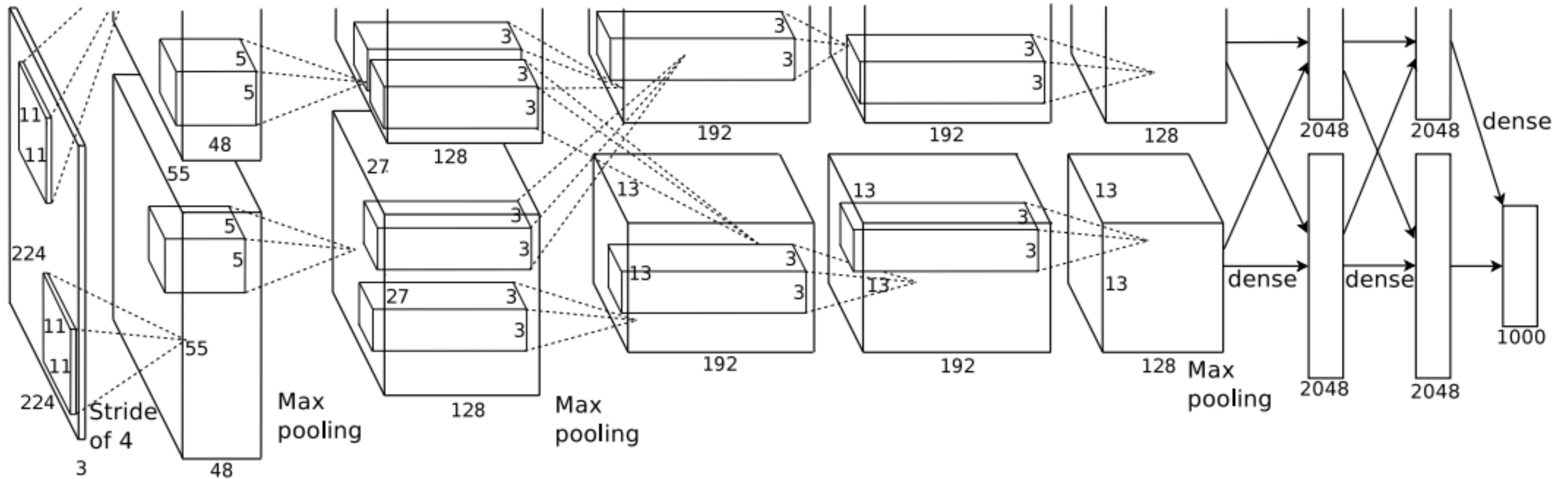


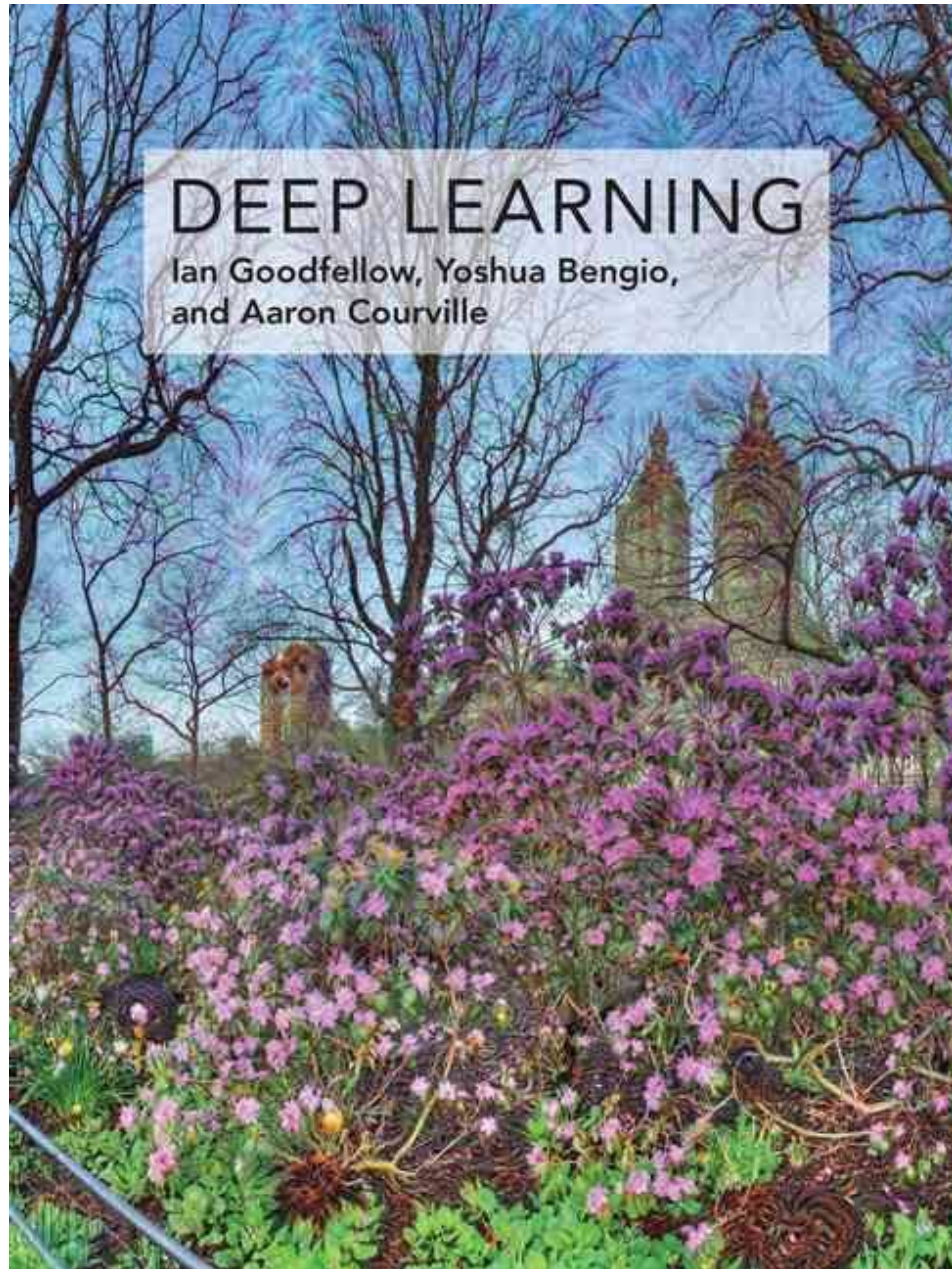
Neural networks

Bill Freeman, Antonio Torralba, Phillip Isola

6.819 / 6.869

Announcements

- Final project proposals due Thursday
- Pytorch and Tensorflow tutorials (location TBA)
  - Pytorch: 10/26 at 3pm, 10/29 at 12pm
  - Tensorflow: 10/30 at 2pm, 10/31 at 6pm



http://www.deeplearningbook.org/

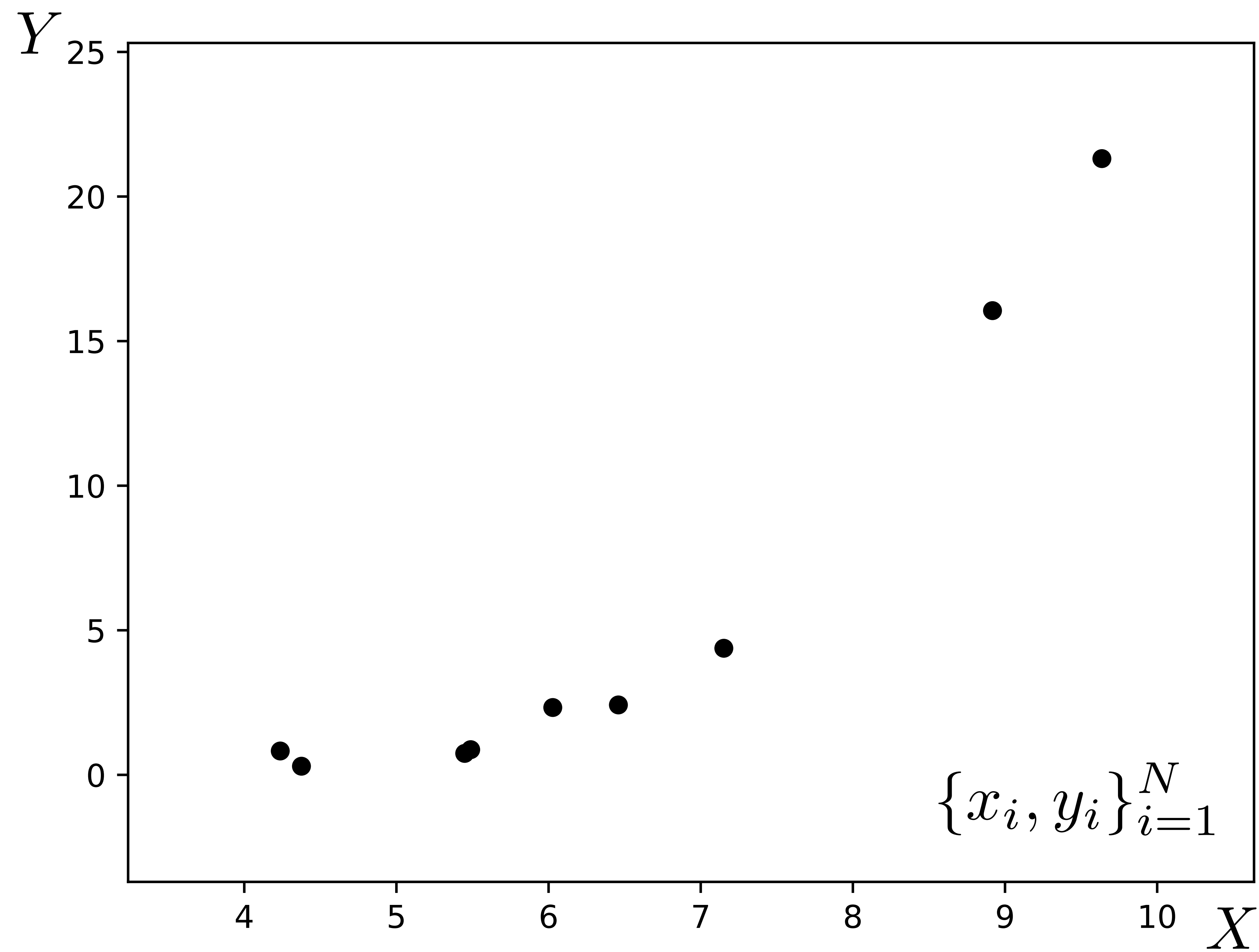
By Ian Goodfellow, Yoshua Bengio and Aaron Courville

November 2016

Today: parts of chapters 5 and 6

Linear regression

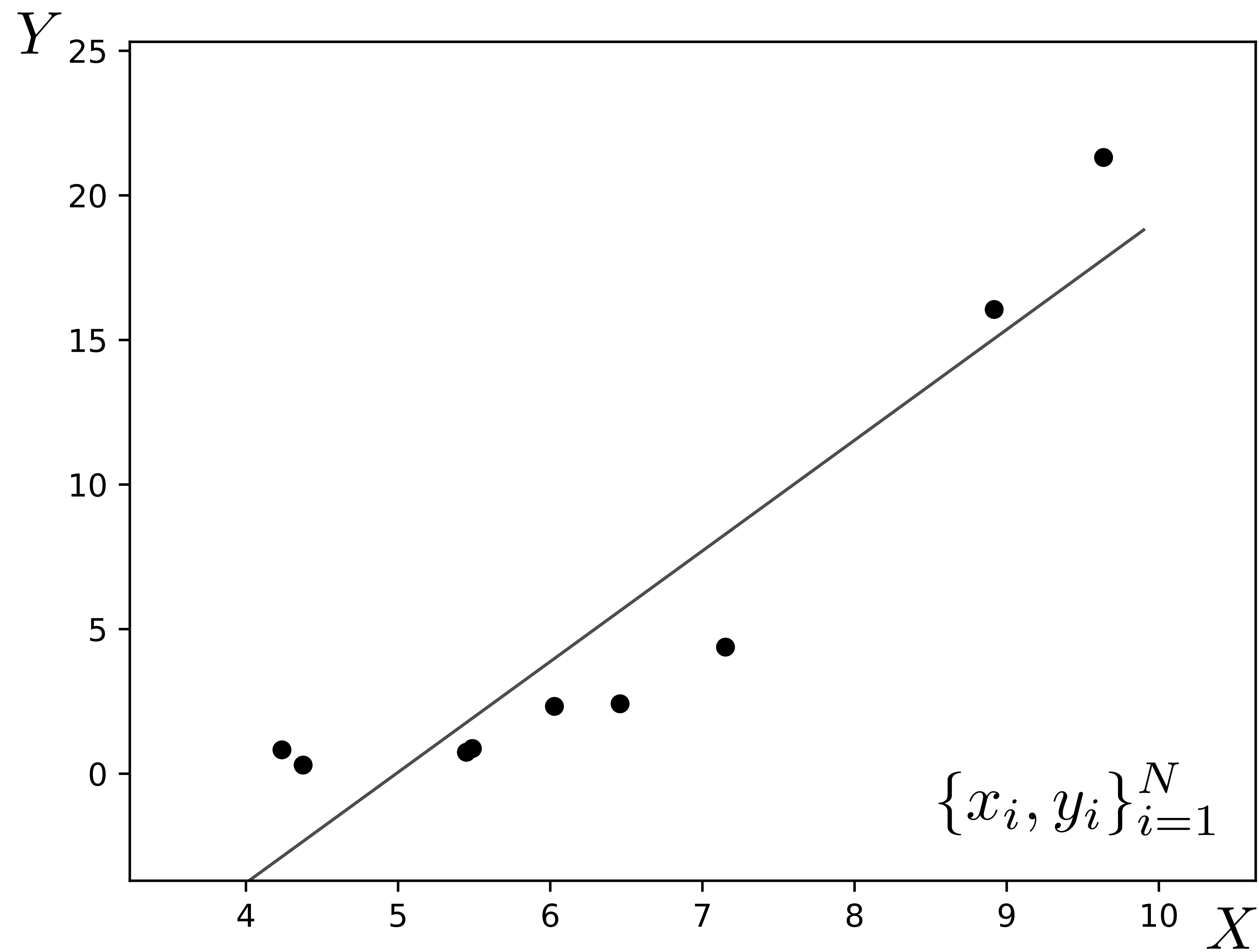
Training data



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

Linear regression

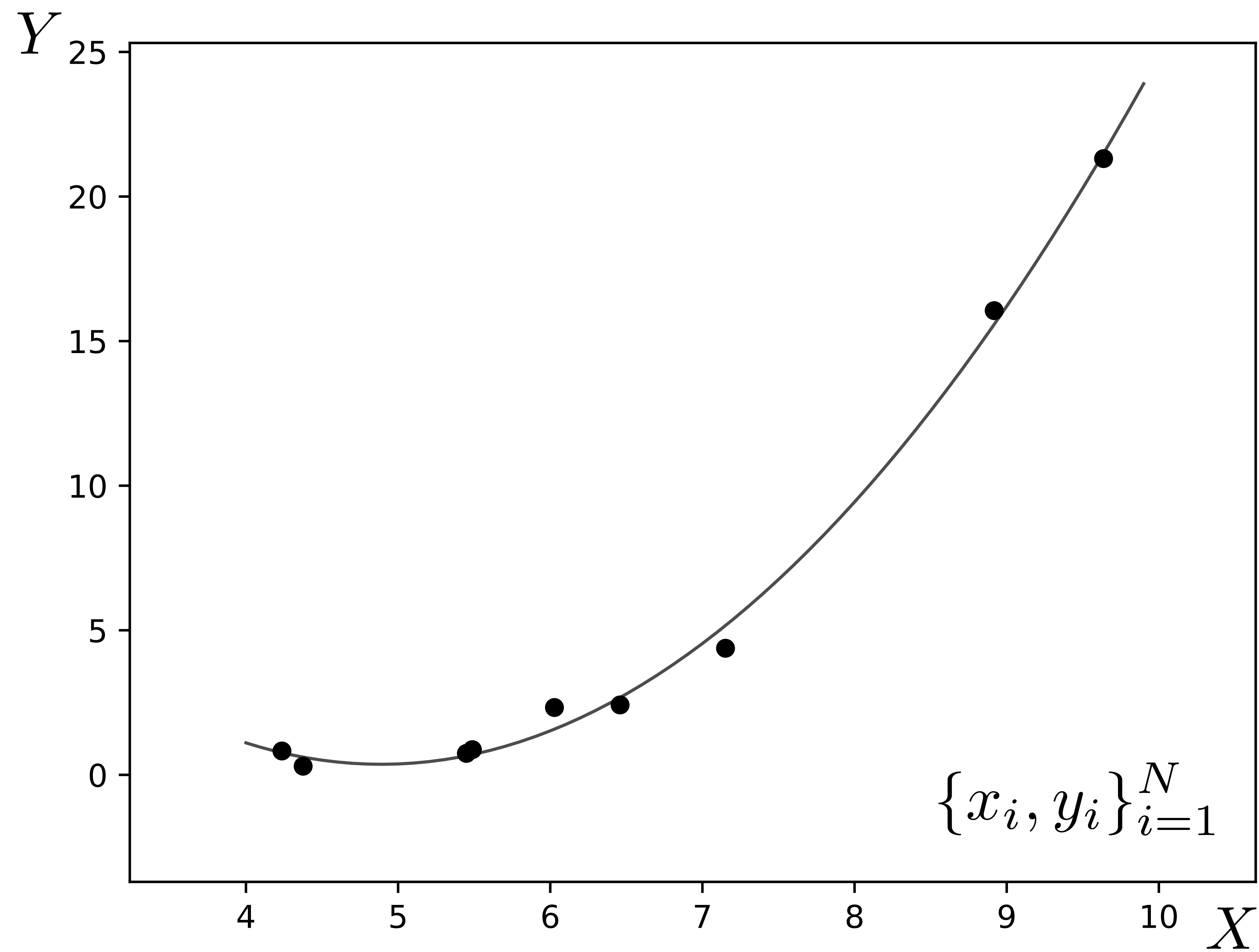
Training data



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

Polynomial regression

Training data

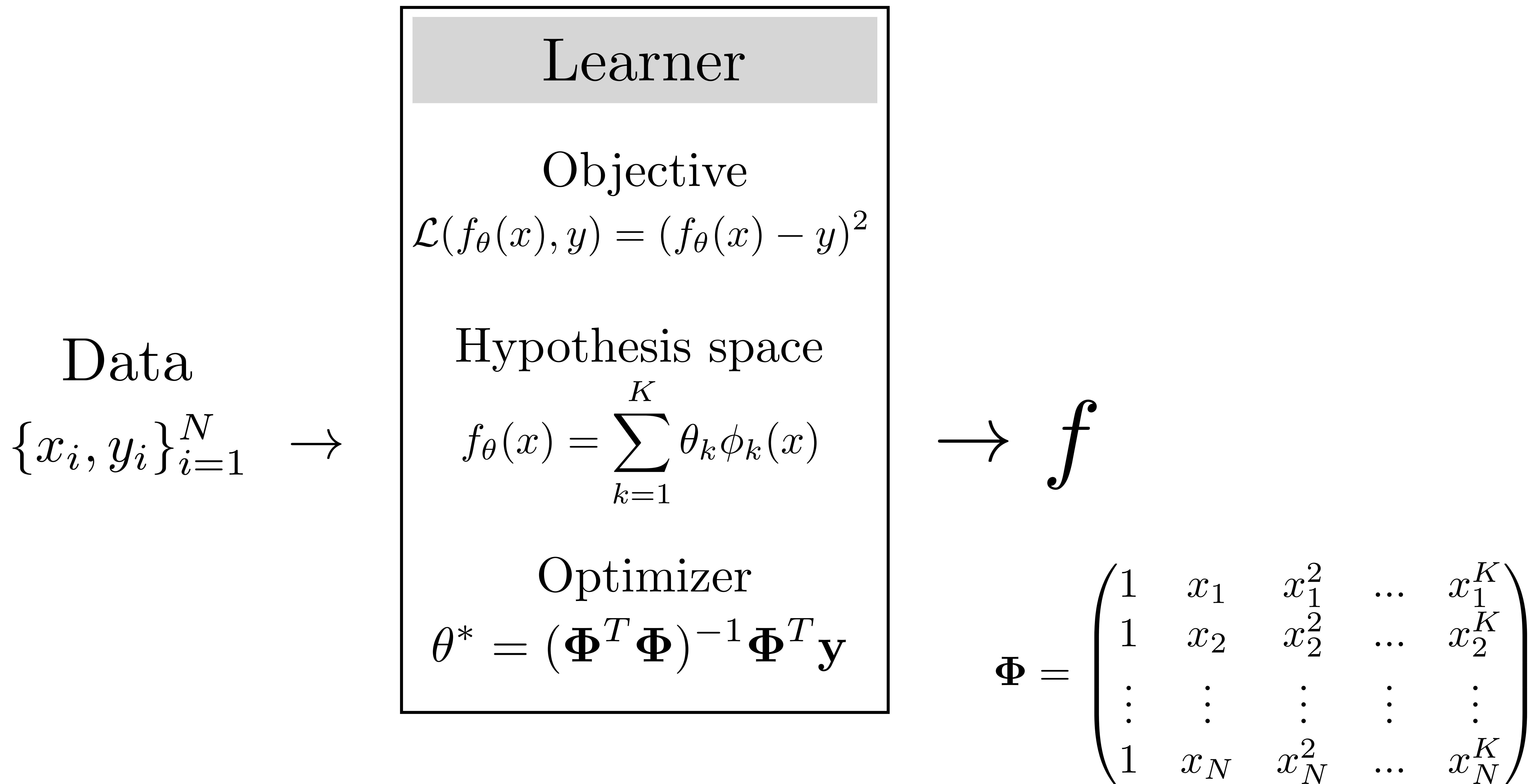


$$f_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

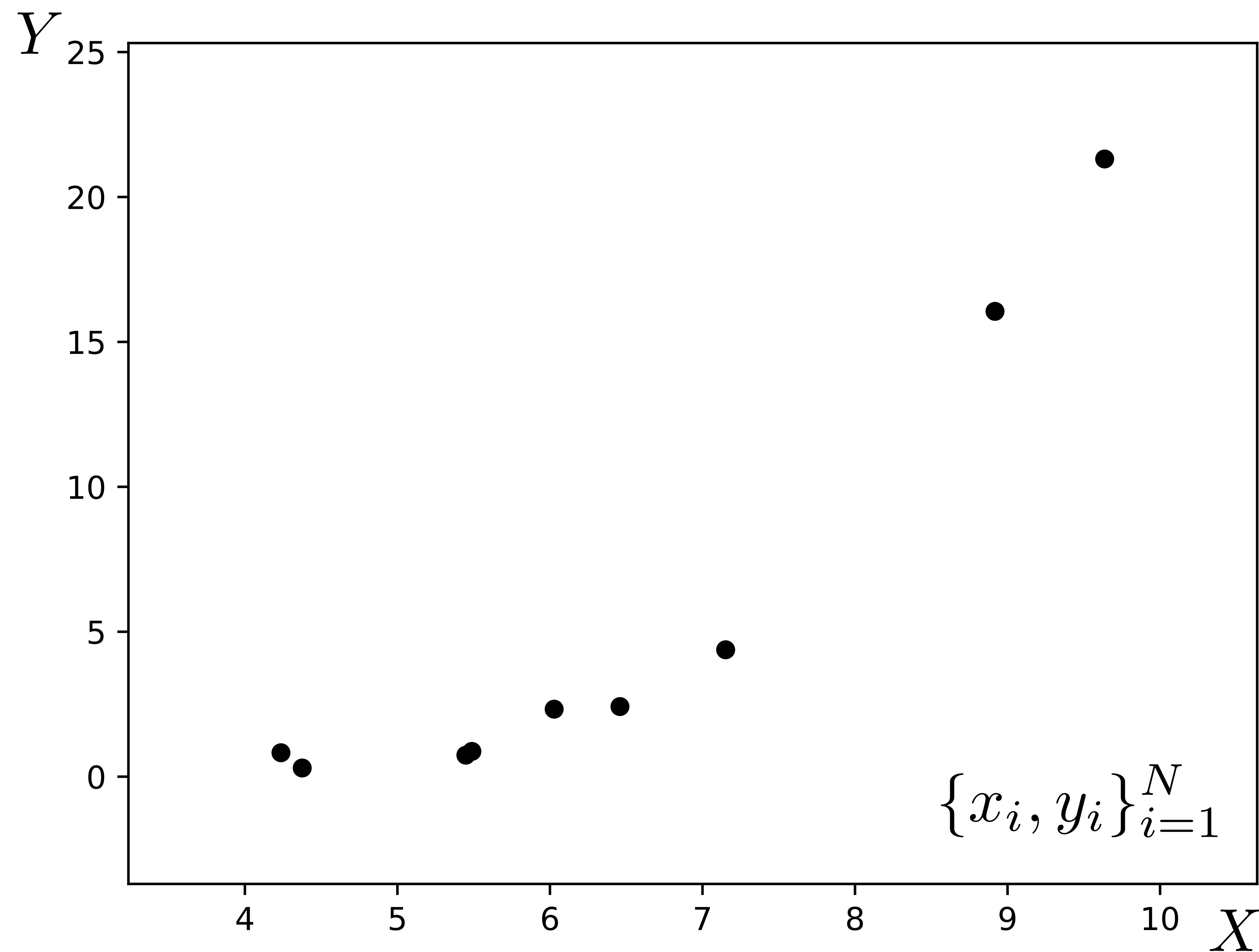
$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

K-th degree polynomial regression

Least-squares polynomial regression



Basis function regression



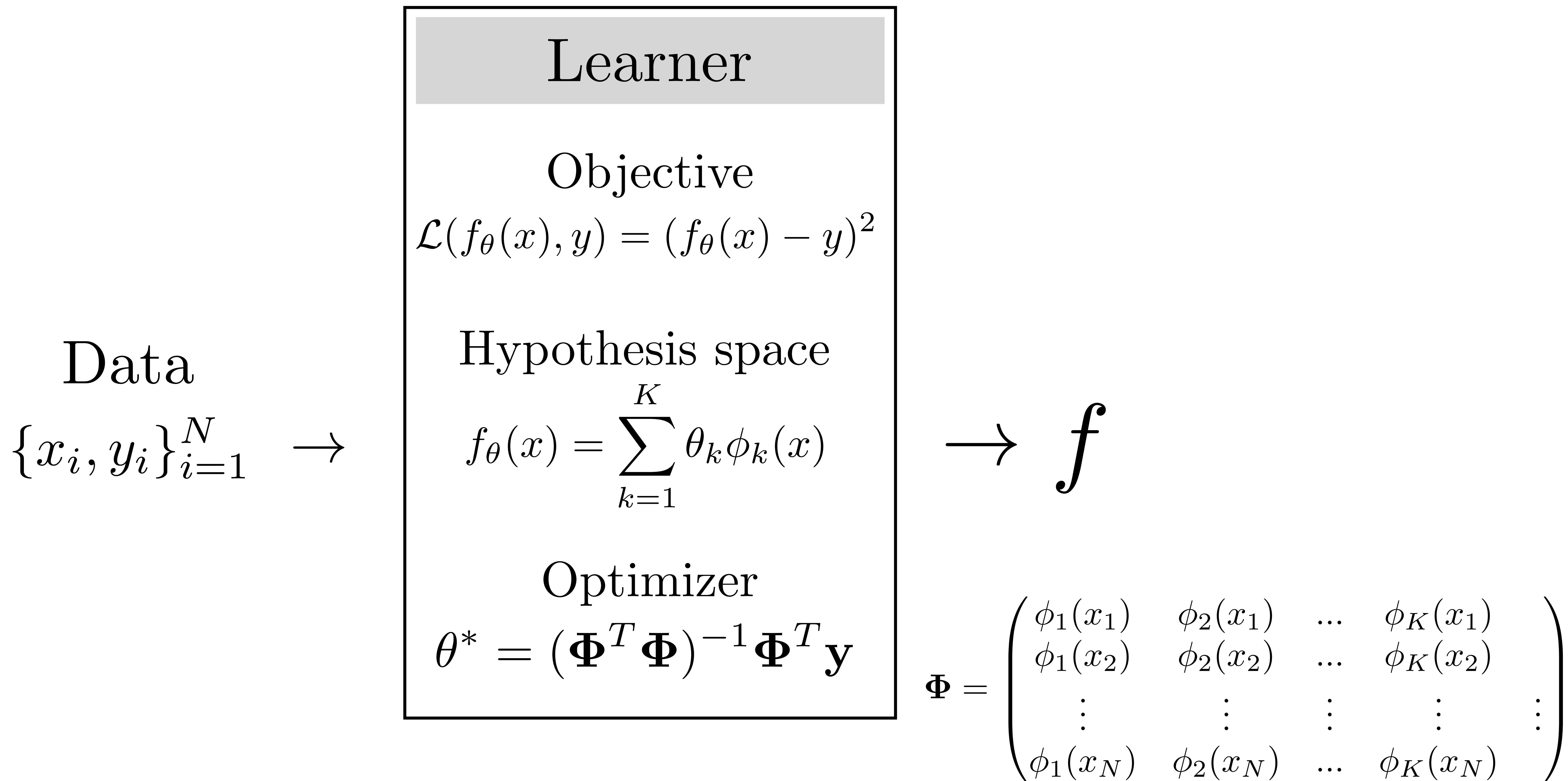
Basis function
(a.k.a. **feature map**)

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

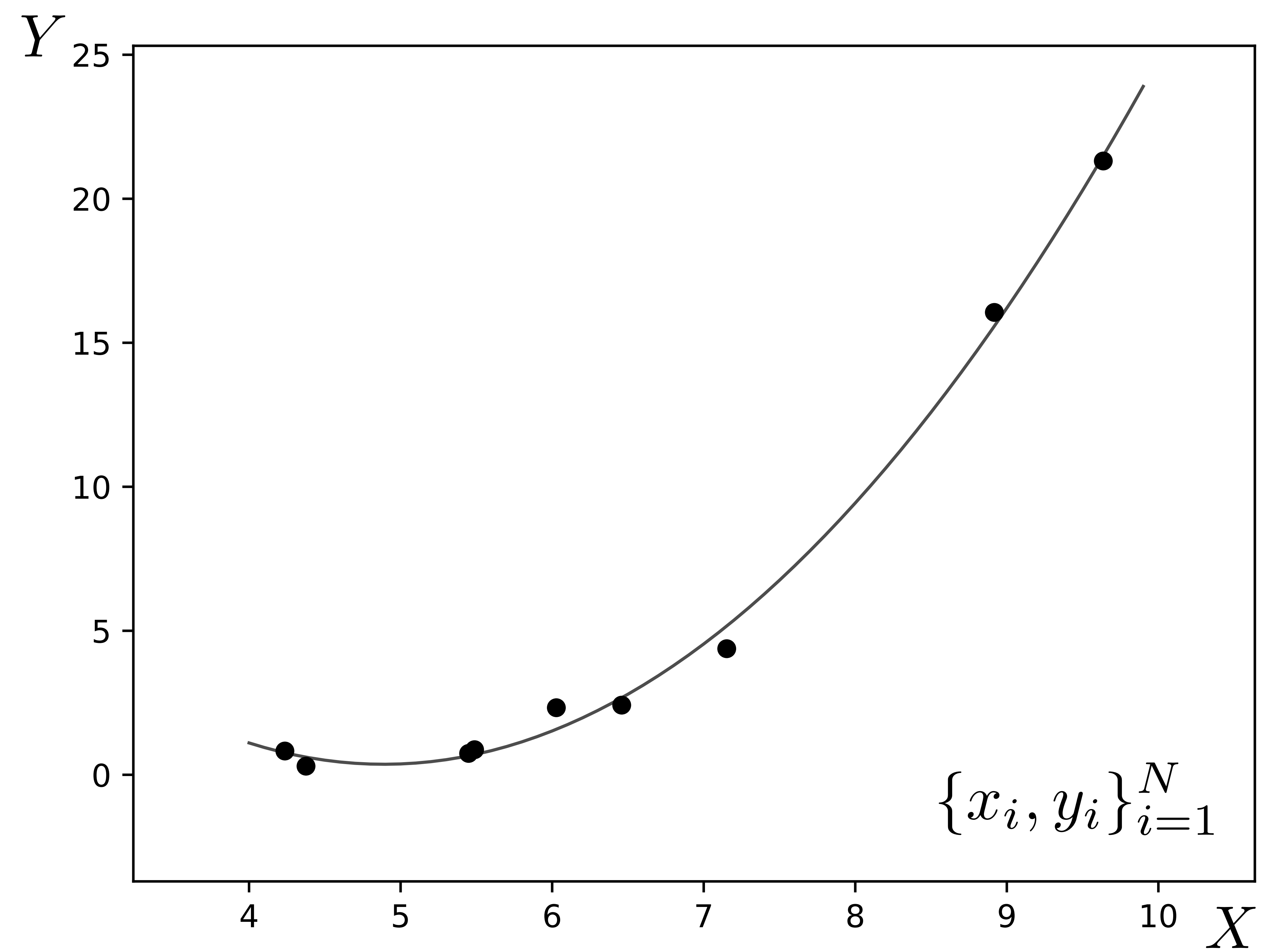
Examples:

1. Polynomials
2. Fourier basis ("Harmonic regression")
3. Image "features"

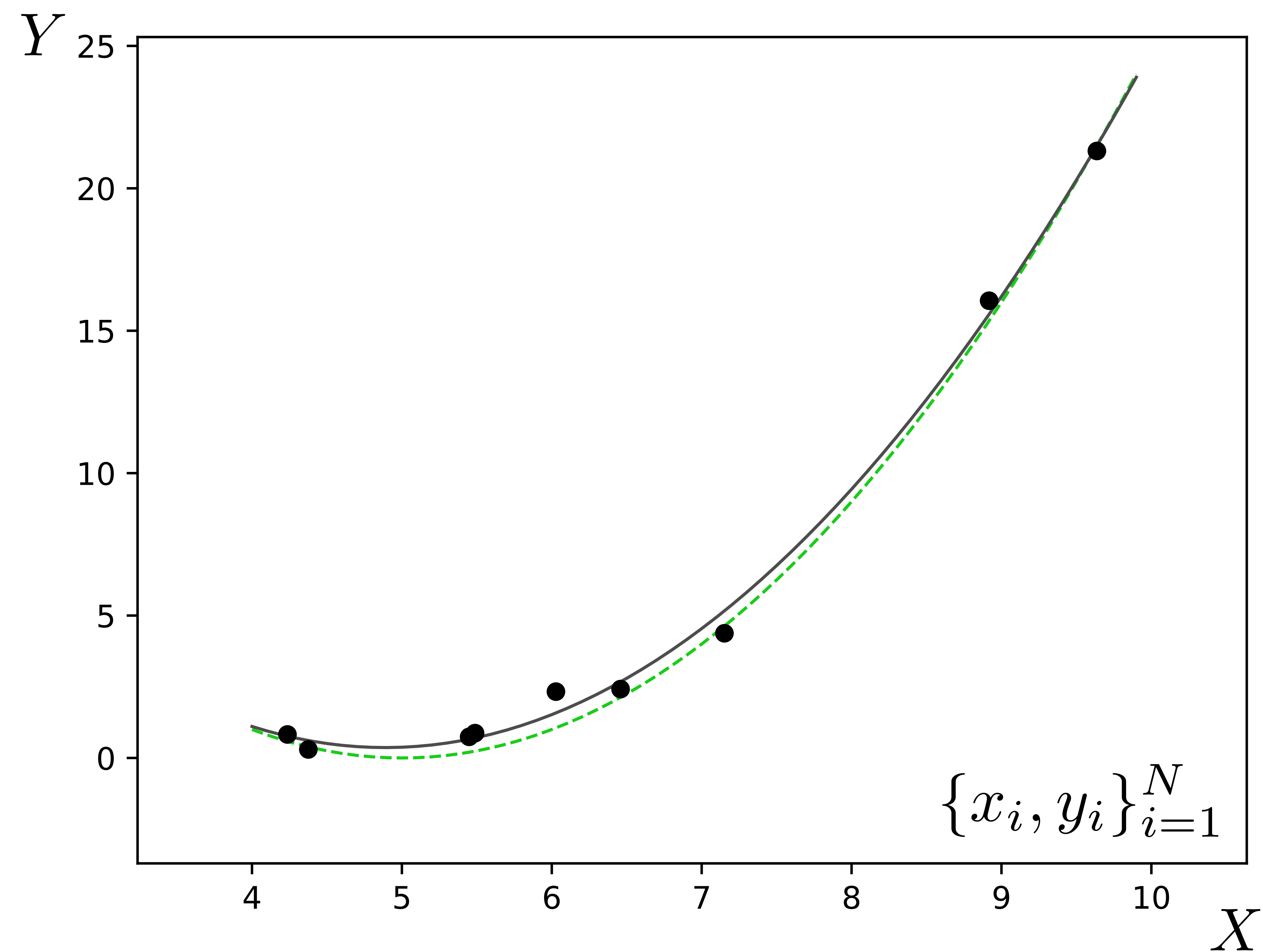
Basis function regression



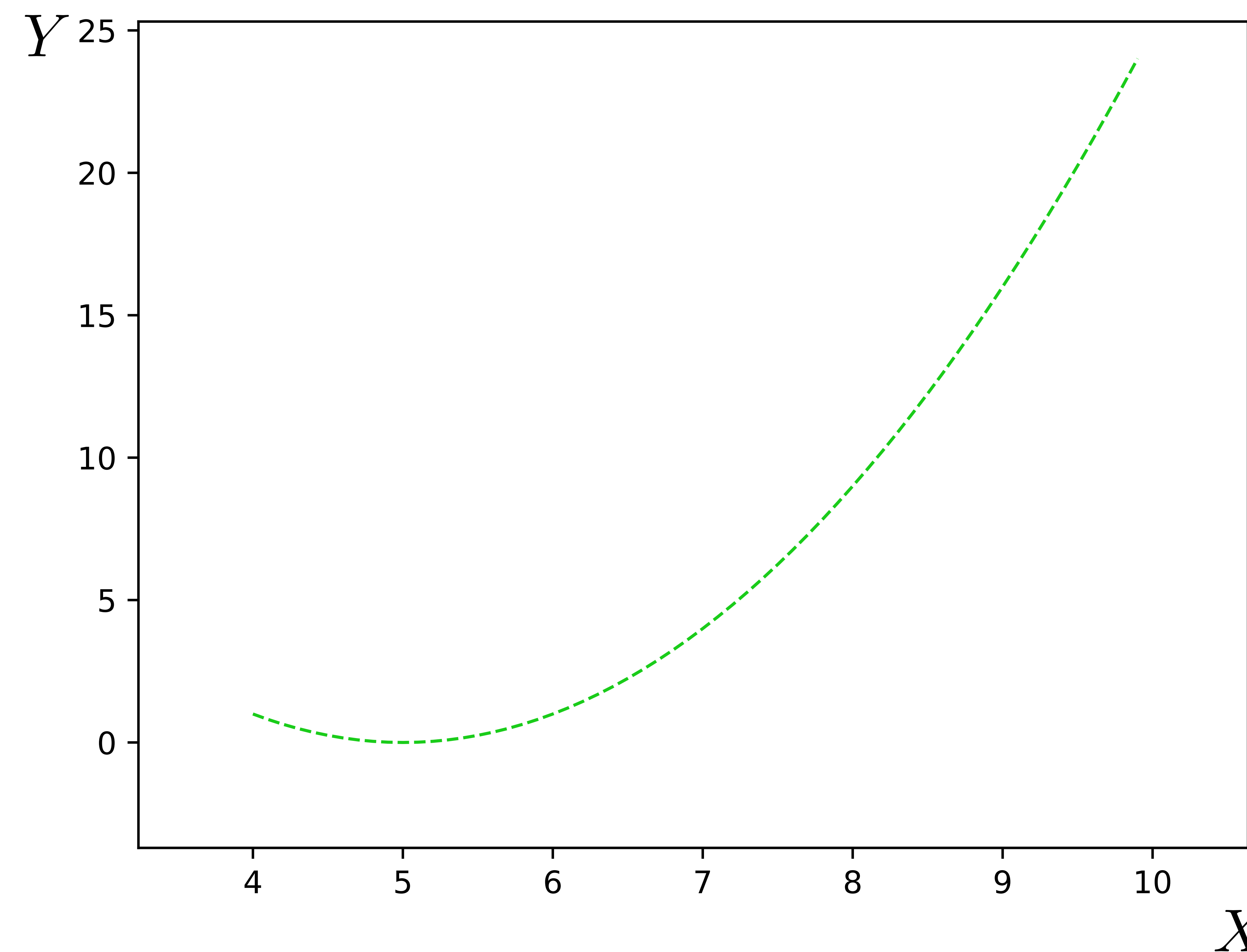
Training data



Training data



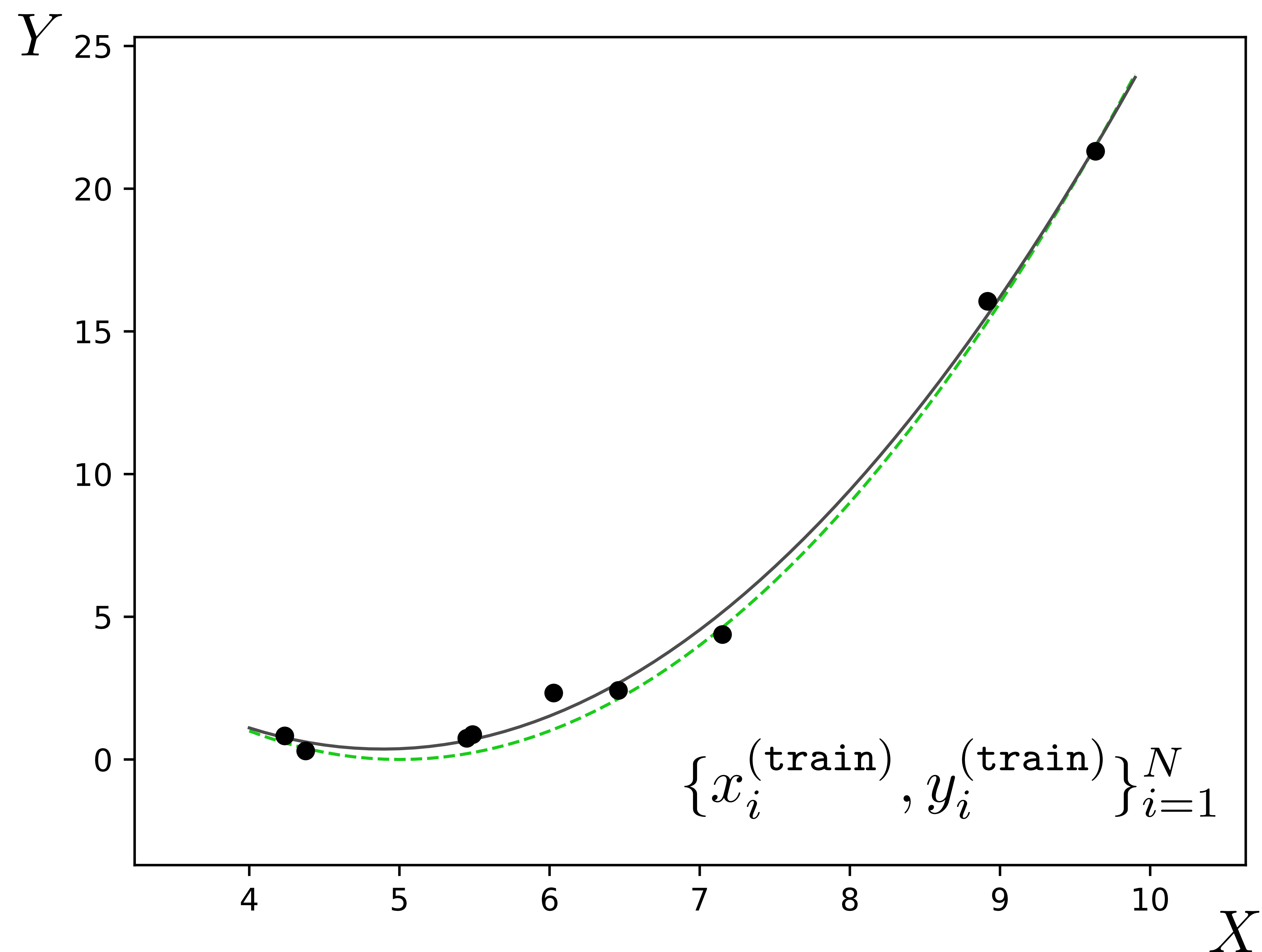
Test data



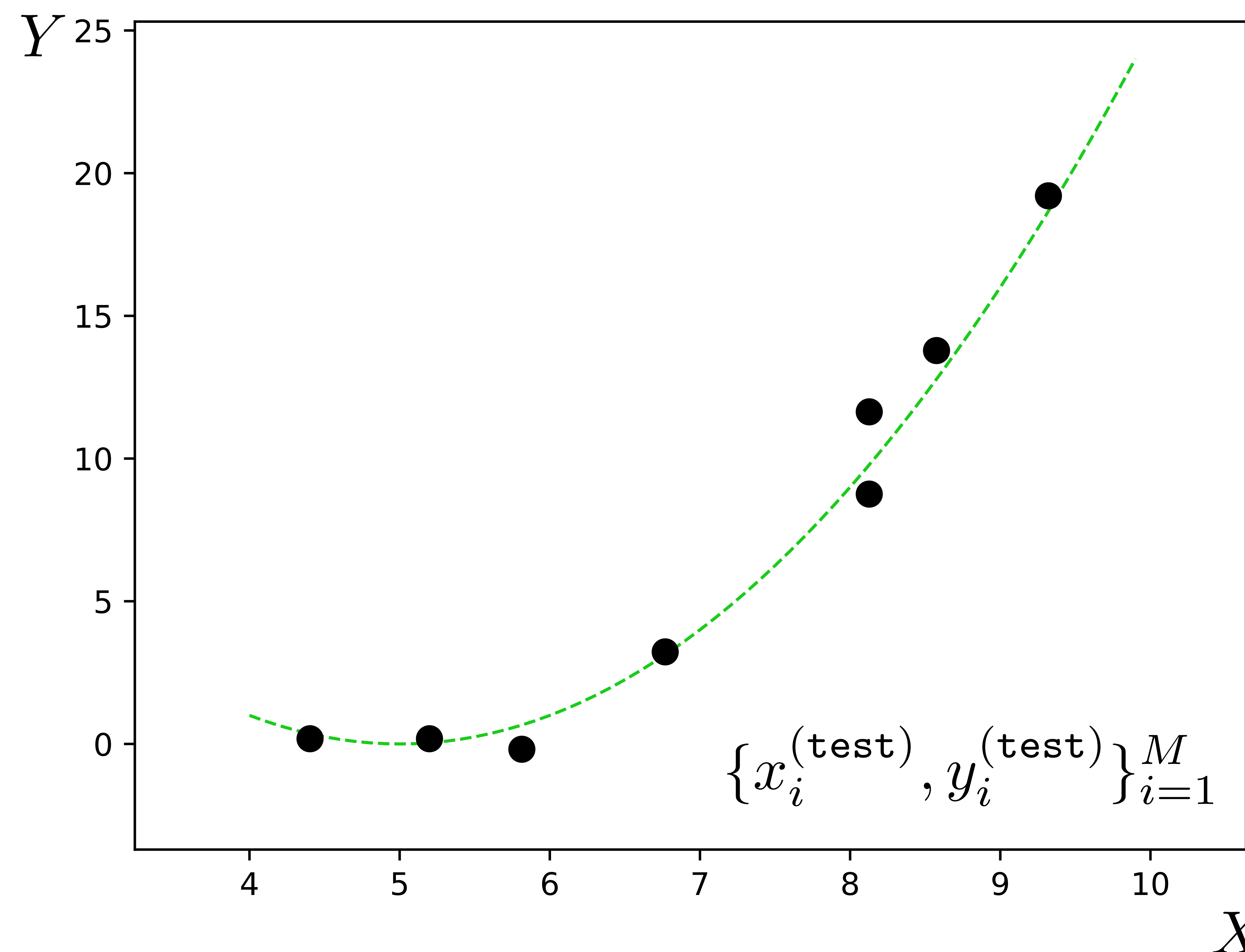
True **data-generating process**

$$p_{\texttt{data}}$$

Training data



Test data



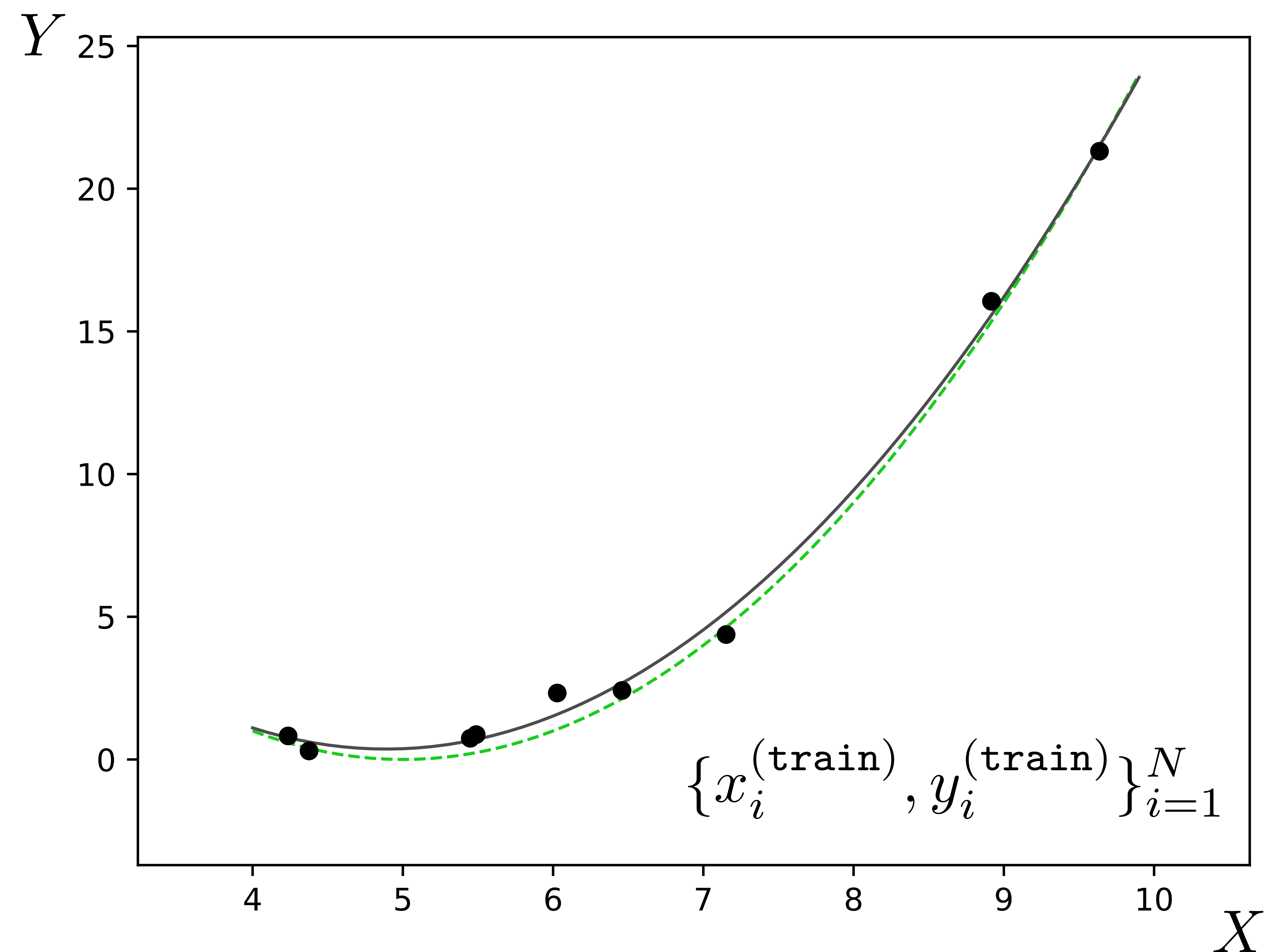
True **data-generating process**

$$p_{\texttt{data}}$$

$$\{x_i^{(\texttt{train})}, y_i^{(\texttt{train})}\} \overset{\texttt{iid}}{\sim} p_{\texttt{data}}$$

$$\{x_i^{(\texttt{test})}, y_i^{(\texttt{test})}\} \overset{\texttt{iid}}{\sim} p_{\texttt{data}}$$

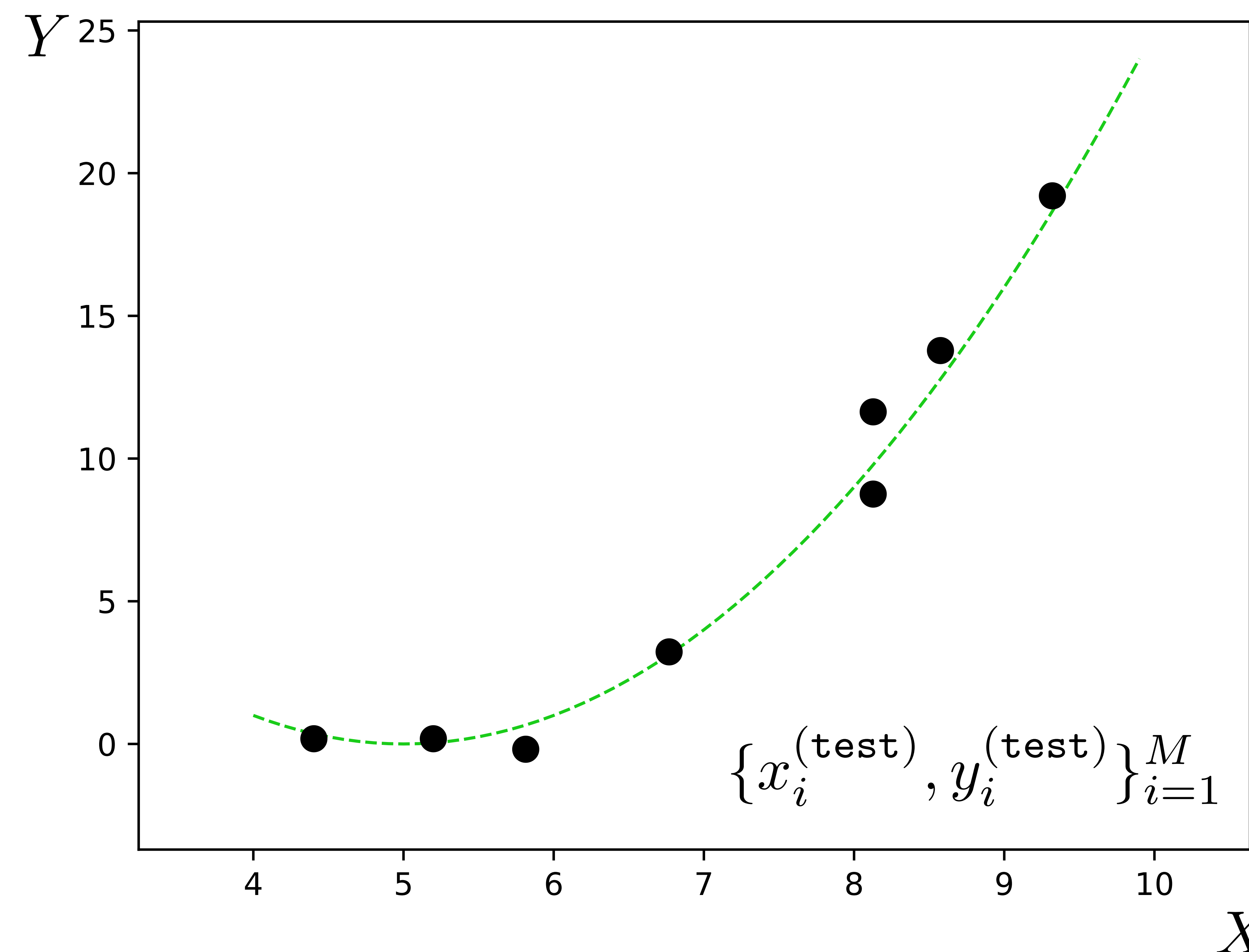
Training data



Training objective:

$$\sum_{i=1}^{N} (f_\theta(x_i^{\texttt{train}}) - y_i^{\texttt{train}})^2$$

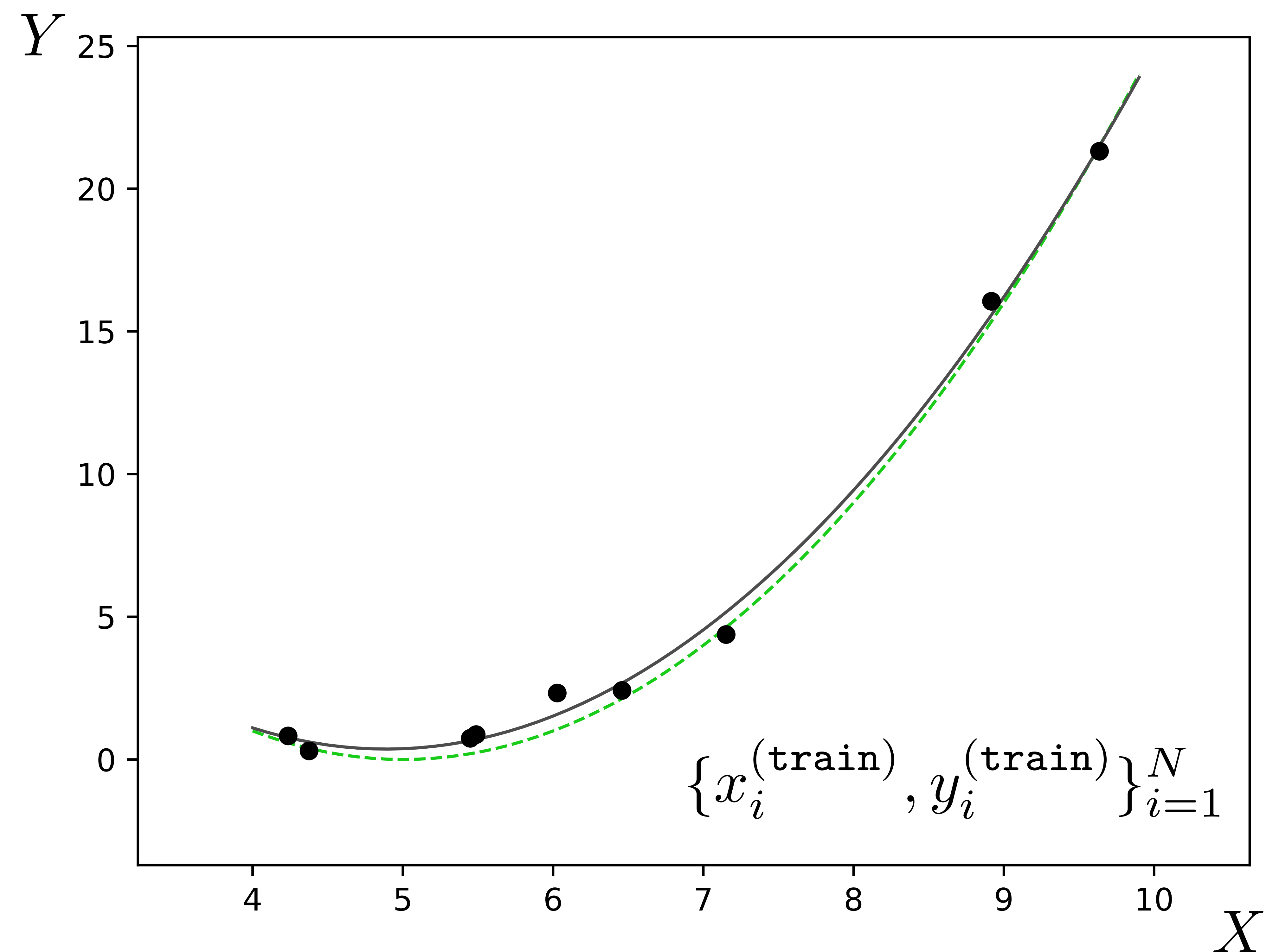
Test data



Test time evaluation:

$$\sum_{i=1}^{M} (f_\theta(x_i^{\texttt{test}}) - y_i^{\texttt{test}})^2$$

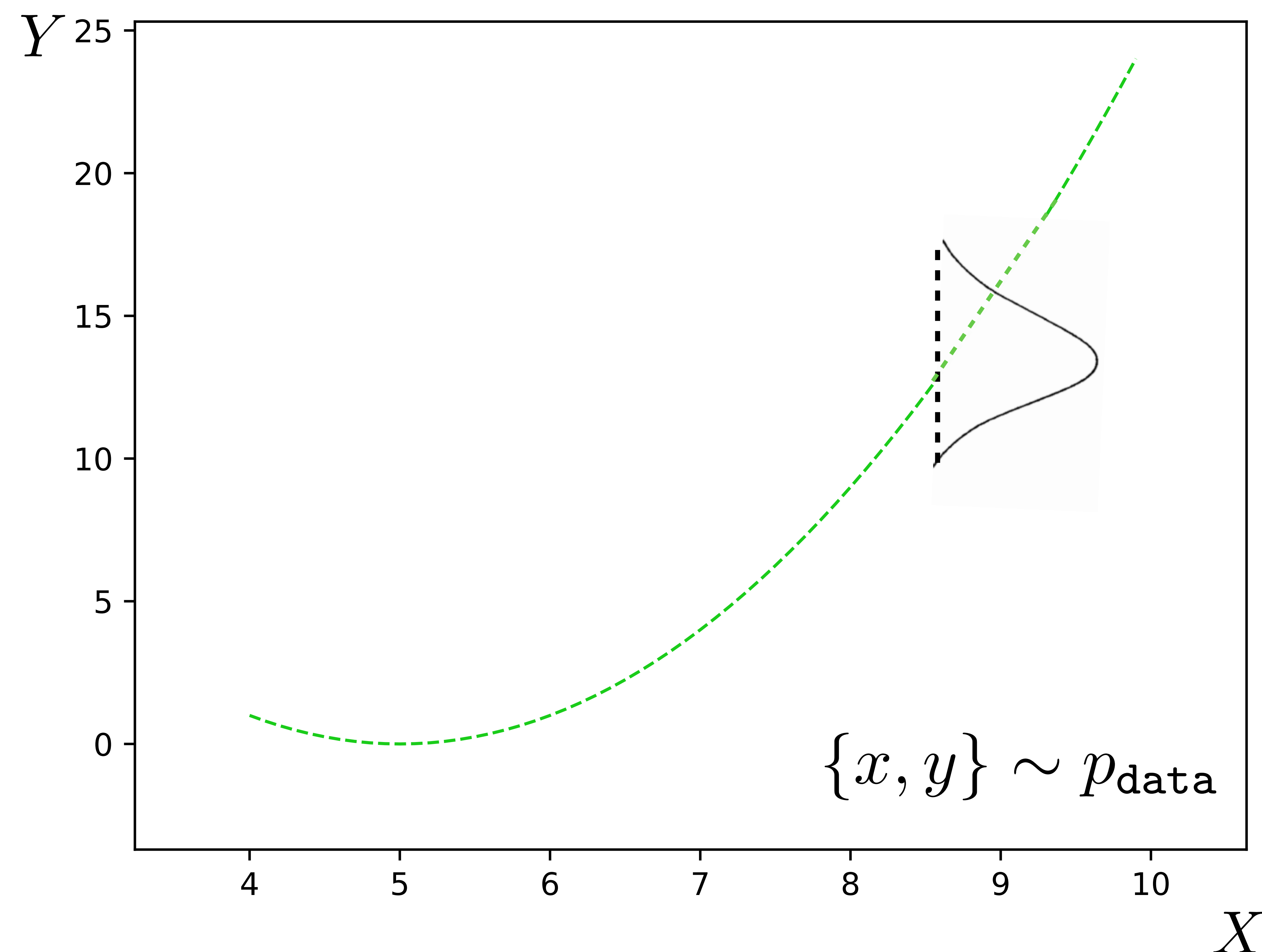
Training data



Training objective:

$$\sum_{i=1}^{N}(f_\theta(x_i^{\texttt{train}}) - y_i^{\texttt{train}})^2$$

Test data



True objective:

$$\mathbb{E}_{\{x,y\}\sim p_{\texttt{data}}}[(f_\theta(x) - y)^2]$$

Generalization

"The central challenge in machine learning is that our algorithm must perform well on new, previously unseen inputs—not just those on which our model was trained. The ability to perform well on previously unobserved inputs is called **generalization**.

