CS 540 Introduction to Artificial Intelligence
Review: Neural Networks and Deep Learning

University of Wisconsin–Madison

Fall 2025, Section 3

October 31, 2025

Announcements

- HW 6 online:
  - Due tonight **10/31 11:59PM**
- HW7 out, due **November 14**
  - Two weeks!
  - Long assignment!
  - Start early!

Neural Networks: Perceptron
Neural Networks: MLP
Deep Learning: CNNs
Deep Learning: ResNets
Deep Learning: RNNs and Transformers
Neural Networks & Deep Learning Review
Search, Games, and Reinforcement Learning

What have we seen?

- Perceptron
- Multilayer Perceptron
- Convolutional Neural Network
  - Filters
  - Padding, Stride
  - Pooling
- ResNet
- Graph Neural Network (briefly)
- Recurrent Neural Network (briefly)
- Transformer
  - Attention

- Loss functions
- Activation functions
- Softmax
- (Stochastic) Gradient Descent
- Backpropagation

Today's Plan:
1. More on transformers
2. Neural networks review

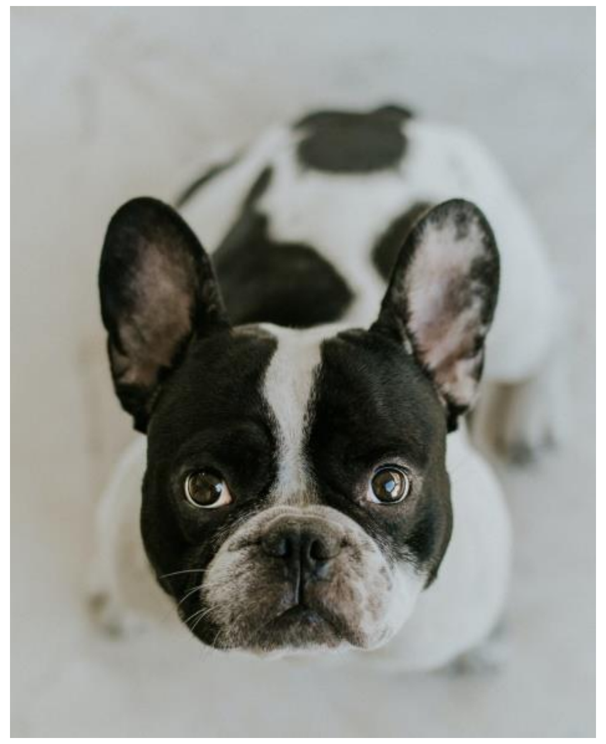
Transformers & Attention

Word Representations and Context

- We use vectors to represent words ("embeddings")
- Recall:
  - One-hot representation
  - Dense embedding
    - Vector captures **meaning**

"dog"

$$[0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0]$$



$$[0.13 \quad 0.87 \quad -0.23 \quad 0.46\,]$$

The attention mechanism produces **contextual embeddings**.

Attempt 1: Naïve Contextual Embedding

- Each token has a fixed embedding vector x_i
- A crude attempt at contextual embedding: average over context
- Equal "attention" to every previous token

Tokens		Fixed Embeddings
1	the	[0.45 0.23]
2	monkey	[0.39 0.72]
3	ate	[0.83 0.61]
4	the	[0.45 0.23]
5	banana	[0.25 0.18]
6	it	[0.63 0.41]
7	was	[0.63 0.41]
8	ripe	[0.70 0.67]
9	wasn't	[0.14 0.61]
10	it	+ [0.63 0.41]
		[4.98 4.93]

In math:
for the i-th token

$$c_i = \frac{1}{i}\sum_{j=1}^{i} x_j$$

Contextual embedding for "it": [0.498 0.493]

Attempt 2: Assigning Weights

- Humans focus selectively — machines can too
- We can assign weights based on relevance
  - Idea: weight similar words highly
  - If $\langle x_i, x_j \rangle$ large, assign large weight
- Then take weighted sum

			Weights	Fixed Embeddings
1	the	the	0.05	[0.45 0.23]
2	monkey	monkey	0.13	[0.39 0.72]
3	ate	ate	0.06	[0.83 0.61]
4	the	the	0.05	[0.45 0.23]
5	banana	banana	0.10	[0.25 0.18]
6	it	it	0.15	[0.63 0.41]
7	was	was	0.03	[0.63 0.41]
8	ripe	ripe	0.14	[0.70 0.67]
9	wasn't	wasn't	0.07	[0.14 0.61]
10	it	it	+ 0.15	[0.63 0.41]

Contextual embedding for "it": [0.37 0.42]

In math: for i-th token

$$r_{ij} = \frac{1}{\sqrt{d}} \langle x_i, x_j \rangle \qquad c_i = \sum_{j=1}^{i} p_{ij} \cdot x_j$$

$$p_{i,:} = \mathrm{softmax}(r_{i,:})$$

Final Attempt: The Attention Mechanism

Previous attempt:

- Used fixed embeddings in three locations.

$$r_{ij} = \frac{1}{\sqrt{d}} \cdot \langle x_i, x_j \rangle$$

$$p_{i,:} = \mathrm{softmax}(r_{i,:})$$

$$c_i = \sum_{j=1}^{i} p_{ij} \cdot x_j$$

In the attention mechanism:

- Each token is associated with **three** vectors
- Query: q_i, the one attended from
- Key: k_j, the one attended to
- Value: v_j, the context being generated

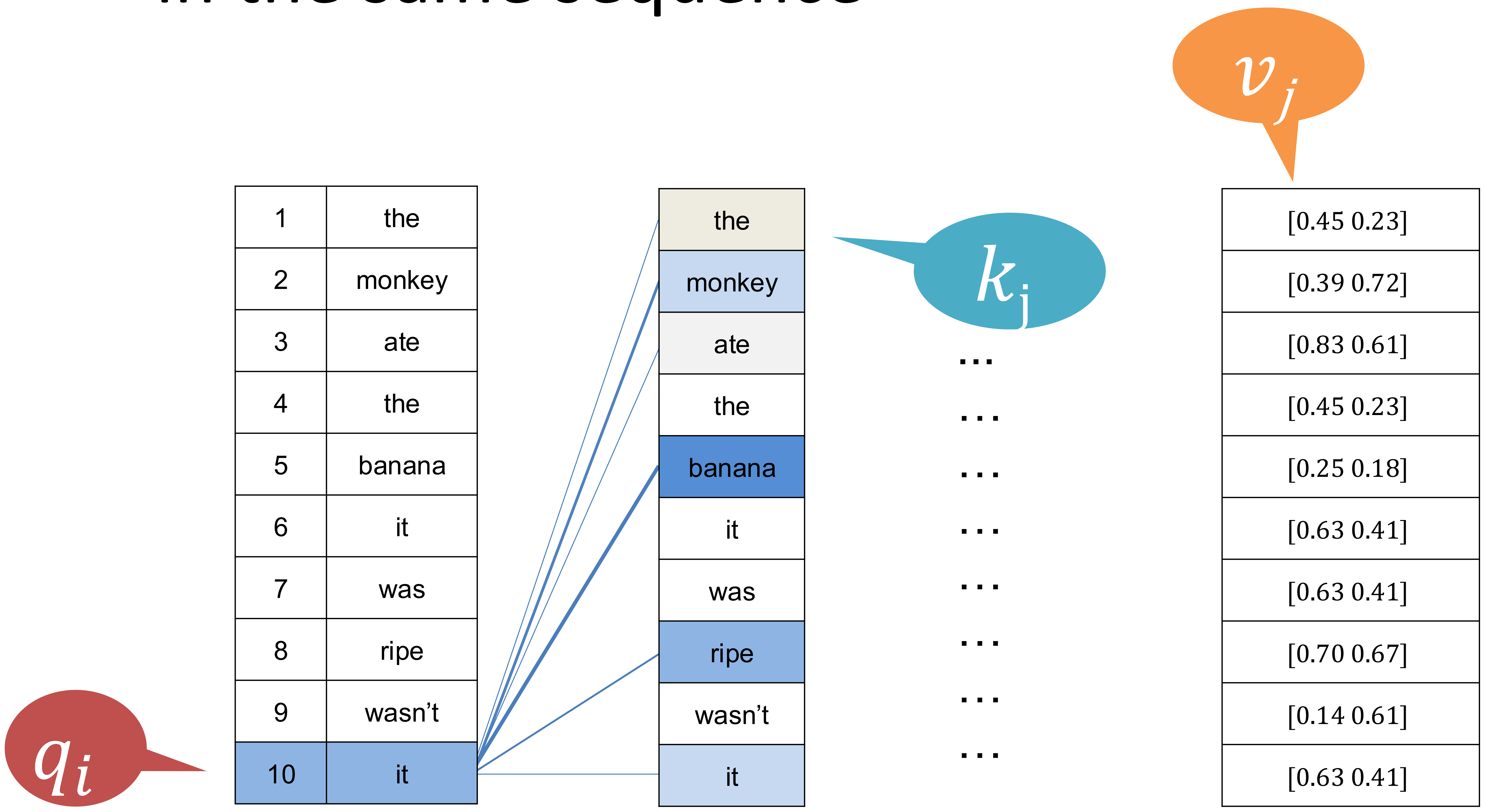
$$r_{ij} = \frac{1}{\sqrt{d}} \cdot \langle \overbrace{q_i}^{\text{Query}}, \overbrace{k_j}^{\text{Key}} \rangle$$

