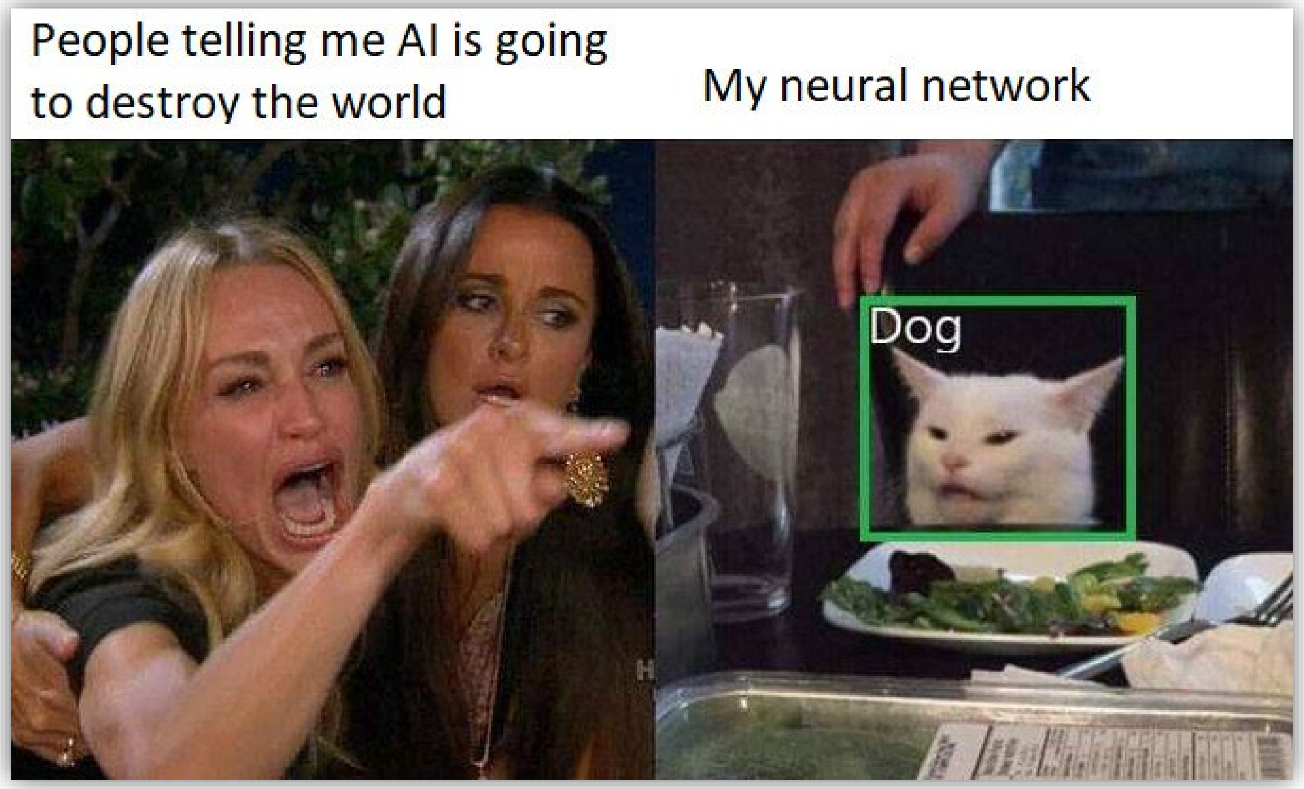
CMSC 491/691

Lecture 9

Neural Networks

for

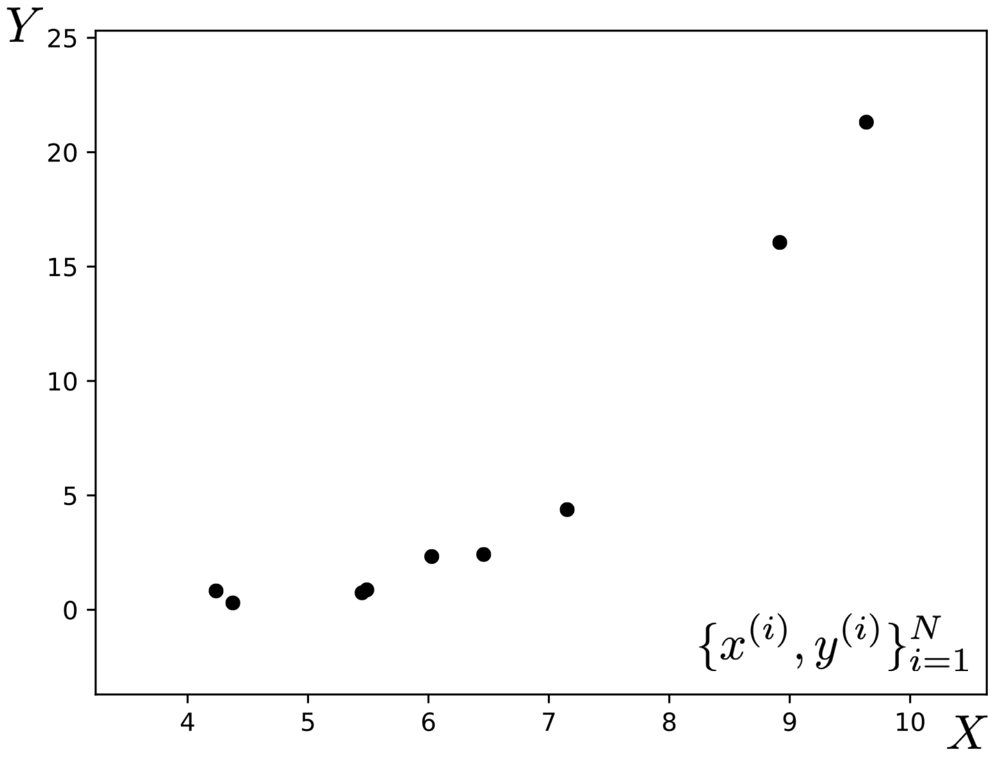
Computer Vision



Some slides from Owens via Isola, Torralba, Freeman

Linear regression

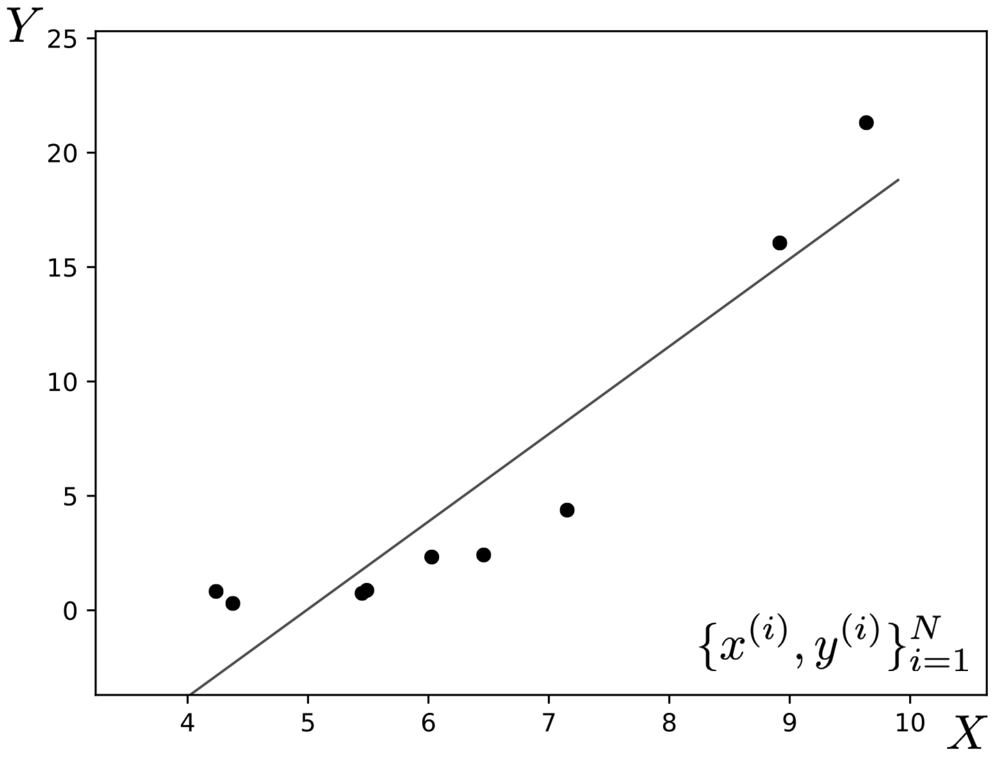
Training data



$$f_\theta(x) = \theta_0 + \theta_1 x$$

Linear regression

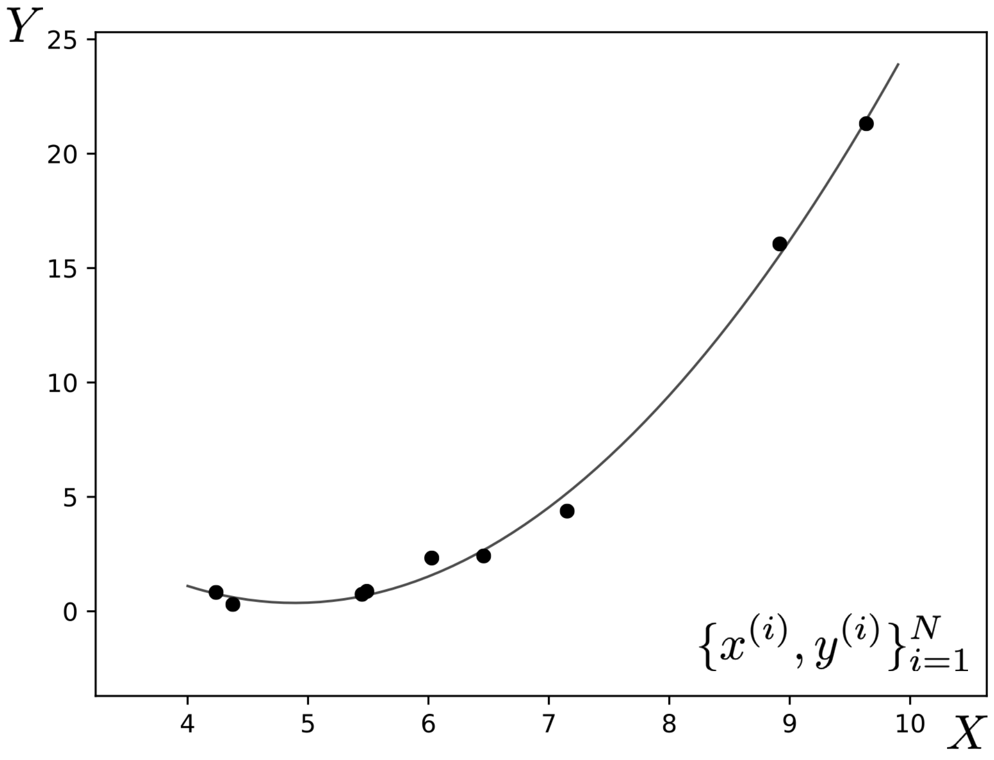
Training data



$$f_\theta(x) = \theta_0 + \theta_1 x$$

Polynomial regression

Training data

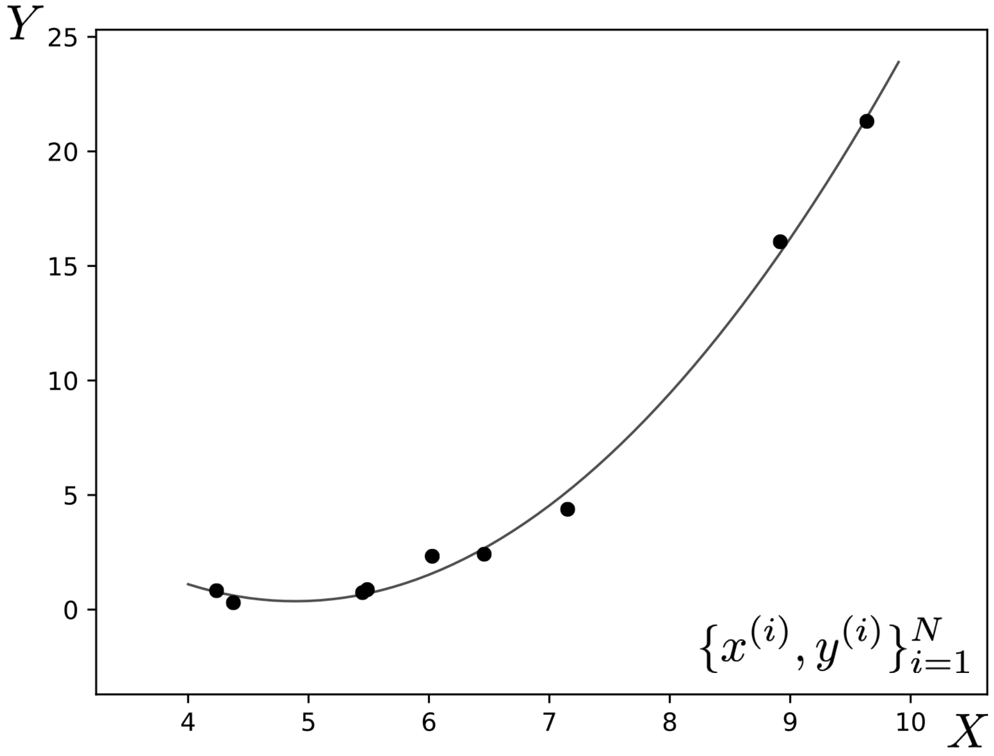


$$f_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

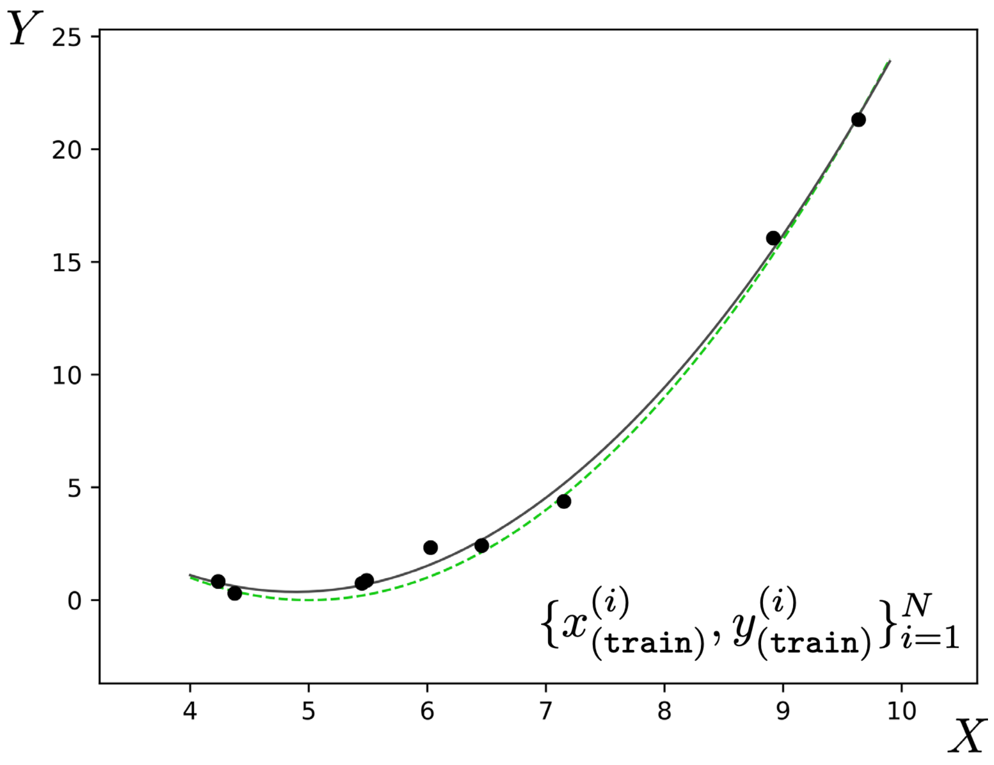
$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

K-th degree polynomial regression

Training data



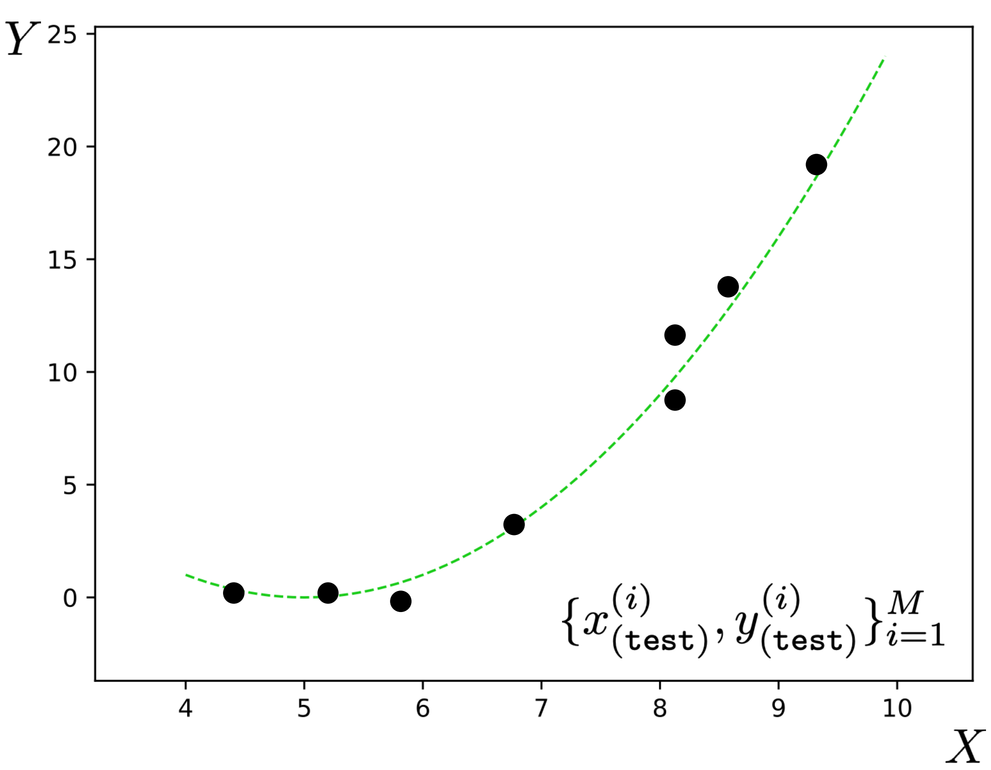
Training data



Training objective:

$$\sum_{i=1}^{N}(f_\theta(x^{(i)}_{\texttt{train}}) - y^{(i)}_{\texttt{train}})^2$$

Test data

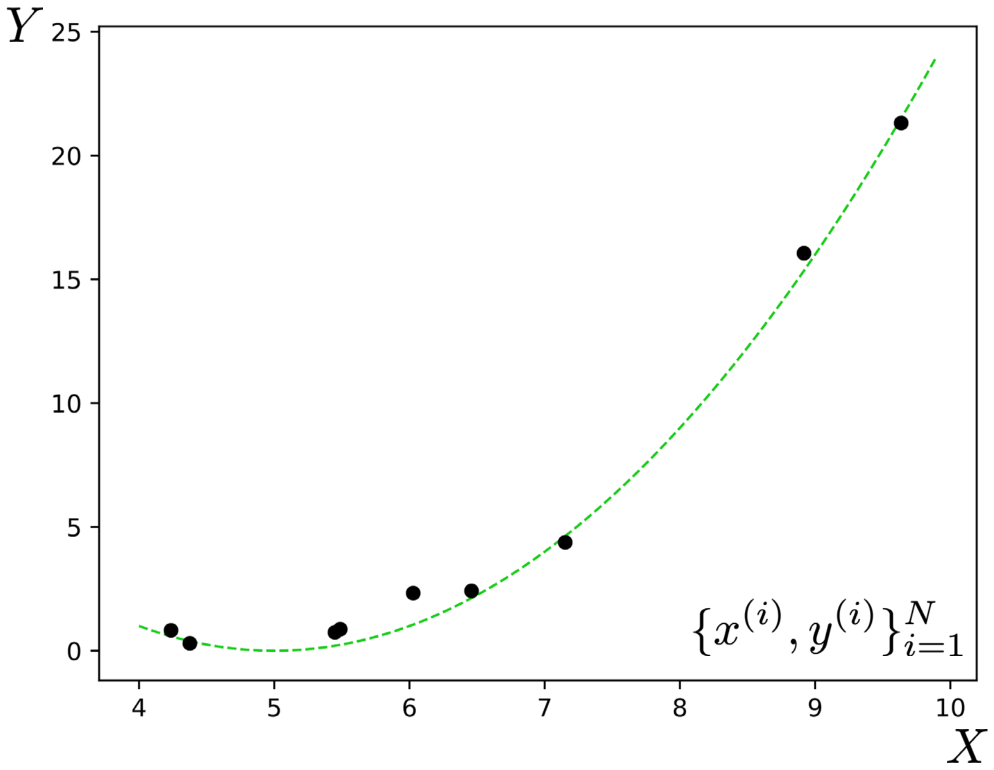


Test time evaluation:

$$\sum_{i=1}^{M}(f_\theta(x^{(i)}_{\texttt{test}}) - y^{(i)}_{\texttt{test}})^2$$

What happens as we add more basis functions?

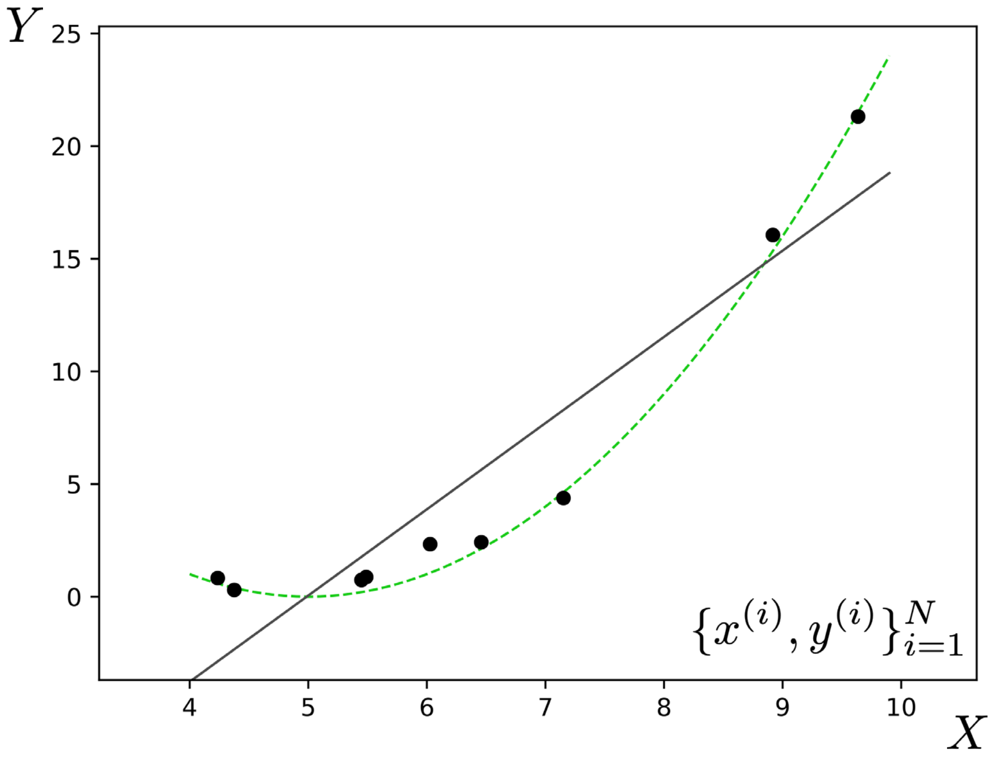
Training data



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

What happens as we add more basis functions?

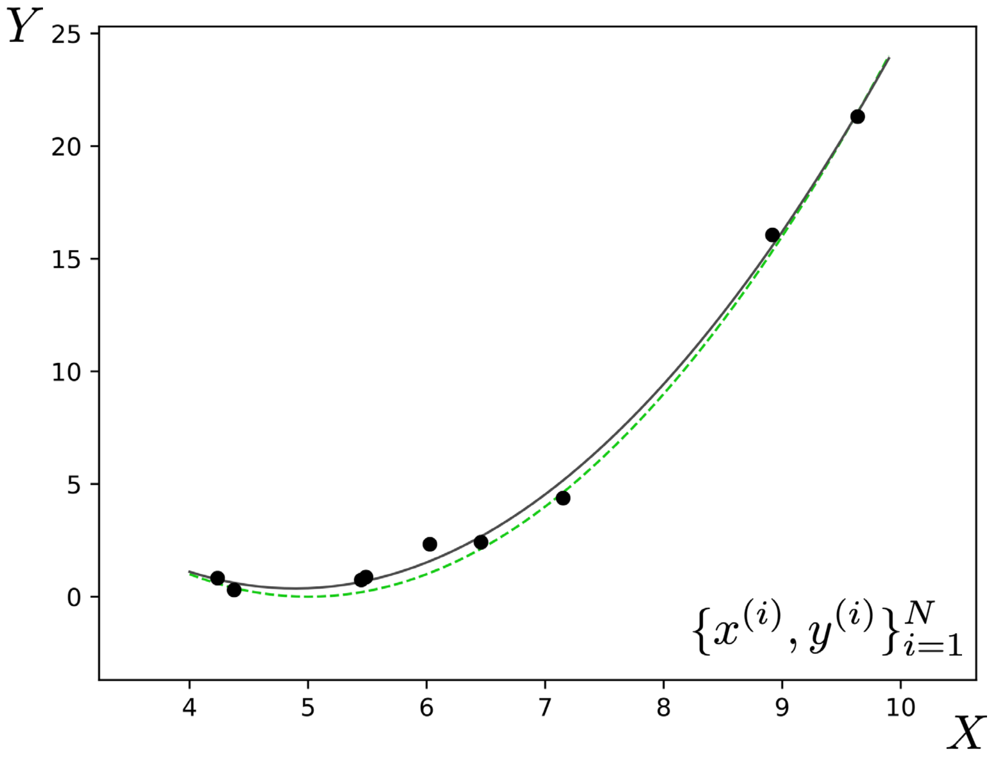
K = 1



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

What happens as we add more basis functions?

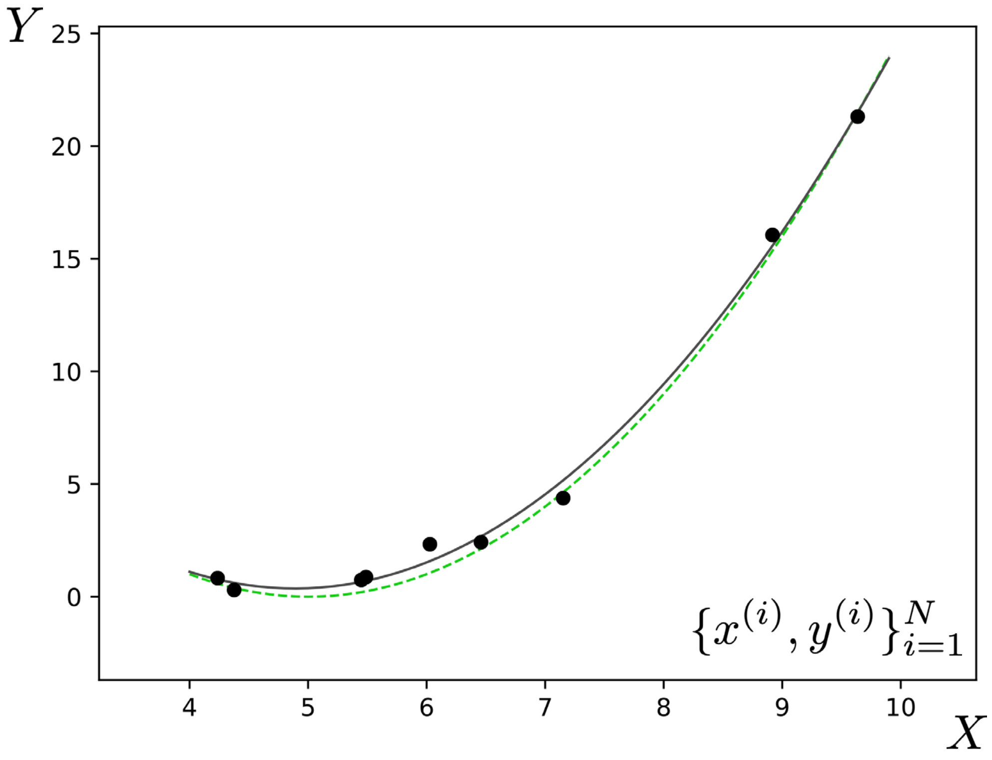
K = 2



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

What happens as we add more basis functions?

