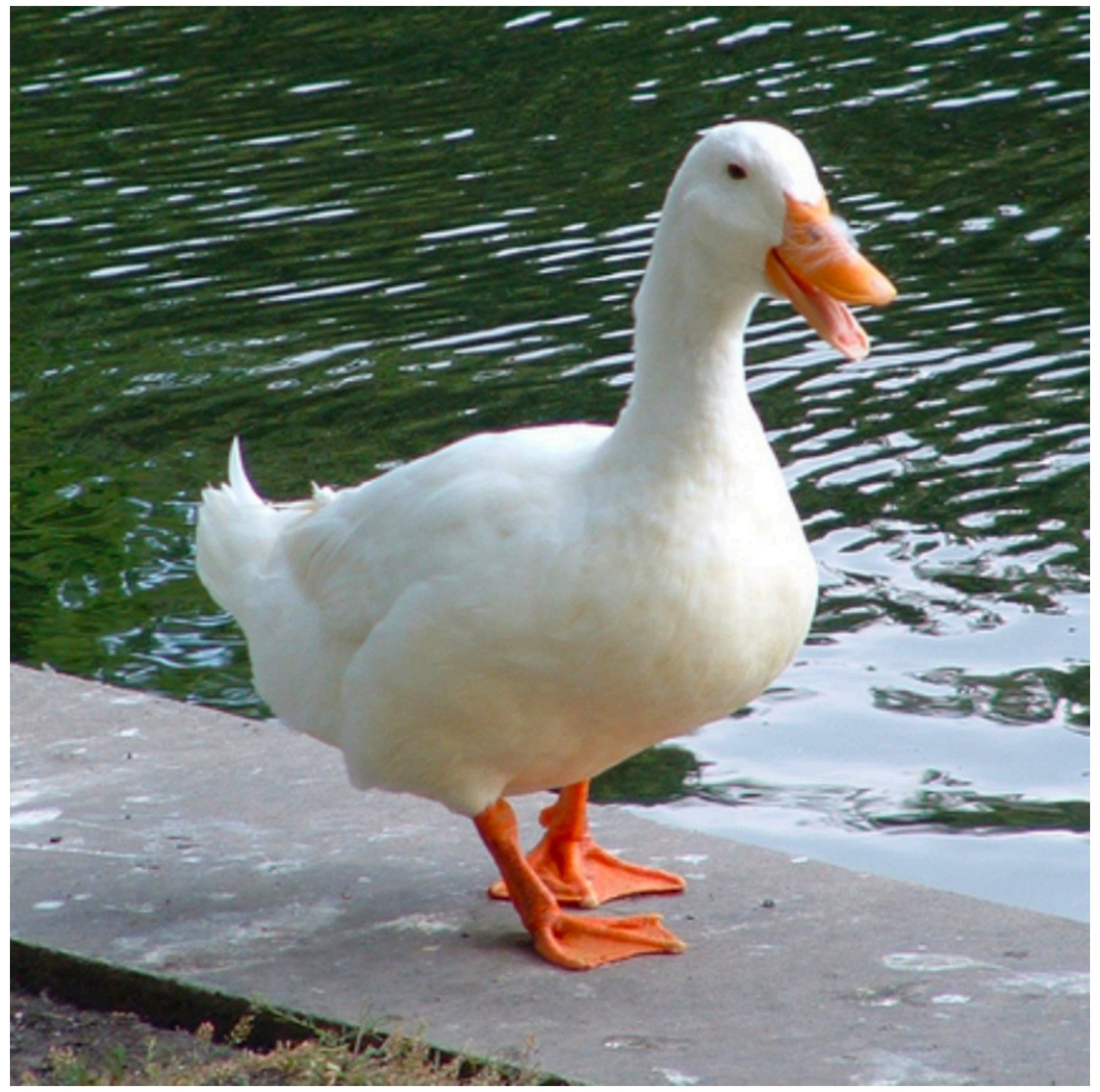
Lecture 6: Linear classifiers

CS 5670: Introduction to Computer Vision

Announcements

- PS1 due tonight
- PS2 out tonight
- PyTorch Colab notebook will be on website
  - Course staff can walk you through it during office hours.

Image classification with linear models



⋮

image **x**

Linear classifier

"Duck"

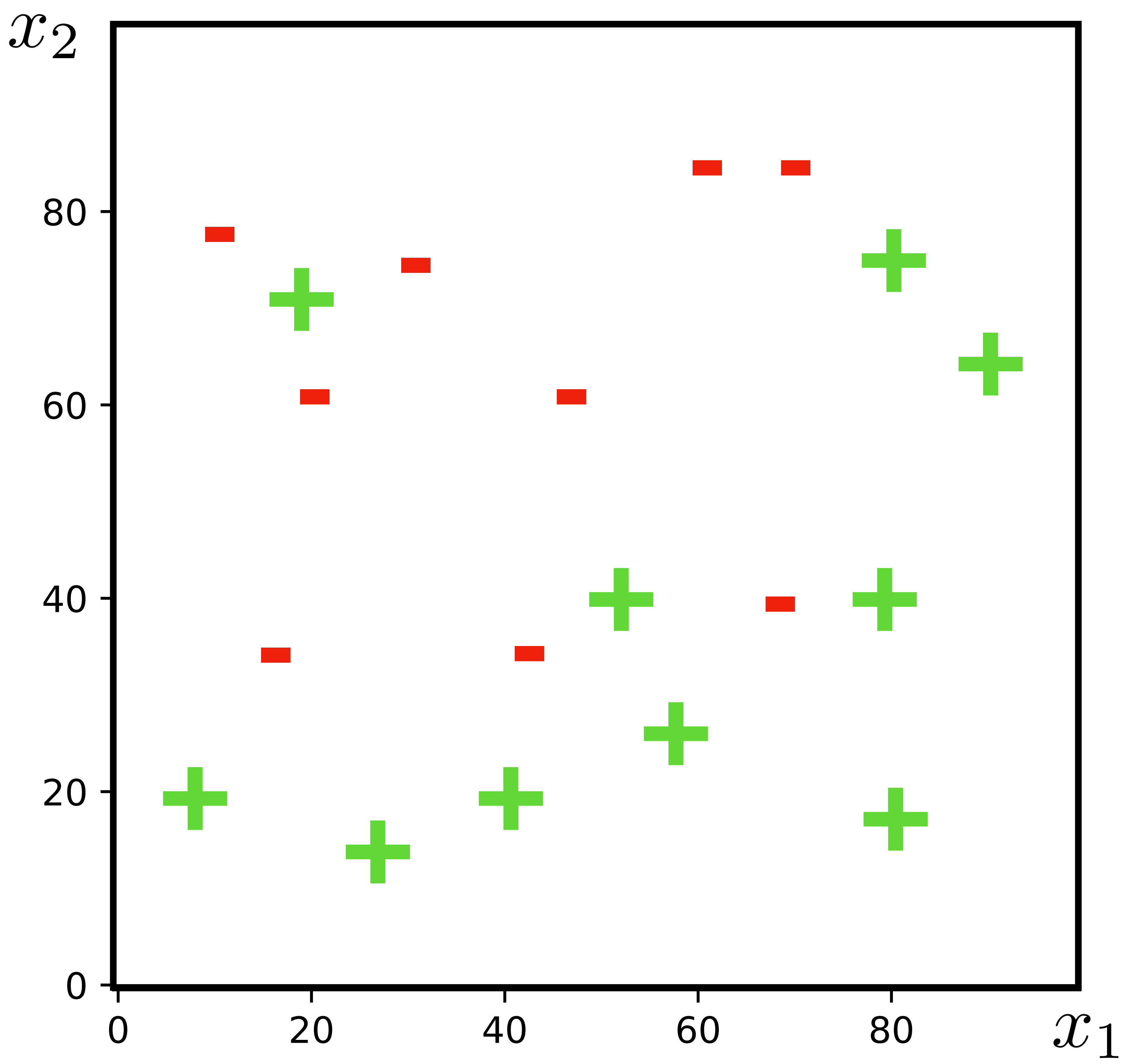
label y

Source: Isola, Torralba, Freeman

A geometric view

Linear decision boundaries

Consider a binary classification problem.

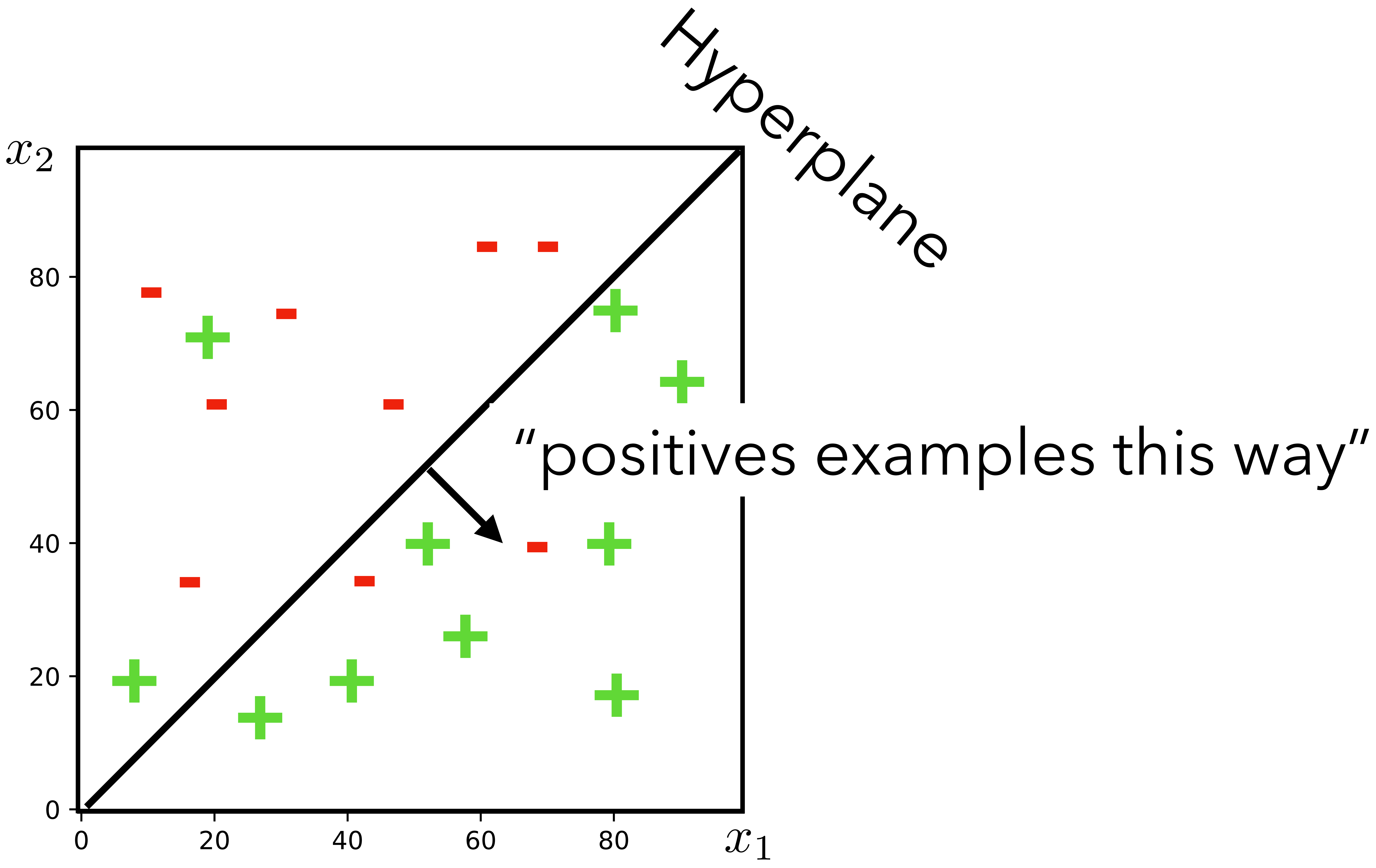


Training data

Example adapted from P. Isola

Linear decision boundaries

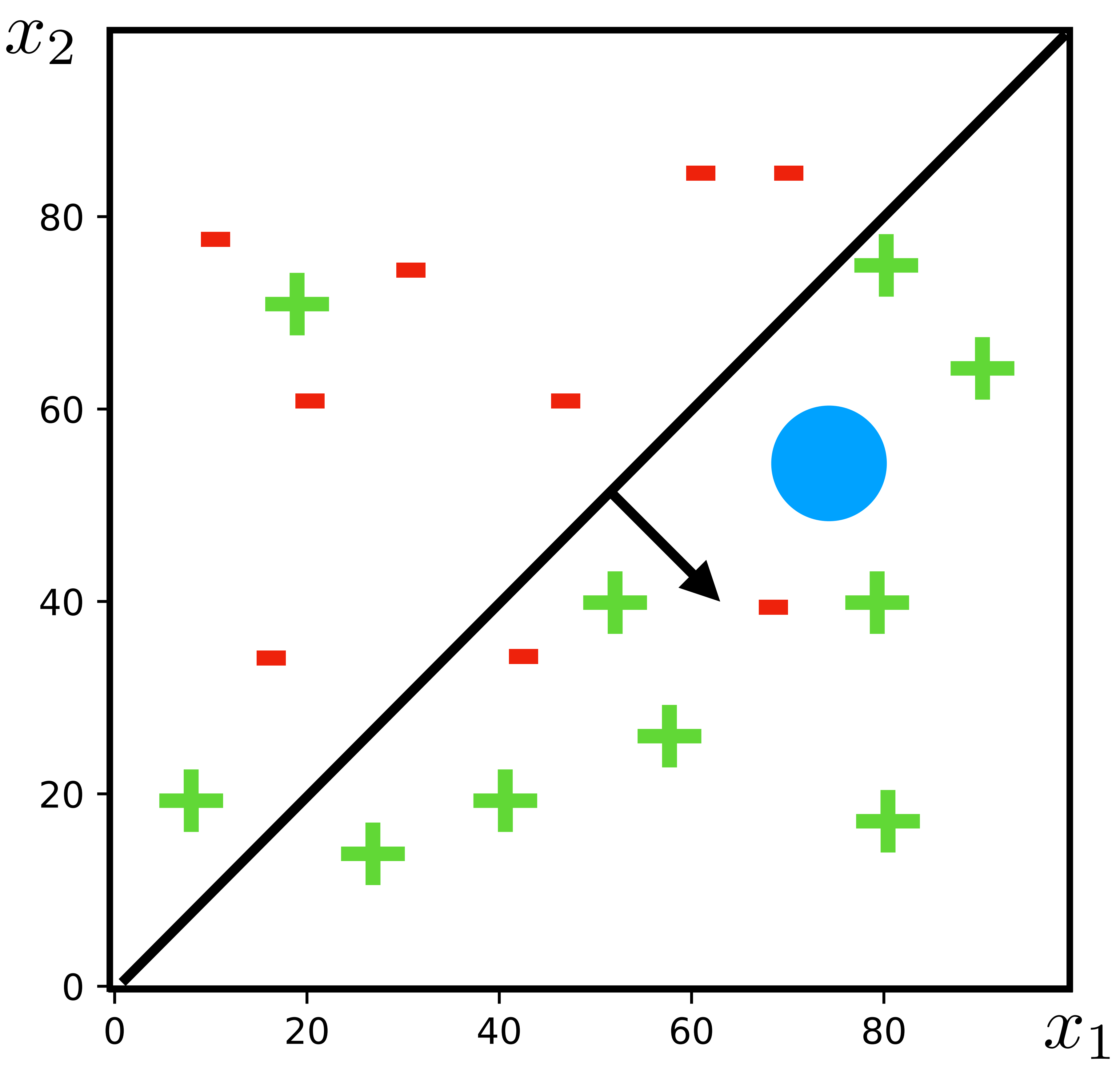
Consider a binary classification problem.



Training data

Example adapted from P. Isola

Linear decision boundaries



"What label is point?"

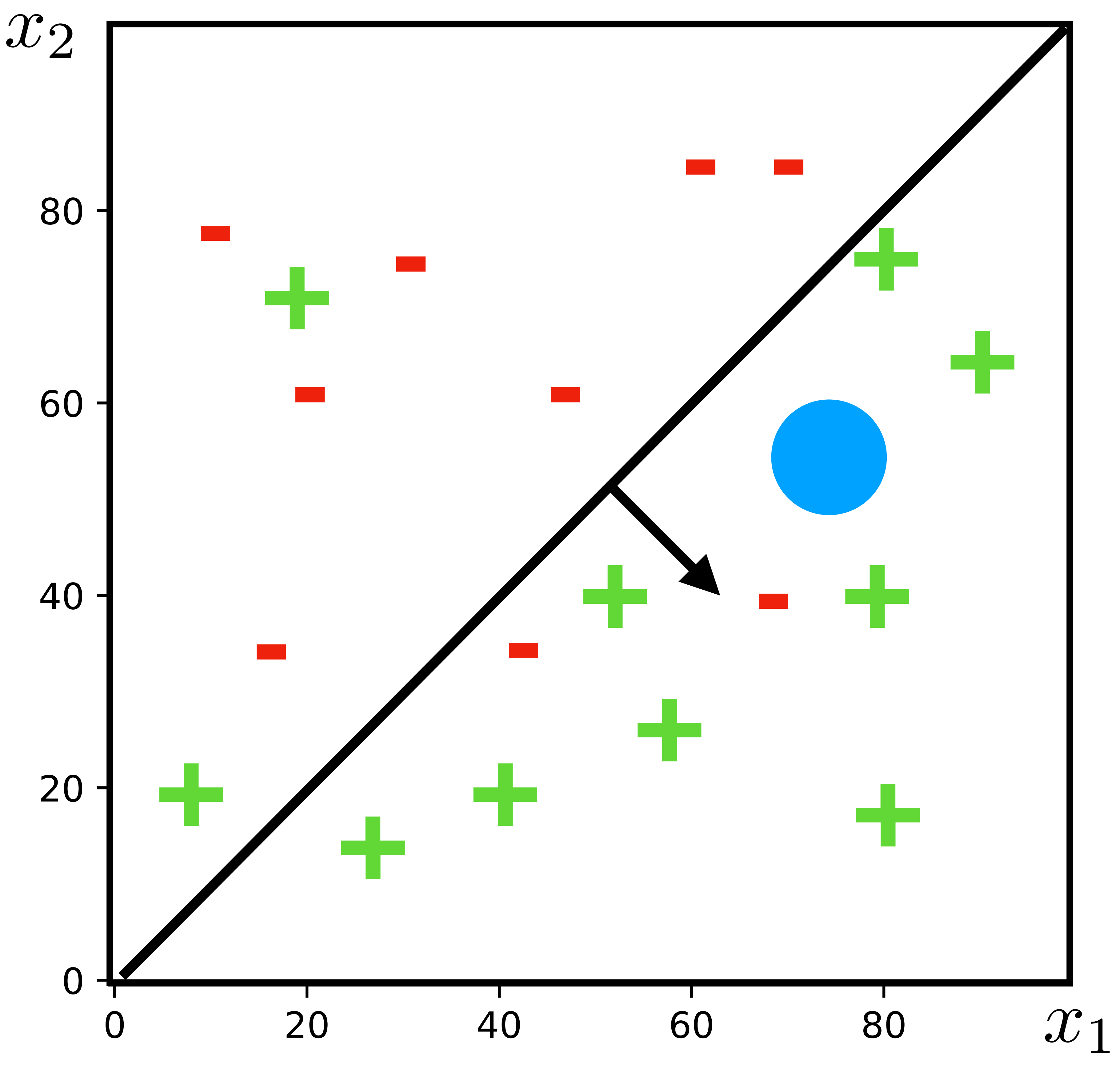
Which side of the hyperplane is x on?

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

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

Example adapted from P. Isola

Linear decision boundaries



"What label is point?"

Notational simplification:

Can get rid of b by "tacking on" a 1 to x

$$\hat{y} = \tilde{\mathbf{x}}^\top \tilde{\mathbf{w}}$$

$$\tilde{x} = \begin{bmatrix} x \\ 1 \end{bmatrix} \qquad \tilde{w} = \begin{bmatrix} w \\ b \end{bmatrix}$$

Multiway classification

For a *k*-class problem, we'll make a matrix "stacking" k hyperplanes as rows of a matrix $W \in \mathbb{R}^{k \times d}$:

$$W = \begin{bmatrix} \text{———} & w_1 & \text{———} \\ \text{———} & w_2 & \text{———} \\ & \ldots & \\ \text{———} & w_k & \text{———} \end{bmatrix}$$

To classify an example: which row has the highest dot product with x?

$$z = Wx + b$$

$$g(\hat{y}) = \text{argmax}_k \, z_k$$

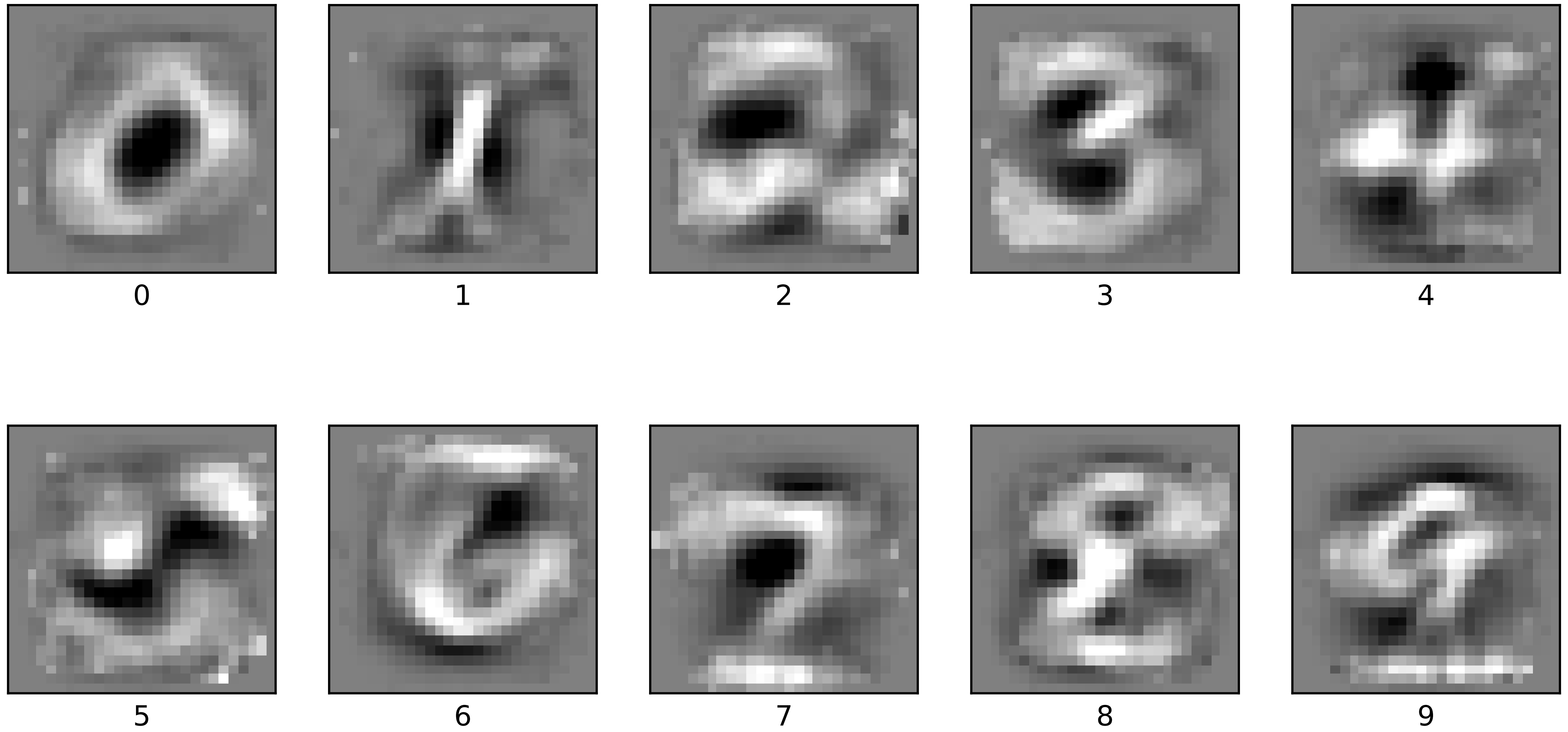
Example: handwritten digits

$$z = Wx + b$$

$$g(\hat{y}) = \operatorname{argmax}_k z_k$$

$$W = \begin{bmatrix} \text{——} & w_1 & \text{——} \\ \text{——} & w_2 & \text{——} \\ & \cdots & \\ \text{——} & w_k & \text{——} \end{bmatrix}$$

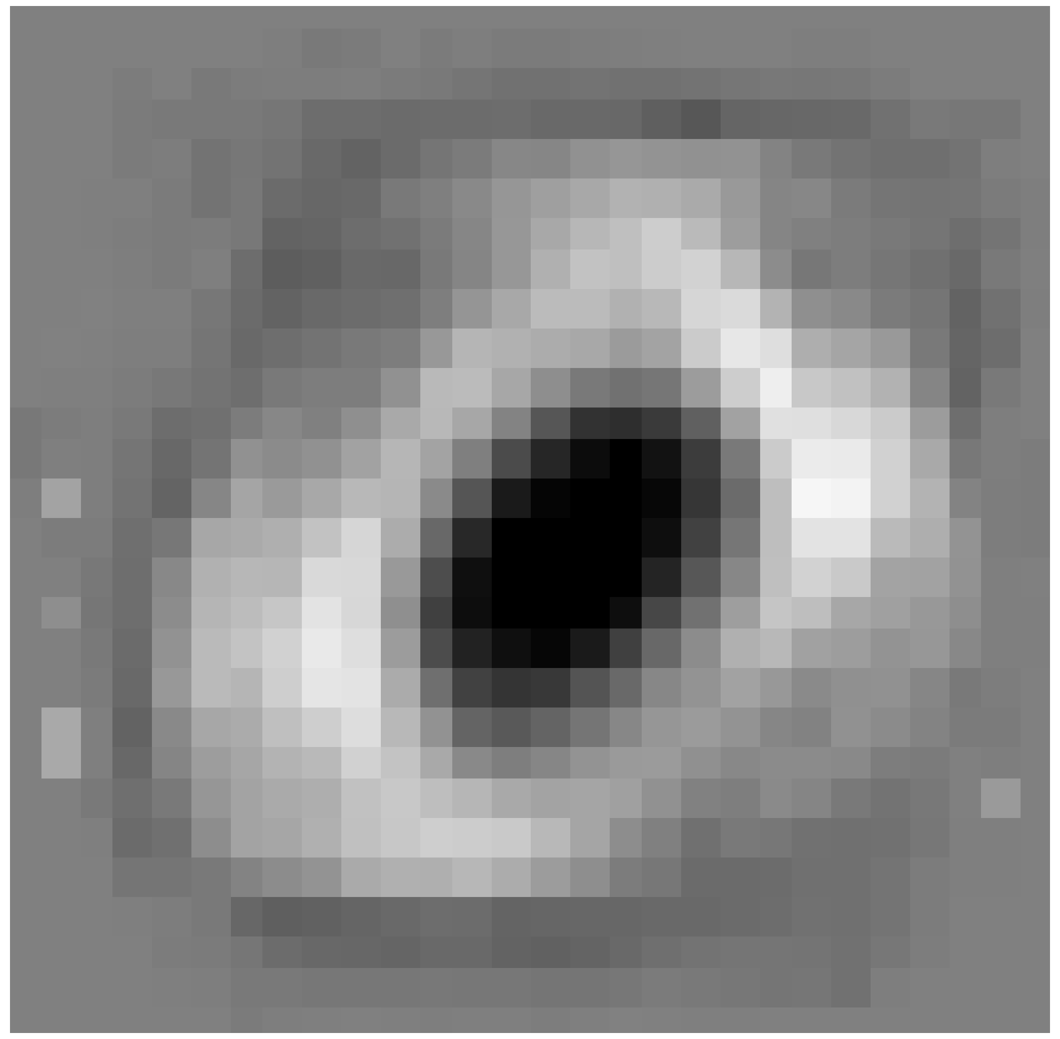
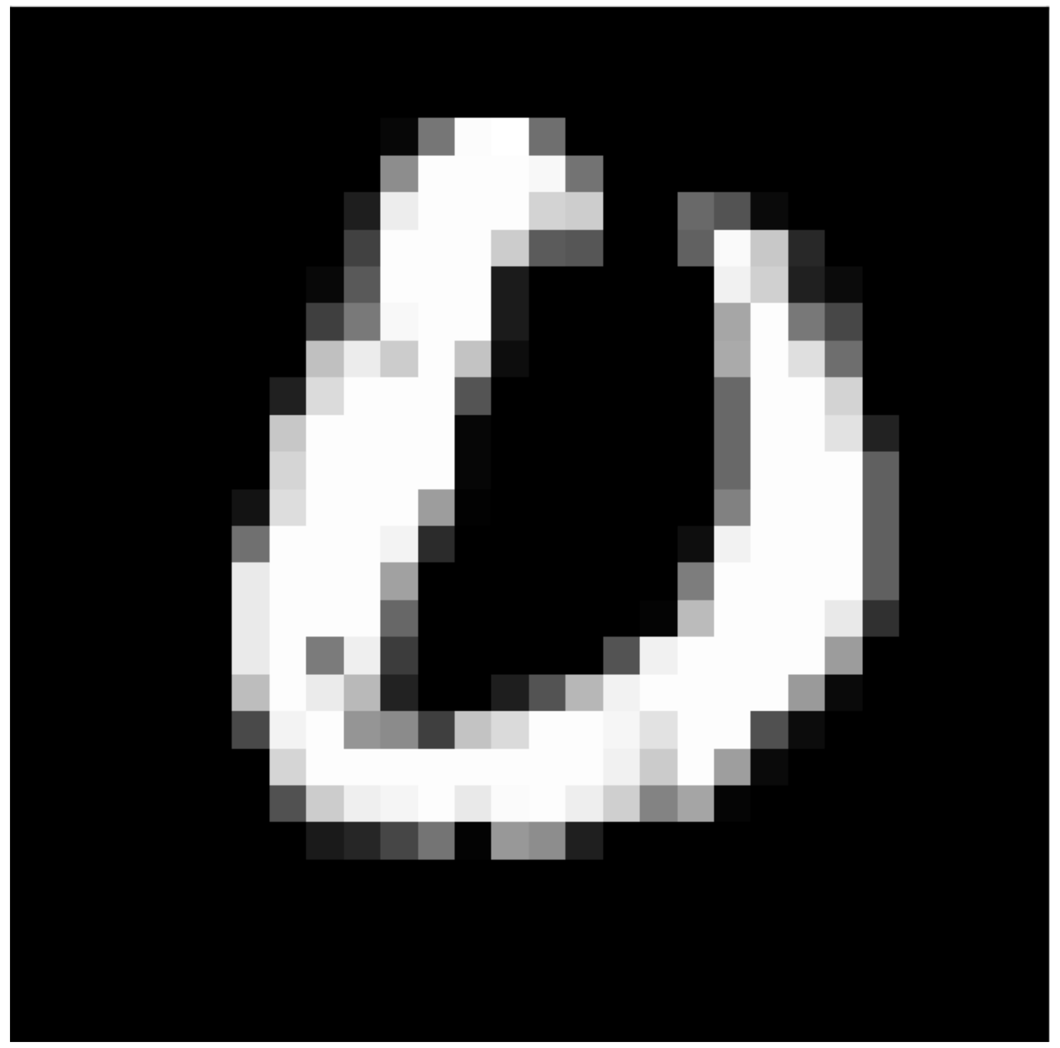
inputs



example of learned weights

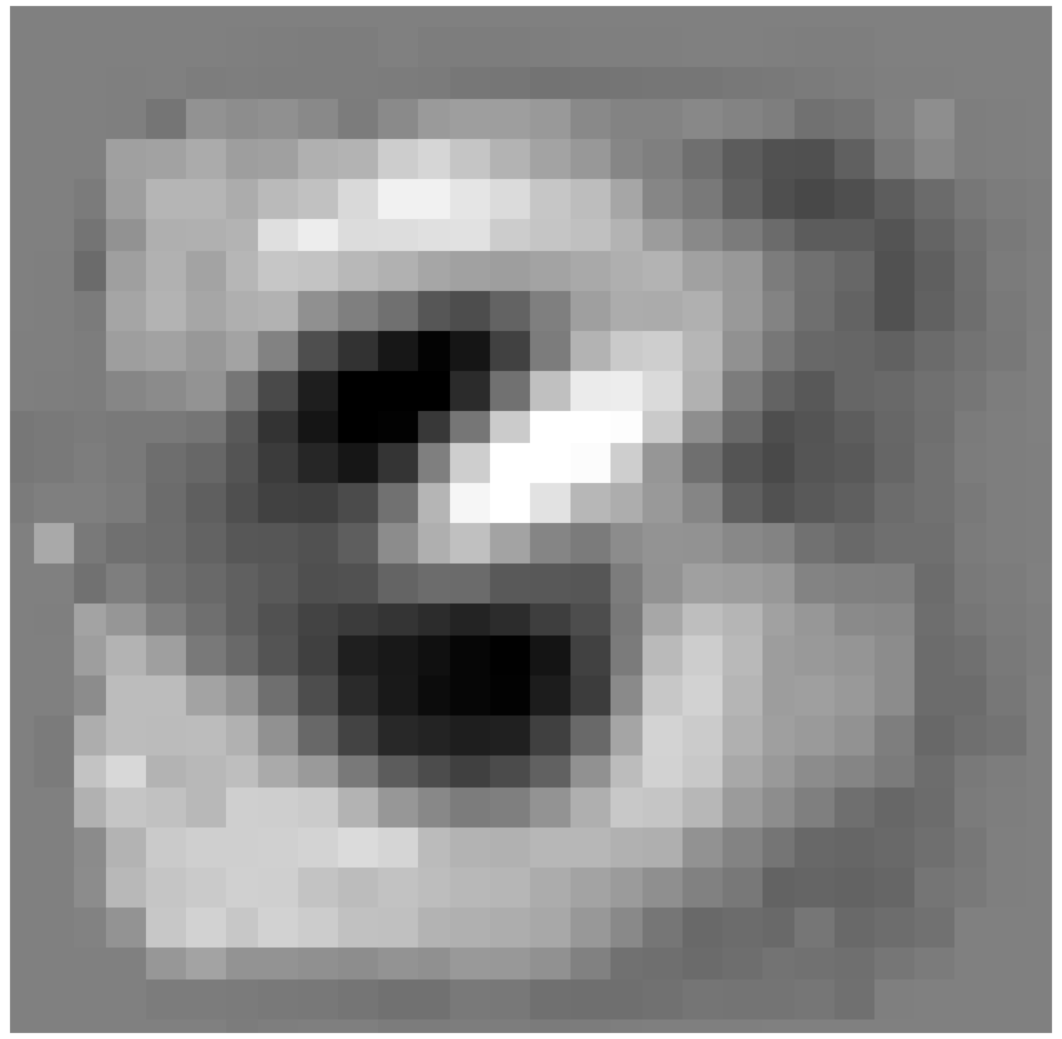
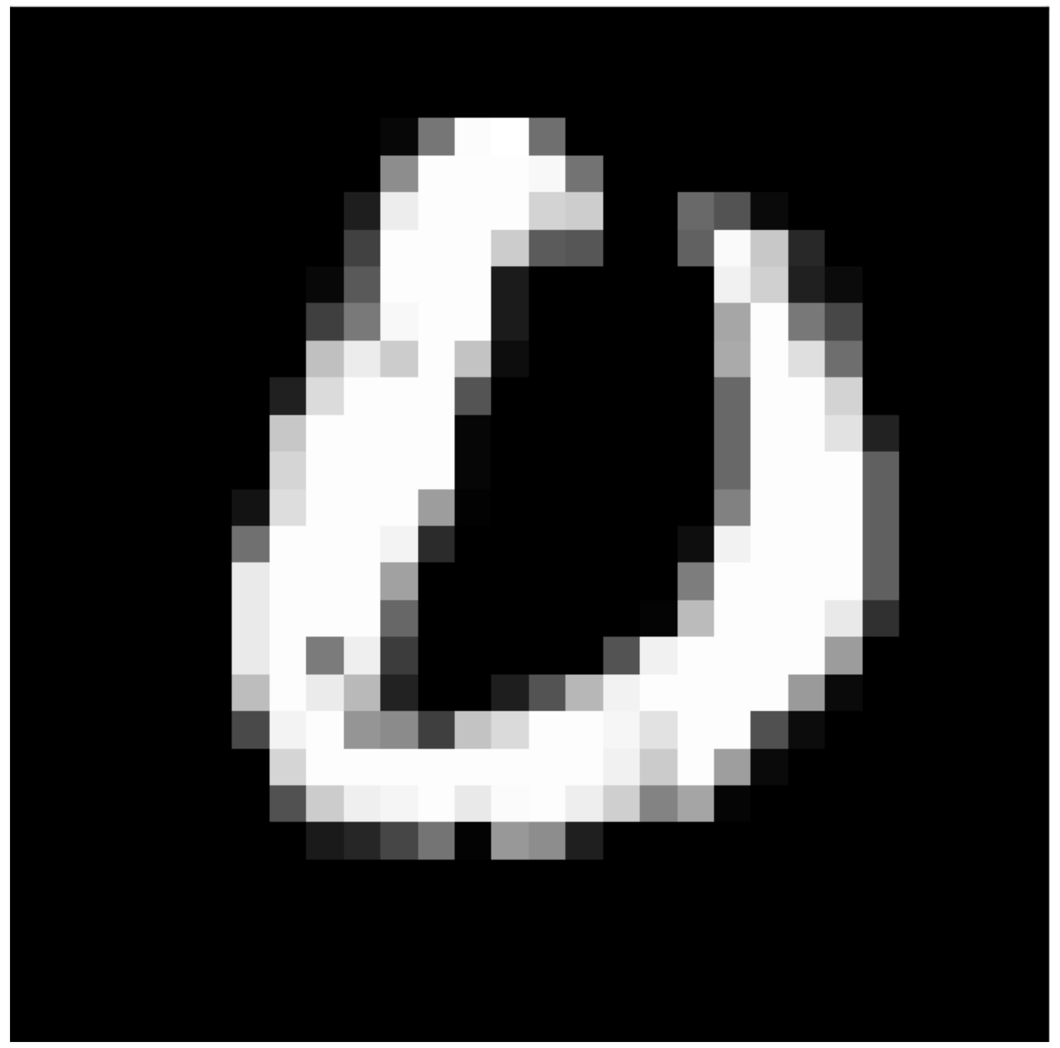
Visualization adapted from scikit-learn.

