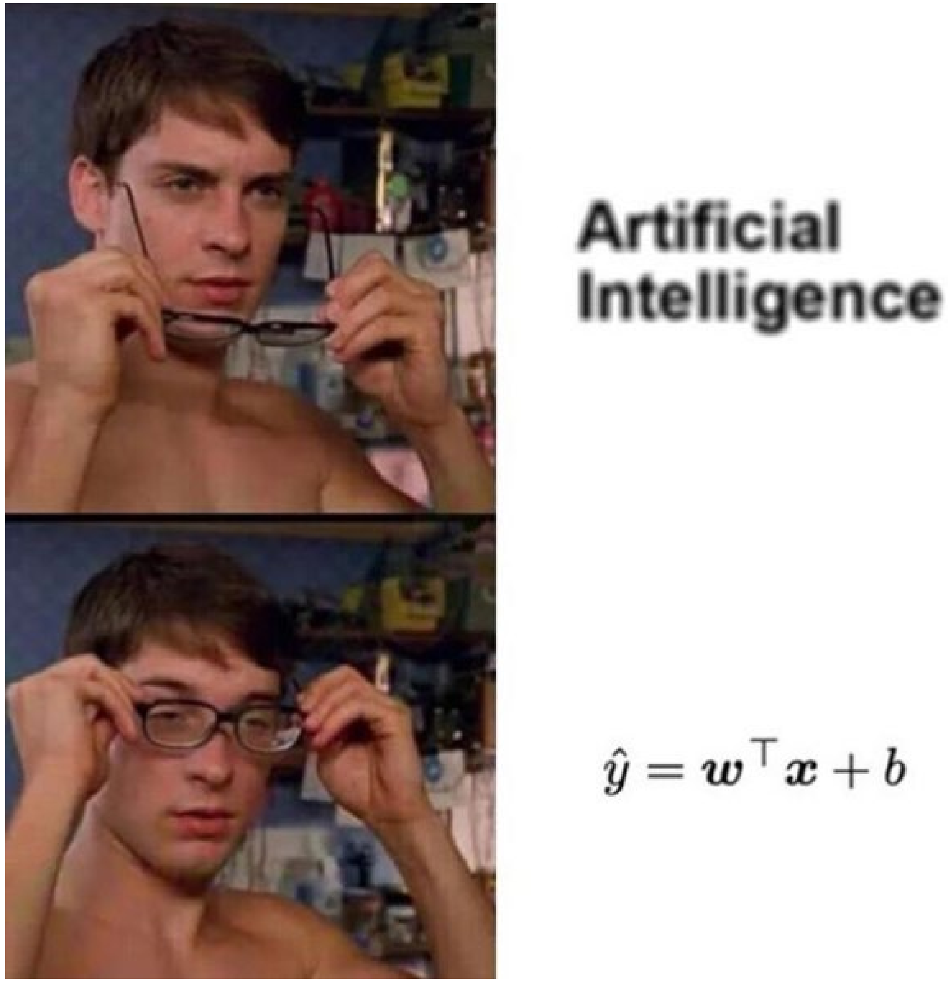
ML Review

CMSC 491/691
Robust Machine Learning



UMBC

Course Staff

Instructor: Tejas Gokhale

Assistant Professor, CSEE

OH:

Wed 2:30 – 3:30 PM ITE 214

gokhale@umbc.edu

TA/Grader ...

???

Course Website

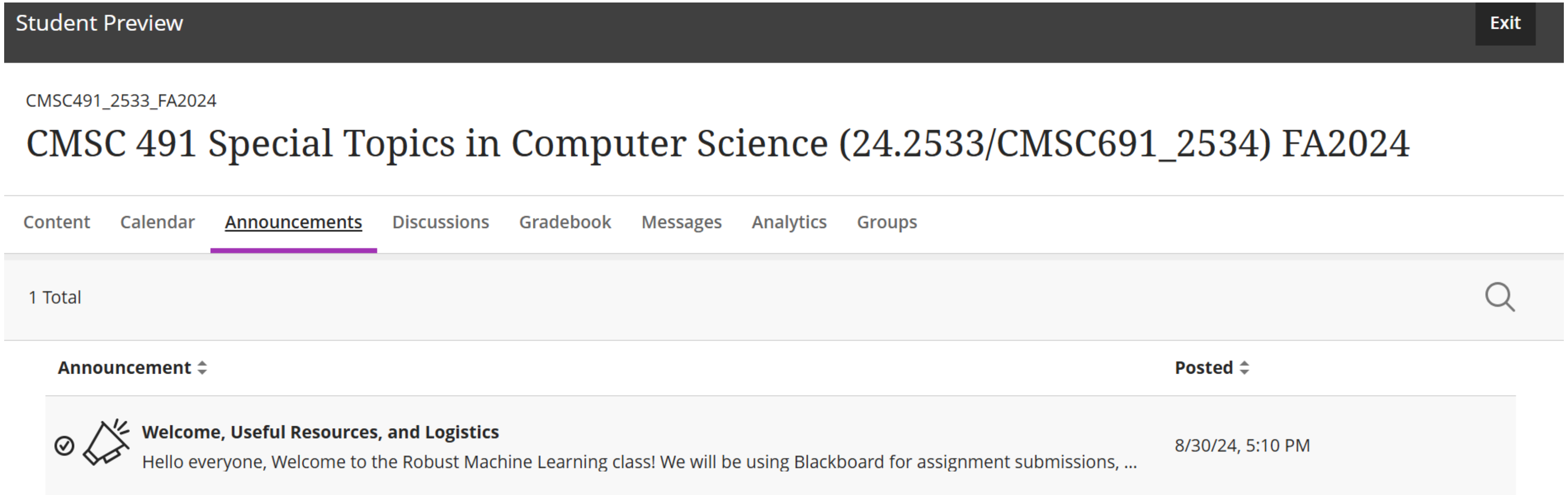
https://courses.cs.umbc.edu/graduate/691rml/

Class Structure: Overview

- [Wed 08/28]: Today, Class Logistics and Introduction
- [Mon 09/02]: Labor Day, NO CLASS
- [Wed 09/04; Mon 09/09; Wed 09/11]: Machine Learning Review
- Every Week after that:
  - MON: [TEJAS] Overview of a Robustness Challenge
  - WED: [STUDENTS] Quiz; Paper Presentations; Class Discussion

Announcements

(1) I sent a "Welcome" announcement on Blackboard. No one read it ...



(2) This week and the next, we will do ML review.

(3) Please sign up for presentations and surveys. Link is in Blackboard Announcement.

(4) Start forming project groups / thinking about ideas. Discuss with me in OFFICE HOURS.

Forcing Function

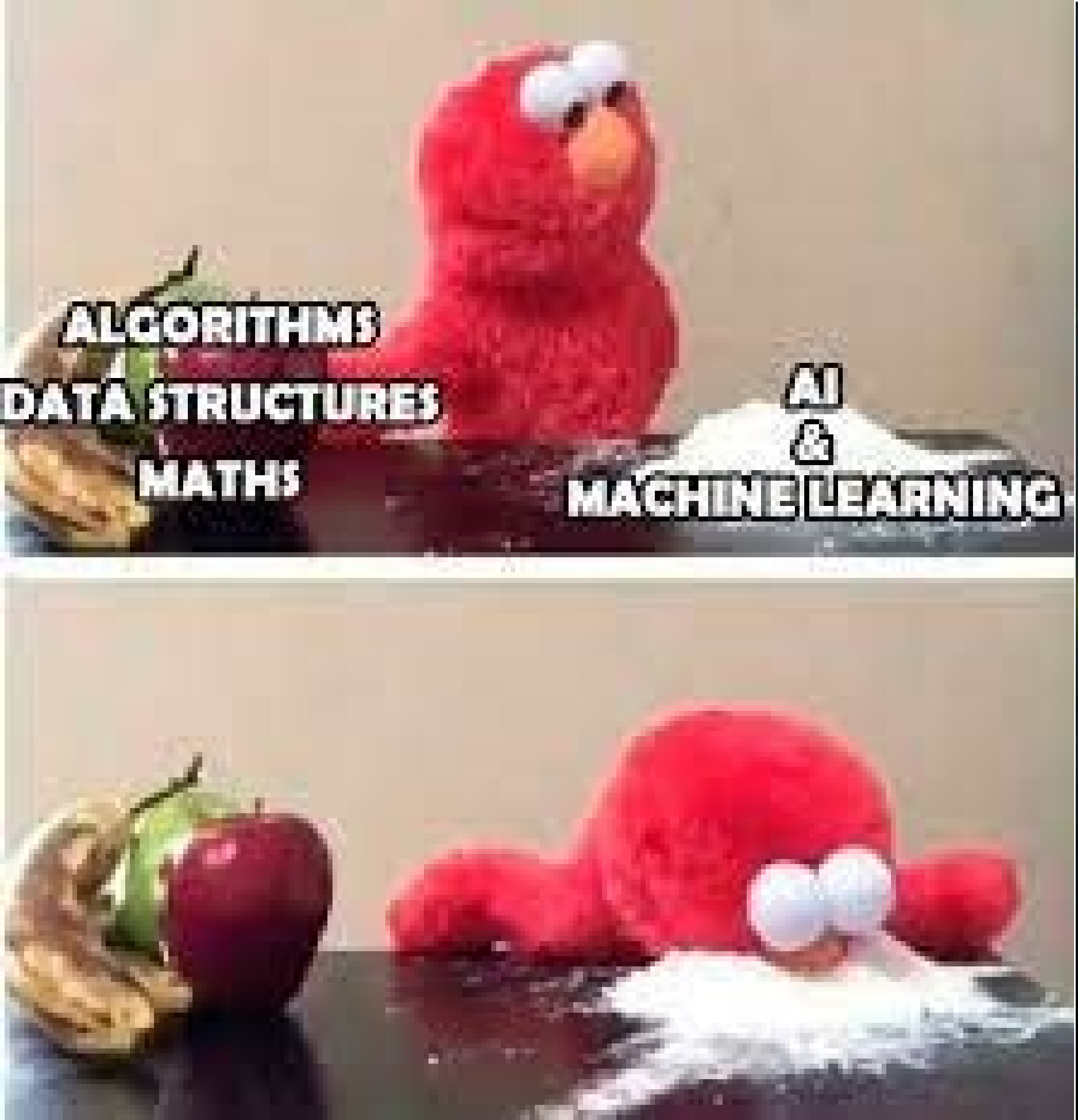
- [CMSC 491/691 RML Presentation and Survey Signup - Google Sheets]

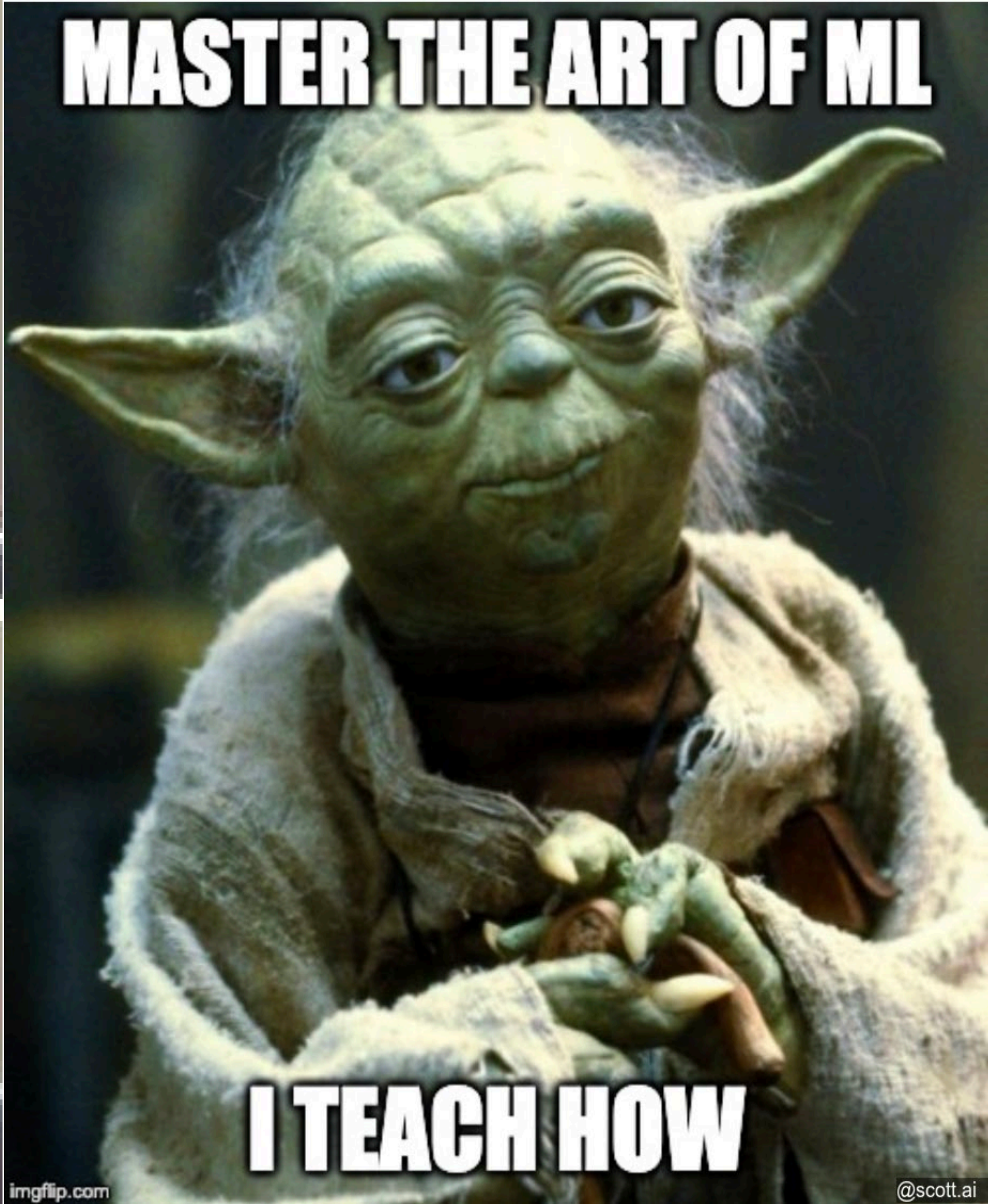
Machine Learning Review

Machine Learning Training and Inference Pipeline

Neural Networks / Deep Learning

ML methods for visual recognition


ALGORITHMS
DATA STRUCTURES
MATHS
AI
&
MACHINE LEARNING


MASTER THE ART OF ML
I TEACH HOW
imgflip.com
@scott.ai


PRETTY, PRETTY, PRETTY,
PRETTY GOOD.


THAT'S GOOD
STUFF.


THAT'S GOOD STUFF!


THAT'S THE GOOD STUFF

Motivating Example: Image Classification

What is this?

{dog, cat, airplane, bus, laptop, chair …}

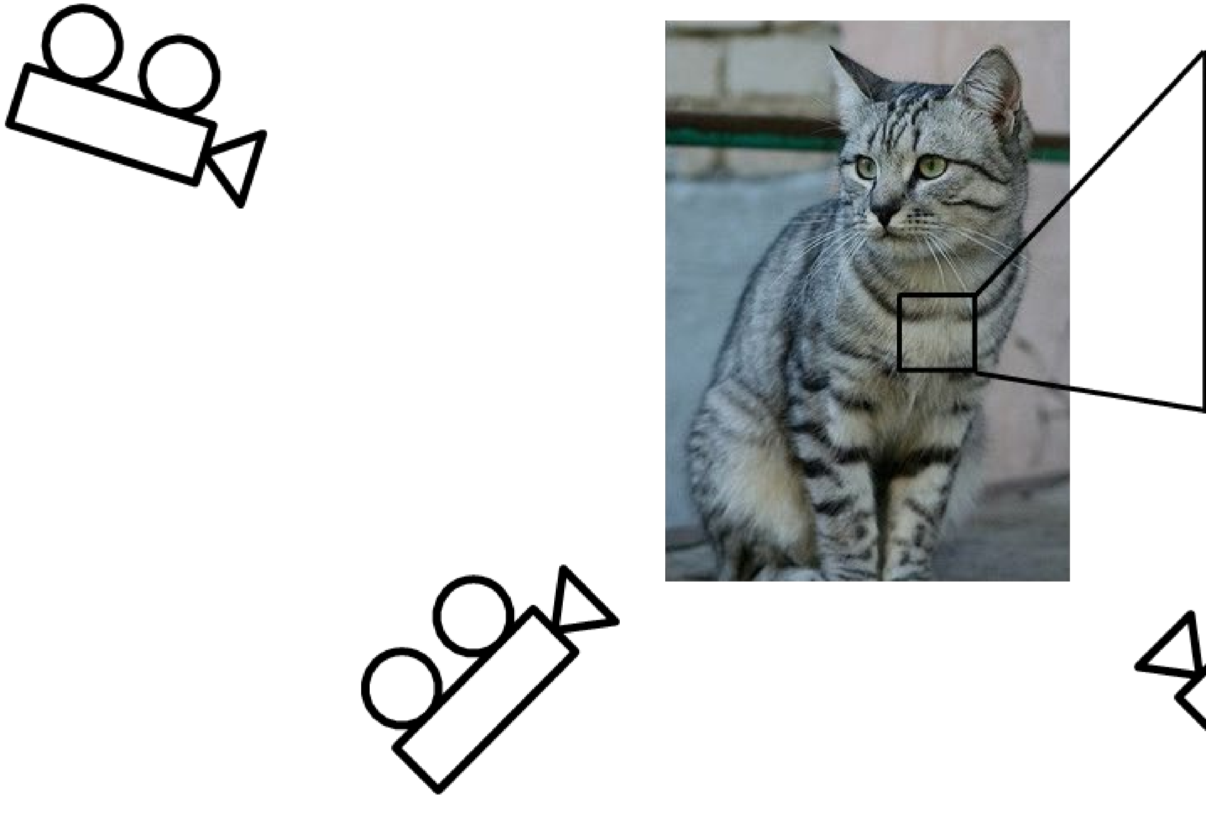
What animal is this ?

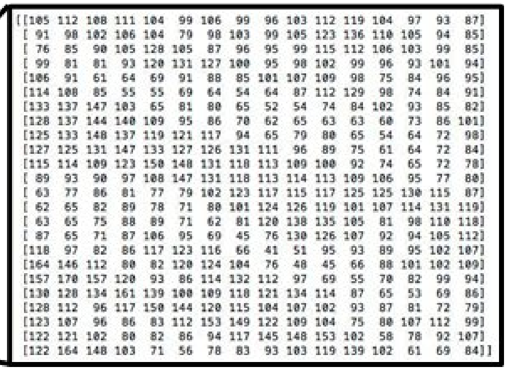
{dog, cat, lion, tiger, duck, cow, giraffe, …}

What type of cat is this?

{Cheshire, Siamese, Persian, Shorthair, Bombay, …}

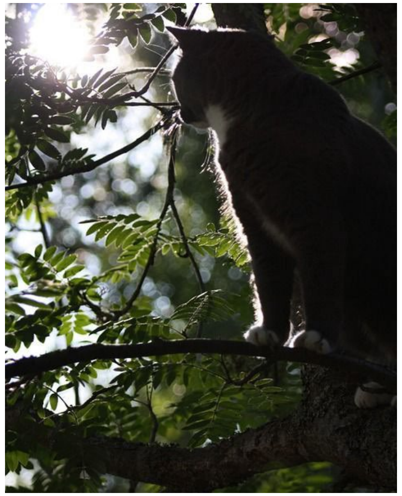
Challenges: Viewpoint Variation





All pixels change when the camera moves!

Challenges: Illumination



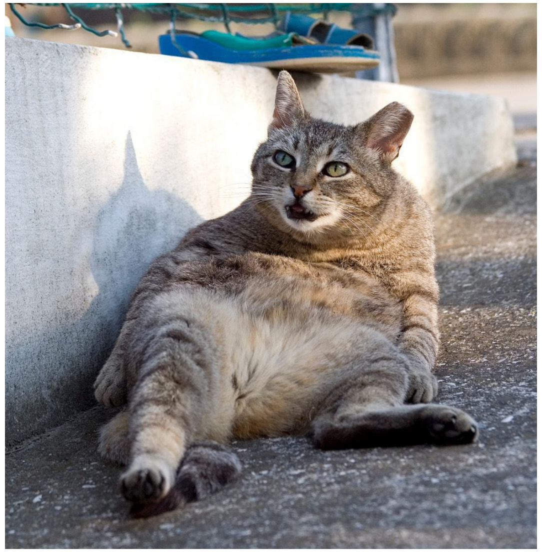
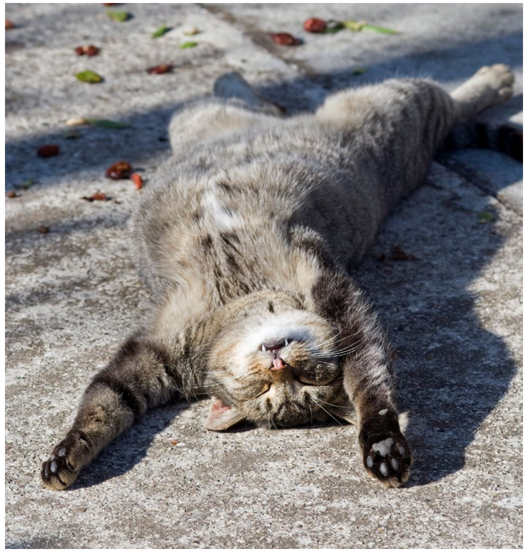
Challenges: Background Clutter

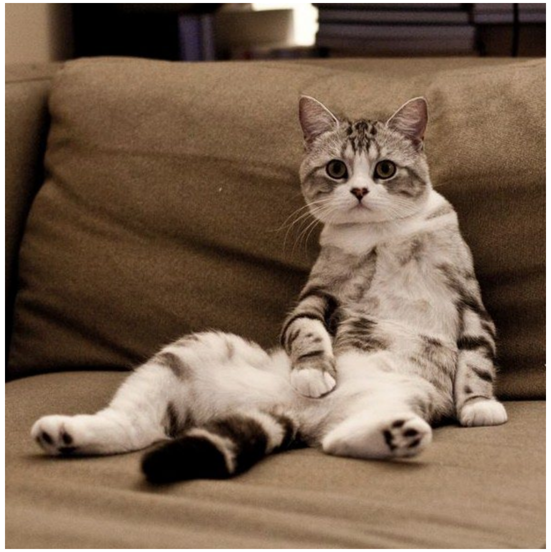
Slide Credit: Fei-Fei Li

Challenges: Occlusion

Slide Credit: Fei-Fei Li

Challenges: Pose and Deformation (Cat Yoga)

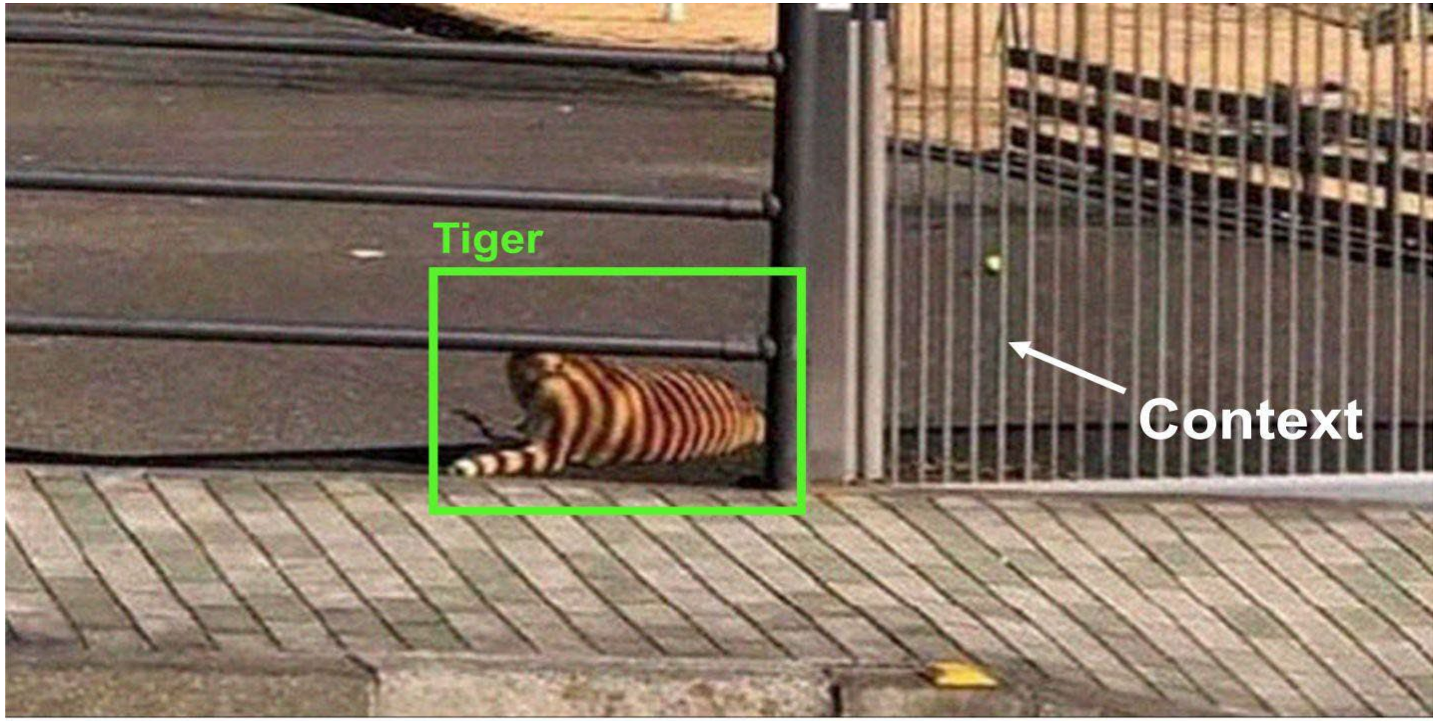


Slide Credit: Fei-Fei Li

Challenges: Inter-Class Variation

Slide Credit: Fei-Fei Li

Challenges: Illusions



Data !!!

An image classifier

```
def classify_image(image):
  # Some magic here?
  return class_label
```

Unlike e.g. sorting a list of numbers,

no obvious way to hard-code the algorithm for recognizing a cat, or other classes.

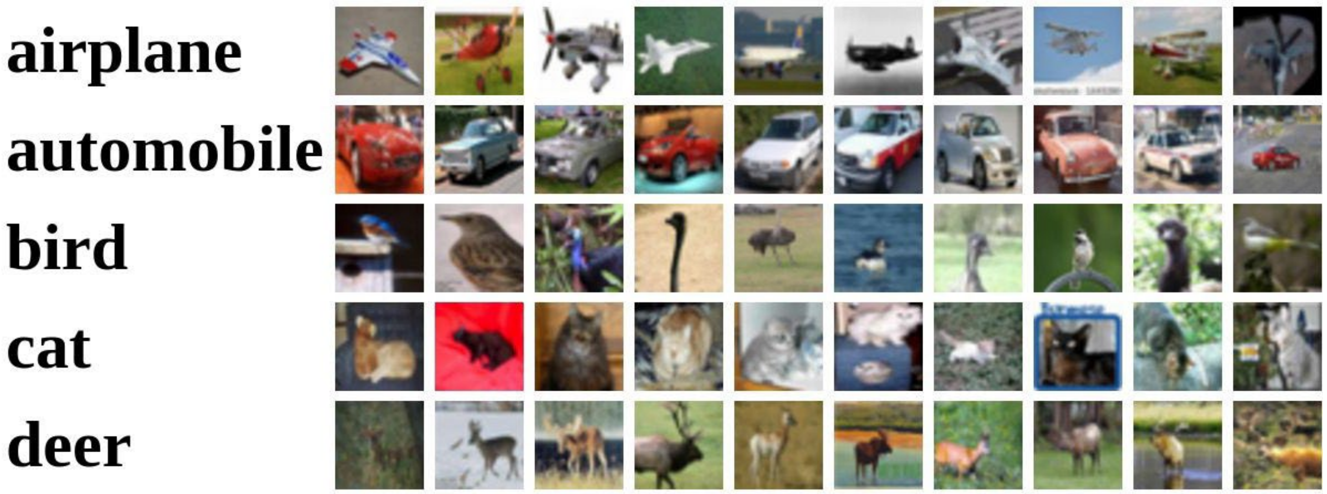
Machine Learning

1. Collect a dataset of images and labels
2. Use Machine Learning algorithms to train a classifier
3. Evaluate the classifier on new images

```python
def train(images, labels):
  # Machine learning!
  return model
```

```python
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

Example training set



Nearest Neighbor Classifier

```
def train(images, labels):
  # Machine learning!
  return model
```

Memorize all data and labels

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

Predict the label of the most similar training image

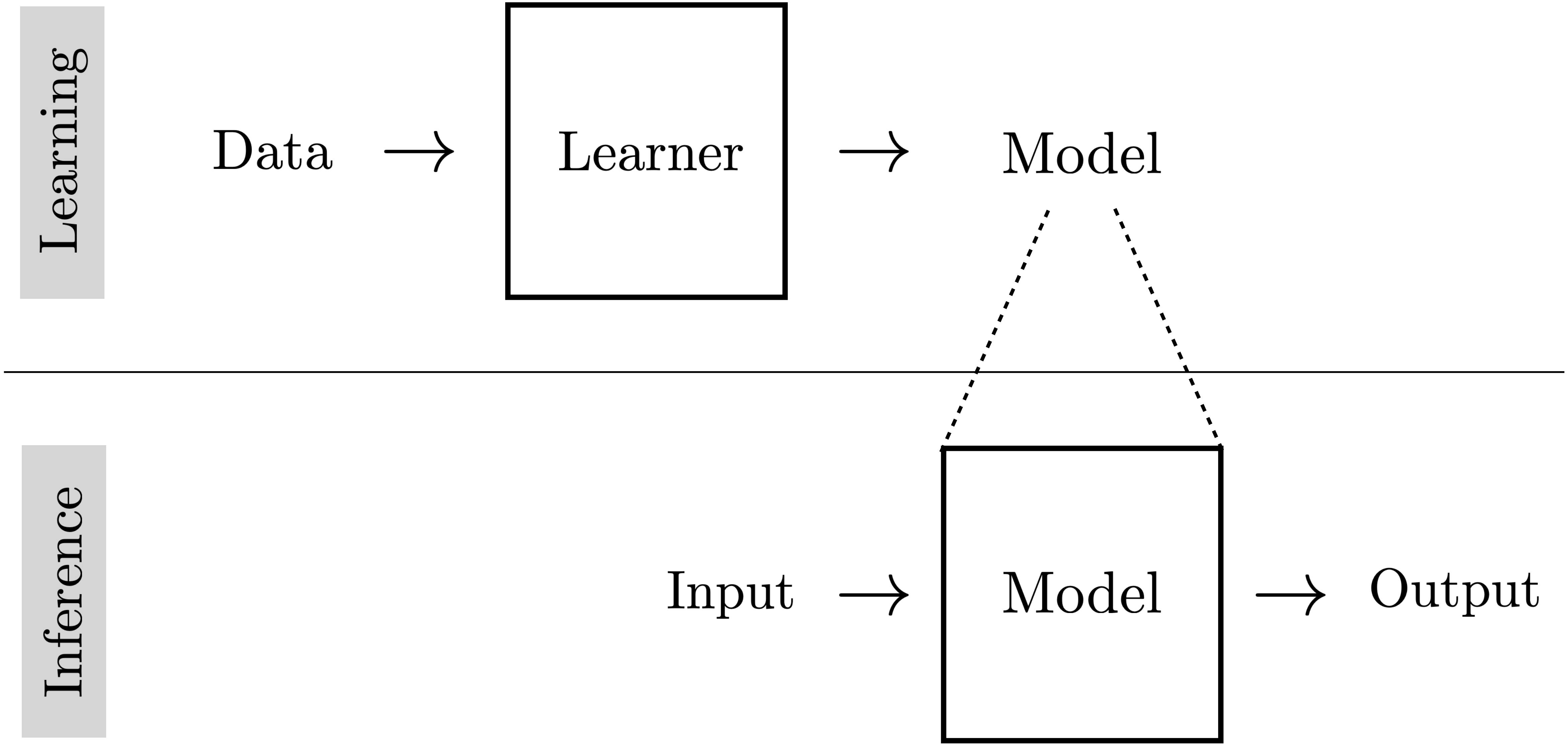
Nearest Neighbor Classifier

deer bird plane cat car

Training data with labels

query data ?

Distance Metric $\Big|$  ,  $\Big| \to \mathbb{R}$



Learning
Data
Learner
Model
Inference
Input
Model
Output

Learning

Da

```python
def train(images, labels):
  # Machine learning!
  return model
```

```python
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

Inference

→ Output

The goal of learning is to extract lessons from past experience in order to solve future problems.

What does ☆ do?

$$2 \star 3 = 36$$

$$7 \star 1 = 49$$

$$5 \star 2 = 100$$

$$2 \star 2 = 16$$

Goal: answer future queries involving ☆

Approach: figure out what ☆ is doing by observing its behavior on examples

Past experience

2 ☆ 3 = 36

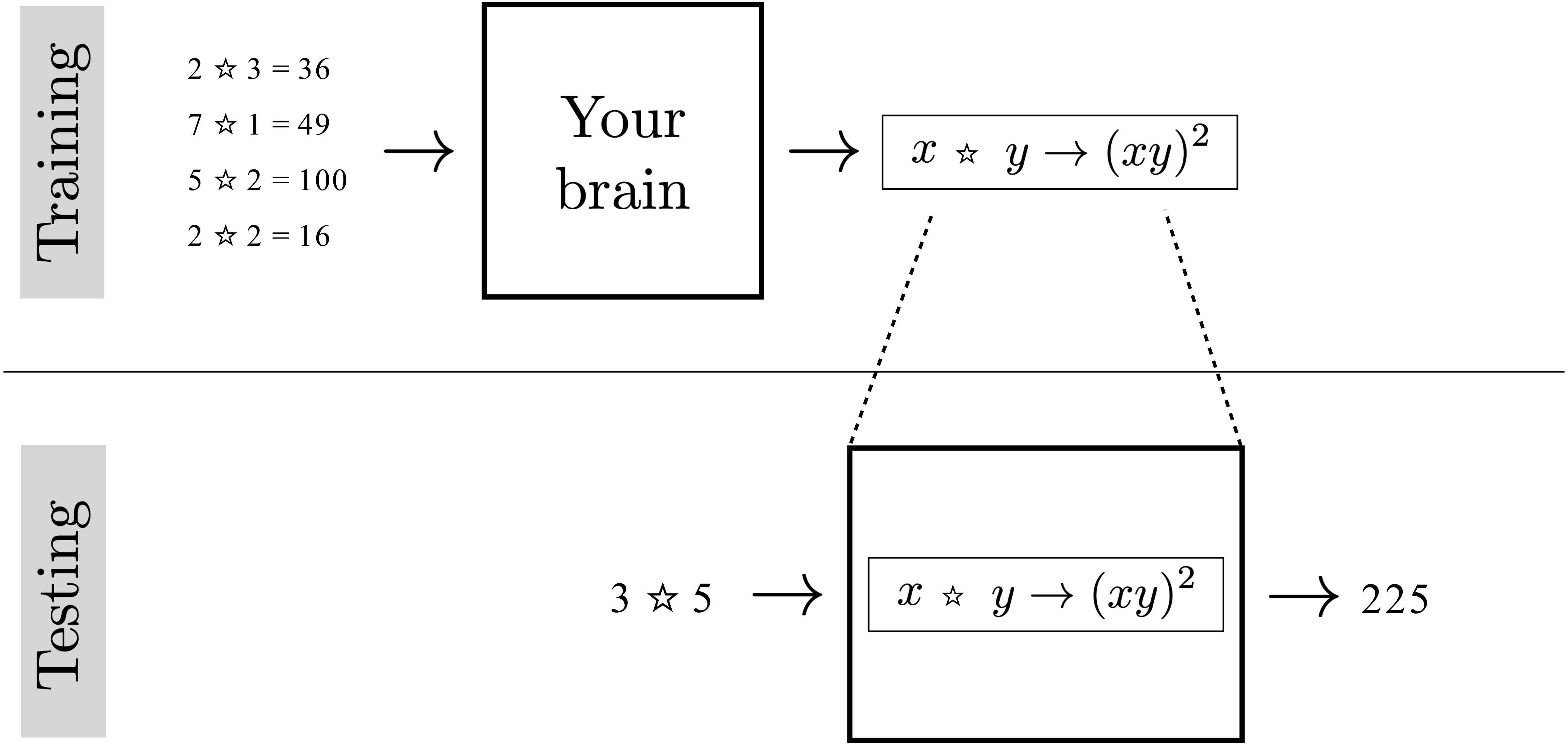
7 ☆ 1 = 49

5 ☆ 2 = 100

2 ☆ 2 = 16

Future query

3 ☆ 5 = ?


Training
2 ☆ 3 = 36
7 ☆ 1 = 49
5 ☆ 2 = 100
2 ☆ 2 = 16
Your brain
$x \star y \to (xy)^2$
Testing
3 ☆ 5
$x \star y \to (xy)^2$
225

Learning from examples

(aka **supervised learning**)

Training data

$\{\texttt{input}{:}[2,3], \texttt{output}{:}36\}$

$\{\texttt{input}{:}[7,1], \texttt{output}{:}49\}$

$\{\texttt{input}{:}[5,2], \texttt{output}{:}100\}$

$\{\texttt{input}{:}[2,2], \texttt{output}{:}16\}$

$\rightarrow$ $\boxed{\text{Learner}}$ $\rightarrow$ f

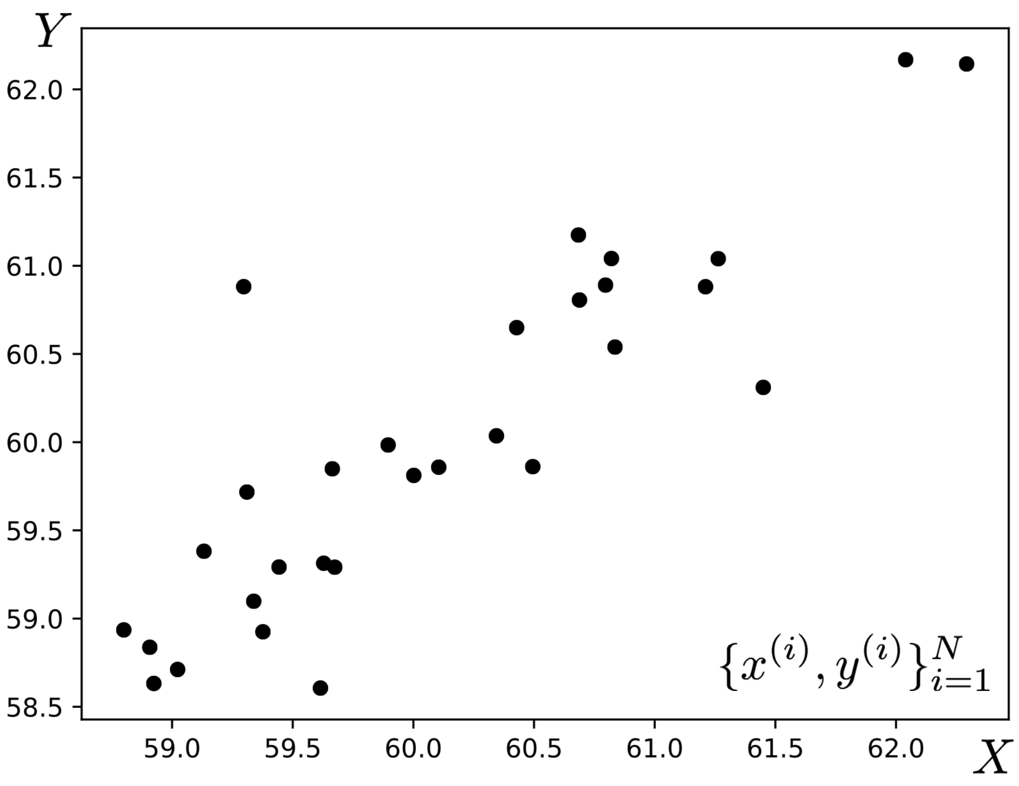
Learning from examples

(aka **supervised learning**)

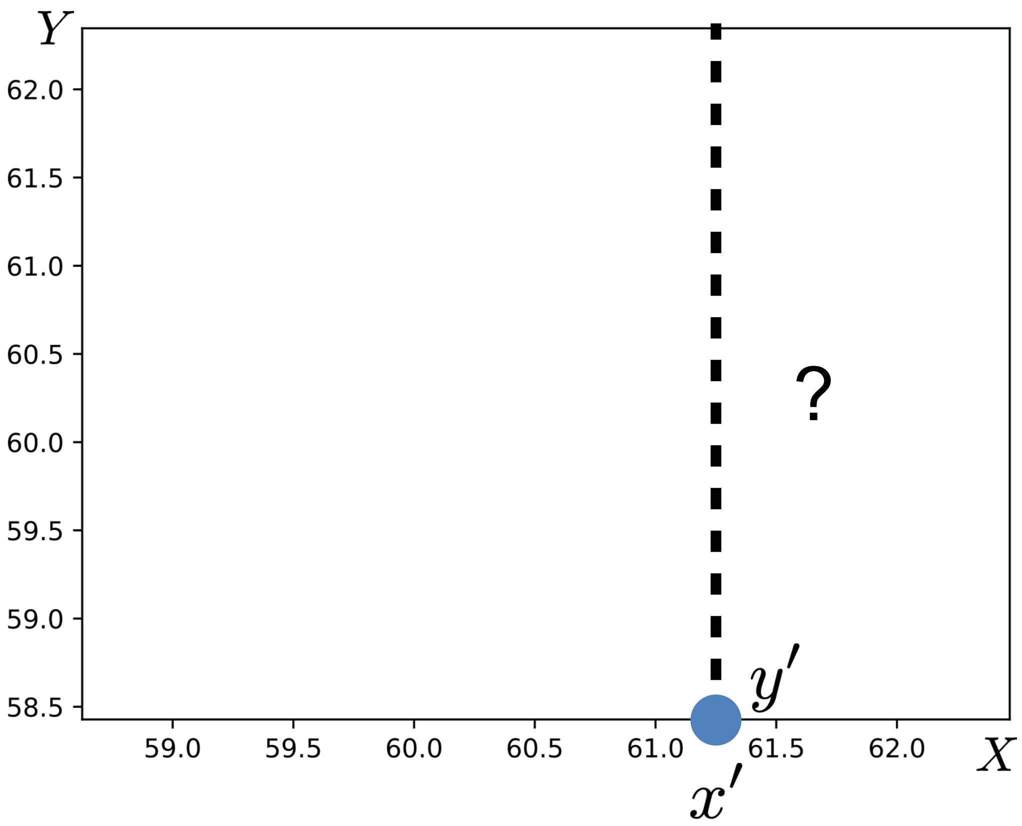
Training data

$$\{x^{(1)}, y^{(1)}\}$$
$$\{x^{(2)}, y^{(2)}\} \quad \rightarrow \quad \boxed{\text{Learner}} \quad \rightarrow \quad f : X \rightarrow Y$$
$$\{x^{(3)}, y^{(3)}\}$$
$$\ldots$$

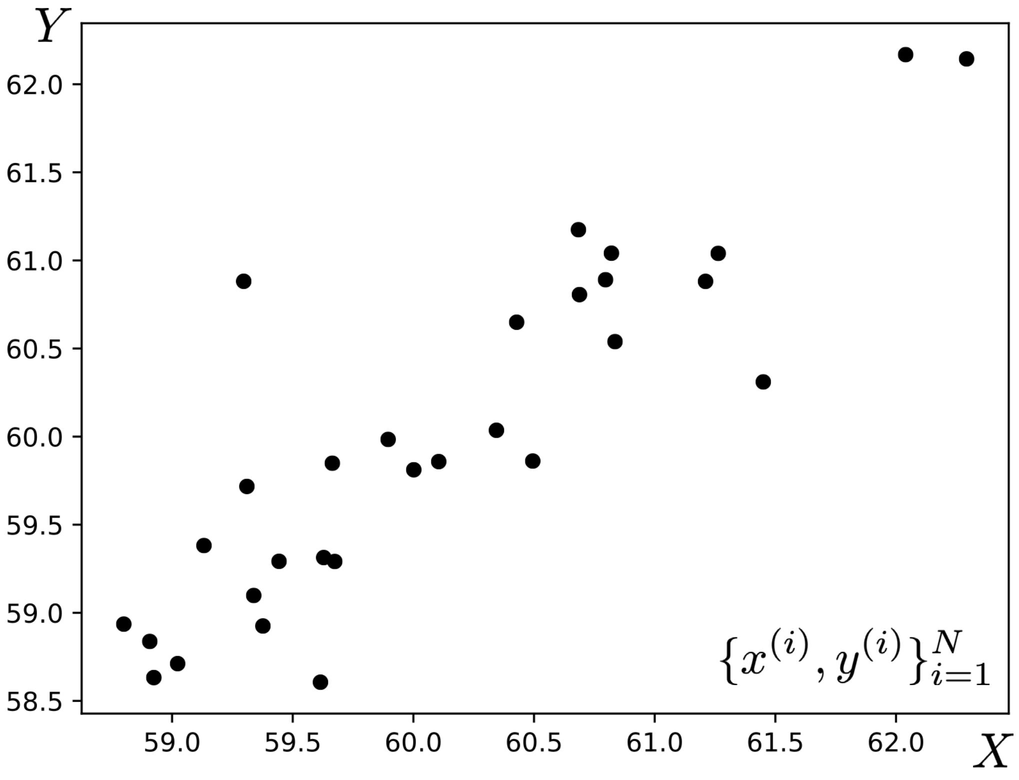
Training data



Test query



Training data

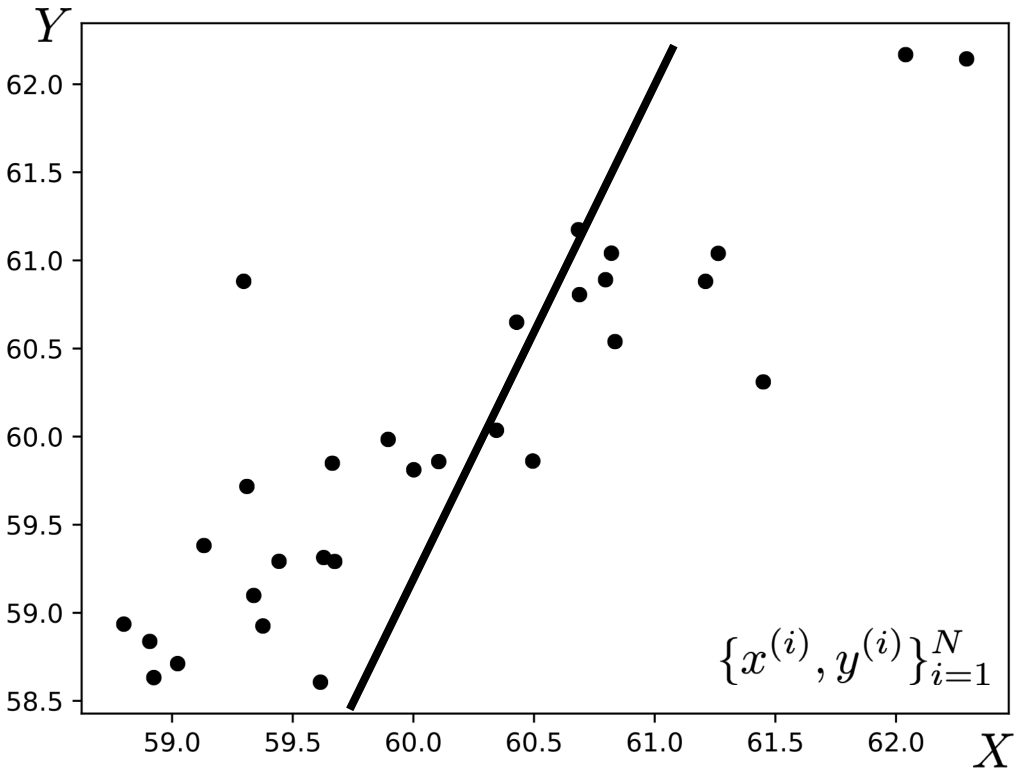


$\rightarrow$ Learner $\rightarrow$ $f_\theta(x) = \theta_1 x + \theta_0$

Hypothesis space

The relationship between X and Y is roughly linear: $y \approx \theta_1 x + \theta_0$

Training data

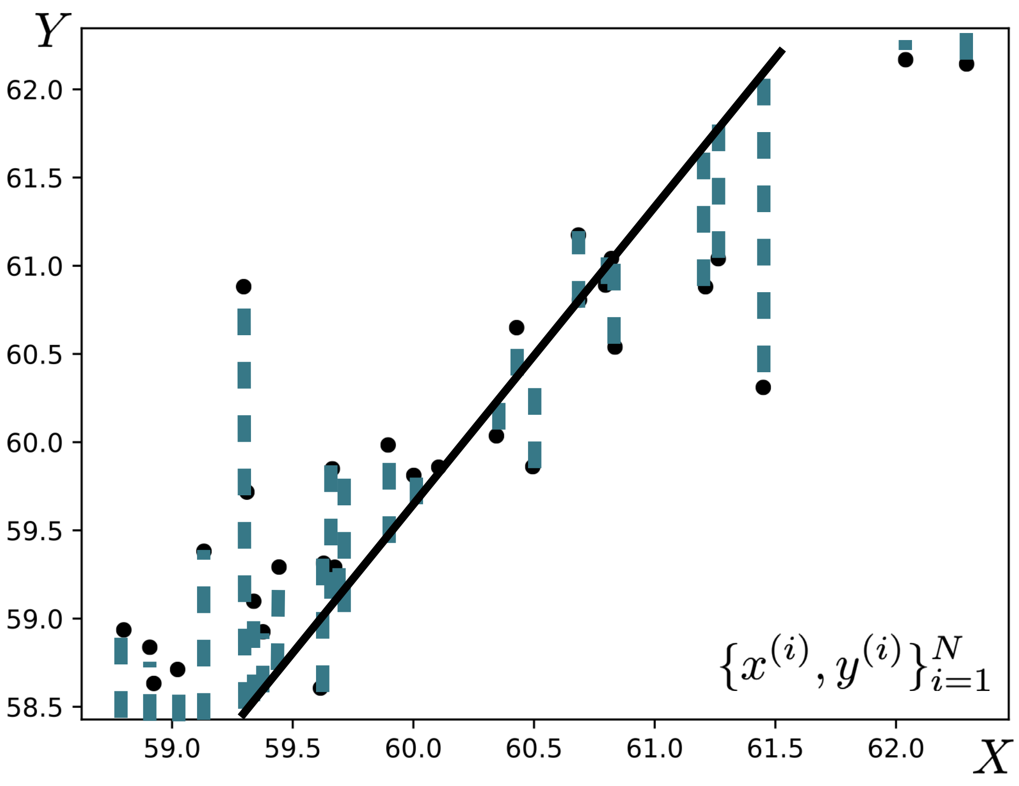


Search for the **parameters**, $\theta = \{\theta_0, \theta_1\}$, that best fit the data.

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

Best fit in what sense?

Training data



Search for the **parameters**, $\theta = \{\theta_0, \theta_1\}$, that best fit the data.

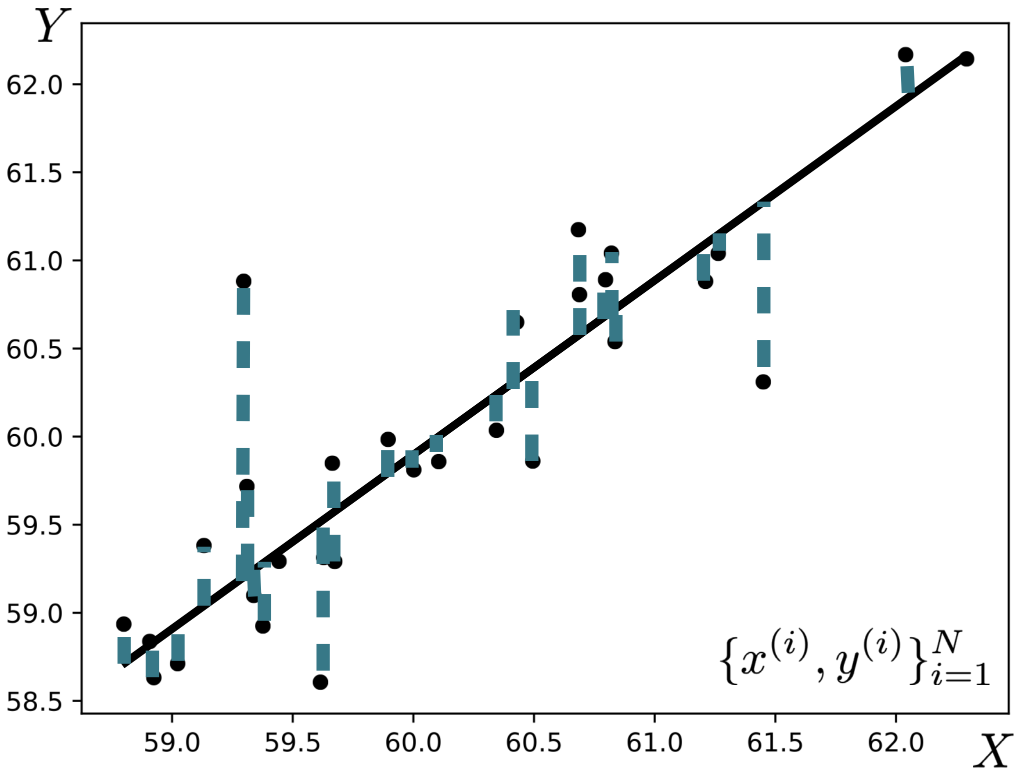
$$f_\theta(x) = \theta_1 x + \theta_0$$

Best fit in what sense?

The least-squares **objective** (aka **loss**) says the best fit is the function that minimizes the squared error between predictions and target values:

$$\mathcal{L}(\hat{y}, y) = (\boxed{\hat{y} - y})^2 \qquad \hat{y} \equiv f_\theta(x)$$

Training data



Search for the **parameters**, $\theta = \{\theta_0, \theta_1\}$, that best fit the data.

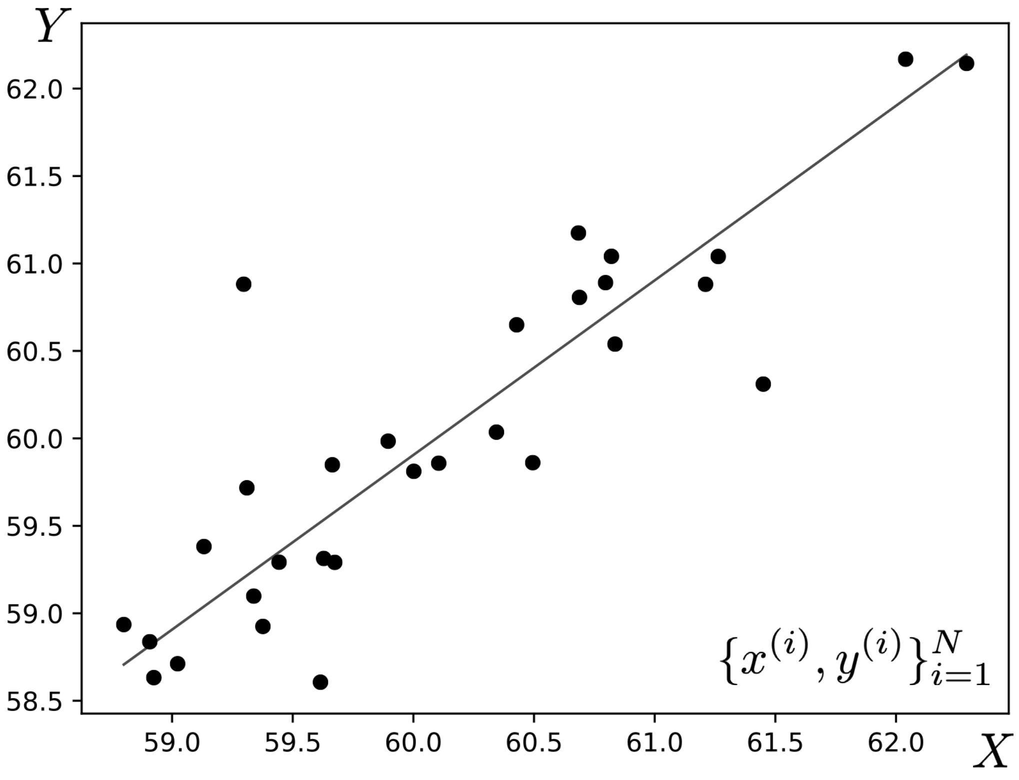
$$f_\theta(x) = \theta_1 x + \theta_0$$

Best fit in what sense?

The least-squares **objective** (aka **loss**) says the best fit is the function that minimizes the squared error between predictions and target values:

$$\mathcal{L}(\hat{y}, y) = \boxed{(\hat{y} - y)}^2 \qquad \hat{y} \equiv f_\theta(x)$$

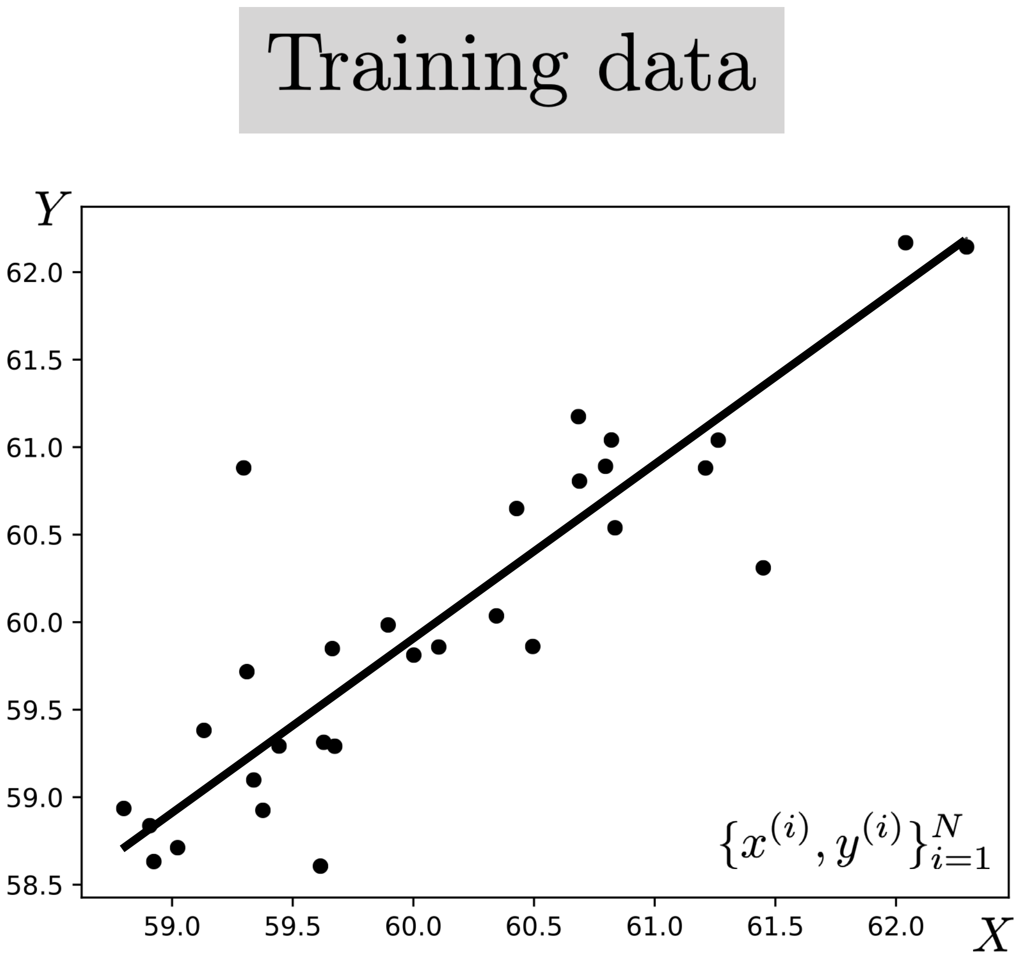
Training data



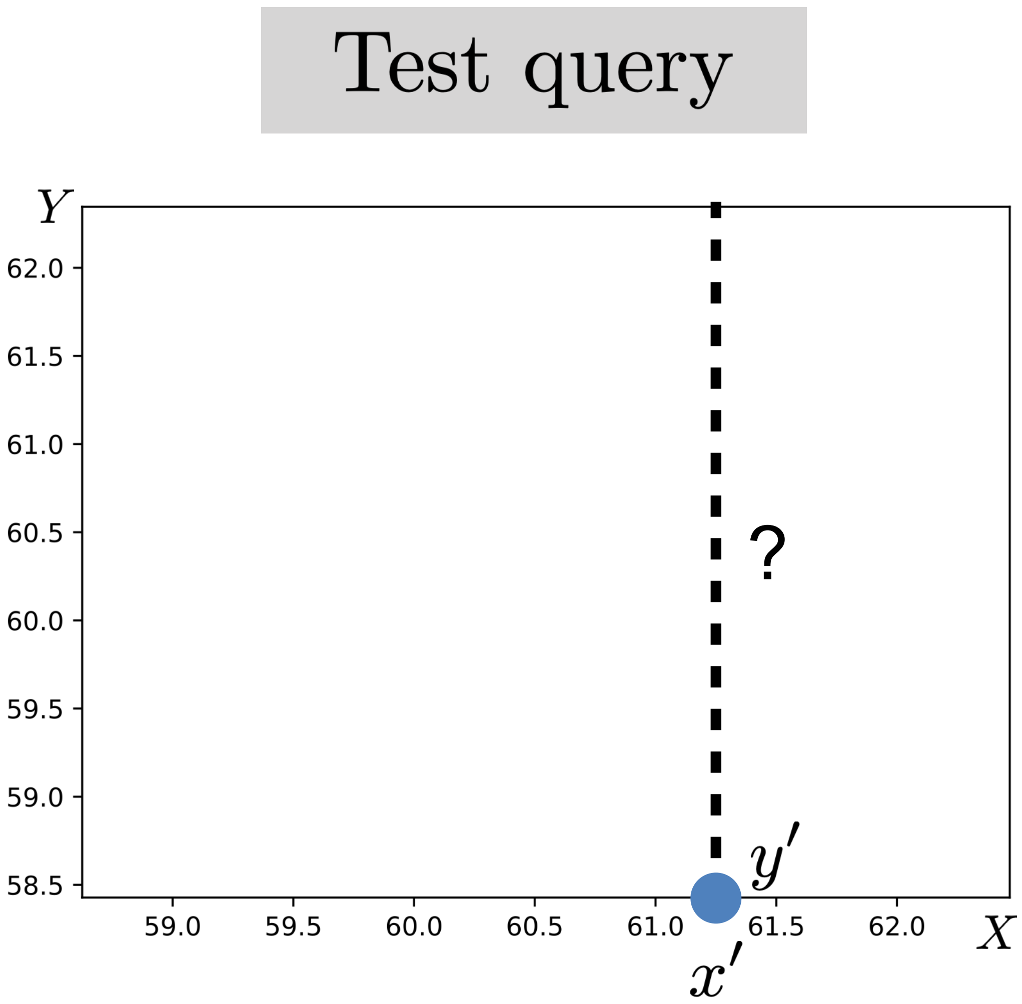
Complete learning problem:

$$\theta^* = \arg\min_{\theta} \sum_{i=1}^{N} (f_\theta(x^{(i)}) - y^{(i)})^2$$

$$= \arg\min_{\theta} \sum_{i=1}^{N} (\theta_1 x^{(i)} + \theta_0 - y^{(i)})^2$$

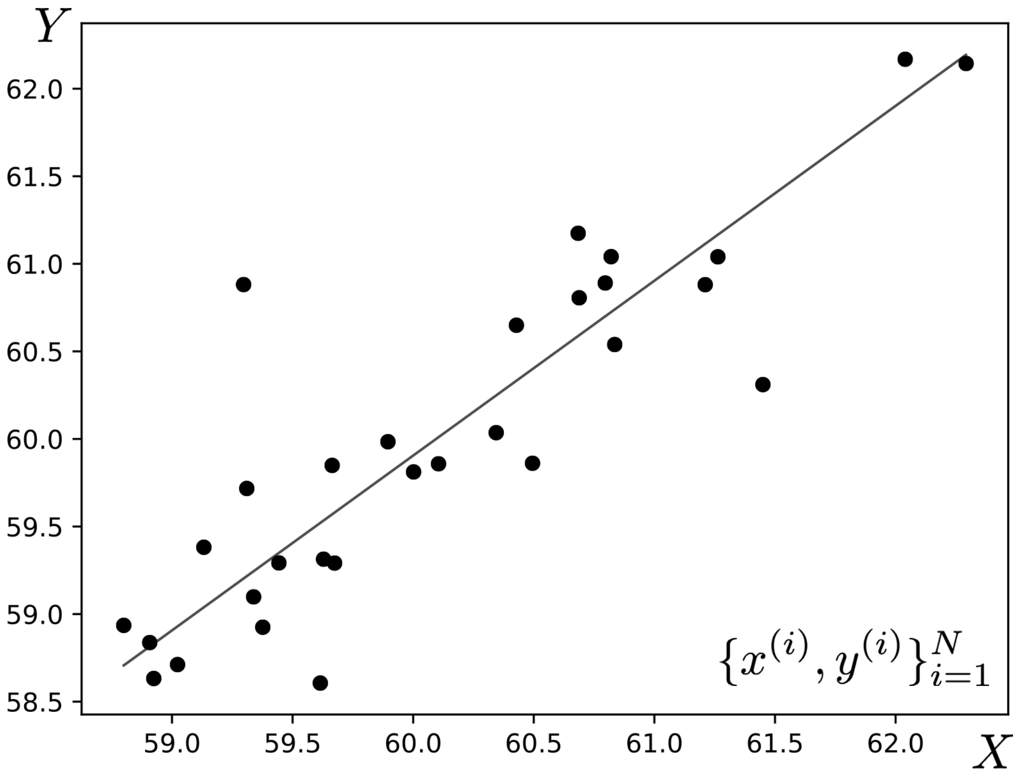


Training data
Y
X
$\{x^{(i)}, y^{(i)}\}_{i=1}^{N}$

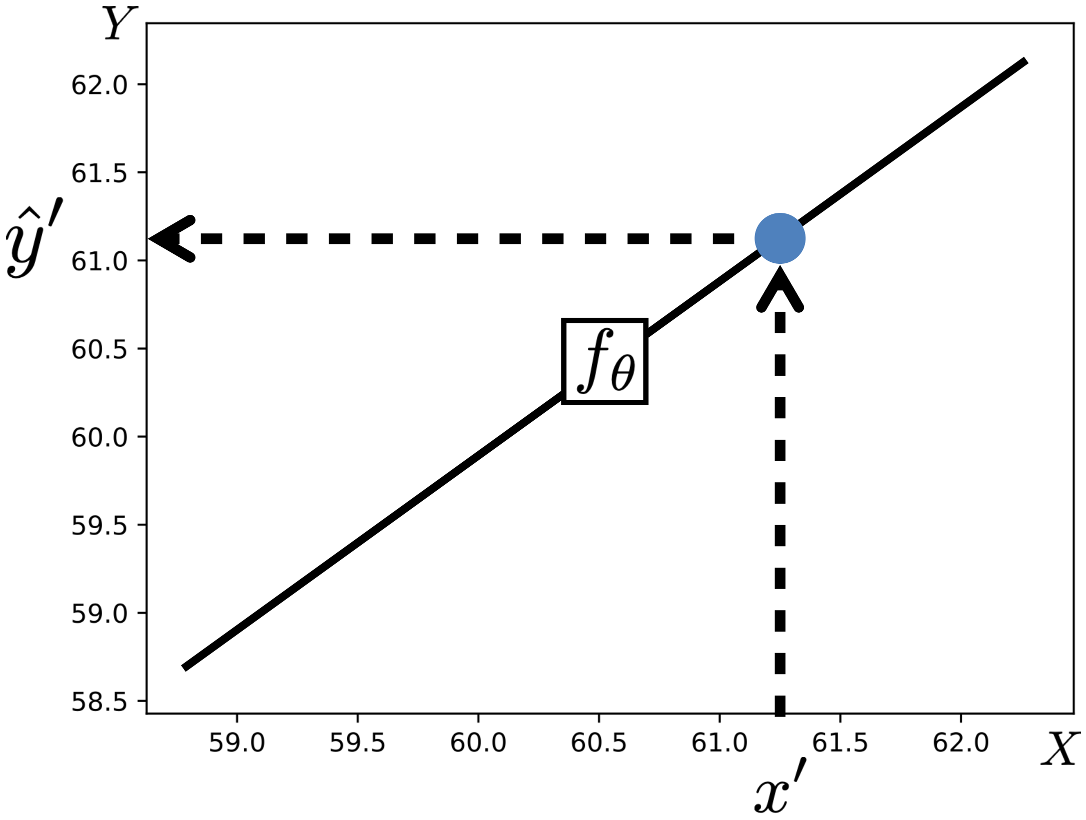


Test query
Y
X
?
y'
x'

Training data



Test query

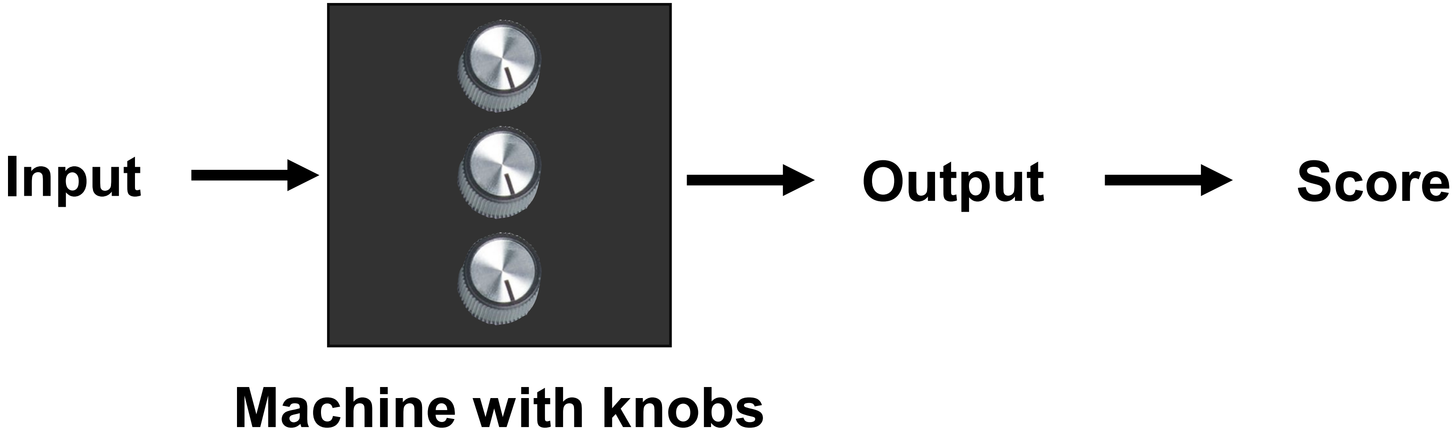


$$x' \dashrightarrow \boxed{f_\theta} \dashrightarrow \hat{y}'$$

How to minimize the objective w.r.t. θ?

$$\theta^* = \arg\min_{\theta} \sum_{i=1}^{N} (f_\theta(x^{(i)}) - y^{(i)})^2$$

Use an **optimizer**!