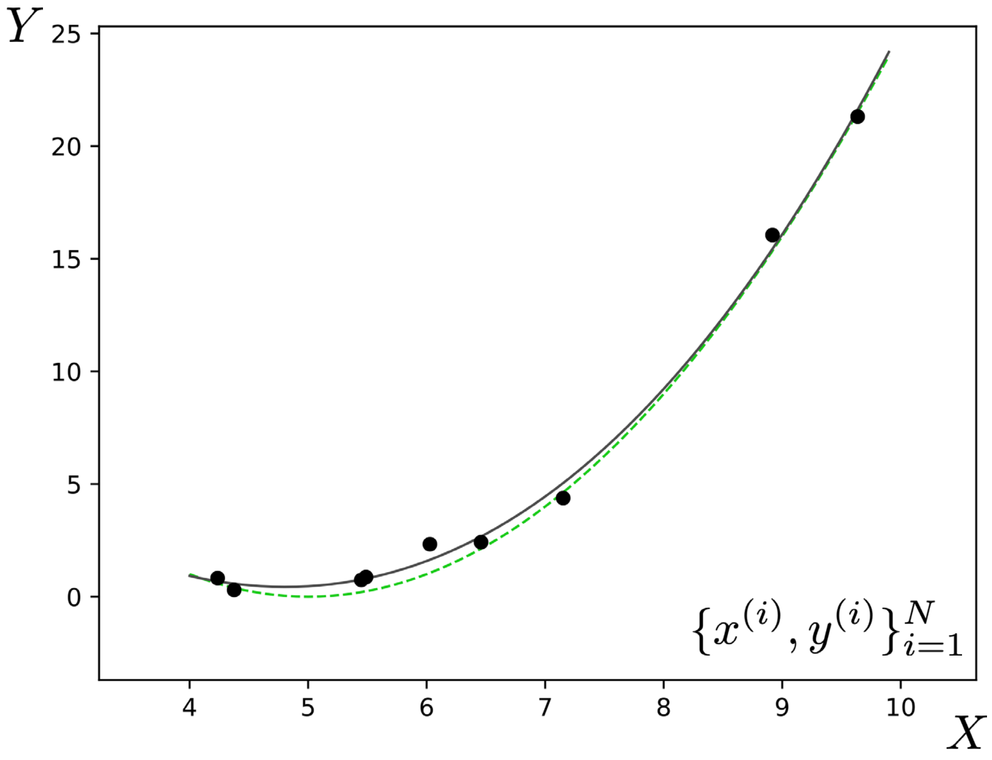
K = 3



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

What happens as we add more basis functions?

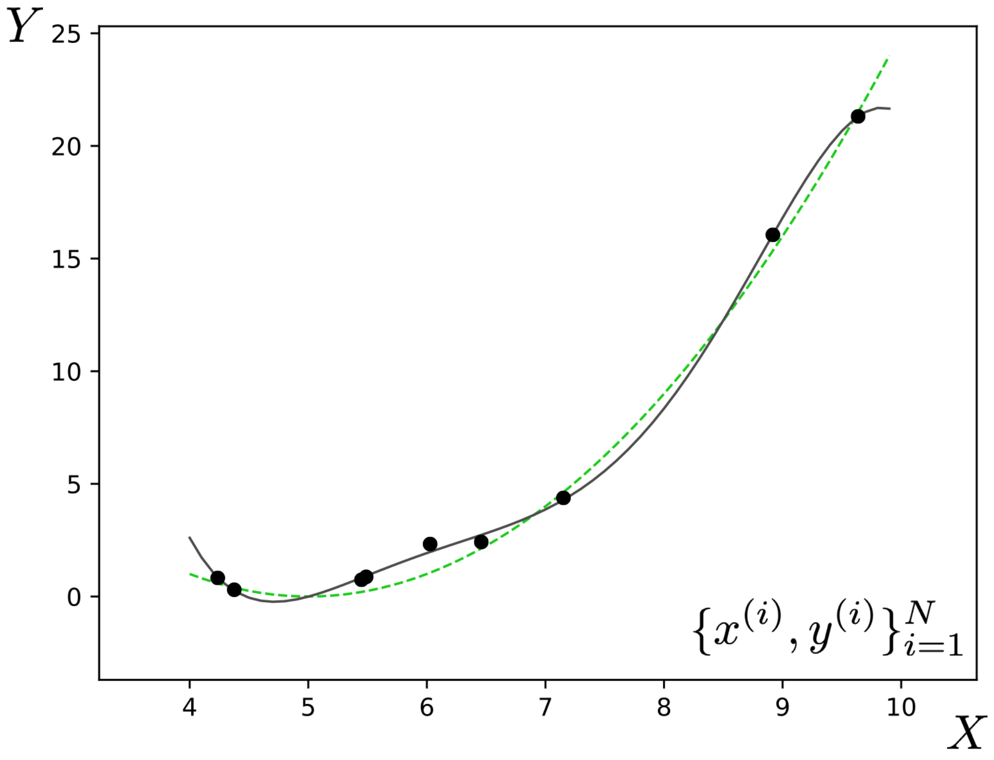
K = 4



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

What happens as we add more basis functions?

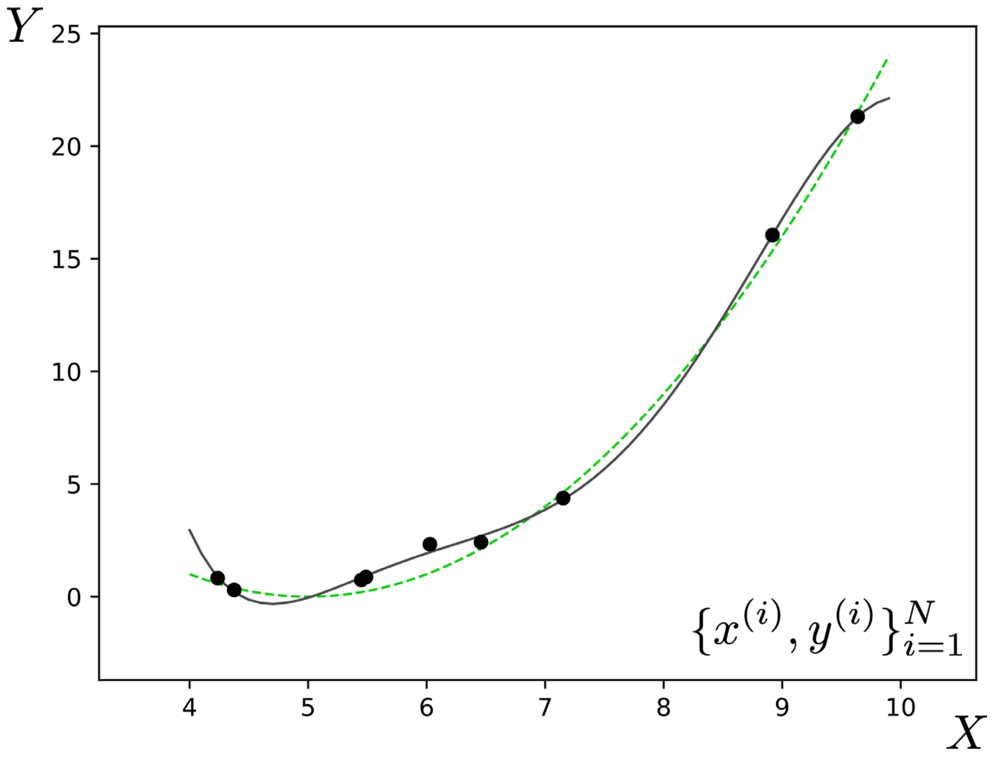
K = 5



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

What happens as we add more basis functions?

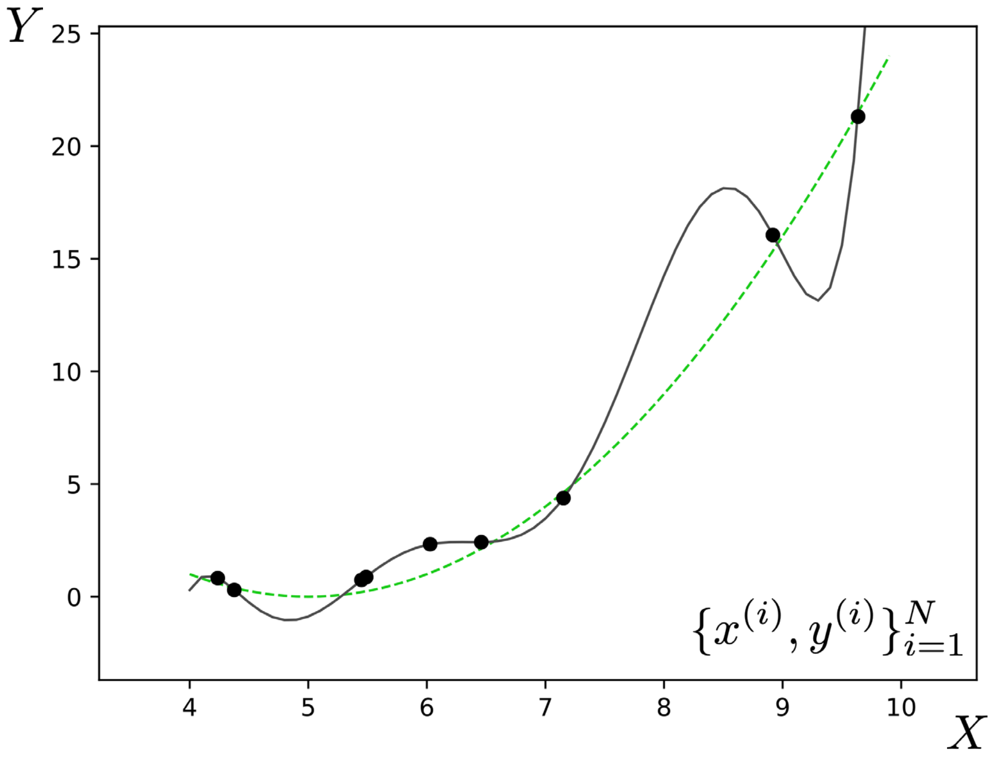
K = 6



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

What happens as we add more basis functions?

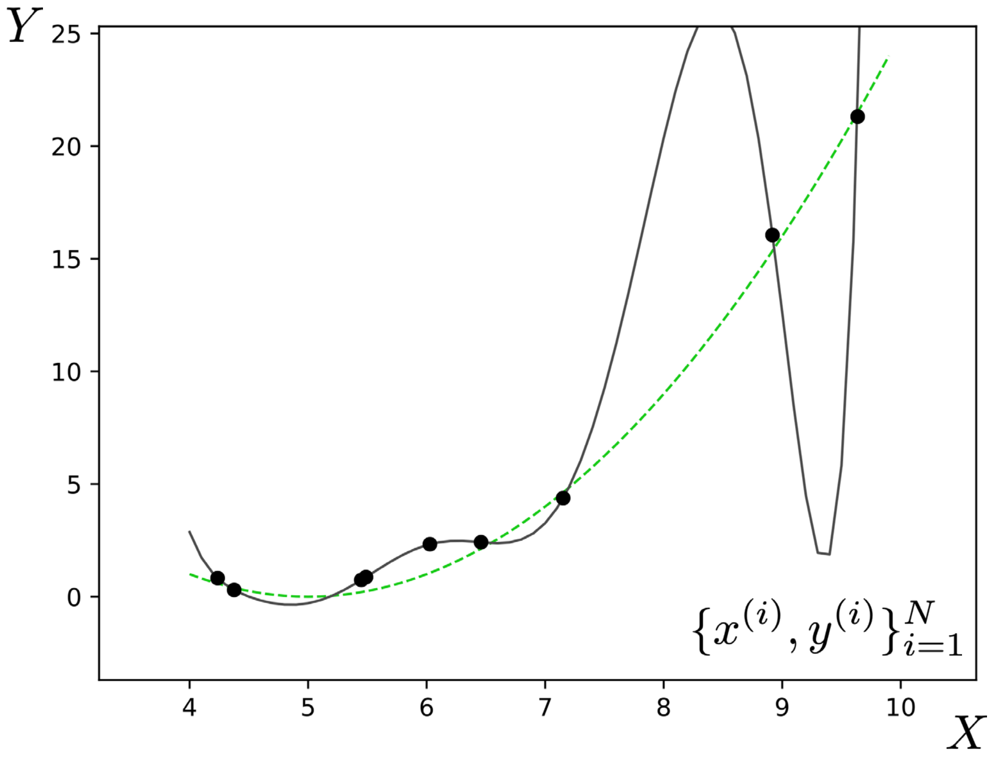
K = 7



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

What happens as we add more basis functions?

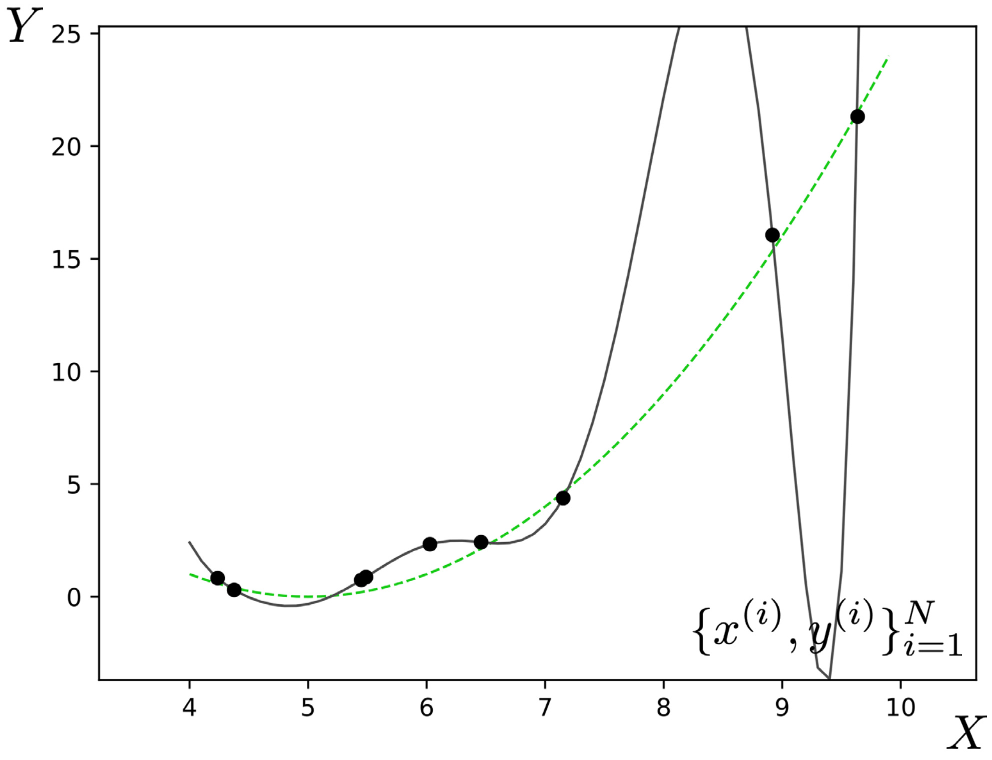
K = 8



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

What happens as we add more basis functions?

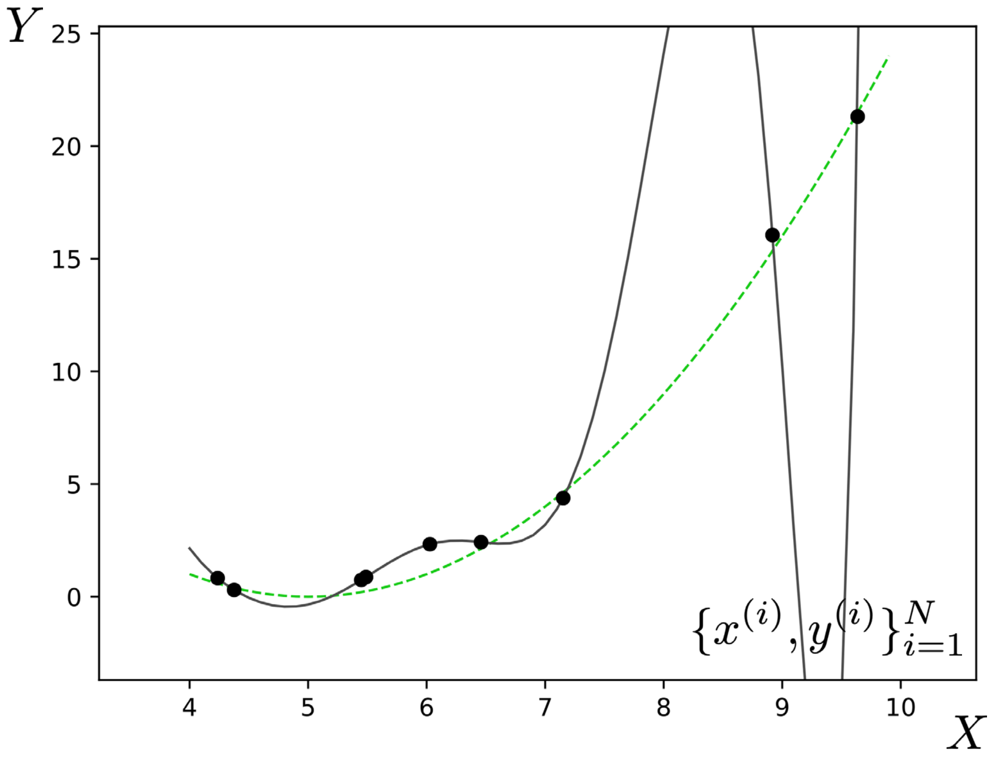
K = 9



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

What happens as we add more basis functions?

K = 10

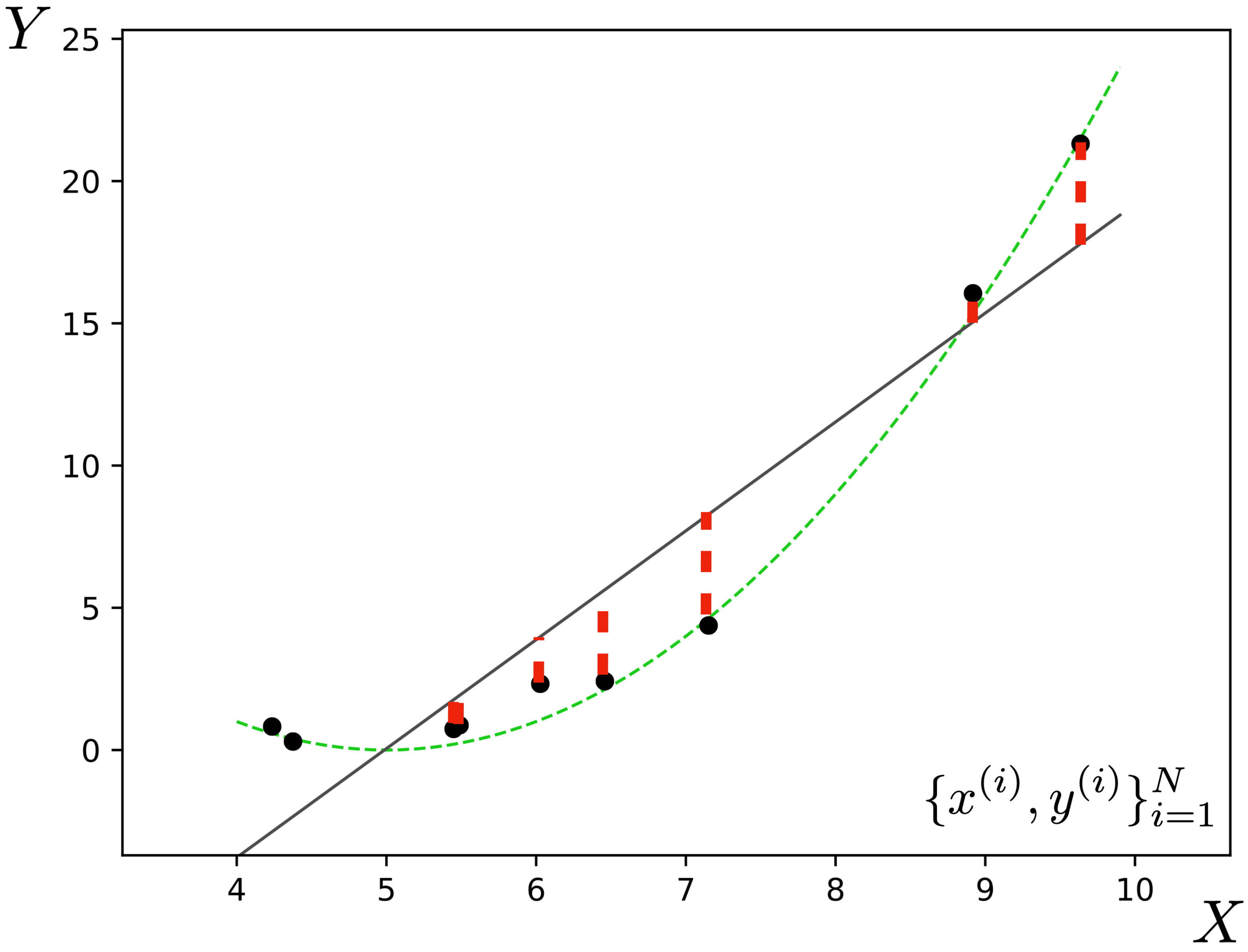


$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

This phenomenon is called **overfitting**.

It occurs when we have too high **capacity** a model, e.g., too many free parameters, too few data points to pin these parameters down.

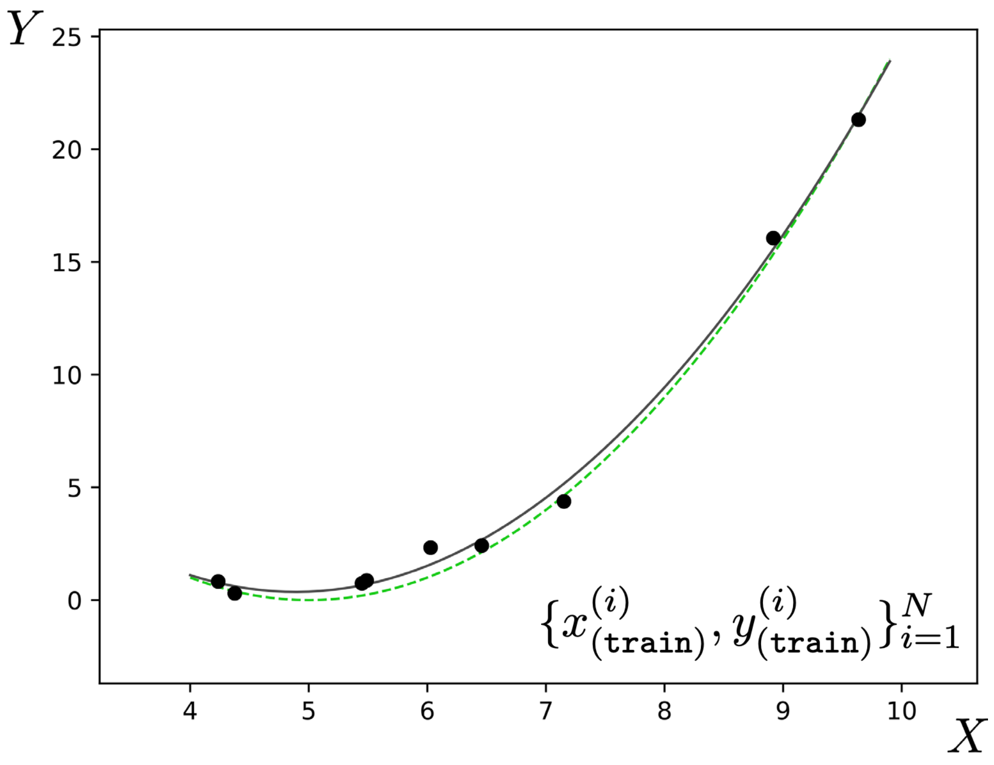
K = 1



When the model does not have the capacity to capture the true function, we call this **underfitting**.

An underfit model will have high error on the training points. This error is known as **approximation error**.

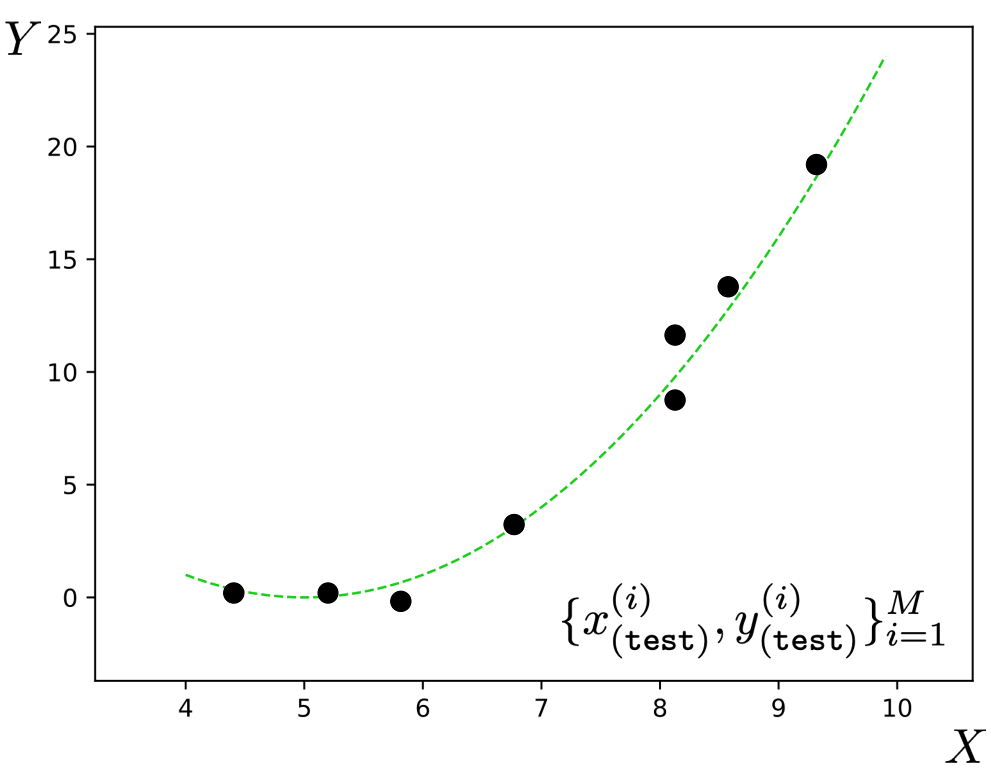
Training data



True **data-generating process**

$p_{\mathtt{data}}$

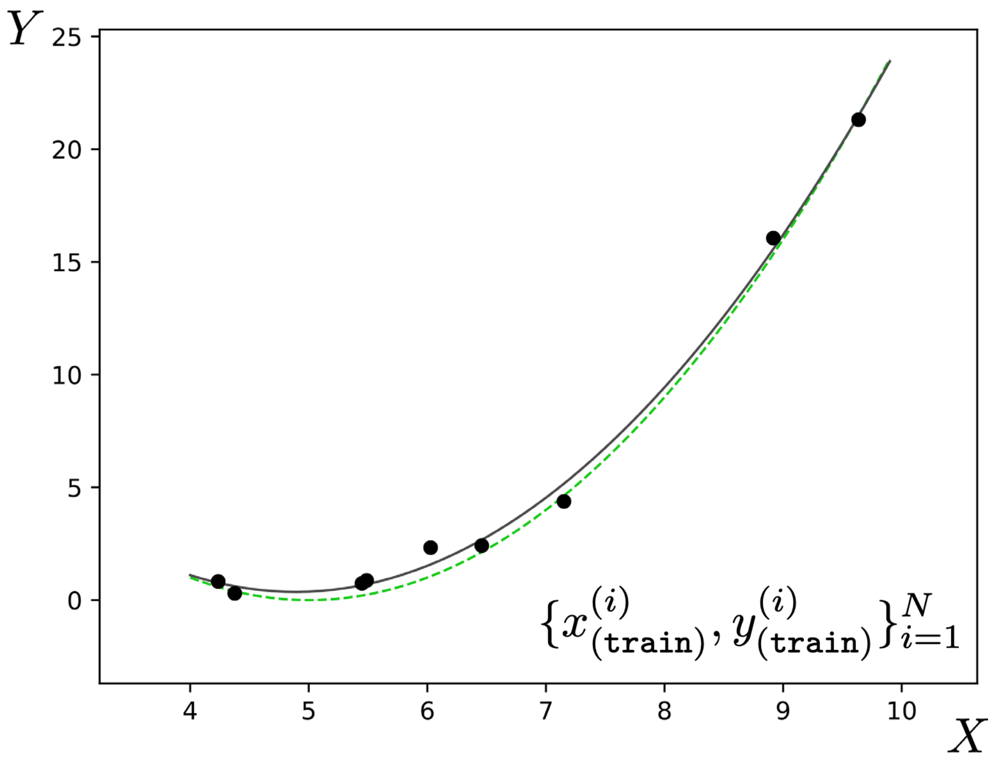
Test data



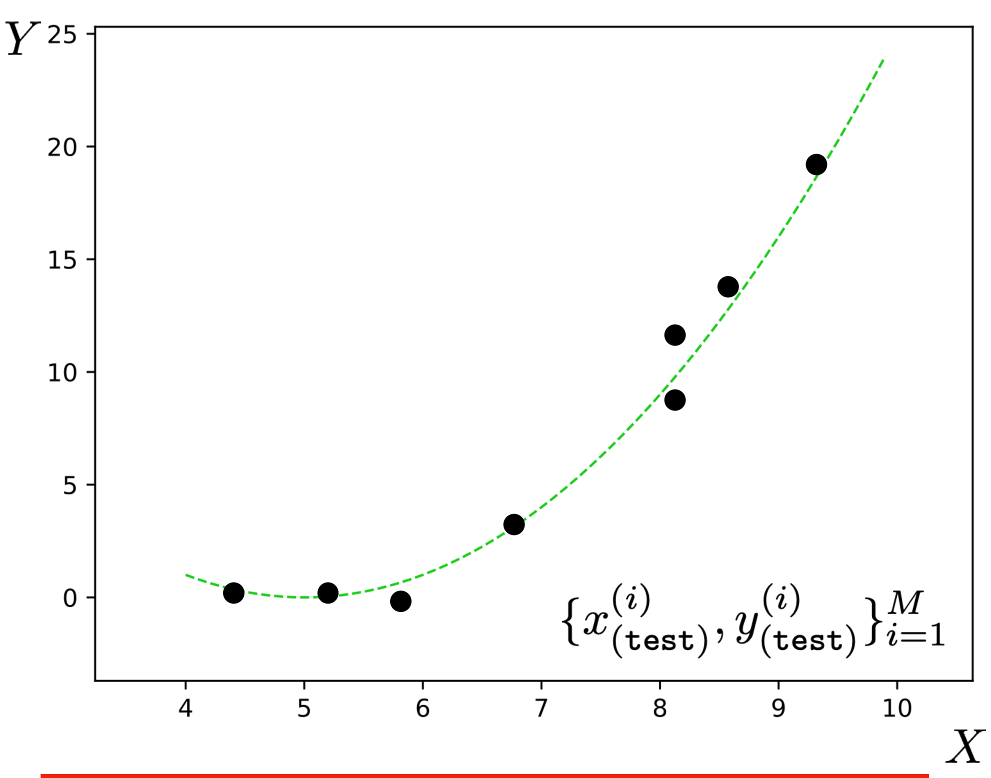
$$\{x^{(i)}_{(\mathtt{train})}, y^{(i)}_{(\mathtt{train})}\} \overset{\mathtt{iid}}{\sim} p_{\mathtt{data}}$$

$$\{x^{(i)}_{(\mathtt{test})}, y^{(i)}_{(\mathtt{test})}\} \overset{\mathtt{iid}}{\sim} p_{\mathtt{data}}$$