Example: handwritten digits

 $\bullet$  = high number

w_0 x

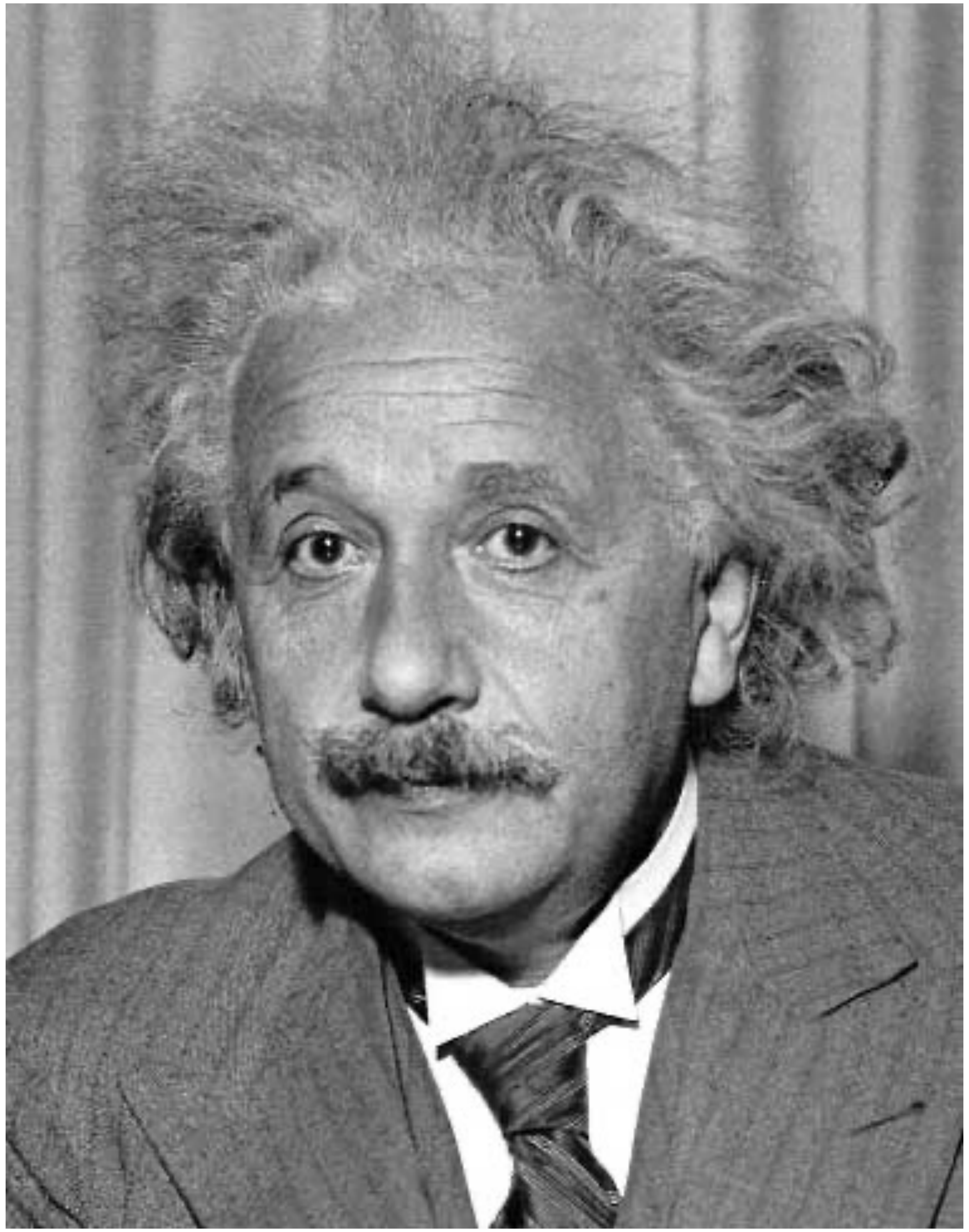
Example: handwritten digits

 $\bullet$  = low number

w_3 x

Recall: image patches as filters

Find  in an image



Source: D. Hoiem

Converting to probabilities

We have: $z = Wx + b$

We want probabilistic predictions: $p(y_k | x)$

i.e. $\sum_k p(y_k | x) = 1$, and $p(y_k | x) \geq 0$

Solution: use the *softmax* function:

$$\hat{y} = \text{softmax}(Wx + b)$$
$$p(y_k | x) = \hat{y}_k$$

non-negative

sums-to-1

$$\text{softmax}(z)_k = \frac{\exp(z_k)}{\sum_i \exp(z_i)}$$

$\mathbf{x}$

f

y

"Fish"

$$\arg\min_{f \in \mathcal{F}} \sum_{i=1}^{N} \mathcal{L}(f(\mathbf{x}_i), y_i)$$

Source: Isola, Torralba, Freeman

Recall: one-hot vectors

Source: Isola, Torralba, Freeman

Recall: loss function

0-1 loss: number of misclassifications

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = 1 - \mathbb{1}(\hat{\mathbf{y}}, \mathbf{y}) \longleftarrow \text{number of misclassifications}$$

Least squares: predict 1 for true class, 0 for others

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{k=1}^{K} (y_k - \hat{y}_k)^2$$

Cross entropy: a good surrogate that we'll be able to optimize:

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

Adapted from P. Isola

Loss function

Simpler than it looks:

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

Therefore:

One-hot vectors

Training data

$\mathbf{x}$ $\mathbf{y}$

{  , [0,0,1] }

{  , [0,1,0] }

{  , [1,0,0] }

Adapted from P. Isola

$\mathbf{x}$

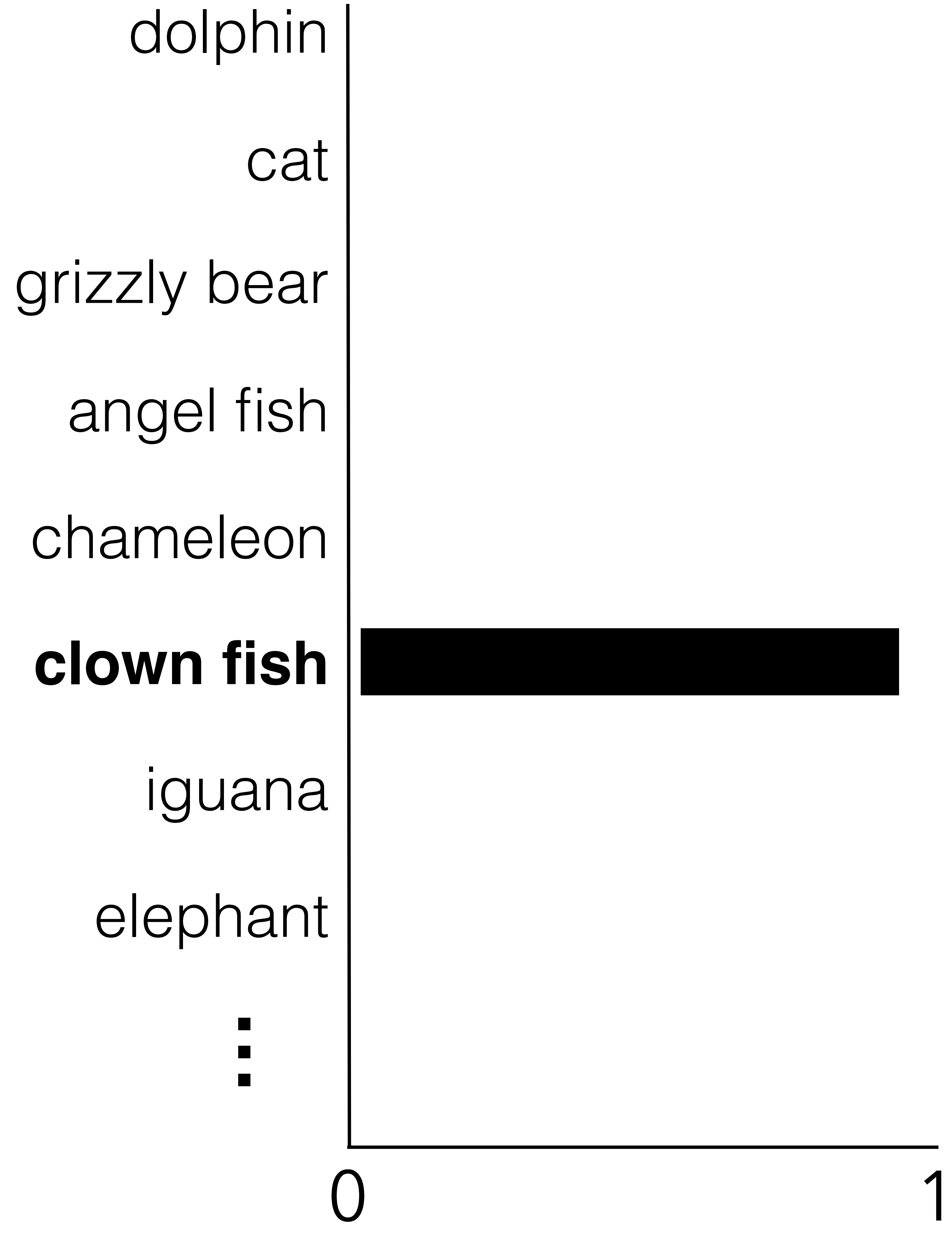
Ground truth label $\mathbf{y}$

[0,0,0,0,0,1,0,0,…]

Source: Isola, Torralba, Freeman

Ground truth label $\mathbf{y}$

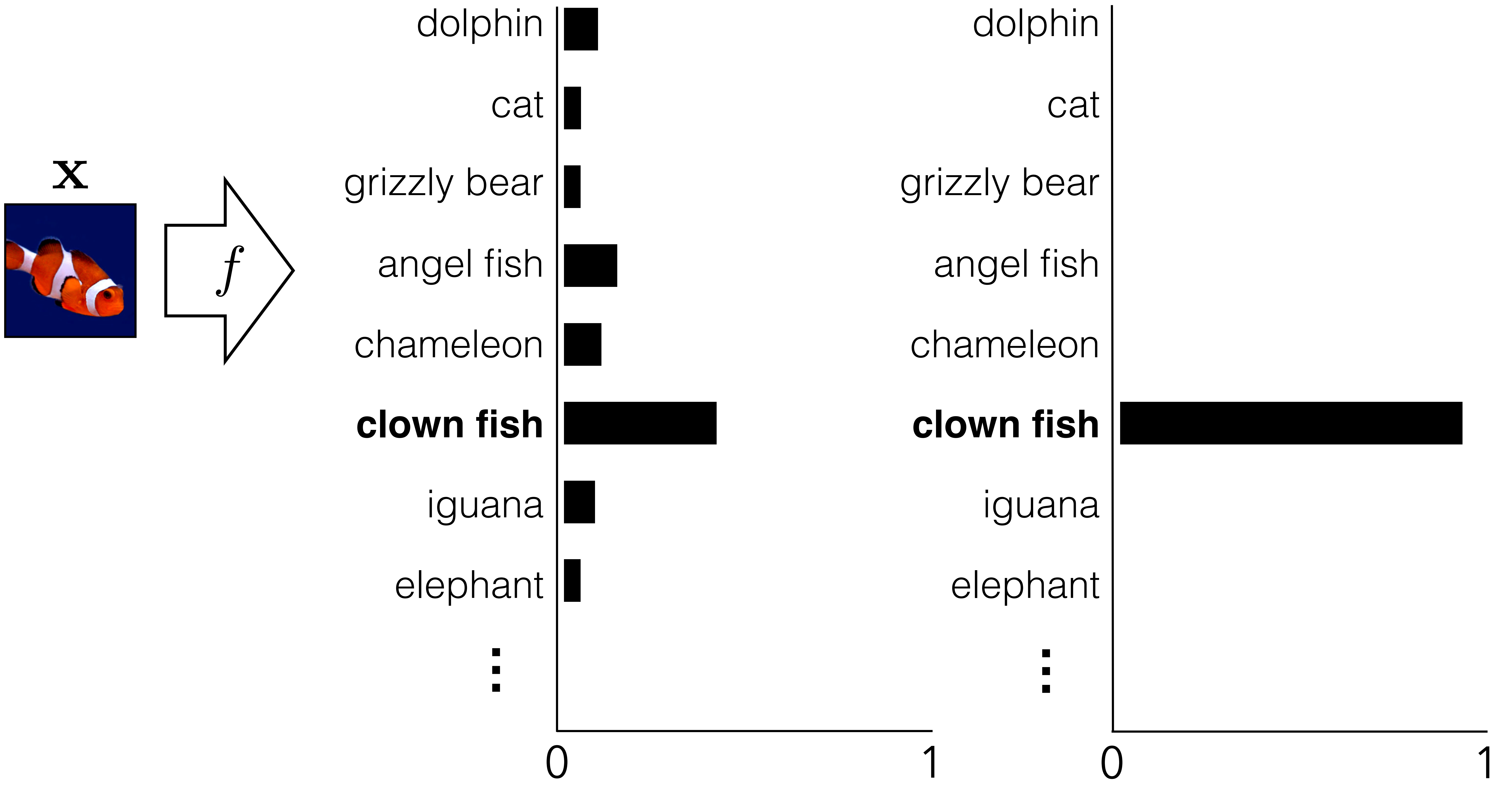
$\mathbf{x}$



Source: Isola, Torralba, Freeman

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

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



Source: Isola, Torralba, Freeman

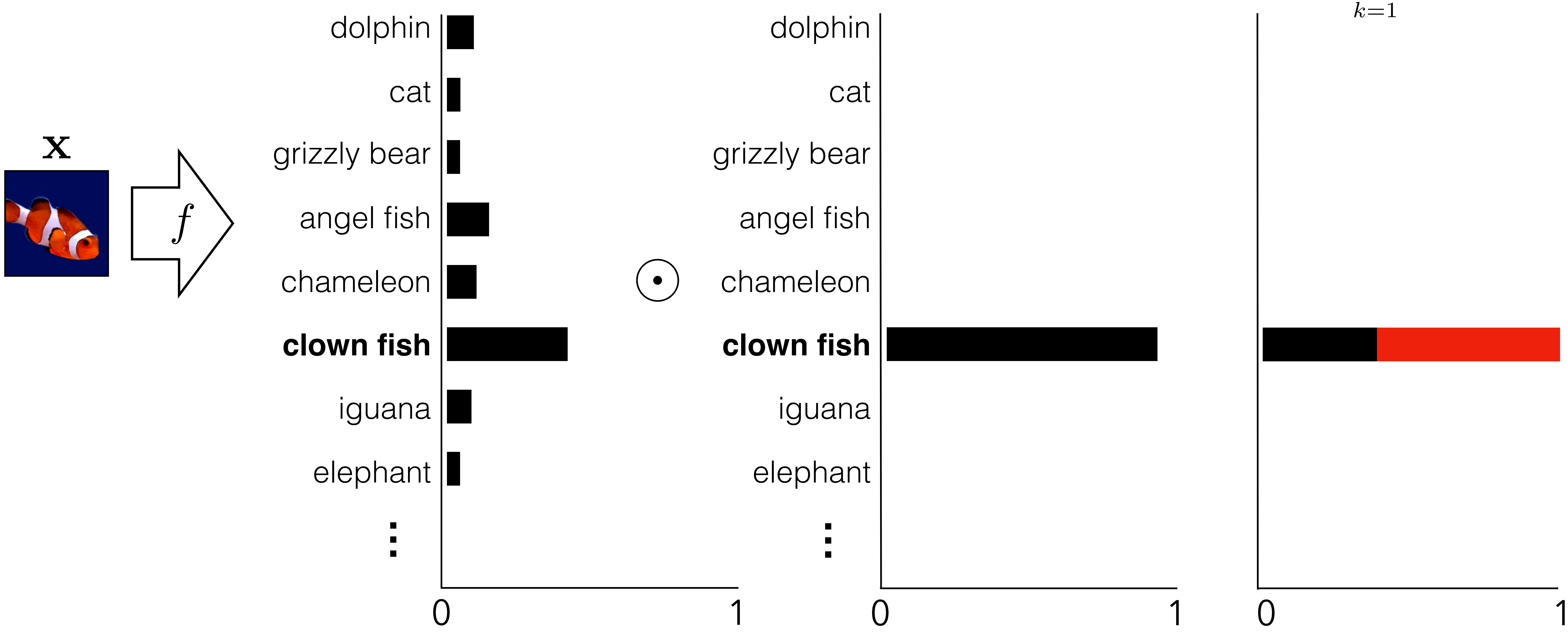
Prediction $\hat{\mathbf{y}}$

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

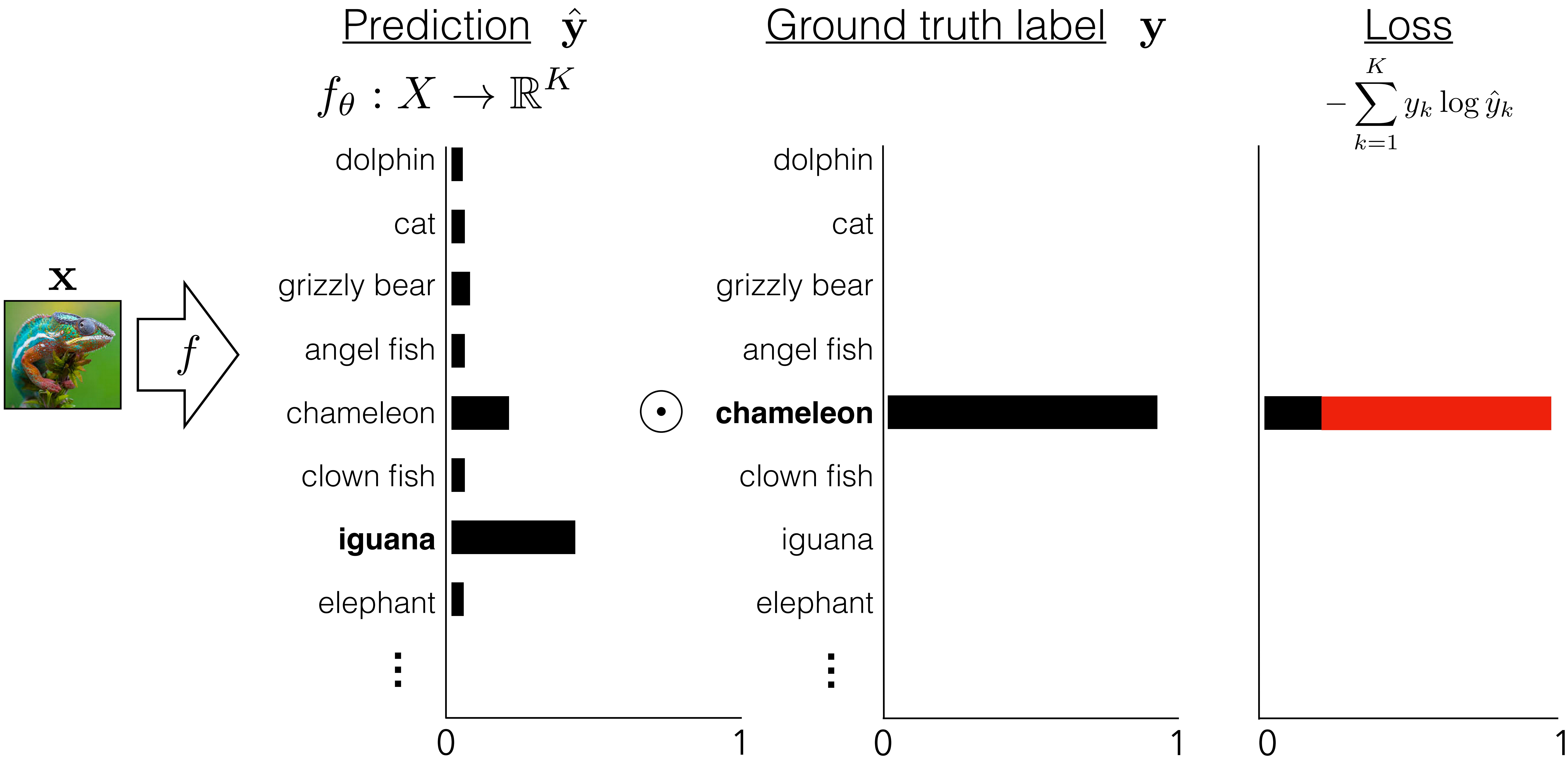
Ground truth label $\mathbf{y}$

Loss

$$-\sum_{k=1}^{K} y_k \log \hat{y}_k$$



Source: Isola, Torralba, Freeman



Source: Isola, Torralba, Freeman

Hinge loss

One more you might see: **hinge loss**

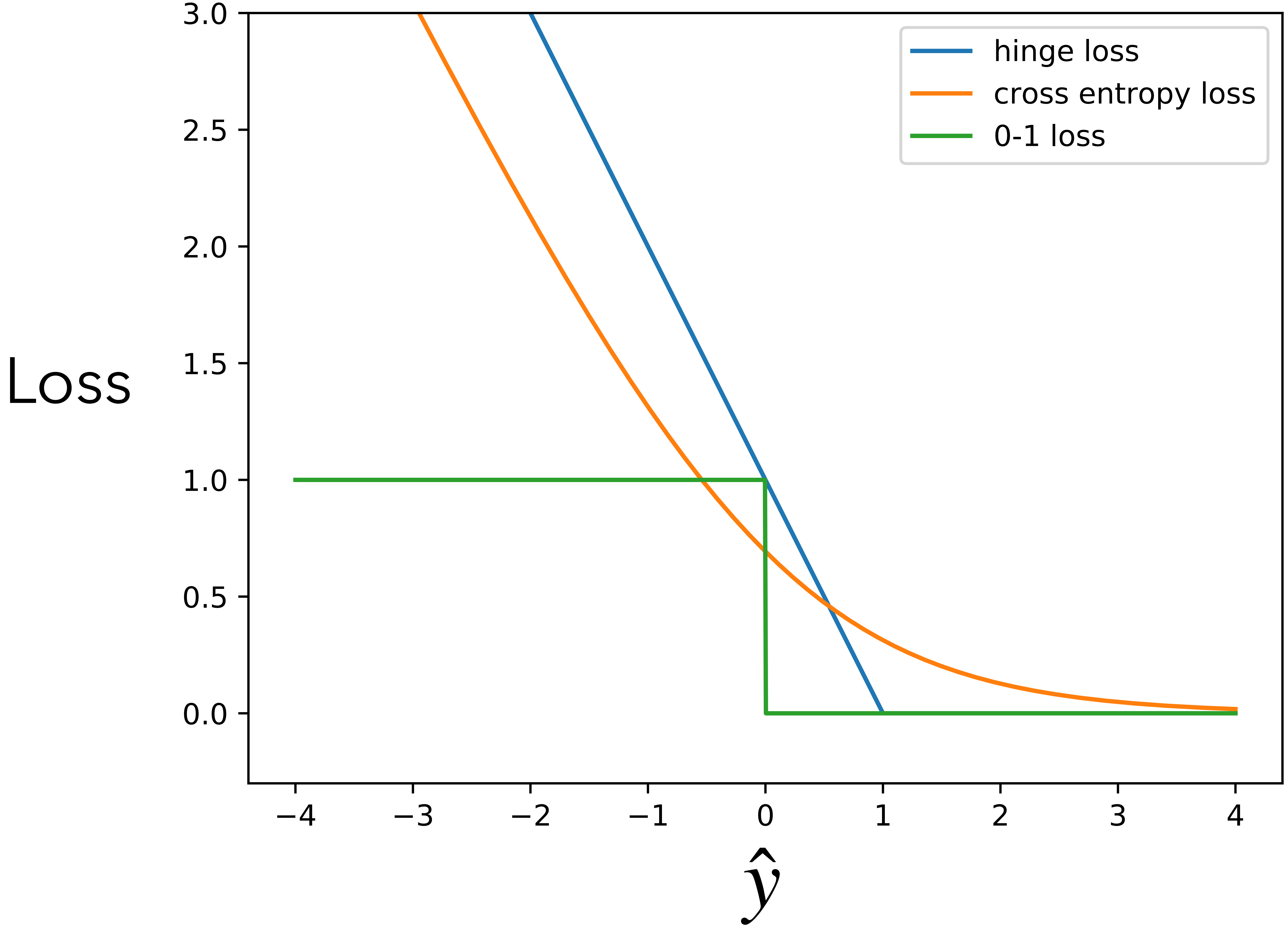
$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = \max(0, 1 - y\hat{y}, 0)$$

⟵ upper bound on 0-1 loss (true class **y** is {+1, -1} instead of {0,1})

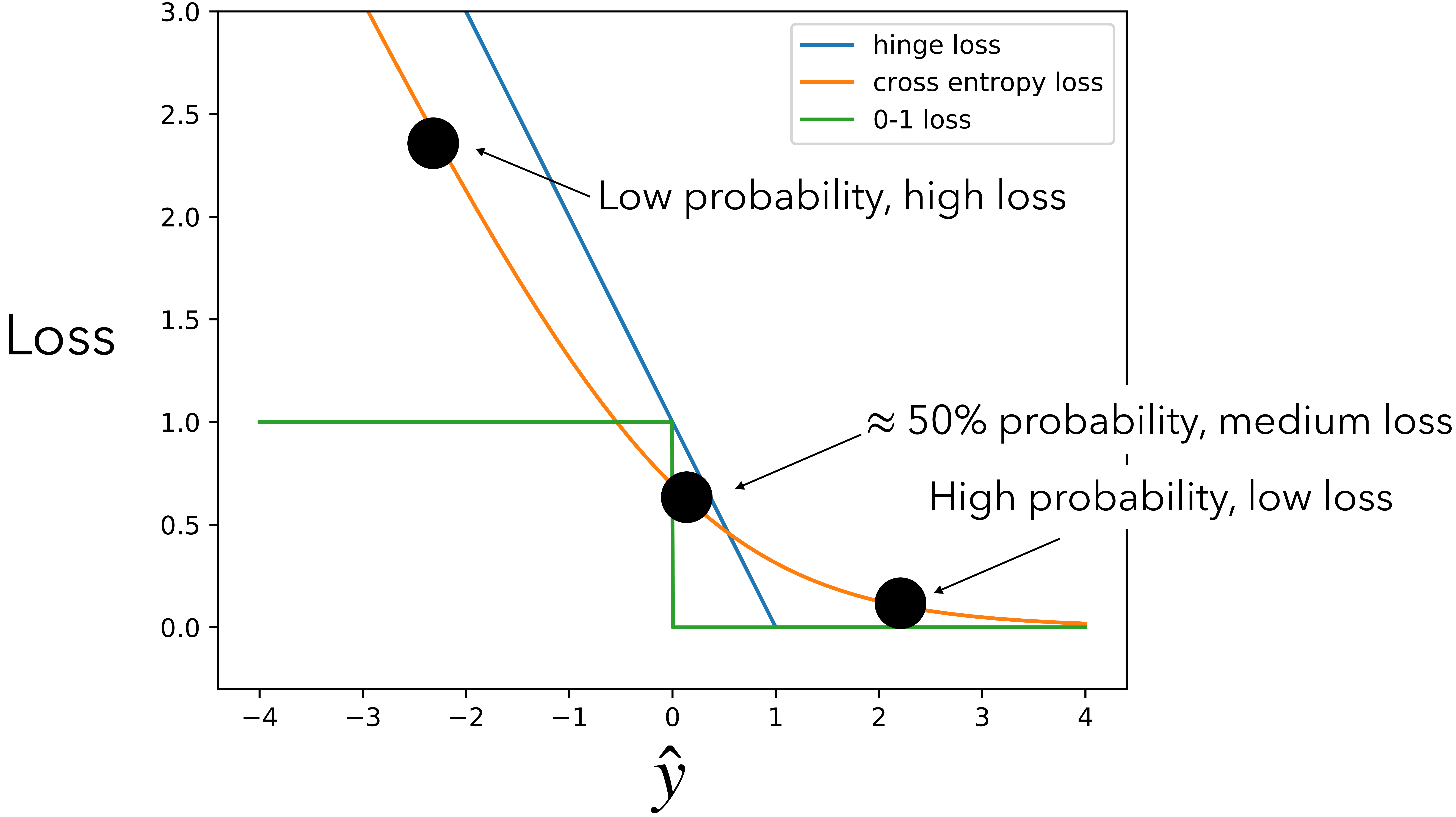
A linear classifier that uses a hinge loss is called a **support vector machine (SVM)**.

It has some very nice properties (they can be made nonlinear with “kernels”). Generally performs similar to logistic regression.

Loss functions

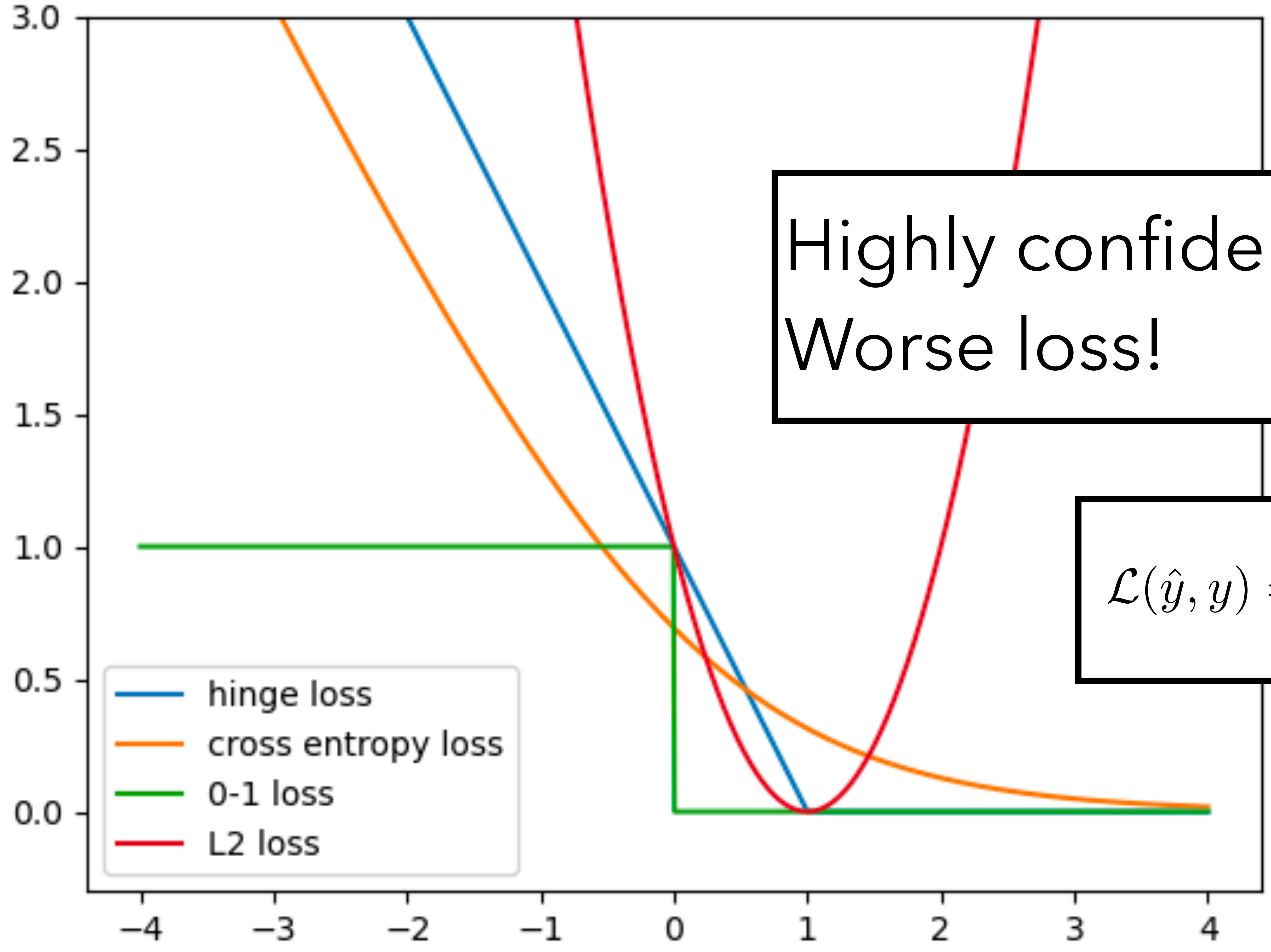


Loss functions



Why not squared loss?

Loss



$\hat{y}$

Highly confident prediction?
Worse loss!

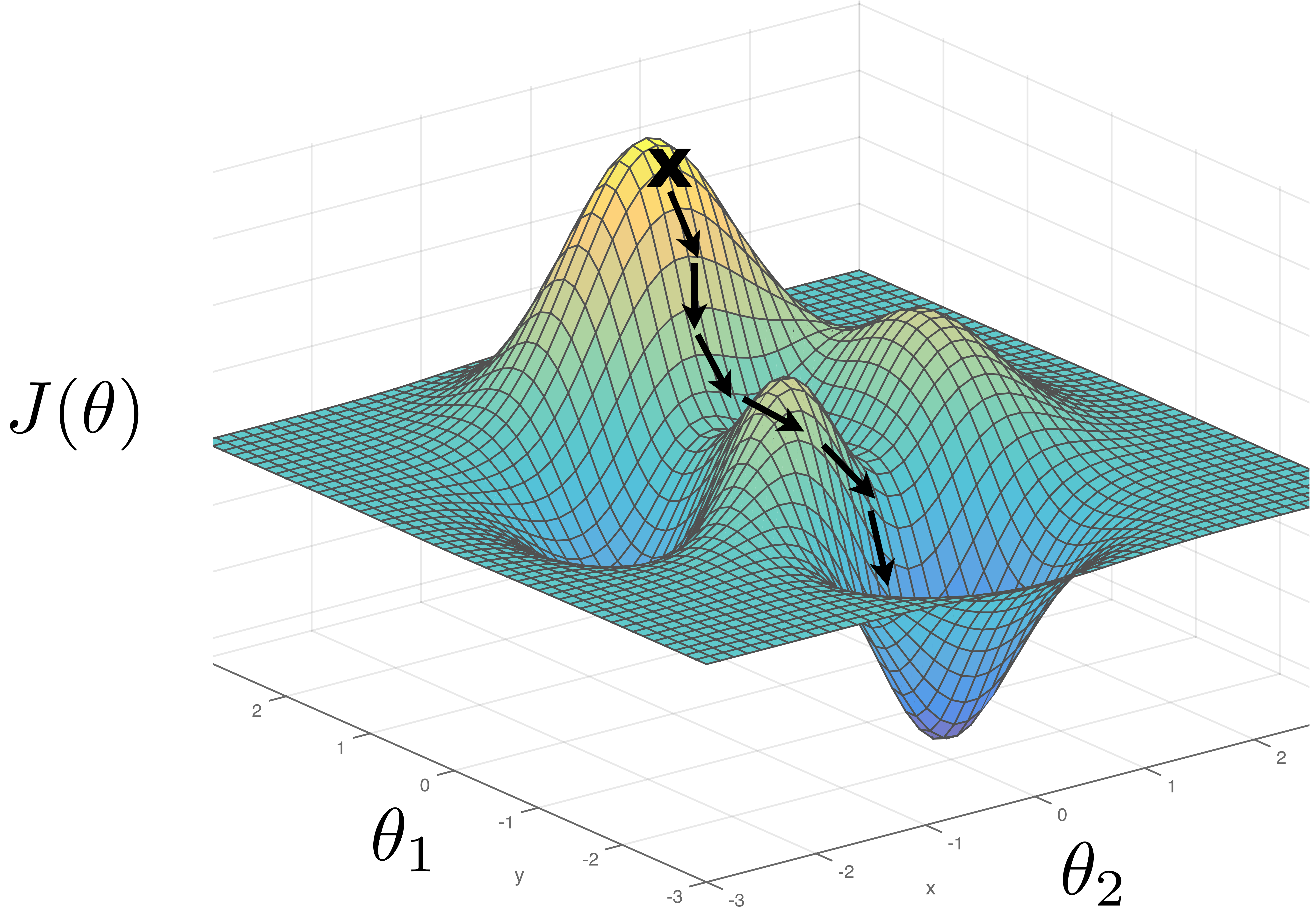
$$\mathcal{L}(\hat{y}, y) = \sum_{k=1}^{K} (y_k - \hat{y}_k)^2$$

How do we learn a classifier?

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

Source: Isola, Torralba, Freeman

Gradient descent



Take direction of "steepest descent"

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

Source: Isola, Torralba, Freeman

Gradient descent

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

One iteration of gradient descent:

$$\theta^{t+1} = \theta^t - \eta_t \nabla_\theta J(\theta)\Big|_{\theta=\theta^t}$$

learning rate

Gradient descent

$$\theta^{t+1} = \theta^t - \eta_t \boxed{\nabla_\theta J(\theta)} \Big|_{\theta=\theta^t}$$

What's this again?

1. Gradient of the loss w.r.t. the classifier's parameters
2. Direction of steepest descent
3. "Local" linear approximation to the function

For a refresher on gradients and partial derivatives:
https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives#partial-derivatives

Gradient descent

Take gradient at current θ

$$\theta^{t+1} = \theta^t - \eta_t \boxed{\nabla_\theta J(\theta)}\Big|_{\theta=\theta^t}$$

Vector of partial derivatives for loss

$$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J}{\theta_1} \\ \frac{\partial J}{\theta_2} \\ \cdots \\ \frac{\partial J}{\theta_d} \end{bmatrix}$$

Estimating the gradient

Idea #1: finite differences $\dfrac{f(x+\epsilon) - f(x)}{\epsilon}$

$$\nabla J(\theta) = \begin{bmatrix} \dfrac{\partial J}{\theta_1} \\ \dfrac{\partial J}{\theta_2} \\ \cdots \\ \dfrac{\partial J}{\theta_d} \end{bmatrix}$$

$$\approx \frac{J(\theta + \epsilon e_2) - J(\theta - \epsilon e_2)}{2\epsilon}$$

where e_2 is vector of zeros except for a 1 in 2nd component

Estimating the gradient

Idea #1: finite differences $\dfrac{f(x+\epsilon)-f(x)}{\epsilon}$

$$\nabla J(\theta) = \begin{bmatrix} \dfrac{\partial J}{\theta_1} \\ \dfrac{\partial J}{\theta_2} \\ \cdots \\ \dfrac{\partial J}{\theta_d} \end{bmatrix}$$

- Easy to compute but slow!
- Inaccurate in some cases, unless careful
- Useful for checking other methods

Estimating the gradient

Idea #2: compute it analytically using calculus.

Simple example: binary labels, squared loss, and regularization on θ

$$J(\theta) = \boxed{\lambda||\theta||^2} + \boxed{\frac{1}{N}\sum_{i=1}^{N}(y_i - \theta^\top x_i)^2}$$

$$\nabla_\theta$$

$$\nabla_\theta J(\theta) = \boxed{2\lambda\theta} - \boxed{\frac{1}{N}\sum_{i=1}^{N}2(y_i - \theta^\top x_i)x_i}$$

Example from D. Fouhey

Analyzing the gradient descent update

Recall update rule: $\theta^{t+1} = \theta^t - \eta_t \nabla_\theta J(\theta)\Big|_{\theta=\theta^t}$

How does this change θ ?

$$-\nabla_\theta J(\theta) =$$

Example from D. Fouhey

Analyzing the gradient descent update

Recall update rule: $\theta^{t+1} = \theta^t - \eta_t \nabla_\theta J(\theta)\Big|_{\theta=\theta^t}$

How does this change θ ?

"Decay" toward 0

Scalar α_i per example

$$-\nabla_\theta J(\theta) = \underbrace{-2\lambda\theta}_{\text{"Decay" toward 0}} - \frac{1}{N}\sum_{i=1}^{N} \underbrace{2(y_i - \theta^\top x_i)}_{\text{Scalar } \alpha_i \text{ per example}} x_i$$

Example from D. Fouhey

Analyzing the gradient descent update

What happens at each example?

$$-\nabla_\theta J(\theta) = -2\lambda\theta - \frac{1}{N}\sum_{i=1}^{N} \overbrace{\boxed{2(y_i - \theta^\top x_i)}}^{\text{Scalar } \alpha_i} x_i$$

If $\theta^\top x_i < y$ (too *low*): then $\theta_{t+1} = \theta + \alpha x_i$ for some $\alpha > 0$

Dot product before: $\theta^\top x_i$

Dot product after: $(\theta + \alpha x_i)^\top x_i = \theta^\top x_i + \alpha x_i^\top x_i$

Example from D. Fouhey

Computational issues

Batch gradient descent

Loss function:

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} L(x_i, y_i, \theta)$$

Its gradient is the sum of gradients for each example:

$$\nabla J(\theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla L(x_i, y_i, \theta)$$

Problem: requires iterating over every training example each gradient step!

Can we speed this up?

Stochastic gradient descent

This is just an average!

$$\nabla J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla L(x_i, y_i, \theta)$$

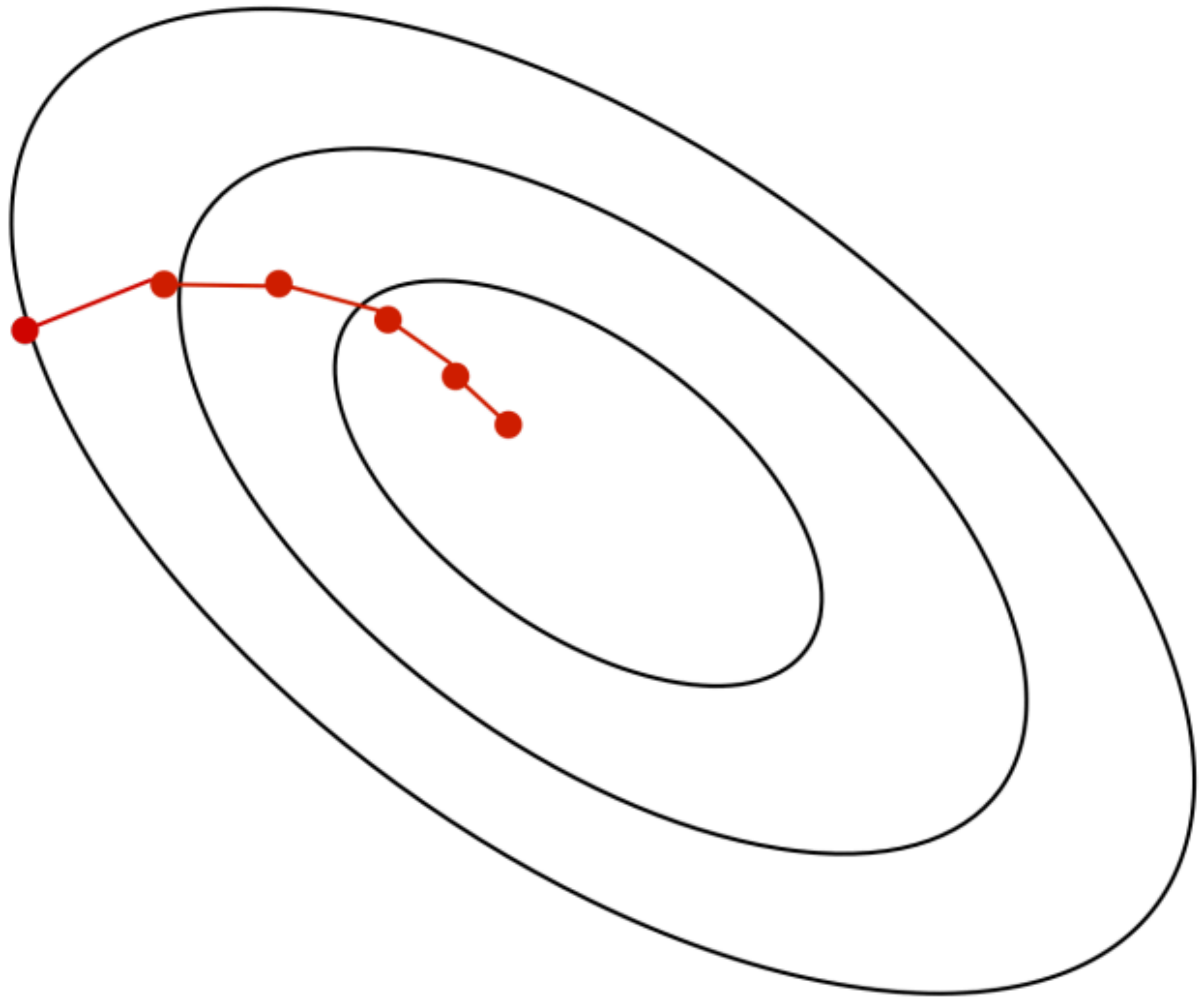
We know from statistics that we can estimate the average of a full "population" from a *sample*.

$$\nabla J(\theta) \approx \frac{1}{|B|} \sum_{i \in B} \nabla L(x_i, y_i, \theta)$$

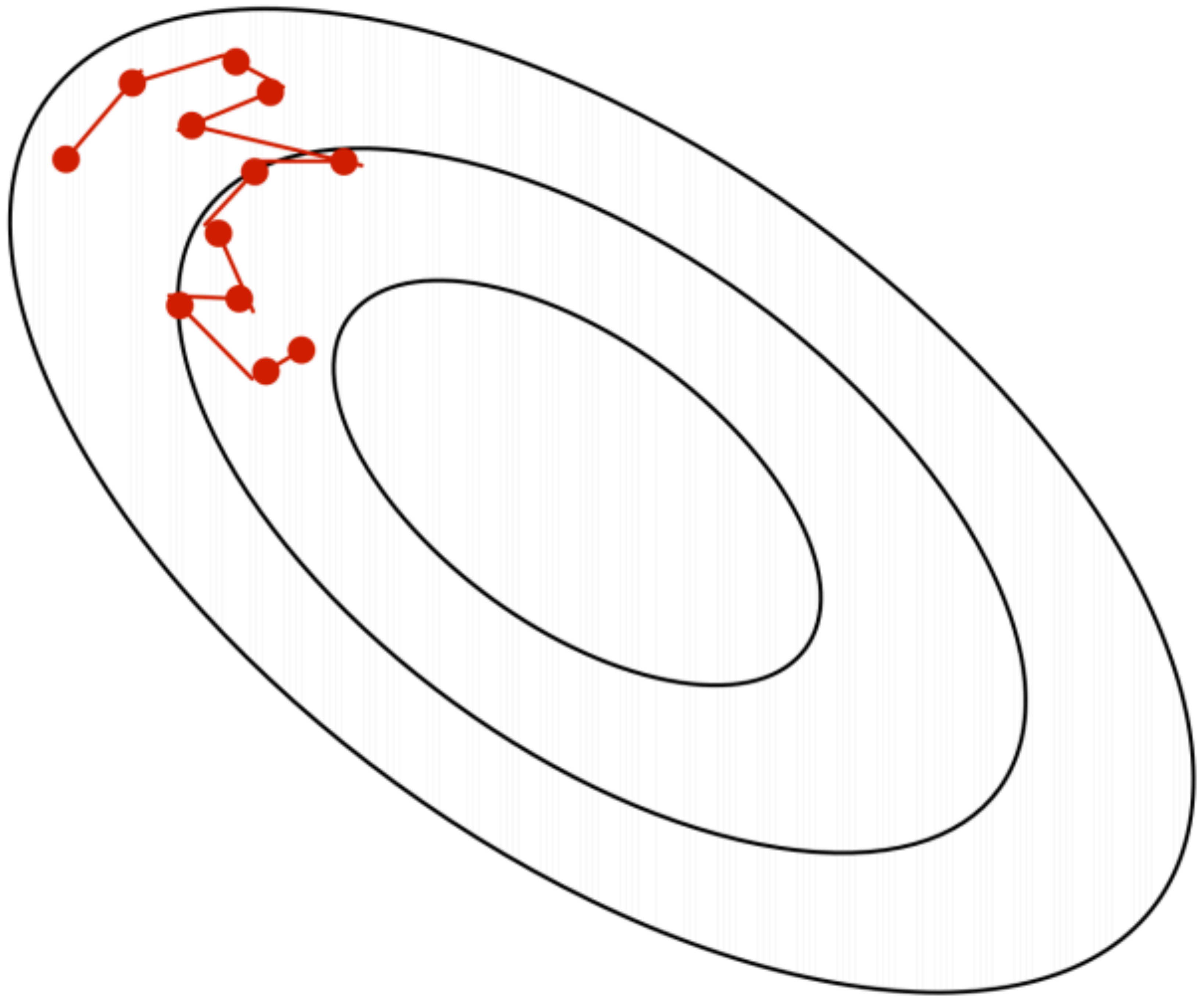
where B is a **minibatch:** a random subset of examples.

This is called **stochastic gradient descent (SGD)**.