$$p_{i,:} = \mathrm{softmax}(r_{i,:})$$

$$c_i = \sum_{j=1}^{i} p_{ij} \cdot \overbrace{v_j}^{\text{Value}}$$

The Attention Mechanism

Each token attends to all previous tokens in the same sequence

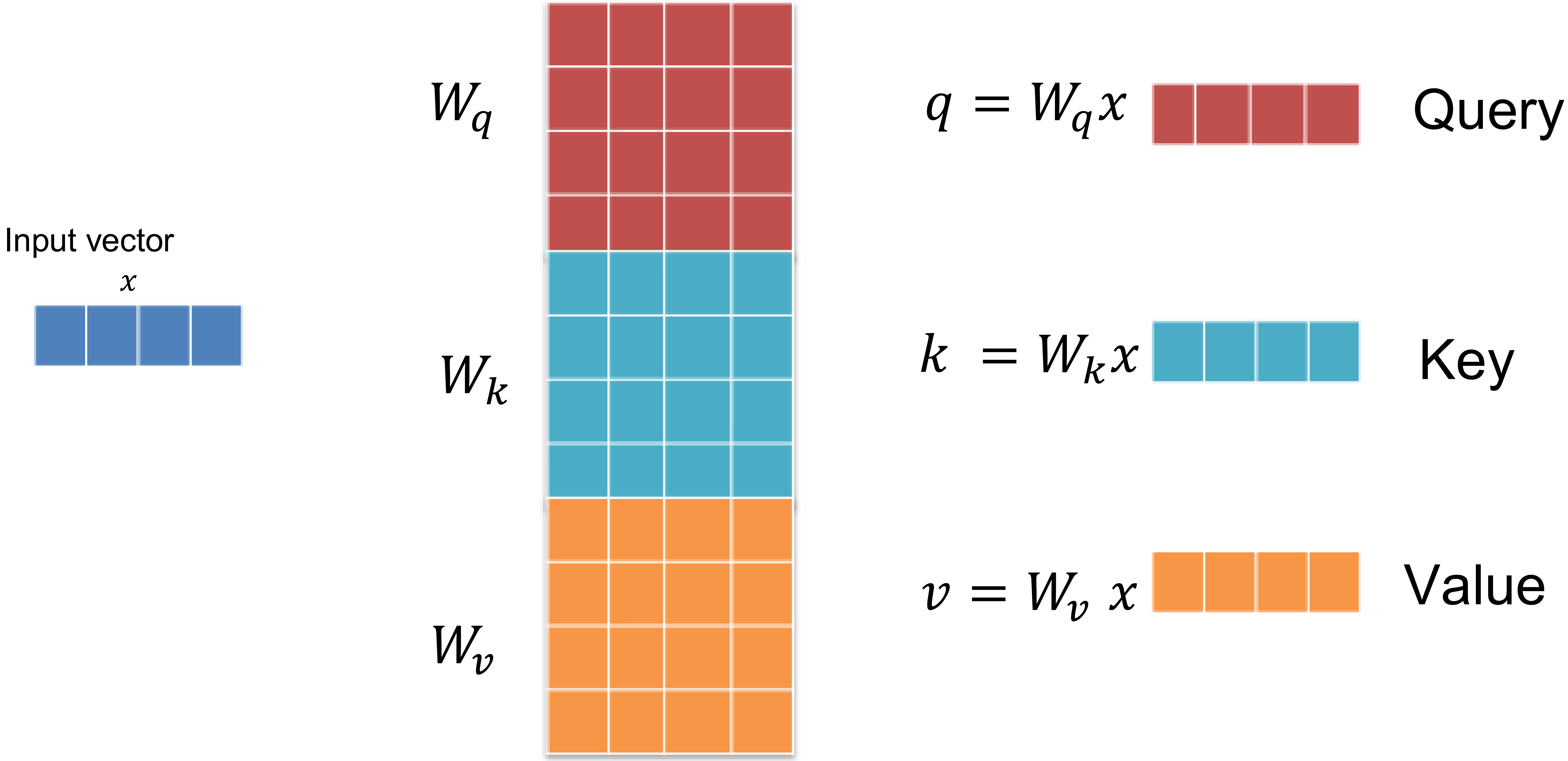


$$r_{ij} = \frac{\langle q_i, k_j \rangle}{\sqrt{d}}$$

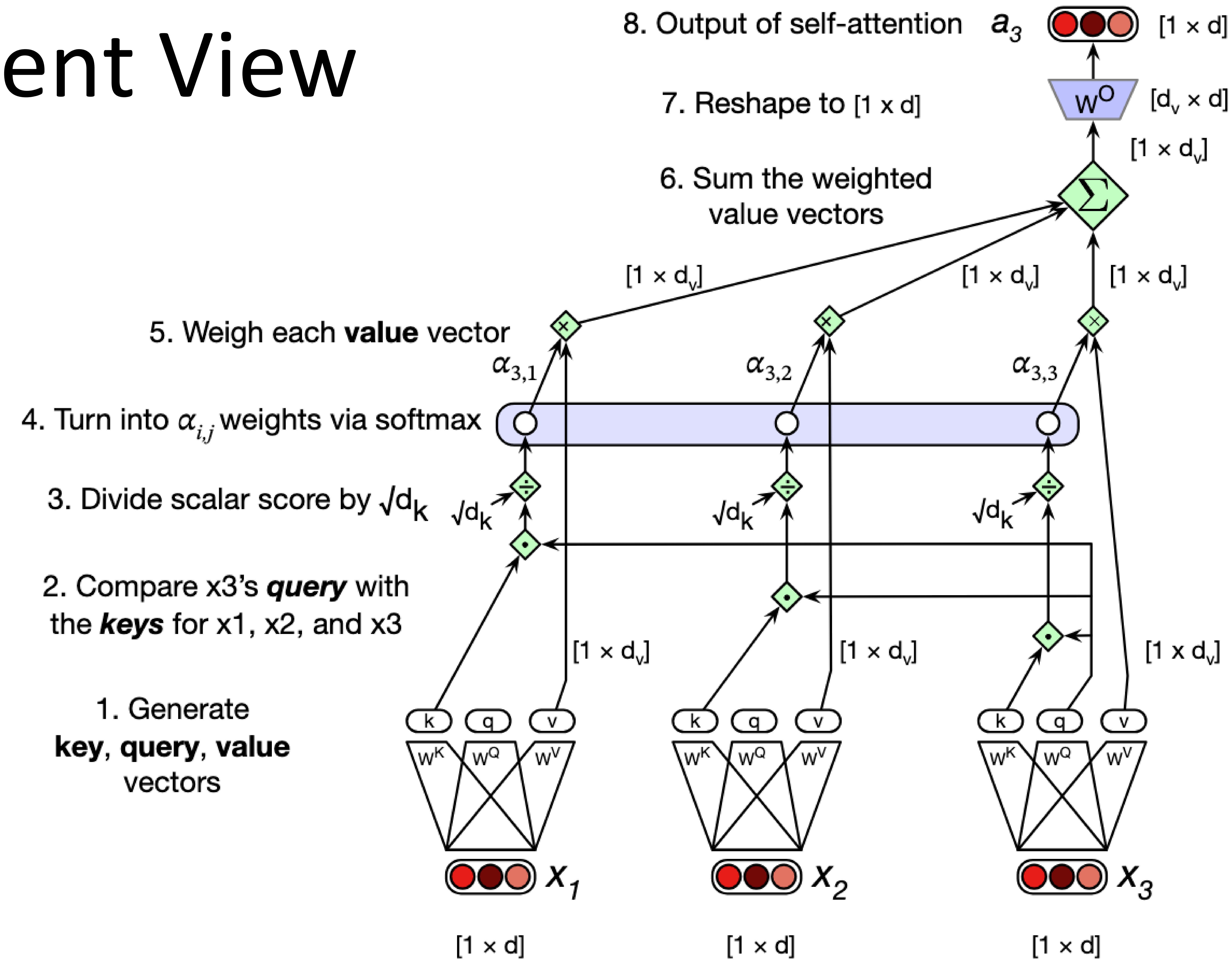
$$p_{i,:} = \mathrm{softmax}(r_{i,:})$$

$$c_i = \sum_j p_{ij} \cdot v_j$$

Query, Key, and Value Matrices



A Different View

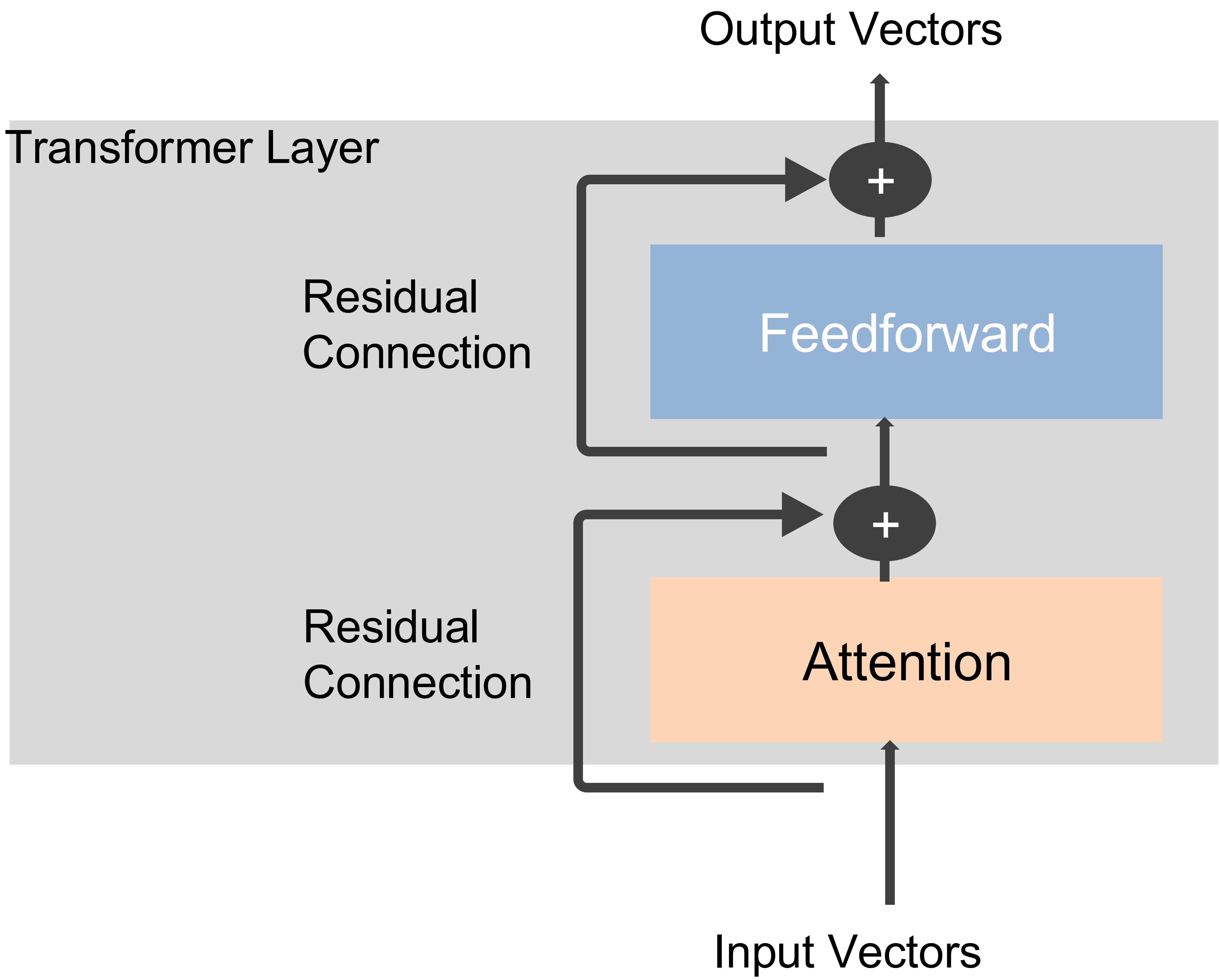


Jurafsky and Martin, *Speech and Language Processing*
https://web.stanford.edu/~jurafsky/slp3/ed3book_aug25.pdf

From Attention to Transformer

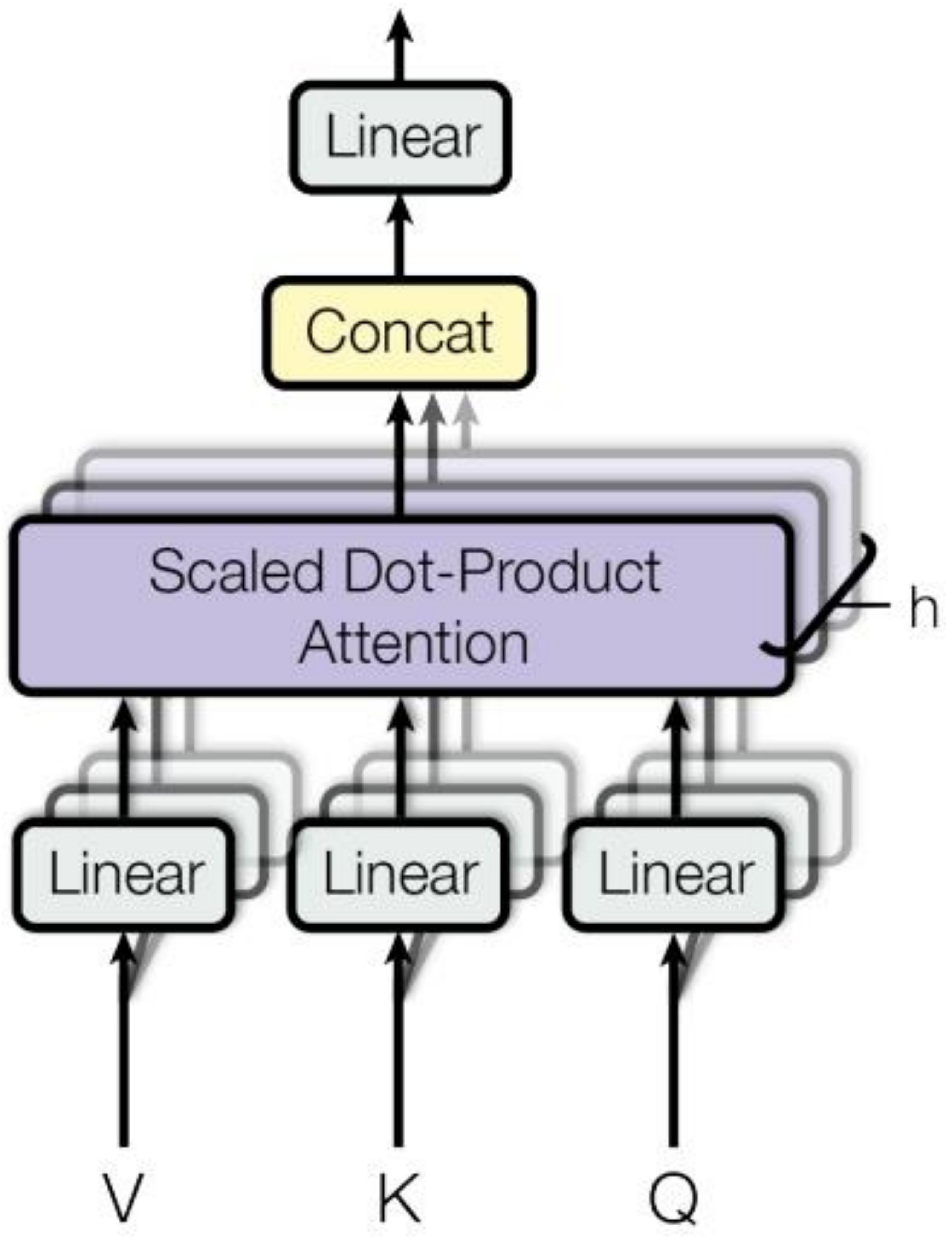
A single layer transformer consists of:

- Attention Mechanism
- Feed-Forward Network
- Residual Connections



Multi-Head Attention

- Outputs combined for richer representations
- Multiple heads learn different relationships
  (syntax, meaning, position)

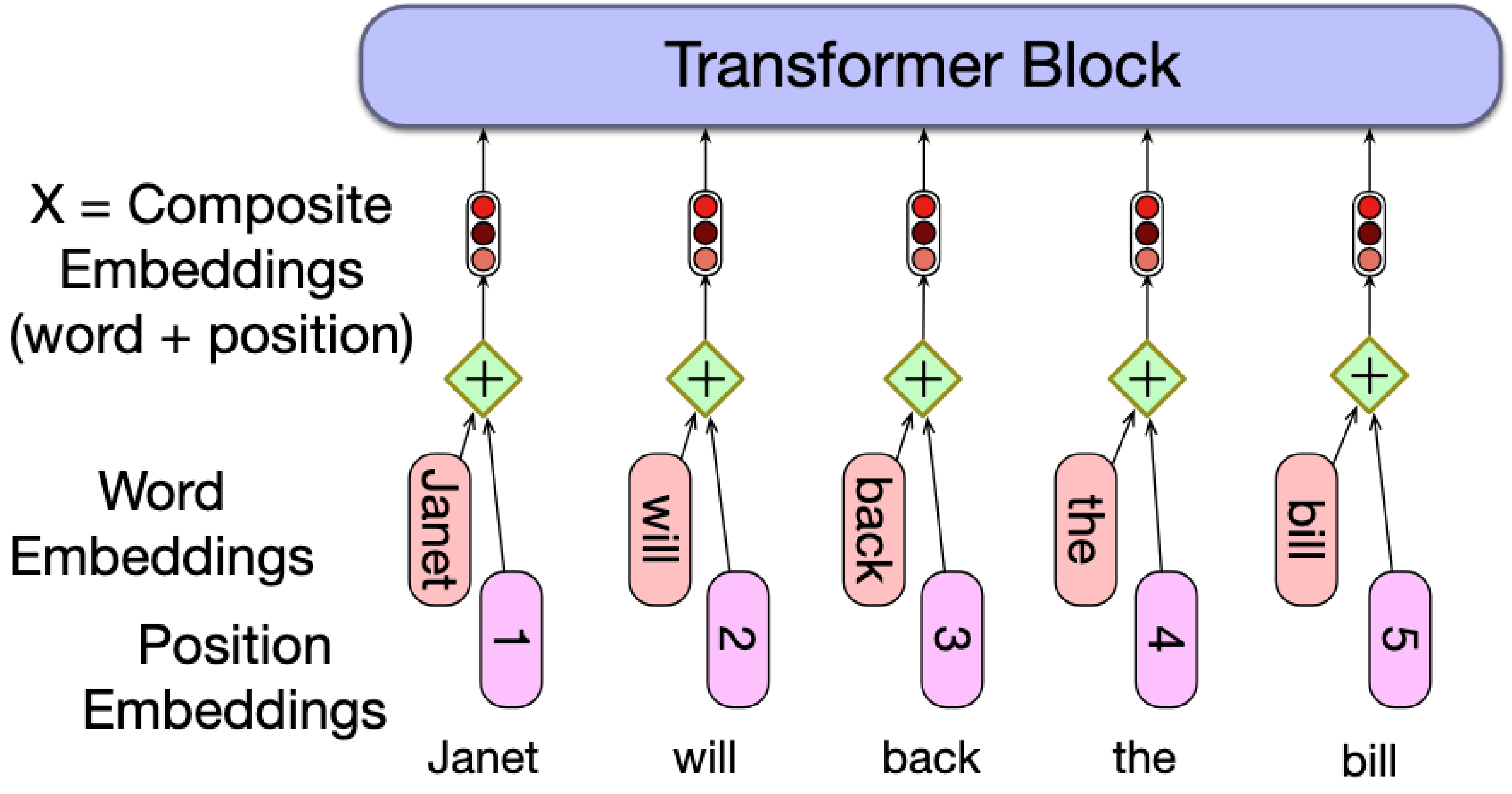


[Vaswani, A., et al. (2017). Attention Is All You Need](#)

Positional Encoding

- Transformers have no recurrence — order must be added explicitly
- **Positional Encoding**: Information about the relative or absolute position of the tokens in the sequence
- Added to the input embeddings

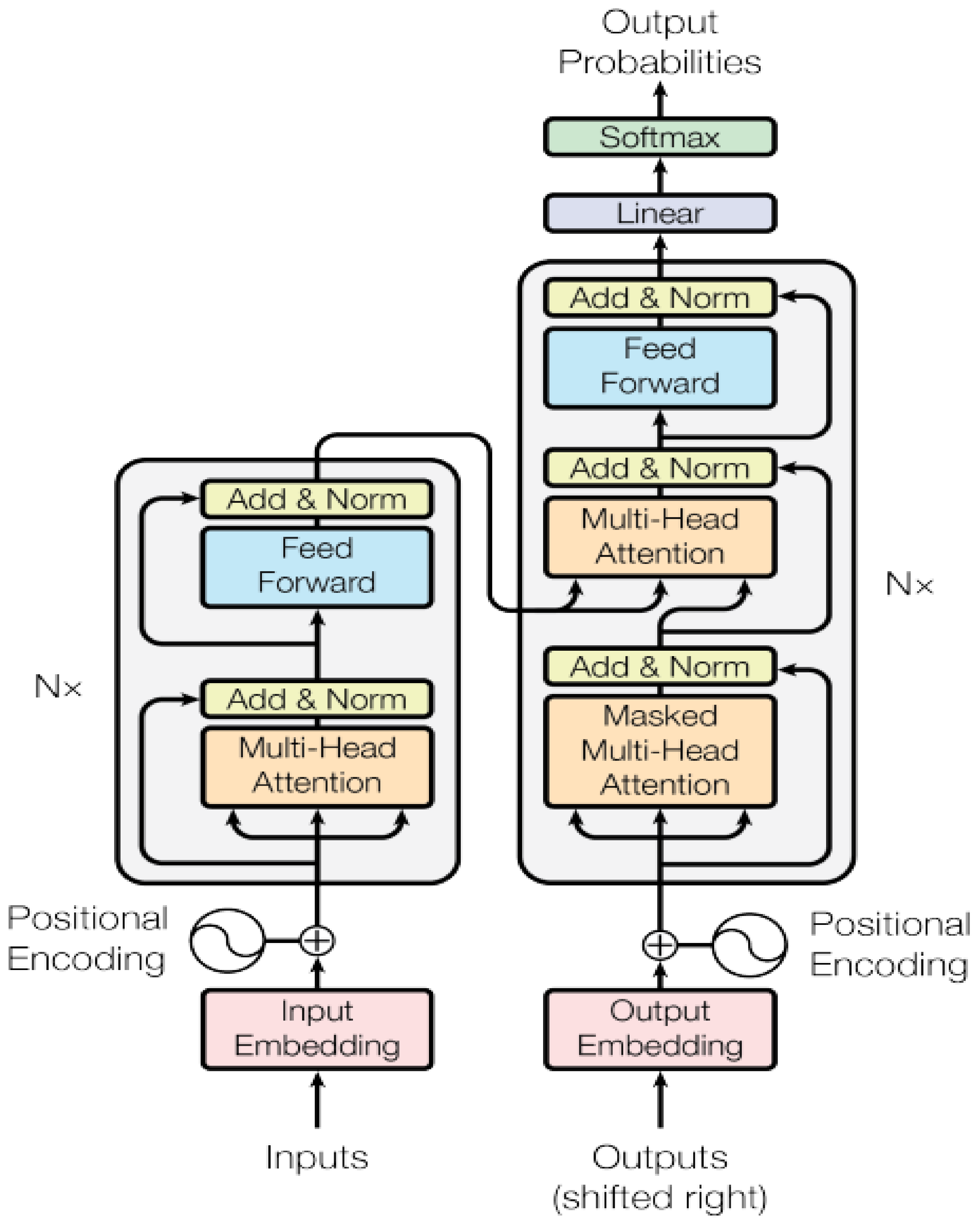
1	the	the	[0.45 0.23]
2	monkey	monkey	[0.39 0.72]
3	ate	ate	[0.83 0.61]
4	the	the	[0.45 0.23]
5	banana	banana	[0.25 0.18]
6	it	it	[0.63 0.41]
7	was	was	[0.63 0.41]
8	ripe	ripe	[0.70 0.67]
9	wasn't	wasn't	[0.14 0.61]
10	it	it	[0.63 0.41]