Training data



Test data

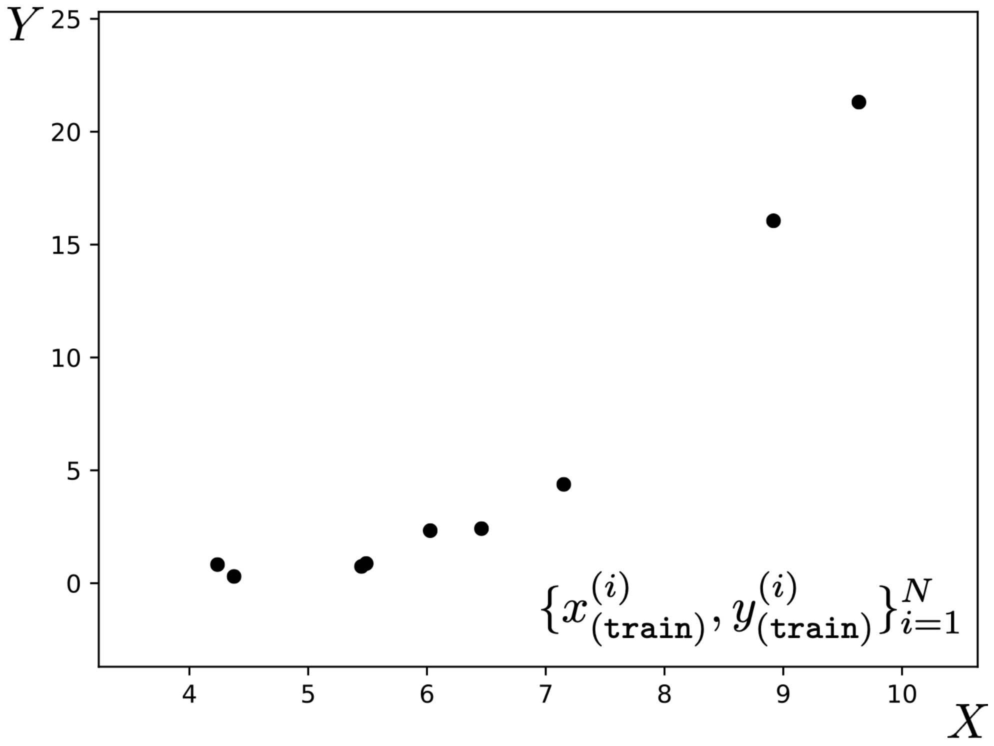


This is a huge assumption!
Almost never true in practice!

$$\{x^{(i)}_{(\texttt{train})}, y^{(i)}_{(\texttt{train})}\} \overset{\texttt{iid}}{\sim} p_{\texttt{data}}$$

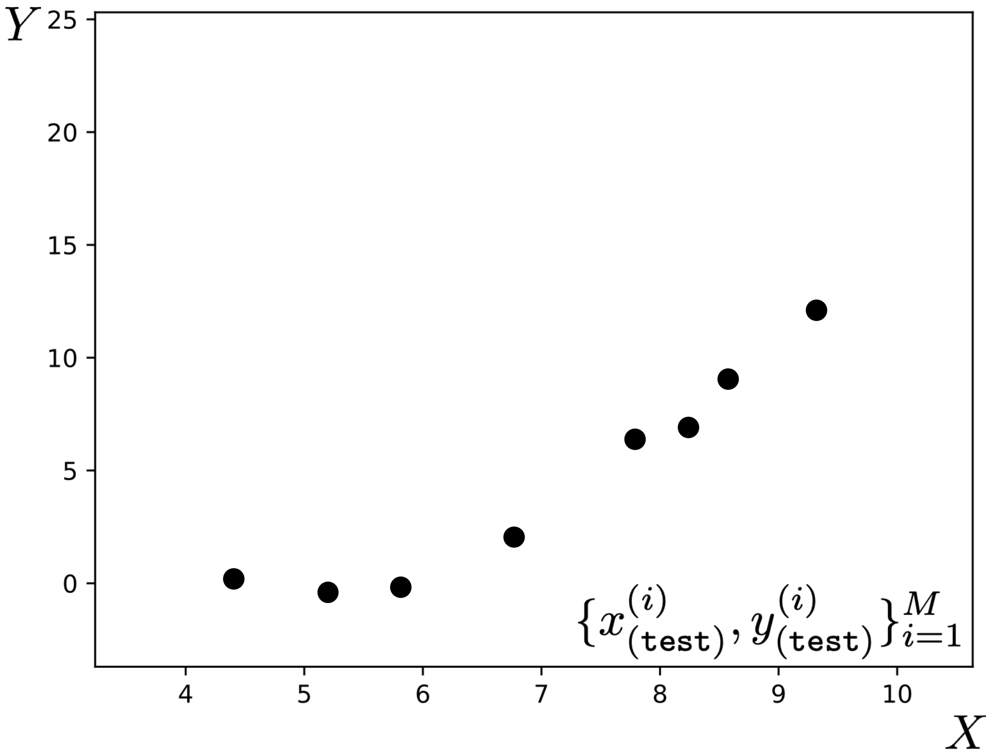
$$\{x^{(i)}_{(\texttt{test})}, y^{(i)}_{(\texttt{test})}\} \overset{\texttt{iid}}{\sim} p_{\texttt{data}}$$

Training data



$\{x^{(i)}_{(\texttt{train})}, y^{(i)}_{(\texttt{train})}\}_{i=1}^{N}$

Test data



$\{x^{(i)}_{(\texttt{test})}, y^{(i)}_{(\texttt{test})}\}_{i=1}^{M}$

$$\{x^{(i)}_{(\texttt{train})}, y^{(i)}_{(\texttt{train})}\} \overset{\texttt{iid}}{\sim} p_{\texttt{train}}$$

$$\{x^{(i)}_{(\texttt{test})}, y^{(i)}_{(\texttt{test})}\} \overset{\texttt{iid}}{\sim} p_{\texttt{test}}$$

Much more commonly, we have

$$p_{\texttt{train}} \neq p_{\texttt{test}}$$

Artificial Intelligence

$$\hat{y} = \boldsymbol{w}^\top \boldsymbol{x} + b$$

Parametric Approach

Image

Array of **32x32x3** numbers
(3072 numbers total)

→ f(**x**,**W**) → **10** numbers giving class scores

↑

W

parameters
or weights

Parametric Approach: Linear Classifier

$$f(x,W) = Wx$$

Image

Array of **32x32x3** numbers
(3072 numbers total)

→ f(**x**,**W**) → **10** numbers giving class scores

↑

W

parameters
or weights

Parametric Approach: Linear Classifier

3072x1

$f(x,W) = Wx$

10x1 10x3072

Image

Array of **32x32x3** numbers
(3072 numbers total)

→ $f(\mathbf{x},\mathbf{W})$ → **10** numbers giving class scores

↑

W

parameters
or weights

Parametric Approach: Linear Classifier

$$f(x,W) = Wx + b$$

- $f(x,W)$: **10x1**
- W: **10x3072**
- x: **3072x1**
- b: **10x1**

Image

Array of **32x32x3** numbers
(3072 numbers total)

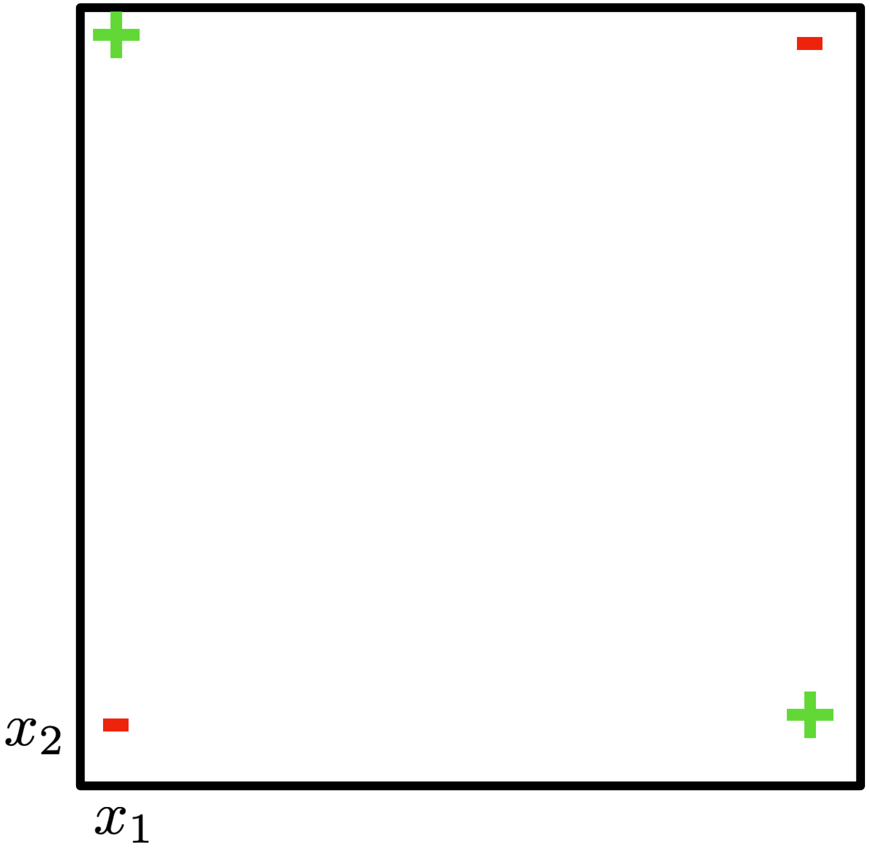
→ $f(\mathbf{x},\mathbf{W})$ → **10** numbers giving class scores

↑

W

parameters
or weights

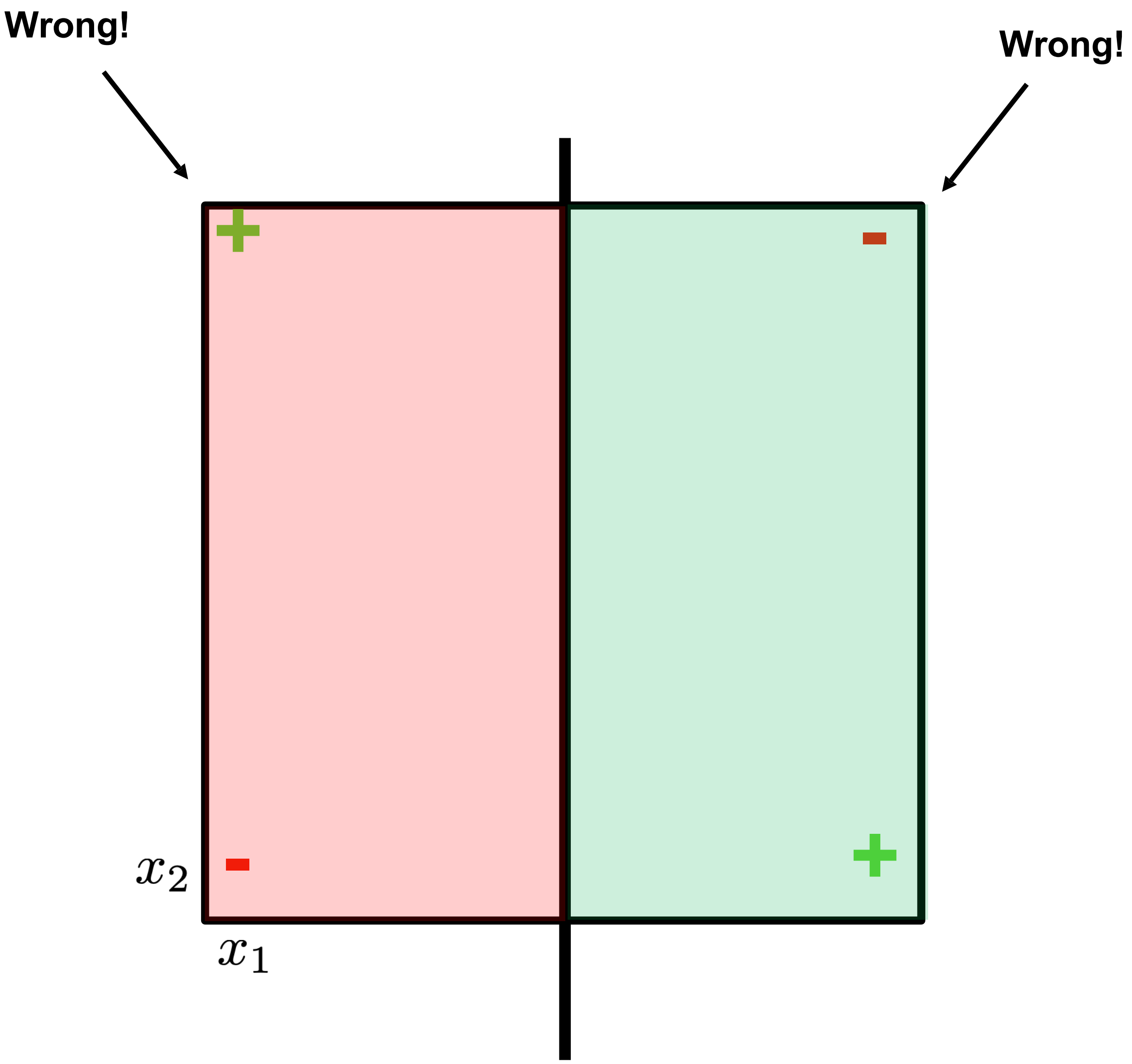
Limitations to linear classifiers



x_1 \ x_2	0	1
0	0	1
1	1	0

XOR

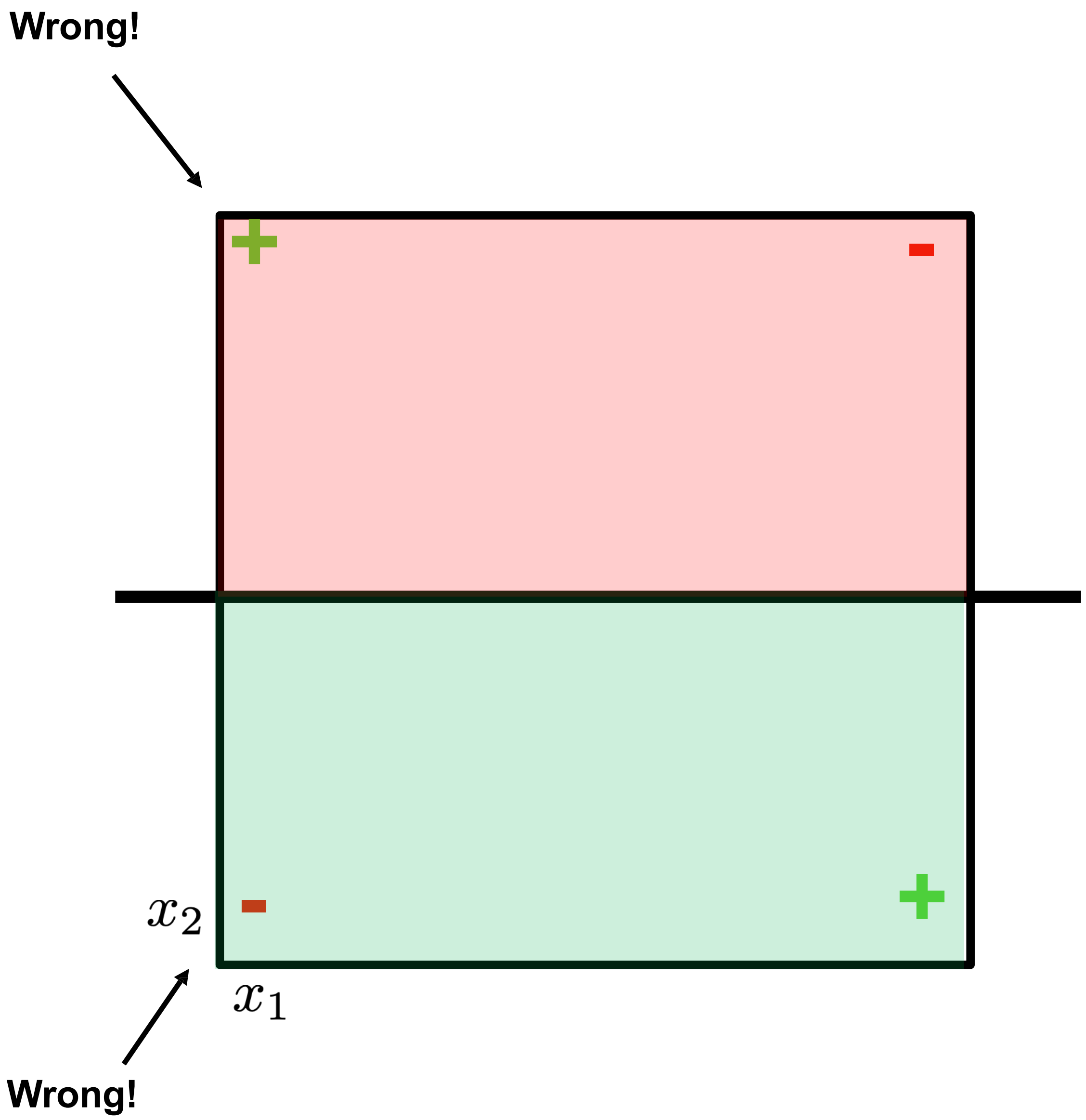
Limitations to linear classifiers



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

XOR

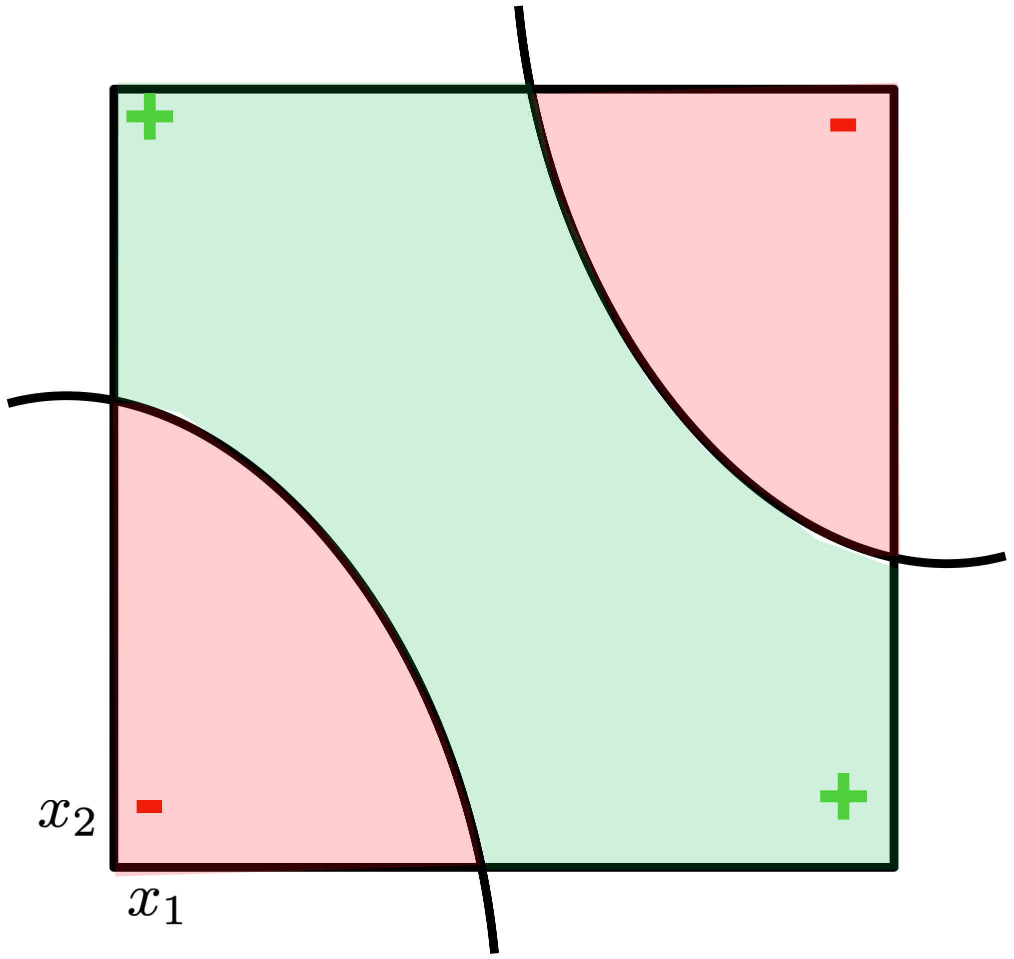
Limitations to linear classifiers



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

XOR

Goal: Non-linear decision boundary

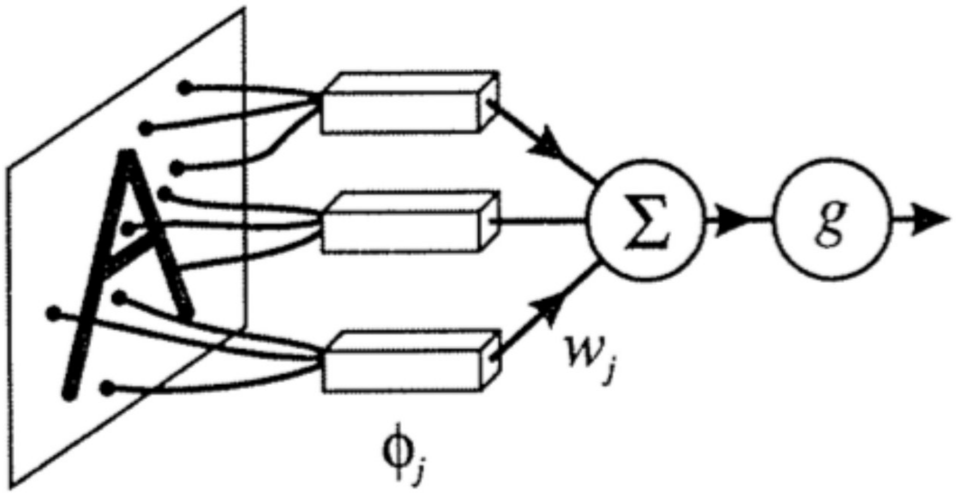


x_1 \ x_2	0	1
0	0	1
1	1	0

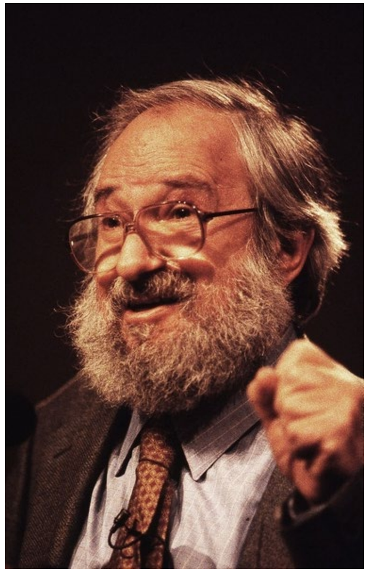
XOR

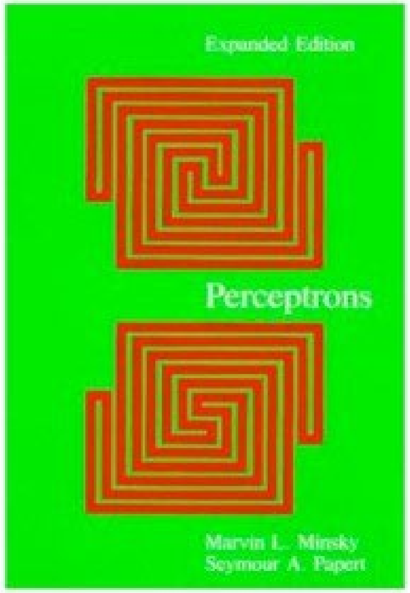
Perceptron

- In 1957 Frank Rosenblatt invented the perceptron
- Computers at the time were too slow to run the perceptron, so Rosenblatt built a special purpose machine with adjustable resistors
- New York Times Reported: “The Navy revealed the embryo of an electronic computer that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence”



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.

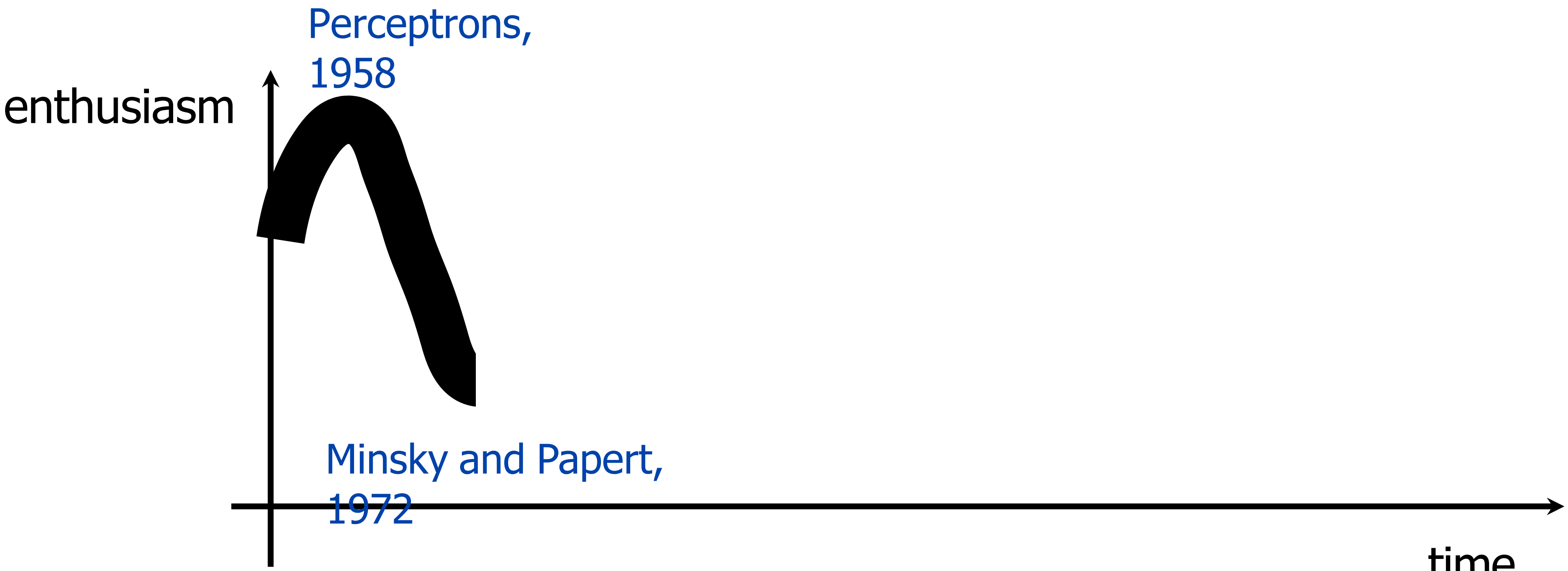
Artificial-intelligence research, which for a time concentrated on the programming of ton Neumann computers, is swinging back to the idea that intelligence might emerge from the activity of networks of neuronlike entities. Minsky and Papert's book was the first example of a mathematical analysis carried 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."

