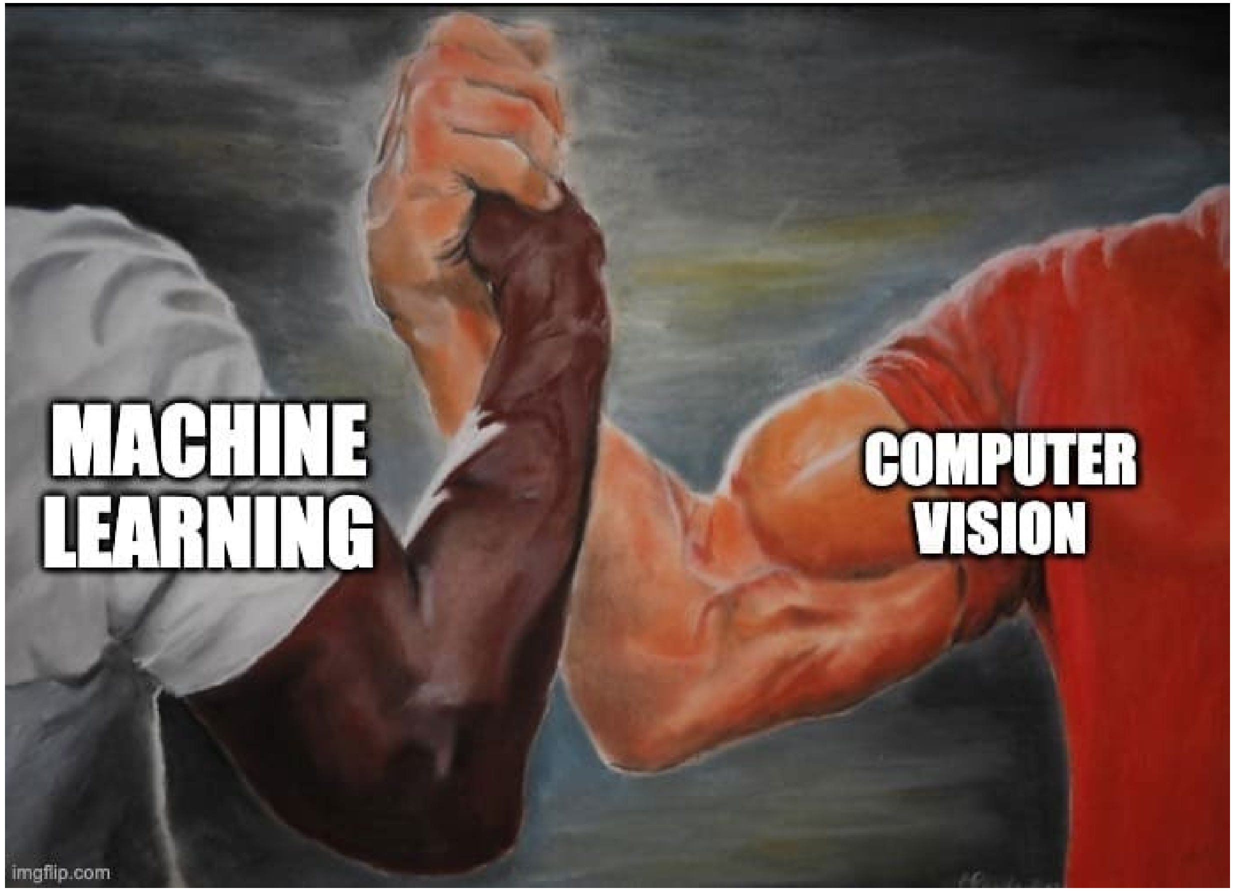
CMSC 472/672

Lecture 7

Machine Learning for Computer Vision



Some slides from Isola

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?

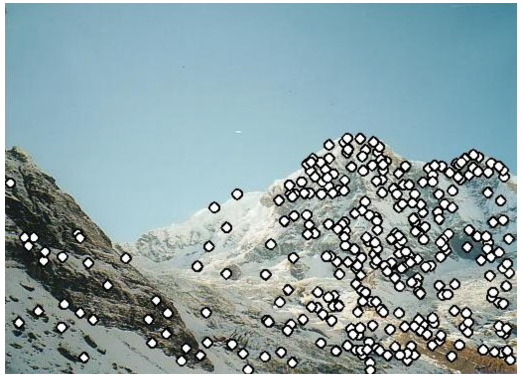
How can a computer vision system make a decision like this?

Ideas

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

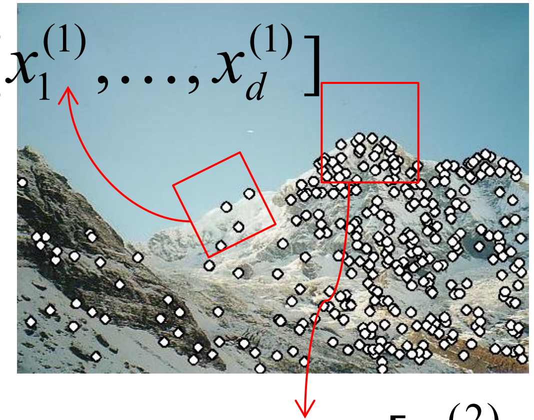
Features: main components

1) Detection: Identify the interest points



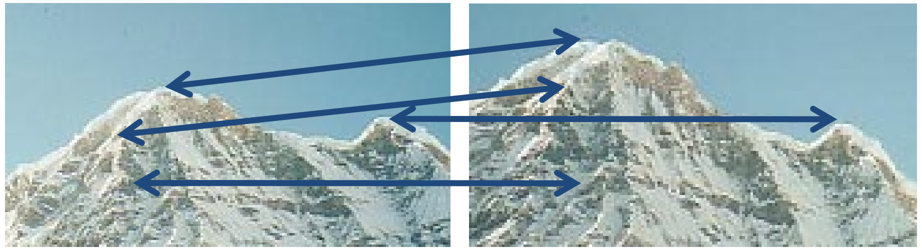
2) **Description: Extract vector feature descriptor surrounding each interest point.**

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



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

3) Matching: Determine correspondence between descriptors in two views



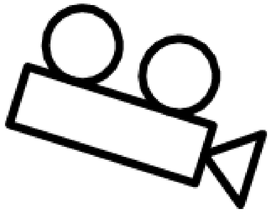
Kristen Grauman

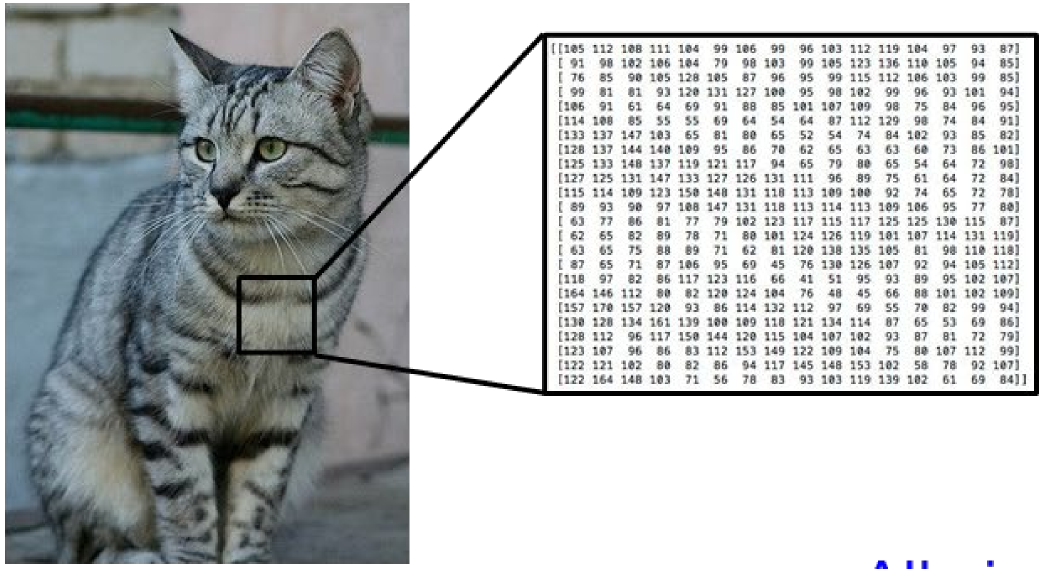
Recap & Motivation

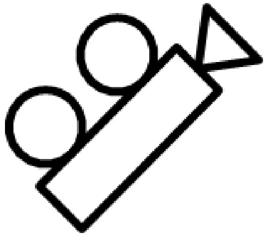
- Image features are “interesting”, “unique” regions in an image
  - Intuitively these are “important”
- So far – we have seen how to detect and describe (*a.k.a. “represent”)* certain types of feature
  - Harris Corners, Blobs, …

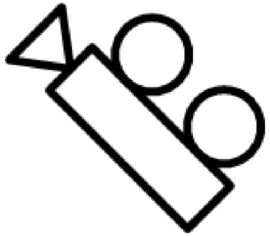
- We had a definition for what a “feature” is
  - Can we learn that from data?

Challenges: Viewpoint Variation



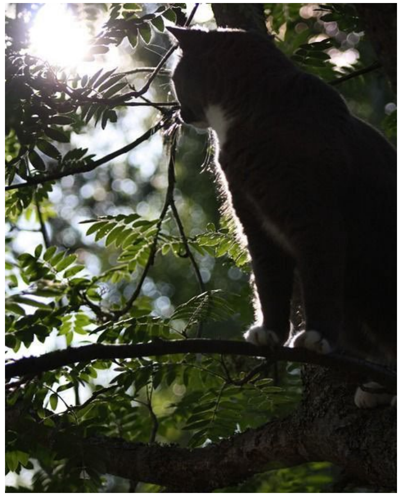






All pixels change when
the camera moves!

Challenges: Illumination

Slide Credit: Fei-Fei Li

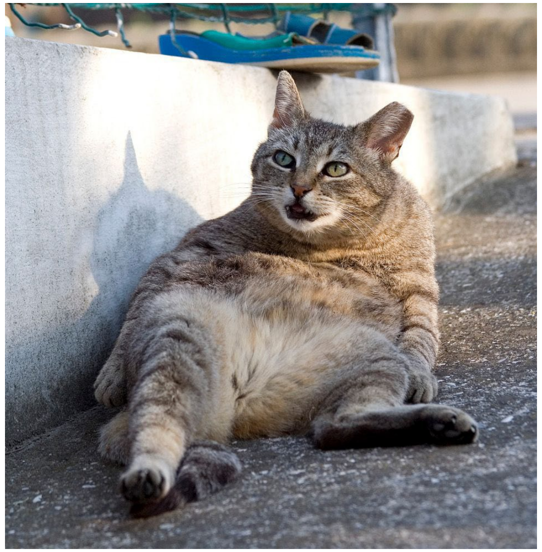
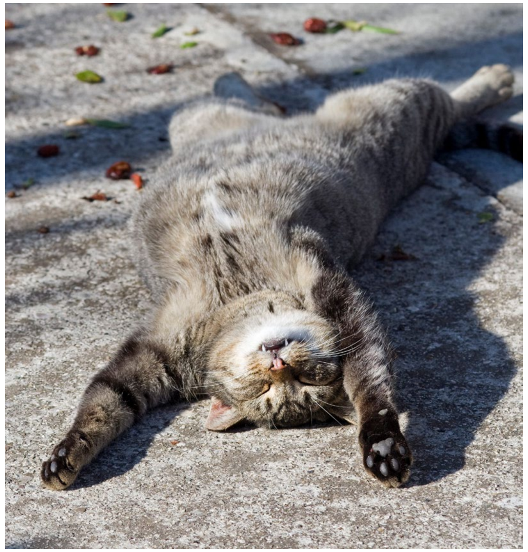
Challenges: Background Clutter

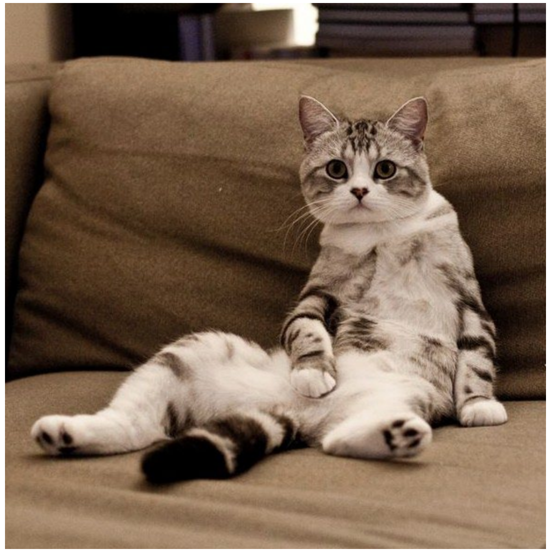
Slide Credit: Fei-Fei Li

Challenges: Occlusion

Slide Credit: Fei-Fei Li

Challenges: Pose and Deformation (Cat Yoga)

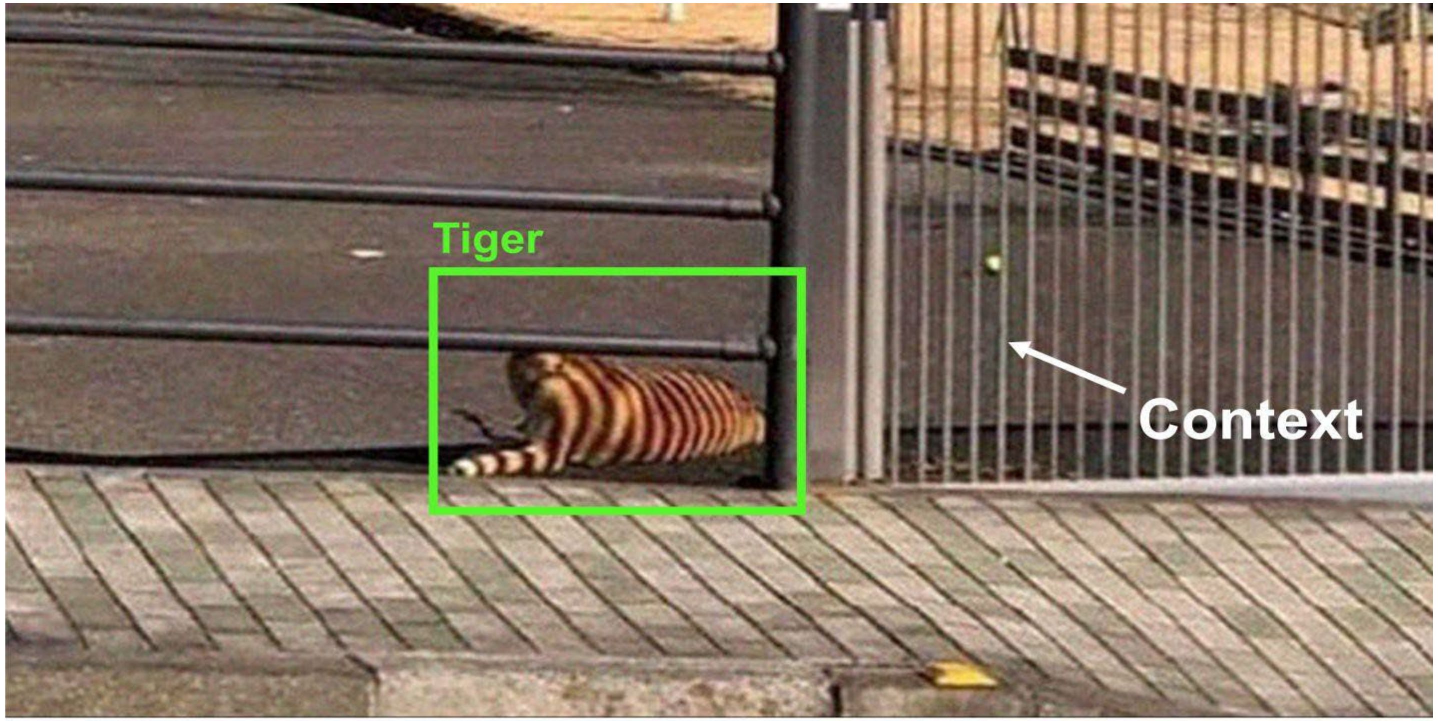


Slide Credit: Fei-Fei Li

Challenges: Inter-Class Variation

Slide Credit: Fei-Fei Li

Challenges: Illusions



Slide Credit: Fei-Fei Li

Data !!!

