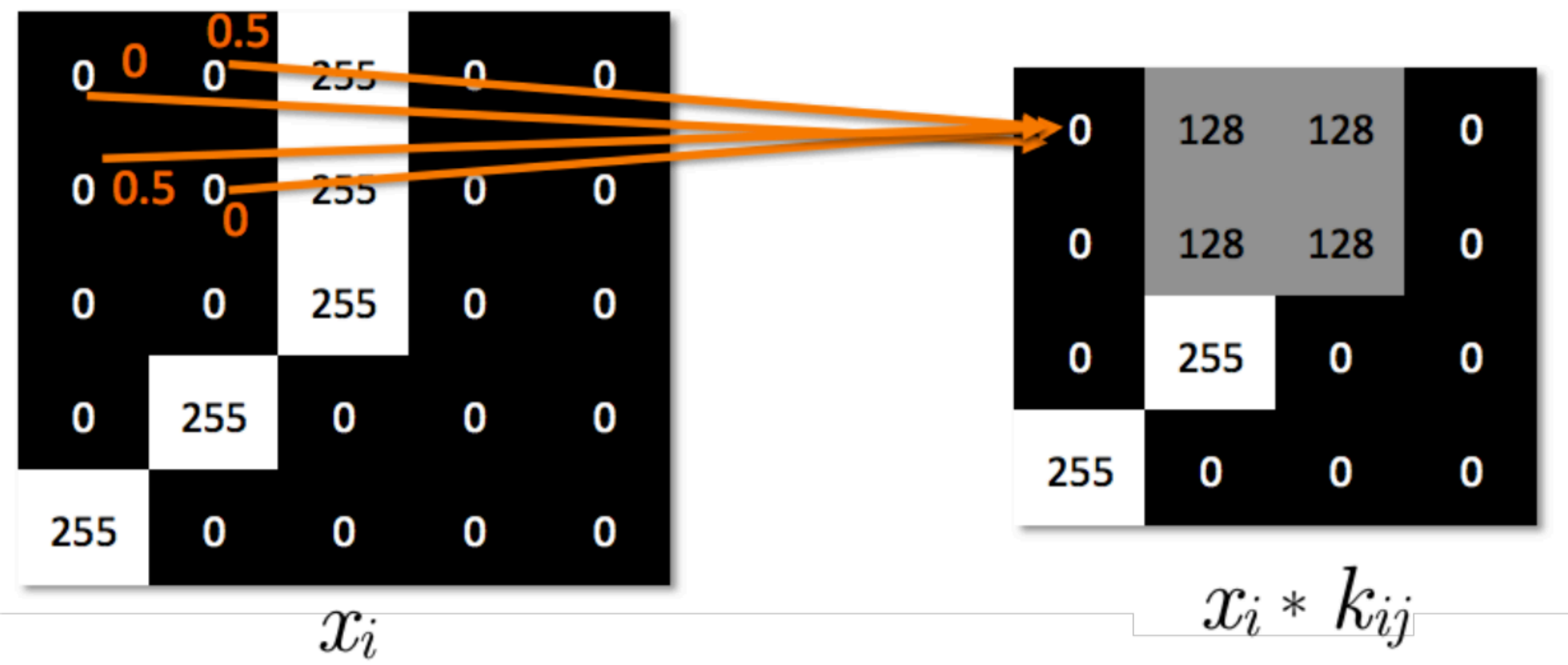


Convolutional Neural Network

Convolution

$$x * k_{ij}, \quad \text{where } W_{ij} = \tilde{W}_{ij}$$





Convolutional Neural Network
Convolution

- Element-wise activation function after convolution
  - ⇒ detector of a feature at any position in the image

$$x * k_{ij}, \quad \text{where } W_{ij} = \tilde{W}_{ij}$$

0	0.5
0.5	0

0	0	255	0	0
0	0	255	0	0
0	0	255	0	0
0	255	0	0	0
255	0	0	0	0

x_i

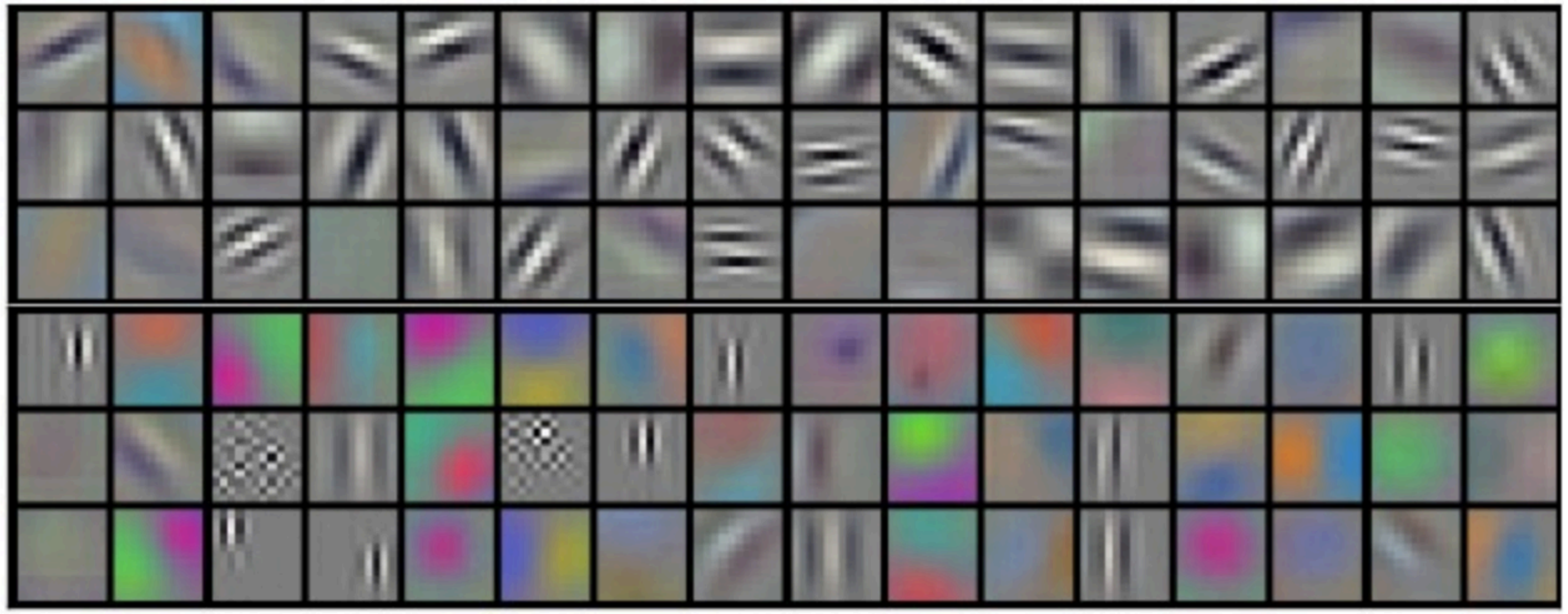
0.02	0.19	0.19	0.02
0.02	0.19	0.19	0.02
0.02	0.75	0.02	0.02
0.75	0.02	0.02	0.02

$\text{sigm}(0.02 \; x_i * k_{ij} - 4)$

Convolutional Neural Network

Learned Kernels

- Example kernels learned by AlexNet

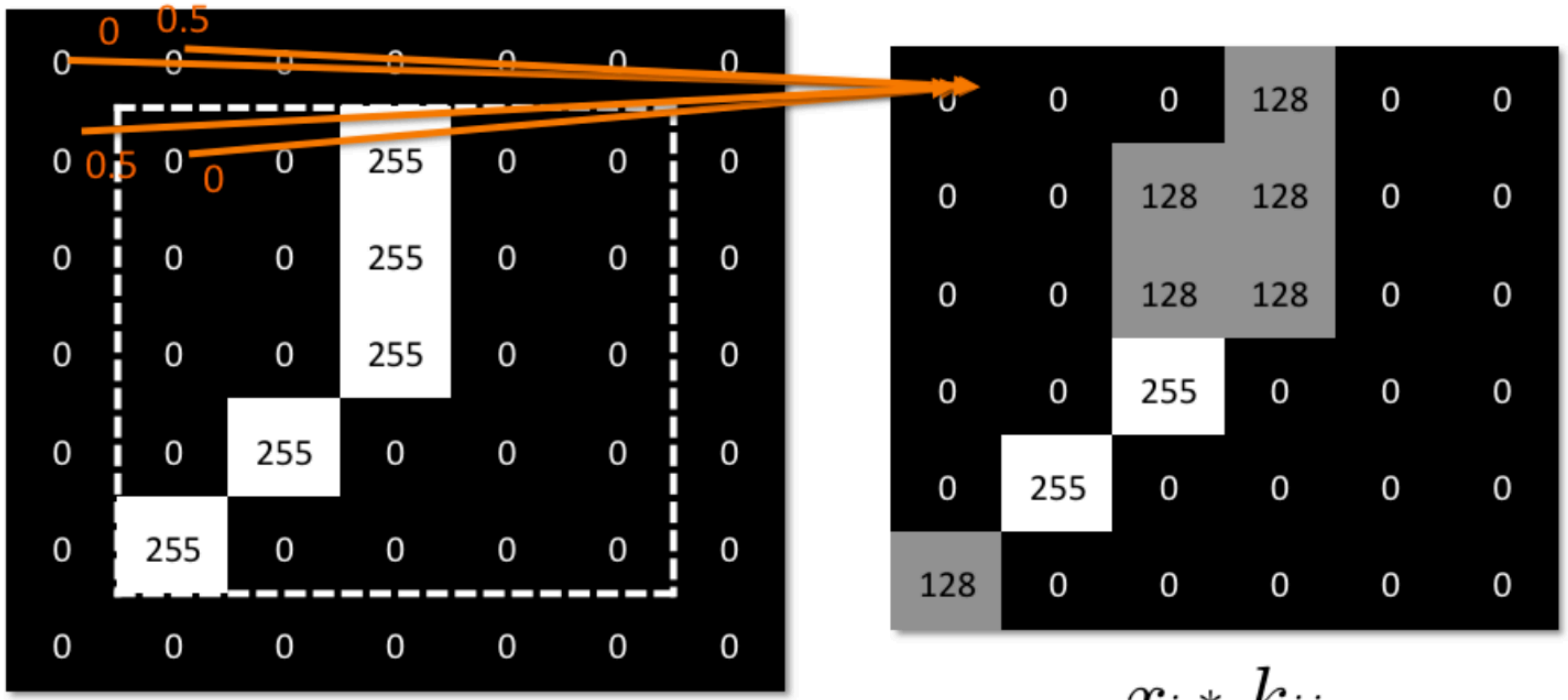


- Number of parameters:
  - Example: 200×200 image, 100 kernels, kernel size 10×10
  - $\Rightarrow 10 \times 10 \times 100$ = 10K parameters

Convolutional Neural Network

Padding

- Use zero padding to allow going over the boundary
    - Easier to control the size of output layer



