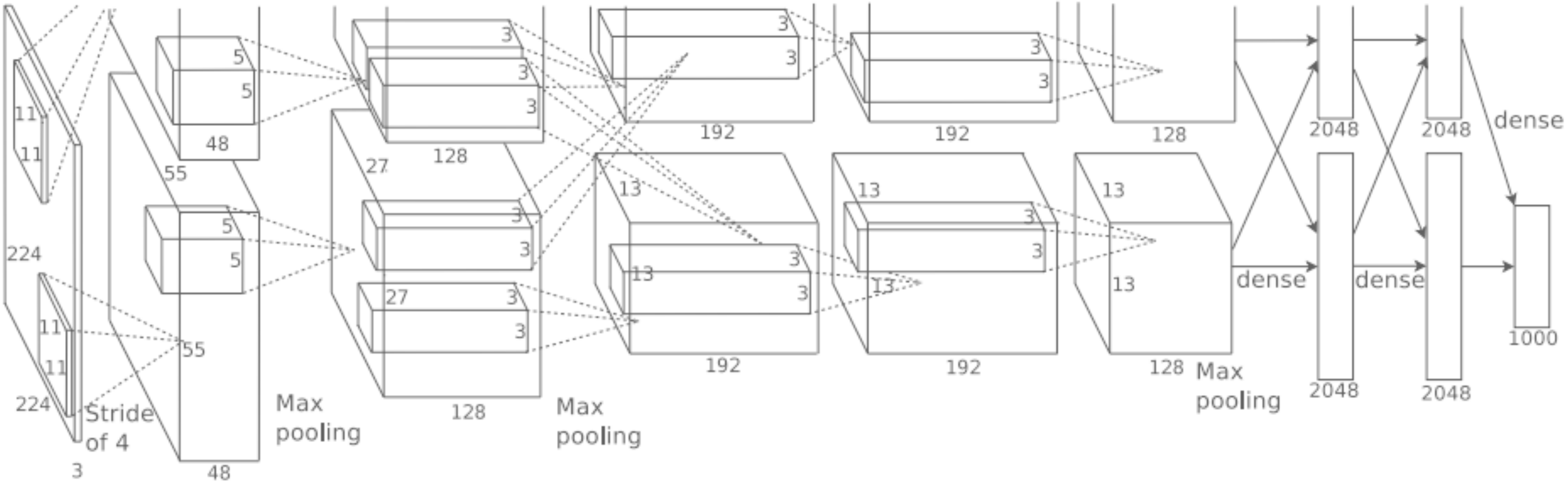
Lecture 9

Neural Networks



9. Neural Networks

- Brief history
- Basic formulation *(hierarchical processing)*
- Optimization via gradient descent
- Layer types *(Linear, Pointwise non-linearity)*
- Linear classification with a perceptron
- Batch processing
- Regularizers
- Normalization

Deep learning

Modeling the visual world is incredibly complicated. We need high capacity models.

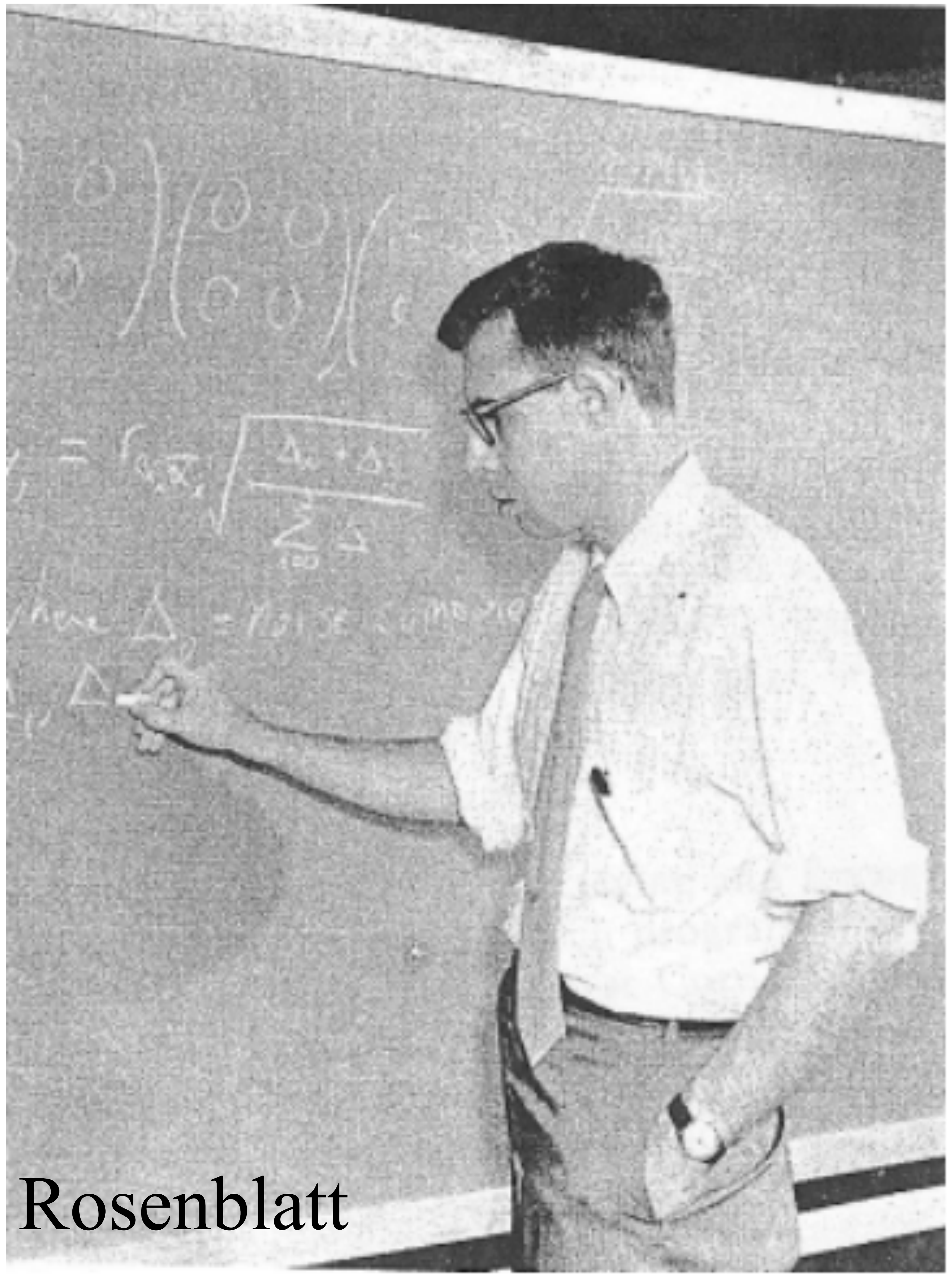
In the past, we didn't have enough data to fit these models. But now we do!

We want a class of **high capacity models** that are **easy to optimize**.

Deep neural networks!

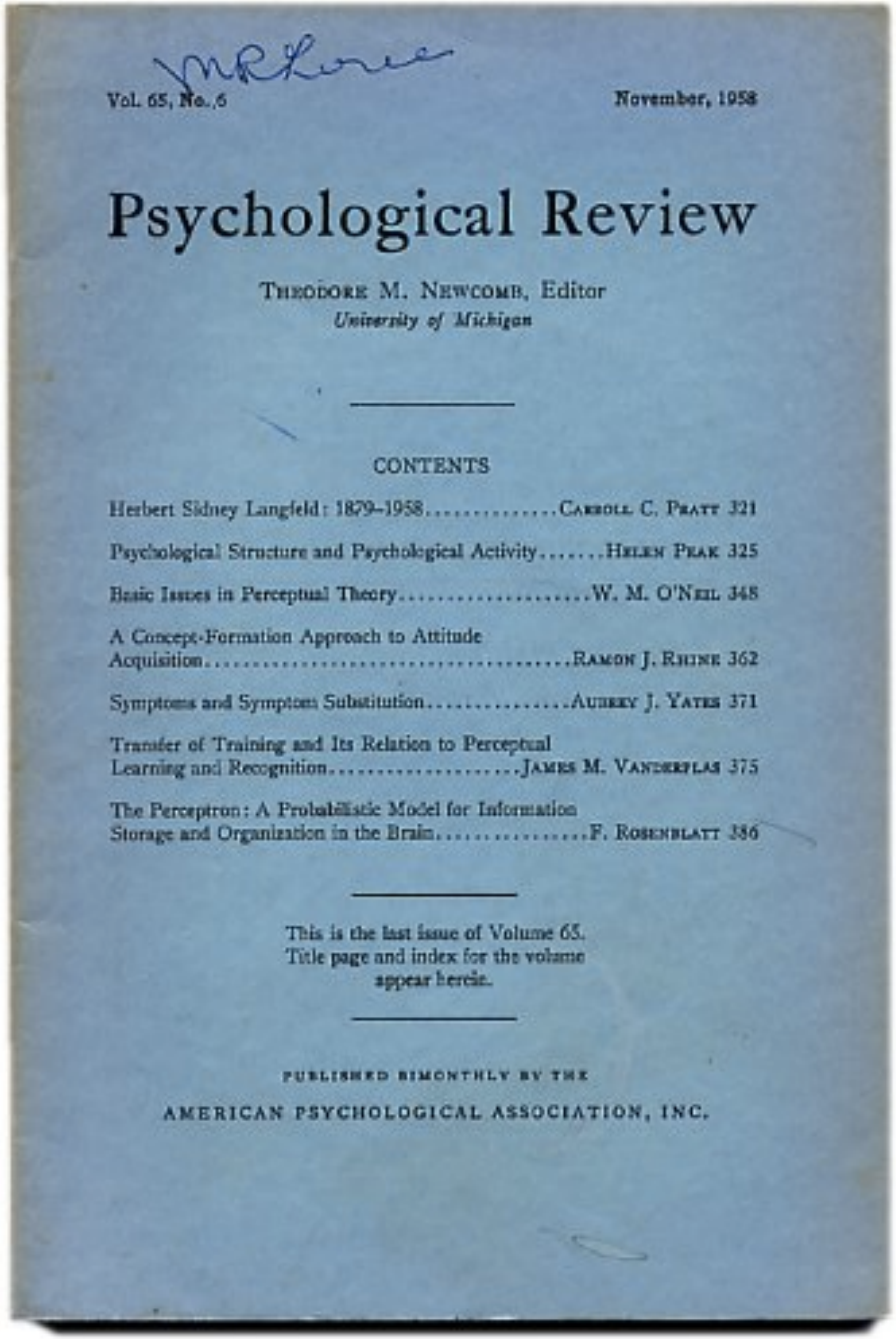
A brief history of Neural Networks

Perceptrons, 1958



Rosenblatt

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



Vol. 65, No. 6 November, 1958

Psychological Review

Theodore M. Newcomb, Editor
University of Michigan

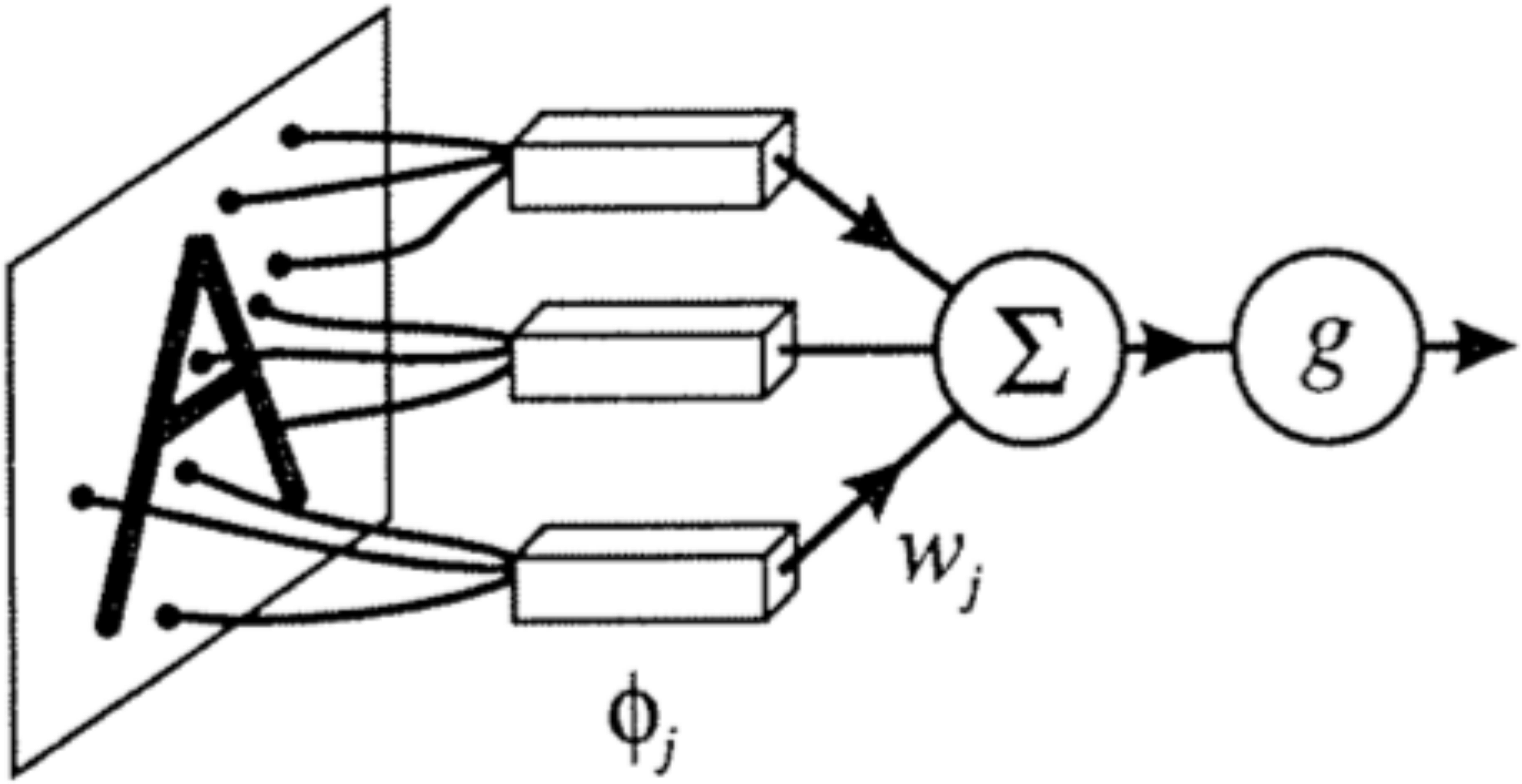
CONTENTS

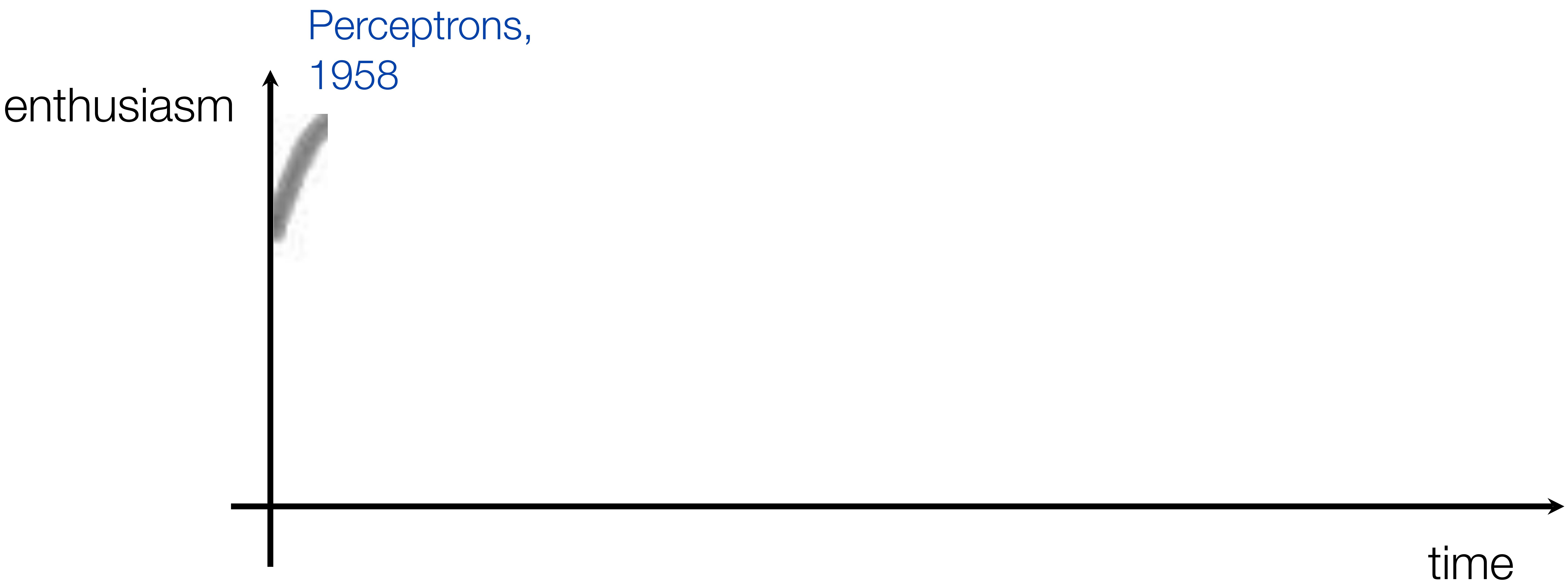
This is the last issue of Volume 65. Title page and index for the volume appear herein.

PUBLISHED BIMONTHLY BY THE
AMERICAN PSYCHOLOGICAL ASSOCIATION, INC.

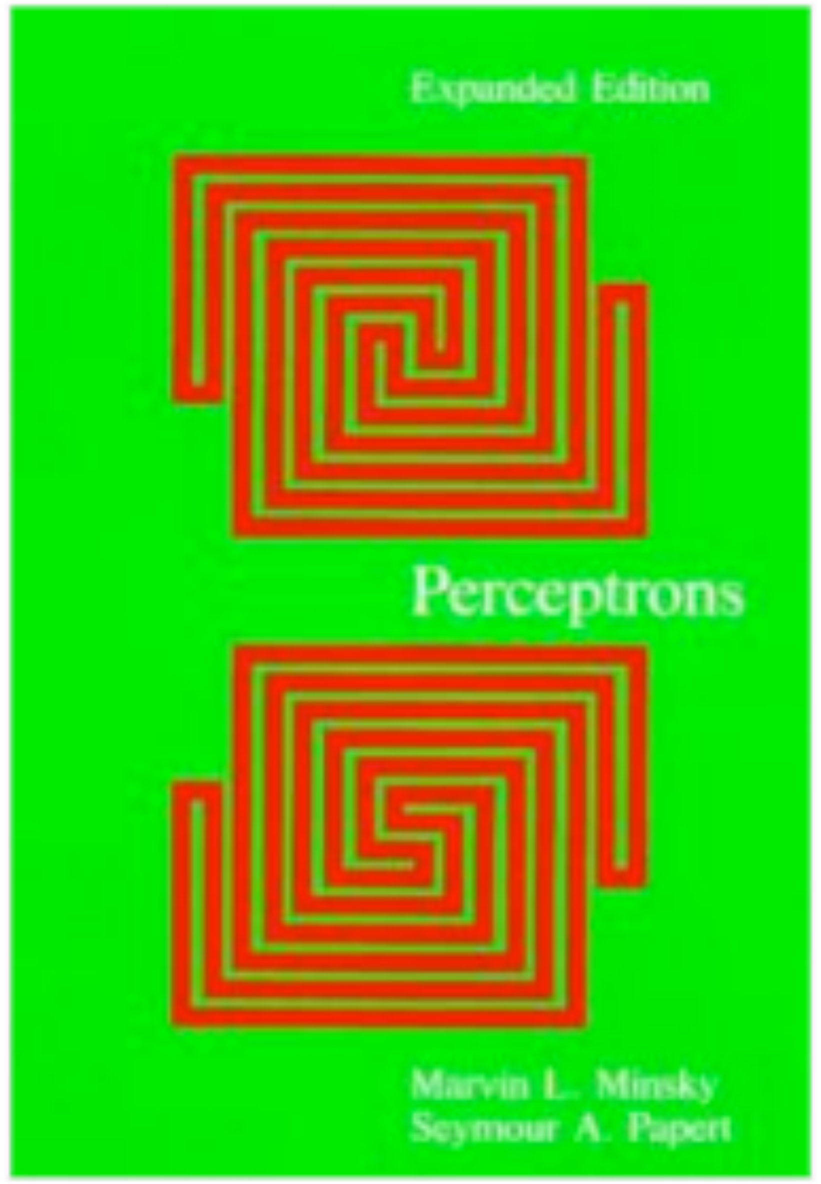
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf

Perceptrons, 1958




Perceptrons,
1958
enthusiasm
time

Minsky and Papert, Perceptrons, 1972



FOR BUYING OPTIONS, START HERE

Select Shipping Destination

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

Perceptrons, expanded edition

An Introduction to Computational Geometry

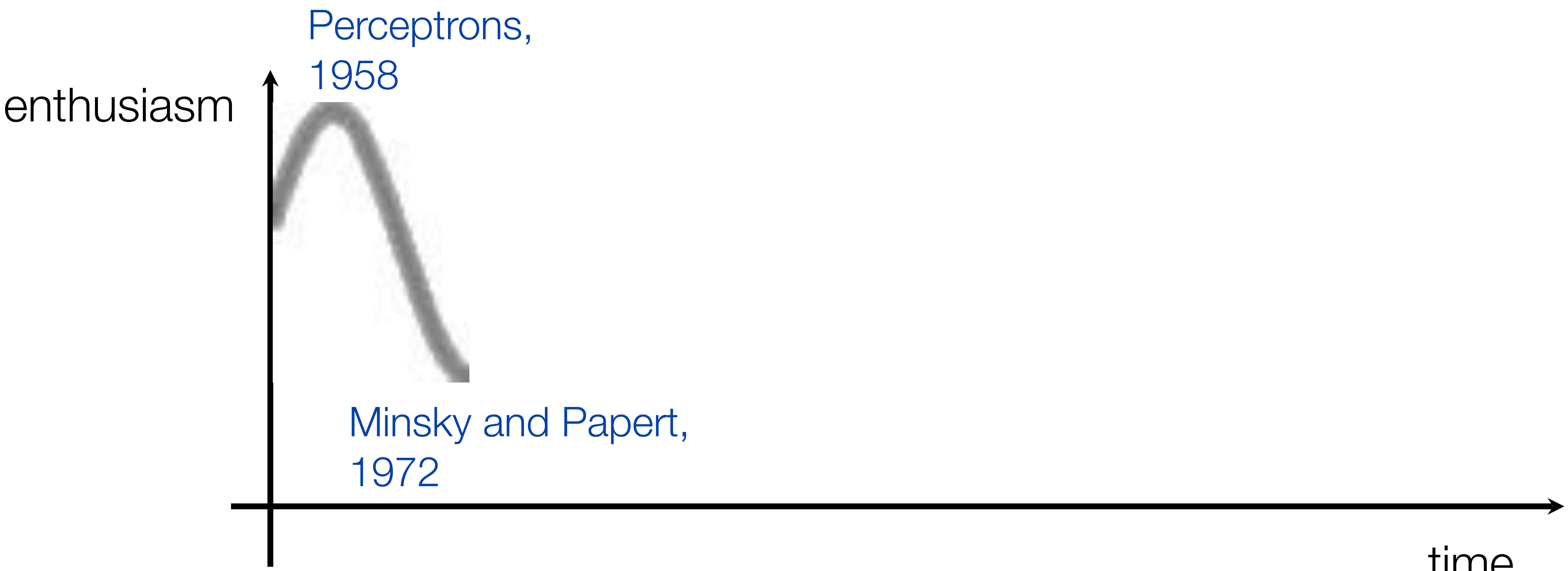
By Marvin Minsky and Seymour A. Papert

Overview

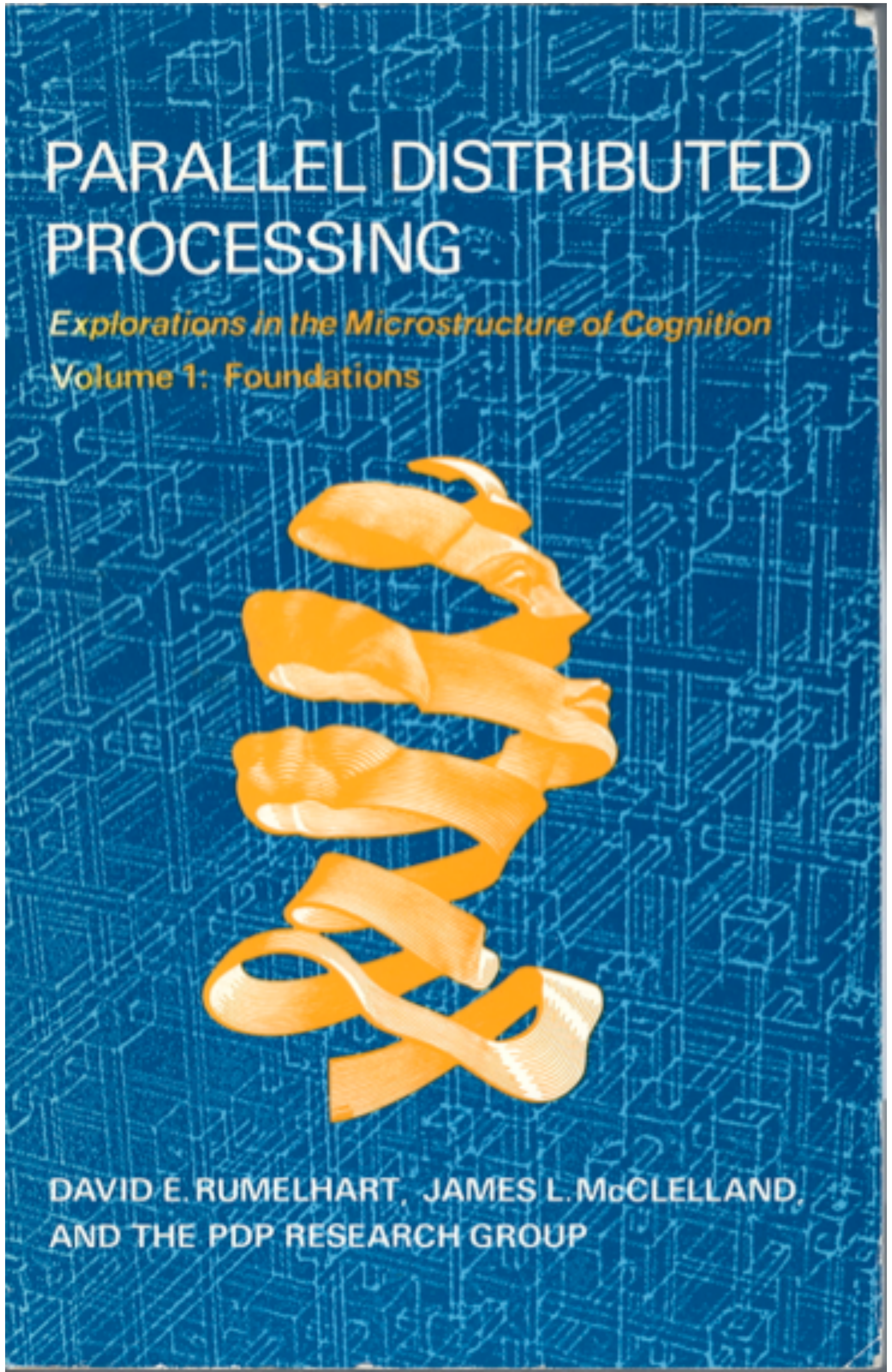
Perceptrons - the first systematic study of parallelism in computation - has remained a classical work on threshold automata networks for nearly two decades. It marked a historical turn in artificial intelligence, and it is required reading for anyone who wants to understand the connectionist counterrevolution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

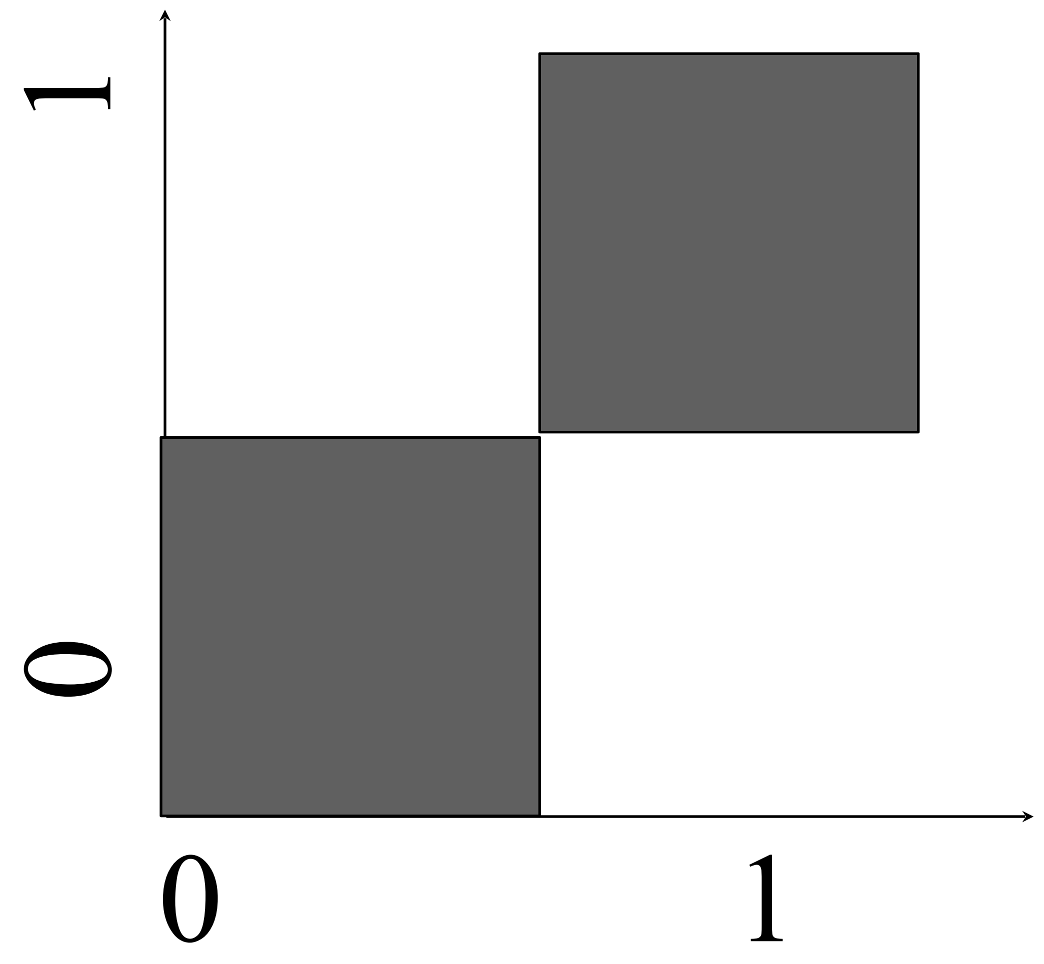

Perceptrons,
1958
enthusiasm
Minsky and Papert,
1972
time

Parallel Distributed Processing (PDP), 1986

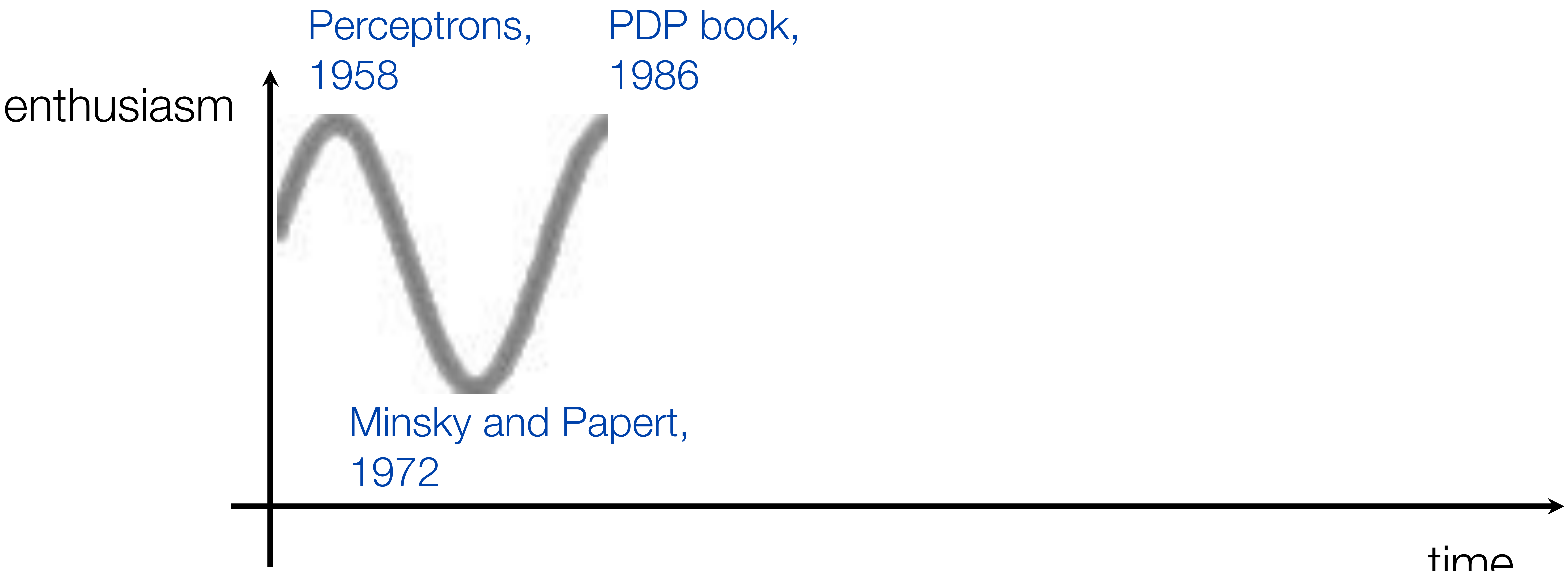


XOR problem

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
enthusiasm
Minsky and Papert,
1972
time

LeCun conv nets, 1998

PROC. OF THE IEEE, NOVEMBER 1998 7

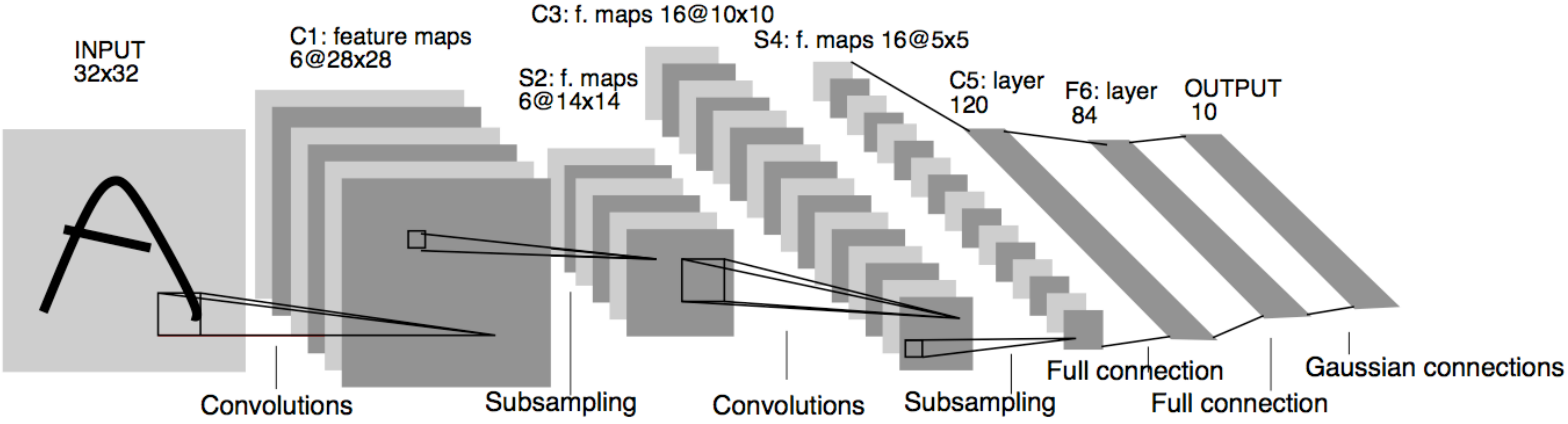


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

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

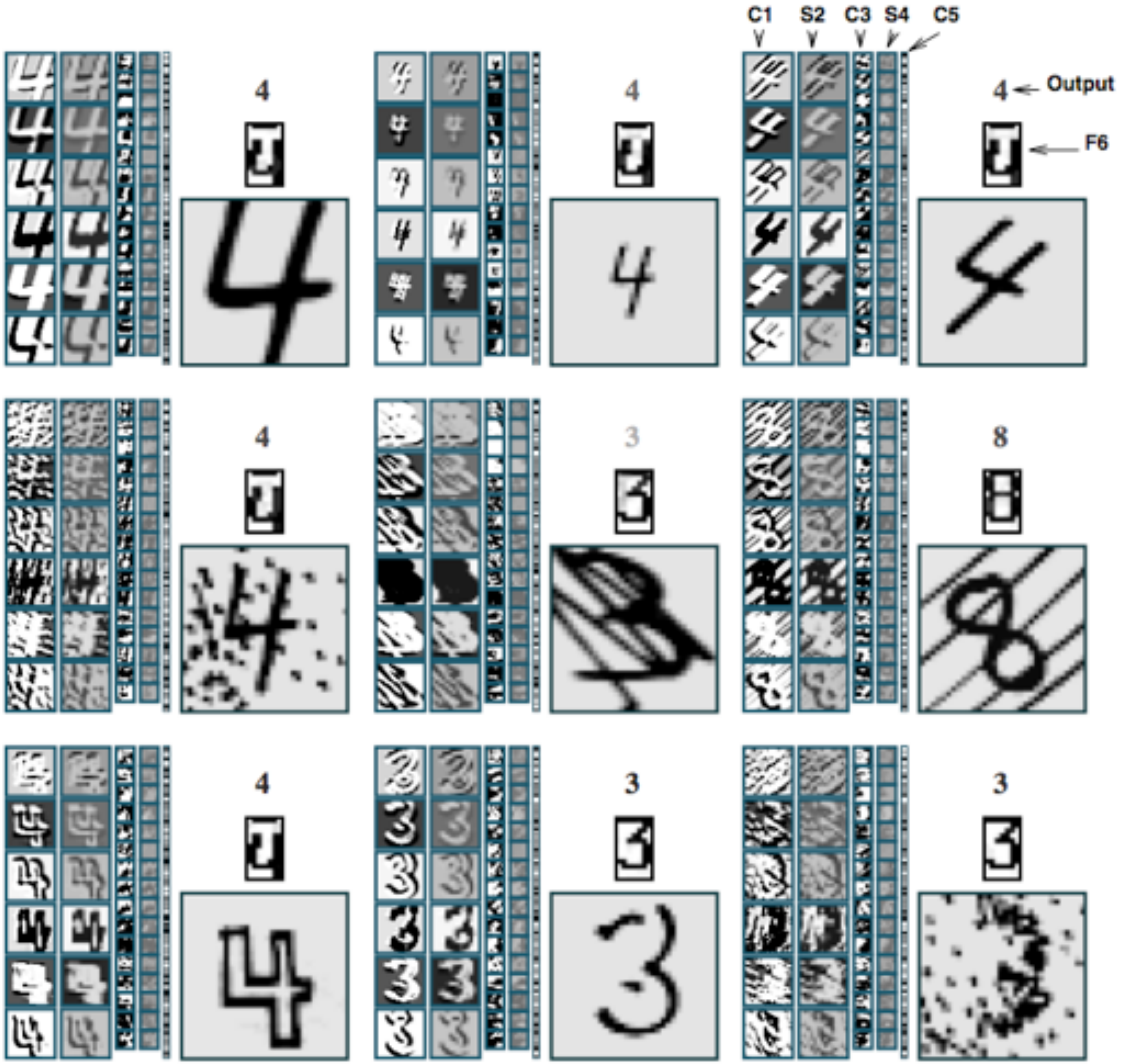
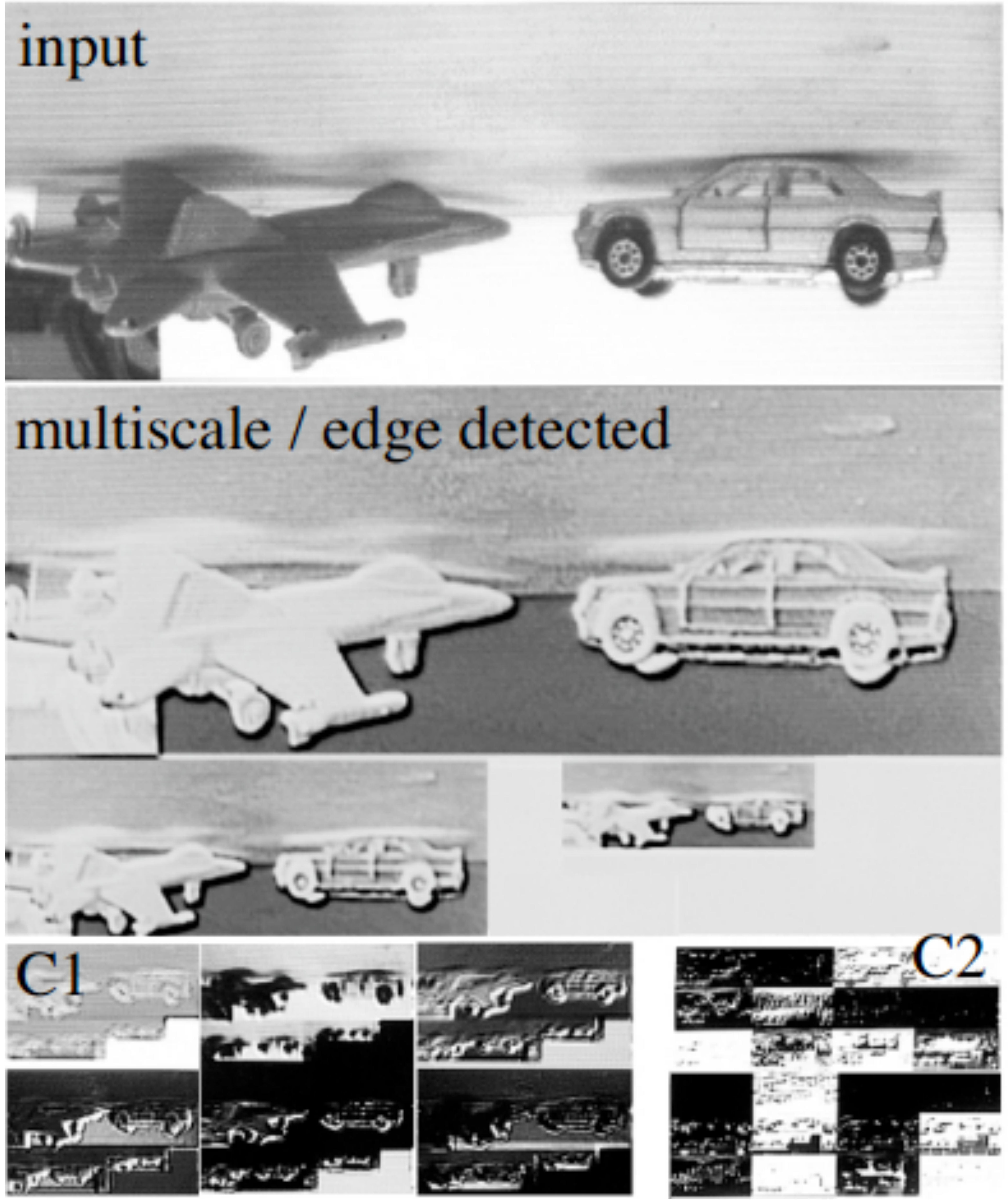


Fig. 13. Examples of unusual, distorted, and noisy characters correctly recognized by LeNet-5. The grey-level of the output label represents the penalty (lighter for higher penalties).



