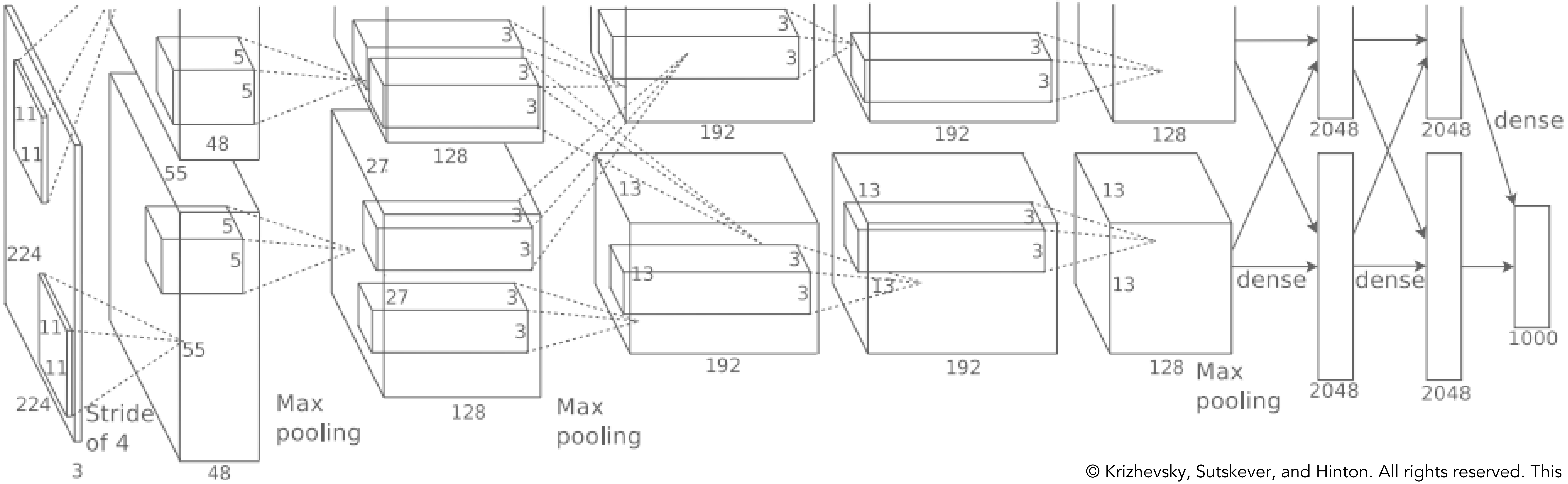
Lecture 1: Introduction to Deep Learning

Speaker: Sara Beery



What is “deep learning”?

1. **Neural nets**: A class of machine learning architectures that use stacks of linear transformations interleaved with pointwise nonlinearities

2. **Differentiable programming**: A programming paradigm where parameterize parts of the program and let gradient-based optimization tune the parameters

Course philosophy

- Breakthroughs in deep learning have been driven by a mixture of theory and practice, and both dimensions are vital for future progress in the field
- This course provides:
  - Theoretical grounding in important deep learning building blocks
  - Practice implementing, understanding, and using those blocks

AI Assistants Policy

- Our policy for using ChatGPT and other AI assistants is *identical* to our policy for using human assistants.
- This is a deep learning class and you *should* try out all the latest AI assistants (they are pretty much all using deep learning). It's very important to play with them to learn what they can do and what they can't do. That's a part of the content of this course.
- Just like you can come to office hours and ask a human questions (about the lecture material, clarifications about pset questions, tips for getting started, etc), you are very welcome to do the same with AI assistants.
- But: just like you are not allowed to ask an expert friend to do your homework for you, you also should not ask an expert AI.
- If it is ever unclear, just imagine the AI as a human and apply the same norm as you would with a human.
- If you work with any AI on a pset, briefly describe which AI and how you used it at the top of the pset (a few sentences is enough).

Why are we here?

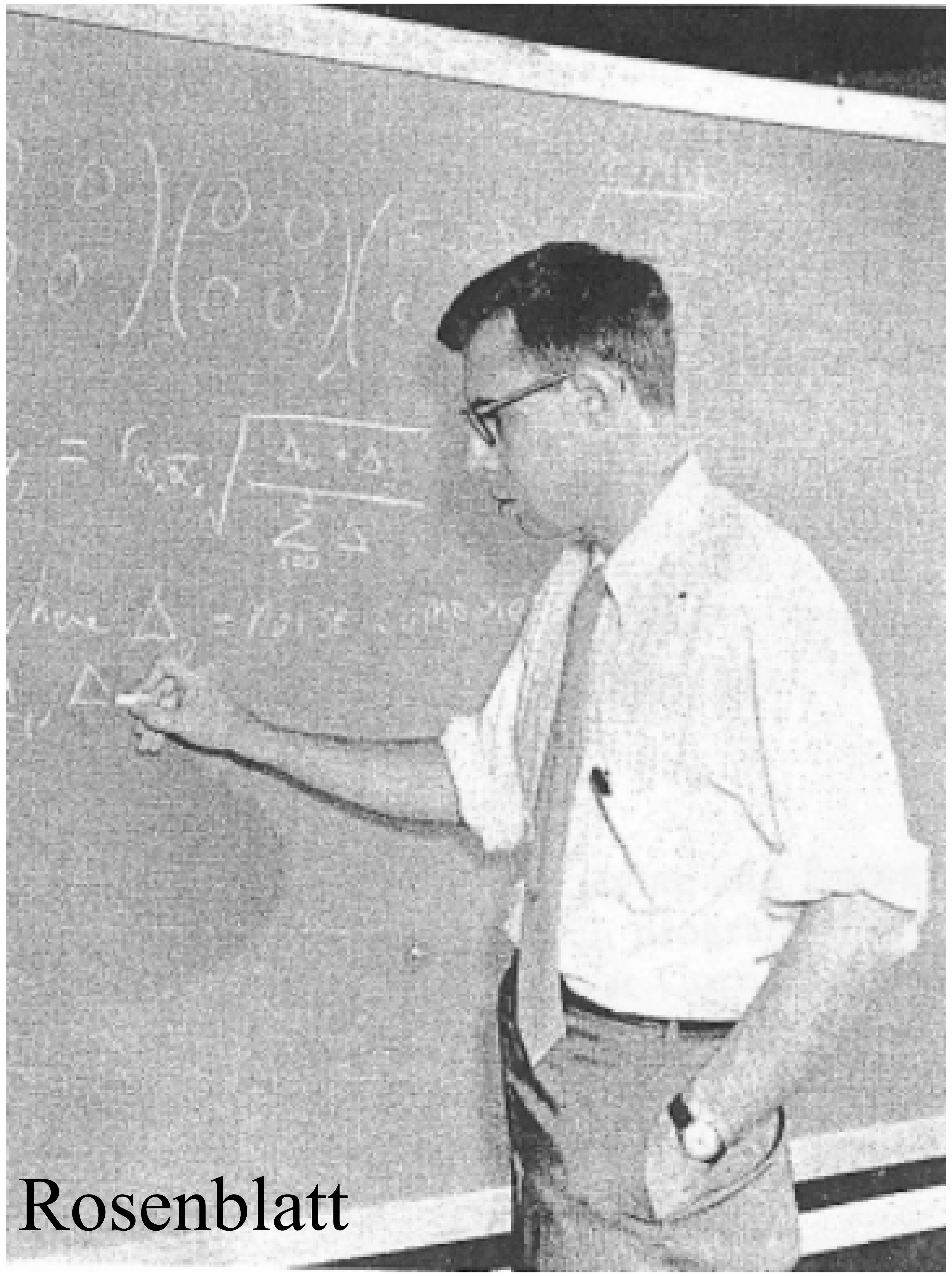
- **What's the goal?**
    - Model complex phenomena in the real world
- **What are complex phonemena?**
    - Natural language, Images, DNA, Ecosystems, Climate Change
- **Why is this hard?**
    - See: complex
- **Existence proof for deep learning as a solution:** The human brain?

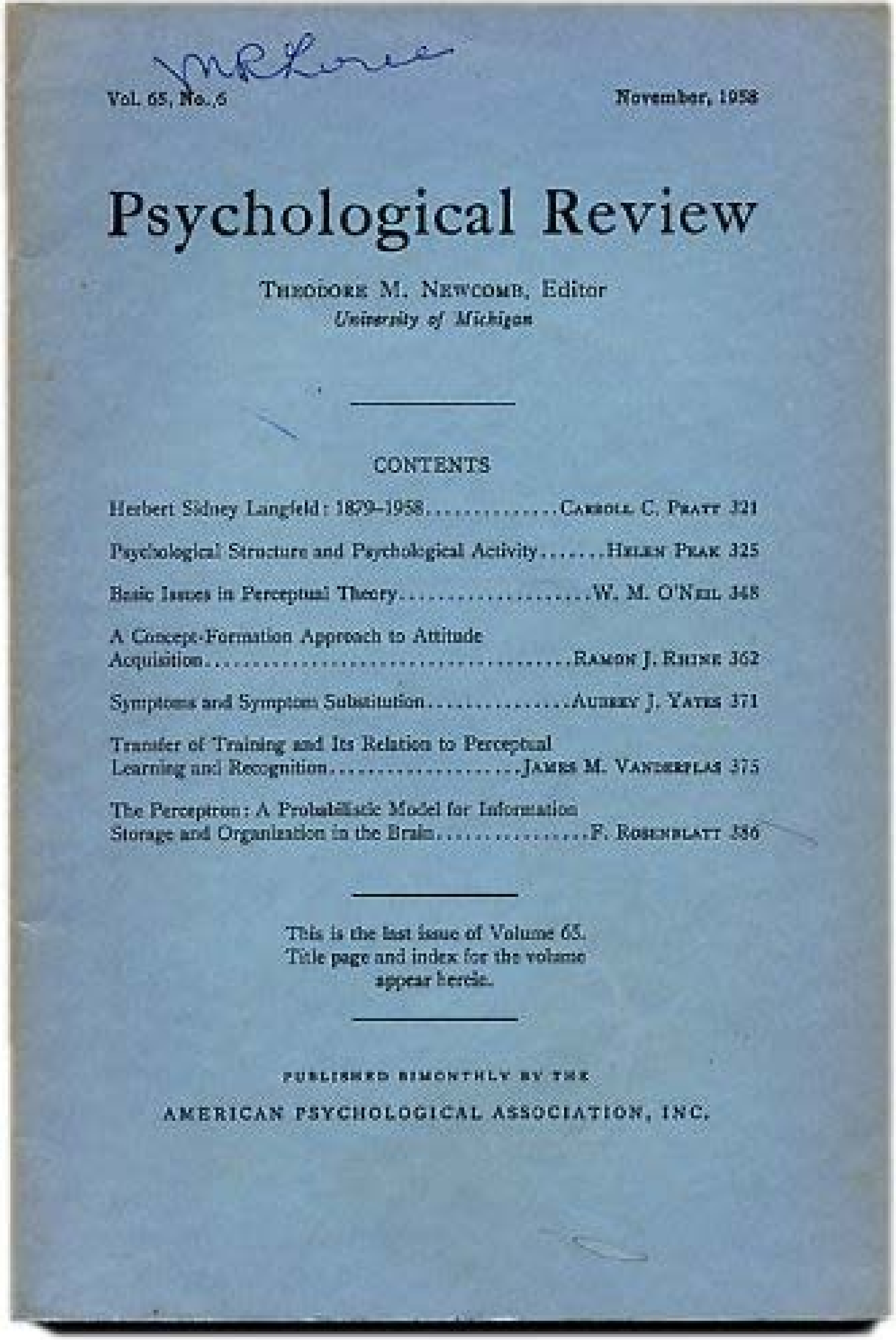
1. Introduction to Deep Learning

- How did we get where we are today? (Brief History)
- What we expect you have seen before (ok if you haven’t!)
- What we will cover in this class

A brief history of Neural Networks

Perceptrons, 1958




Vol. 65, No. 6 November, 1958

Psychological Review

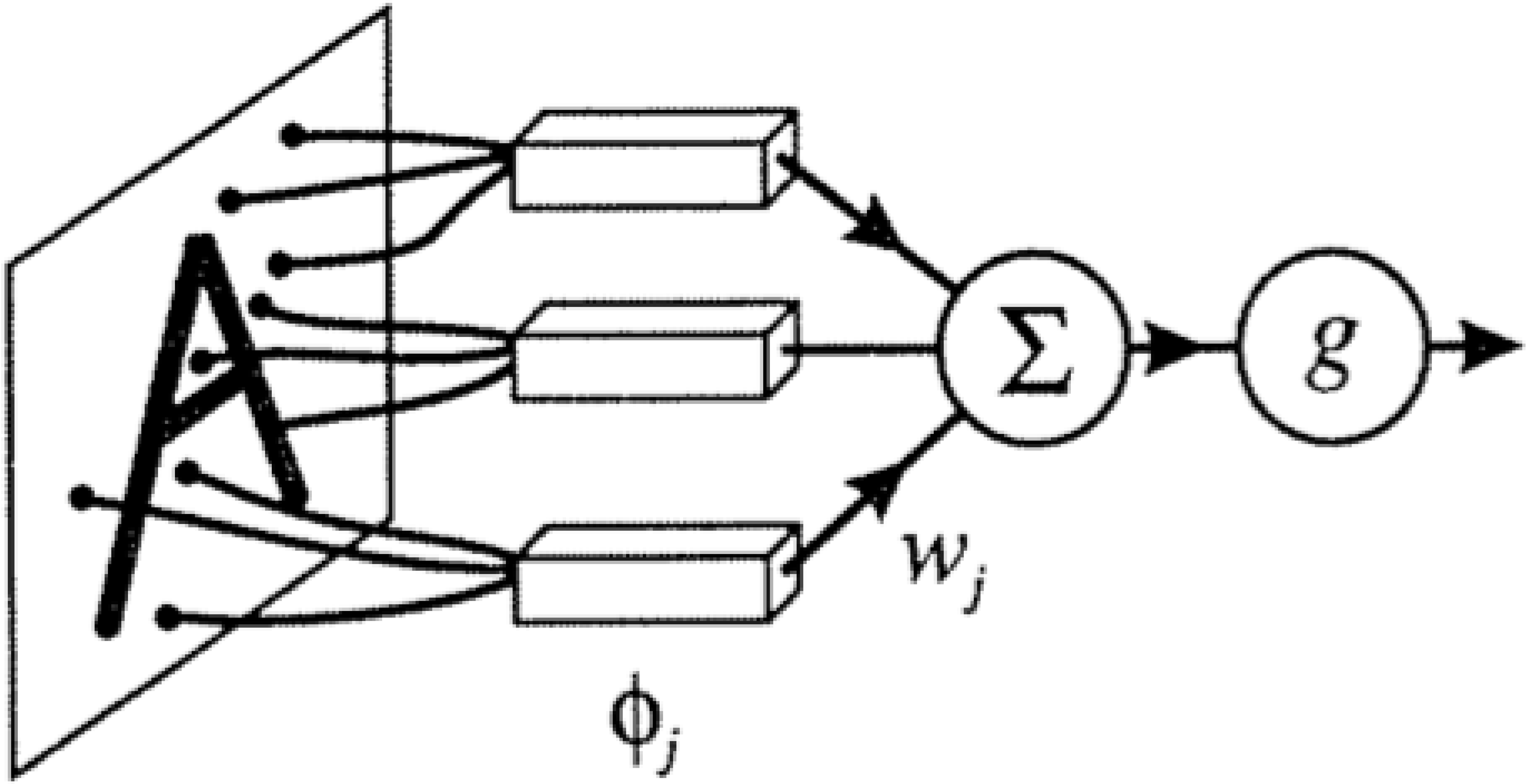
Theodore M. Newcomb, Editor
University of Michigan

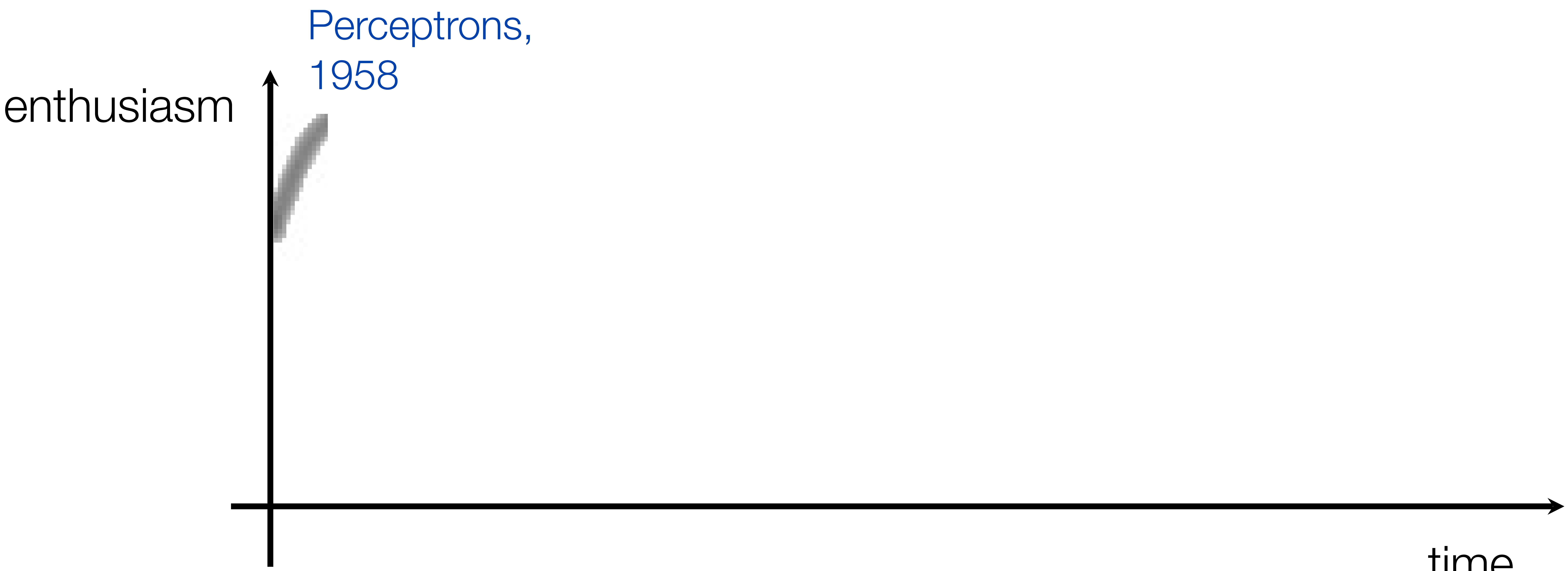
CONTENTS

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

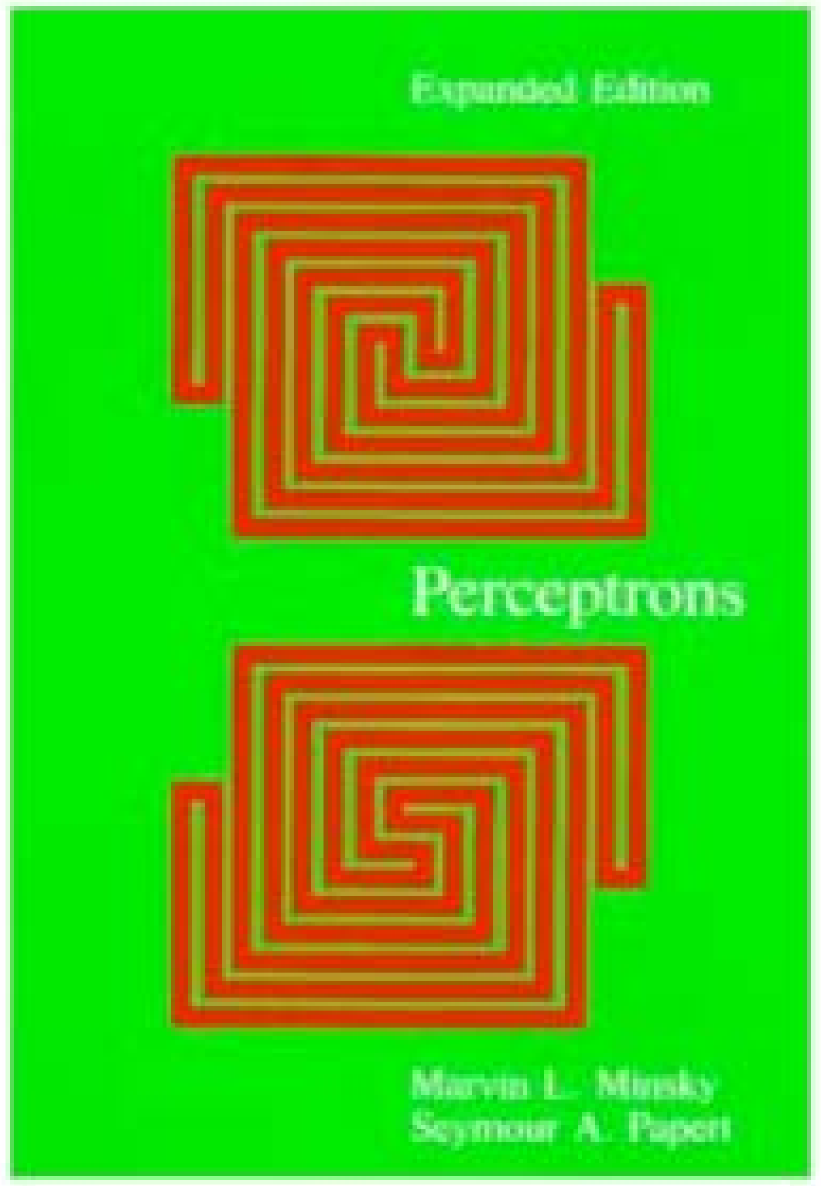
Published bimonthly by the
American Psychological Association, Inc.

Perceptrons, 1958




Perceptrons,
1958
enthusiasm
time

Minsky and Papert, Perceptrons, 1972



FOR BUYING OPTIONS, START HERE

Select Shipping Destination

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

Perceptrons, expanded edition

An Introduction to Computational Geometry

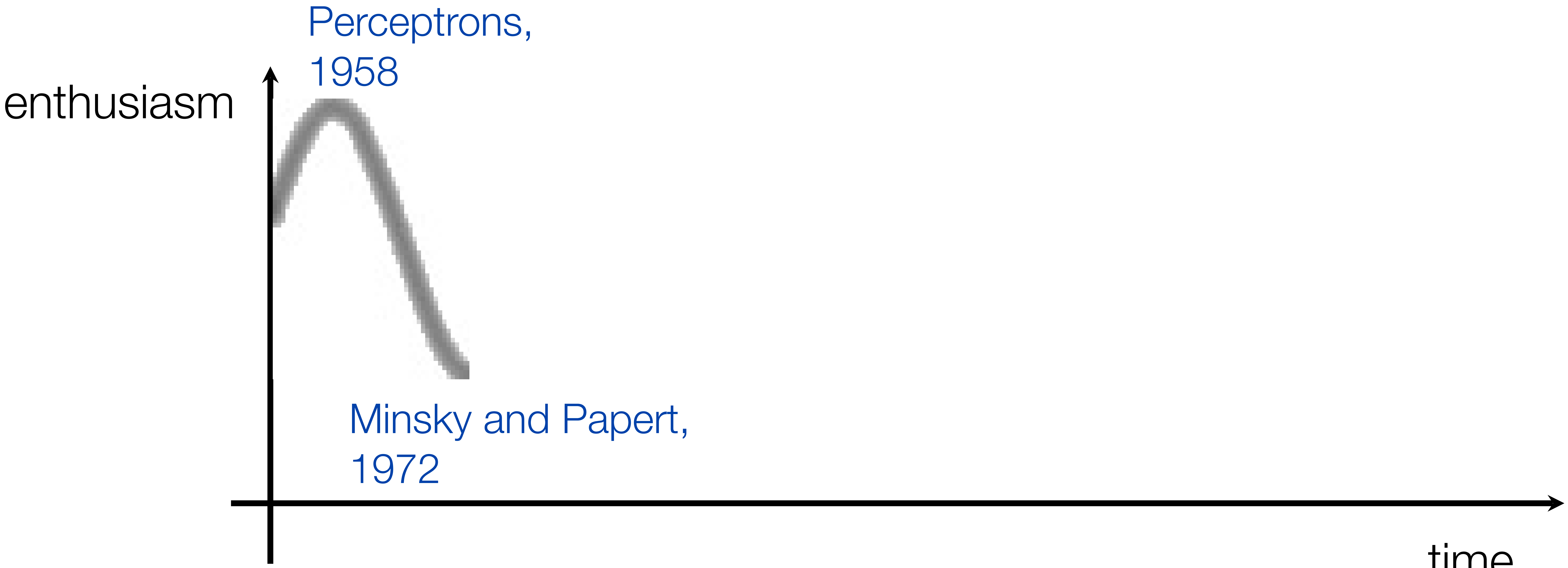
By Marvin Minsky and Seymour A. Papert

Overview

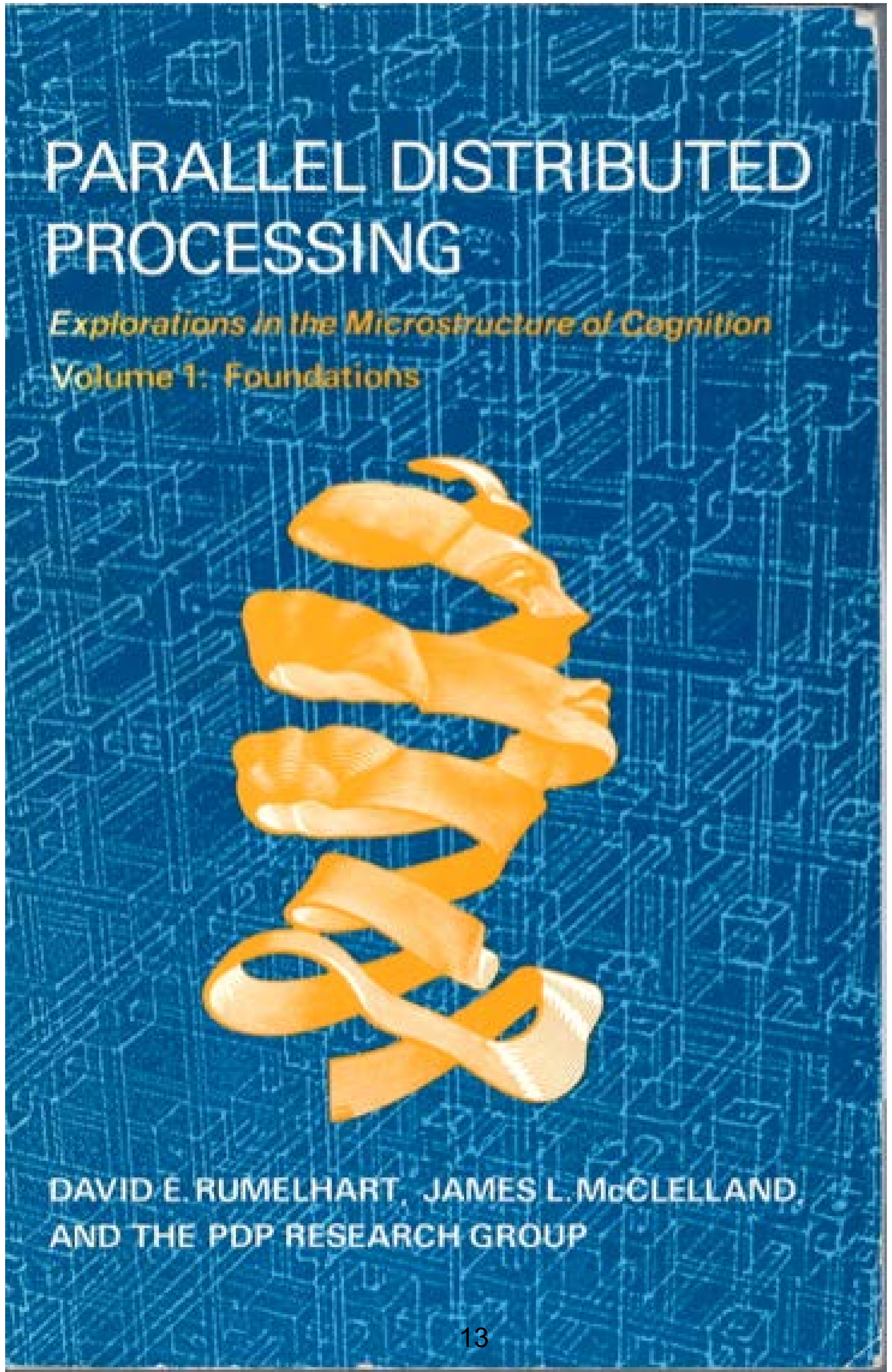
Perceptrons - the first systematic study of parallelism in computation - has remained a classical work on threshold automata networks for nearly two decades. It marked a historical turn in artificial intelligence, and it is required reading for anyone who wants to understand the connectionist counterrevolution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

