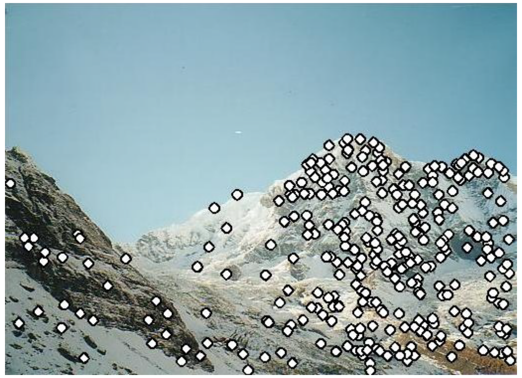
CMSC 491/691

Features Recap

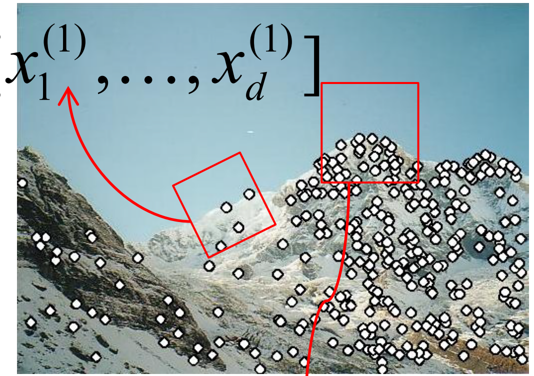
Features: main components

1) Detection: Identify the interest points



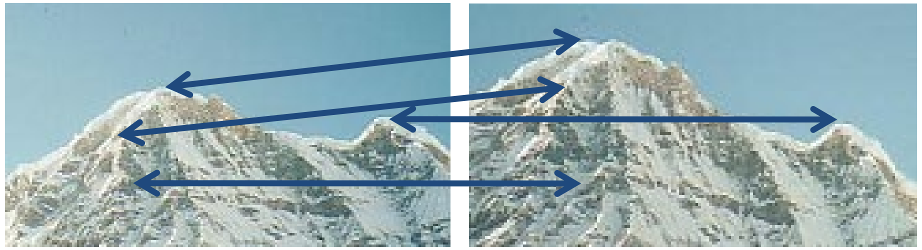
2) **Description: Extract vector feature descriptor surrounding each interest point.**

$$\mathbf{x}_1 = [x_1^{(1)}, \ldots, x_d^{(1)}]$$



$$\mathbf{x}_2 = [x_1^{(2)}, \ldots, x_d^{(2)}]$$

3) Matching: Determine correspondence between descriptors in two views



Kristen Grauman

SIFT

(Scale Invariant Feature Transform)

SIFT describes both a **detector** and **descriptor**

1. Multi-scale extrema detection

2. Keypoint localization

3. Orientation assignment

4. Keypoint descriptor

Scale Invariant Feature Transform

- Take 16x16 square window around detected feature
- Compute edge orientation (angle of the gradient - 90°) for each pixel
- Throw out weak edges (threshold gradient magnitude)
- Create histogram of surviving edge orientations
- Shift the bins so that the biggest one is first

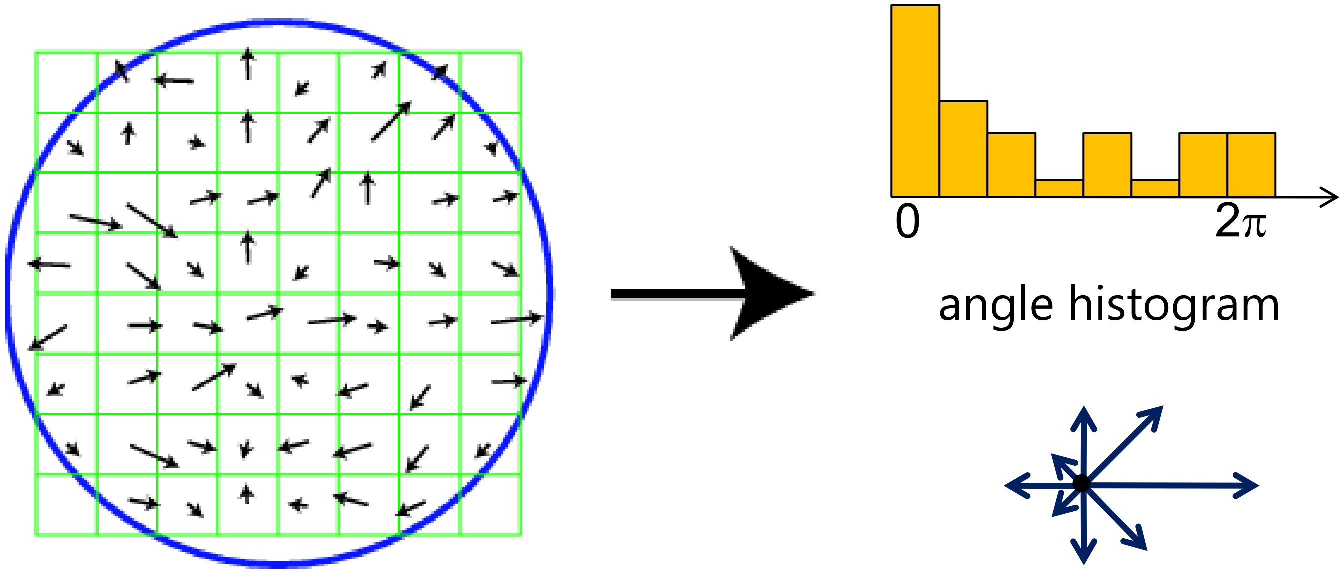
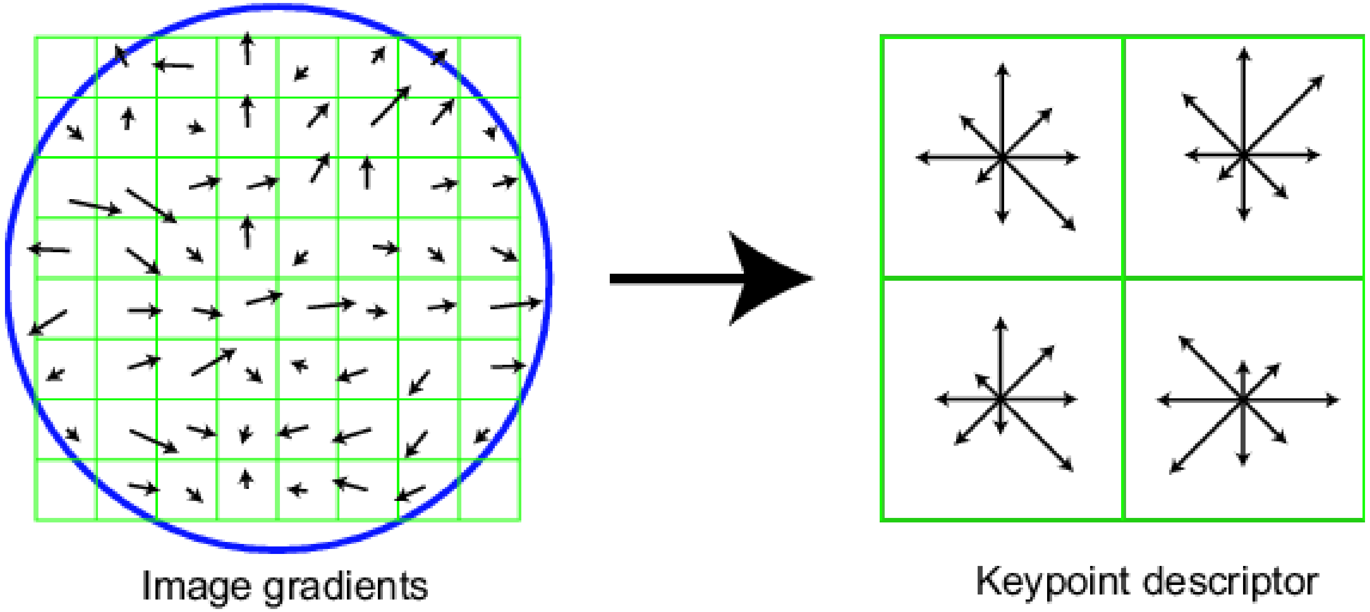


Image gradients

SIFT descriptor

Full version

- Divide the 16x16 window into a 4x4 grid of cells (2x2 case shown below)
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128 dimensional descriptor



Properties of SIFT

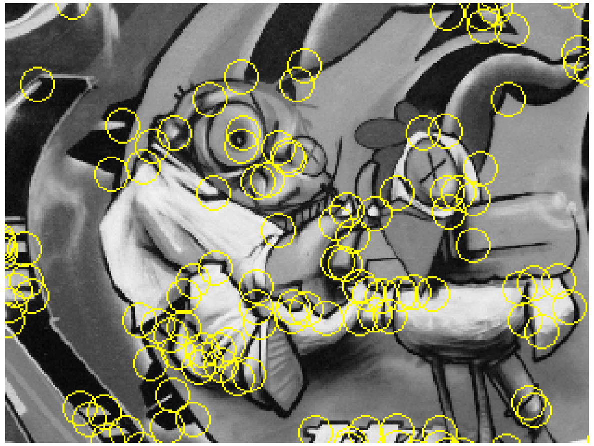
Extraordinarily robust matching technique

- Can handle changes in viewpoint (up to about 60 degree out of plane rotation)
- Can handle significant changes in illumination (sometimes even day vs. night (below))
- Pretty fast—hard to make real-time, but can run in <1s for moderate image sizes
- Lots of code available

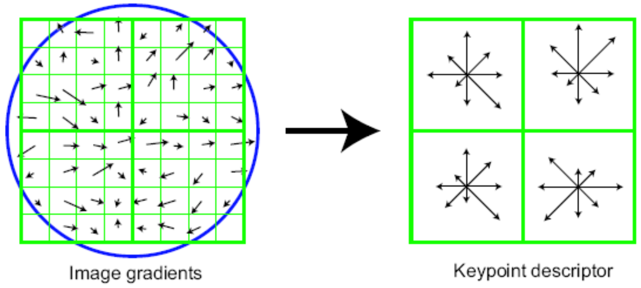
 

Feature Detection and Description

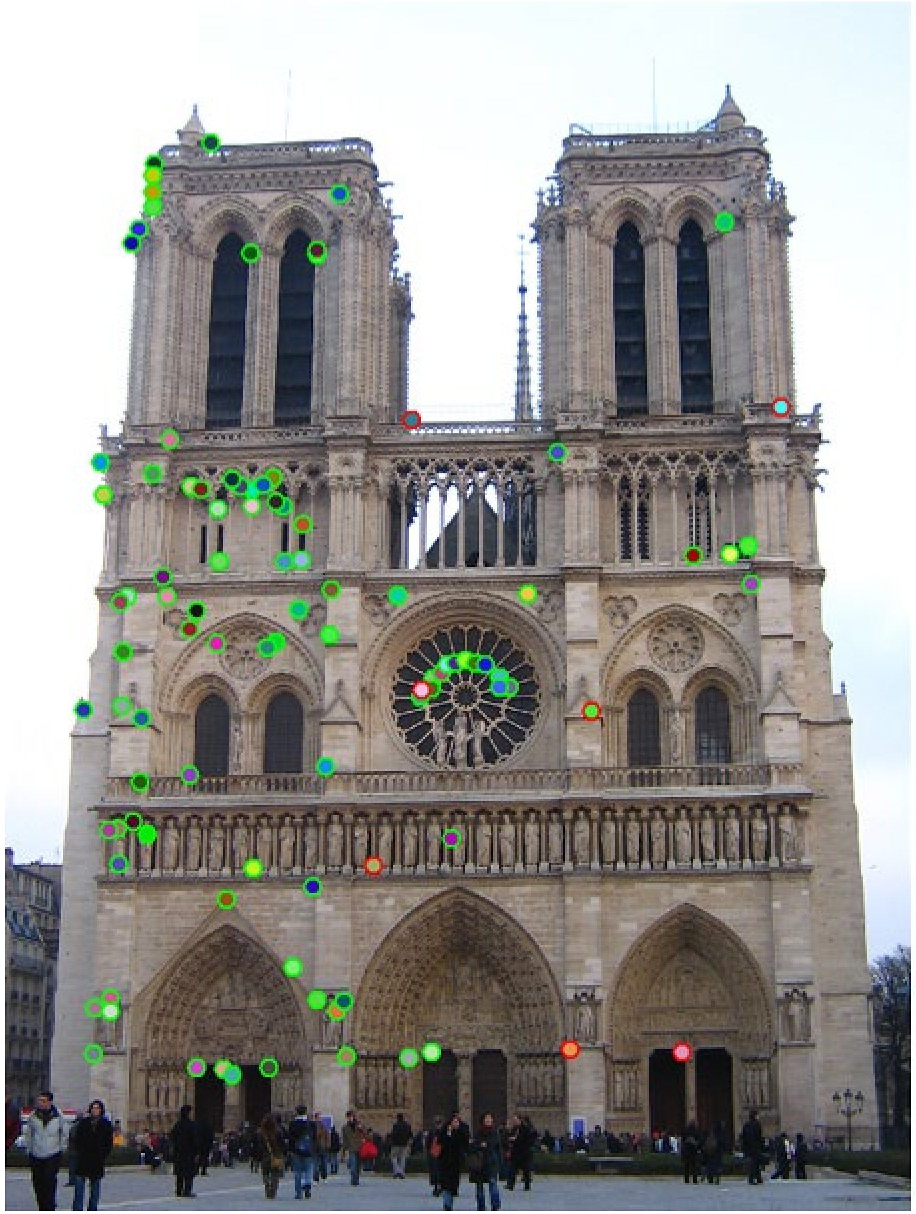
- Feature detection: repeatable and distinctive
  - Corners, blobs
  - Harris, DoG

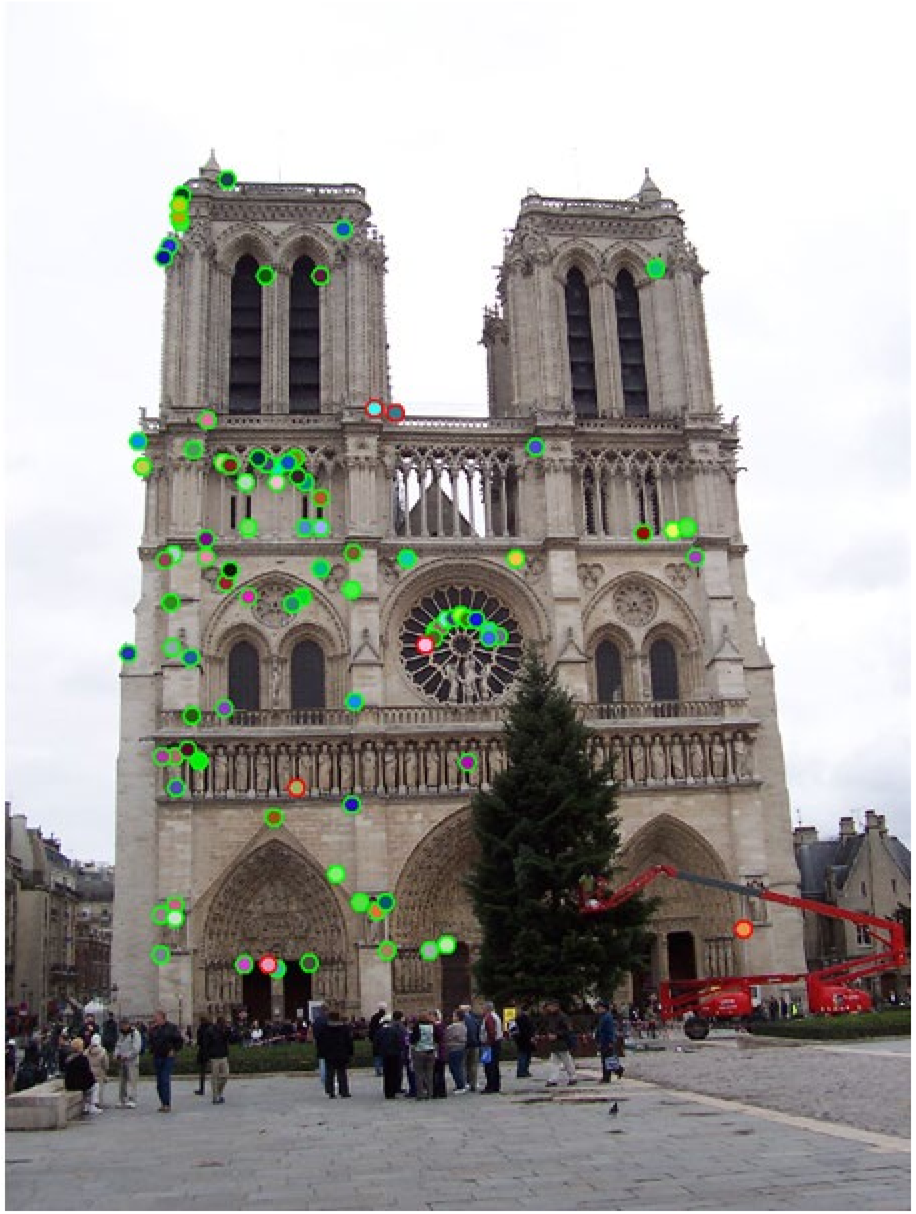


- Descriptors: robust and selective
  - spatial histograms of orientation
  - SIFT and variants are typically good for stitching and recognition
  - But, need not stick to one



Which features match?





Feature matching

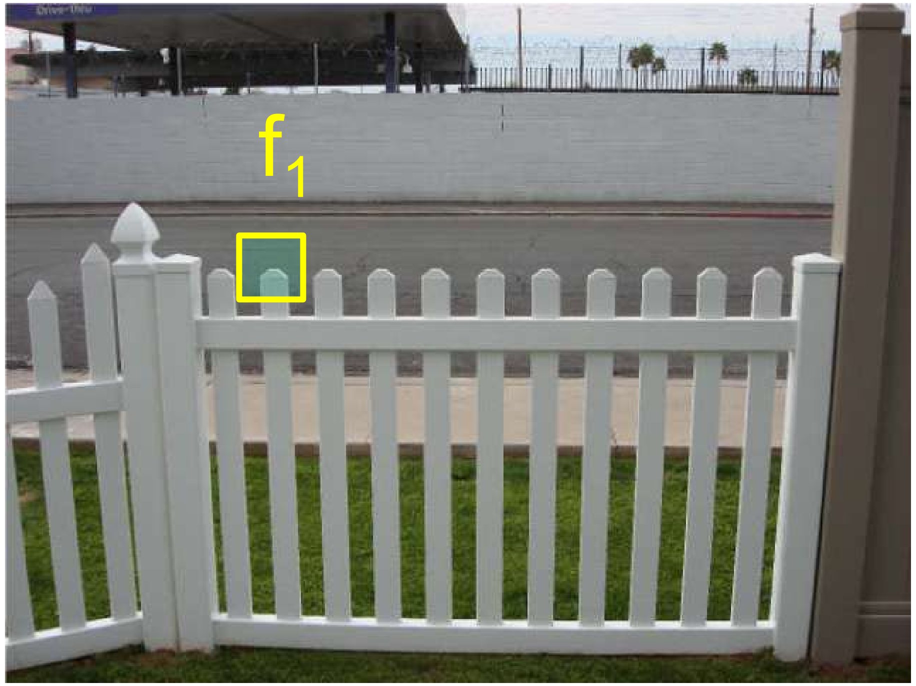
Given a feature in I_1, how to find the best match in I_2?

1. Define distance function that compares two descriptors
2. Test all the features in I_2, find the one with min distance

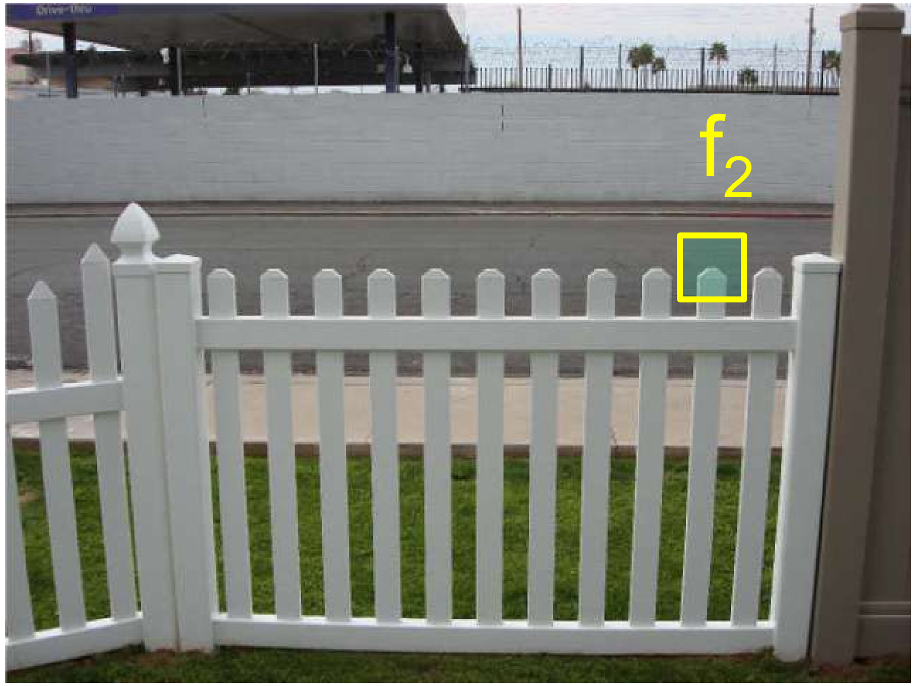
Feature distance

How to define the difference between two features f_1, f_2?

- Simple approach: L_2 distance, $|| f_1 - f_2 ||$
- can give small distances for ambiguous (incorrect) matches



I_1

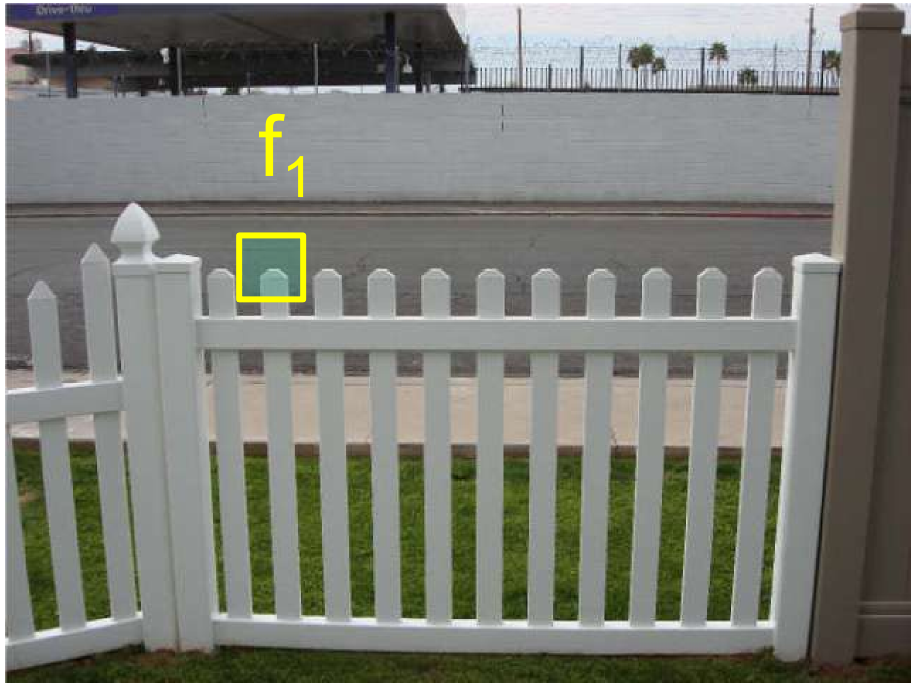


I_2

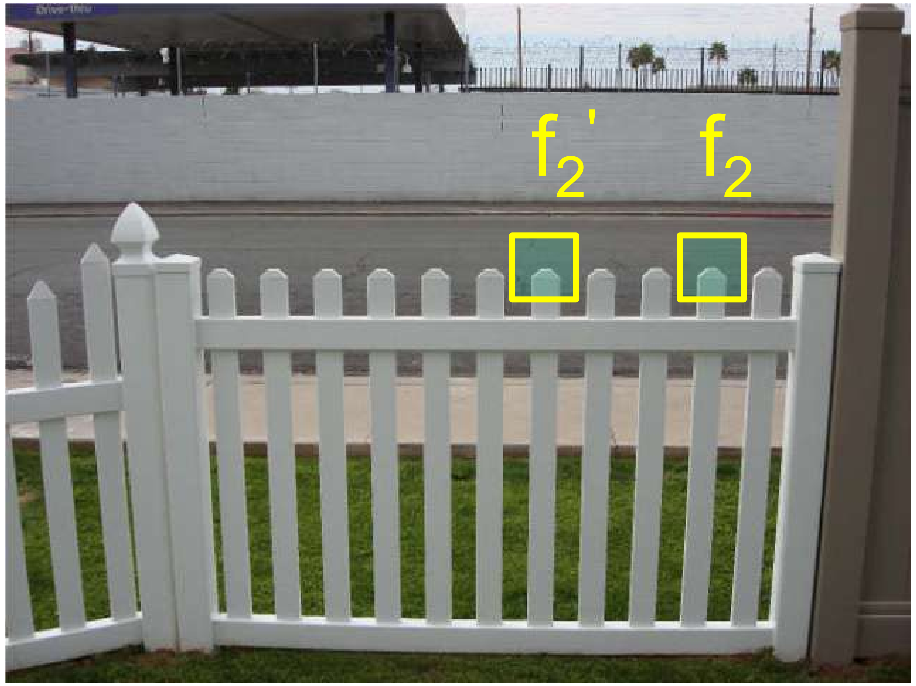
Feature distance

How to define the difference between two features f_1, f_2?