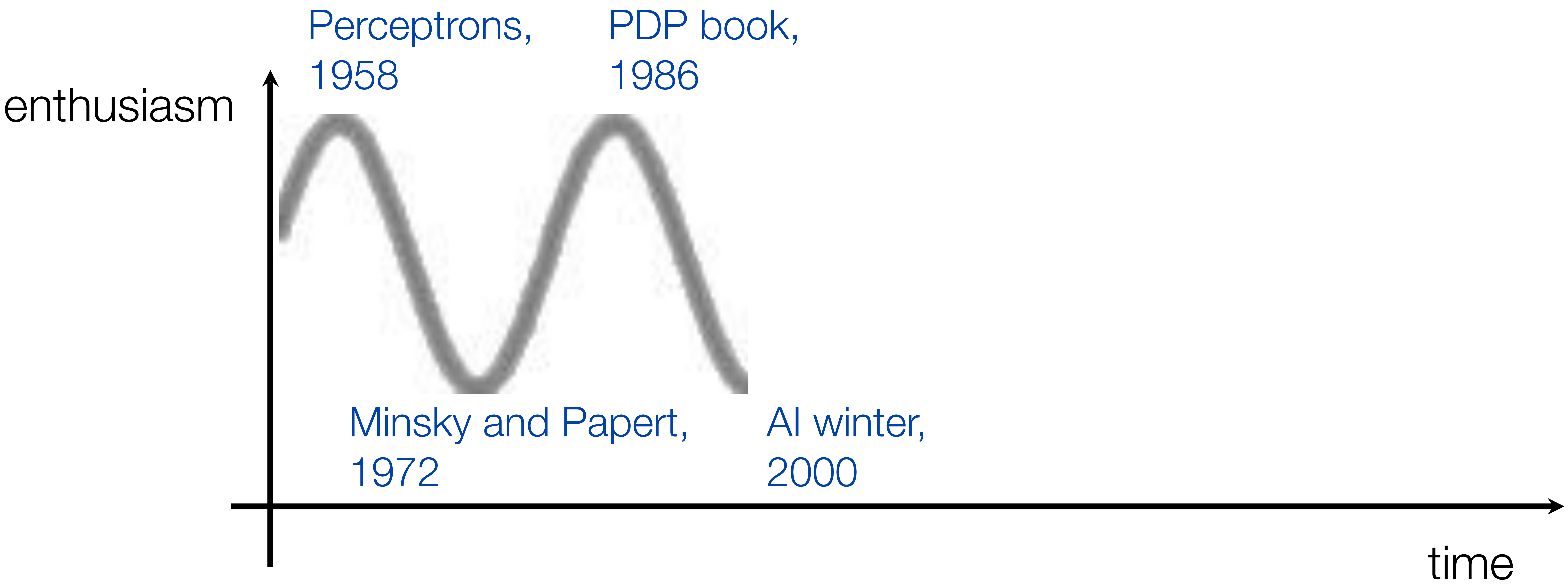
http://pub.clement.farabet.net/ecvw09.pdf

Neural networks to recognize handwritten digits?
yes

Neural networks for tougher problems?
not really

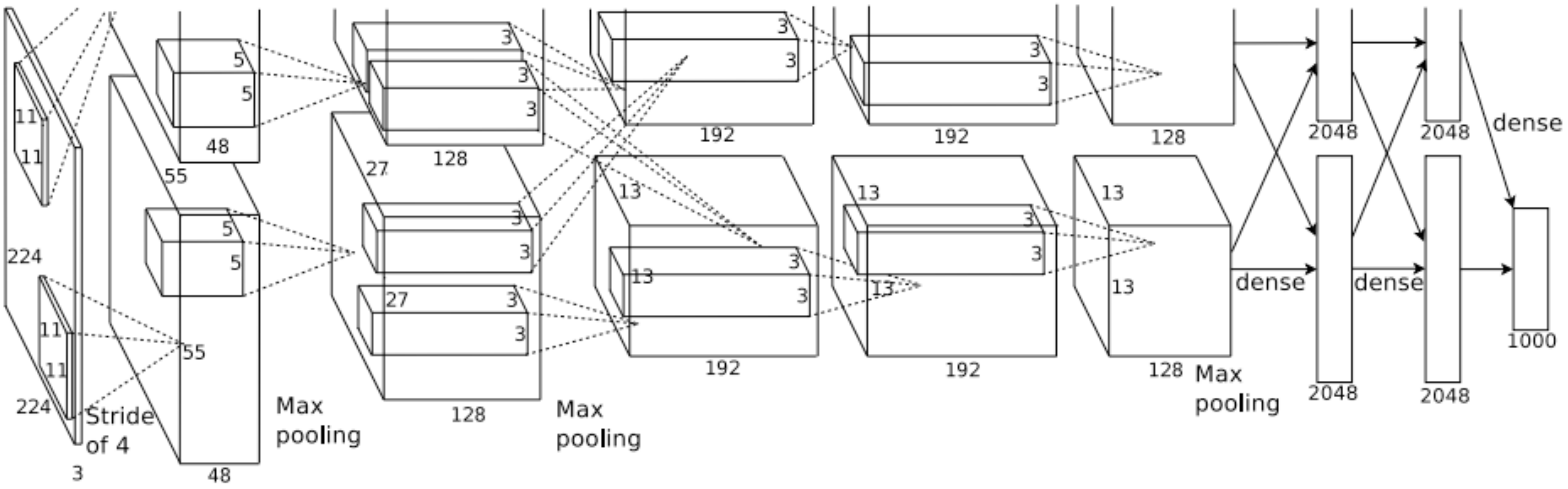
Neural Information Processing Systems 2000

- 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:
  - title words predictive of paper acceptance: “Belief Propagation” and “Gaussian”.
  - title words predictive of paper rejection: “Neural” and “Network”.


Perceptrons,
1958
PDP book,
1986
enthusiasm
Minsky and Papert,
1972
AI winter,
2000
time

Krizhevsky, Sutskever, and Hinton, NeurIPS 2012

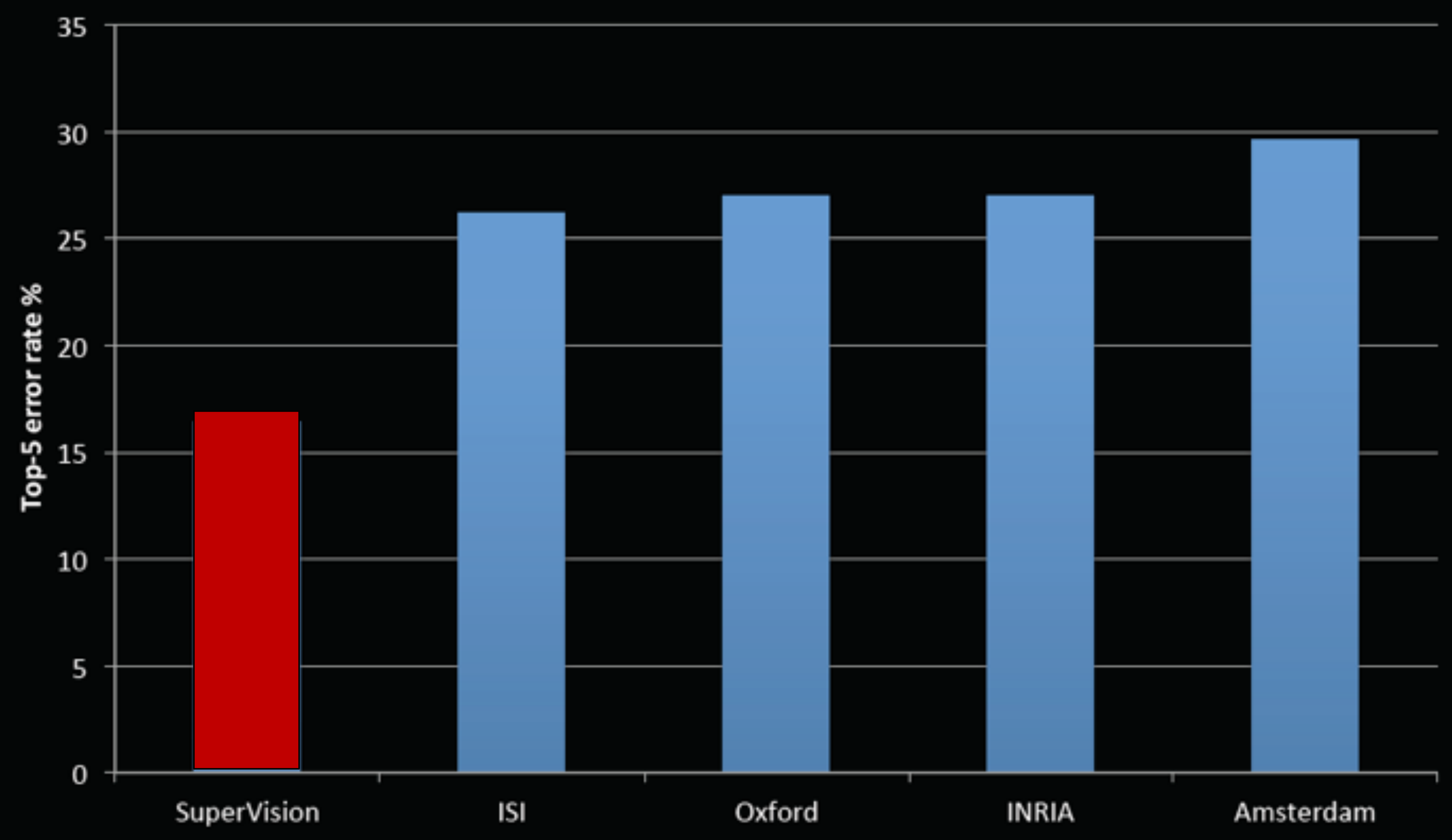
“Alexnet”



Slide from Rob Fergus, NYU

ImageNet Classification 2012

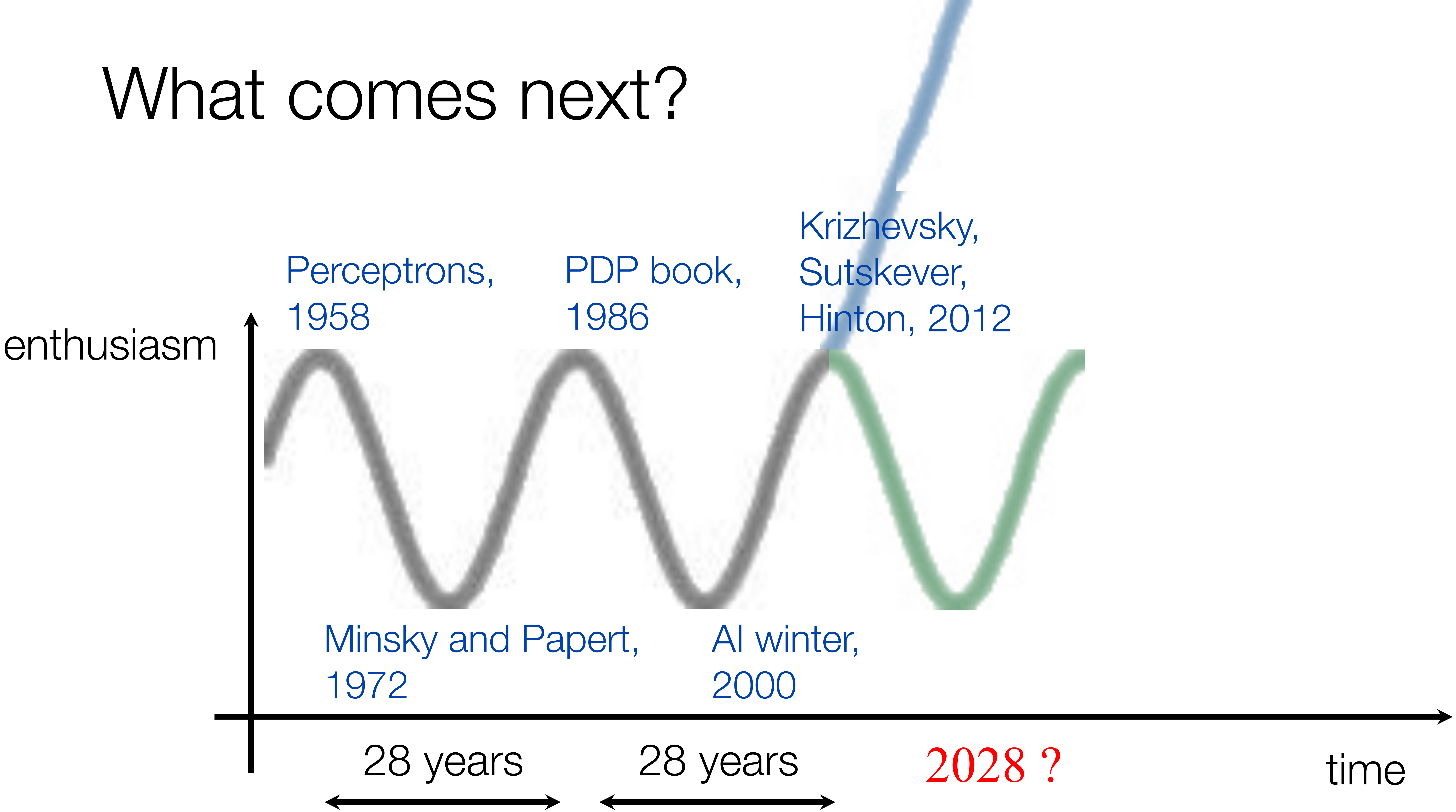
- Krizhevsky et al. -- 16.4% error (top-5)
- Next best (non-convnet) – 26.2% error



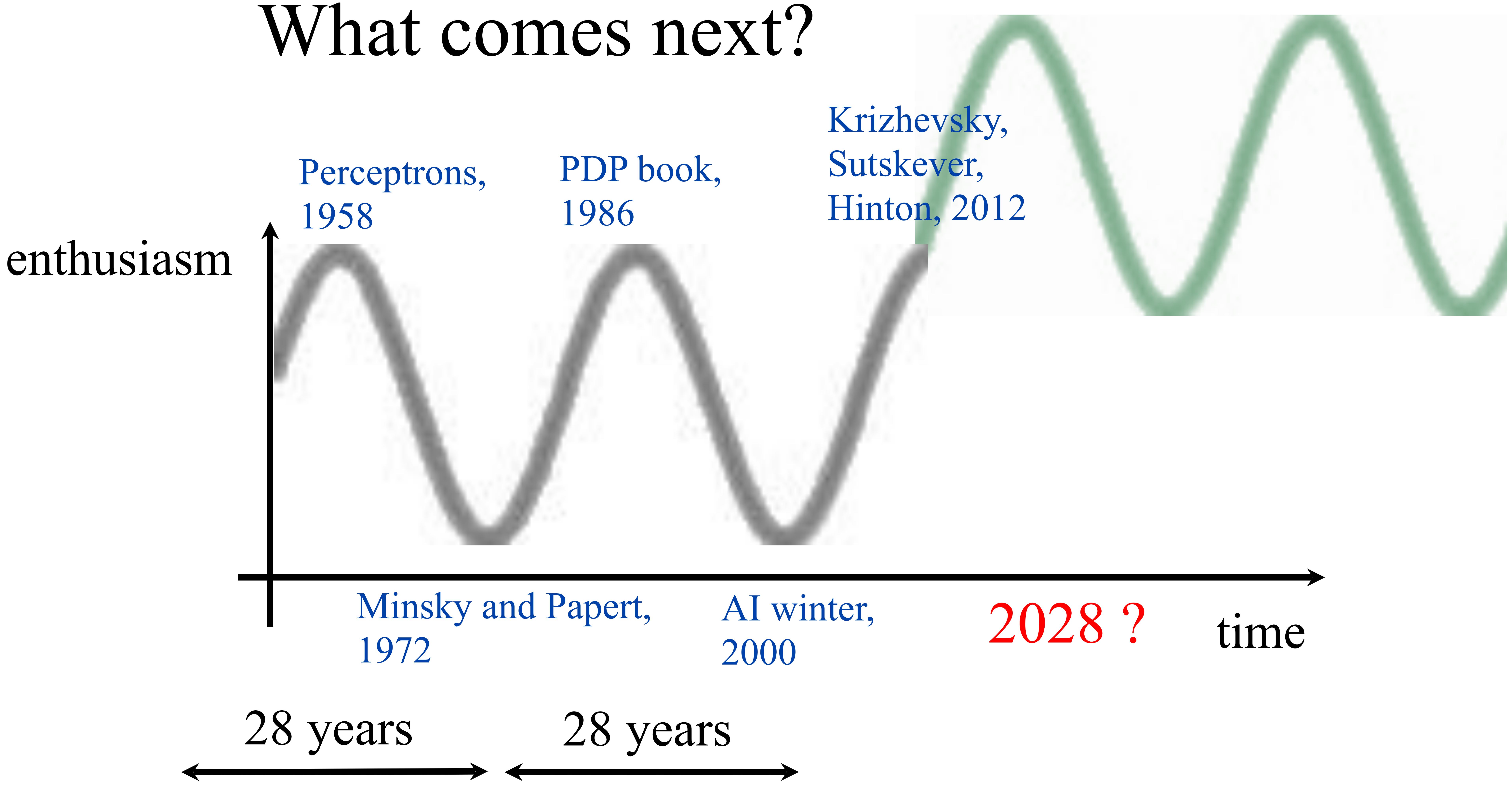
Krizhevsky, Sutskever, and Hinton, NeurIPS 2012

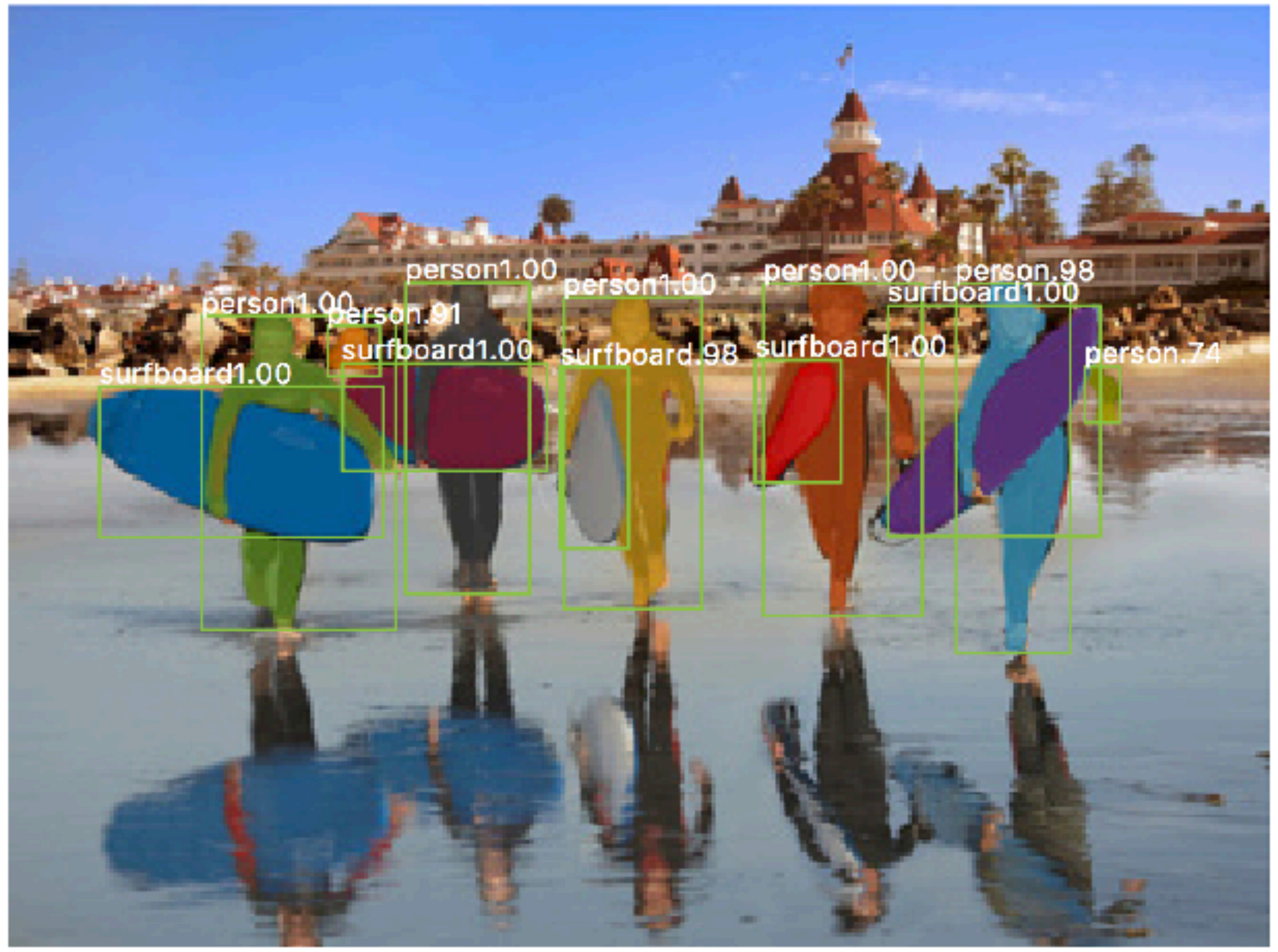

28 years
28 years
Perceptrons, 1958
PDP book, 1986
Krizhevsky, Sutskever, Hinton, 2012
enthusiasm
Minsky and Papert, 1972
AI winter, 2000
time

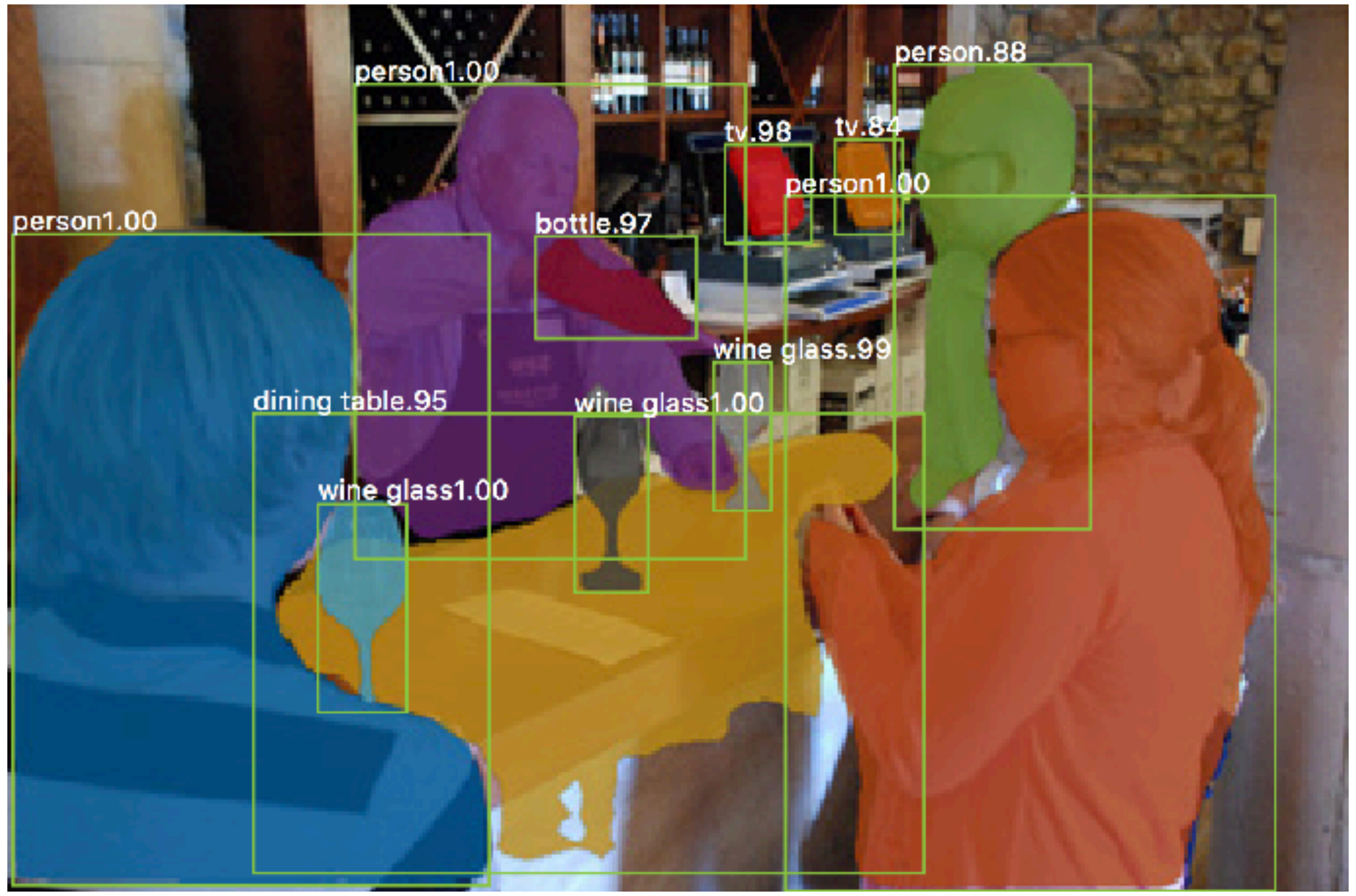
What comes next?



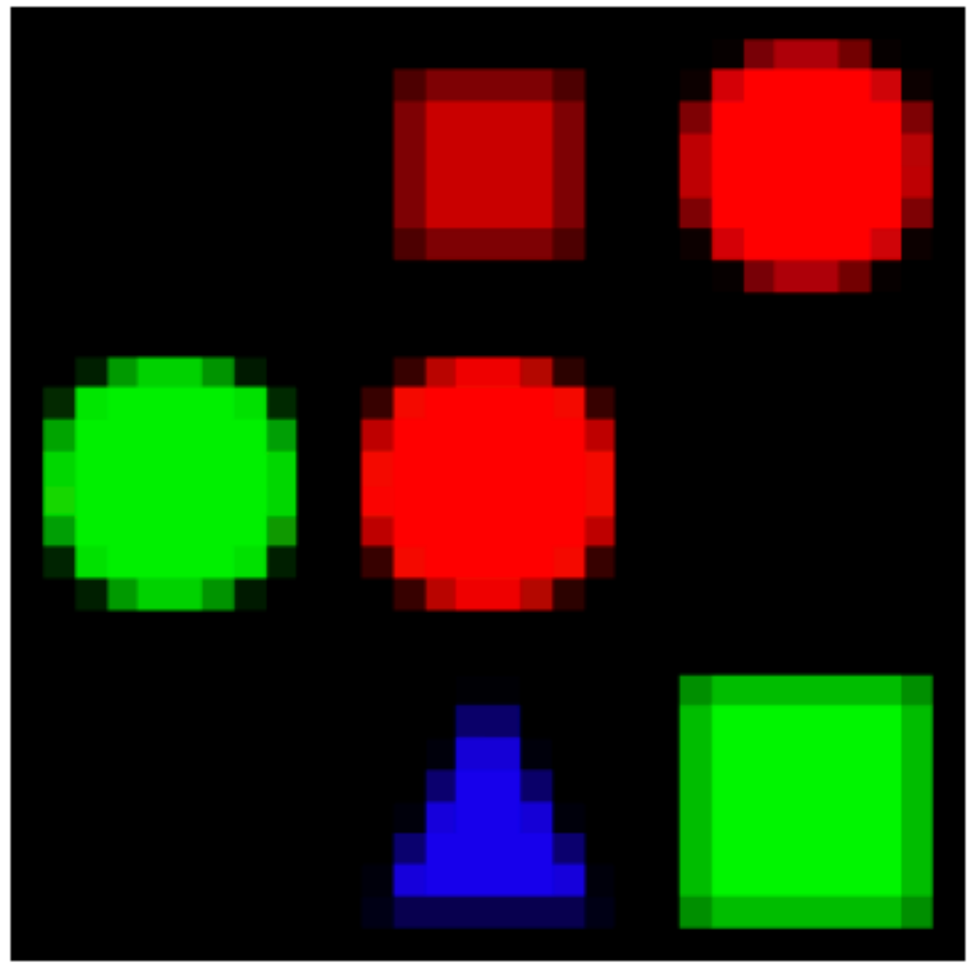
What comes next?







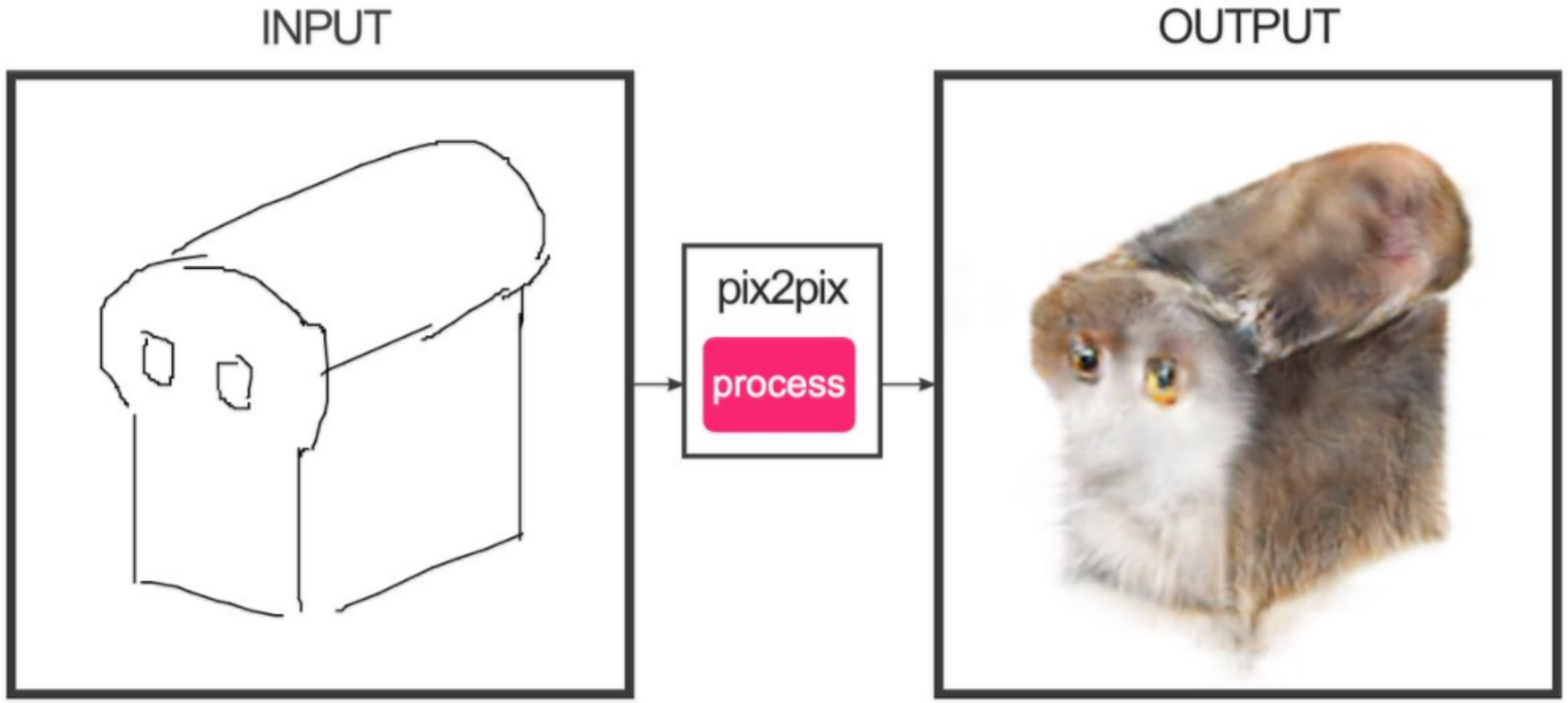
[“Mask RCNN”, He et al. 2017]

 *what color is the vase?*	 *is the bus full of passen-gers?*	 *is there a red shape above a circle?*

`classify[color](`
`  attend[vase])`
	`measure[is](`
	`  combine[and](`
	`    attend[bus],`
	`    attend[full])`
		`measure[is](`
		`  combine[and](`
		`   attend[red],`
		`   re-attend[above](`
		`    attend[circle])))`

green (green)	yes (yes)	no (no)

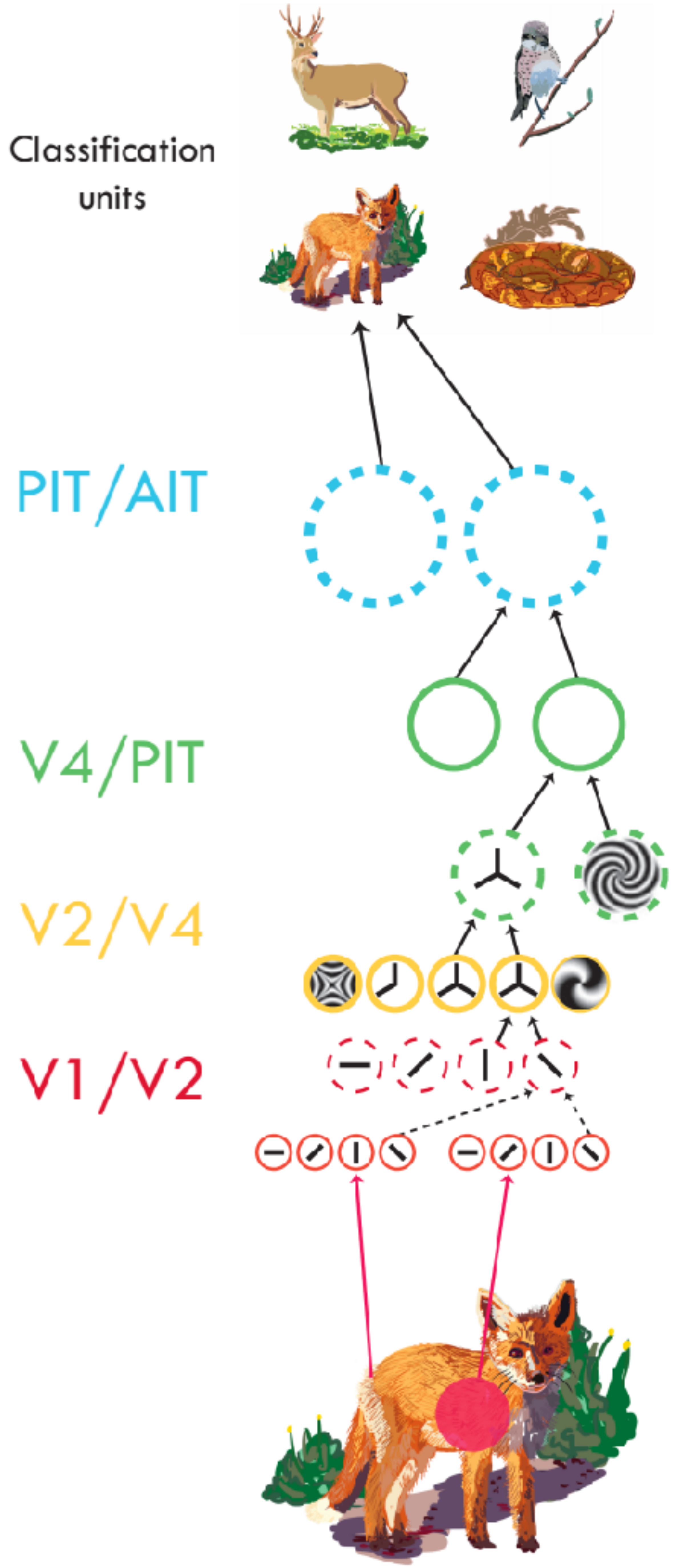
[“Neural module networks”, Andreas et al. 2017]