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

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

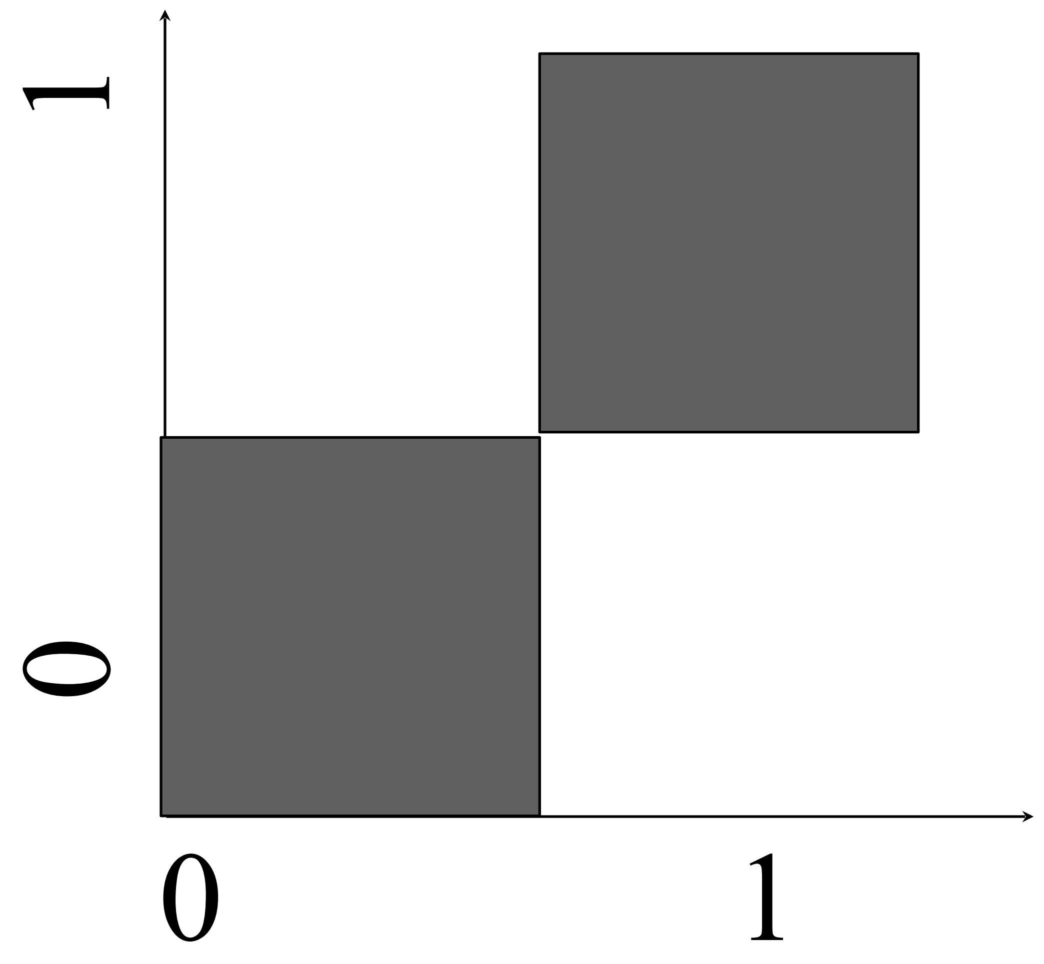

Perceptrons,
1958
enthusiasm
Minsky and Papert,
1972
time

Parallel Distributed Processing (PDP), 1986

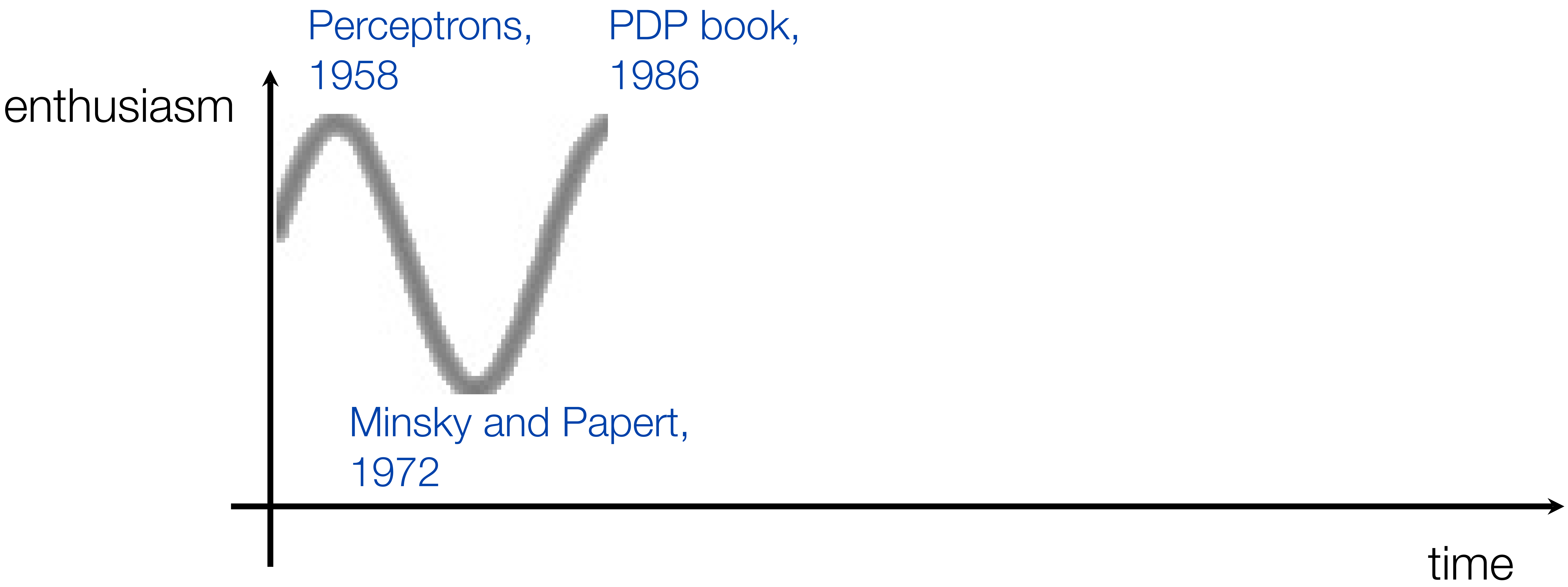


XOR problem

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



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


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

LeCun conv nets, 1998

PROC. OF THE IEEE, NOVEMBER 1998 7

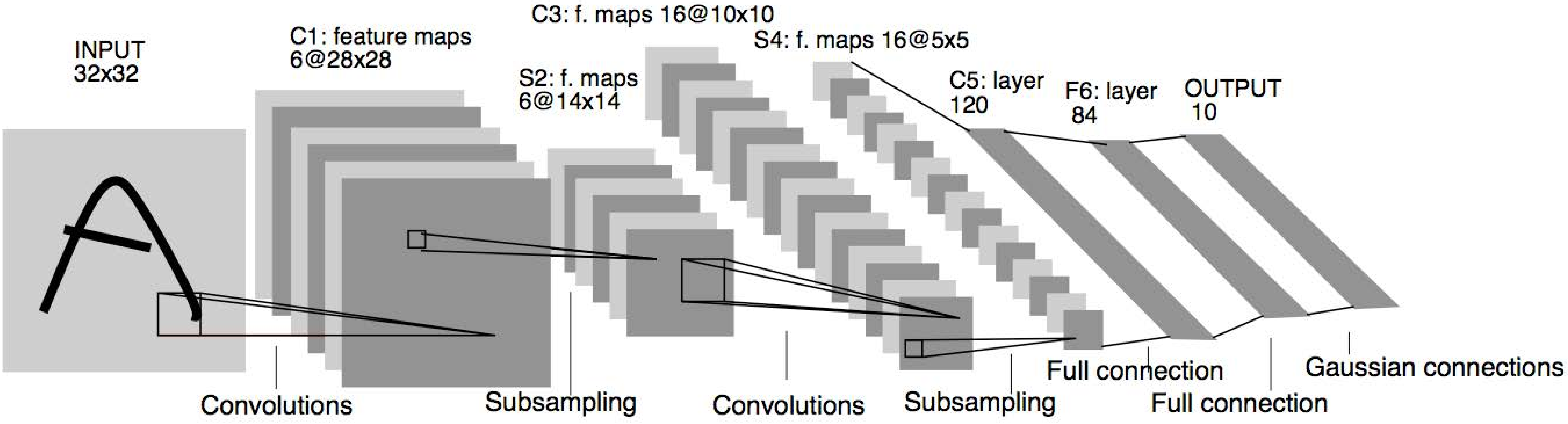


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Demos:

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

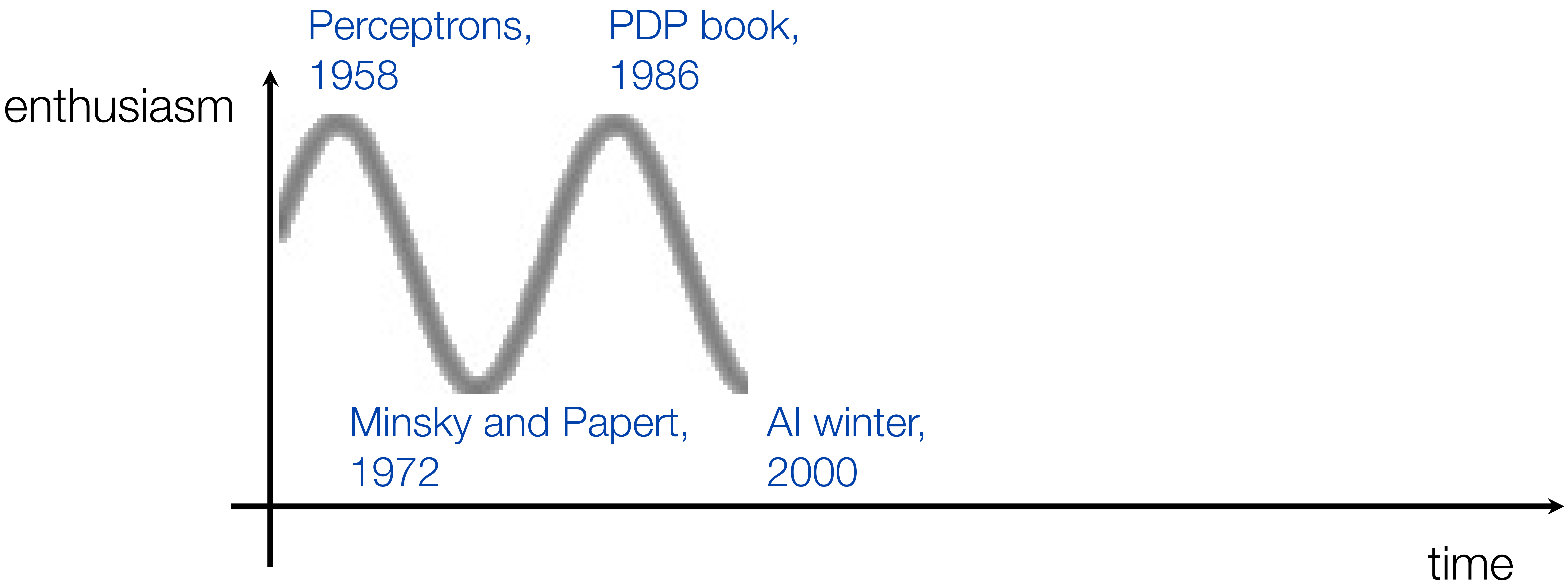
Neural Information Processing Systems 2000

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

- For the 2000 conference:

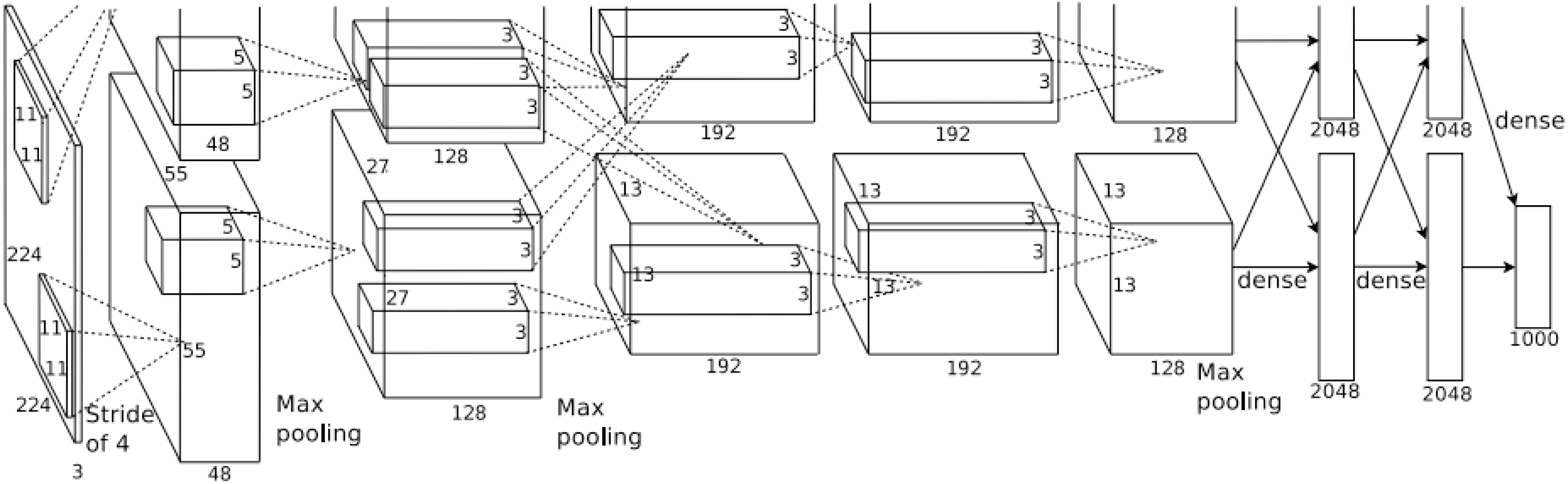
  – title words predictive of paper acceptance: "Belief Propagation" and "Gaussian".

  – title words predictive of paper rejection: "Neural" and "Network".

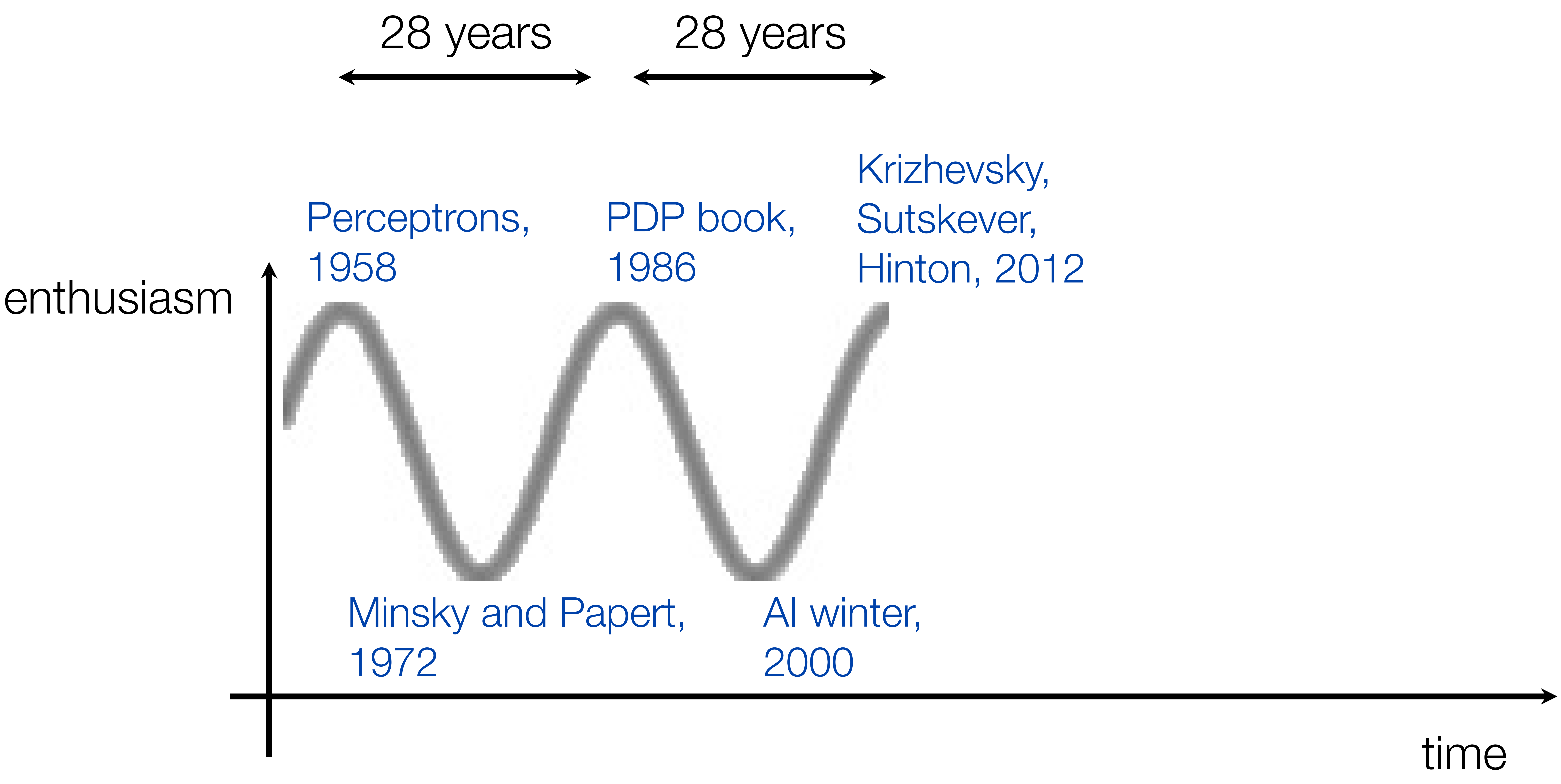

Perceptrons,
1958
PDP book,
1986
enthusiasm
Minsky and Papert,
1972
AI winter,
2000
time

Krizhevsky, Sutskever, and Hinton, NeurIPS 2012

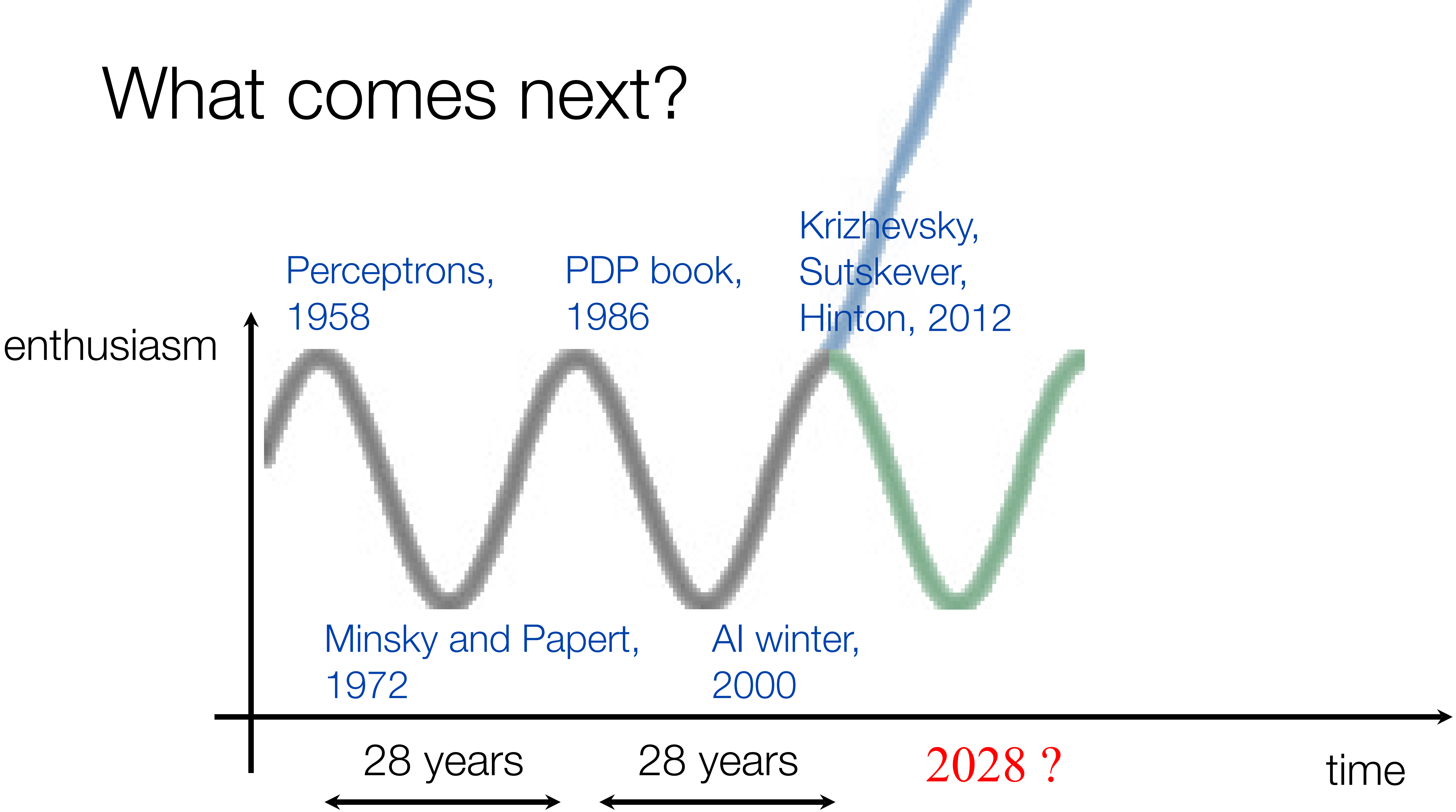
“Alexnet”



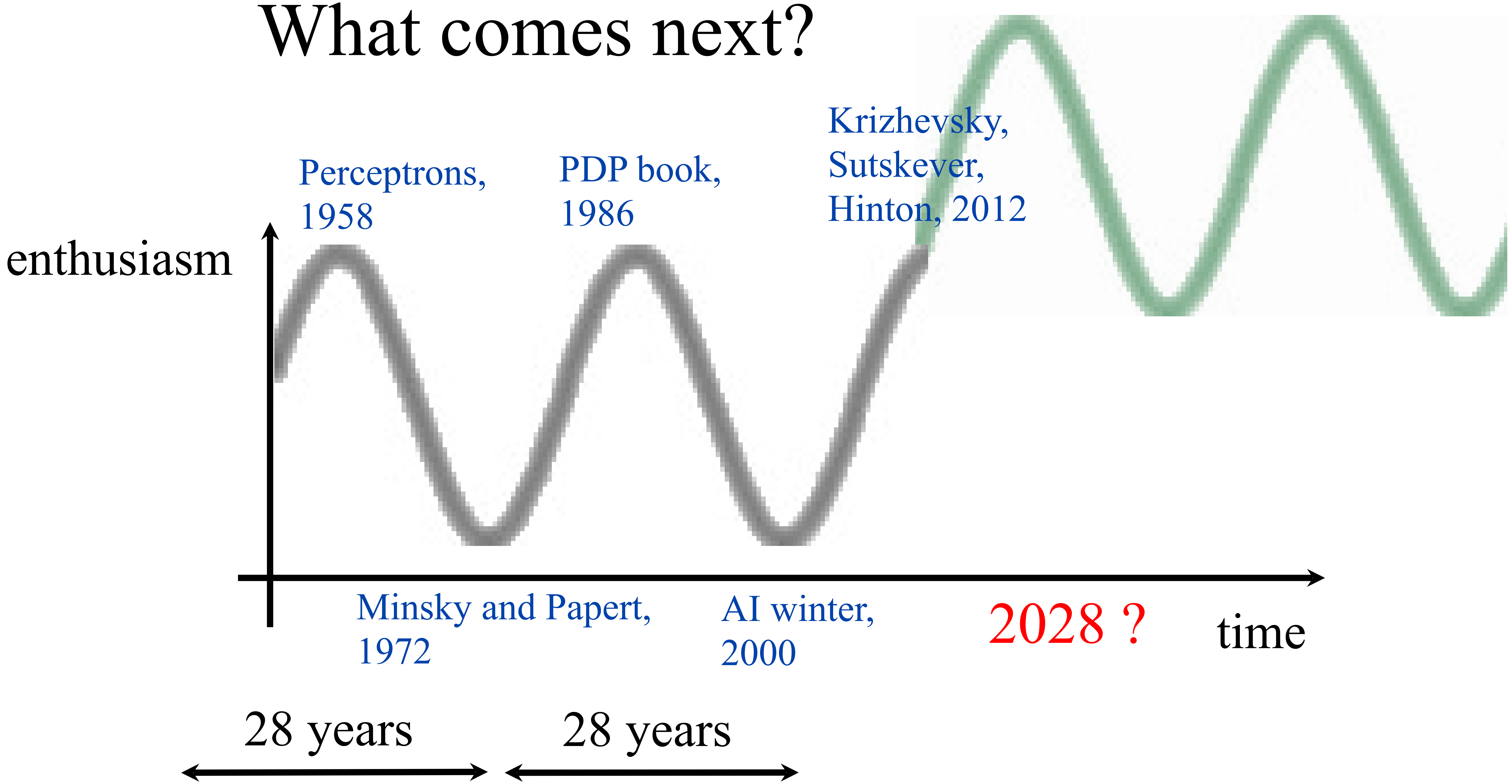
Krizhevsky, Sutskever, and Hinton, NeurIPS 2012


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

What comes next?



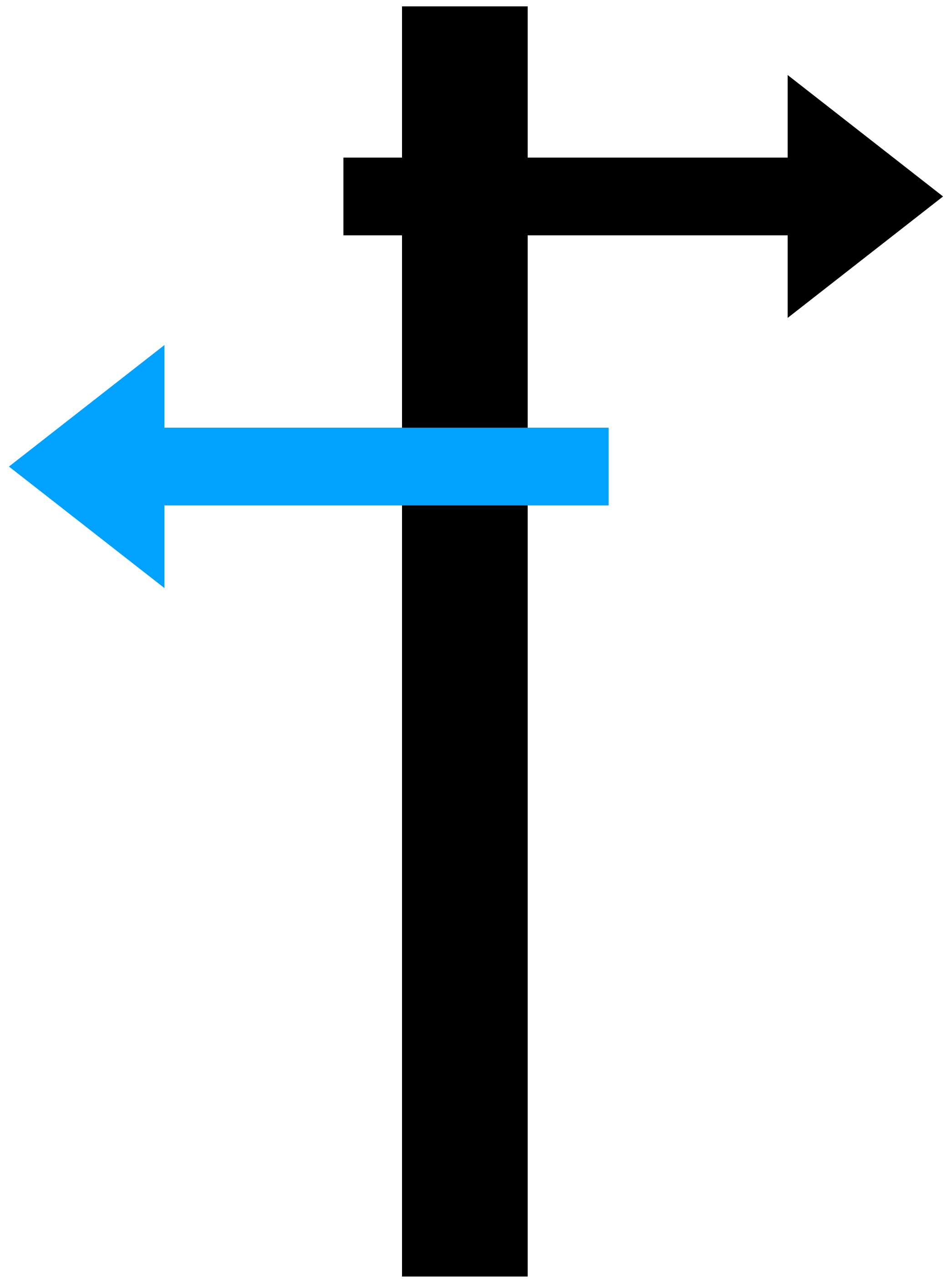
What comes next?



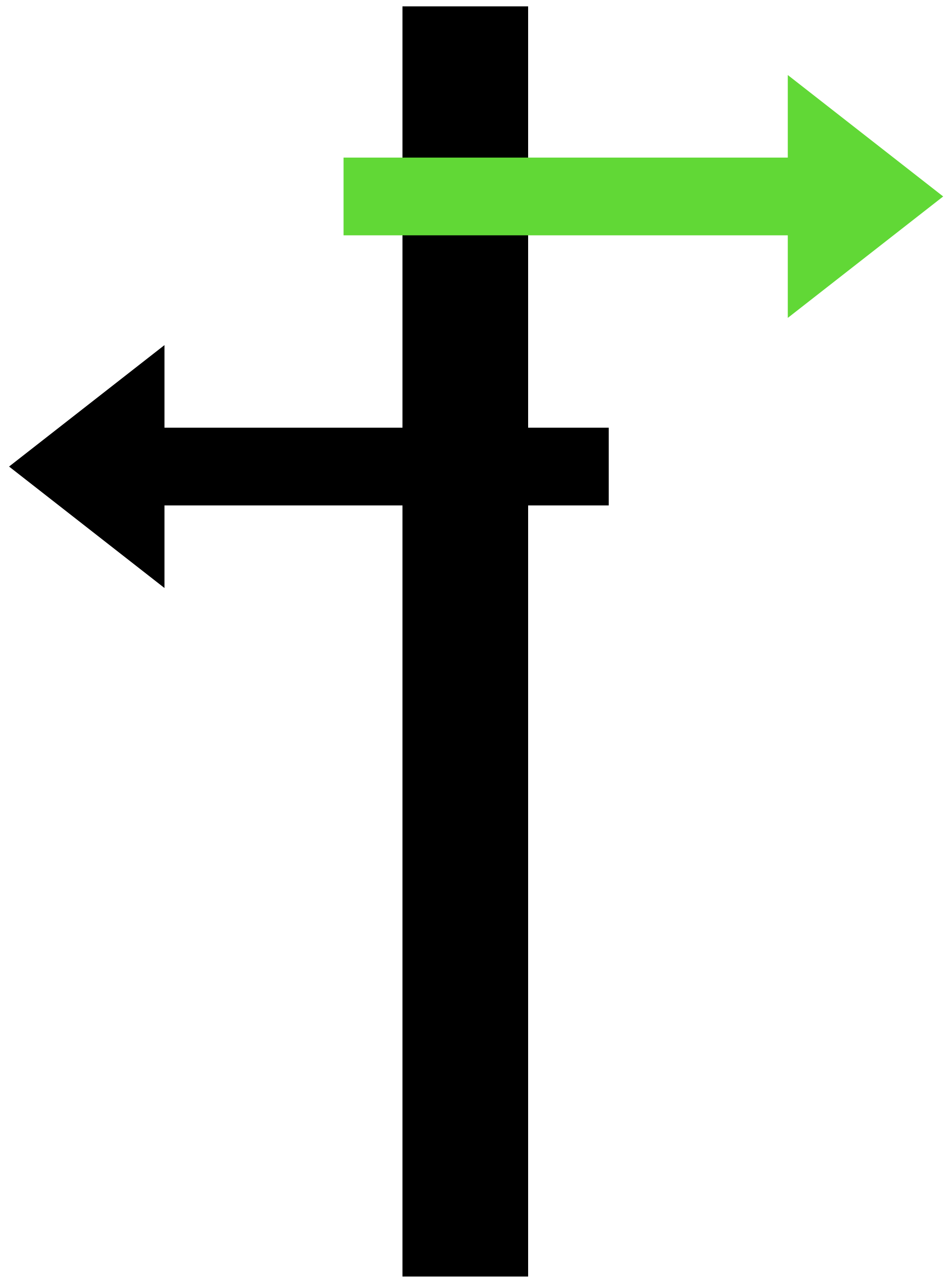
What is deep learning today?

