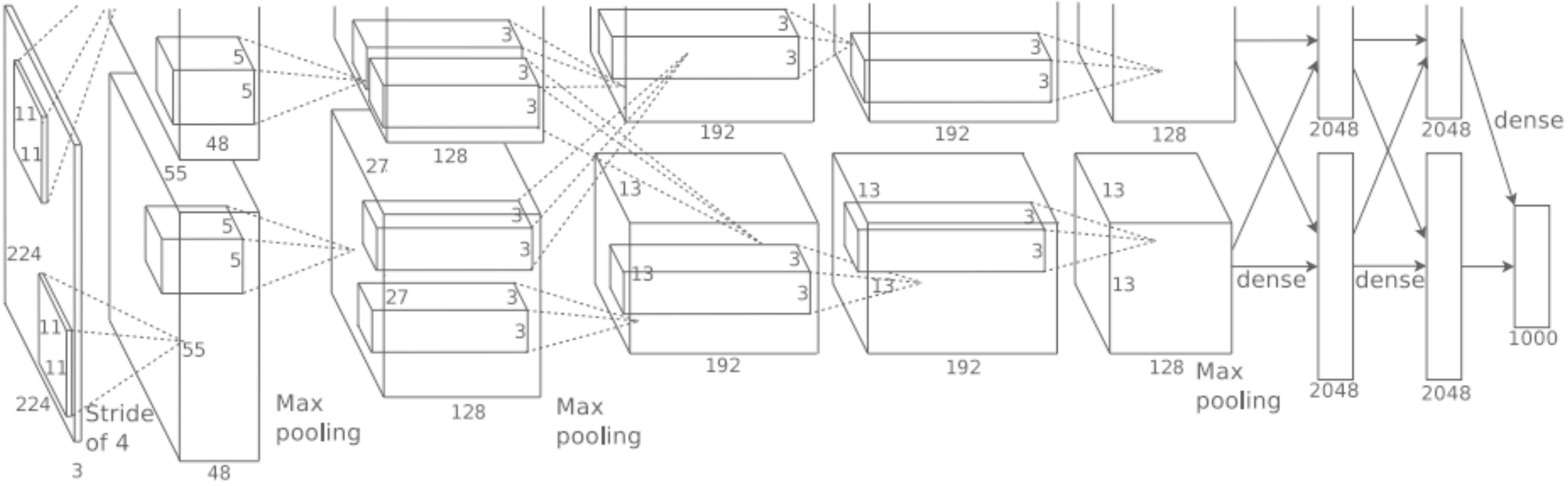
Lecture 9

Neural Networks



6.869/6.819 Advances in Computer Vision

Spring 2022

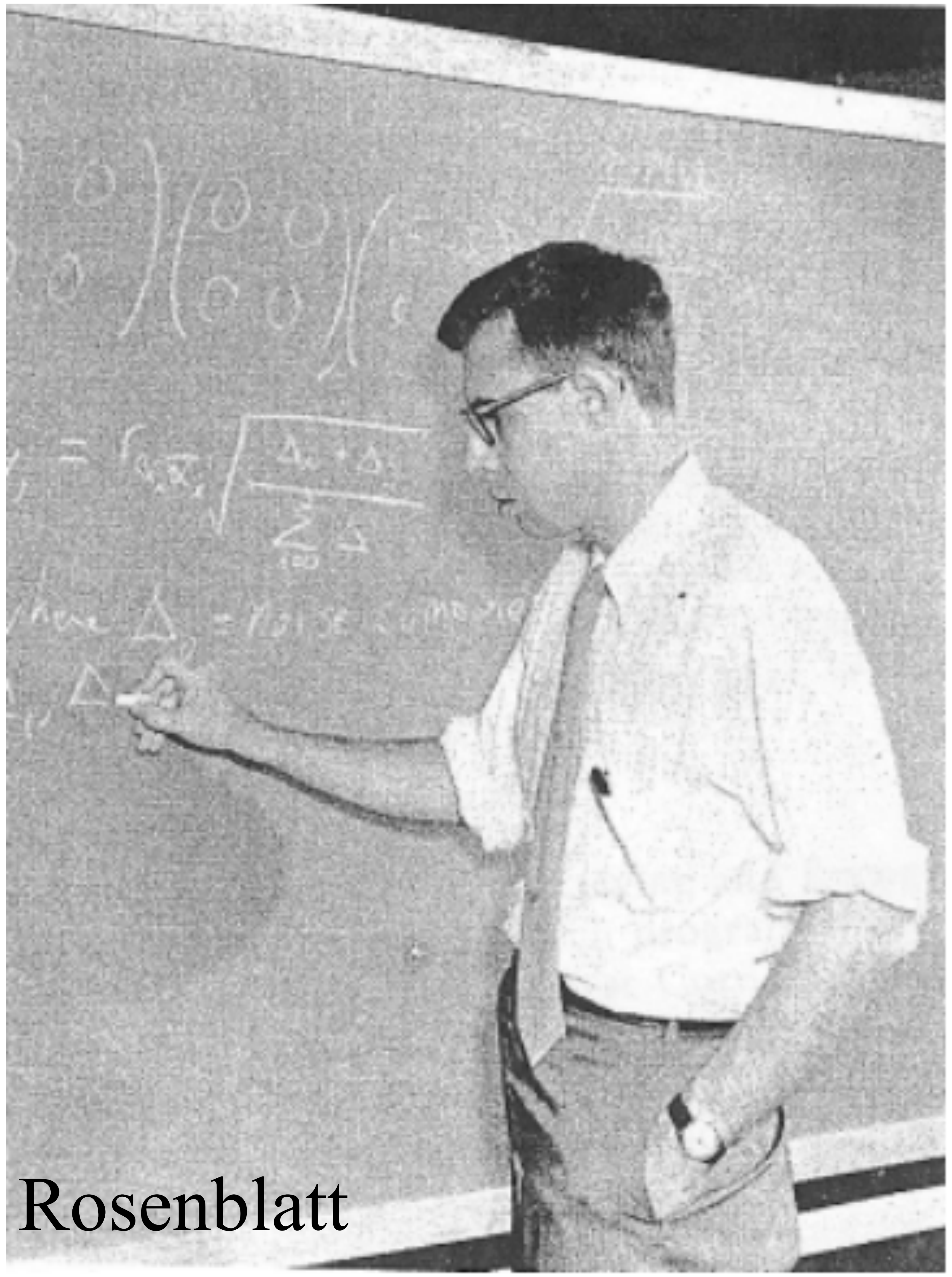
Bill Freeman, Phillip Isola

1. Introduction to Deep Learning

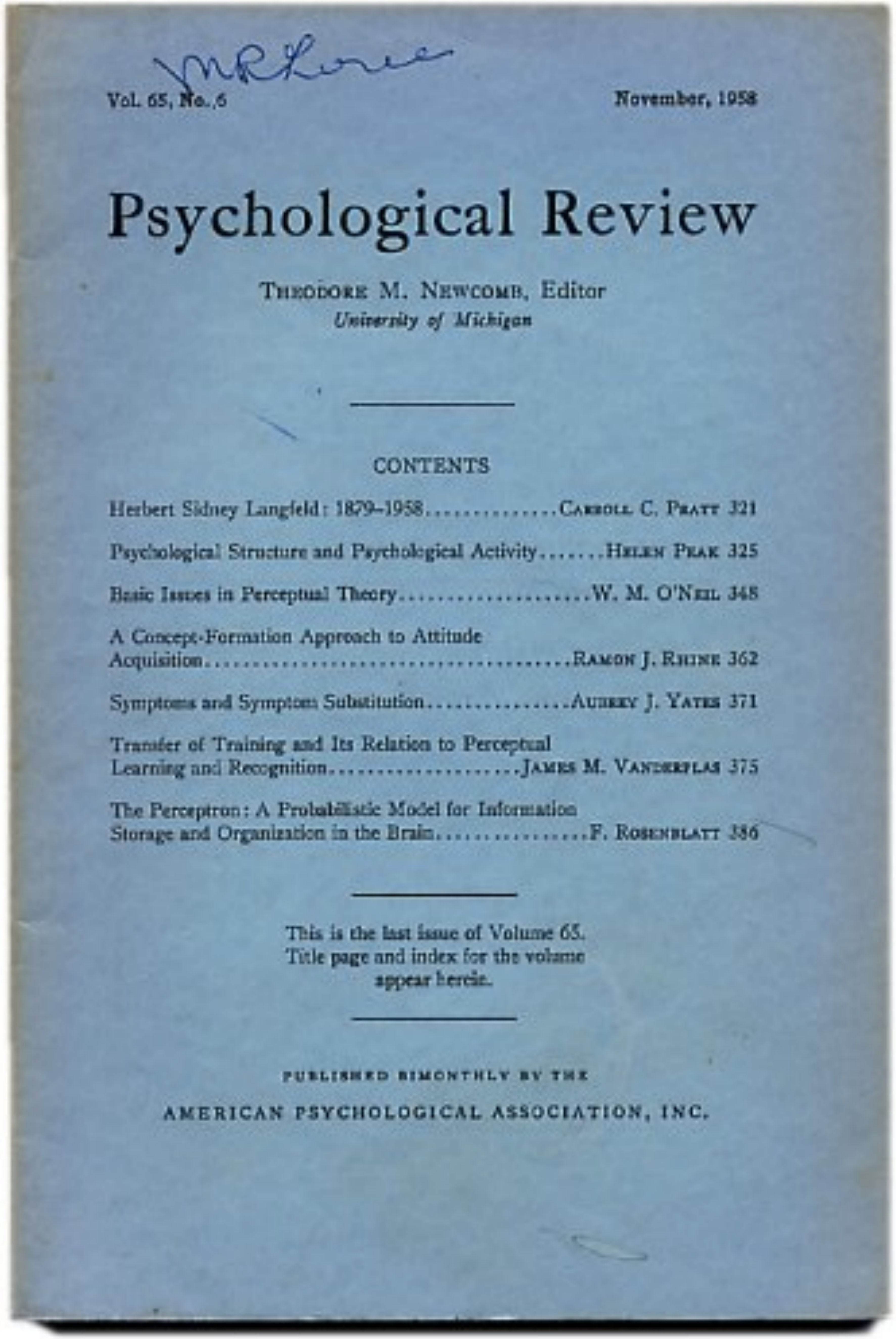
- Brief history
- Basic formulation *(hierarchical processing)*
- Optimization via gradient descent
- Layer types *(Linear, Pointwise non-linearity)*
- Everything is a tensor
- Deep nets as data transformers

A brief history of Neural Networks

Perceptrons, 1958



http://www.ecse.rpi.edu/homepages/nagy/PDF_chrono/2011_Nagy_Pace_FR.pdf. Photo by George Nagy



Vol. 65, No. 6 November, 1958

Psychological Review

THEODORE M. NEWCOMB, Editor
University of Michigan

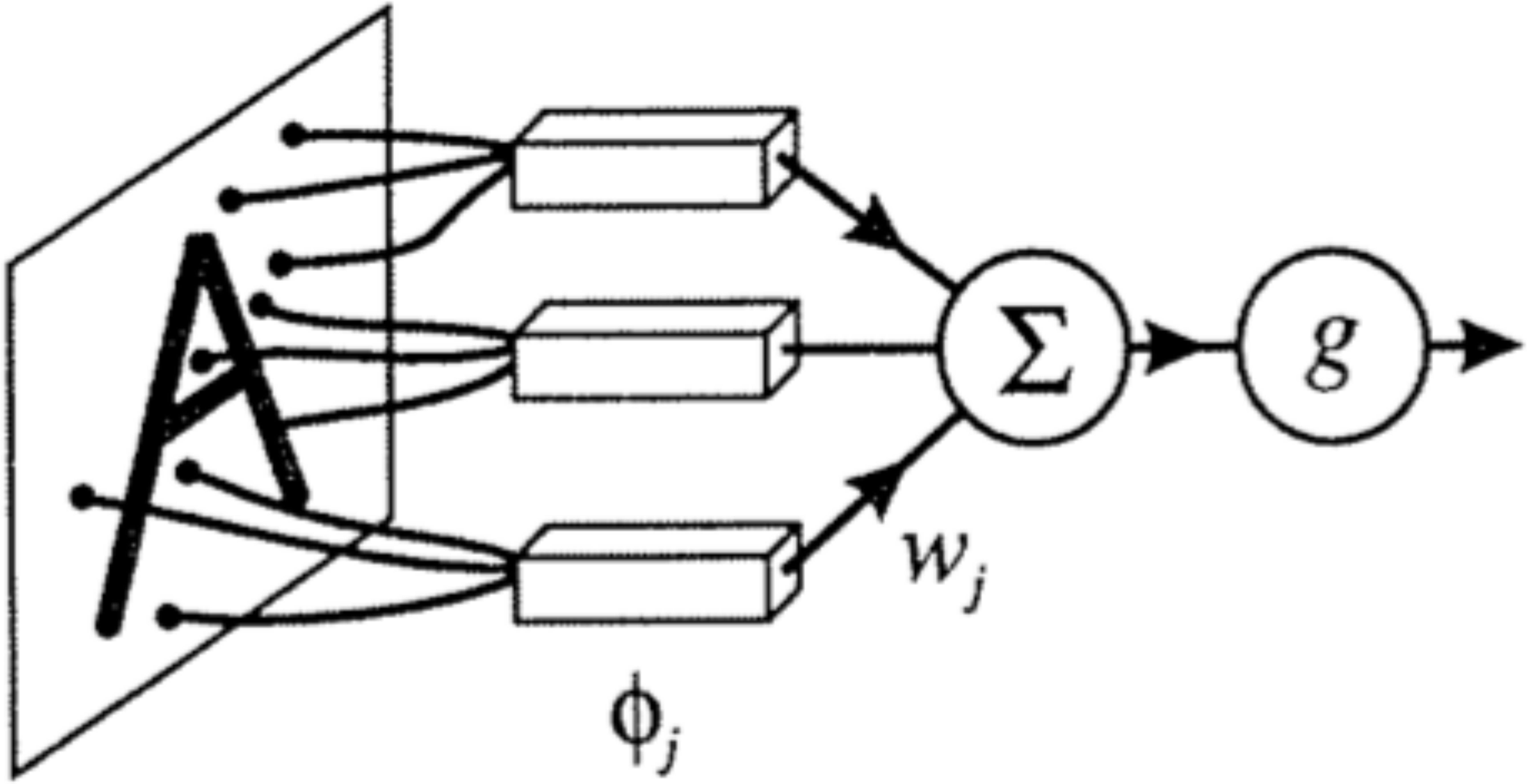
CONTENTS

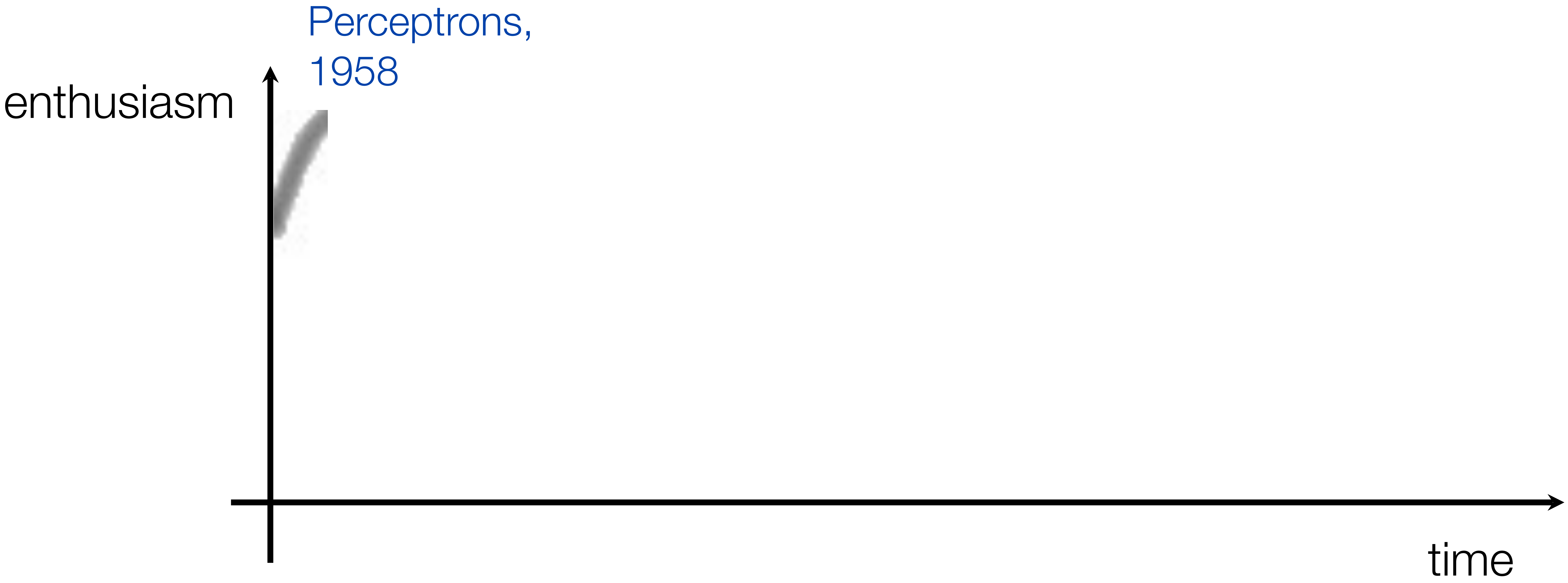
This is the last issue of Volume 65.
Title page and index for the volume
appear herein.

PUBLISHED BIMONTHLY BY THE
AMERICAN PSYCHOLOGICAL ASSOCIATION, INC.

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf

Perceptrons, 1958




Perceptrons,
1958
enthusiasm
time

Minsky and Papert, Perceptrons, 1972

Perceptrons, expanded edition

An Introduction to Computational Geometry

By Marvin Minsky and Seymour A. Papert

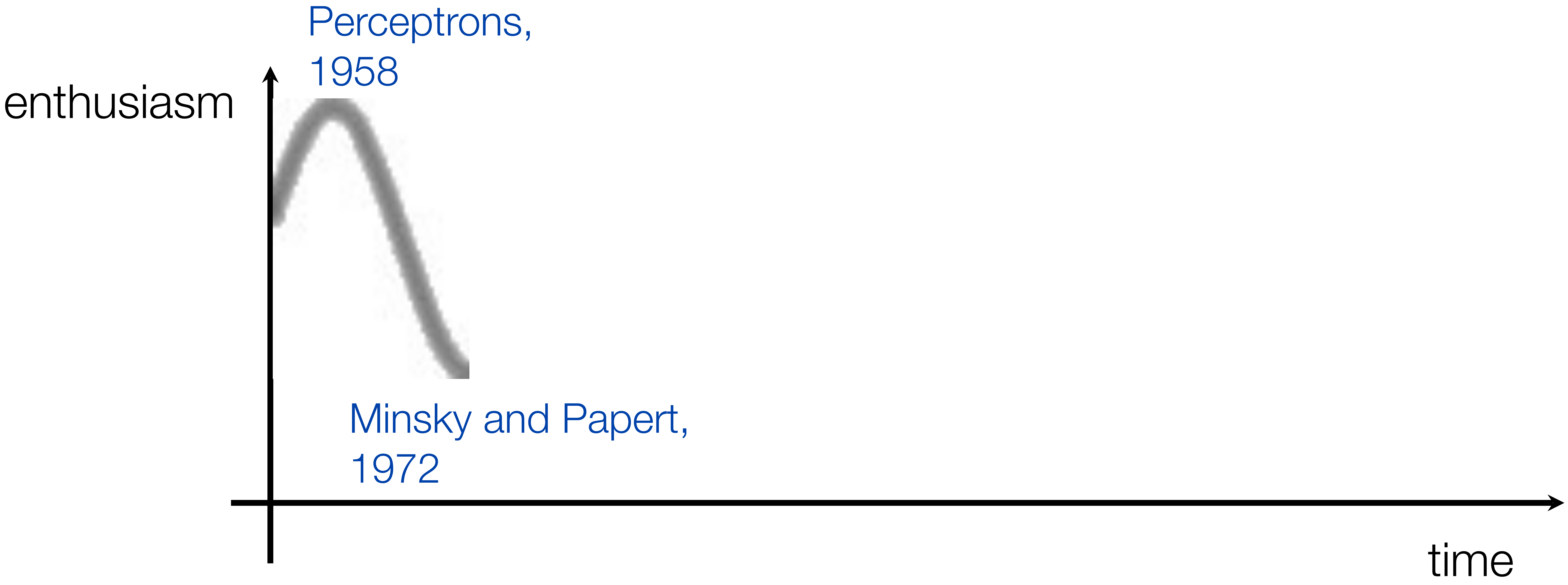
Paperback | **$35.00 Short** | **£24.95** | ISBN: 9780262631112 | 308 pp. | 6 x 8.9 in | December 1987

Overview

Perceptrons - the first systematic study of parallelism in computation - has remained a classical work on threshold automata networks for nearly two decades. It marked a historical turn in artificial intelligence, and it is required reading for anyone who wants to understand the connectionist counterrevolution that is going on today.

Artificial-intelligence research, which for a time concentrated on the programming of ton Neumann computers, is swinging back to the idea that intelligence might emerge from the activity of networks of neuronlike entities. Minsky and Papert's book was the first example of a mathematical analysis carried far enough to show the exact limitations of a class of computing machines that could seriously be considered as models of the brain. Now the new developments in mathematical tools, the recent interest of physicists in the theory of disordered matter, the new insights into and psychological models of how the brain works, and the evolution of fast computers that can simulate networks of automata have given *Perceptrons* new importance.

Witnessing the swing of the intellectual pendulum, Minsky and Papert have added a new chapter in which they discuss the current state of parallel computers, review developments since the appearance of the 1972 edition, and identify new research directions related to connectionism. They note a central theoretical challenge facing connectionism: the challenge to reach a deeper understanding of how "objects" or "agents" with individuality can emerge in a network. Progress in this area would link connectionism with what the authors have called "society theories of mind."



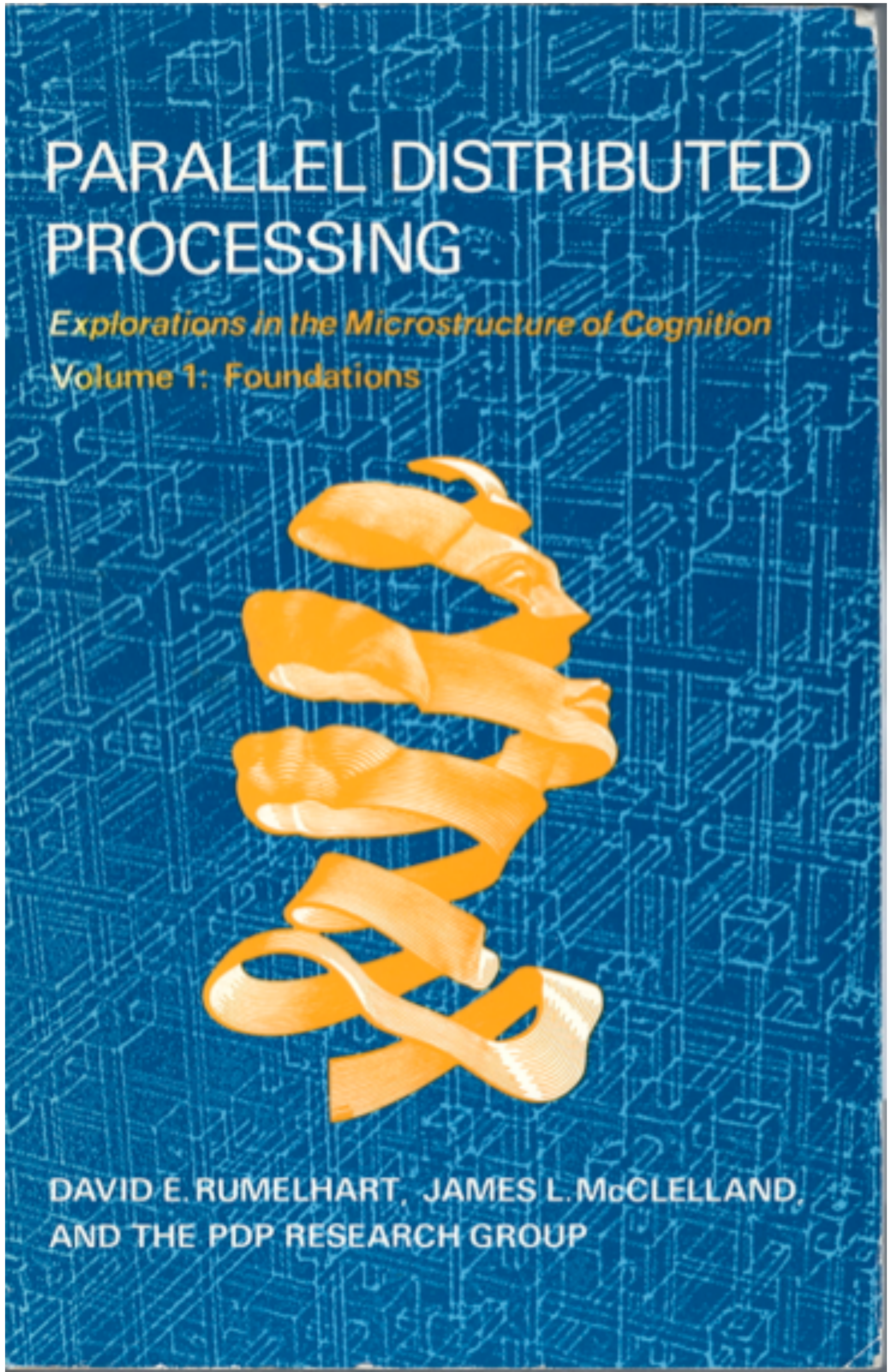
Perceptrons,
1958

enthusiasm

Minsky and Papert,
1972

time

Parallel Distributed Processing (PDP), 1986



PARALLEL DISTRIBUTED PROCESSING

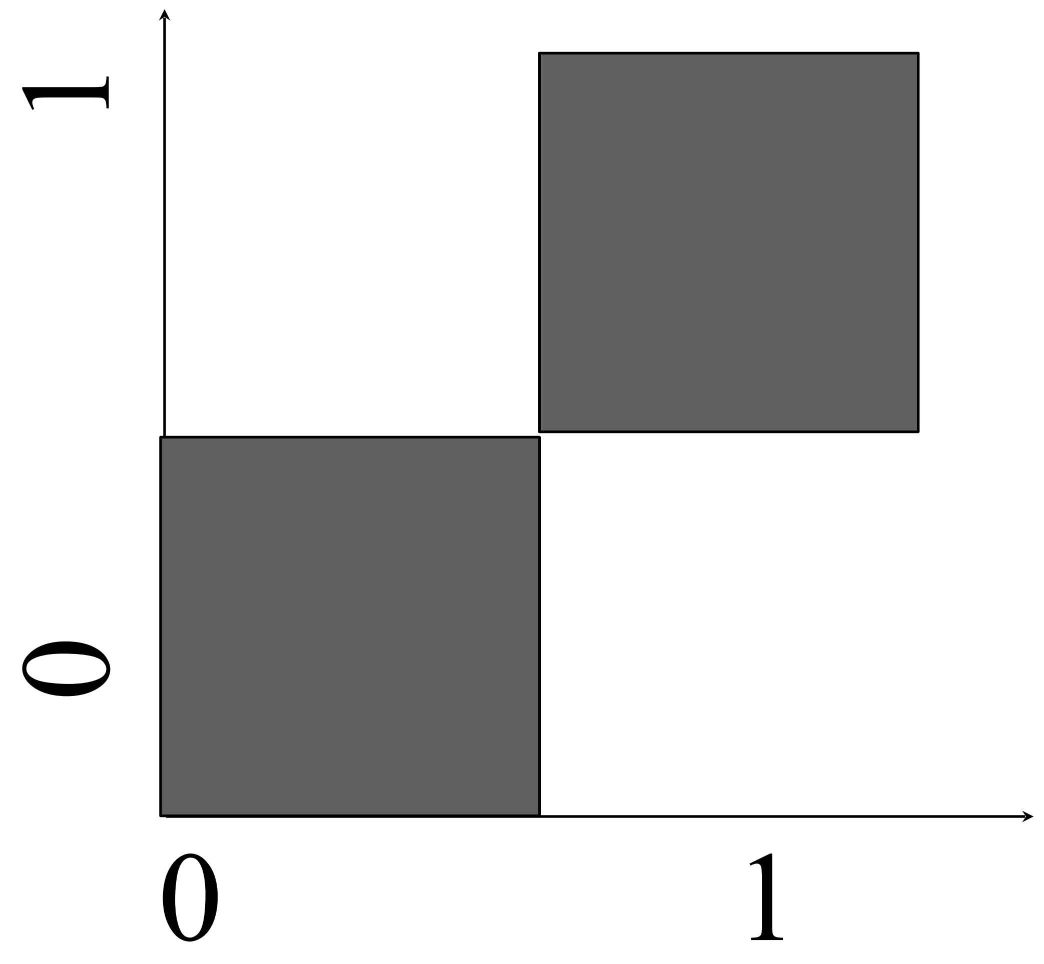
Explorations in the Microstructure of Cognition

Volume 1: Foundations

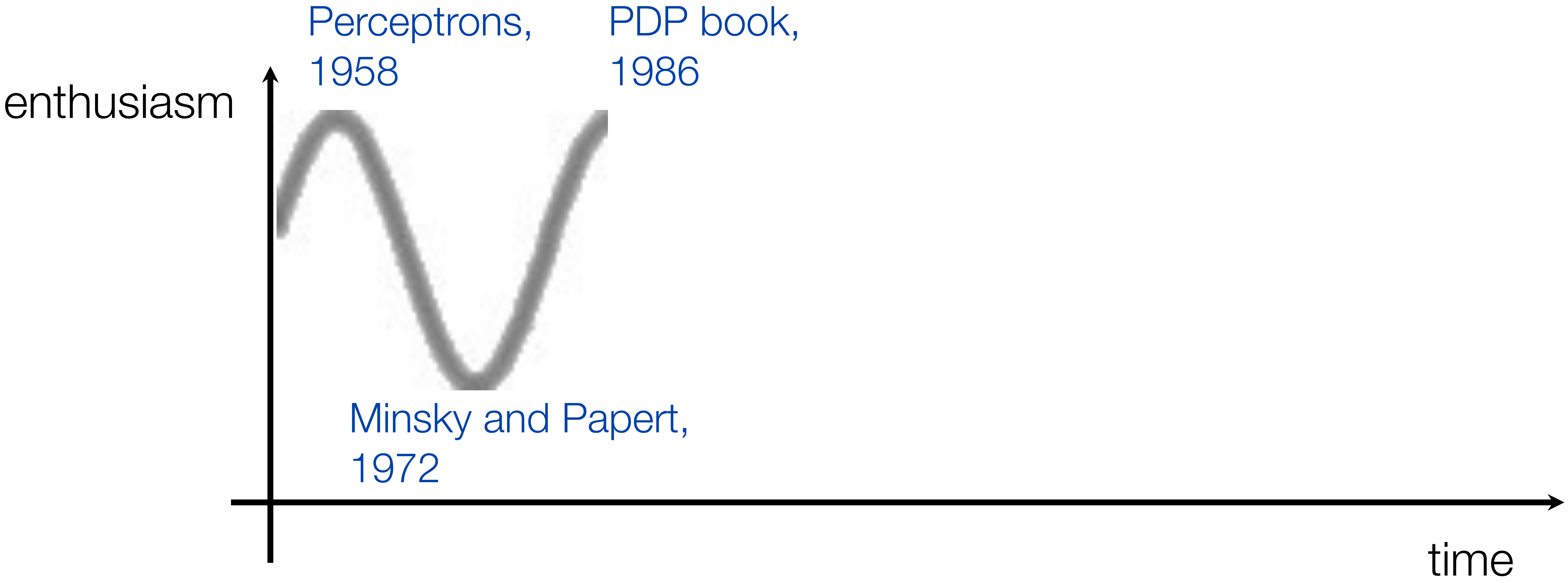
DAVID E. RUMELHART, JAMES L. McCLELLAND,
AND THE PDP RESEARCH GROUP

XOR problem

Inputs		Output
0	0	0
1	0	1
0	1	1
1	1	0



PDP authors pointed to the backpropagation algorithm as a breakthrough, allowing multi-layer neural networks to be trained. Among the functions that a multi-layer network can represent but a single-layer network cannot: the XOR function.



Perceptrons, 1958

PDP book, 1986

Minsky and Papert, 1972

LeCun conv nets, 1998

PROC. OF THE IEEE, NOVEMBER 1998 7

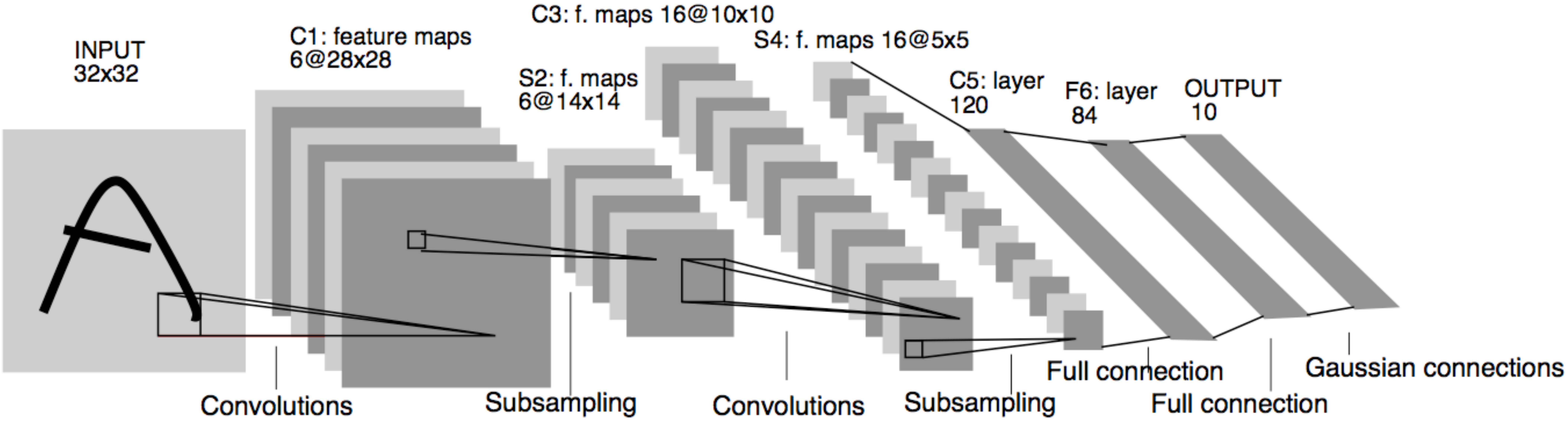
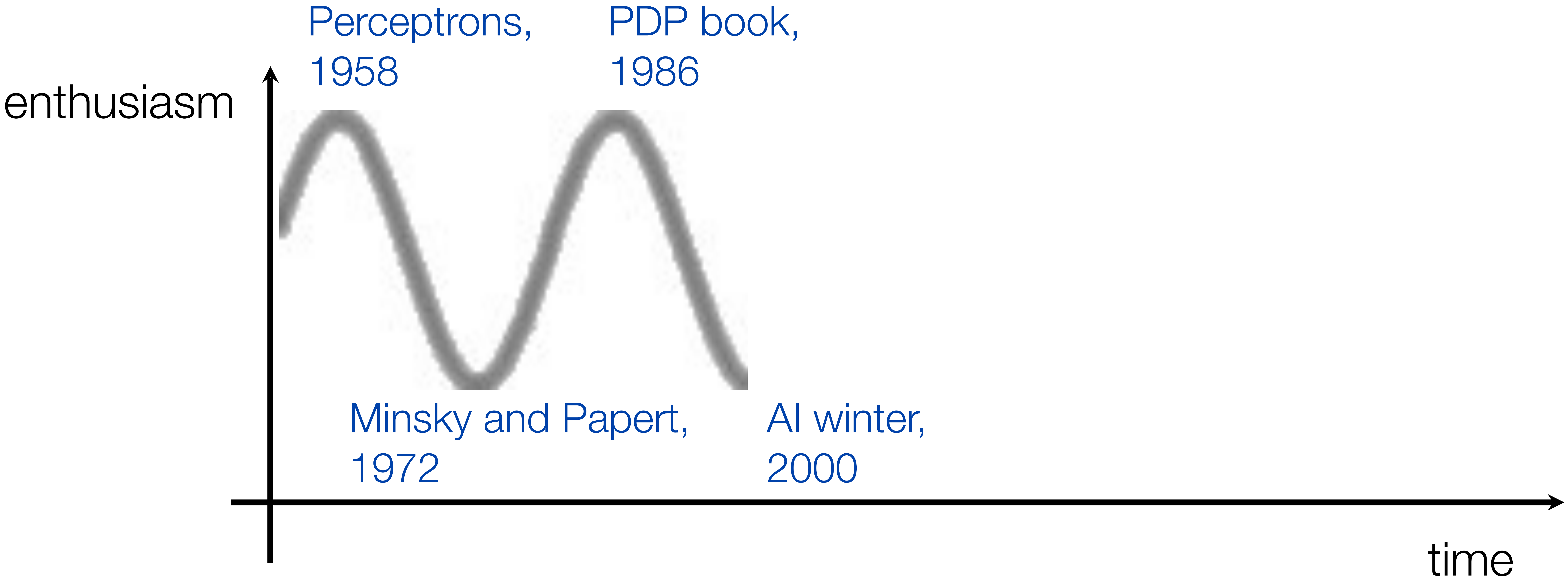


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

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

Neural 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:
  - <u>title words predictive of paper acceptance</u>: "Belief Propagation" and "Gaussian".
  - <u>title words predictive of paper rejection</u>: "Neural" and "Network".



