Lecture 7

Neural Network

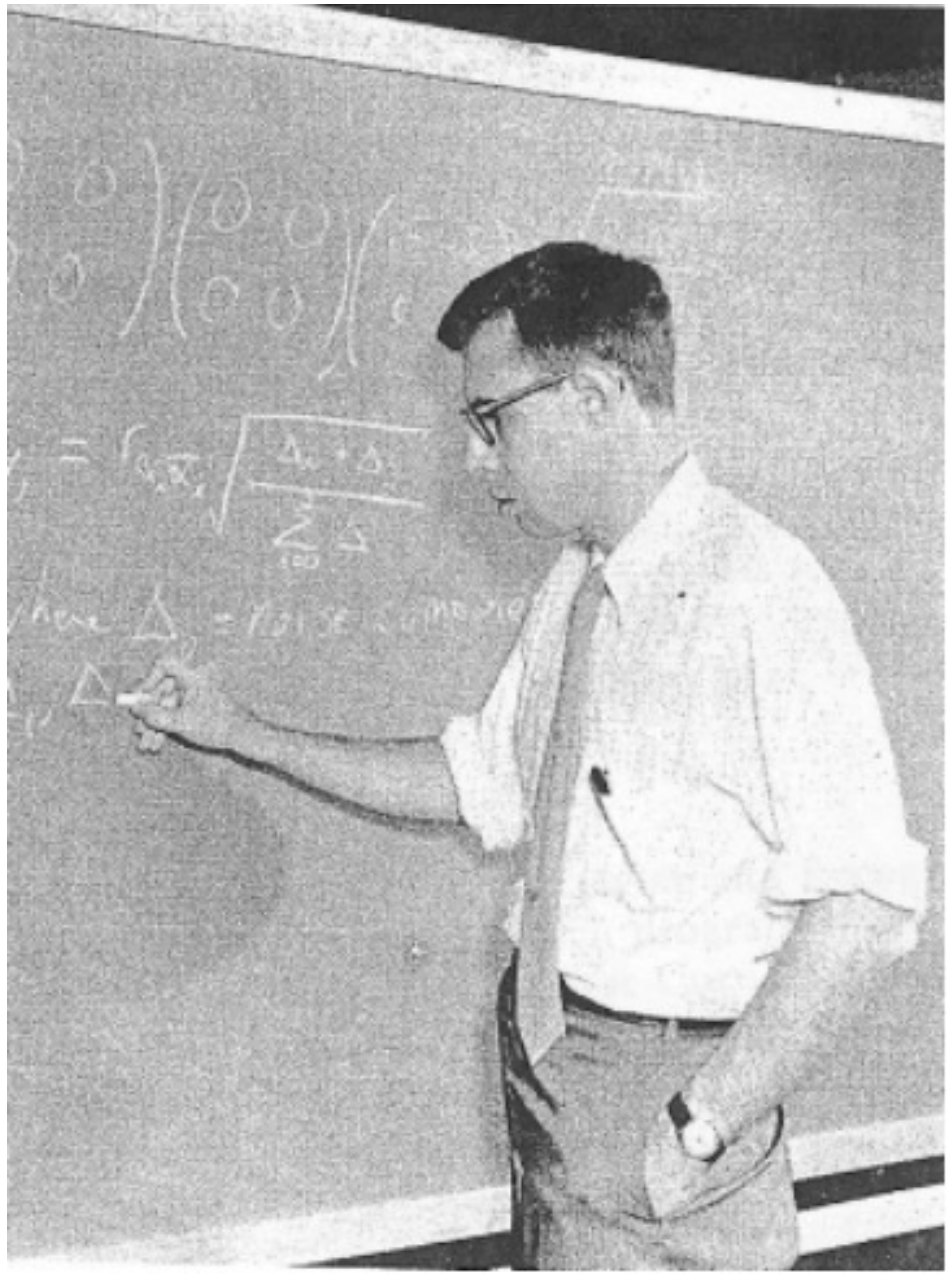
Neural network history

First stage

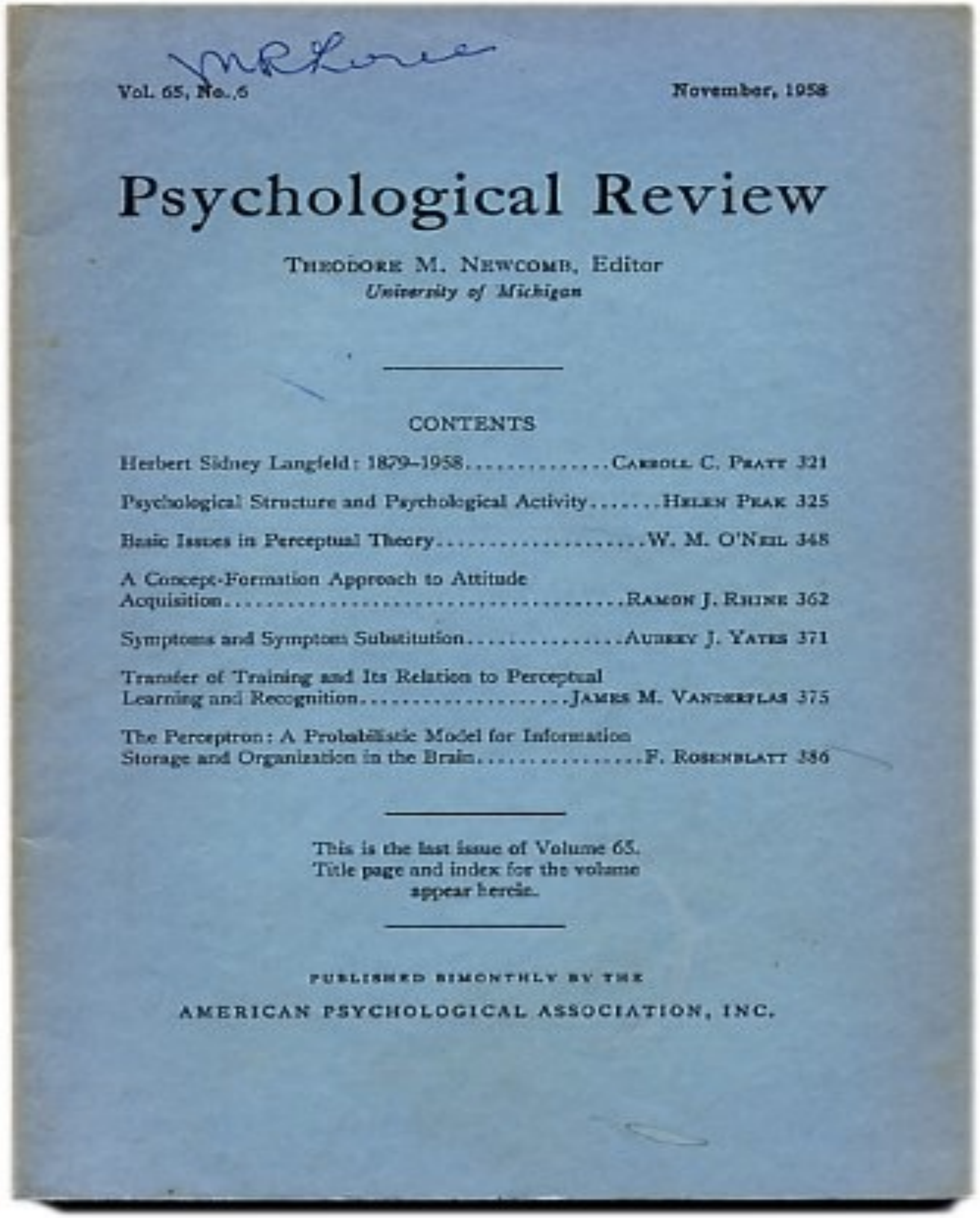
- In 1943, McCulloch and Pitts proposed the first neural model, i.e., M-P neuron model, and proved in principle that the artificial neural network can calculate any arithmetic and logical function.
- In 1958, Rosenblatt proposed Perceptron and its learning rule
- In 1960, Widrow and Hoff proposed Adaline and the Least Mean Square (LMS) algorithm
- In 1969, Minsky and Papert published the book《Perceptrons》, which pointed out that single-layer neural network cannot solve non-linear problems, and it is unknown whether it is possible to train multiple-layer networks. This conclusion directly pushed neural network research into an "ice age"

Neural network history

First stage - Perceptrons, 1958



Rosenblatt



Vol. 65, No. 6 November, 1958

Psychological Review

Theodore M. Newcomb, Editor
University of Michigan

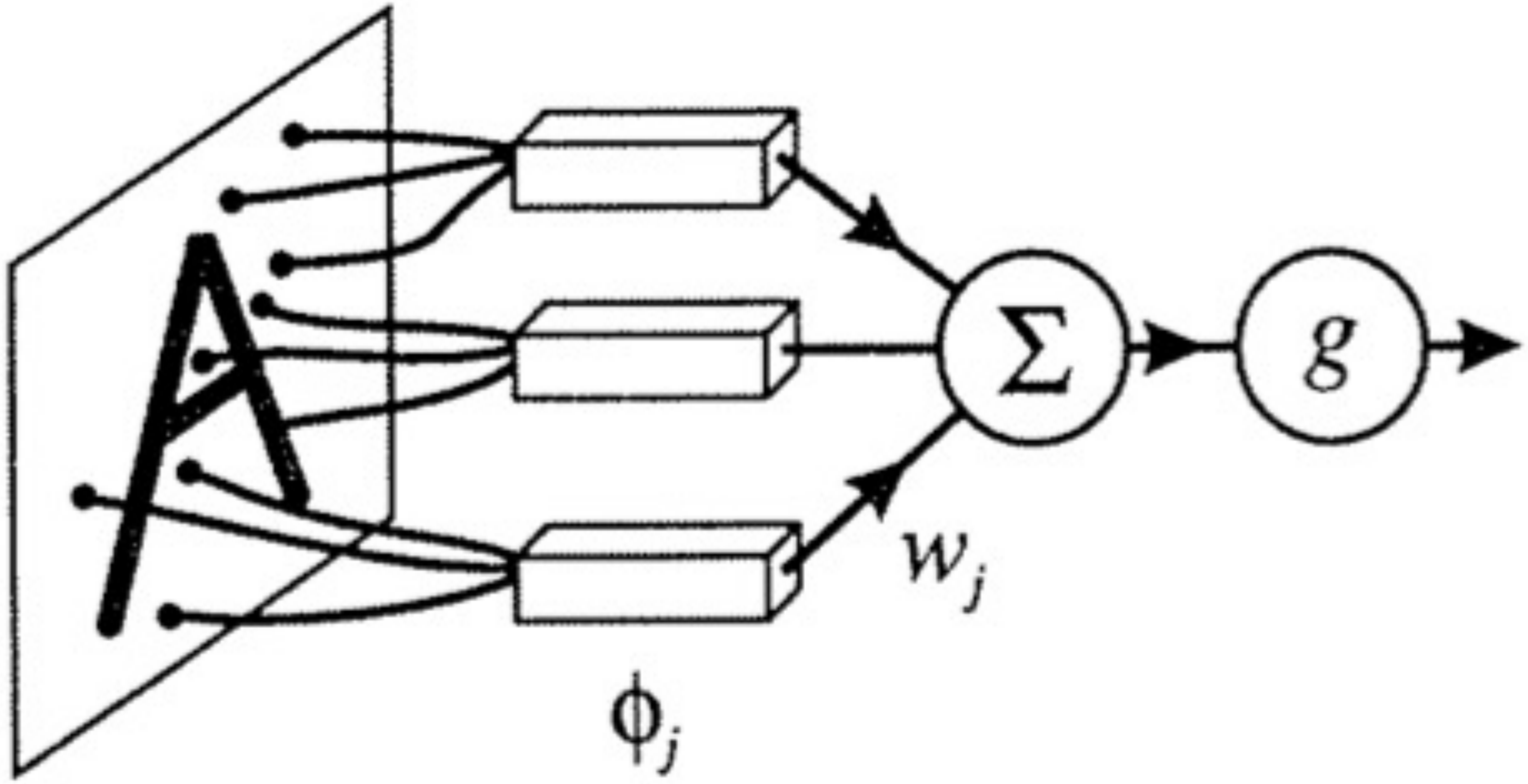
CONTENTS

This is the last issue of Volume 65.
Title page and index for the volume appear herein.

Published bimonthly by the
American Psychological Association, Inc.

Neural network history

First stage - Perceptrons, 1958



Neural network history

First stage - Minsky and Papert, Perceptrons, 1969

Perceptrons: an introduction to computational geometry is a book written by Marvin Minsky and Seymour Papert and published in 1969. An edition with handwritten corrections and additions was released in the early 1970s. An expanded edition was further published in 1988 after the revival of neural networks, containing a chapter dedicated to counter the criticisms made of it in the 1980s.

Perceptrons: an introduction to computational geometry

Author	Marvin Minsky, Seymour Papert
Publication date	1969
ISBN	0 262 13043 2

The main subject of the book is the perceptron, a type of artificial neural network developed in the late 1950s and early 1960s. The book was dedicated to psychologist Frank Rosenblatt, who in 1957 had published the first model of a "Perceptron".[1] Rosenblatt and Minsky knew each other since adolescence, having studied with a one-year difference at the Bronx High School of Science.[2] They became at one point central figures of a debate inside the AI research community, and are known to have promoted loud discussions in conferences, yet remained friendly.[3]

This book is the center of a long-standing controversy in the study of artificial intelligence. It is claimed that pessimistic predictions made by the authors were responsible for a change in the direction of research in AI, concentrating efforts on so-called "symbolic" systems, a line of research that petered out and contributed to the so-called AI winter of the 1980s, when AI's promise was not realized.[4]

The crux of *Perceptrons* is a number of mathematical proofs which acknowledge some of the perceptrons' strengths while also showing major limitations.[3] The most important one is related to the computation of some predicates, such as the XOR function, and also the important connectedness predicate. The problem of connectedness is illustrated at the awkwardly colored cover of the book, intended to show how humans themselves have difficulties in computing this predicate.[5] One reviewer, Earl Hunt, noted that the XOR function is difficult for humans to acquire as well during concept learning experiments.[6]

Neural network history

Second stage

- In 1982, the physicist Hopfield proposed a recursive network with associative memory and optimized computing power, i.e., the Hopfield network
- In 1986, Rumelhart et.al. published the PDP book《Parallel Distributed Processing: Explorations in the Microstructures of Cognition》, in which the BP algorithm was reinvented
- In 1987, IEEE holds the first international conference on neural networks in San Diego, Carlifornia（ICNN）
- In the early 1990s, statistical learning theory and SVM have emerged, while neural networks were suffering from the lacking of theories, heavily relying on trial-and-error and full of tricks. Neural networks enter another winter.

Neural network history

LeCun conv nets, 1998

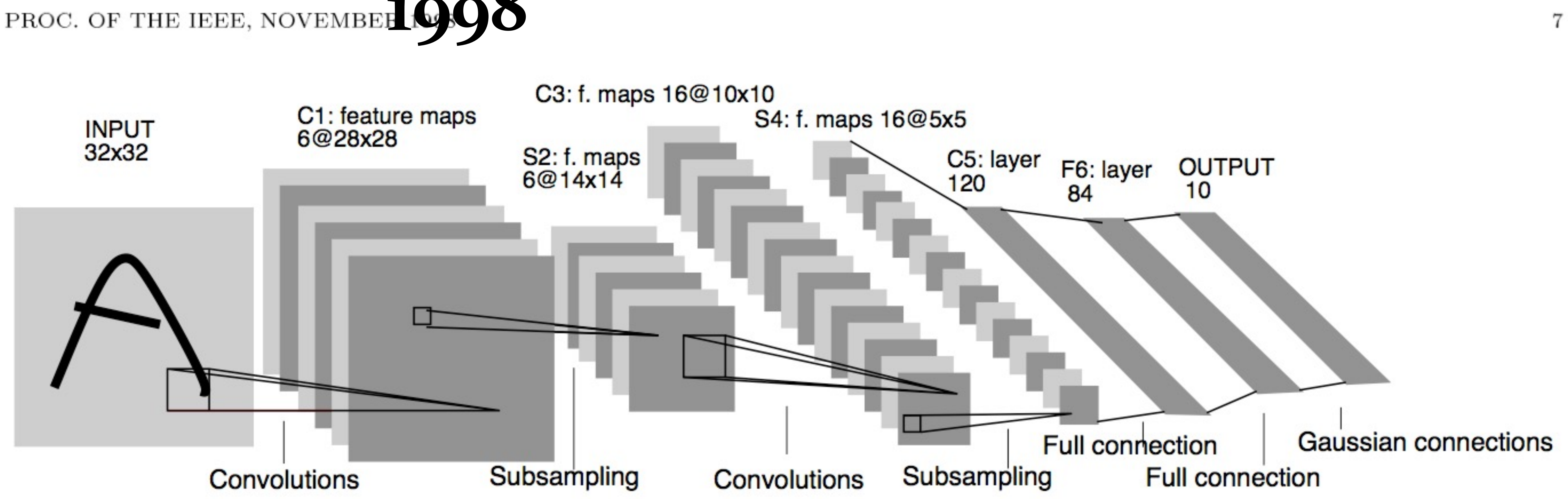


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Demos:

http://yann.lecun.com/exdb/lenet/index.html

Neural network history

Neural Information Processing Systems 2000

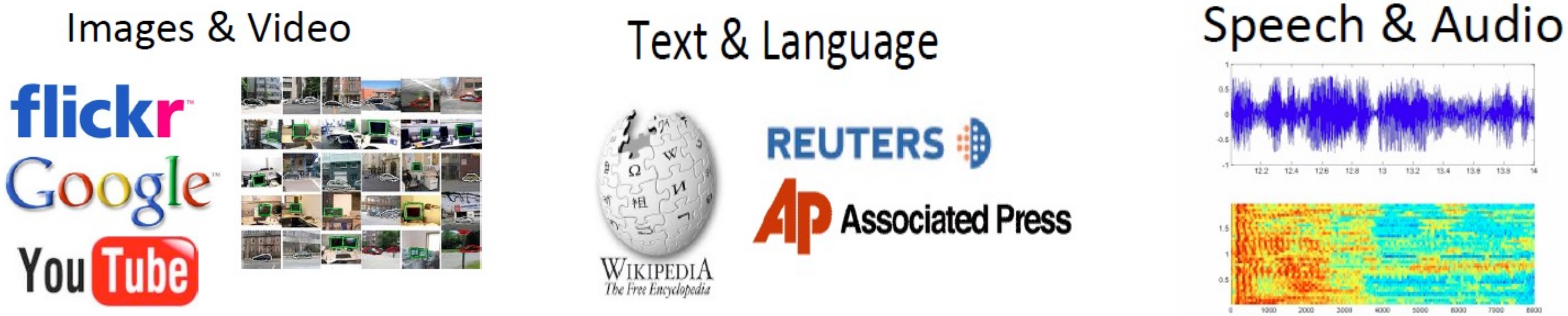
Neural Information Processing Systems, is the premier conference on machine learning. Evolved from an interdisciplinary conference to a machine learning conference.

For the 2000 conference:

Neural network history

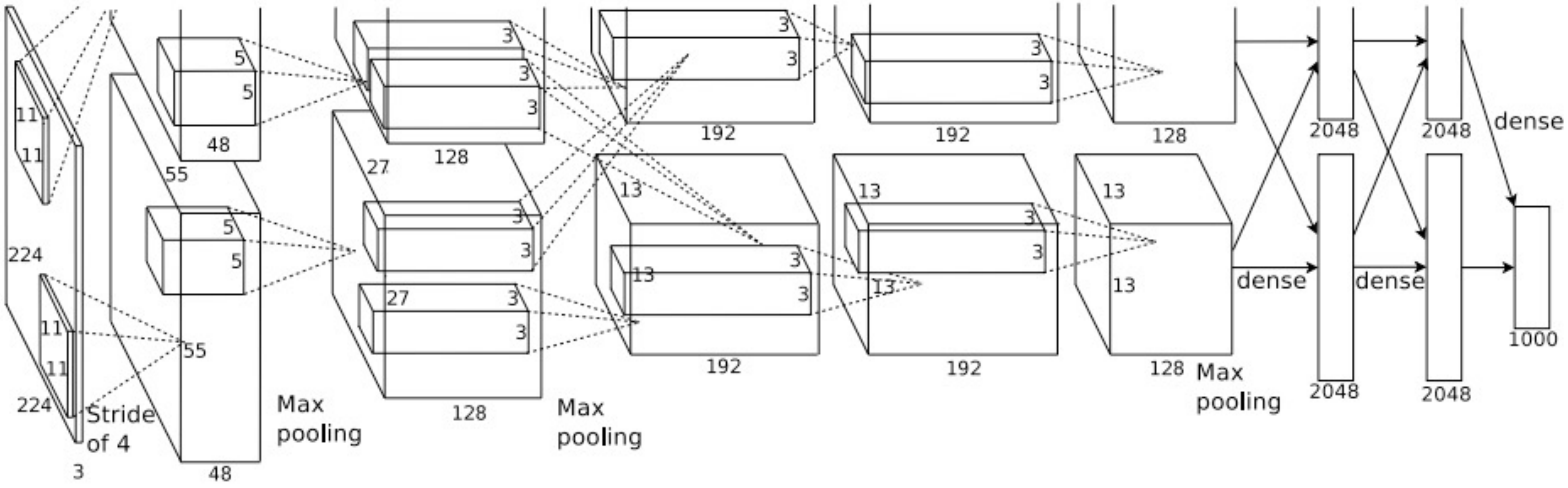
Third stage

- In 2006, Hinton proposed the Deep Belief Network (DBN), which makes the optimization of deep models relative easy through “pre-training+fine-tuning”
- In 2012, the Hinton team participated in the ImageNet competition and won the championship of the year with a score of 10% over the second place using the CNN model
- With the advent of the cloud computing and big data era, computing power has greatly improved, making deep learning models have achieved great success in various fields



Neural network history

Third stage - AlexNet
Krizhevsky, Sutskever, and Hinton, NeurIPS 2012



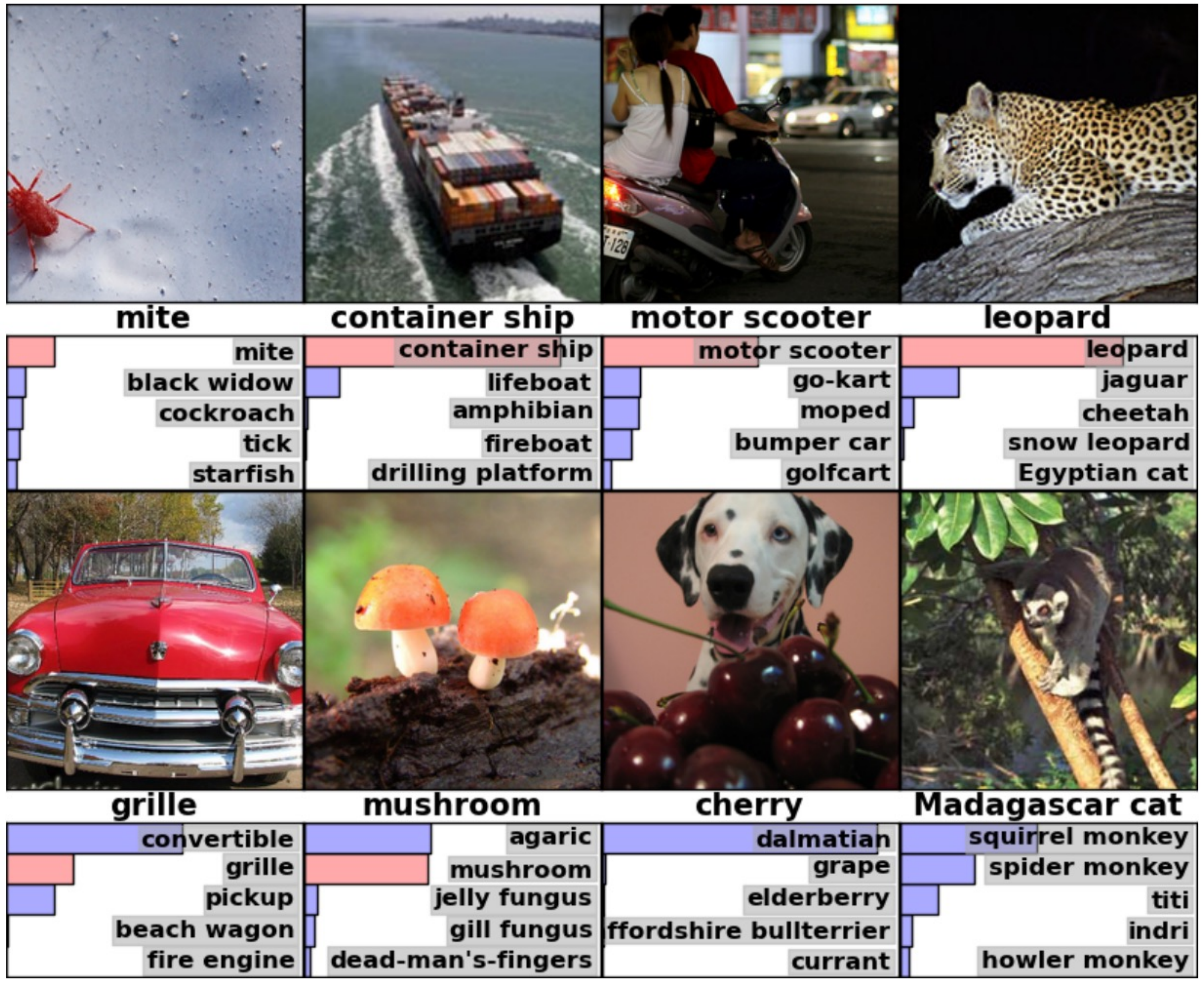
AlexNet consists of **5 Convolutional Layers** and **3 Fully Connected Layers**.

Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. NIPS 2012. Citations till April 7, 2024: **127789**

Neural network history

Third stage - AlexNet

Krizhevsky, Sutskever, and Hinton, NeurIPS 2012



ResNet

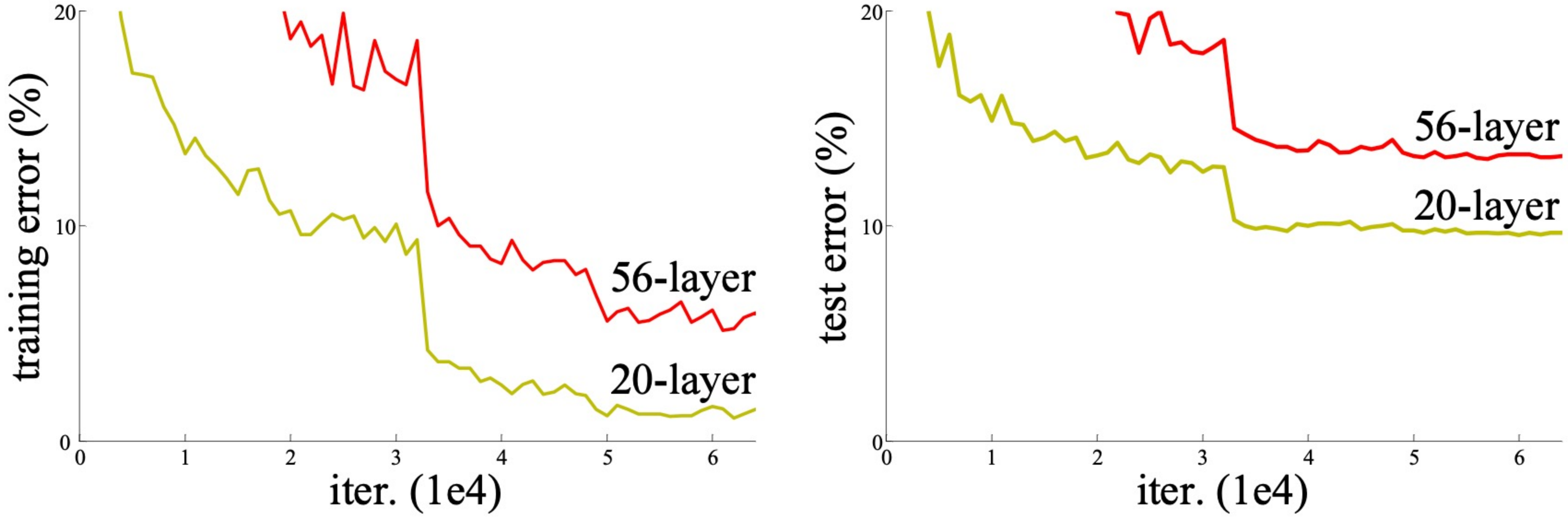


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Kaiming He et. al. Deep Residual Learning for Image Recognition. CVPR 2015. Citations till April 7, 2024: **212268**

ResNet

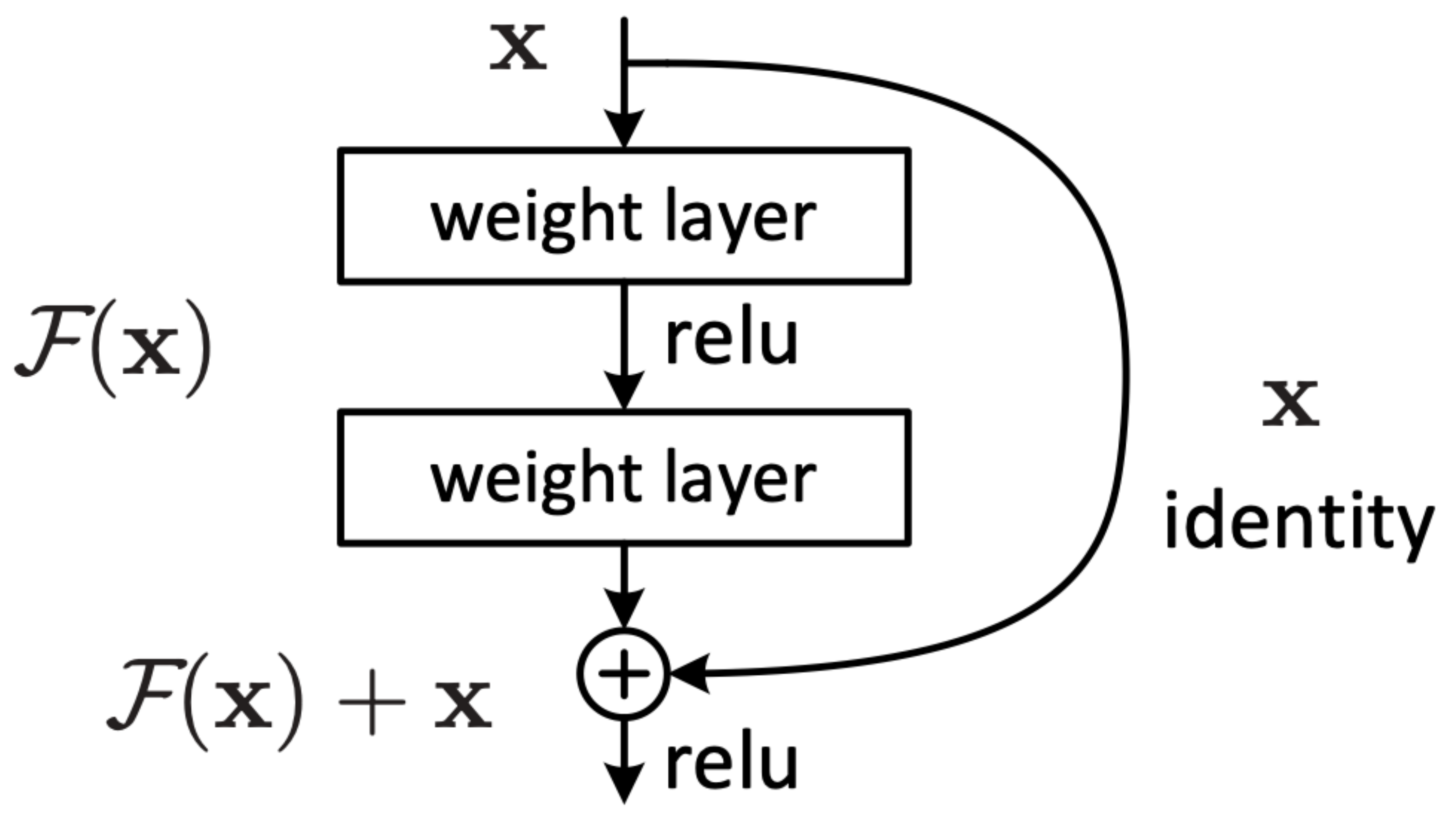
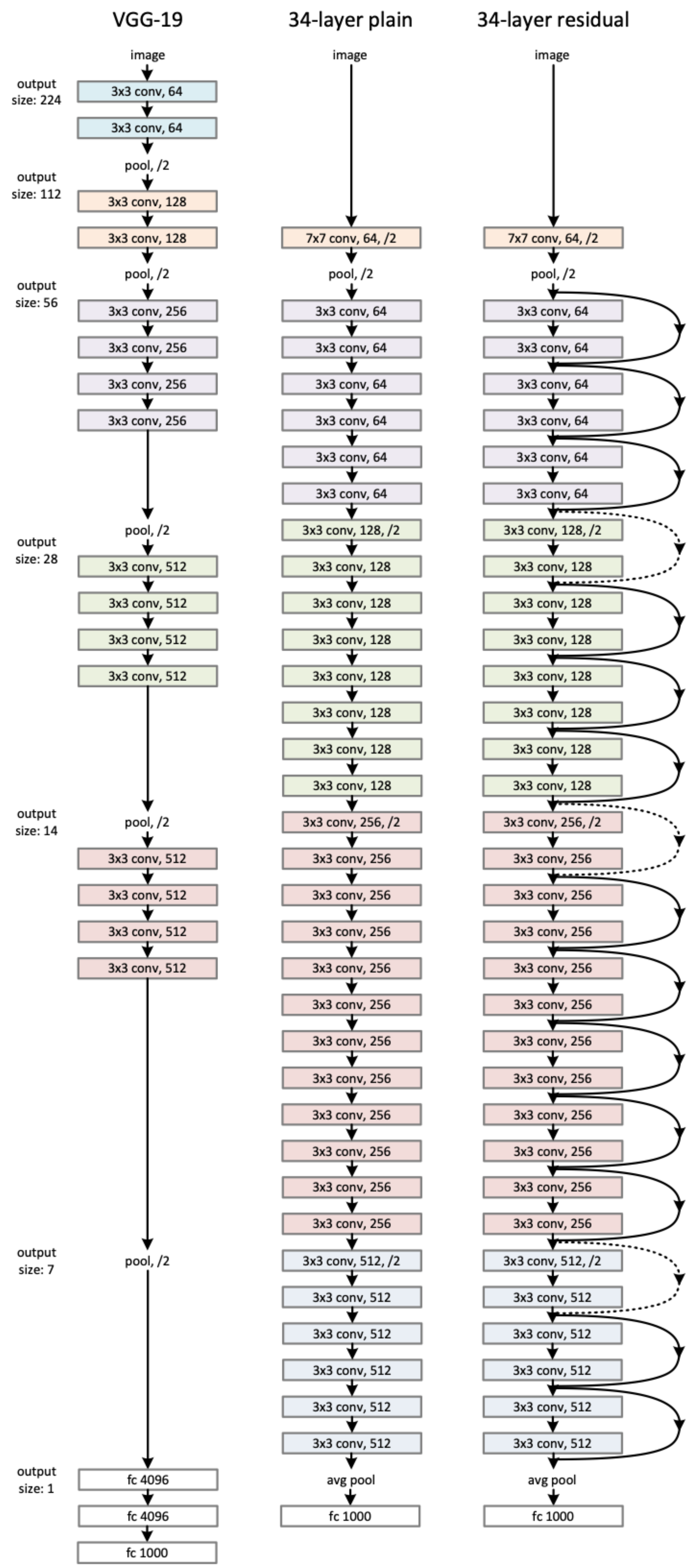


Figure 2. Residual learning: a building block.