Ivy Tasi @ivymyt

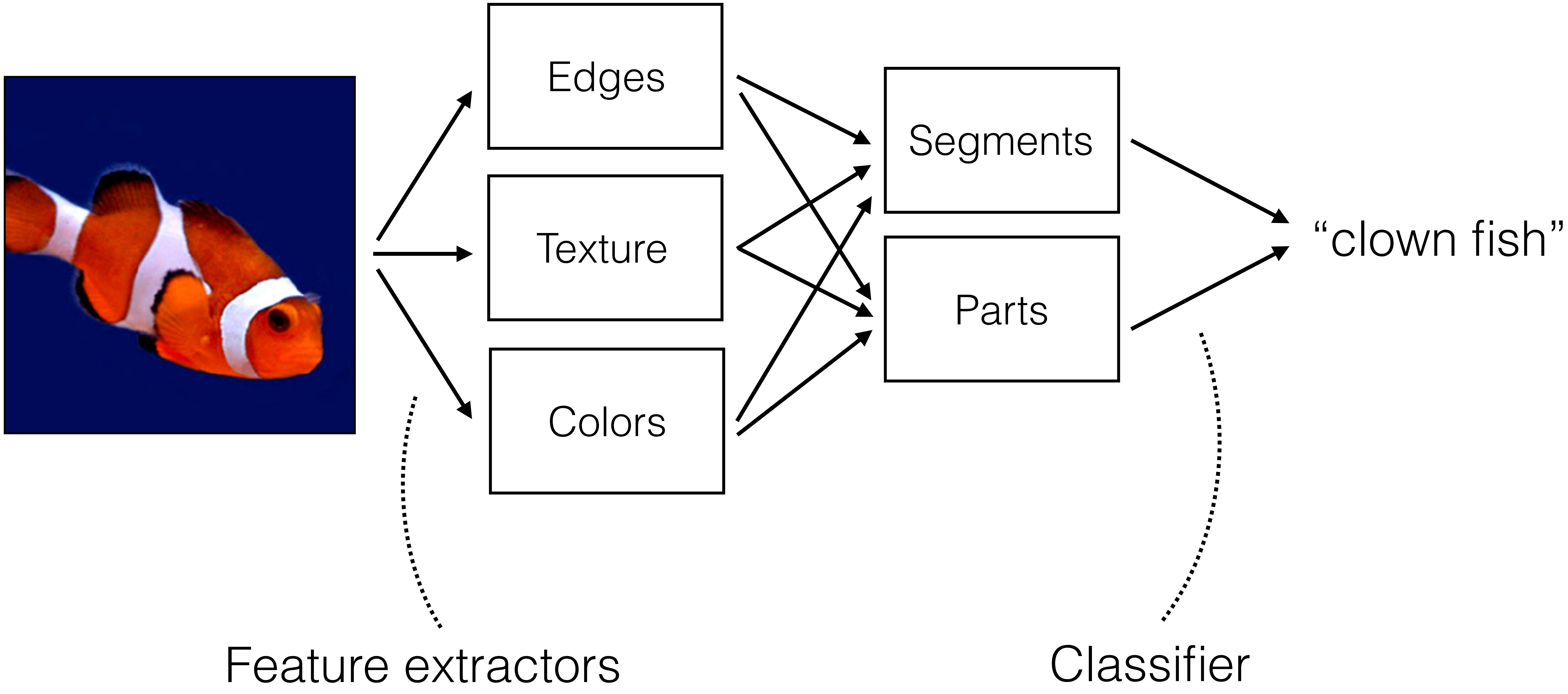
Vitaly Vidmirov @vvid

["pix2pix", Isola et al. 2017]

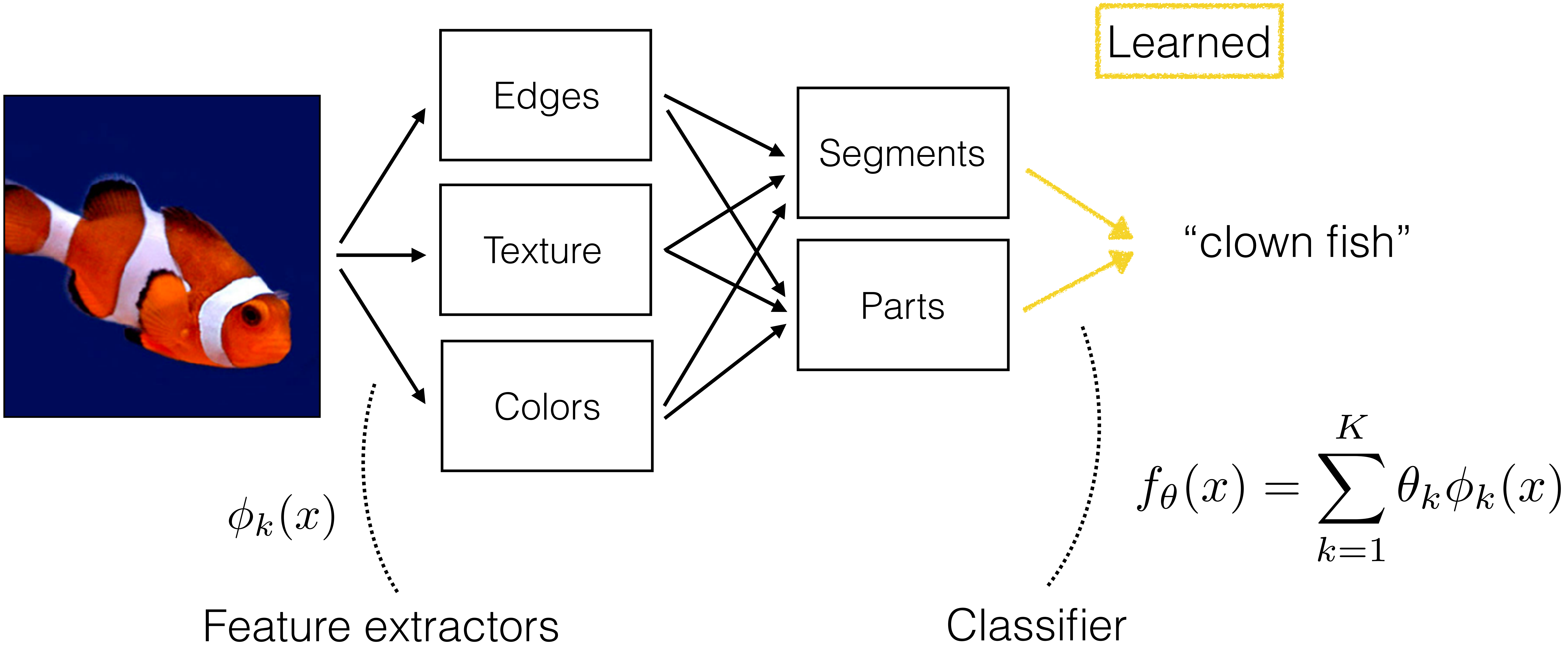


Serre, 2014

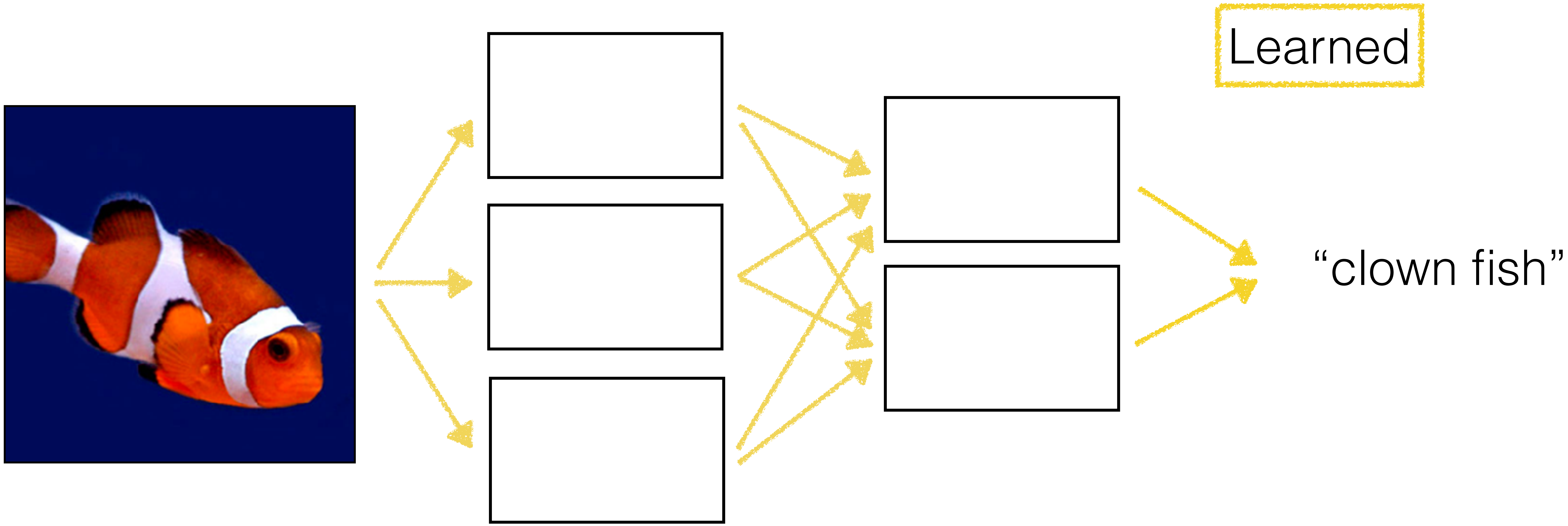
Object recognition



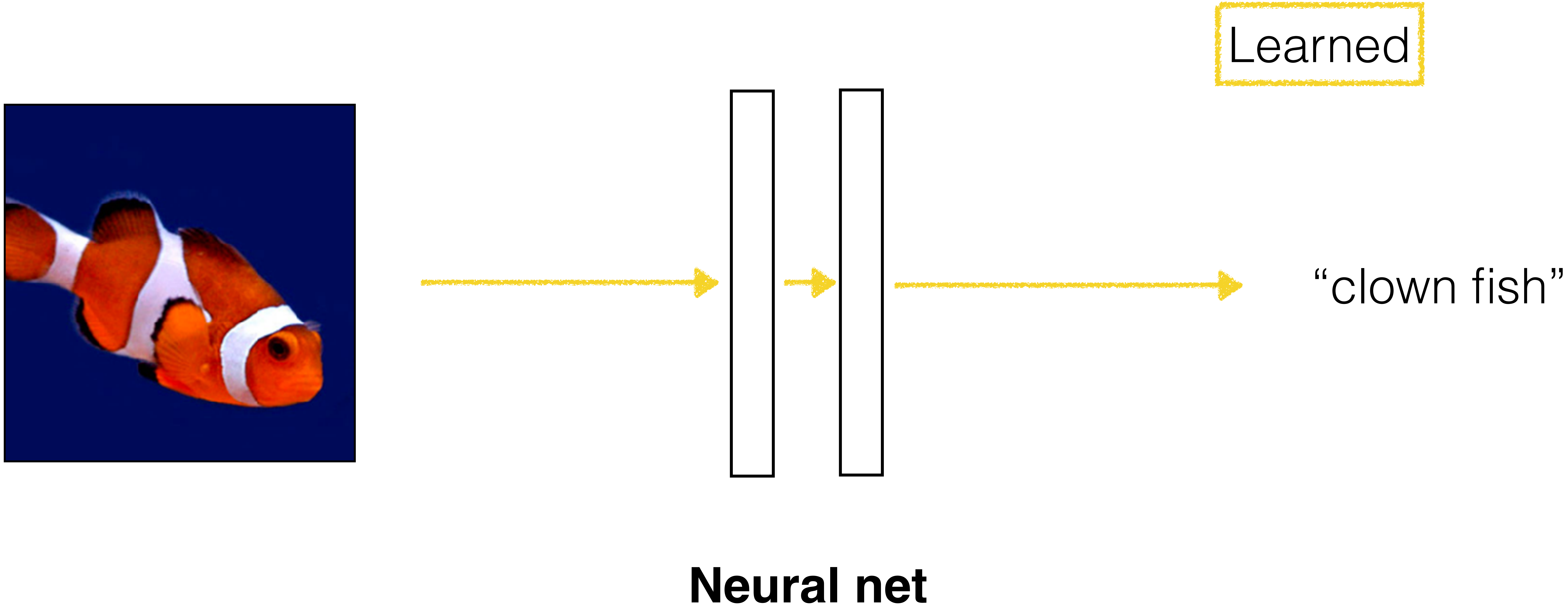
Object recognition



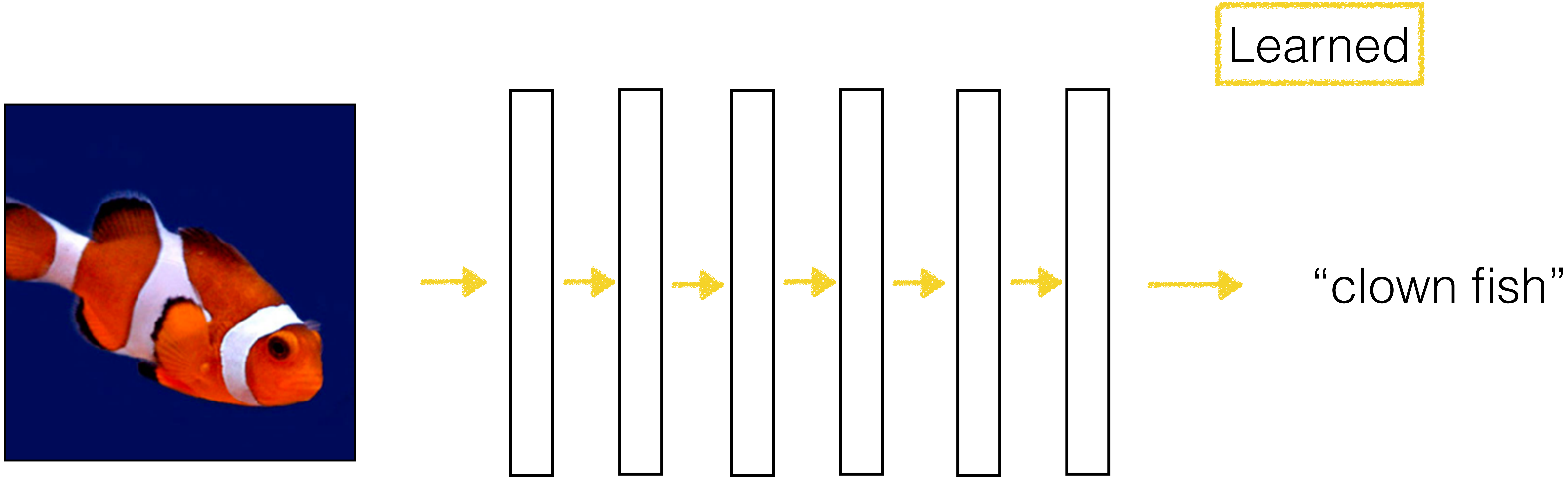
Object recognition



Object recognition

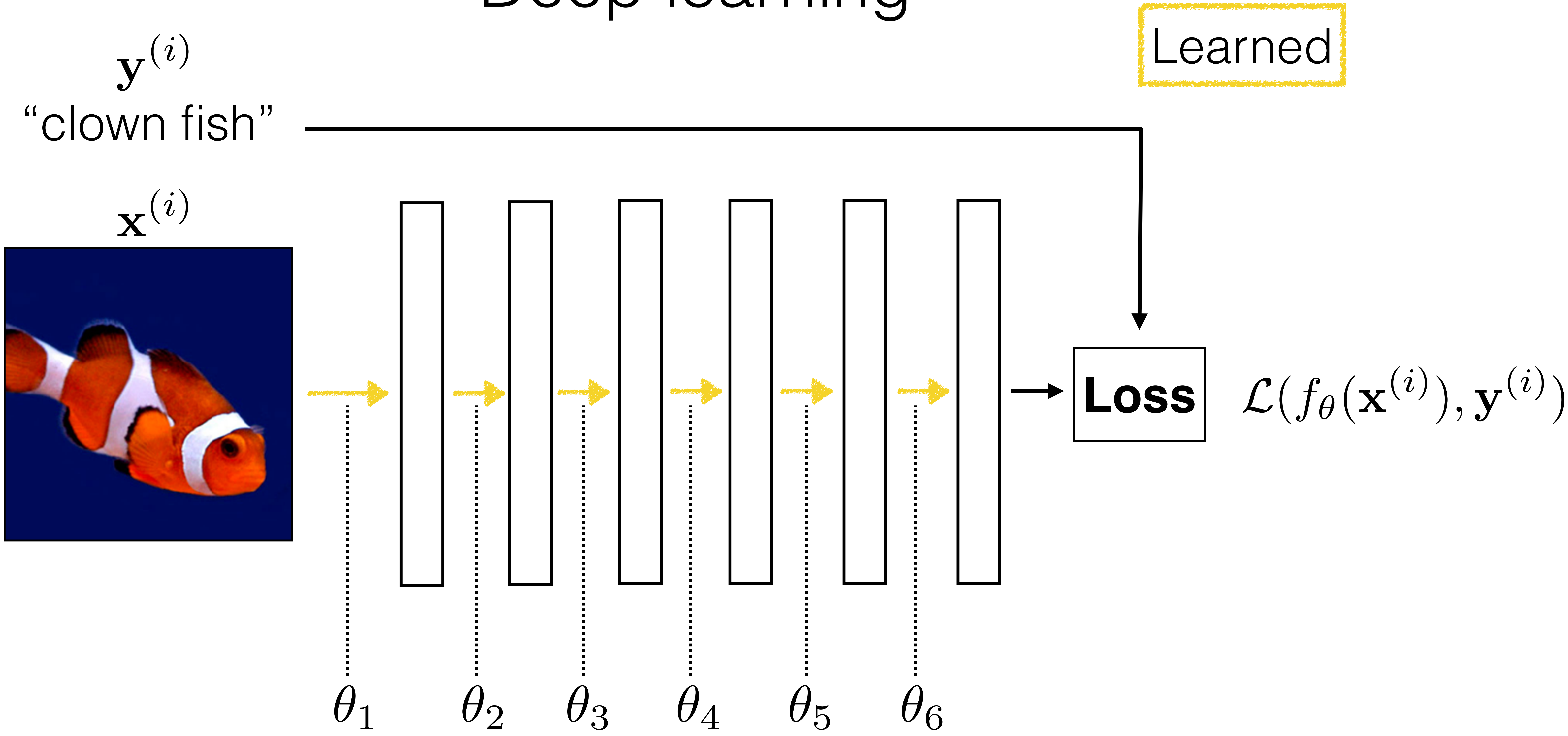


Object recognition



Deep neural net

Deep learning

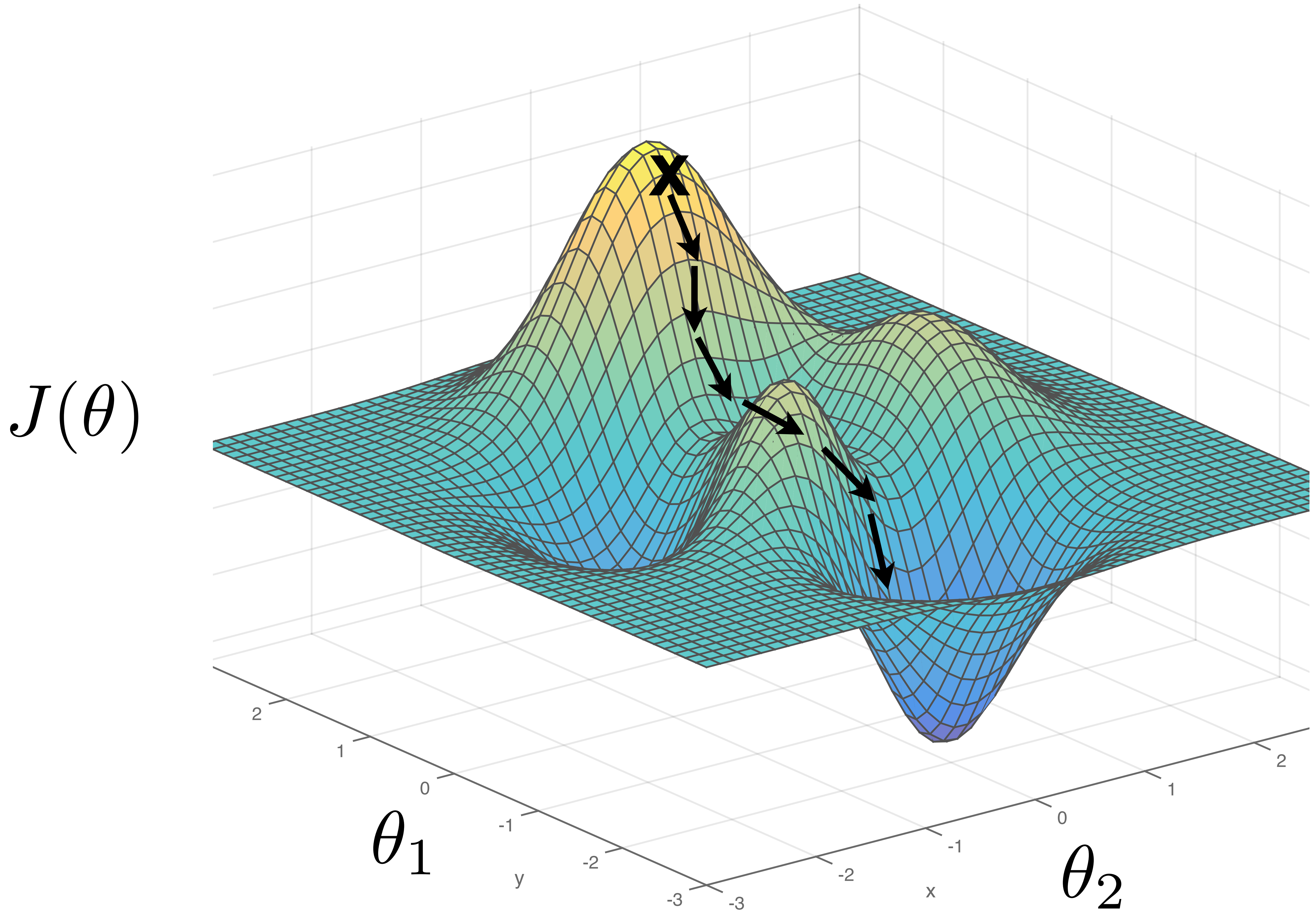


$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$$

Gradient descent

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

Gradient descent



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

Gradient descent

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

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)

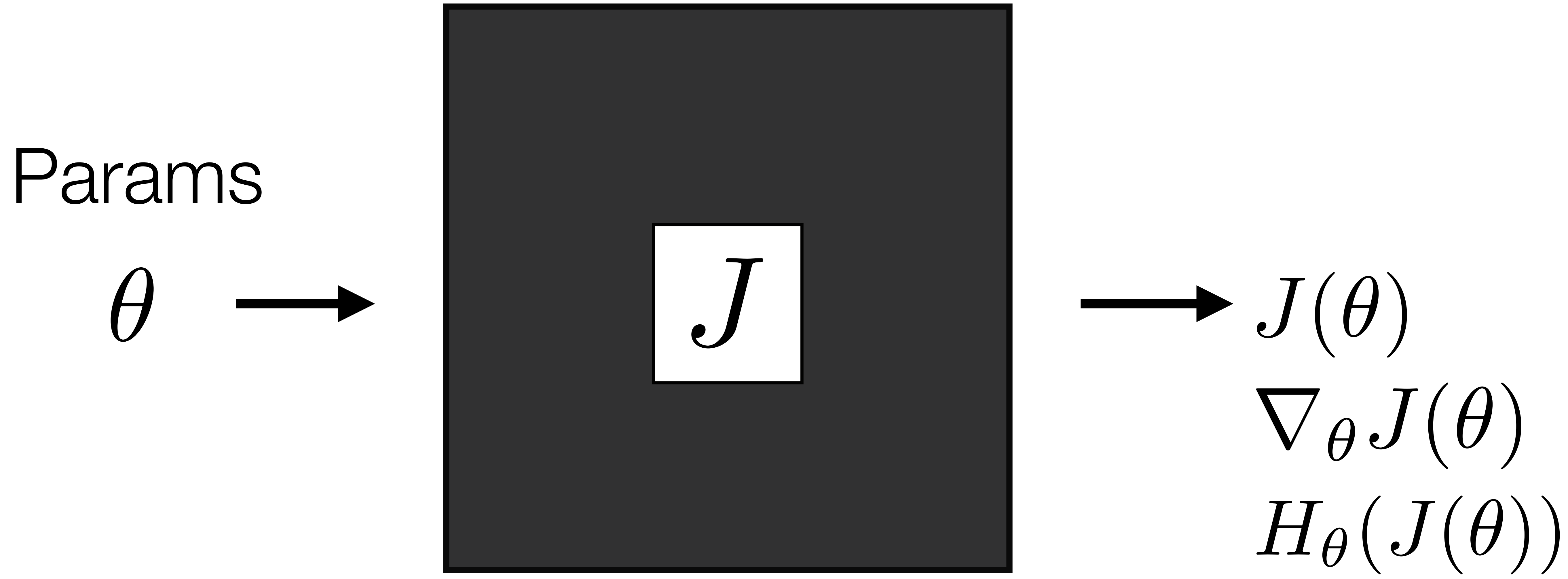
- Want to minimize overall loss function **J**, which is sum of individual losses over each example.
- In Stochastic gradient descent, compute gradient on sub-set (batch) of data.

  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

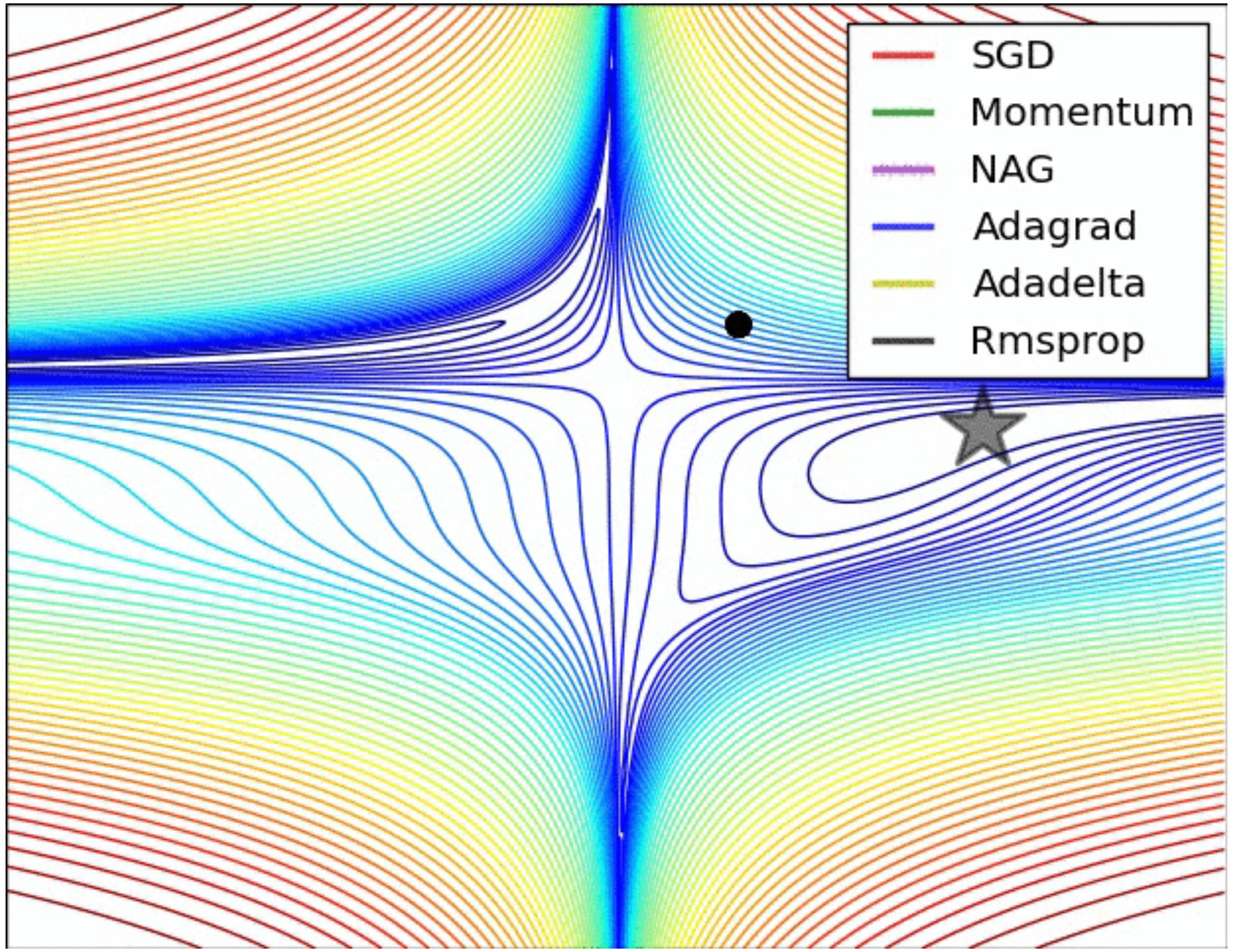
Optimization



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

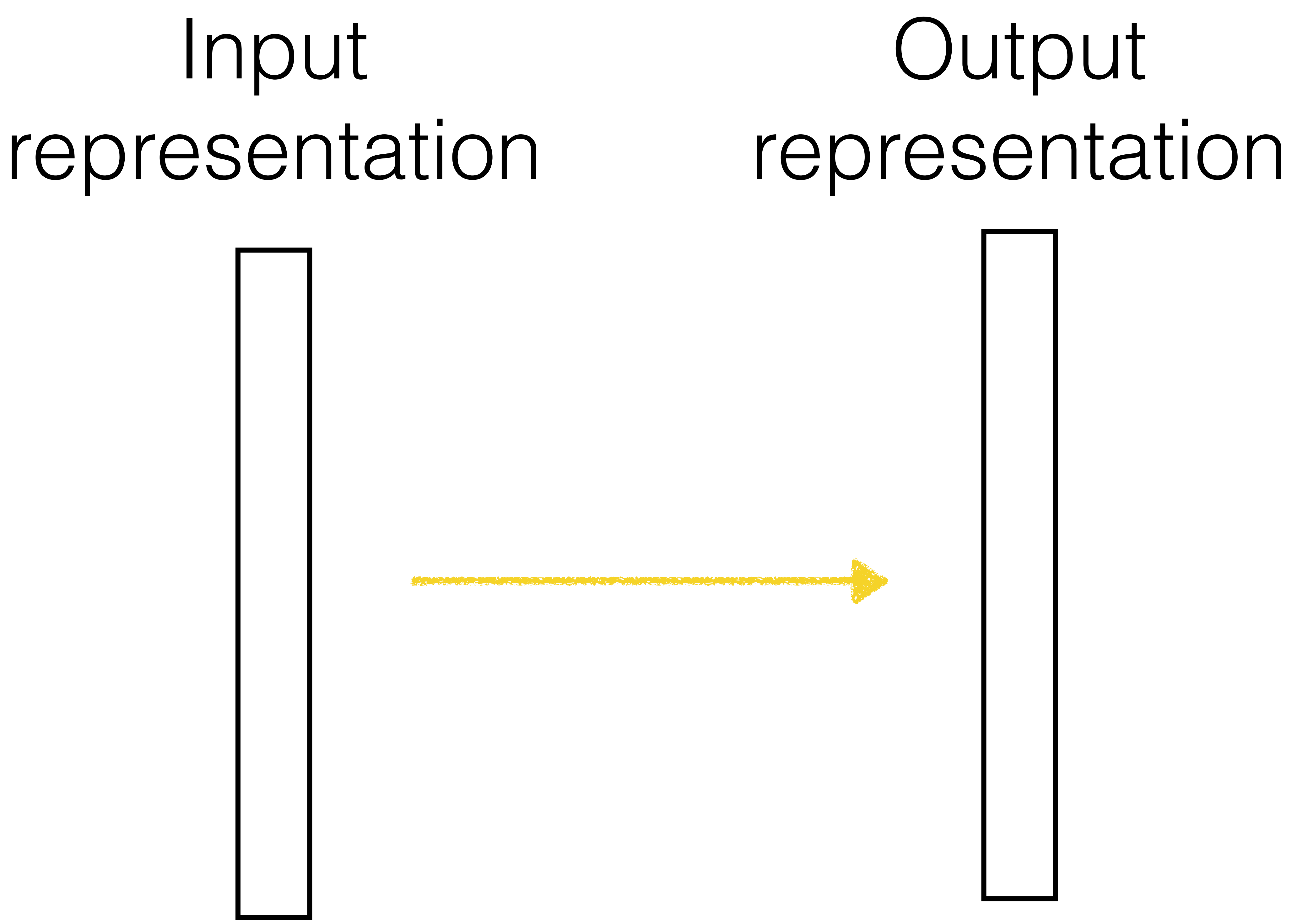
- What's the knowledge we have about J?
  - We can evaluate $J(\theta)$ ← Black box optimization
  - We can evaluate $J(\theta)$ and $\nabla_\theta J(\theta)$ (**Gradient**) ← First order optimization
  - We can evaluate $J(\theta)$, $\nabla_\theta J(\theta)$, and $H_\theta(J(\theta))$ (**Hessian**) ← Second order optimization

Comparison of gradient descent variants



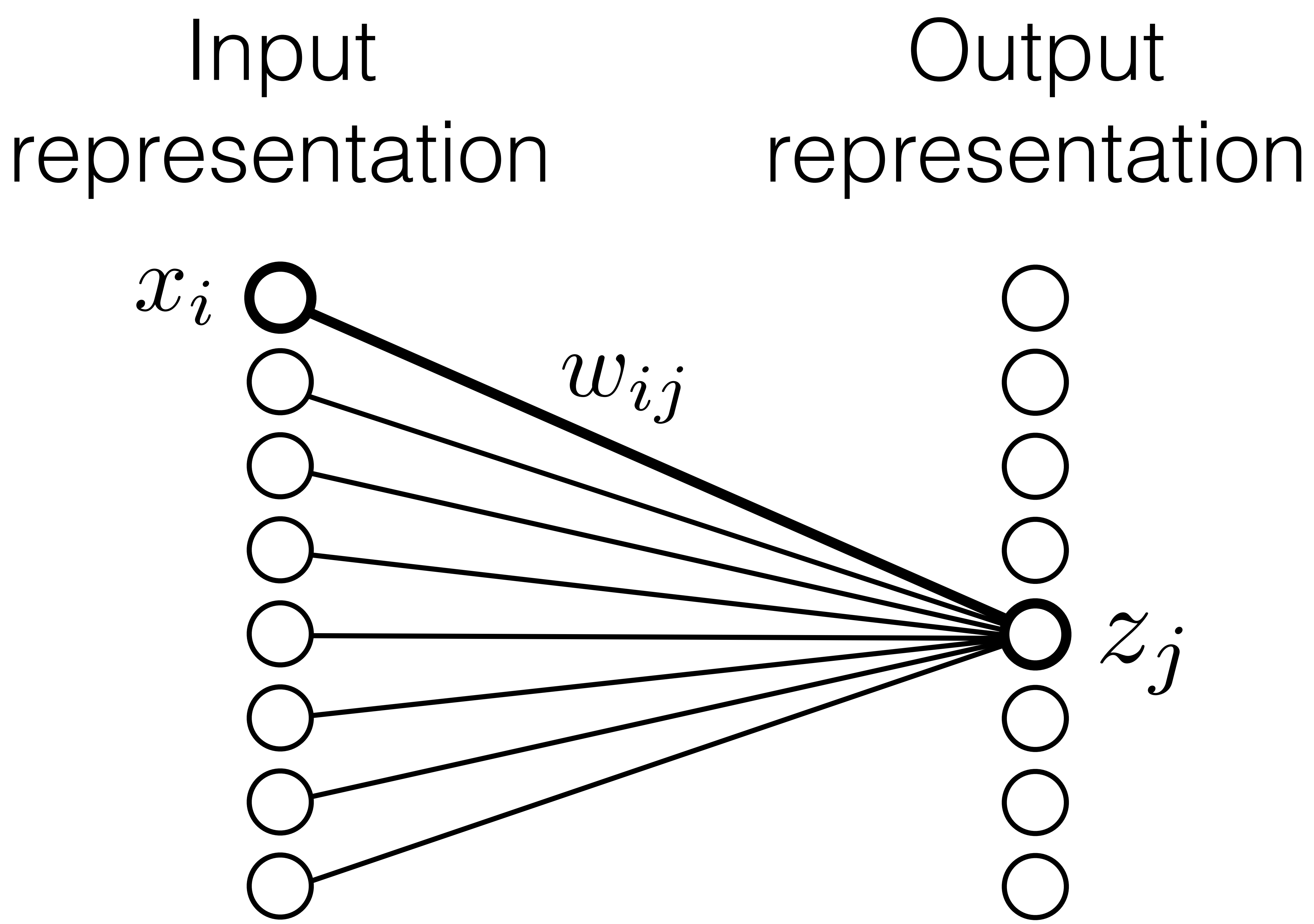
[http://ruder.io/optimizing-gradient-descent/]