- Better approach: ratio distance = $|| f_1 - f_2 || / || f_1 - f_2' ||$
  - f_2 is the best SSD match to f_1 in I_2
  - f_2' is the 2nd best SSD match to f_1 in I_2
  - gives large values for ambiguous matches

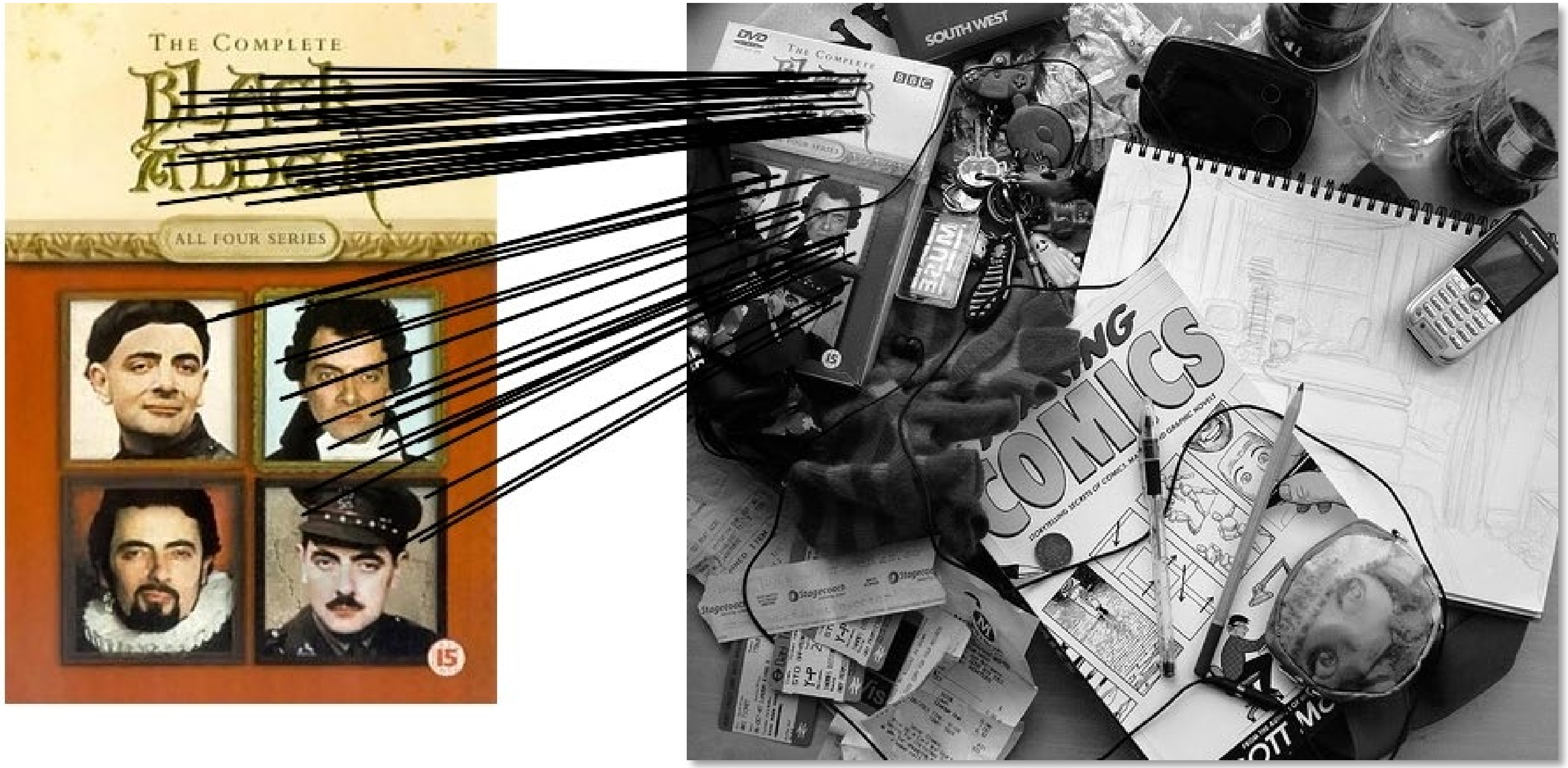


I_1



I_2

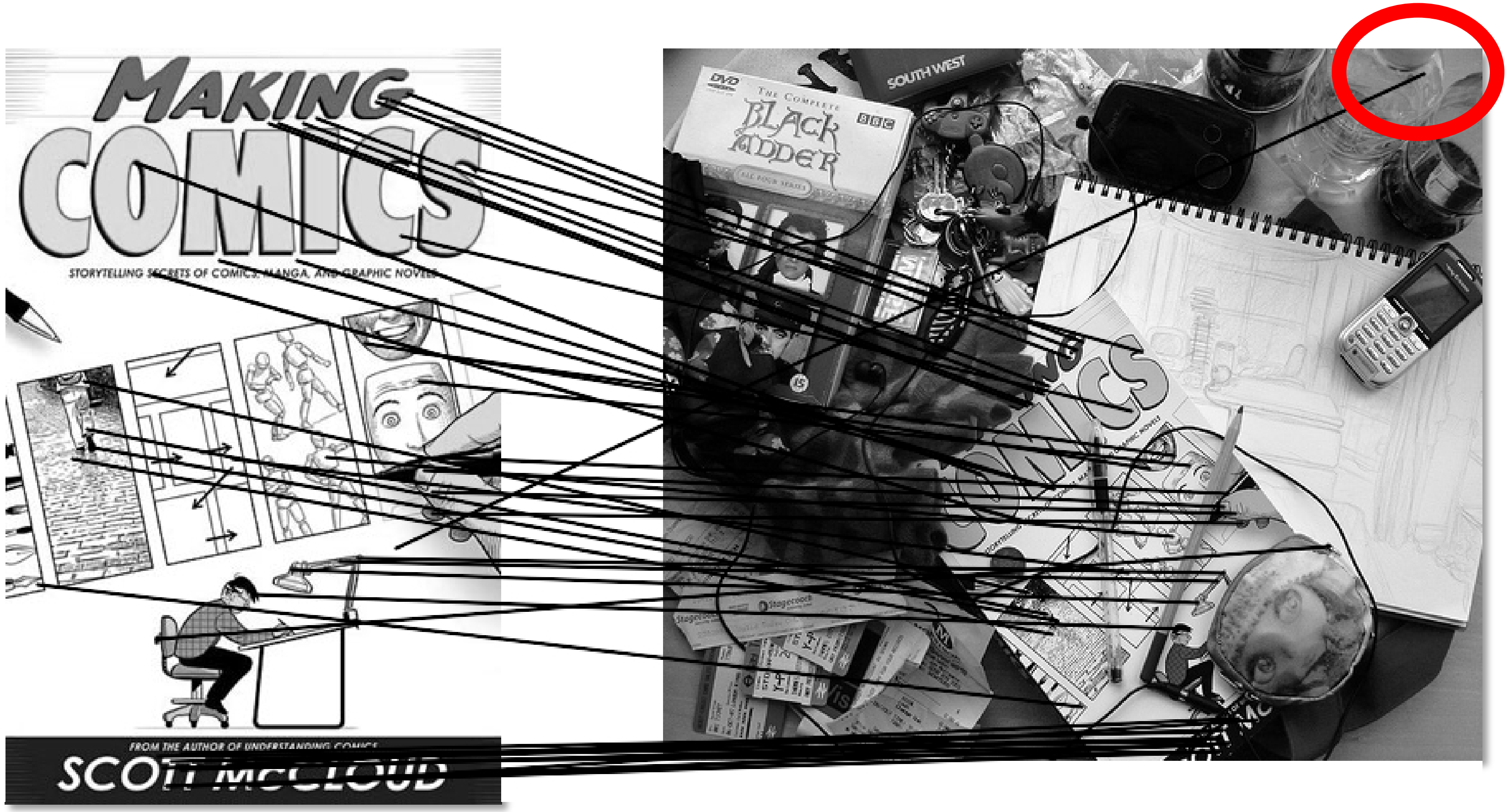
Feature matching example



58 matches (thresholded by ratio score)

Feature matching example

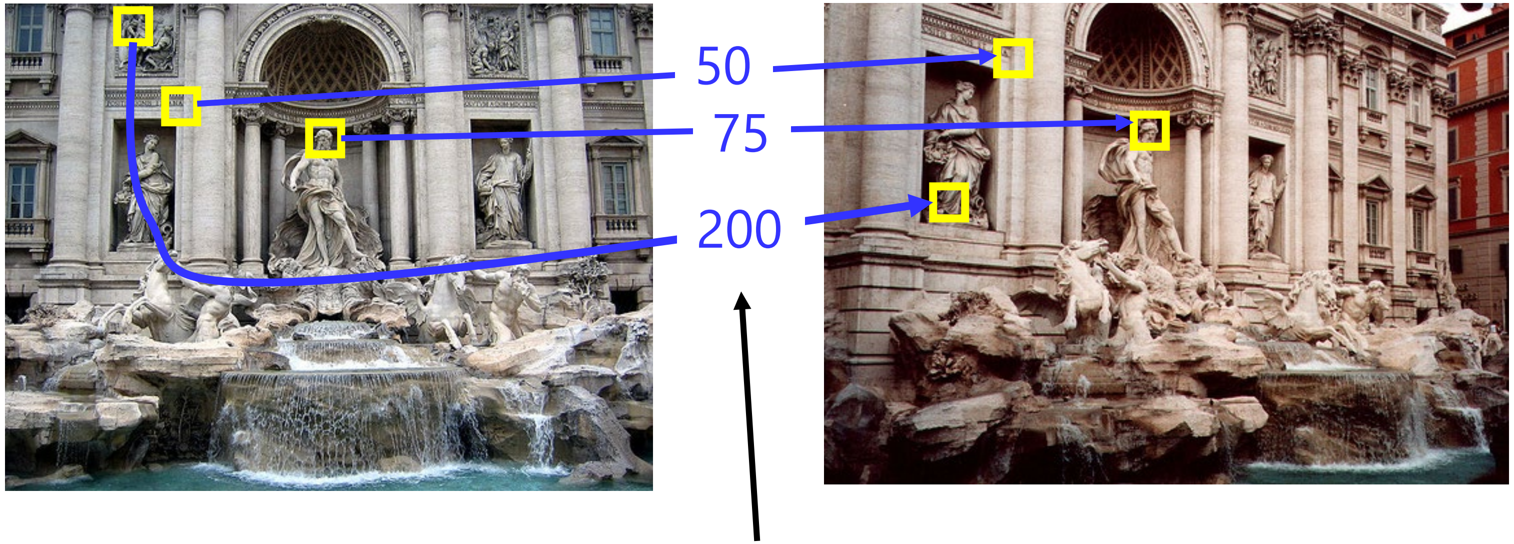
We'll deal with **outliers** later



51 matches (thresholded by ratio score)

Evaluating the results

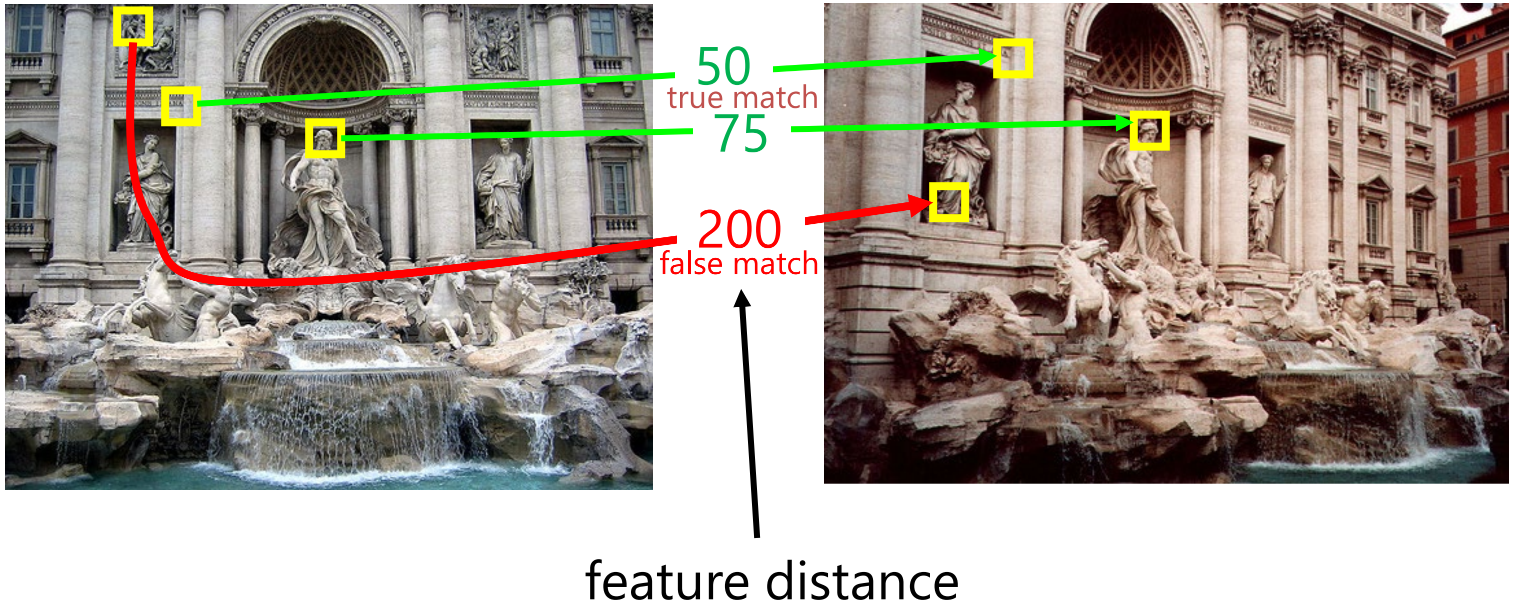
How can we measure the performance of a feature matcher?



feature distance

True/false positives

How can we measure the performance of a feature matcher?

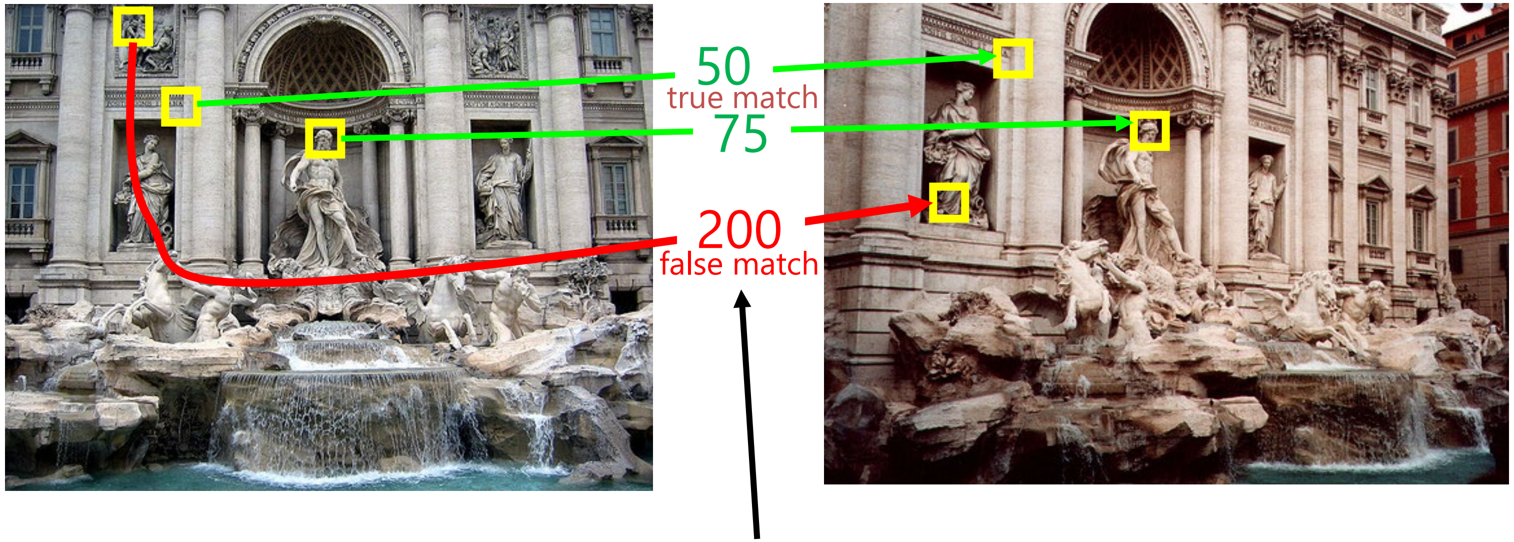


The distance threshold affects performance

– True positives = # of detected matches that survive the threshold that are correct
– False positives = # of detected matches that survive the threshold that are incorrect

True/false positives

How can we measure the performance of a feature matcher?



feature distance

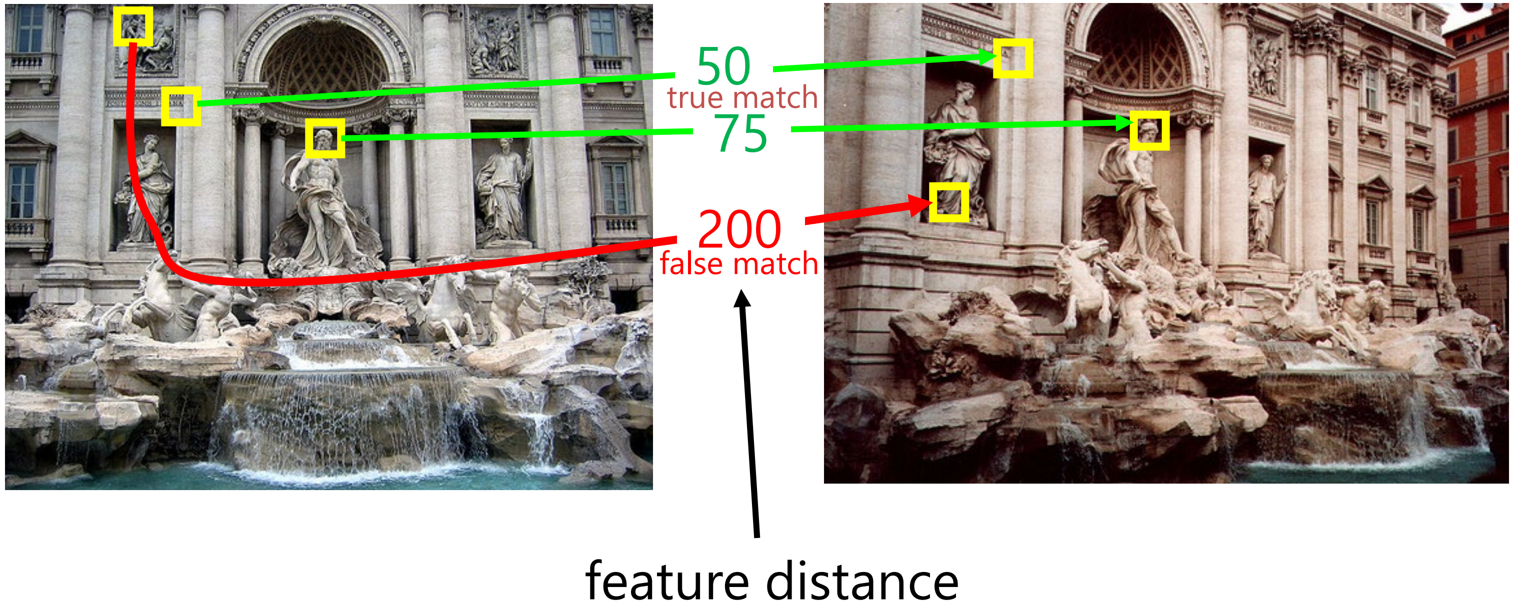
Suppose we want to **maximize true positives**.

How do we set the threshold?

(Note: we keep all matches with distance below the threshold.)

True/false positives

How can we measure the performance of a feature matcher?



Suppose we want to **minimize false positives**.

How do we set the threshold?

(Note: we keep all matches with distance below the threshold.)

Example

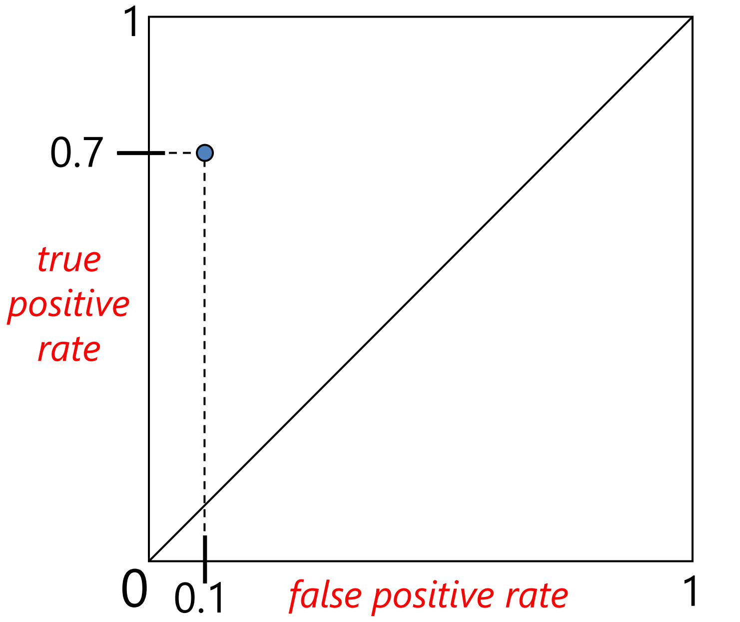
- Suppose our matcher computes 1,000 matches between two images
  - 800 are correct matches, 200 are incorrect (according to an oracle that gives us ground truth matches)
  - A given threshold (e.g., ratio distance = 0.6) gives us 600 correct matches and 100 incorrect matches that survive the threshold
  - True positive rate = 600 / 800 = ¾
  - False positive rate = 100 / 200 = ½

Evaluating the results

How can we measure the performance of a feature matcher?

$$\frac{\text{\# true positives surviving threshold}}{\text{\# total correct matches (positives)}}$$

recall



$$\frac{\text{\# false positives surviving threshold}}{\text{\# total incorrect matches (negatives)}}$$

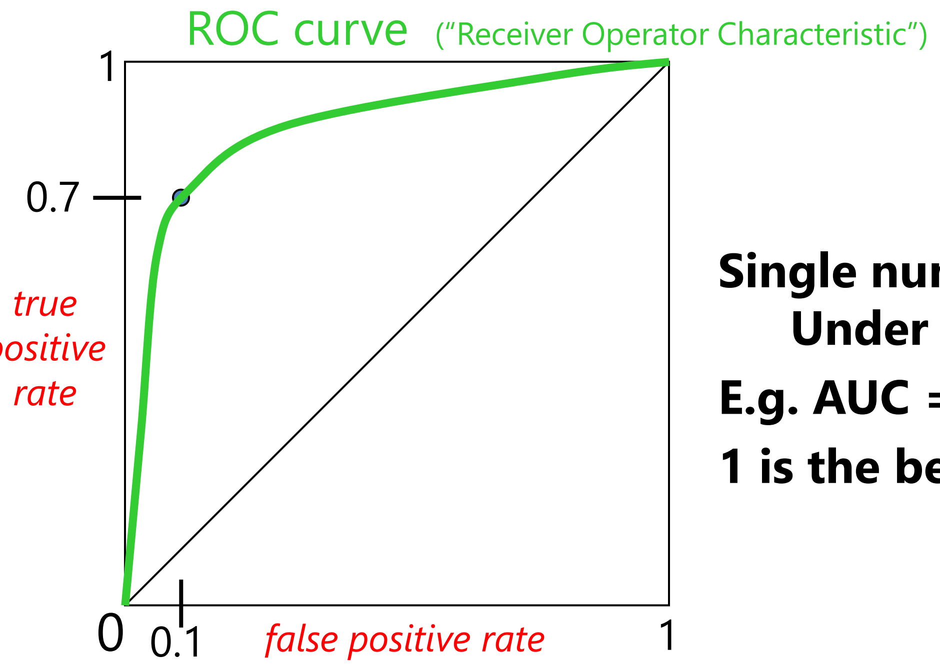
1 - *specificity*

Evaluating the results

How can we measure the performance of a feature matcher?

$$\text{true positive rate} = \frac{\text{\# true positives surviving threshold}}{\text{\# total correct matches (positives)}}$$

recall



$$\text{false positive rate} = \frac{\text{\# false positives surviving threshold}}{\text{\# total incorrect matches (negatives)}}$$

1 - *specificity*

Single number: Area Under the Curve (AUC)

E.g. AUC = 0.87

1 is the best

ROC curves – summary

- By thresholding the match distances at different thresholds, we can generate sets of matches with different true/false positive rates
- ROC curve is generated by computing rates at a set of threshold values swept through the full range of possible threshold
- Area under the ROC curve (AUC) summarizes the performance of a feature pipeline (higher AUC is better)