$$\boxed{\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}.}$$

$$\boxed{\mathcal{F} = W_2\sigma(W_1\mathbf{x})}$$

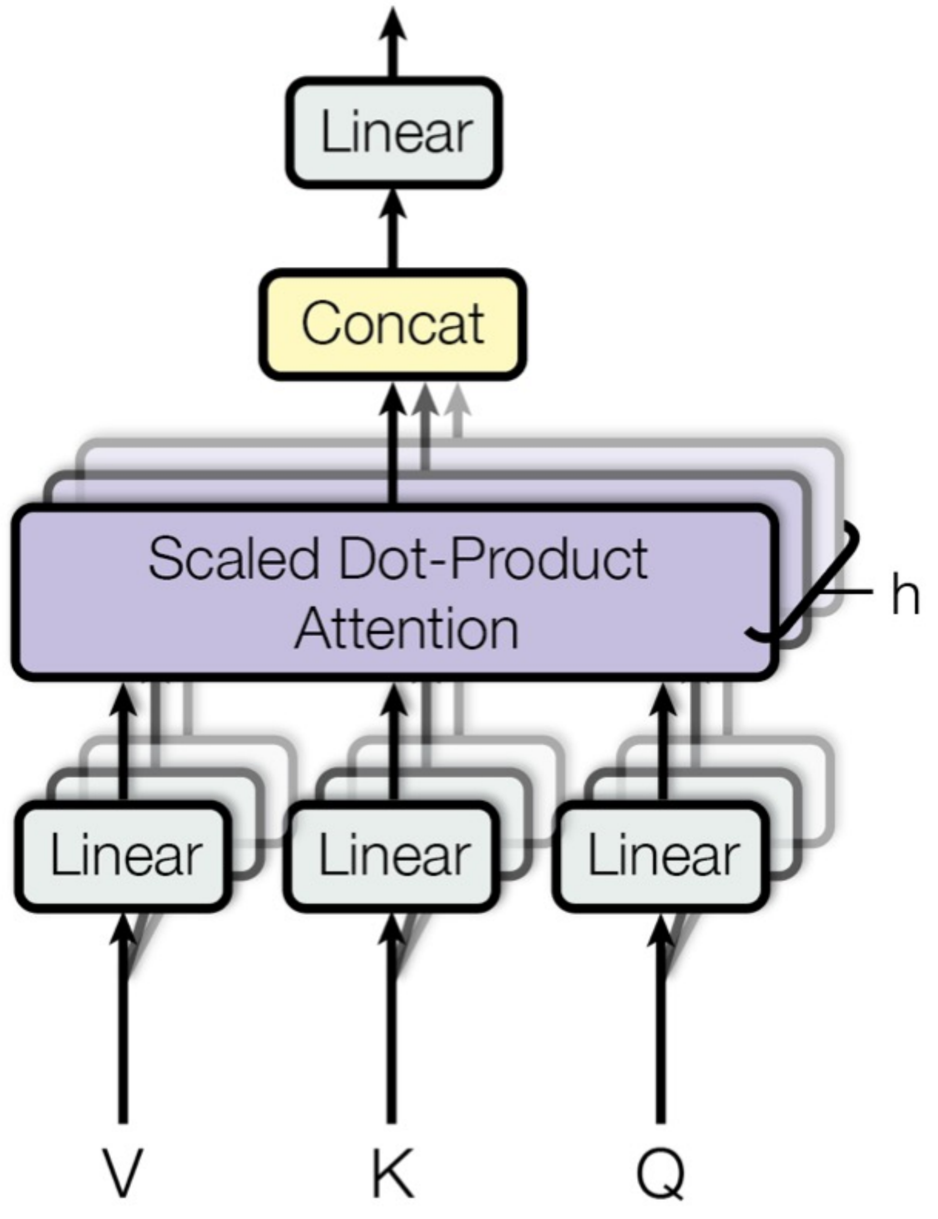
- σ denotes ReLU
- W_s is only used when matching dimensions



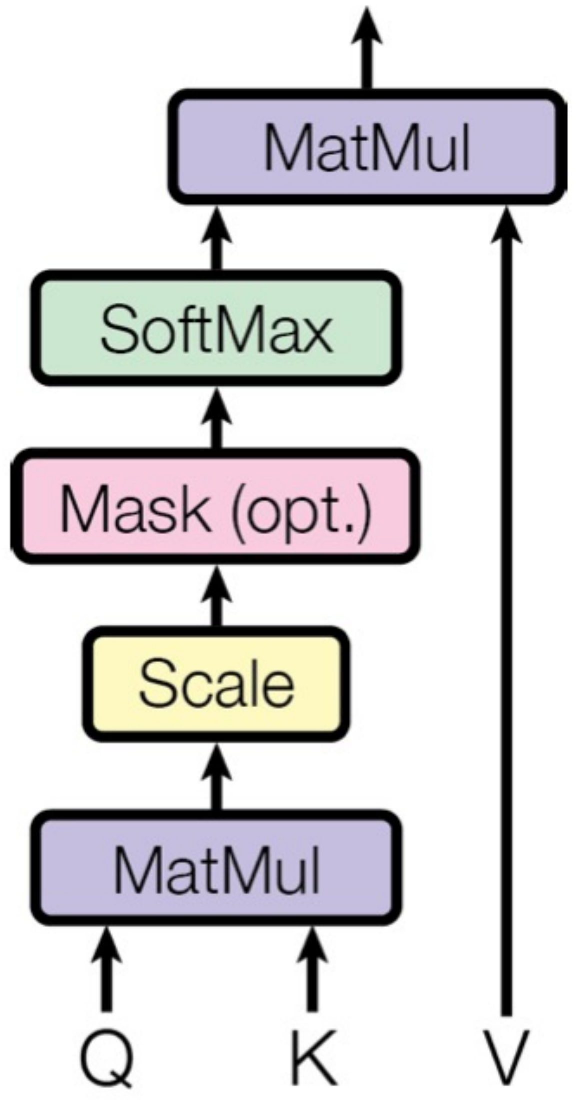
Transformer

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Multi-Head Attention



Scaled Dot-Product Attention



$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

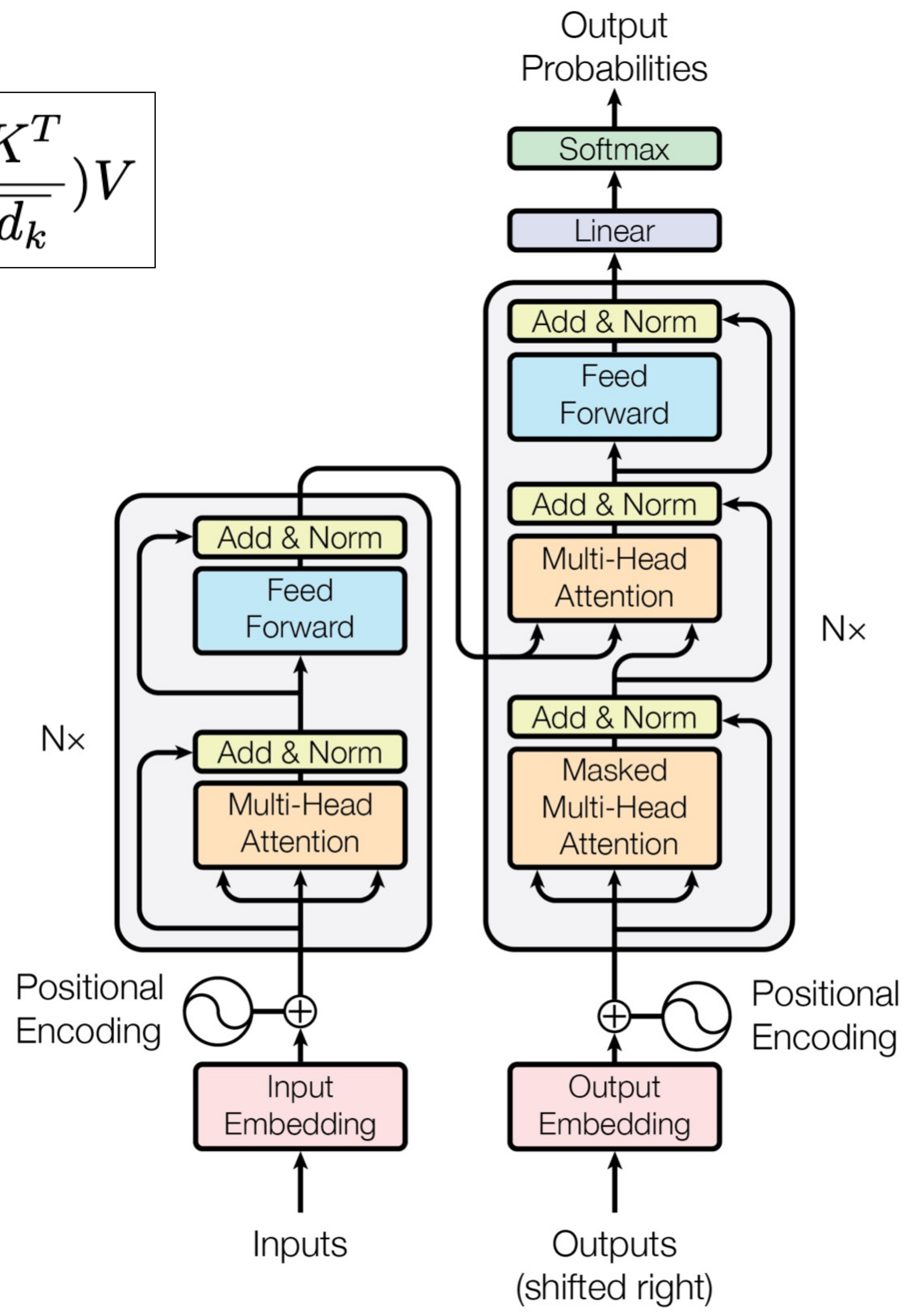
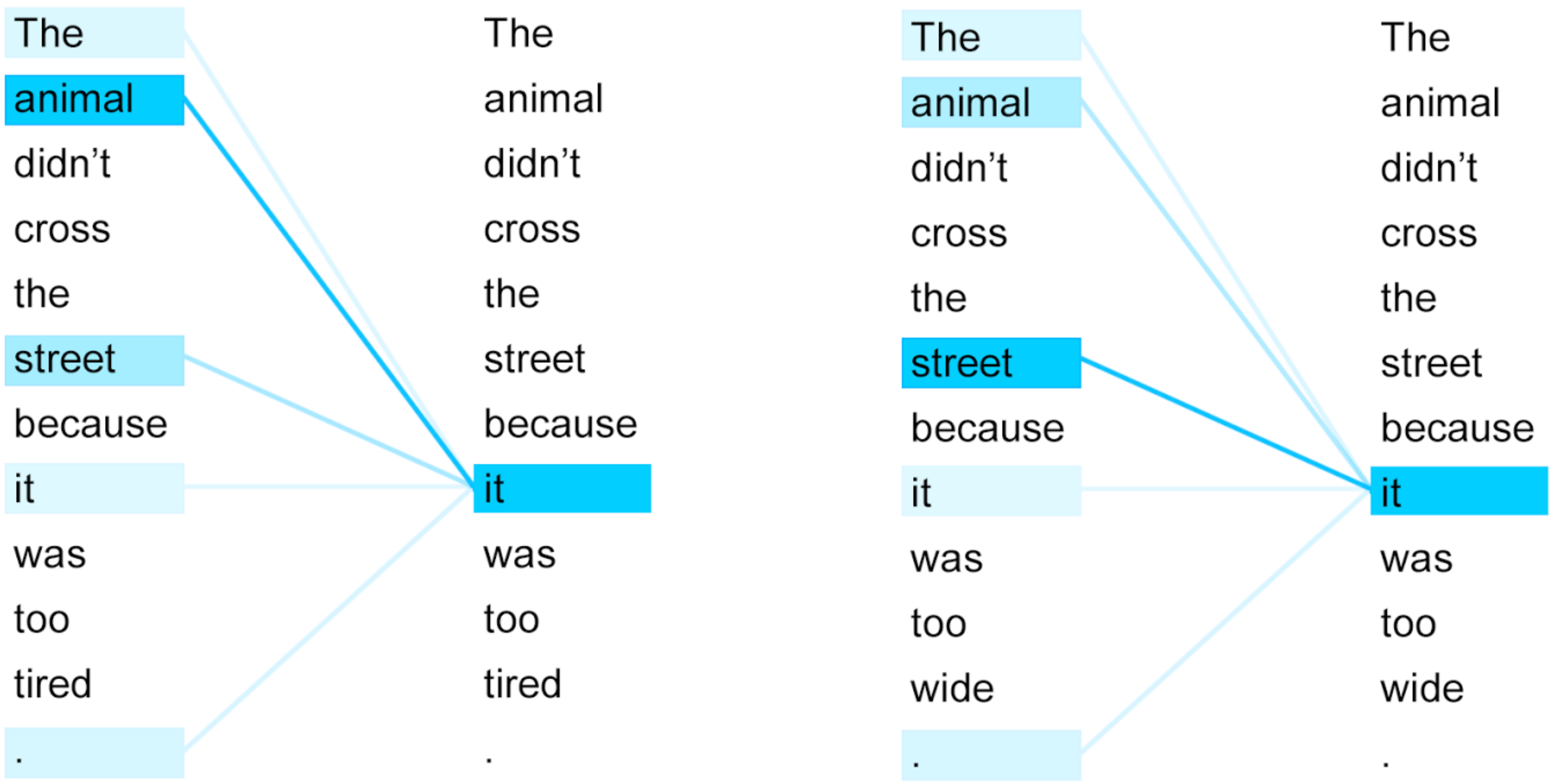


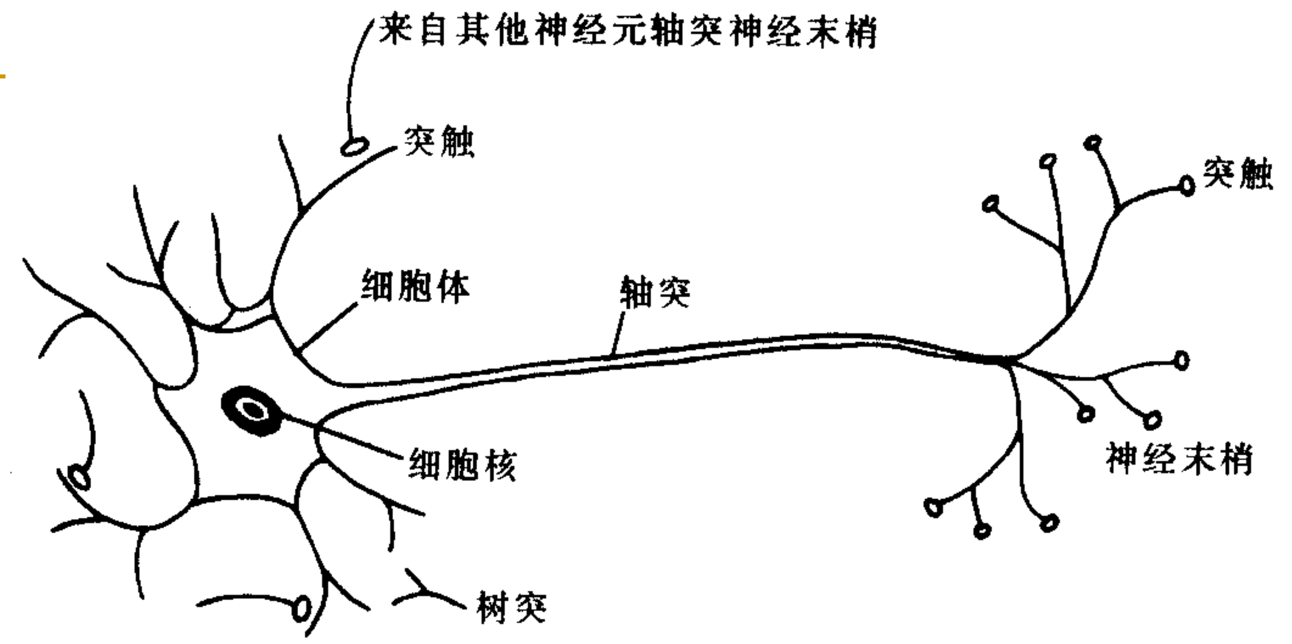
Figure 1: The Transformer - model architecture.

Ashish Vaswani et. al. Attention Is All You Need. NIPS 2017. Citations till April 7, 2024: 115657

Attention Visualization



Neuron Model



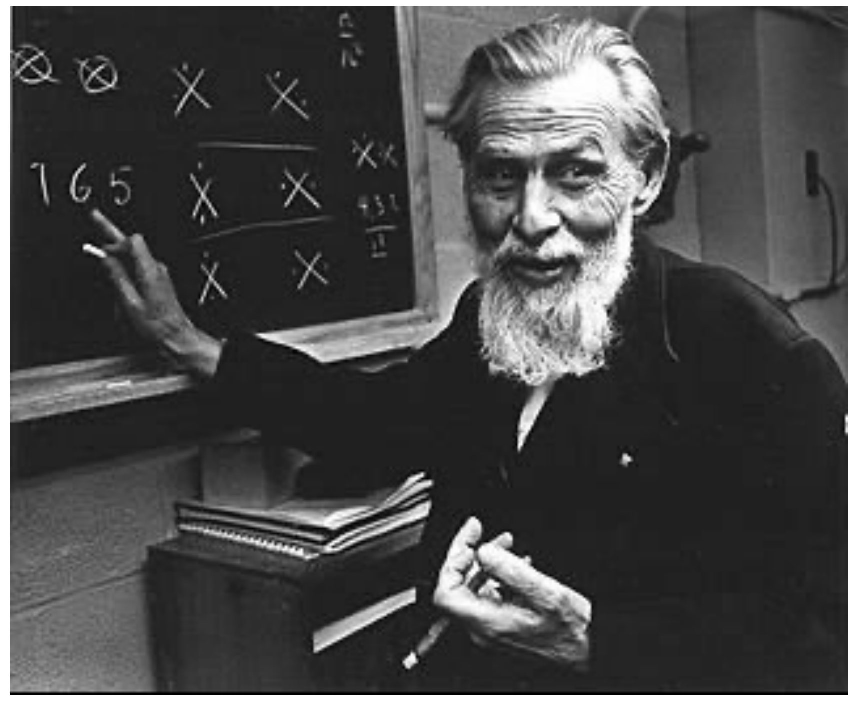
Definitions of neural networks

"Neural networks are massively parallel interconnected networks of simple elements and their hierarchical organizations which are intended to interact with the objects of the real world in the same way as biological nervous systems do" [Kohonen, 1988]

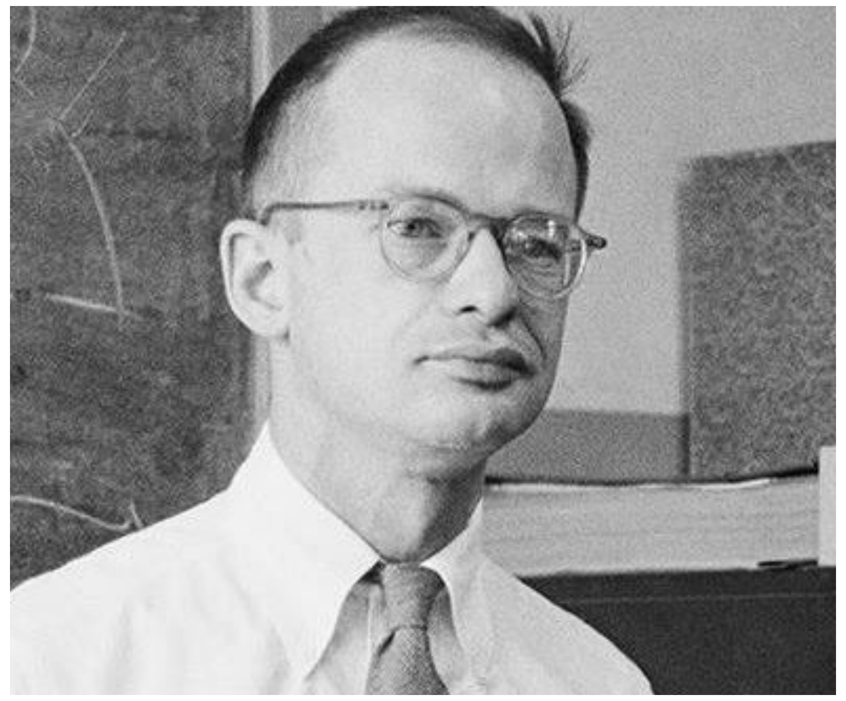
- In the context of machine leaning, neural networks refer to "neural networks learning", or in other words, the intersection of machine learning research and neural networks research
- The basic element of neural networks is neuron, which is the "simple element" in the above definition
- Biological neural networks： the neurons, when "excited", send neurotransmitters to interconnected neurons to change their electric potentials. When the electric potential exceeds a threshold, the neuron is activated, and it will send neurotransmitters to other neurons.

M-P Neuron Model [McCulloch and Pitts, 1943]

- Input： receive input signals from neurons
- Process： The weighted sum of received signals is compared against the threshold
- Output： the output signal is produced by the activation function



Warren S. McCulloch (1898-1969)



Walter Pitts (1923-1969)

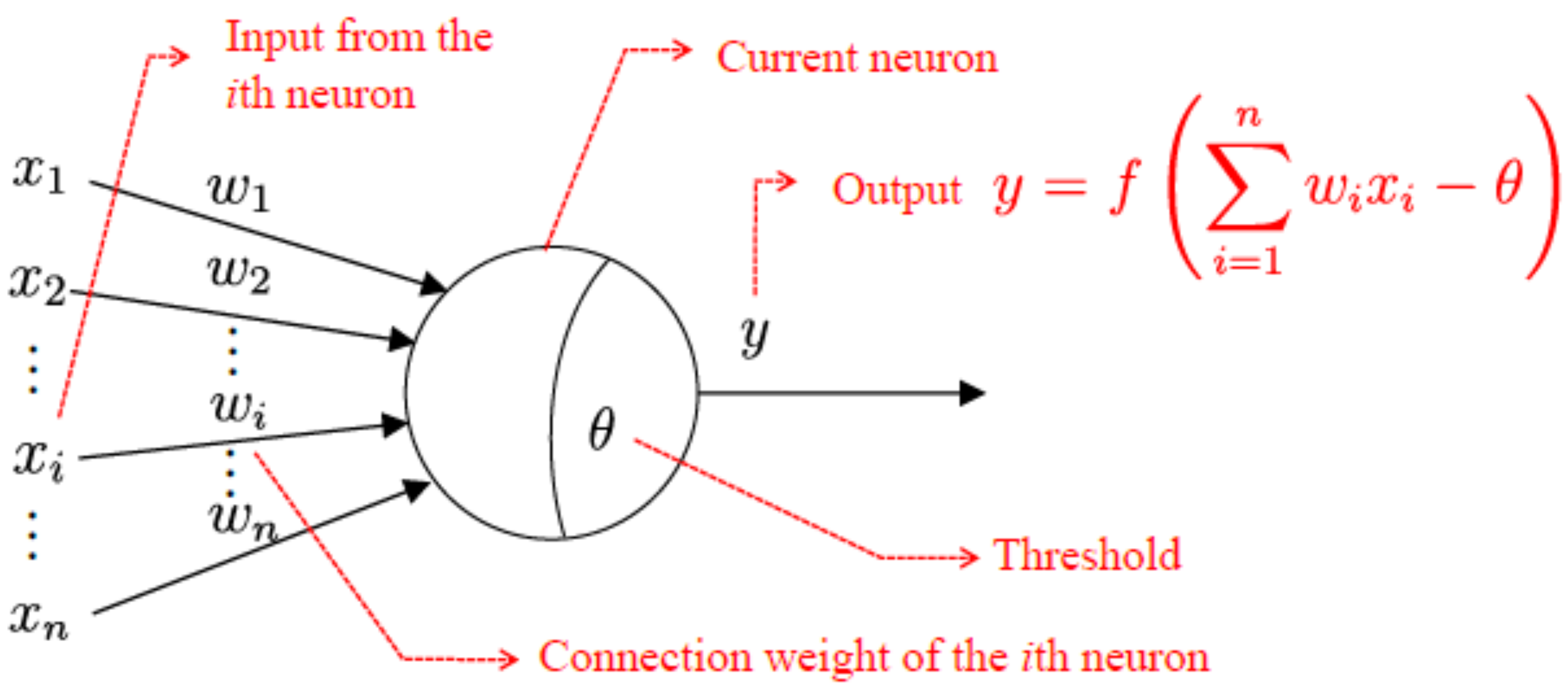
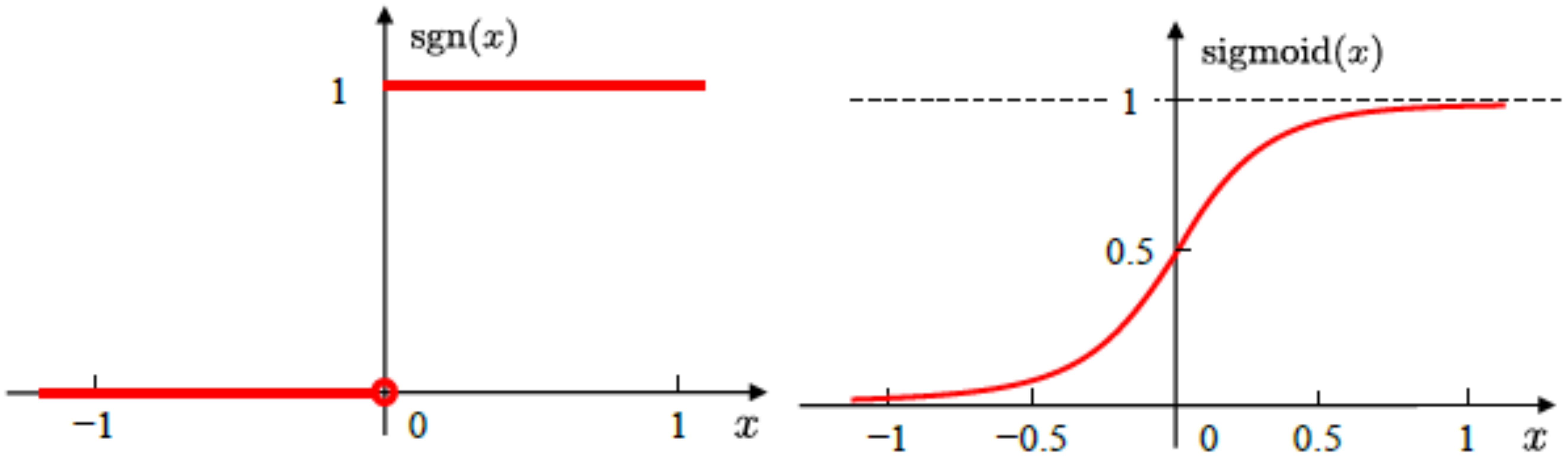


Fig. 5.1: The M-P neuron model.

Activation Function

- The ideal activation function is the step function, which maps the input value to “0” (non-excited) and “1” (excited).
- The step function are discontinuous and non-smooth, we often use the sigmoid function instead.



$$\mathrm{sgn}(x) = \begin{cases} 1, & x \geqslant 0; \\ 0, & x < 0. \end{cases}$$

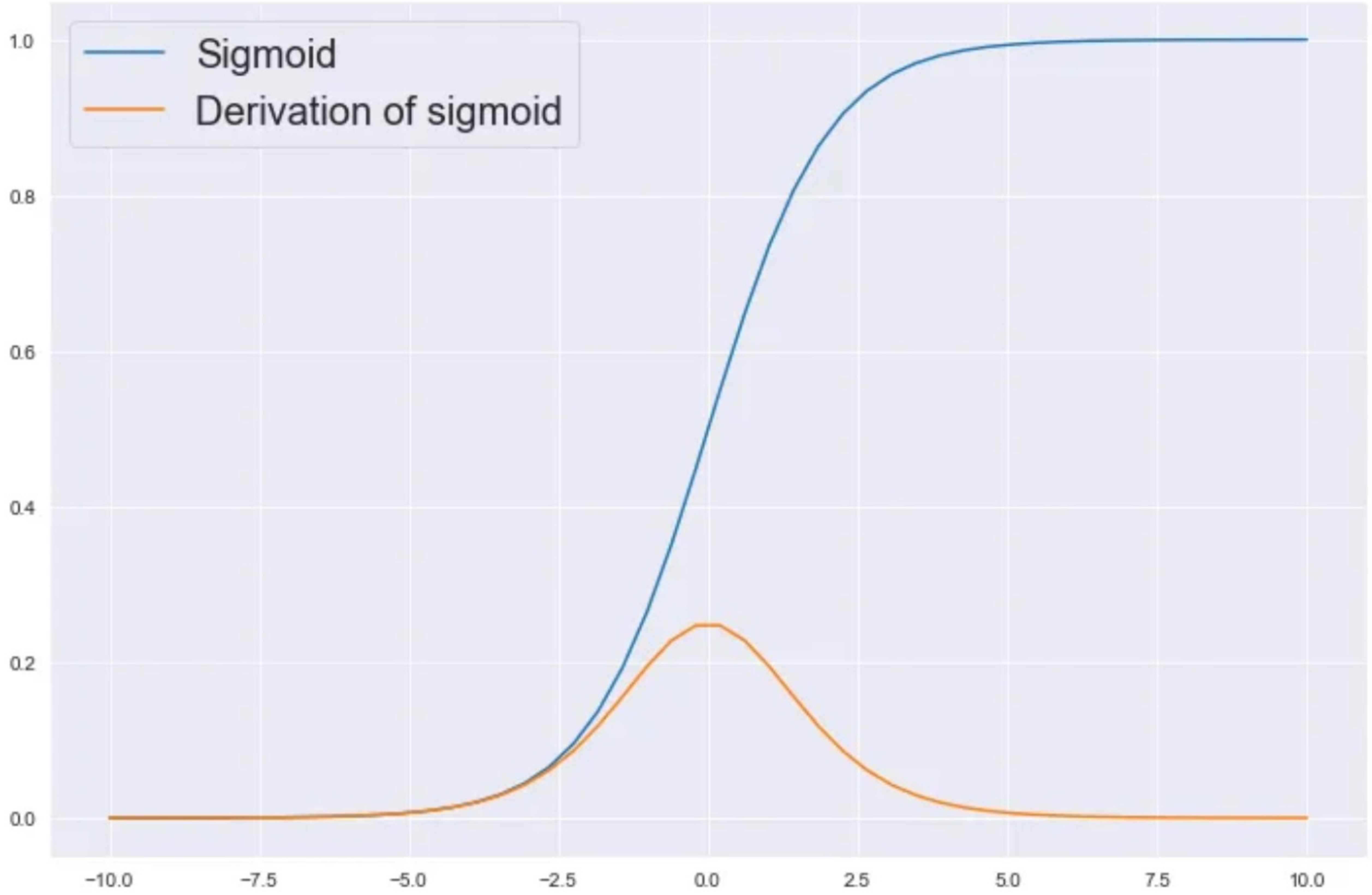
(a) Step function.

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

(b) Sigmoid function.

Fig. 5.2: Typical neuron activation functions.

Activation Function



Vanishing Gradient?

Perceptron

Perceptron is a binary classifier consisting of two layers of neurons. The input layer receives signals and transmits them to the output M-P neuron (threshold logic unit)

Perceptron can easily implement “AND”, “OR” and “NOT”

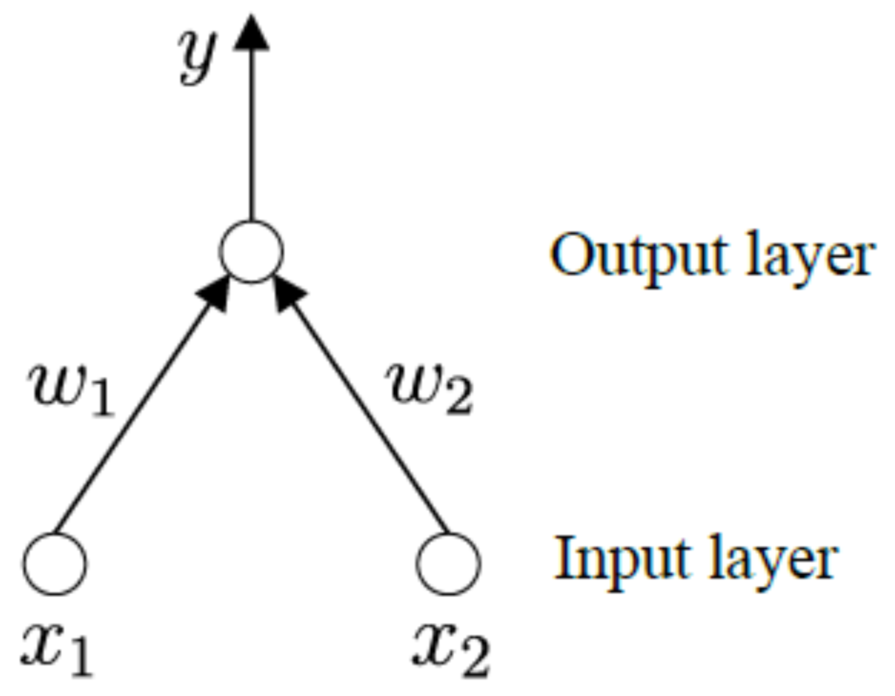


Fig. 5.3: A perceptron with two input neurons.

- “AND” $x_1 \wedge x_2$: letting $w_1 = w_2 = 1, \theta = 2$
  $y = f(1 \cdot x_1 + 1 \cdot x_2 - 2)$, and $y = 1$ iff $x_1 = x_2 = 1$
- “OR” $x_1 \vee x_2$: letting $w_1 = w_2 = 1, \theta = 0.5$, then
  $y = f(1 \cdot x_1 + 1 \cdot x_2 - 0.5)$, and $y = 1$ iff $x_1 = 1$ or $x_2 = 1$.
- “Not” $\neg x_1$: letting $w_1 = -0.6, w_2 = 0, \theta = -0.5$, then
  $y = f(-0.6 \cdot x_1 + 0 \cdot x_2 + 0.5)$, $y = 0$ when $x_1 = 1$ and $y = 1$ when $x_1 = 0$

Perceptron

Perceptron solving the “AND”, “OR” and “Not” problems

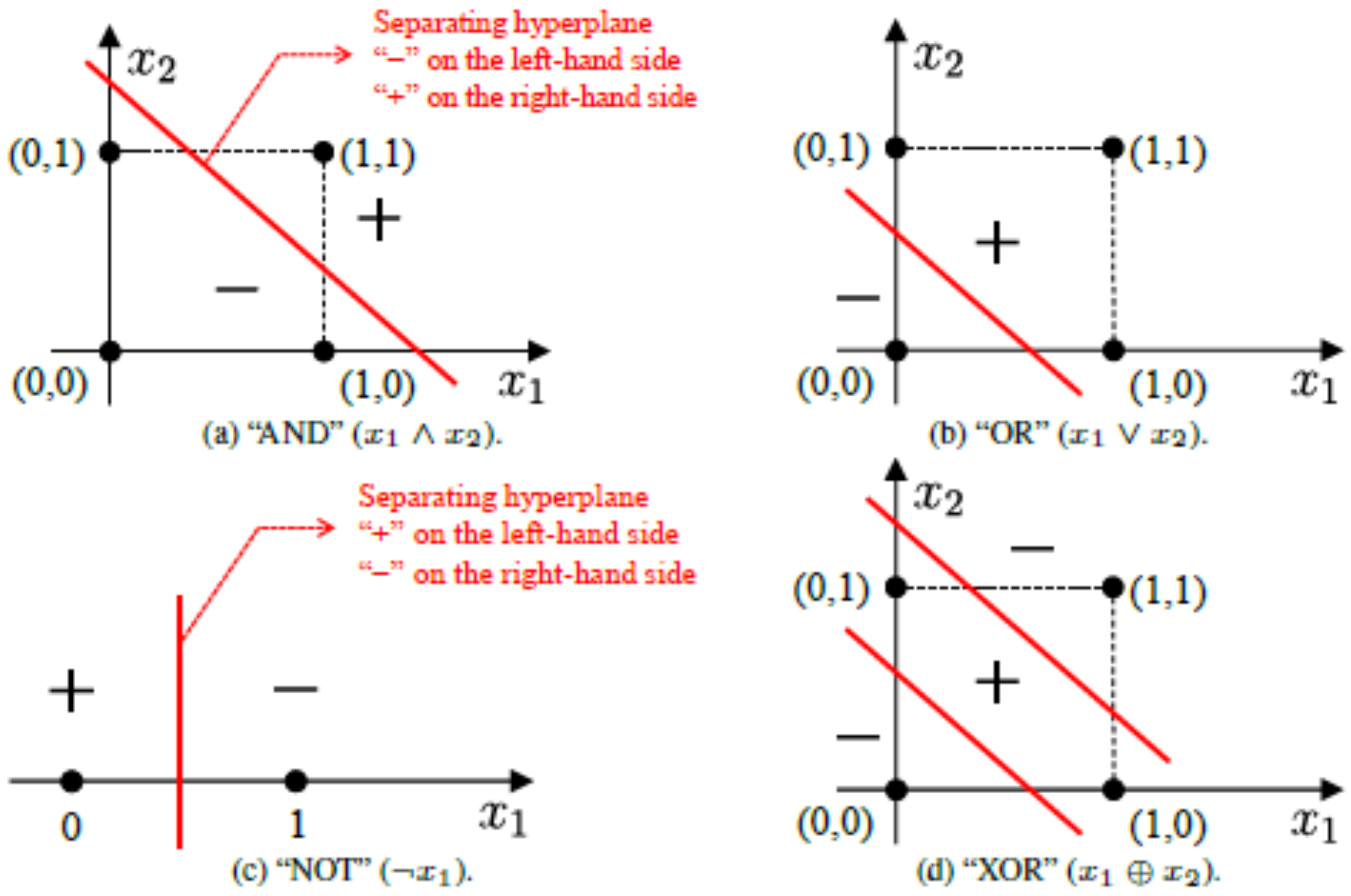
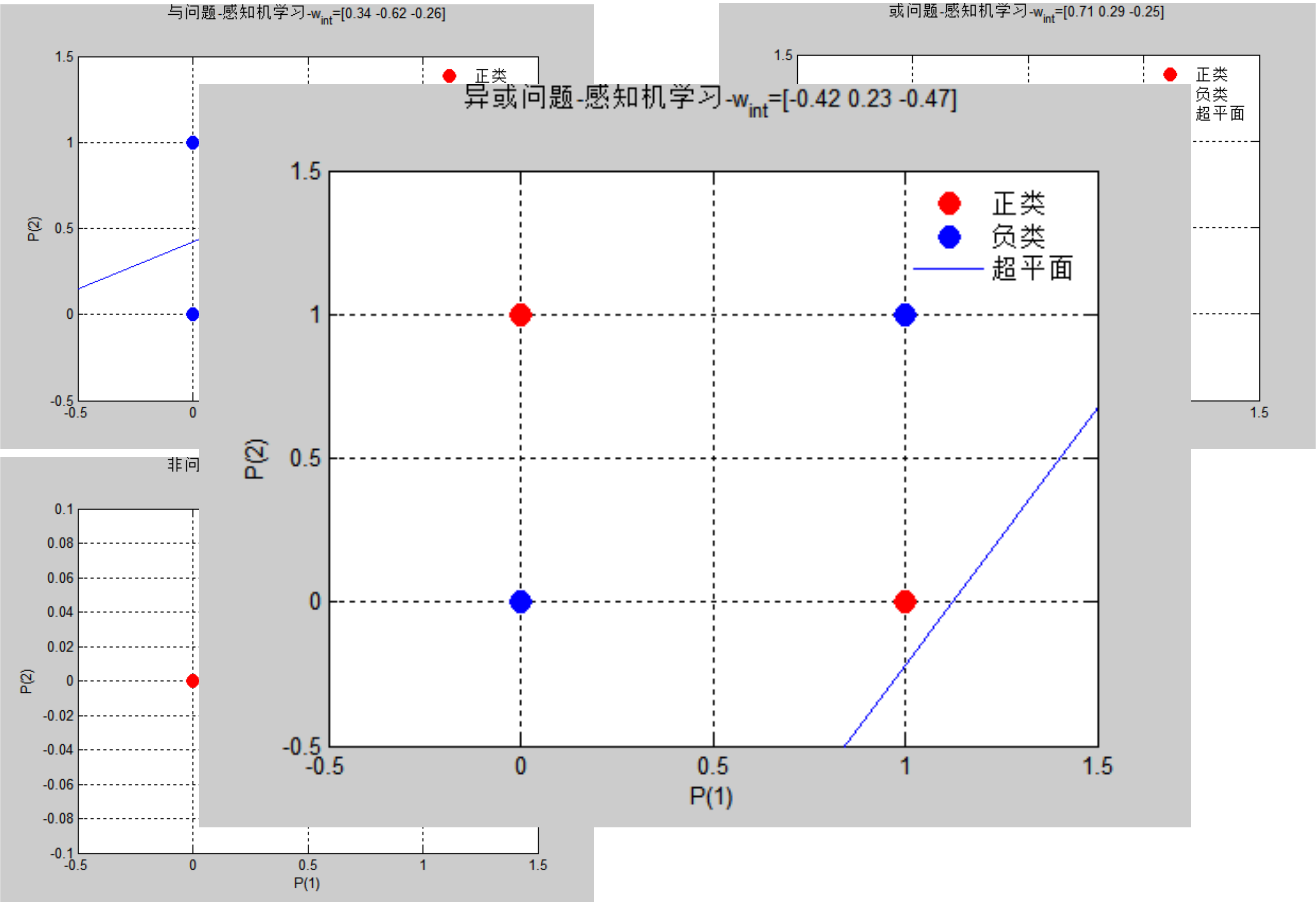


Fig. 5.4: “AND”, “OR” and “NOT” are linearly separable problems. “XOR” is a non-linearly separable problem.