Jurafsky and Martin, *Speech and Language Processing*
https://web.stanford.edu/~jurafsky/slp3/ed3book_aug25.pdf

Transformer Architecture

- Encoder–Decoder structure
- **Encoder**: maps an input sequence to a sequence of continuous representations z.
  - Useful for classification/translation
- **Decoder**: Given z, the decoder generates an output sequence of symbols one element at a time.
  - Useful for generation



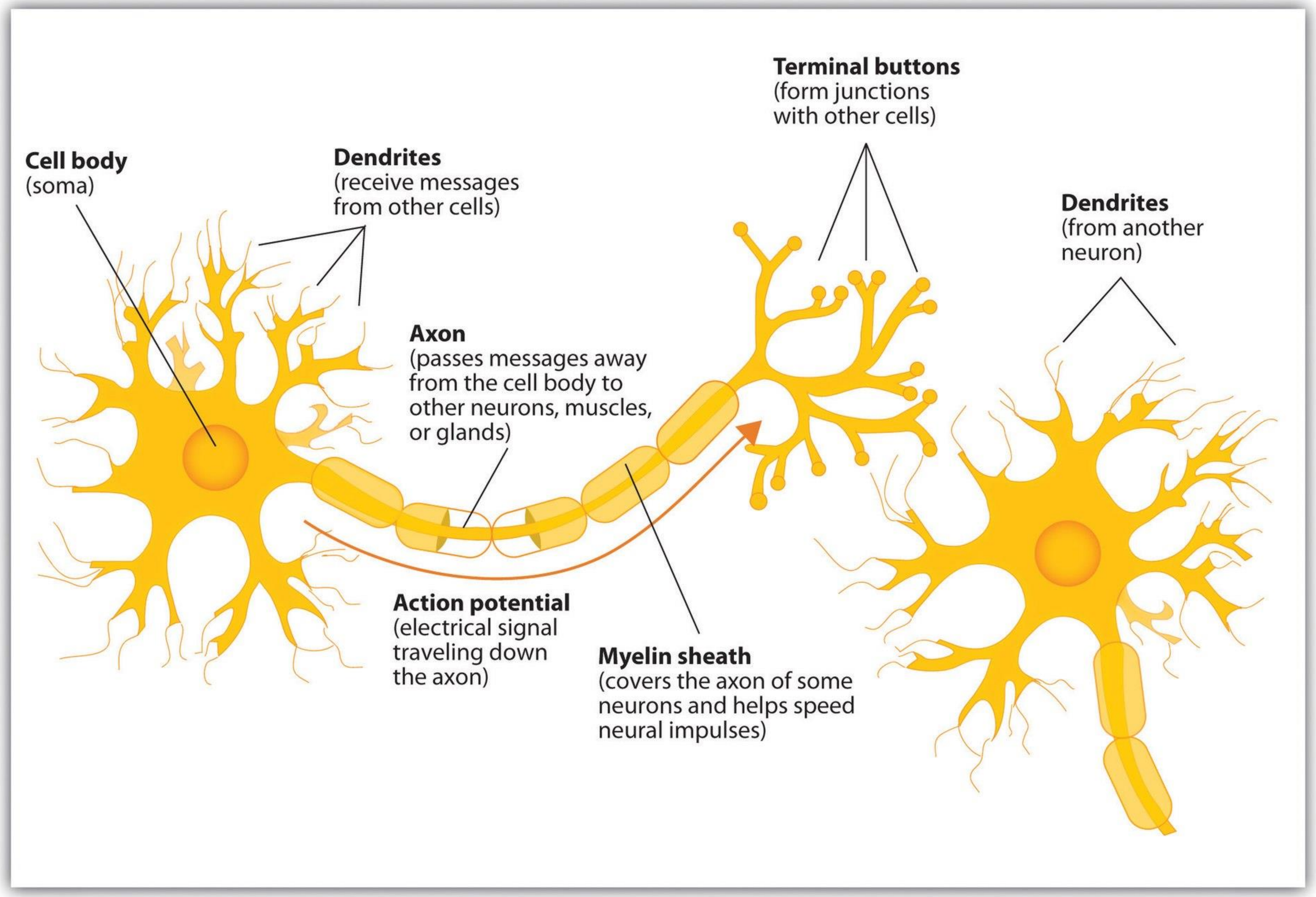
[Vaswani, A., et al. (2017). Attention Is All You Need](#)

Neural Networks Review

Inspiration from neuroscience

- Inspirations from human brains
- Networks of simple and homogenous units (a.k.a **neuron**)

(wikipedia)



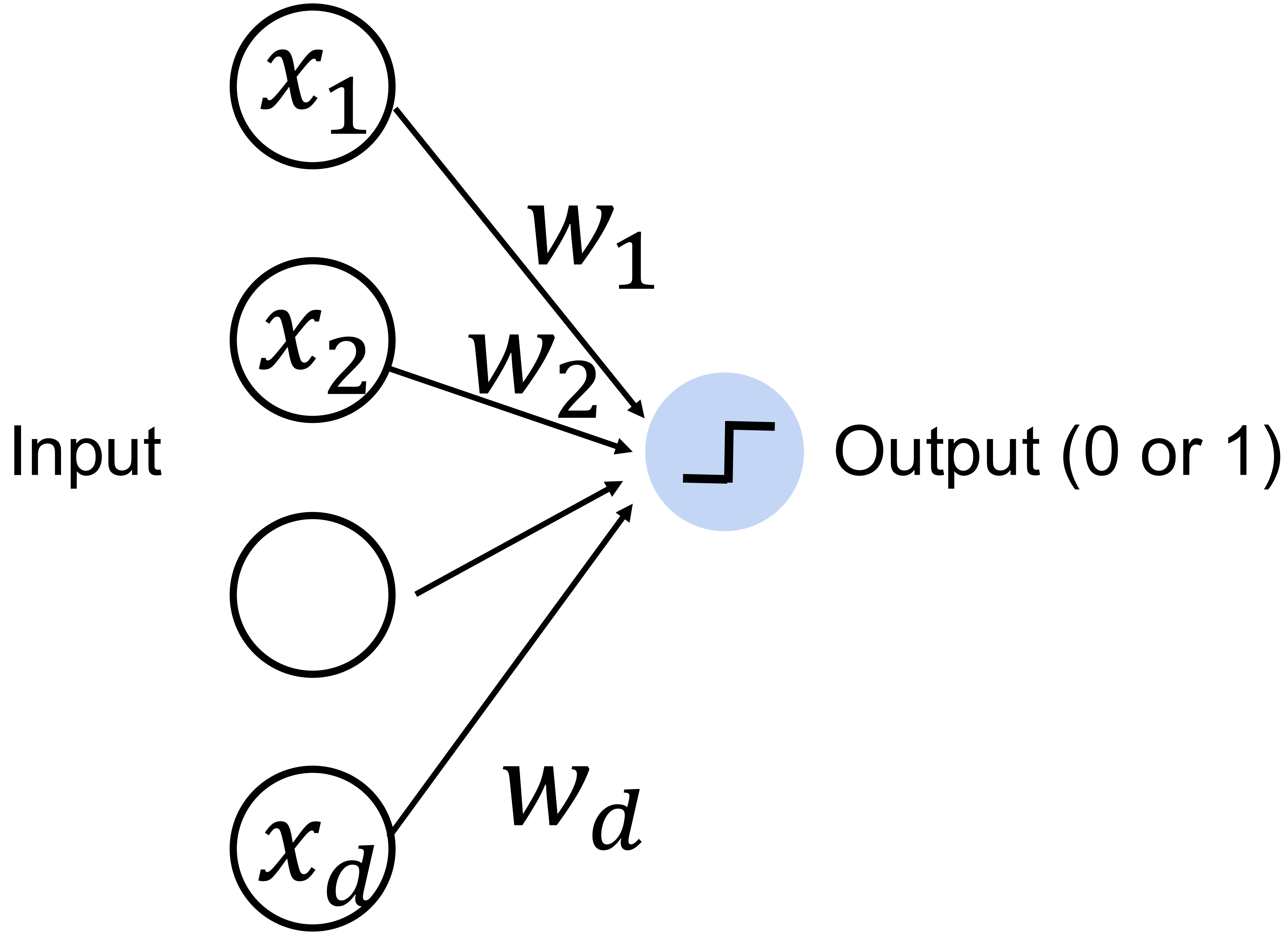
Perceptron

- Given input $\mathbf{x}$, weight $\mathbf{w}$ and bias b, perceptron outputs:

$$o = \sigma(\mathbf{w}^\top \mathbf{x} + b) \qquad \sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Activation function

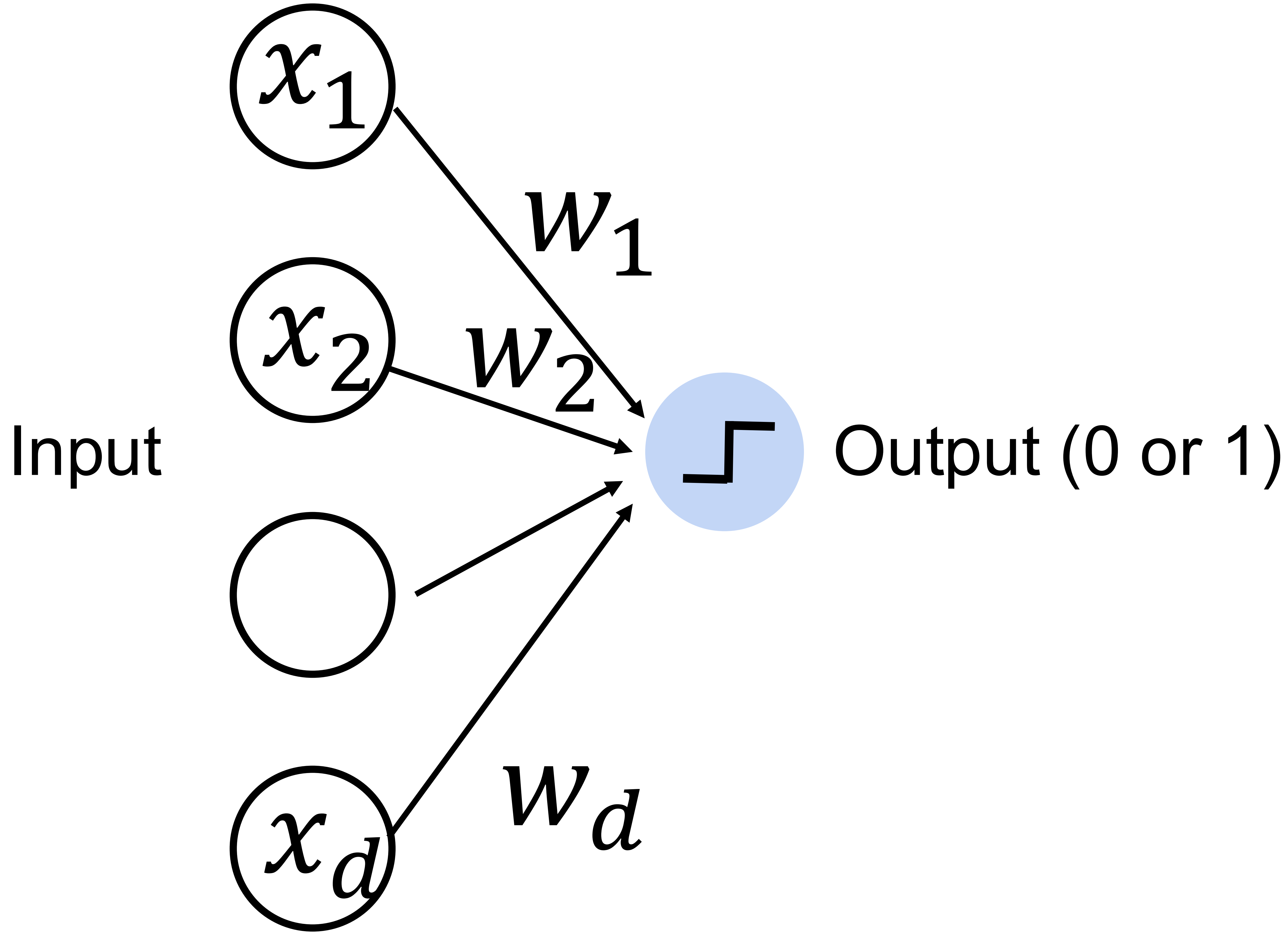
Cats vs. dogs?



Perceptron

- Goal: learn parameters $\mathbf{w} = \{w_1, w_2, \ldots, w_d\}$ and b to minimize the classification error

Cats vs. dogs?

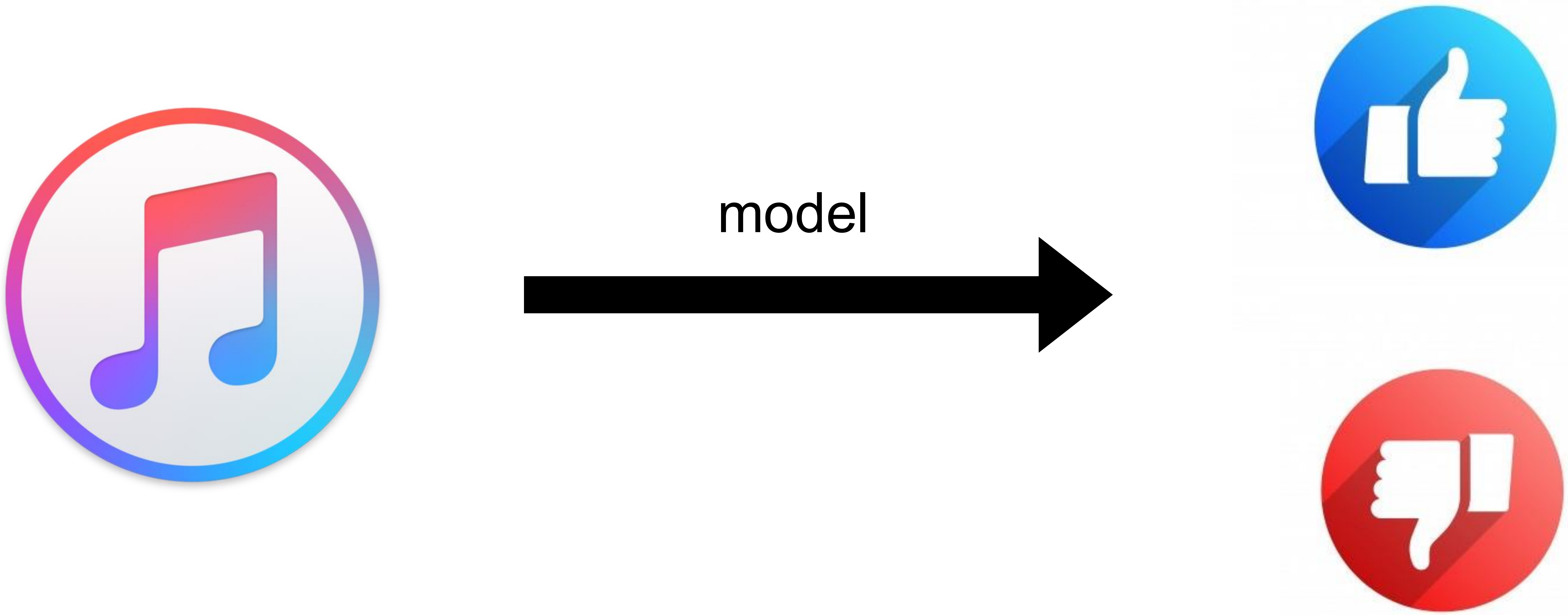


The Perceptron Learning Rule

Perceptron Learning Algorithm
<u>Input:</u> dataset (X, y) number of steps T, step size η
1. Initialize w_0, b_0 2. For $t = 1, 2, \dots, T$ 3. Pick random (x_i, y_i) 4. Predict $\hat{y}_i \leftarrow \sigma(\langle w_{t-1}, x_i \rangle + b_{t-1})$ 5. If $\hat{y}_i \neq y_\mathrm{i}$: 6. $w_t \leftarrow w_{t-1} + \eta(y_i - \hat{y}_i)x_i$ 7. $b_t \leftarrow b_{t-1} + \eta(y_i - \hat{y}_i)$ 8. Return w_T