Lots of applications

Features are used for:

- Image alignment (e.g., mosaics)
- 3D reconstruction
- Motion tracking
- Object recognition
- Indexing and database retrieval
- Robot navigation
- … other

Feature Matching is Useful for …

Object instance recognition

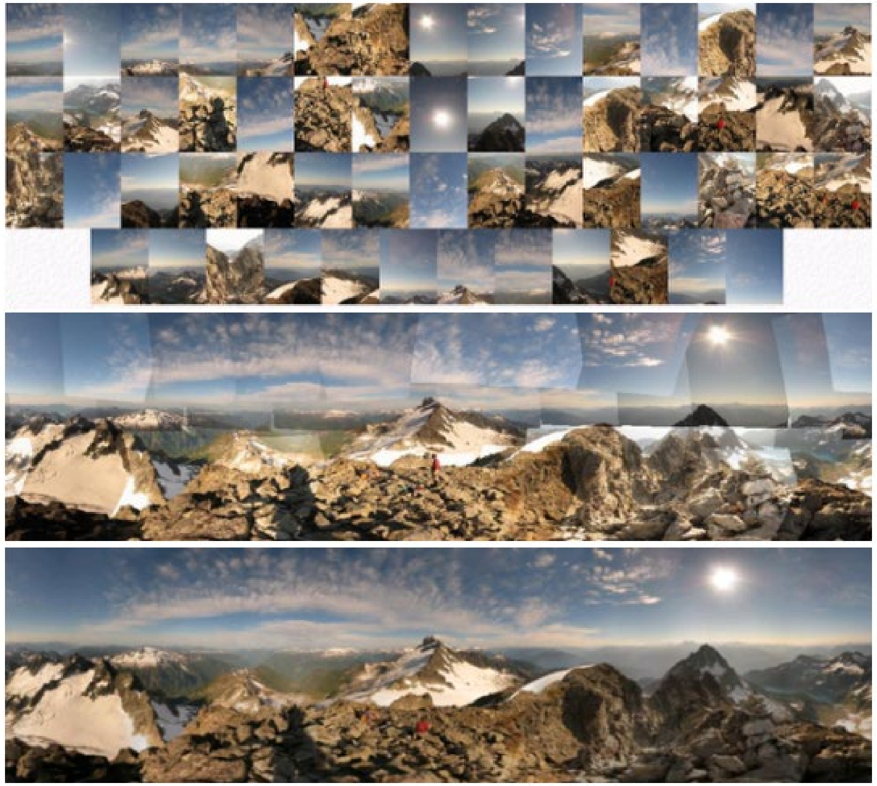
Schmid and Mohr 1997

Sivic and Zisserman, 2003

Rothganger et al. 2003

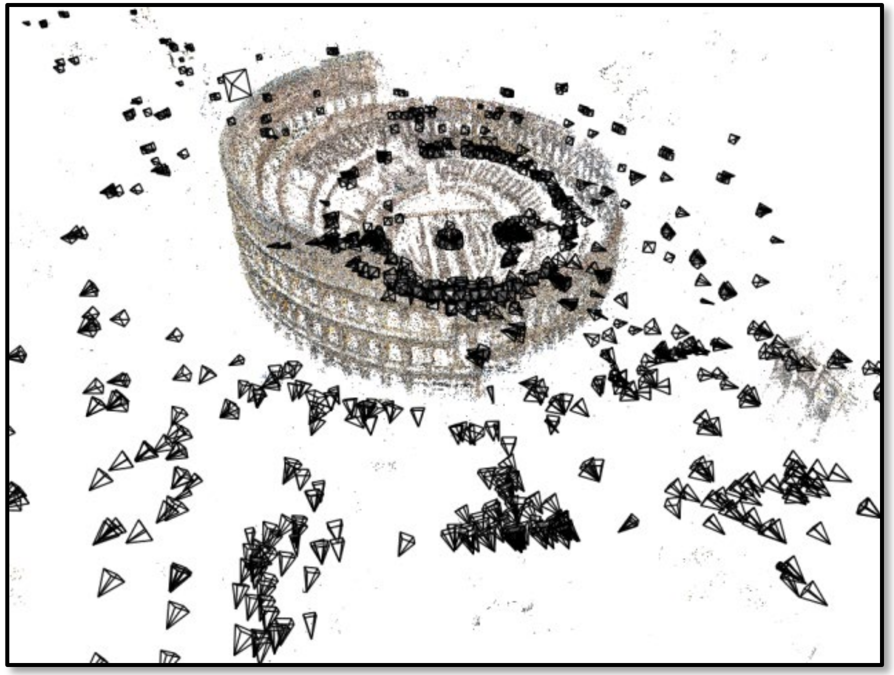
Lowe 2002

Image mosaicing



3D Reconstruction

Internet Photos ("Colosseum")



Reconstructed 3D cameras and points

Augmented Reality

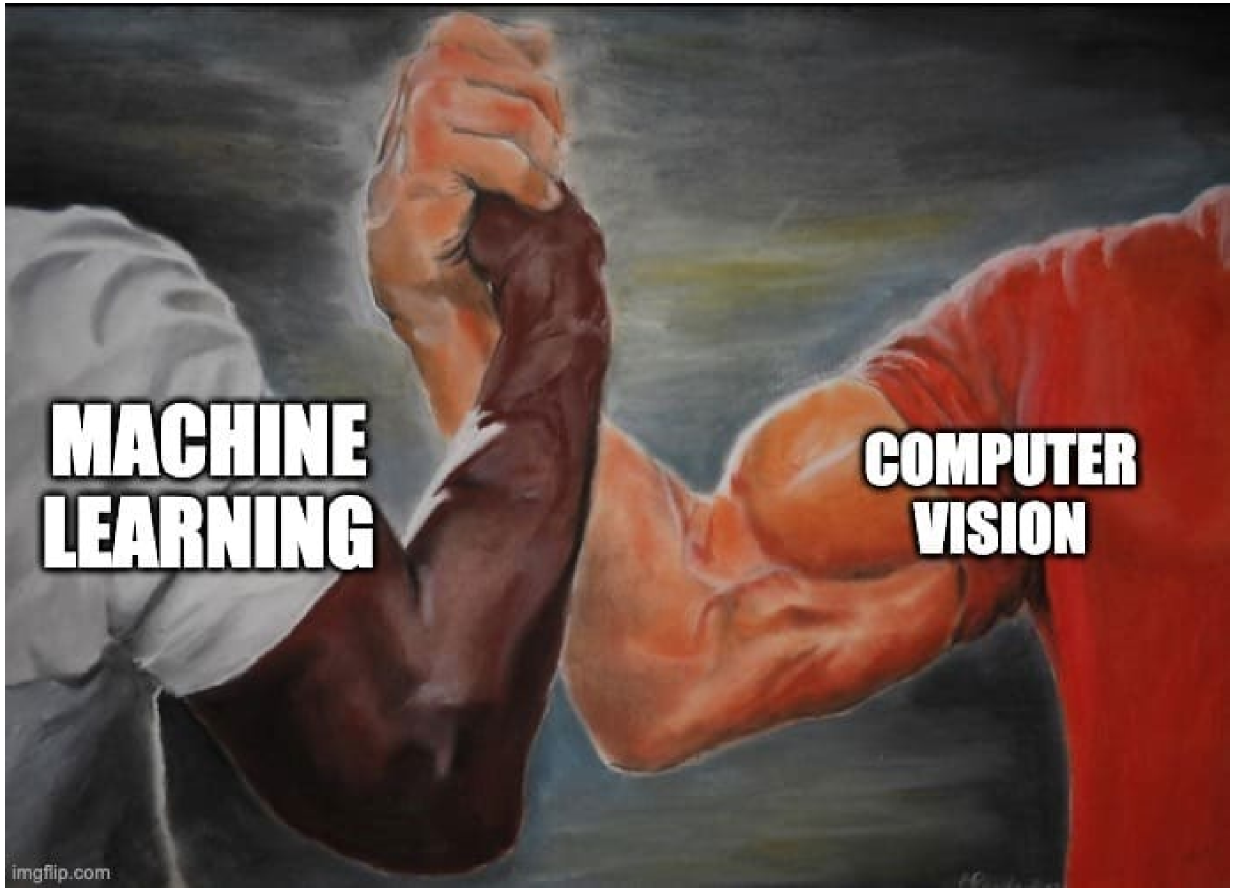
Now,

The Good Stuff
You’ve All Been Waiting For …

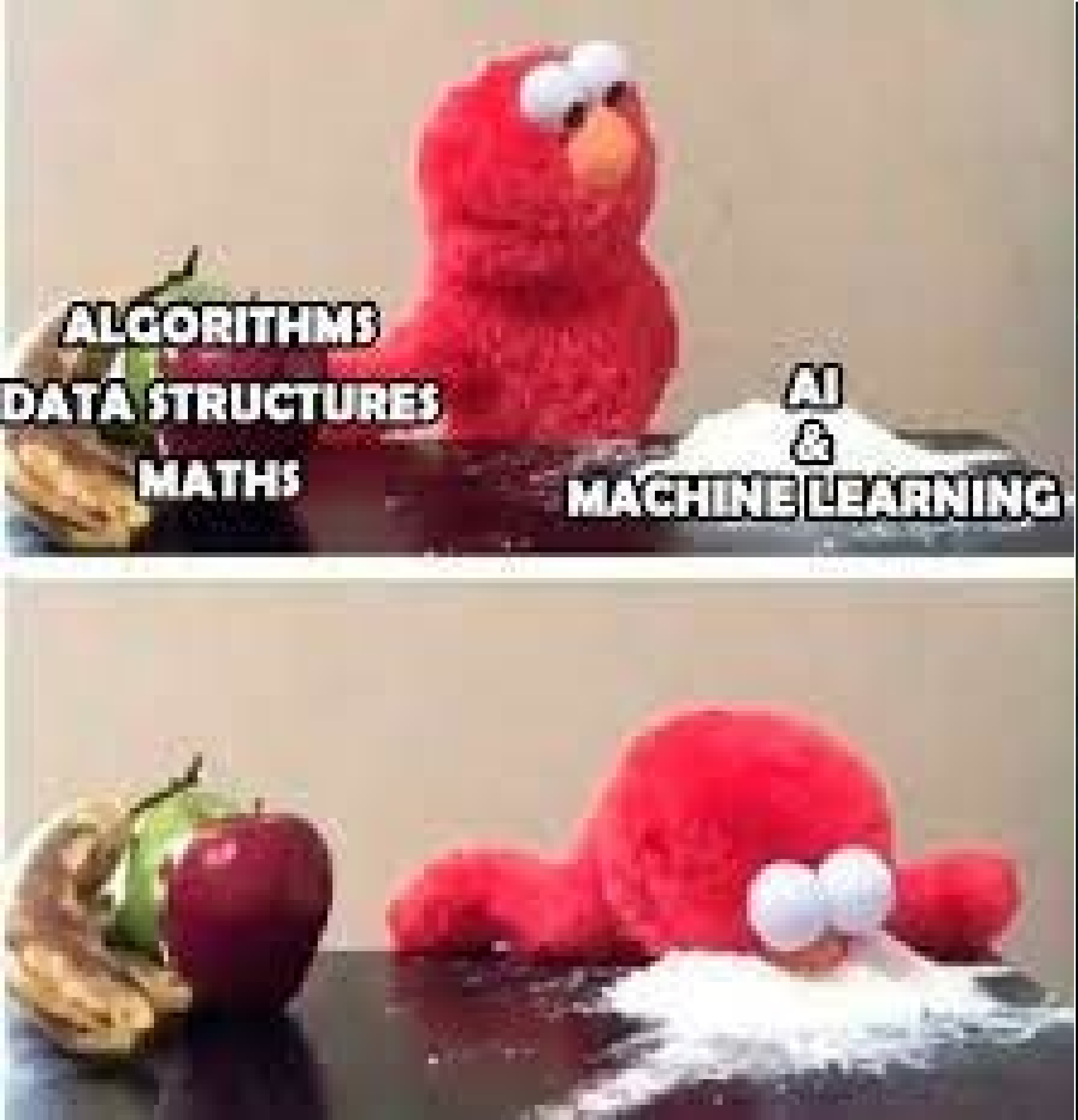
CMSC 491/691

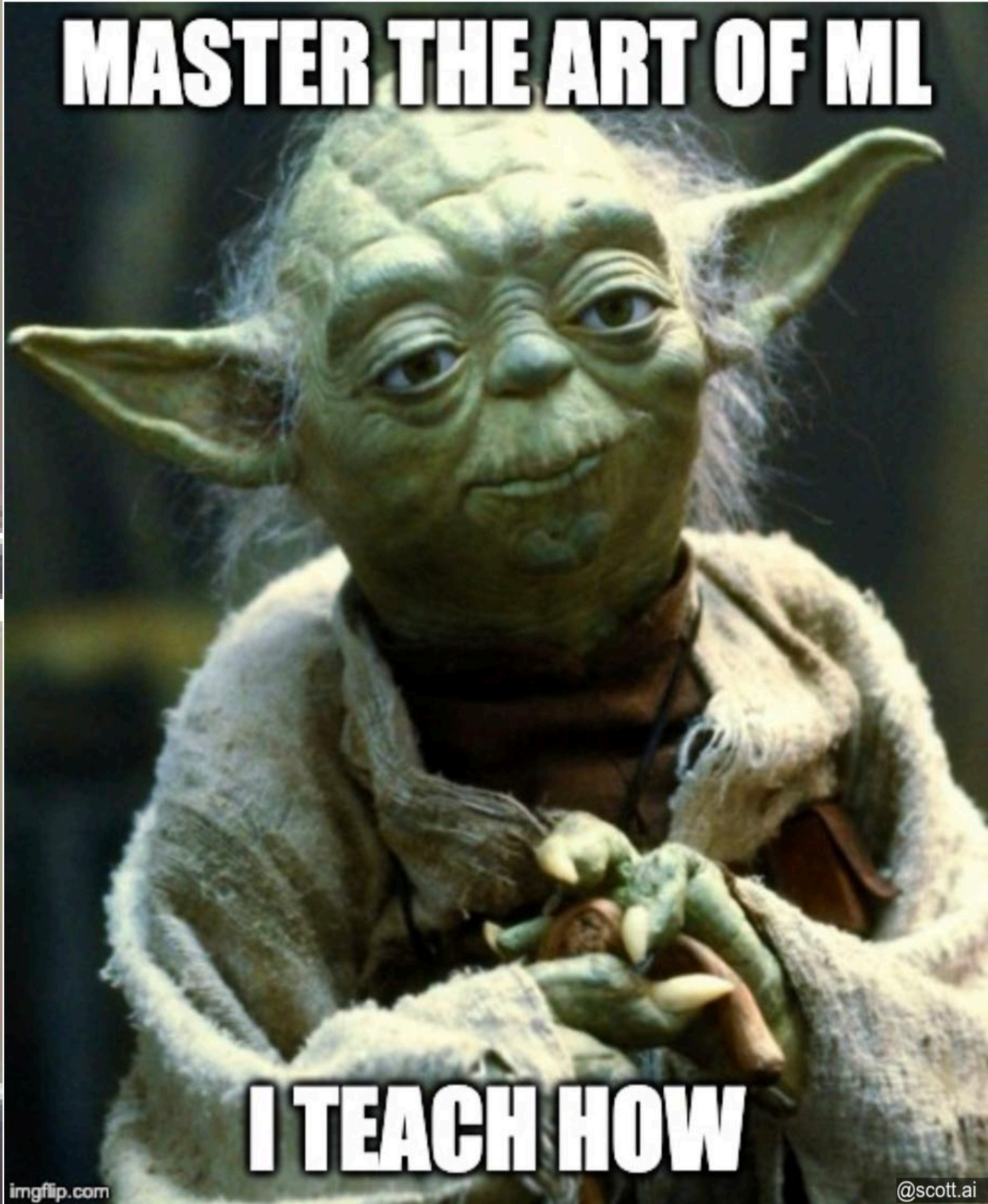
Lecture 7

Machine Learning
for
Computer Vision



Some slides from Isola


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

Motivation: 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, …}

Motivation: Image Classification

Central question: What "category" is this?

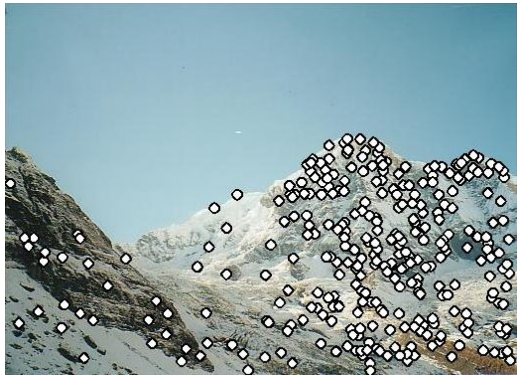
How can a computer vision system make a decision like this?

Ideas

- Based on colors, textures, shapes, edges, …
- Based on features !!!

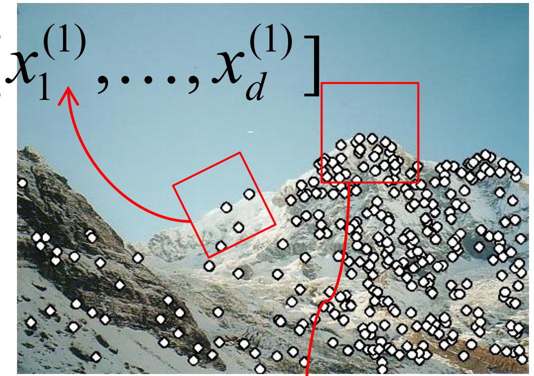
Features: main components

1) Detection: Identify the interest points



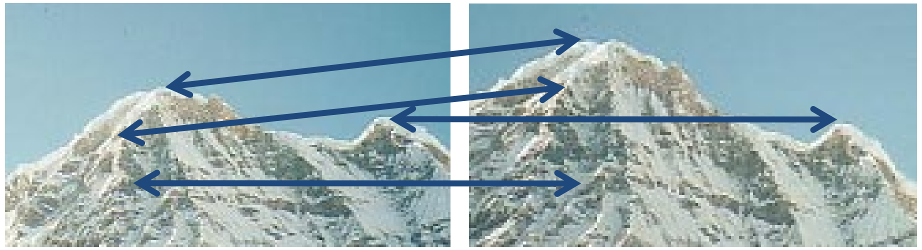
2) **Description: Extract vector feature descriptor surrounding each interest point.**

$$\mathbf{x}_1 = [x_1^{(1)}, \dots, x_d^{(1)}]$$



$$\mathbf{x}_2 = [x_1^{(2)}, \dots, x_d^{(2)}]$$

3) Matching: Determine correspondence between descriptors in two views

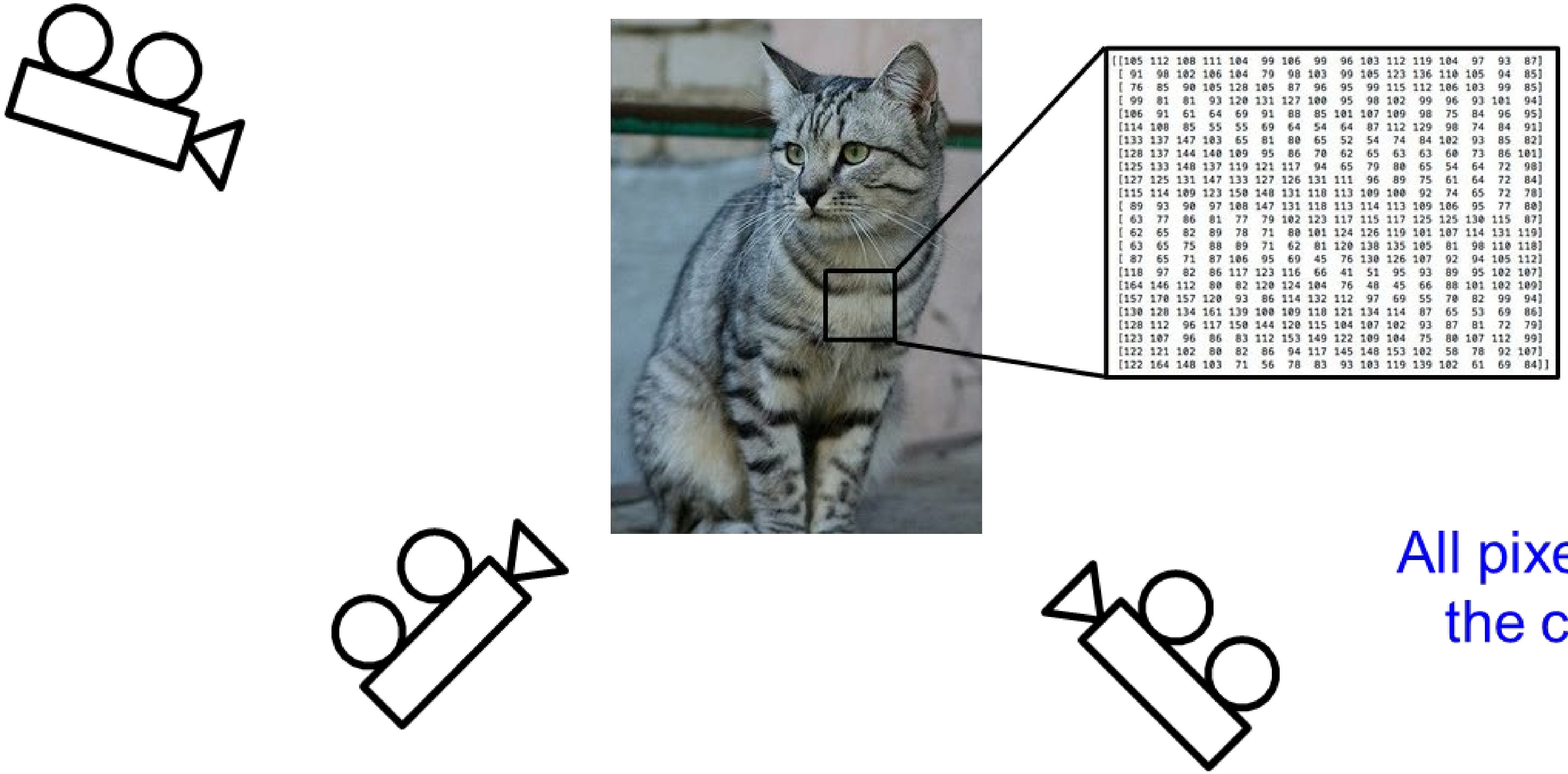


Kristen Grauman

Recap & Motivation

- Image features are “interesting”, “unique” regions in an image
  - Intuitively these are “important”
- So far – we have seen how to detect and describe (*a.k.a. “represent”)* certain types of feature
  - Harris Corners, Blobs, …
- We had a definition for what a “feature” is
  - Can we learn that from data?

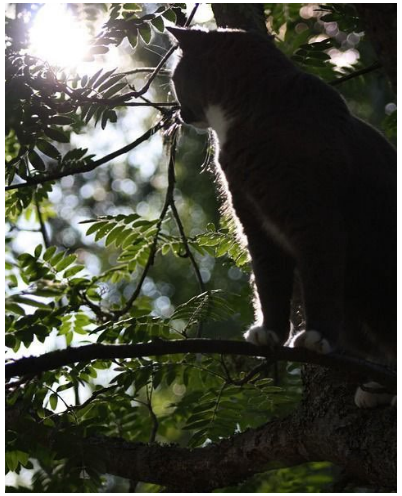
Challenges: Viewpoint Variation



All pixels change when
the camera moves!