Stochastic gradient descent



Batch gradient descent

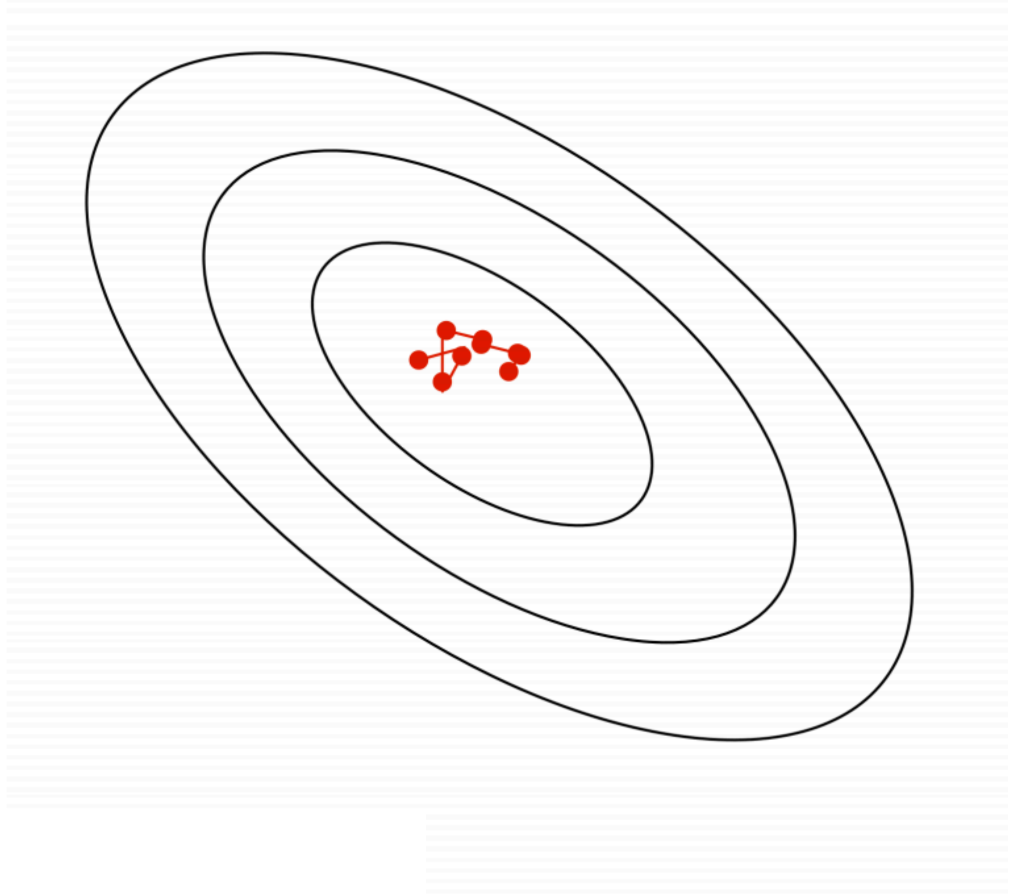


Stochastic gradient descent

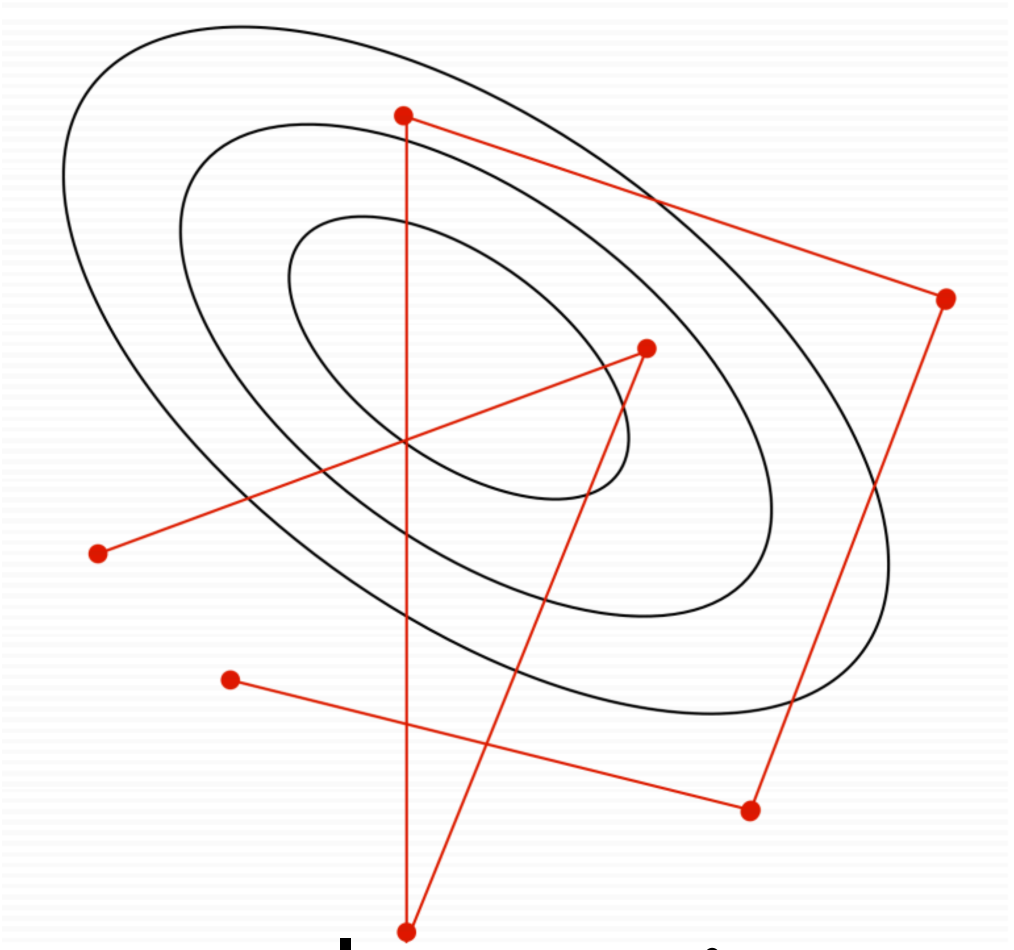
Source: R. Grosse

Learning rate

- Sensitive to the learning rate: $\theta^{t+1} = \theta^t - \eta_t \nabla_\theta J(\theta)\Big|_{\theta=\theta^t}$



Small learning

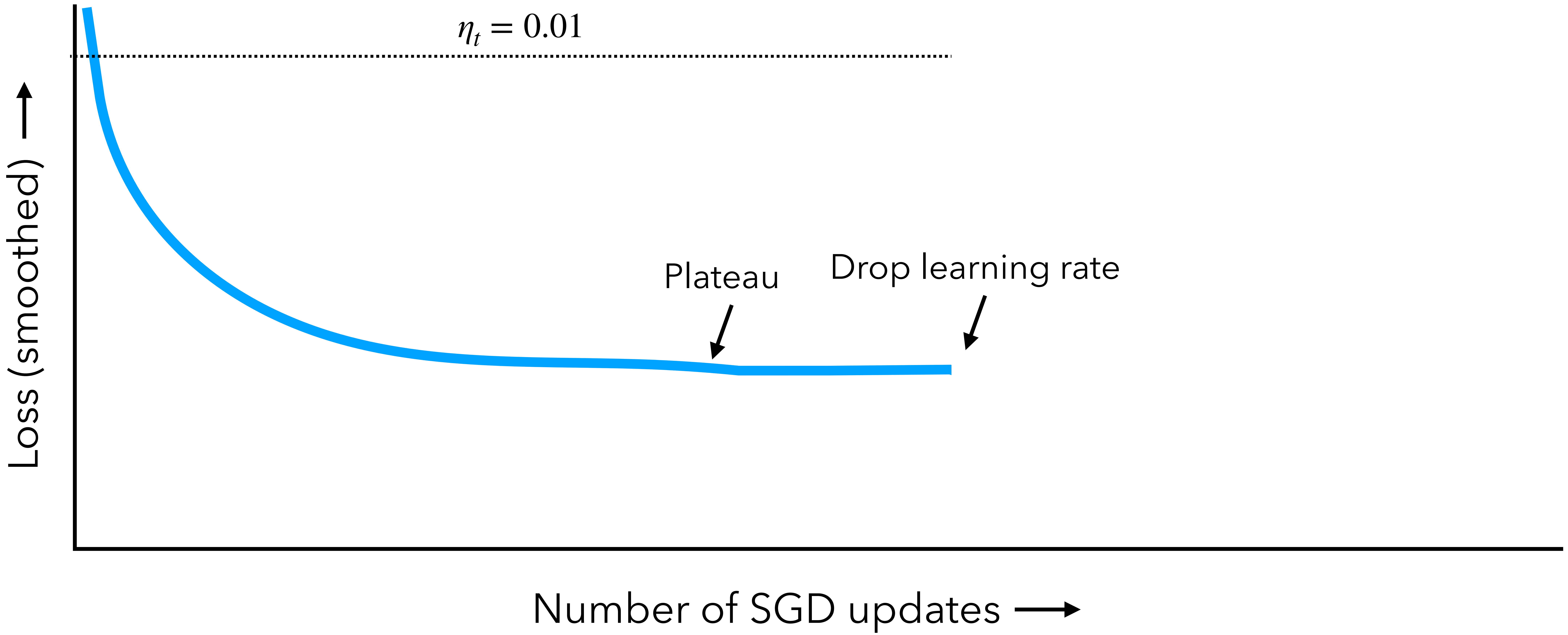


Large learning rate

Figure source: R. Grosse

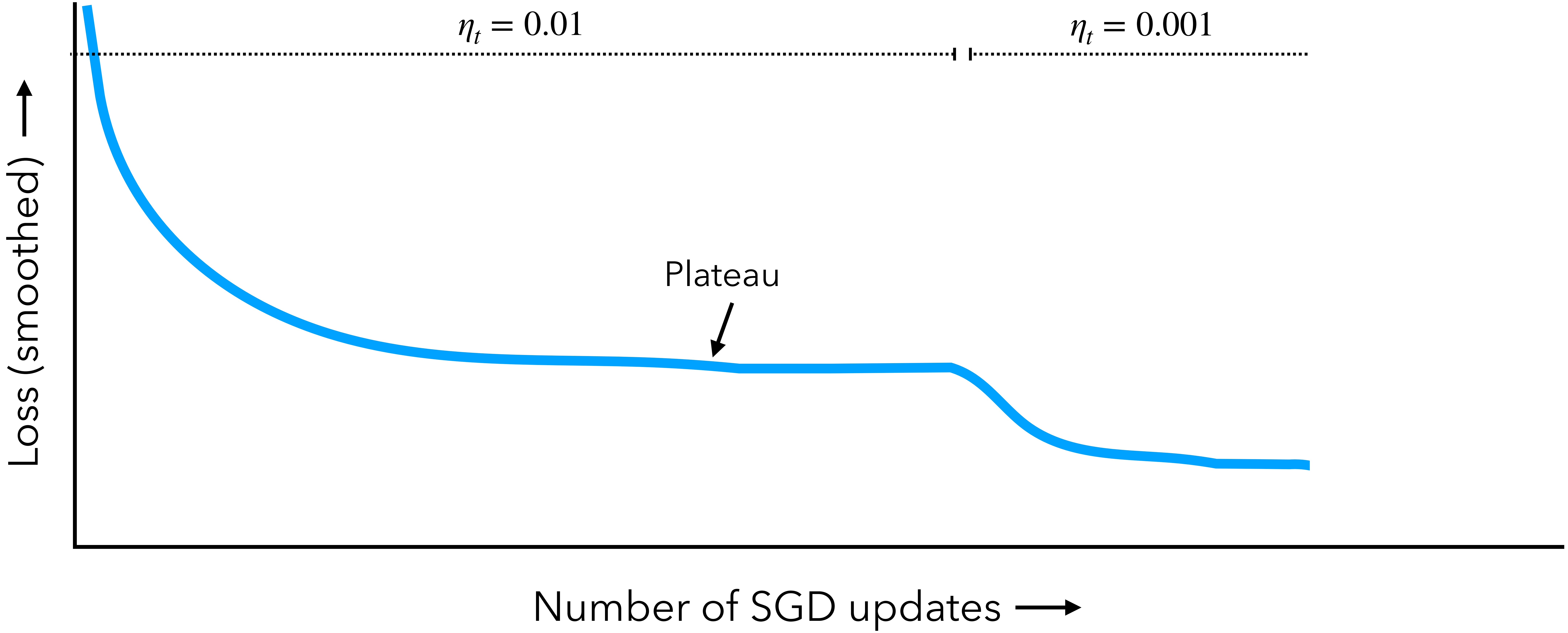
Learning rate schedules

- Start with high learning rate, and decrease over time.



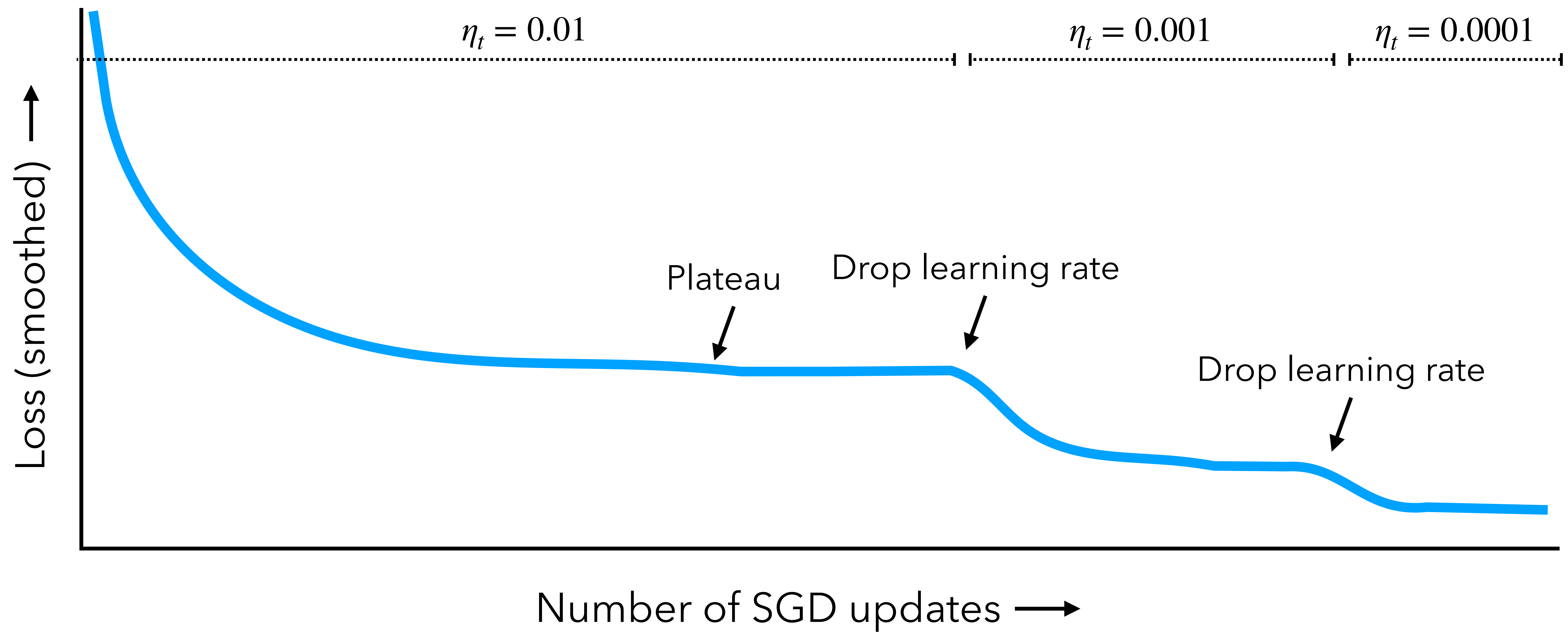
Learning rate schedules

- Start with high learning rate, and decrease over time.



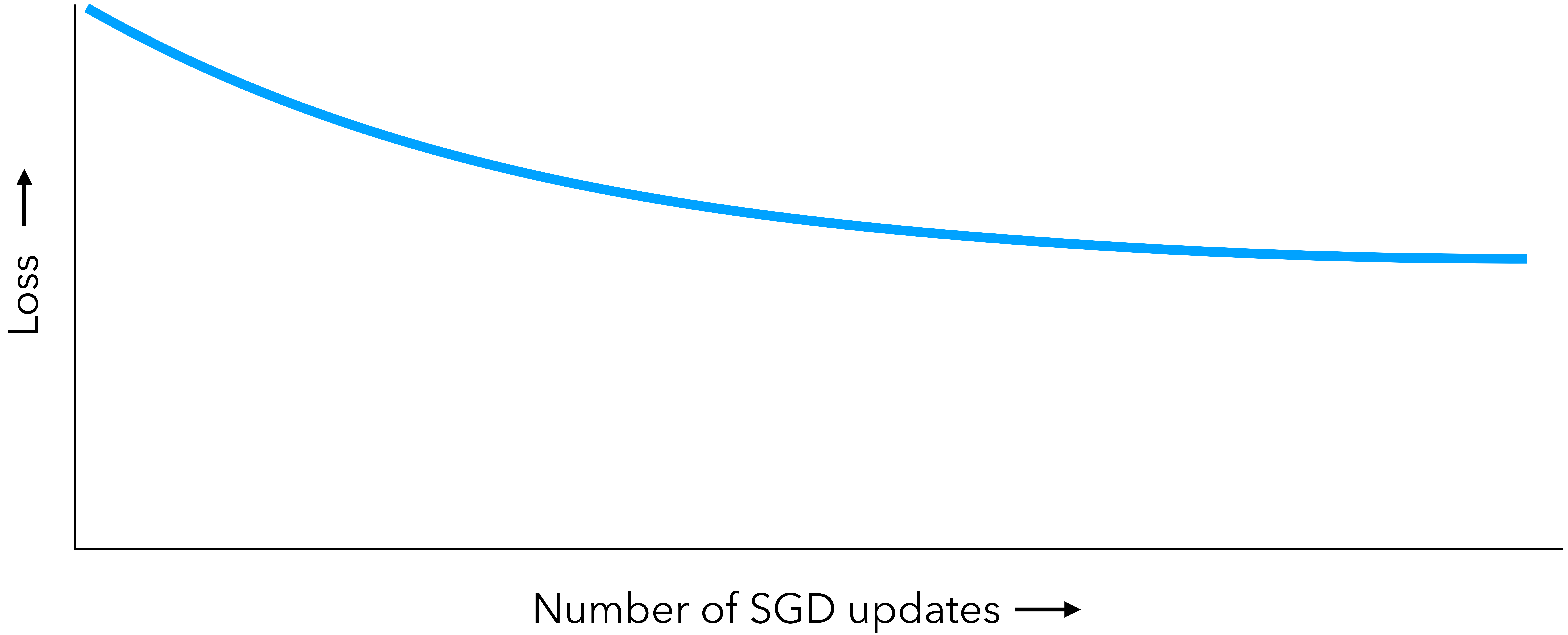
Learning rate schedules

- Start with high learning rate, and decrease over time.



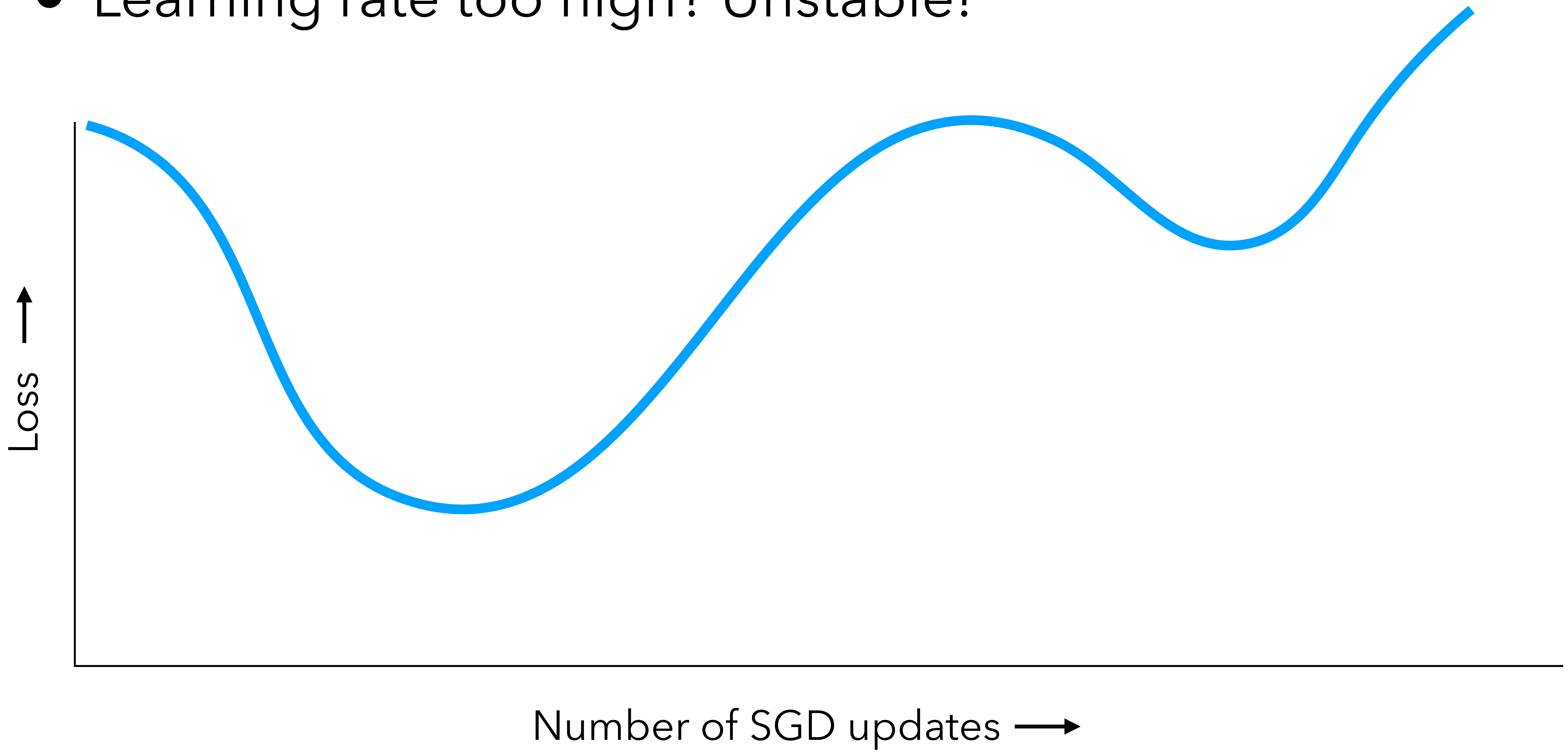
Learning rate schedules

- What if you always use a small learning rate?
- Loss goes down extremely slowly



Learning rate schedules

- Learning rate too high? Unstable!



Learning rate schedules

- Choosing the initial learning rate is surprisingly hard. Often requires grid search, i.e., trying many rates and choosing the best.
- When do you drop the learning rate? Some strategies:
    - Wait until validation or training loss plateaus, then drop it (e.g., by a factor of 10).
    - Smoothly drop the learning rate over time. Requires choosing the rate of dropping.
    - Warm up: make the beginning of training easier. Start learning rate at 0 and gradually increase for the first few iterations.
- Another option: decrease *and* increase the learning rate using a periodic function (e.g., cosine) [Loshchilov & Hutter, 2017]

Regularization

- As before, add L_2 regularization $R(\theta) = ||\theta||^2$ model parameters.
- **Early stopping:** stop training when validation loss increases, and revert to previous checkpoint.

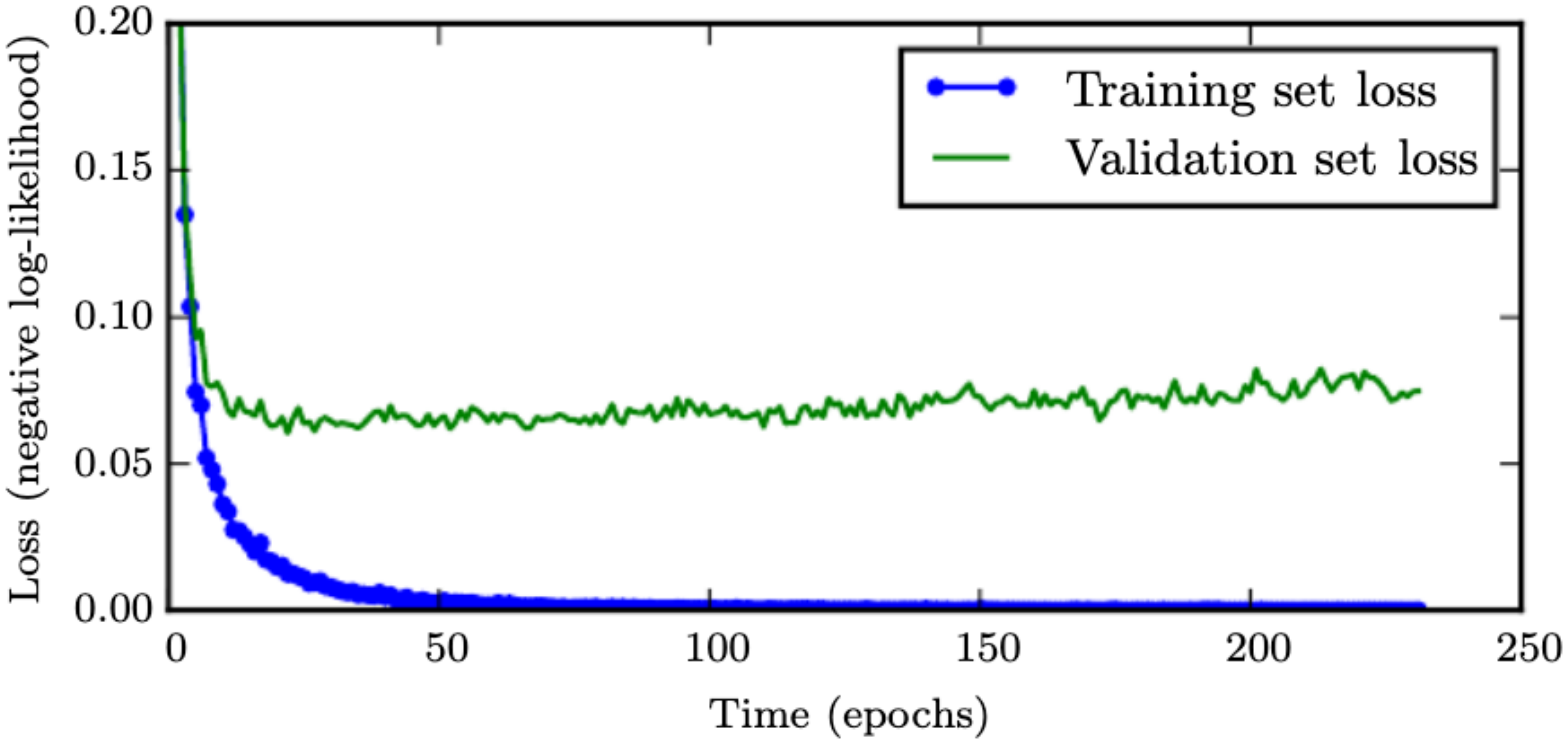
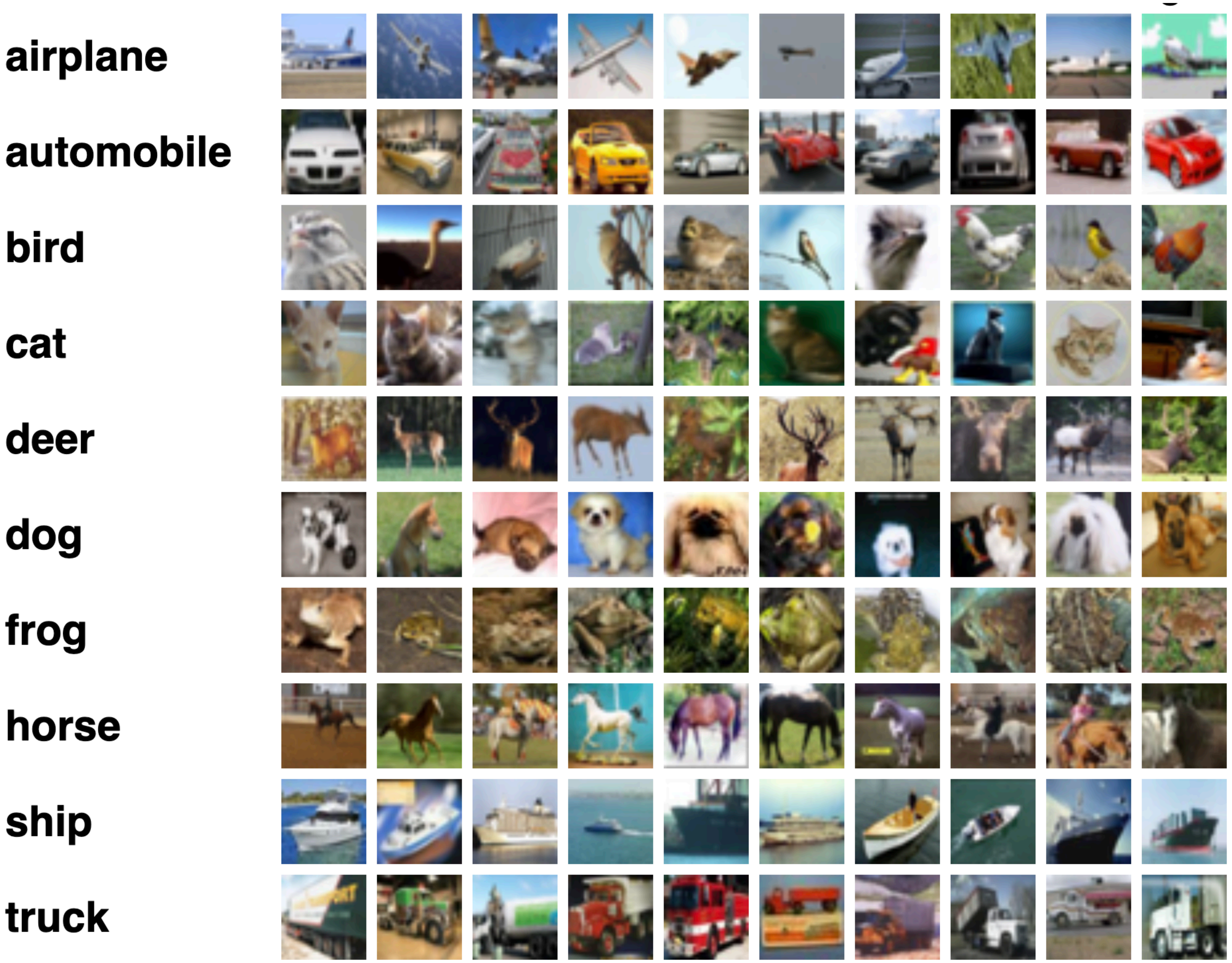


Figure source: [Goodfellow et al., 2016

PS2: intro to machine learning



Part #1: classification

- Object recognition with tiny images
- K-nearest neighbor
- Logistic regression with SGD
- Cross-validation

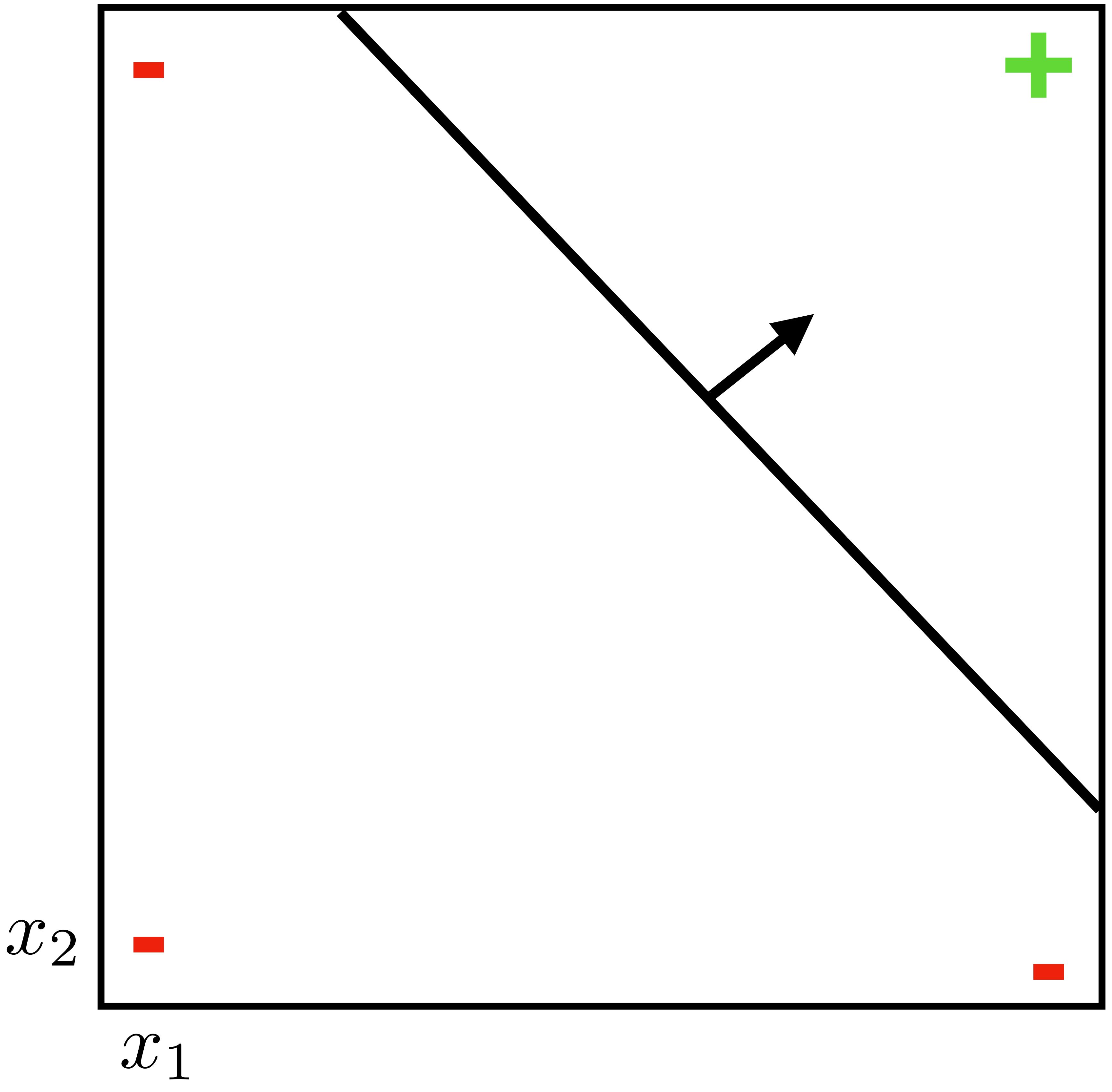
Can be completed after today's class.

Part #2: neural nets with PyTorch

- Neural nets in PyTorch
- Convolutional networks

Covered this Weds. and next Mon.

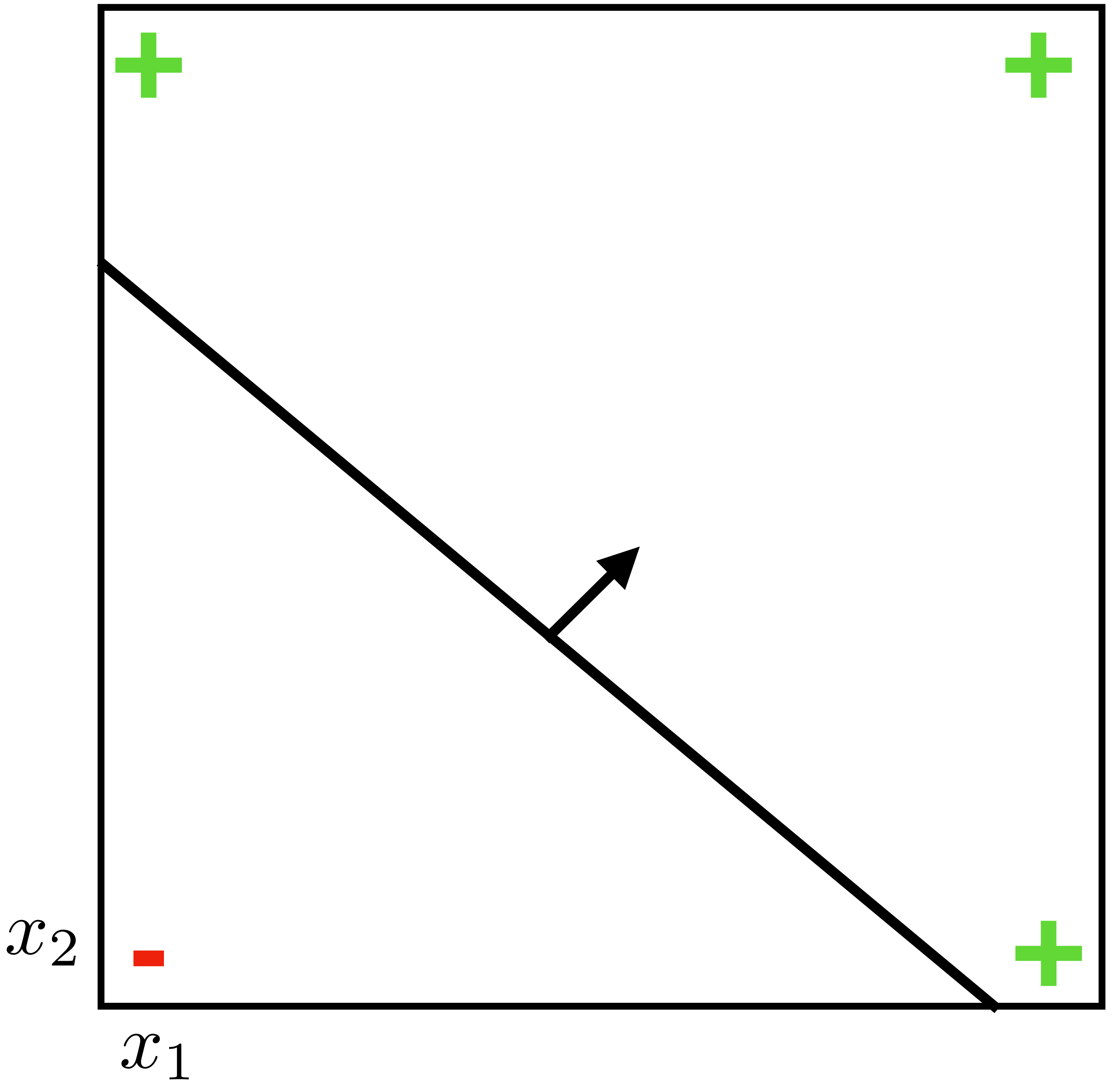
Limitations to linear classifiers



		x_2	
		0	1
x_1	0	0	0
	1	0	1

AND

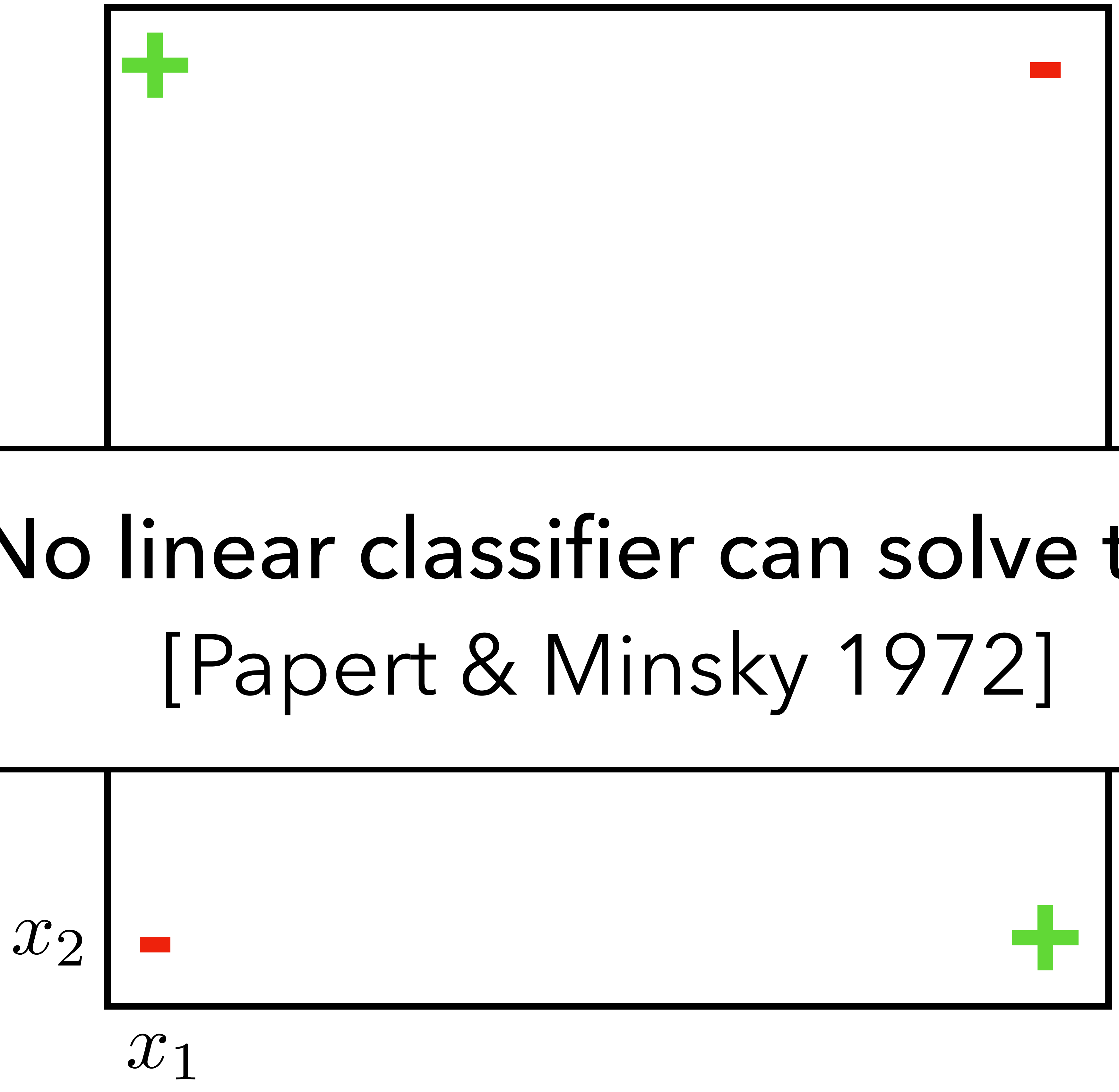
Limitations to linear classifiers



		x_2 = 0	x_2 = 1
x_1	0	0	1
	1	1	1

OR

Limitations to linear classifiers



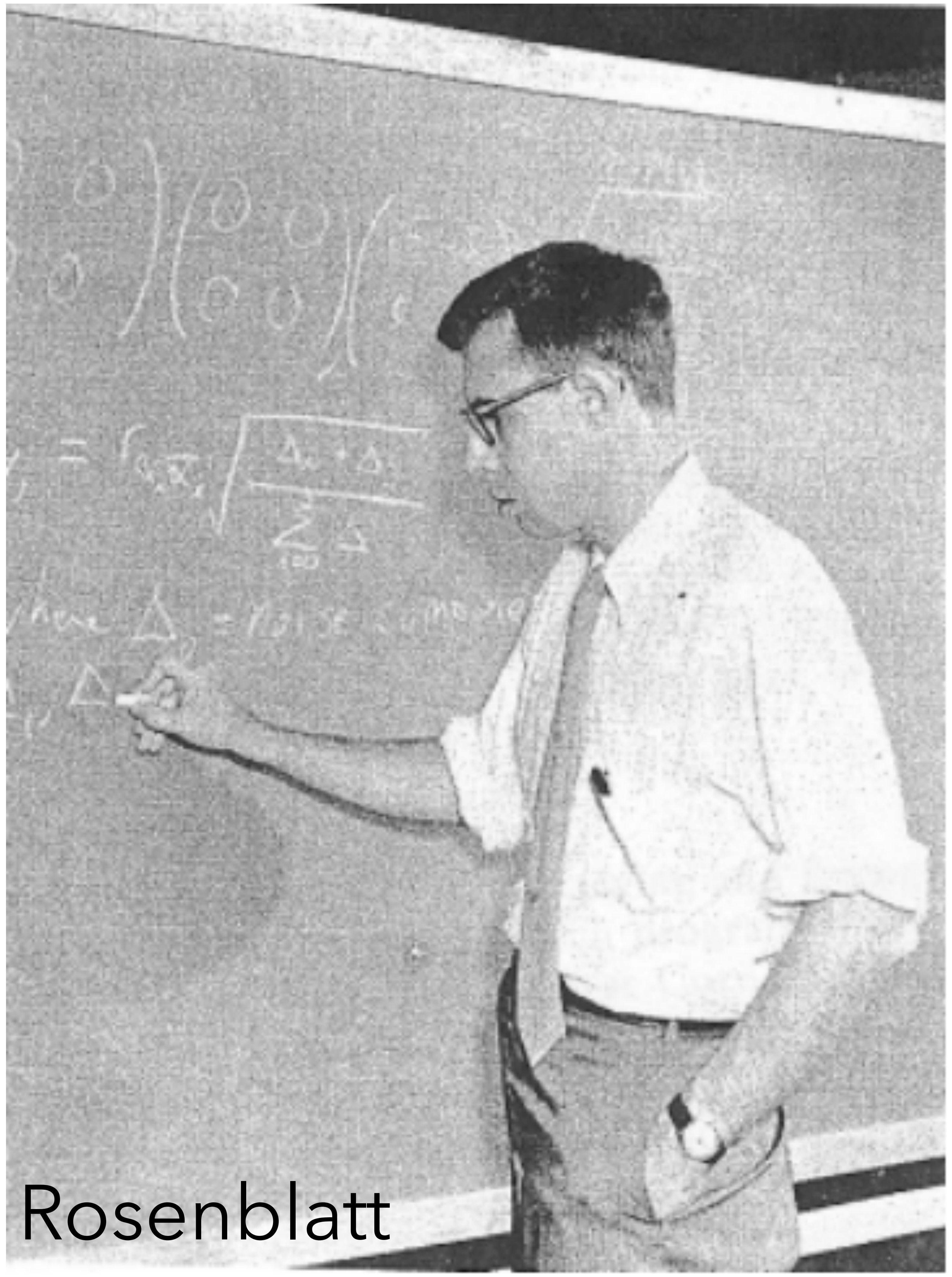
No linear classifier can solve this
[Papert & Minsky 1972]

		x_2	
		0	1
x_1	0	1	0
	1	0	1

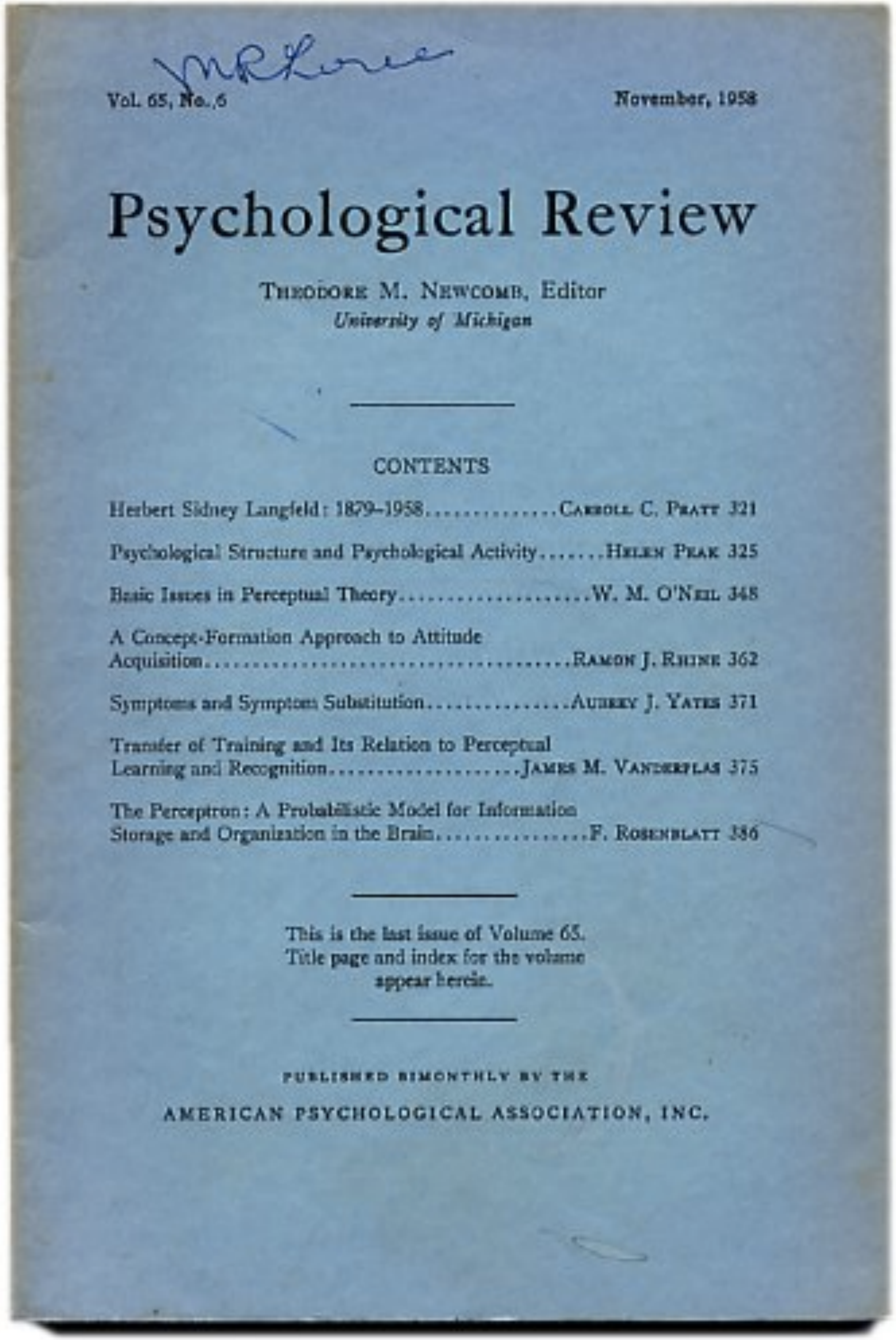
XOR

Next class: Neural networks

Perceptrons, 1958



[http://www.ecse.rpi.edu/homepages/nagy/PDF_chrono/2011_Nagy_Pace_FR.pdf.](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