Perceptron

Perceptron solving the "AND", "OR" and "Not" problems



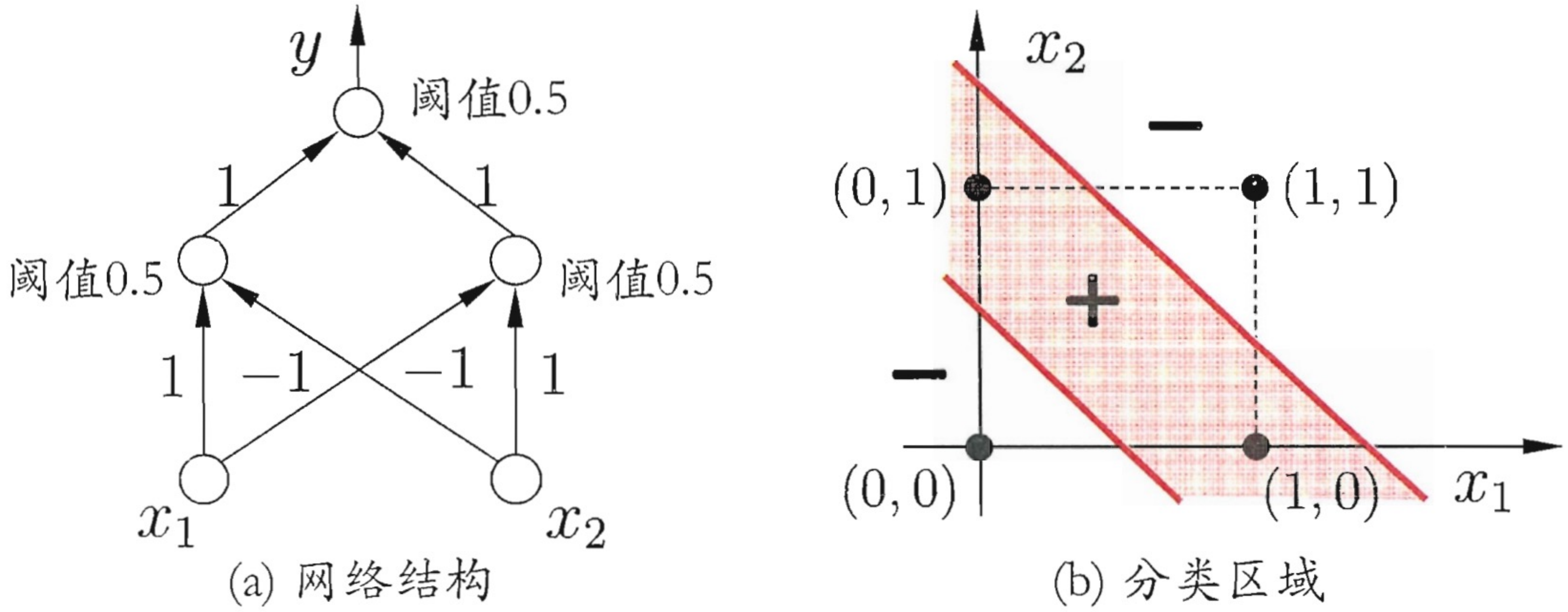
Multi-layer Network

The learning ability of perceptrons

- The learning process is guaranteed to converge for linear separable problems； otherwise fluctuation will happen. [Minsky and Papert, 1969]
- The learning ability of one-layer perceptrons is rather weak, which can only solve linear separable problems.
- In fact, the "AND", "OR" and "NOT" problems are all linear separable problems and the learning process is guaranteed to converge to an appropriate weight vector. However, perceptrons cannot solve non-linearly separable problems like "XOR".

Multi-layer Network

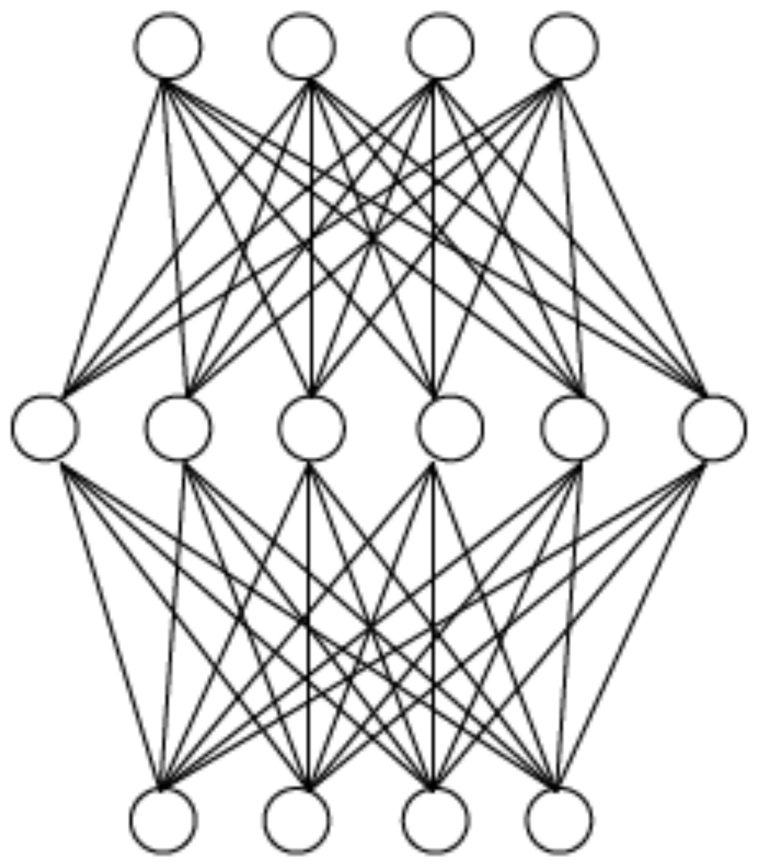
- A two-layer perceptron that solves the "XOR" problem.

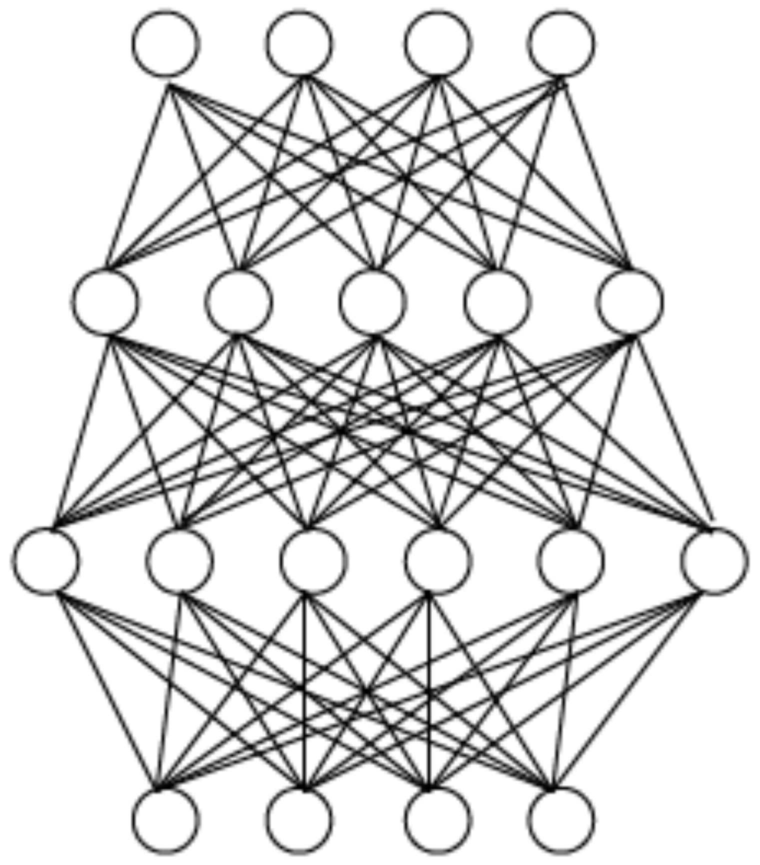


- The neuron layer between the input and output layer is known as the hidden layer, which has activation functions as the output layer.

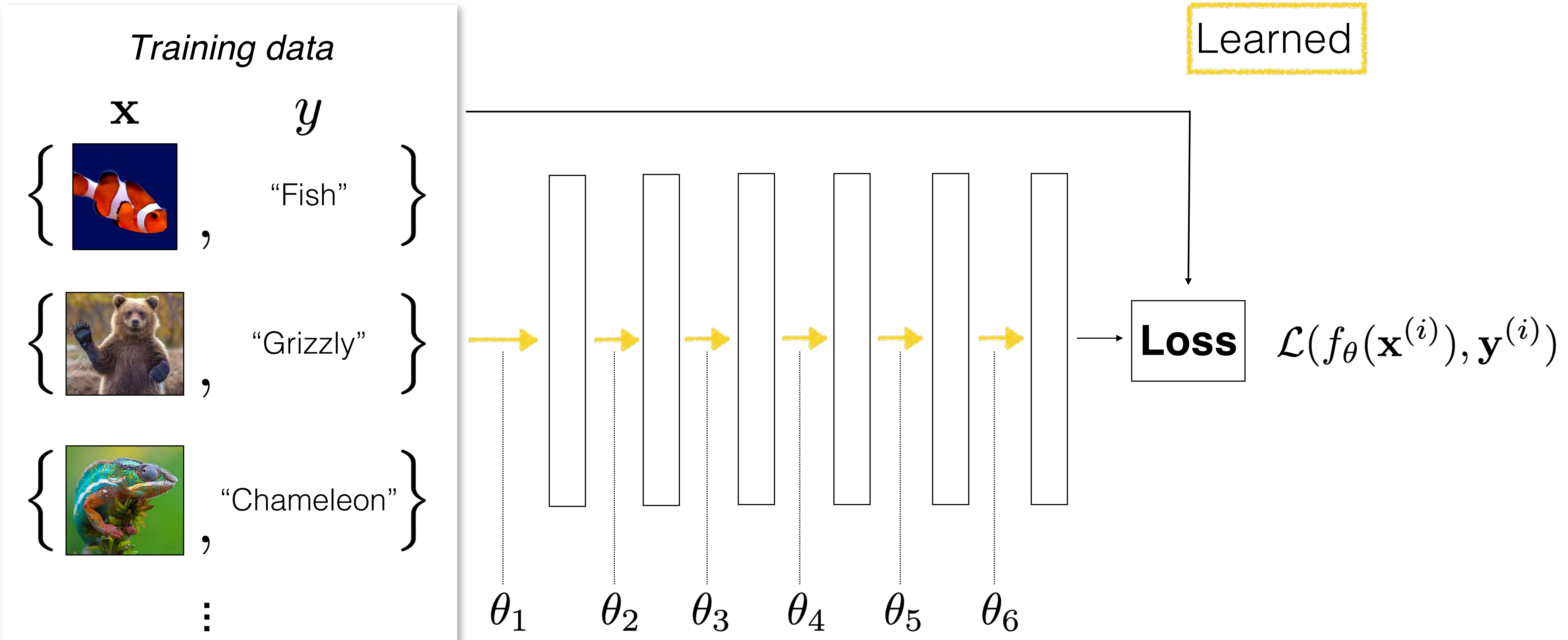
Multi-layer Network

- Definition： the neurons in each layer are fully connected with the neurons in the next layer.
- Forward： the input layer receives external signals, the hidden and output layers process (output) the signals.
- Learning： learning from data to adjust the "connection weights" and "thresholds".
- Multi-layer networks： neural networks with hidden layer(s)


(a) Single hidden layer.


(b) Two hidden layers.

Deep learning

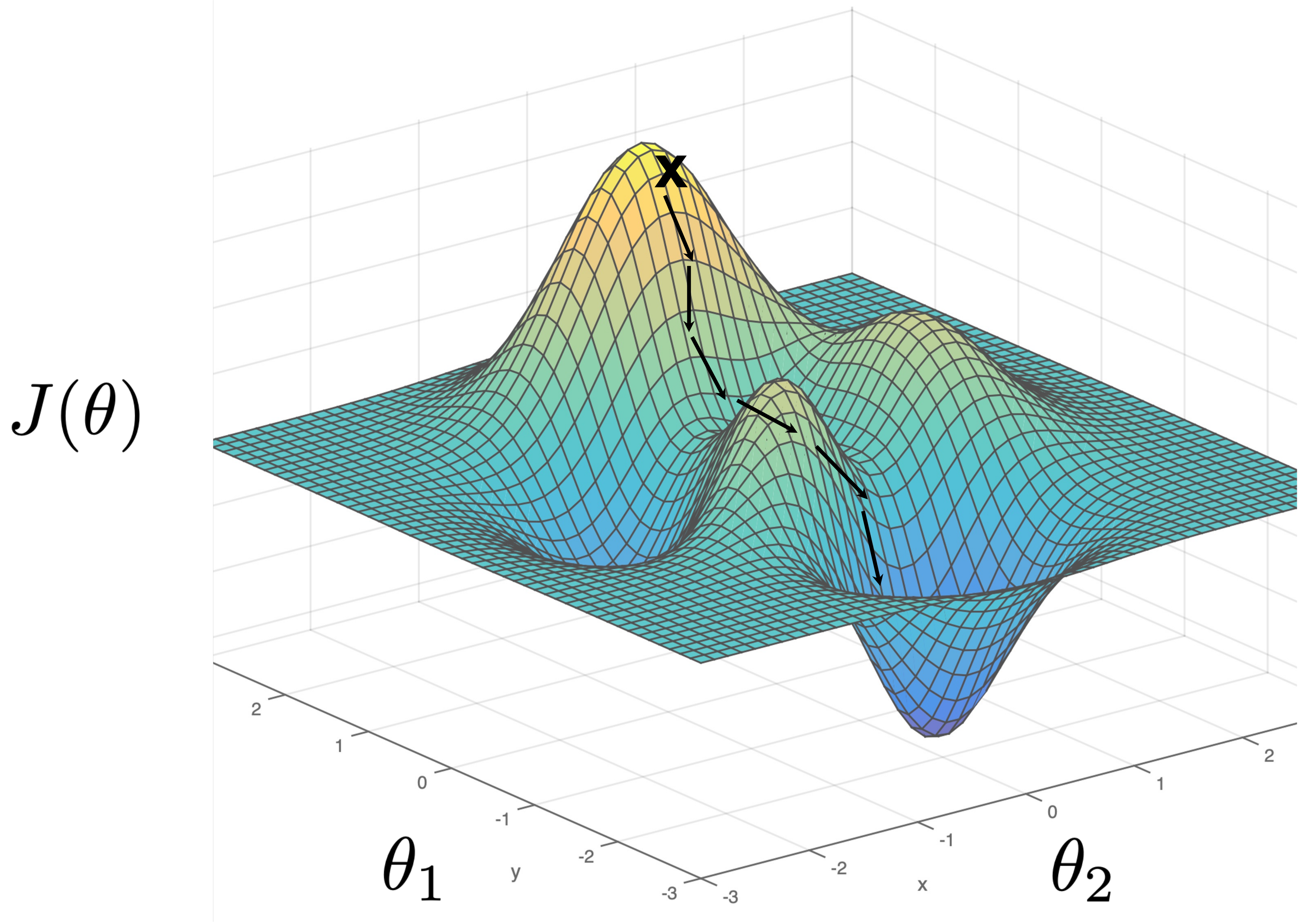


$$\theta^* = \arg\min_{\theta} \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$$

Gradient descent

$$\boxed{\theta^* = \arg\min_{\theta} \underbrace{\sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})}_{J(\theta)}}$$

Gradient descent



$$\boxed{\theta^* = \arg\min_{\theta} J(\theta)}$$

Gradient descent

$$\boxed{\theta^* = \arg\min_{\theta} \underbrace{\sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})}_{J(\theta)}}$$

One iteration of gradient descent:

$$\theta^{t+1} = \theta^t - \eta_t \frac{\partial J(\theta)}{\partial \theta}\bigg|_{\theta=\theta^t}$$

learning rate

Gradient descent

$$\boxed{\theta^* = \arg\min_{\theta} \underbrace{\sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})}_{J(\theta)}}$$

For large N, computing J in every iteration can be expensive

Stochastic gradient descent (SGD)

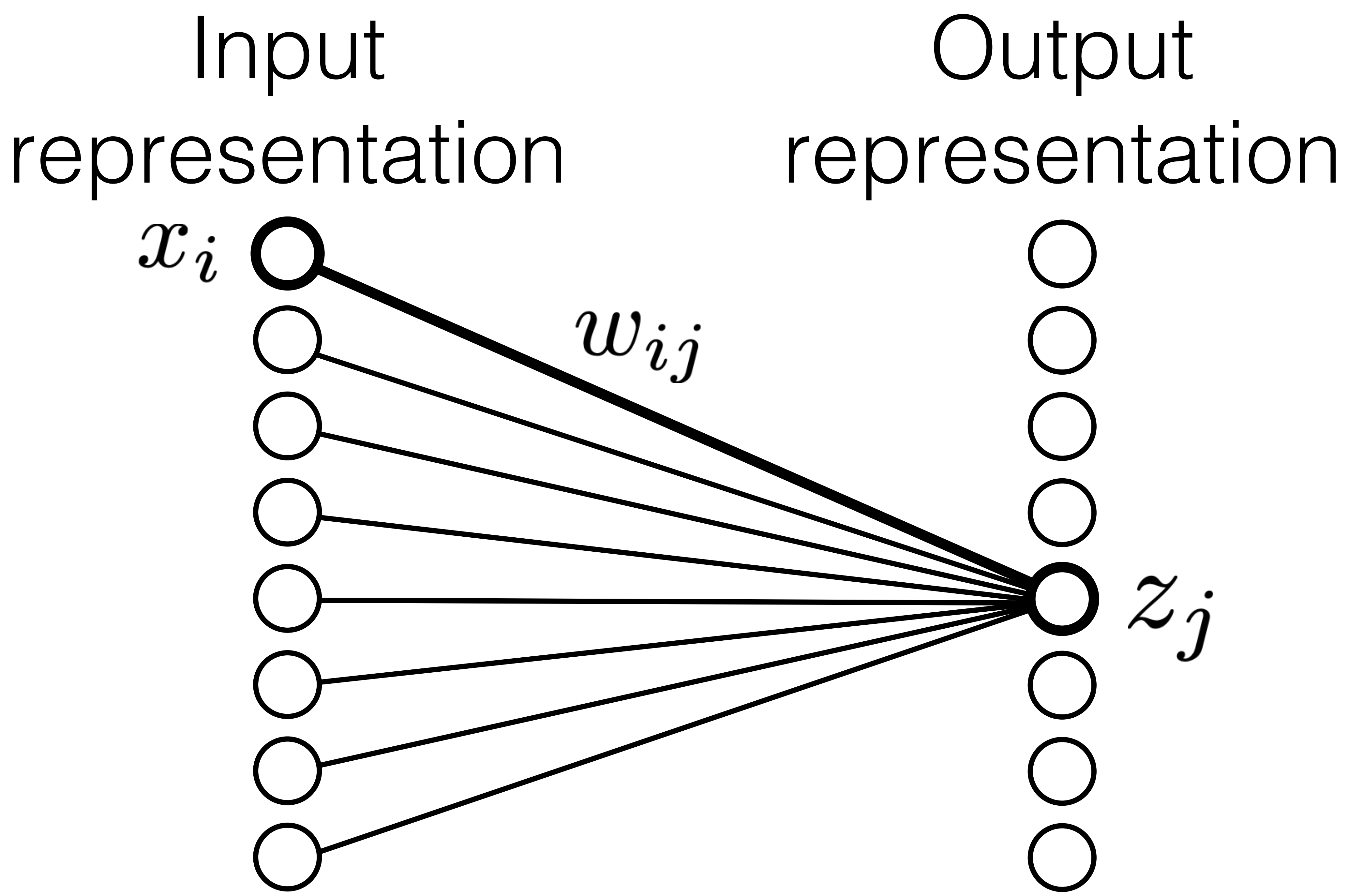
- Want to minimize overall loss function **J**, which is sum of individual losses over each example.
- In Stochastic gradient descent, compute gradient on sub-set (batch) of data.

  If batchsize=1 then θ is updated after each example.

  If batchsize=N (full set) then this is standard gradient descent.
- Gradient direction is noisy, relative to average over all examples (standard gradient descent).
- Advantages
  - Faster: approximate total gradient with small sample
  - Implicit regularizer
- Disadvantages
  - High variance, unstable updates

Computation in a neural net

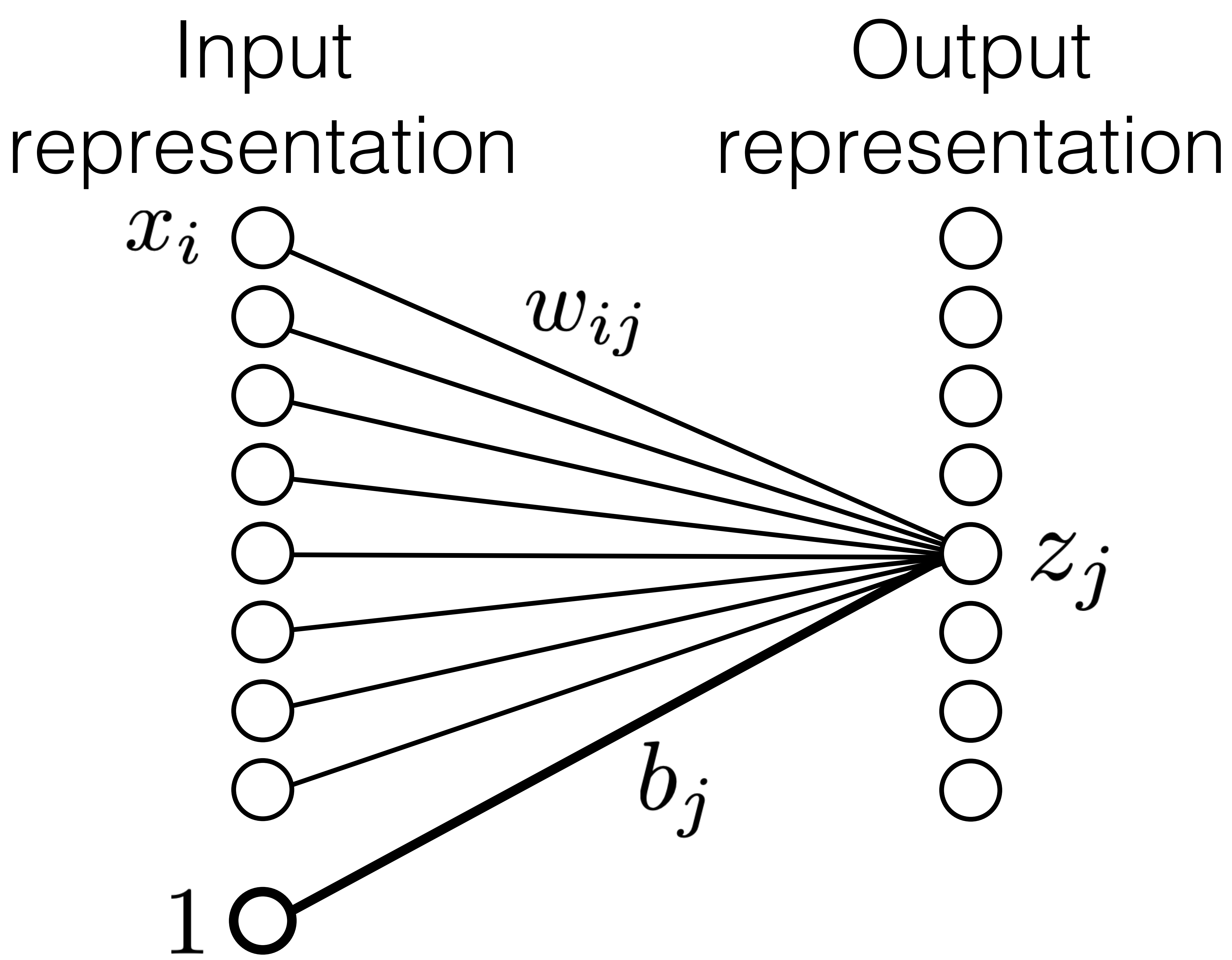
Linear layer



$$z_j = \sum_i w_{ij} x_i$$

Computation in a neural net

Linear layer



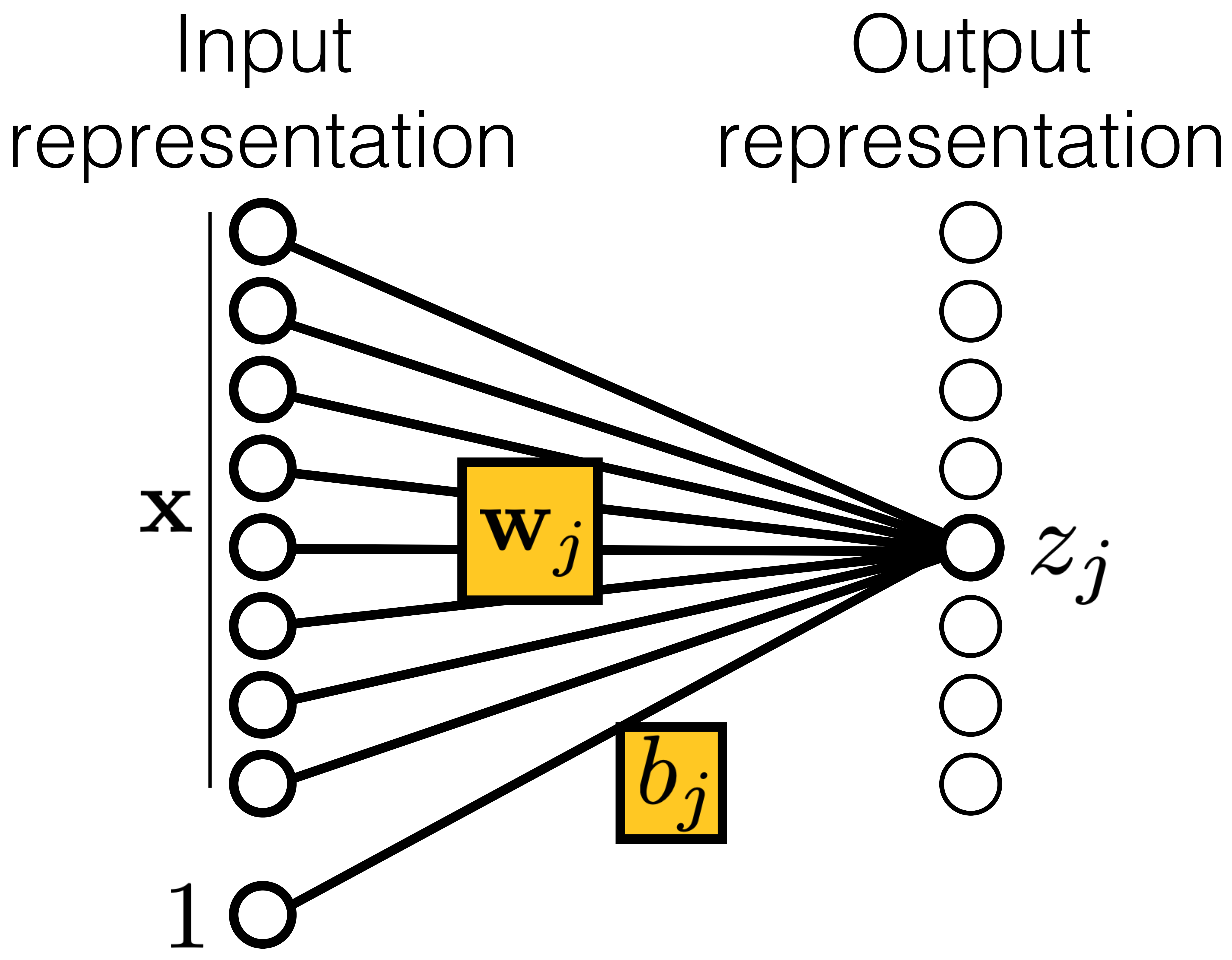
$$z_j = \sum_i w_{ij} x_i + b_j$$

weights

bias

Computation in a neural net

Linear layer



weights

$$z_j = \mathbf{x}^T \mathbf{w}_j + b_j$$

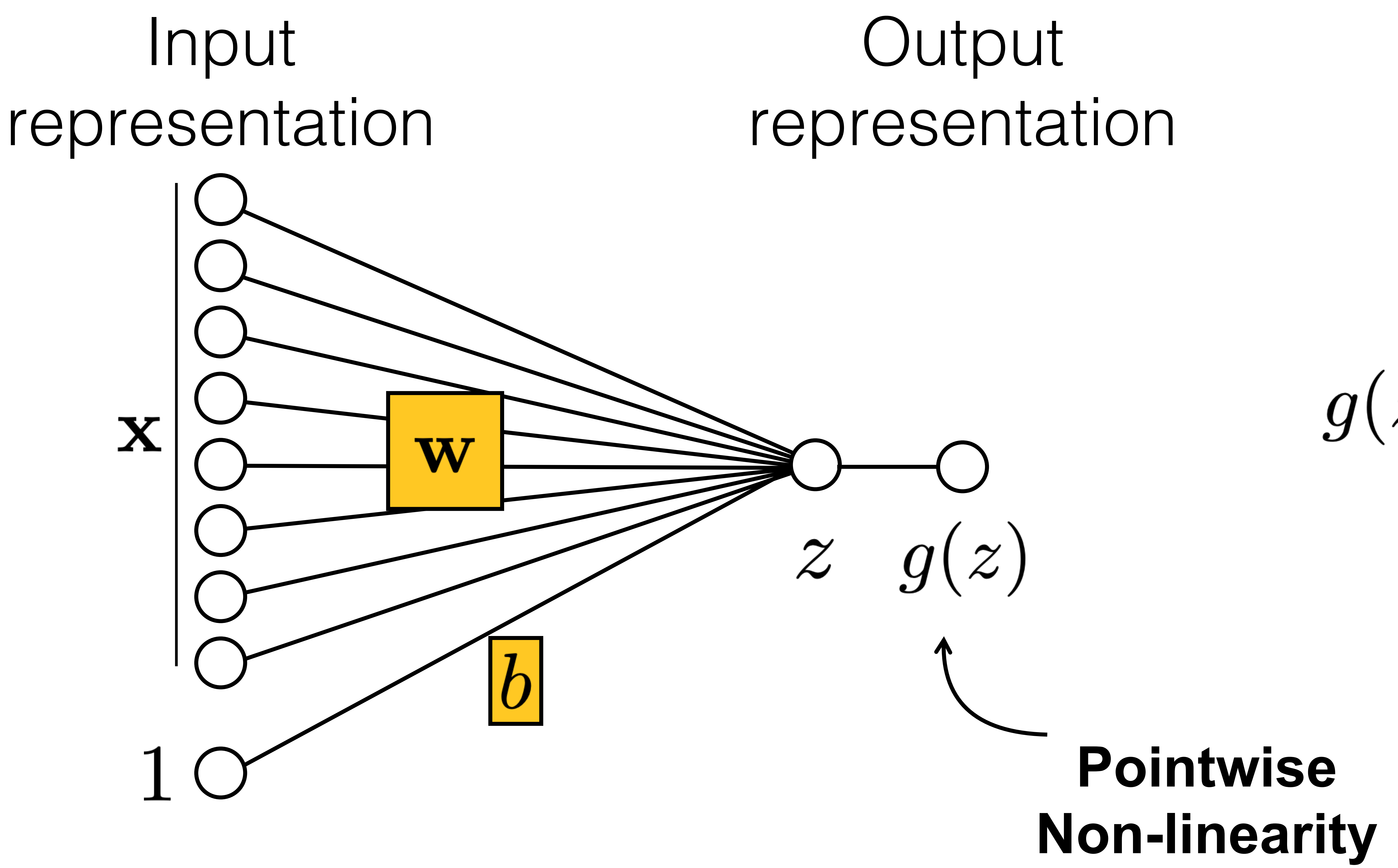
bias

$$\theta = \{\mathbf{W}, \mathbf{b}\}$$

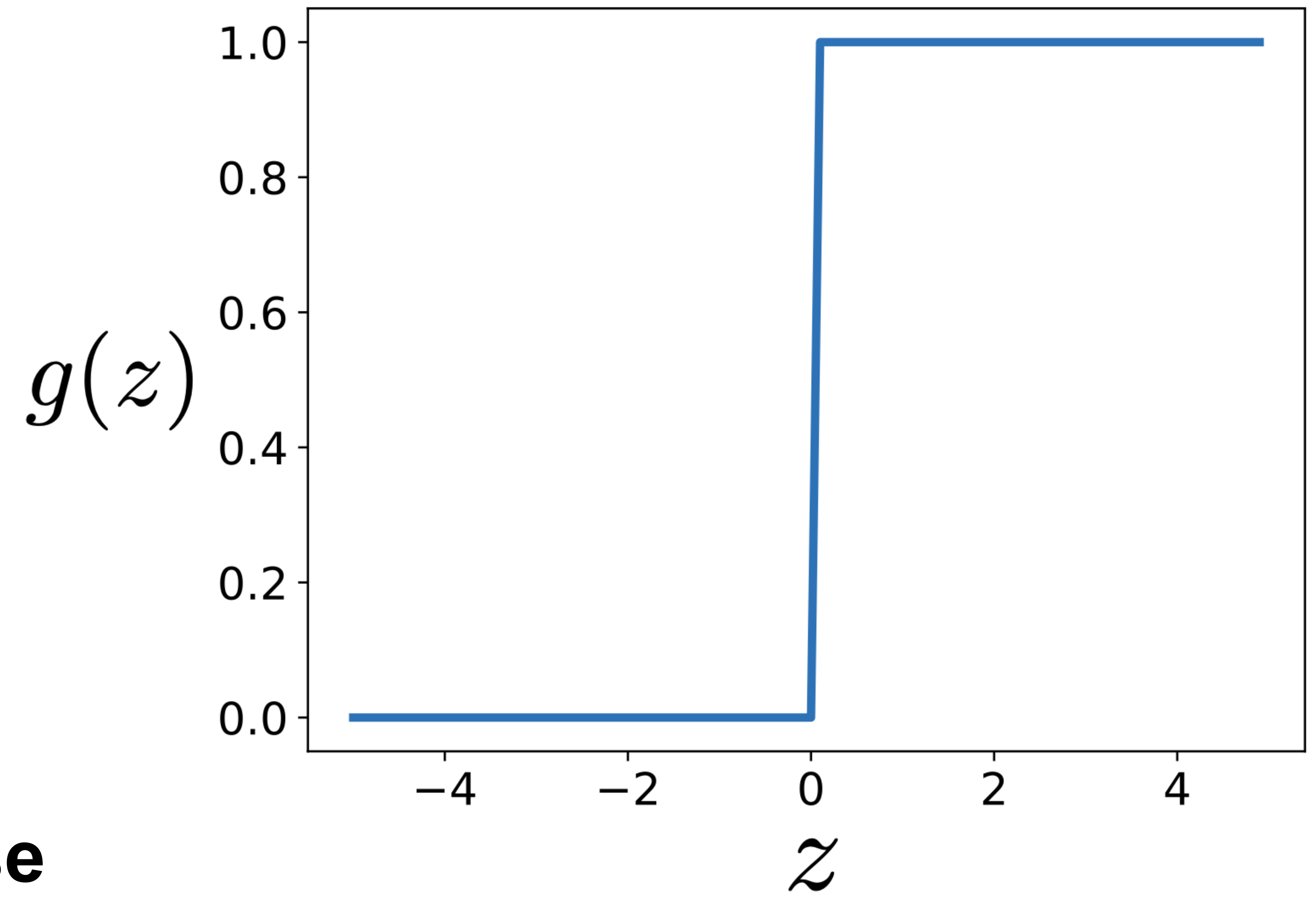
parameters of the model

Computation in a neural net

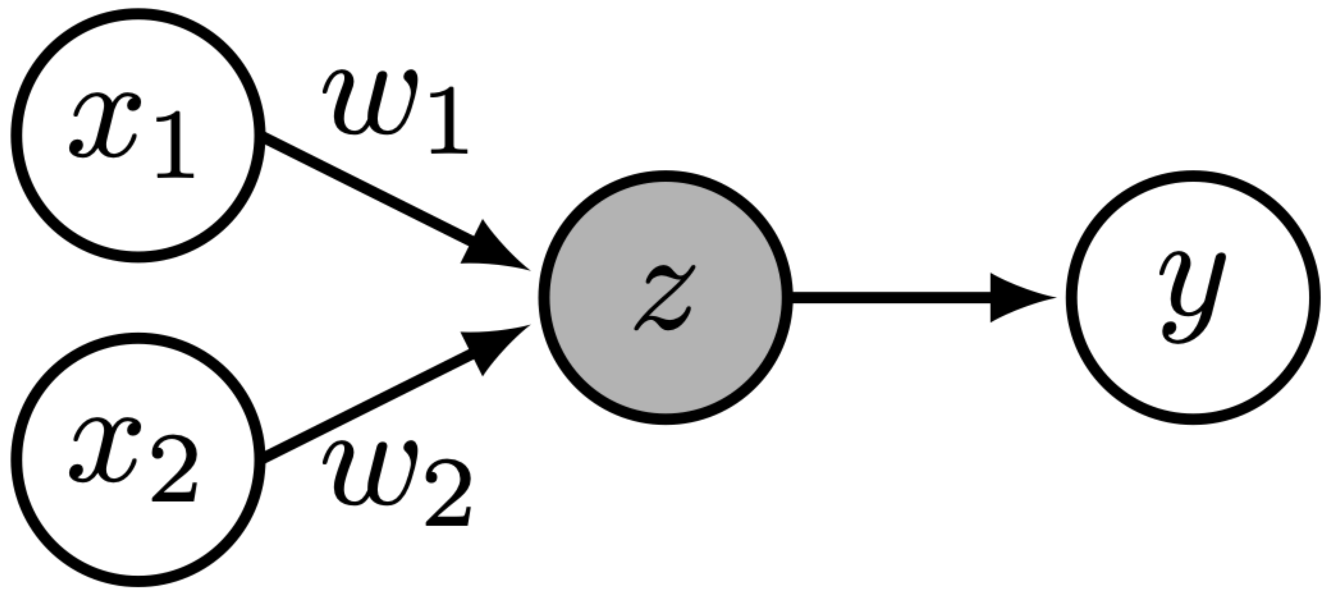
"Perceptron"



$$g(z) = \begin{cases} 1, & \text{if} \quad z > 0 \\ 0, & \text{otherwise} \end{cases}$$