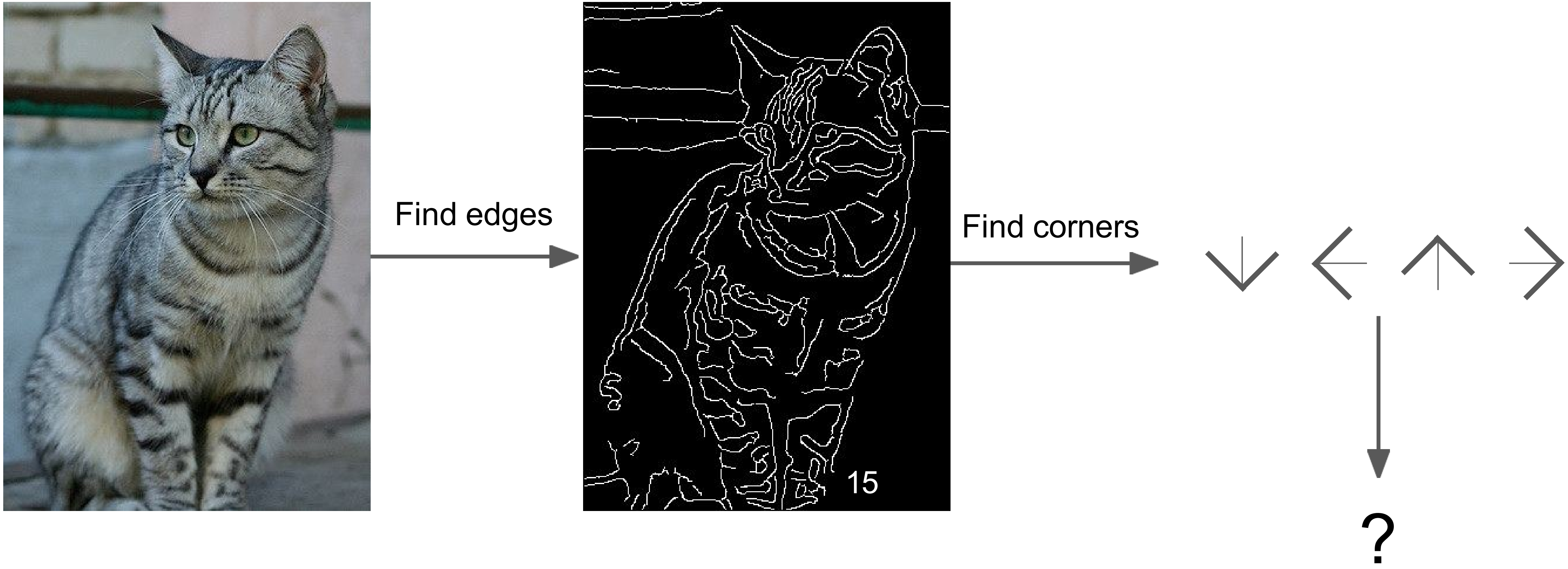
An image classifier

```
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.

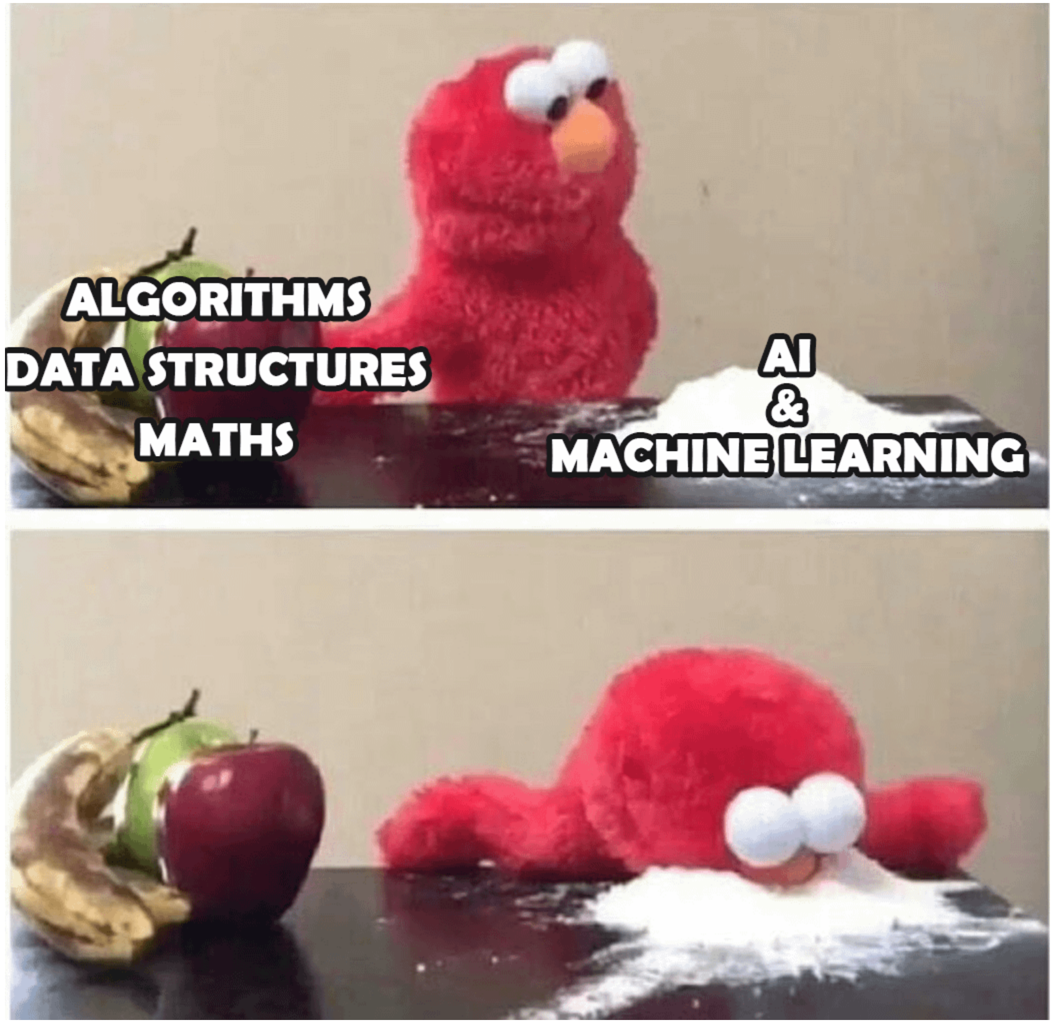
Can we use features to make the decision?



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

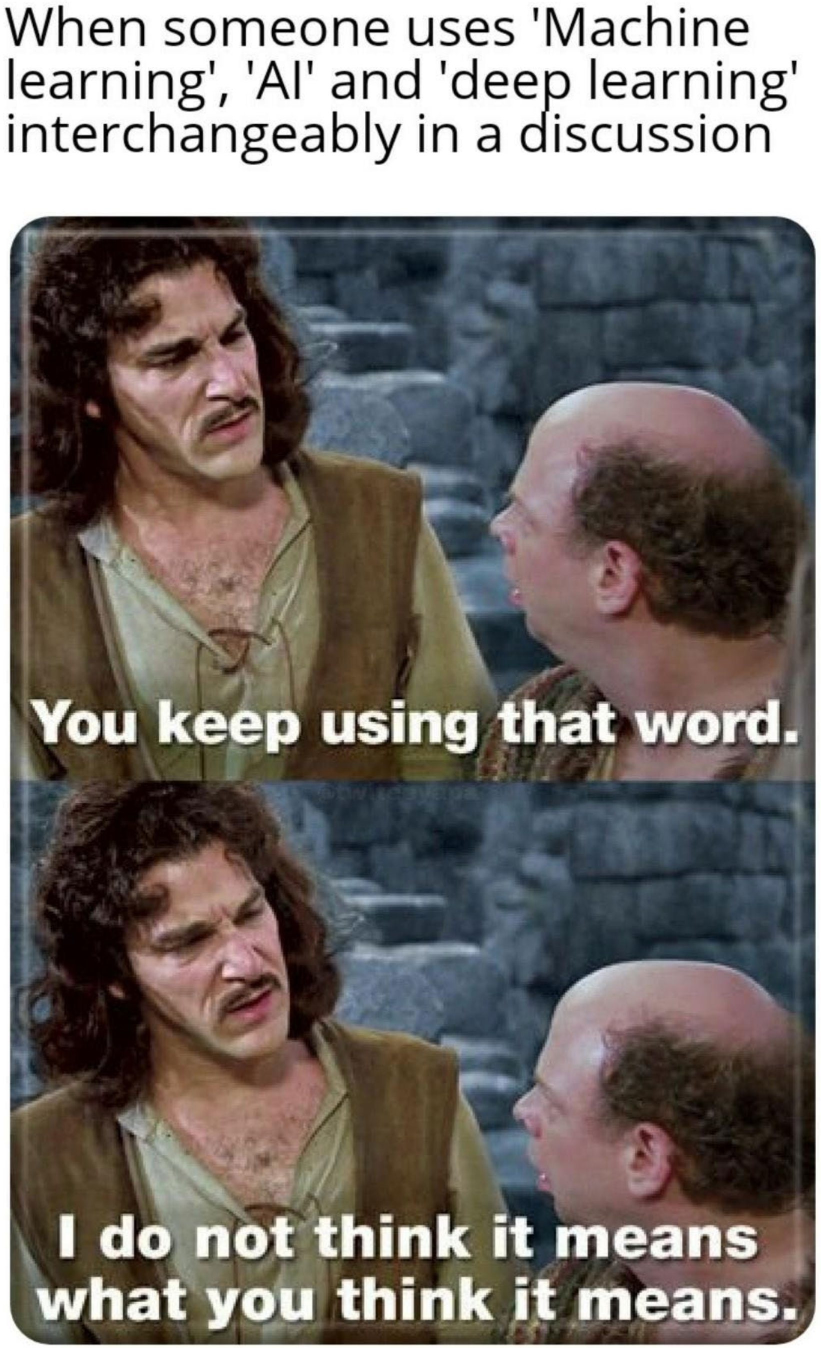
Get Ready for

The Good Stuff

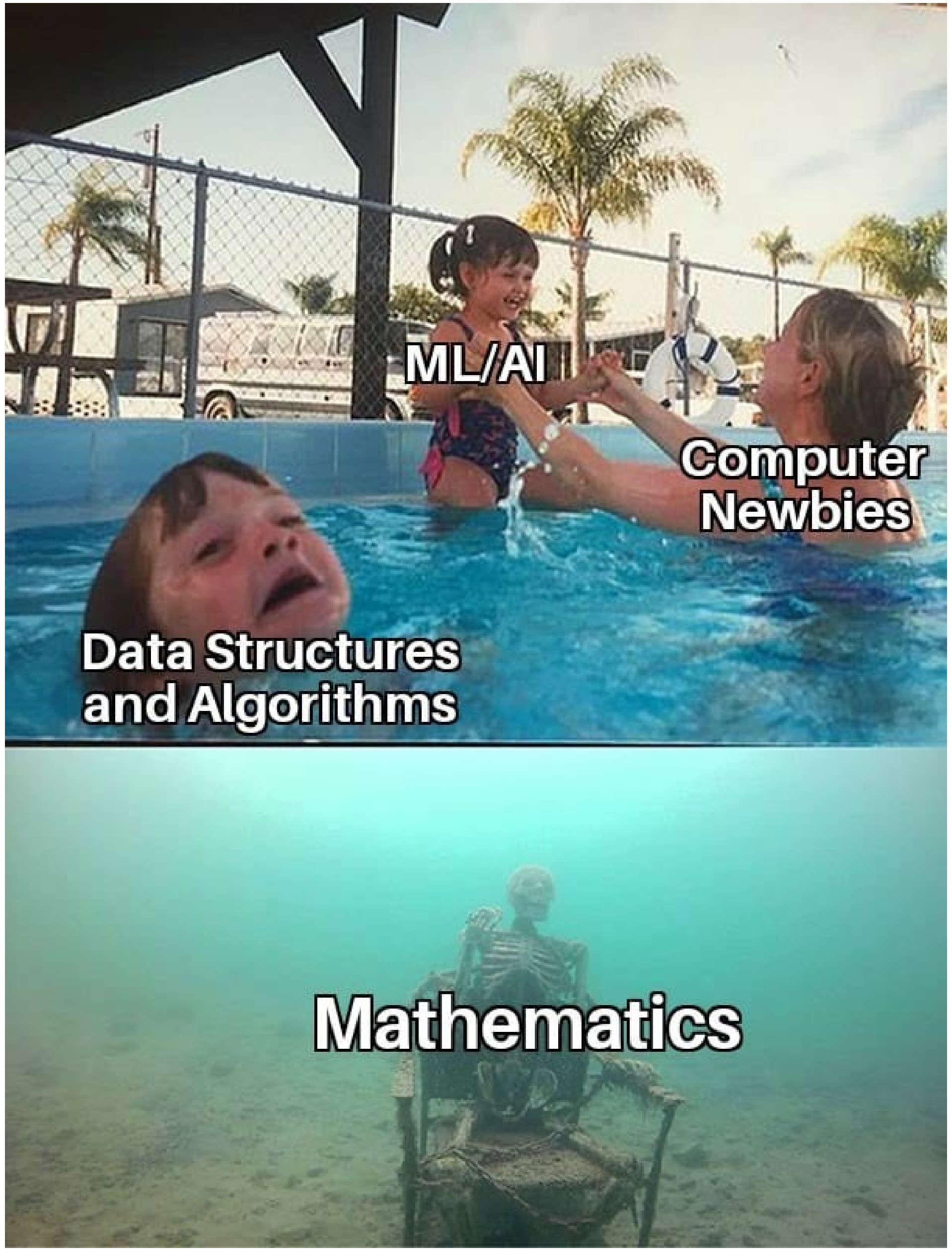


Wall Street / Silicon Valley

can you please stop

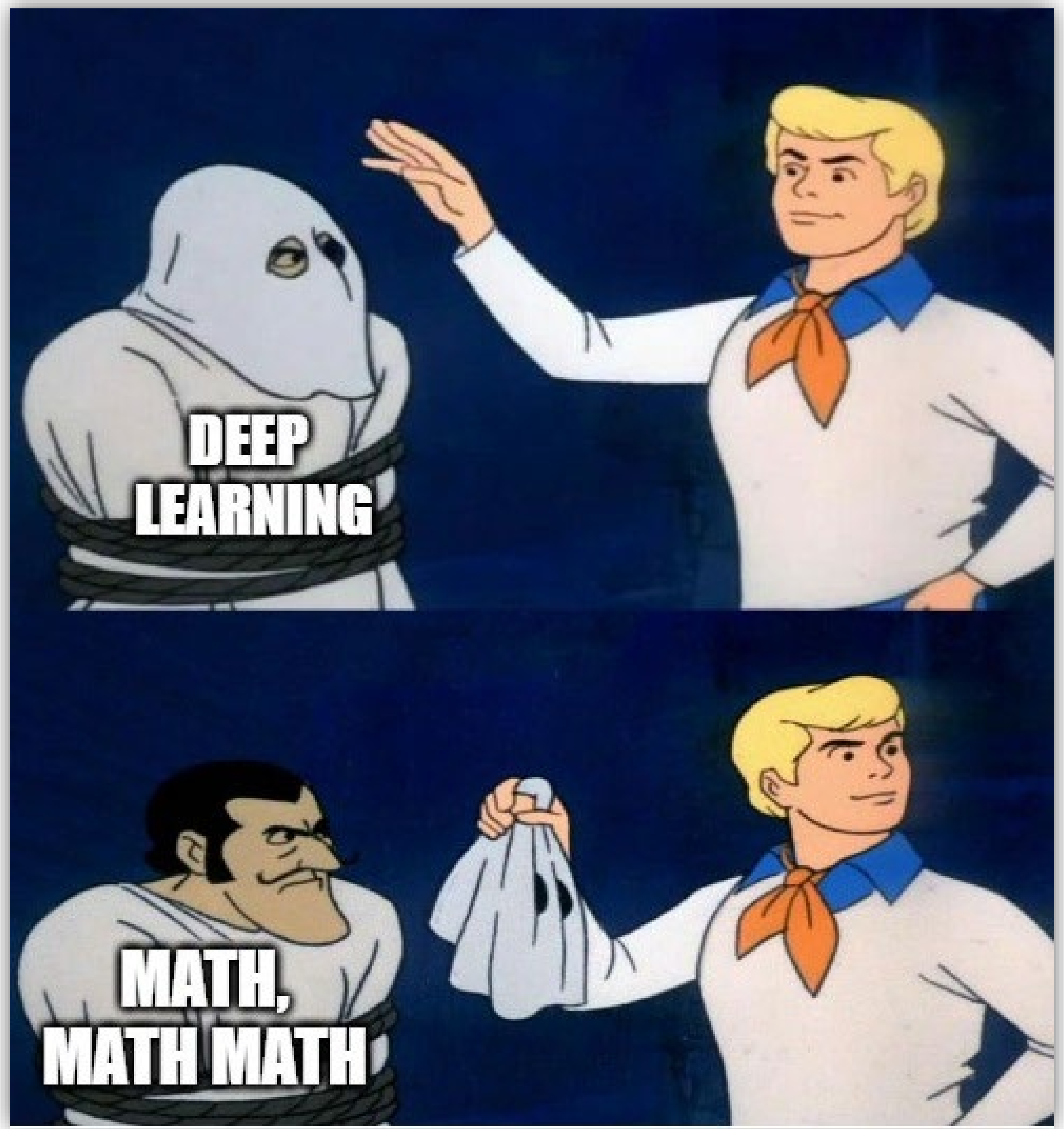


Know
your ancestors



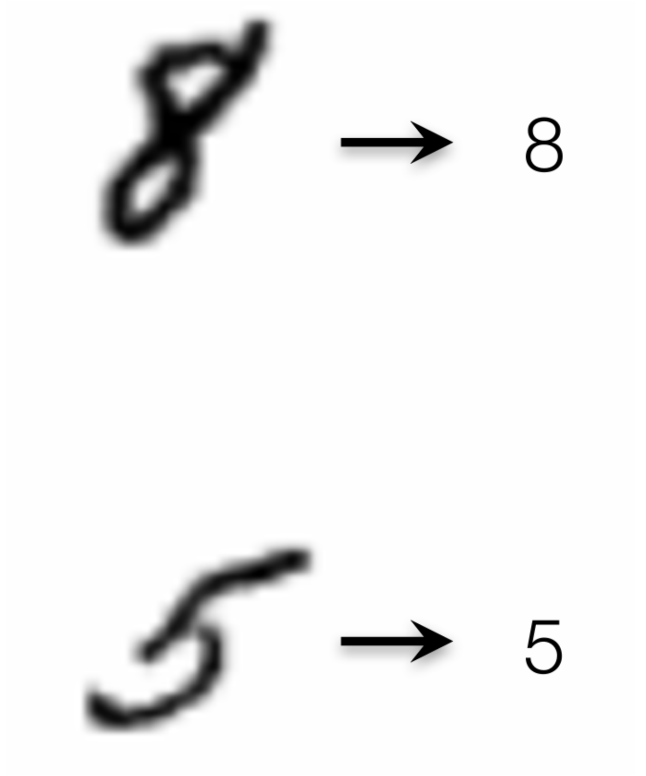
The Open Secret

The Open Secret



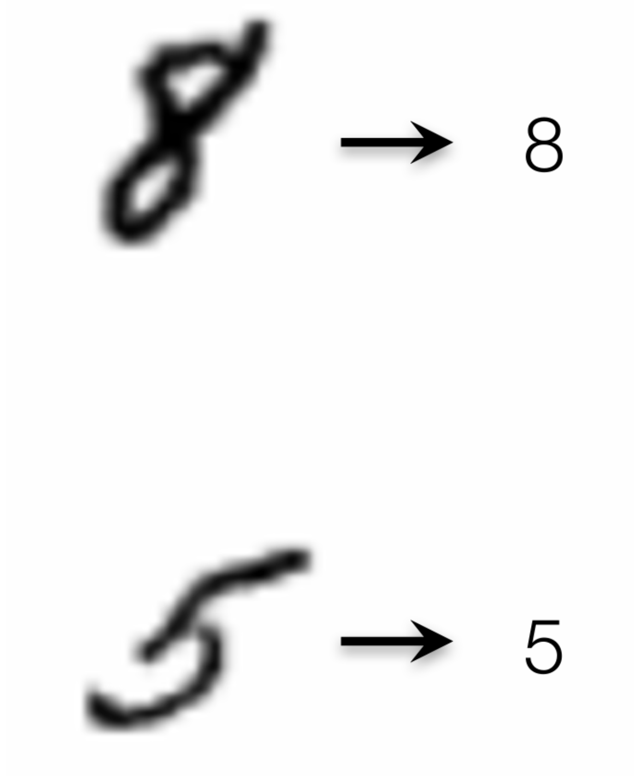
“Learn” ?

- Let’s look at a “programming” task
- The task:
  Write a program that outputs the number in a 28x28 grayscale image



“Learn” ?

- **Approach 1:** try to write a program by hand
  - o How would you do it ?



“Learn” ?

- **Approach 1:** try to write a program by hand
  - How would you do it ?
- **Approach 2:** (the machine learning approach)
  - Collect a large “dataset” of digit images
  - “Label” them with the corresponding numbers $(0, 1, \dots, 9)$
  - Let the system “write its own program” to map from images to numbers

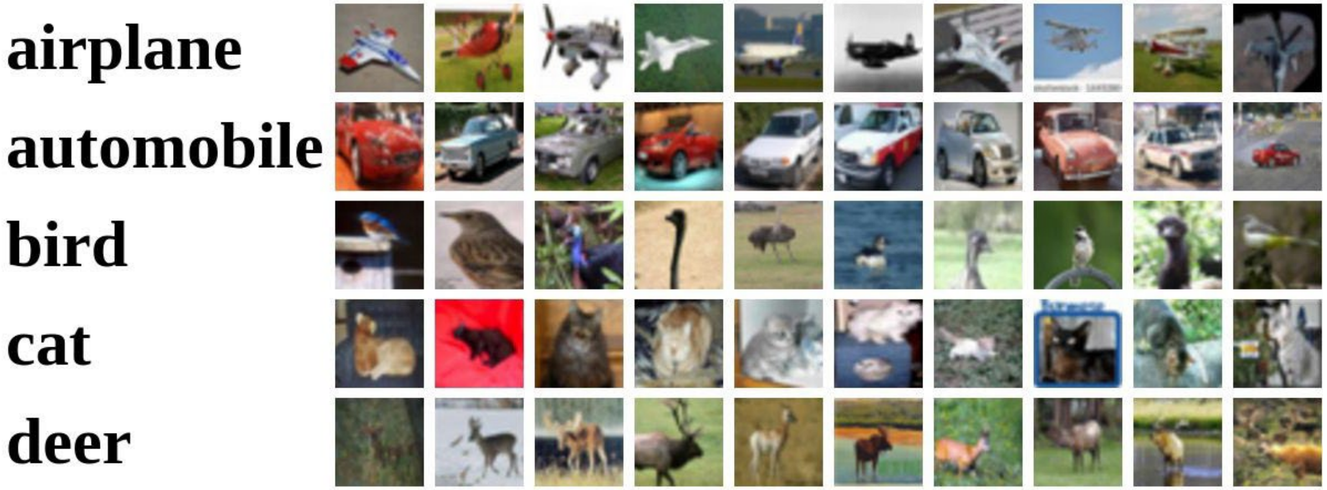
Machine Learning

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



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.

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

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

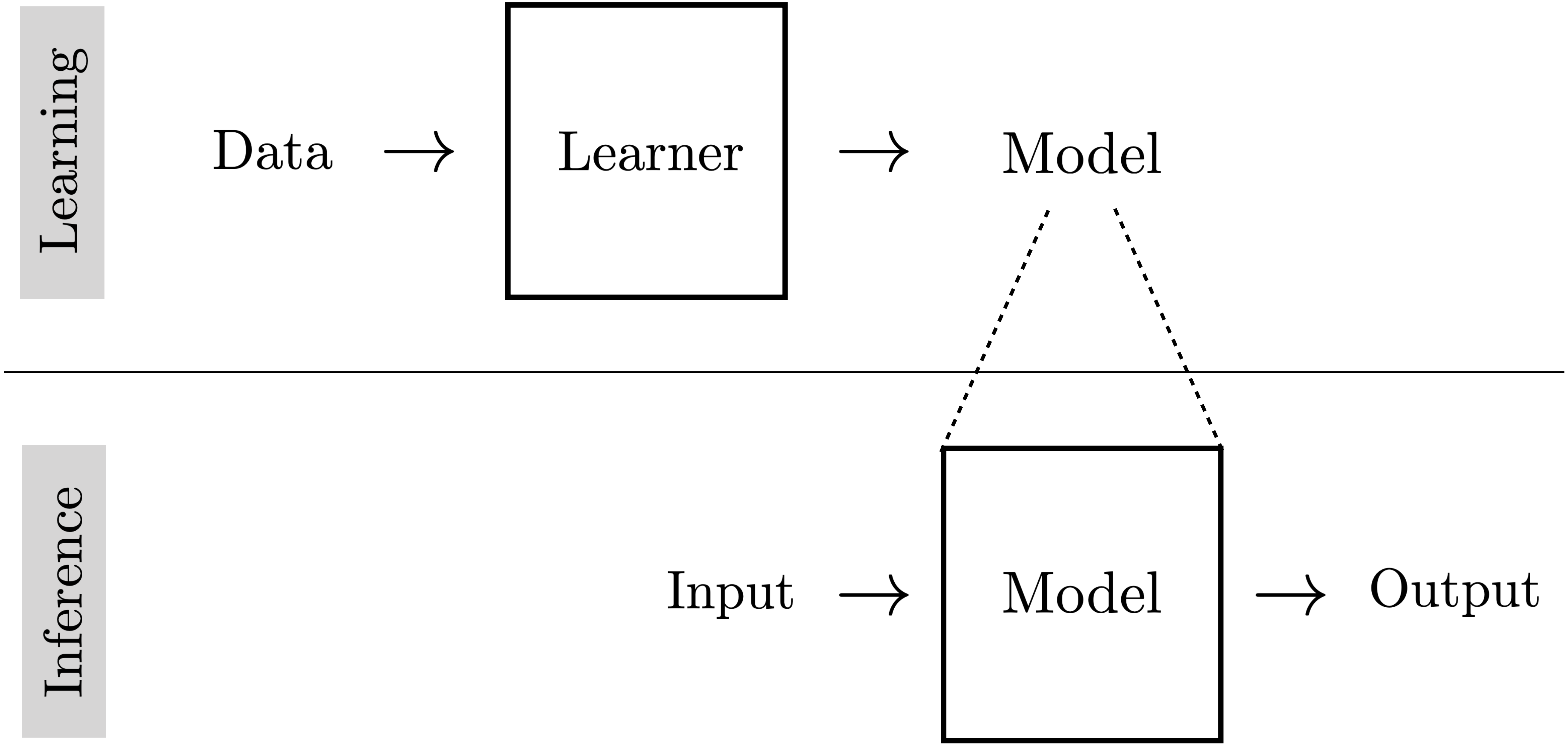
=

pixel-wise absolute value differences

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

→ 456

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


Learning
Data
Learner
Model
Inference
Input
Model
Output

Learning

Da

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

Inference

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

→ Output

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