… [this is what] separates machine learning from optimization."

— Deep Learning textbook (Goodfellow et al.)

What does ☆ do?

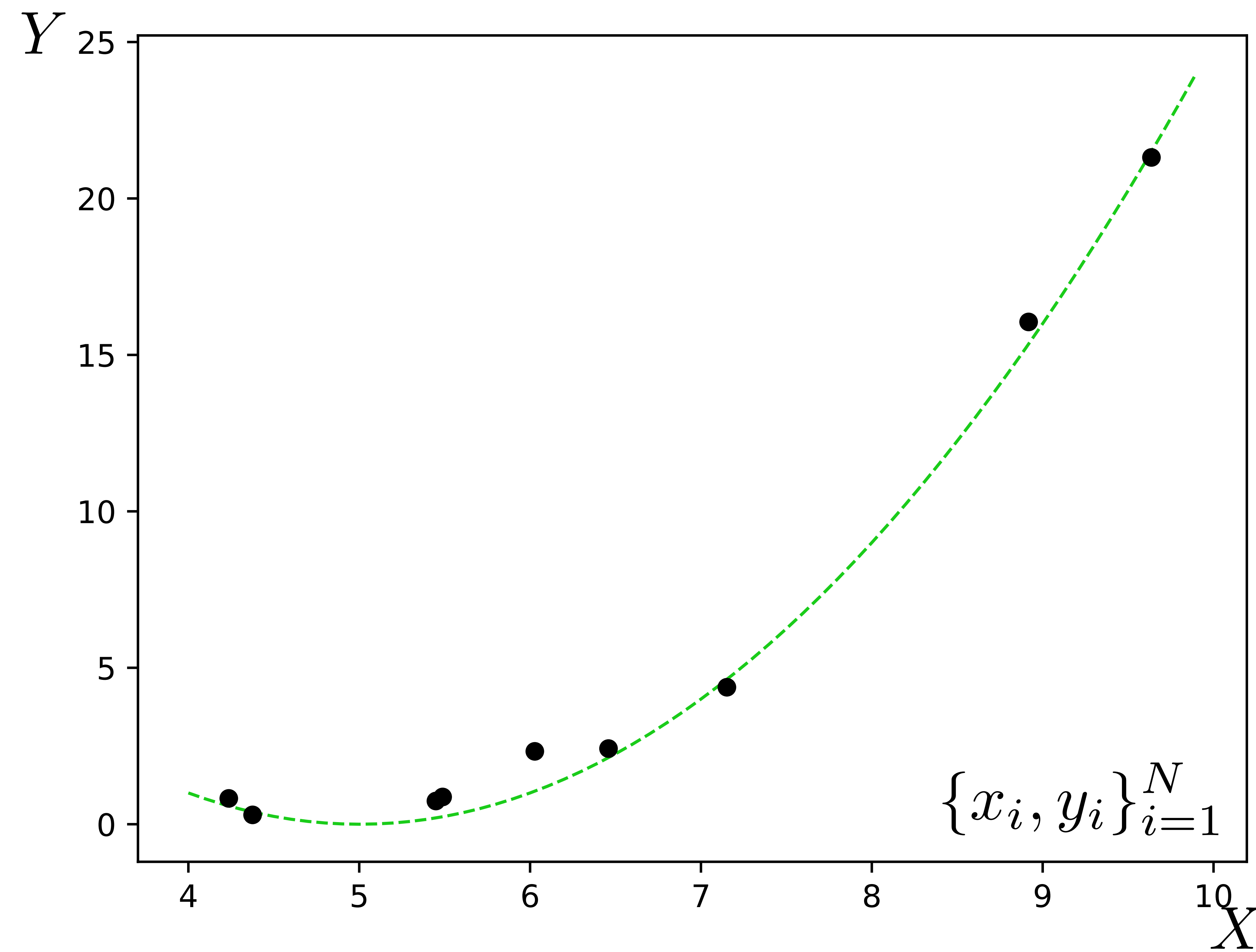
2 ☆ 3 = 36

7 ☆ 1 = 49

5 ☆ 2 = 100

2 ☆ 2 = 16

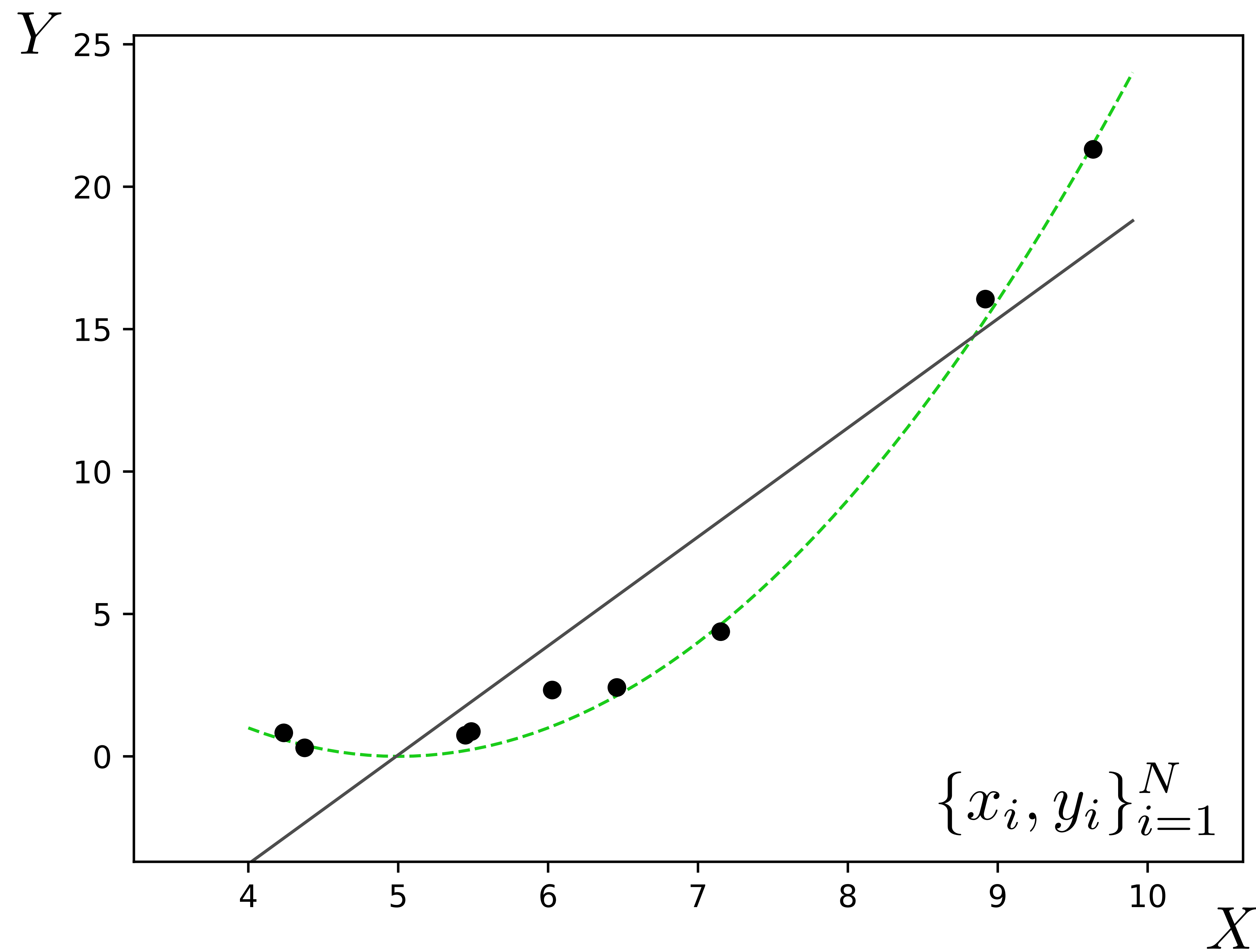
What happens as we add more basis functions?



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

What happens as we add more basis functions?

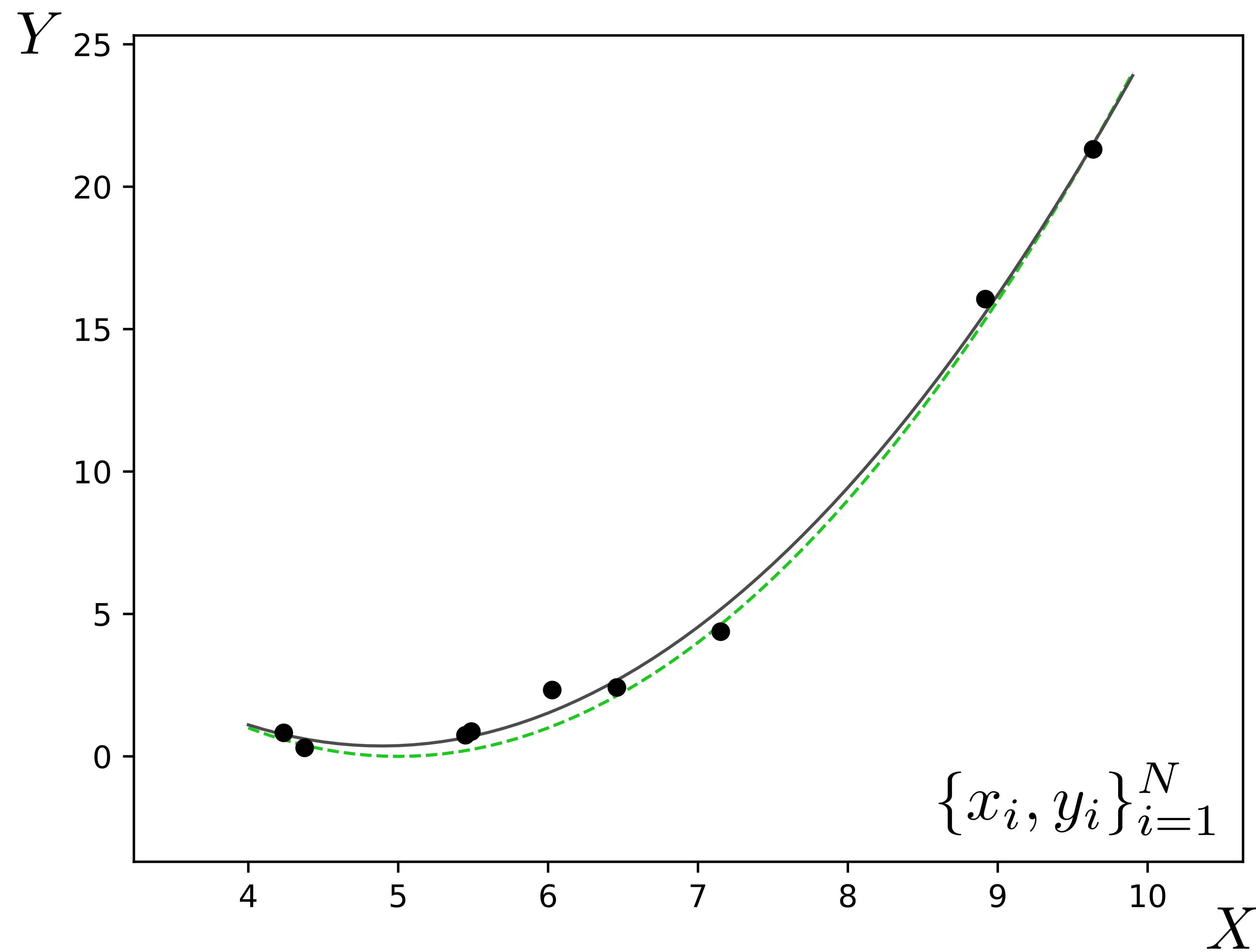
K = 1



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

What happens as we add more basis functions?

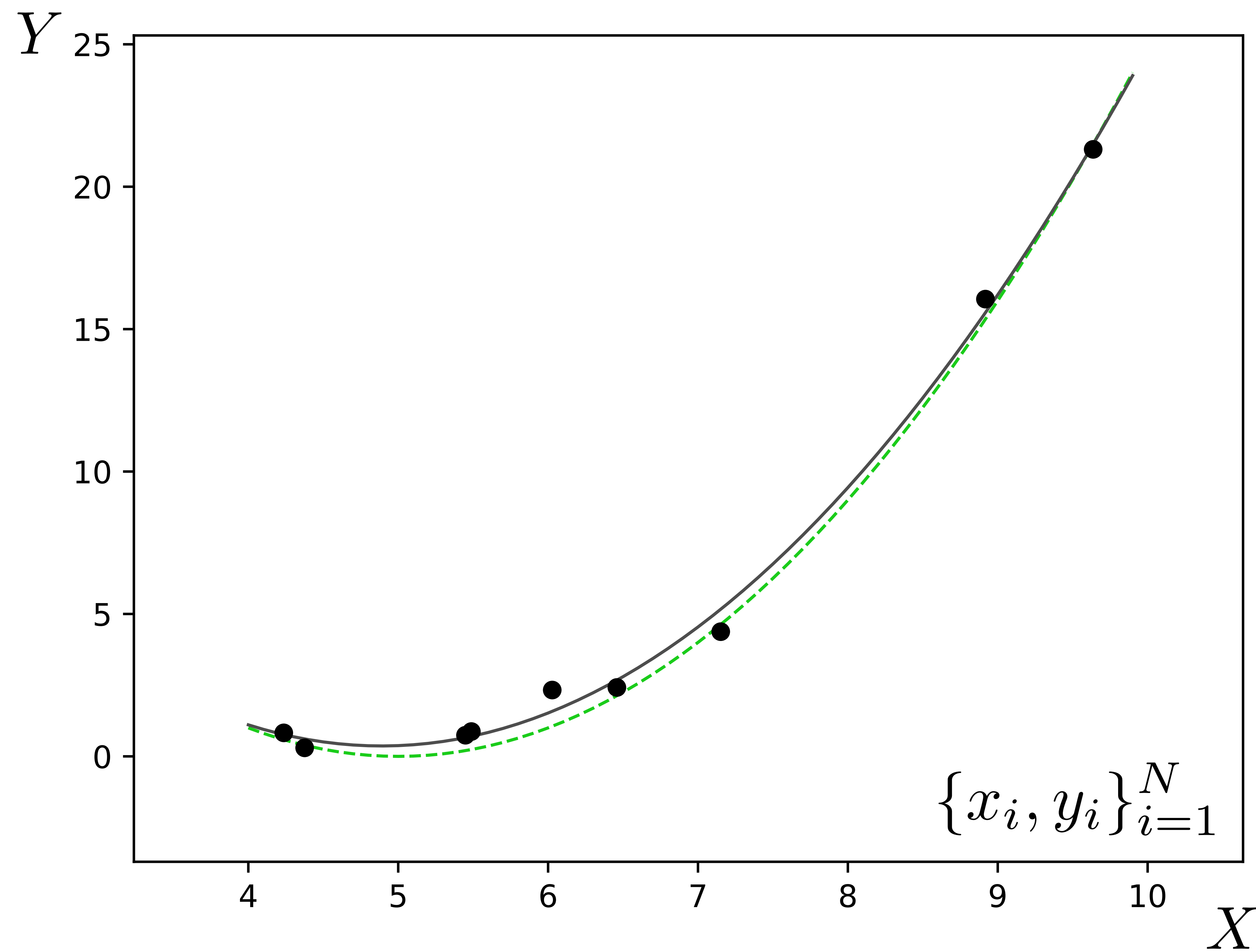
K = 2



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

What happens as we add more basis functions?

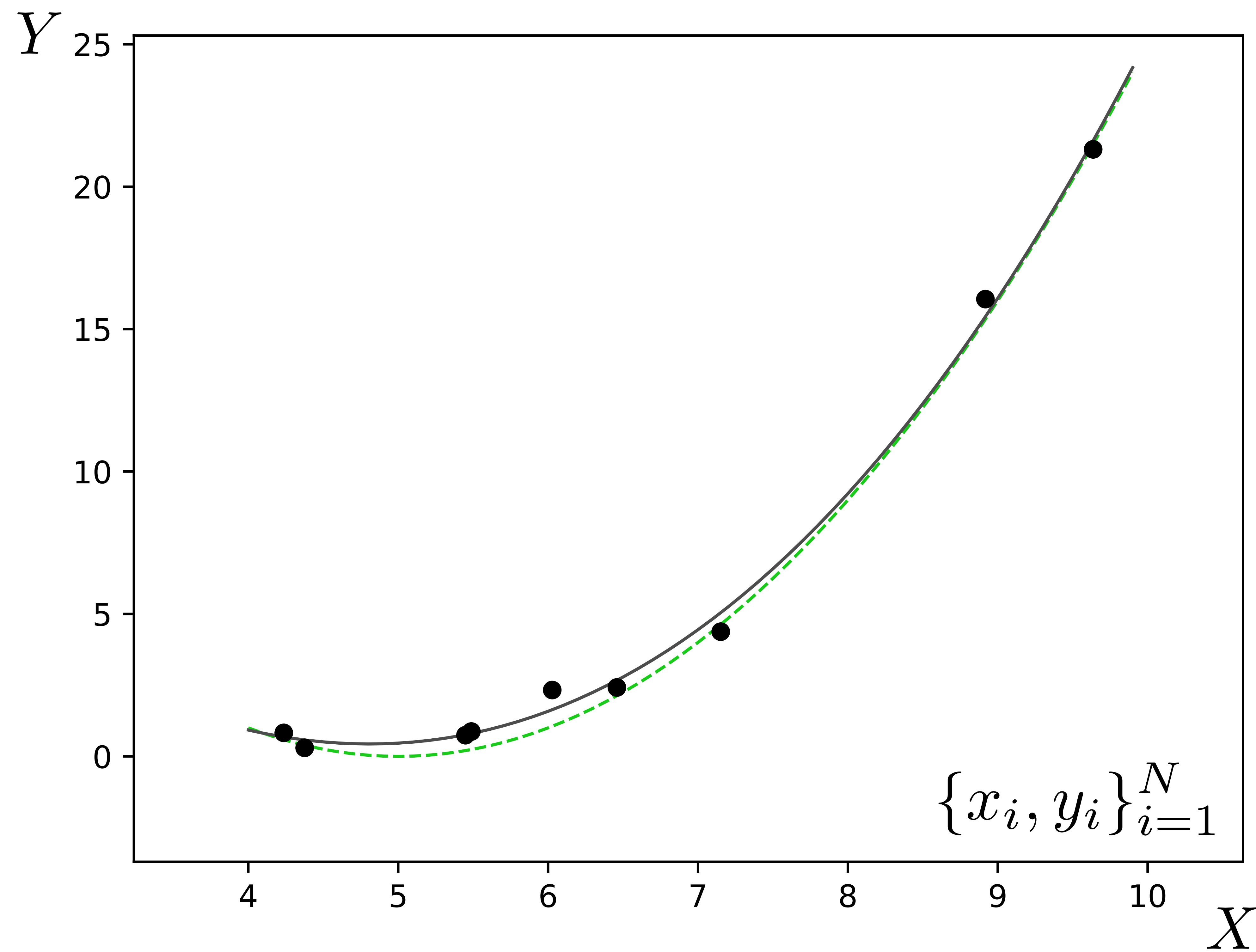
K = 3



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

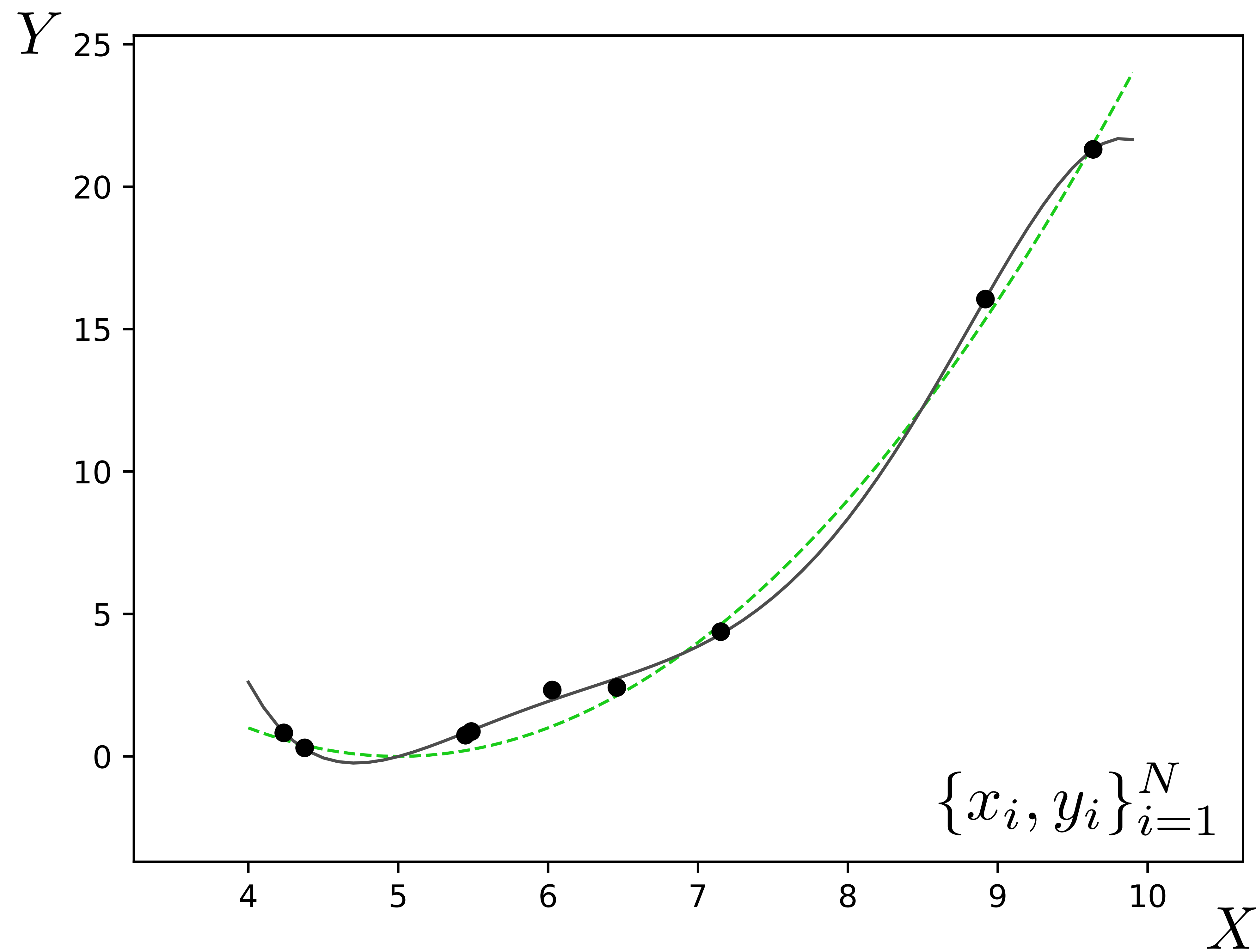
What happens as we add more basis functions?

K = 4



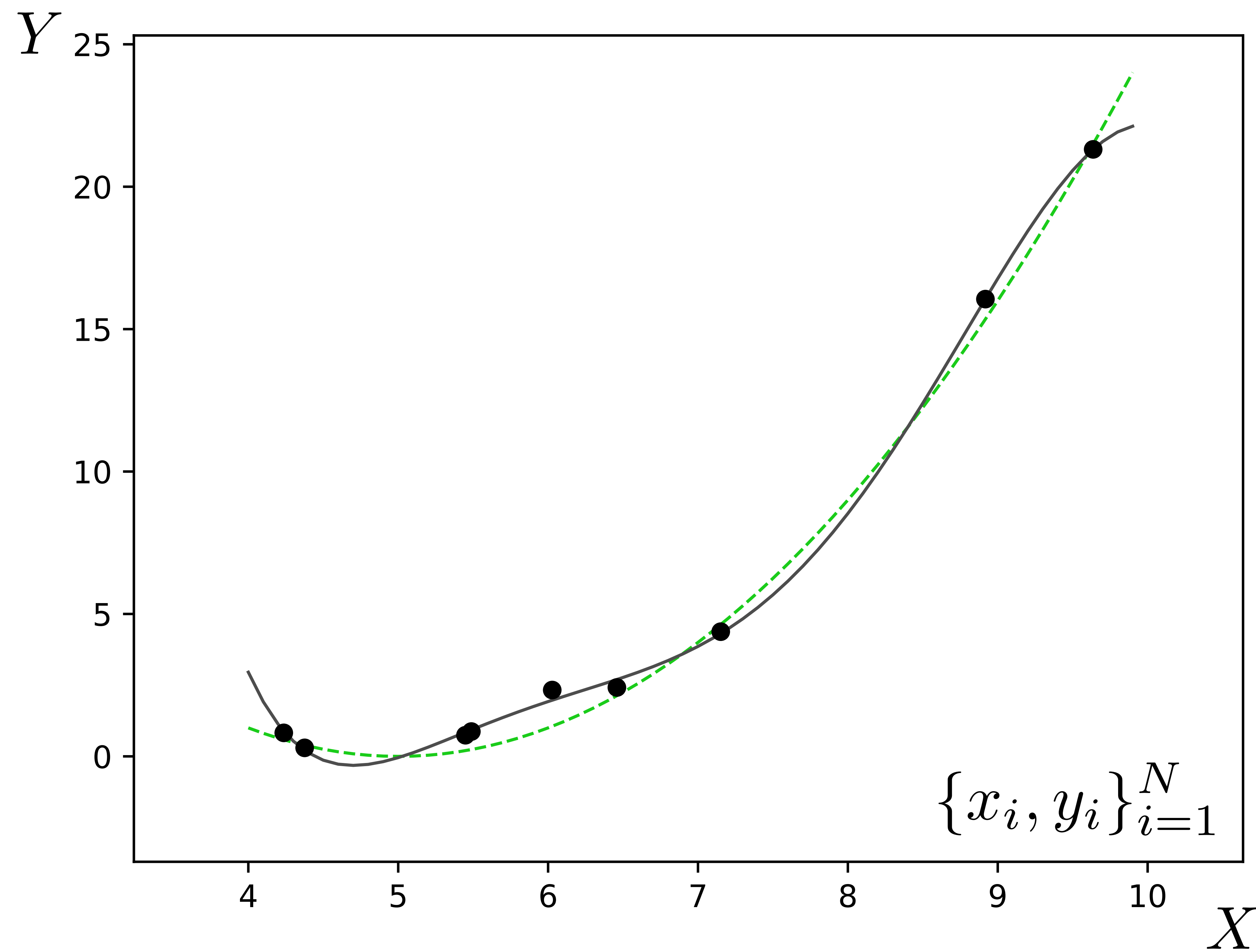
$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

What happens as we add more basis functions?



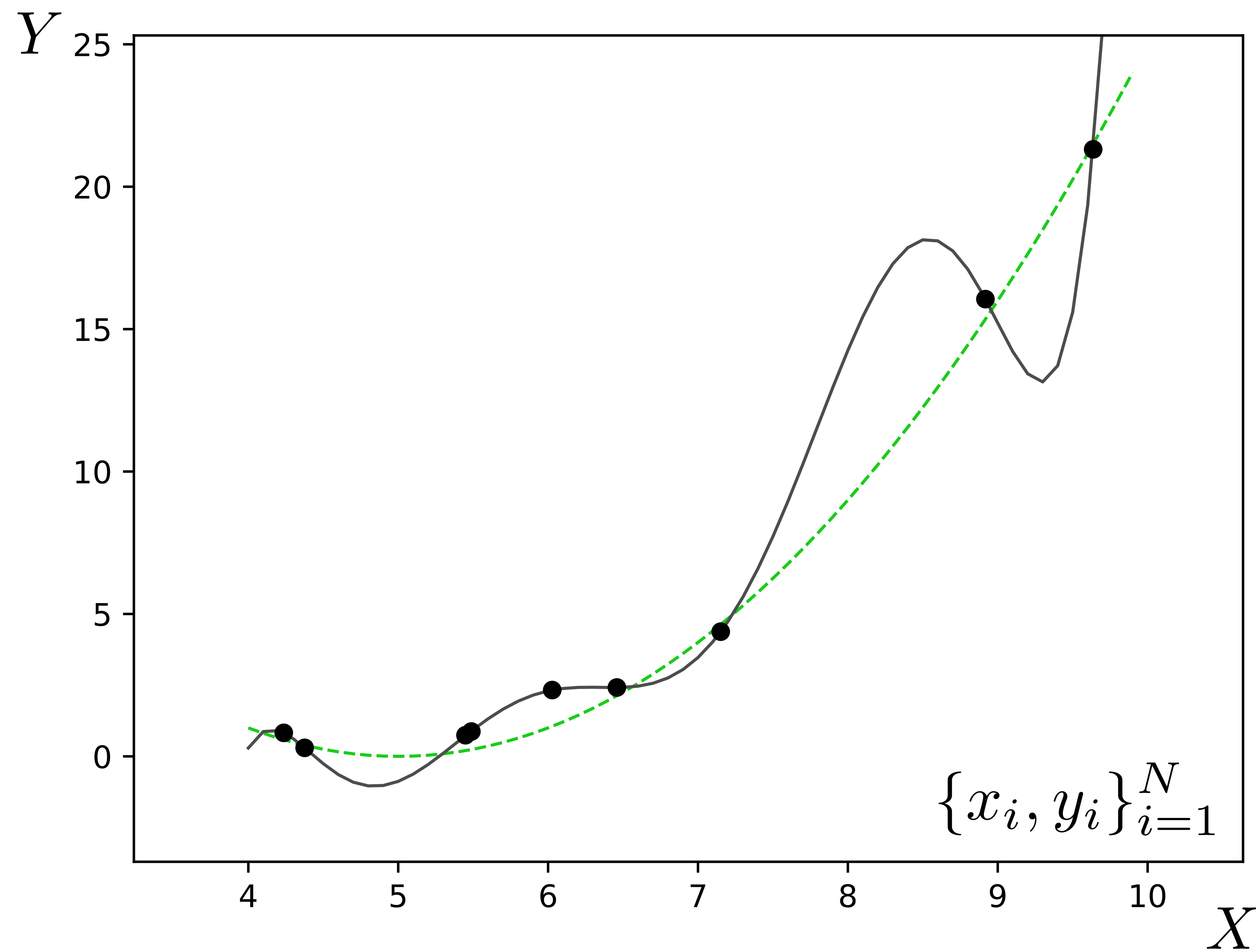
$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

What happens as we add more basis functions?



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

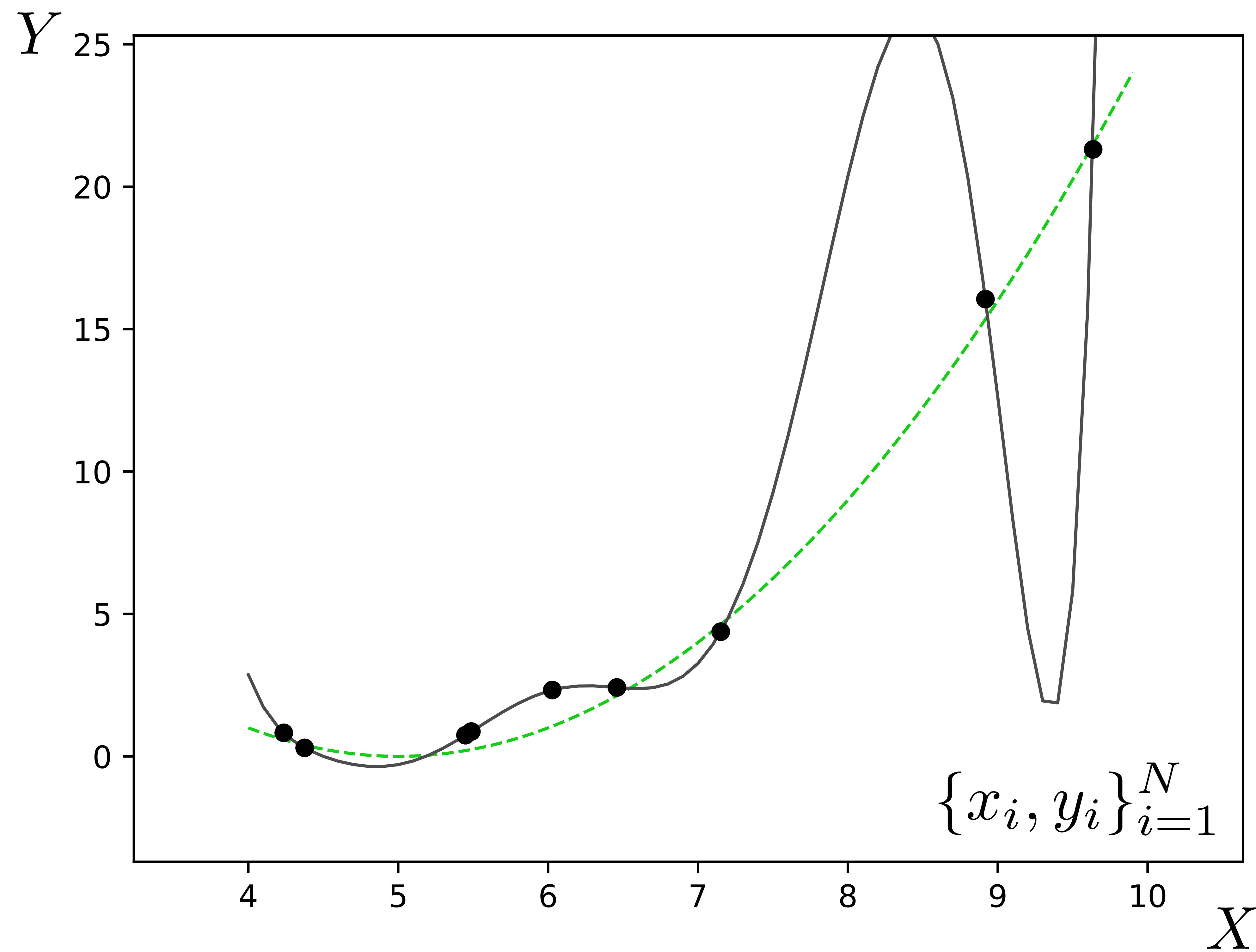
What happens as we add more basis functions?



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

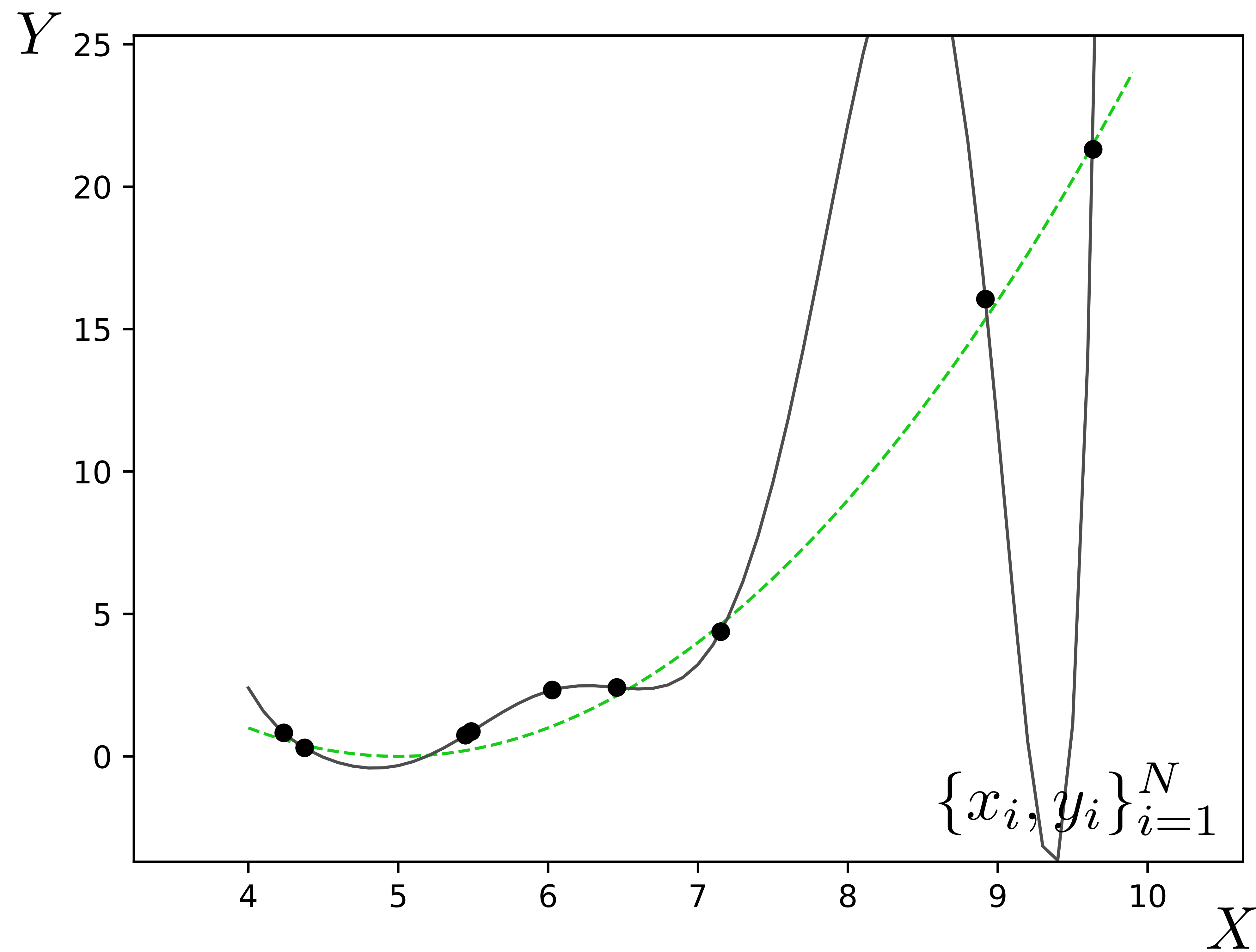
What happens as we add more basis functions?

K = 8



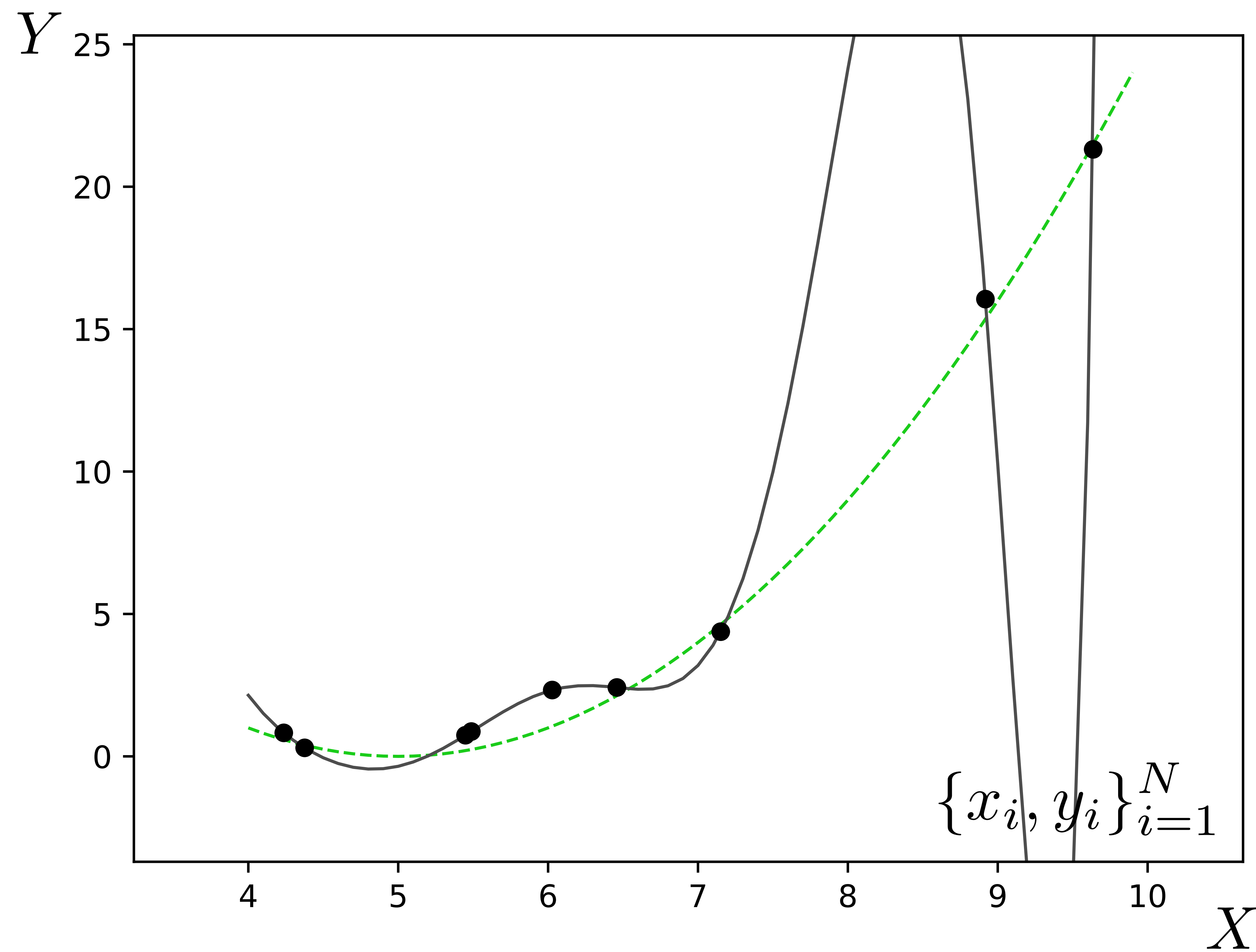
$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

What happens as we add more basis functions?



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

What happens as we add more basis functions?

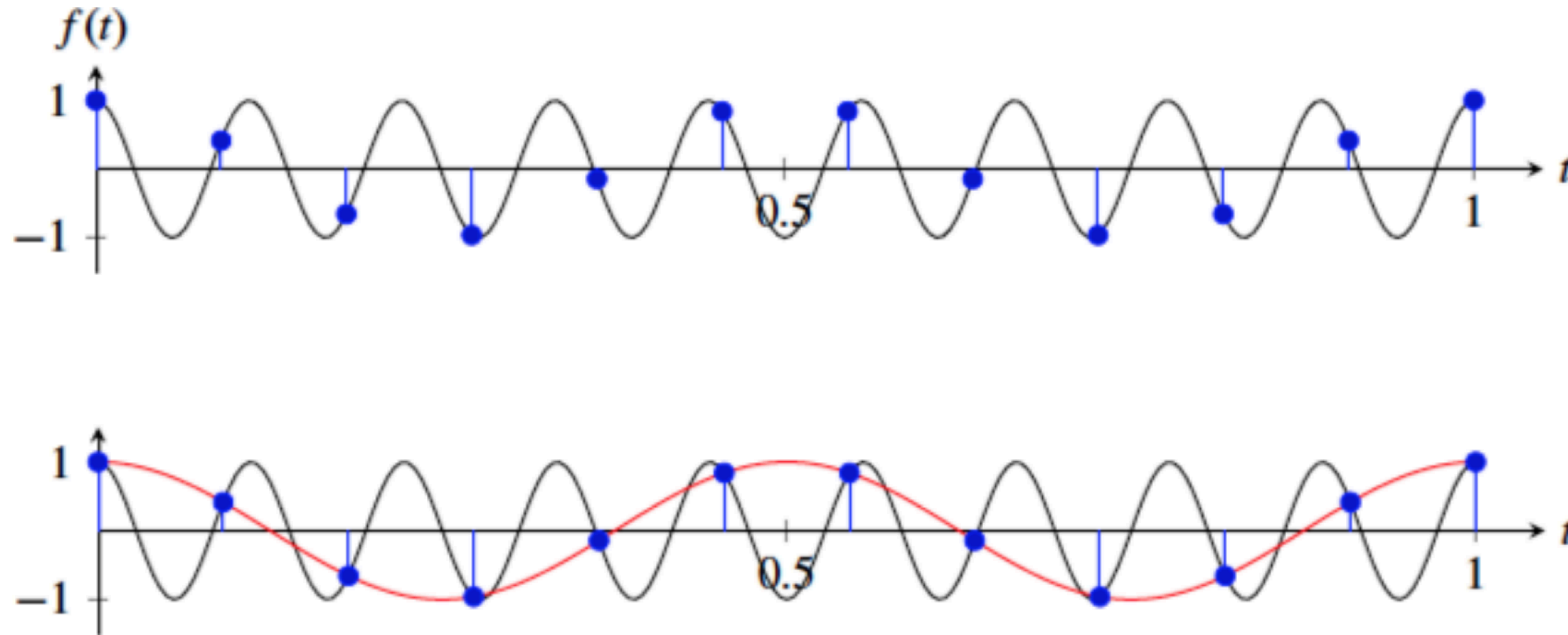


$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

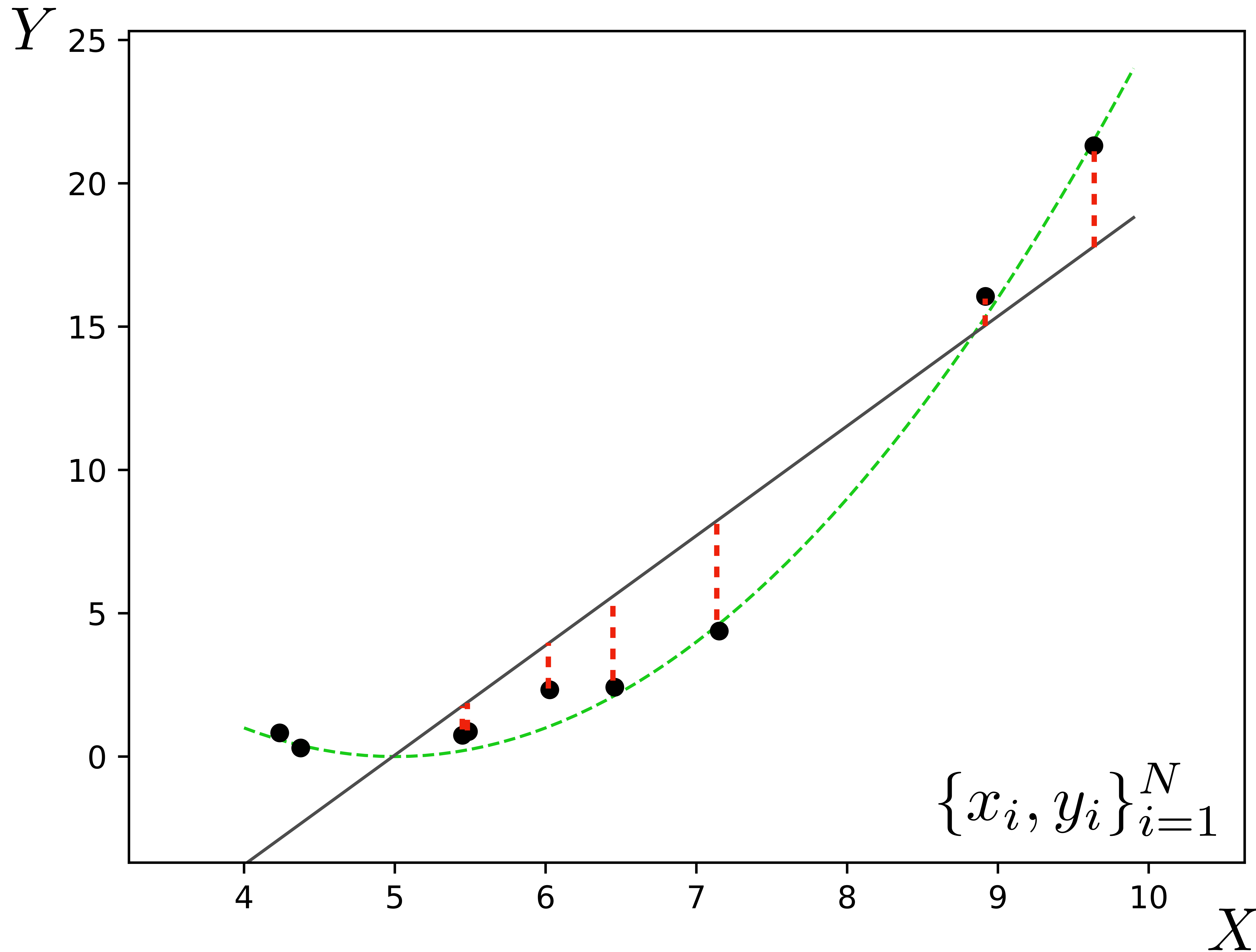
This phenomenon is called **overfitting**.

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.

Aliasing

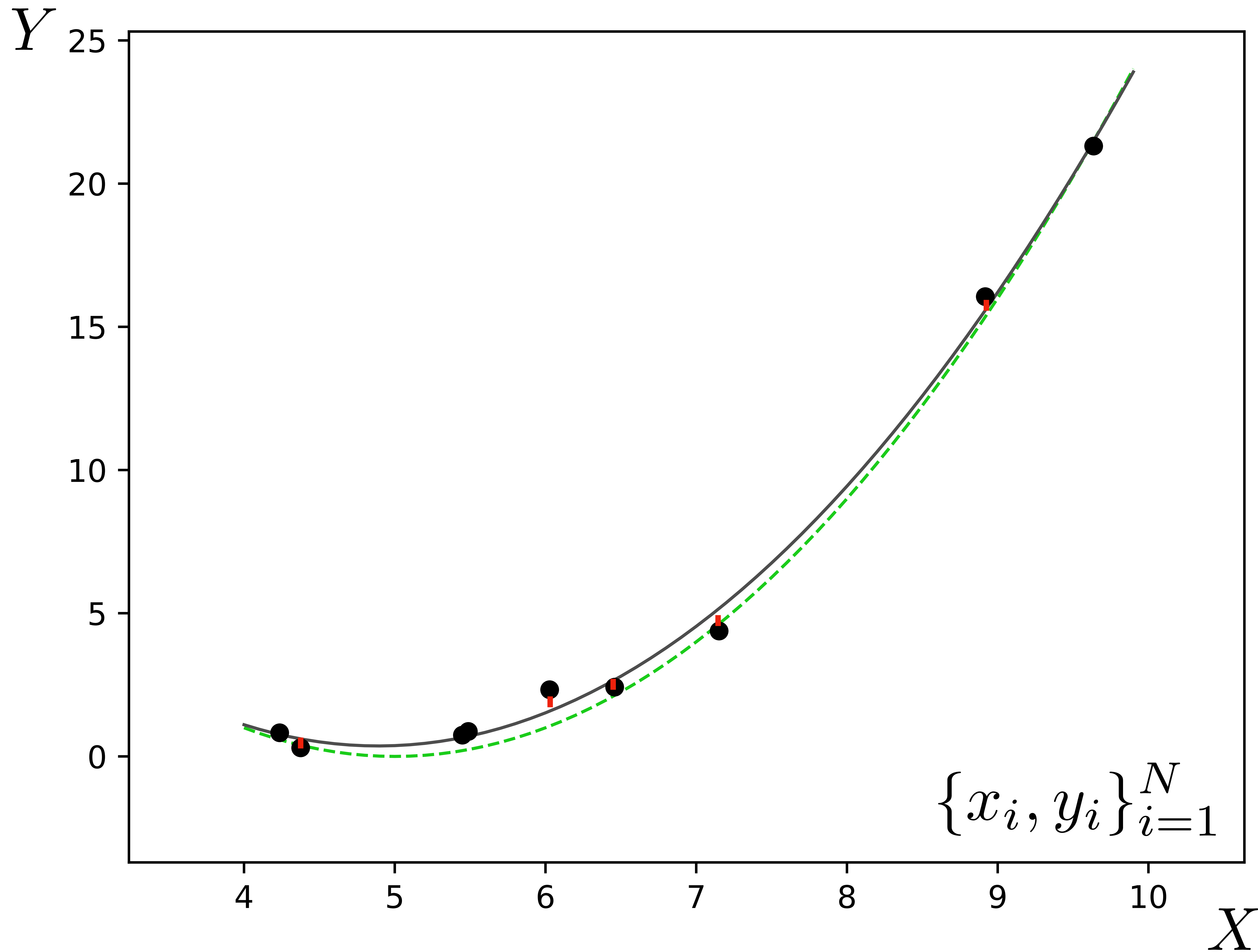


Both waves fit the same samples. Aliasing consists in "perceiving" the red wave when the actual input was the blue wave.

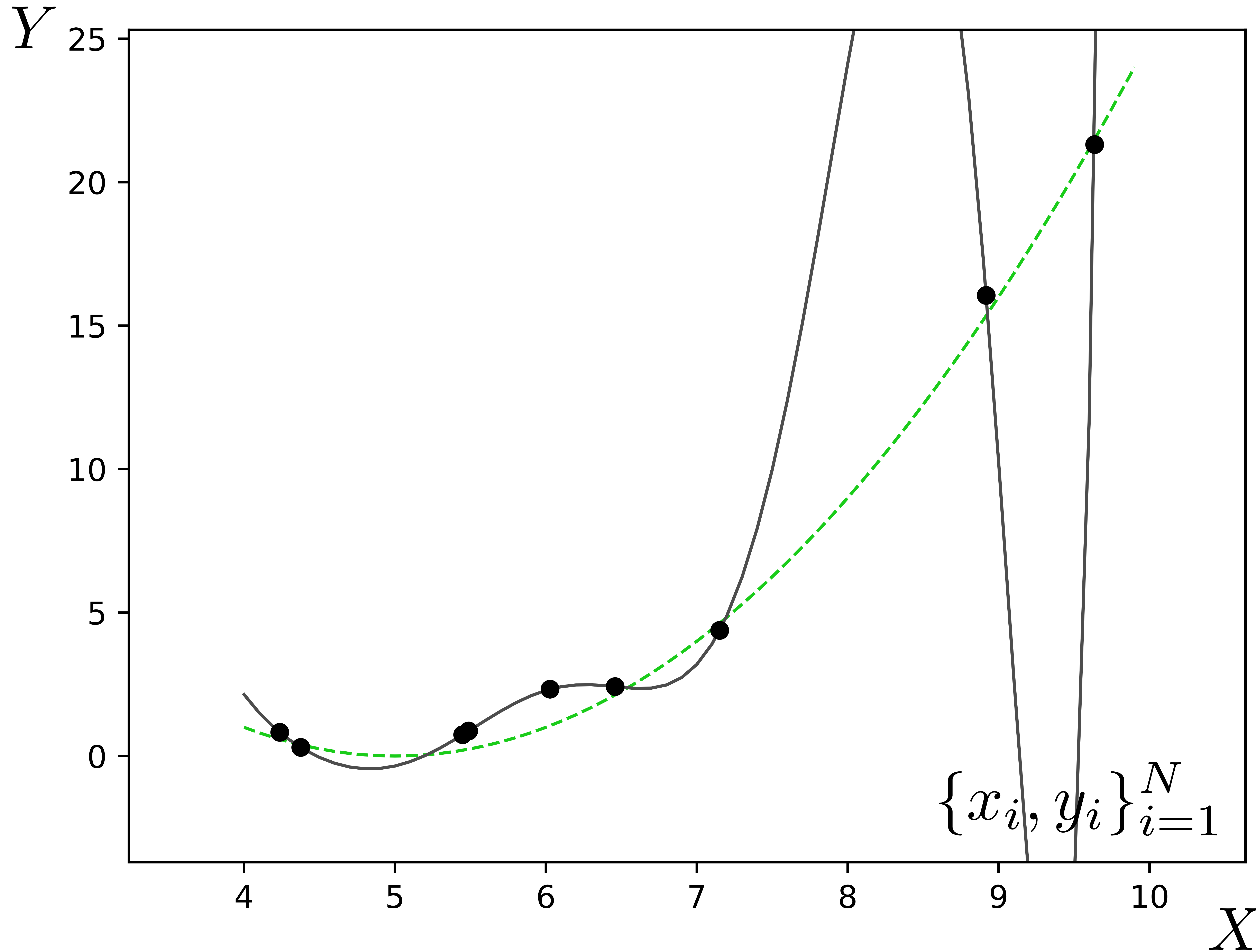


When the model does not have the capacity to capture the true function, we call this **underfitting**.

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



The true function is a quadractic, so a quadractic model (K=2) fits quite well.

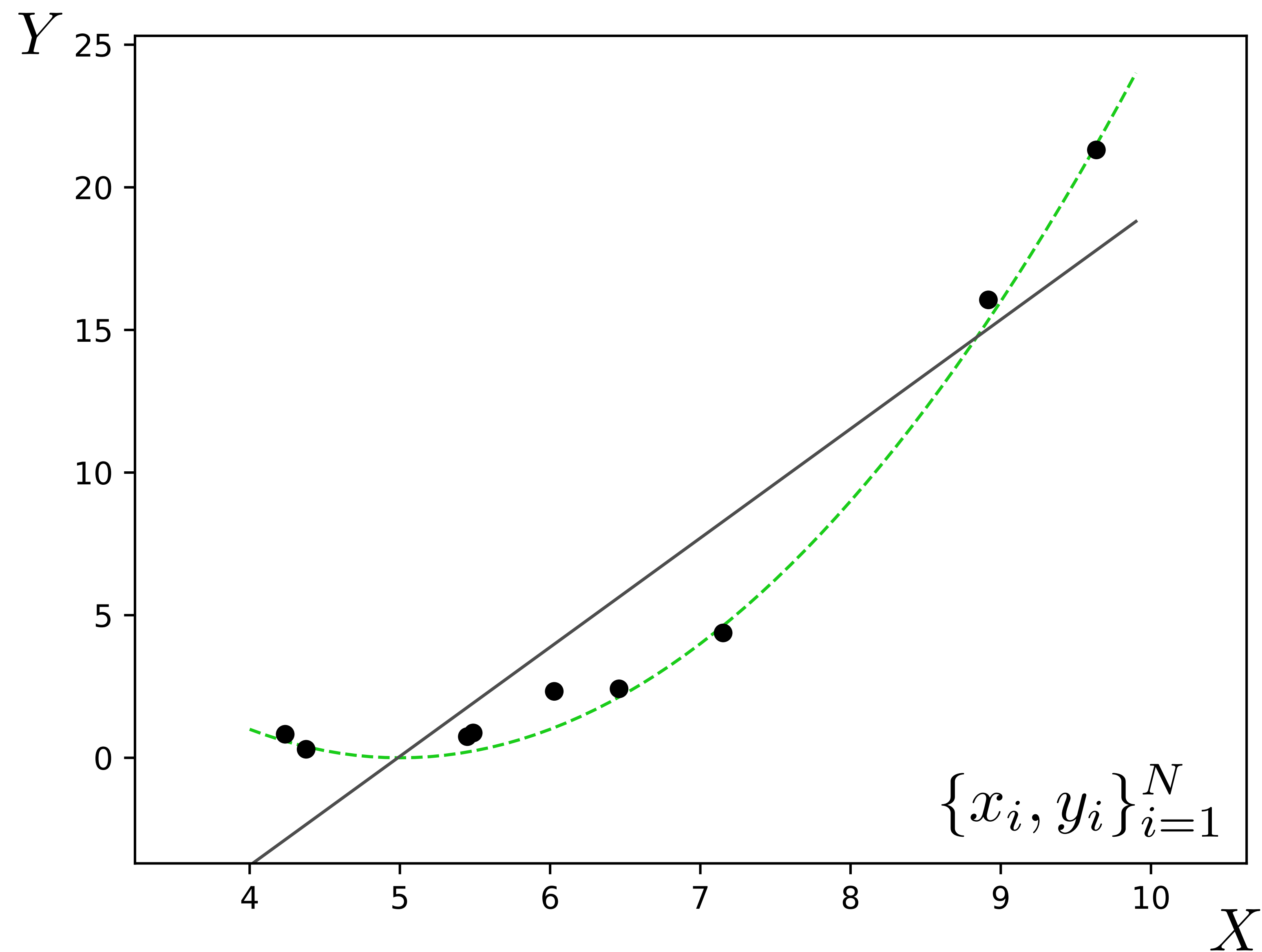


Now we have zero approximation error — the curve passes exactly through each training point.

But we have high **generalization error**, reflected in the gap between the true function and the fit line. We want to do well on *novel* queries, which will be sampled from the green curve (plus noise).