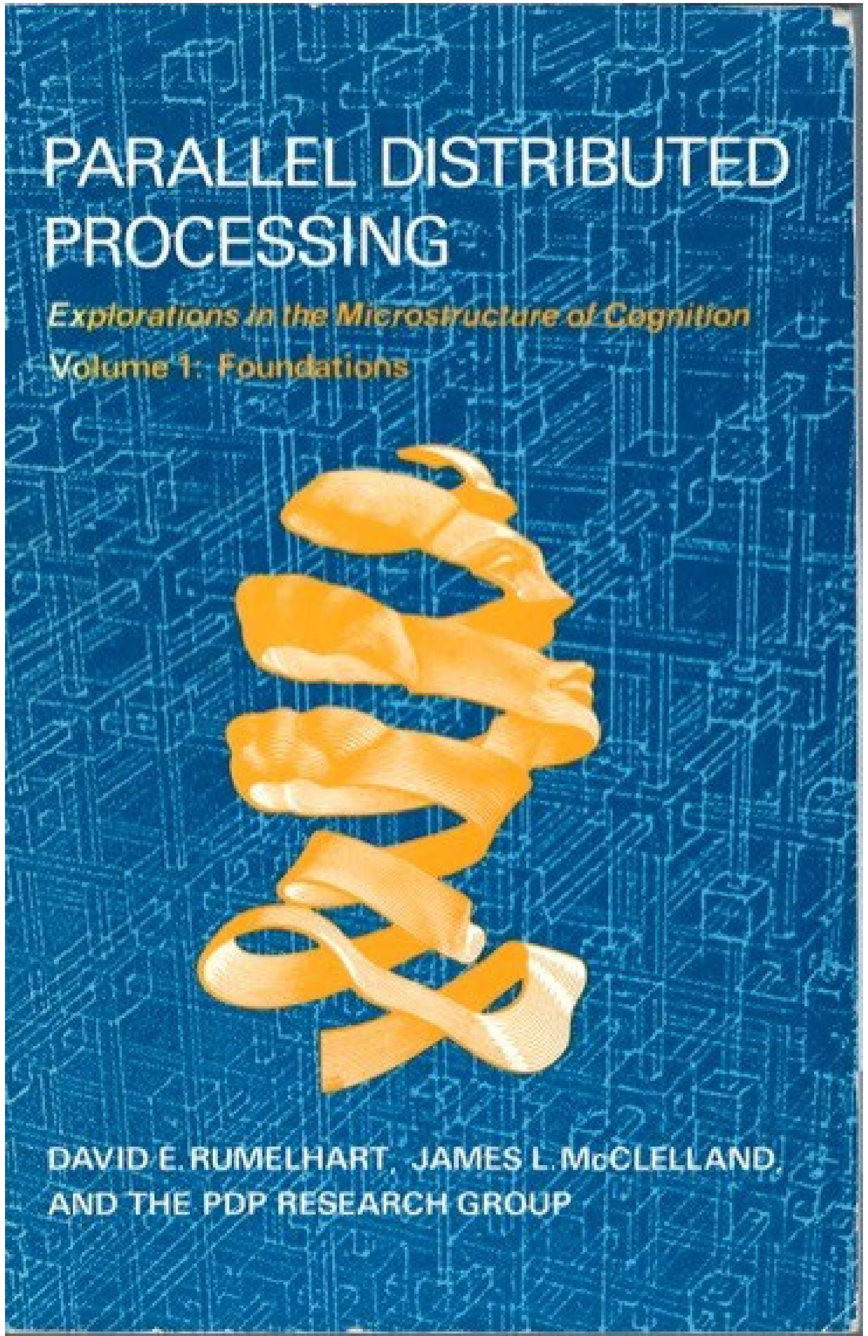
FOR BUYING OPTIONS, START HERE

Select Shipping Destination

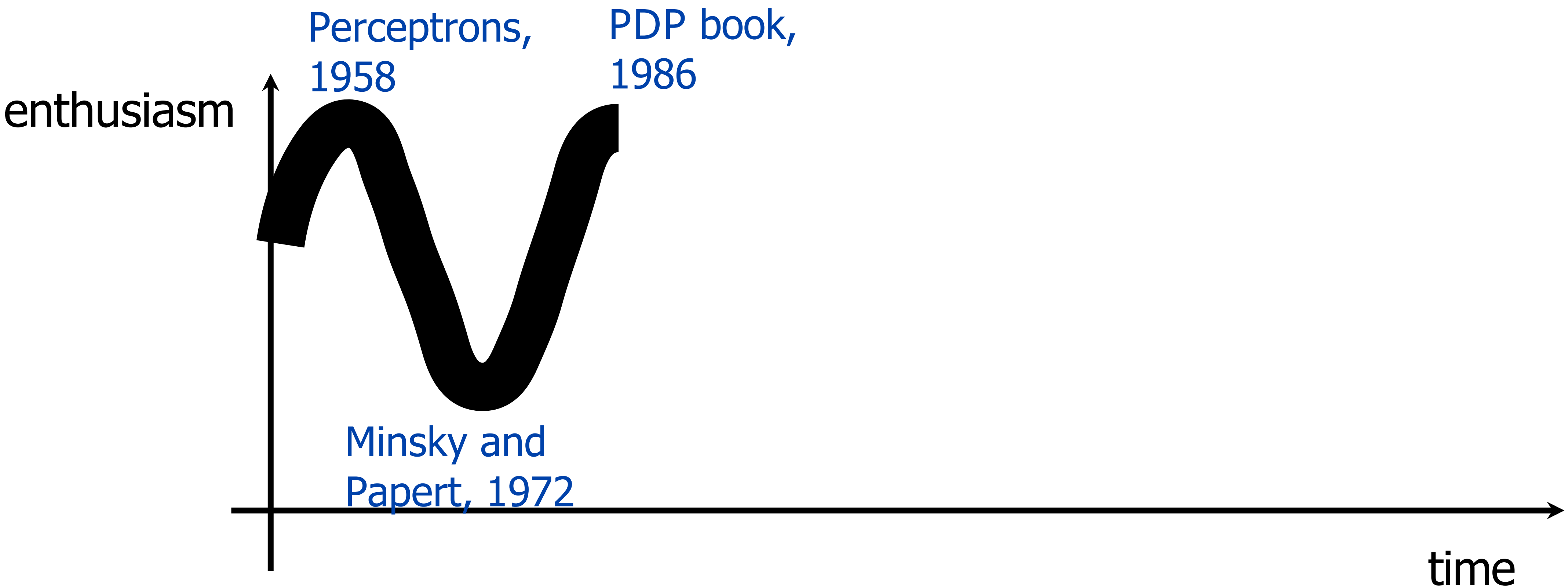
Paperback | **$35.00 Short** | **£24.95** | ISBN: 9780262631112 | 308 pp. | 6 x 8.9 in | December 1987


Perceptrons,
1958
enthusiasm
Minsky and Papert,
1972
time

Parallel Distributed Processing (PDP), 1986



Source: Isola, Torralba, Freeman



Source: Isola, Torralba, Freeman

LeCun convolutional neural networks

PROC. OF THE IEEE, NOVEMBER 1998

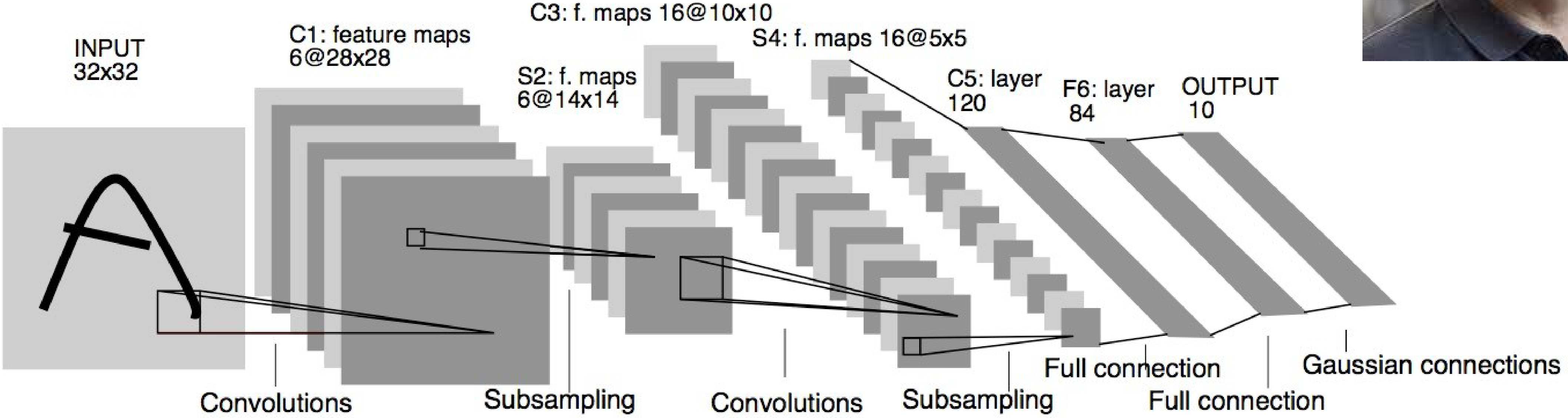


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

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

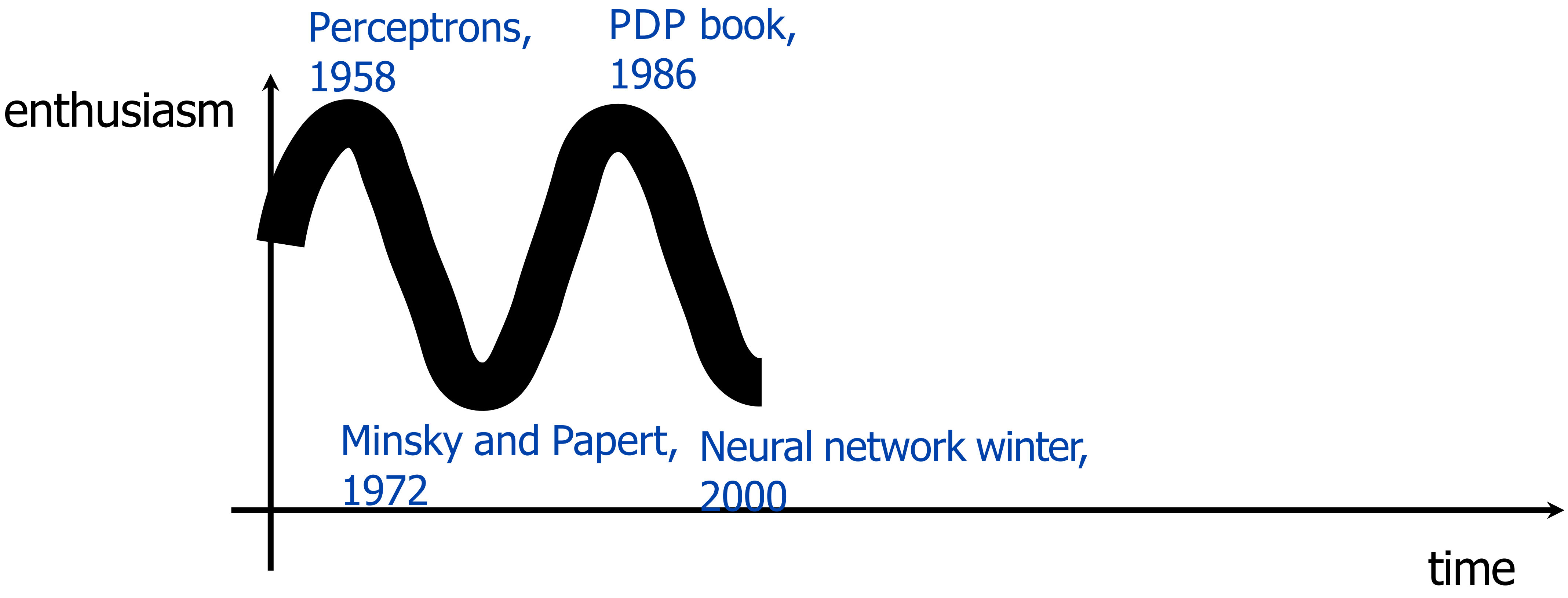
Source: Isola, Torralba, Freeman

Yann LeCun

Was at Bell Labs when this video was recorded

Now
Prof @ NYU
Chief Scientist @ Meta

Turing Award 2018 (shared with Hinton and Bengio)

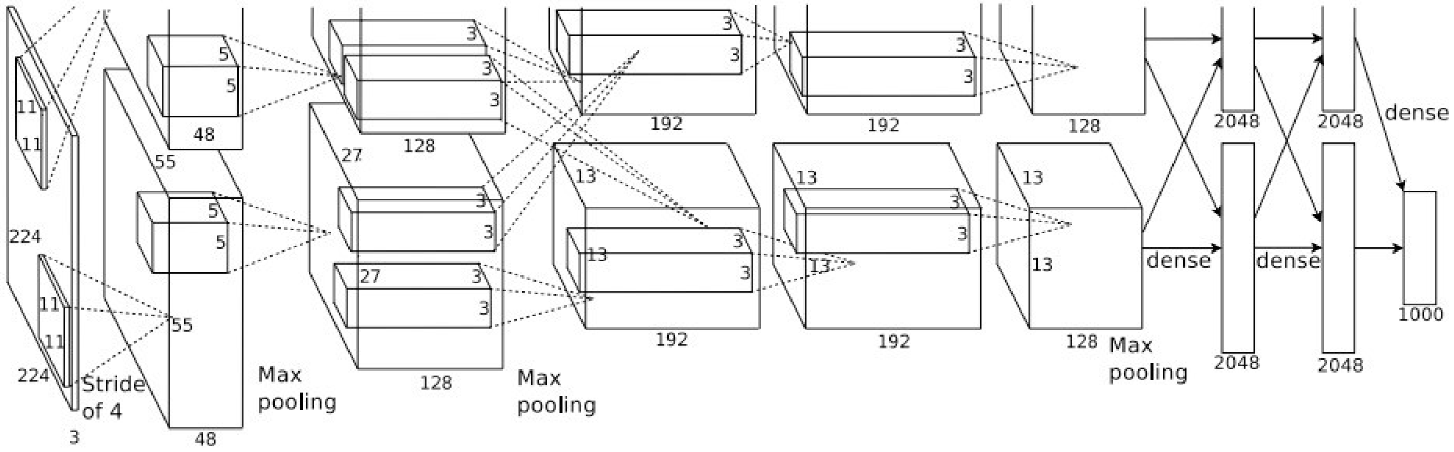

Perceptrons,
1958
PDP book,
1986
enthusiasm
Minsky and Papert,
1972
Neural network winter,
2000
time

ImageNet:
First (?) large-scale computer vision dataset

- Millions of images; 1000 categories
- **PI: Fei-Fei Li**
  - Then: Prof, Princeton
  - Now: Prof, Stanford
- 2019 Longuet-Higgins Prize
  - Some argued that Li deserved the 2018 Turing Award along with Hinton, LeCun, Bengio
  - Their work could not have been empirically tested without ImageNet!

Krizhevsky, Sutskever, and Hinton, NeurIPS 2012

“AlexNet”

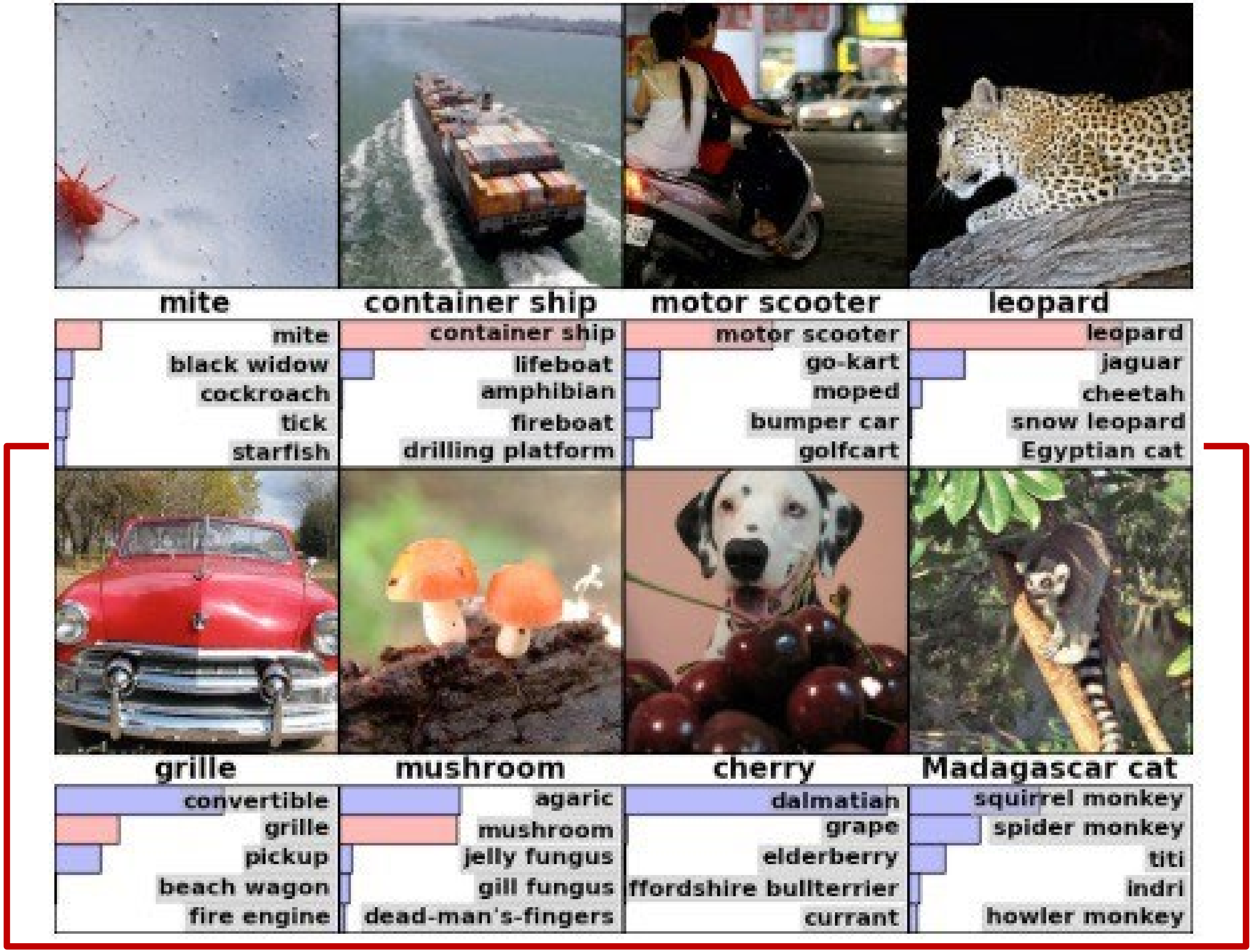


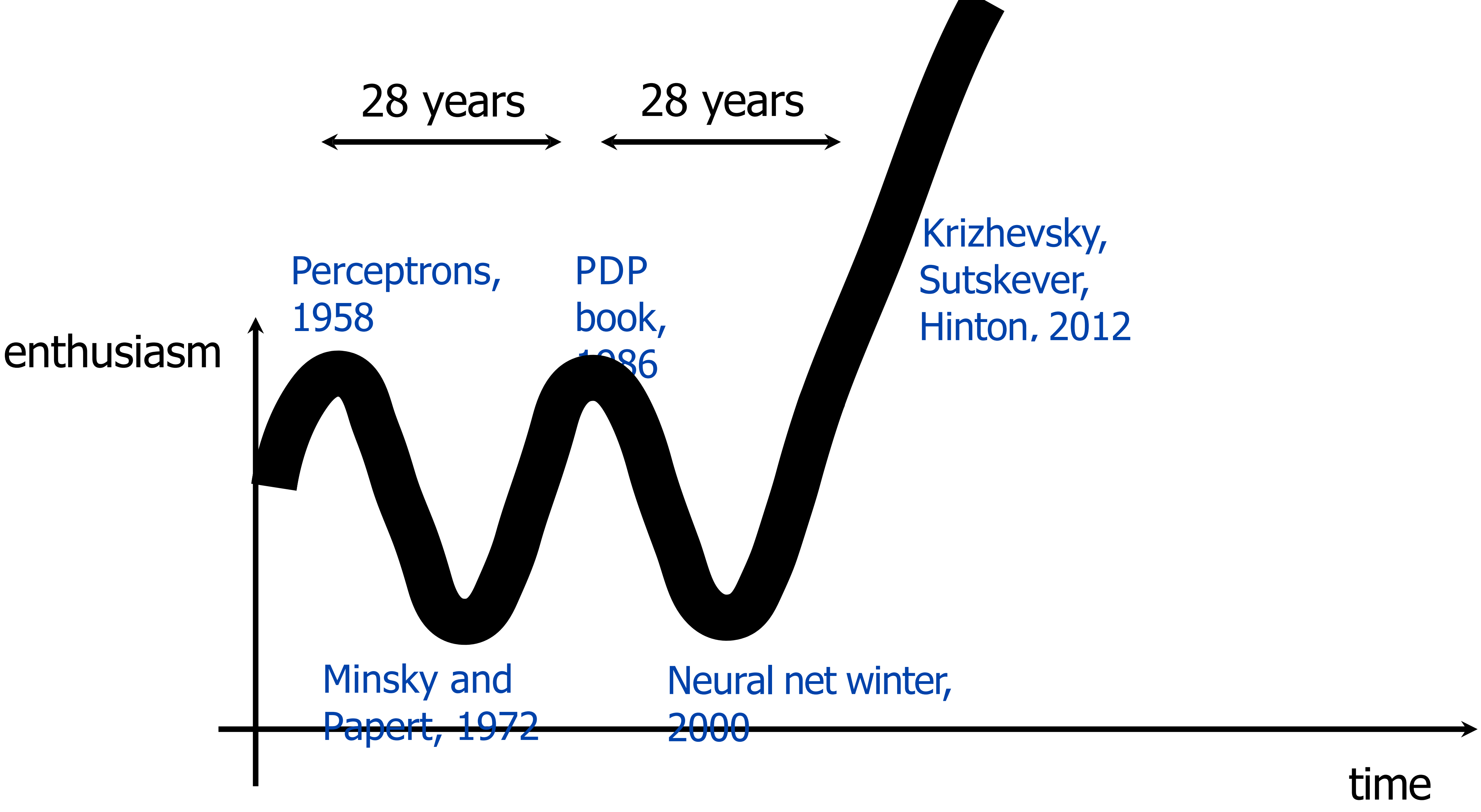
Got all the “pieces” right, e.g.,

- Trained on ImageNet
- 8 layer architecture (for reference: today we have architectures with 100+ layers)
- Allowed for multi-GP training

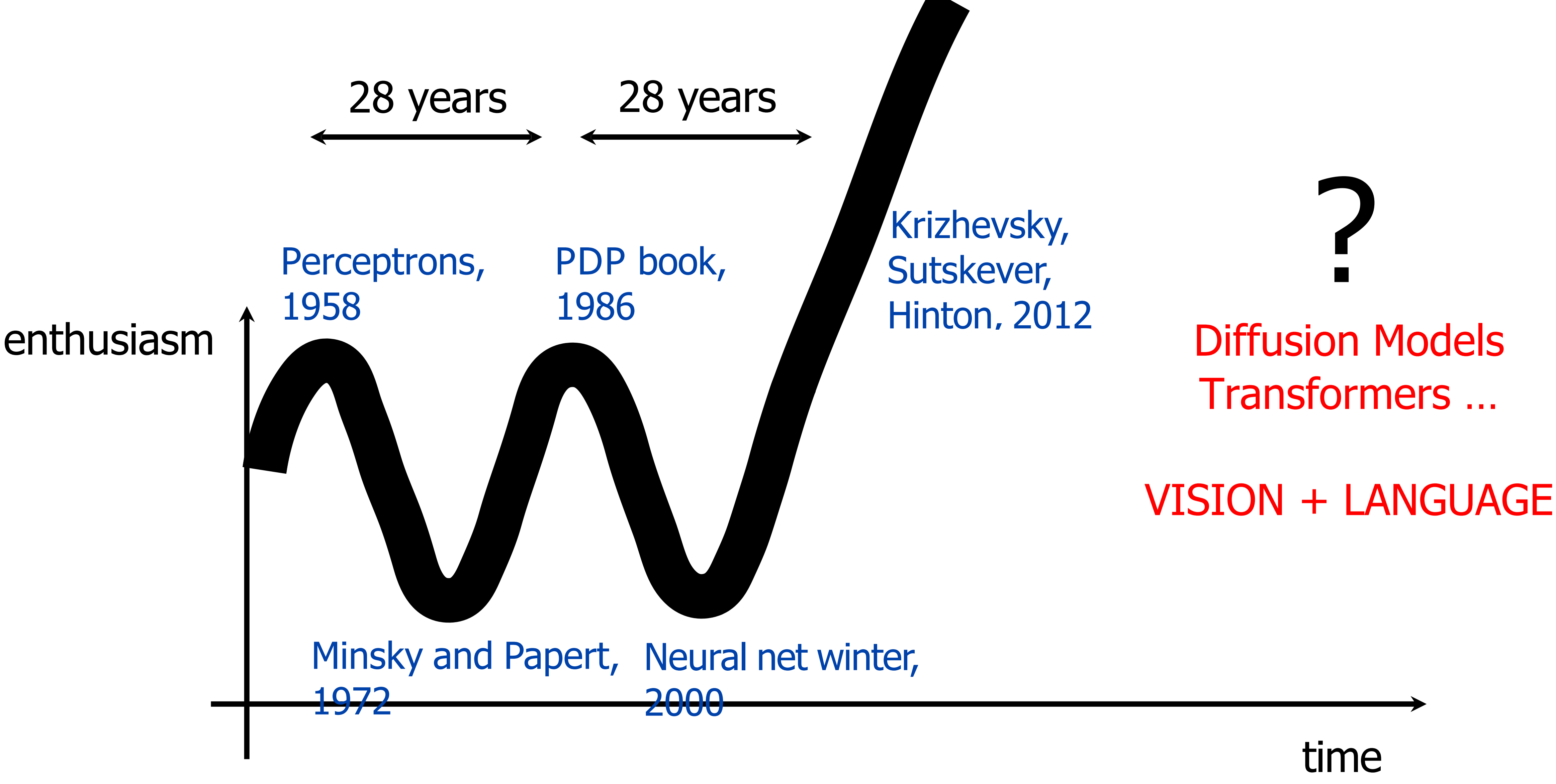
Krizhevsky, Sutskever, and Hinton, NeurIPS 2012