- Autograd (pytorch, tensorflow)
- Billion+ data point datasets
- Parallel training on thousands of GPUs
- Billion+ parameter architectures
- Million+ dollar training costs
- Shockingly good results
- Massive isn't necessary - e.g. Stable Diffusion
- Open source community and modular reuse

Signposting for the rest of the lecture



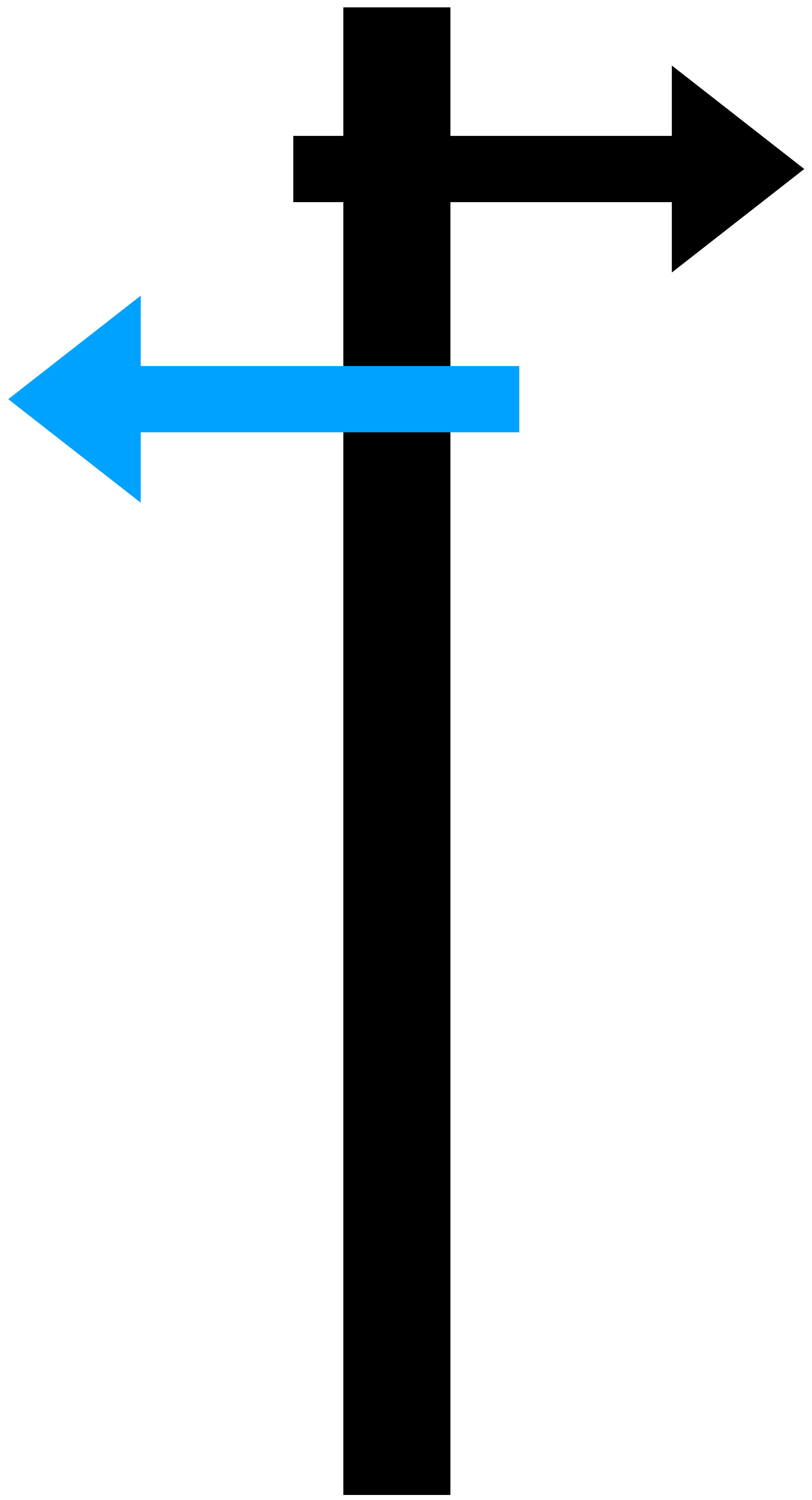
What we expect you to have seen before



What we will cover in this class

What we expect you to have seen before

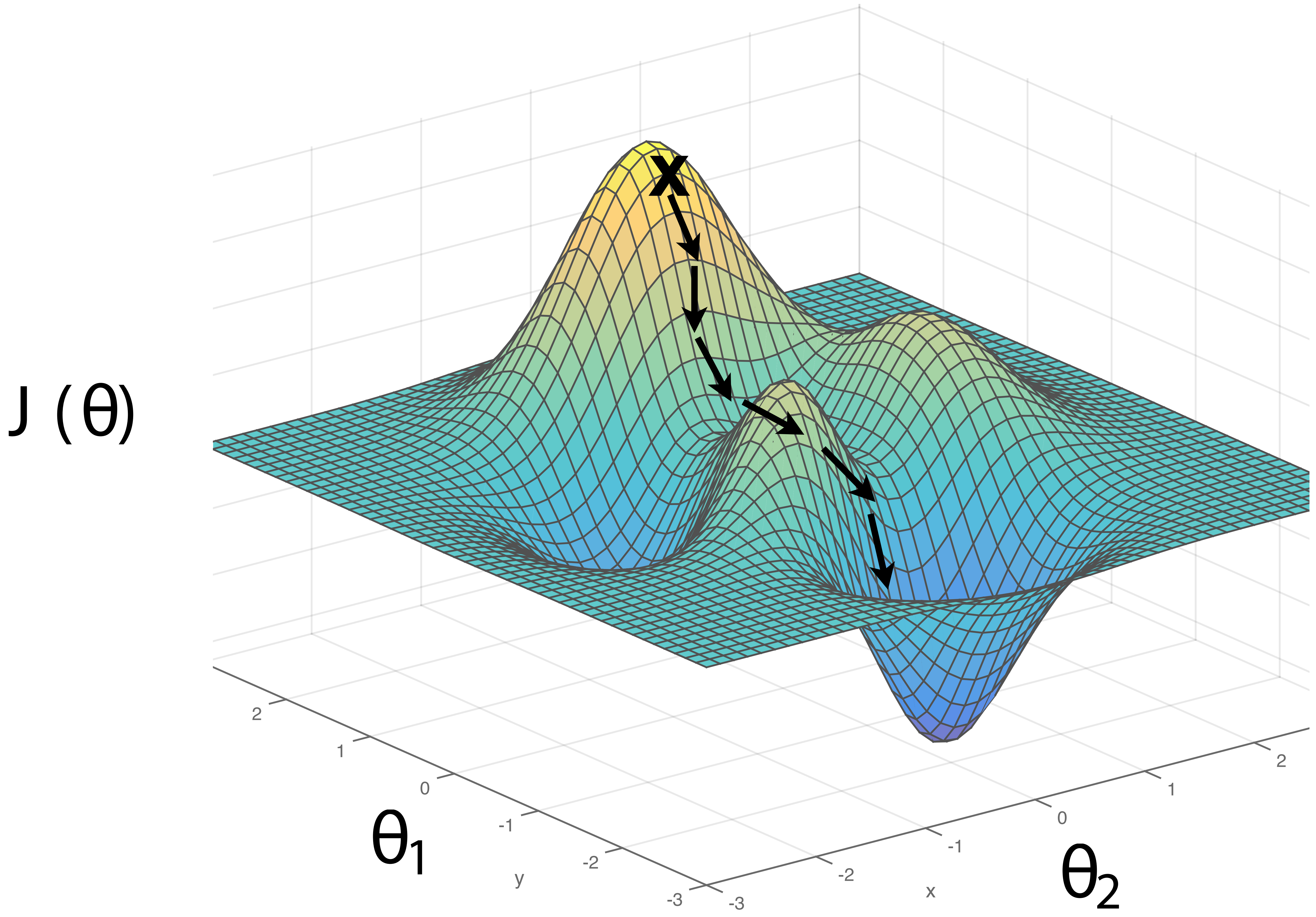
- Gradient descent



Gradient descent

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

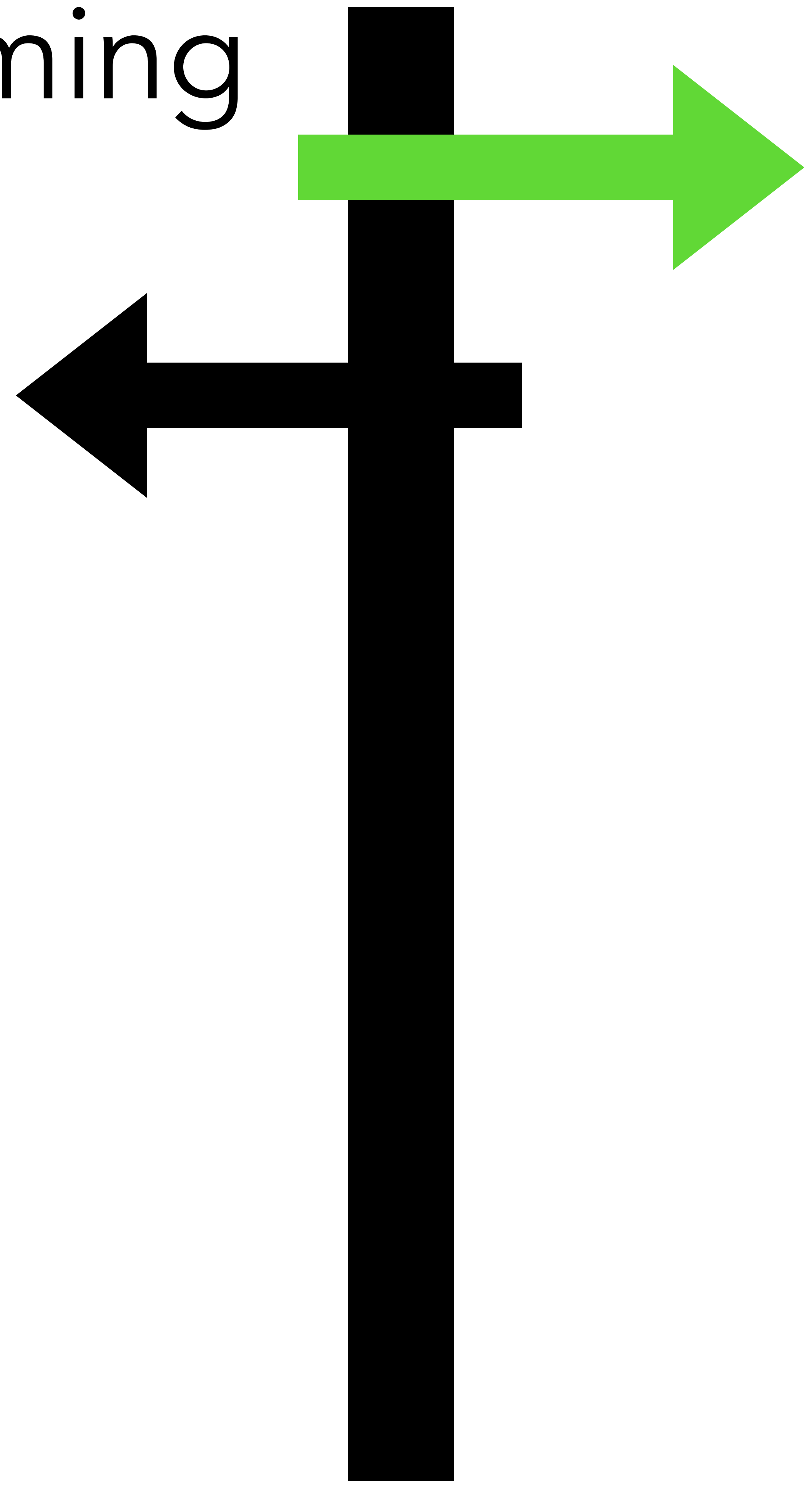
Gradient descent



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

What we'll cover in this class

- Backprop and differentiable programming



Gradient descent

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

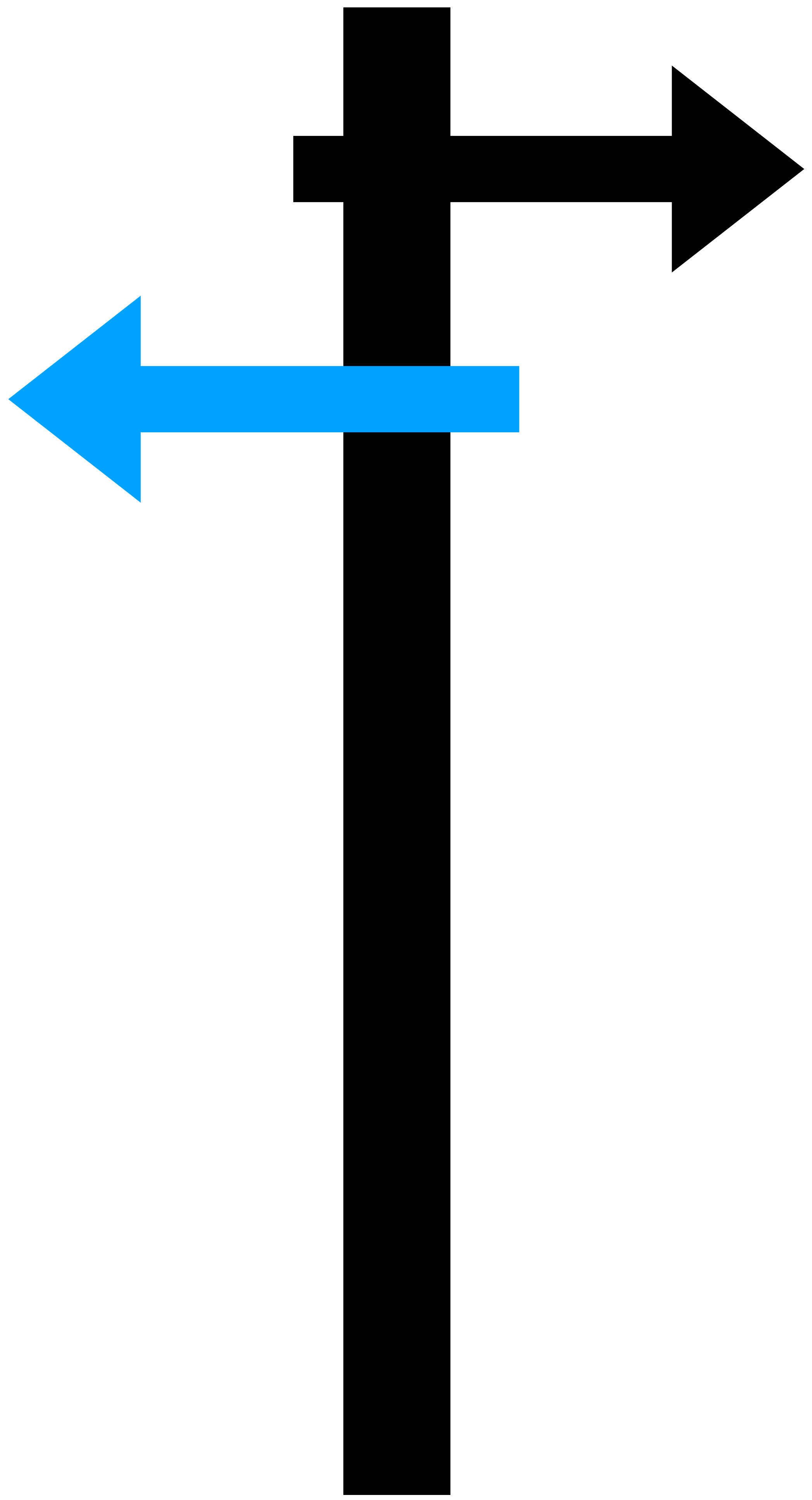
Lecture 2: Backprop and Differentiable Programming

$$\theta^{t+1} = \theta^{t} - \eta_t \frac{\partial \ldots}{\partial \theta}\Big|_{\theta=\theta^t}$$

learning rate

What we expect you to have seen before

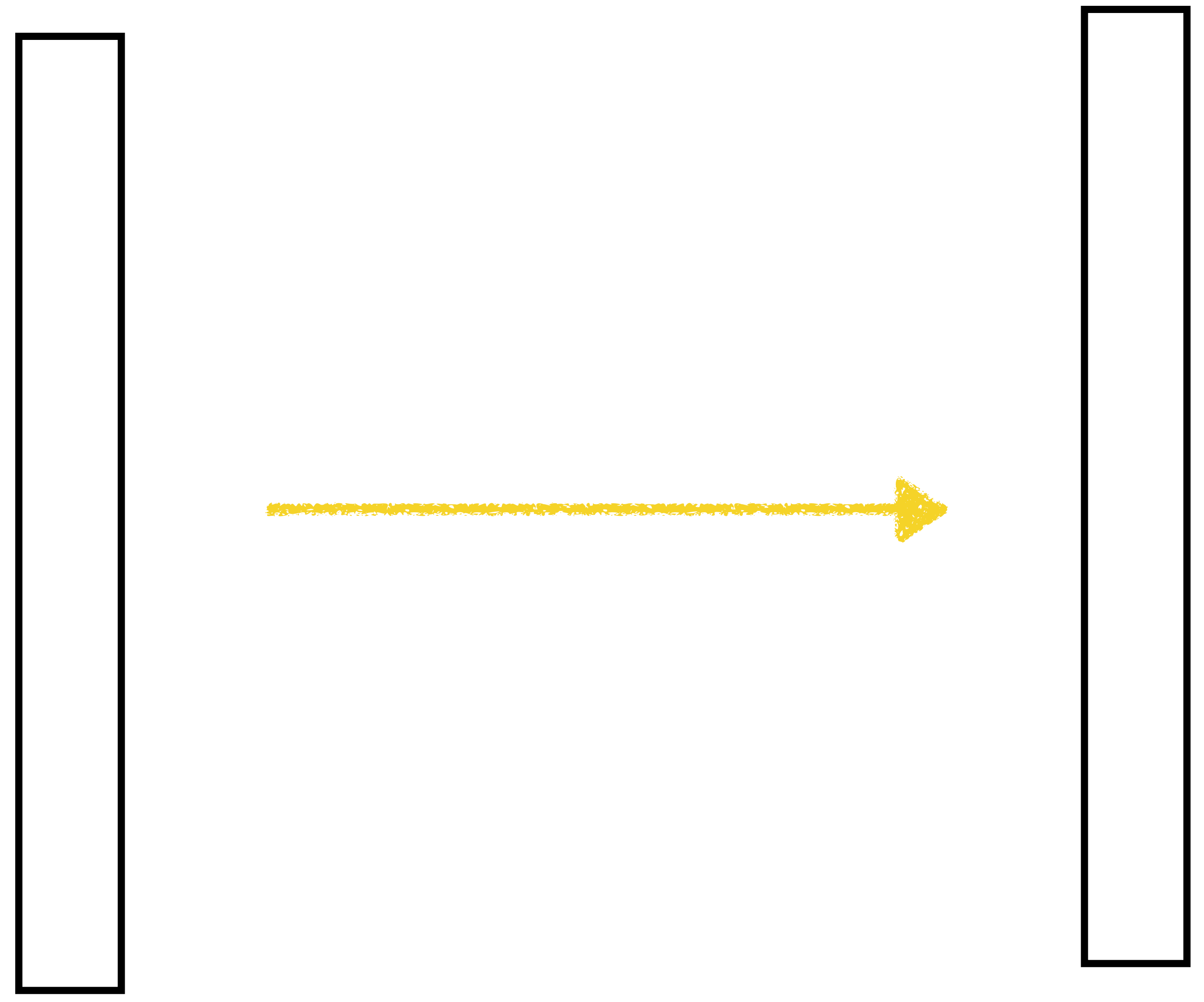
- Gradient descent
- MLPs, Nonlinearities (ReLu)