Perceptrons,
1958

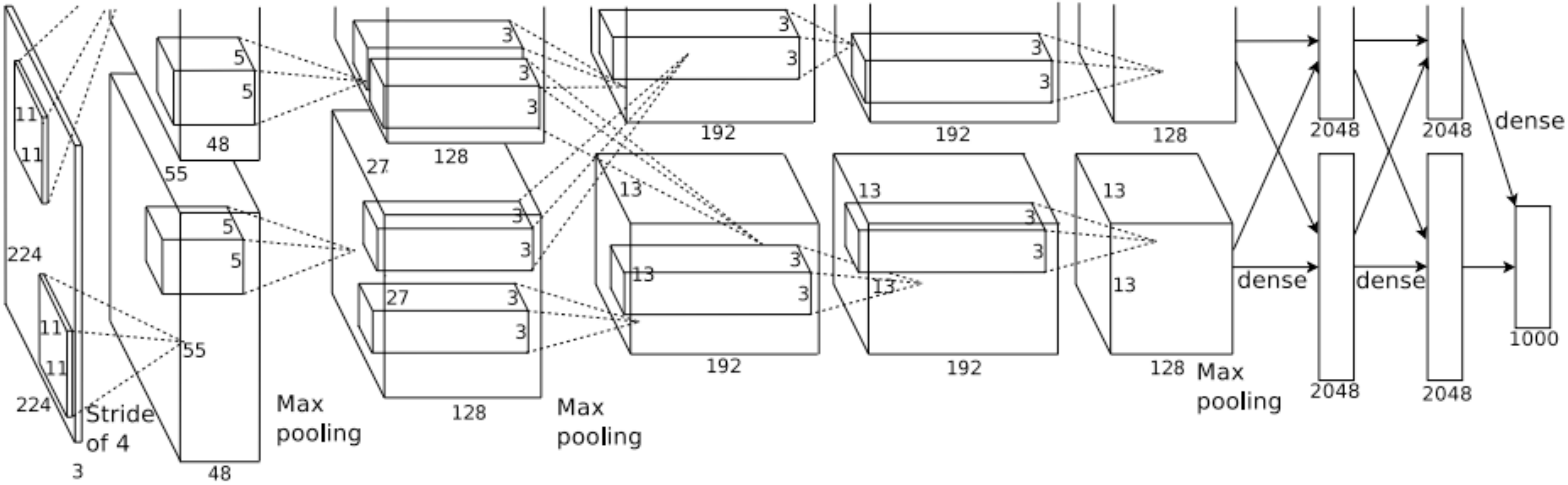
PDP book,
1986

Minsky and Papert,
1972

AI winter,
2000

Krizhevsky, Sutskever, and Hinton, NeurIPS 2012

“Alexnet”



Krizhevsky, Sutskever, and Hinton, NeurIPS 2012

28 years

28 years

Perceptrons,
1958

PDP book,
1986

Krizhevsky,
Sutskever,
Hinton, 2012

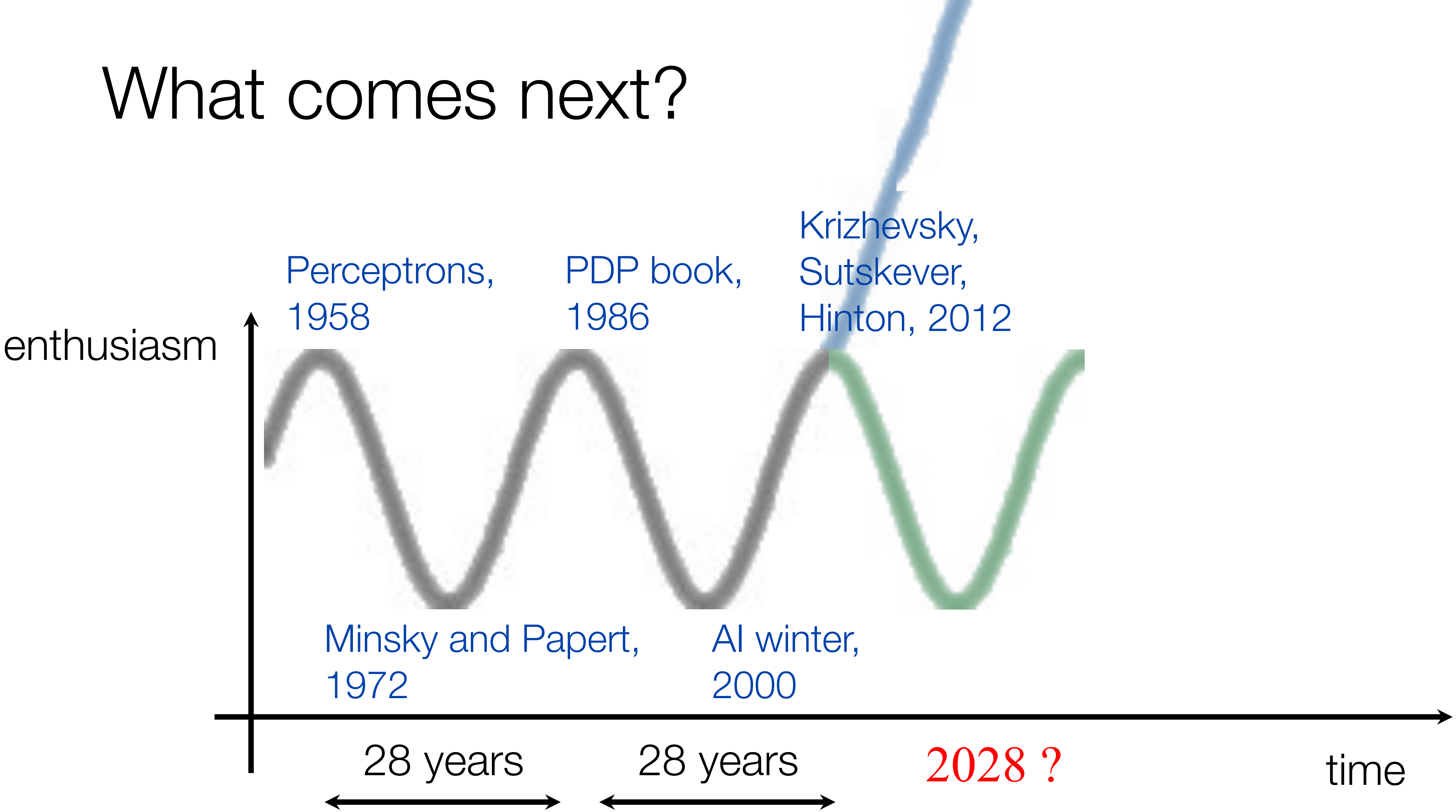
enthusiasm

Minsky and Papert,
1972

AI winter,
2000

time

What comes next?



enthusiasm

Perceptrons, 1958

PDP book, 1986

Krizhevsky, Sutskever, Hinton, 2012

Minsky and Papert, 1972

AI winter, 2000

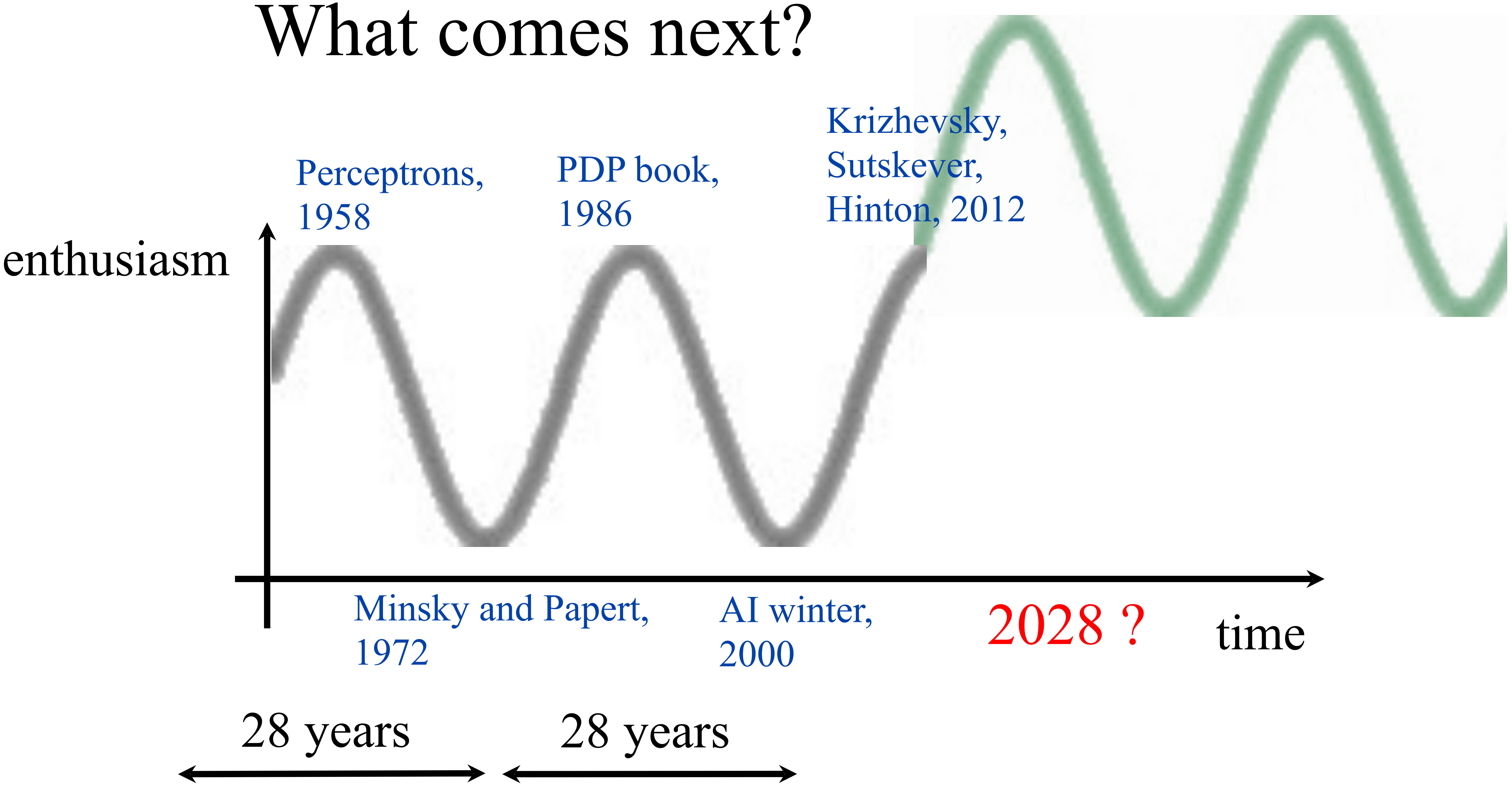
28 years

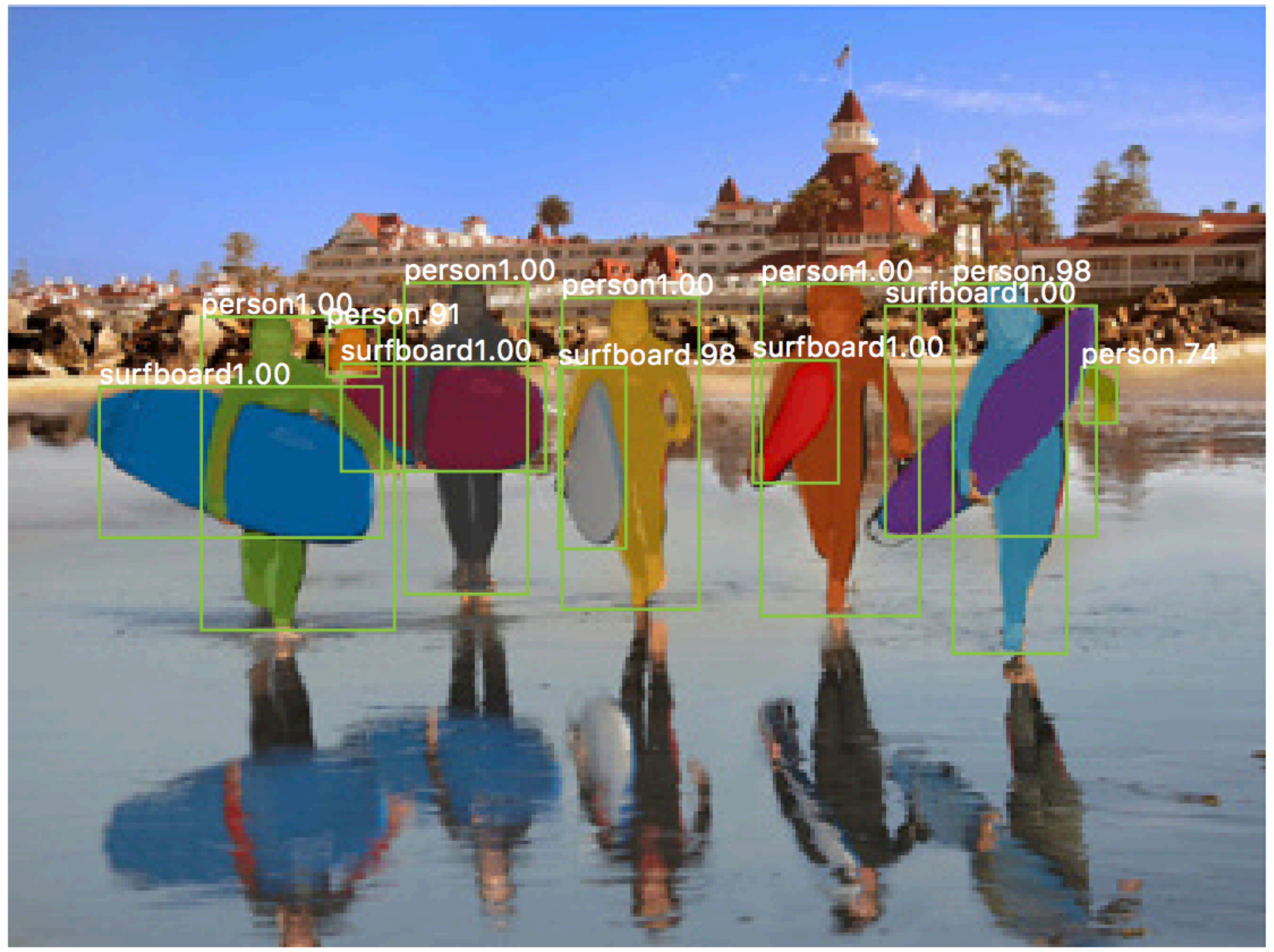
28 years

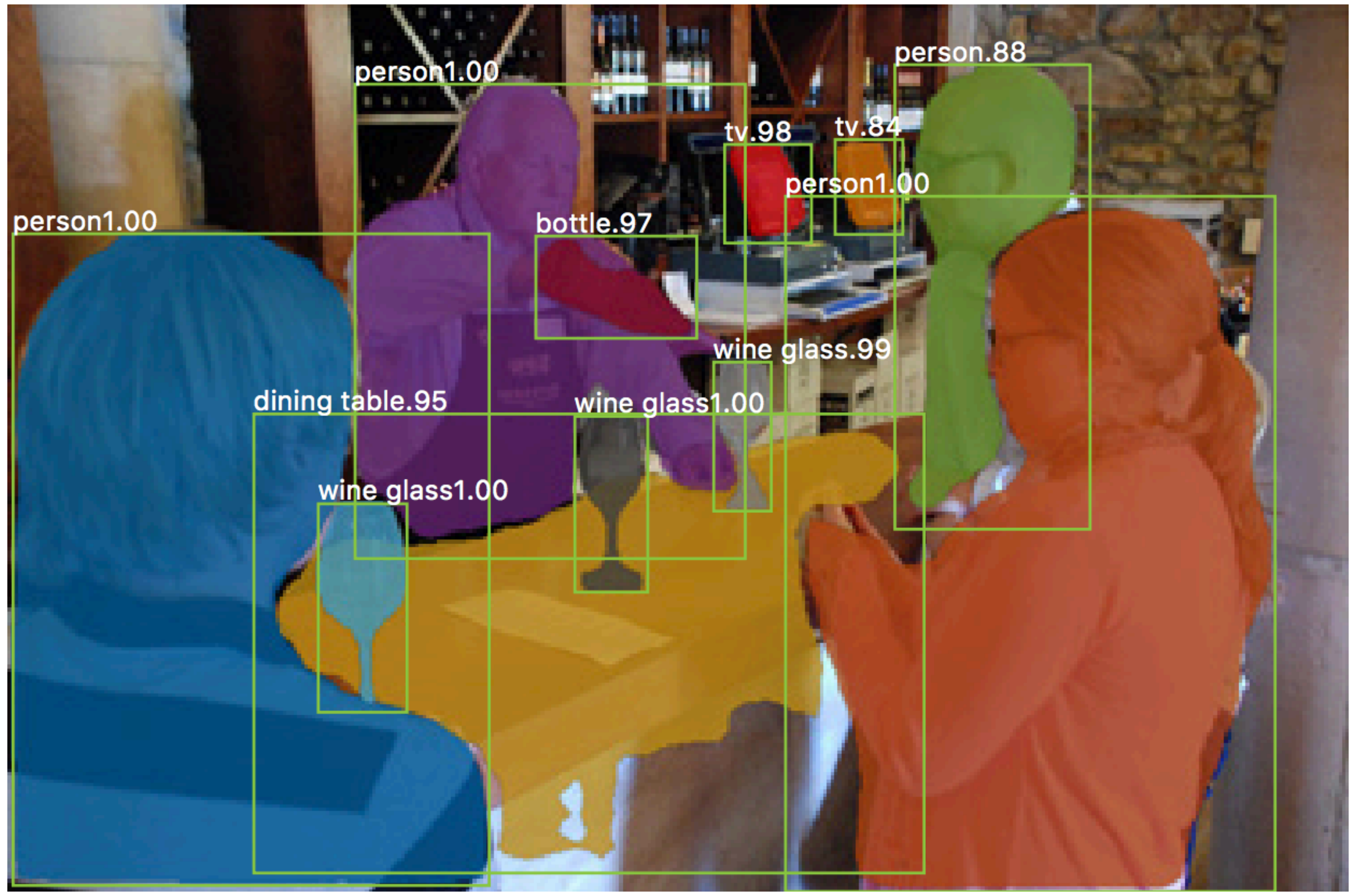
2028 ?

time

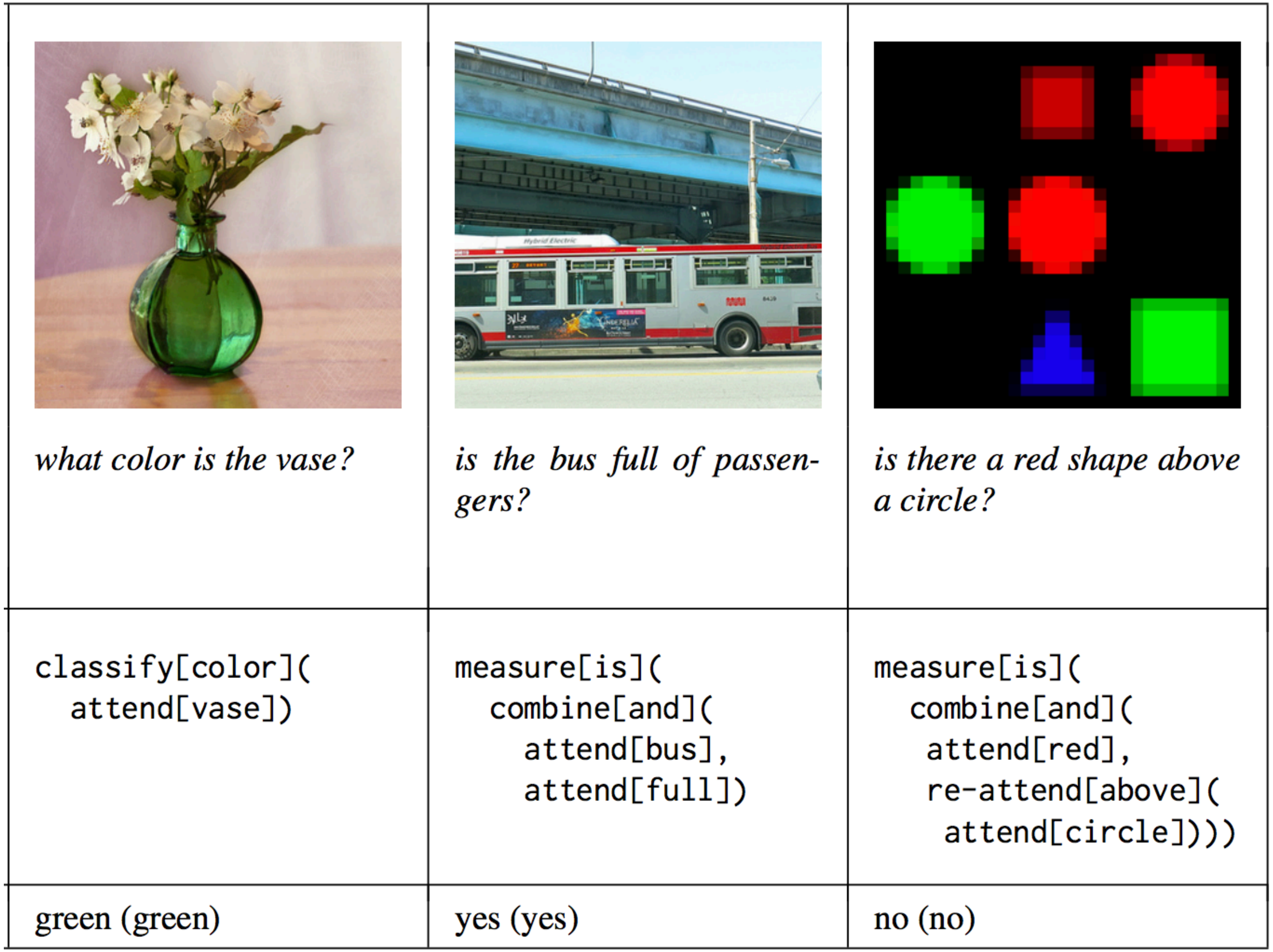
What comes next?







[“Mask RCNN”, He et al. 2017]



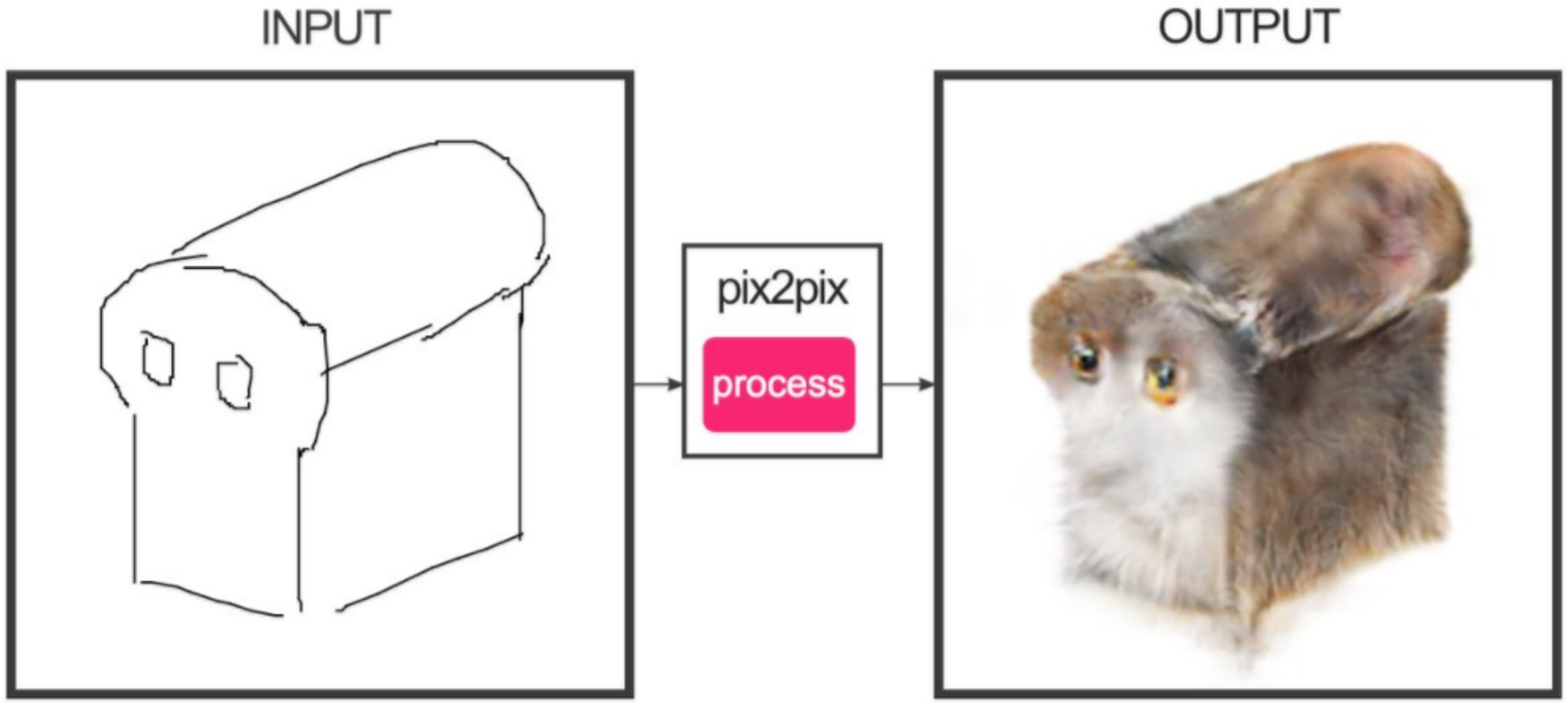
what color is the vase?	*is the bus full of passengers?*	*is there a red shape above a circle?*
green (green)	yes (yes)	no (no)

```
classify[color](
  attend[vase])
```

```
measure[is](
  combine[and](
    attend[bus],
    attend[full])
```

```
measure[is](
  combine[and](
   attend[red],
   re-attend[above](
    attend[circle])))
```

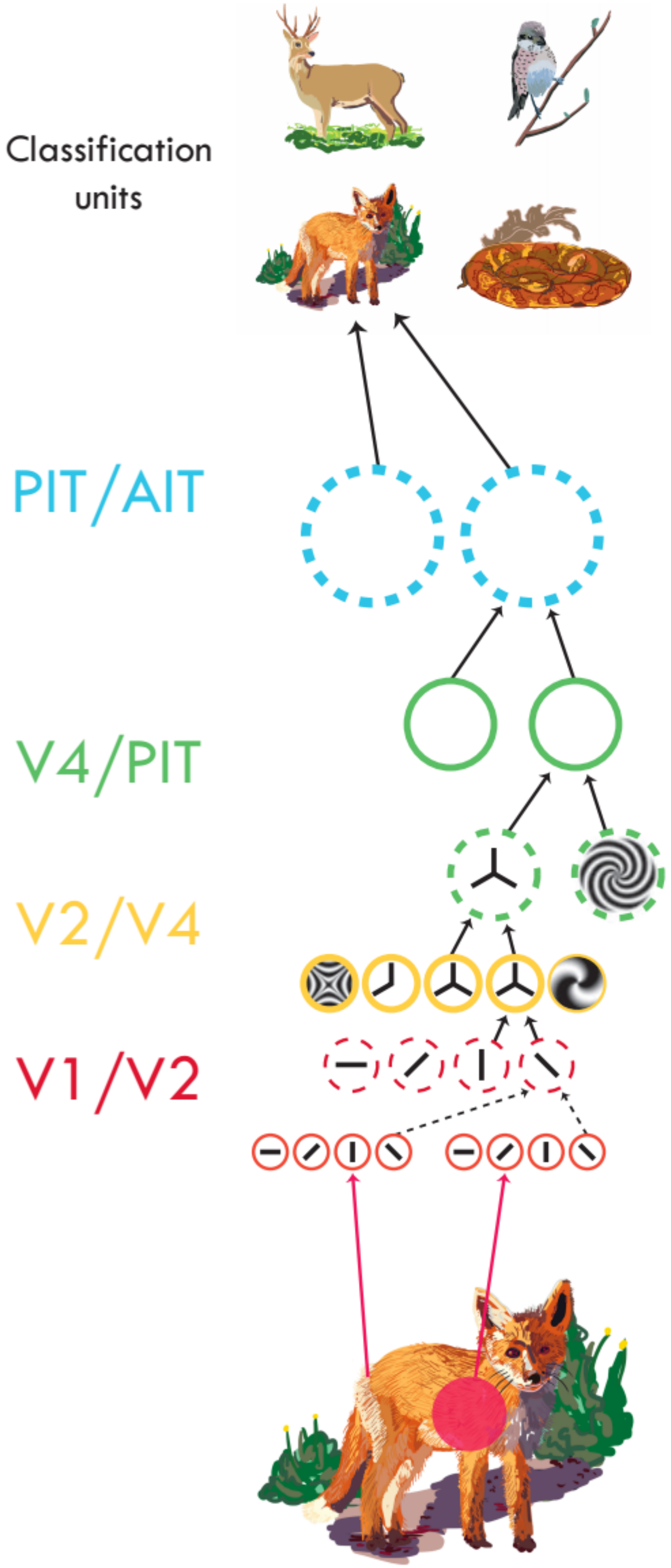
[“Neural module networks”, Andreas et al. 2017]



Ivy Tasi @ivymyt

Vitaly Vidmirov @vvid

[“pix2pix”, Isola et al. 2017]



Classification units

PIT/AIT

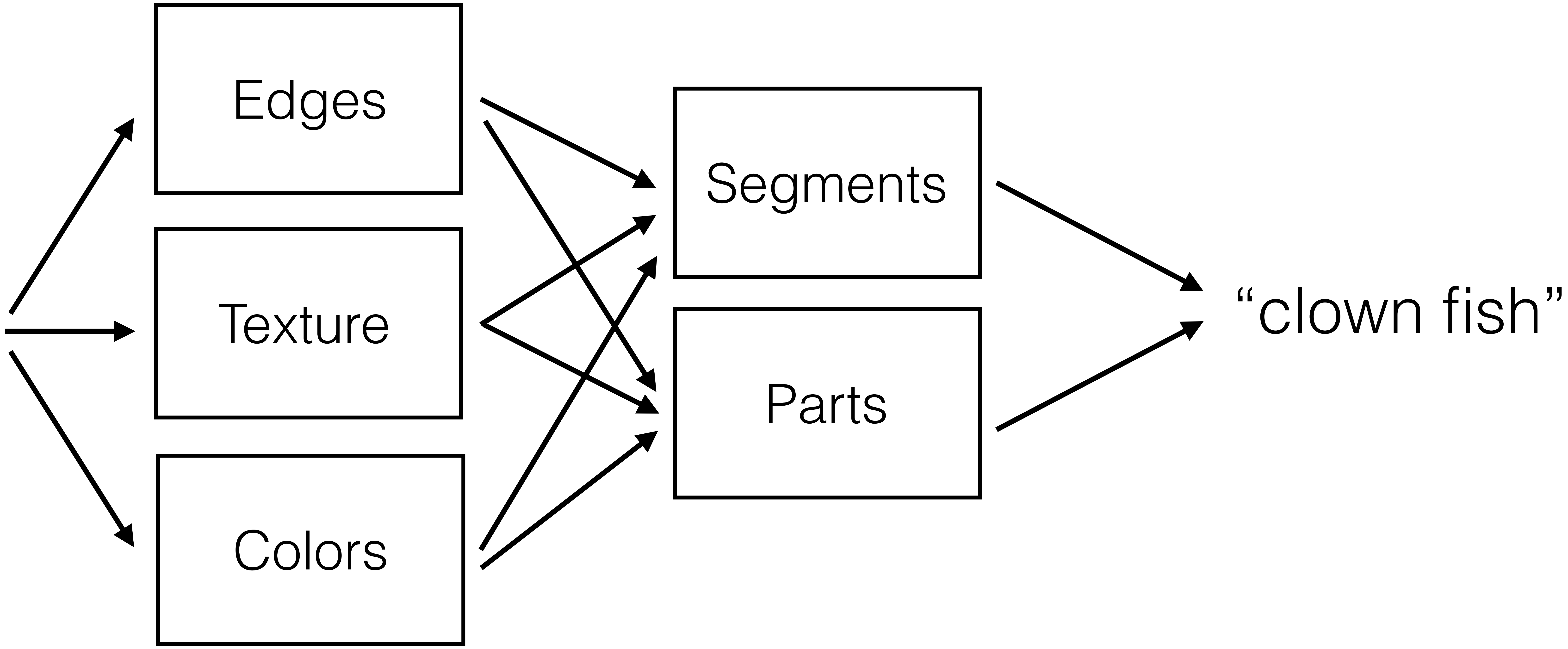
V4/PIT

V2/V4

V1/V2

Serre, 2014

Image classification



Edges

Texture

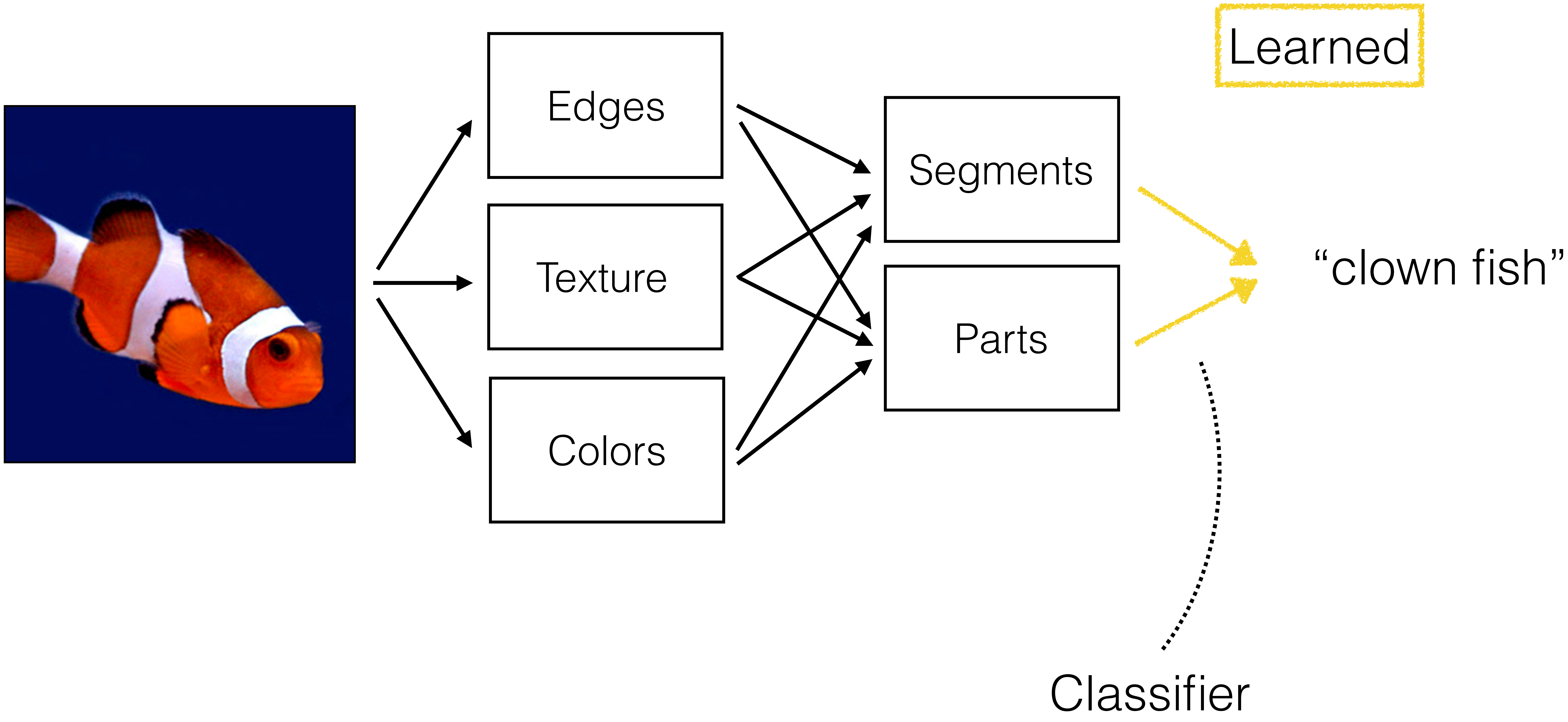
Colors

Segments

Parts

“clown fish”

Image classification



Learned

Edges

Texture

Colors

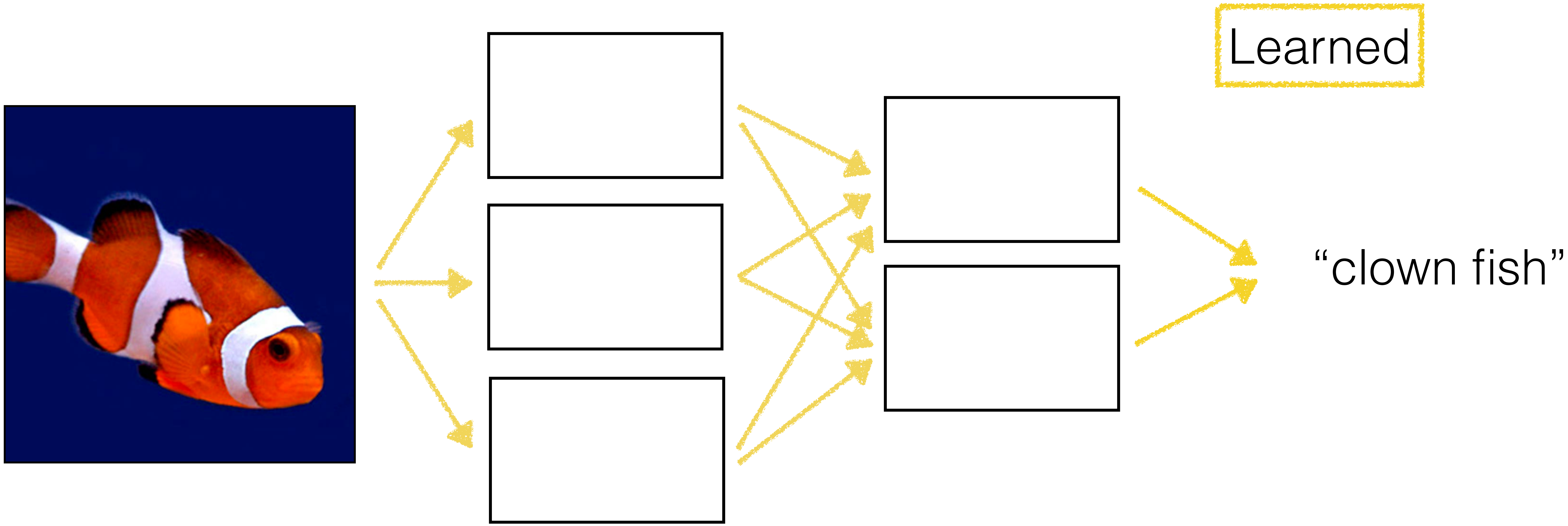
Segments

Parts

“clown fish”

Classifier

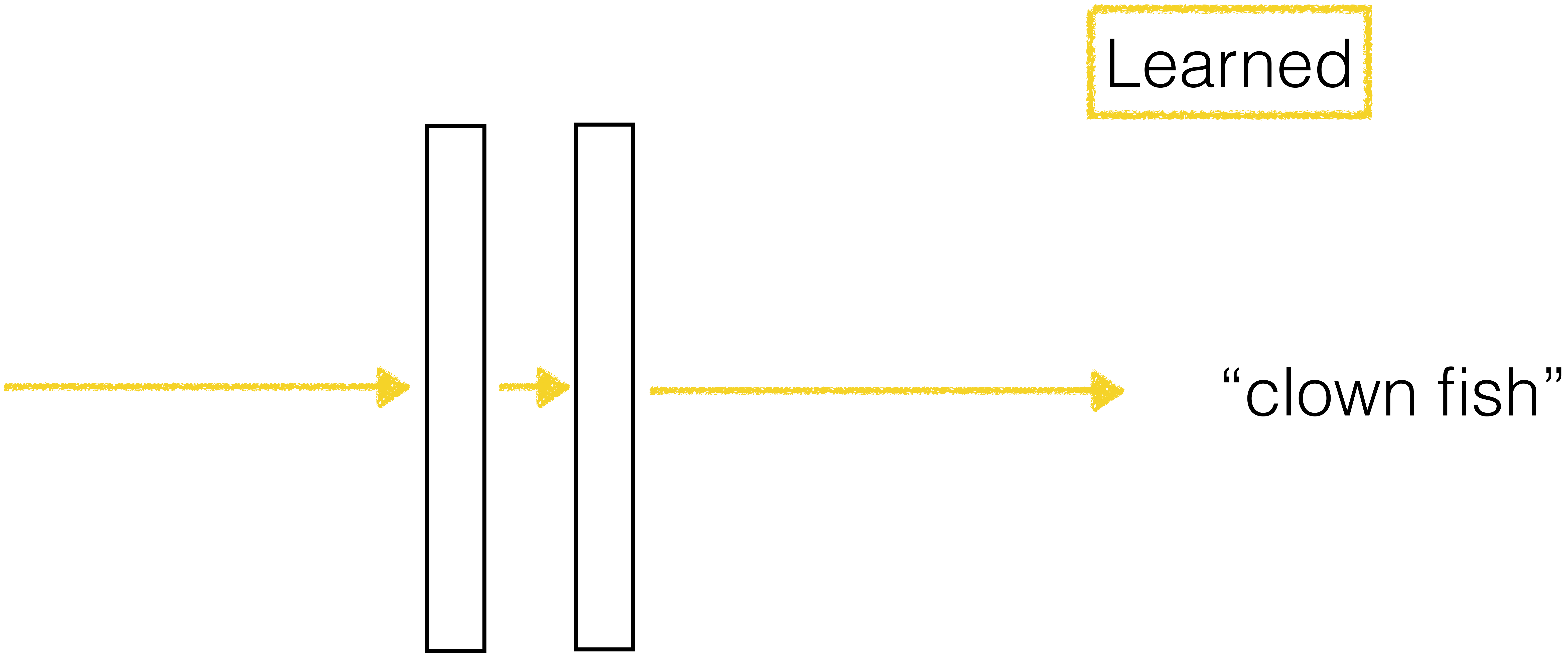
Image classification



Learned

"clown fish"