Underfitting

K = 1

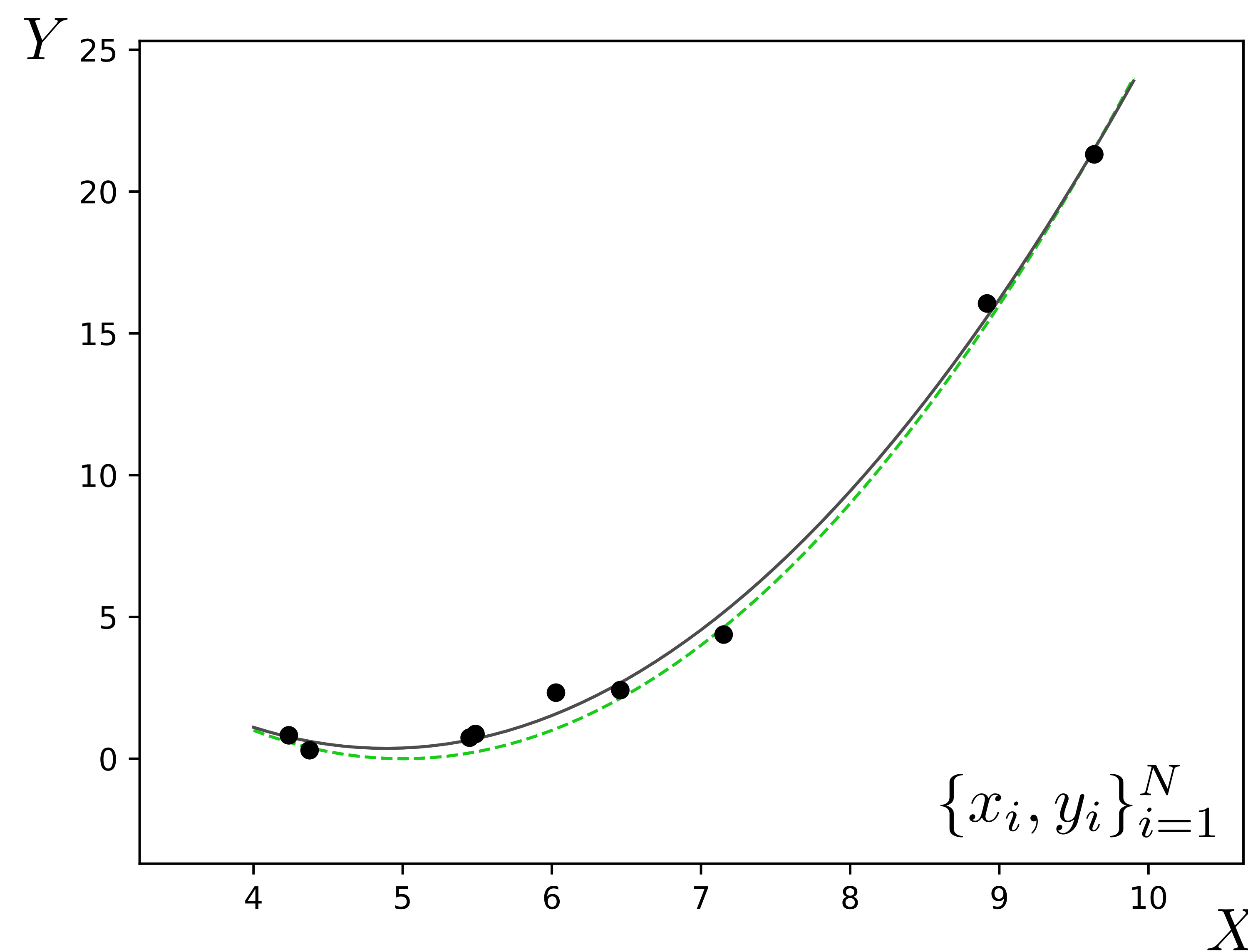


High error on train set

High error on test set

Appropriate model

K = 2

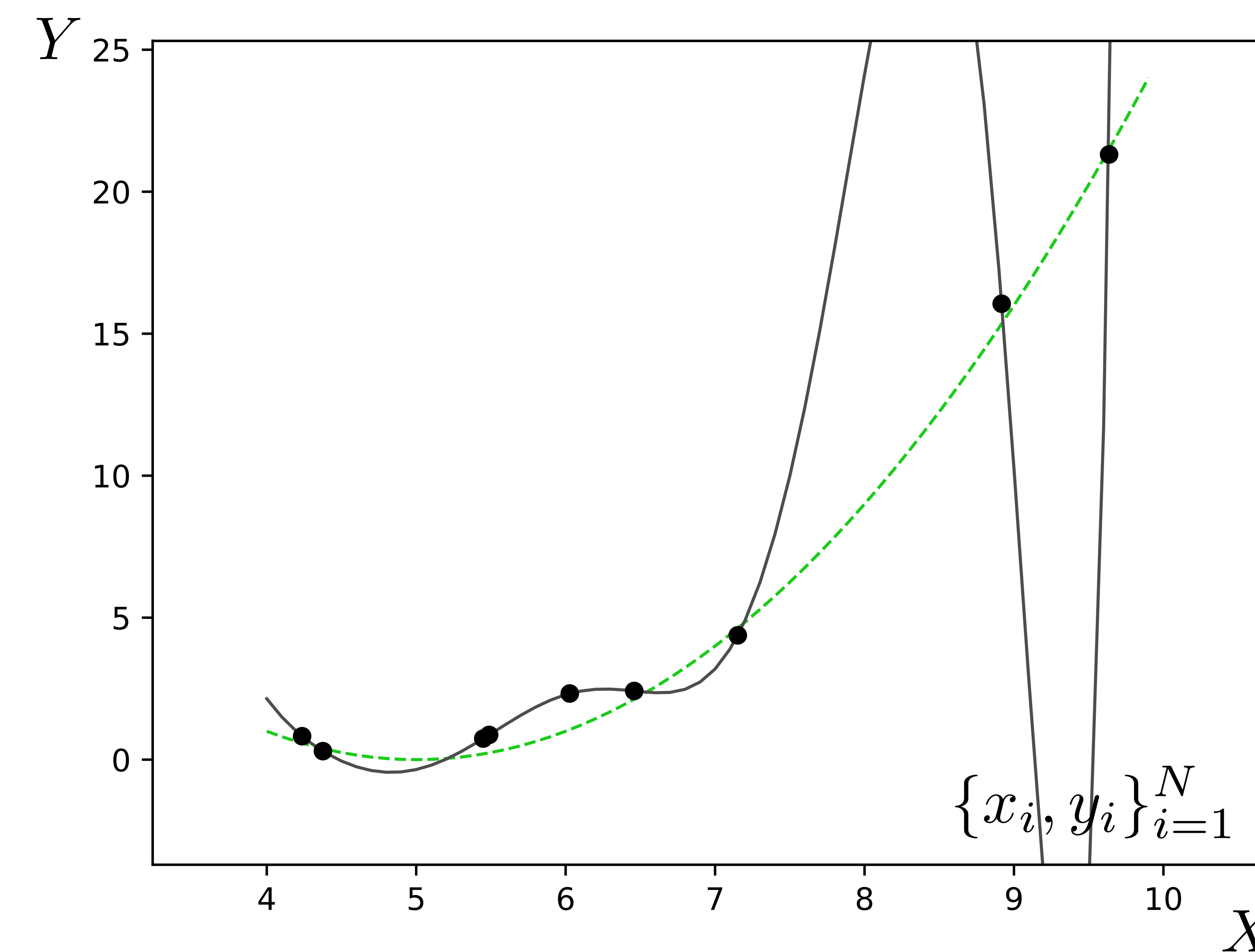


Low error on train set

Low error on test set

Overfitting

K = 10

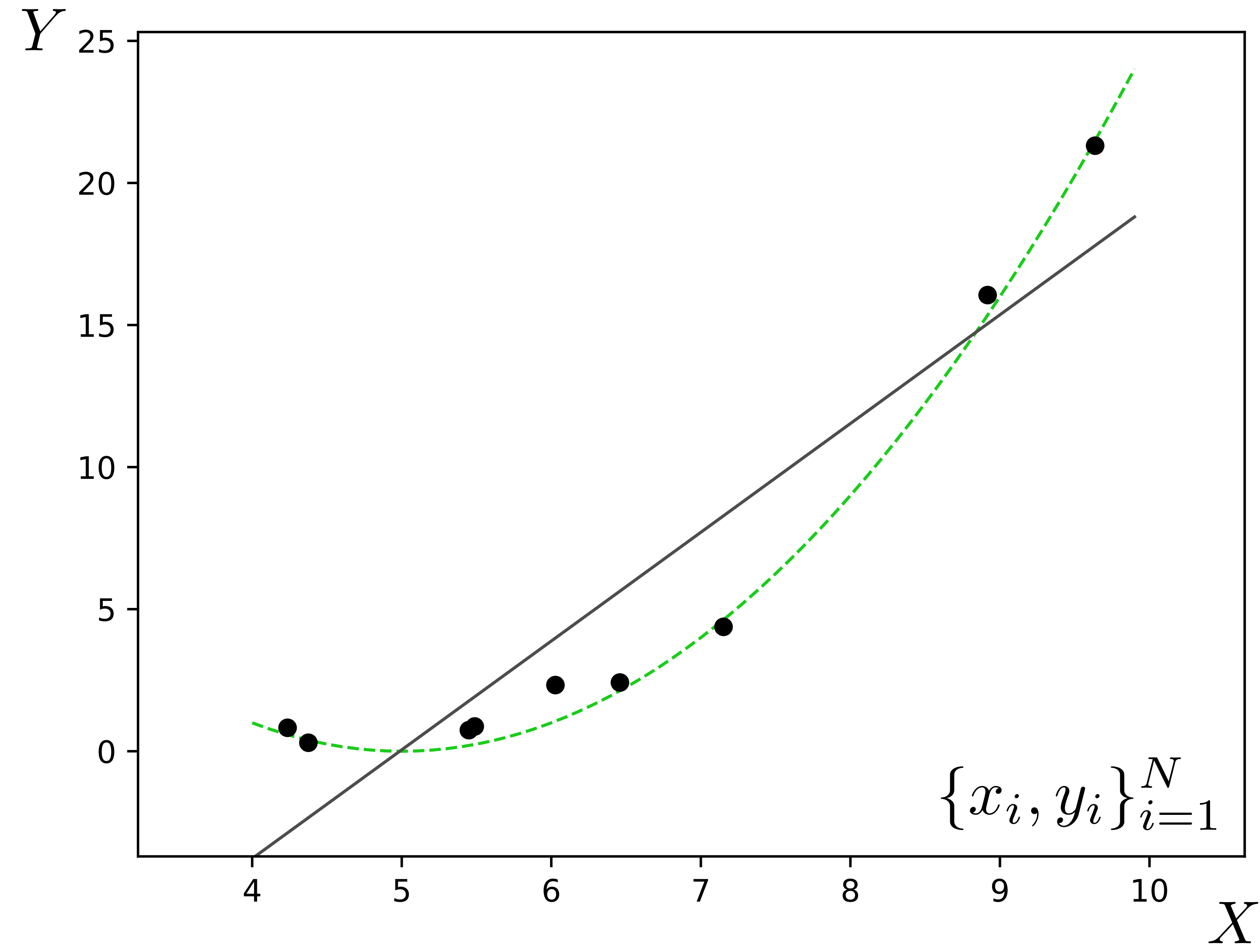


Lowest error on train set

High error on test set

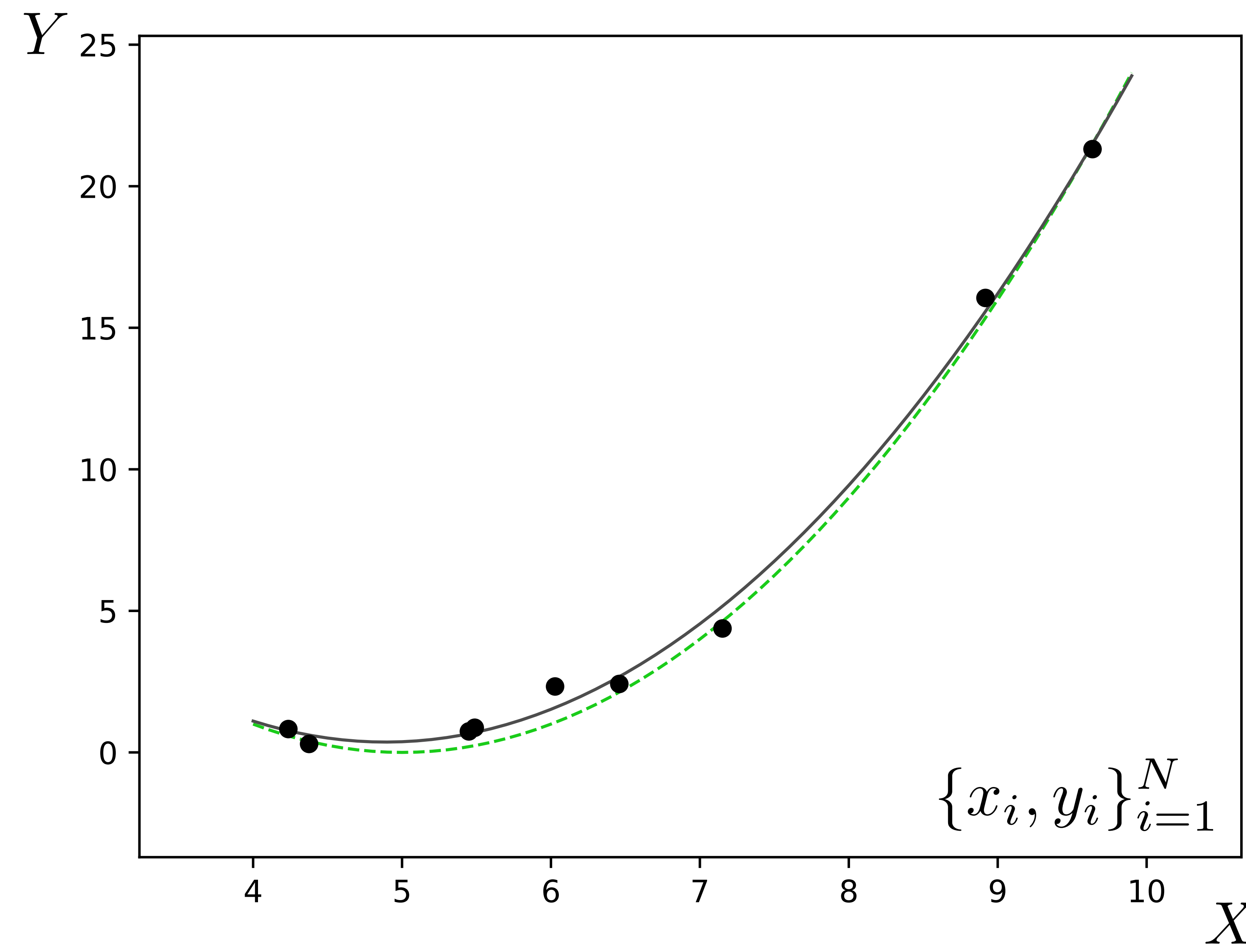
Underfitting

K = 1



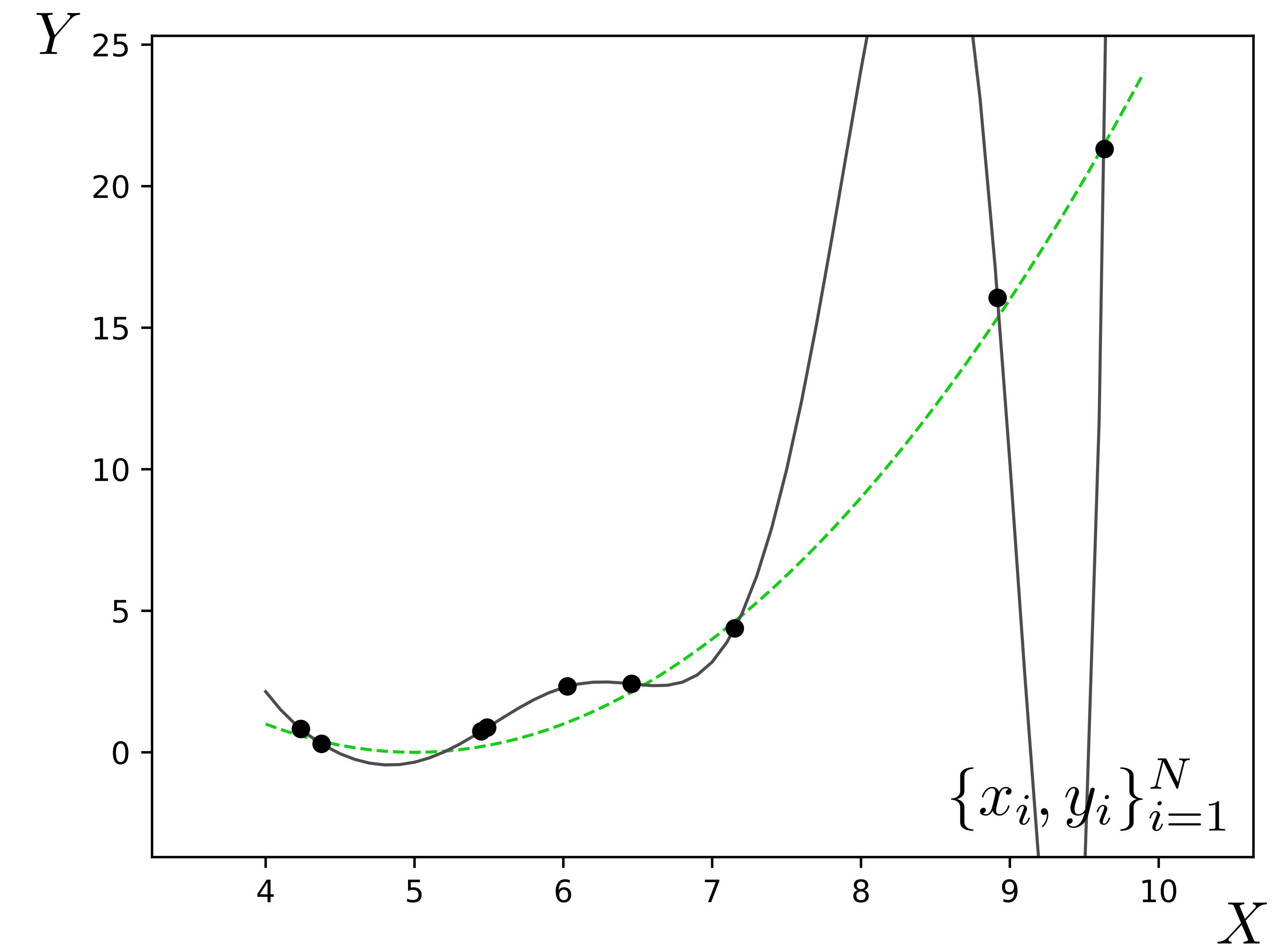
Appropriate model

K = 2



Overfitting

K = 10



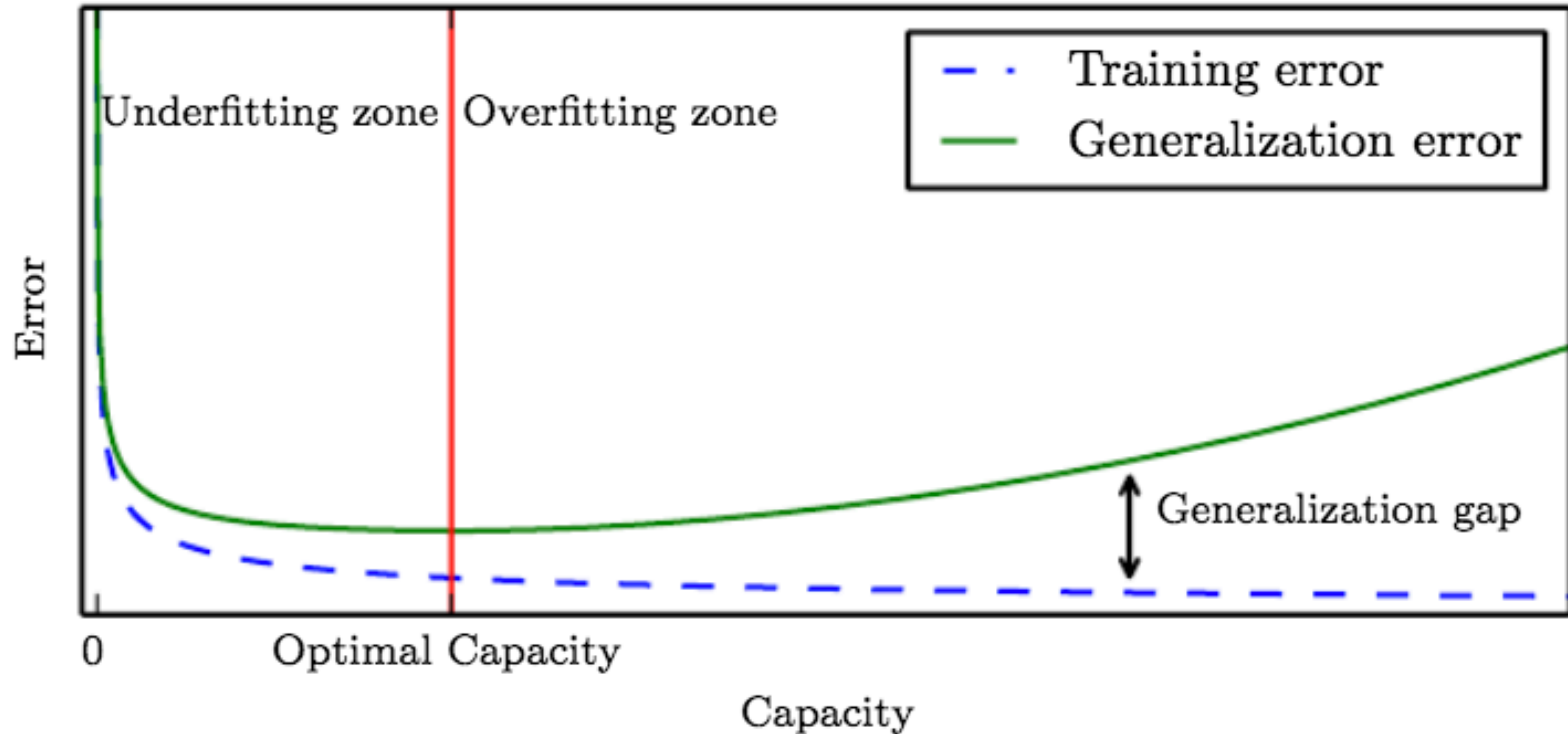
We need to control the **capacity** of the model (e.g., use the appropriate number of free parameters).

The capacity may be defined as the number of hypotheses under consideration in the hypothesis space.

Complex models with many free parameters have high capacity.

Simple models have low capacity.

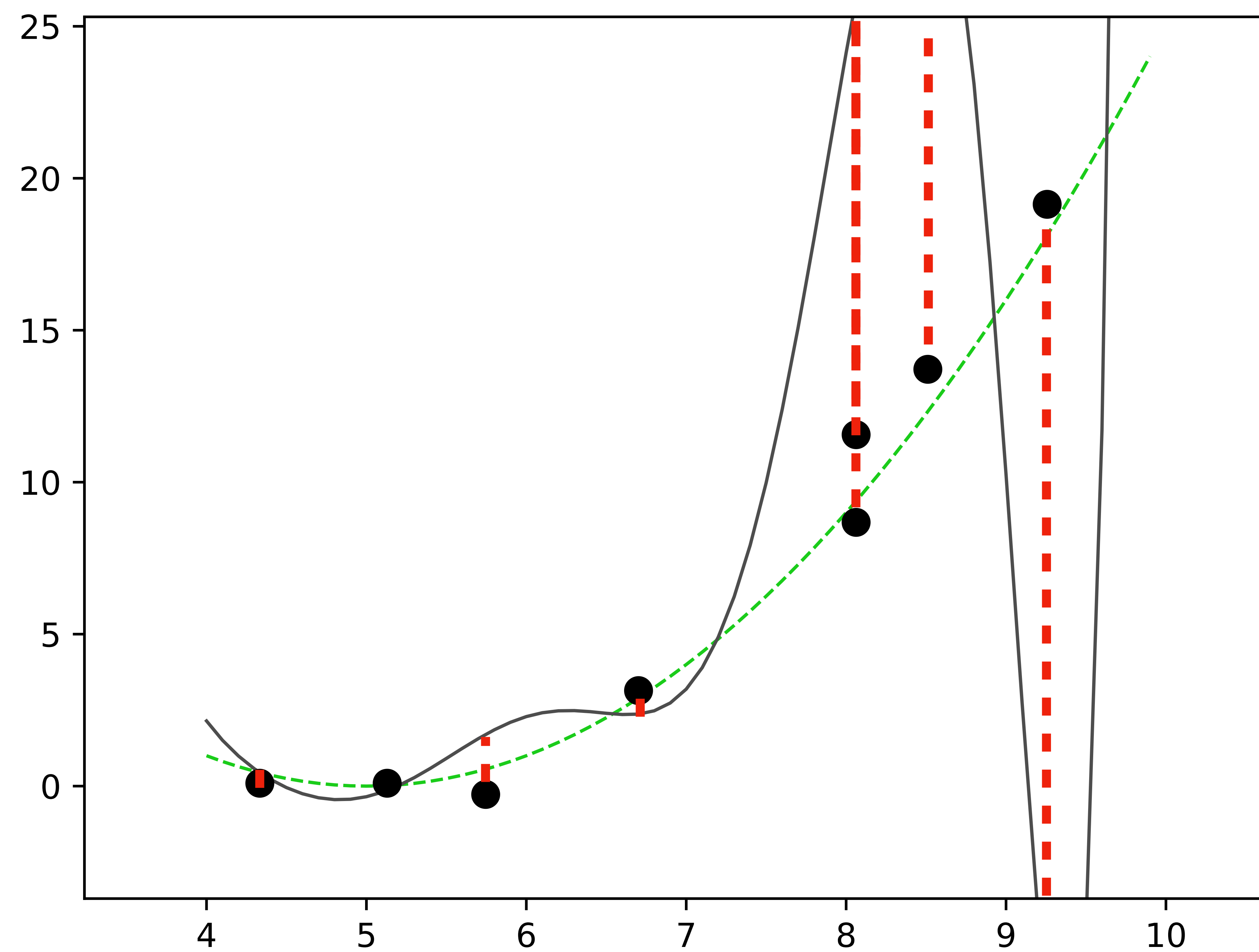
Training error versus generalization error

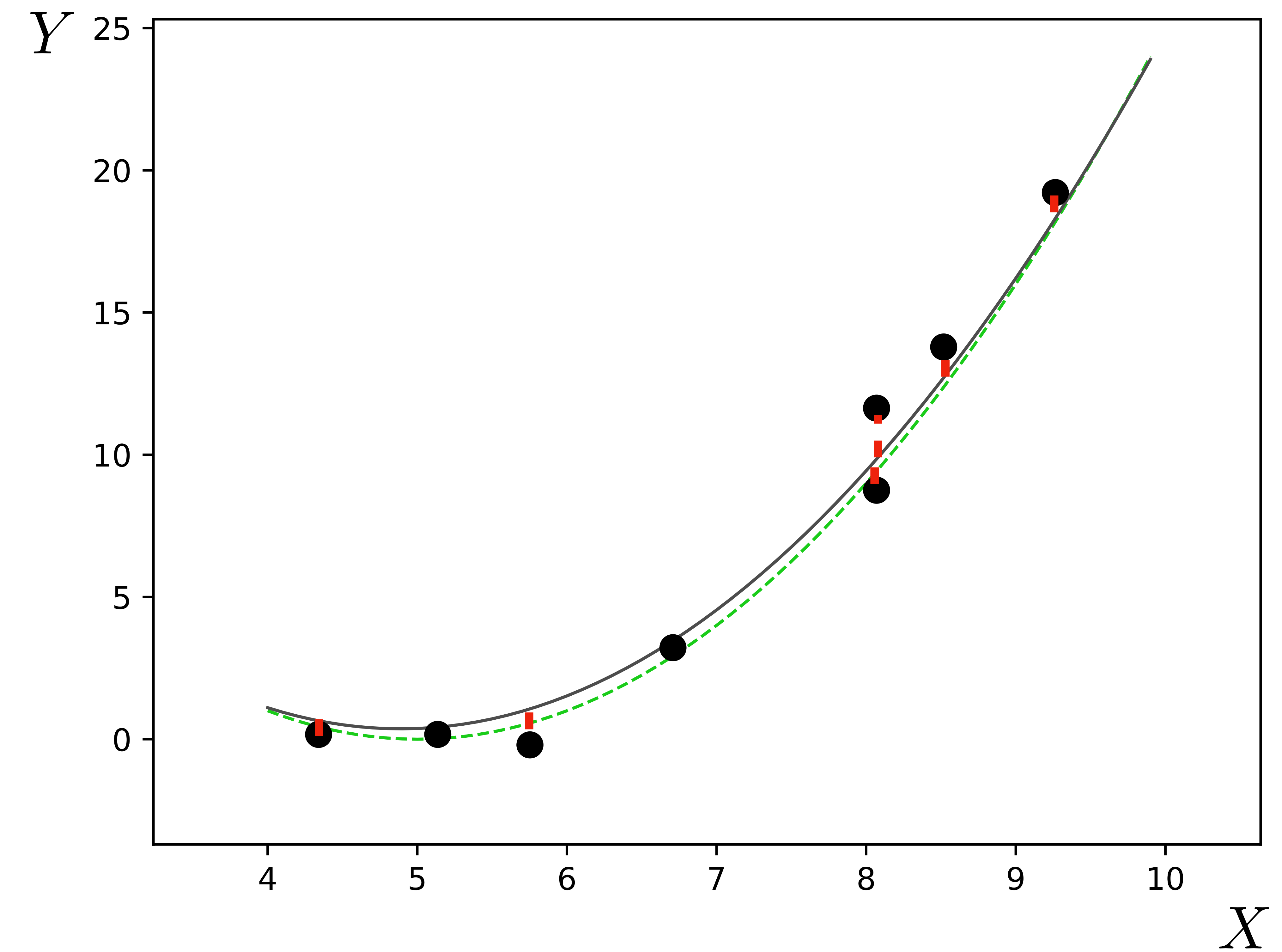


["Deep Learning", Goodfellow et al.]

How do we know if we are underfitting or overfitting?

Test data $\{x_i^{(\texttt{test})}, y_i^{(\texttt{test})}\}_{i=1}^{M}$





Cross validation: measure prediction error on test data

Fitting just right

Underfitting?

1. add more parameters (more features, more layers, etc.)

Overfitting?

1. remove parameters
2. add **regularizers**

Selecting a *hypothesis space* of functions with just the right capacity is known as **model selection**

Regularization

Emprirical risk minimization:

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

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

Regularized least squares

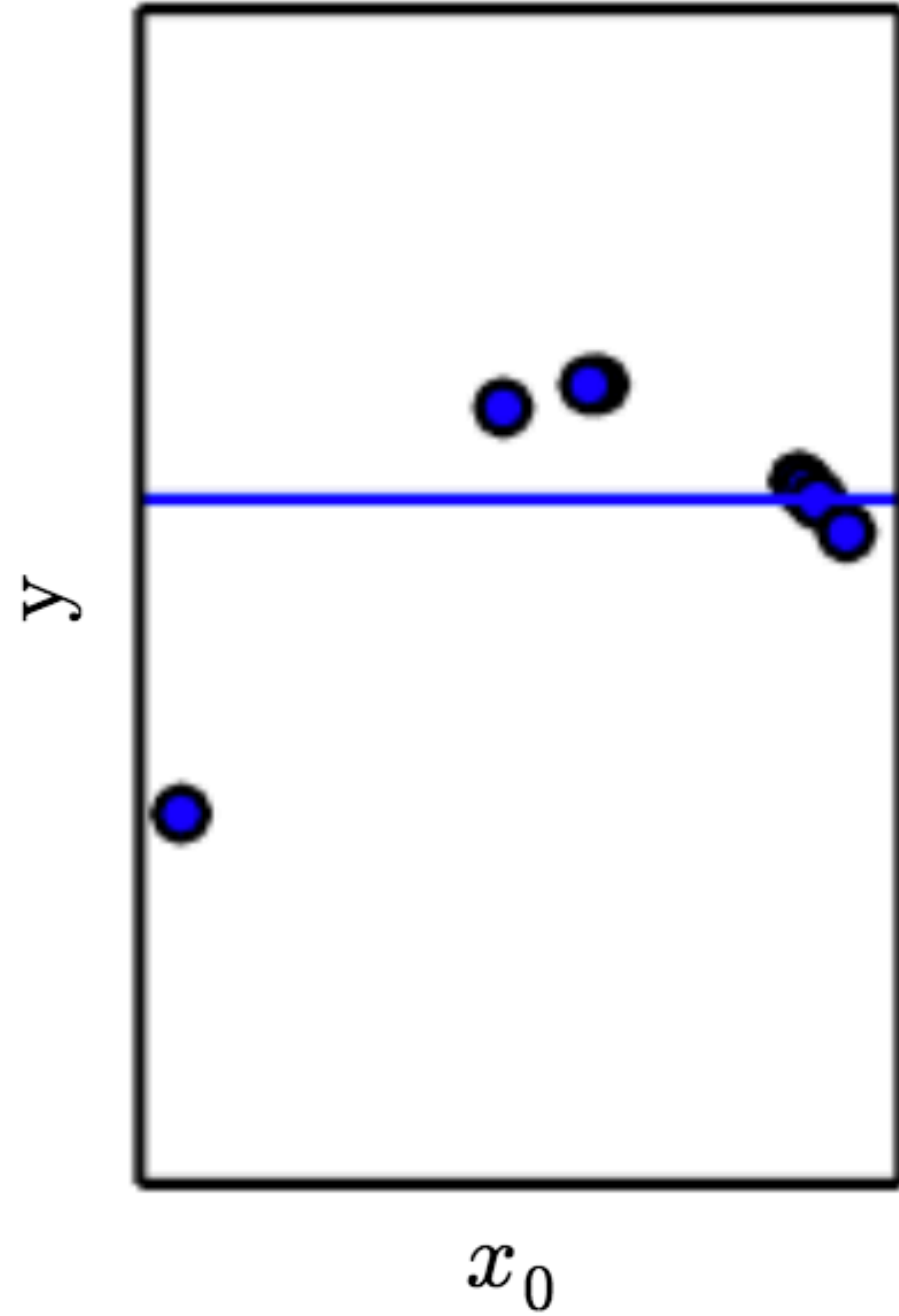
$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

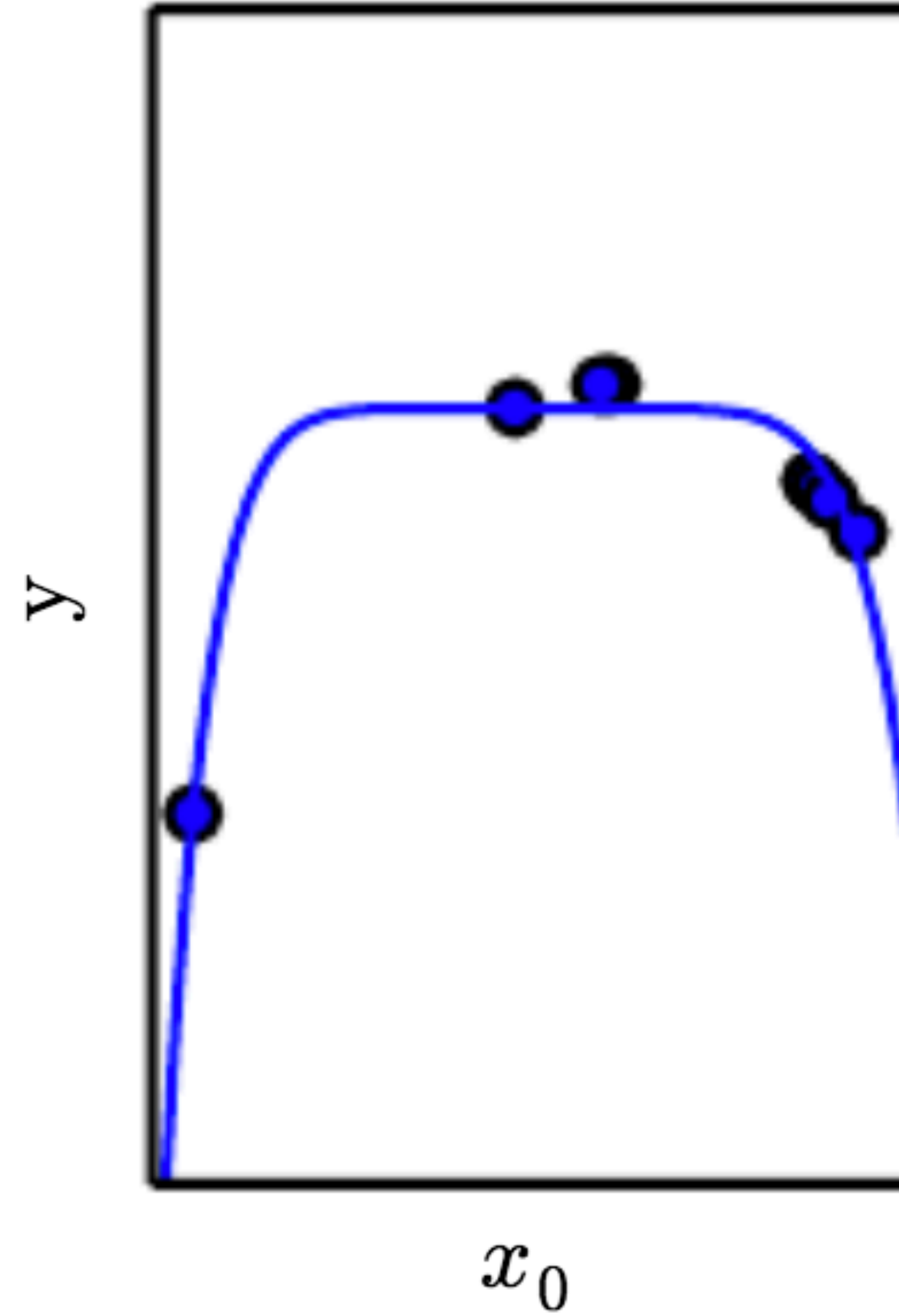
$$R(\theta) = \lambda \left\|\theta\right\|_2^2$$

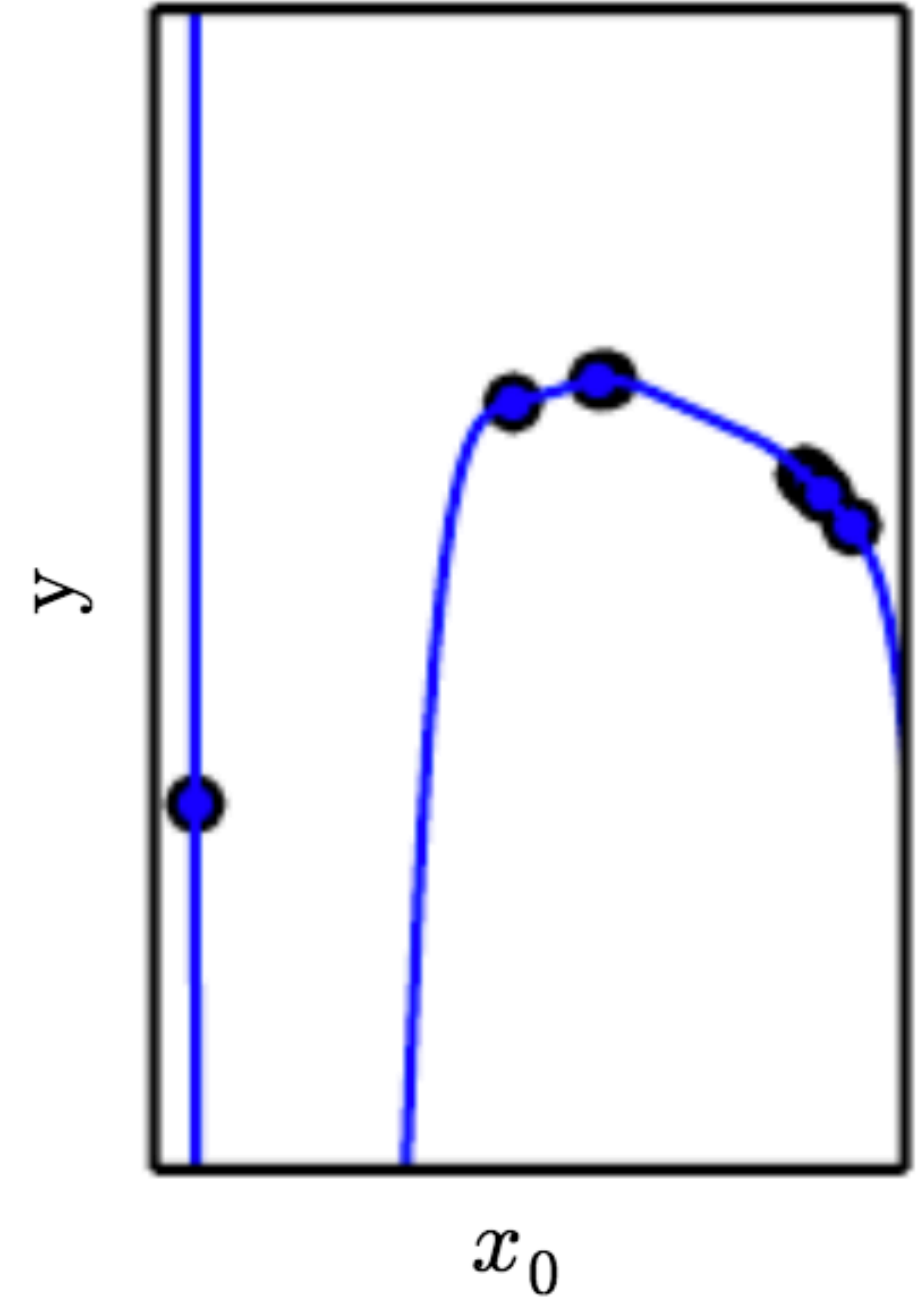
← Only use polynomial terms if you really need them! Most terms should be zero

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

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

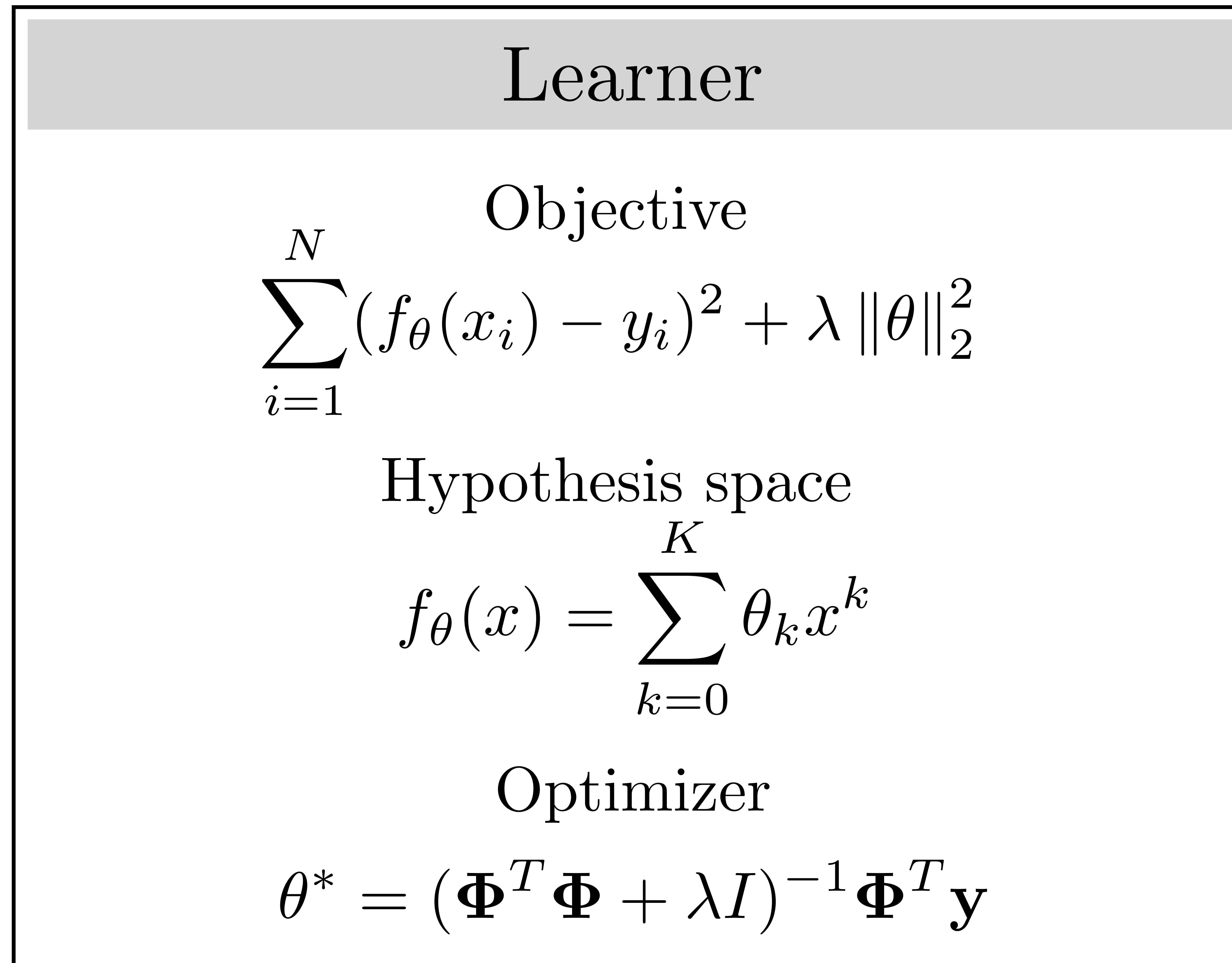




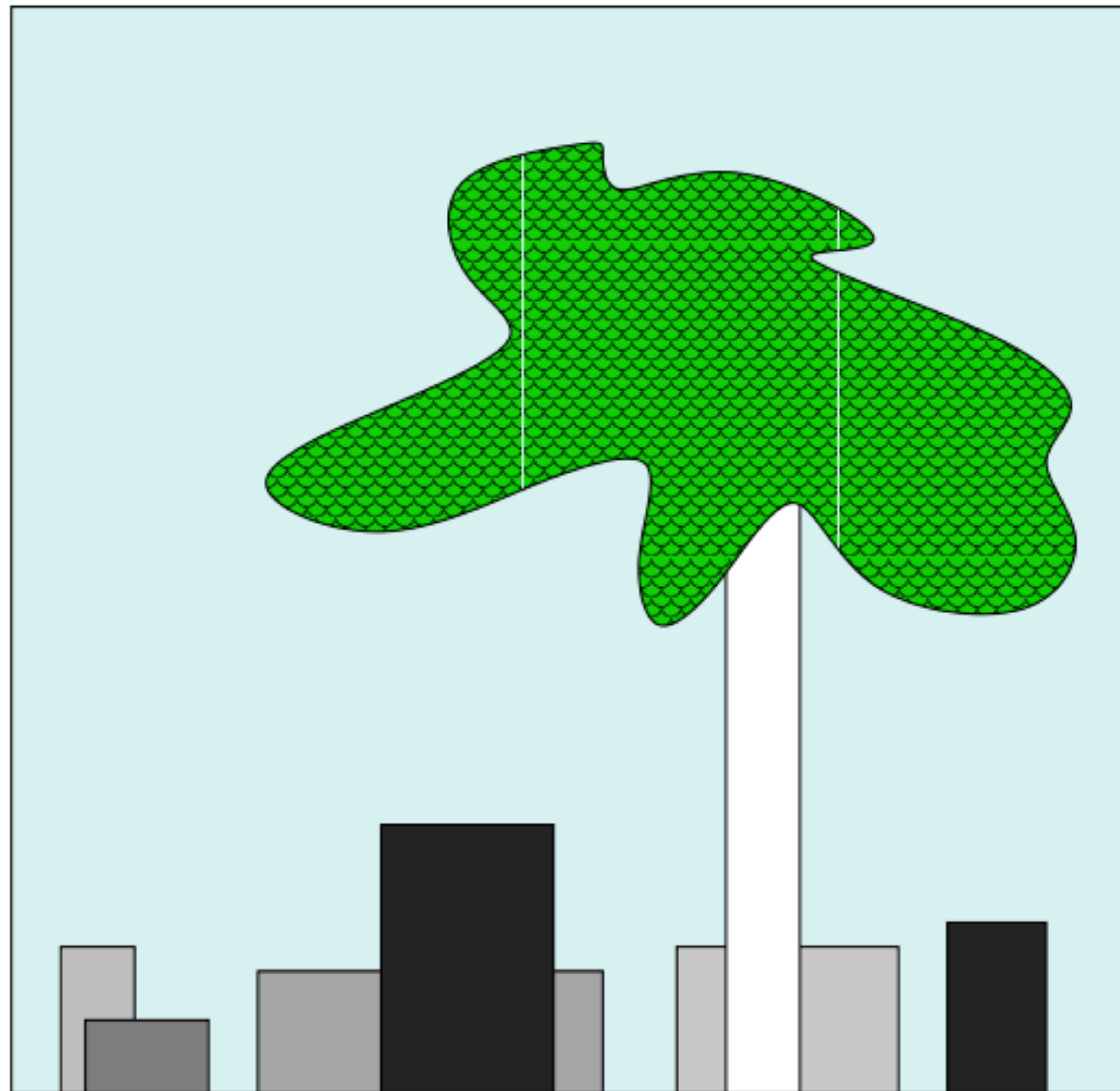


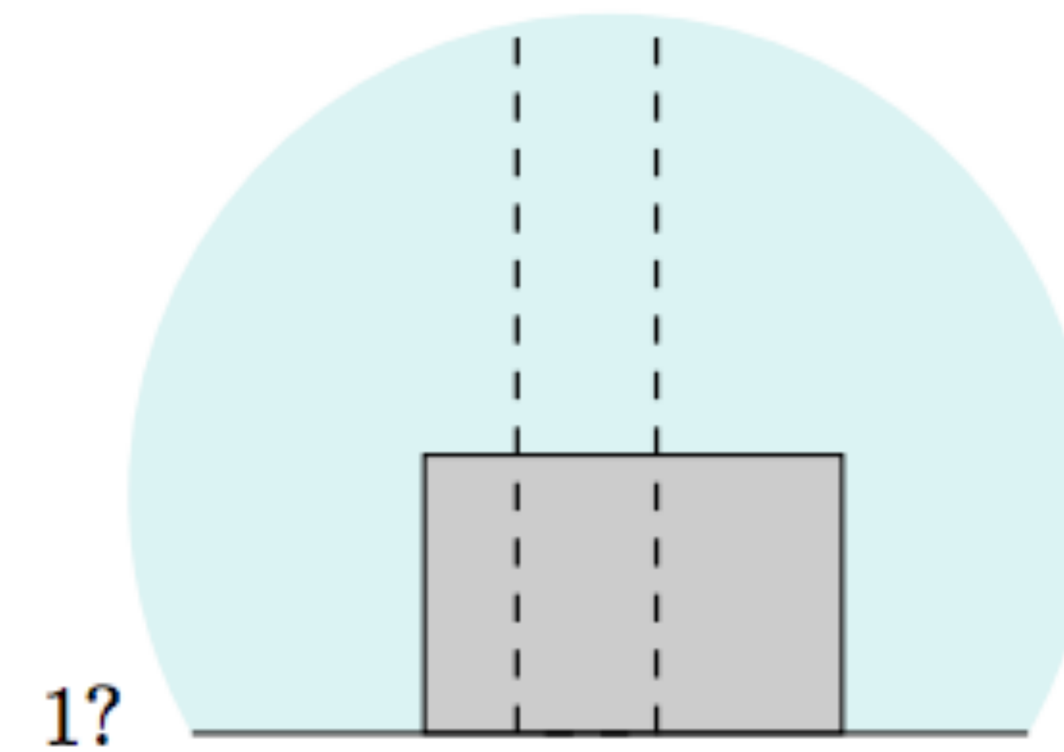
[“Deep Learning”, Goodfellow et al.]

Regularized polynomial least squares regression



Aside: what's the best prior? What's a principle for selecting it?





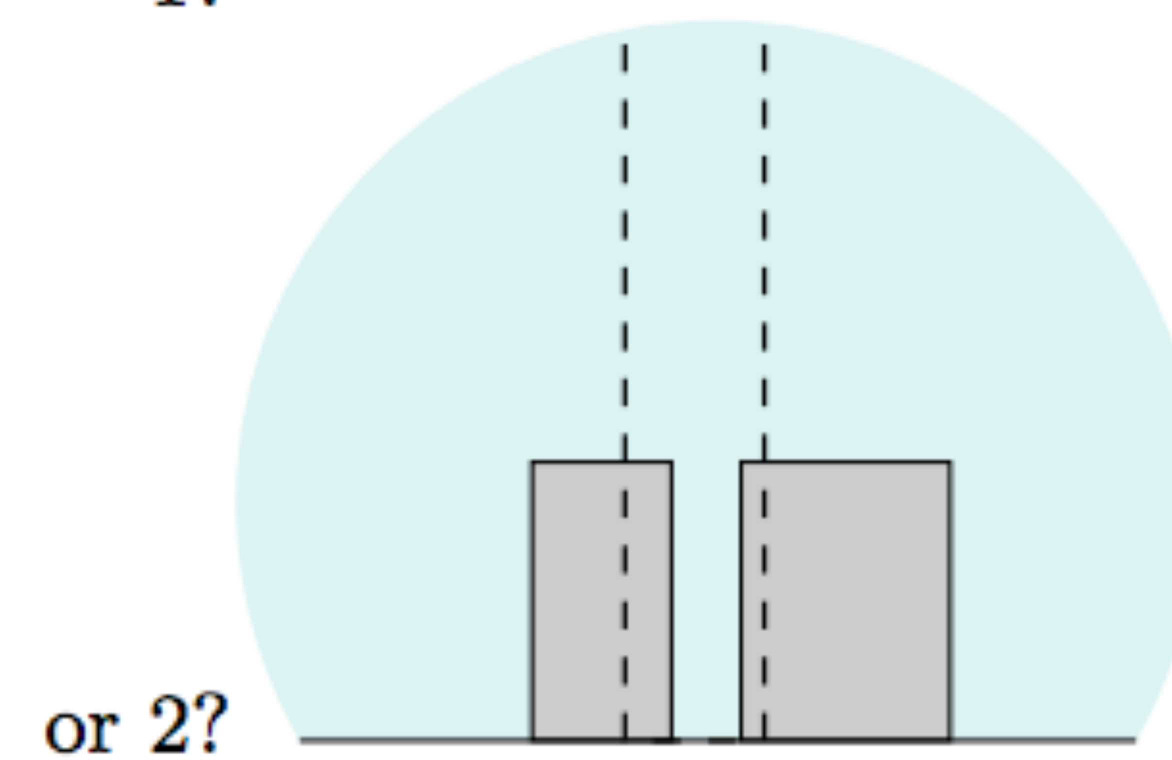
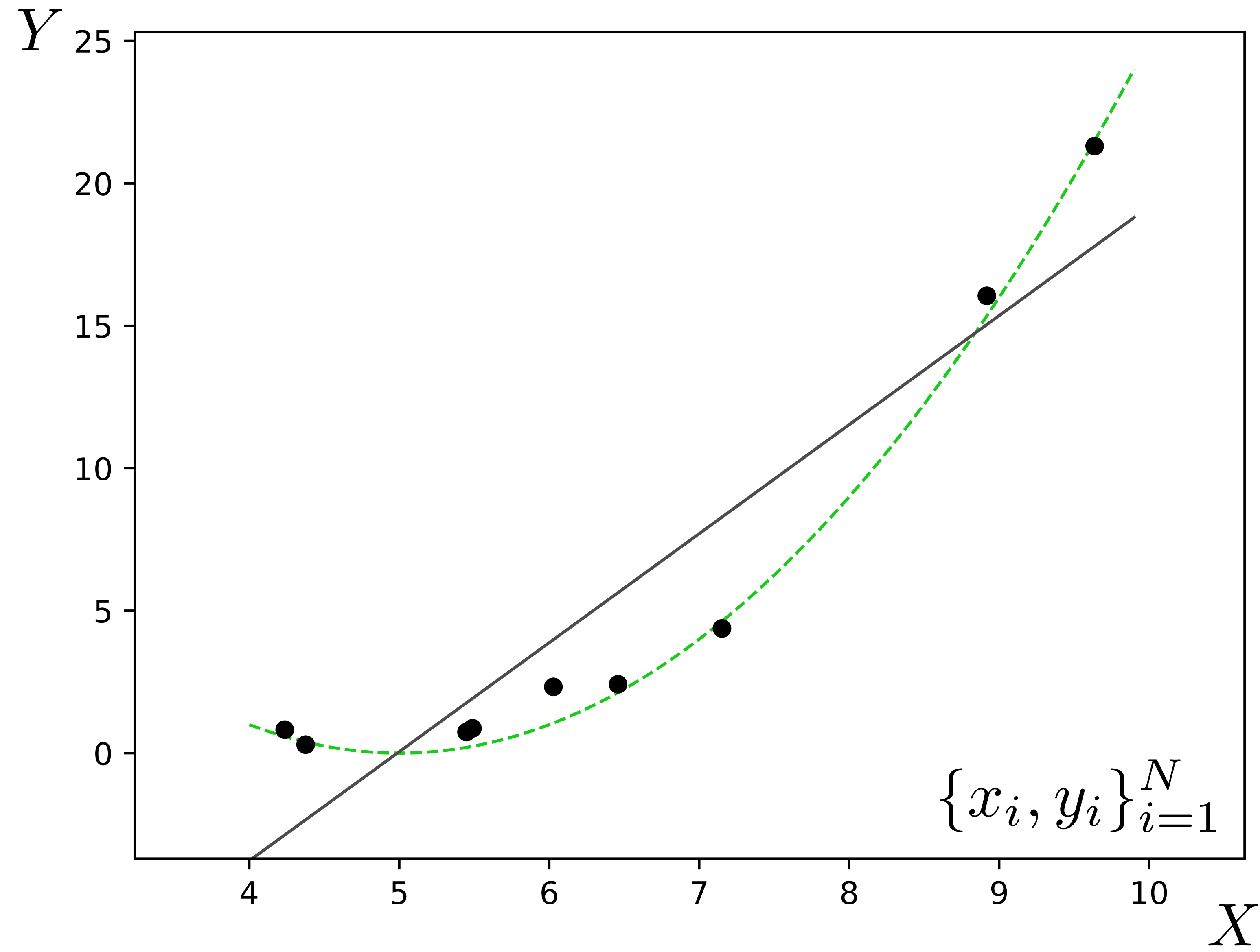


Figure 28.2. How many boxes are behind the tree?

[ITILA book, Chapter 28, MacKay, http://www.inference.org.uk/itprnn/book.pdf]

Underfitting

K = 1

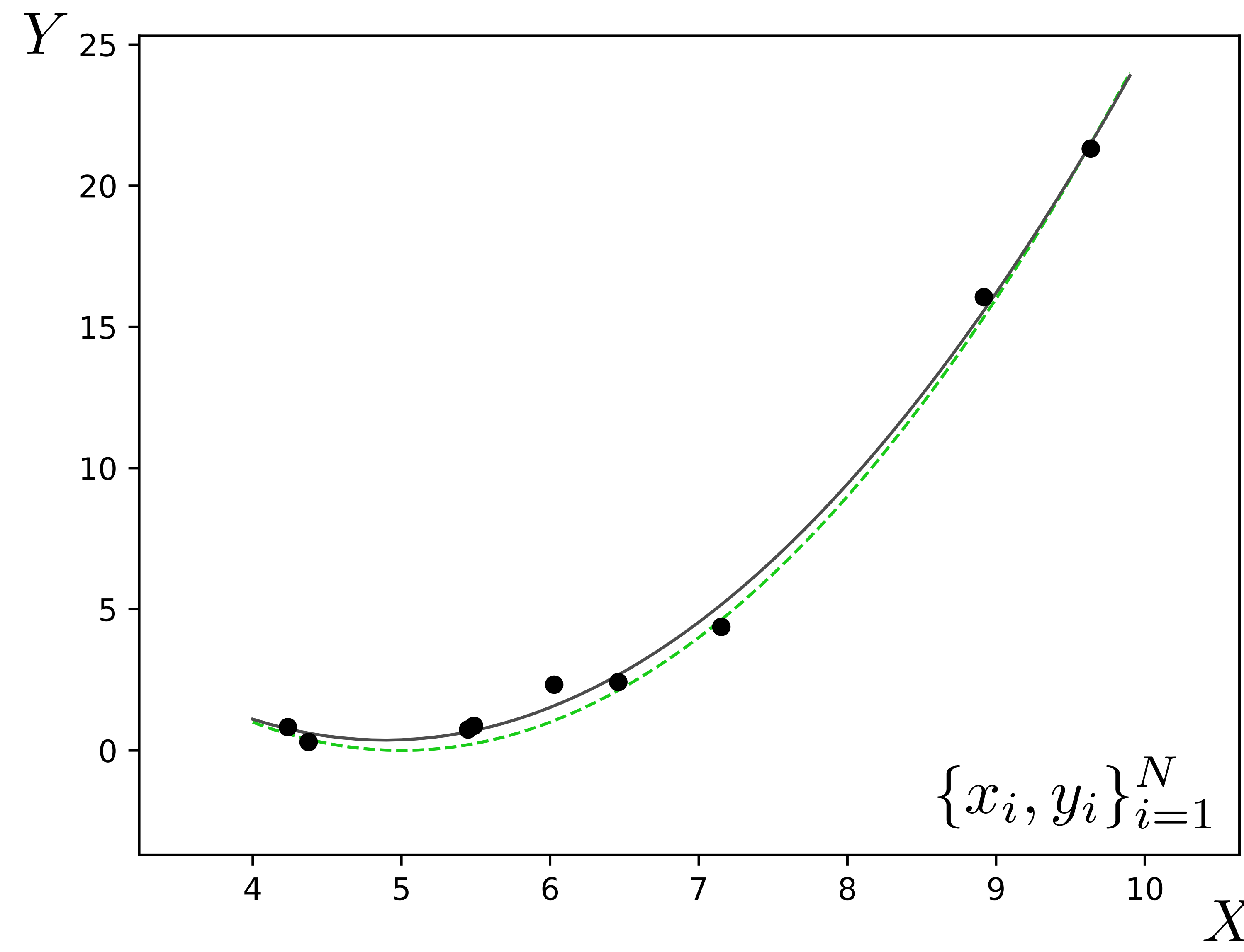


Simple model

Doesn’t fit the training data

Appropriate model

K = 2

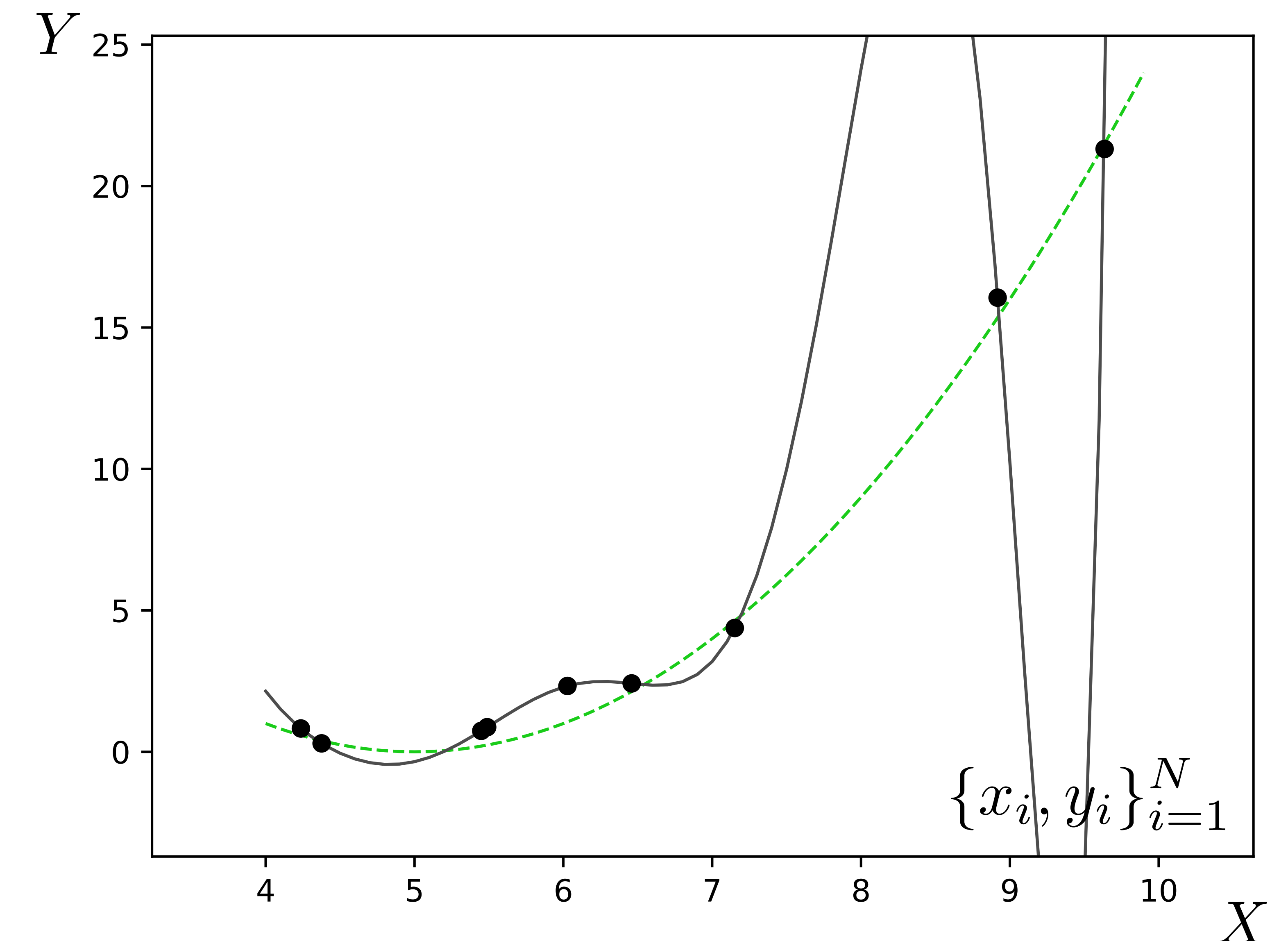


Simple model

Fits the training data

Overfitting

K = 10



Complex model

Fits the training data

Deep learning

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

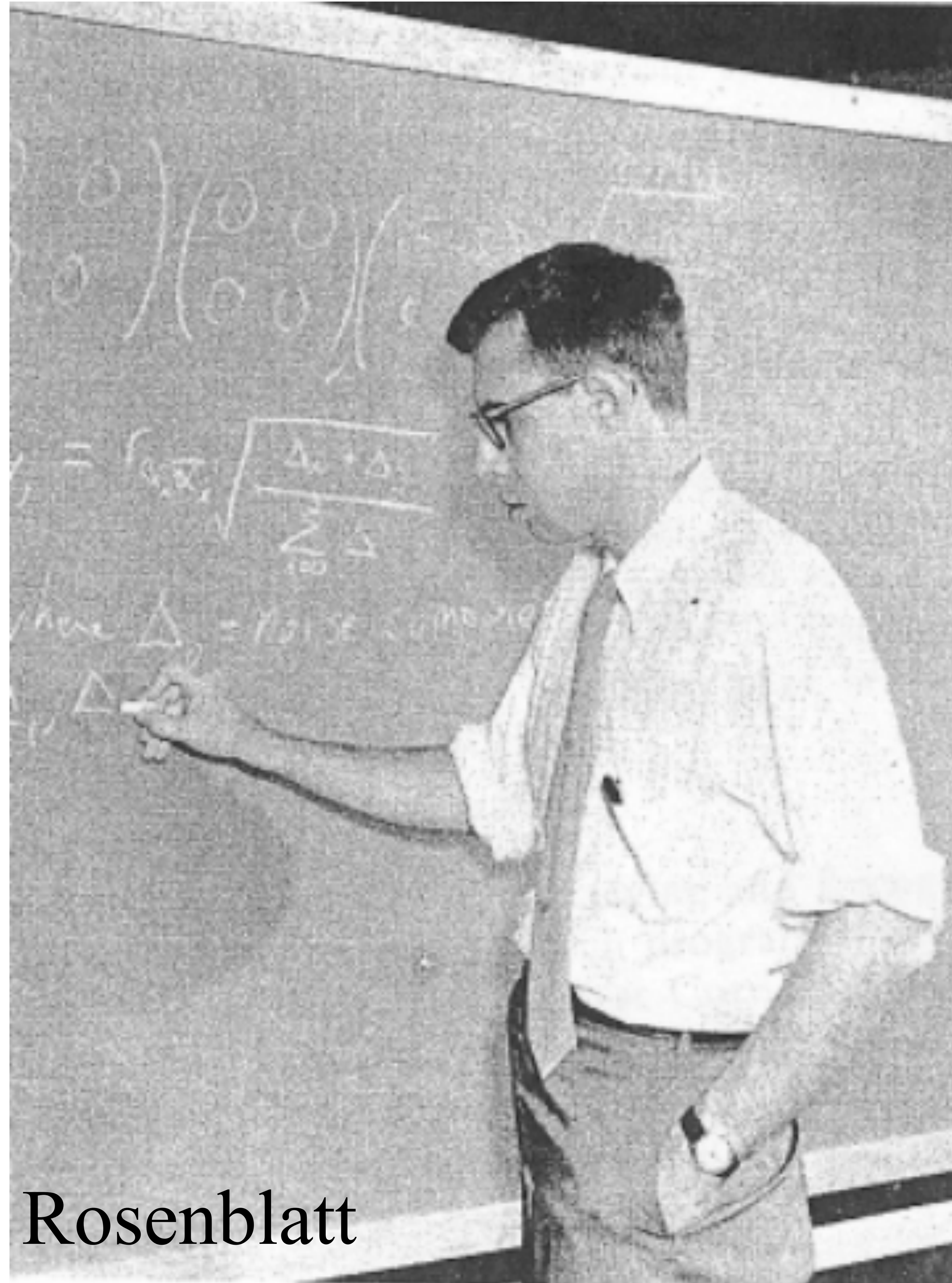
In the past, we didn't have enough data to fit these models. But now we do!