How to minimize the objective w.r.t. θ?

In the linear case:

Learning problem

$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} (\theta_1 x^{(i)} + \theta_0 - y^{(i)})^2$$

$$J(\theta) = \sum_{i=1}^{N} (\theta_1 x^{(i)} + \theta_0 - y^{(i)})^2 = (\mathbf{y} - \mathbf{X}\theta)^T(\mathbf{y} - \mathbf{X}\theta)$$

$$\mathbf{X} = \begin{pmatrix} x^{(1)} & 1 \\ x^{(2)} & 1 \\ \vdots & \vdots \\ x^{(N)} & 1 \end{pmatrix} \qquad \theta = \begin{pmatrix} \theta_1 & \theta_0 \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{pmatrix}$$

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

$$\frac{\partial J(\theta)}{\partial \theta} = 0$$

$$\frac{\partial J(\theta)}{\partial \theta} = 2(\mathbf{X}^T\mathbf{X}\theta - \mathbf{X}^T\mathbf{y})$$

$$2(\mathbf{X}^T\mathbf{X}\theta^* - \mathbf{X}^T\mathbf{y}) = 0$$

$$\mathbf{X}^T\mathbf{X}\theta^* = \mathbf{X}^T\mathbf{y}$$

Solution

$$\boxed{\theta^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}}$$

Empirical Risk Minimization

(formalization of supervised learning)

Linear least squares learning problem

$$\boxed{\theta^* = \arg\min_{\theta} \sum_{i=1}^{N} (\theta_1 x^{(i)} + \theta_0 - y^{(i)})^2}$$

Empirical Risk Minimization

(formalization of supervised learning)

Objective function
(loss)

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

Hypothesis space

Training data

Case study #1: Linear least squares

Data

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

Learner

Objective

$$\mathcal{L}(f_\theta(x), y) = (f_\theta(x) - y)^2$$

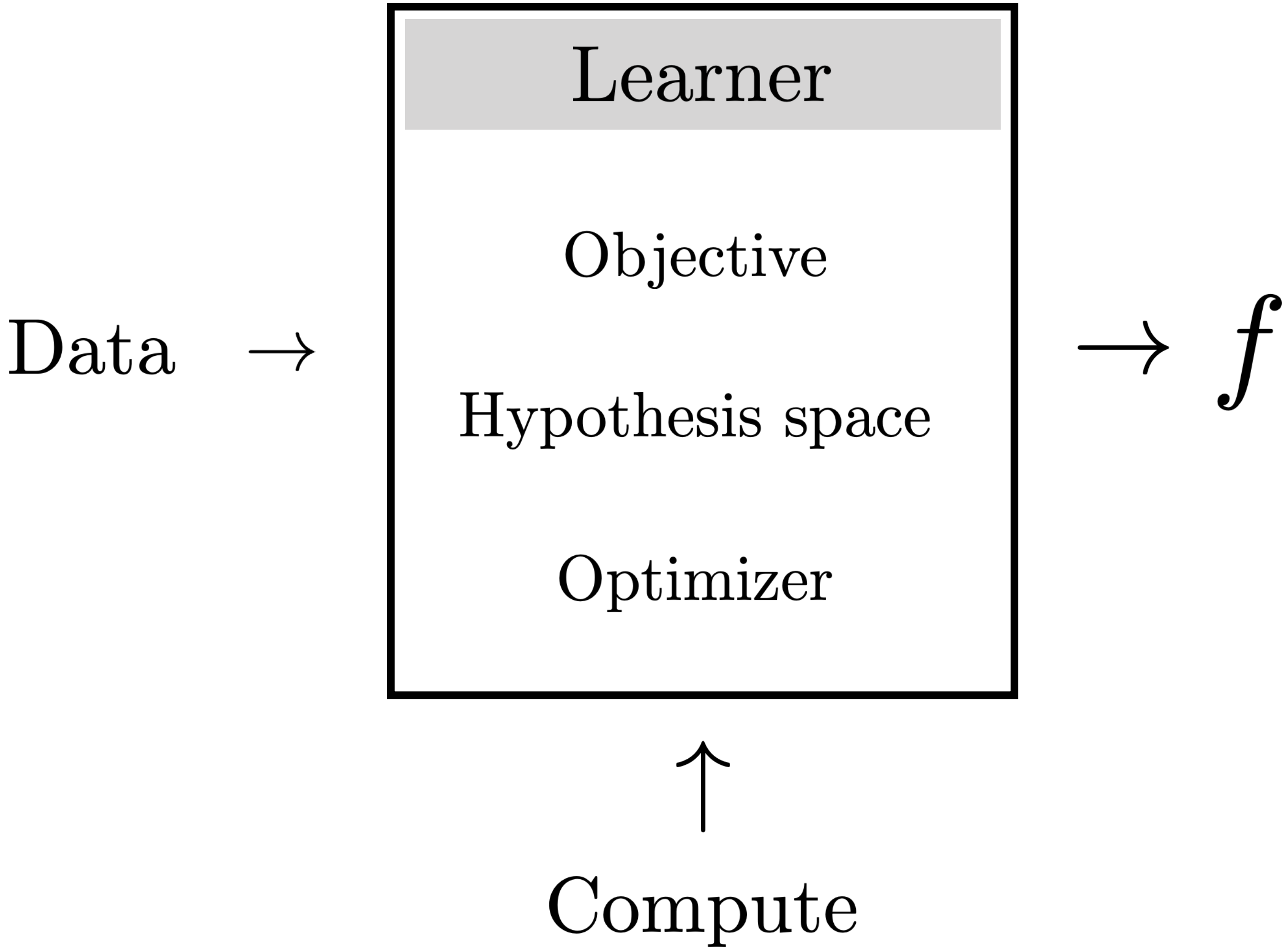
Hypothesis space

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

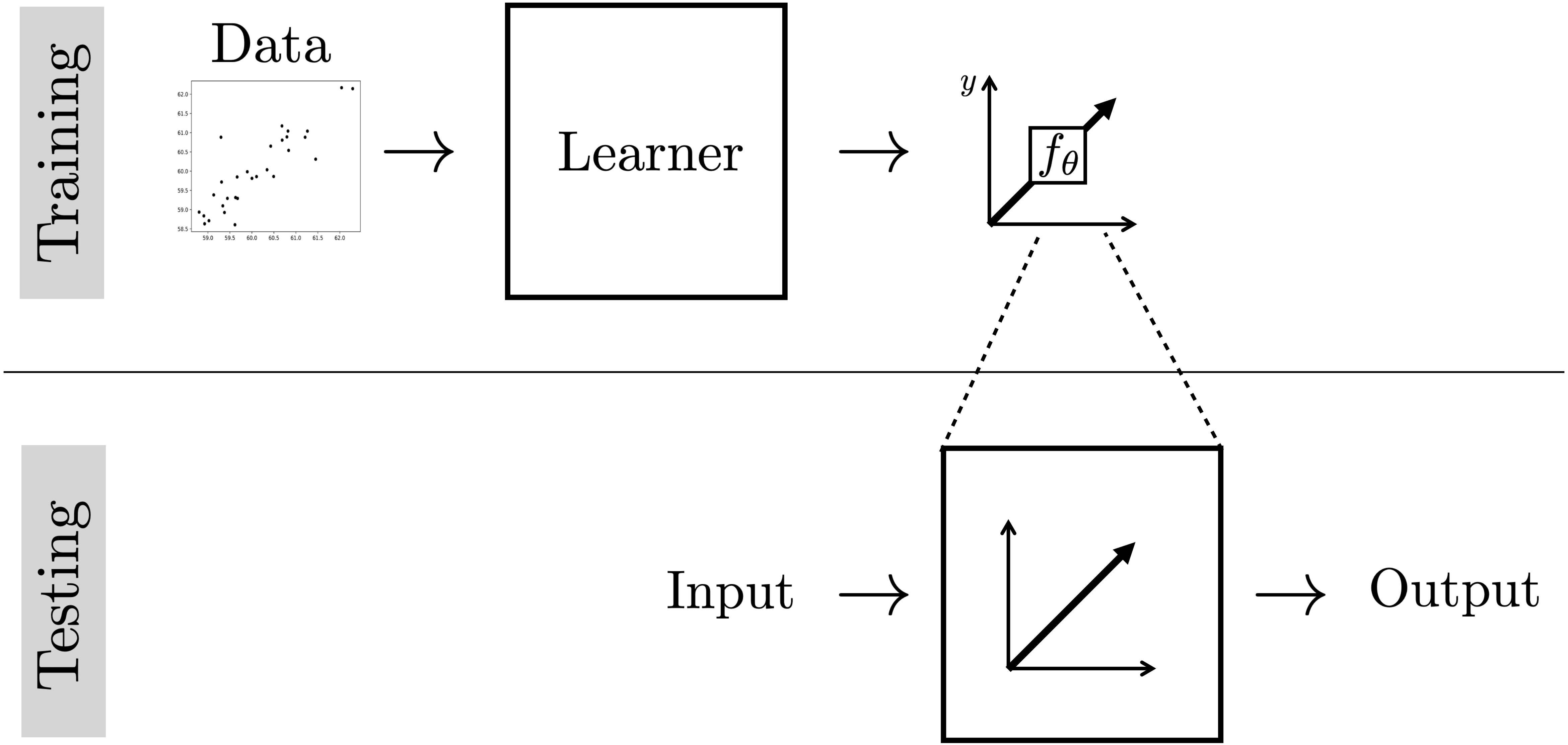
Optimizer

$$\theta^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

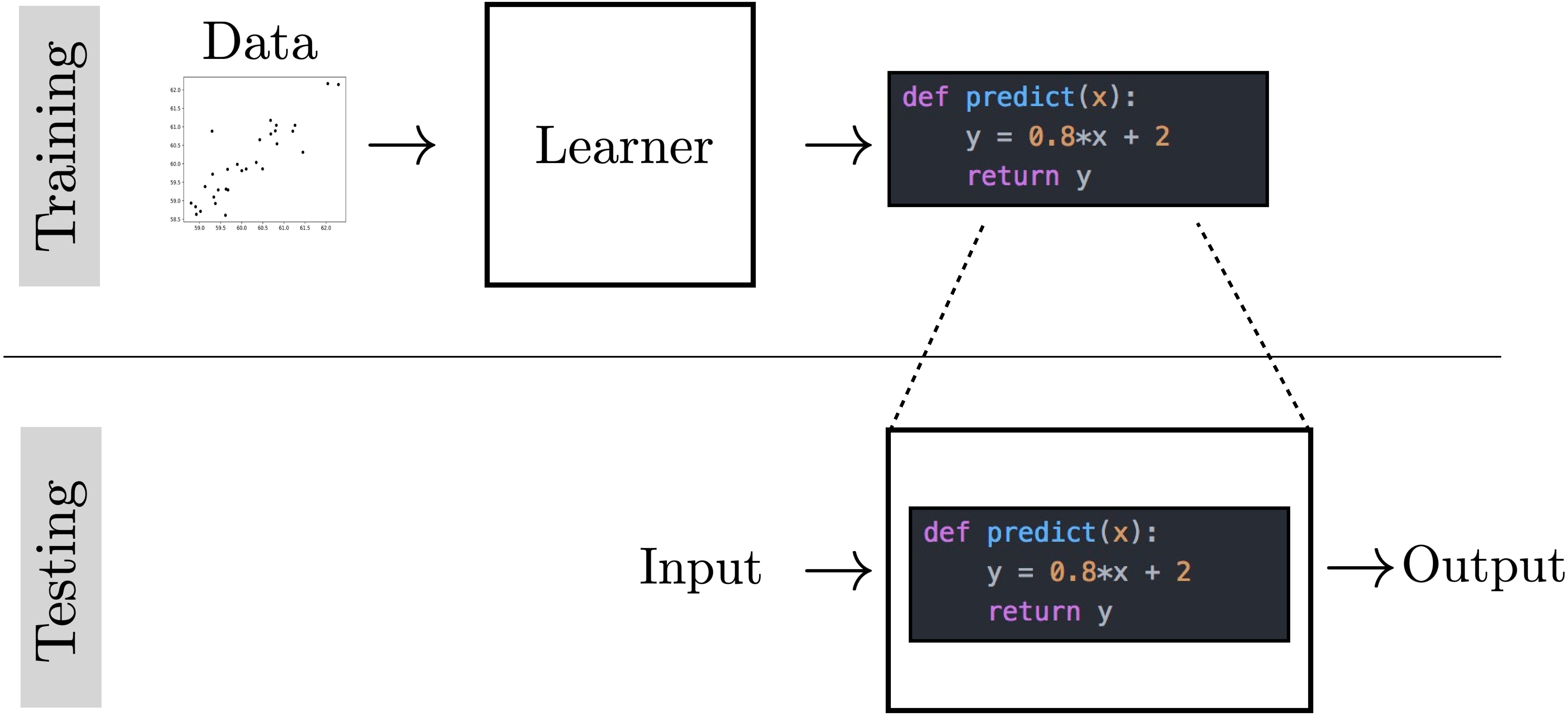
$\rightarrow f$


Learner
Objective
Hypothesis space
Optimizer
Data
f
Compute

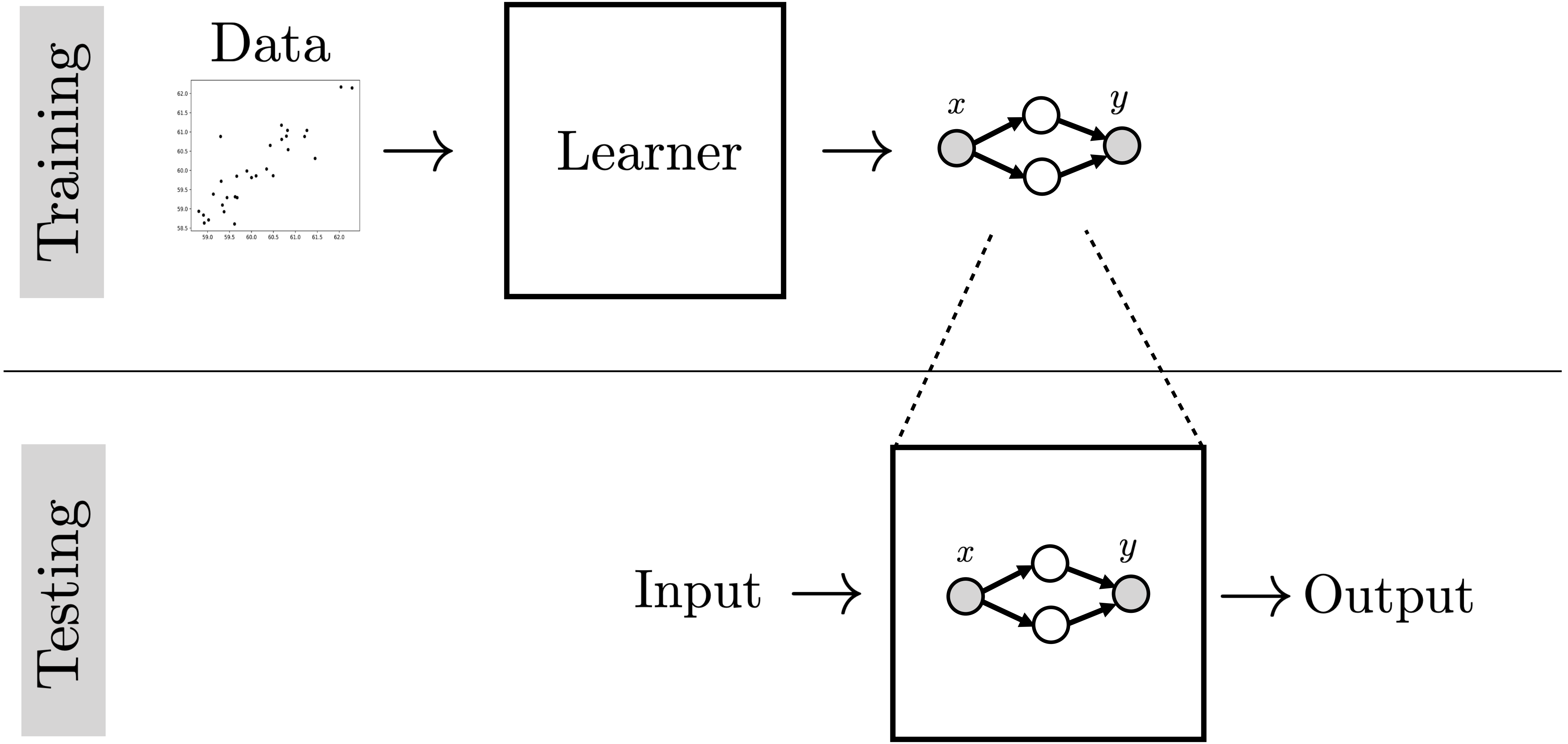
Example 1: Linear least squares



Example 2: Program induction



Example 3: Deep learning



Learning for vision

Big questions:

1. How do you represent the input and output?

2. What is the objective?

3. What is the hypothesis space? (e.g., linear, polynomial, neural net?)

4. How do you optimize? (e.g., gradient descent, Newton's method?)

5. What data do you train on?

Image classification

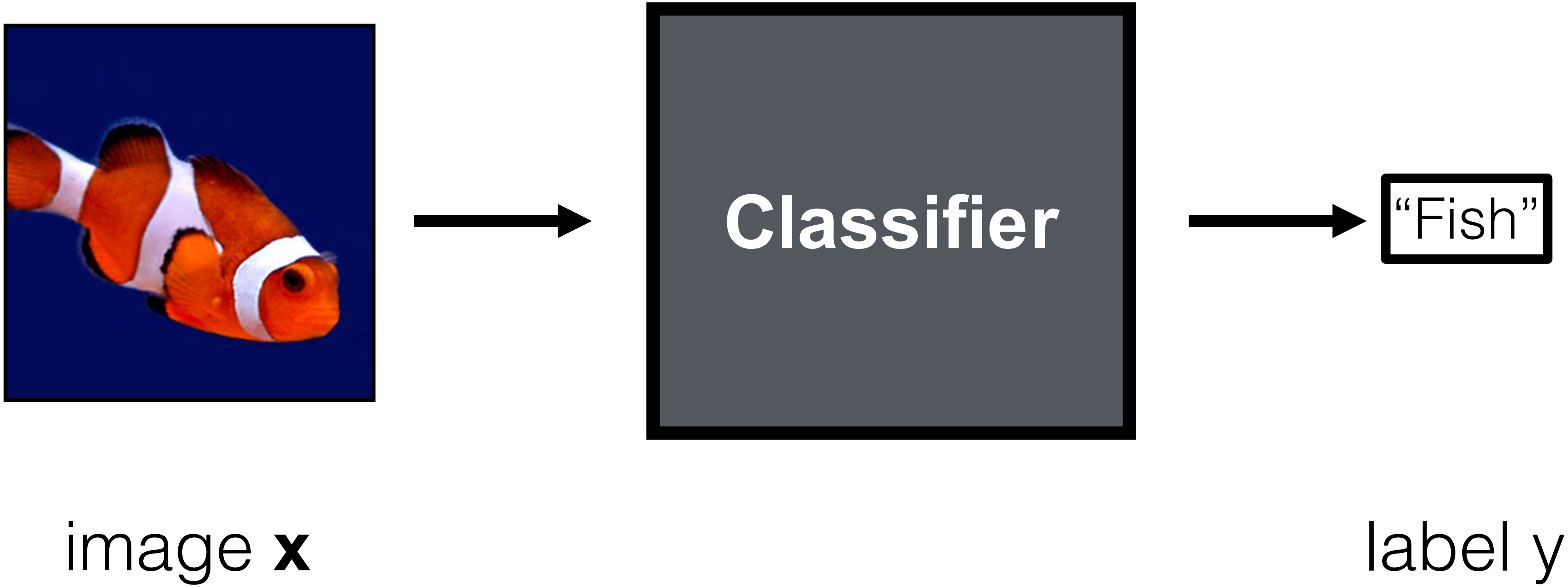


Image classification

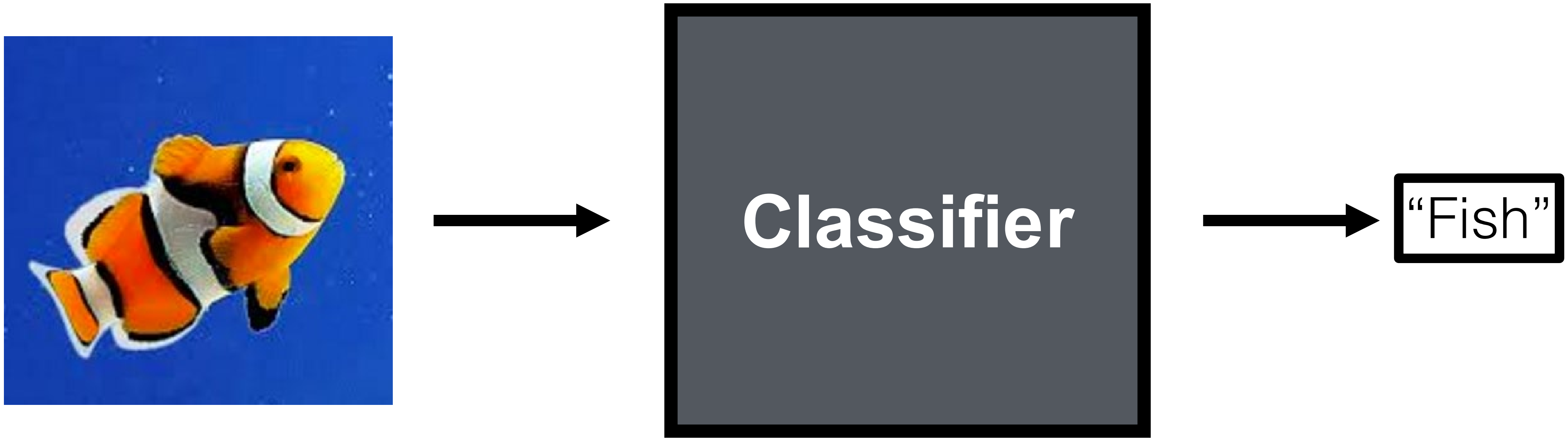


image **x**

label y

Image classification

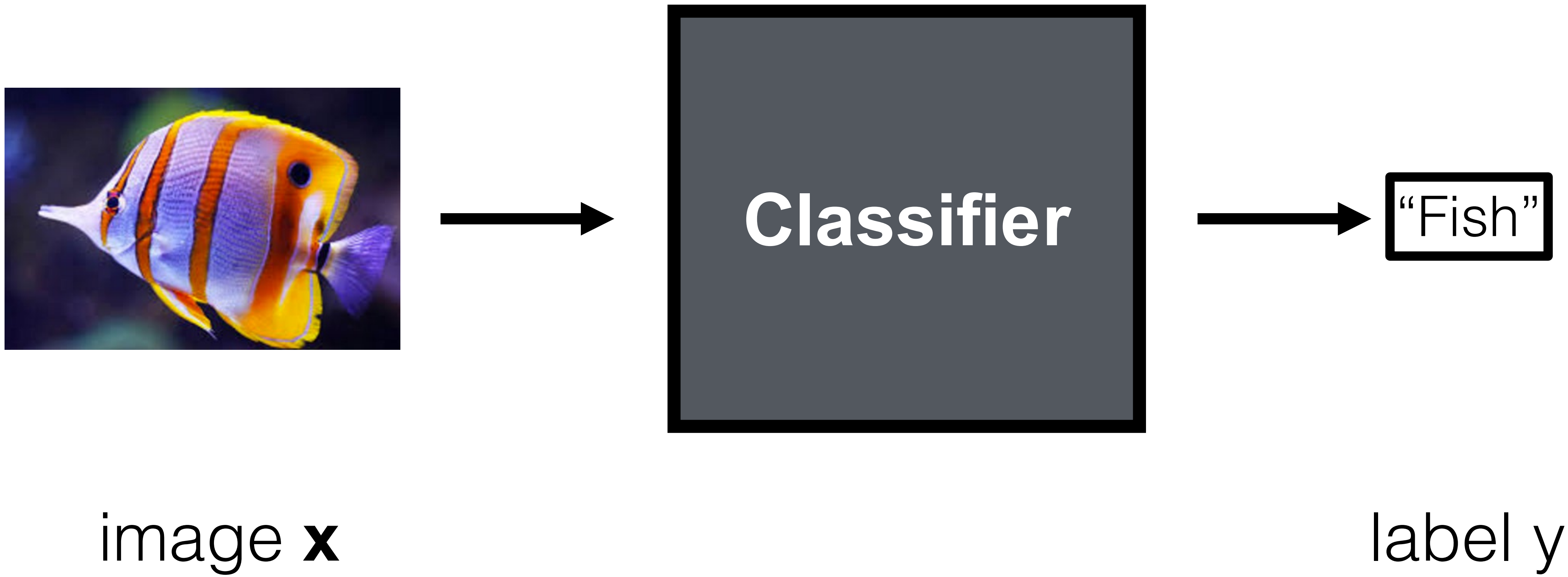
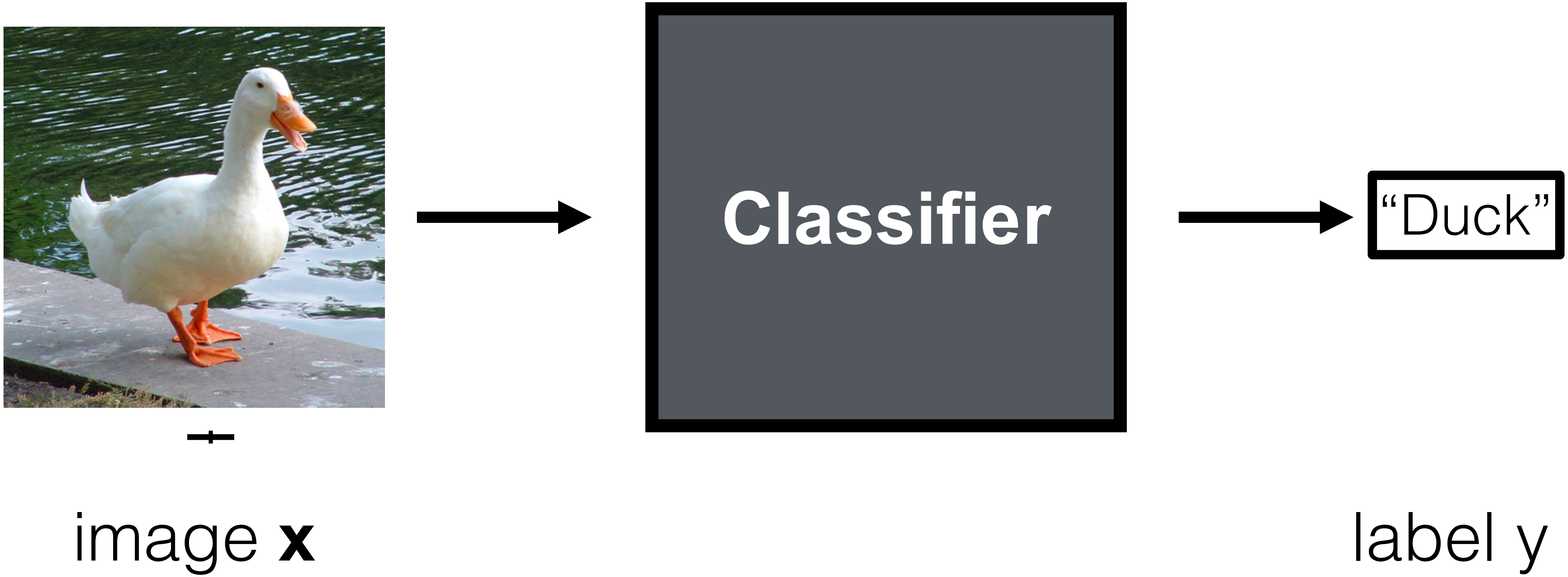
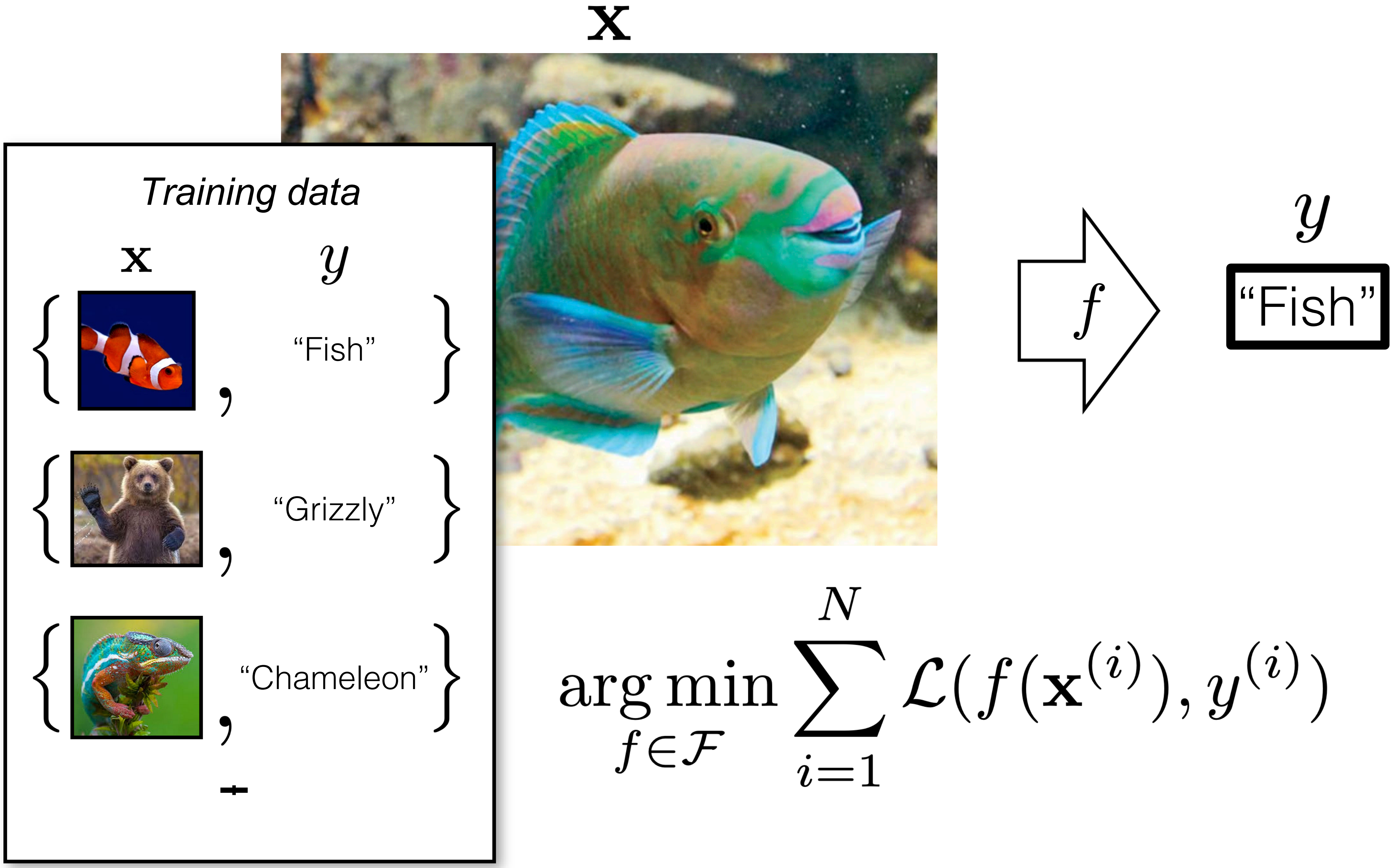


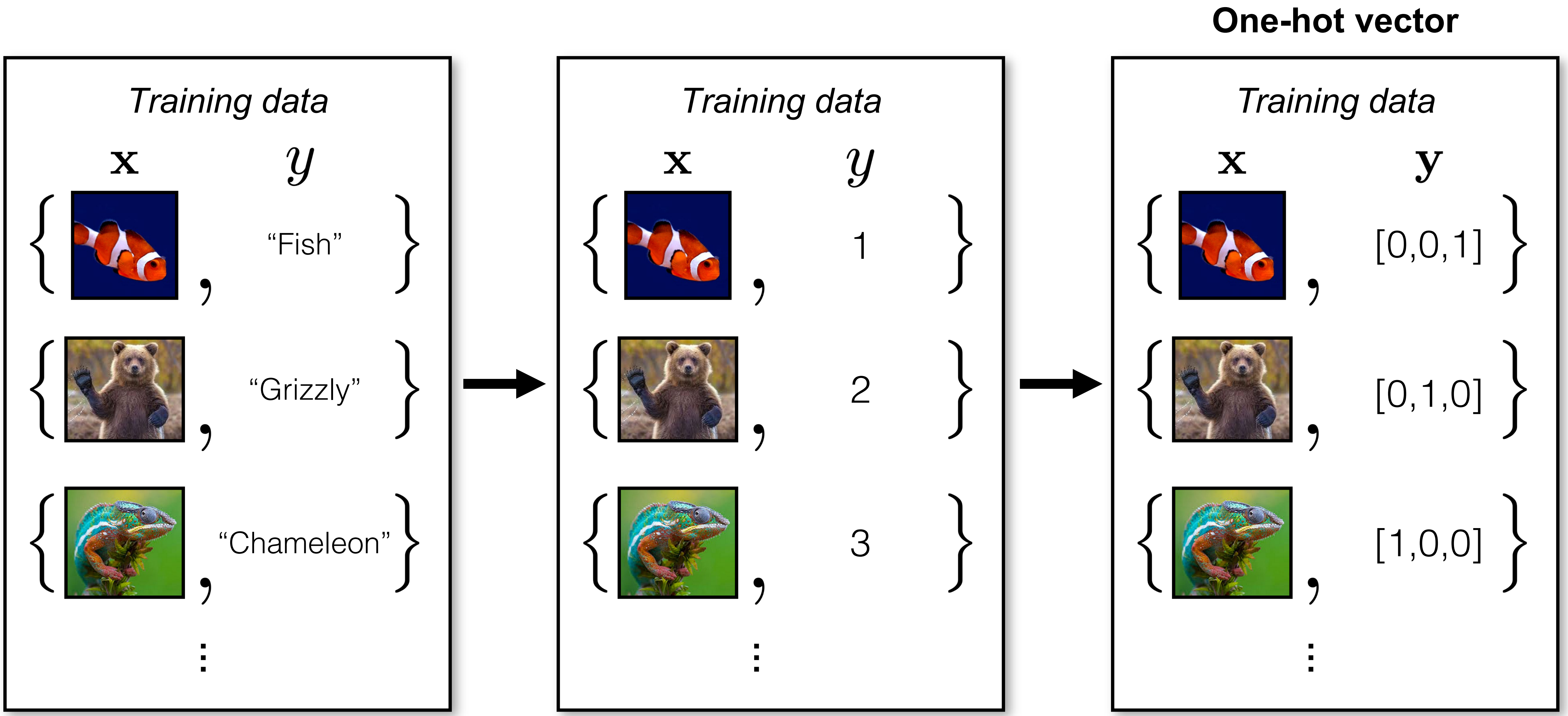
Image classification





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

How to represent class labels?



What should the loss be?

0-1 loss (number of misclassifications)

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = \mathbb{1}(\hat{\mathbf{y}} = \mathbf{y}) \longleftarrow$$ discrete, NP-hard to optimize!

Cross entropy

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

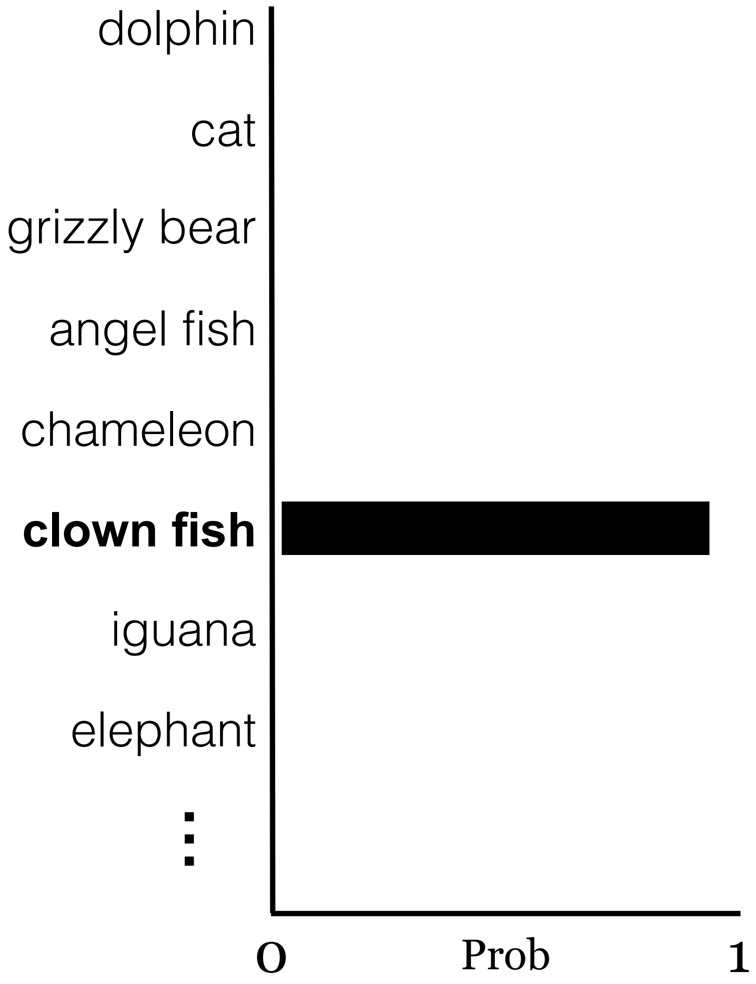
Ground truth label $\mathbf{y}$

$\mathbf{x}$

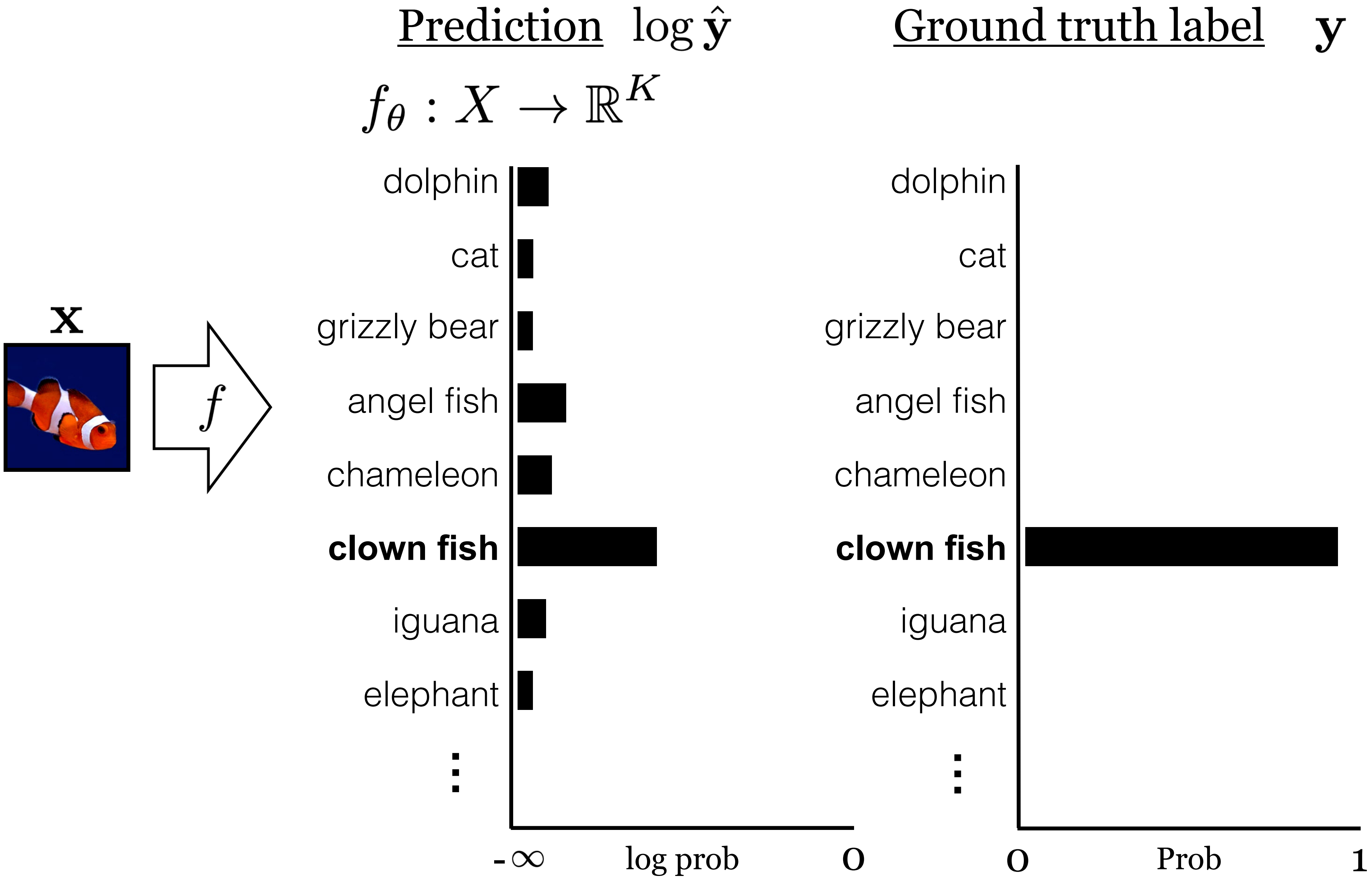
[0,0,0,0,0,1,0,0,…]

Ground truth label $\mathbf{y}$

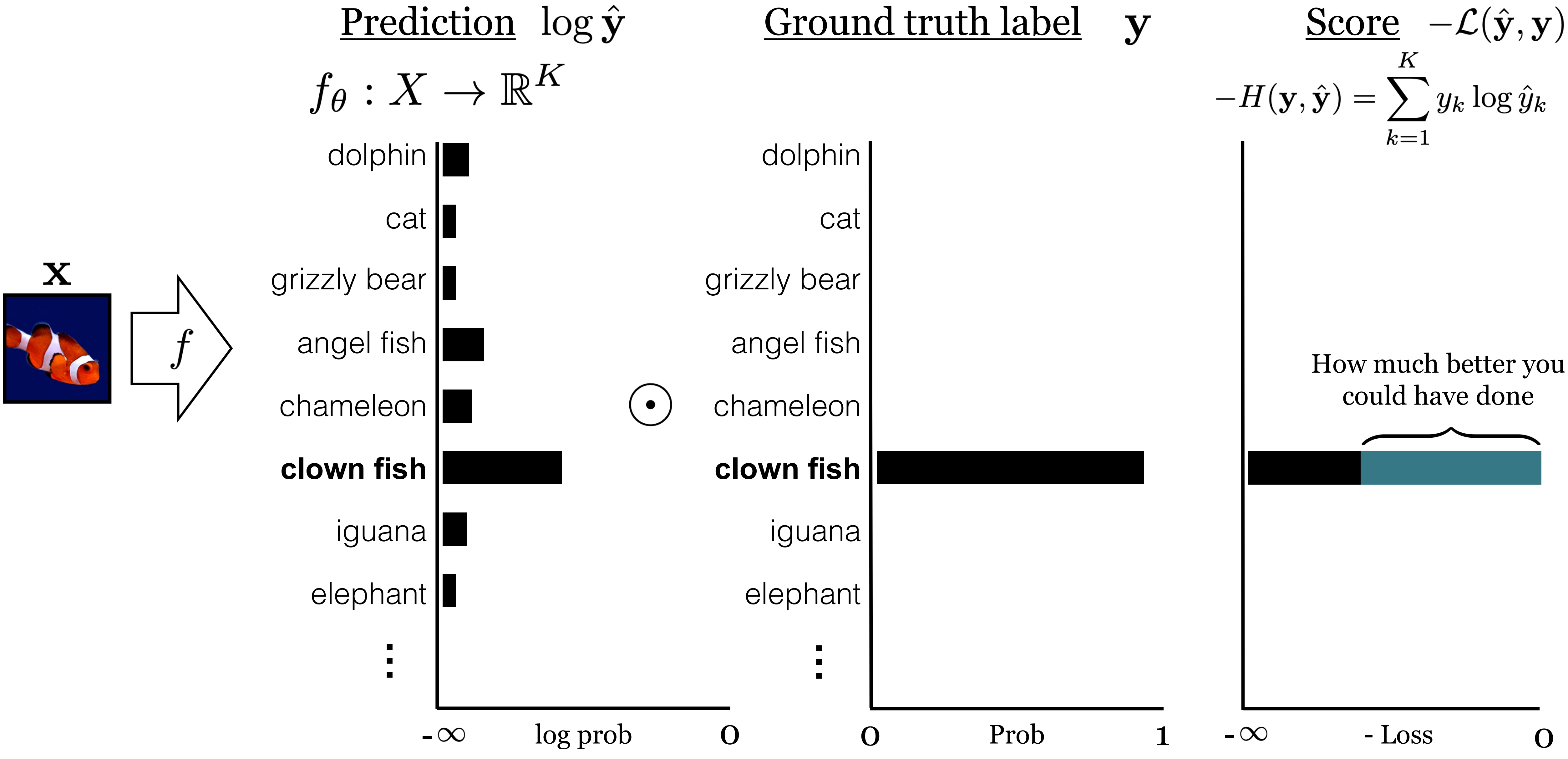
$\mathbf{x}$

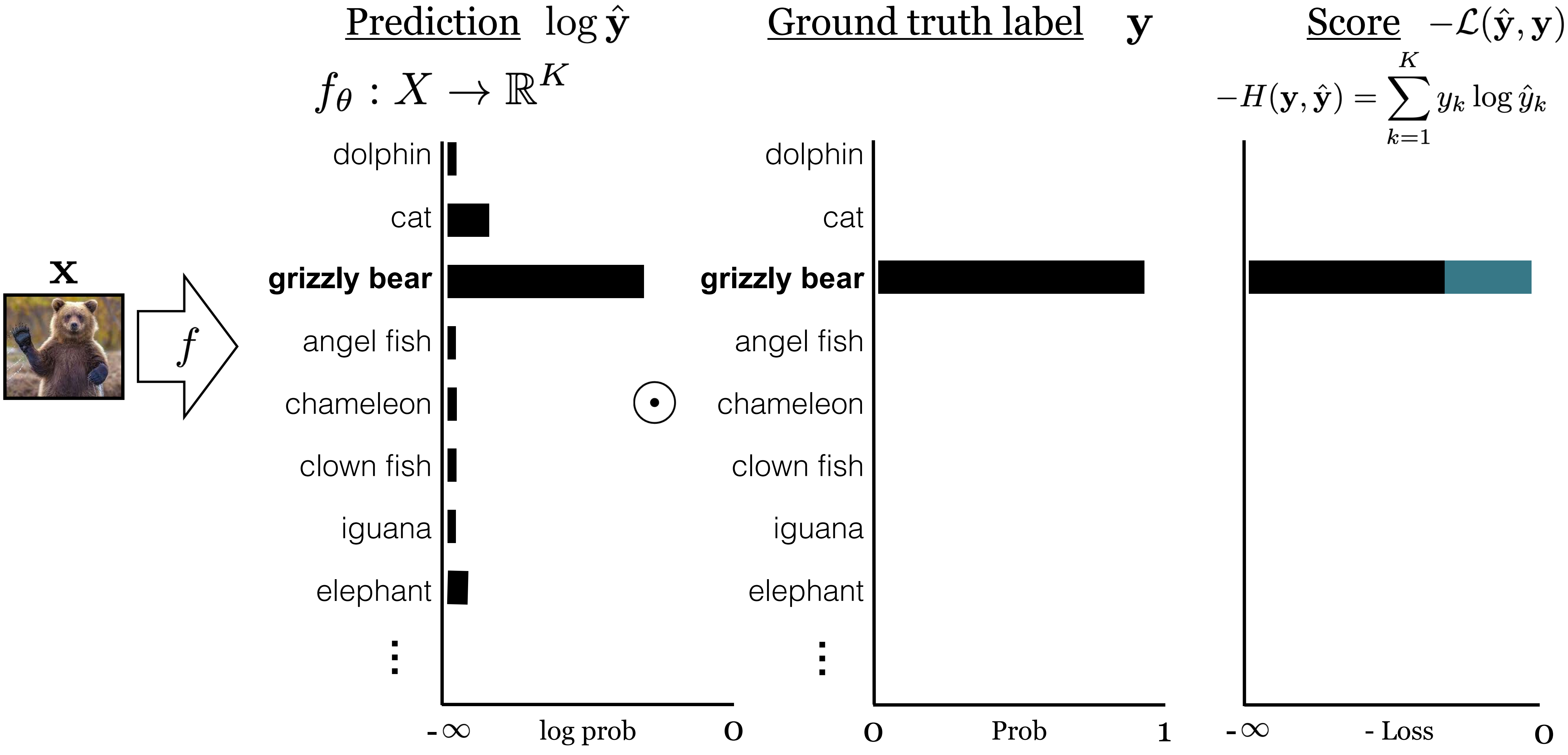


∞

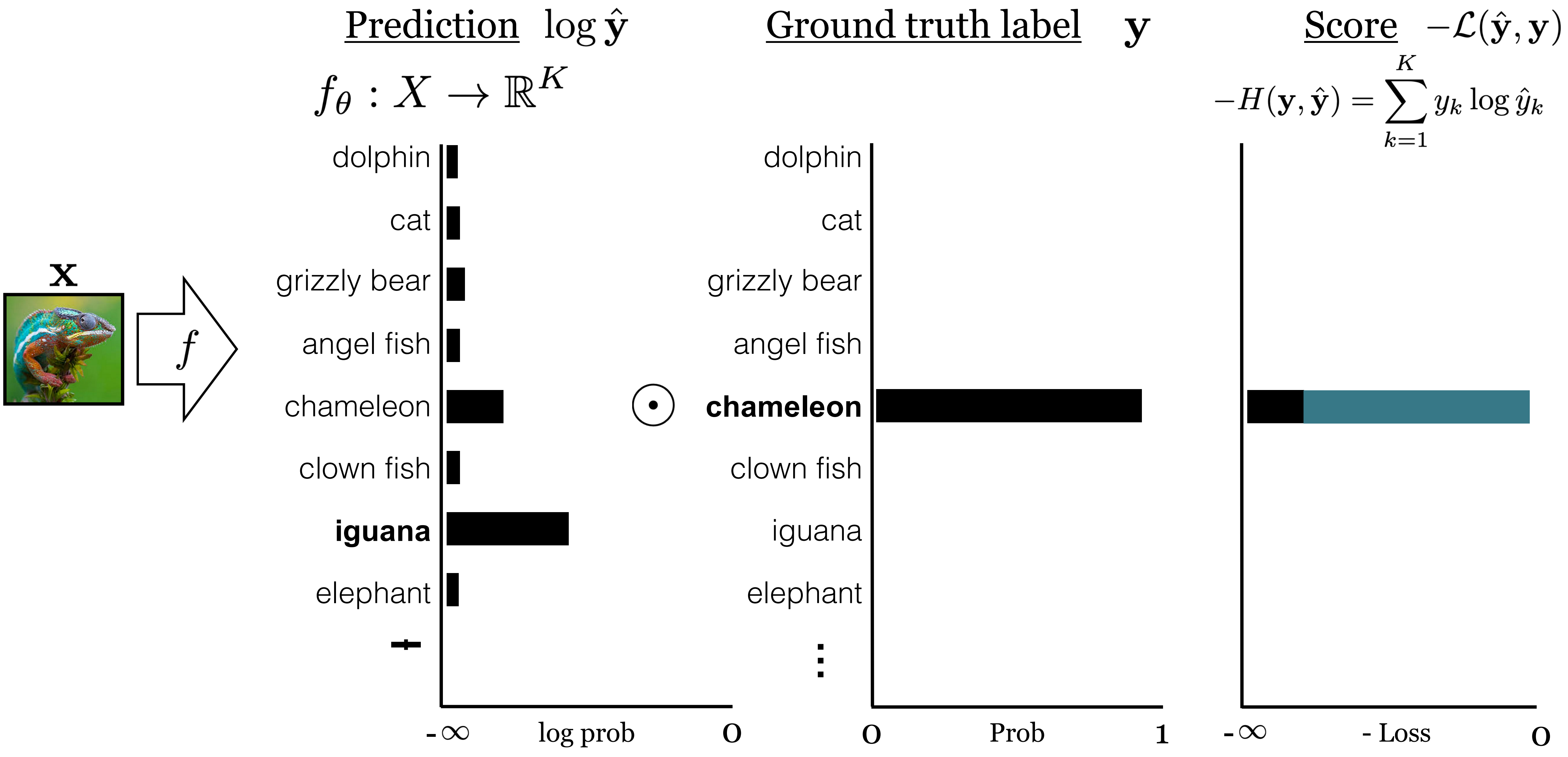


Prediction $\log \hat{\mathbf{y}}$
Ground truth label $\mathbf{y}$
$f_\theta : X \to \mathbb{R}^K$
$\mathbf{x}$
f
dolphin
cat
grizzly bear
angel fish
chameleon
clown fish
iguana
elephant
⋮
-∞
log prob
0
dolphin
cat
grizzly bear
angel fish
chameleon
clown fish
iguana
elephant
⋮
0
Prob
1


Prediction $\log \hat{\mathbf{y}}$
Ground truth label $\mathbf{y}$
Score $-\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$
$f_\theta : X \to \mathbb{R}^K$
$-H(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{k=1}^{K} y_k \log \hat{y}_k$
$\mathbf{x}$
f
dolphin
cat
grizzly bear
angel fish
chameleon
clown fish
iguana
elephant
$\odot$
How much better you could have done
-∞
log prob
0
Prob
1
- Loss



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



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

Softmax regression (a.k.a. multinomial logistic regression)

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

$$\mathbf{z} = f_\theta(\mathbf{x})$$

logits: vector of K scores, one for each class

$$\hat{\mathbf{y}} = \mathtt{softmax}(\mathbf{z})$$

squash into a non-negative vector that sums to 1 — i.e. **a probability mass function!**

$$\hat{y}_j = \frac{e^{-z_j}}{\sum_{k=1}^{K} e^{-z_k}}$$

$$\hat{\mathbf{y}} =$$

Softmax regression (a.k.a. multinomial logistic regression)

Probabilistic interpretation:

$$\hat{\mathbf{y}} \equiv [P_\theta(Y = 1|X = \mathbf{x}), \dots, P_\theta(Y = K|X = \mathbf{x})]$$ ← predicted probability of each class given input **x**

$$H(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$$ ← picks out the -log likelihood of the ground truth class $\mathbf{y}$ under the model prediction $\hat{\mathbf{y}}$

$$f^* = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{N} H(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)})$$ ← max likelihood learner!

Softmax regression (a.k.a. multinomial logistic regression)

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

$$\mathbf{z} = f_\theta(\mathbf{x})$$

$$\hat{\mathbf{y}} = \texttt{softmax}(\mathbf{z})$$

Data

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

Learner

Objective

$$\mathcal{L}(\mathbf{y}, f_\theta(\mathbf{x})) = H(\mathbf{y}, \texttt{softmax}(f_\theta(\mathbf{x})))$$

$$\to f$$

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**.

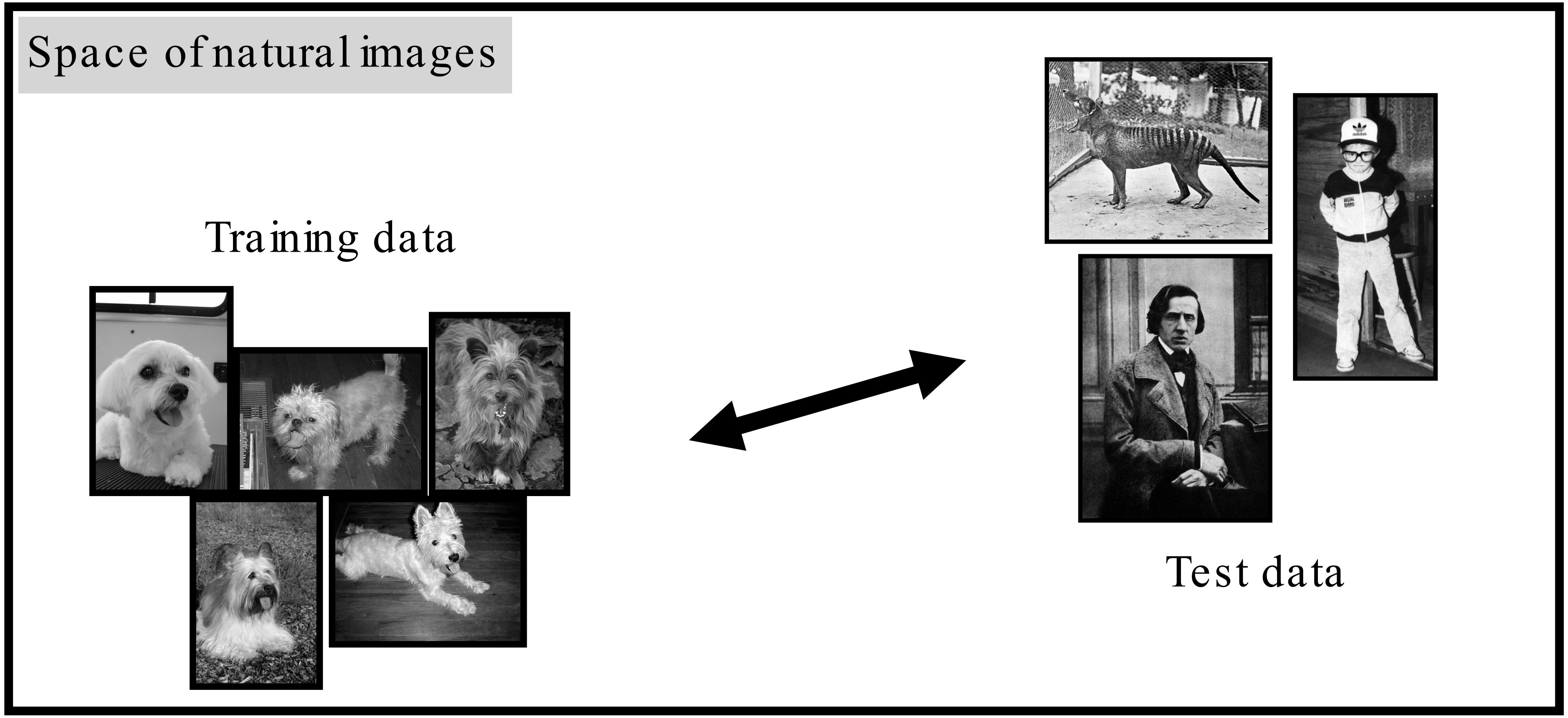
… [this is what] separates machine learning from optimization.”

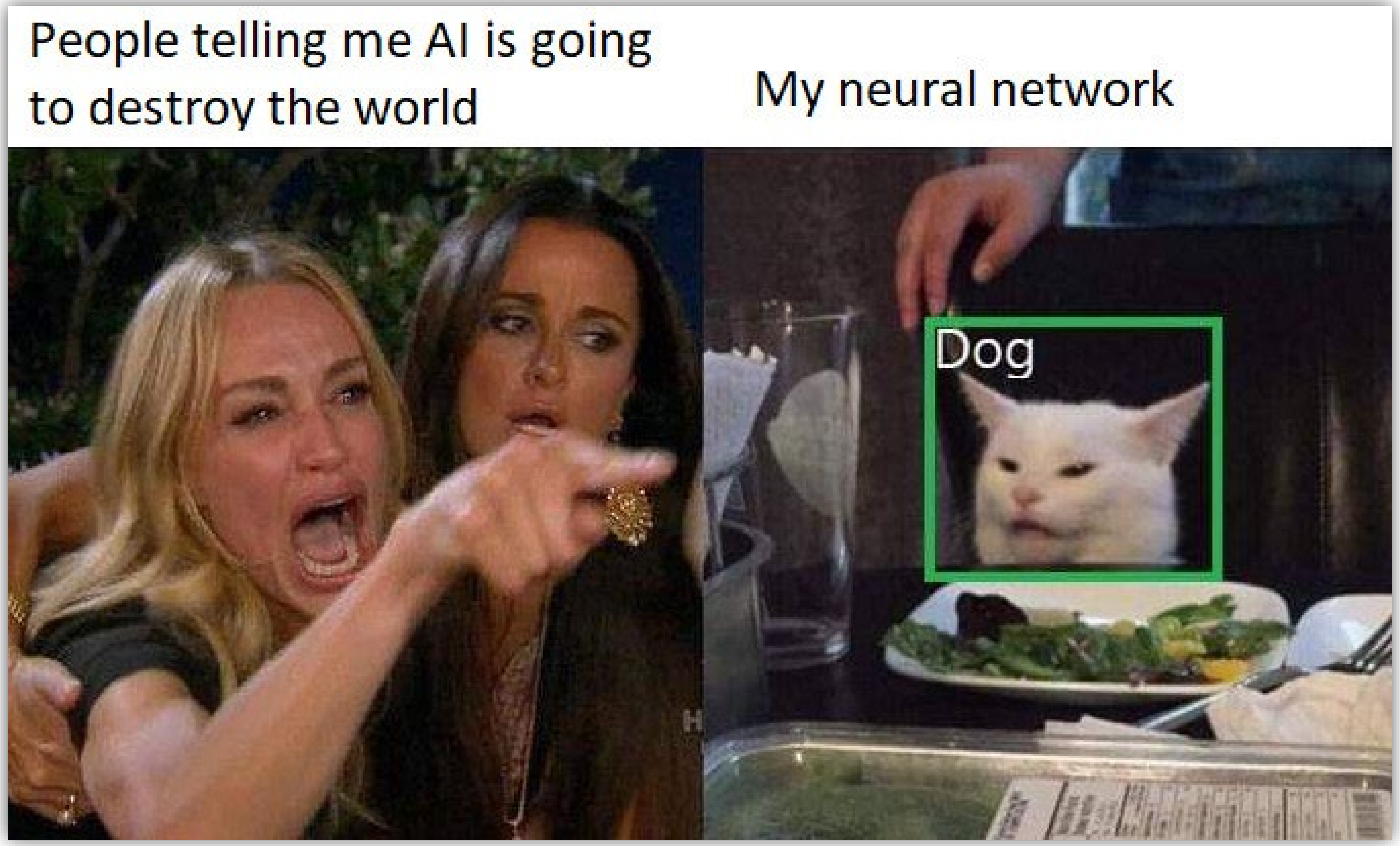
— Deep Learning textbook (Goodfellow et al.)

training domain

testing domain
(where we actual use our model)

Domain gap between $p_{\texttt{train}}$ and $p_{\texttt{test}}$
will cause us to fail to generalize.