Computation in a neural net



Computation in a neural net

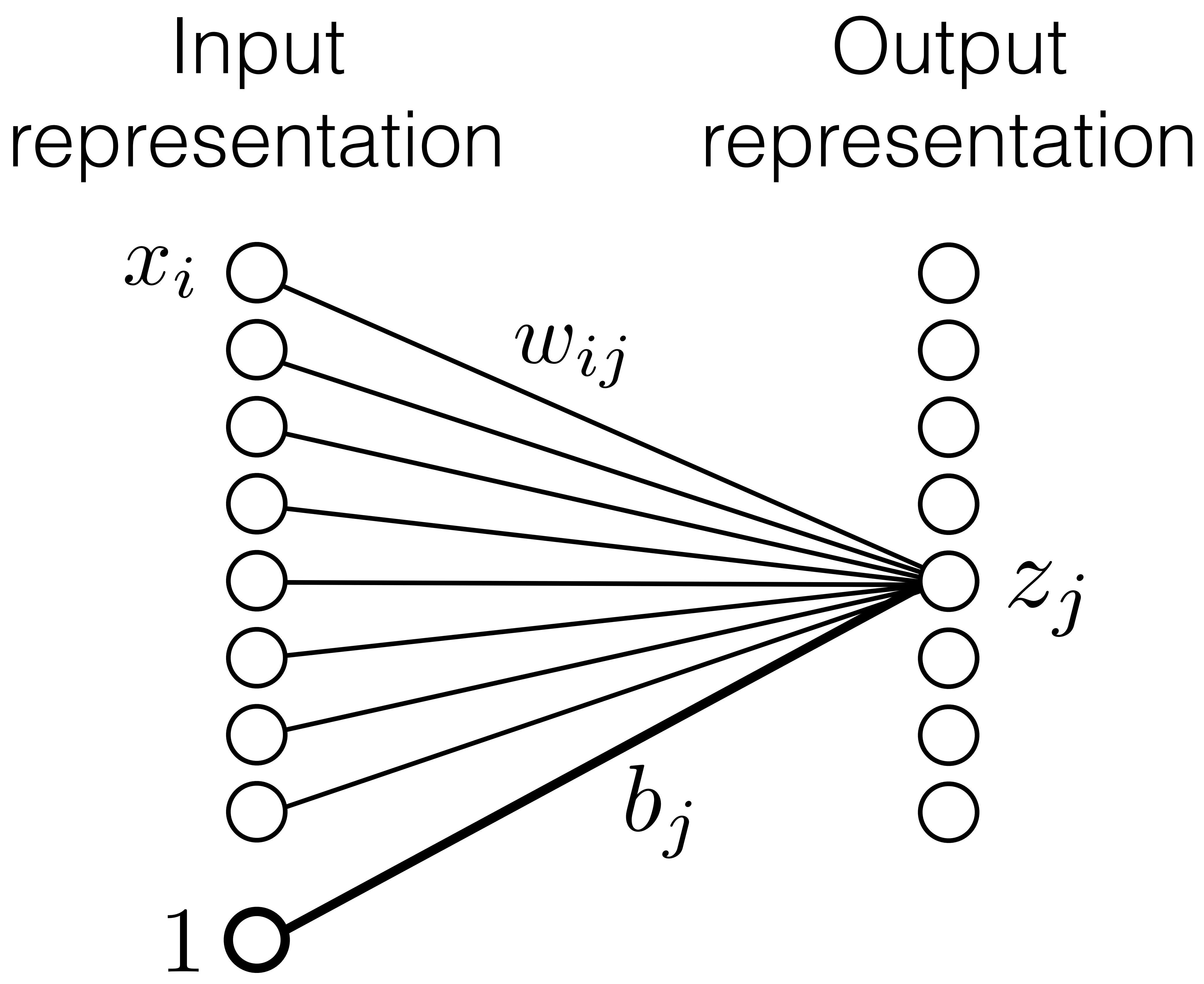
Linear layer



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

Computation in a neural net

Linear layer



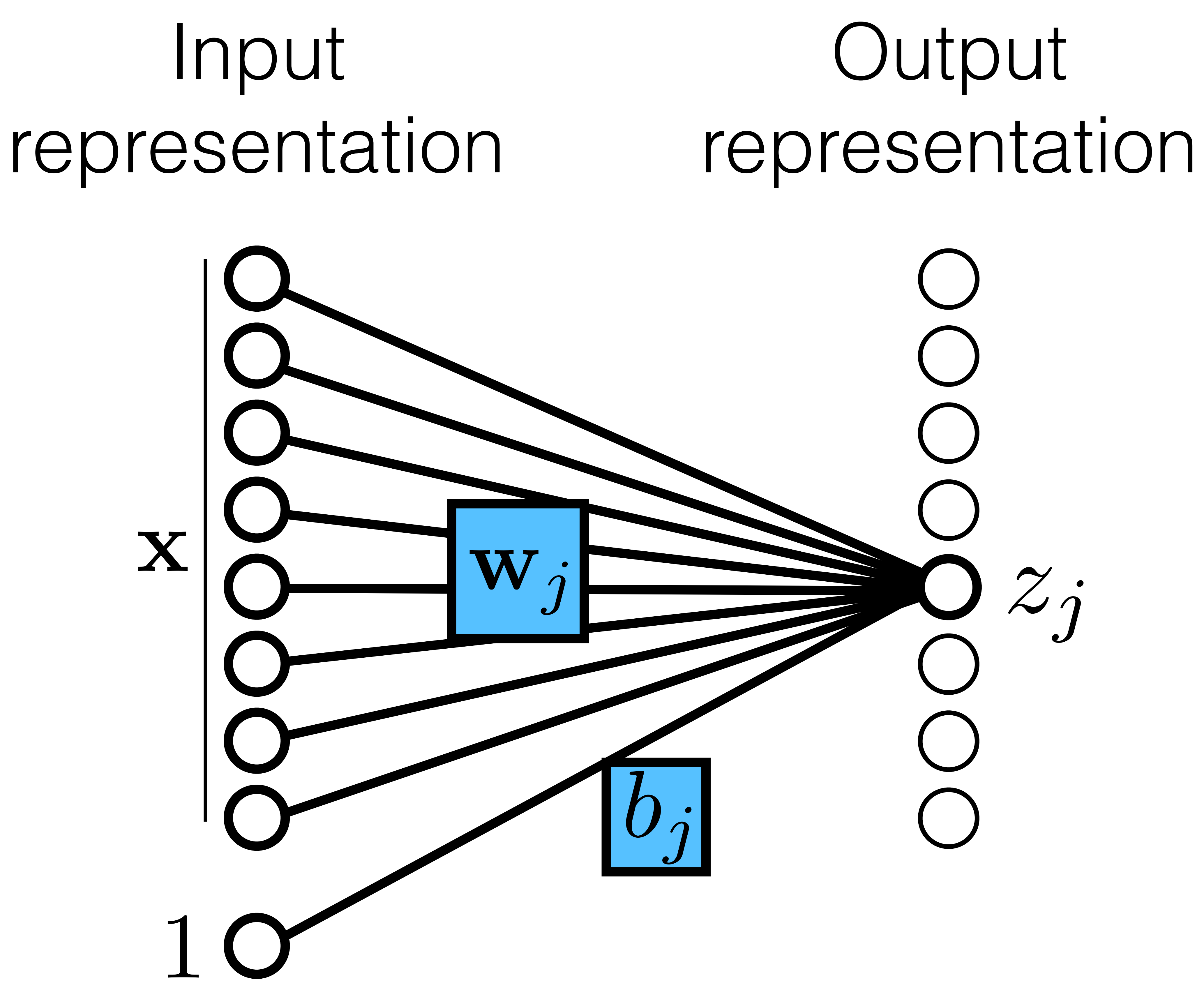
$$z_j = \sum_i w_{ij} x_i + b_j$$

weights

bias

Computation in a neural net

Linear layer



weights

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

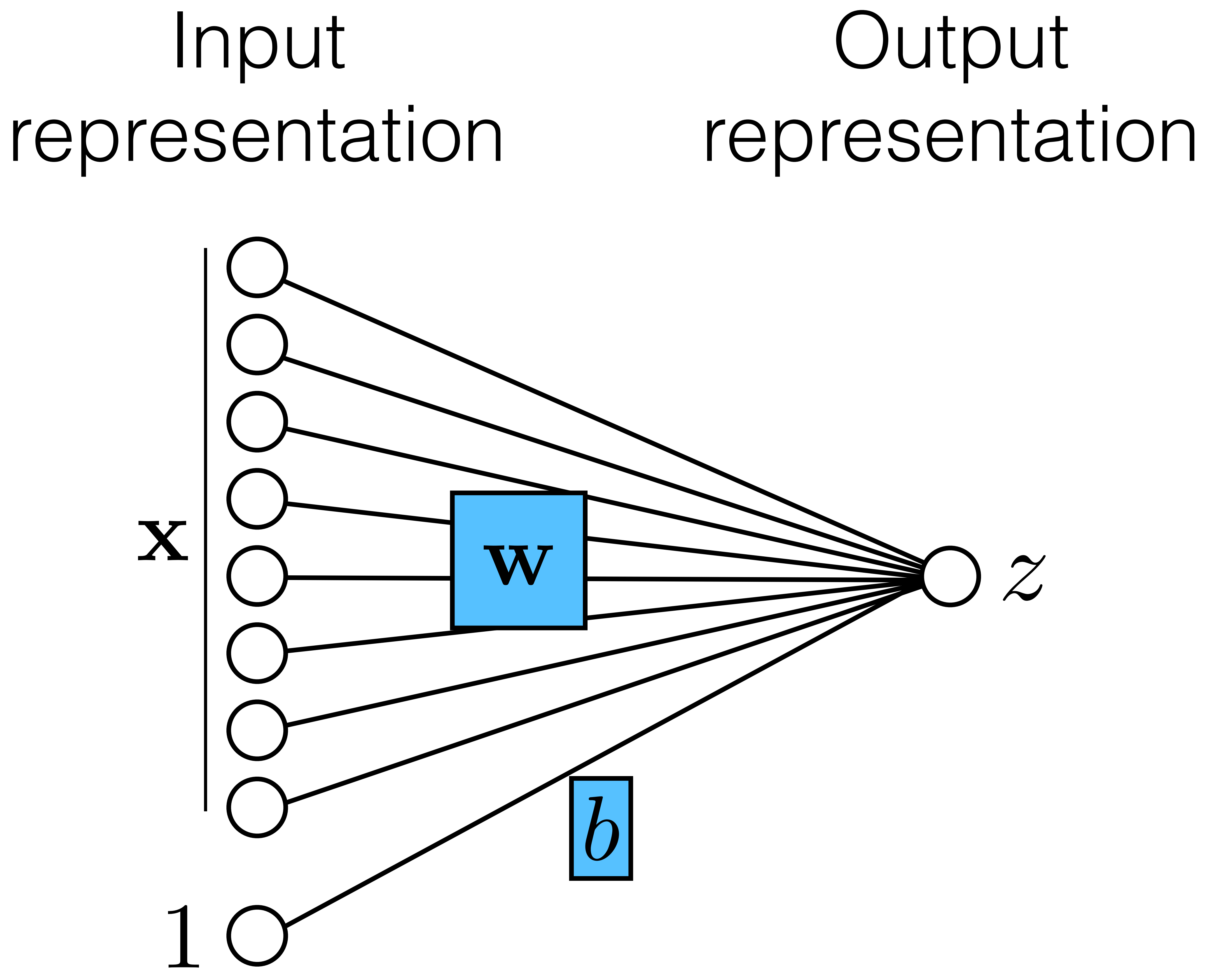
bias

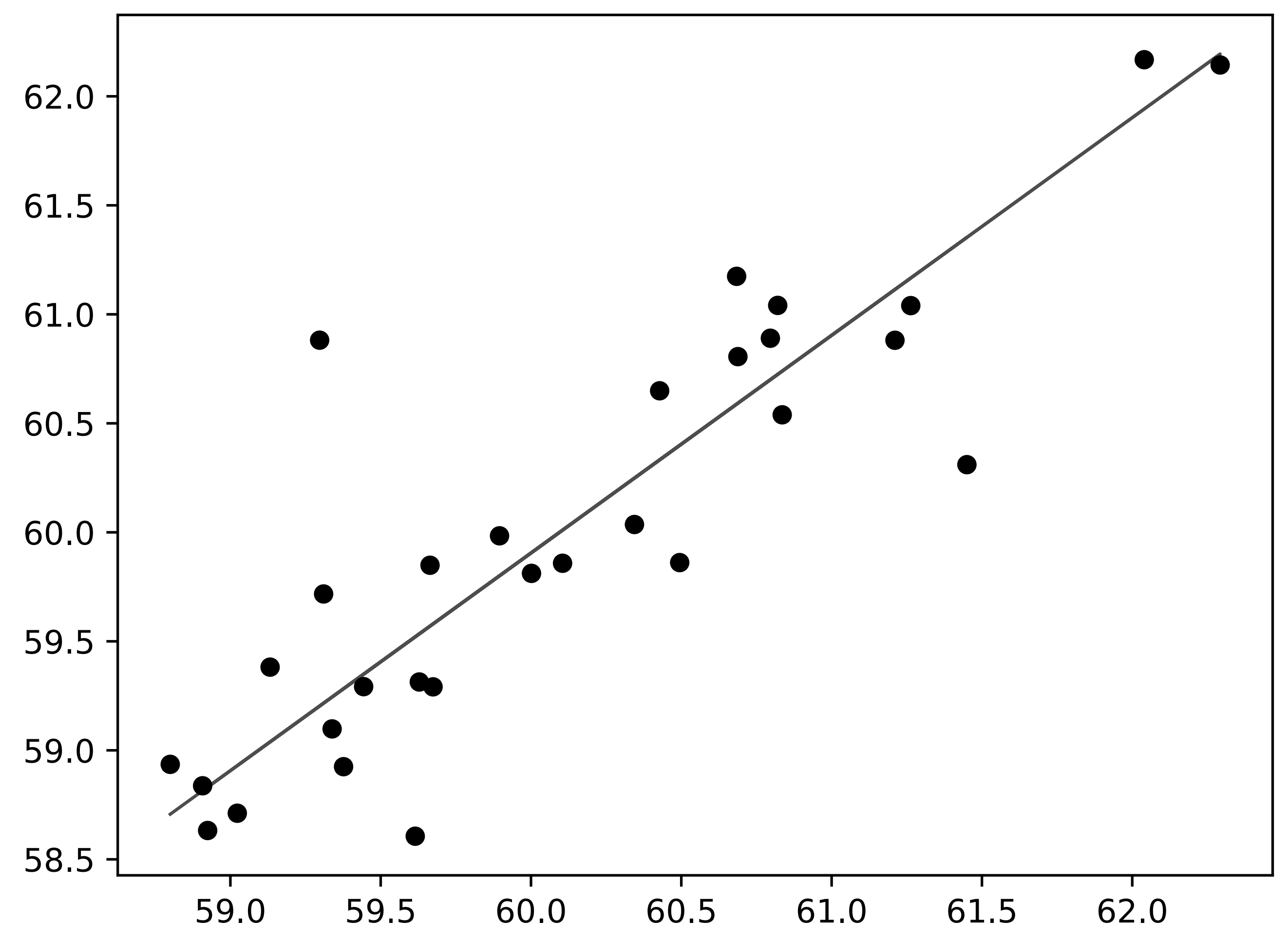
$$\theta = \{\mathbf{W}, \mathbf{b}\}$$

parameters of the model

Example: linear regression with a neural net

Linear layer





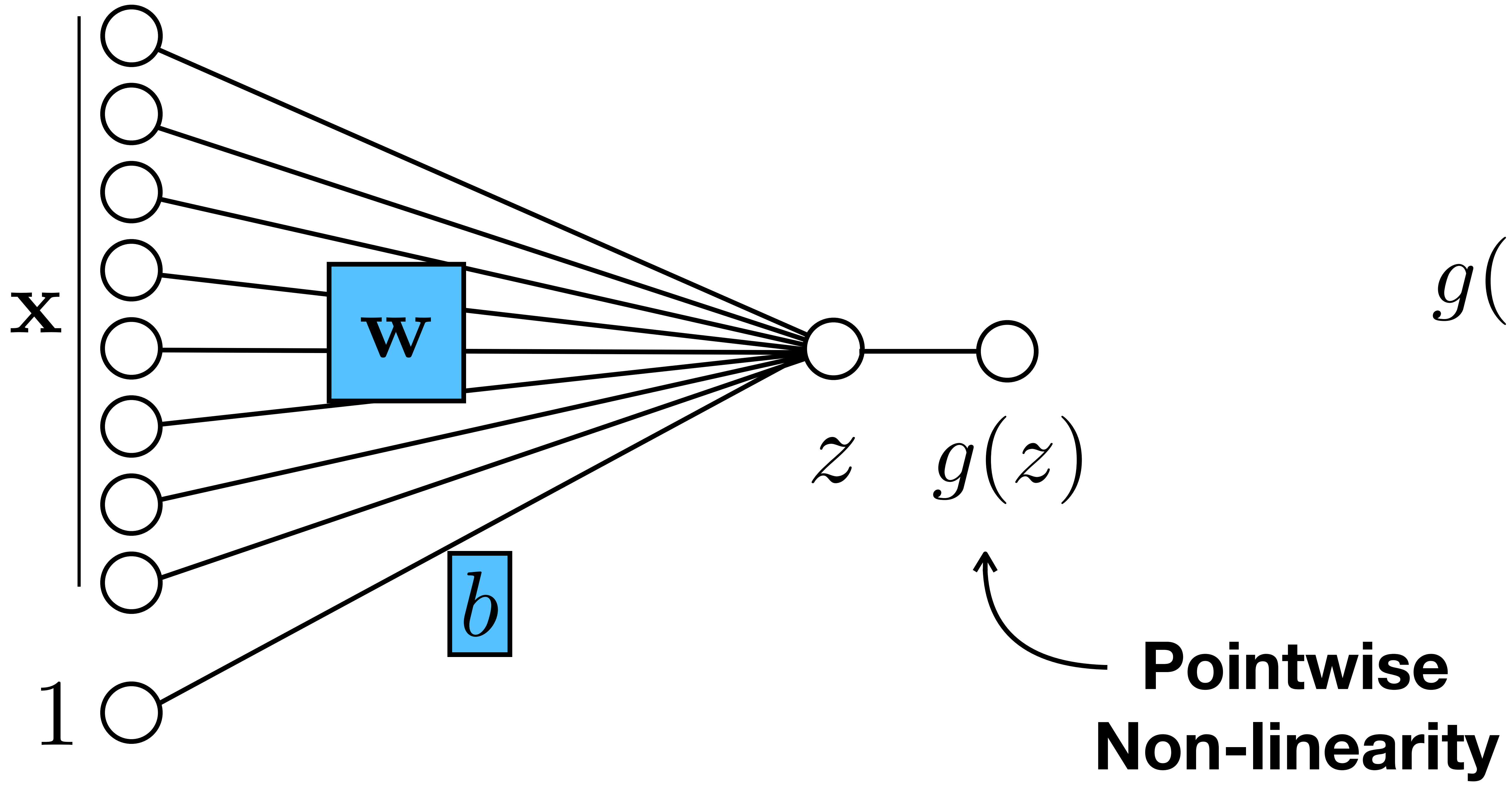
$$f_{\mathbf{w},b}(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + b$$

Computation in a neural net

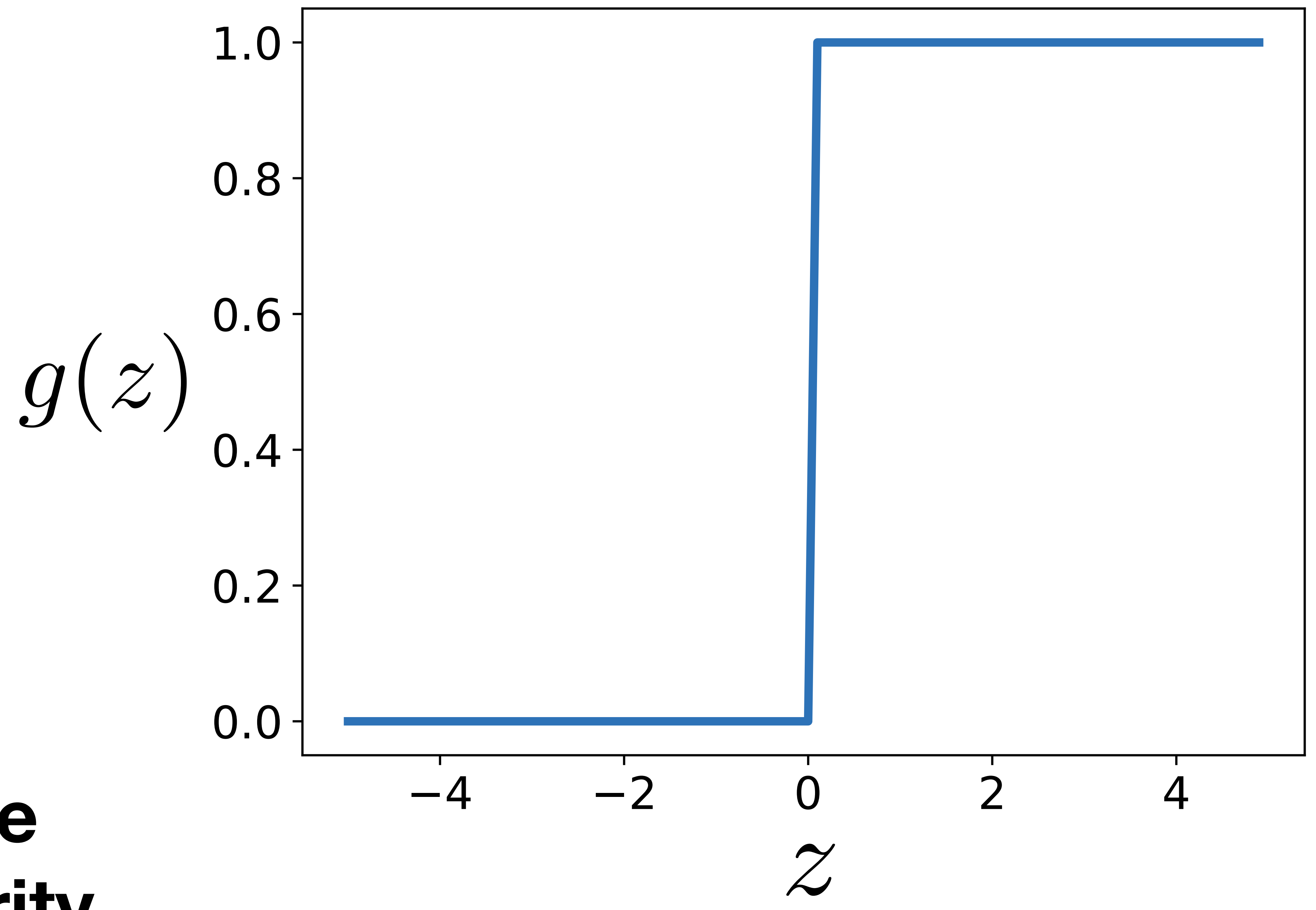
"Perceptron"

Input representation

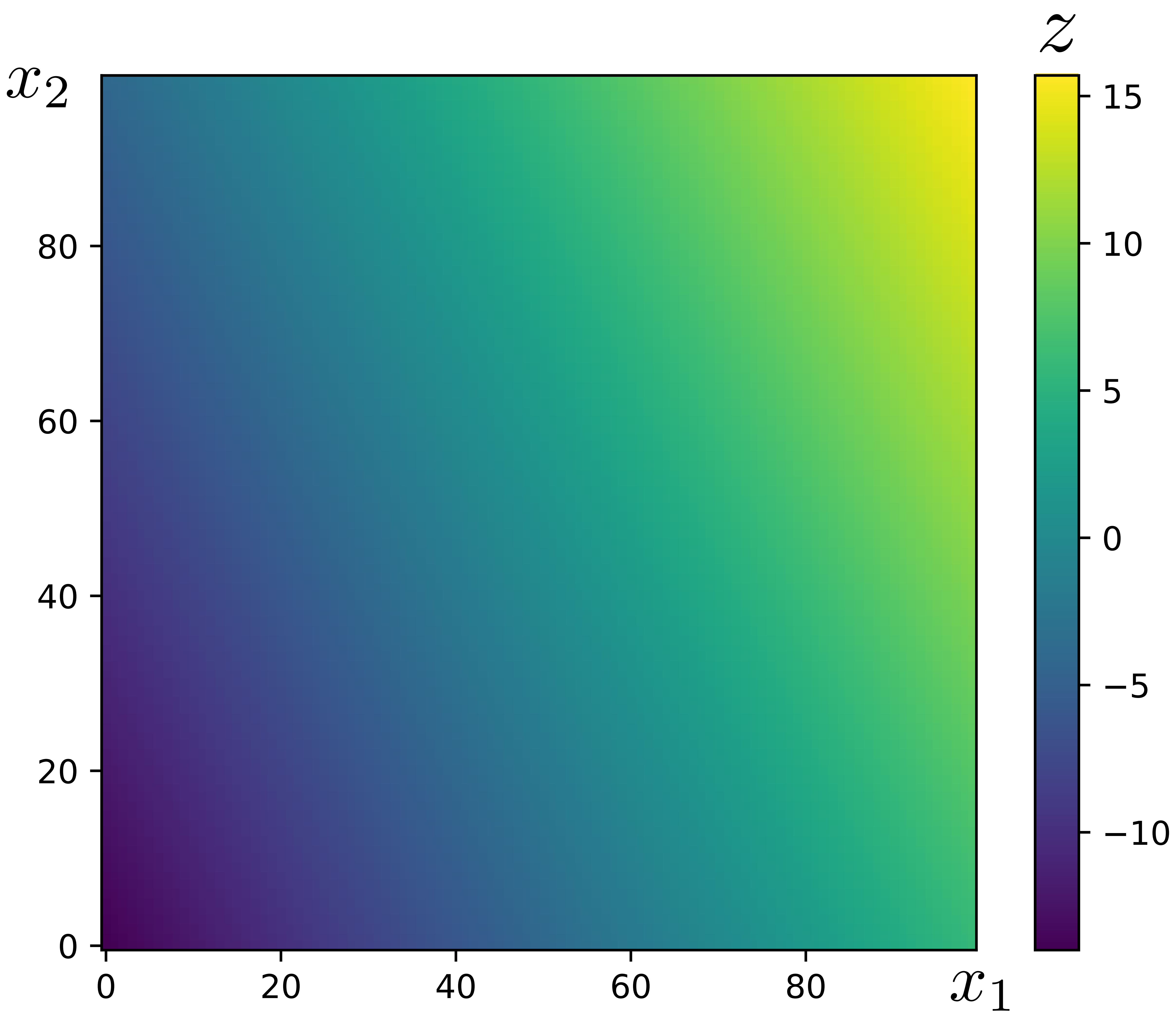
Output representation



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

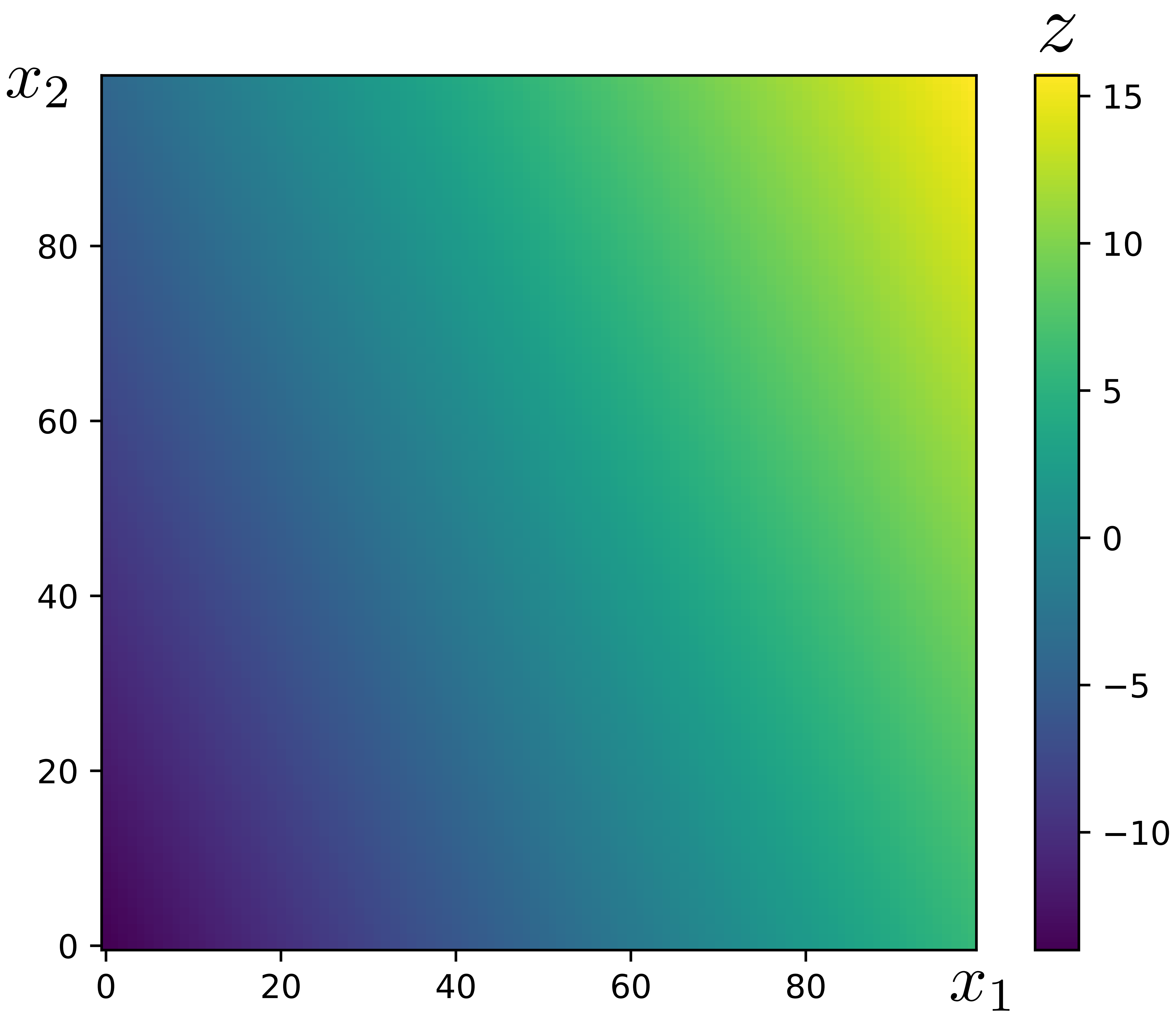


Example: linear classification with a perceptron



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

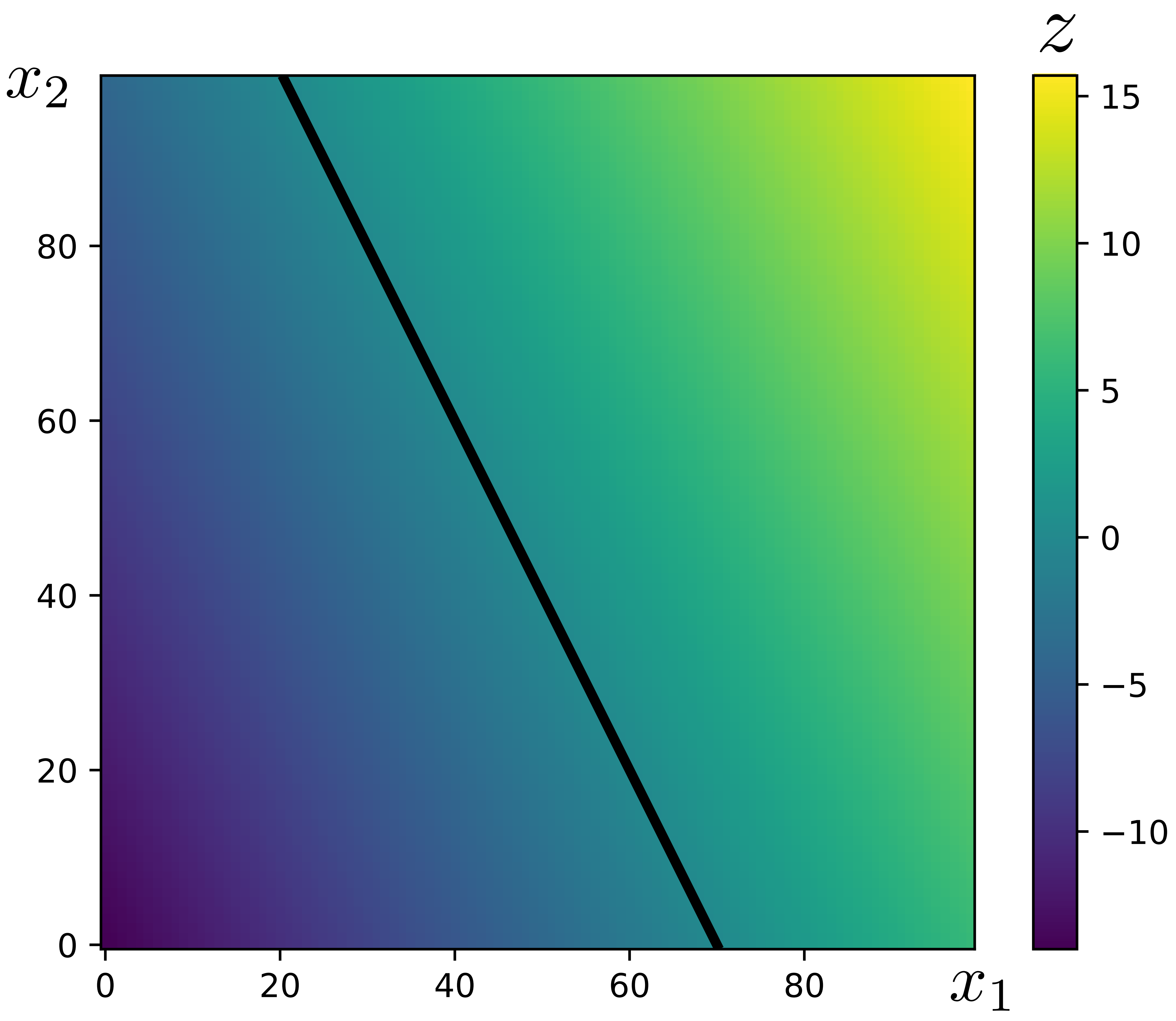
Example: linear classification with a perceptron



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

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

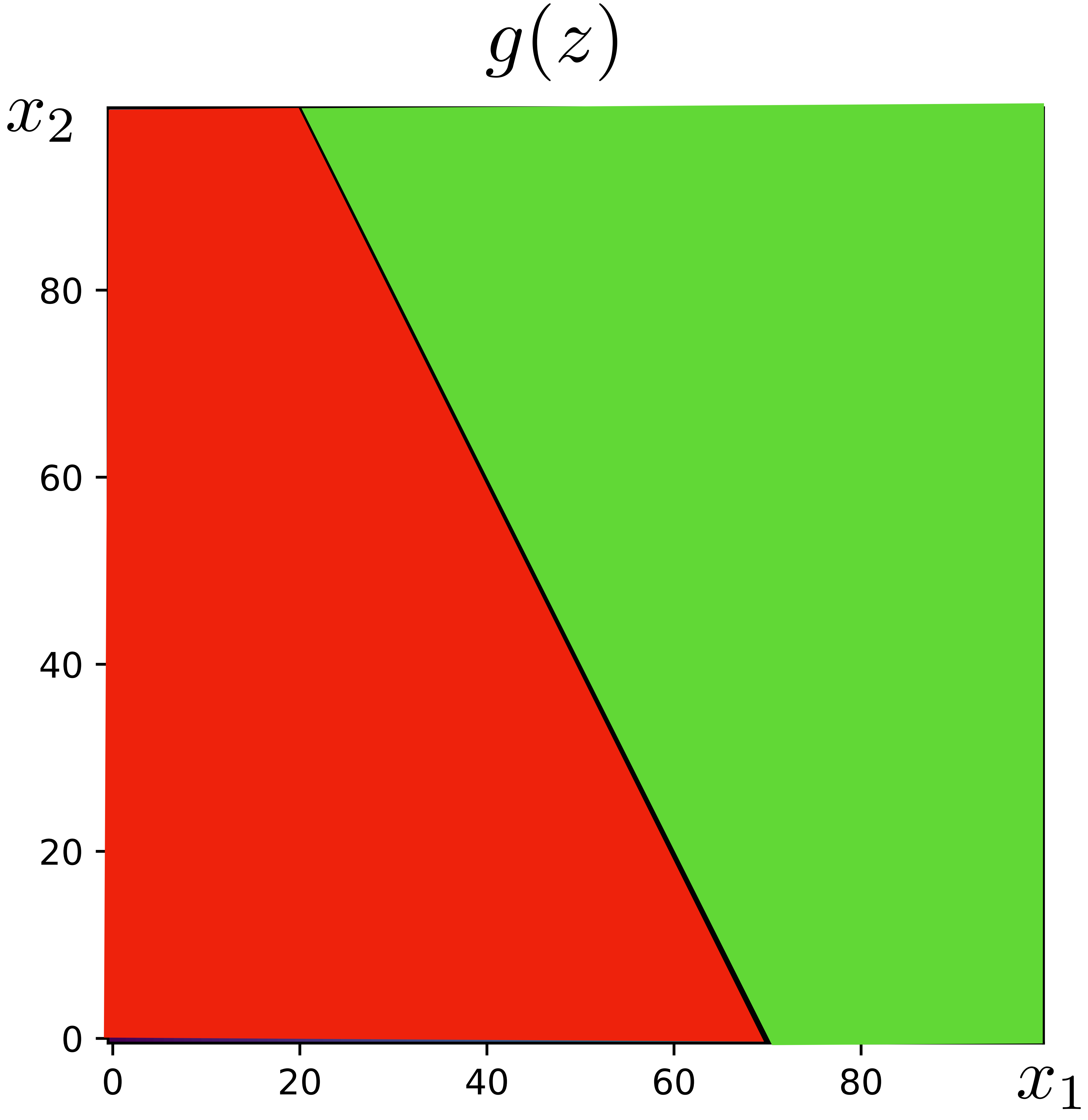
Example: linear classification with a perceptron



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

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

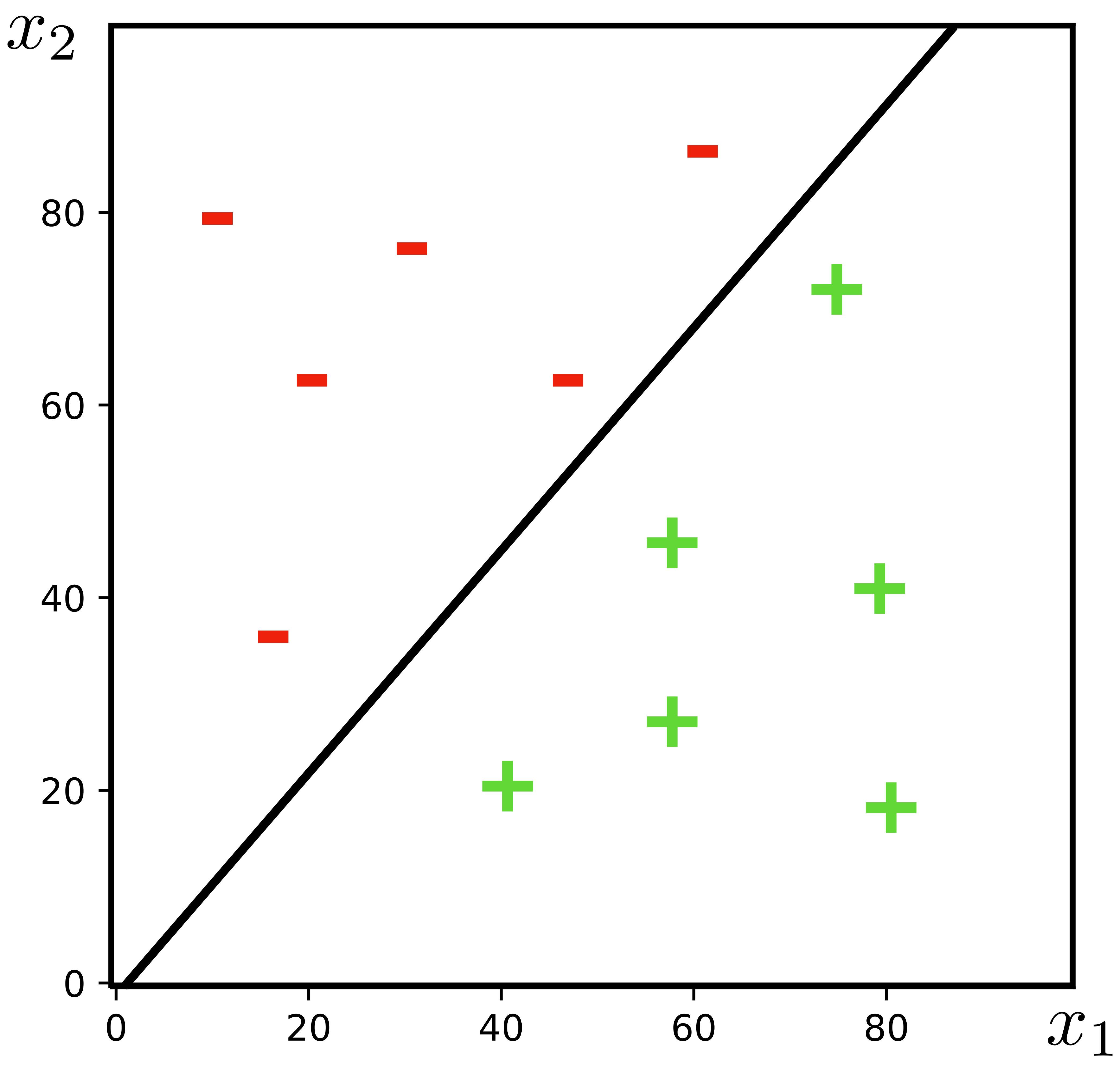
Example: linear classification with a perceptron



