Machine Learning

*Edited using slides from MITs 6.869, and Stanford CS231n

Outline

1. Introduction
2. Image enhancement
3. Frequency domain operations
4. Image descriptors
5. **Machine learning**
6. Neural networks
7. Segmentation and object detection
8. Morphological processing
9. Geometric transformations
10. Motion analysis and optical flow
11. Compression
12. Other topics

Machine Learning

- The tools we have seen so far in the class rely on internal image content
- Image content is used to build feature vectors that allow retrieval or classification
- This can be a tedious process and can involve re-desgining features to better handle new datasets
- An alternate approach is to rely on labelled training data and let machine learning tools build the desired features.

Image Inpainting

J. Hays and A. Efros, Scene Completion Using Millions of Photographs, SIGGRAPH 2007

Image Inpainting

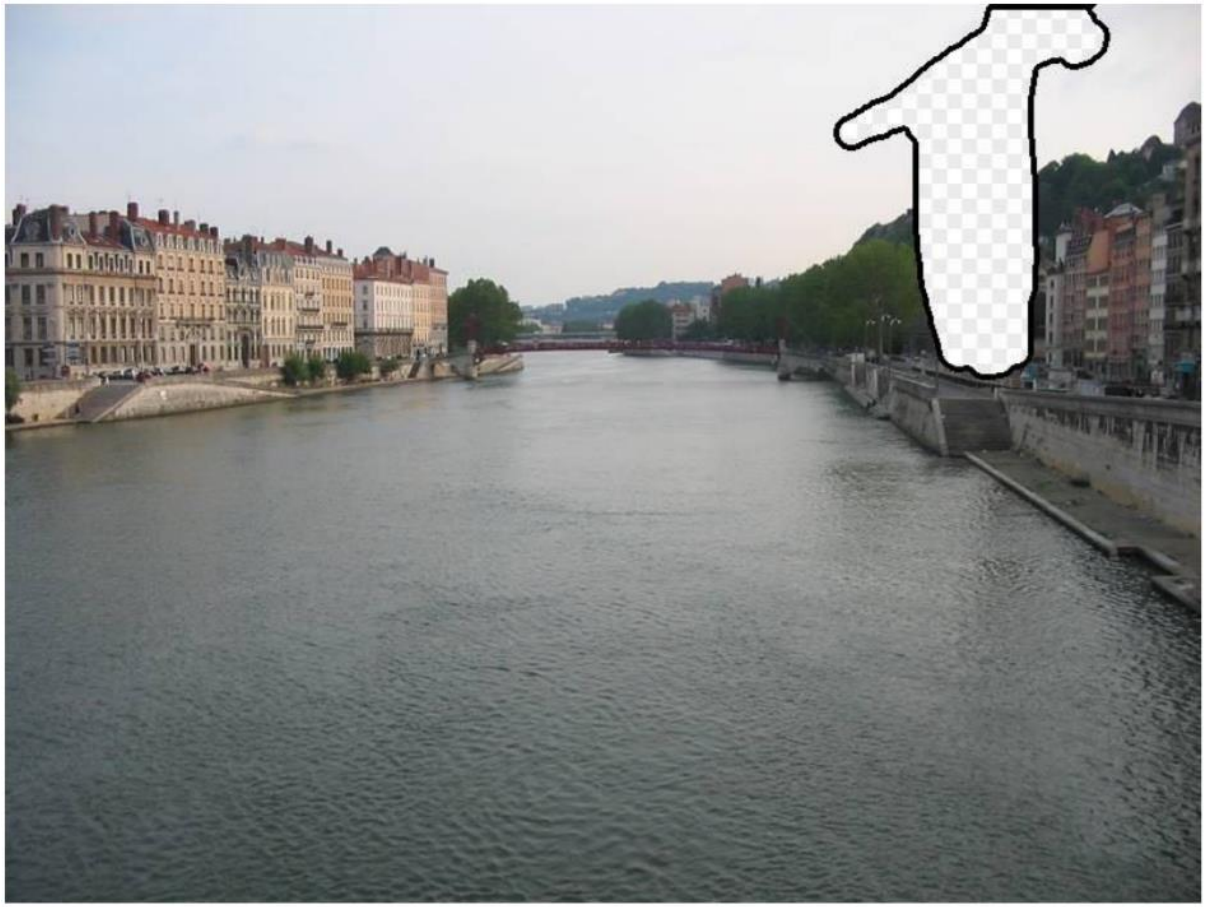
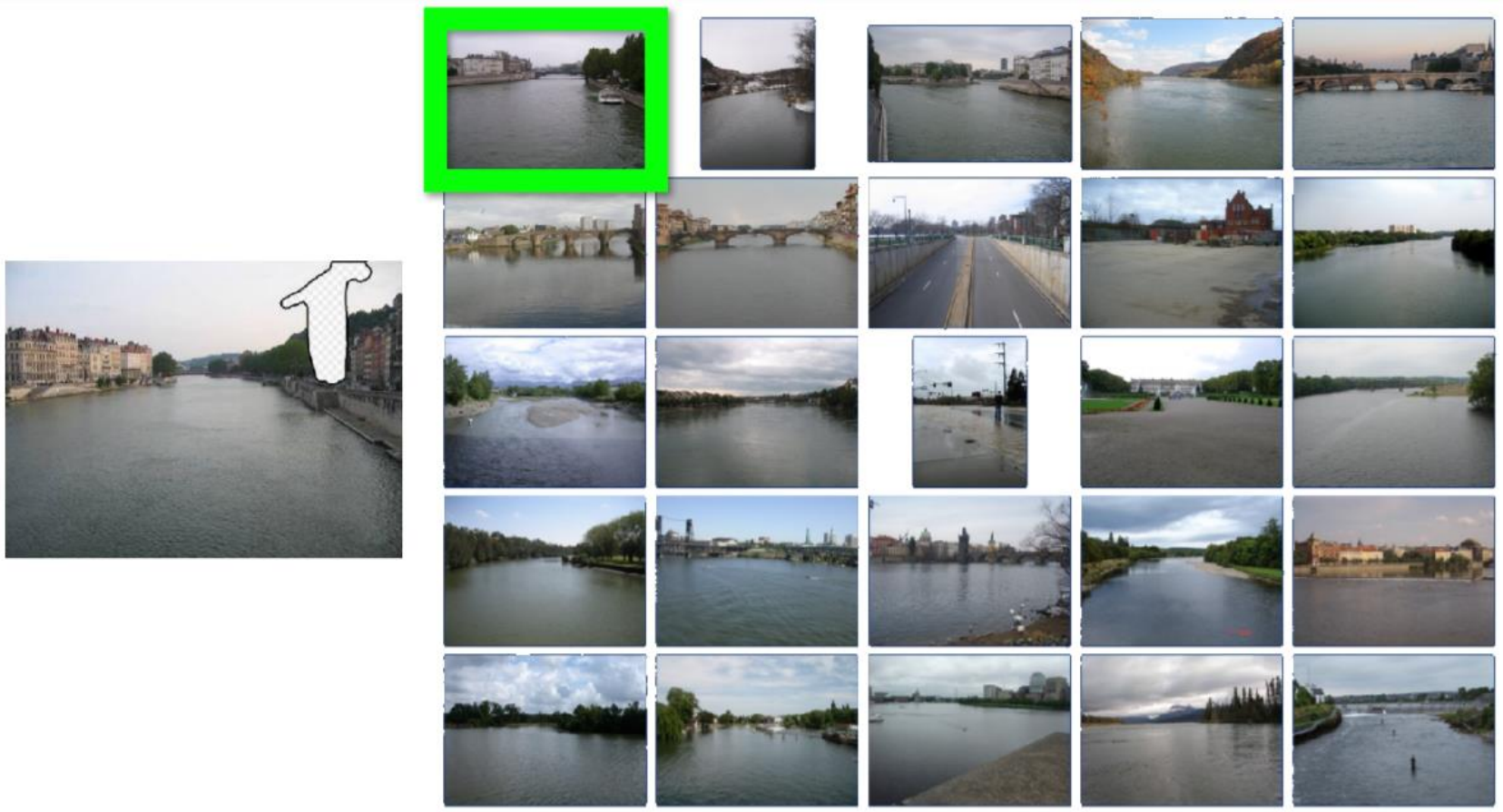