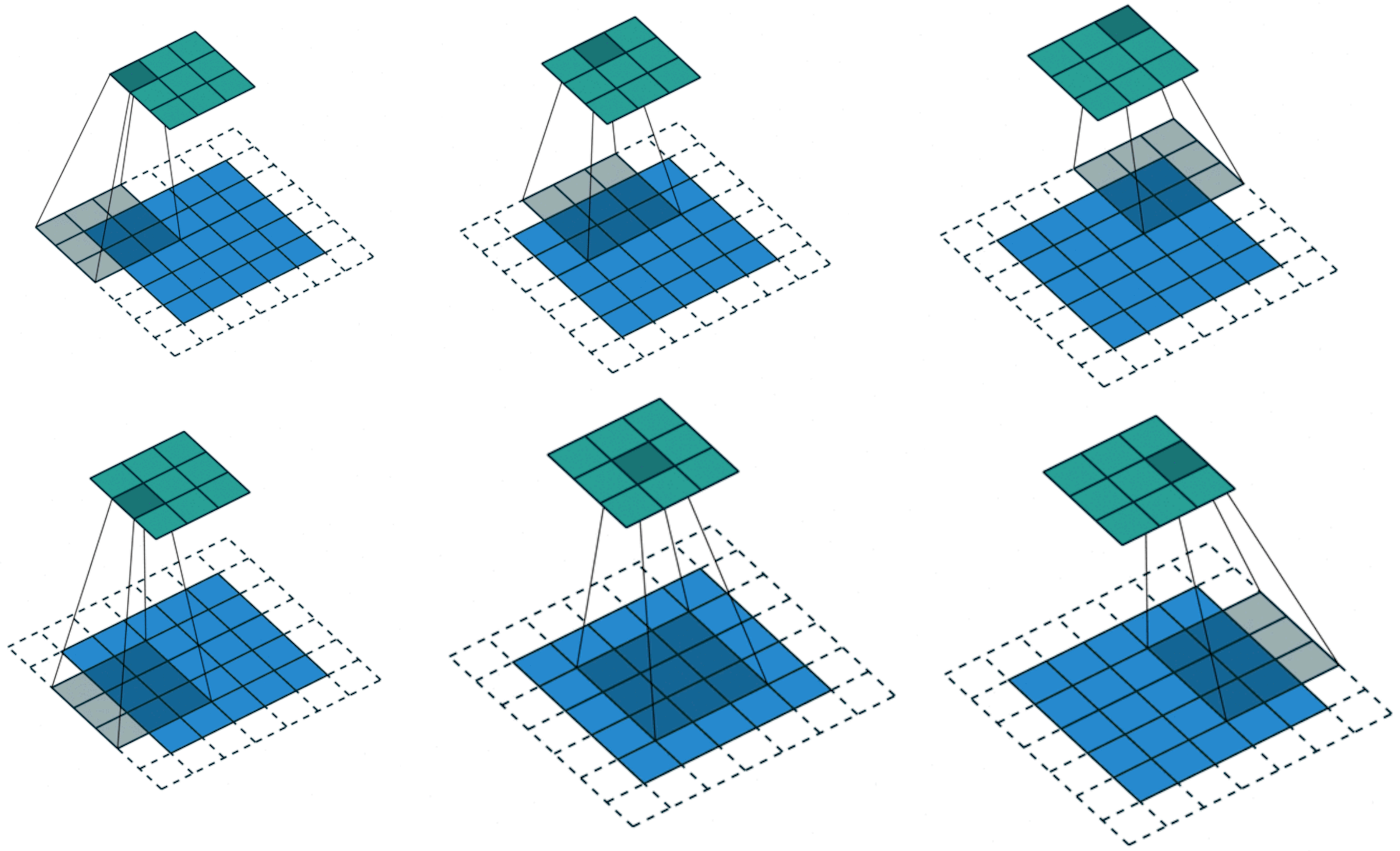
x_i $\quad$ $x_i * k_{ij}$

Convolutional Neural Network

Strides

- Stride: The amount of movement between applications of the filter to the input image

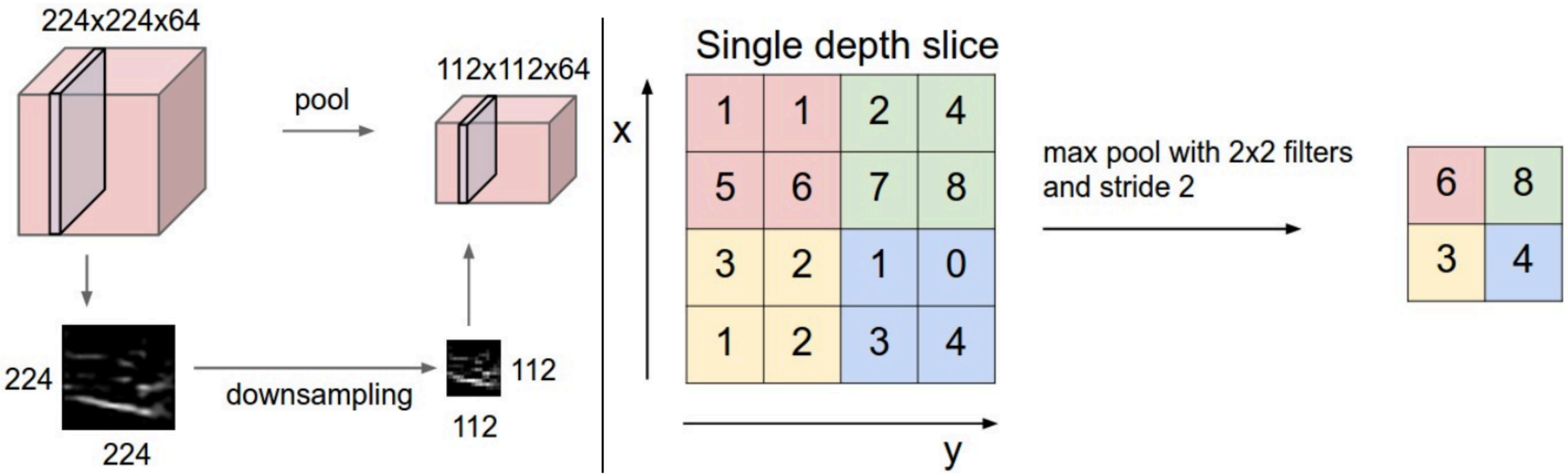
- Stride $(1,1)$: no stride



Convolutional Neural Network

Pooling

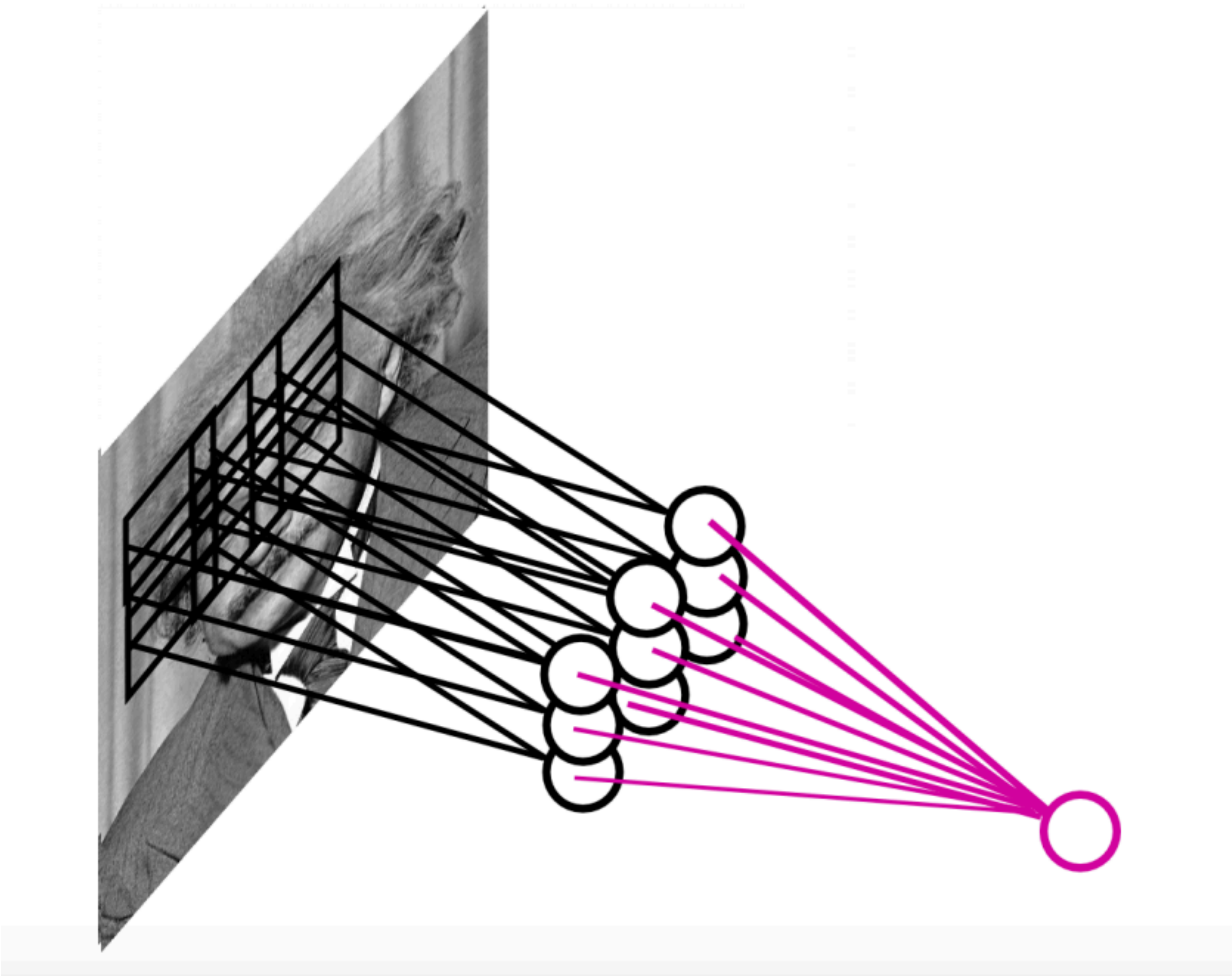
- It’s common to insert a pooling layer in-between successive convolutional layers
- Reduce the size of presentation, down-sampling
- Example: Max pooling



Convolutional Neural Network

Pooling

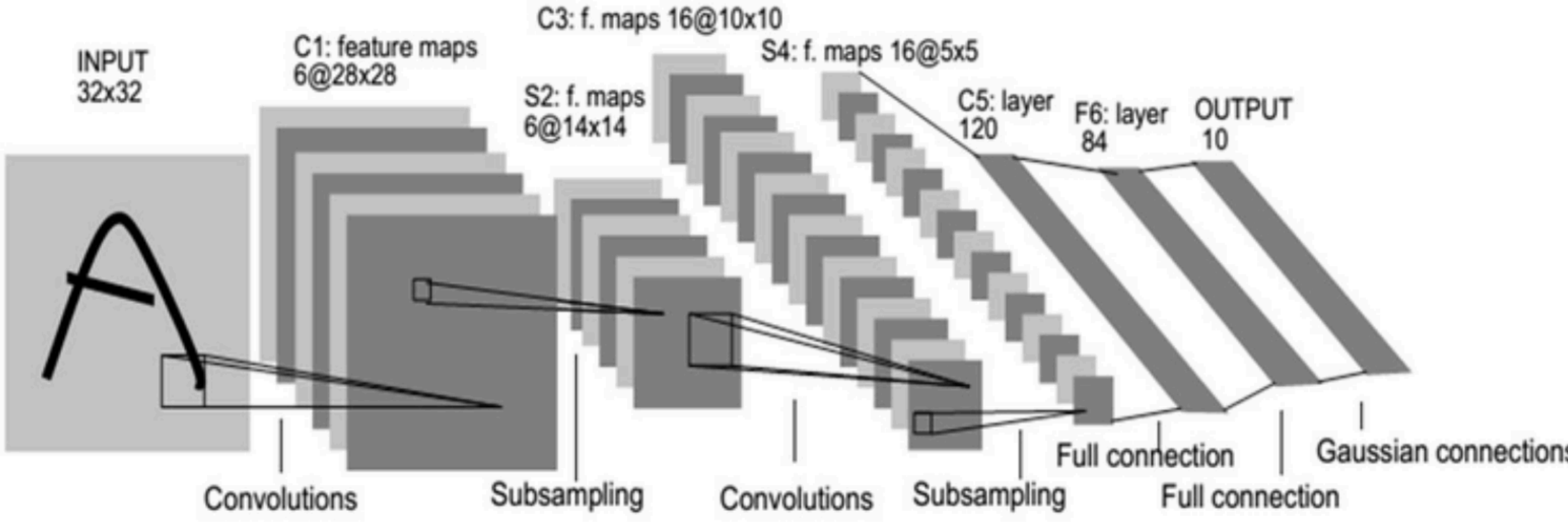
- By pooling, we gain robustness to the exact spatial location of features