Krizhevsky, Sutskever, and Hinton, NeurIPS 2012



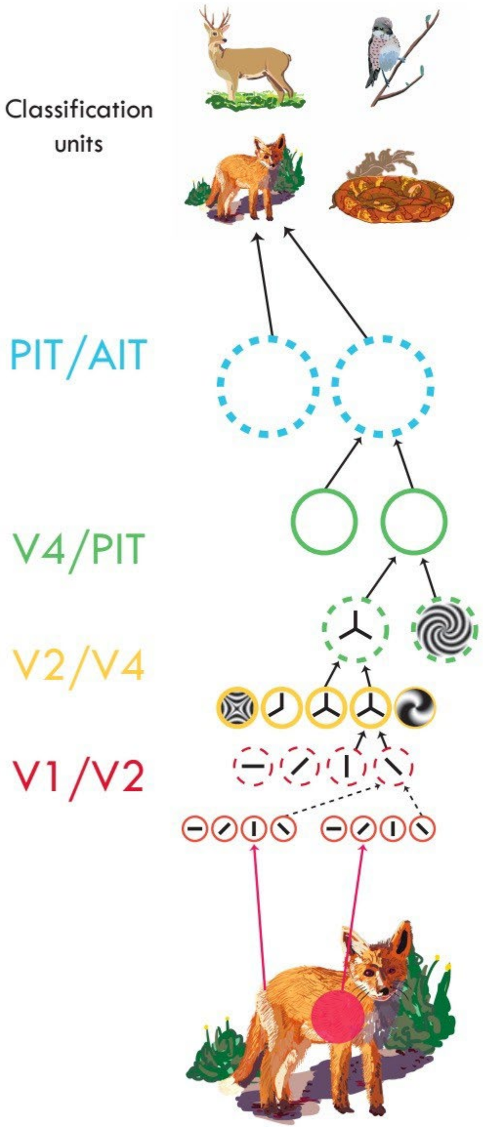


Source: Isola, Torralba, Freeman



Source: Isola, Torralba, Freeman

Inspiration: Hierarchical Representations

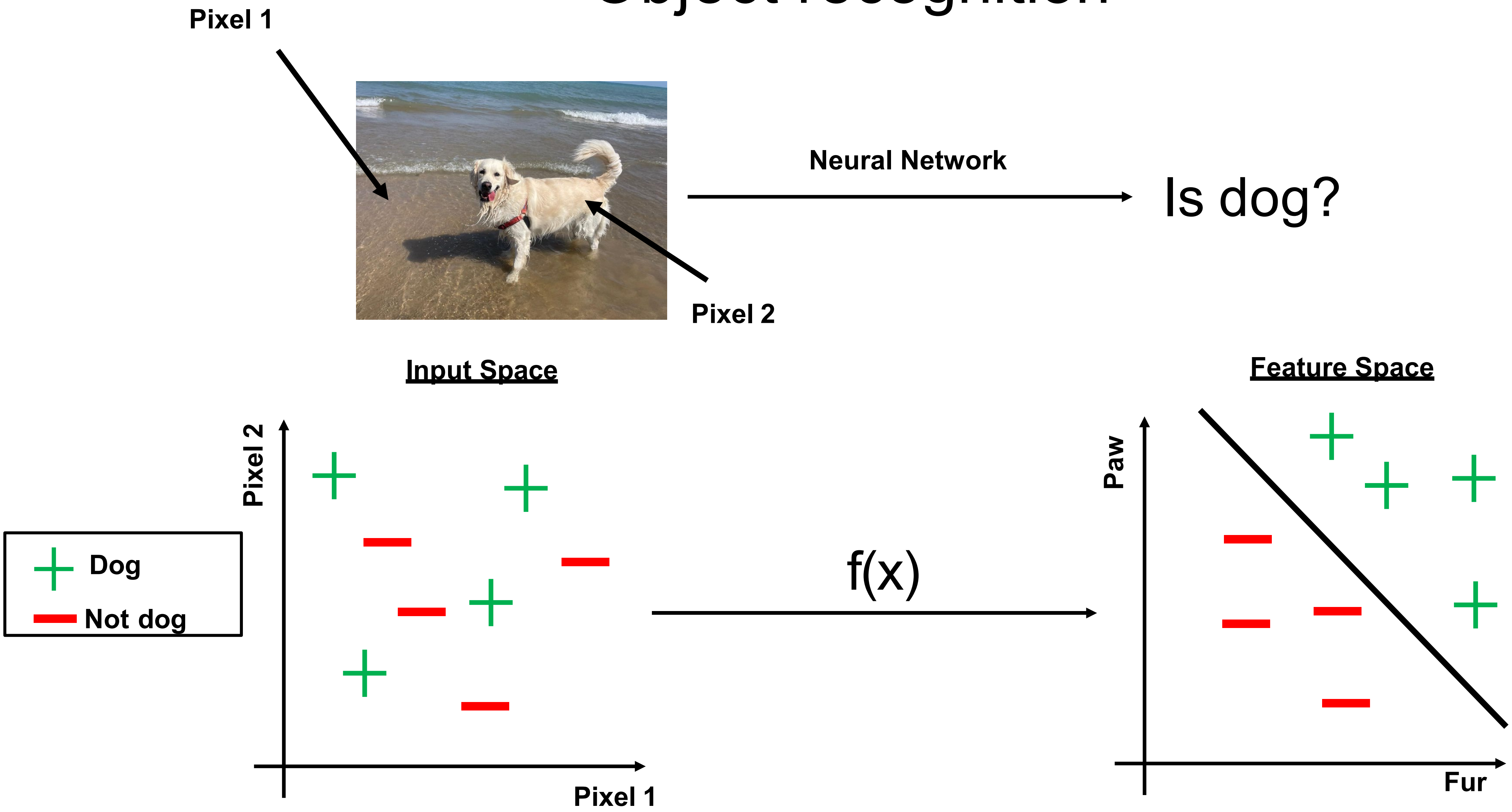


[Serre, 2014]

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

Source: Isola, Torralba, Freeman

Object recognition

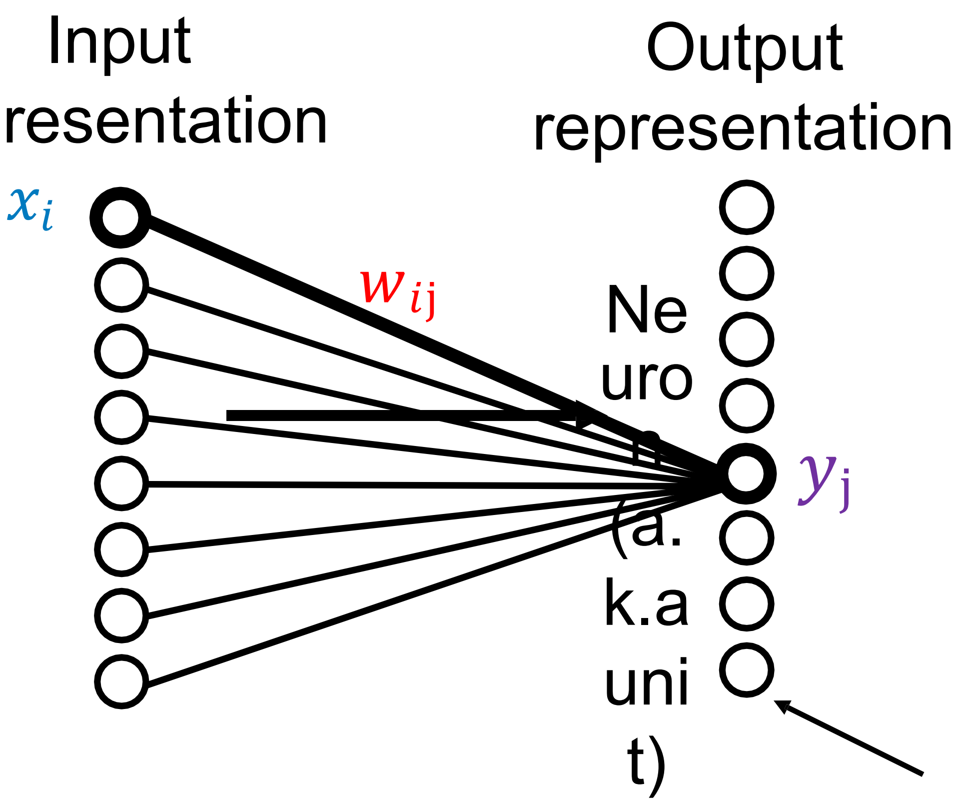


Goal: automatically learn a function that maps data from the input space to a feature space, i.e., "feature learning”, rather than use hand-crafted features

Computation in a neural net

Let's say we have some 1D input that we want to convert to some new feature space:

Linear layer



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

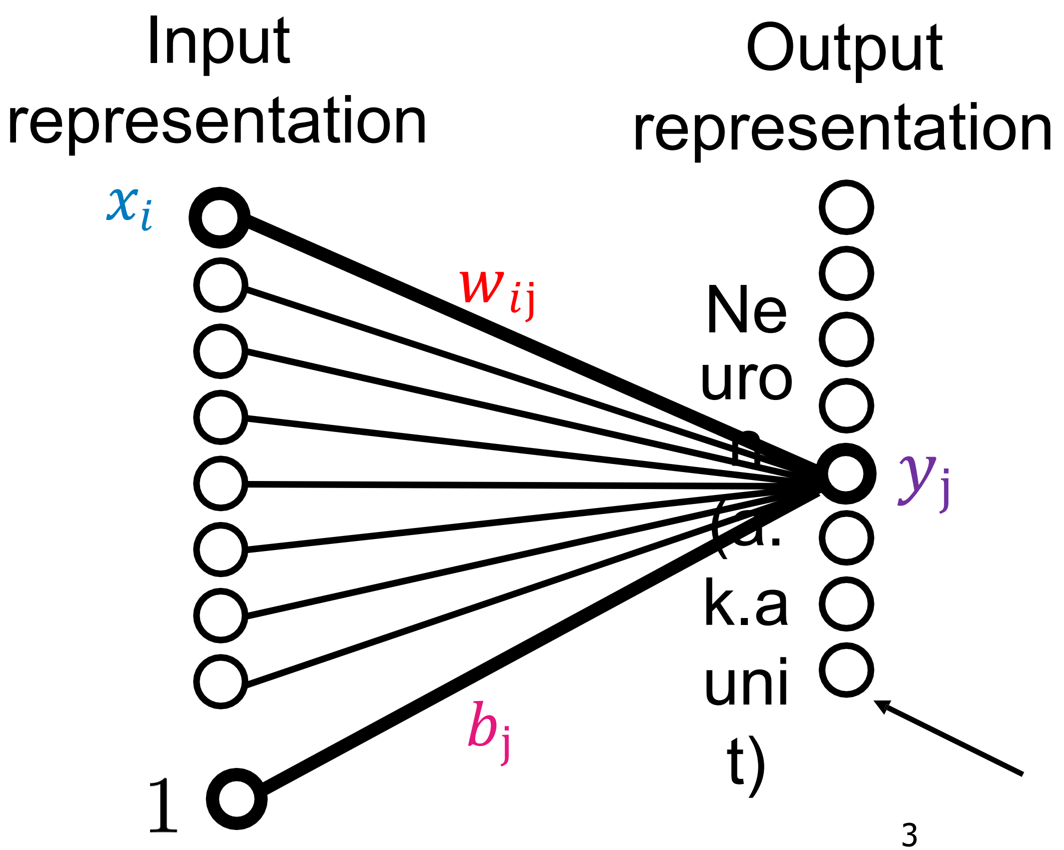
weights

Adapted from: Isola, Torralba, Freeman

Computation in a neural net

Let's say we have some 1D input that we want to convert to some new feature space

Linear layer



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

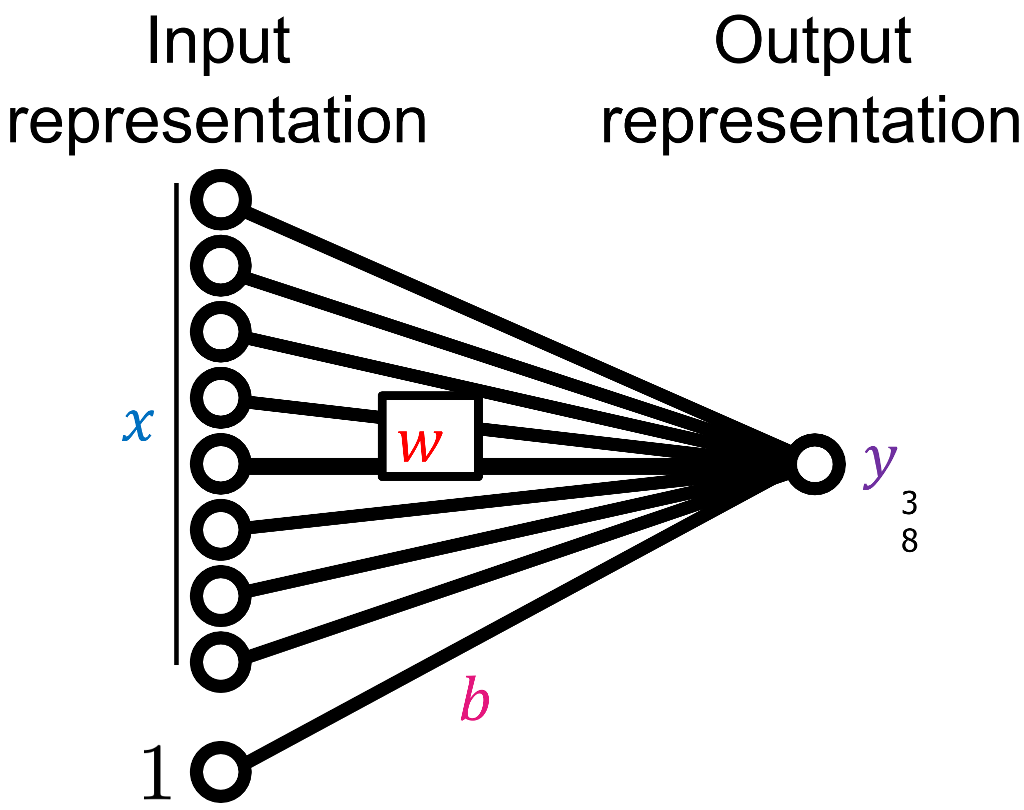
weights

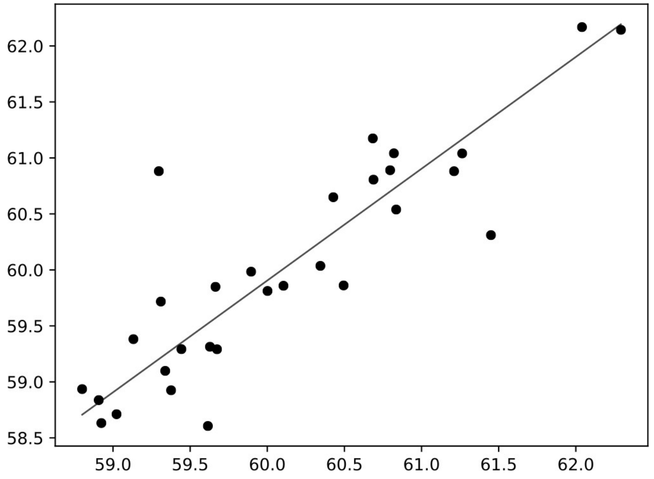
bias

Adapted from: Isola, Torralba, Freeman

Example: Linear Regression

Linear layer



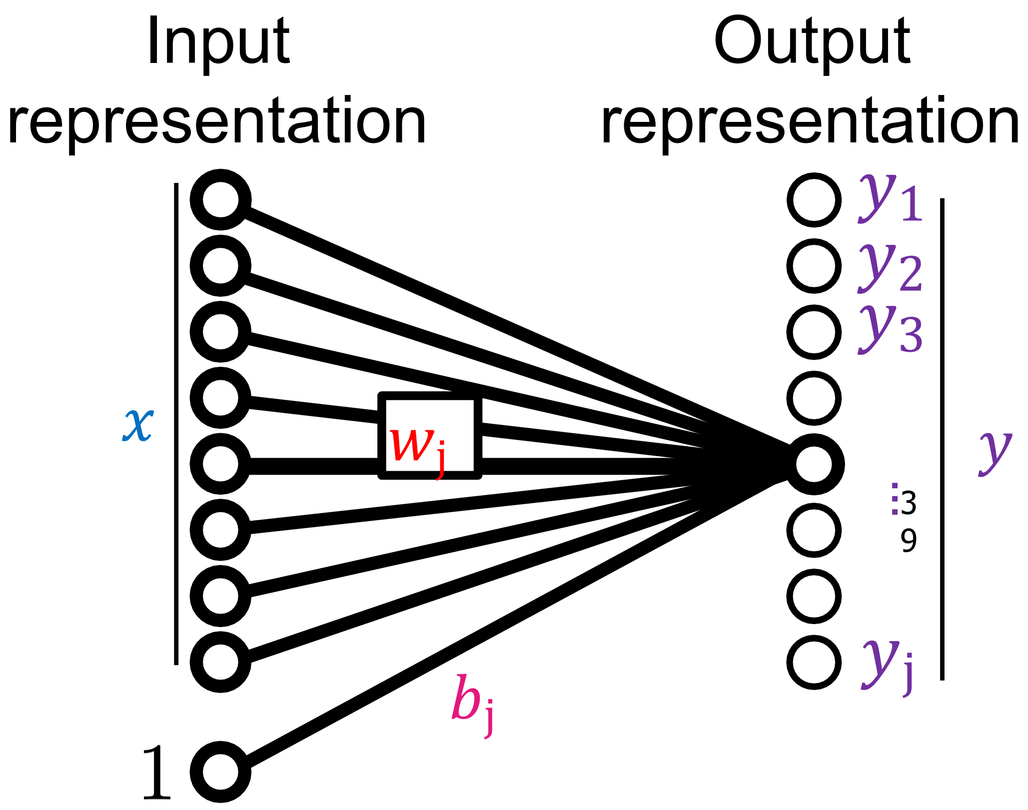


$$f_{\mathbf{w},b}(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + b$$

Adapted from: Isola, Torralba, Freeman

Computation in a neural net – Full Layer

Linear layer



$$y = Wx + b$$

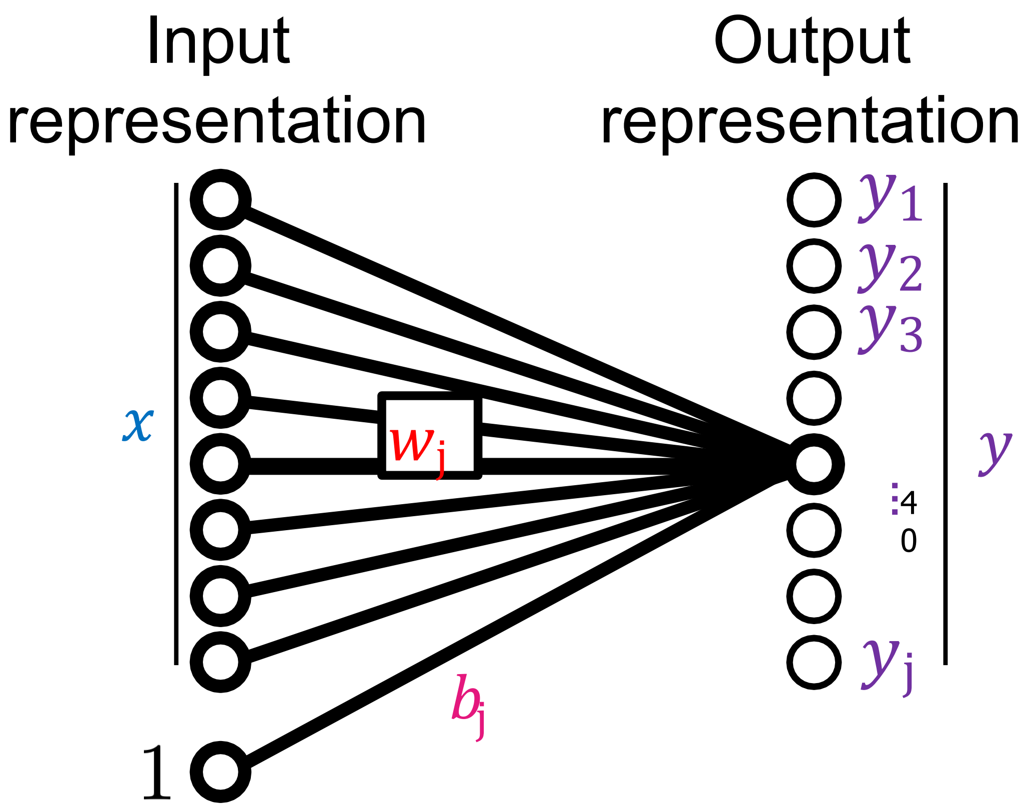
$$\begin{bmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{j1} & \cdots & w_{jn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_j \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_j \end{bmatrix}$$

parameters of the model: $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{b}\}$

Adapted from: Isola, Torralba, Freeman

Computation in a neural net – Full Layer

Linear layer



Full layer

$$y = Wx + b$$

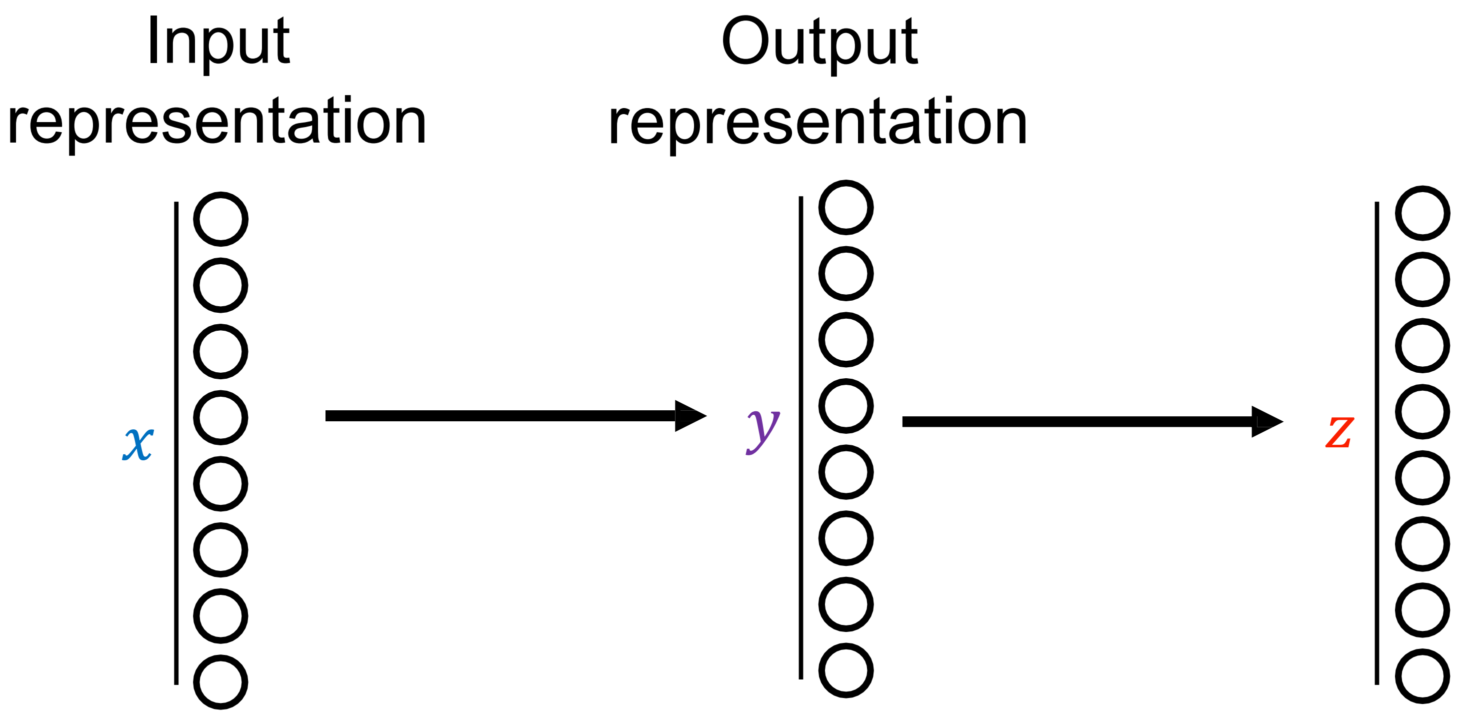
$$\begin{bmatrix} w_{11} & \cdots & w_{jn} & b_1 \\ \vdots & \ddots & \vdots & \vdots \\ w_{j1} & \cdots & w_{jn} & b_j \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \\ 1 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_j \end{bmatrix}$$

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

Adapted from: Isola, Torralba, Freeman

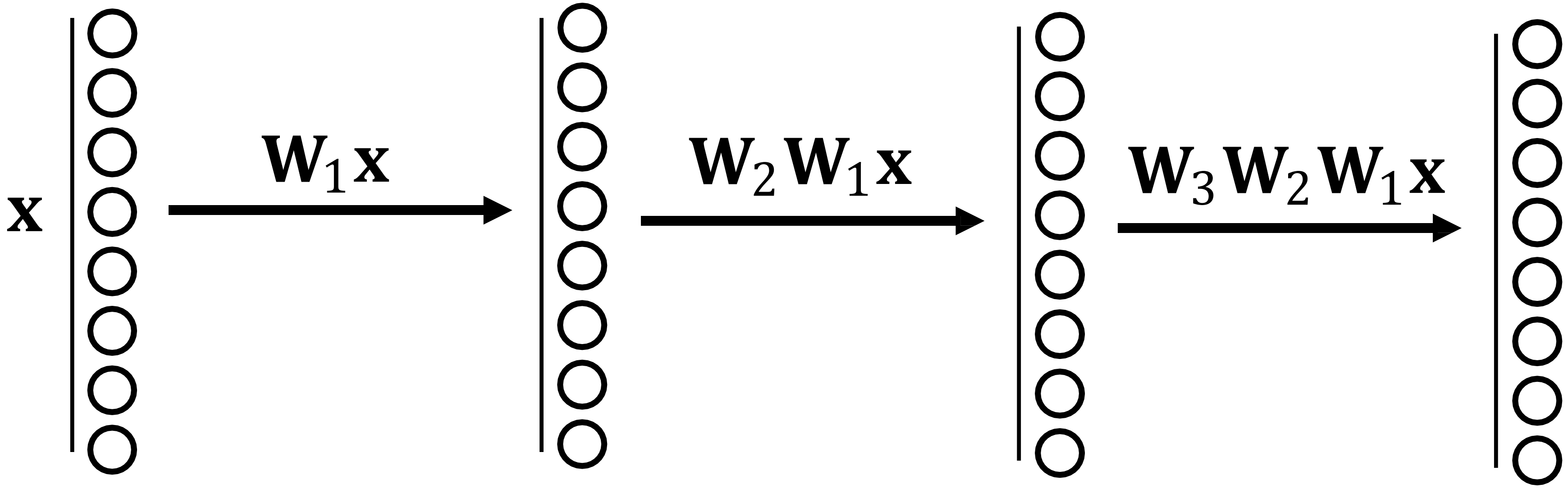
Computation in a neural net – Recap

We can now transform our input representation vector into some output representation vector using a bunch of linear combinations of the input:



We can repeat this as many times as we want!

What is the problem with this idea?



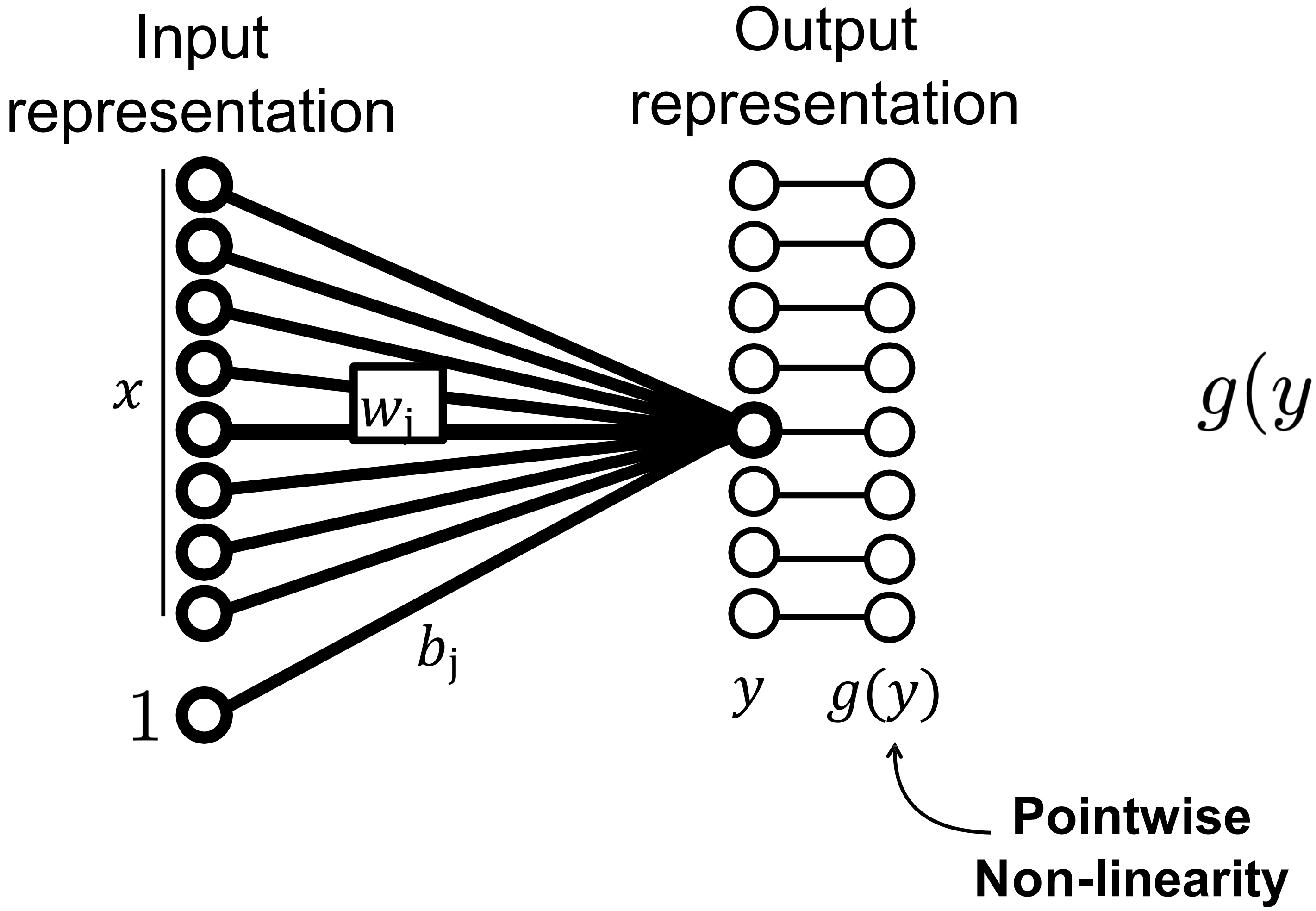
Can be expressed as single linear layer!