Slide Credit: Fei-Fei Li

Challenges: Illumination

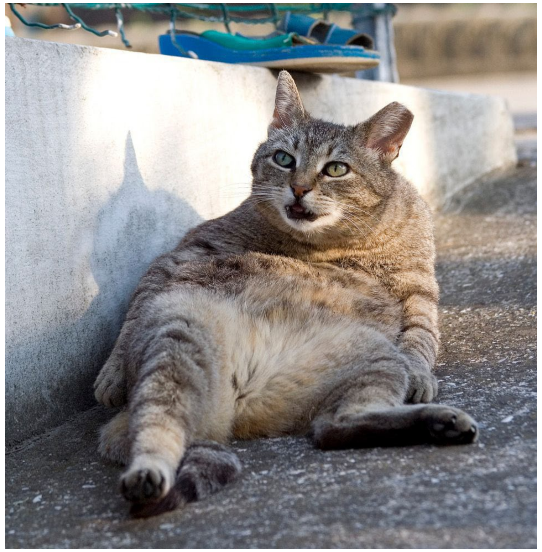
Challenges: Background Clutter

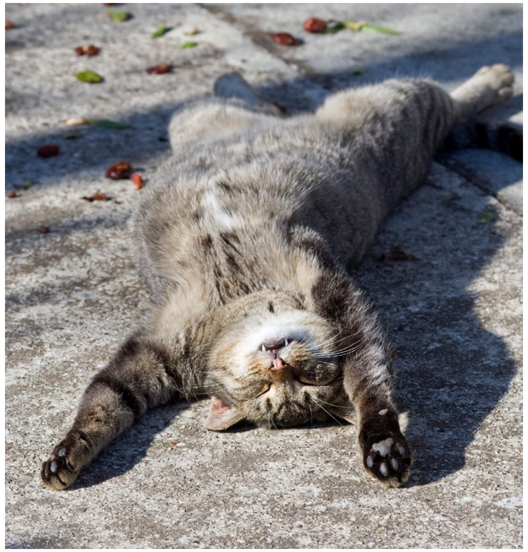
Slide Credit: Fei-Fei Li

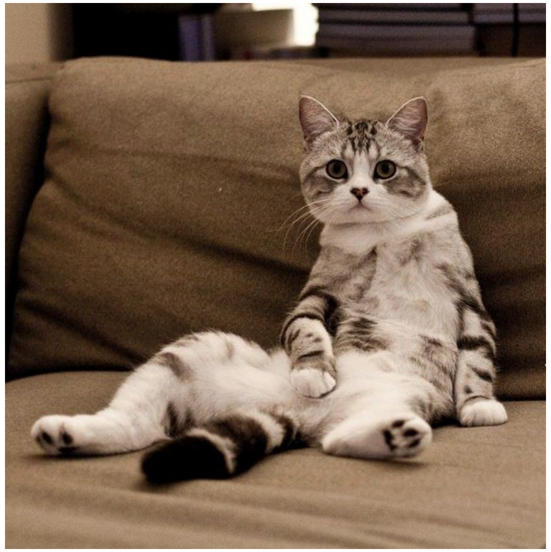
Challenges: Occlusion



Challenges: Pose and Deformation (Cat Yoga)



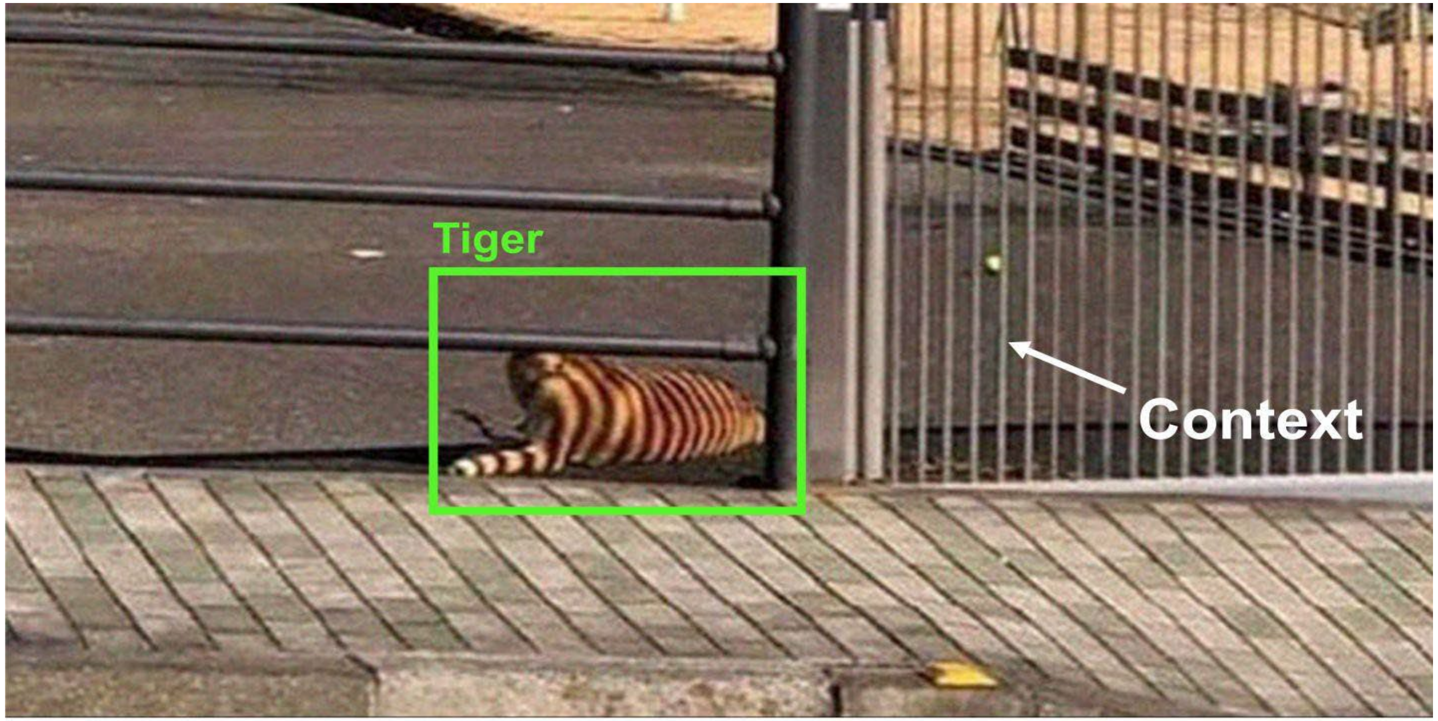




Challenges: Inter-Class Variation

Slide Credit: Fei-Fei Li

Challenges: Illusions



Slide Credit: Fei-Fei Li

Data !!!