Computation in a neural net

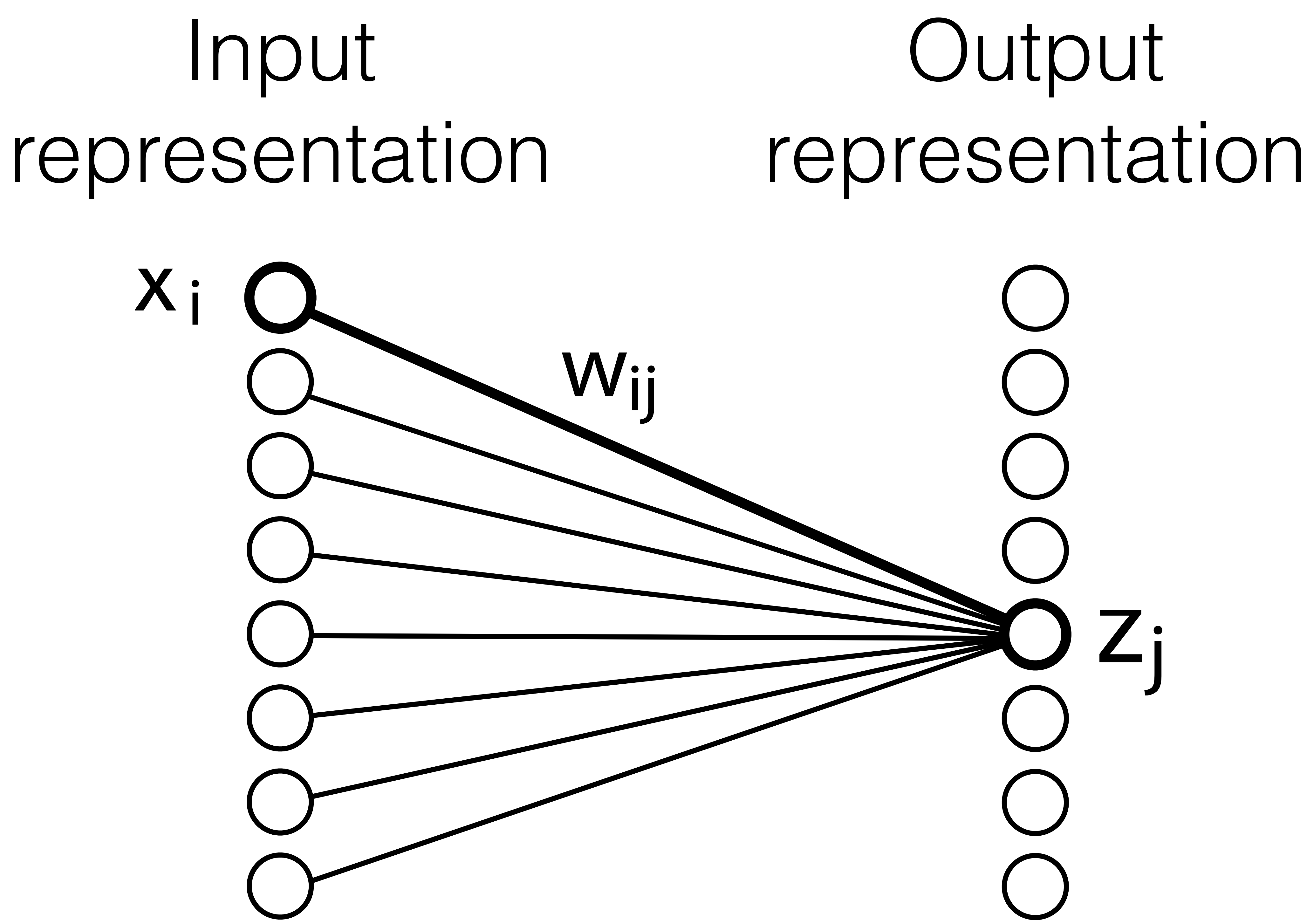
Input representation

Output representation



Computation in a neural net

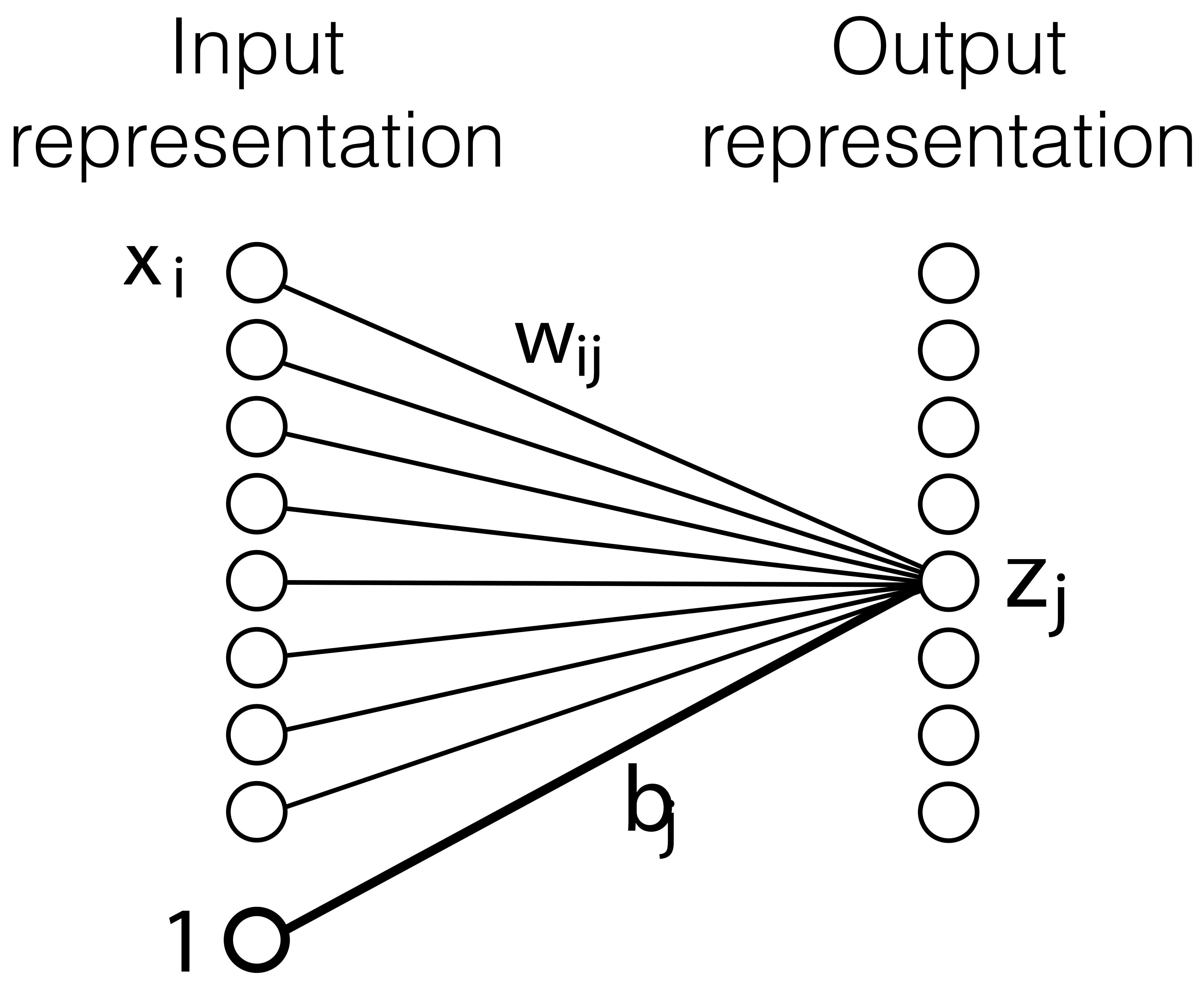
Linear layer



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

Computation in a neural net

Linear layer



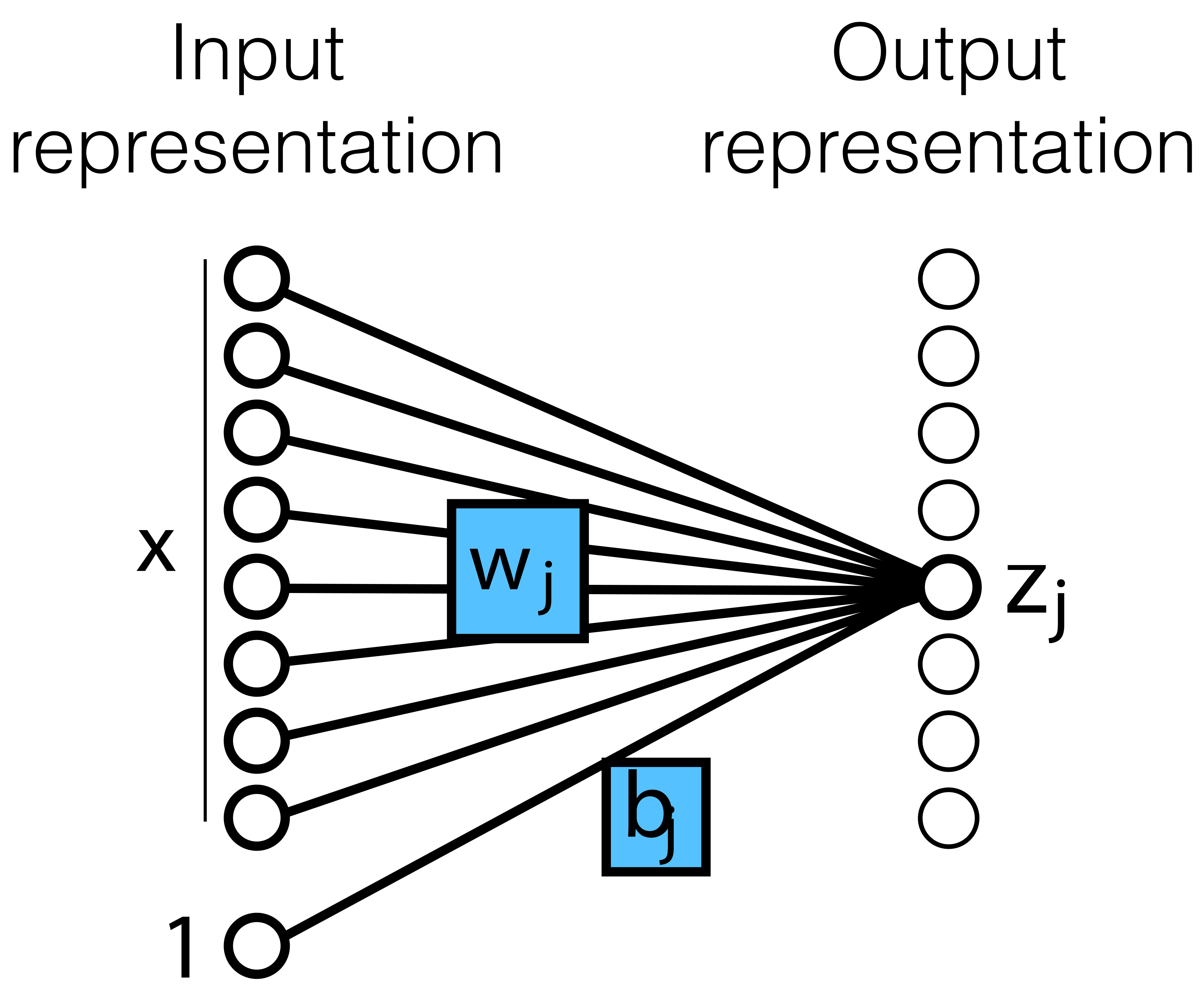
$$z_j = \sum_i w_{ij} x_i + b_j$$

weights

bias

Computation in a neural net

Linear layer



weights

$$z_j = x^T w_j + b_j$$

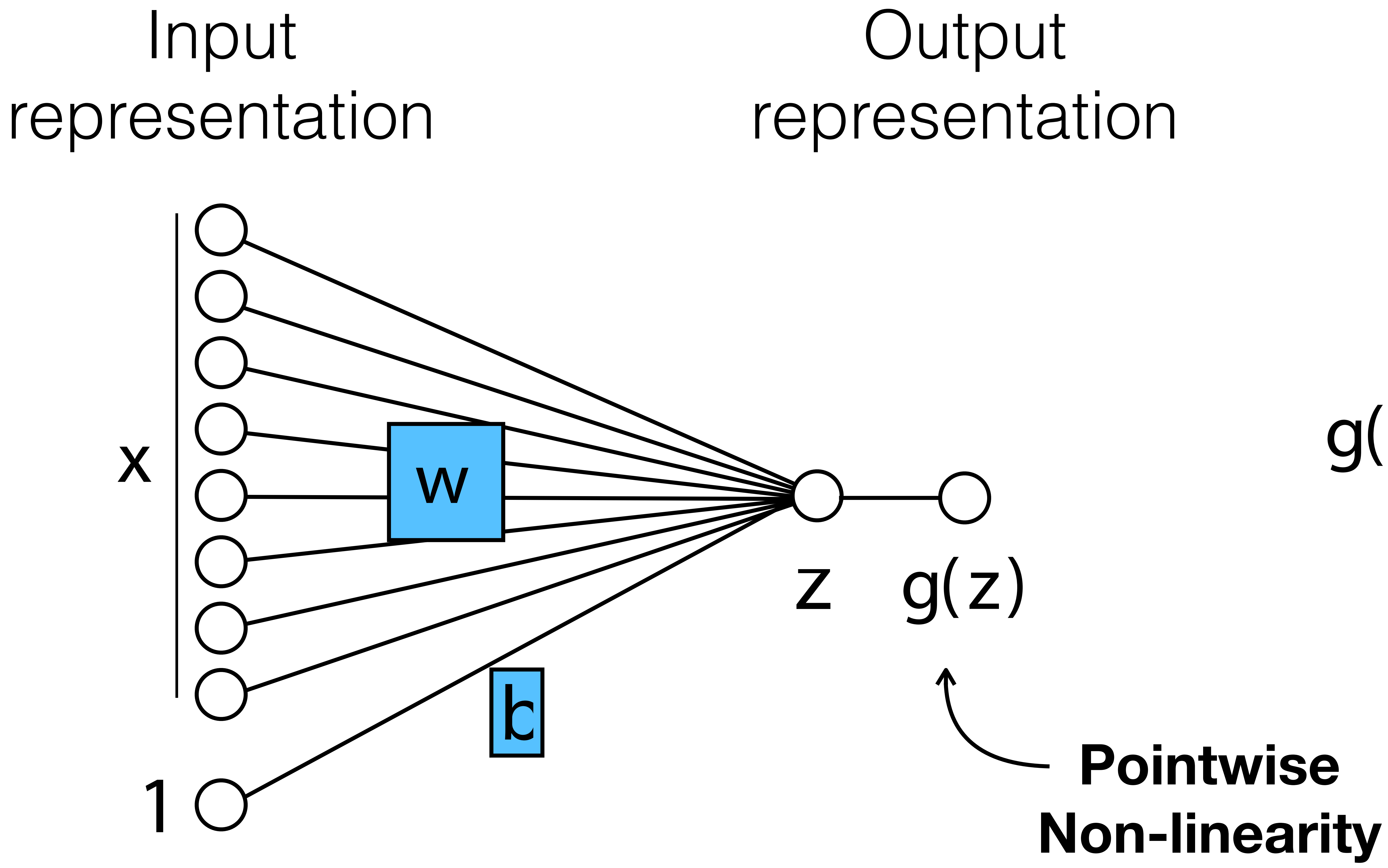
bias

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

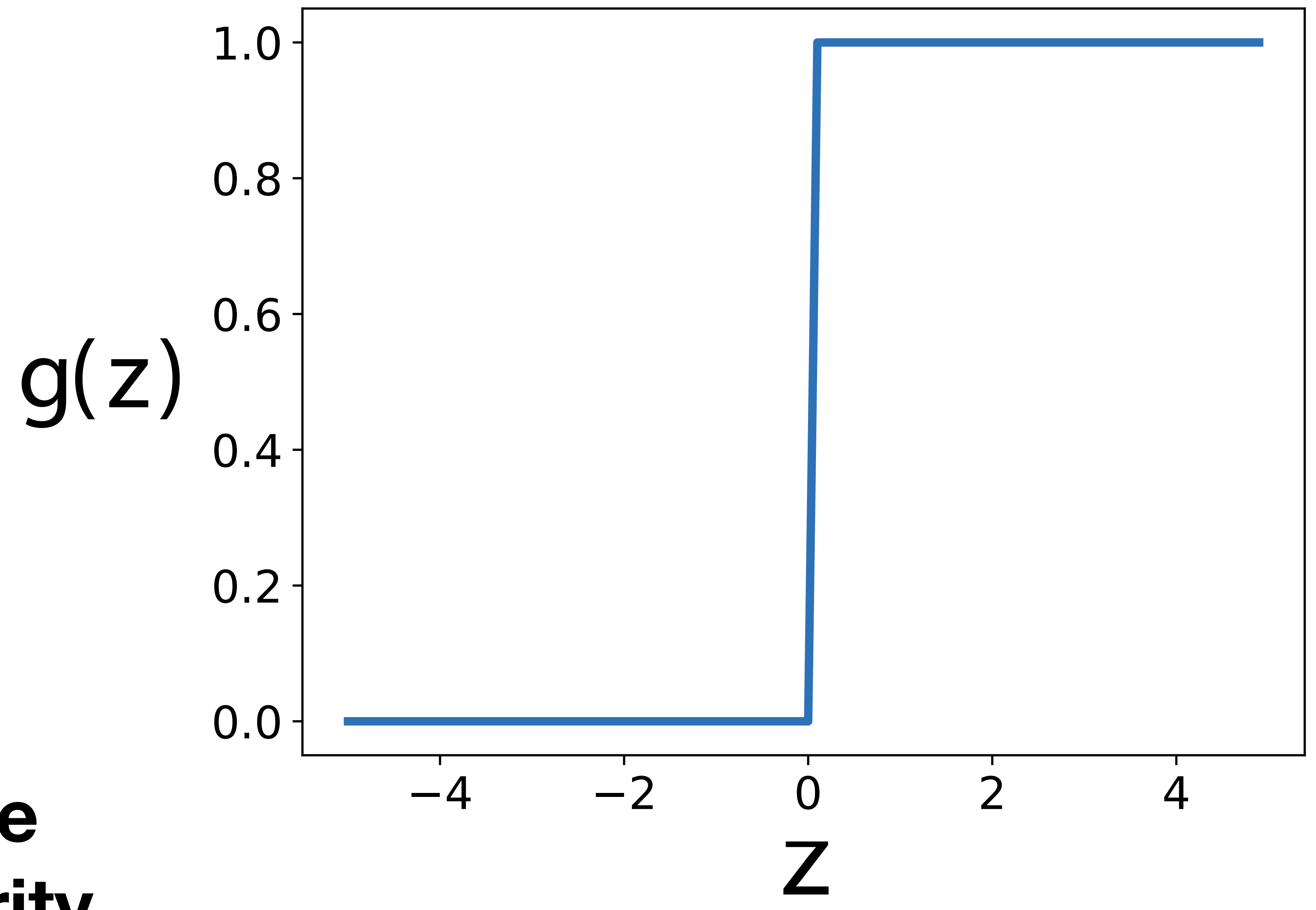
parameters of the model

Computation in a neural net

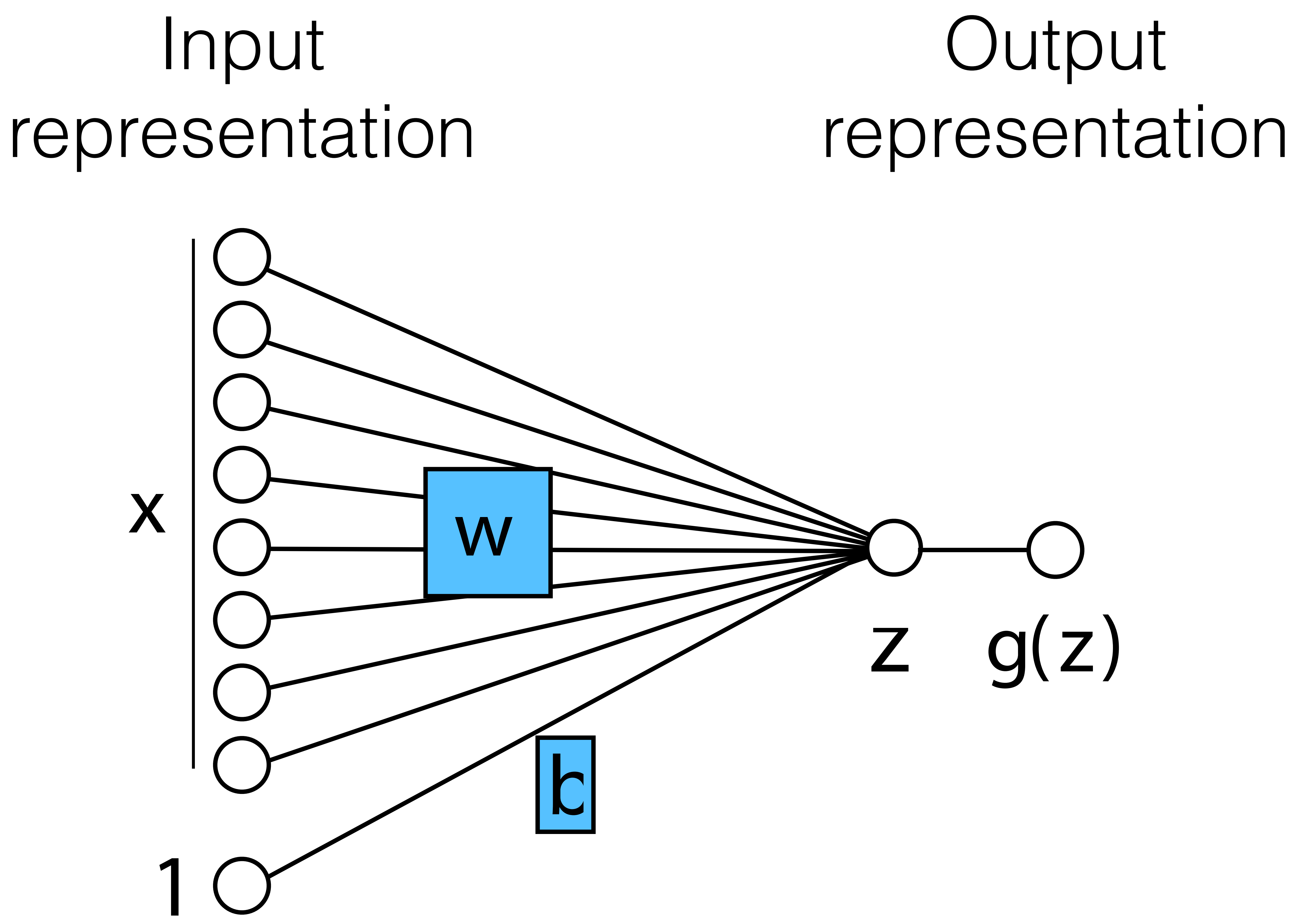
“Perceptron”



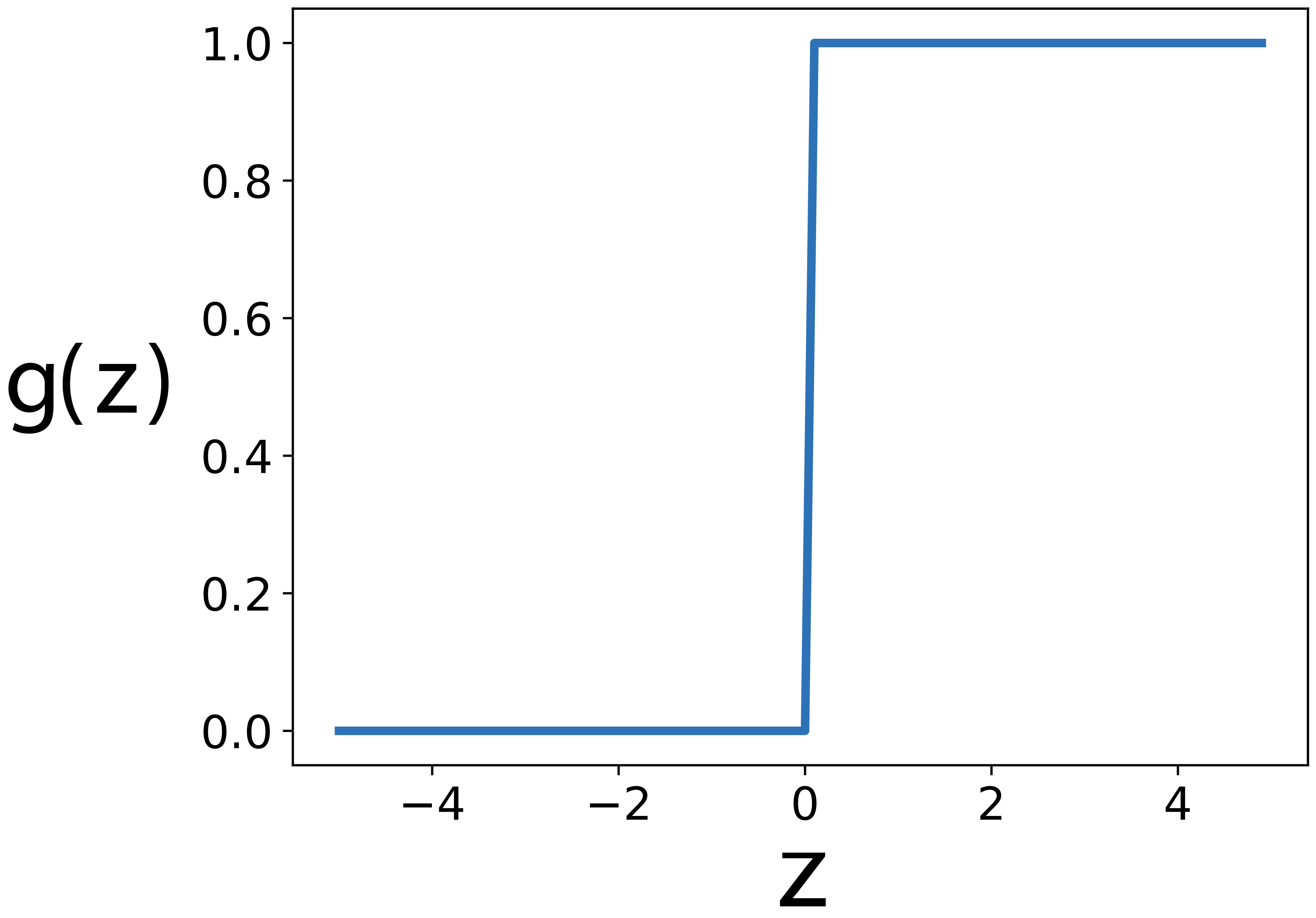
$$g(z) = \begin{cases} 1, & \text{if} \quad z > 0 \\ 0, & \text{otherwise} \end{cases}$$



Computation in a neural net



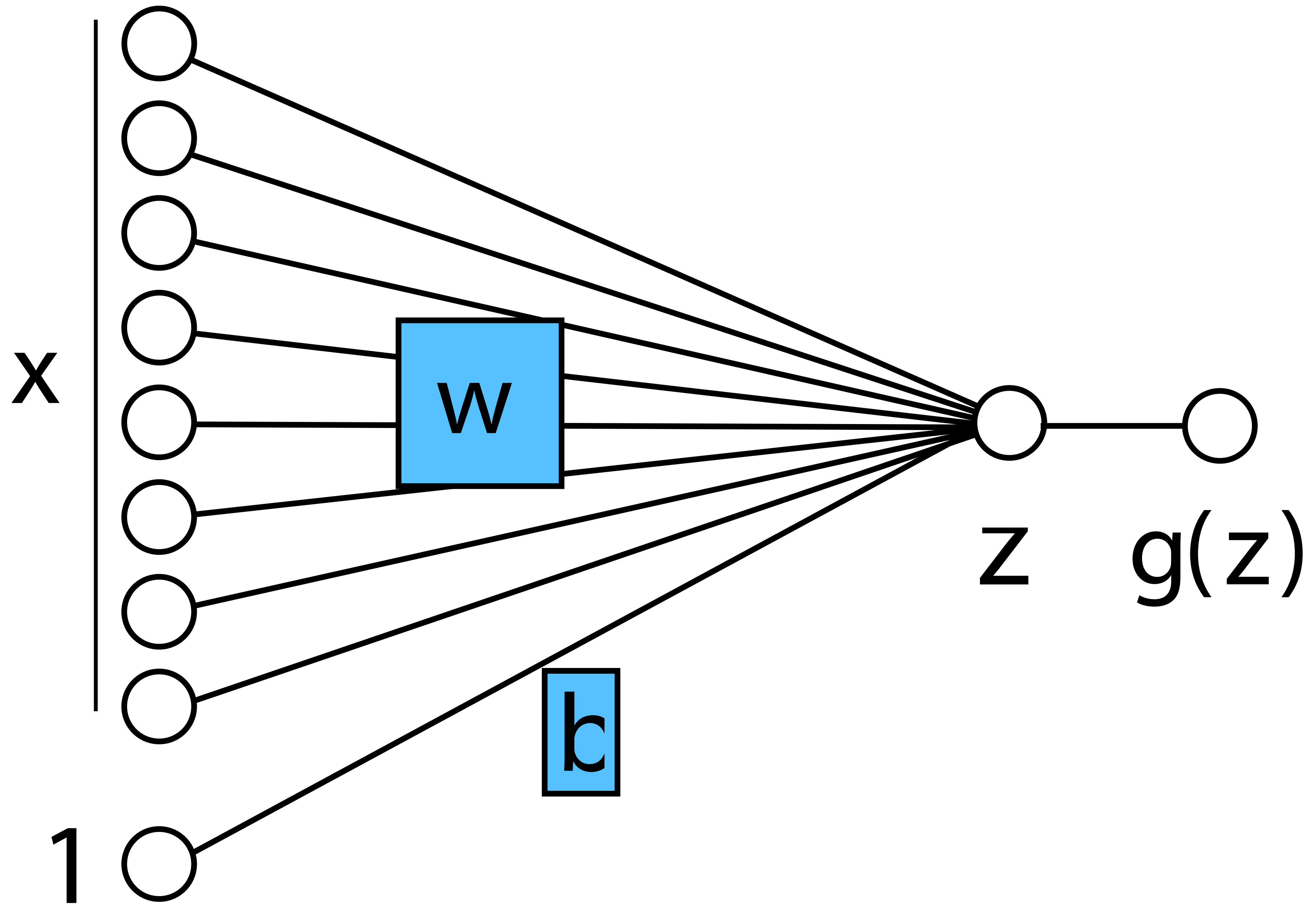
$$g(z) = \begin{cases} 1, & \text{if} \quad z > 0 \\ 0, & \text{otherwise} \end{cases}$$



Computation in a neural net — nonlinearity

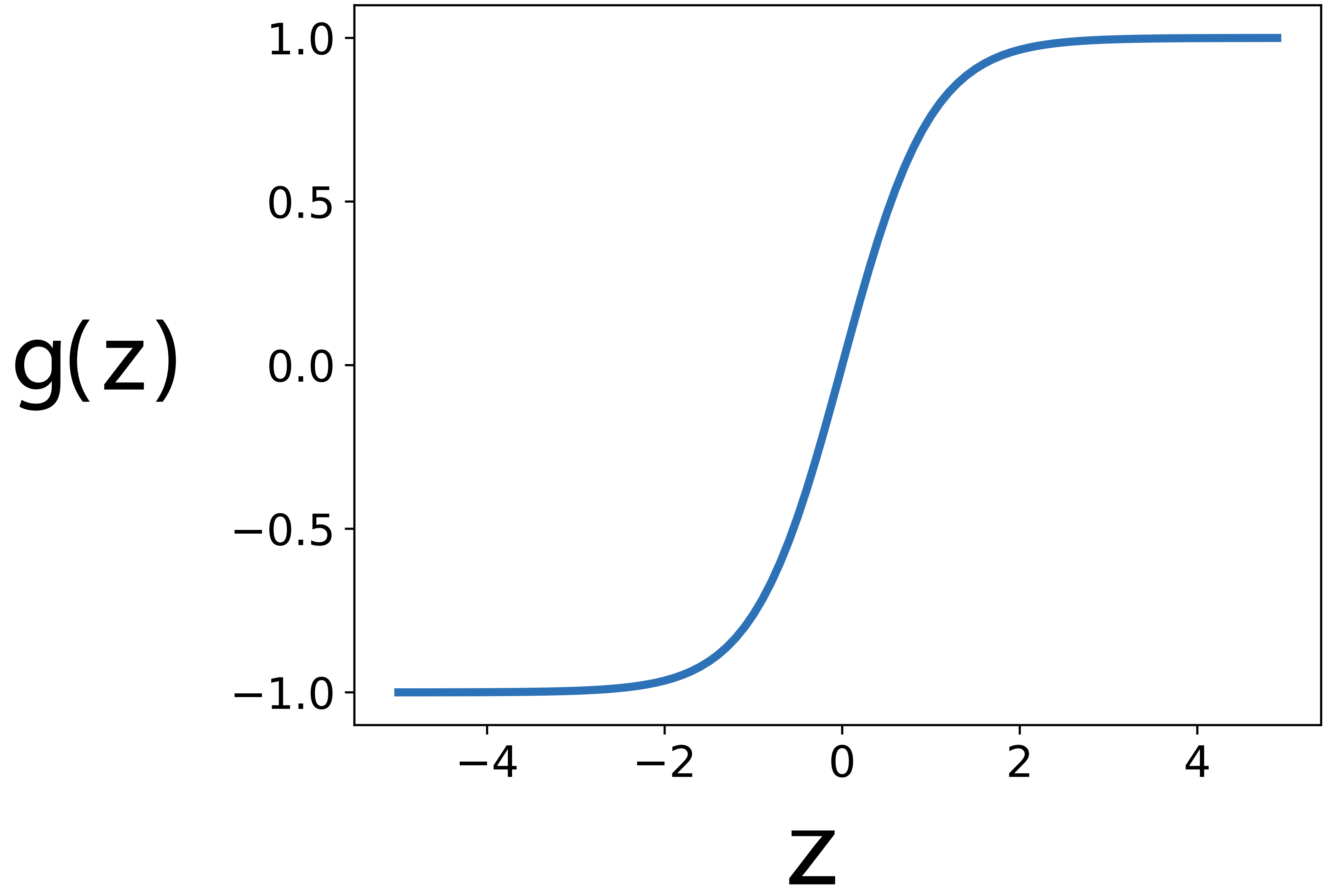
Input representation

Output representation



Tanh

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

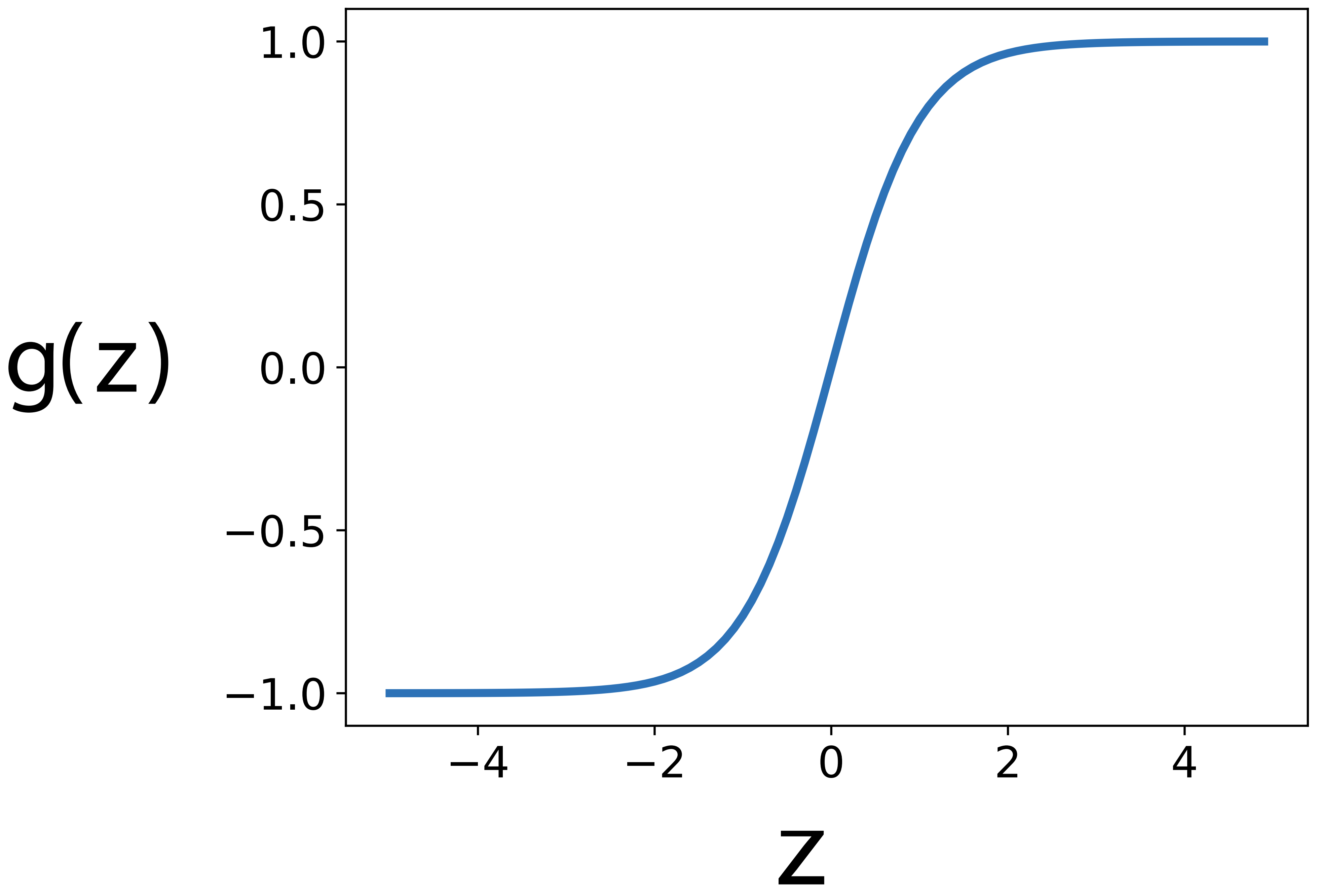


Computation in a neural net — nonlinearity

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

Tanh

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

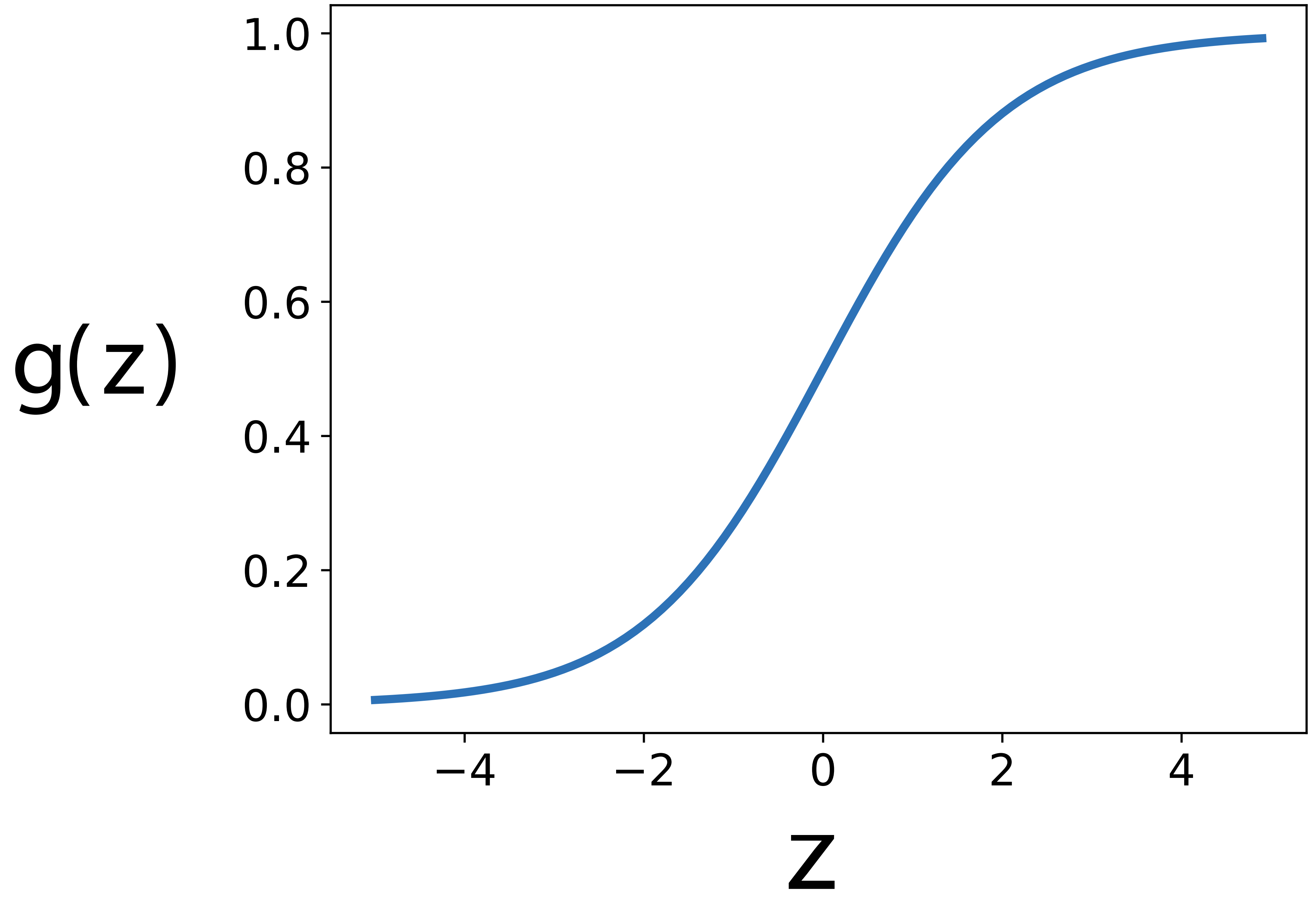


Computation in a neural net — nonlinearity

- Interpretation as firing rate of neuron
- Bounded between [0,1]
- Saturation for large +/- inputs
- Gradients go to zero
- Outputs centered at 0.5
  (poor conditioning)
- Not used in practice

Sigmoid

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

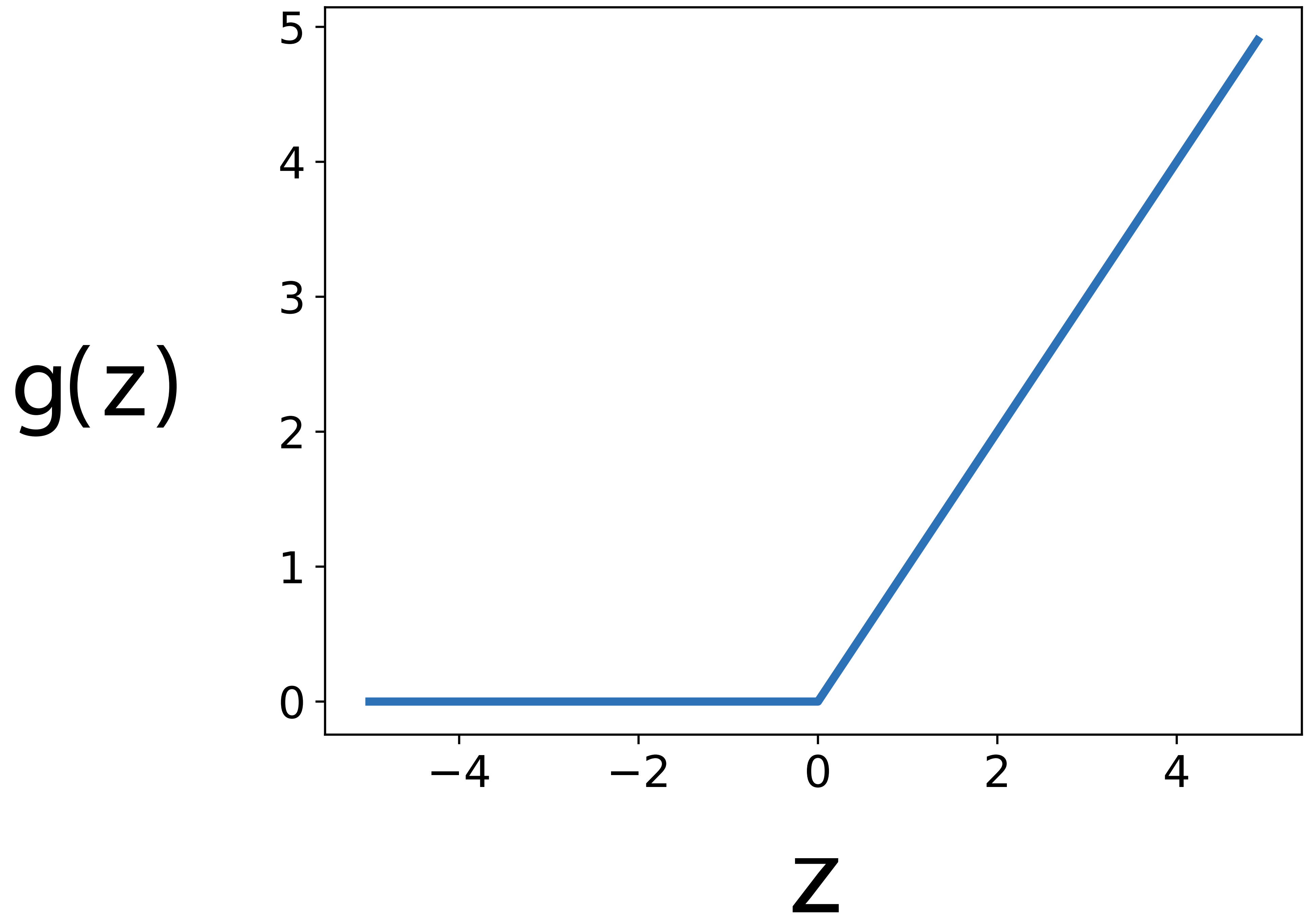


Computation in a neural net — nonlinearity

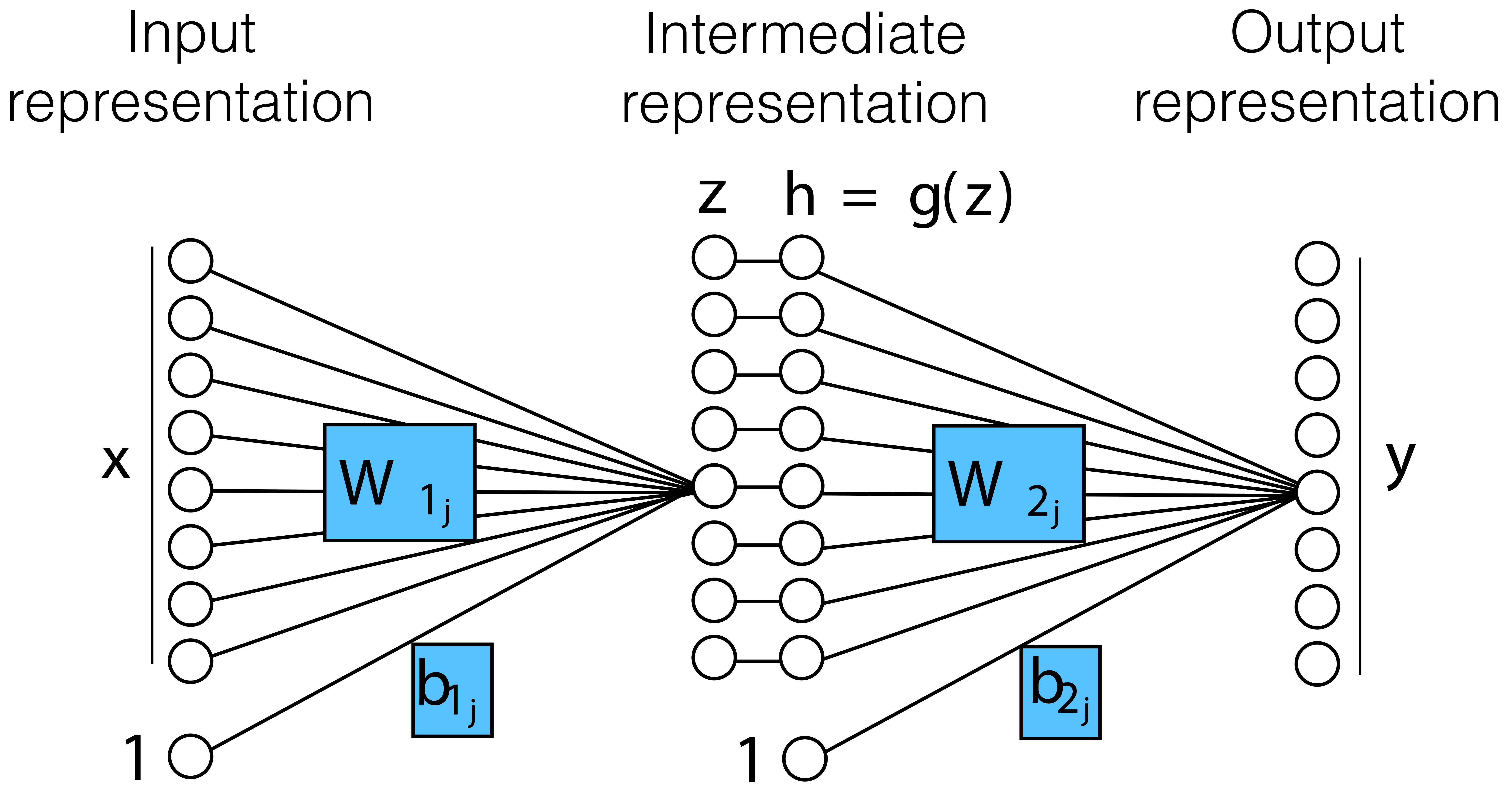
- Unbounded output (on positive side)
- Efficient to implement: $\frac{\partial g}{\partial z} = \begin{cases} 0, & \text{if} \quad z < 0 \\ 1, & \text{if} \quad z \geq 0 \end{cases}$
- Also seems to help convergence (see 6x speedup vs tanh in [Krizhevsky et al.])
- Drawback: if strongly in negative region, unit is dead forever (no gradient).
- Default choice: widely used in current models.