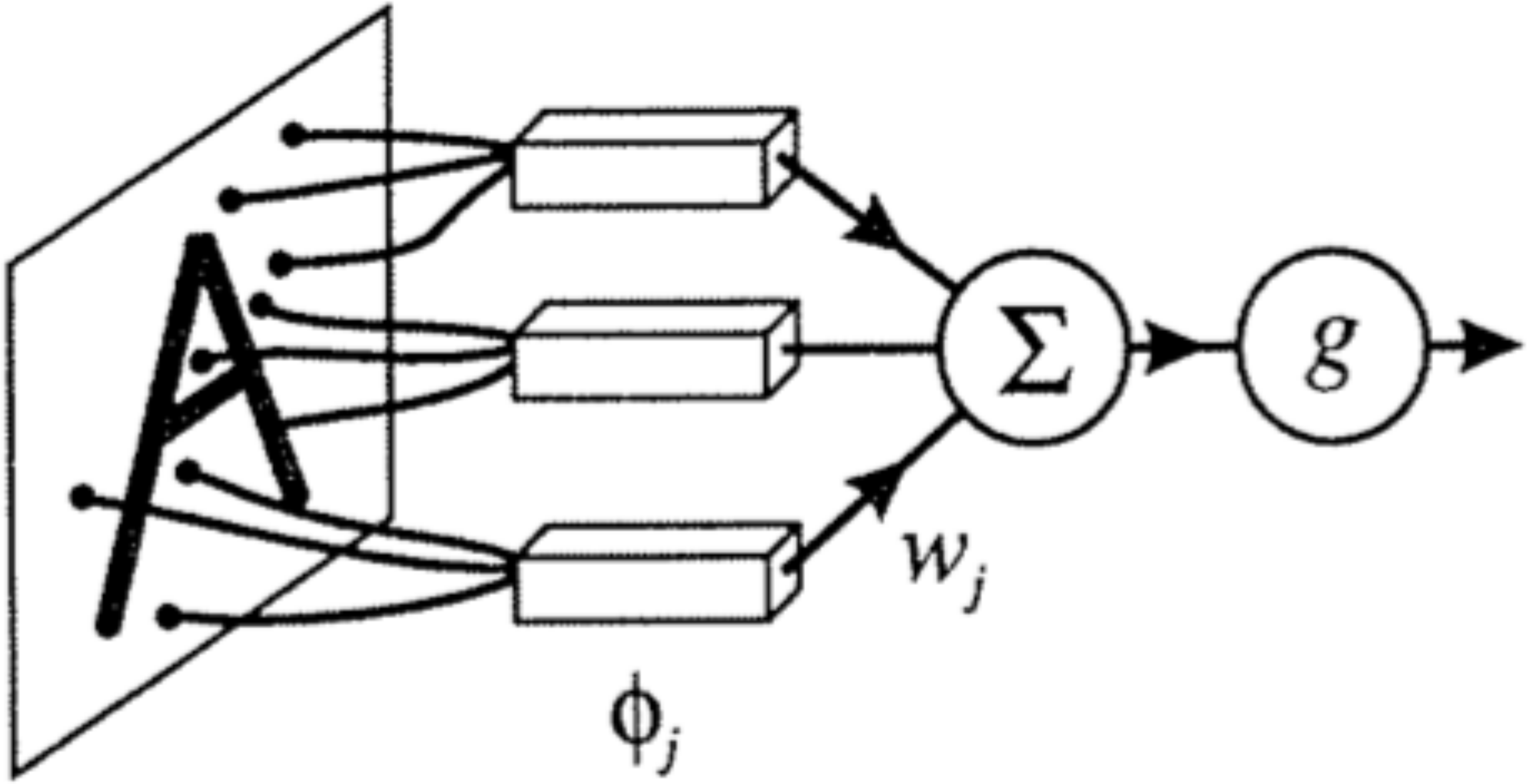
This is the last issue of Volume 65.
Title page and index for the volume appear herein.

PUBLISHED BIMONTHLY BY THE
AMERICAN PSYCHOLOGICAL ASSOCIATION, INC.

Source: Isola, Torralba, Freeman

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

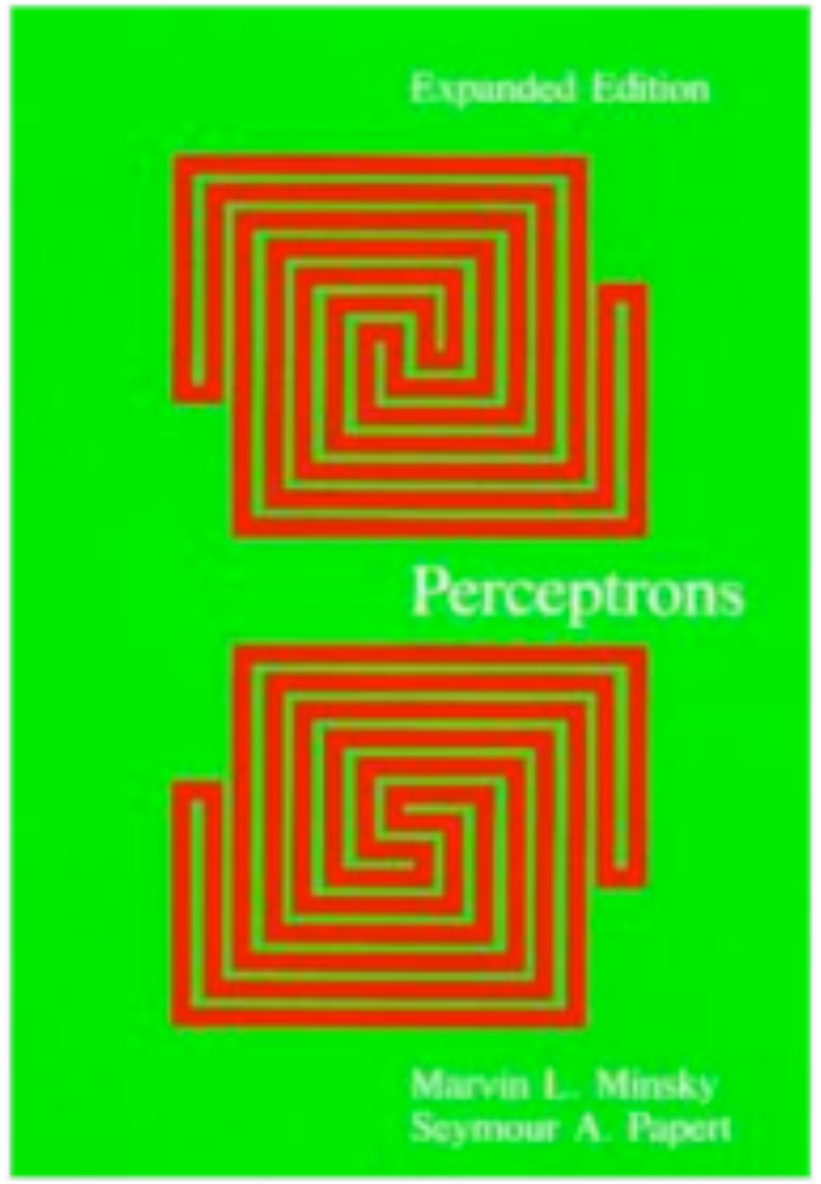
Perceptrons, 1958



Very similar to the linear models we've seen.

Source: Isola, Torralba, Freeman

Minsky and Papert, Perceptrons, 1972



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."

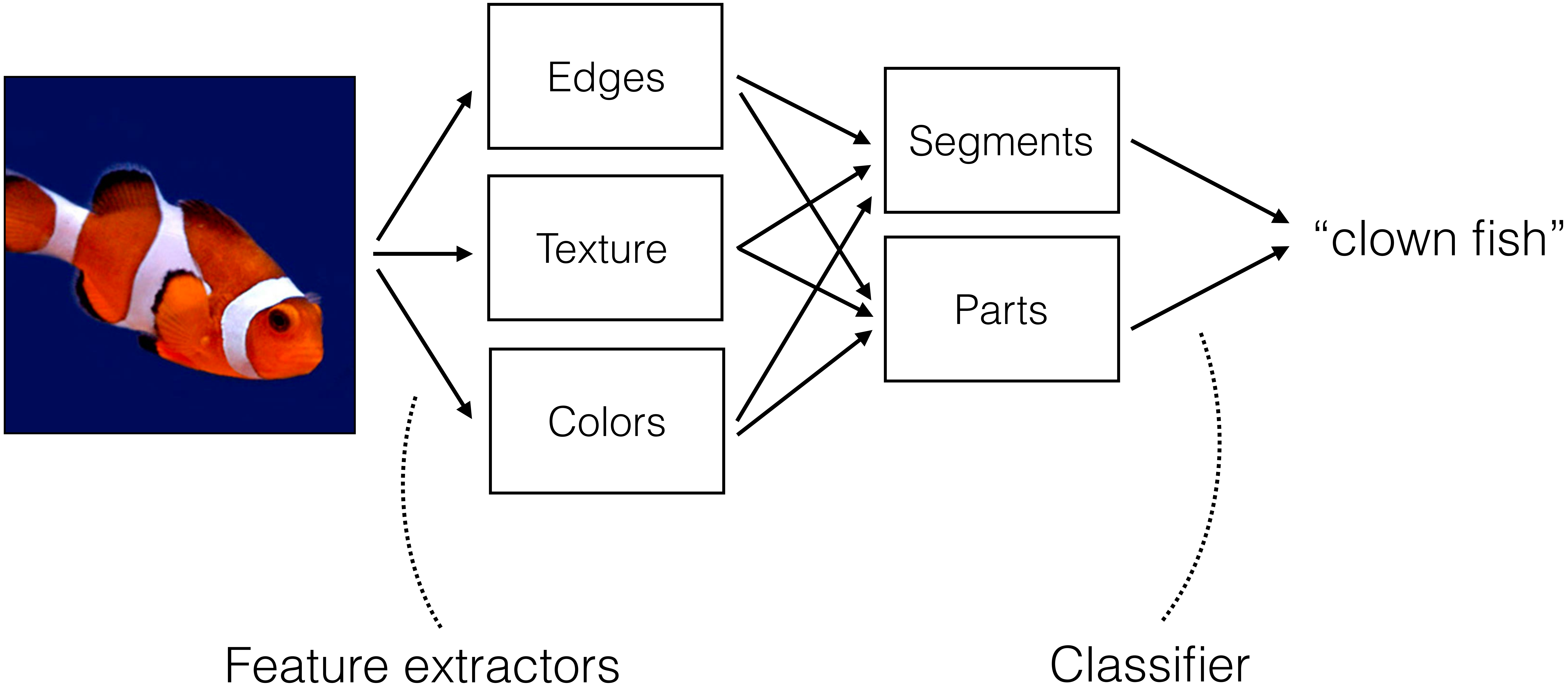
FOR BUYING OPTIONS, START HERE

Select Shipping Destination

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

Source: Isola, Torralba, Freeman

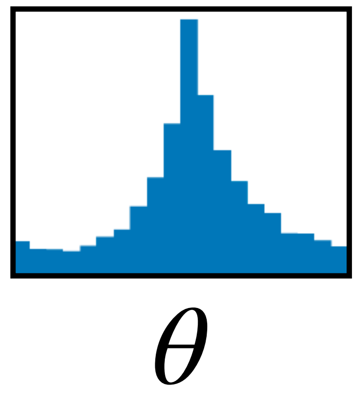
“Classic” recognition **without** neural nets



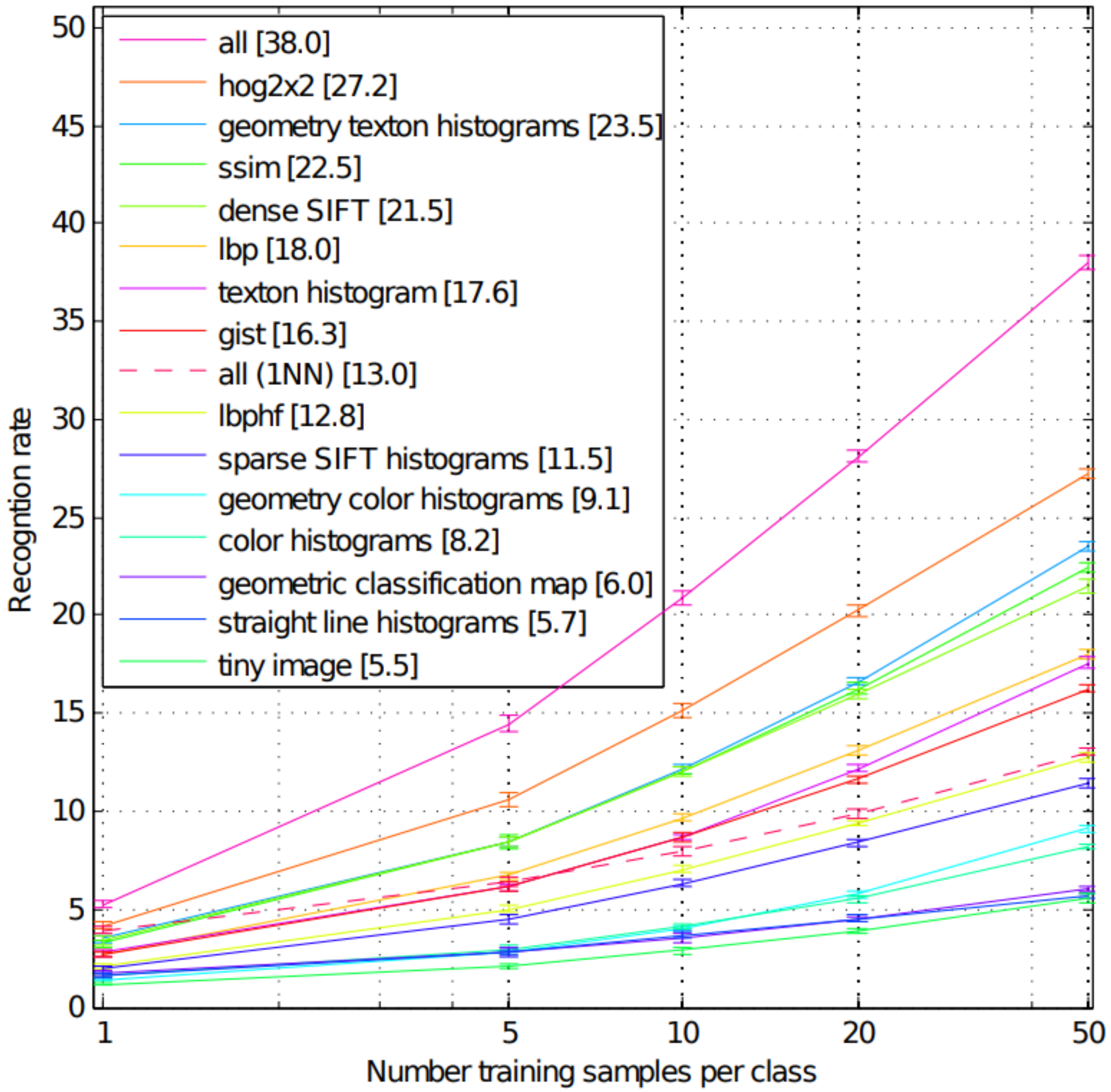
Source: Isola, Torralba, Freeman

“Classic” recognition **without** neural nets

Hand-crafted features

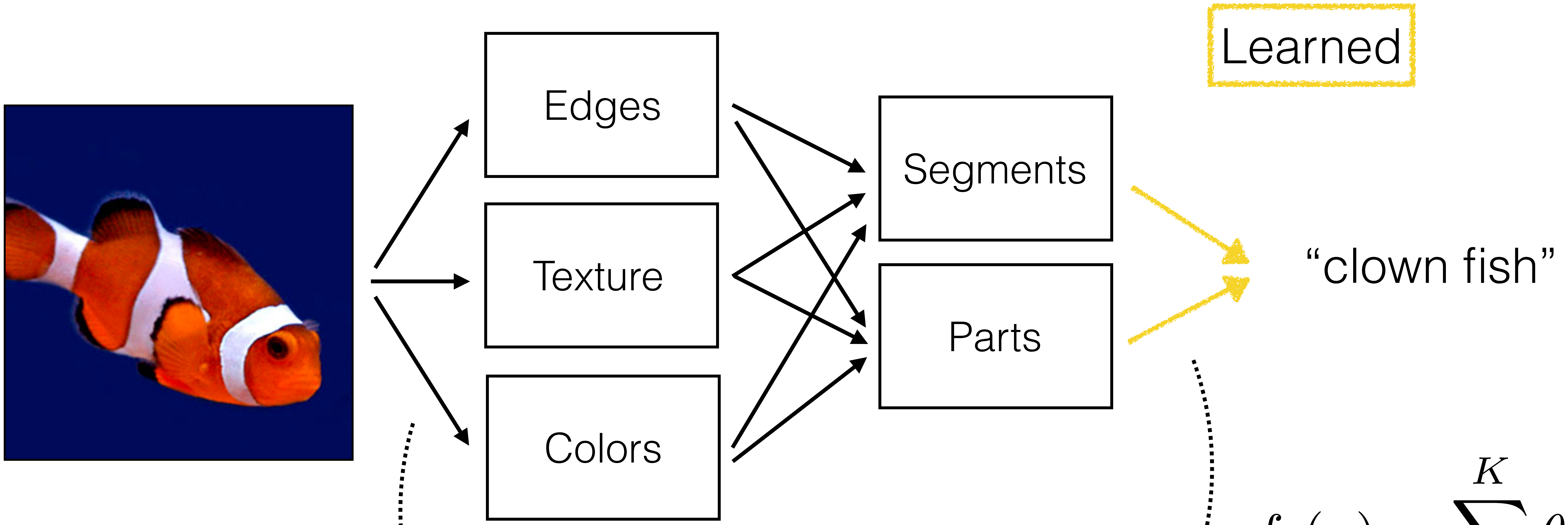


edge orientation histogram



[Xiao et al., “SUN database”, 2010]

“Classic” object recognition **without** neural nets



$$\phi_k(x)$$

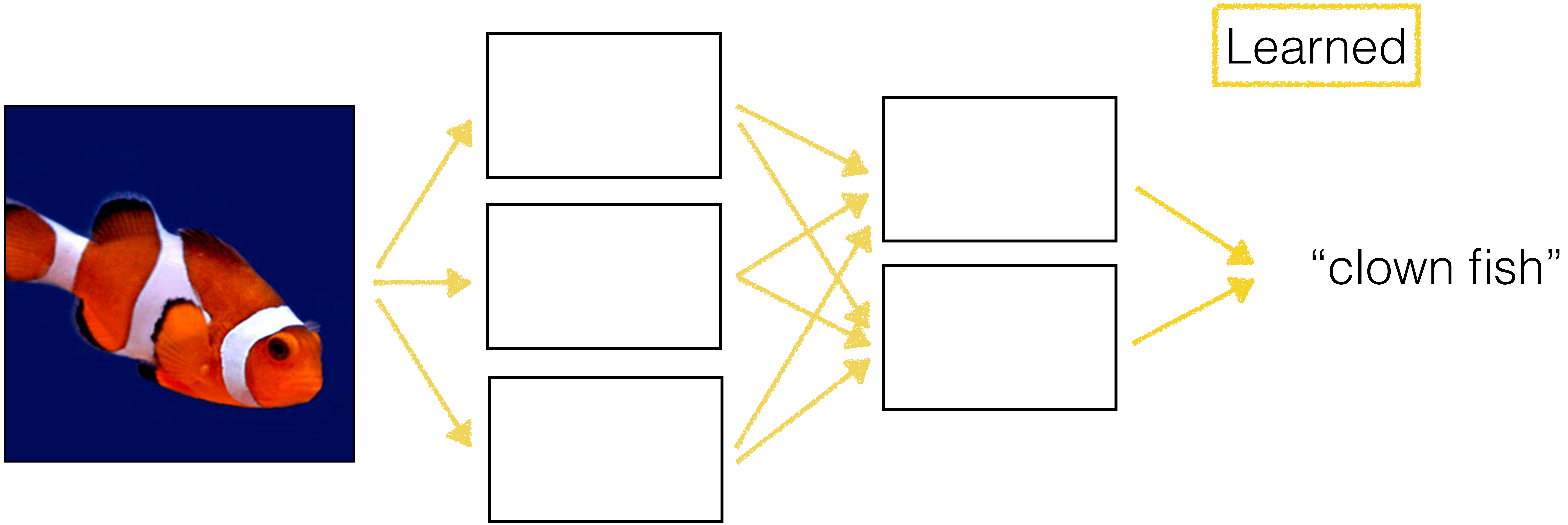
$$f_\theta(x) = \sum_{k=1}^{K} \theta_k \phi_k(x)$$

Feature extractors

Classifier

Source: Isola, Torralba, Freeman

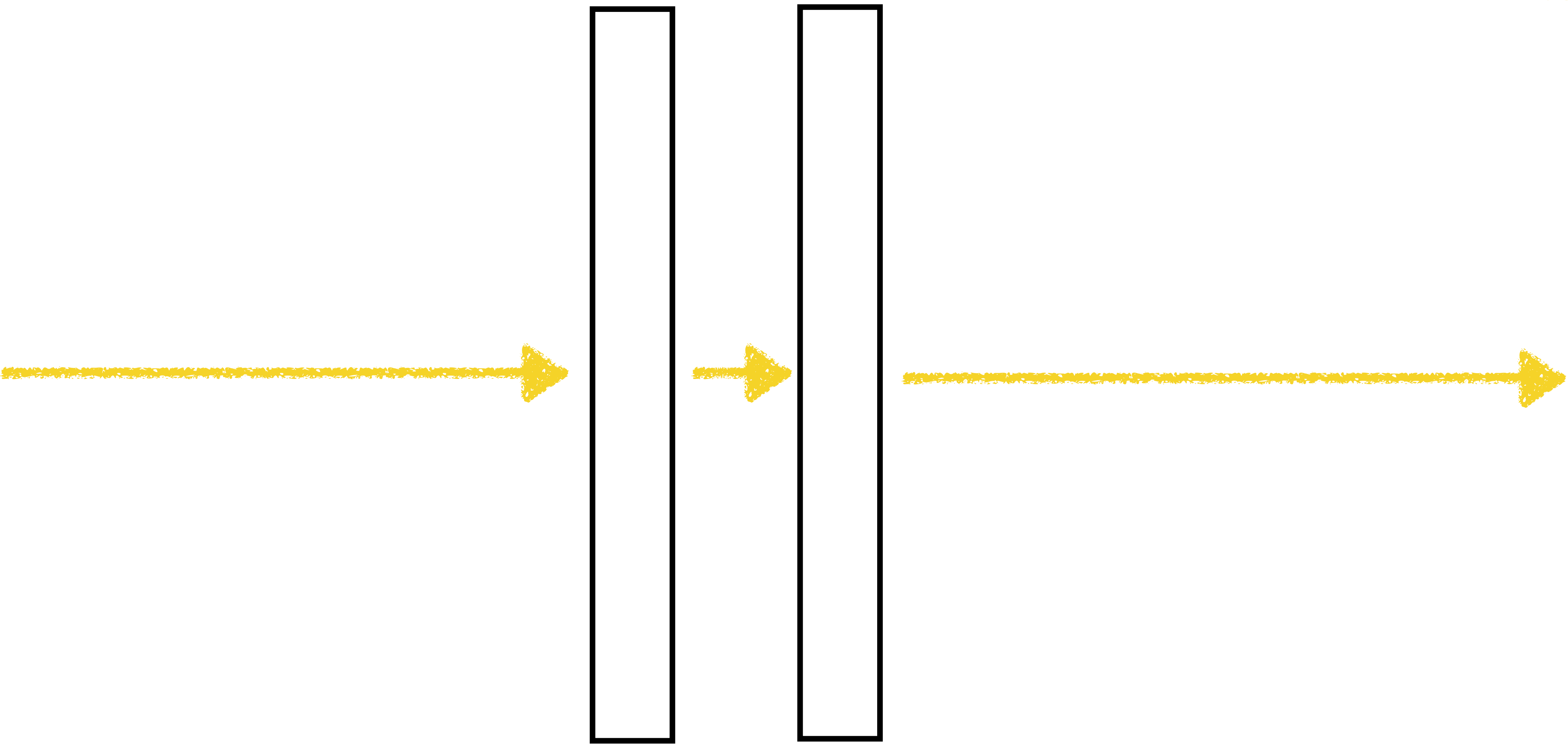
Object recognition



Source: Isola, Torralba, Freeman

Object recognition

Learned

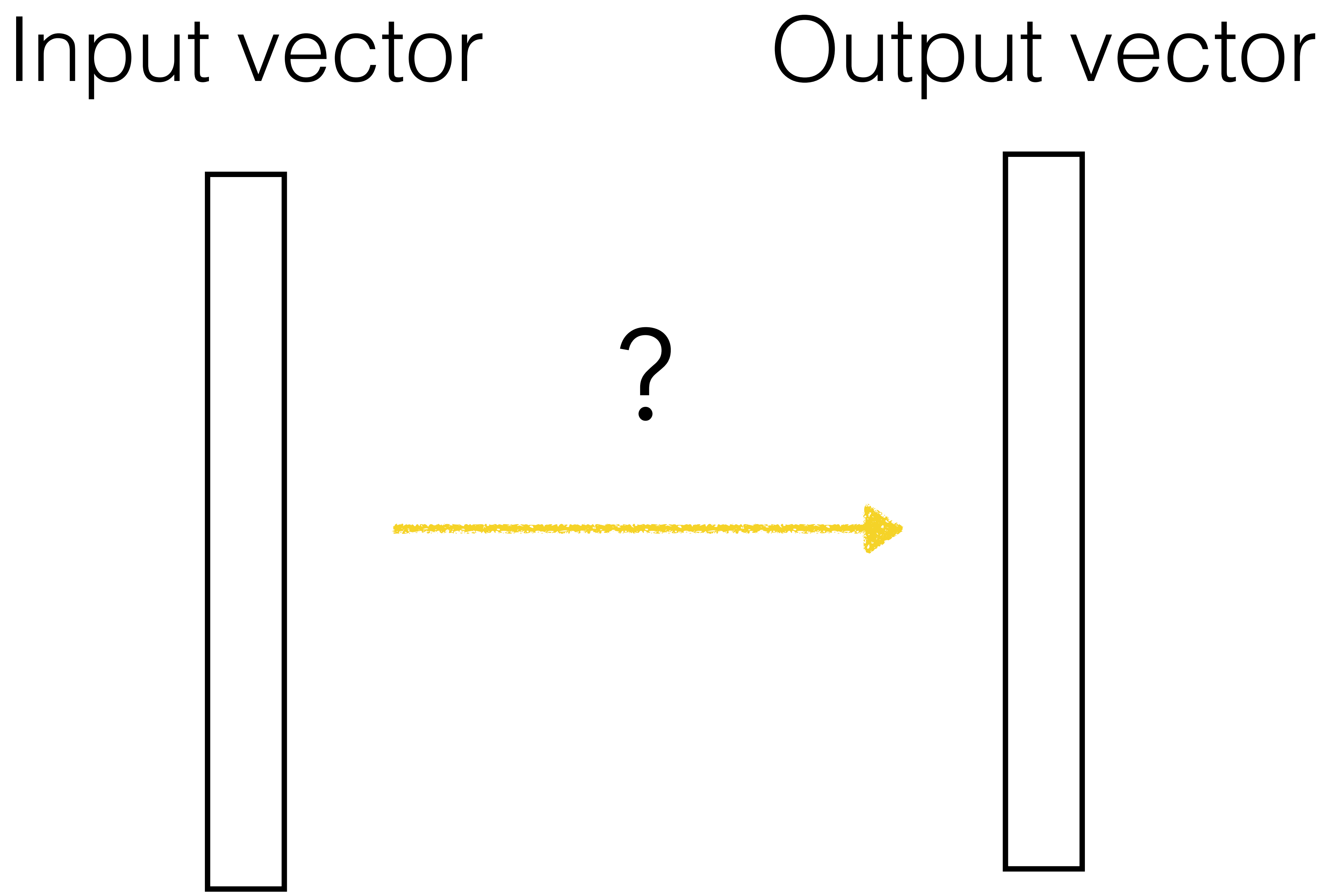


“clown fish”

Neural net

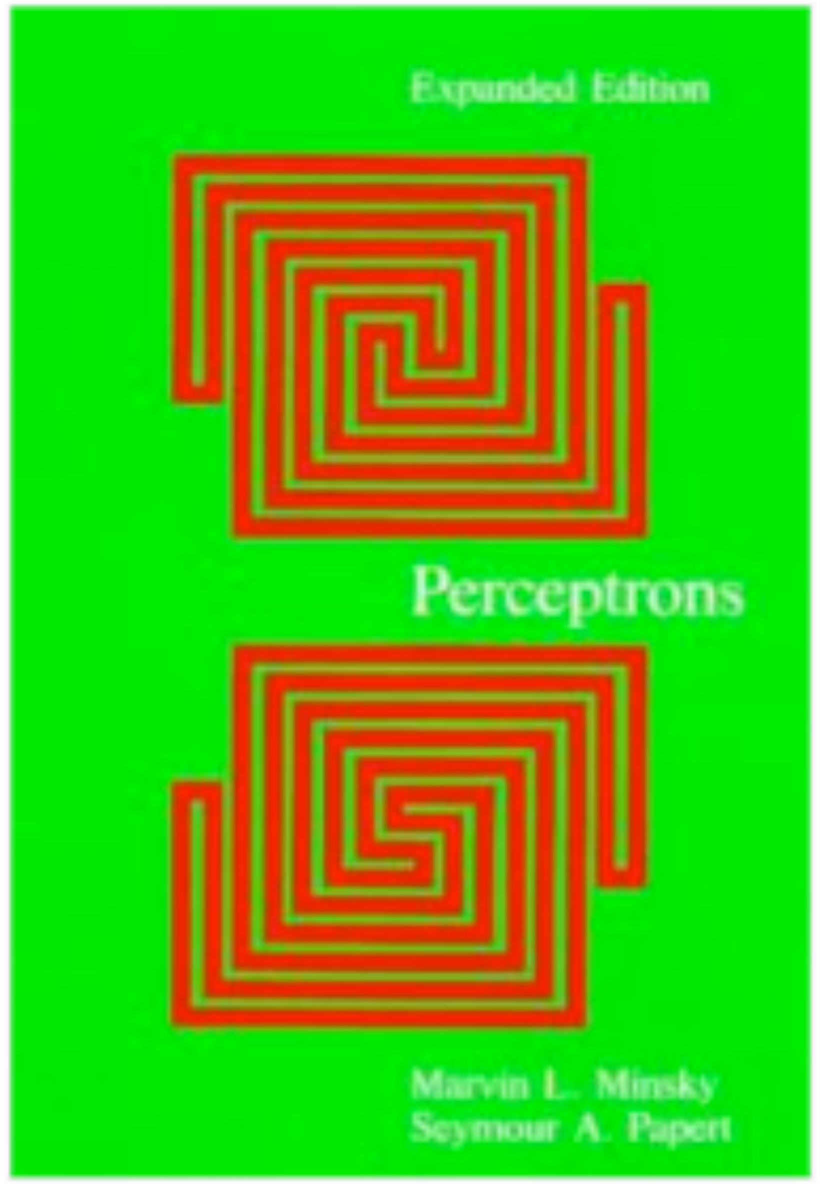
Source: Isola, Torralba, Freeman

Computation in a neural net



Next lecture: neural networks

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

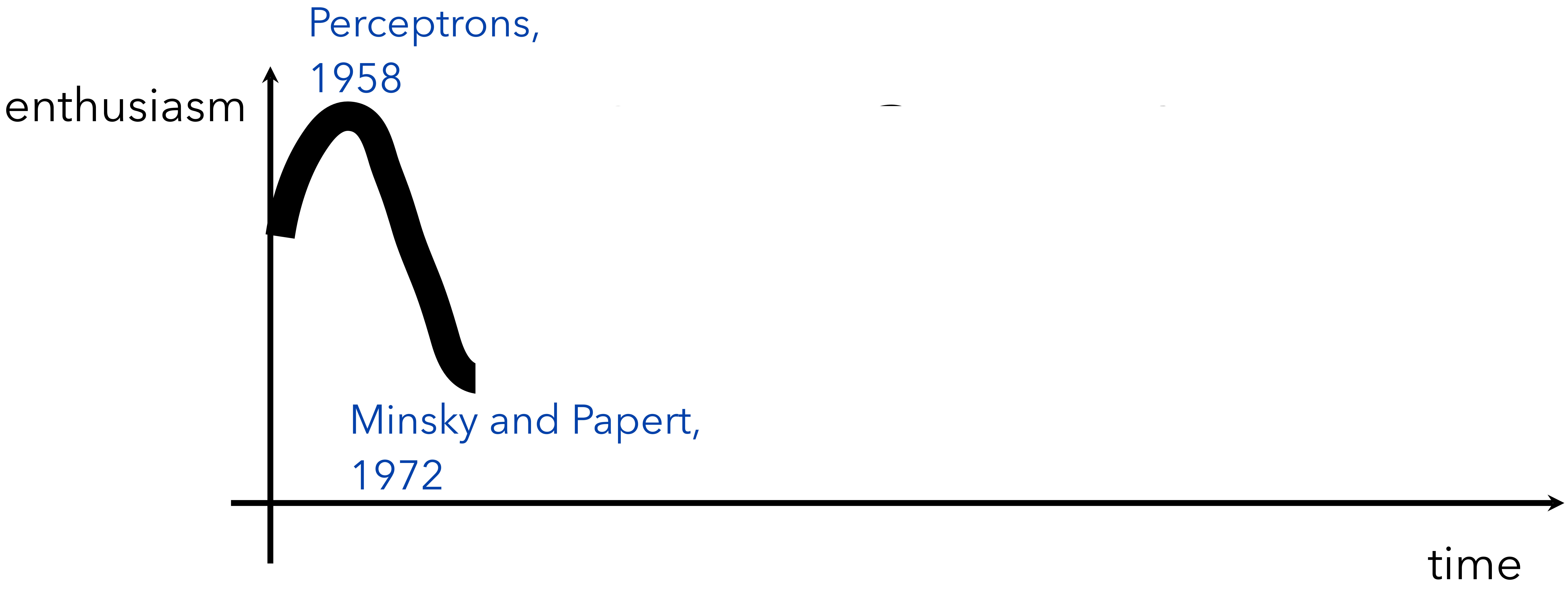
Overview

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.

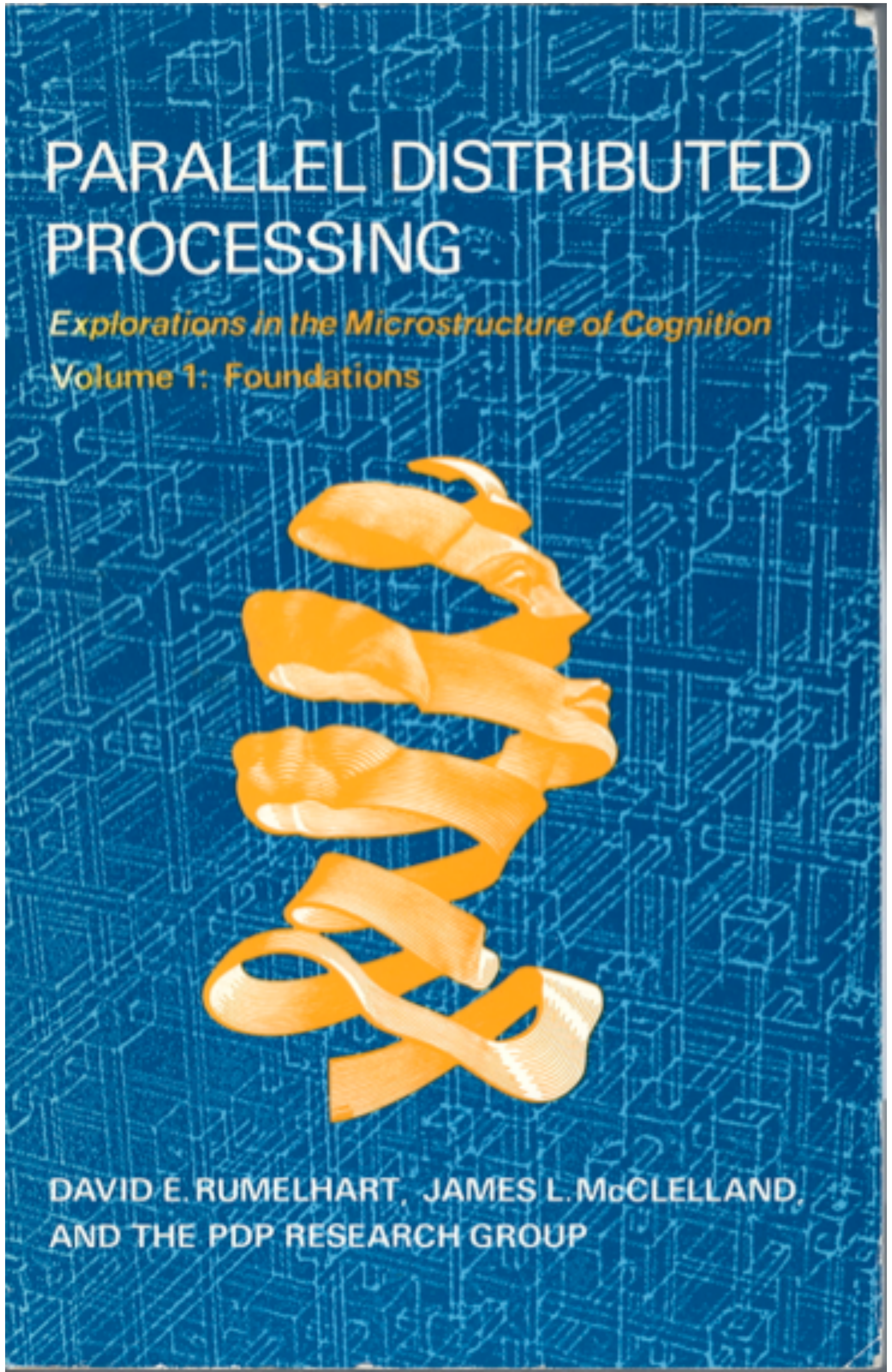
Witnessing the swing of the intellectual pendulum, Minsky and Papert have added a new chapter in which they discuss the current state of parallel computers, review developments since the appearance of the 1972 edition, and identify new research directions related to connectionism. They note a central 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."