Let's LEARN. 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

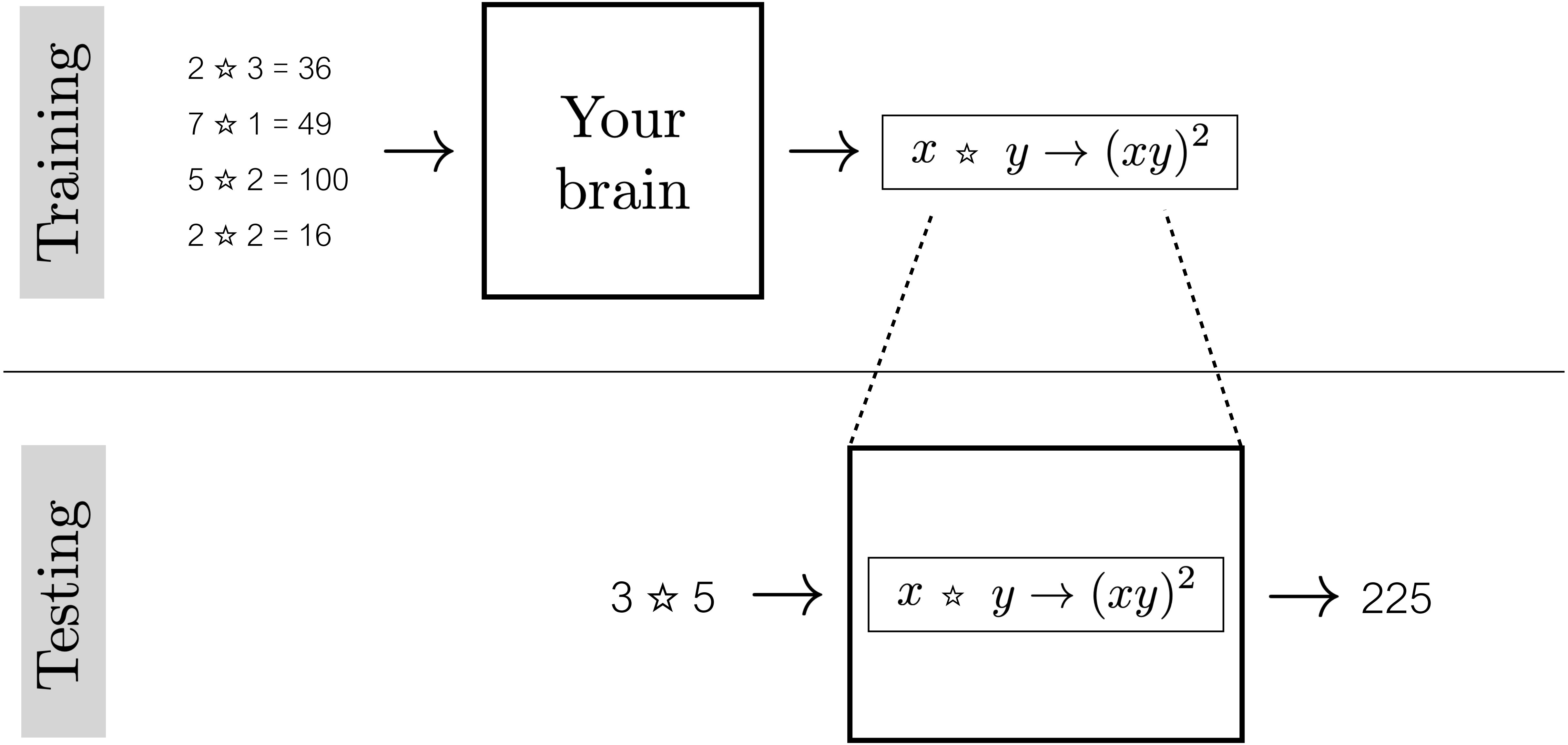
7 ☆ 1 = 49

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

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

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

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

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

$\rightarrow$ Learner $\rightarrow$ f

The goal of learning is

to extract lessons from past experience

in order to solve future problems.

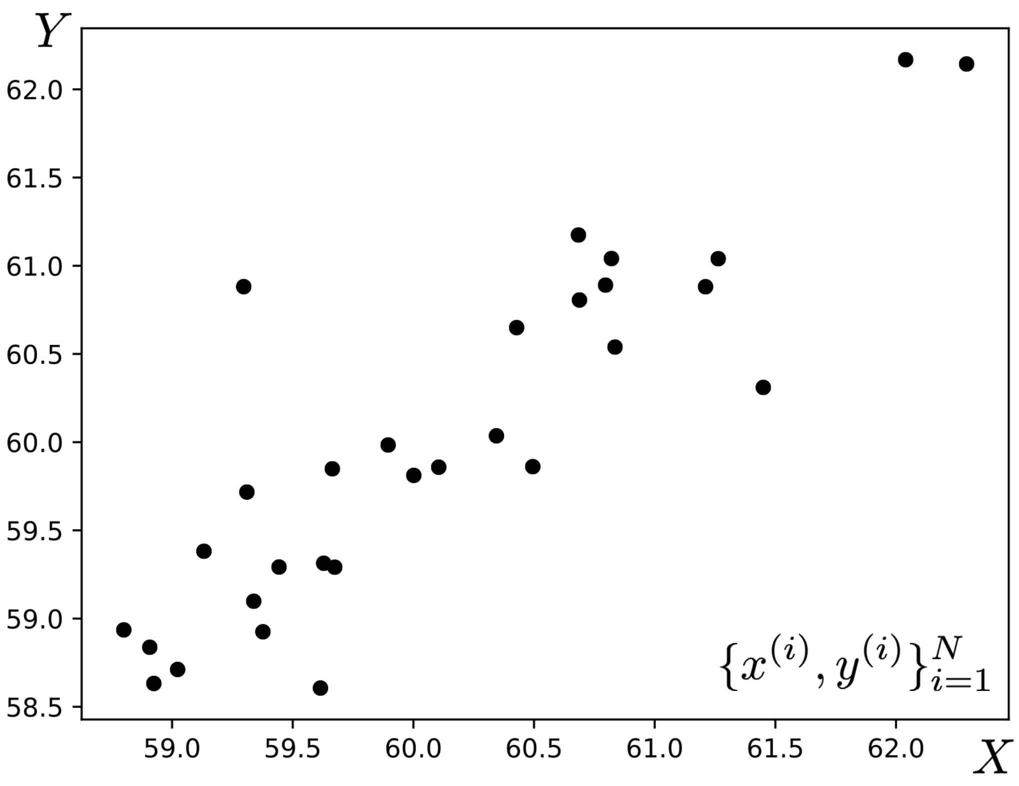
Learning from examples

(aka **supervised learning**)

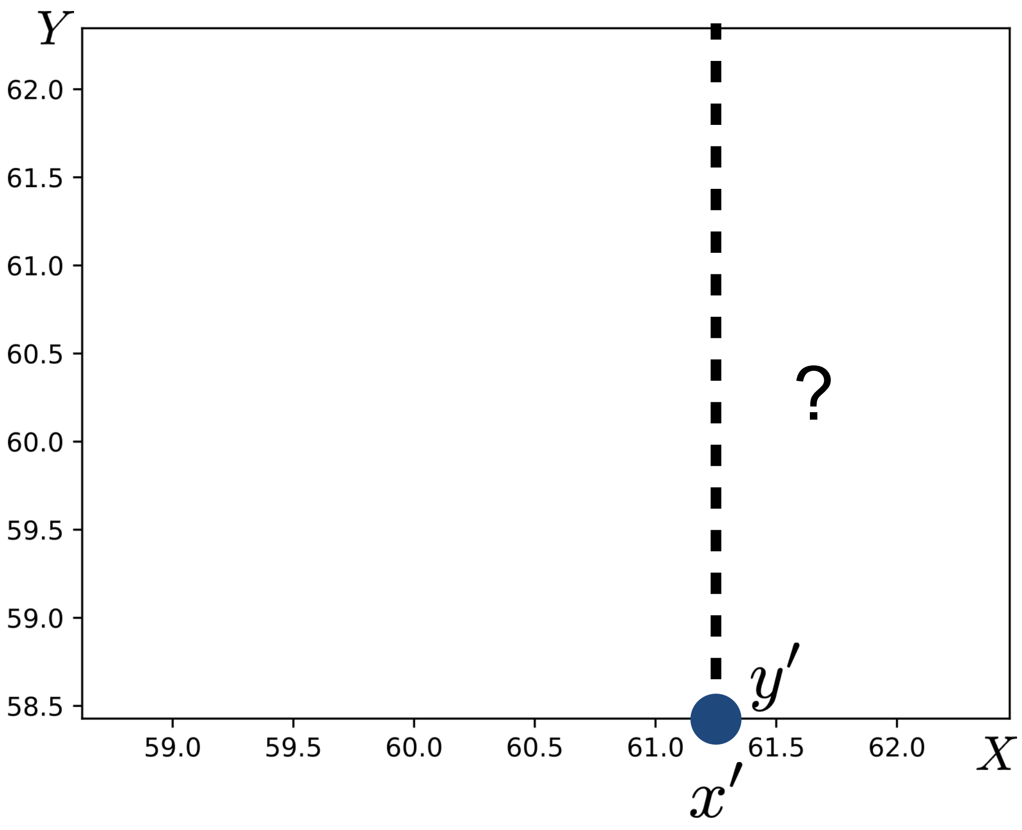
Training data

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

Training data



Test query



Real-World Application:
A Model for Predicting Electricity Use

- What will the peak power consumption be in <your-favorite-city> tomorrow?

- Difficult to answer this question without data
  - Difficult to build an "a priori" model from first principles …

- Relatively easy to record consumption history (the utility company has this data)

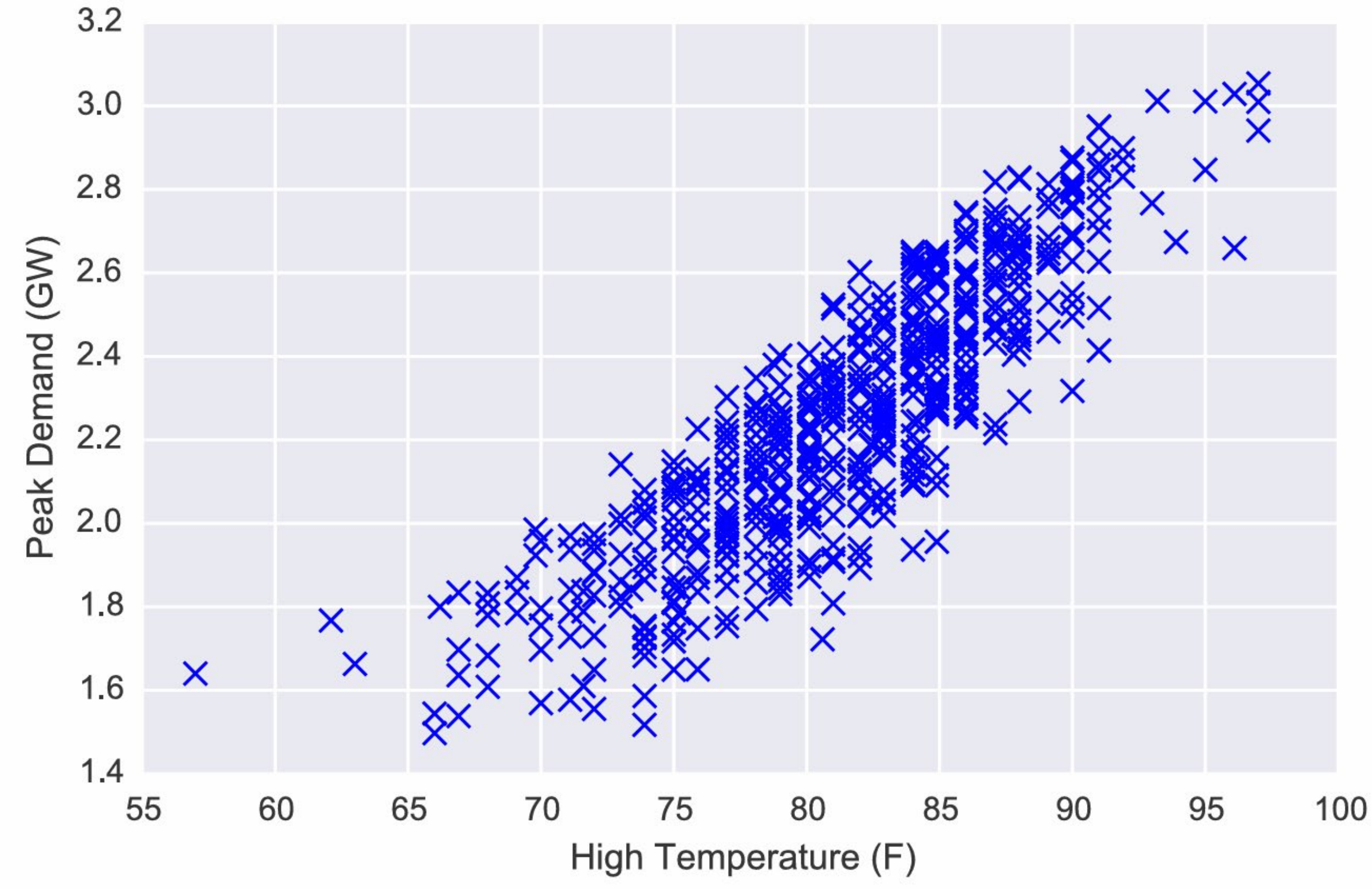
Date	High Temperature (F)	Peak Demand (GW)
2011-06-01	84.0	2.651
2011-06-02	73.0	2.081
2011-06-03	75.2	1.844
2011-06-04	84.9	1.959
…	…	…

- Relatively easy to record features that may affect consumption:
  - temperature

example from Zico Kolter

Real-World Application:
A Model for Predicting Electricity Use

- What will the peak power consumption be in <your-favorite-city> tomorrow?

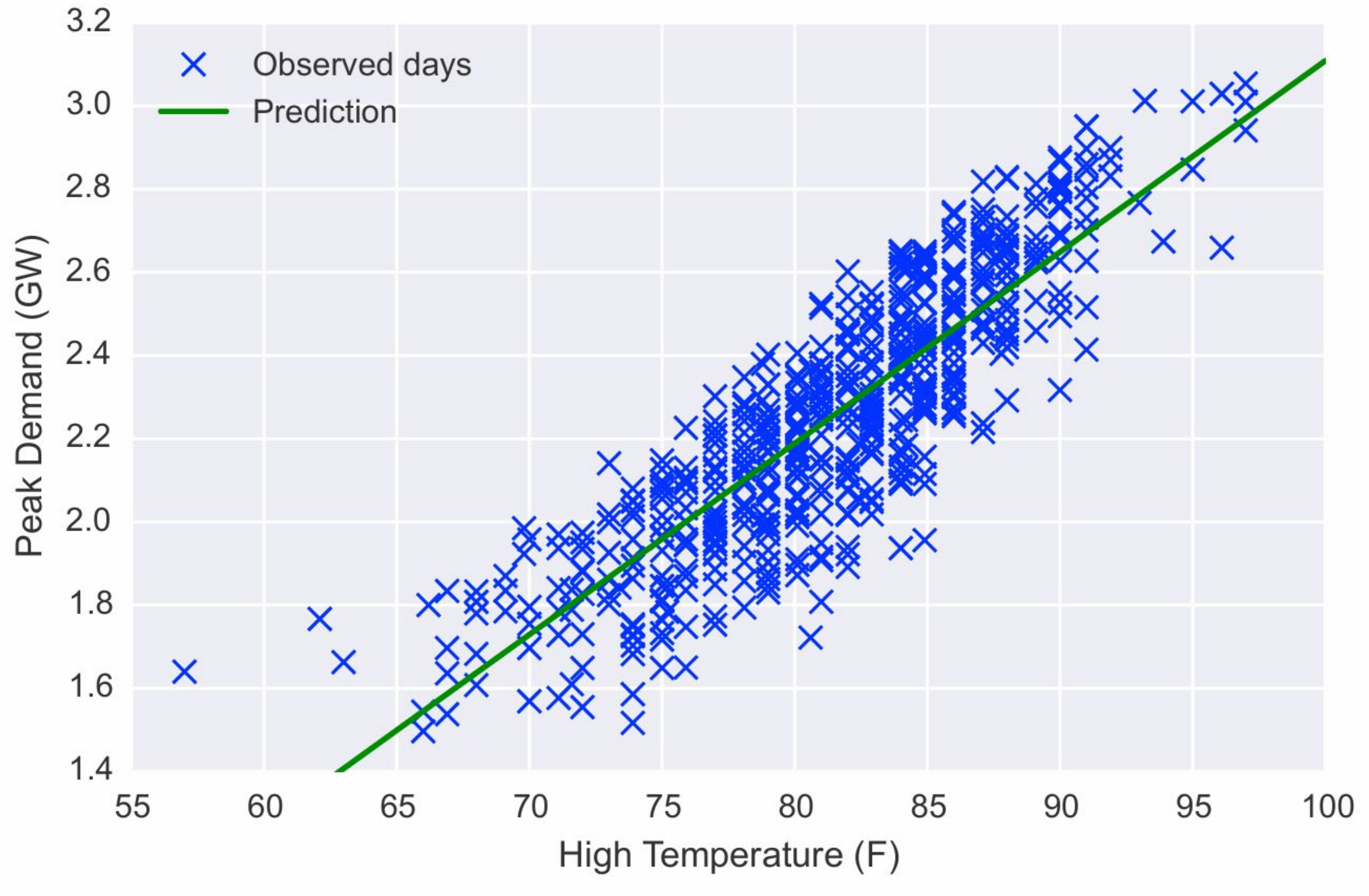


← **DATA**

example from Zico Kolter

Real-World Application:
A Model for Predicting Electricity Use

- What will the peak power consumption be in <your-favorite-city> tomorrow?



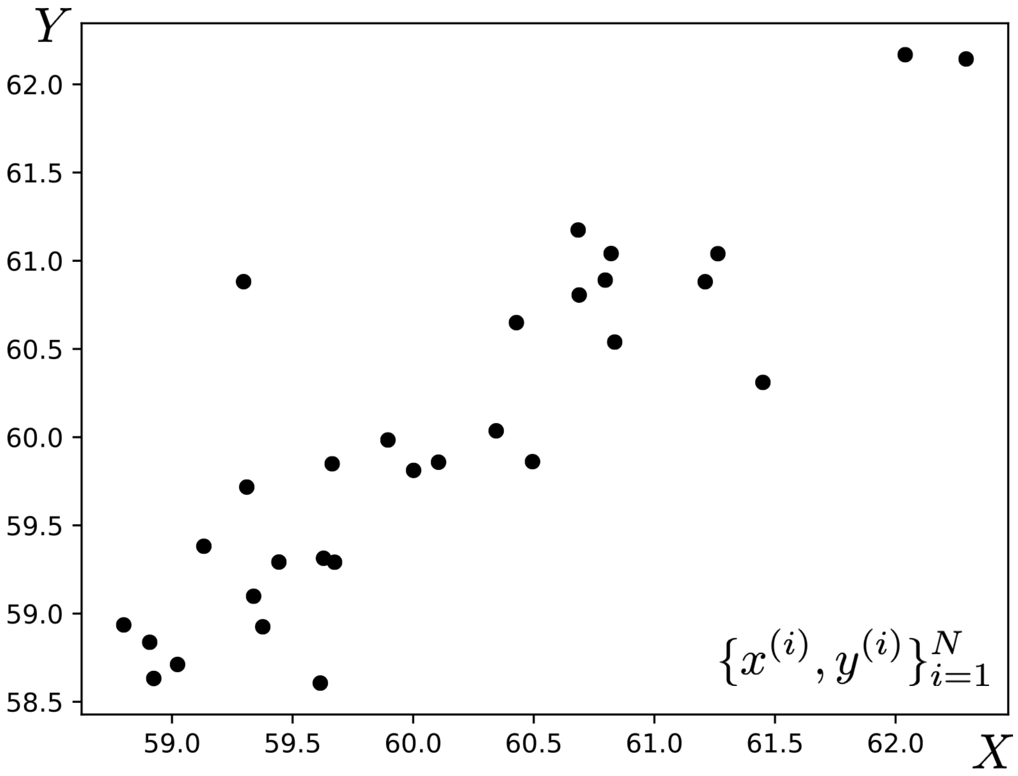
← **PREDICTION**

example from Zico Kolter

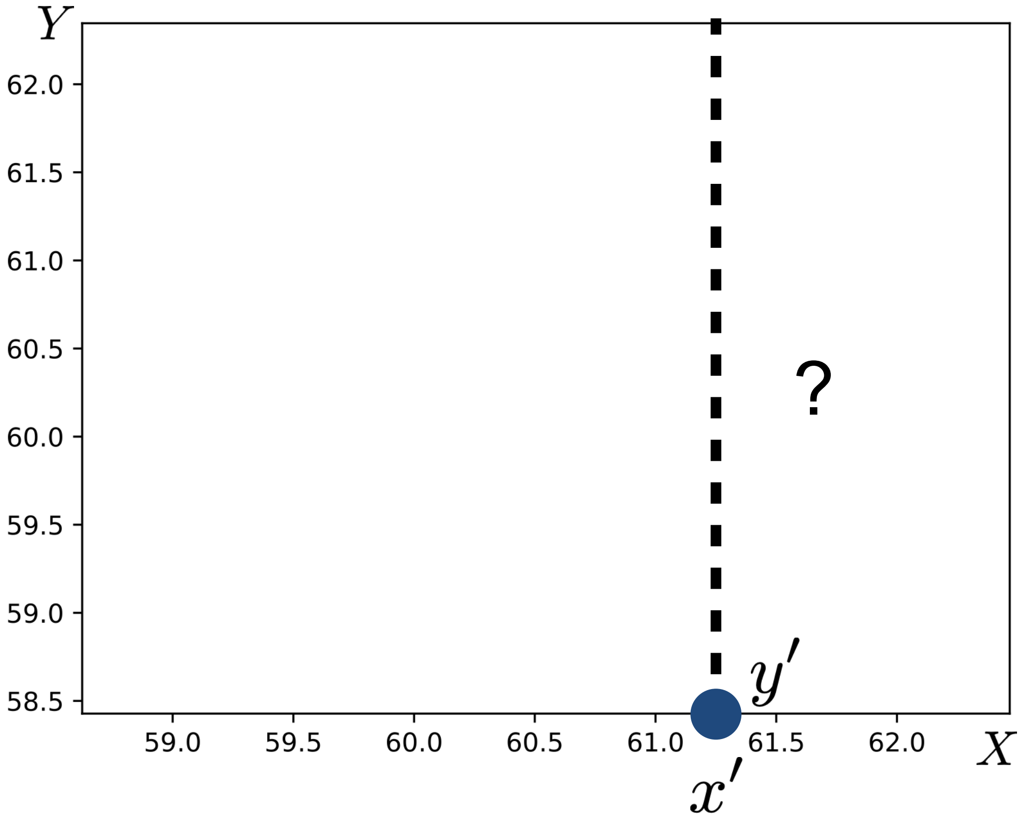
The essence of machine learning:

- A pattern exists