Convolutional Neural Network

Example: LeNet5

- Input: 32×32 images (MNIST)
- Convolution 1: 6 5×5filters, stride 1
  - Output: 6 28×28 maps
- Pooling 1: 2×2 max pooling, stride 2
  - Output: 6 14×14 maps
- Convolution 2: 16 5×5 filters, stride 1
  - Output: 16 10×10 maps
- Pooling 2: 2×2 max pooling with stride 2
  - Output: 16 5×5 maps (total 400 values)
- 3 fully connected layers: $120 \Rightarrow 84 \Rightarrow 10$ neurons



Convolutional Neural Network

Training

- Training:
  - Apply SGD to minimize in-sample training error
  - Backpropagation can be extended to convolutional layer and pooling layer to compute gradient!
  - Millions of parameters $\Rightarrow$ easy to overfit

Convolutional Neural Network

Revisit Alexnet

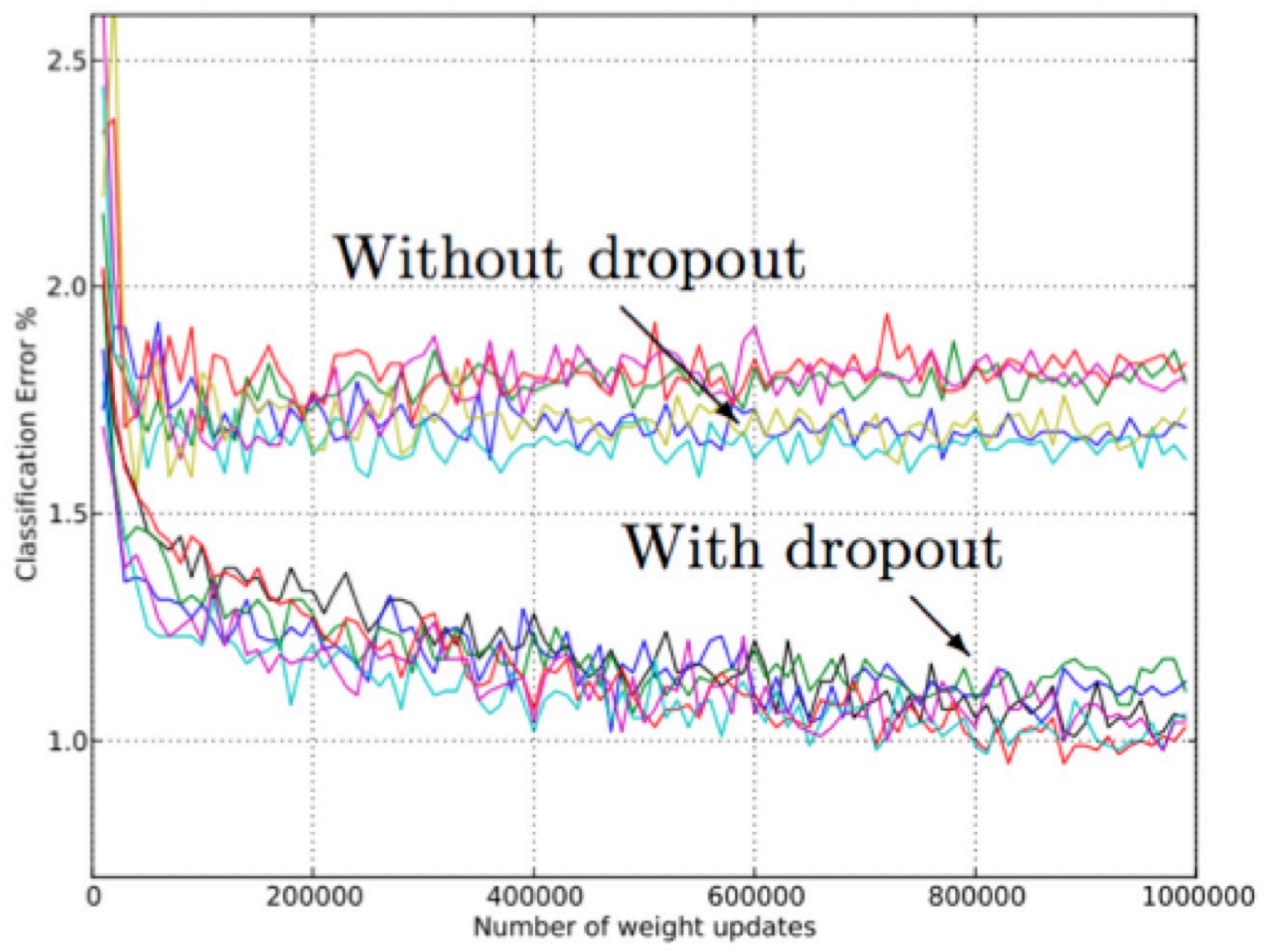
- Dropout: 0.5 (in FC layers)
- A lot of data augmentation
- Momentum SGD with batch size 128, momentum factor 0.9
- L2 weight decay (L2 regularization)
- Learning rate: 0.01, decreased by 10 every time when reaching a stable validation accuracy

Convolutional Neural Network
Dropout

- One of the most effective regularization for deep neural networks

Method	CIFAR-10	CIFAR-100
Conv Net + max pooling (hand tuned)	15.60	43.48
Conv Net + stochastic pooling (Zeiler and Fergus, 2013)	15.13	42.51
Conv Net + max pooling (Snoek et al., 2012)	14.98	-
Conv Net + max pooling + dropout fully connected layers	14.32	41.26
Conv Net + max pooling + dropout in all layers	12.61	**37.20**
Conv Net + maxout (Goodfellow et al., 2013)	**11.68**	38.57

Table 4: Error rates on CIFAR-10 and CIFAR-100.

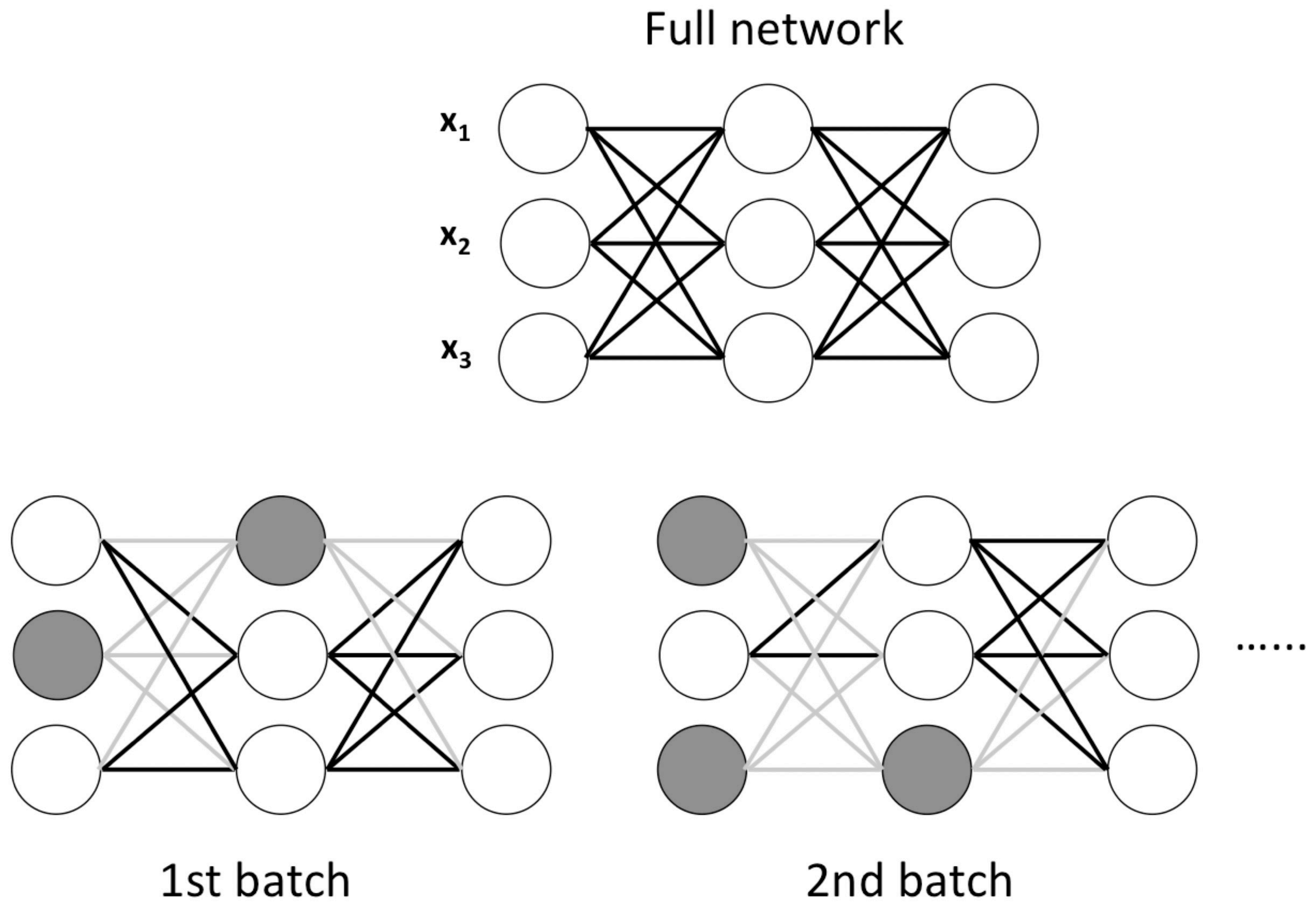


Srivastava et al, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, 2014.

Convolutional Neural Network

Dropout(training)

- Dropout in the **training** phase:
  - For each batch, turn off each neuron (including inputs) with a probability $1 - \alpha$
  - Zero out the removed nodes/edges and do backpropogation



Convolutional Neural Network

Dropout(test)

- The model is different from the full model:
- Each neuron computes
  - $x_i^{(l)} = B\sigma(\sum_j W_{ij}^{(l)} x_j^{(l-1)} + b_i^{(l)})$
  - Where B is Bernoulli variable that takes 1 with probability α
- The expected output of the neuron:
  - $E[x_i^{(l)}] = \alpha\sigma(\sum_j W_{ij}^{(l)} x_j^{(l-1)} + b_i^{(l)})$
- Use the expected output at test time $\Rightarrow$ multiply all the weights by α