Rectified linear unit (ReLU)

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

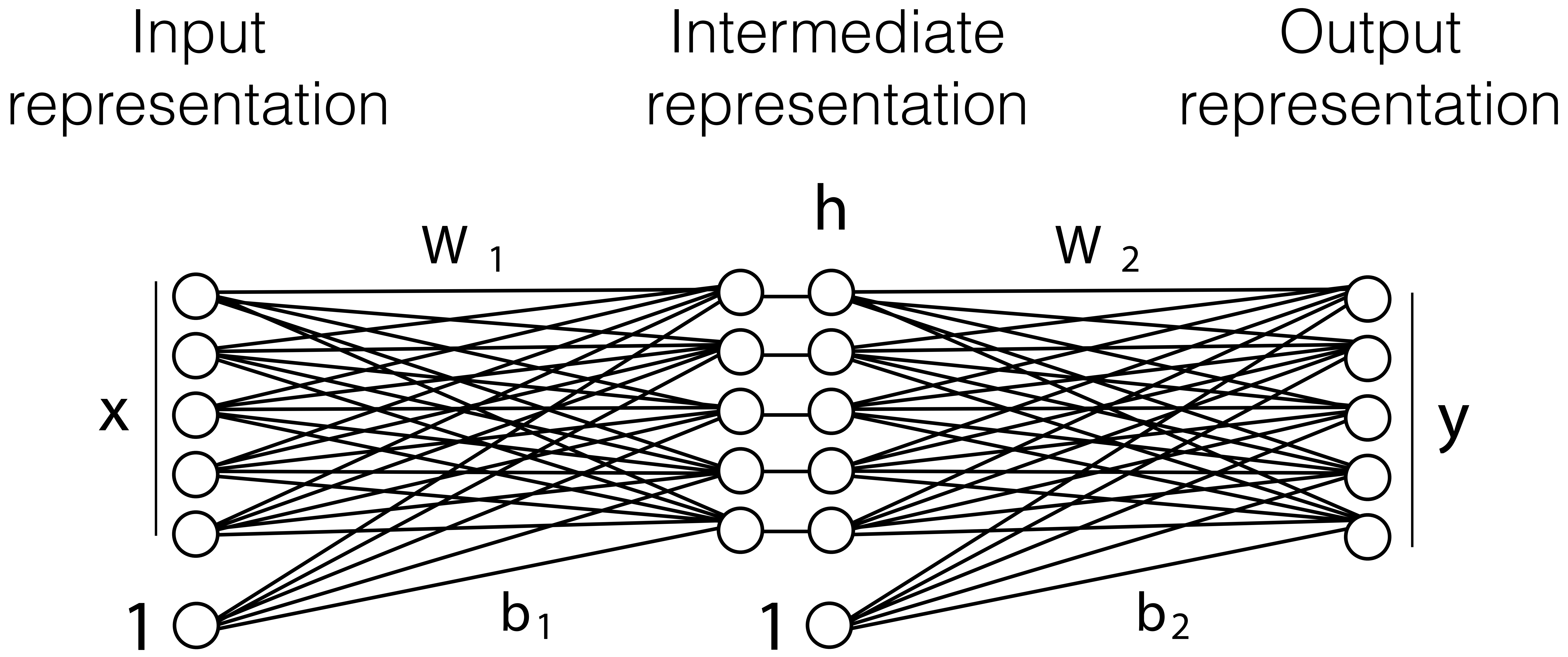


Stacking layers



z, h = **"hidden units"**

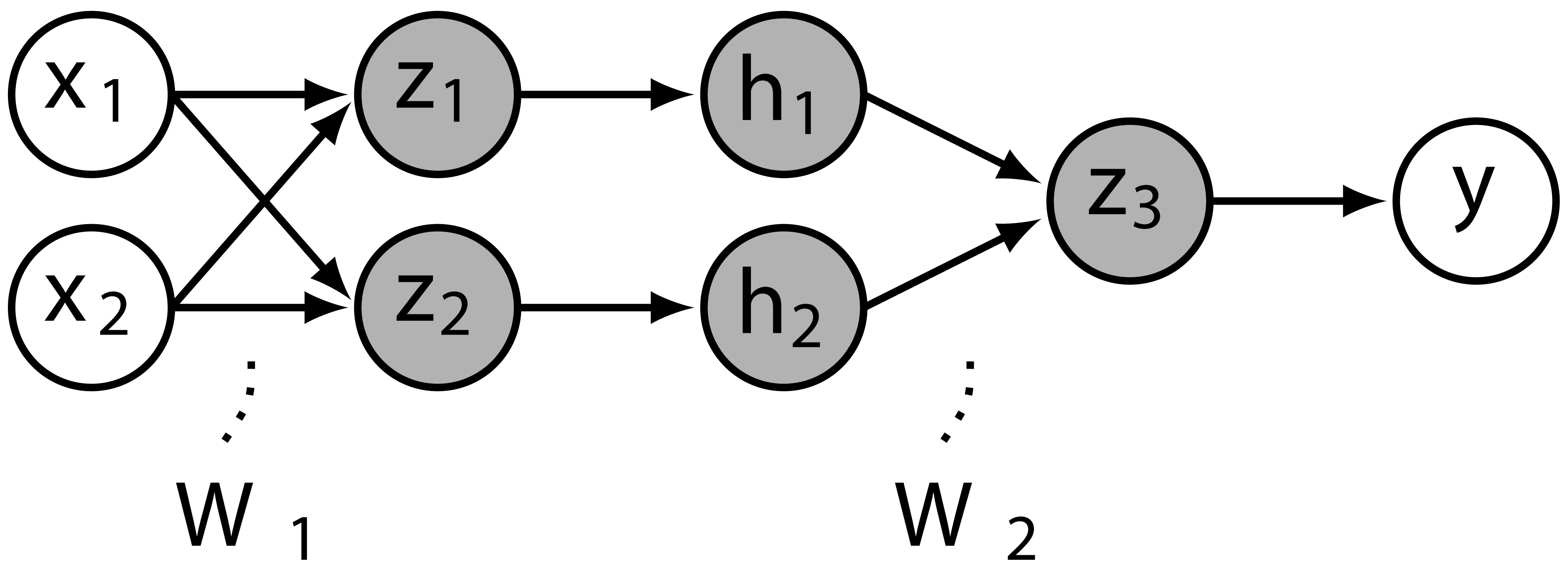
Stacking layers



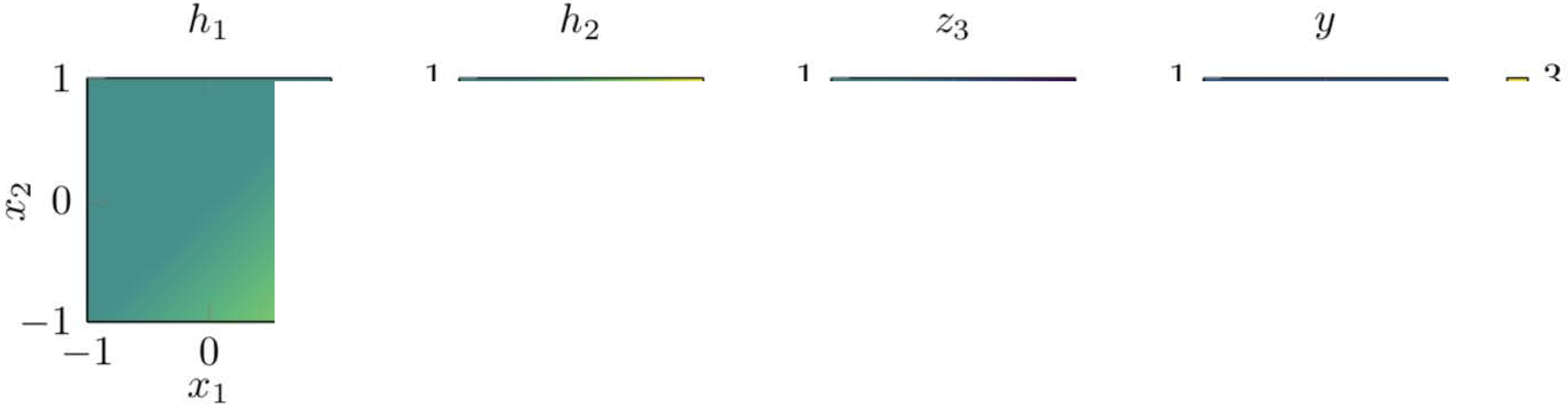
$$h = g(W_1 x + b_1) \qquad y = g(W_2 h + b_2)$$

$$\theta = \{W_1, \ldots, W_L, b_1, \ldots, b_L\}$$

Example: nonlinear classification with a deep net

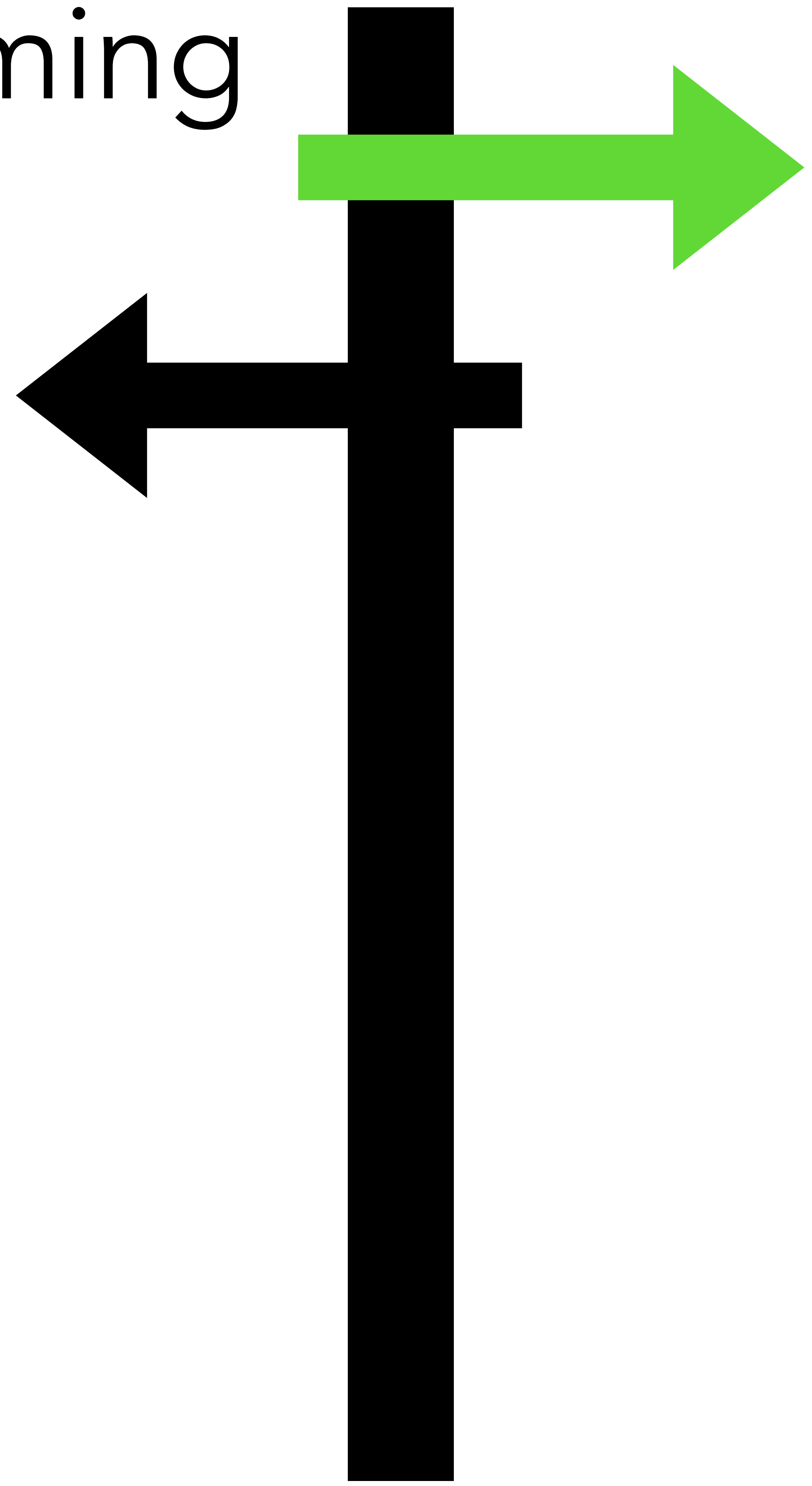


$$z = W_1 x + b_1$$
$$h = g(z)$$
$$z_3 = W_2 h + b_2$$
$$y = 1(z_3 > 0)$$



What we'll cover in this class

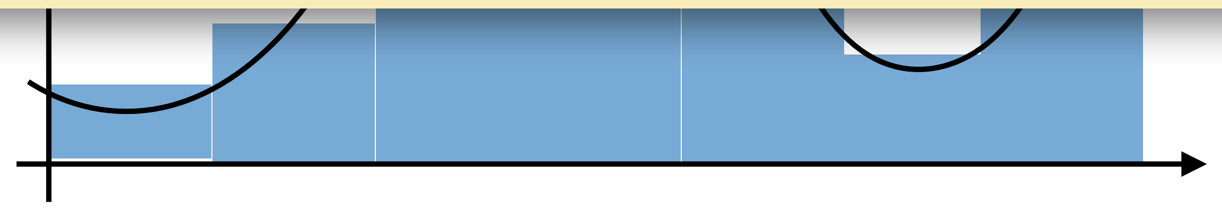
- Backprop and differentiable programming
- Why we can approximate



Representational power

- 1 layer? Linear decision surface.
- 2+ layers? In theory, can represent any function. Assuming non-trivial non-linearity.

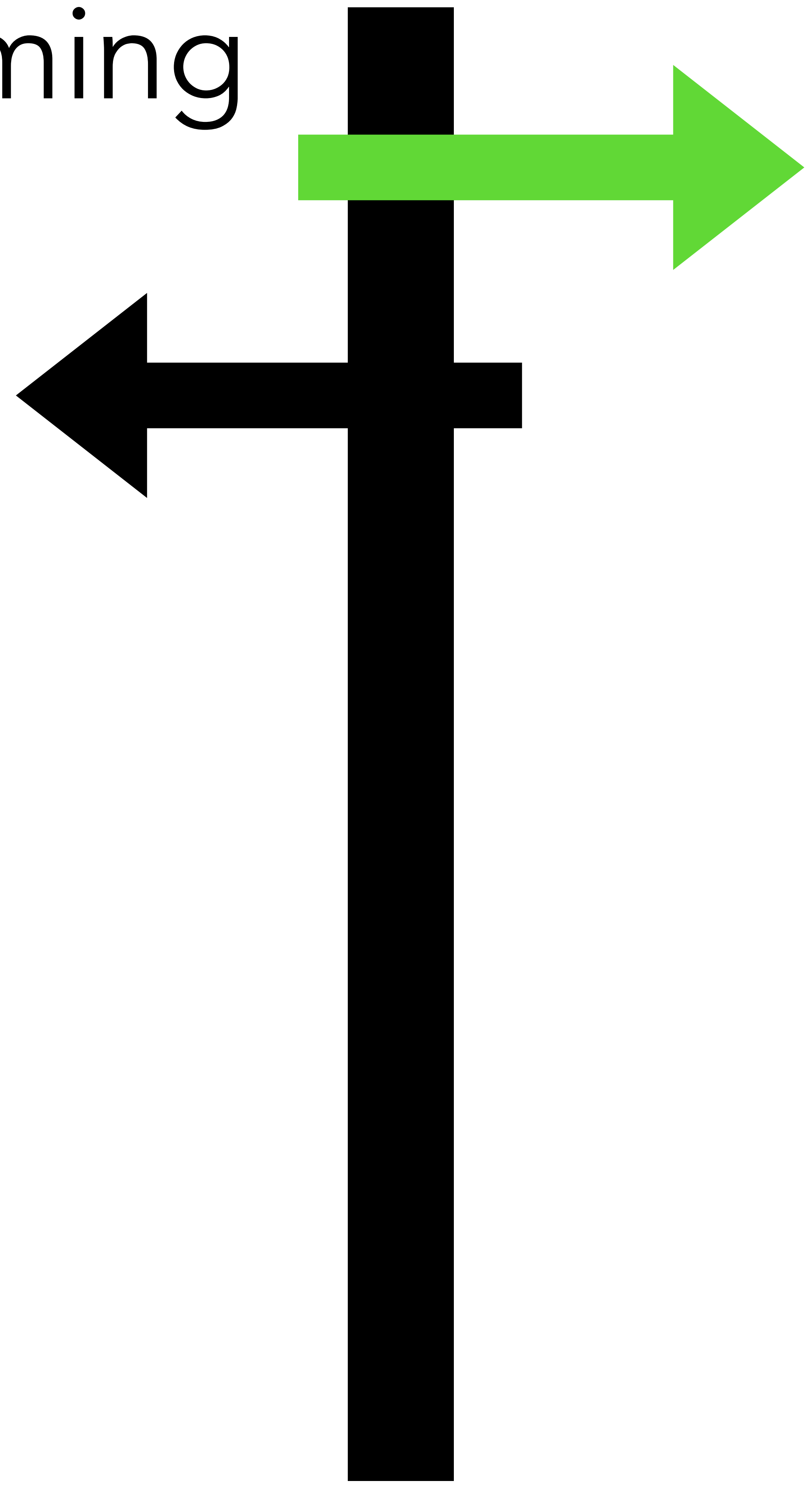
Lecture 3: Approximation theory



- But issue is efficiency: very wide two layers vs narrow deep model? In practice, more layers helps.

What we'll cover in this class

- Backprop and differentiable programming
- Why we can approximate
- Architectures



Deep nets

Linear

Non-linearity

Classify

Architectures

Lecture 4: CNNs

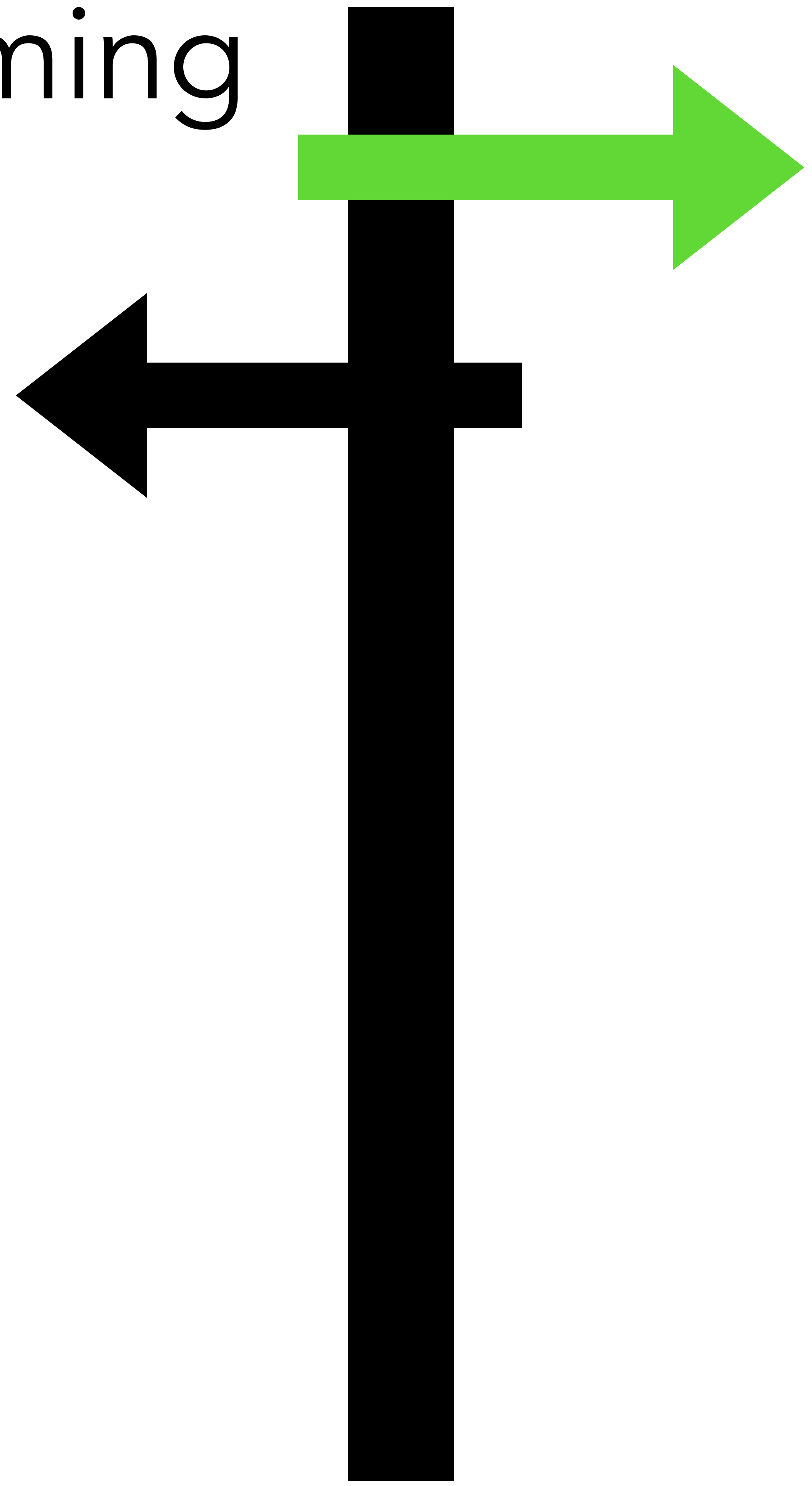
Lecture 5: GNNs

Lecture 9: Transformers

Lecture 11: RNNs

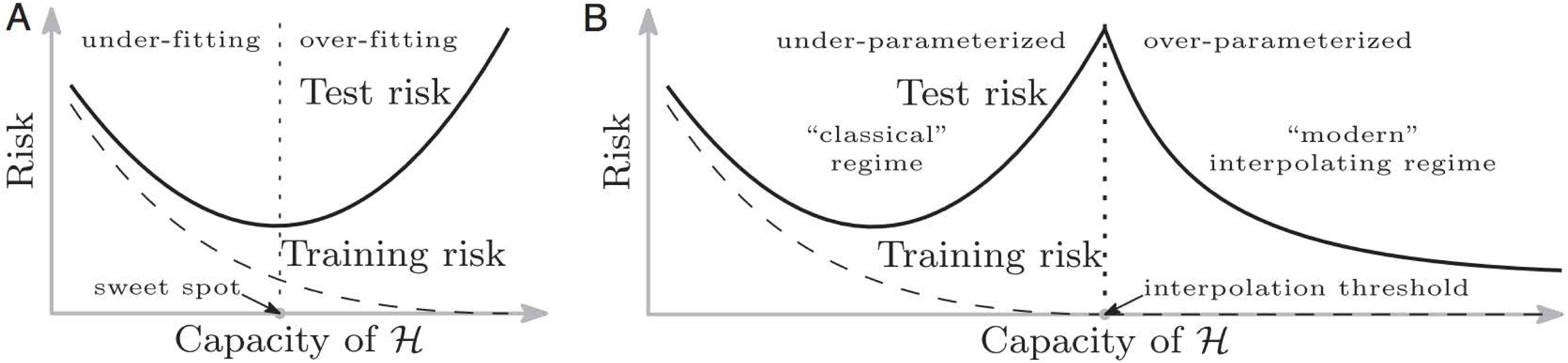
What we'll cover in this class

- Backprop and differentiable programming
- Why we can approximate
- Architectures
- When and why can we generalize



Why do deep nets generalize?

- Deep nets have so many parameters they could just act like look up tables, regurgitating their training data
- Instead, they learn rules that generalize
- Defies classical theory!



[Double-descent: Belkin, Hsu, Ma, Mandal, PNAS 2019]

The simplicity hypothesis

Classical theory:

Lecture 7: Generalization theory

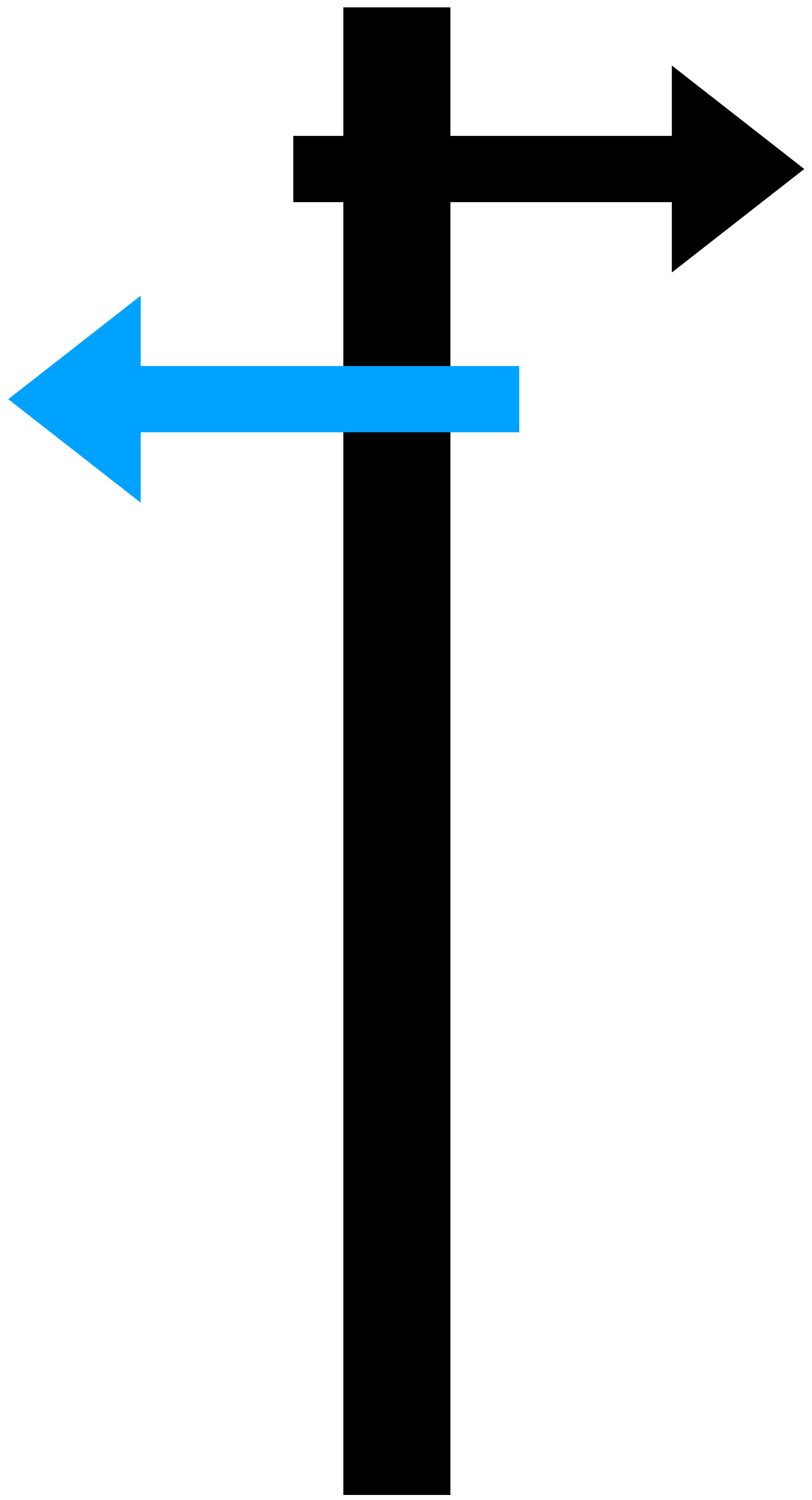
Lecture 17: OOD generalization

Emerging theory:

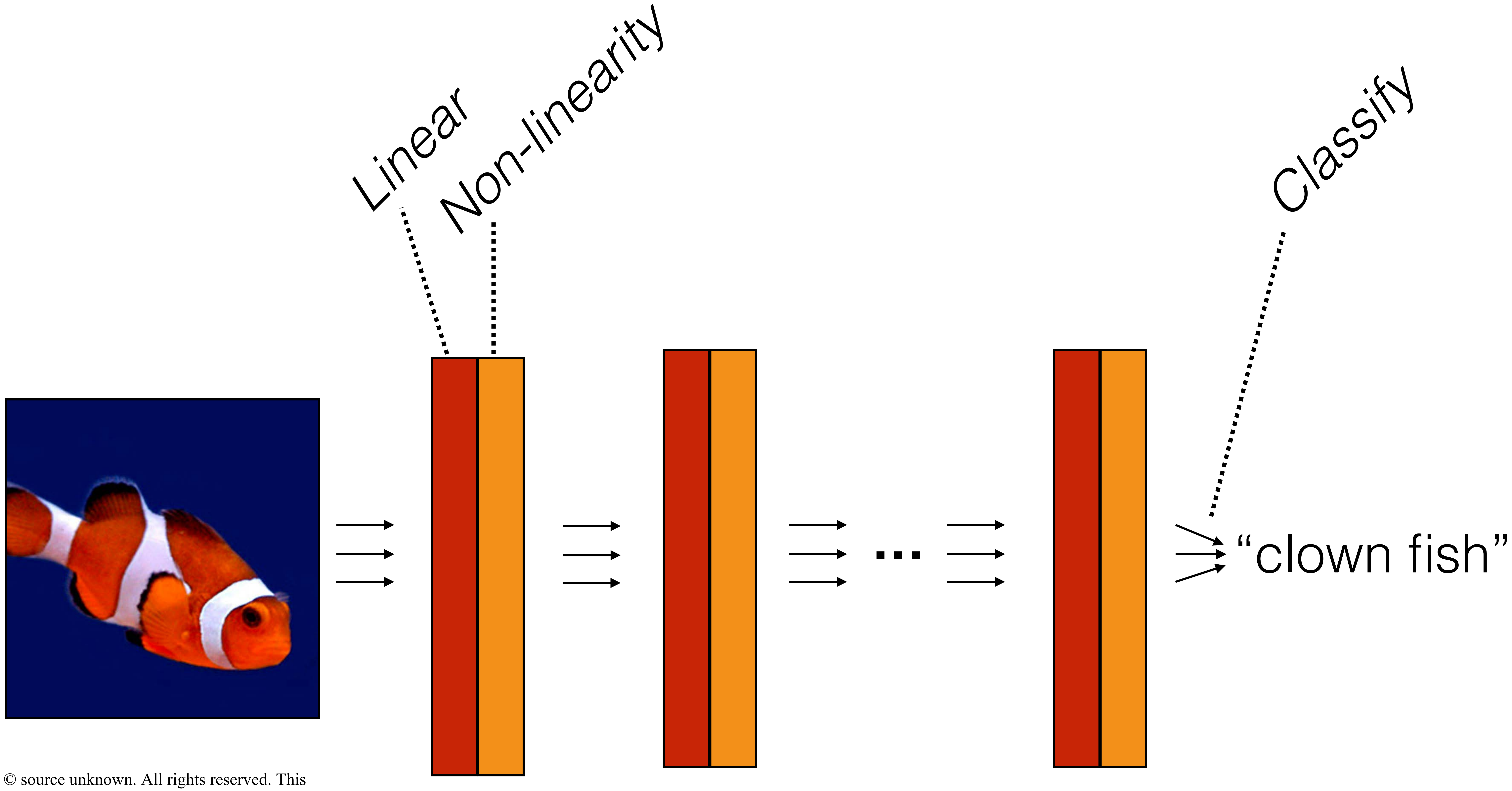
deep nets learn *simple* functions that generalize