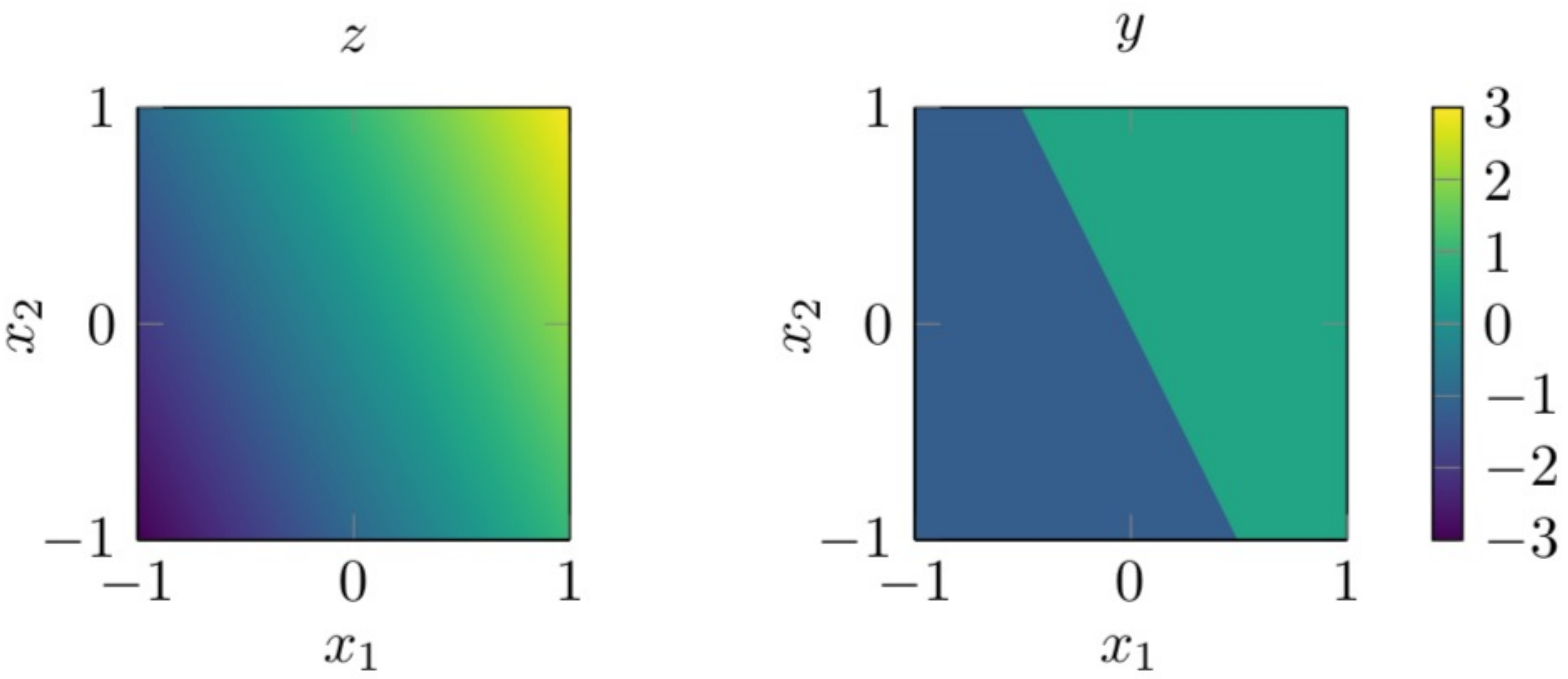


Example: linear classification with a perceptron

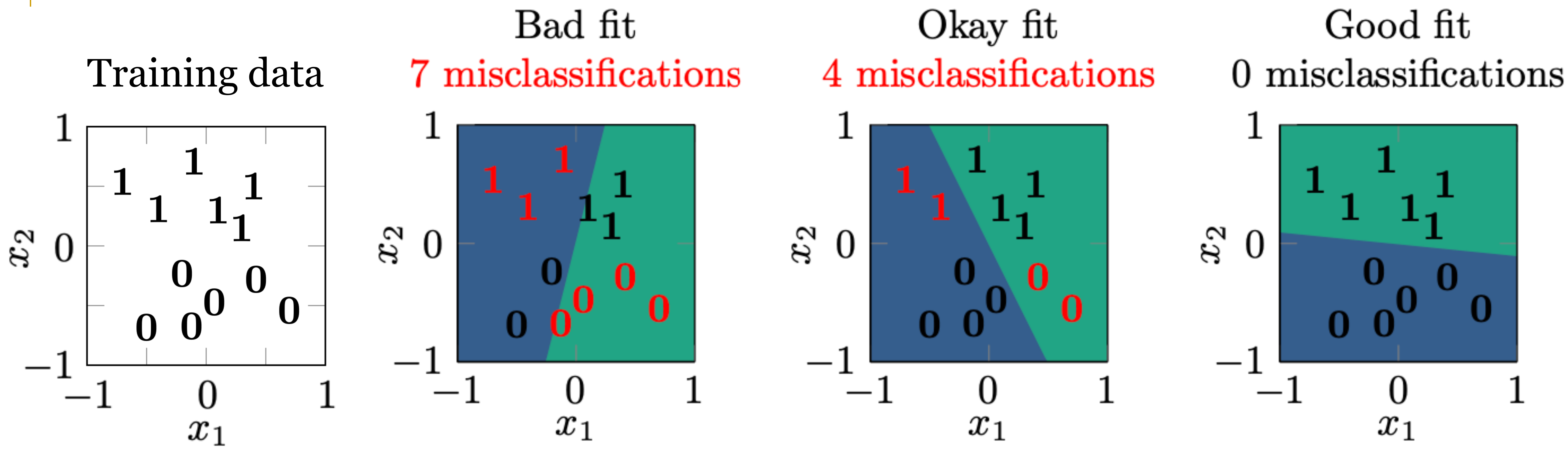


$$z = \mathbf{x}^T \mathbf{w} + b$$

$$y = g(z)$$

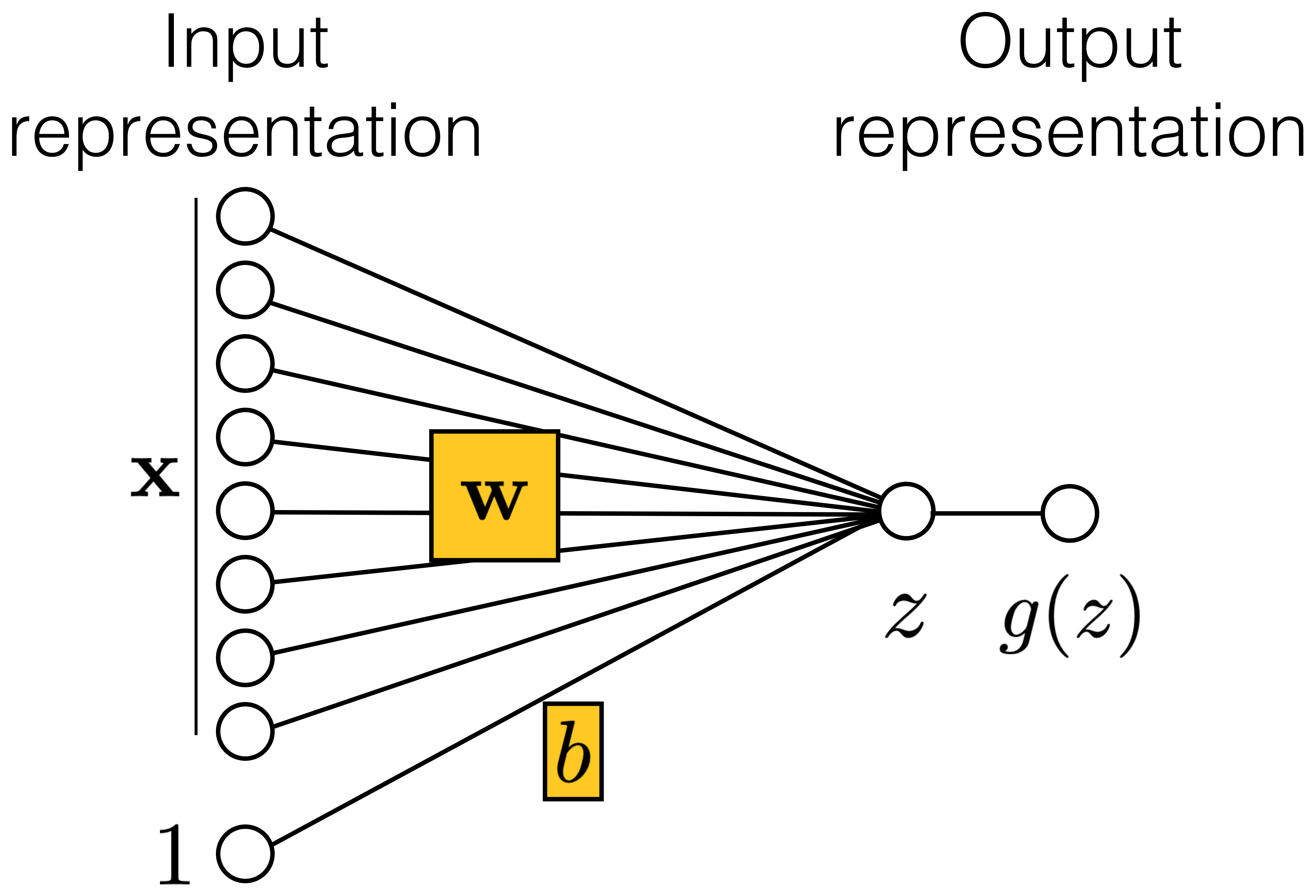


One layer neural net (perceptron) can perform linear classification!

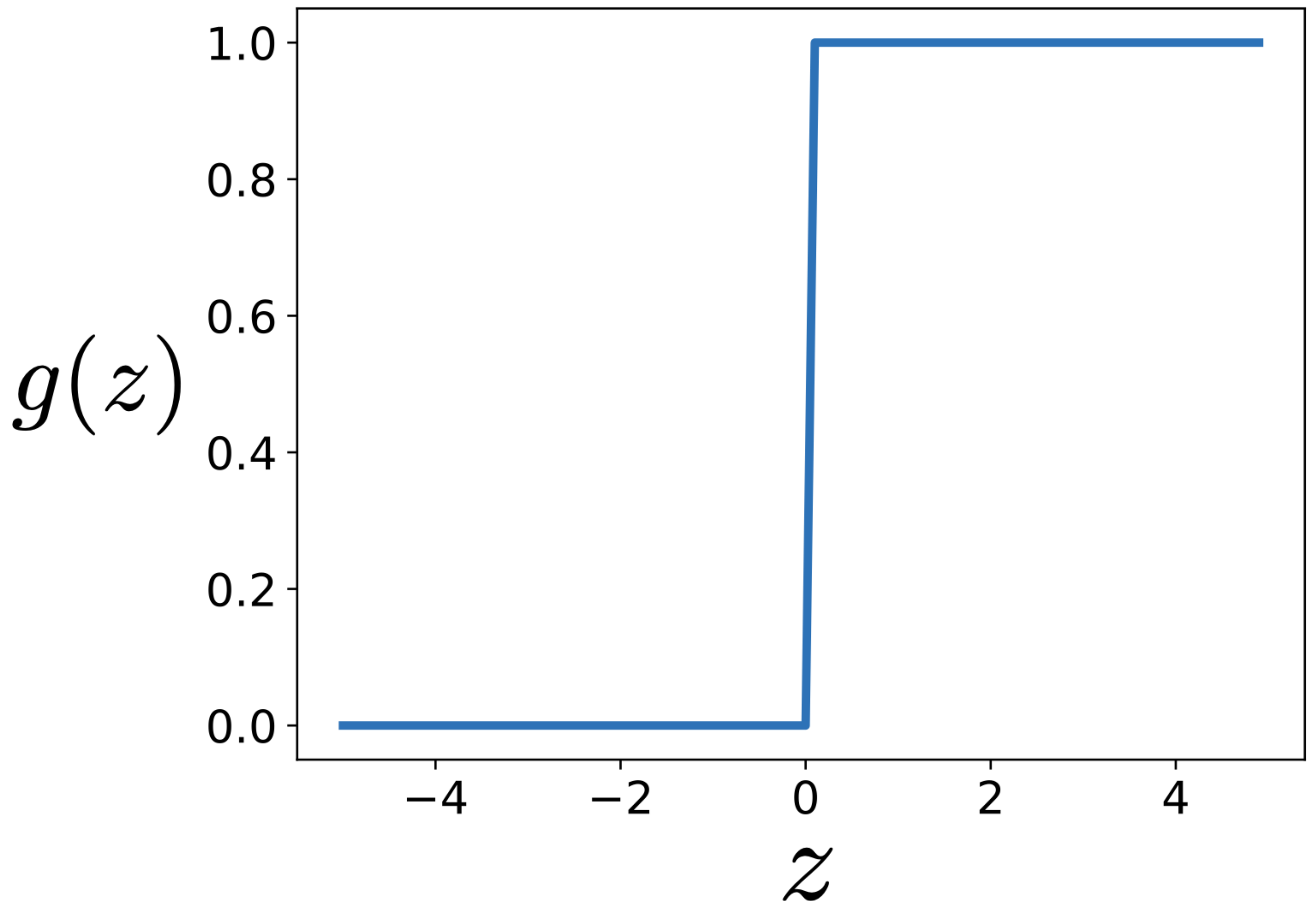


$$\mathbf{w}^*, b^* = \arg\min_{\mathbf{w},b} \sum_{i=1}^{N} \mathcal{L}(g(z^{(i)}), y^{(i)})$$

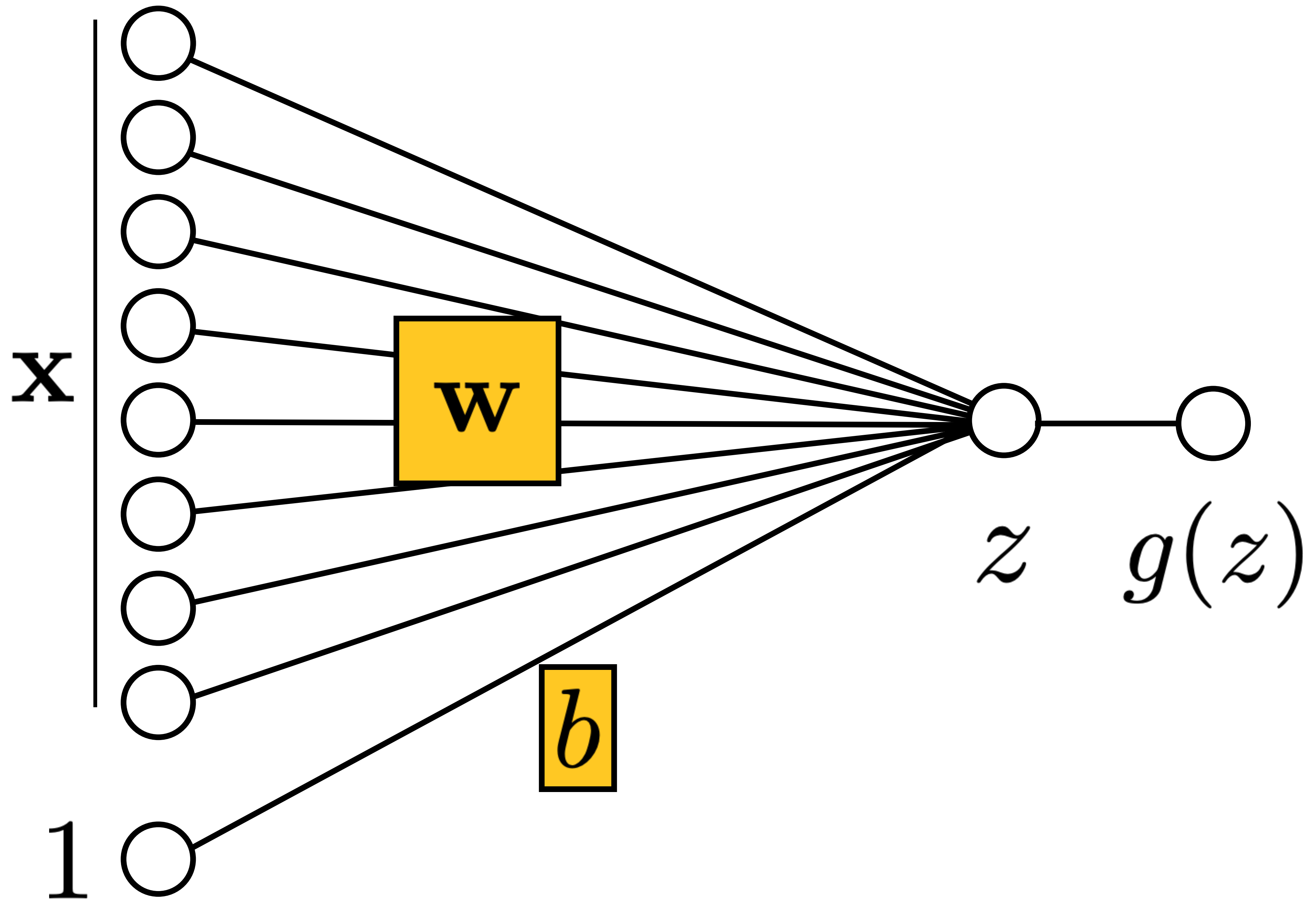
Non-differentiable non-linearity



$$g(z) = \begin{cases} 1, & \text{if} \quad z > 0 \\ 0, & \text{otherwise} \end{cases}$$

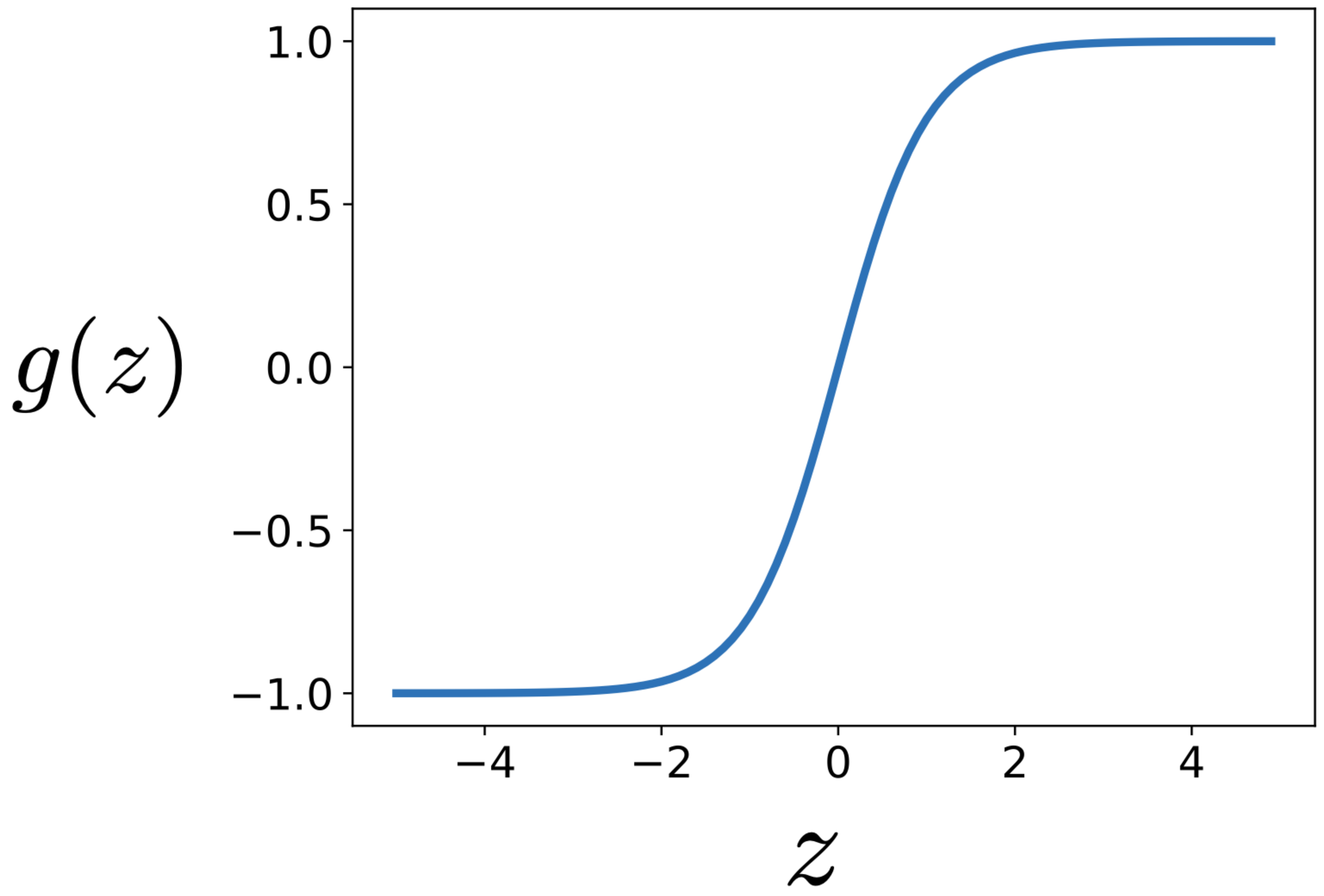


Non-linearity with soft activation



Tanh

$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

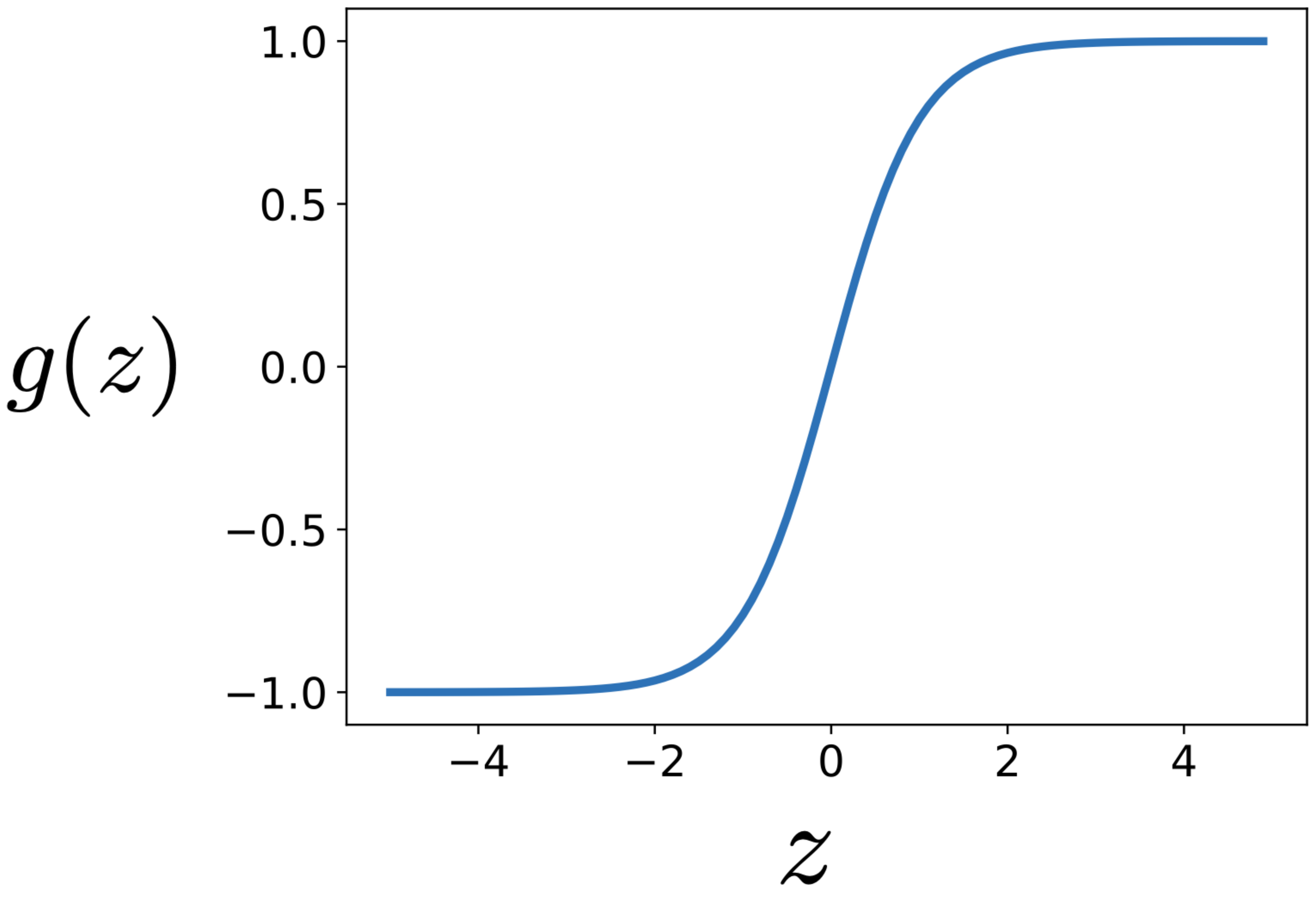


Computation in a neural net — non-linearity

- Bounded between [-1,+1]
- Saturation for large +/- inputs
- Gradients go to zero (vanishing gradients)
- Outputs centered at 0
- tanh(z) = 2 sigmoid(2z) –1
- Derivative of tanh: $1 - \tanh(z)^2$

Tanh

$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

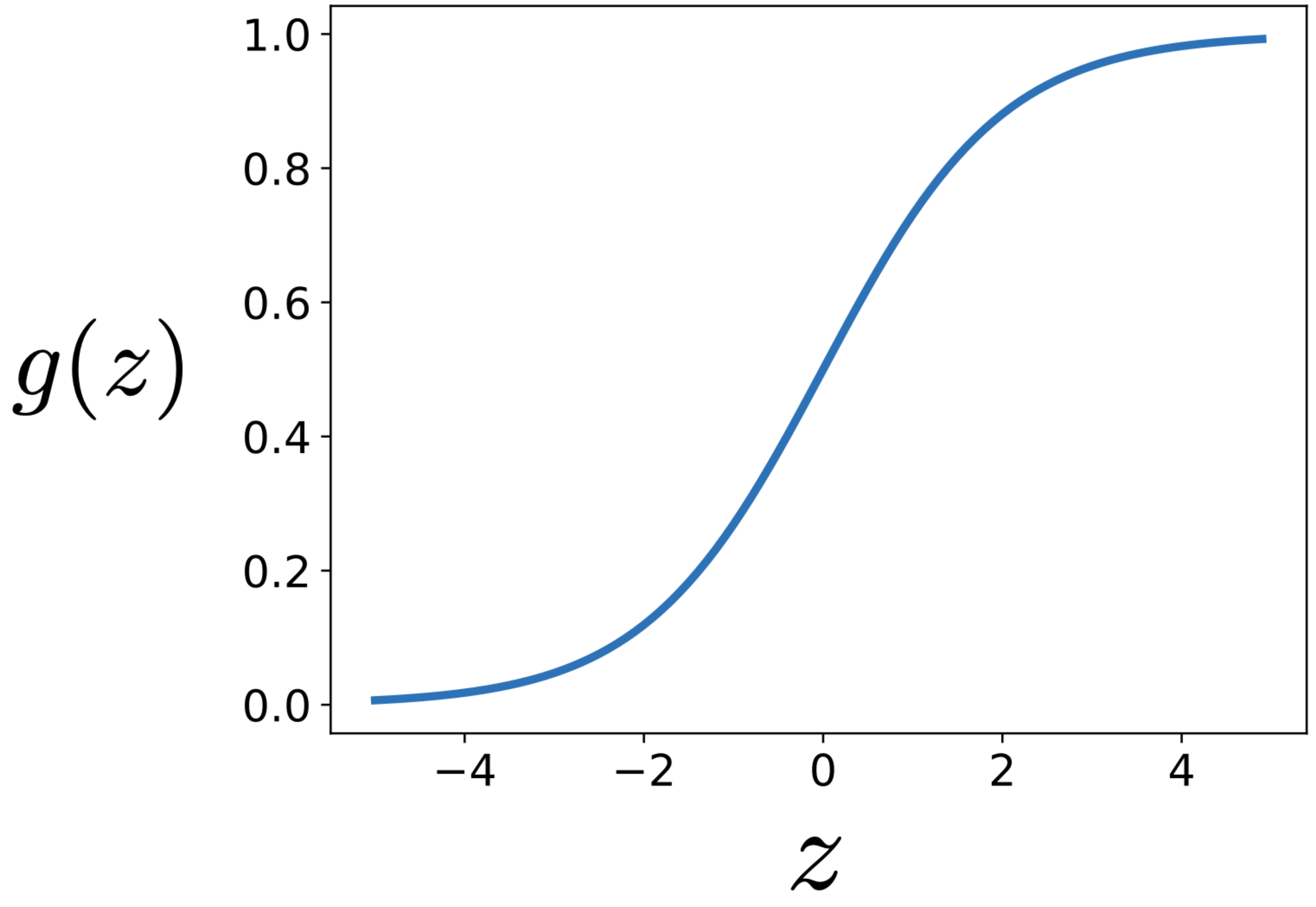


Computation in a neural net — non-linearity

- Interpretation as firing rate of neuron
- Bounded between [0,1]
- Saturation for large +/- inputs
- Gradients go to zero
- Outputs centered at 0.5
  (poor conditioning)
- Not used in practice

Sigmoid

$$g(z) = \frac{1}{1 + e^{-h}}$$

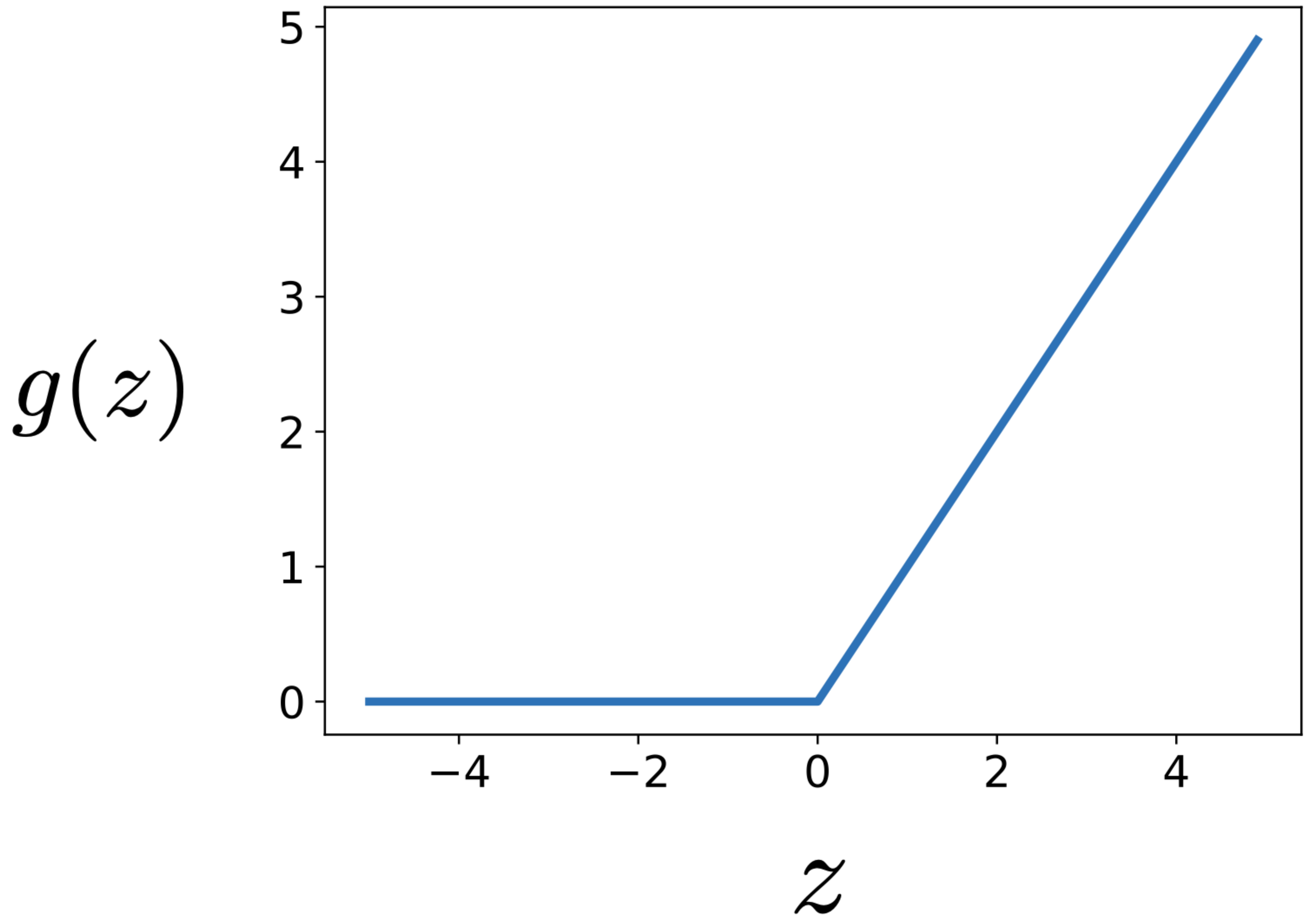


Computation in a neural net — non-linearity

- Unbounded output (on positive side)
- Efficient to implement: $\frac{\partial g}{\partial z} = \begin{cases} 0, & \text{if} \quad z < 0 \\ 1, & \text{if} \quad z \geq 0 \end{cases}$
- Also seems to help convergence (see 6x speedup vs tanh in [Krizhevsky et al.])
- Drawback: if strongly in negative region, unit is dead forever (no gradient).
- Default choice: widely used in current models.