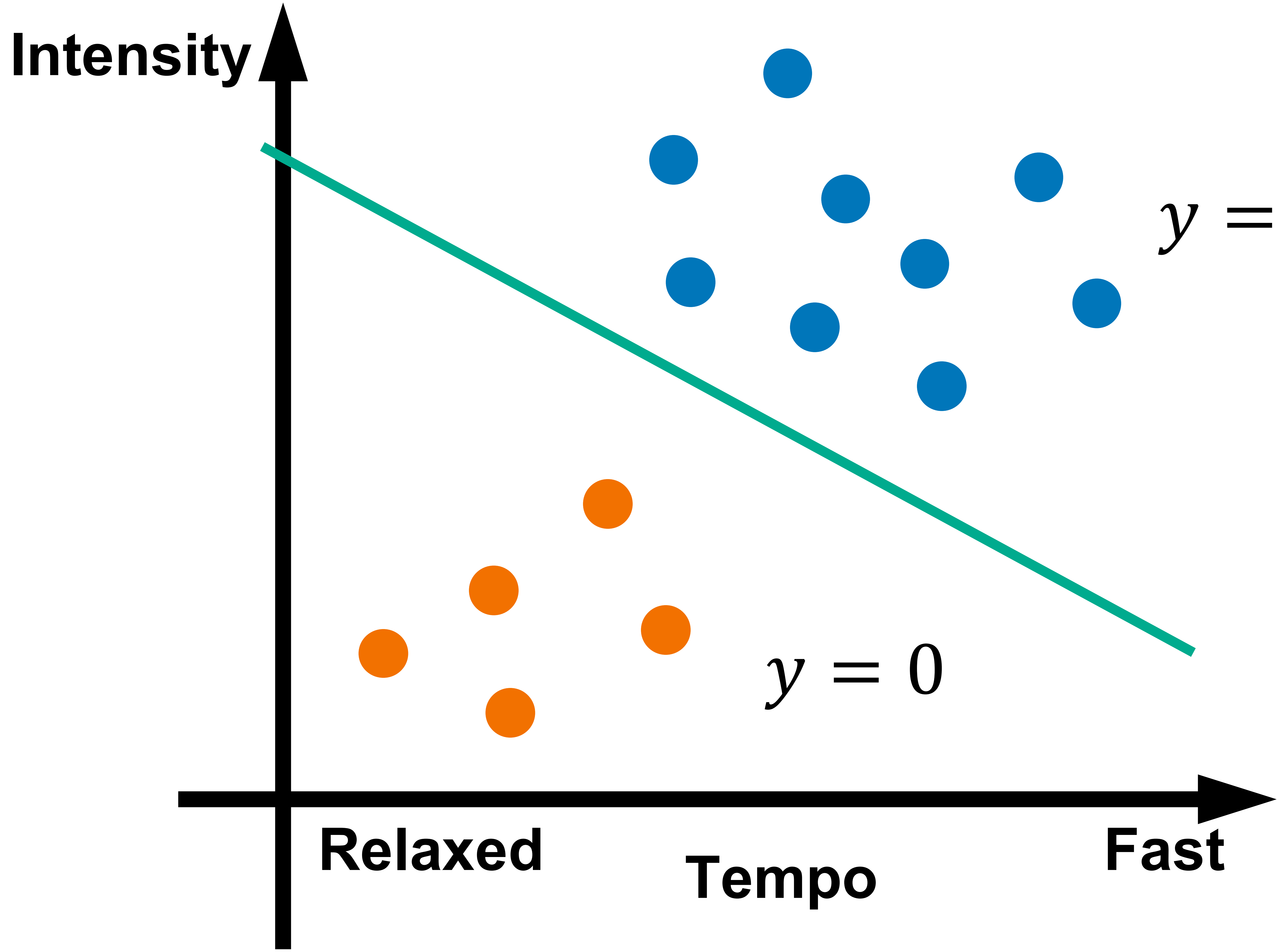
Gradient Descent
<u>Input:</u> dataset (X, y), loss function L, number of steps T, step size η
1. Initialize w_0 2. For $t = 1, 2, \dots, T$ 3. Calculate $g_t = \nabla L(w_{t-1}; X, y)$ 4. Update $w_t \leftarrow w_{t-1} - \eta g_t$ 5. Return w_T

Example 2: Predict whether a user likes a song or not



Example 2: Predict whether a user likes a song or not using Perceptron

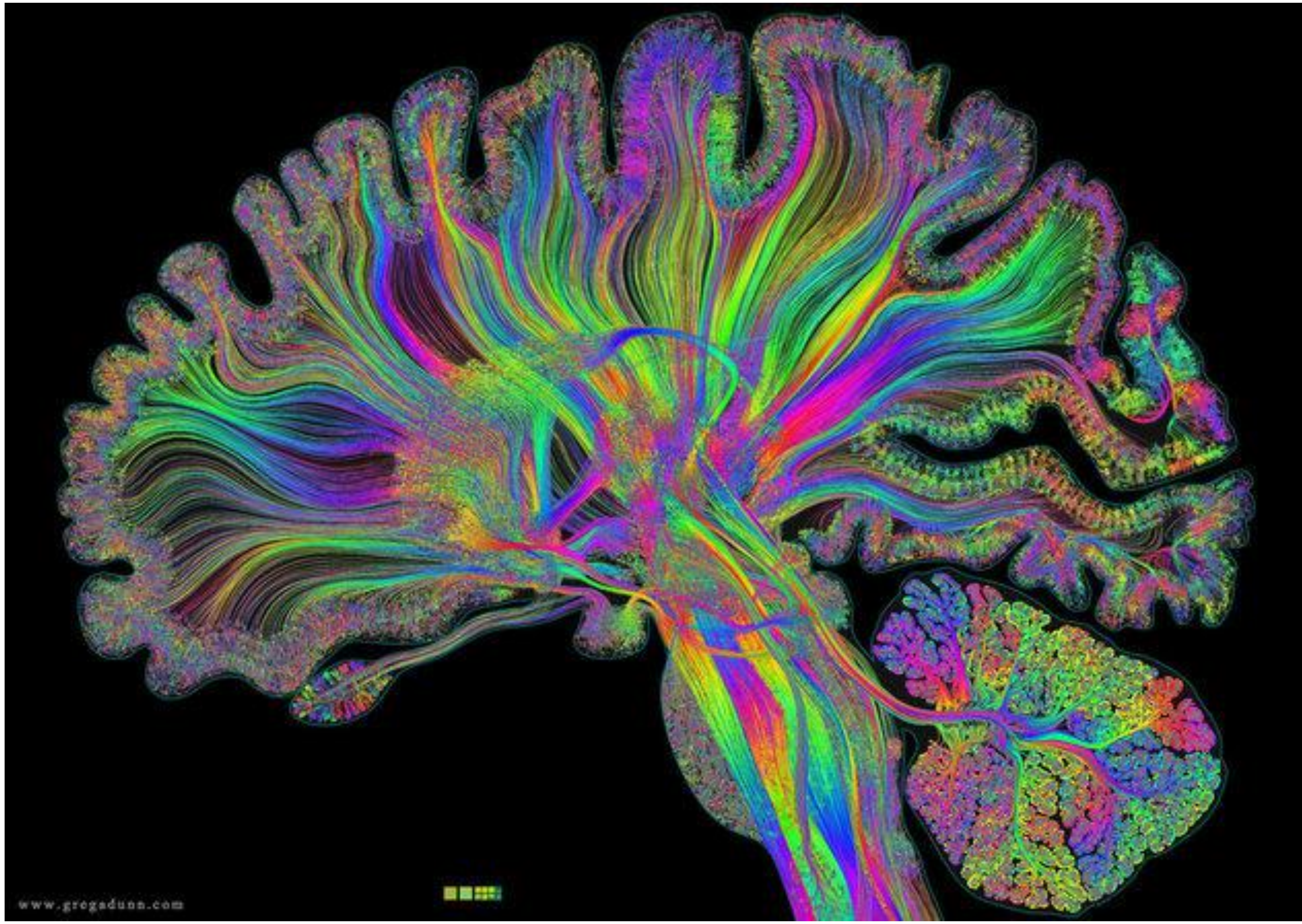
User Sharon



DisLike

Like

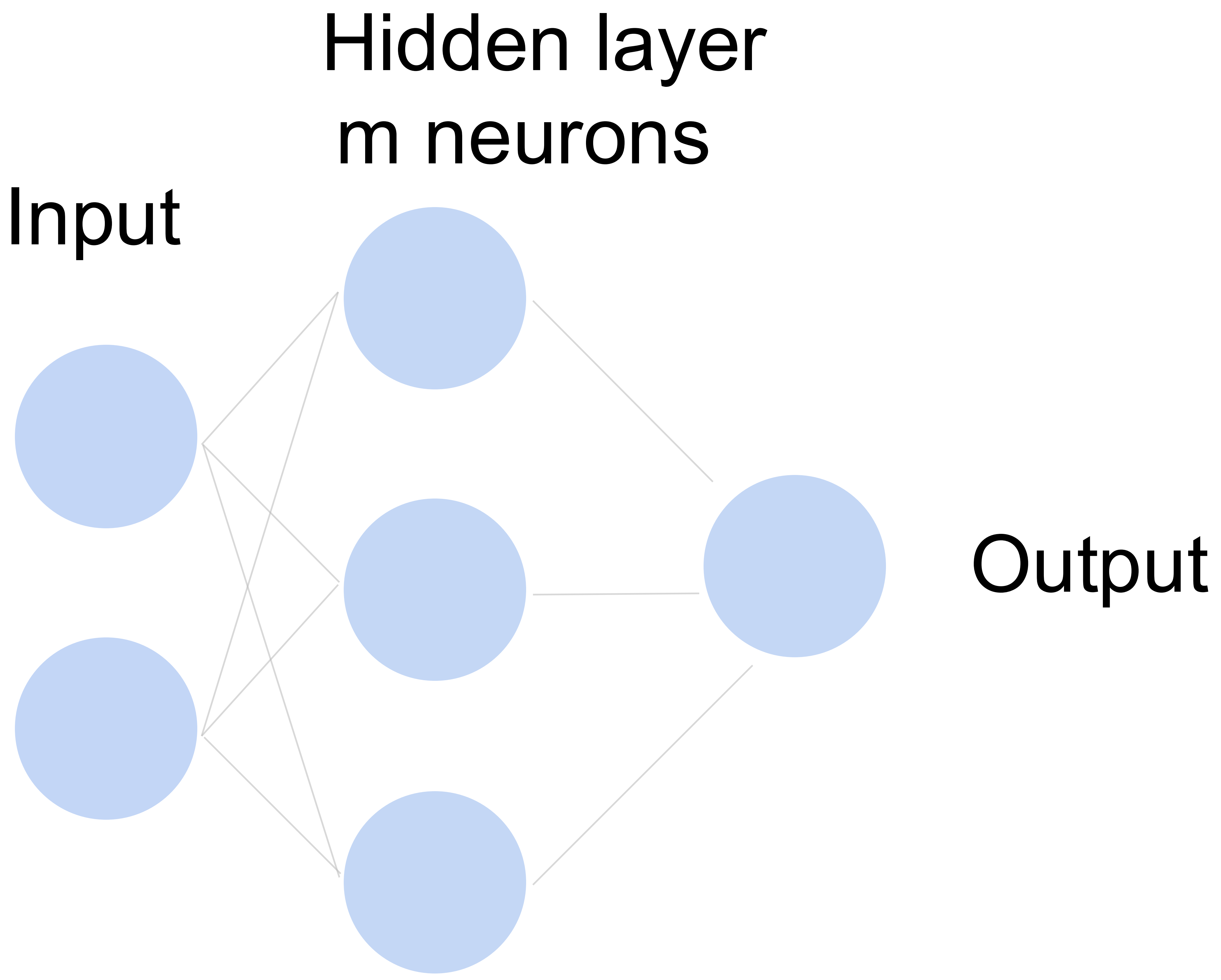
Multilayer Perceptron



Single Hidden Layer

How to classify

Cats vs. dogs?

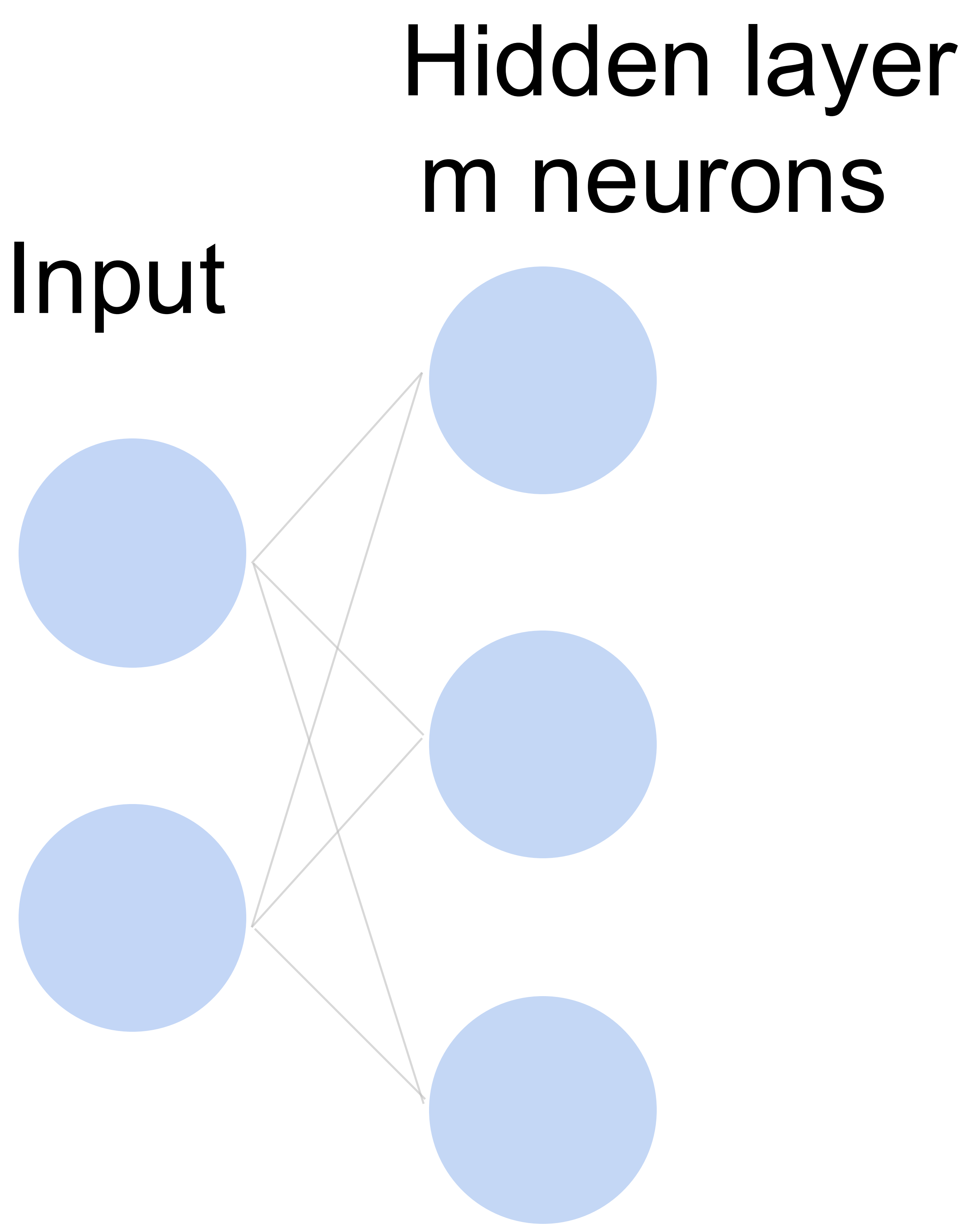


Single Hidden Layer

- Input $\mathbf{x} \in \mathbb{R}^d$
- Hidden $\mathbf{W} \in \mathbb{R}^{m \times d}, \mathbf{b} \in \mathbb{R}^m$
- Intermediate output