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

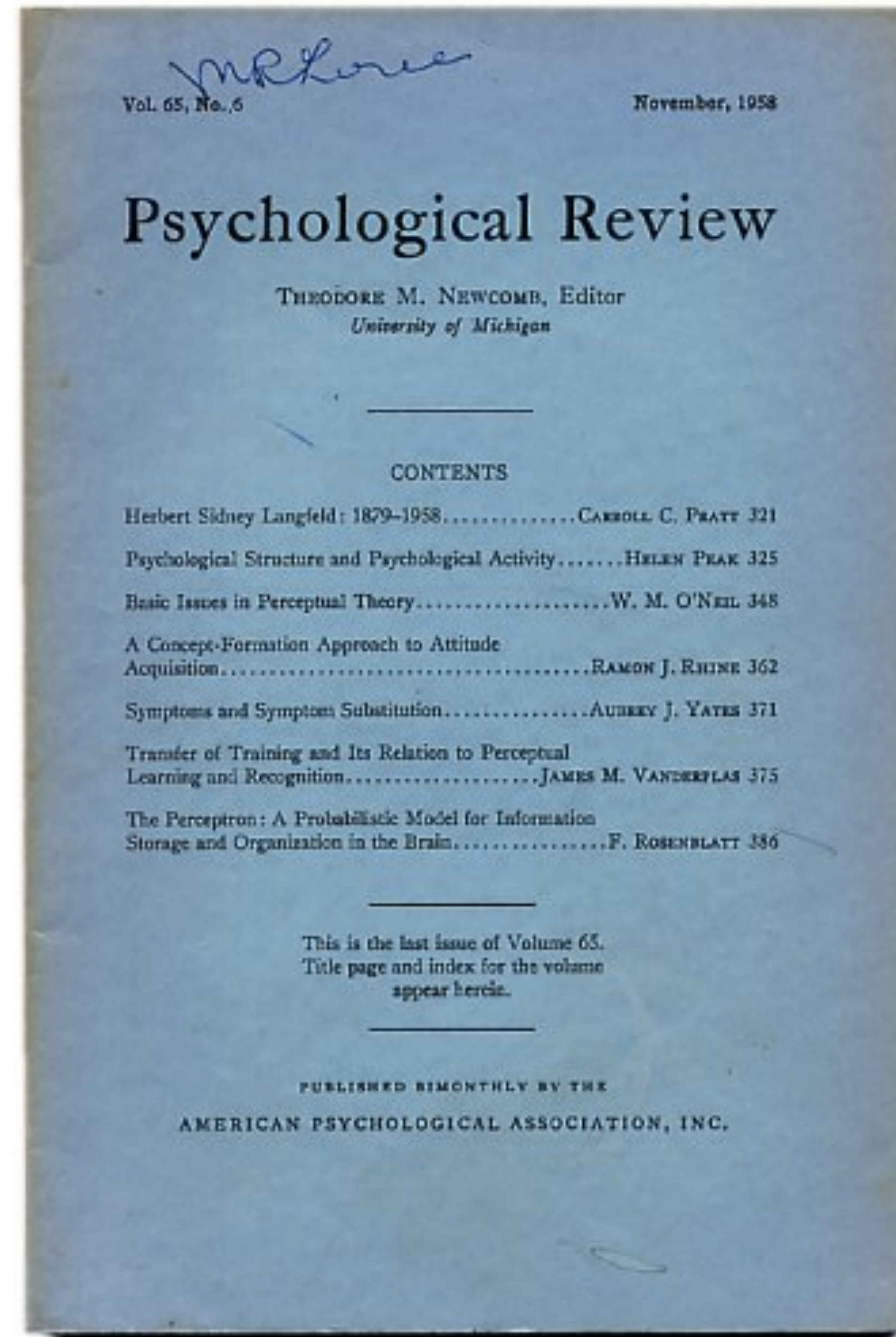
Deep neural networks!

A brief history of Neural Networks

Perceptrons, 1958



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

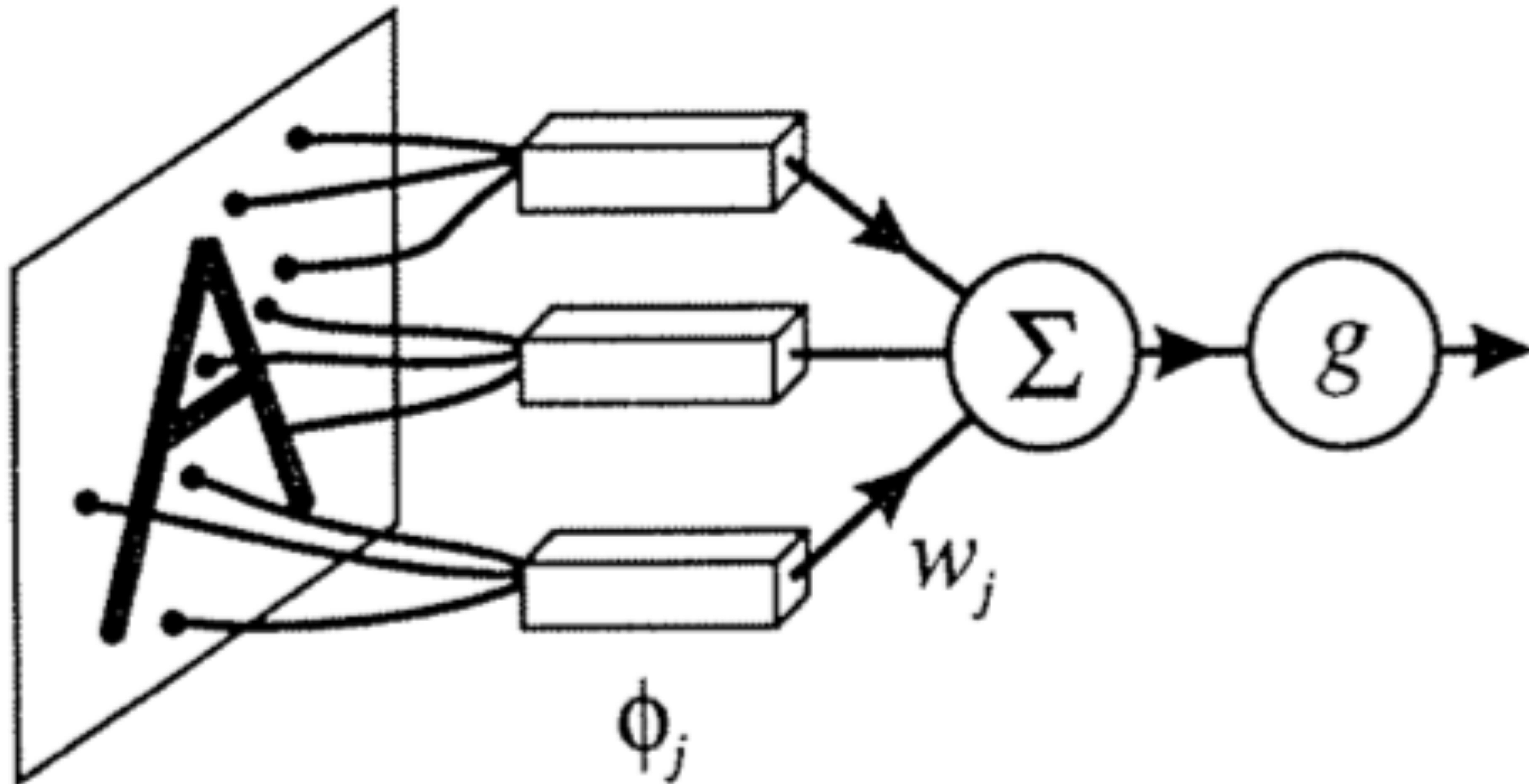
Herbert Sidney Langfeld: 1879–1958 .... Carroll C. Pratt 321
Psychological Structure and Psychological Activity .... Helen Peak 325
Basic Issues in Perceptual Theory .... W. M. O'Neil 348
A Concept-Formation Approach to Attitude Acquisition .... Ramon J. Rhine 362
Symptoms and Symptom Substitution .... Aubrey J. Yates 371
Transfer of Training and Its Relation to Perceptual Learning and Recognition .... James M. Vanderplas 375
The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain .... F. Rosenblatt 386

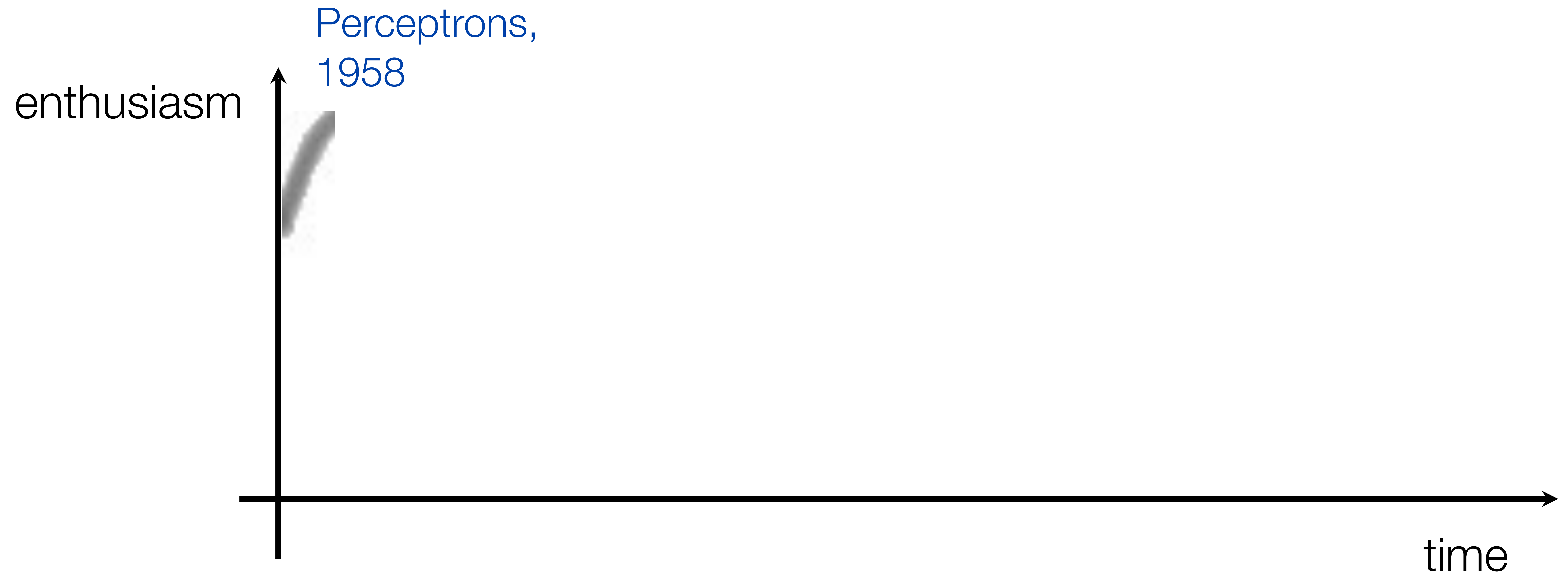
This is the last issue of Volume 65. Title page and index for the volume appear herein.

PUBLISHED BIMONTHLY BY THE
AMERICAN PSYCHOLOGICAL ASSOCIATION, INC.

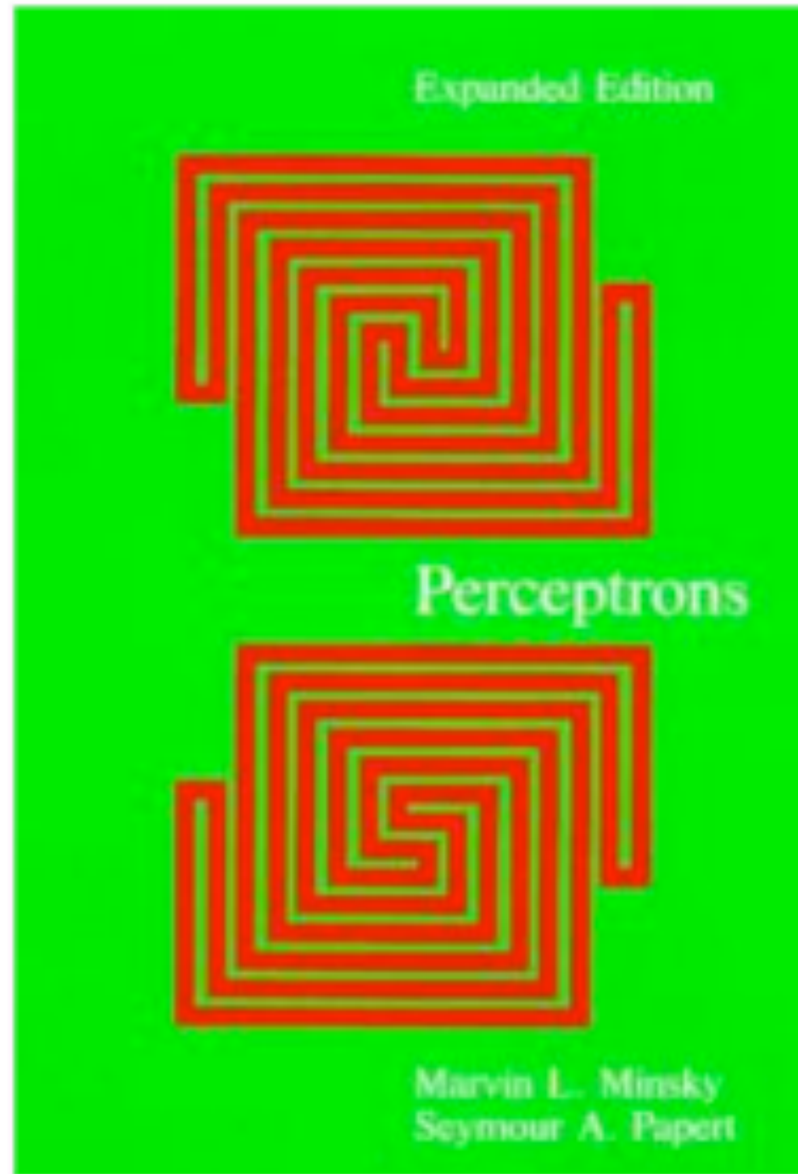
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf

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

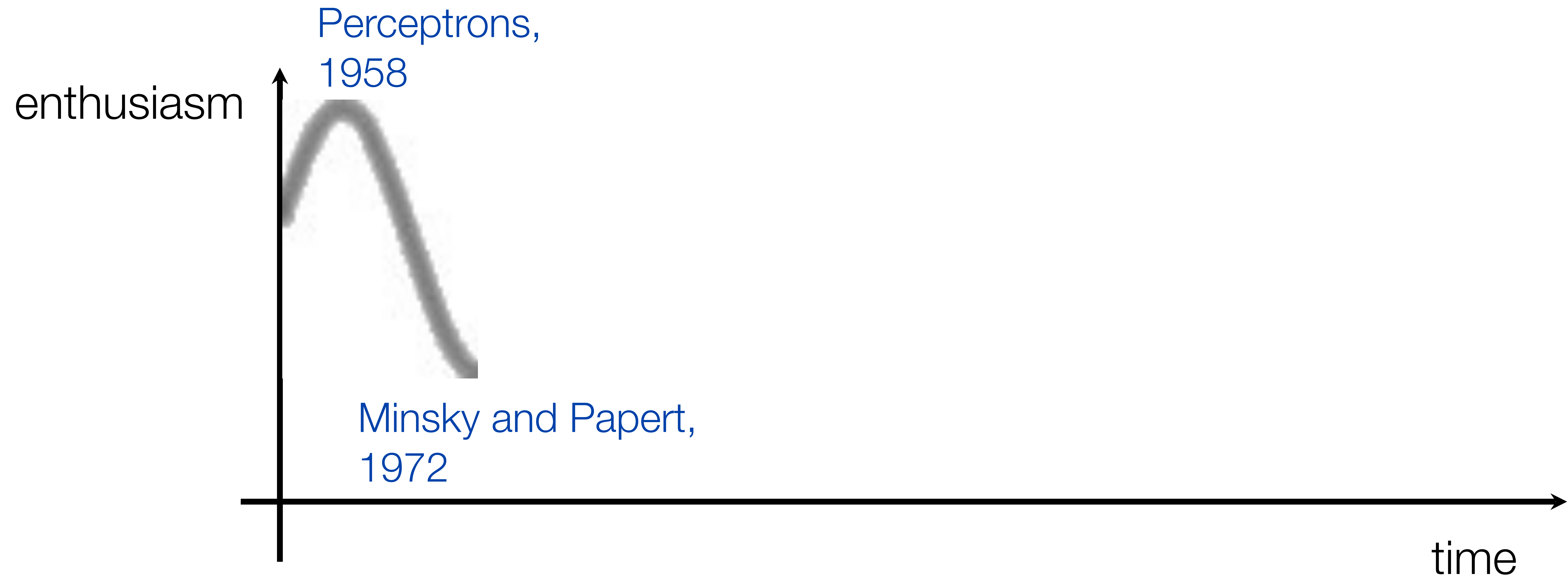
By Marvin Minsky and Seymour A. Papert

Overview

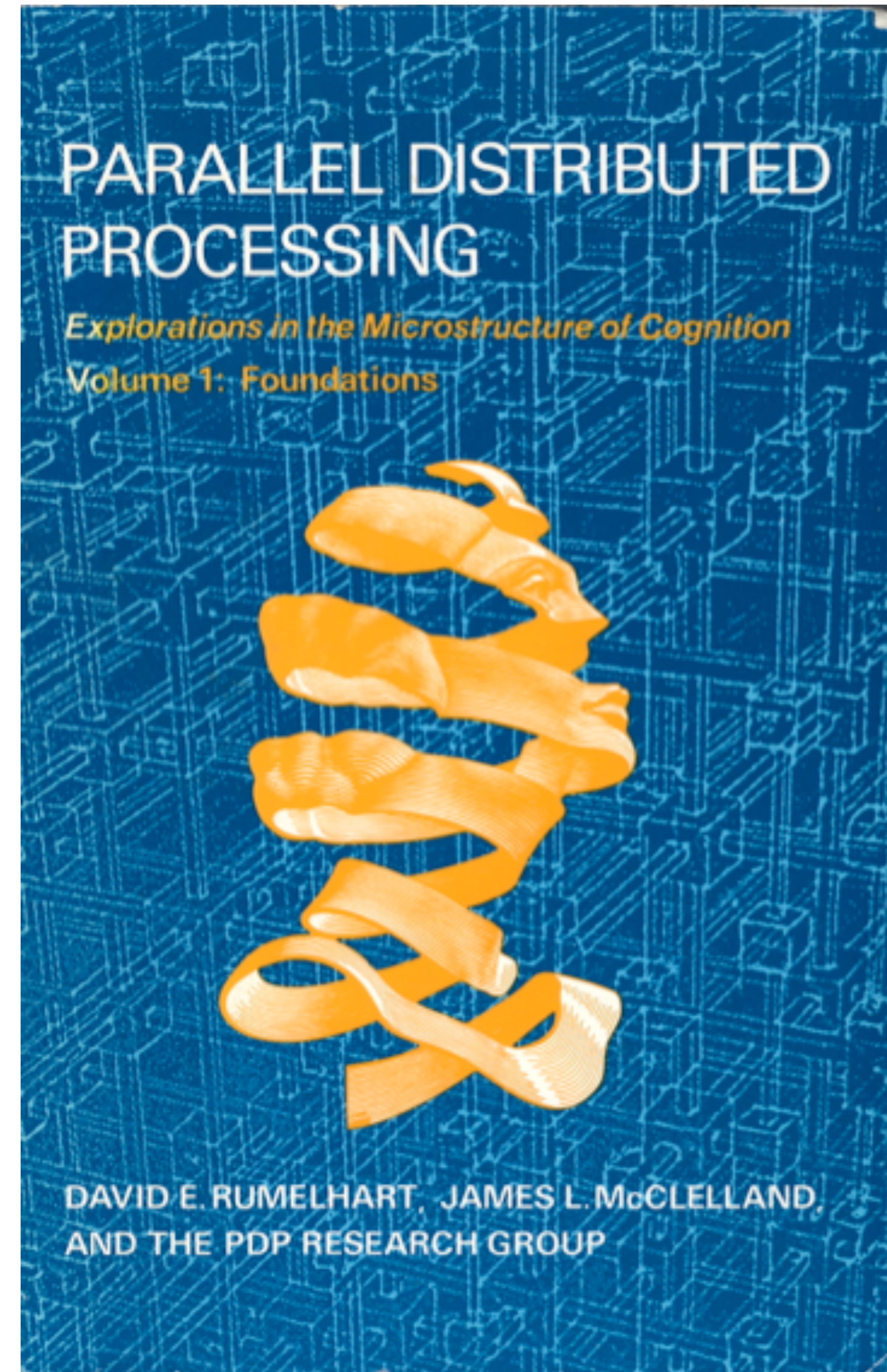
Perceptrons - the first systematic study of parallelism in computation - has remained a classical work on threshold automata networks for nearly two decades. It marked a historical turn in artificial intelligence, and it is required reading for anyone who wants to understand the connectionist counterrevolution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

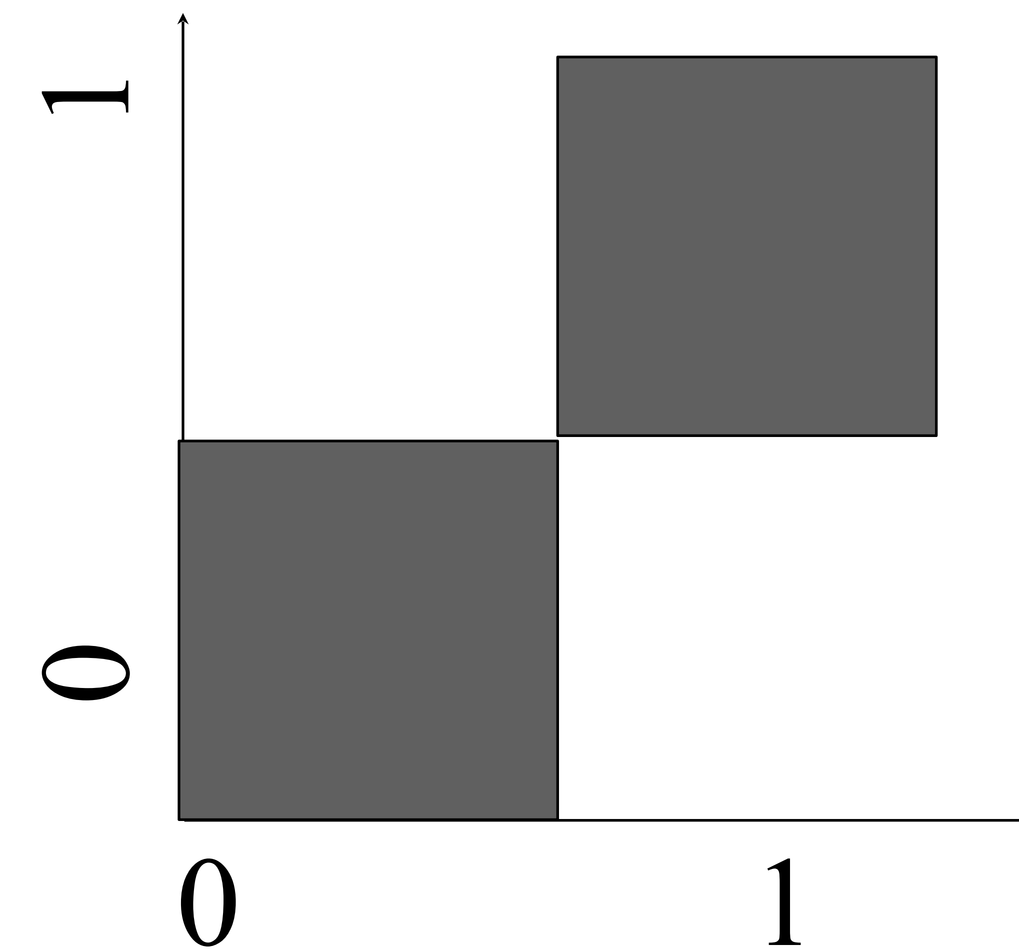

Perceptrons,
1958
enthusiasm
Minsky and Papert,
1972
time

Parallel Distributed Processing (PDP), 1986

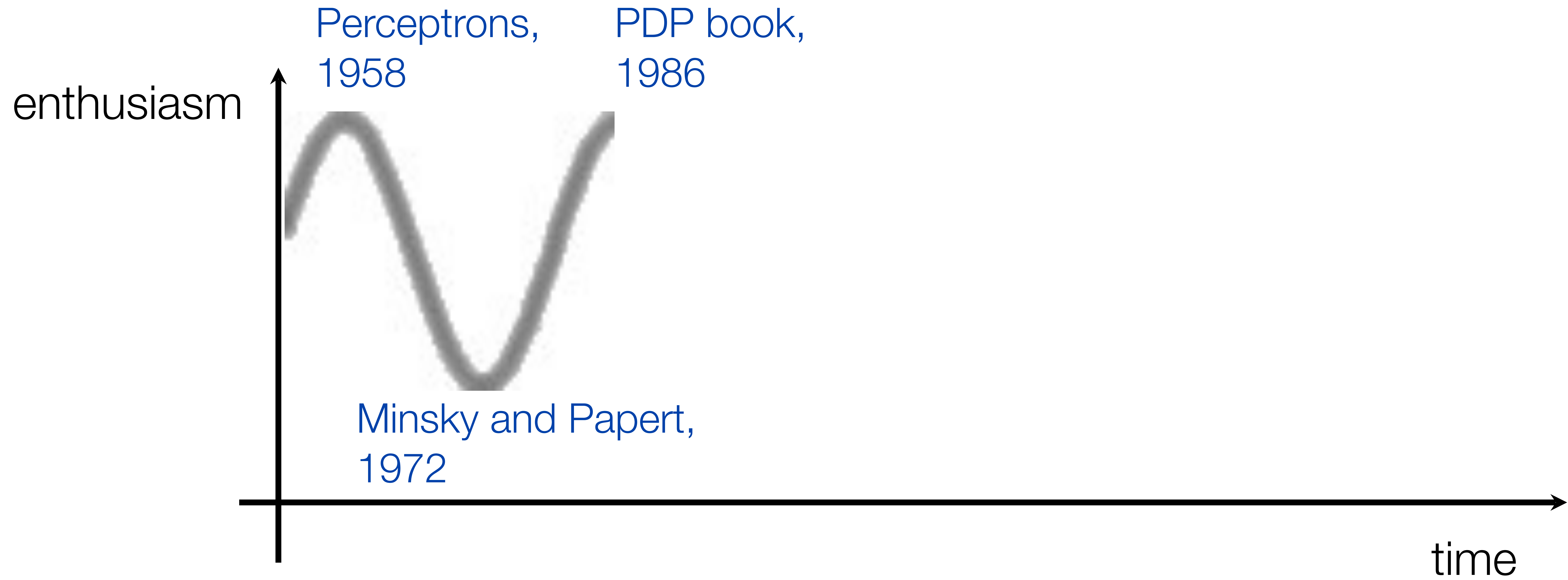


XOR problem

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

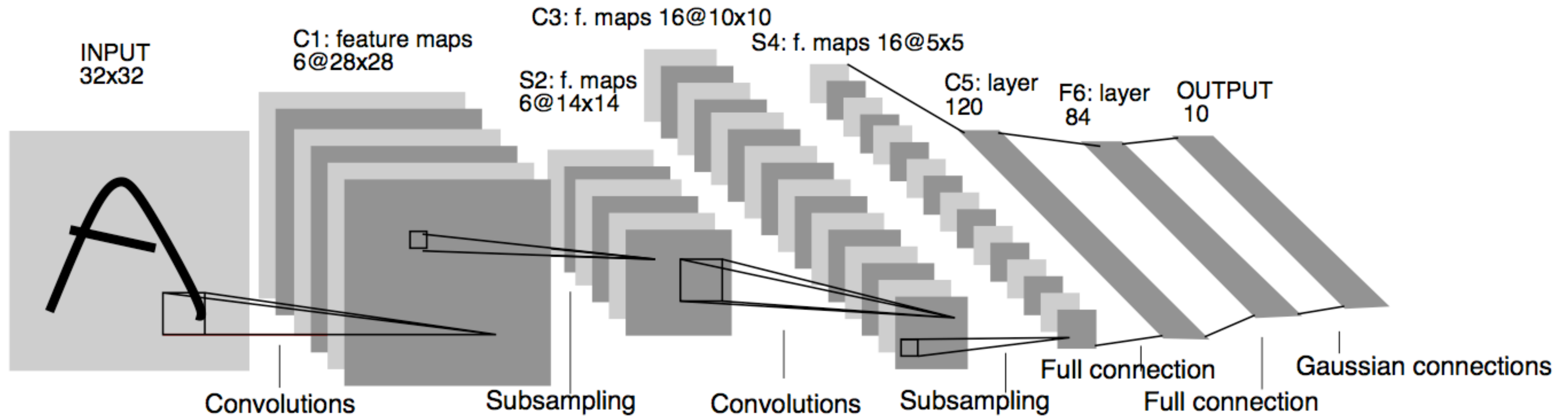


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

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

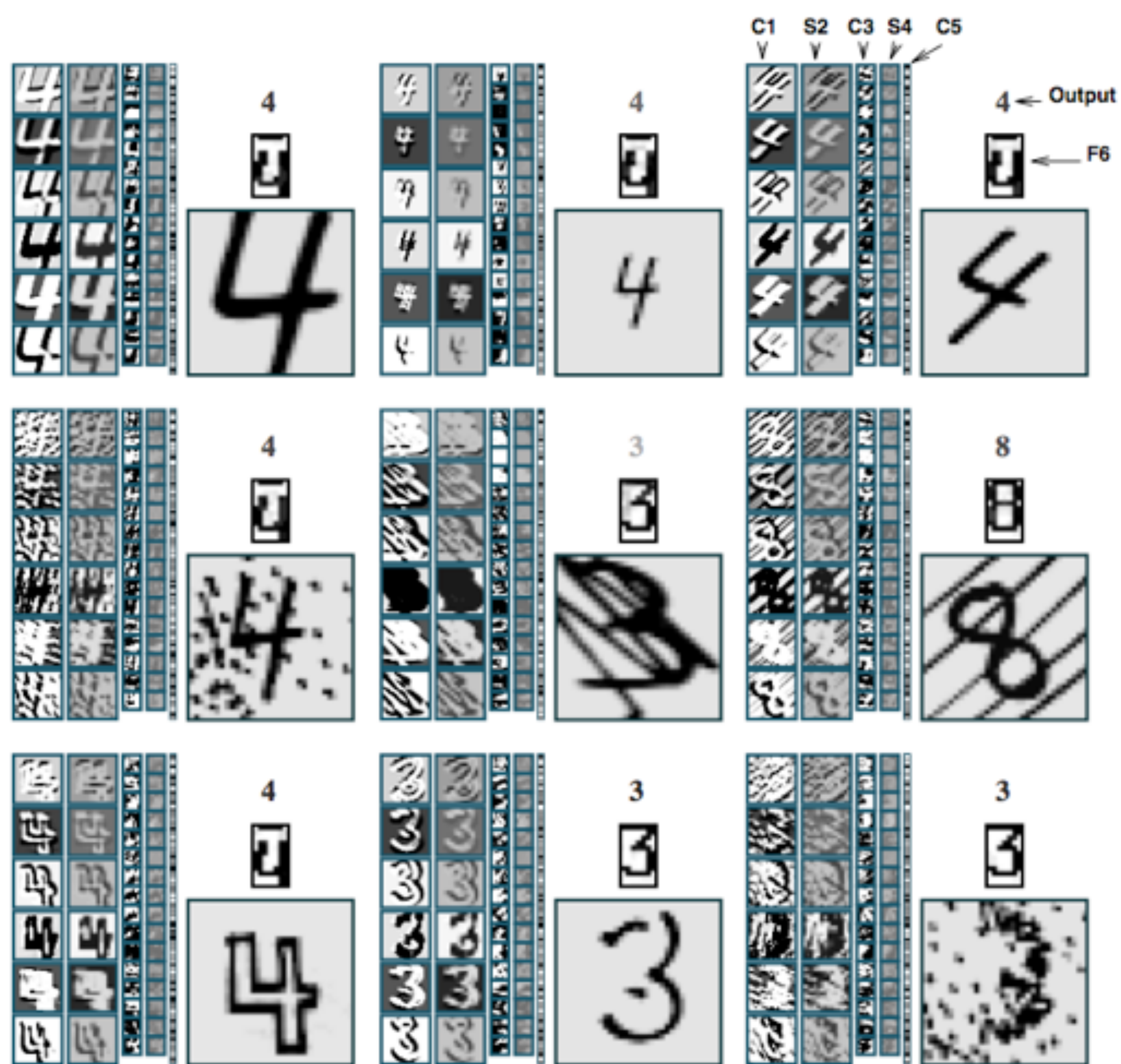
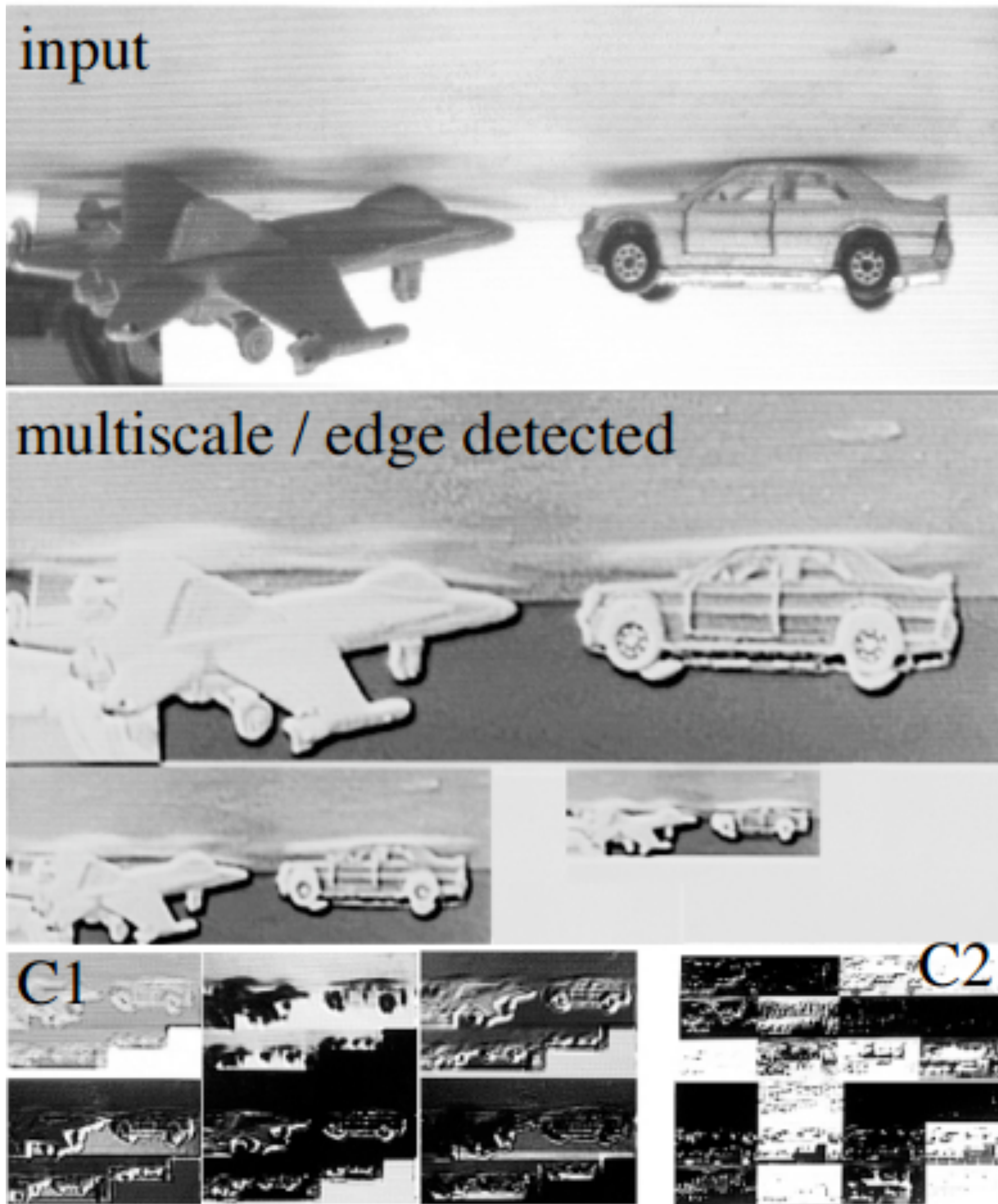


Fig. 13. Examples of unusual, distorted, and noisy characters correctly recognized by LeNet-5. The grey-level of the output label represents the penalty (lighter for higher penalties).



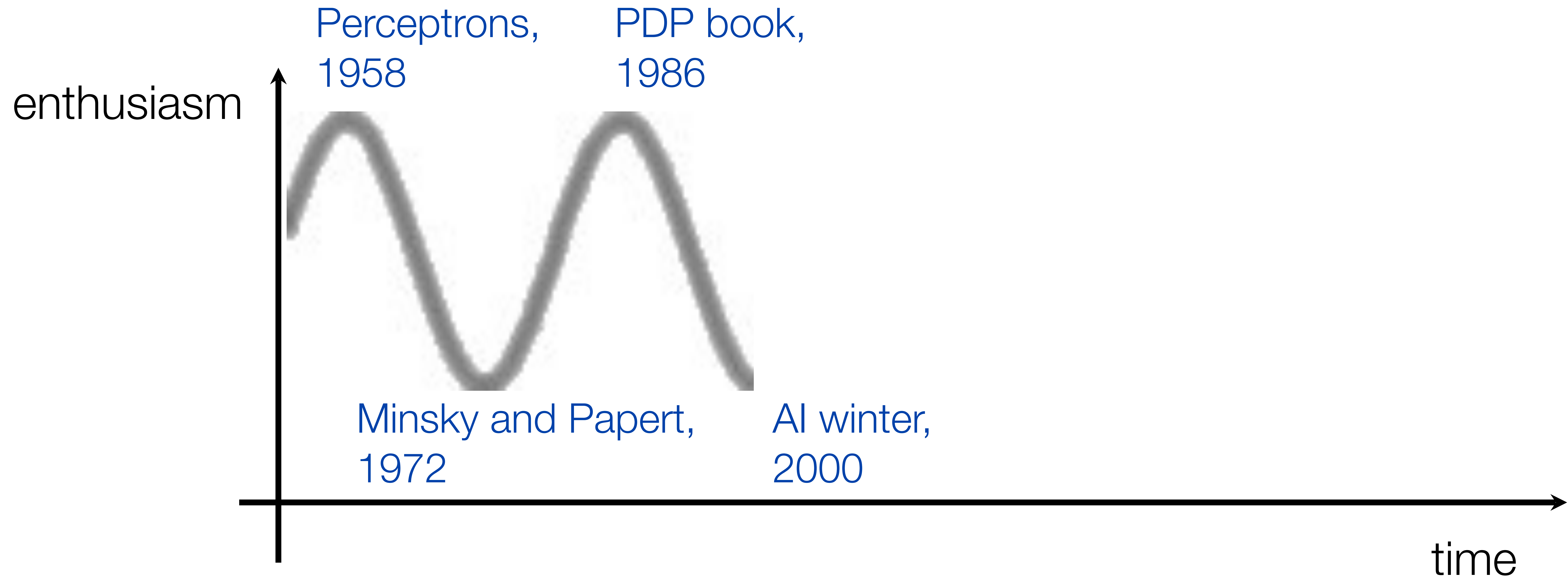
http://pub.clement.farabet.net/ecvw09.pdf

Neural networks to recognize handwritten digits?
yes

Neural networks for tougher problems?
not really

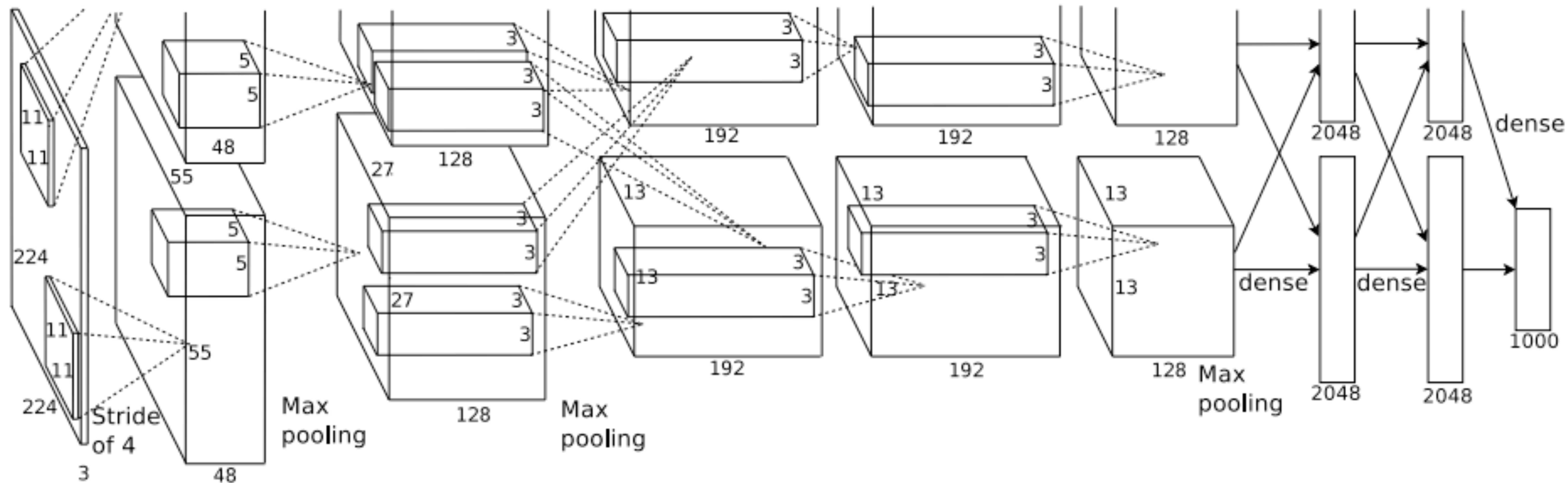
NIPS 2000

- NIPS, Neural Information Processing Systems, is the premier conference on machine learning. Evolved from an interdisciplinary conference to a machine learning conference.
- For the NIPS 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, NIPS 2012

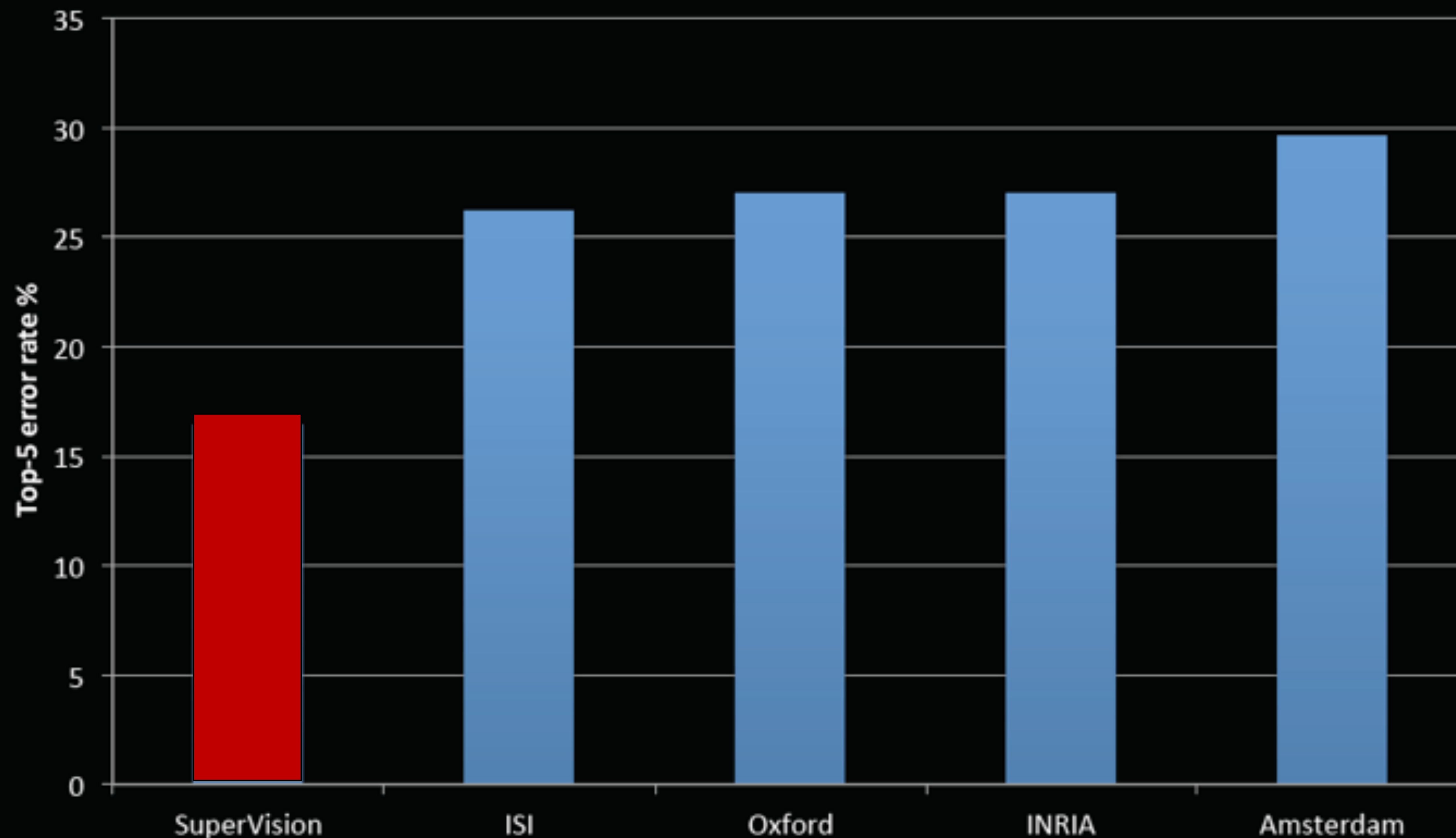
“Alexnet”



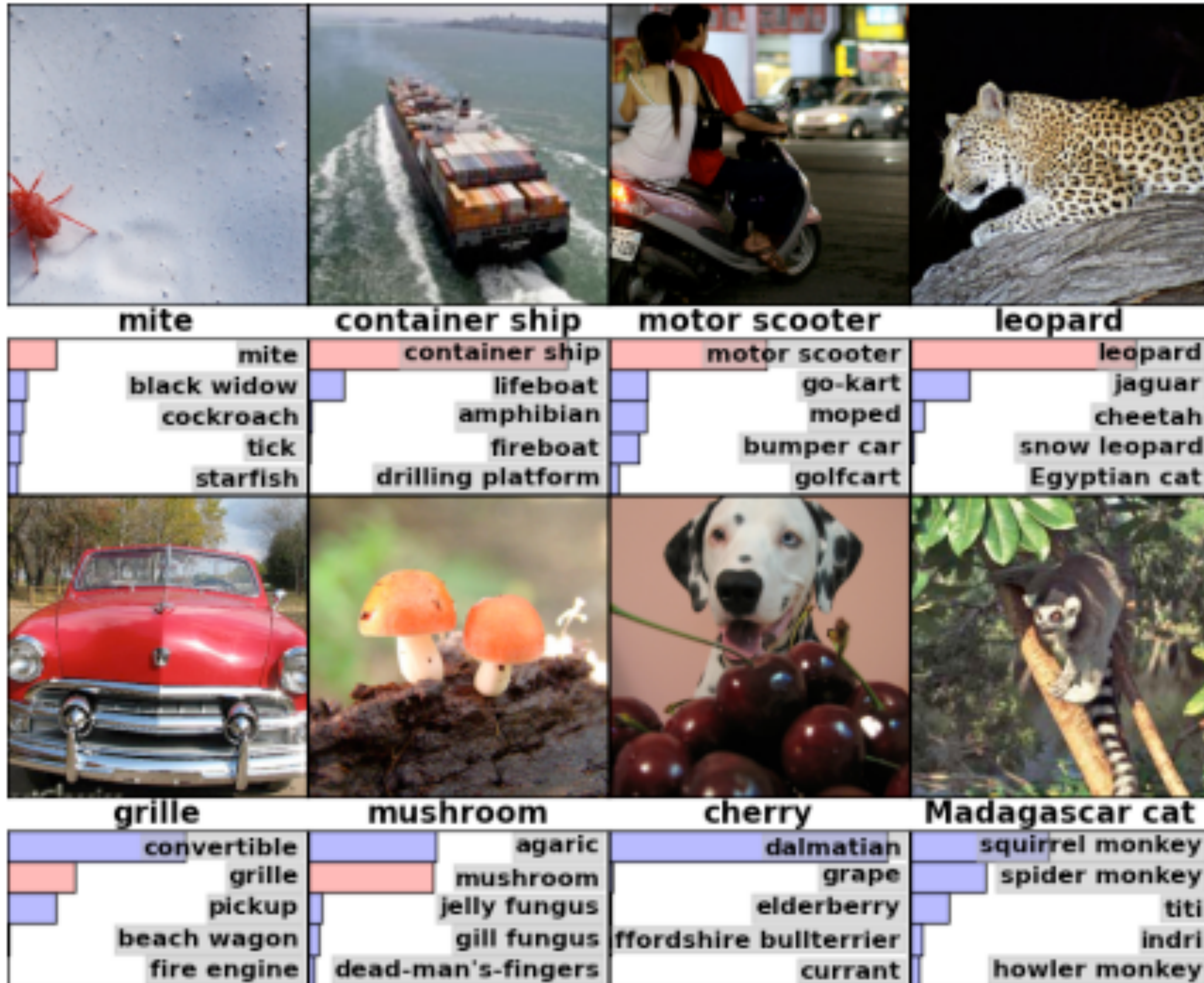
Slide from Rob Fergus, NYU

ImageNet Classification 2012

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

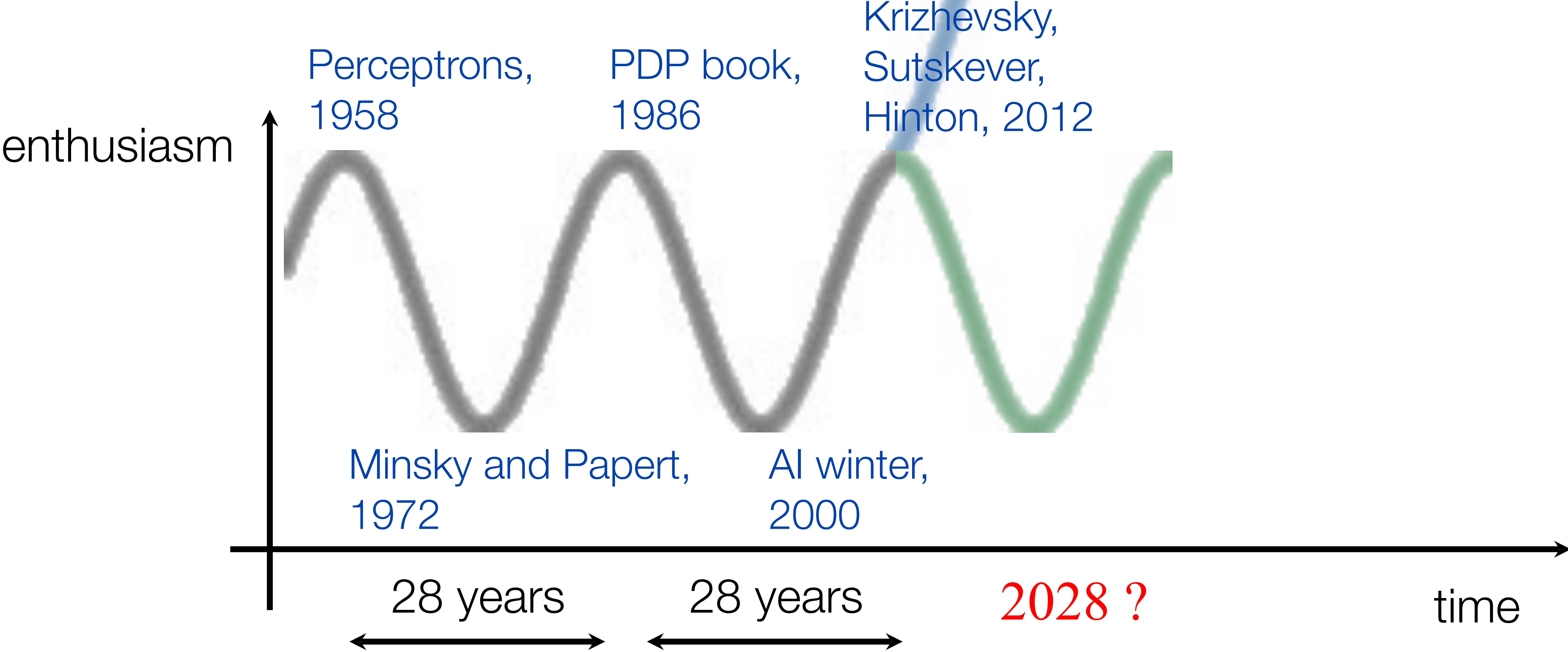


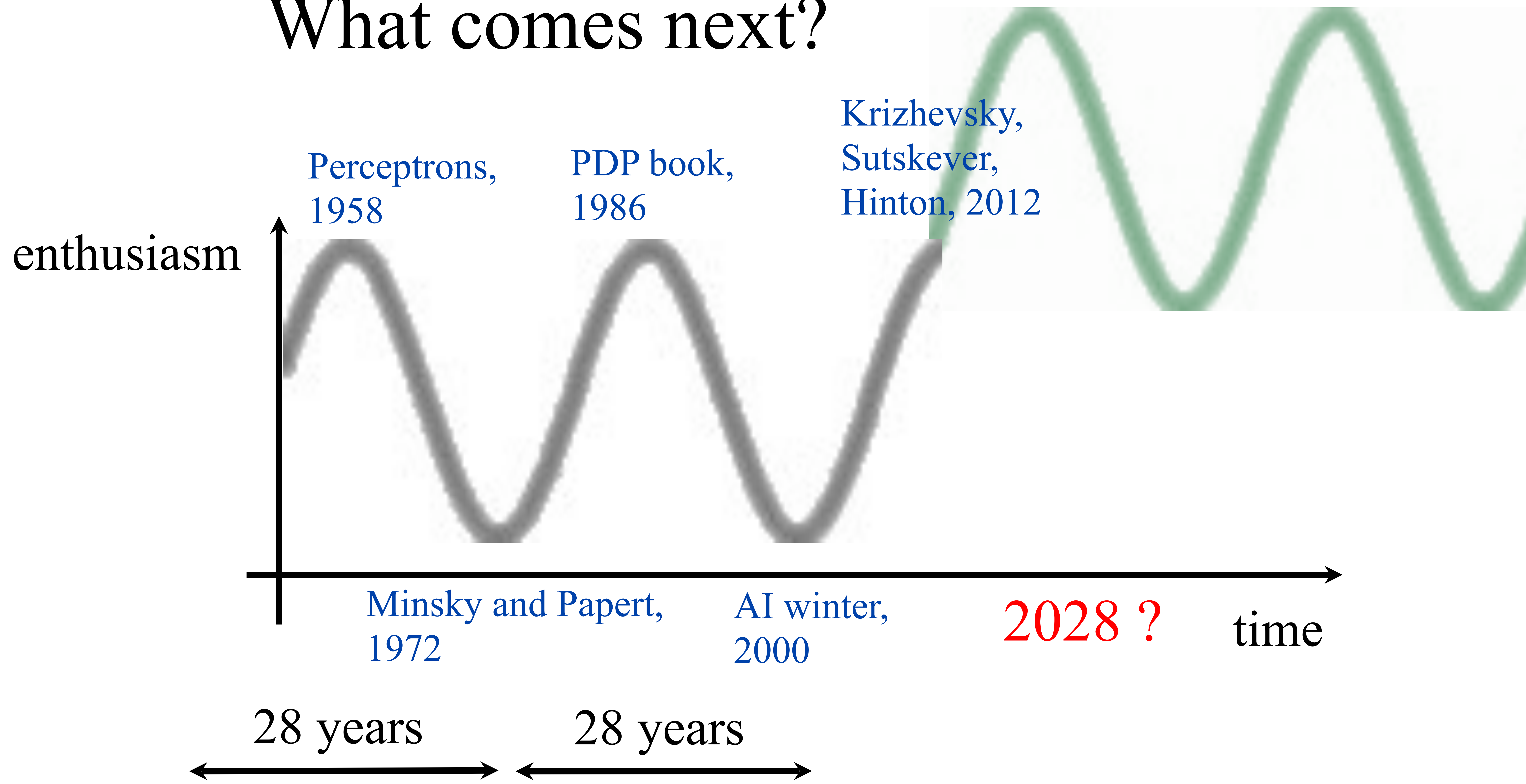
Krizhevsky, Sutskever, and Hinton, NIPS 2012

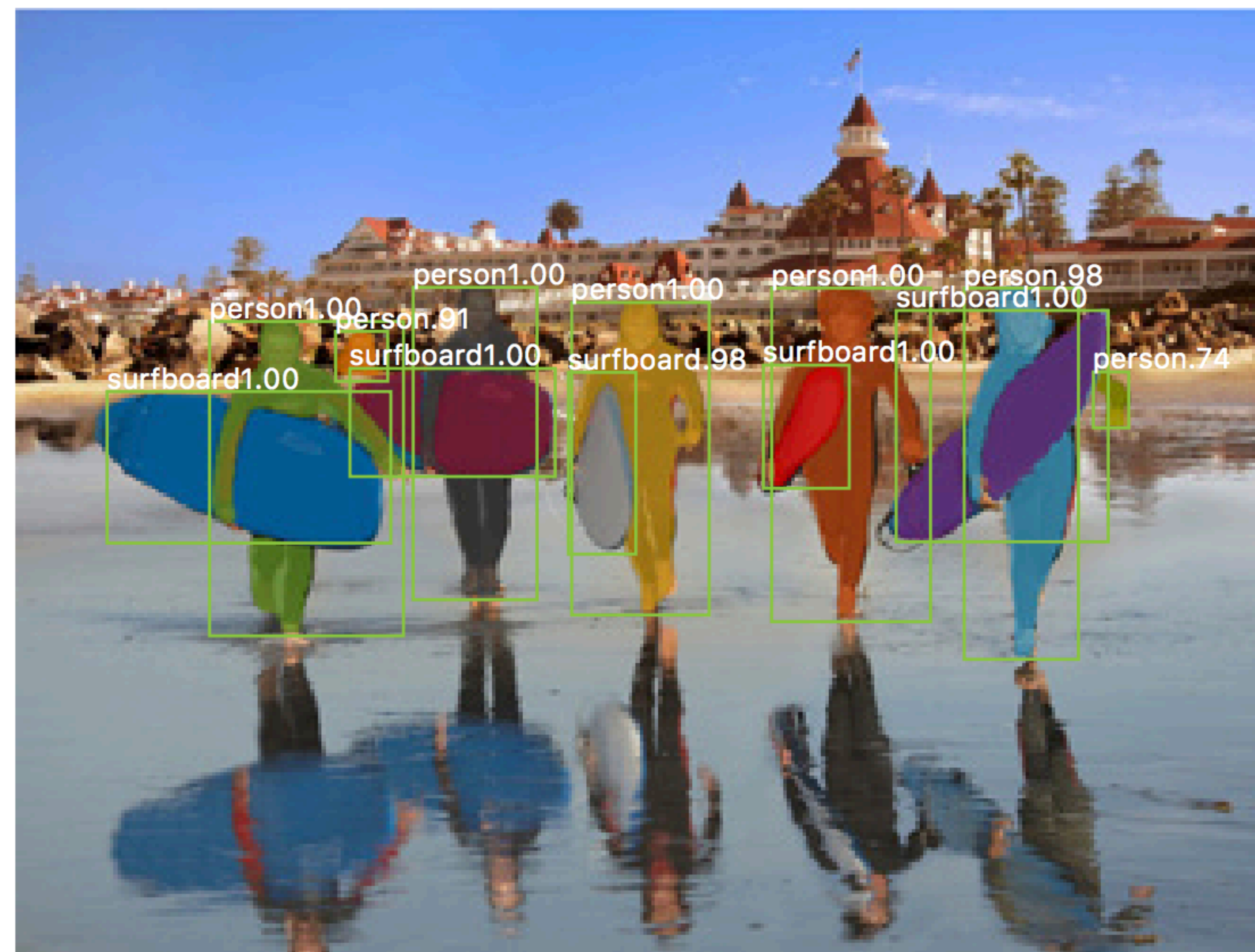


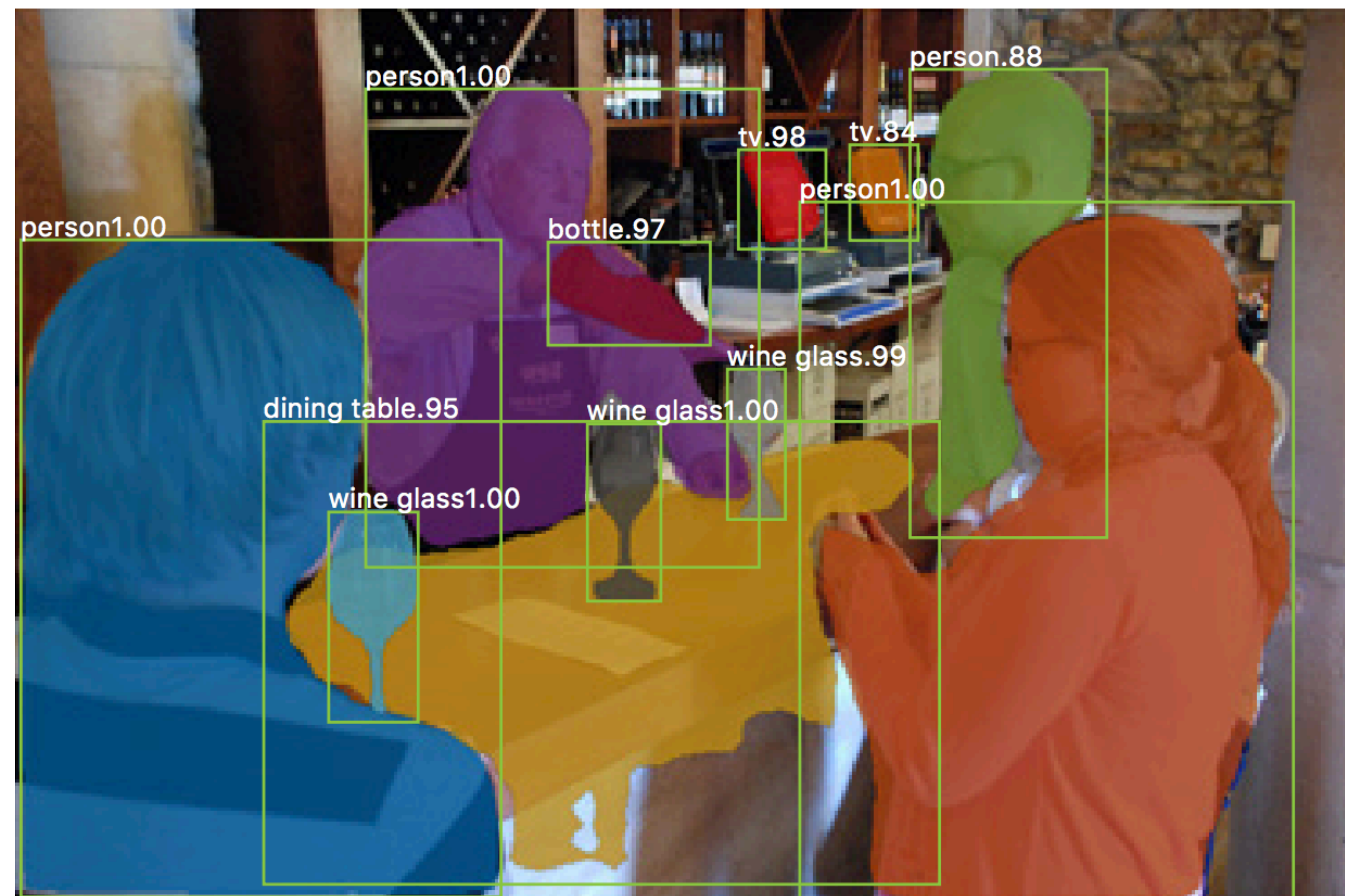
28 years
28 years
Perceptrons,
1958
PDP book,
1986
Krizhevsky,
Sutskever,
Hinton, 2012
enthusiasm
Minsky and Papert,
1972
AI winter,
2000
time

What comes next?