Rectified linear unit (ReLU)

$$g(z) = \max(0, z)$$

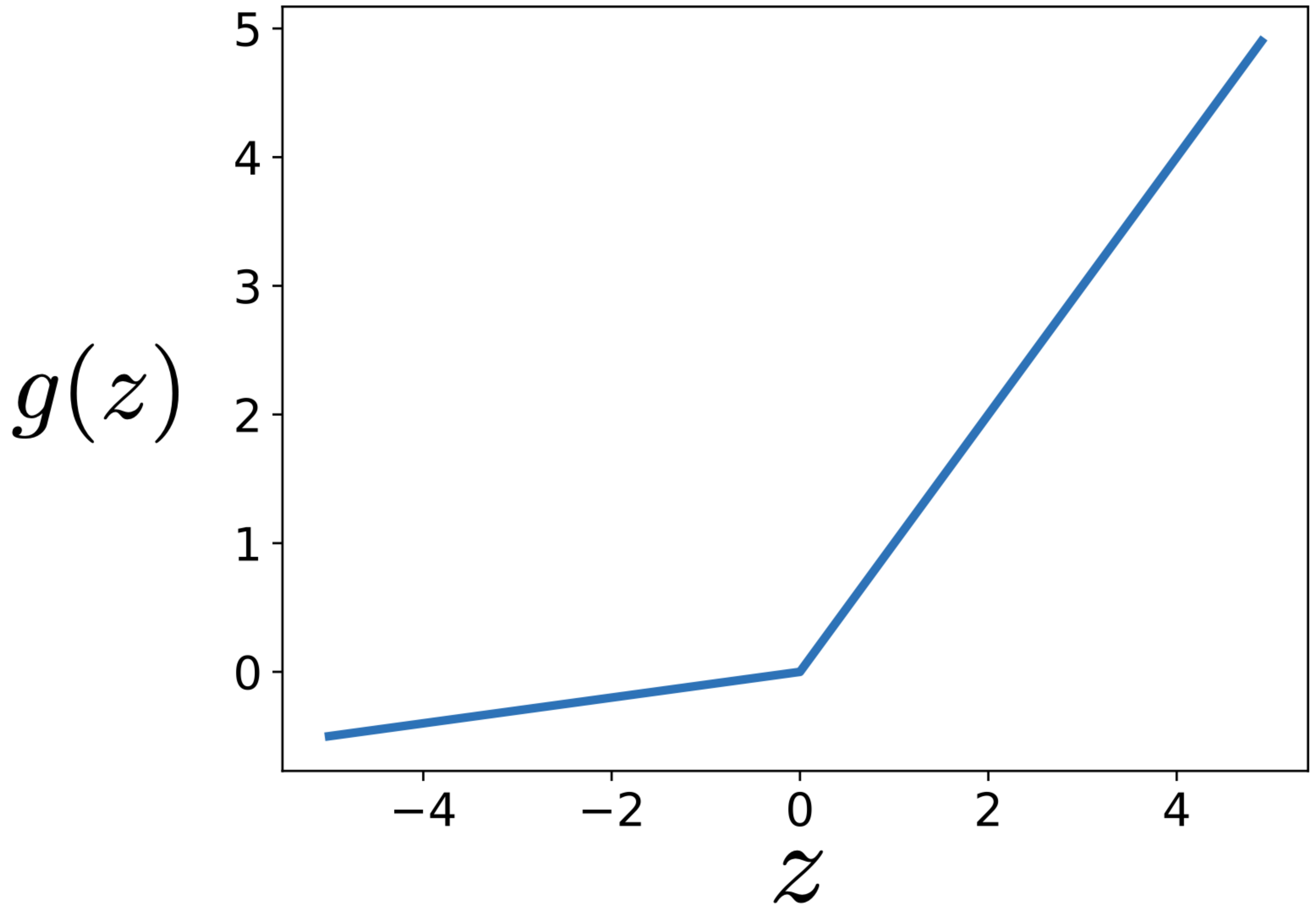


Computation in a neural net — non-linearity

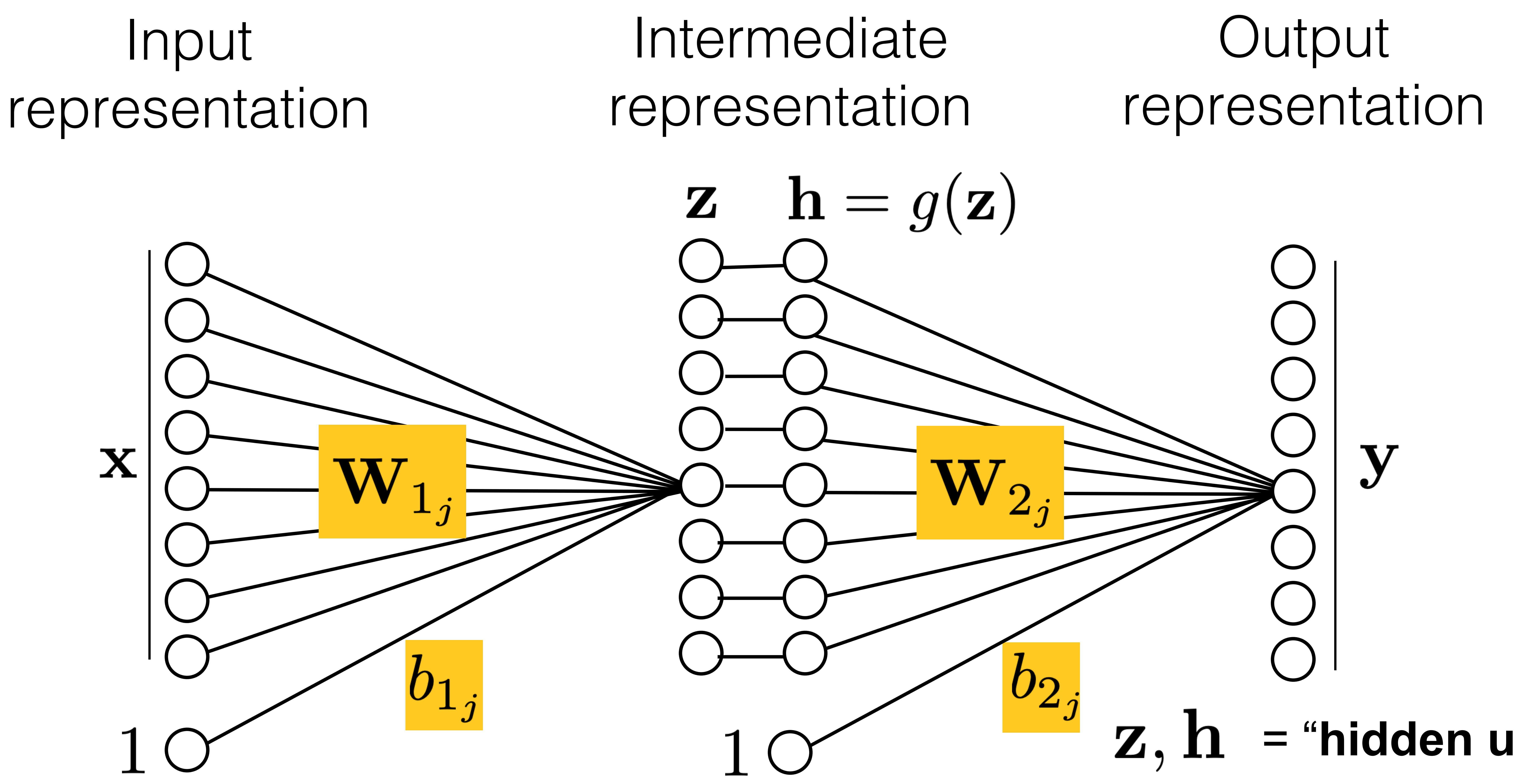
- where α is small (e.g. 0.02)
- Efficient to implement: $\frac{\partial g}{\partial z} = \begin{cases} -a, & \text{if} \quad z < 0 \\ 1, & \text{if} \quad z \geq 0 \end{cases}$
- Also known as probabilistic ReLU (PReLU)
- Has non-zero gradients everywhere (unlike ReLU)
- α can also be learned (see Kaiming He et al. 2015).

Leaky ReLU

$$g(z) = \begin{cases} \max(0, z), & \text{if} \quad z \geq 0 \\ a \min(0, z), & \text{if} \quad z < 0 \end{cases}$$



Stacking layers - Multi-layer Perceptron (MLP)

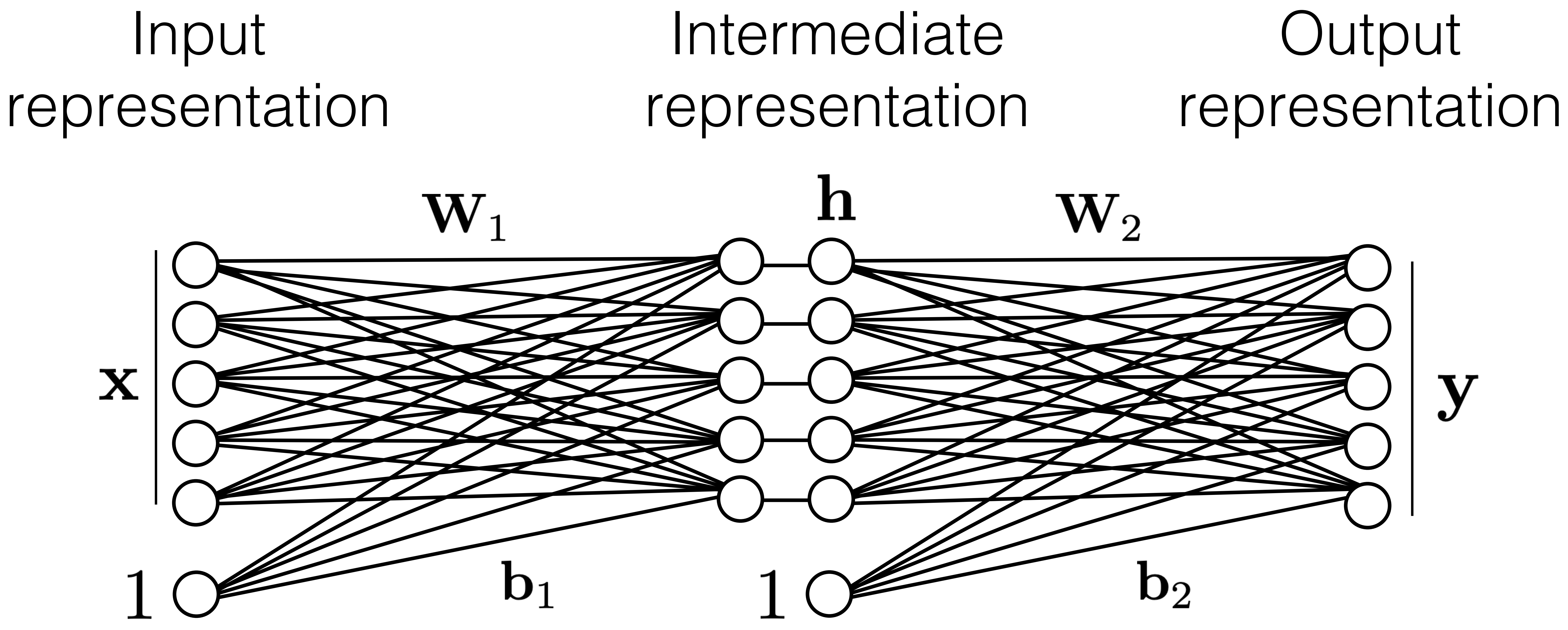


$\mathbf{z}, \mathbf{h}$ = **"hidden units"**

$\mathbf{z}$ = **"pre-activation hidden layer"**

$\mathbf{h}$ = **"post-activation hidden layer"**

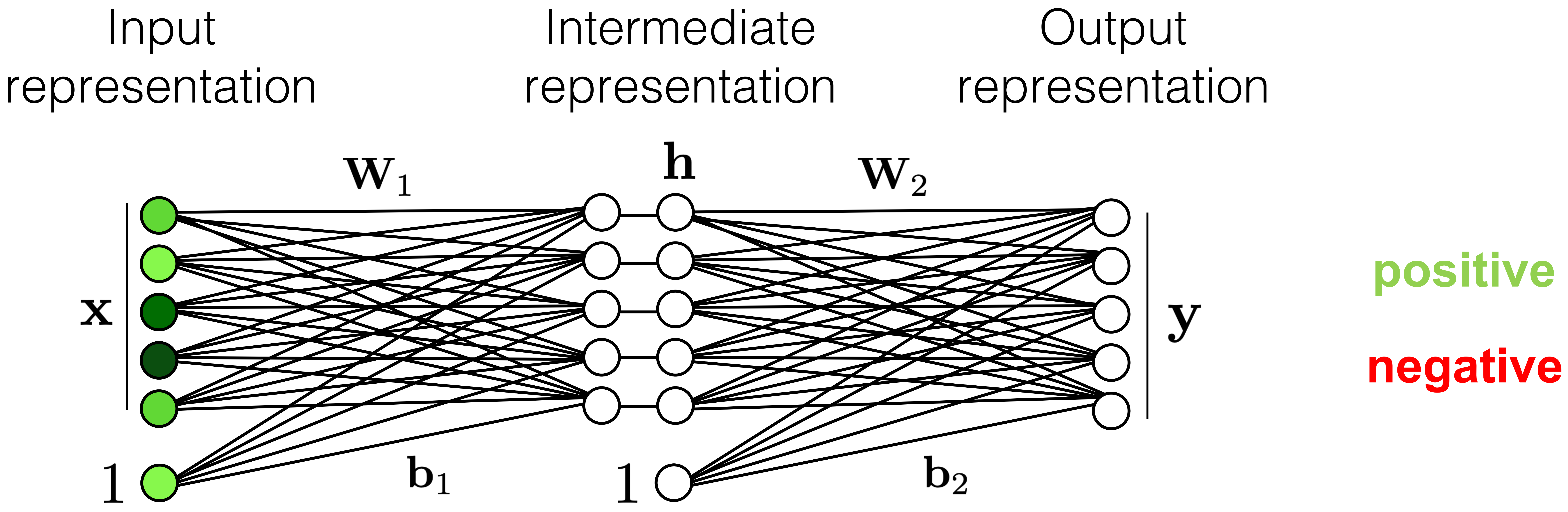
Stacking layers - fully connected layers



$$\mathbf{h} = g(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \qquad \mathbf{y} = g(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2)$$

$$\theta = \{\mathbf{W}_1, \ldots, \mathbf{W}_L, \mathbf{b}_1, \ldots, \mathbf{b}_L\}$$

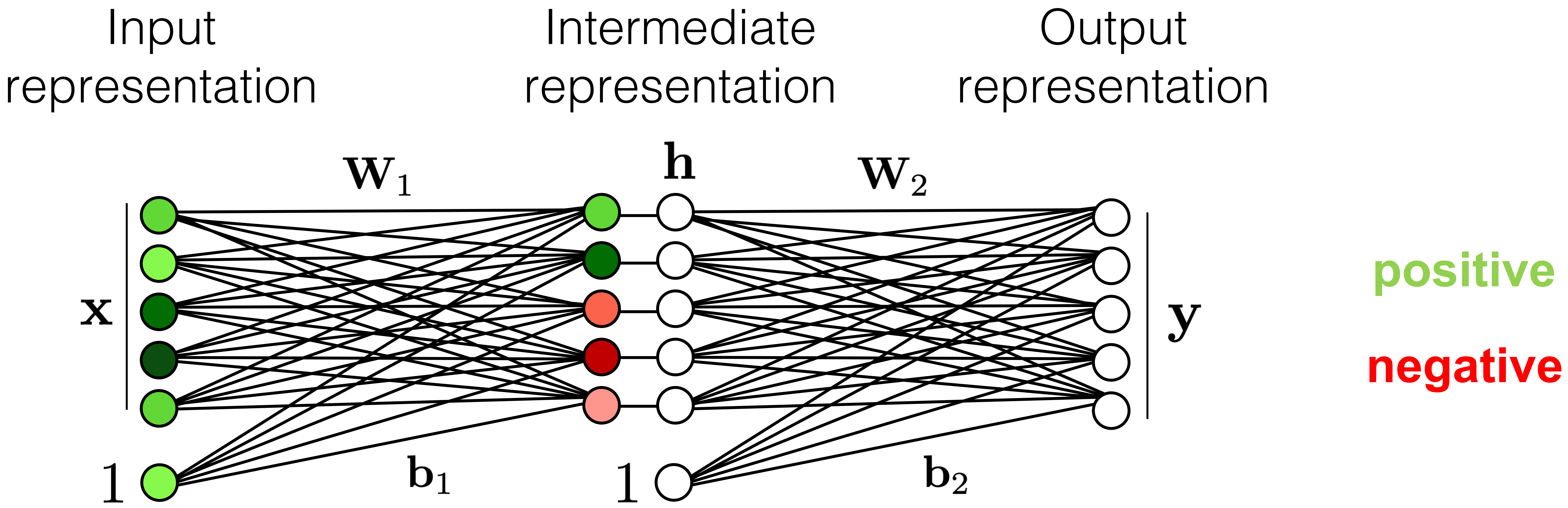
Example: how signal evolves



$$\mathbf{h} = g(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \qquad \mathbf{y} = g(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2)$$

$$\theta = \{\mathbf{W}_1, \ldots, \mathbf{W}_L, \mathbf{b}_1, \ldots, \mathbf{b}_L\}$$

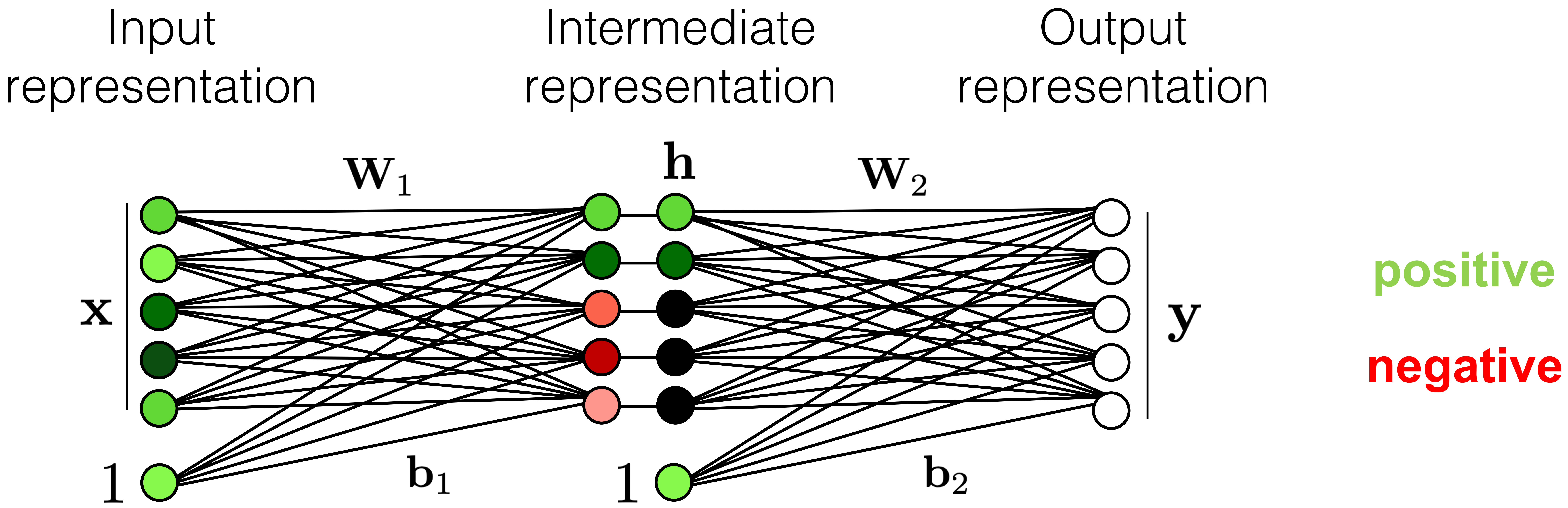
Example: how signal evolves



$$\mathbf{h} = g(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \qquad \mathbf{y} = g(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2)$$

$$\theta = \{\mathbf{W}_1, \ldots, \mathbf{W}_L, \mathbf{b}_1, \ldots, \mathbf{b}_L\}$$

Example: how signal evolves



$$\mathbf{h} = g(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \qquad \mathbf{y} = g(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2)$$

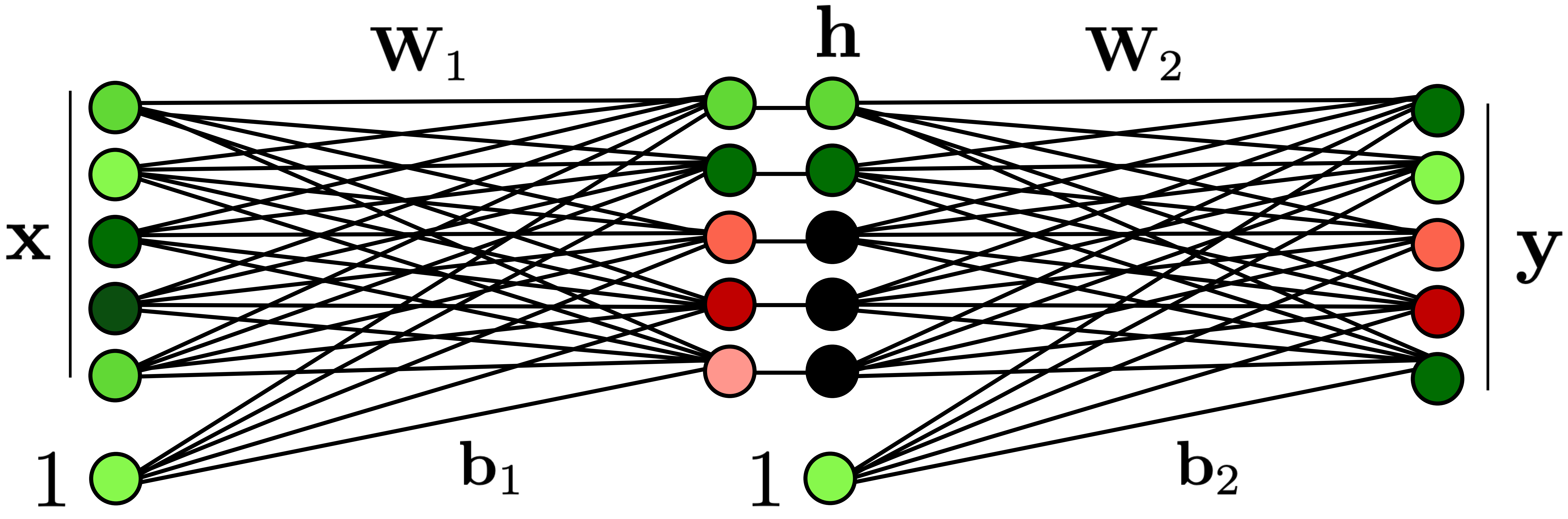
$$\theta = \{\mathbf{W}_1, \ldots, \mathbf{W}_L, \mathbf{b}_1, \ldots, \mathbf{b}_L\}$$

Example: how signal evolves

Input representation

Intermediate representation

Output representation



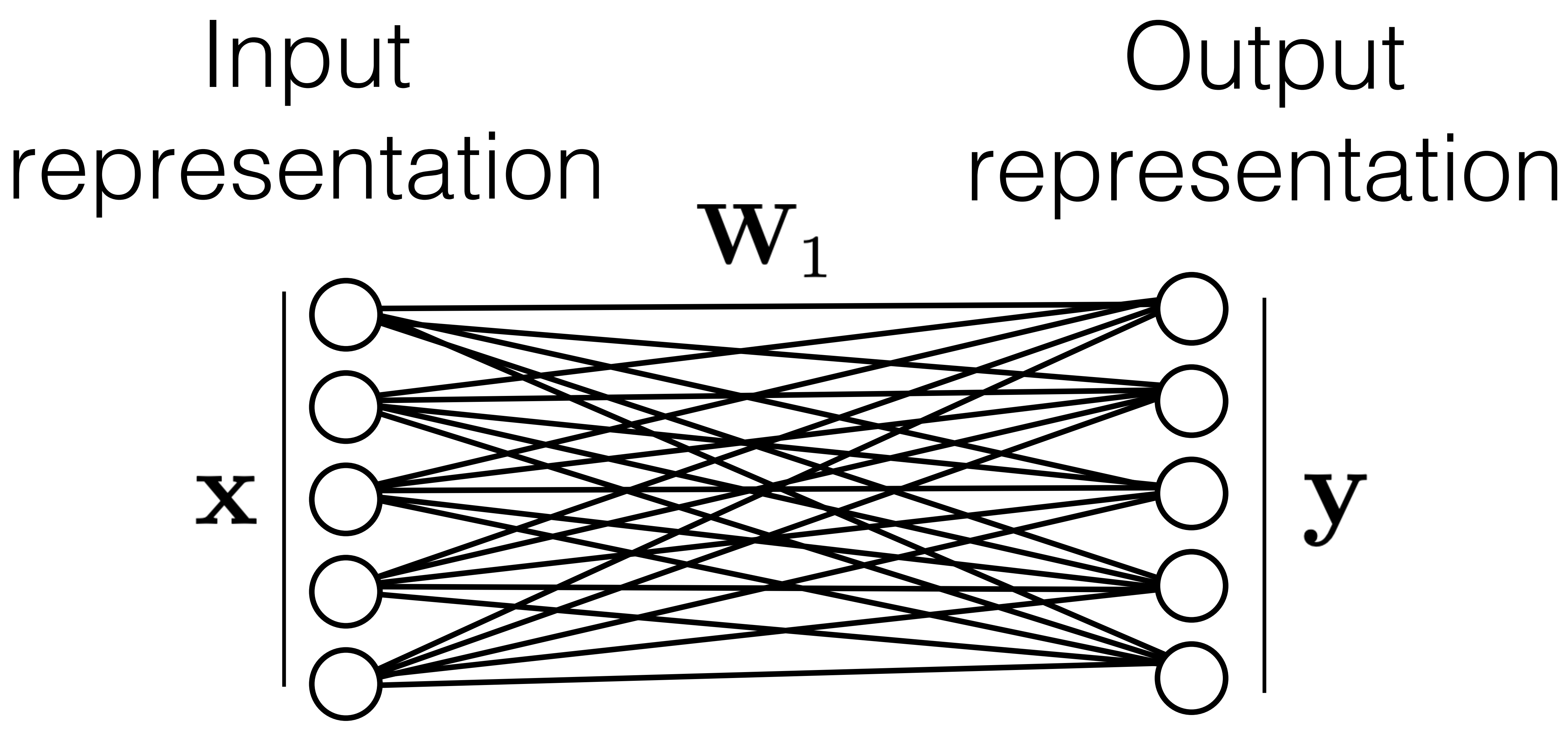
positive

negative

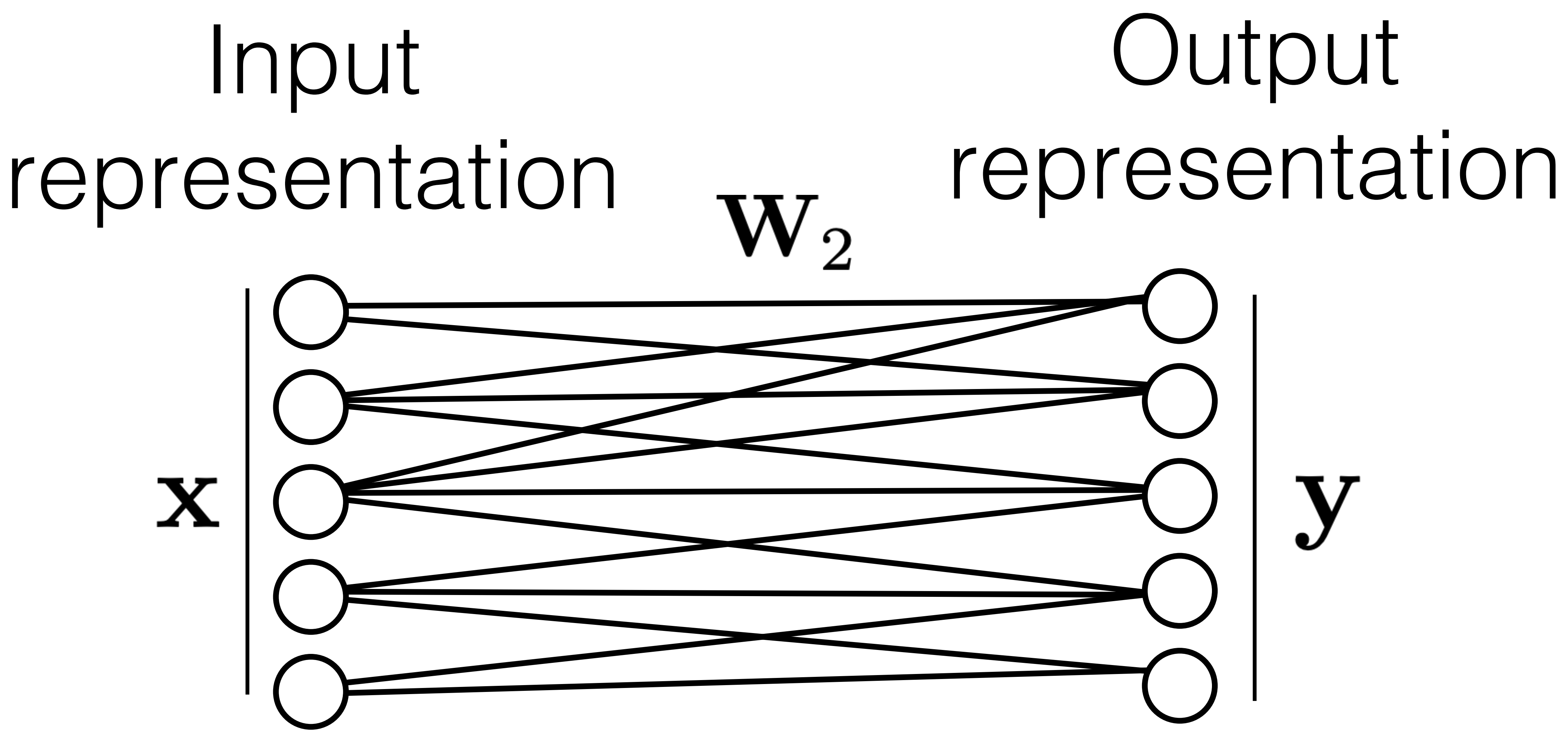
$$\mathbf{h} = g(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \qquad \mathbf{y} = g(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2)$$

$$\theta = \{\mathbf{W}_1, \ldots, \mathbf{W}_L, \mathbf{b}_1, \ldots, \mathbf{b}_L\}$$

Connectivity patterns

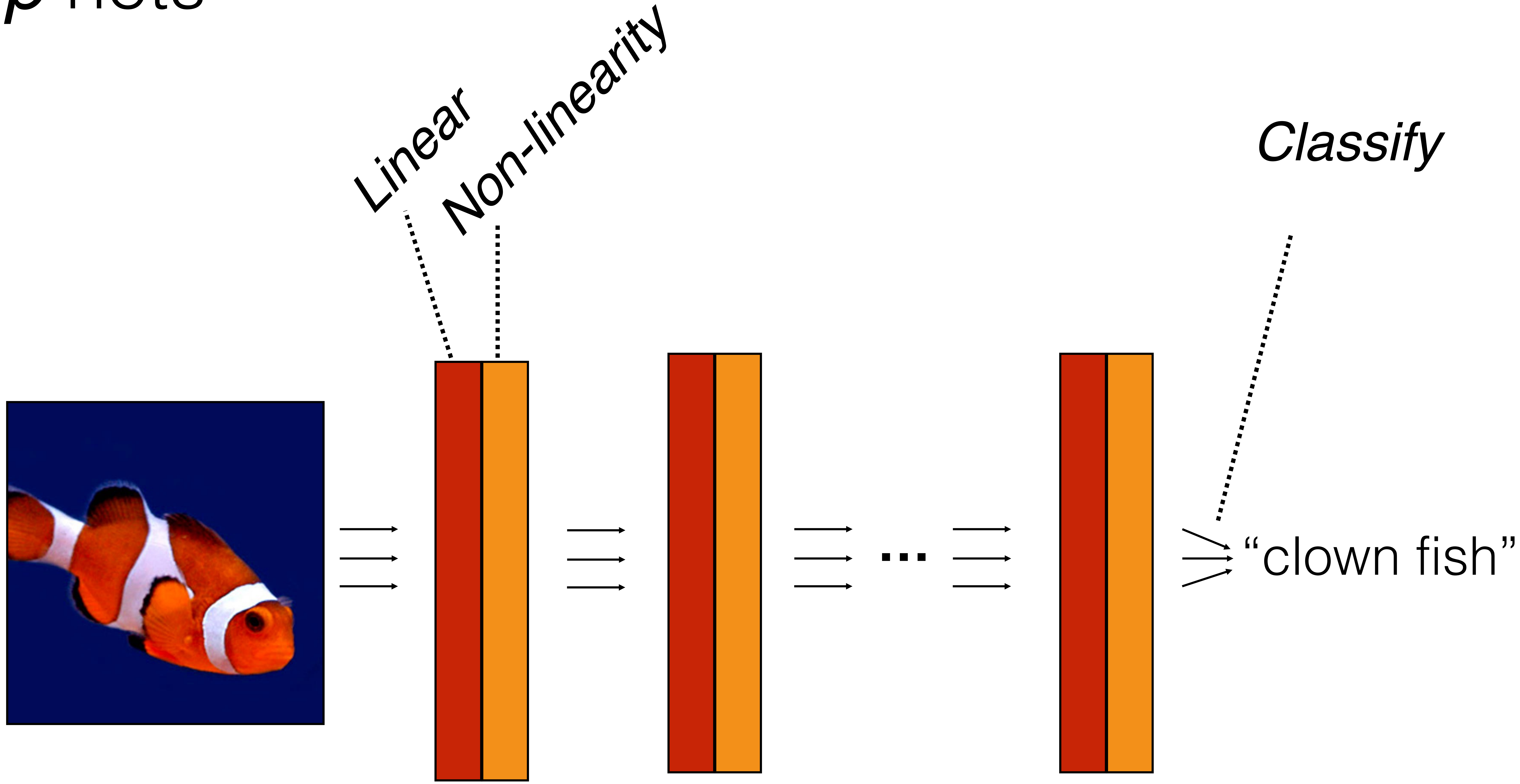


Fully connected layer



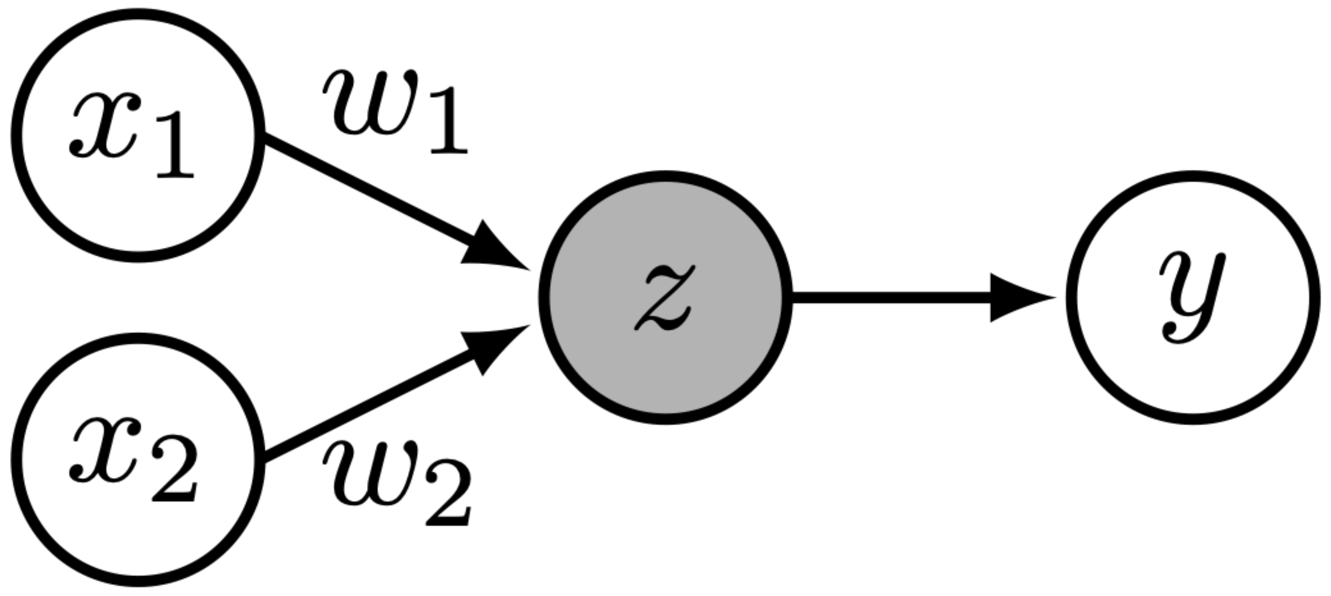
Locally connected layer
(Sparse W)