People telling me AI is going to destroy the world
My neural network
Dog

Parametric Approach

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

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

Image

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

$\longrightarrow$ f(**x**,**W**) $\longrightarrow$ **10** numbers giving class scores

W

parameters
or weights

Parametric Approach: Linear Classifier

3072x1

$$\boxed{f(x,W)} = \boxed{W}\boxed{x}$$

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

$$\underbrace{\boxed{f(x,W)}}_{10\text{x}1} = \underbrace{\boxed{W}}_{10\text{x}3072}\overbrace{\boxed{x}}^{3072\text{x}1} + \boxed{b}\ \ 10\text{x}1$$

Image

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

$$\longrightarrow f(\mathbf{x}, \mathbf{W}) \longrightarrow$$

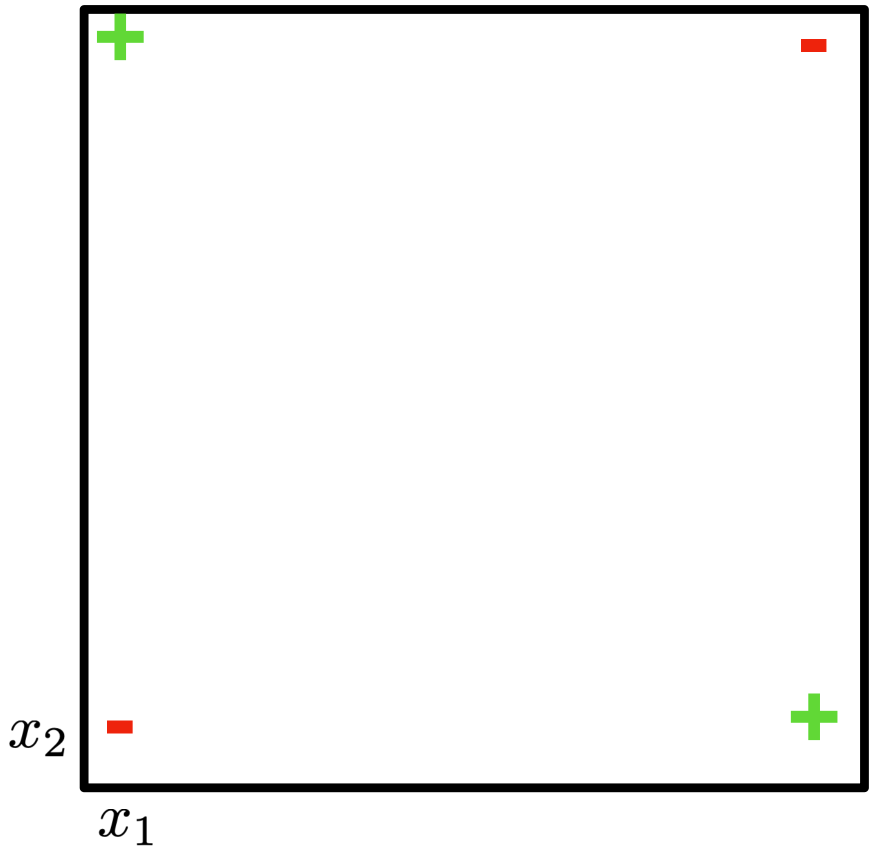
10 numbers giving
class scores

$$\uparrow$$

W

parameters
or weights

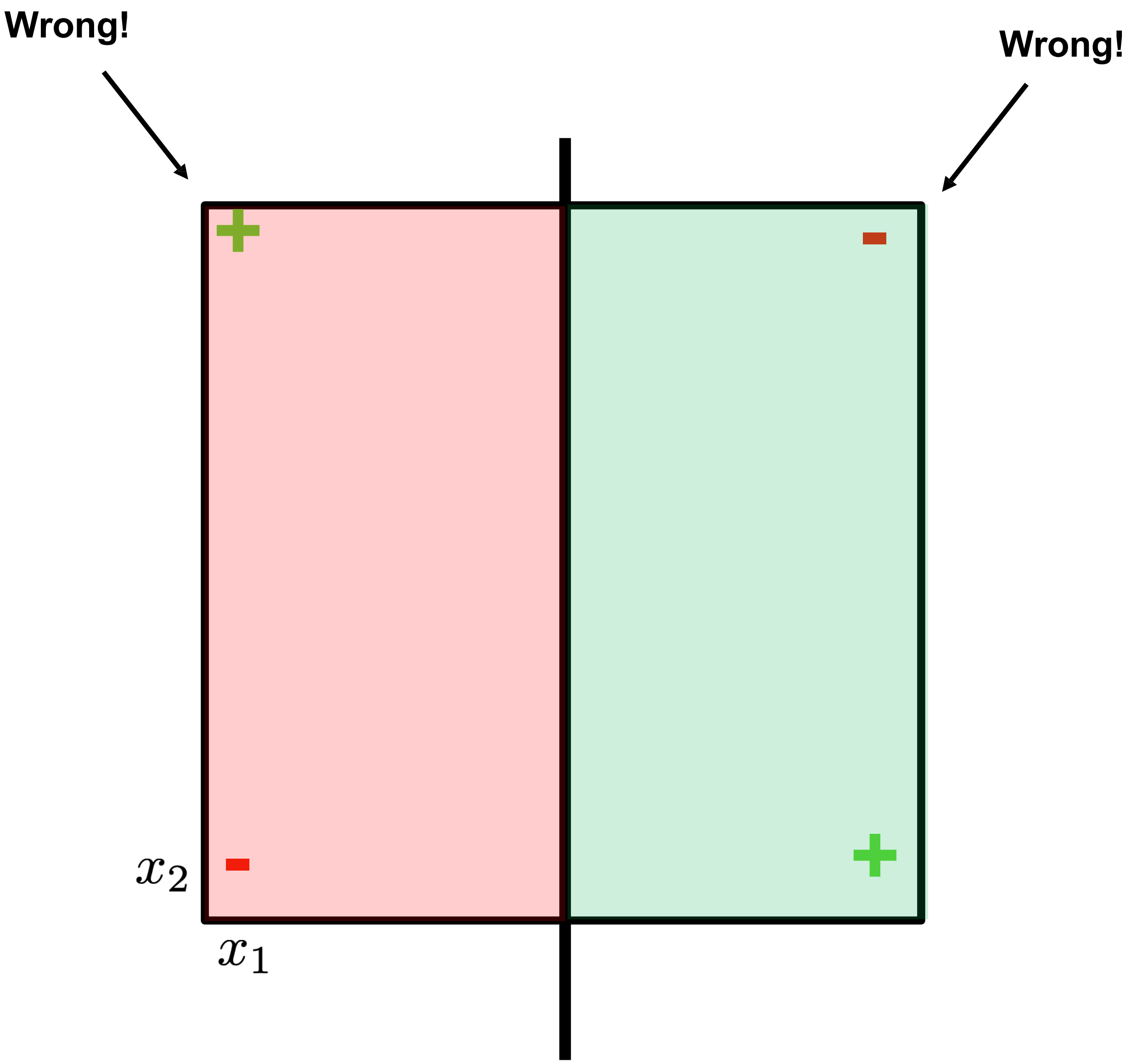
Limitations to linear classifiers



		x_2	
		0	1
x_1	0	0	1
	1	1	0

XOR

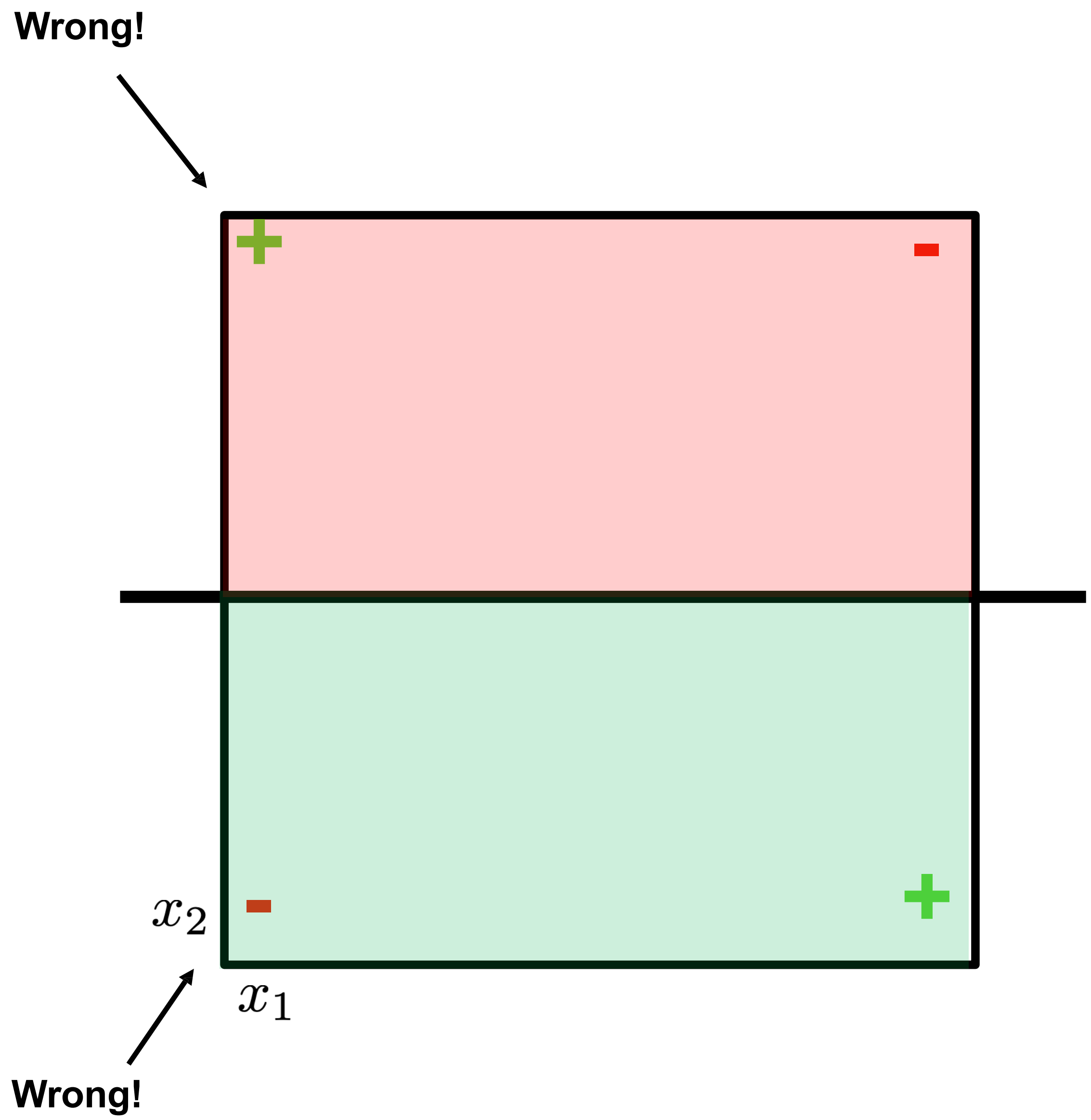
Limitations to linear classifiers



		x_2	
		0	1
x_1	0	0	1
	1	1	0

XOR

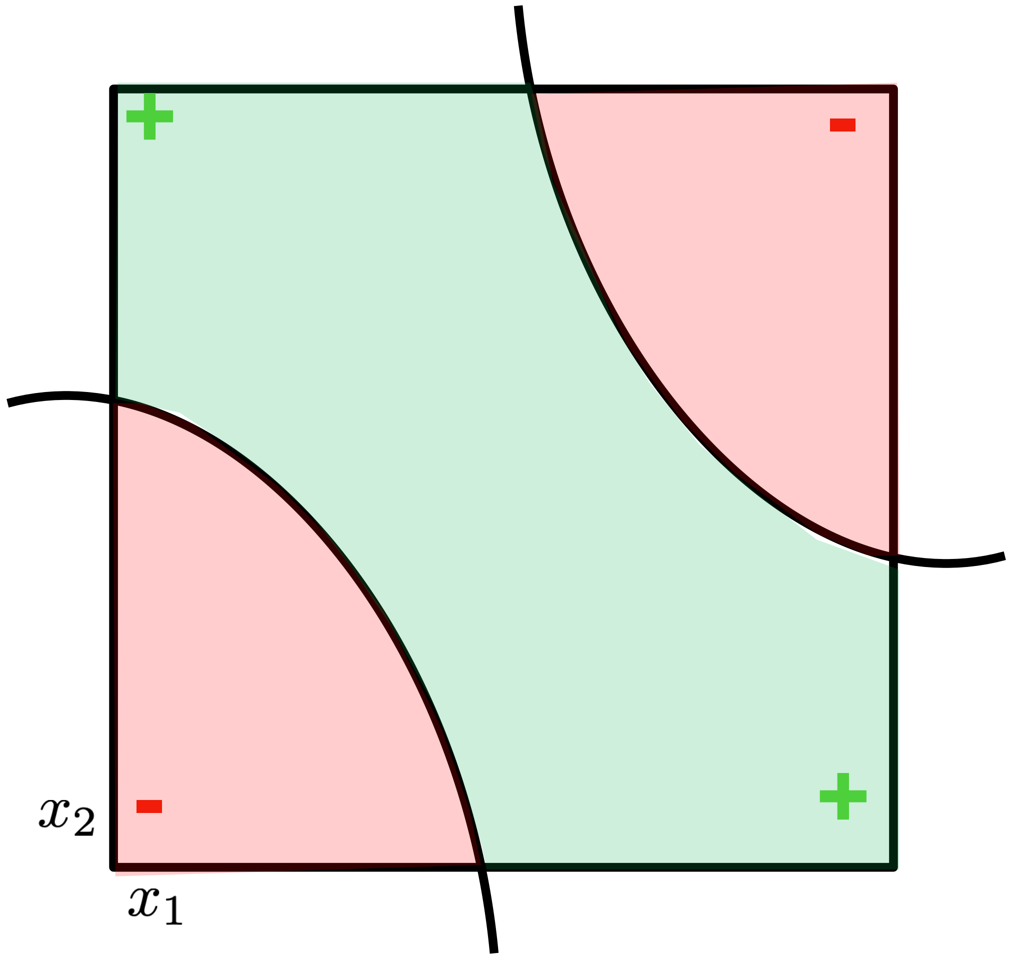
Limitations to linear classifiers



		x_2	
		0	1
x_1	0	0	1
	1	1	0

XOR

Goal: Non-linear decision boundary

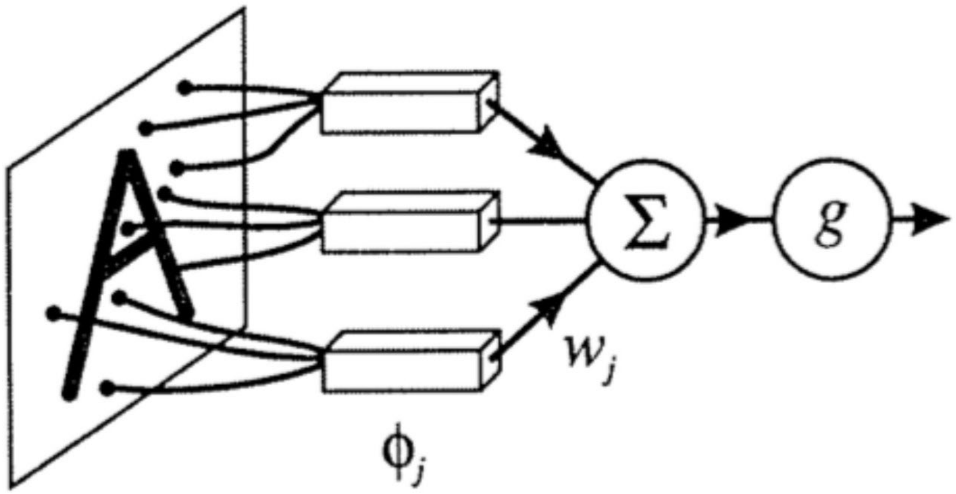


		x_2	
		0	1
x_1	0	0	1
	1	1	0

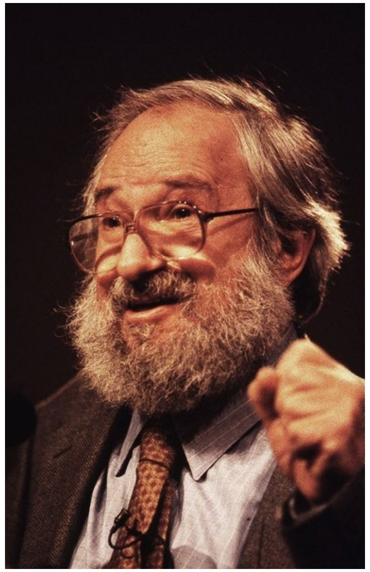
XOR

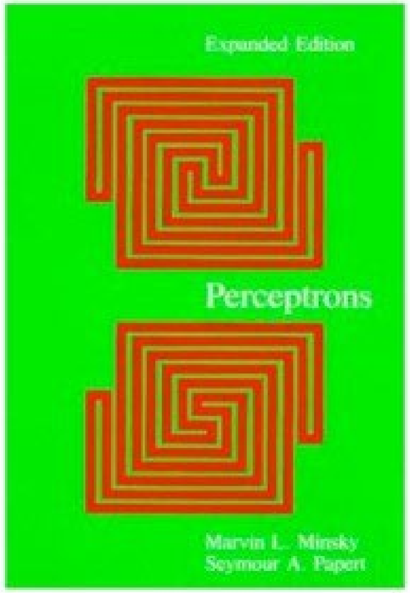
Perceptron

- 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.

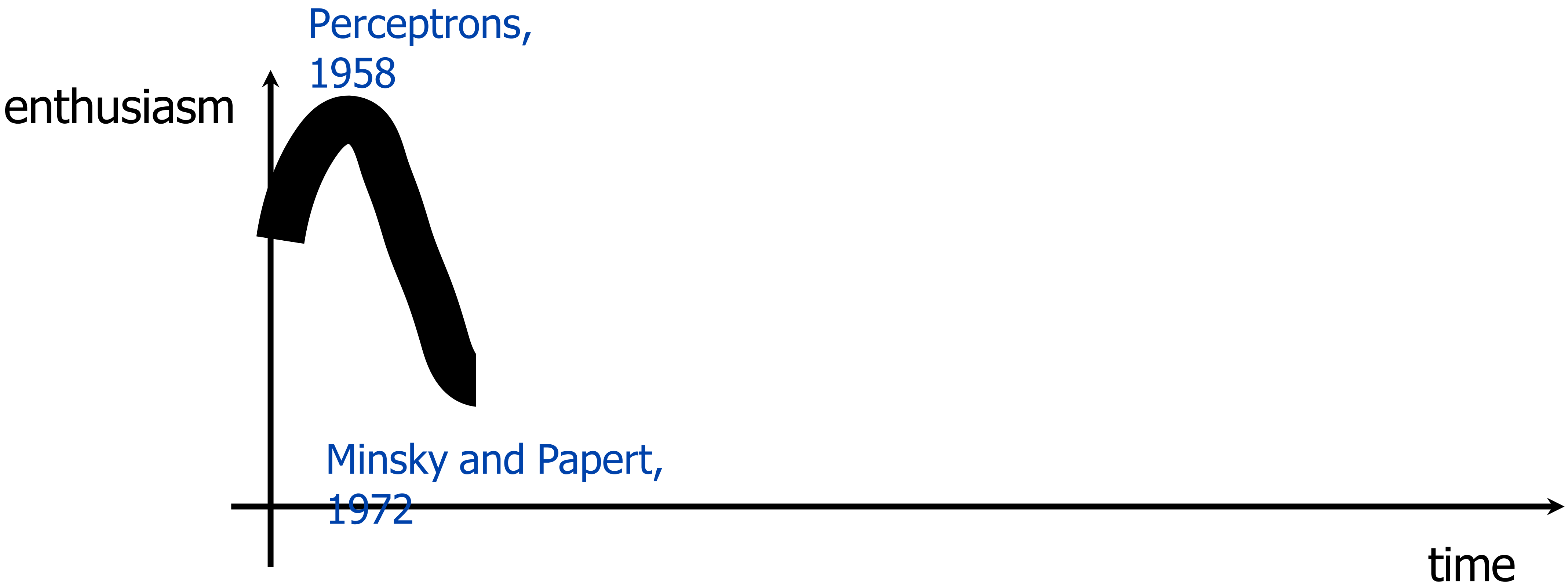
Artificial-intelligence research, which for a time concentrated on the programming of ton Neumann computers, is swinging back to the idea that intelligence might emerge from the activity of networks of neuronlike entities. Minsky and Papert's book was the first example of a mathematical analysis carried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

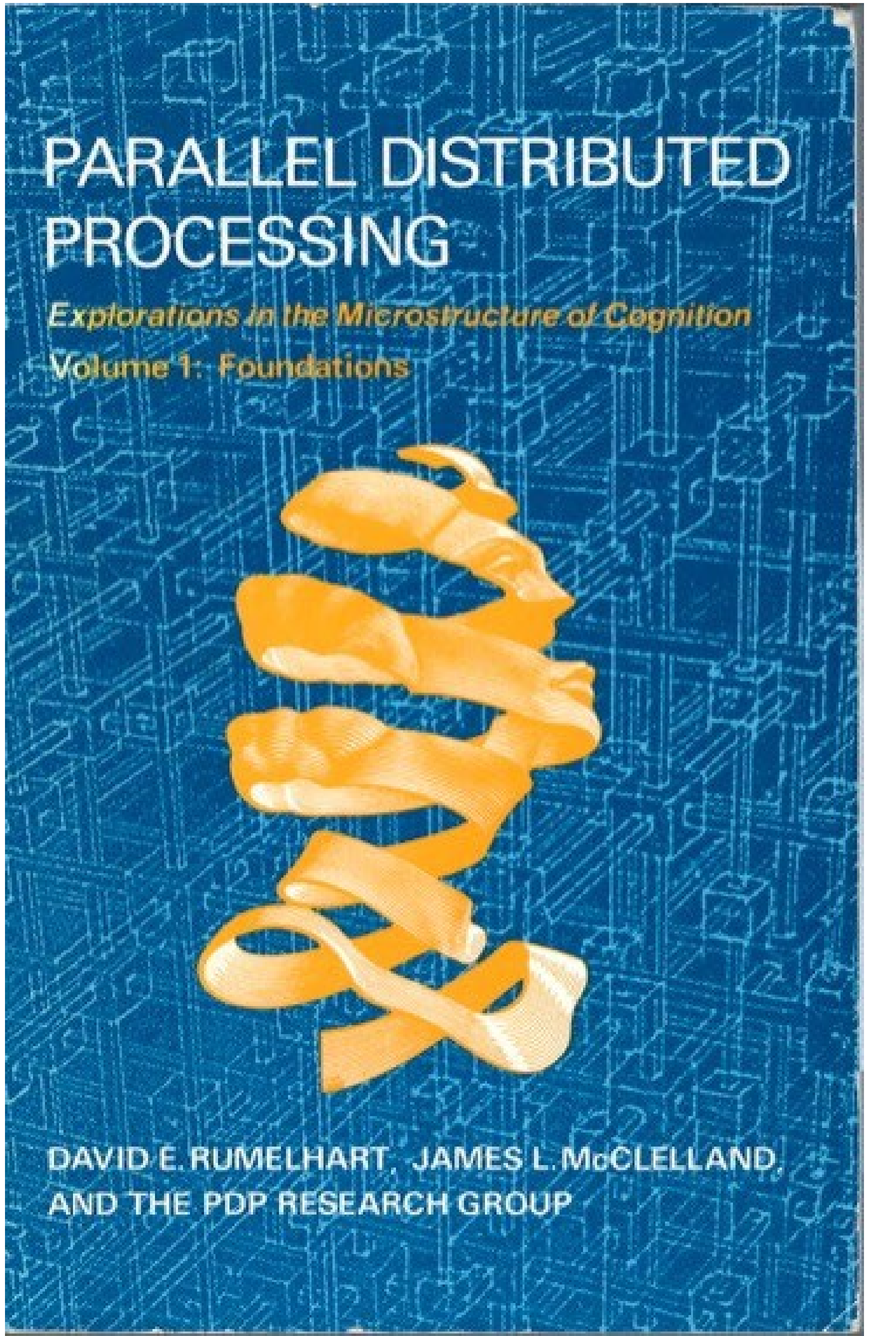
FOR BUYING OPTIONS, START HERE

Select Shipping Destination

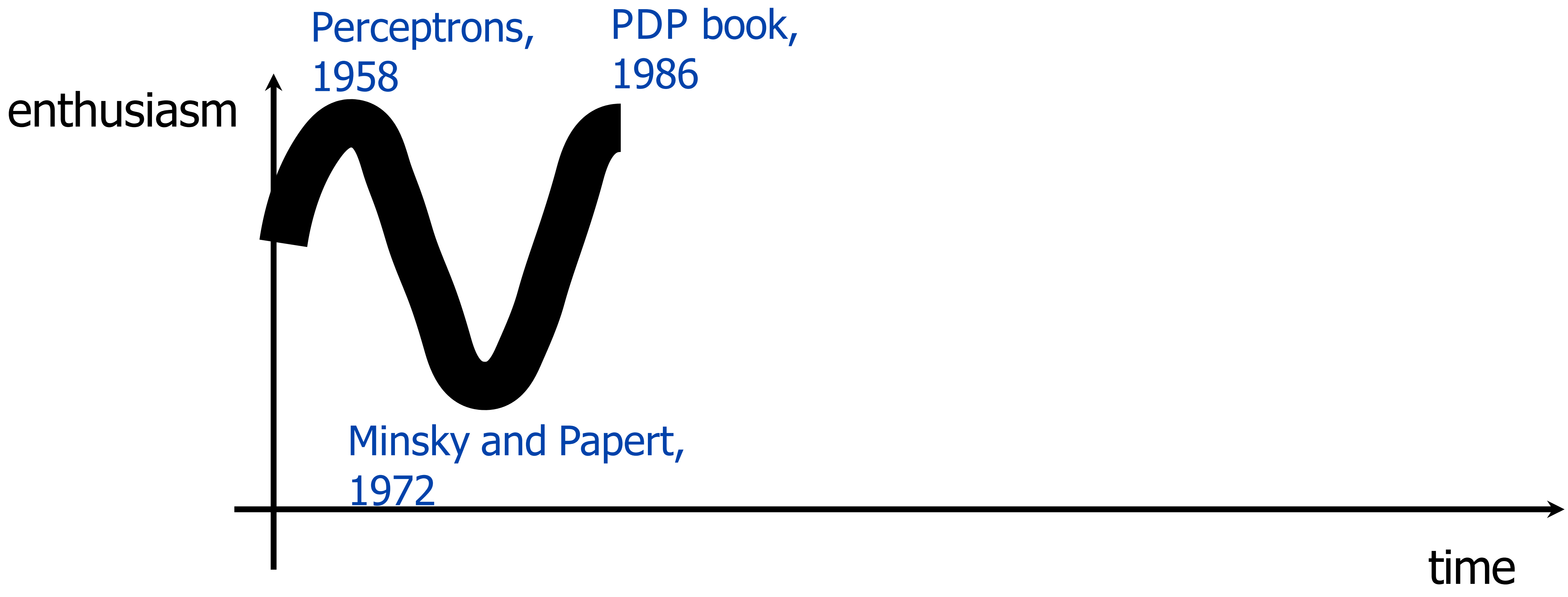
Paperback | $35.00 Short | £24.95 |
ISBN: 9780262631112 | 308 pp. | 6 x 8.9 in | December 1987


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

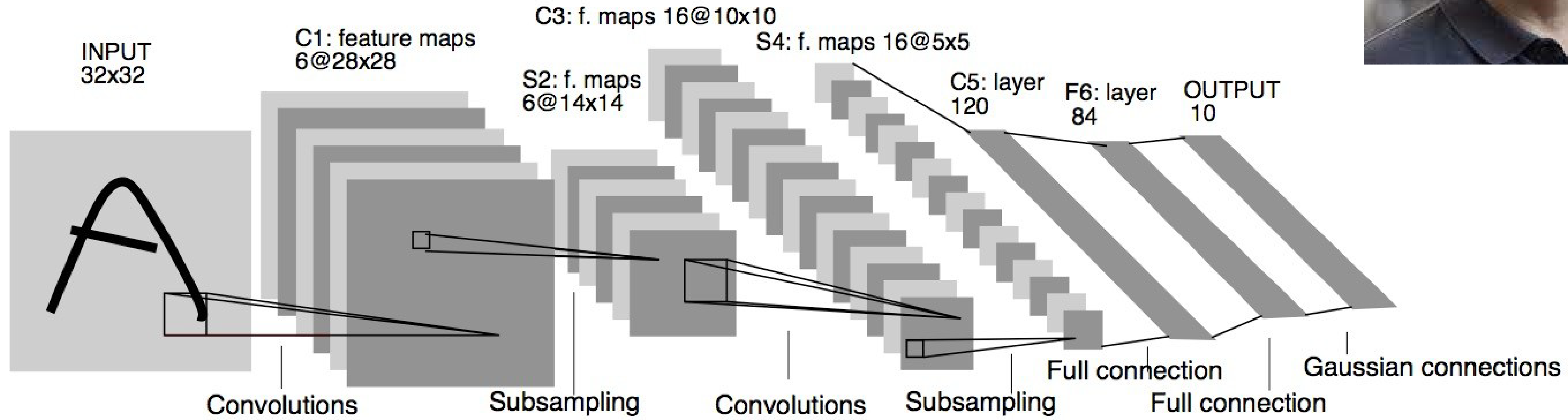


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

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

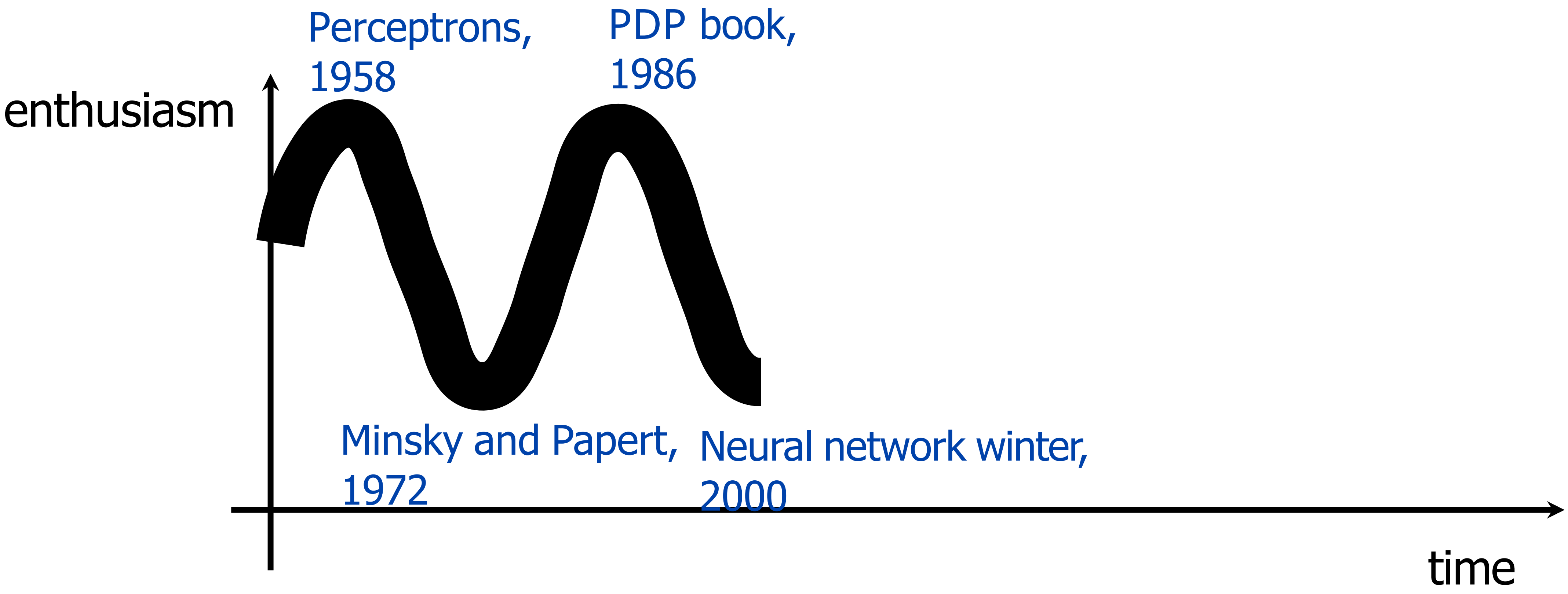
Source: Isola, Torralba, Freeman

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)

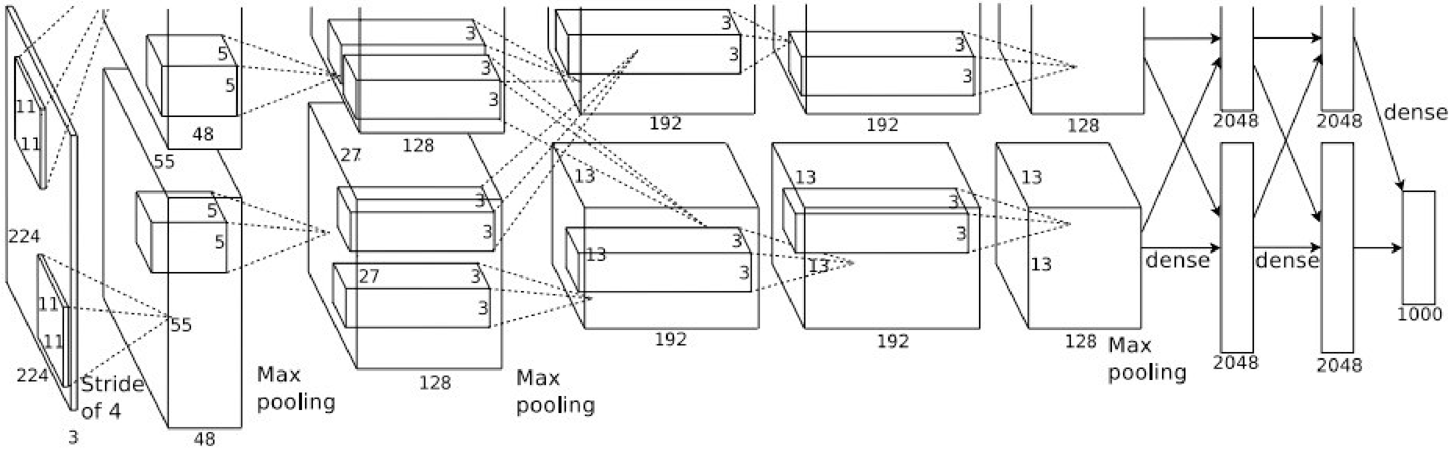

Perceptrons,
1958
PDP book,
1986
enthusiasm
Minsky and Papert,
1972
Neural network winter,
2000
time

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"

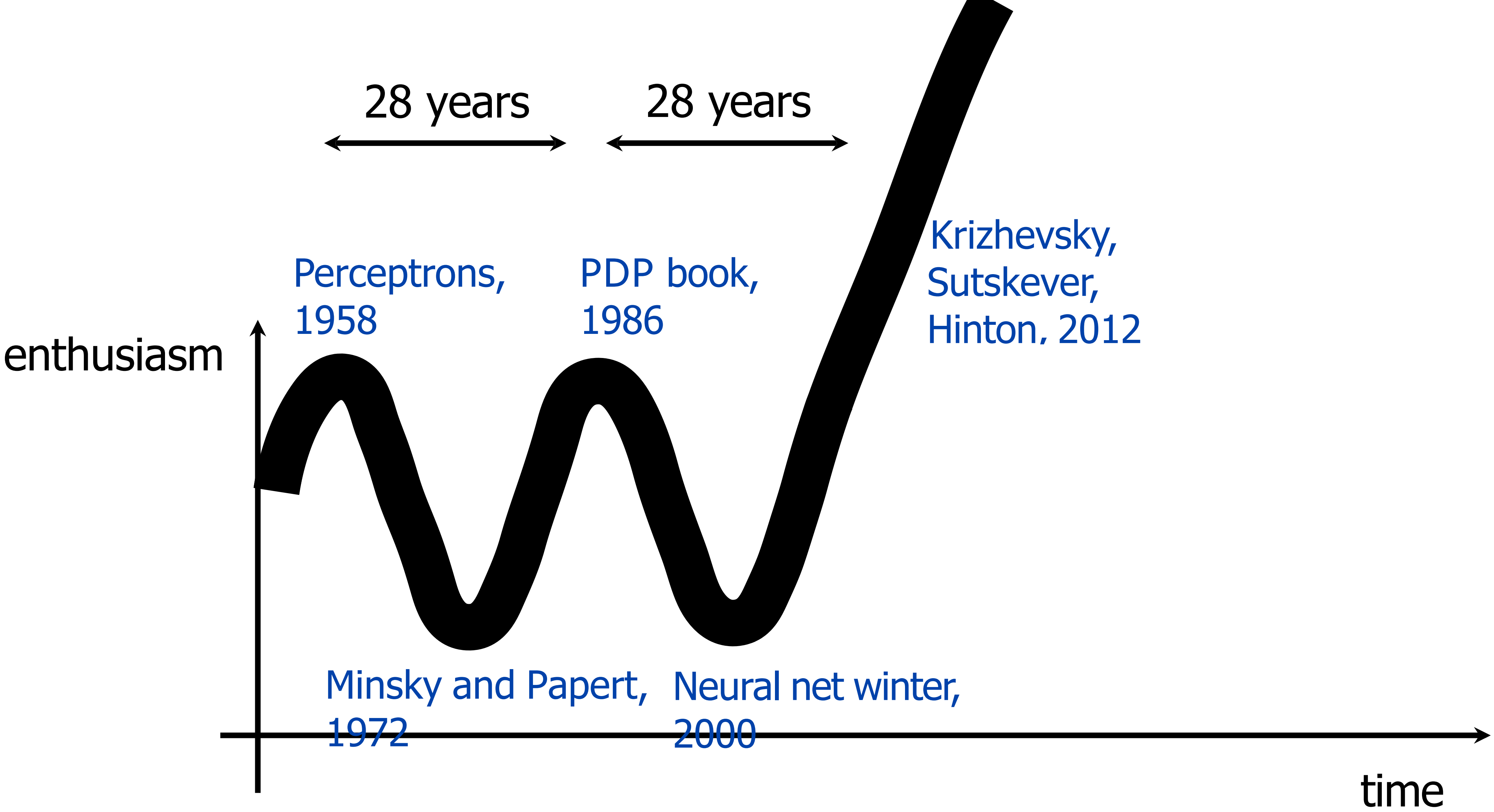


Got all the "pieces" right, e.g.,

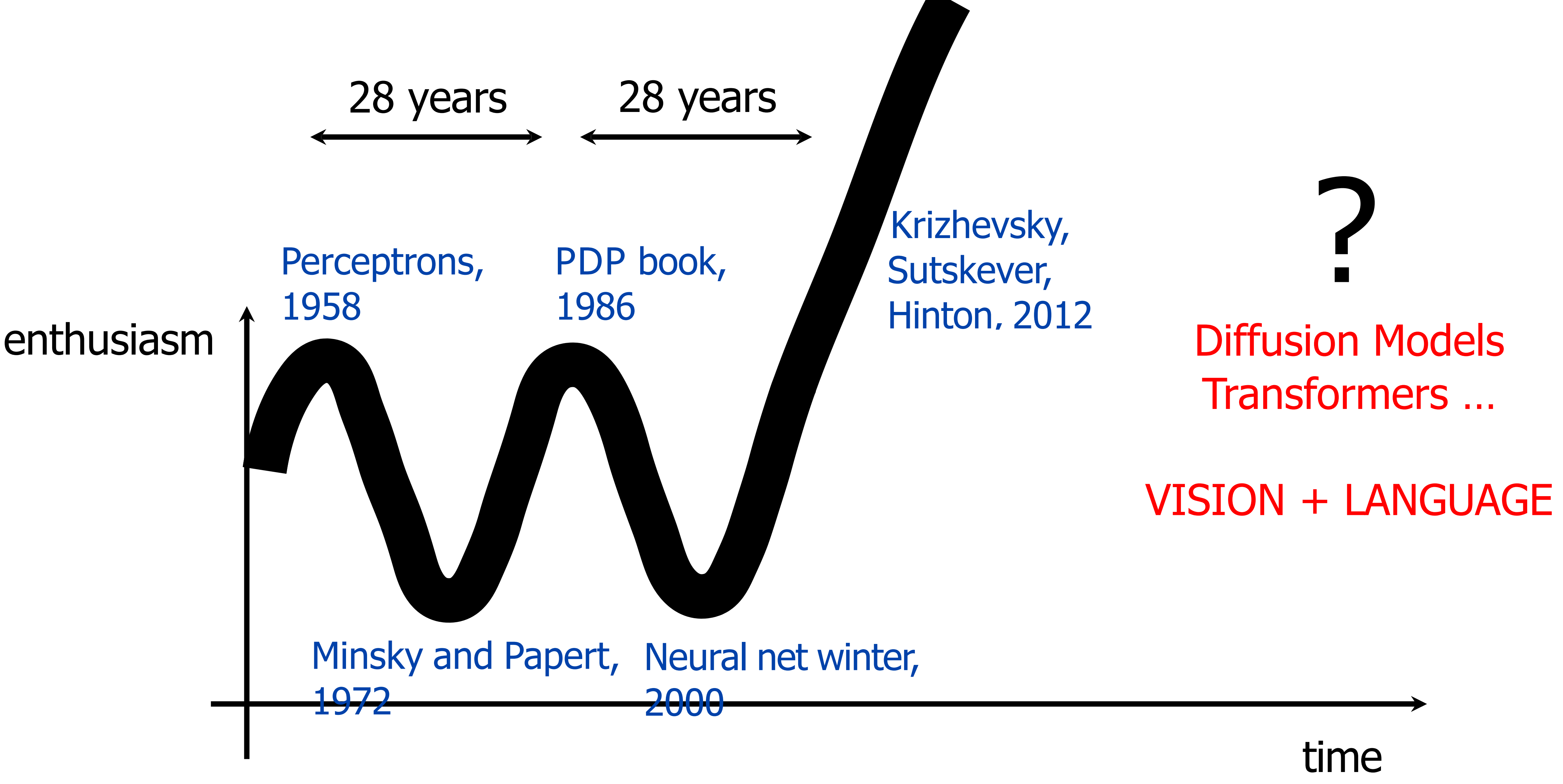
- Trained on ImageNet
- 8 layer architecture (for reference: today we have architectures with 100+ layers)
- Allowed for multi-GP training

Krizhevsky, Sutskever, and Hinton, NeurIPS 2012

Krizhevsky, Sutskever, and Hinton, NeurIPS 2012

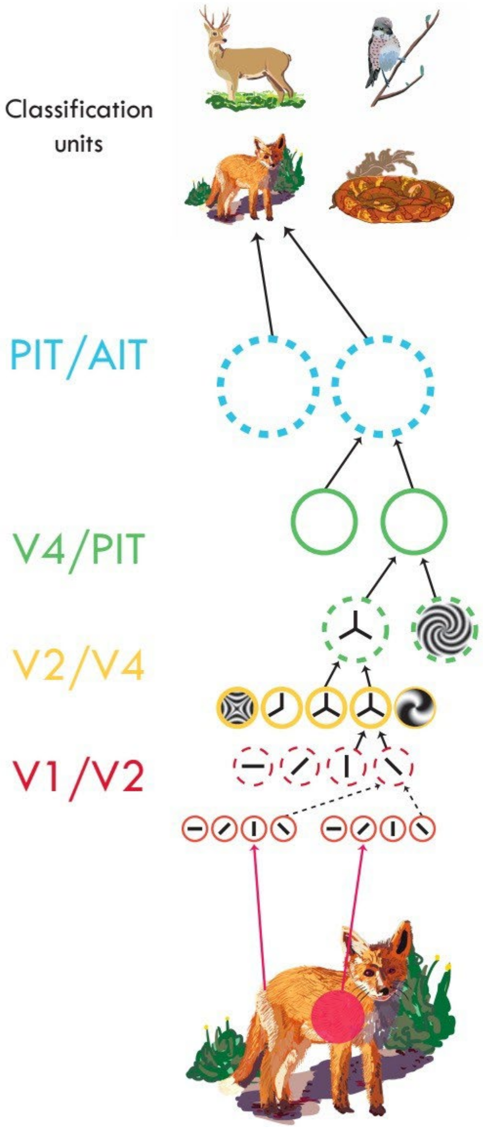


Source: Isola, Torralba, Freeman



Source: Isola, Torralba, Freeman

Inspiration: Hierarchical Representations

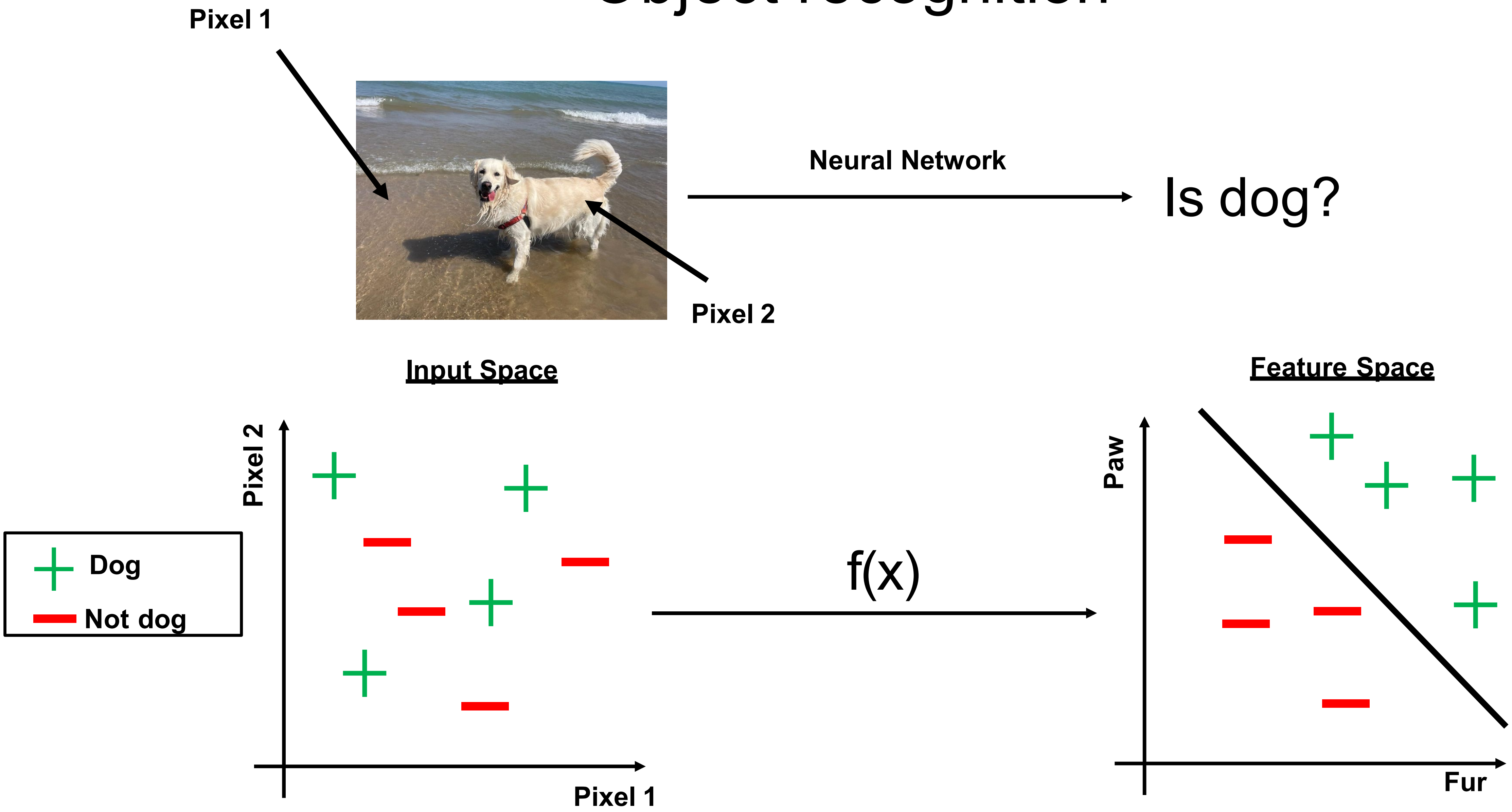


[Serre, 2014]

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

Source: Isola, Torralba, Freeman

Object recognition

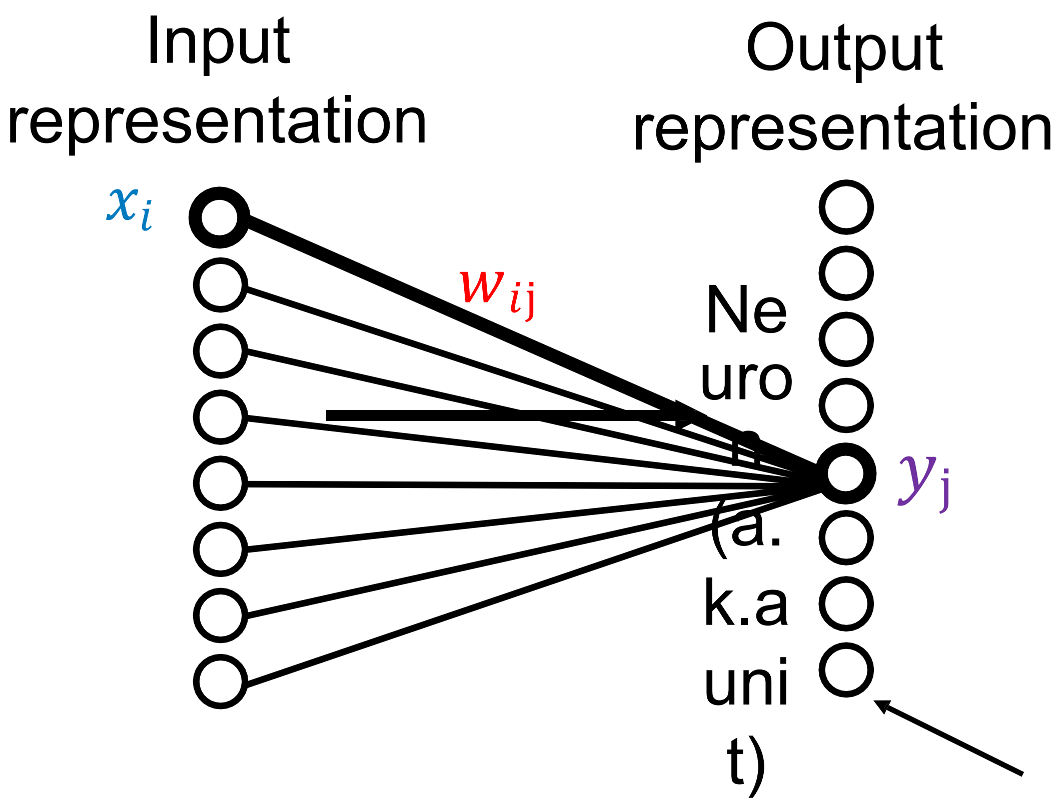


Goal: automatically learn a function that maps data from the input space to a feature space, i.e., "feature learning", rather than use hand-crafted features

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



weights

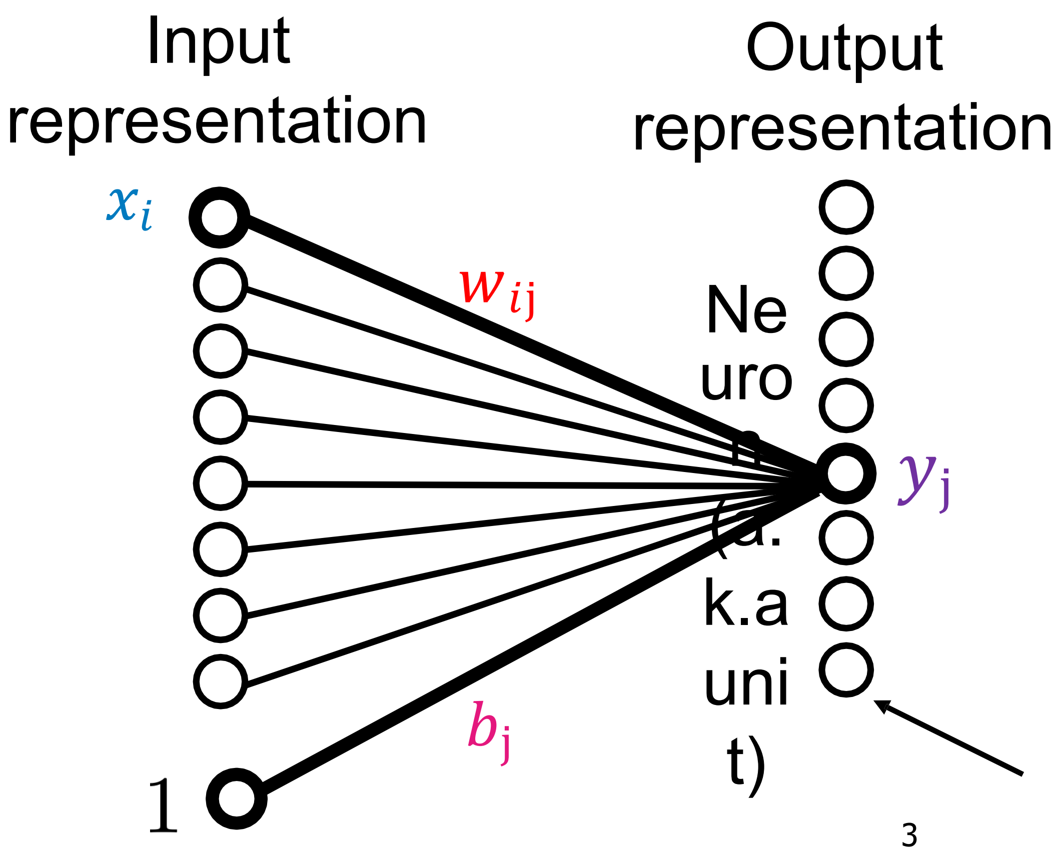
$$y_{\mathrm{j}} = \sum_i w_{i\mathrm{j}} x_i$$

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



weights

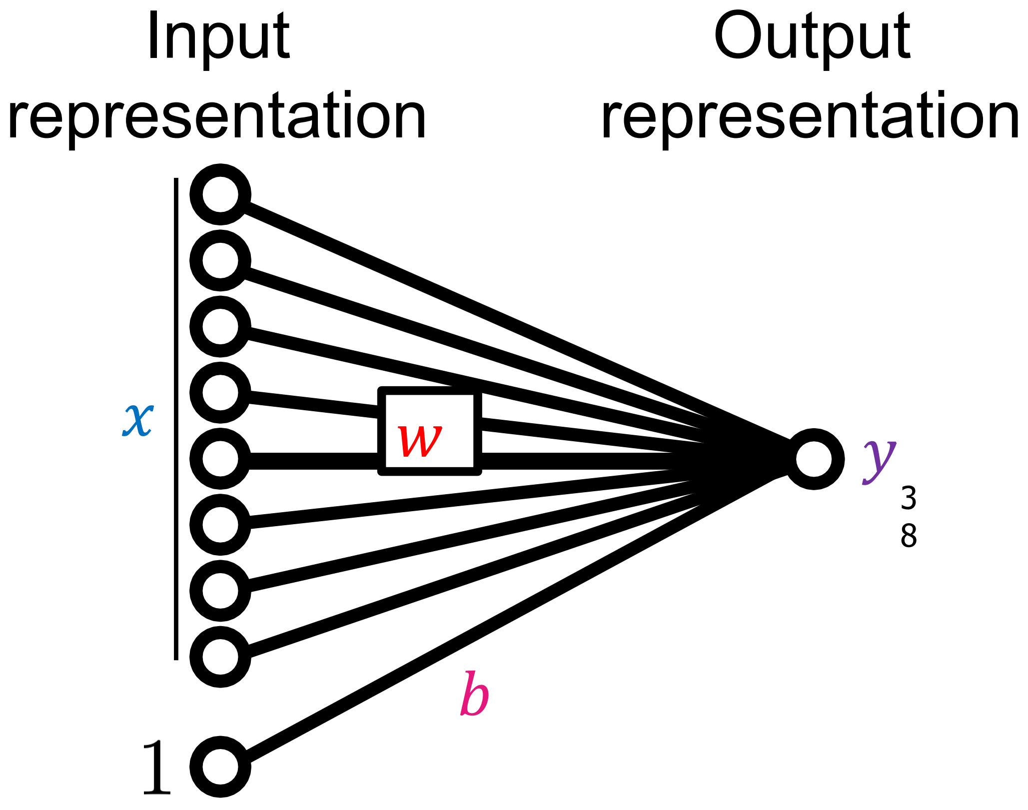
$$y_j = \sum_i w_{ij} x_i + b_j$$

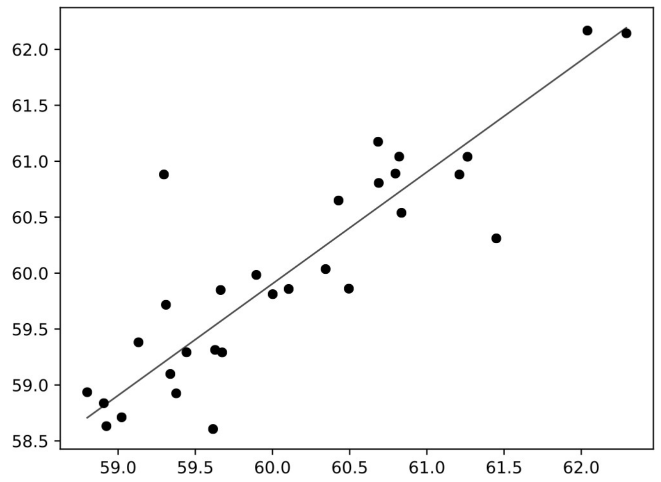
bias

Adapted from: Isola, Torralba, Freeman

Example: Linear Regression

Linear layer



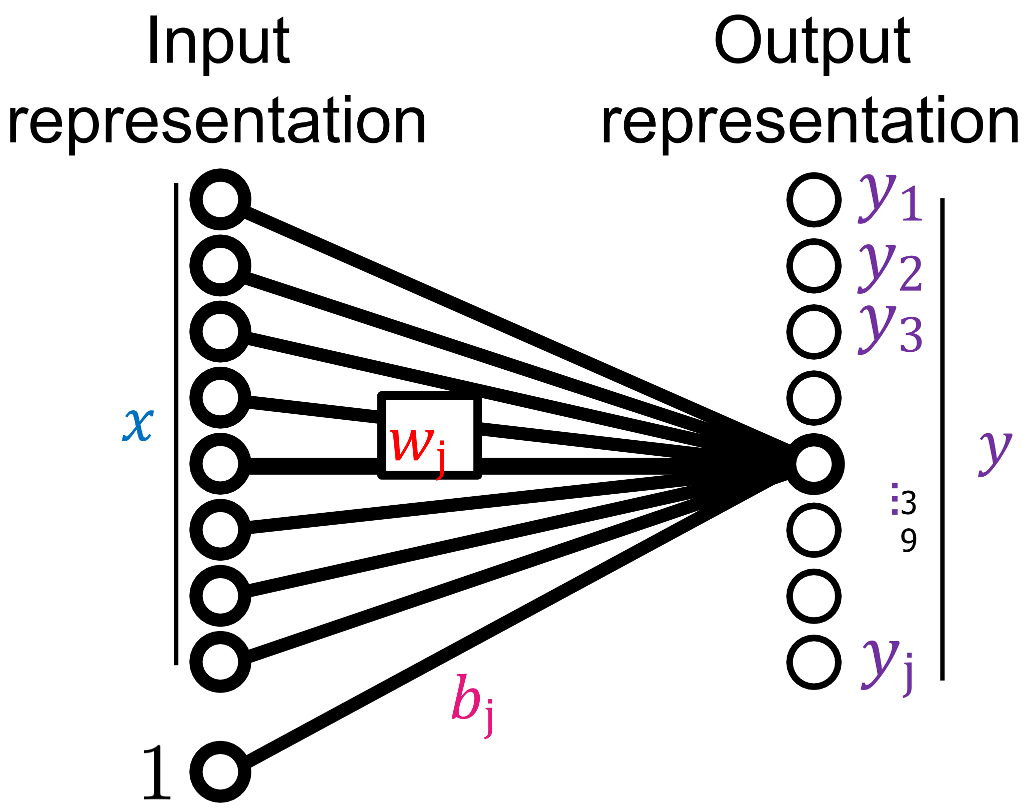


$$f_{\mathbf{w},b}(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + b$$

Adapted from: Isola, Torralba, Freeman

Computation in a neural net – Full Layer

Linear layer



$$y = Wx + b$$

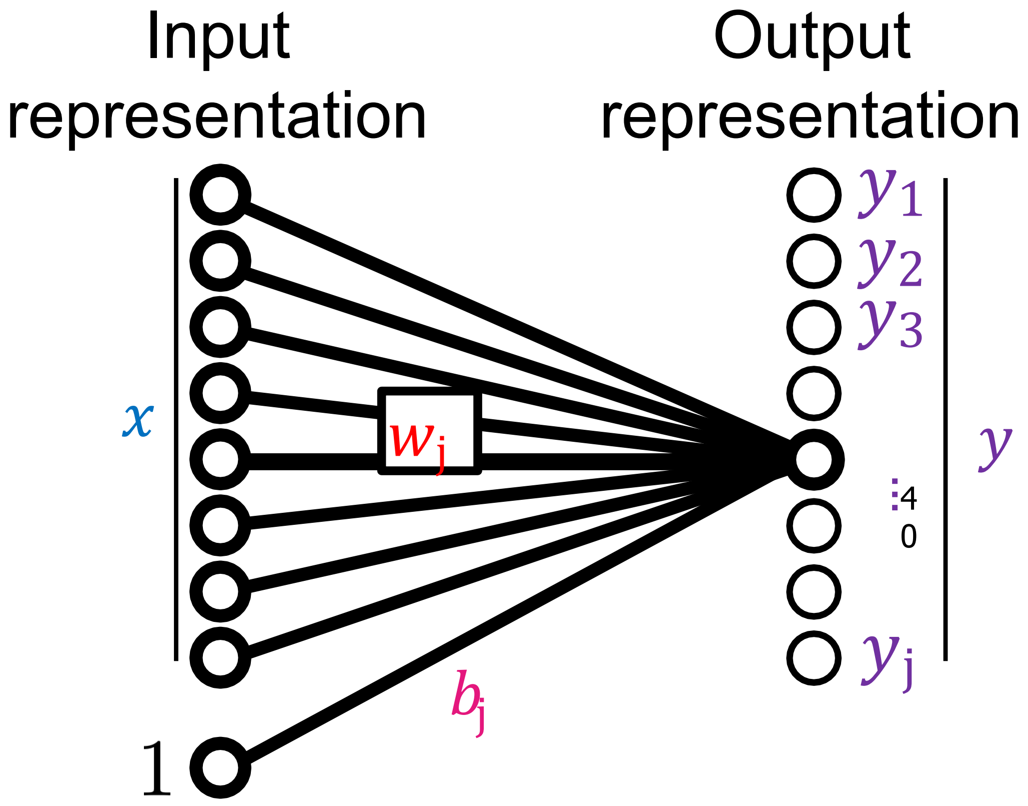
$$\begin{bmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{j1} & \cdots & w_{jn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_j \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_j \end{bmatrix}$$

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 $\mathbf{x}$

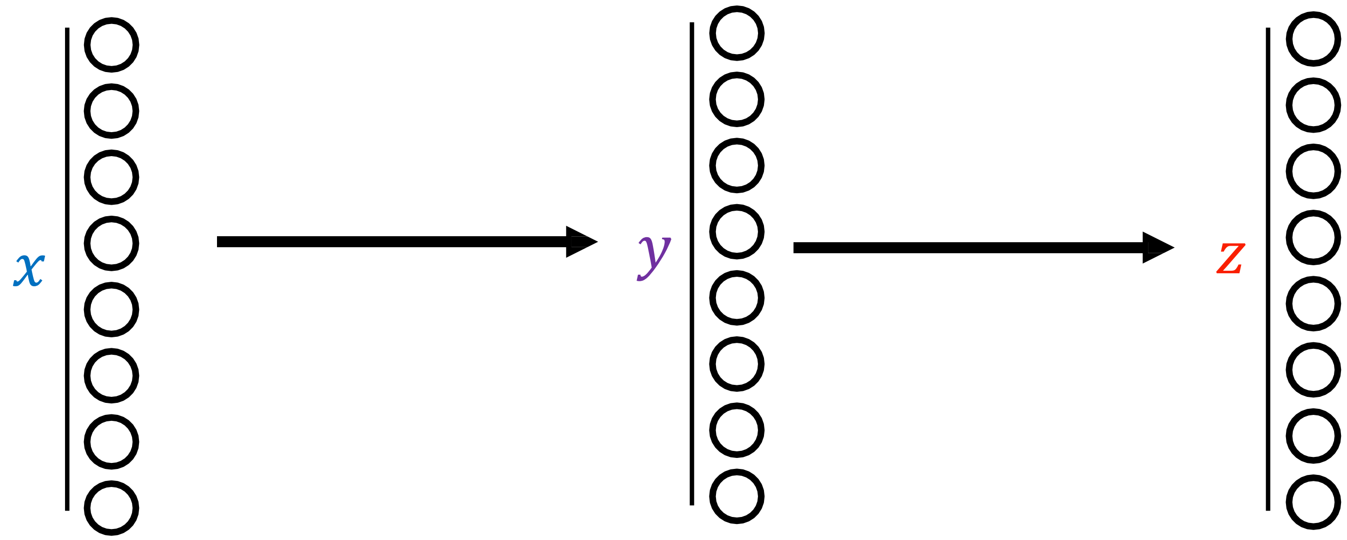
Adapted from: Isola, Torralba, Freeman

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:

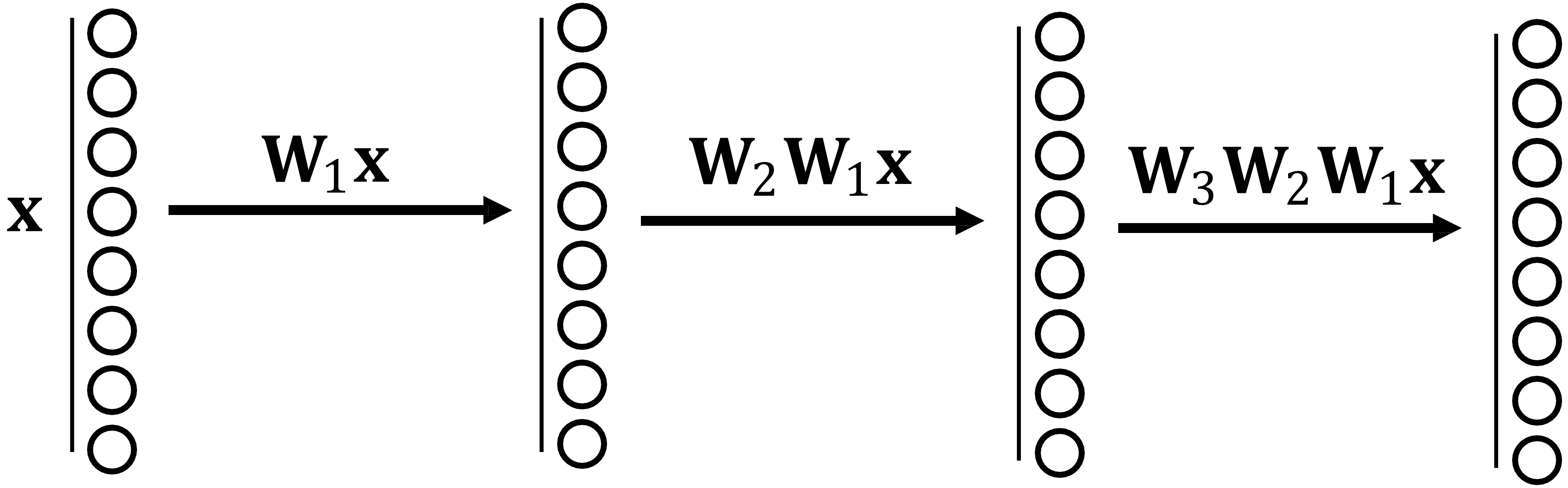
Input representation

Output representation



We can repeat this as many times as we want!

What is the problem with this idea?

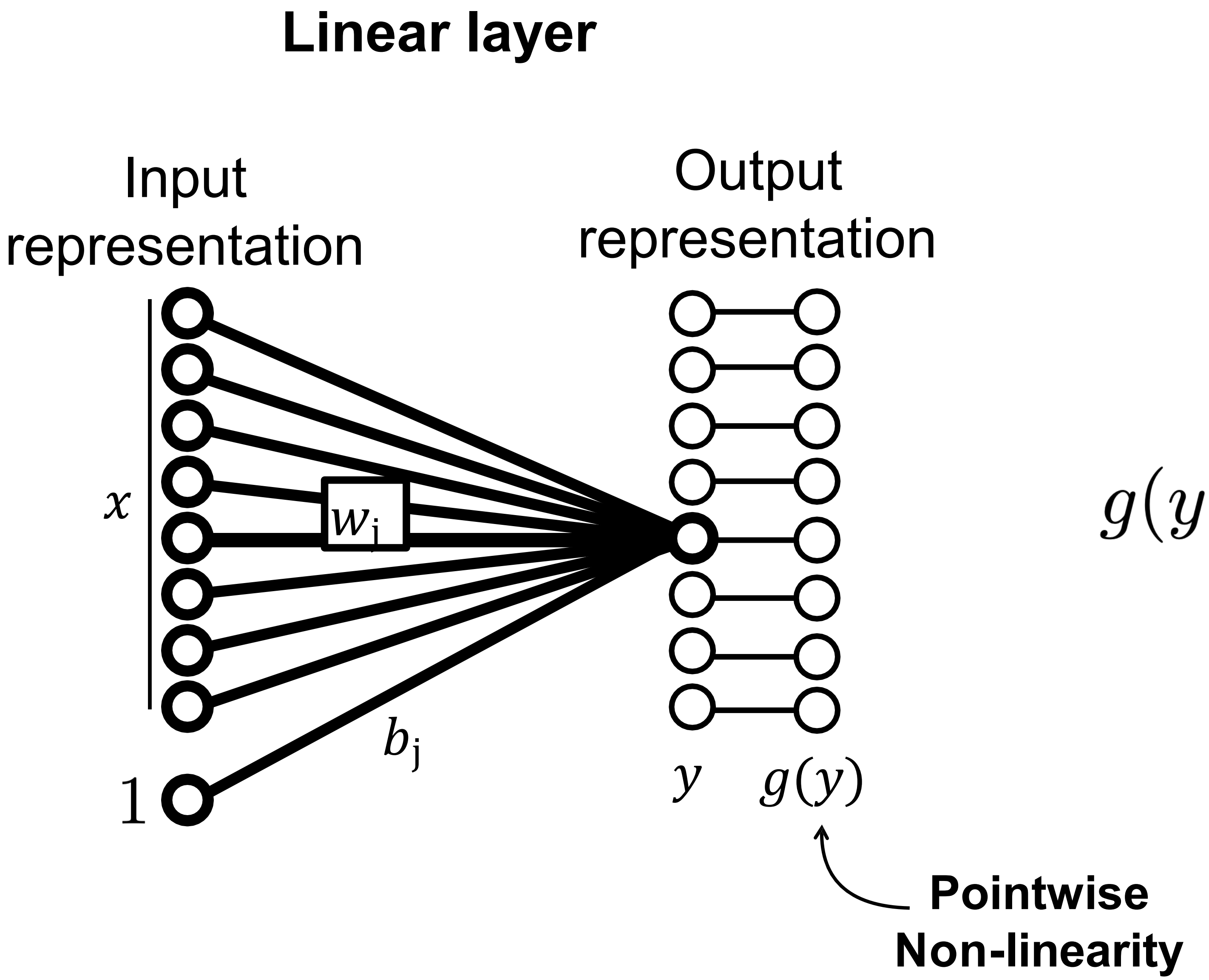


Can be expressed as single linear layer!