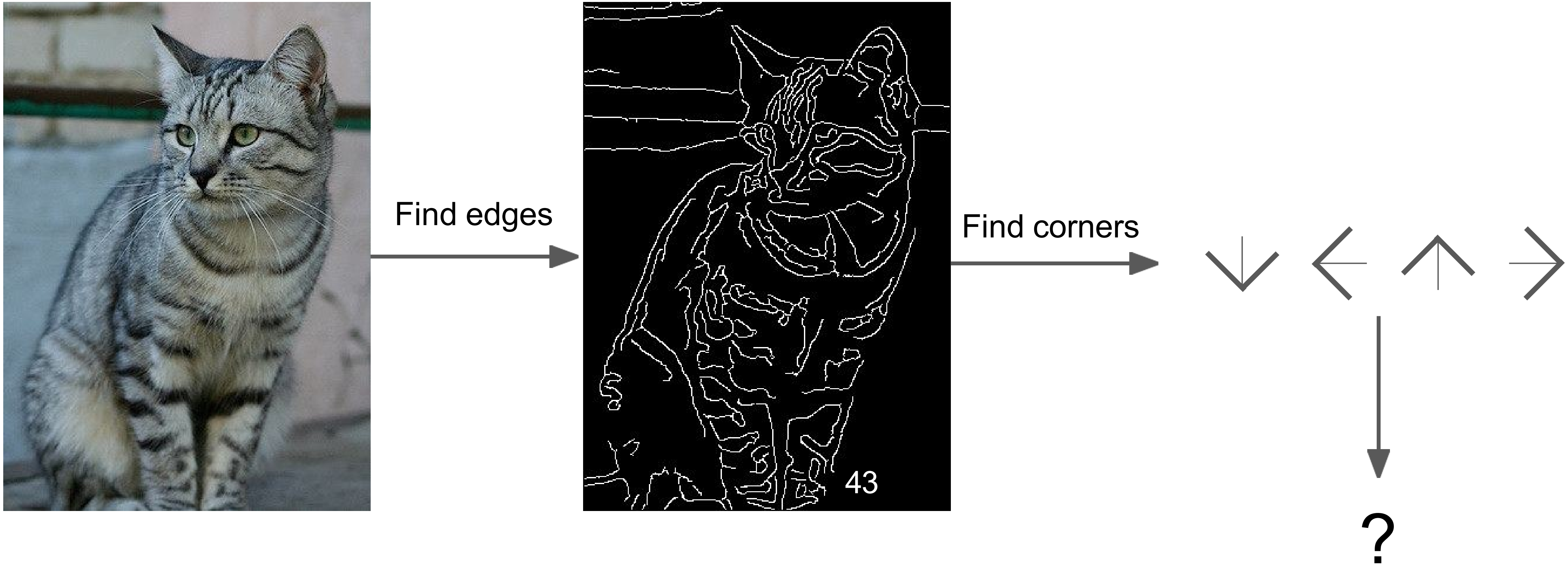
An image classifier

```
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.

Can we use features to make the decision?



John Canny, “A Computational Approach to Edge Detection”, IEEE TPAMI 1986

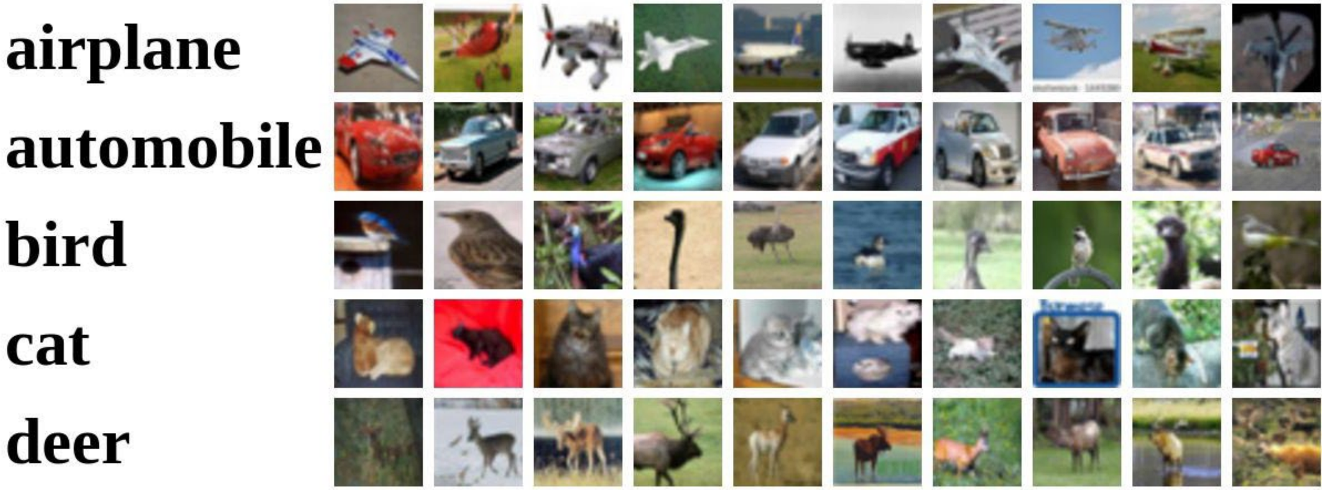
Machine Learning

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 $|$  ,  $|$ $\rightarrow \mathbb{R}$

Distance Metric to compare images

L1 distance: $$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

test image

56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

-

training image

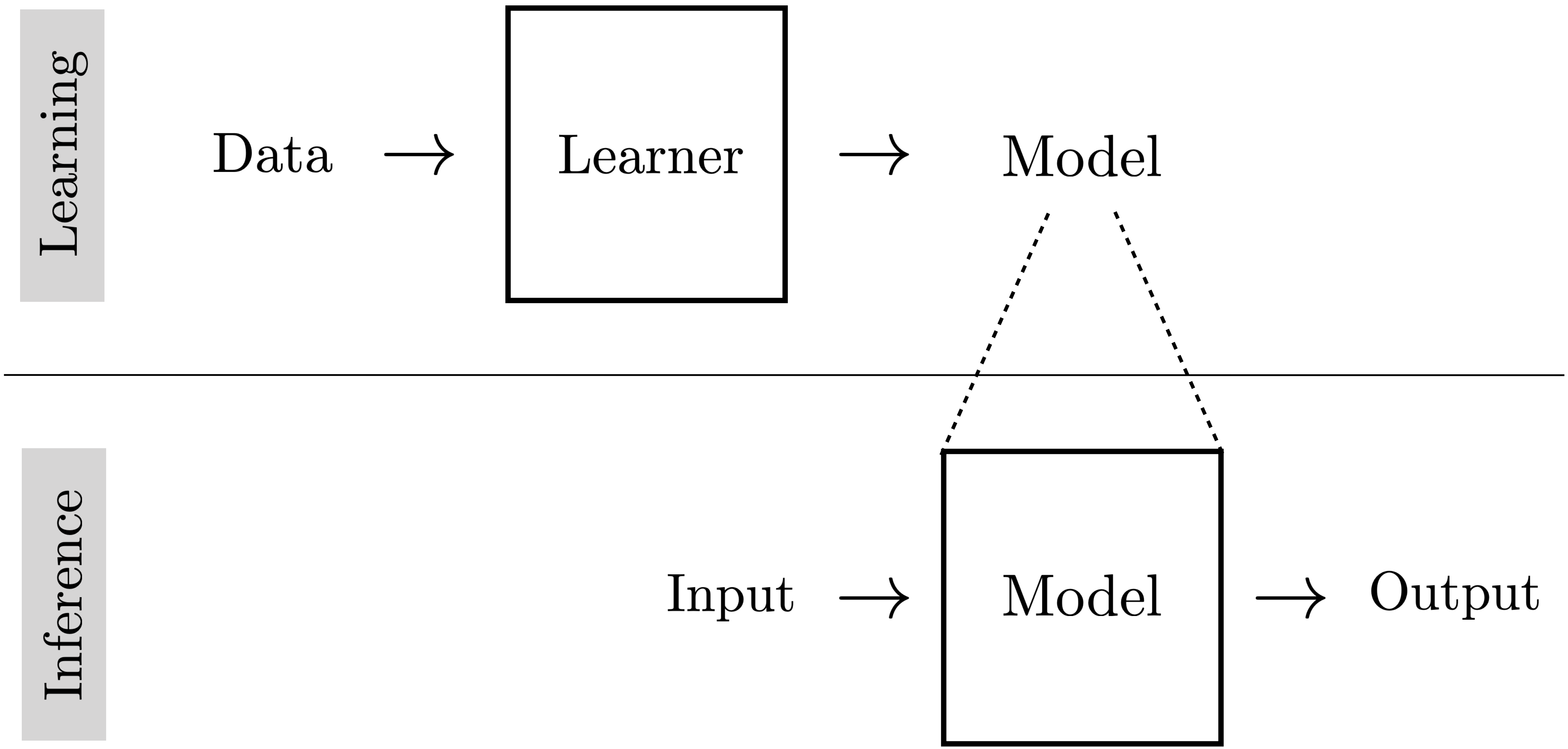
10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

=

pixel-wise absolute value differences

46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

$\rightarrow$ 456


Learning
Data
Learner
Model
Inference
Input
Model
Output

Learning

Da

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

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

→ Output

Inference

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

What does ☆ do?

$2 ☆ 3 = 36$

$7 ☆ 1 = 49$

$5 ☆ 2 = 100$

$2 ☆ 2 = 16$

Goal: answer future queries involving ☆

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

Past experience

2 ☆ 3 = 36

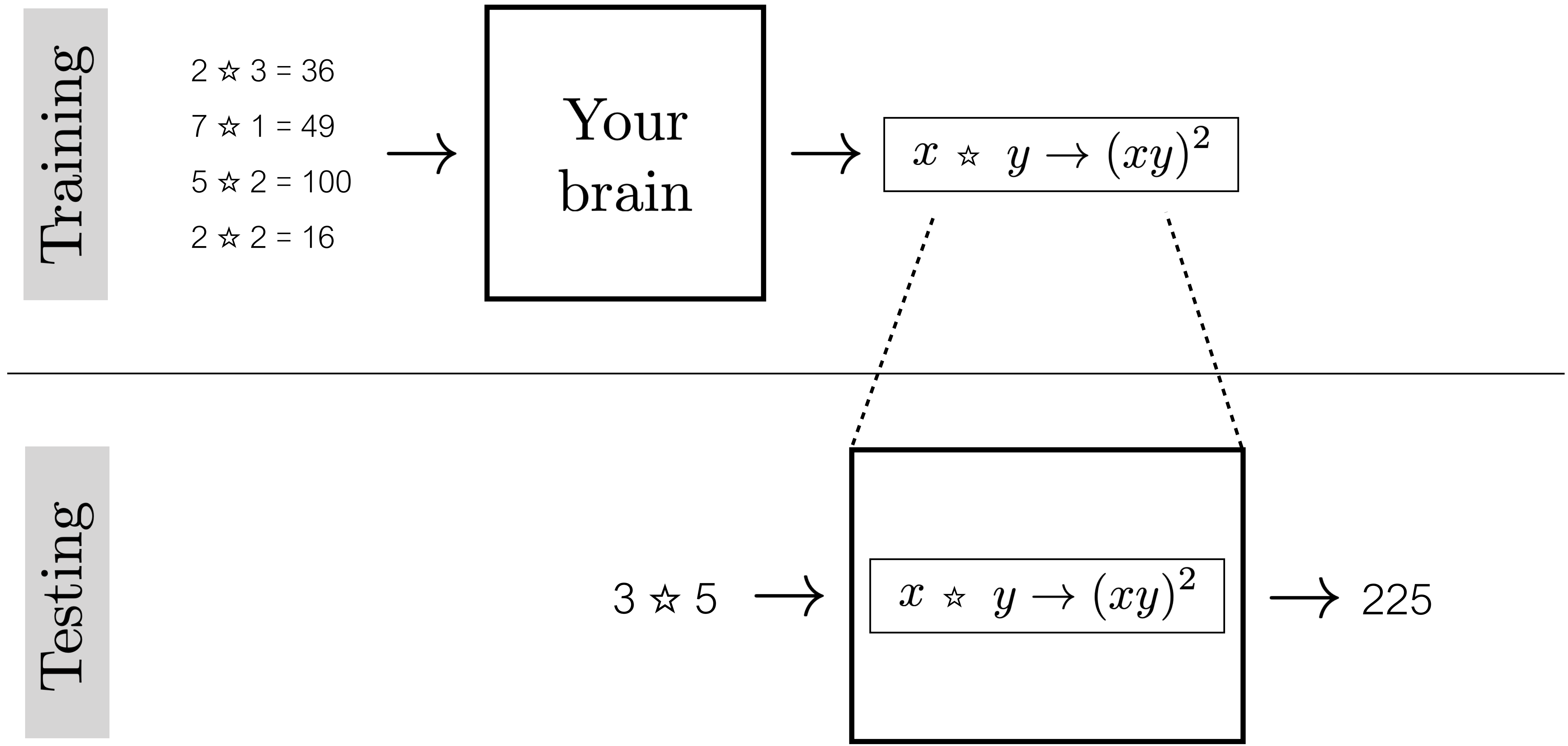
7 ☆ 1 = 49

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

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

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

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

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

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

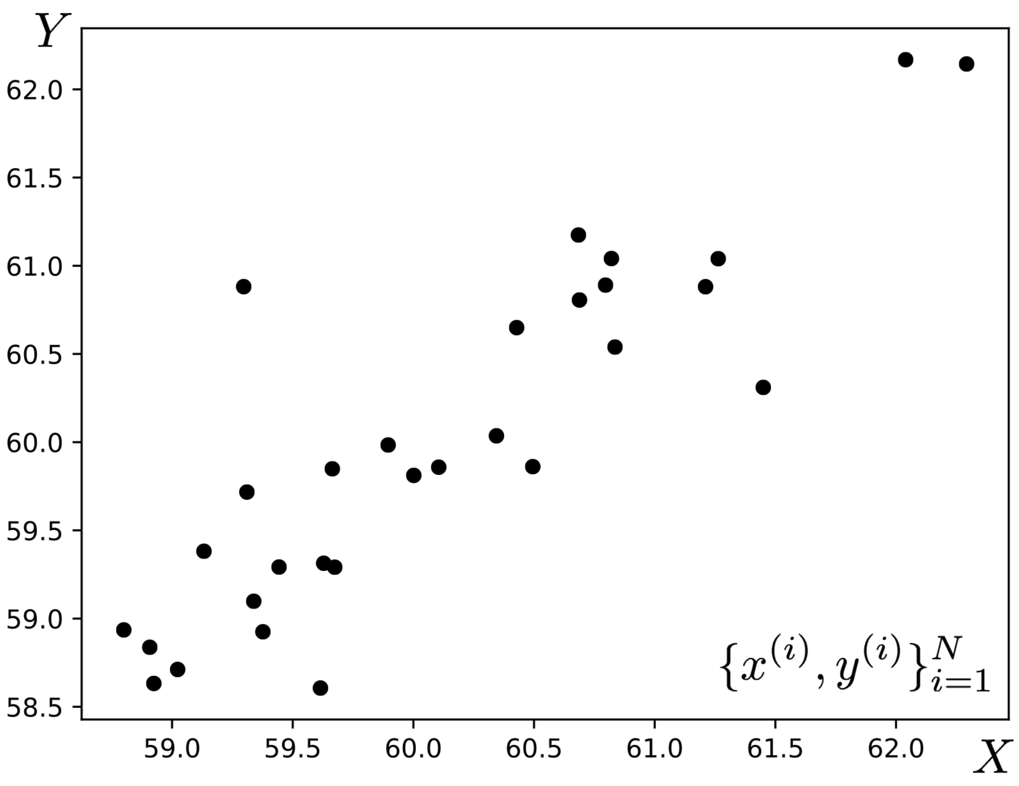
Learning from examples

(aka **supervised learning**)

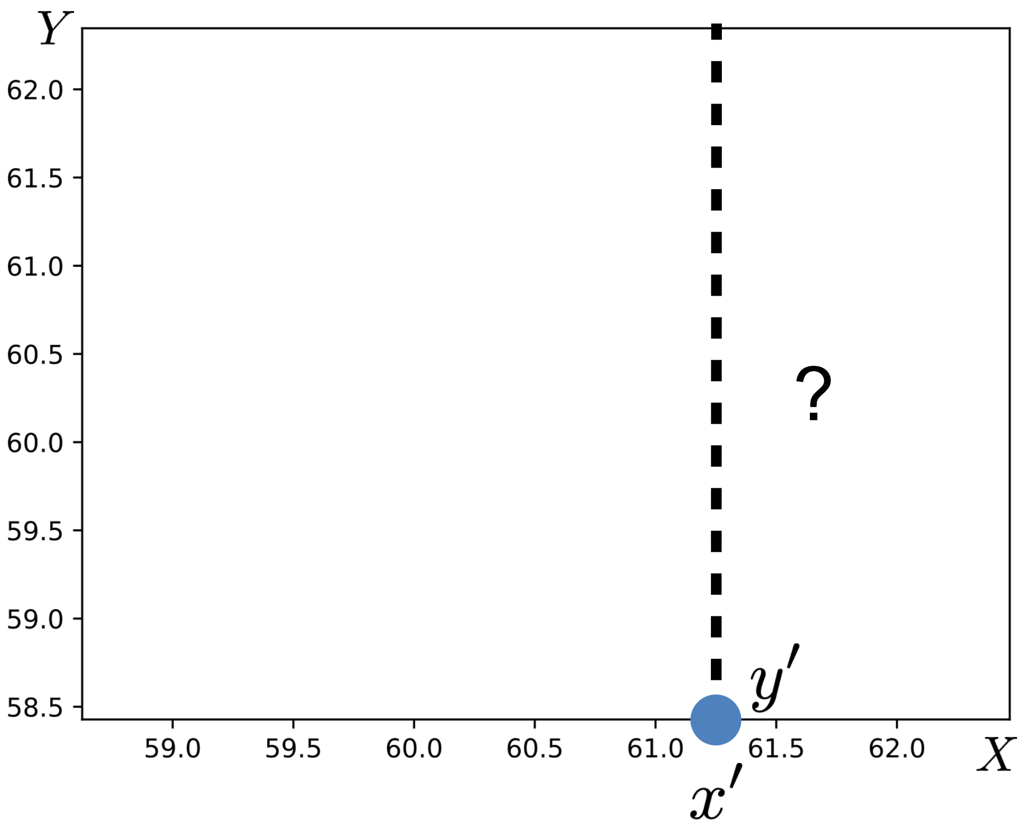
Training data

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

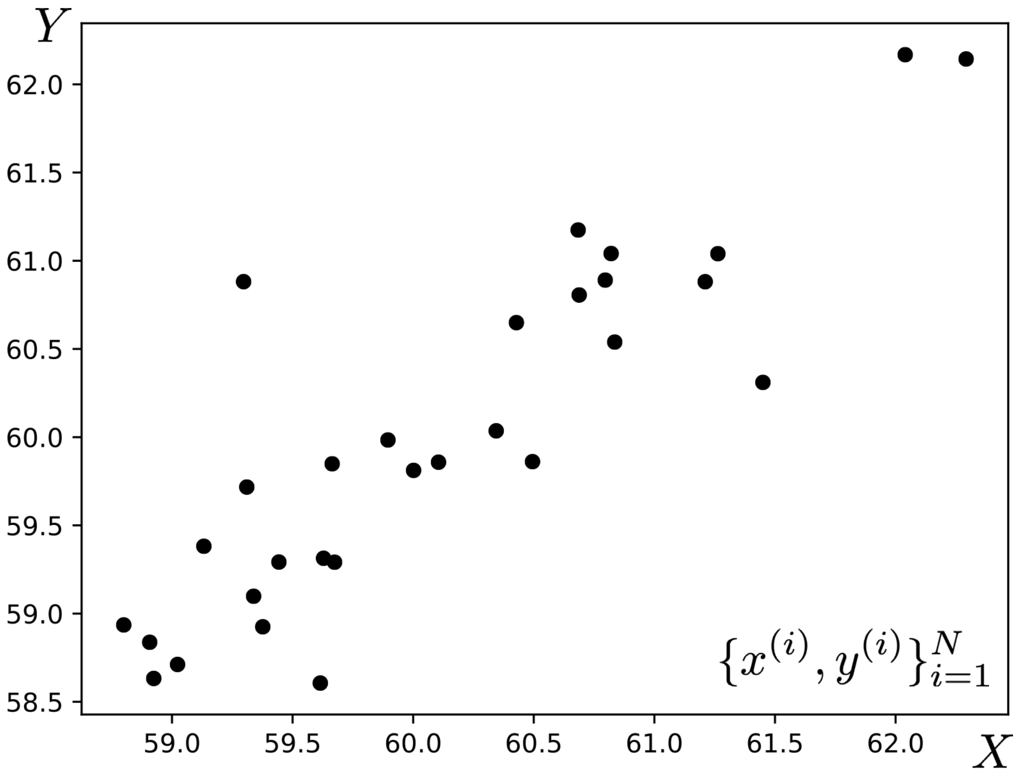
Training data



Test query



Training data

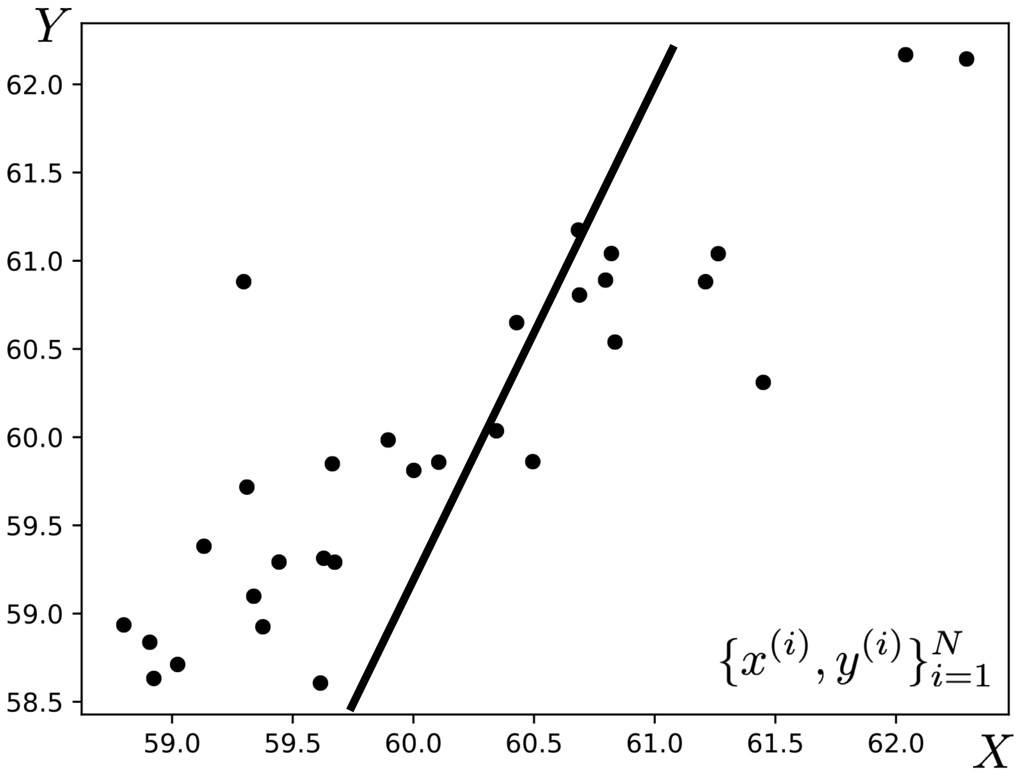


$\rightarrow$ Learner $\rightarrow$ $f_\theta(x) = \theta_1 x + \theta_0$