Image classification

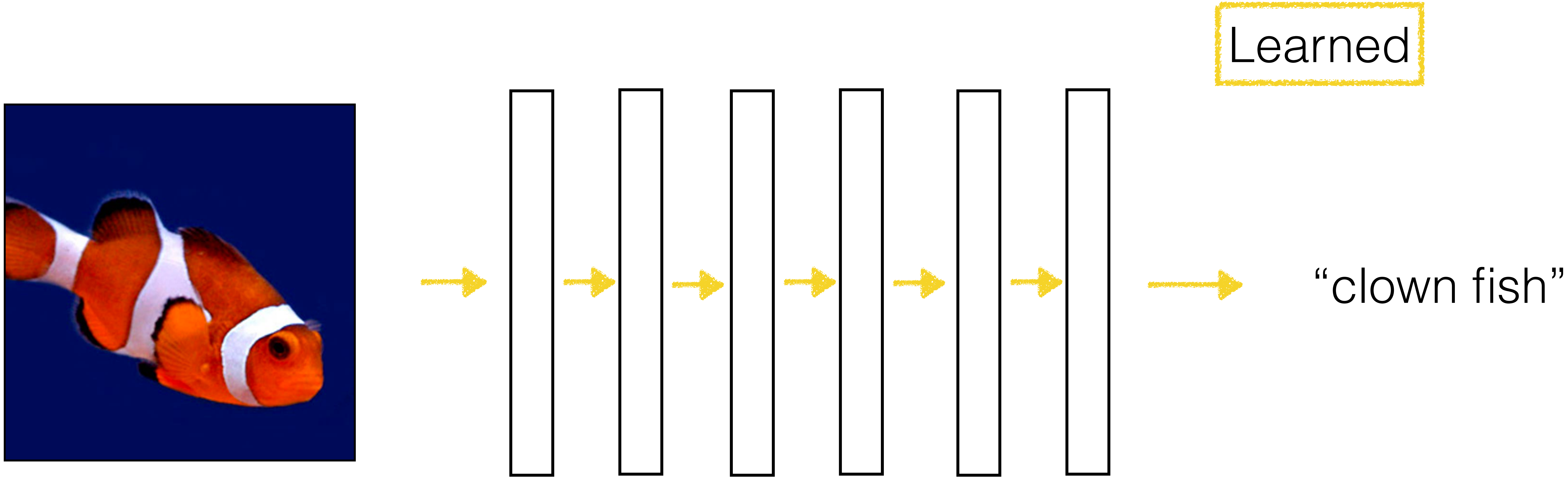


Learned

"clown fish"

Neural net

Image classification

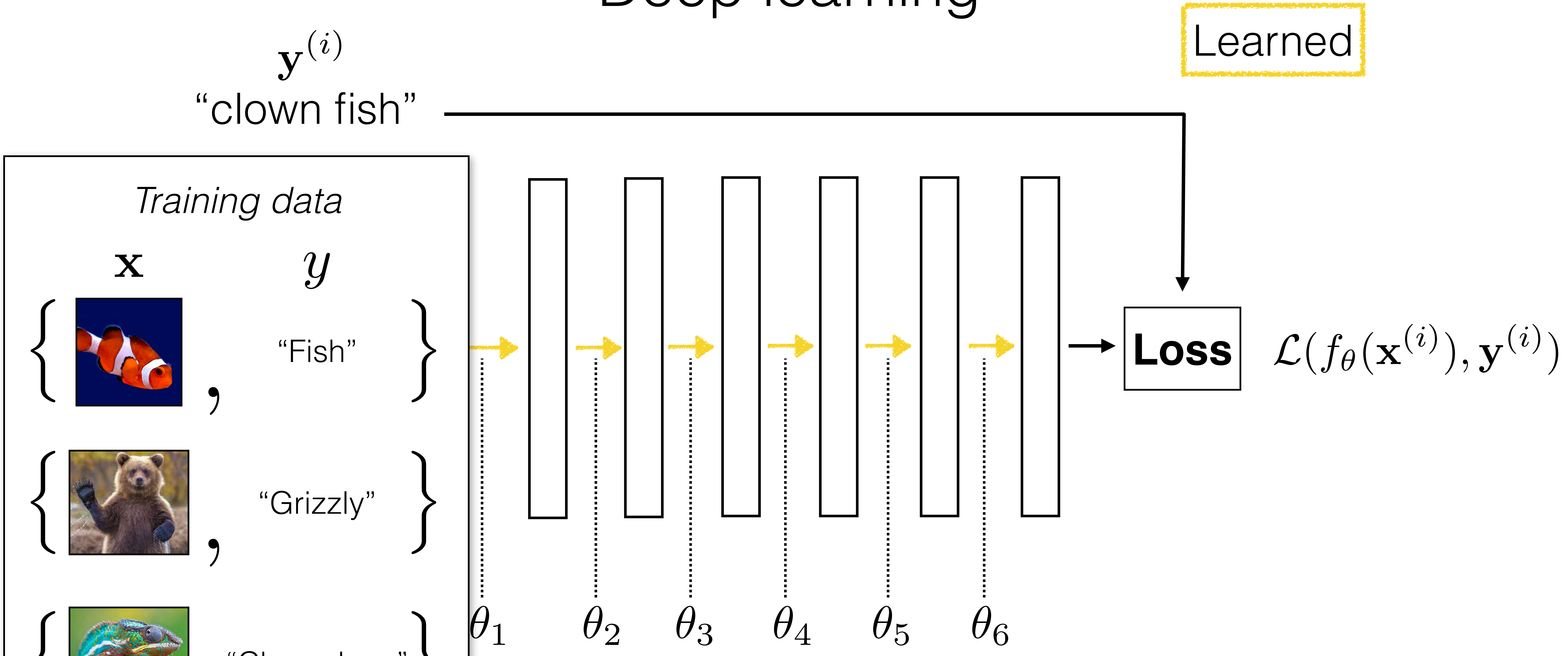


Learned

“clown fish”

Deep neural net

Deep learning



Learned

$\mathbf{y}^{(i)}$

"clown fish"

Training data

$\mathbf{x}$ y

$\{$, "Fish" $\}$

$\{$, "Grizzly" $\}$

$\{$, "Chameleon" $\}$

$\vdots$

Loss

$\mathcal{L}(f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$

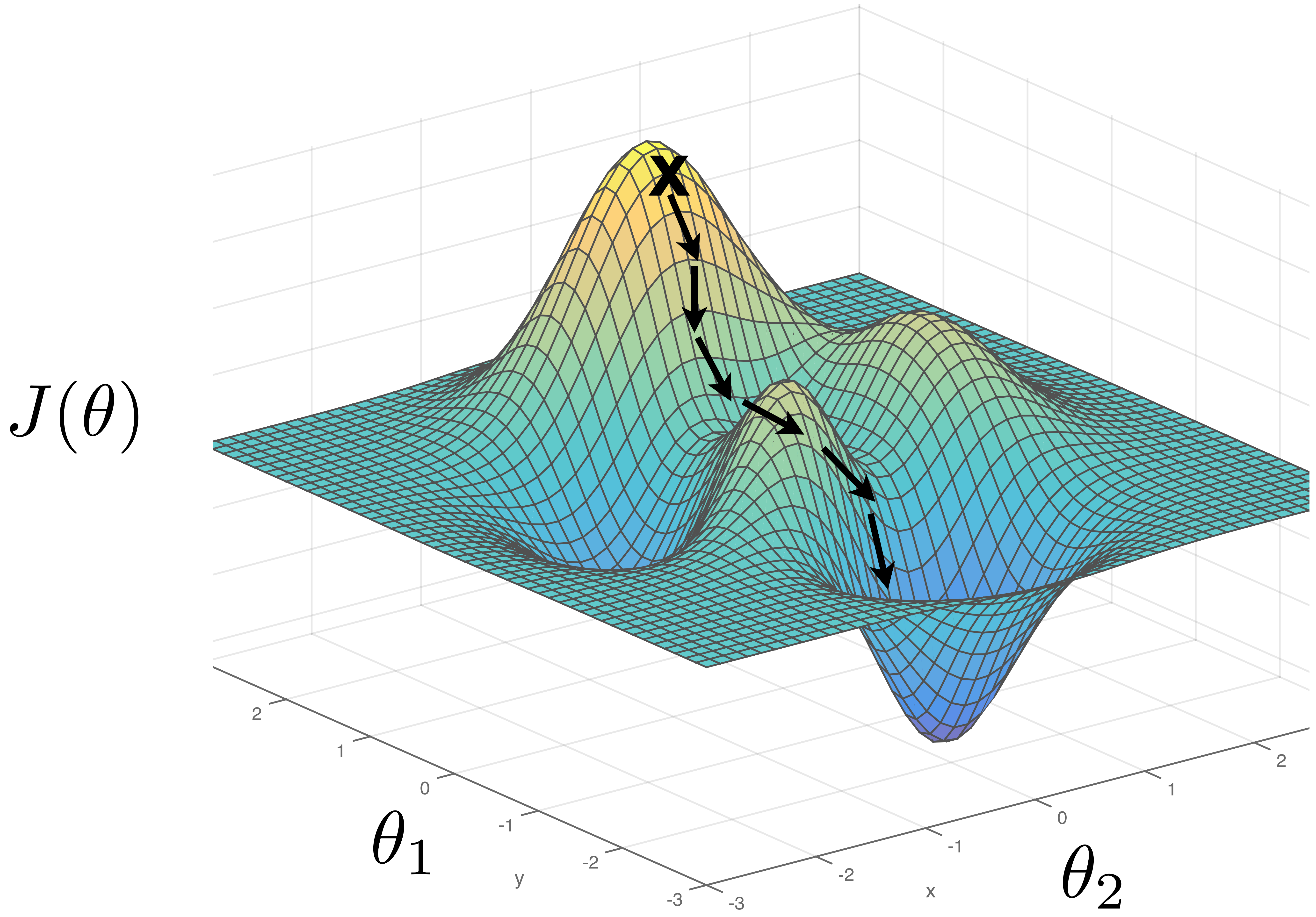
θ_1 θ_2 θ_3 θ_4 θ_5 θ_6

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

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

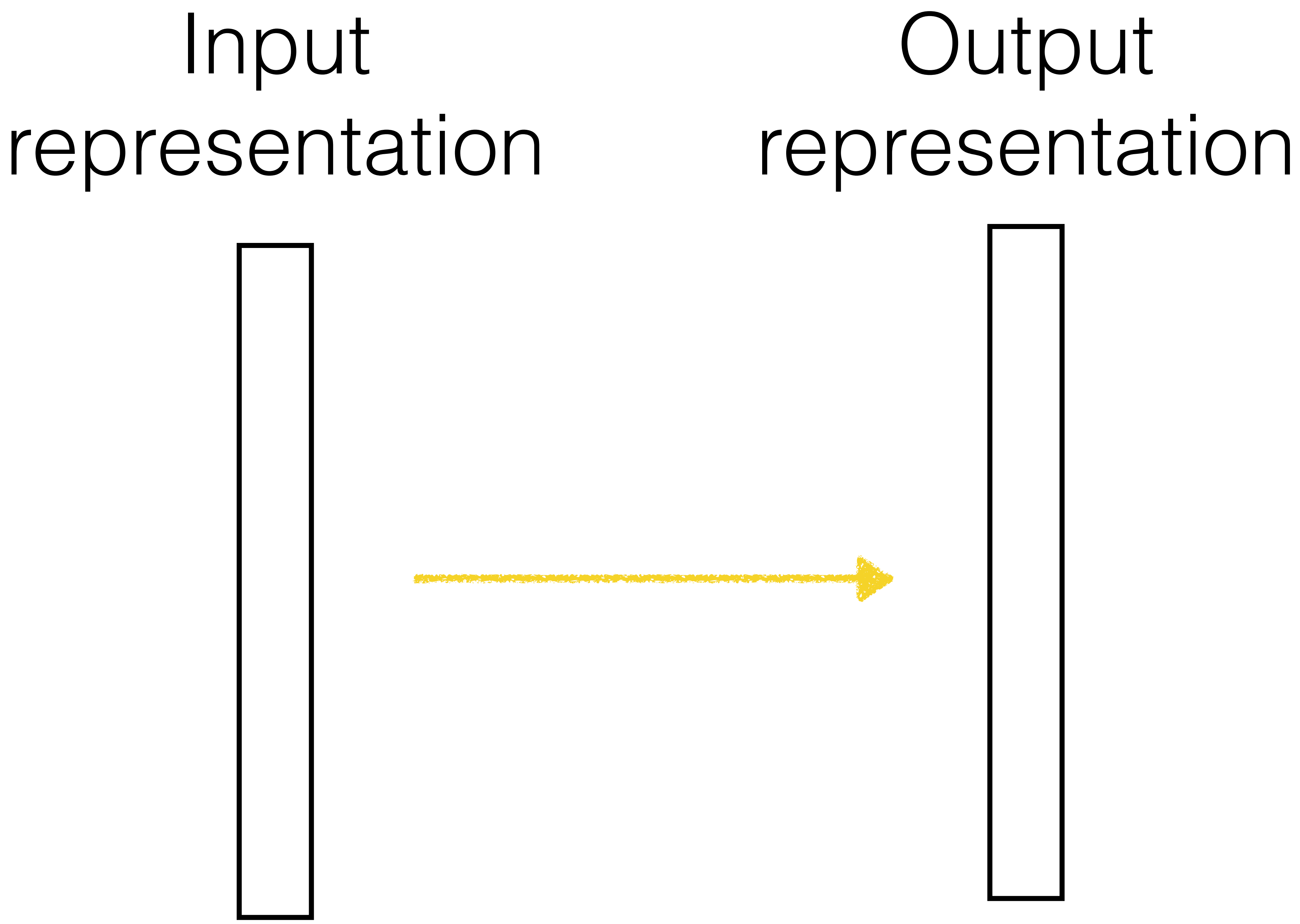
Stochastic gradient descent (SGD)

- 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

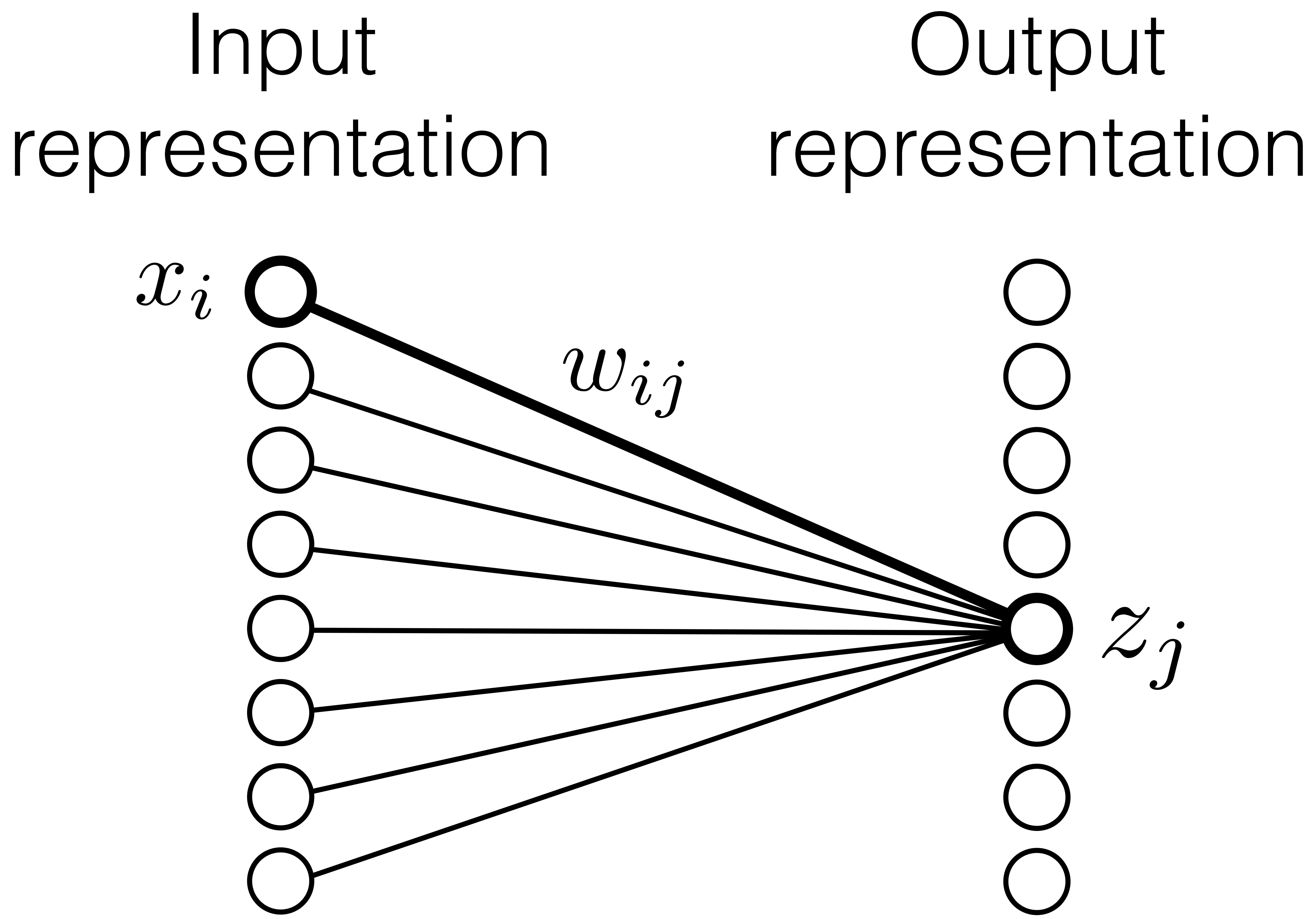
Computation in a neural net


Input representation

Output representation

Computation in a neural net

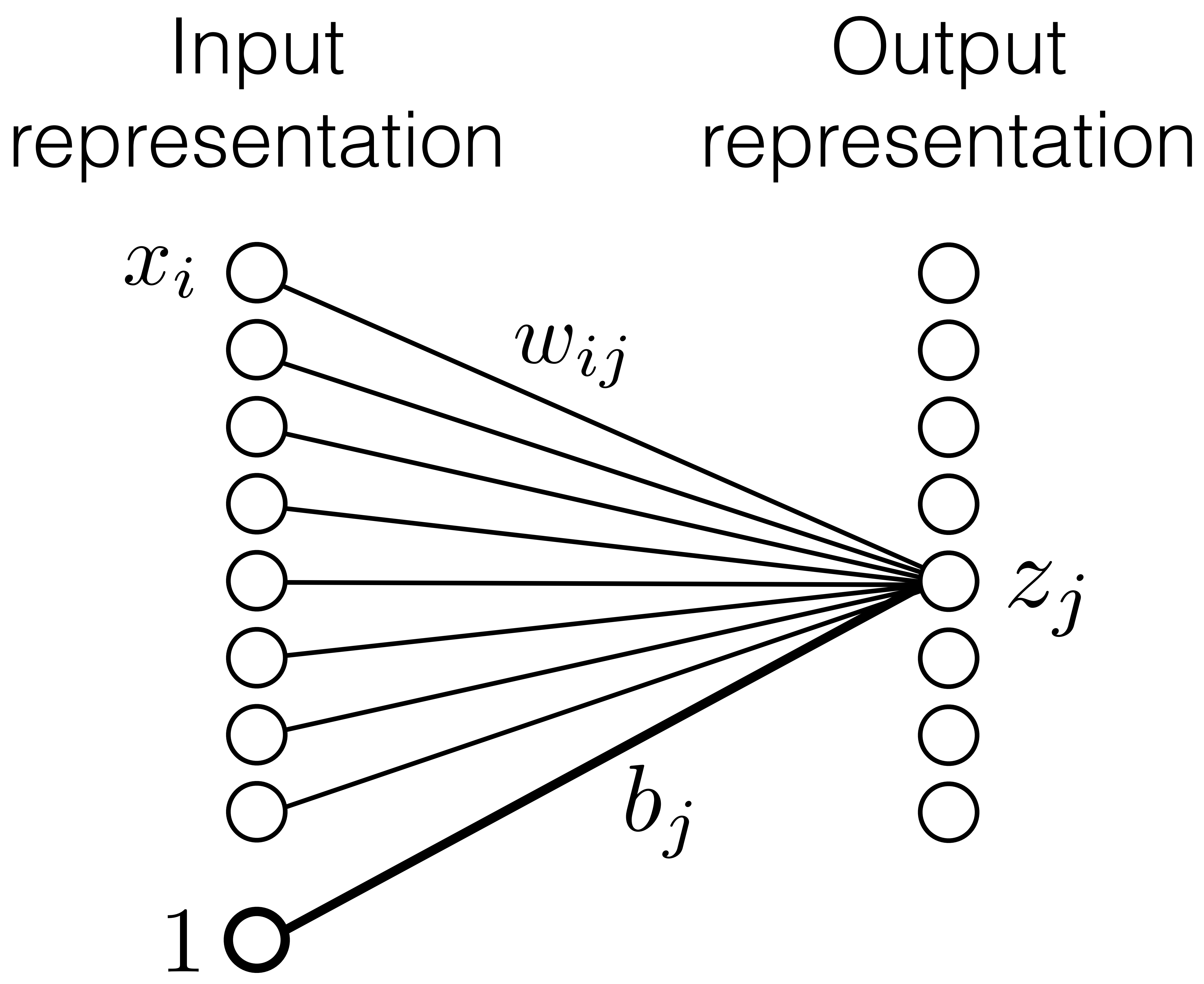
Linear layer



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

Computation in a neural net

Linear layer

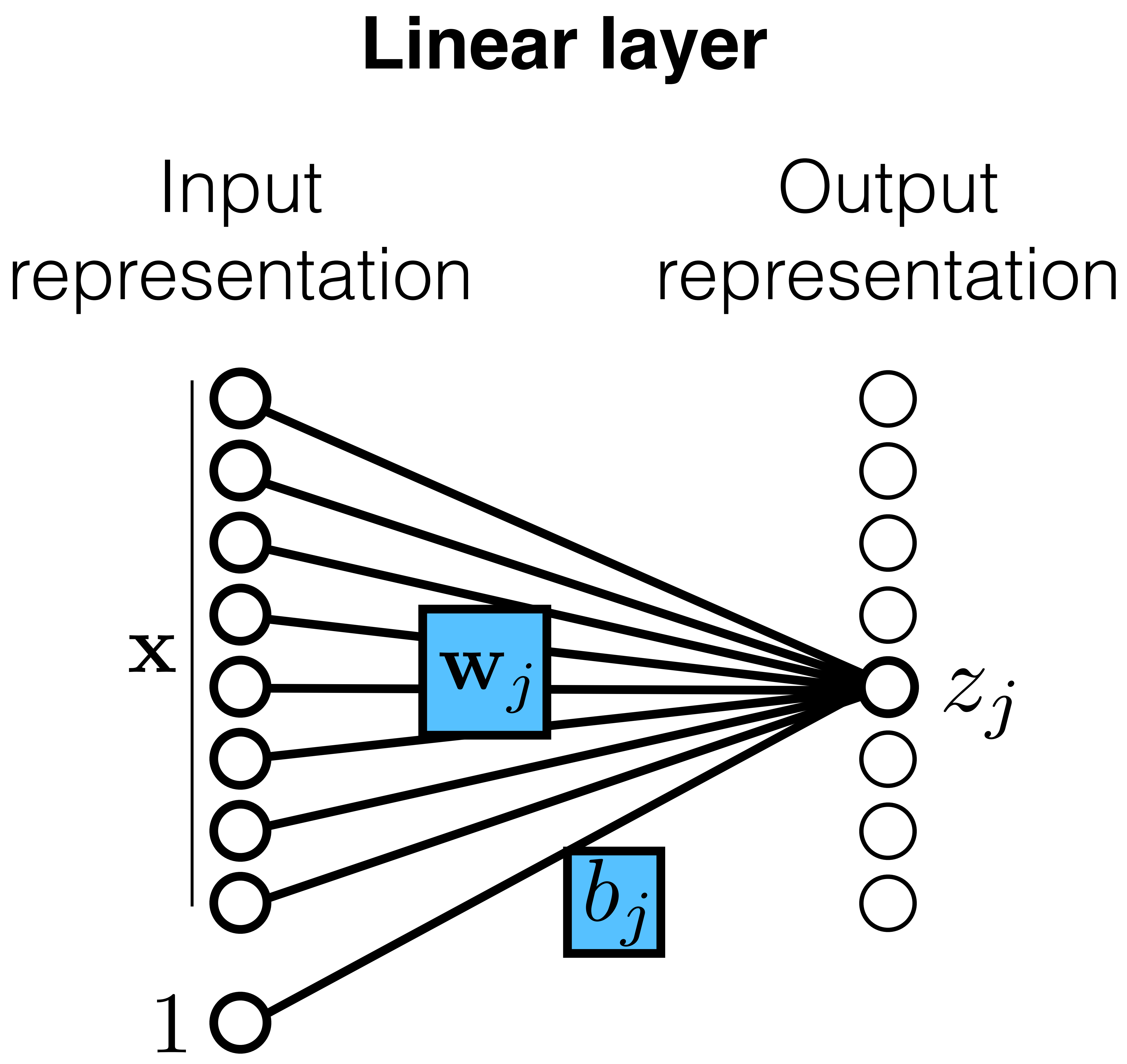


$$z_j = \sum_i w_{ij} x_i + b_j$$

weights

bias

Computation in a neural net



weights

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

bias

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

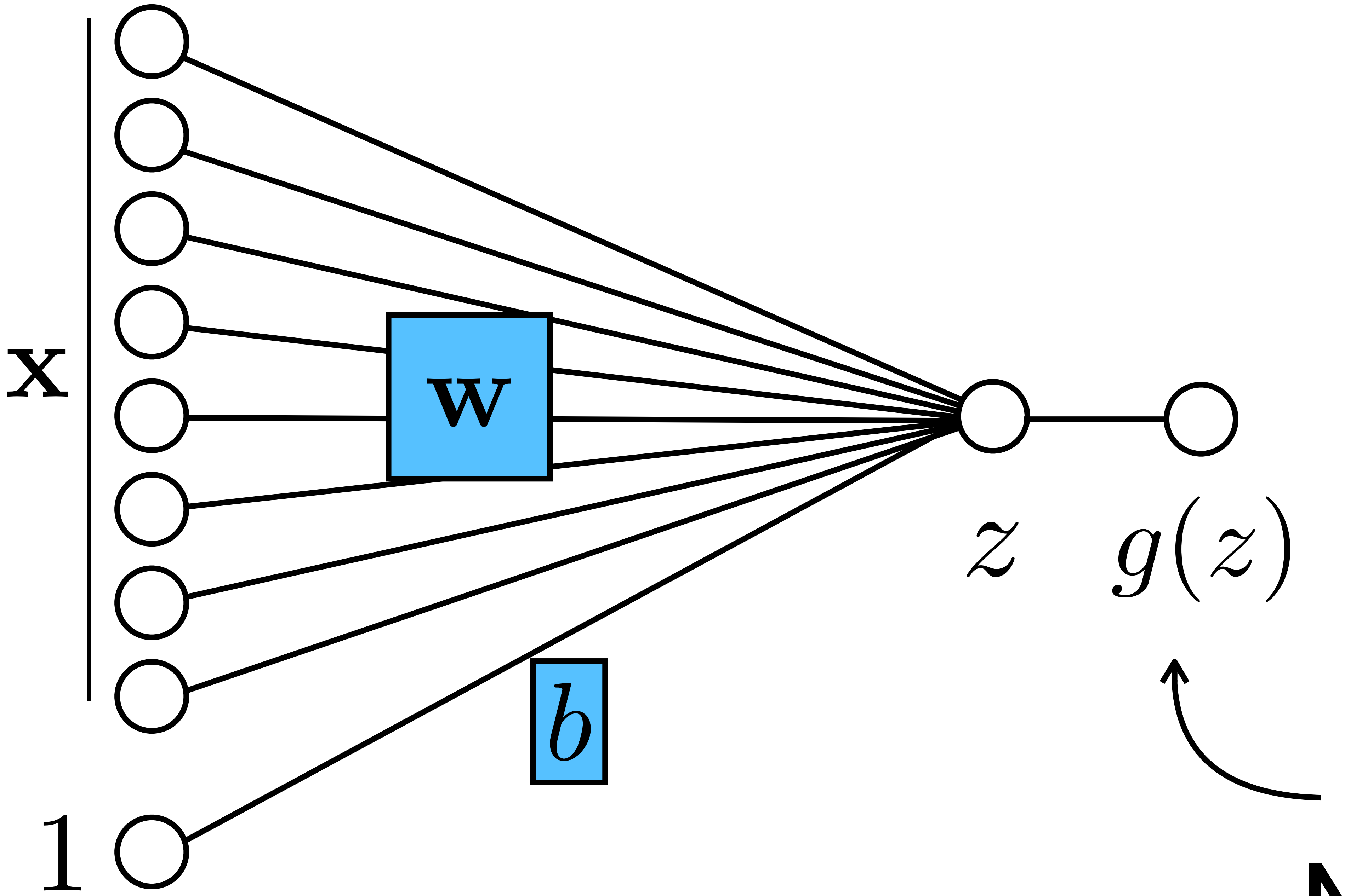
parameters of the model

Computation in a neural net

“Perceptron”