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

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

Example: linear classification with a perceptron

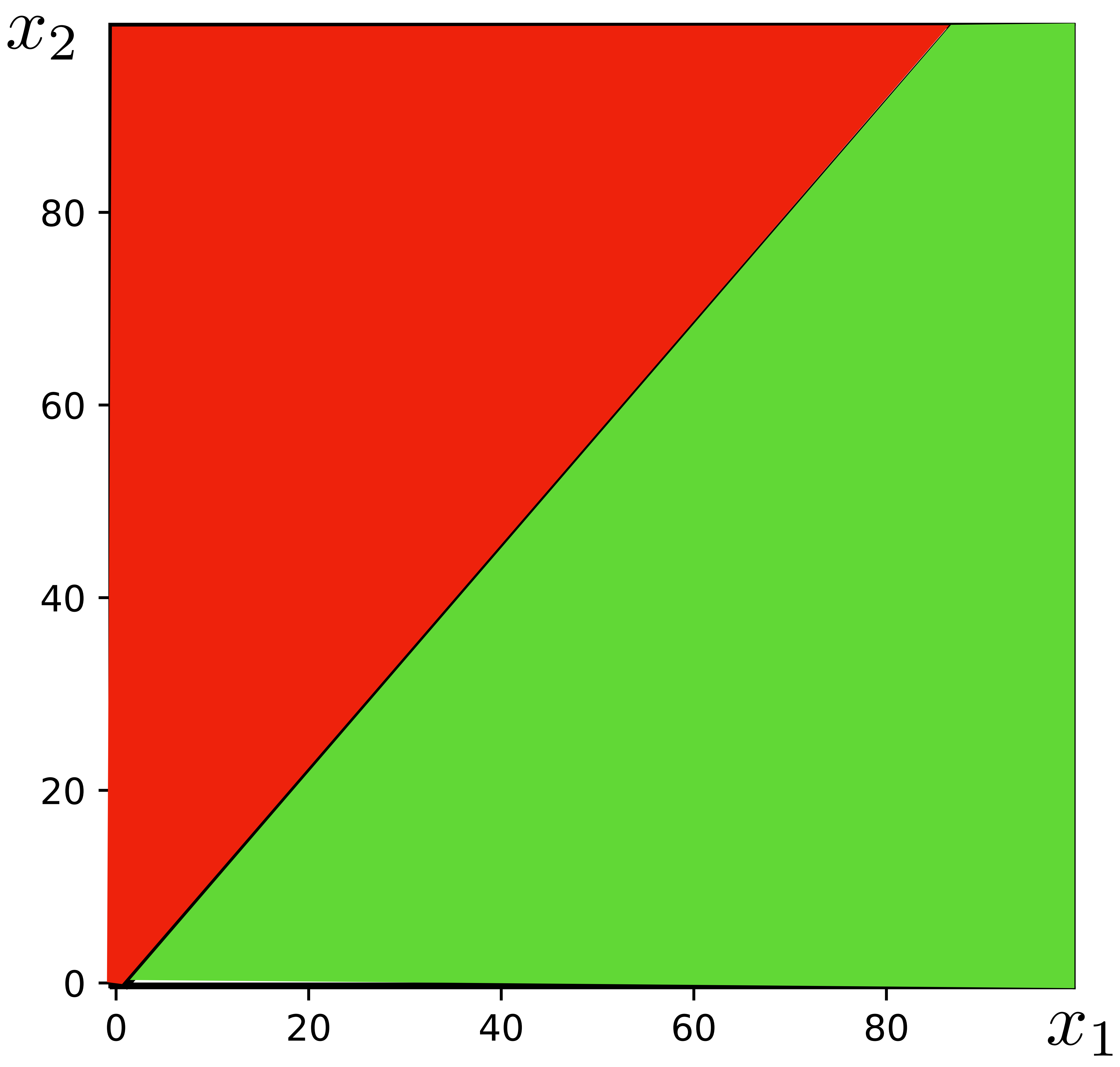


$$z = \mathbf{x}^T \mathbf{w} + b$$

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

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

Example: linear classification with a perceptron



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

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

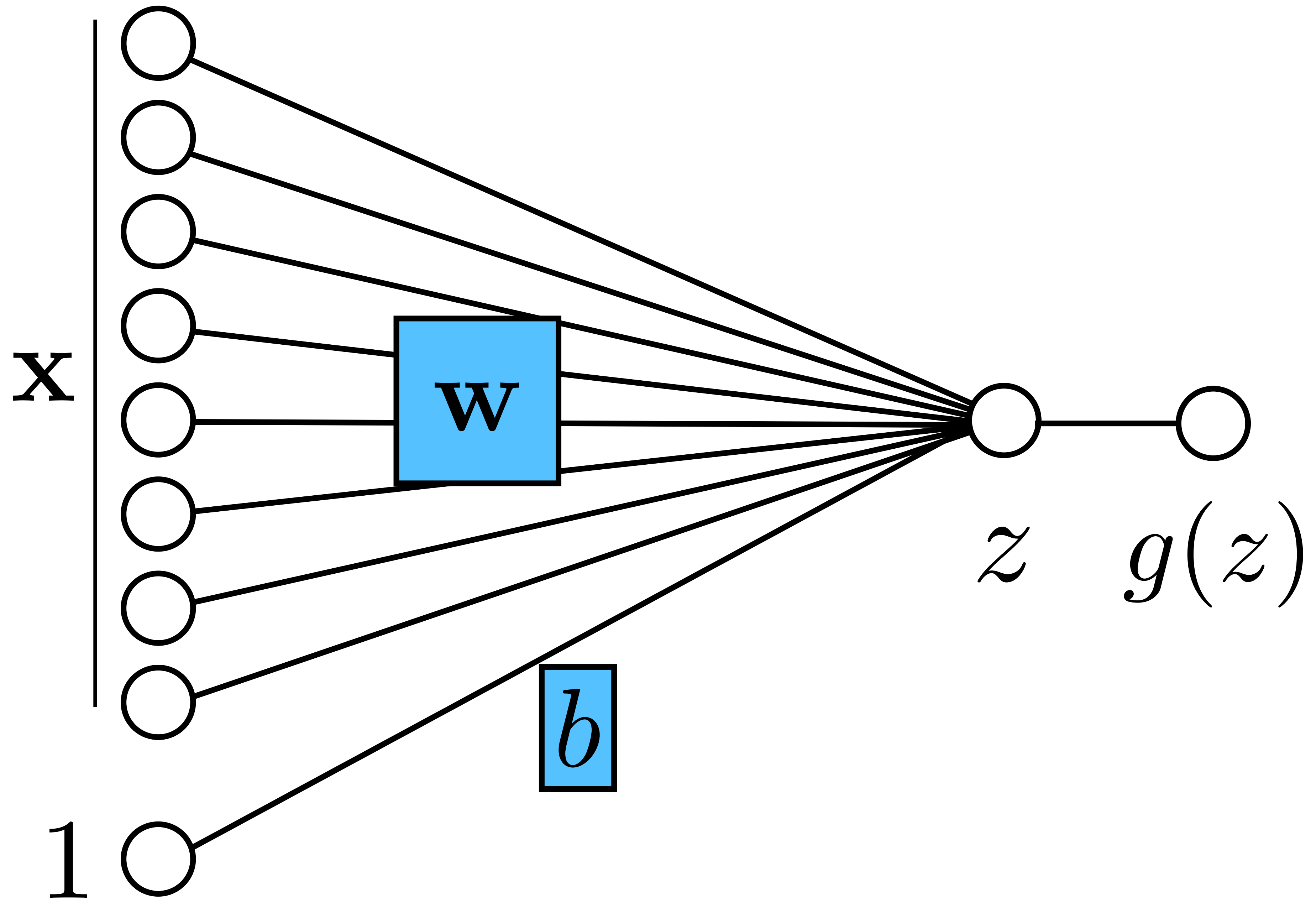
$$\mathbf{w}^*, b^* = \arg\min_{\mathbf{w}, b} \sum_{i=1}^{N} \mathcal{L}(g(z^{(i)}), y^{(i)})$$

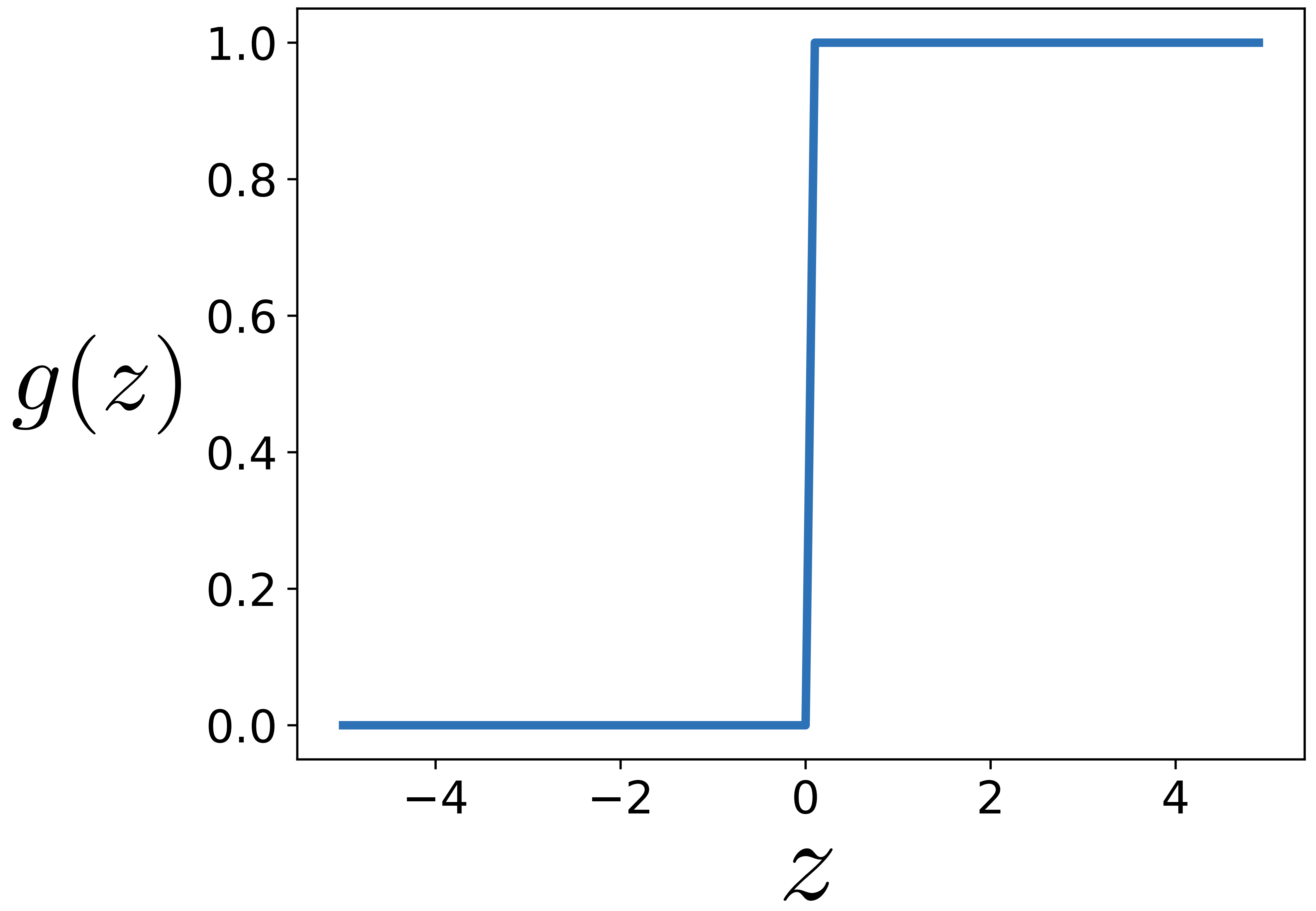
Computation in a neural net

Input representation

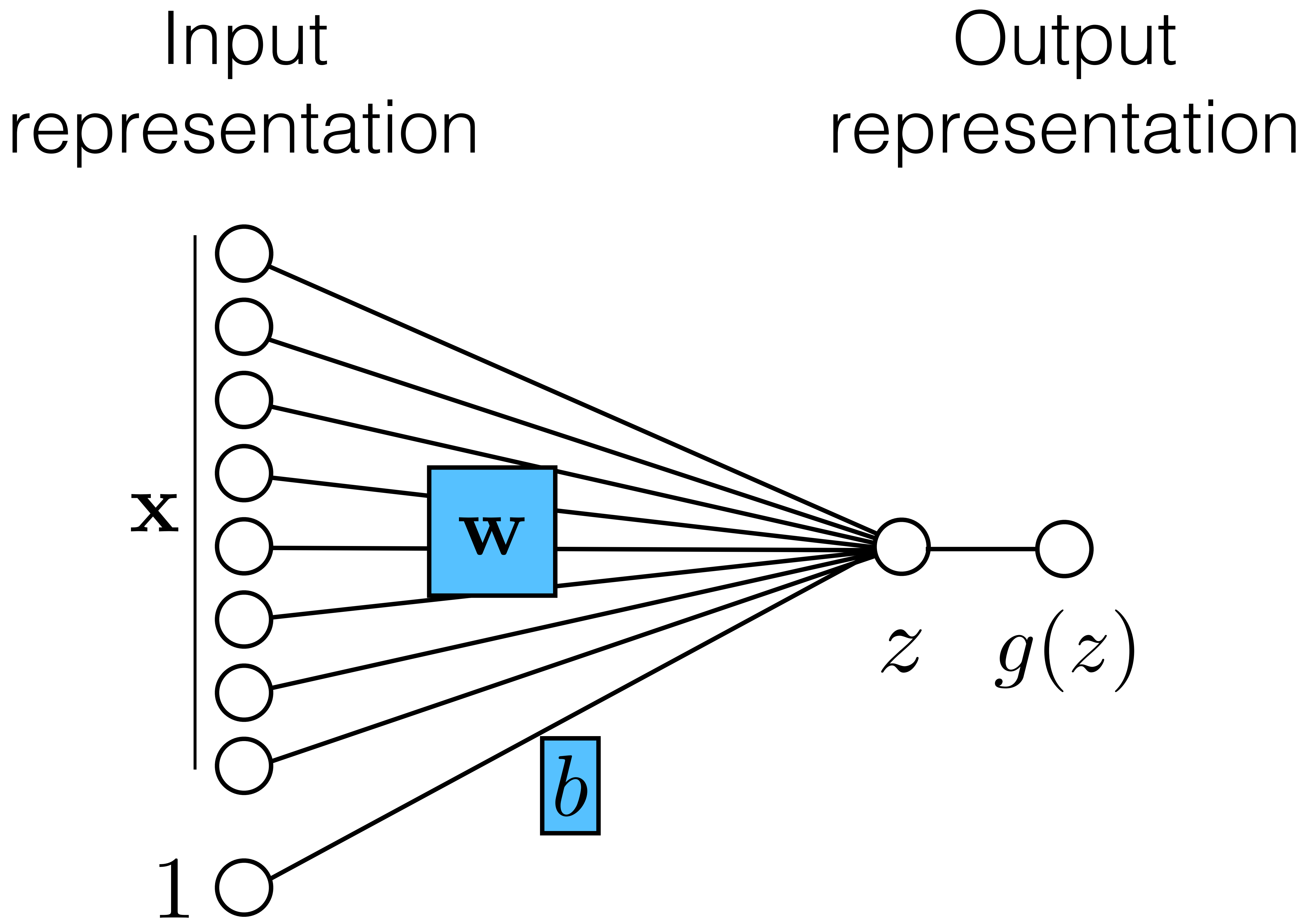
Output representation

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



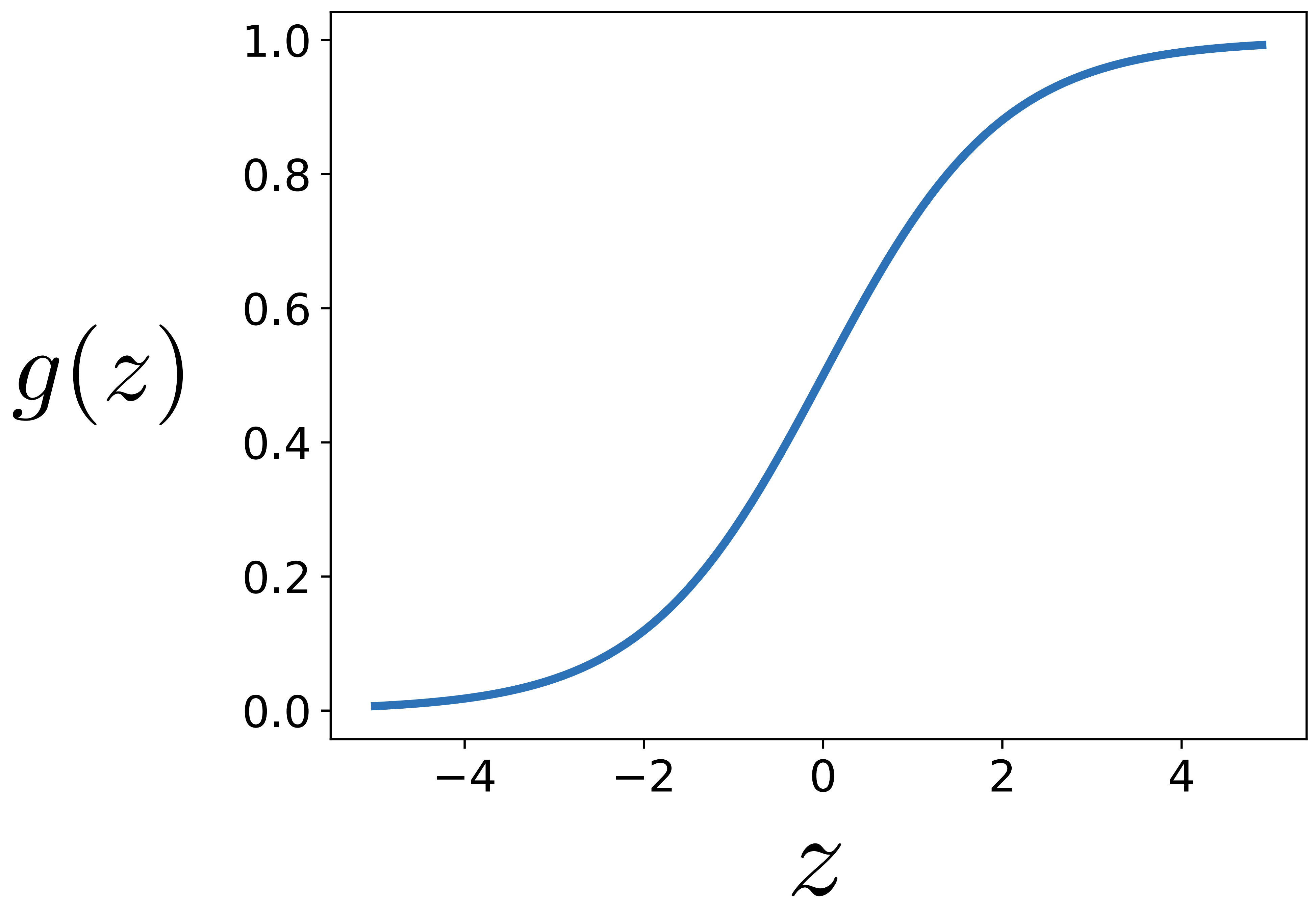


Computation in a neural net — nonlinearity



Sigmoid

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

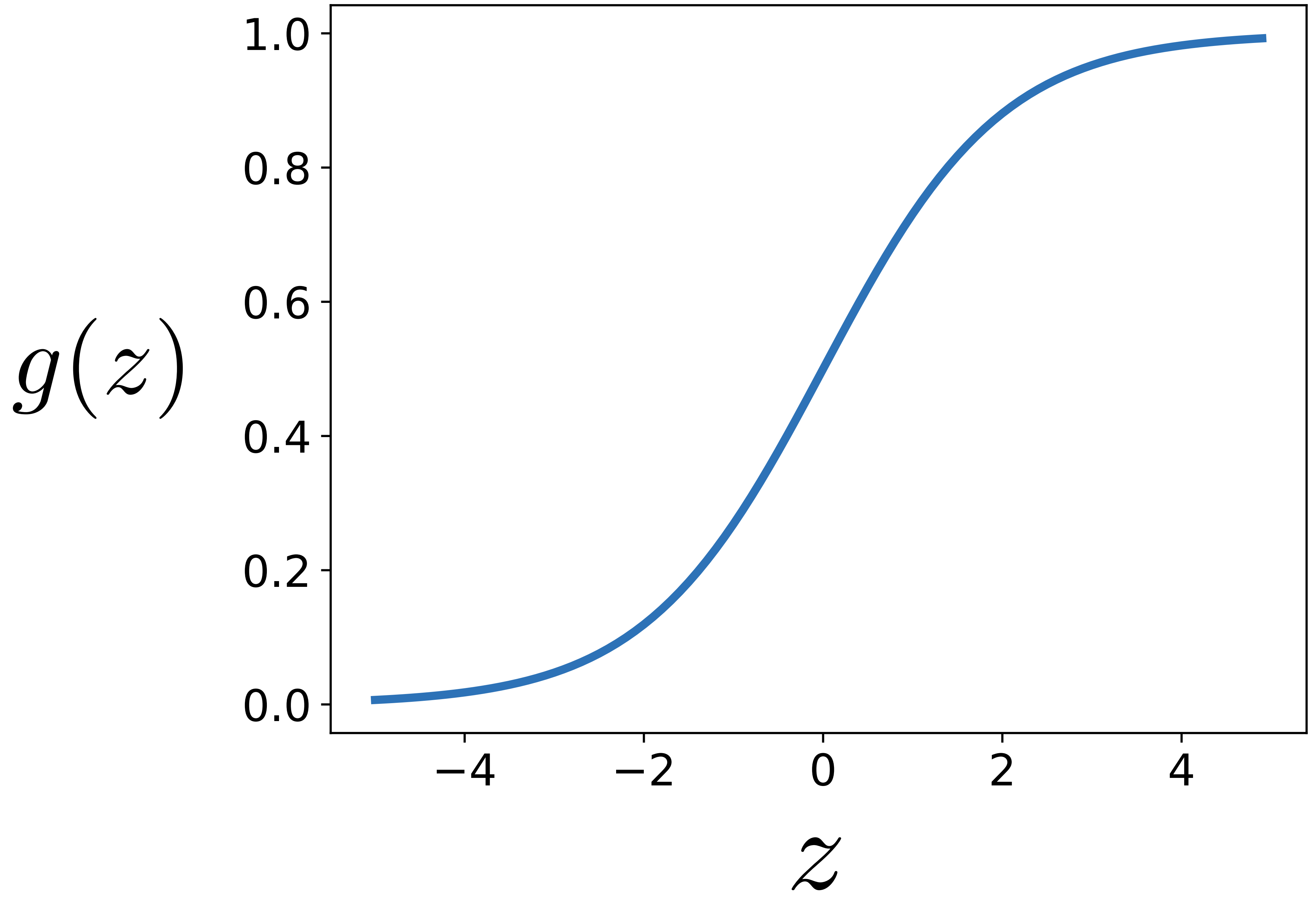


Computation in a neural net — nonlinearity

- 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

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



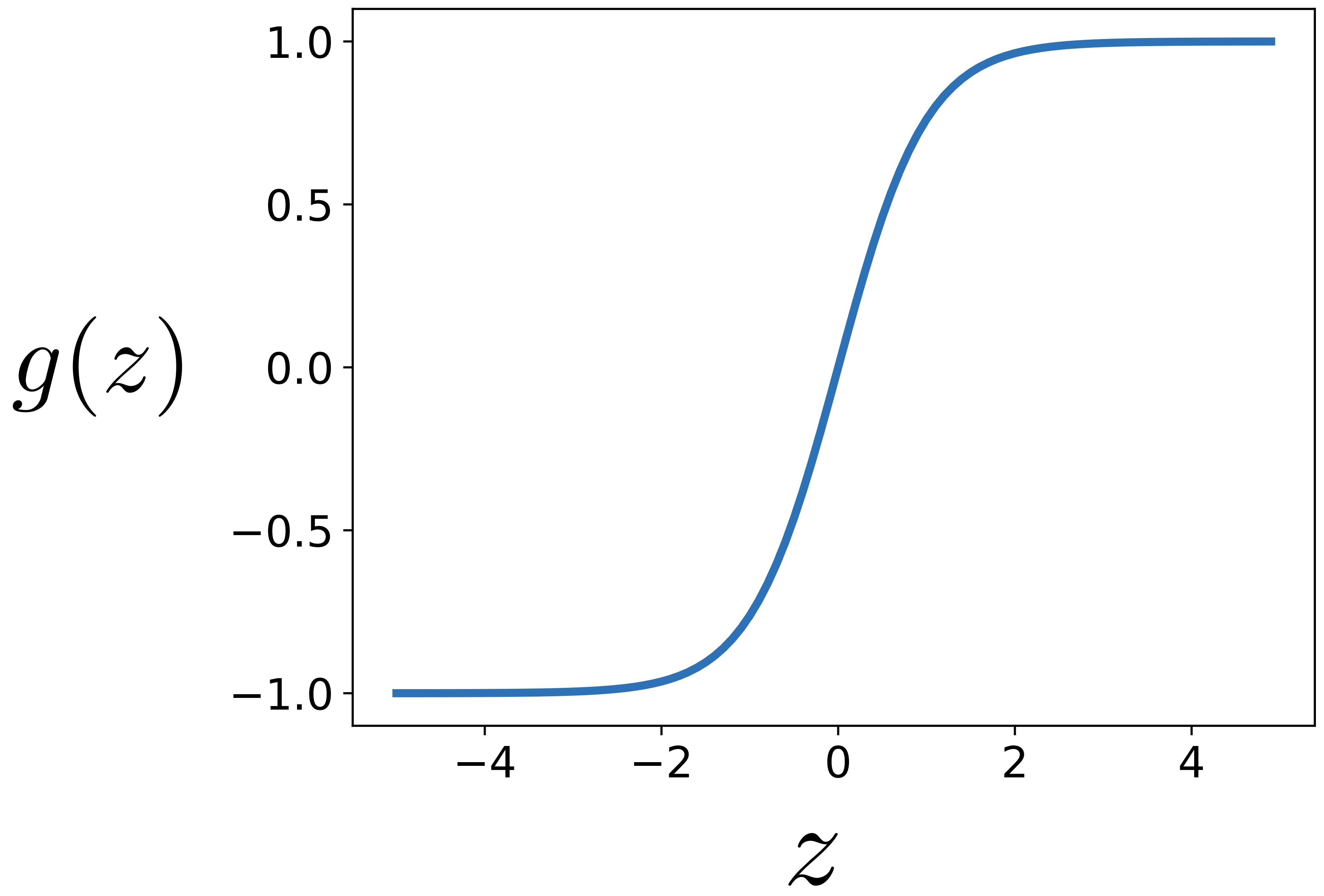
Computation in a neural net — nonlinearity

- Bounded between [-1,+1]
- Saturation for large +/- inputs
- Gradients go to zero
- Outputs centered at 0
- Preferable to sigmoid

tanh(z) = 2 sigmoid(2z) –1

Tanh

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

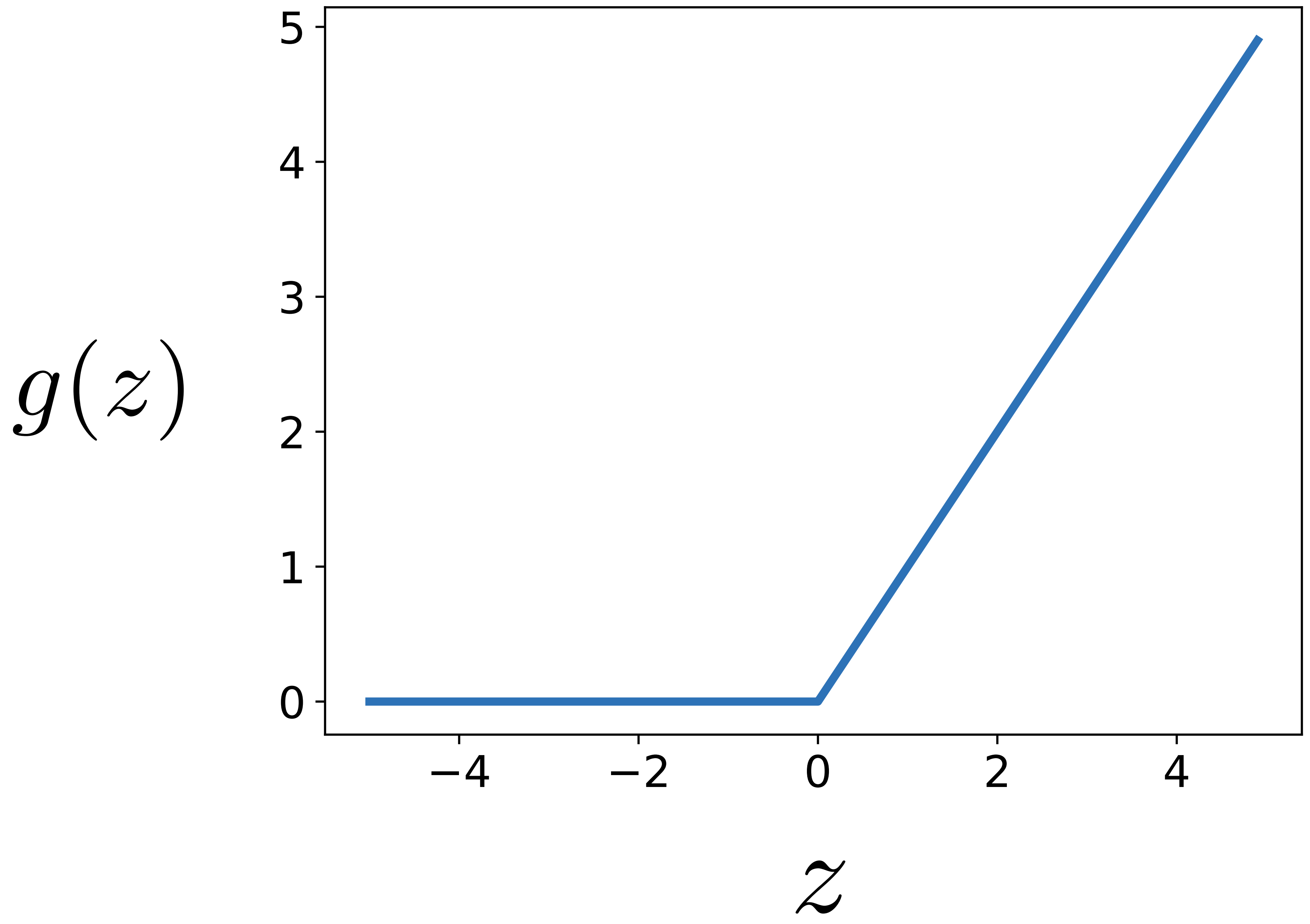


Computation in a neural net — nonlinearity

- 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)

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

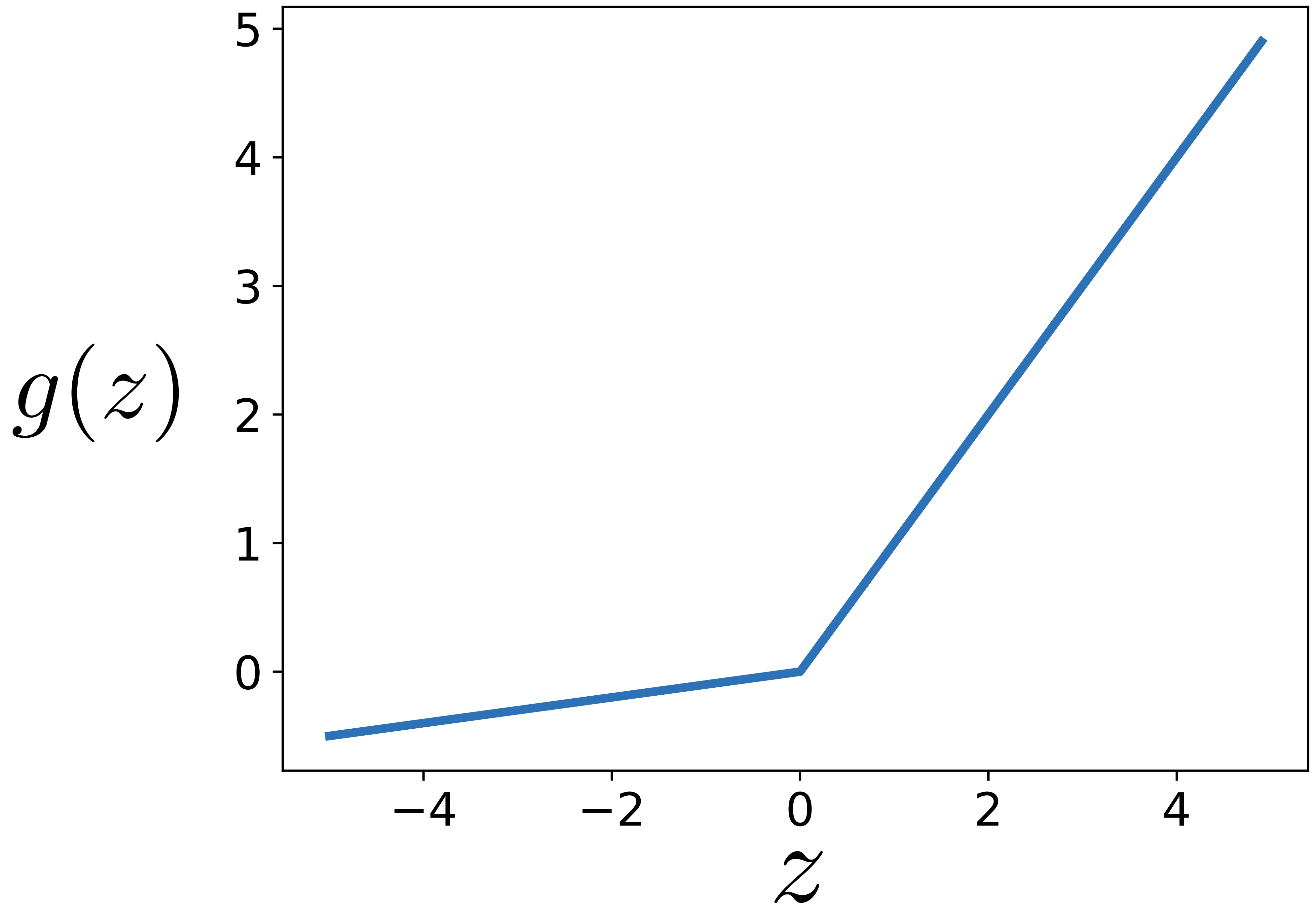


Computation in a neural net — nonlinearity

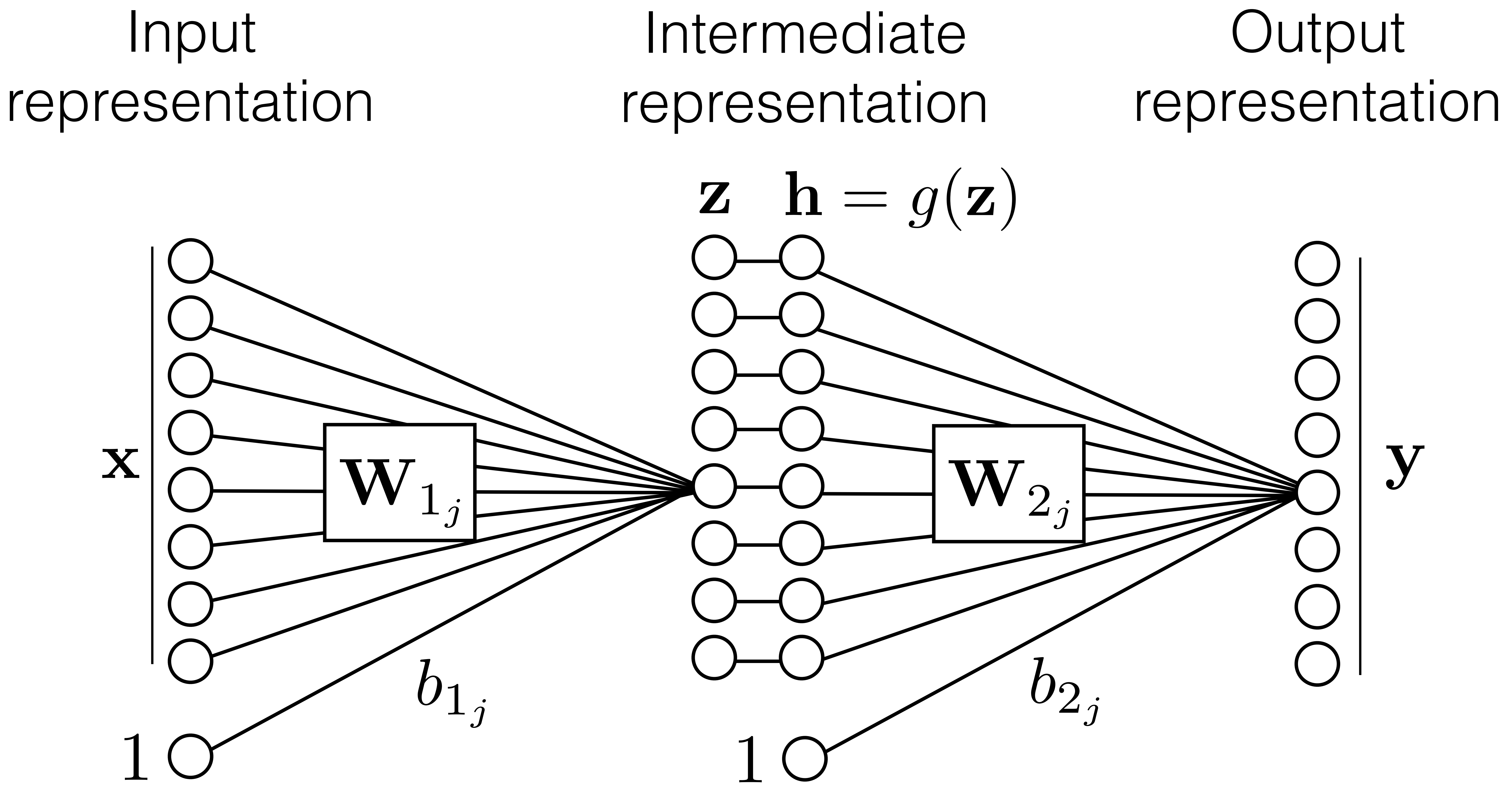
- where α is small (e.g. 0.02)
- Efficient to implement: $\frac{\partial g}{\partial z} = \begin{cases} -a, & \text{if} \quad z < 0 \\ 1, & \text{if} \quad z \geq 0 \end{cases}$
- Also known as probabilistic ReLU (PReLU)
- Has non-zero gradients everywhere (unlike ReLU)
- α can also be learned (see Kaiming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