What we expect you to have seen before

- Gradient descent
- MLPs, Nonlinearities (ReLu)
- Softmax, cross-entropy loss



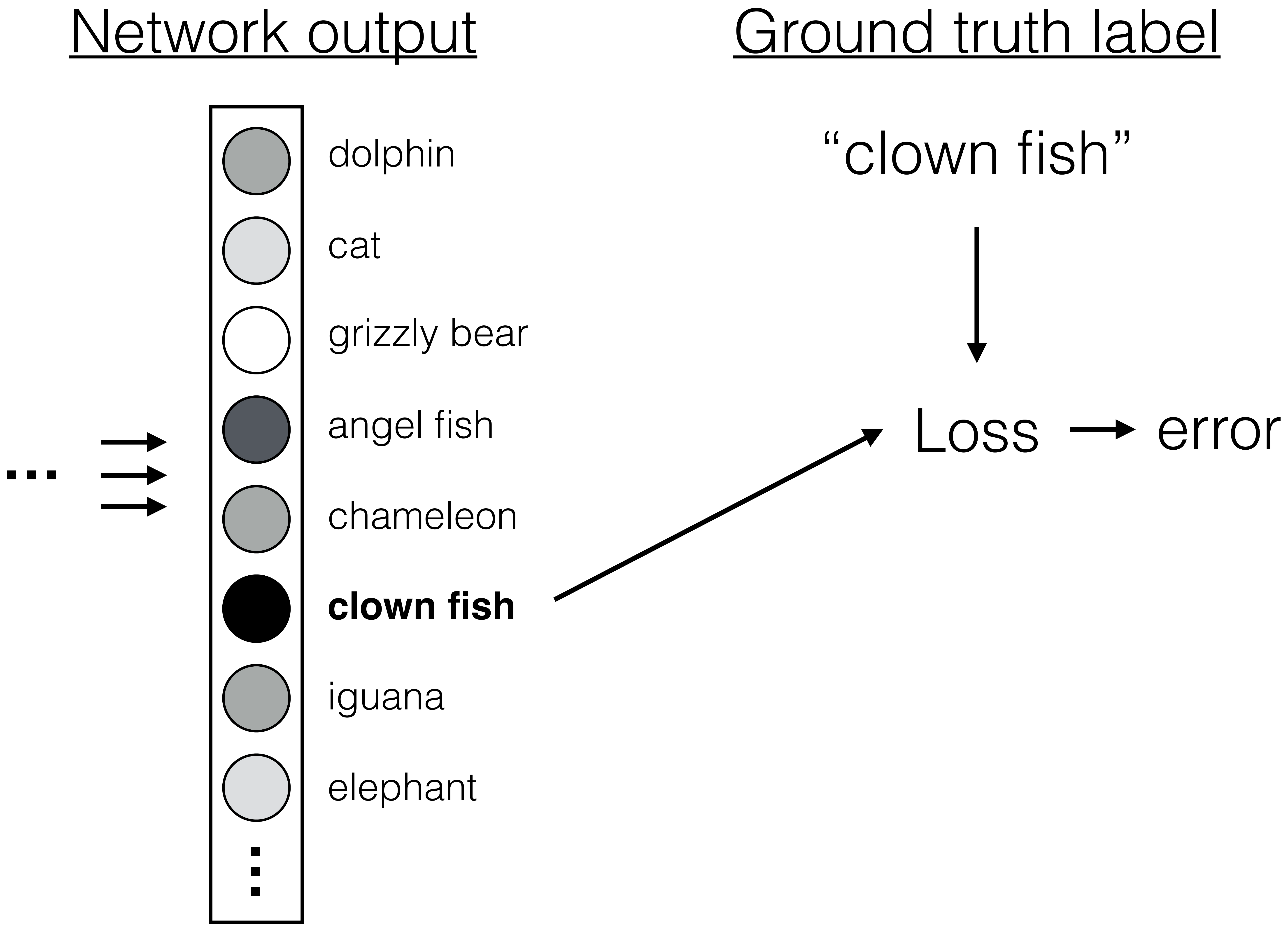
Deep nets



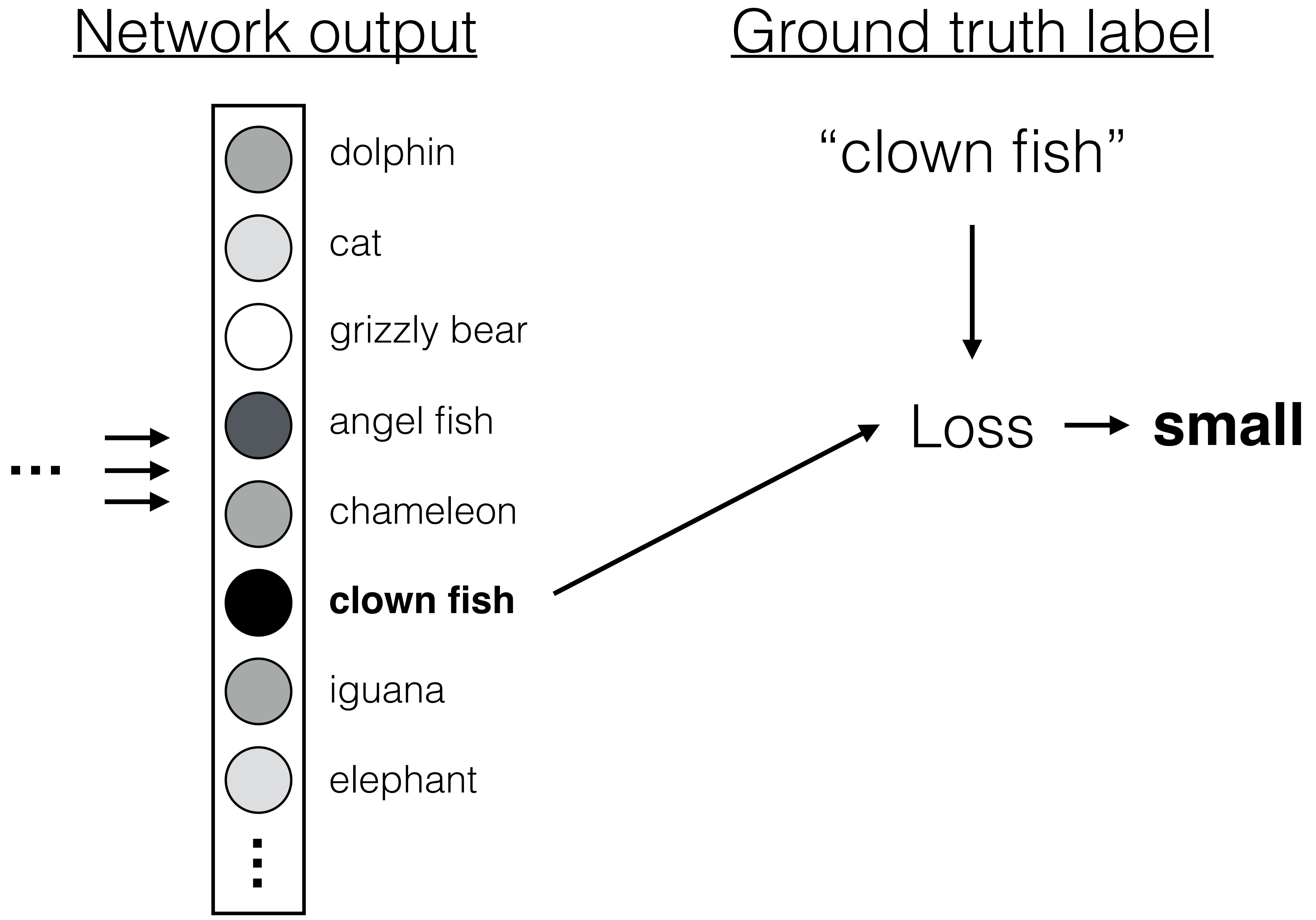
$$f(x) = f_L(f_{L-1}(\ldots f_2(f_1(x))))$$

Classifier layer

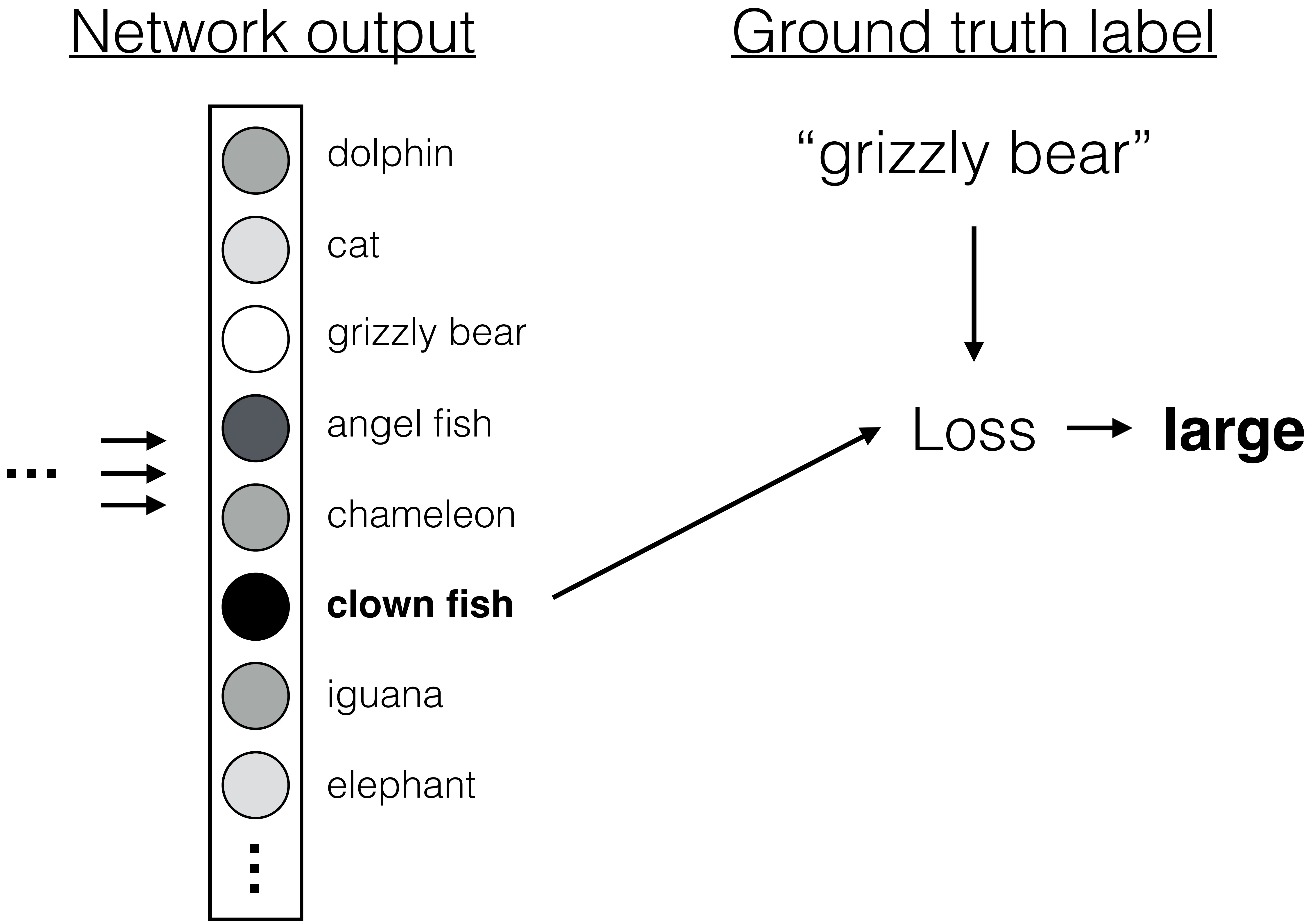
Loss function



Loss function

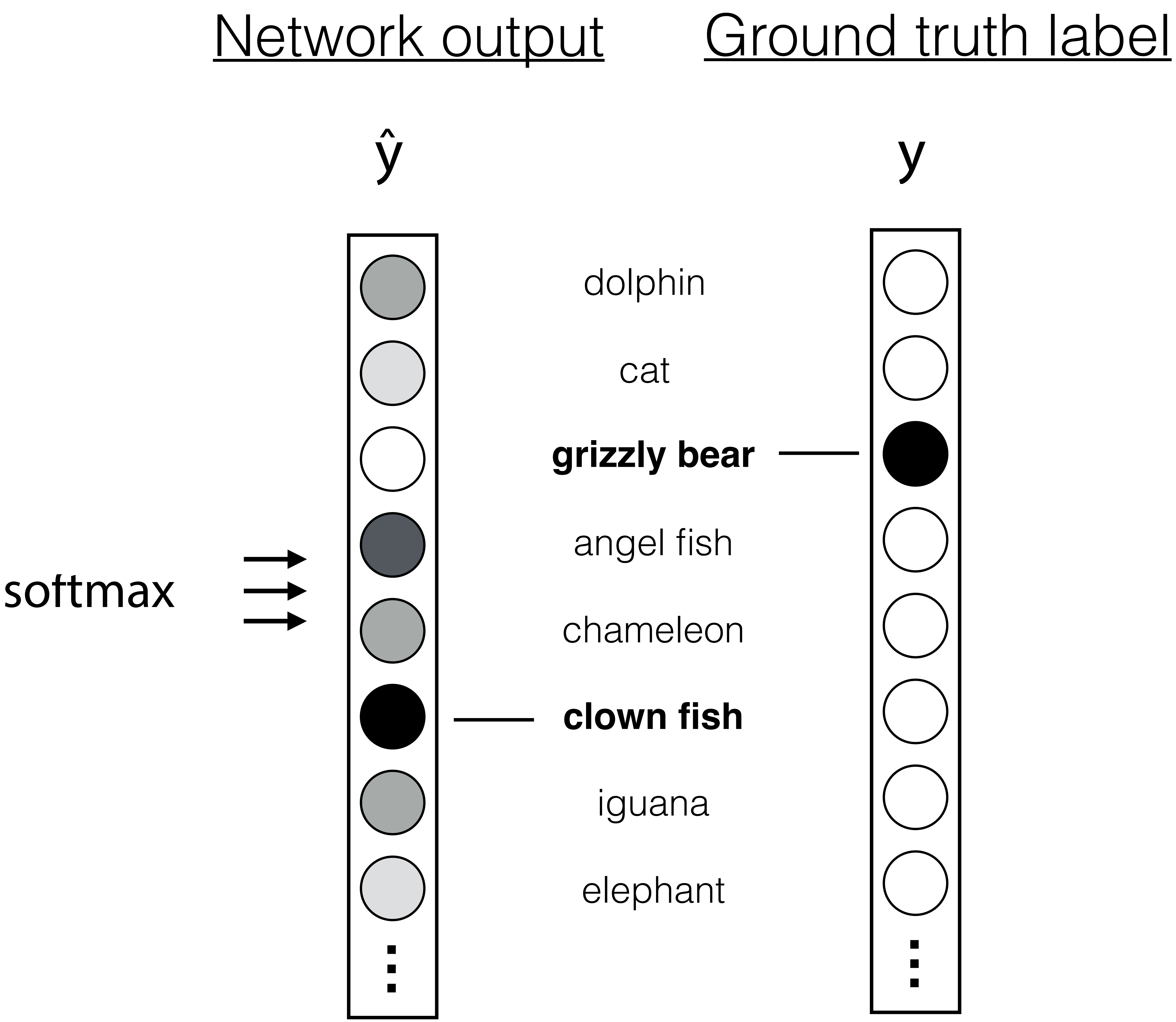


Loss function



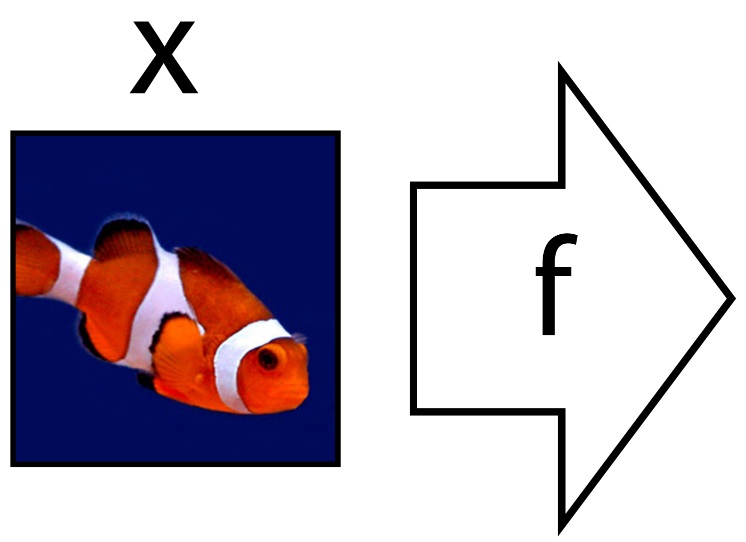
Probability of the observed data under the model

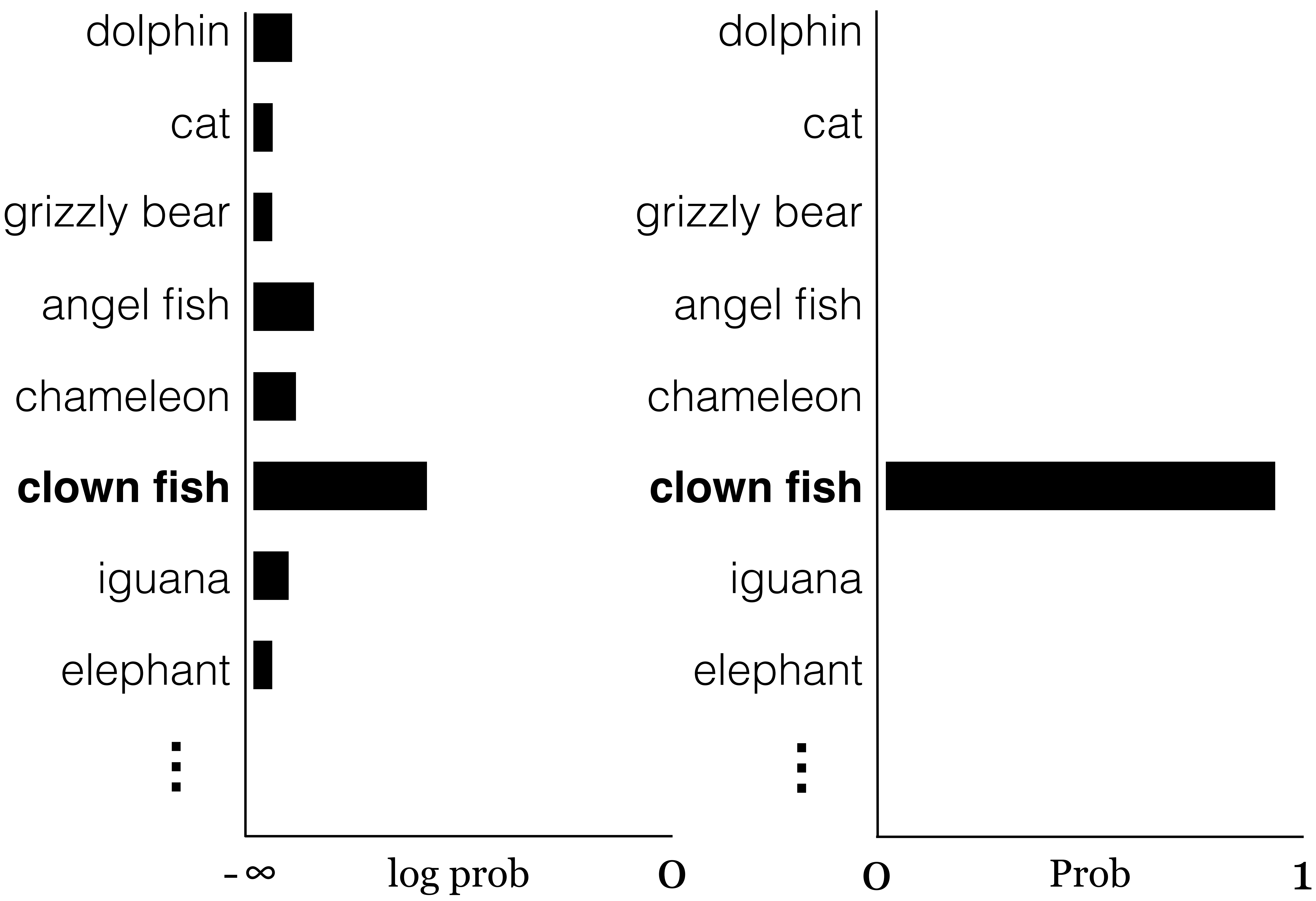
$$H(y, \hat{y}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$$



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

$f_\theta : X \qquad R^K$





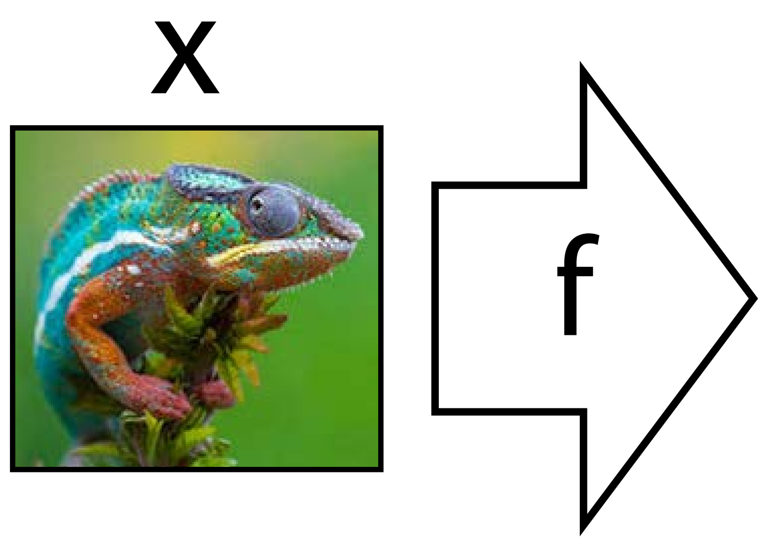

Prediction log ŷ
f θ : X R K
Ground truth label y
Score – L (ŷ, y)
– H (y, ŷ) = Σ_{k=1}^{K} y_k log ŷ_k
x
f
dolphin
cat
grizzly bear
angel fish
chameleon
clown fish
iguana
elephant
⋮
⊙
dolphin
cat
grizzly bear
angel fish
chameleon
clown fish
iguana
elephant
⋮
How much better you could have done
- ∞
log prob
0
0
Prob
1
- ∞
- Loss
0

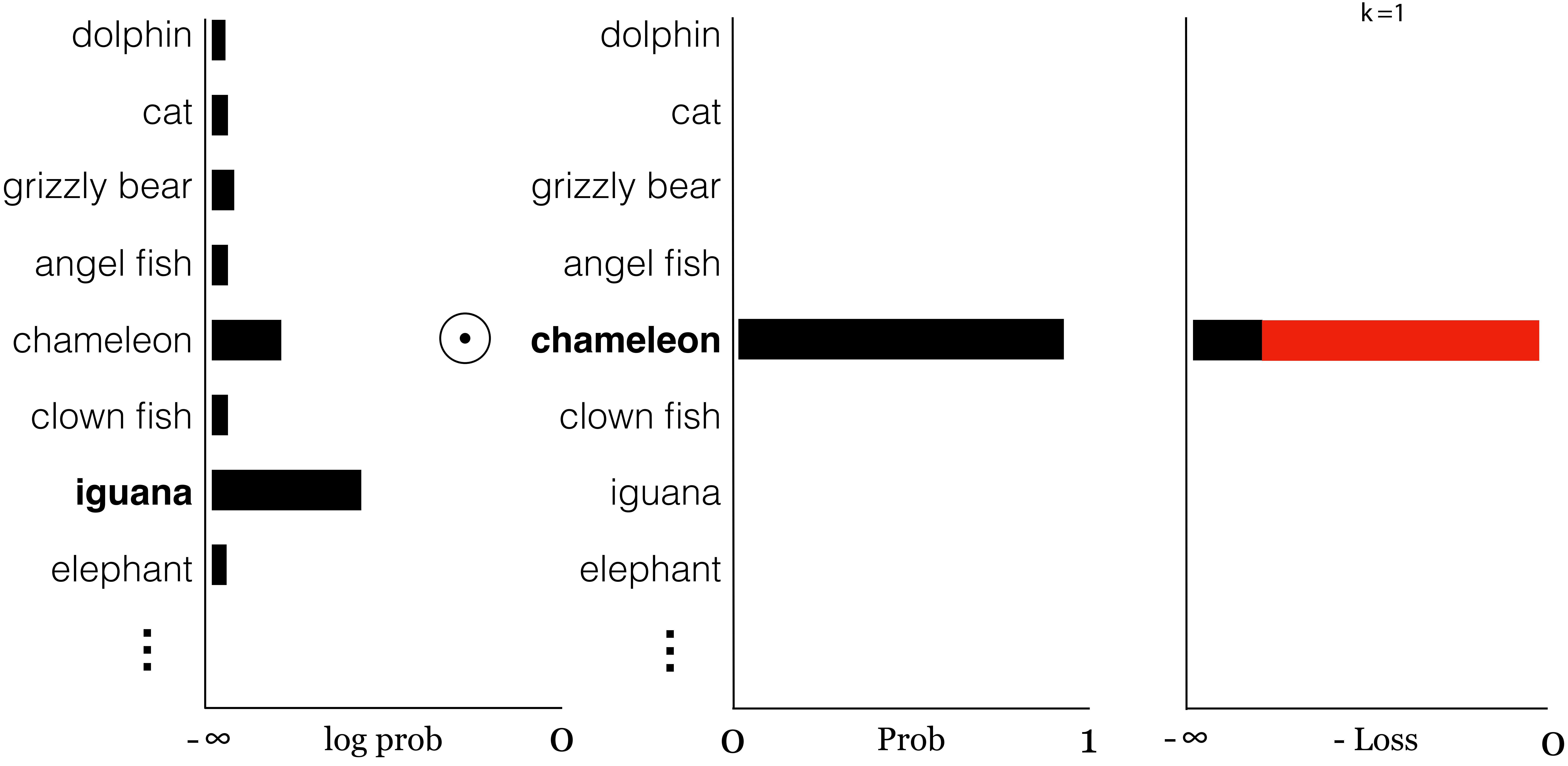

Prediction log ŷ
$f_\theta : X \quad R^K$
Ground truth label y
Score −L(ŷ, y)
$-H(y, \hat{y}) = \sum_{k=1}^{K} y_k \log \hat{y}_k$
x
f
dolphin
cat
grizzly bear
angel fish
chameleon
clown fish
iguana
elephant
⋮
- ∞
log prob
0
0
Prob
1
- ∞
- Loss
0

Prediction $\log \hat{y}$ **Ground truth label** y **Score** $-L(\hat{y}, y)$

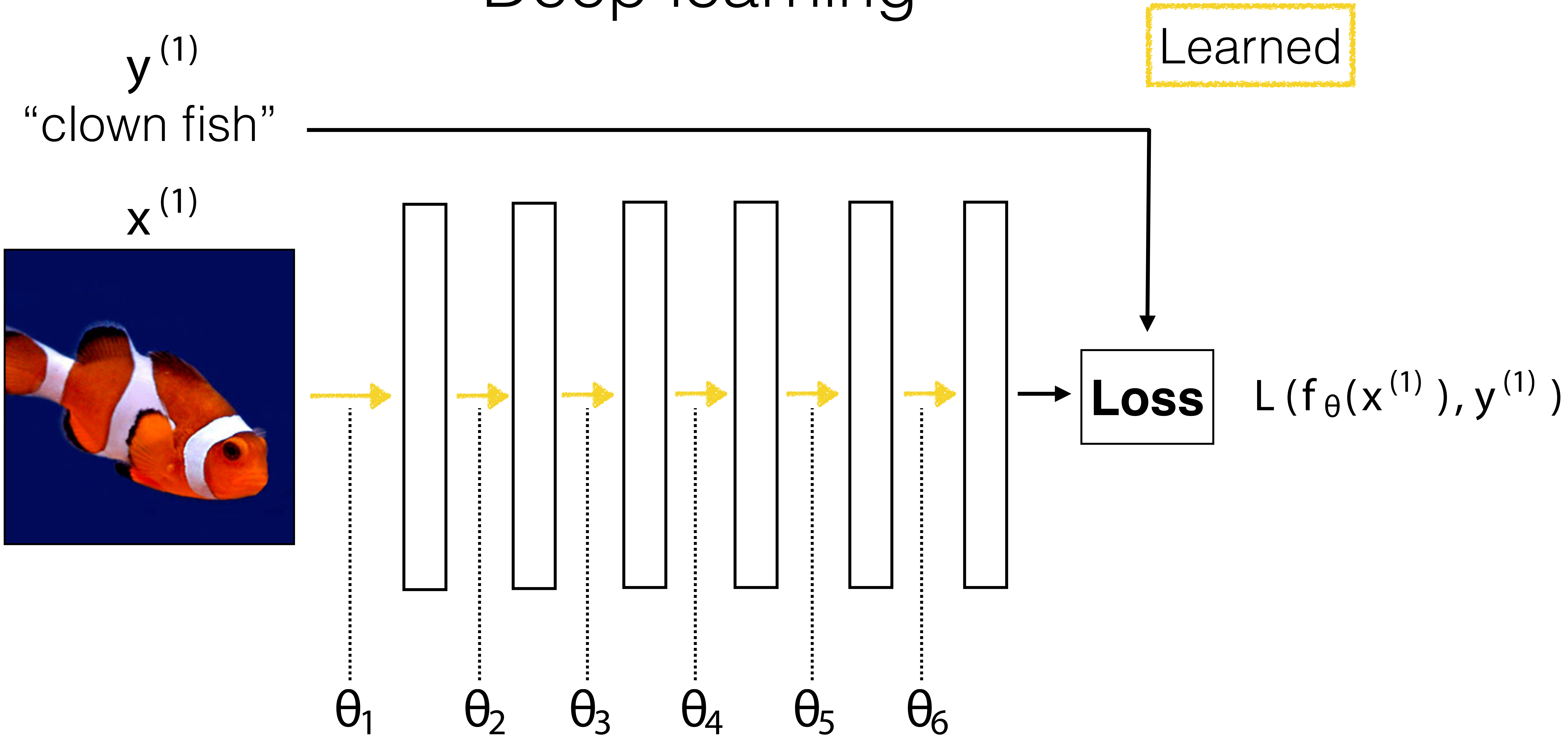
$$f_\theta : X \qquad R^K$$

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



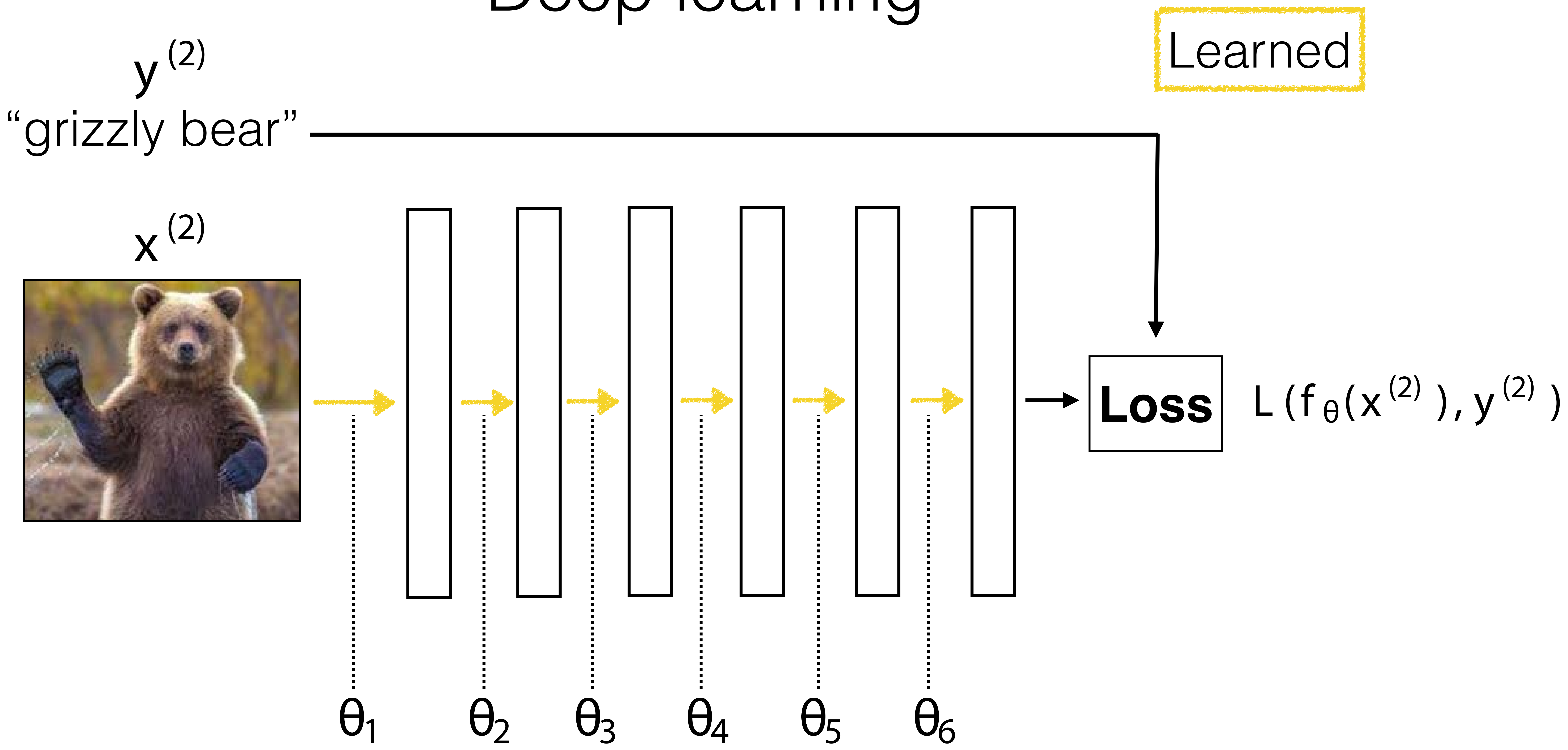


Deep learning



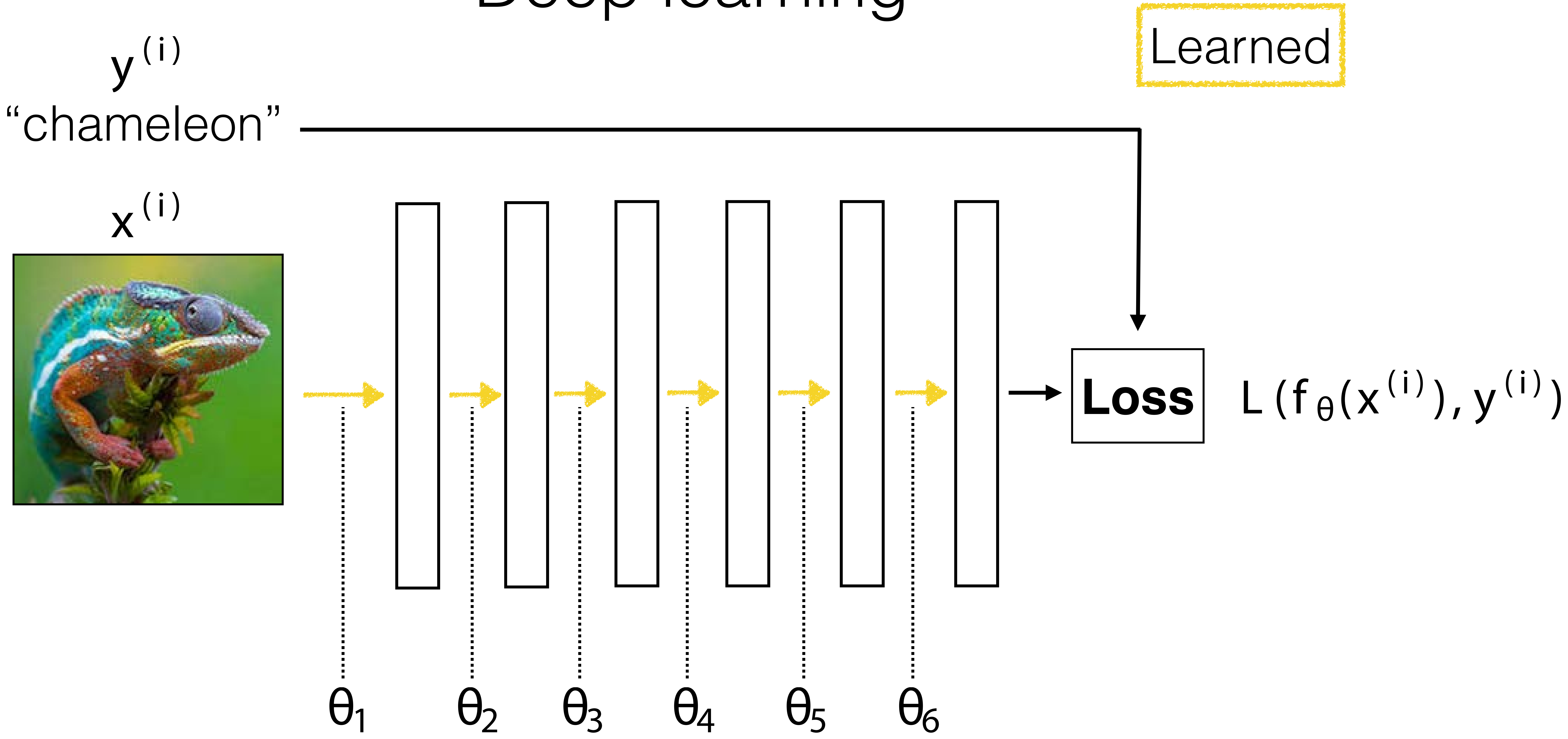
$$\theta = \arg\min_{\theta} \sum_{i=1}^{N} L(f_\theta(x^{(i)}), y^{(i)})$$