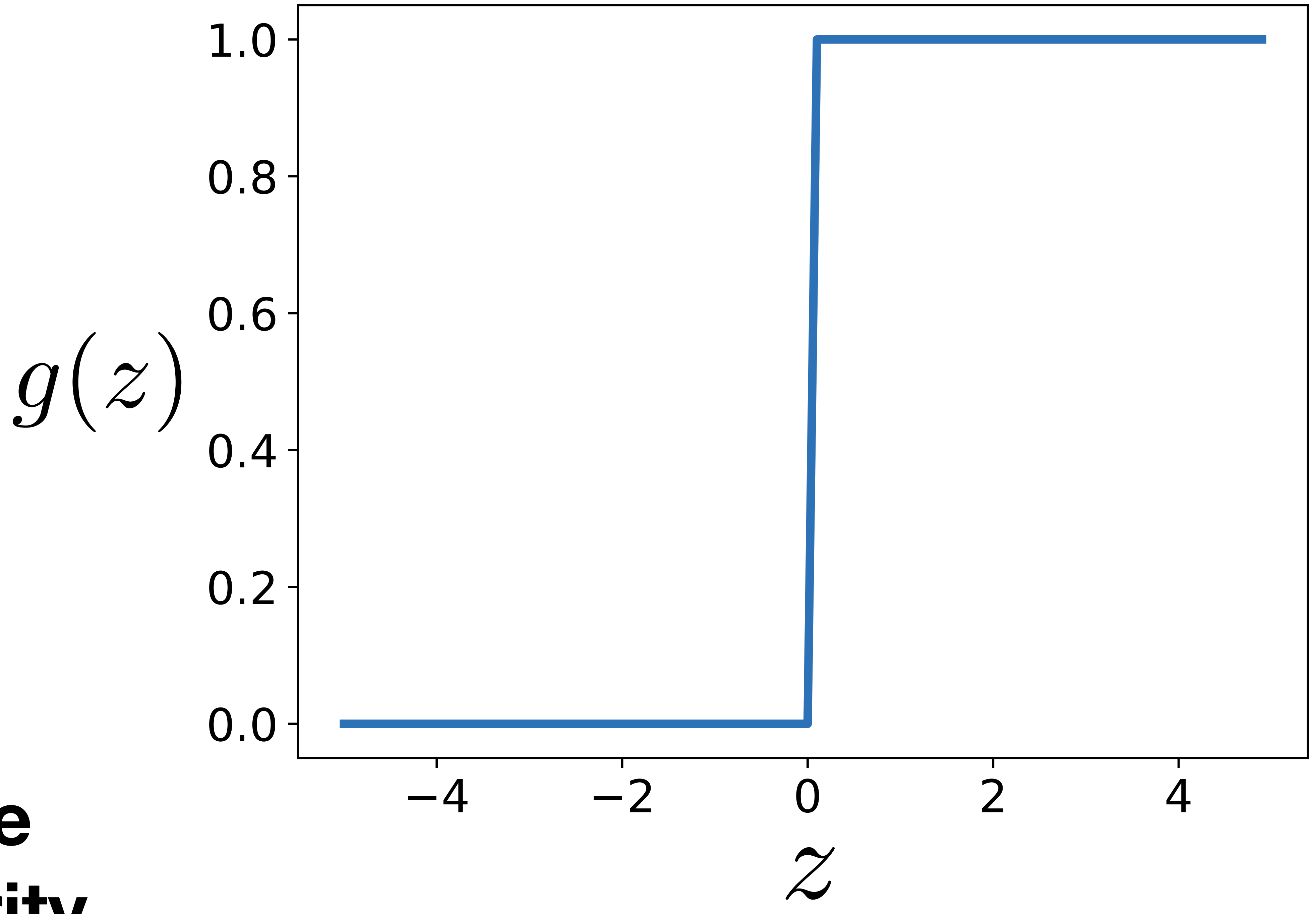
Input representation

Output representation

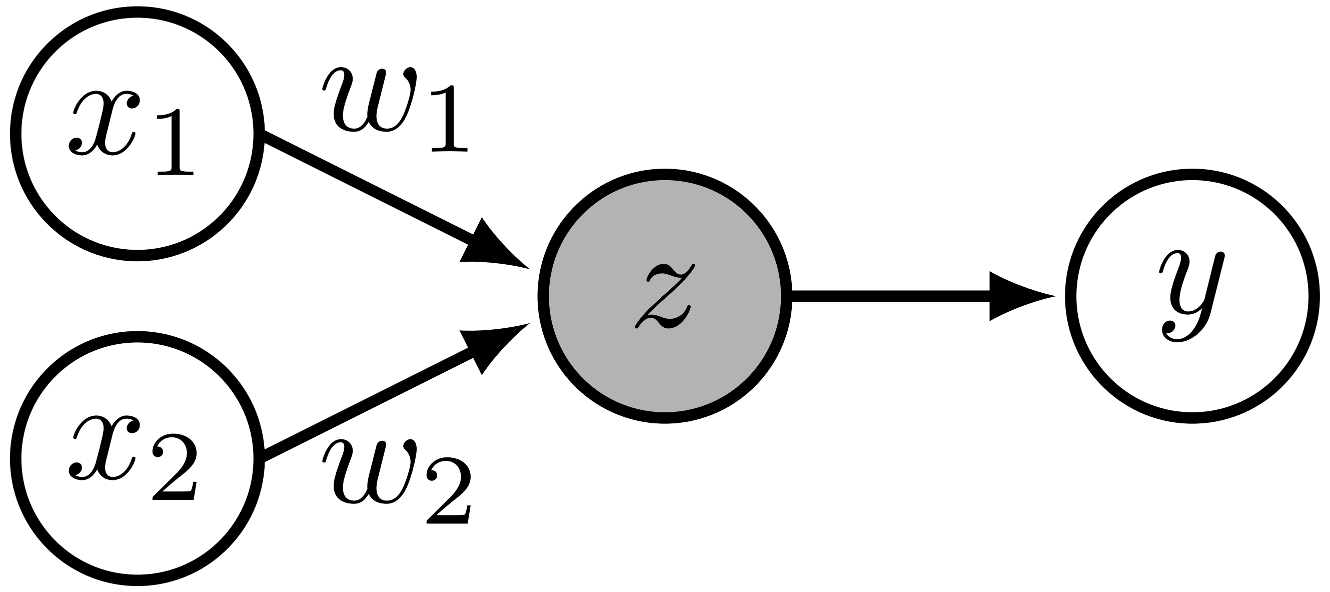


Pointwise Non-linearity

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

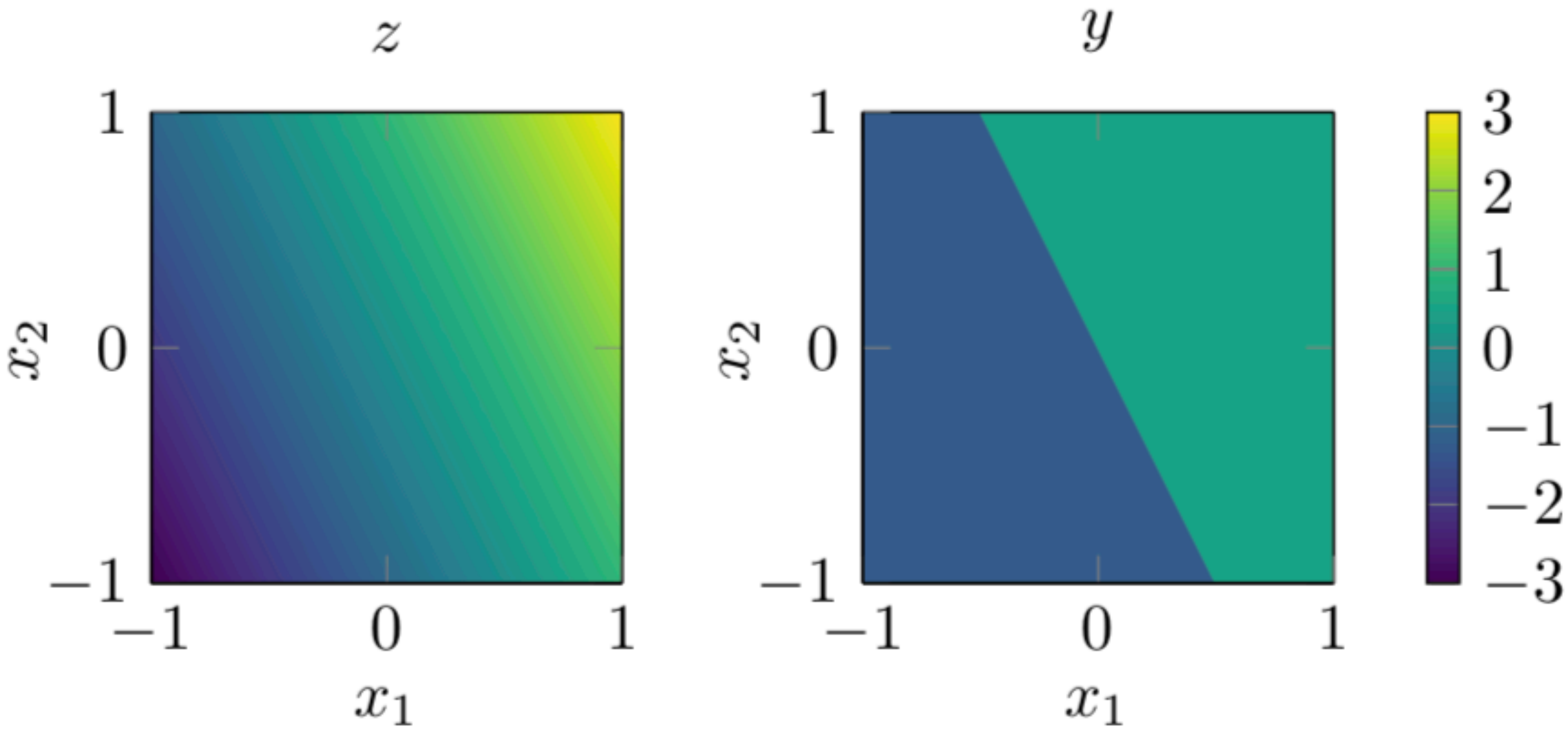


Example: linear classification with a perceptron

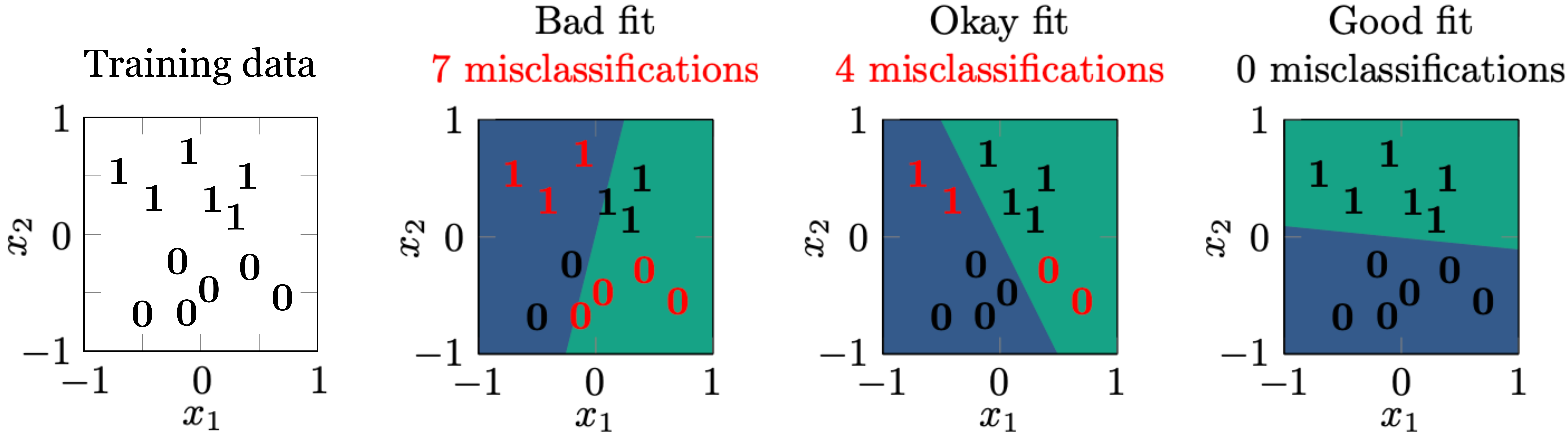


$$z = \mathbf{x}^T \mathbf{w} + b$$

$$y = g(z)$$



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



Training data

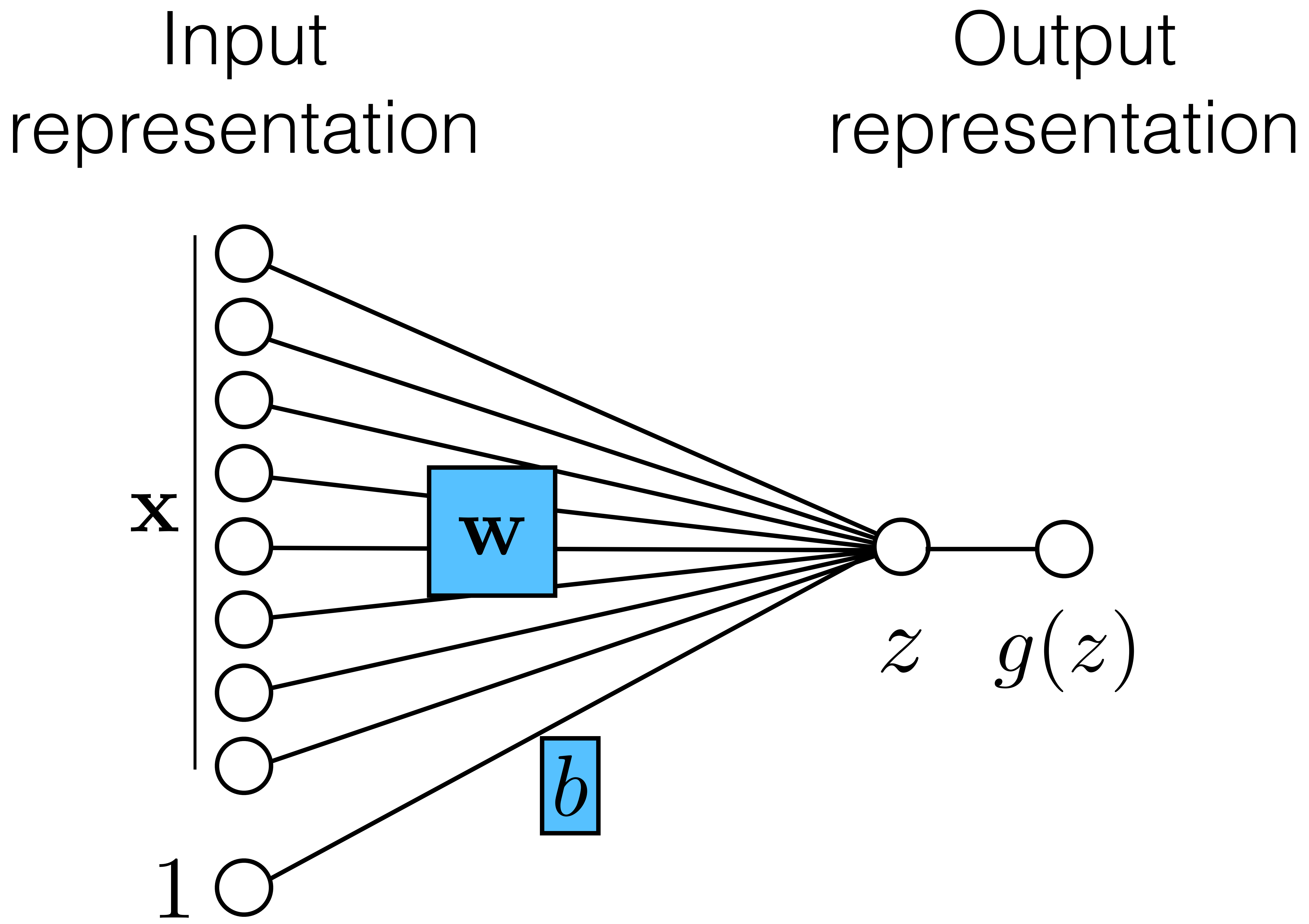
Bad fit
7 misclassifications

Okay fit
4 misclassifications

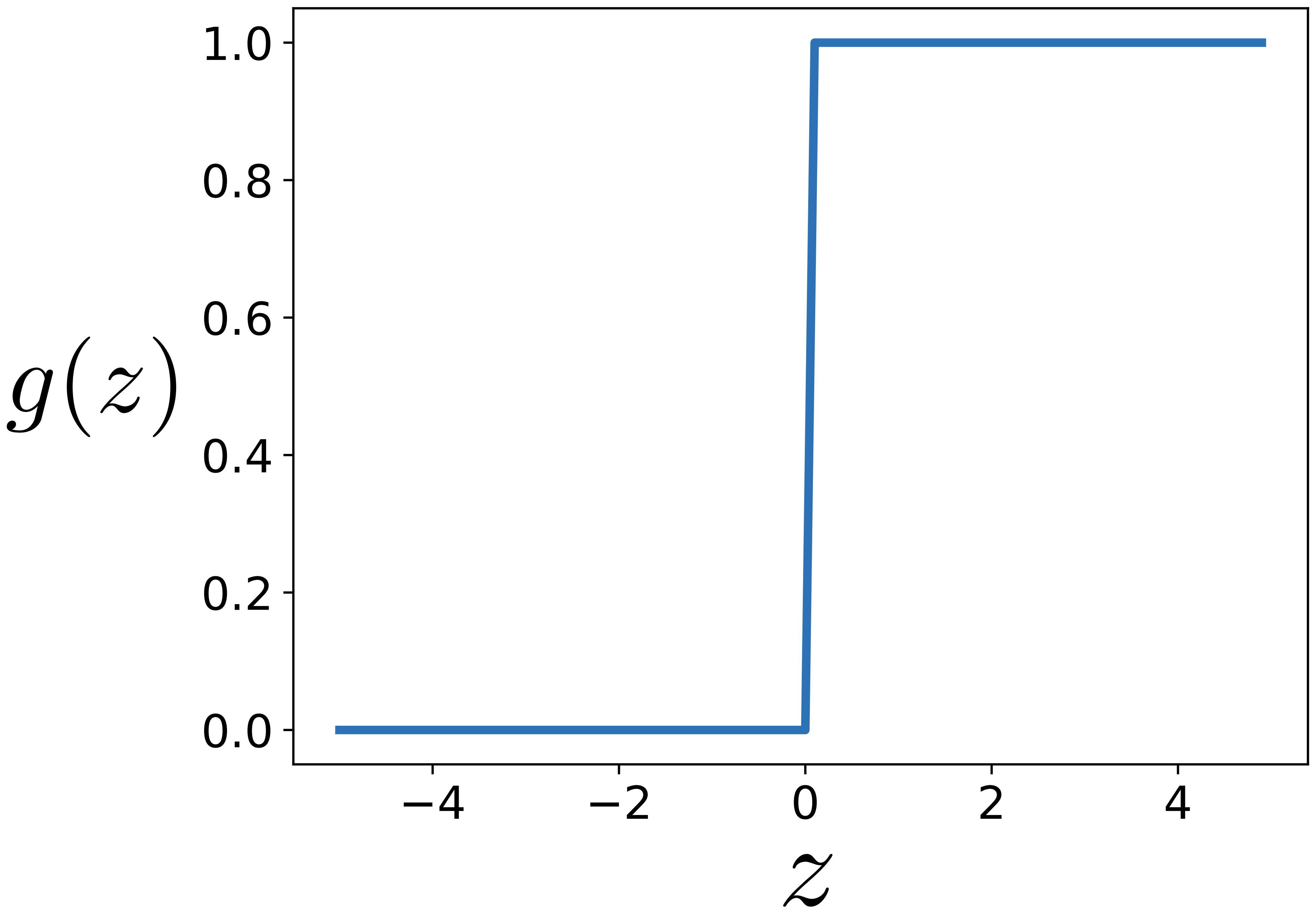
Good fit
0 misclassifications

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

Computation in a neural net



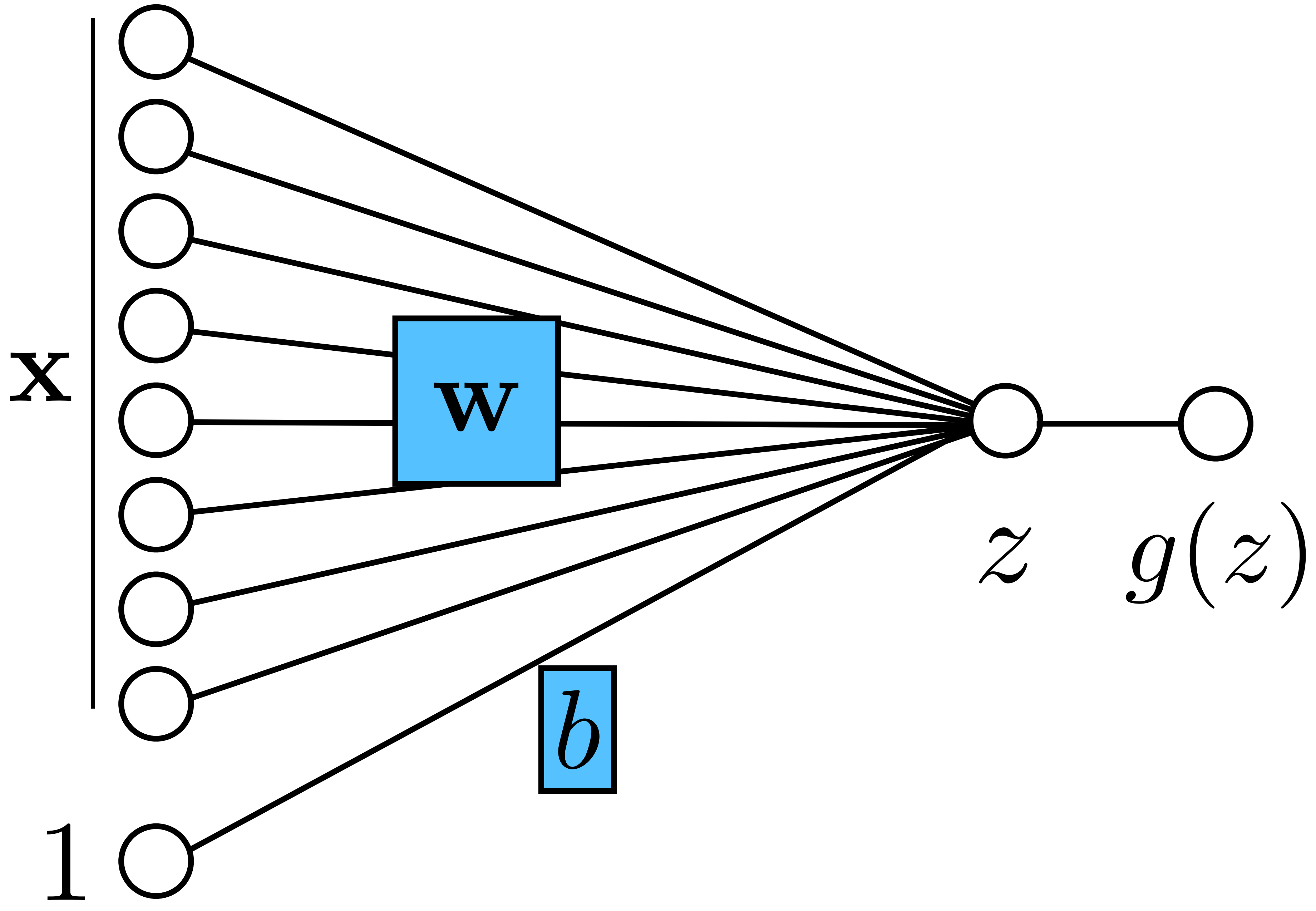
$$g(z) = \begin{cases} 1, & \text{if} \quad z > 0 \\ 0, & \text{otherwise} \end{cases}$$



Computation in a neural net — nonlinearity

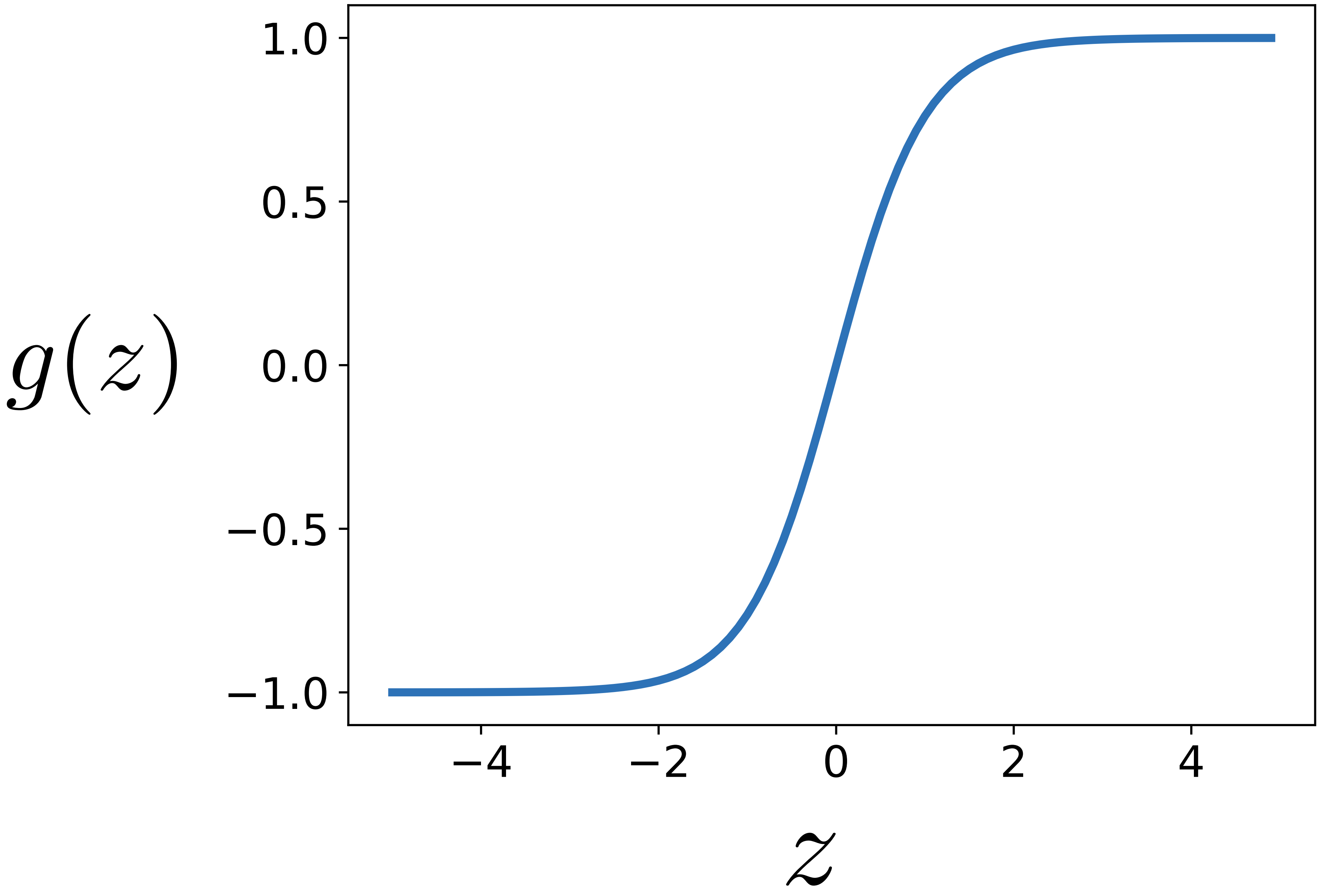
Input representation

Output representation



Tanh

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

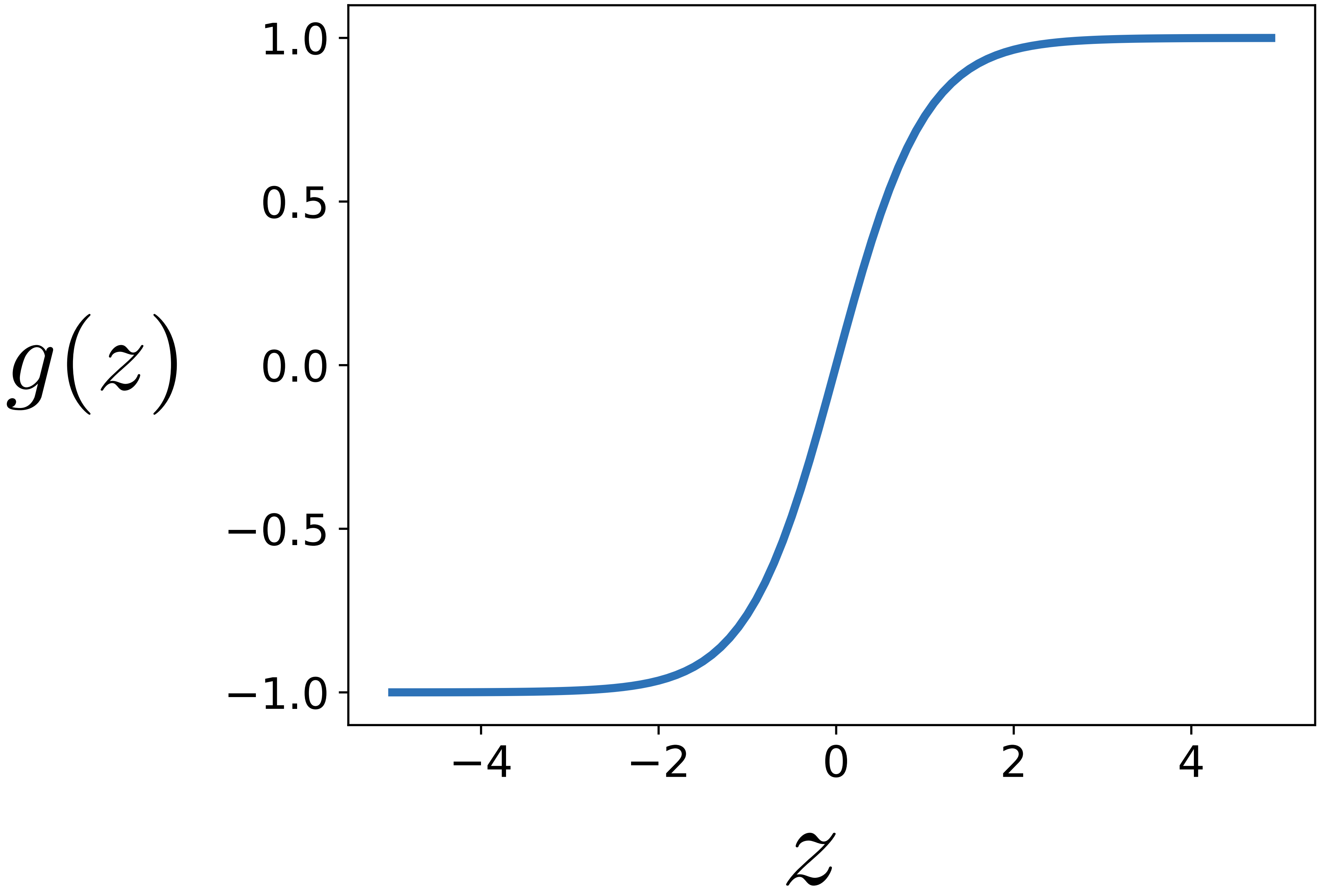


Computation in a neural net — nonlinearity

- Bounded between [-1,+1]
- Saturation for large +/- inputs
- Gradients go to zero
- Outputs centered at 0
- tanh(z) = 2 sigmoid(2z) –1

Tanh

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

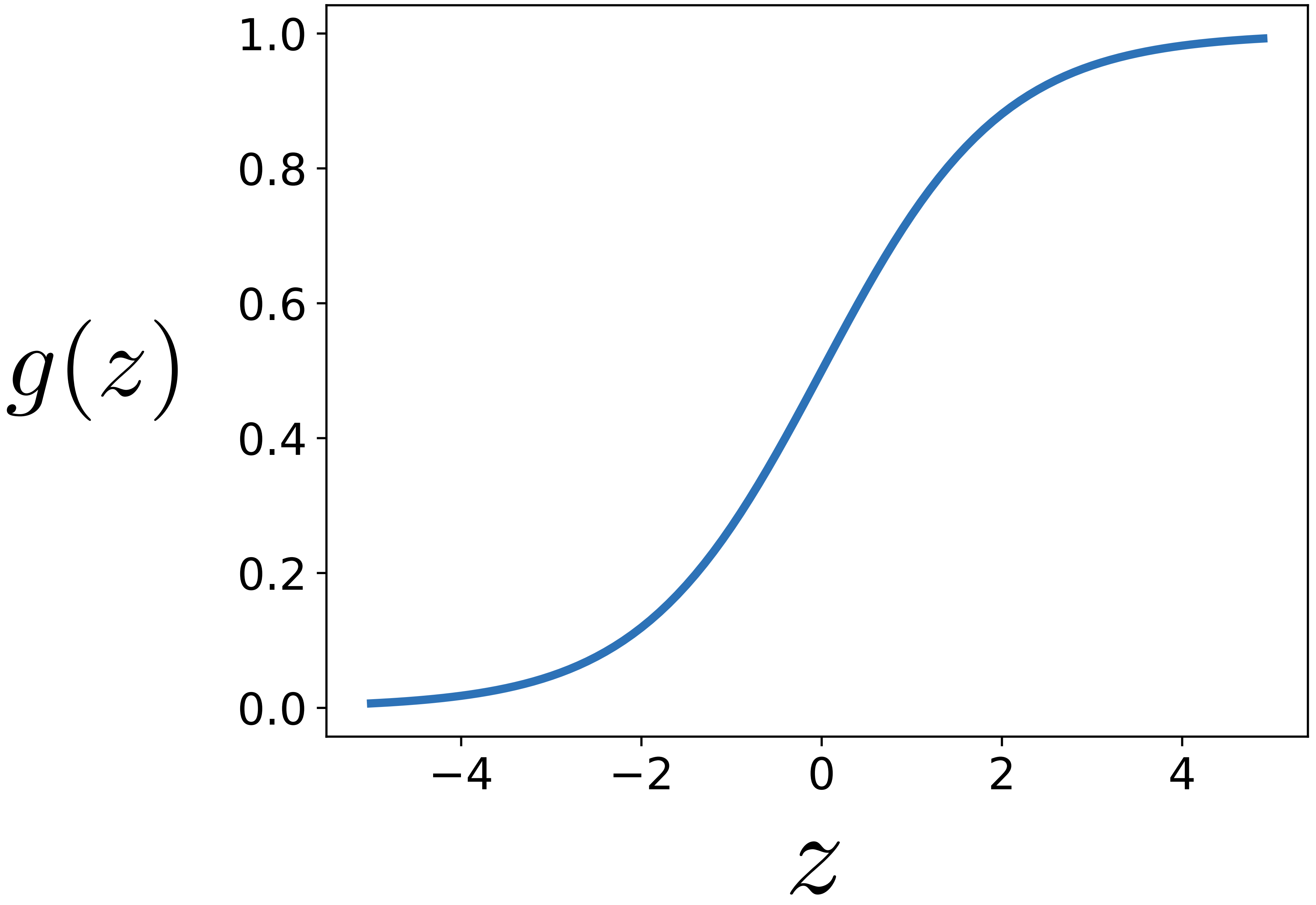


Computation in a neural net — nonlinearity

- 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

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

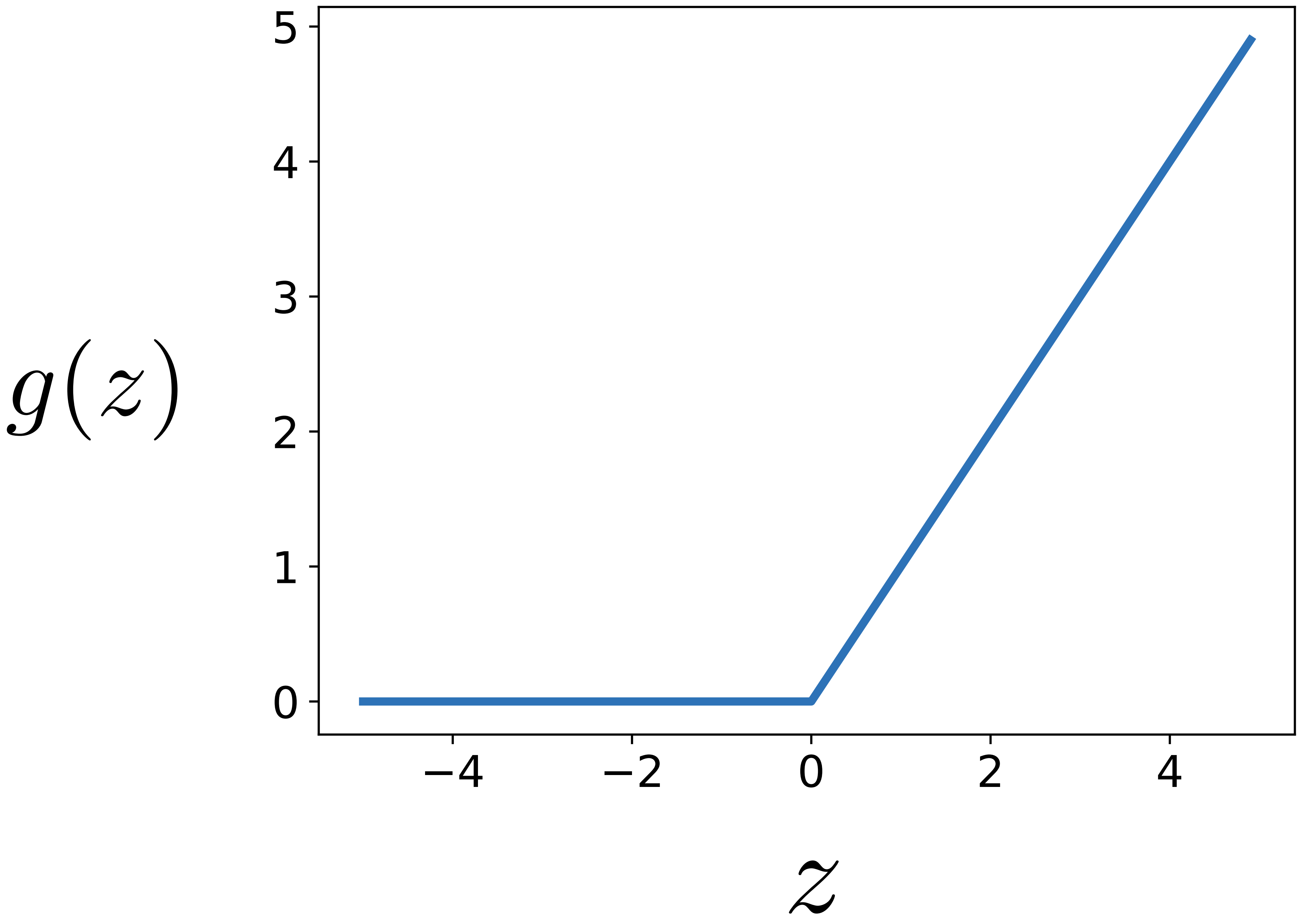


Computation in a neural net — nonlinearity

- 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)

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

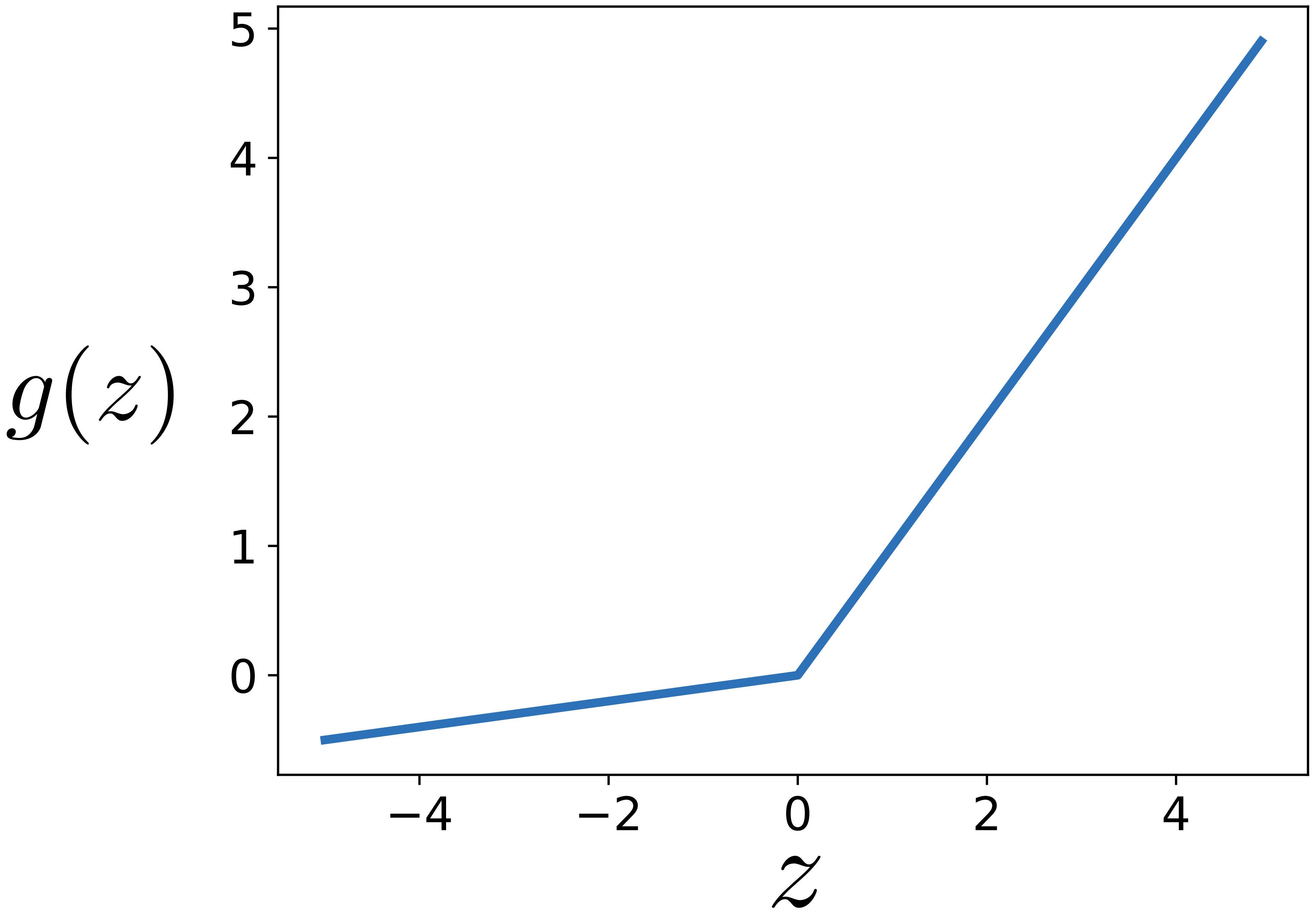


Computation in a neural net — nonlinearity

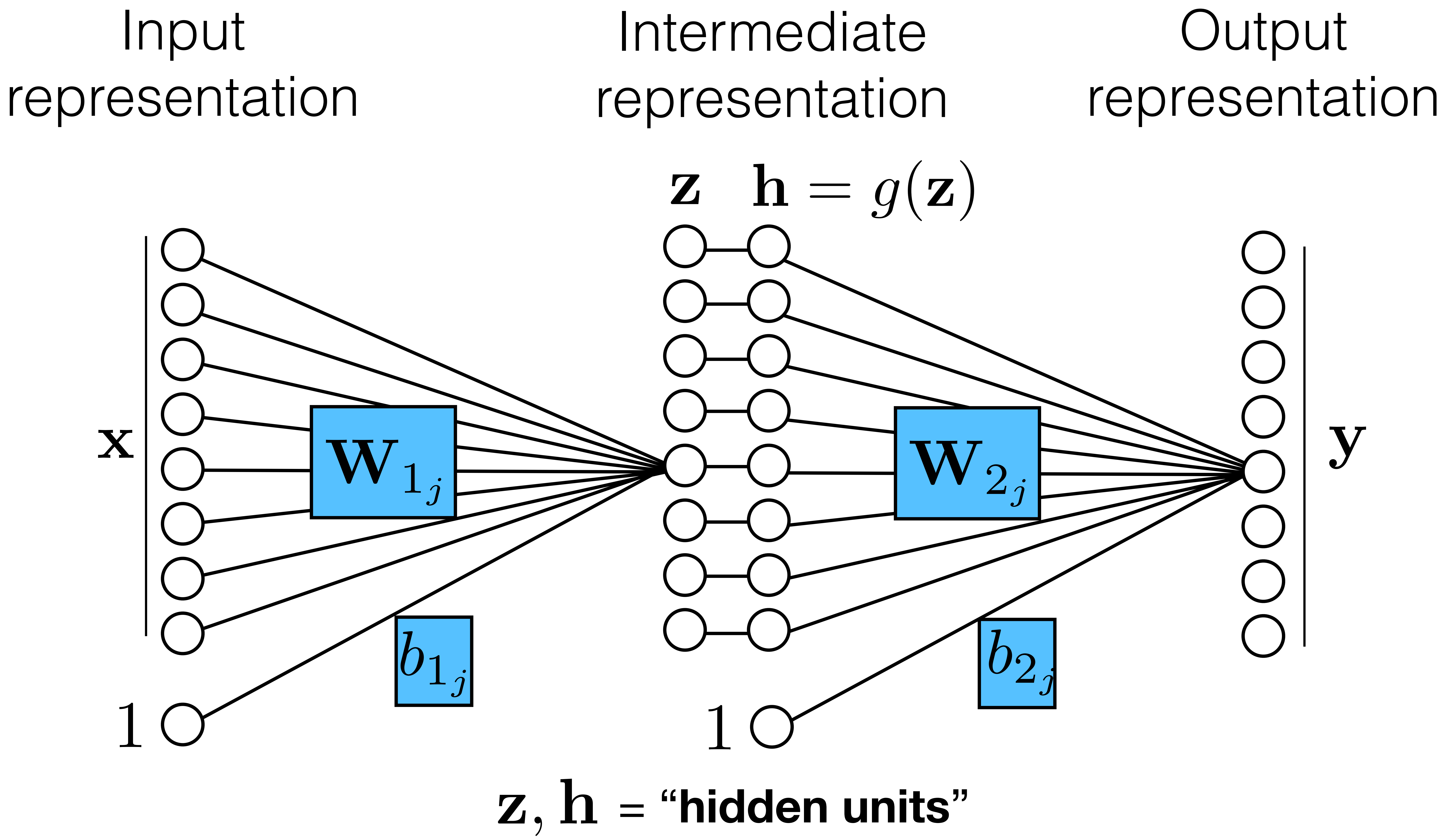
- where α is small (e.g. 0.02)
- Efficient to implement: $\frac{\partial g}{\partial z} = \begin{cases} -a, & \text{if} \quad z < 0 \\ 1, & \text{if} \quad z \geq 0 \end{cases}$
- Also known as probabilistic ReLU (PReLU)
- Has non-zero gradients everywhere (unlike ReLU)
- α can also be learned (see Kaiming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

Leaky ReLU

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

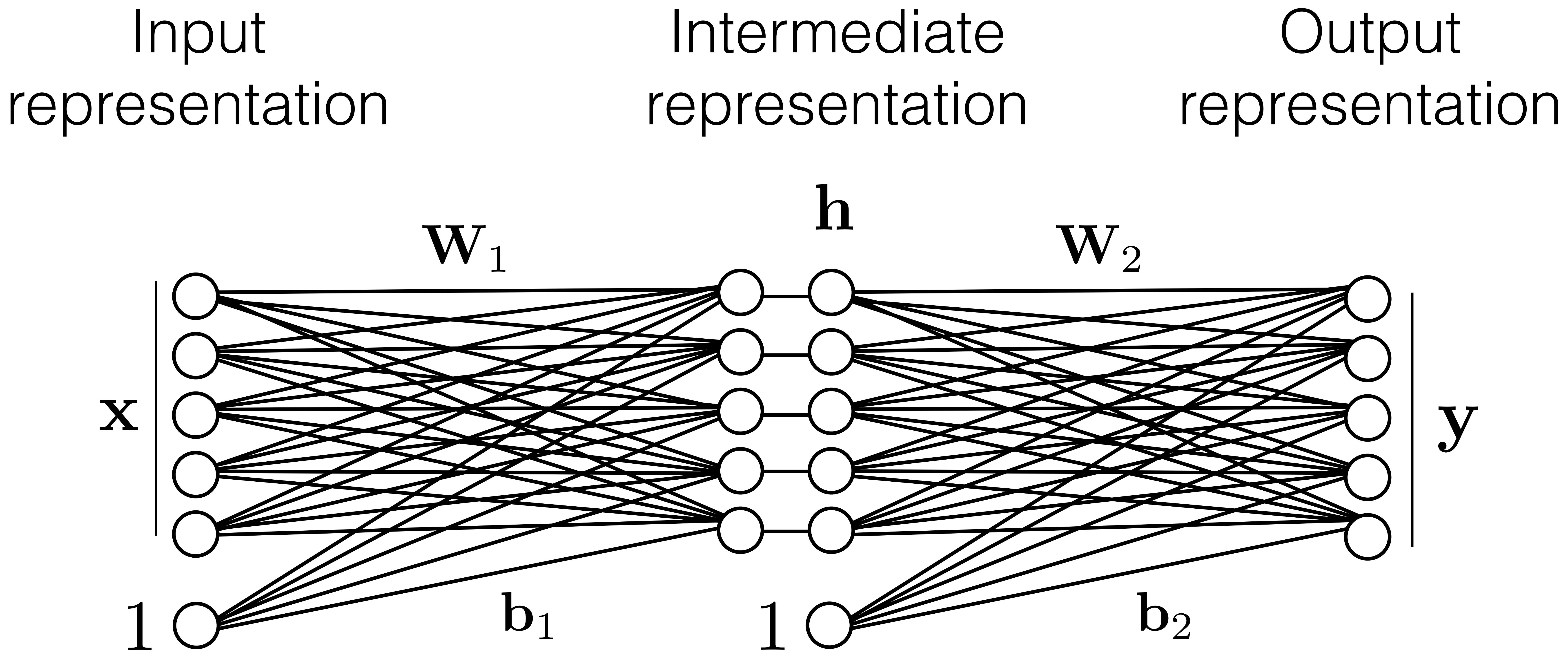


Stacking layers



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

Stacking layers



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

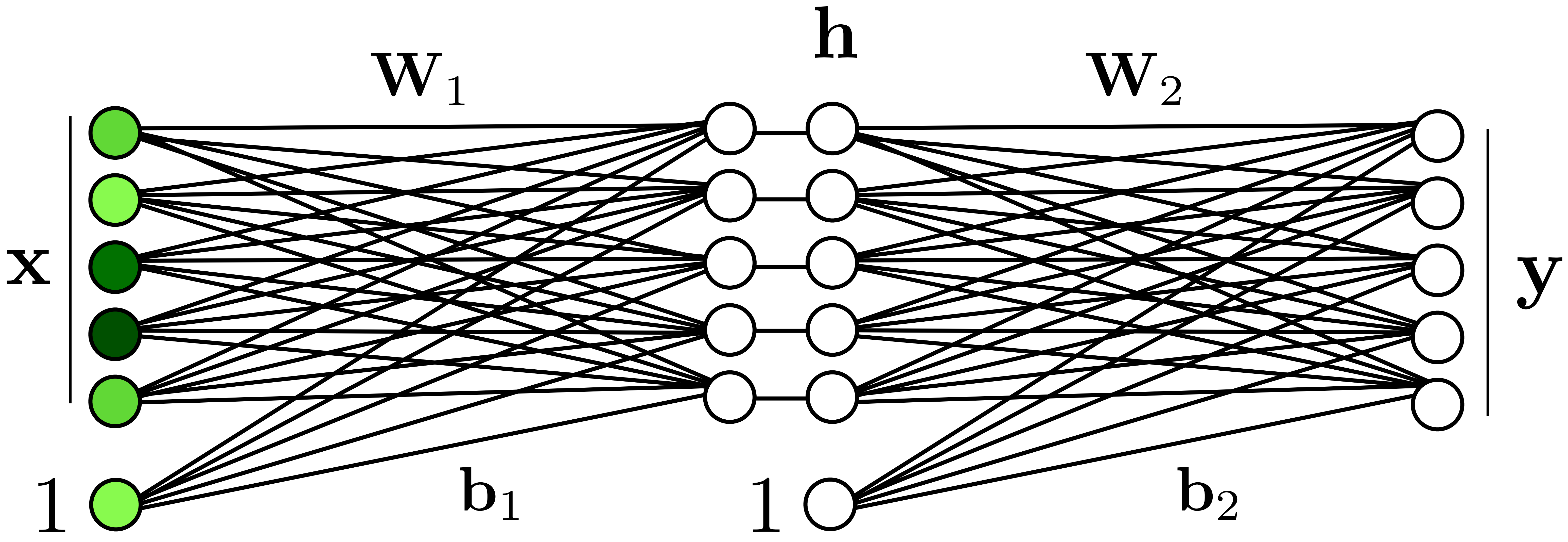
$$\theta = \{\mathbf{W}_1, \ldots, \mathbf{W}_L, \mathbf{b}_1, \ldots, \mathbf{b}_L\}$$