Image Inpainting

Image Inpainting



… 200 scene matches

Image Inpainting

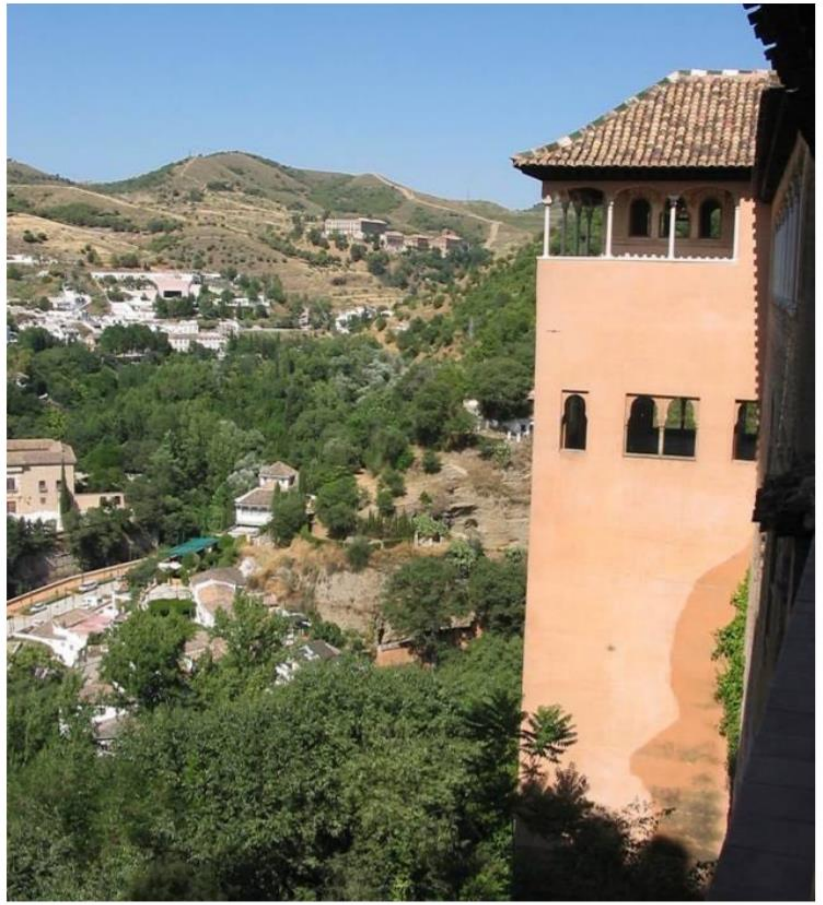