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

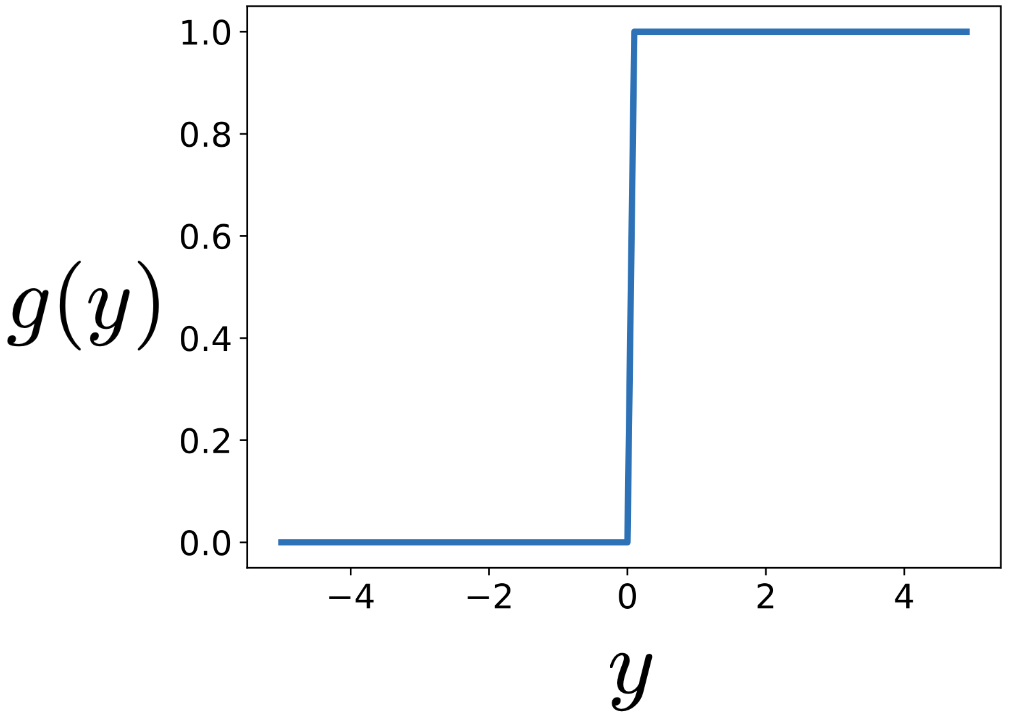
Limited power: can't solve XOR ☹

Solution: simple nonlinearity

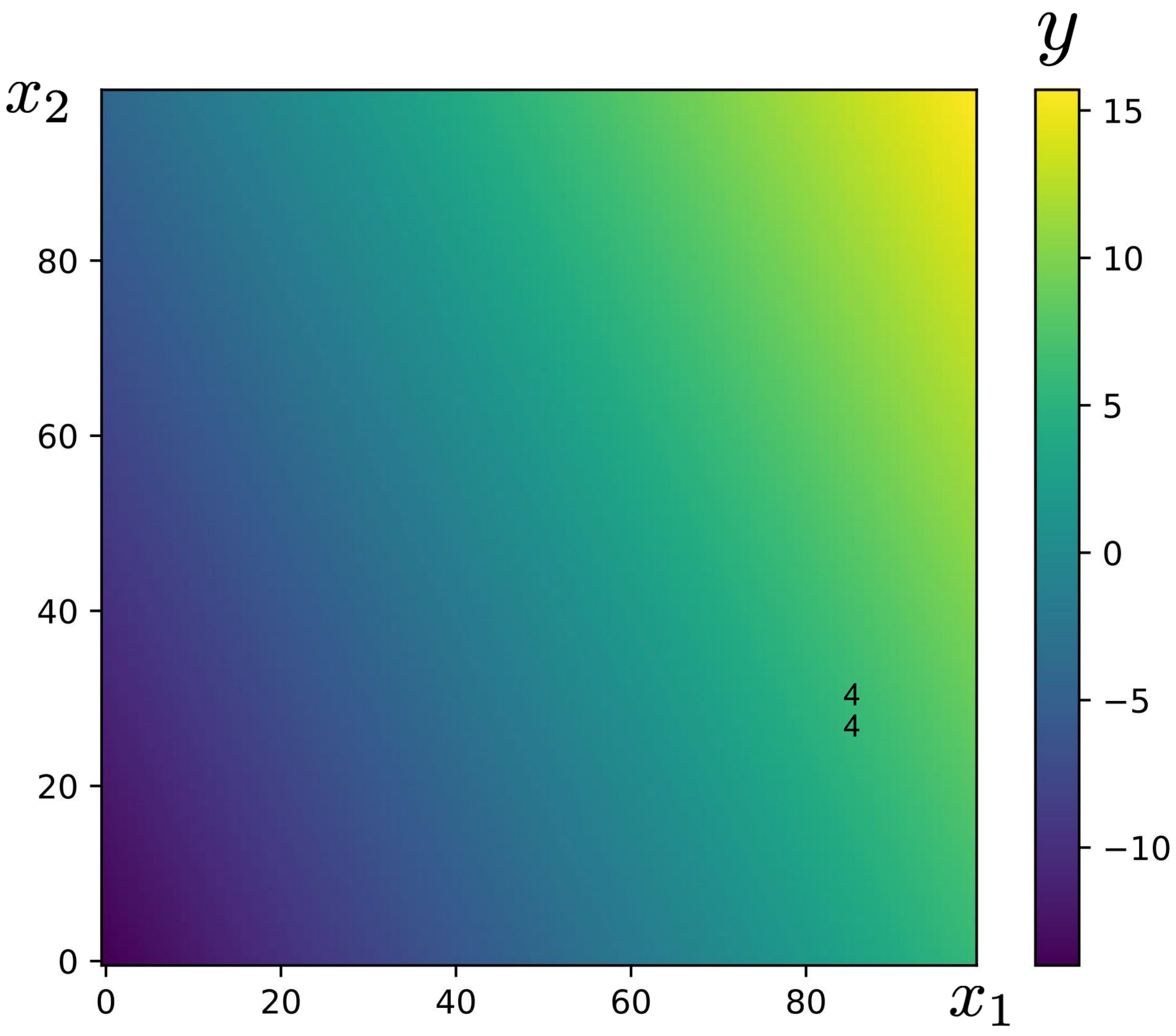
Linear layer



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



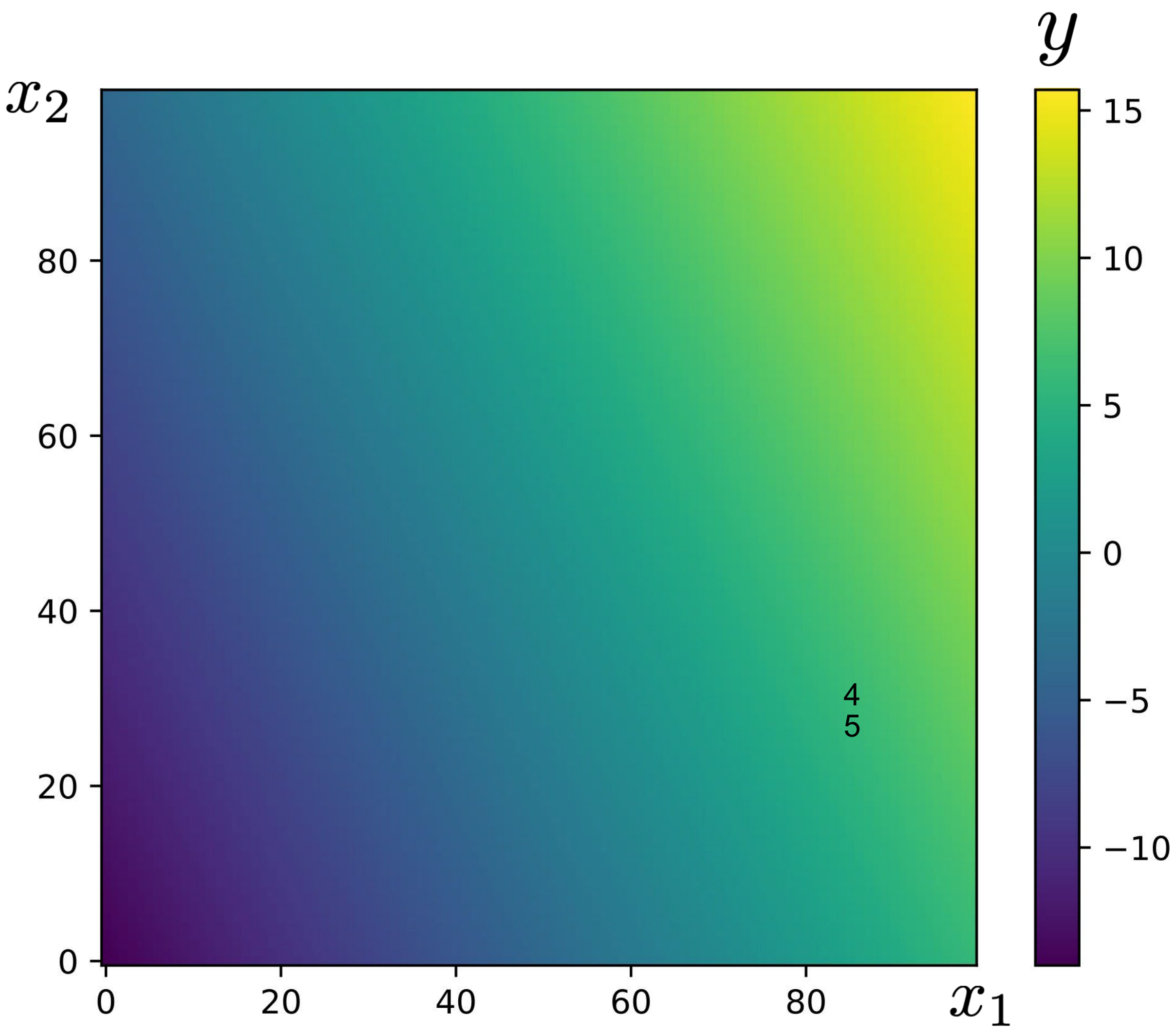
Example: linear classification with a perceptron



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

Source: Isola, Torralba, Freeman

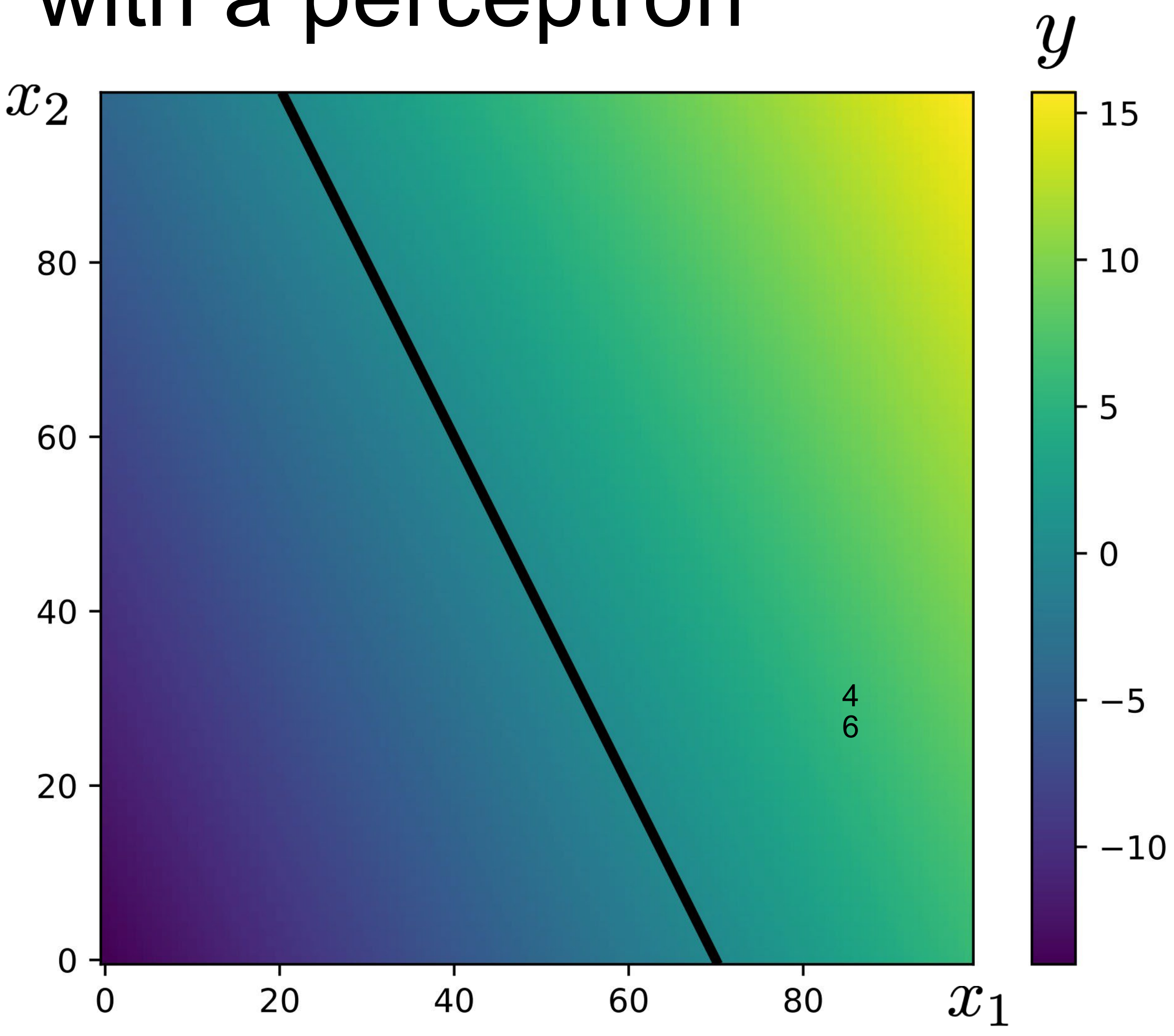
Example: linear classification with a perceptron



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

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

Example: linear classification with a perceptron

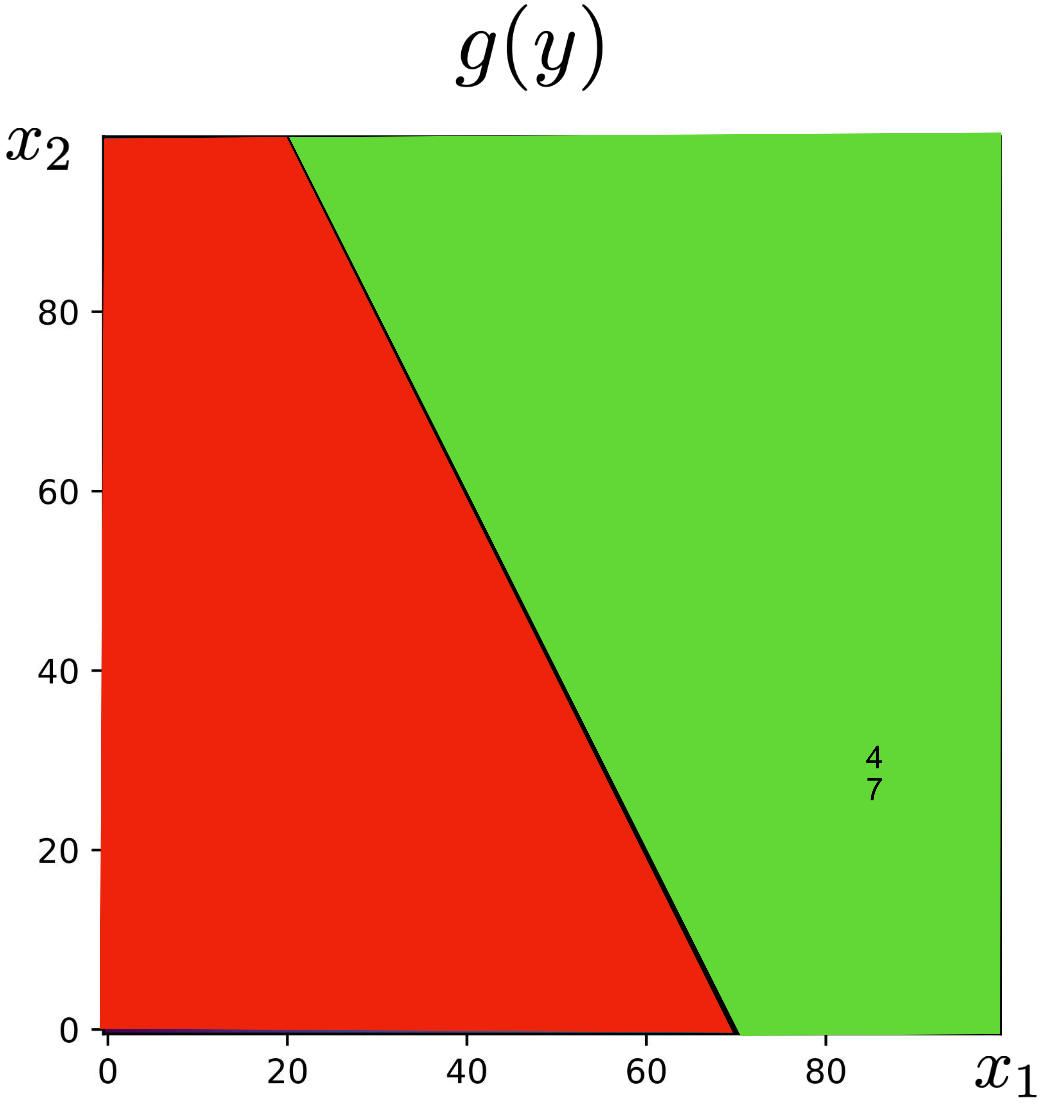


$$y = \mathbf{x}^T \mathbf{w} + b$$

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

"when y is greater than 0, set all pixel values to 1 (green), otherwise, set all pixel values to 0 (red)"

Example: linear classification with a perceptron



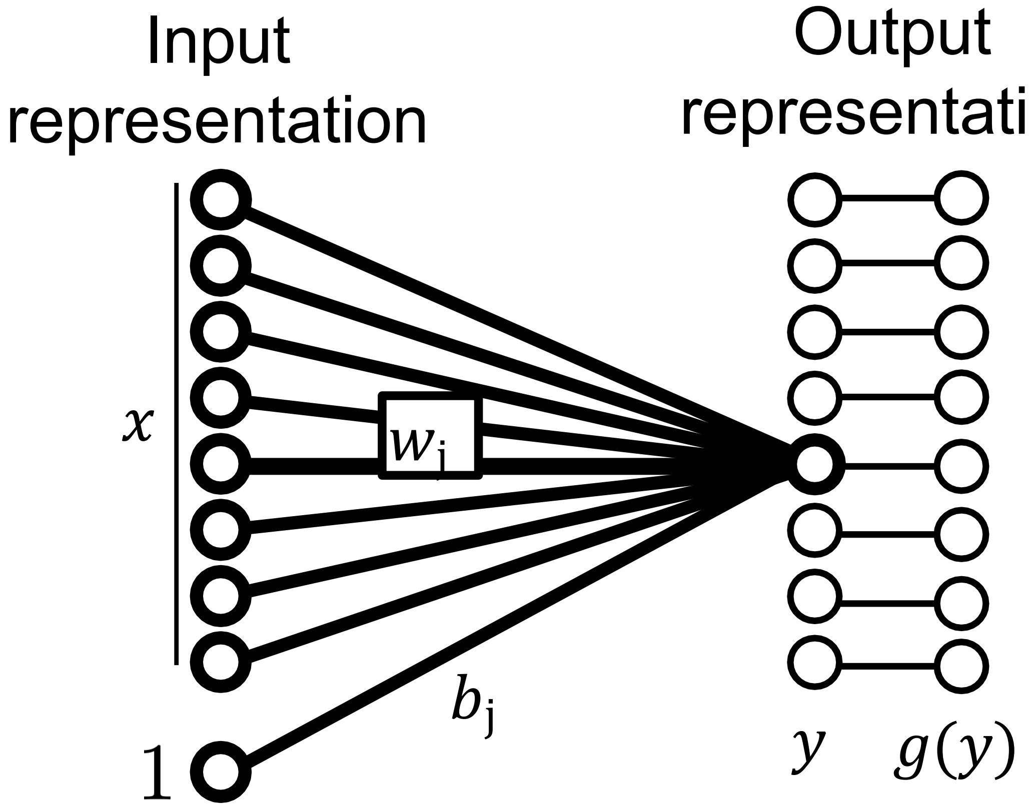
$$y = \mathbf{x}^T \mathbf{w} + b$$

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

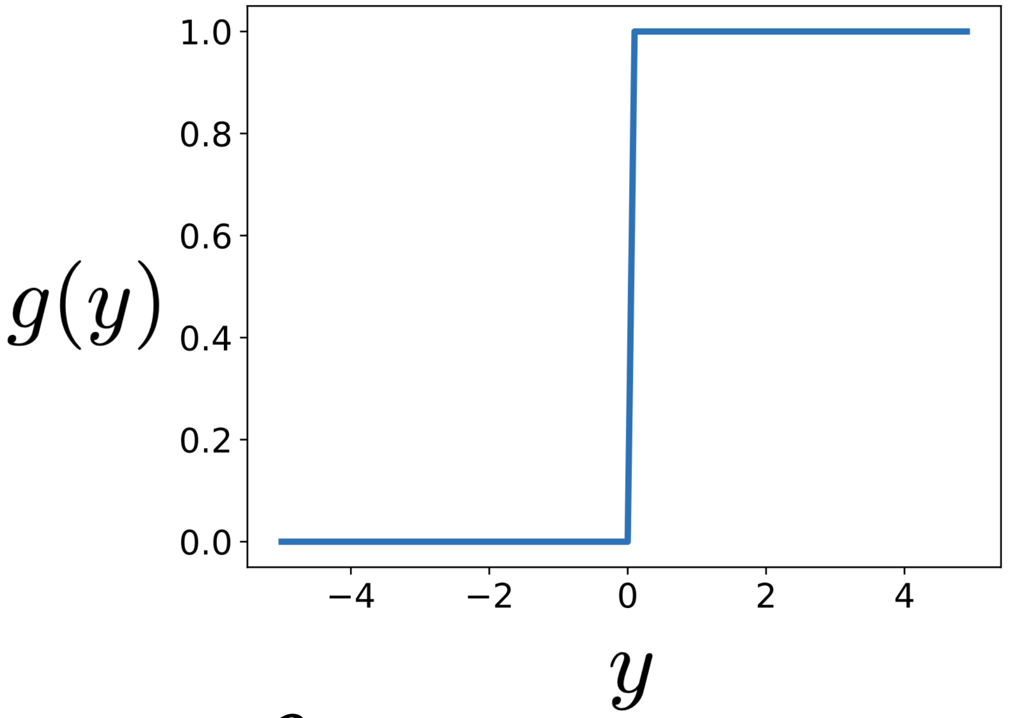
"when y is greater than 0, set all pixel values to 1 (green), otherwise, set all pixel values to 0 (red)"

Computation in a neural net - nonlinearity

Linear layer



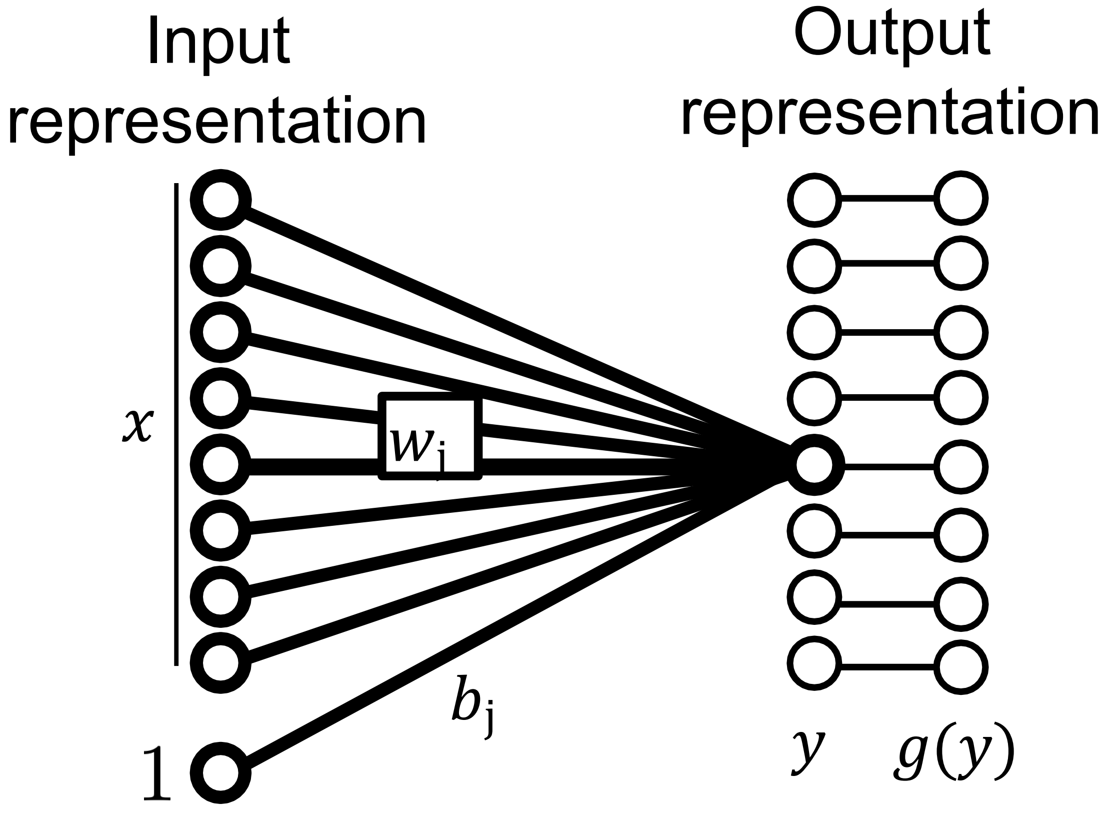
$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$



Can't use with gradient descent, $\frac{\partial}{\partial y} g = 0$

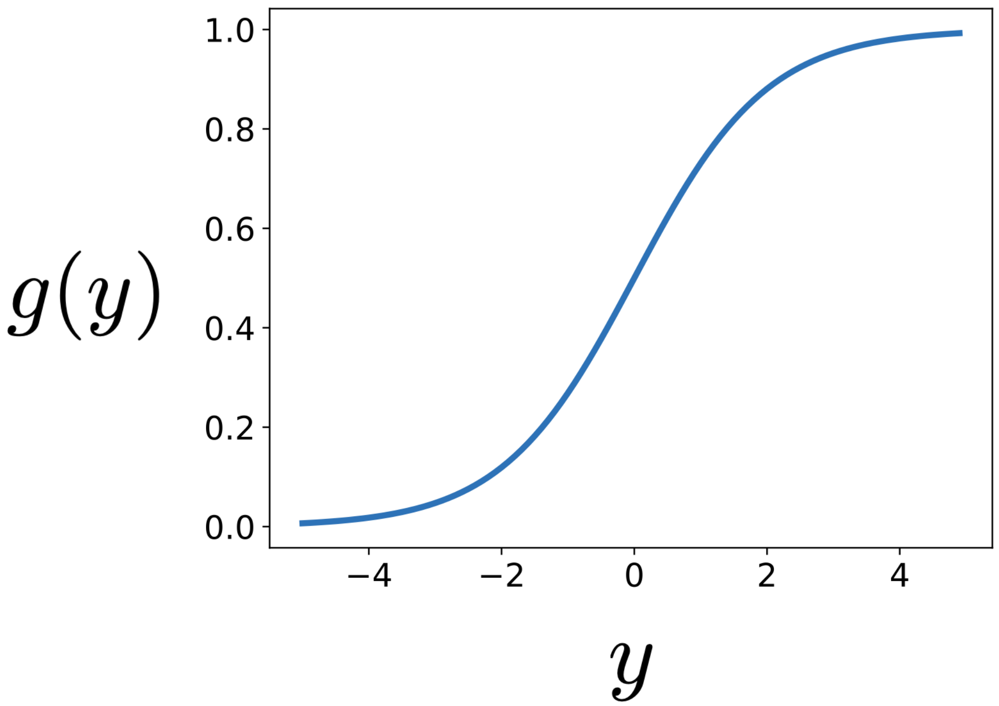
Computation in a neural net - nonlinearity