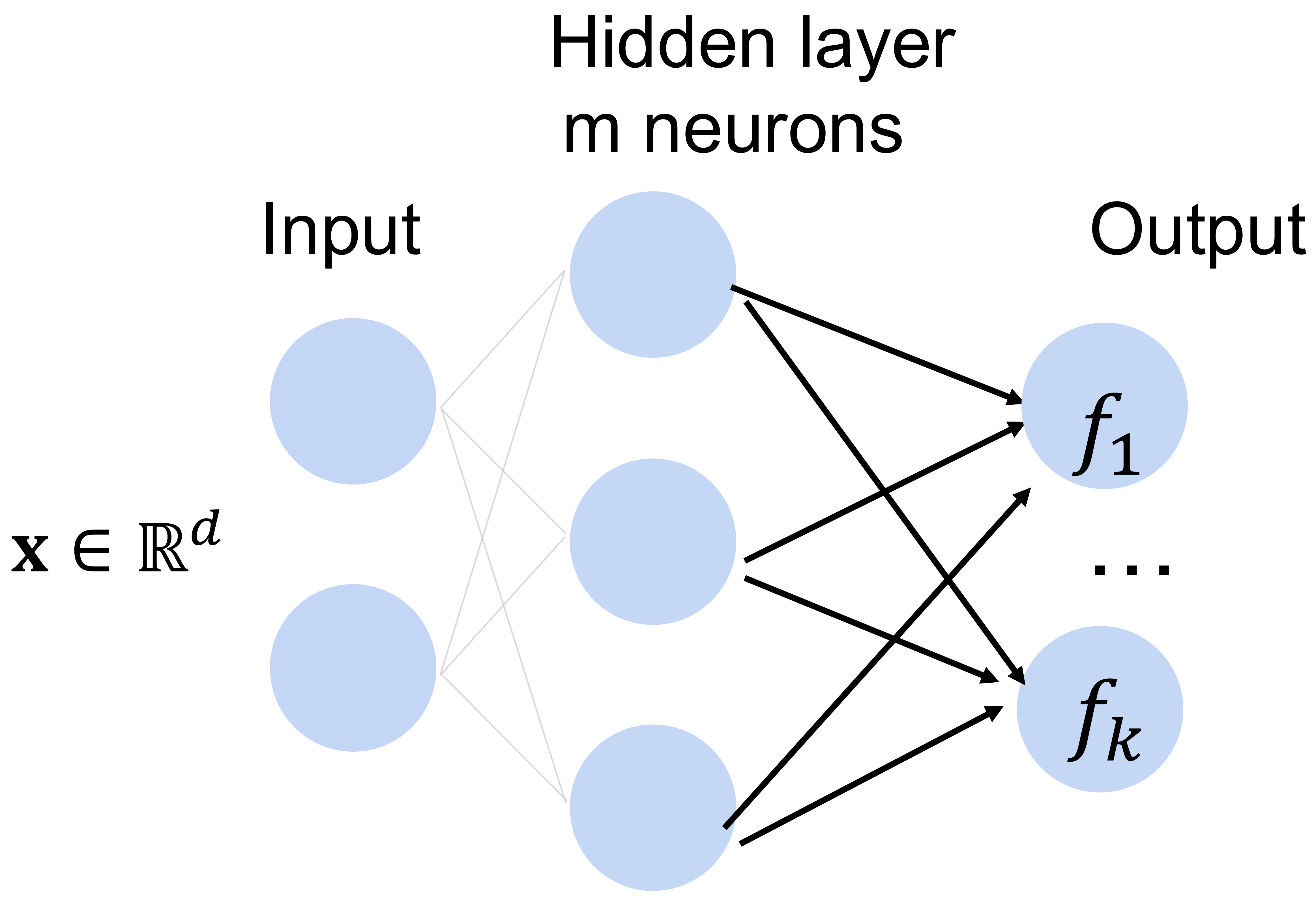
$$\mathbf{h} \quad = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

σ is an element-wise activation function



Multi-class classification

Turns outputs *f* into *k* probabilities (sum up to 1 across *k* classes)

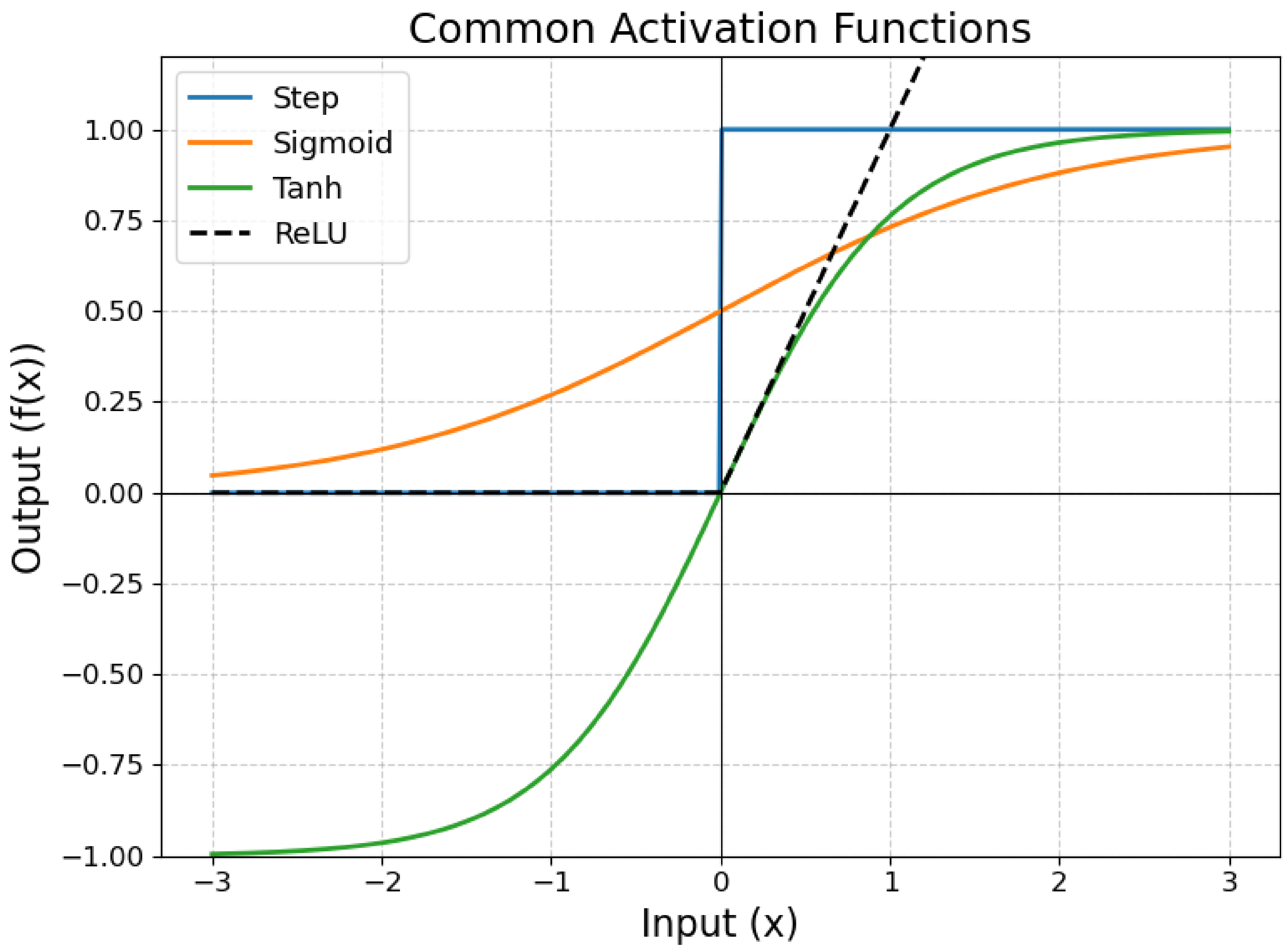


$$p(y|\mathbf{x}) = softmax(\mathbf{f}) = \frac{\exp f_y(x)}{\sum_i^k \exp f_i(x)}$$

Activation Functions

Used for neural network hidden nodes

- Step/Hard Threshold: $\sigma(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & o.w. \end{cases}$
- Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$
- tanh: $\sigma(z) = \tanh(\mathrm{z}) = \frac{e^{2z}-1}{e^{2z}+1}$
- ReLU: $\sigma(z) = \max\{0, z\}$



Loss Functions: Regression

- Squared Error: $\ell(y, \hat{y}) = (\hat{y} - y)^2$

- If our model predicts $\hat{y} = f_\theta(x)$, we write

$$\ell(\theta; x, y) = (f_\theta(x) - y)^2$$

- Over a dataset of n examples: MSE

$$L(\theta; X, y) = \frac{1}{n}\sum_{i=1}^{n}(f_\theta(x_i) - y_i)^2$$

Loss Functions: Classification

- Misclassification Error
  - Used for perceptron

$$\ell(y, \hat{y}) = \begin{cases} 0 & \text{if } y = \hat{y} \\ 1 & \text{if } y \neq \hat{y} \end{cases}$$

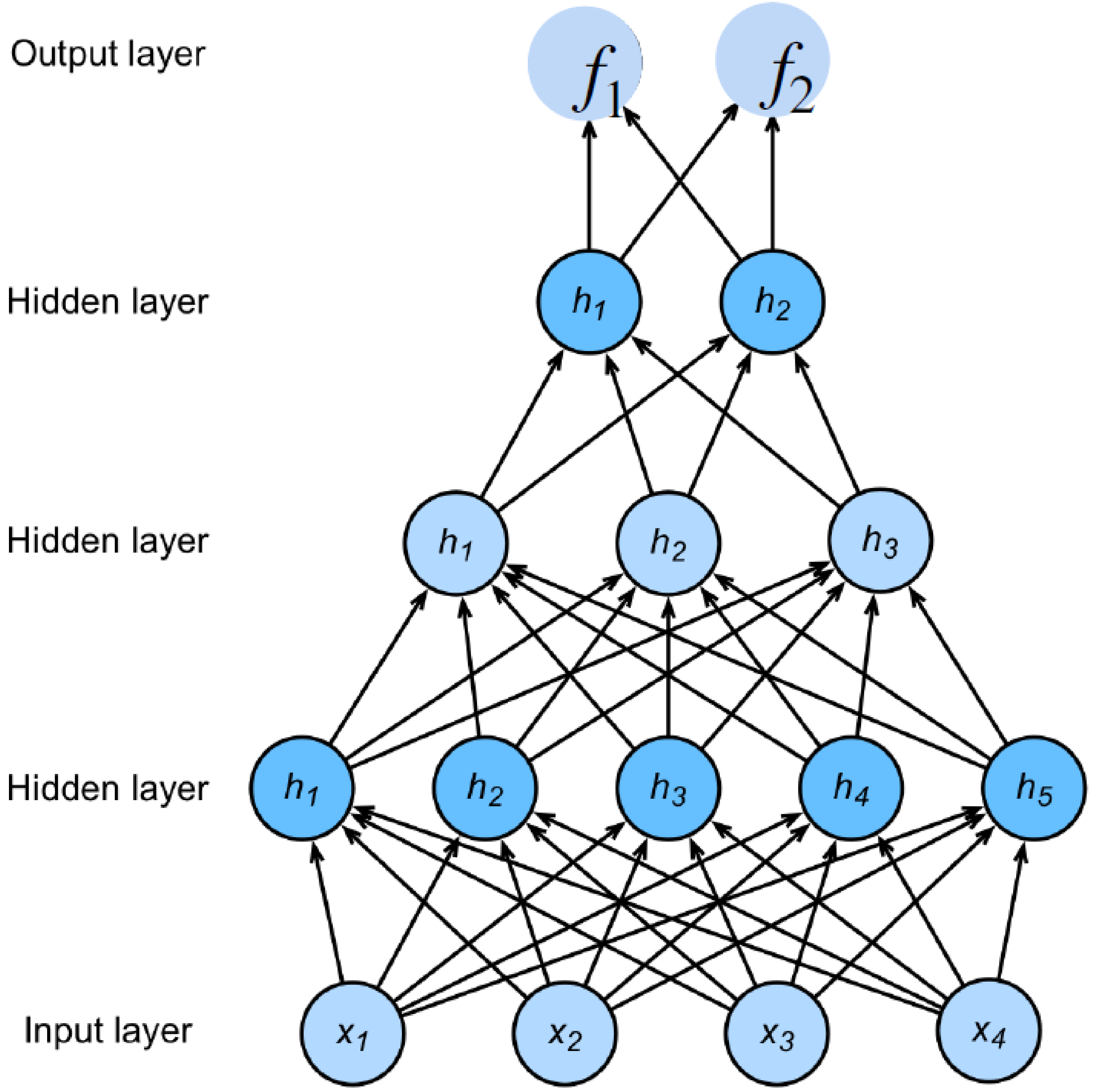
- Binary Cross-Entropy
  - $y \in \{0,1\}$, $\hat{y} \in [0,1]$

$$\ell(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

- Cross Entropy
  - For $k \geq 2$ classes
  - True label y is one-hot vector
  - Prediction $\hat{y}$ is a probability distribution over k classes (like the output of softmax)

$$\ell(y, \hat{y}) = -\sum_{i=1}^{k} y_i \log \hat{y}_i$$

Deep neural networks (DNNs)



$$\mathbf{h}_1 = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{h}_2 = \sigma(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$$

$$\mathbf{h}_3 = \sigma(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3)$$

$$\mathbf{f} = \mathbf{W}_4\mathbf{h}_3 + \mathbf{b}_4$$

$$\mathbf{y} = \mathrm{softmax}(\mathbf{f})$$

NNs are composition of nonlinear functions

Gradients and Gradient Descent

- When f maps vector $x \in \mathbb{R}^d$ to scalar $f(x) \in \mathbb{R}$

$$\nabla f(x) \in \mathbb{R}^d$$

- We will work with functions of many vectors (and matrices!)

$$L(\theta; X, y)$$

- Write gradient with respect to θ

$$\nabla_\theta L(\theta_{t-1}; X, y)$$

Gradient Descent
<u>Input:</u> dataset (X, y), loss function L, number of steps T, step size η
1. Initialize θ_0 2. For $t = 1,2, \ldots, T$ 3. Calculate $g_t = \nabla_\theta L(\theta_{t-1}; X, y)$ 4. Update $\theta_t \leftarrow \theta_{t-1} - \eta g_t$ 5. Return θ_T

(Minibatch) Stochastic Gradient Descent

- Gradient descent uses loss on entire dataset every step:

$$\nabla L(\theta; X, y) = \sum_{i=1}^{n} \nabla \ell(\theta; x_i, y_i)$$

- This is extremely inefficient!