Source: Isola, Torralba, Freeman

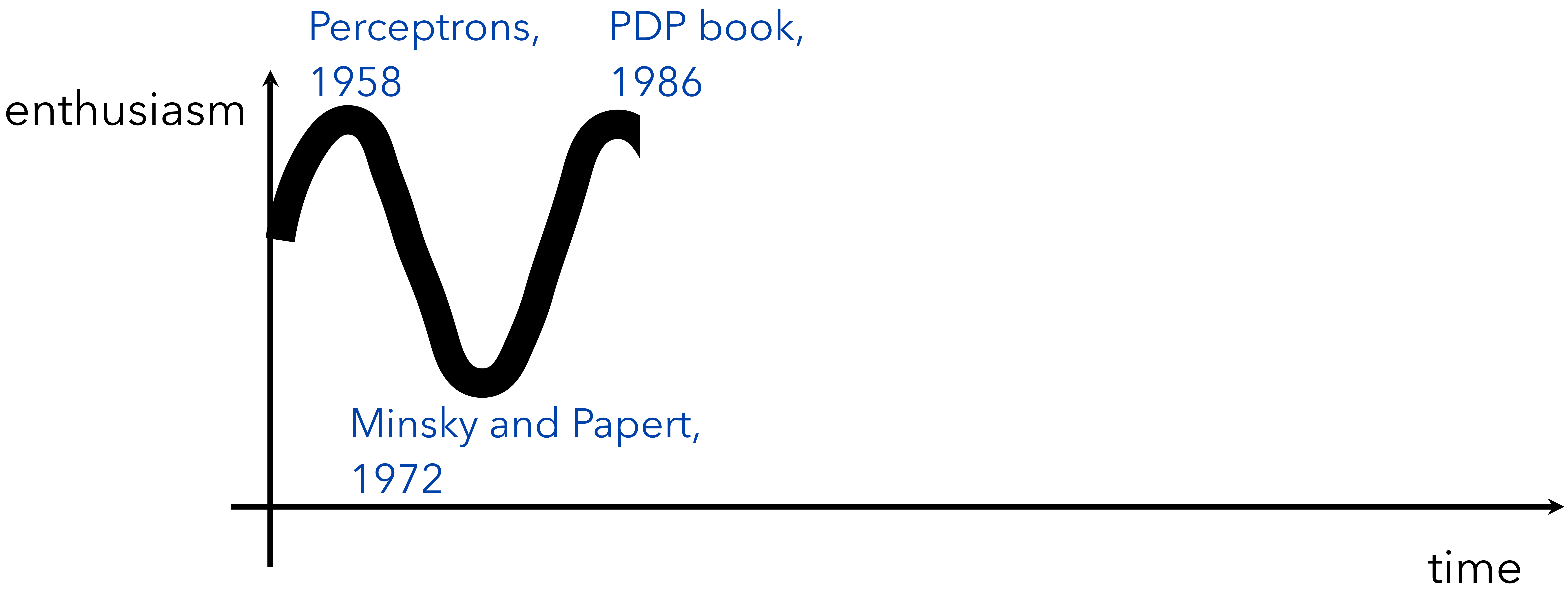


Based on slide by: Isola, Torralba, Freeman

Parallel Distributed Processing (PDP), 1986



Source: Isola, Torralba, Freeman



Source: Isola, Torralba, Freeman

LeCun convolutional neural networks

PROC. OF THE IEEE, NOVEMBER 1998 7

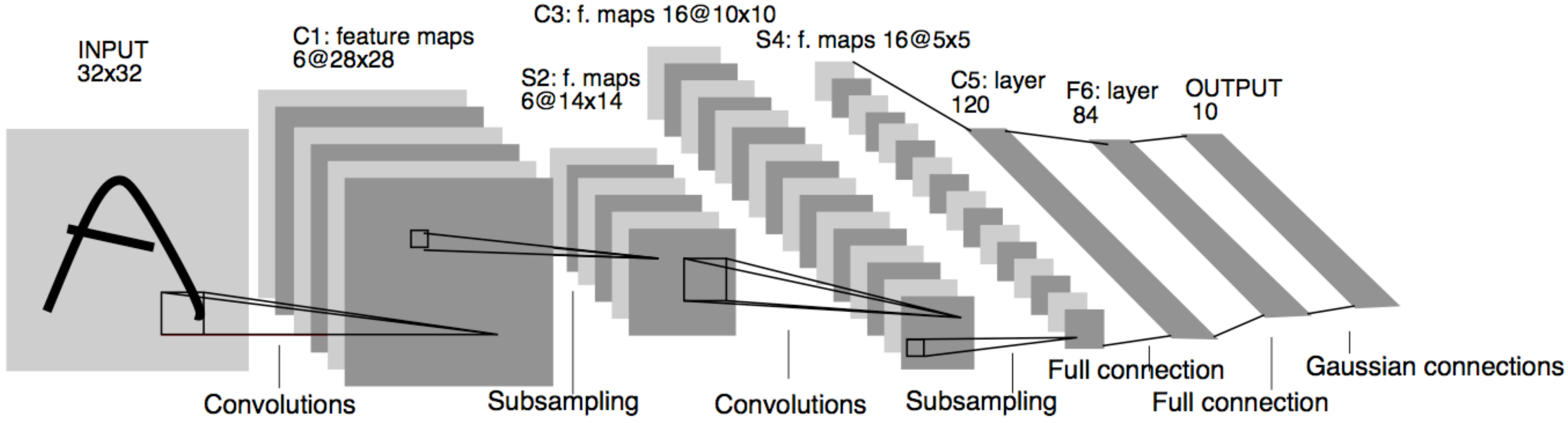


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

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

Source: Isola, Torralba, Freeman

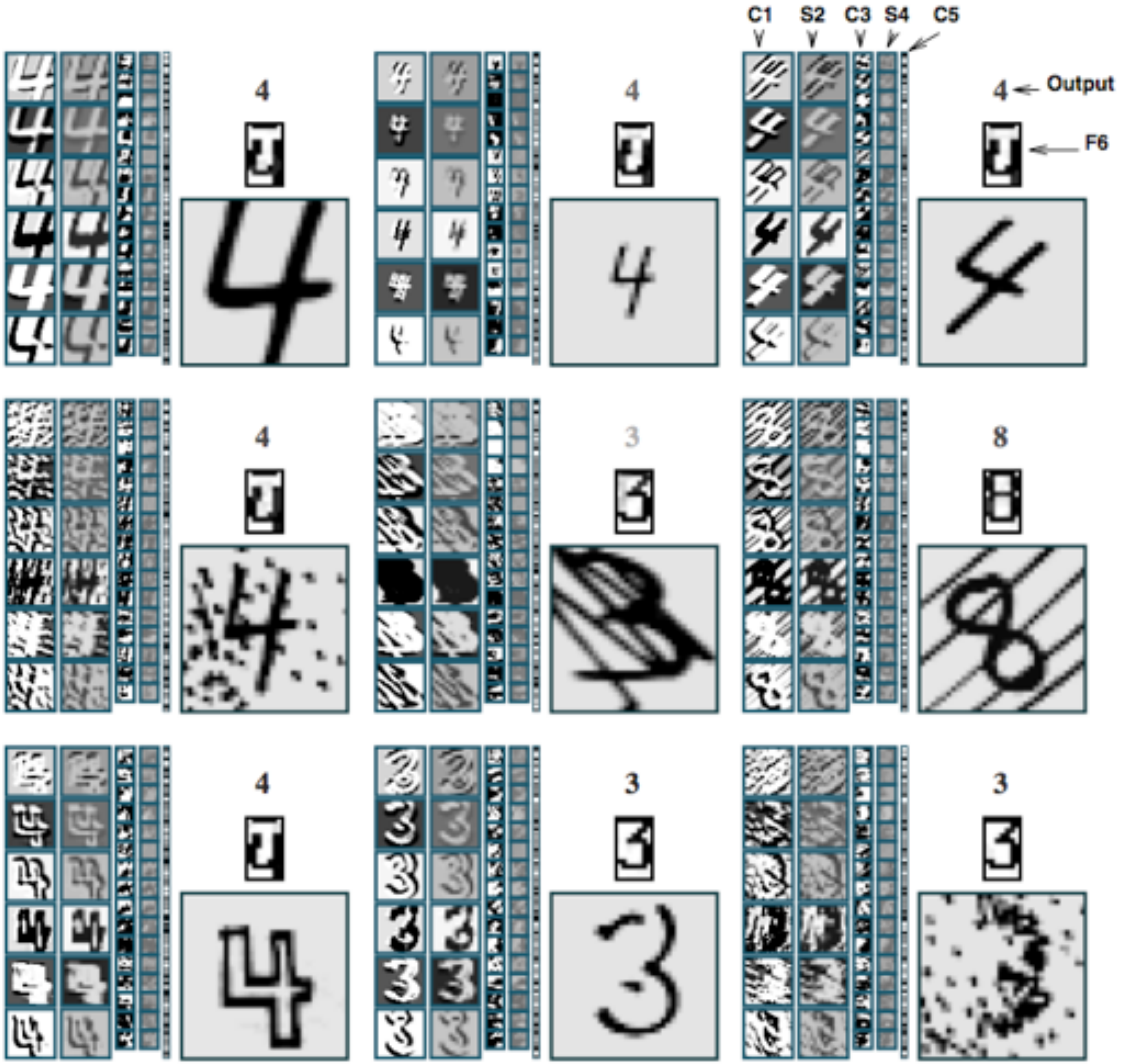
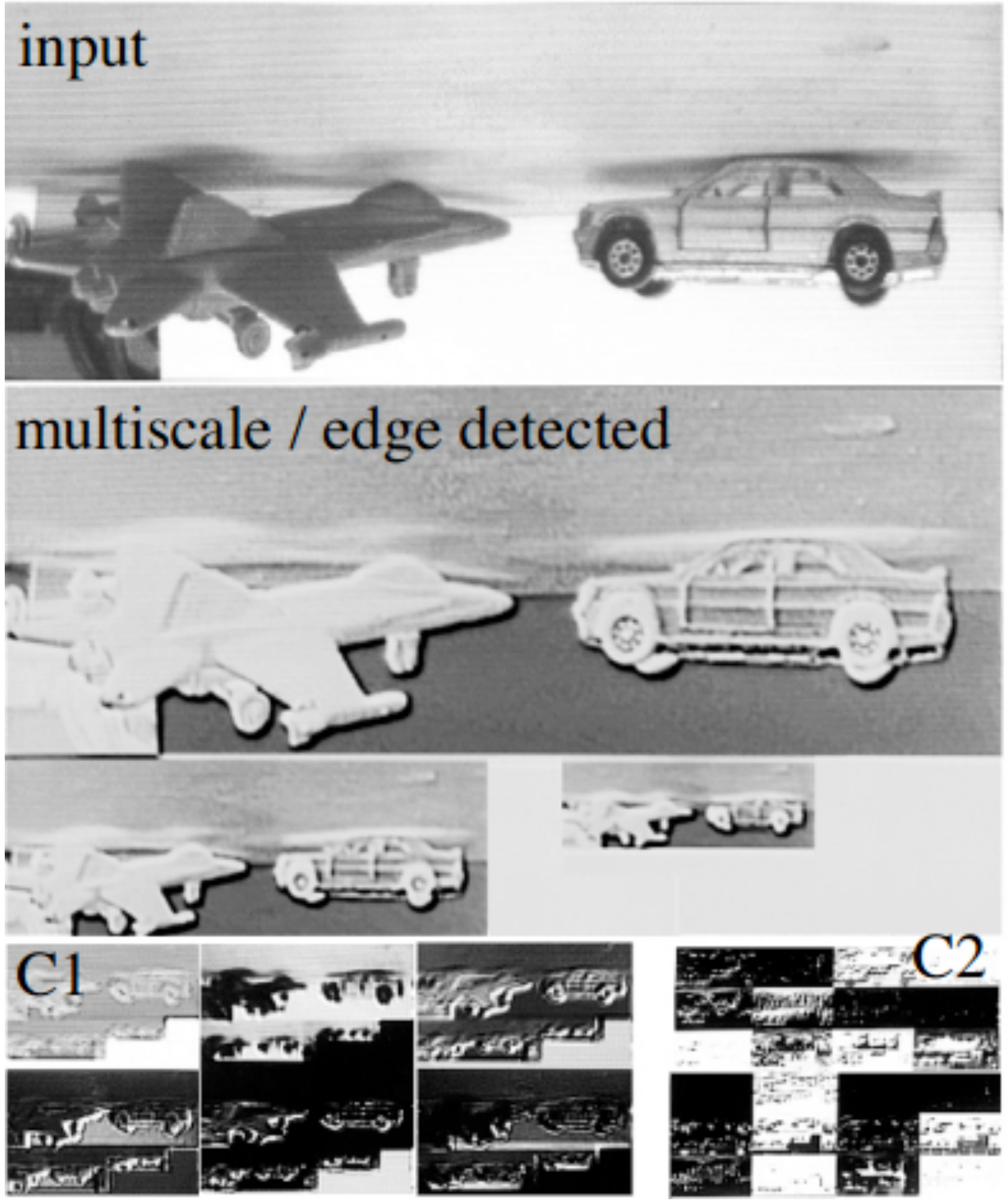


Fig. 13. Examples of unusual, distorted, and noisy characters correctly recognized by LeNet-5. The grey-level of the output label represents the penalty (lighter for higher penalties).

Source: Isola, Torralba, Freeman



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

Neural networks to recognize handwritten digits and human faces?

yes

Neural networks for tougher problems?

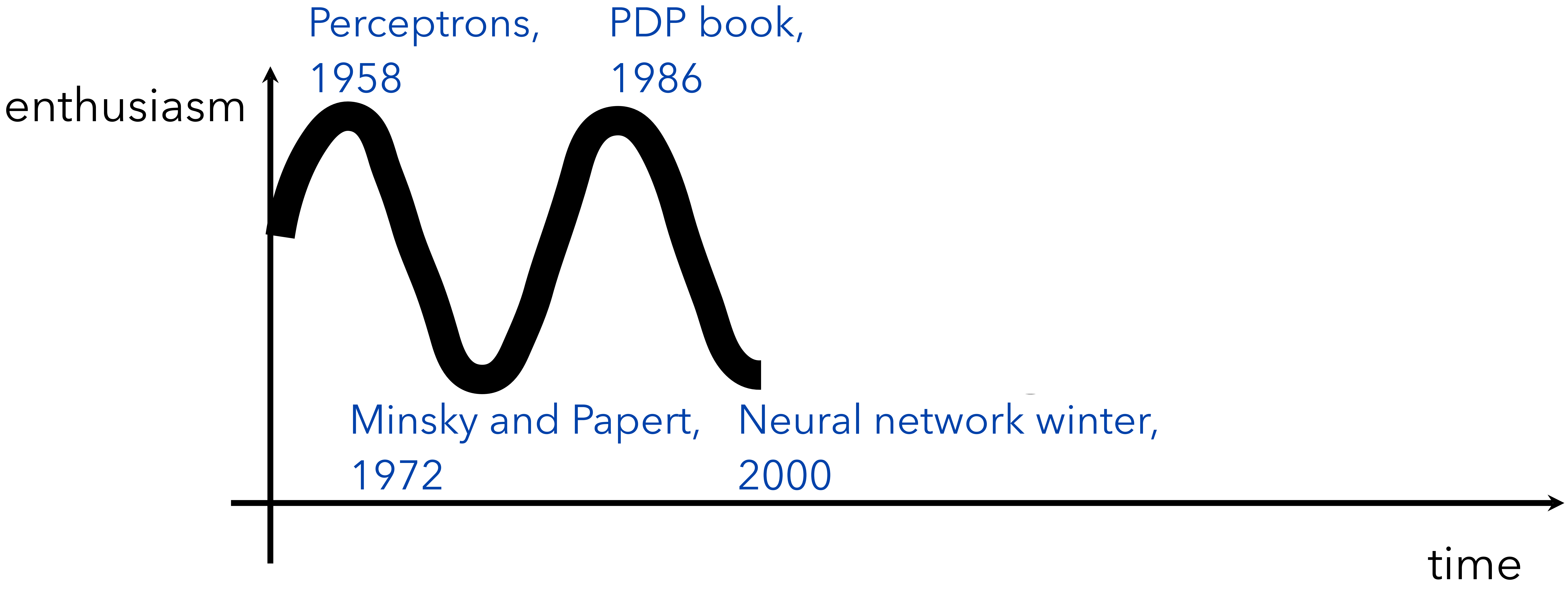
not really

Source: Isola, Torralba, Freeman

Machine learning circa 2000

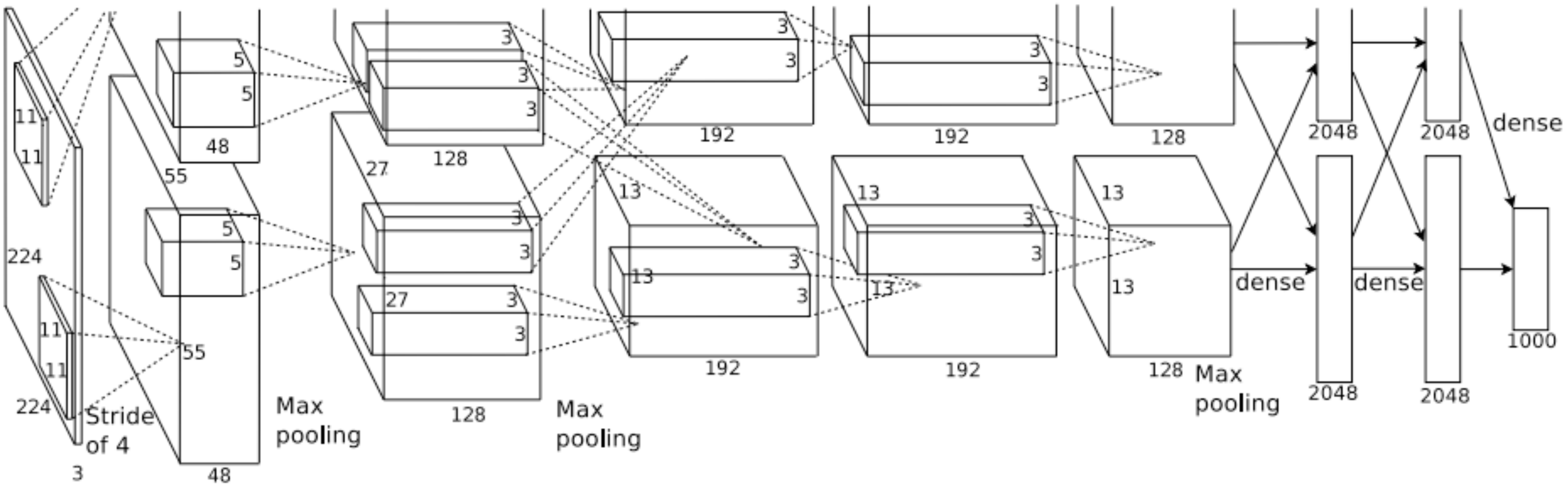
- Neural Information Processing Systems (NeurIPS), is a top conference on machine learning.
- For the 2000 conference:
  - <u>title words predictive of paper acceptance</u>: "Belief Propagation" and "Gaussian".
  - <u>title words predictive of paper rejection</u>: "Neural" and "Network".

Source: Isola, Torralba, Freeman



Source: Isola, Torralba, Freeman

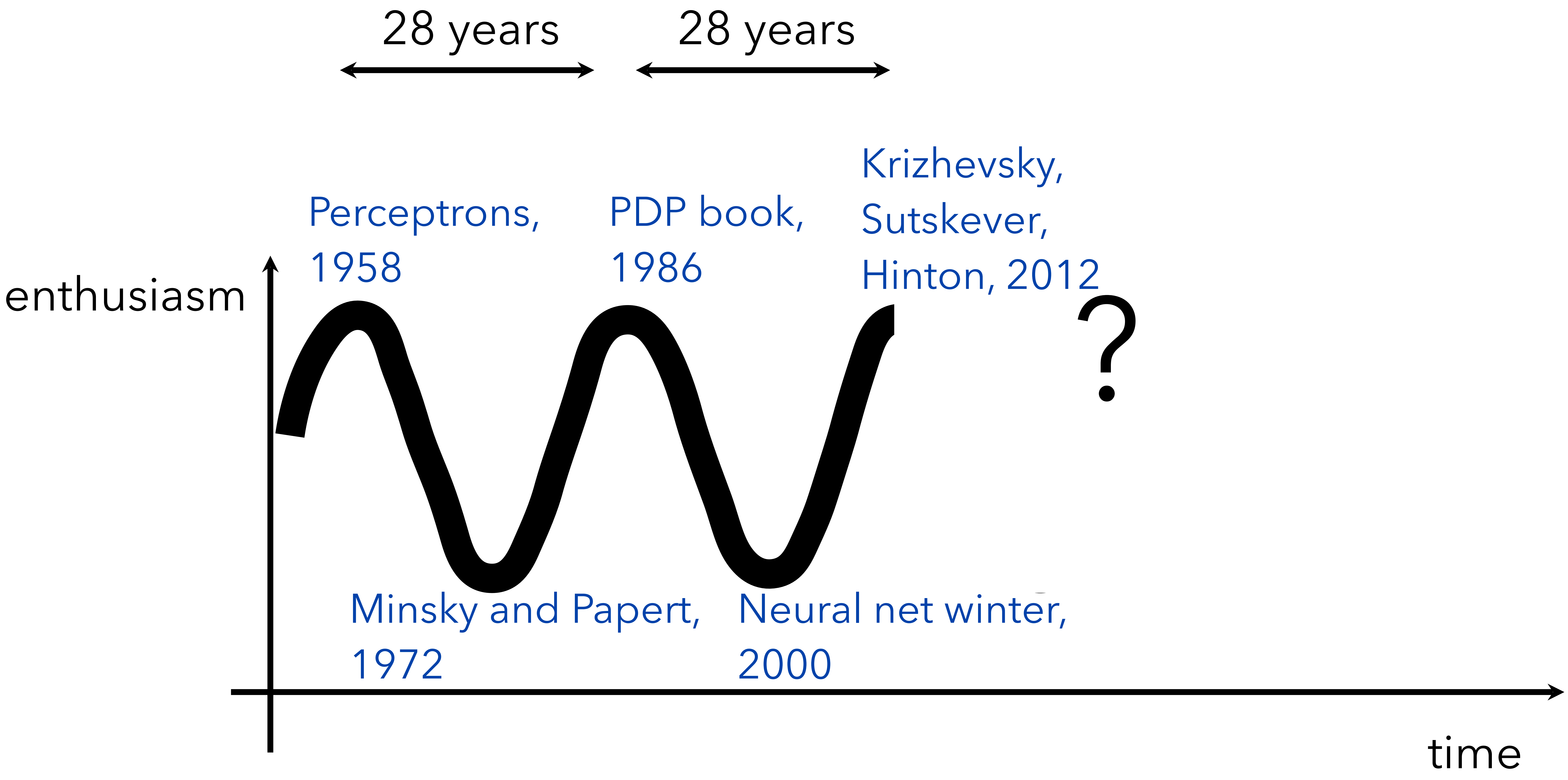
Krizhevsky, Sutskever, and Hinton, NeurIPS 2012
"AlexNet"



Source: Isola, Torralba, Freeman

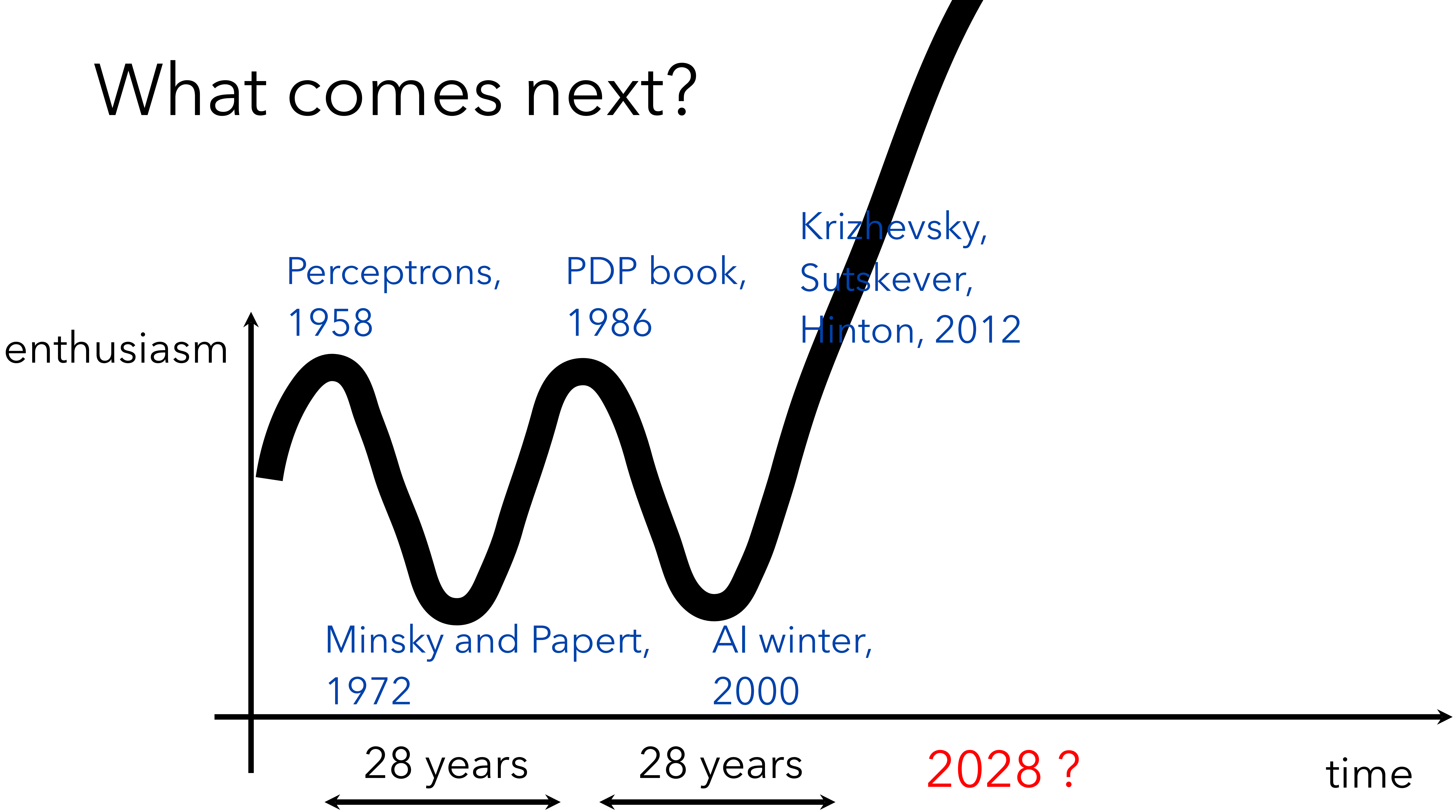
Krizhevsky, Sutskever, and Hinton, NeurIPS 2012

Source: Isola, Torralba, Freeman



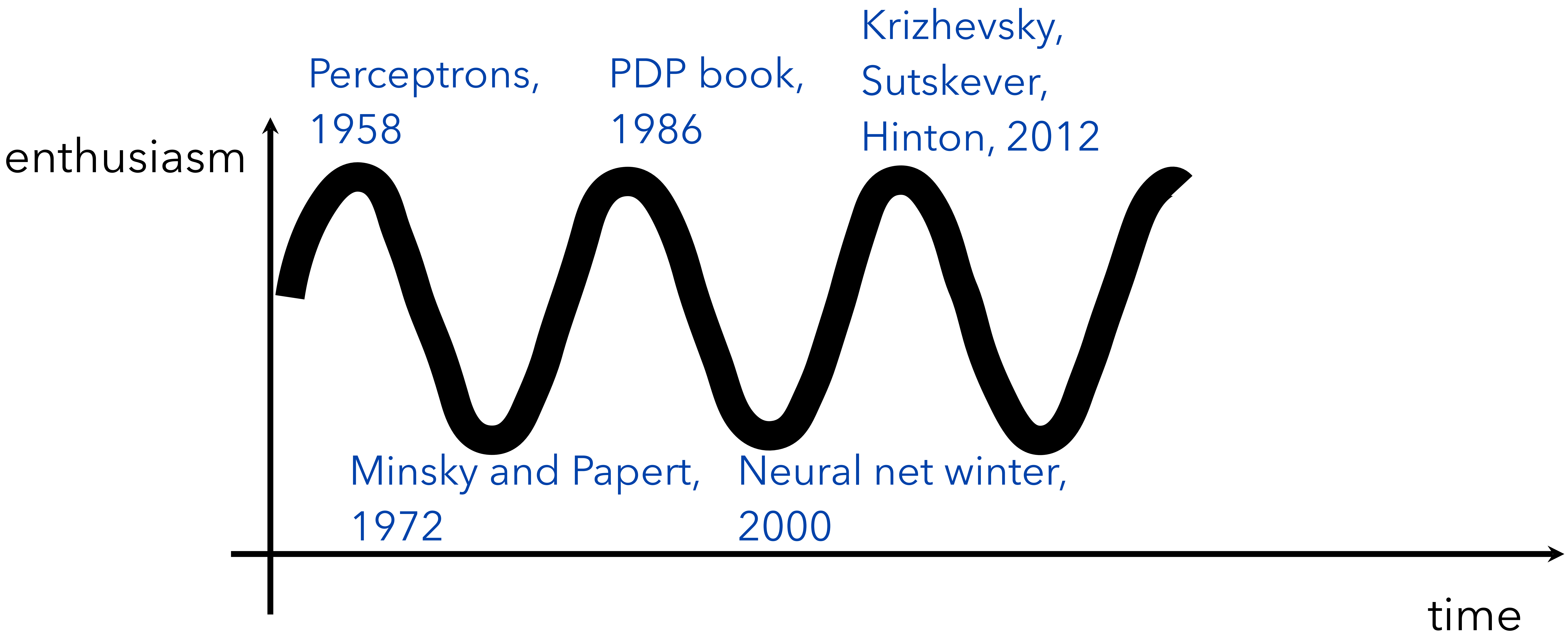
Source: Isola, Torralba, Freeman

What comes next?



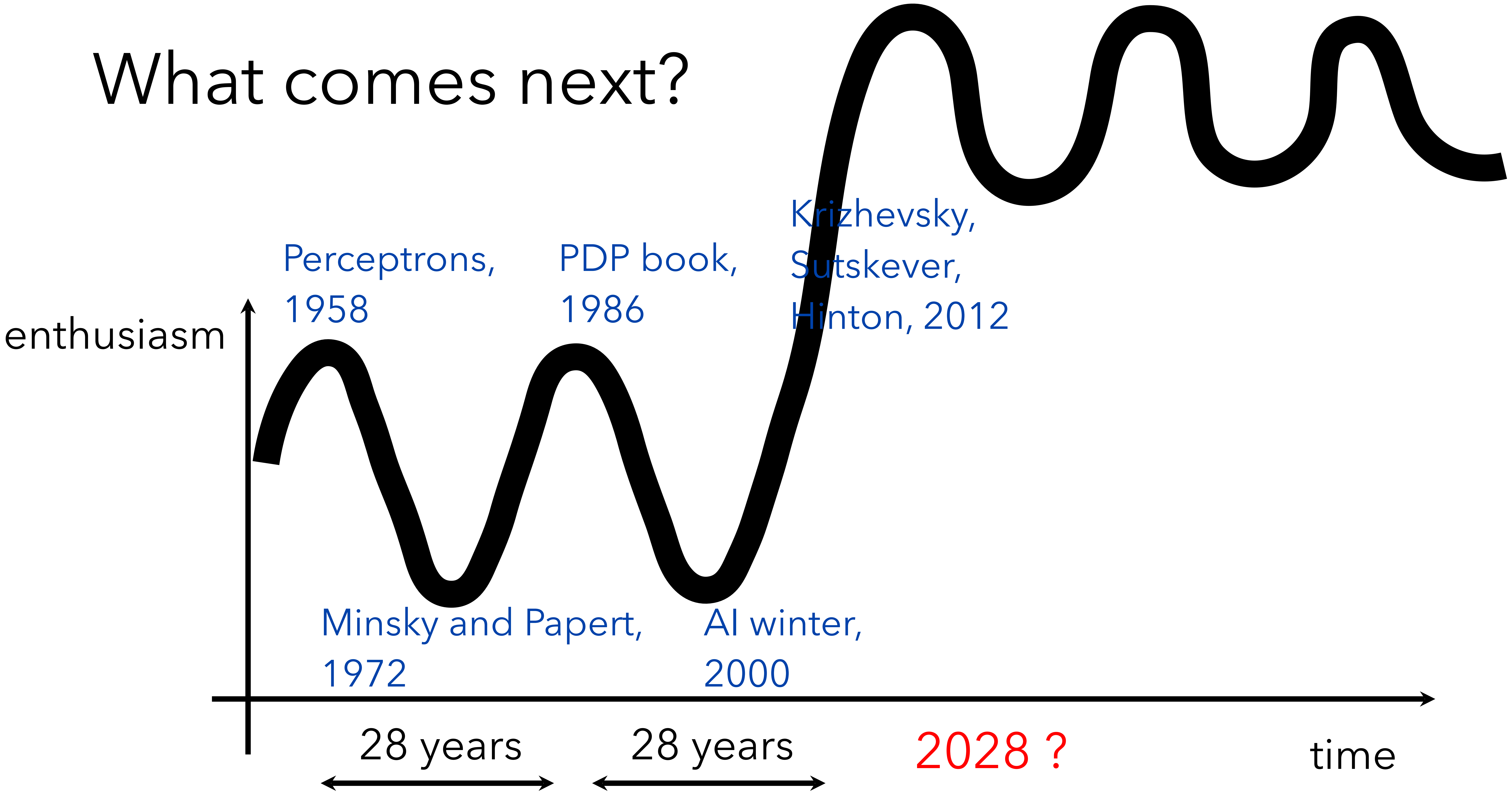
Source: Isola, Torralba, Freeman

What comes next?



Source: Isola, Torralba, Freeman

What comes next?



Source: Isola, Torralba, Freeman

Inspiration: Neurons

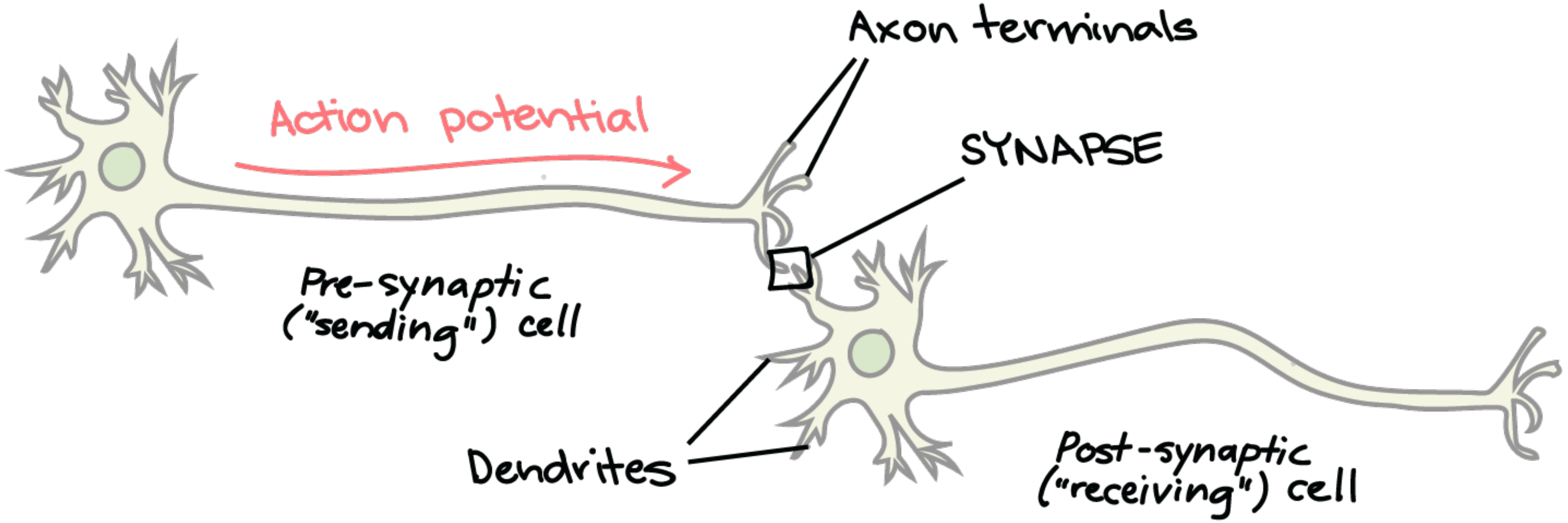
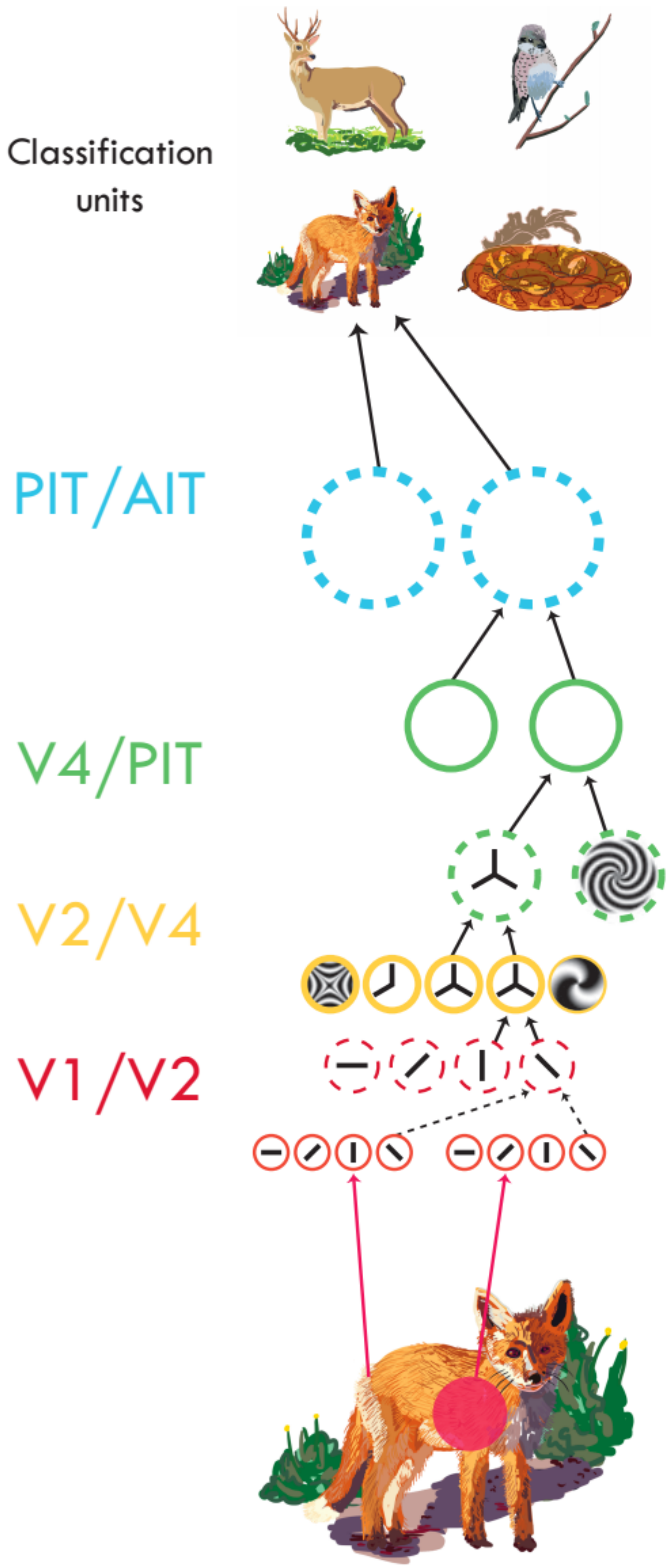


Image source: Khan academy

Inspiration: Hierarchical Representations

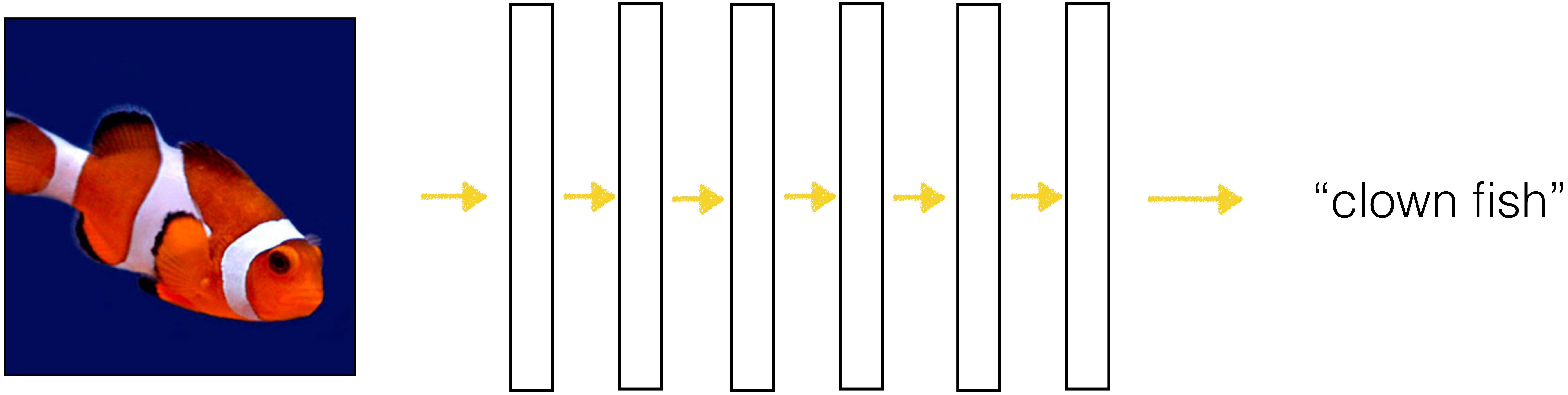


[Serre, 2014]

Best to treat as *inspiration*. The neural nets we'll talk about aren't very biologically plausible.