Convolutional Neural Network

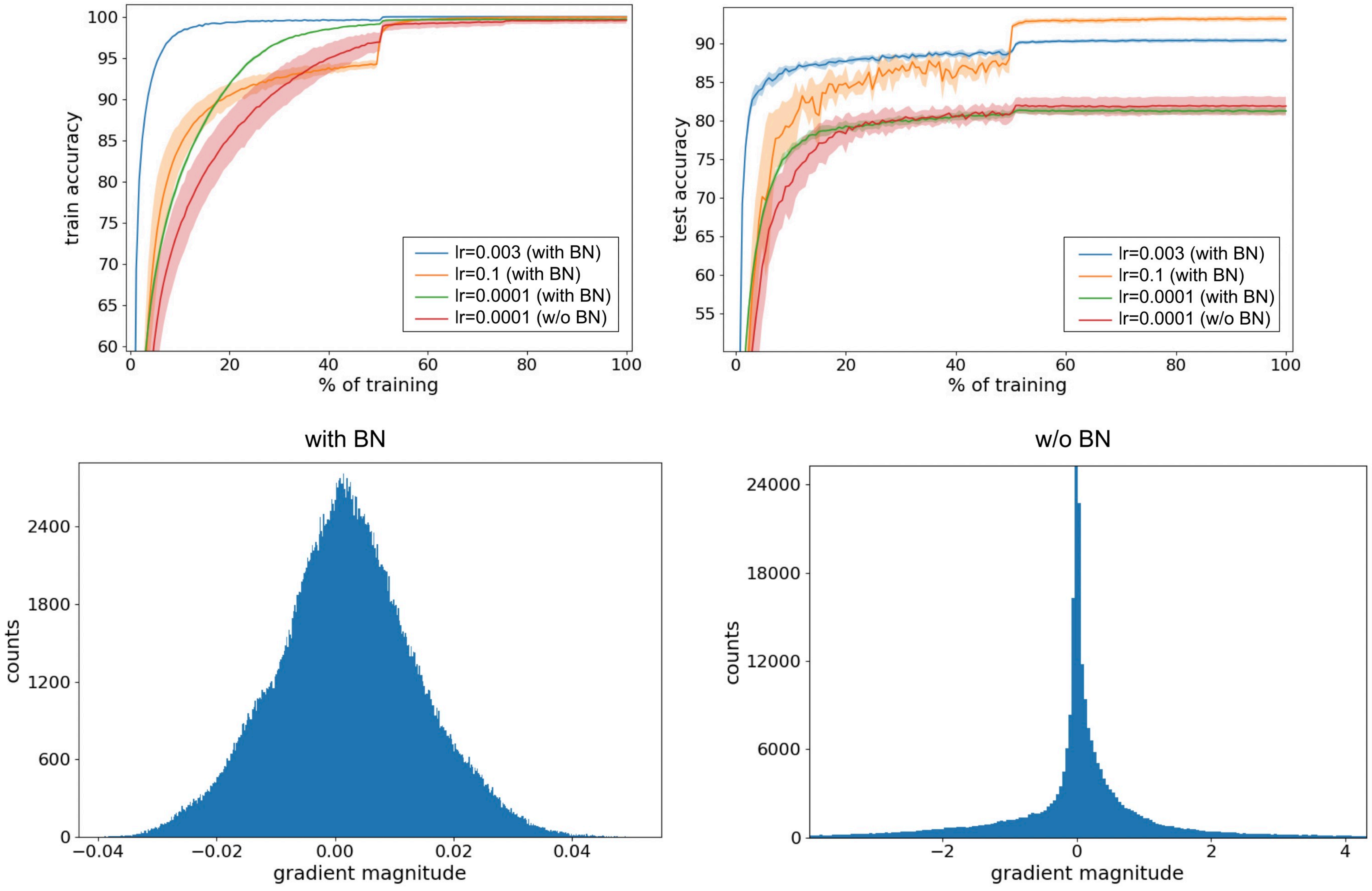
Batch Normalization

- Initially proposed to reduce co-variate shift
- $$O_{b,c,x,y} \leftarrow \gamma \frac{I_{b,c,x,y} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}} + \beta \quad \forall b, c, x, y,$$
- $\mu_c = \frac{1}{|B|} \sum_{b,x,y} I_{b,c,x,y}$: the mean for channel c, and σ_c standard deviation.
- γ and β: two learnable parameters

Convolutional Neural Network

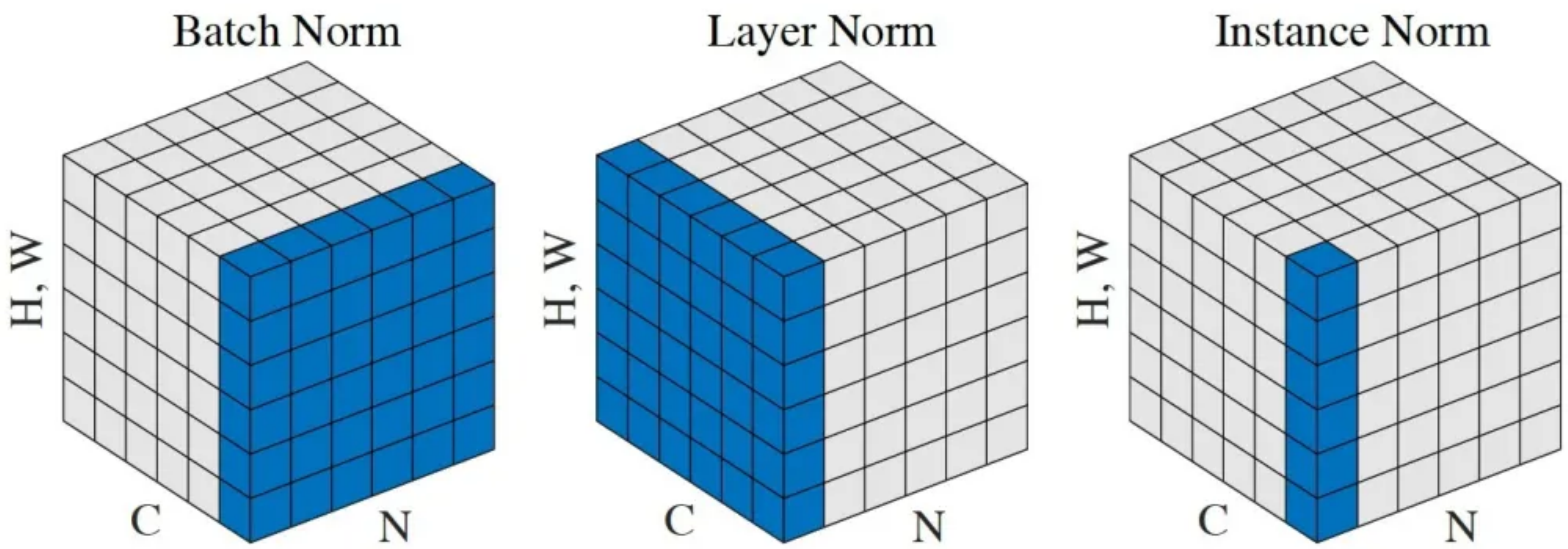
Batch Normalization

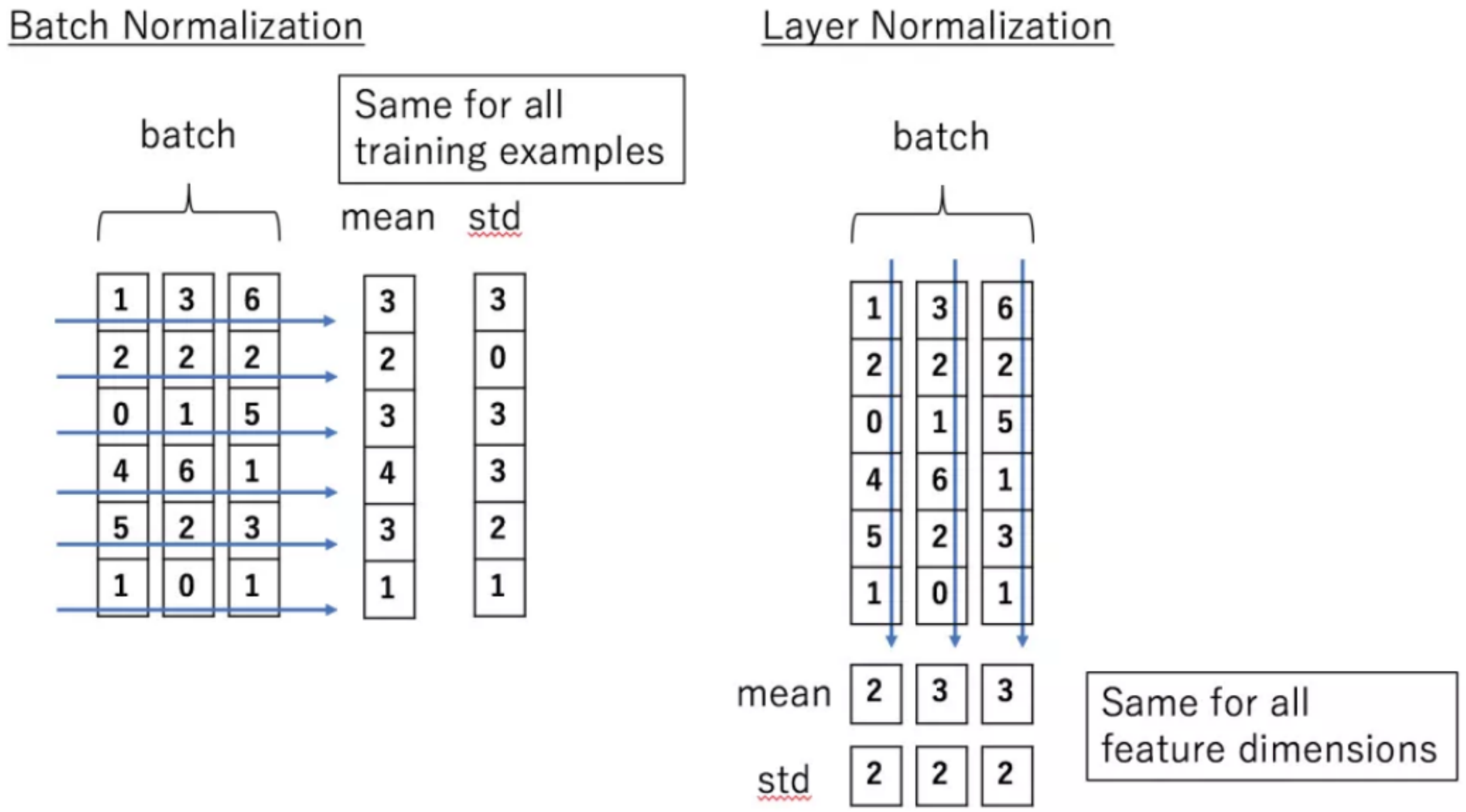
- Couldn't reduce covariate shift (Ilyas et al 2018)
- Allow larger learning rate
  - Constraint the gradient norm