Hypothesis space

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

Training data

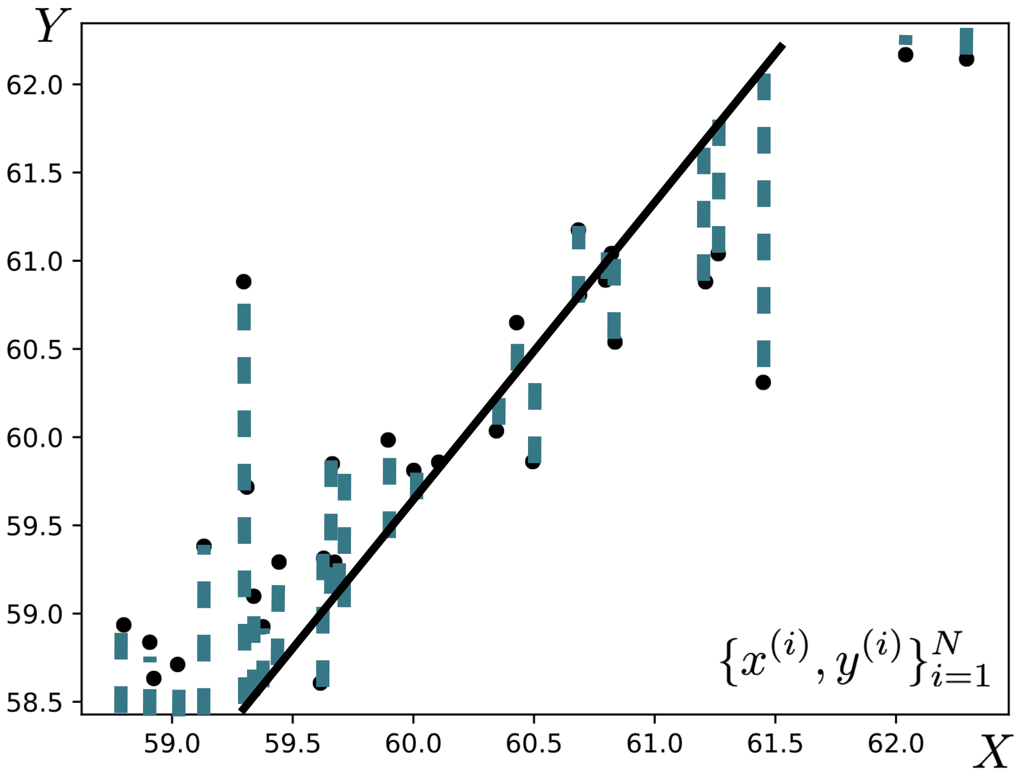


Search for the **parameters**, $\theta = \{\theta_0, \theta_1\}$, that best fit the data.

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

Best fit in what sense?

Training data



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

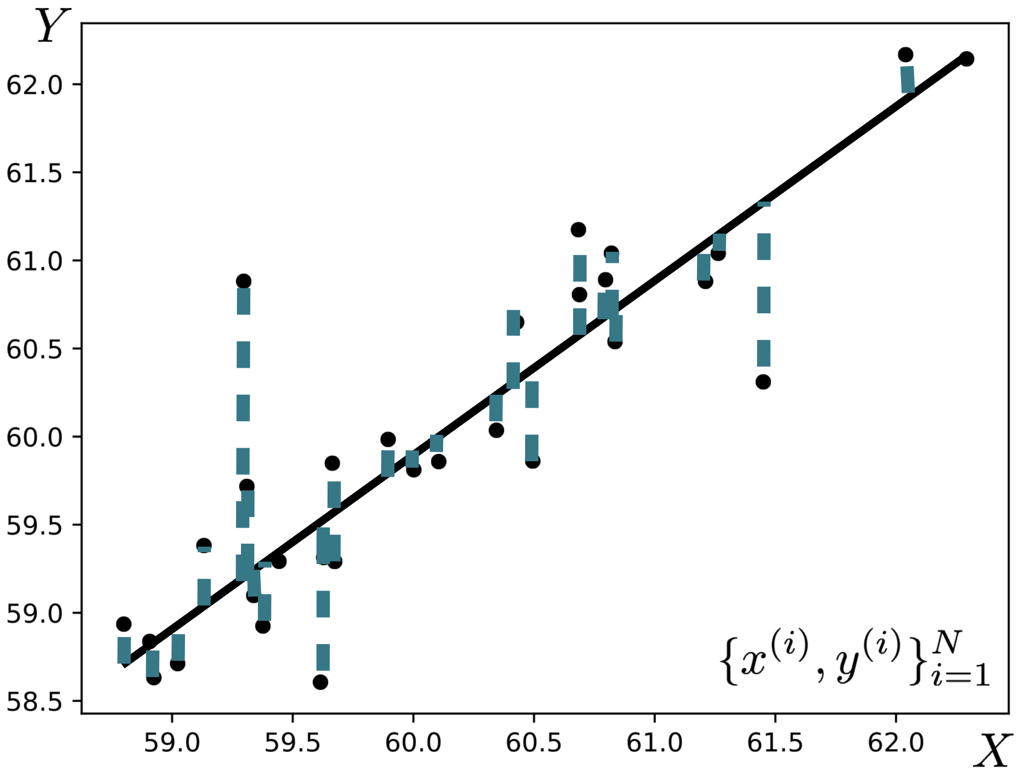
$$f_\theta(x) = \theta_1 x + \theta_0$$

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.

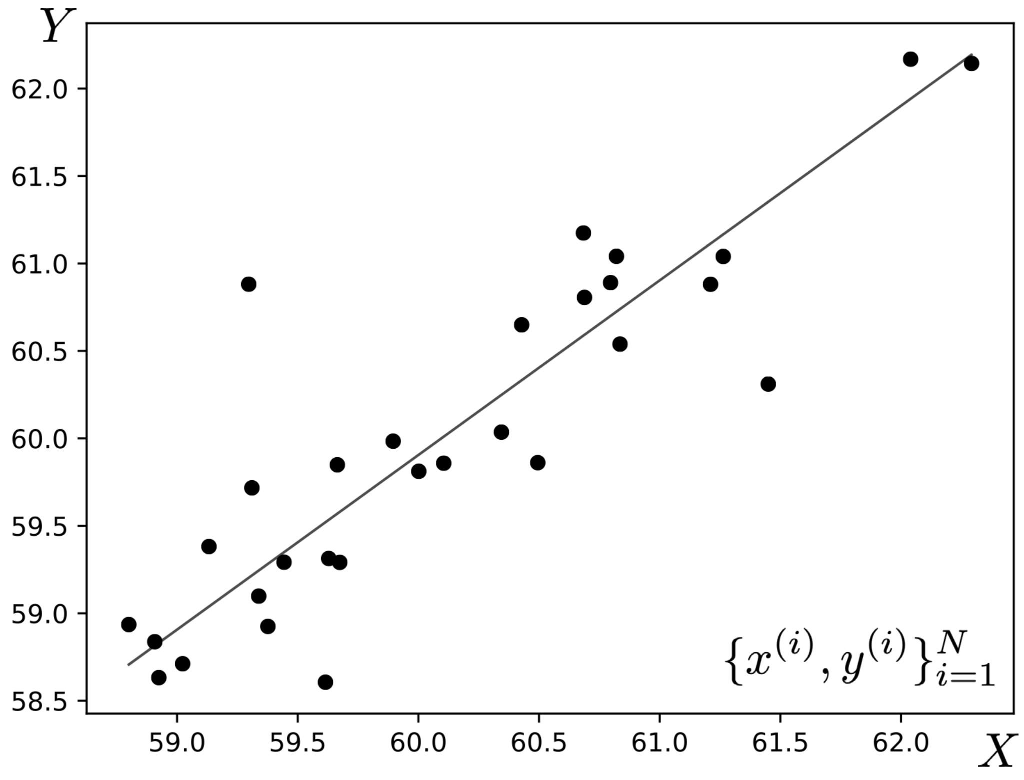
$$f_\theta(x) = \theta_1 x + \theta_0$$

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

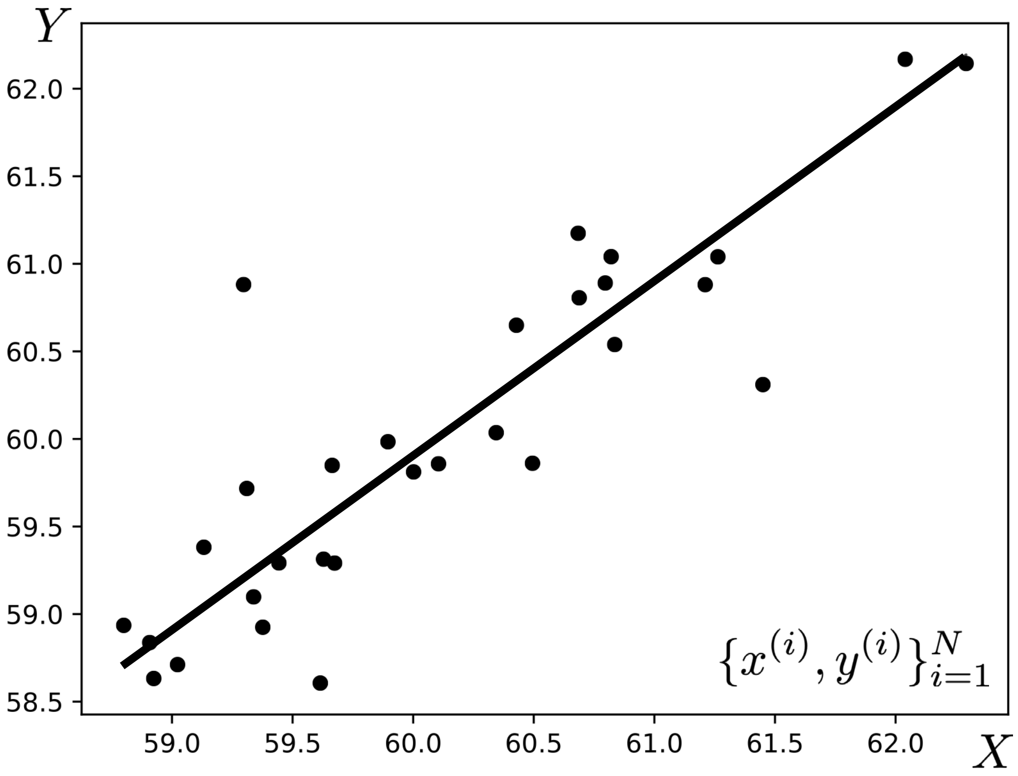


Complete learning problem:

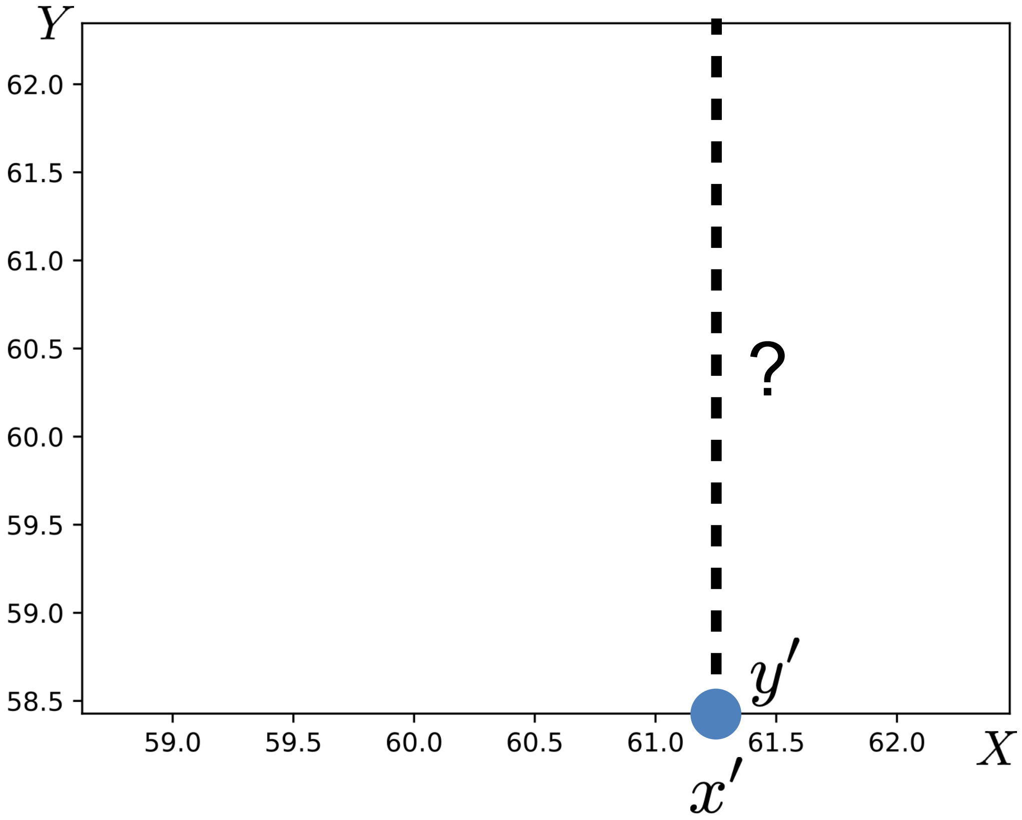
$$\theta^* = \arg\min_{\theta} \sum_{i=1}^{N} (f_\theta(x^{(i)}) - y^{(i)})^2$$

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

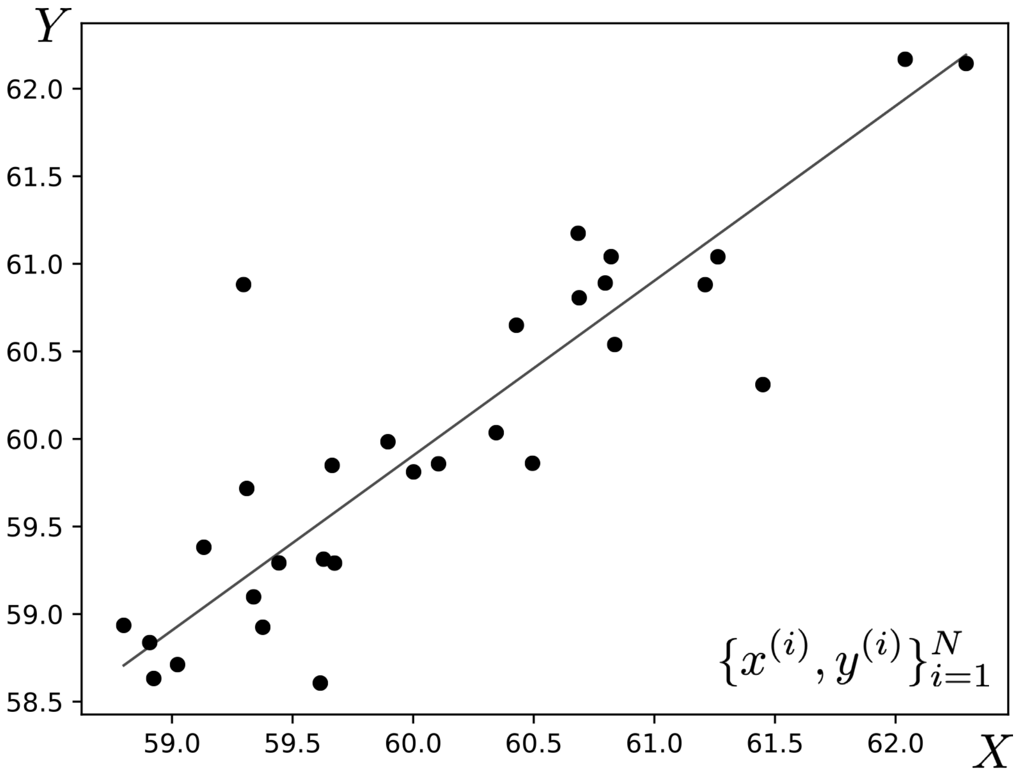
Training data



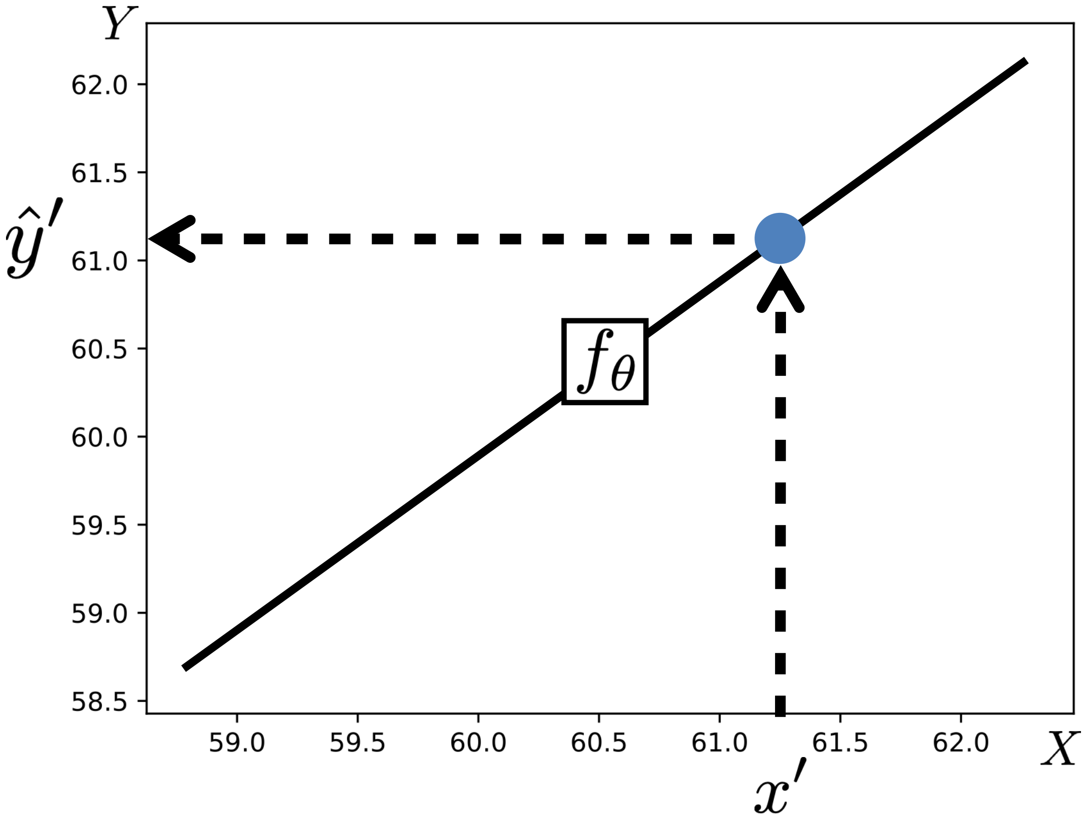
Test query



Training data



Test query

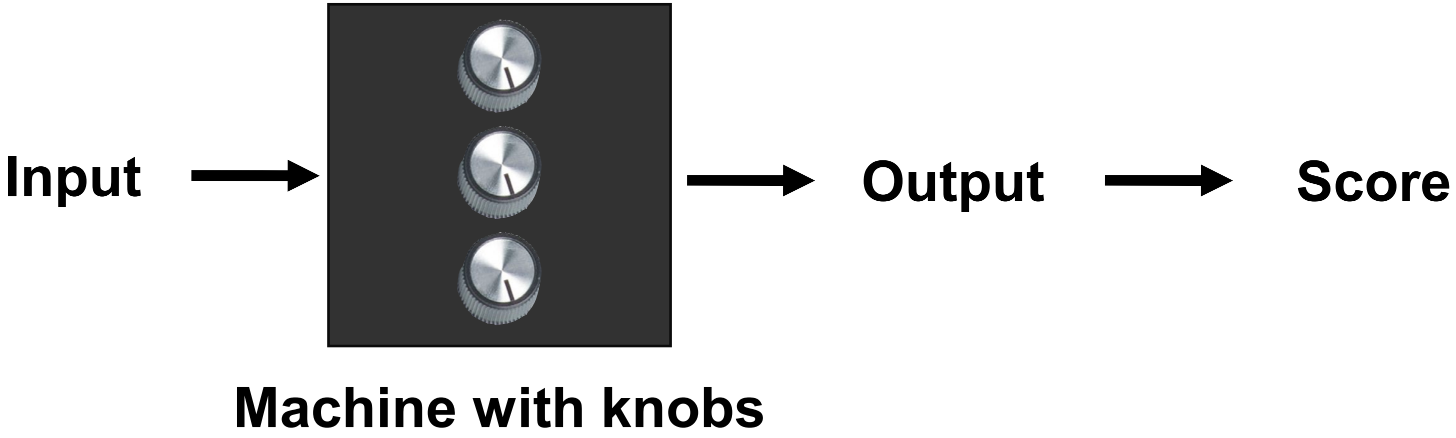


$$x' \dashrightarrow \boxed{f_\theta} \dashrightarrow \hat{y}'$$

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

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

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

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

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

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

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

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

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

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

Solution

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

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

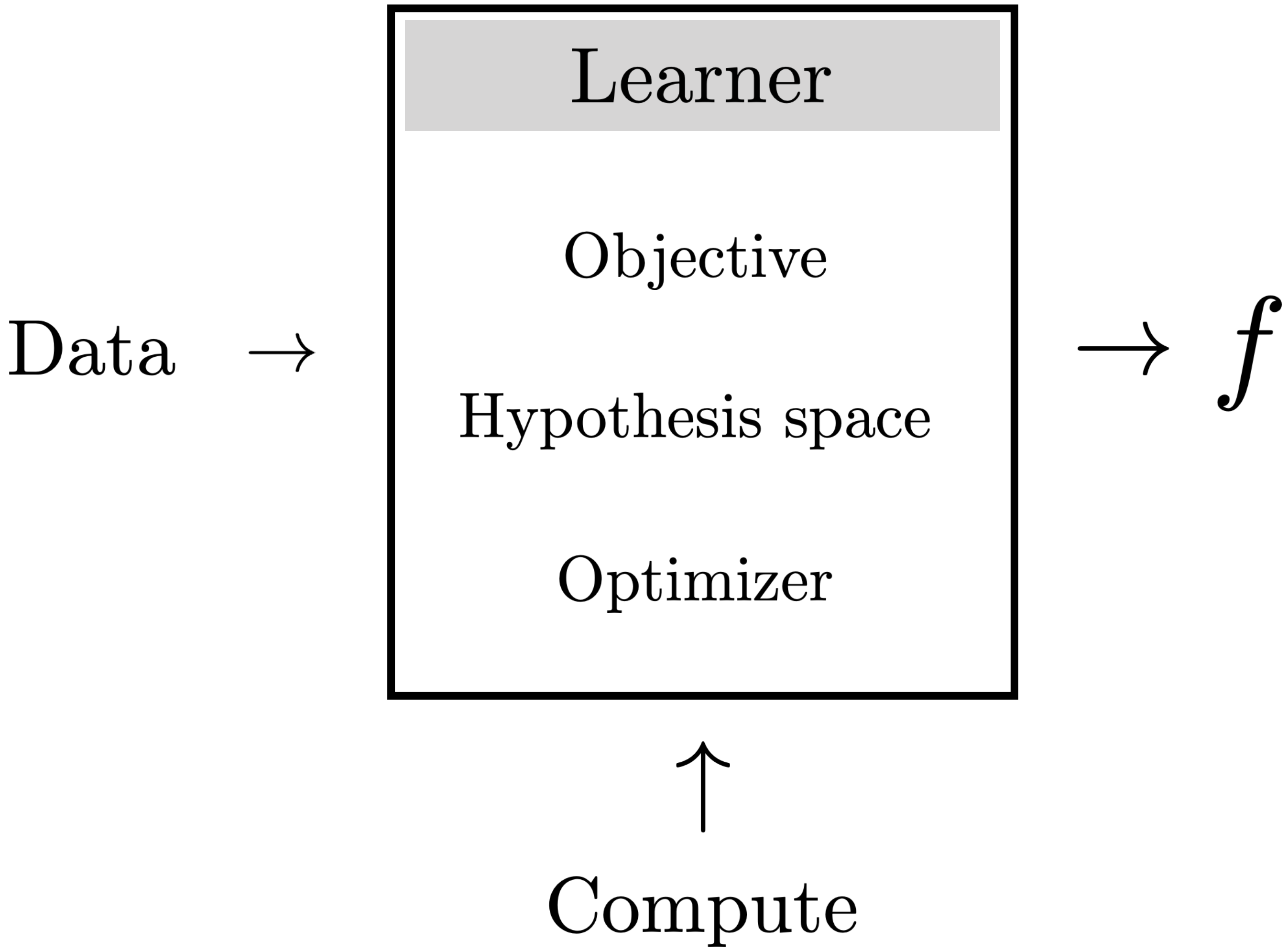
Hypothesis space

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

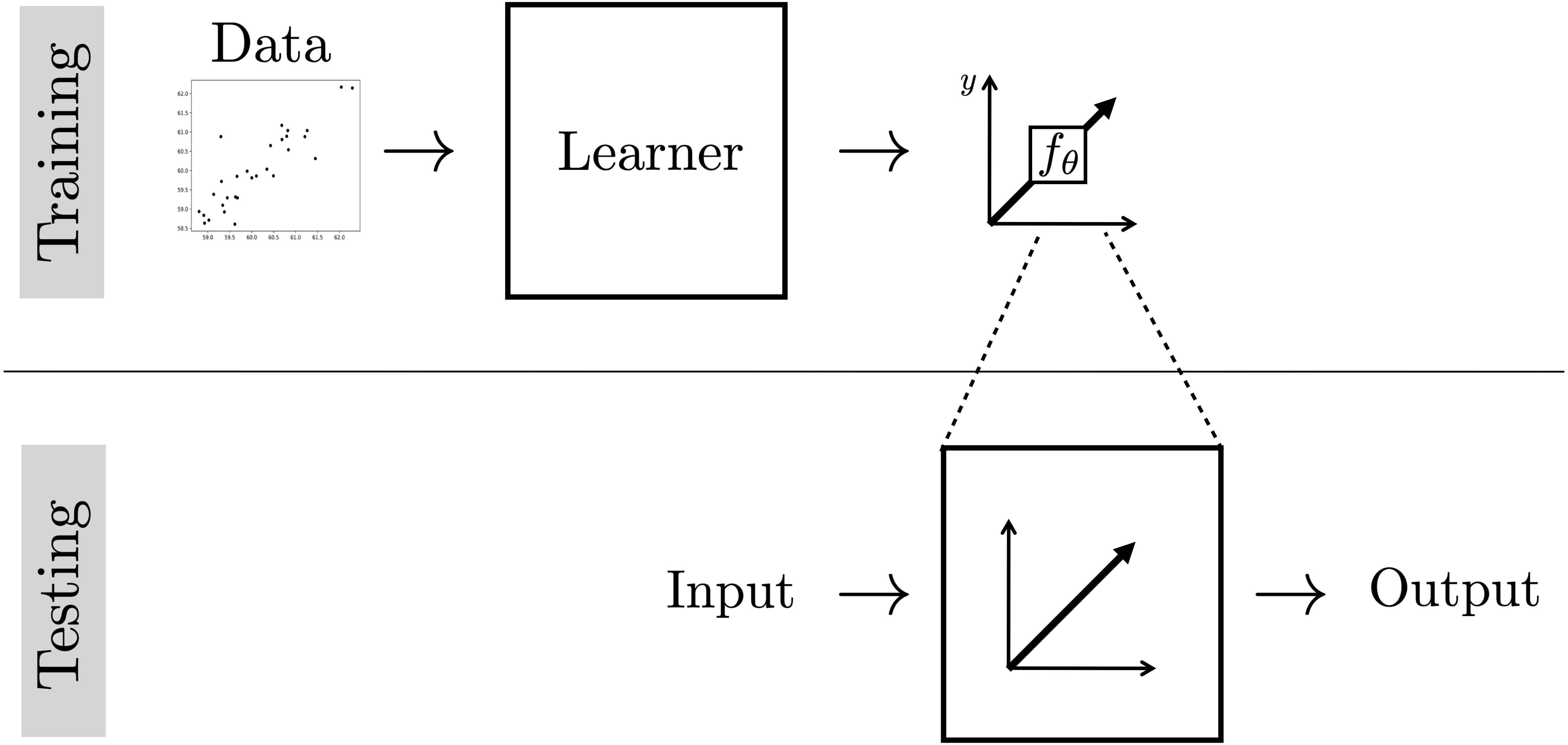
Optimizer

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