Source: Isola, Torralba, Freeman

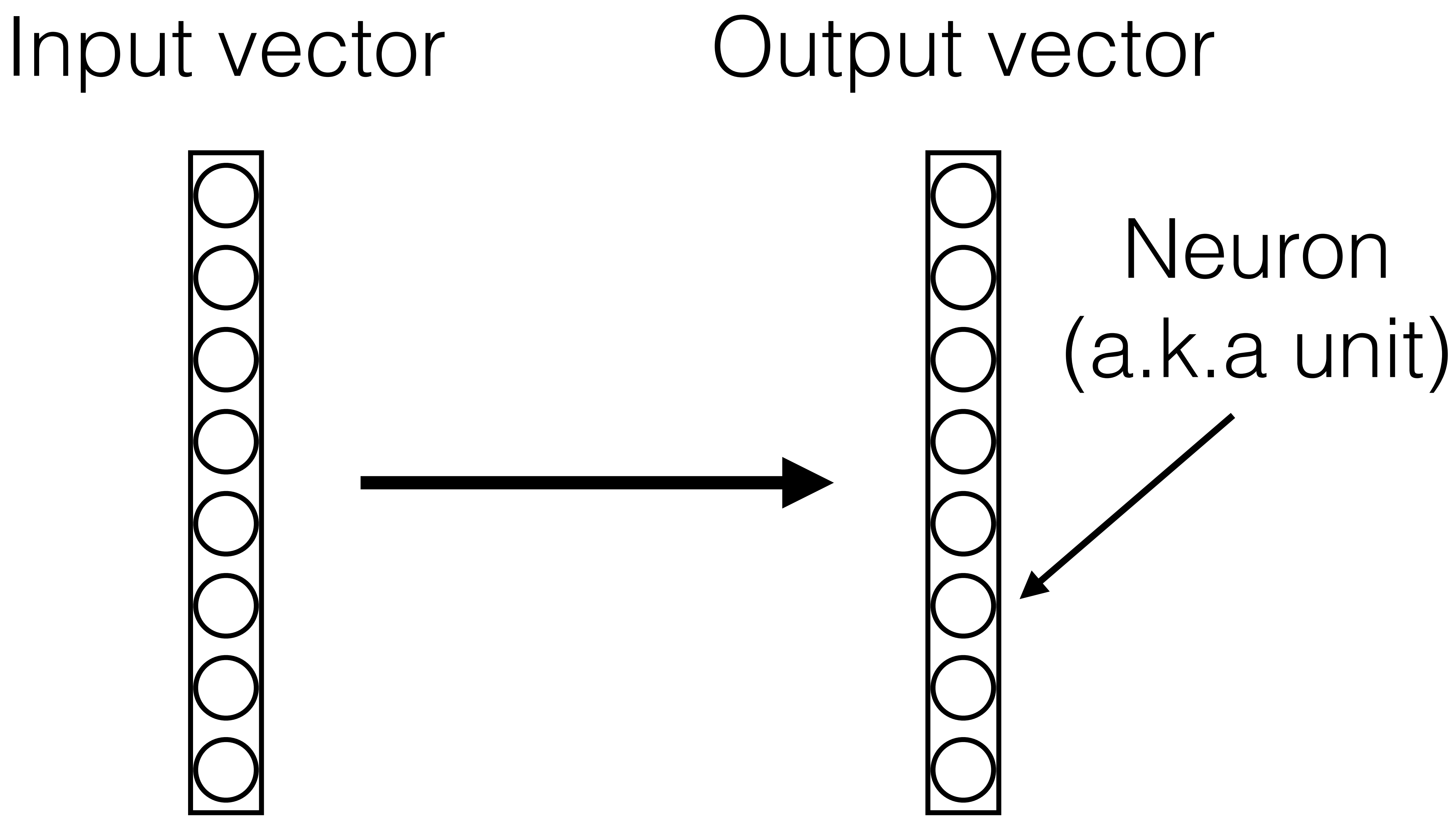
Object recognition



Neural network

Source: Isola, Torralba, Freeman

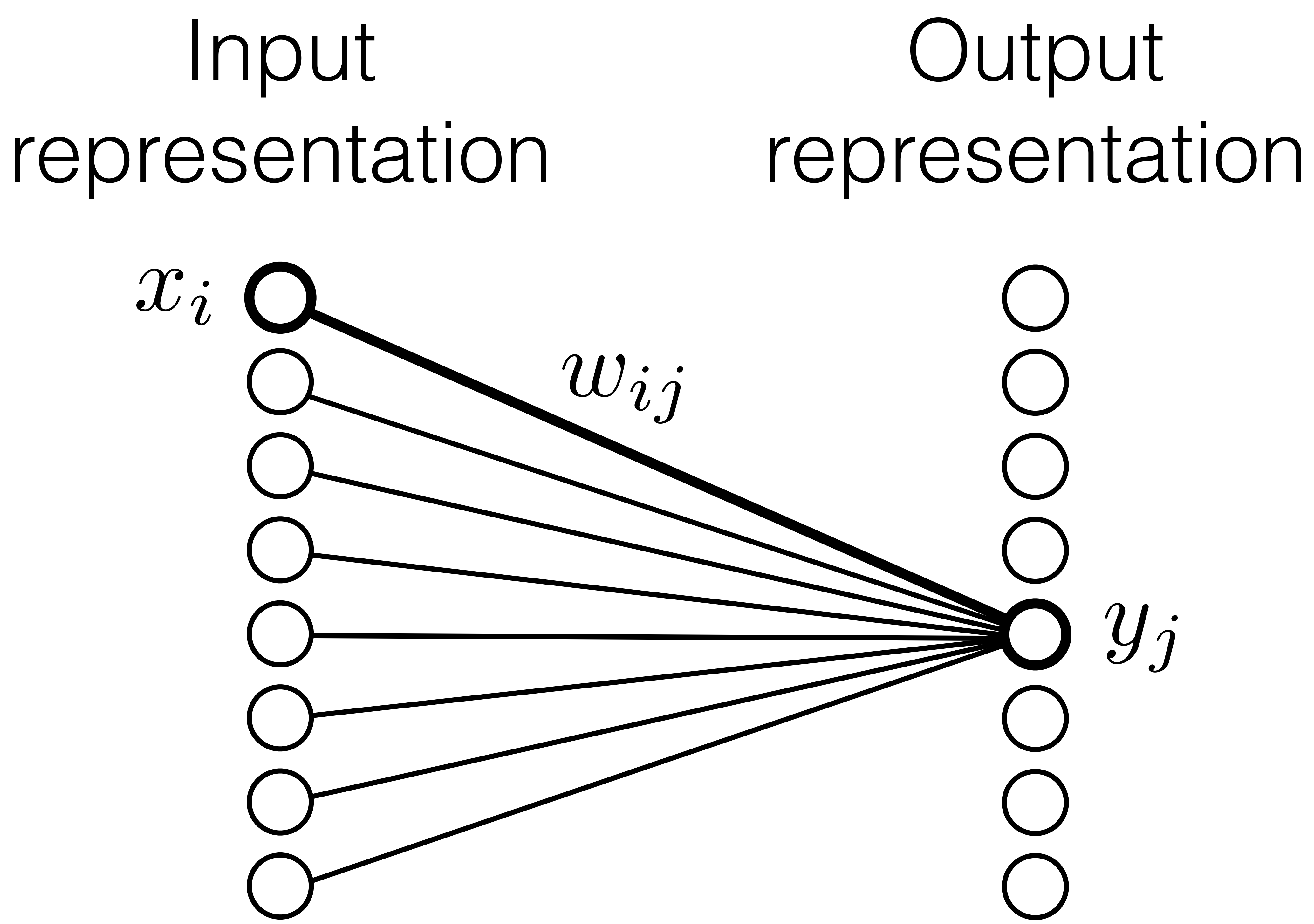
Computation in a neural net



Adapted from: Isola, Torralba, Freeman

Computation in a neural net

Linear layer

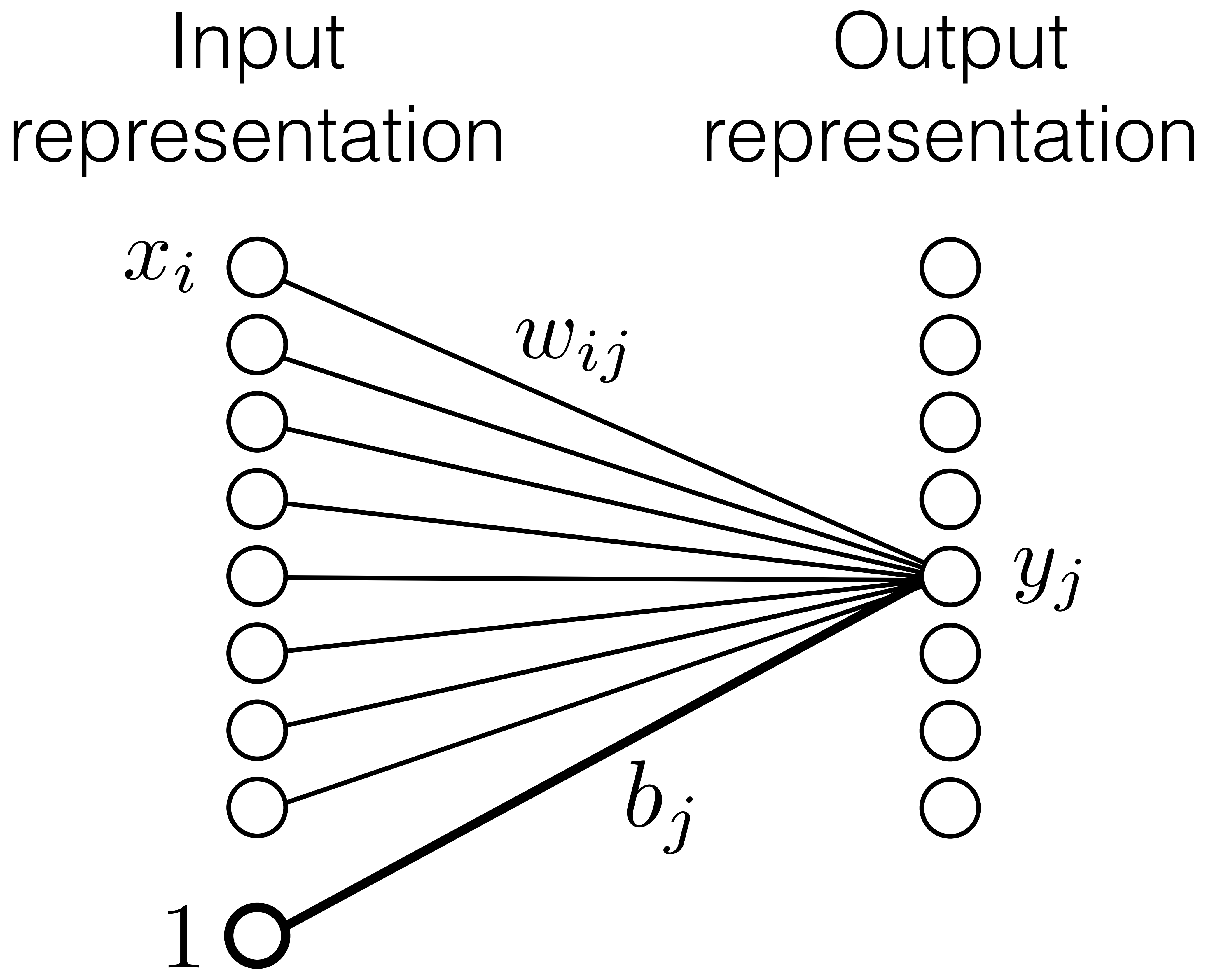


$$y_j = \sum_i w_{ij} x_i$$

Source: Isola, Torralba, Freeman

Computation in a neural net

Linear layer



weights

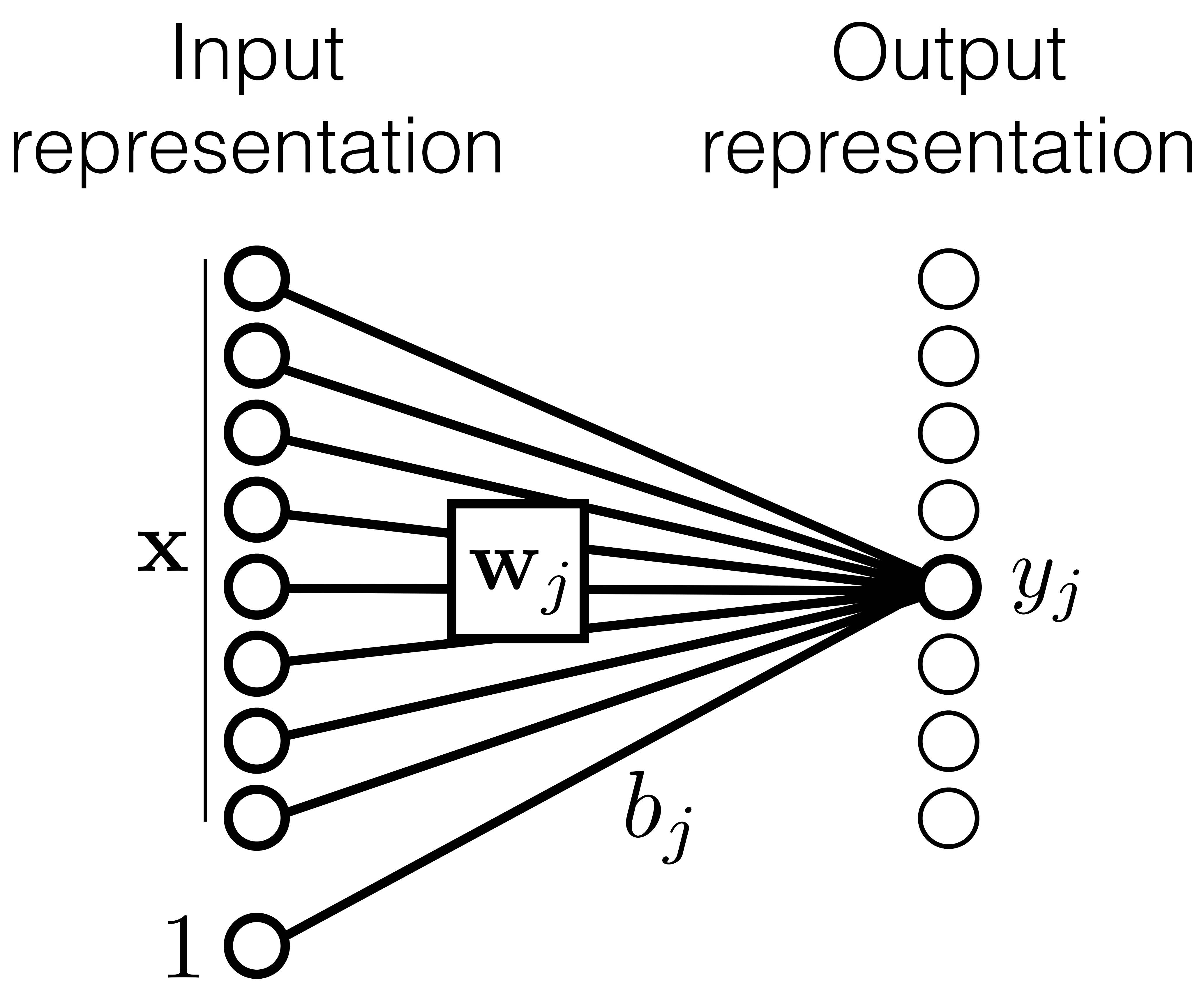
$$y_j = \sum_i w_{ij} x_i + b_j$$

bias

Source: Isola, Torralba, Freeman

Computation in a neural net

Linear layer



weights

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

bias

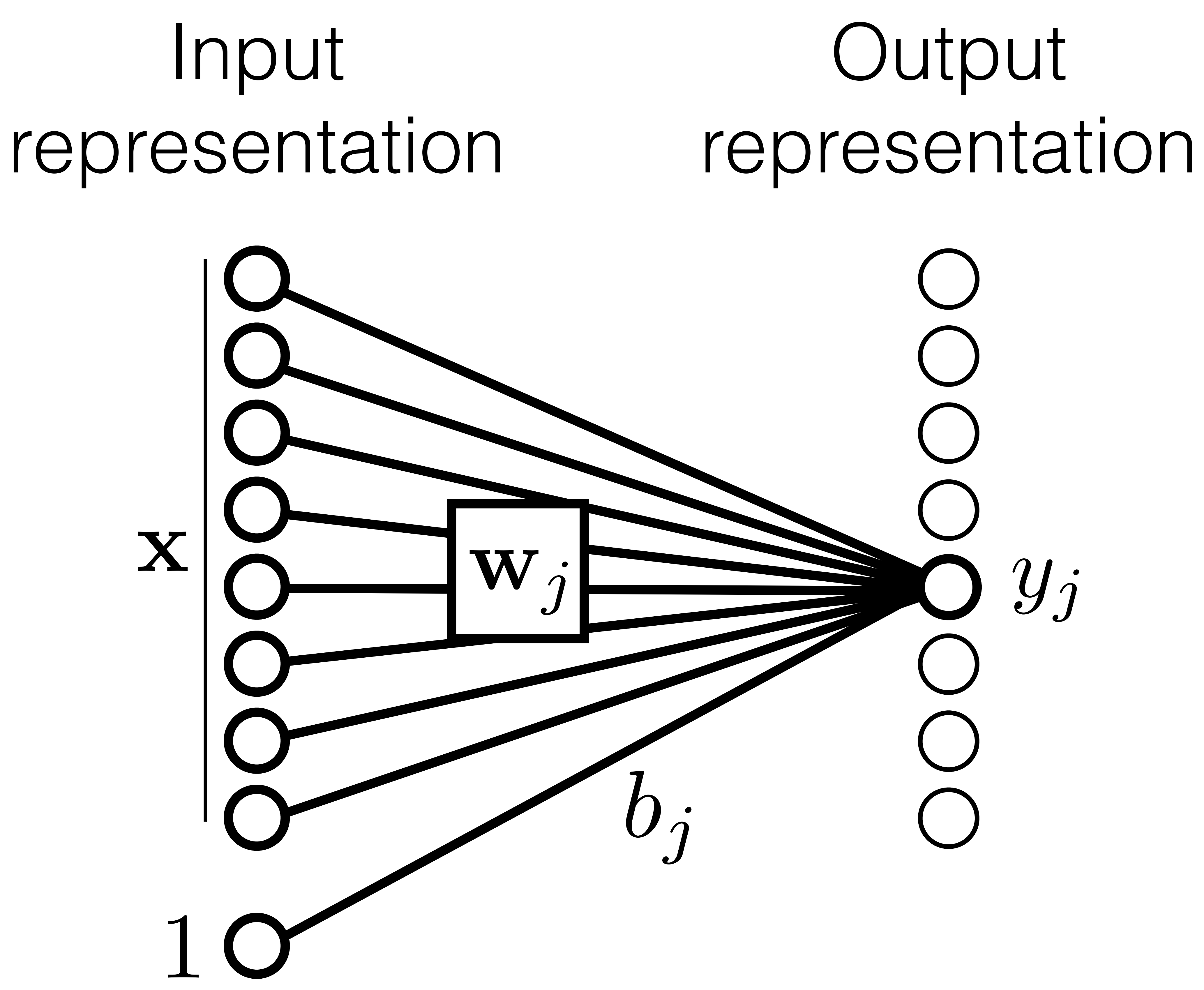
$$\theta = \{\mathbf{W}, \mathbf{b}\}$$

parameters of the model

Source: Isola, Torralba, Freeman

Computation in a neural net

Linear layer



Full layer

weights

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

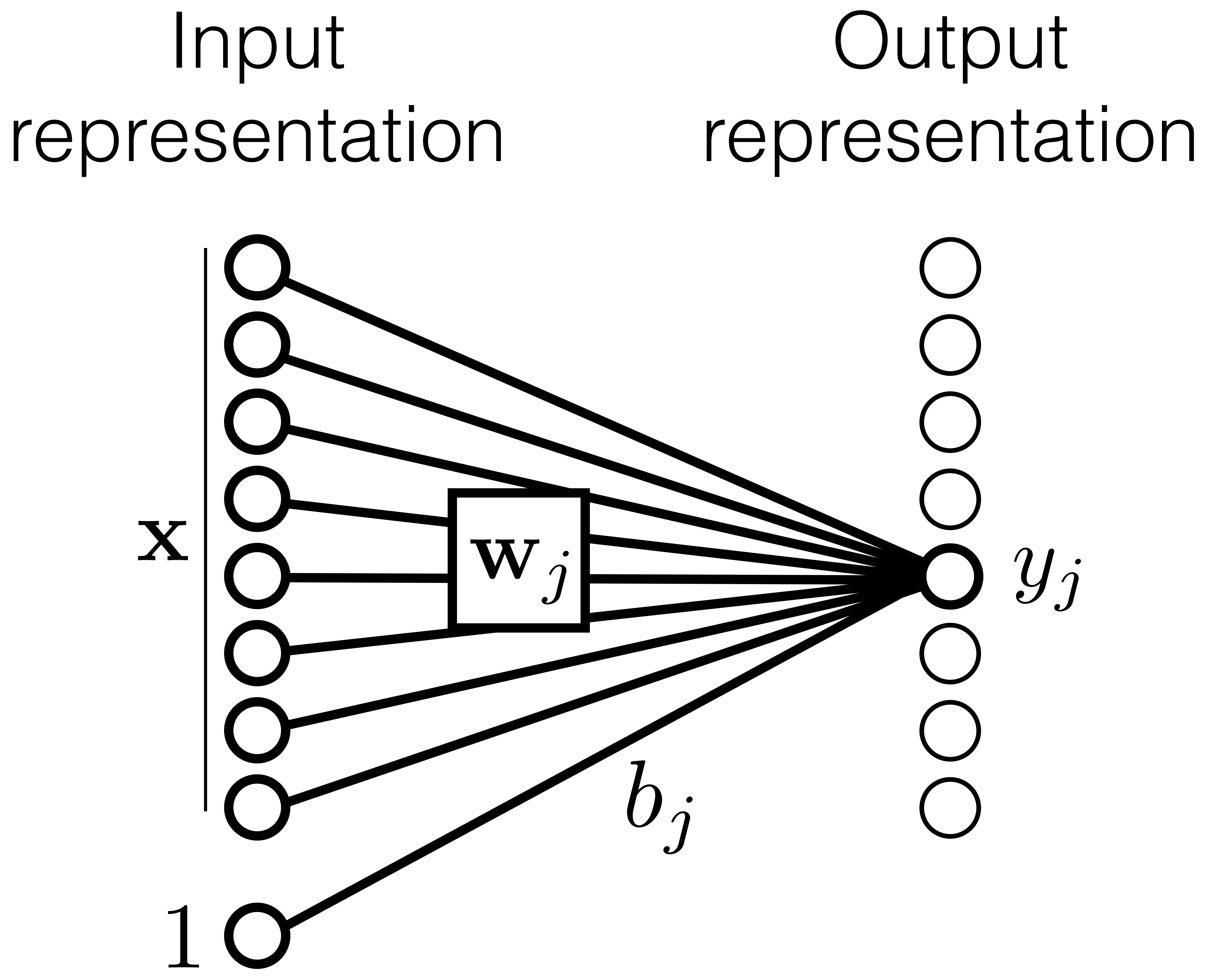
bias

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

parameters of the model

Computation in a neural net

Linear layer



Full layer

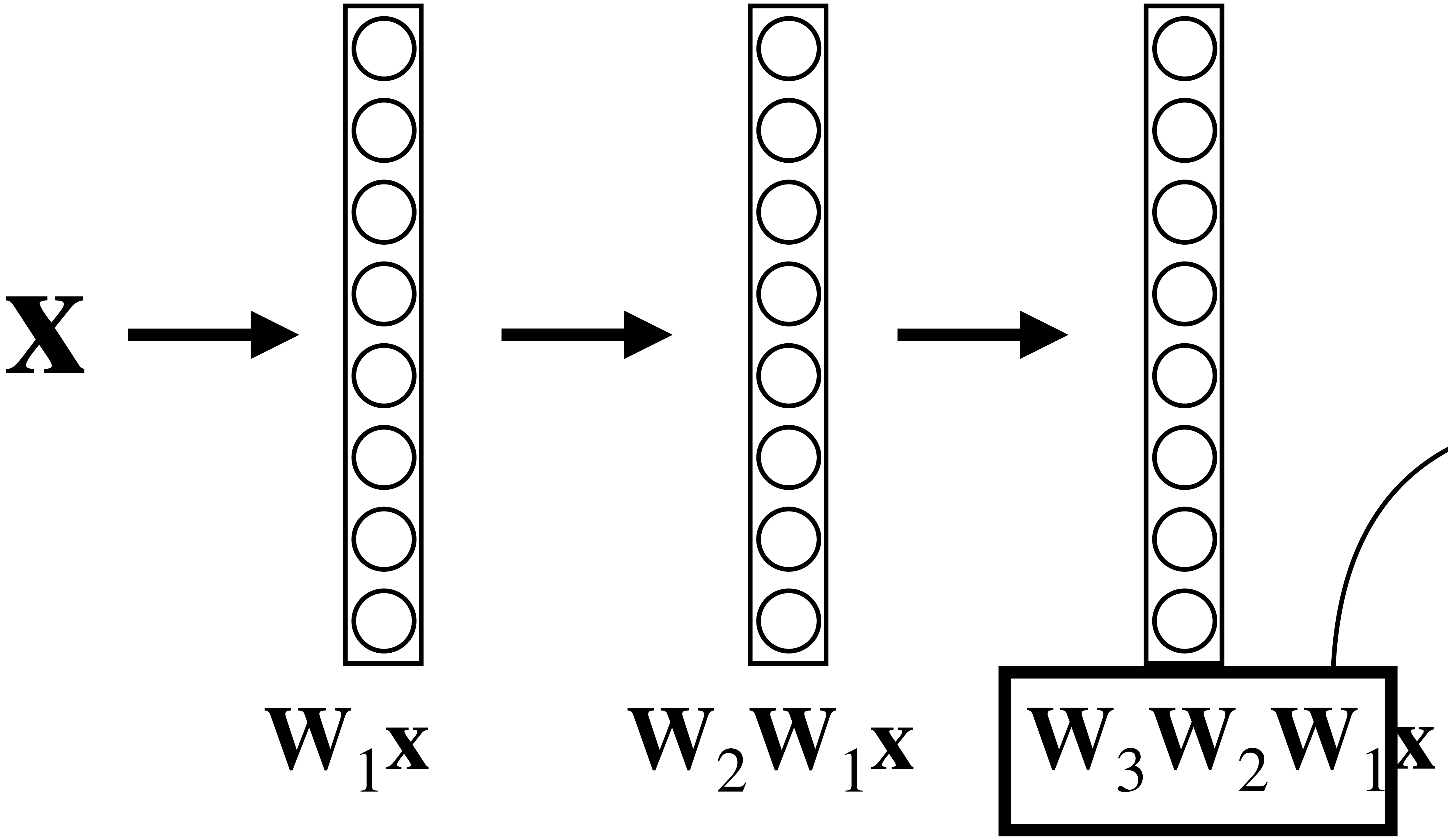
weights + bias

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

Can again simplify notation by appending a 1 to $\mathbf{x}$

What’s the problem with this idea?

Consider stacking multiple layers:



Can be expressed as single linear layer!

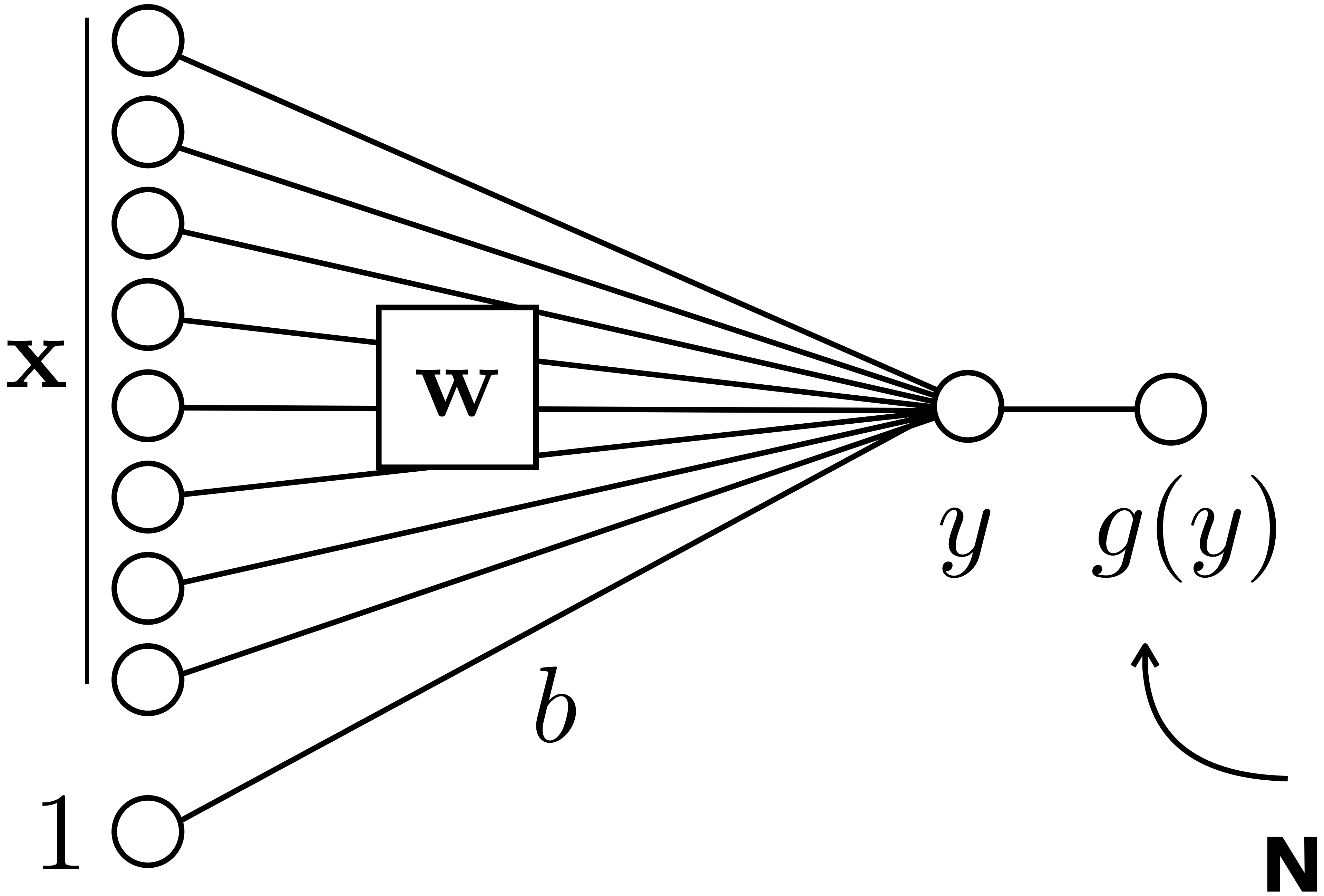
$$\left(\prod_i \mathbf{W}_i\right)\mathbf{x} = \hat{\mathbf{W}}\mathbf{x}$$

Limited power, e.g. can’t solve XOR.

Solution: simple nonlinearity

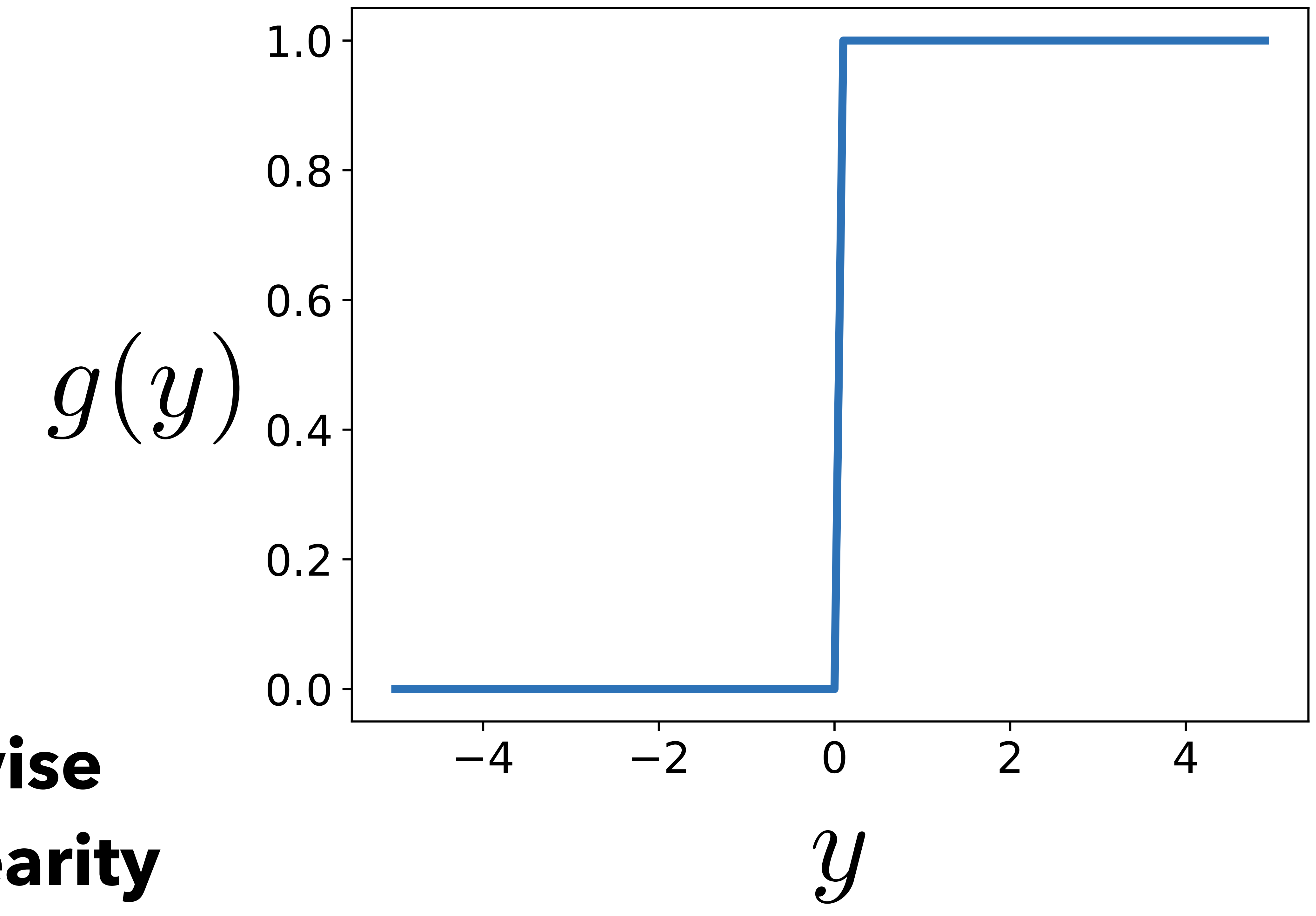
Input representation

Output representation



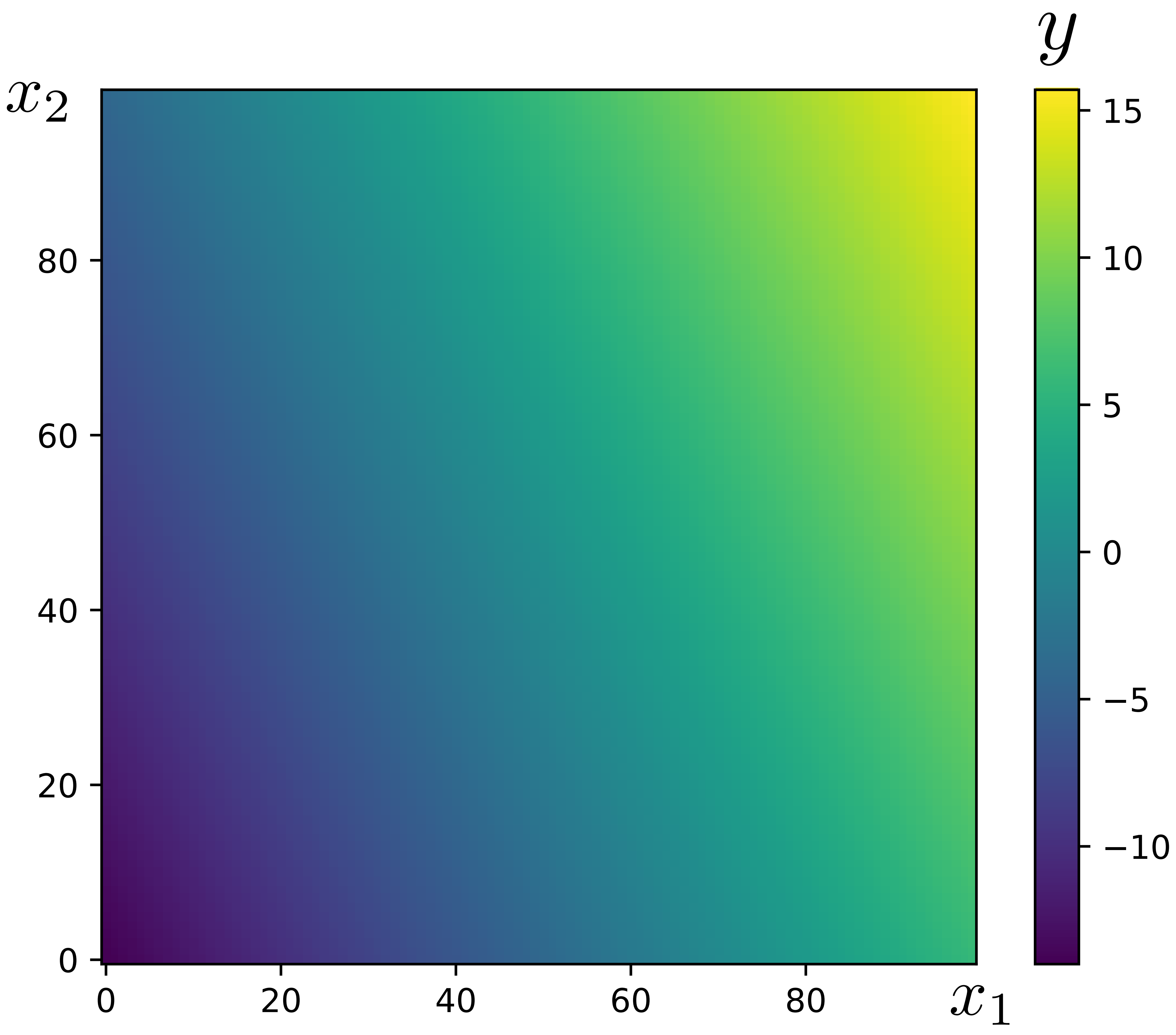
Pointwise Non-linearity

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



Source: Isola, Torralba, Freeman

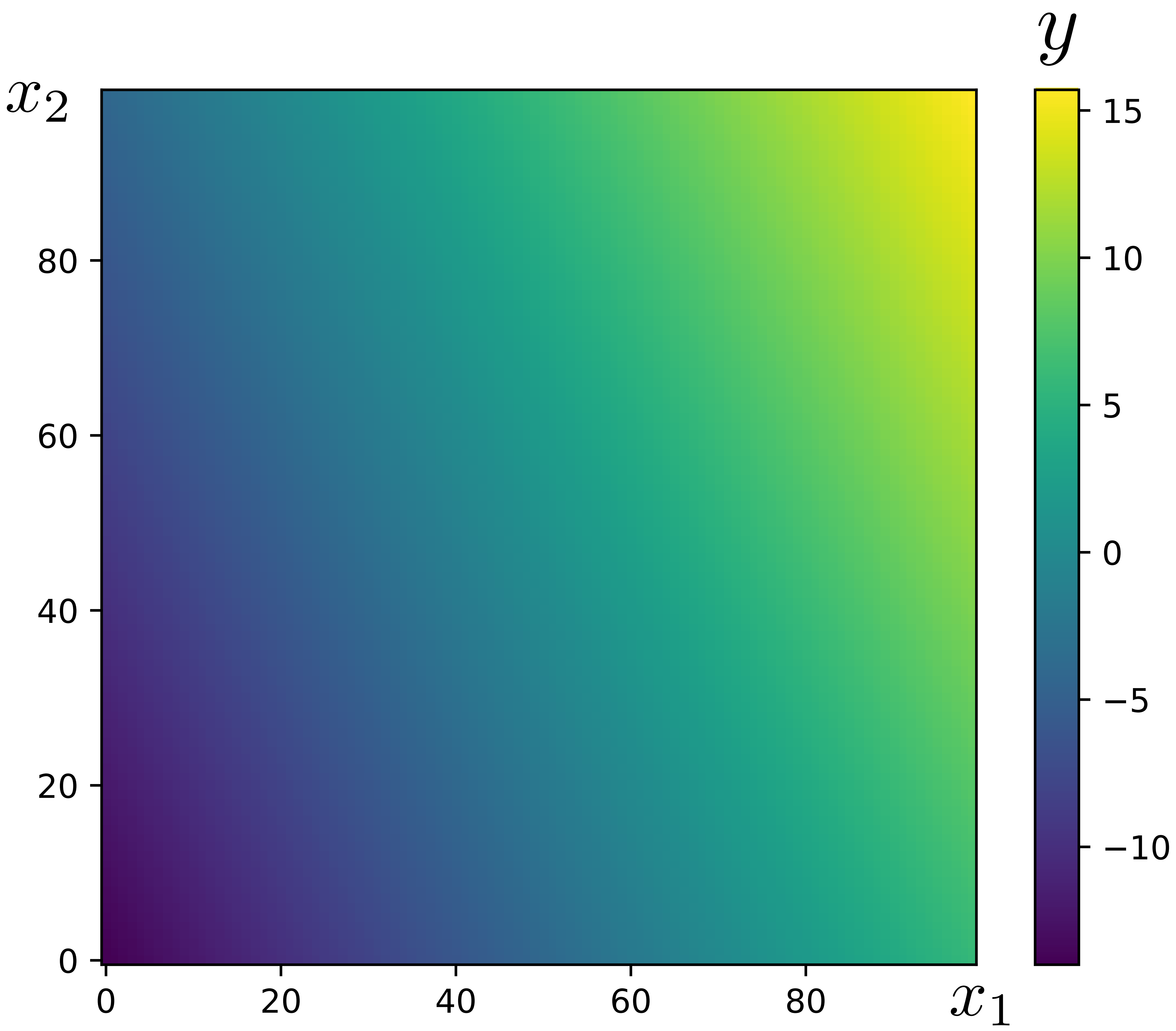
Example: linear classification with a perceptron