Convolutional Neural Network

Other normalization

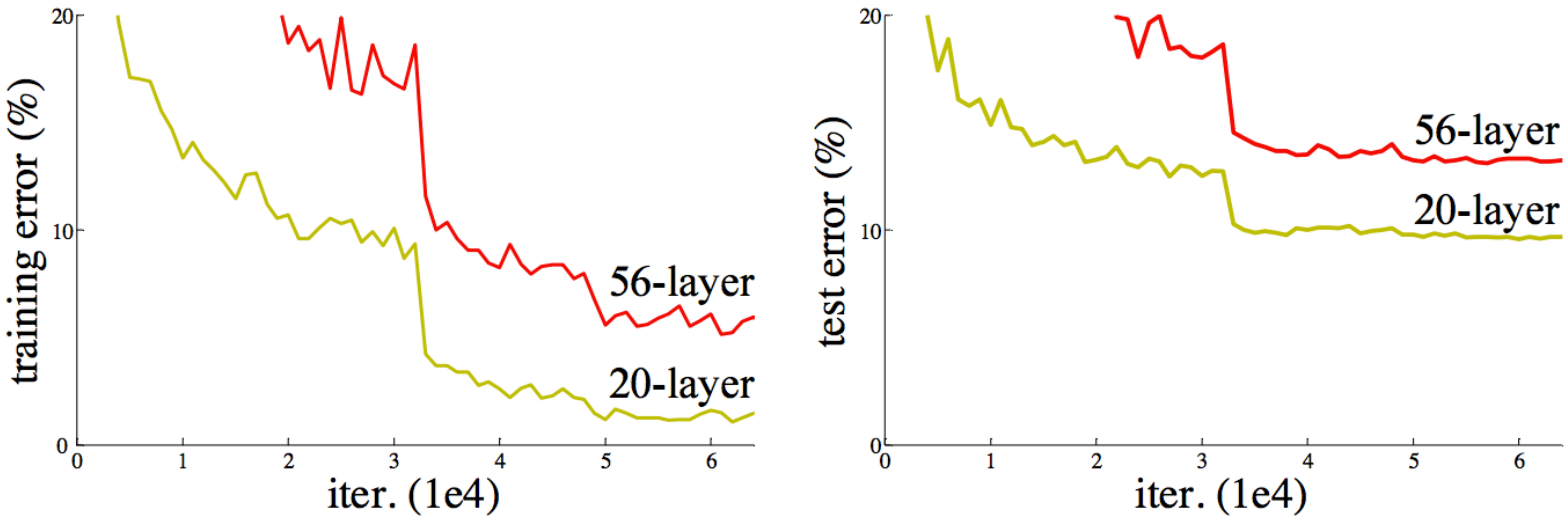




Convolutional Neural Network

Residual Networks

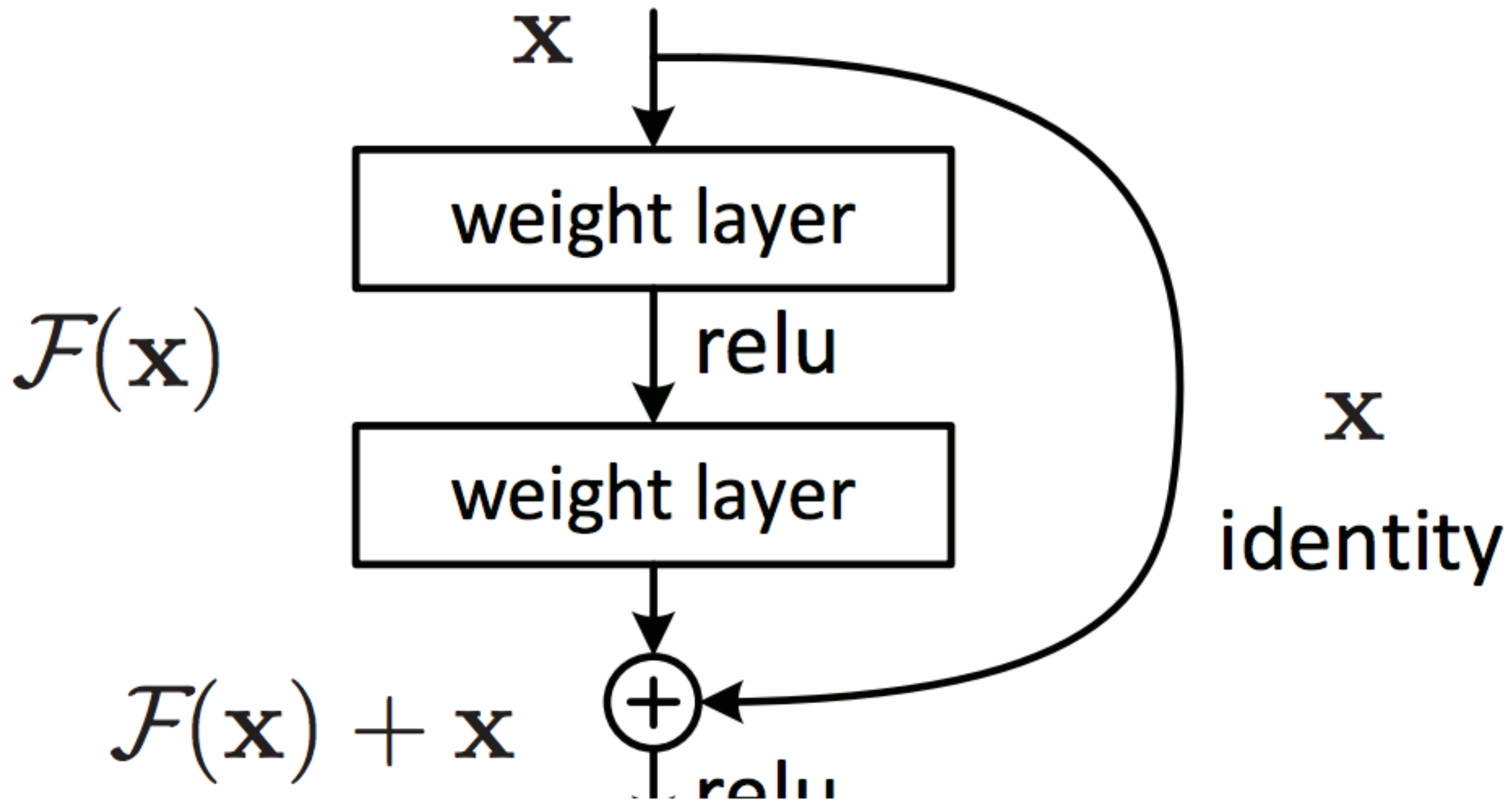
- Very deep convnets do not train well — vanishing gradient problem



Convolutional Neural Network

Residual Networks

- Key idea: introduce ``pass through'' into each layer



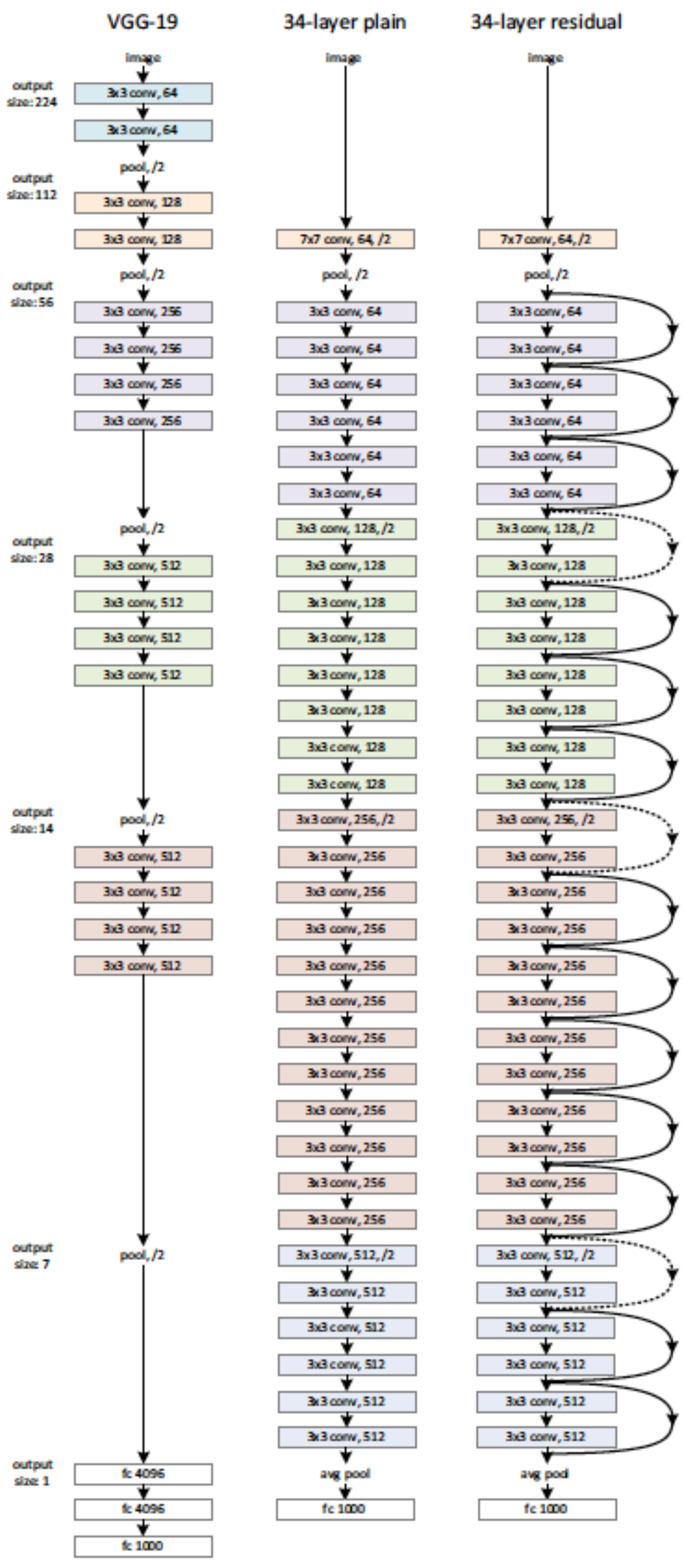
- Thus, only residual needs to be learned

Convolutional Neural Network

Residual Networks

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

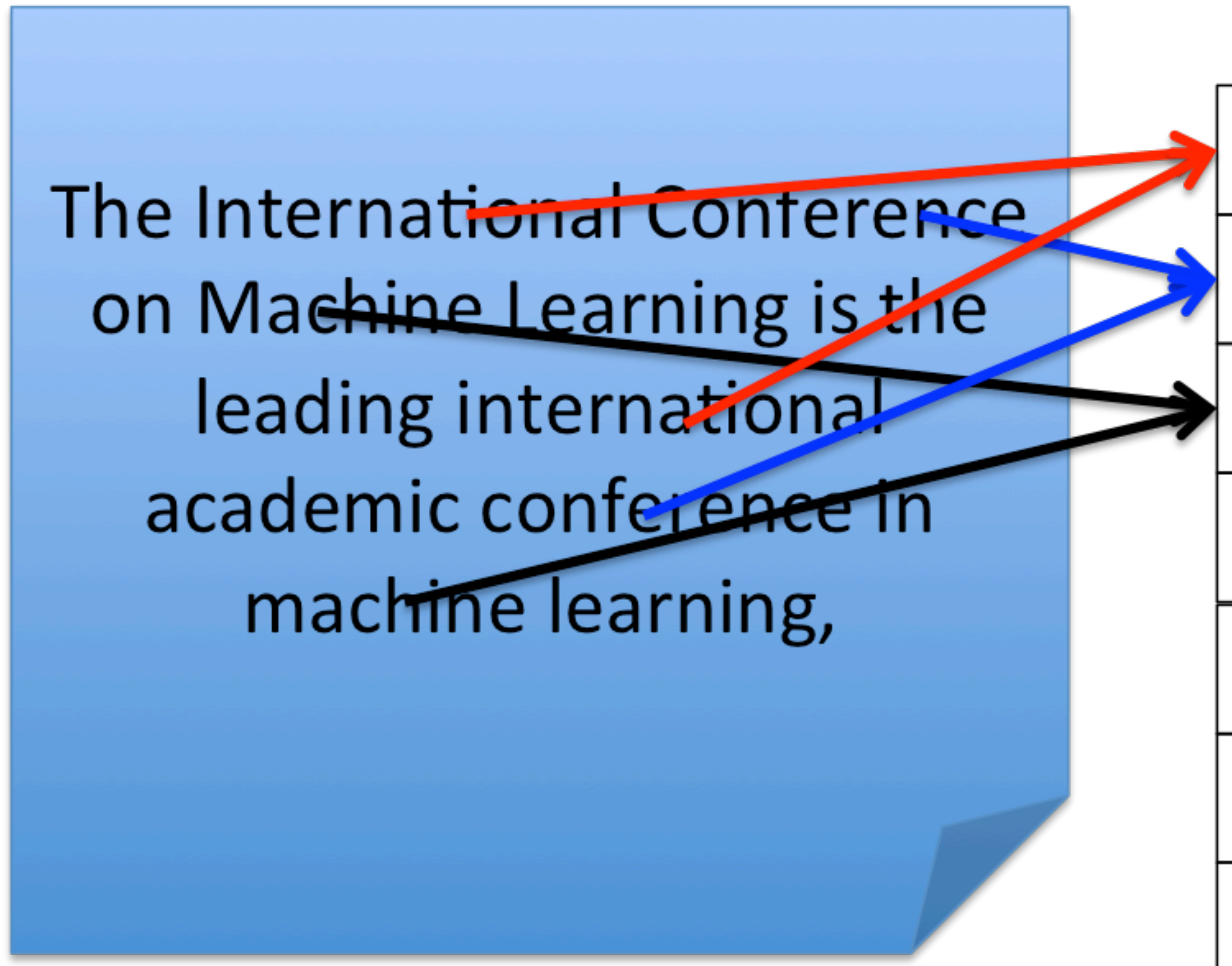
Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).