- We cannot pin down the pattern as an equation
- We need to approximate the pattern as a function of the input
    - Using data!
    - For vision: the pattern is in terms of pixel intensities and features.

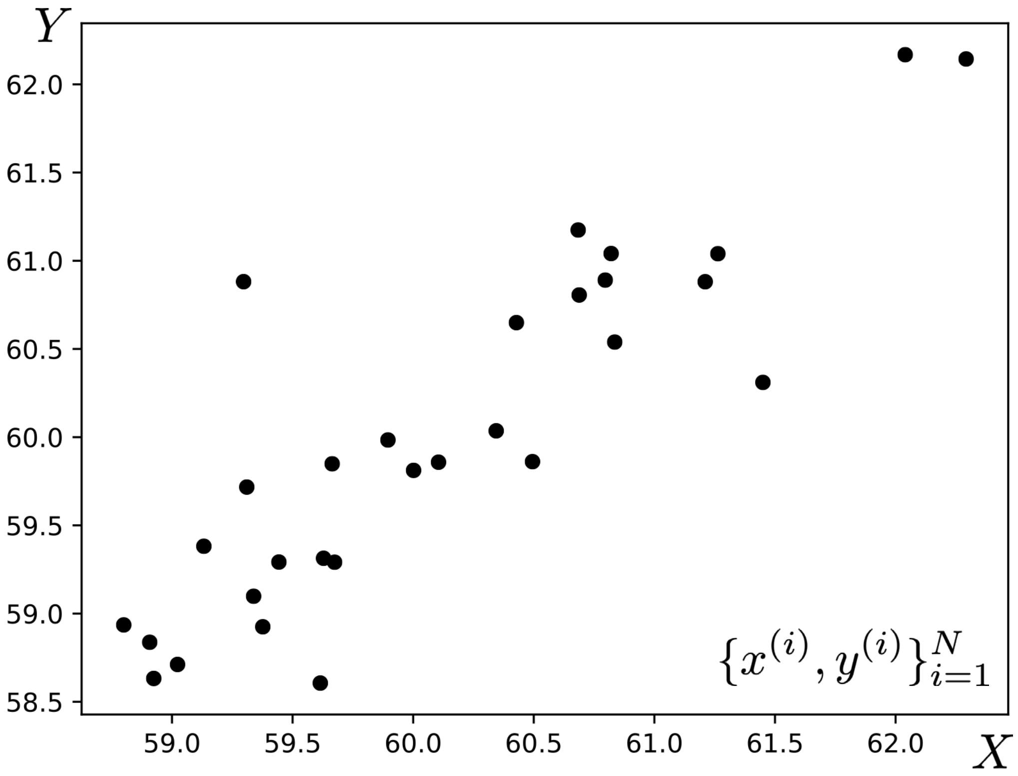
Training data



Test query



Training data

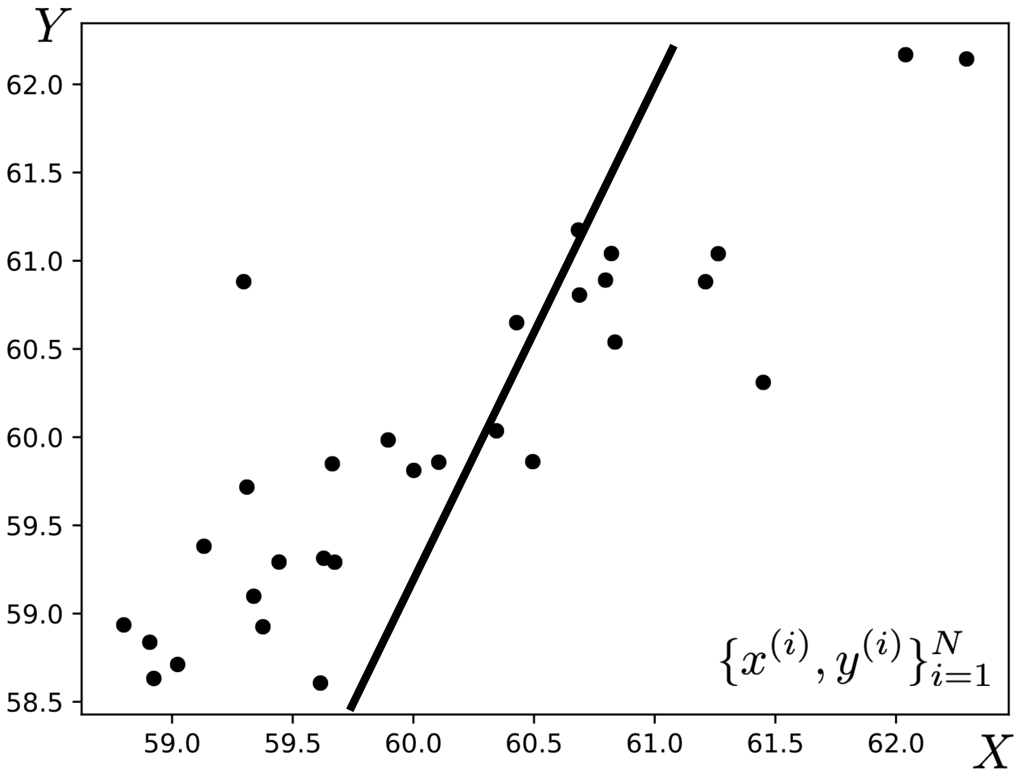


$\rightarrow$ Learner $\rightarrow$ $f_\theta(x) = \theta_1 x + \theta_0$

Hypothesis space

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

Training data

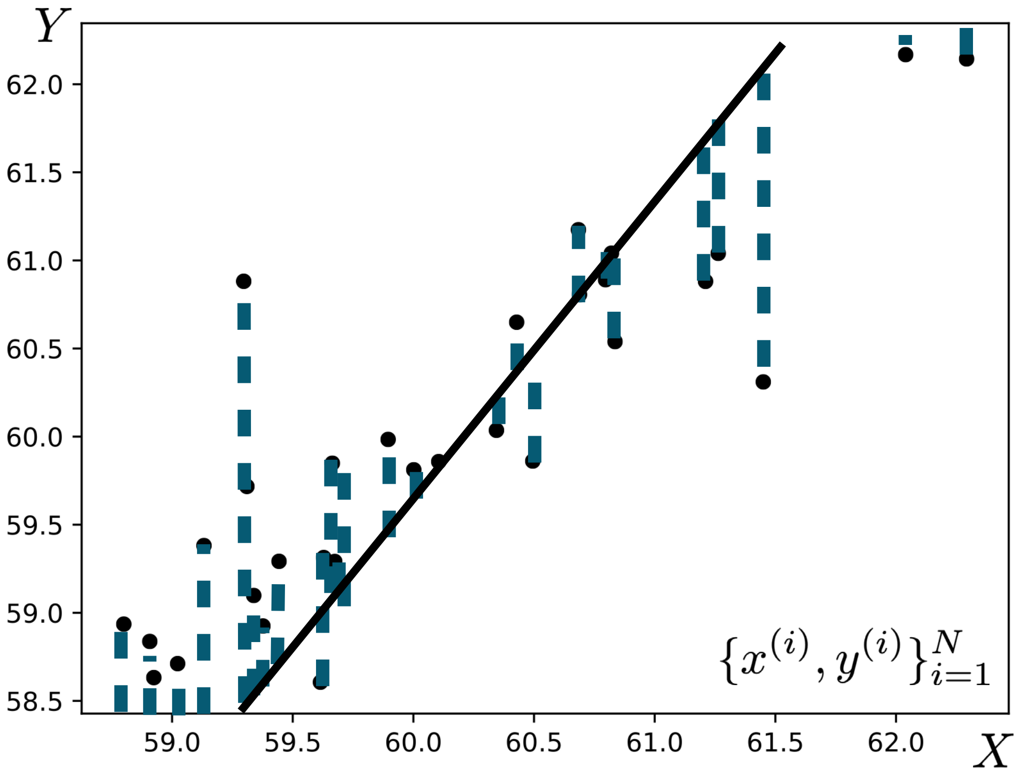


Search for the **parameters**, $\theta = \{\theta_0, \theta_1\}$, that best fit the data.

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

Best fit in what sense?

Training data



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

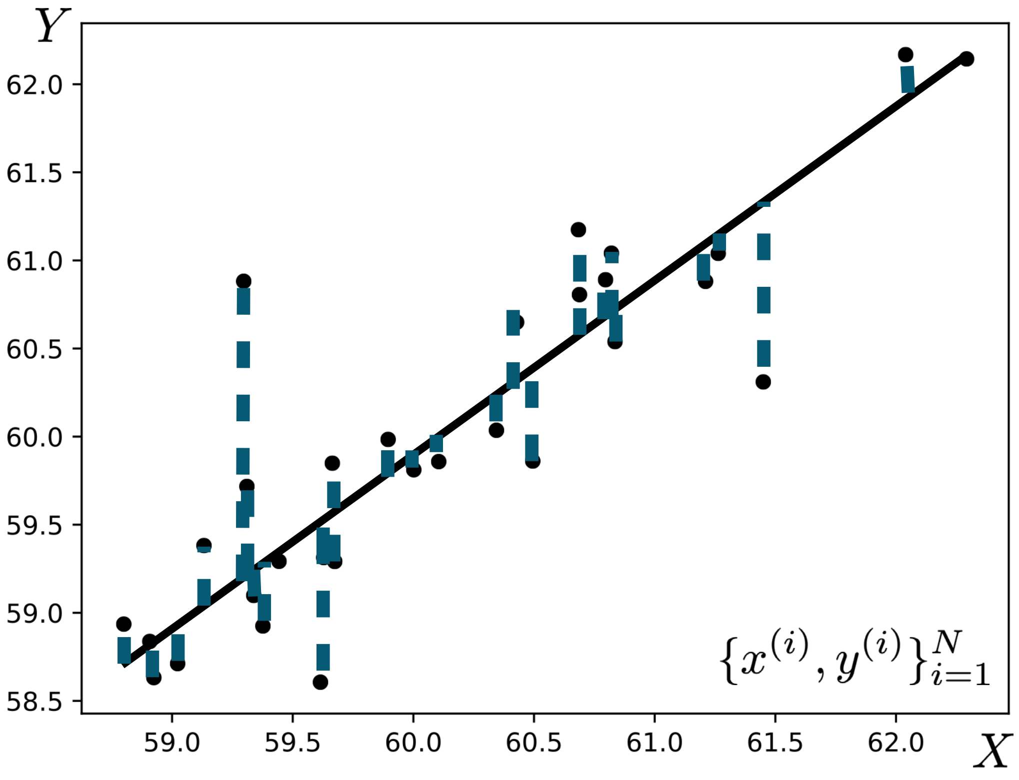
$$f_\theta(x) = \theta_1 x + \theta_0$$

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:

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

Training data



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

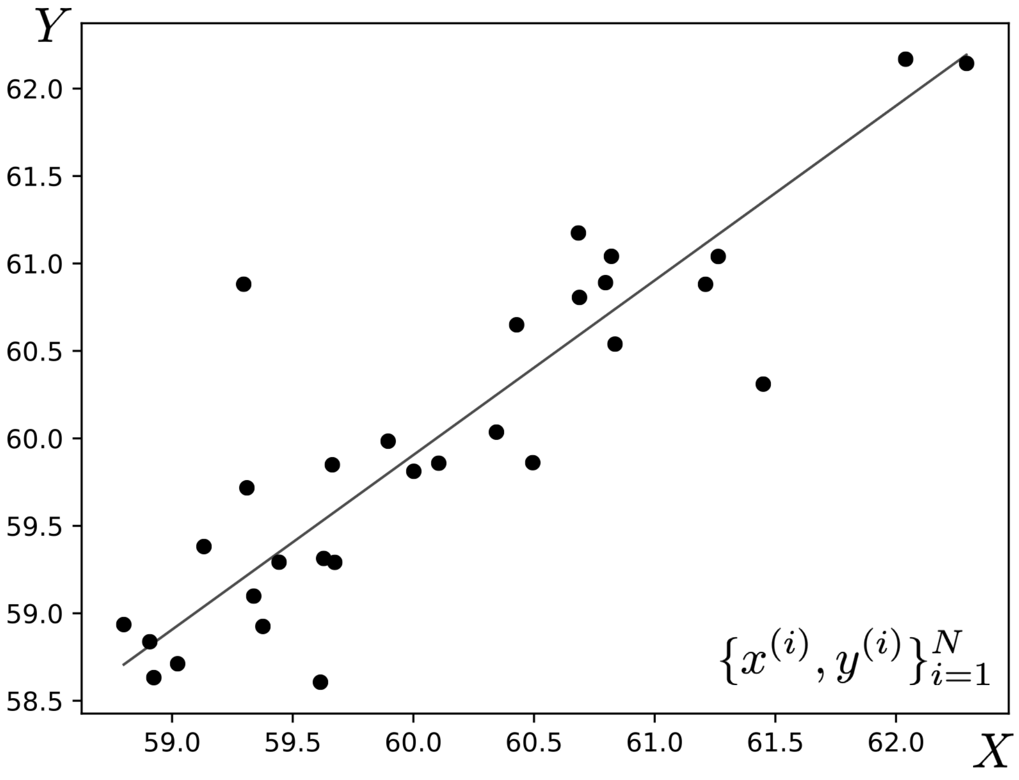
$$f_\theta(x) = \theta_1 x + \theta_0$$

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:

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

Training data

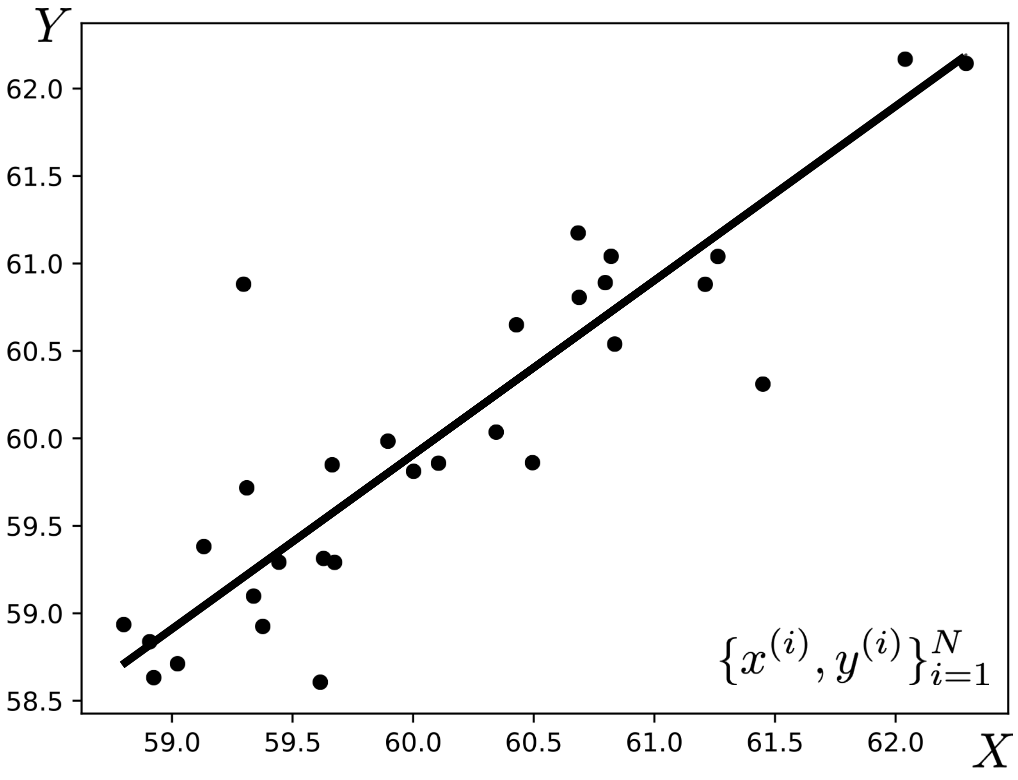


Complete learning problem:

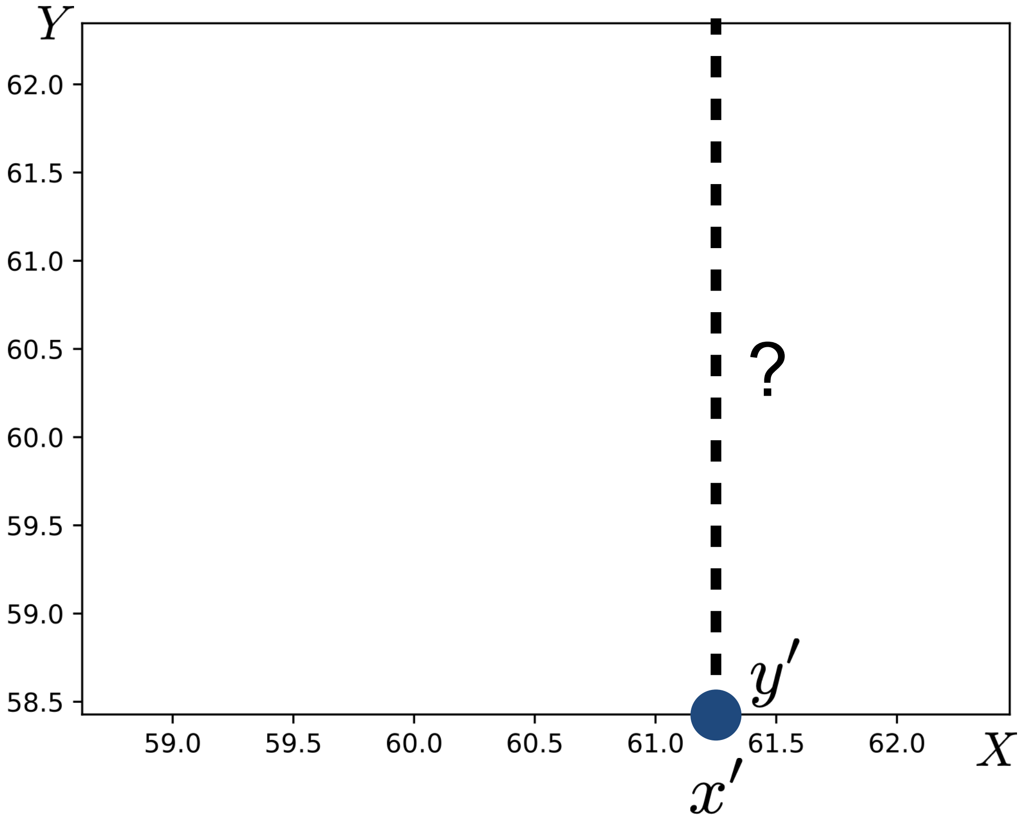
$$\theta^* = \arg\min_{\theta} \sum_{i=1}^{N} (f_\theta(x^{(i)}) - y^{(i)})^2$$

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

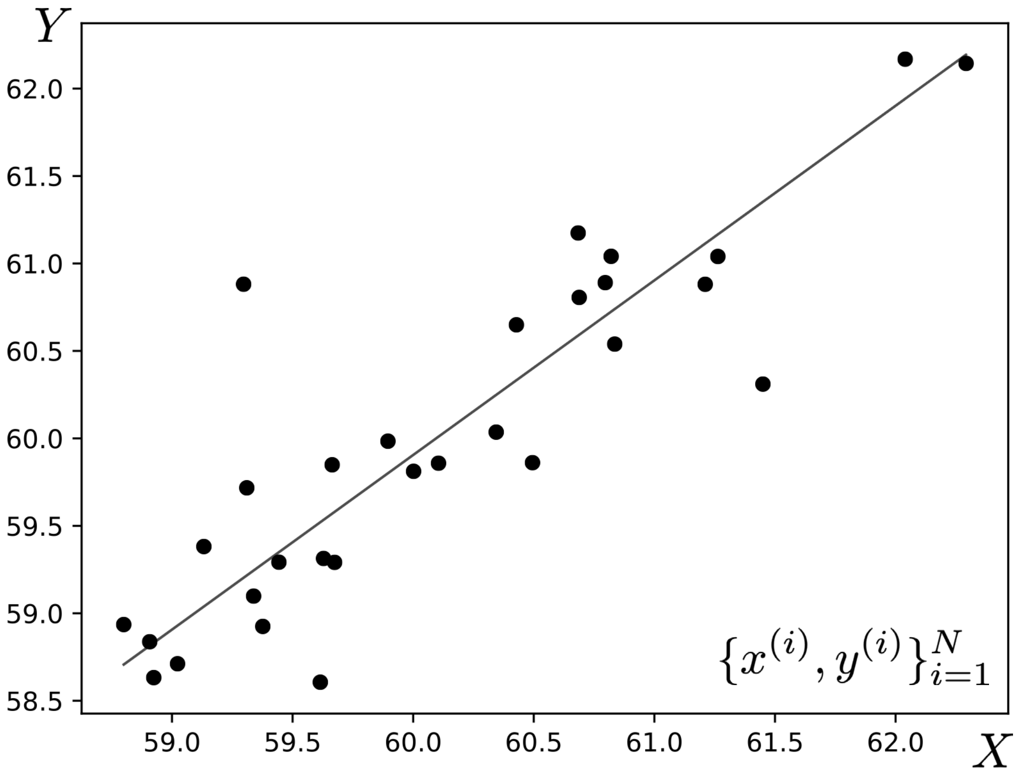
Training data



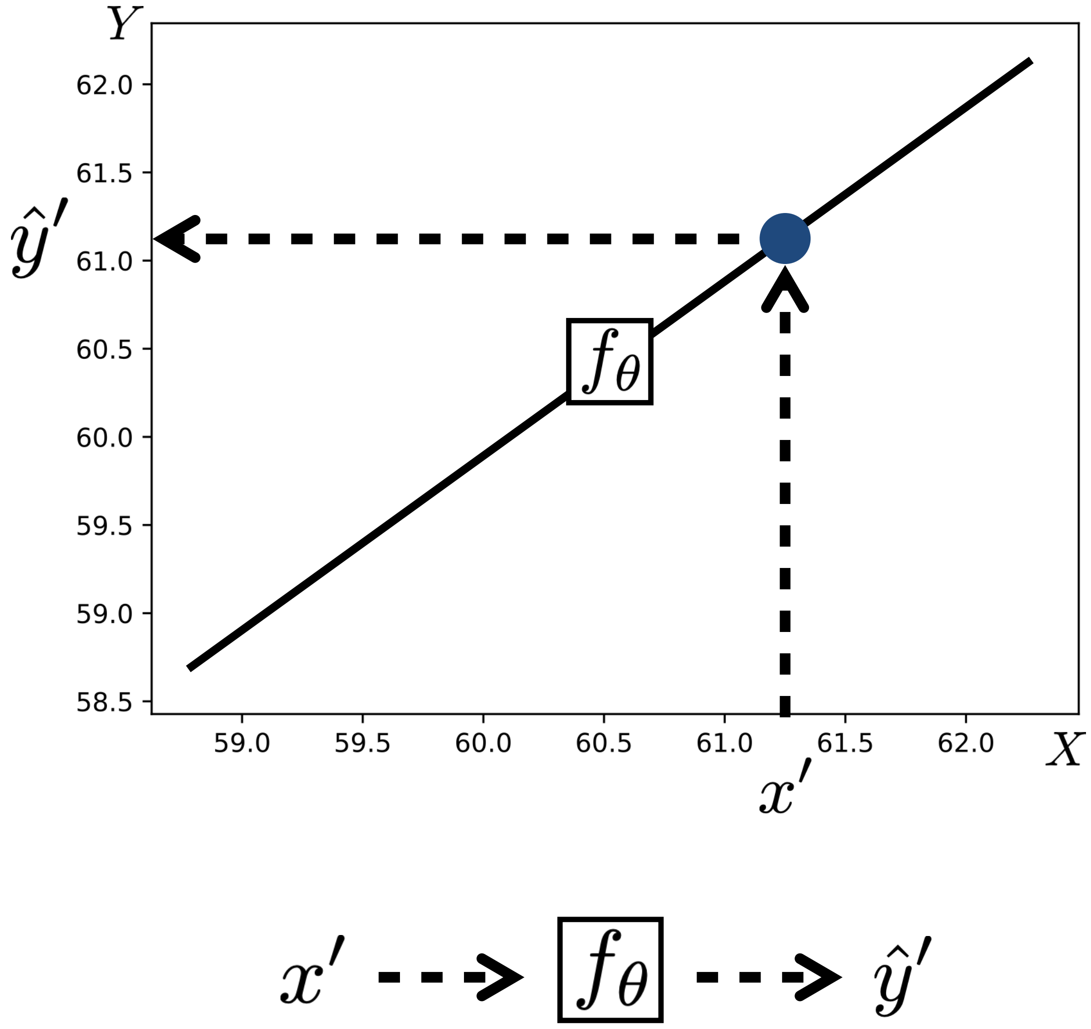
Test query



Training data



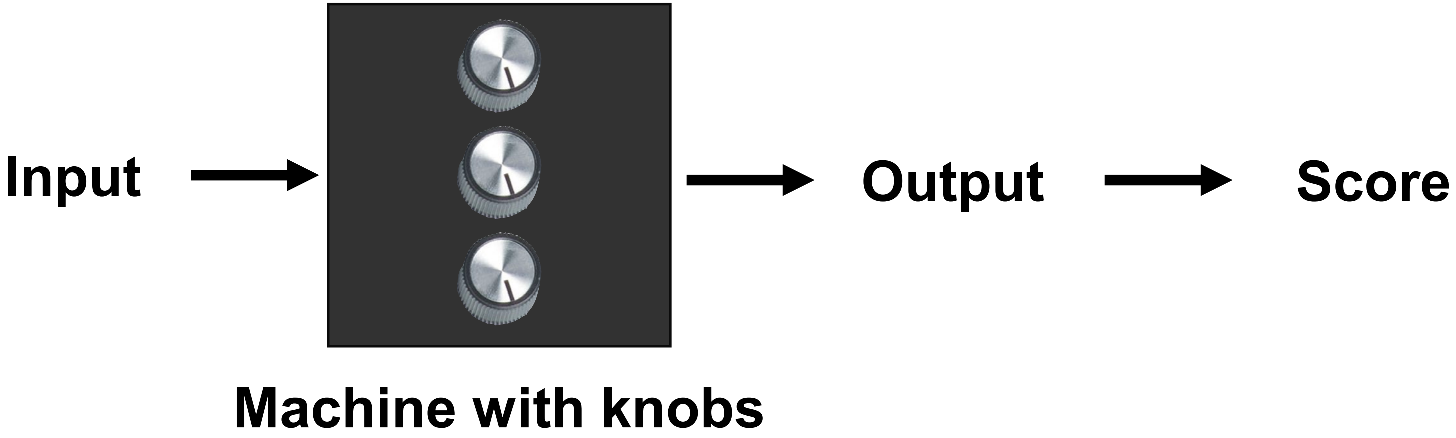
Test query

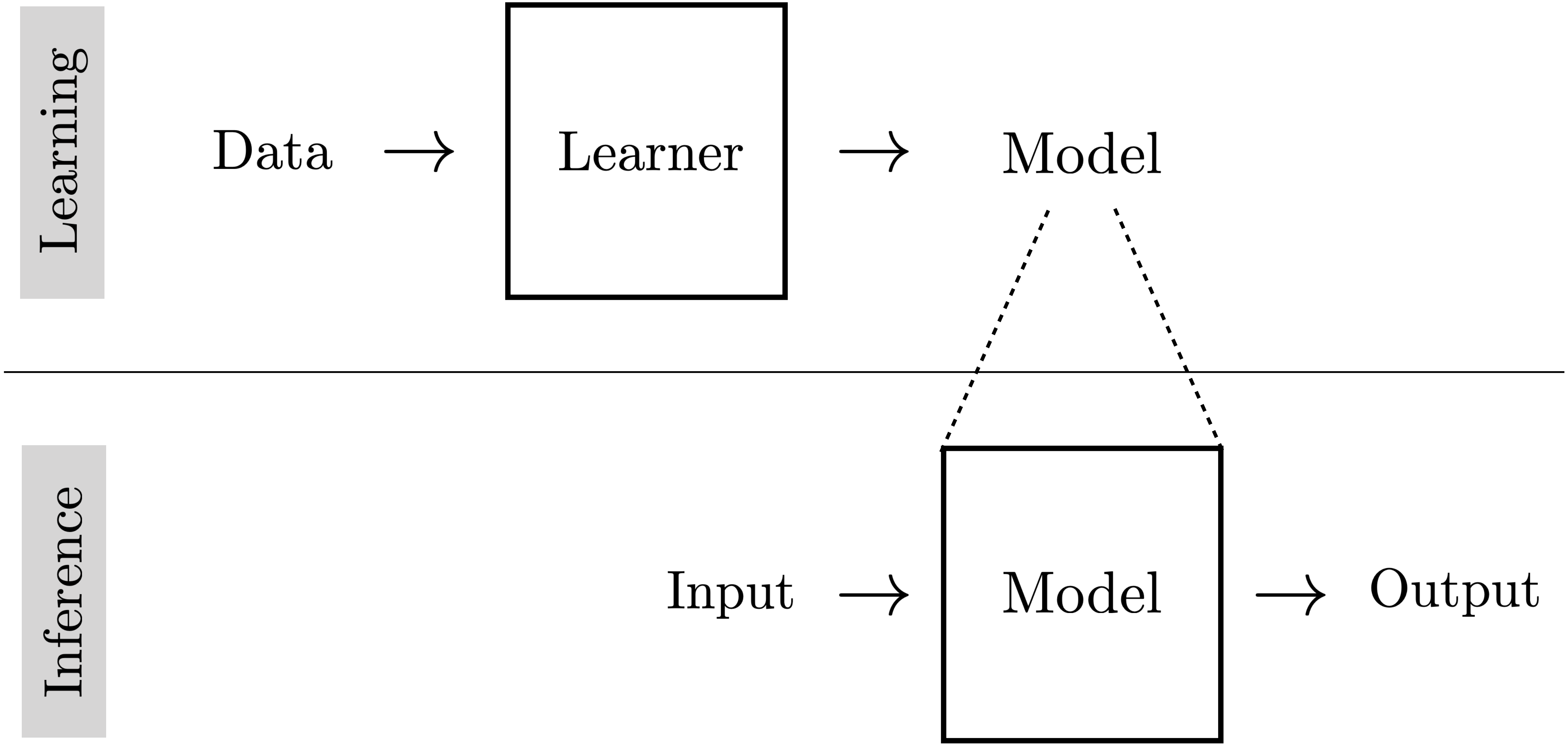


How to minimize the objective w.r.t. θ?

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

Use an **optimizer**!




Learning
Data
Learner
Model
Inference
Input
Model
Output