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

Source: Isola, Torralba, Freeman

Example: linear classification with a perceptron

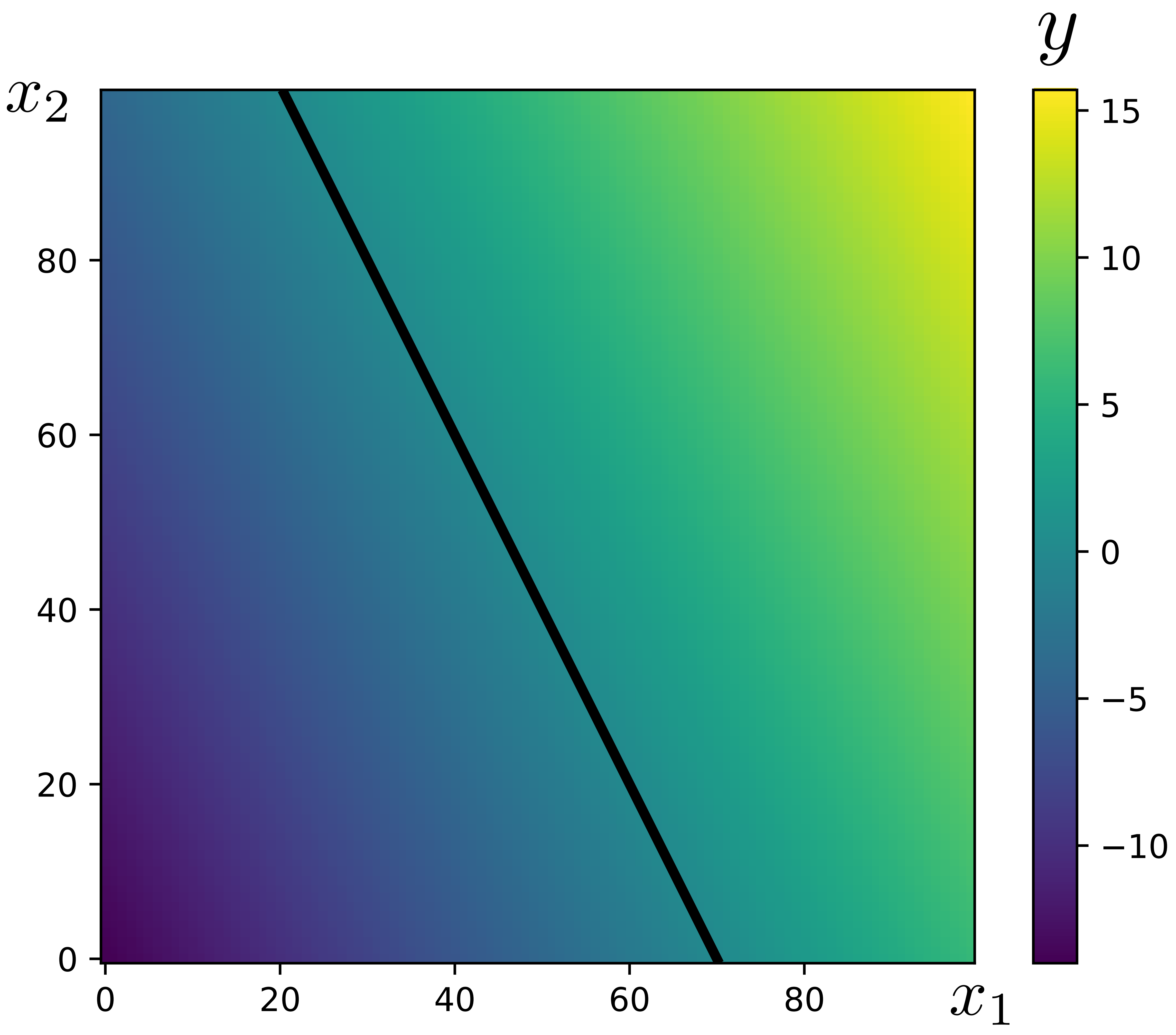


$$y = \mathbf{x}^T \mathbf{w} + b$$

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

Source: Isola, Torralba, Freeman

Example: linear classification with a perceptron

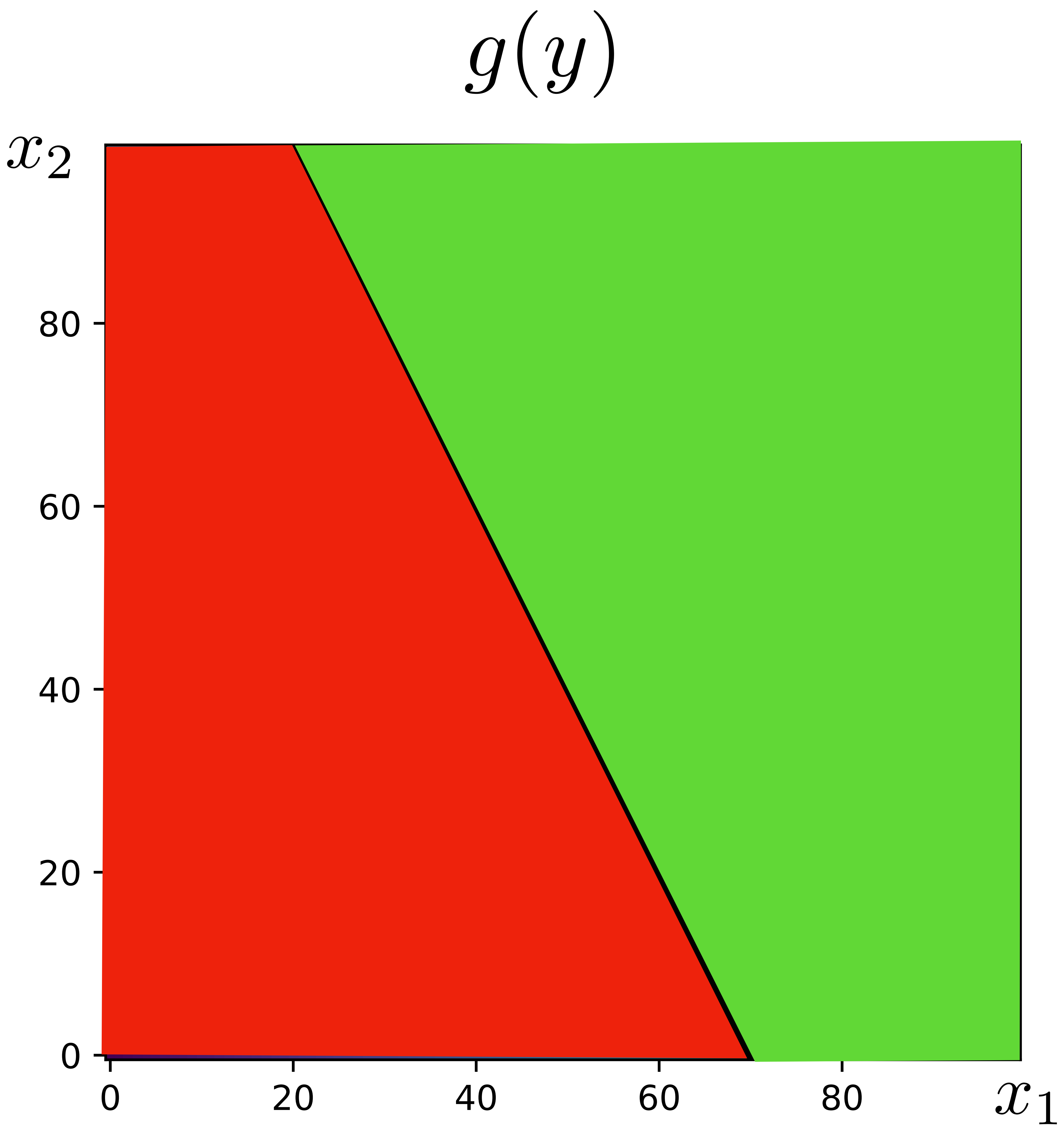


$$y = \mathbf{x}^T \mathbf{w} + b$$

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

Source: Isola, Torralba, Freeman

Example: linear classification with a perceptron



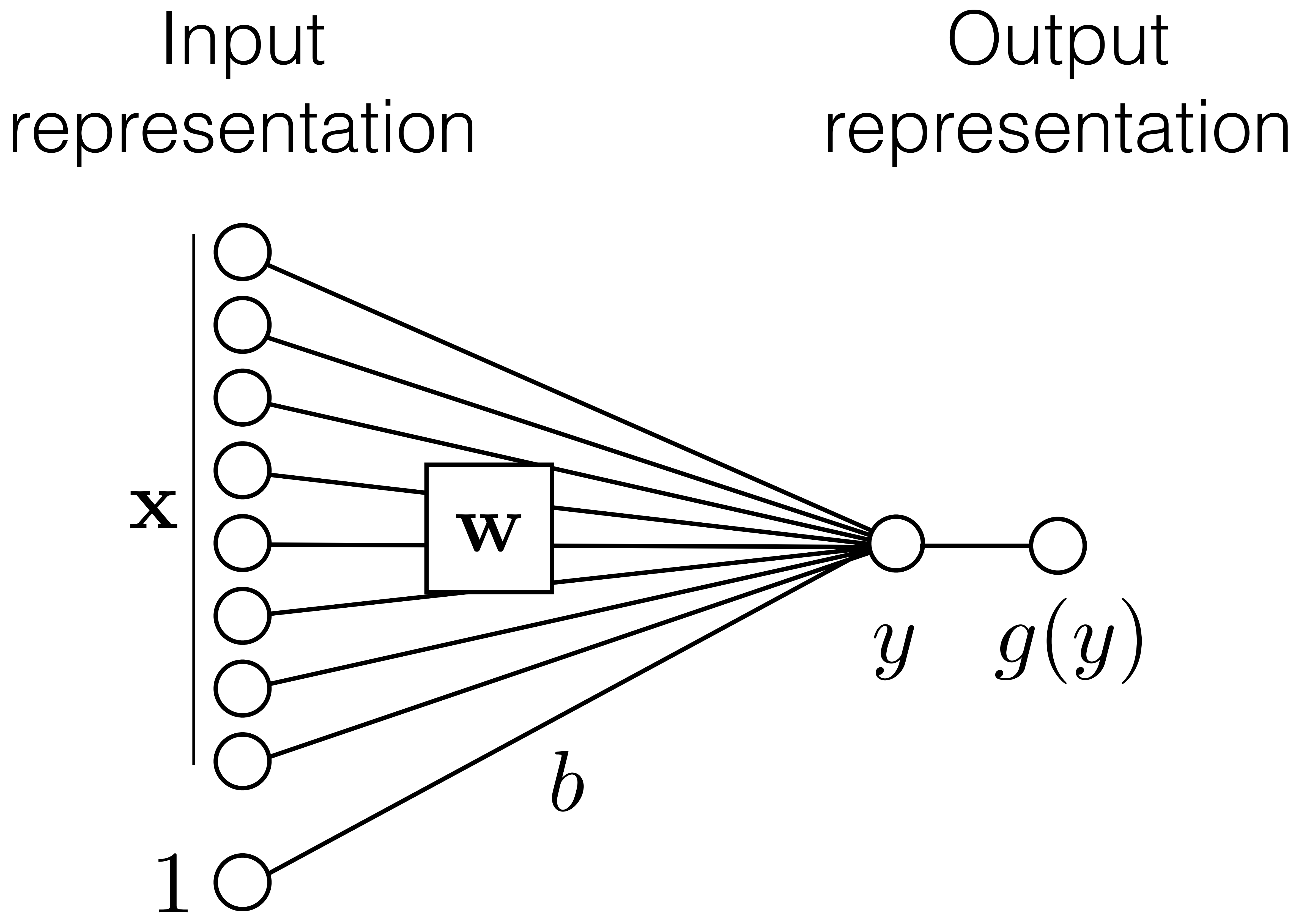
$$y = \mathbf{x}^T \mathbf{w} + b$$

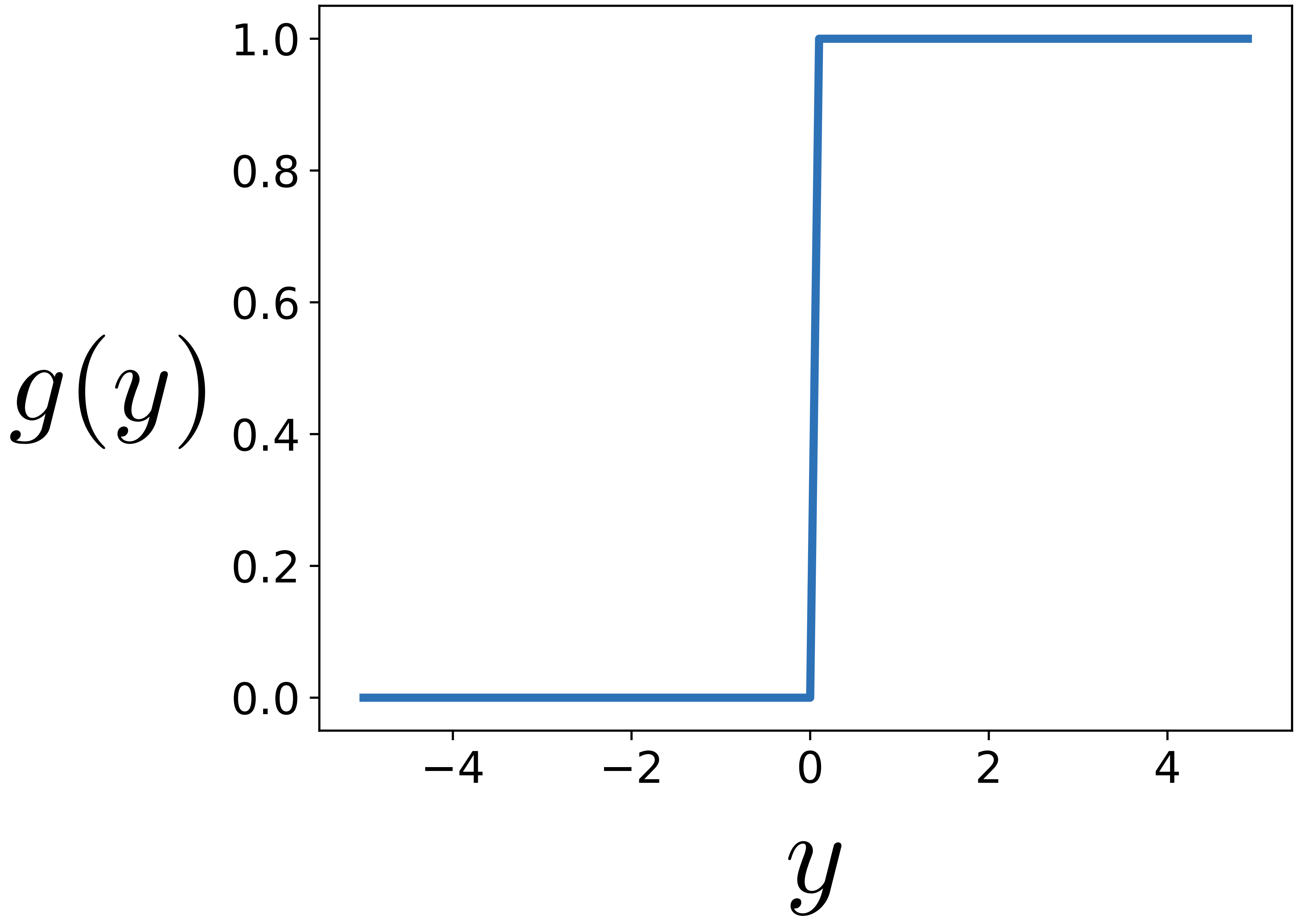
$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$

Source: Isola, Torralba, Freeman

Computation in a neural net — nonlinearity

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





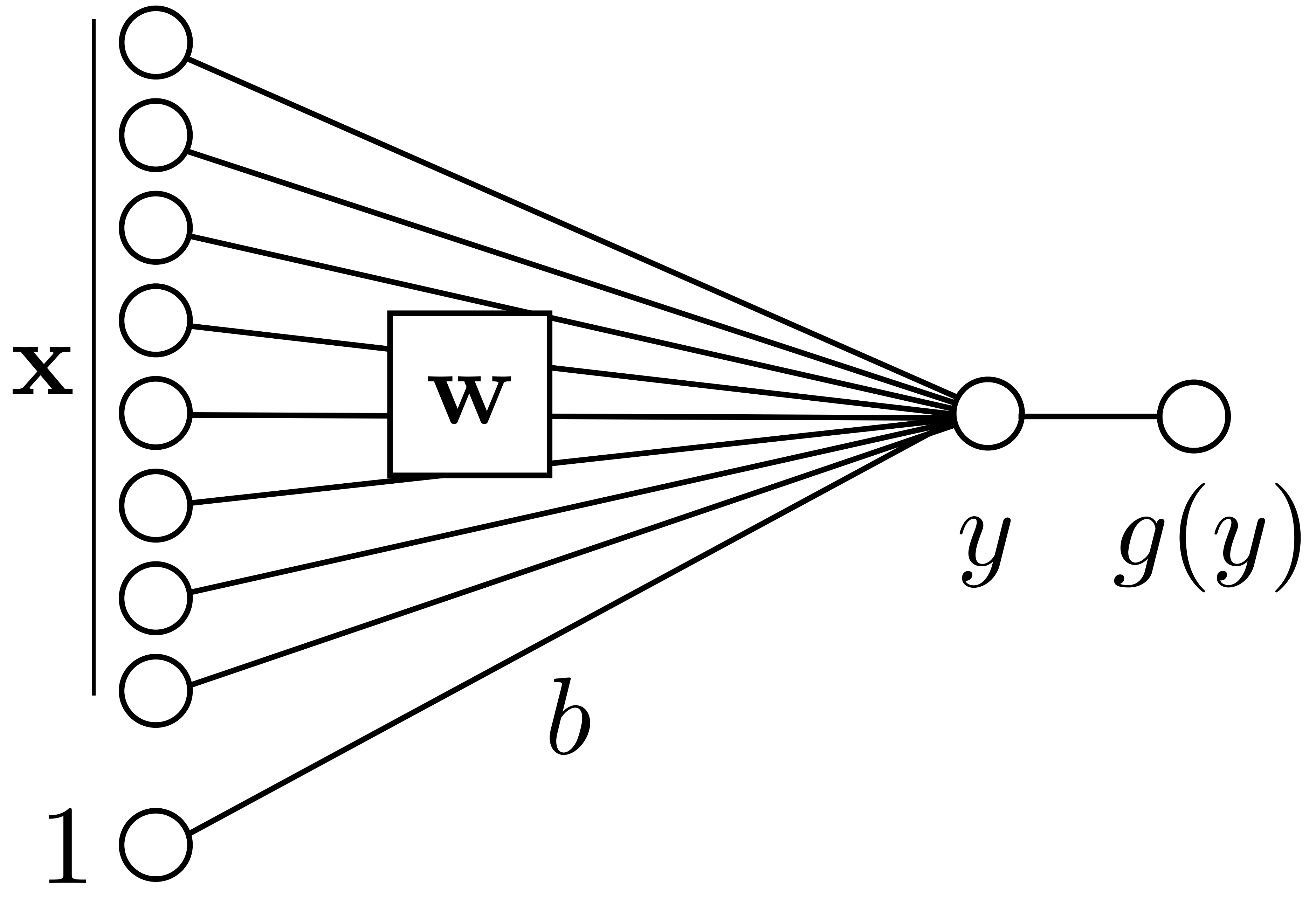
Can't use with gradient descent, $\nabla g = 0$

Source: Isola, Torralba, Freeman

Computation in a neural net — nonlinearity

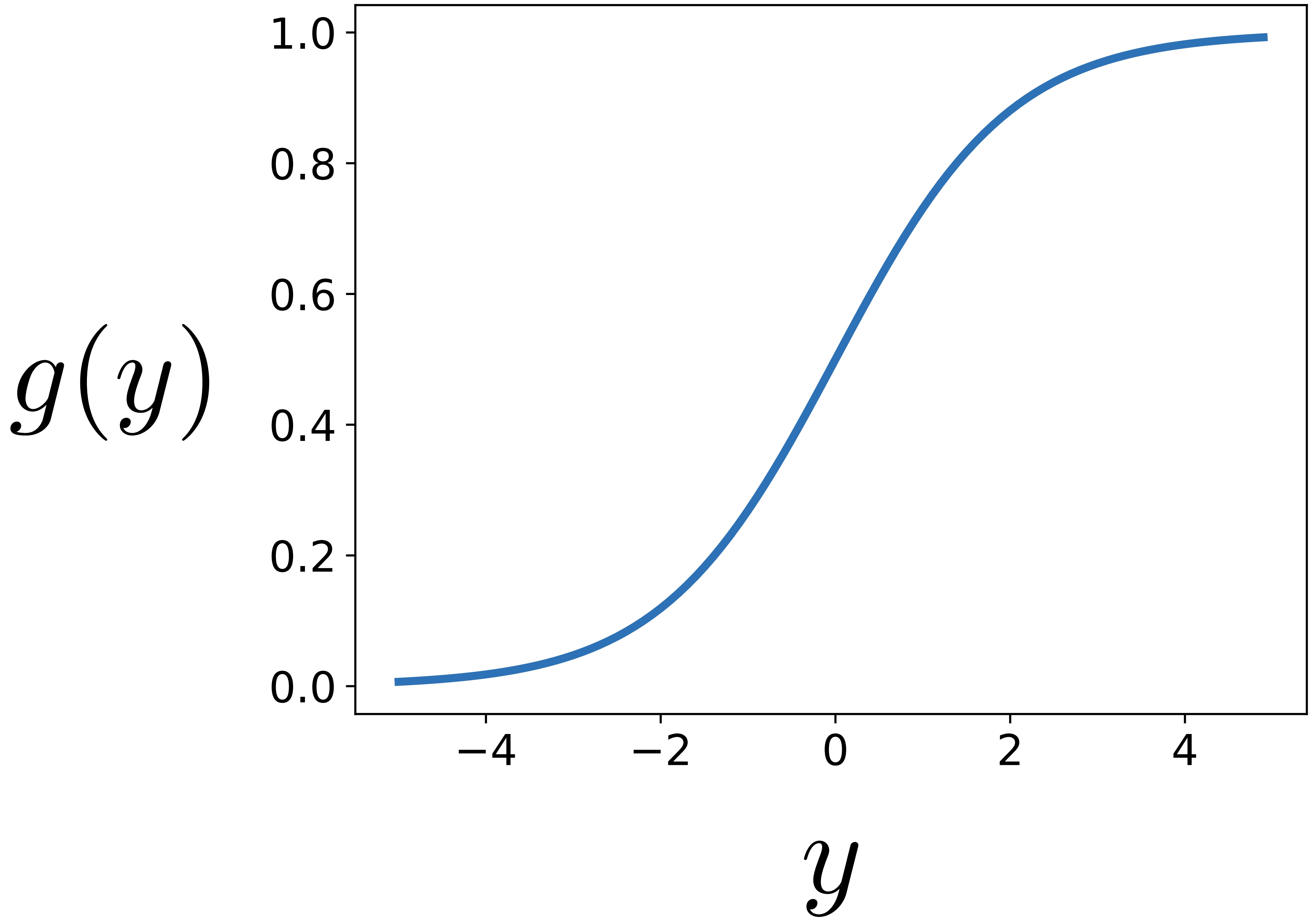
Input representation

Output representation



Sigmoid

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



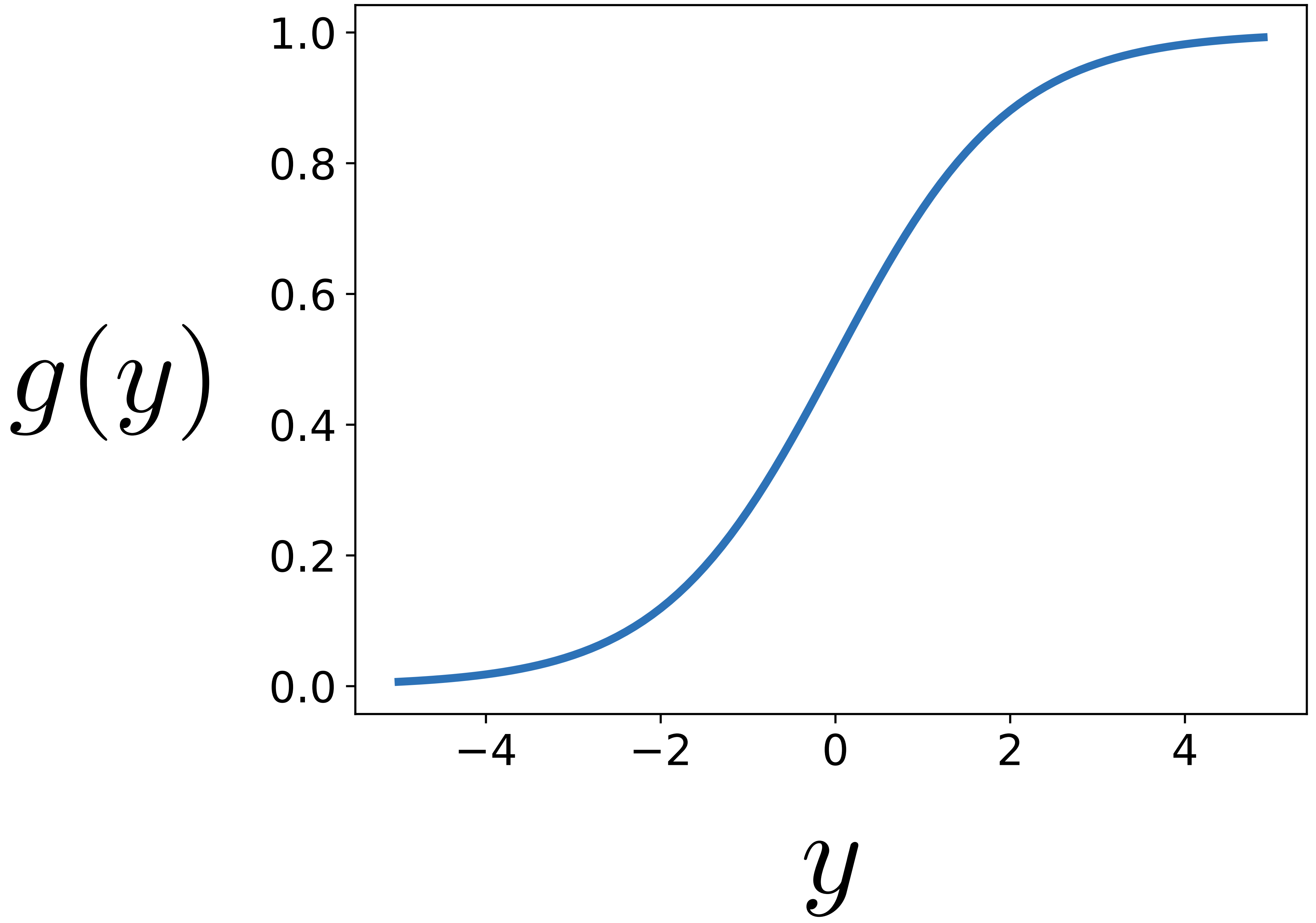
Source: Isola, Torralba, Freeman

Computation in a neural net — nonlinearity

- Interpretation as firing rate of neuron
- Bounded between [0,1]
- Saturation for large +/- inputs
- Gradients go to zero
- Centered at 0.5. Better in practice to use: $\tanh(y) = 2g(y) - 1$

Sigmoid

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



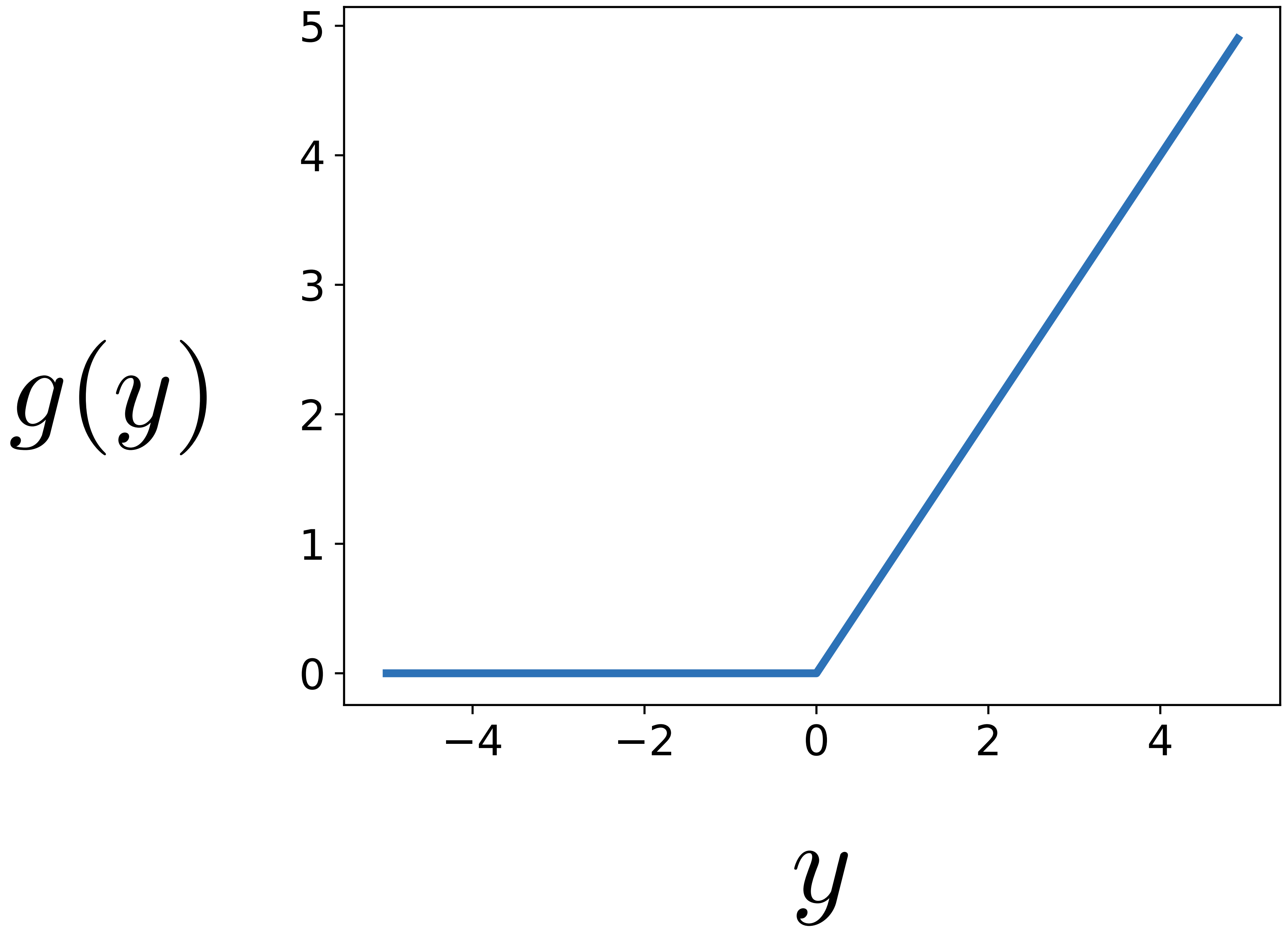
Source: Isola, Torralba, Freeman

Computation in a neural net — nonlinearity

- Unbounded output (on positive side)
- Efficient to implement: $\frac{\partial g}{\partial y} = \begin{cases} 0, & \text{if} \quad y < 0 \\ 1, & \text{if} \quad y \geq 0 \end{cases}$
- 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)

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



Source: Isola, Torralba, Freeman

Computation in a neural net — nonlinearity

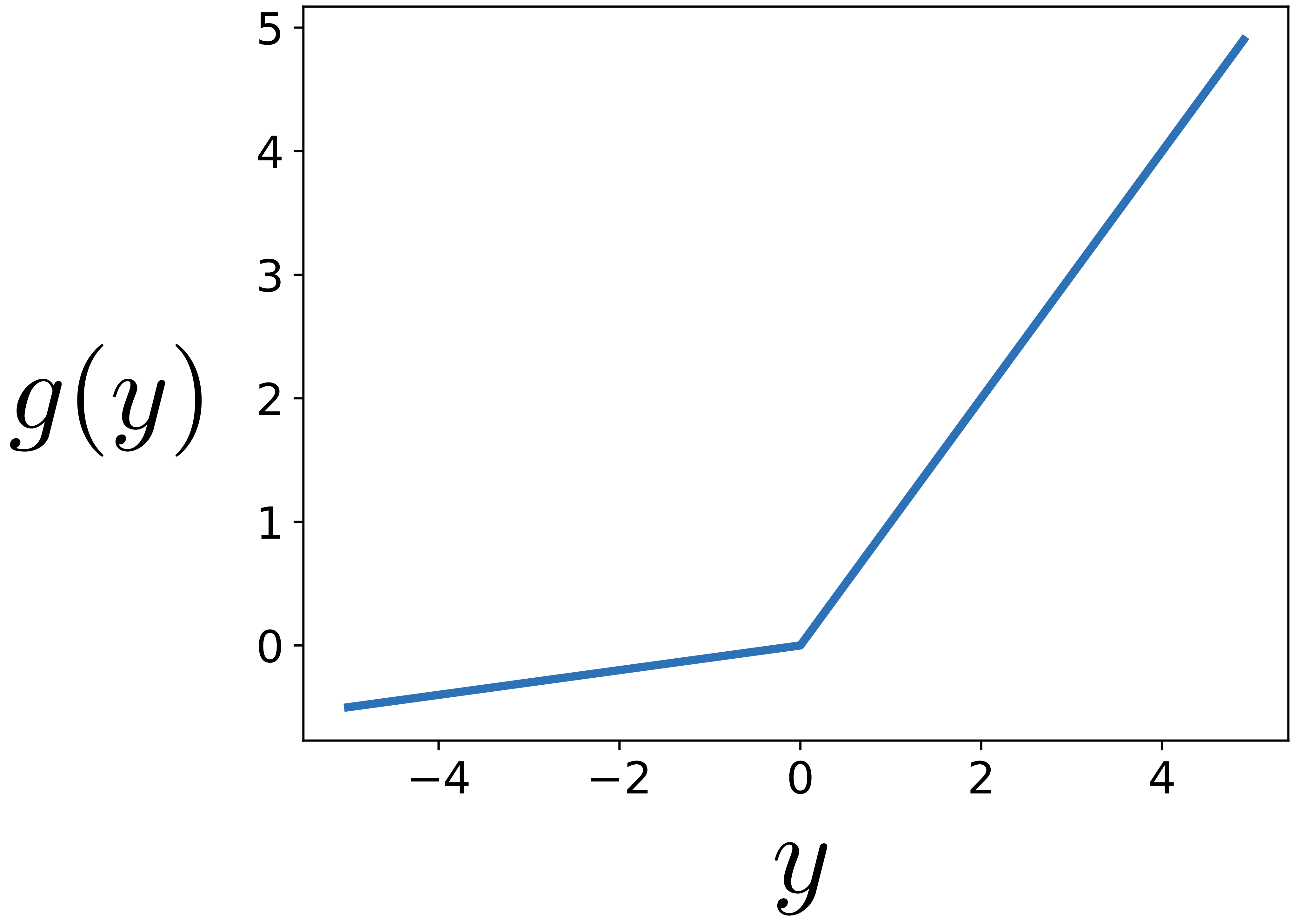
- where a is small (e.g. 0.02)

- Efficient to implement: $\frac{\partial g}{\partial y} = \begin{cases} -a, & \text{if} \quad y < 0 \\ 1, & \text{if} \quad y \geq 0 \end{cases}$

- Has non-zero gradients everywhere (unlike ReLU)

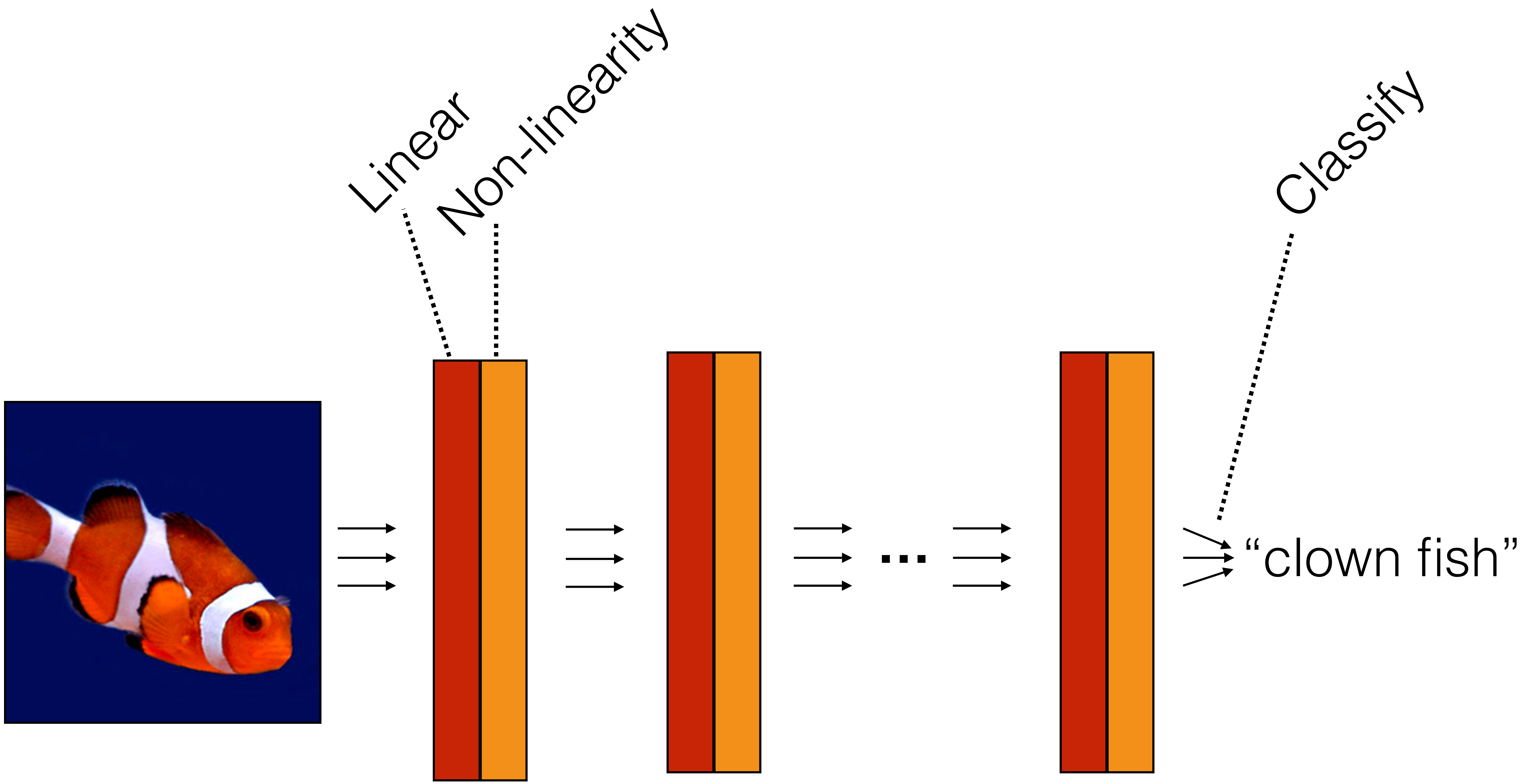
Leaky ReLU

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



Source: Isola, Torralba, Freeman

Deep nets



$$f(\mathbf{x}) = f_L(\dots f_2(f_1(\mathbf{x})))$$

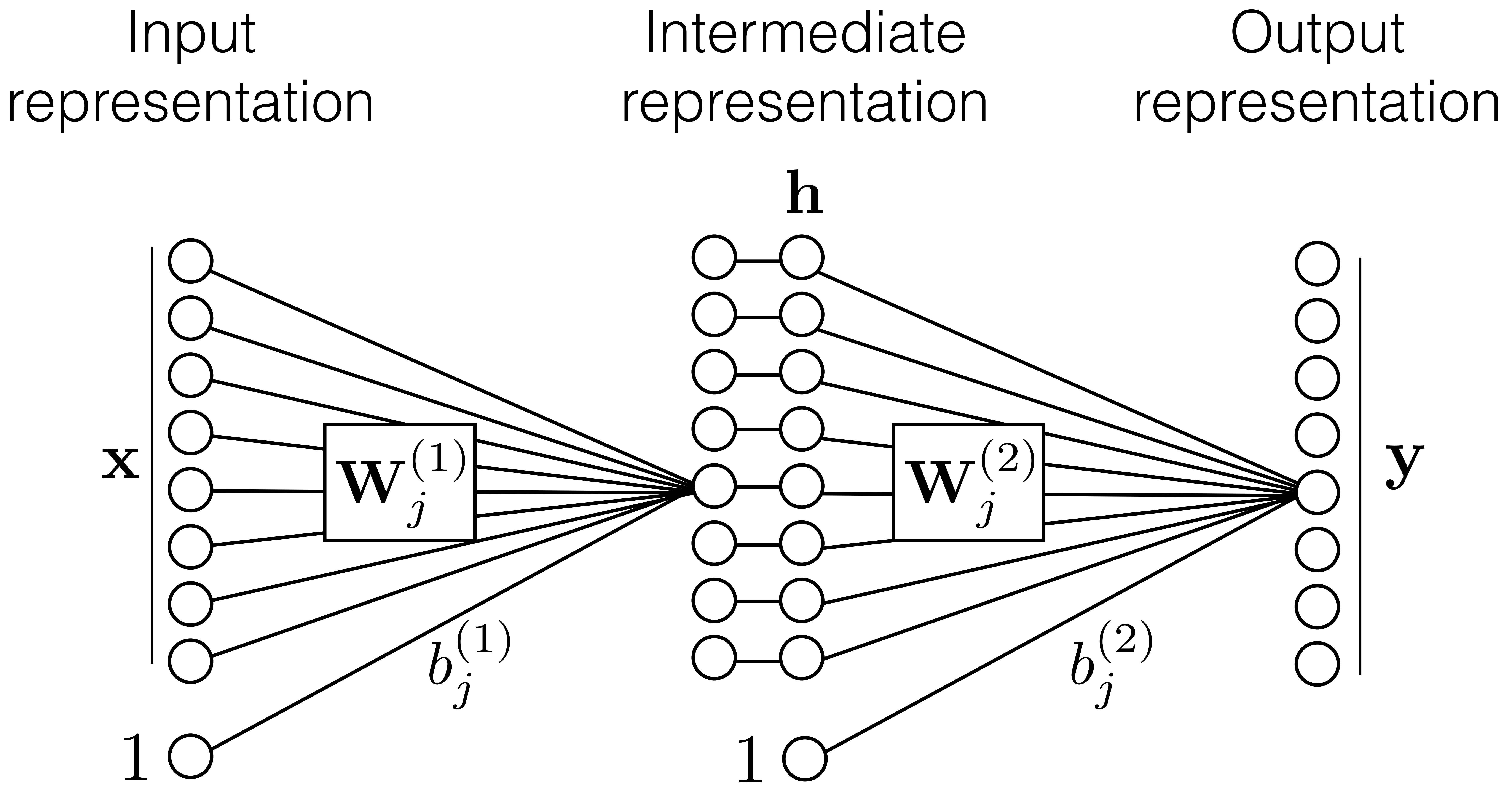
Source: Isola, Torralba, Freeman

Computation has a simple form

$$\mathbf{y} = \mathbf{W}^{(n)} g(\mathbf{W}^{(n-1)} \ldots g(\mathbf{W}^{(3)} g(\mathbf{W}^{(2)} (g(\mathbf{W}^{(1)} \mathbf{x})))))$$

- Composition of linear functions with nonlinearities in between
- E.g. matrix multiplications with ReLU, $\mathbf{max(0,\ x)}$ afterwards
- Do a matrix multiplication, set all negative values to 0, repeat