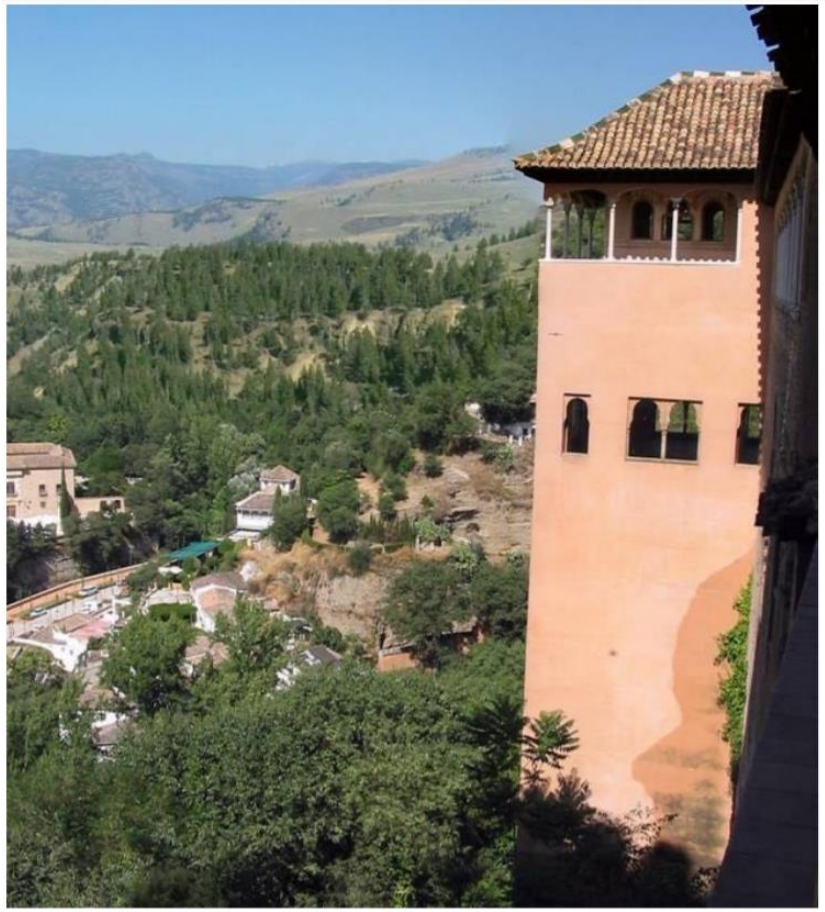
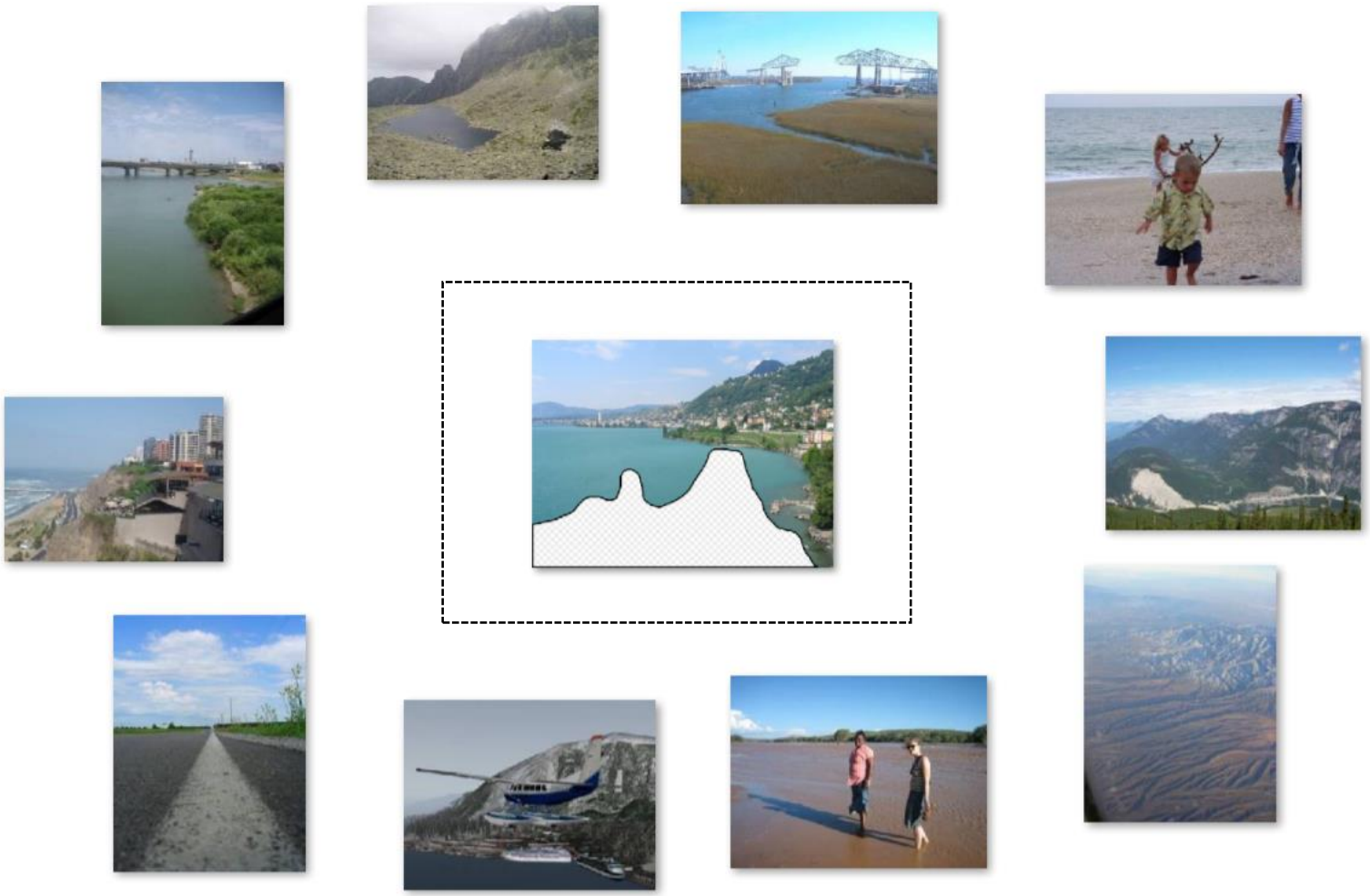