Deep learning



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

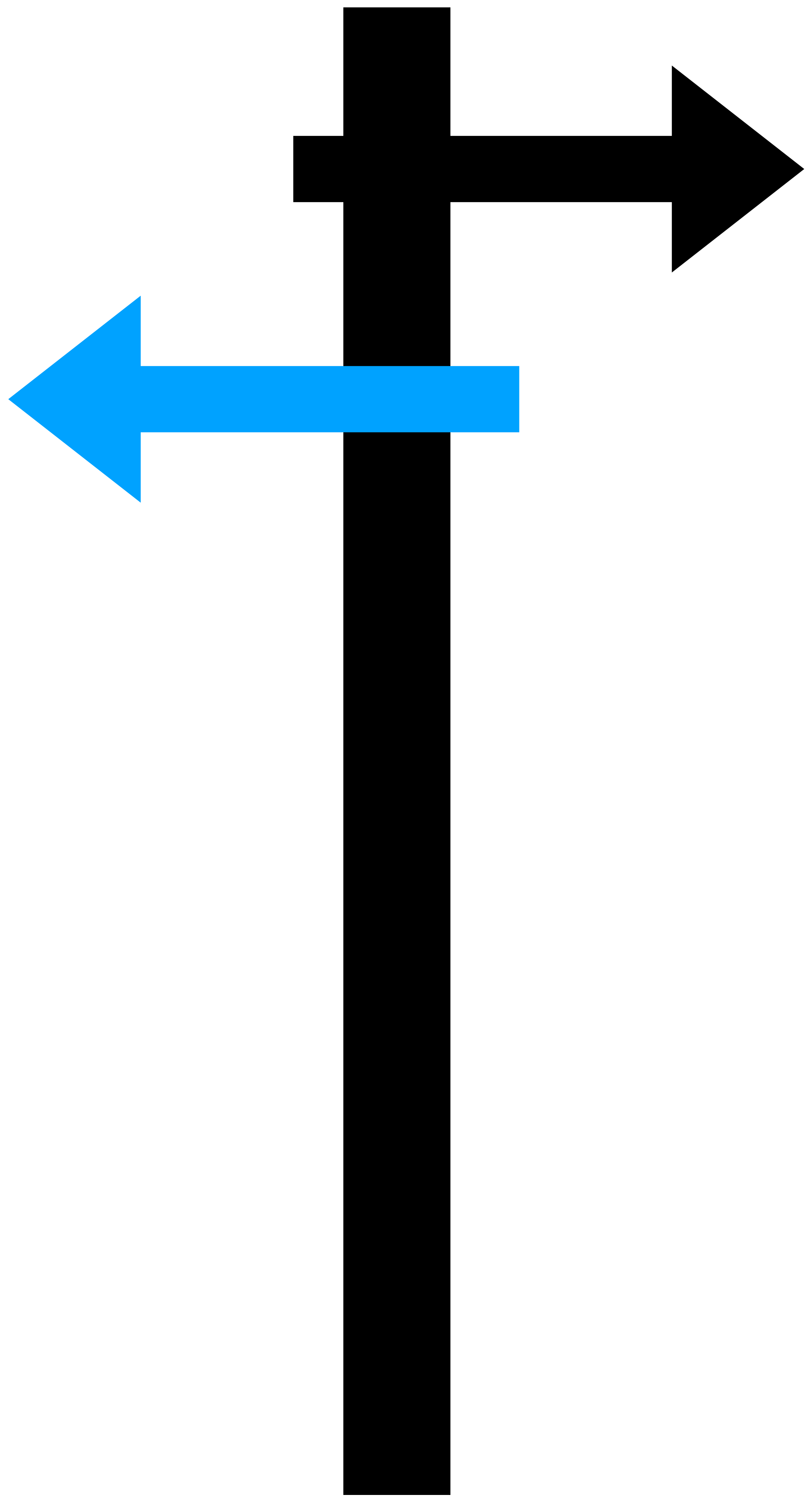
Deep learning



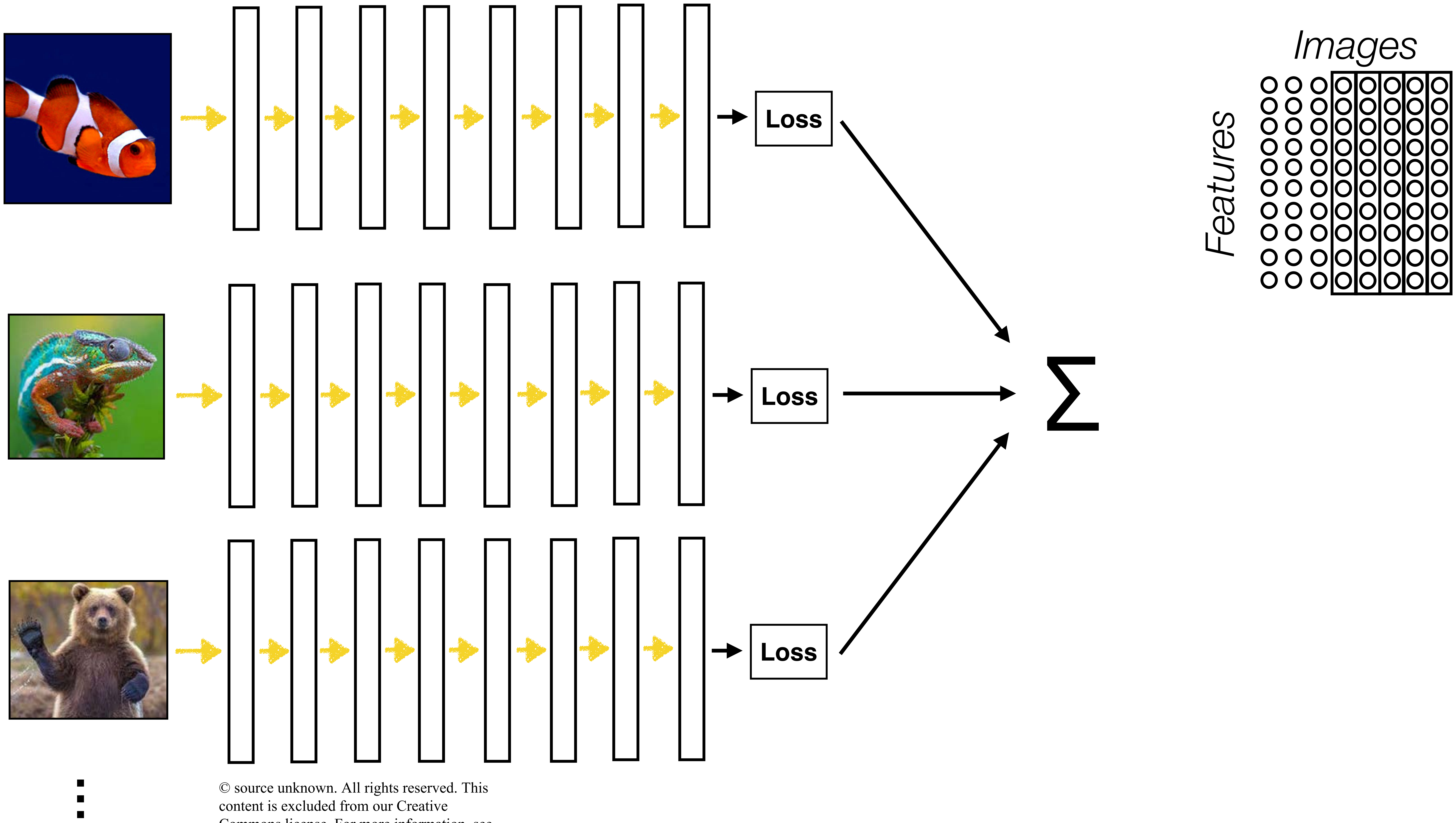
$$\theta = \arg\min_{\theta} \sum_{i=1}^{N} L(f_\theta(x^{(i)}), y^{(i)})$$

What we expect you to have seen before

- Gradient descent
- MLPs, Nonlinearities (ReLu)
- Softmax, cross-entropy loss
- Parallel processing, tensors

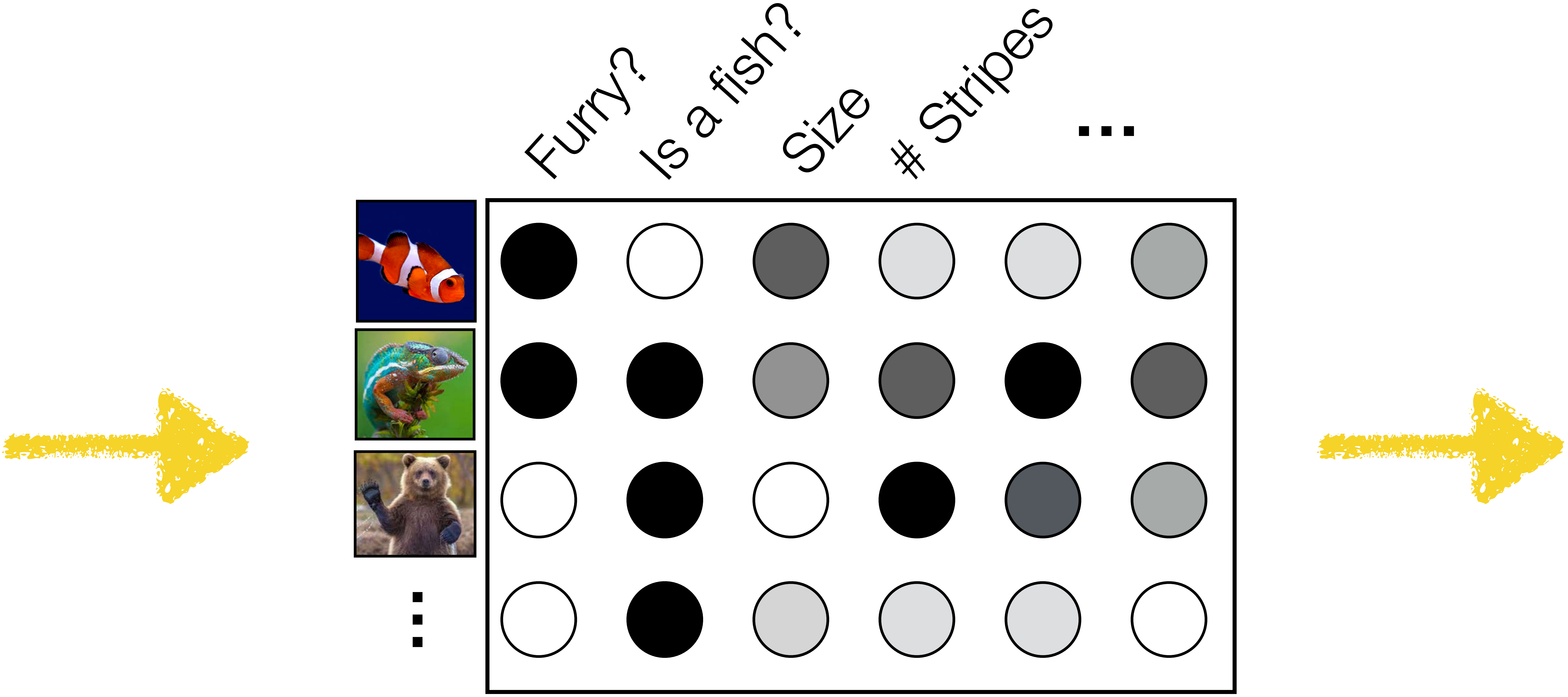


Batch (parallel) processing



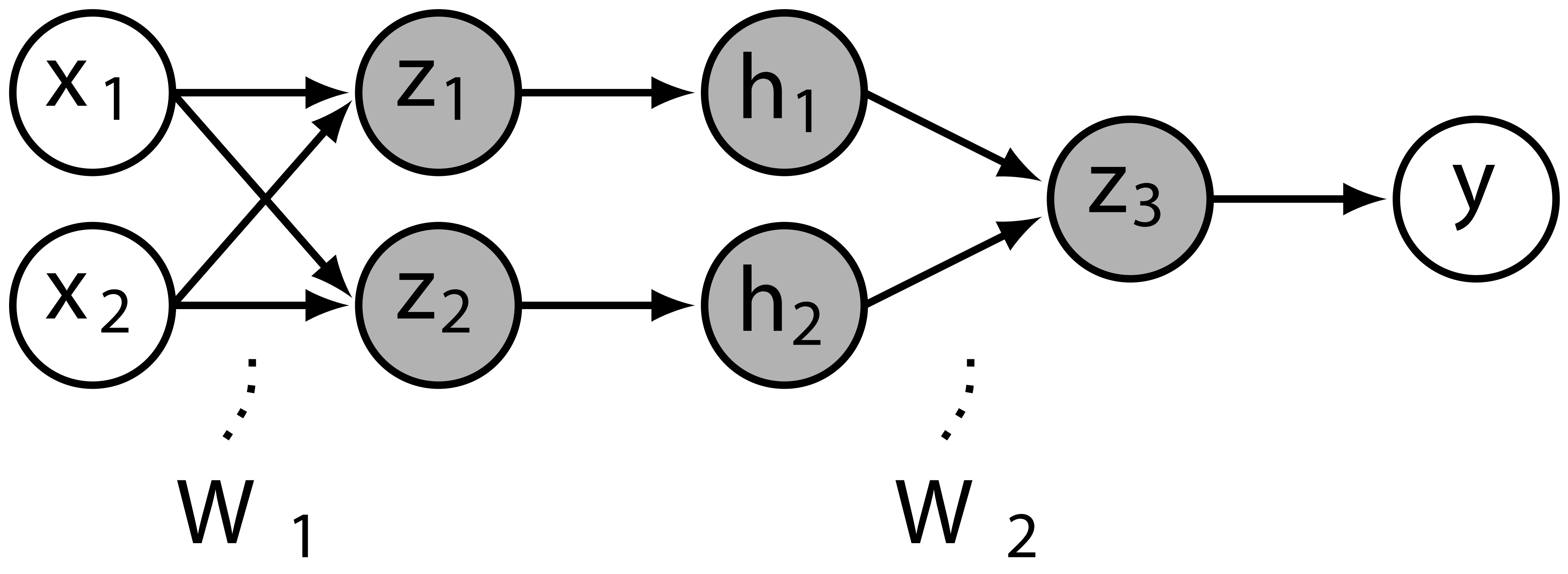
Tensors

(multi-dimensional arrays)



Each layer is a representation of the data

Everything is a tensor



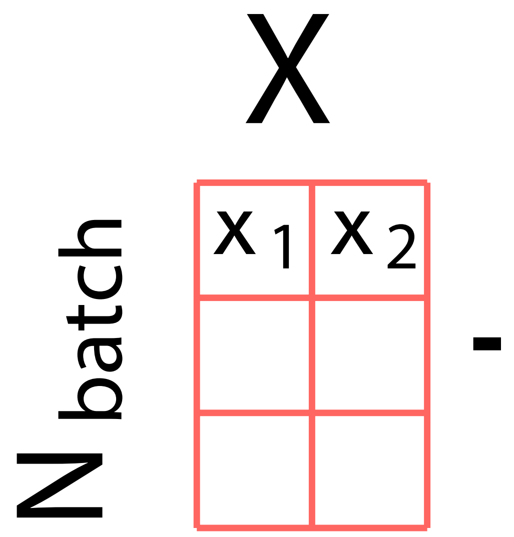
$$z = W_1 x + b_1$$

$$h = g(z)$$

$$z_3 = W_2 h + b_2$$

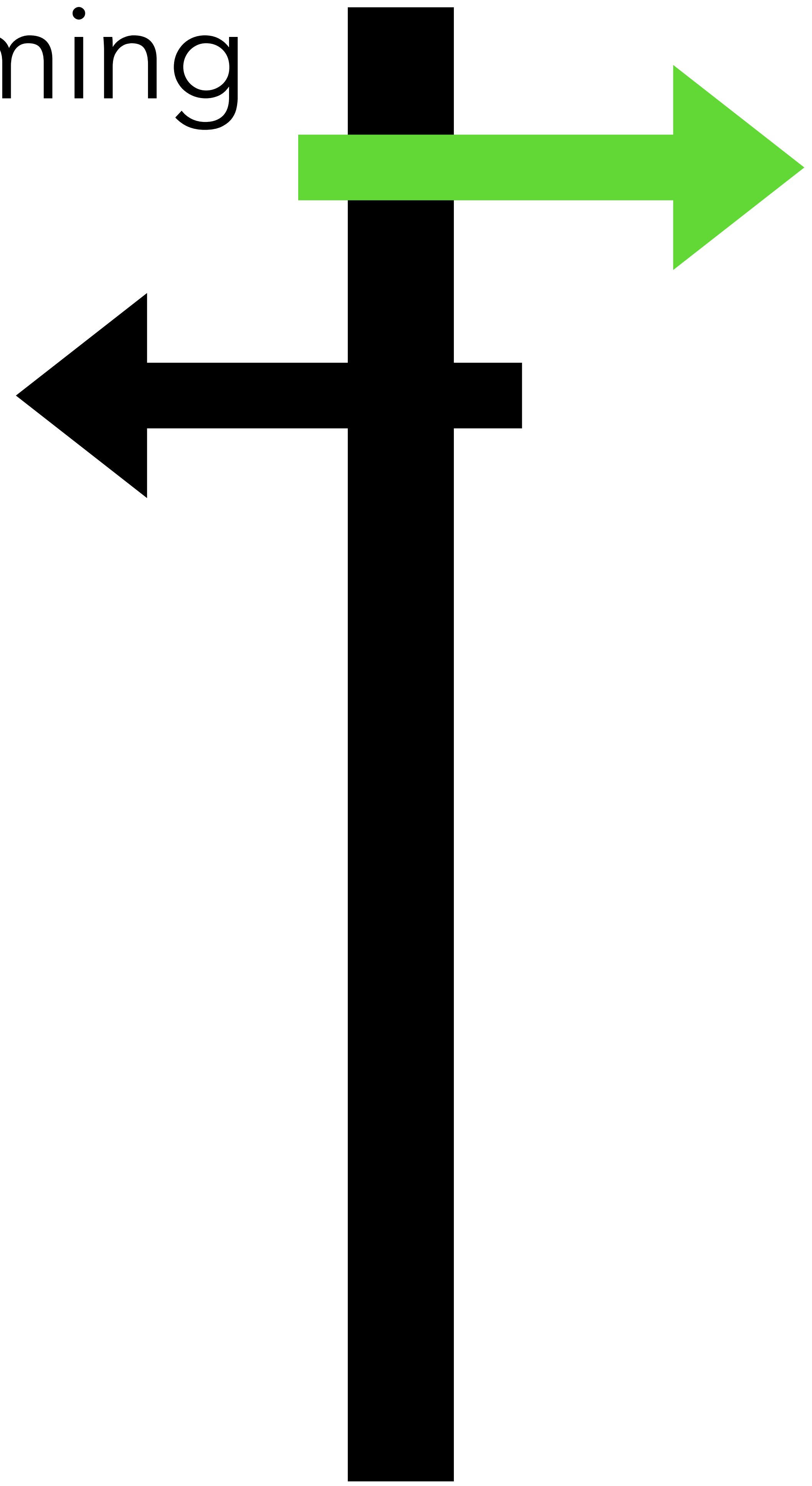
$$y = 1(z_3 > 0)$$

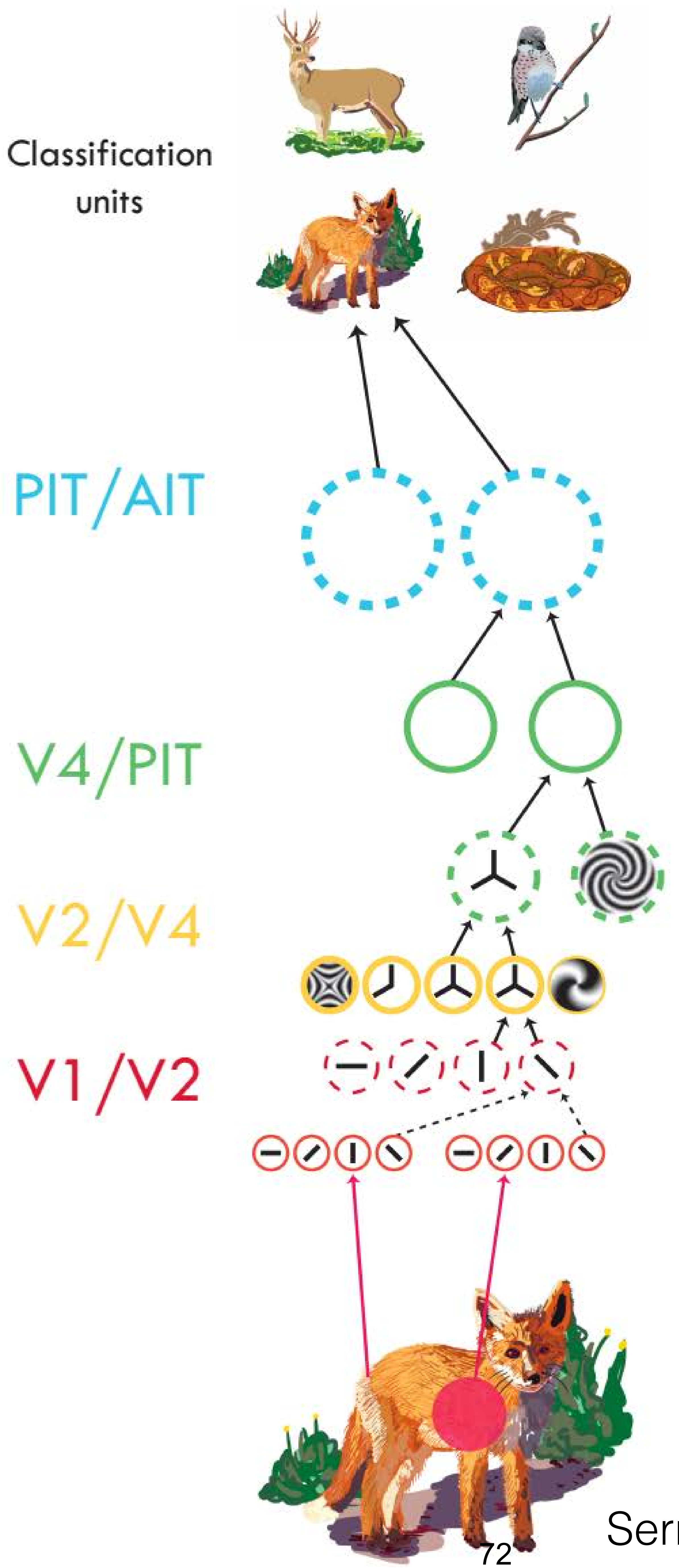
Tensor processing with batch size = 3:



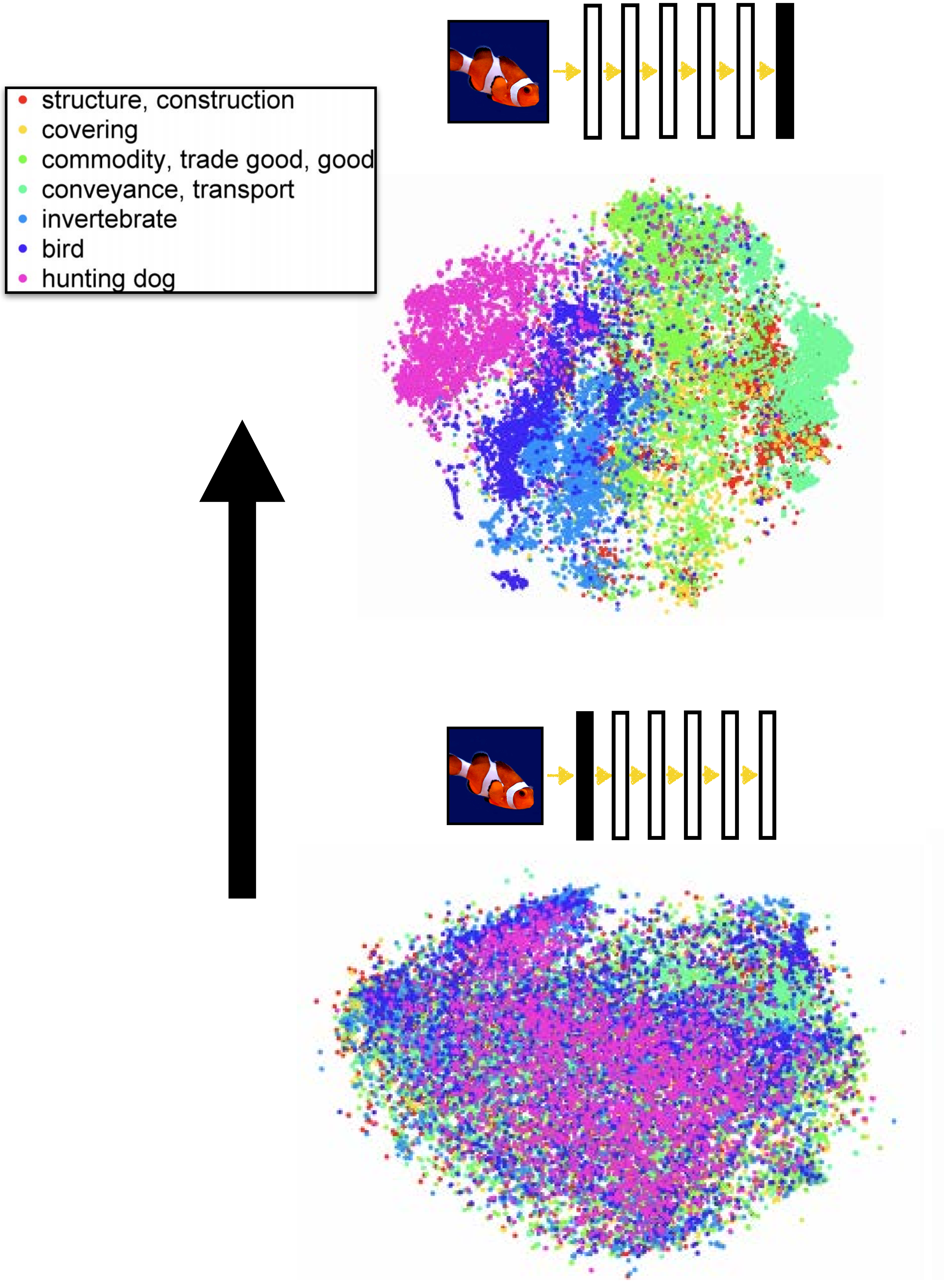
What we'll cover in this class

- Backprop and differentiable programming
- Why we can approximate
- Architectures
- When and why can we generalize
- How deep networks represent data