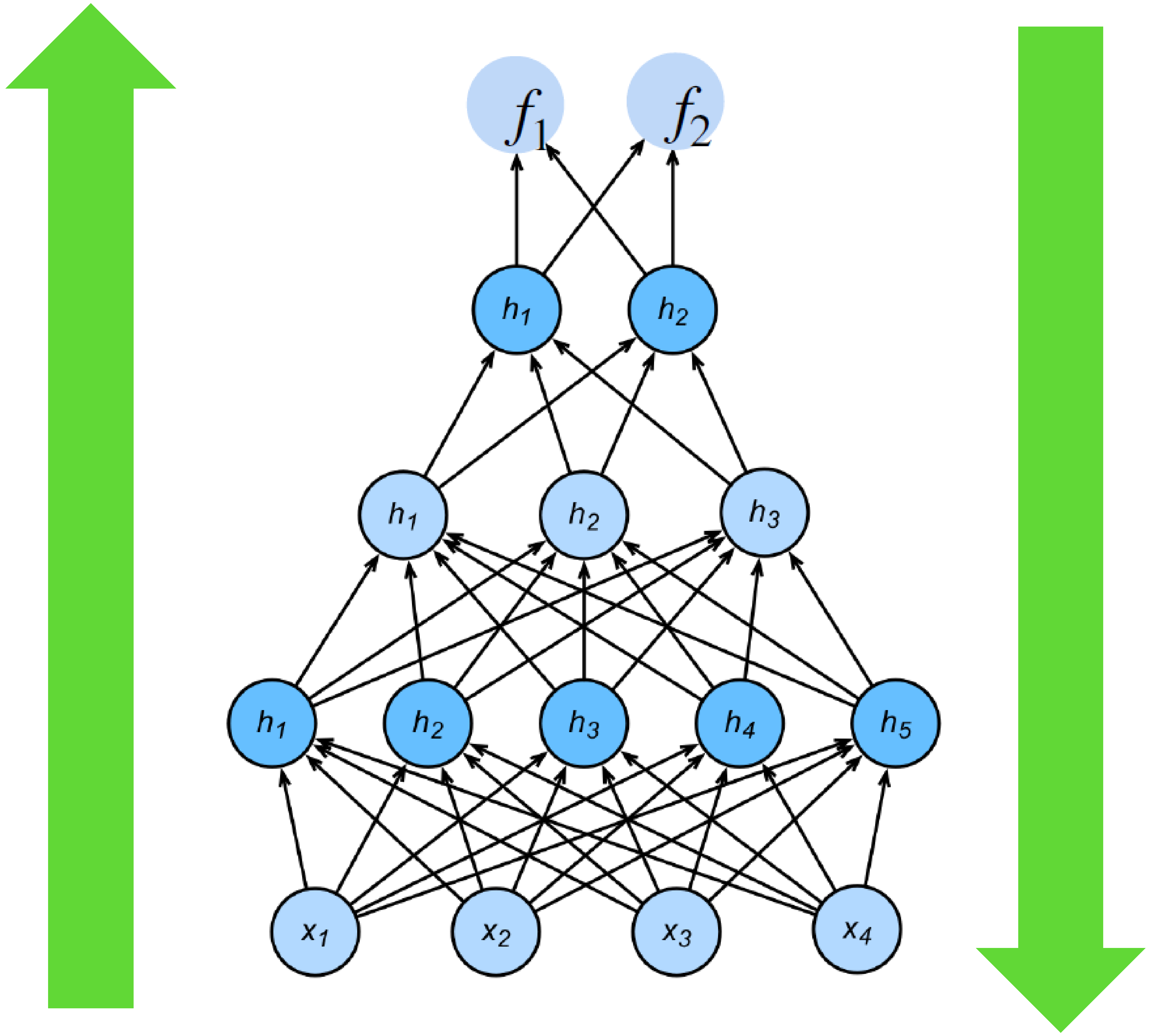
- On big datasets, better to update based on a few examples

Stochastic Gradient Descent
Input: dataset (X, y), loss function L, number of steps T, step size η, batch size m
1. Initialize θ_0 2. For $t = 1,2, \dots, T$ 3. Select random $(x_1, y_1), \dots, (x_m, y_m)$ 4. Calculate $g_t = \sum_{i=1}^{m} \nabla_\theta \ell(\theta_{t-1}; x_i, y_i)$ 5. Update $\theta_t \leftarrow \theta_{t-1} - \eta g_t$ 6. Return θ_T

Backpropagation: An Efficient Algorithm for Gradients in Neural Networks

Forward pass:

Start with input layer, compute all hidden nodes and outputs layer-by-layer



Backward pass:

Start with output layer, compute all partial derivatives layer-by-layer

The Backpropagation Algorithm: Layer-by-Layer, Backwards

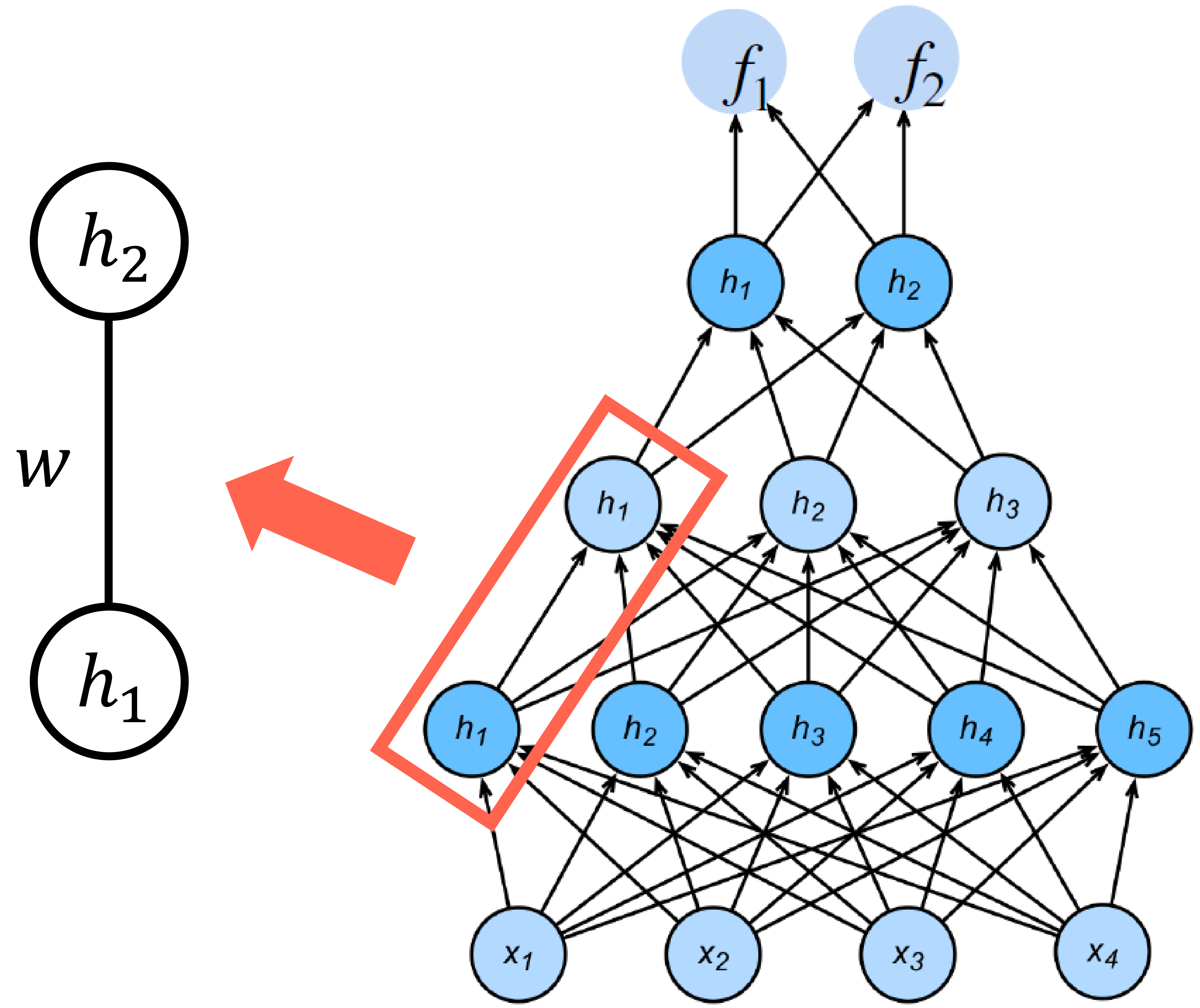
Want to find partial derivative for weight w in middle of network

Connects from h_1 to h_2

Chain rule: $\frac{\partial \ell}{\partial w} = \frac{\partial \ell}{\partial h_2} \times \frac{\partial h_2}{\partial w}$

We already computed this

This is simple to compute



How to classify

Cats vs. dogs?

Dual

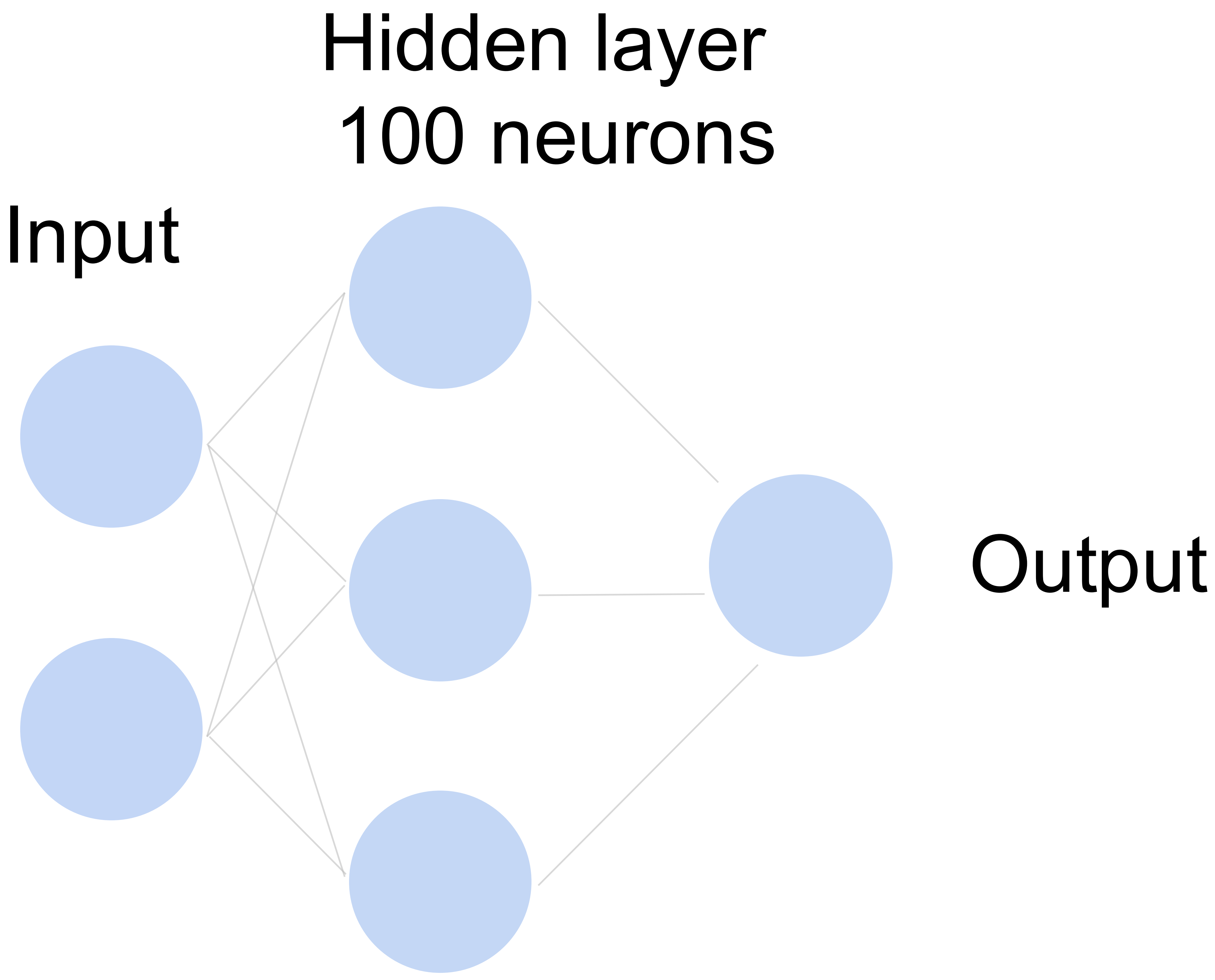
12MP

wide-angle and
telephoto cameras

36M floats in a RGB image!

Fully Connected Networks

Cats vs. dogs?



~ 36M elements x 100 = ~**3.6B** parameters!

Convolutions come to rescue!

Where is Waldo?

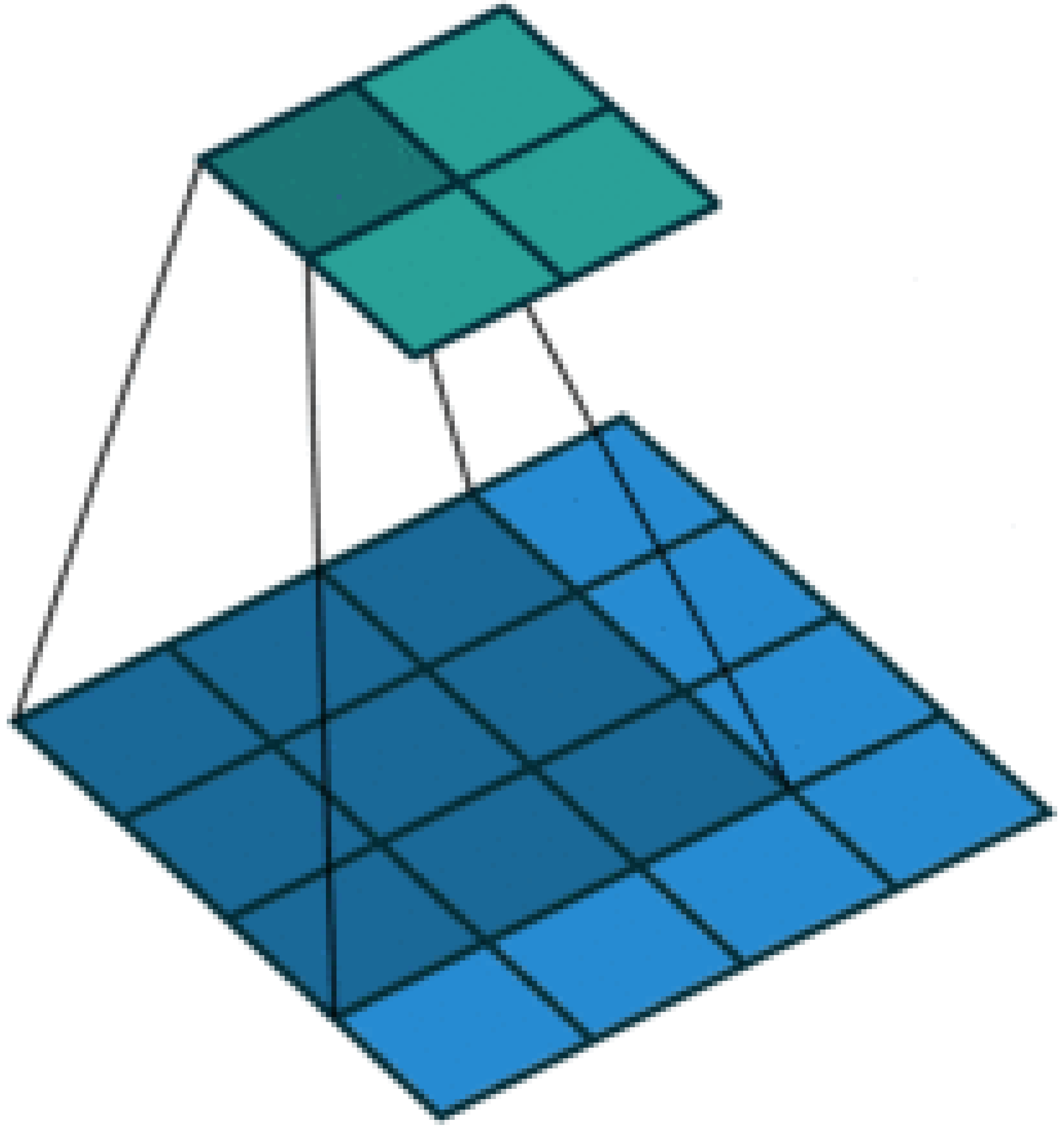
Why Convolution?