Leaky ReLU

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

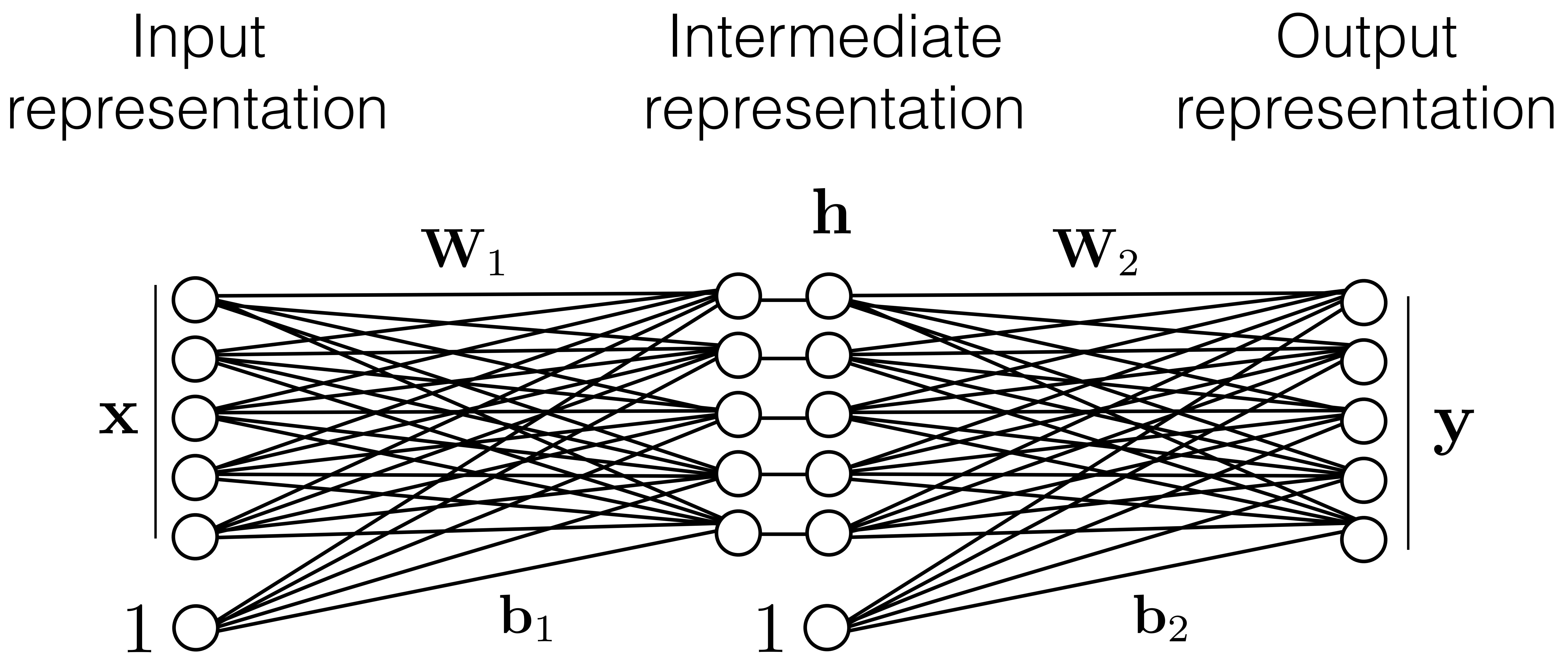


Stacking layers



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

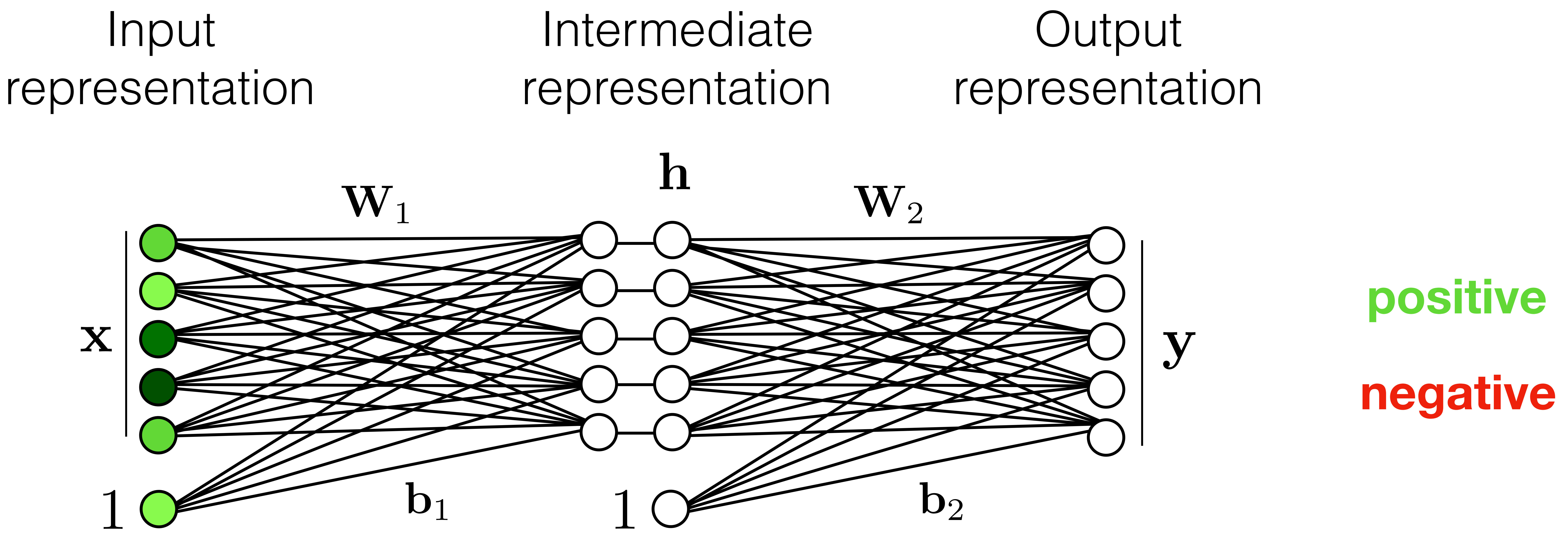
Stacking layers



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

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

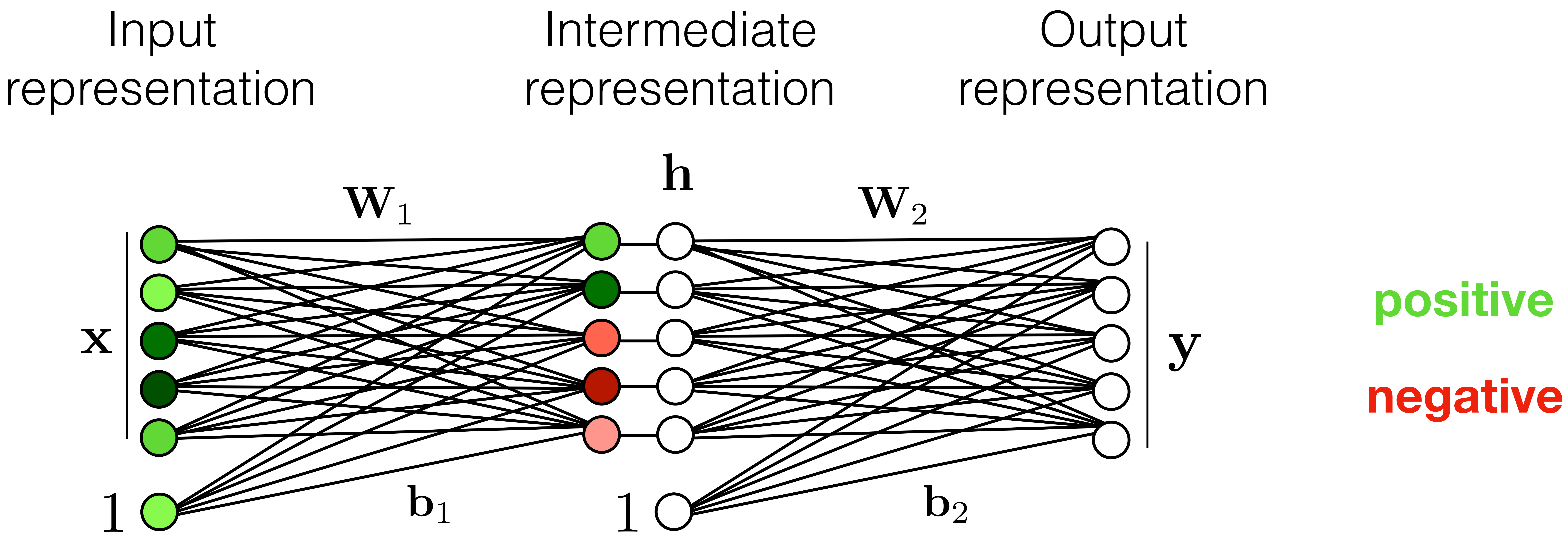
Stacking layers



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

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

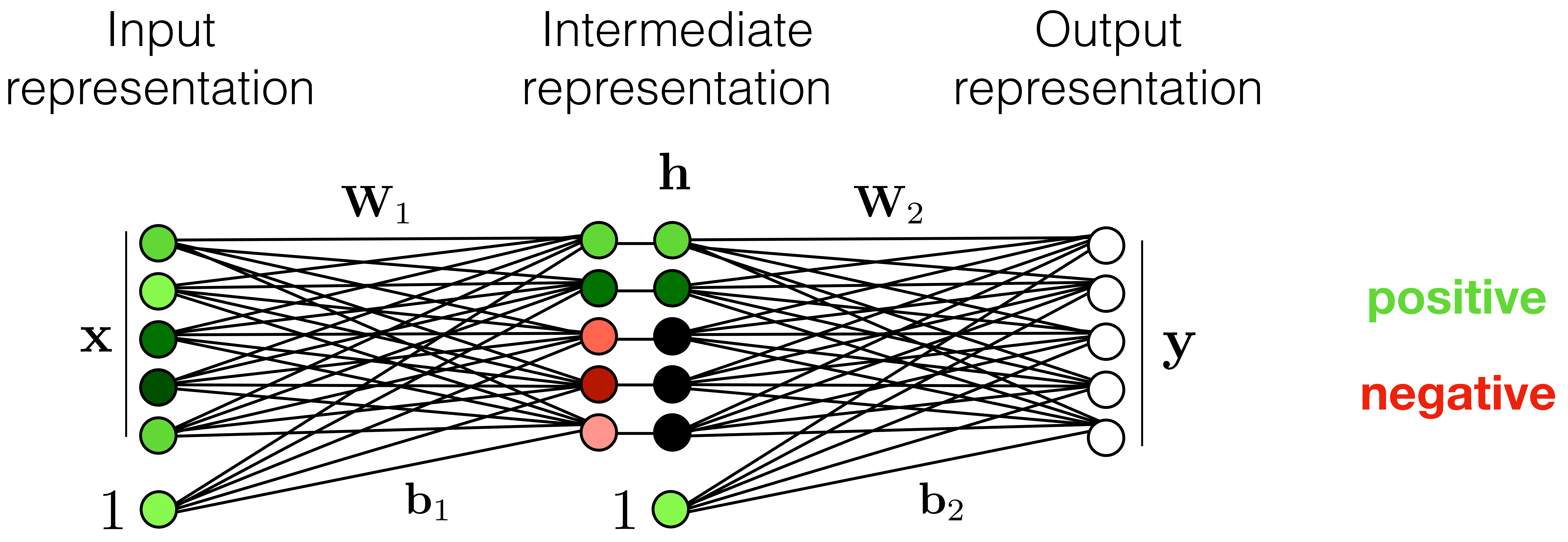
Stacking layers



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

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

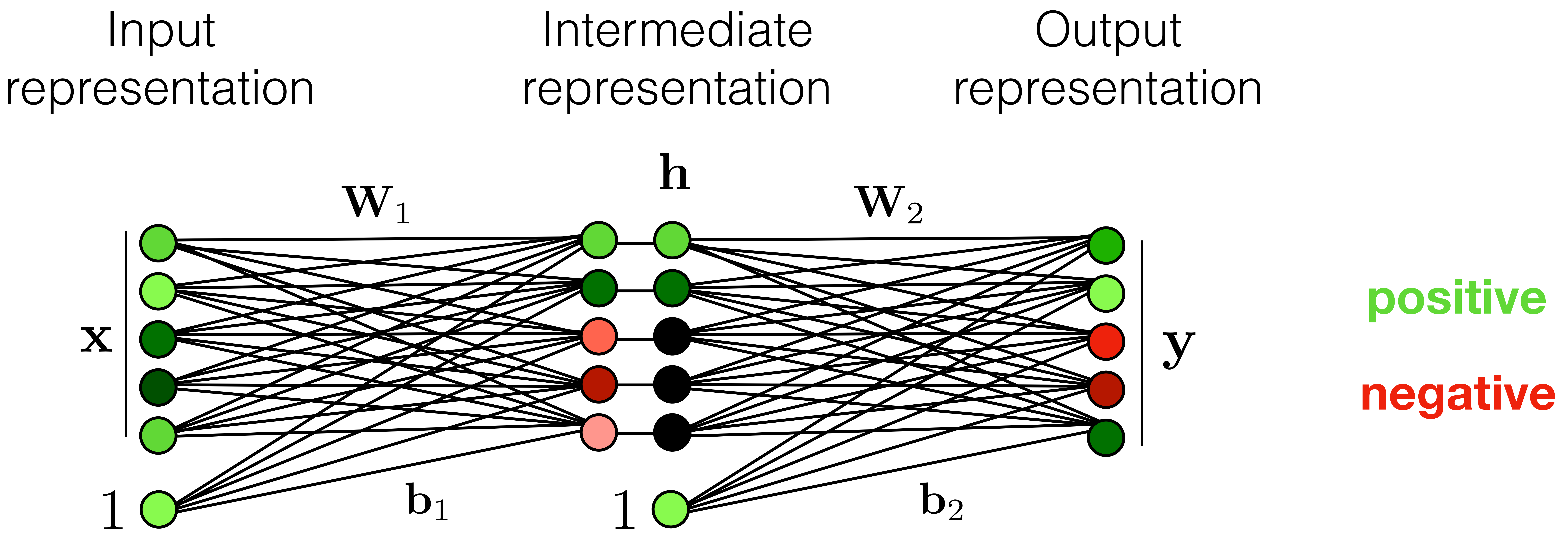
Stacking layers



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

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

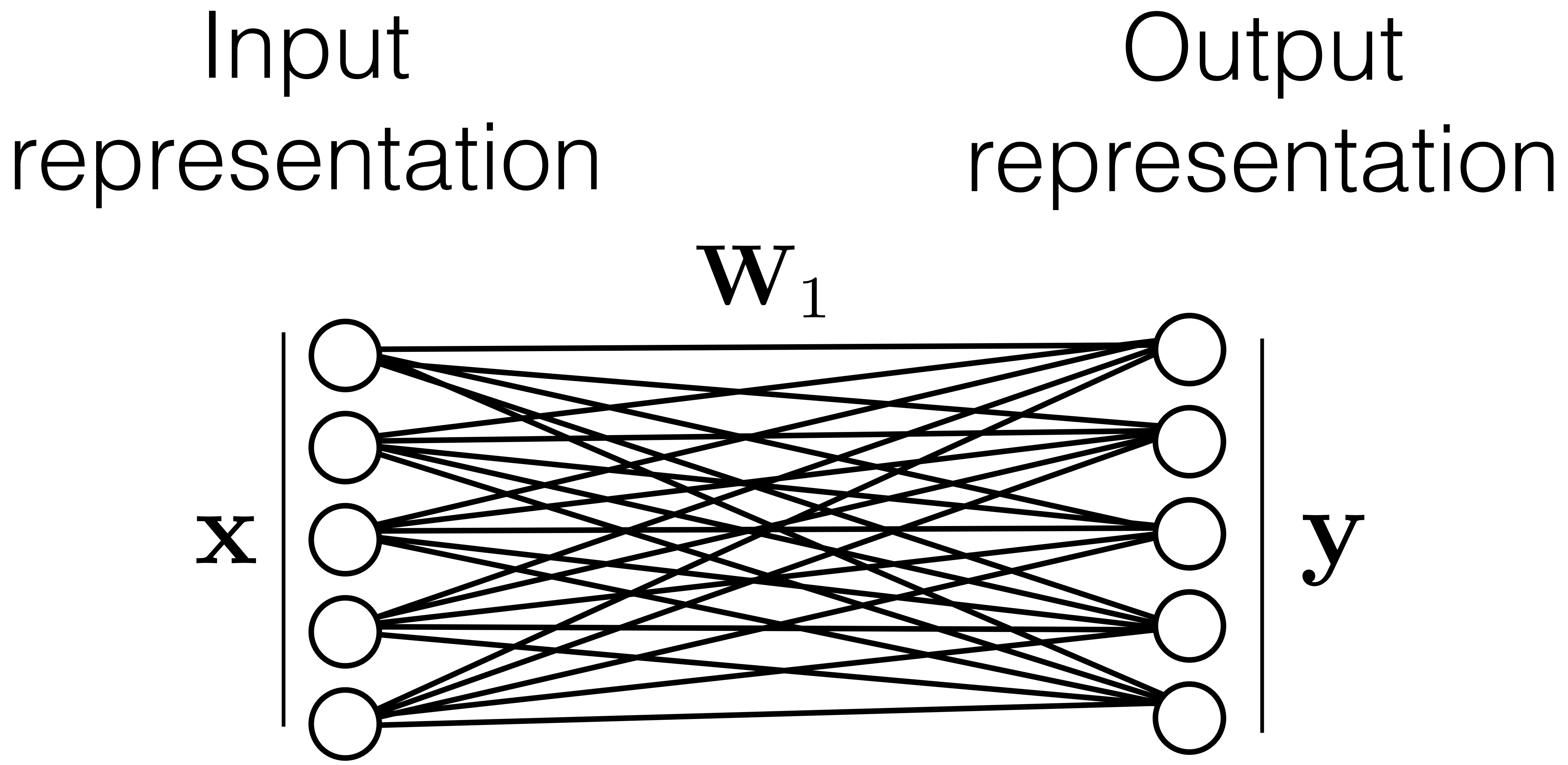
Stacking layers



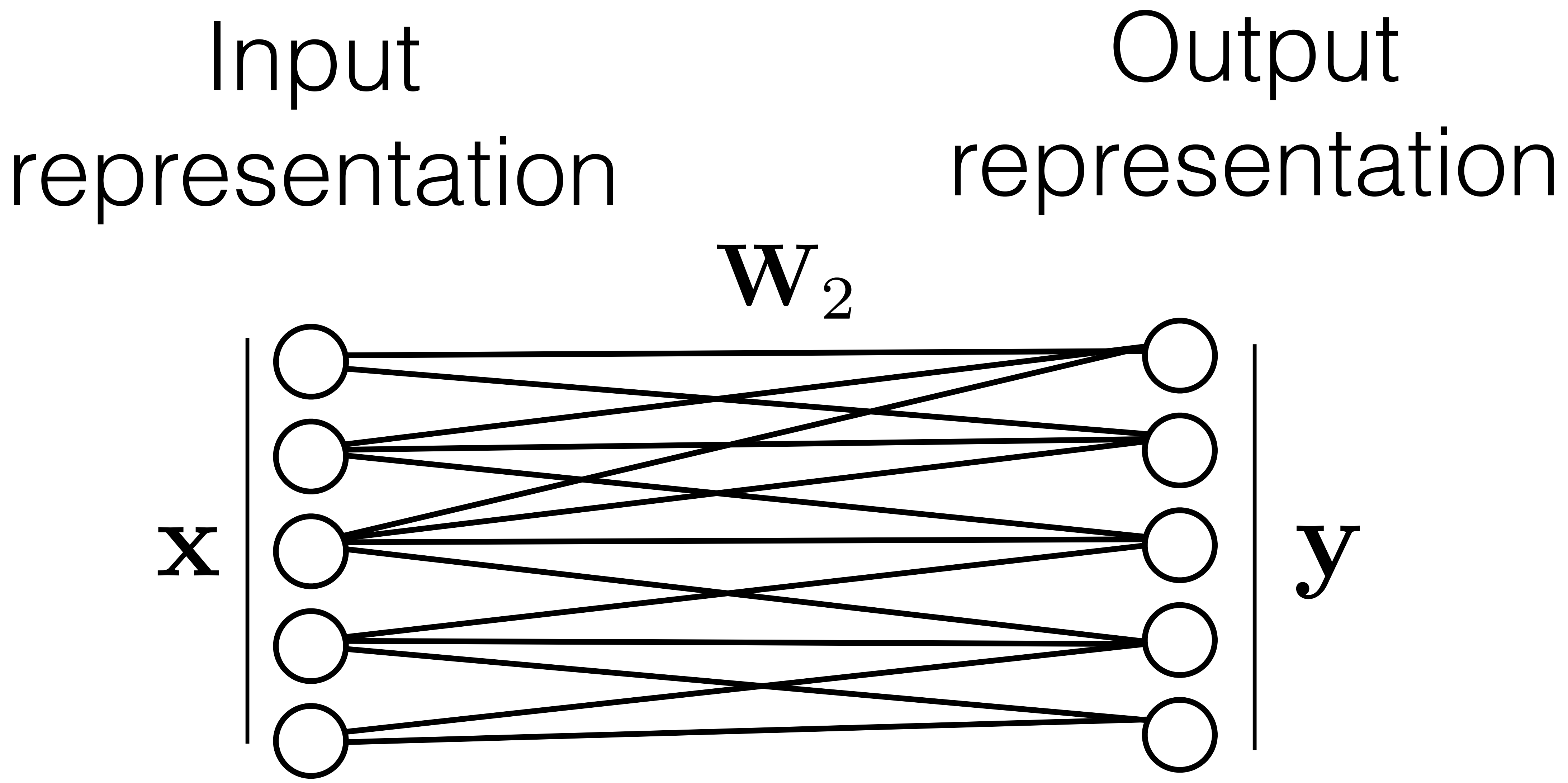
$$\mathbf{h} = g(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \qquad \mathbf{y} = g(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2)$$

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

Connectivity patterns

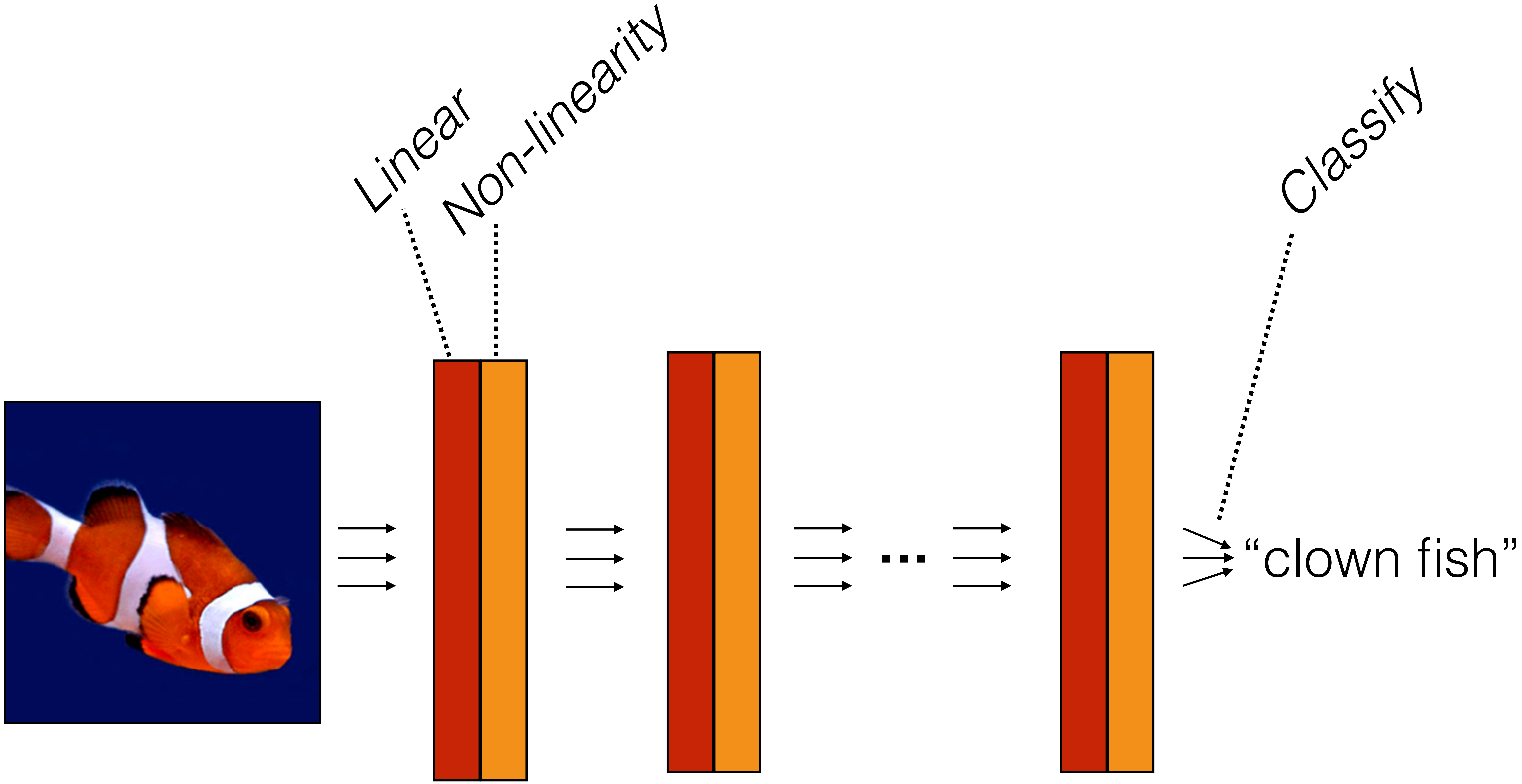


Fully connected layer



Locally connected layer
(Sparse W)

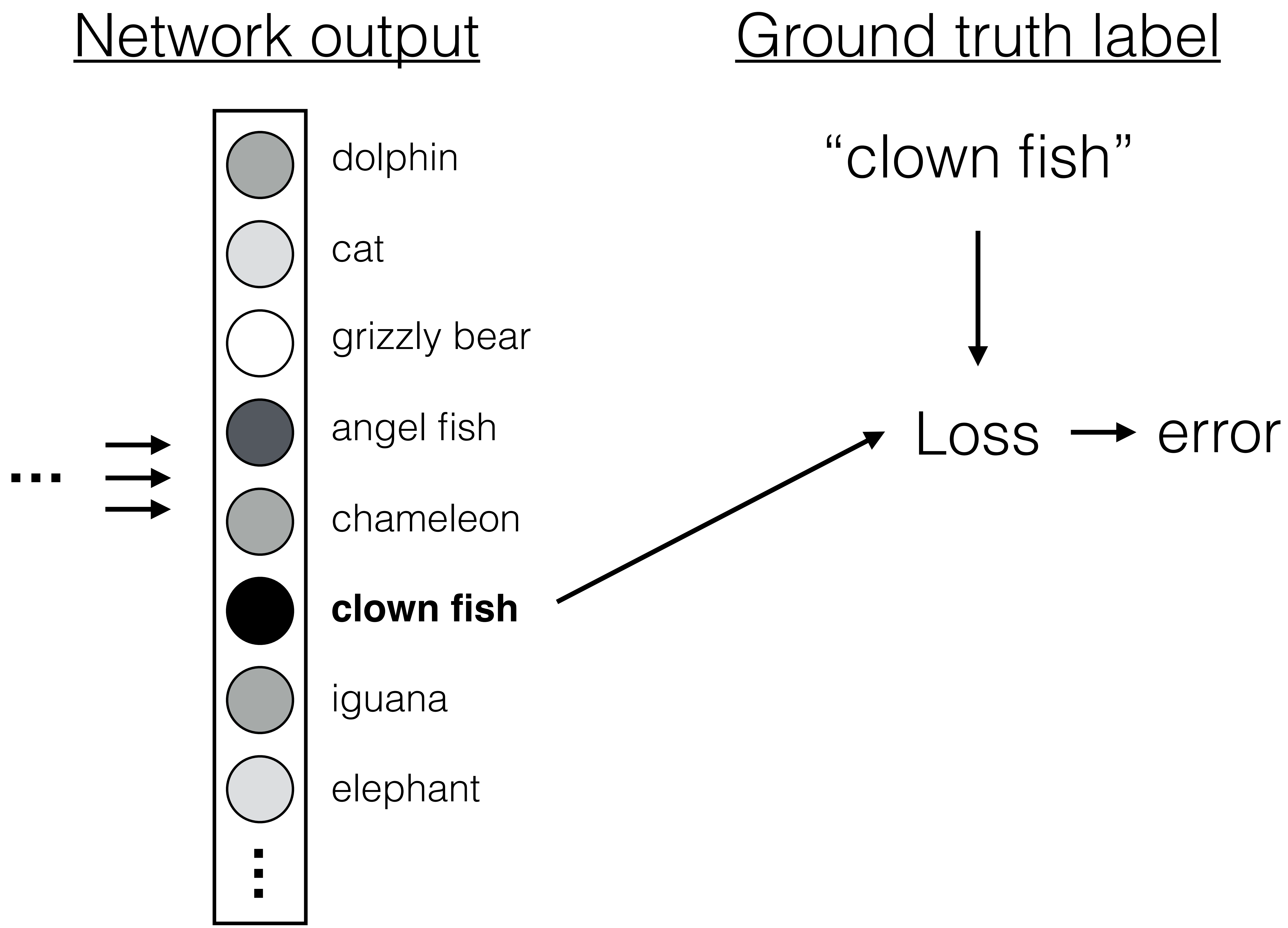
Deep nets



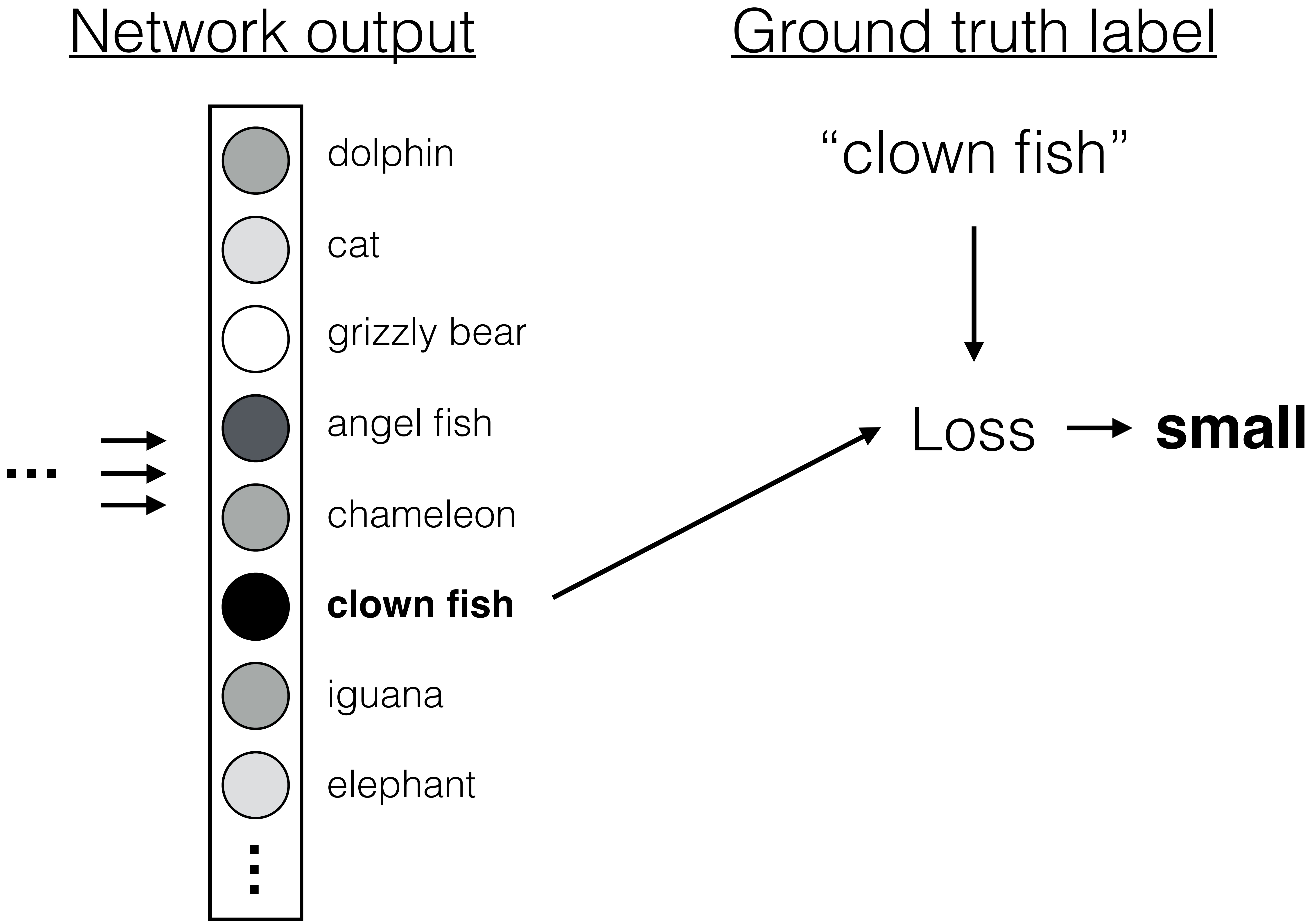
$$f(\mathbf{x}) = f_L(f_{L-1}(\ldots f_2(f_1(\mathbf{x}))))$$

Classifier layer

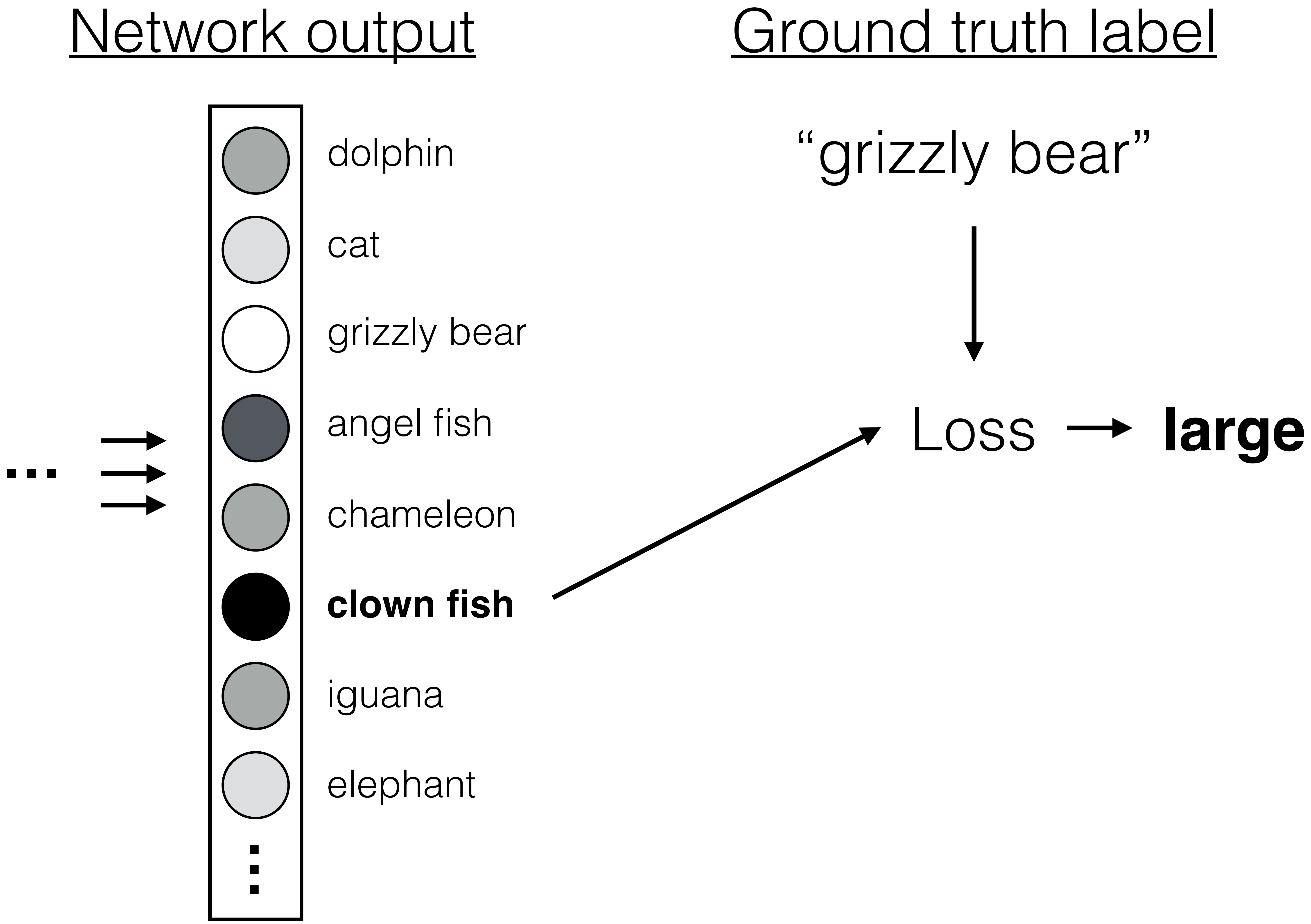
Loss function



Loss function

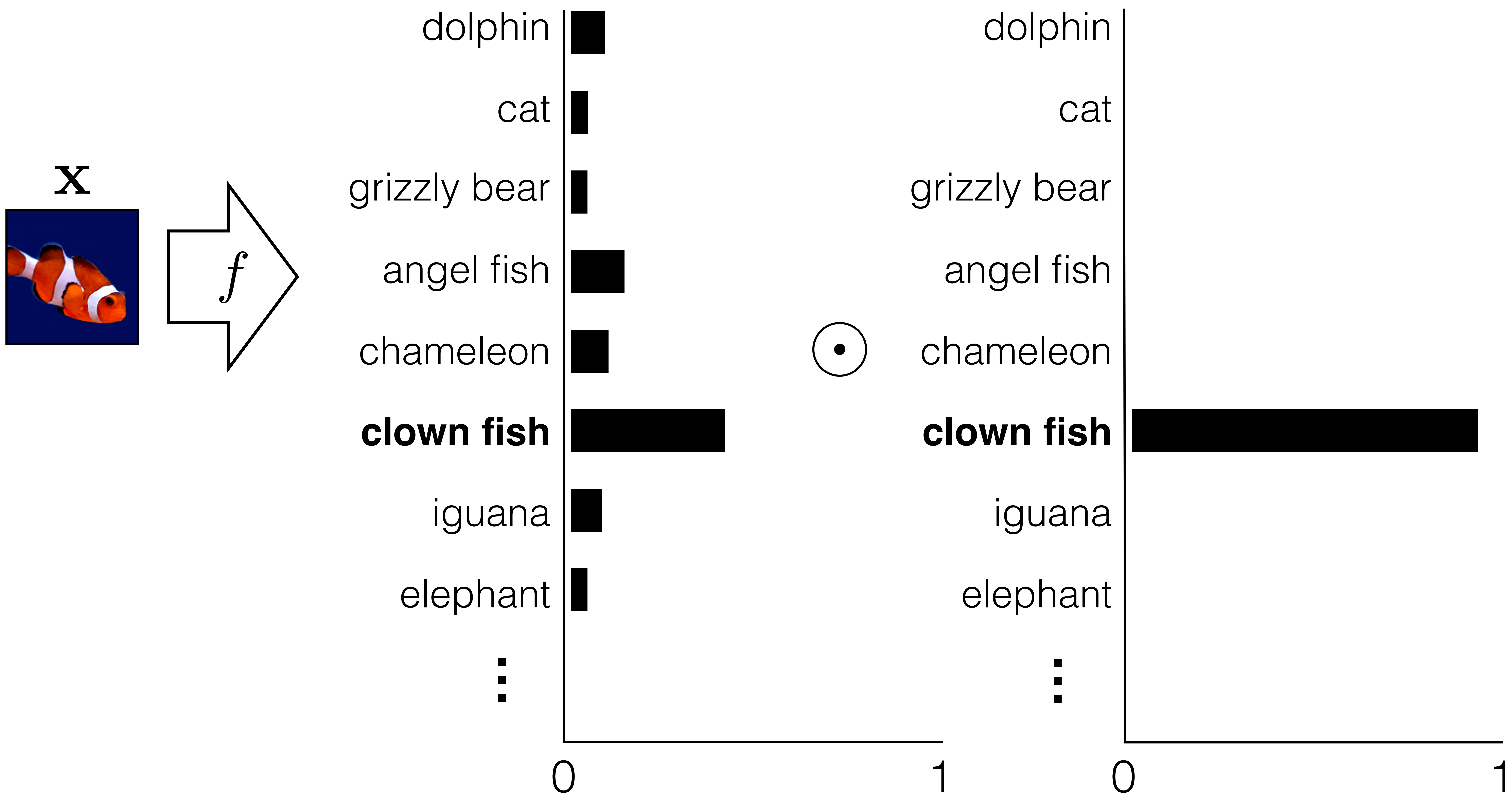


Loss function

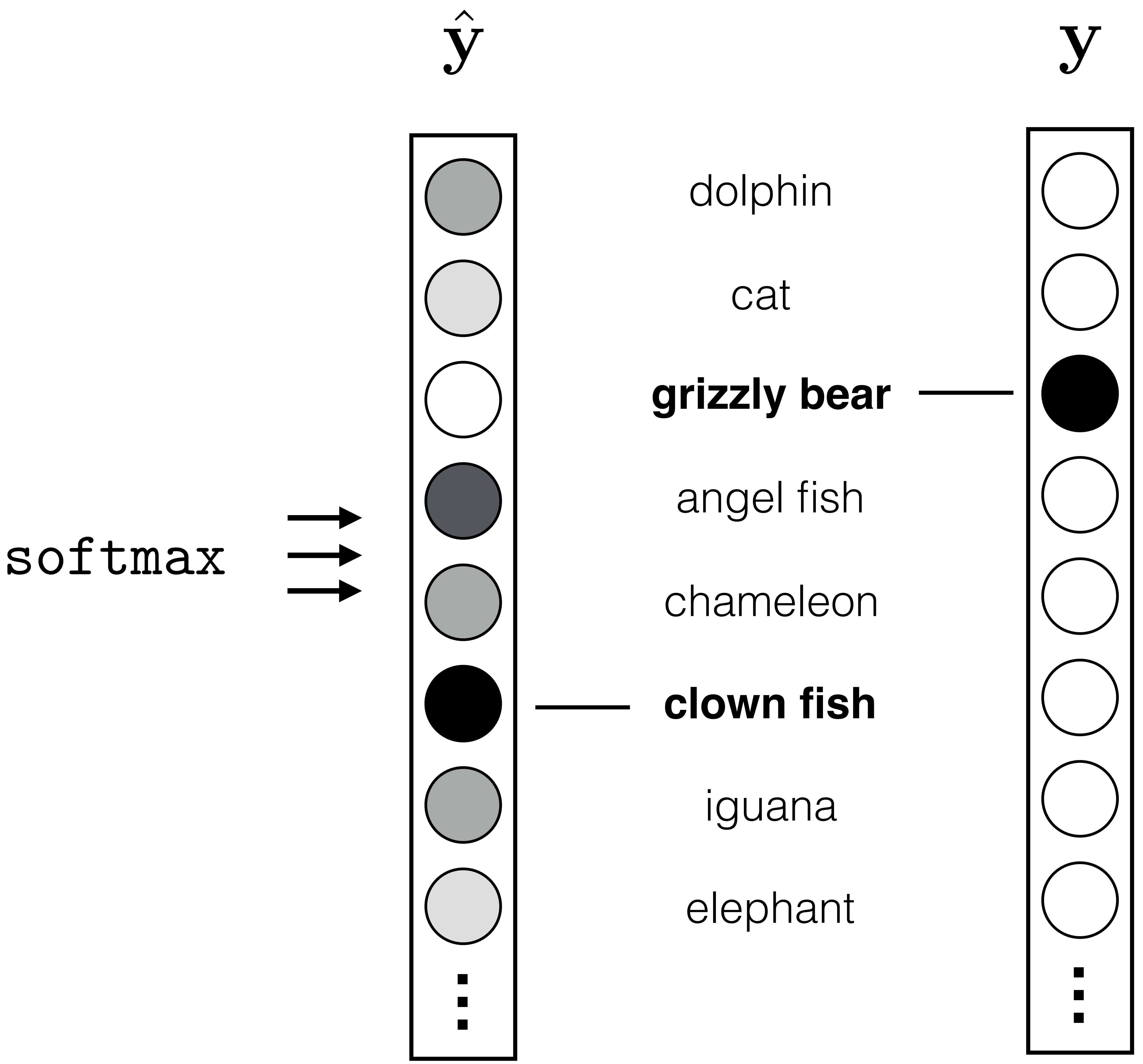


Prediction $\hat{\mathbf{y}}$ Ground truth label $\mathbf{y}$

$$f_\theta : X \to \mathbb{R}^K$$



Network output   Ground truth label



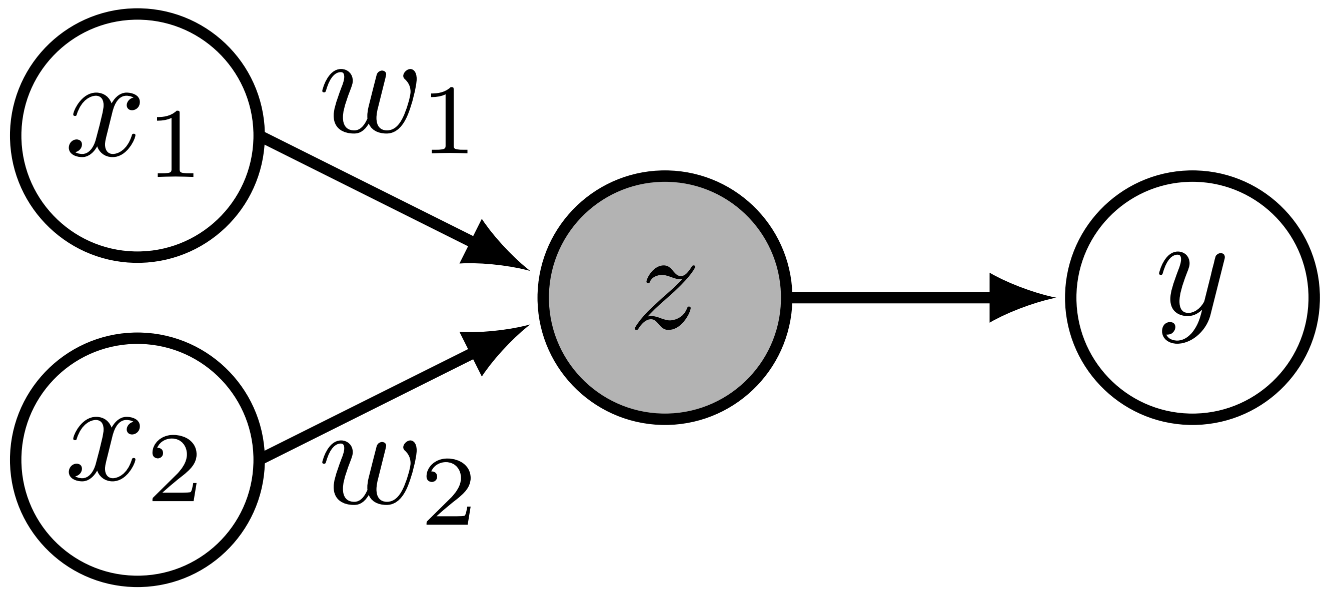
Probability of the observed data under the model