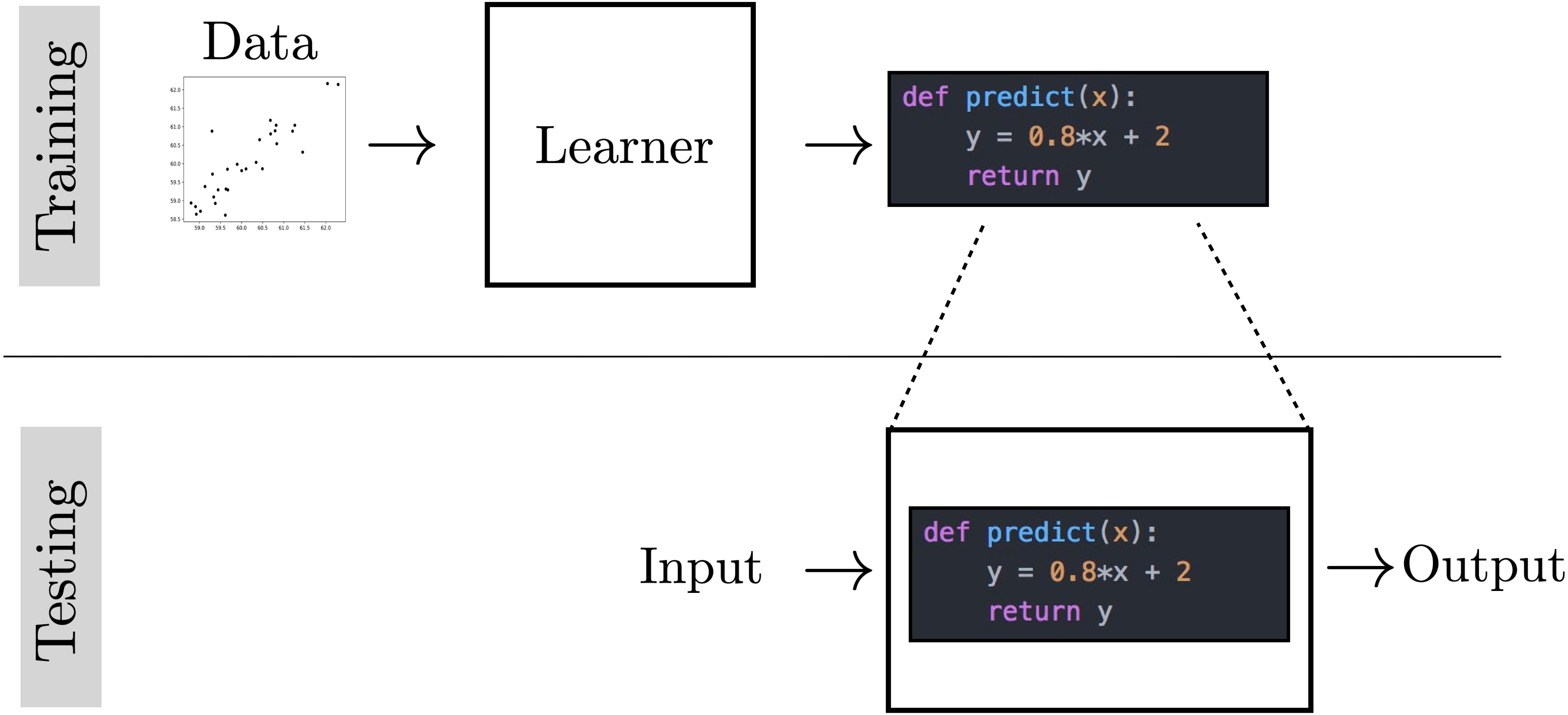
$\rightarrow f$


Learner
Objective
Hypothesis space
Optimizer
Data
f
Compute

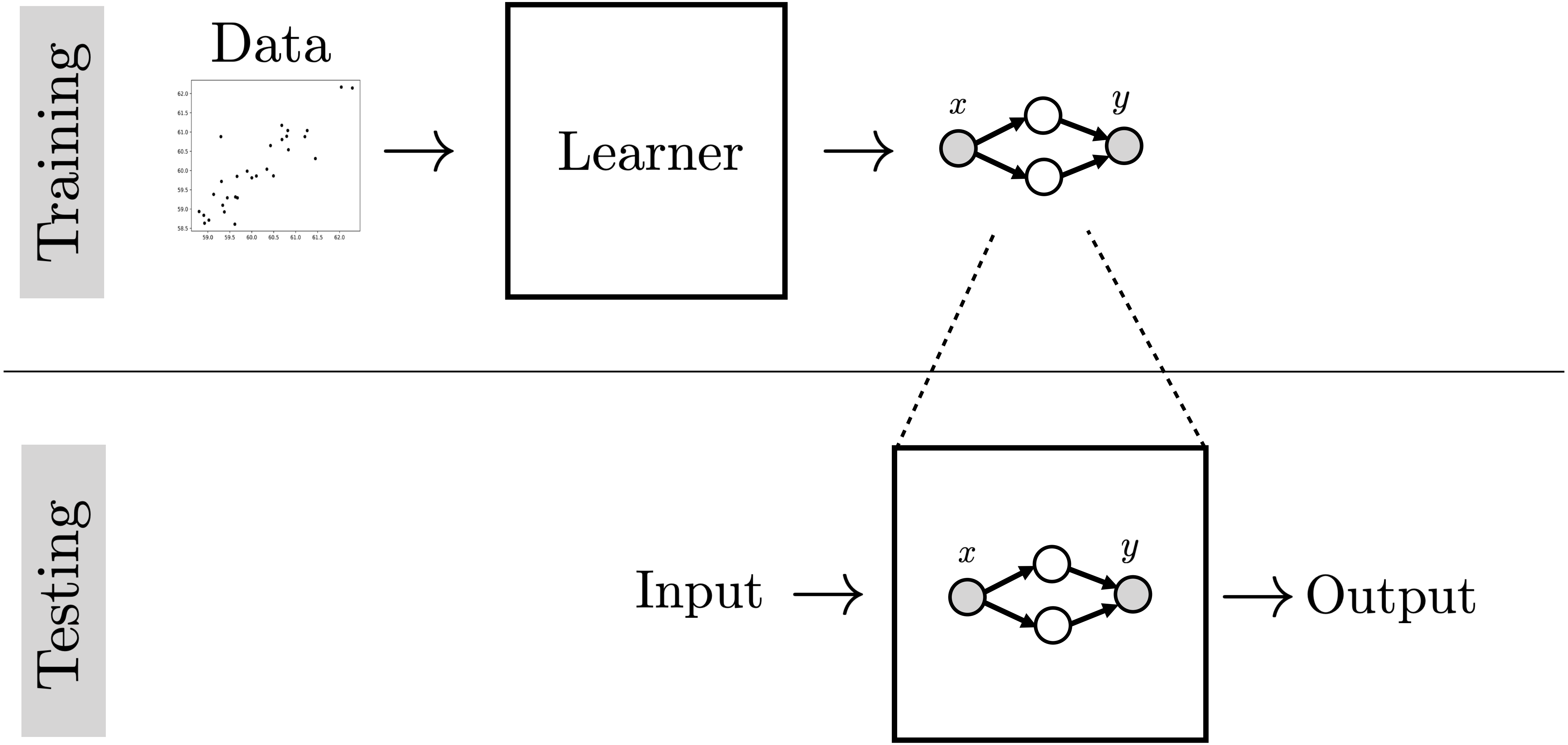
Example 1: Linear least squares

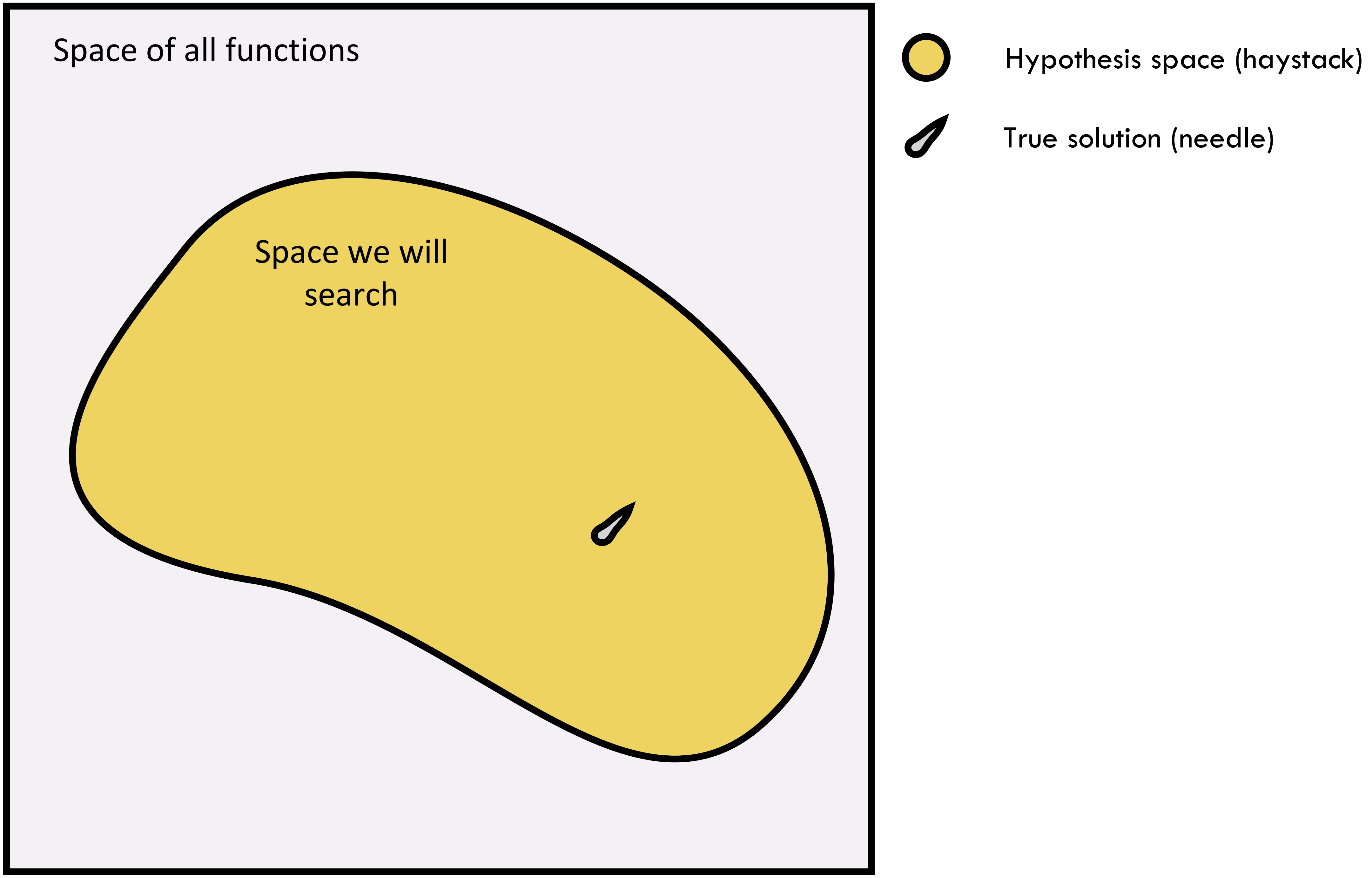


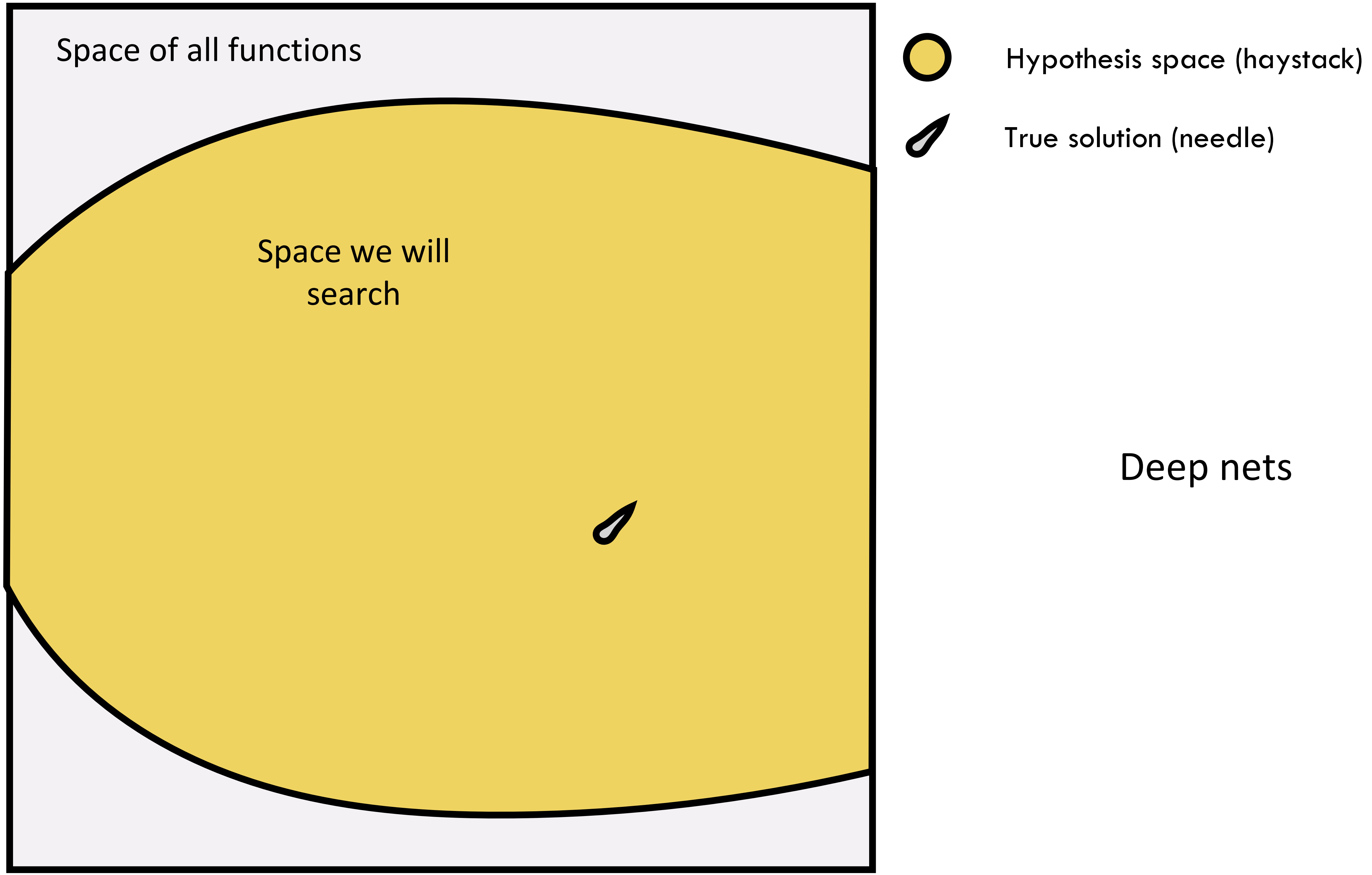
Example 2: Program induction



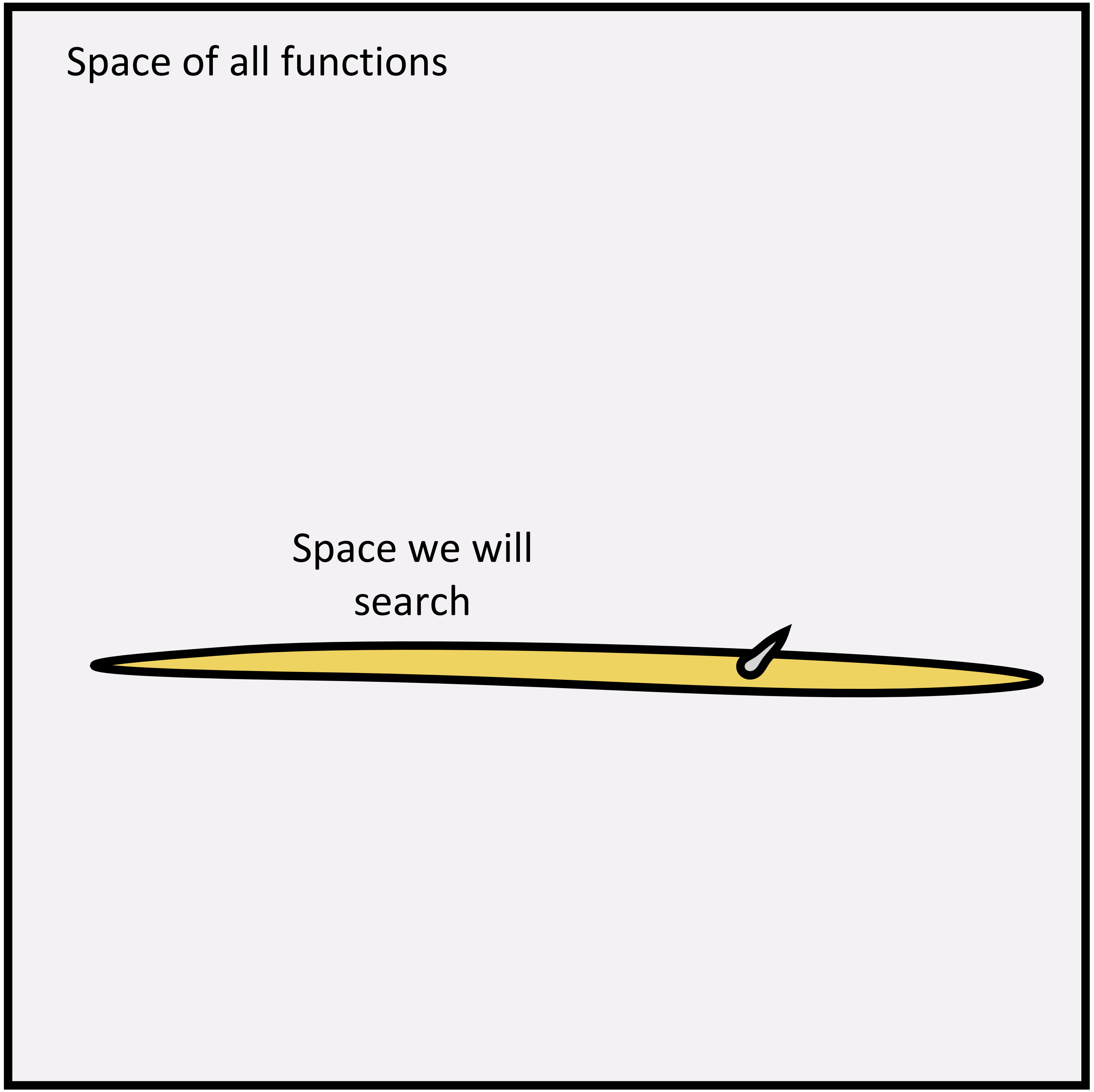
Example 3: Deep learning

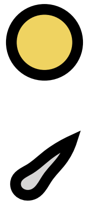



Space of all functions
Space we will search
Hypothesis space (haystack)
True solution (needle)


Space of all functions
Space we will search
Hypothesis space (haystack)
True solution (needle)

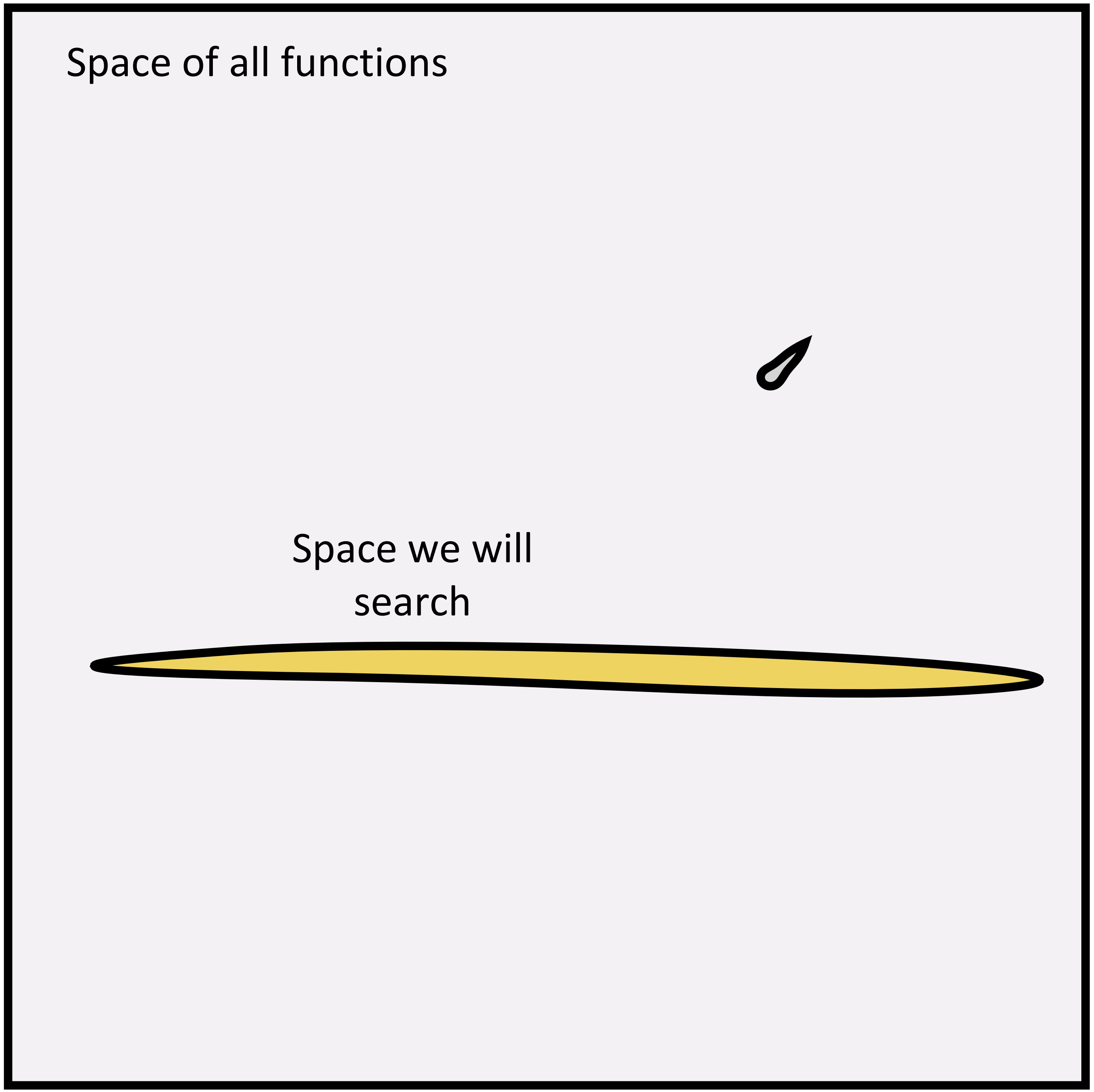
Deep nets





Linear functions

True solution is linear

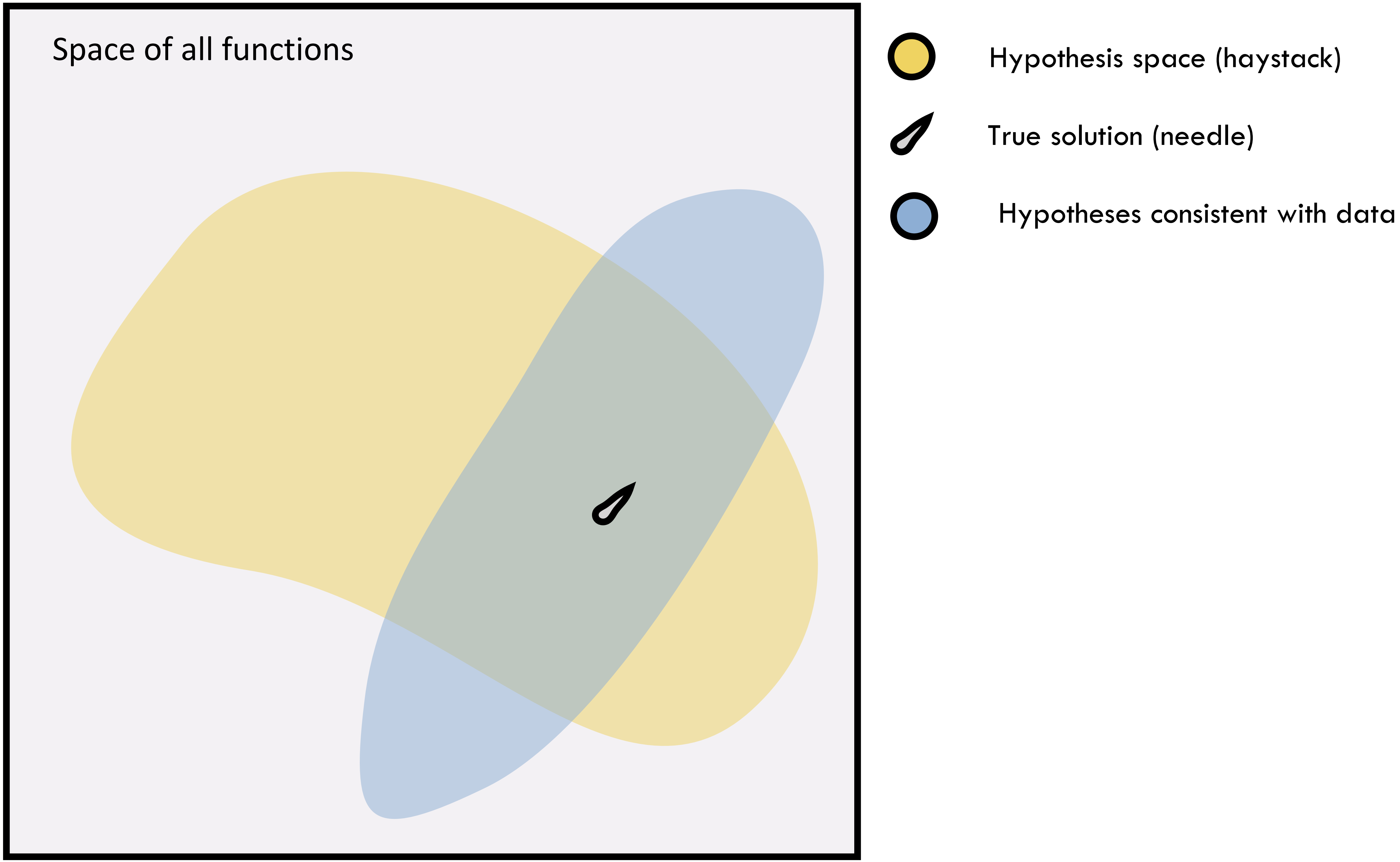


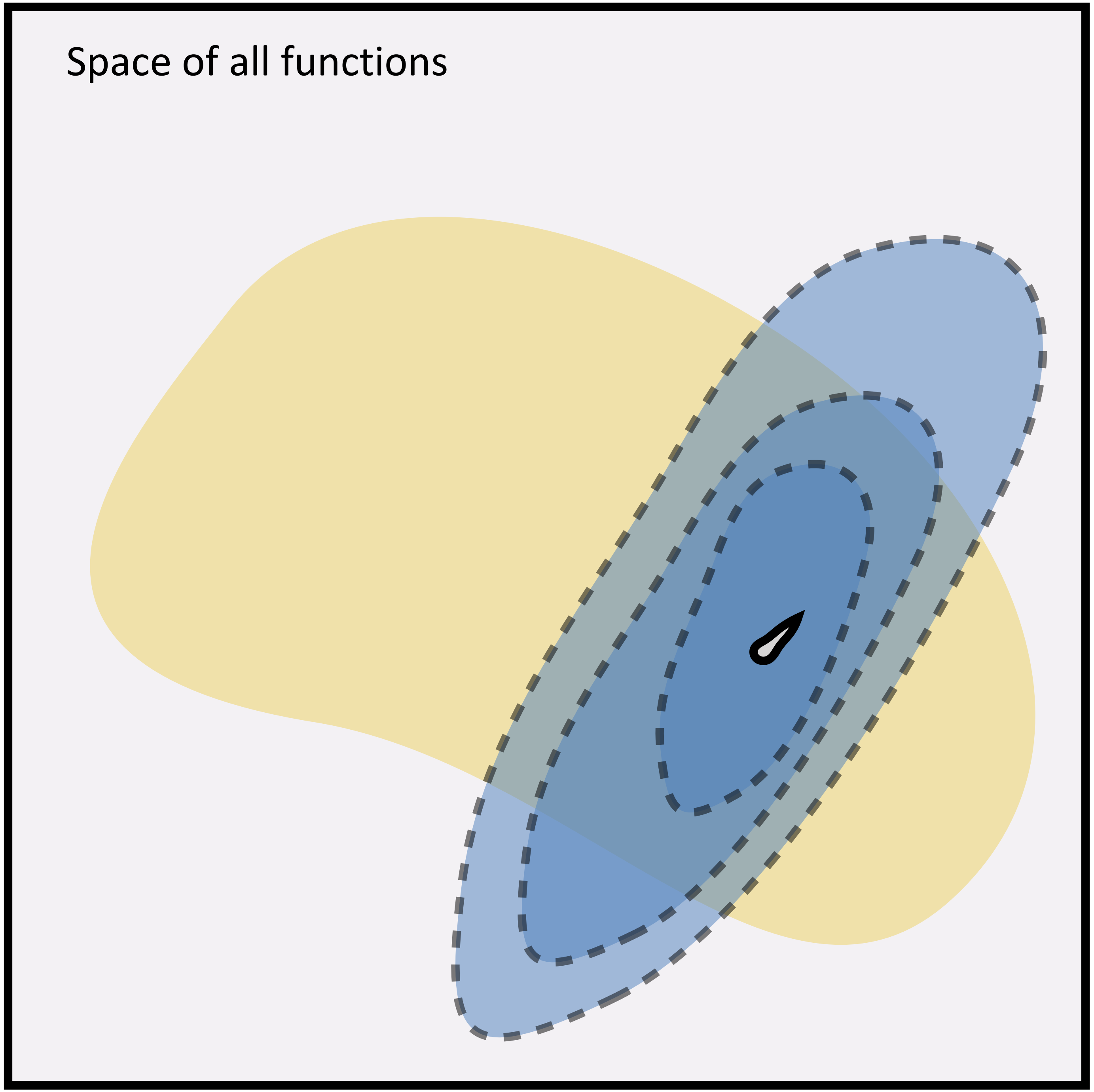
Hypothesis space (haystack)

True solution (needle)

Linear functions

True solution is nonlinear


Space of all functions
Hypothesis space (haystack)
True solution (needle)
Hypotheses consistent with data

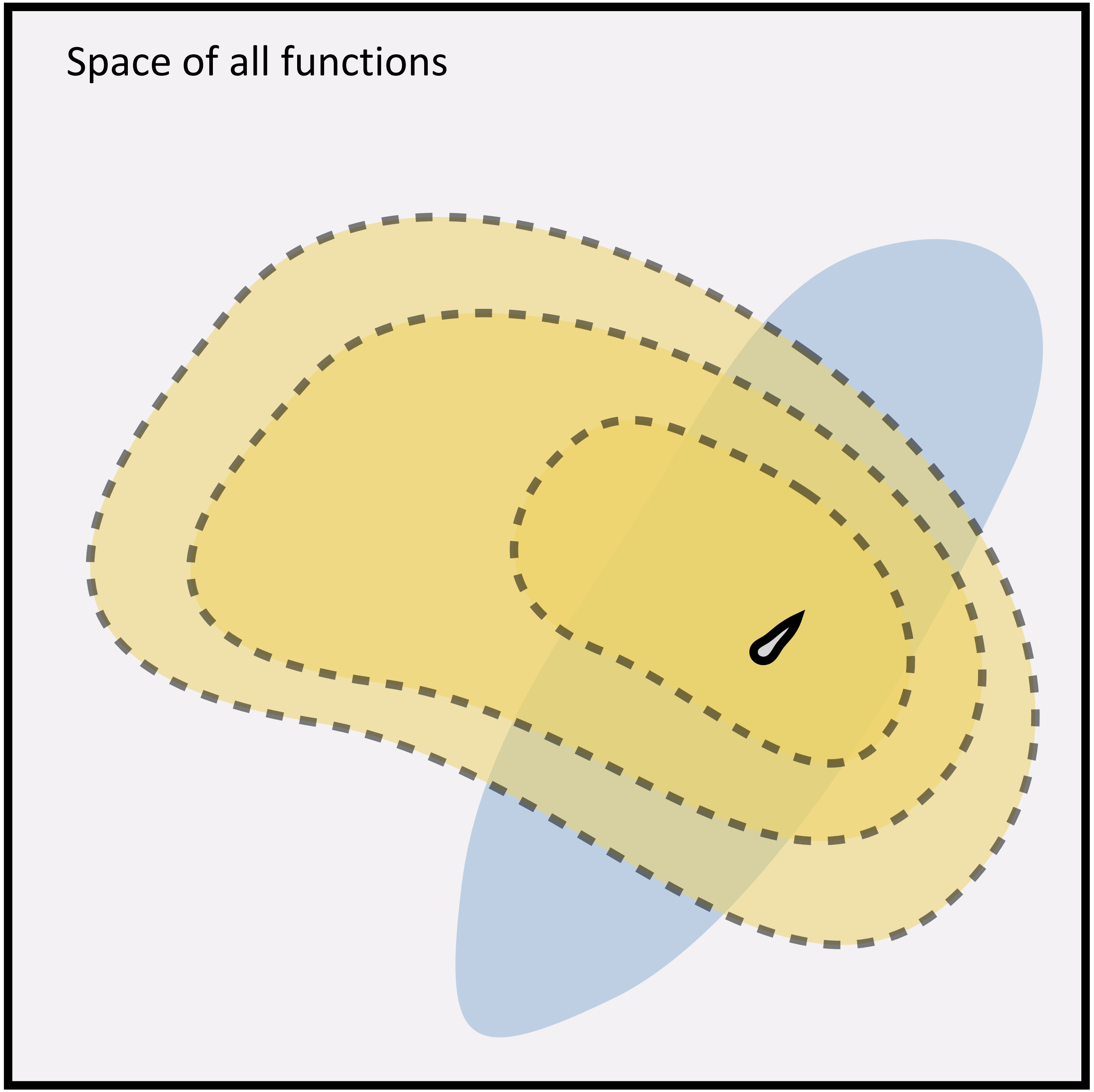


Hypothesis space (haystack)

True solution (needle)

Hypotheses consistent with data

What happens as we increase the data?



Hypothesis space (haystack)

True solution (needle)

Hypotheses consistent with data

What happens as we shrink the hypothesis space?

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?

Case study #2: Image classification

1. How do you represent the input and output?

2. What is the objective?

3. Assume hypothesis space is sufficienly expressive

4. Assume we optimize perfectly

5. Assume we train on exactly the data we care about

Image classification

Classifier

“Fish”

image **x**

label y

Image classification

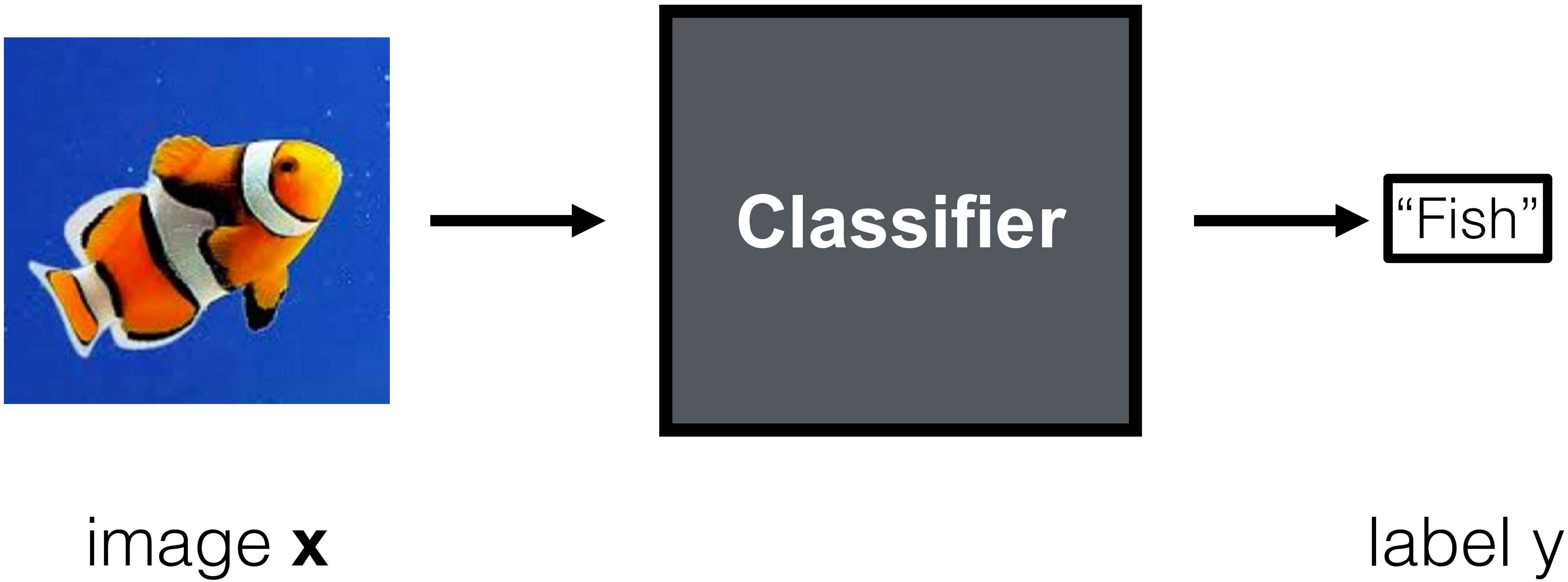


Image classification

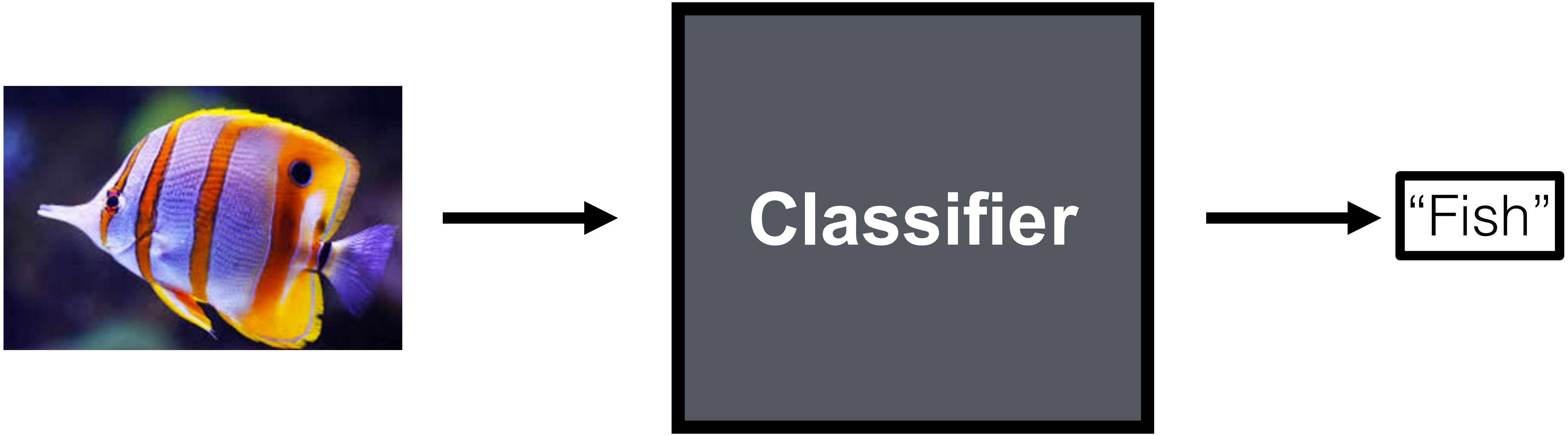
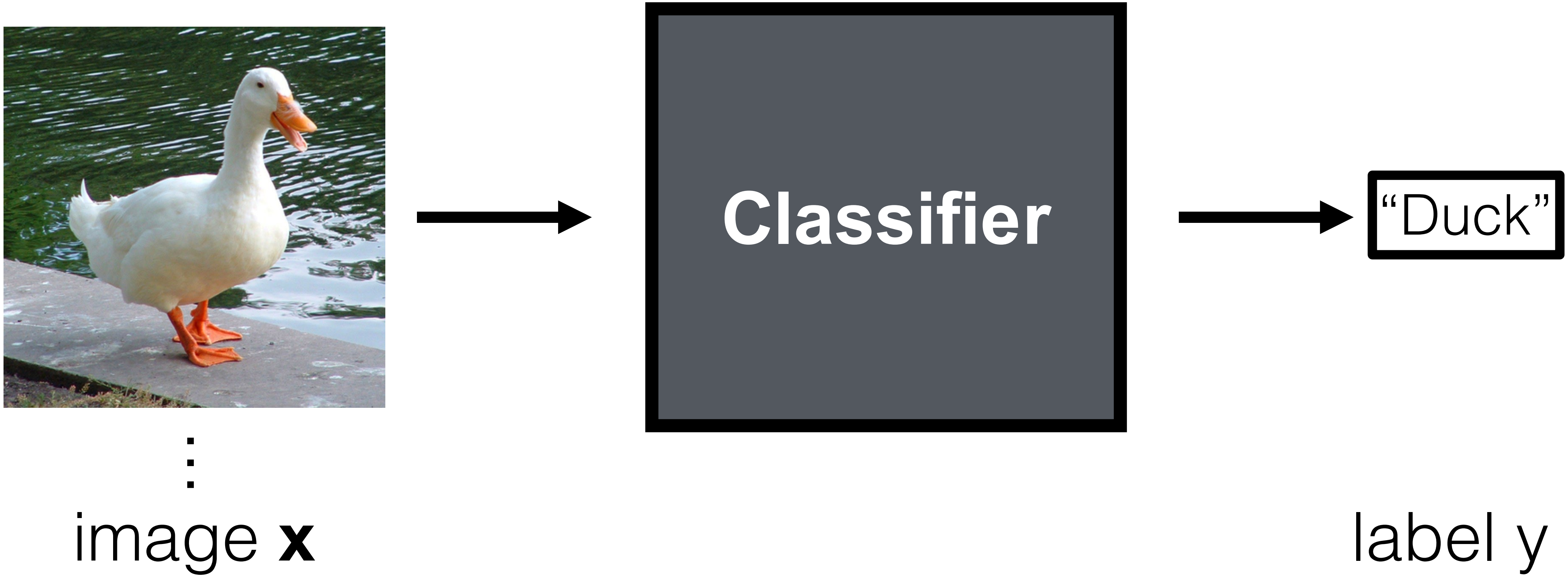
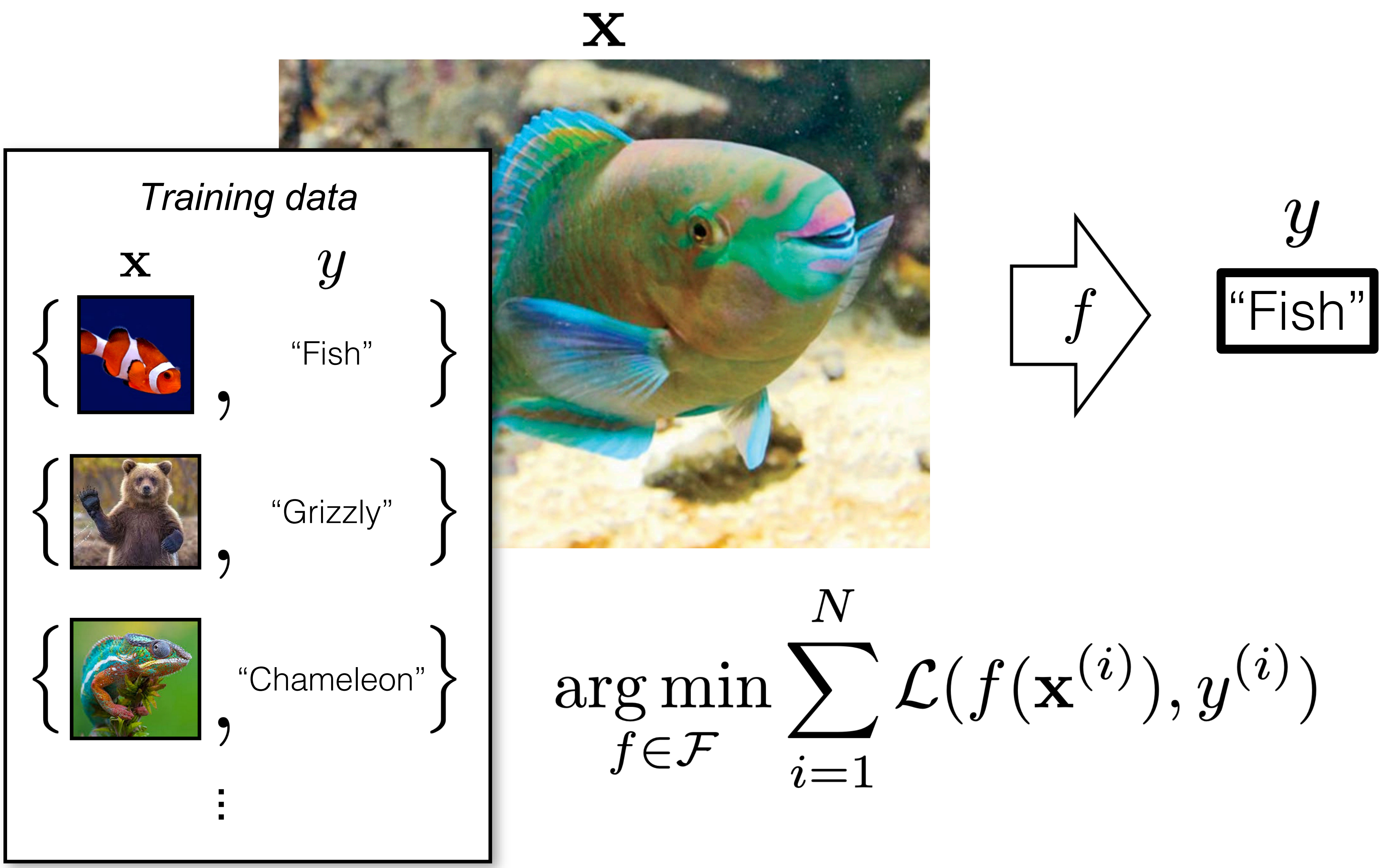


image **x**

label y

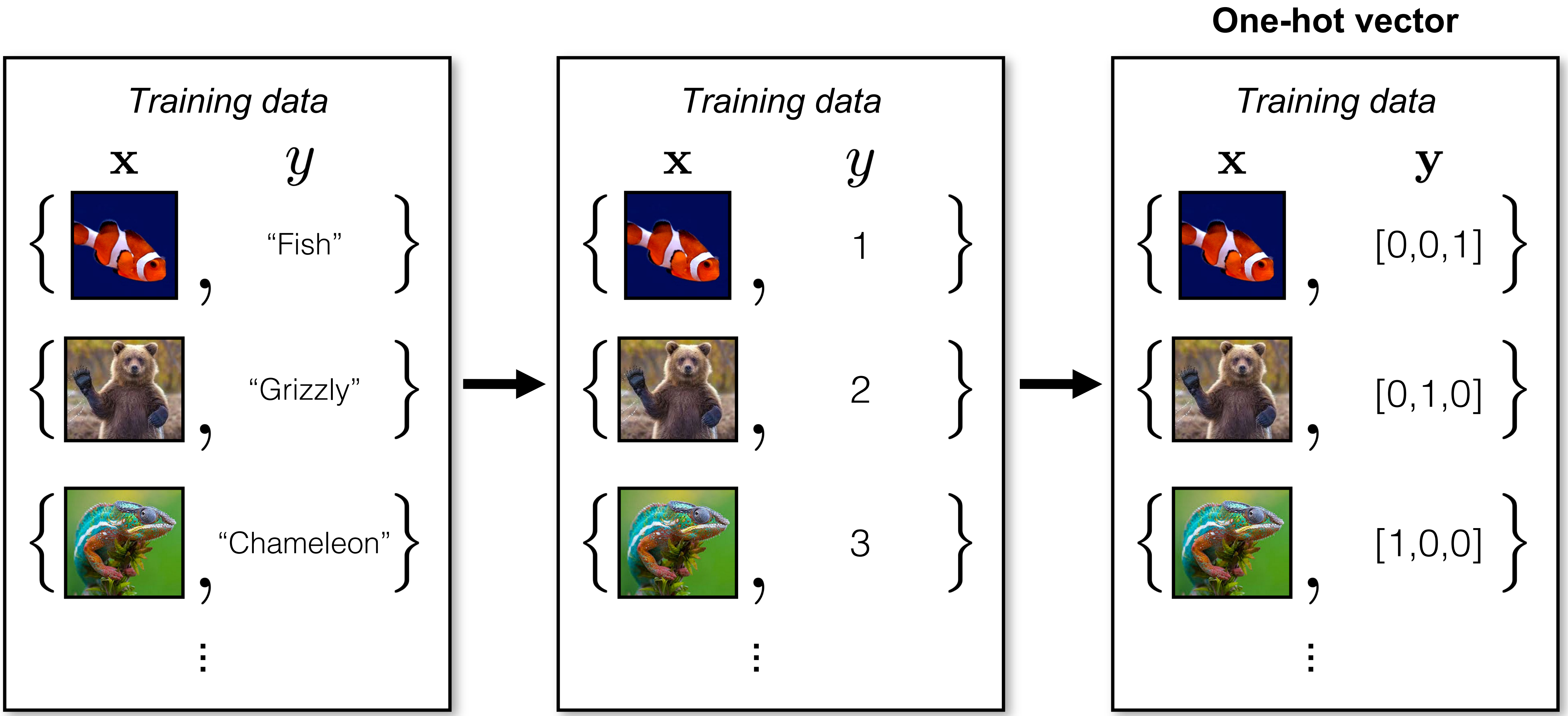
Image classification