- Translation Invariance
- Locality

2-D Convolution

$$\underset{\text{Input}}{\begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 3 & 4 & 5 \\ \hline 6 & 7 & 8 \\ \hline \end{array}} * \underset{\text{Kernel}}{\begin{array}{|c|c|}\hline 0 & 1 \\ \hline 2 & 3 \\ \hline \end{array}} = \underset{\text{Output}}{\begin{array}{|c|c|}\hline 19 & 25 \\ \hline 37 & 43 \\ \hline \end{array}}$$

$$\begin{aligned} 0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 &= 19, \\ 1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 &= 25, \\ 3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 &= 37, \\ 4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 &= 43. \end{aligned}$$



(vdumoulin@ Github)

2-D Convolution Layer

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 19 & 25 \\ 37 & 43 \end{bmatrix}$$

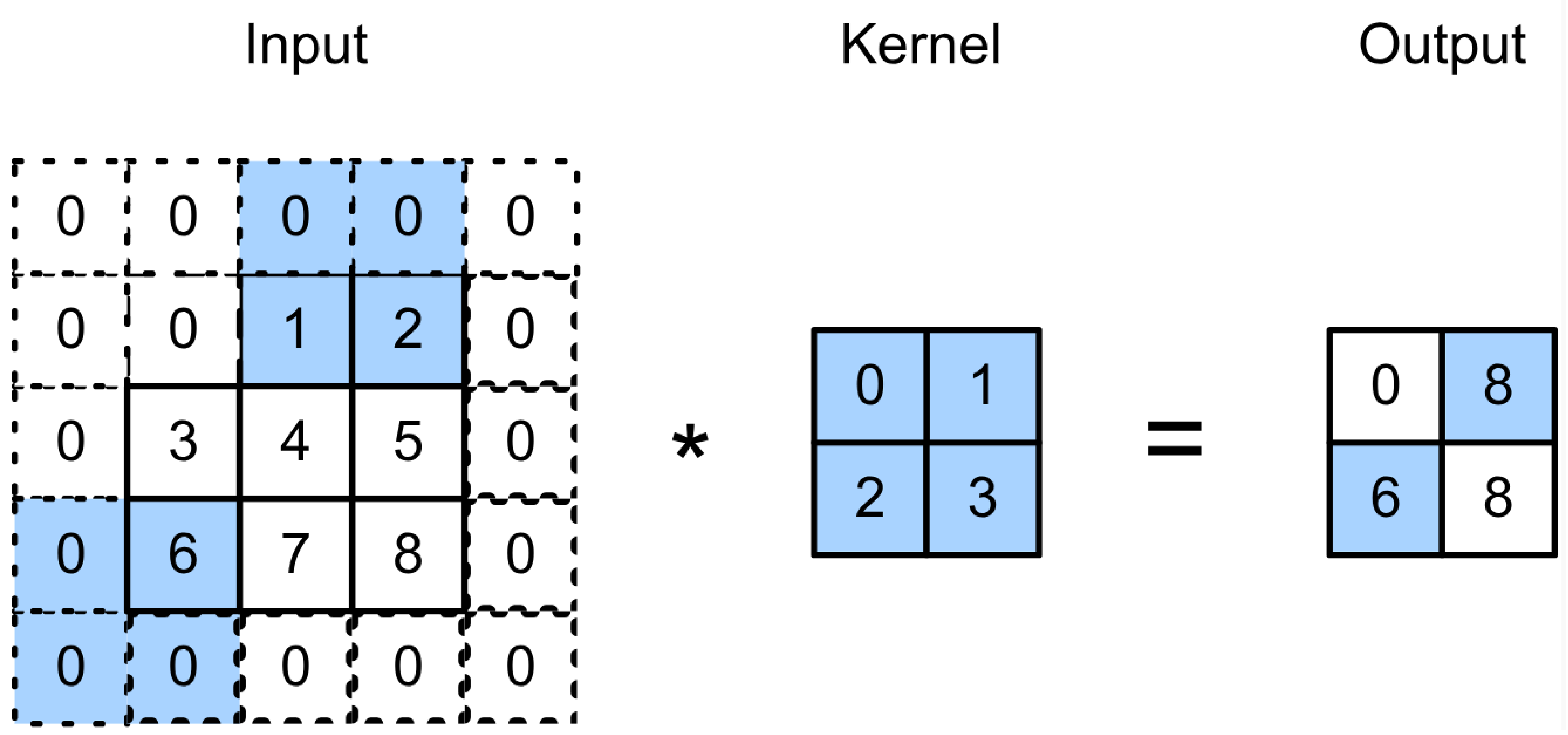
- $\mathbf{X}$: $n_h \times n_w$ input matrix
- $\mathbf{W}$: $k_h \times k_w$ kernel matrix
- b: scalar bias
- $\mathbf{Y}$: $(n_h - k_h + 1) \times (n_w - k_w + 1)$ output matrix

$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} + b$$

- **W** and *b* are learnable parameters

2-D Convolution Layer with Stride and Padding

- Stride is the #rows/#columns per slide
- Padding adds rows/columns around input
- Output shape



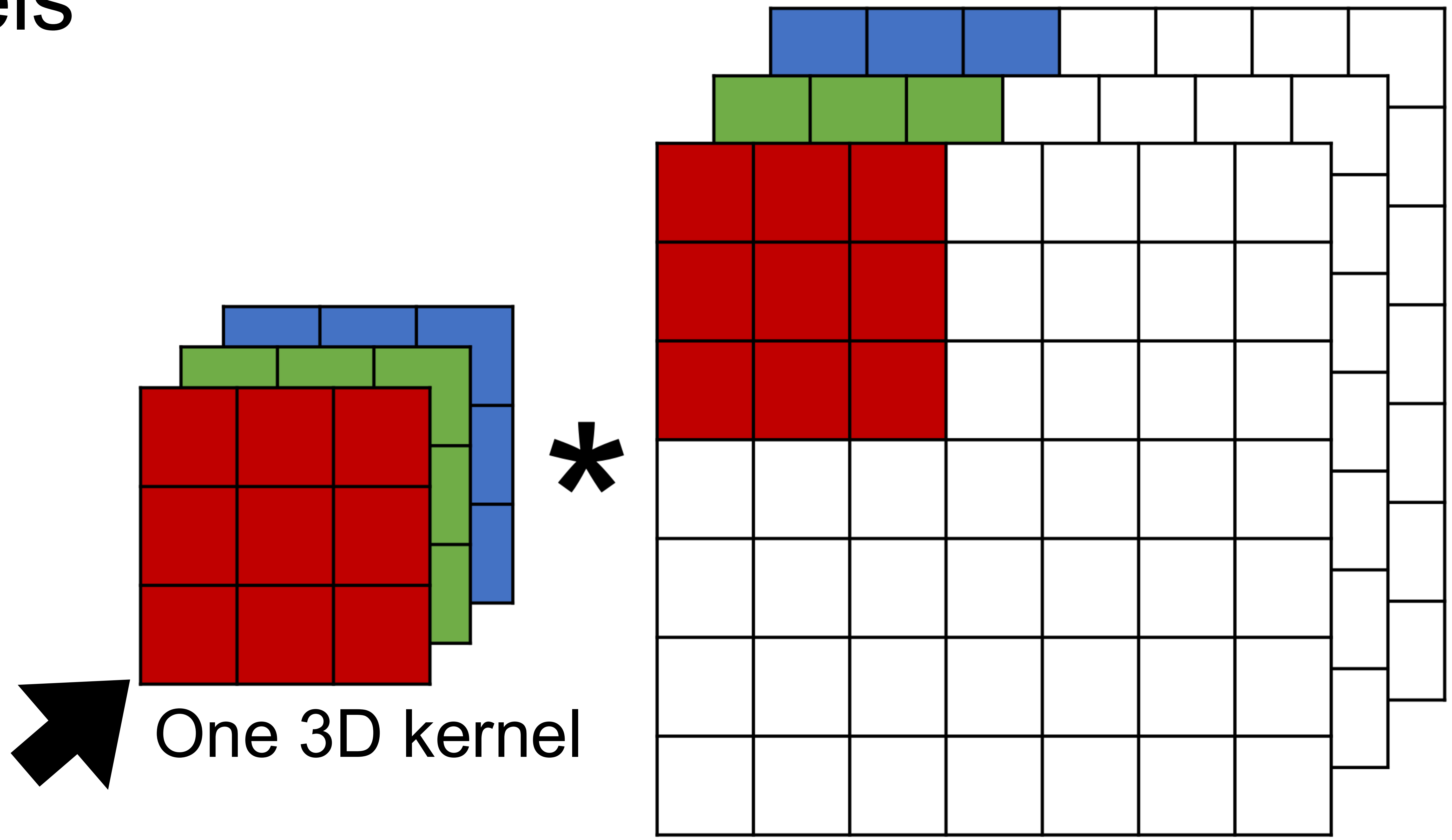
Kernel/filter size

$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$$

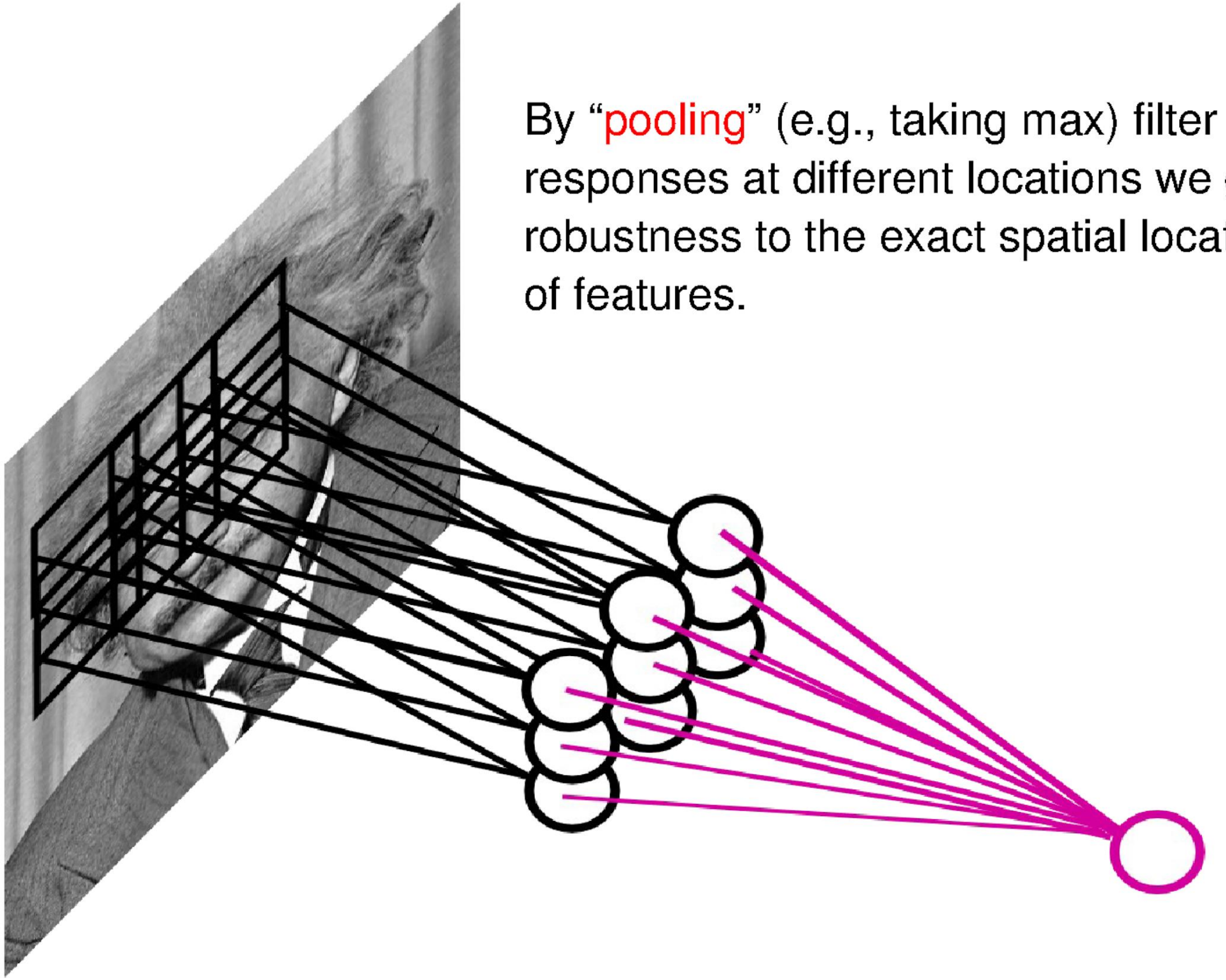
Input size **Pad** **Stride**

Multiple Input Channels

- Input and kernel can be 3D, e.g., an RGB image have 3 channels
- Have a 2D kernel for each channel, and then sum results over channels