Image Inpainting

Image Inpainting

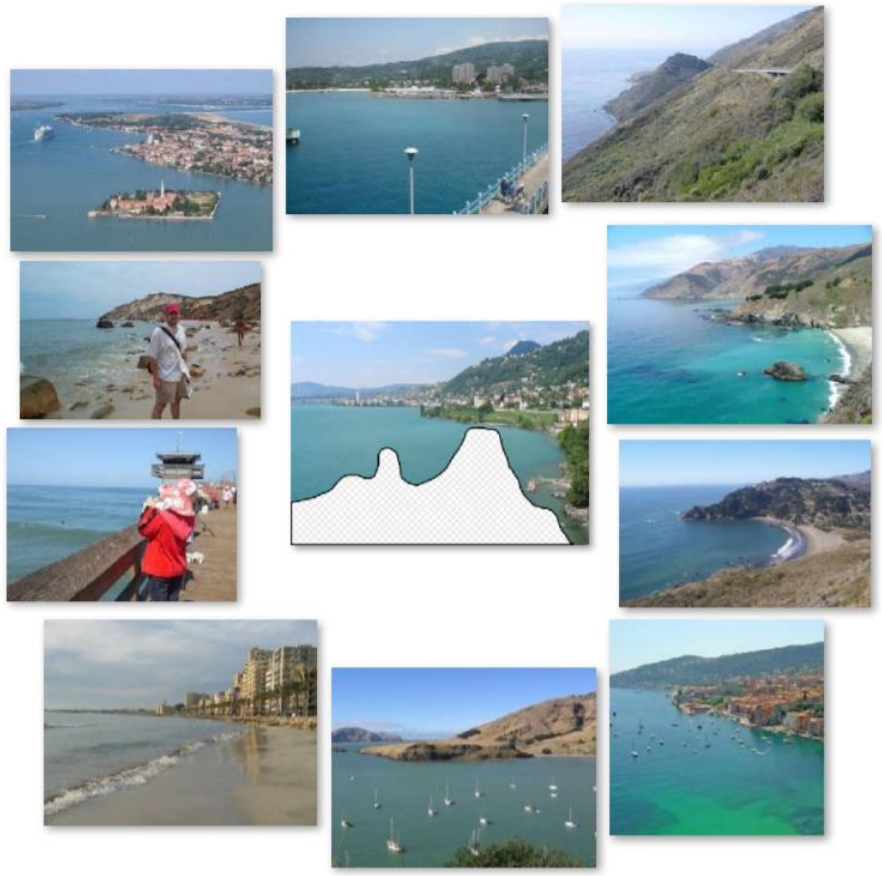


How ?