Deep nets



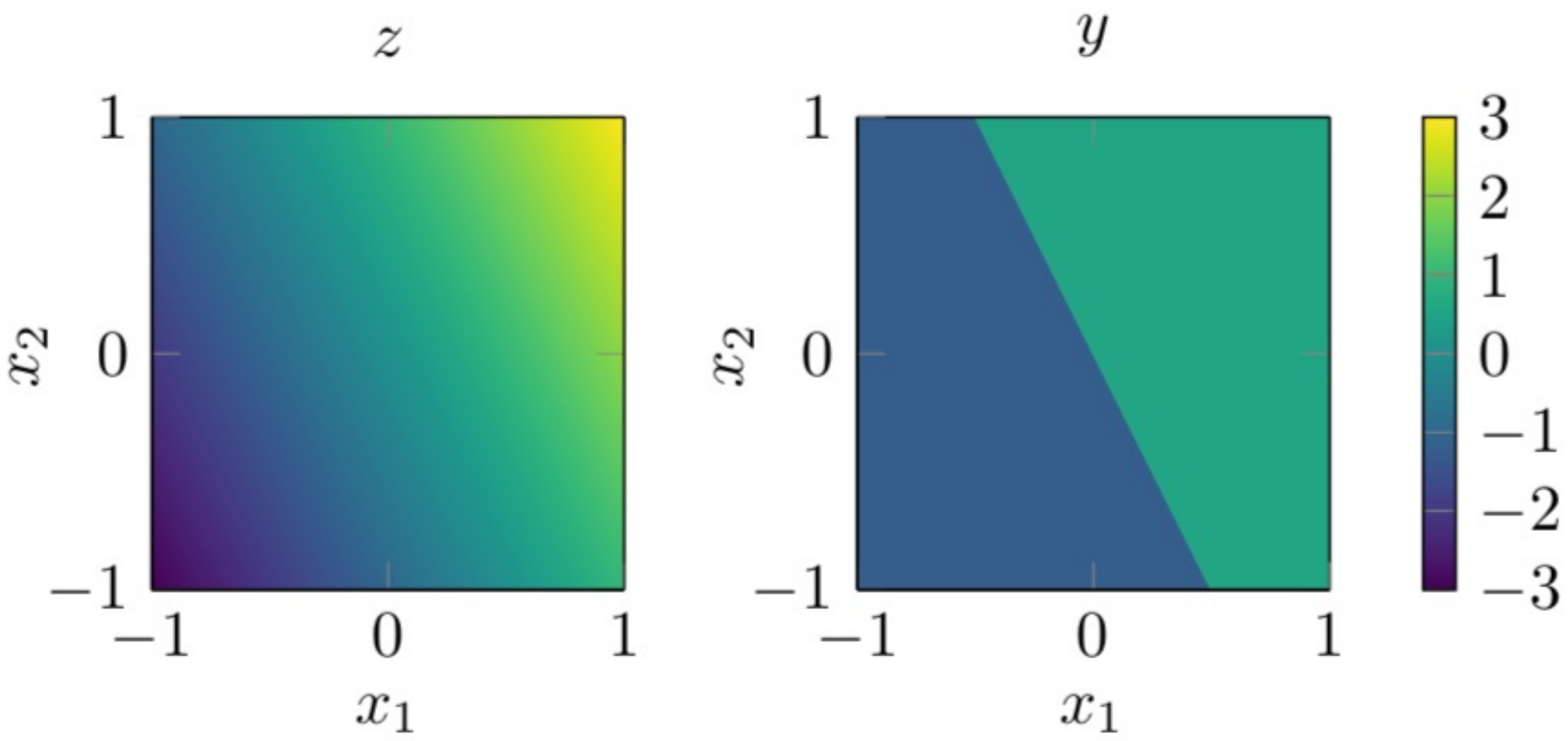
$$f(\mathbf{x}) = f_L(f_{L-1}(\dots f_2(f_1(\mathbf{x}))))$$

Example: linear classification with a perceptron



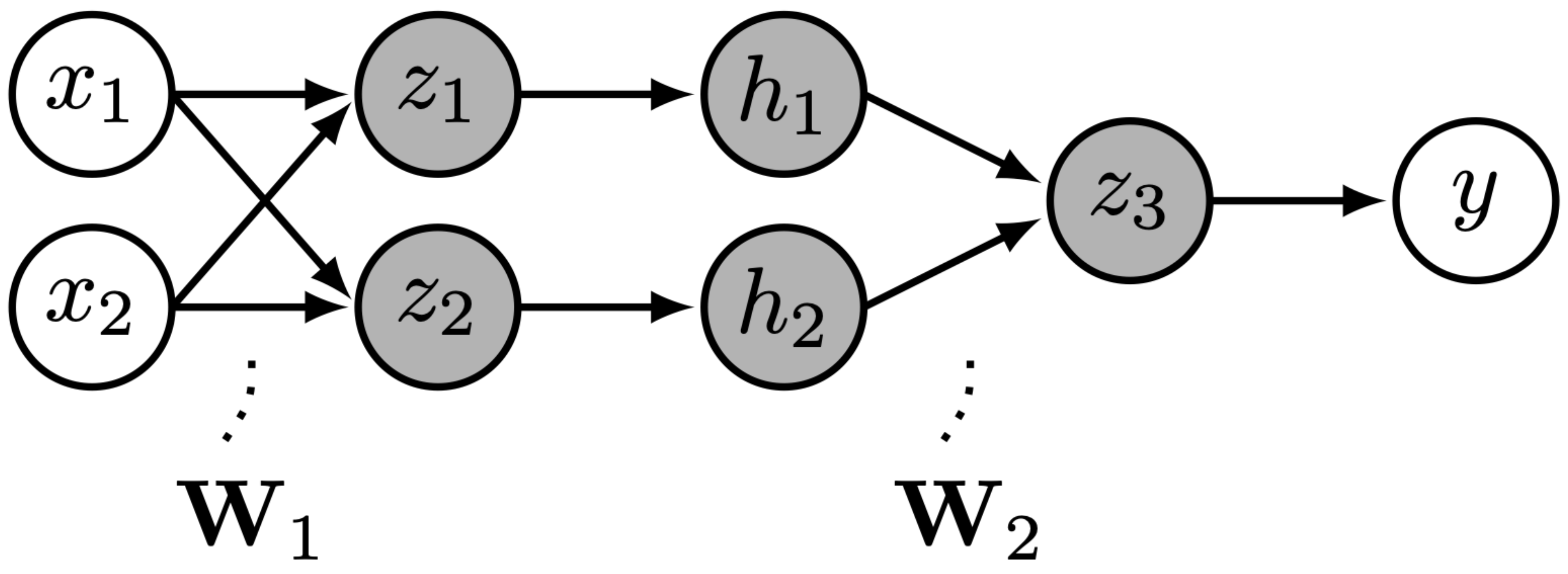
$$z = \mathbf{x}^T \mathbf{w} + b$$

$$y = g(z)$$

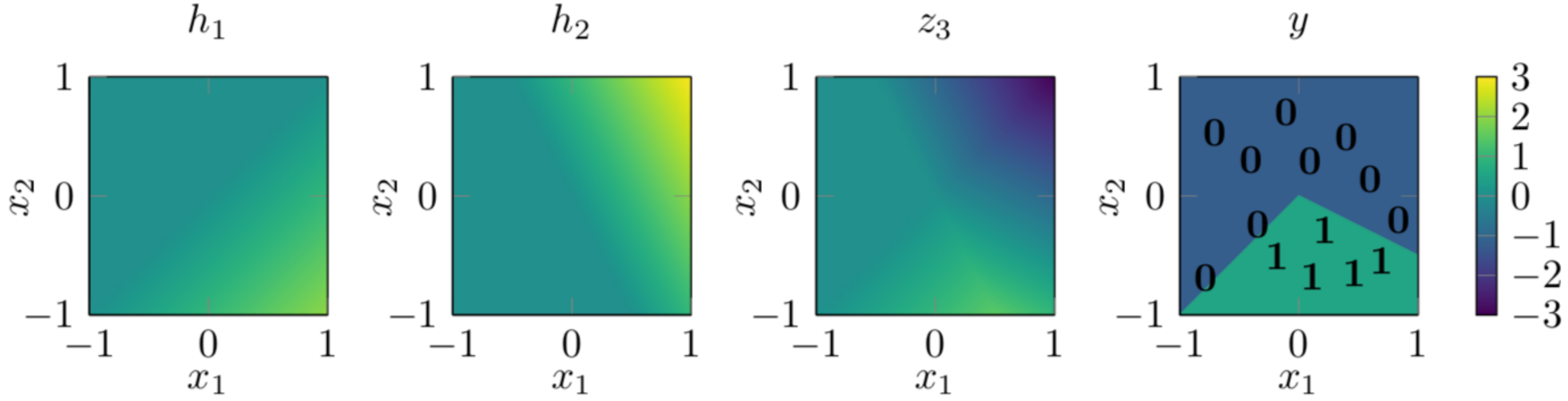


One layer neural net (perceptron) can perform linear classification!

Example: nonlinear classification with a deep net net



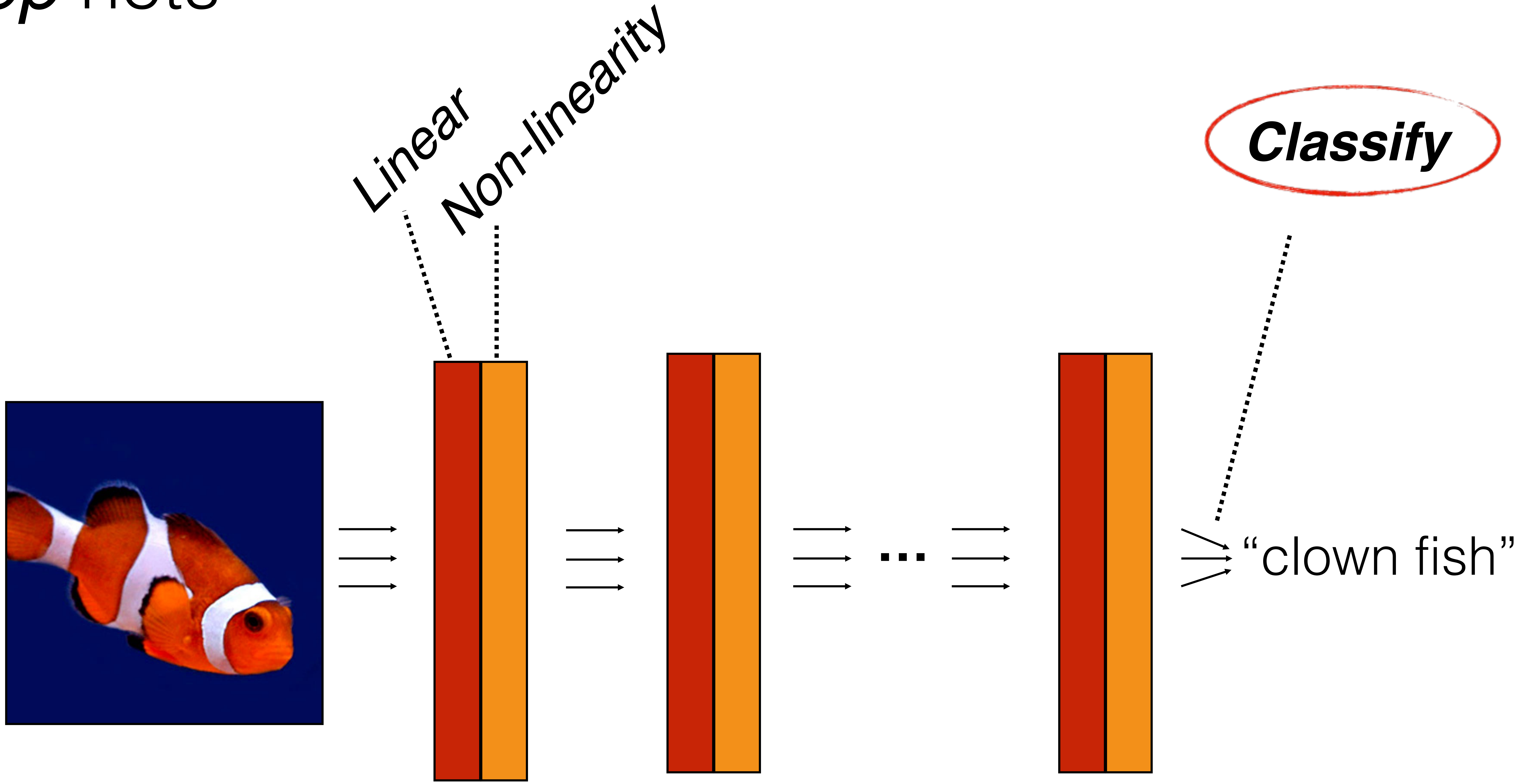
$$
\begin{aligned}
\mathbf{z} &= \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1 \\
\mathbf{h} &= g(\mathbf{z}) \\
z_3 &= \mathbf{W}_2 \mathbf{h} + b_2 \\
y &= 1(z_3 > 0)
\end{aligned}
$$



Representational power

- 1 layer? Linear decision surface.
- 2+ layers? In theory, can represent any function. Assuming non-trivial non-linearity.
  - Bengio 2009, http://www.iro.umontreal.ca/~bengioy/papers/ftml.pdf
  - Bengio, Courville, Goodfellow book http://www.deeplearningbook.org/contents/mlp.html
  - Simple proof by M. Neilsen http://neuralnetworksanddeeplearning.com/chap4.html
  - D. Mackay book http://www.inference.phy.cam.ac.uk/mackay/itprnn/ps/482.491.pdf
- But issue is efficiency: very wide two layers vs narrow deep model? In practice, more layers helps.

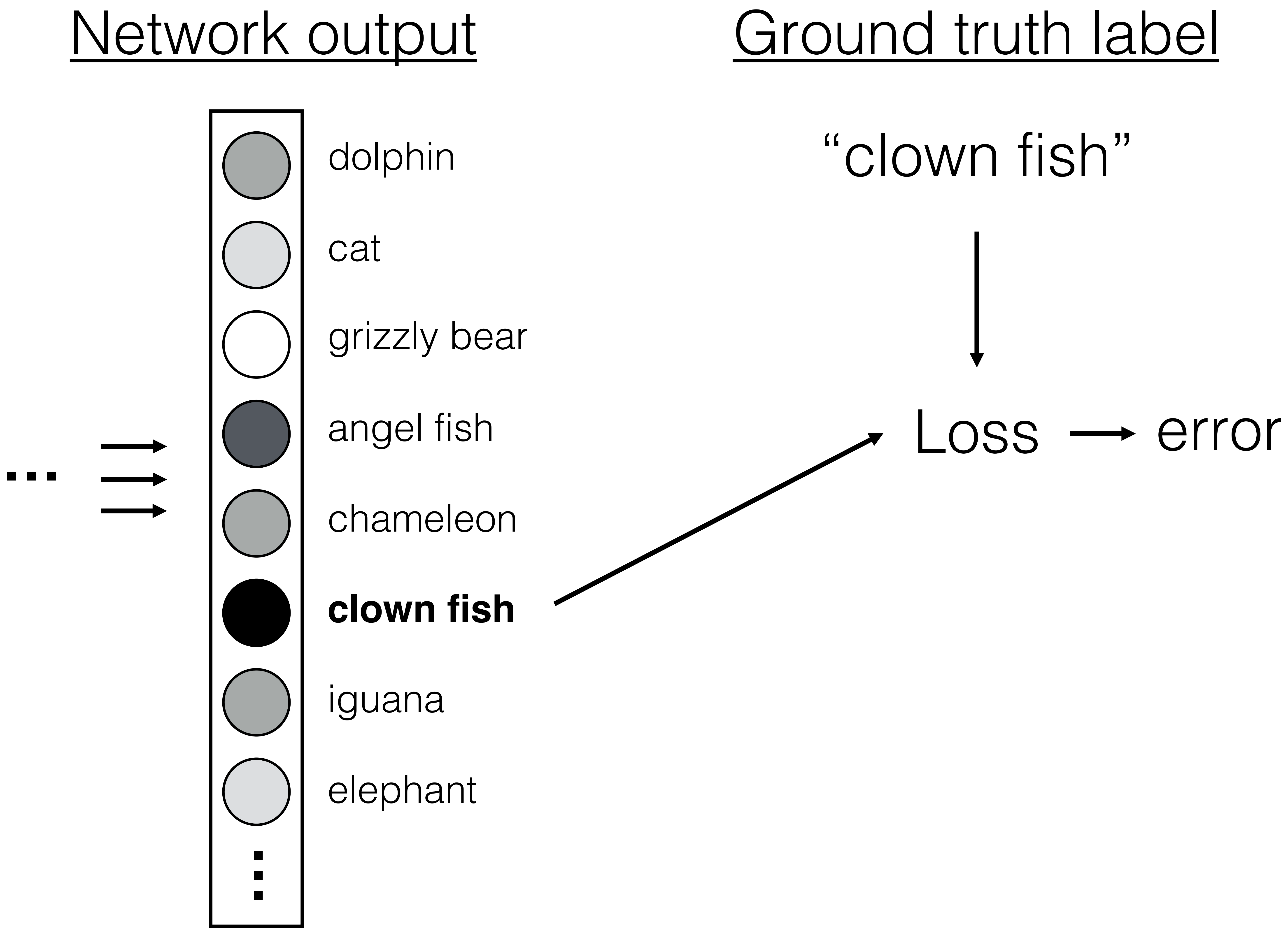
Deep nets



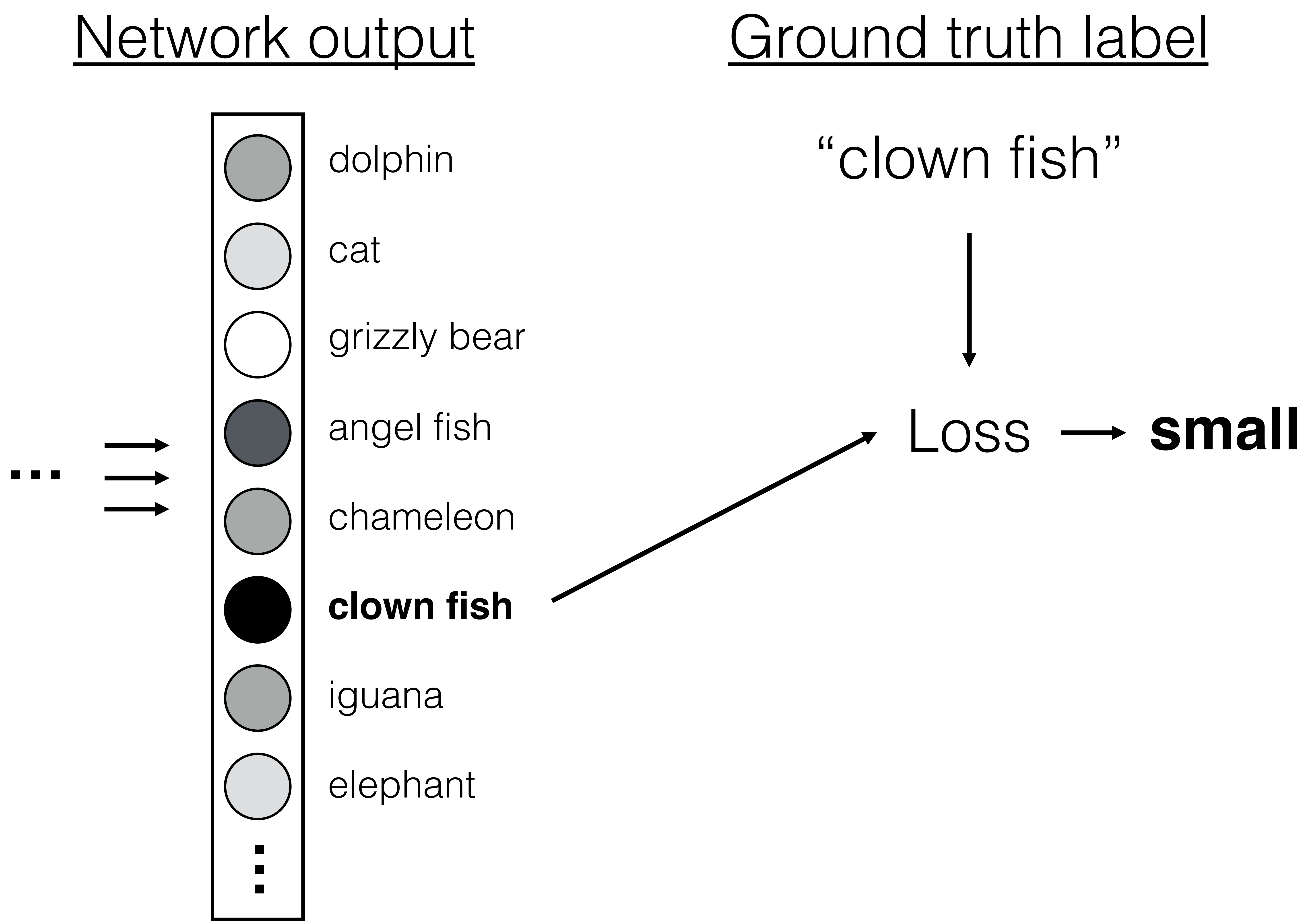
$$f(\mathbf{x}) = f_L(f_{L-1}(\dots f_2(f_1(\mathbf{x}))))$$

Classifier layer

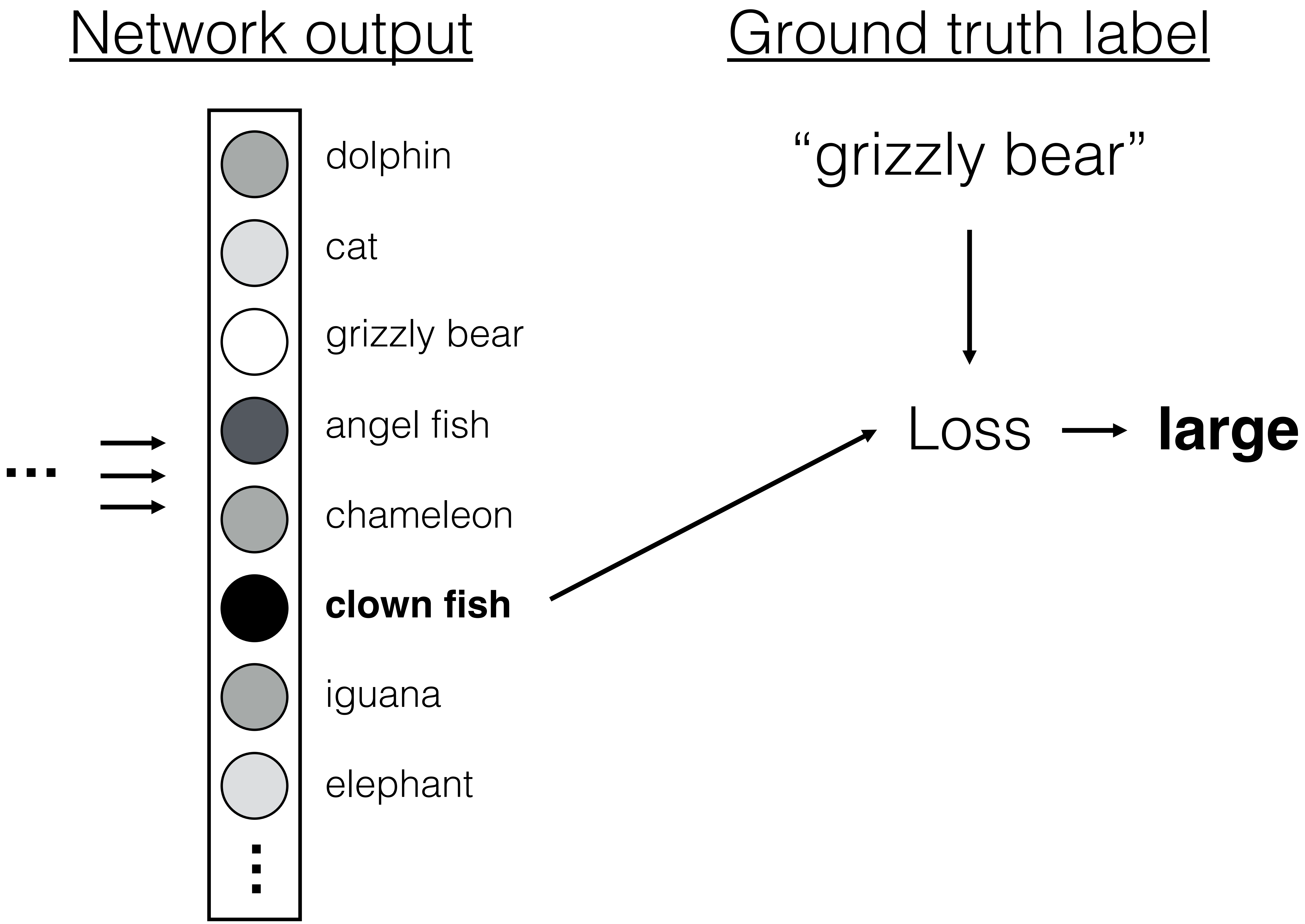
Loss function



Loss function



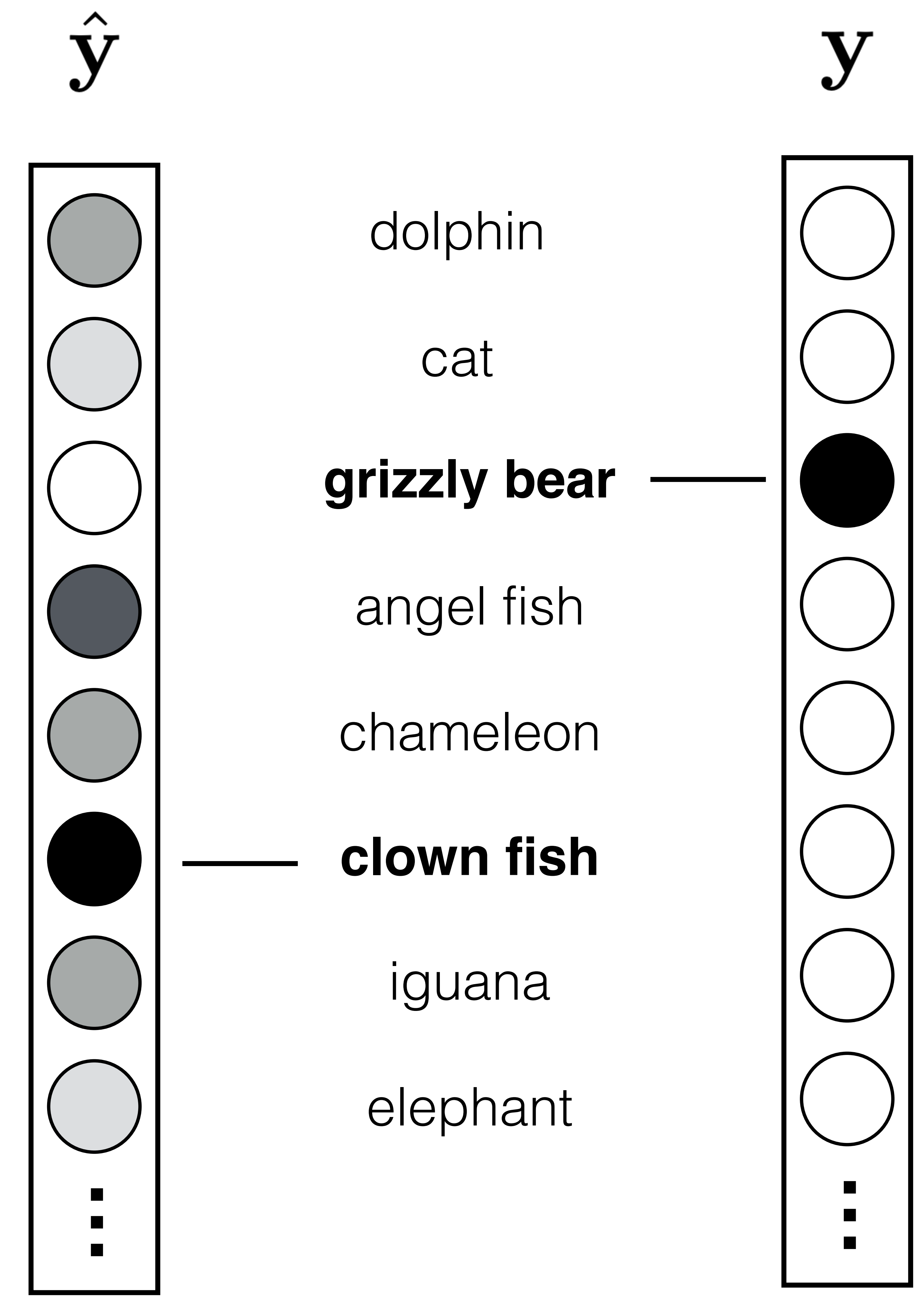
Loss function



Network output   Ground truth label

$$\text{Softmax } \sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

`softmax`



Probability of the observed data under the model

$$H(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$$

Loss Function

Intuitively Understanding the Cross Entropy

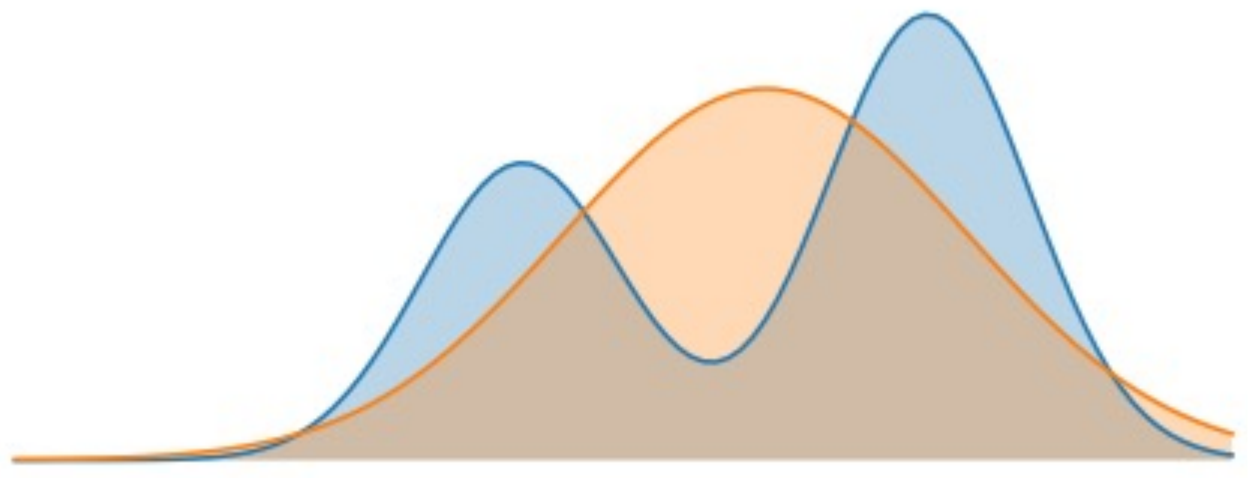
$$H(P^* | P) = -\sum_i \underbrace{P^*(i)}_{\text{TRUE CLASS DISTIRBUTION}} \log \underbrace{P(i)}_{\text{PREDICTED CLASS DISTIRBUTION}}$$

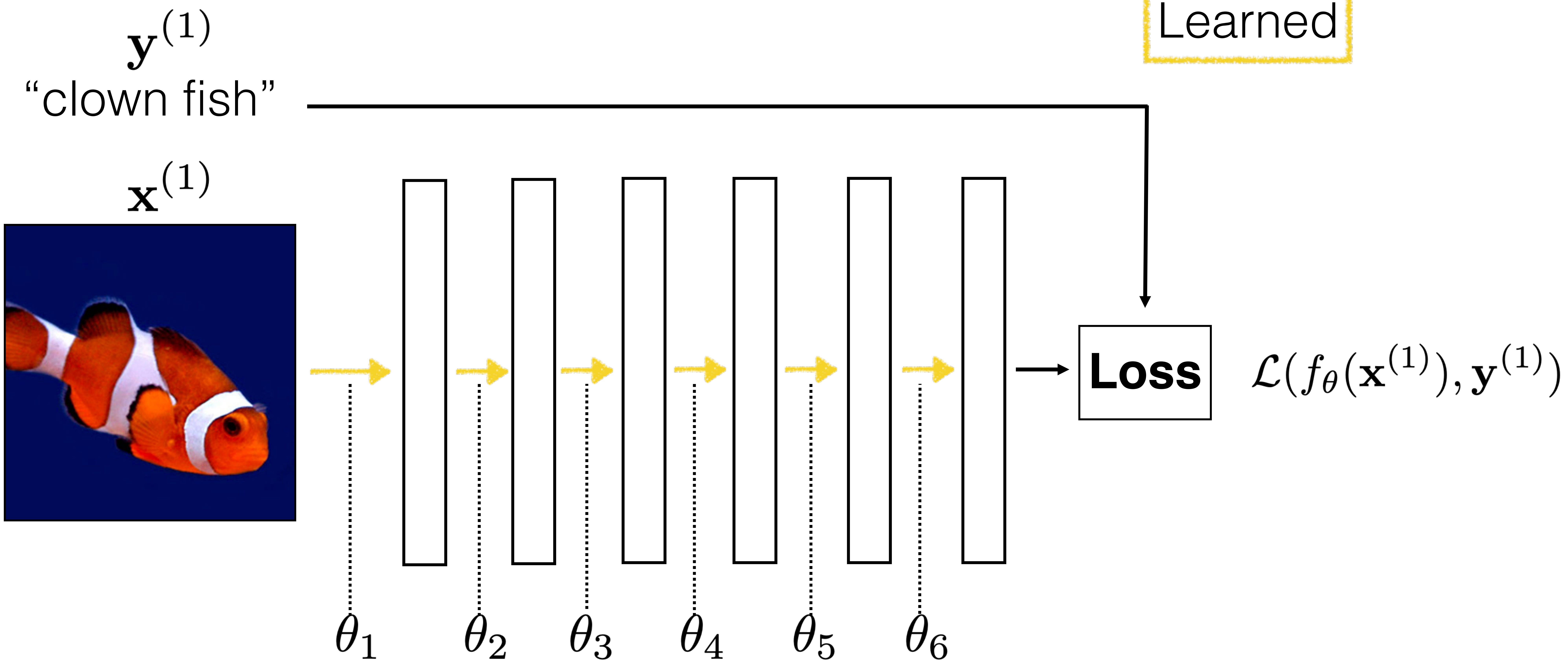
Intuitively Understanding the KL divergence

KL is not symmetric

$$D_{KL}(P||Q) = \sum_i P(i)\log\frac{P(i)}{Q(i)}$$

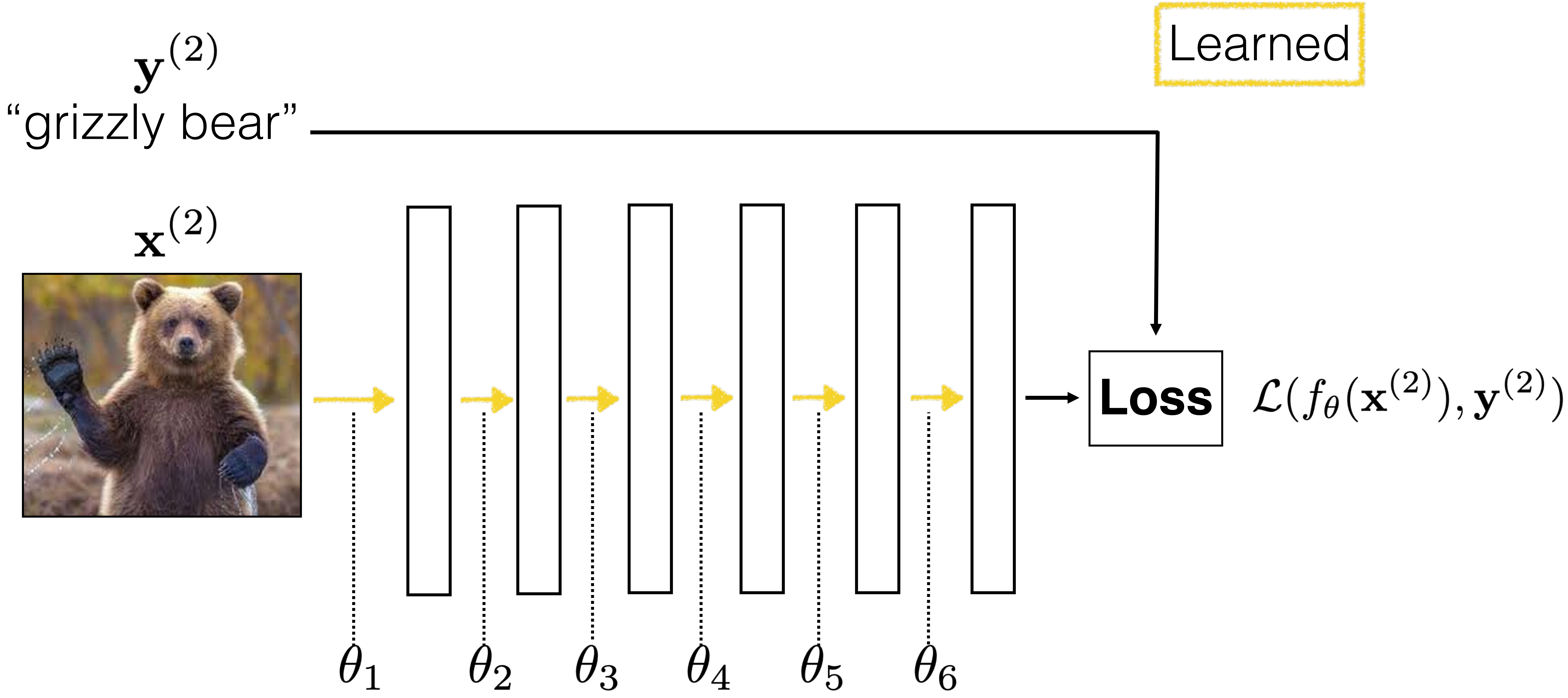
$$p_1\log\frac{p_1}{q_1} + p_2\log\frac{p_2}{q_2}$$