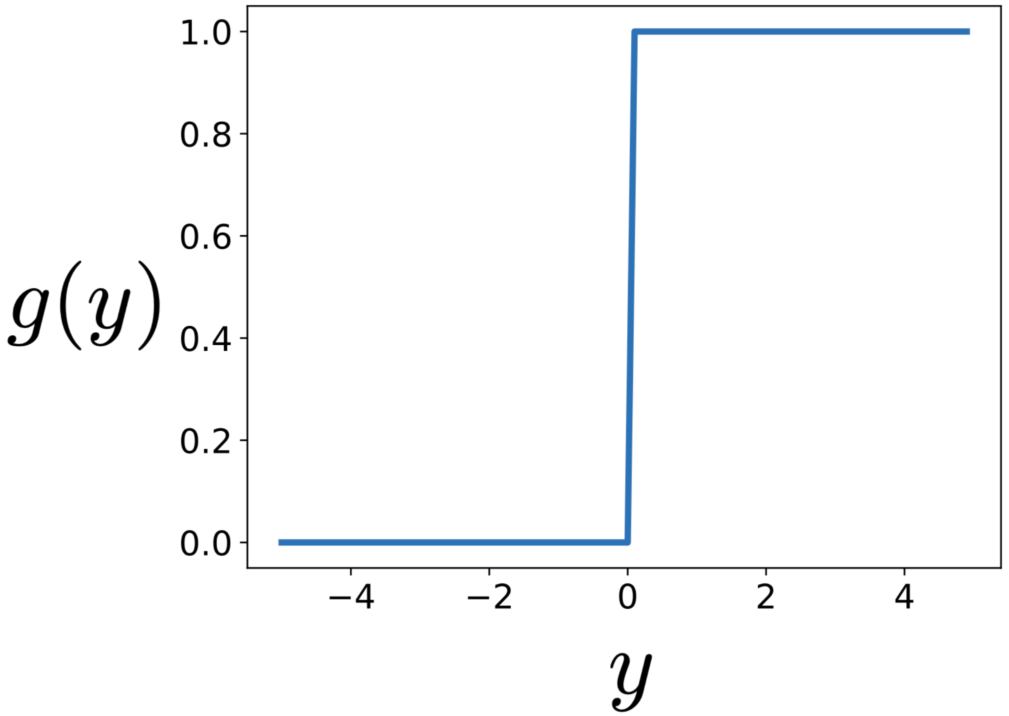
$$\left(\prod_i \mathbf{W}_i\right)\mathbf{x} = \hat{\mathbf{W}}\mathbf{x}$$

Limited power: can't solve XOR ☹

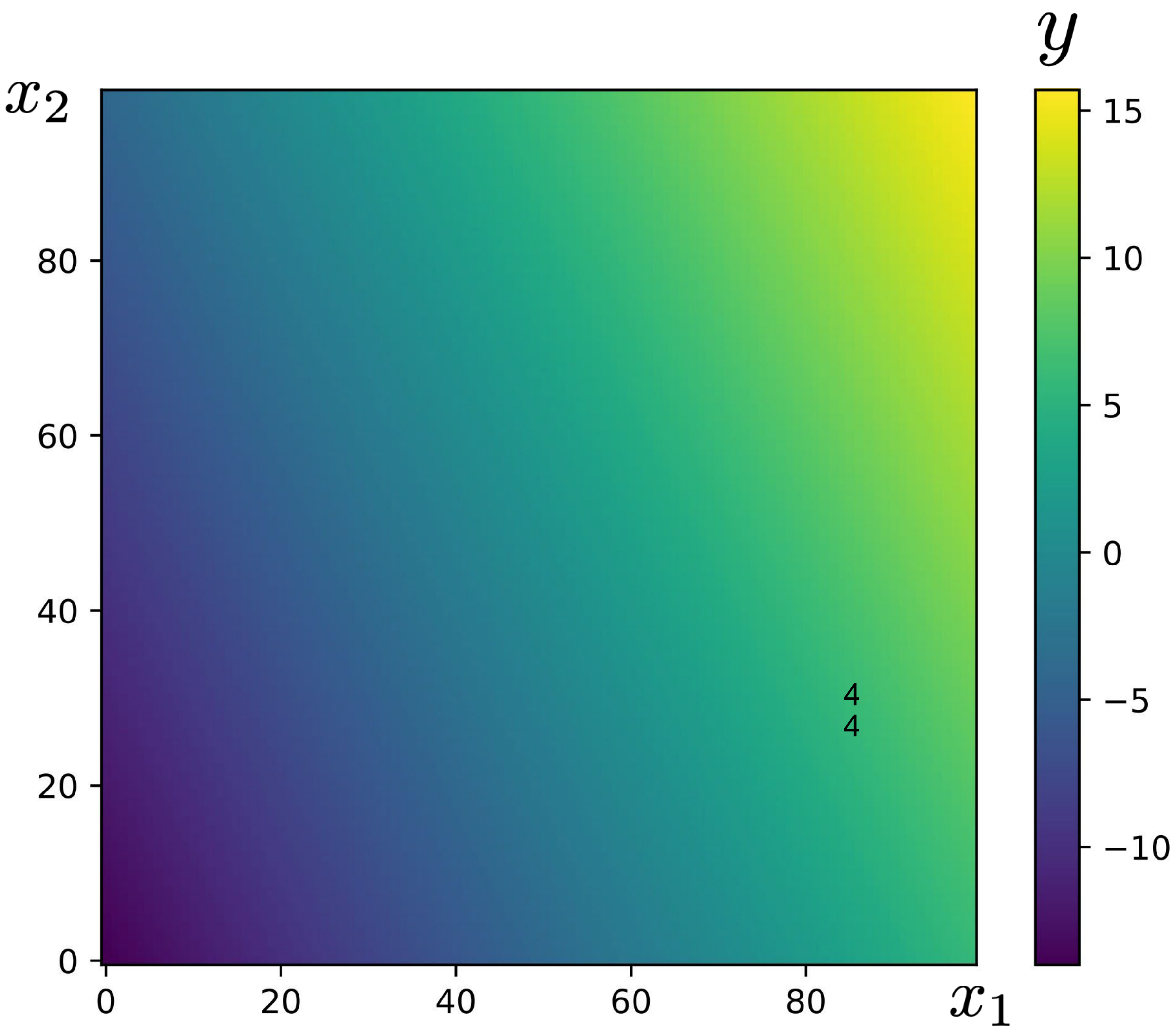
Solution: simple nonlinearity



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



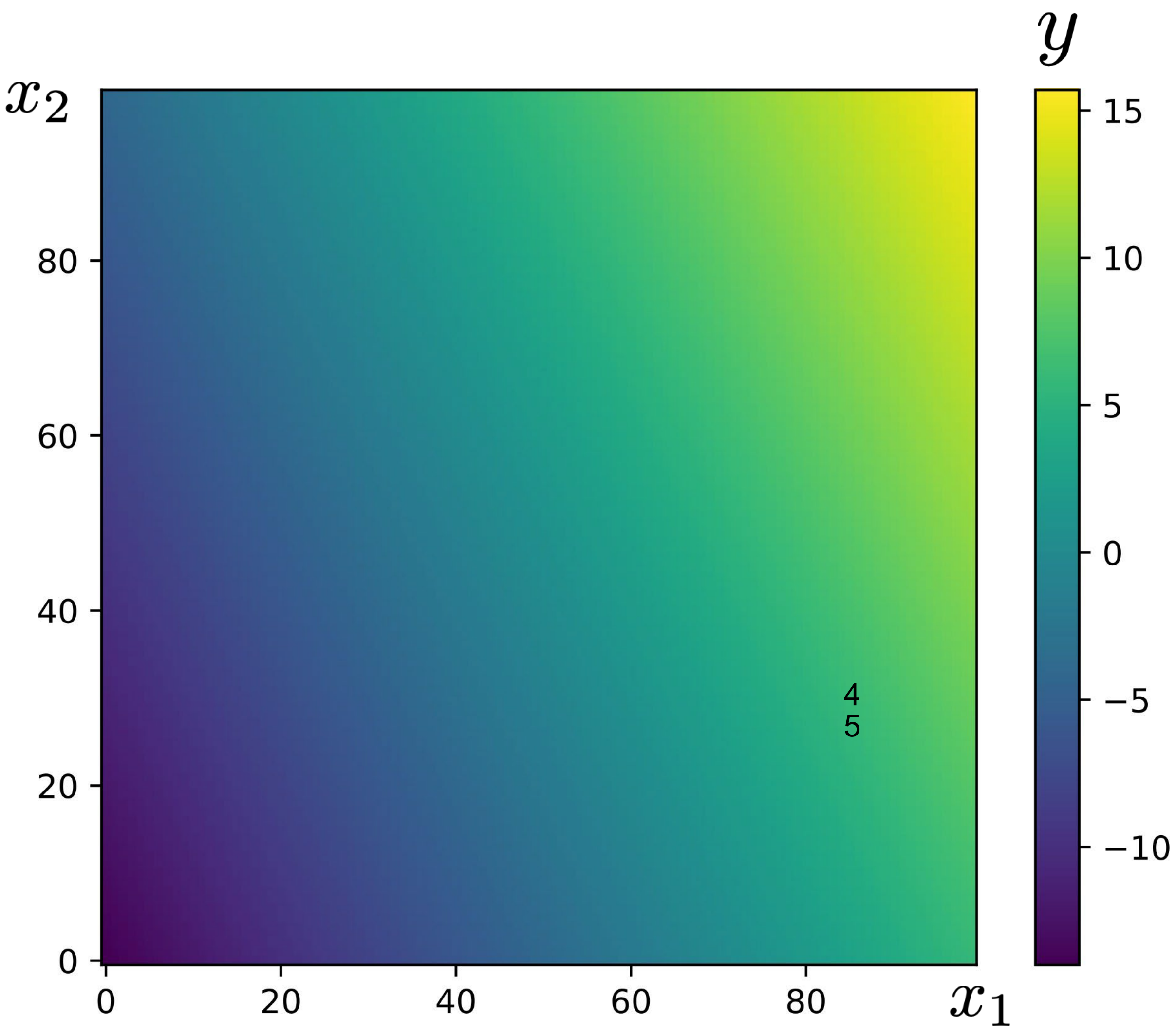
Example: linear classification with a perceptron



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

Source: Isola, Torralba, Freeman

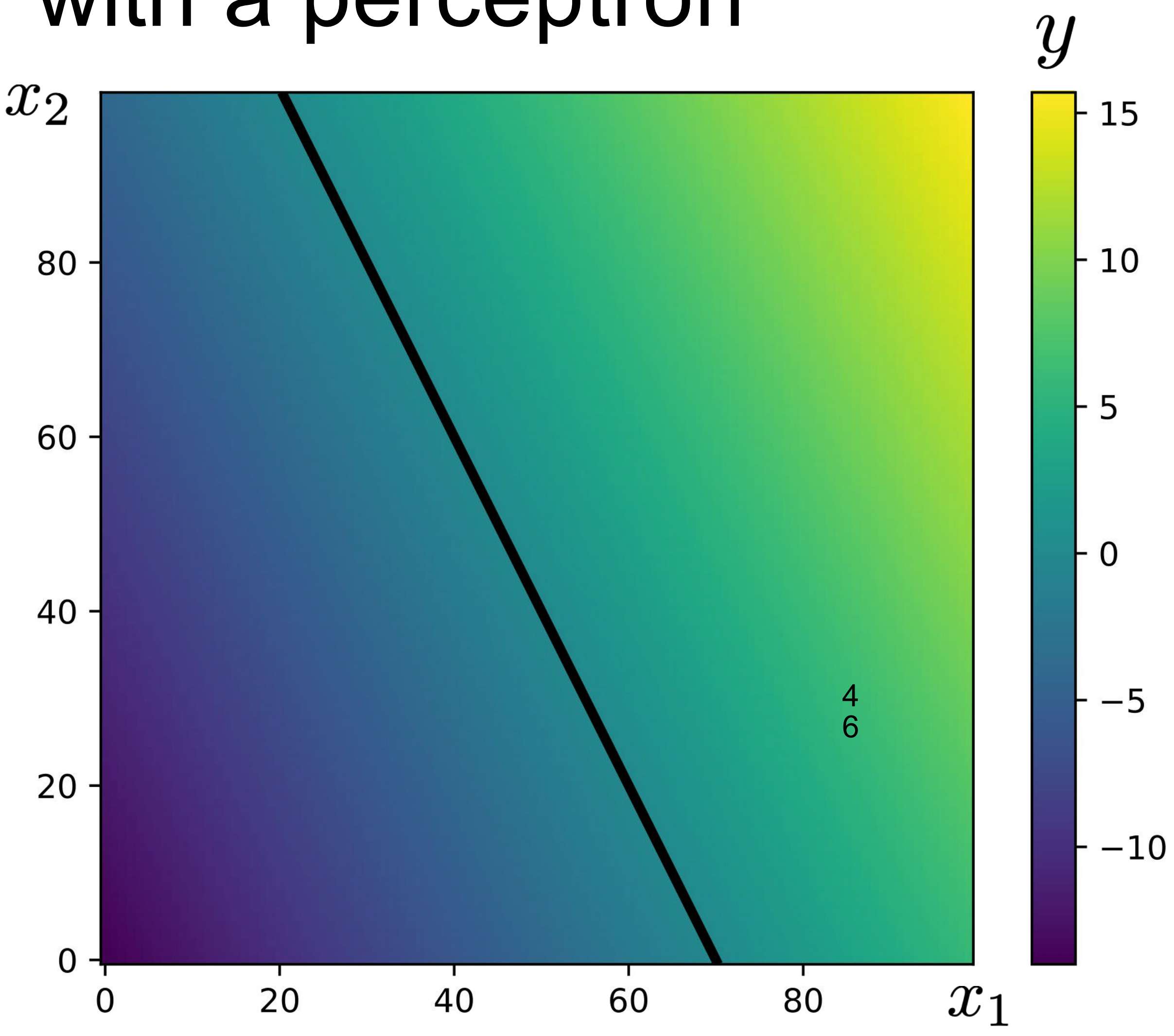
Example: linear classification with a perceptron



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

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

Example: linear classification with a perceptron

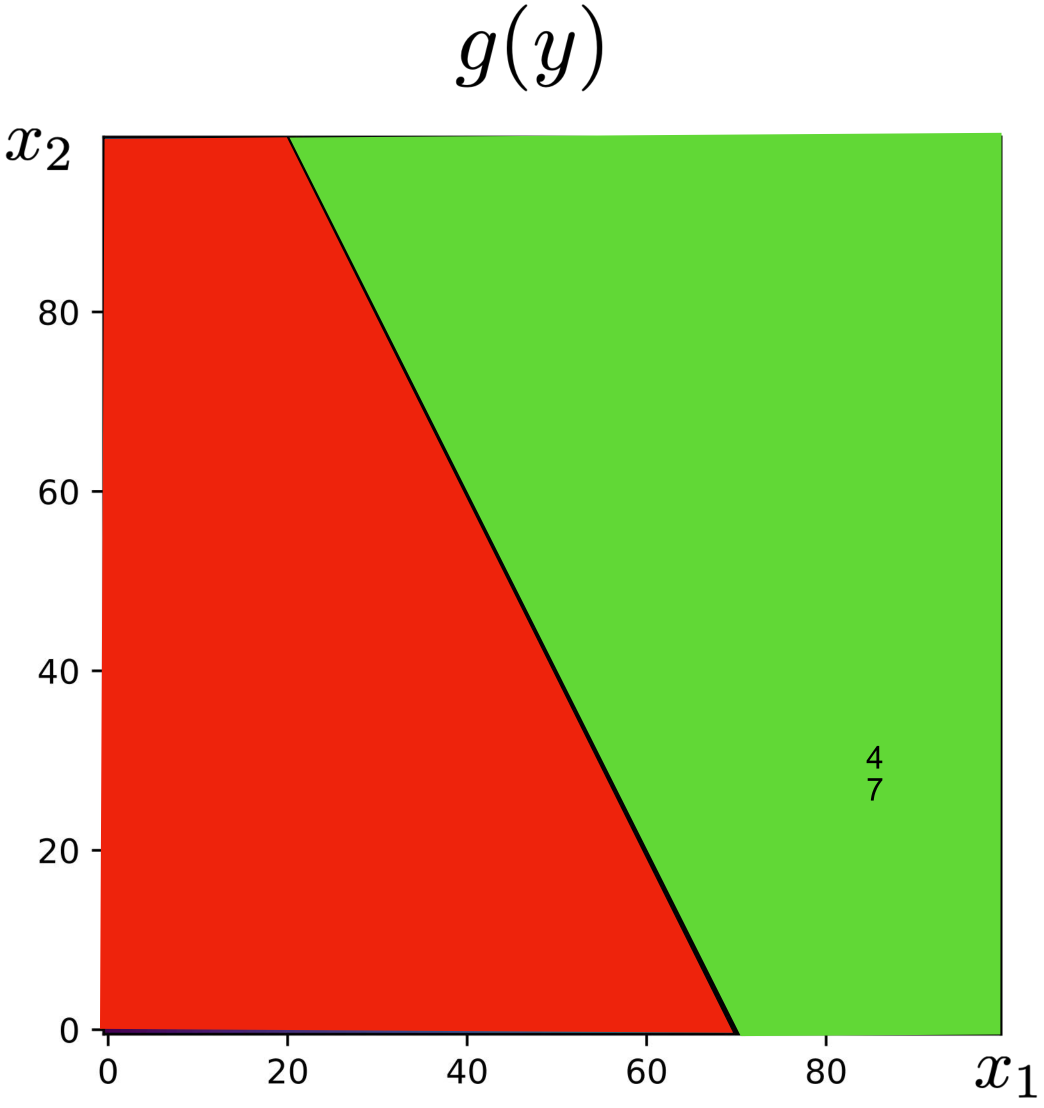


$$y = \mathbf{x}^T \mathbf{w} + b$$

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

"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

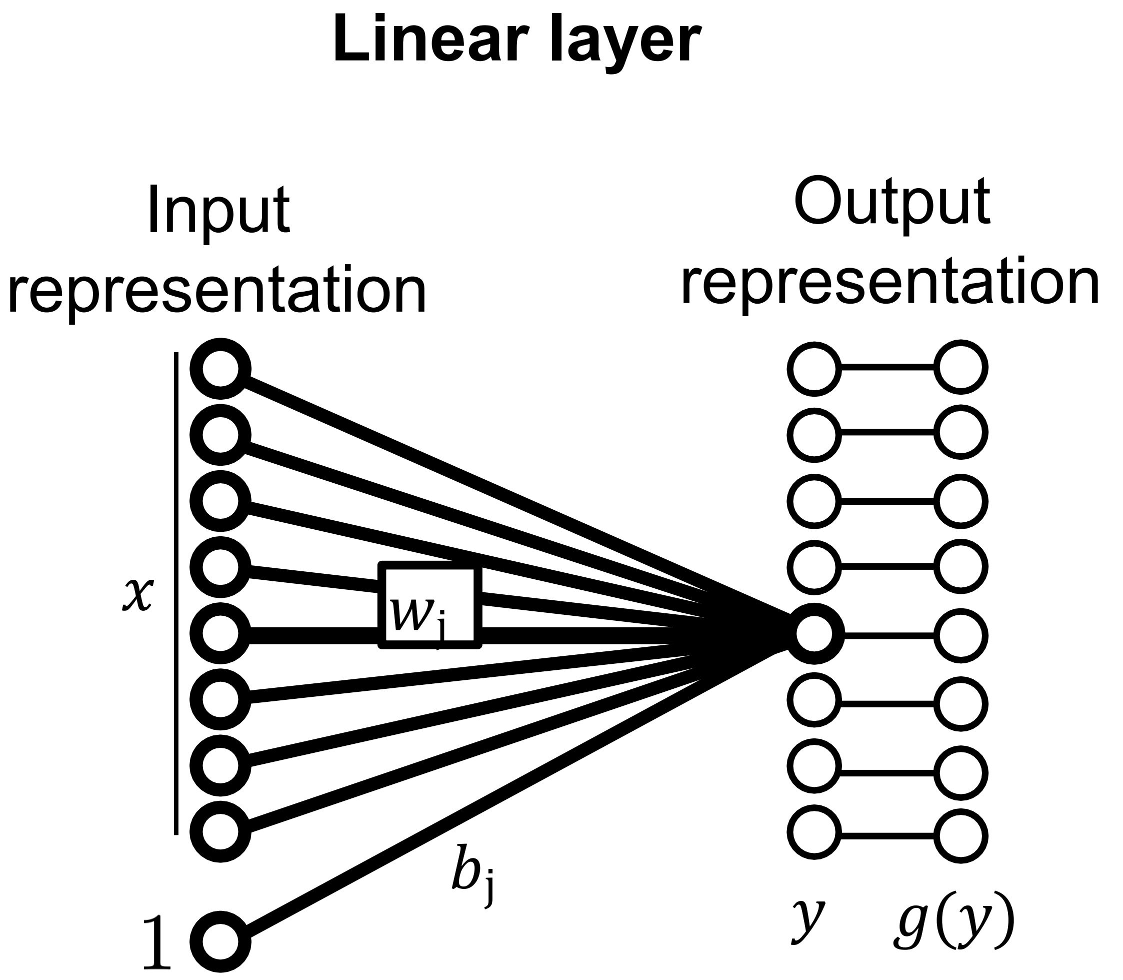


$$y = \mathbf{x}^T \mathbf{w} + b$$

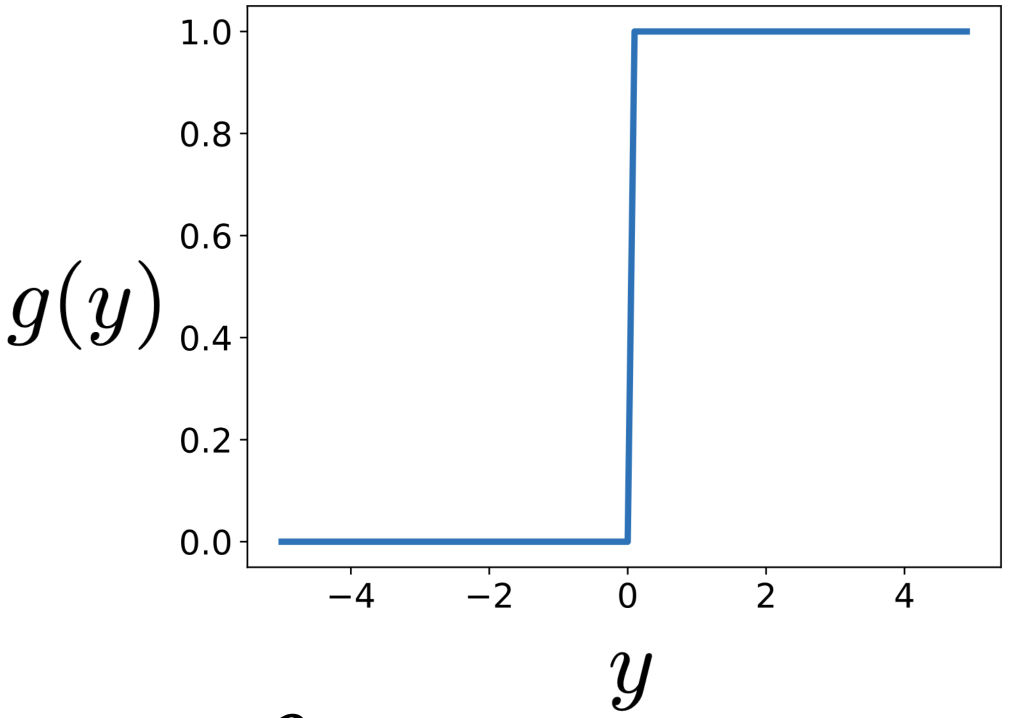
$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$

"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



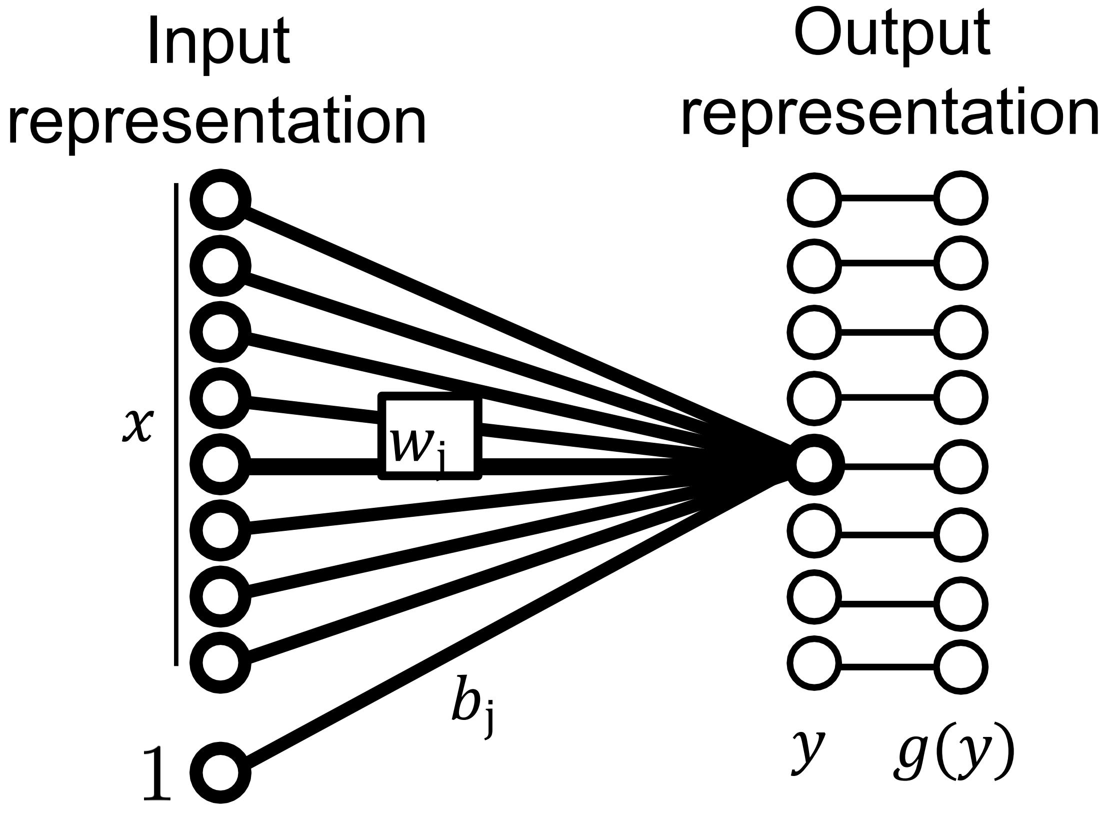
$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$



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

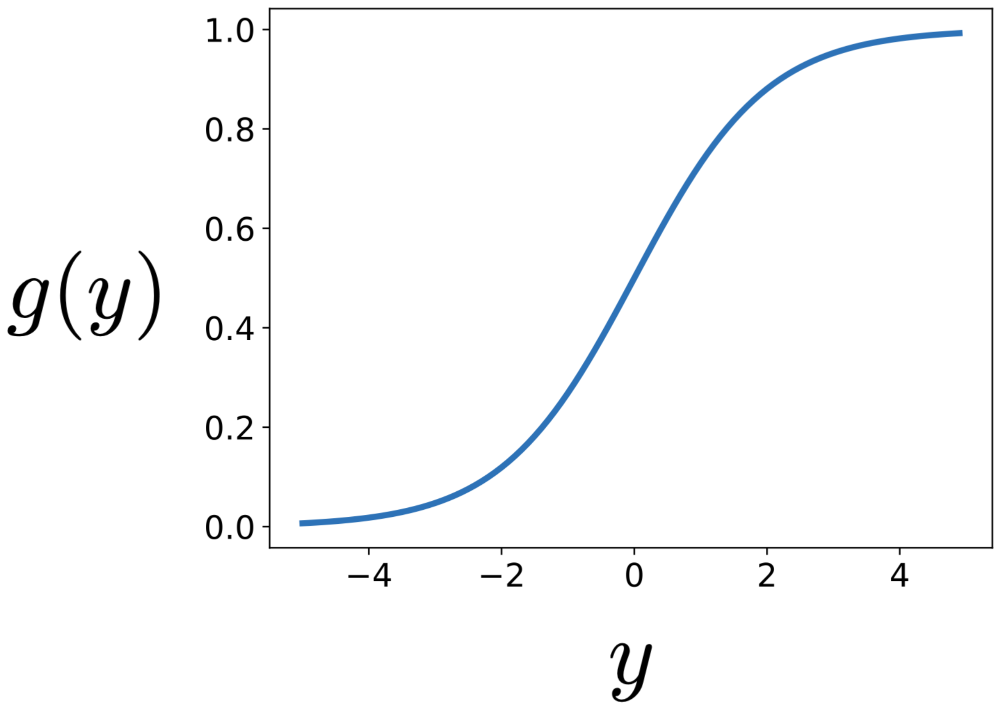
Computation in a neural net - nonlinearity

Linear layer



Sigmoid

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

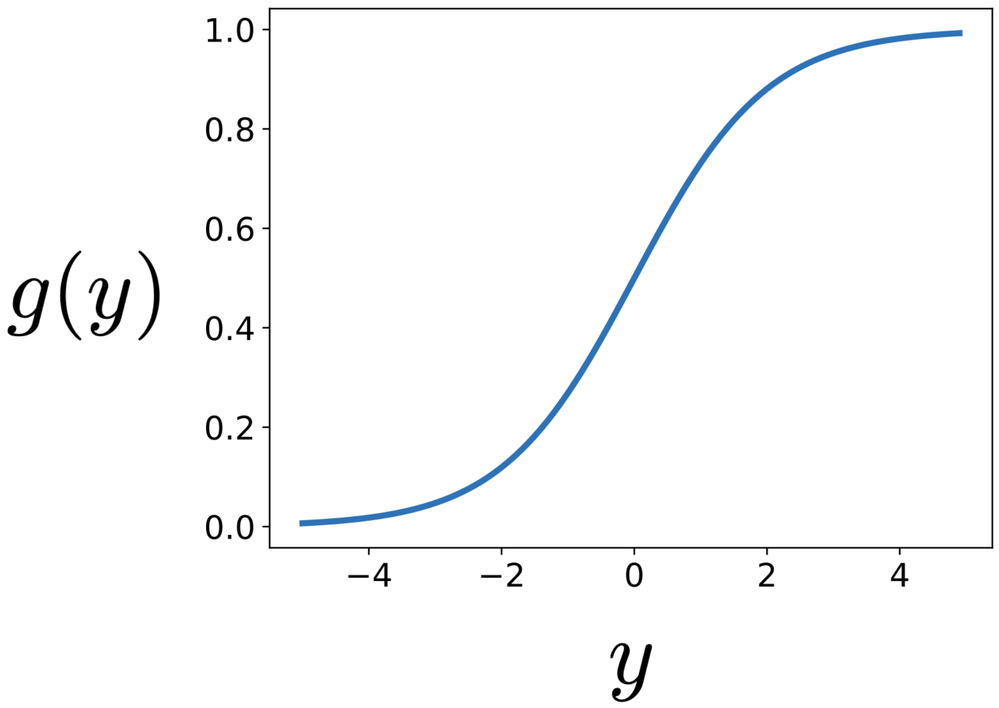


Computation in a neural net - nonlinearity

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

Sigmoid

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

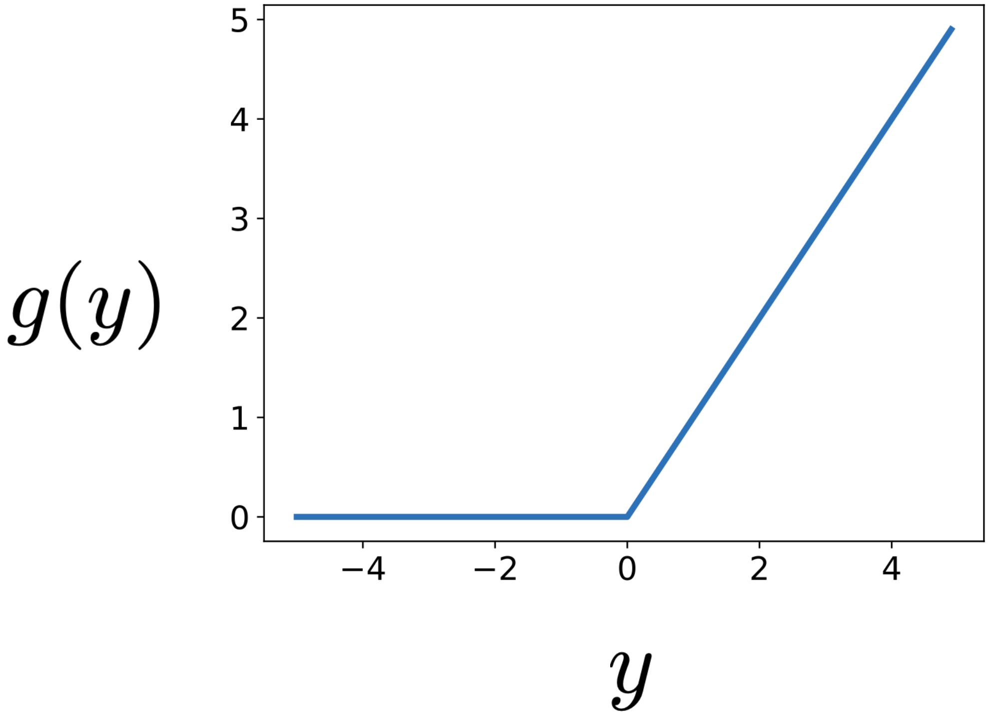


Computation in a neural net — nonlinearity

- 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)

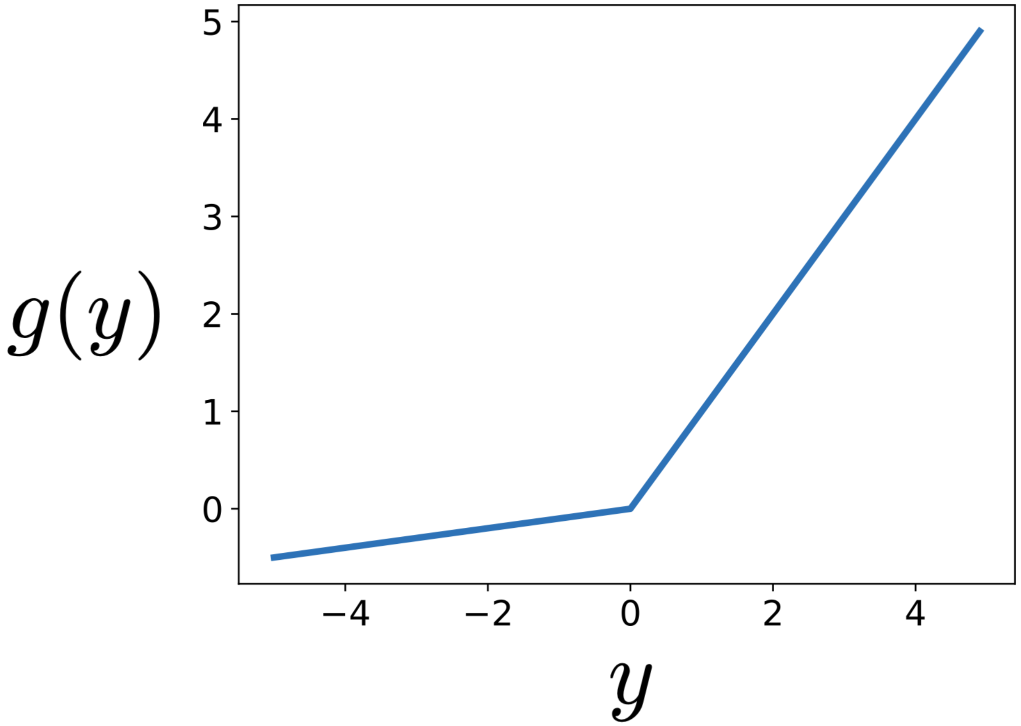
- Efficient to implement:

- Has non-zero gradients everywhere (unlike ReLU)

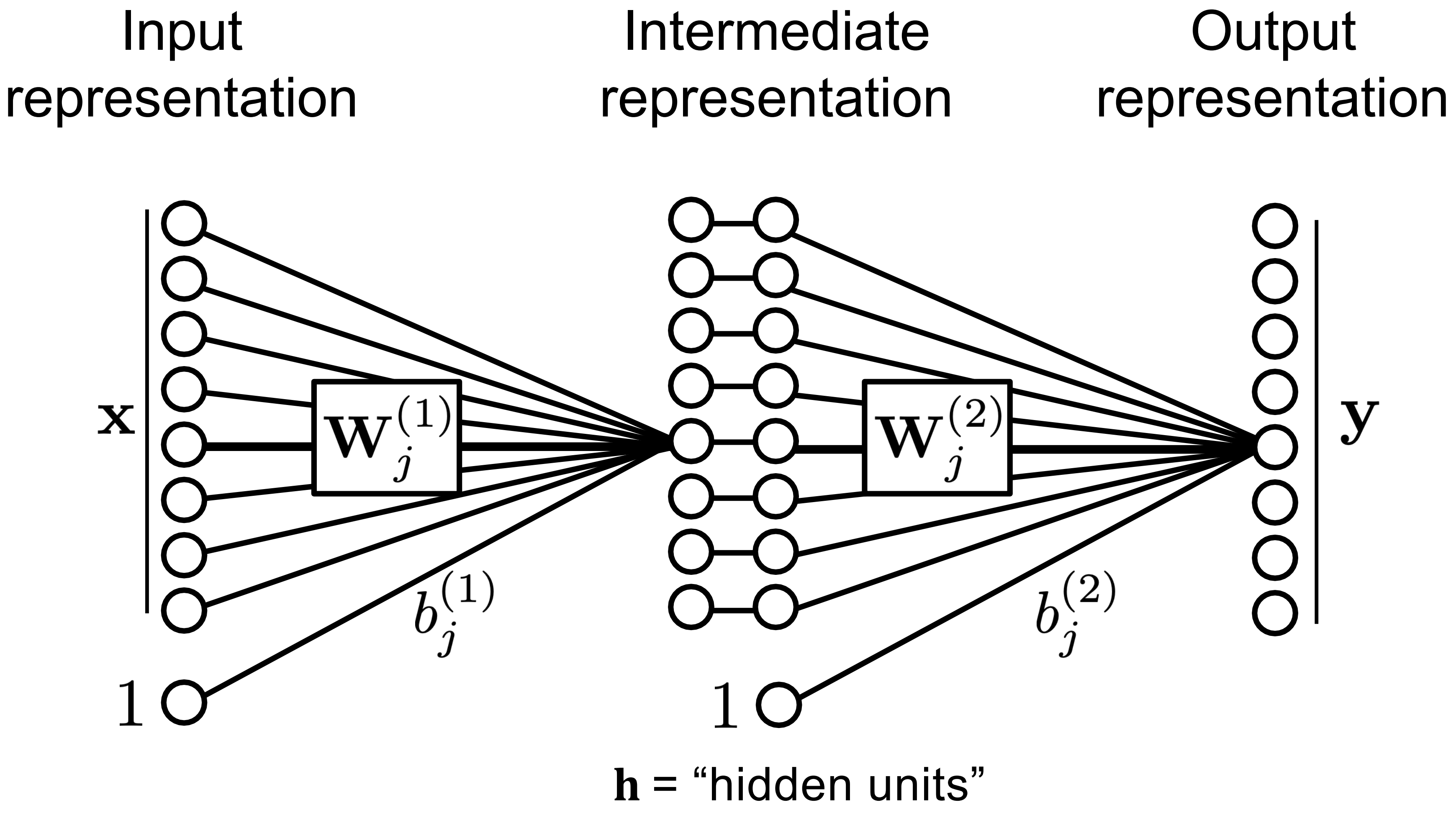
$$\frac{\partial g}{\partial y} = \begin{cases} -a, & \text{if} \quad y < 0 \\ 1, & \text{if} \quad y \geq 0 \end{cases}$$

Leaky ReLU

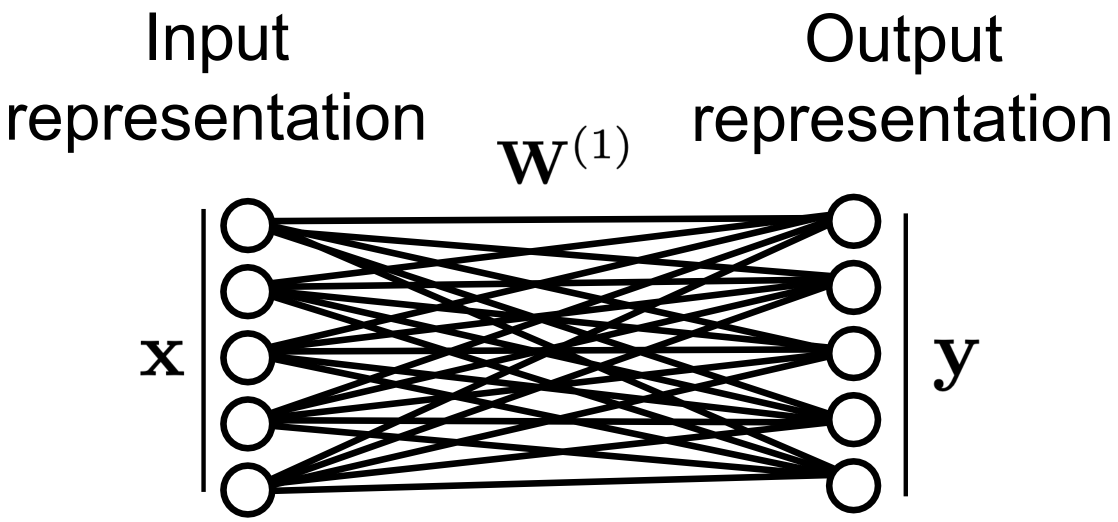
$$g(y) = \begin{cases} \max(0, y), & \text{if} \quad y \geq 0 \\ a \min(0, y), & \text{if} \quad y < 0 \end{cases}$$



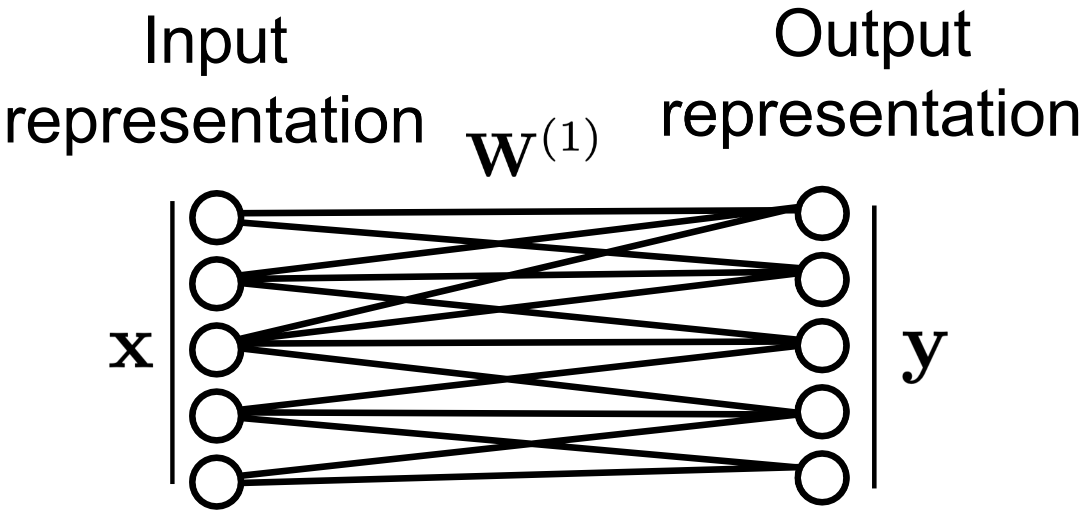
Stacking layers



Connectivity patterns

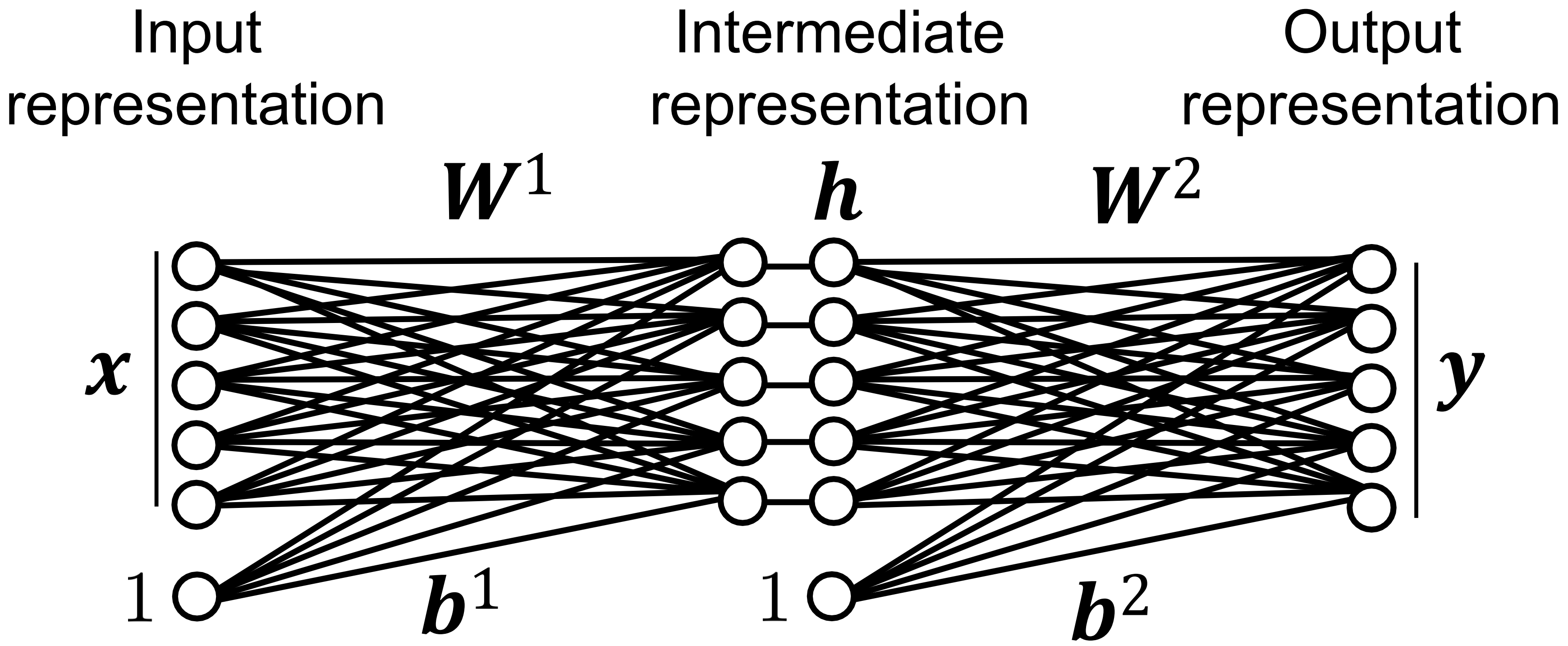


Fully connected layer



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

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

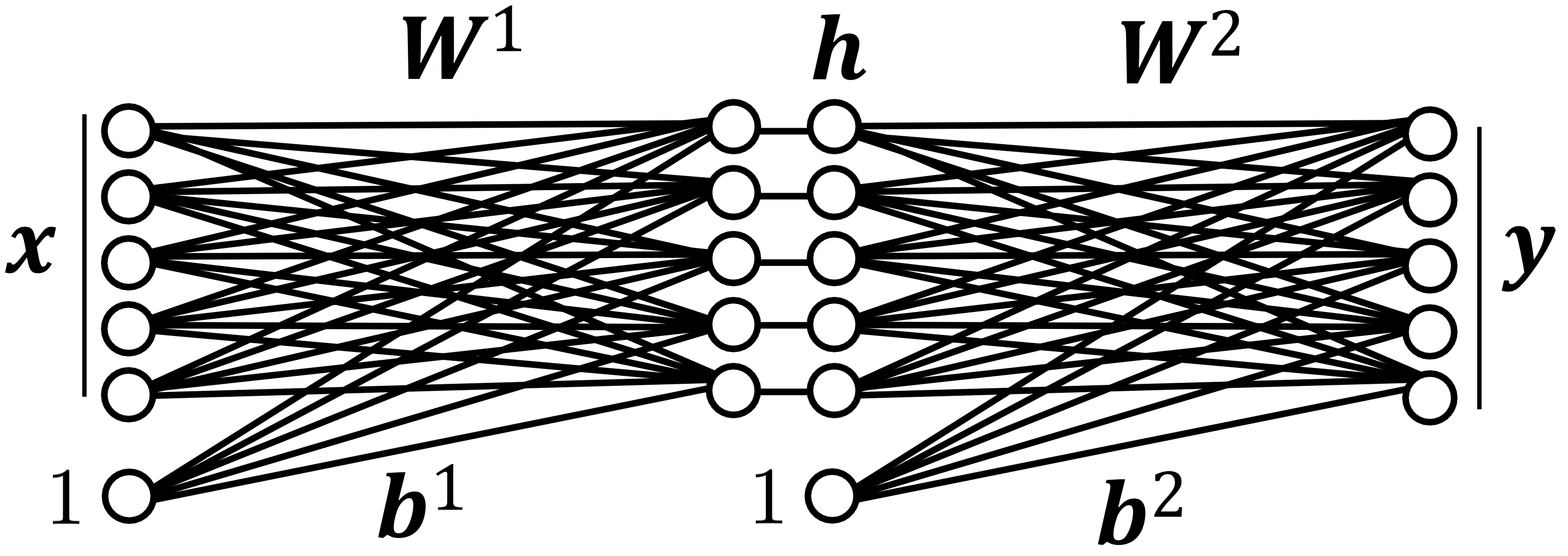
Source: Isola, Torralba, Freeman

Stacking layers

Input representation

Intermediate representation

Output representation



positive

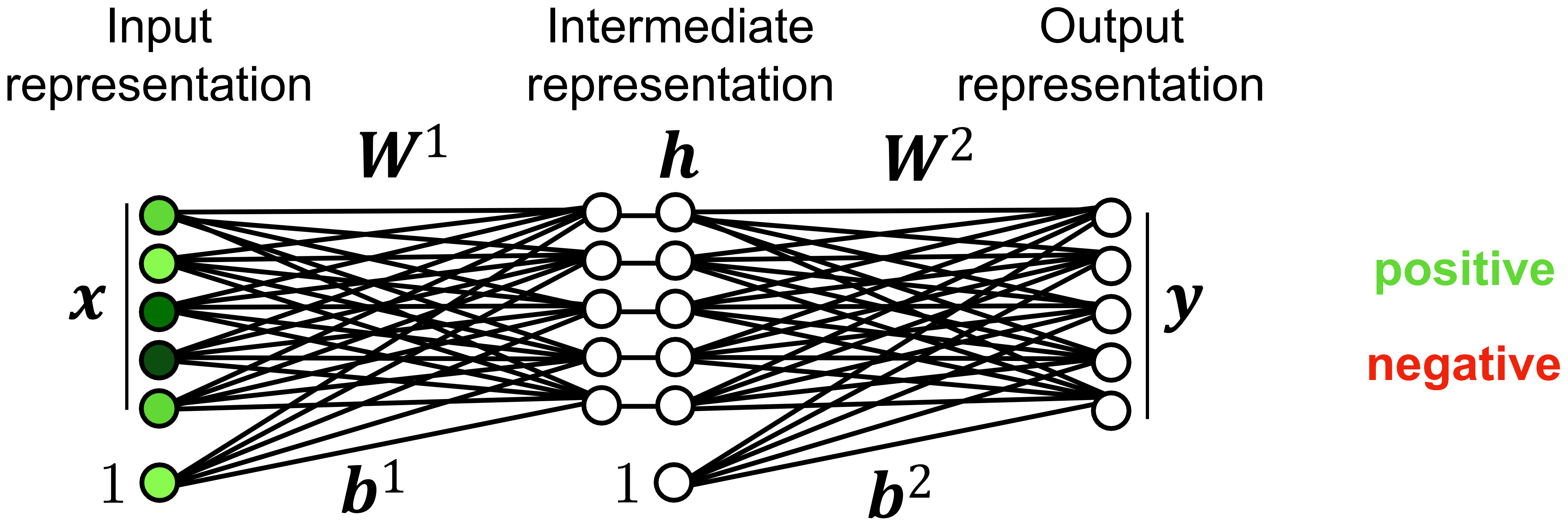
negative

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

ReLU

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

Stacking layers



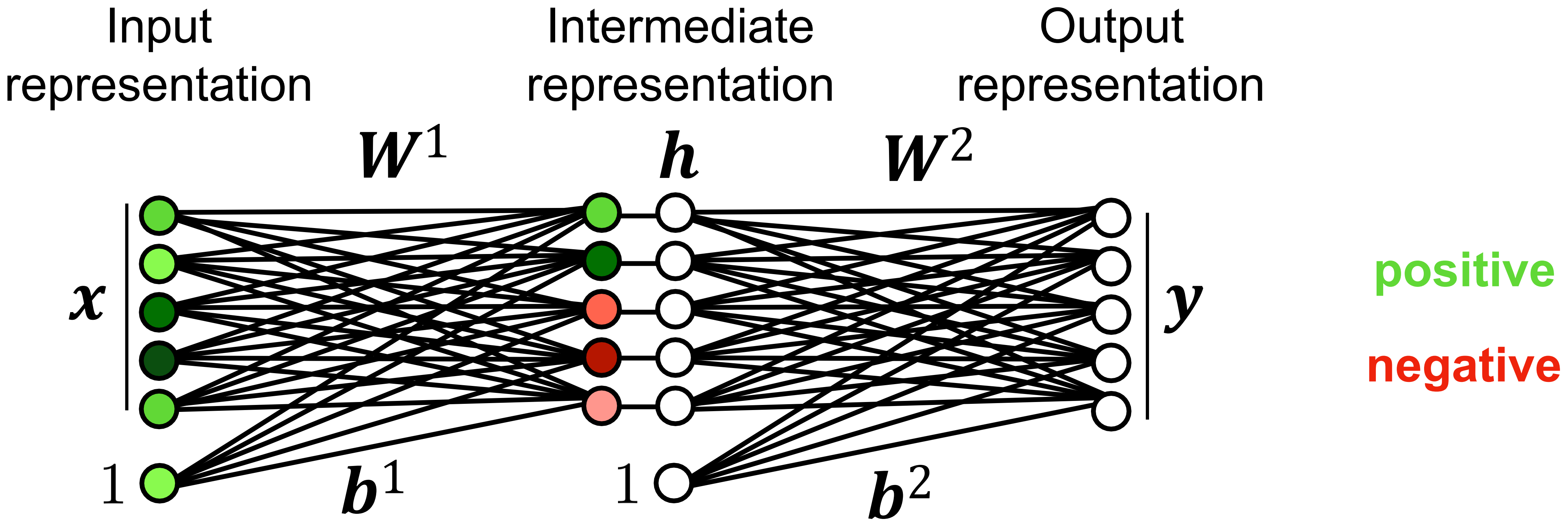
$$\boldsymbol{h} = g(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1) \quad \boldsymbol{y} = g(\boldsymbol{W}^2\boldsymbol{h} + \boldsymbol{b}^2)$$

ReLU

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

Source: Isola, Torralba, Freeman

Stacking layers

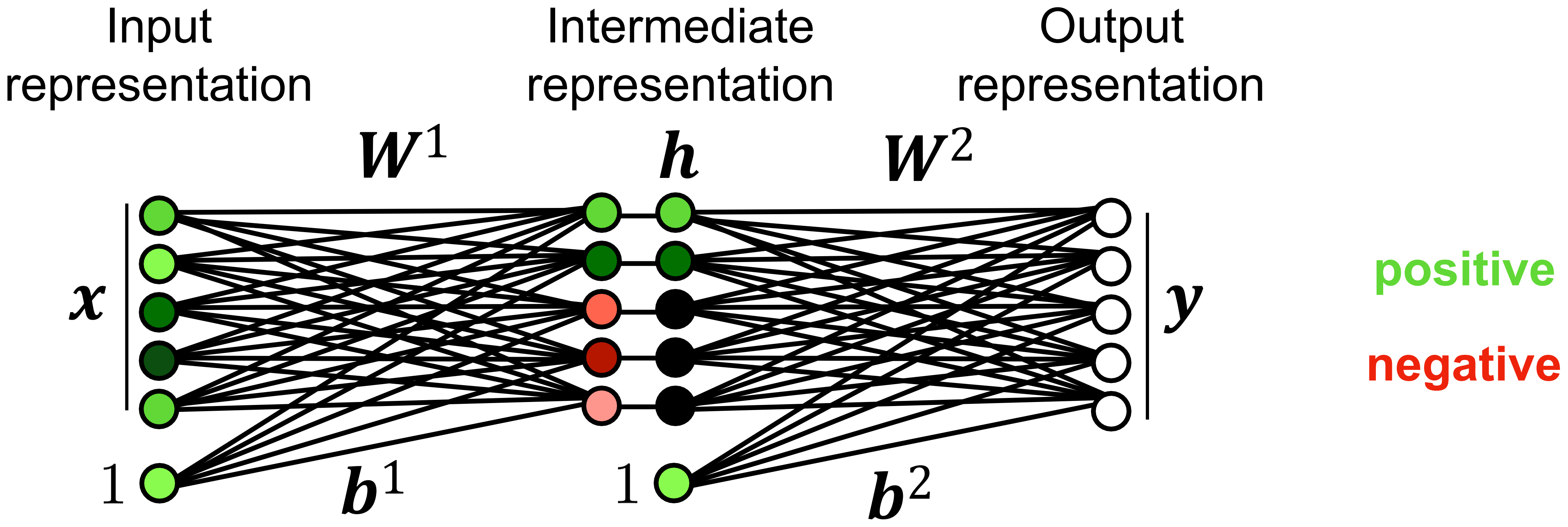


$$\boldsymbol{h} = g(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1) \qquad \boldsymbol{y} = g(\boldsymbol{W}^2\boldsymbol{h} + \boldsymbol{b}^2)$$

ReLU

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

Stacking layers

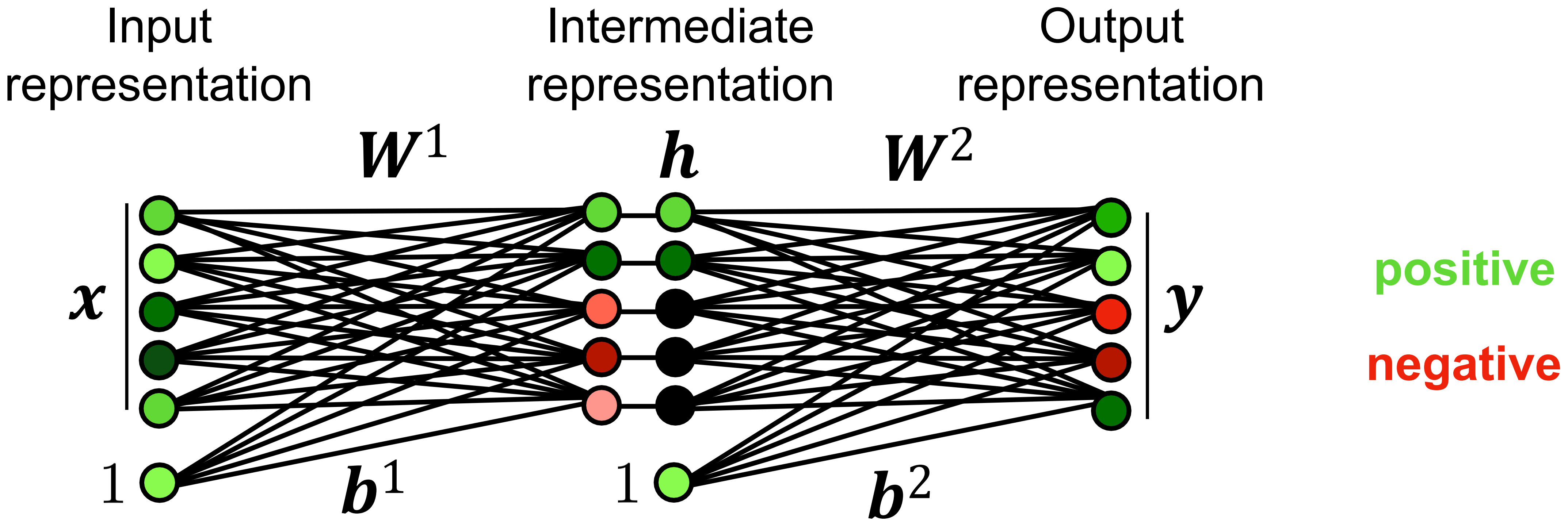


$$\boldsymbol{h} = g(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1) \quad \boldsymbol{y} = g(\boldsymbol{W}^2\boldsymbol{h} + \boldsymbol{b}^2)$$

ReLU

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

Stacking layers

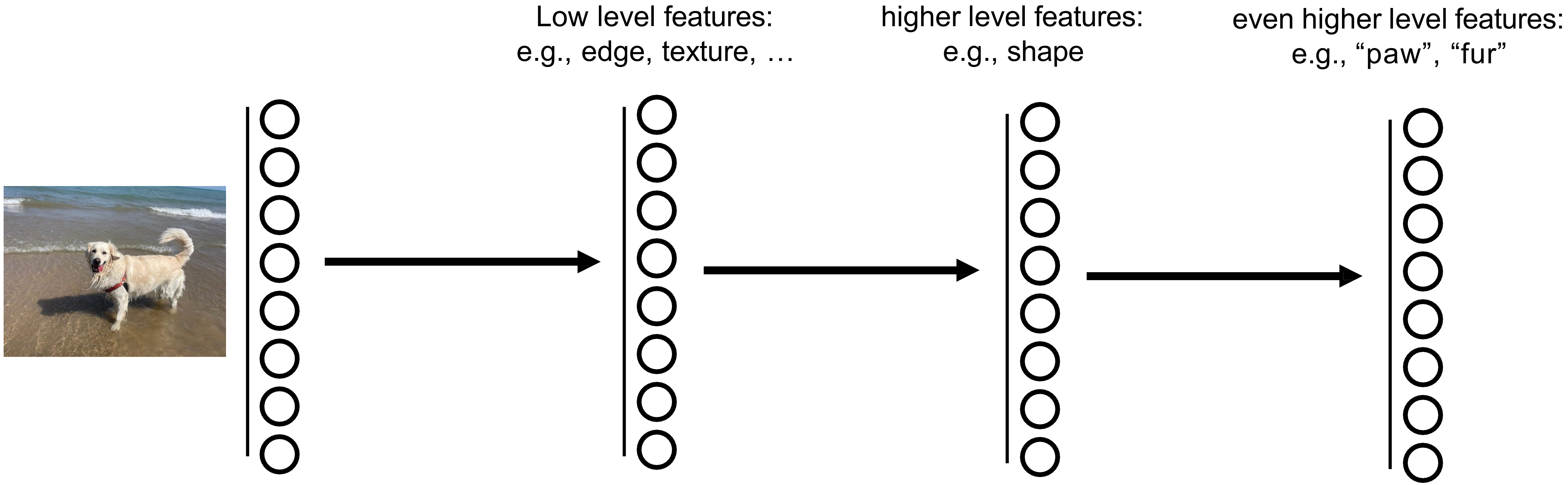


$$\boldsymbol{h} = g(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1) \quad \boldsymbol{y} = g(\boldsymbol{W}^2\boldsymbol{h} + \boldsymbol{b}^2)$$

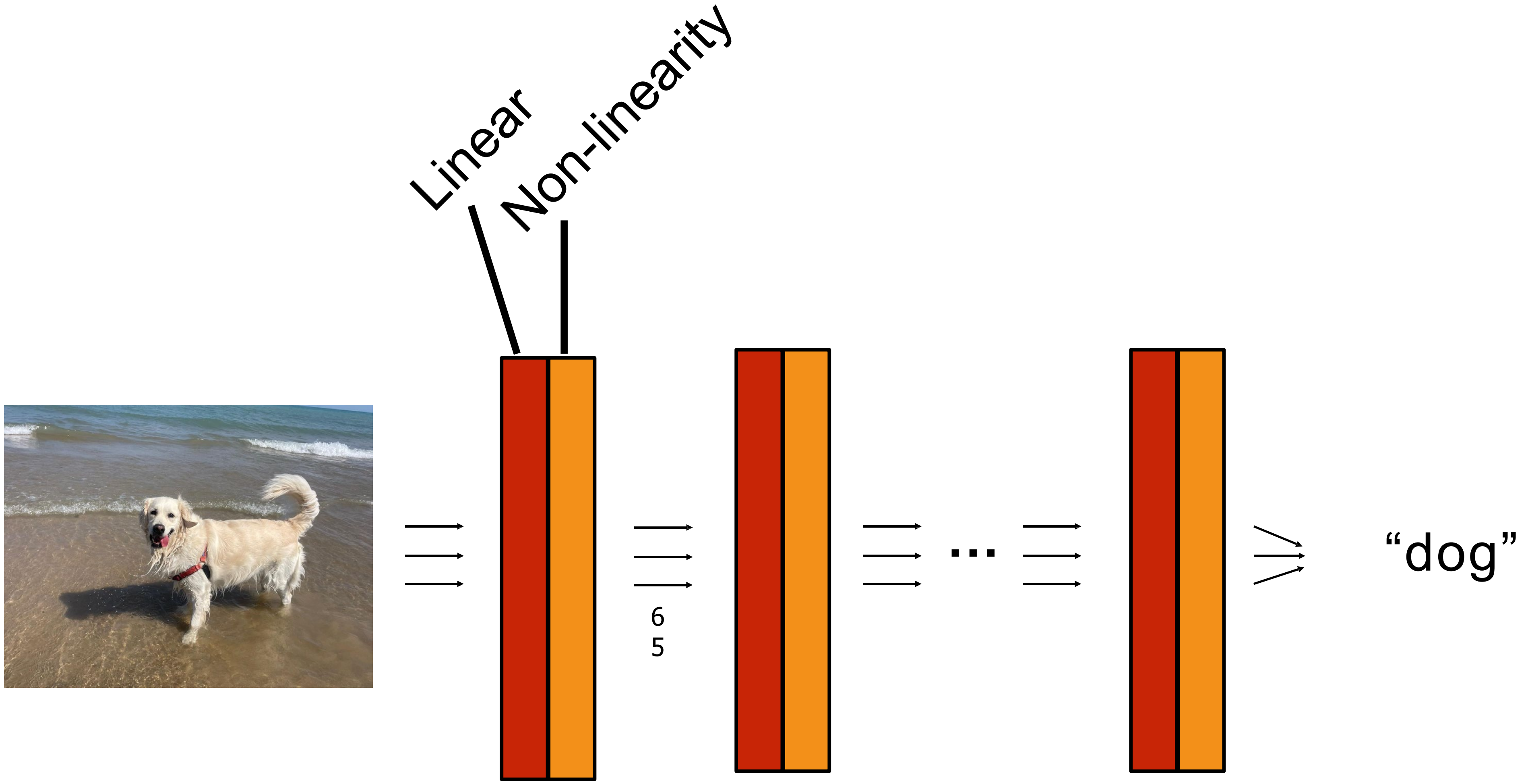
ReLU

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

Stacking layers - What’s actually happening?