Stacking layers

Input representation

Intermediate representation

Output representation



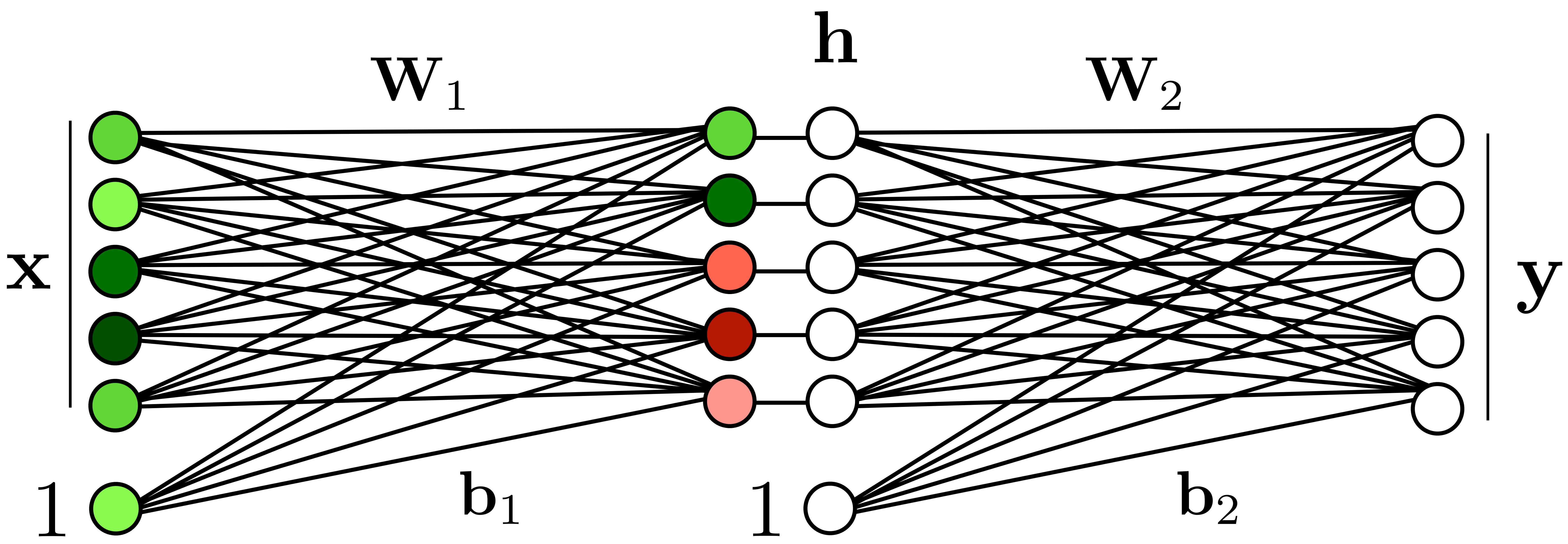
positive

negative

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

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

Stacking layers



positive

negative

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

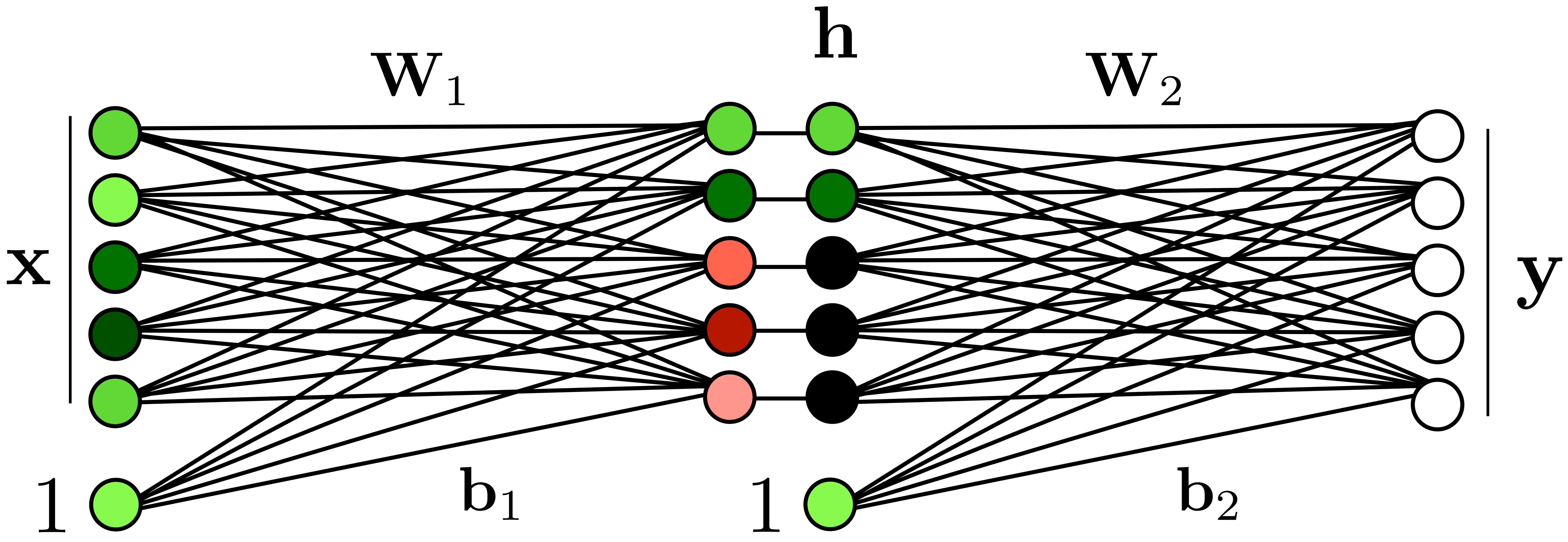
$$\theta = \{\mathbf{W}_1, \ldots, \mathbf{W}_L, \mathbf{b}_1, \ldots, \mathbf{b}_L\}$$

Stacking layers

Input representation

Intermediate representation

Output representation



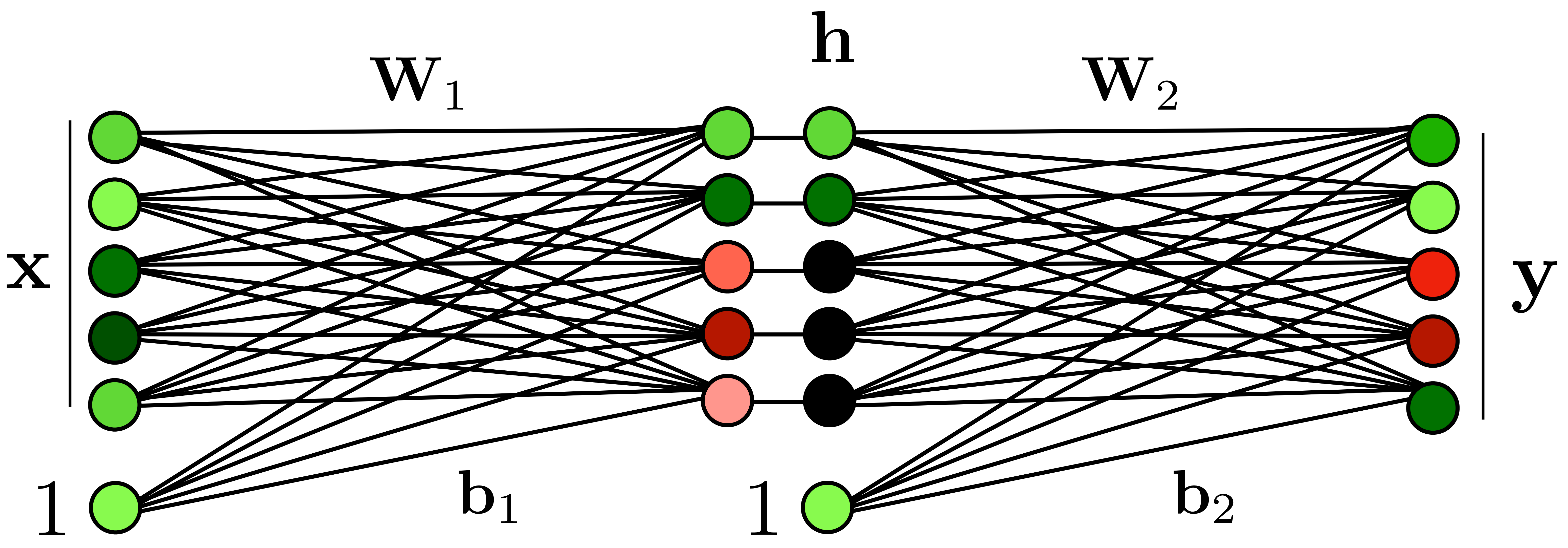
positive

negative

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

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

Stacking layers



Input representation

Intermediate representation

Output representation

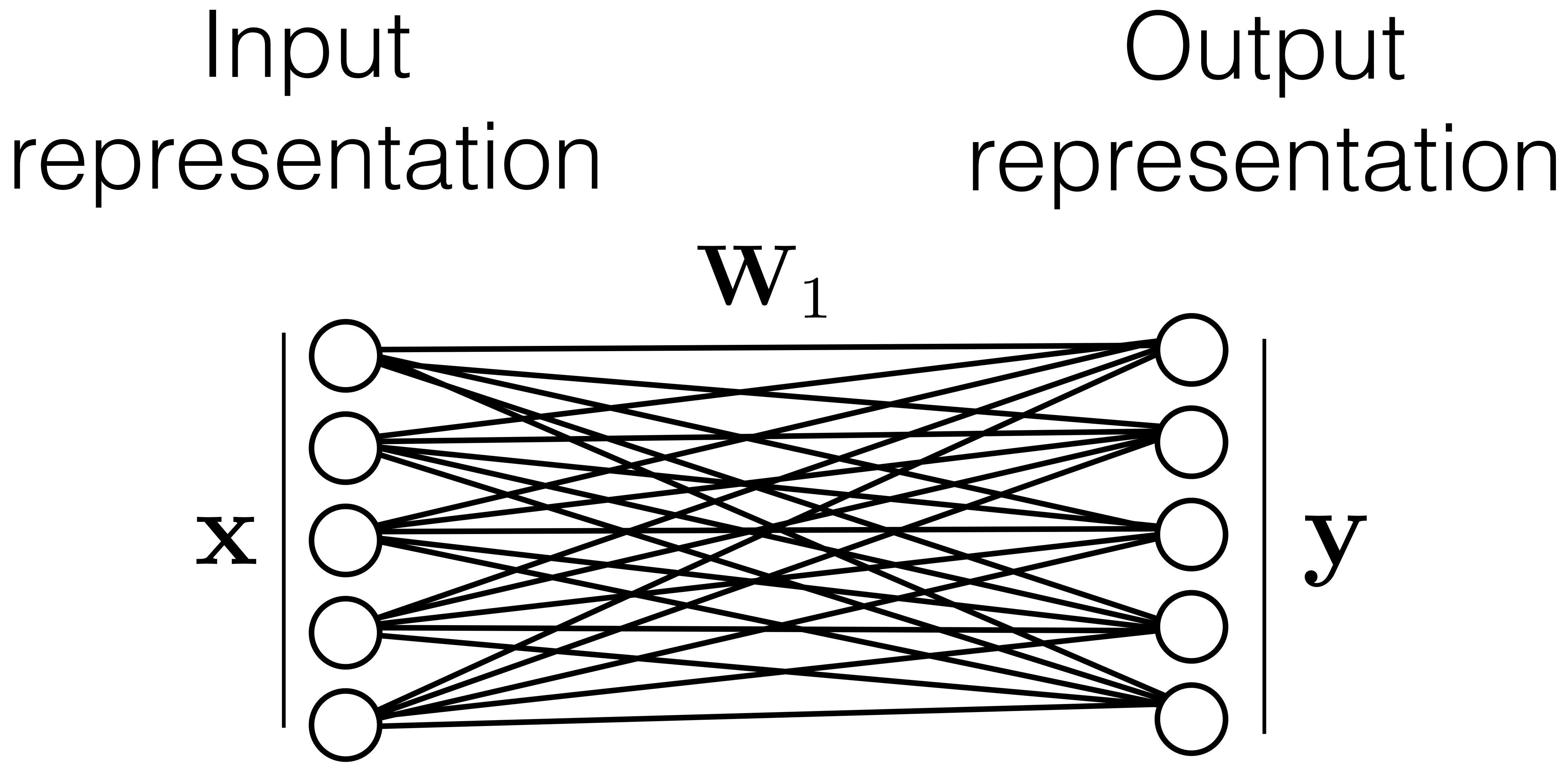
positive

negative

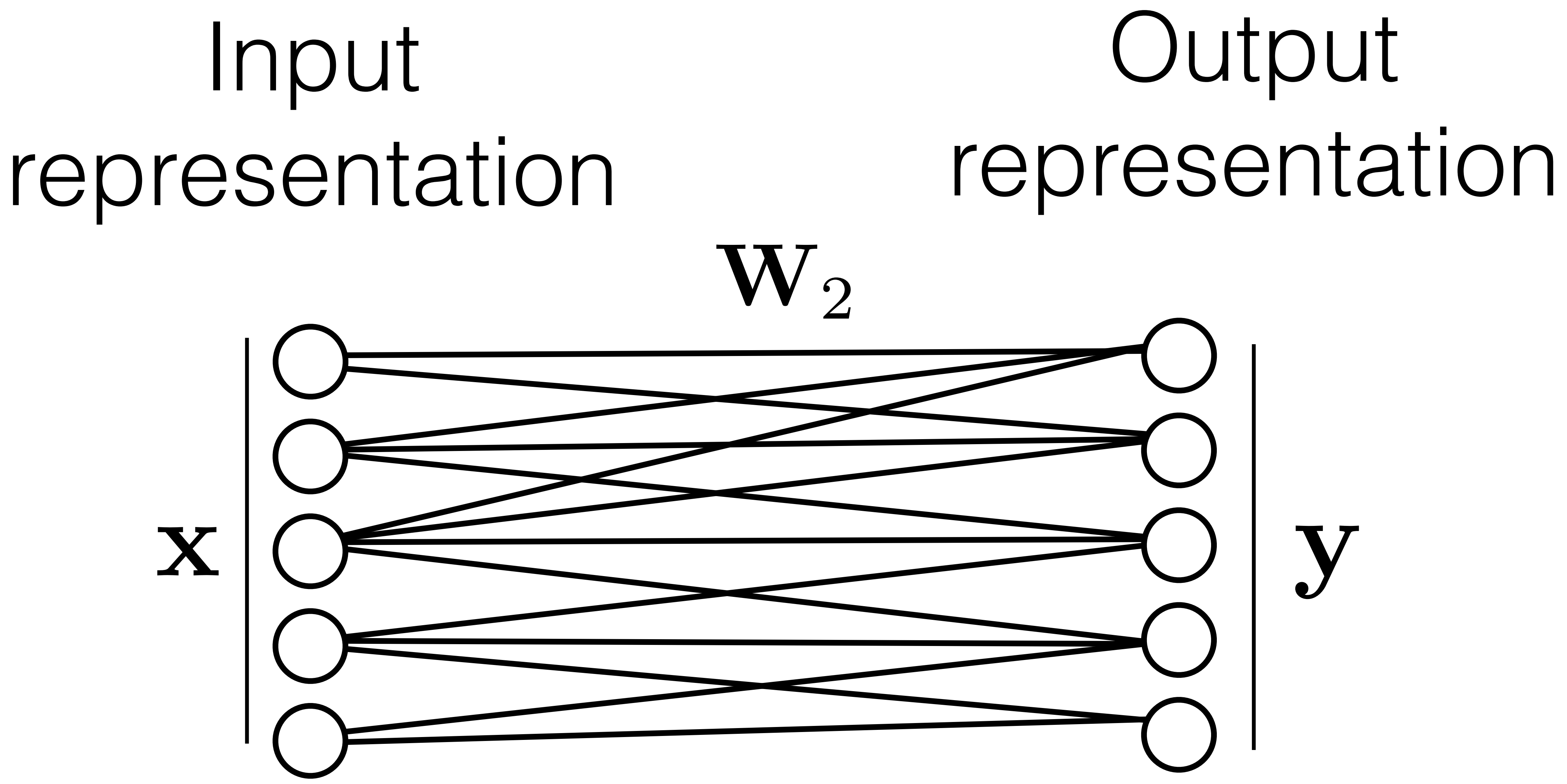
$$\mathbf{h} = g(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \qquad \mathbf{y} = g(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2)$$

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

Connectivity patterns

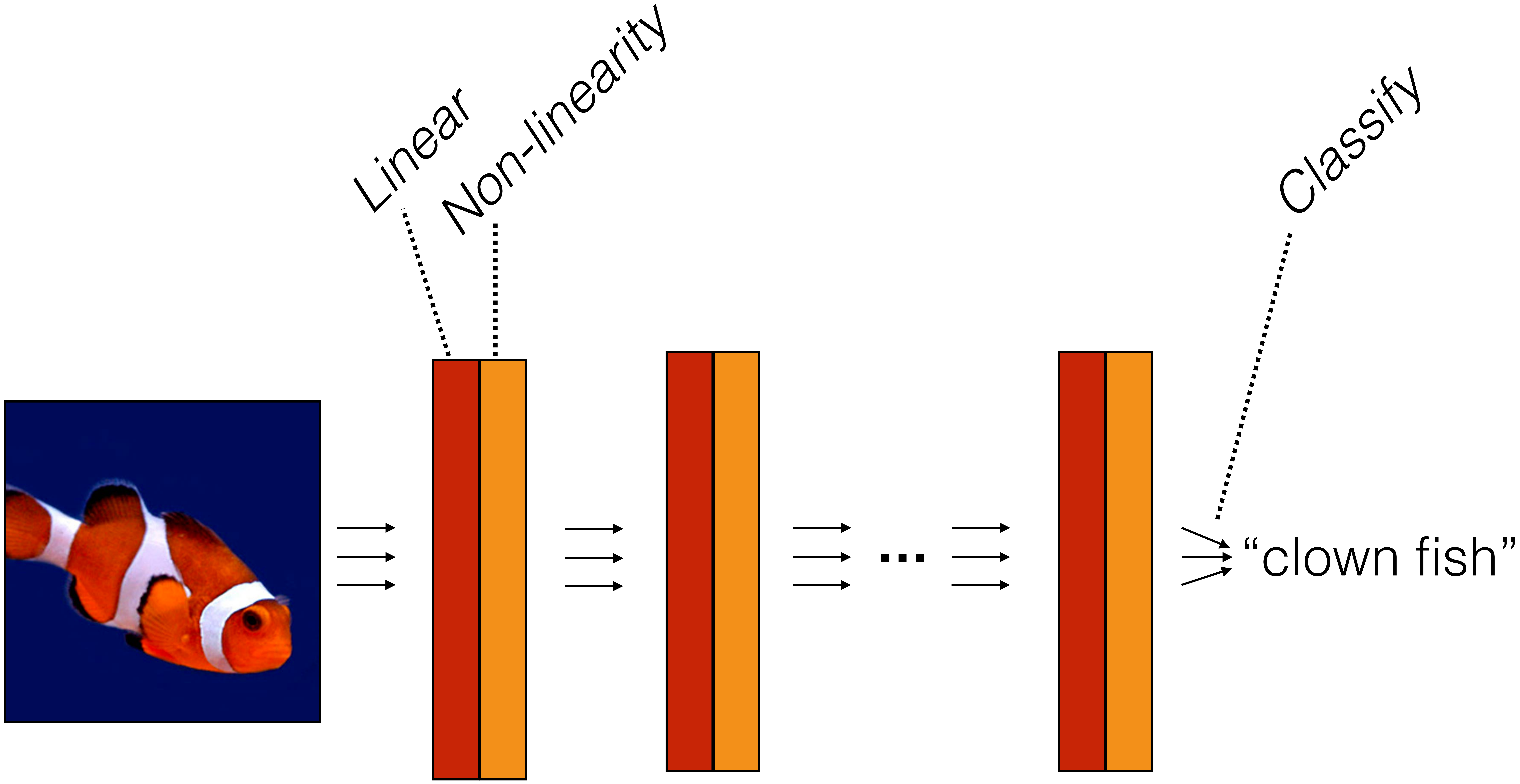


Fully connected layer



Locally connected layer
(Sparse W)

Deep nets



Linear

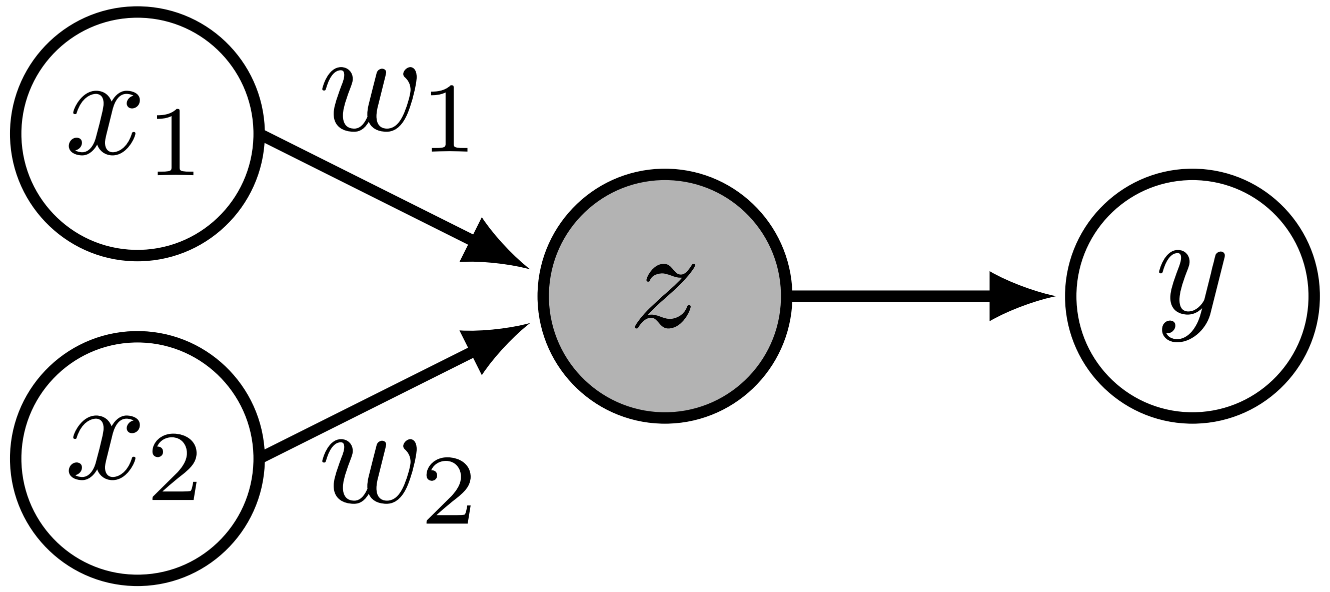
Non-linearity

Classify

"clown fish"

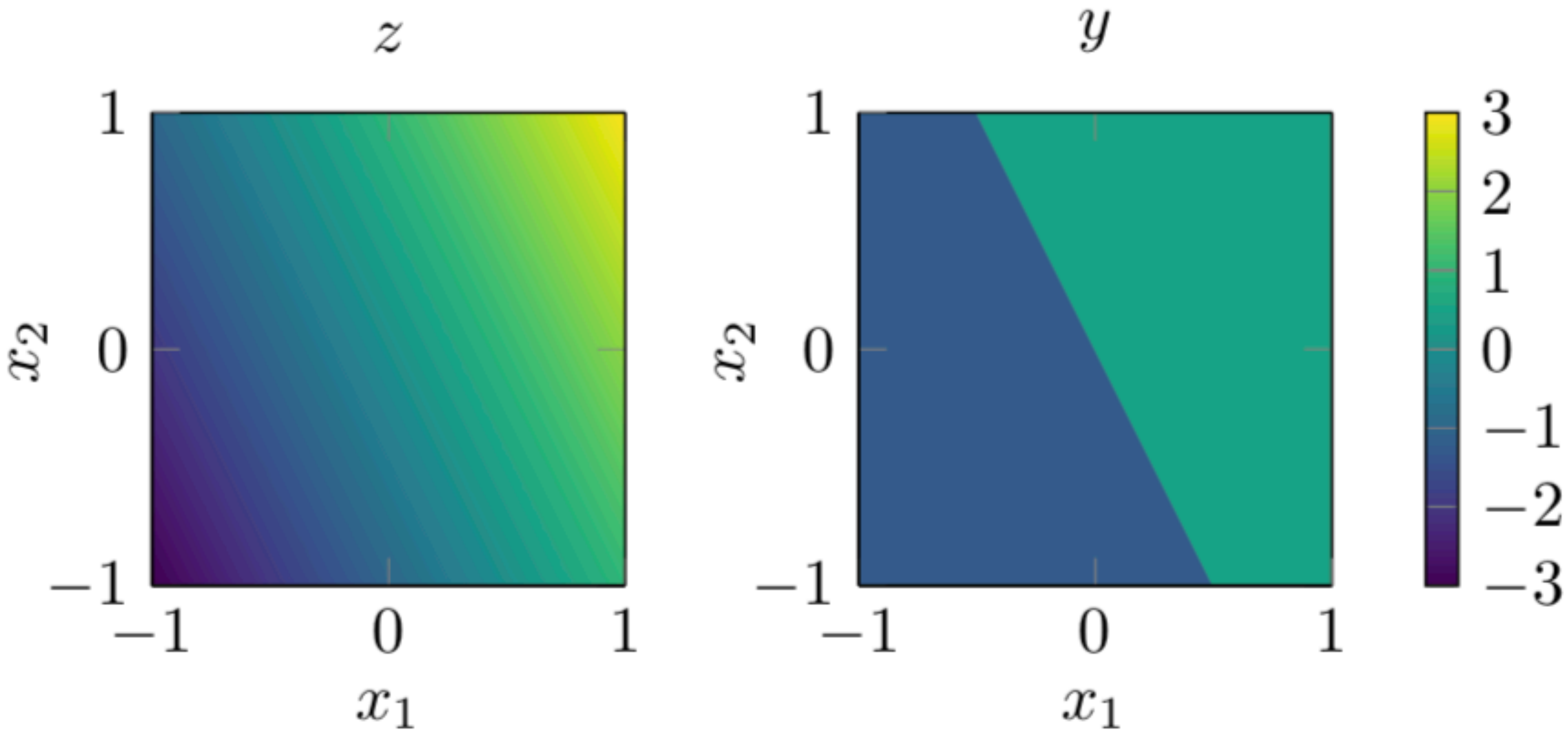
$$f(\mathbf{x}) = f_L(f_{L-1}(\dots f_2(f_1(\mathbf{x}))))$$

Example: linear classification with a perceptron



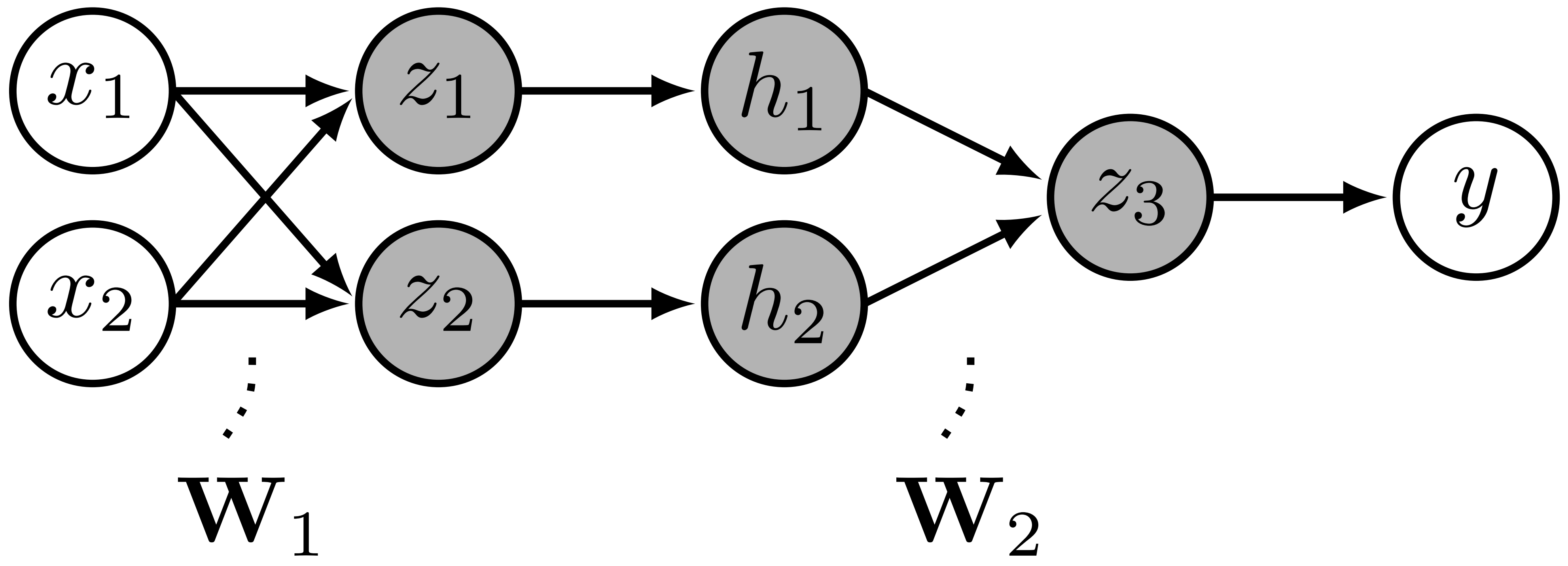
$$z = \mathbf{x}^T \mathbf{w} + b$$

$$y = g(z)$$



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

Example: nonlinear classification with a deep net net

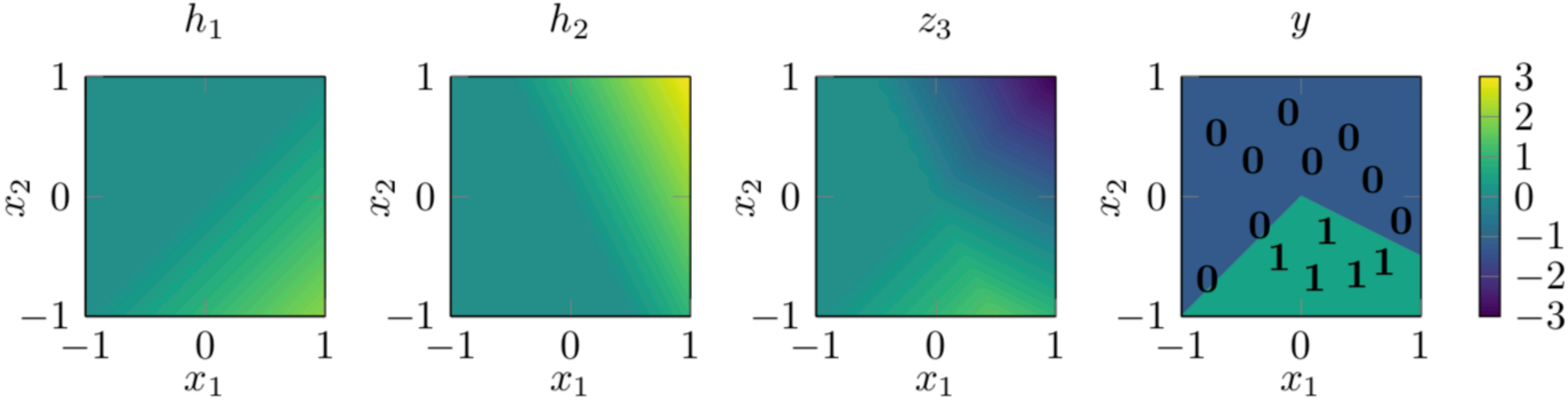


$$\mathbf{z} = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1$$

$$\mathbf{h} = g(\mathbf{z})$$

$$z_3 = \mathbf{W}_2\mathbf{h} + b_2$$

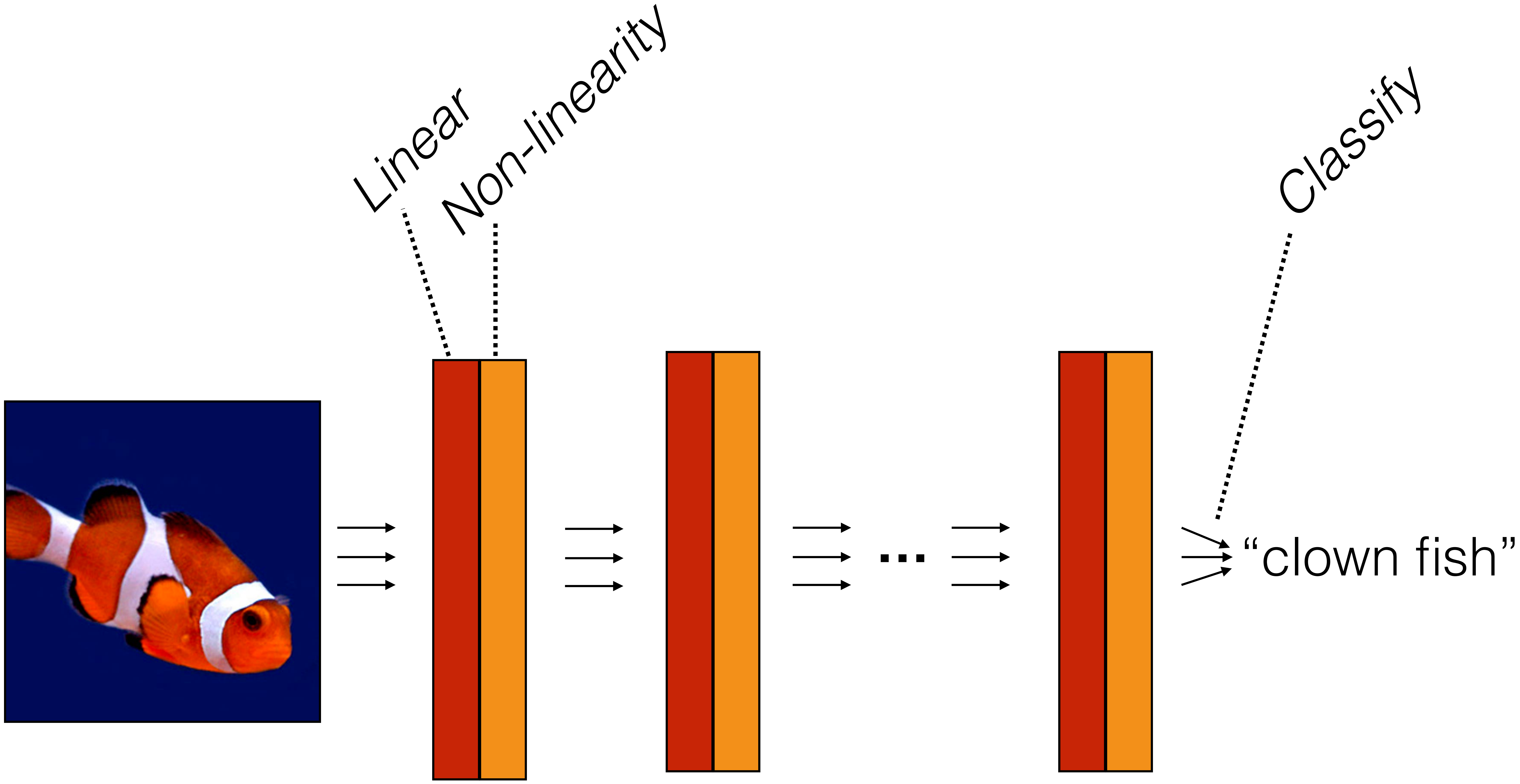
$$y = 1(z_3 > 0)$$



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

Last layer

dolphin

cat

grizzly bear

angel fish

chameleon

clown fish

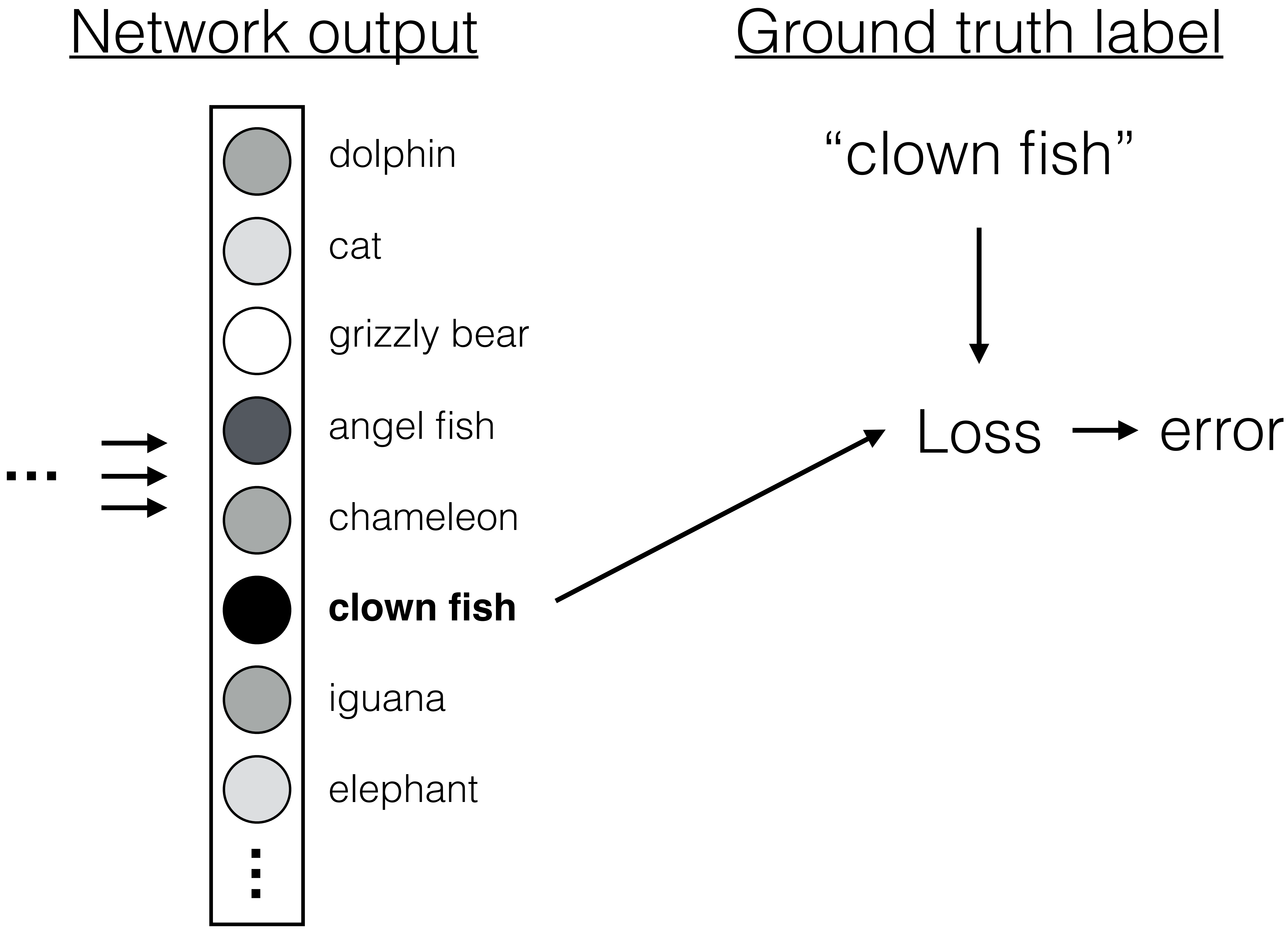
iguana

elephant

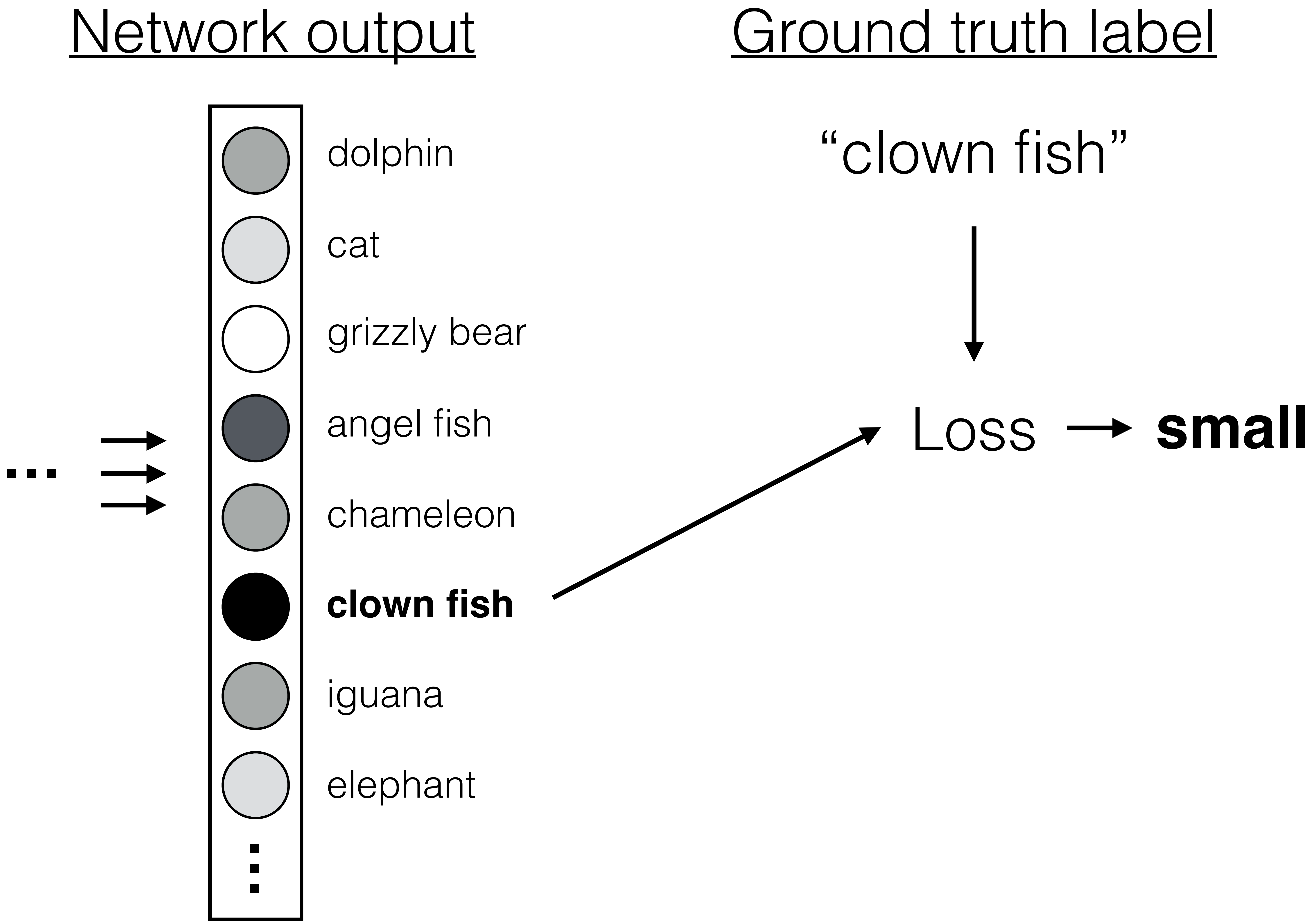
argmax

“clown fish”