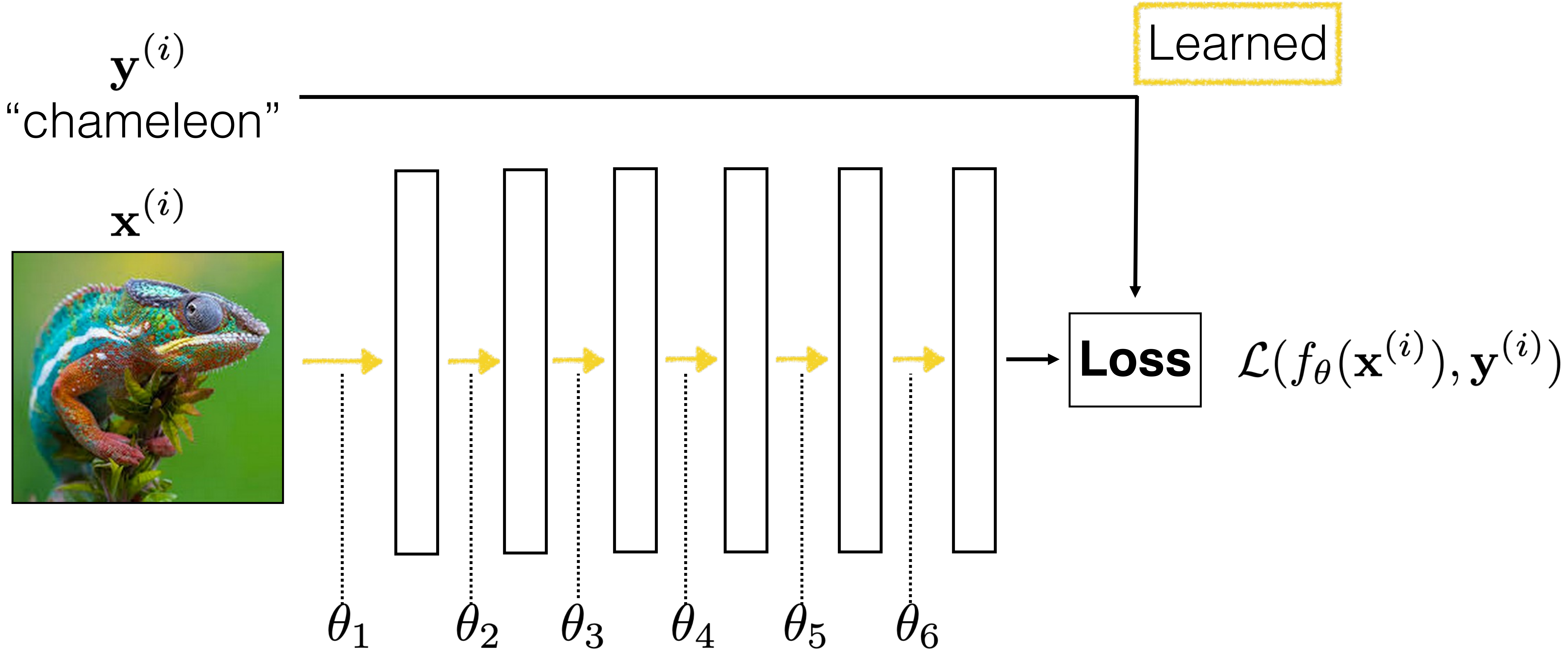


Learned

$$\theta^* = \arg\min_{\theta} \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$$

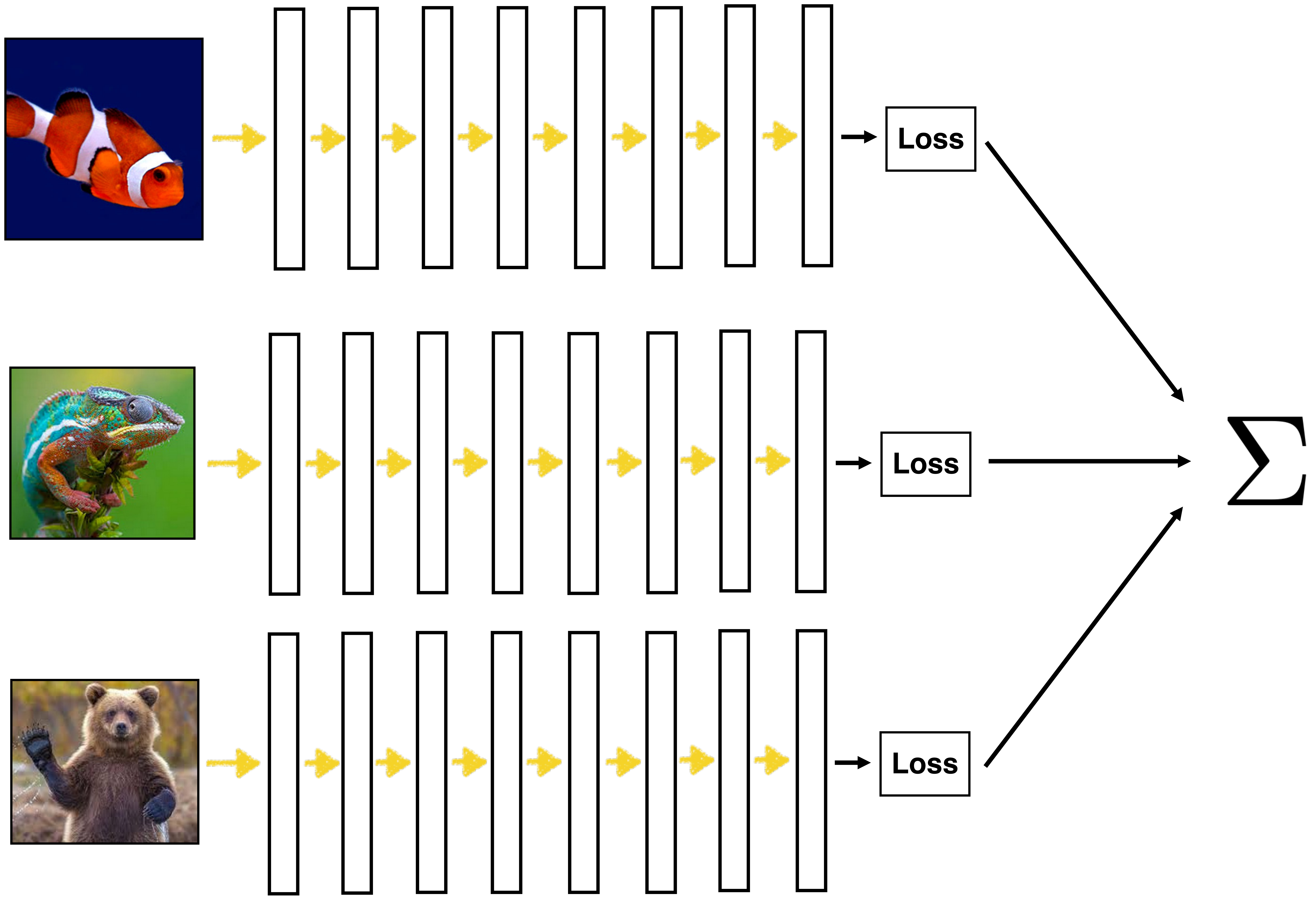


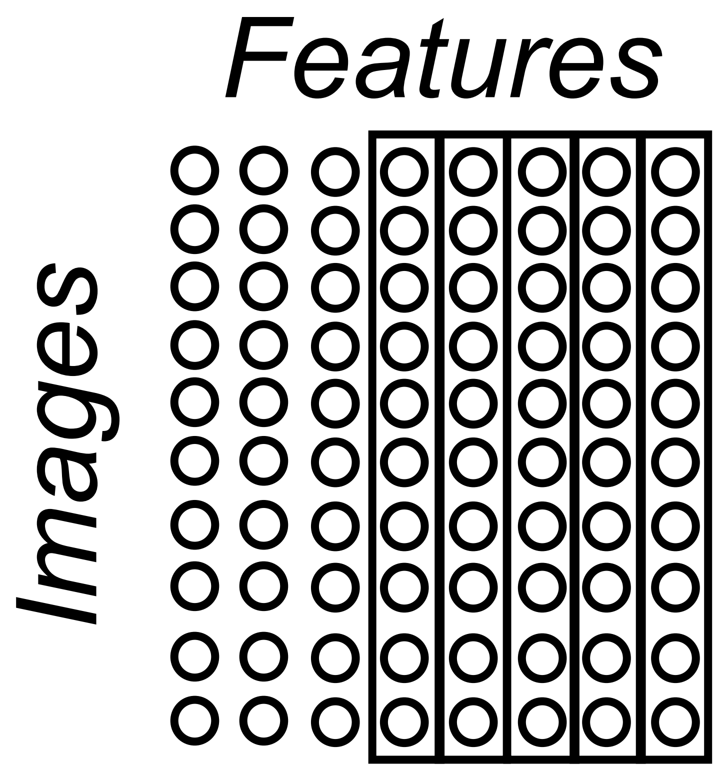
$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$$



$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$$

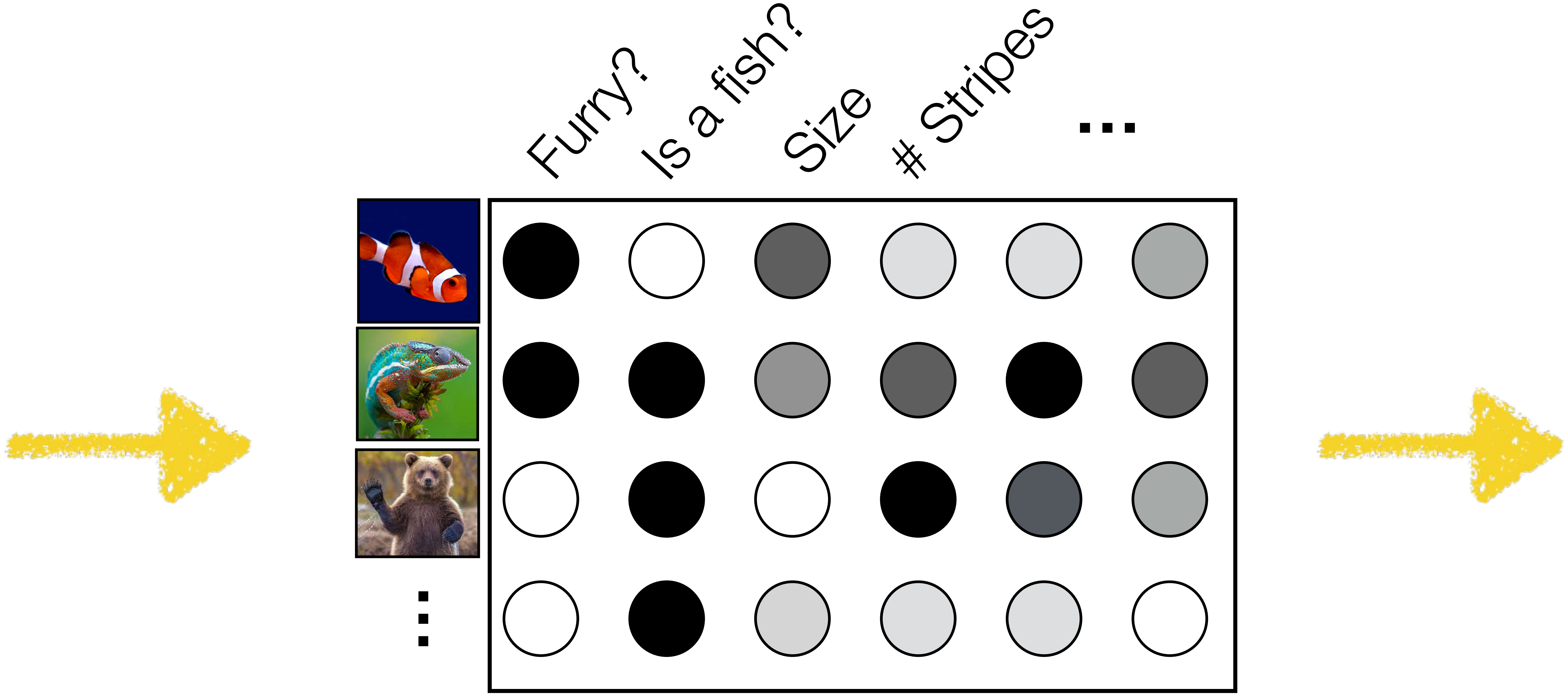
Batch (parallel) processing





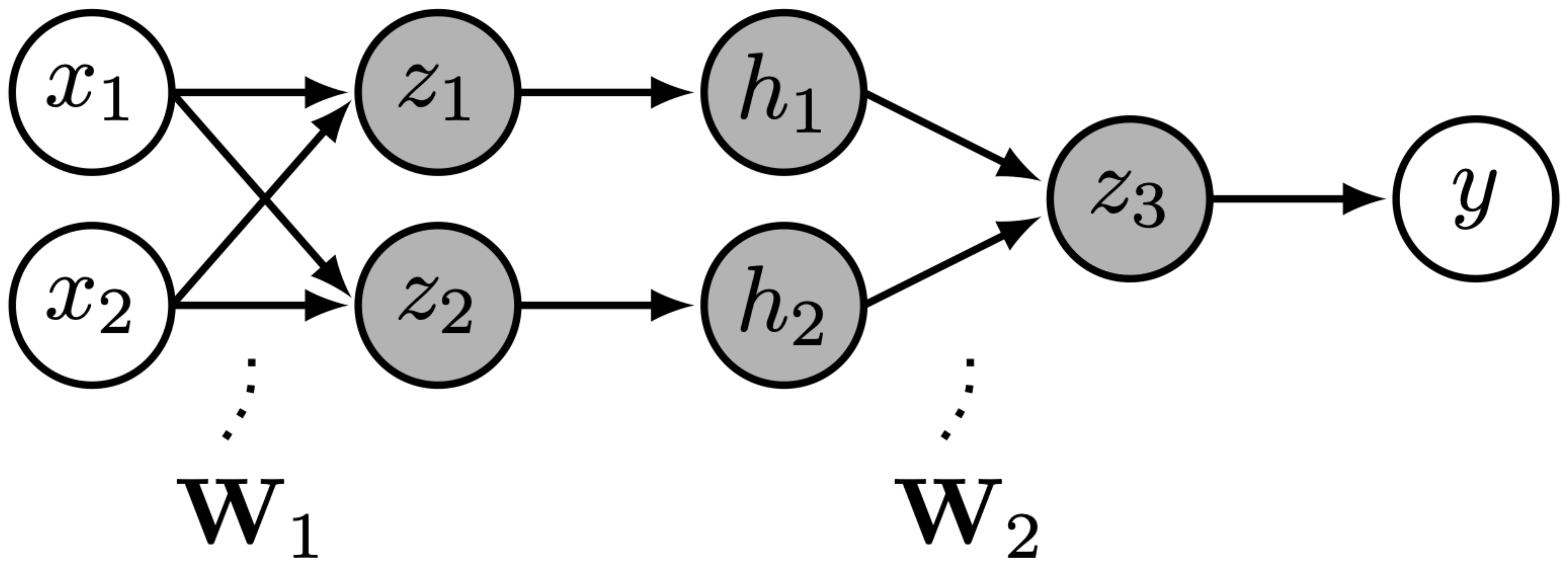
Tensors

(multi-dimensional arrays)



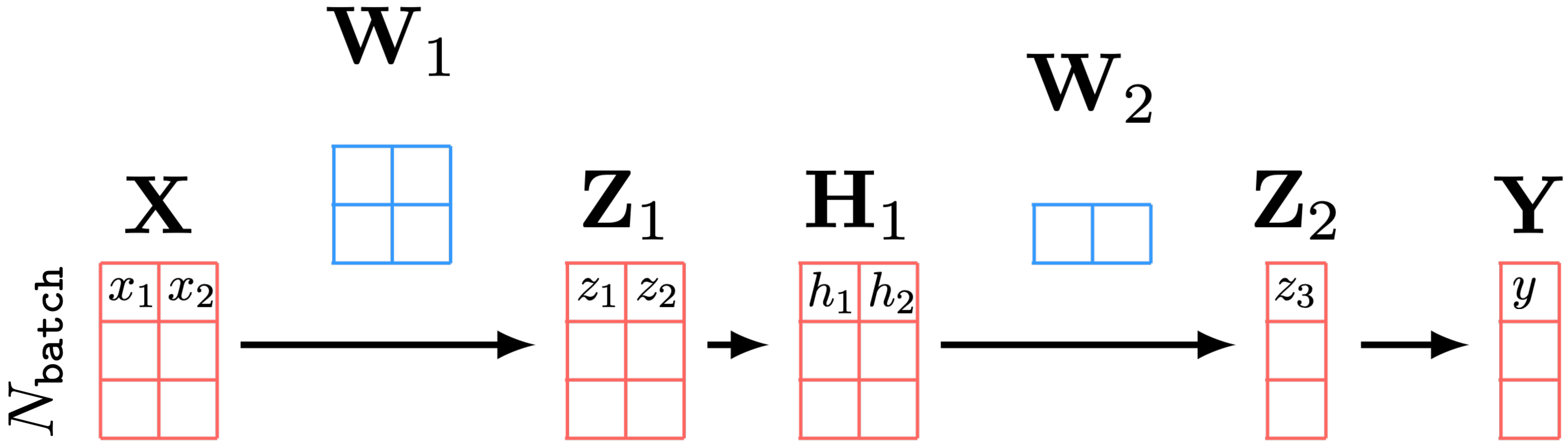
Each layer is a representation of the data

Everything is a tensor



$$\mathbf{z} = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1$$
$$\mathbf{h} = g(\mathbf{z})$$
$$z_3 = \mathbf{W}_2\mathbf{h} + b_2$$
$$y = 1(z_3 > 0)$$

Tensor processing with batch size = 3:



Regularizing deep nets

Deep nets have millions of parameters!

On many datasets, it is easy to overfit — we may have more free parameters than data points to constrain them.

How can we prevent the network from overfitting?

1. Fewer neurons, fewer layers
2. Weight decay and other regularizers
3. Normalization layers
4. …

Recall: regularized least squares

$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

$$R(\theta) = \lambda \left\|\theta\right\|_2^2 \longleftarrow$$

Only use polynomial terms if you really need them! Most terms should be zero

ridge regression, a.k.a., **Tikhonov regularization**

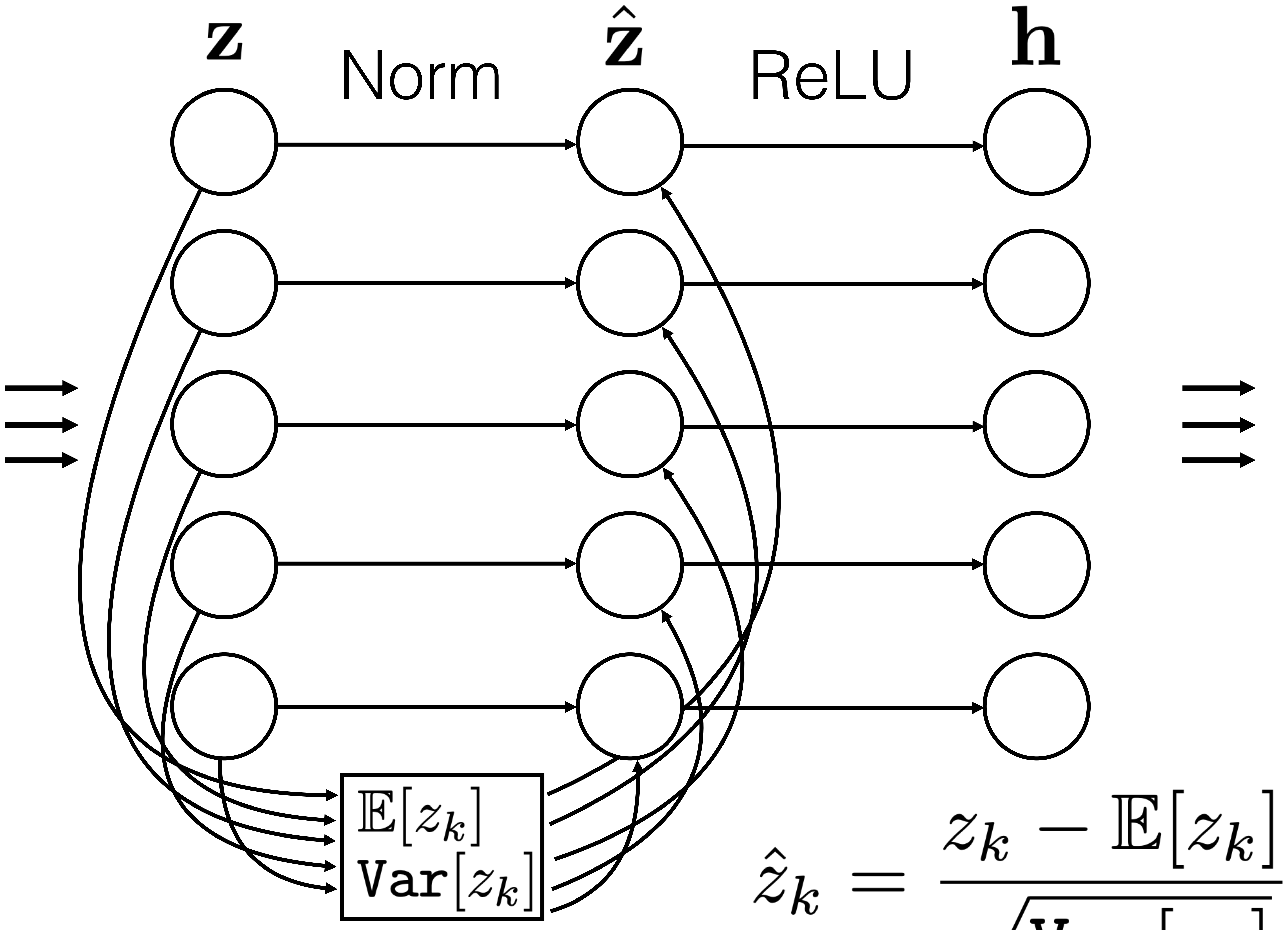
Regularizing the weights in a neural net

$$\theta^* = \arg\min_{\theta} \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)}) + R(\theta)$$

$$R(\mathbf{W}) = \lambda \|\mathbf{W}\|_2^2 \longleftarrow$$ **weight decay**

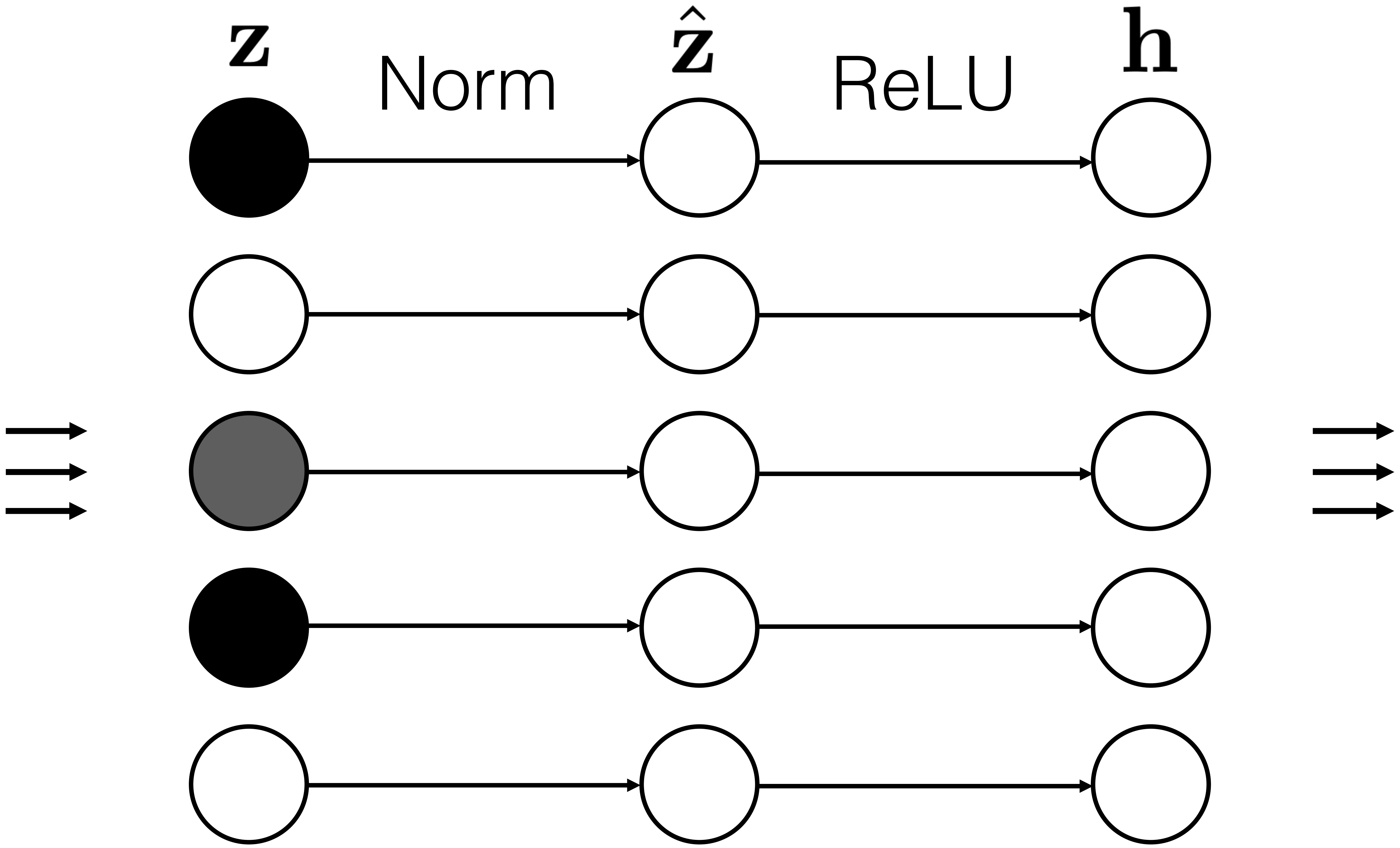
"We prefer to keep weights small."

Normalization layers



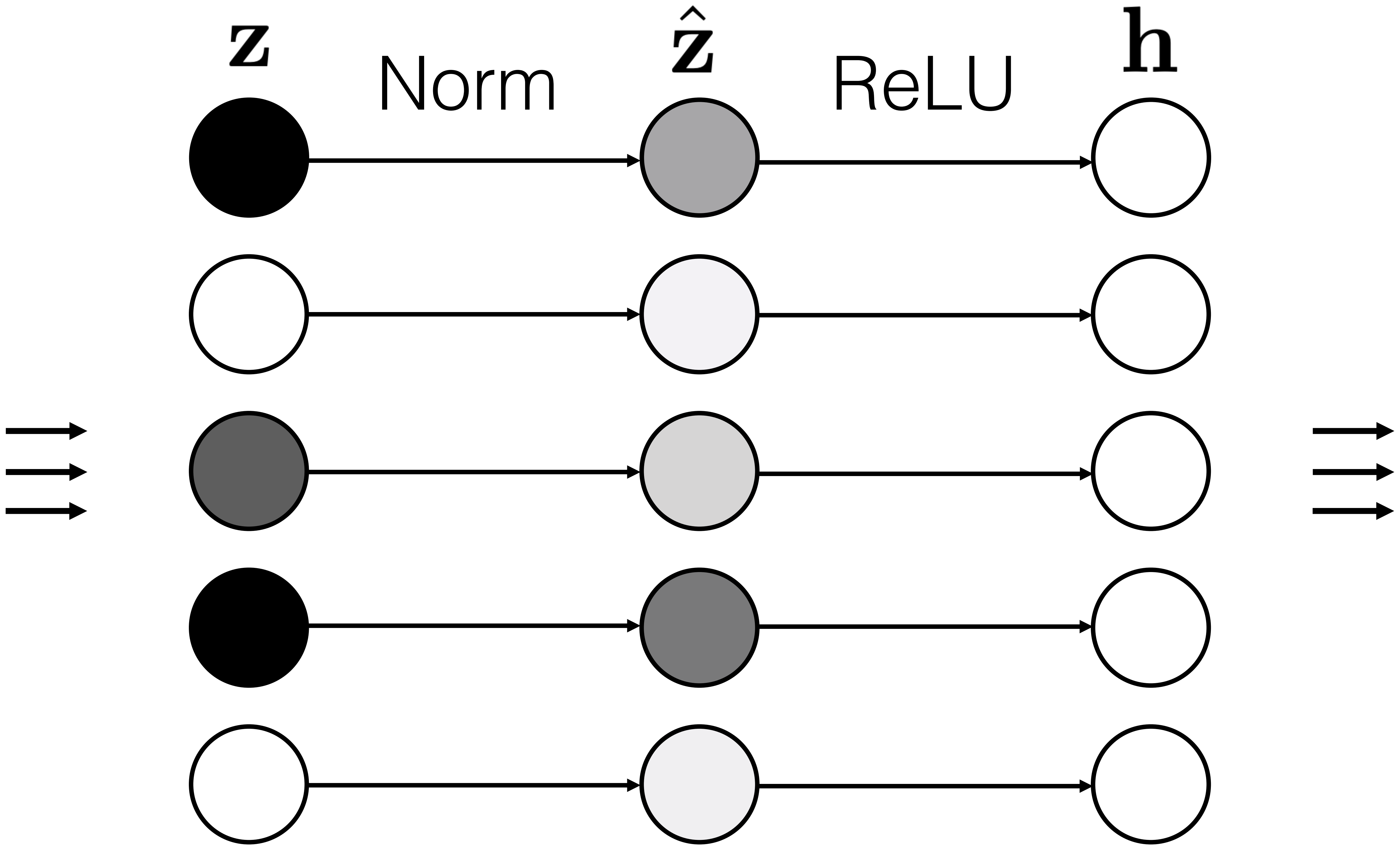
$$\hat{z}_k = \frac{z_k - \mathbb{E}[z_k]}{\sqrt{\mathtt{Var}[z_k]}}$$

Normalization layers



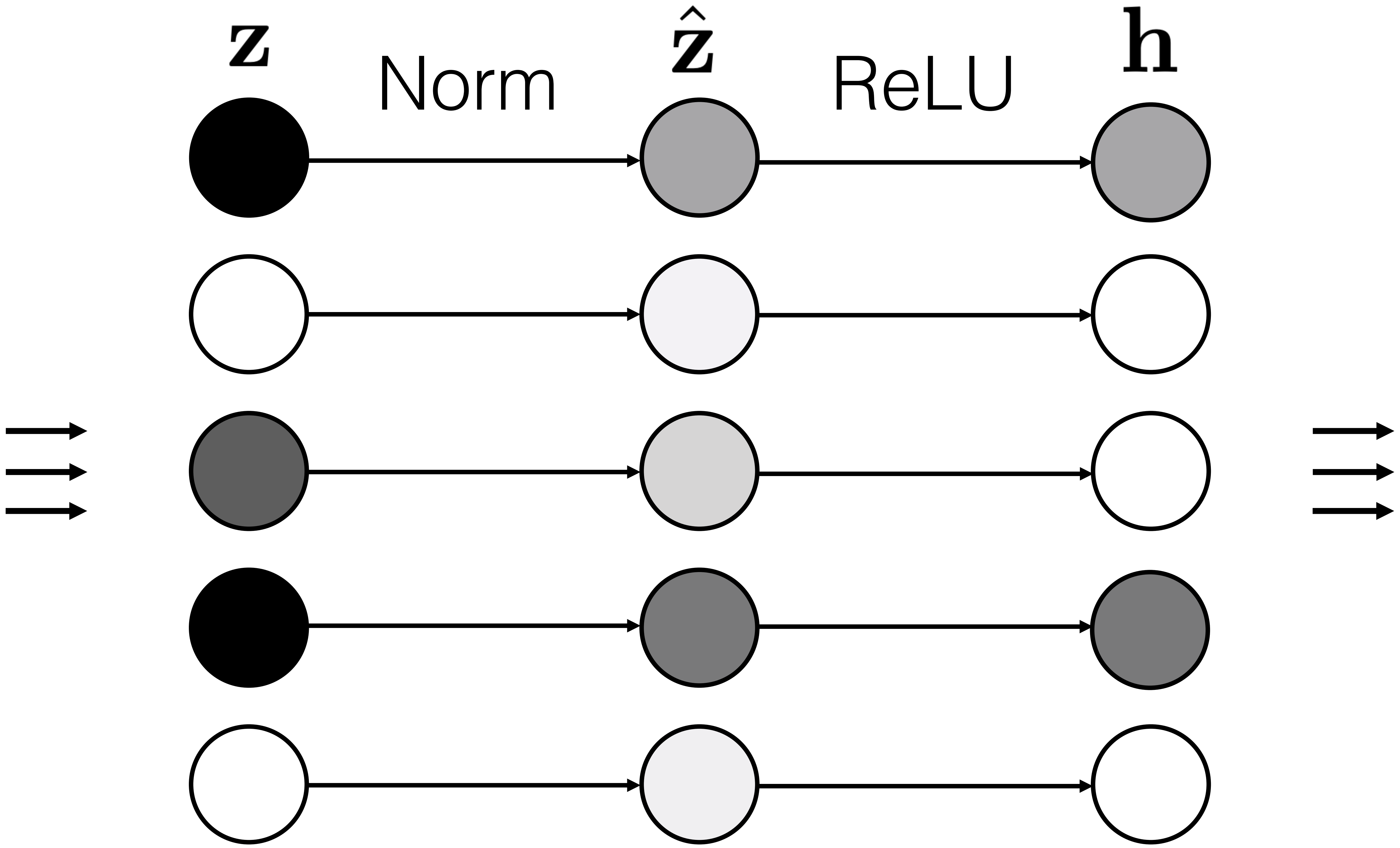
$$\hat{z}_k = \frac{z_k - \mathbb{E}[z_k]}{\sqrt{\mathtt{Var}[z_k]}}$$

Normalization layers



$$\hat{z}_k = \frac{z_k - \mathbb{E}[z_k]}{\sqrt{\mathtt{Var}[z_k]}}$$

Normalization layers



$$\hat{z}_k = \frac{z_k - \mathbb{E}[z_k]}{\sqrt{\mathtt{Var}[z_k]}}$$

Normalization layers

Keep track of mean and variance of a unit (or a population of units) over time.

Standardize unit activations by subtracting mean and dividing by variance.

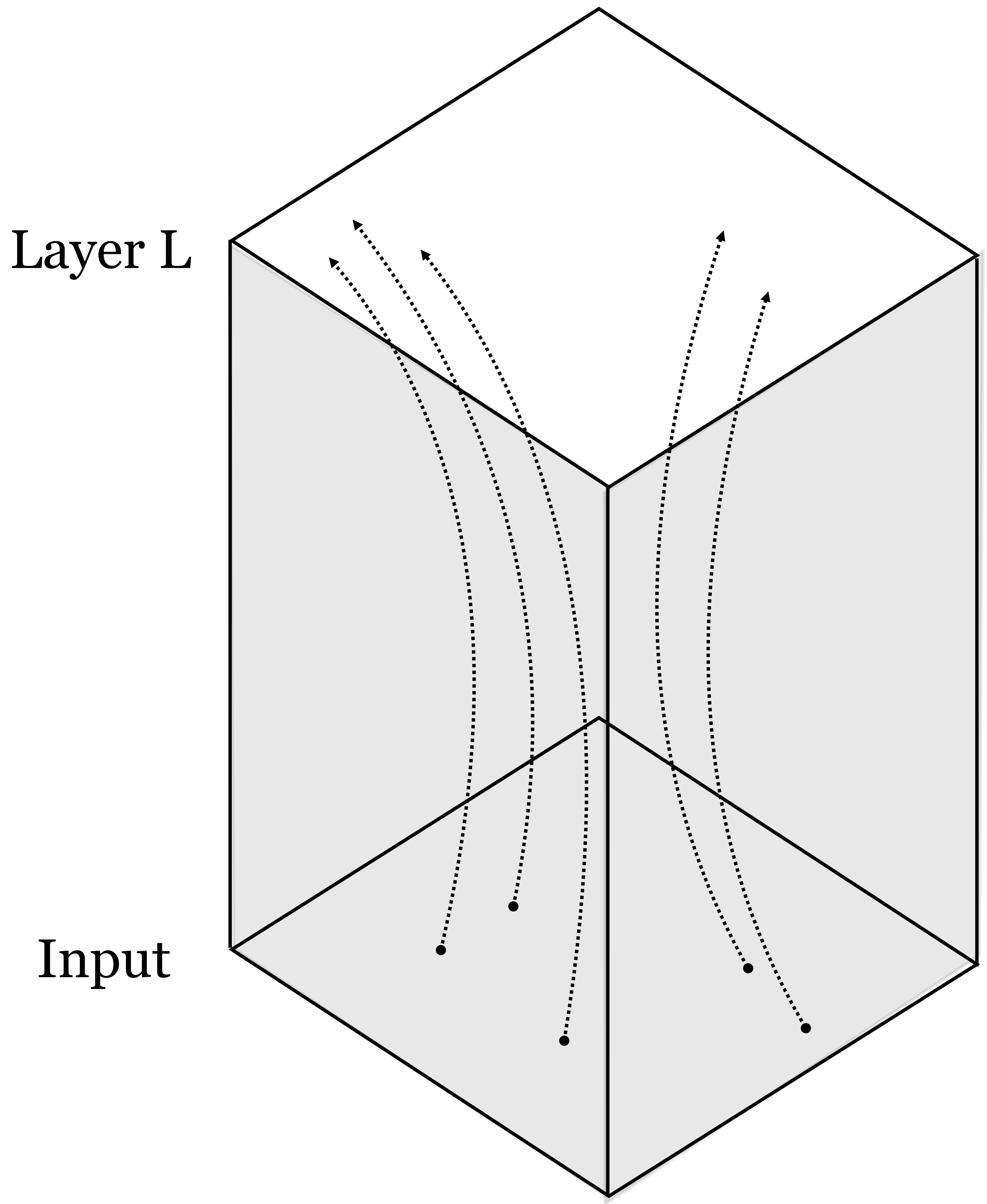
Squashes units into a **standard range**, avoiding overflow.

Also achieves **invariance** to mean and variance of the training signal.