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

In the linear case:

Learning problem

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

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

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

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

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

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

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

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

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

Solution

Empirical Risk Minimization

(formalization of supervised learning)

Linear least squares learning problem

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

Empirical Risk Minimization

(formalization of supervised learning)

Objective function
(loss)

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

Training data

Hypothesis space

Case study #1: Linear least squares

Data

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

Learner

Objective

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

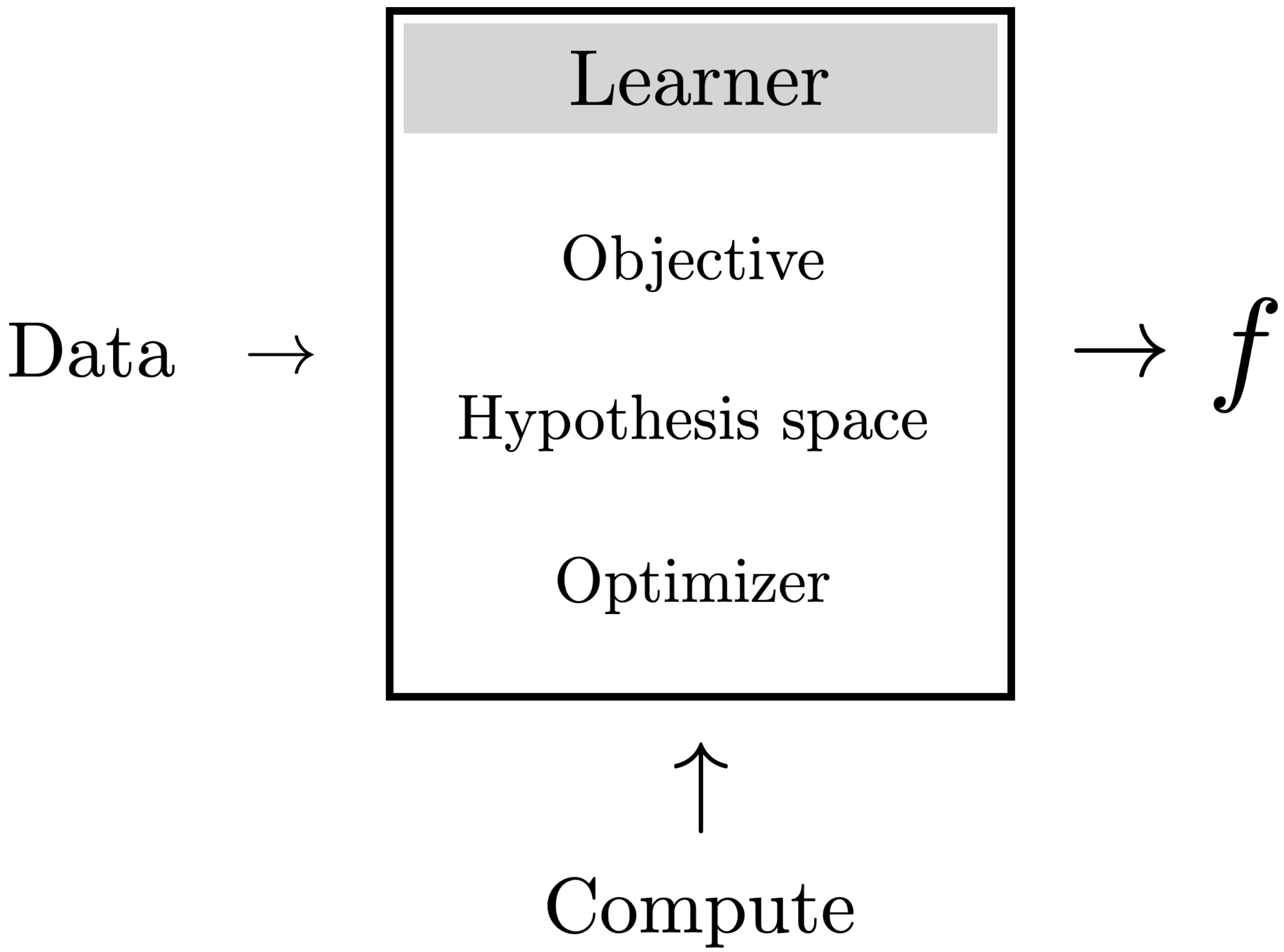
Hypothesis space

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

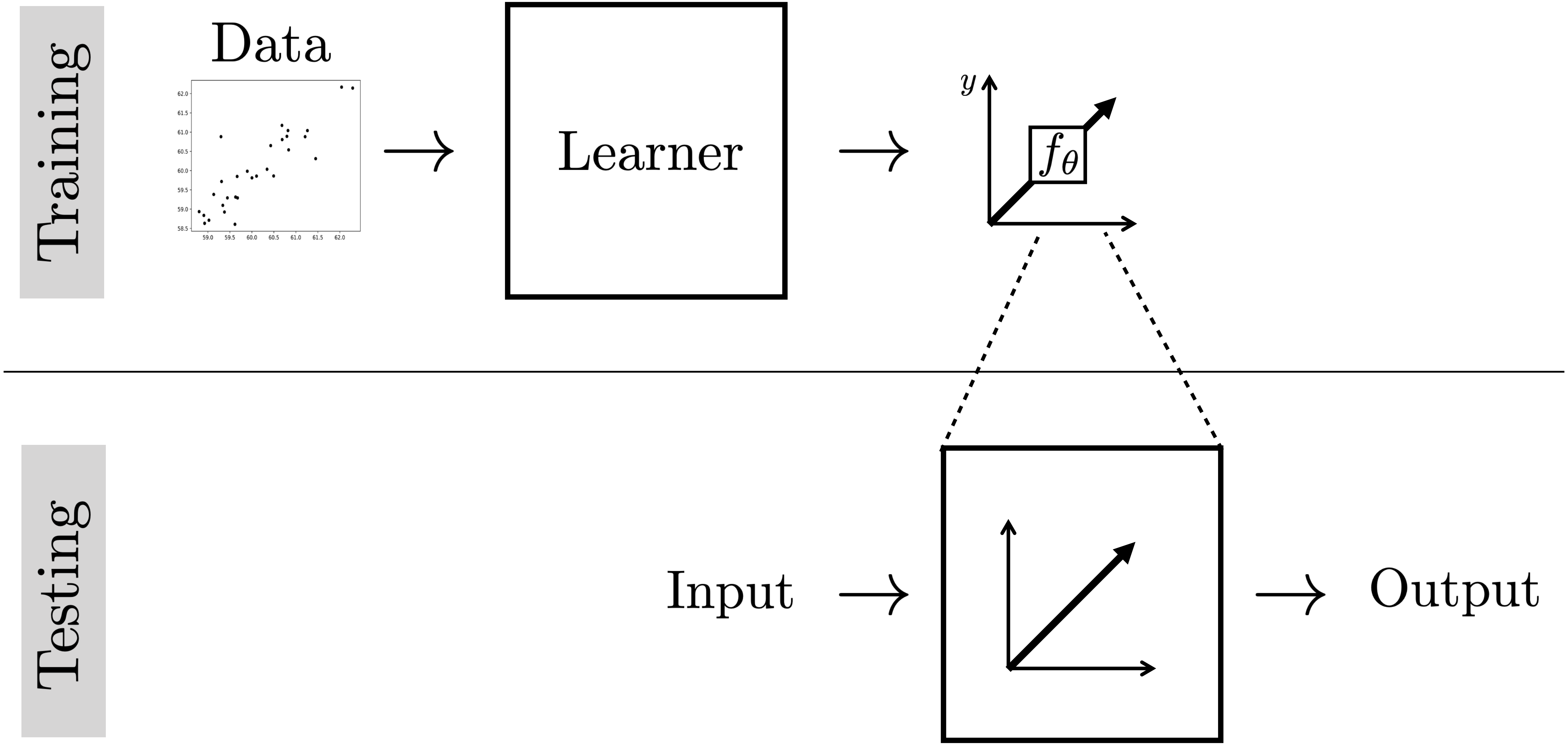
Optimizer

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

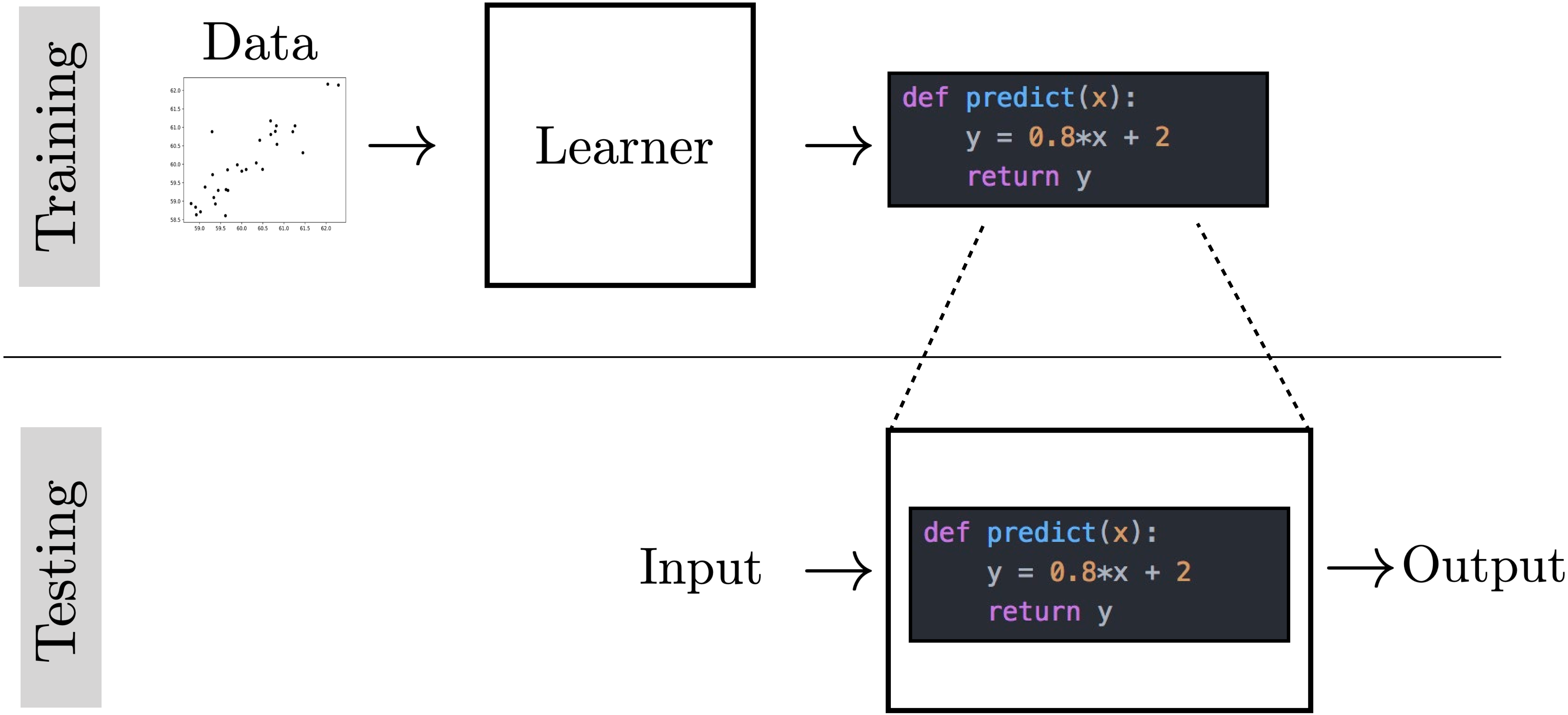
$\rightarrow f$


Learner
Objective
Hypothesis space
Optimizer
Data
f
Compute

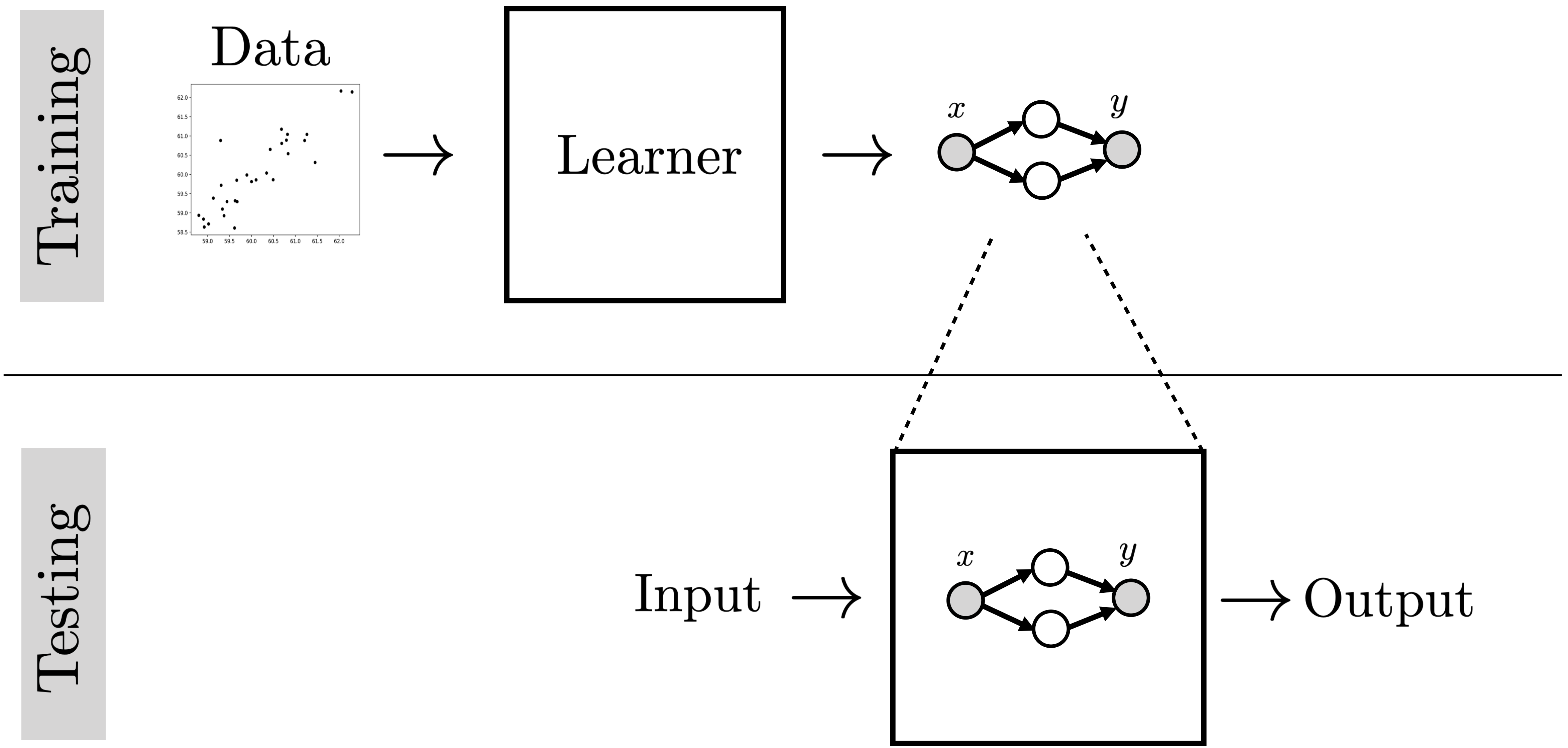
Example 1: Linear least squares

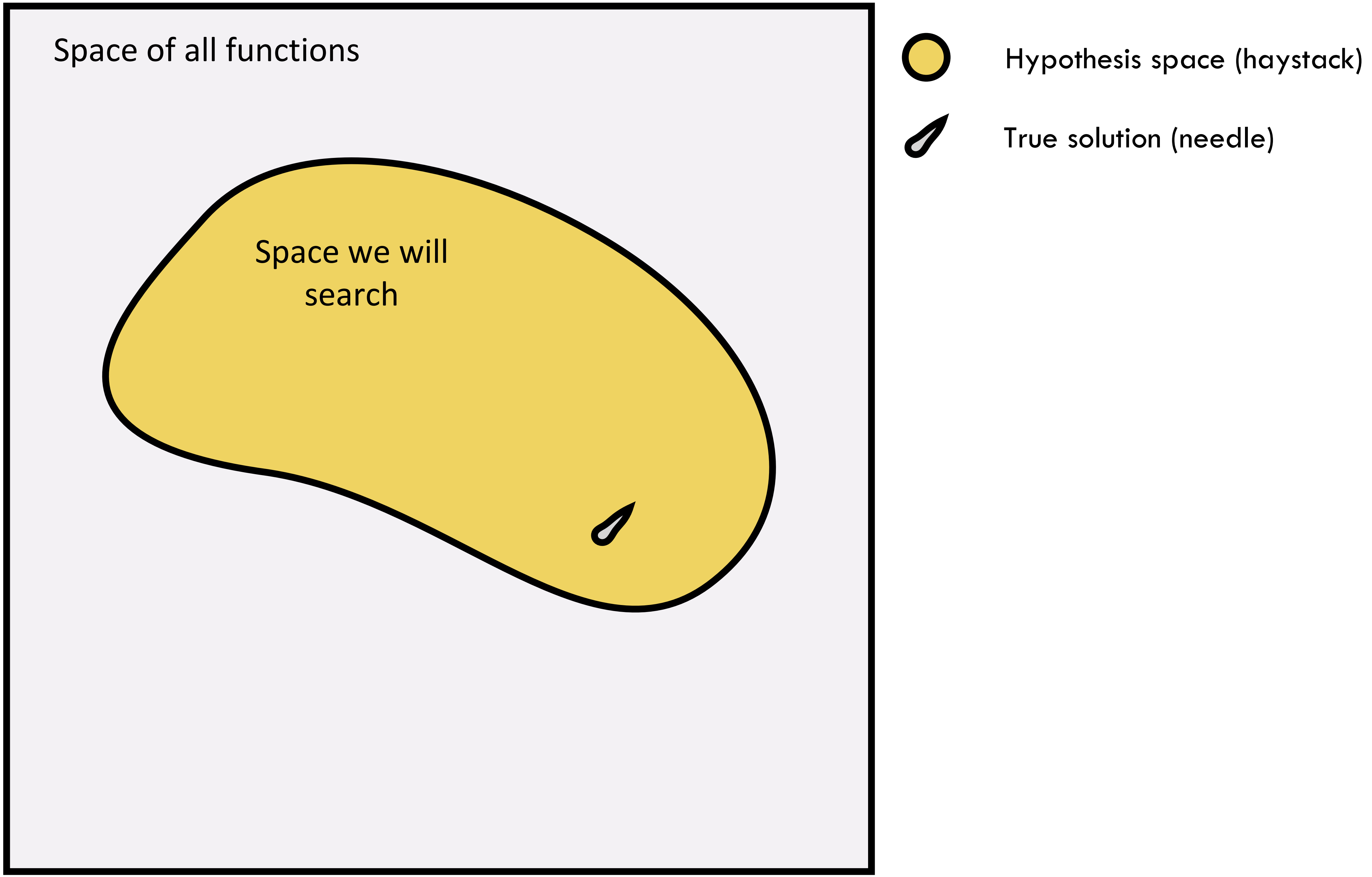


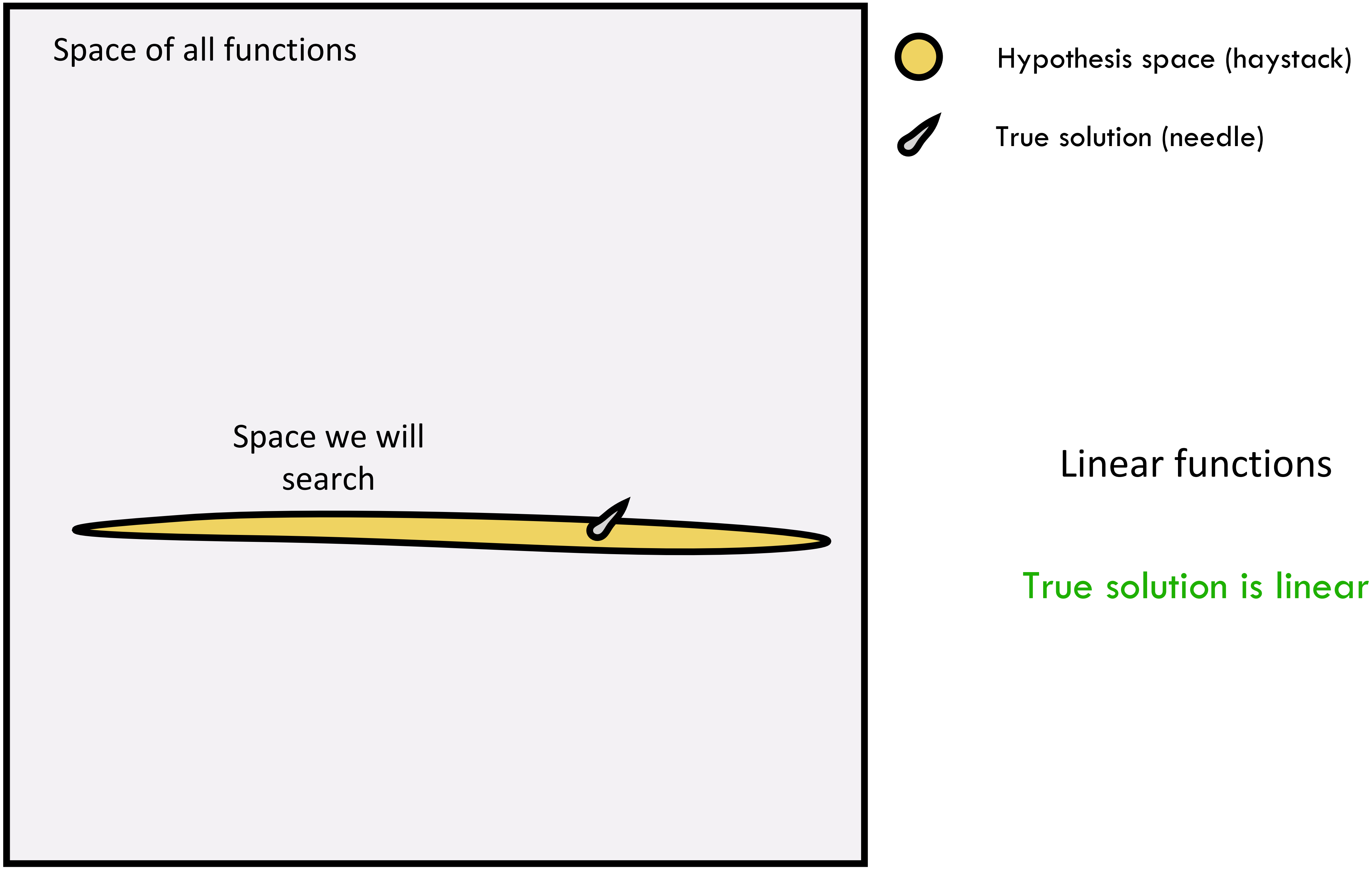
Example 2: Program Induction



Example 3: "Deep" Learning (with Neural Networks)

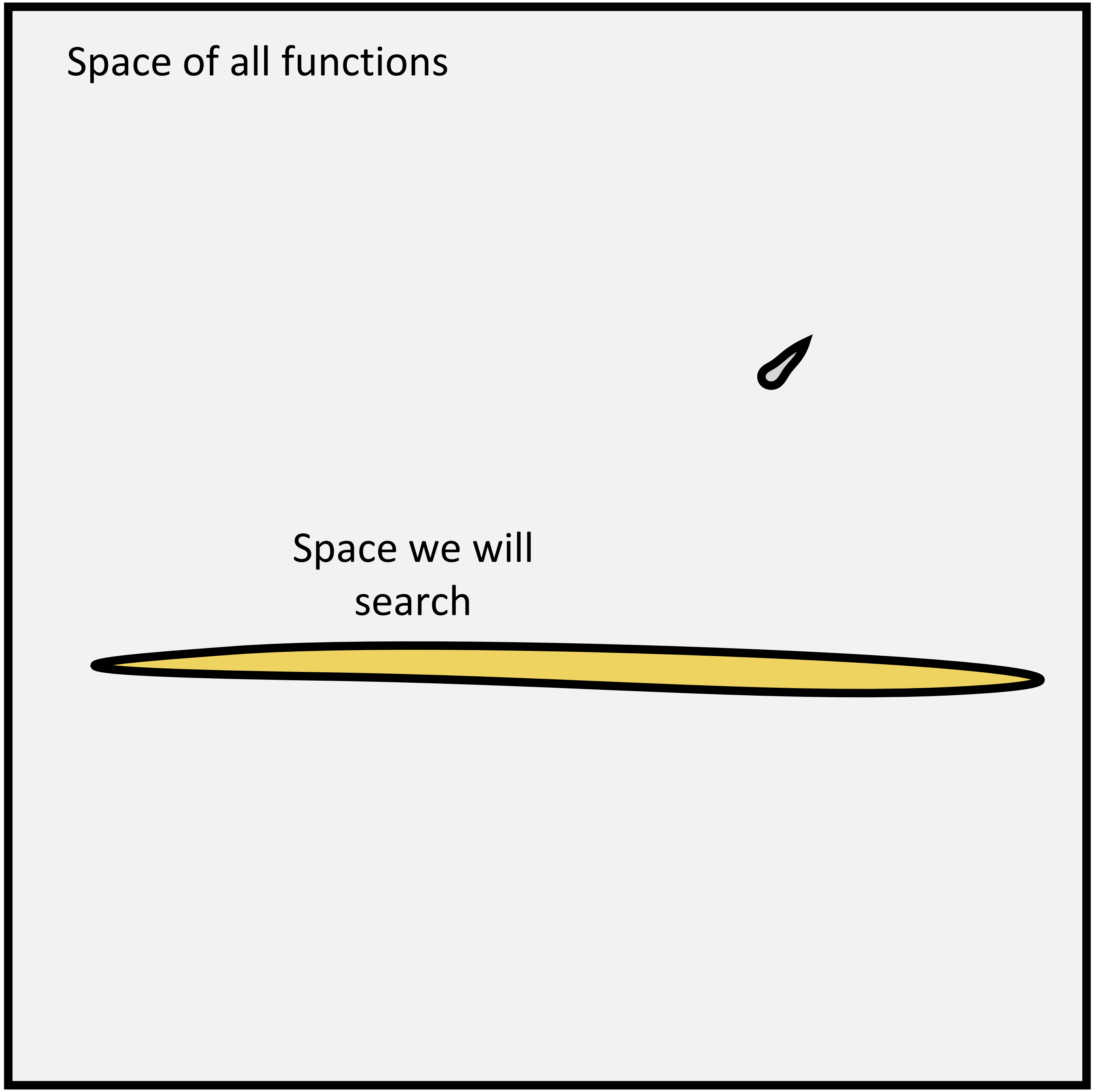


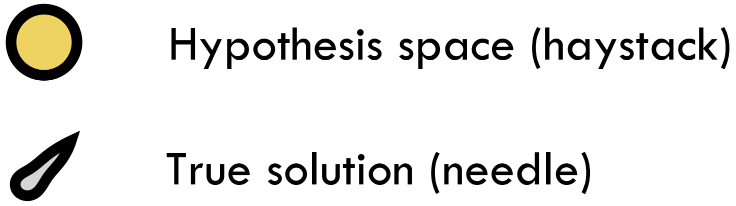

Space of all functions
Space we will search
Hypothesis space (haystack)
True solution (needle)



Linear functions