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

Representational power

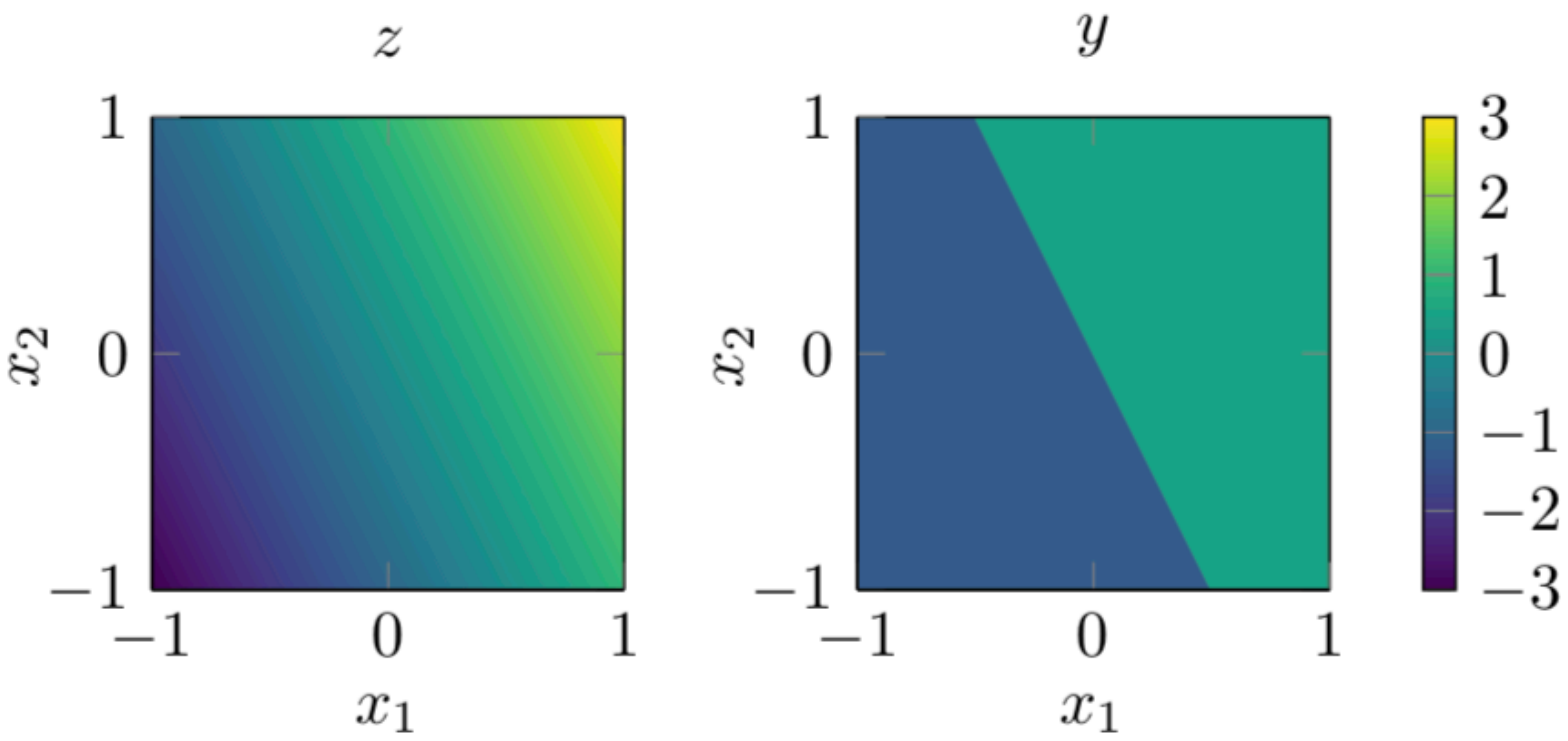
- 1 layer? Linear decision surface.
- 2+ layers? In theory, can represent any function. Assuming non-trivial non-linearity.
  - Bengio 2009,
    http://www.iro.umontreal.ca/~bengioy/papers/ftml.pdf
  - Bengio, Courville, Goodfellow book
    http://www.deeplearningbook.org/contents/mlp.html
  - Simple 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.

Example: linear classification with a perceptron



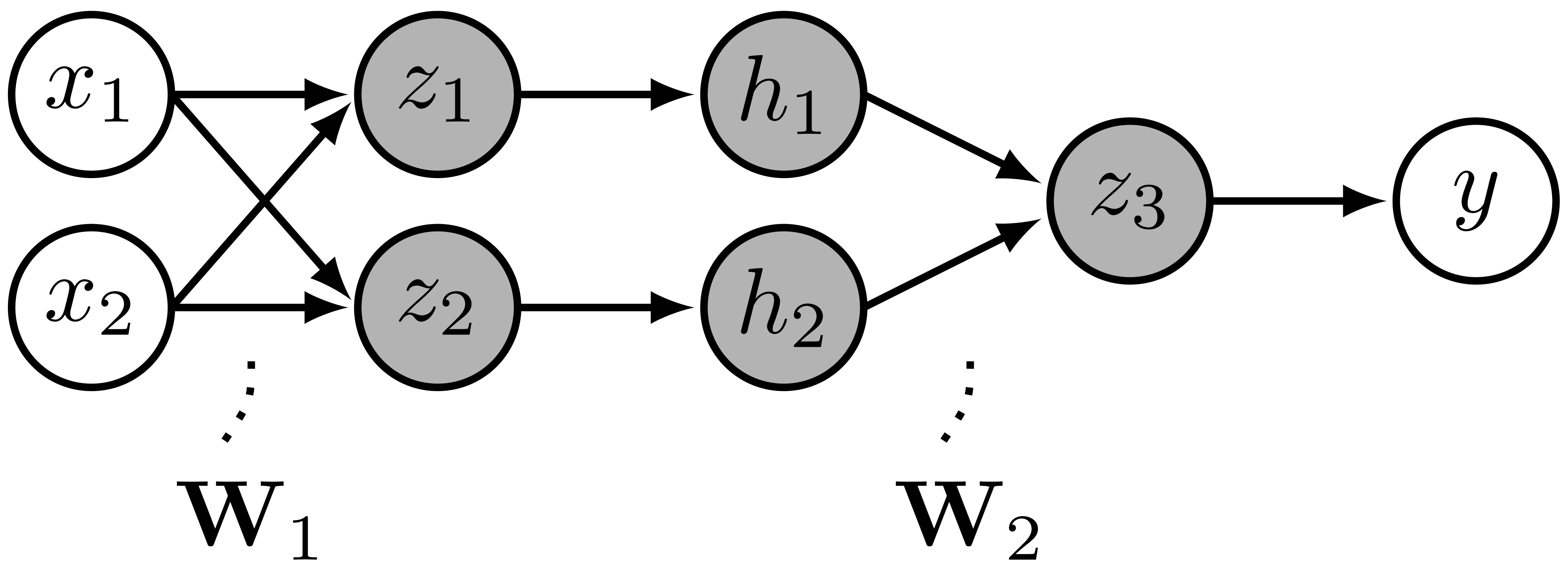
$$z = \mathbf{x}^T \mathbf{w} + b$$

$$y = g(z)$$

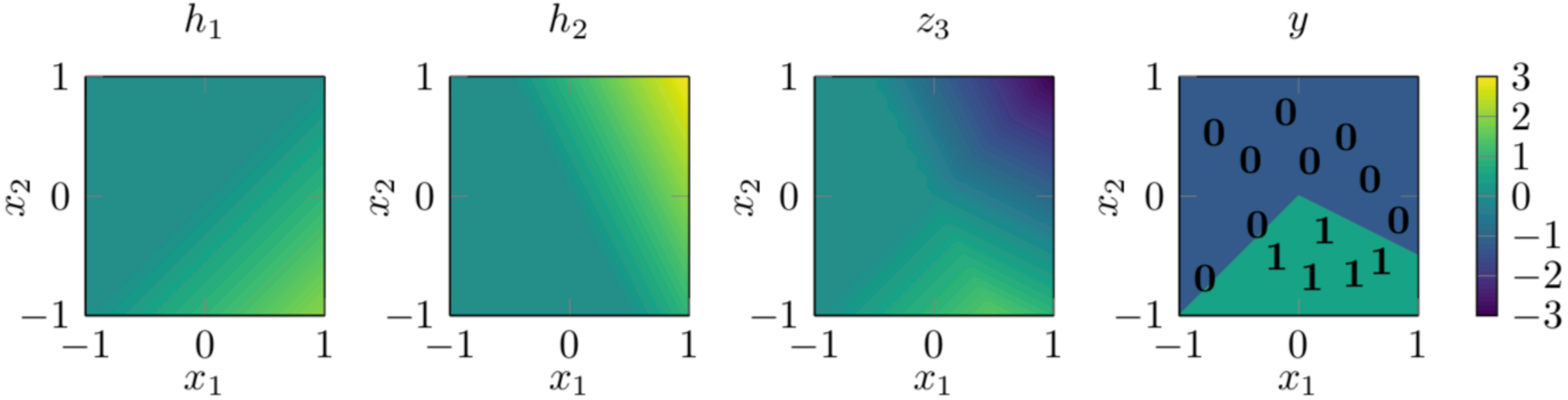


One layer neural net (perceptron) can perform linear classification.

Example: nonlinear classification with a deep net net

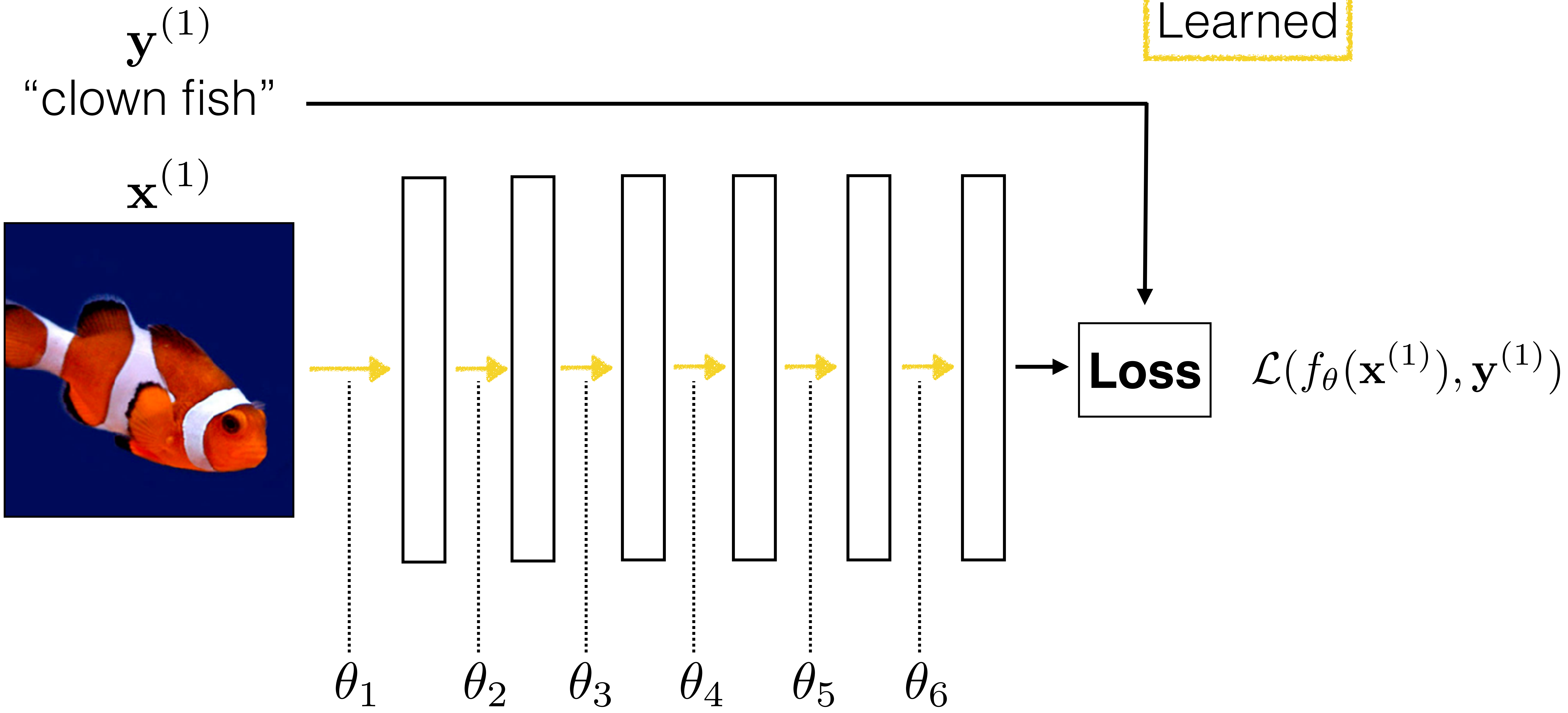


$$\mathbf{z} = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1$$
$$\mathbf{h} = g(\mathbf{z})$$
$$\mathbf{y} = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$$



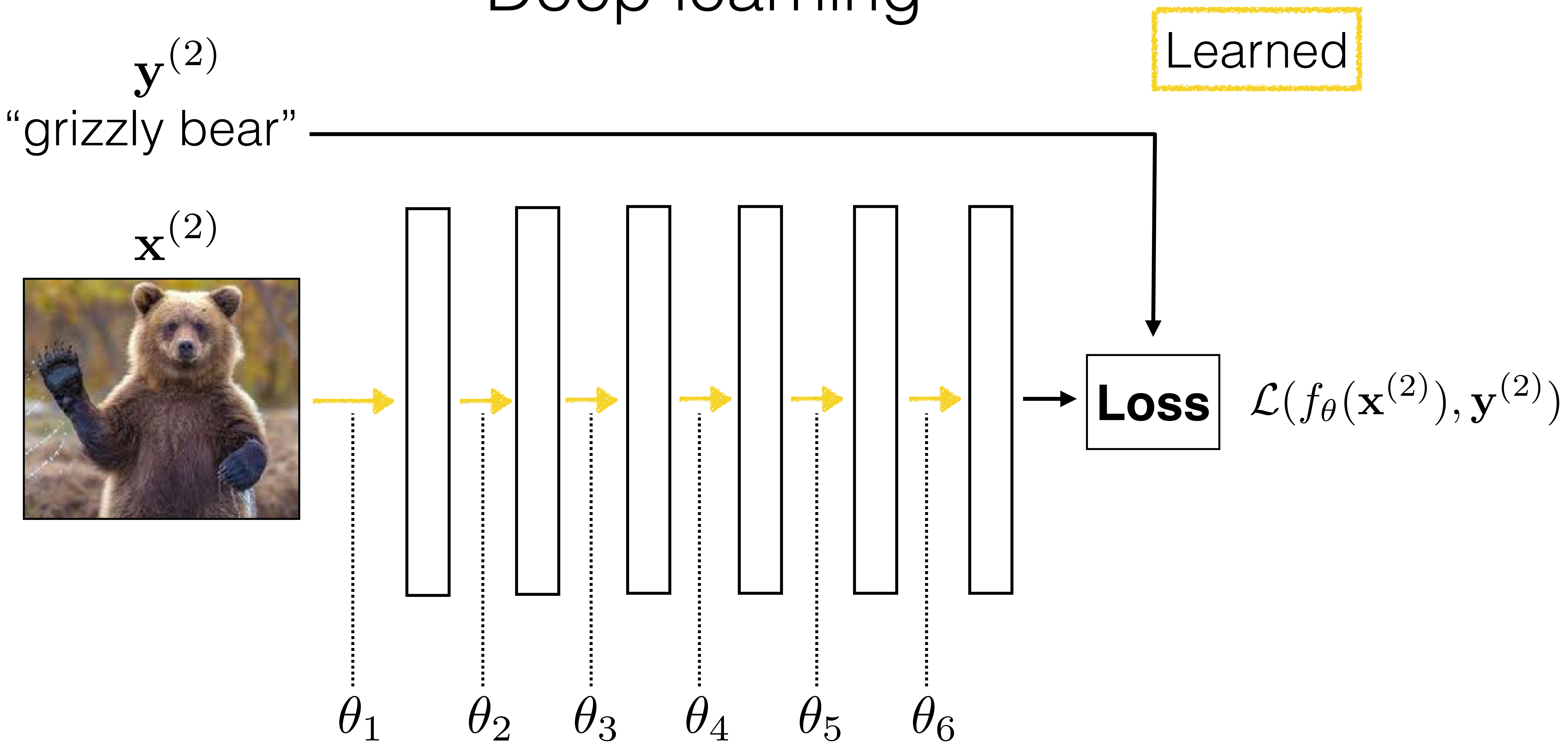
Deep learning

Learned



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

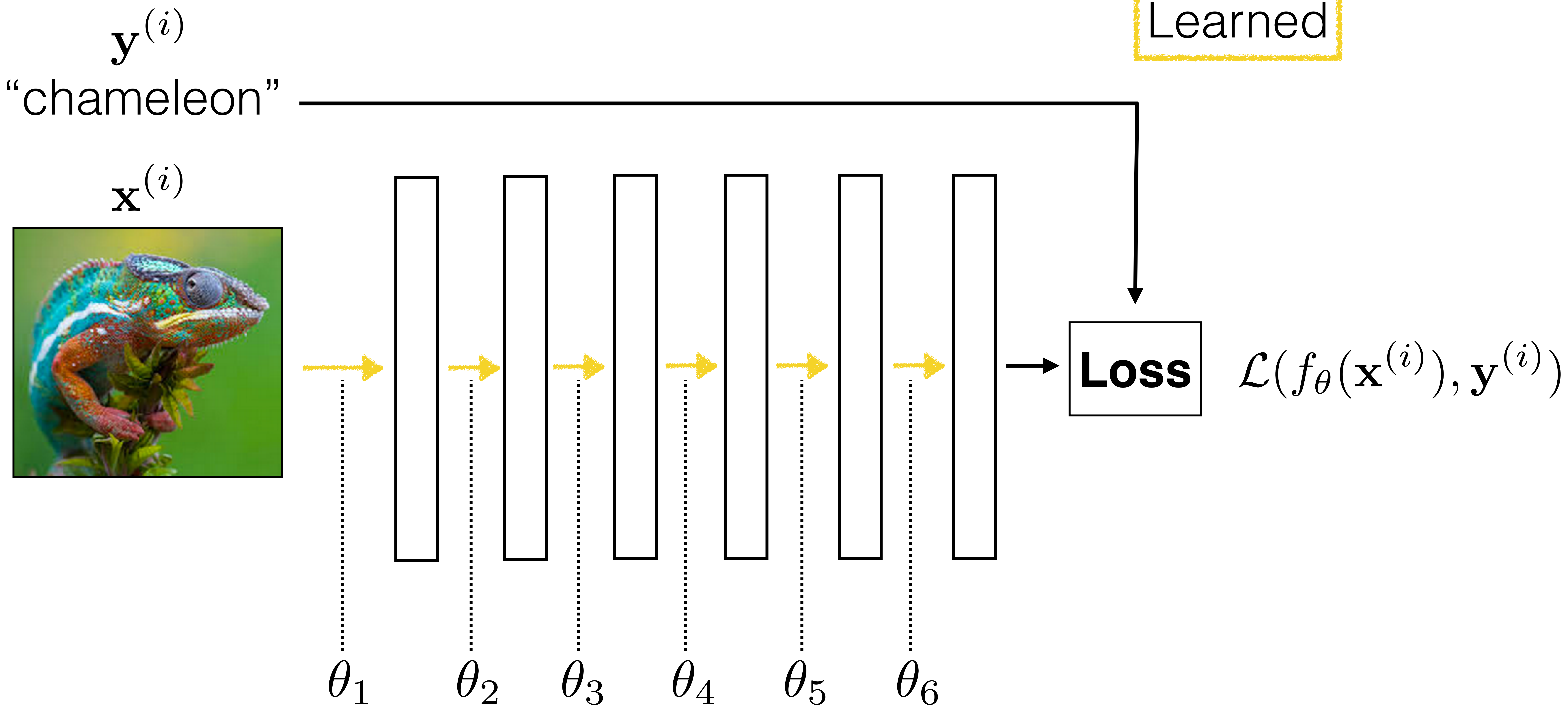
Deep learning



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

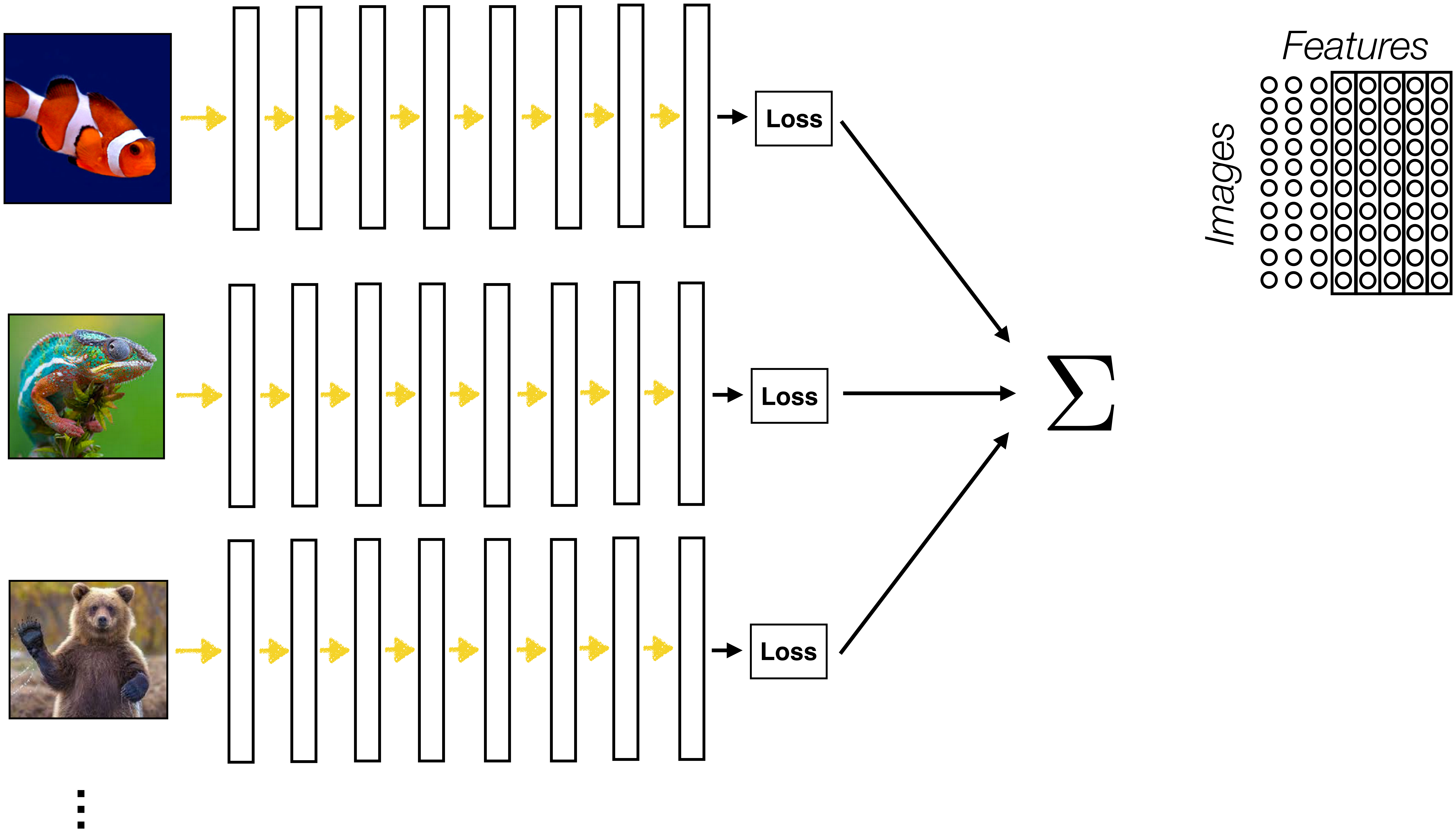
Deep learning

Learned



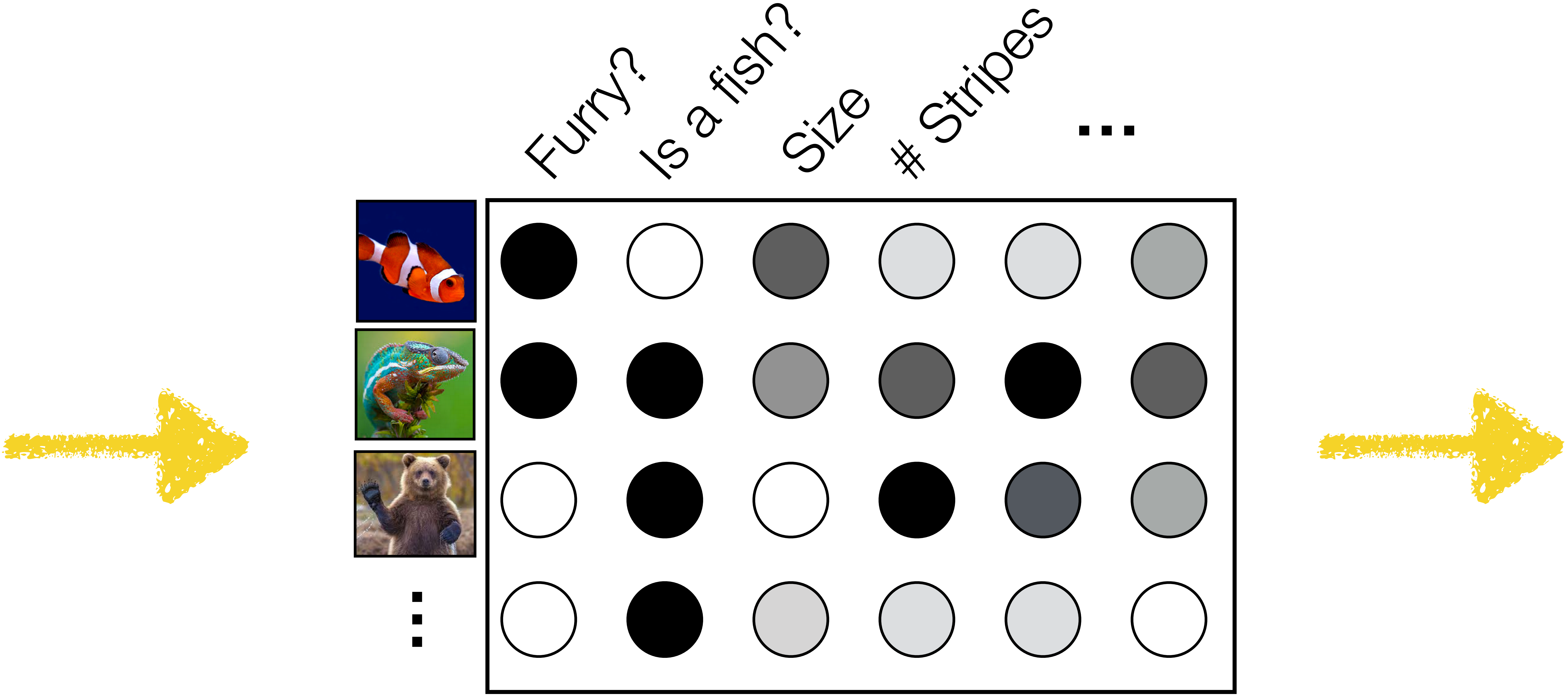
$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$$

Batch (parallel) processing



Tensors

(multi-dimensional arrays)



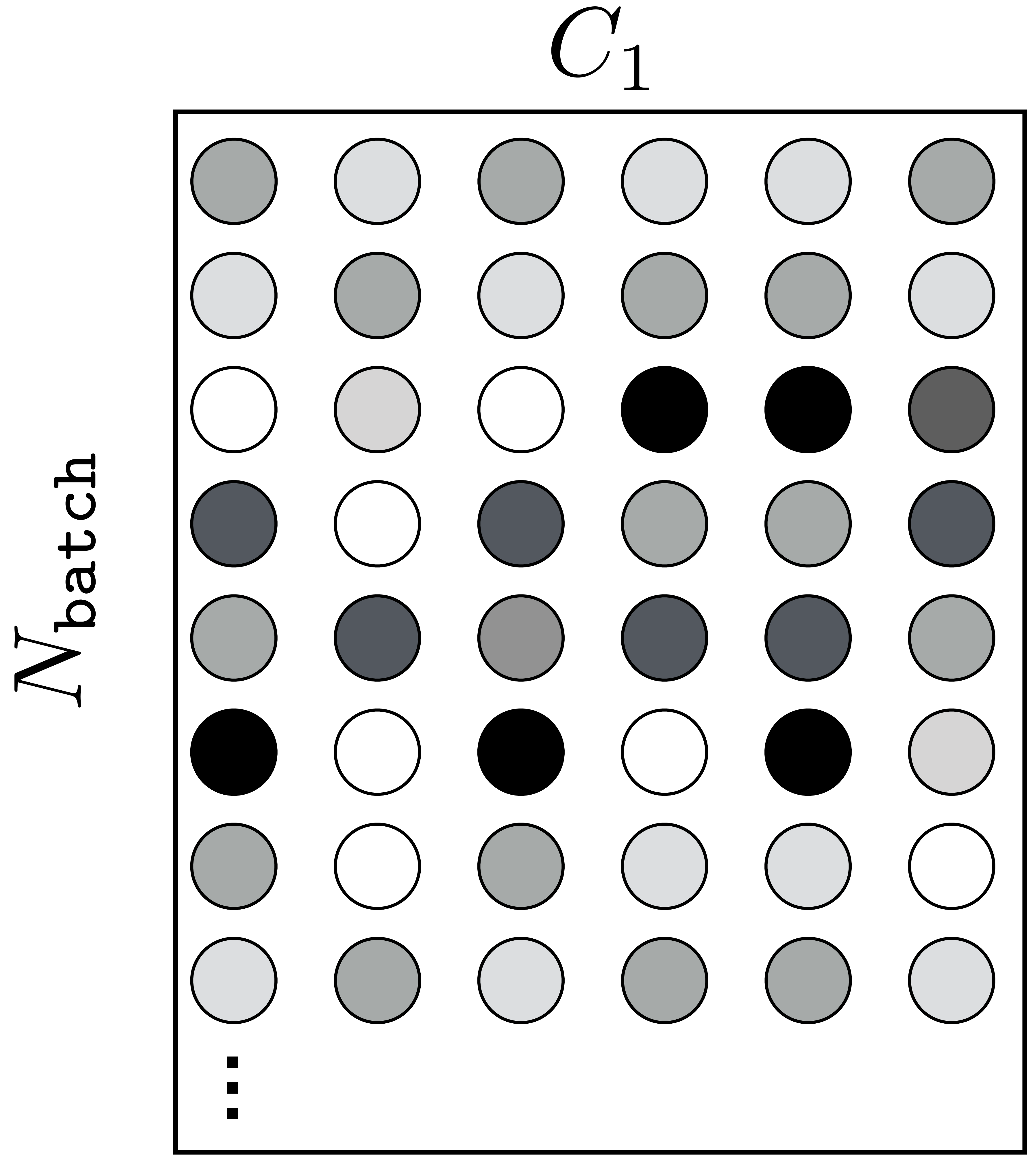
Each layer is a representation of the data

Tensors

(multi-dimensional arrays)

$$\mathbf{h}_1 \in \mathbb{R}^{N_{\texttt{batch}} \times C_1}$$

neurons
features
units
“channels”

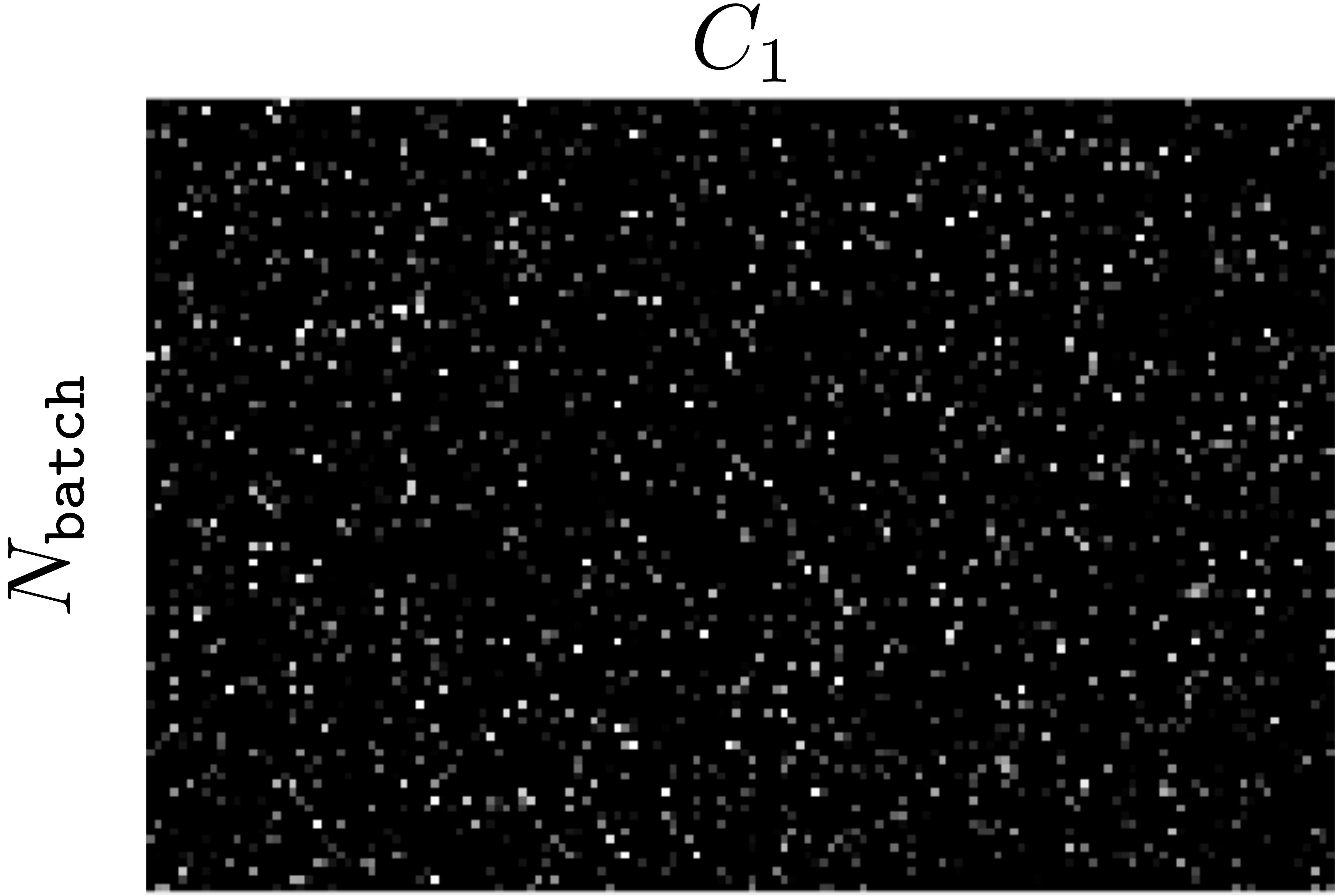


Tensors

(multi-dimensional arrays)