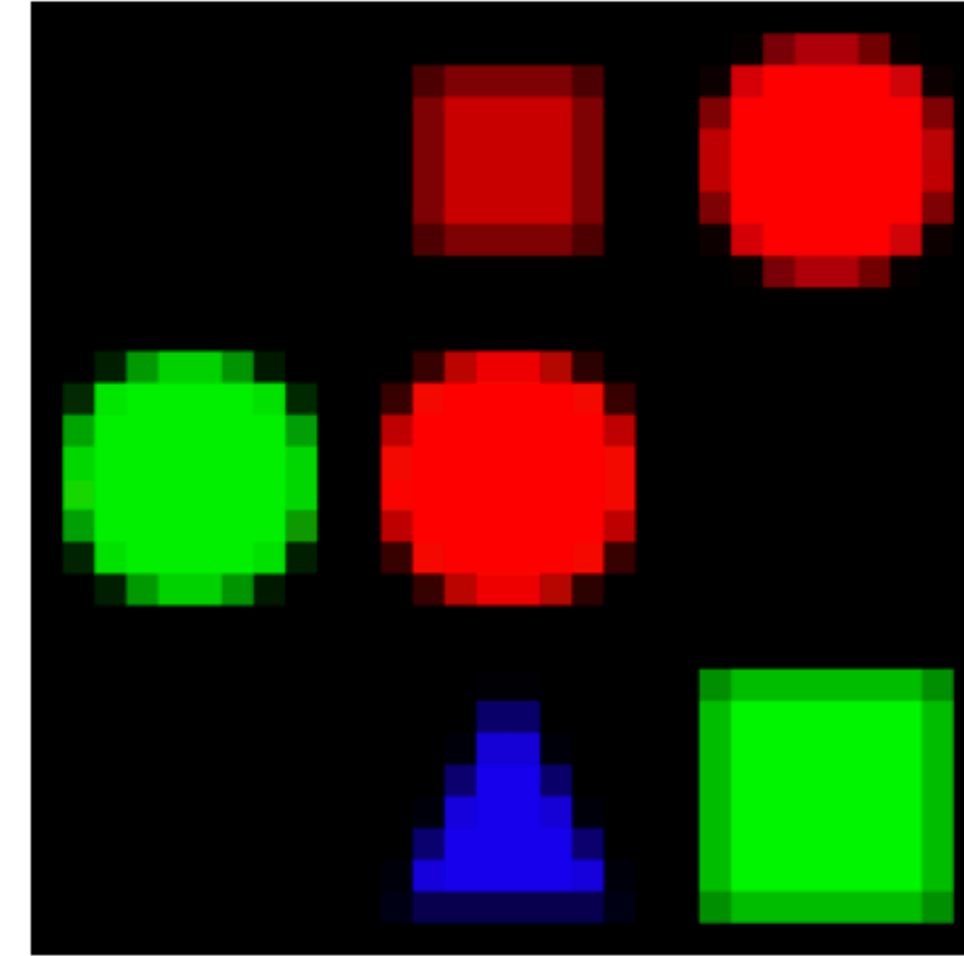



What comes next?
Perceptrons,
1958
PDP book,
1986
Krizhevsky,
Sutskever,
Hinton, 2012
enthusiasm
Minsky and Papert,
1972
AI winter,
2000
2028 ?
time
28 years
28 years





["Mask RCNN", He et al. 2017]

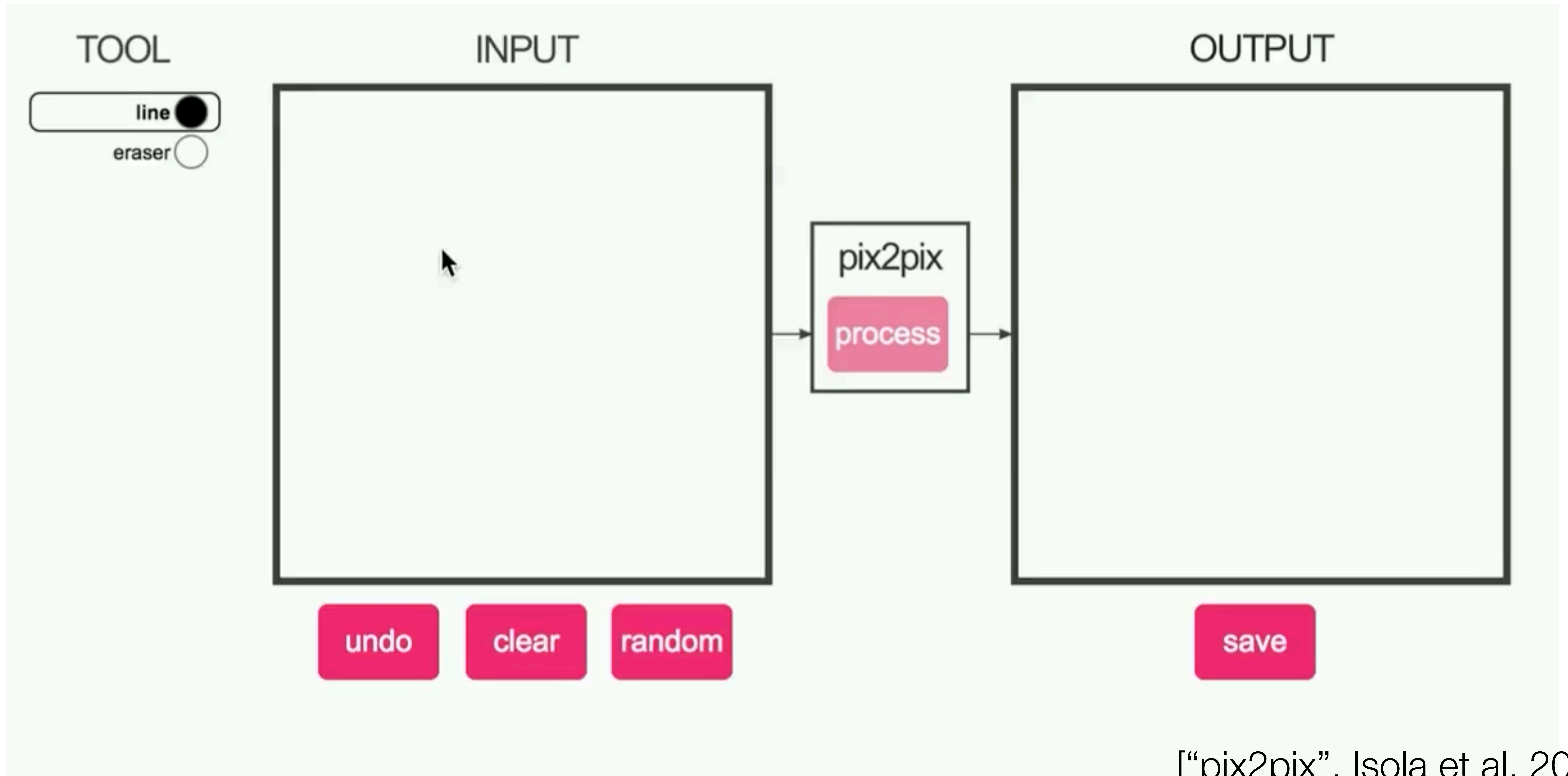
 what color is the vase?	 *is the bus full of passengers?*	 *is there a red shape above a circle?*

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

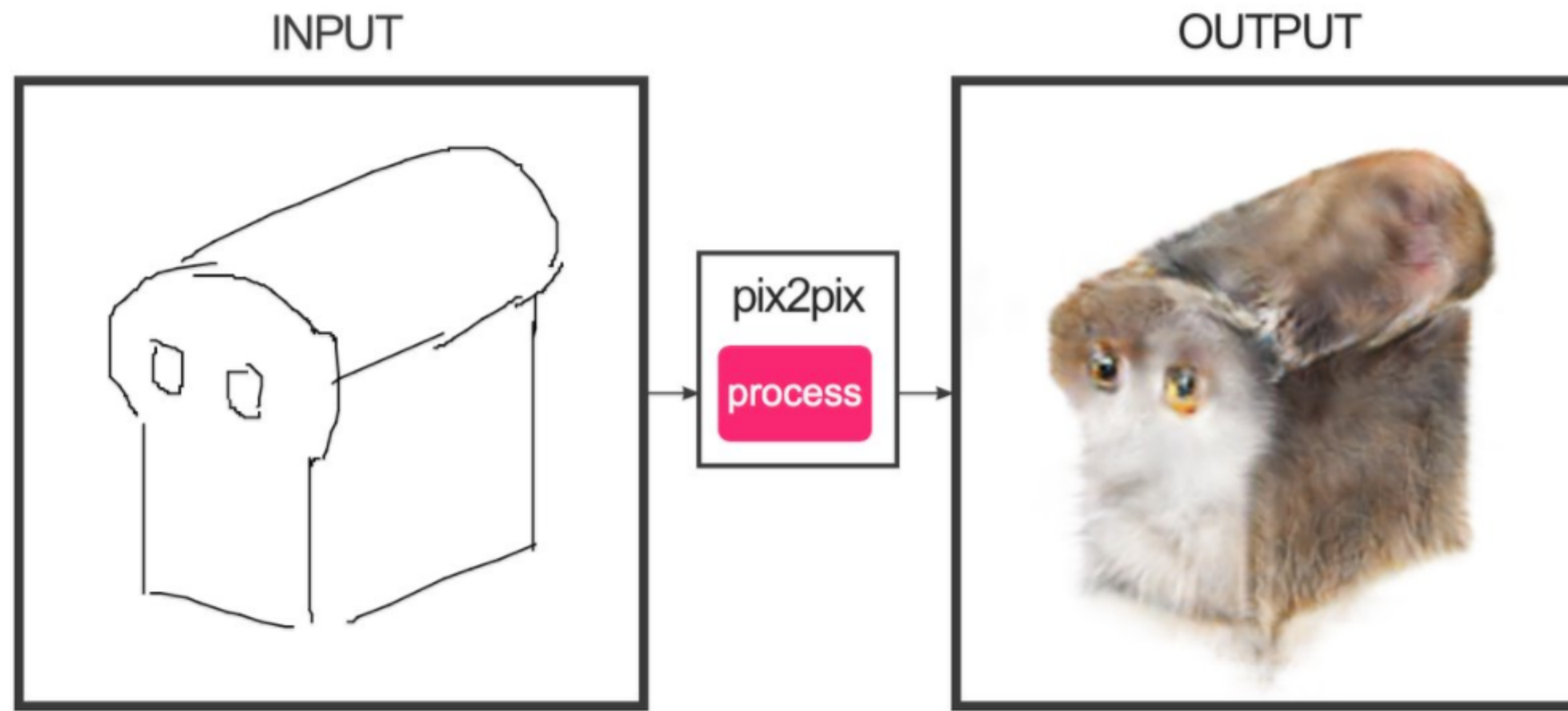
green (green)	yes (yes)	no (no)

["Neural module networks", Andreas et al. 2017]

#edges2cats [Chris Hesse]

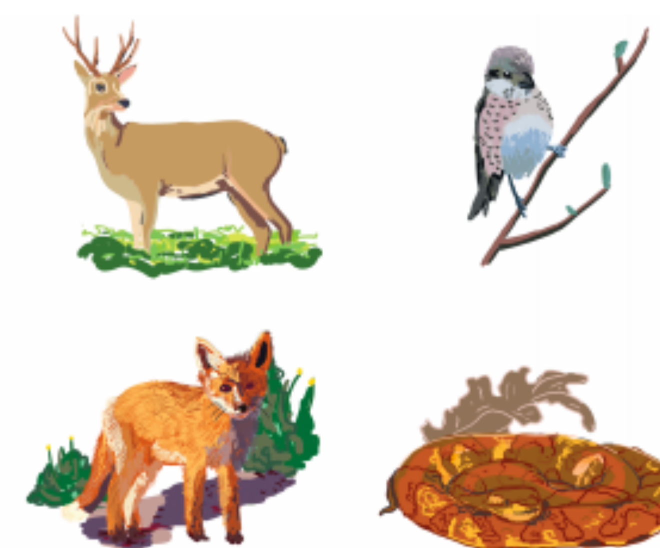


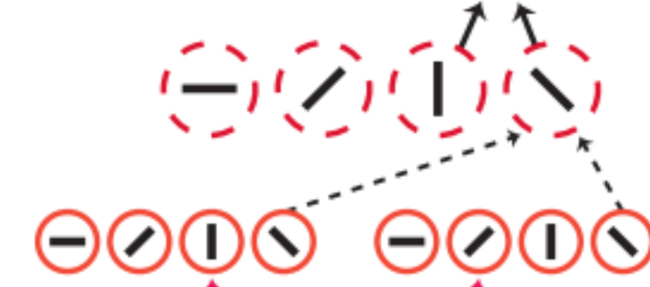
["pix2pix", Isola et al. 2017]



Ivy Tasi @ivymyt

Vitaly Vidmirov @vvid





Serre, 2014

Image transformations

Input image

Edge normals

Edge strength

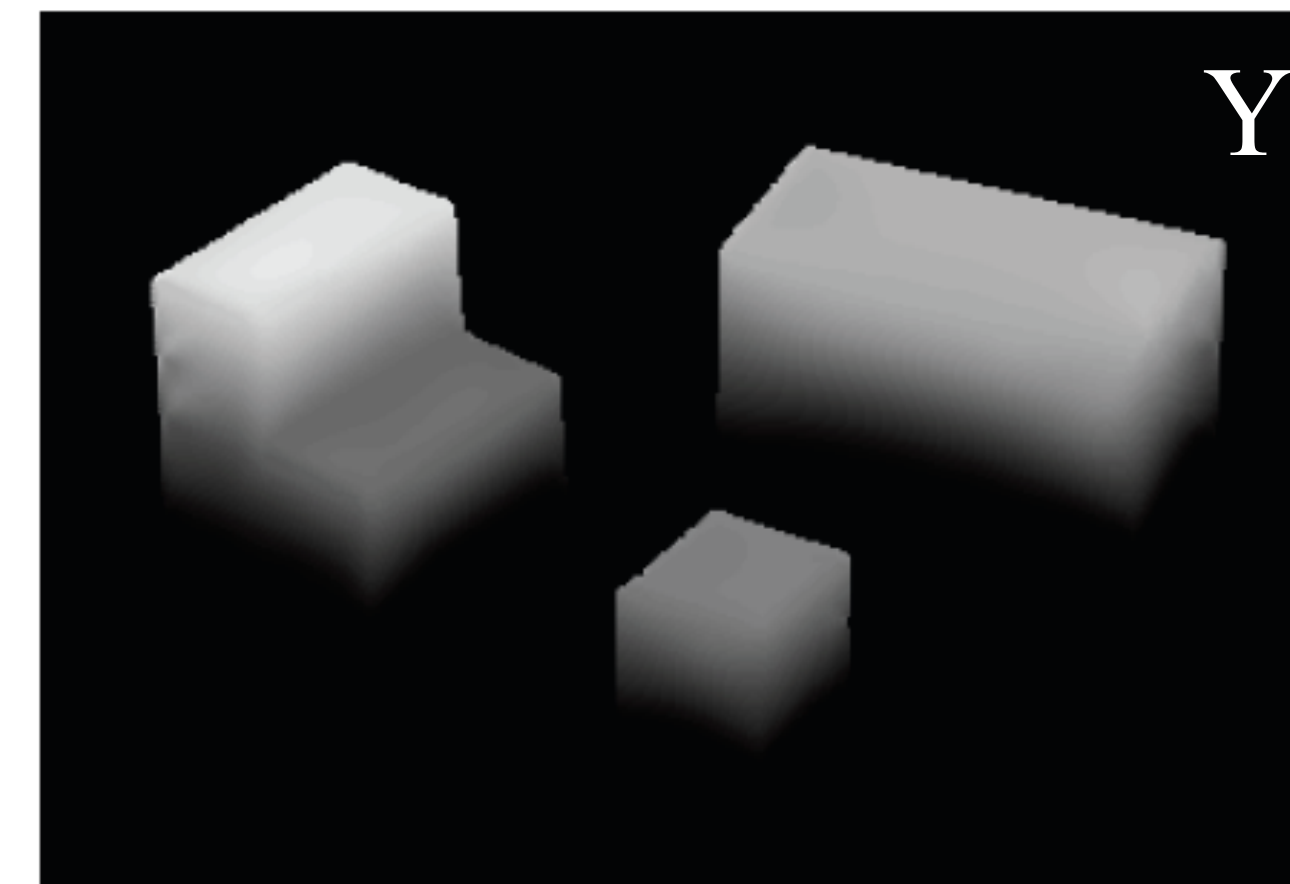
3D orientation

Contact edges

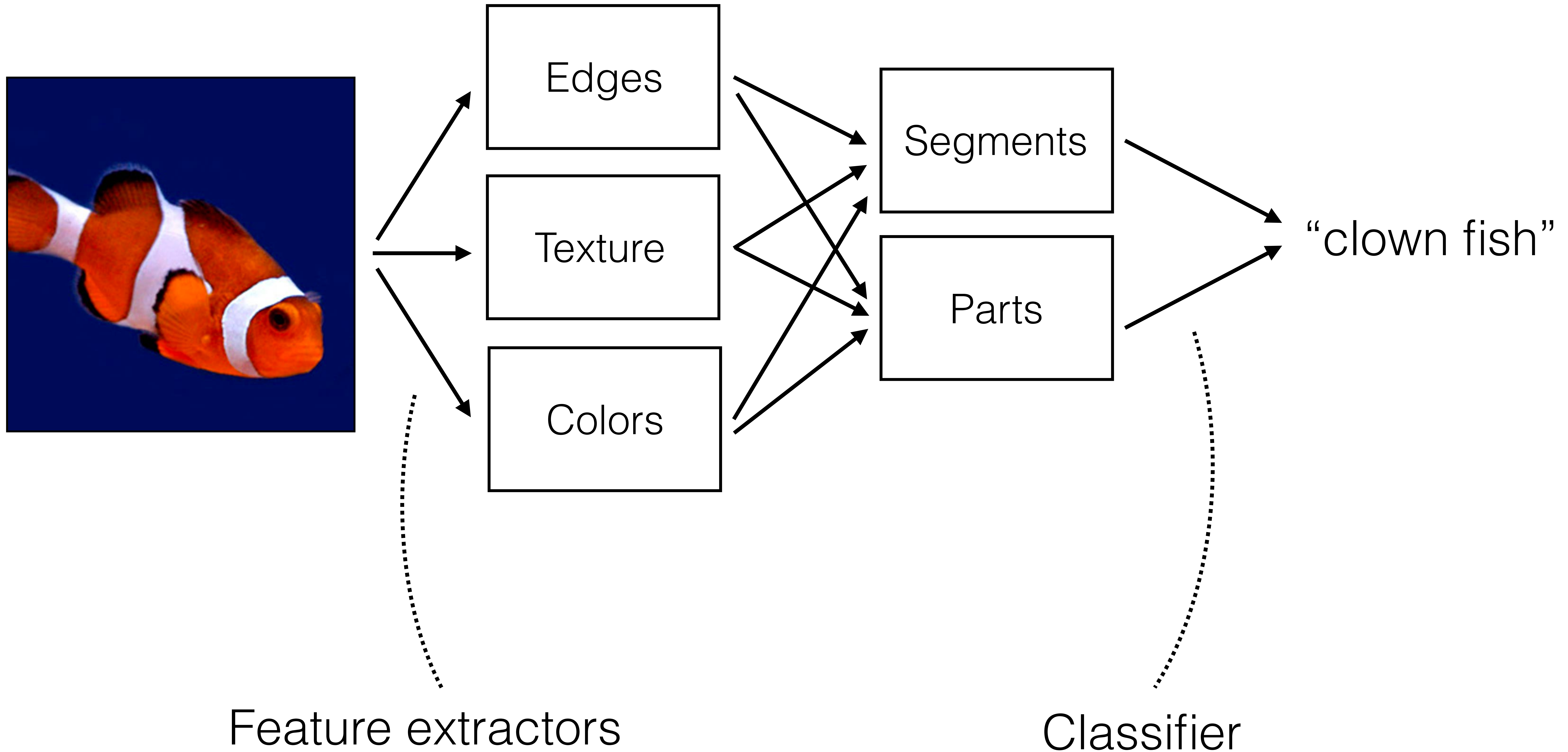
Depth discontinuities

1. From pixels to edges

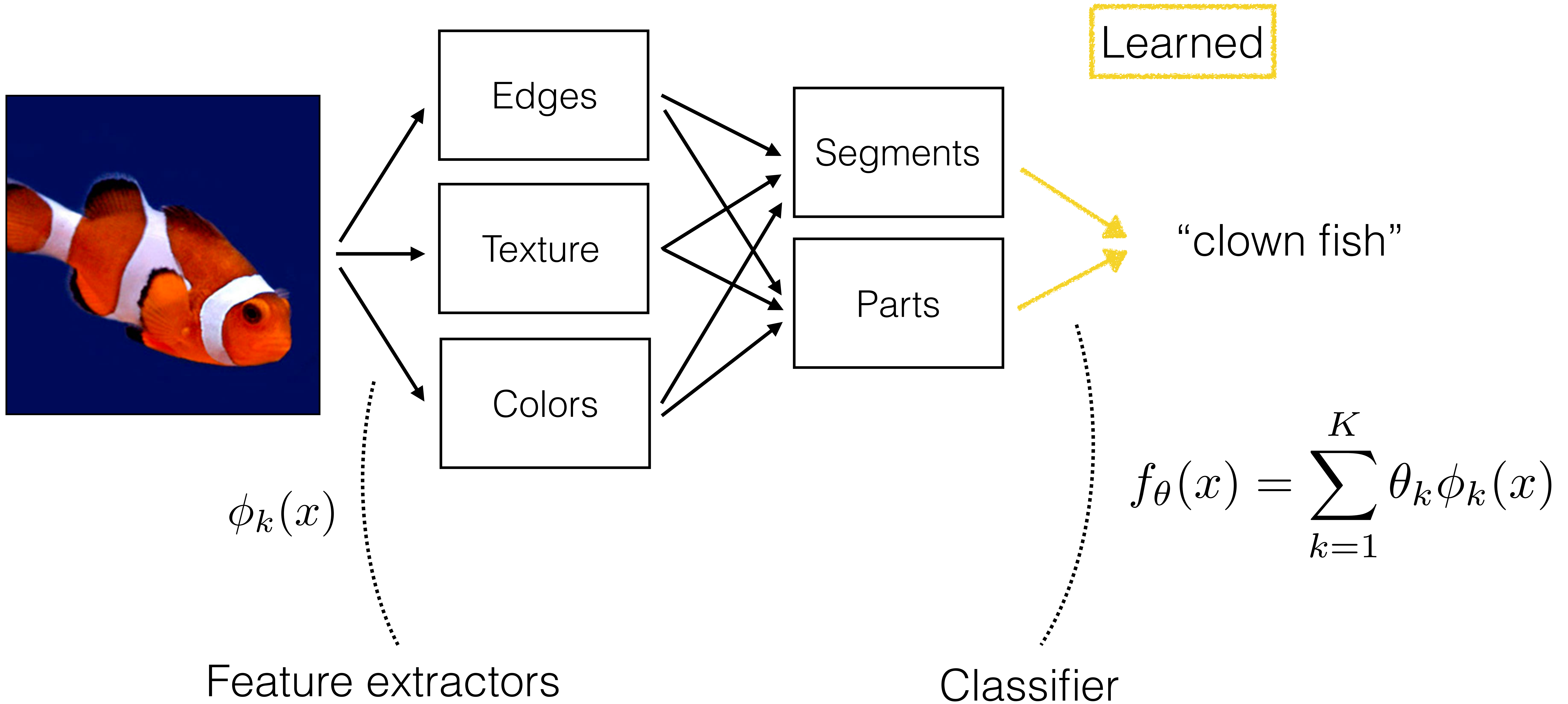
2. From edges to geometric primitives



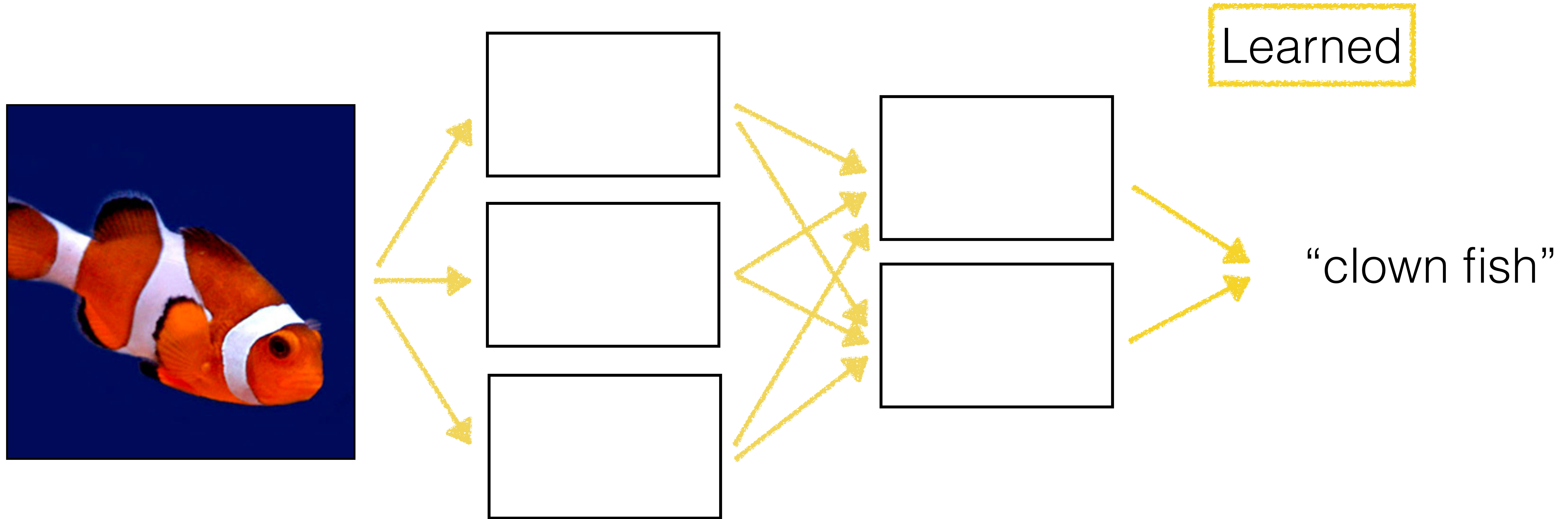
Object recognition



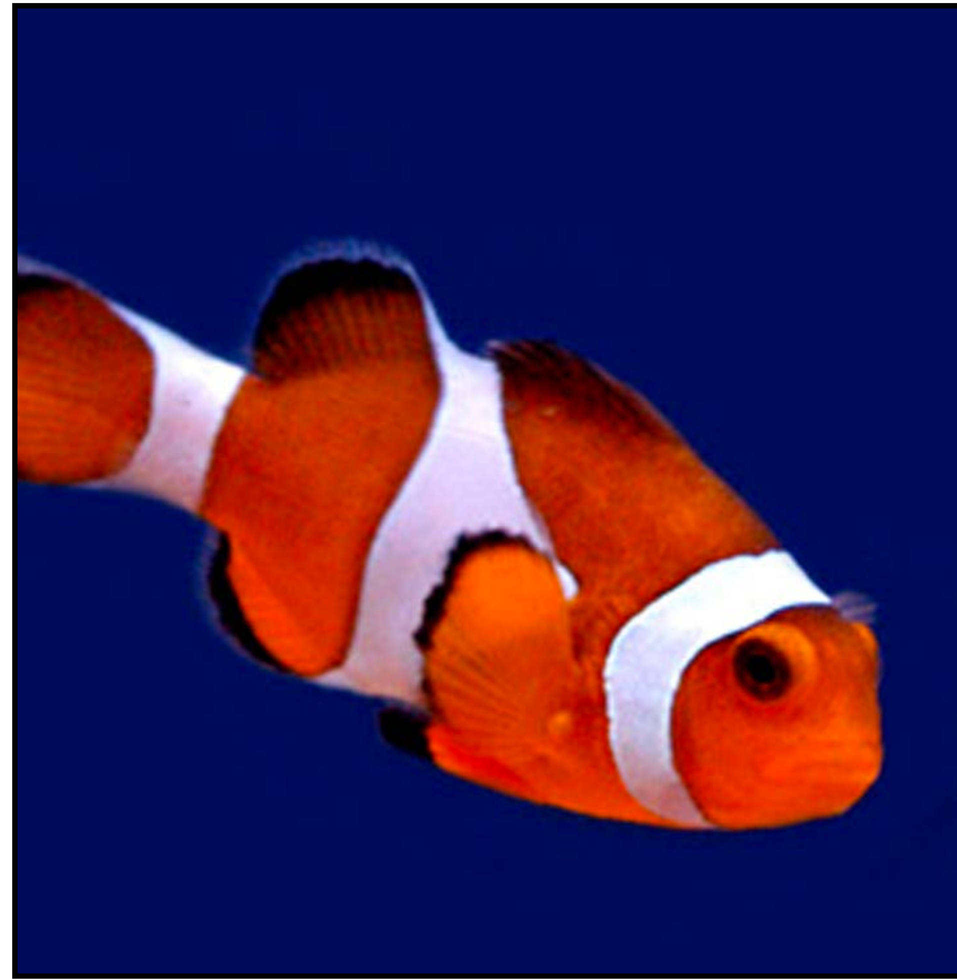
Object recognition

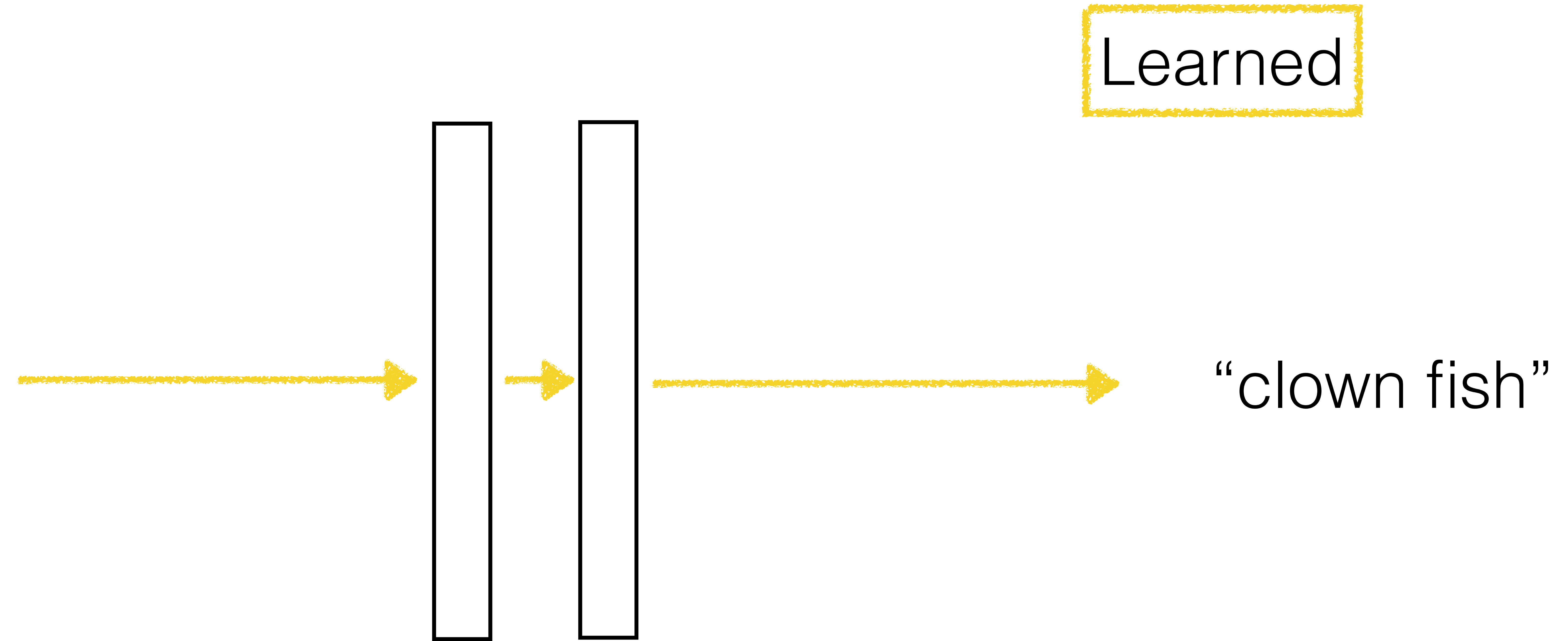


Object recognition



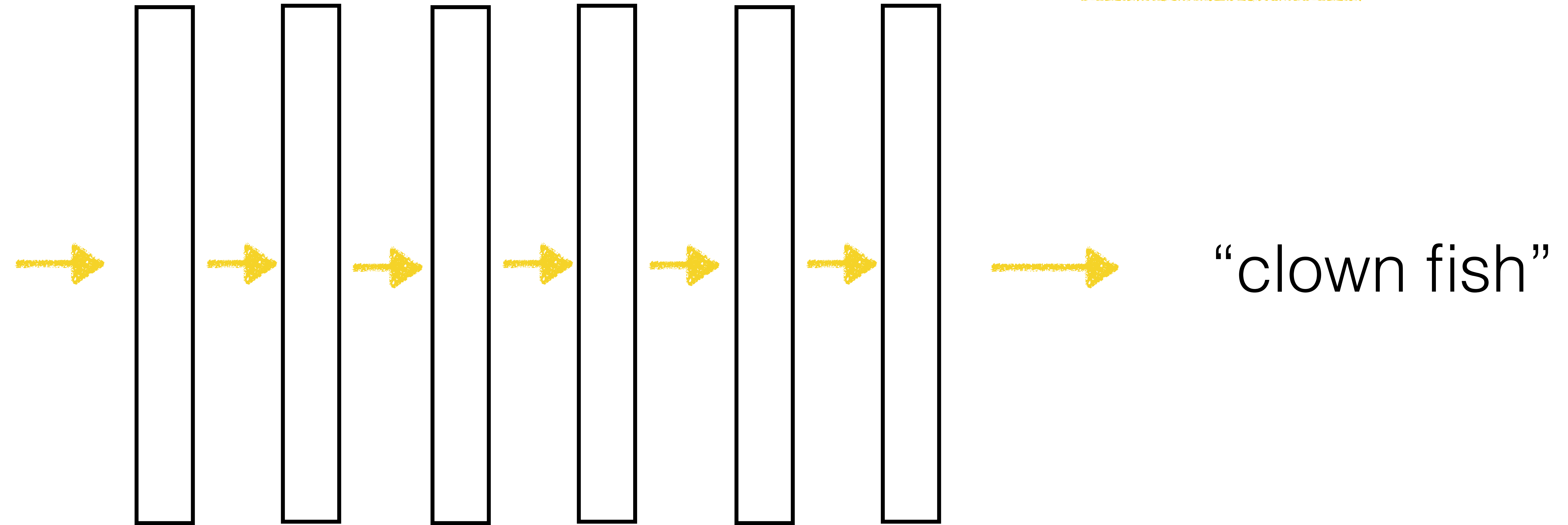
Object recognition





Neural net

Object recognition

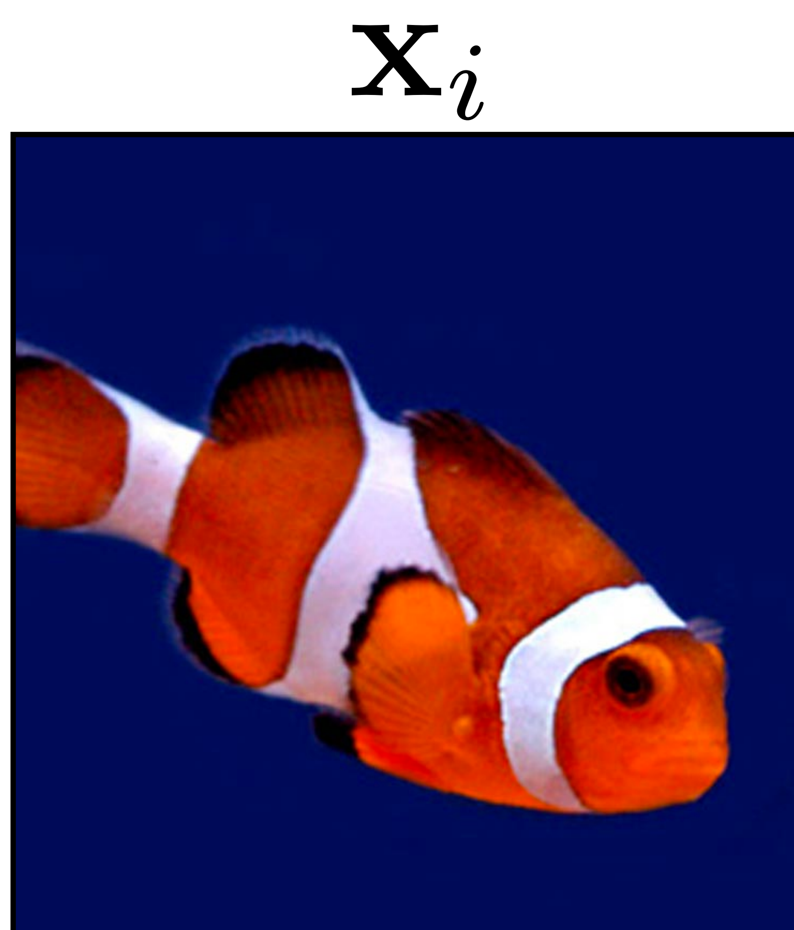


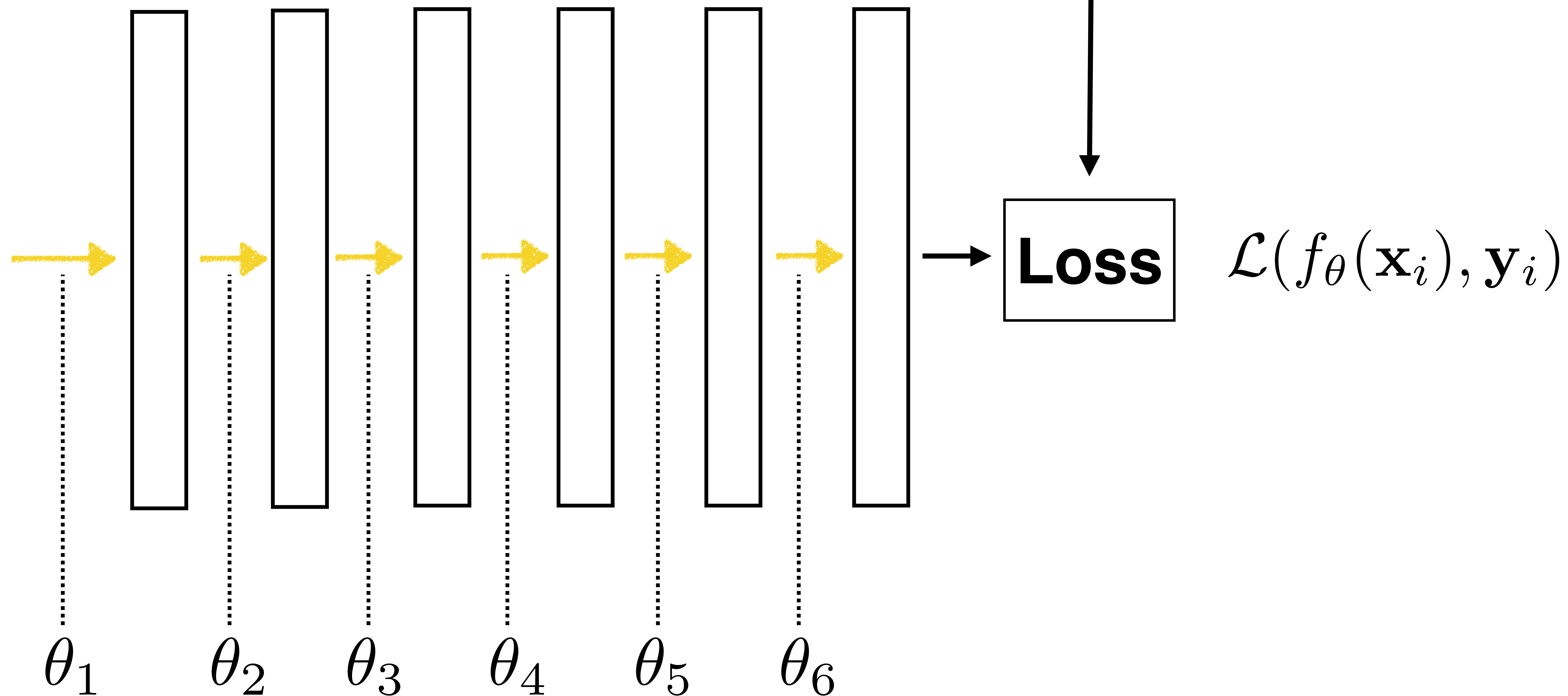
Learned

"clown fish"

Deep neural net

Deep learning



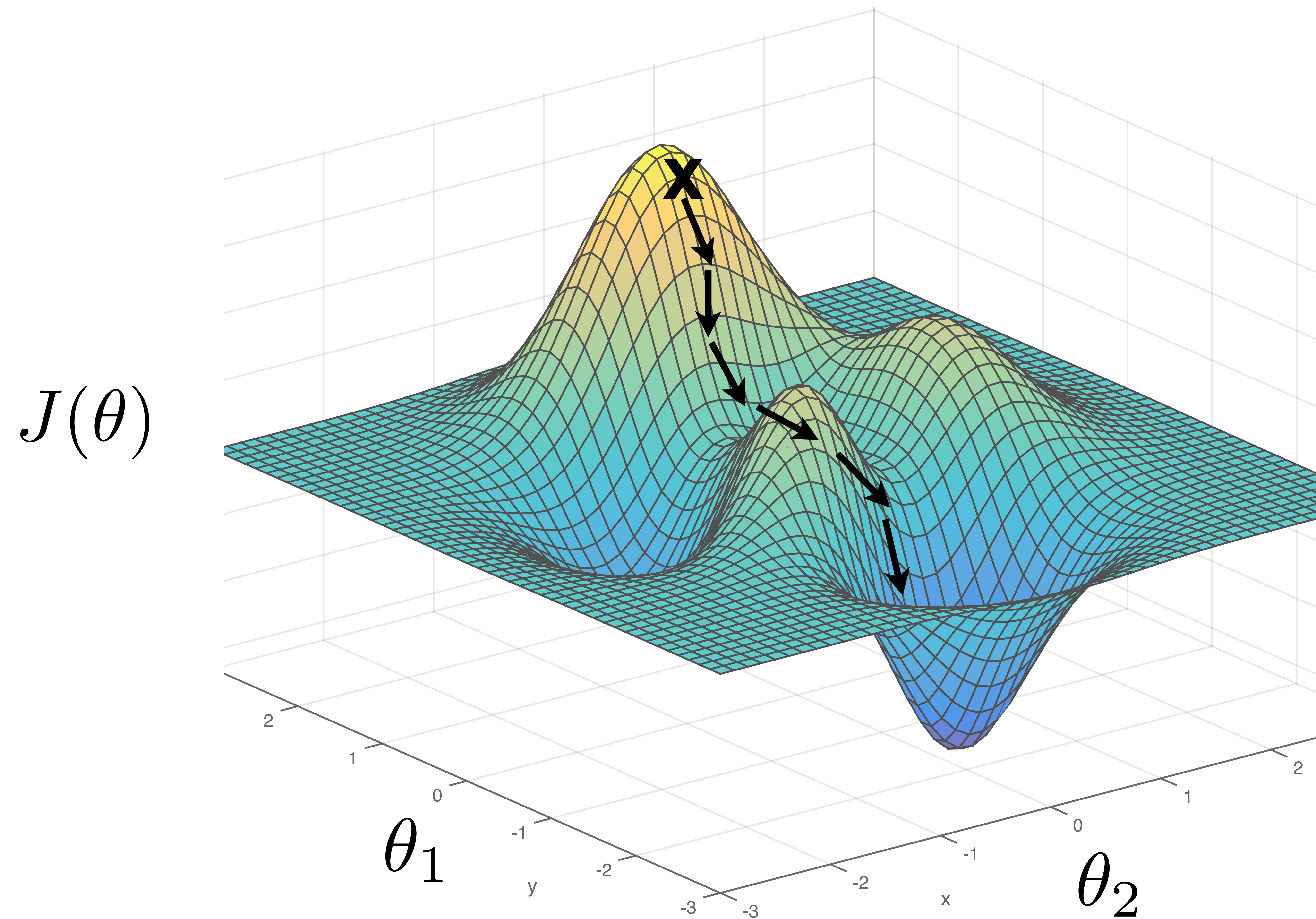


$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)$$

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:

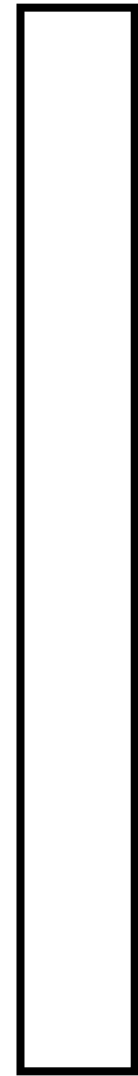
$$\theta^{t+1} = \theta^t - \eta_t \frac{\partial J(\theta)}{\partial \theta}\bigg|_{\theta=\theta^t}$$

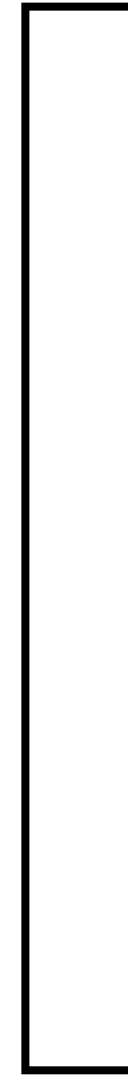
learning rate

Computation in a neural net

Input
representation

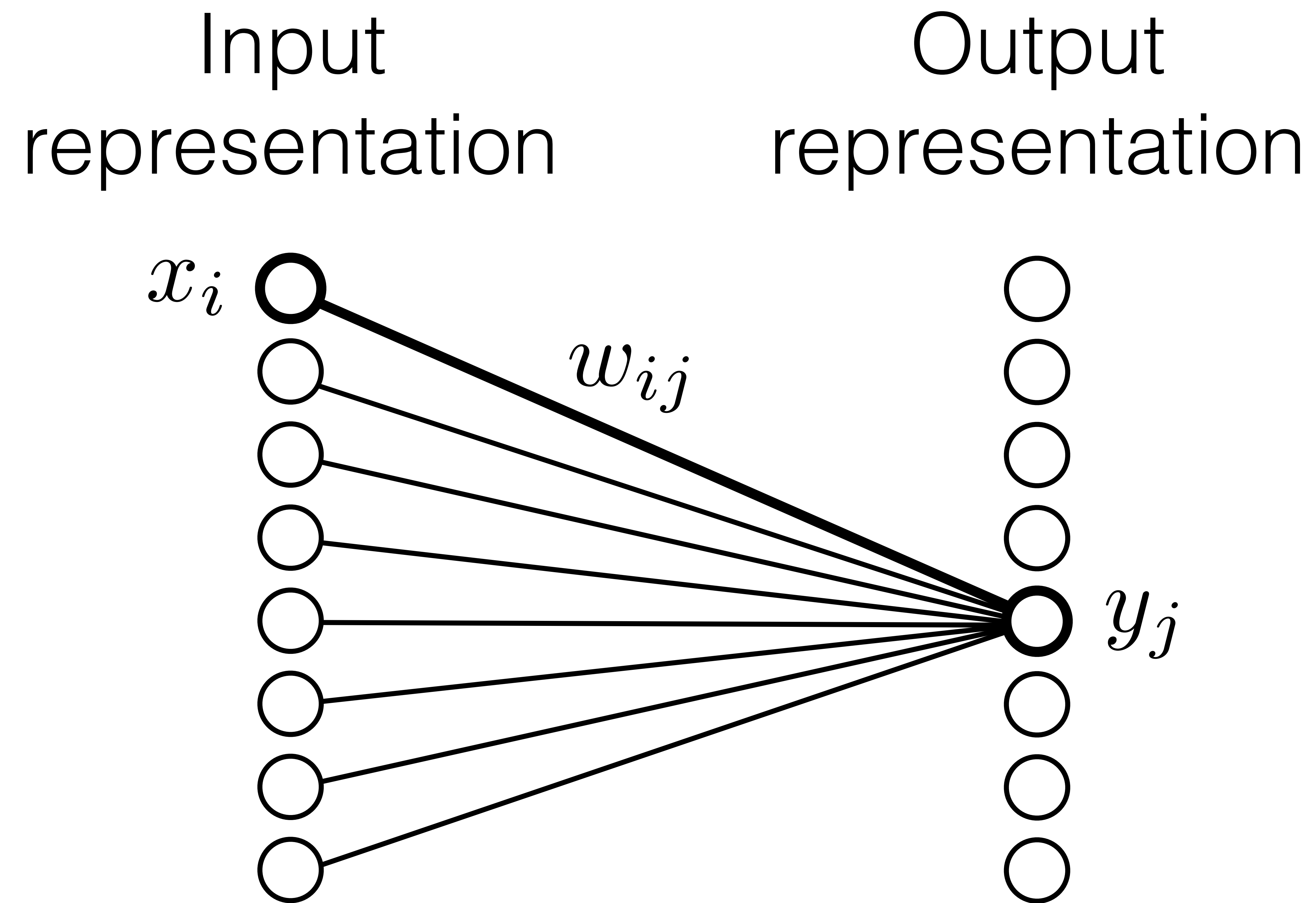
Output
representation





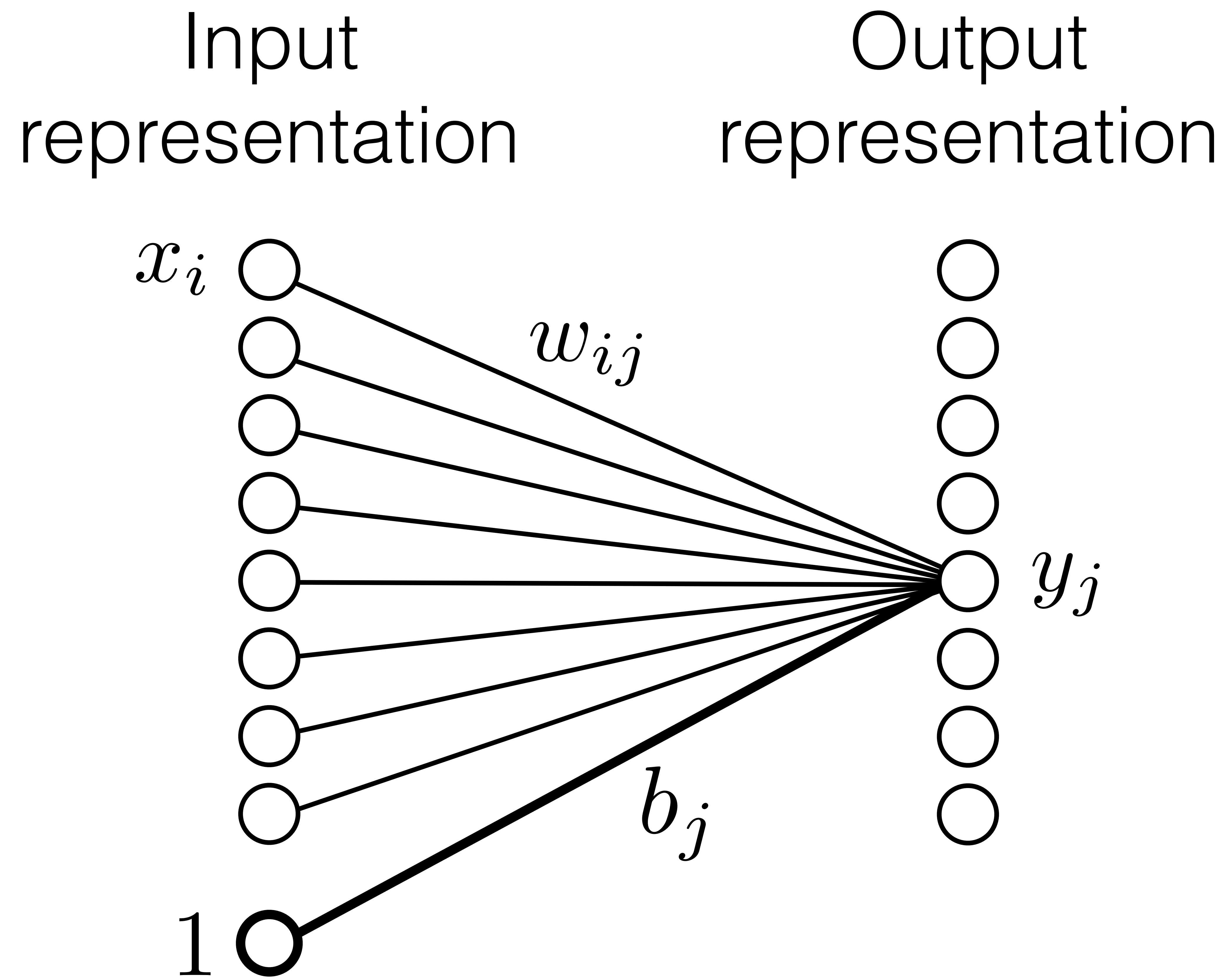
Computation in a neural net

Linear layer



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

Computation in a neural net



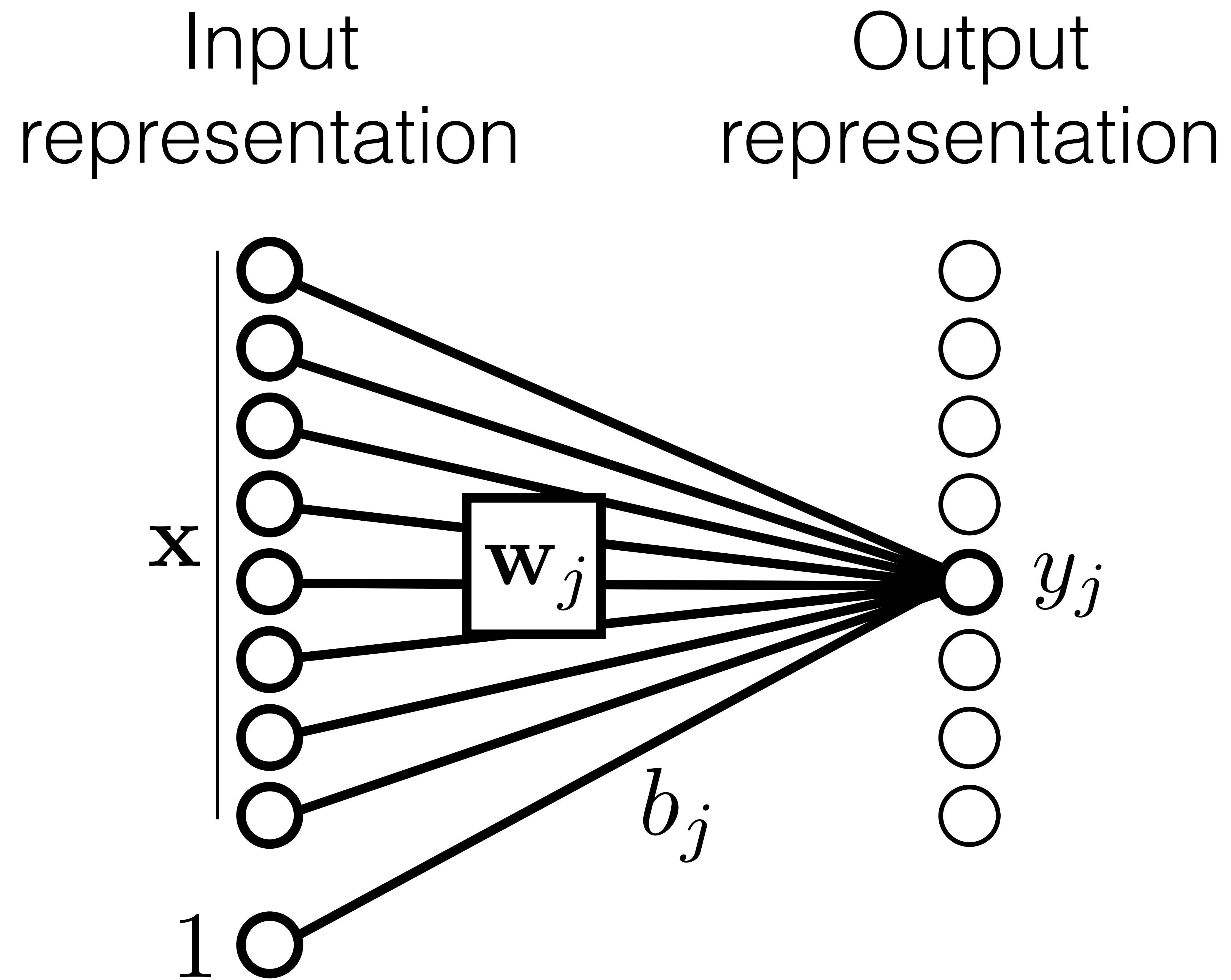
weights

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

bias

Computation in a neural net

Linear layer



weights

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

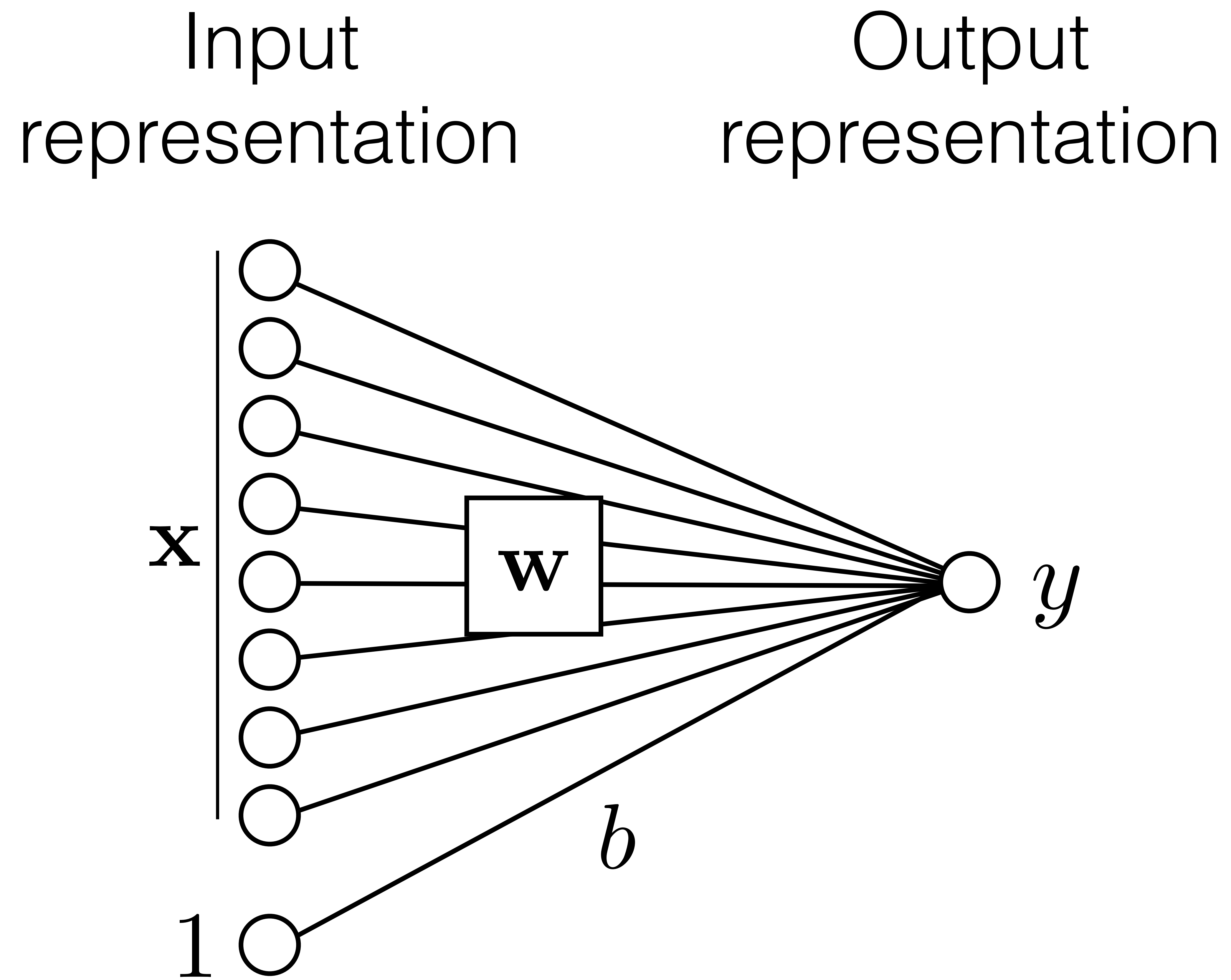
bias

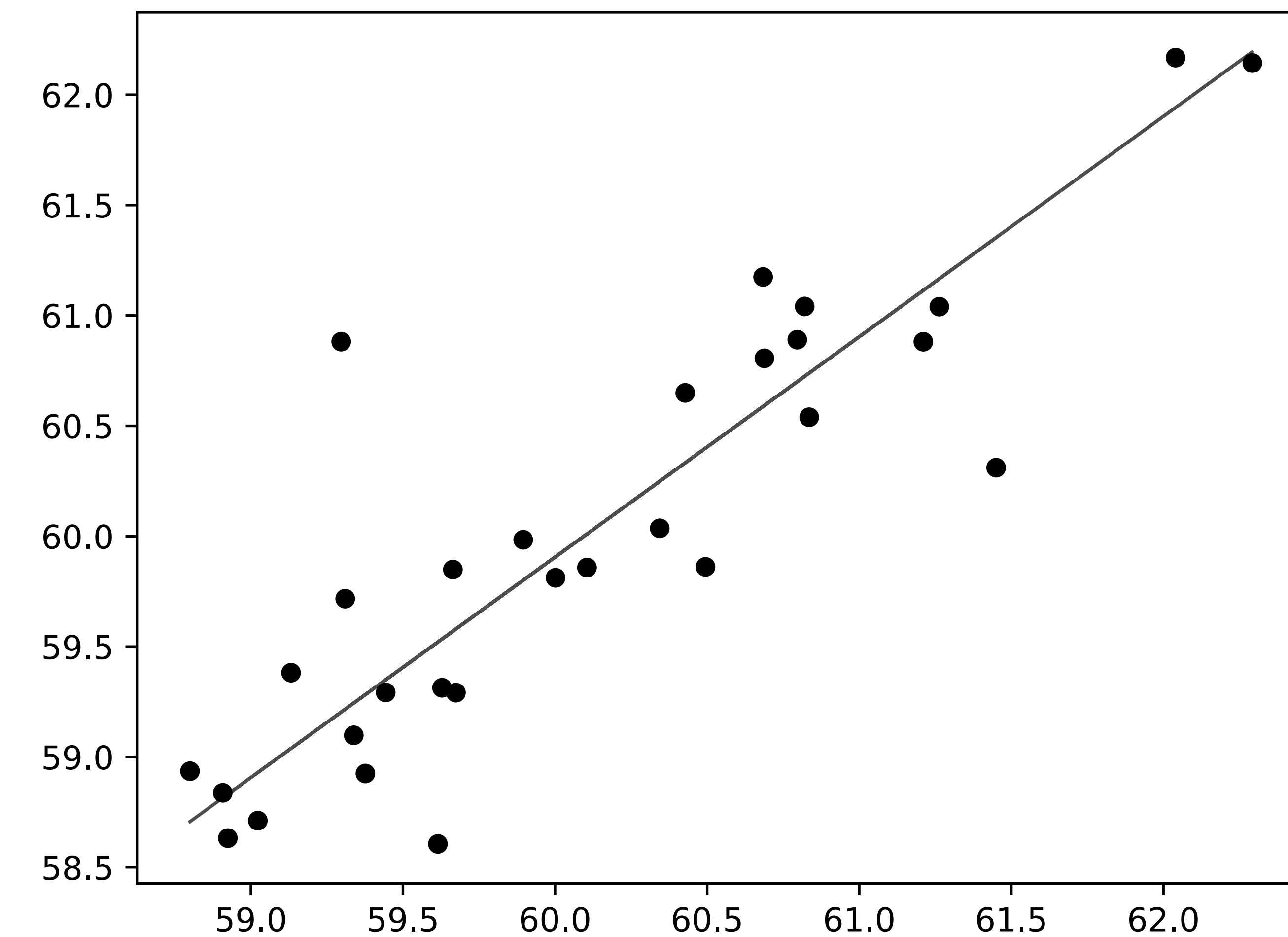
$$\theta = \{\mathbf{W}, \mathbf{b}\}$$

parameters of the model

Example: linear regression with a neural net

Linear layer





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

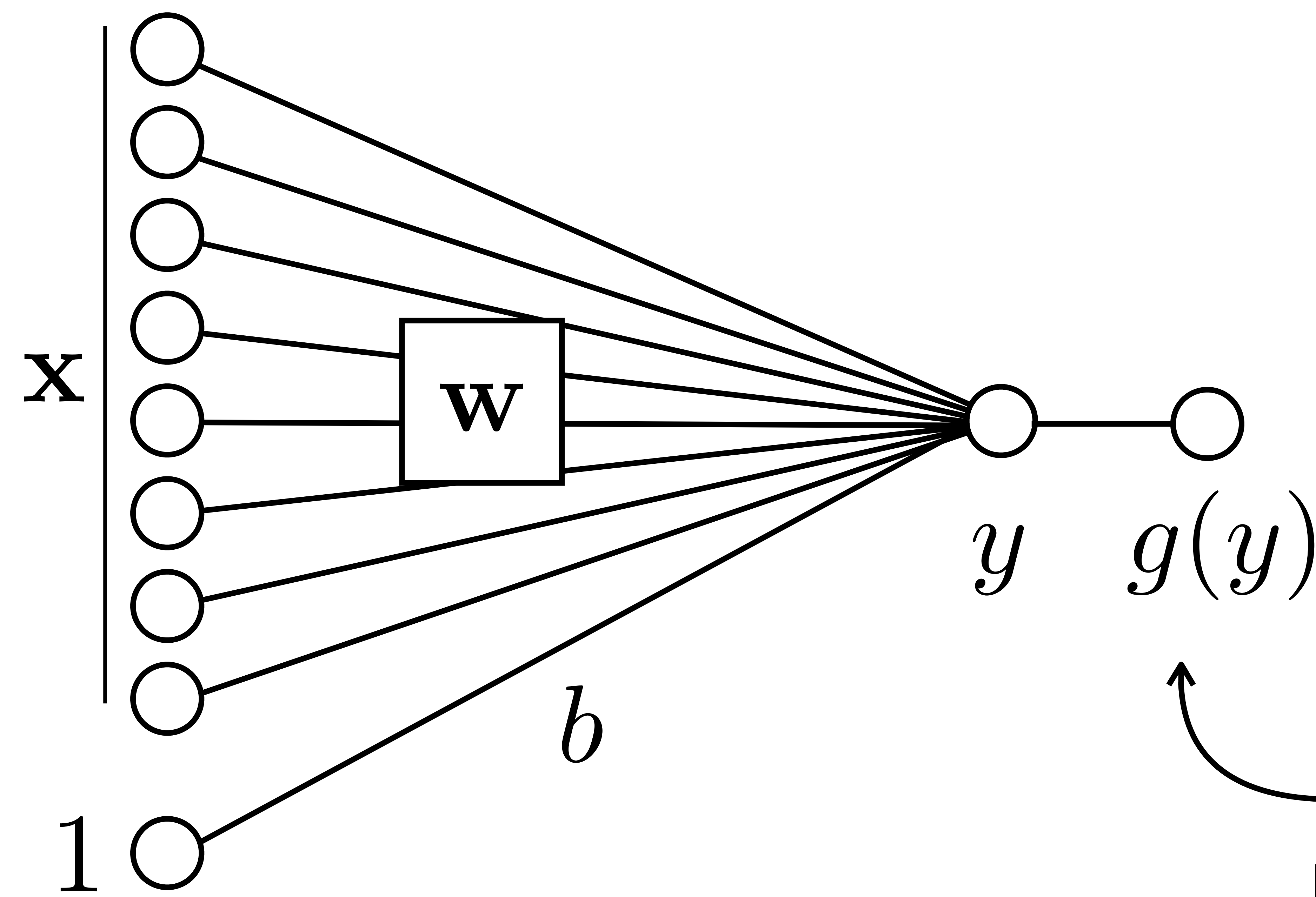
Computation in a neural net

"Perceptron"