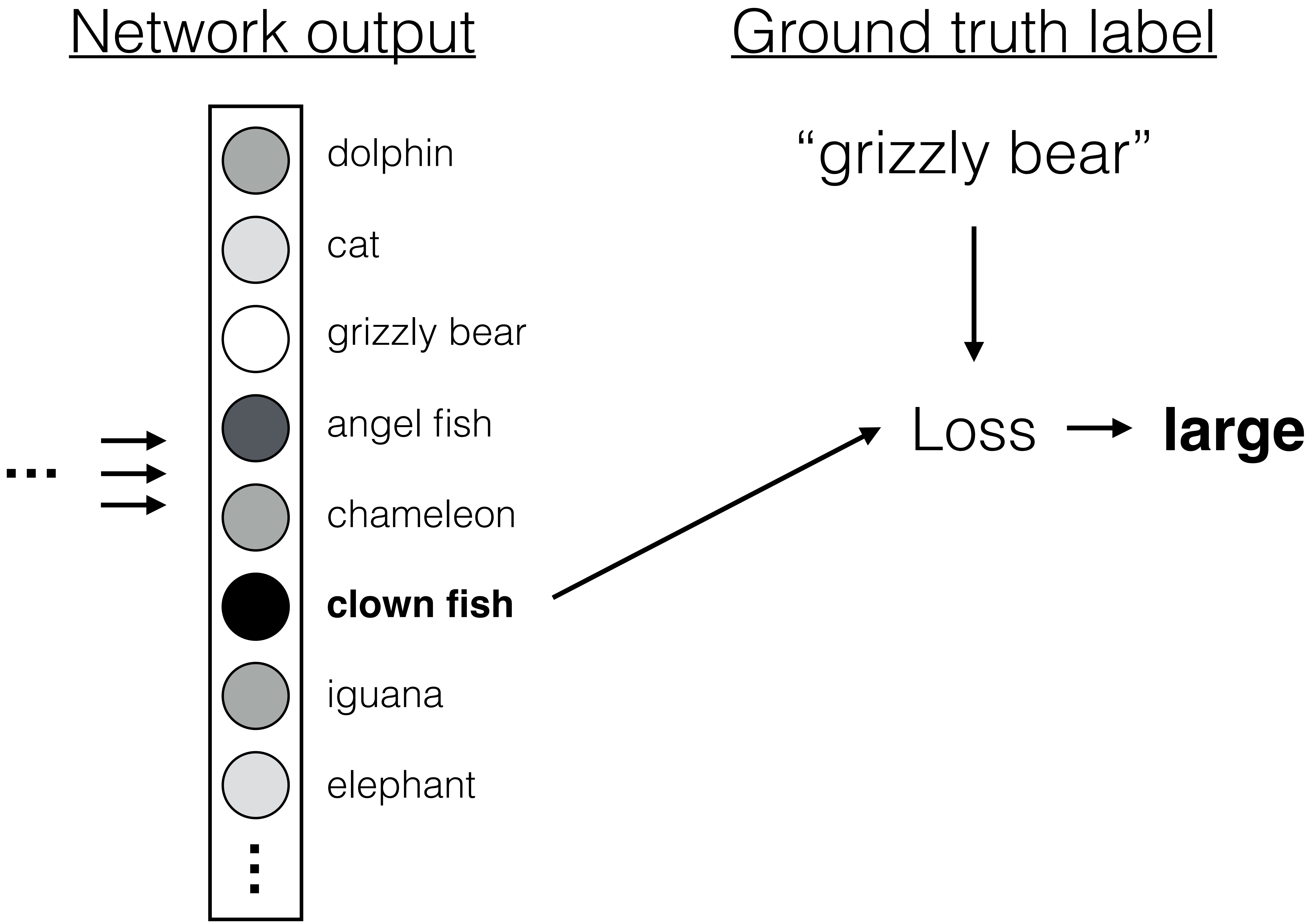
Loss function

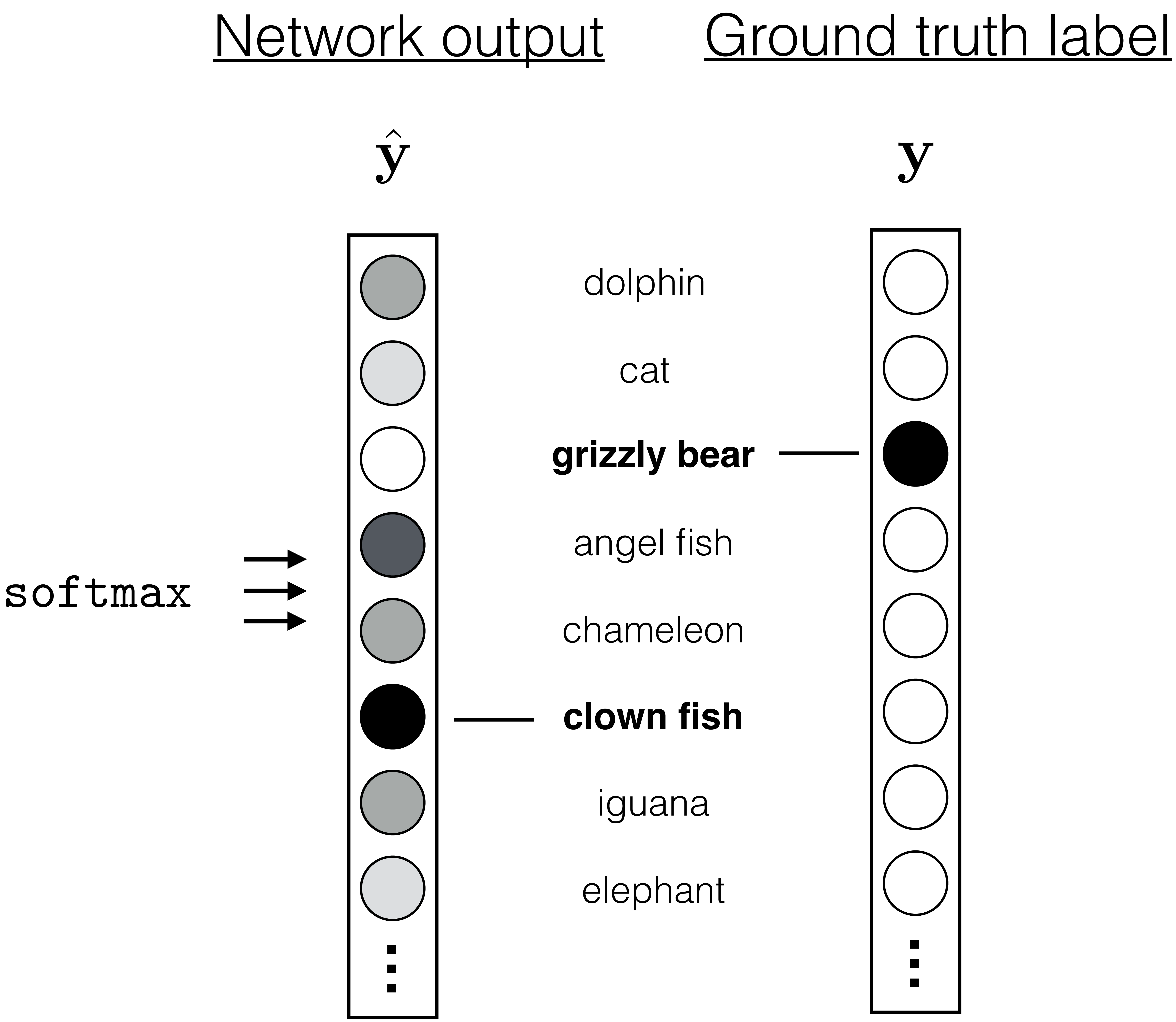


Loss function



Loss function

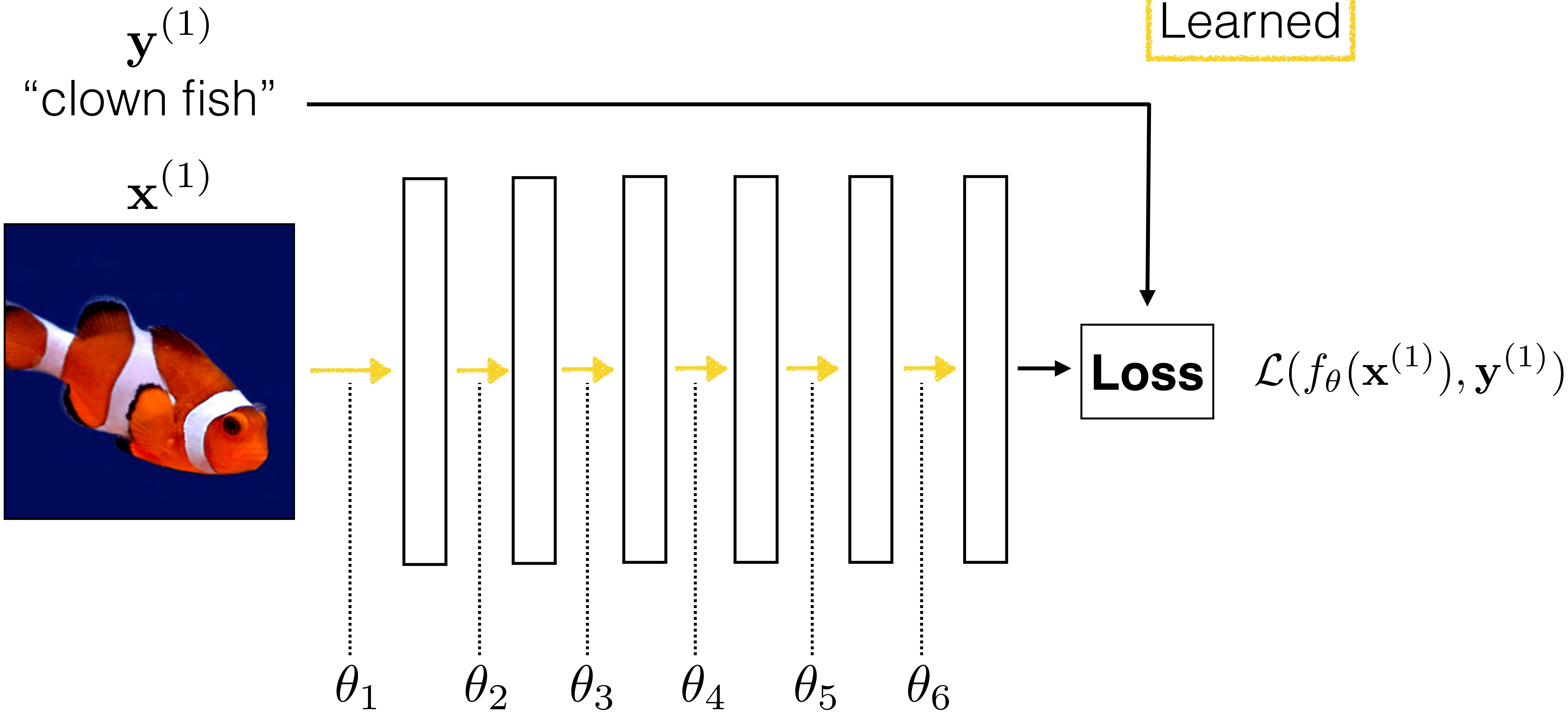




Probability of the observed data under the model

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

Deep learning



$\mathbf{y}^{(1)}$

“clown fish”

$\mathbf{x}^{(1)}$

Loss

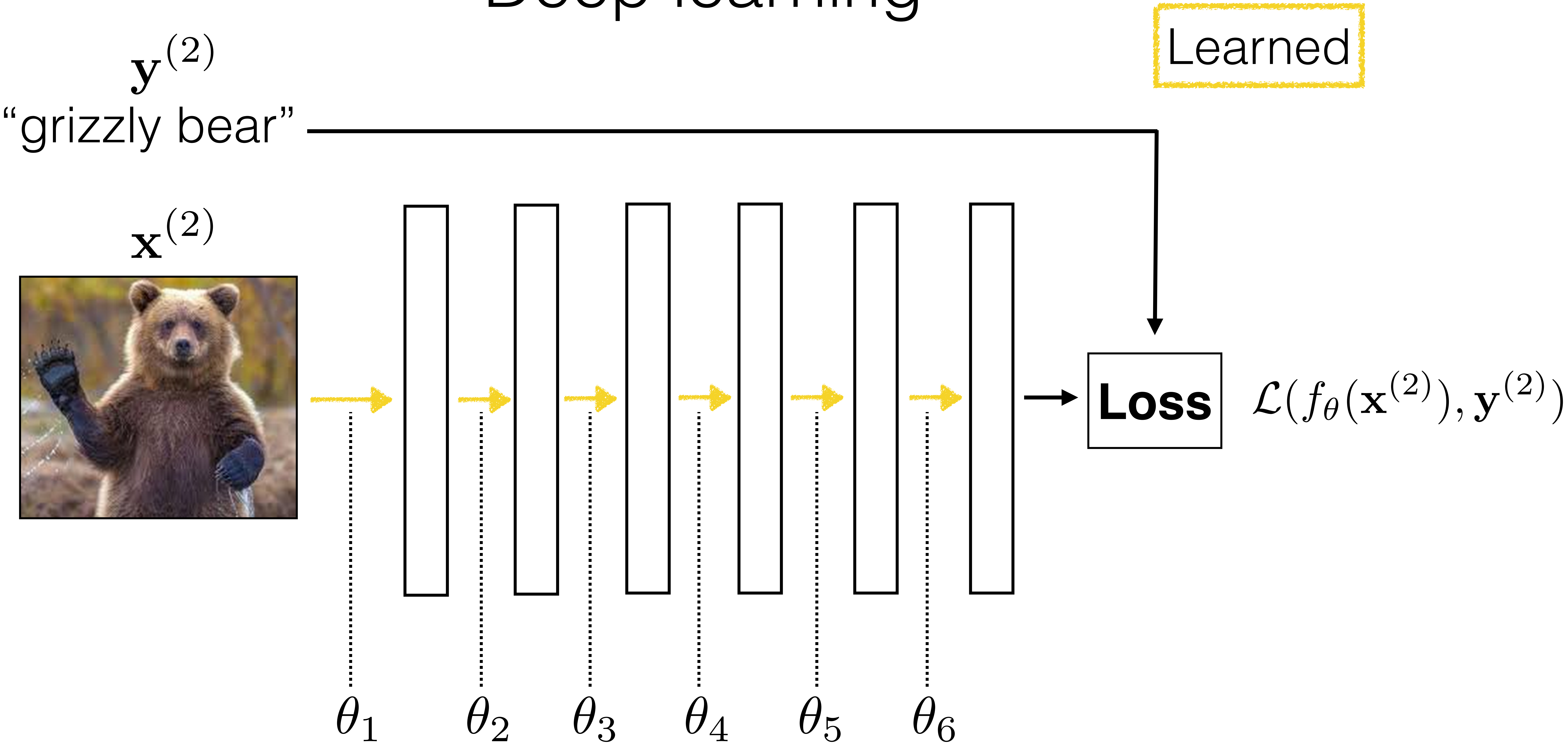
$\mathcal{L}(f_\theta(\mathbf{x}^{(1)}), \mathbf{y}^{(1)})$

θ_1 θ_2 θ_3 θ_4 θ_5 θ_6

Learned

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

Deep learning



Learned

$\mathbf{y}^{(2)}$

"grizzly bear"

$\mathbf{x}^{(2)}$

Loss

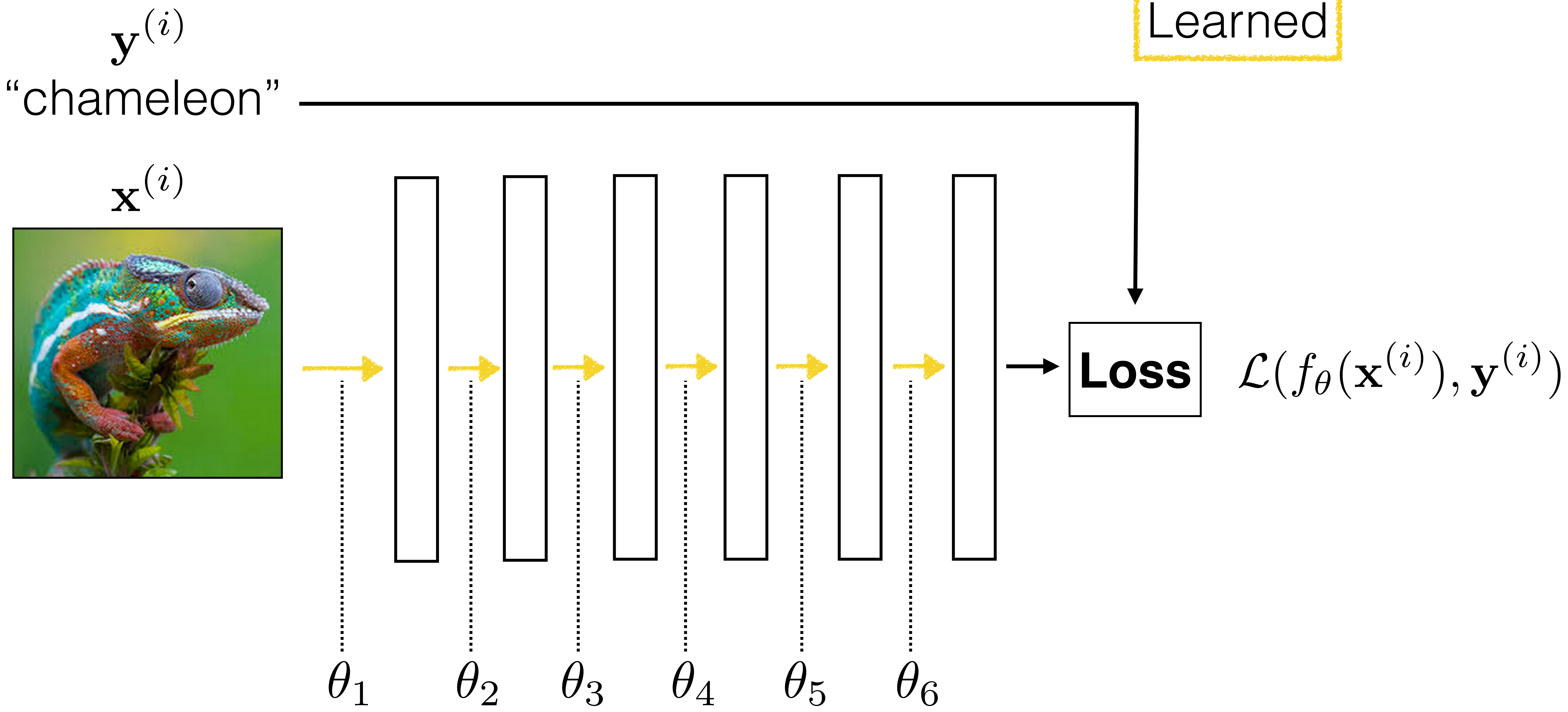
$\mathcal{L}(f_\theta(\mathbf{x}^{(2)}), \mathbf{y}^{(2)})$

θ_1 θ_2 θ_3 θ_4 θ_5 θ_6

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

Deep learning

Learned



$\mathbf{y}^{(i)}$

"chameleon"

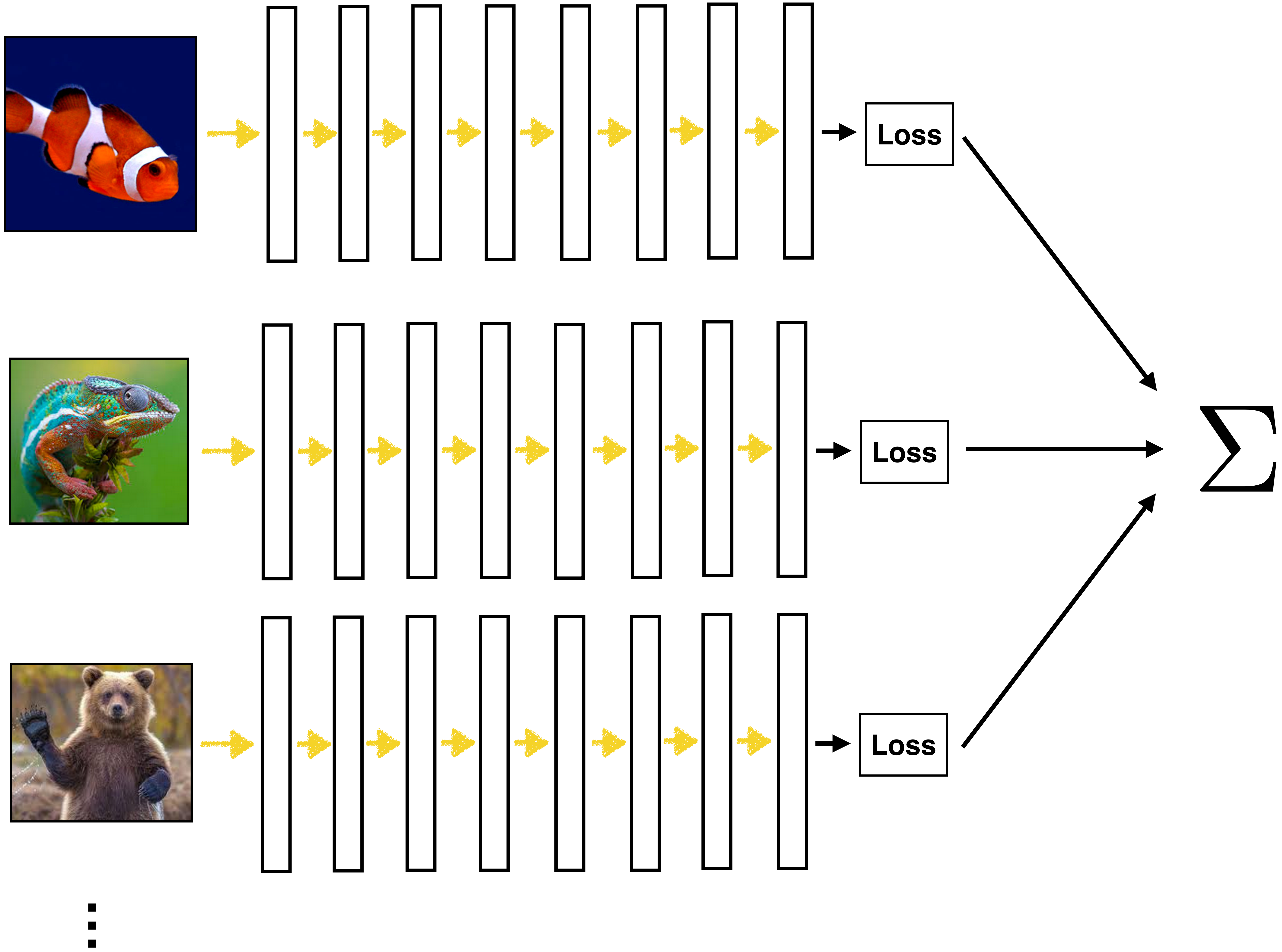
$\mathbf{x}^{(i)}$

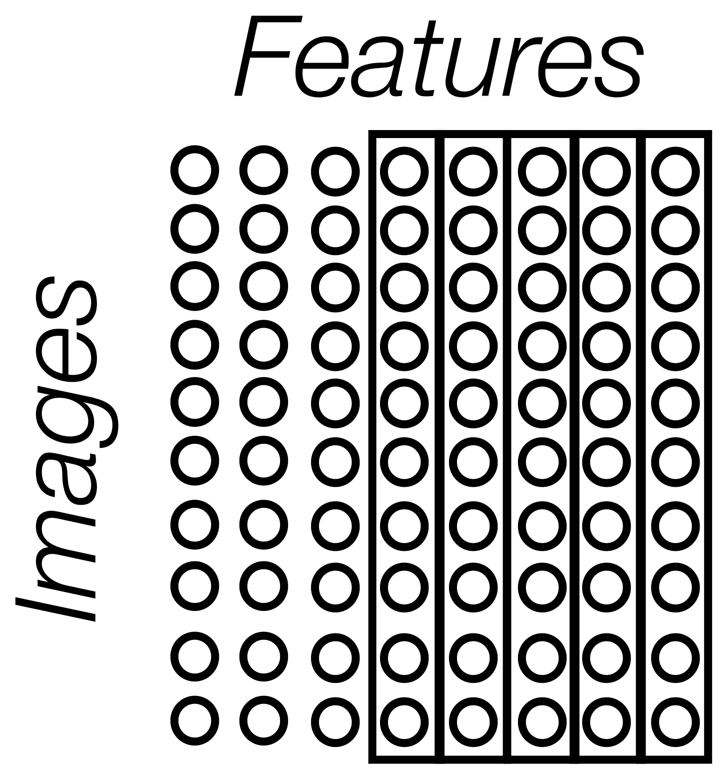
θ_1 θ_2 θ_3 θ_4 θ_5 θ_6

Loss $\mathcal{L}(f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$

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

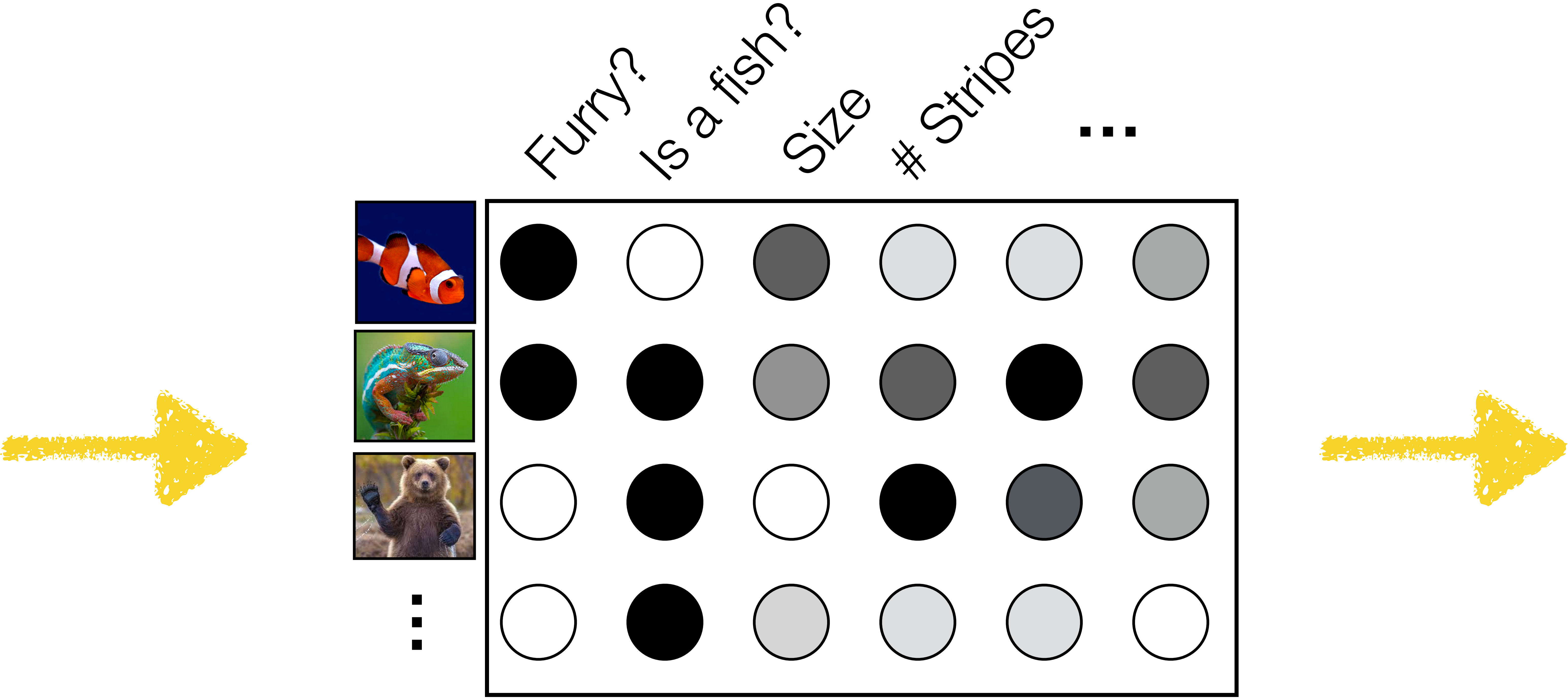
Batch (parallel) processing





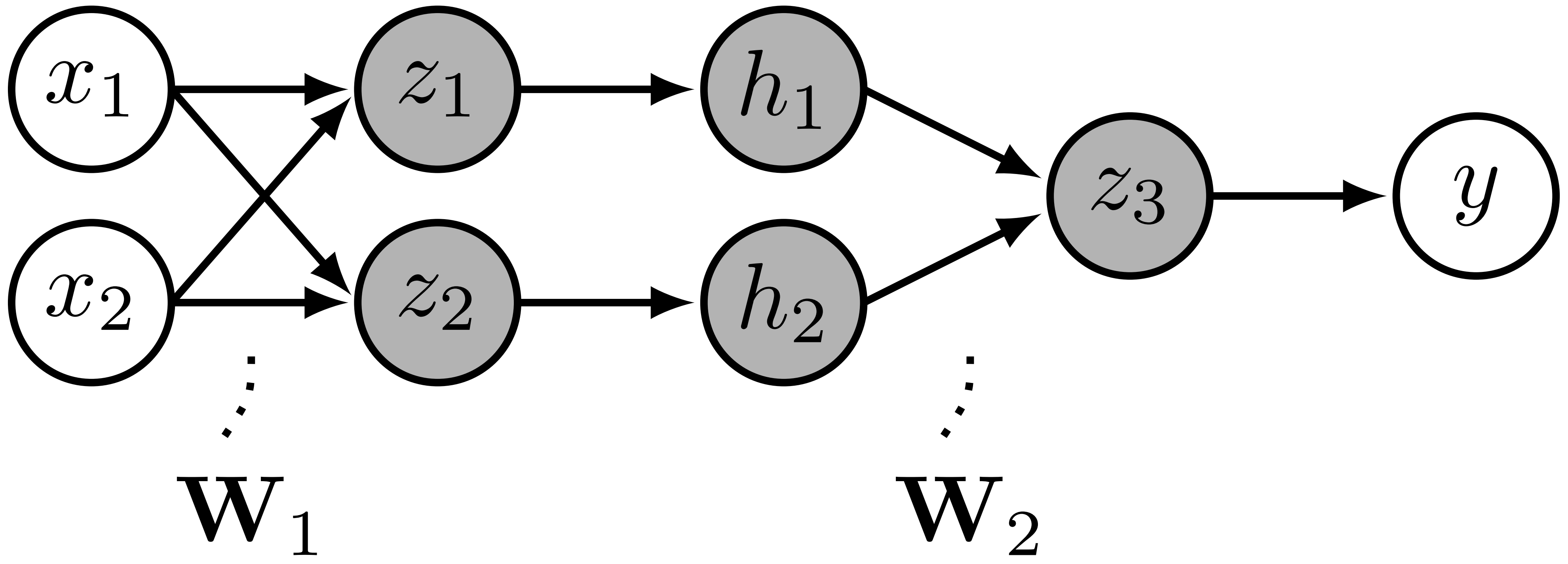
Tensors

(multi-dimensional arrays)



Each layer is a representation of the data

Everything is a tensor



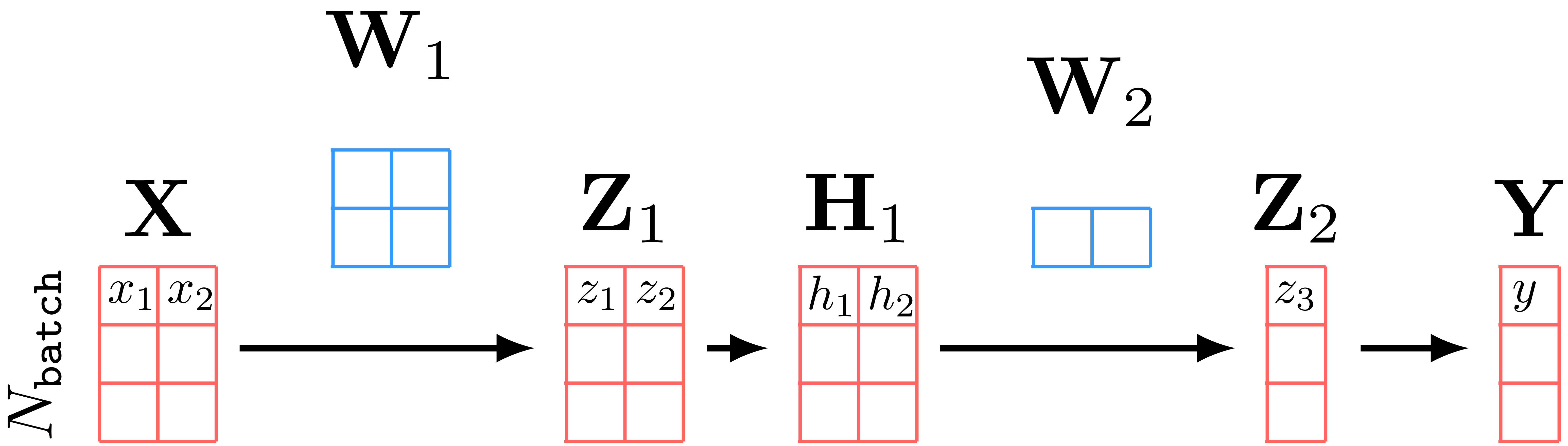
$$\mathbf{z} = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1$$

$$\mathbf{h} = g(\mathbf{z})$$

$$z_3 = \mathbf{W}_2\mathbf{h} + b_2$$

$$y = 1(z_3 > 0)$$

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**

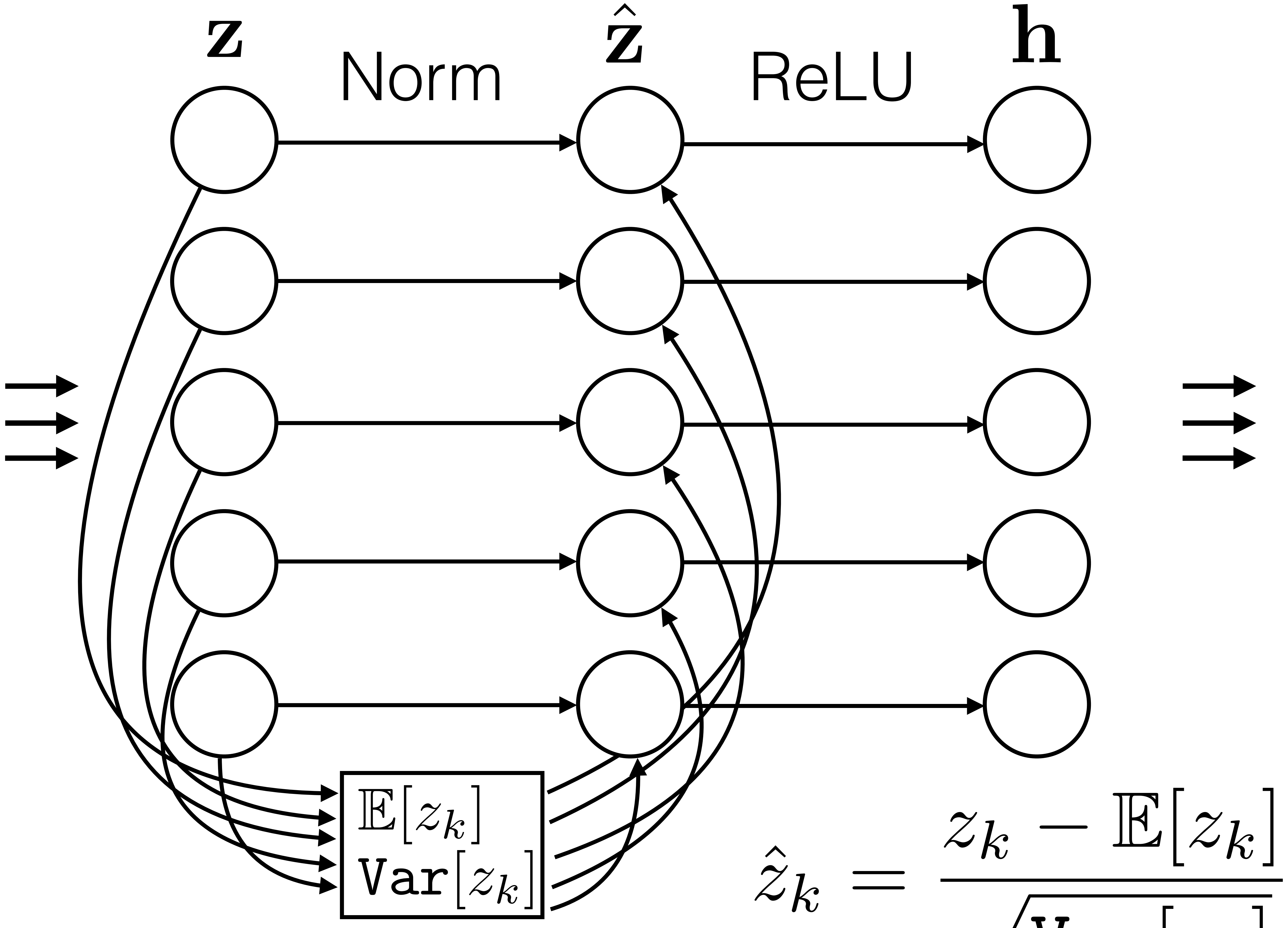
Regularizing the weights in a neural net

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

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

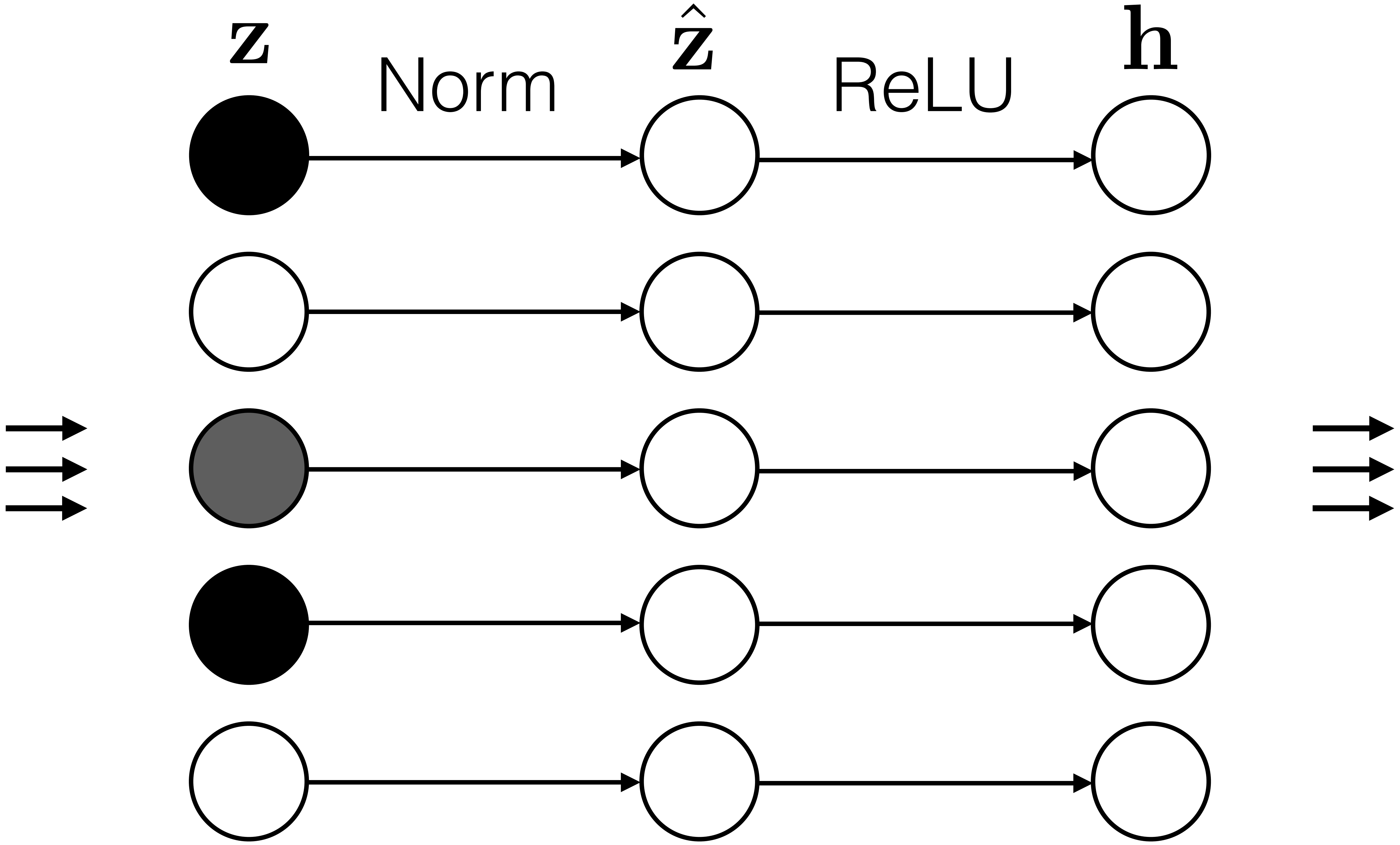
“We prefer to keep weights small.”