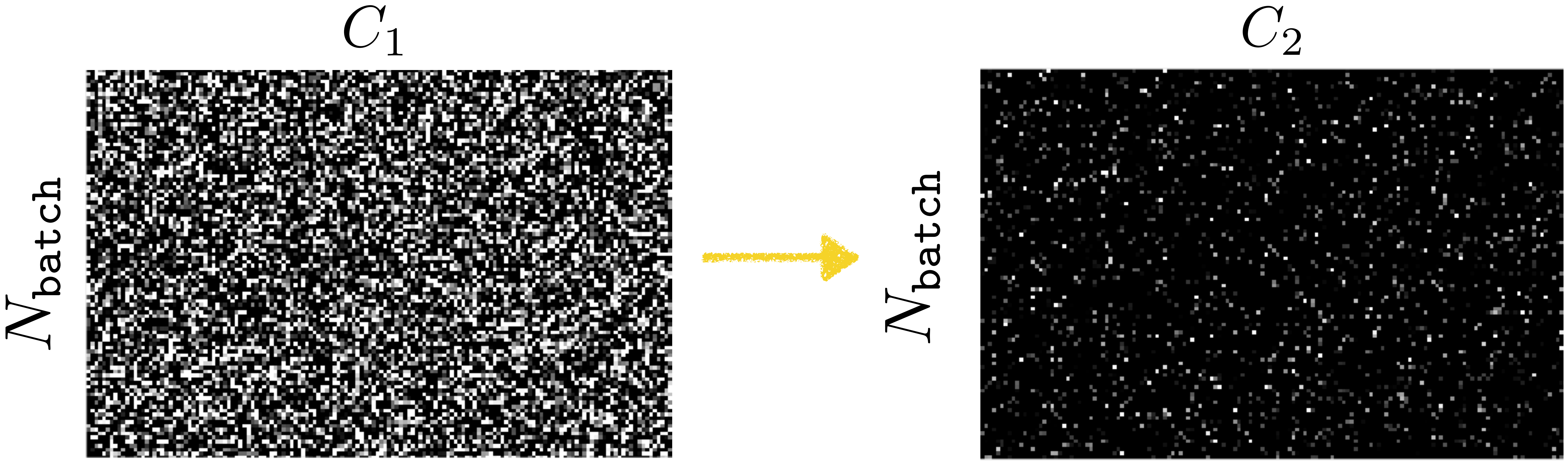
$$\mathbf{h}_1 \in \mathbb{R}^{N_{\texttt{batch}} \times C_1}$$

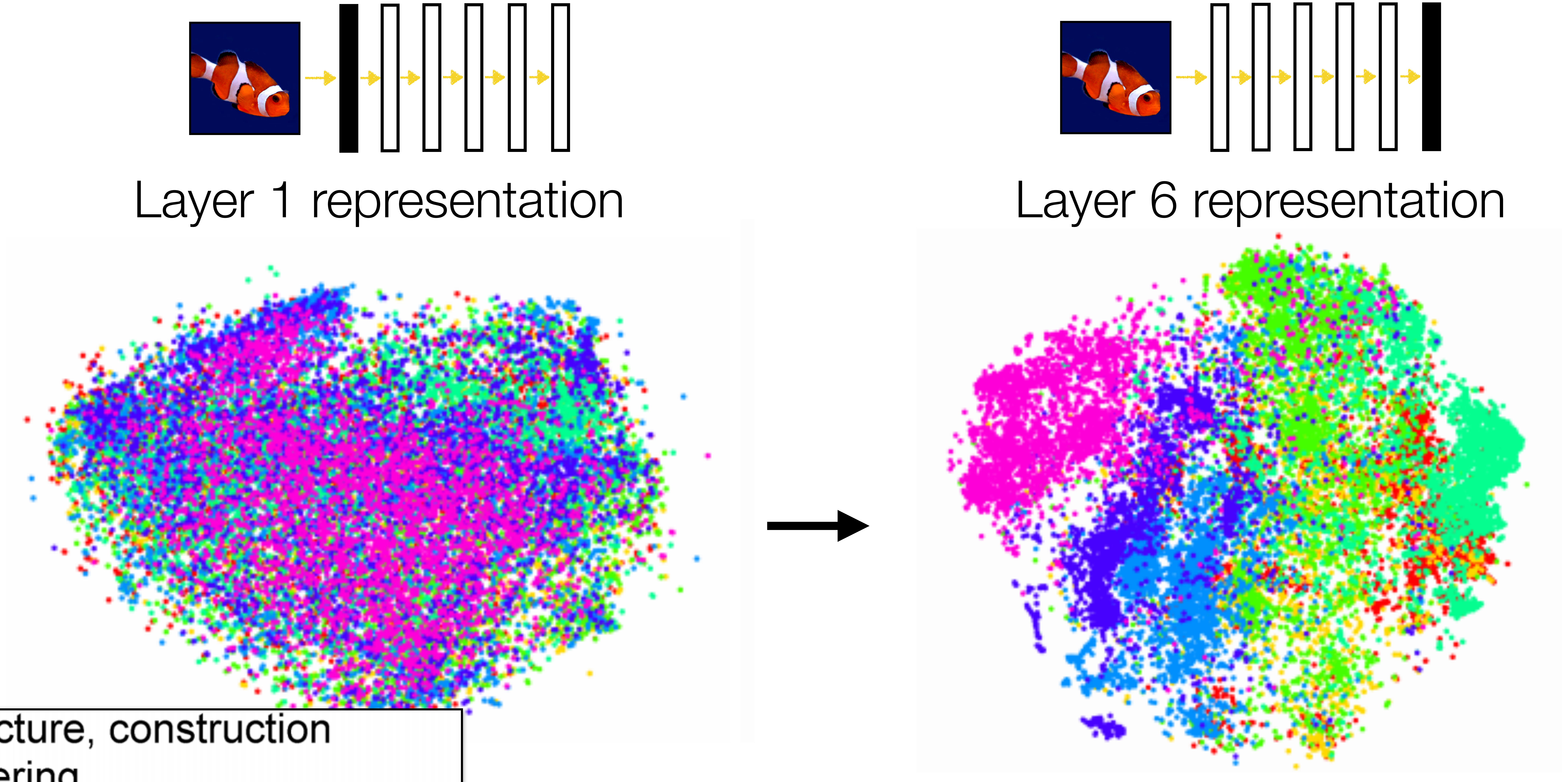
neurons
features
units
“channels”

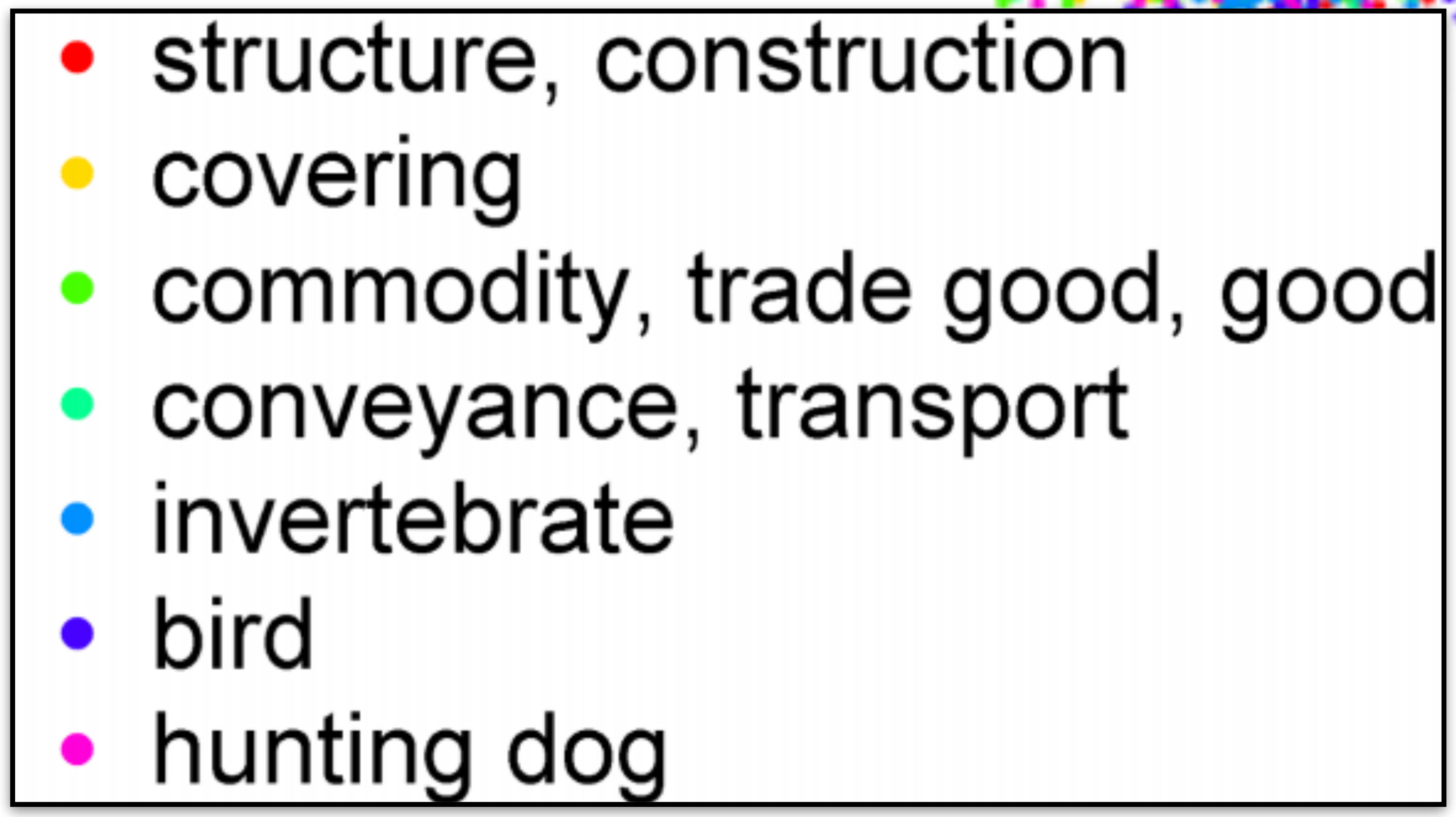


“Tensor flow”

$\mathbf{h}_1 \in \mathbb{R}^{N_{\texttt{batch}} \times C_1}$ $\mathbf{h}_2 \in \mathbb{R}^{N_{\texttt{batch}} \times C_2}$







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

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

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 regularize to prevent the network from overfitting?

1. Fewer neurons, fewer layers
2. Weight decay
3. Dropout
4. Normalization layers
5. …

Recall: regularized least squares

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

$$R(\theta) = \lambda \left\|\theta\right\|_2^2$$ ⟵ Only use polynomial terms if you really need them! Most terms should be zero

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

Probabilistic interpretation: R is a Gaussian **prior** over values of the parameters.

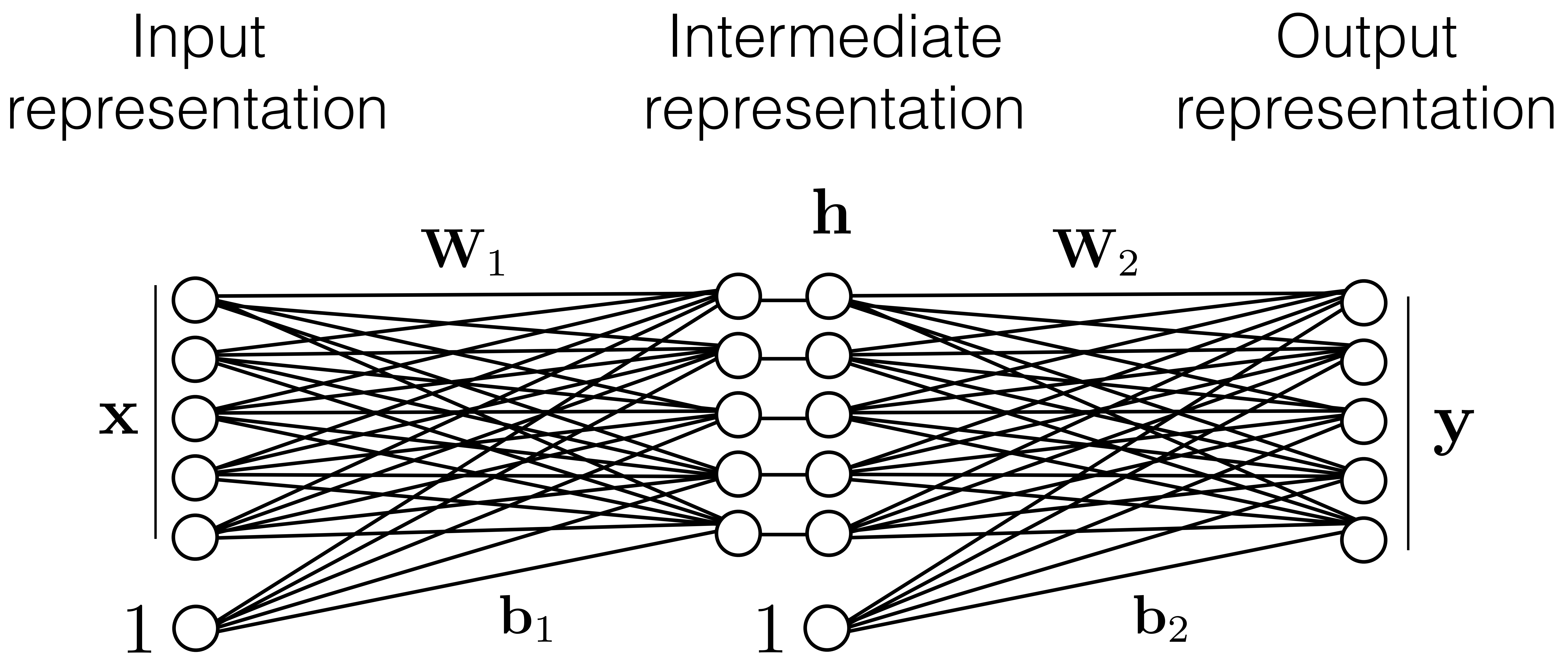
Regularizing the weights in a neural net

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

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

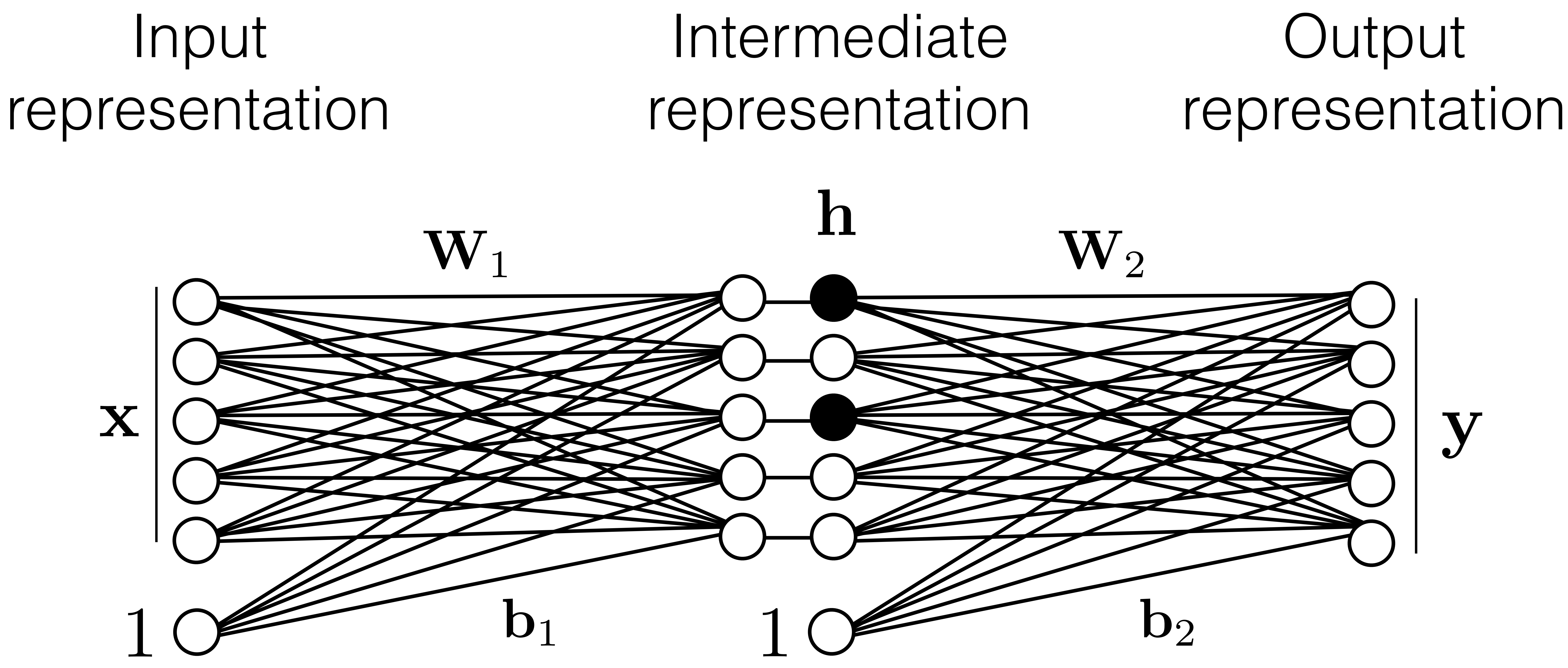
"We prefer to keep weights small."

Dropout

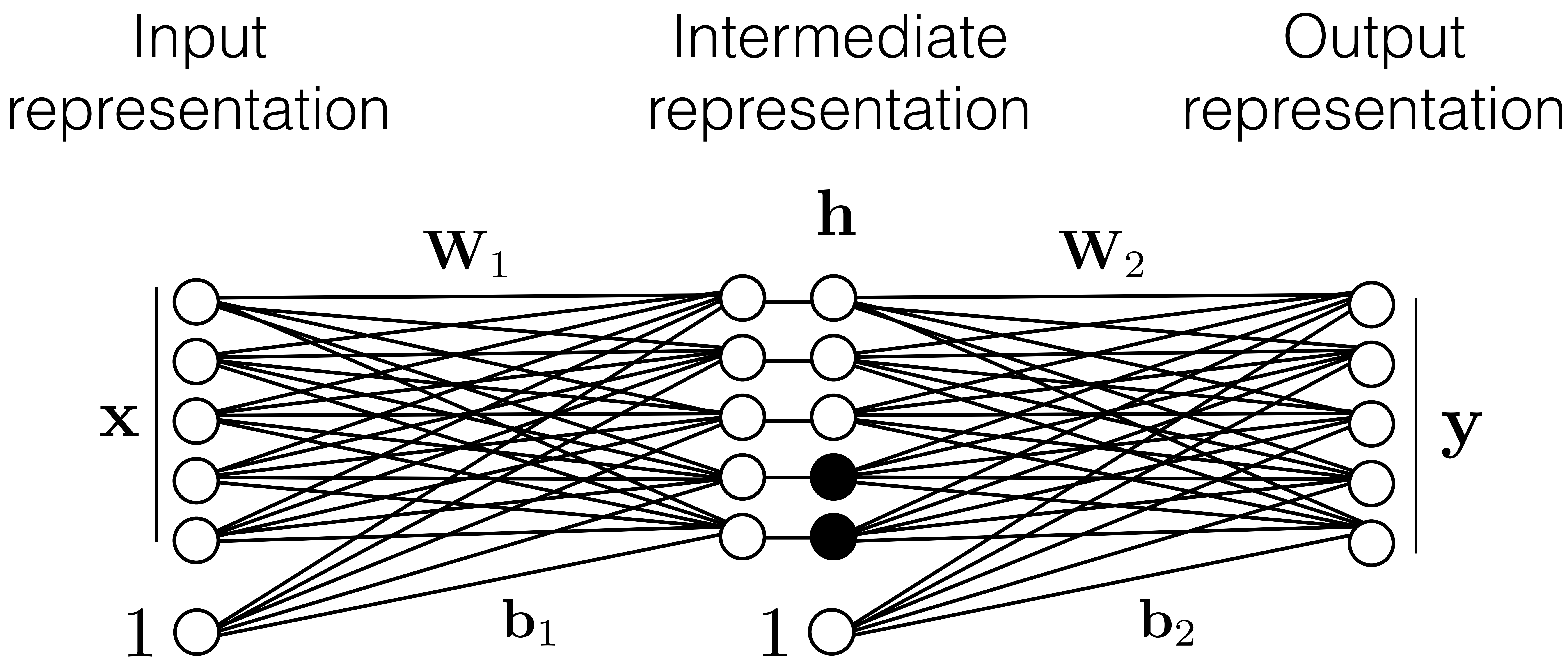


Input
representation
Intermediate
representation
Output
representation
$\mathbf{W}_1$
$\mathbf{h}$
$\mathbf{W}_2$
$\mathbf{x}$
$\mathbf{y}$
1
$\mathbf{b}_1$
1
$\mathbf{b}_2$

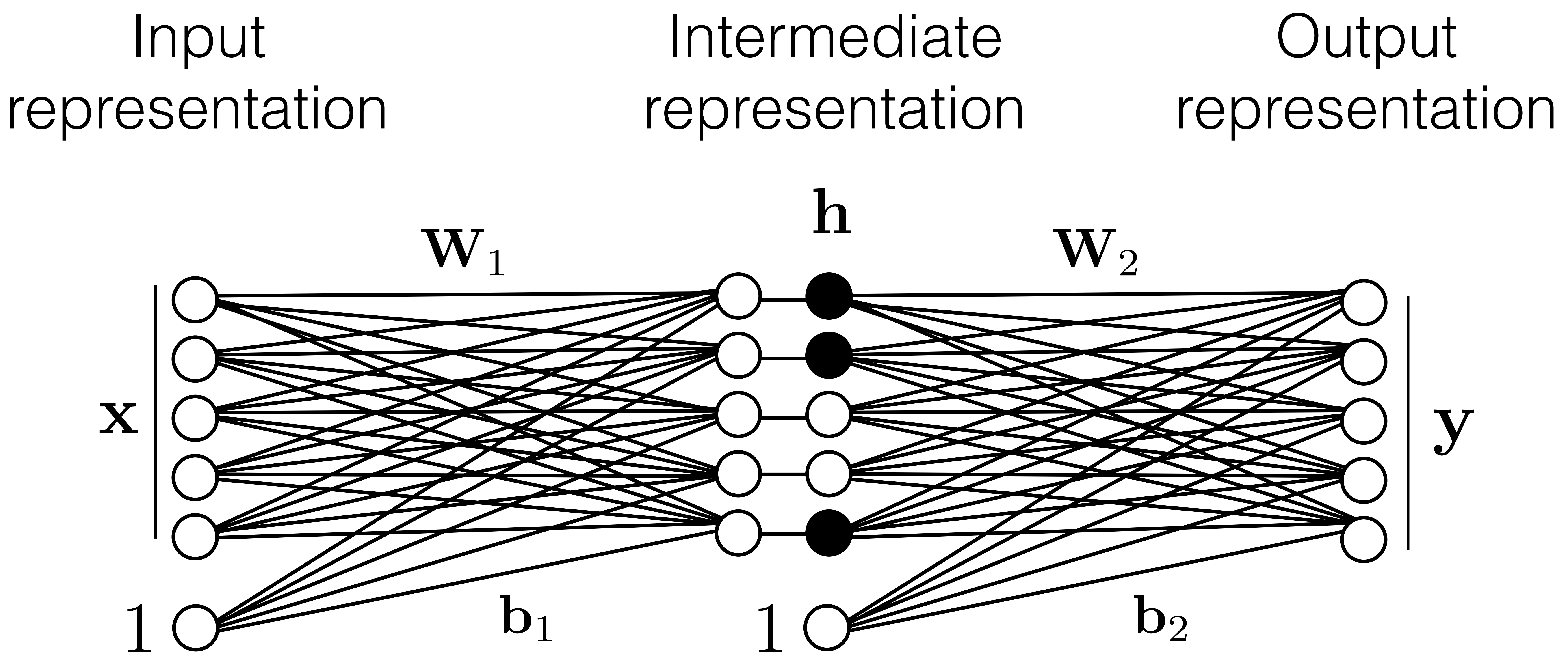
Dropout


Input
representation
Intermediate
representation
Output
representation
h
W1
W2
x
y
1
b1
1
b2

Dropout


Input
representation
Intermediate
representation
Output
representation
h
$\mathbf{W}_1$
$\mathbf{W}_2$
x
y
1
$\mathbf{b}_1$
1
$\mathbf{b}_2$

Dropout


Input
representation
Intermediate
representation
Output
representation
$\mathbf{h}$
$\mathbf{W}_1$
$\mathbf{W}_2$
$\mathbf{x}$
$\mathbf{y}$
1
$\mathbf{b}_1$
1
$\mathbf{b}_2$

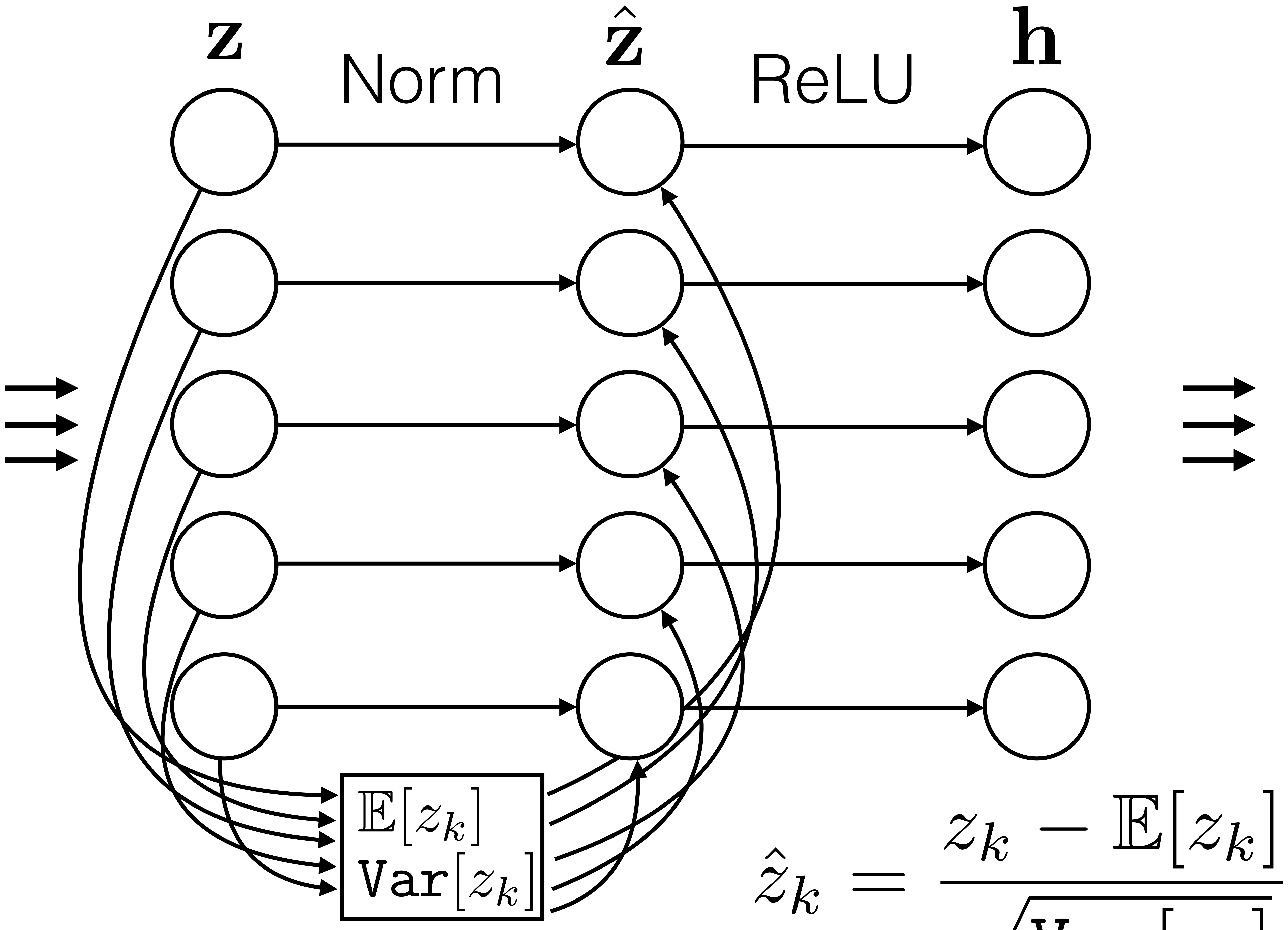
Dropout

Randomly zero out hidden units.

Prevents network from relying too much on spurious correlations between different hidden units.

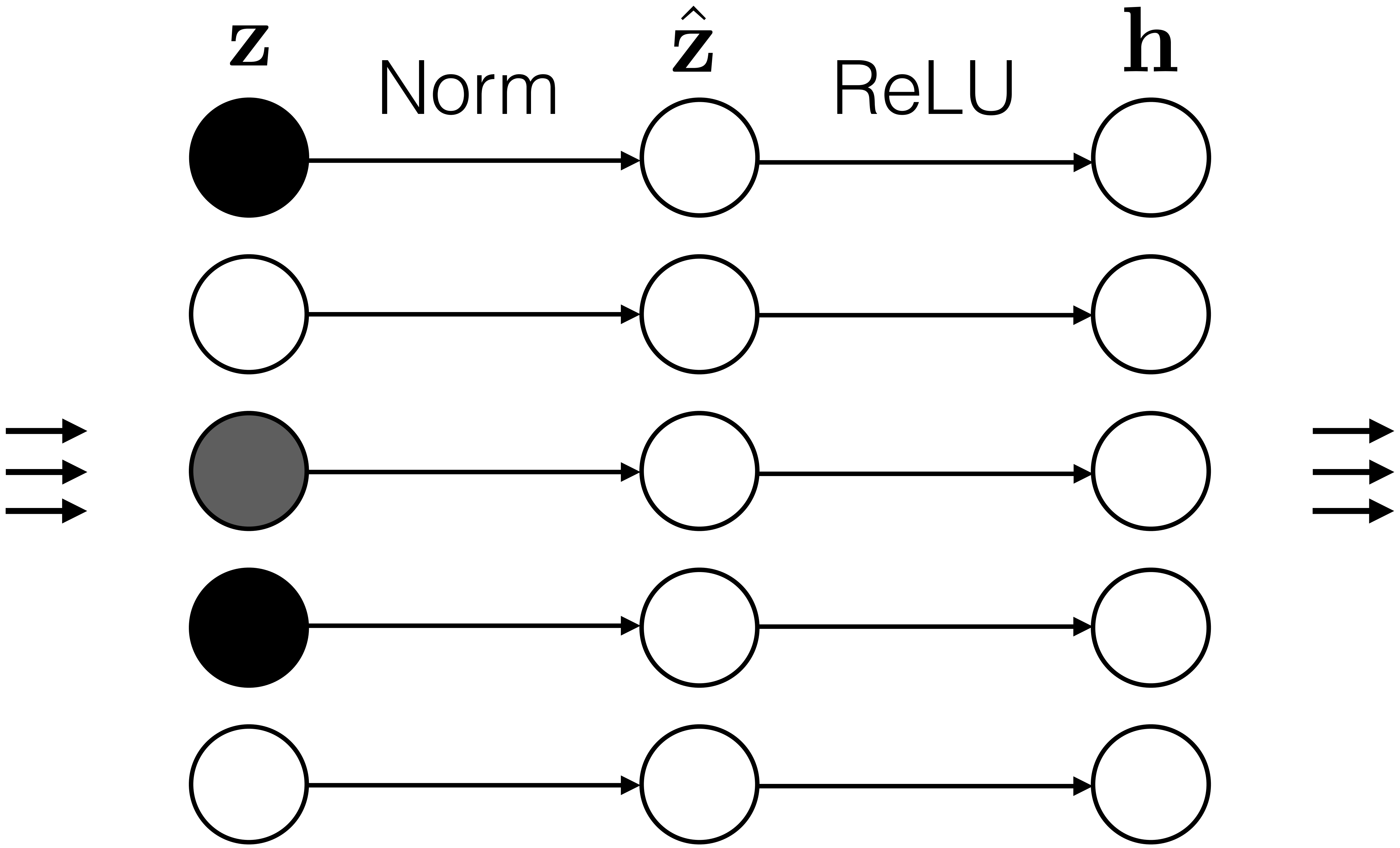
Can be understood as averaging over an exponential **ensemble** of subnetworks. This averaging smooths the function, thereby reducing the effective capacity of the network.

Normalization layers



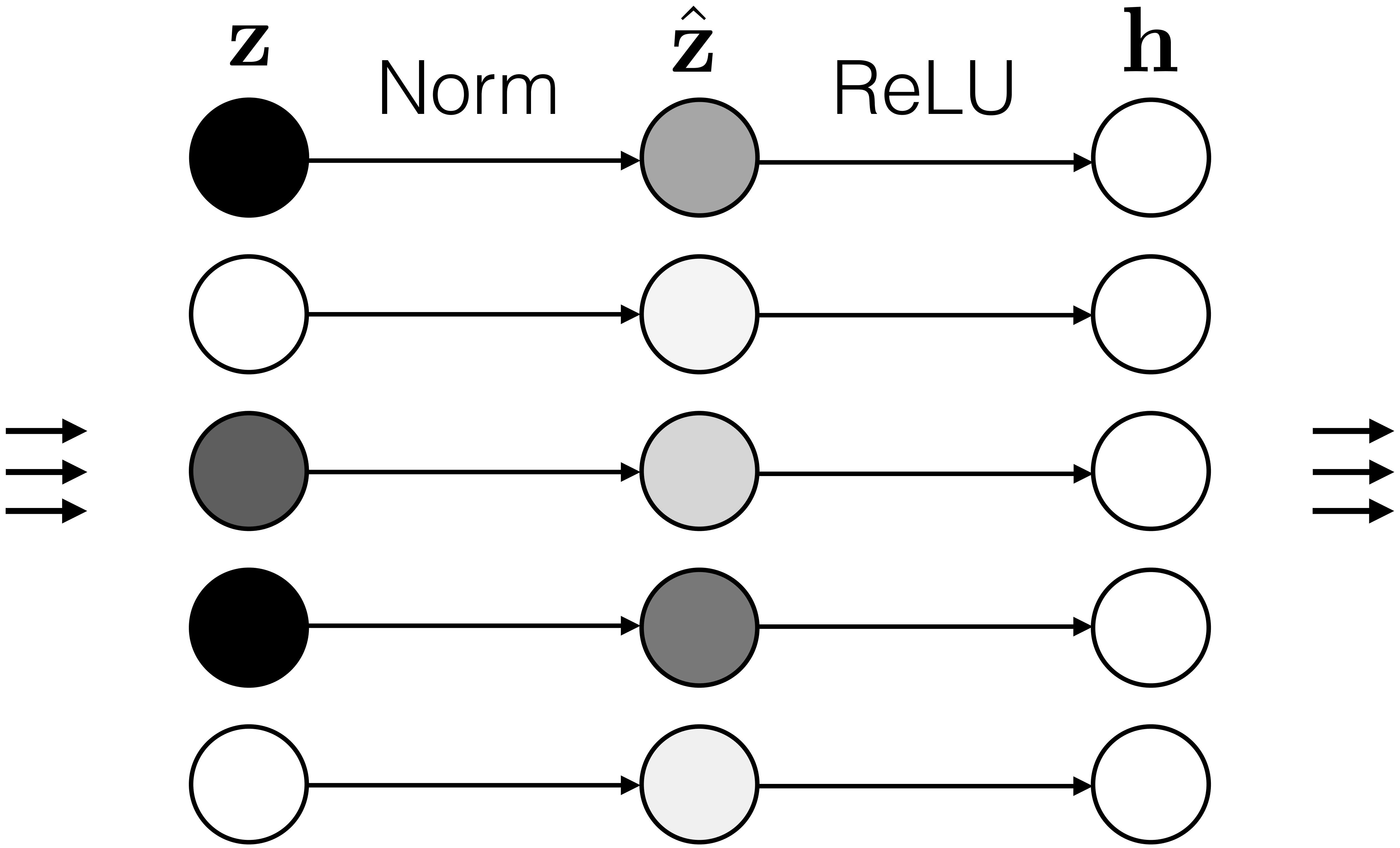
$$\hat{z}_k = \frac{z_k - \mathbb{E}[z_k]}{\sqrt{\mathtt{Var}[z_k]}}$$

Normalization layers



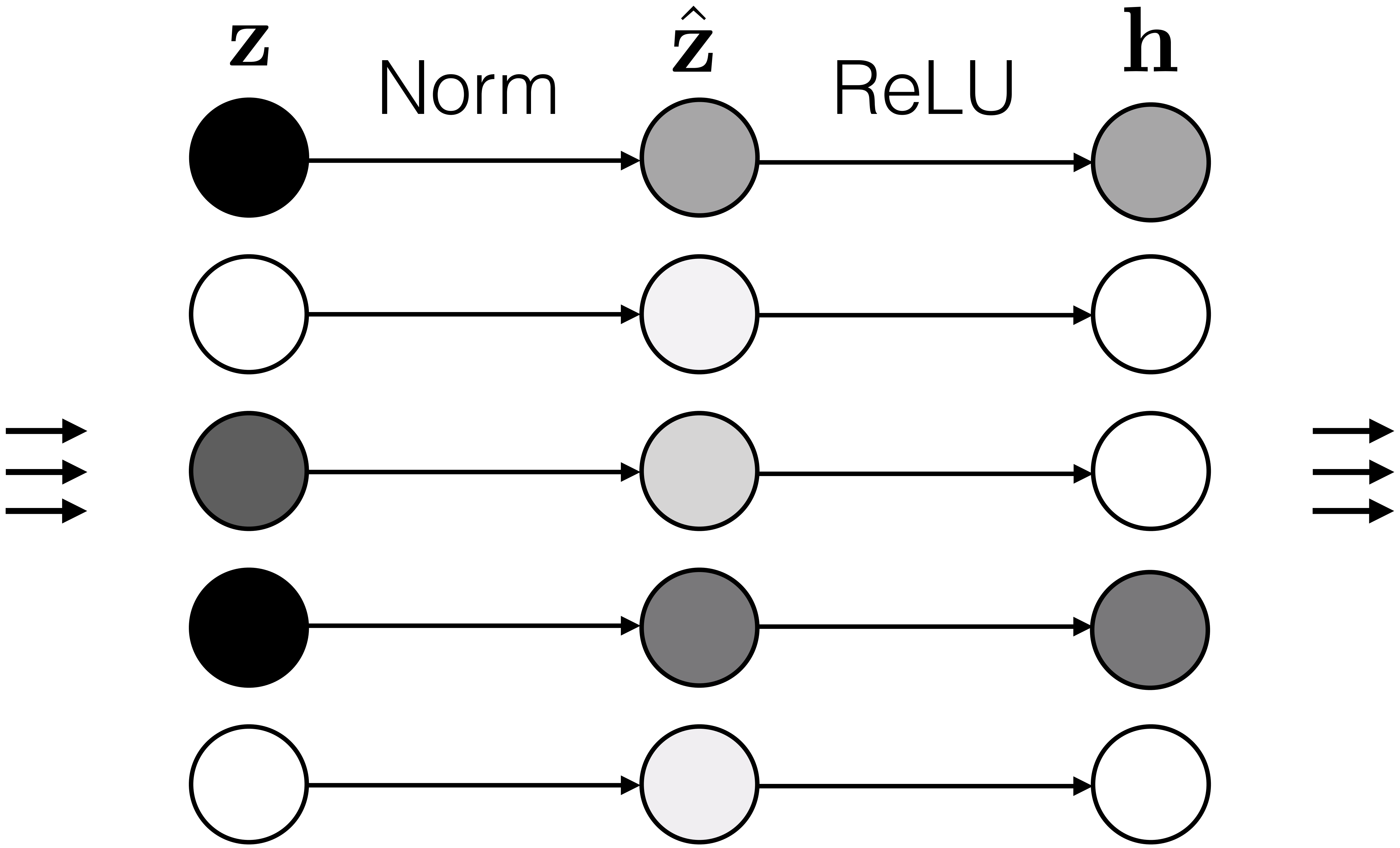
$$\hat{z}_k = \frac{z_k - \mathbb{E}[z_k]}{\sqrt{\texttt{Var}[z_k]}}$$

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.

9. Neural Networks

- Brief history
- Basic formulation *(hierarchical processing)*
- Optimization via gradient descent
- Layer types *(Linear, Pointwise non-linearity)*
- Linear classification with a perceptron
- Batch processing
- Regularizers
- Normalization