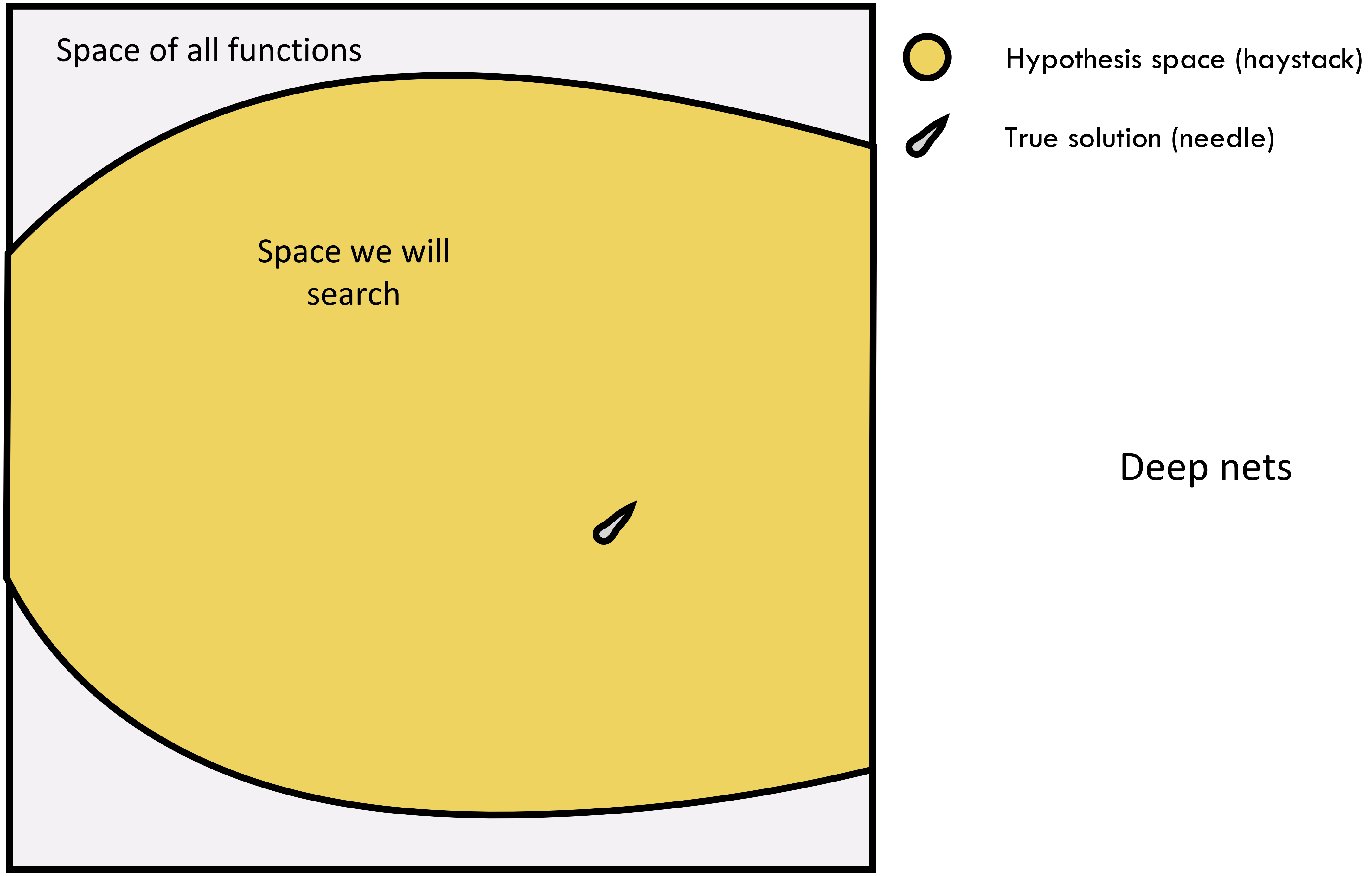
True solution is linear





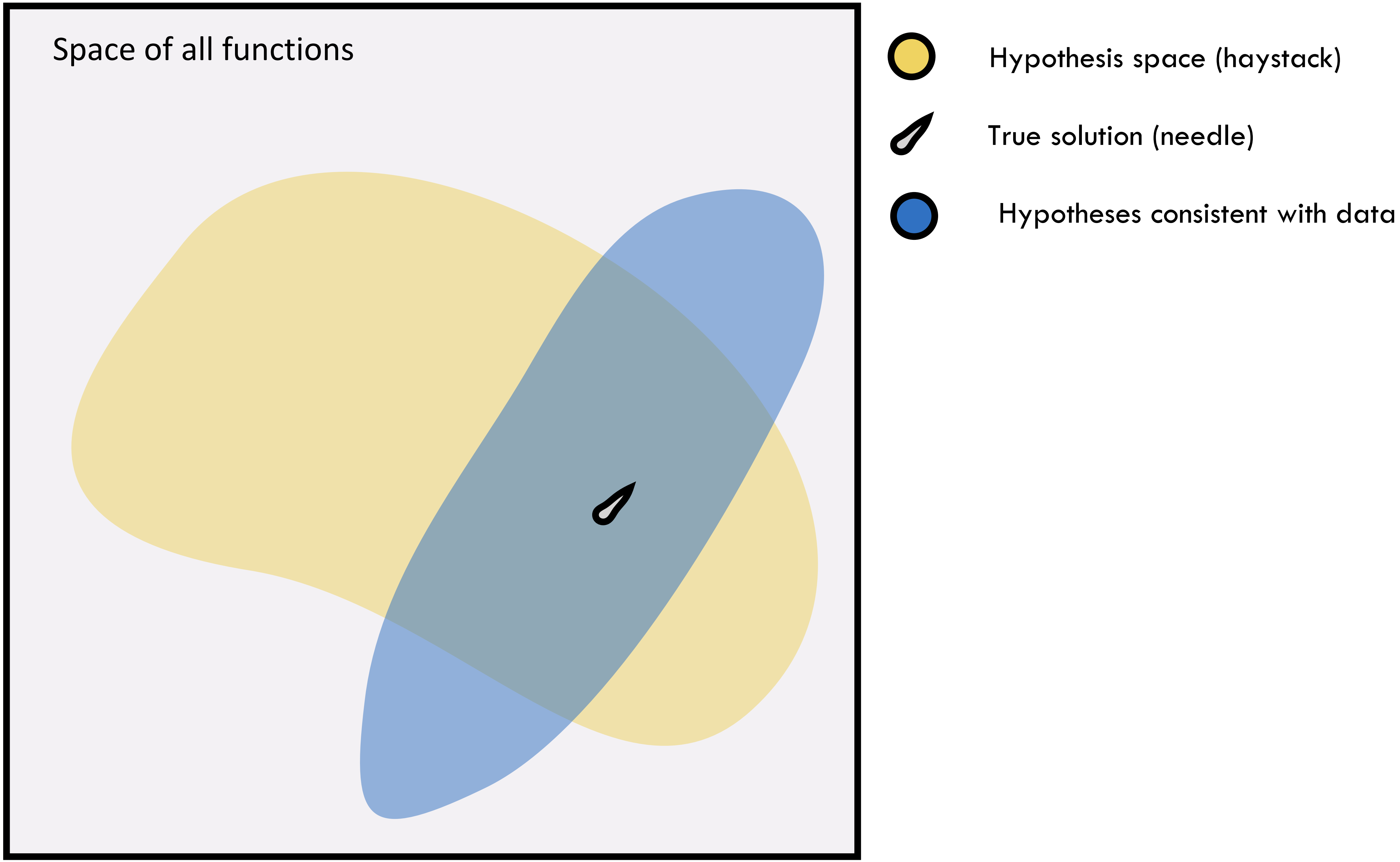
Linear functions

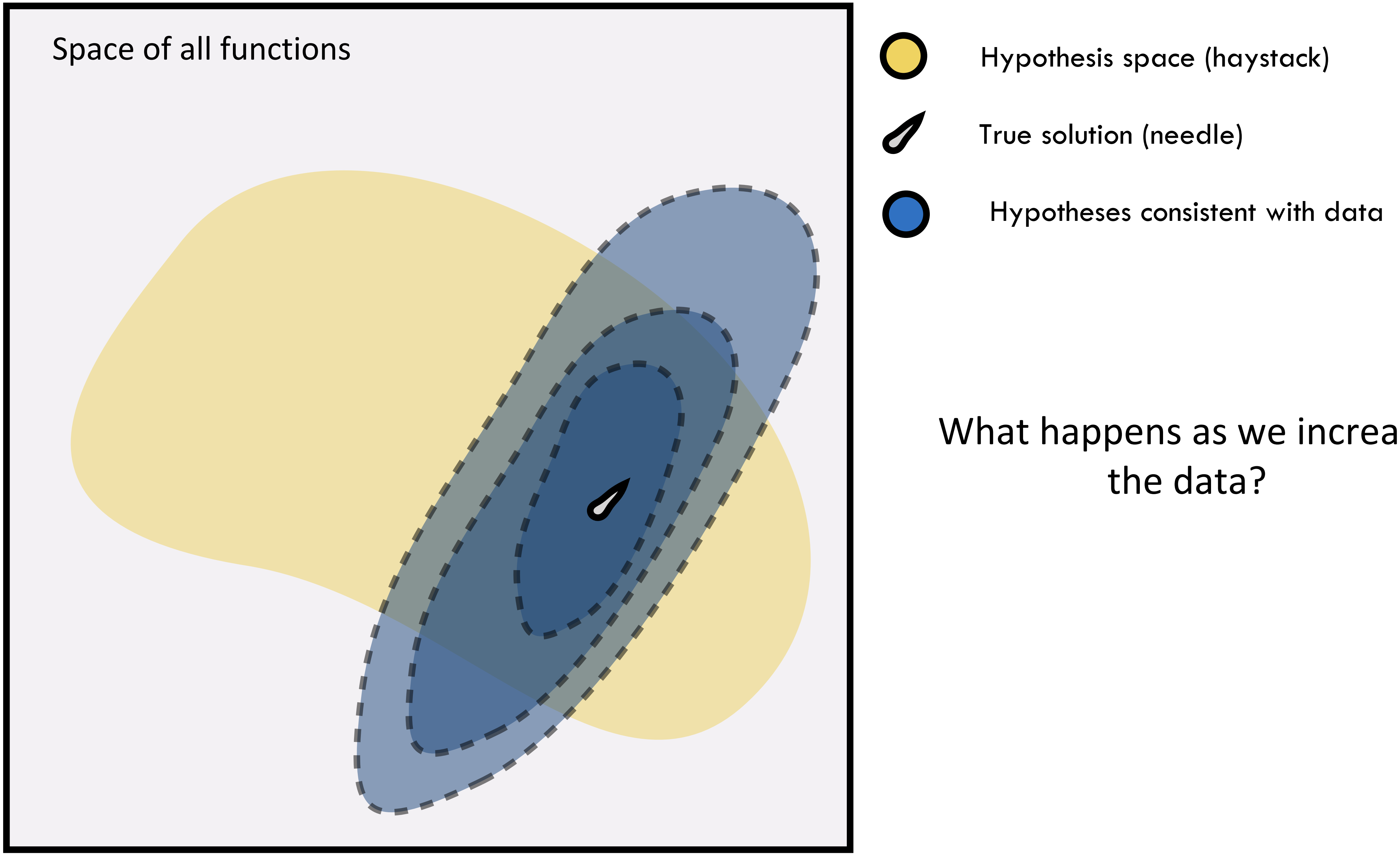
True solution is nonlinear



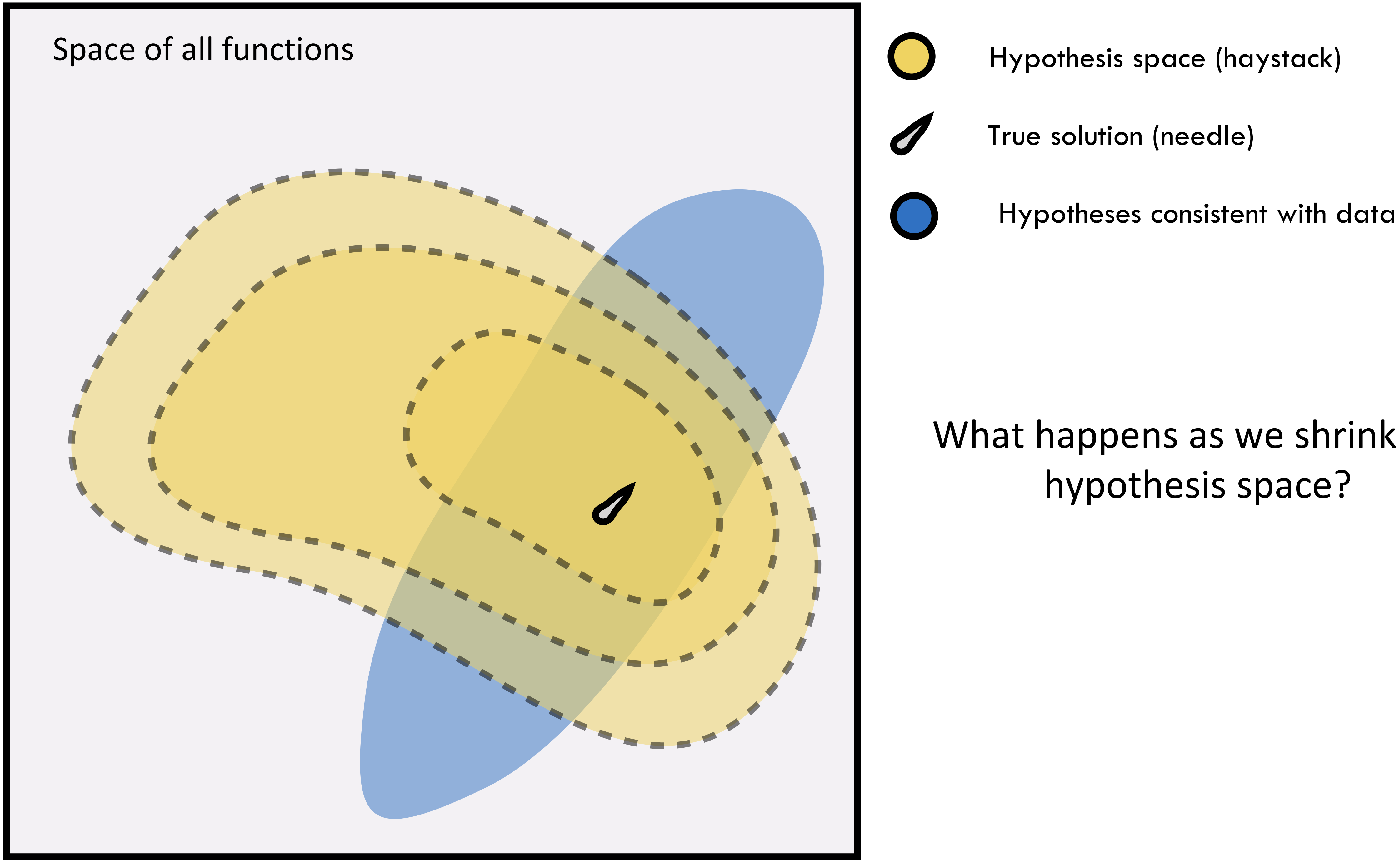
Space of all functions
Space we will search
Hypothesis space (haystack)
True solution (needle)

Deep nets


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



What happens as we increase the data?



What happens as we shrink the hypothesis space?

The essence of machine learning:

- A pattern exists
- We cannot pin down the pattern as an equation
- We need to approximate the pattern as a function of the input
  - Using a set of observations (data) to uncover an underlying process

Regression vs. Classification

- Regression tasks: predicting real-valued outputs $y \in \mathbb{R}$
- Classification tasks: predicting discrete-valued quantity y
  - Binary Classification $y \in \{-1, 1\}$
  - Multiclass Classification $y \in \{1, 2, \dots, k\}$

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

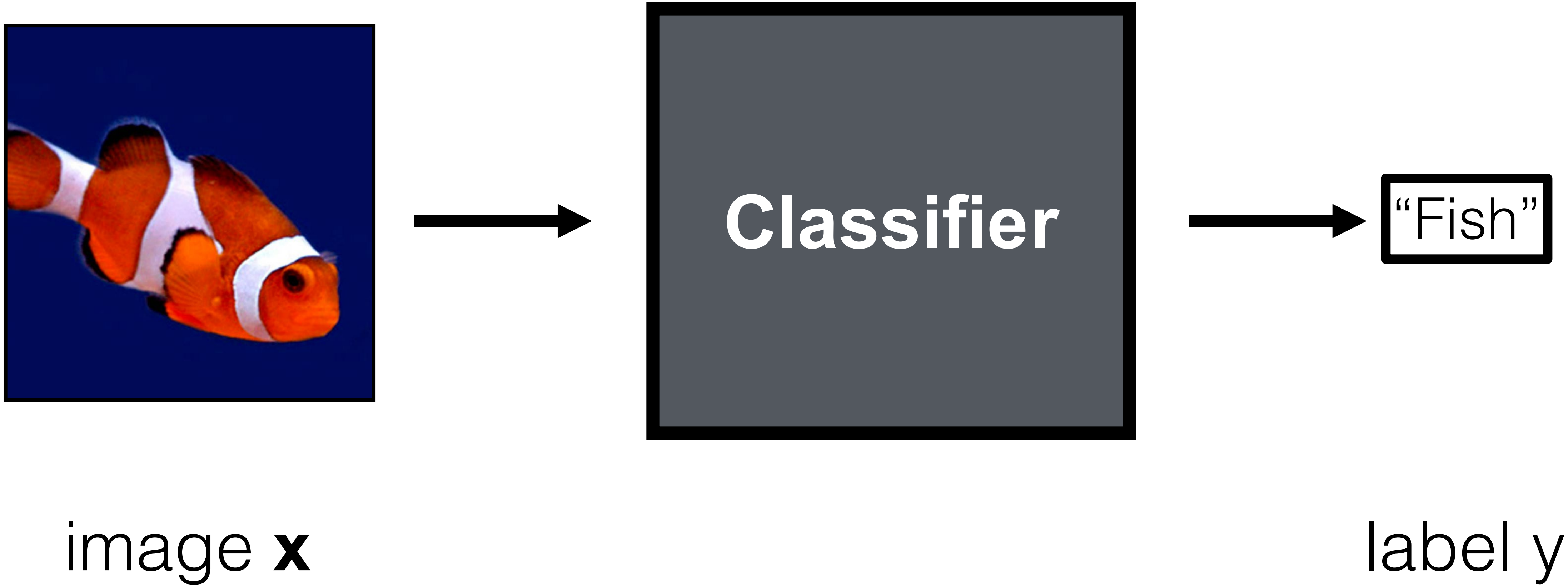


Image classification

image **x**

label y

Image classification

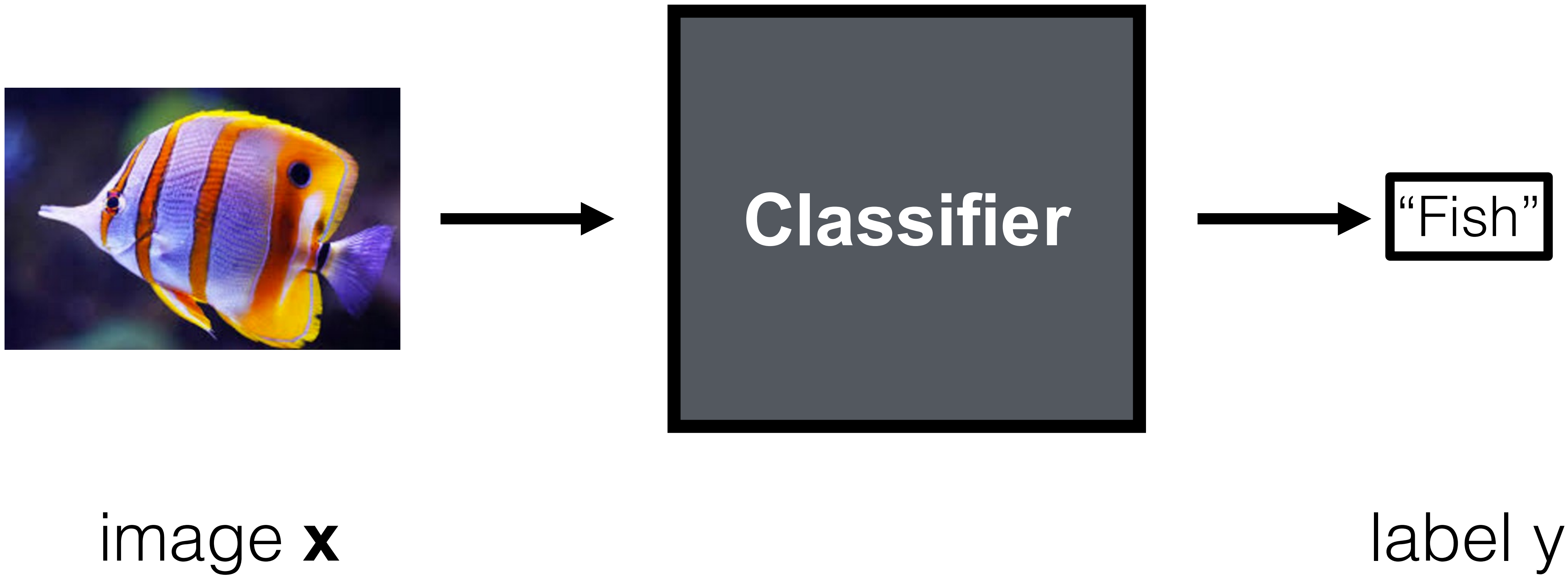
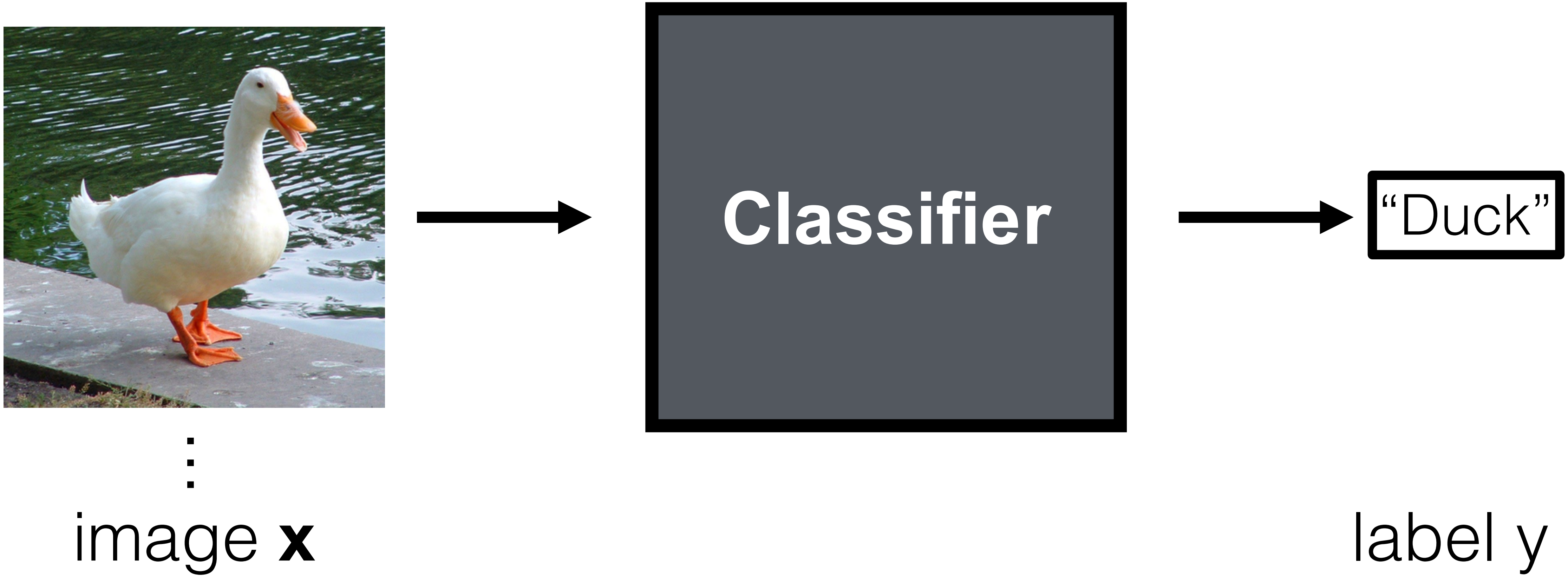
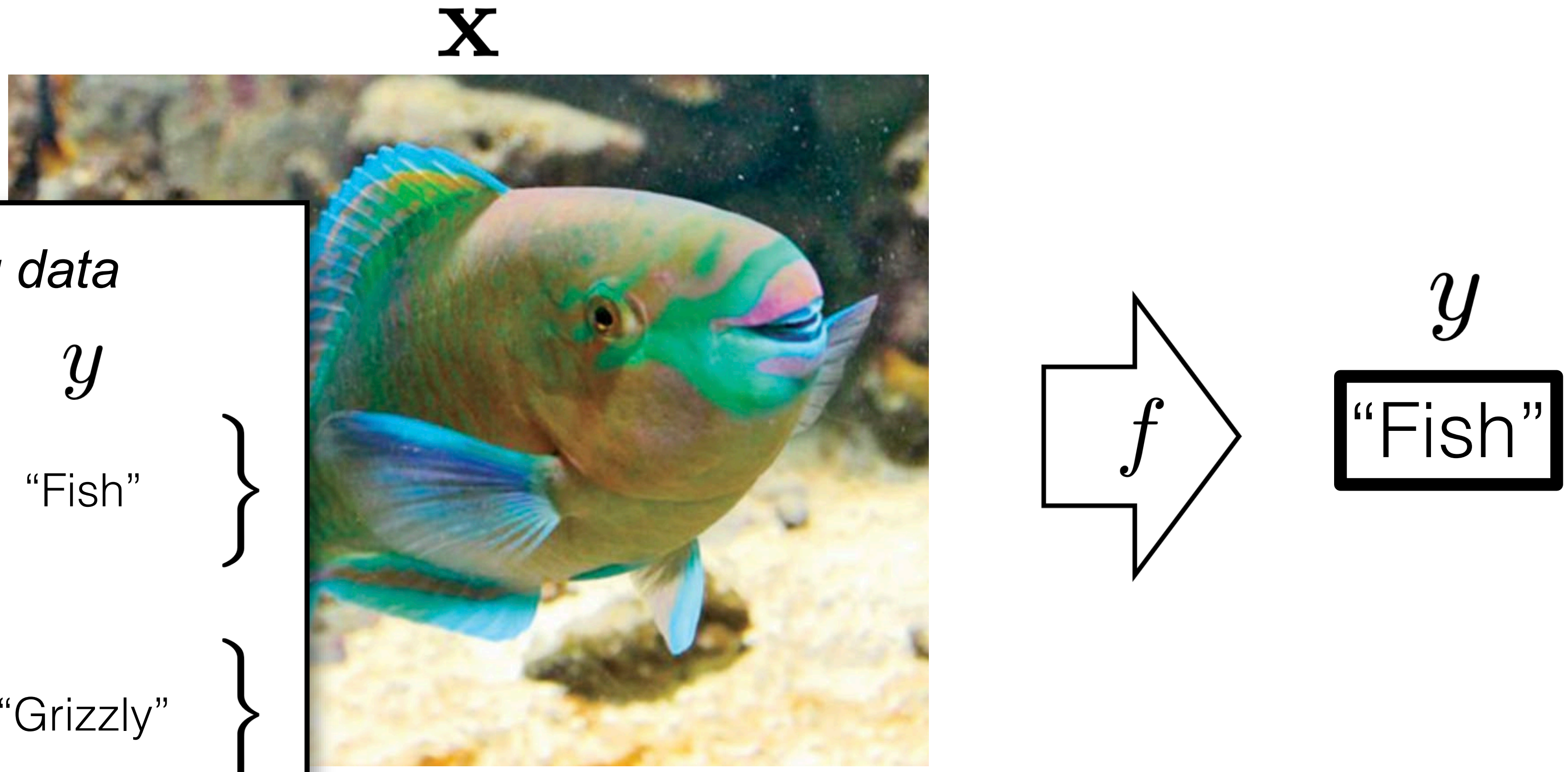


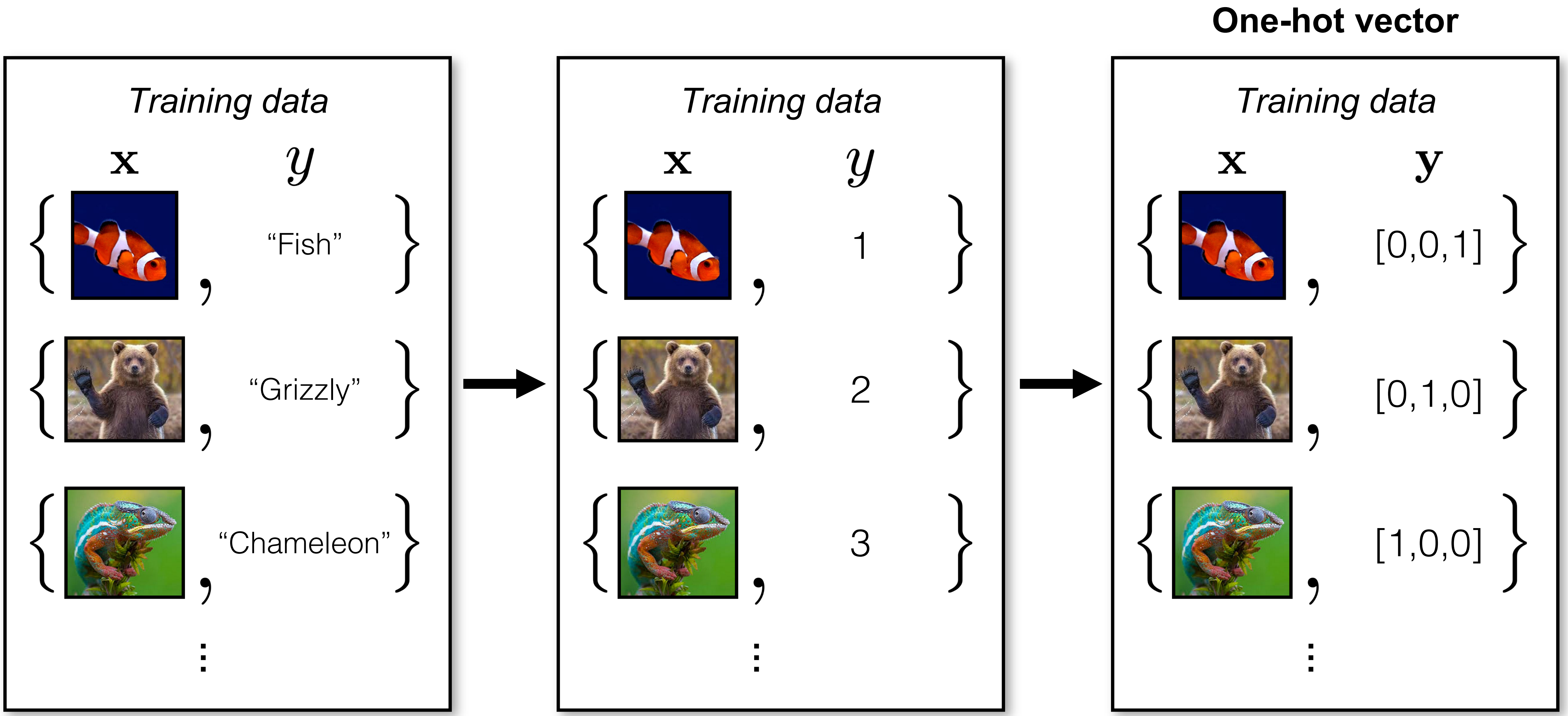
Image classification





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

How to represent class labels?



What should the loss be?

0-1 loss (number of misclassifications)

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = \mathbb{1}(\hat{\mathbf{y}} = \mathbf{y})$$

← 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

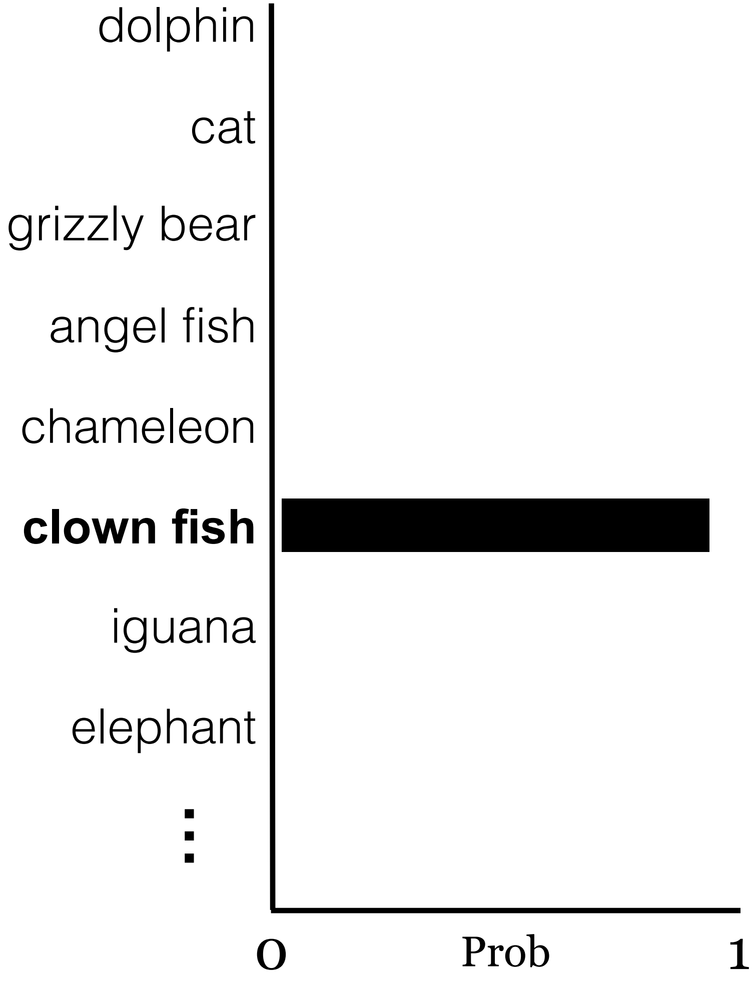
Ground truth label $\mathbf{y}$

$\mathbf{x}$

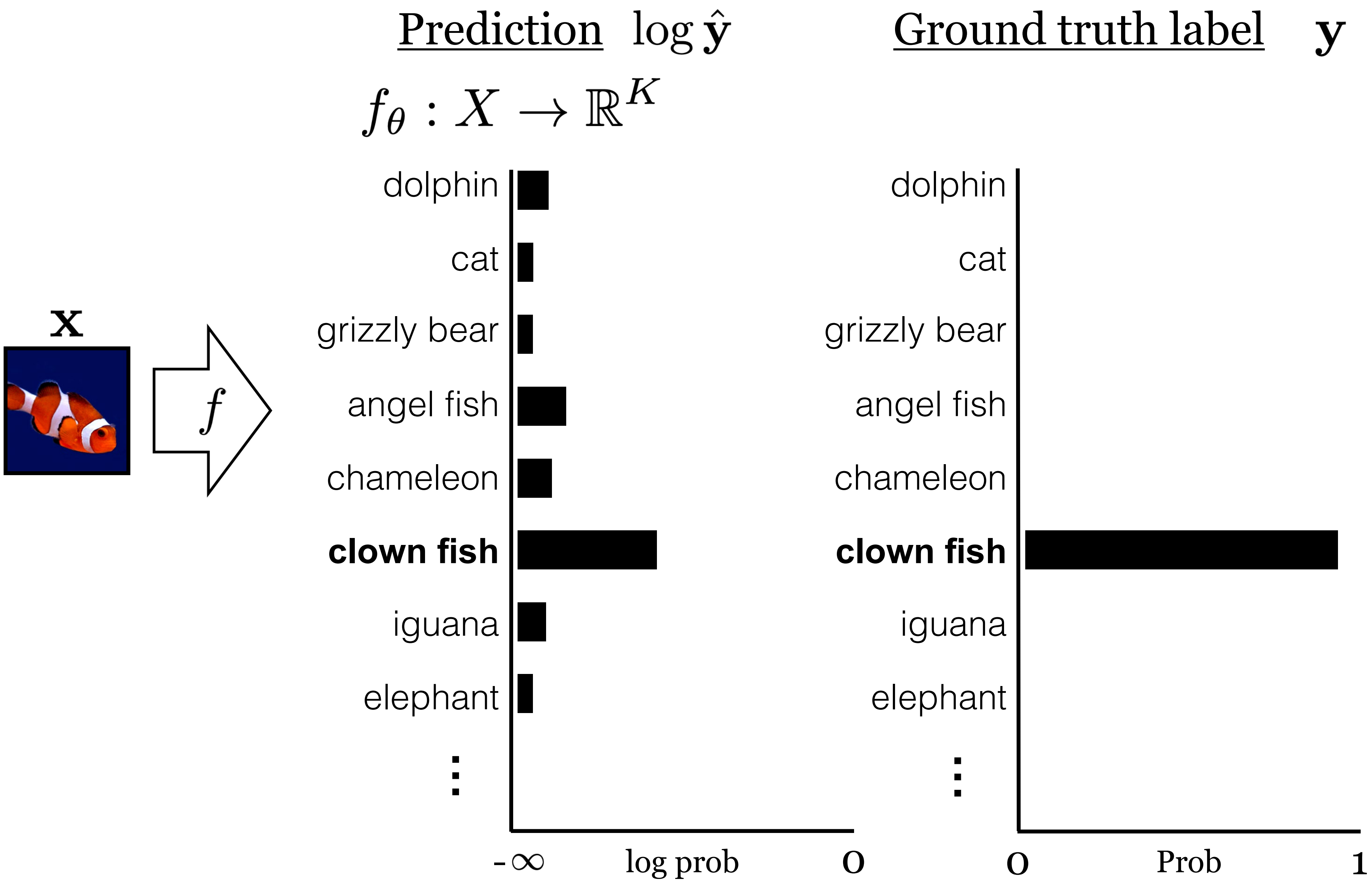
[0,0,0,0,0,1,0,0,…]

Ground truth label **y**

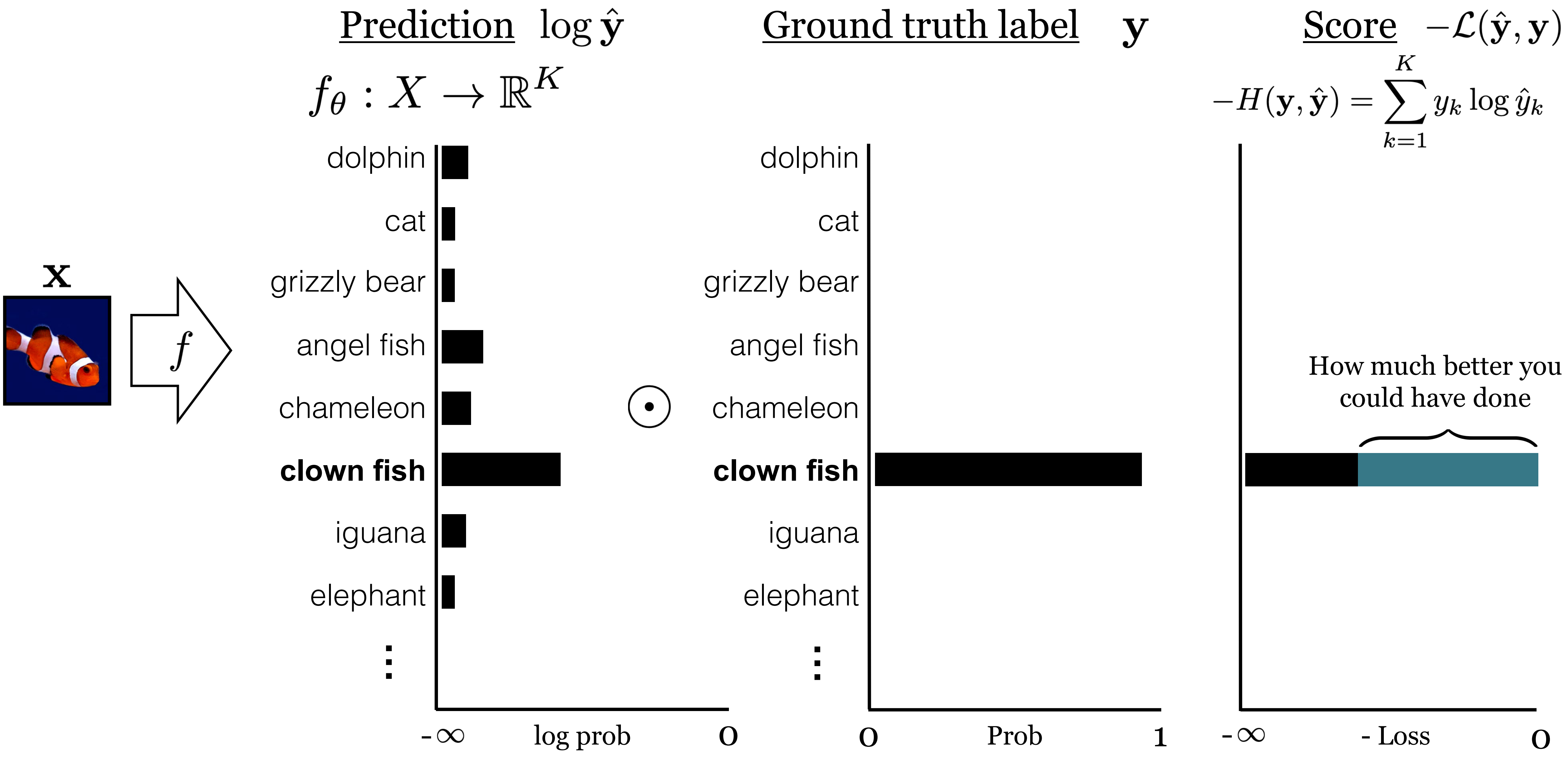
x



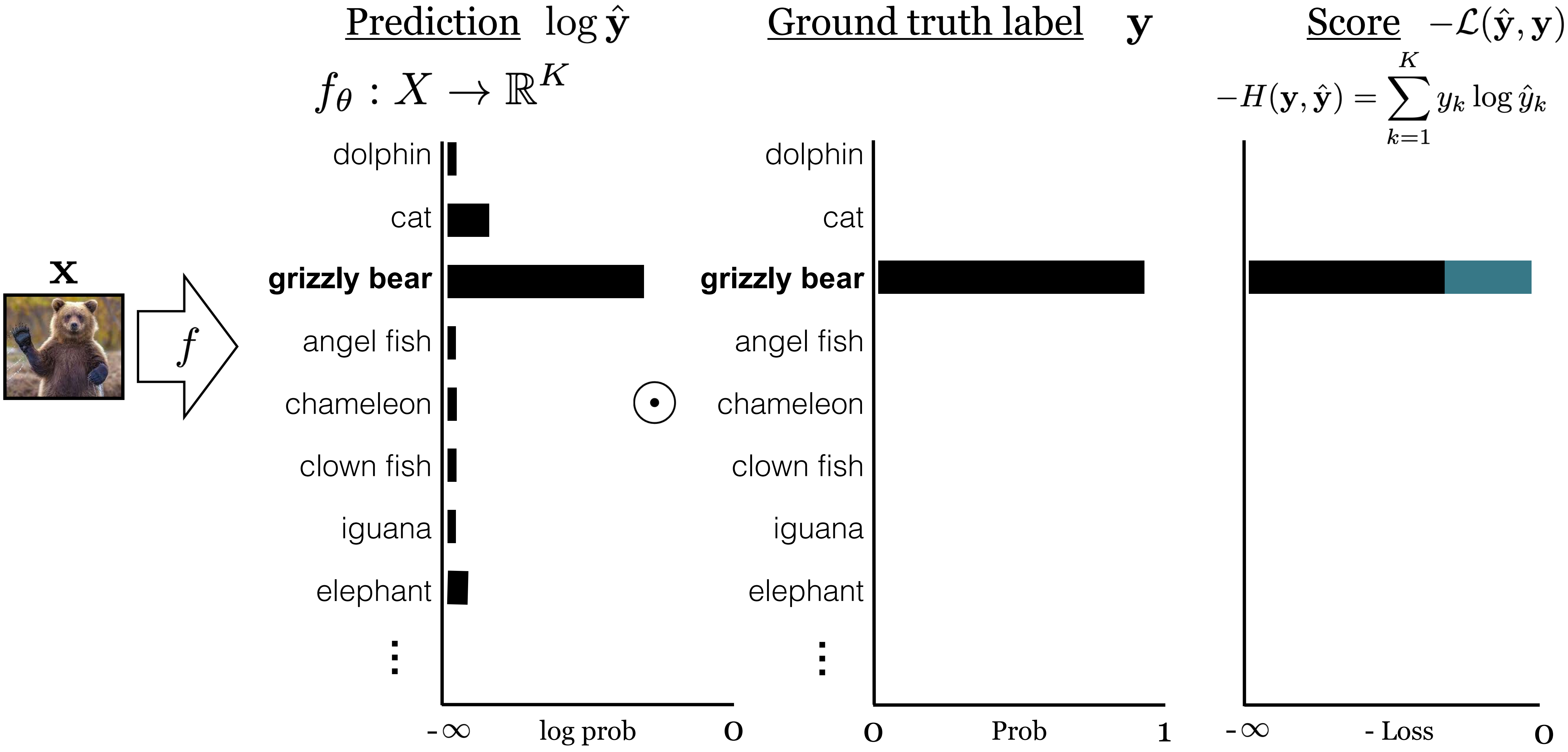
∞