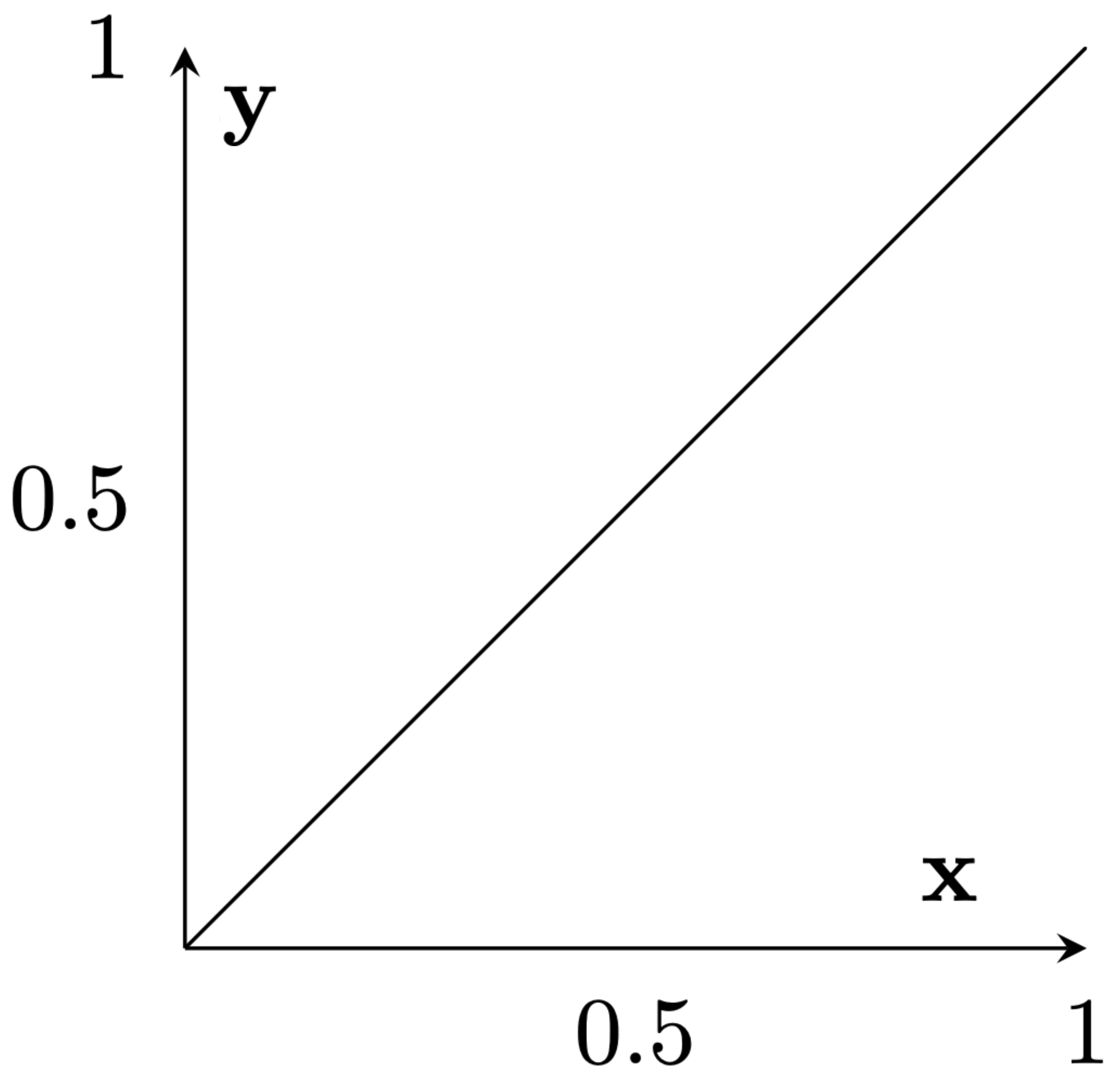
Both these properties reduce the effective capacity of the model, i.e. regularize the model.

Deep nets are data transformers

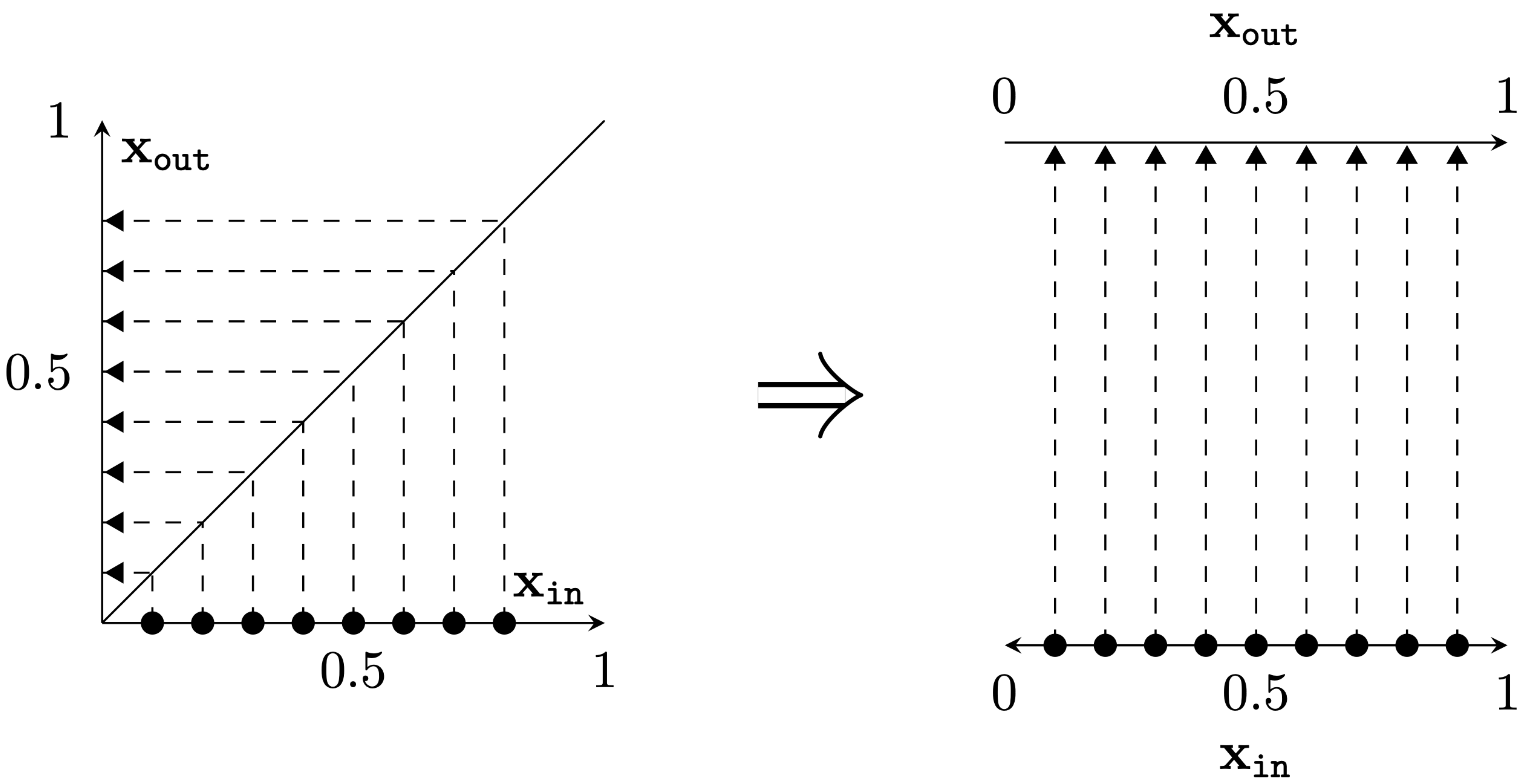
- Deep nets transform datapoints, layer by layer
- Each layer is a different *representation* of the data
- We call these representations **embeddings**



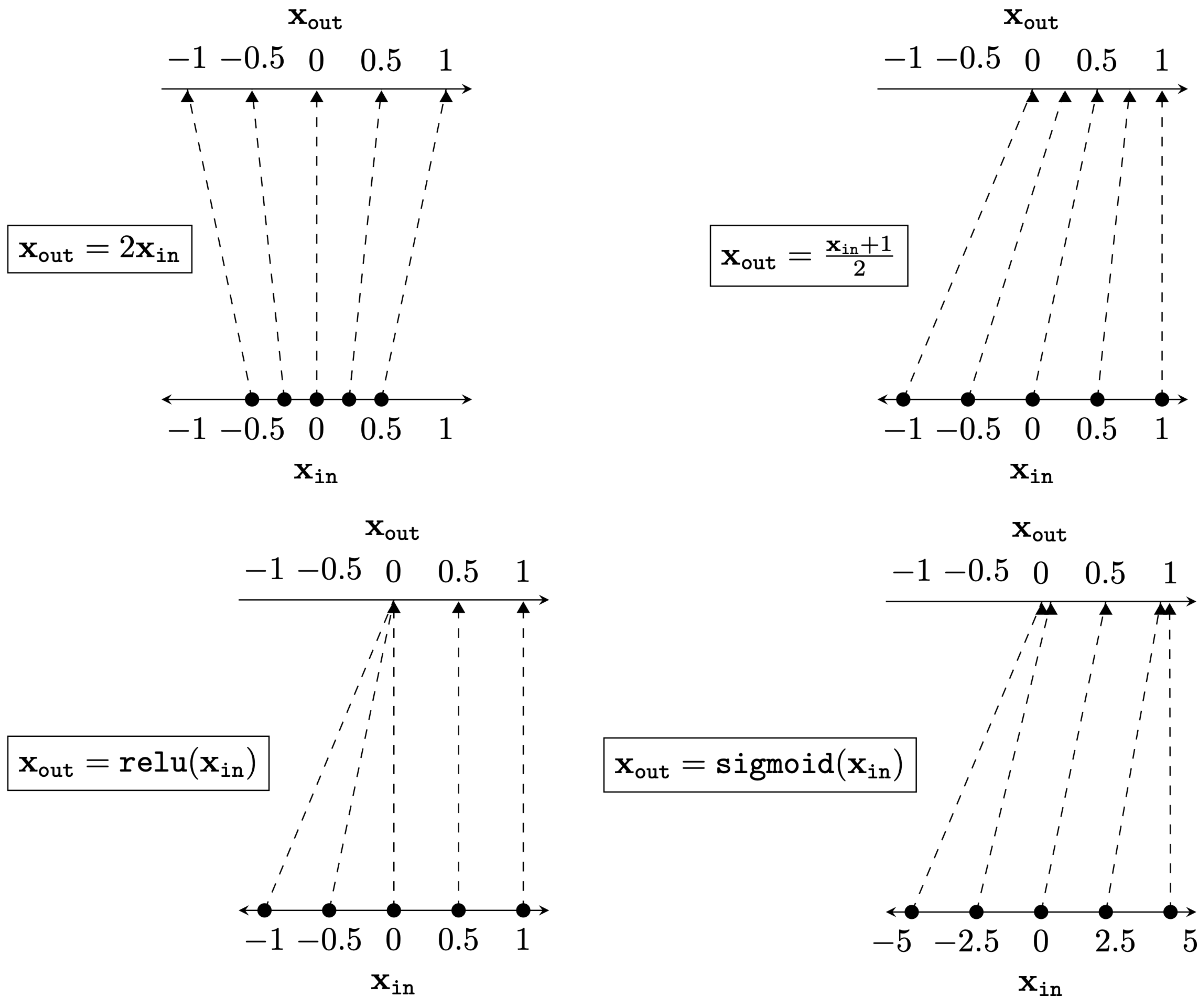
Two different ways to represent a function



Two different ways to represent a function



Data transformations for a variety of neural net layers

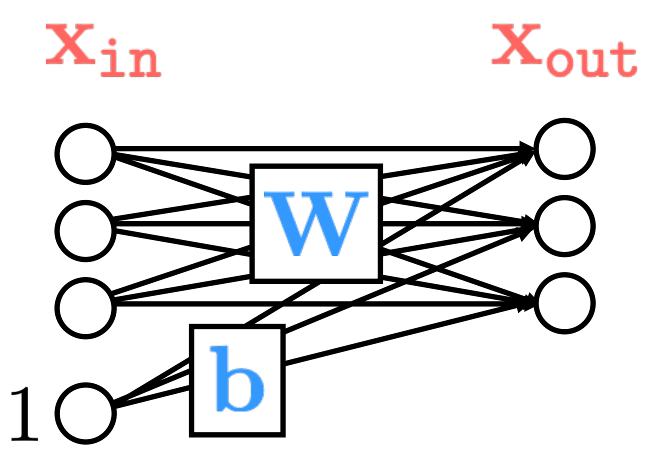


Activations

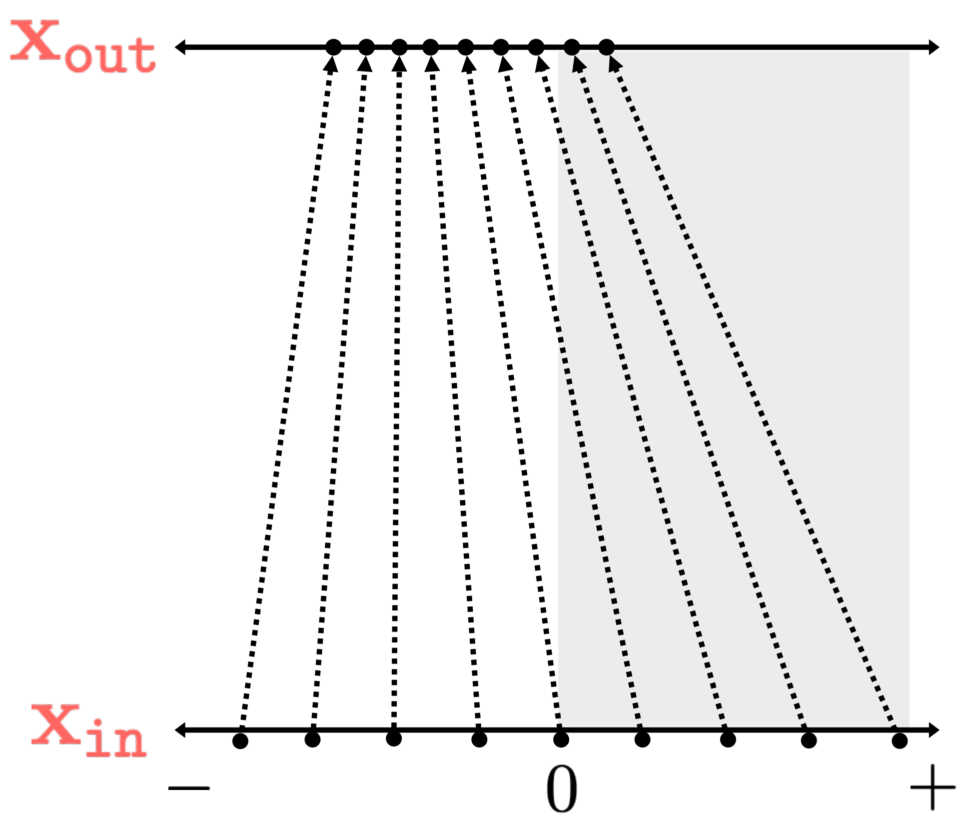
Parameters

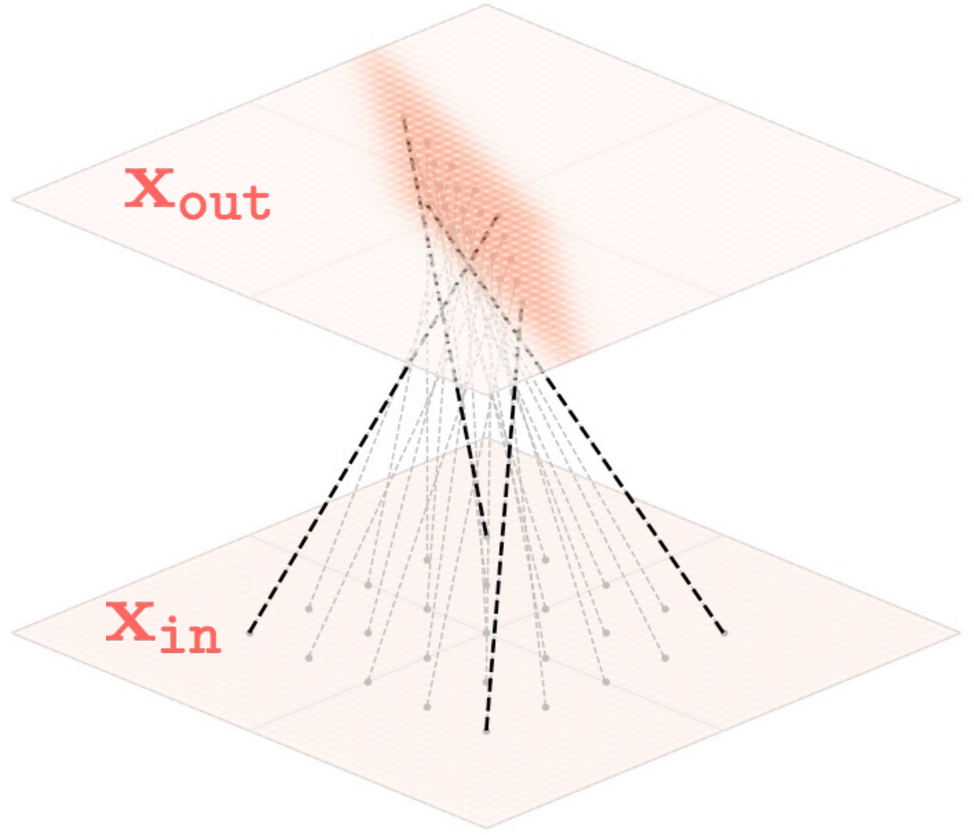
Wiring graph | Equation | Mapping 1D | Mapping 2D

linear

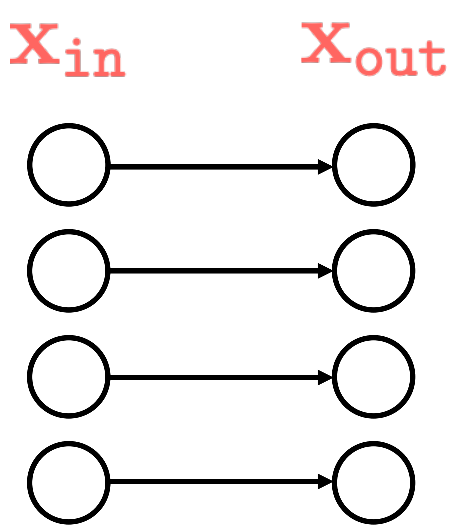


$$\mathbf{x}_{\text{out}} = \mathbf{W}\mathbf{x}_{\text{in}} + \mathbf{b}$$

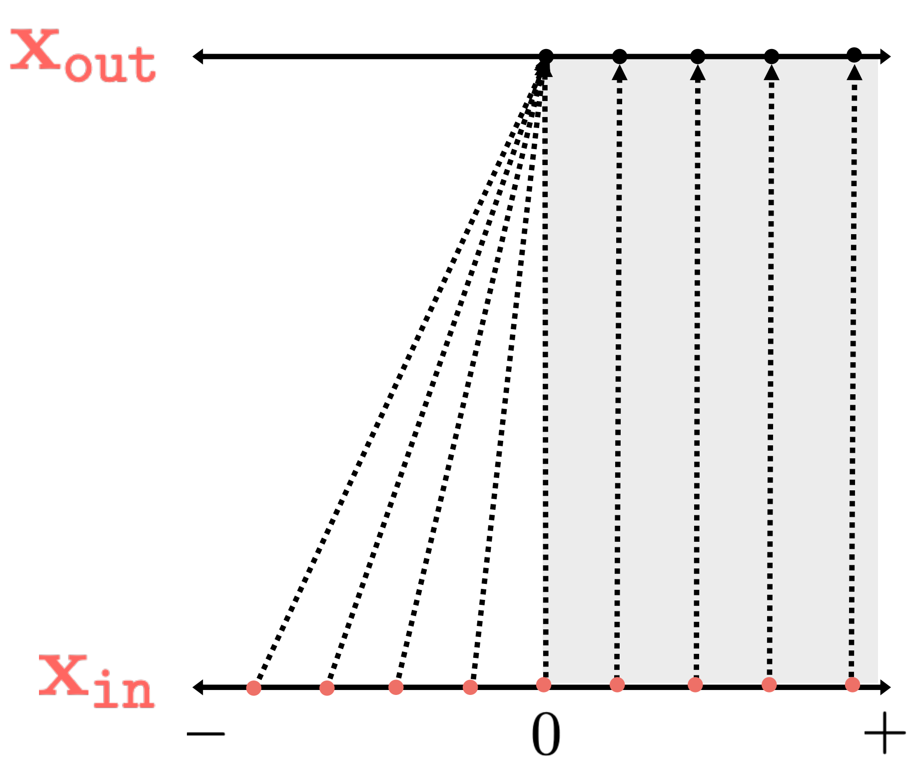


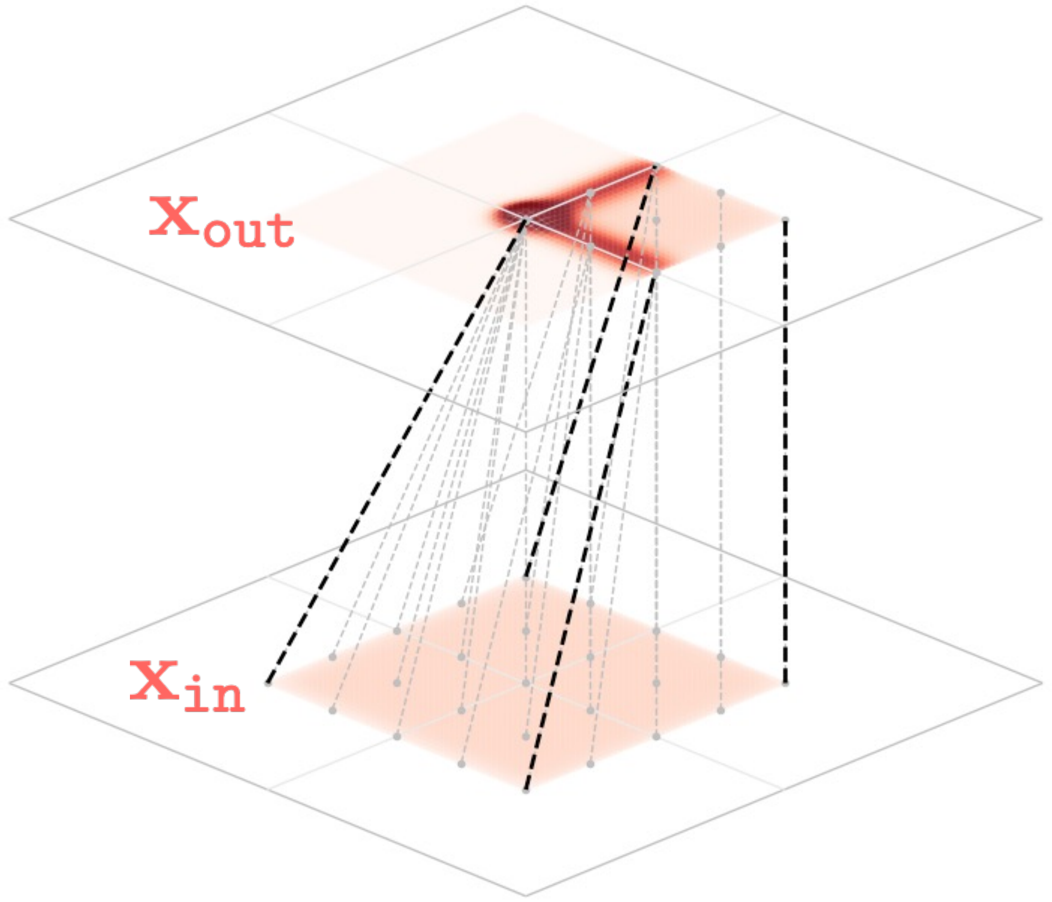


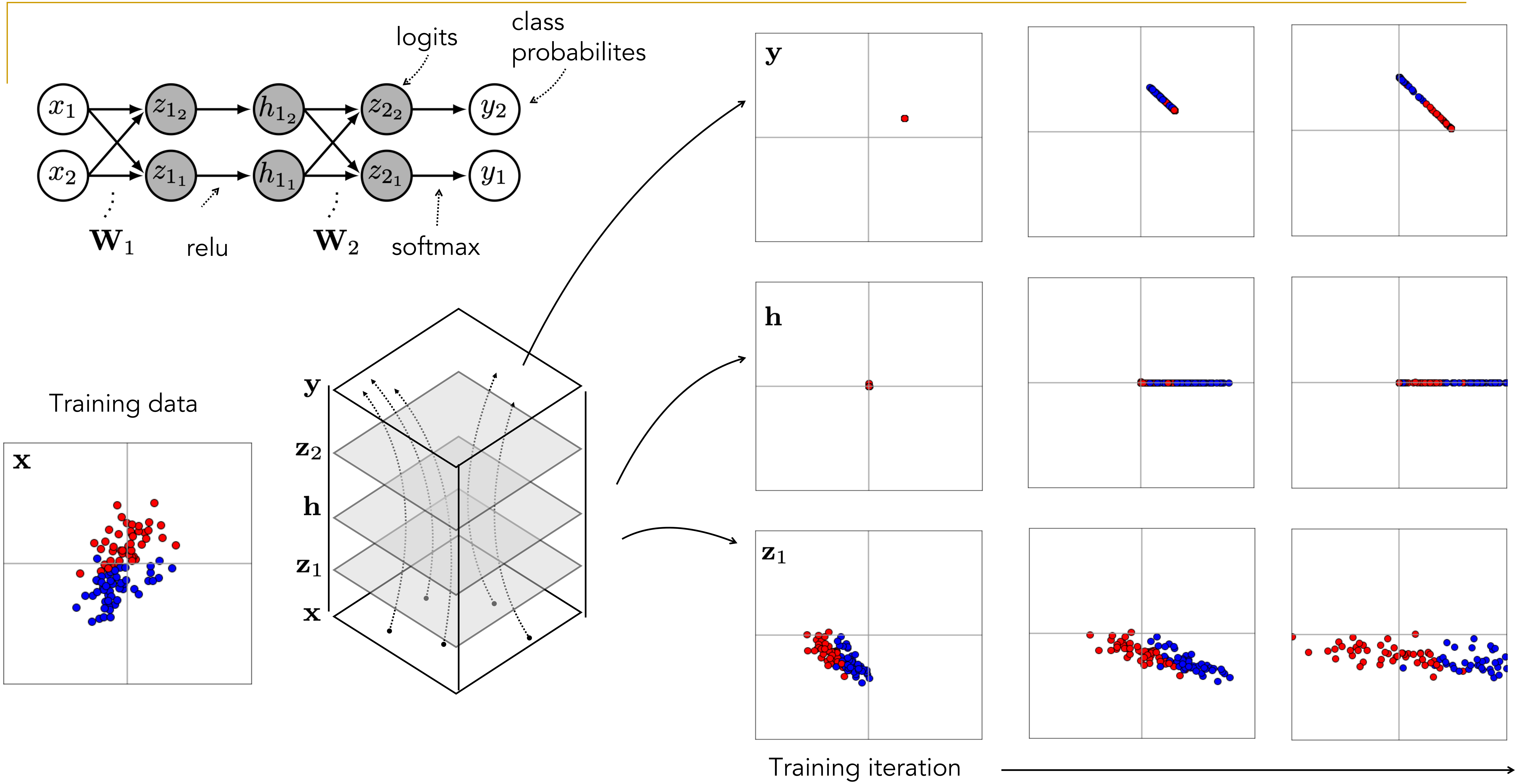
relu

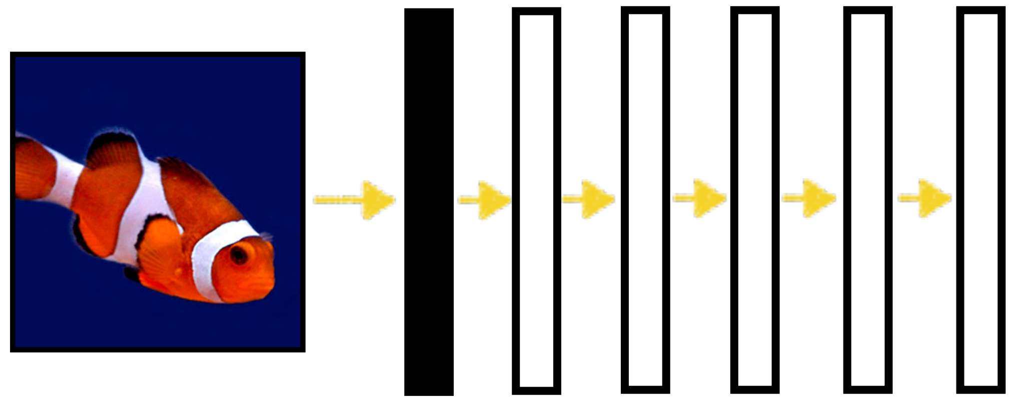


$$x_{\text{out}_i} = \max(x_{\text{in}_i}, 0)$$

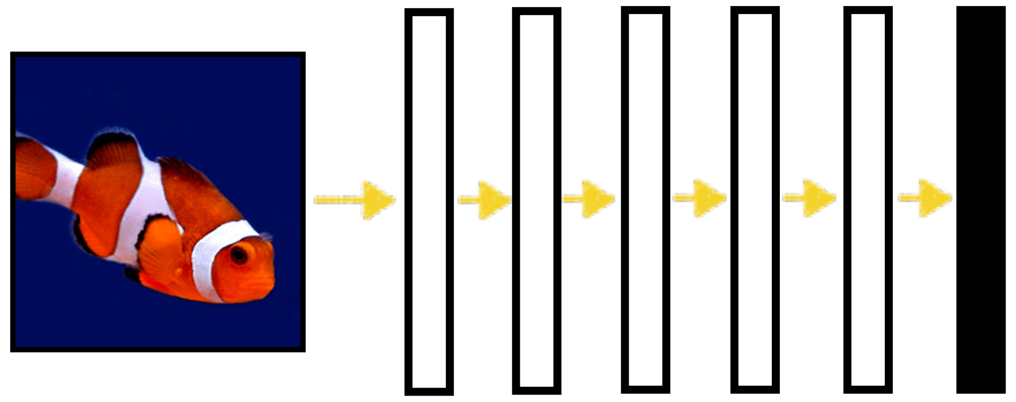




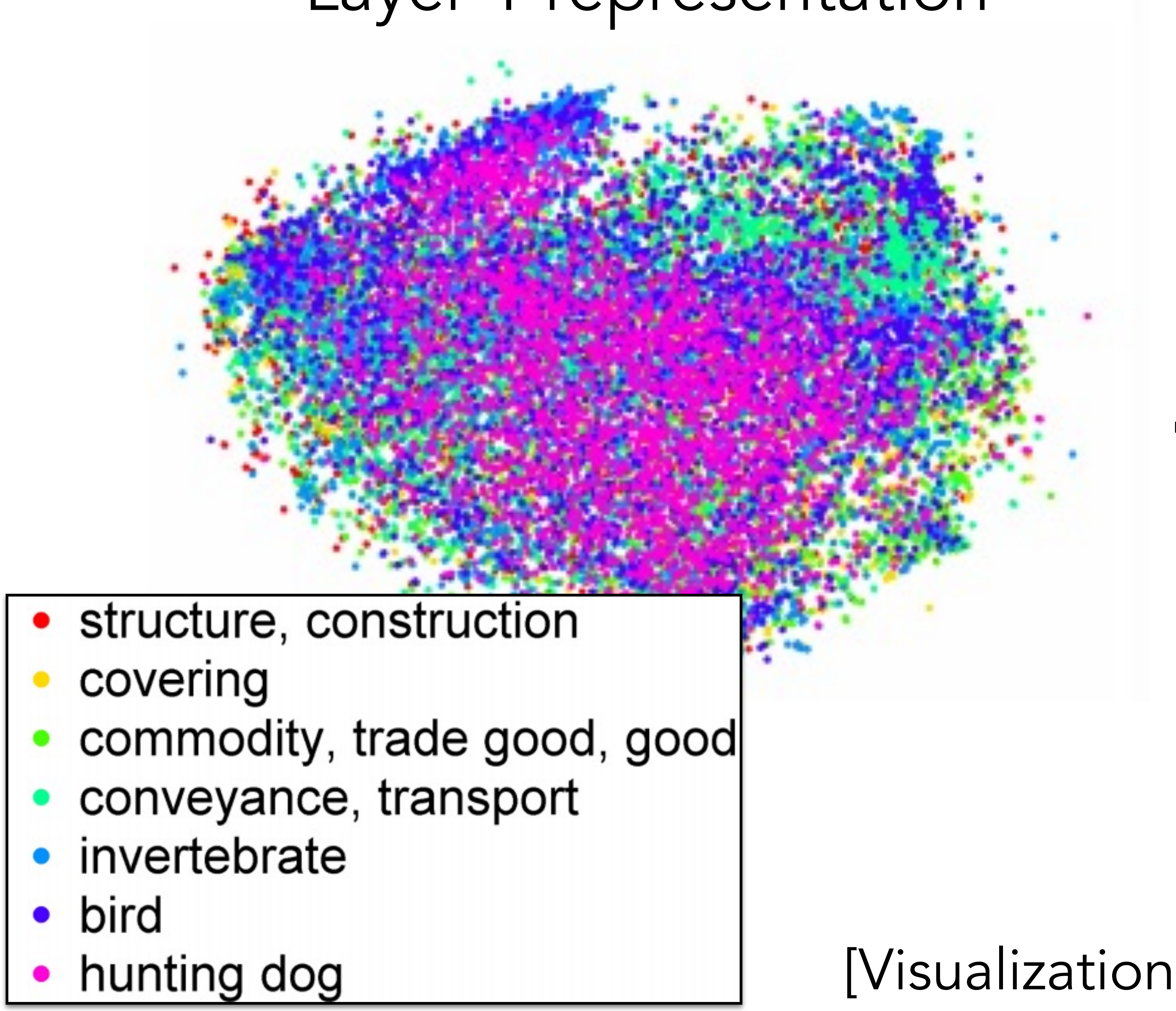

logits
class probabilites
x_1
x_2
z_{1_2}
z_{1_1}
h_{1_2}
h_{1_1}
z_{2_2}
z_{2_1}
y_2
y_1
$\mathbf{W}_1$
relu
$\mathbf{W}_2$
softmax
Training data
$\mathbf{x}$
$\mathbf{y}$
$\mathbf{z}_2$
$\mathbf{h}$
$\mathbf{z}_1$
$\mathbf{x}$
$\mathbf{y}$
$\mathbf{h}$
$\mathbf{z}_1$
Training iteration

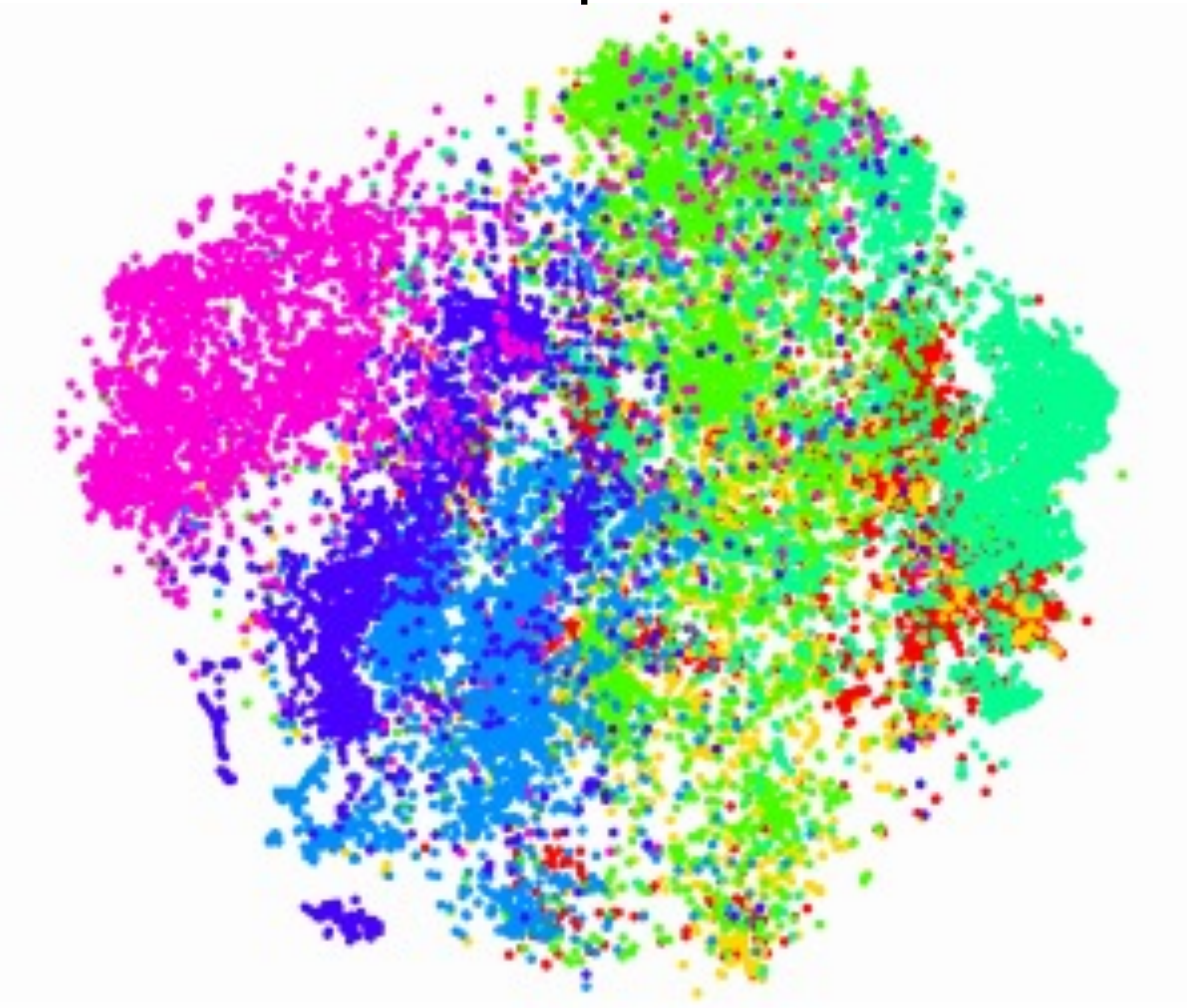


Layer 1 representation



Layer 6 representation





[DeCAF, Donahue, Jia, et al. 2013]

[Visualization technique : t-sne, van der Maaten & Hinton, 2008]