Linear layer



Sigmoid

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

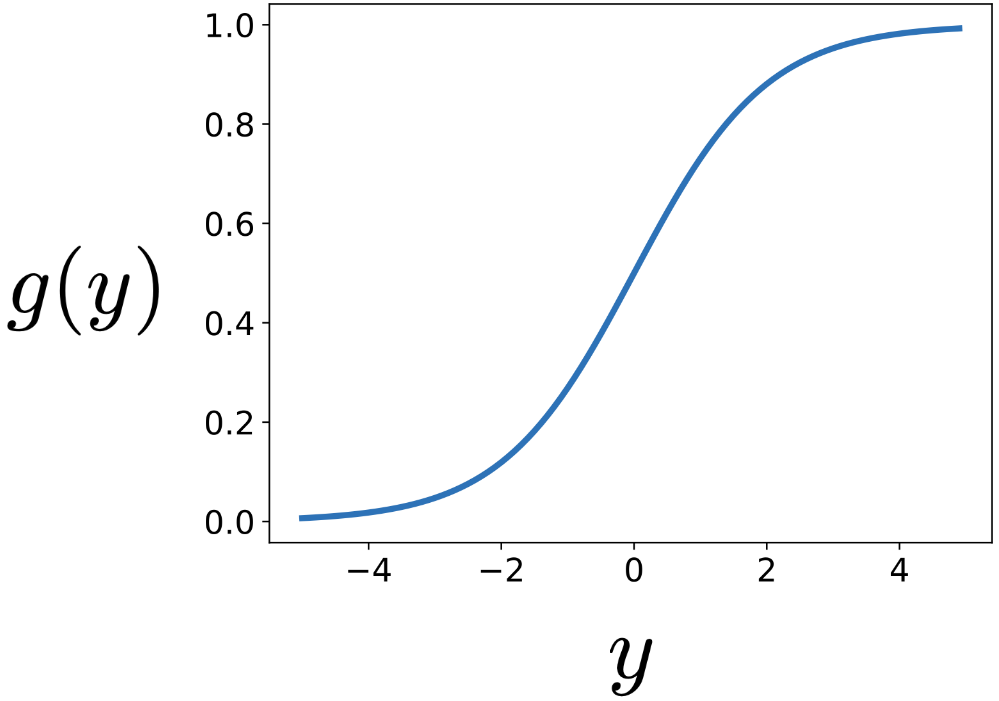


Computation in a neural net - nonlinearity

- Bounded between [0,1]
- Saturation for large +/- inputs
- Gradients go to zero

Sigmoid

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

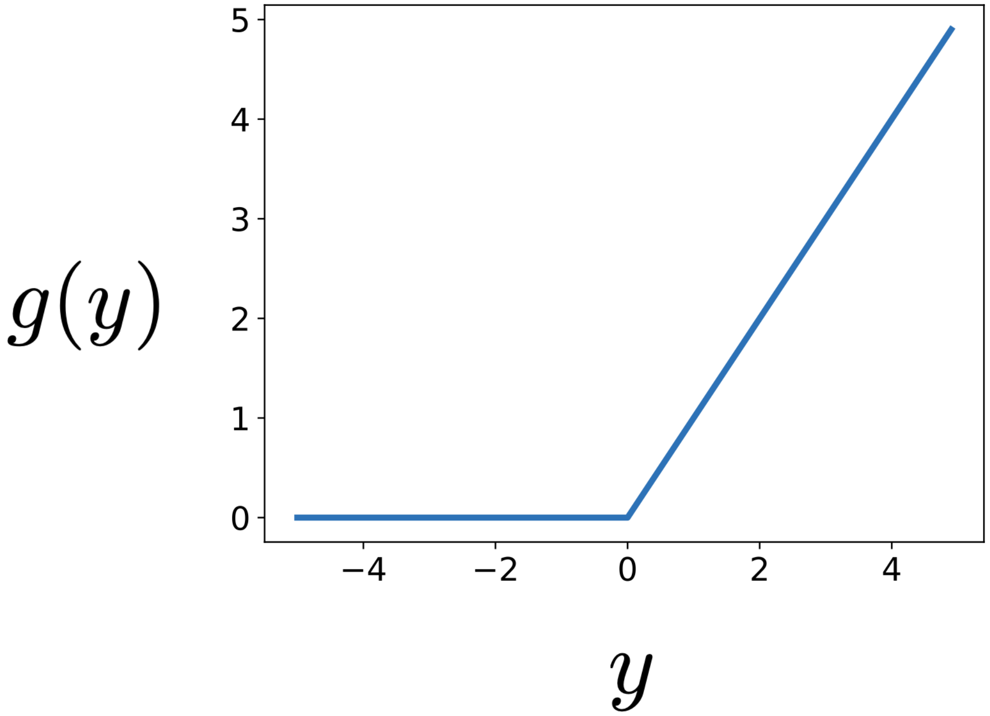


Computation in a neural net — nonlinearity

- 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 (6x speedup vs. tanh in [Krizhevsky et al. 2012])
- 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 α is small (e.g., 0.02)

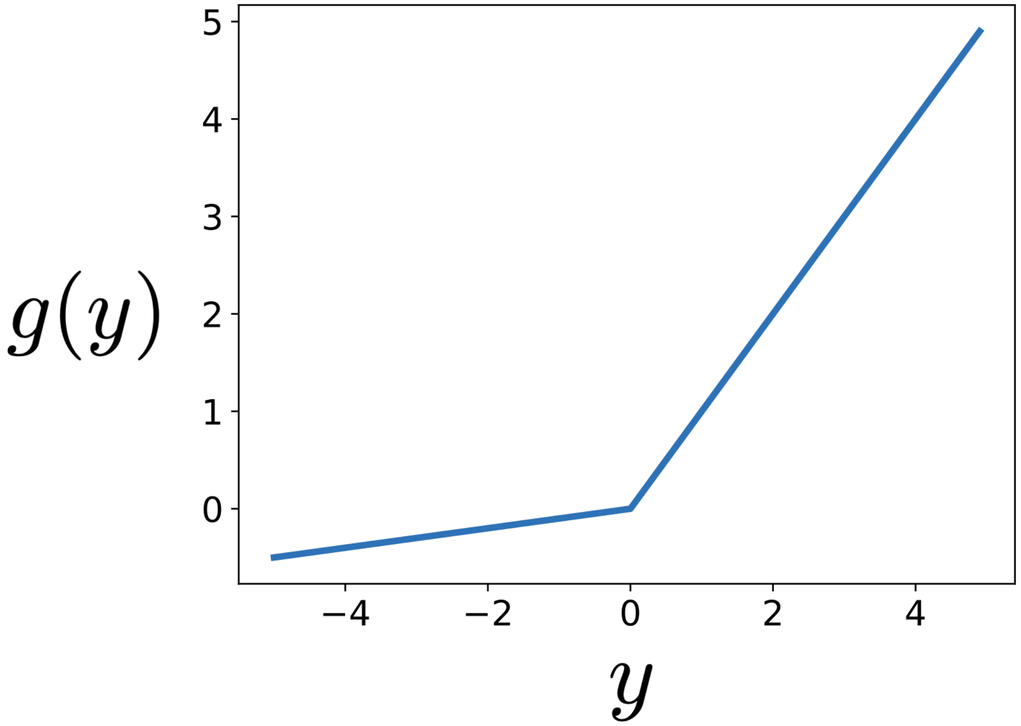
- Efficient to implement:

- Has non-zero gradients everywhere (unlike ReLU)

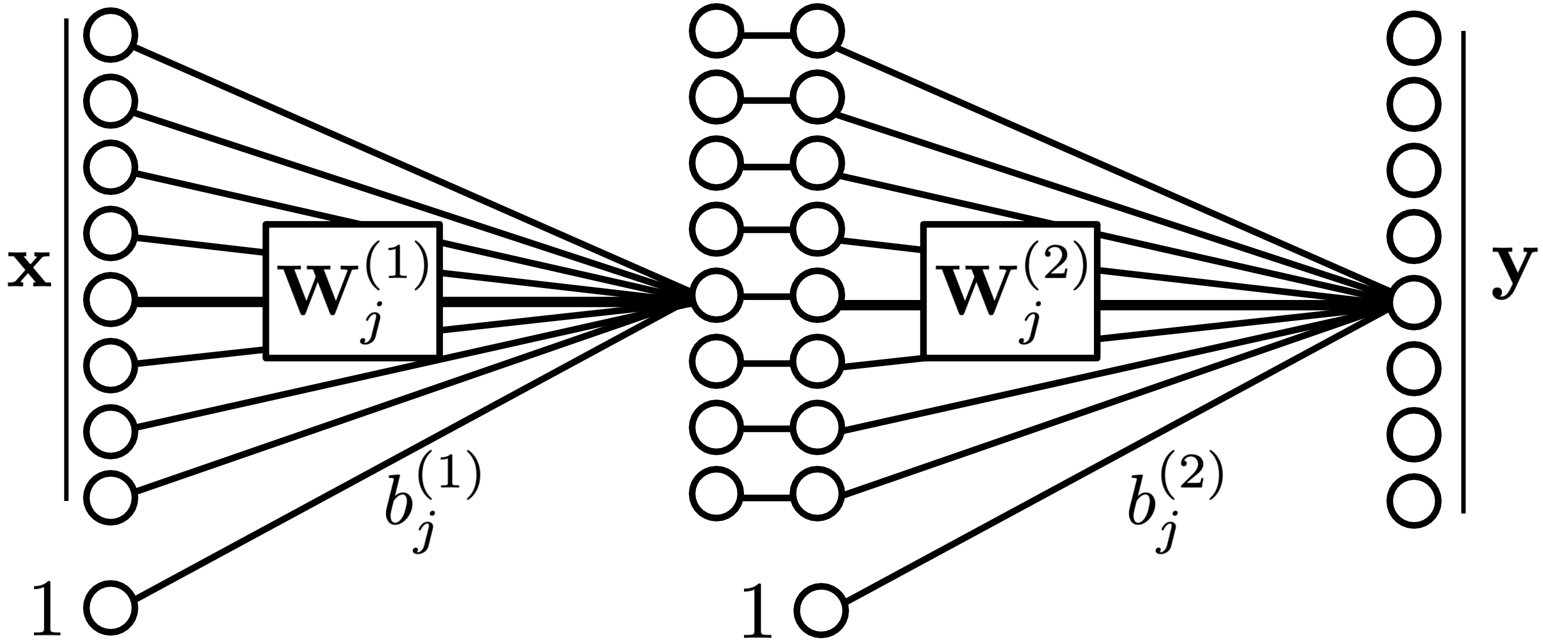
$$\frac{\partial g}{\partial y} = \begin{cases} -a, & \text{if} \quad y < 0 \\ 1, & \text{if} \quad y \geq 0 \end{cases}$$

Leaky ReLU

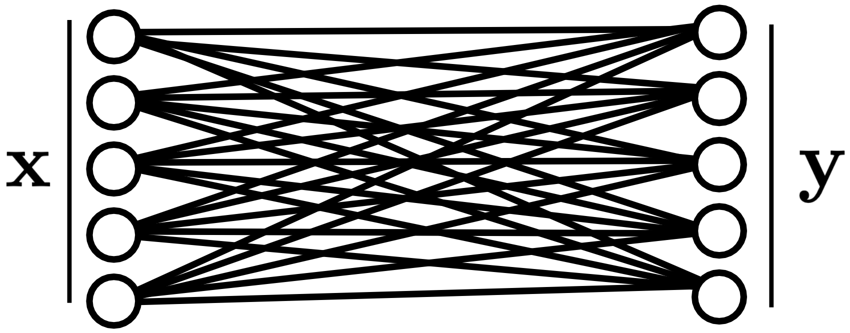
$$g(y) = \begin{cases} \max(0, y), & \text{if} \quad y \geq 0 \\ a\min(0, y), & \text{if} \quad y < 0 \end{cases}$$



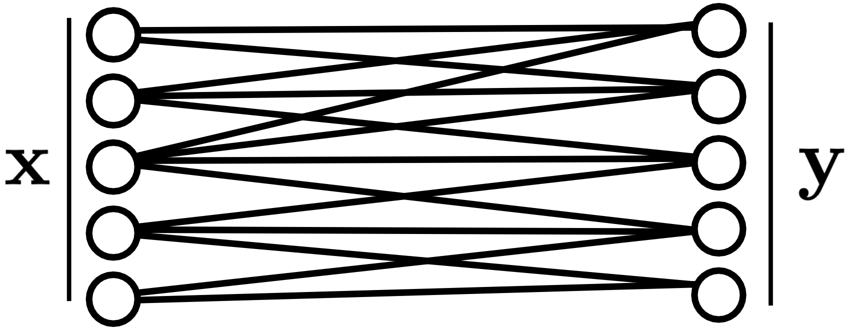
Stacking layers



Connectivity patterns

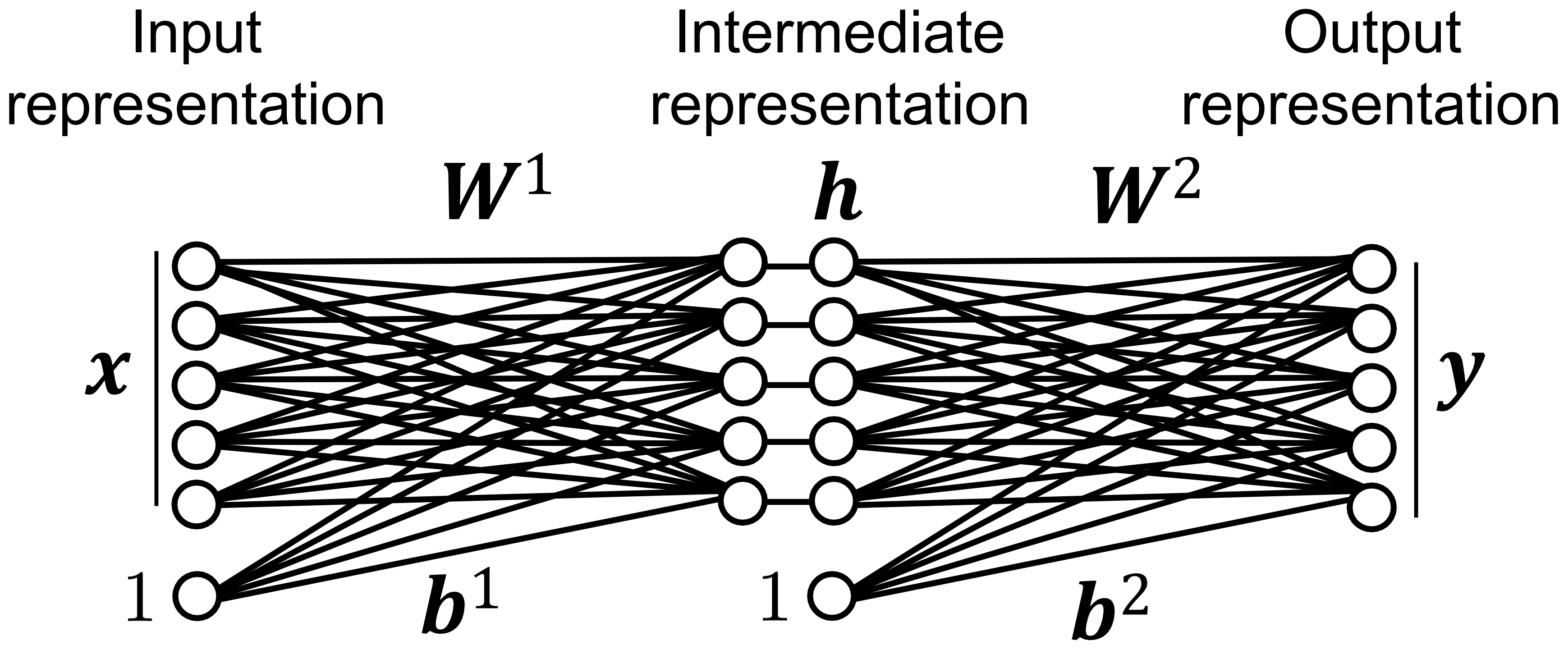


Fully connected layer



Locally connected layer
(Sparse W)

Stacking layers



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

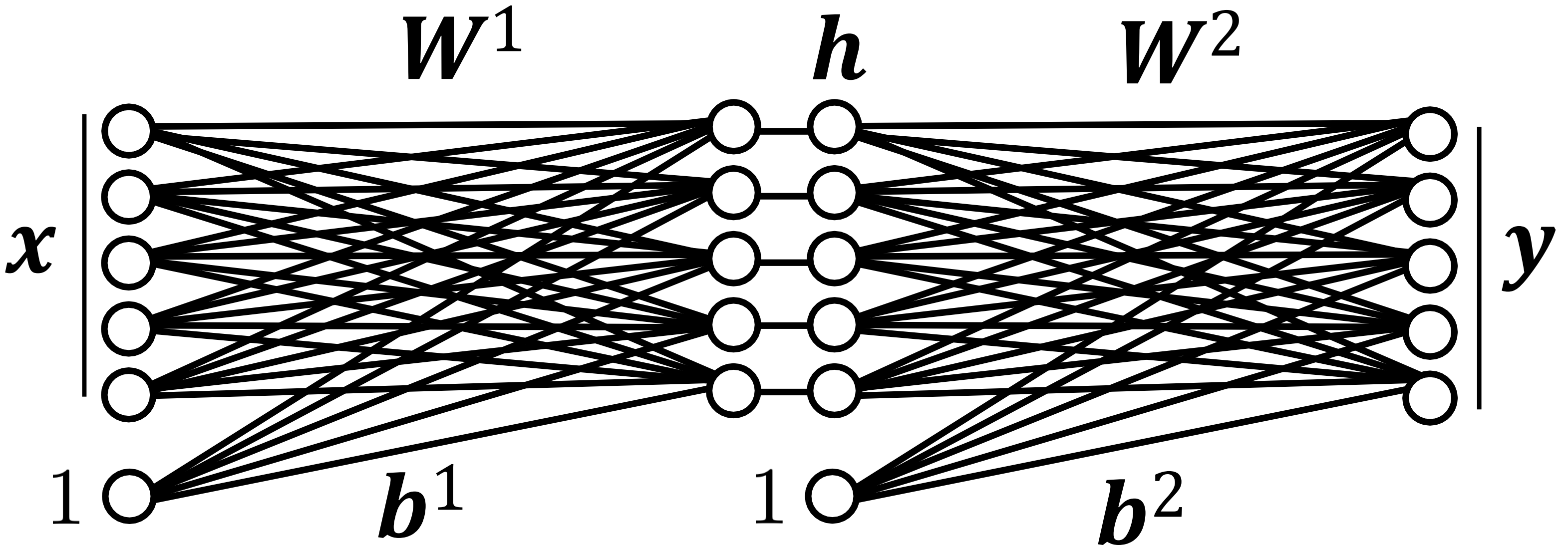
ReLU

$$\theta = \{\boldsymbol{W}^1, \dots, \boldsymbol{W}^L, \boldsymbol{b}^1, \dots, \boldsymbol{b}^L\}$$

Source: Isola, Torralba,

Stacking layers

Input representation | Intermediate representation | Output representation



positive

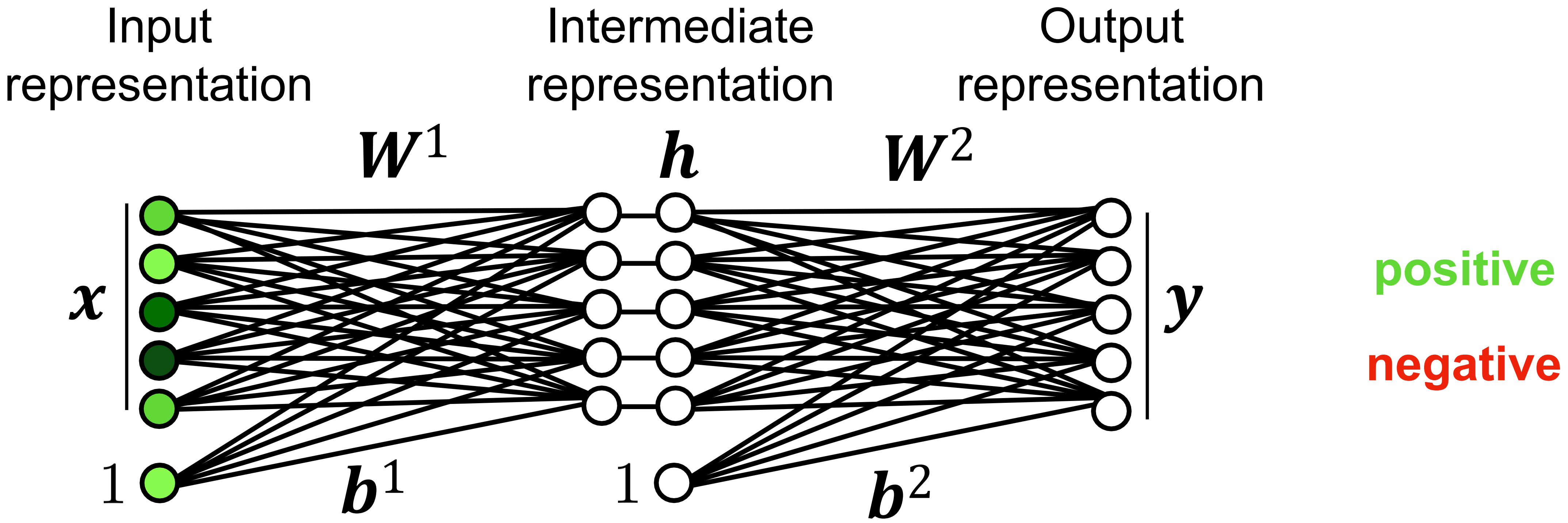
negative

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

ReLU

$$\theta = \{\boldsymbol{W}^1, \dots, \boldsymbol{W}^L, \boldsymbol{b}^1, \dots, \boldsymbol{b}^L\}$$

Stacking layers



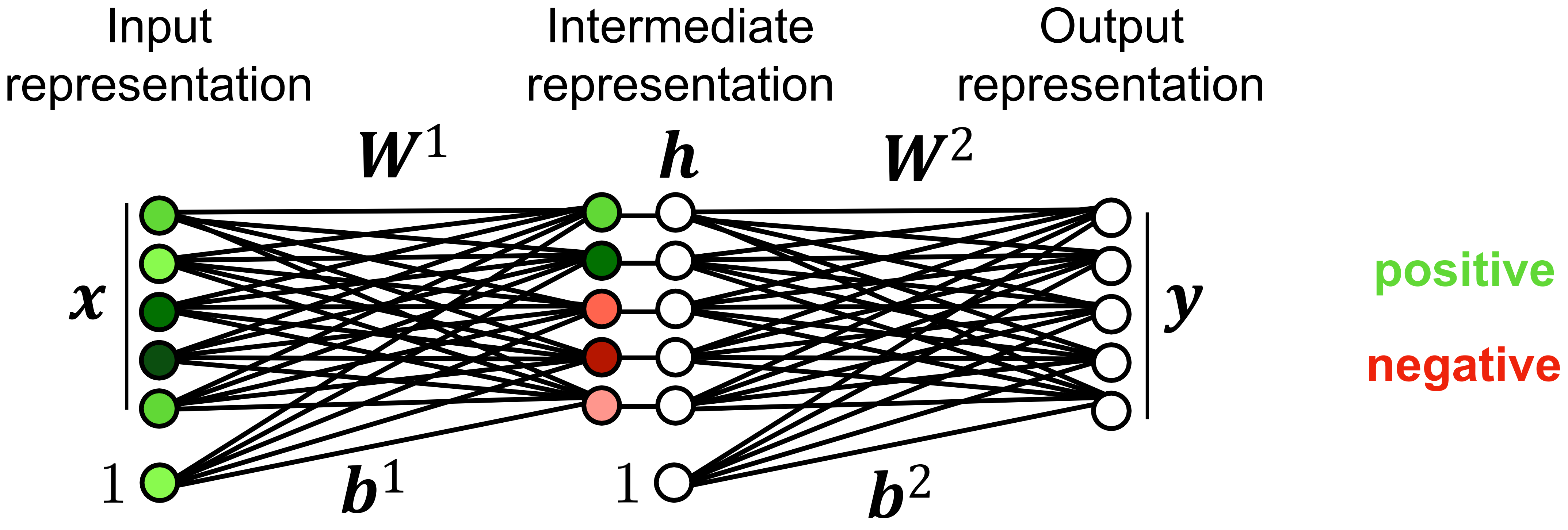
$$\boldsymbol{h} = g(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1) \quad \boldsymbol{y} = g(\boldsymbol{W}^2\boldsymbol{h} + \boldsymbol{b}^2)$$

ReLU

$$\theta = \{\boldsymbol{W}^1, \dots, \boldsymbol{W}^L, \boldsymbol{b}^1, \dots, \boldsymbol{b}^L\}$$

Source: Isola, Torralba, Freeman

Stacking layers

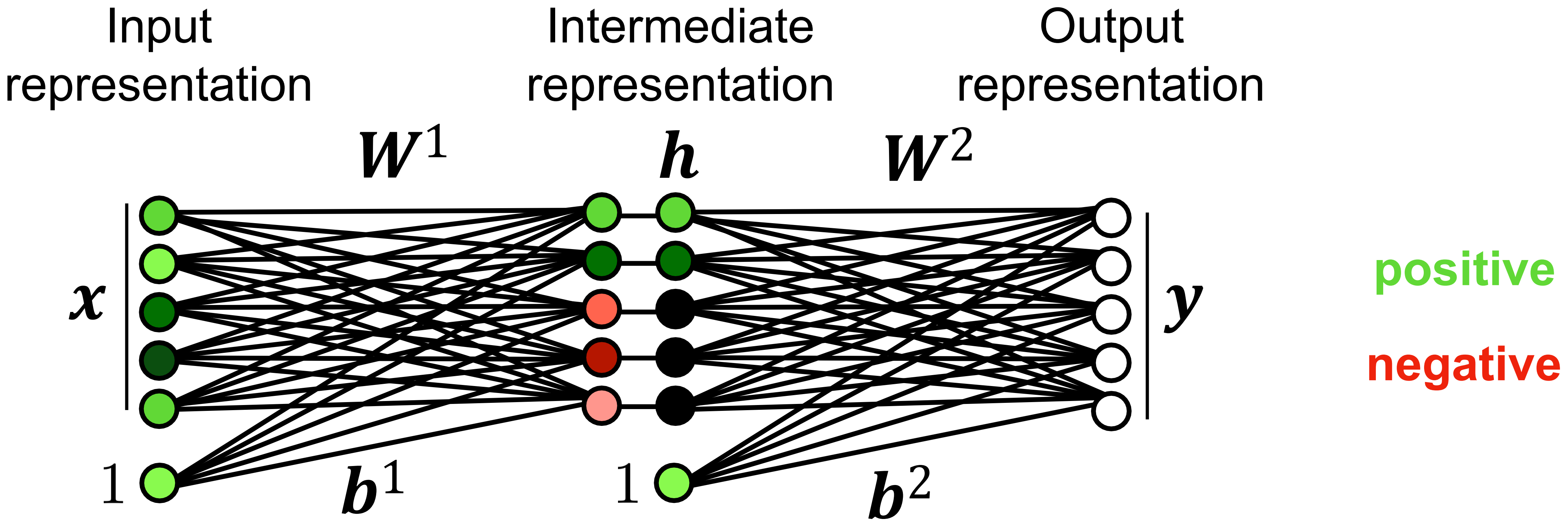


$$\boldsymbol{h} = g(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1) \quad \boldsymbol{y} = g(\boldsymbol{W}^2\boldsymbol{h} + \boldsymbol{b}^2)$$

ReLU

$$\theta = \{\boldsymbol{W}^1, \dots, \boldsymbol{W}^L, \boldsymbol{b}^1, \dots, \boldsymbol{b}^L\}$$

Stacking layers

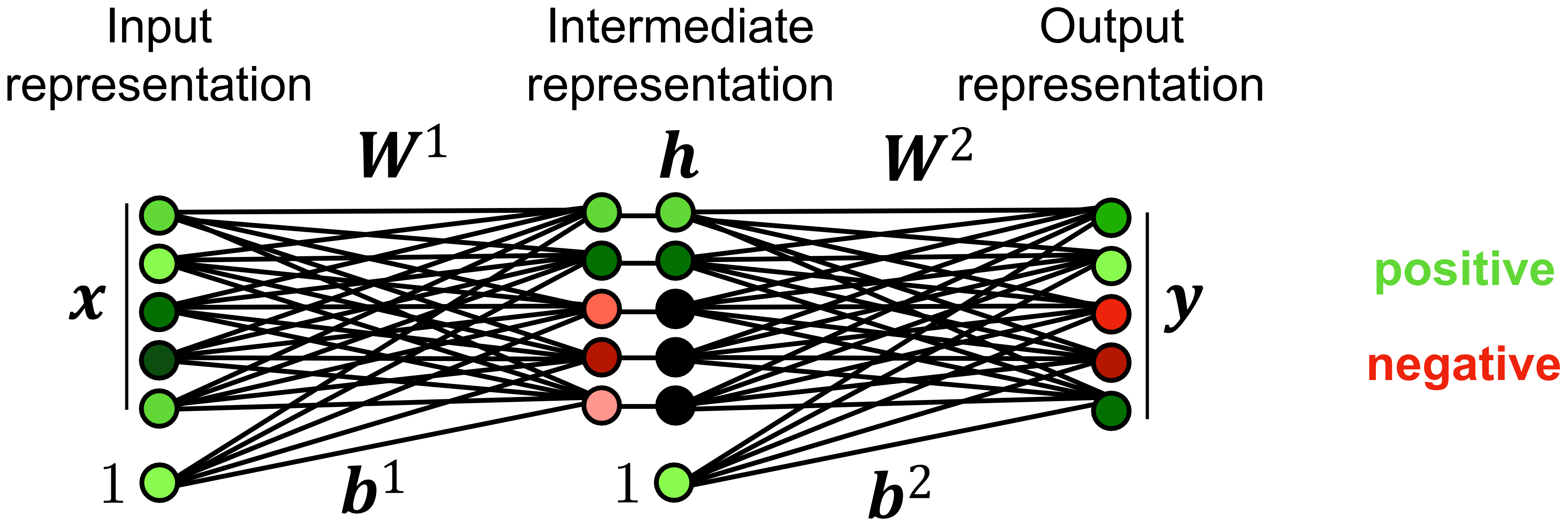


$$\boldsymbol{h} = g(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1) \quad \boldsymbol{y} = g(\boldsymbol{W}^2\boldsymbol{h} + \boldsymbol{b}^2)$$

ReLU

$$\theta = \{\boldsymbol{W}^1, \dots, \boldsymbol{W}^L, \boldsymbol{b}^1, \dots, \boldsymbol{b}^L\}$$

Stacking layers



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

ReLU

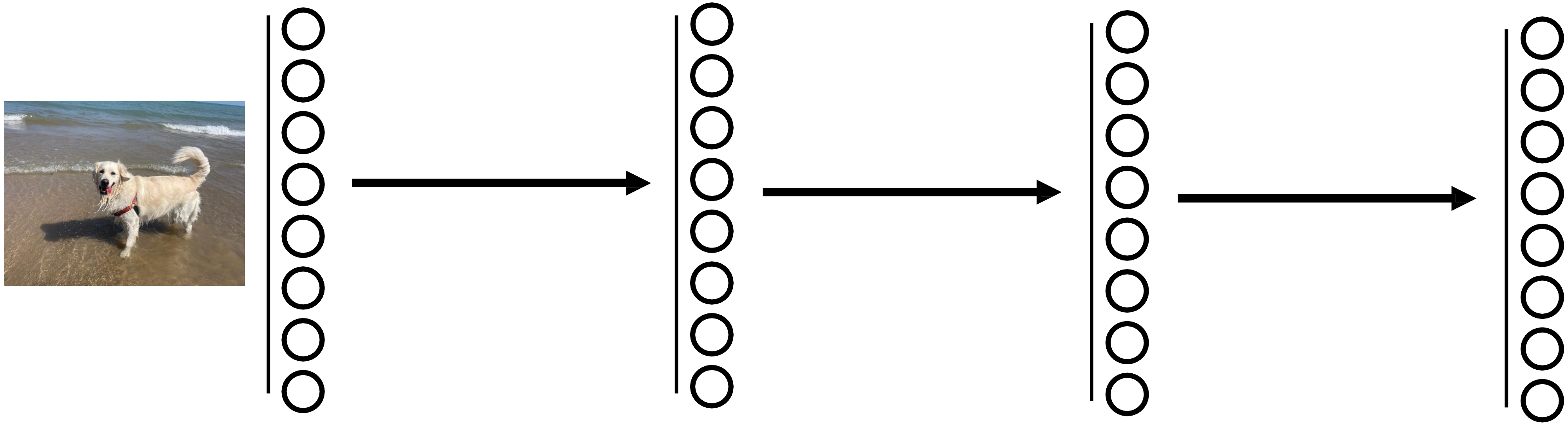
$$\theta = \{\boldsymbol{W}^1, \ldots, \boldsymbol{W}^L, \boldsymbol{b}^1, \ldots, \boldsymbol{b}^L\}$$

Stacking layers - What's actually happening?