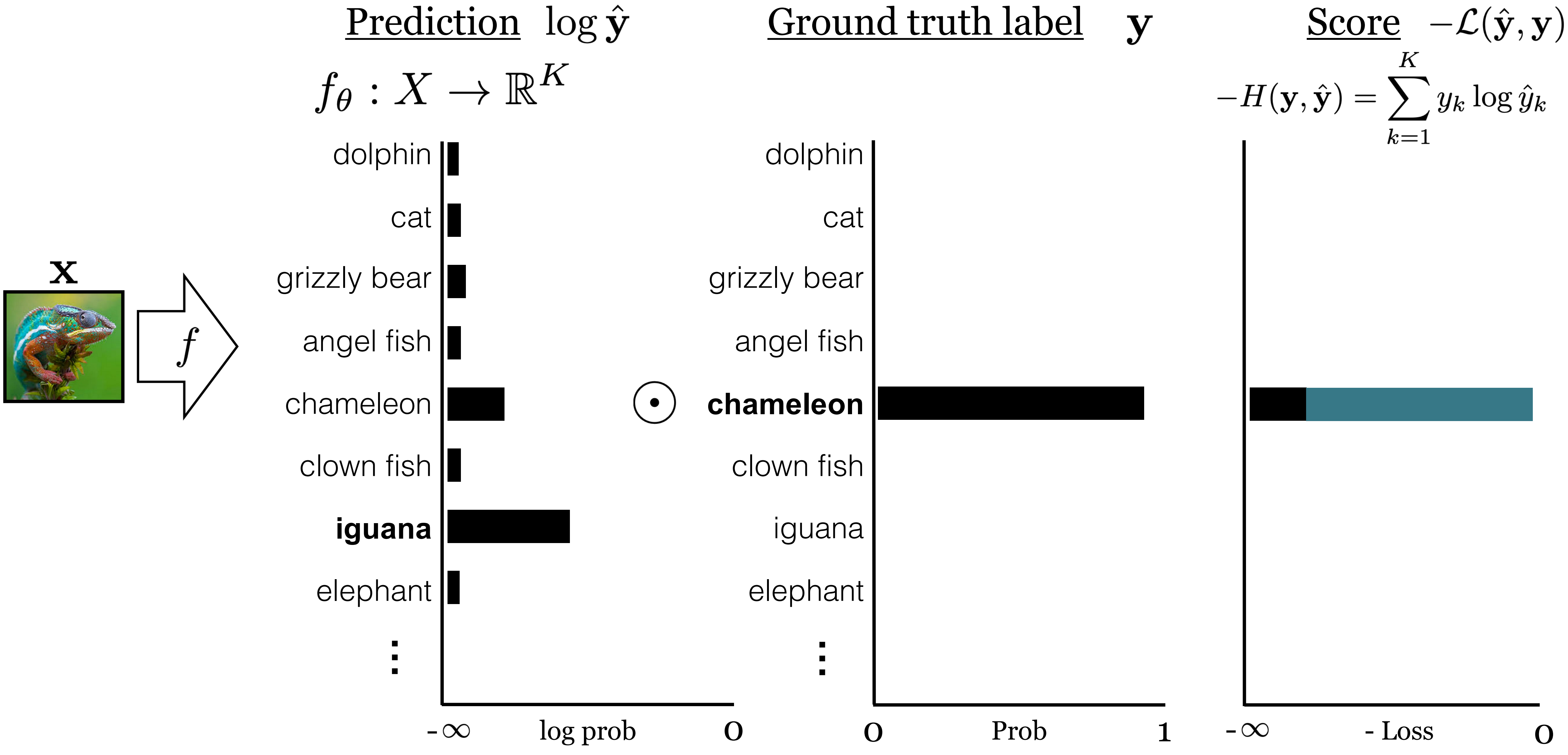
Prediction $\log \hat{\mathbf{y}}$
Ground truth label $\mathbf{y}$
$f_\theta : X \to \mathbb{R}^K$
$\mathbf{x}$
f
dolphin
cat
grizzly bear
angel fish
chameleon
clown fish
iguana
elephant
⋮
$-\infty$
log prob
0
Prob
1



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

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

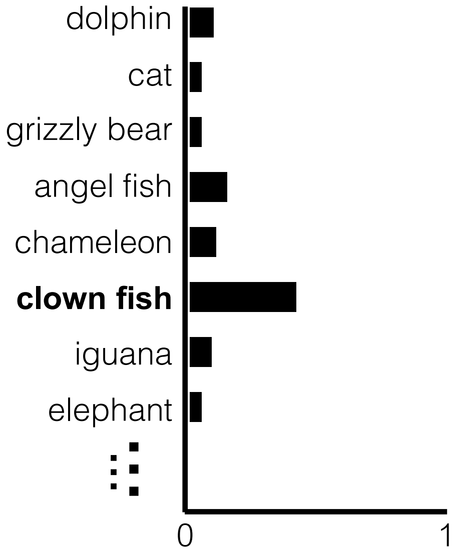
$$f_\theta : X \to \mathbb{R}^K$$

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

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

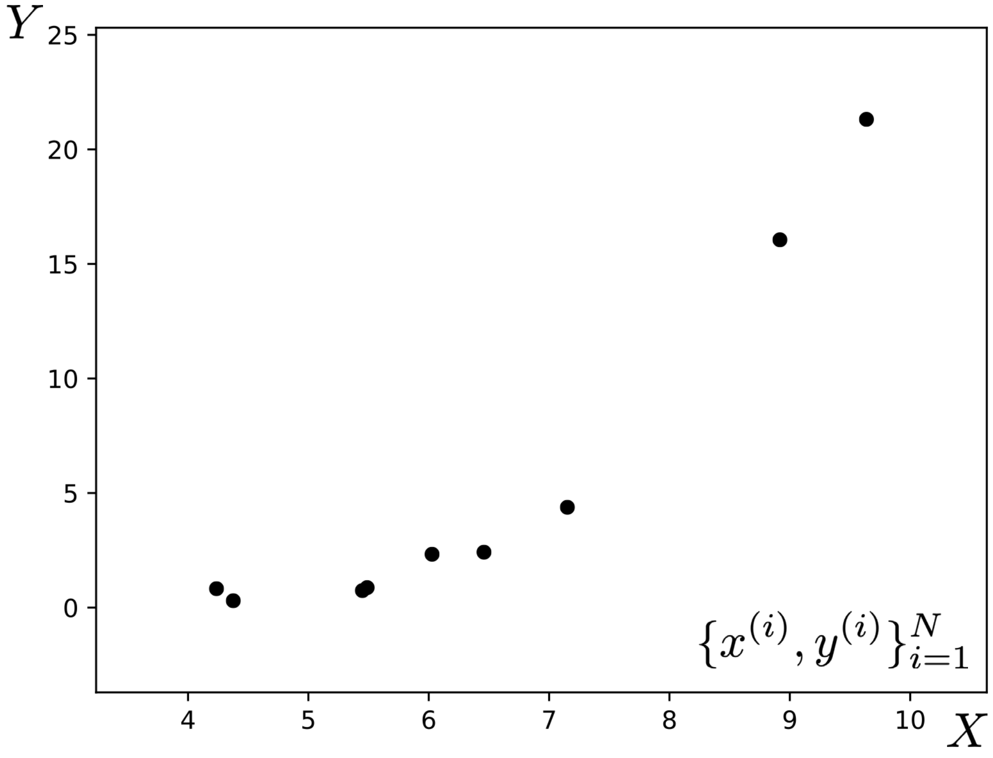
Recap:

Linear Regression

(f_θ is a linear function)

Linear regression

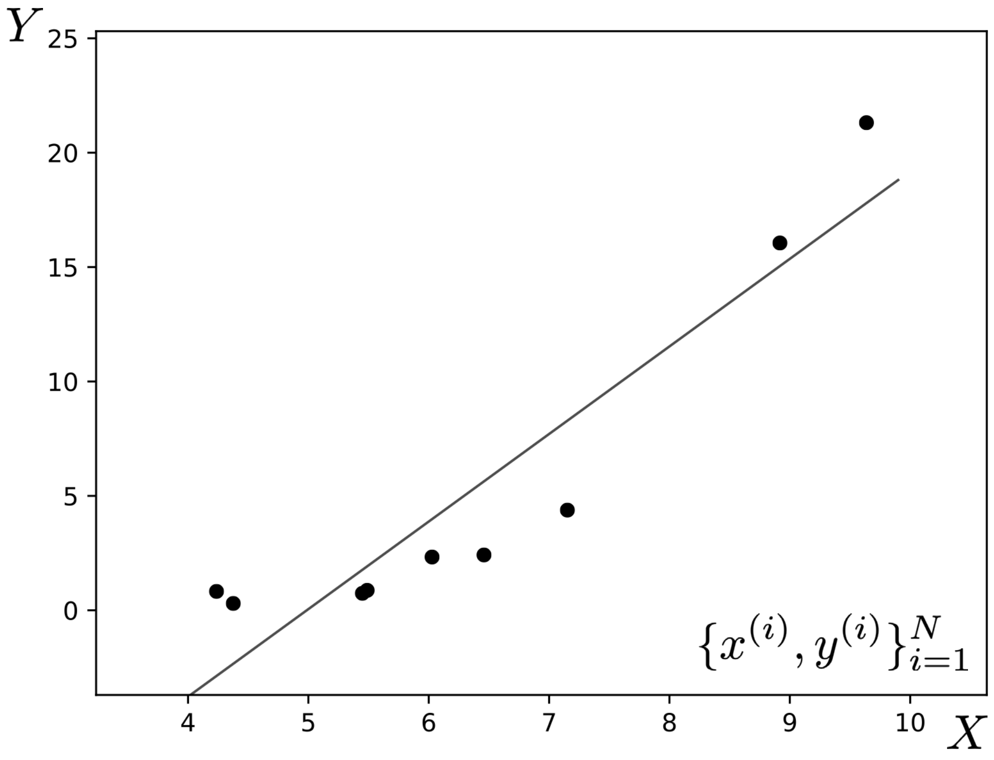
Training data



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

Linear regression

Training data



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

Linear Regression

(f_θ is a linear function)

Linear Regression

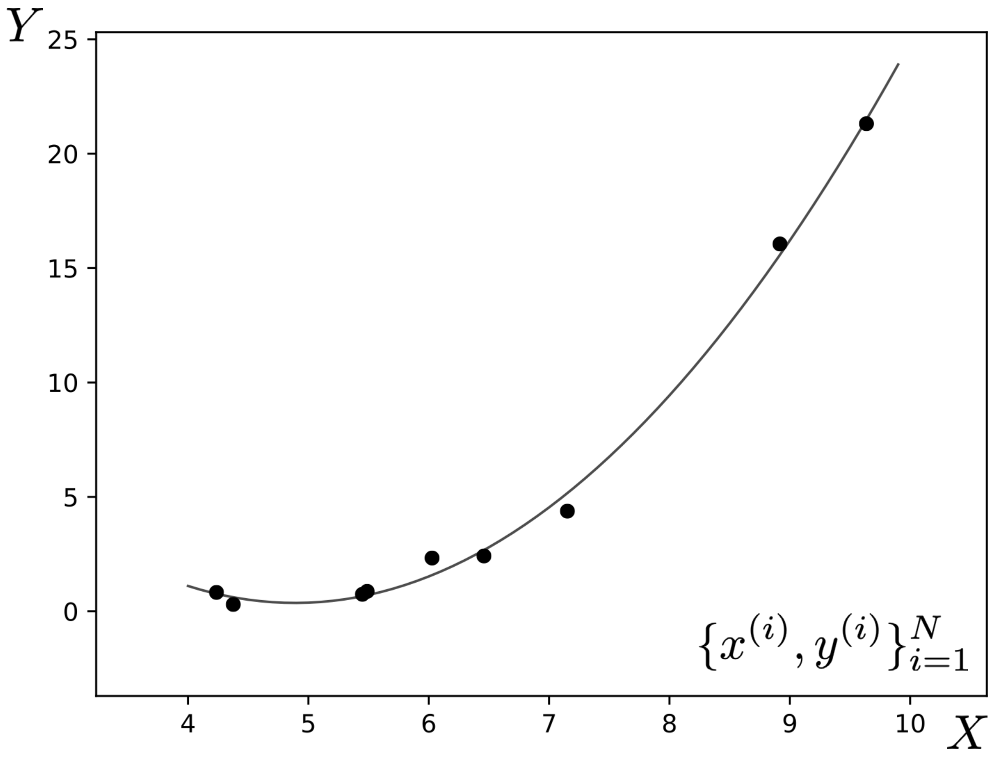
(f_θ is a linear function)

Polynomial Regression

(f_θ is a polynomial function)

Polynomial regression

Training data

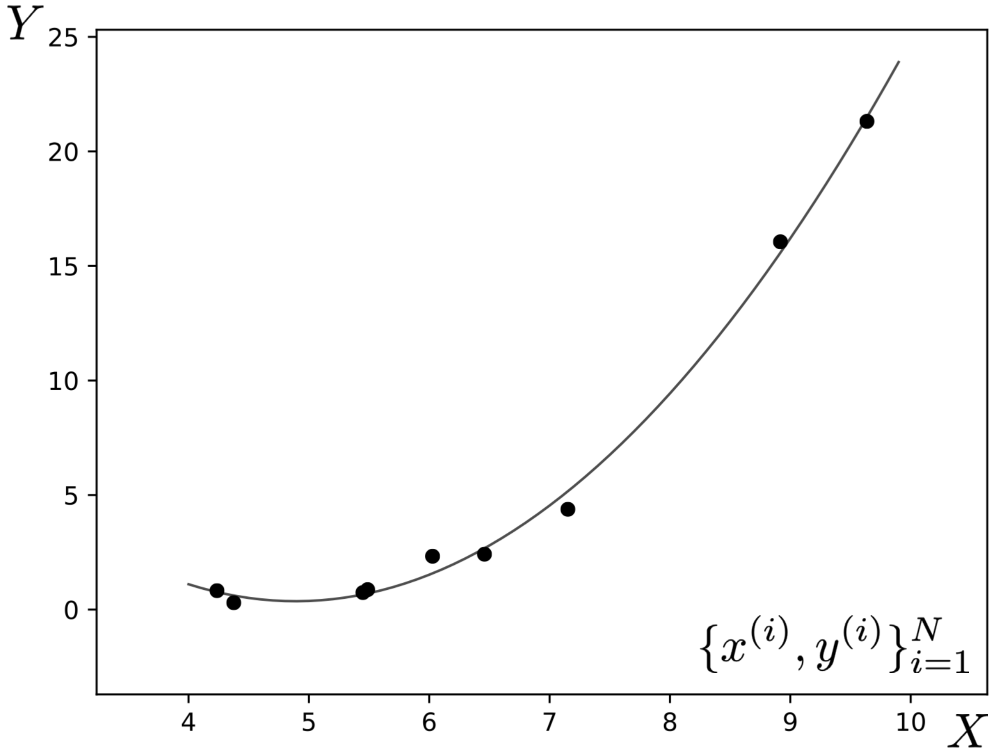


$$f_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

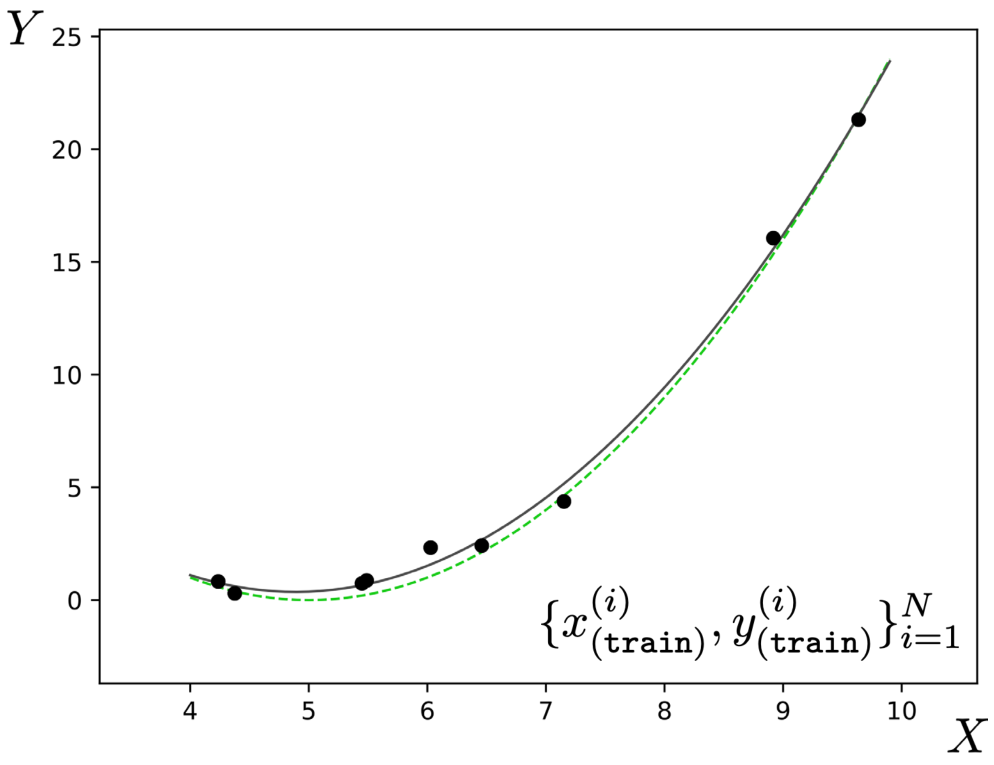
$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

K-th degree polynomial regression

Training data



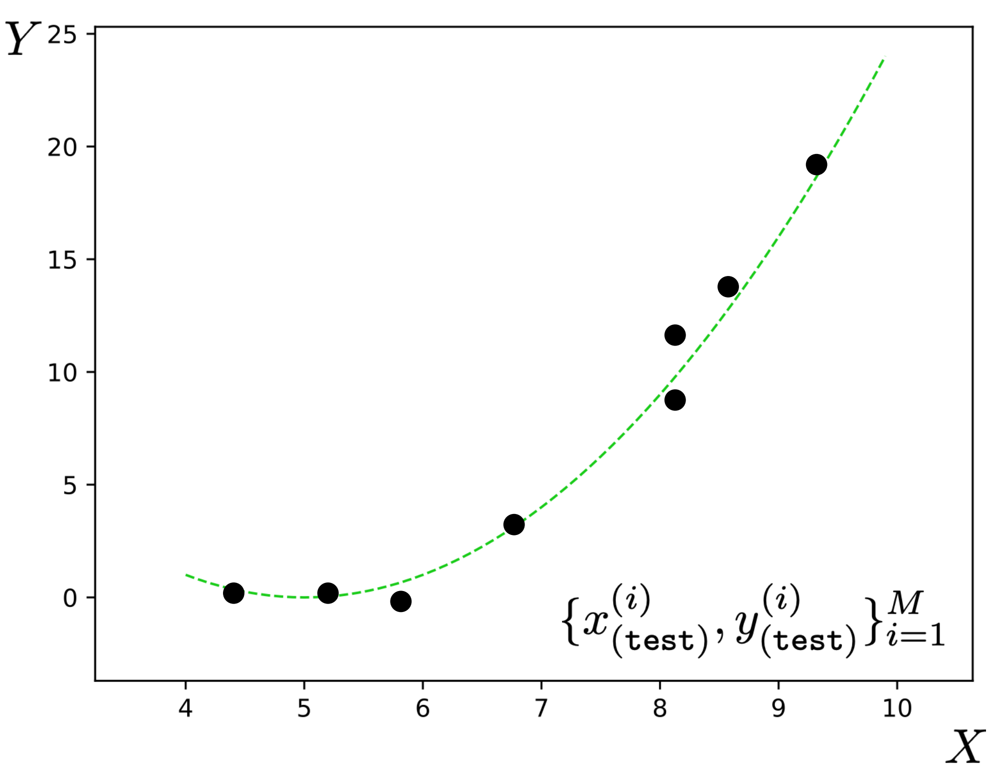
Training data



Training objective:

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

Test data

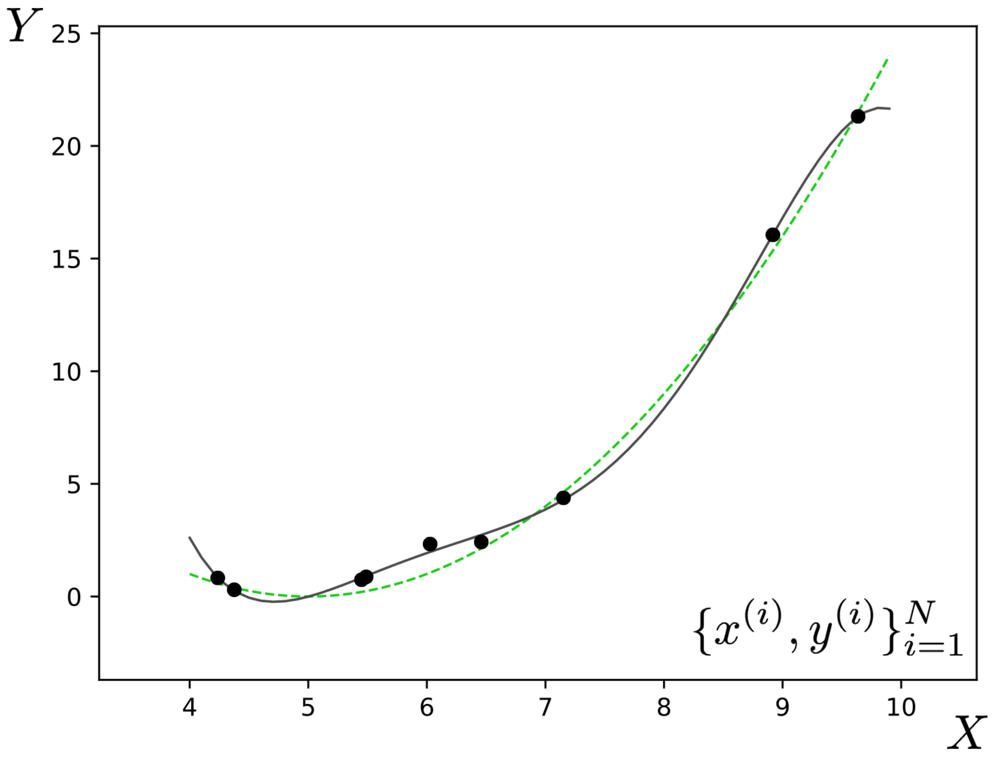


Test time evaluation:

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

What happens as we add more basis functions?

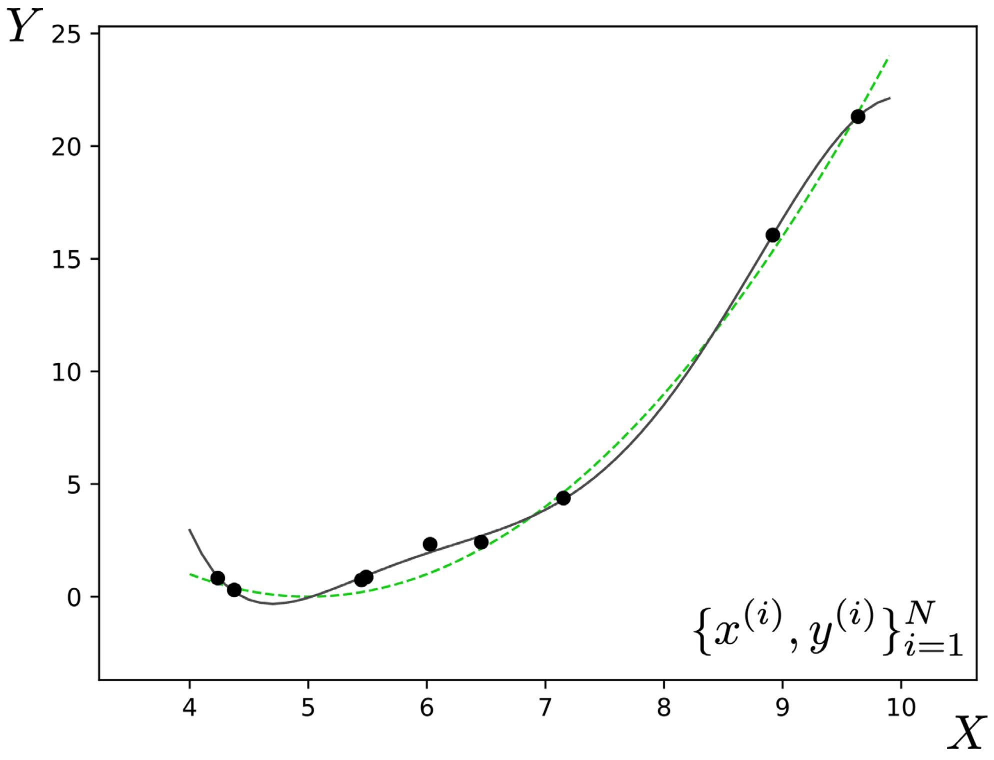
K = 5



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

What happens as we add more basis functions?

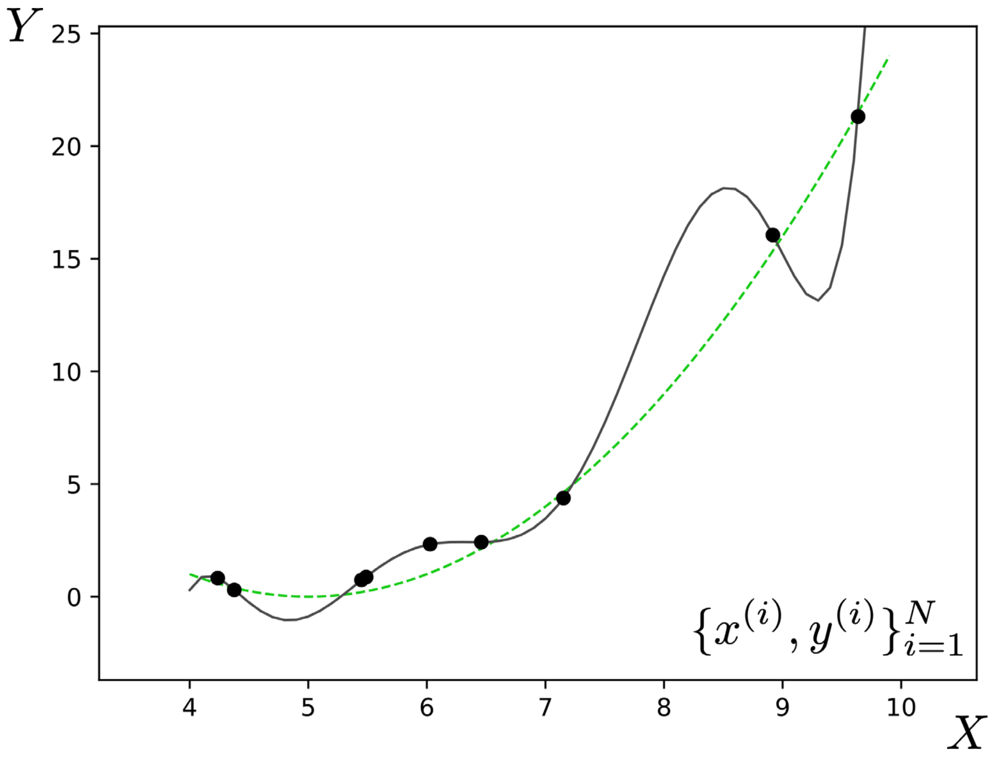
K = 6



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