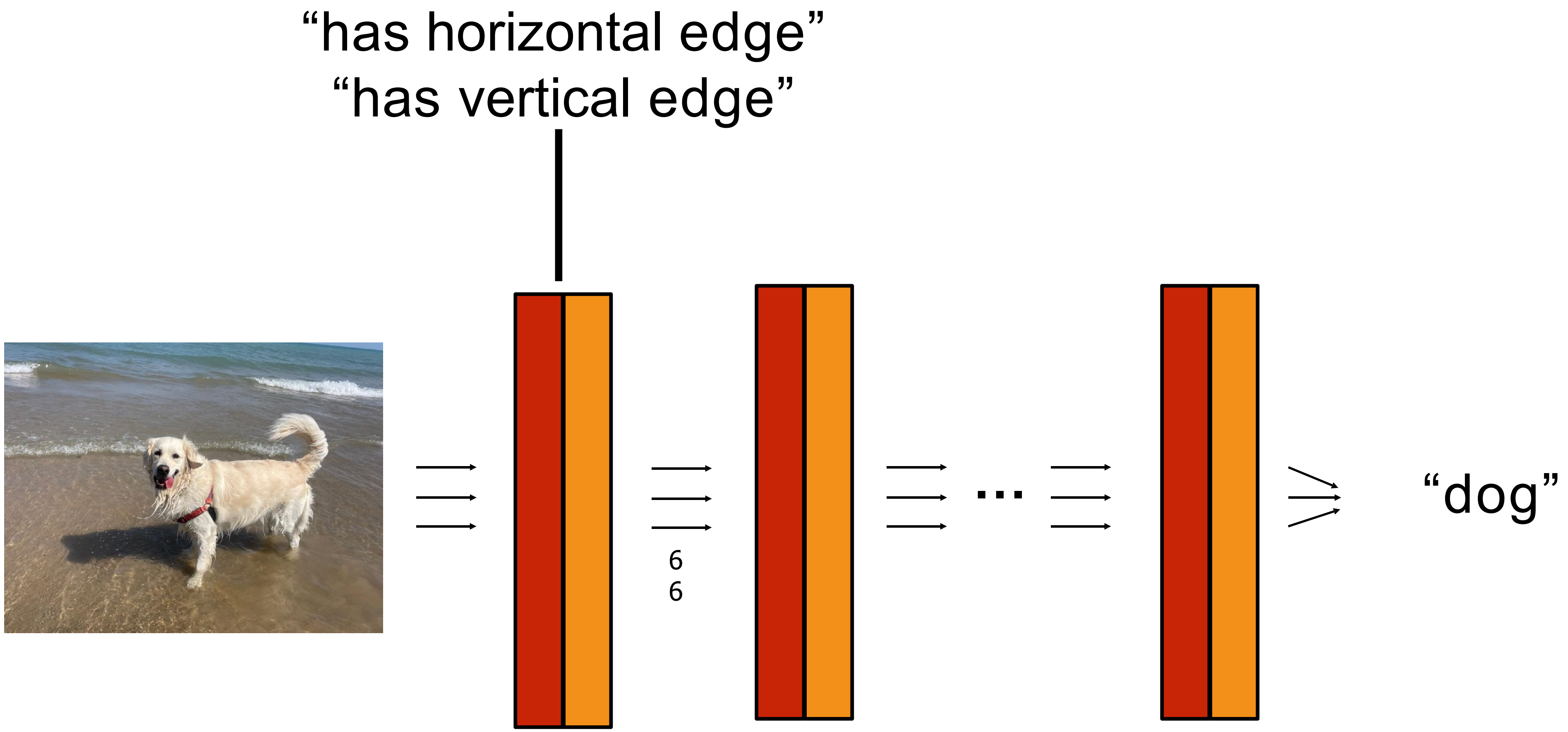
Deep nets



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

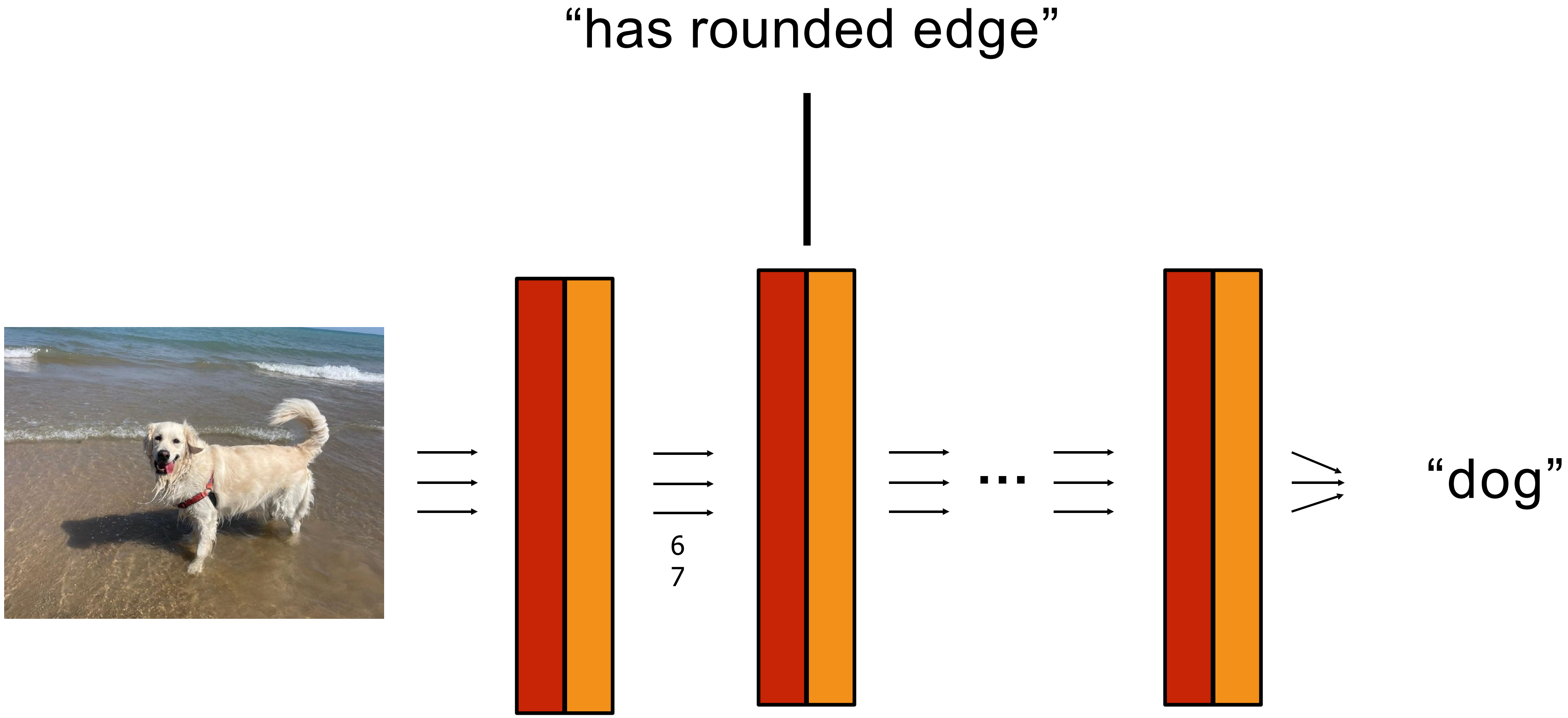
Source: Isola, Torralba, Freeman

Deep nets - Intuition



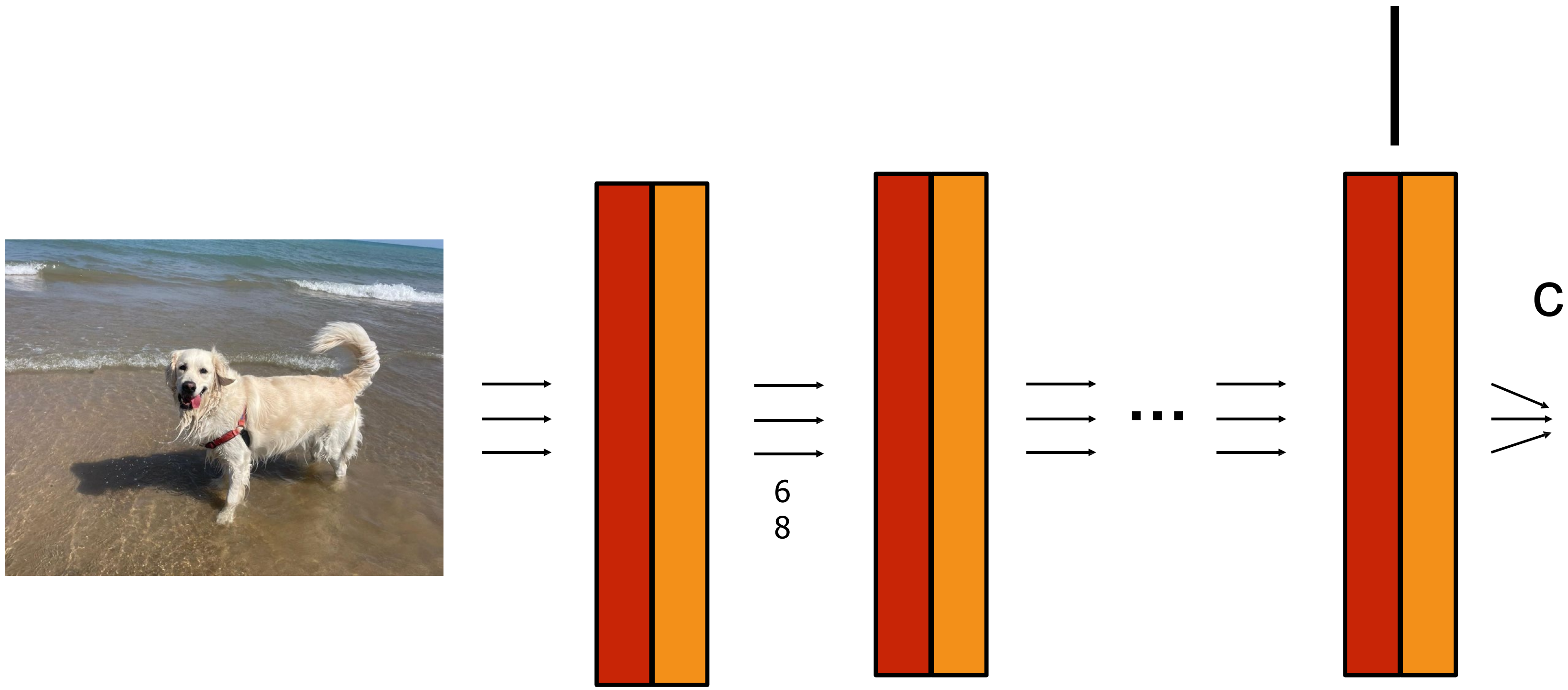
Source: Isola, Torralba, Freeman

Deep nets - Intuition



Source: Isola, Torralba, Freeman

Deep nets - Intuition

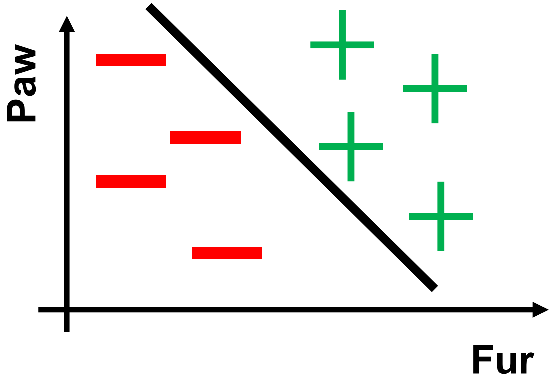


Source: Isola, Torralba, Freeman

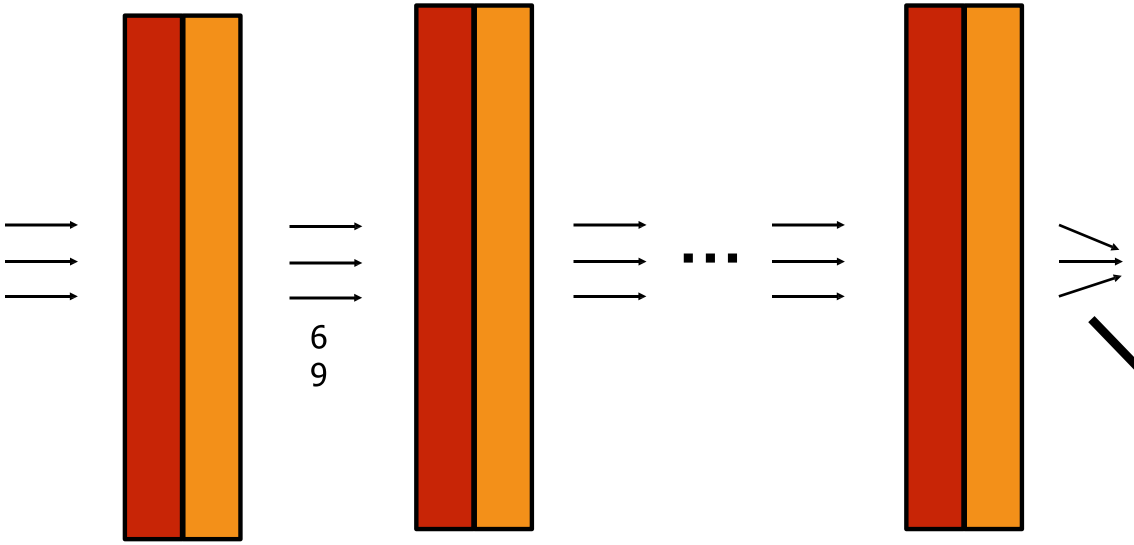
Deep nets - Intuition

Recall:

Feature Space



“has white fur”
“has paw”
etc



“dog”

Classify

Source: Isola, Torralba, Freeman

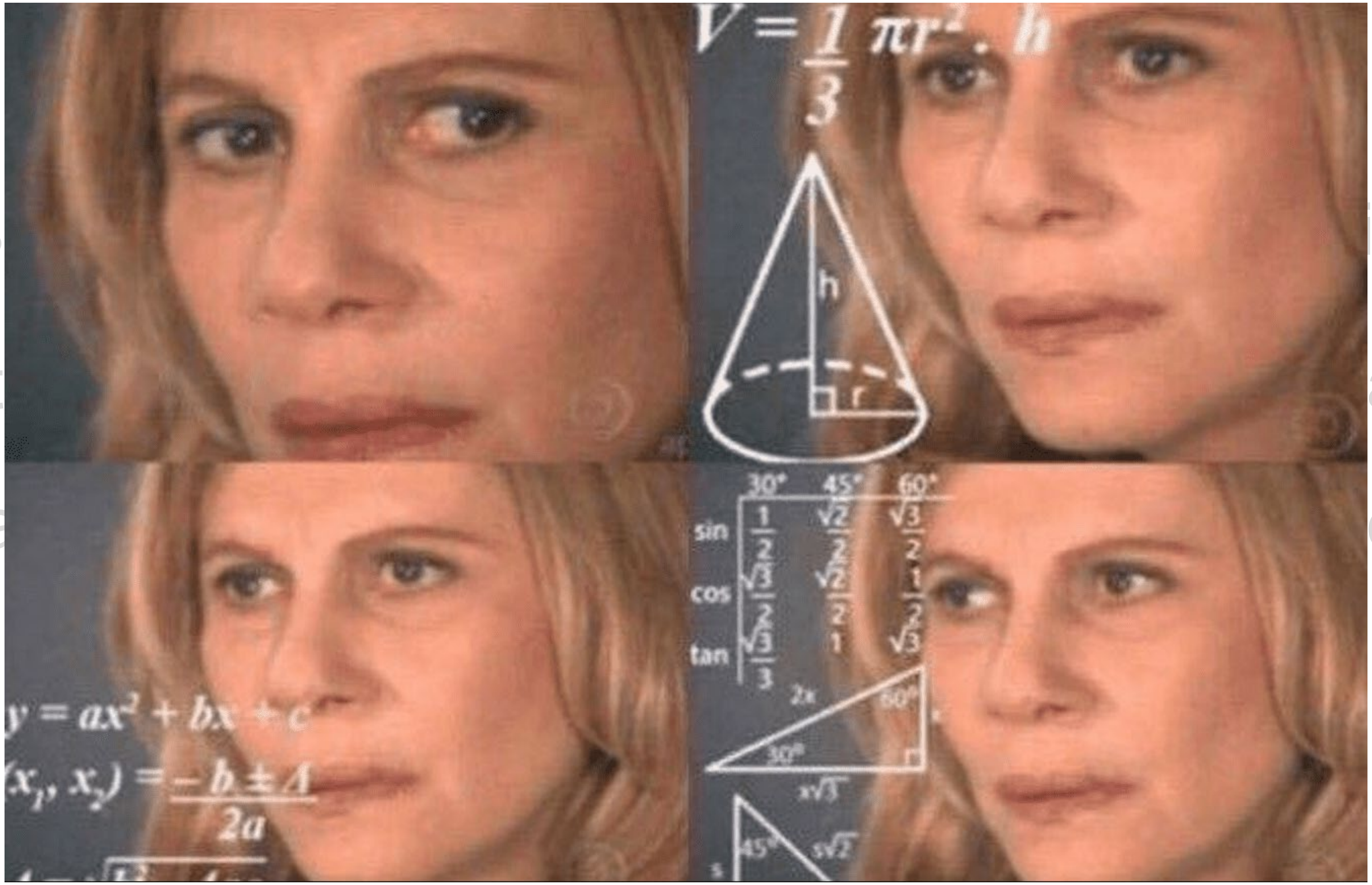
Computation has a simple form

- 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?

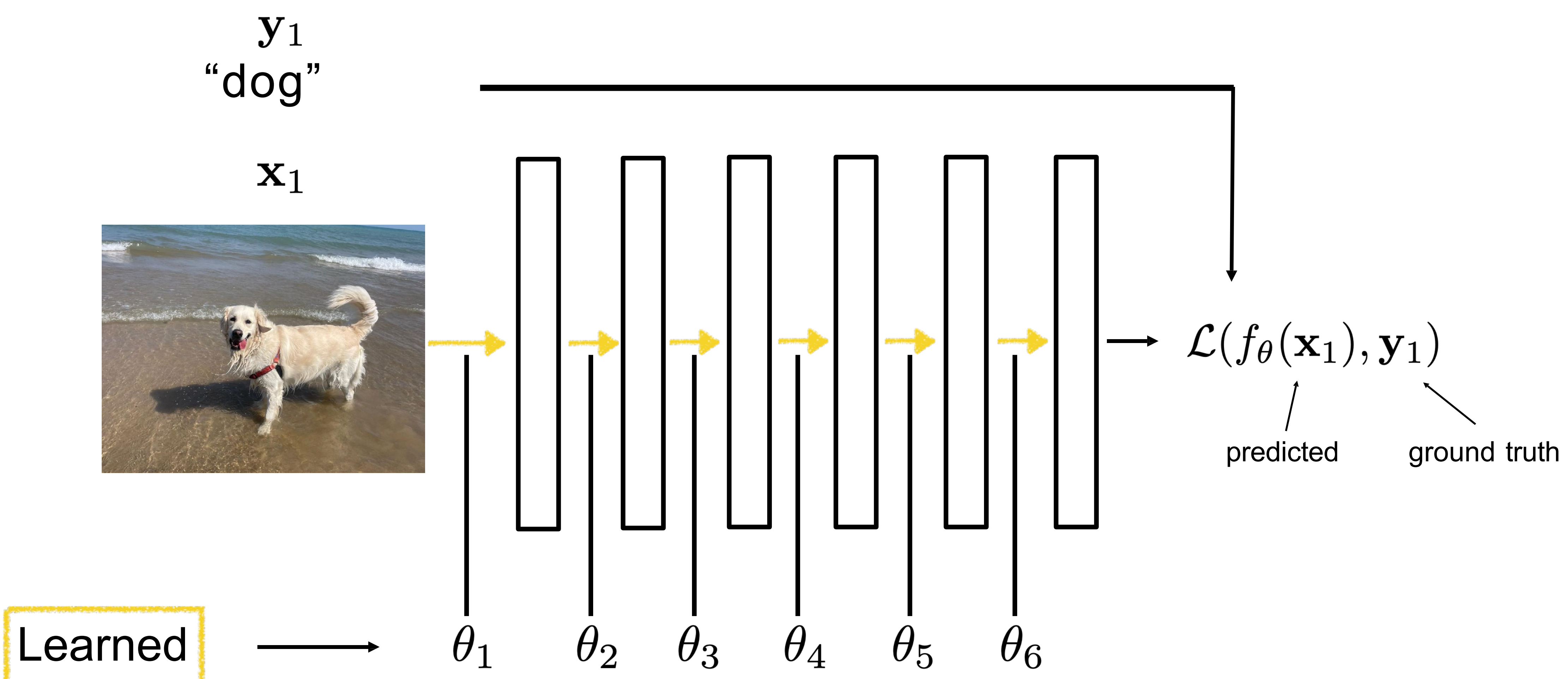
Computation has a simple form

- 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?

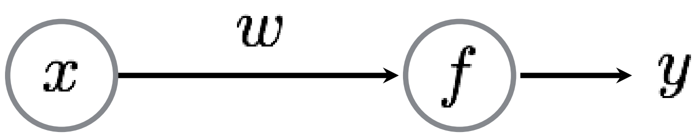


$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)$$


LEFT
EXIT 12
global minima
local minima
*gradient descent

Let’s start easy

world's smallest neural network!
(a "perceptron")



$$y = wx$$

(a.k.a. line equation, linear regression)

Training a Neural Network

Given a set of samples and a Perceptron

$$\{x_i, y_i\}$$

$$y = f_{\mathrm{PER}}(x; w)$$

Estimate the parameter of the Perceptron

$$w$$

Given training data:

x	y
10	10.1
2	1.9
3.5	3.4
1	1.1

What do you think the weight parameter is?

$$y = wx$$

Given training data:

x	y
10	10.1
2	1.9
3.5	3.4
1	1.1

What do you think the weight parameter is?

$$y = wx$$

not so obvious as the network gets more complicated so we use …

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

Modify weight w such that $\hat{y}$ gets **'closer'** to y

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

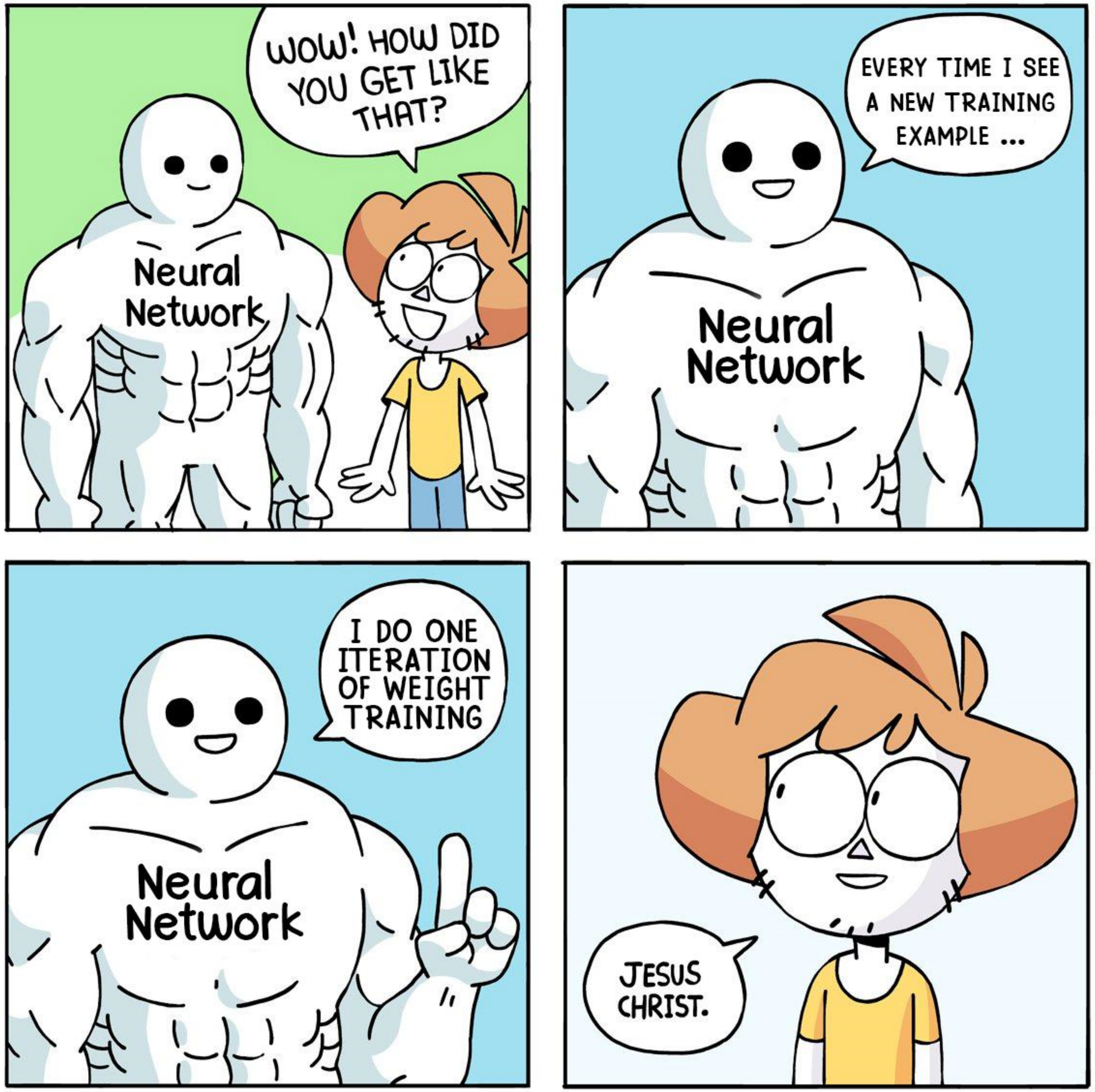
$$\hat{y} = wx$$

Modify weight w such that $\hat{y}$ gets **'closer'** to y

perceptron parameter

perceptron output

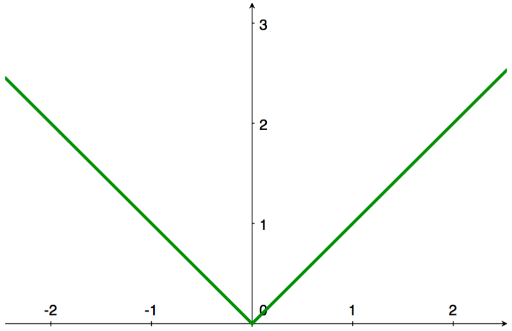
true label


WOW! HOW DID YOU GET LIKE THAT?
Neural Network
EVERY TIME I SEE A NEW TRAINING EXAMPLE ...
Neural Network
I DO ONE ITERATION OF WEIGHT TRAINING
Neural Network
JESUS CHRIST.

SHEN COMIX

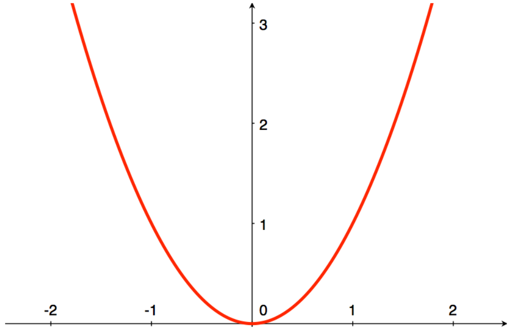
L1 Loss

$$\ell(\hat{y}, y) = |\hat{y} - y|$$



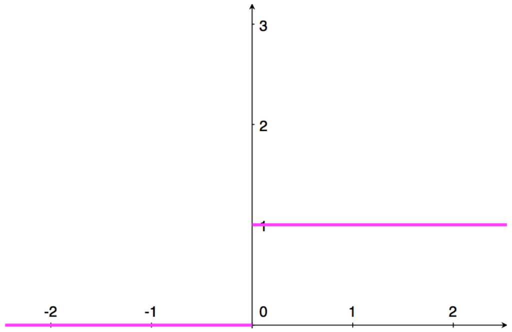
L2 Loss

$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$



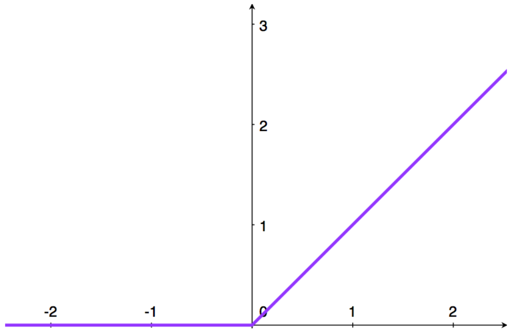
Zero-One Loss

$$\ell(\hat{y}, y) = \mathbf{1}[\hat{y} = y]$$

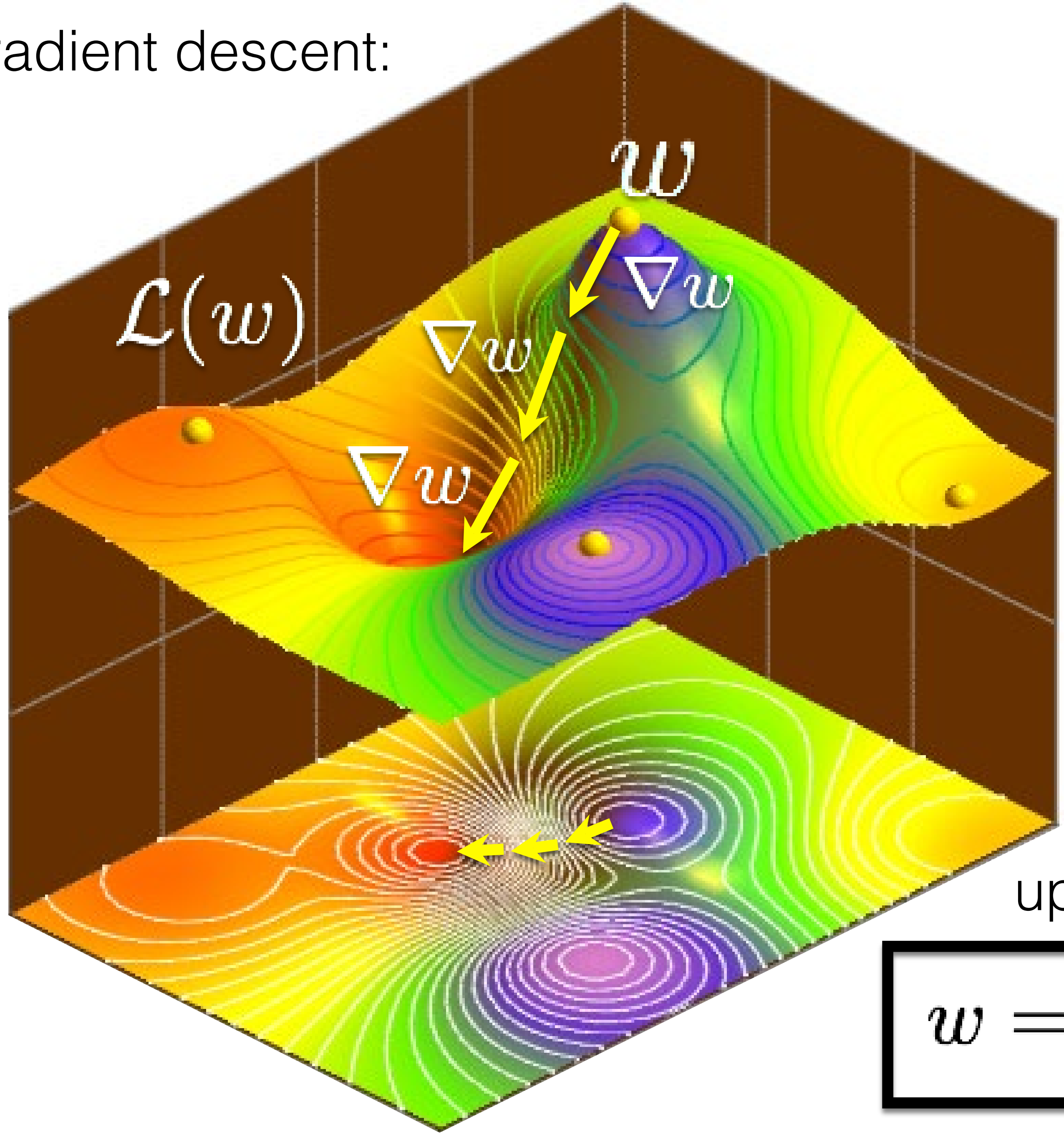


Hinge Loss

$$\ell(\hat{y}, y) = \max(0, 1 - y \cdot \hat{y})$$



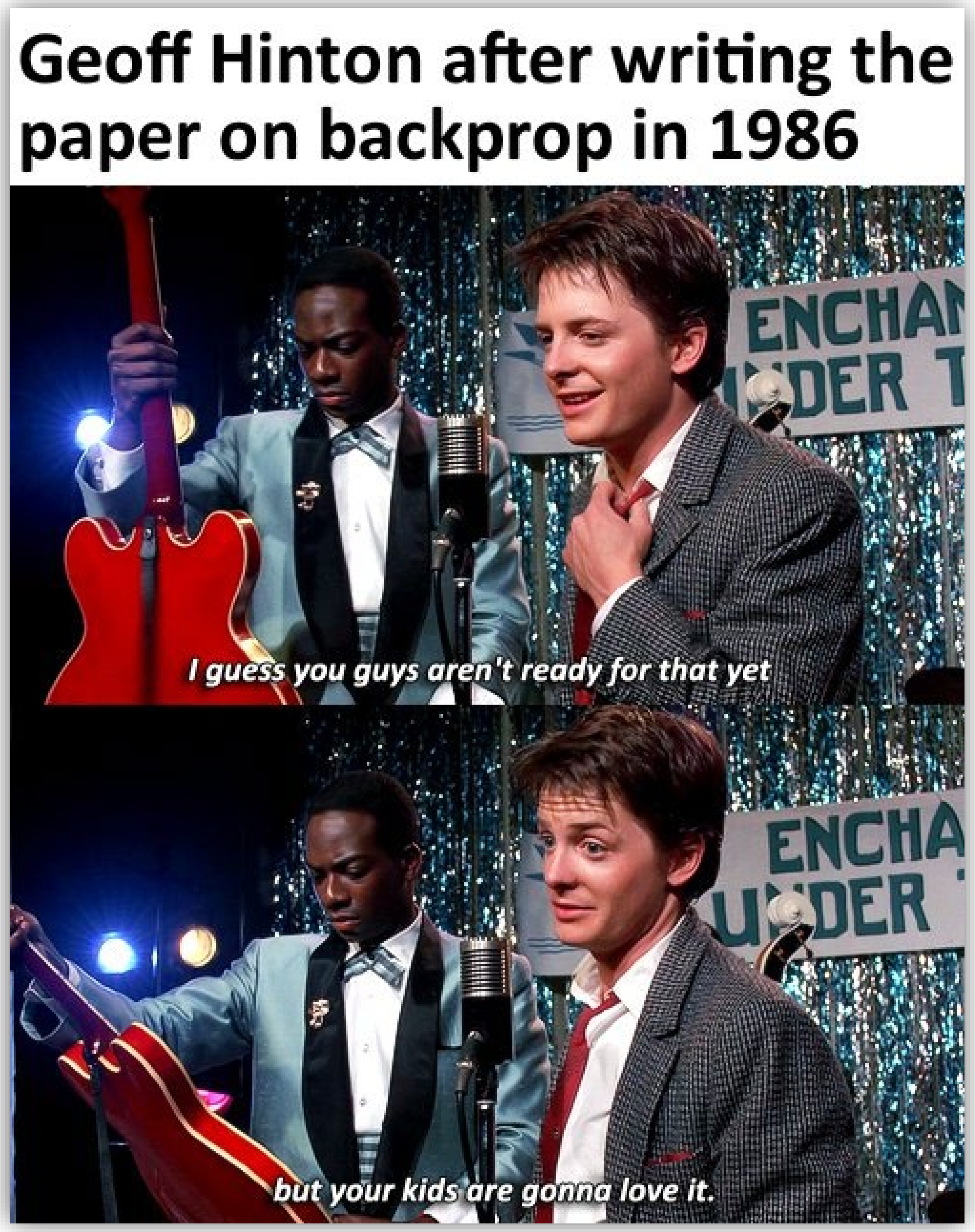
Gradient descent:



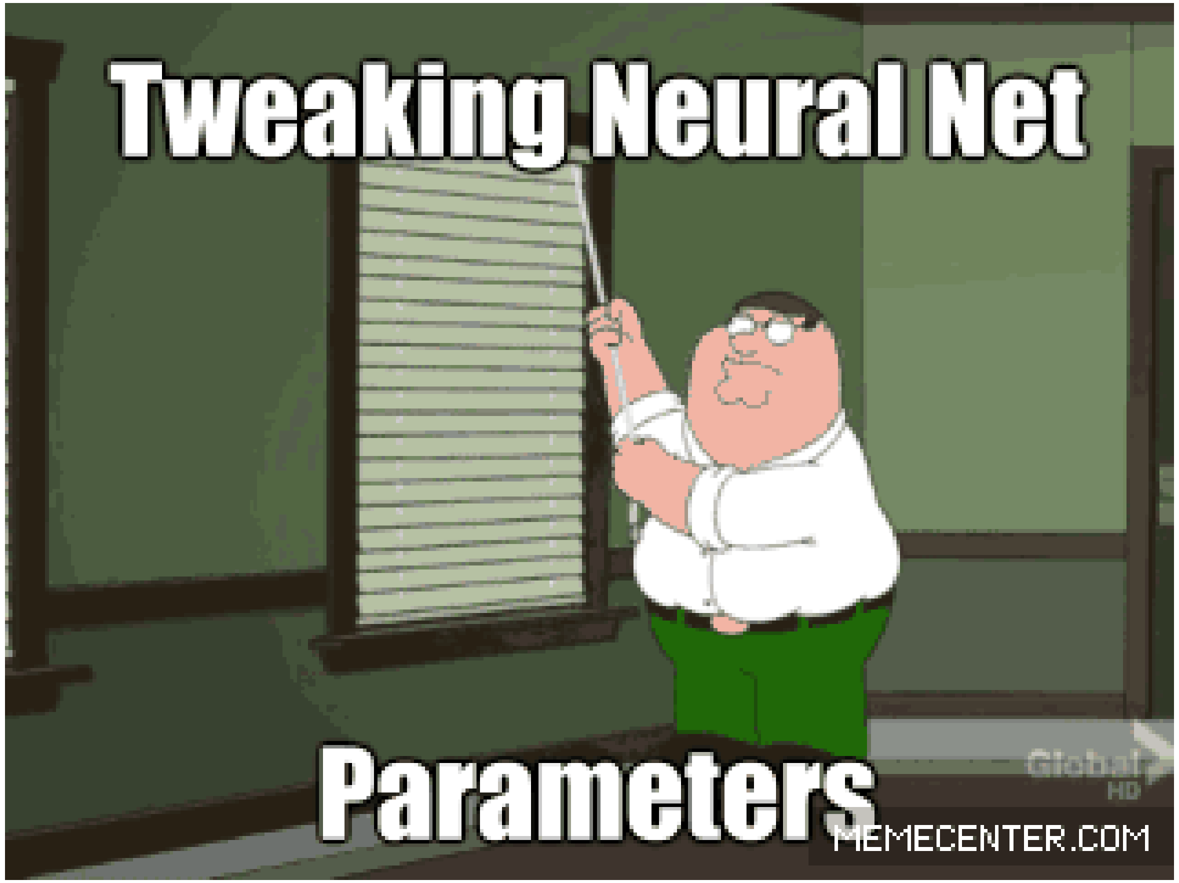
update rule:

$$\boxed{w = w - \nabla w}$$

Backpropagation



Backpropagation



$$\frac{d\mathcal{L}}{dw}$$ …is the rate at which **this** will change…

$$\mathcal{L} = \frac{1}{2}(y - \hat{y})^2$$

the loss function

… per unit change of **this**

$$y = wx$$

the weight parameter

Let's compute the derivative…

Compute the derivative

$$
\begin{aligned}
\frac{d\mathcal{L}}{dw} &= \frac{d}{dw}\left\{\frac{1}{2}(y-\hat{y})^2\right\} \\
&= -(y-\hat{y})\frac{dwx}{dw} \\
&= -(y-\hat{y})x = \nabla w
\end{aligned}
$$

That means the weight update for **gradient descent** is:

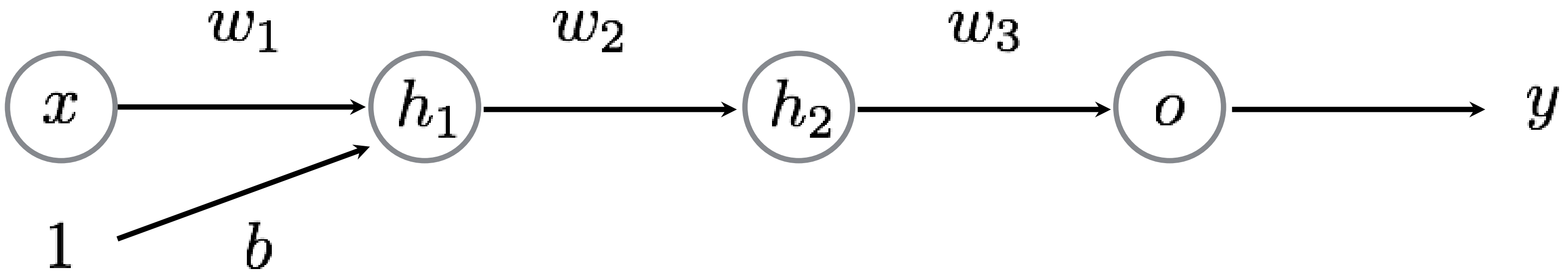
$$
\begin{aligned}
w &= w - \nabla w \quad \text{move in direction of negative gradient} \\
&= w + (y-\hat{y})x
\end{aligned}
$$

Gradient Descent (world's smallest perceptron)

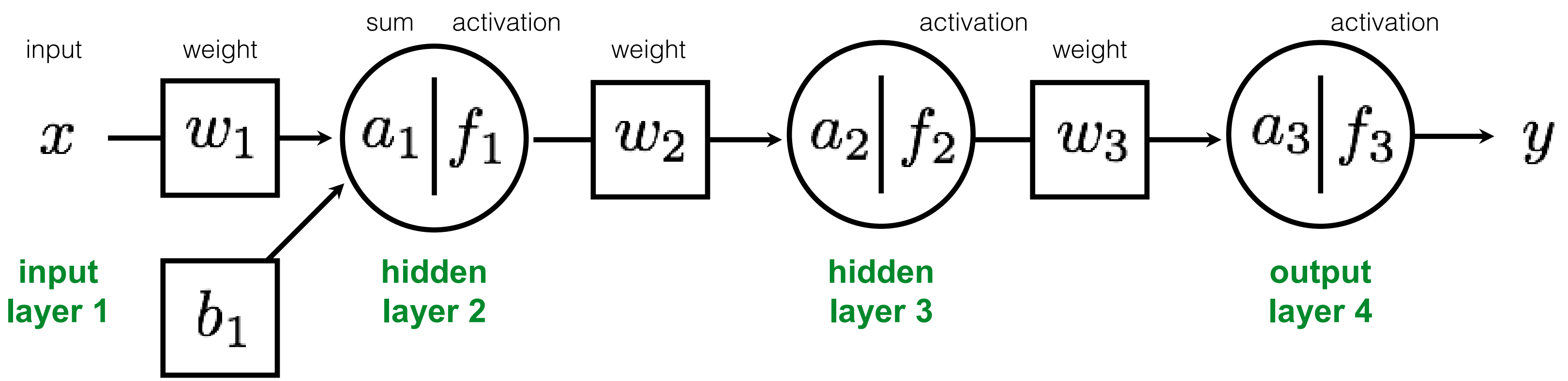
For each sample $\{x_i, y_i\}$

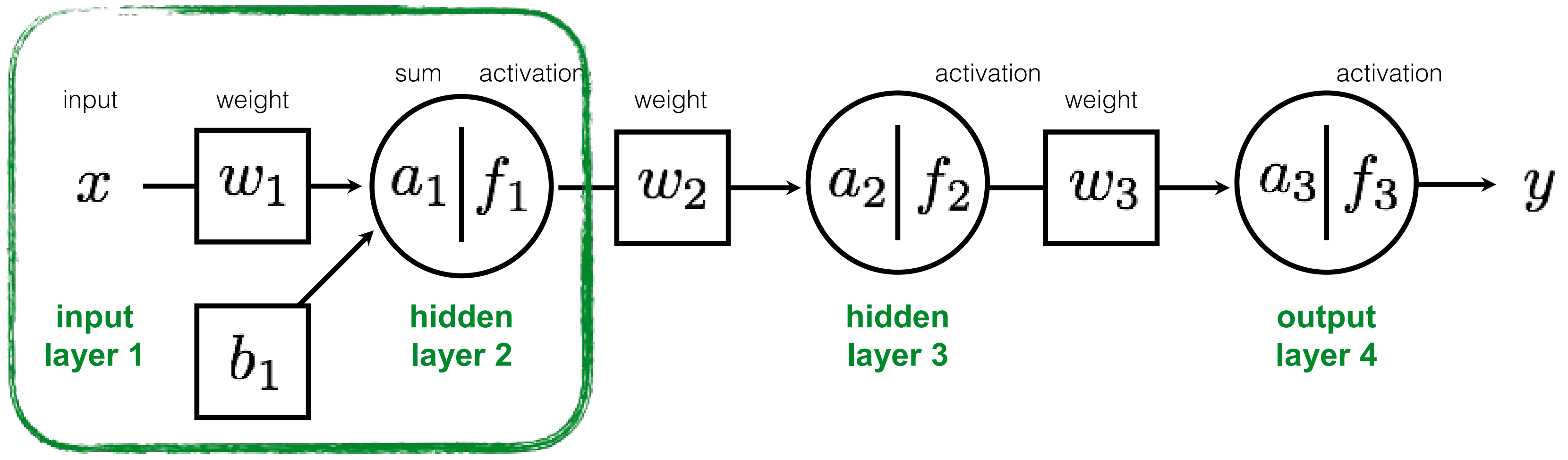
1. Predict
   a. Forward pass $\hat{y} = wx_i$
   b. Compute Loss $\mathcal{L}_i = \frac{1}{2}(y_i - \hat{y})^2$
2. Update
   a. Back Propagation $\frac{d\mathcal{L}_i}{dw} = -(y_i - \hat{y})x_i = \nabla w$
   b. Gradient update $w = w - \nabla w$

multi-layer perceptron

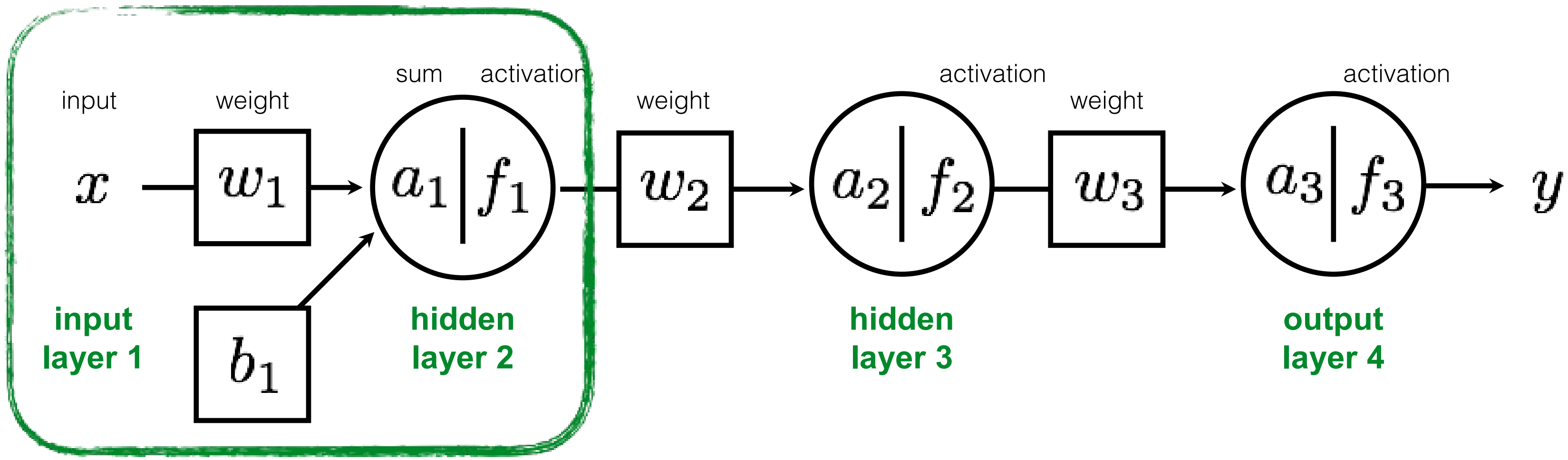


function of **FOUR** parameters and **FOUR** layers!

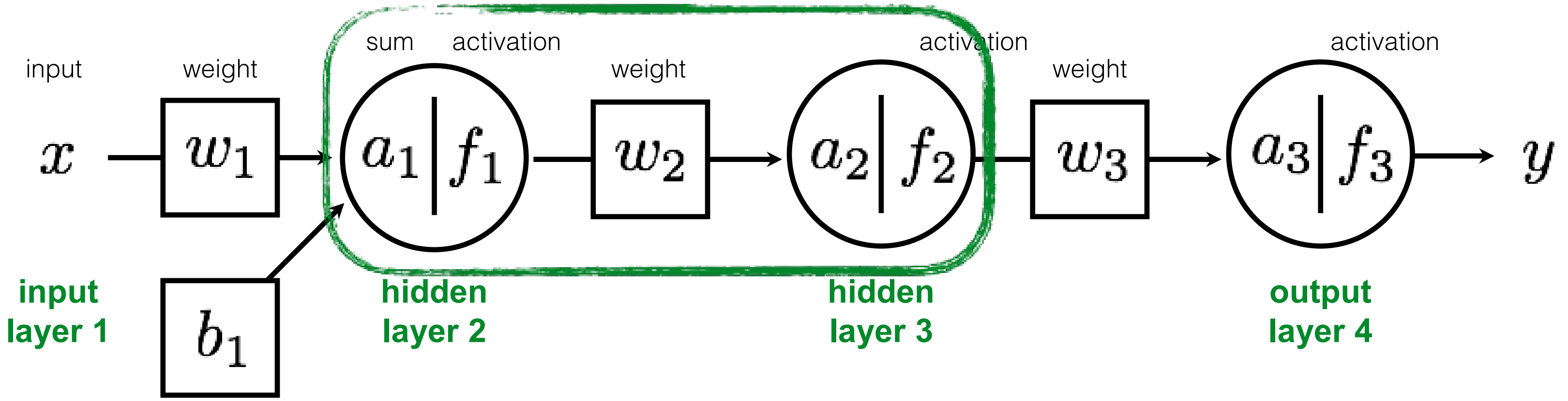

input
weight
sum
activation
weight
activation
weight
activation
x
w_1
a_1
f_1
w_2
a_2
f_2
w_3
a_3
f_3
y
b_1
input
layer 1
hidden
layer 2
hidden
layer 3
output
layer 4



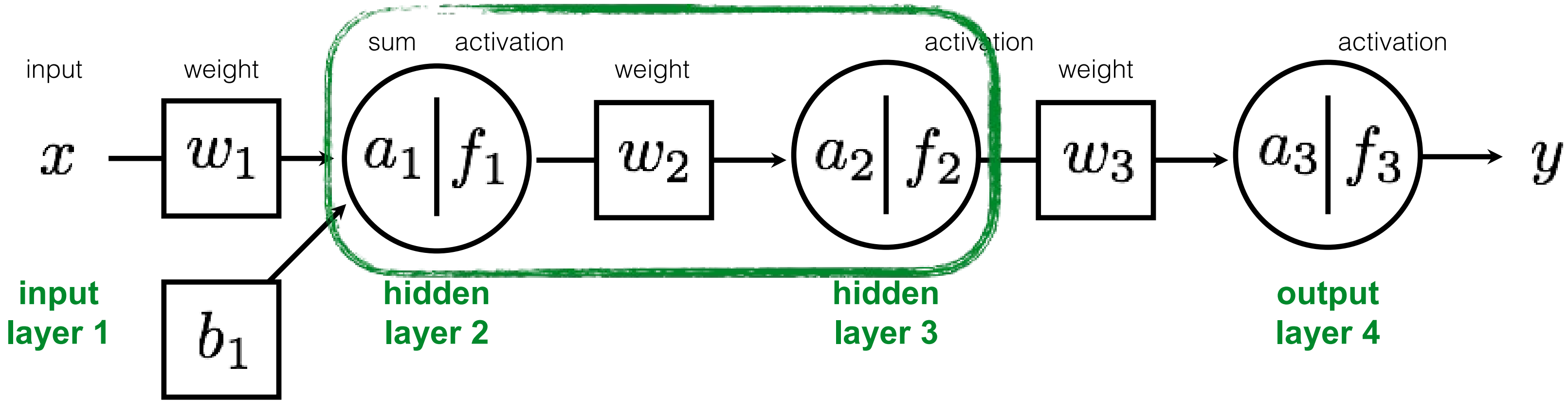
input
weight
sum
activation
weight
activation
weight
activation
x
w_1
a_1 | f_1
w_2
a_2 | f_2
w_3
a_3 | f_3
y
b_1
input layer 1
hidden layer 2
hidden layer 3
output layer 4



$$a_1 = w_1 \cdot x + b_1$$

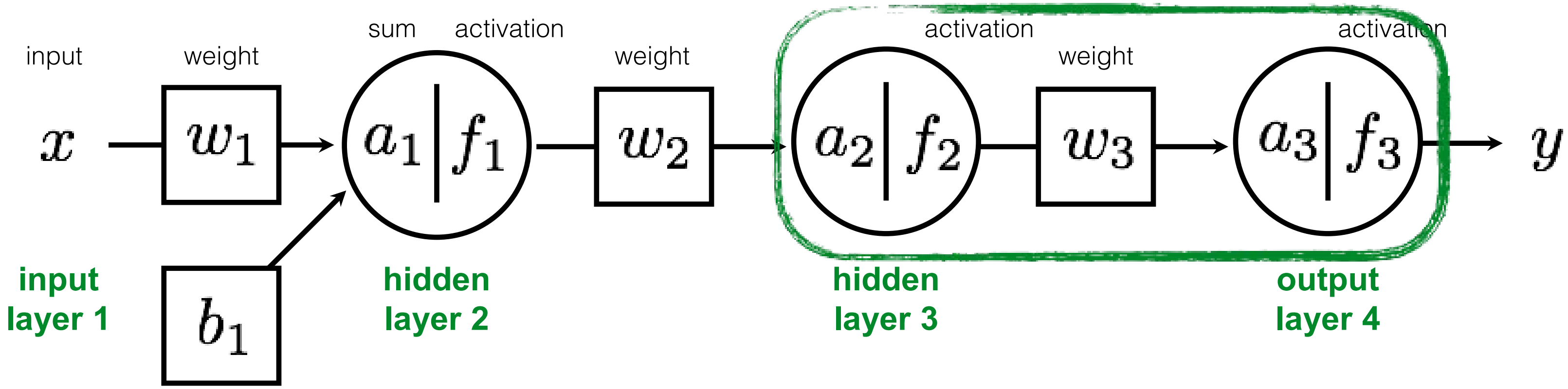


$$a_1 = w_1 \cdot x + b_1$$



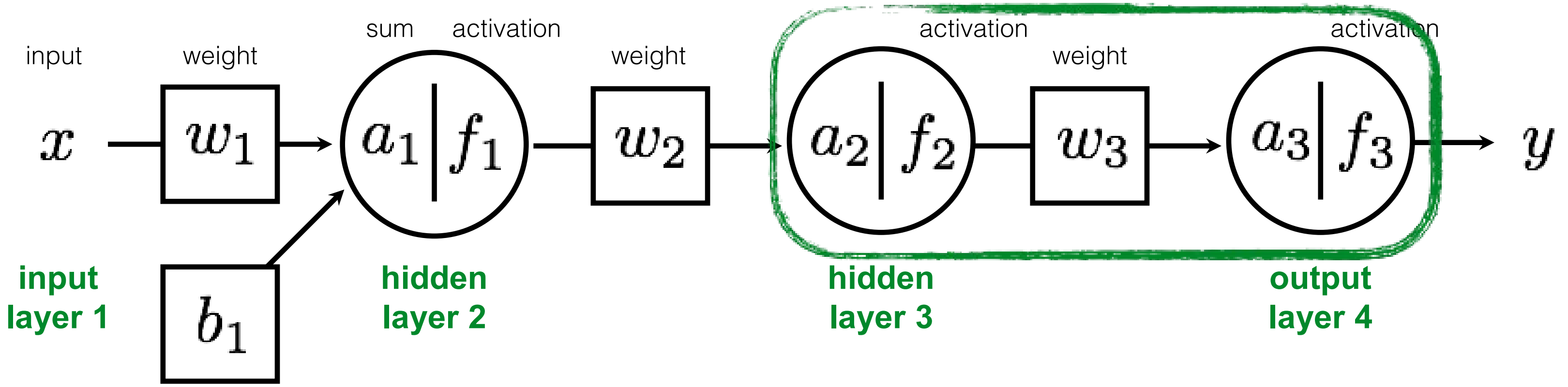
$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$



$$a_1 = w_1 \cdot x + b_1$$

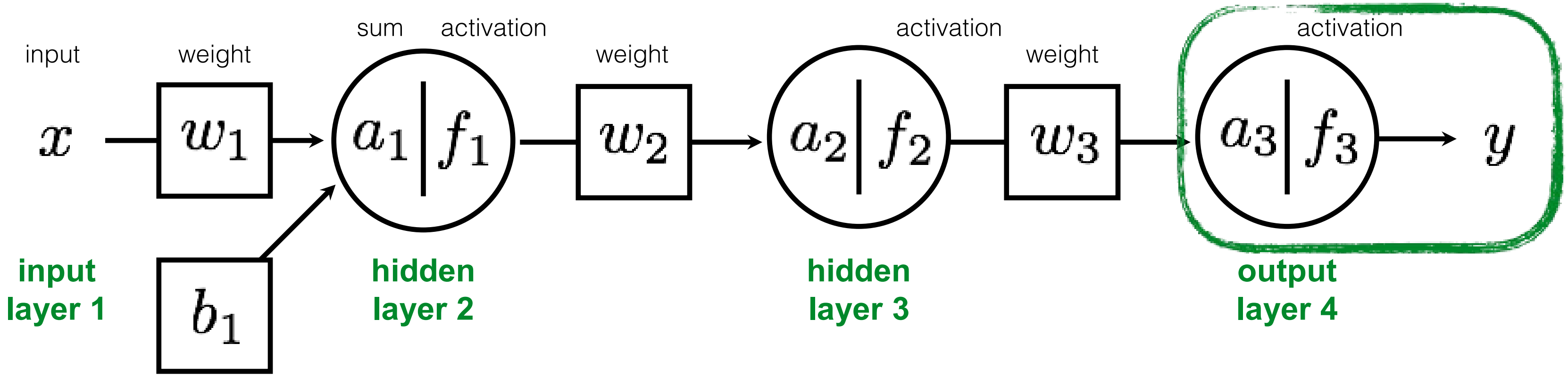
$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

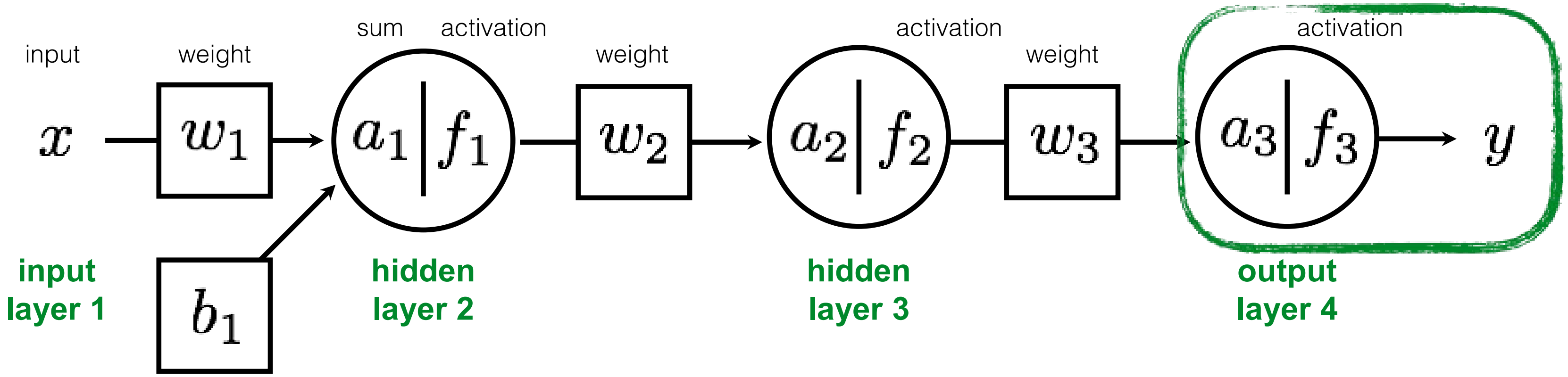
$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

Entire network can be written out as one long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

We need to train the network:

What is known? What is unknown?

Entire network can be written out as a long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

known

We need to train the network:

What is known? What is unknown?

Entire network can be written out as a long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

activation function
sometimes has
parameters

unknown

We need to train the network:

What is known? What is unknown?

Learning an MLP

Given a set of samples and a MLP

$$\{x_i, y_i\}$$

$$y = f_{\text{MLP}}(x; \theta)$$

Estimate the parameters of the MLP

$$\theta = \{f, w, b\}$$

Gradient Descent

For each **random** sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
   b. Compute Loss
2. Update
   a. Back Propagation $\frac{\partial \mathcal{L}}{\partial \theta}$ vector of parameter partial derivatives
   b. Gradient update $\theta \leftarrow \theta - \eta \nabla \theta$

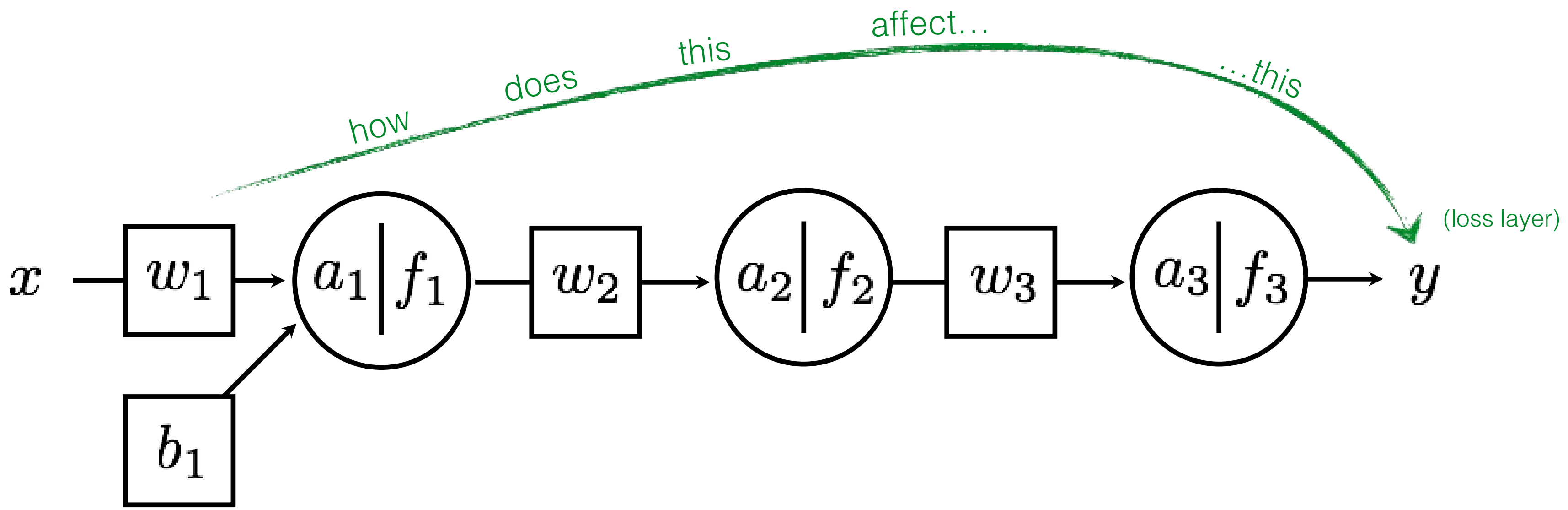
vector of parameter update equations

So we need to compute the partial derivatives

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \left[\frac{\partial \mathcal{L}}{\partial w_3} \frac{\partial \mathcal{L}}{\partial w_2} \frac{\partial \mathcal{L}}{\partial w_1} \frac{\partial \mathcal{L}}{\partial b} \right]$$

Remember,

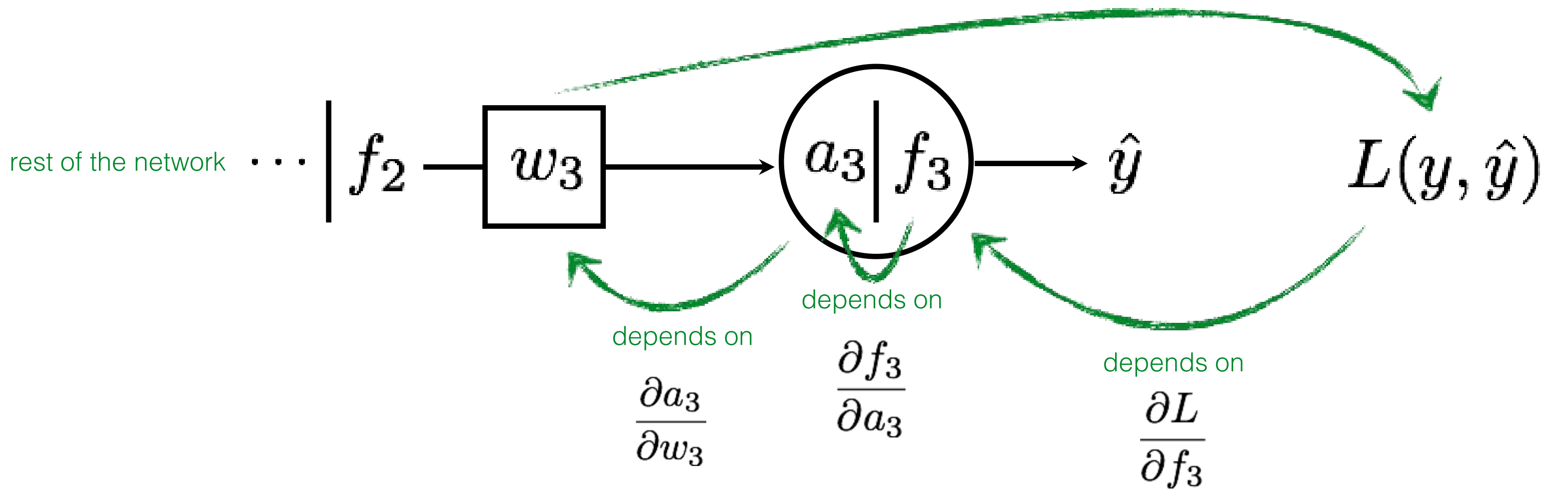
Partial derivative $\frac{\partial L}{\partial w_1}$ describes…

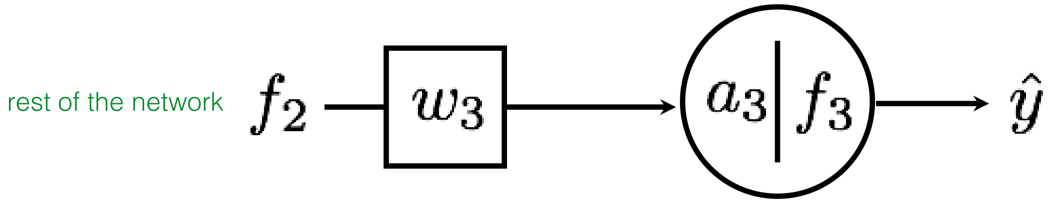


According to the chain rule…

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

Intuitively, the effect of weight on loss function : $\frac{\partial L}{\partial w_3}$

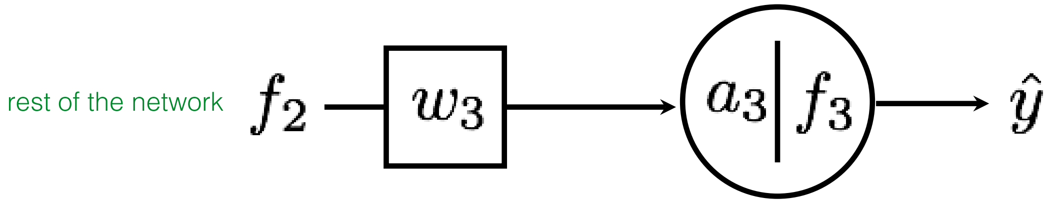




$$L(y, \hat{y})$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3}$$

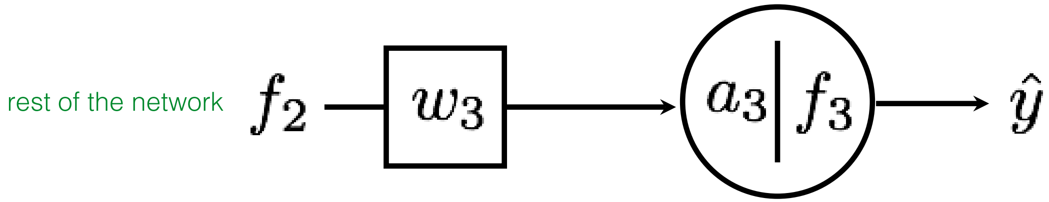
Chain Rule!



$$L(y, \hat{y})$$

$$\begin{aligned}\frac{\partial L}{\partial w_3} &= \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3} \\ &= -\eta(y - \hat{y})\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}\end{aligned}$$

Just the partial
derivative of L2 loss



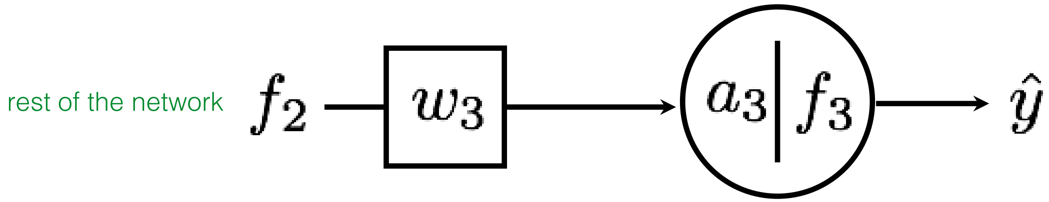
$$L(y, \hat{y})$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$= -\eta(y - \hat{y})\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

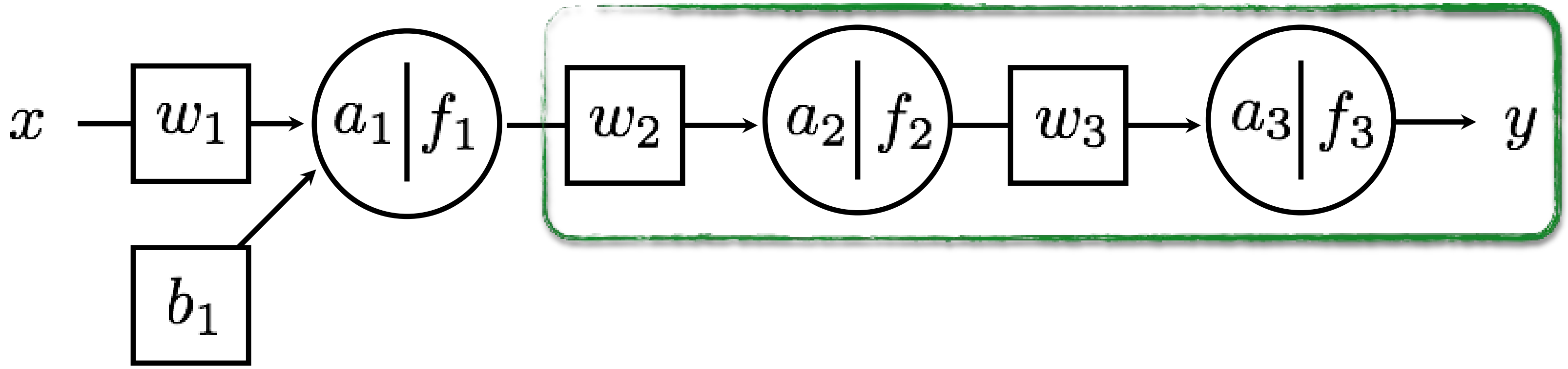
Let’s use a Sigmoid function

$$\frac{ds(x)}{dx} = s(x)(1 - s(x))$$

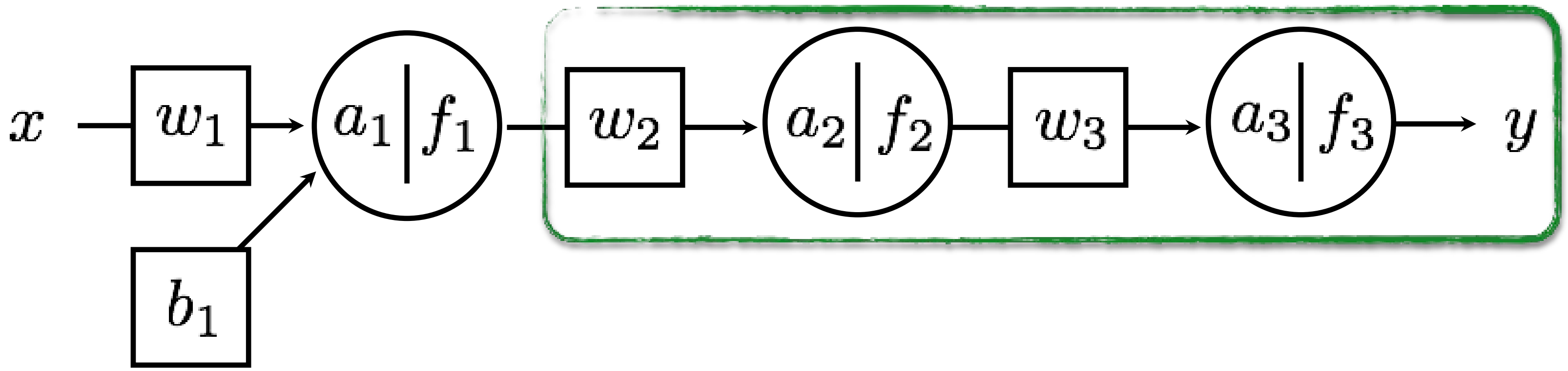


$L(y, \hat{y})$

$$
\begin{aligned}
\frac{\partial L}{\partial w_3} &= \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3} \\
&= -\eta(y-\hat{y})\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3} \\
&= -\eta(y-\hat{y})f_3(1-f_3)\frac{\partial a_3}{\partial w_3} \\
&= -\eta(y-\hat{y})f_3(1-f_3)f_2
\end{aligned}
$$



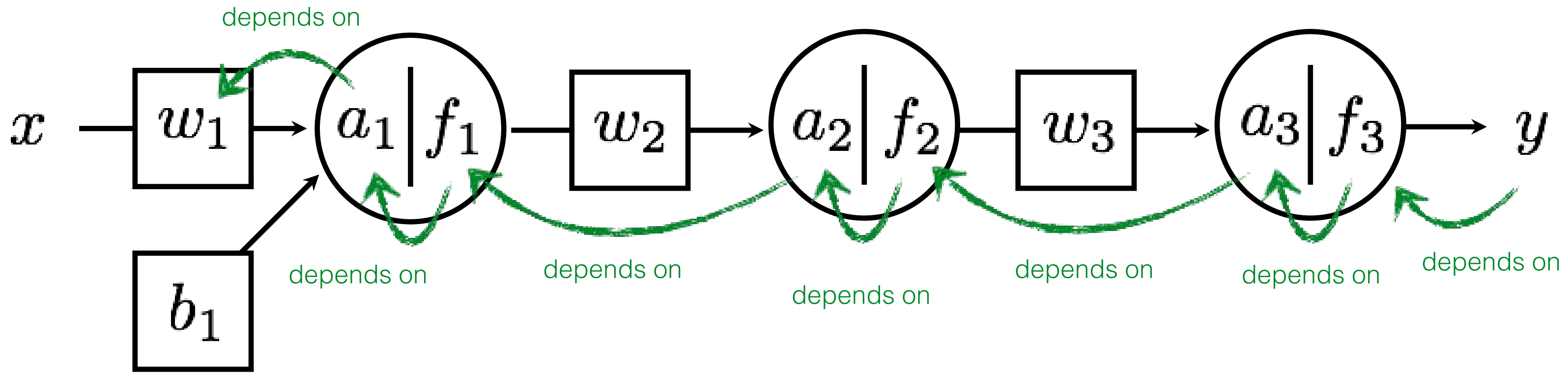
$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial w_2}$$



$$\frac{\partial L}{\partial w_2} = \boxed{\frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3}} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial w_2}$$

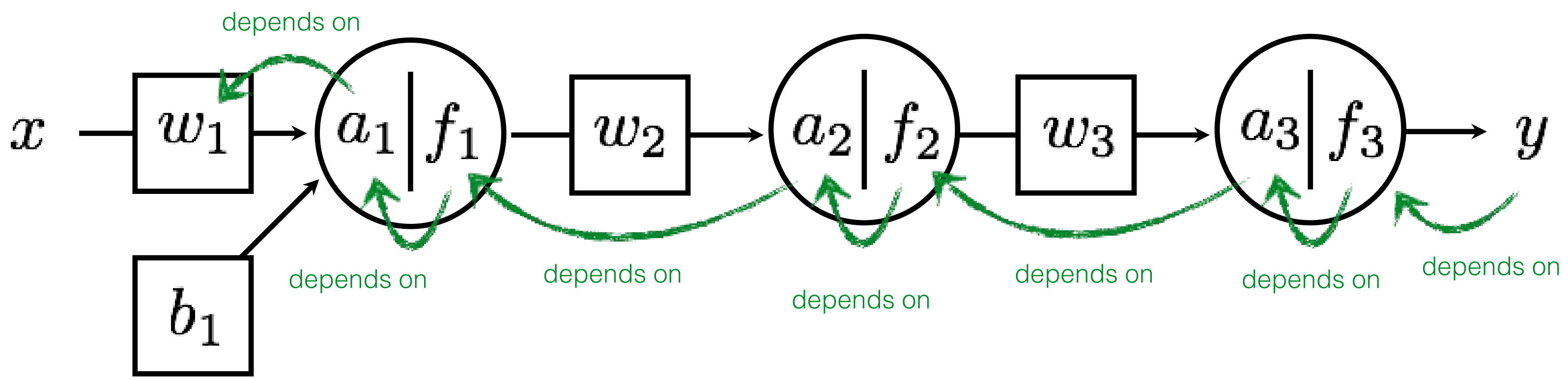
already computed.
re-use (propagate)!