Pooling

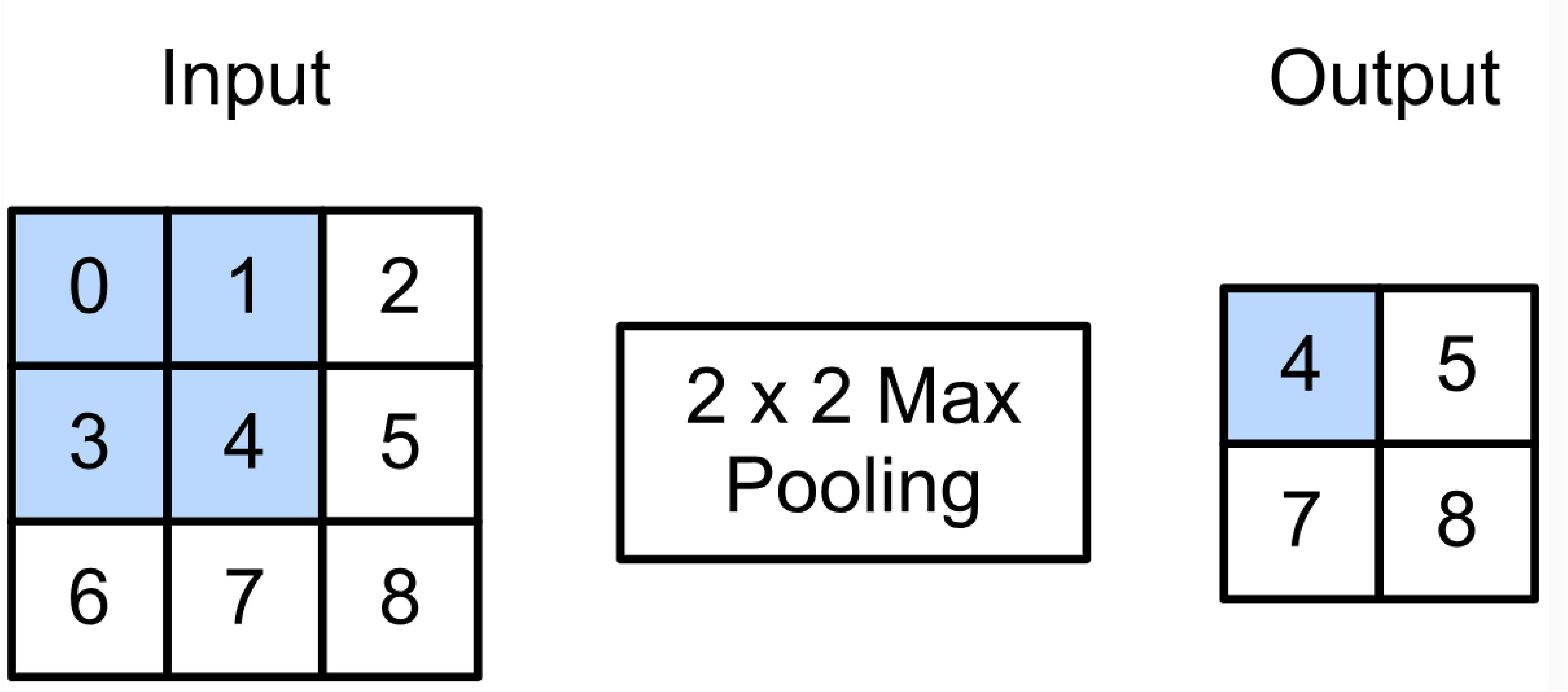


By "pooling" (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

Slides Credit: Deep Learning Tutorial by Marc'Aurelio Ranzato

2-D Max Pooling

- Returns the maximal value in the sliding window

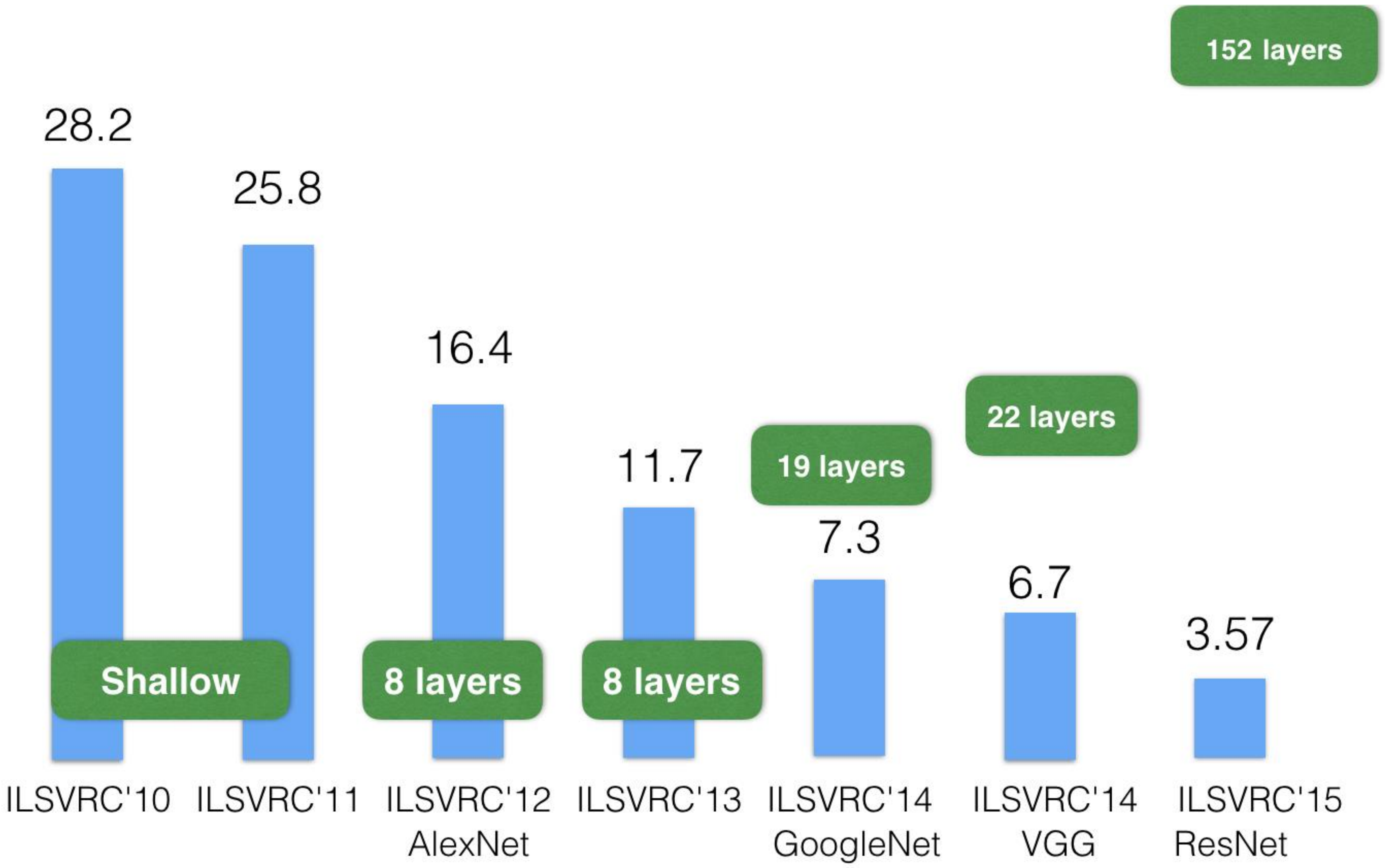


max(0,1,3,4) = 4

LeNet Architecture

gluon-cv.mxnet.io

ResNet: Going deeper in depth



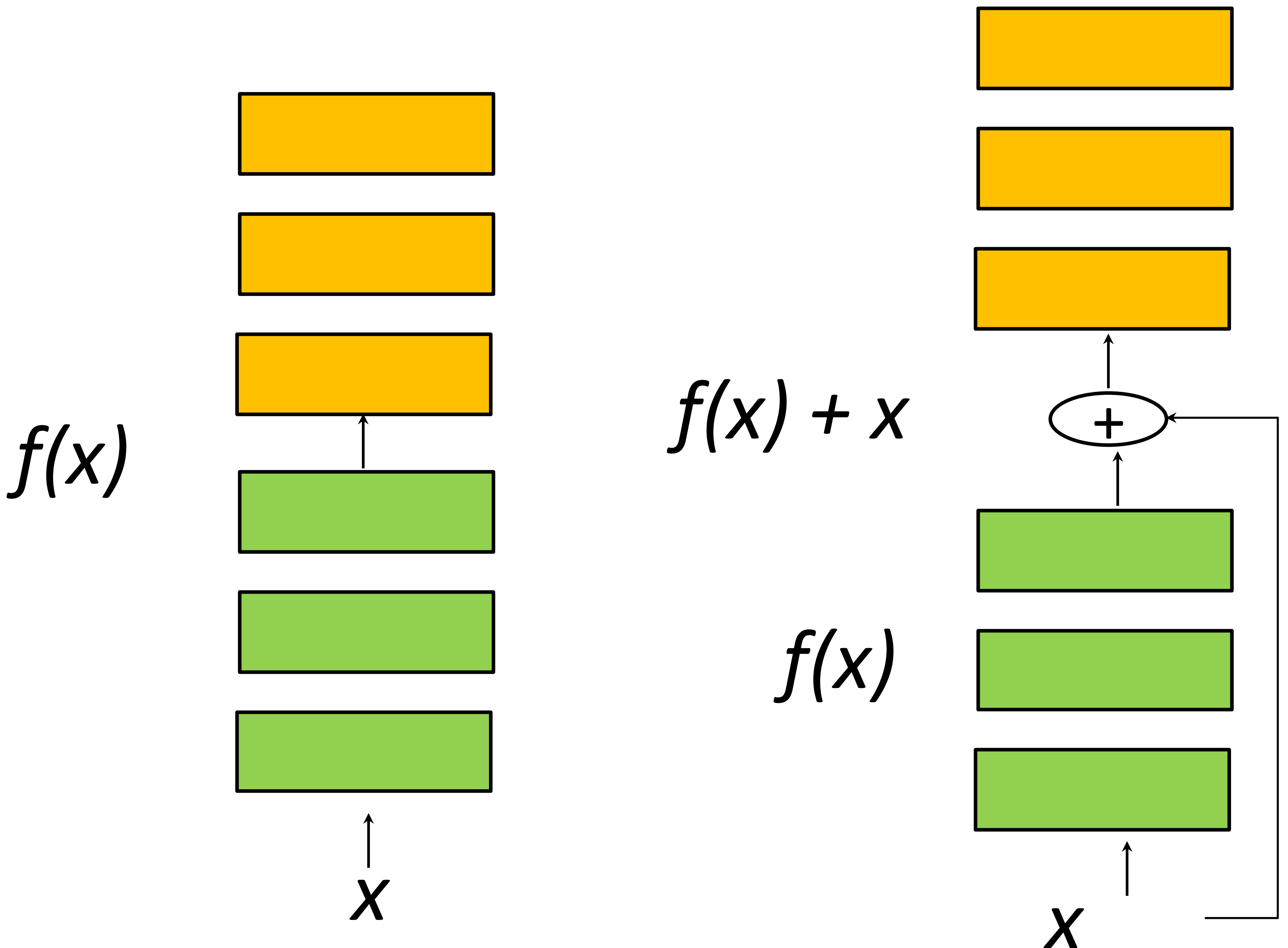
ImageNet Top-5 error%

[He et al. 2015]

Residual Connections

Idea: Identity might be hard to learn, but zero is easy!

- Make all the weights tiny, produces zero for output
- Can easily transform learning identity to learning zero:



Left: Conventional layers block

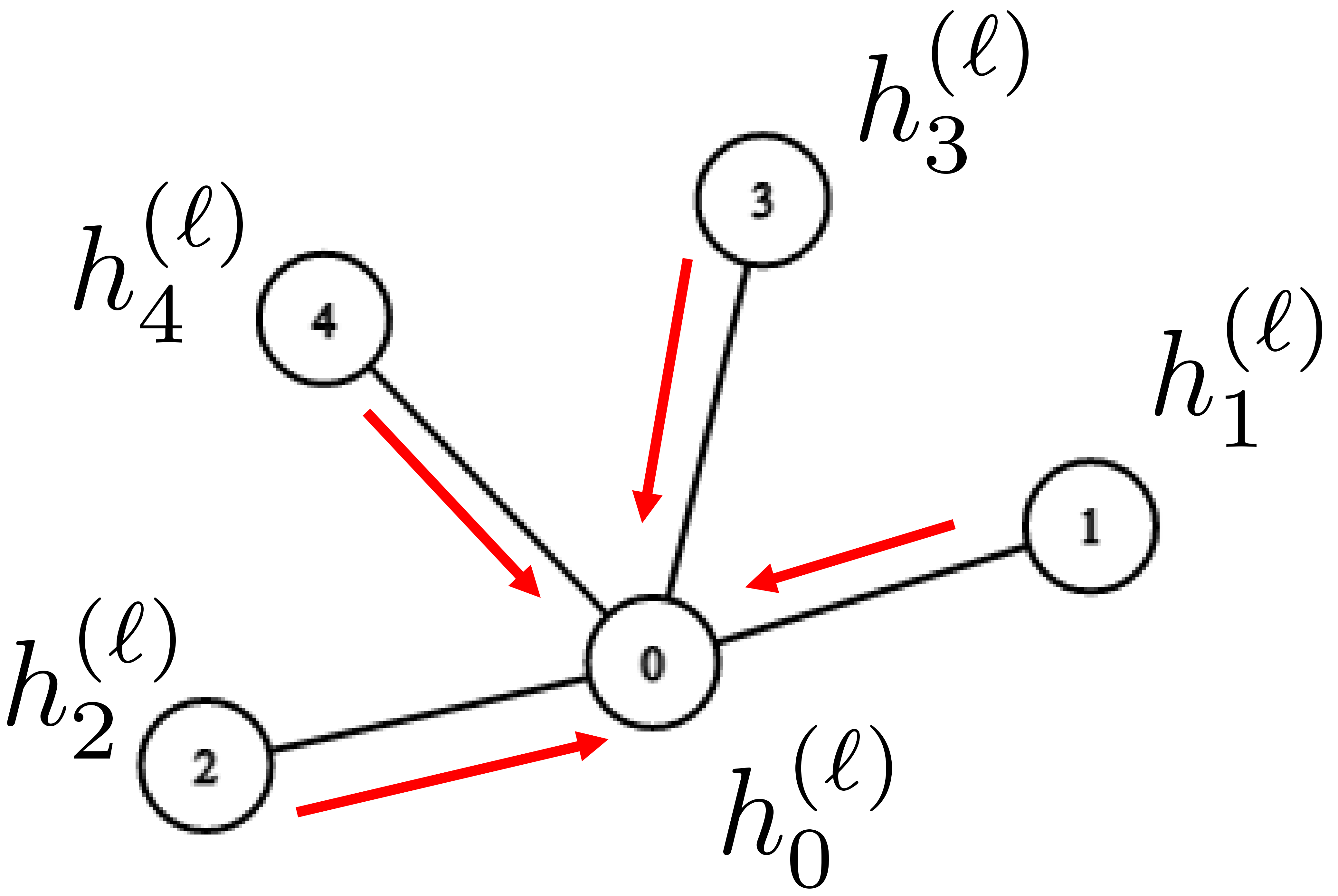
Right: **Residual** layer block

To learn identity $f(x) = x$, layers now need to learn $f(x) = 0$ ➔ easier

Graph Neural Networks

Model connections between data (e.g. social networks)

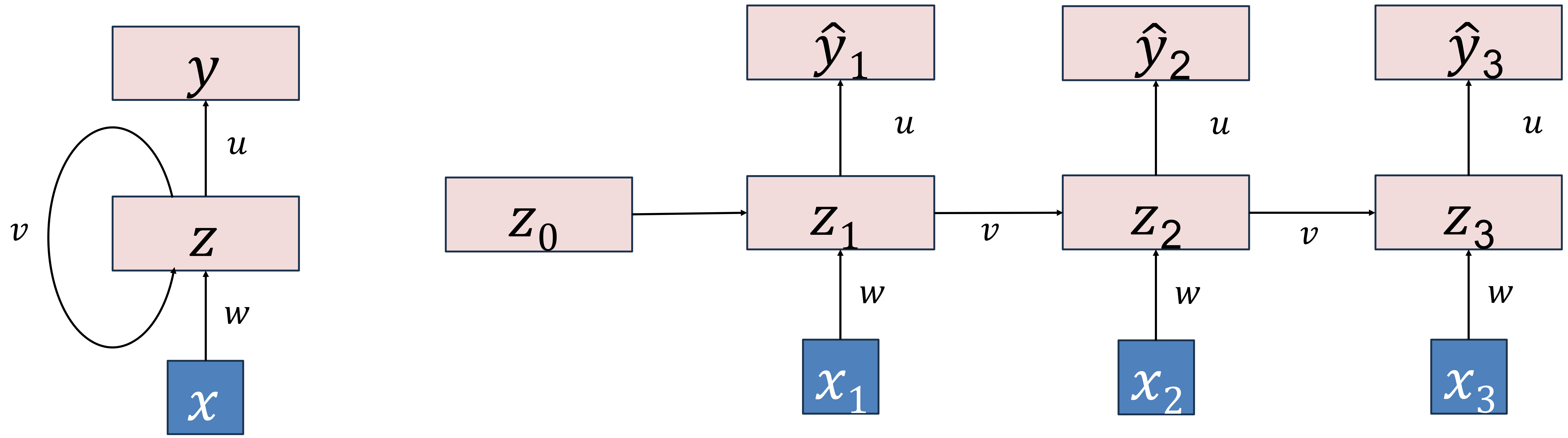
- Difference: “graph mixing” component
- At each layer, get representation at each node
- Combine node’s representation with neighboring nodes



- “**Aggregate**” and “**Update**” rules

Recurrent Neural Networks (RNNs)

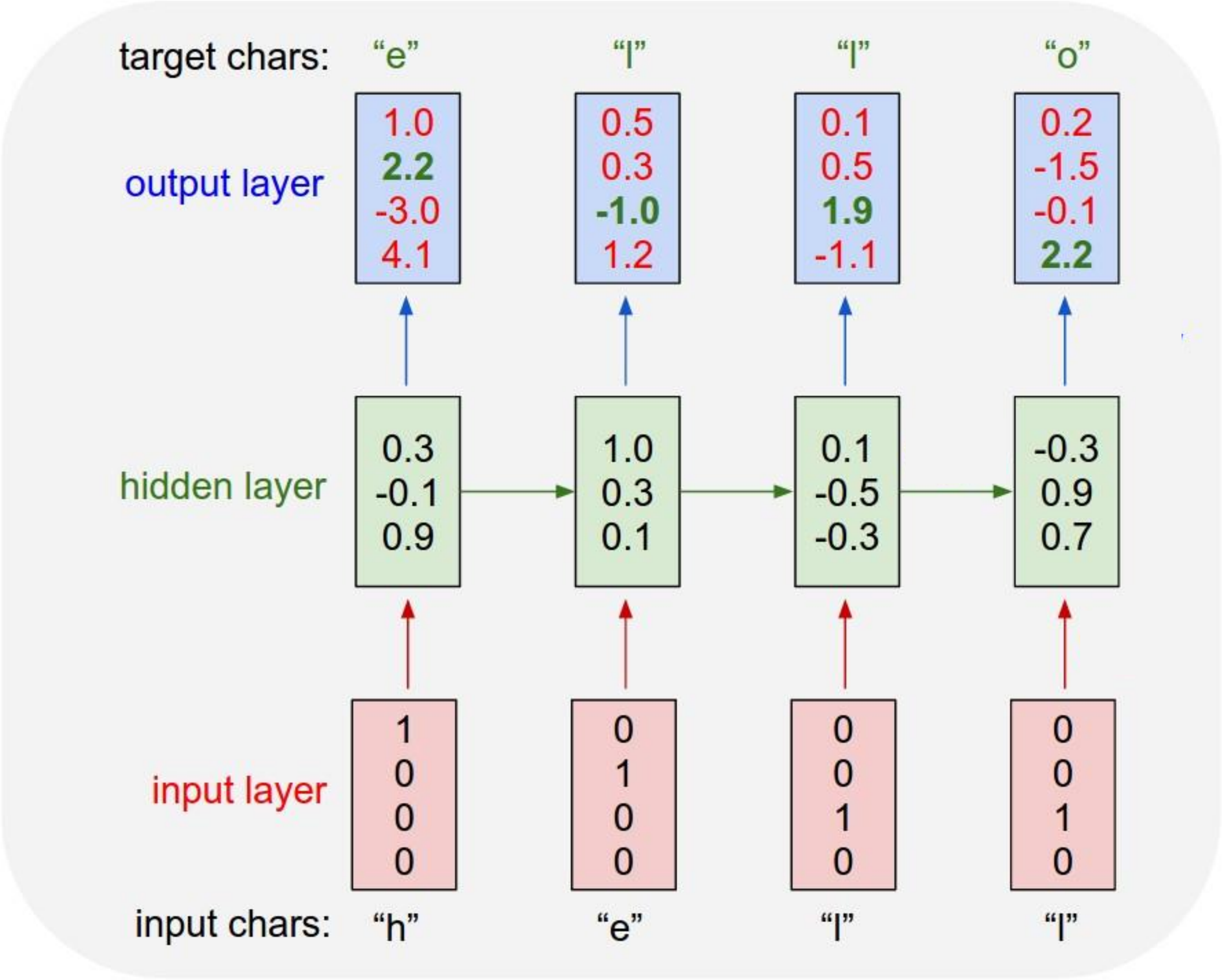
- RNNs introduce **cycles** in the computational graph
- Allowing information to persist; **memory**



RNNs for language modeling

- Simple example
  - 4 tokens: “h”, “e”, “l”, “o”
  - Hidden state has 3 dimensions

- Training: try to make output match targets
- Generation: sample!
  - (Same as with n-gram)



https://karpathy.github.io/2015/05/21/rnn-effectiveness/

Thank you!

Some of the slides in these lectures have been adapted from materials developed by Alex Smola and Mu Li:

https://courses.d2l.ai/berkeley-stat-157/index.html