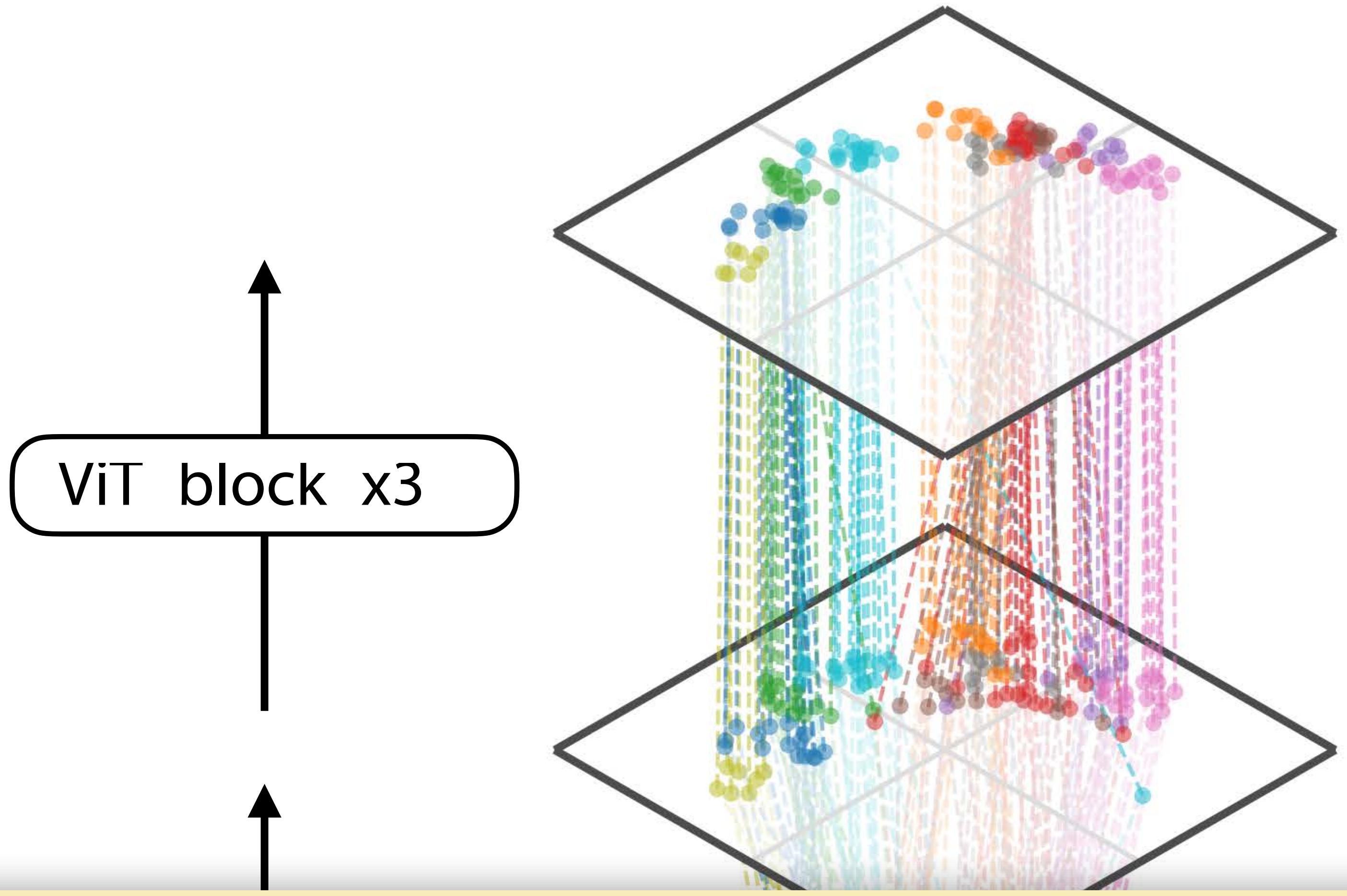




Serre, 2014



Donahue, 2013

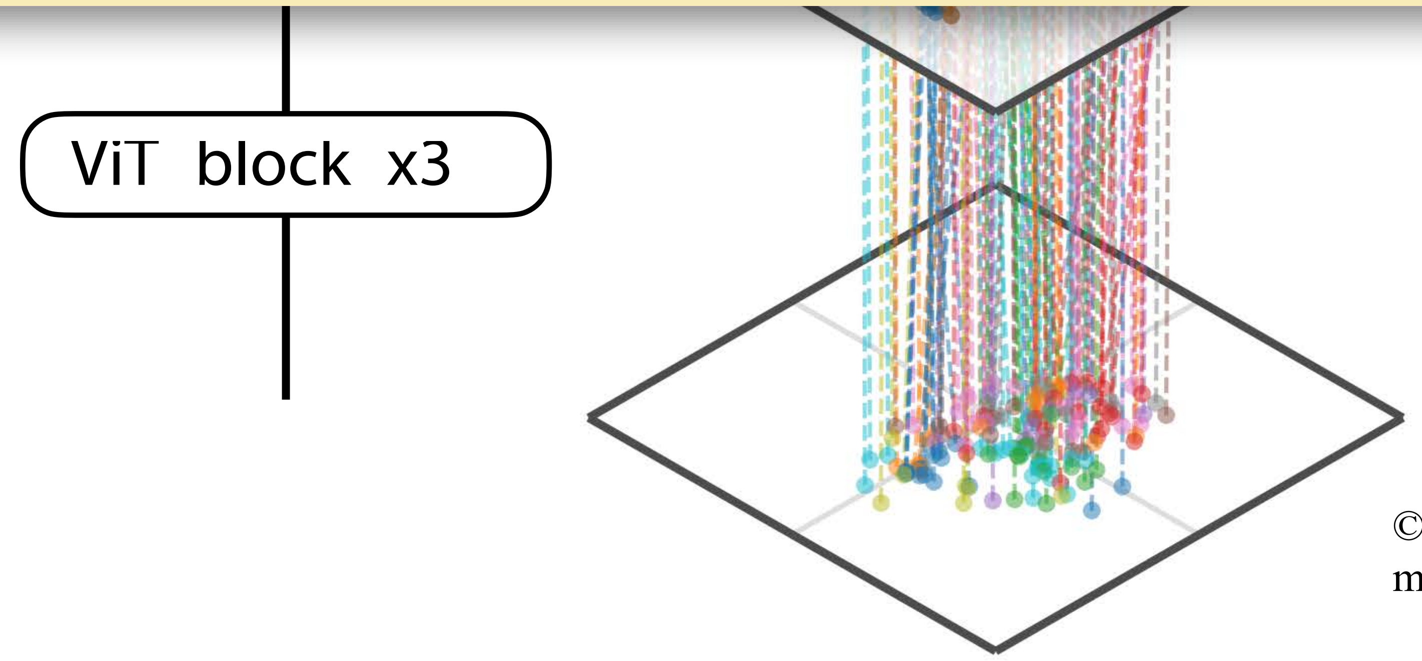


Representation Learning

Lecture 11: Reconstruction-based

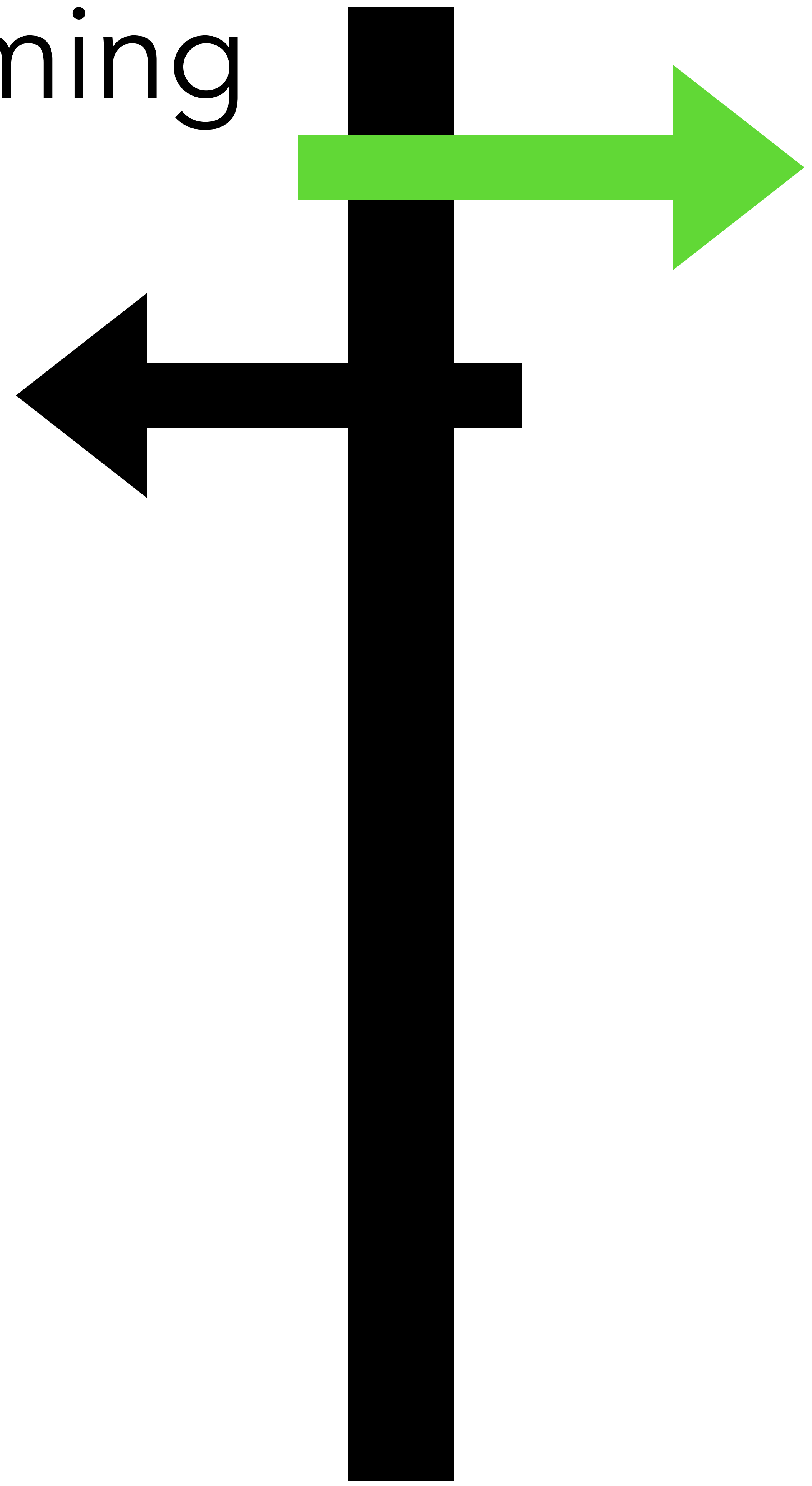
Lecture 12: Similarity-based

Lecture 13: Theory

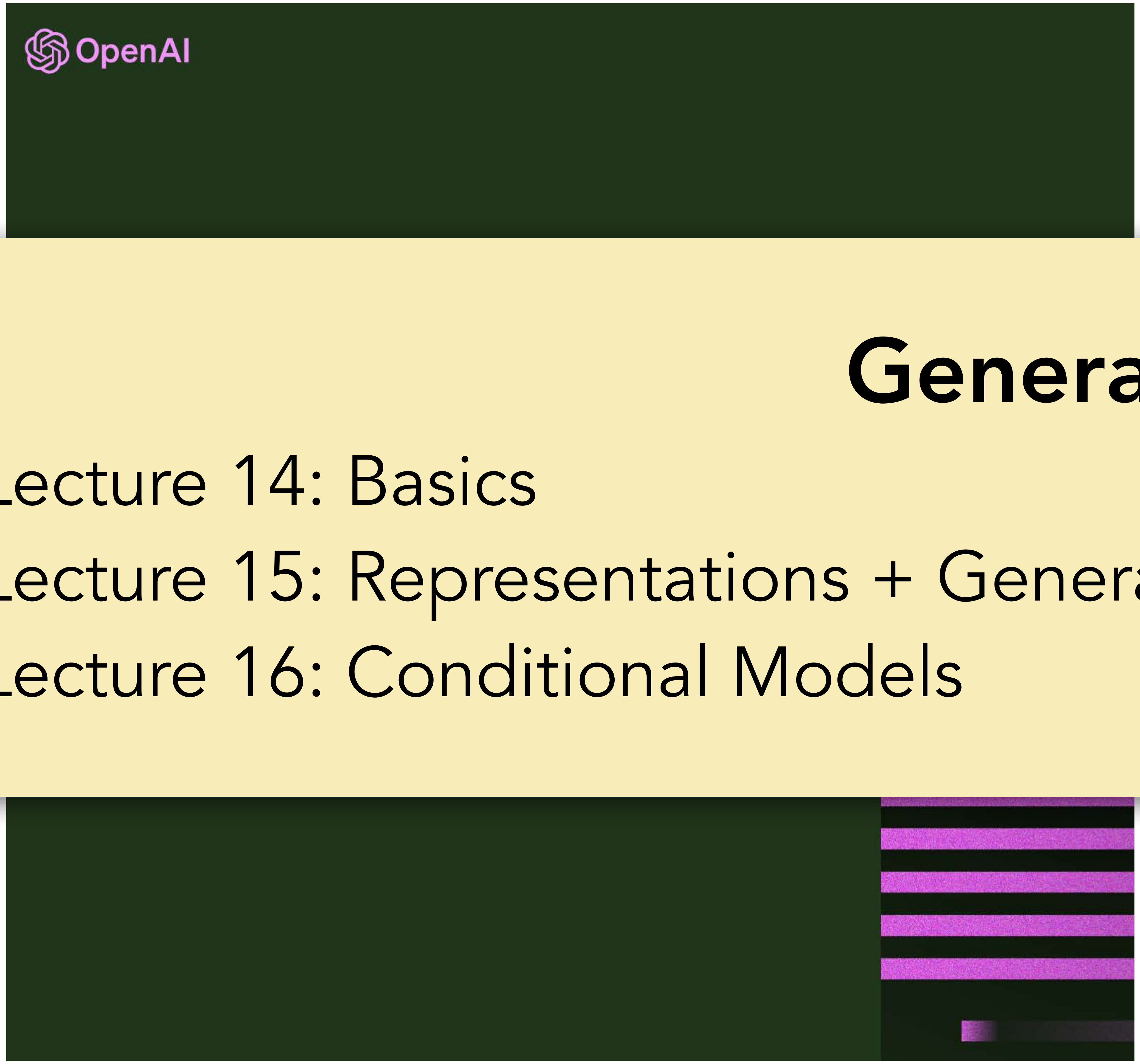


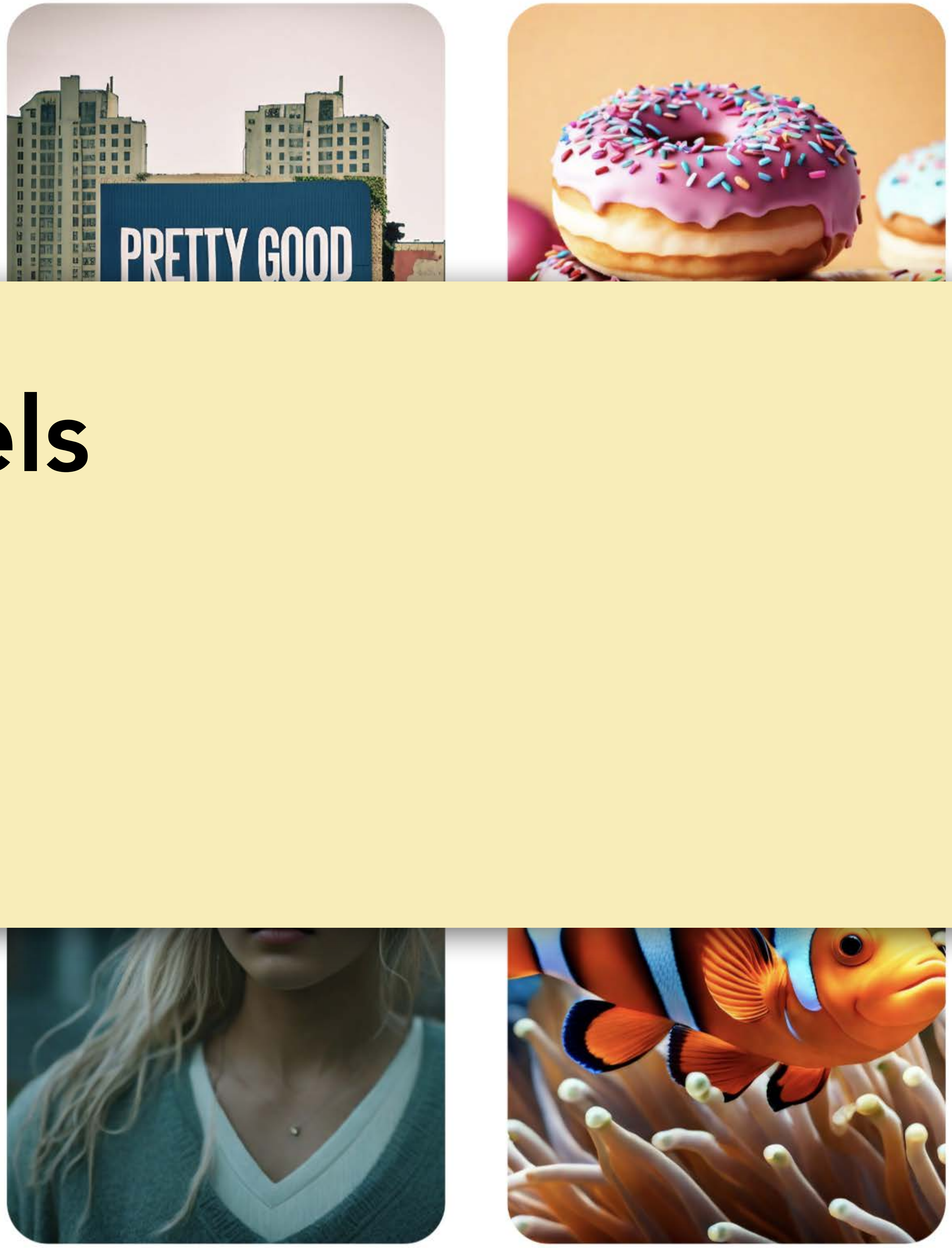
What we’ll cover in this class

- Backprop and differentiable programming
- Why we can approximate
- Architectures
- When and why can we generalize
- How deep networks represent data
- Generative Models



Generative Models





Generative Models

Lecture 14: Basics

Lecture 15: Representations + Generation

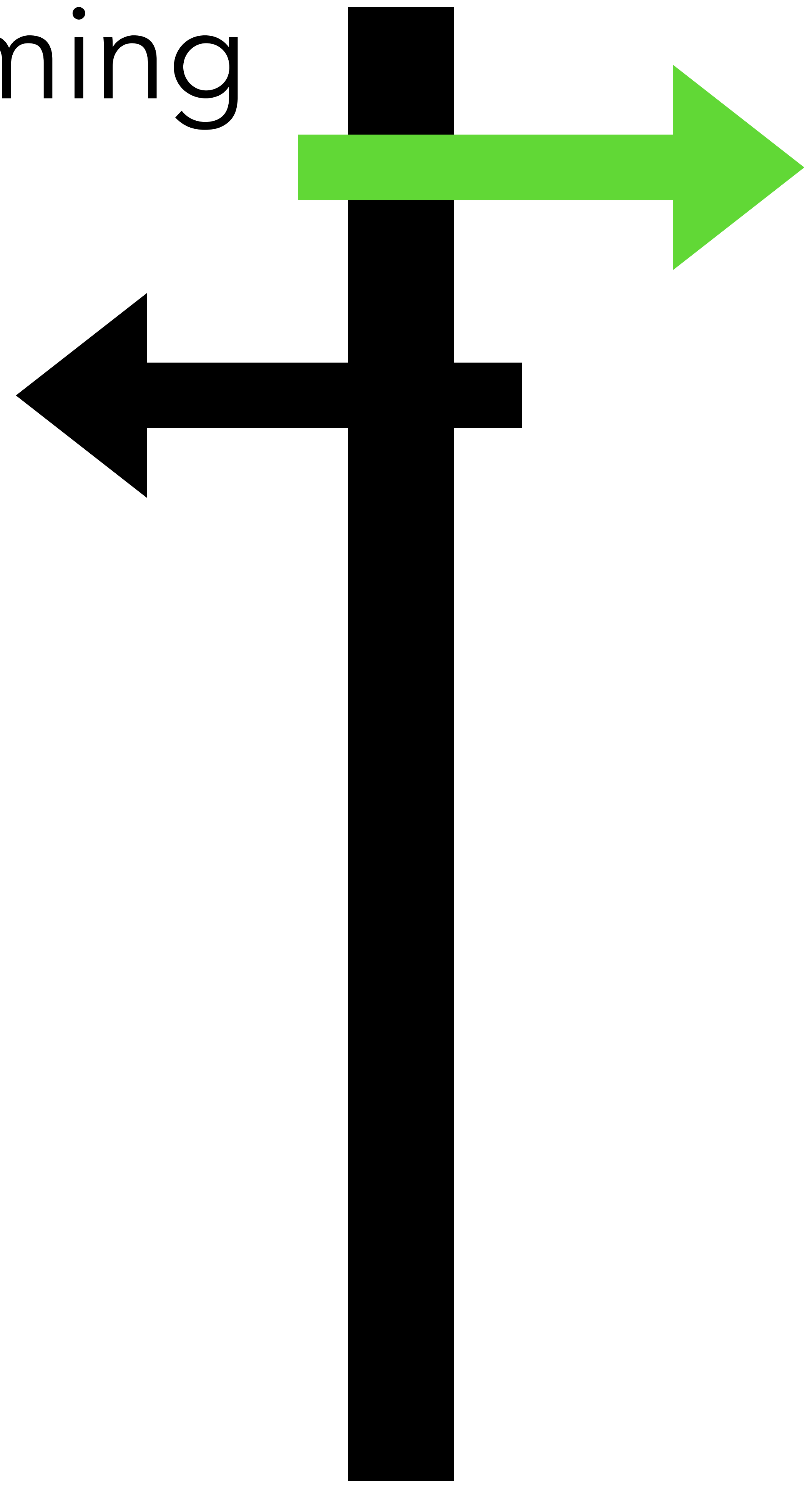
Lecture 16: Conditional Models

openai.com

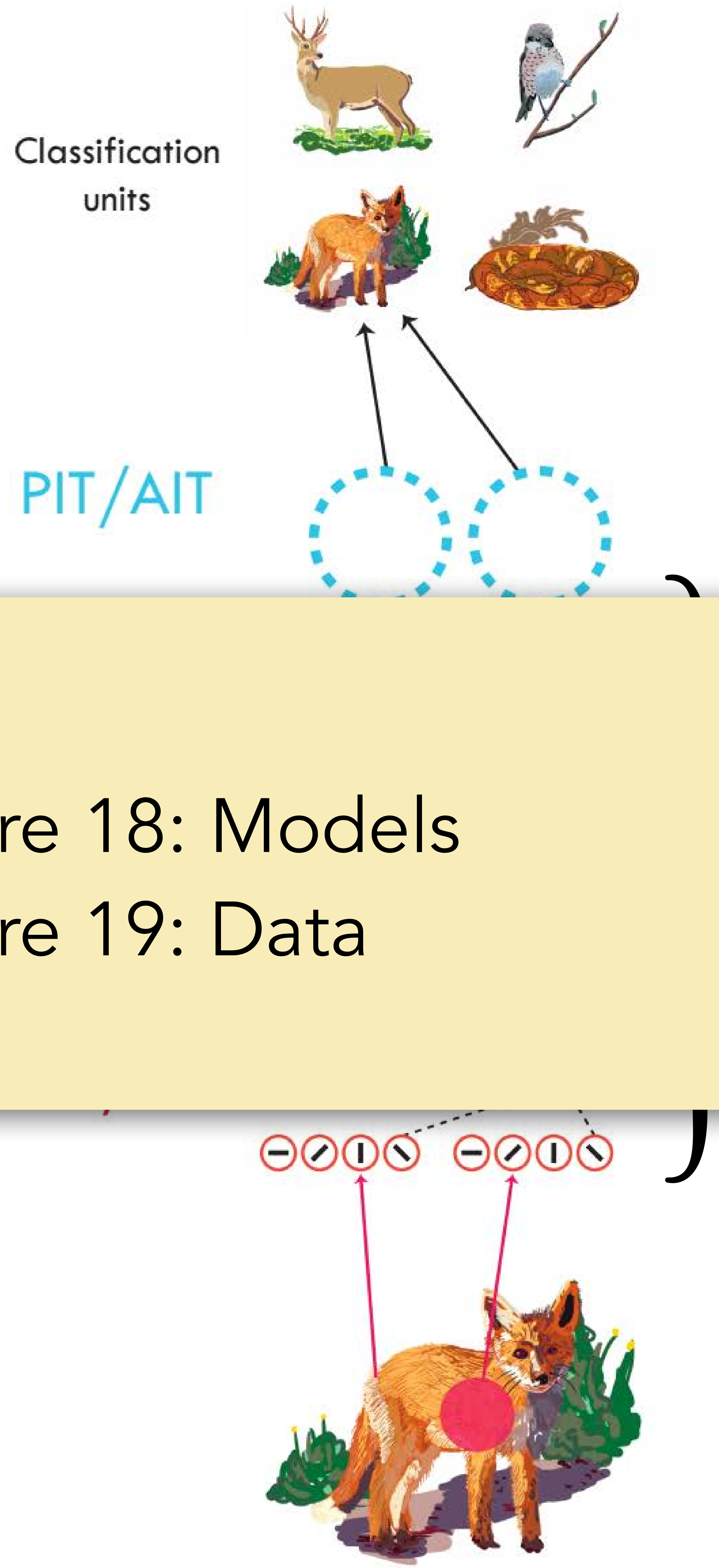
stability.ai

What we'll cover in this class

- Backprop and differentiable programming
- Why we can approximate
- Architectures
- When and why can we generalize
- How deep networks represent data
- Generative Models
- Reusing weights



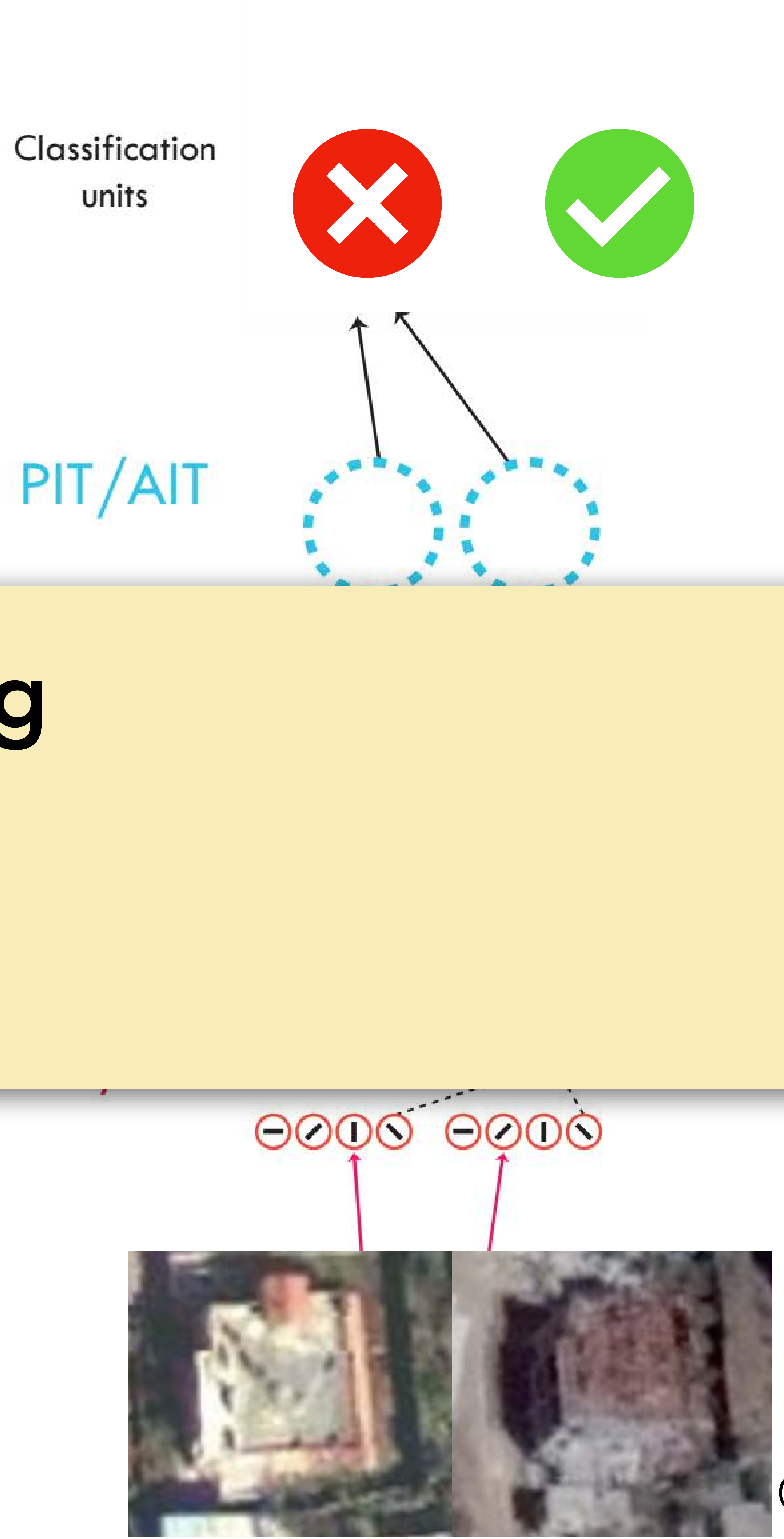
Reuse



Transfer Learning

Lecture 18: Models

Lecture 19: Data

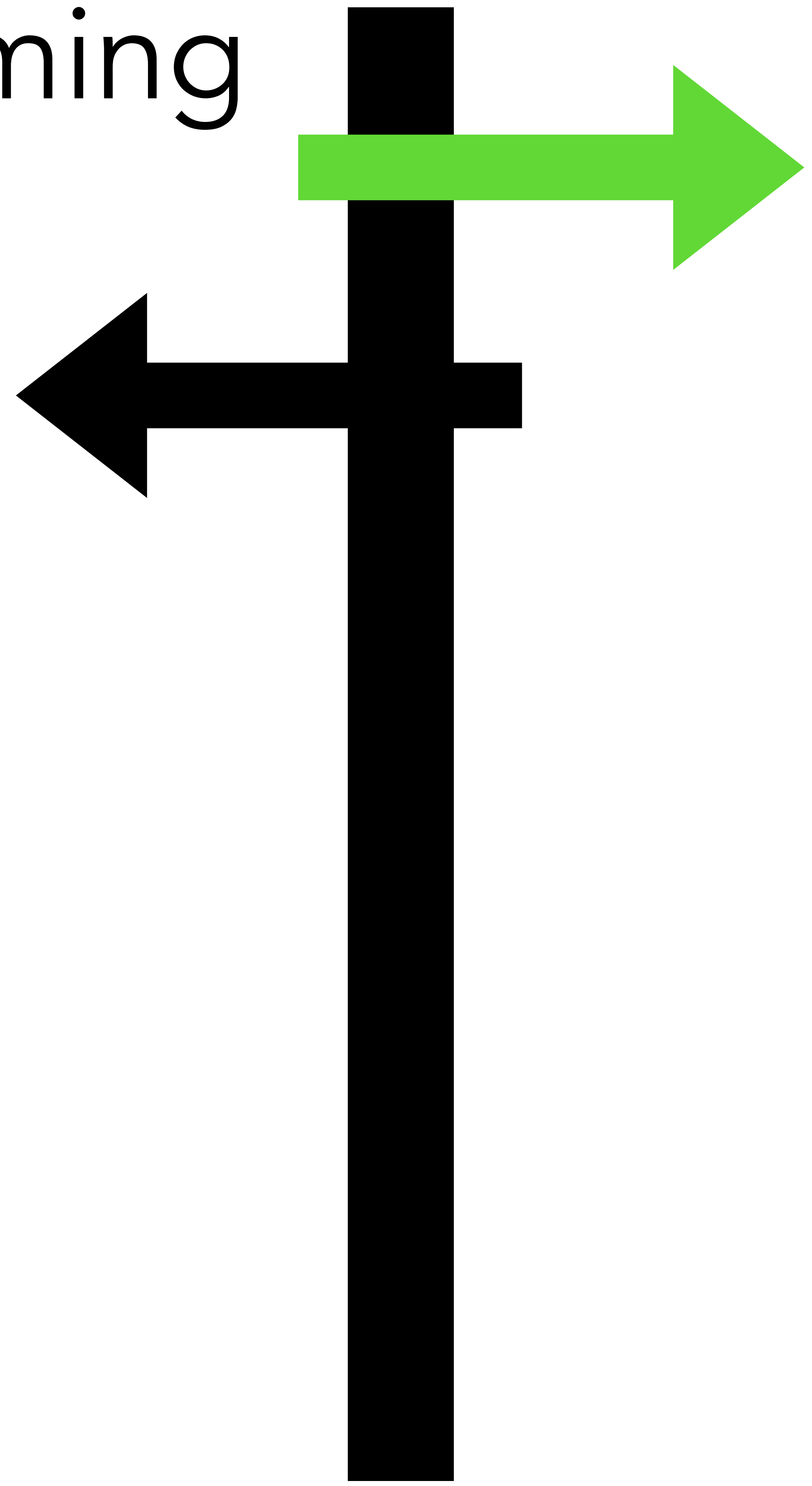


Serre, 2014

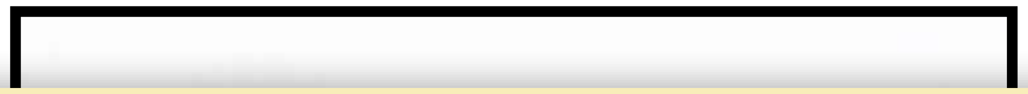
Gandour, 2018

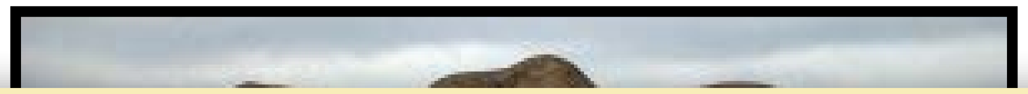
What we'll cover in this class

- Backprop and differentiable programming
- Why we can approximate
- Architectures
- When and why can we generalize
- How deep networks represent data
- Generative models
- Reusing weights
- Scaling



Scale



Scale in Deep Learning

Lecture 6: Scaling Rules for Optimization

Lecture 22: Scaling laws

Lecture 23: Automatic gradient descent

Neurons | Neurons | Neurons | Neurons

1. Introduction to Deep Learning

- How did we get where we are today? (Brief History)
- What we expect you have seen before (ok if you haven't!)
- What we will cover in this class