The chain rule says…



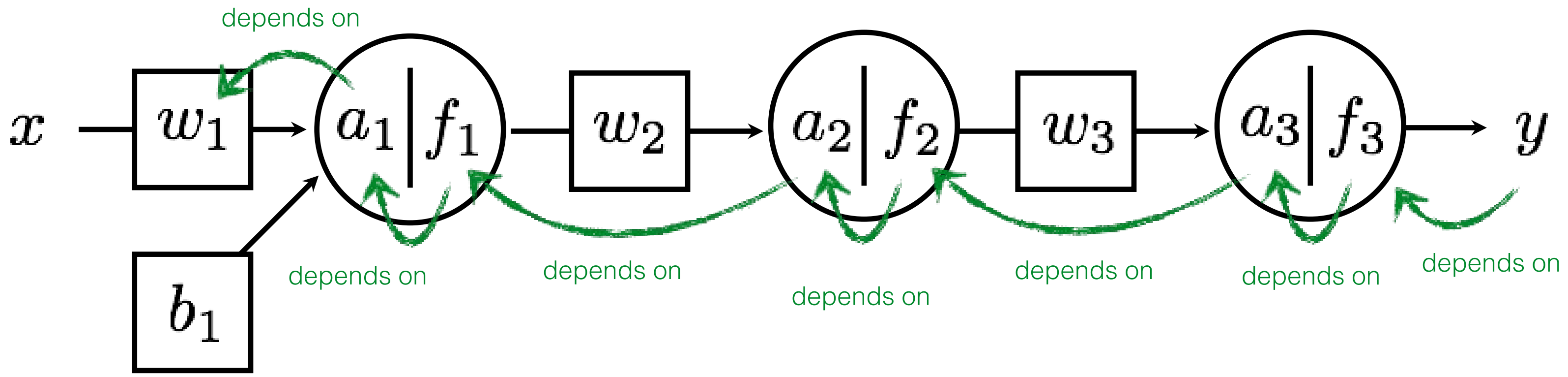
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

The chain rule says…



$$\frac{\partial L}{\partial w_1} = \boxed{\frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

already computed.
re-use (propagate)!

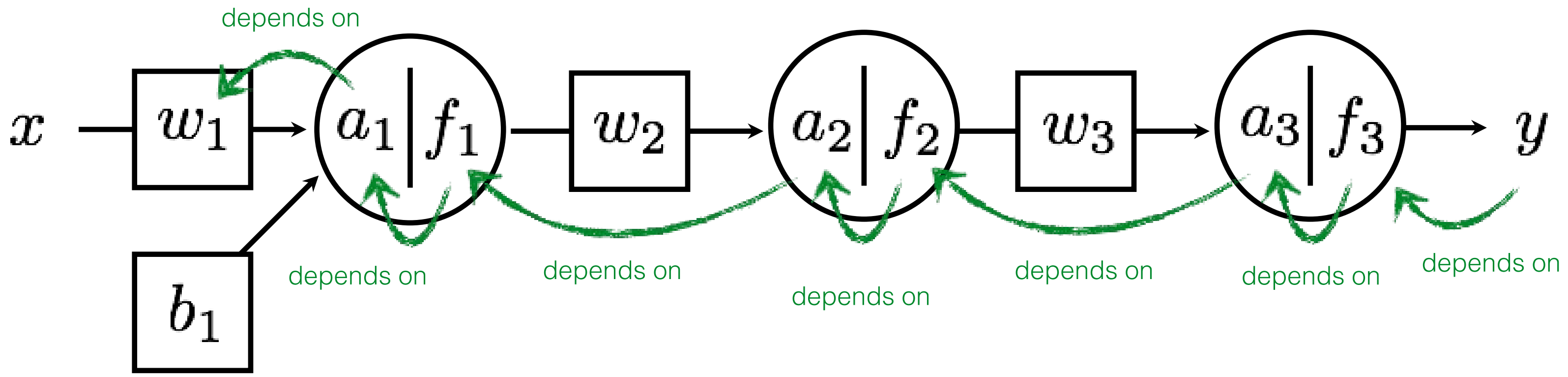


$$\frac{\partial \mathcal{L}}{\partial w_3} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}}\frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial b}$$

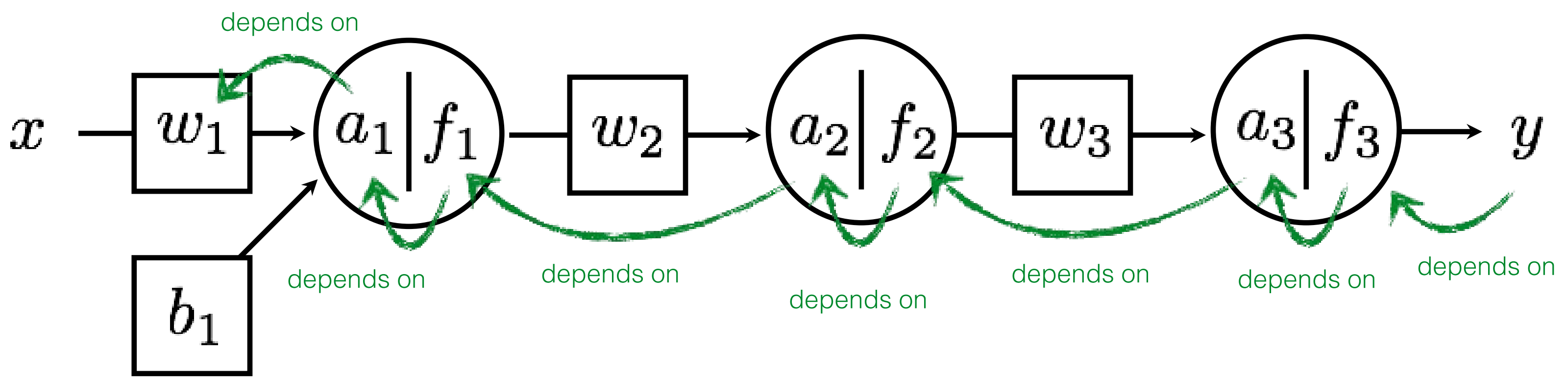


$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}}\frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial b}$$



$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}}\frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}}\frac{\partial a_1}{\partial b}$$

Gradient Descent

For each example sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
   b. Compute Loss $\mathcal{L}_i$
2. Update
   a. Back Propagation

$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$
$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$
$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$
$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial b}$$

   b. Gradient update

$$w_3 = w_3 - \eta \nabla w_3$$
$$w_2 = w_2 - \eta \nabla w_2$$
$$w_1 = w_1 - \eta \nabla w_1$$
$$b = b - \eta \nabla b$$

Gradient Descent

For each example sample $\{x_i, y_i\}$

1. Predict

   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$

   b. Compute Loss $\mathcal{L}_i$

2. Update

   a. Back Propagation $\dfrac{\partial \mathcal{L}}{\partial \theta}$

   vector of parameter partial derivatives

   b. Gradient update $\theta \leftarrow \theta - \eta \dfrac{\partial \mathcal{L}}{\partial \theta}$

   vector of parameter update equations

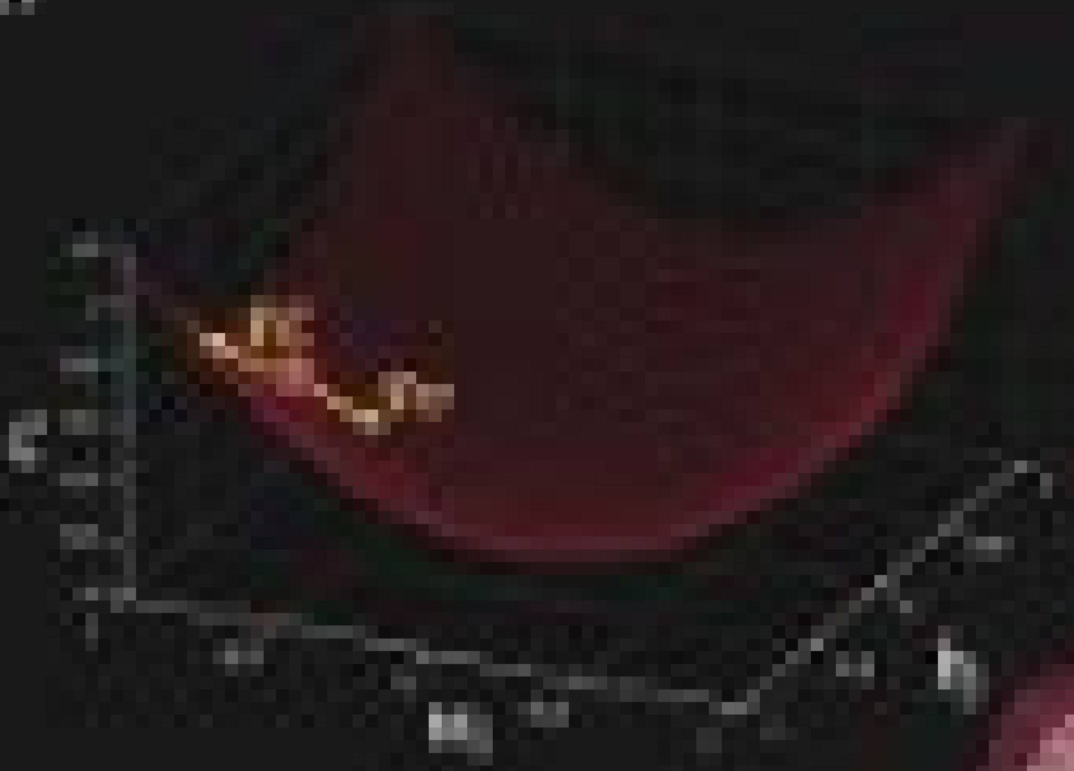
Gradient Descent

$$\nabla C = \left[\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right]^T \qquad \Delta v = \begin{bmatrix}\Delta v_1 \\ \Delta v_2\end{bmatrix}$$

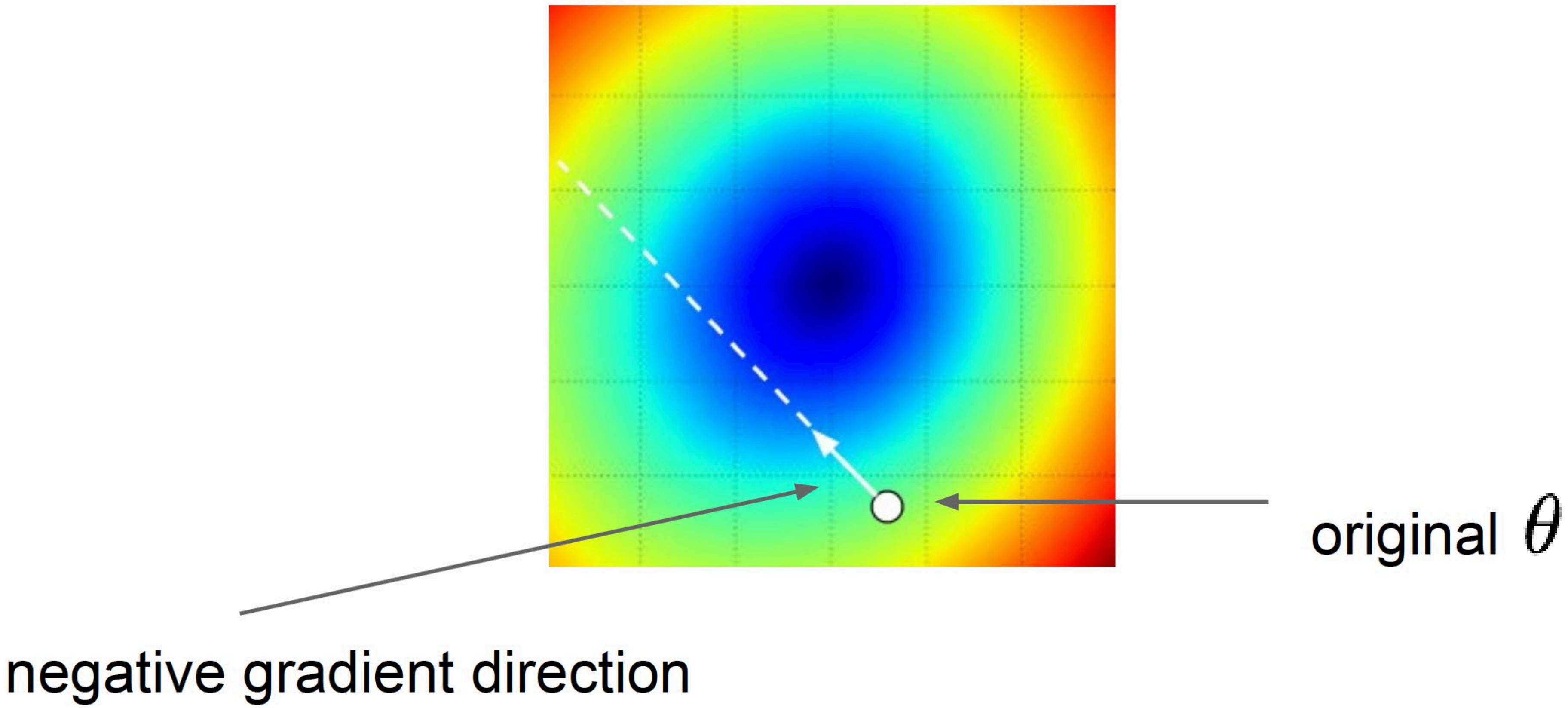
$$\Delta C \approx \nabla C \cdot \Delta v$$

Let $\Delta v = -\eta \nabla C$

$$\Delta C = -\eta \|\nabla C\|^2$$



Learning rates



$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$$

Step size: learning rate
Too big: will miss the minimum
Too small: slow convergence

Learning rate scheduling

- Use different learning rate at each iteration.
- Most common choice:

$$\eta_t = \frac{\eta_0}{\sqrt{t}}$$

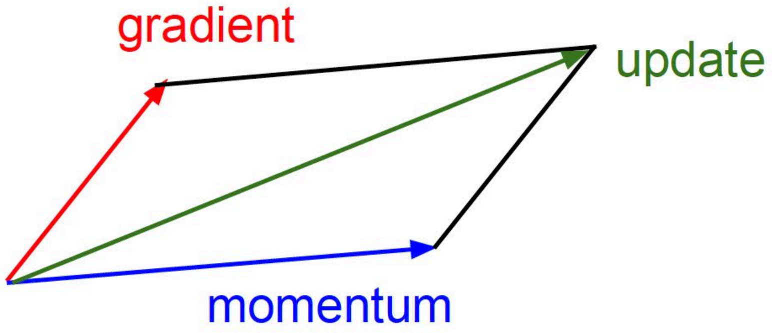
Need to select initial learning rate η_0

More modern choice: Adaptive learning rates.

$$\eta_t = G\left(\left\{\frac{\partial L}{\partial \theta}\right\}_{i=0}^{t}\right)$$

Many choices for G (Adam, Adagrad, Adadelta).

Momentum Update

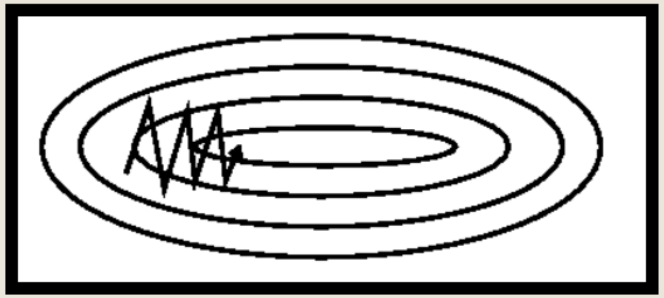


$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$$

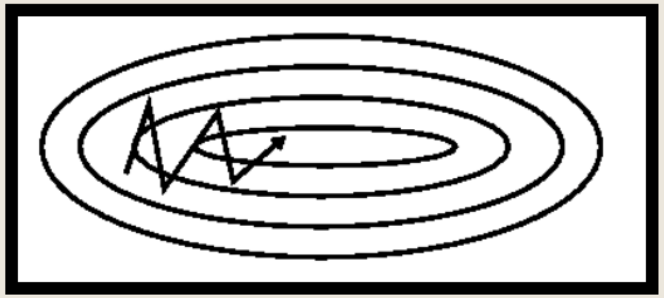
$$\Delta\theta \leftarrow w\frac{\partial L}{\partial \theta} + (1 - w)\Delta\theta$$

Take direction history into account!

```
weights_grad = evaluate_gradient(loss_fun, data, weights)
vel = vel * 0.9 - step_size * weights_grad
weights += vel
```



(Fig. 2a)



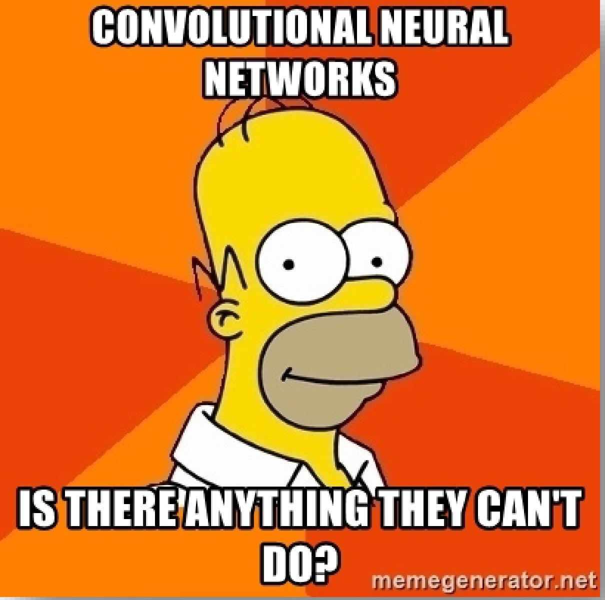
(Fig. 2b)

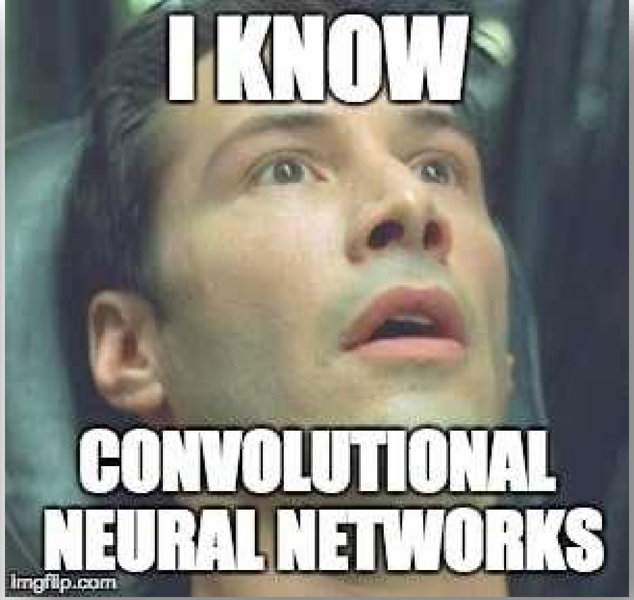
Many other ways to perform optimization…

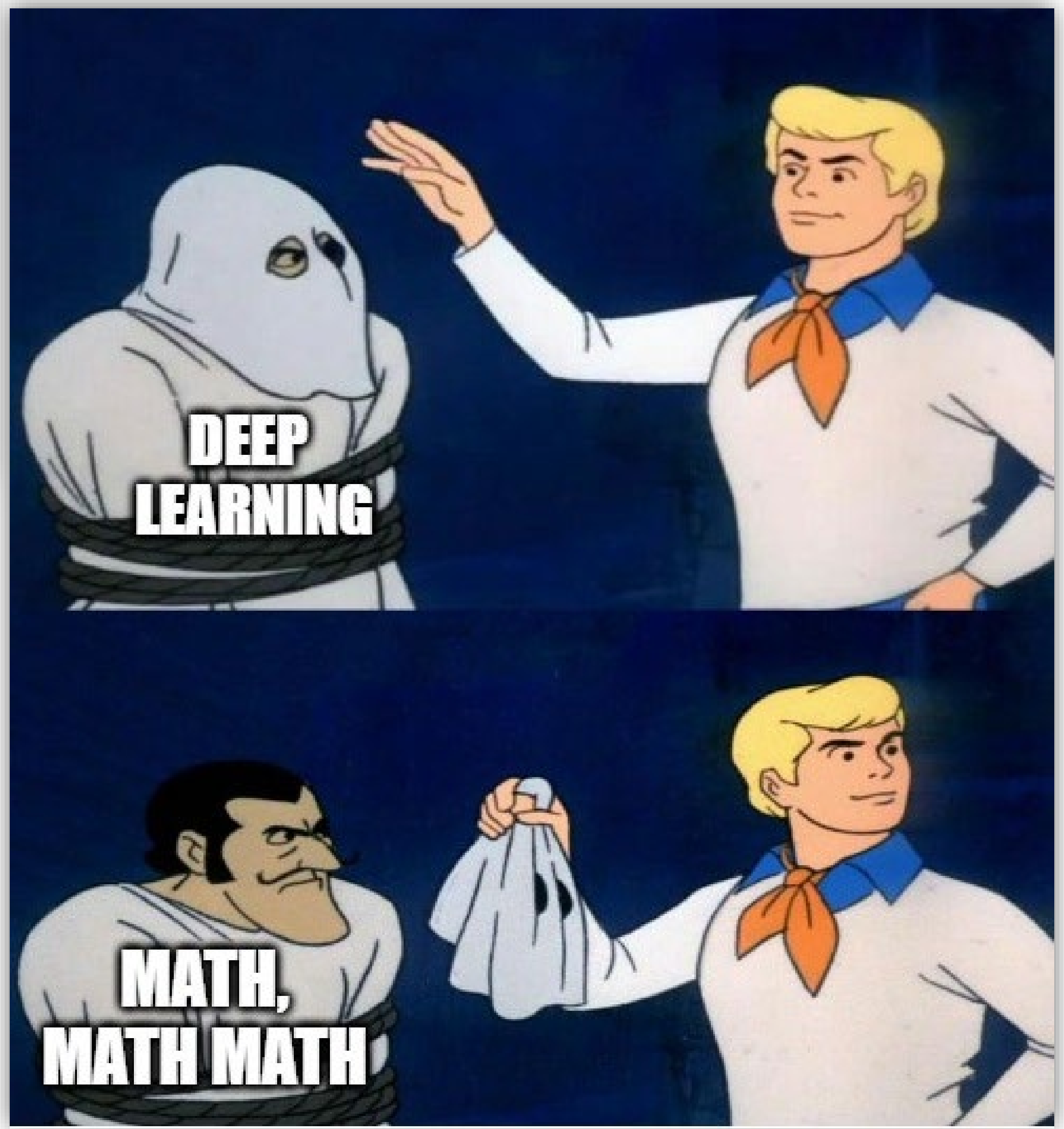
- Second order methods that use the Hessian (or its approximation): BFGS, **LBFGS**, etc.

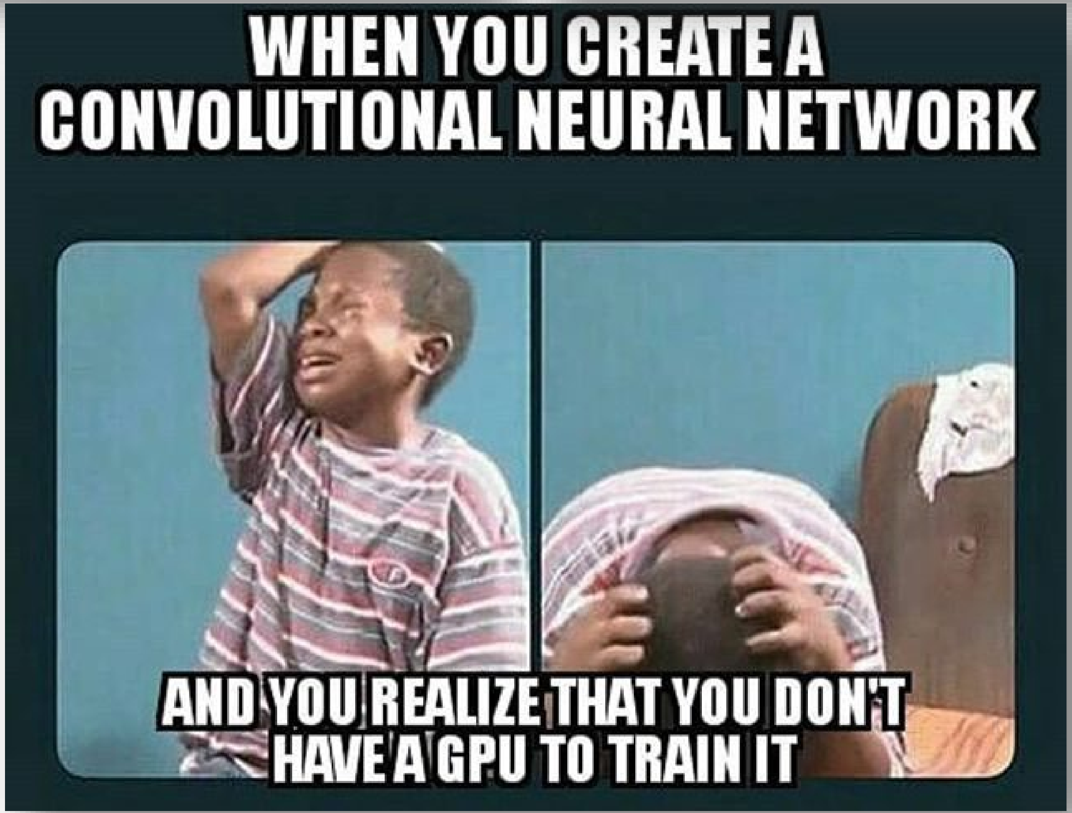
- Currently, the lesson from the trenches is that well-tuned SGD+Momentum is very hard to beat for CNNs.

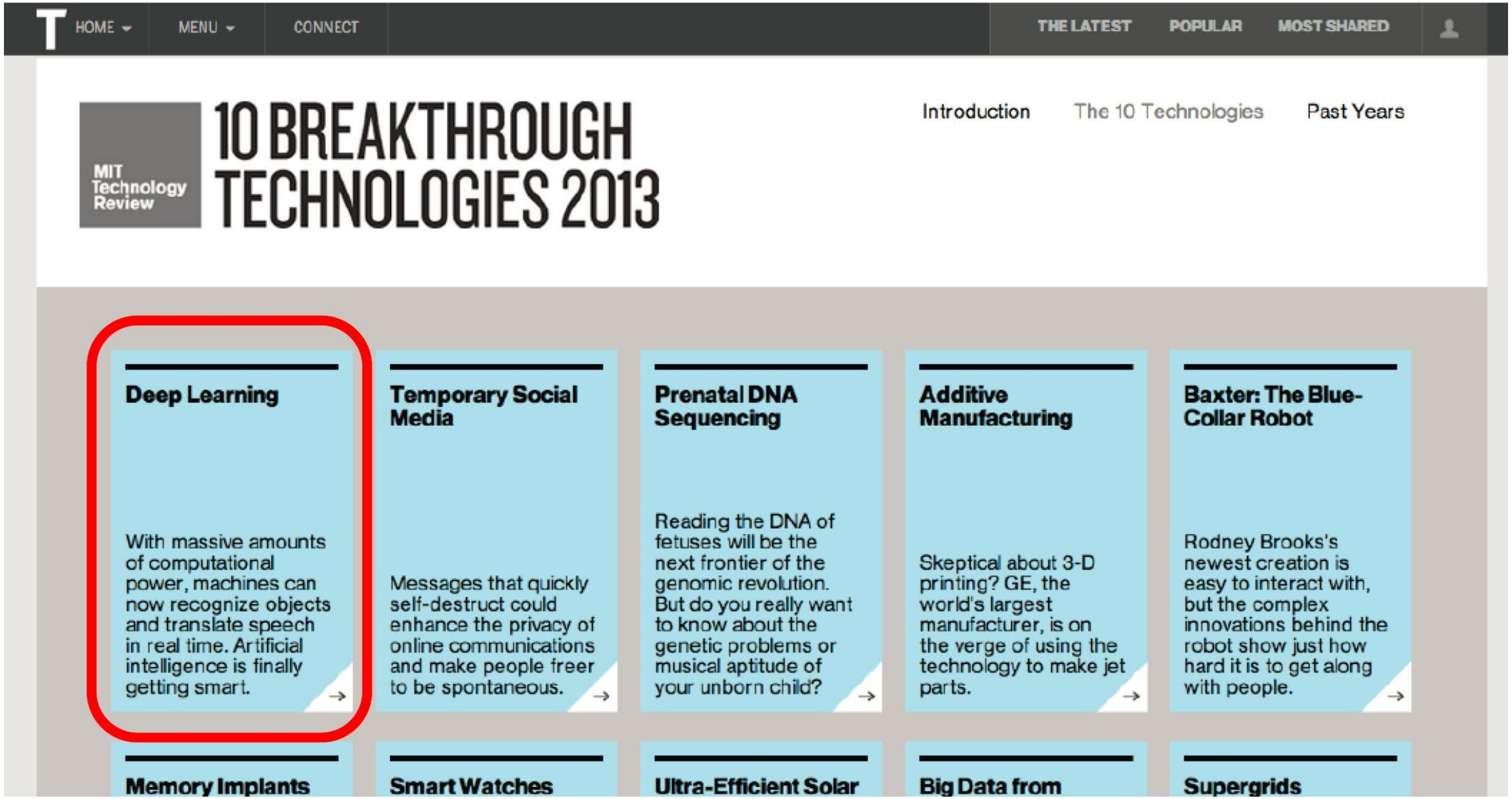
- No consensus on Adam etc.: Seem to give faster performance to worse local minima.


CONVOLUTIONAL NEURAL NETWORKS
IS THERE ANYTHING THEY CAN'T DO?
memegenerator.net


I KNOW
CONVOLUTIONAL NEURAL NETWORKS
imgflip.com


DEEP LEARNING
MATH, MATH MATH


WHEN YOU CREATE A CONVOLUTIONAL NEURAL NETWORK
AND YOU REALIZE THAT YOU DON'T HAVE A GPU TO TRAIN IT



HOME
MENU
CONNECT
THE LATEST
POPULAR
MOST SHARED
MIT Technology Review
10 BREAKTHROUGH TECHNOLOGIES 2013
Introduction
The 10 Technologies
Past Years
Deep Learning
With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart.
Temporary Social Media
Messages that quickly self-destruct could enhance the privacy of online communications and make people freer to be spontaneous.
Prenatal DNA Sequencing
Reading the DNA of fetuses will be the next frontier of the genomic revolution. But do you really want to know about the genetic problems or musical aptitude of your unborn child?
Additive Manufacturing
Skeptical about 3-D printing? GE, the world's largest manufacturer, is on the verge of using the technology to make jet parts.
Baxter: The Blue-Collar Robot
Rodney Brooks's newest creation is easy to interact with, but the complex innovations behind the robot show just how hard it is to get along with people.
Memory Implants
Smart Watches
Ultra-Efficient Solar
Big Data from
Supergrids

(Unrelated) Dog vs Food

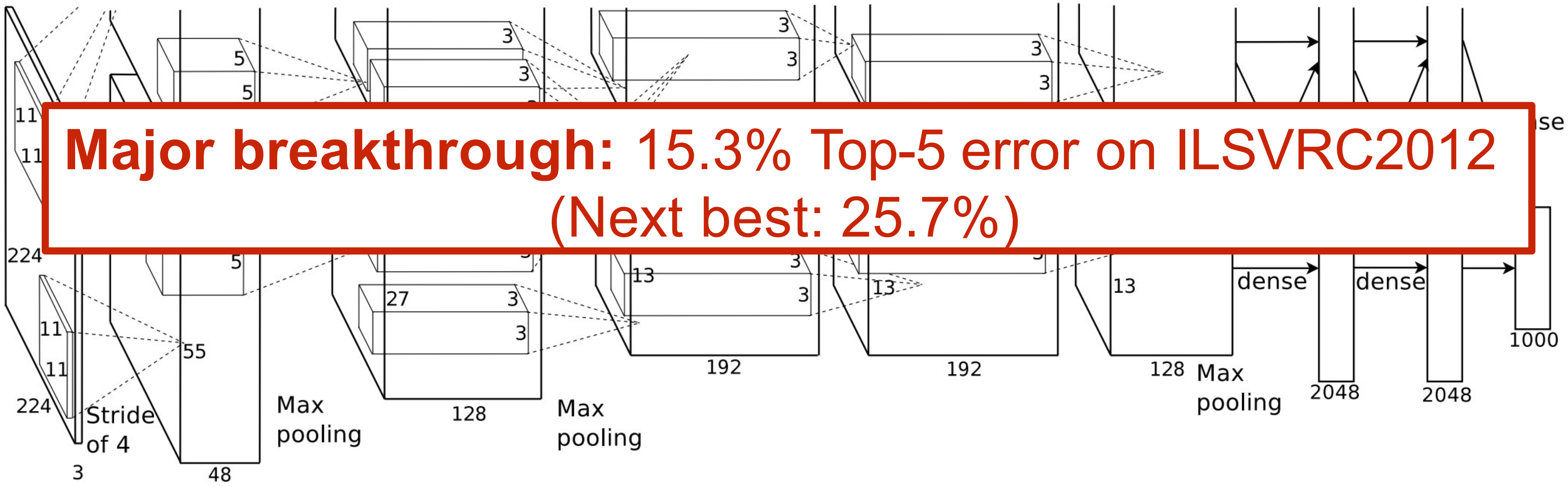
[Karen Zack, @teenybiscuit]

(Unrelated) Dog vs Food

[Karen Zack, @teenybiscuit]

CNNs in 2012: “SuperVision” (aka “AlexNet”)

“AlexNet” — Won the ILSVRC2012 Challenge

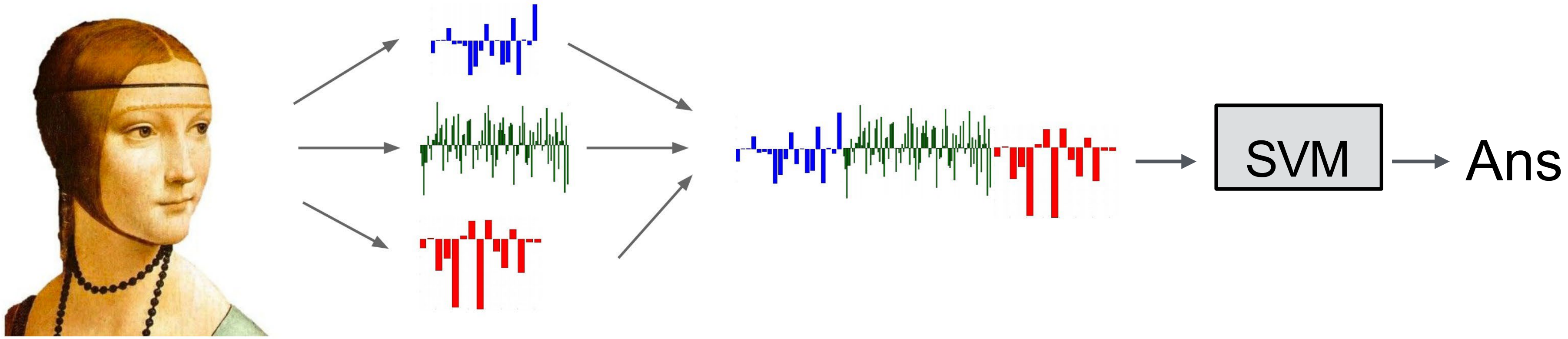


Major breakthrough: 15.3% Top-5 error on ILSVRC2012 (Next best: 25.7%)

Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network’s input is 150,528-dimensional, and the number of neurons in the network’s remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

[Krizhevsky, Sutskever, Hinton. NIPS 2012]

Recap: Before Deep Learning



Input Pixels

Extract Features

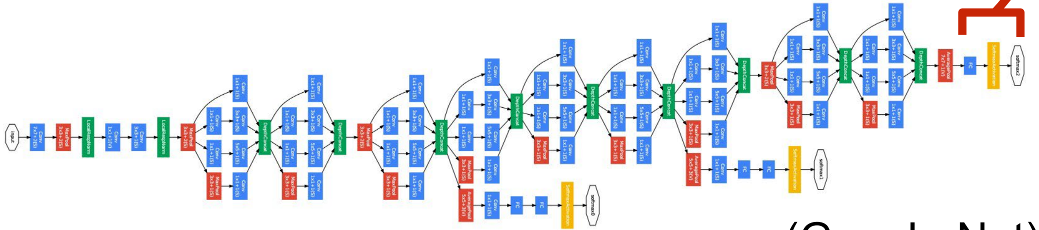
*Concatenate into a vector **x***

Linear Classifier

Figure: Karpathy 2016

The last layer of (most) CNNs are linear classifiers

This piece is just a linear classifier



(GoogLeNet)

Ans

Input Pixels

Perform everything with a big neural network, trained end-to-end

Key: perform enough processing so that by the time you get to the end of the network, the classes are linearly separable

ConvNets

They’re just neural networks with
3D activations and weight sharing

What shape should the activations have?

$$x \rightarrow \boxed{\text{Layer}} \rightarrow h^{(1)} \rightarrow \boxed{\text{Layer}} \rightarrow h^{(2)} \rightarrow \cdots \rightarrow f$$

- The input is an image, which is 3D (RGB channel, height, width)

What shape should the activations have?

$$x \rightarrow \boxed{\text{Layer}} \rightarrow h^{(1)} \rightarrow \boxed{\text{Layer}} \rightarrow h^{(2)} \rightarrow \cdots \rightarrow f$$

- The input is an image, which is 3D (RGB channel, height, width)

- We could flatten it to a 1D vector, but then we lose structure

What shape should the activations have?

$$x \rightarrow \boxed{\text{Layer}} \rightarrow h^{(1)} \rightarrow \boxed{\text{Layer}} \rightarrow h^{(2)} \rightarrow \cdots \rightarrow f$$

- The input is an image, which is 3D (RGB channel, height, width)

- We could flatten it to a 1D vector, but then we lose structure

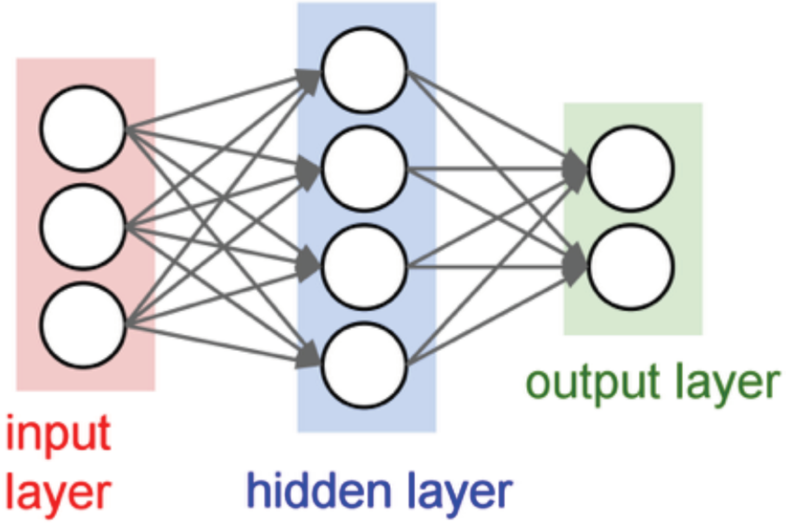
- What about keeping everything in 3D?

ConvNets

They’re just neural networks with
3D activations and weight sharing

3D Activations

before:

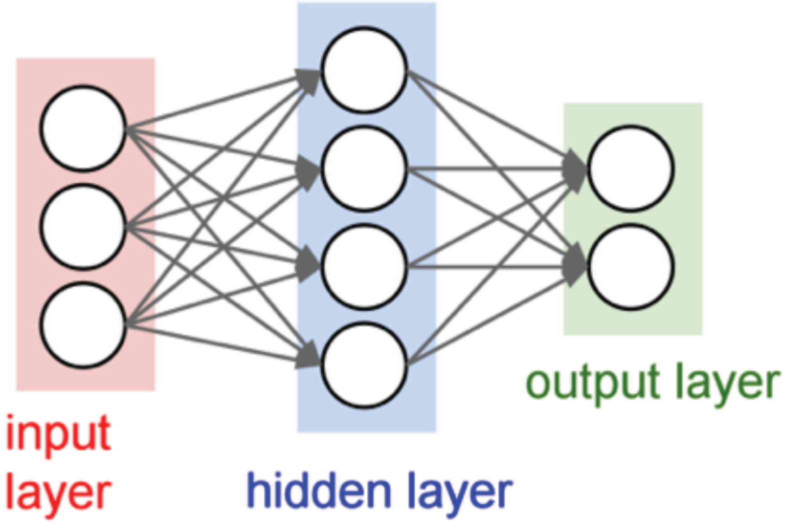


(1D vectors)

Figure: Andrej Karpathy

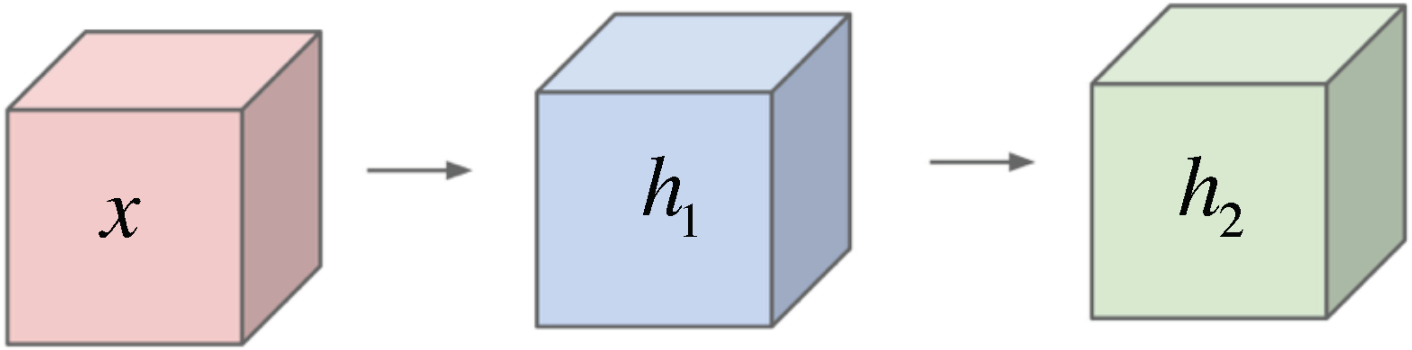
3D Activations

before:



(1D vectors)

now:



(3D arrays)

Figure: Andrej Karpathy

3D Activations

All Neural Net activations arranged in **3 dimensions:**

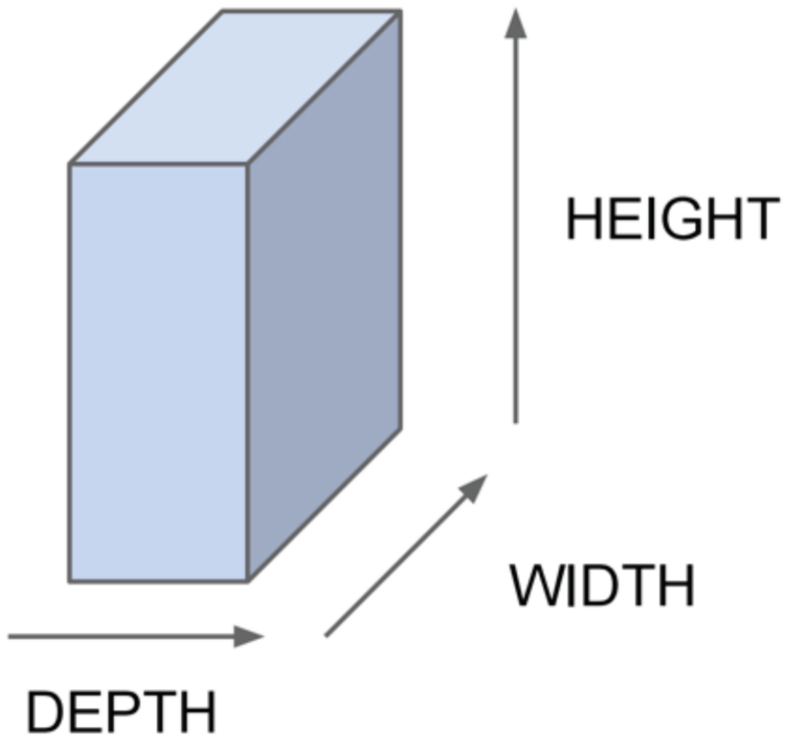
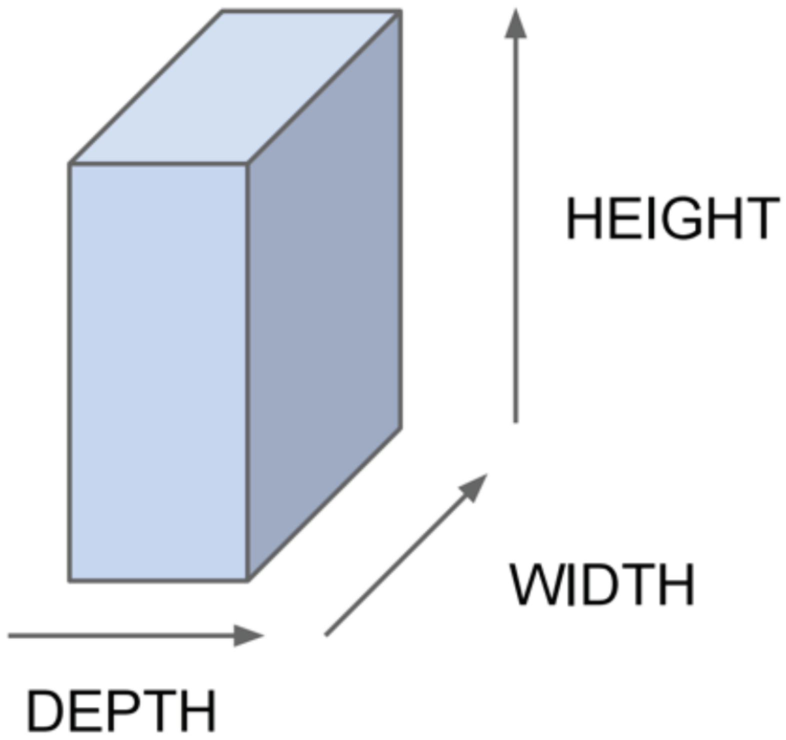


Figure: Andrej Karpathy

3D Activations

All Neural Net activations arranged in **3 dimensions:**



For example, a CIFAR-10 image is a 3x32x32 volume
(3 depth — RGB channels, 32 height, 32 width)

Figure: Andrej Karpathy

3D Activations

1D Activations:

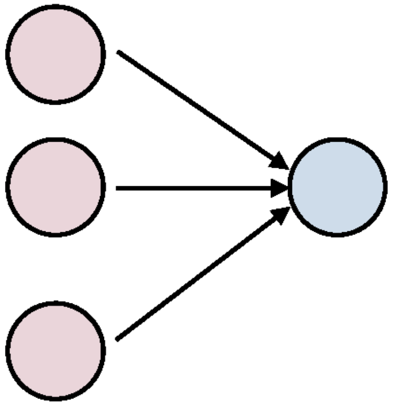
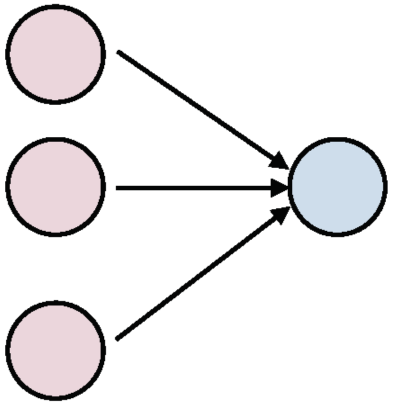


Figure: Andrej Karpathy

3D Activations

1D Activations:



3D Activations:

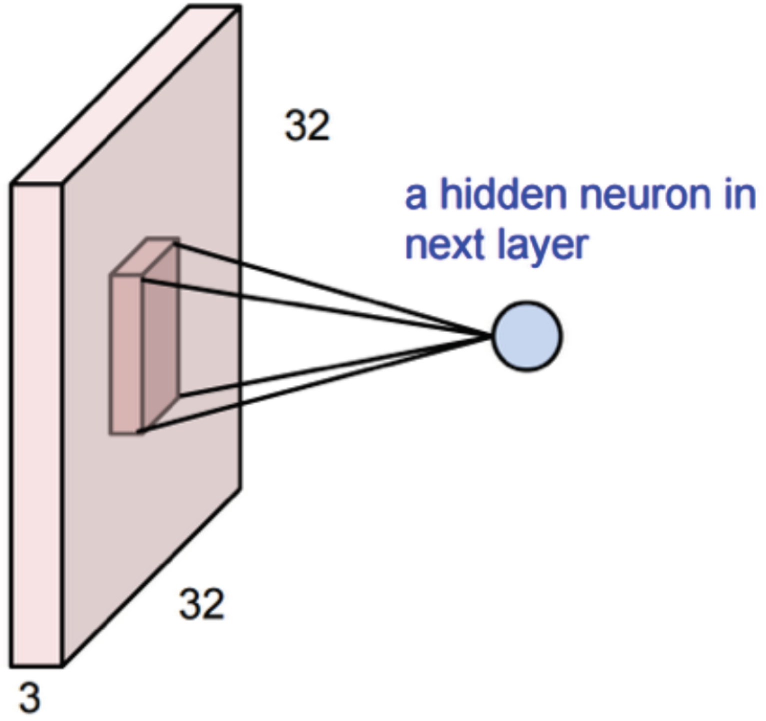


Figure: Andrej Karpathy

3D Activations

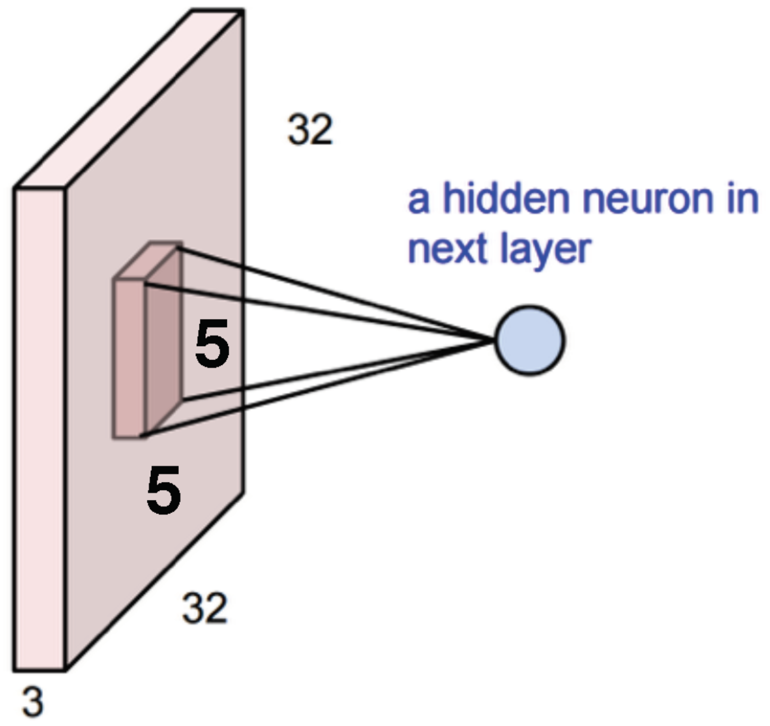
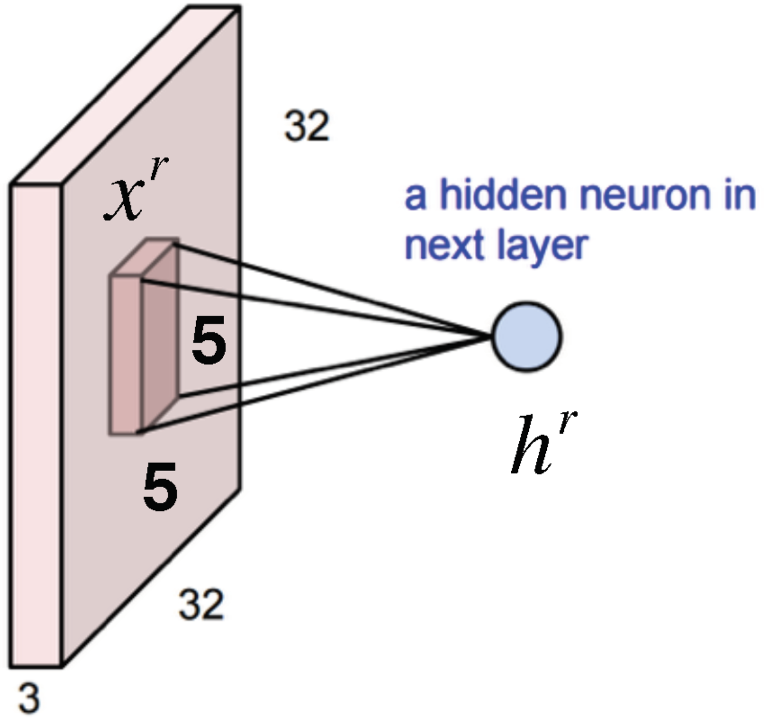


Figure: Andrej Karpathy

- The input is 3x32x32
- This neuron depends on a 3x5x5 chunk of the input
- The neuron also has a 3x5x5 set of weights and a bias (scalar)

3D Activations

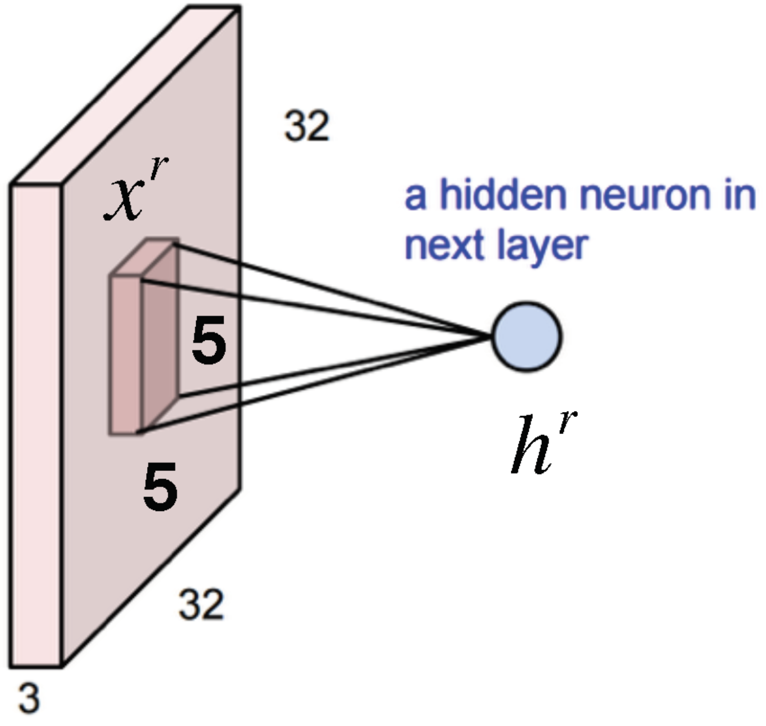


Example: consider the region of the input "x^r"

With output neuron h^r

Figure: Andrej Karpathy

3D Activations



Example: consider the region of the input "x^r"

With output neuron h^r

Then the output is:

$$h^r = \sum_{ijk} x^r{}_{ijk} W_{ijk} + b$$

Figure: Andrej Karpathy

3D Activations

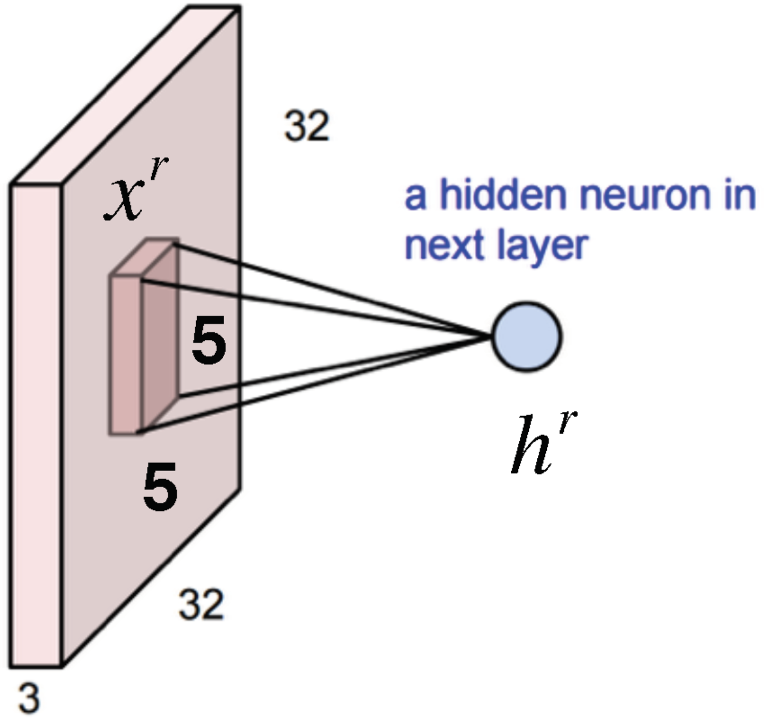


Figure: Andrej Karpathy

Example: consider the region of the input "x^r"

With output neuron h^r

Then the output is:

$$h^r = \sum_{ijk} x^r{}_{ijk} W_{ijk} + b$$

Sum over 3 axes

3D Activations

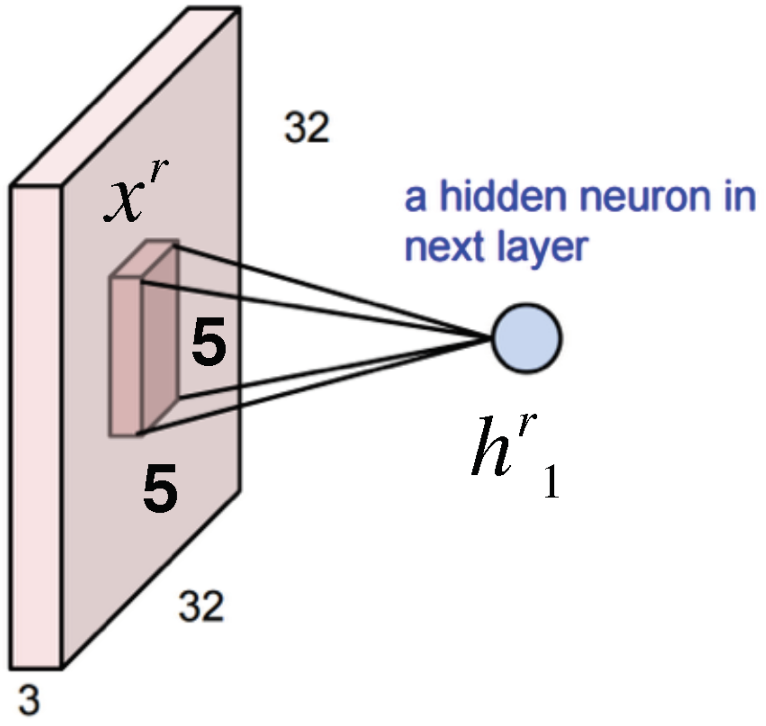


Figure: Andrej Karpathy

3D Activations

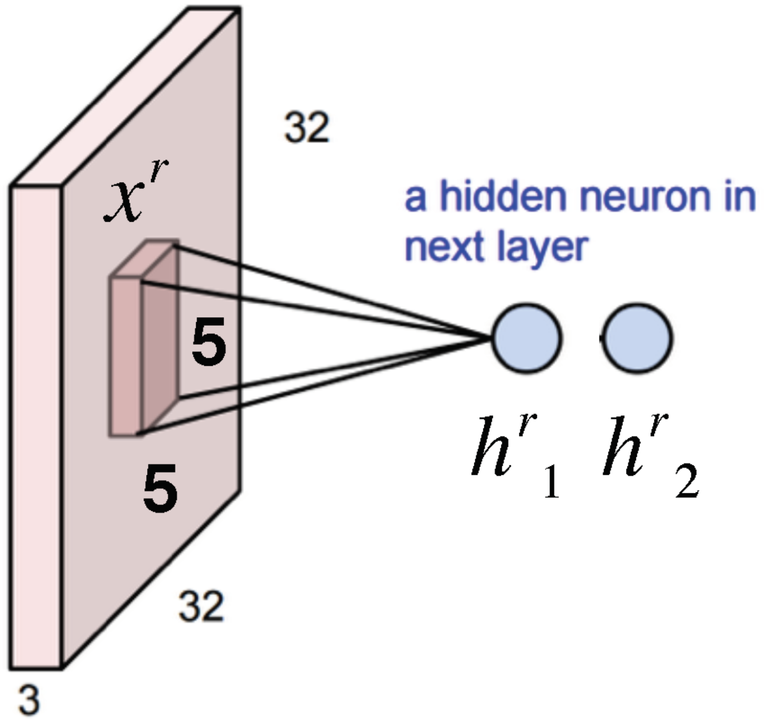
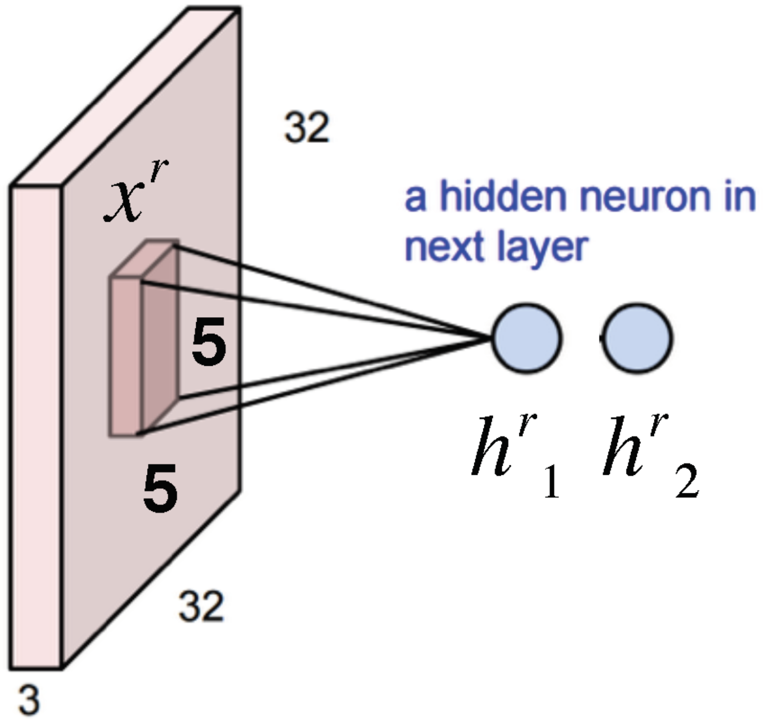


Figure: Andrej Karpathy

3D Activations



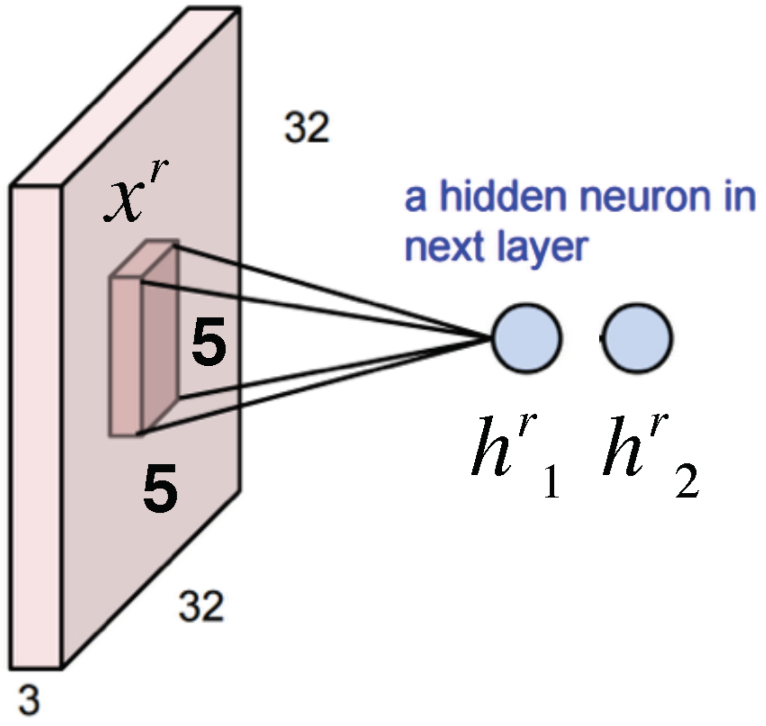
With **2** output neurons

$$h^r{}_1 = \sum_{ijk} x^r{}_{ijk} W_{1ijk} + b_1$$

$$h^r{}_2 = \sum_{ijk} x^r{}_{ijk} W_{2ijk} + b_2$$

Figure: Andrej Karpathy

3D Activations



With **2** output neurons

$$h^r{}_1 = \sum_{ijk} x^r{}_{ijk} W_{\boxed{1}ijk} + b_{\boxed{1}}$$

$$h^r{}_2 = \sum_{ijk} x^r{}_{ijk} W_{\boxed{2}ijk} + b_{\boxed{2}}$$

Figure: Andrej Karpathy

3D Activations

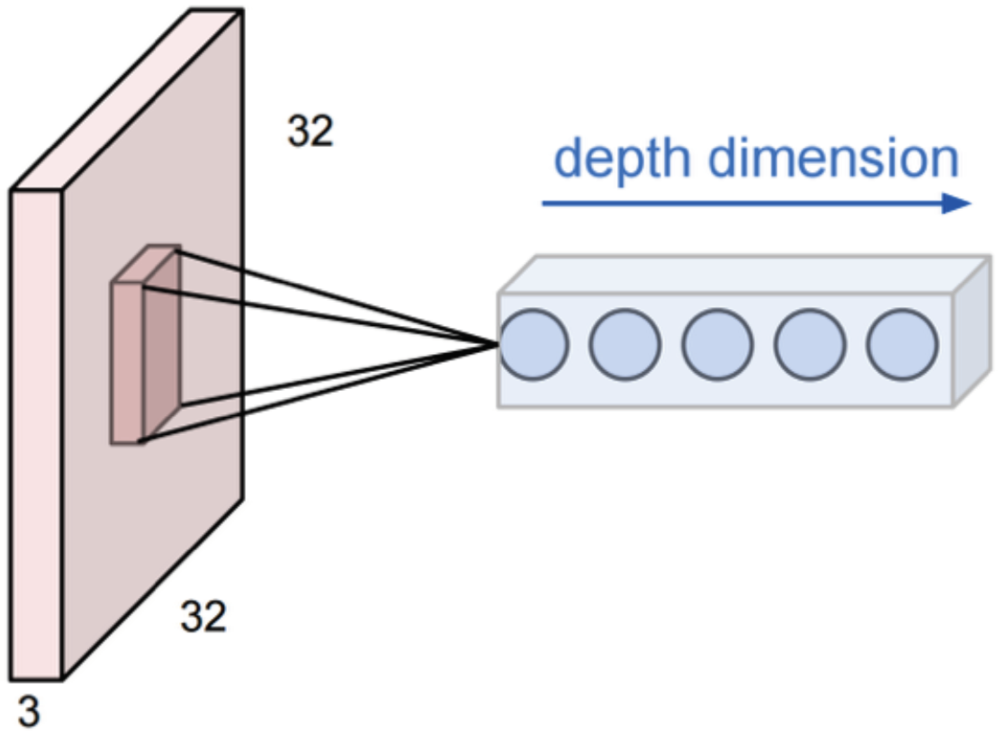
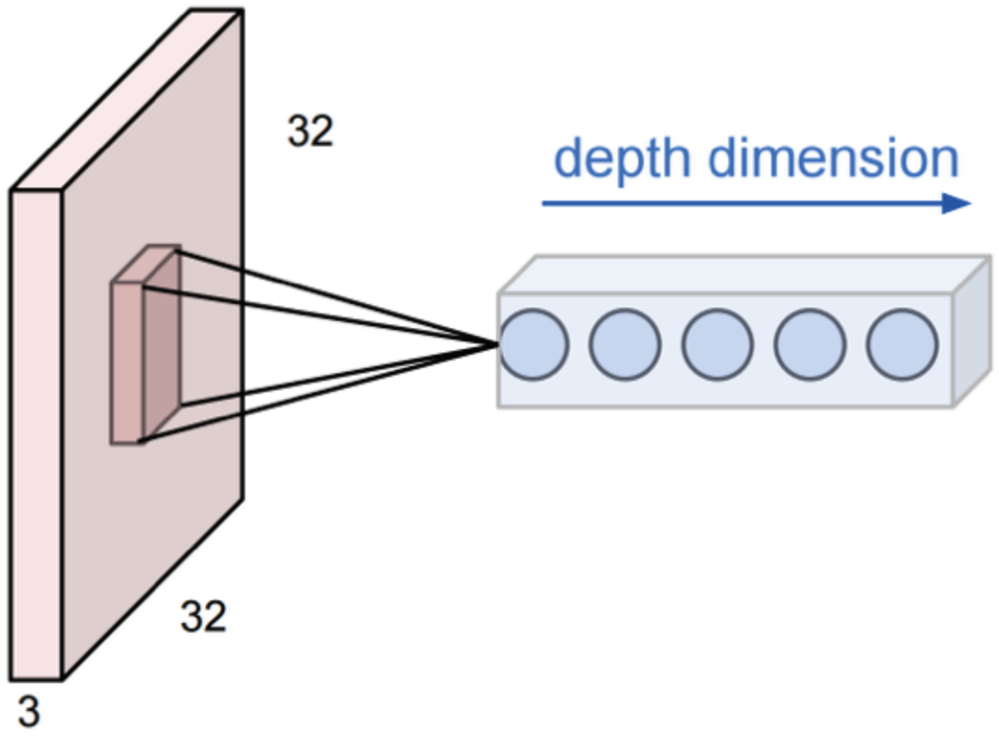