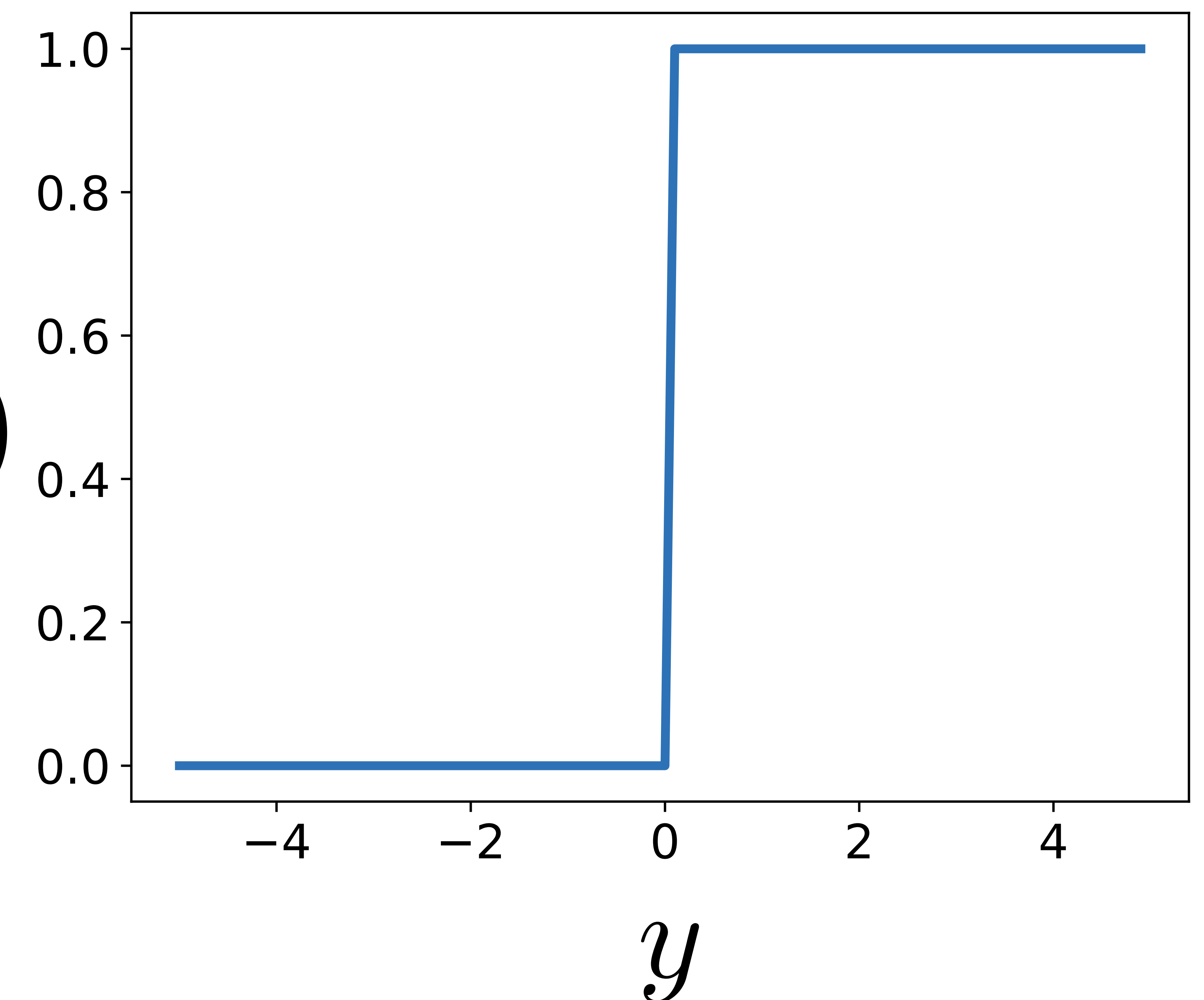
Input representation

Output representation

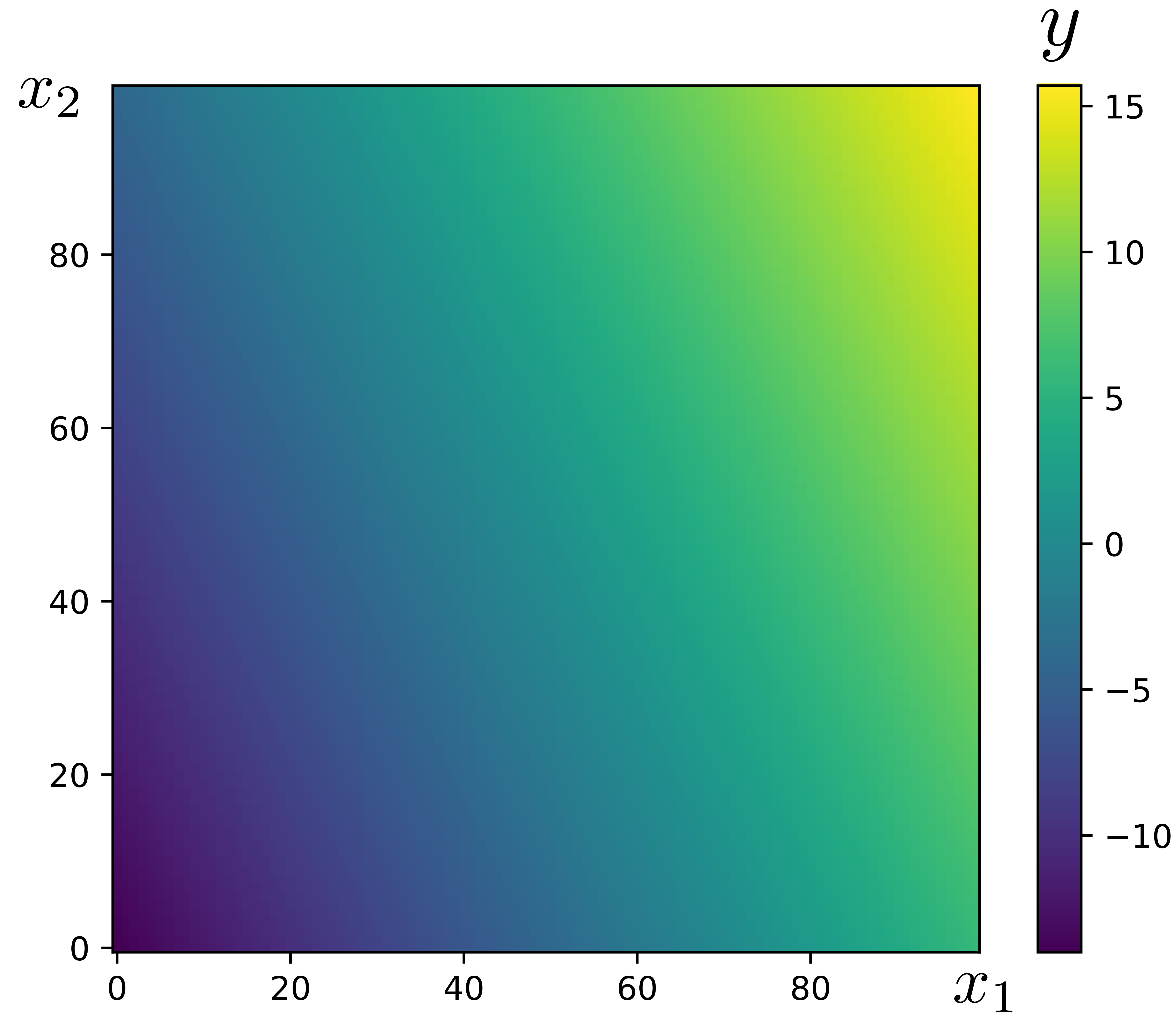
$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$



Pointwise Non-linearity

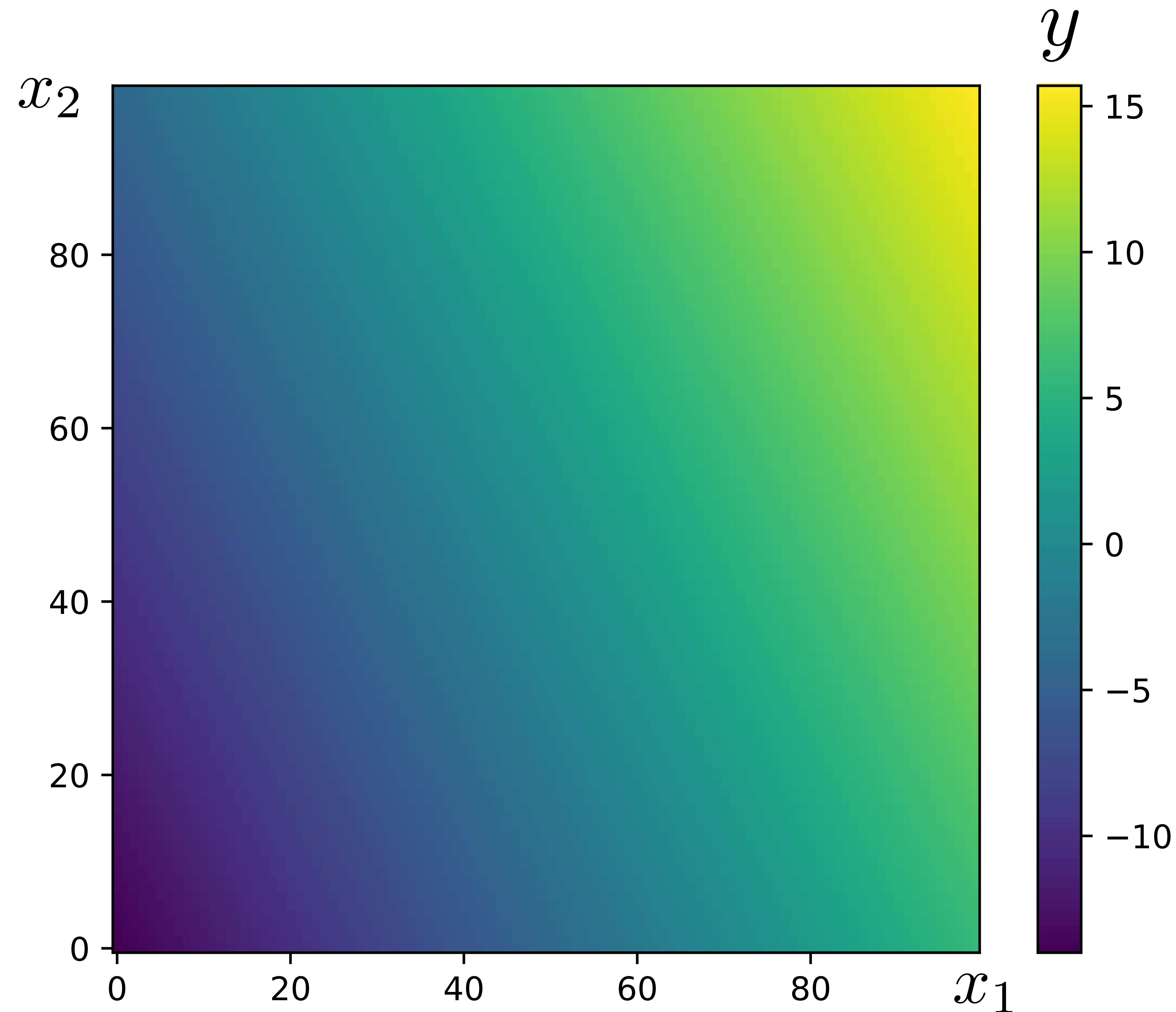


Example: linear classification with a perceptron



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

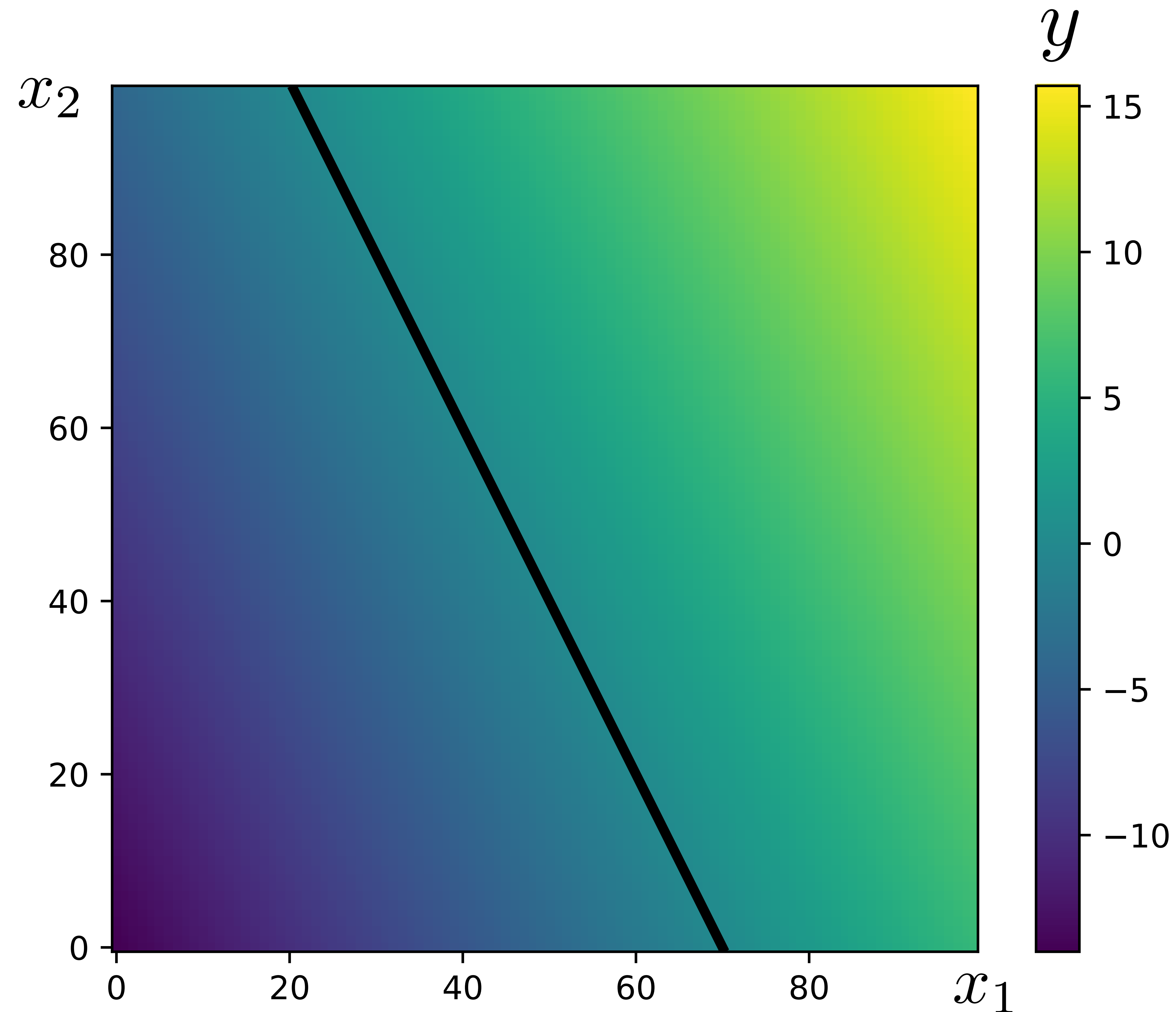
Example: linear classification with a perceptron



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

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

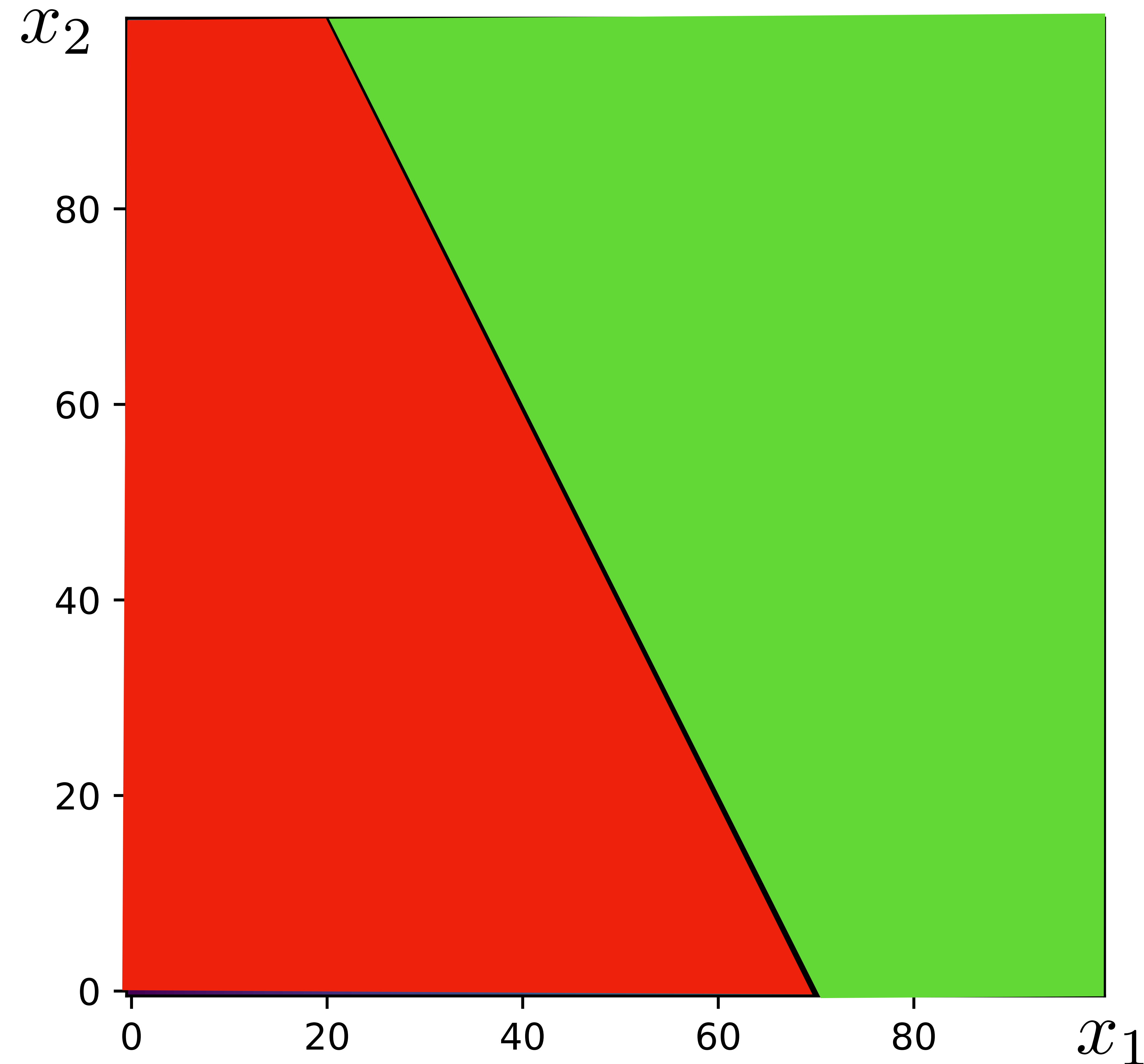
Example: linear classification with a perceptron



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

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

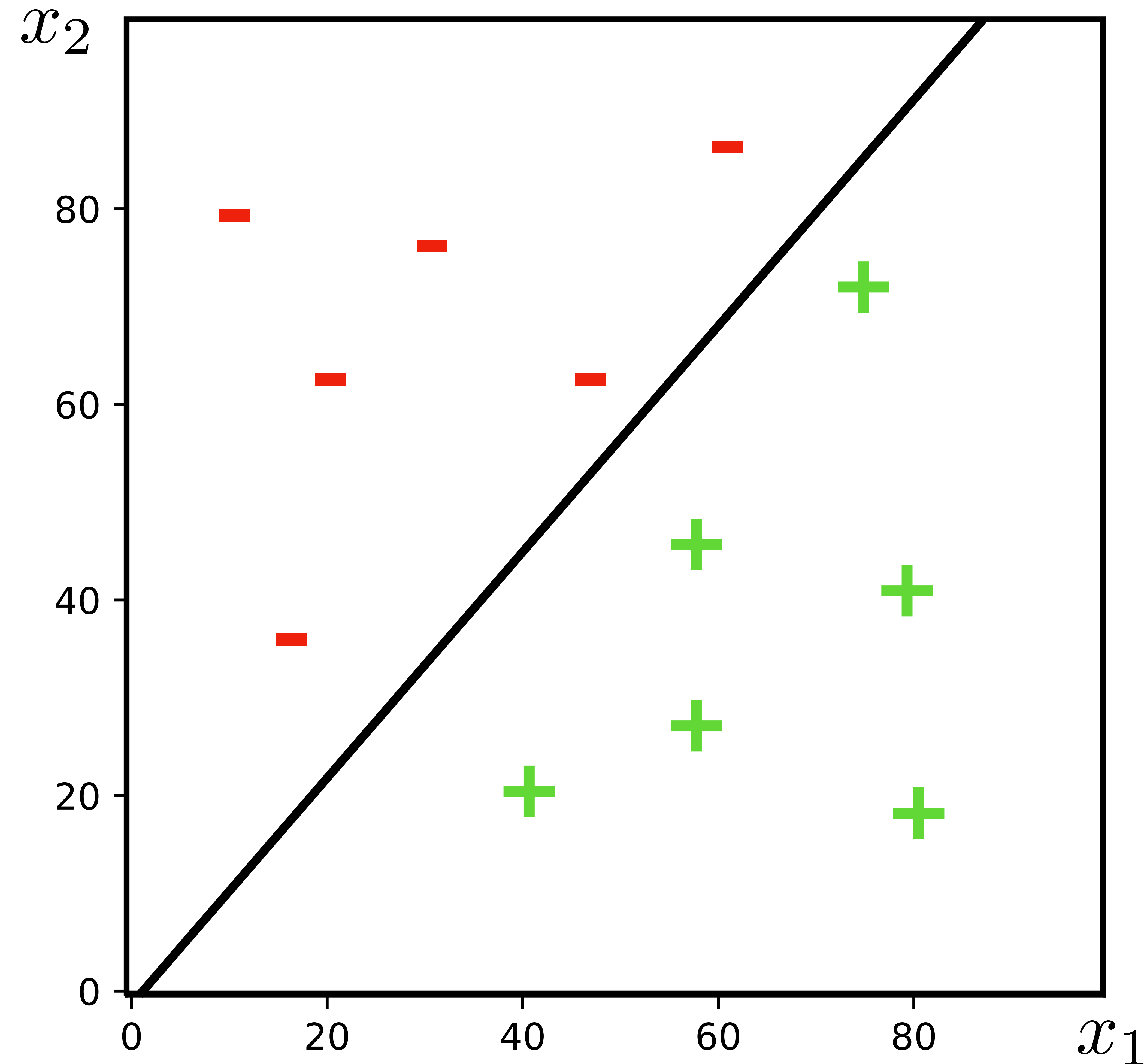
Example: linear classification with a perceptron



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

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

Example: linear classification with a perceptron

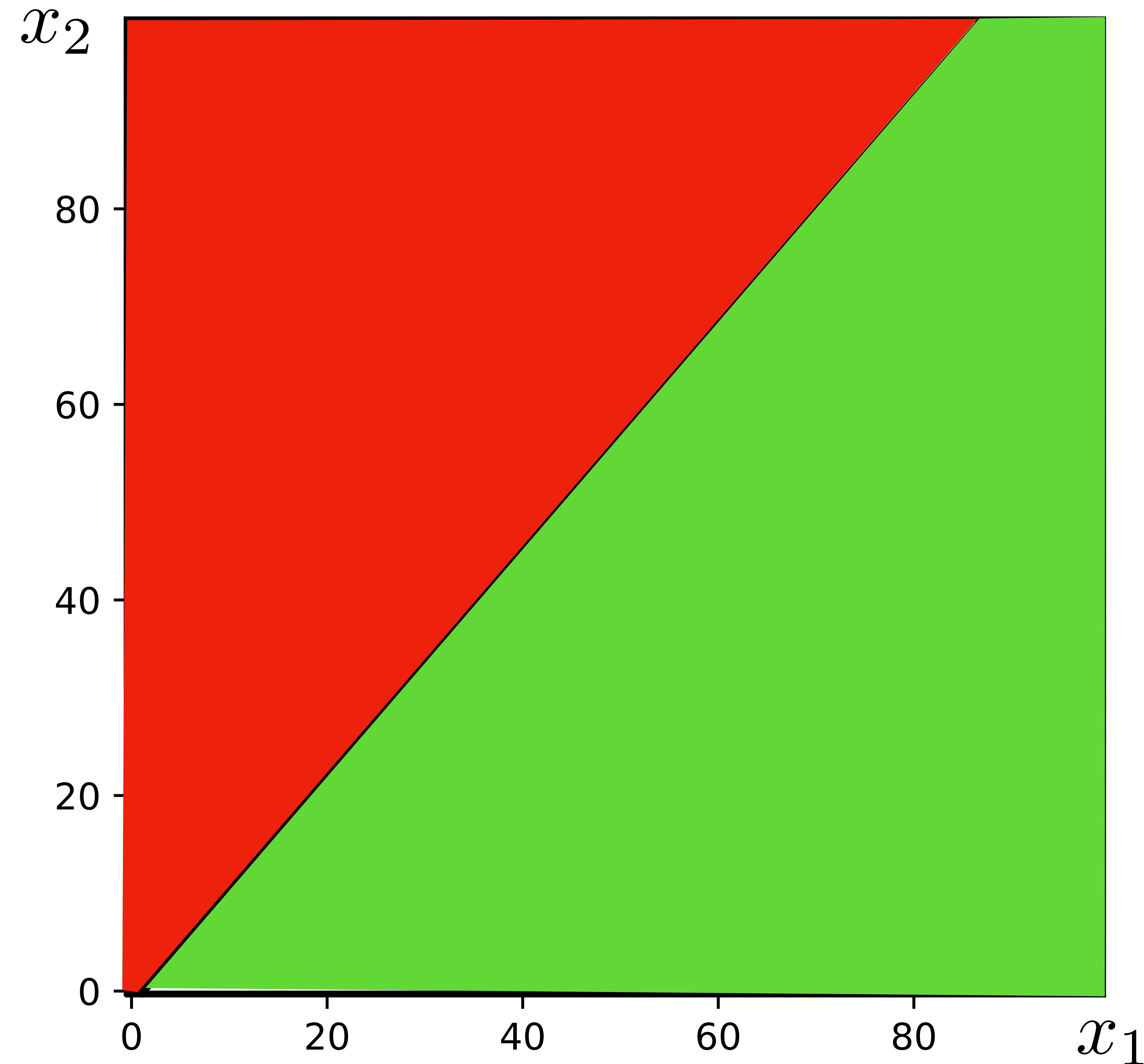


$$\hat{y} = \mathbf{x}^T \mathbf{w} + b$$

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

$$\mathbf{w}^*, b^* = \arg\min_{\mathbf{w}, b} \mathcal{L}(g(\hat{y}), y_i)$$

Example: linear classification with a perceptron



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

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

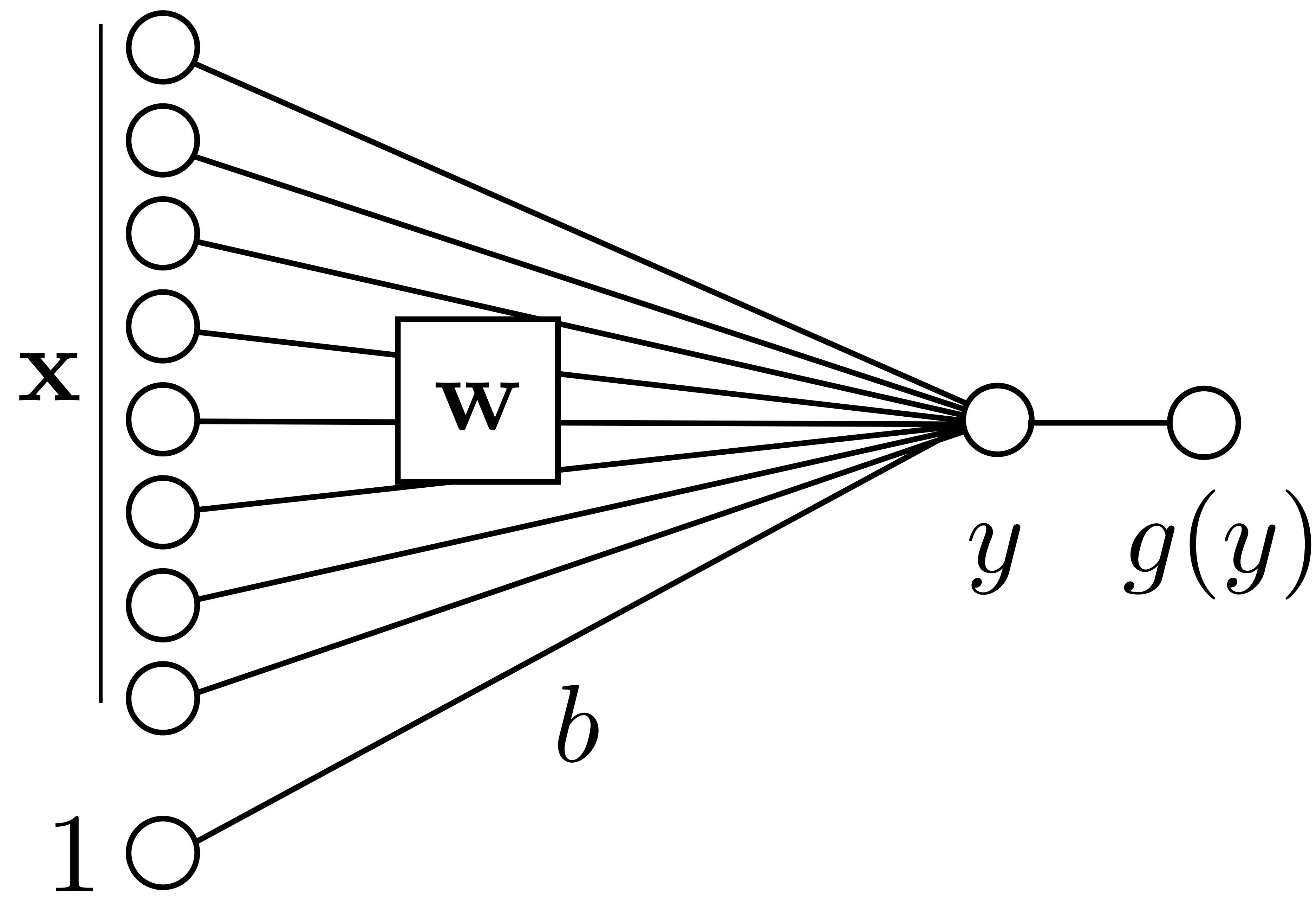
$$\mathbf{w}^*, b^* = \arg\min_{\mathbf{w}, b} \mathcal{L}(g(\hat{y}), y_i)$$

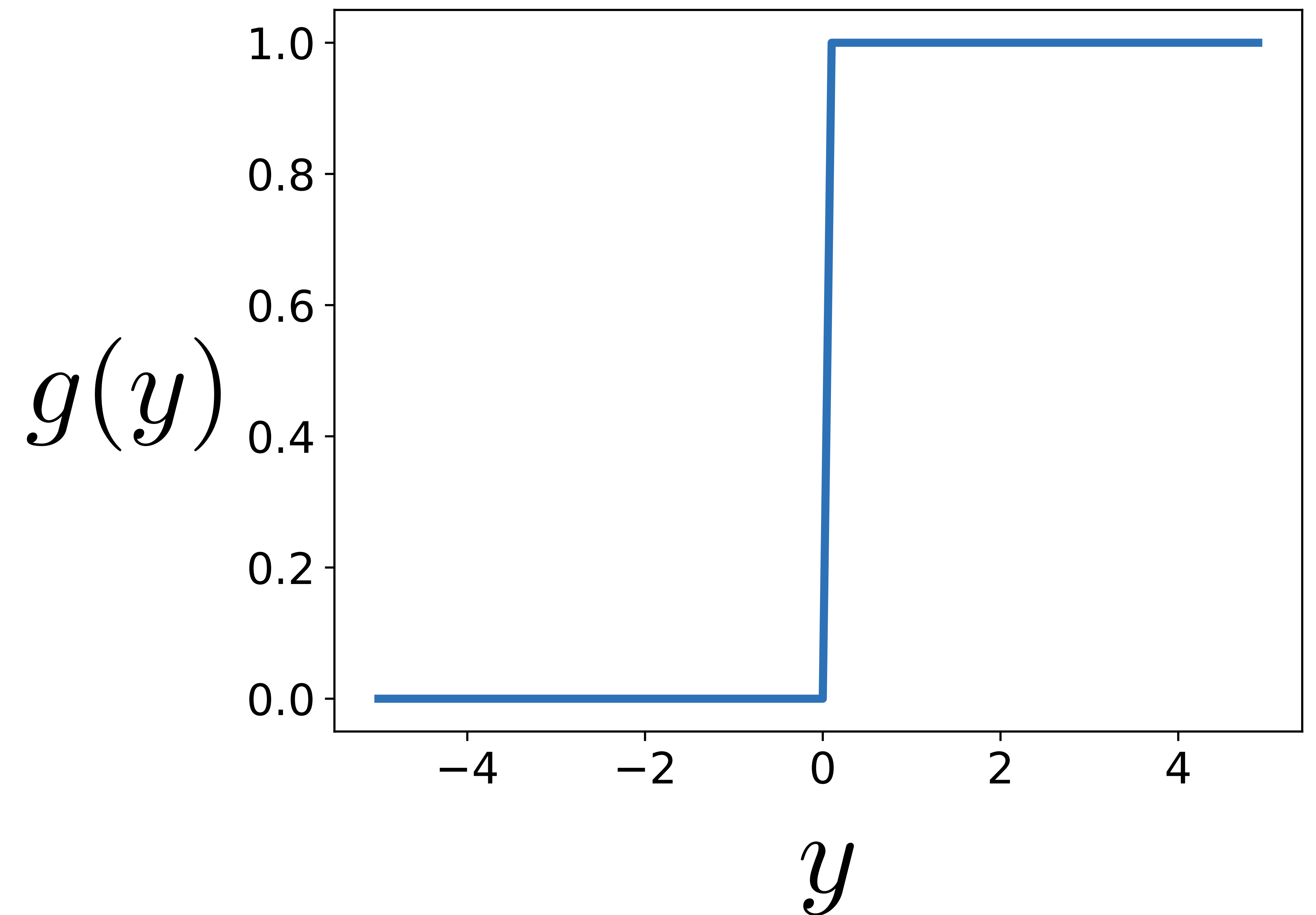
Computation in a neural net

Input representation

Output representation

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

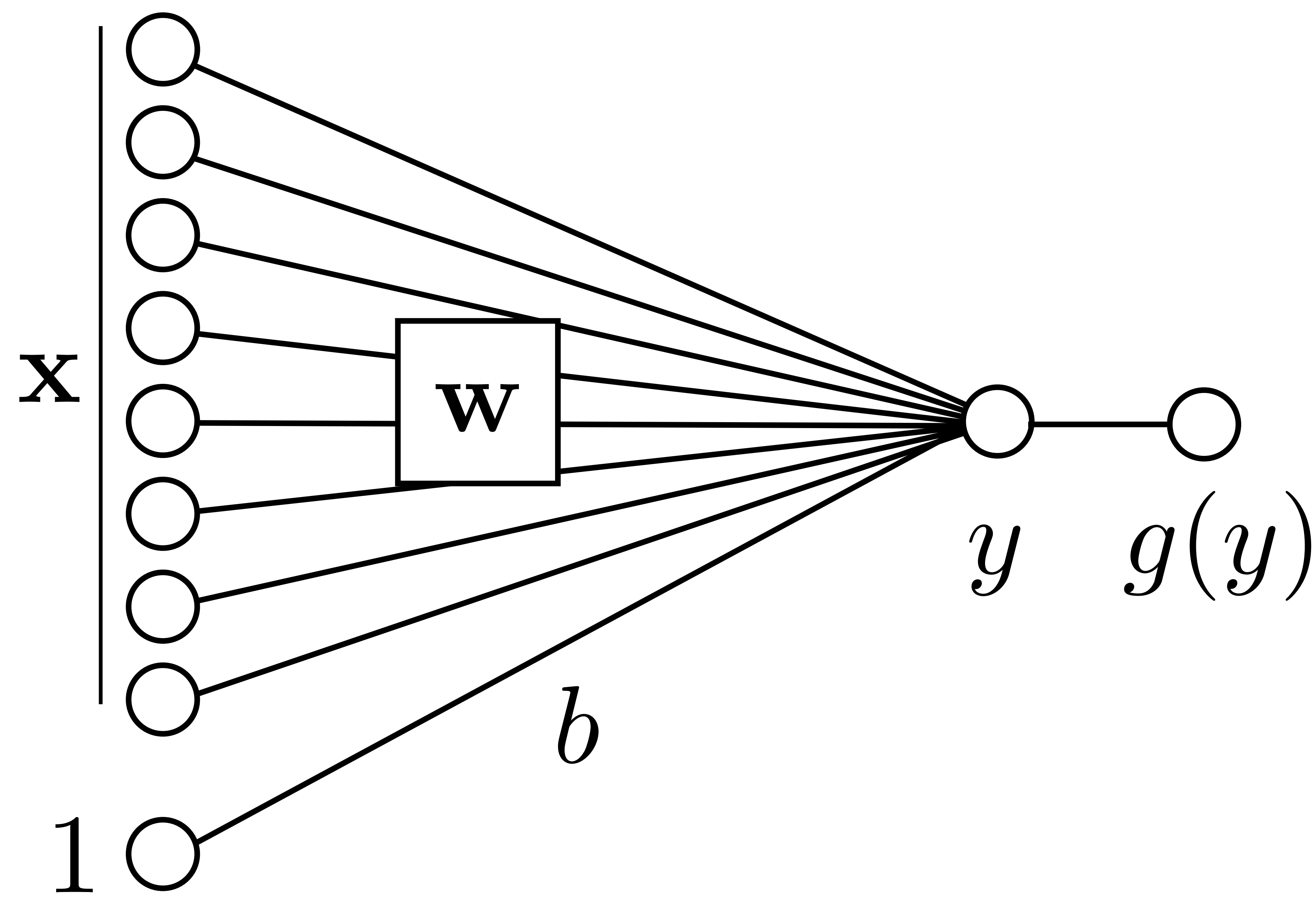




Computation in a neural net — nonlinearity

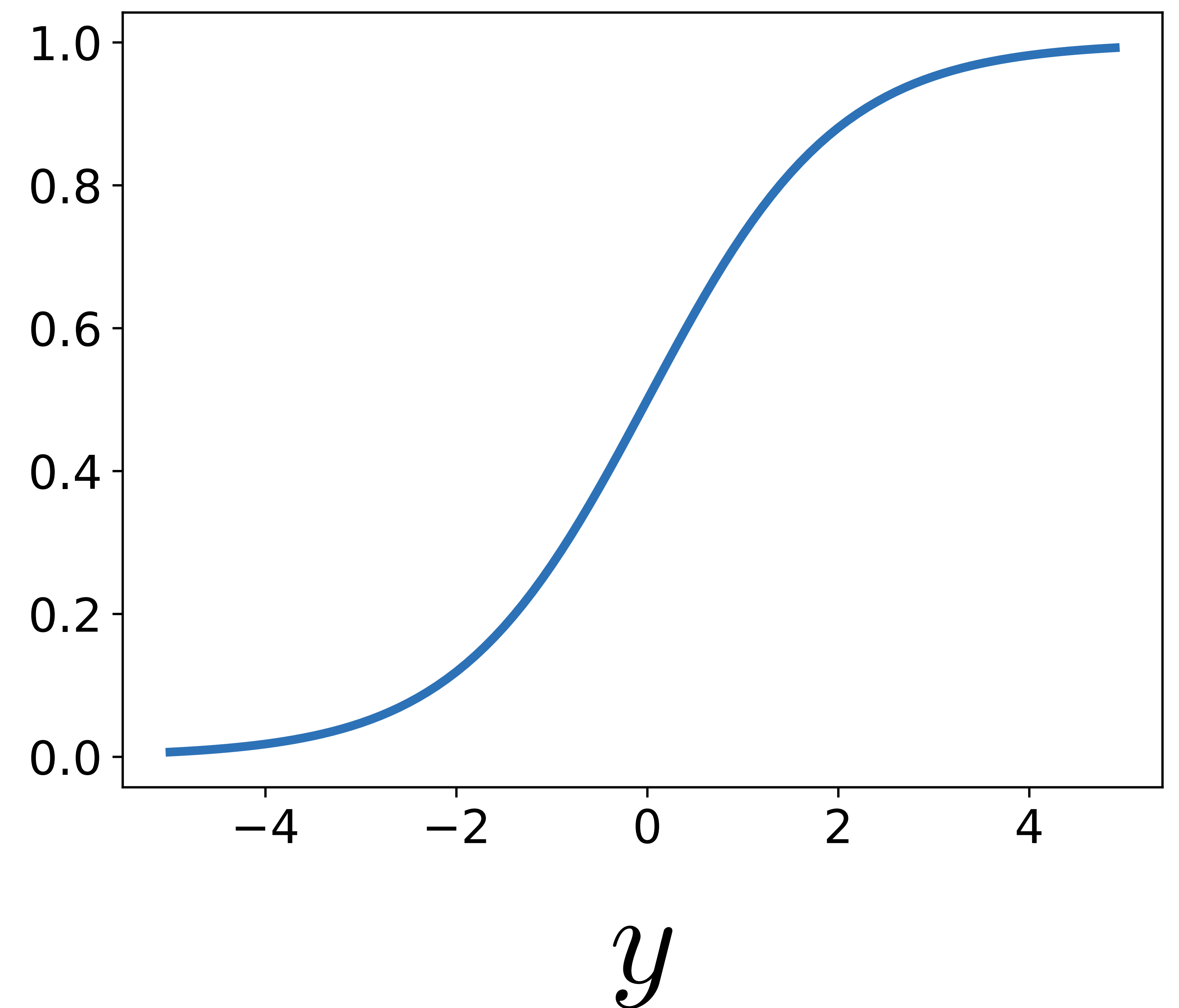
Input representation

Output representation



Sigmoid

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

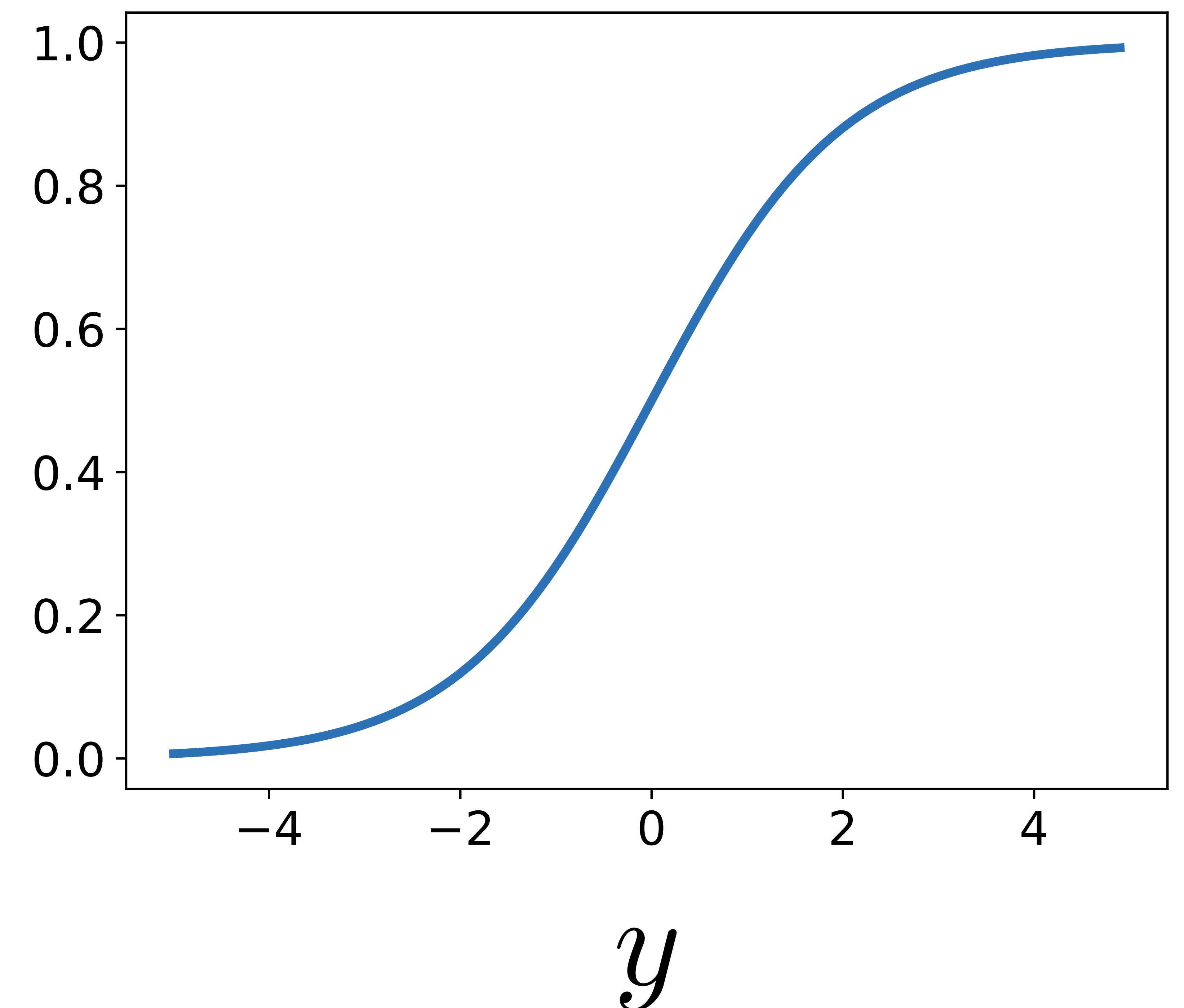


Computation in a neural net — nonlinearity

- 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

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



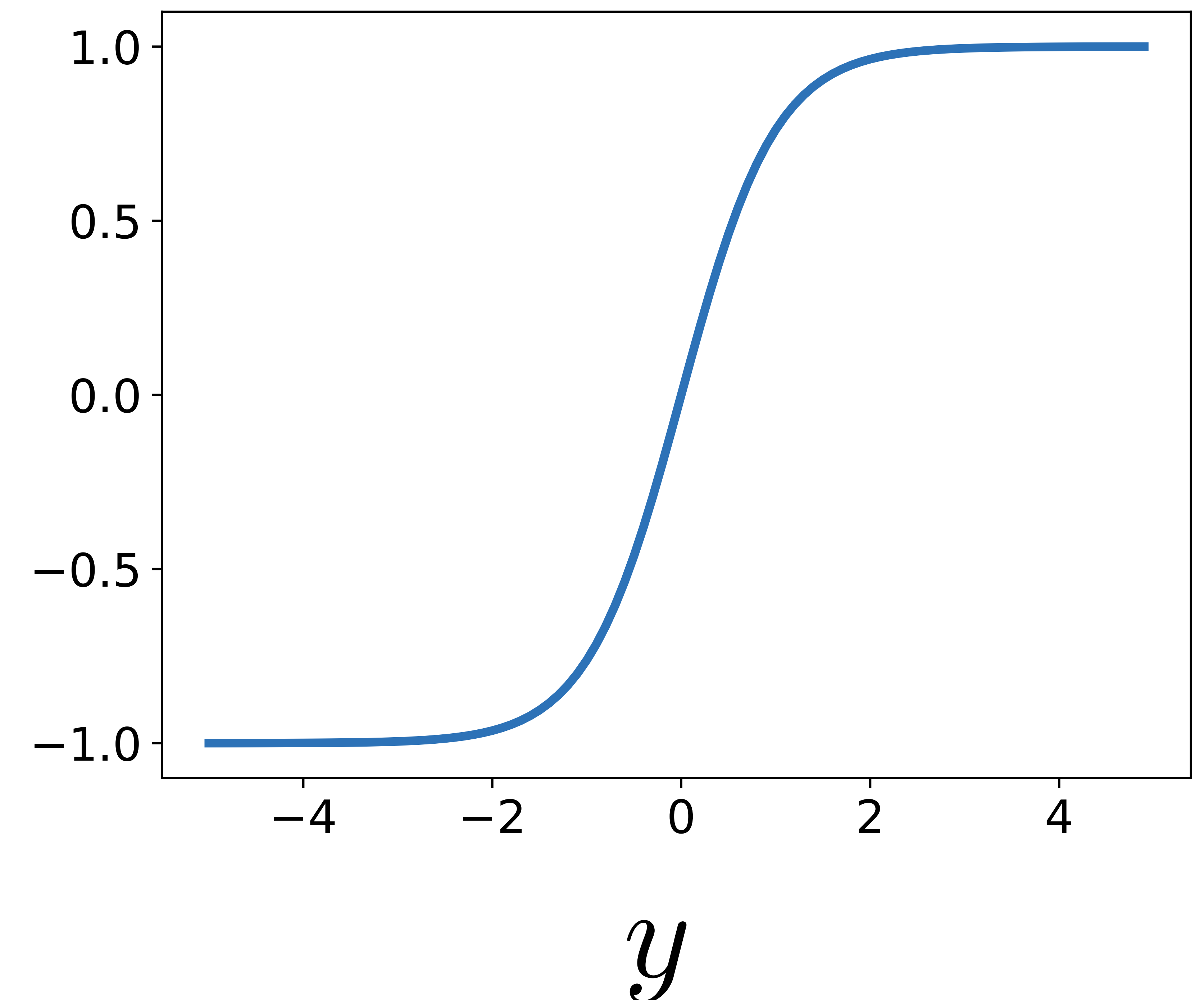
Computation in a neural net — nonlinearity

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

tanh(x) = 2 sigmoid(2x) –1

Tanh

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



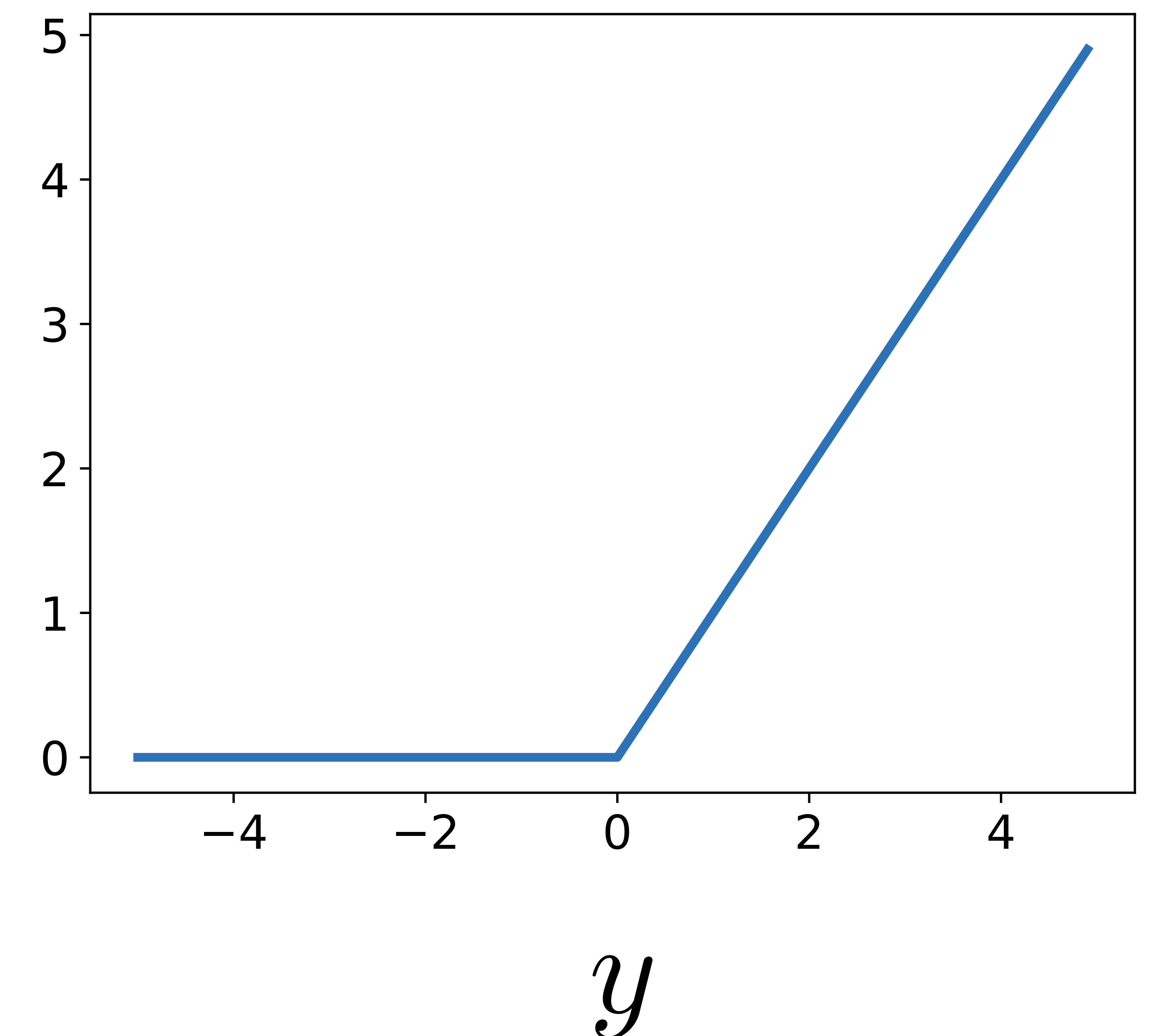
Computation in a neural net — nonlinearity

- 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 (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)