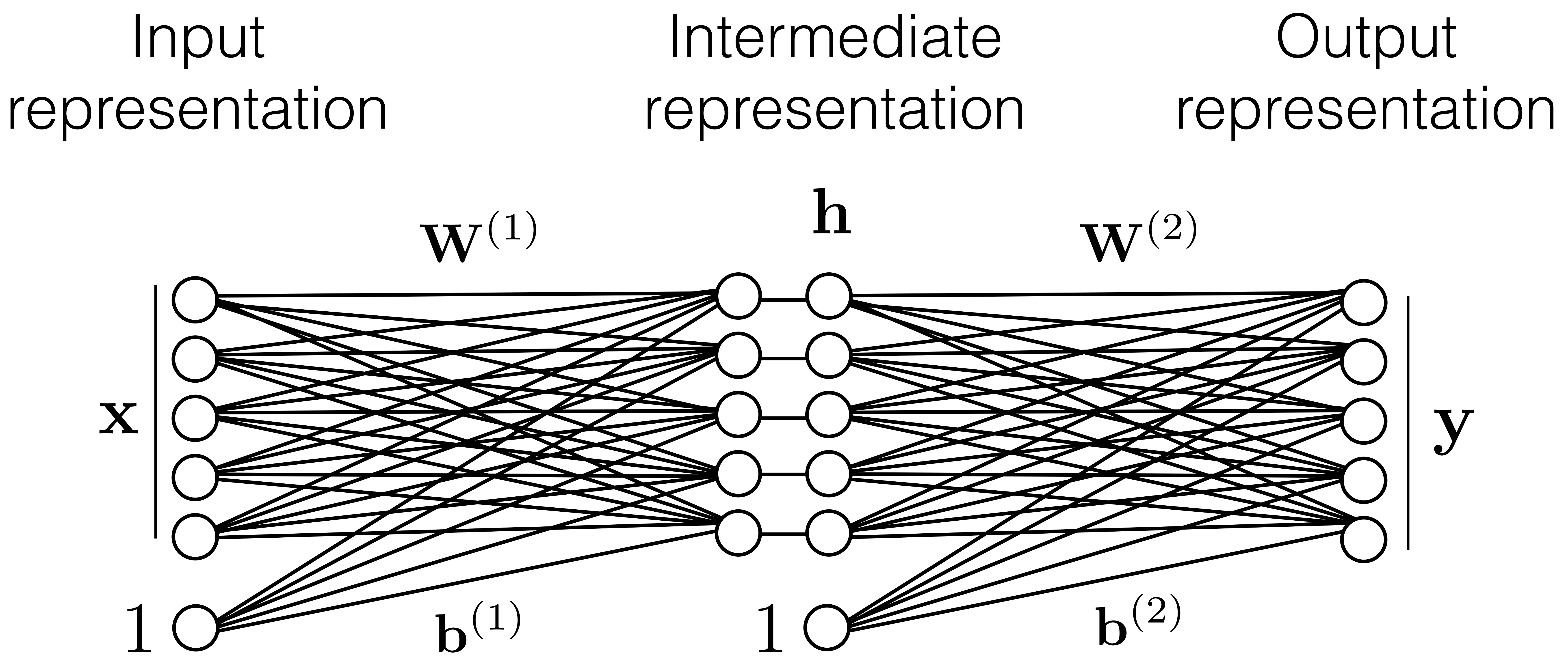
Stacking layers



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

Source: Isola, Torralba, Freeman

Stacking layers

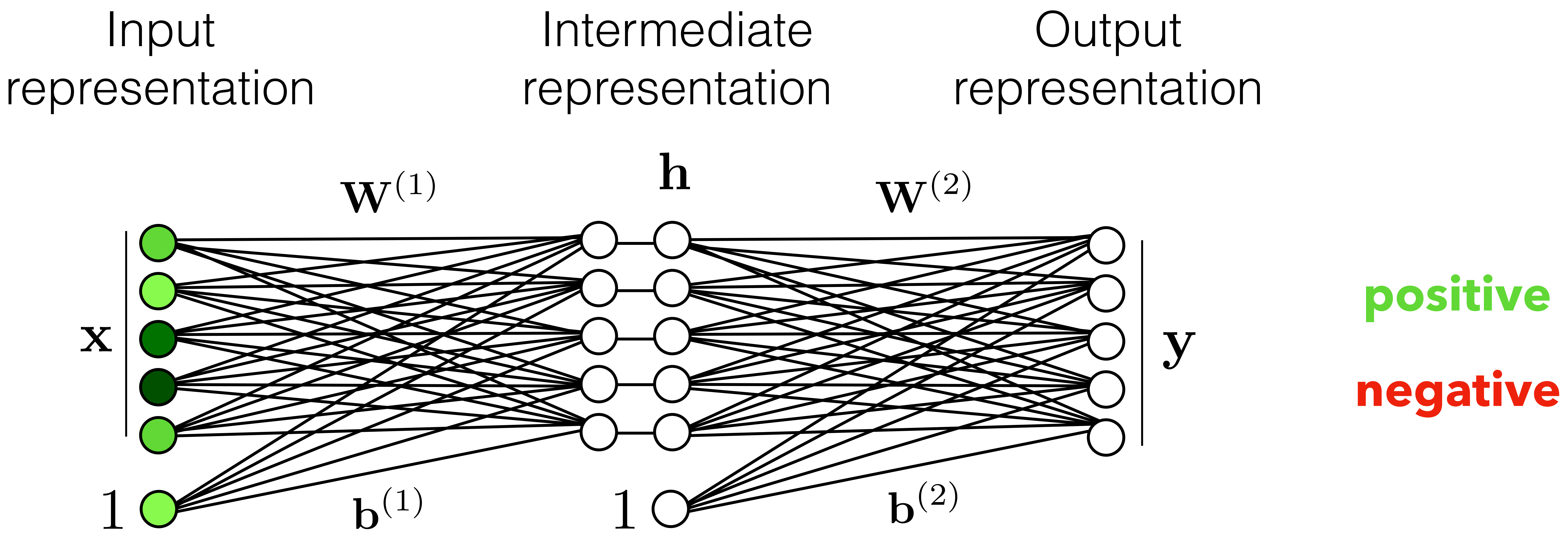


$$\mathbf{h} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \qquad \mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

$$\theta = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}$$

Source: Isola, Torralba, Freeman

Stacking layers



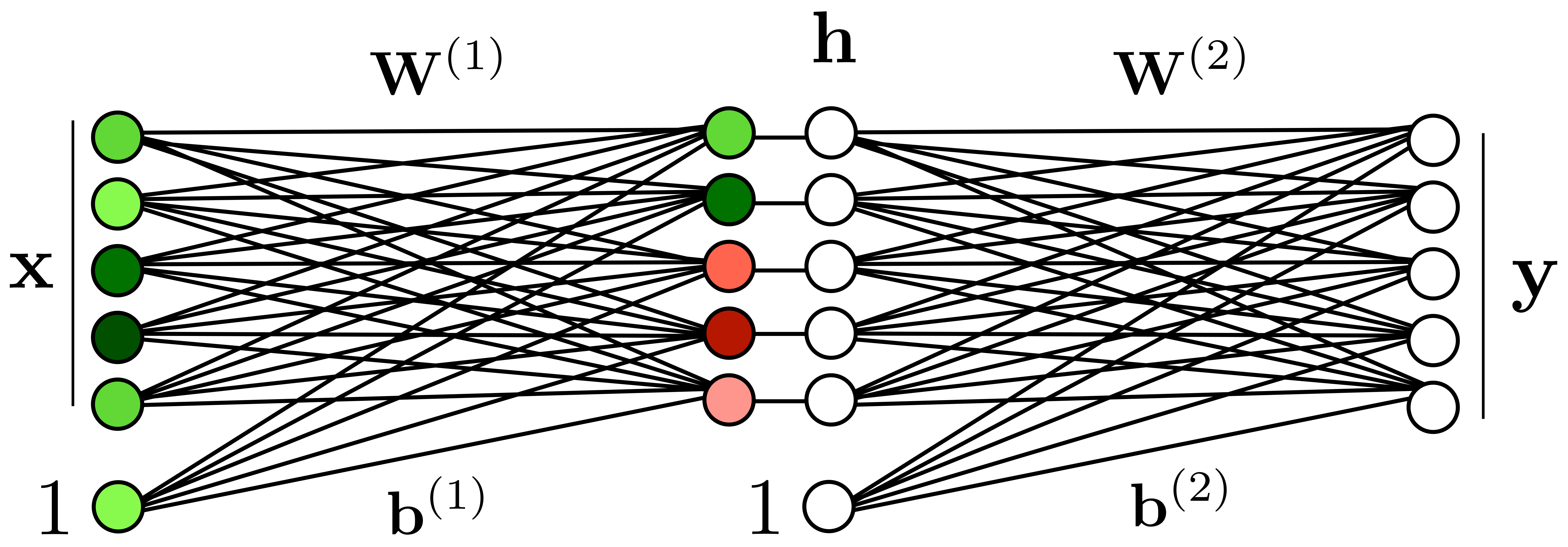
$$\mathbf{h} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \qquad \mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

$$\theta = \{\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \dots, \mathbf{b}^{(L)}\}$$

Source: Isola, Torralba, Freeman

Stacking layers

Input representation | Intermediate representation | Output representation



positive

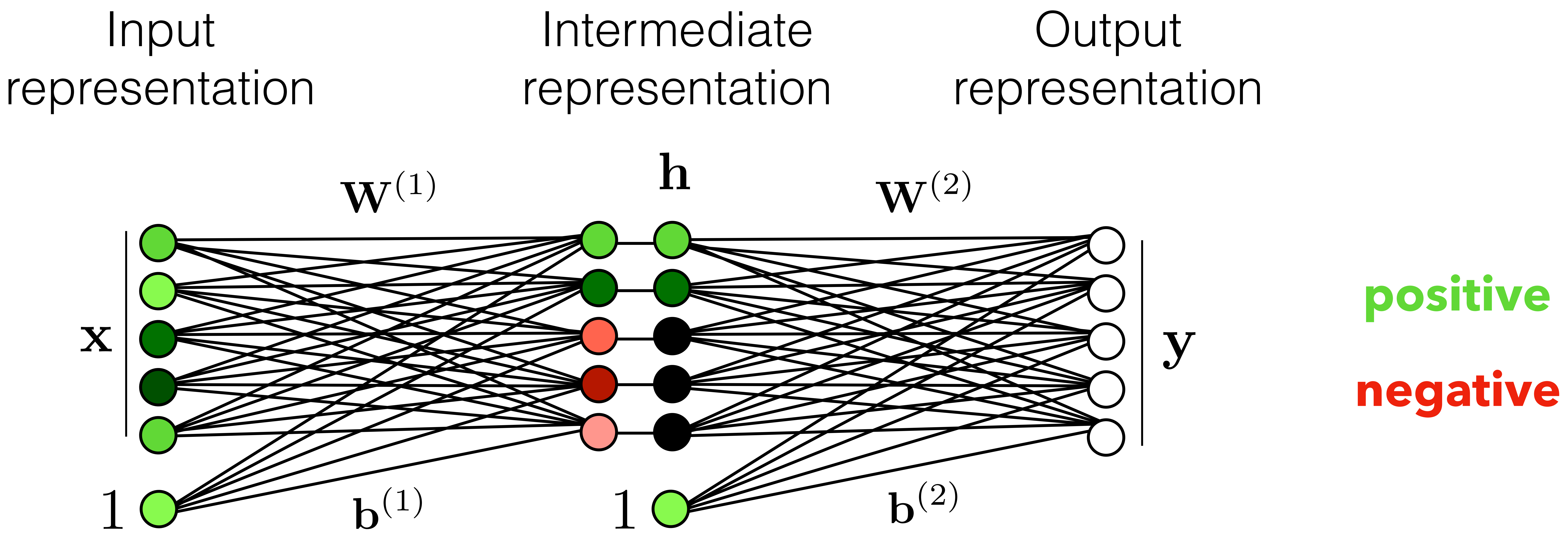
negative

$$\mathbf{h} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \qquad \mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

$$\theta = \{\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \dots, \mathbf{b}^{(L)}\}$$

Source: Isola, Torralba, Freeman

Stacking layers

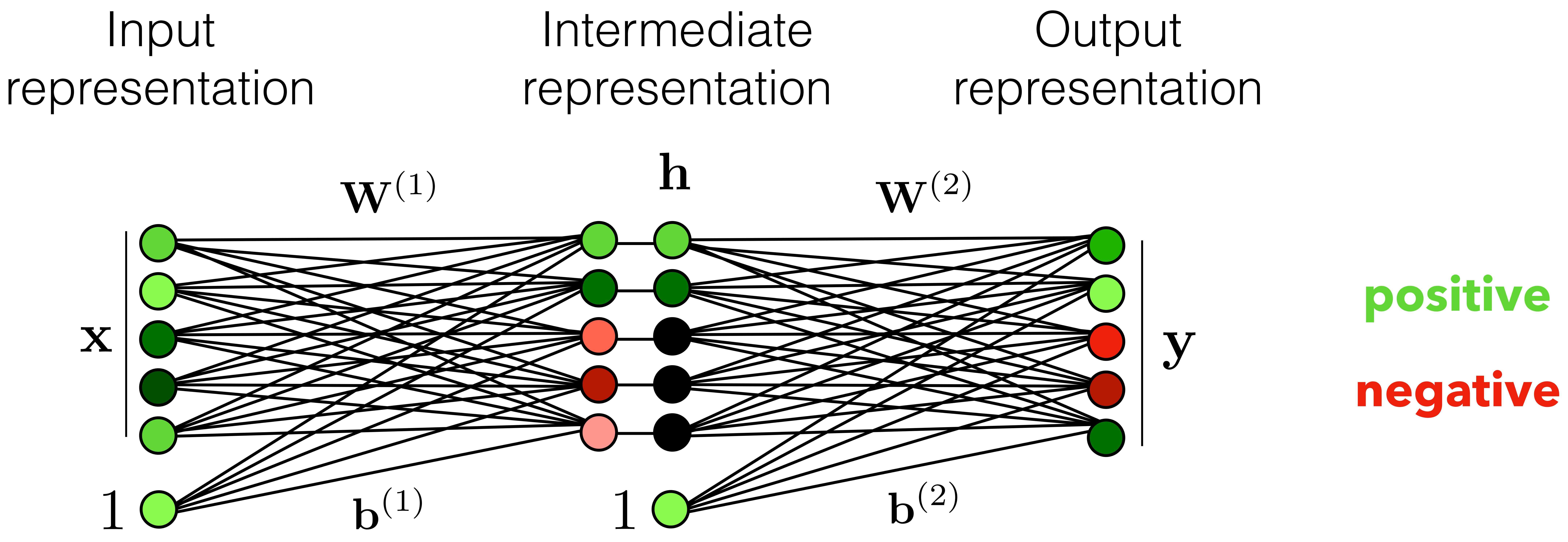


$$\mathbf{h} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \qquad \mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

$$\theta = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}$$

Source: Isola, Torralba, Freeman

Stacking layers

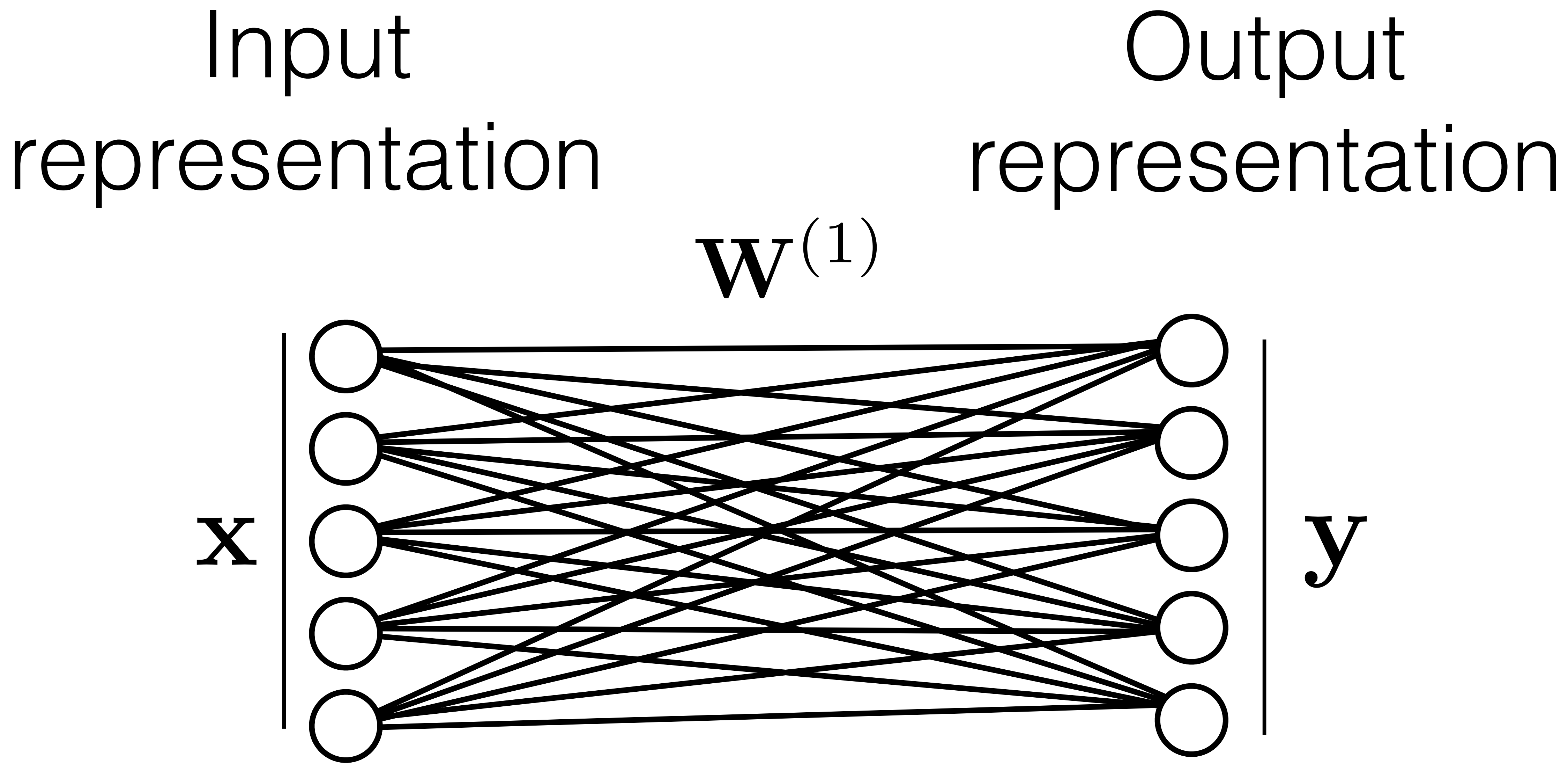


$$\mathbf{h} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \qquad \mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

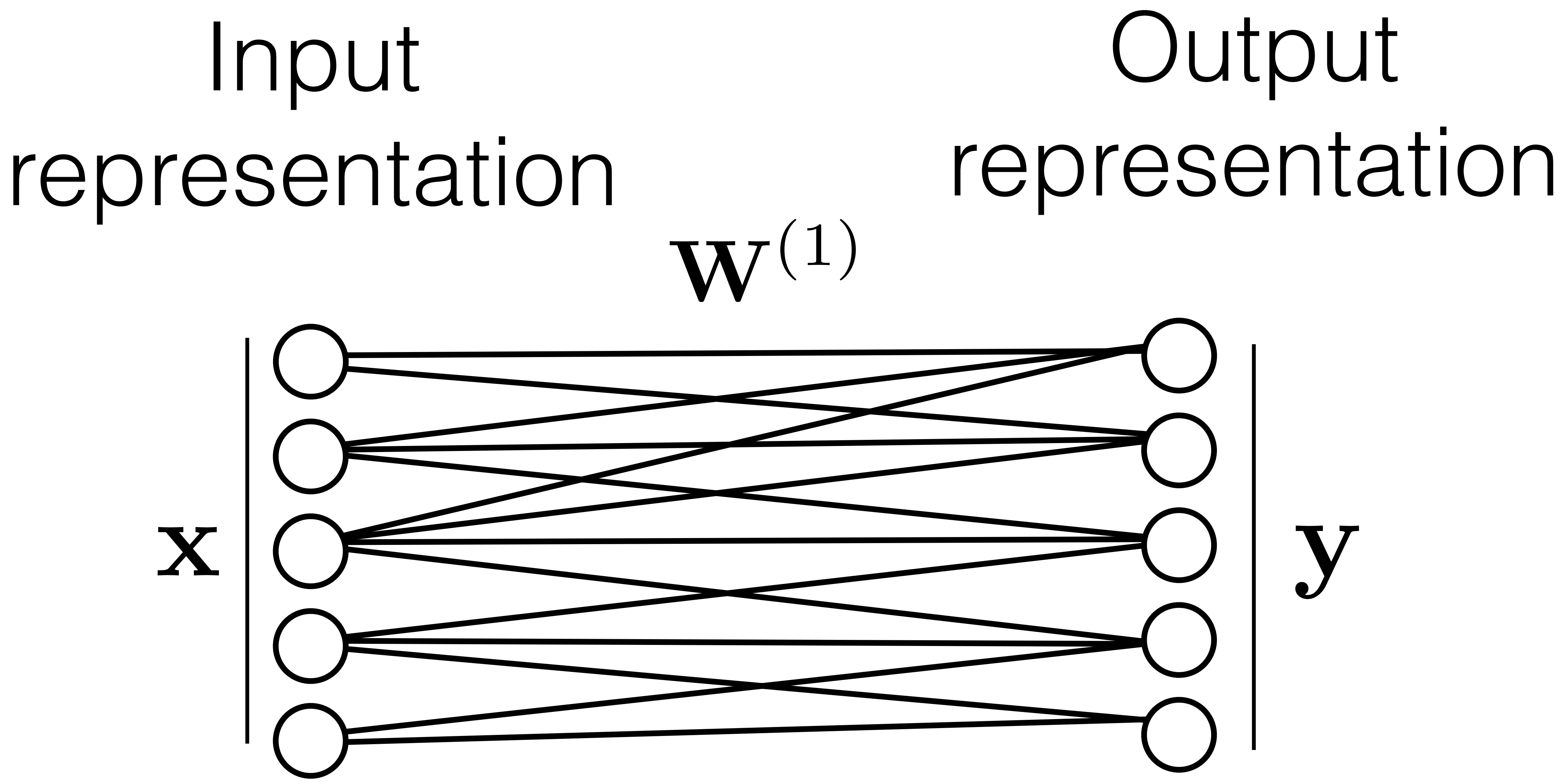
$$\theta = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}$$

Source: Isola, Torralba, Freeman

Connectivity patterns



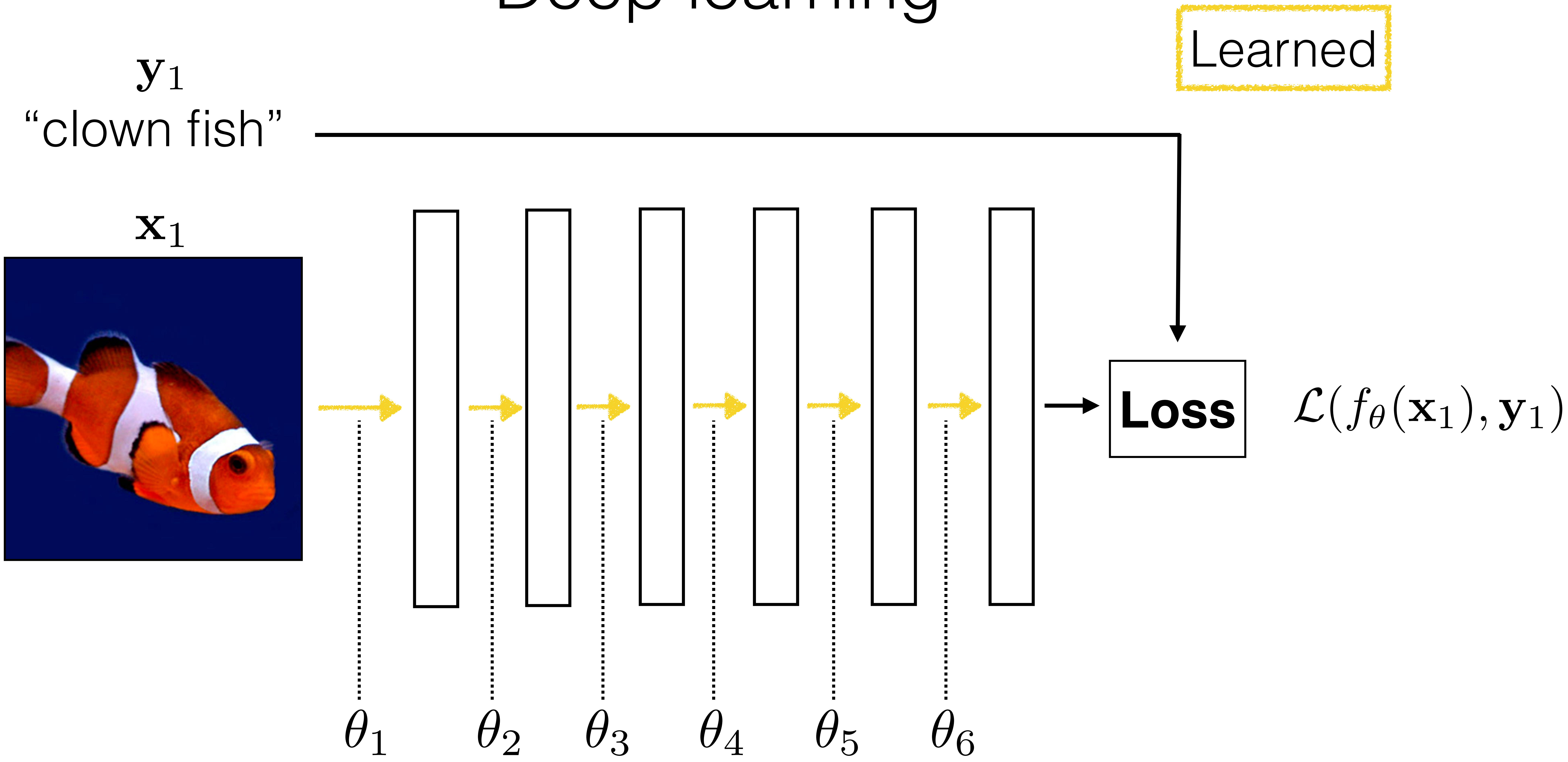
Fully connected layer



Locally connected layer
(Sparse W)

Source: Isola, Torralba, Freeman

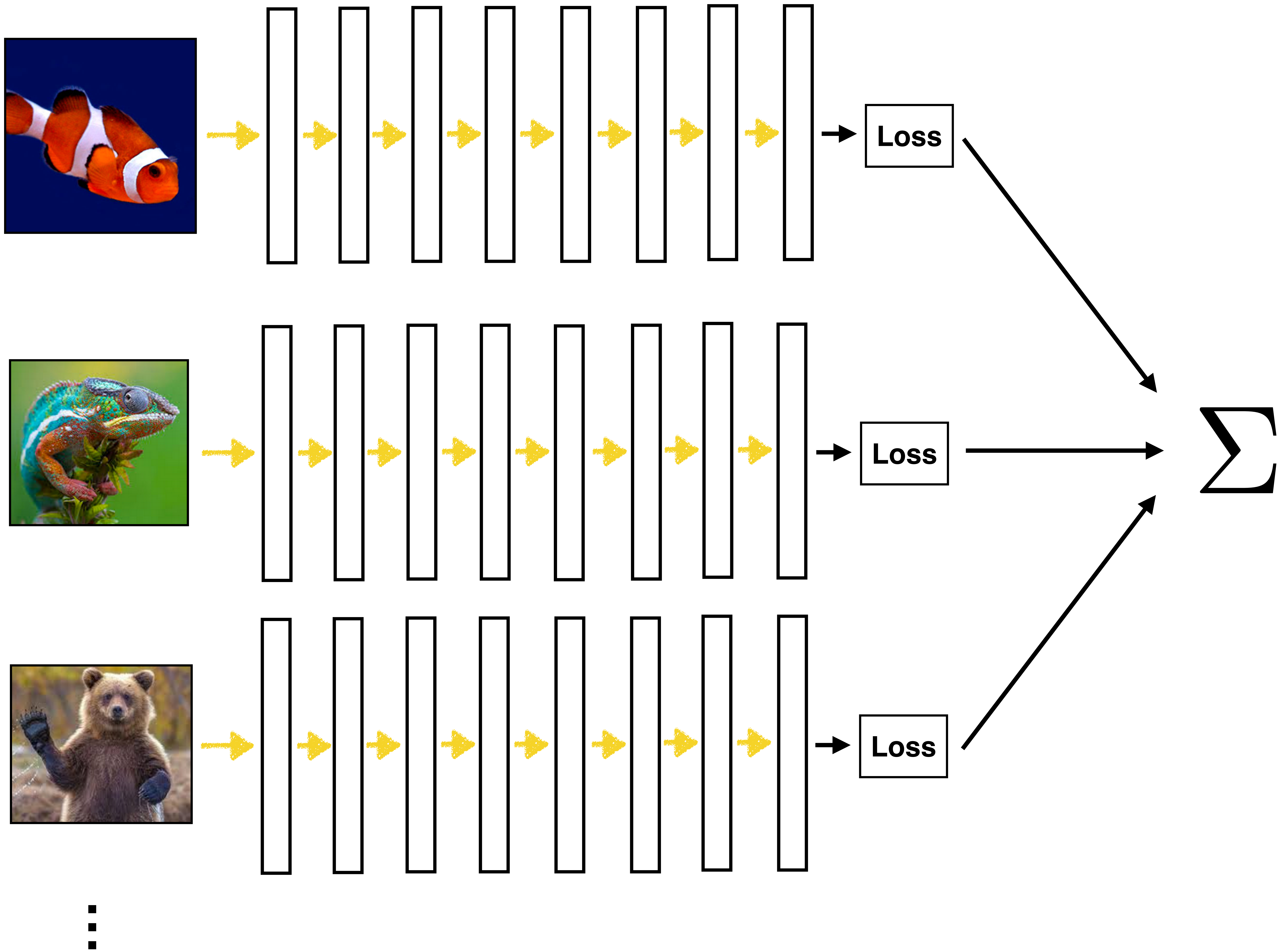
Deep learning

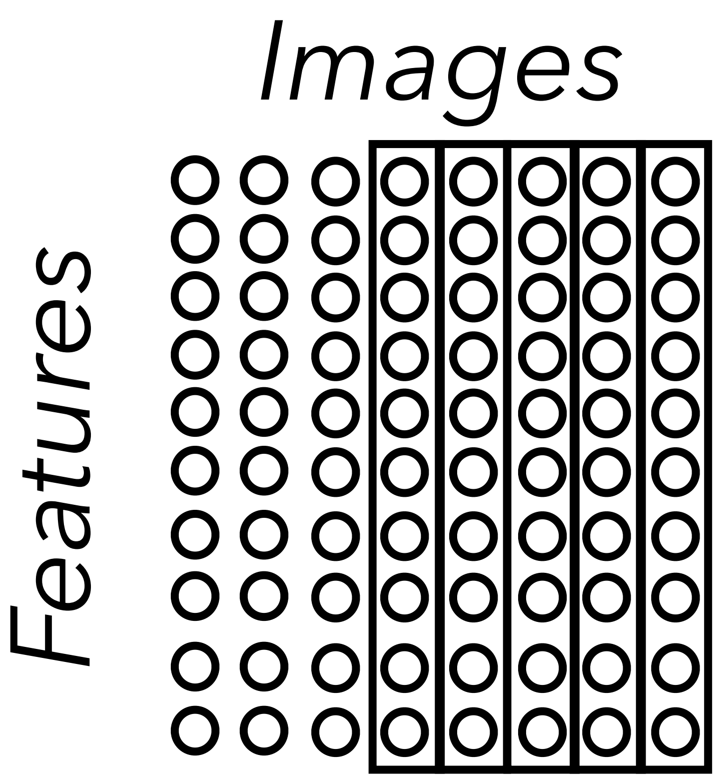


$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)$$

Source: Isola, Torralba, Freeman

Batch processing

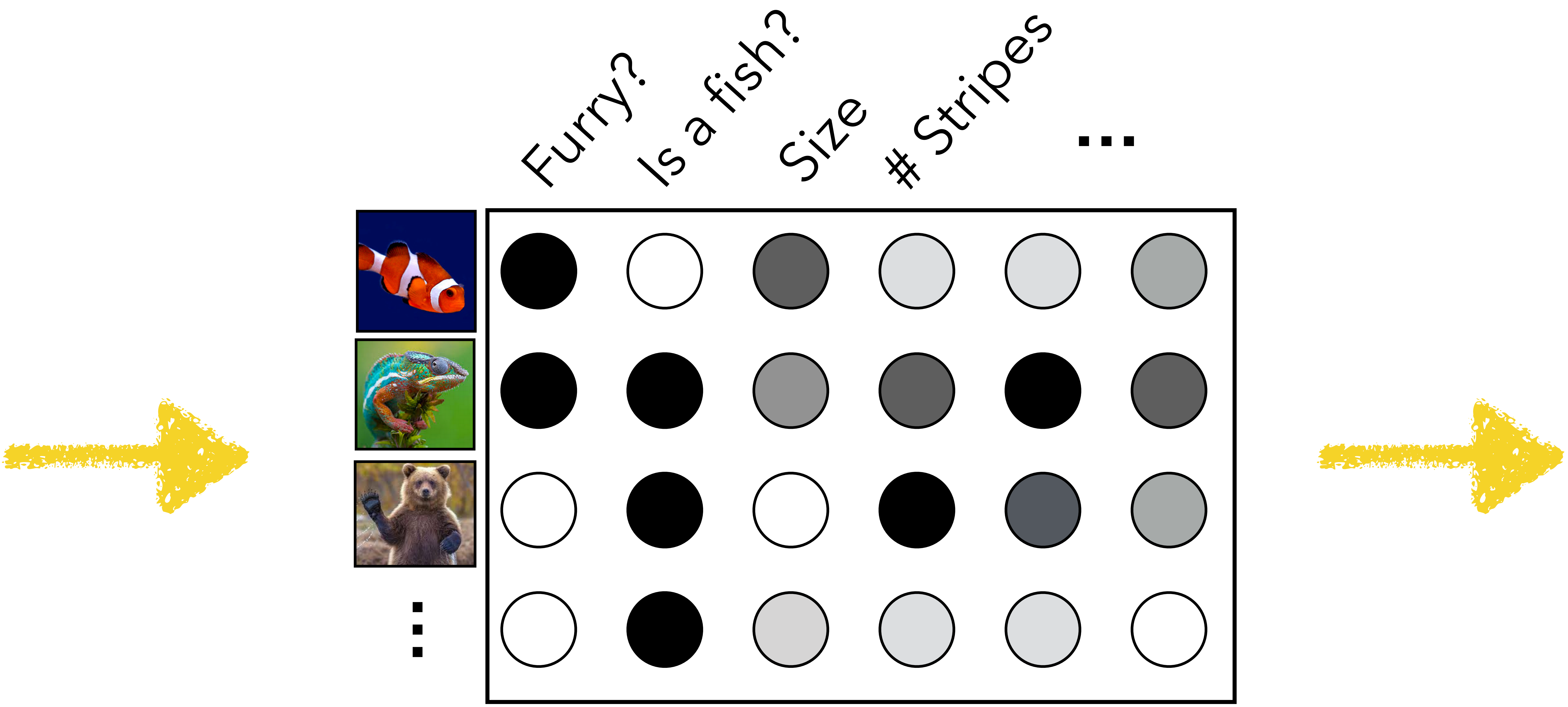




Source: Isola, Torralba, Freema

Tensors

(multi-dimensional arrays)



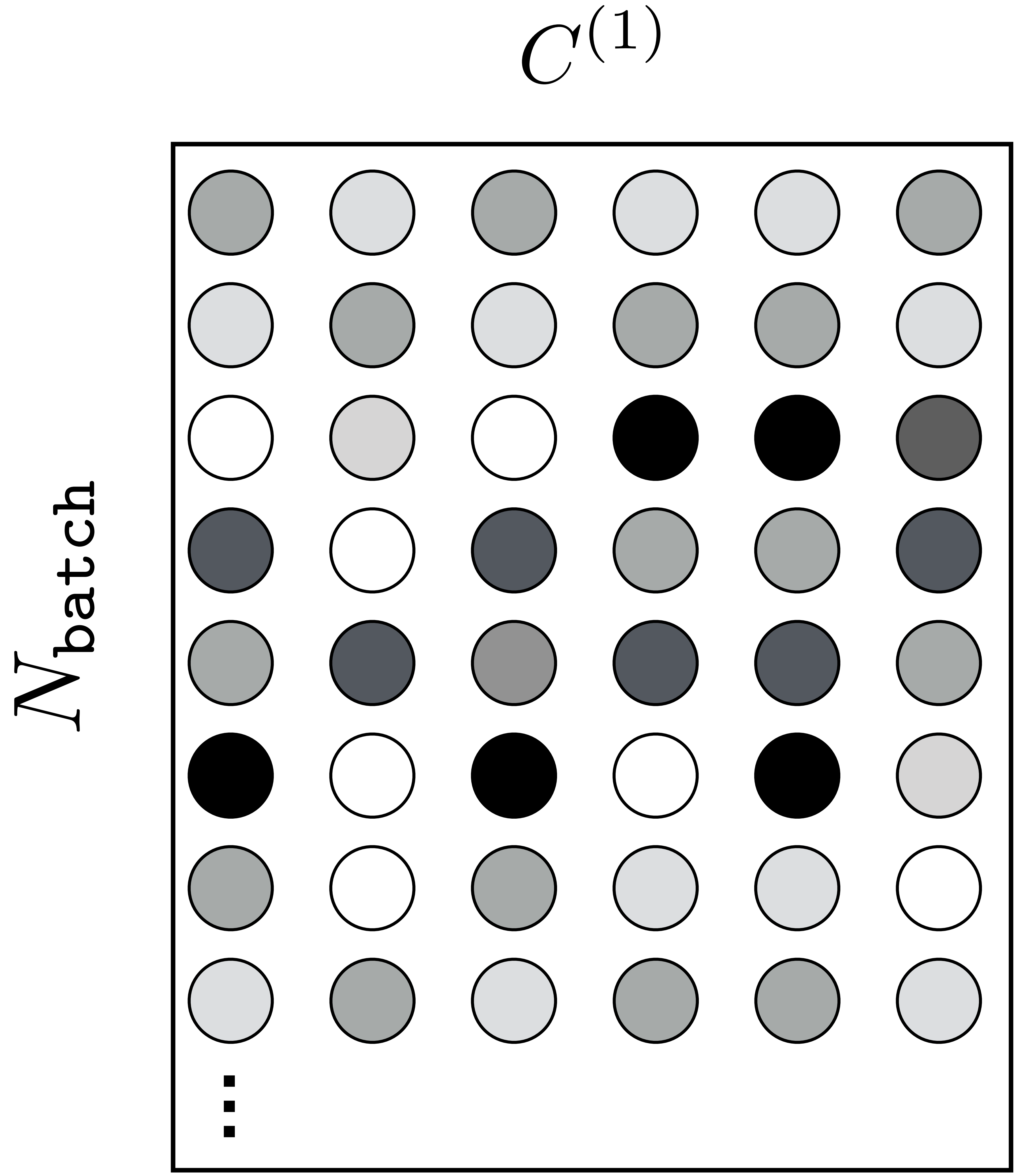
Each layer is a representation of the data

Source: Isola, Torralba, Freeman

Tensors
(multi-dimensional arrays)

$$\mathbf{h}^{(1)} \in \mathbb{R}^{N_{\texttt{batch}} \times C^{(1)}}$$

neurons
features
units
"channels"



Source: Isola, Torralba, Freeman

Tensors
(multi-dimensional arrays)

$$\mathbf{h}^{(1)} \in \mathbb{R}^{N_{\texttt{batch}} \times C^{(1)}}$$

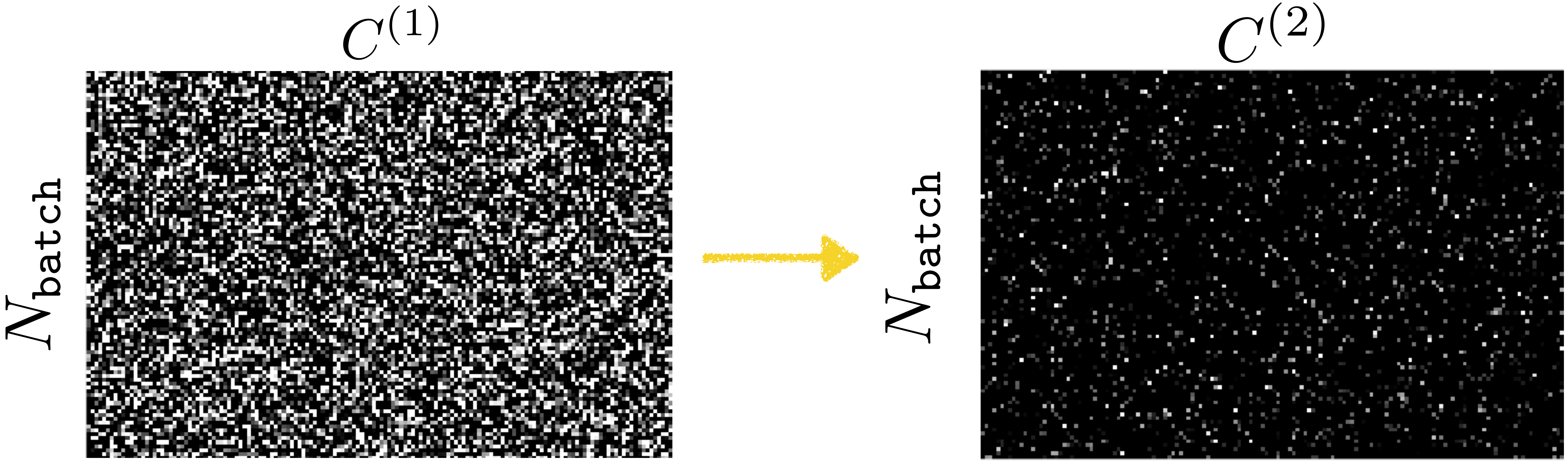
\# neurons
\# features
\# units
\# "channels"

Source: Isola, Torralba, Freeman

Tensors

(multi-dimensional arrays)

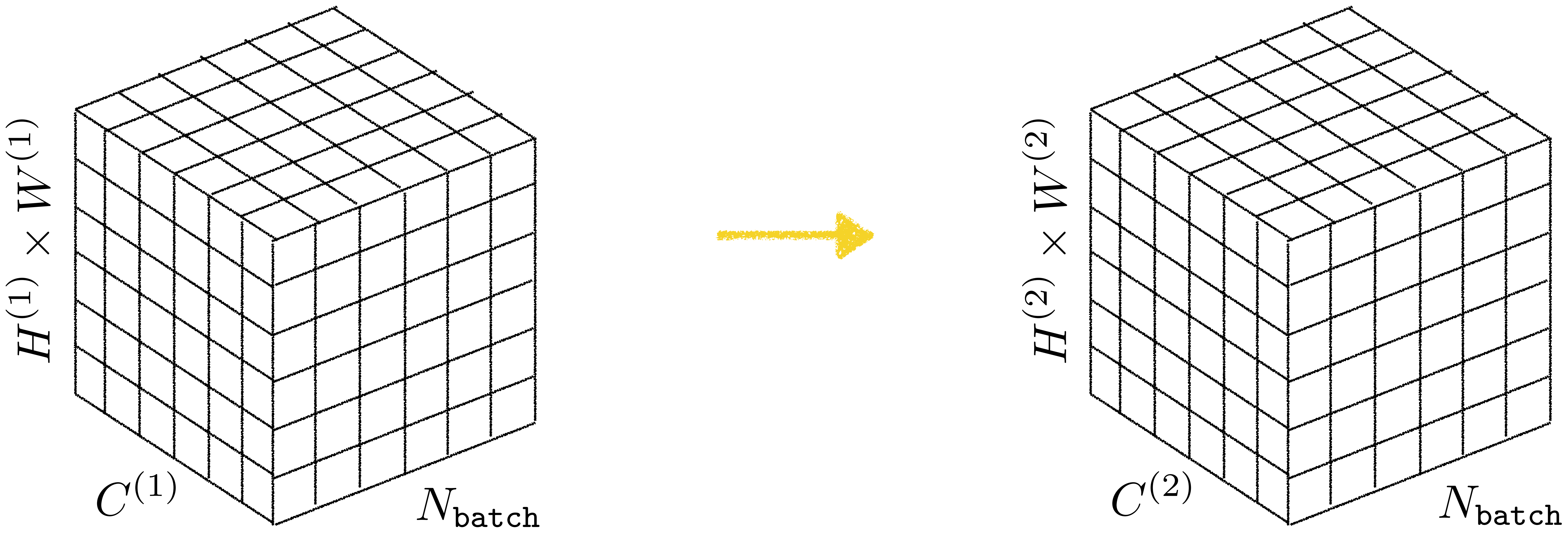
$$\mathbf{h}^{(1)} \in \mathbb{R}^{N_{\mathtt{batch}} \times C^{(1)}} \qquad \mathbf{h}^{(2)} \in \mathbb{R}^{N_{\mathtt{batch}} \times C^{(2)}}$$



Source: Isola, Torralba, Freeman

Processing a layer

$$\mathbf{h}^{(1)} \in \mathbb{R}^{N_{\texttt{batch}} \times H^{(1)} \times W^{(1)} \times C^{(1)}} \qquad \mathbf{h}^{(2)} \in \mathbb{R}^{N_{\texttt{batch}} \times H^{(2)} \times W^{(2)} \times C^{(2)}}$$



Source: Isola, Torralba, Freeman