Figure: Andrej Karpathy

3D Activations

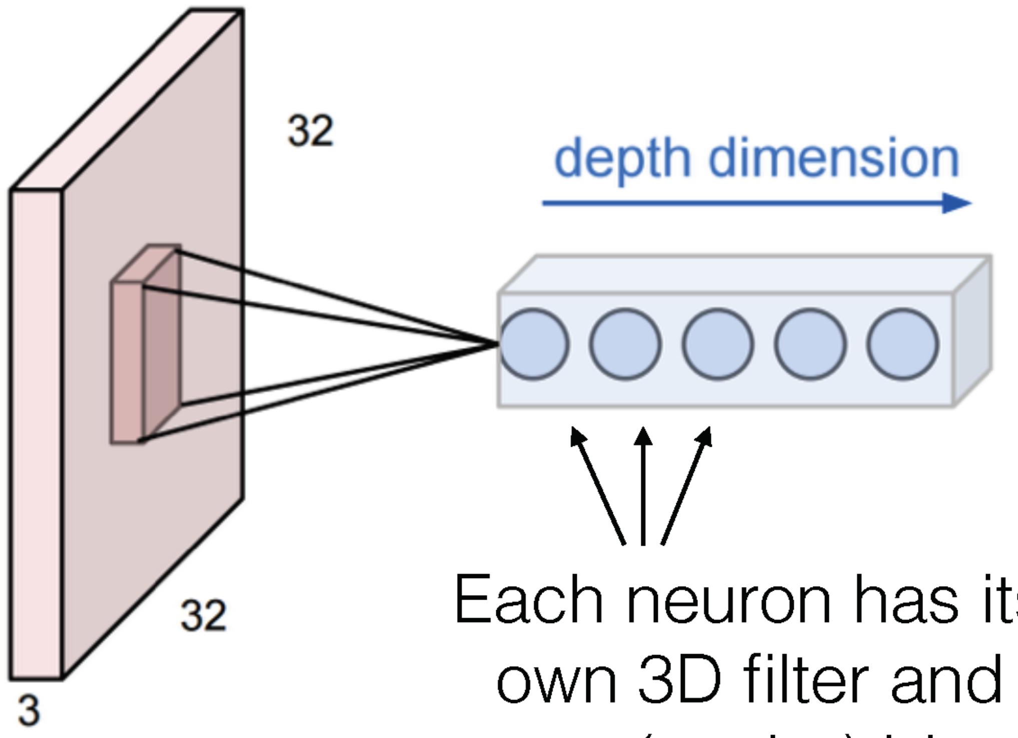


We can keep adding more outputs

These form a column in the output volume: [depth x 1 x 1]

Figure: Andrej Karpathy

3D Activations



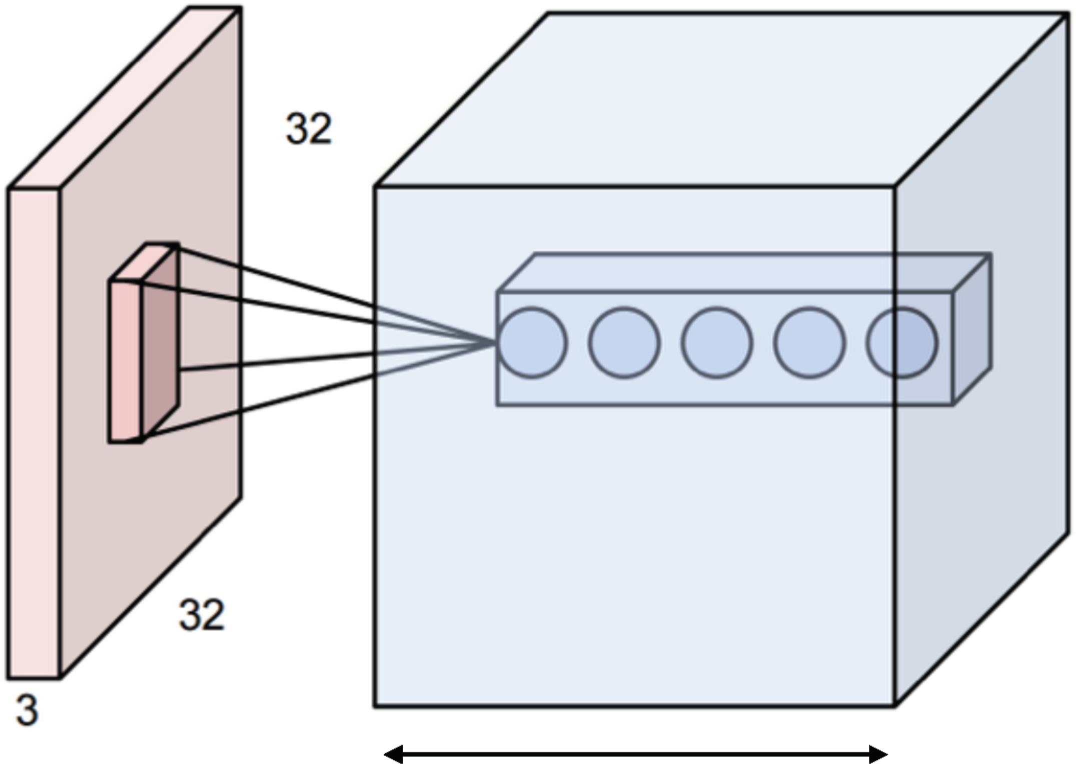
Each neuron has its own 3D filter and own (scalar) bias

We can keep adding more outputs

These form a column in the output volume: [depth x 1 x 1]

Figure: Andrej Karpathy

3D Activations

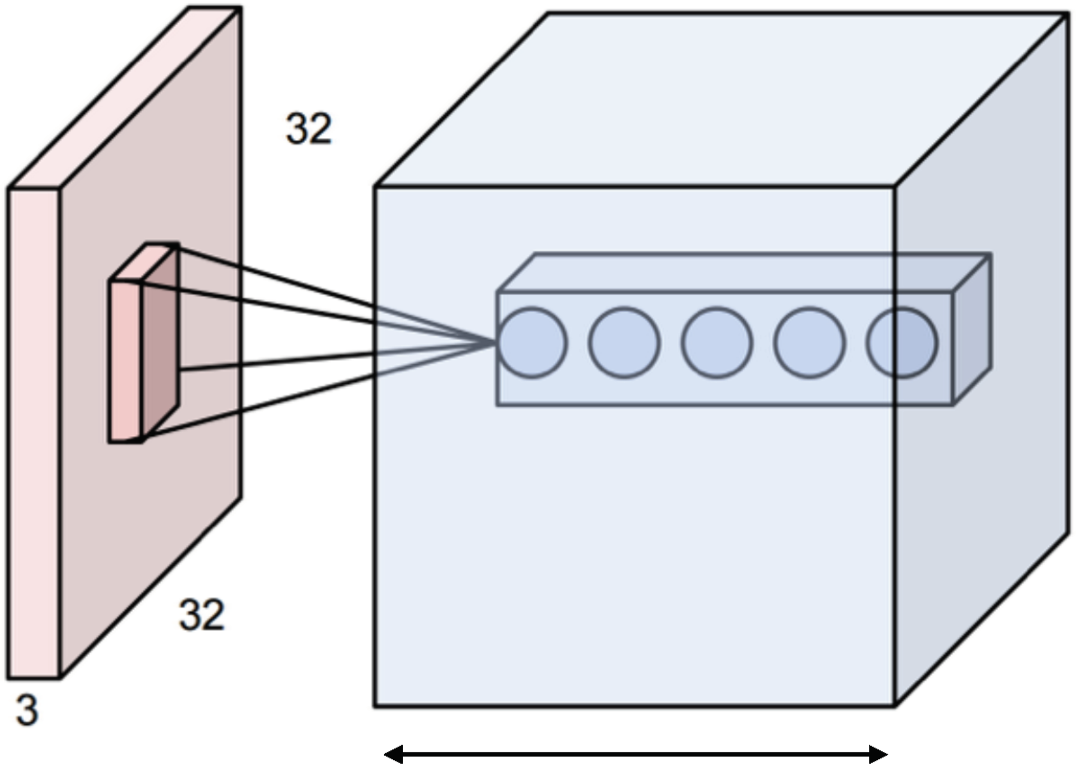


D sets of weights
(also called filters)

Now repeat this
across the input

Figure: Andrej Karpathy

3D Activations



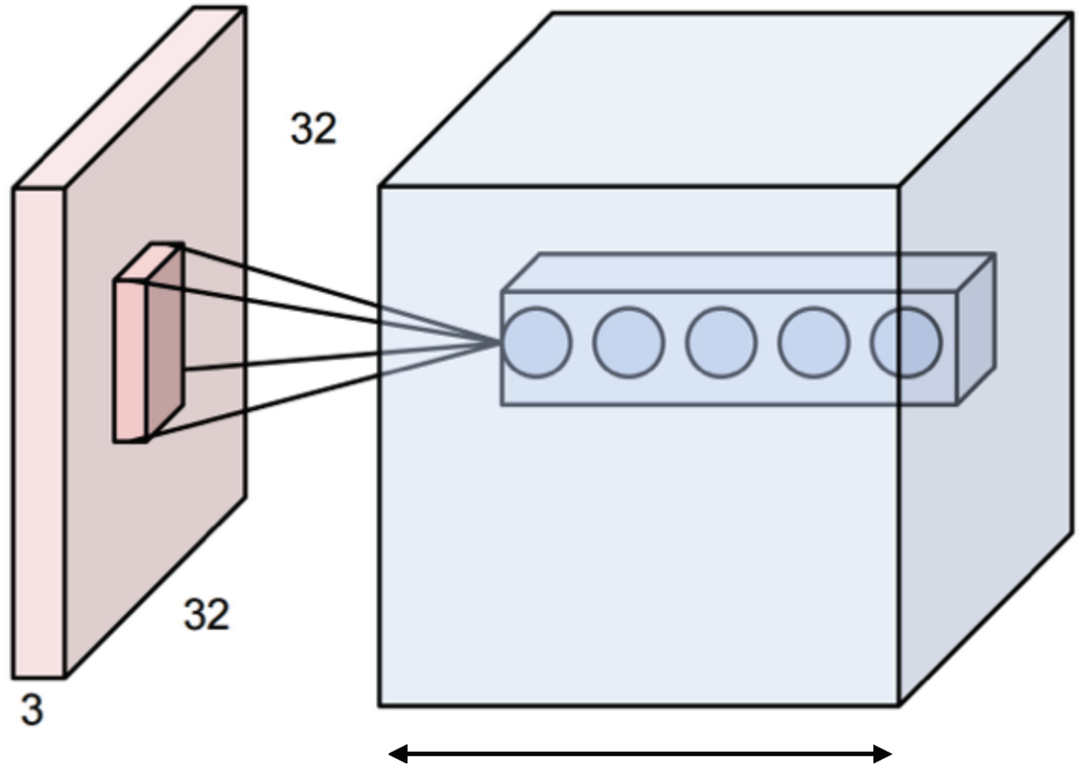
D sets of weights
(also called filters)

Now repeat this across the input

Weight sharing:
Each filter shares the same weights (but each depth index has its own set of weights)

Figure: Andrej Karpathy

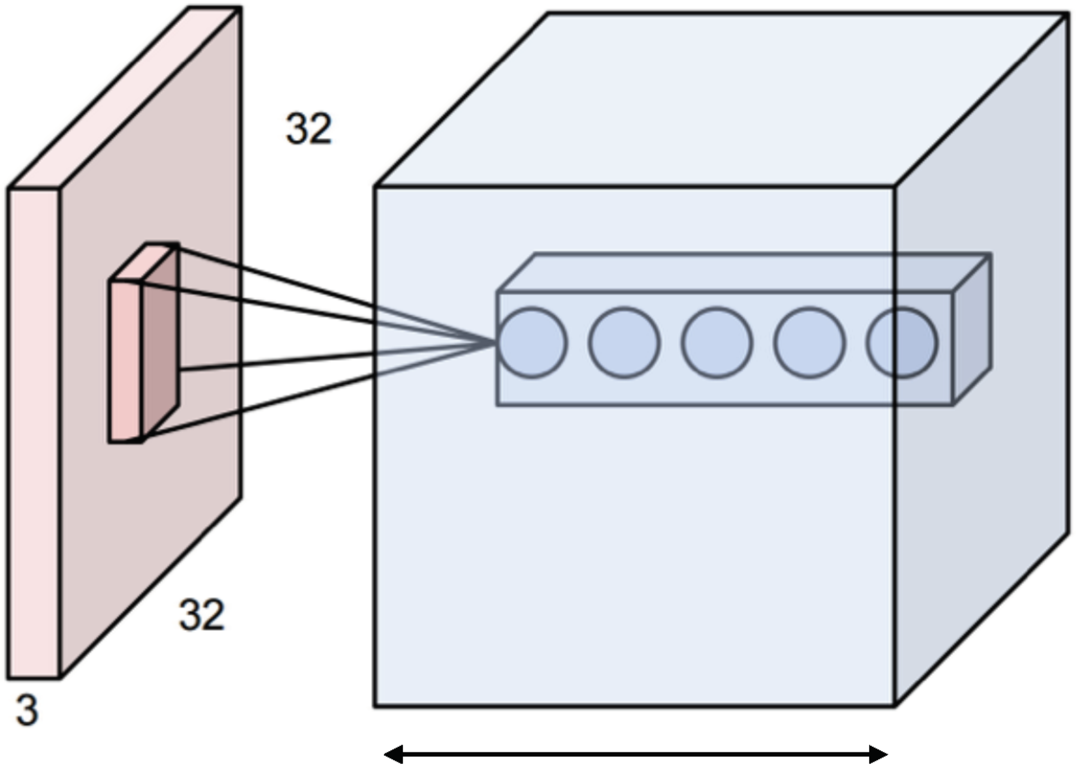
3D Activations



D sets of weights
(also called filters)

Figure: Andrej Karpathy

3D Activations

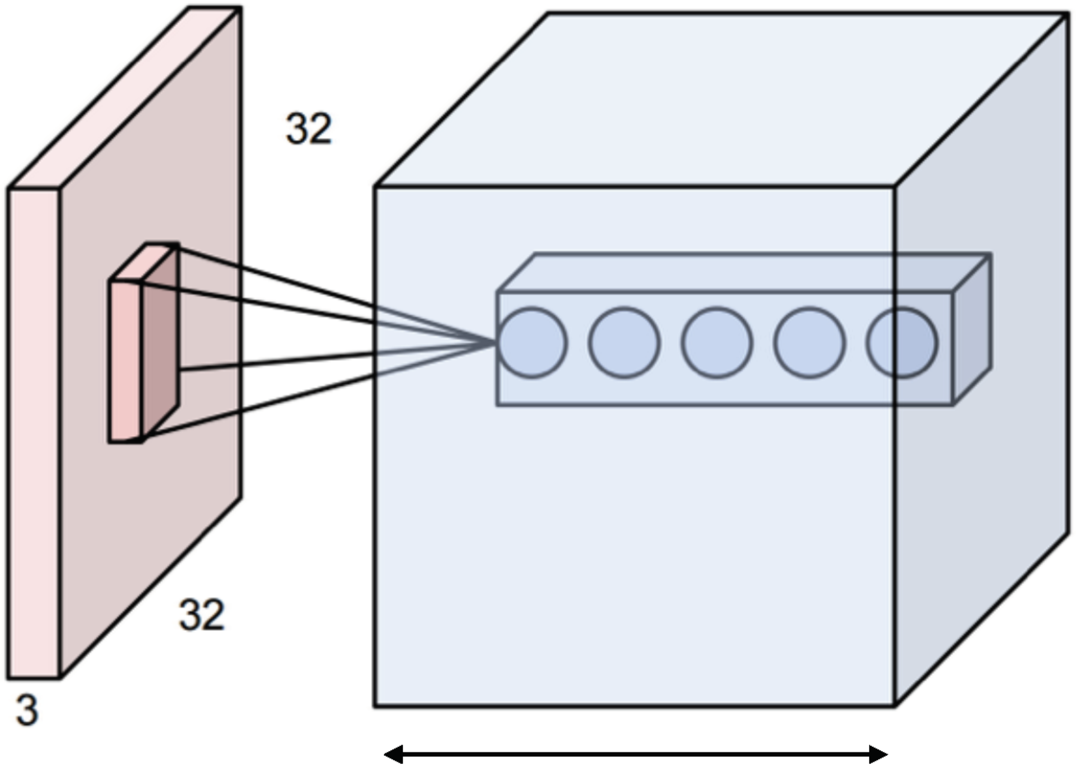


D sets of weights
(also called filters)

With weight sharing, this is called **convolution**

Figure: Andrej Karpathy

3D Activations



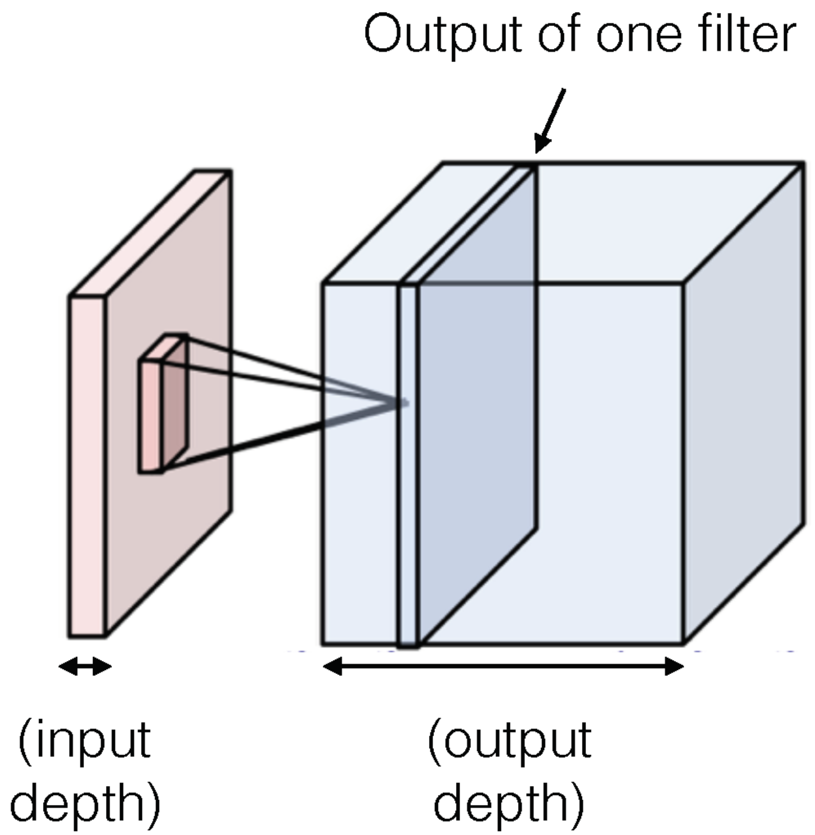
D sets of weights
(also called filters)

With weight sharing, this is called **convolution**

Without weight sharing, this is called a **locally connected layer**

Figure: Andrej Karpathy

3D Activations

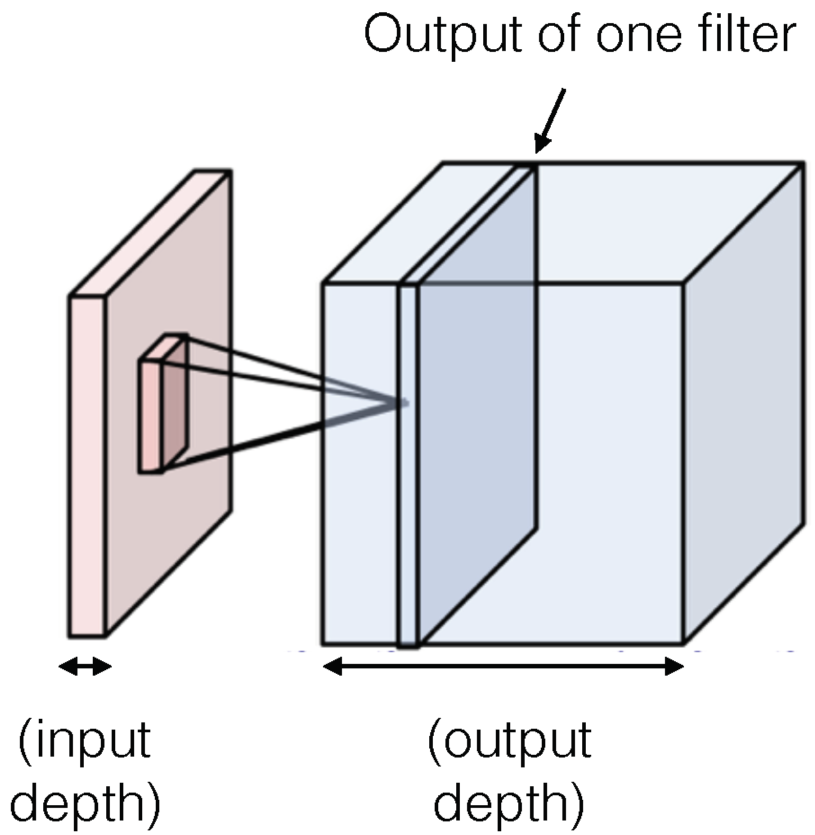


One set of weights gives one slice in the output

To get a 3D output of depth *D*, use *D* different filters

In practice, ConvNets use many filters (~64 to 1024)

3D Activations



One set of weights gives one slice in the output

To get a 3D output of depth *D*, use *D* different filters

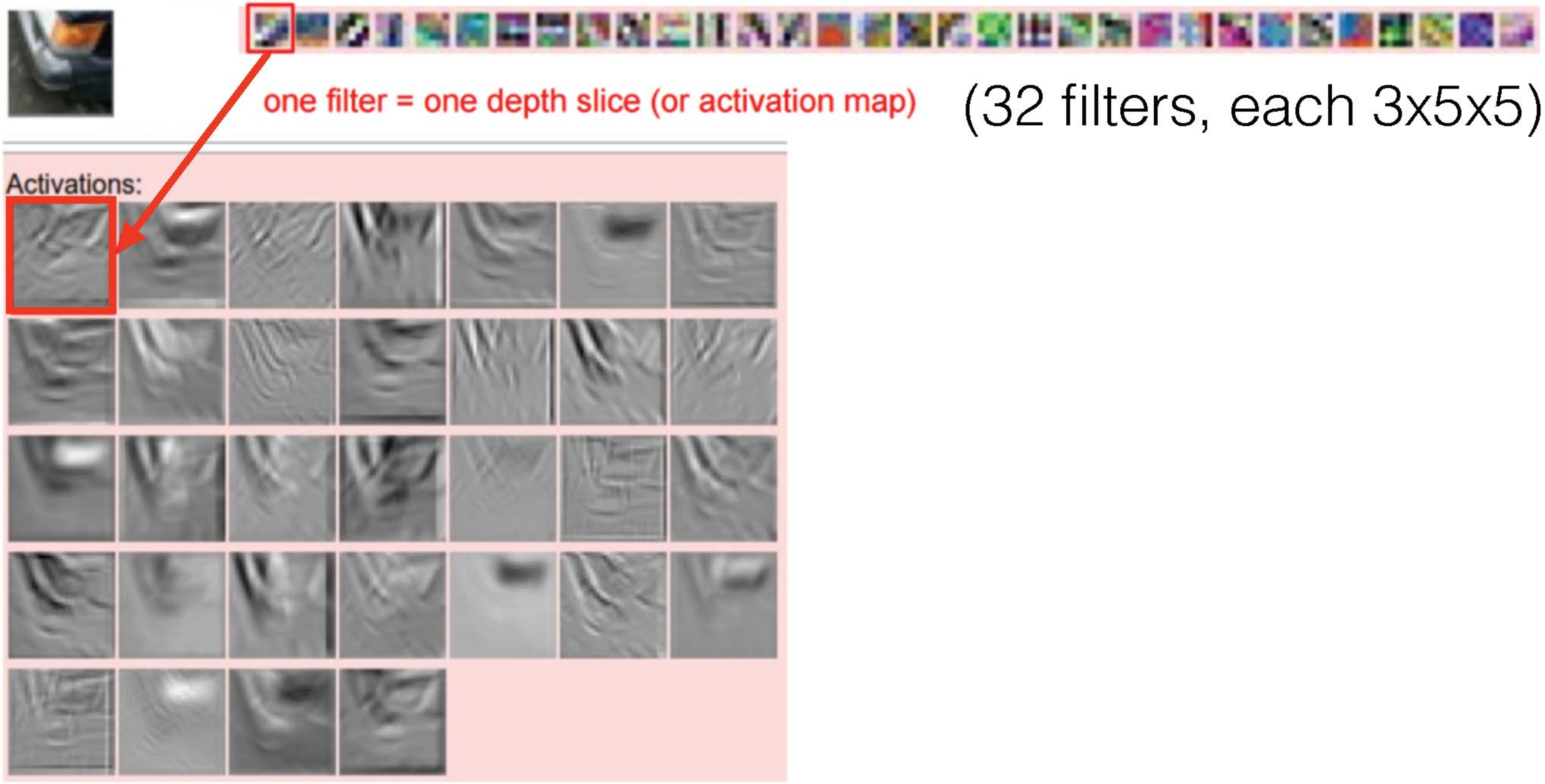
In practice, ConvNets use many filters (~64 to 1024)

All together, the weights are **4** dimensional:
(output depth, input depth, kernel height, kernel width)

3D Activations

We can unravel the 3D cube and show each layer separately:

(Input)



(32 filters, each 3x5x5)

Figure: Andrej Karpathy

3D Activations

We can unravel the 3D cube and show each layer separately:

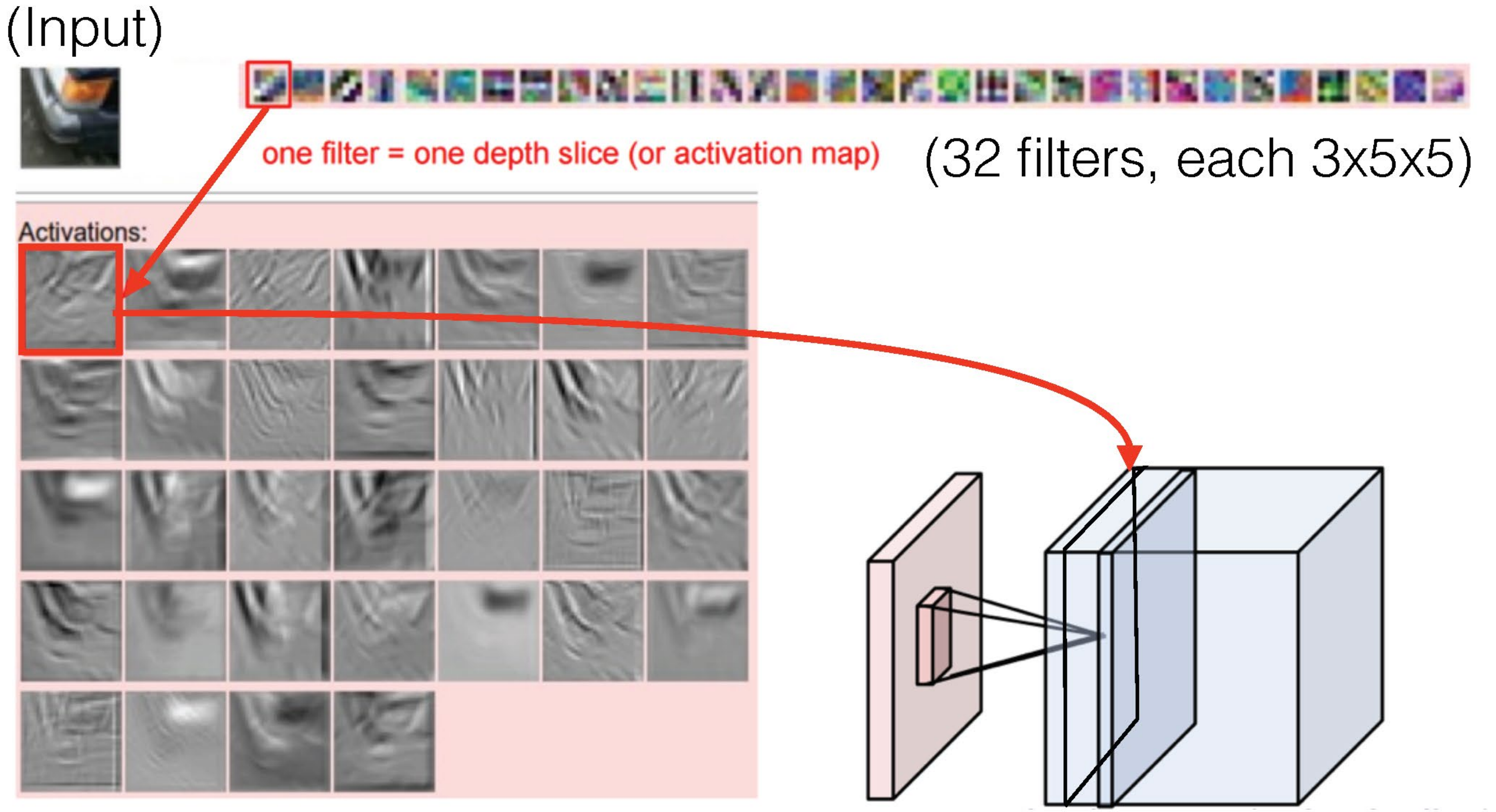


Figure: Andrej Karpathy

3D Activations

We can unravel the 3D cube and show each layer separately:

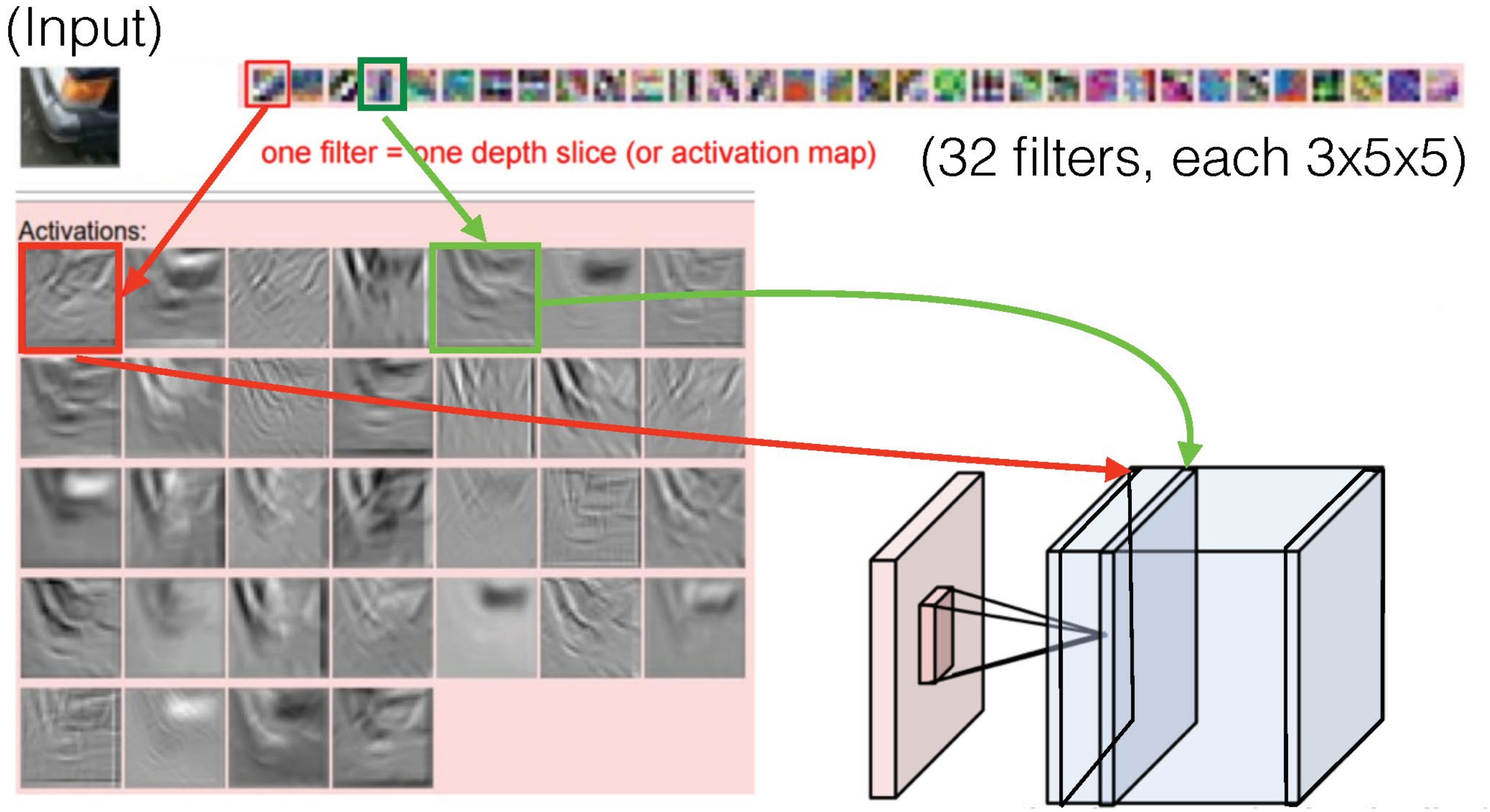


Figure: Andrej Karpathy

3D Activations

We can unravel the 3D cube and show each layer separately:

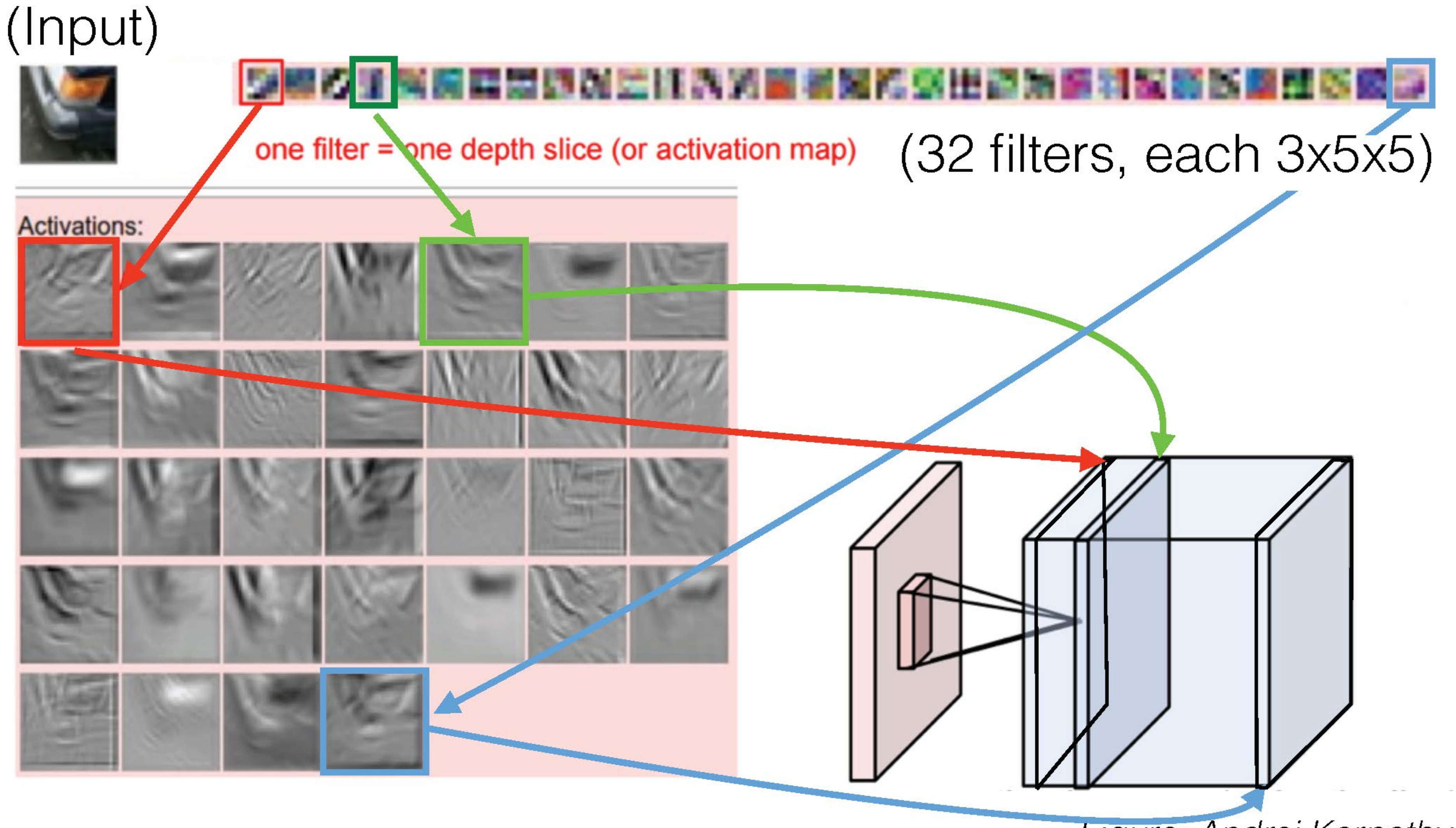
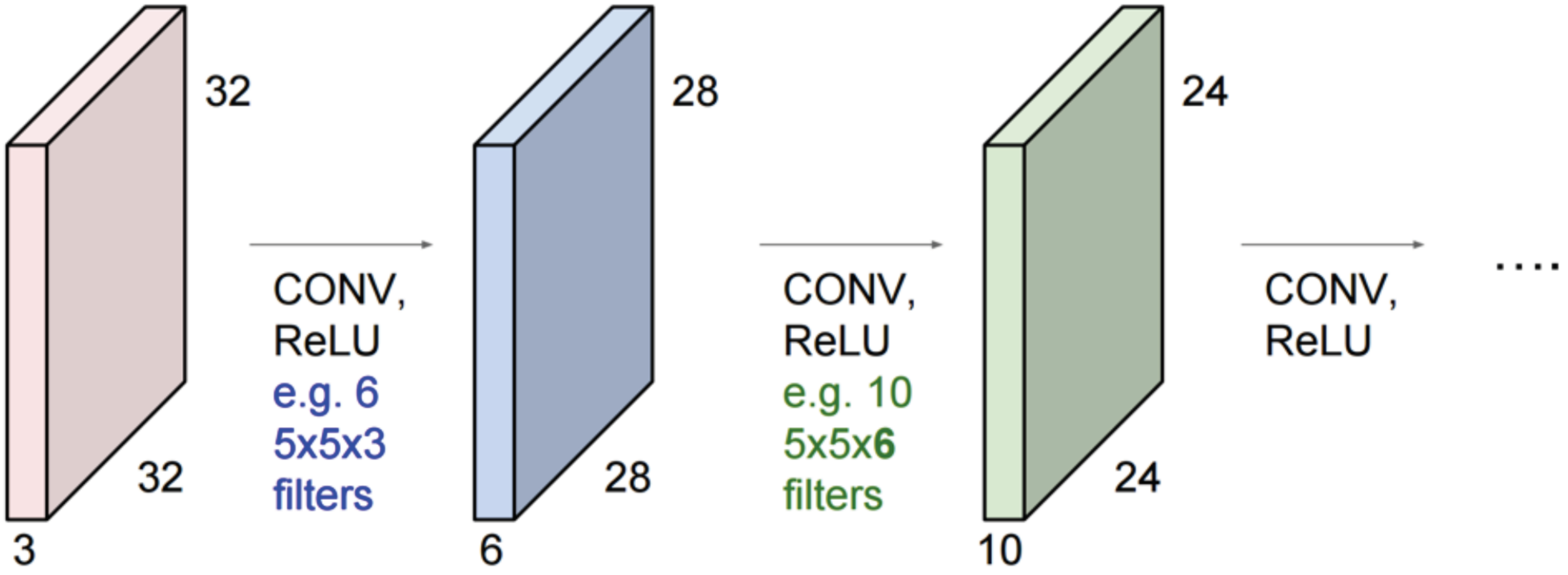


Figure: Andrej Karpathy

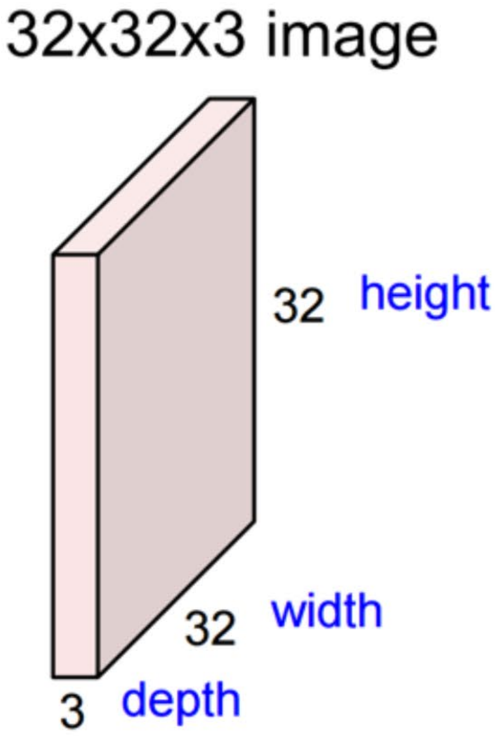
(Recap)

A **ConvNet** is a sequence of convolutional layers, interspersed with activation functions (and possibly other layer types)



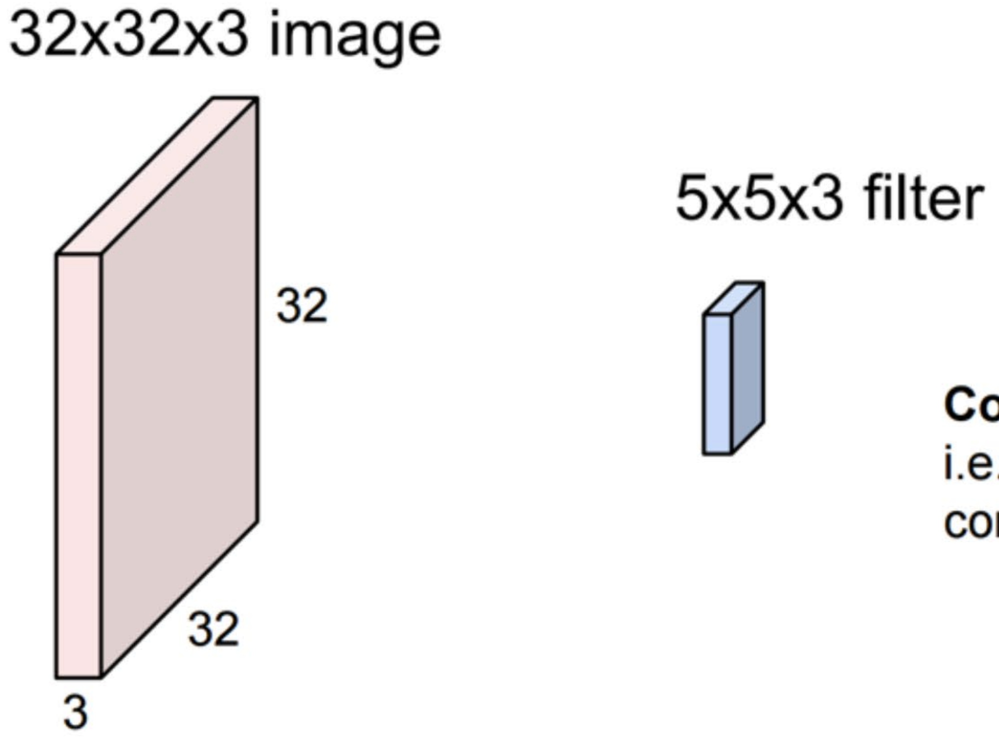
(Recap)

Convolution Layer



(Recap)

Convolution Layer



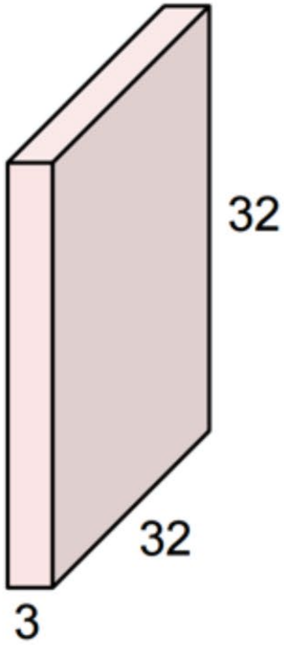
Convolve the filter with the image
i.e. "slide over the image spatially,
computing dot products"

(Recap)

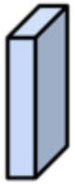
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



5x5x3 filter



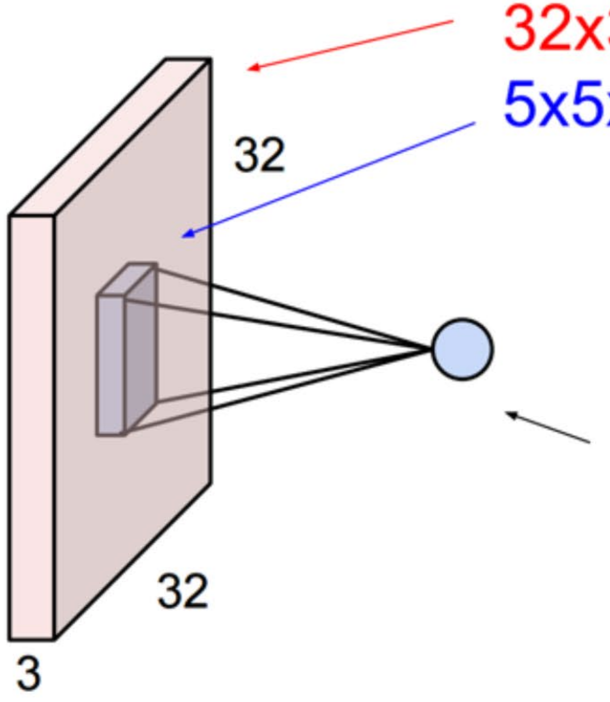
Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

(Recap)

Convolution Layer

32x32x3 image

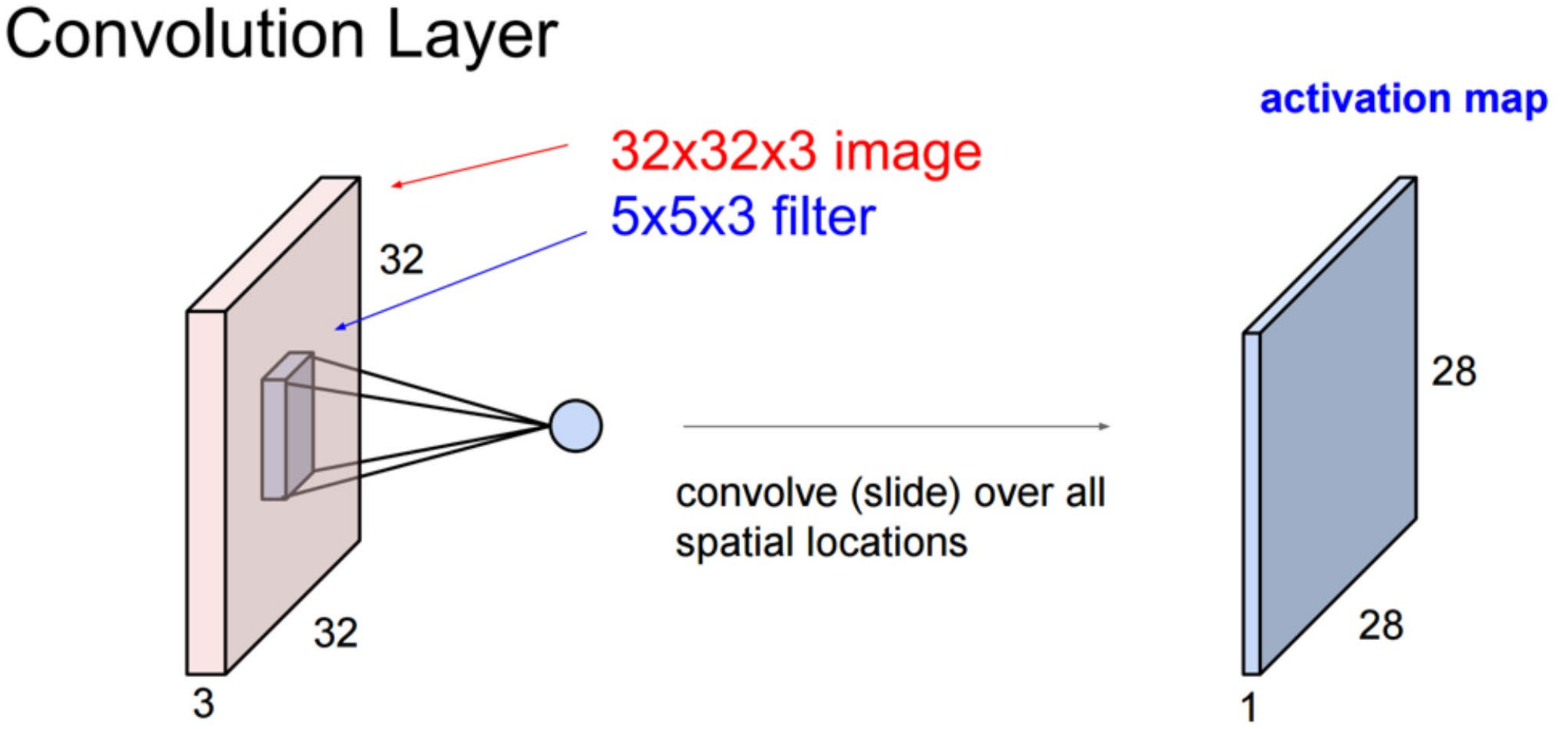
5x5x3 filter w



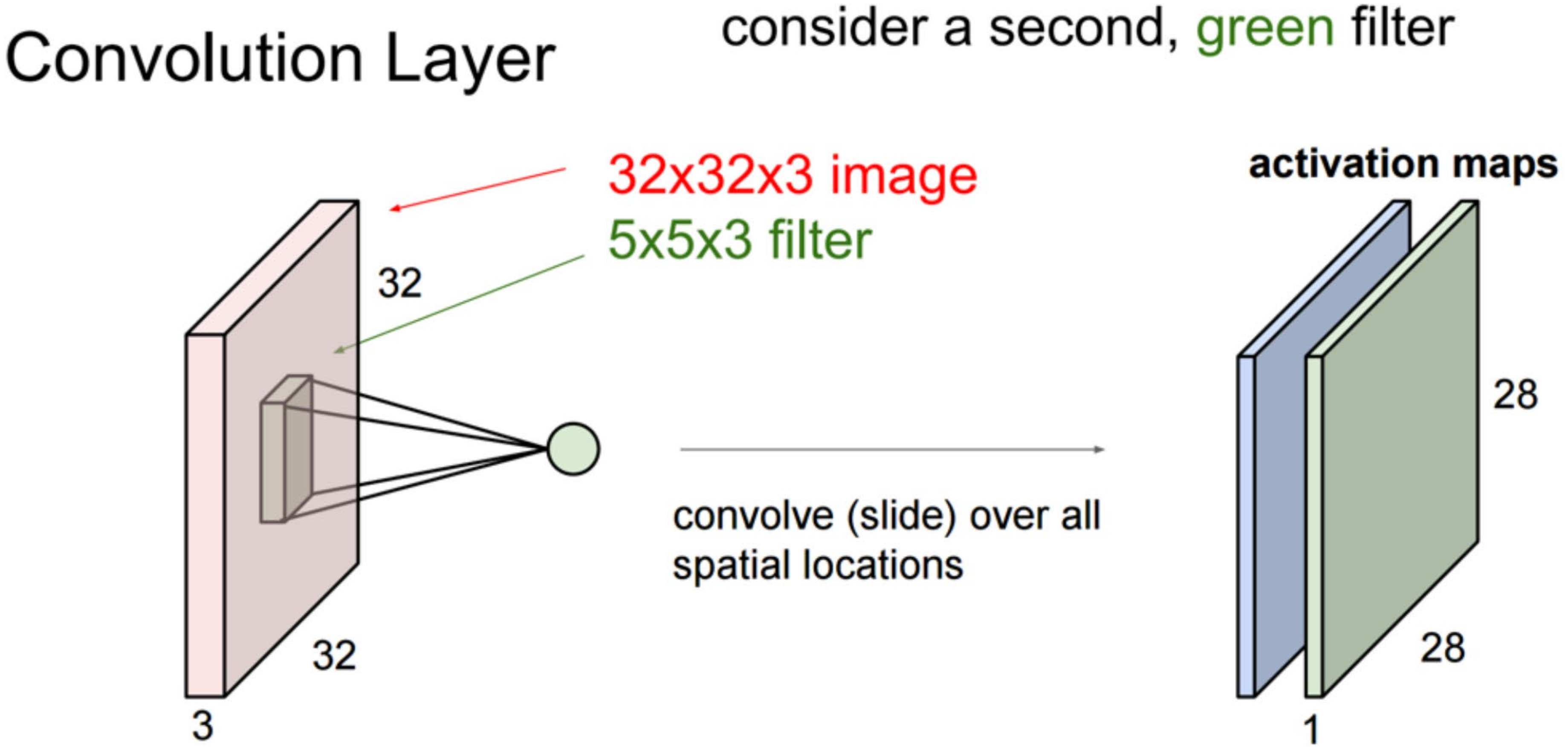
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

(Recap)


Convolution Layer
activation map
32x32x3 image
5x5x3 filter
32
32
3
convolve (slide) over all
spatial locations
28
28
1

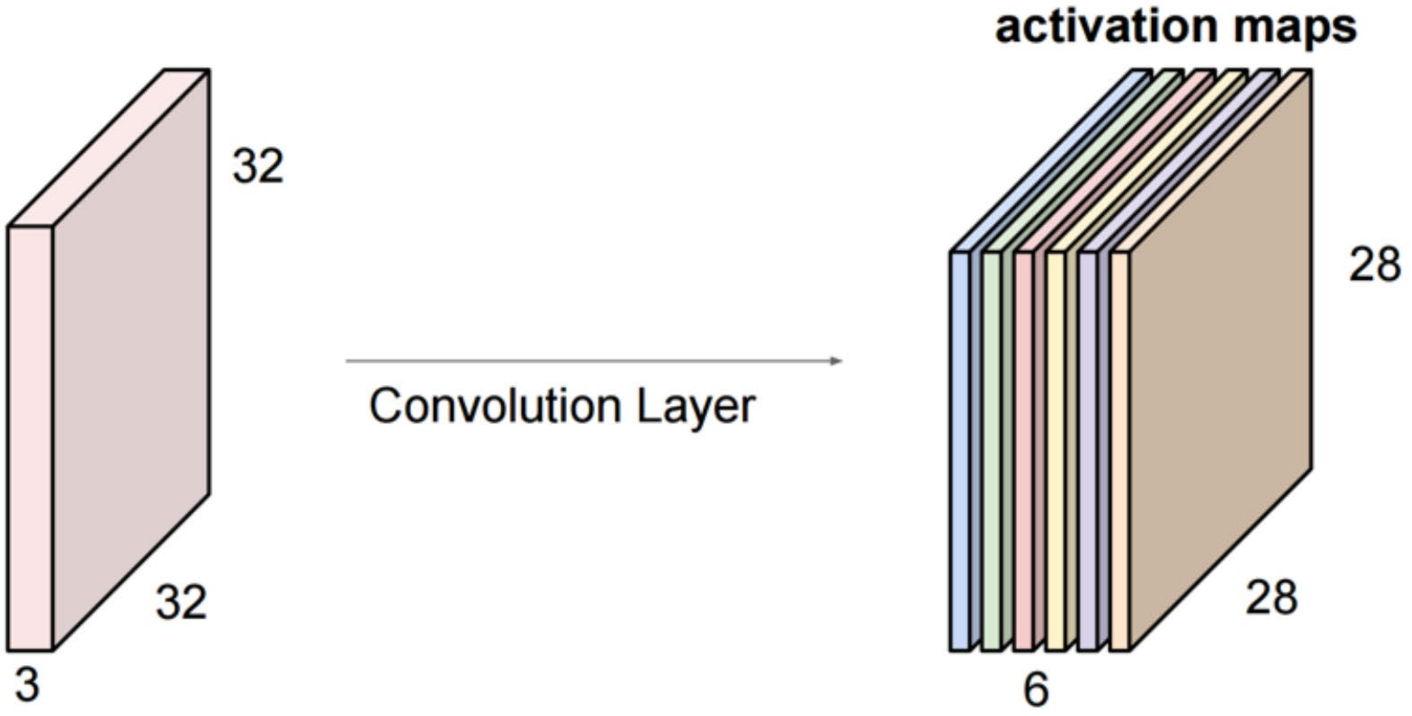
(Recap)



Convolution Layer
consider a second, green filter
32x32x3 image
5x5x3 filter
32
32
3
convolve (slide) over all
spatial locations
activation maps
28
28
1

(Recap)

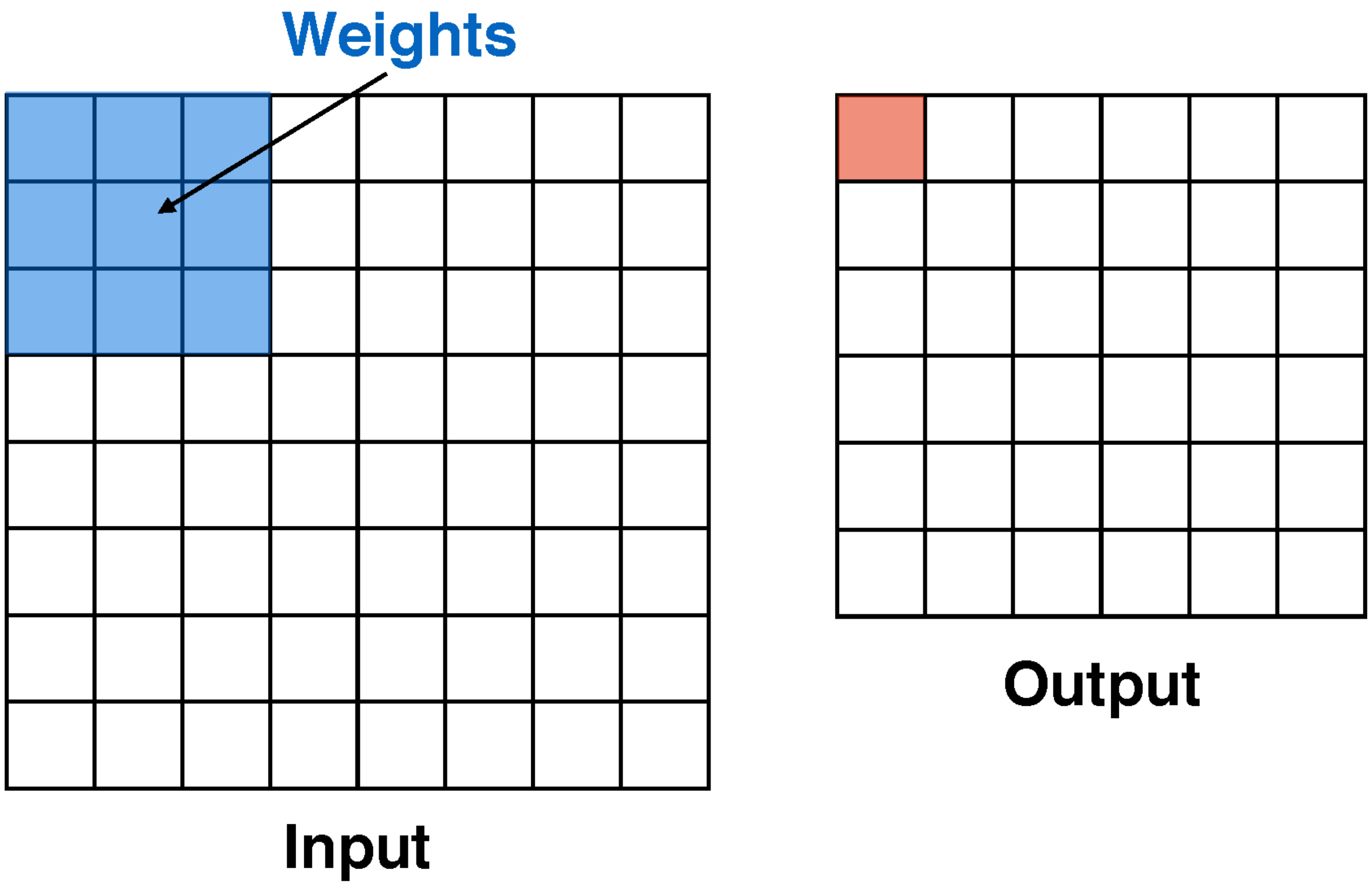
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

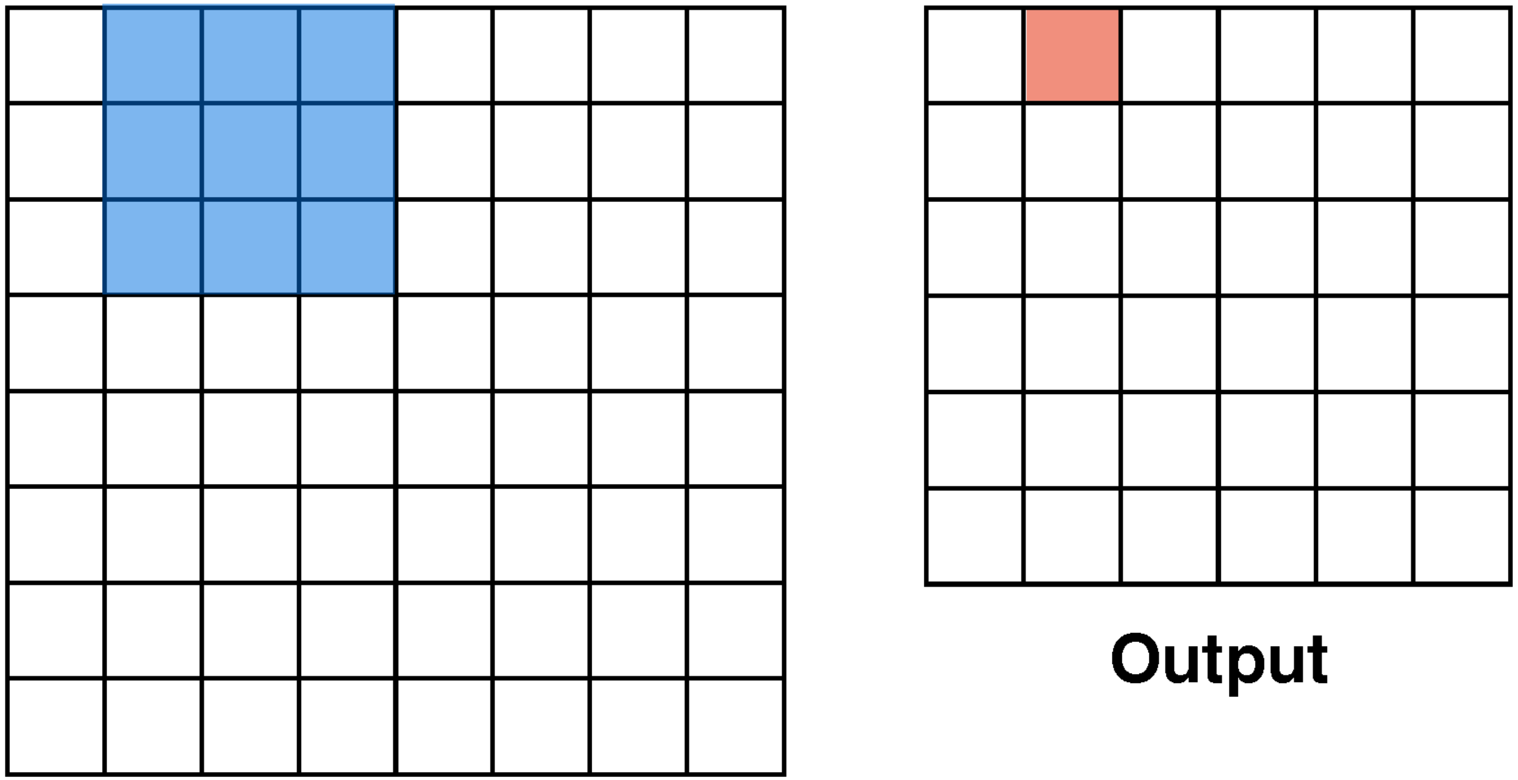
Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



Convolution: Stride

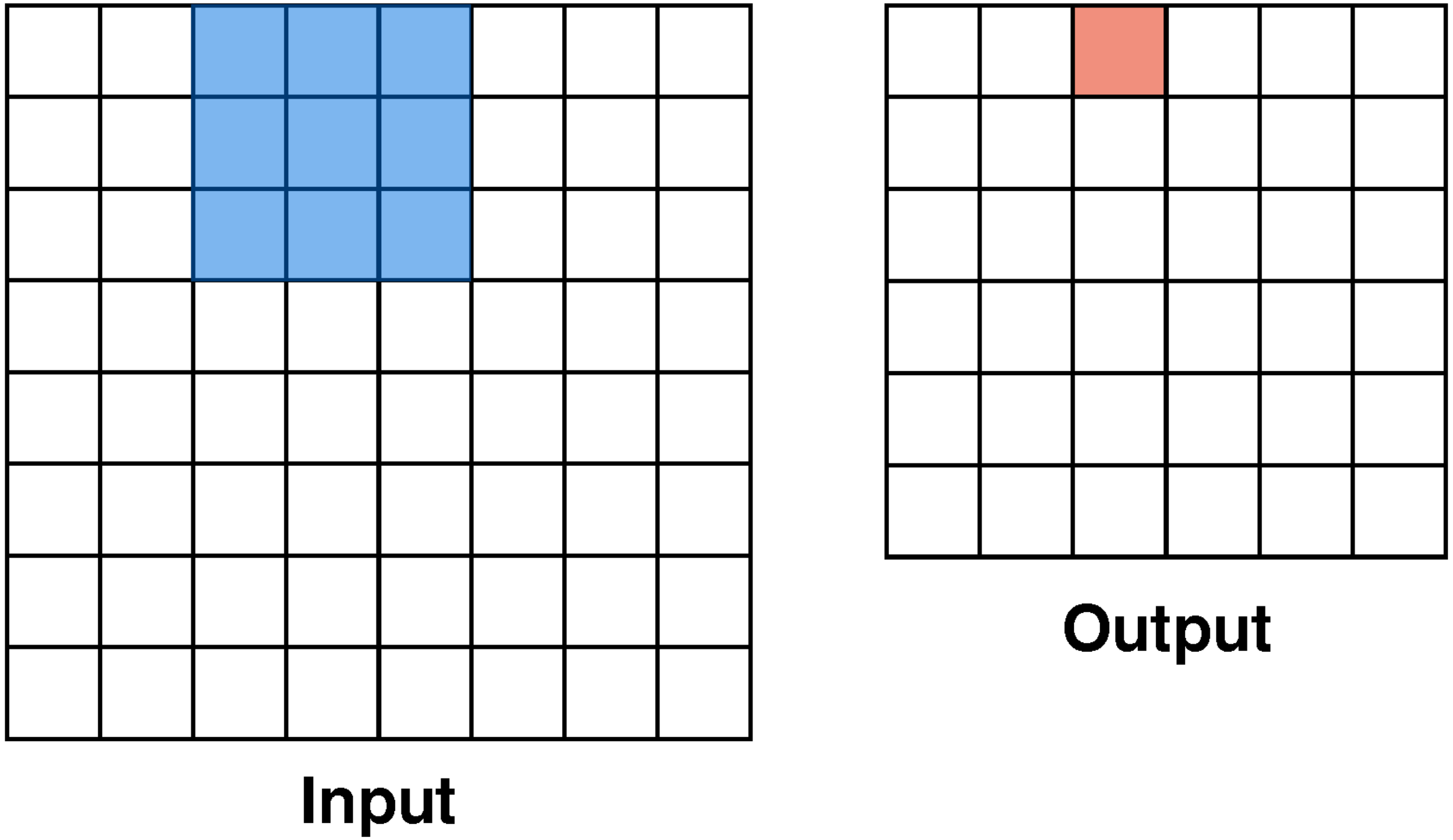
During convolution, the weights "slide" along the input to generate each output