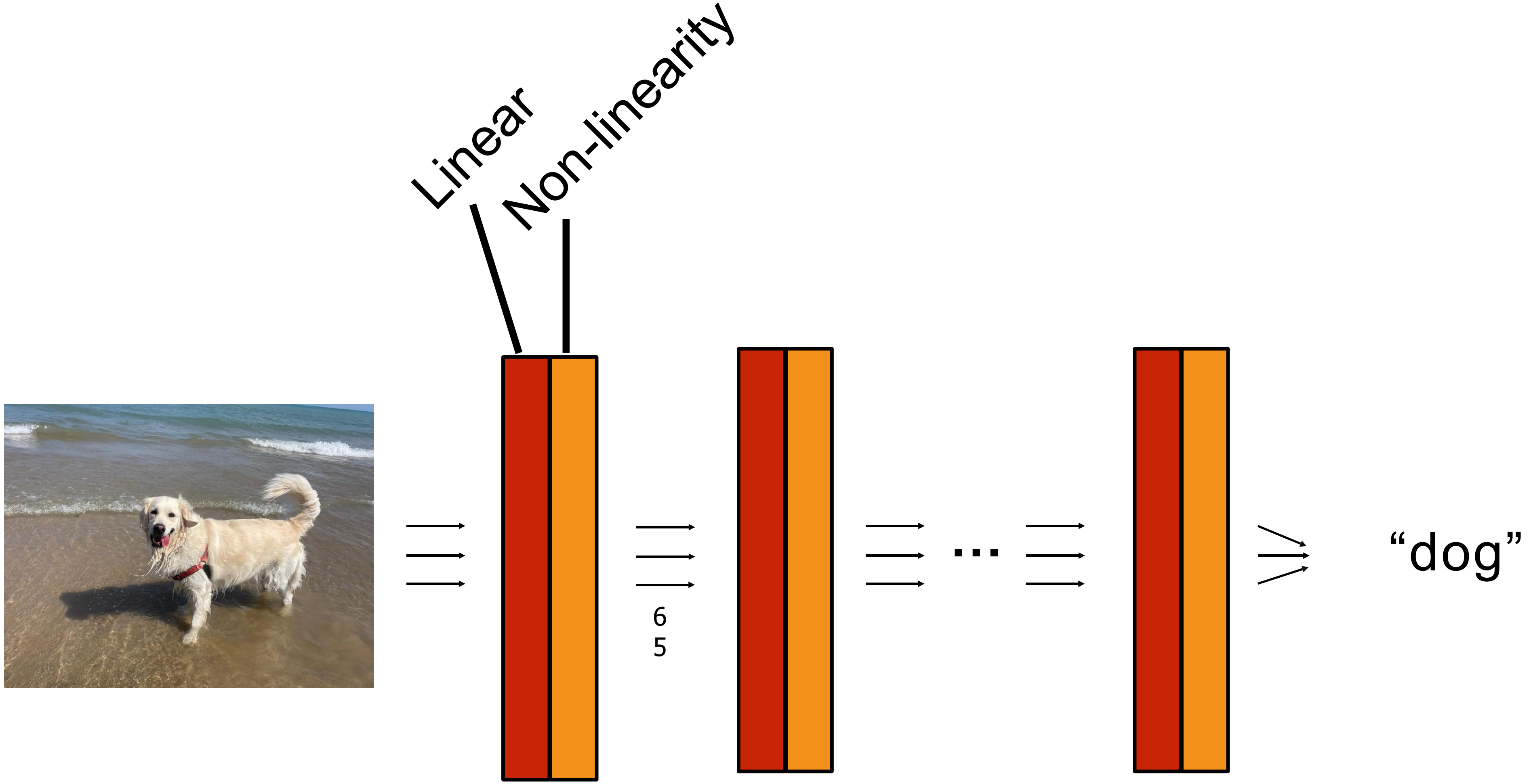
Low level features:
e.g., edge, texture, …

higher level features:
e.g., shape

even higher level features:
e.g., "paw", "fur"



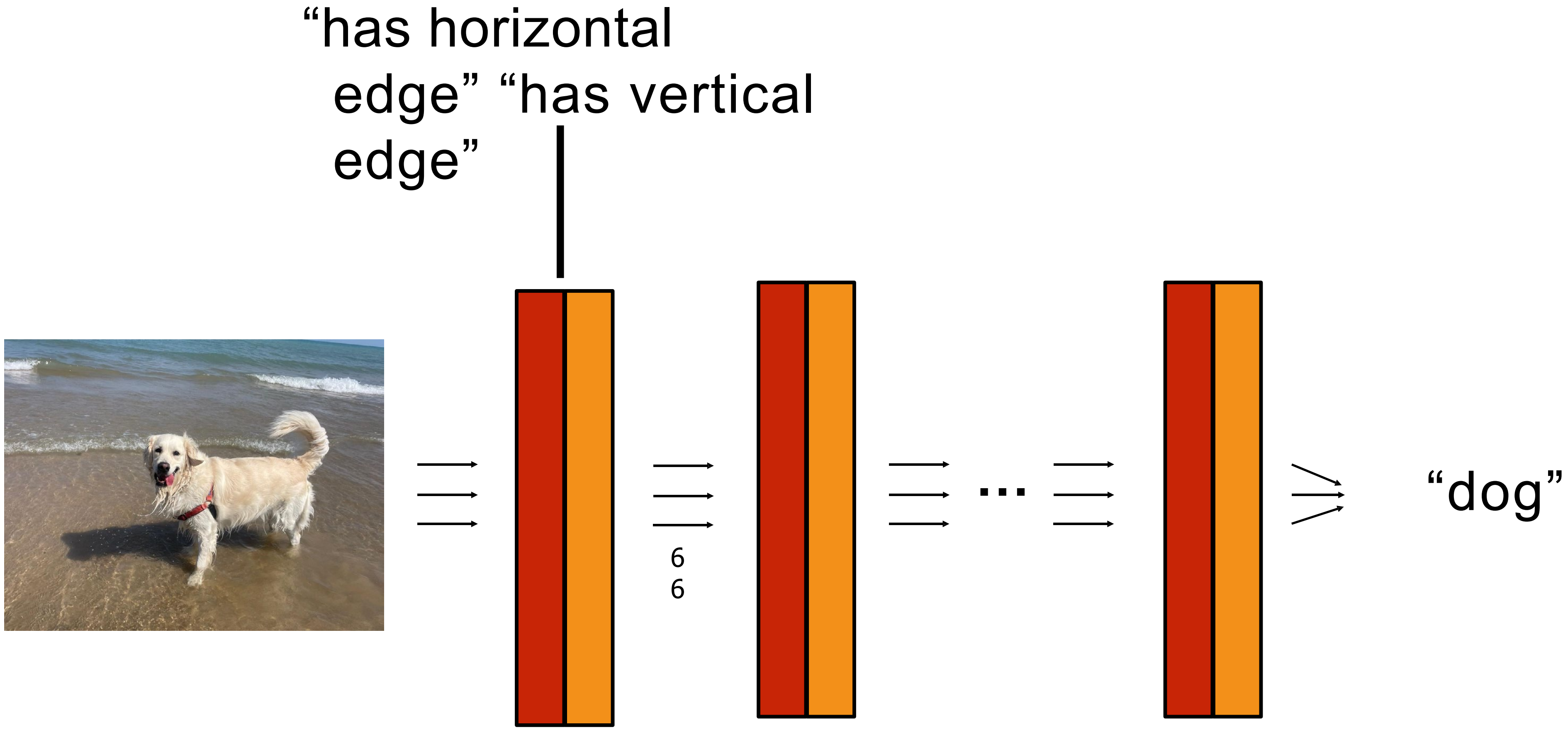
Deep nets



$$f(x) = f_L(\ldots \quad (\quad)$$
$$f_3(f_2(f_1(x)))$$

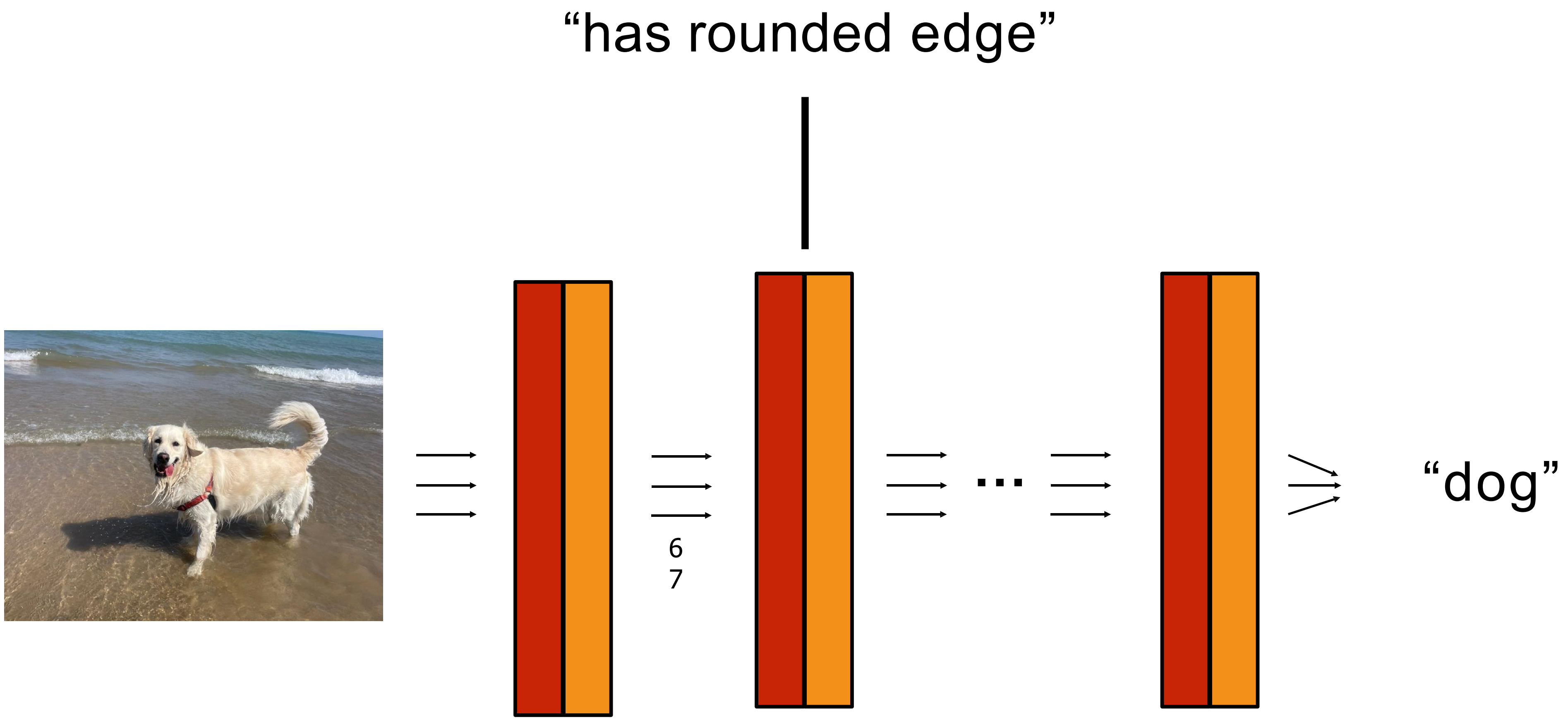
Source: Isola, Torralba,

Deep nets - Intuition



Source: Isola, Torralba,

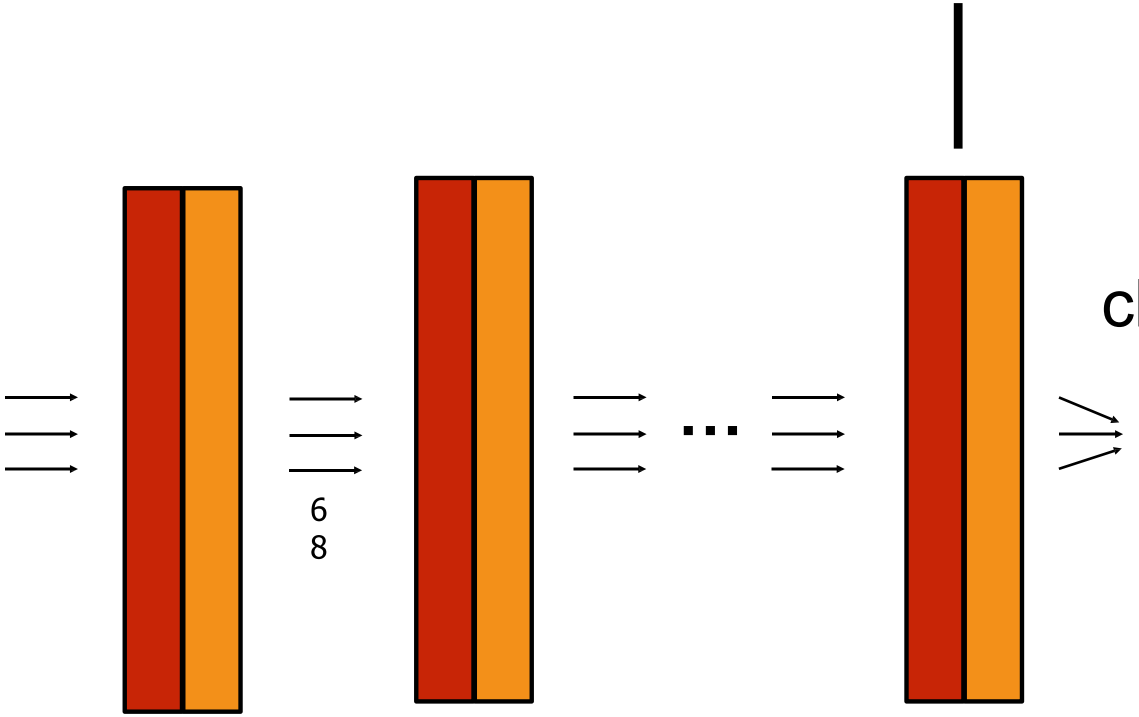
Deep nets - Intuition



Source: Isola, Torralba, Freeman

Deep nets - Intuition

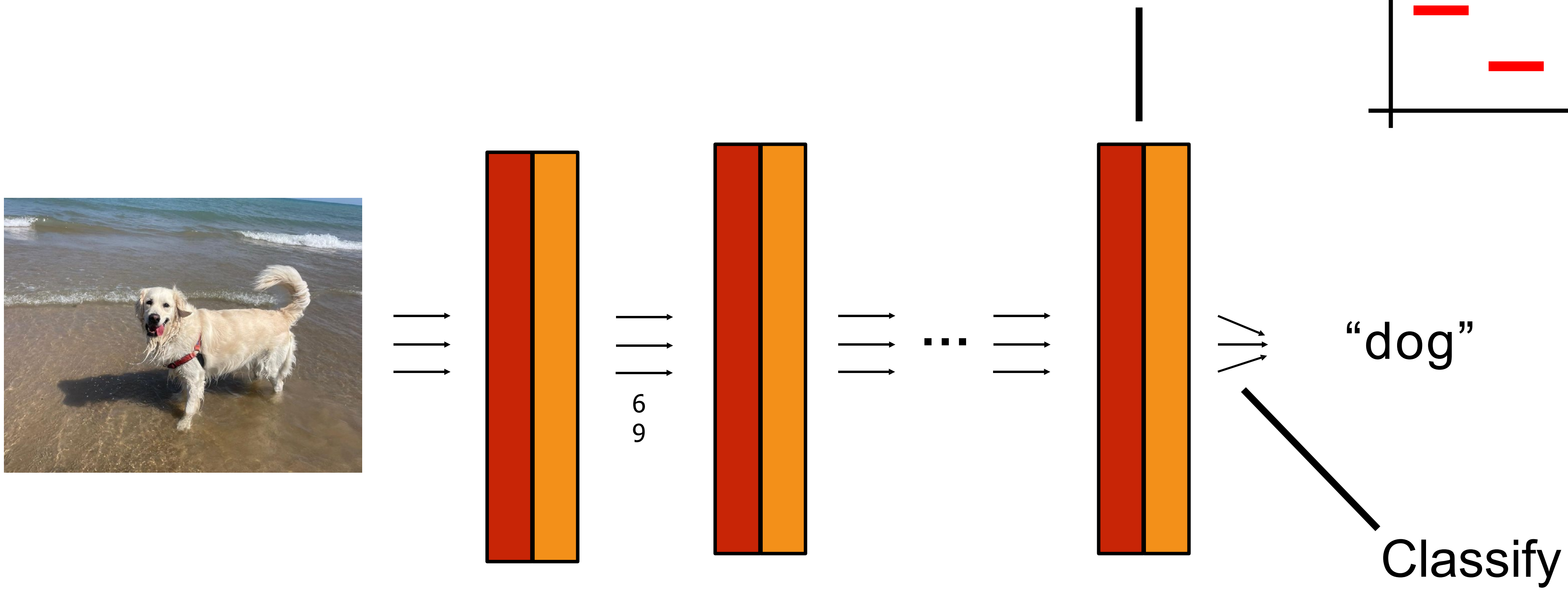
"has white fur" "has paw" etc



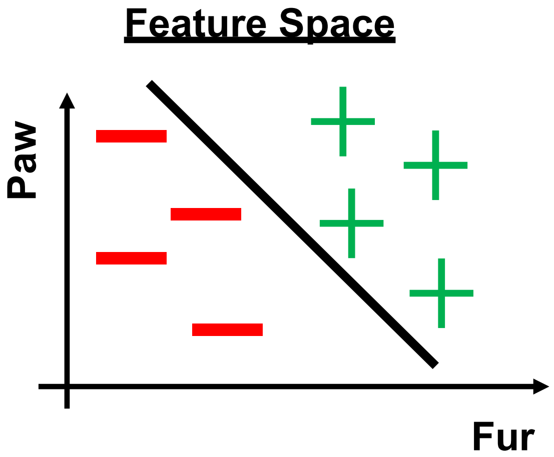
How do we make a classification?

"dog"

Deep nets - Intuition



Recall:



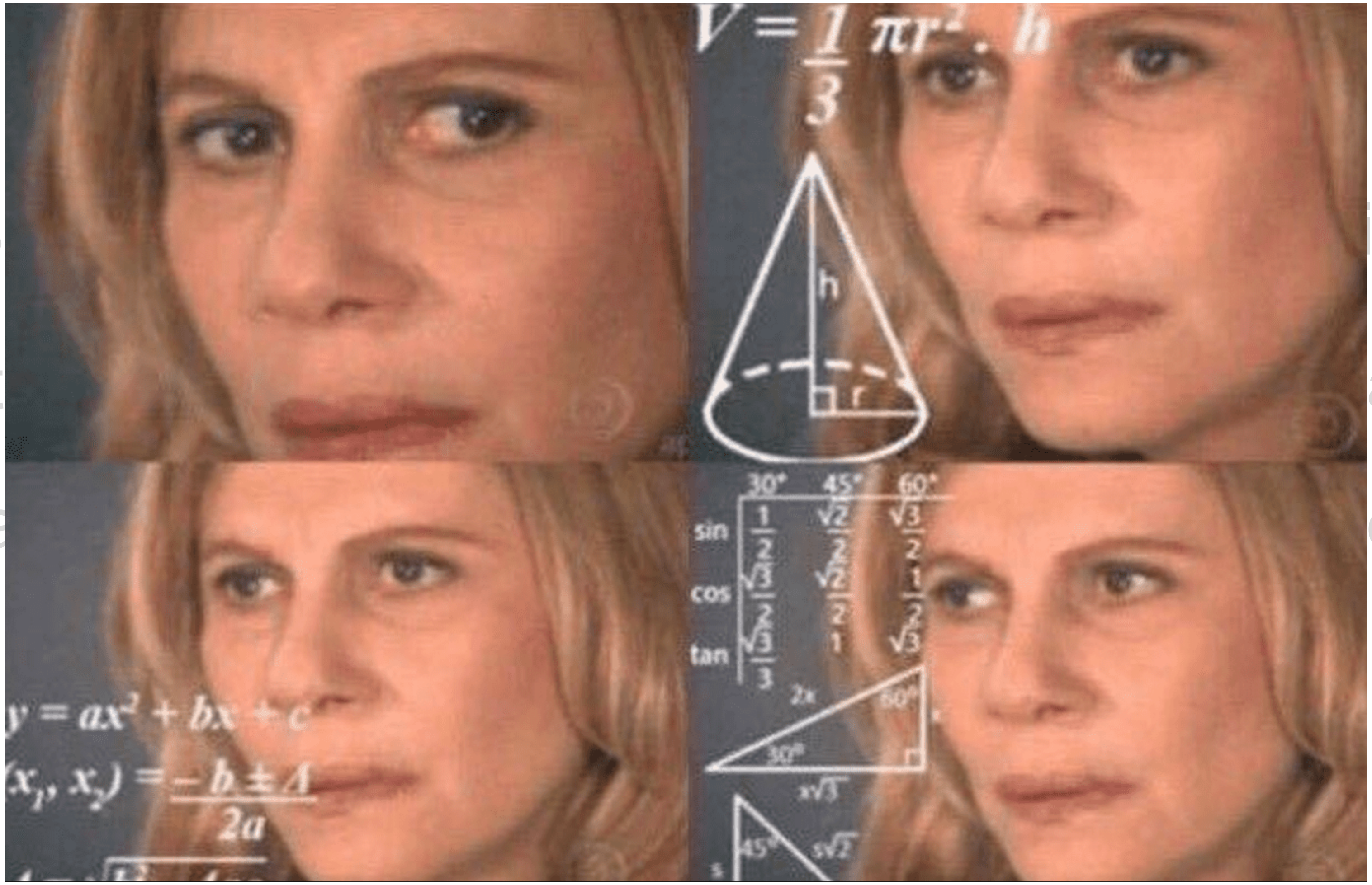
Source: Isola, Torralba, Freeman

Computation has a simple form

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

But where do we get the weights from?

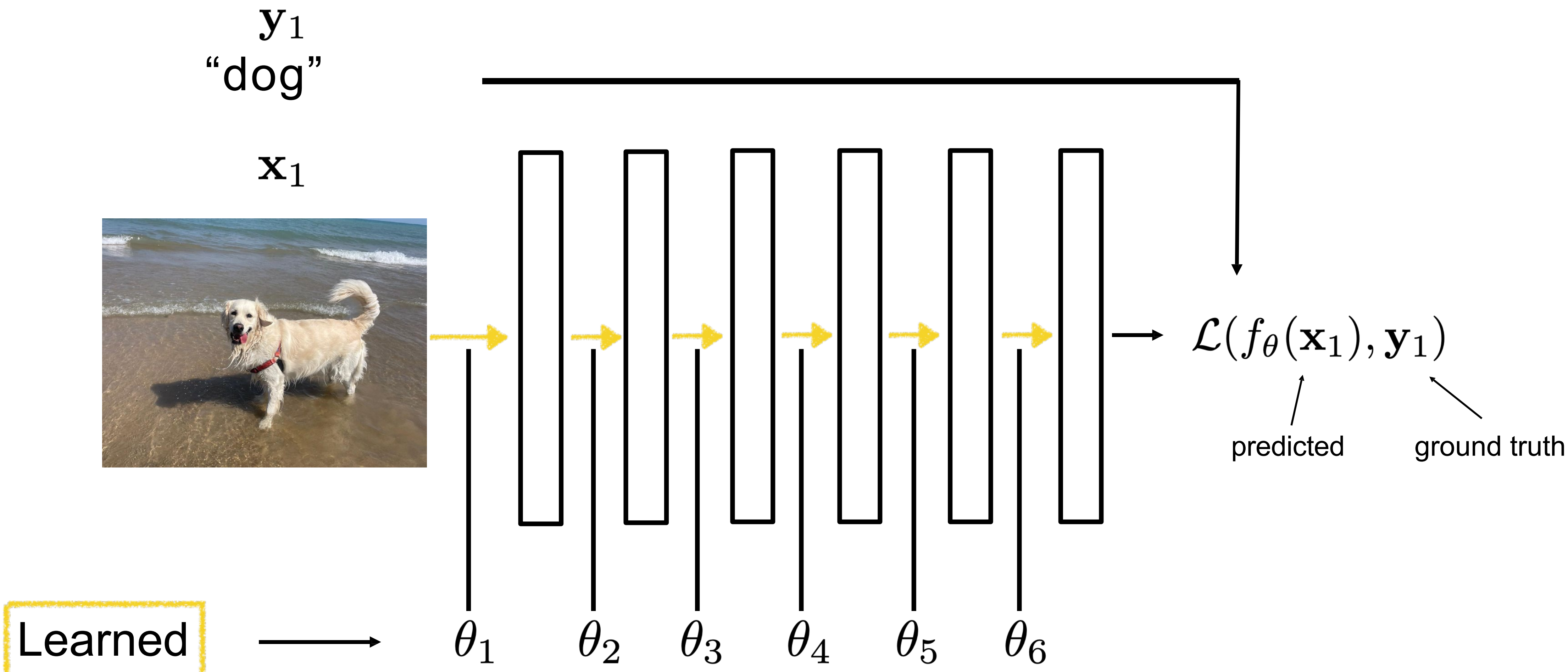
Computation has a simple form



- Compos… …in between
- E.g. mat… …afterwards
- Do a ma… …o 0, repeat

But where do we get the weights from?

How would we learn the parameters?



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