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

$g(y)$



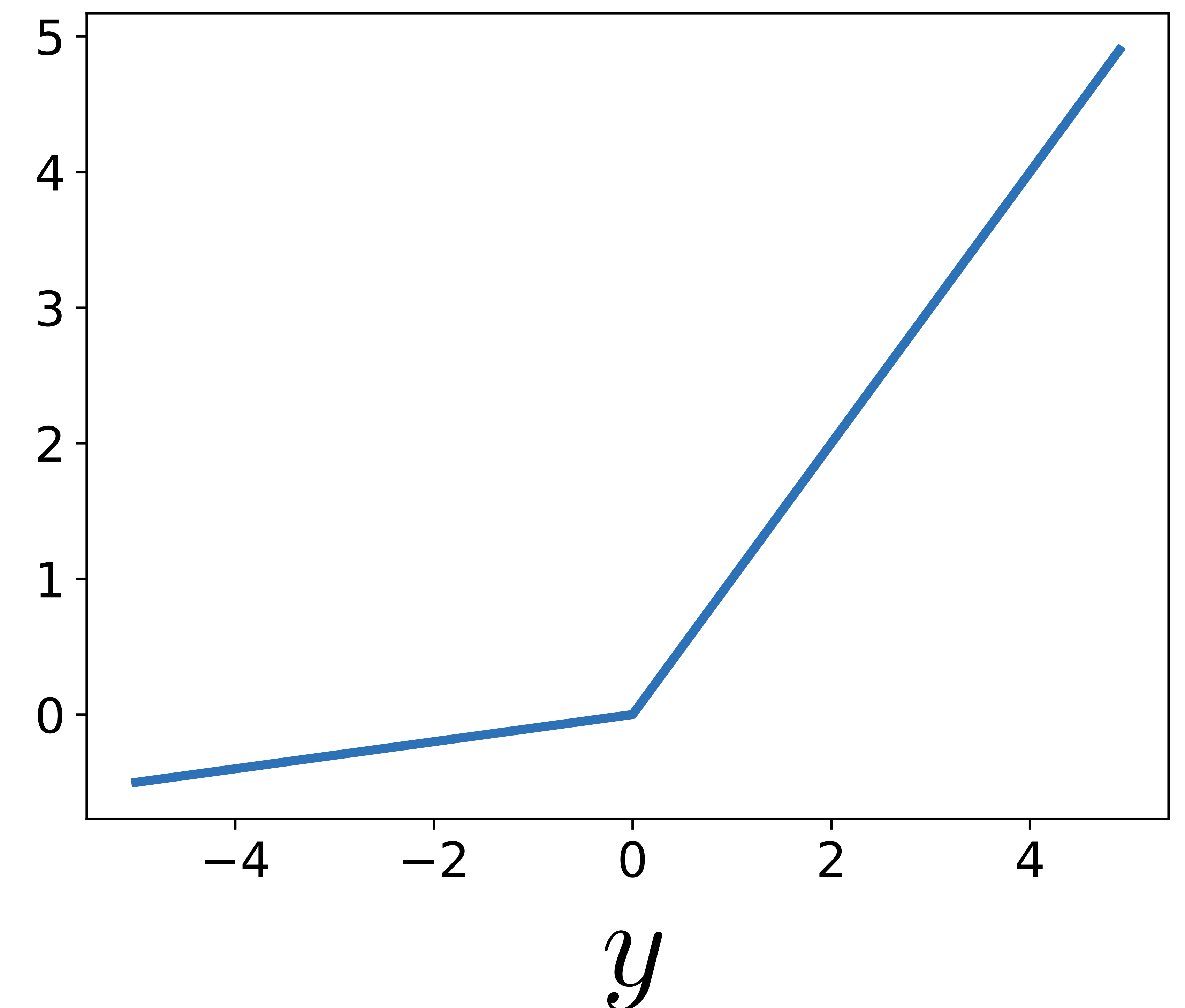
Computation in a neural net — nonlinearity

Leaky ReLU

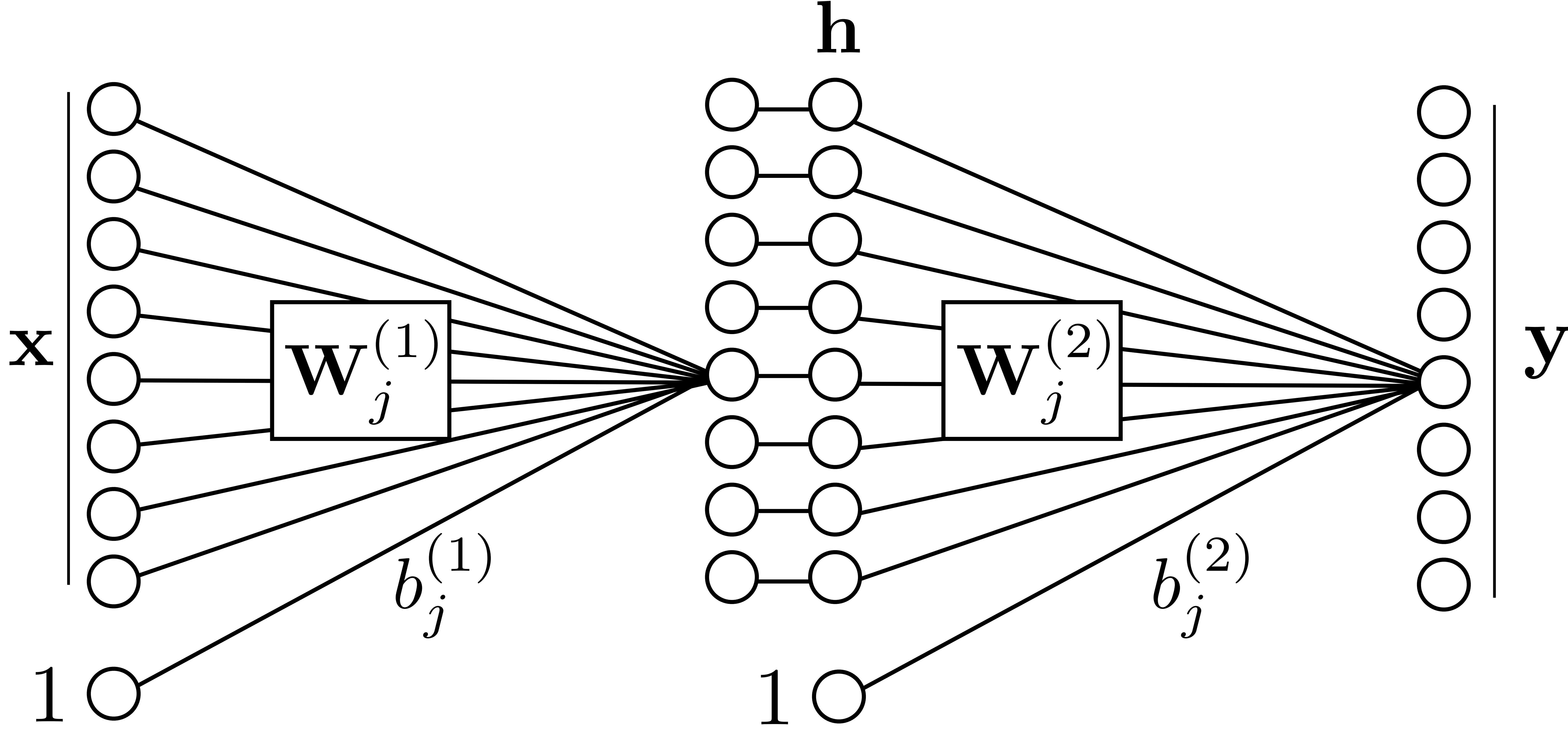
$$g(y) = \begin{cases} \max(0, y), & \text{if} \quad y \geq 0 \\ a \min(0, y), & \text{if} \quad y < 0 \end{cases}$$

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

$g(y)$

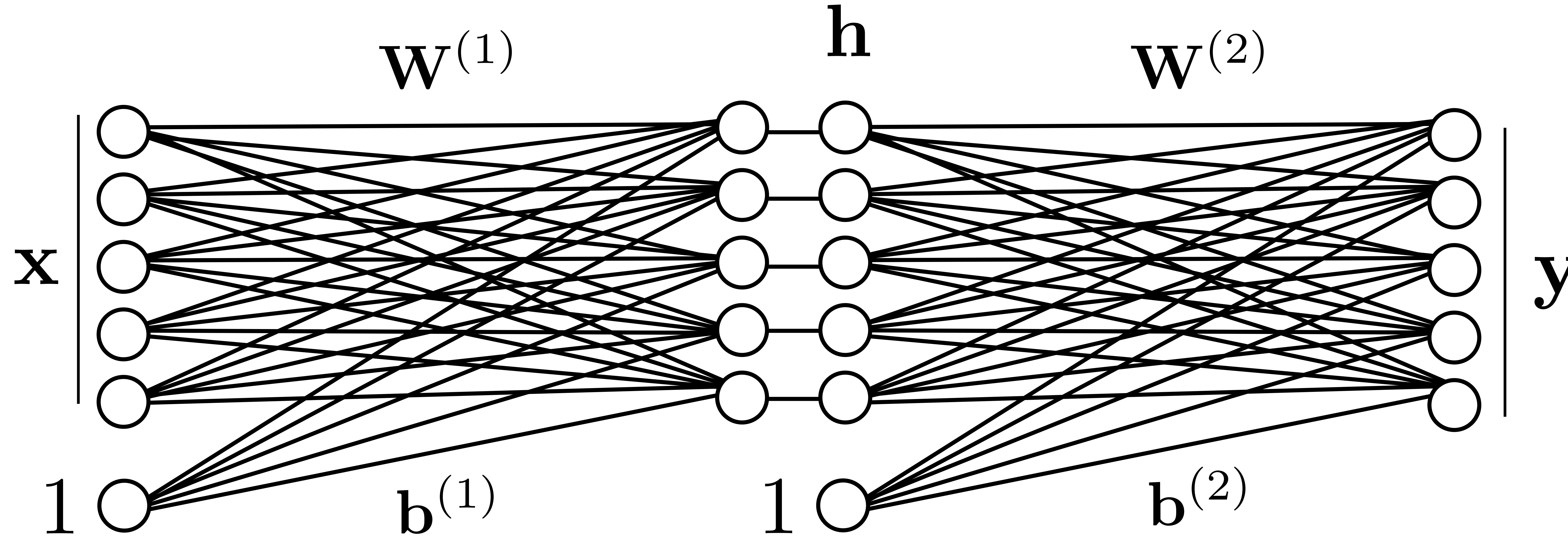


Stacking layers



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

Stacking layers



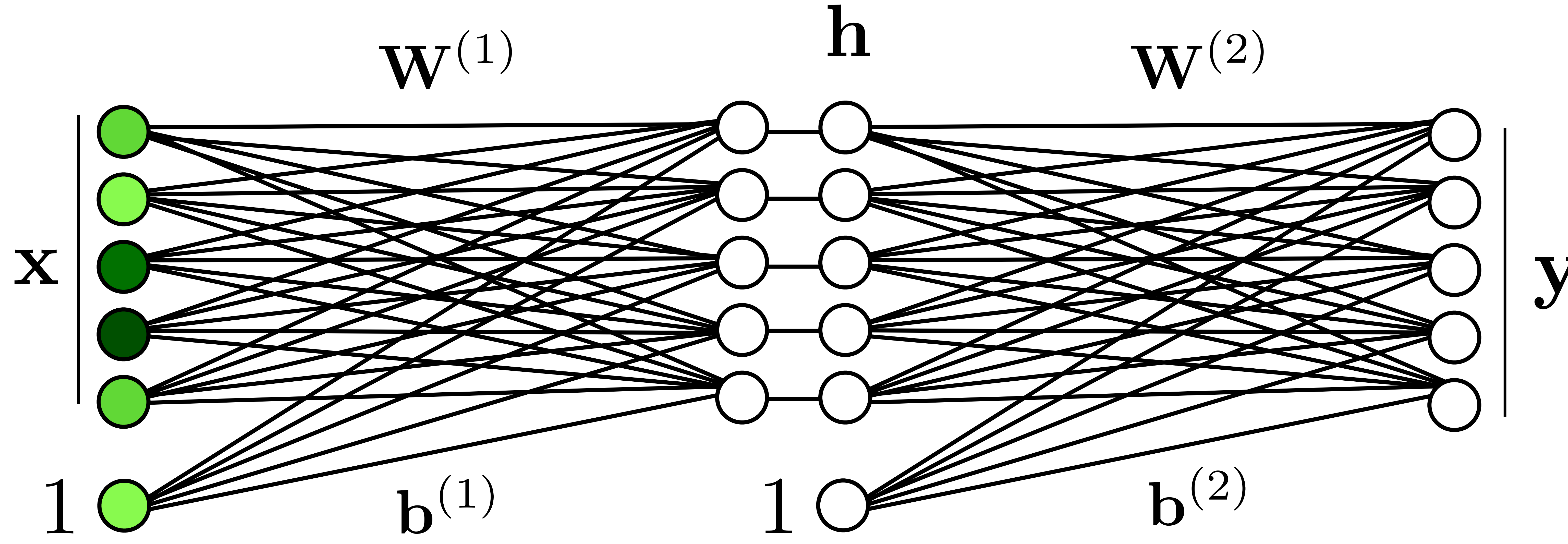
$$\theta = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}$$

Stacking layers

Input representation

Intermediate representation

Output representation



positive

negative

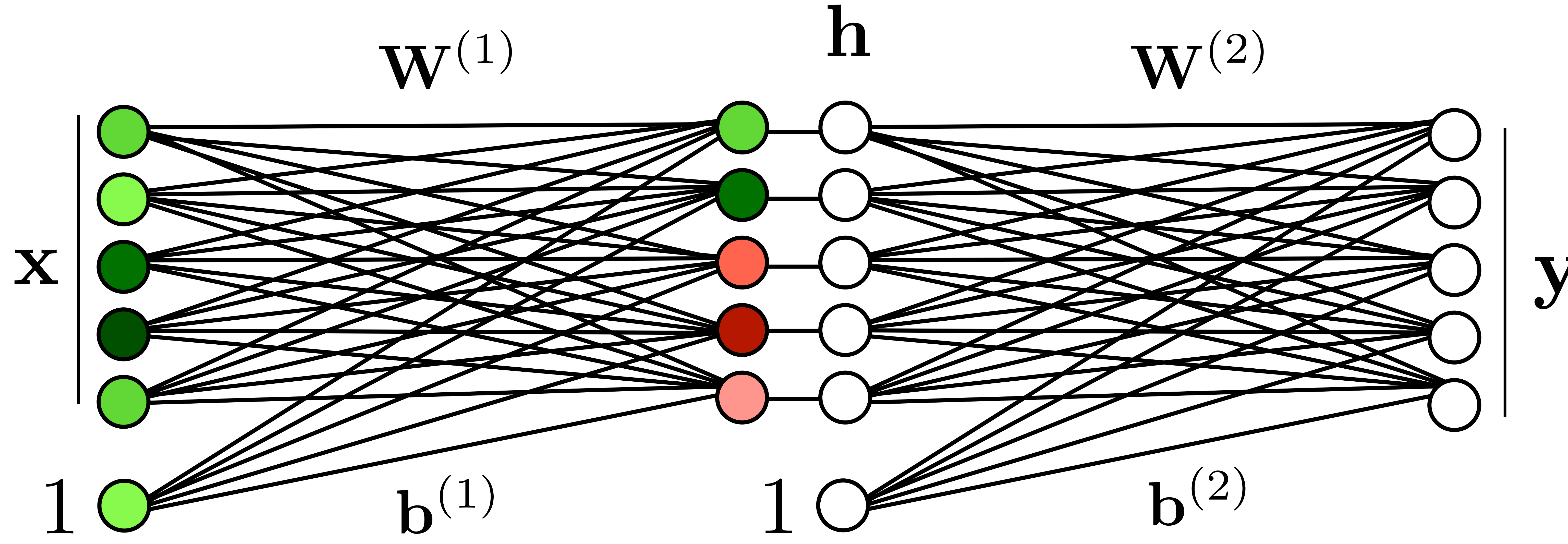
$$\theta = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}$$

Stacking layers

Input representation

Intermediate representation

Output representation



positive

negative

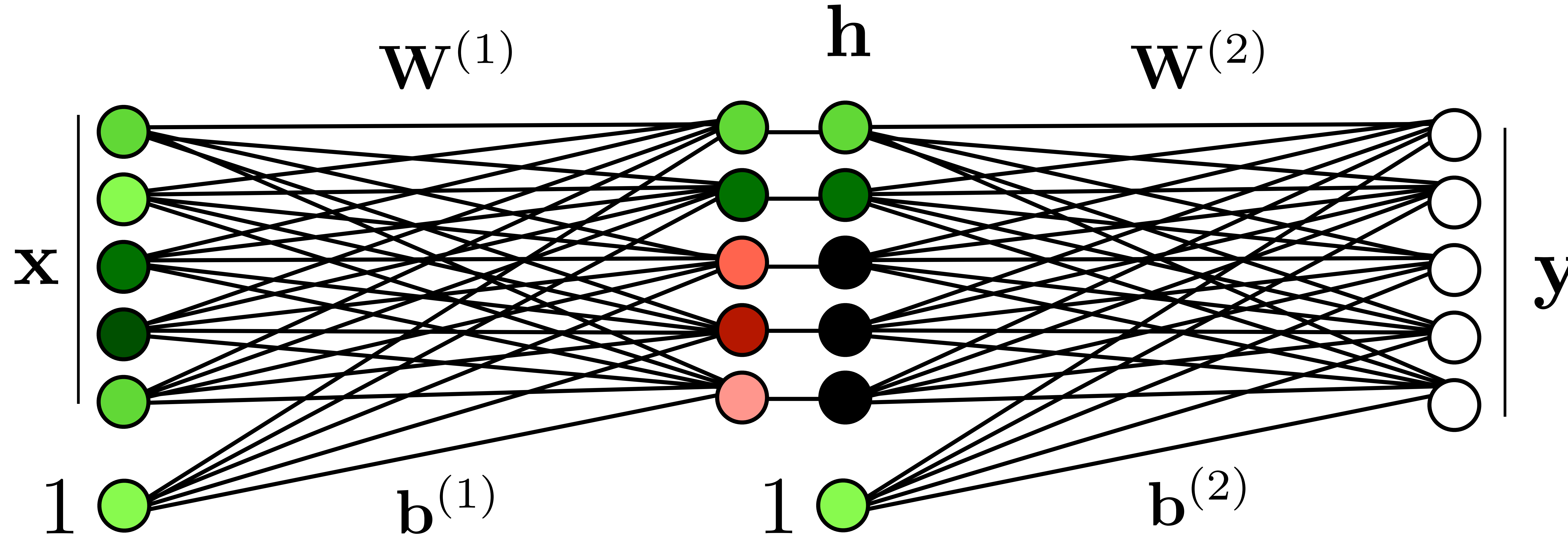
$$\theta = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}$$

Stacking layers

Input representation

Intermediate representation

Output representation



positive

negative

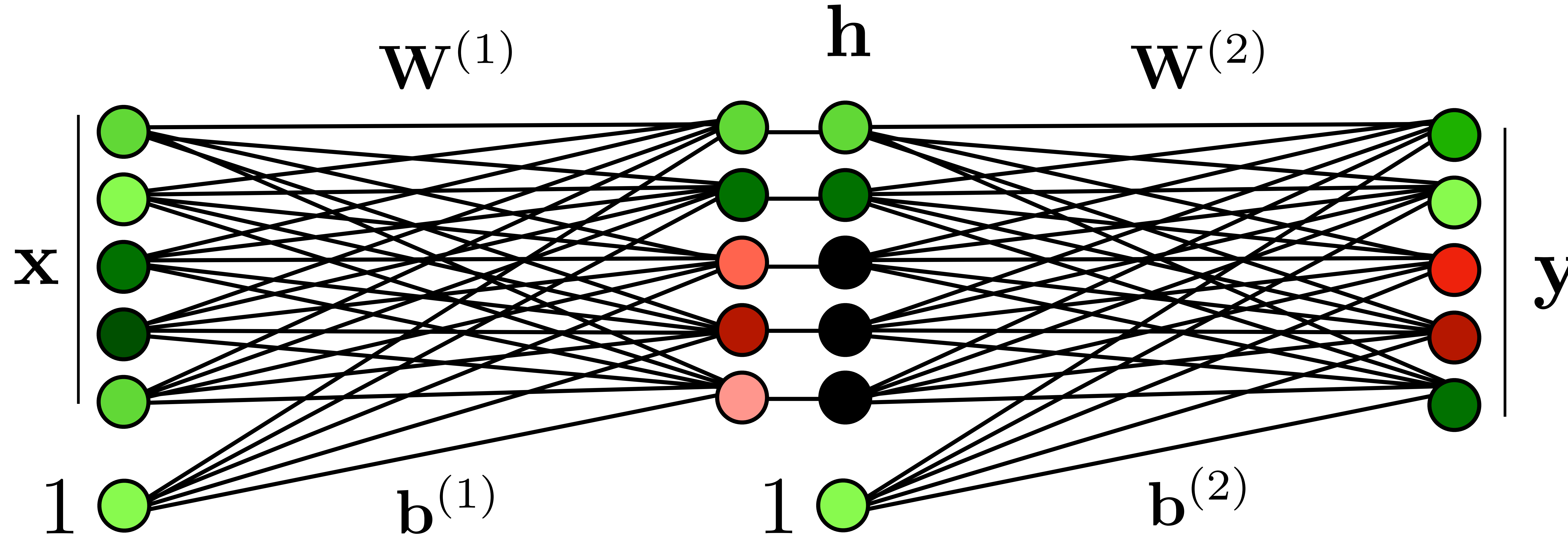
$$\theta = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}$$

Stacking layers

Input representation

Intermediate representation

Output representation



positive

negative

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

Representational power

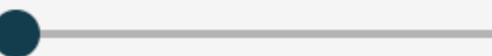
- 1 layer? Linear decision surface.
- 2+ layers? In theory, can represent any function. Assuming non-trivial non-linearity.
  - Bengio 2009,
    http://www.iro.umontreal.ca/~bengioy/papers/ftml.pdf
  - Bengio, Courville, Goodfellow book
    http://www.deeplearningbook.org/contents/mlp.html
  - Simple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

DATA

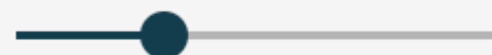
Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 0



Batch size: 10



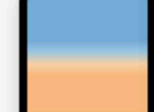
REGENERATE

FEATURES

Which properties do you want to feed in?

X_1

X_2



X_1^2

X_2^2

X_1X_2

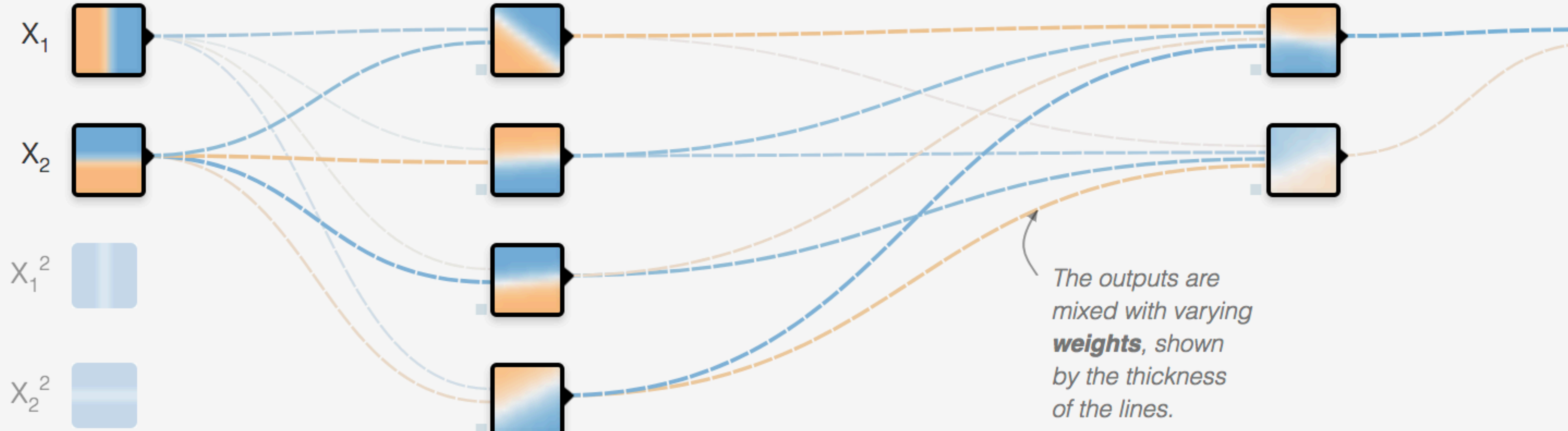
$\sin(X_1)$

$\sin(X_2)$

+ − 2 HIDDEN LAYERS

+ − 4 neurons

+ − 2 neurons

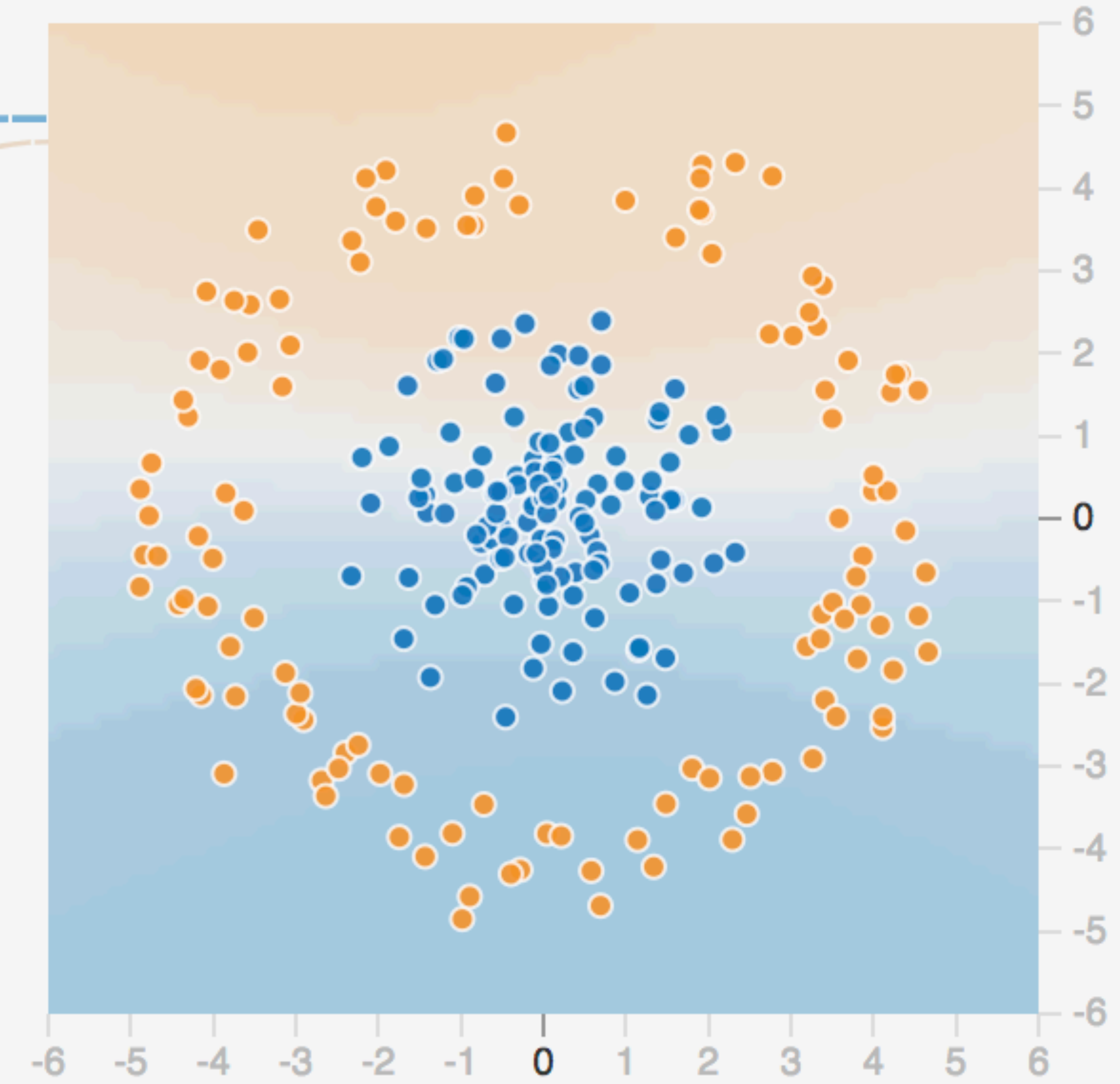


*This is the output from one **neuron**. Hover to see it larger.*

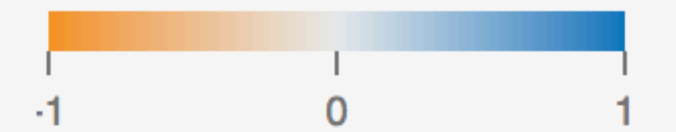
OUTPUT

Test loss 0.540

Training loss 0.555



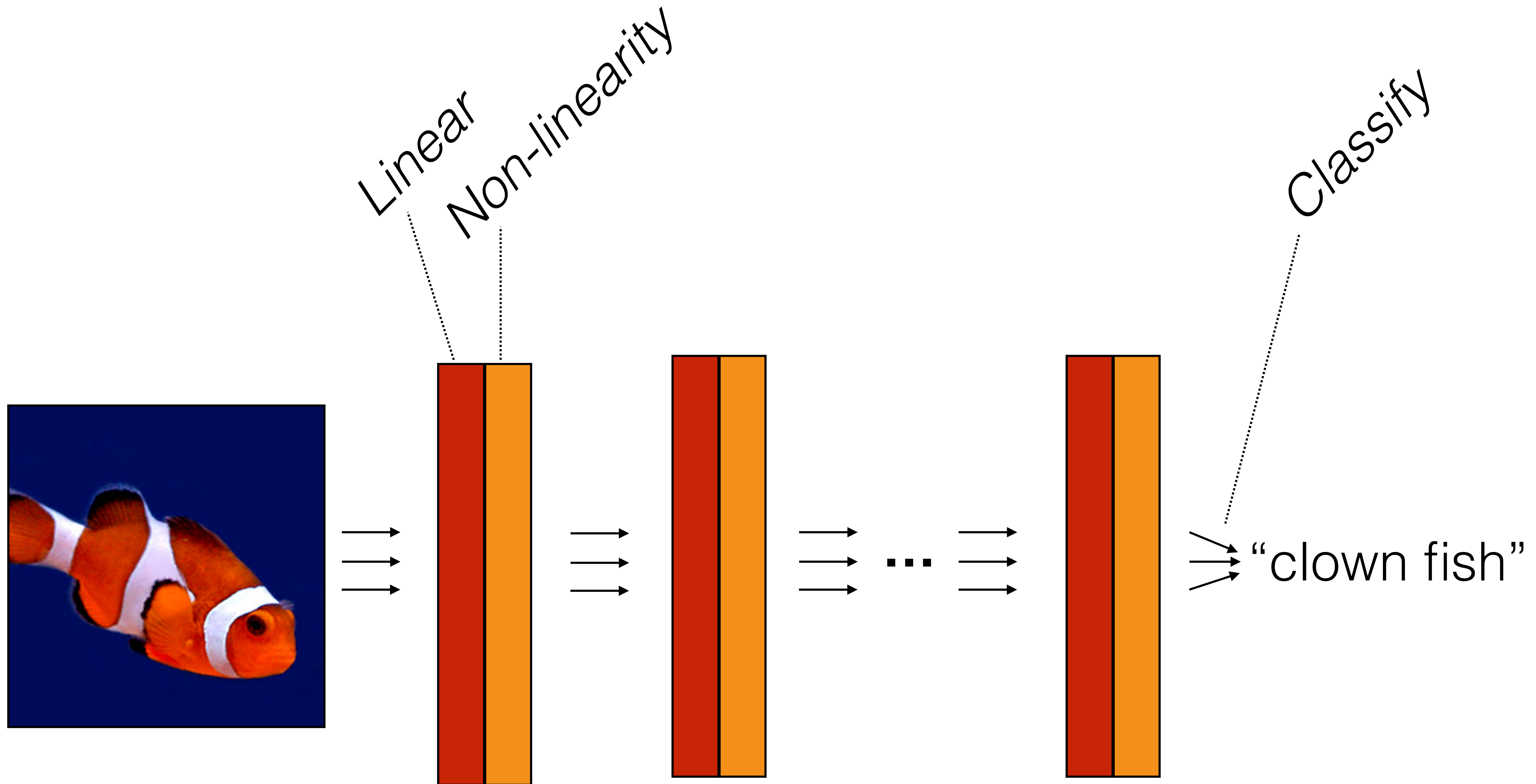
Colors shows data, neuron and weight values.



☐ Show test data ☐ Discretize output

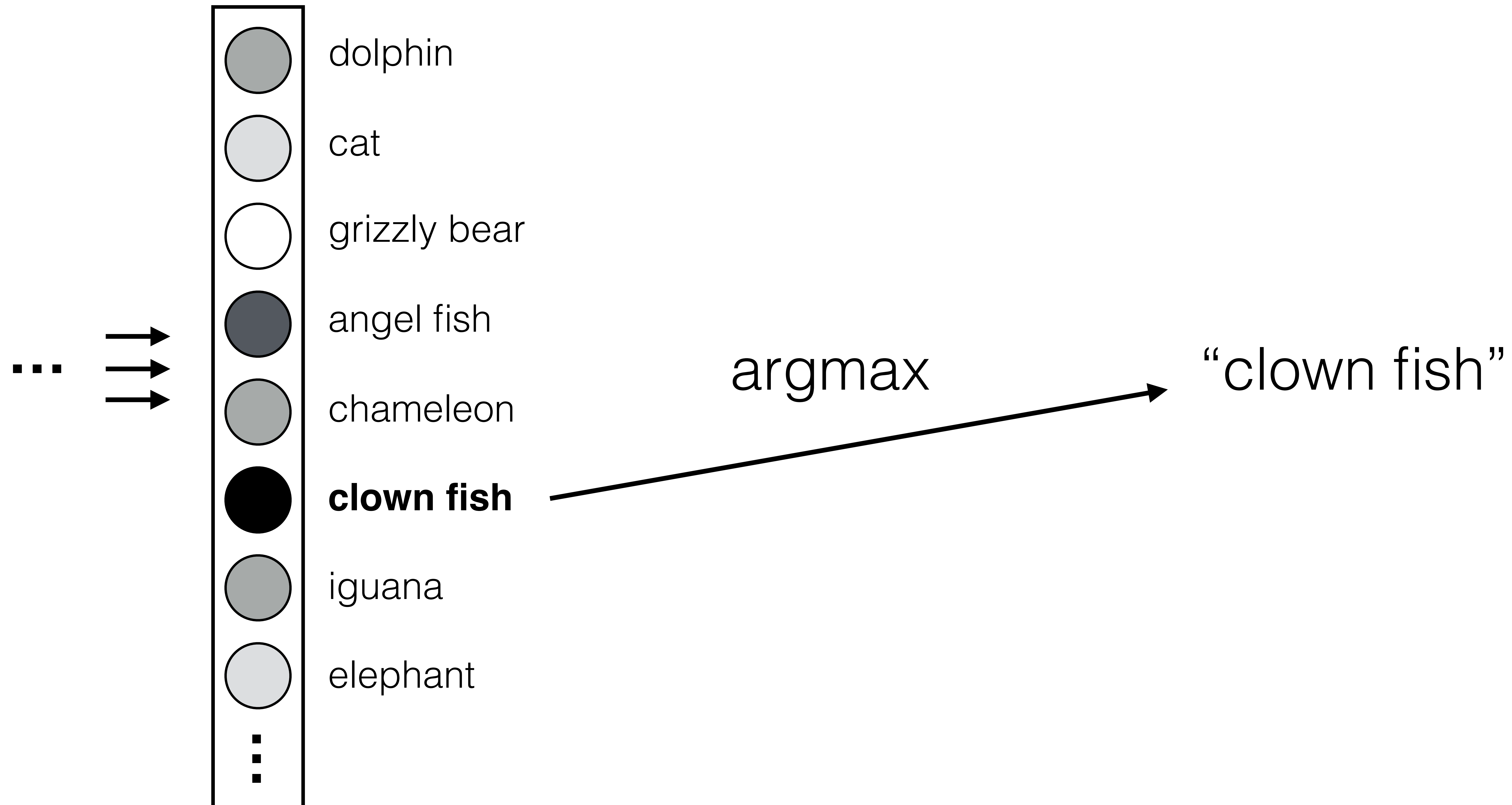
[http://playground.tensorflow.org]

Deep nets

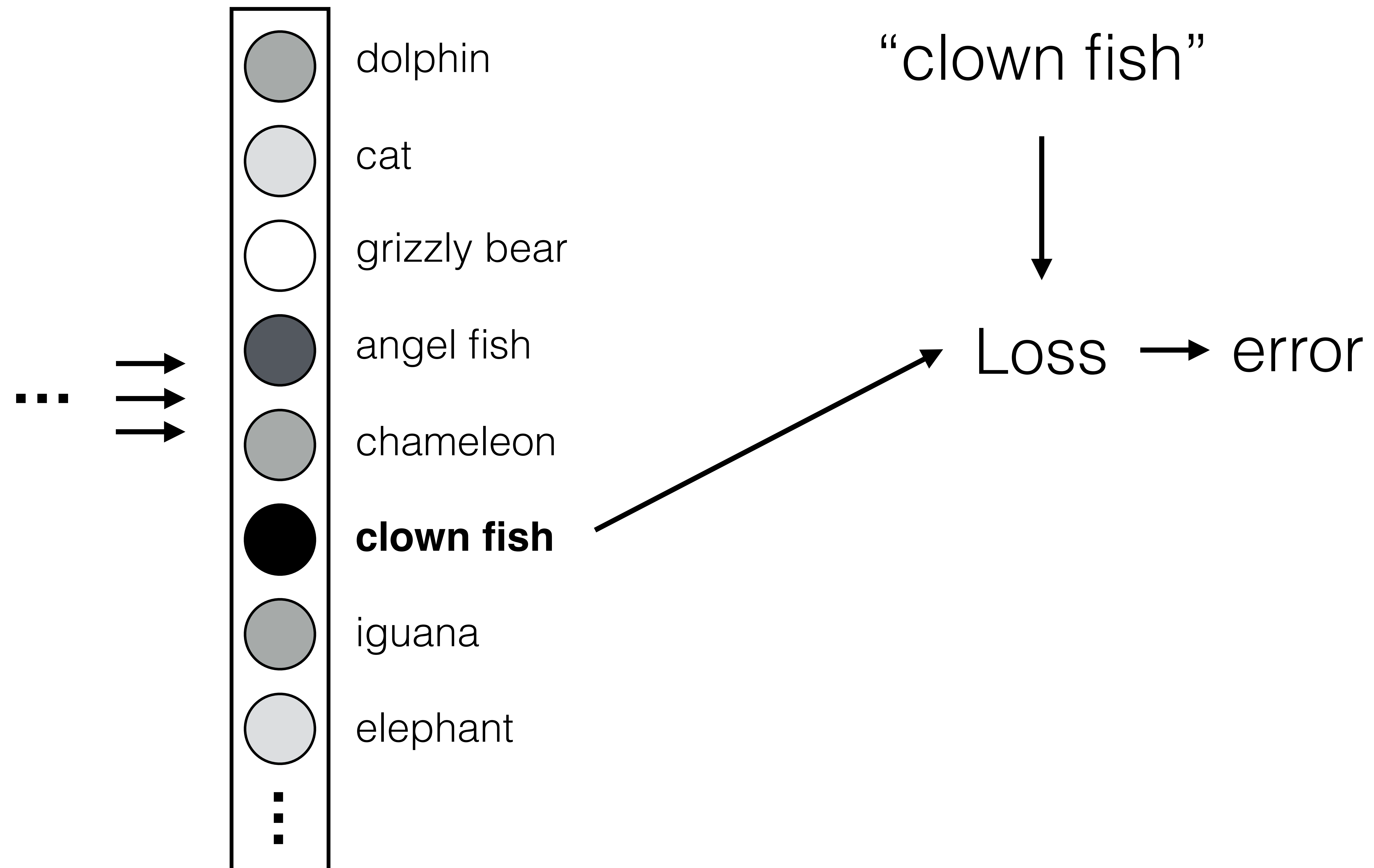


$$f(\mathbf{x}) = f_L(\ldots f_2(f_1(\mathbf{x})))$$

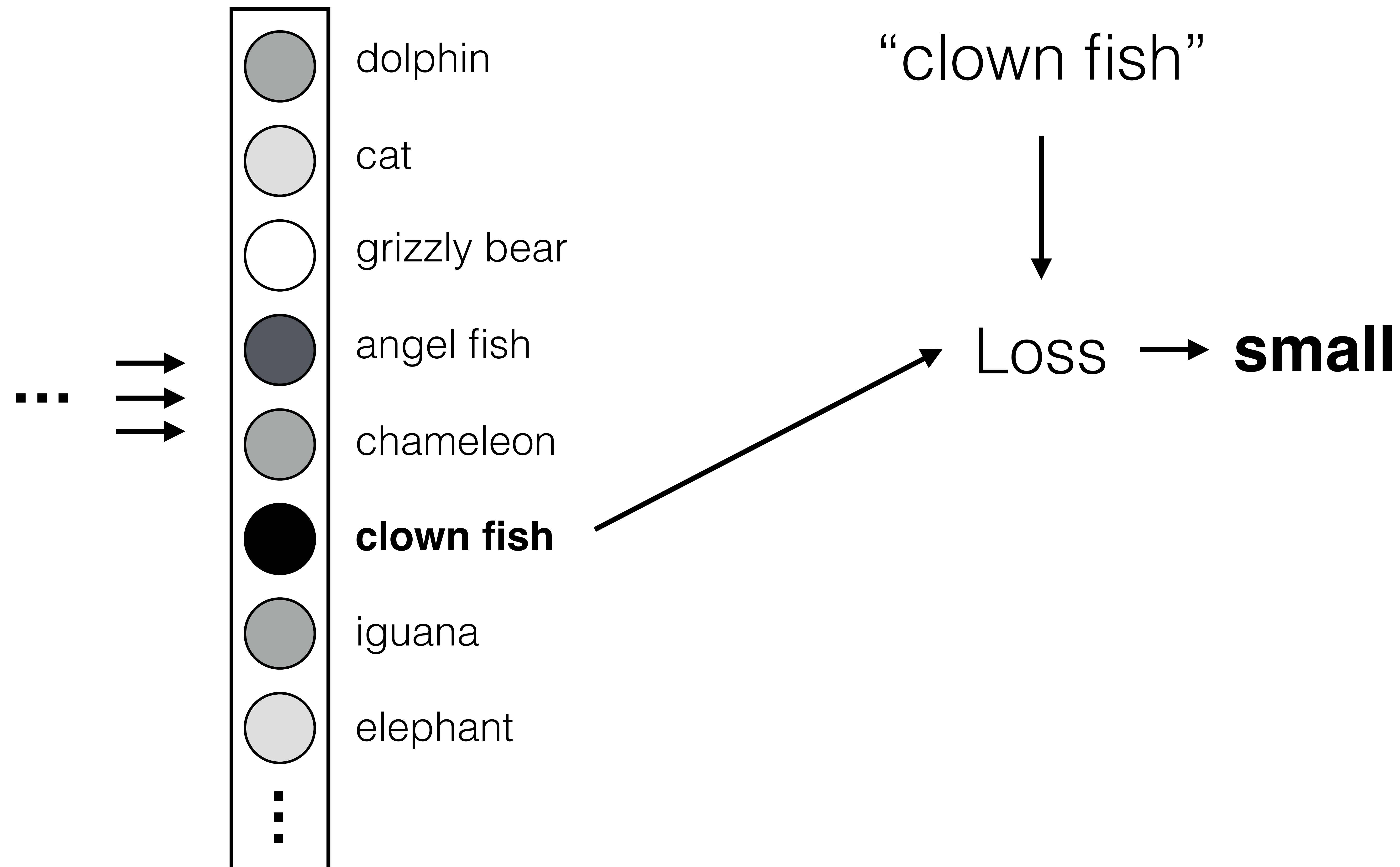
Classifier layer



Loss function



Loss function

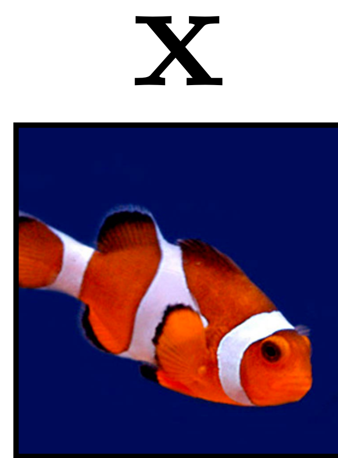


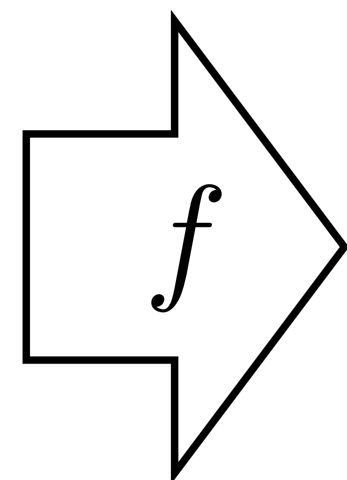
Loss function

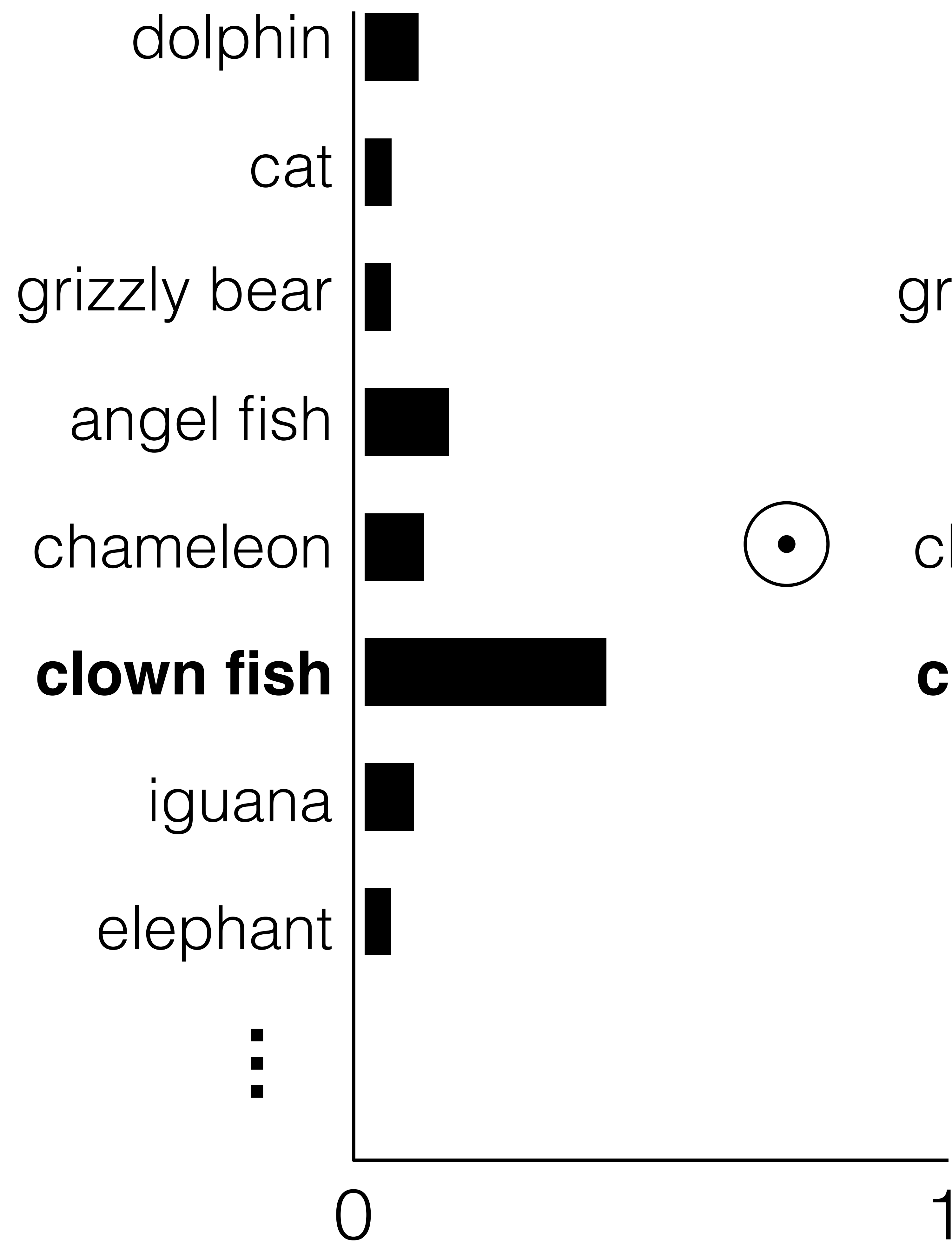
Prediction $\hat{\mathbf{y}}$

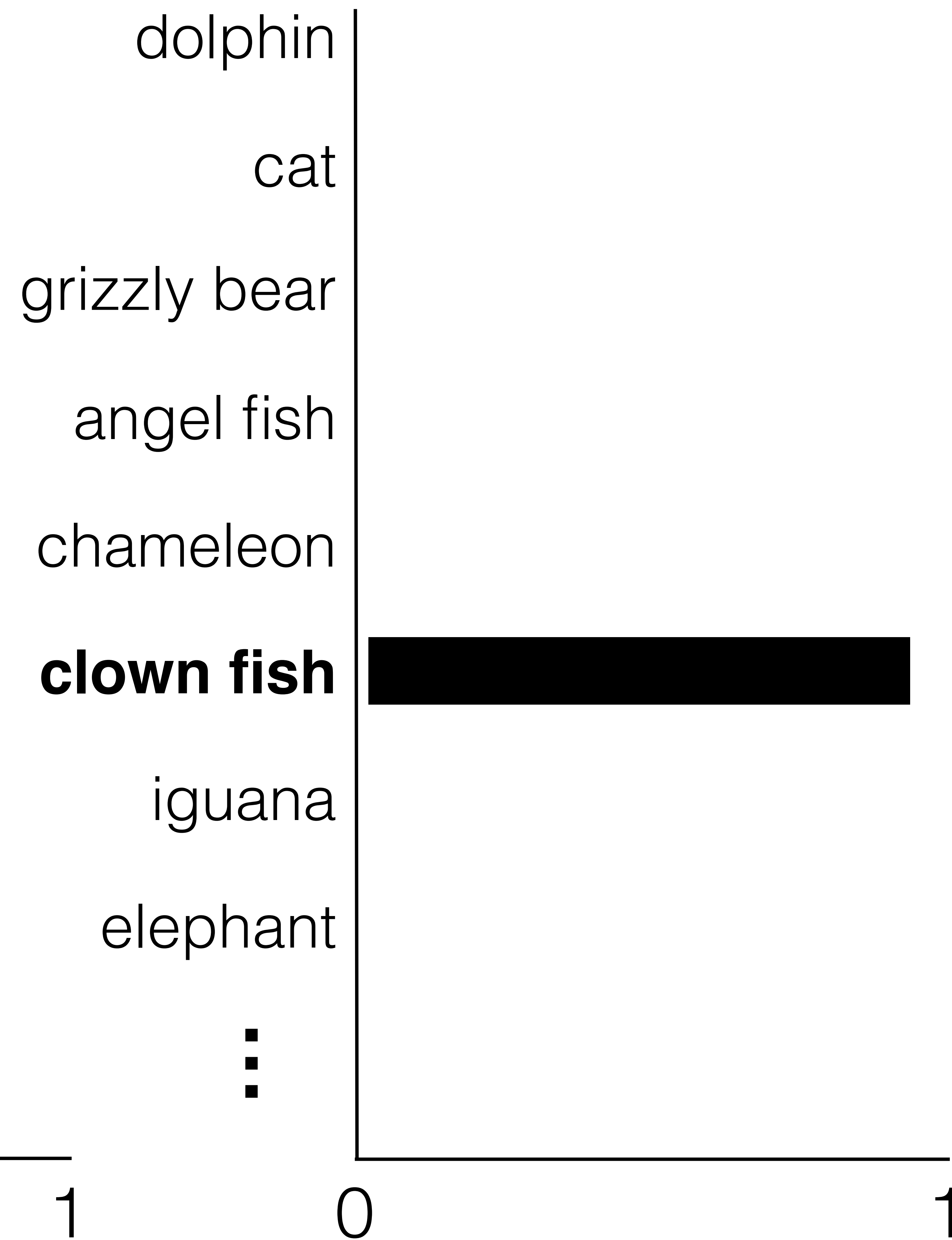
Ground truth label $\mathbf{y}$

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

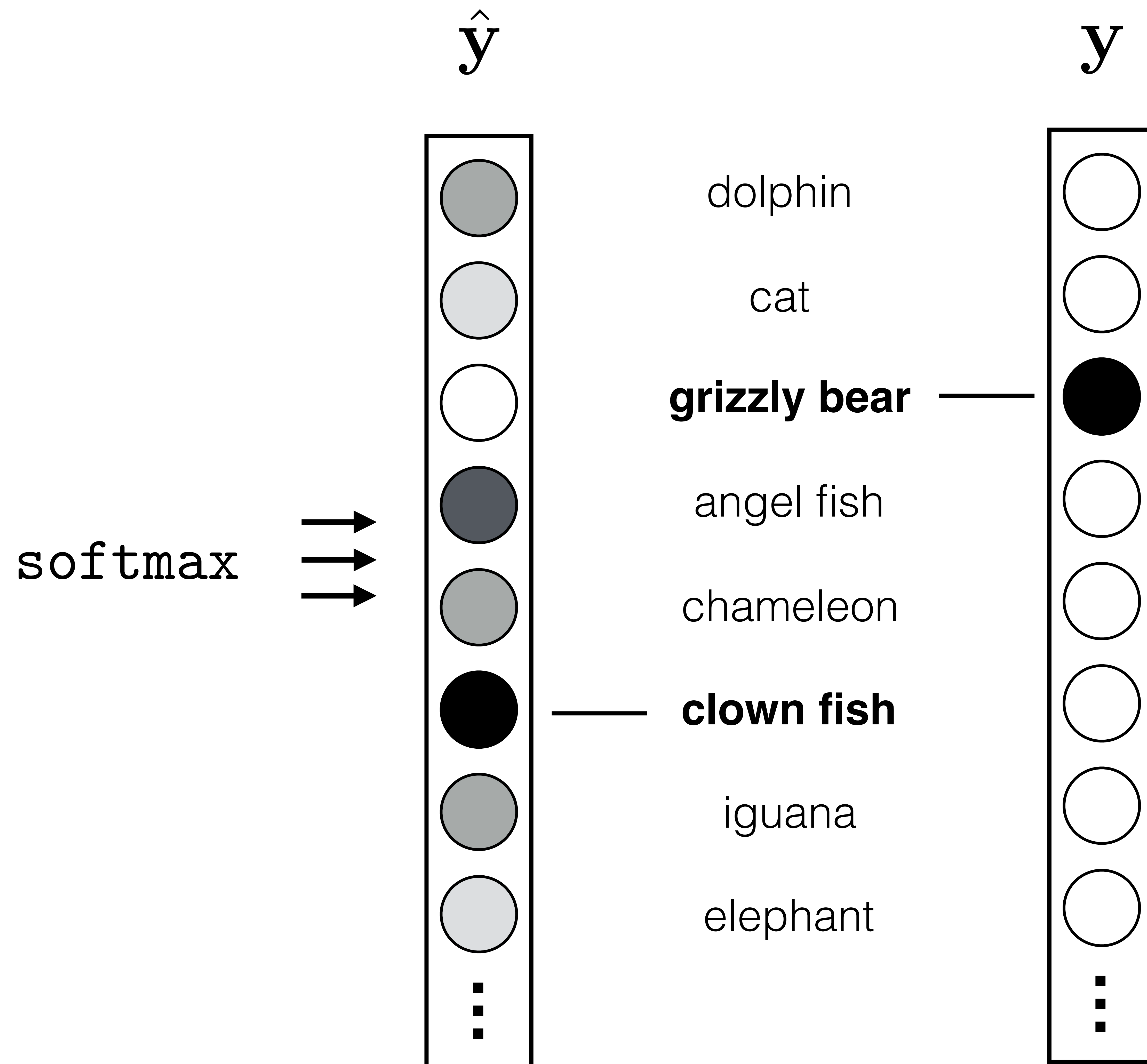








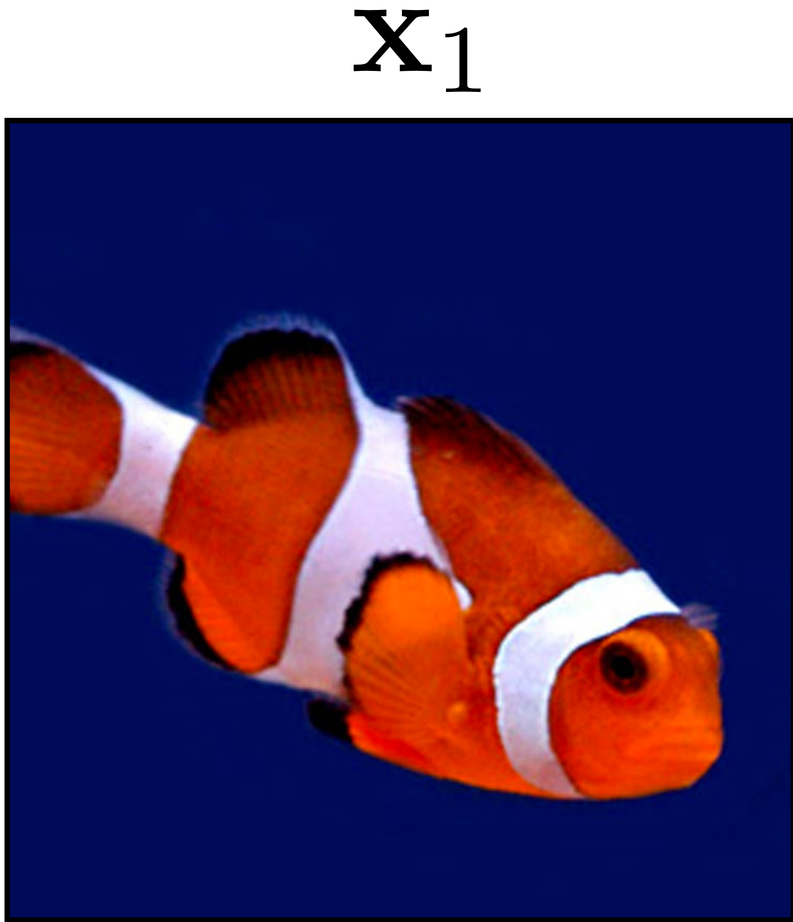
Network output Ground truth label

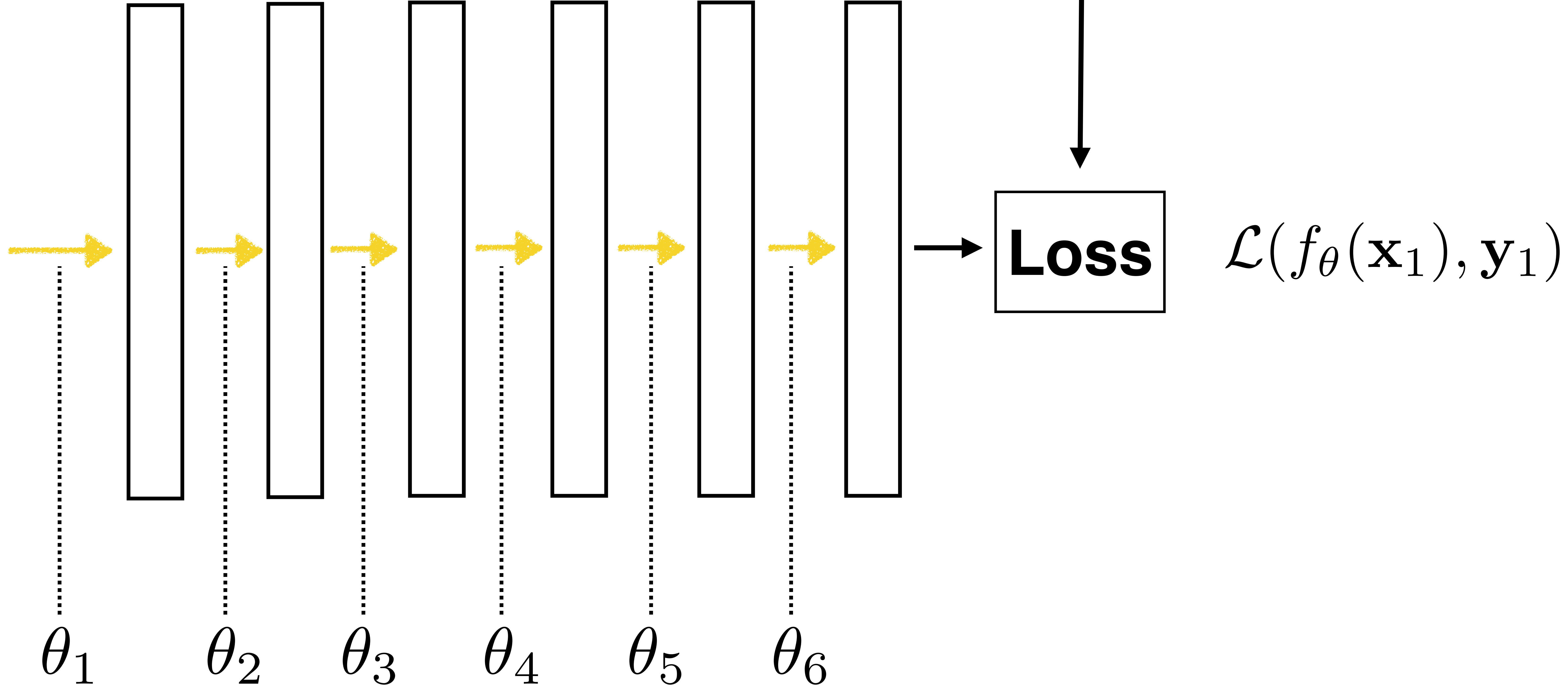


Probability of the observed data under the model

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

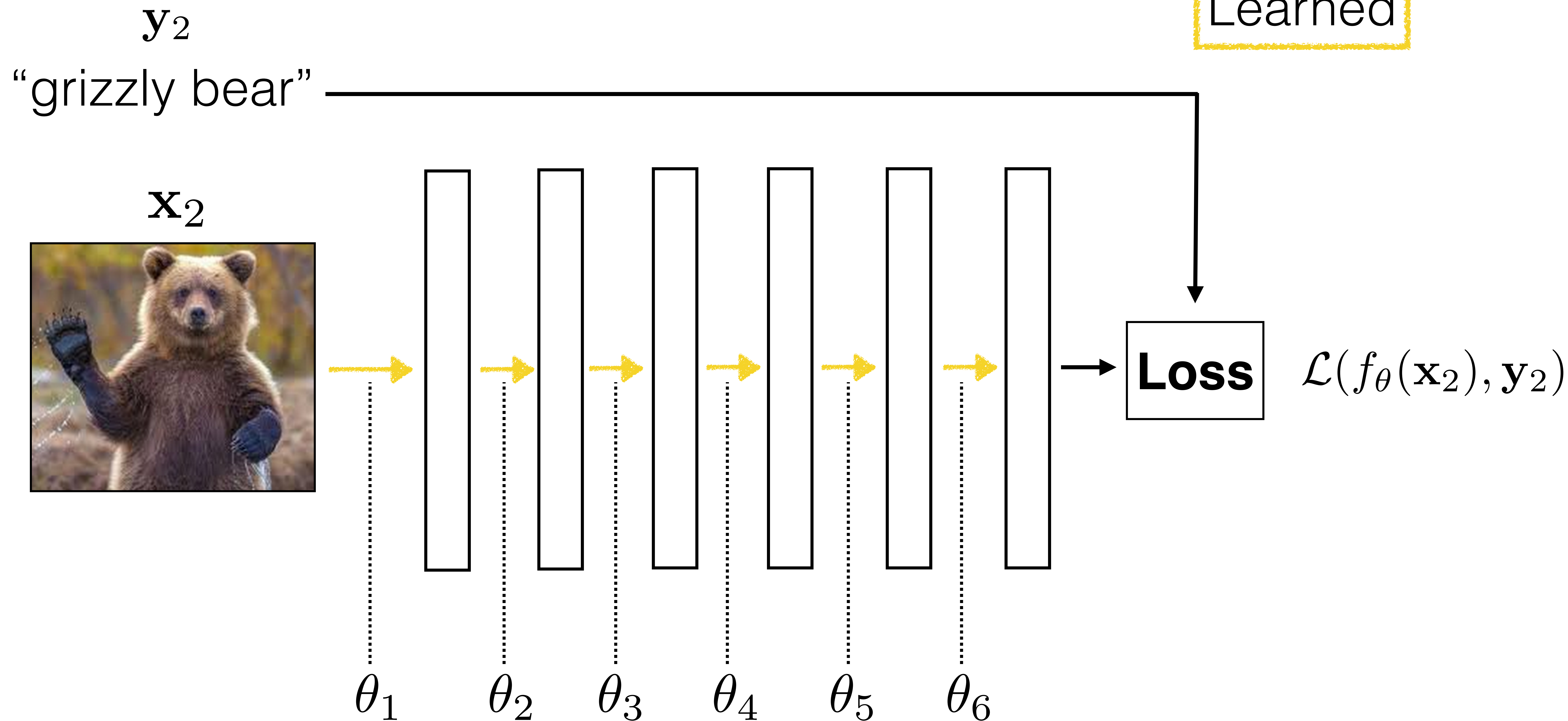
Deep learning



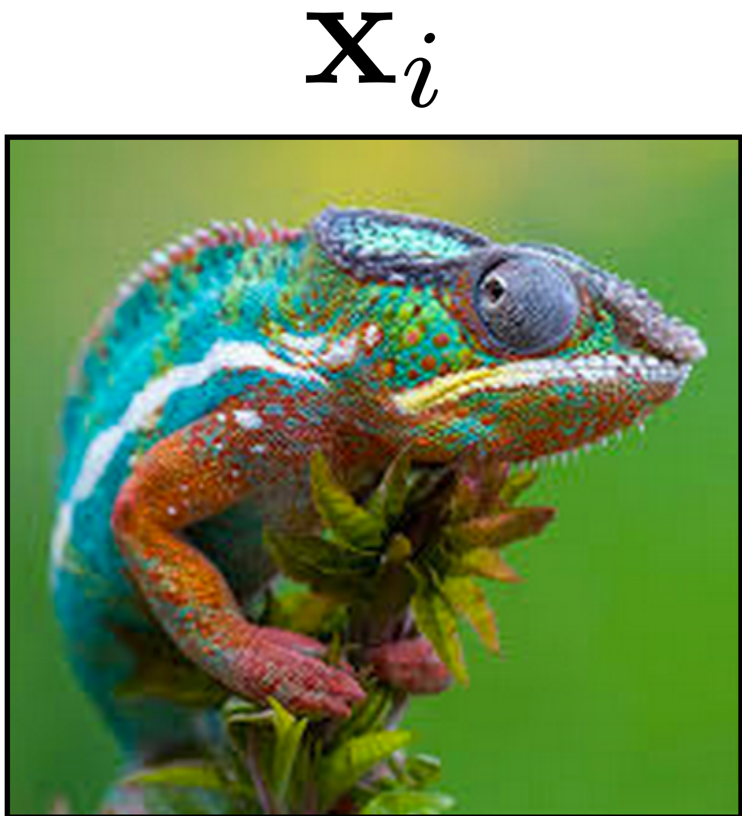


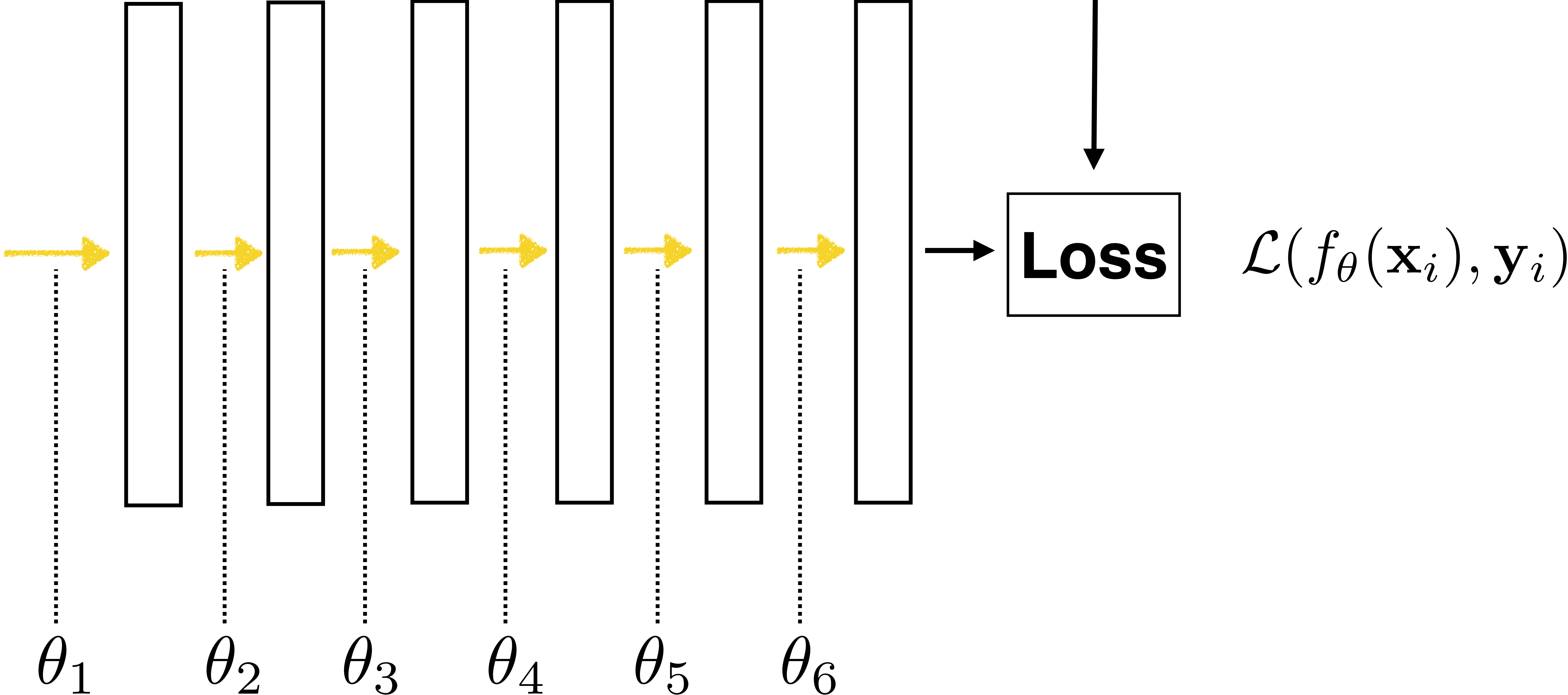
$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)$$