What happens as we add more basis functions?

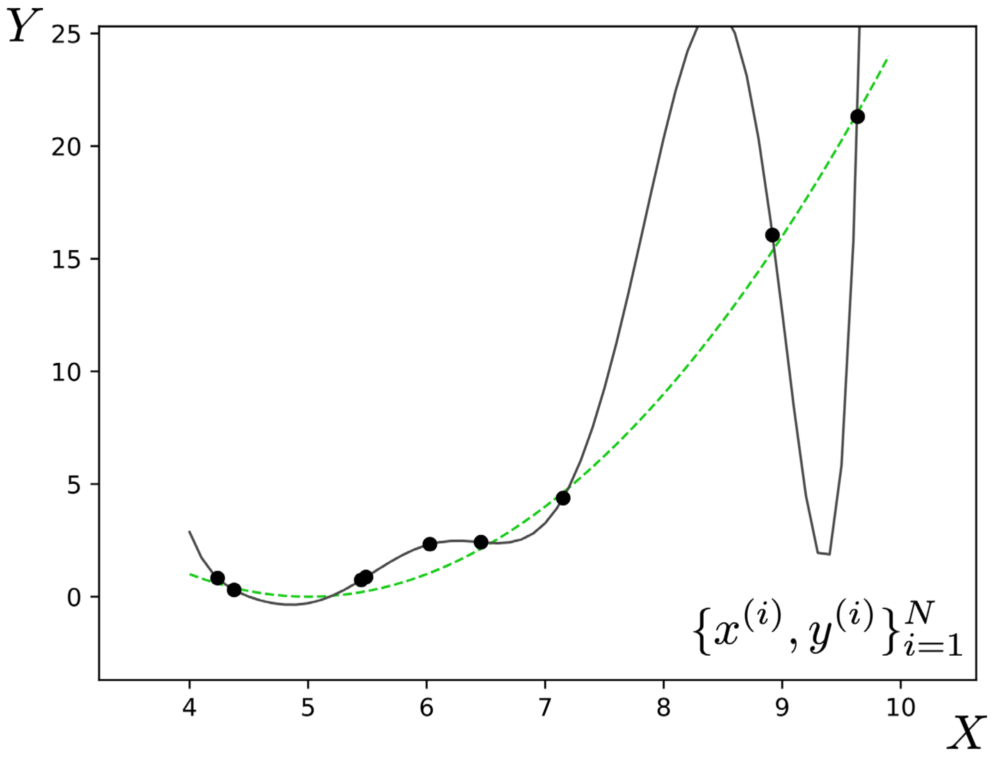
K = 7



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

What happens as we add more basis functions?

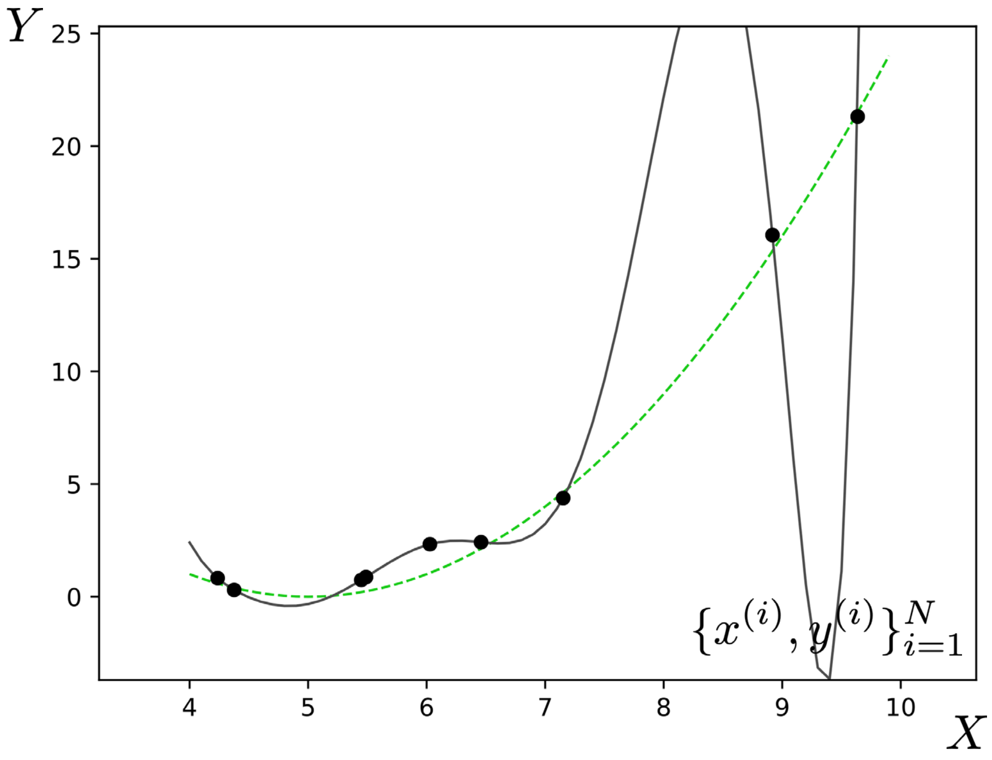
K = 8



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

What happens as we add more basis functions?

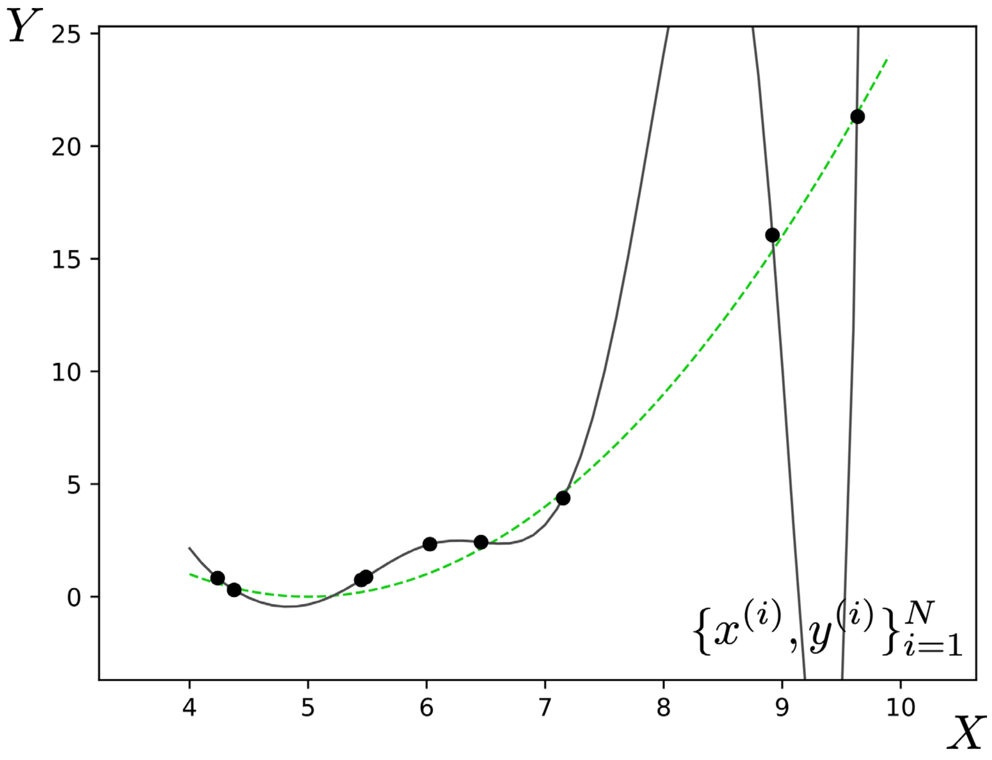
K = 9



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

What happens as we add more basis functions?

K = 10

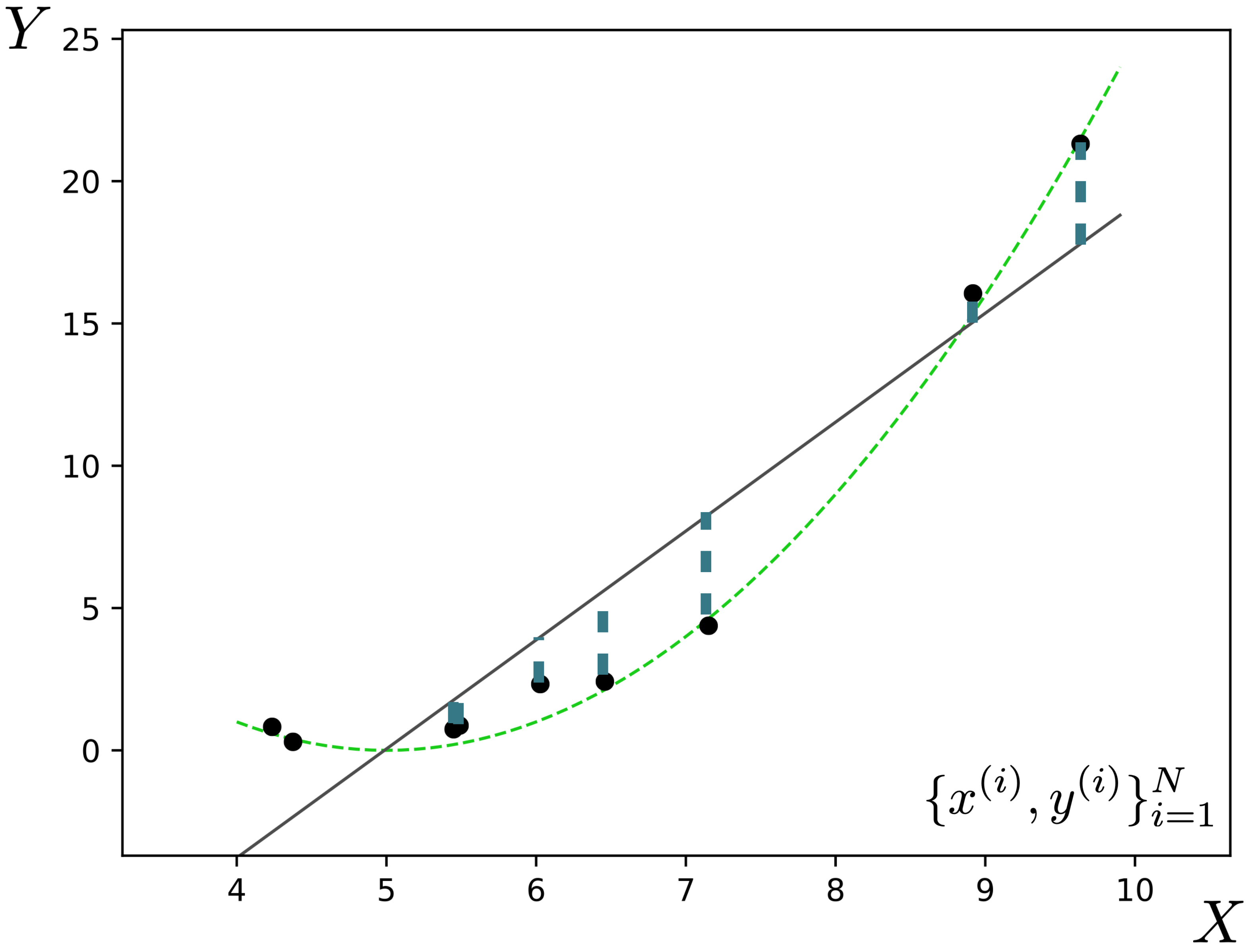


$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

This phenomenon is called **overfitting**.

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

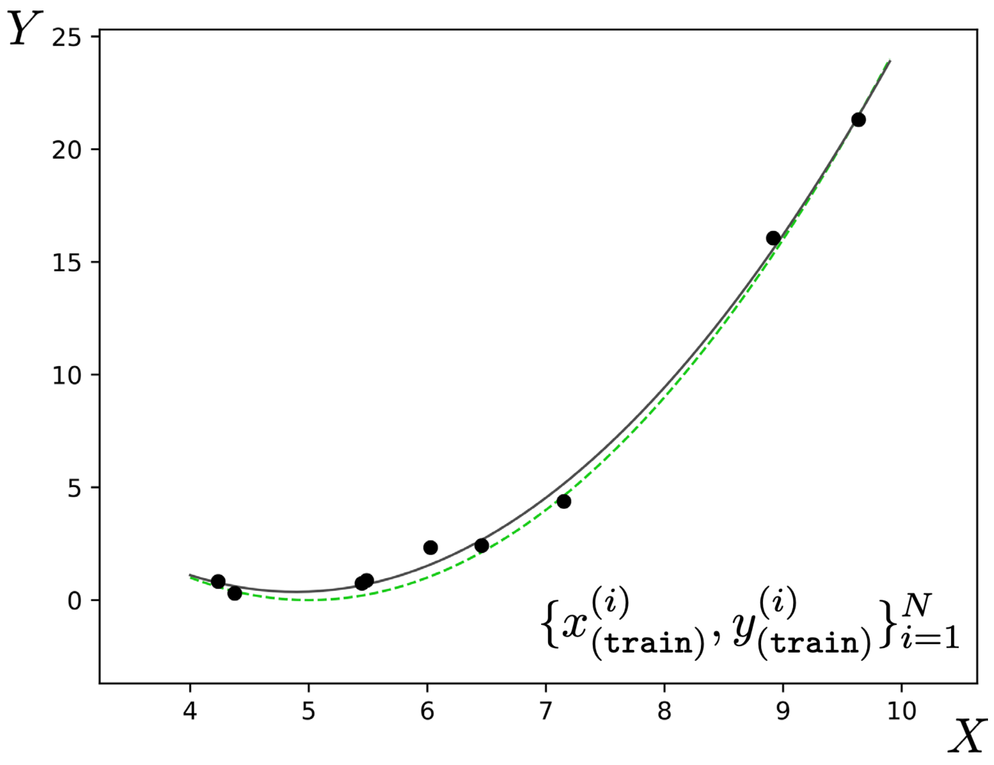
K = 1



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

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

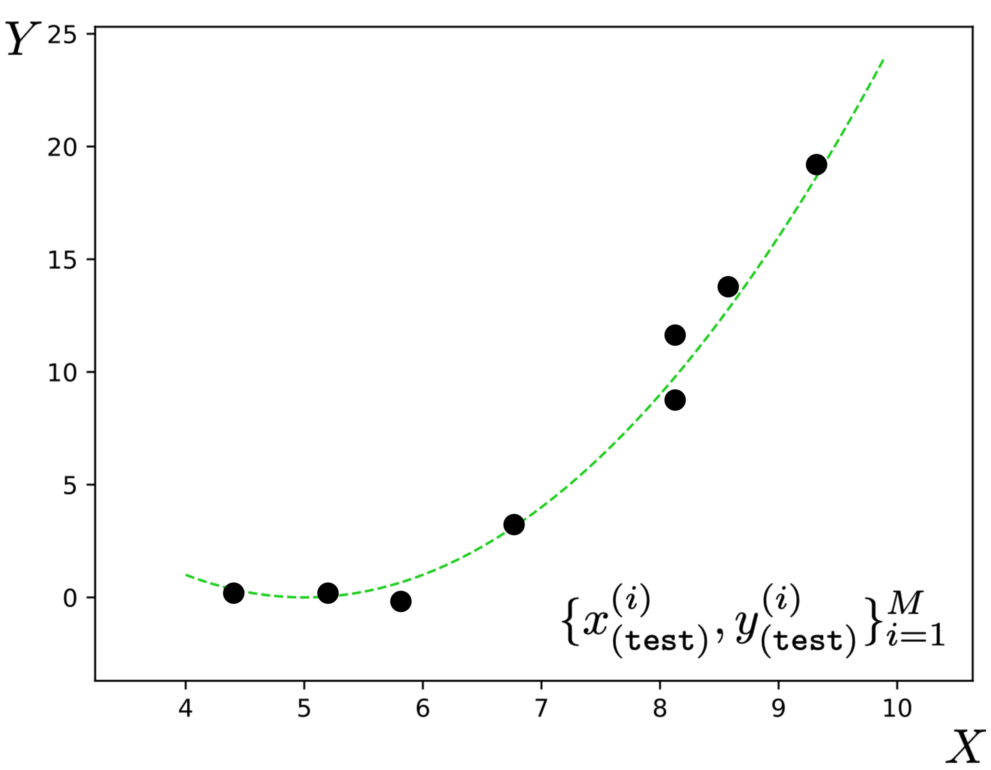
Training data



True **data-generating process**

$p_{\texttt{data}}$

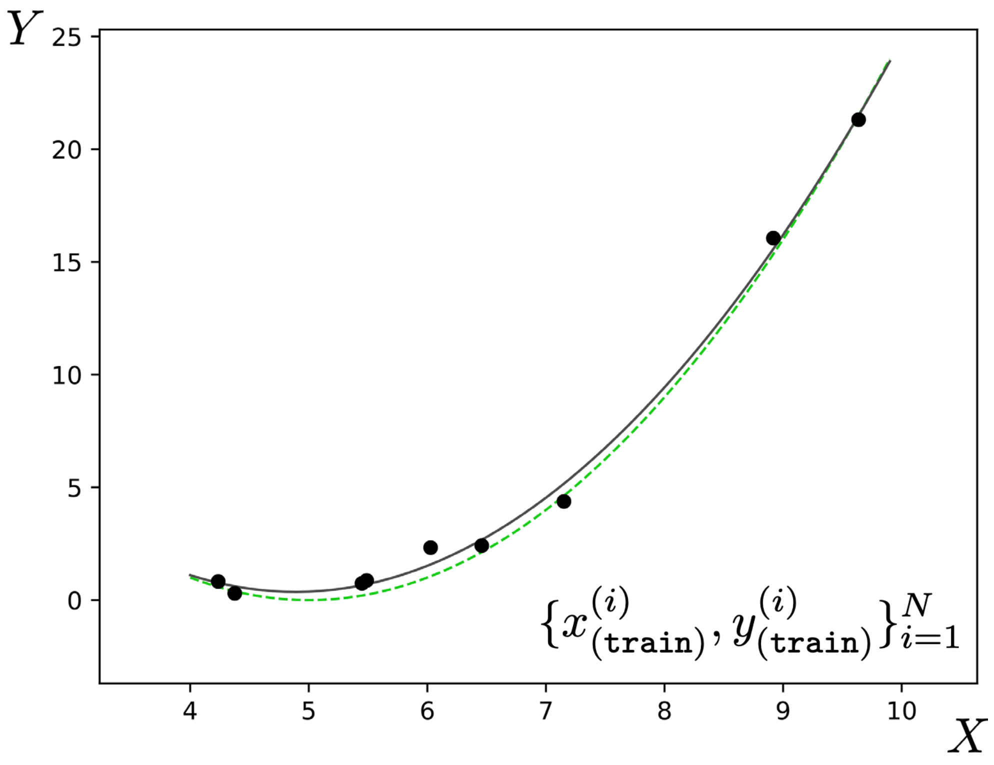
Test data



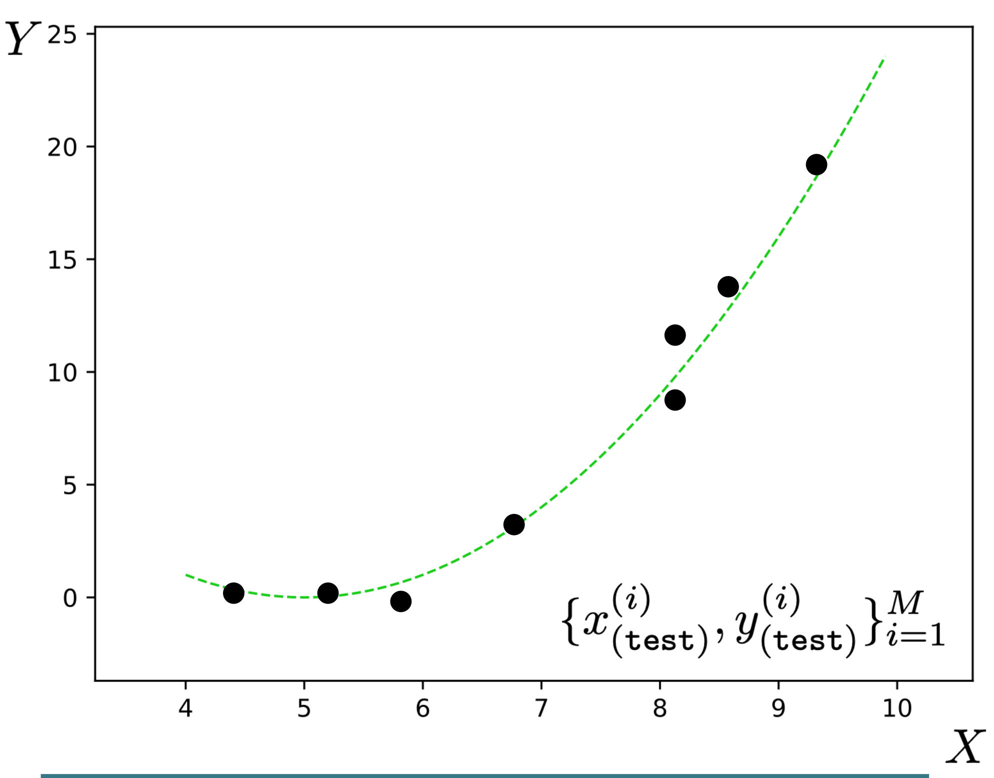
$$\{x^{(i)}_{(\texttt{train})}, y^{(i)}_{(\texttt{train})}\} \overset{\texttt{iid}}{\sim} p_{\texttt{data}}$$

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

Training data



Test data

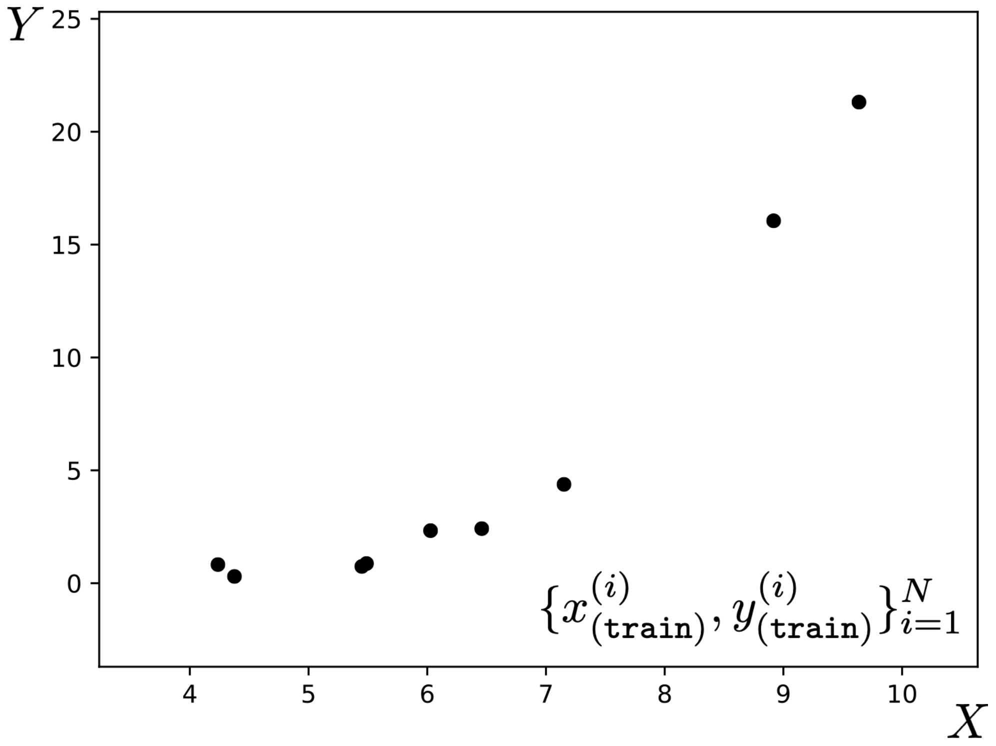


This is a huge assumption!
Almost never true in practice!

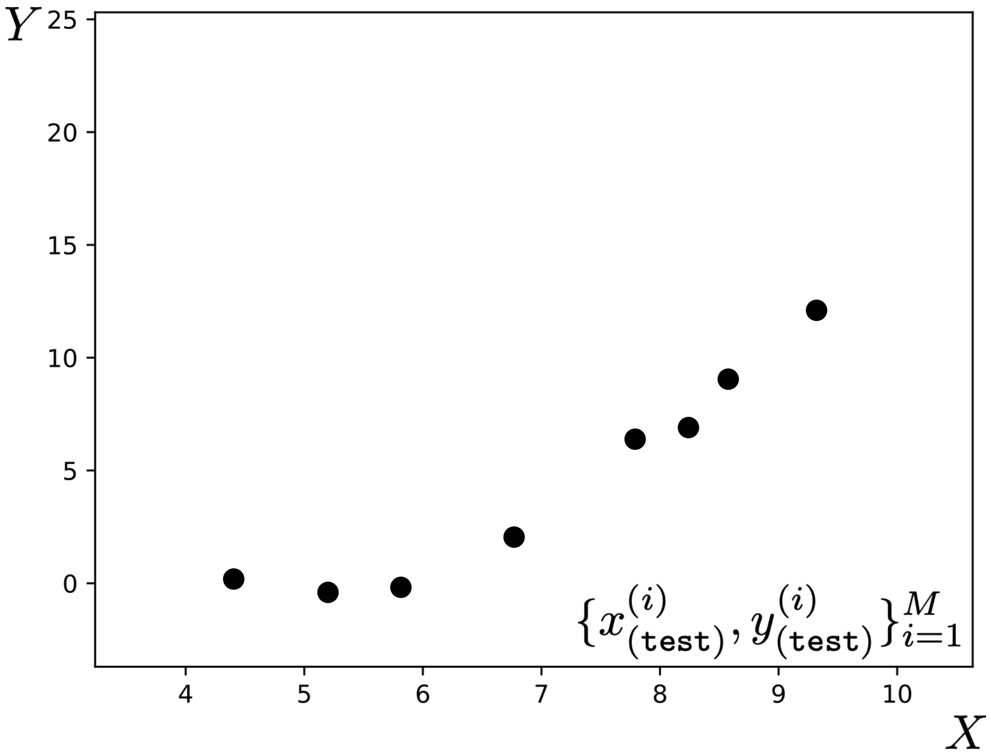
$$\{x^{(i)}_{(\texttt{train})}, y^{(i)}_{(\texttt{train})}\} \overset{\texttt{iid}}{\sim} p_{\texttt{data}}$$

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

Training data



Test data



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

Much more commonly, we have

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

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.)

testing domain
(where we actual use our model)

training domain

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