Normalization layers



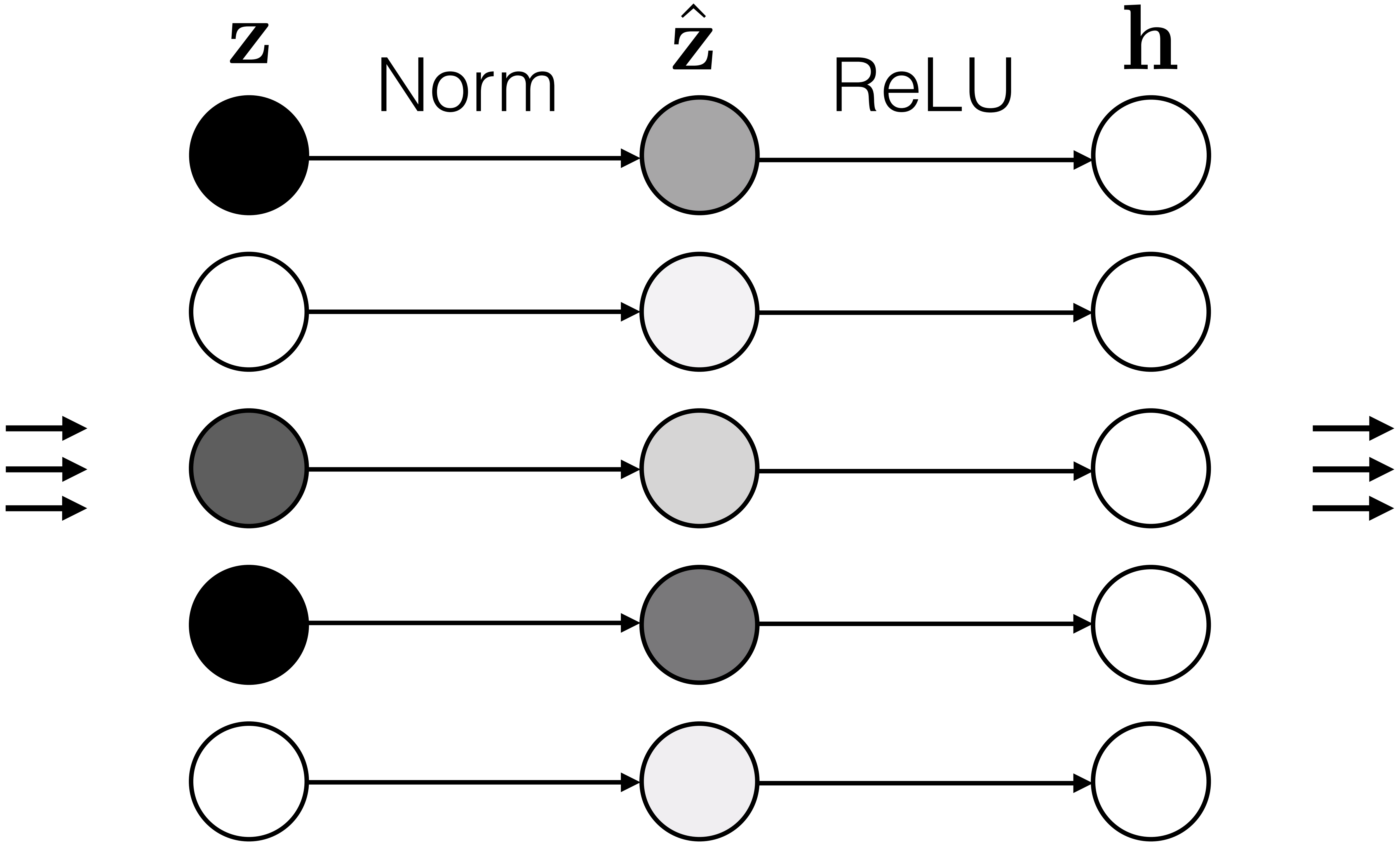
$$\hat{z}_k = \frac{z_k - \mathbb{E}[z_k]}{\sqrt{\texttt{Var}[z_k]}}$$

Normalization layers



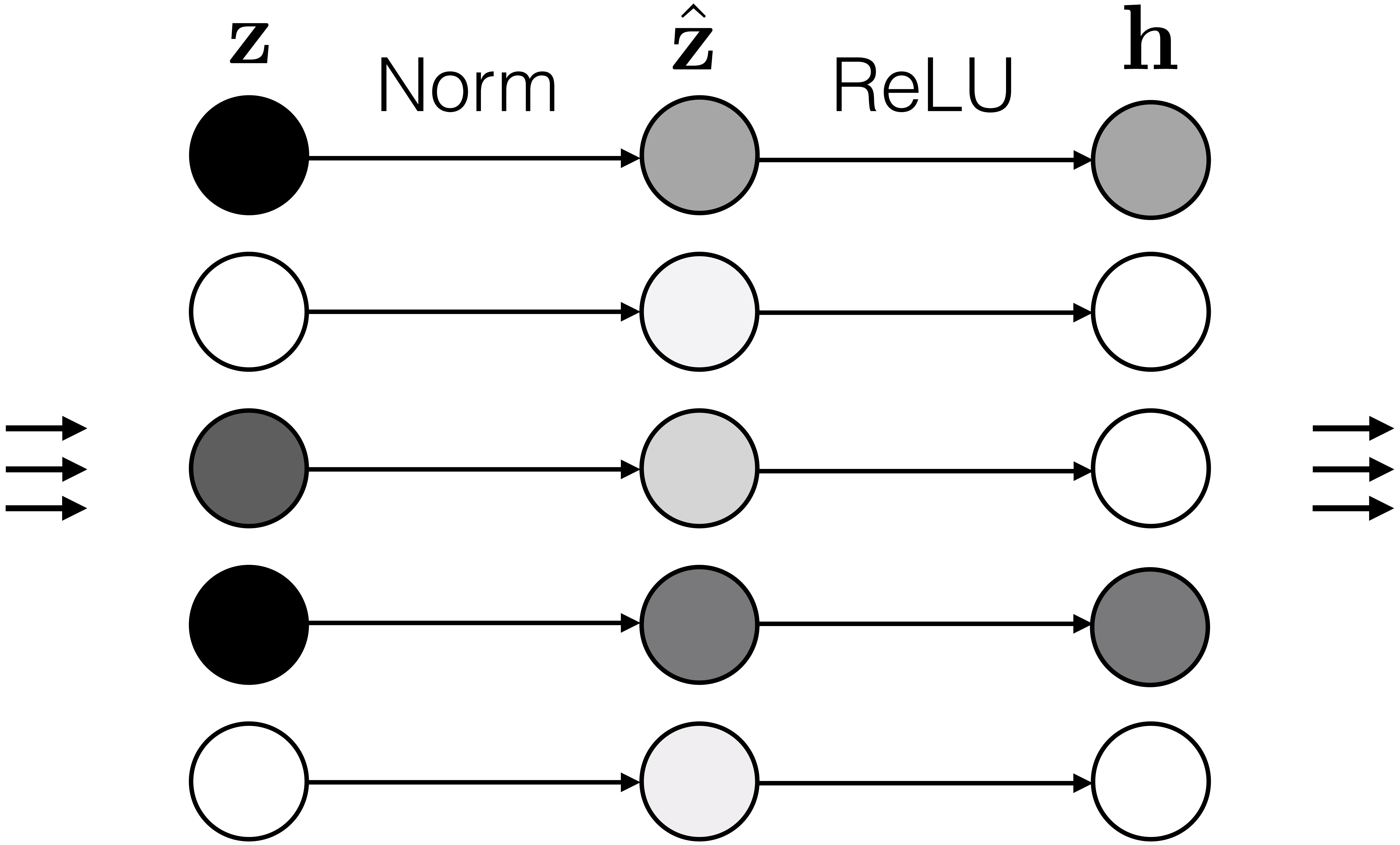
$$\hat{z}_k = \frac{z_k - \mathbb{E}[z_k]}{\sqrt{\texttt{Var}[z_k]}}$$

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.

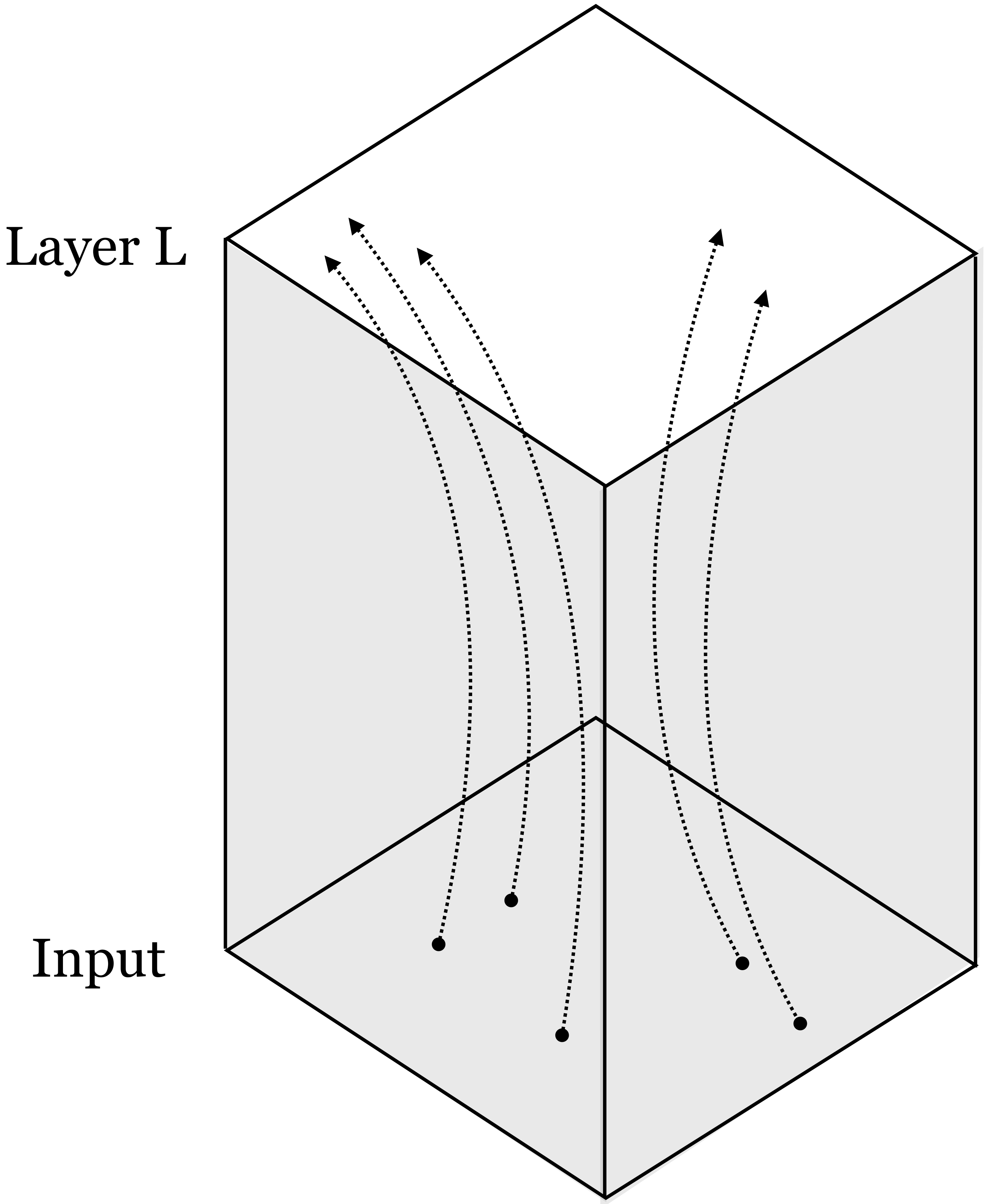
Squashes units into a **standard range**, avoiding overflow.

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

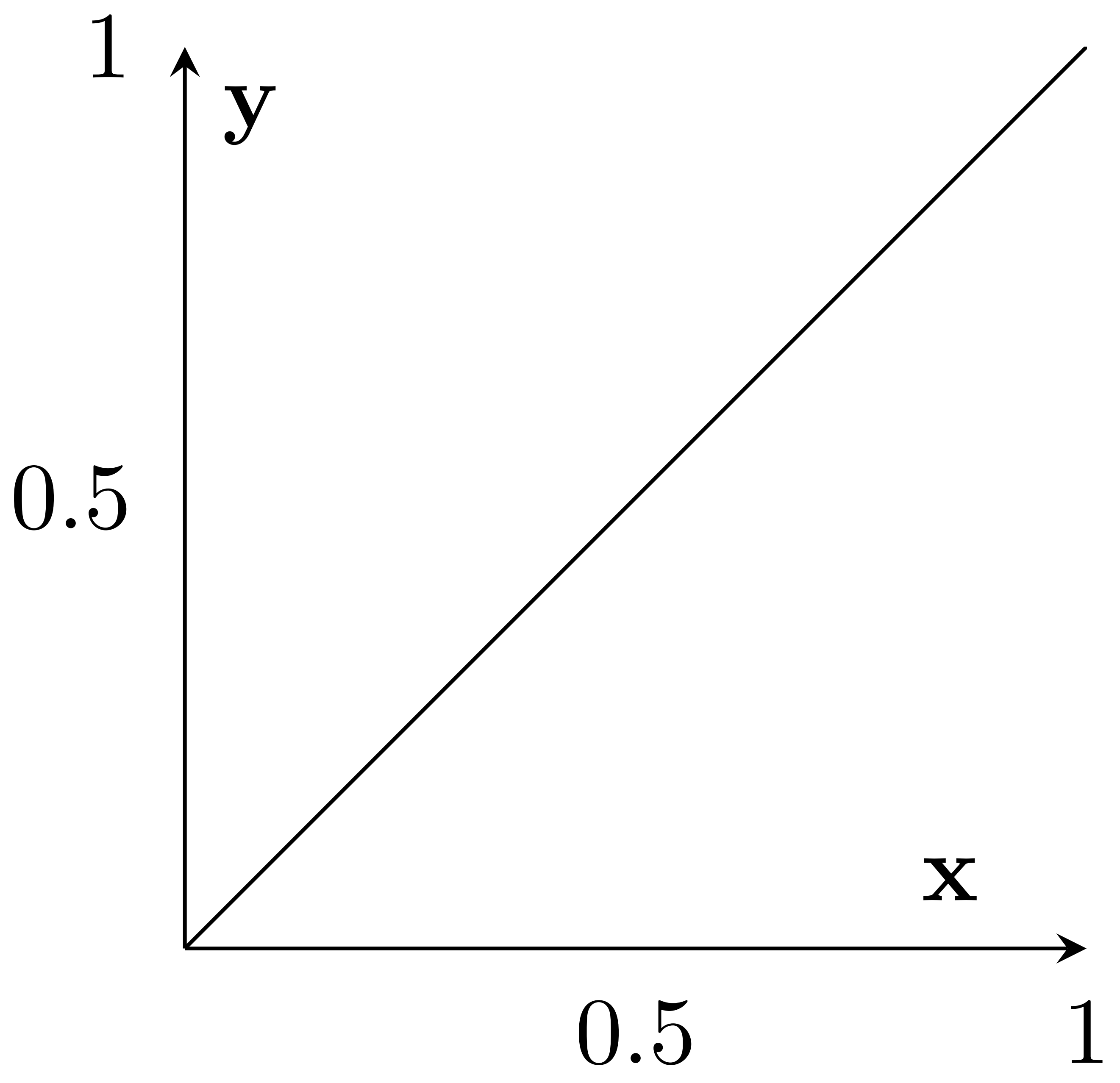
Both these properties reduce the effective capacity of the model, i.e. regularize the model.

Deep nets are data transformers

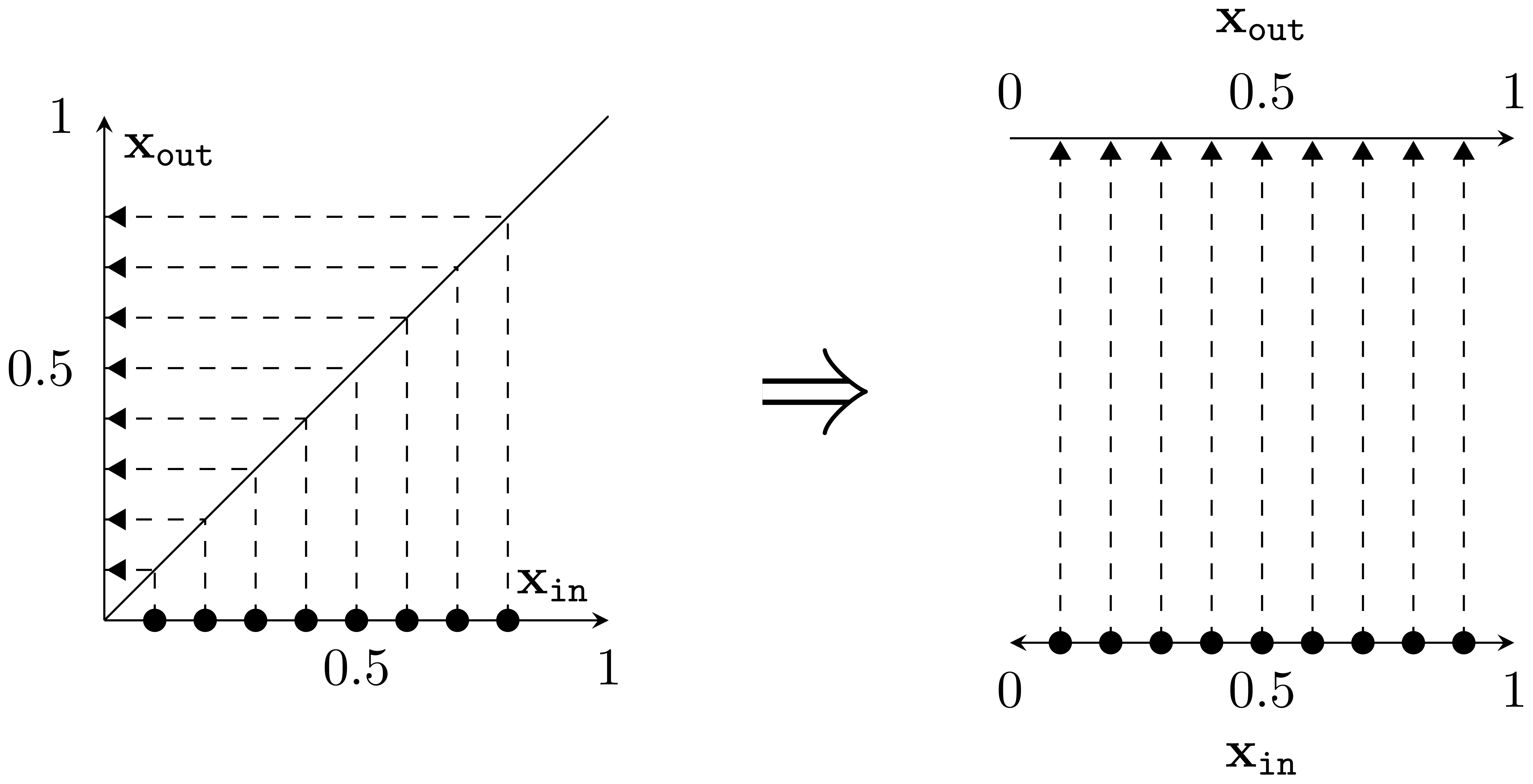
- Deep nets transform datapoints, layer by layer
- Each layer is a different *representation* of the data
- We call these representations **embeddings**



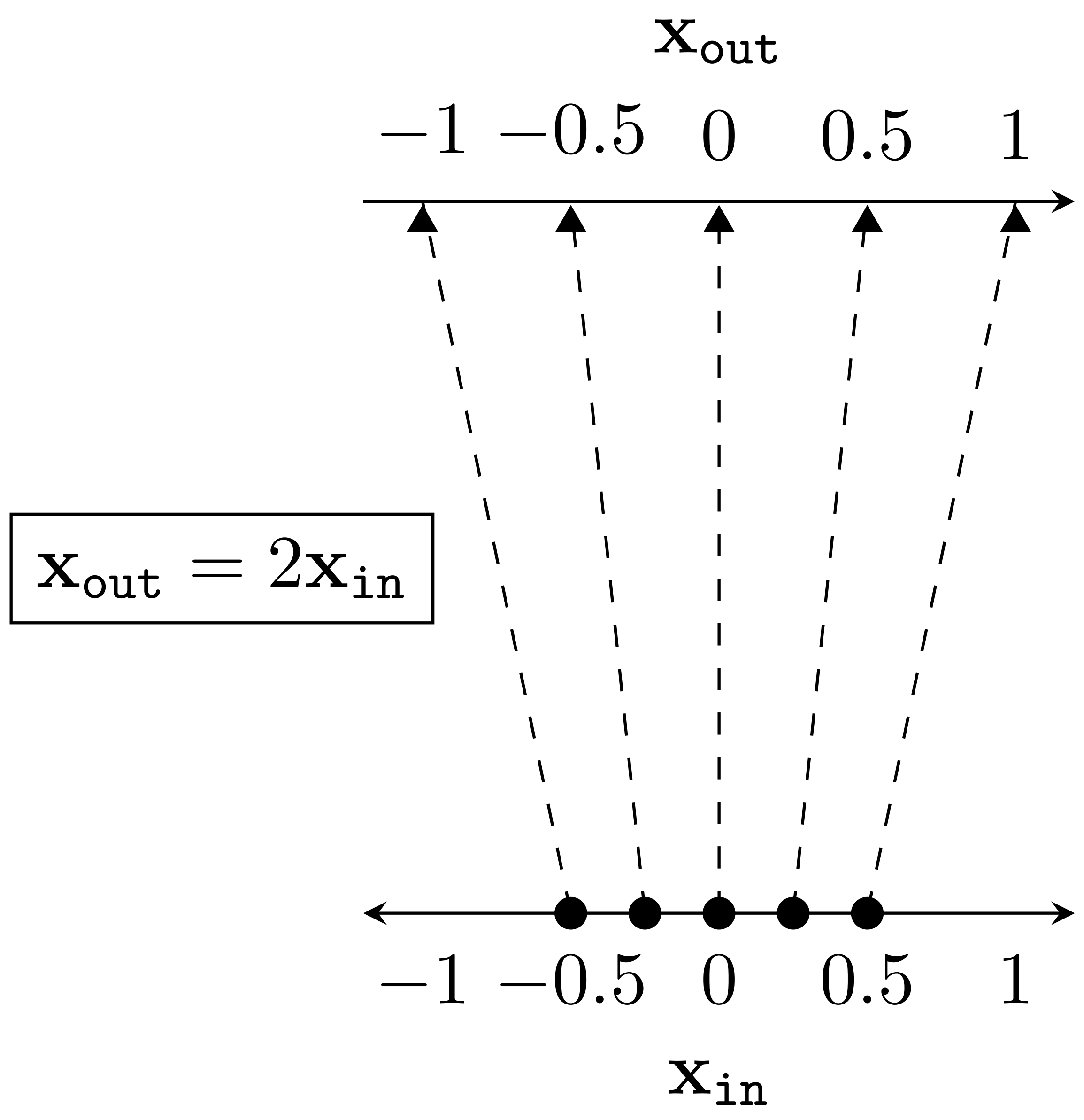
Two different ways to represent a function



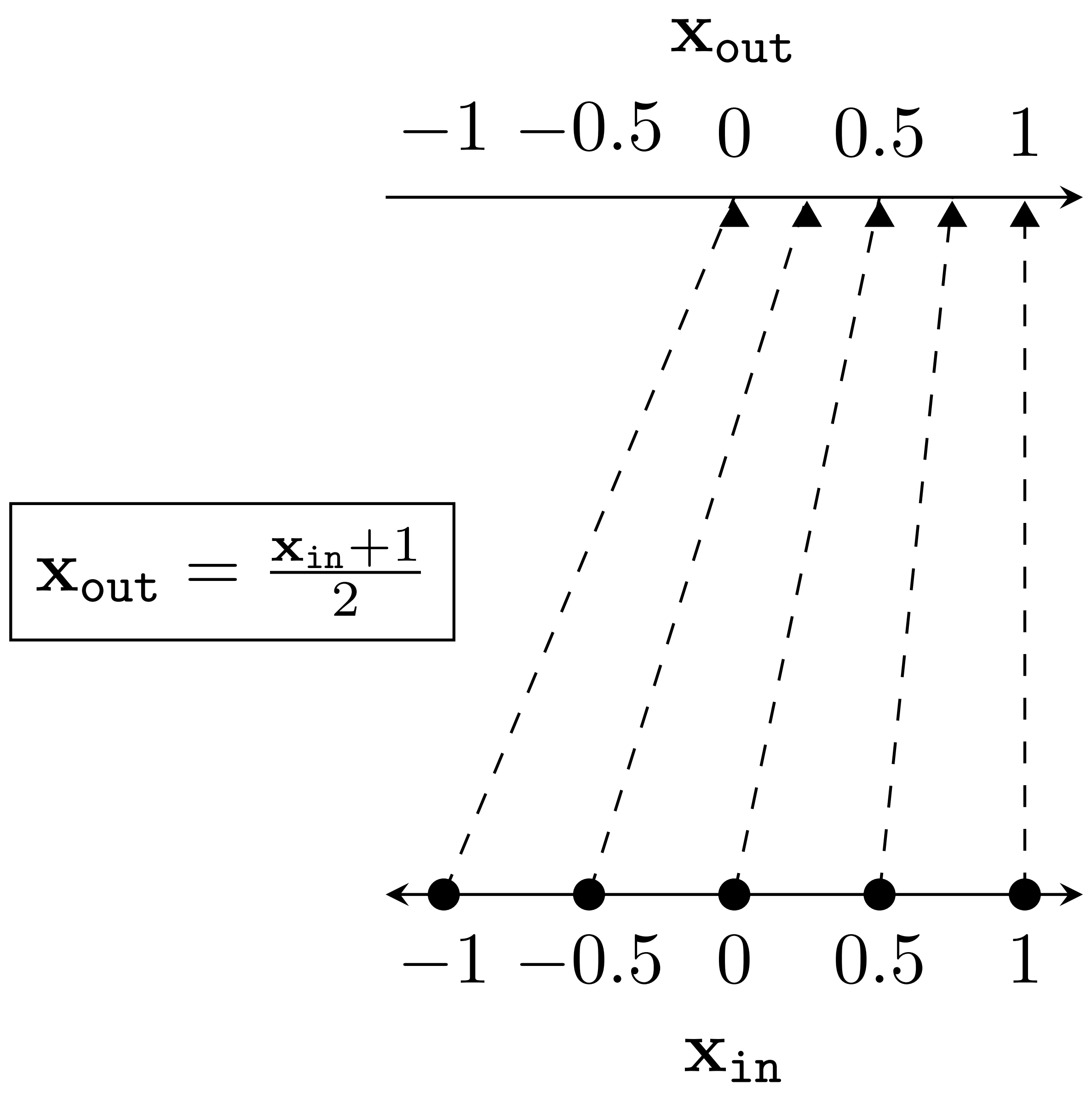
Two different ways to represent a function



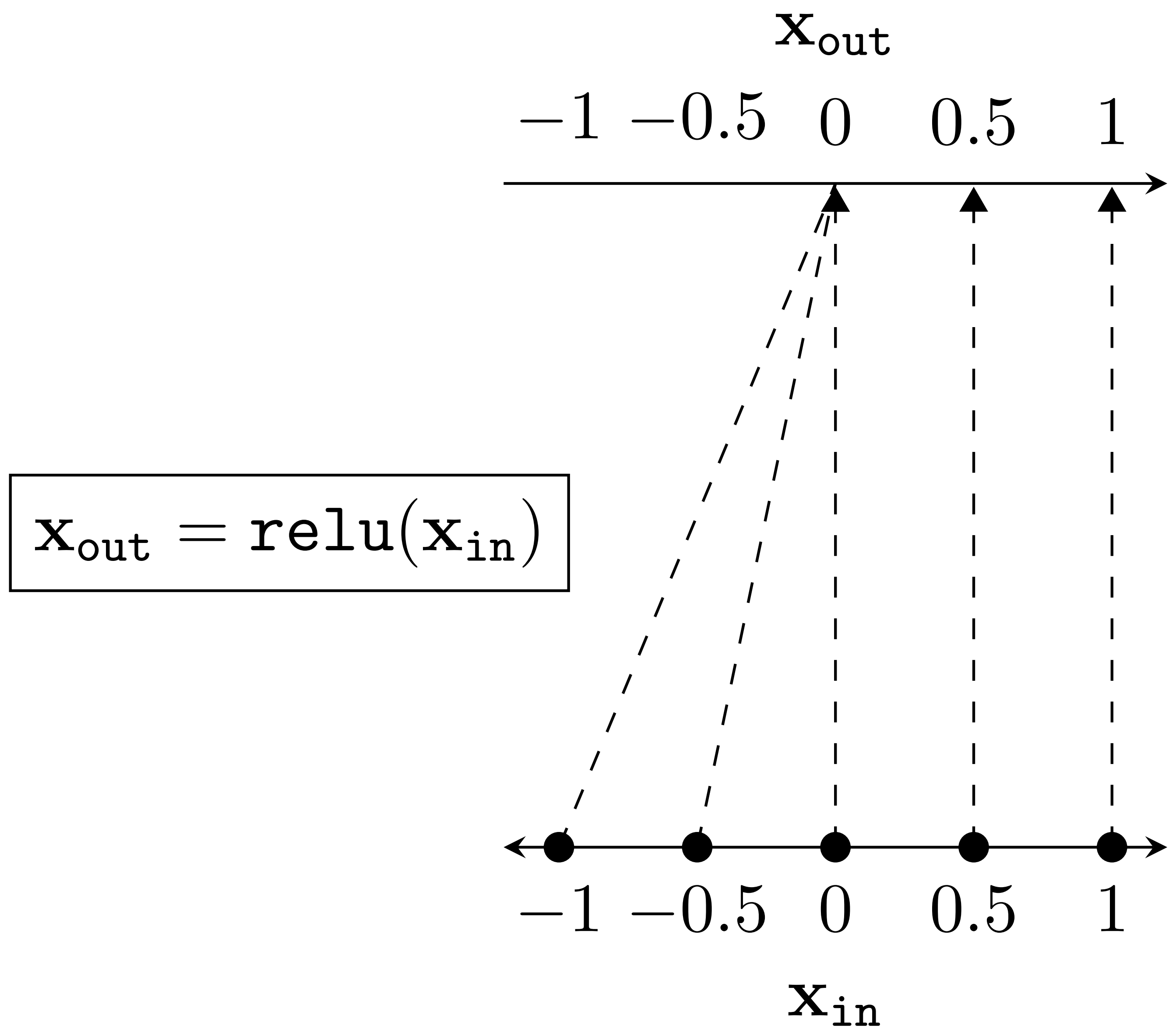
Data transformations for a variety of neural net layers