Input　　　　Output

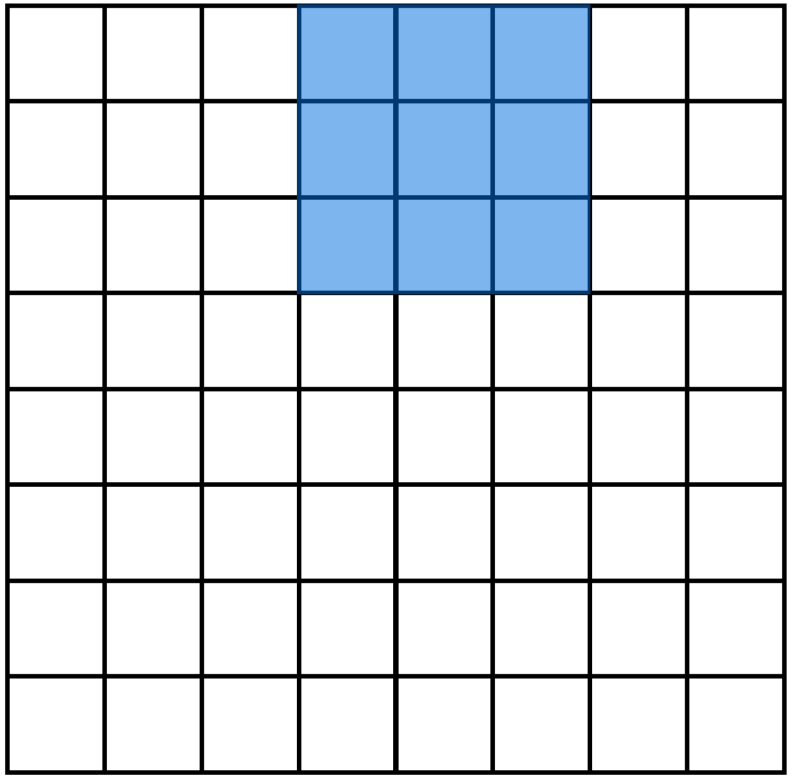
Convolution: Stride

During convolution, the weights "slide" along the input to generate each output

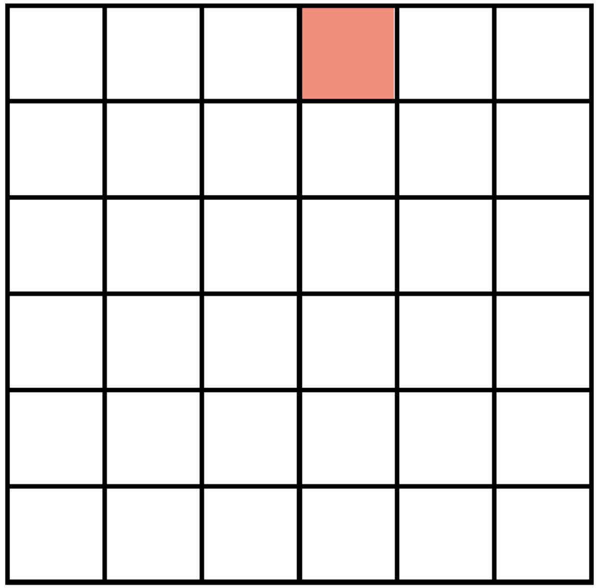


Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



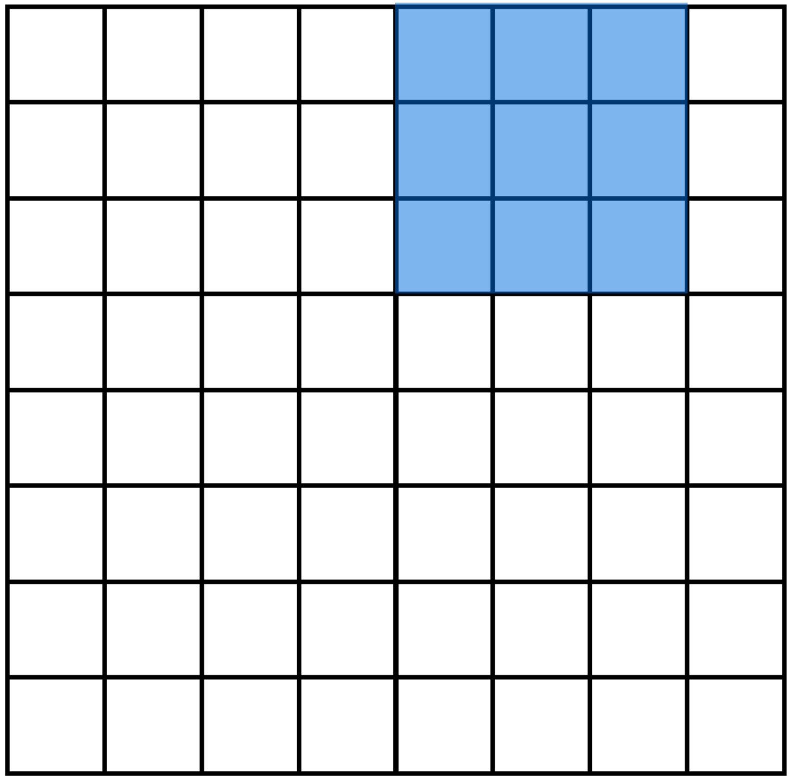
Input



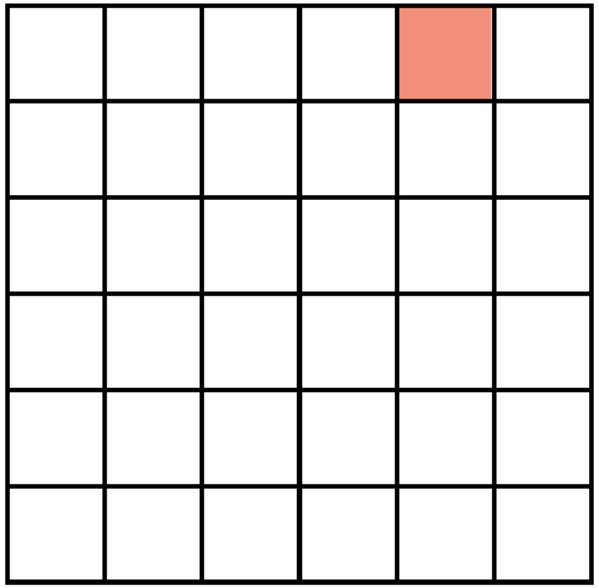
Output

Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



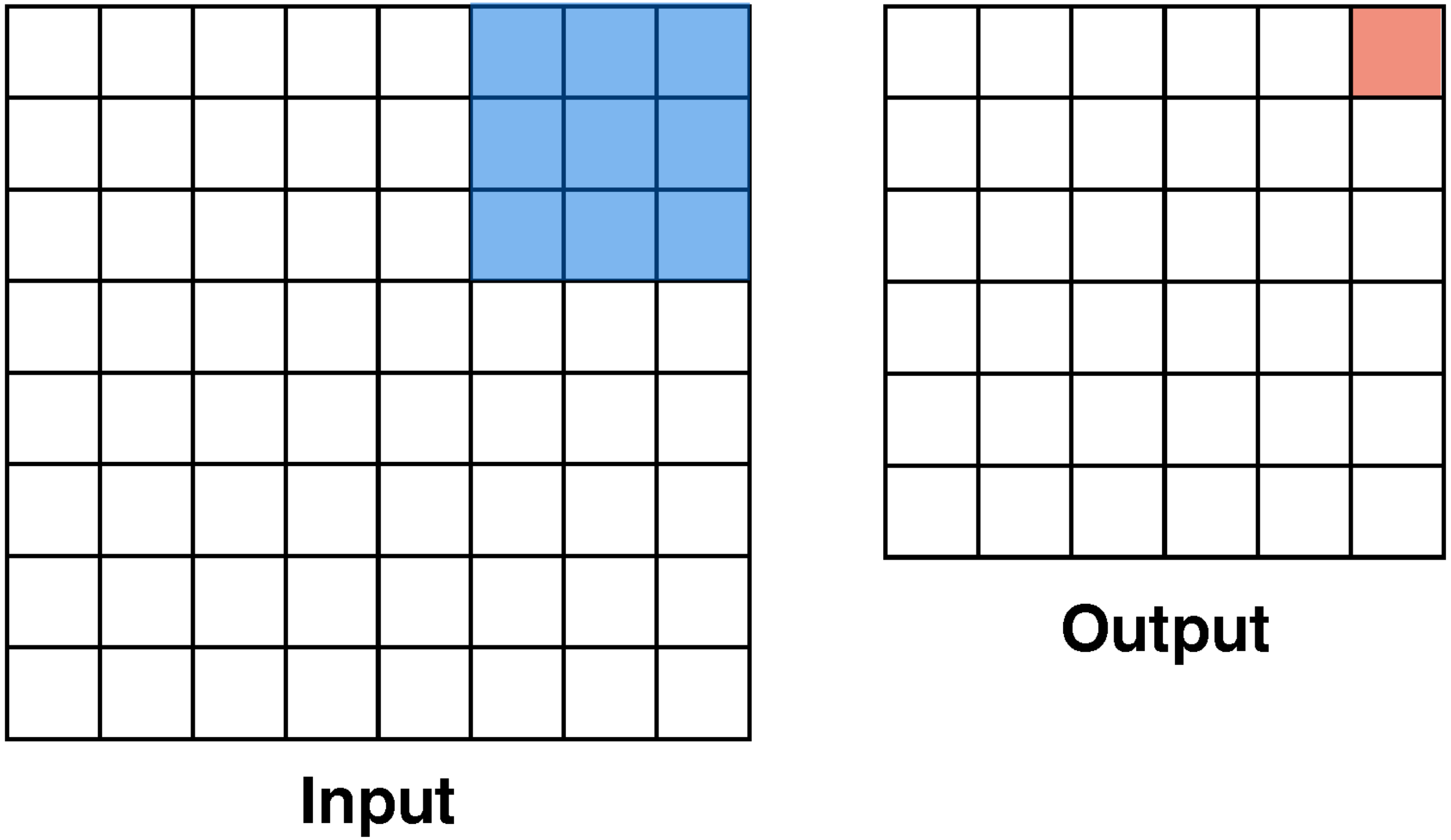
Input



Output

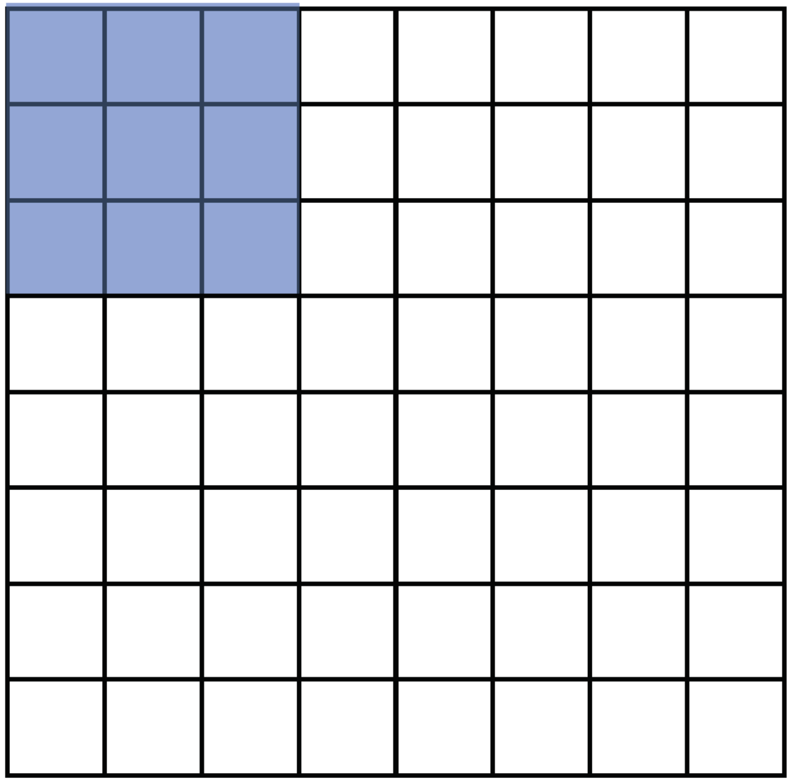
Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



Convolution: Stride

During convolution, the weights "slide" along the input to generate each output



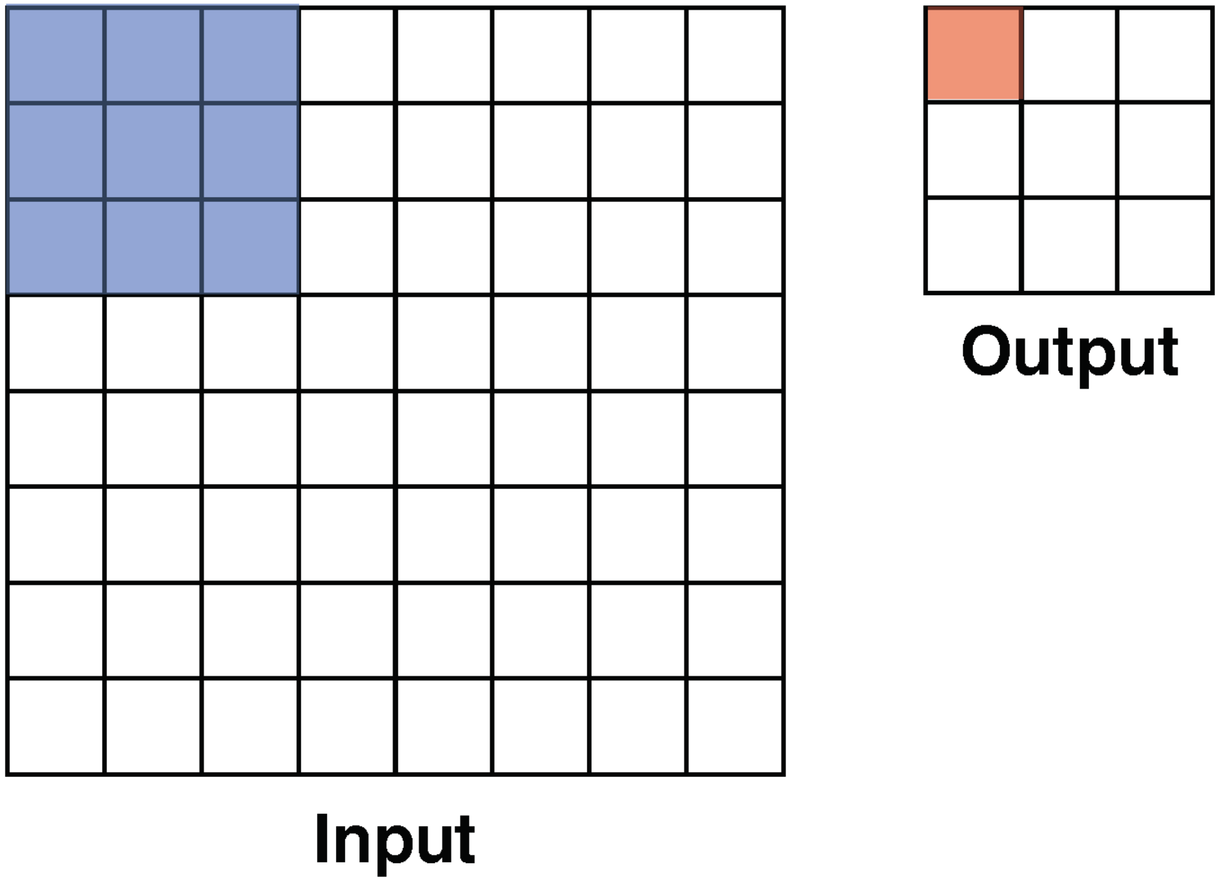
Recall that at each position, we are doing a **3D** sum:

$$h^r = \sum_{ijk} x^r{}_{ijk} W_{ijk} + b$$

(channel, row, column)

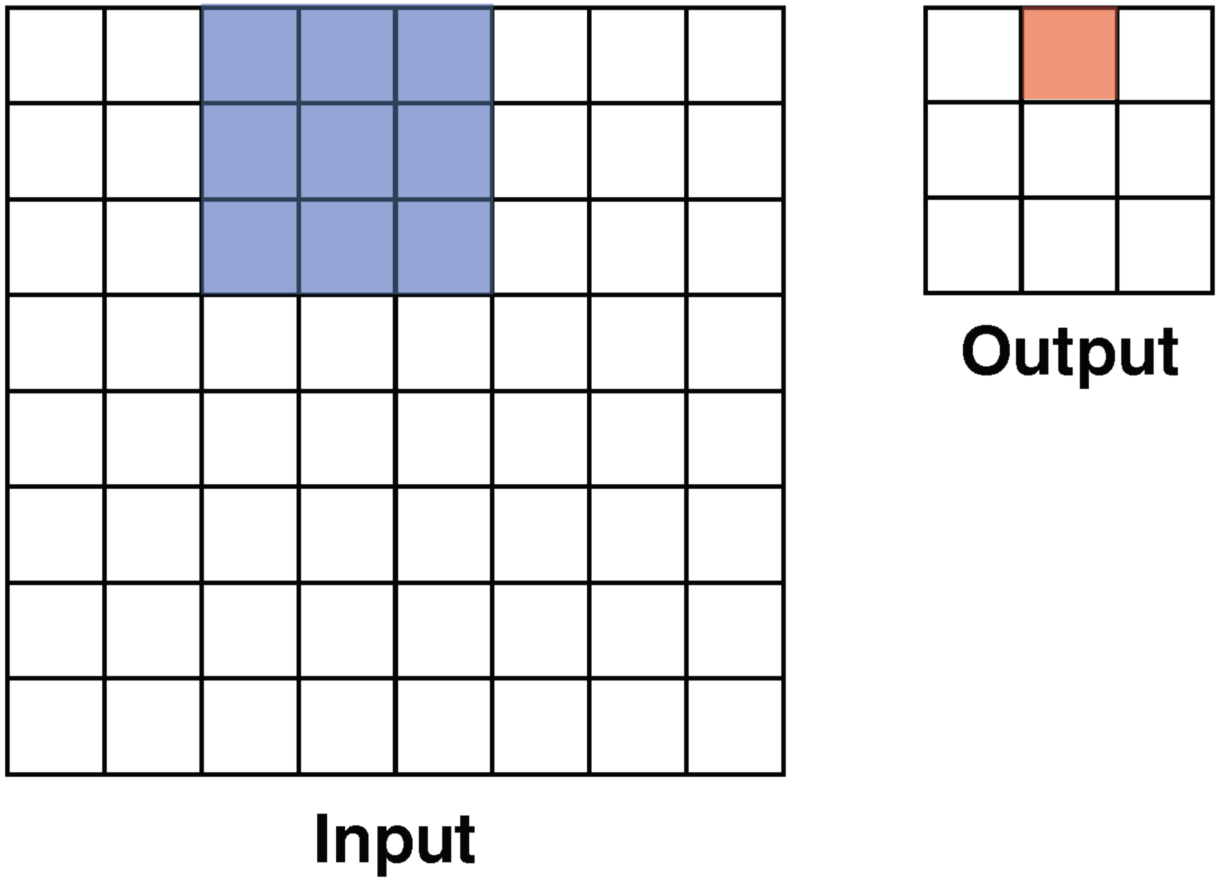
Convolution: Stride

But we can also convolve with a **stride**, e.g. stride = 2



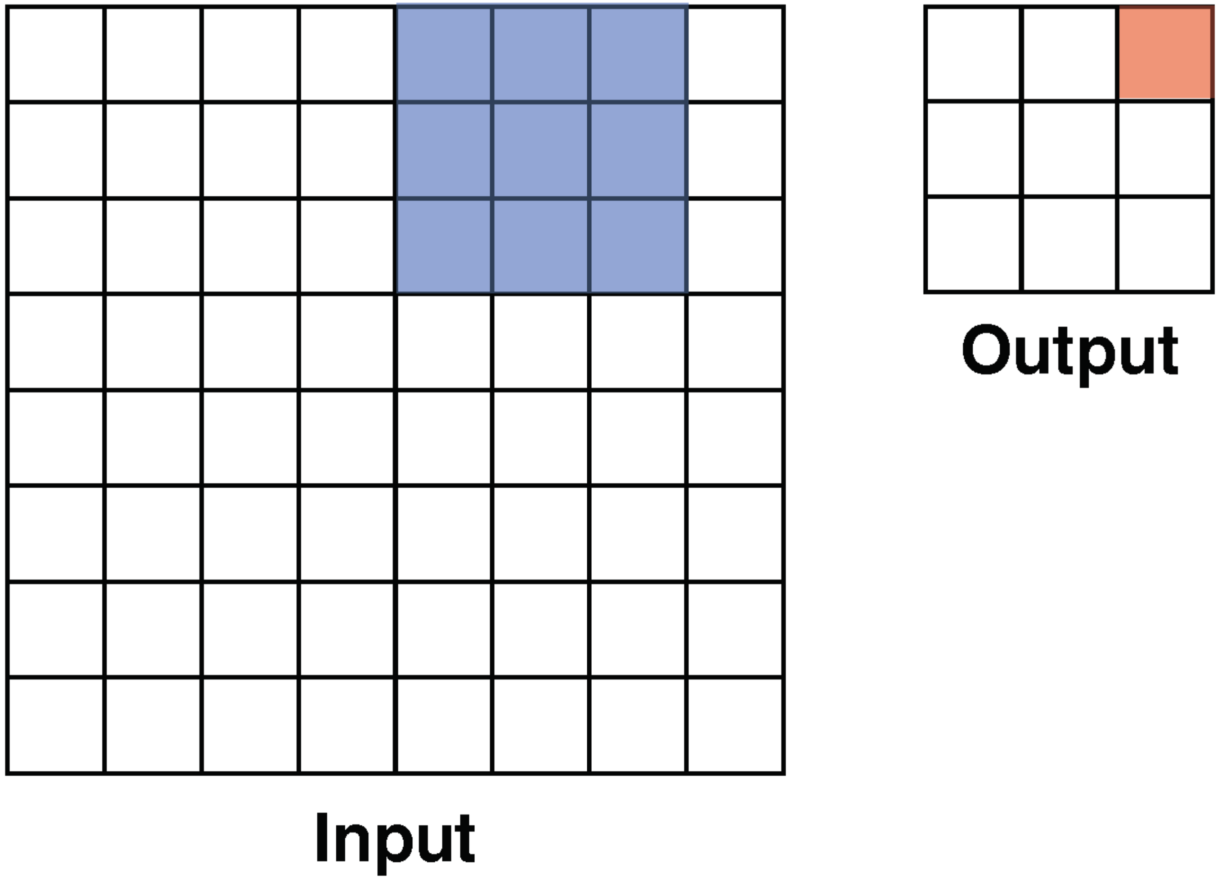
Convolution: Stride

But we can also convolve with a **stride**, e.g. stride = 2



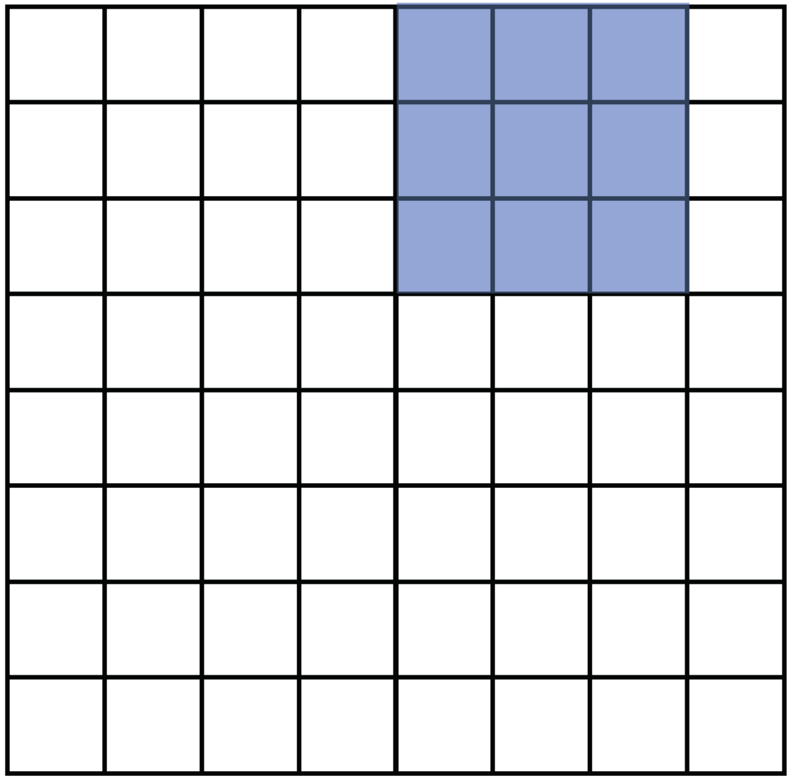
Convolution: Stride

But we can also convolve with a **stride**, e.g. stride = 2

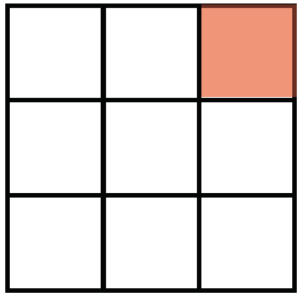


Convolution: Stride

But we can also convolve with a **stride**, e.g. stride = 2



Input



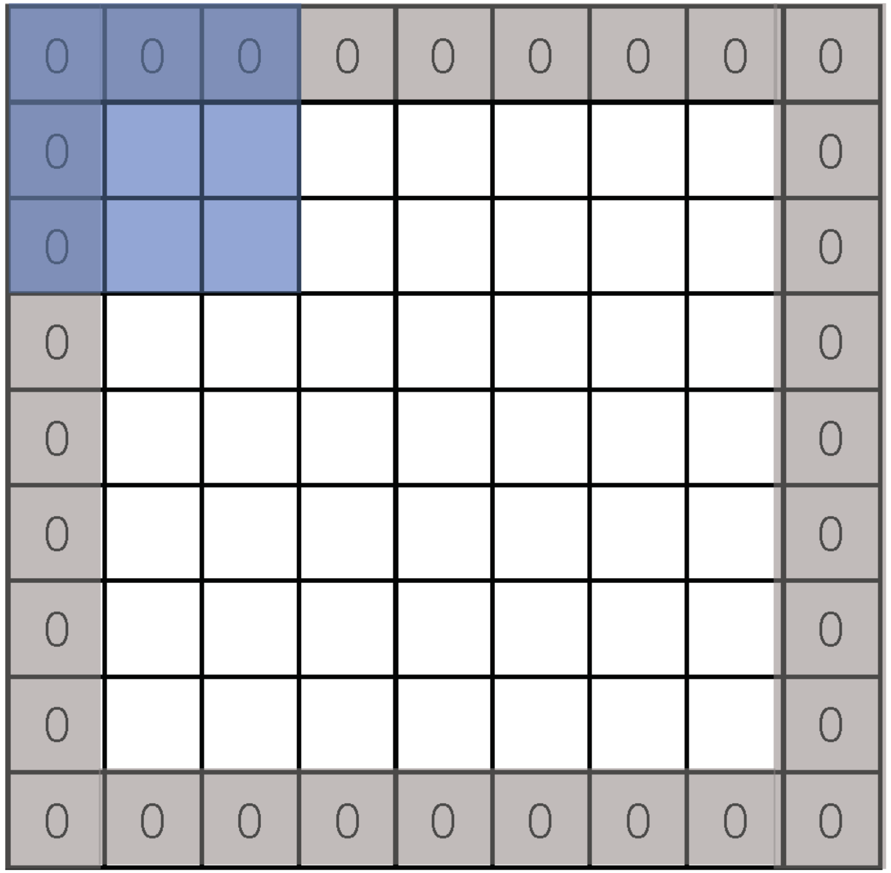
Output

- *Notice that with certain strides, we may not be able to cover all of the input*

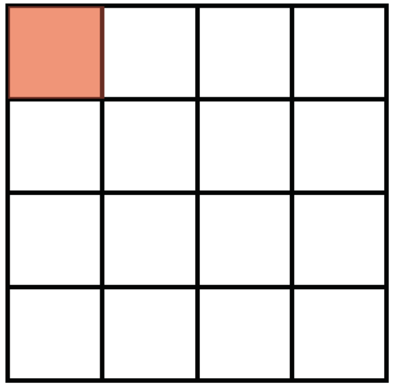
- *The output is also half the size of the input*

Convolution: Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



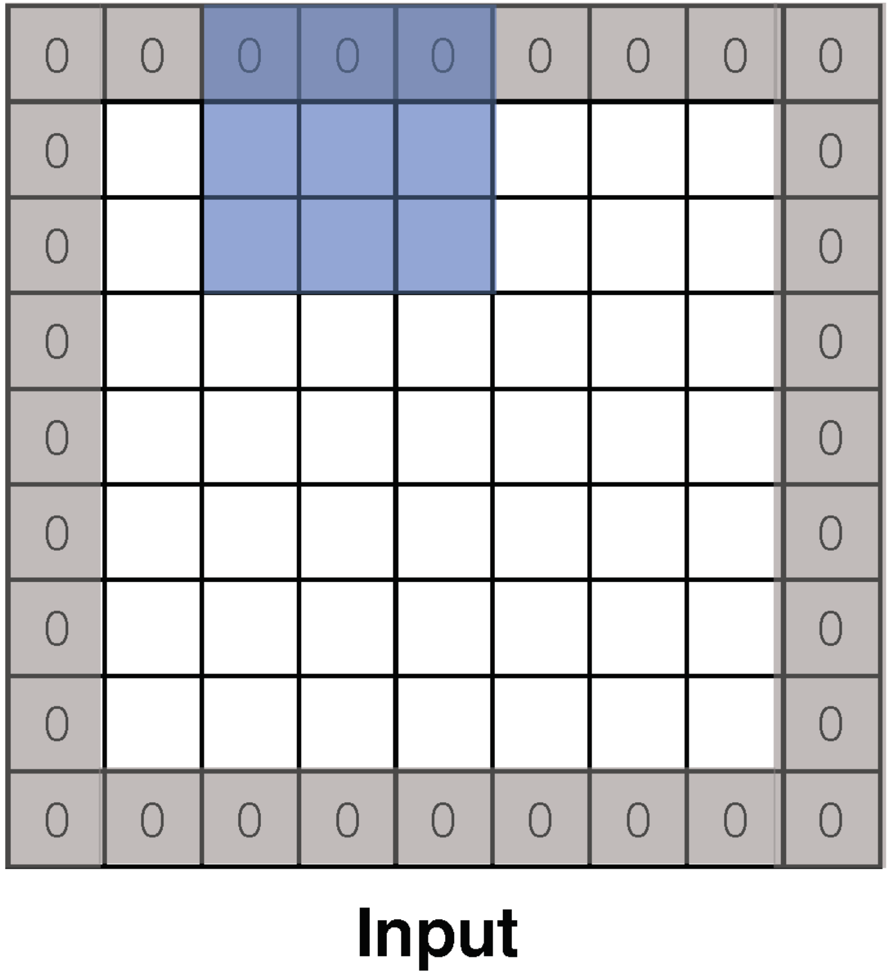
Input

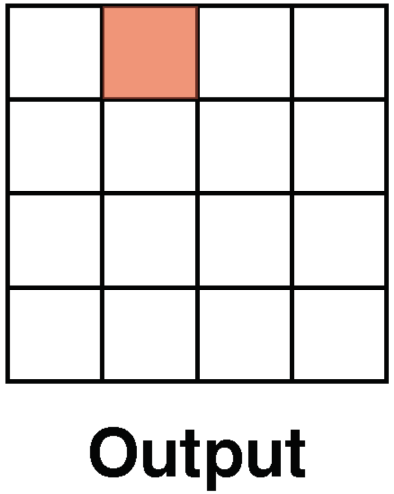


Output

Convolution: Padding

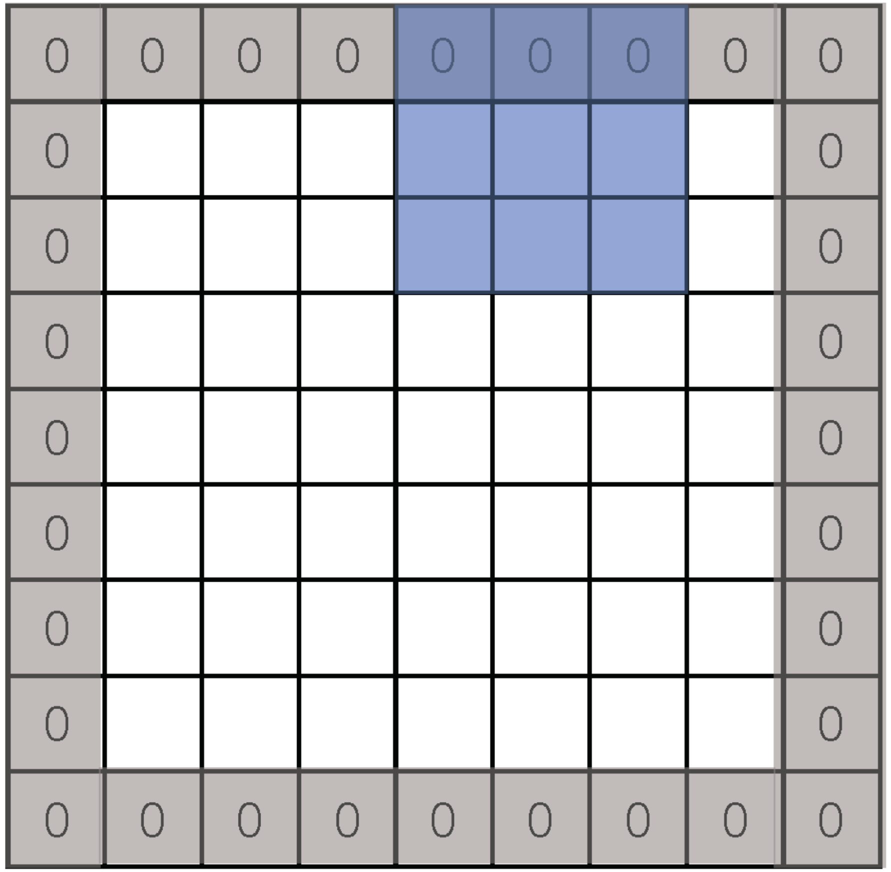
We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



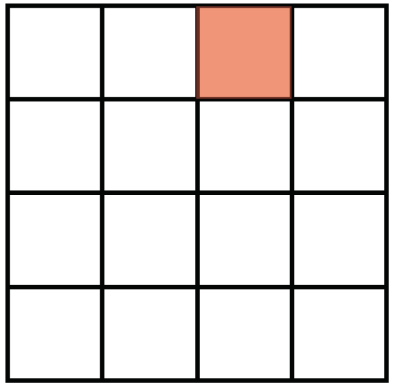


Convolution: Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



Input

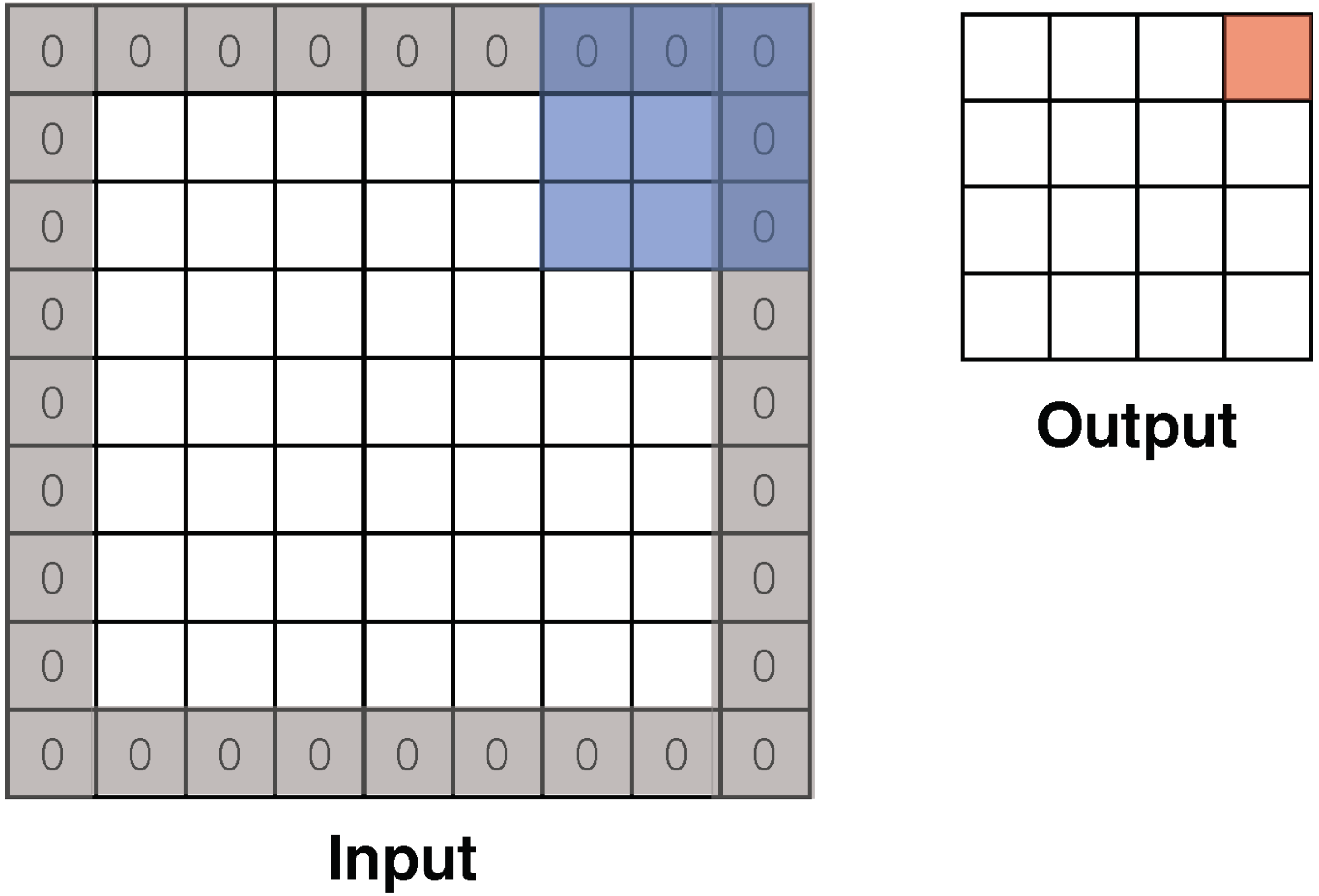


Output

Convolution: Padding

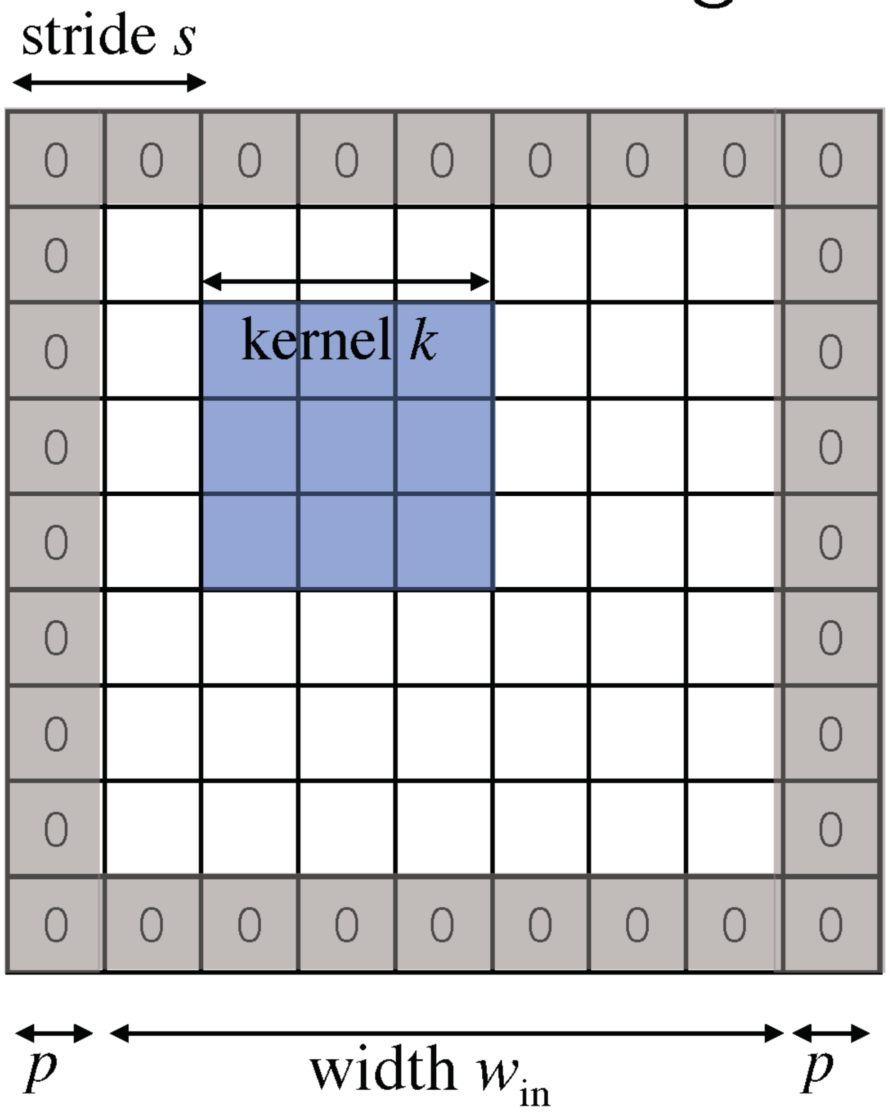
We can also pad the input with zeros.

Here, **pad = 1, stride = 2**



Convolution:

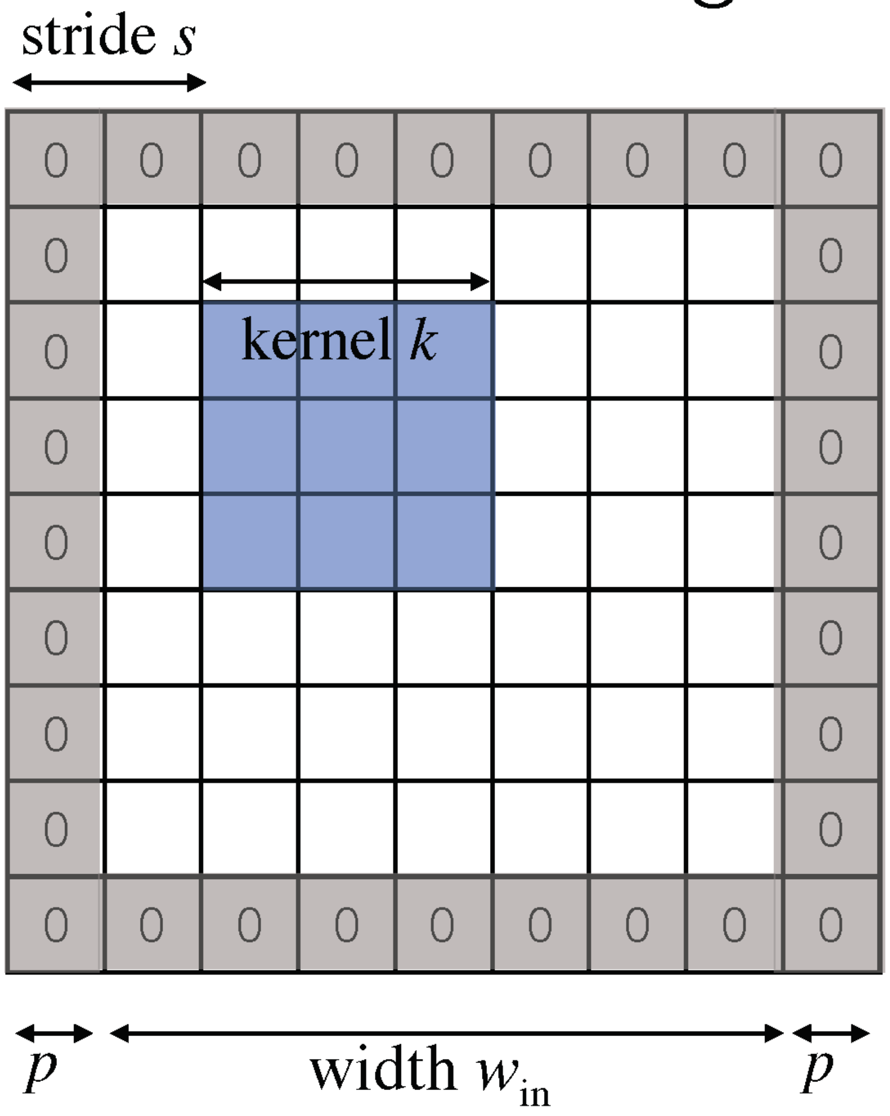
How big is the output?



In general, the output has size:

$$w_{\text{out}} = \left\lfloor \frac{w_{\text{in}} + 2p - k}{s} \right\rfloor + 1$$

Convolution:
How big is the output?



Example: k=3, s=1, p=1

$$w_{out} = \left\lfloor \frac{w_{in} + 2p - k}{s} \right\rfloor + 1$$

$$= \left\lfloor \frac{w_{in} + 2 - 3}{1} \right\rfloor + 1$$

$$= w_{in}$$

VGGNet [Simonyan 2014] uses filters of this shape

Pooling

For most ConvNets, **convolution** is often followed by **pooling**:

- Creates a smaller representation while retaining the most important information
- The "max" operation is the most common
- Why might "avg" be a poor choice?

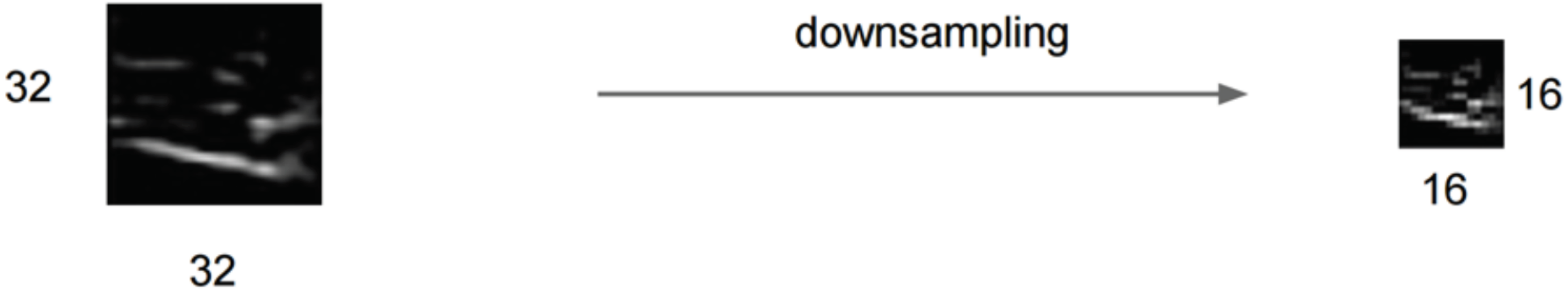
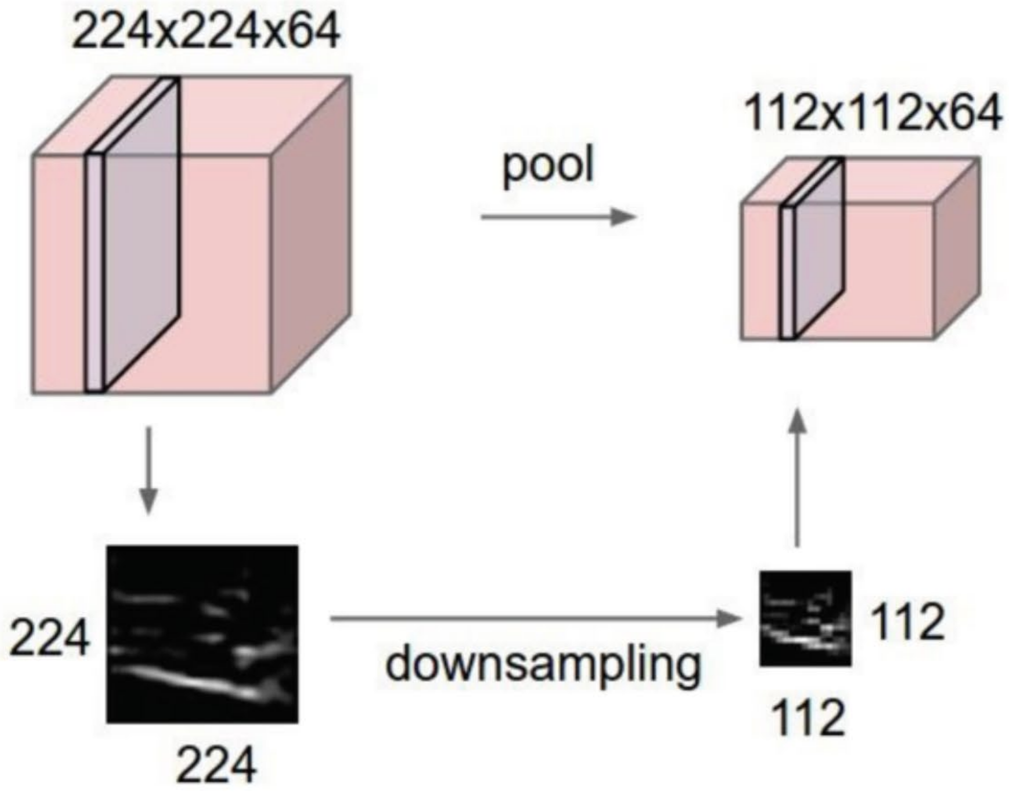


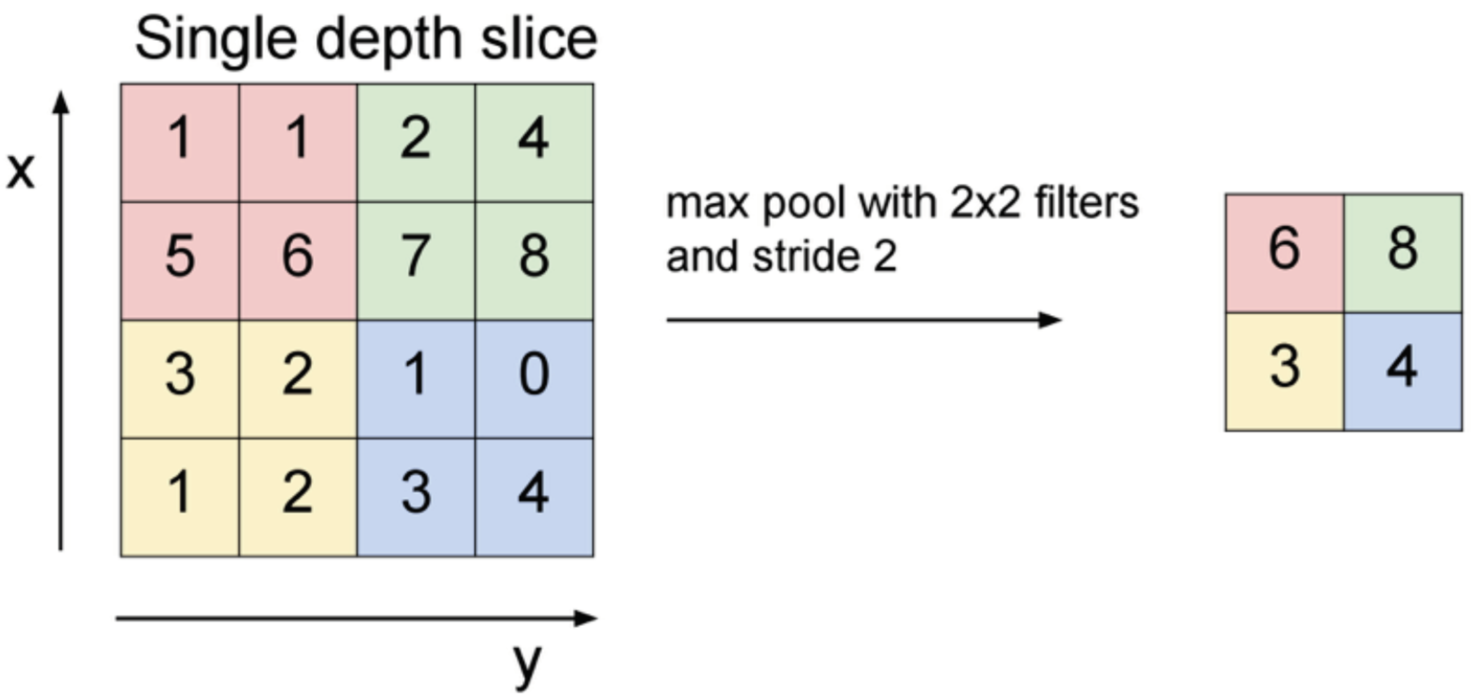
Figure: Andrej Karpathy

Pooling

- makes the representations smaller and more manageable
- operates over each activation map independently:



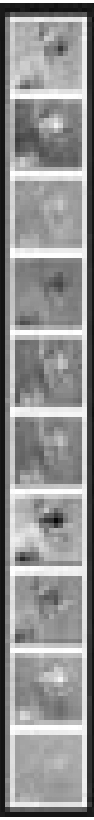
Max Pooling



What's the backprop rule for max pooling?

- In the forward pass, store the index that took the max
- The backprop gradient is the input gradient at that index

Figure: Andrej Karpathy



Example ConvNet

Figure: Andrej Karpathy

Example ConvNet

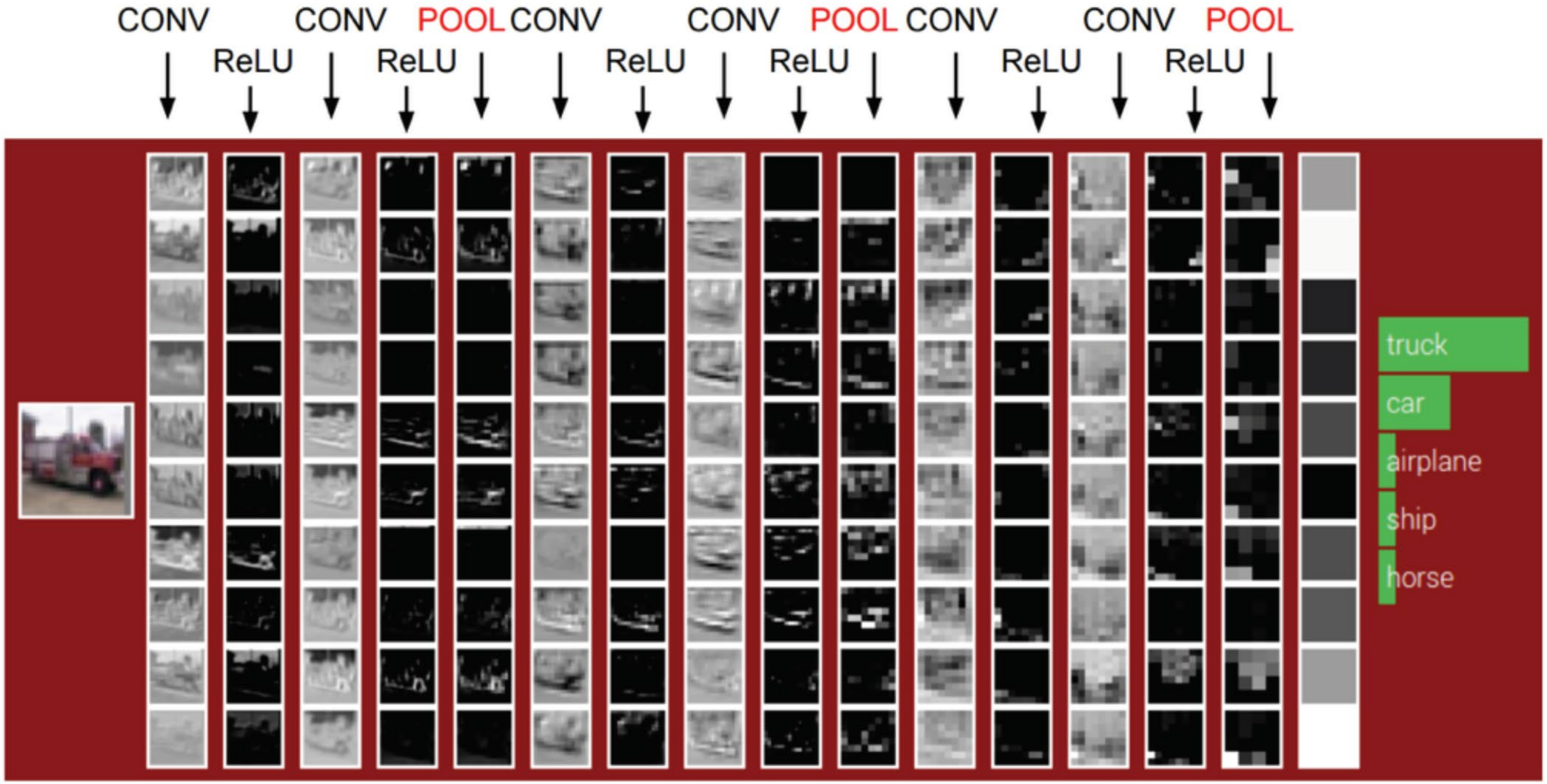


Figure: Andrej Karpathy

Example ConvNet

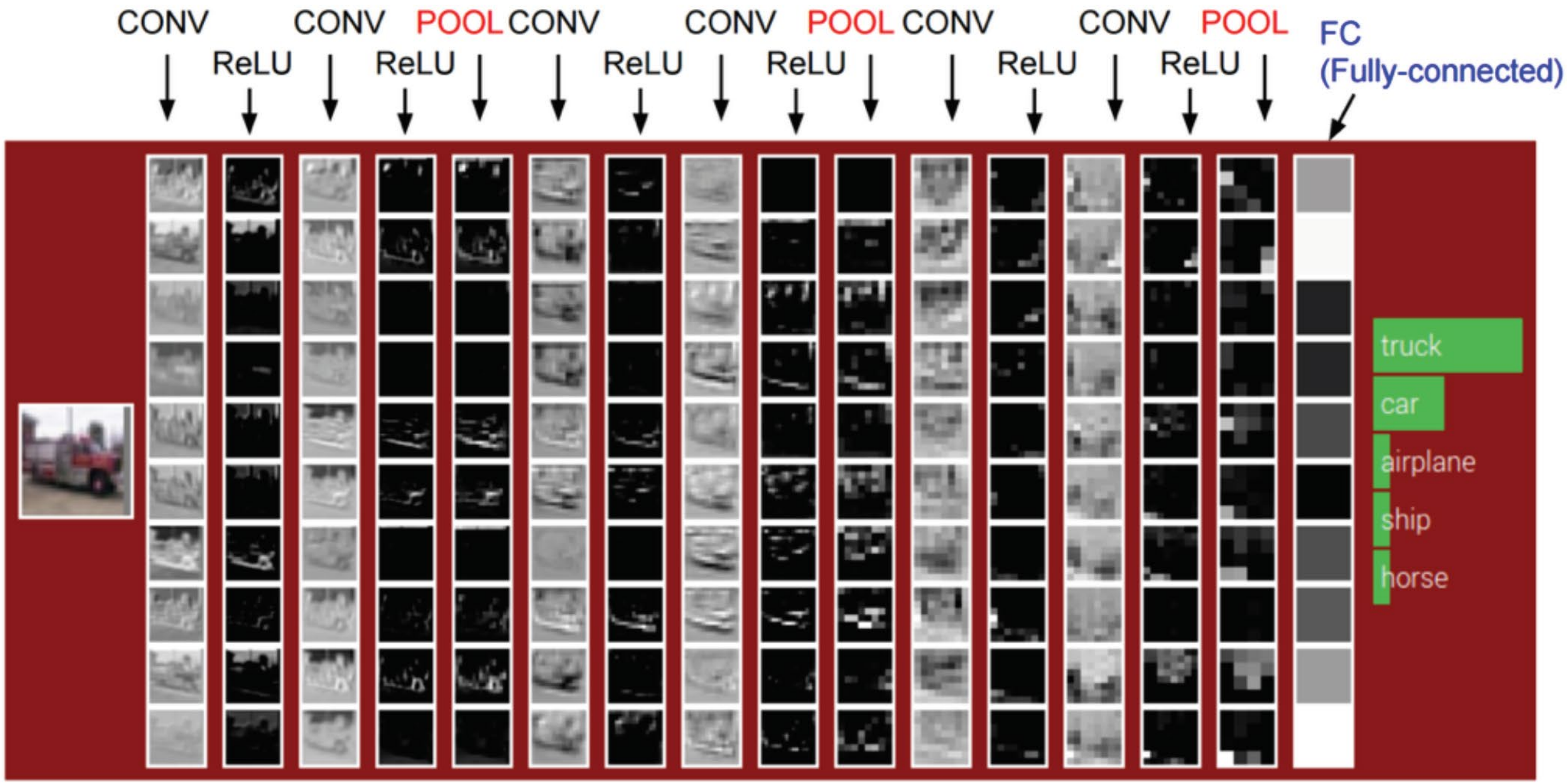
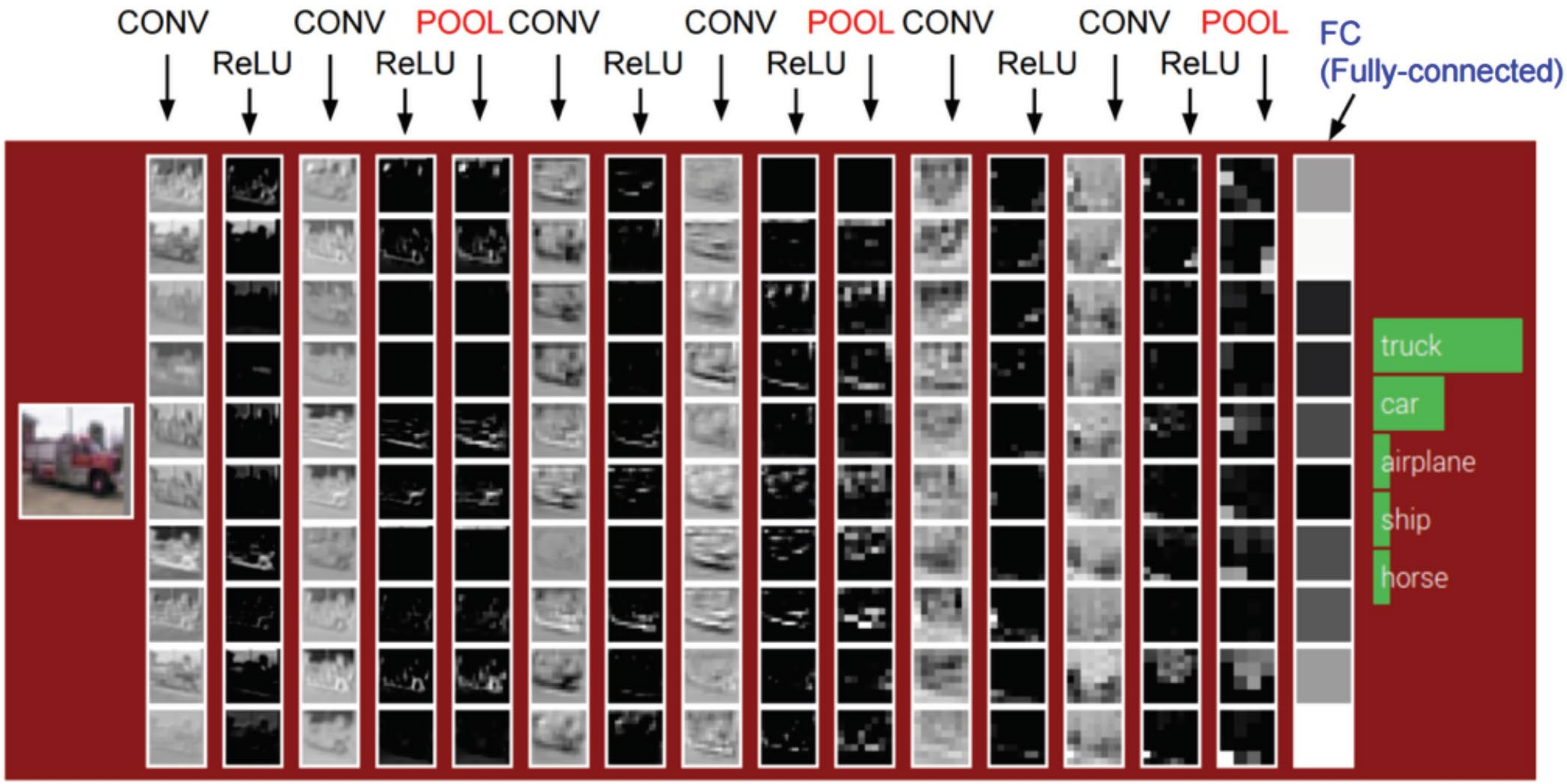


Figure: Andrej Karpathy

Example ConvNet

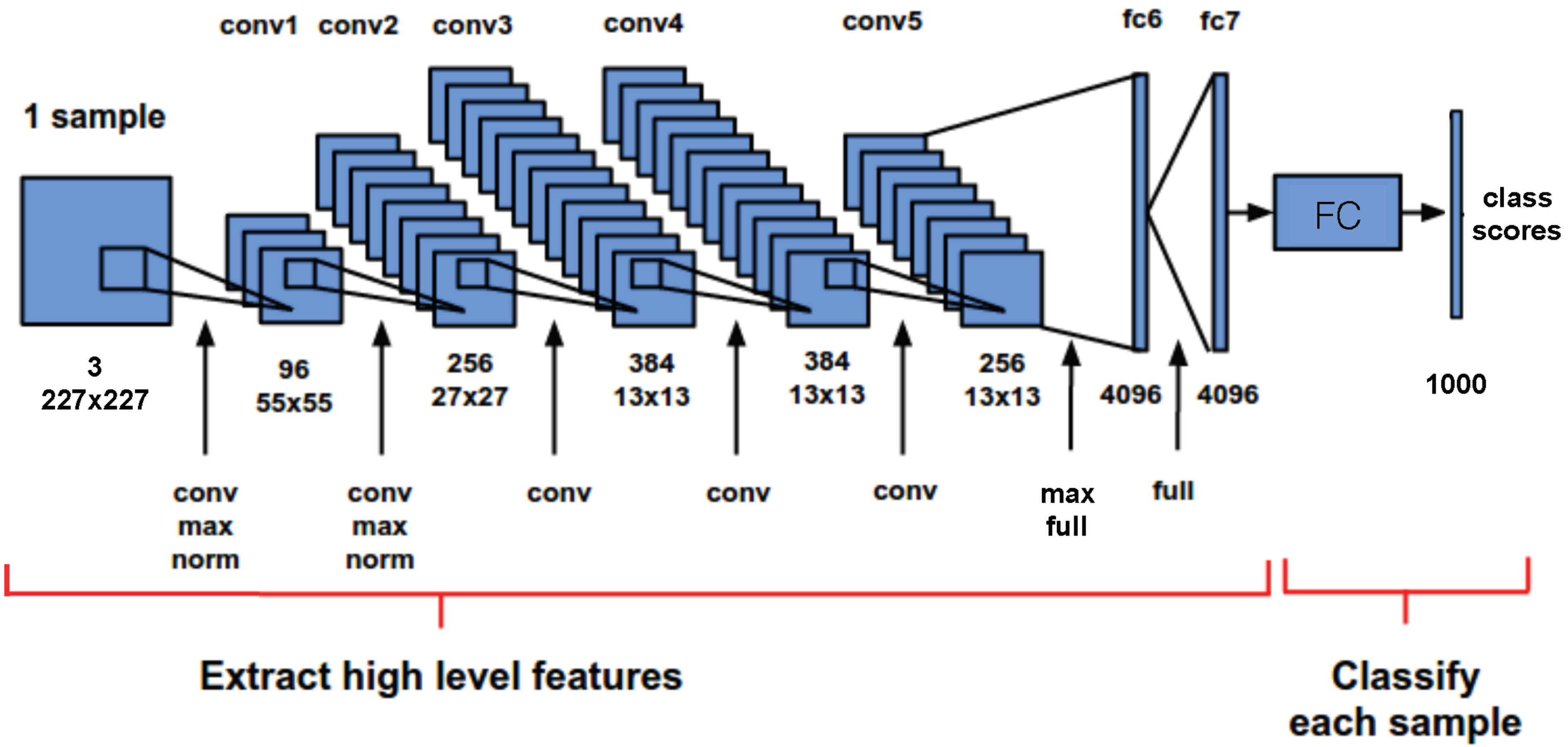


10x3x3 conv filters, stride 1, pad 1

2x2 pool filters, stride 2

Figure: Andrej Karpathy

Example: AlexNet [Krizhevsky 2012]



“max”: max pooling
“norm”: local response normalization
“full”: fully connected

Figure: [Karnowski 2015] *(with corrections)*