Representation for sentence/document

Bag of word

- A classical way to represent NLP data
- Each sentence (or document) is represented by a d-dimensional vector $\mathbf{x}$, where x_i is number of occurrences of word i
- number of features = number of potential words (very large)



(international)	2
(conference)	2
(machine)	2
(train)	0
(learning)	2
(leading)	1
(totoro)	0

Representation for sentence/document

Feature generation for documents

- Bag of n-gram features ($n = 2$):
  - 10,000 words $\Rightarrow 10000^2$ potential features

The International Conference on Machine Learning is the leading international academic conference in machine learning,

(international)	2
(conference)	2
(machine)	2
(train)	0
(learning)	2
(leading)	1
(totoro)	0

(international conference)	1
(machine learning)	2
(leading international)	1
(totoro tiger)	0
(tiger woods)	0
(international academic)	1
(international academic)	1

Representation for sentence/document

Bag of word + linear model

- Example: text classification (e.g., sentiment prediction, review score prediction)
- Linear model: $y \approx \text{sign}(w^T x)$ (e.g., by linear SVM/logistic regression)
- w_i: the ``contribution'' of each word

Representation for sentence/document

Bag of word + Fully connected network

- $f(x) = W_L\sigma(W_{L-1}\cdots\sigma(W_0 x))$
- The first layer W_0 is a d_1 by d matrix:
    - Each column w_i is a d_1 dimensional representation of i-th word (word embedding)
    - $W_0 x = x_1 w_1 + x_2 w_2 + \cdots + x_d w_d$ is a linear combination of these vectors
    - W_0 is also called the word embedding matrix
    - Final prediction can be viewed as an $L-1$ layer network on $W_0 x$ (average of word embeddings)
- Not capturing the sequential information

Recurrent Neural Network

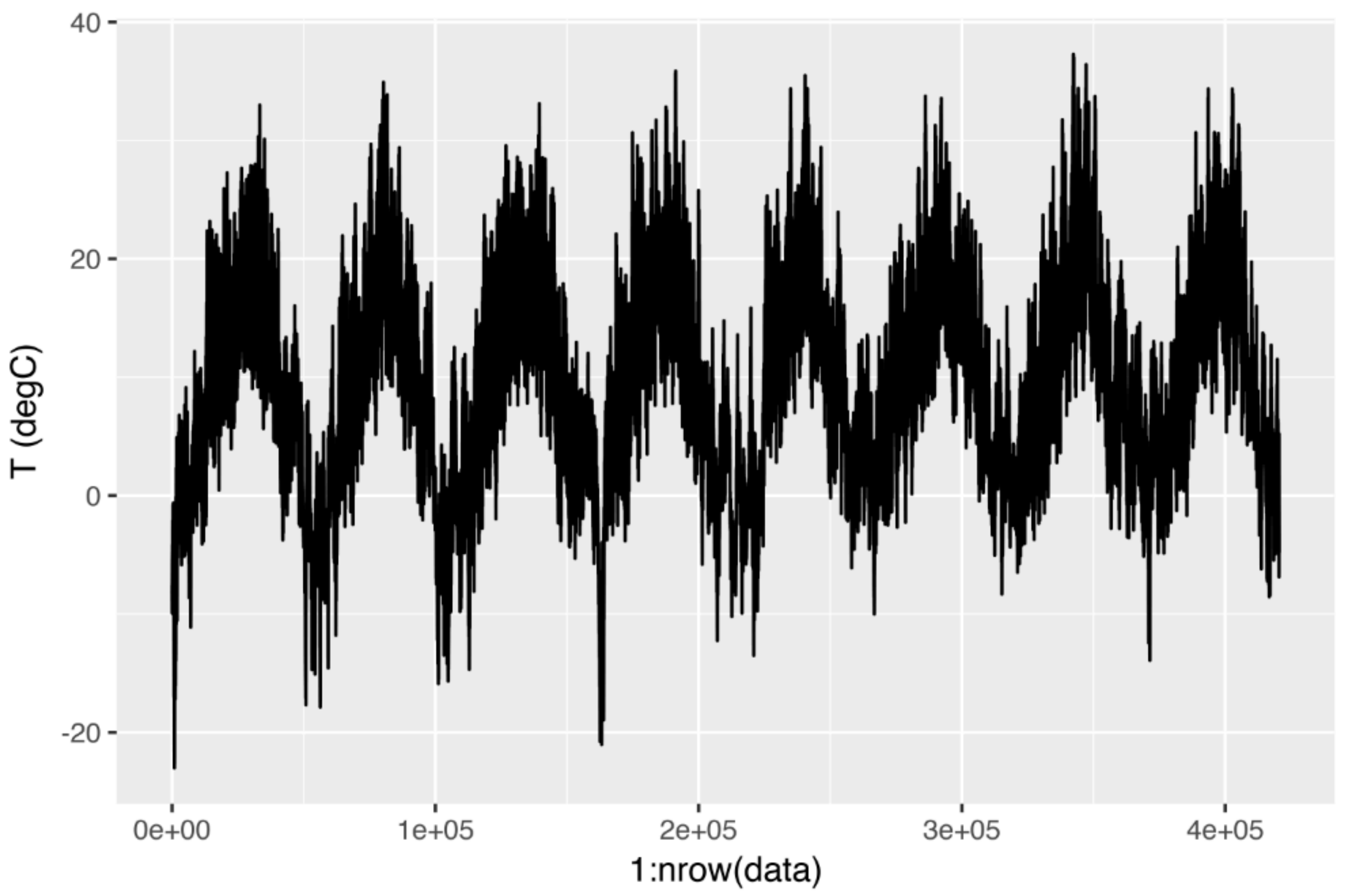
Time series/Sequence data

- Input: $\{x_1, x_2, \cdots, x_T\}$
  - Each x_t is the feature at time step t
  - Each x_t can be a d-dimensional vector
- Output: $\{y_1, y_2, \cdots, y_T\}$
  - Each y_t is the output at step t
  - Multi-class output or Regression output:
    - $y_t \in \{1,2,\cdots,L\}$ or $y_t \in \mathbb{R}$

Recurrent Neural Network

Example: Time Series Prediction

- Climate Data:
  - x_t: temperature at time t
  - y_t: temperature (or temperature change) at time $t + 1$
- Stock Price: Predicting stock price



Recurrent Neural Network

Example: Language Modeling

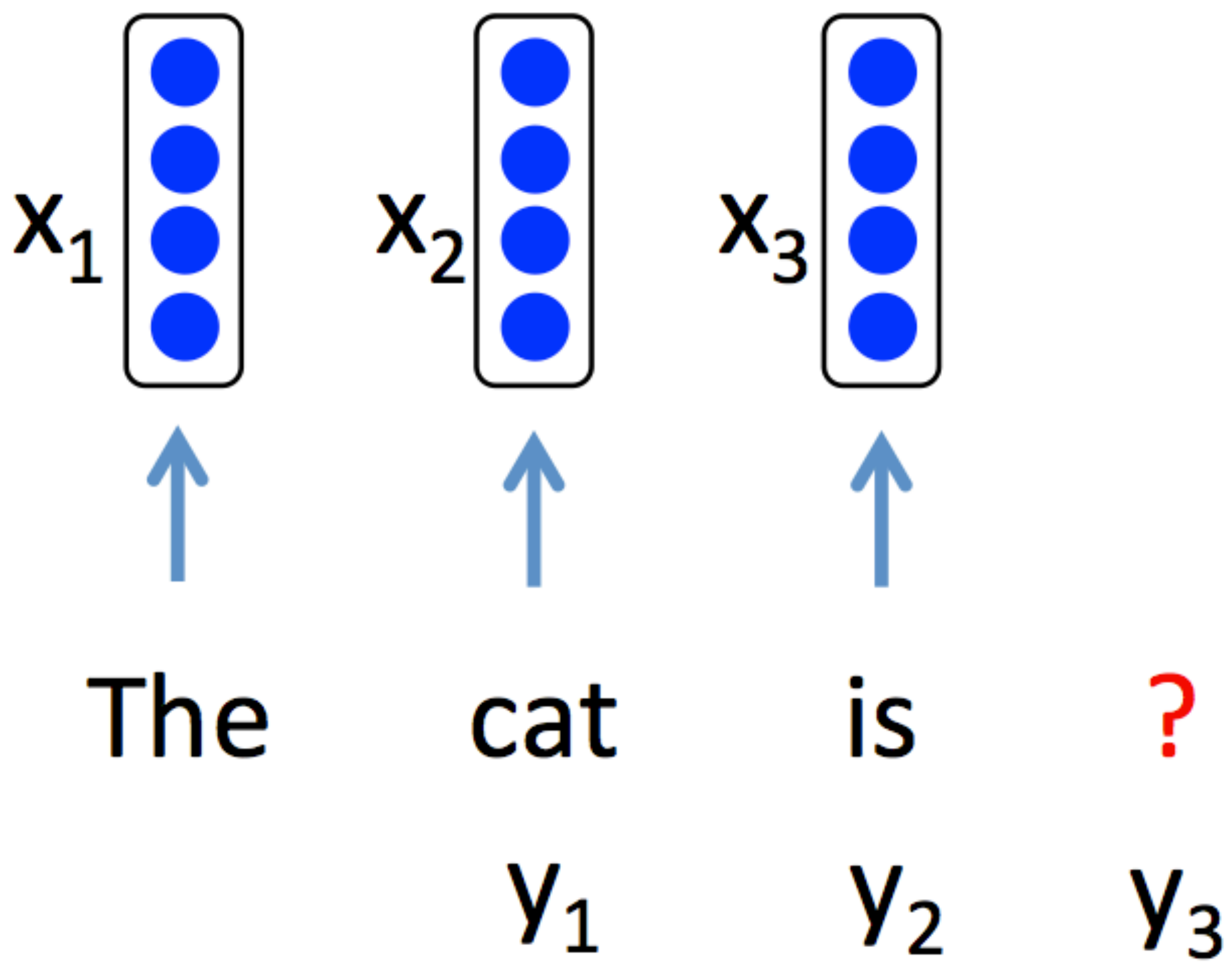
The cat is ?

Recurrent Neural Network

Example: Language Modeling

The cat is ?

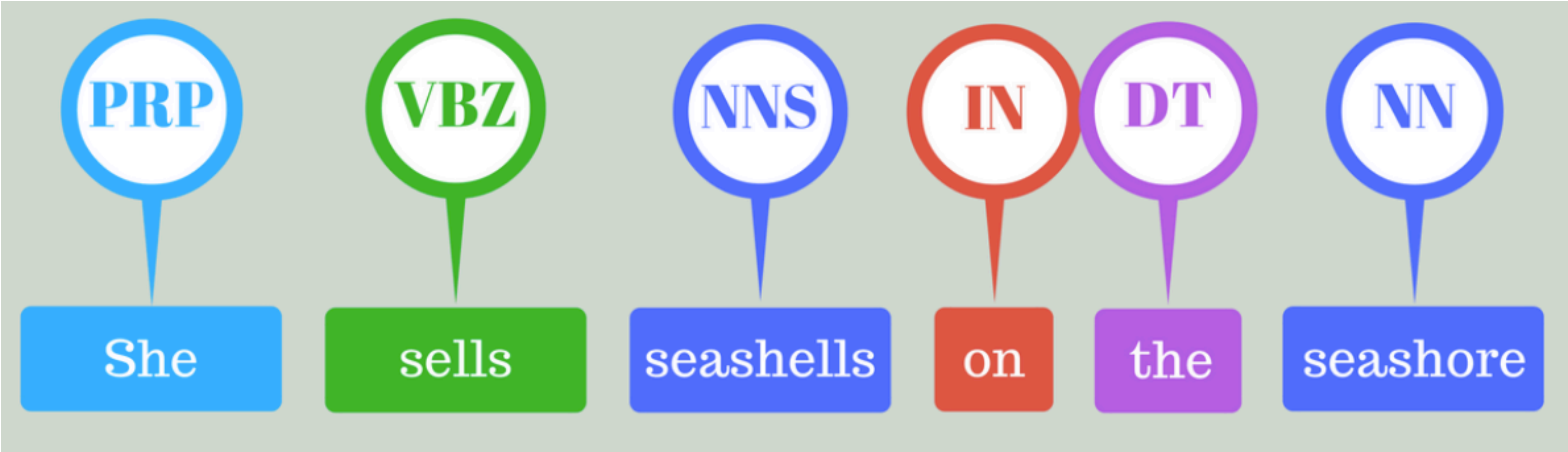
- x_t: one-hot encoding to represent the word at step t ($[0,\ldots,0,1,0,\ldots,0]$)
- y_t: the next word
    - $y_t \in \{1, \cdots, V\}$ V: Vocabulary size