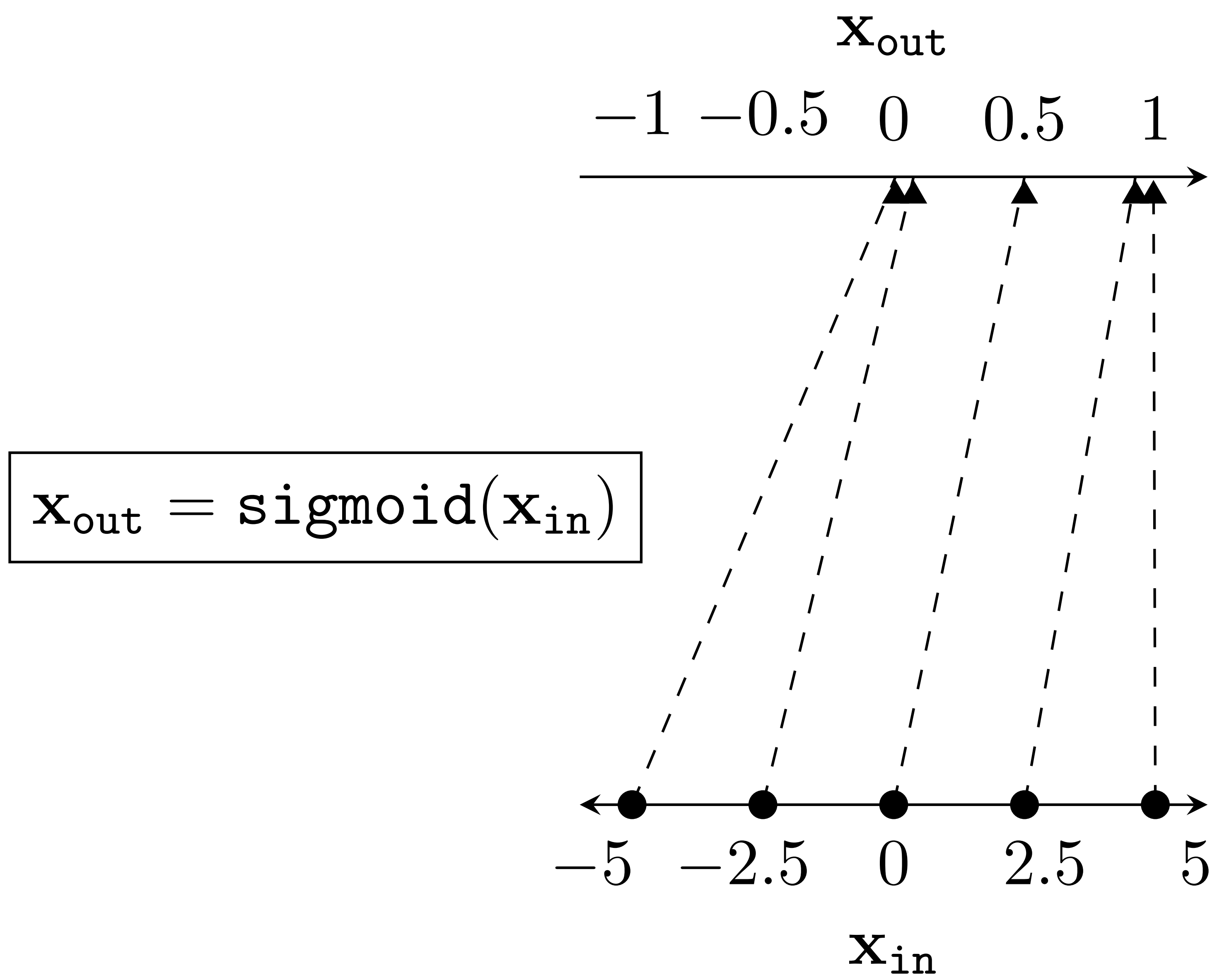
$$\mathbf{x}_{\mathtt{out}} = 2\mathbf{x}_{\mathtt{in}}$$



$$\mathbf{x}_{\mathtt{out}} = \frac{\mathbf{x}_{\mathtt{in}}+1}{2}$$



$$\mathbf{x}_{\mathtt{out}} = \mathtt{relu}(\mathbf{x}_{\mathtt{in}})$$



$$\mathbf{x}_{\mathtt{out}} = \mathtt{sigmoid}(\mathbf{x}_{\mathtt{in}})$$

🟥 Activations

🟦 Parameters

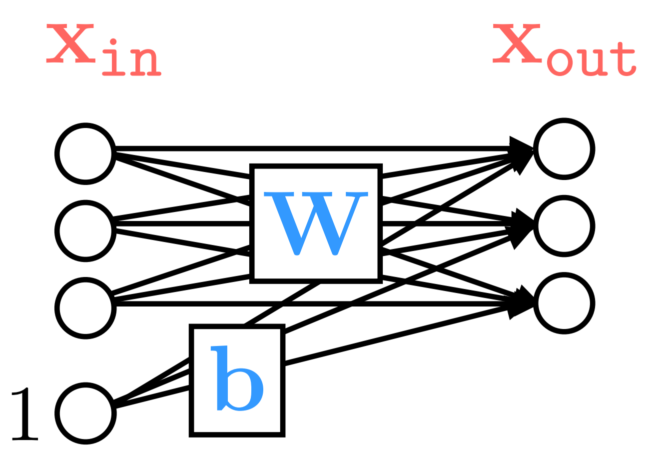
Wiring graph

Equation

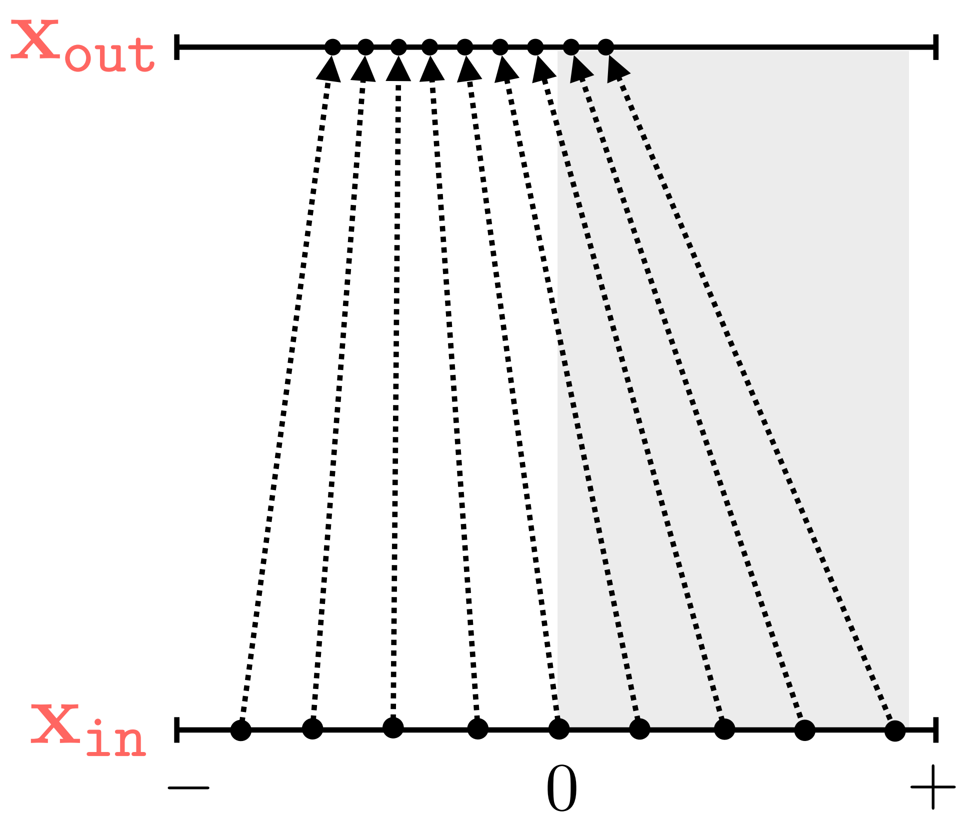
Mapping 1D

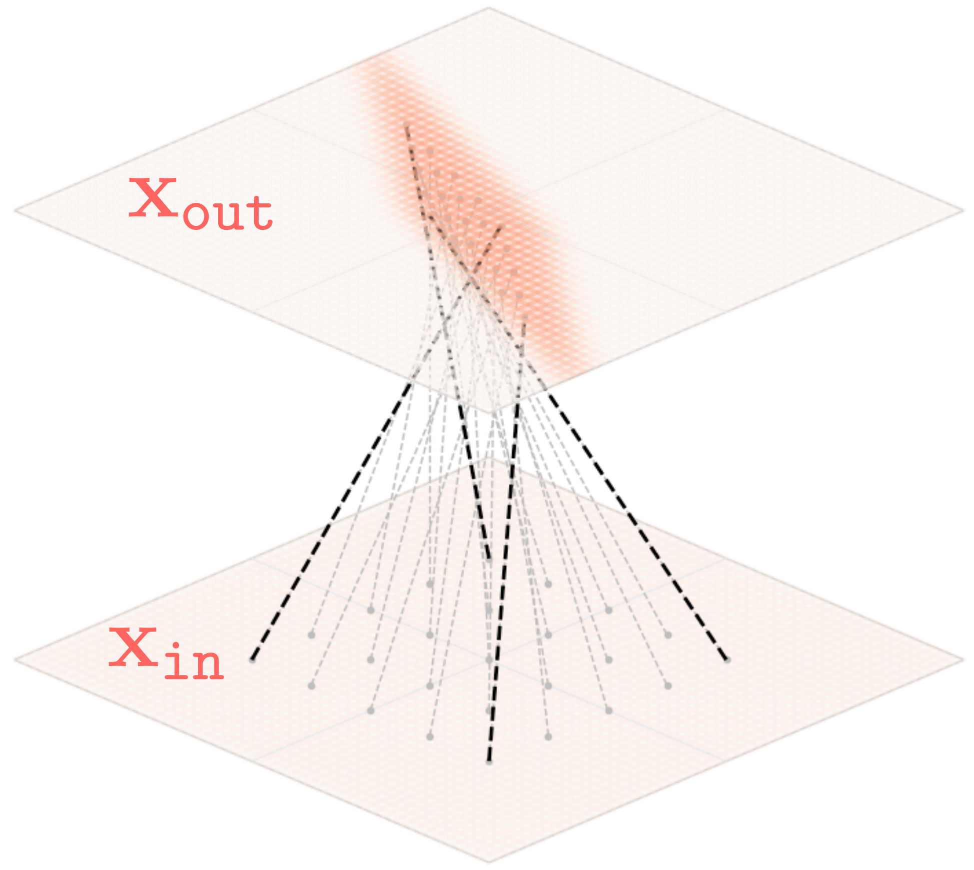
Mapping 2D

`linear`

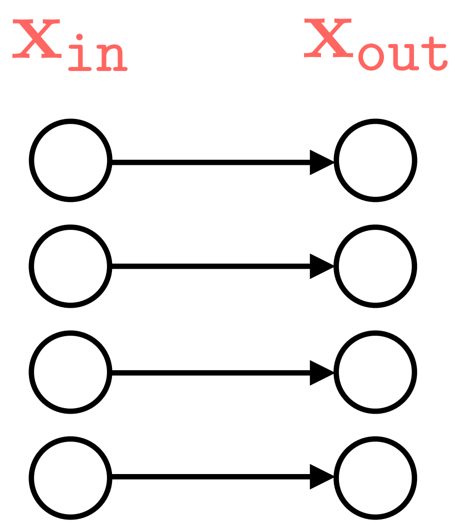


$$\mathbf{x}_{\text{out}} = \mathbf{W}\mathbf{x}_{\text{in}} + \mathbf{b}$$

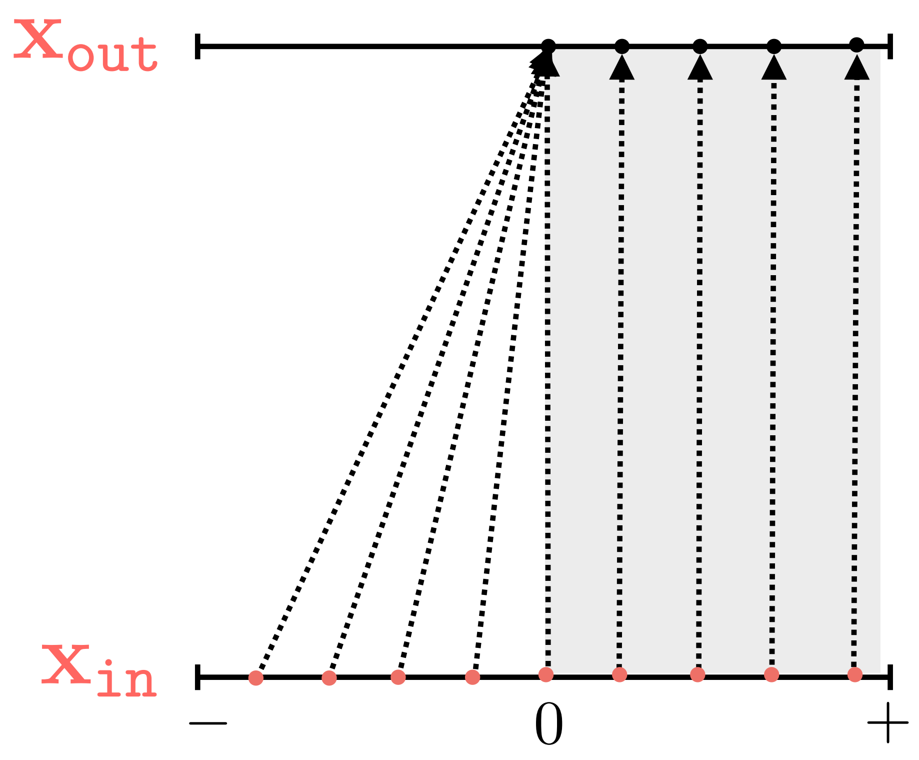


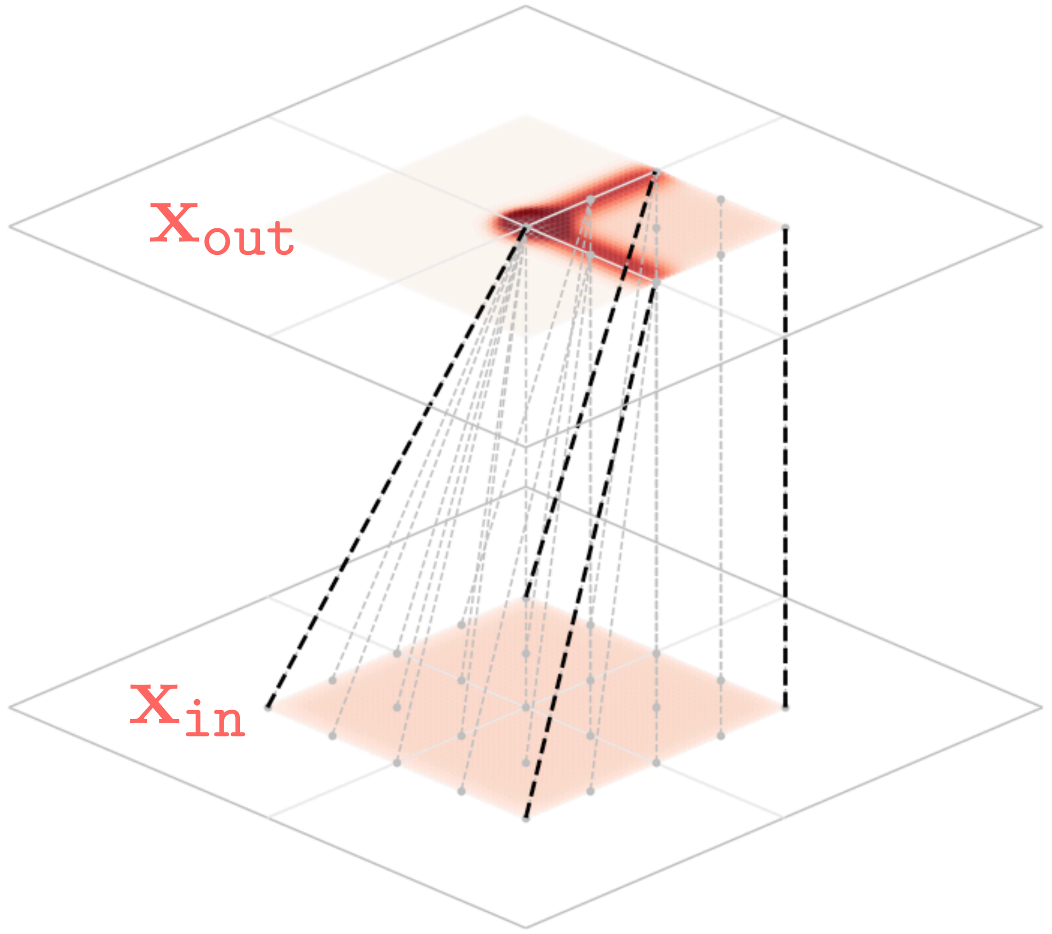


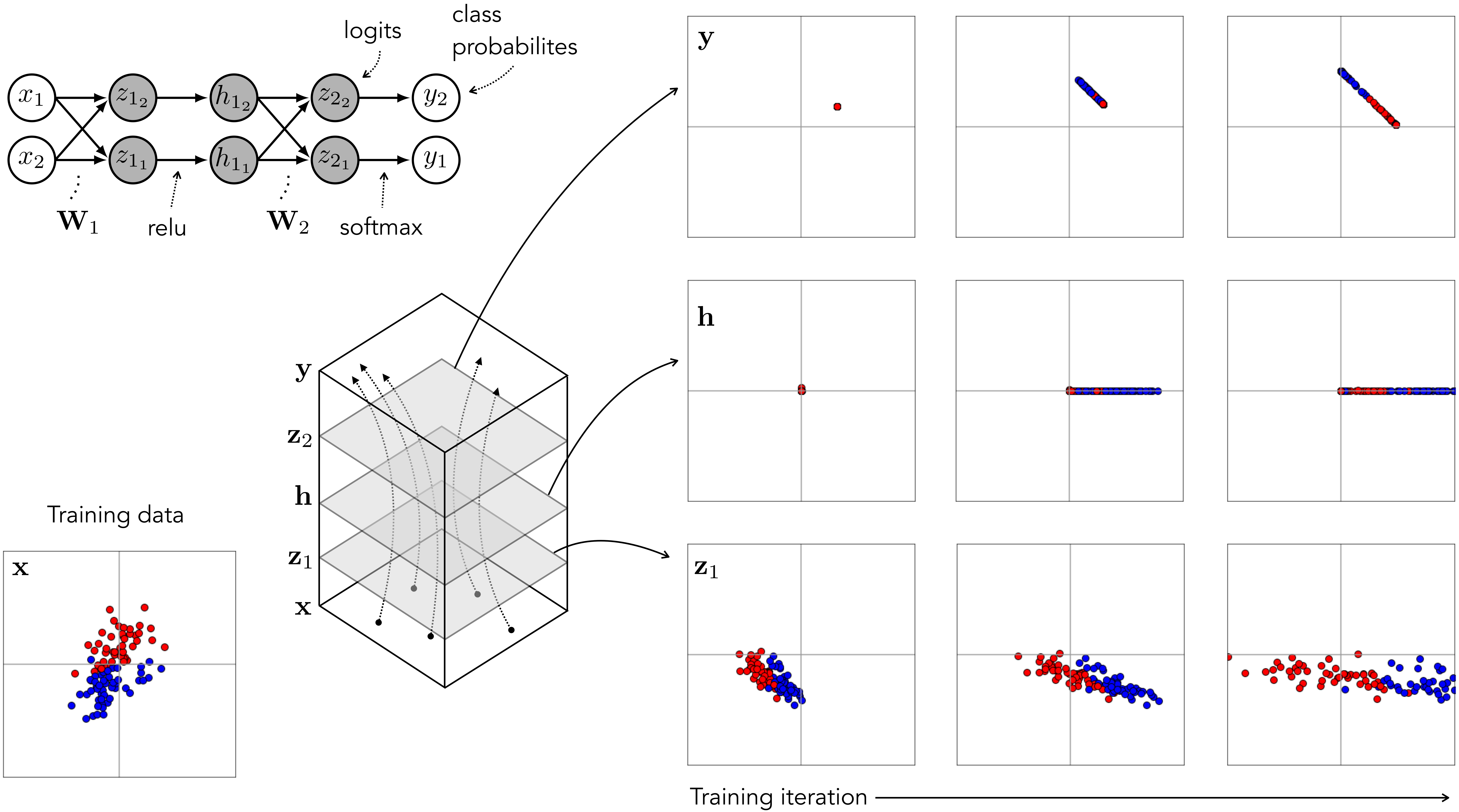
`relu`

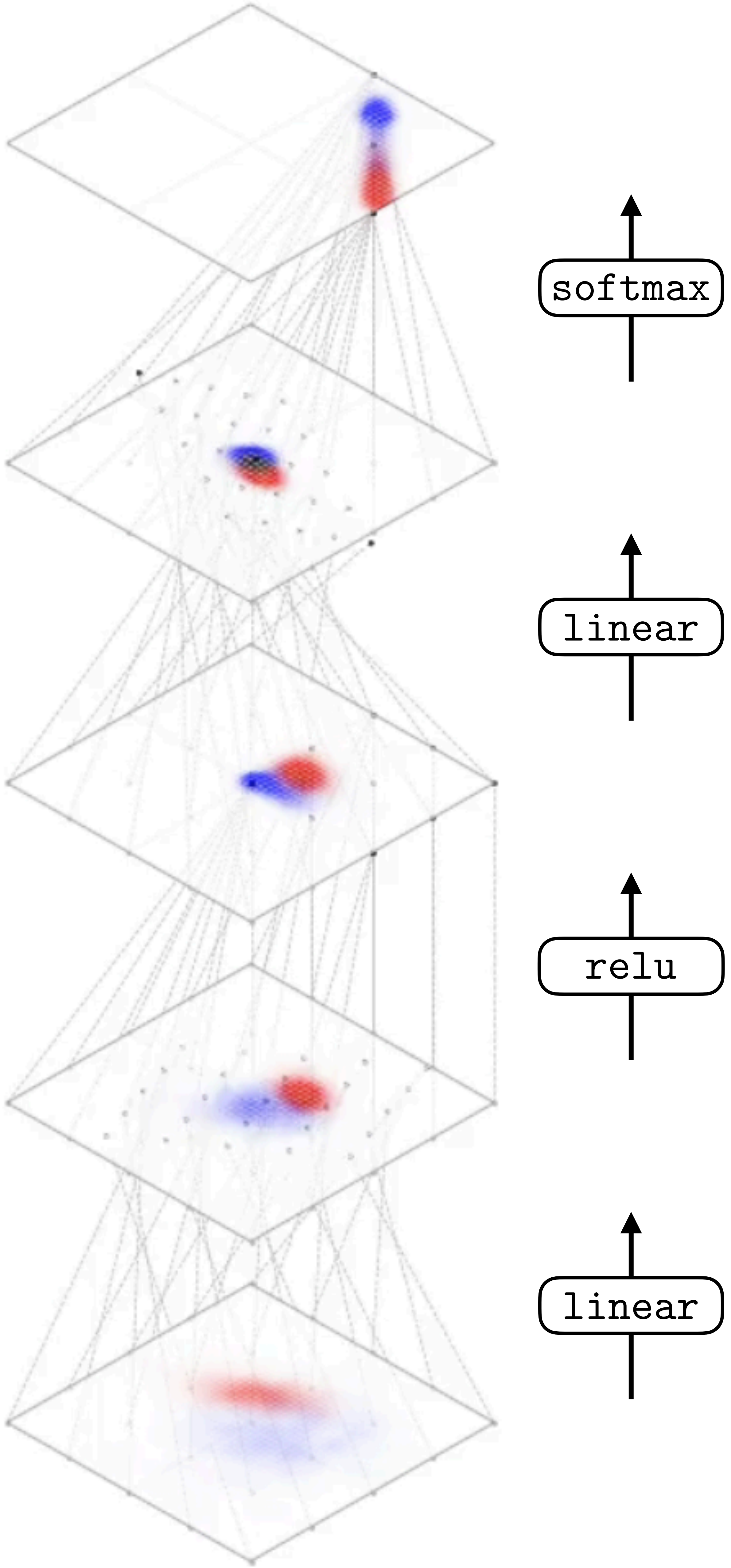


$$x_{\text{out}_i} = \max(x_{\text{in}_i}, 0)$$






class probabilites
logits
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
$\mathbf{y}$
$\mathbf{z}_2$
$\mathbf{h}$
$\mathbf{z}_1$
$\mathbf{x}$
Training data
$\mathbf{x}$
$\mathbf{y}$
$\mathbf{h}$
$\mathbf{z}_1$
Training iteration

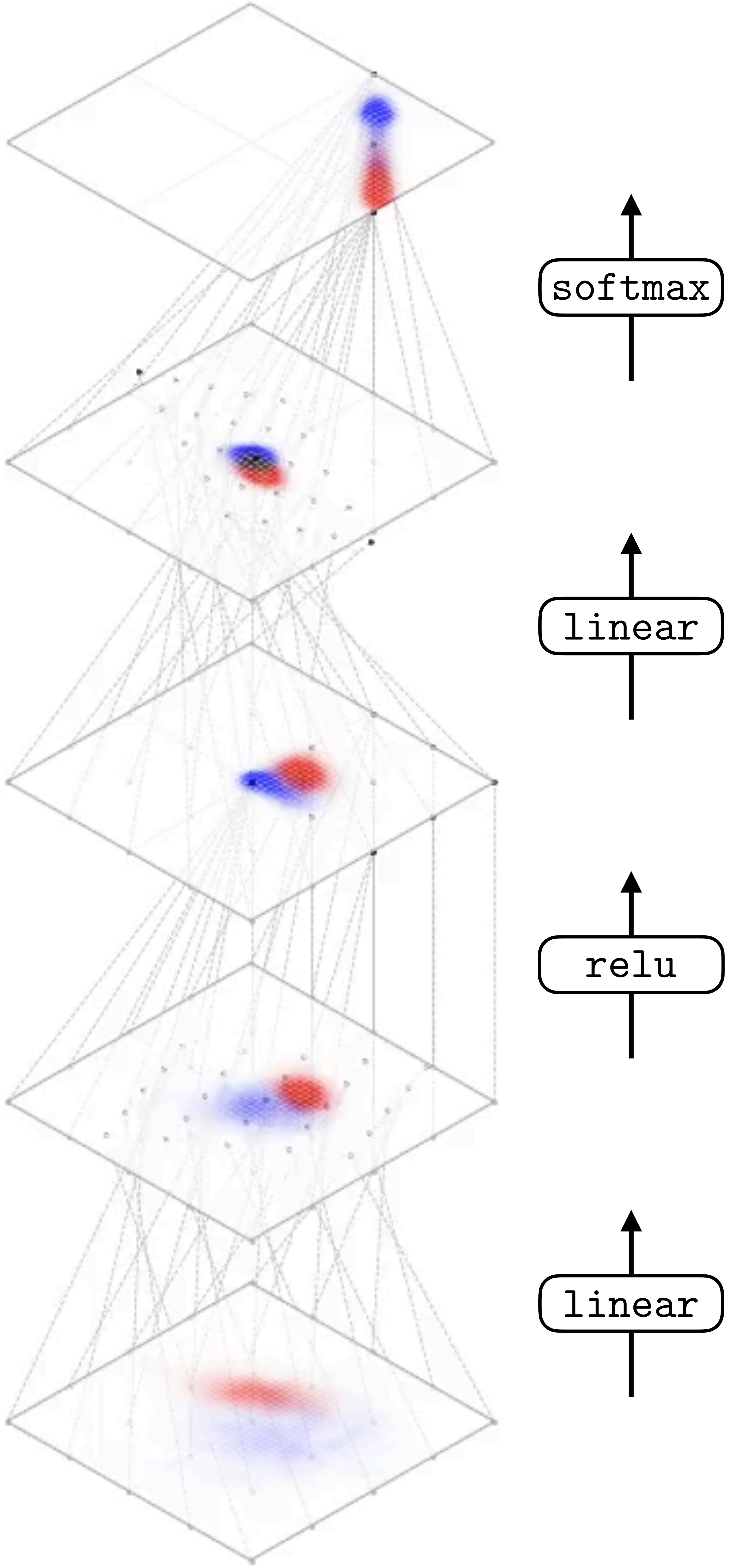


`softmax`

`linear`

`relu`

`linear`

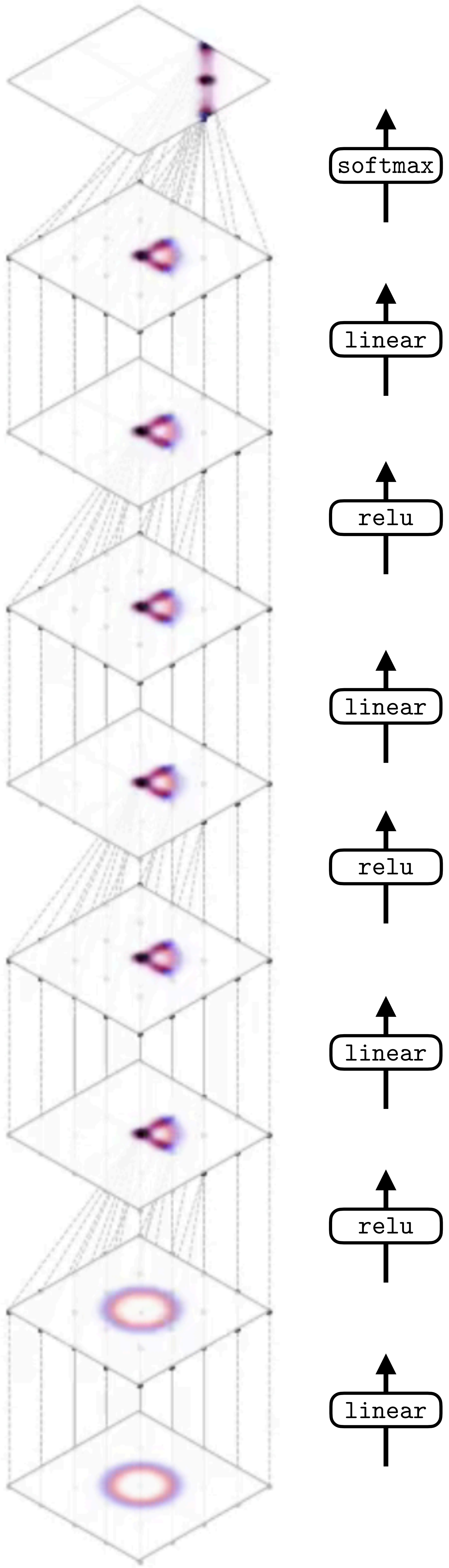


`softmax`

`linear`

`relu`

`linear`



softmax

linear

relu

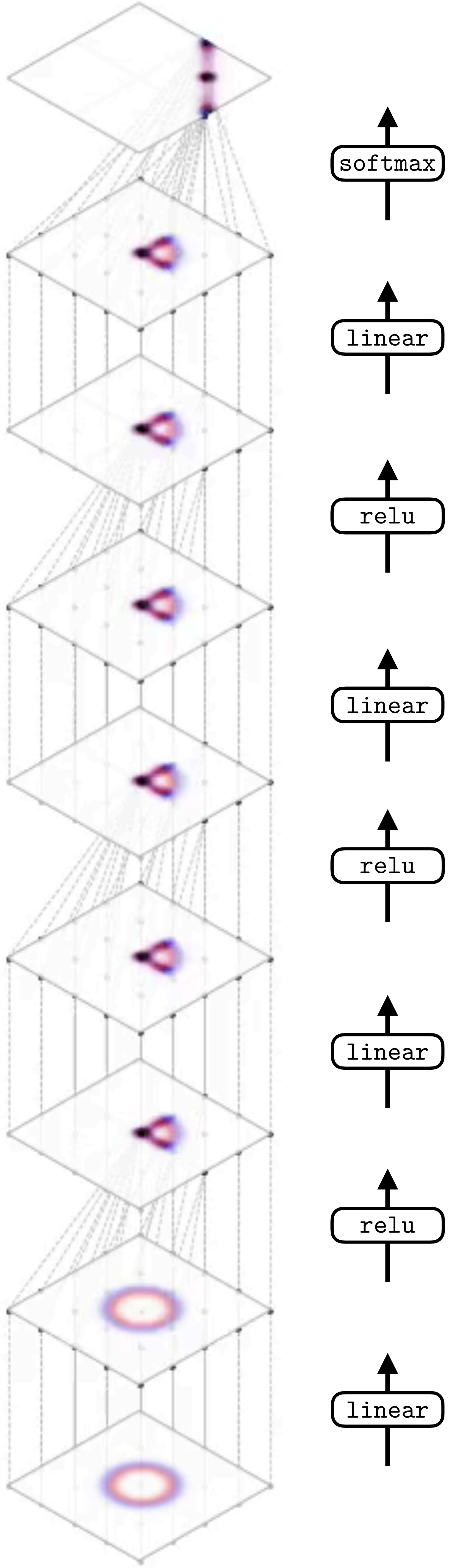
linear

relu

linear

relu

linear



softmax

linear

relu

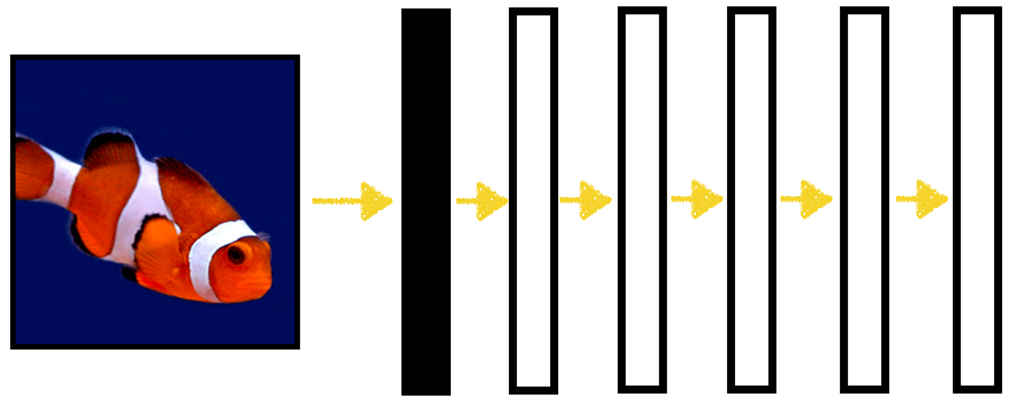
linear

relu

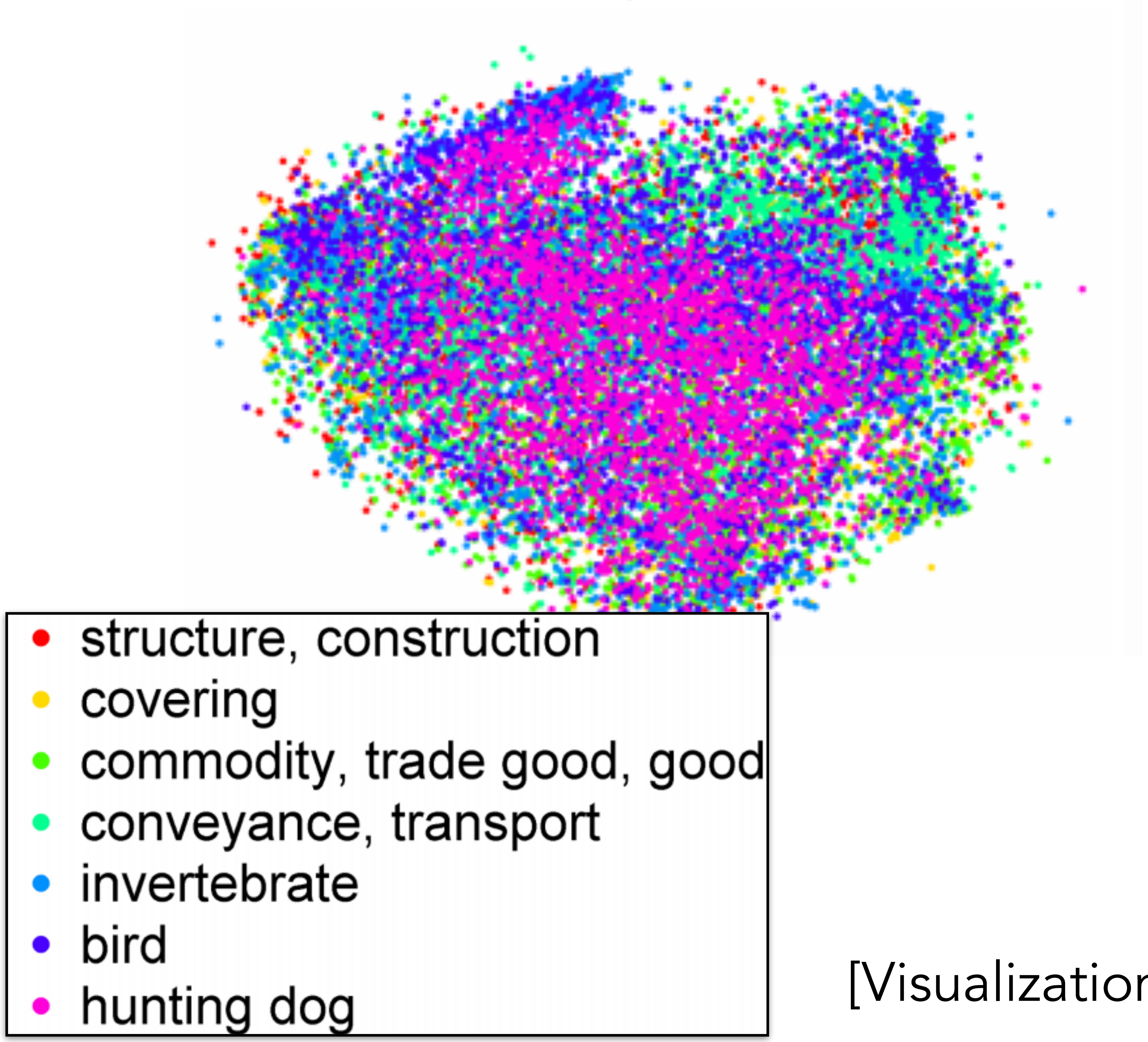
linear

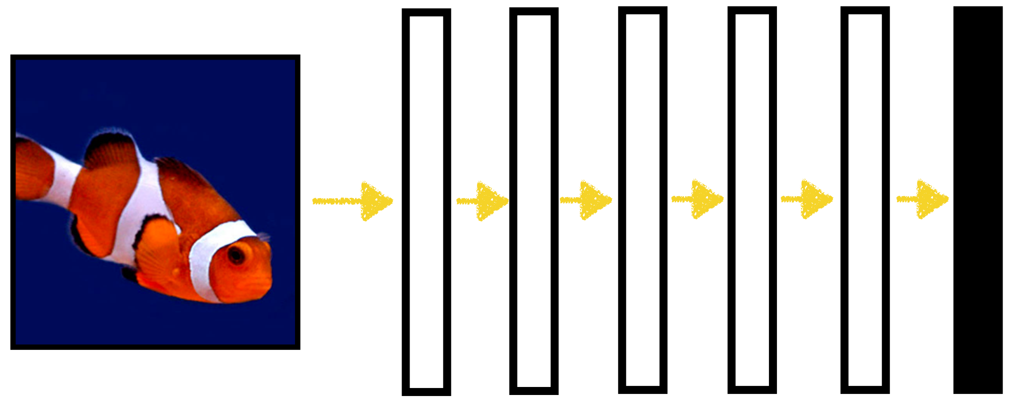
relu

linear

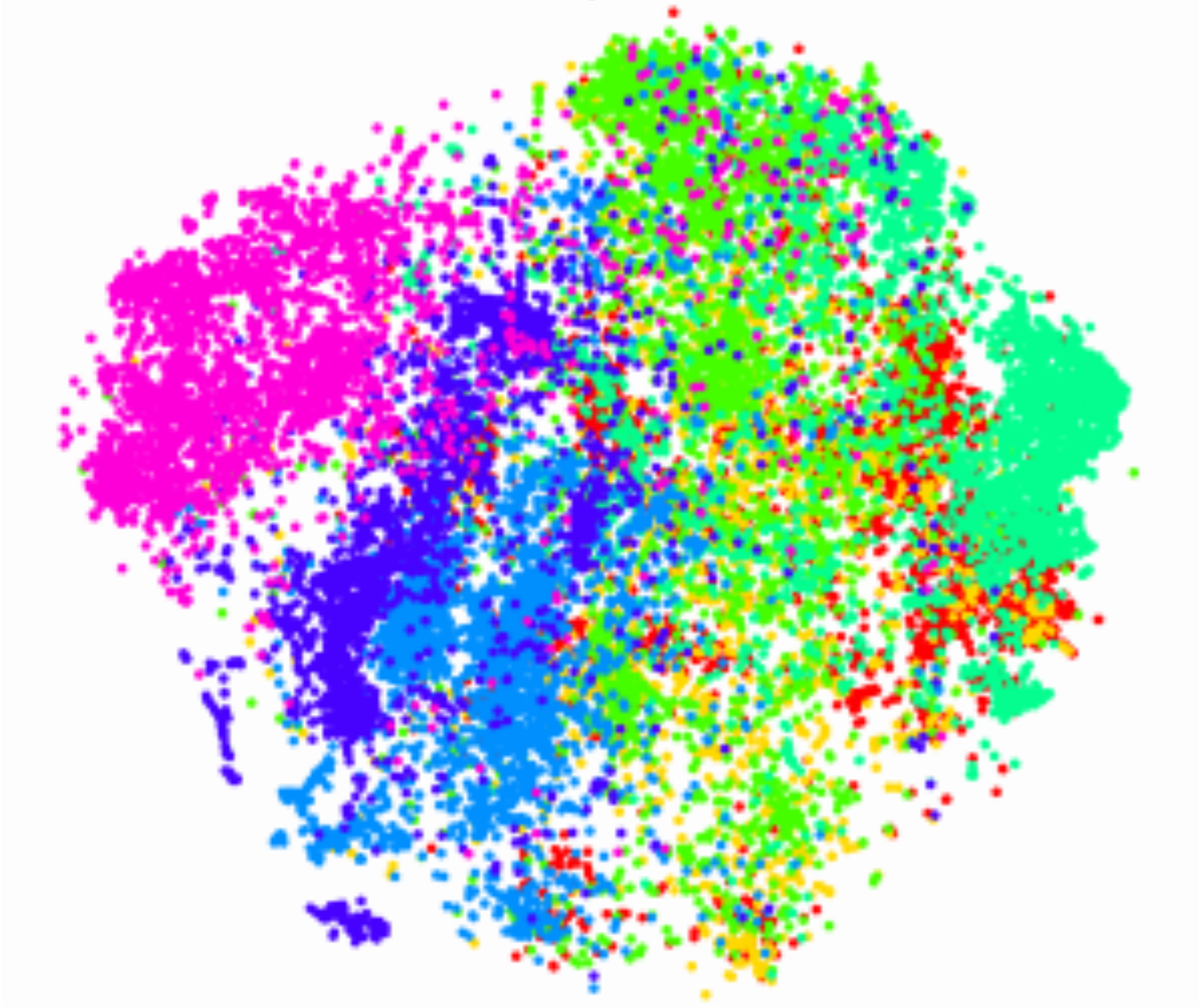


Layer 1 representation





Layer 6 representation



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

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

1. Introduction to Deep Learning

- Brief history
- Basic formulation *(hierarchical processing)*
- Optimization via gradient descent
- Layer types *(Linear, Pointwise non-linearity)*
- Everything is a tensor
- Deep nets as data transformers