Deep learning

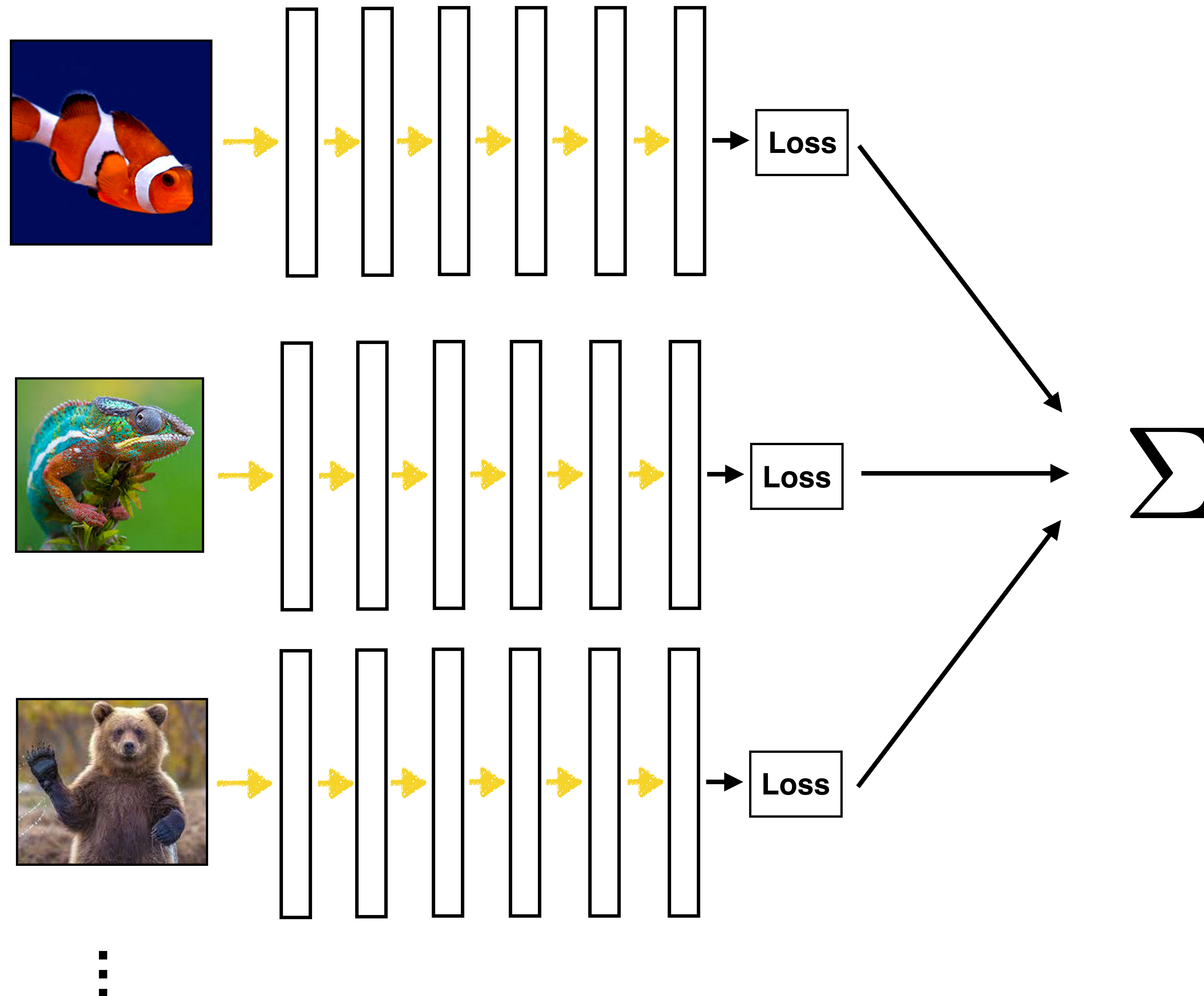


Deep learning



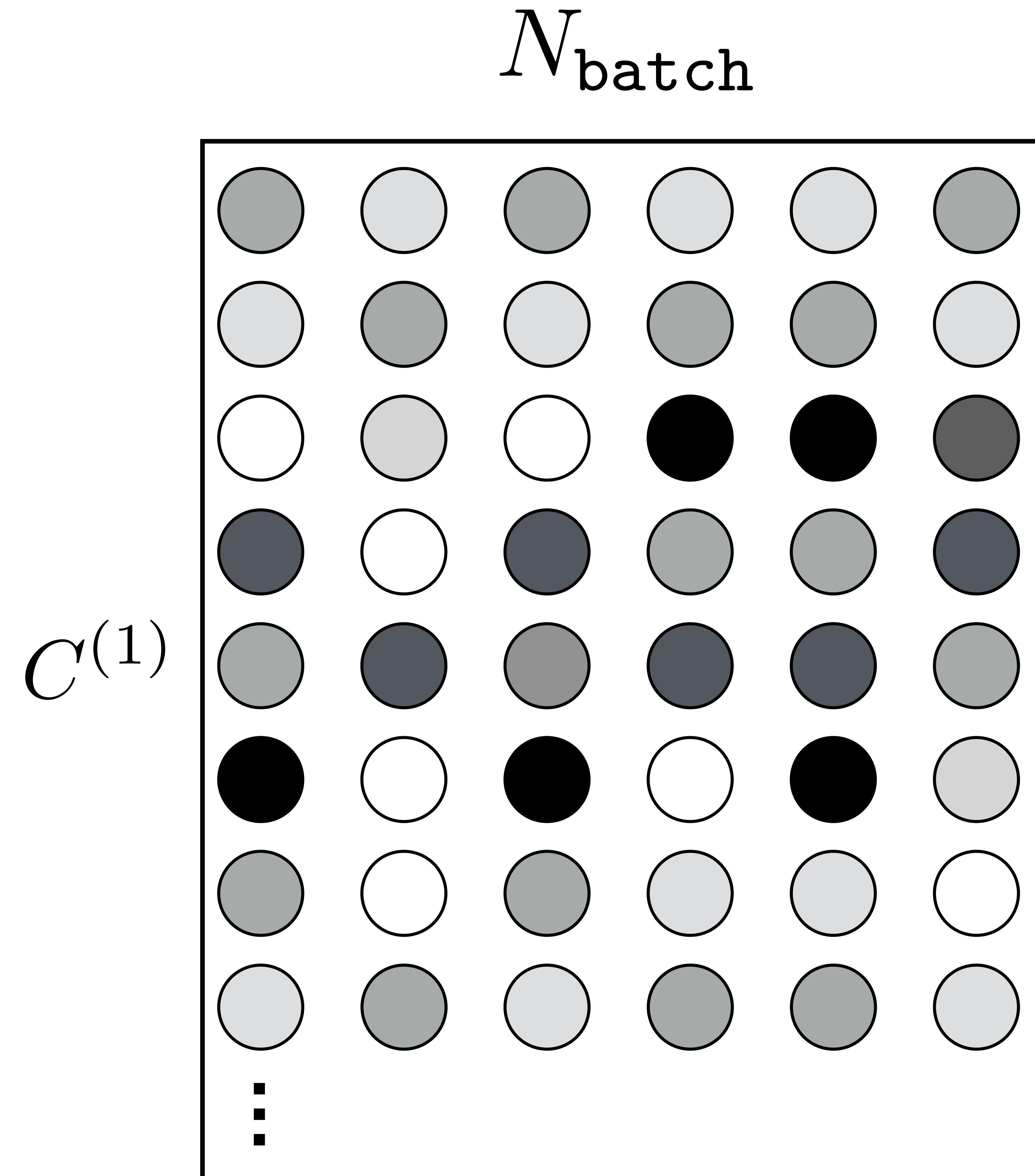


Batch (parallel) processing



Tensors

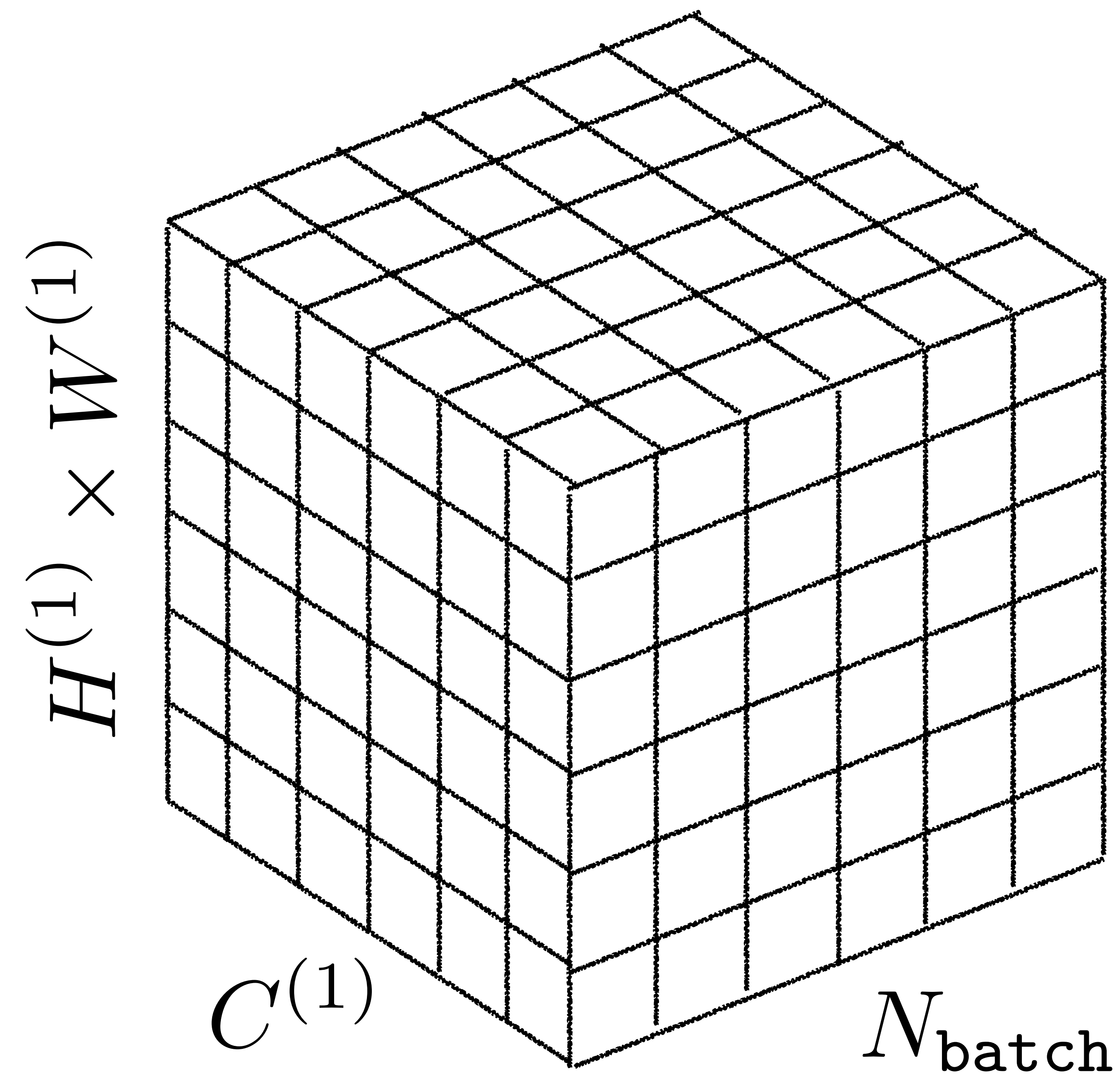
$$\mathbf{h}^{(1)} \in \mathbb{R}^{N_{\texttt{batch}} \times C^{(1)}}$$

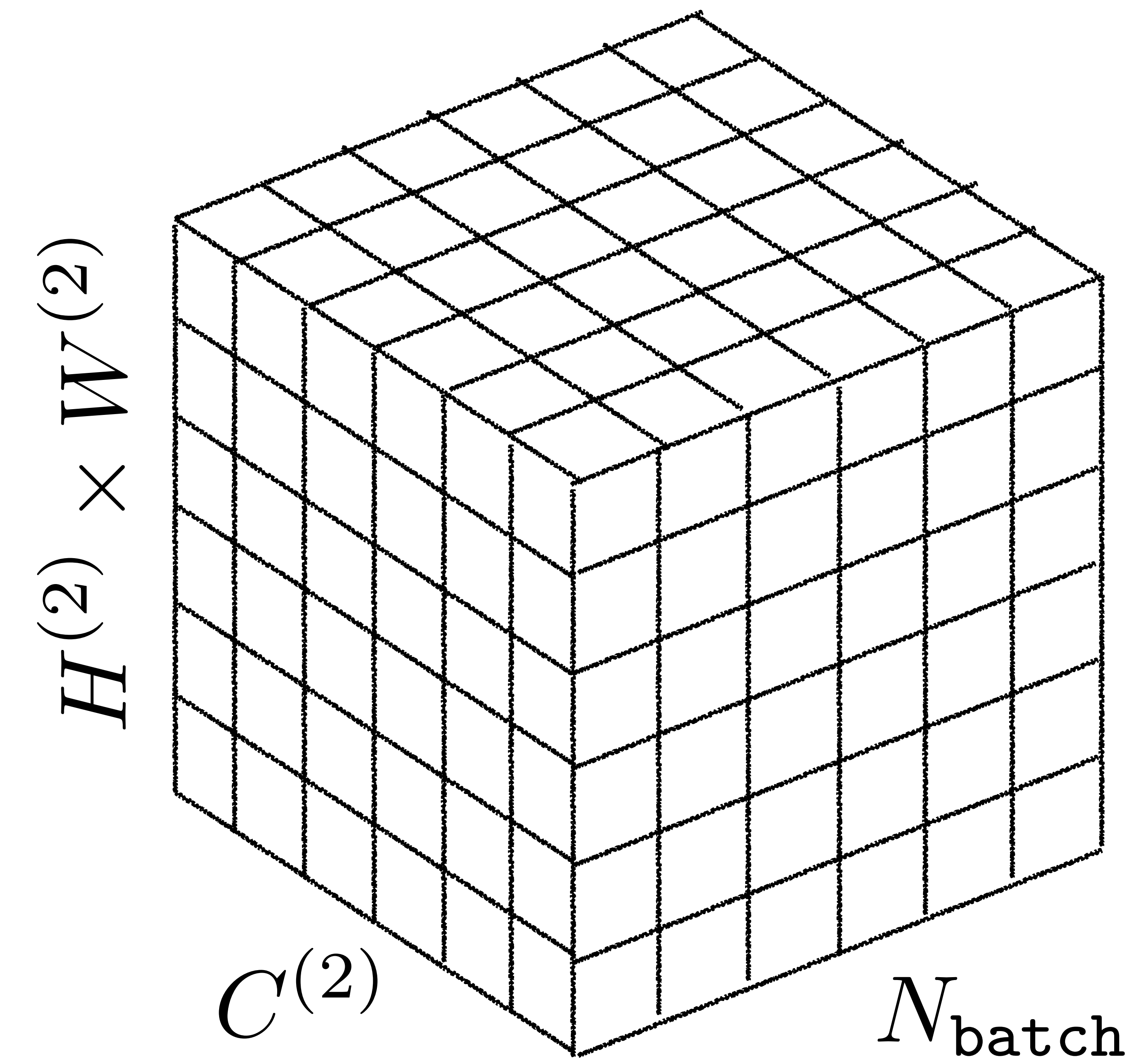


"Tensor flow"

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

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





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.

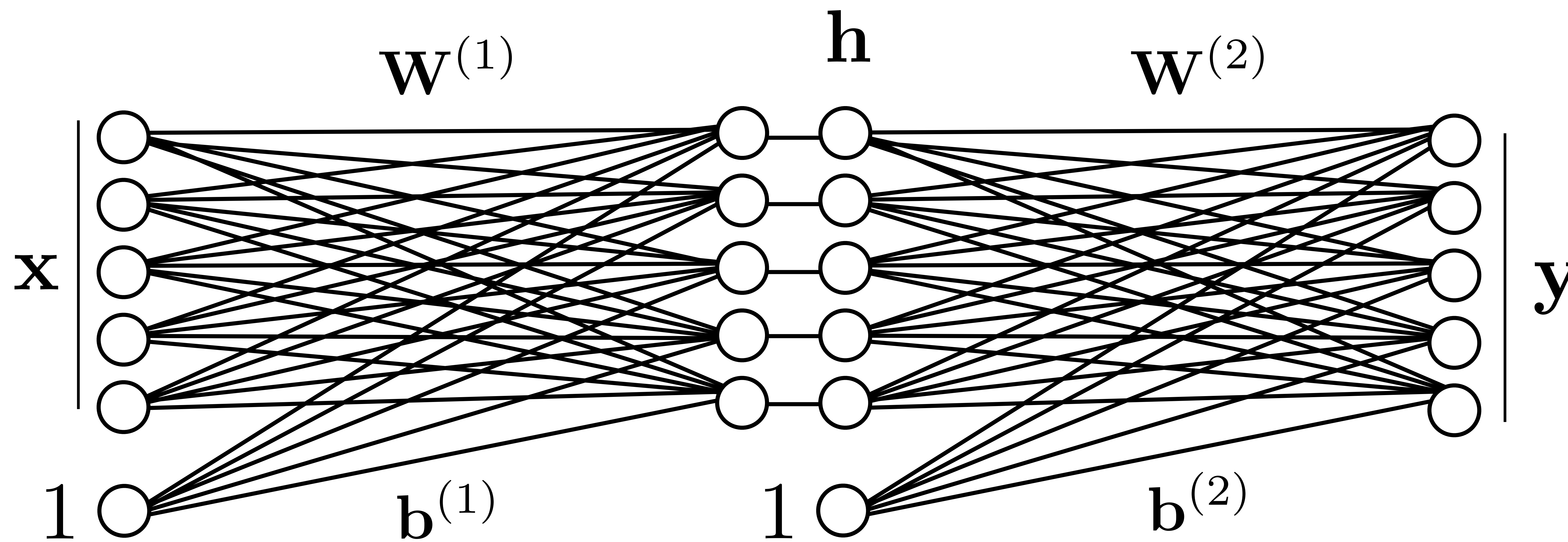
Regularizing the weights in a neural net

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

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

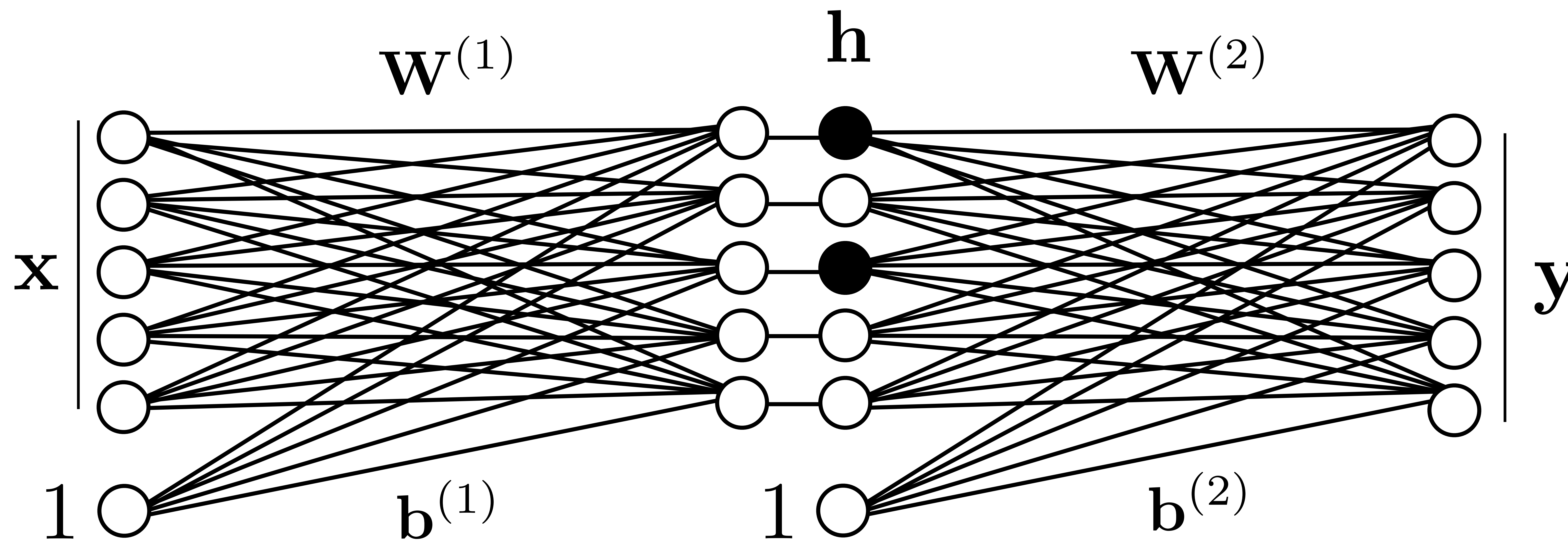
“We prefer to keep weights small.”

Dropout



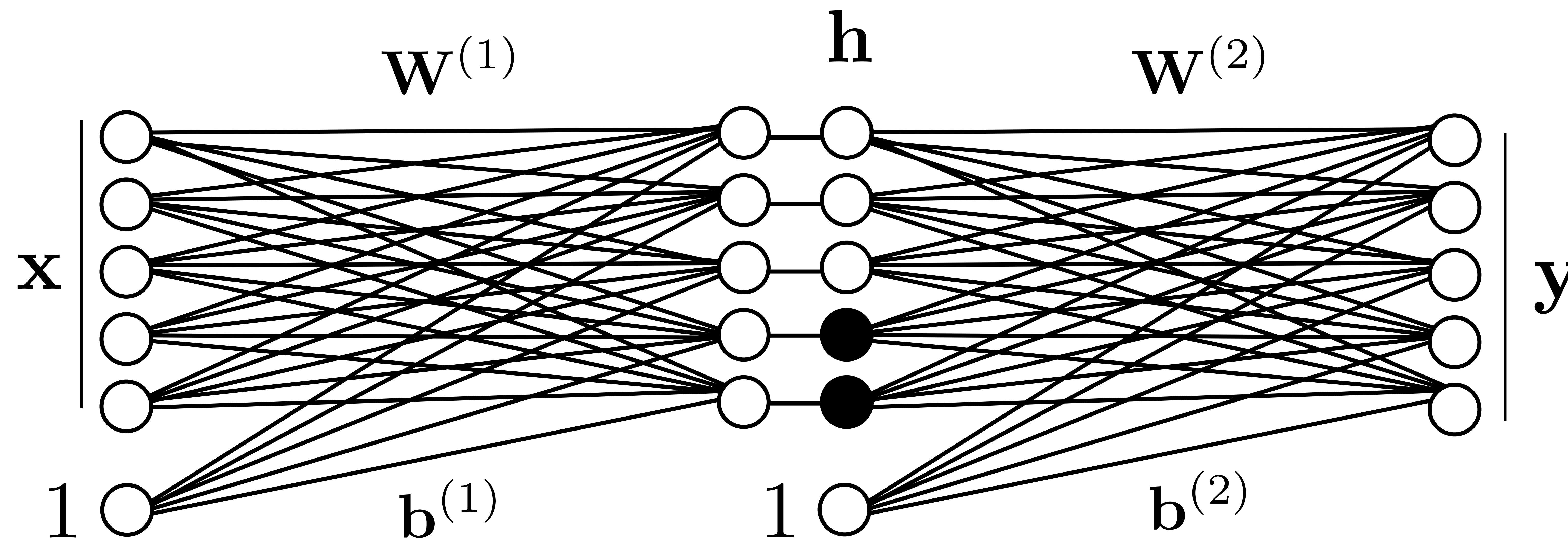
$$\theta = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}$$

Dropout



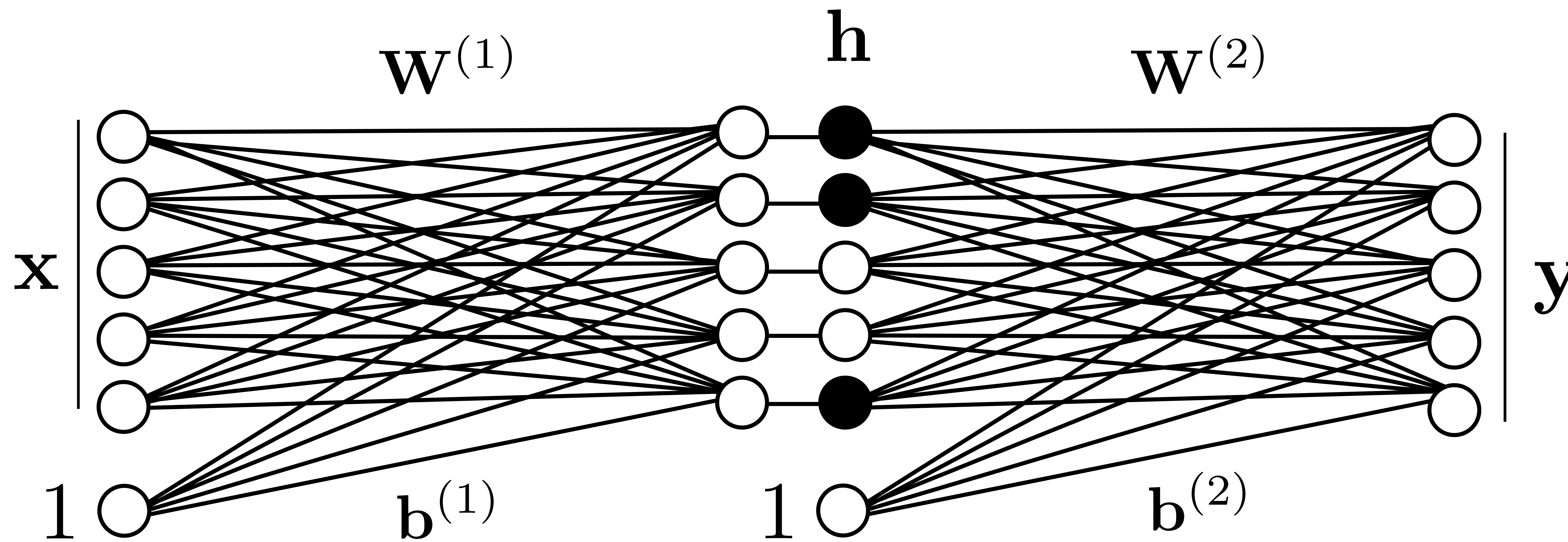
$$\theta = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}$$

Dropout



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

Dropout



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

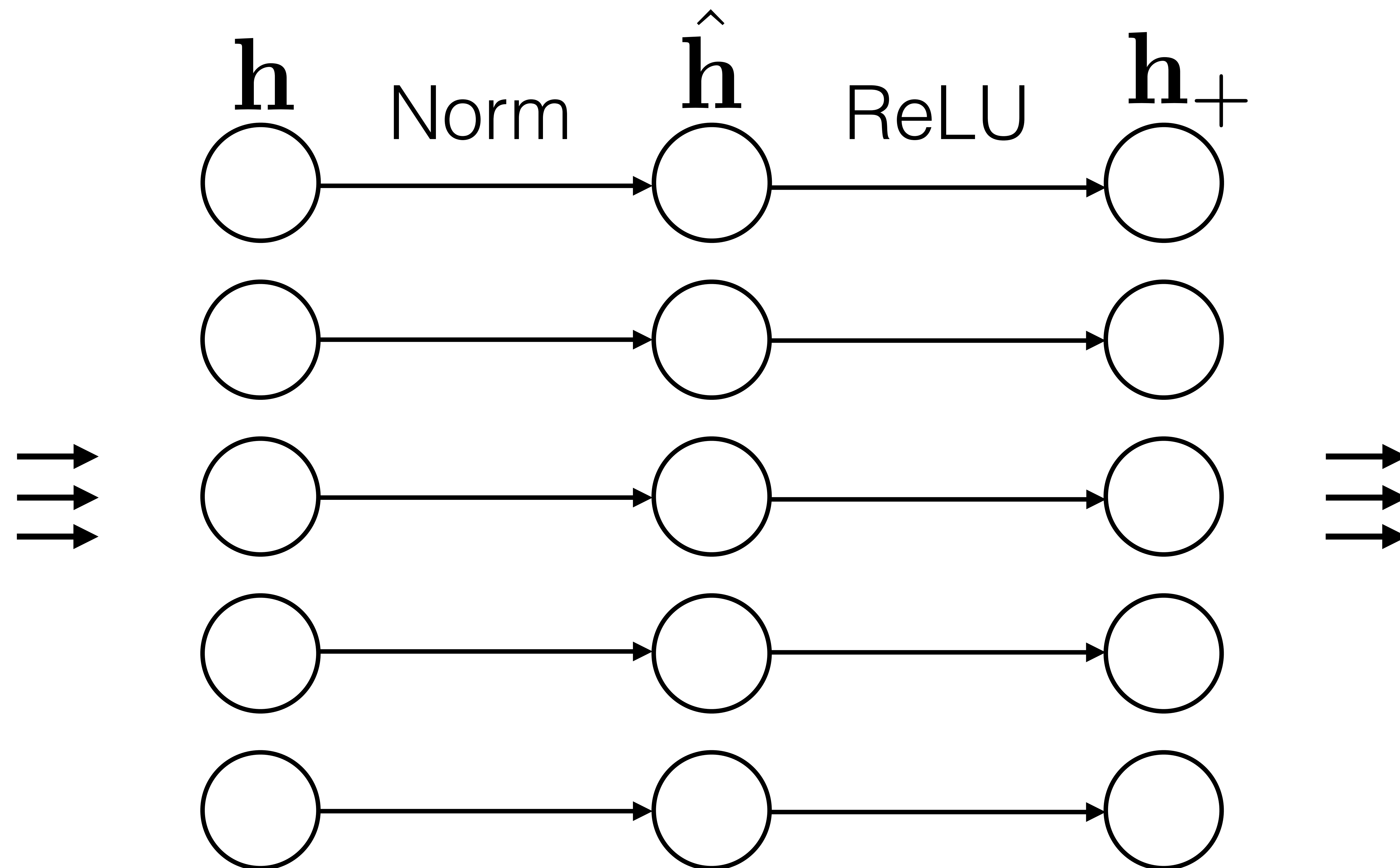
Dropout

Randomly zero out hidden units.

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

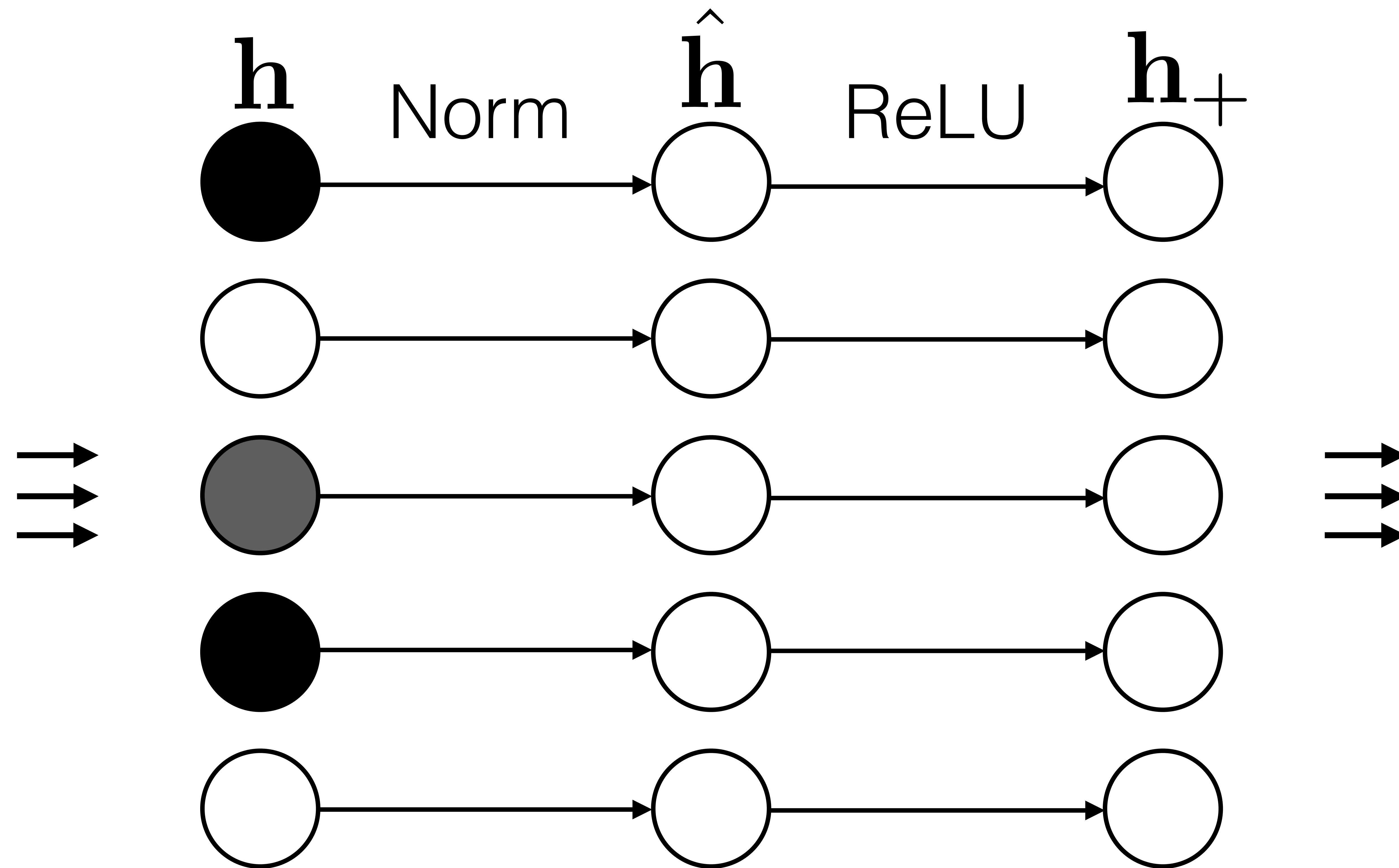
Can be understood as averaging over an exponential **ensemble** of subnetworks. This averaging smooths the function, thereby reducing the effective capacity of the network.

Normalization layers



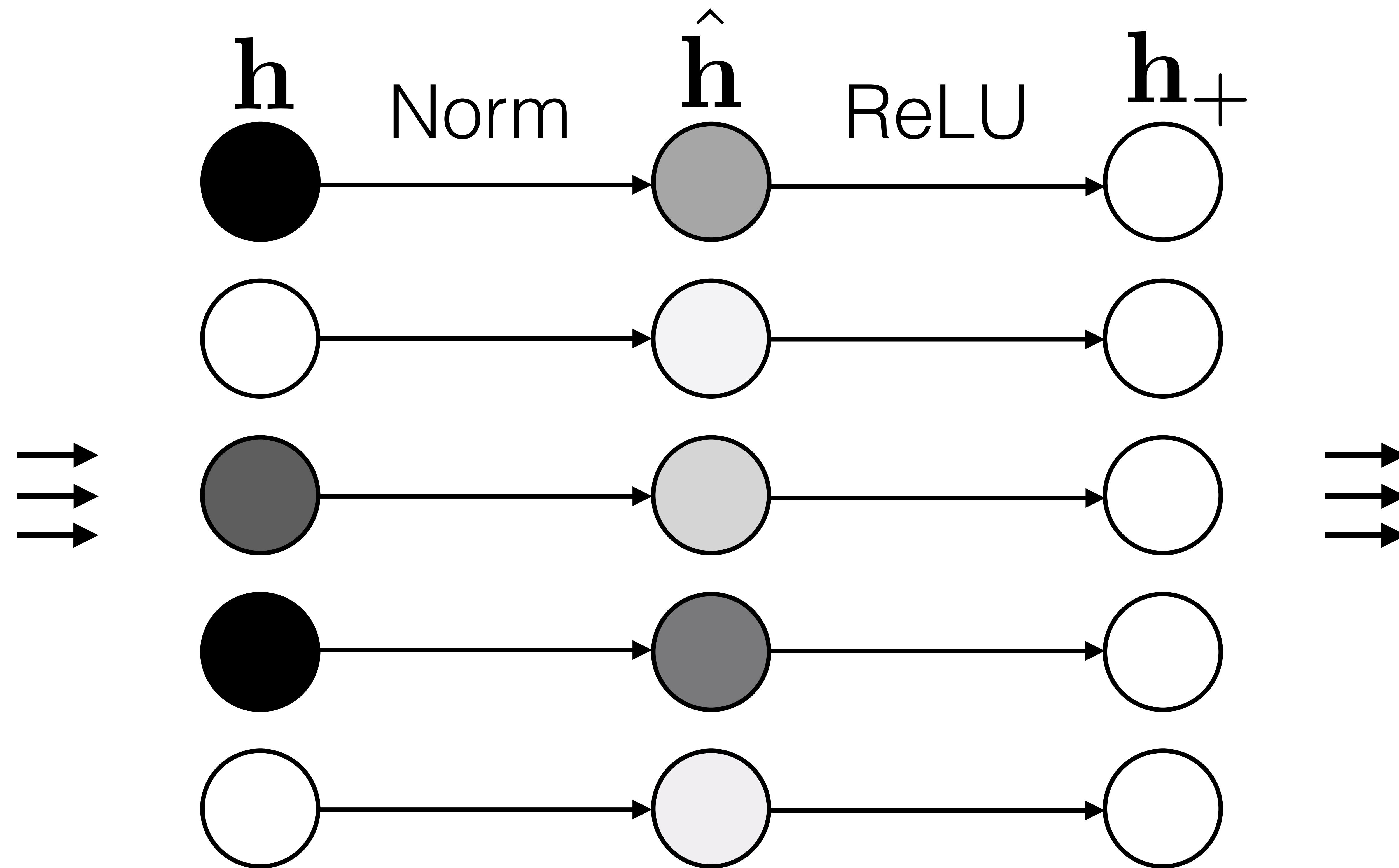
$$\hat{h}_k = \frac{h_k - \mathbb{E}[h_k]}{\sqrt{\texttt{Var}[h_k]}}$$

Normalization layers



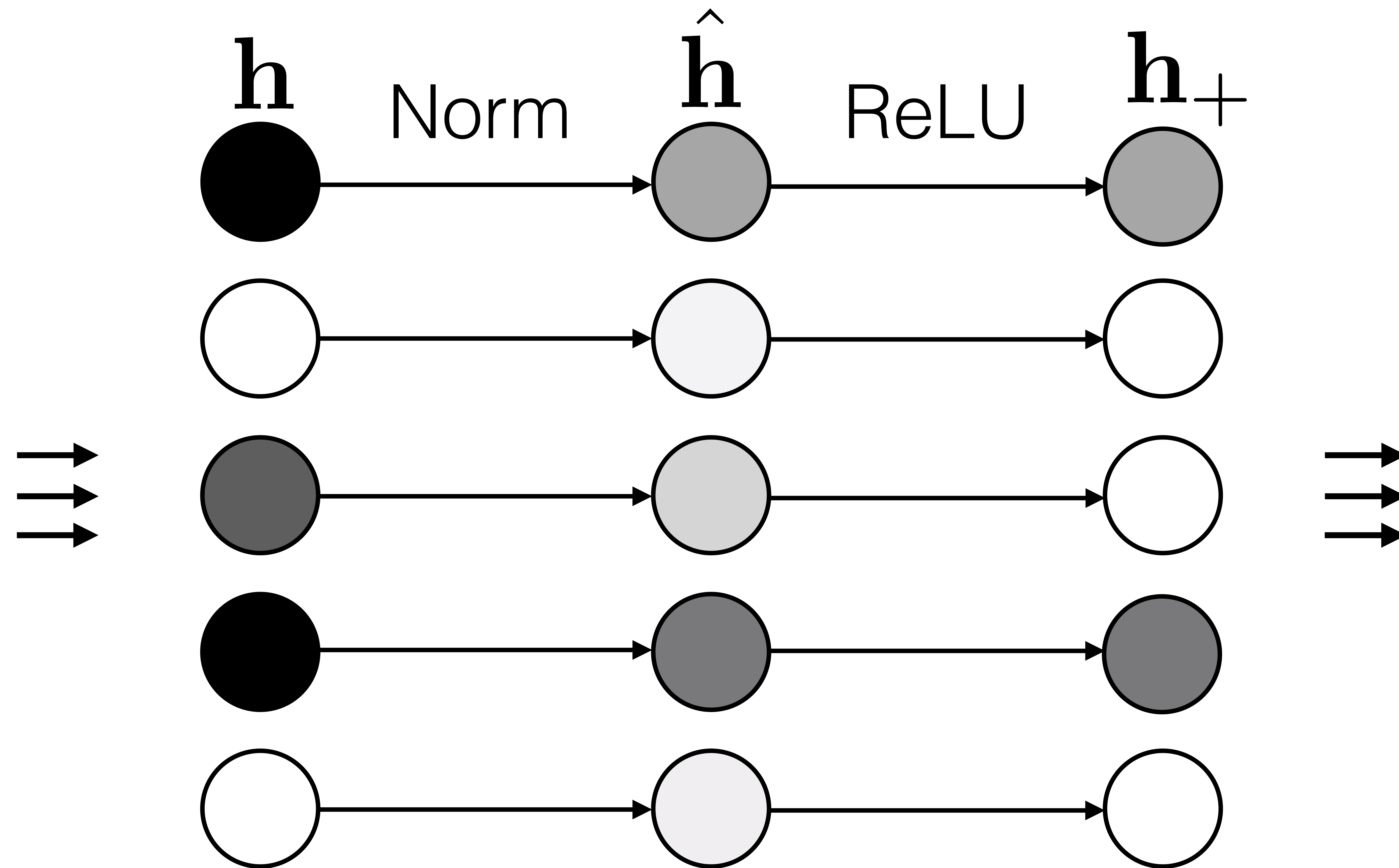
$$\hat{h}_k = \frac{h_k - \mathbb{E}[h_k]}{\sqrt{\mathtt{Var}[h_k]}}$$

Normalization layers



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

Normalization layers



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

Normalization layers

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

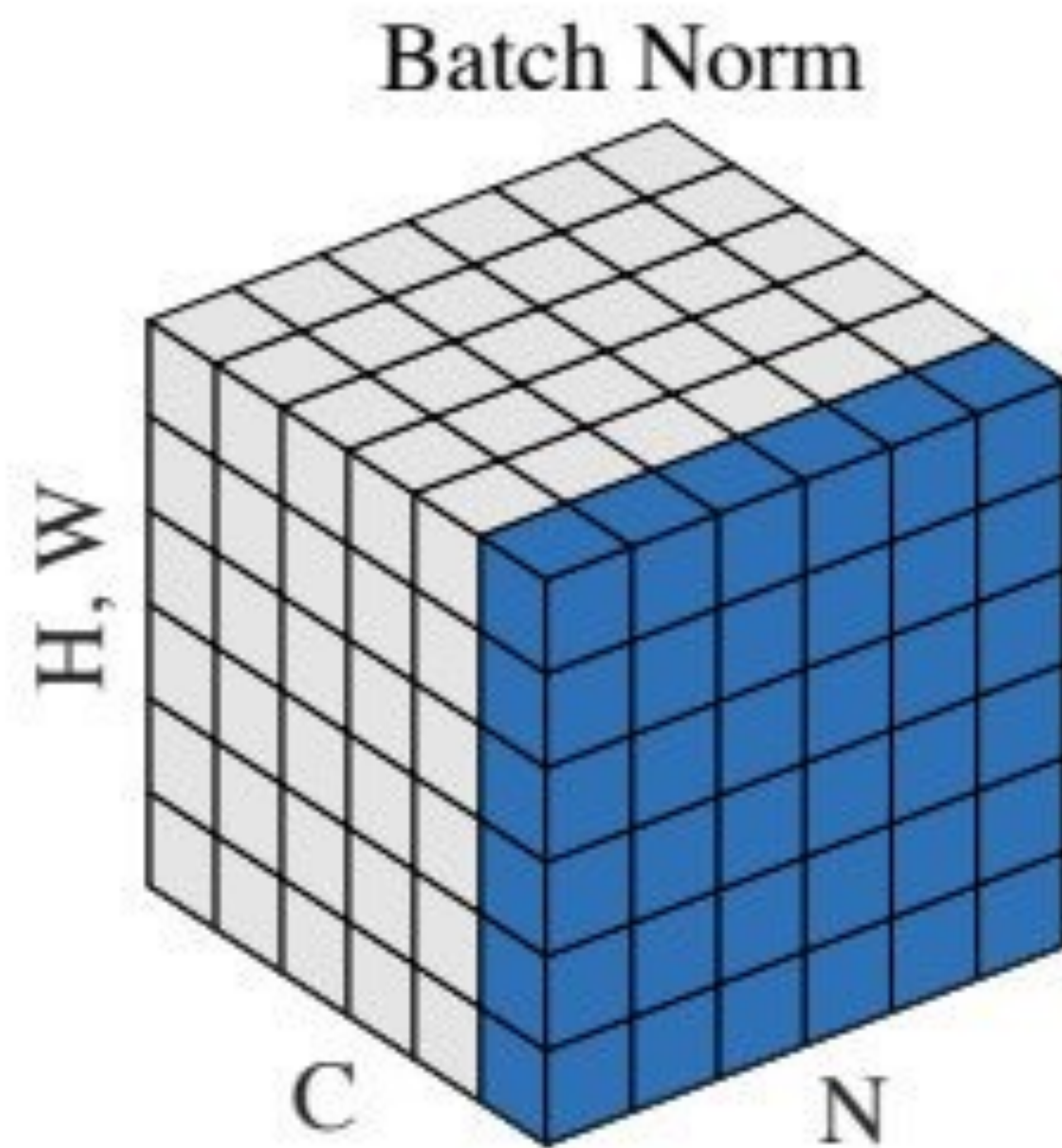
Standardize unit activations by subtracting mean and dividing by variance.

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.

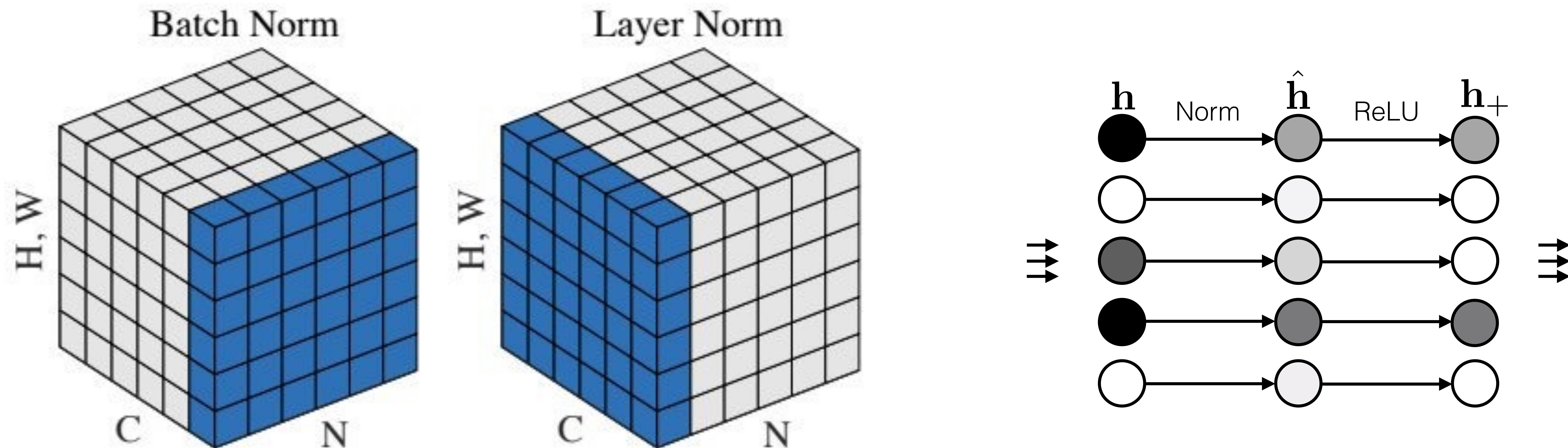
Normalization layers



Normalize w.r.t. a single hidden unit's pattern of activation over training examples (a batch of examples).

[Figure from Wu & He, arXiv 2018]

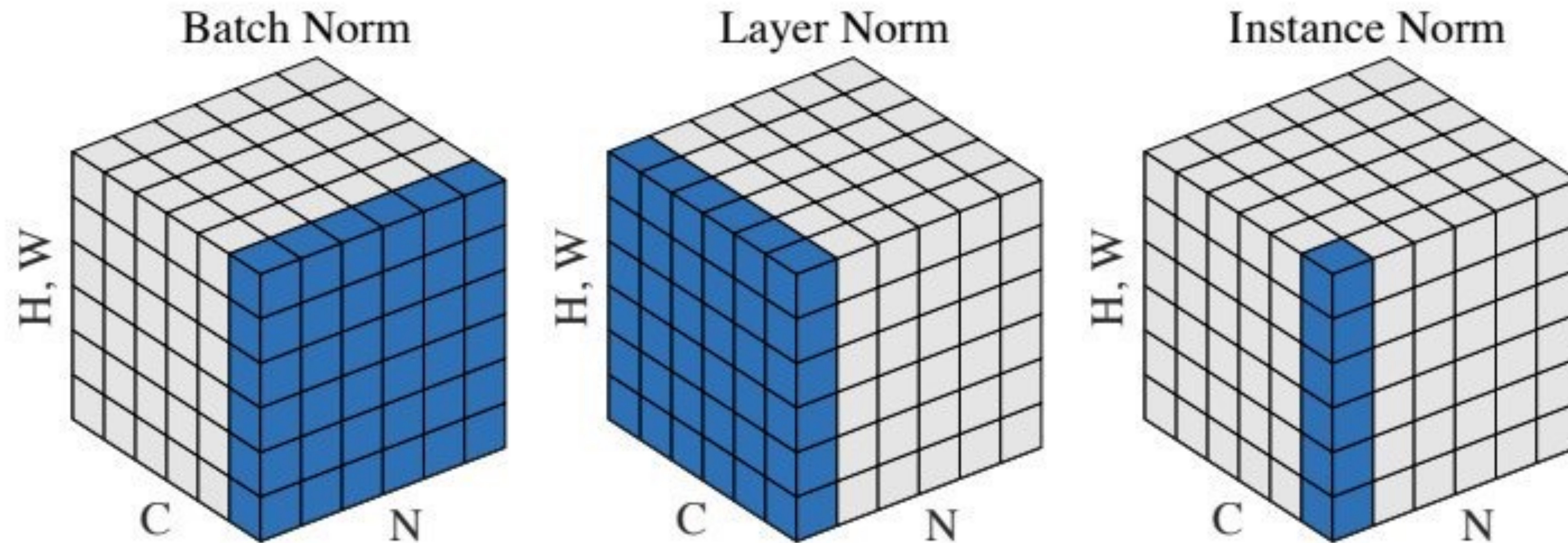
Normalization layers



Normalize w.r.t. the mean and variance of the activations of all the hidden units (neurons) on this layer (c).

[Figure from Wu & He, arXiv 2018]

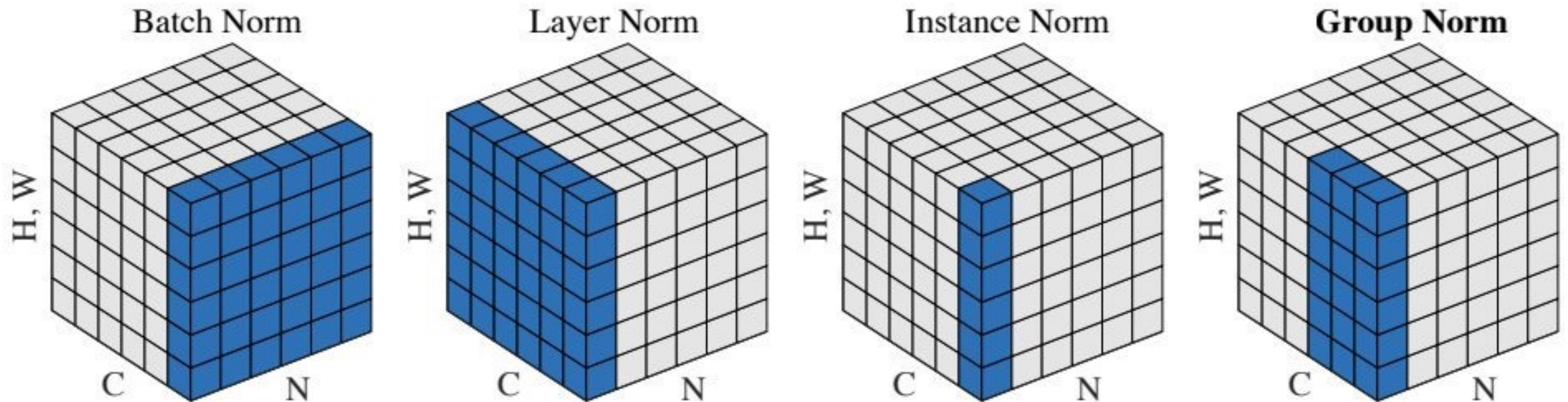
Normalization layers



Normalize w.r.t. the mean and variance of the activations of all the hidden units (neurons) on this layer (c) that process this particular location (h,w) in the image.

[Figure from Wu & He, arXiv 2018]

Normalization layers



Might as well…

[Figure from Wu & He, arXiv 2018]

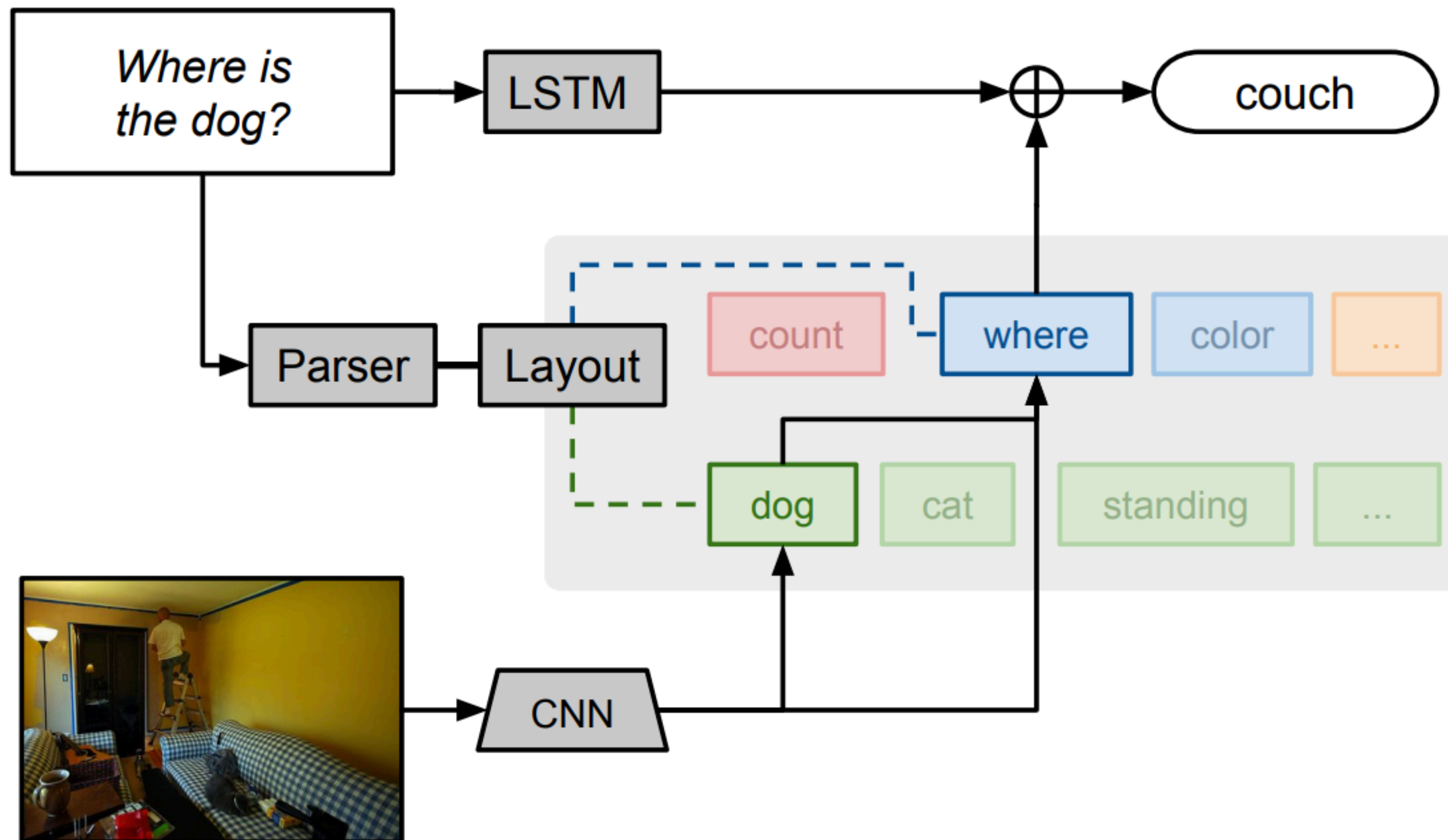
Differentiable programming

Deep nets are popular for at least two reasons:

1. High capacity
2. Easy to optimize (differentiable)

A more general term for models with these two properties is **differentiable programming**.

Differentiable programming



[Figure from “Neural Module Networks”, Andreas et al. 2017]

Next up:

1. Optimization of NNs (backprop)
2. CNNs