Recurrent Neural Network

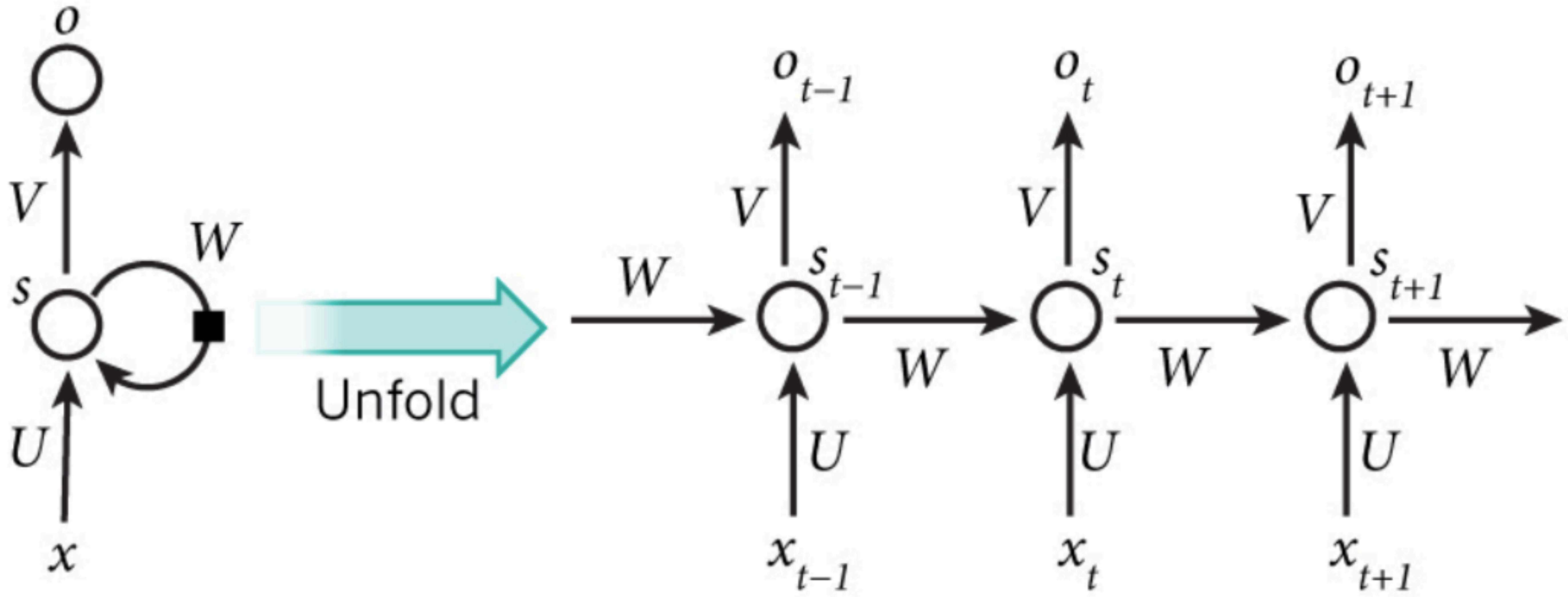
Example: POS Tagging

- Part of Speech Tagging:
  - Labeling words with their Part-Of-Speech (Noun, Verb, Adjective, …)
  - x_t: a vector to represent the word at step t
  - y_t: label of word t



Recurrent Neural Network

Example: POS Tagging



- x_t: t-th input
- s_t: hidden state at time t (`memory'' of the network)
  - $s_t = f(Ux_t + Ws_{t-1})$
  - W: transition matrix, U: word embedding matrix, s_0 usually set to be 0
- Predicted output at time t:
  - $o_t = \arg\max_i (Vs_t)_i$

Recurrent Neural Network

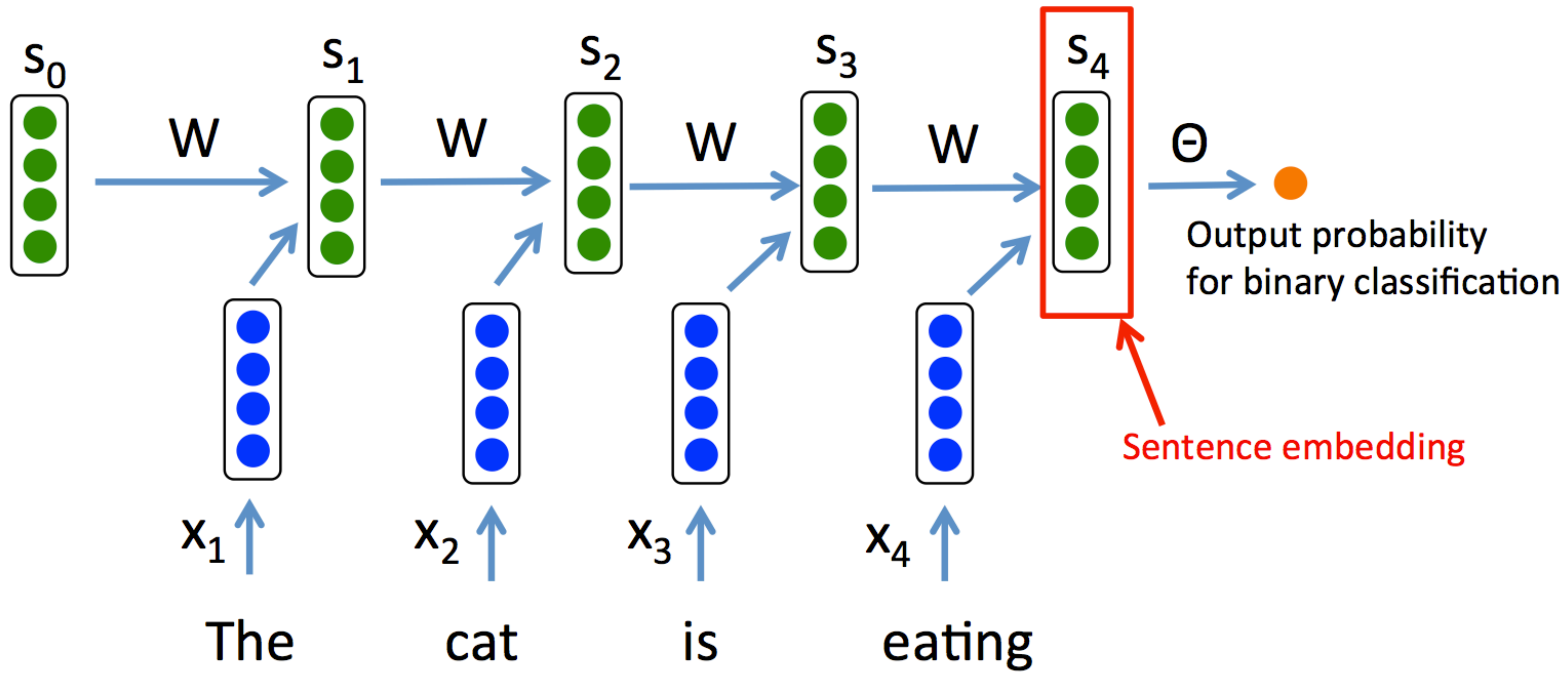
Recurrent Neural Network (RNN)

- Training: Find U, W, V to minimize empirical loss:
- Loss of a sequence:
  - $\sum_{t=1}^{T} \text{loss}(Vs_t, y_t)$
  - (s_t is a function of U, W, V)
- Loss on the whole dataset:
  - Average loss over all sequences
- Solved by SGD/Adam

Recurrent Neural Network

RNN: Text Classification

- Not necessary to output at each step
- Text Classification:
  - sentence → category
  - Output only at the final step
- Model: add a fully connected network to the final embedding



Recurrent Neural Network

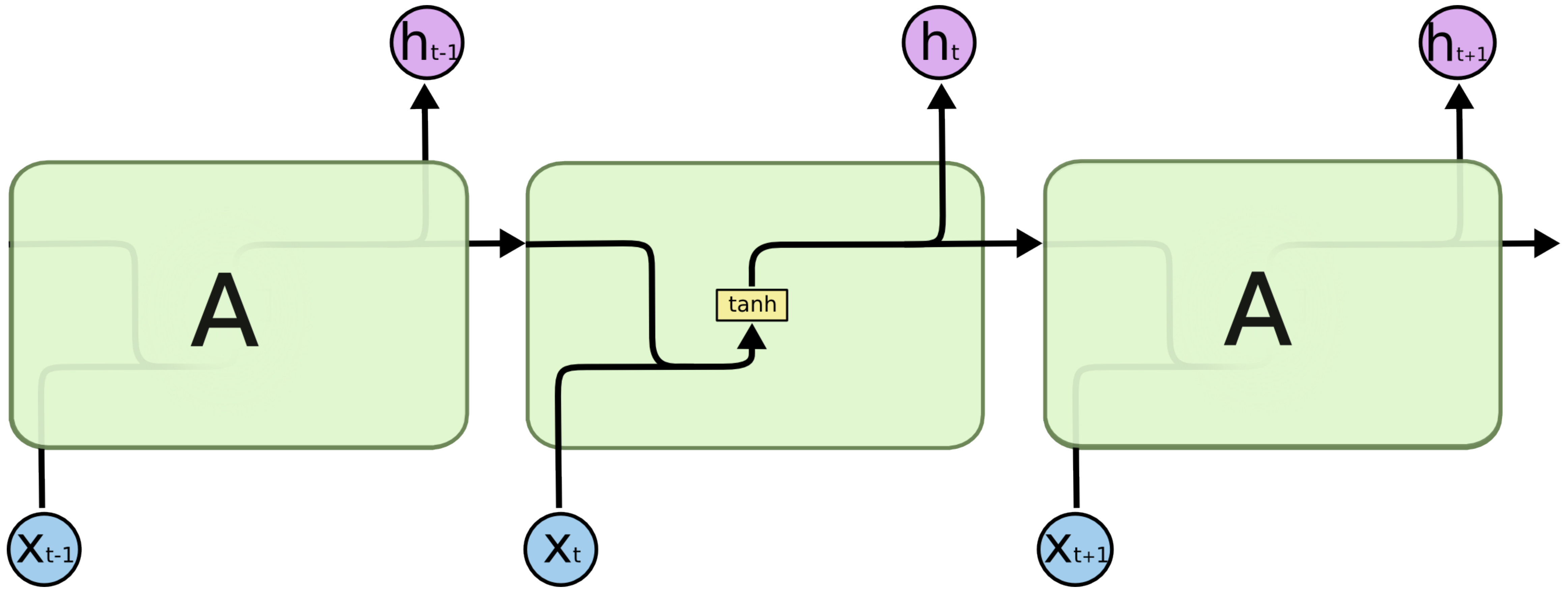
Problems of Classical RNN

- Hard to capture long-term dependencies
- Hard to solve (vanishing gradient problem)
- Solution:
  - LSTM (Long Short Term Memory networks)
  - GRU (Gated Recurrent Unit)
  - …