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

How to represent class labels?



What should the loss be?

0-1 loss (number of misclassifications)

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

Cross entropy

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

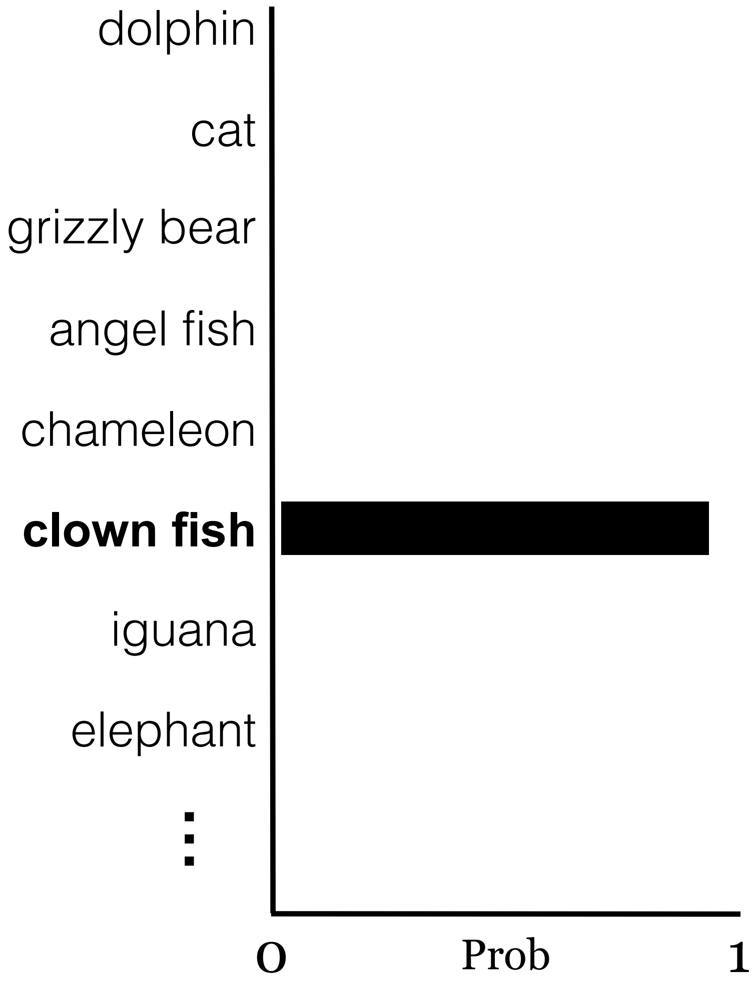
Ground truth label $\mathbf{y}$

$\mathbf{x}$

[0,0,0,0,0,1,0,0,…]

Ground truth label $\mathbf{y}$

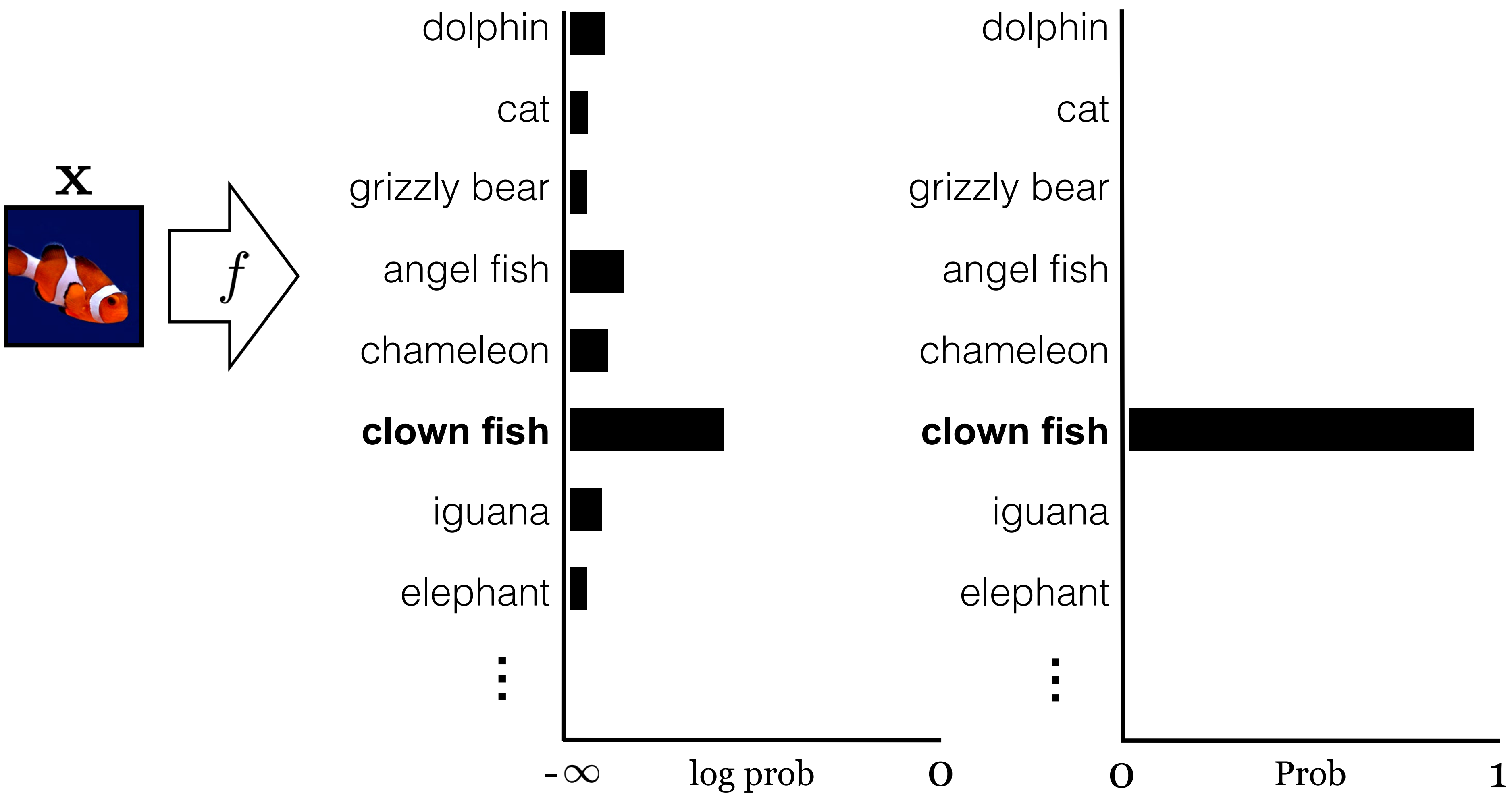
$\mathbf{x}$

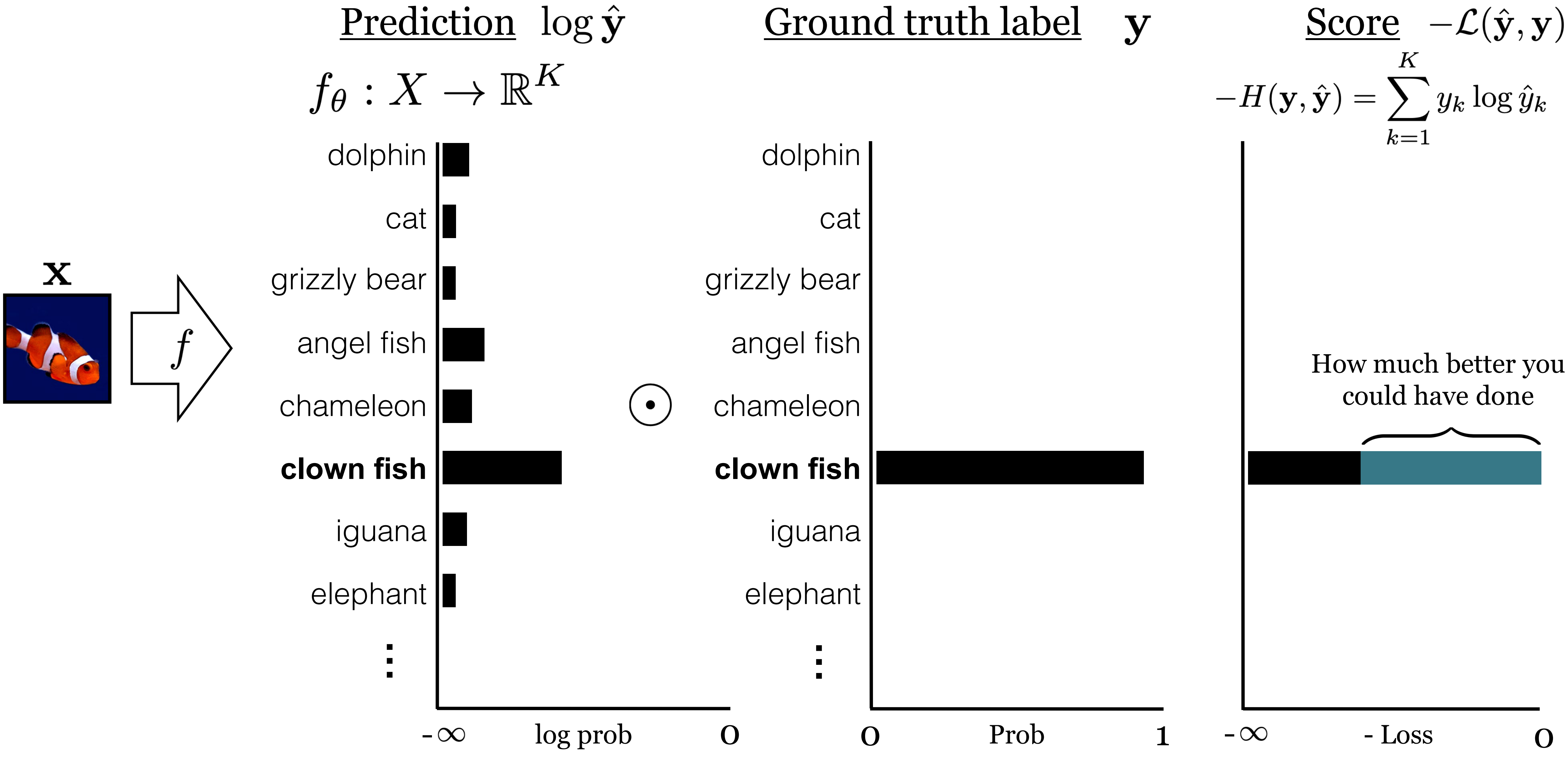


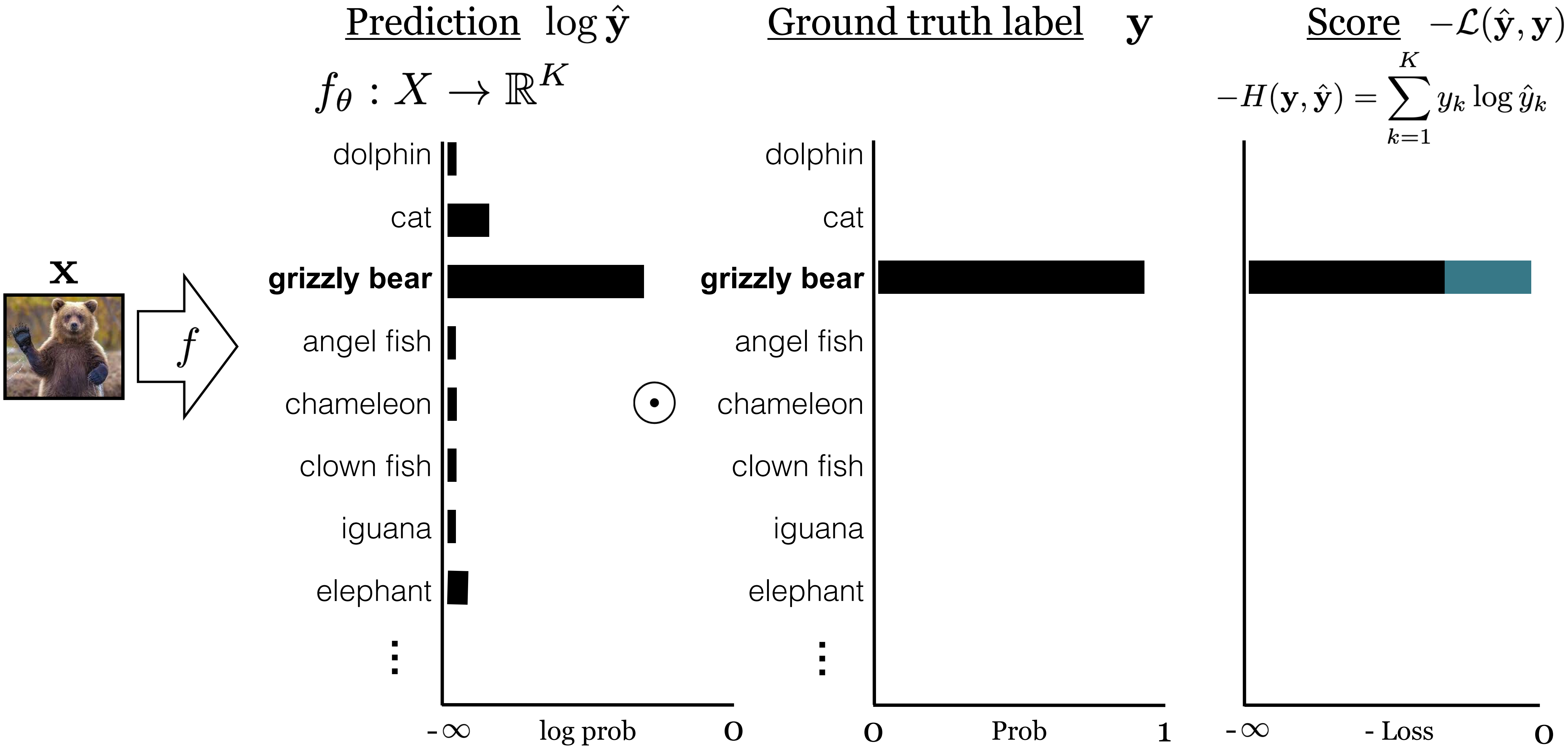
∞

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

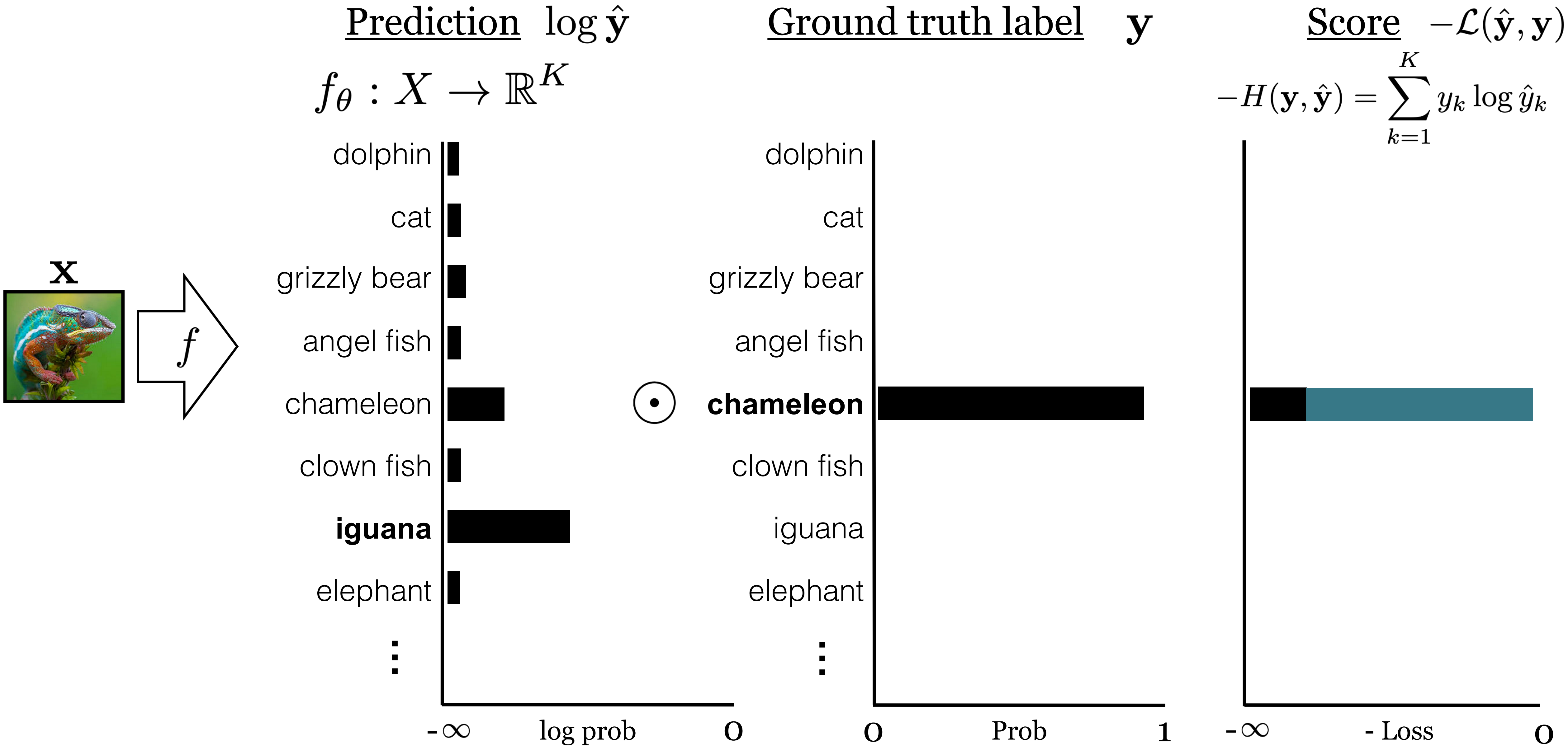
$$f_\theta : X \to \mathbb{R}^K$$




Prediction $\log \hat{\mathbf{y}}$
$f_\theta : X \to \mathbb{R}^K$
Ground truth label $\mathbf{y}$
Score $-\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$
$-H(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{k=1}^{K} y_k \log \hat{y}_k$
$\mathbf{x}$
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



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
$\odot$
dolphin
cat
grizzly bear
angel fish
chameleon
clown fish
iguana
elephant
$-\infty$ log prob 0
0 Prob 1
$-\infty$ - Loss 0


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

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

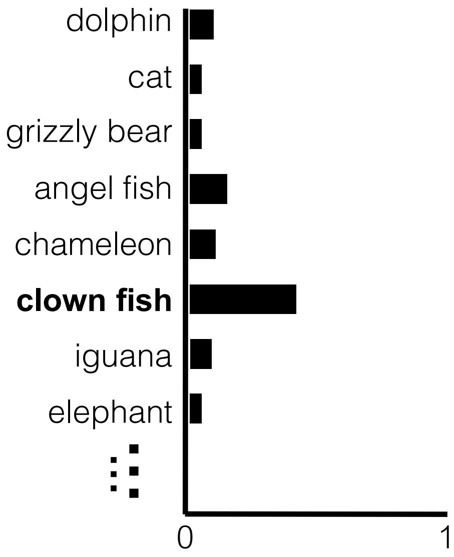
$$f_\theta : X \to \mathbb{R}^K$$

$$\mathbf{z} = f_\theta(\mathbf{x})$$ ← **logits**: vector of K scores, one for each class

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

$$\hat{\mathbf{y}} \equiv [P_\theta(Y = 1|X = \mathbf{x}), \ldots, P_\theta(Y = K|X = \mathbf{x})]$$ ← 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)

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

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

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

Data

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

Learner

Objective

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

$\to f$

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.