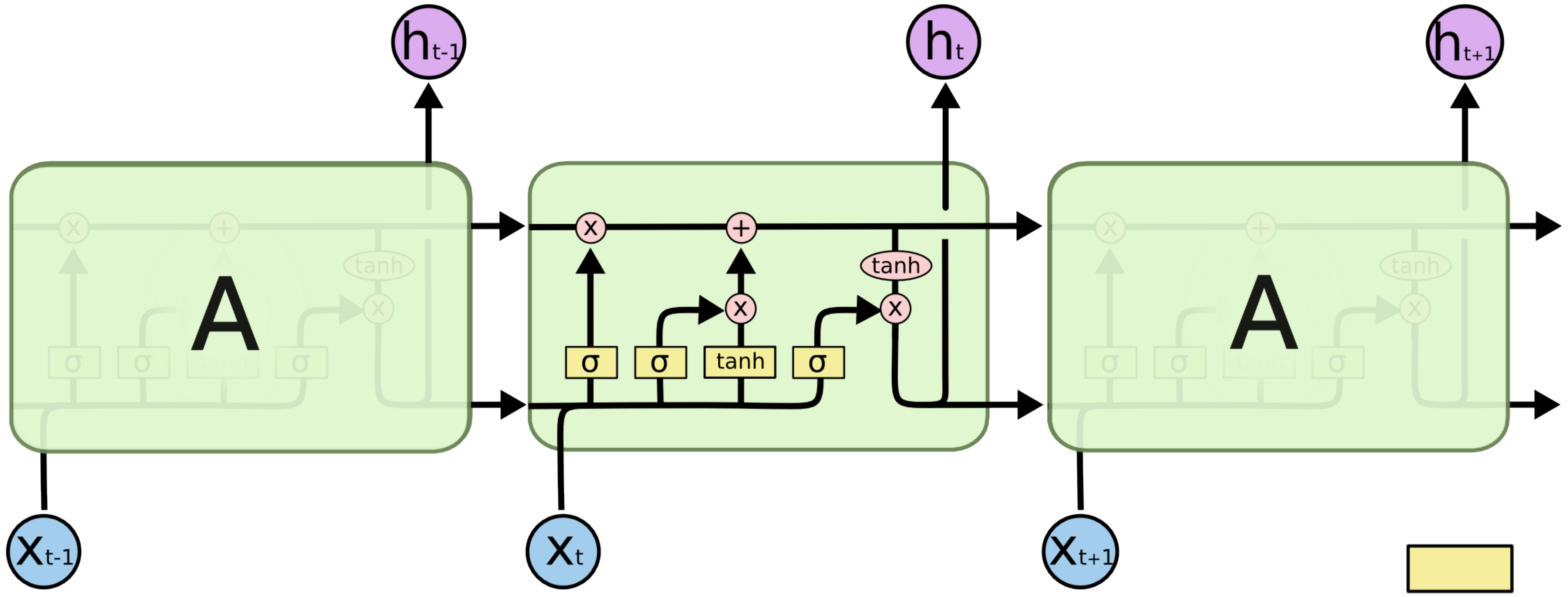
Recurrent Neural Network

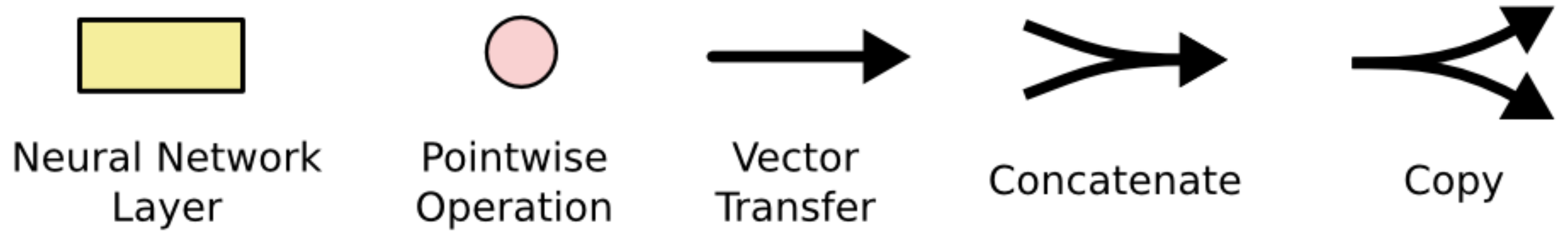
LSTM

- RNN:



- LSTM:

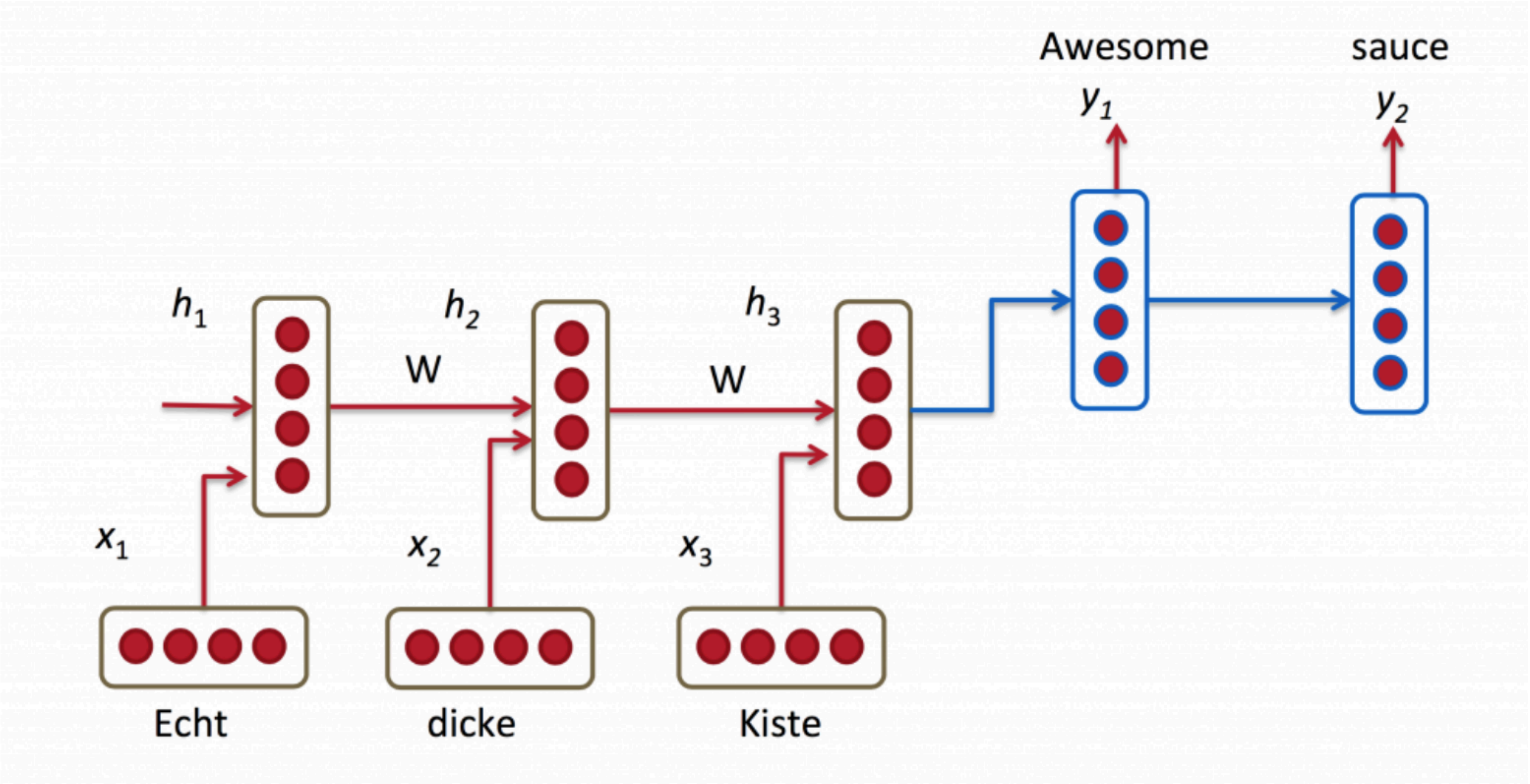




Recurrent Neural Network

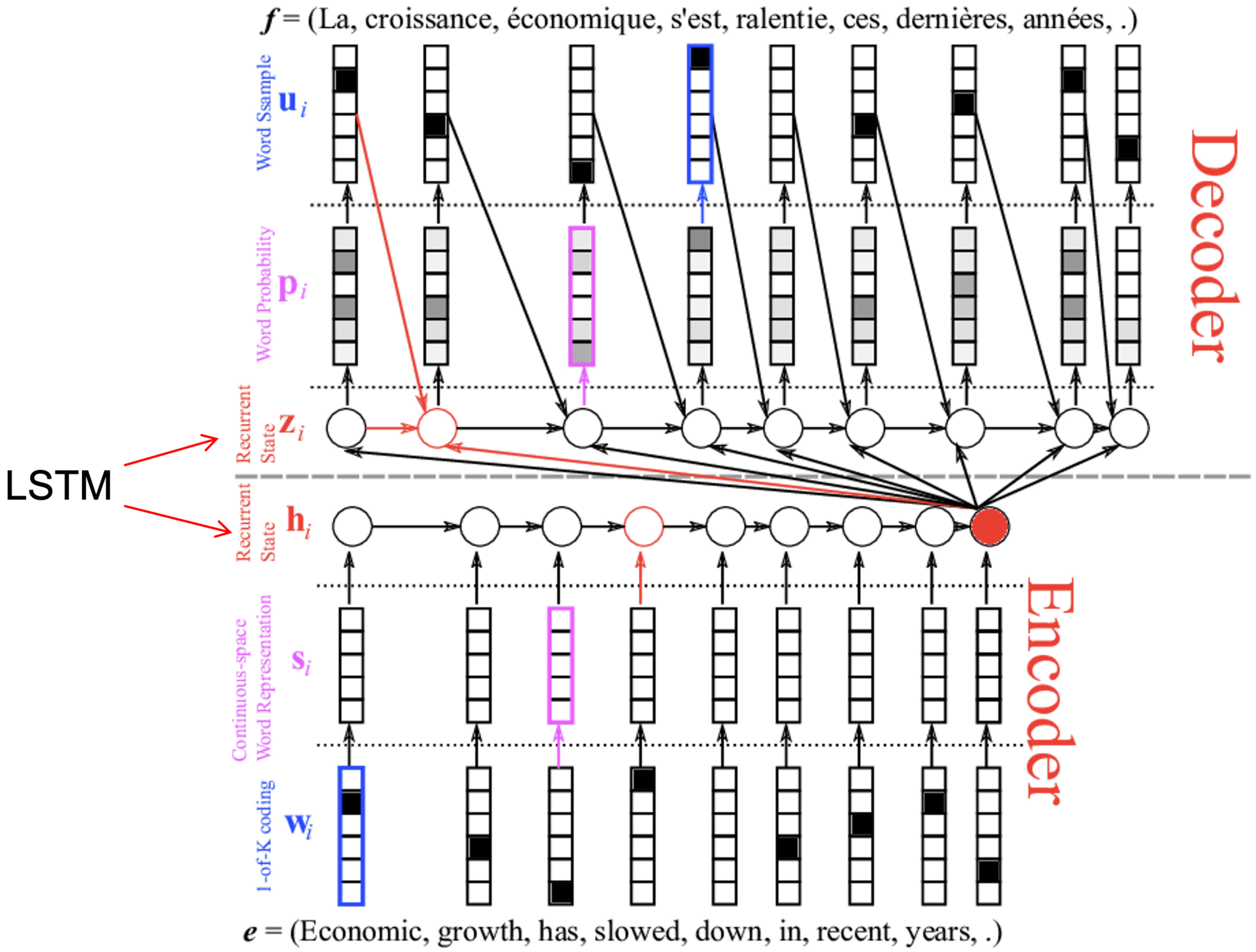
Neural Machine Translation (NMT)

- Out the translated sentence from an input sentence
- Training data: a set of input-output pairs (supervised setting)
- Encoder-decoder approach:
    - Encoder: Use (RNN/LSTM) to encode the input sentence input a latent vector
    - Decoder: Use (RNN/LSTM) to generate a sentence based on the latent vector



Recurrent Neural Network

Neural Machine Translation



Recurrent Neural Network

Attention in NMT

- Usually, each output word is only related to a subset of input words (e.g., for machine translation)
- Let u be the current decoder latent state, $v_1, \ldots, v_n$ be the latent sate for each input word
- Compute the weight of each state by
  - $p = \text{Softmax}(u^T v_1, \ldots, u^T v_n)$
- Compute the context vector by $Vp = p_1 v_1 + \ldots + p_n v_n$

Recurrent Neural Network

Attention in NMT