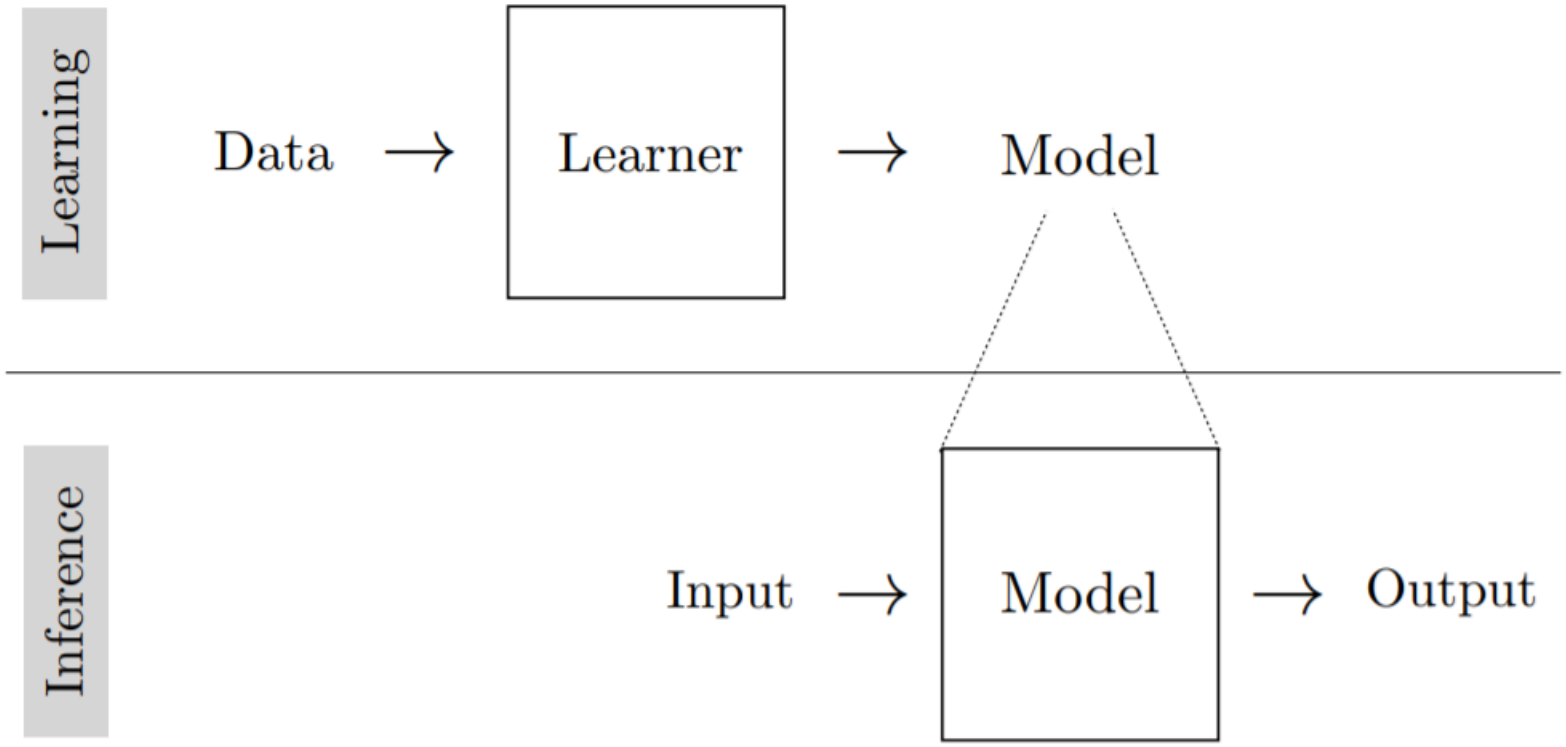
Nearest neighbor images from 20 thousand images

How ? Power of data



Nearest neighbor images from **20 million** images

Learning and Inference



What does ☆ do?

$2 \star 3 = 36$

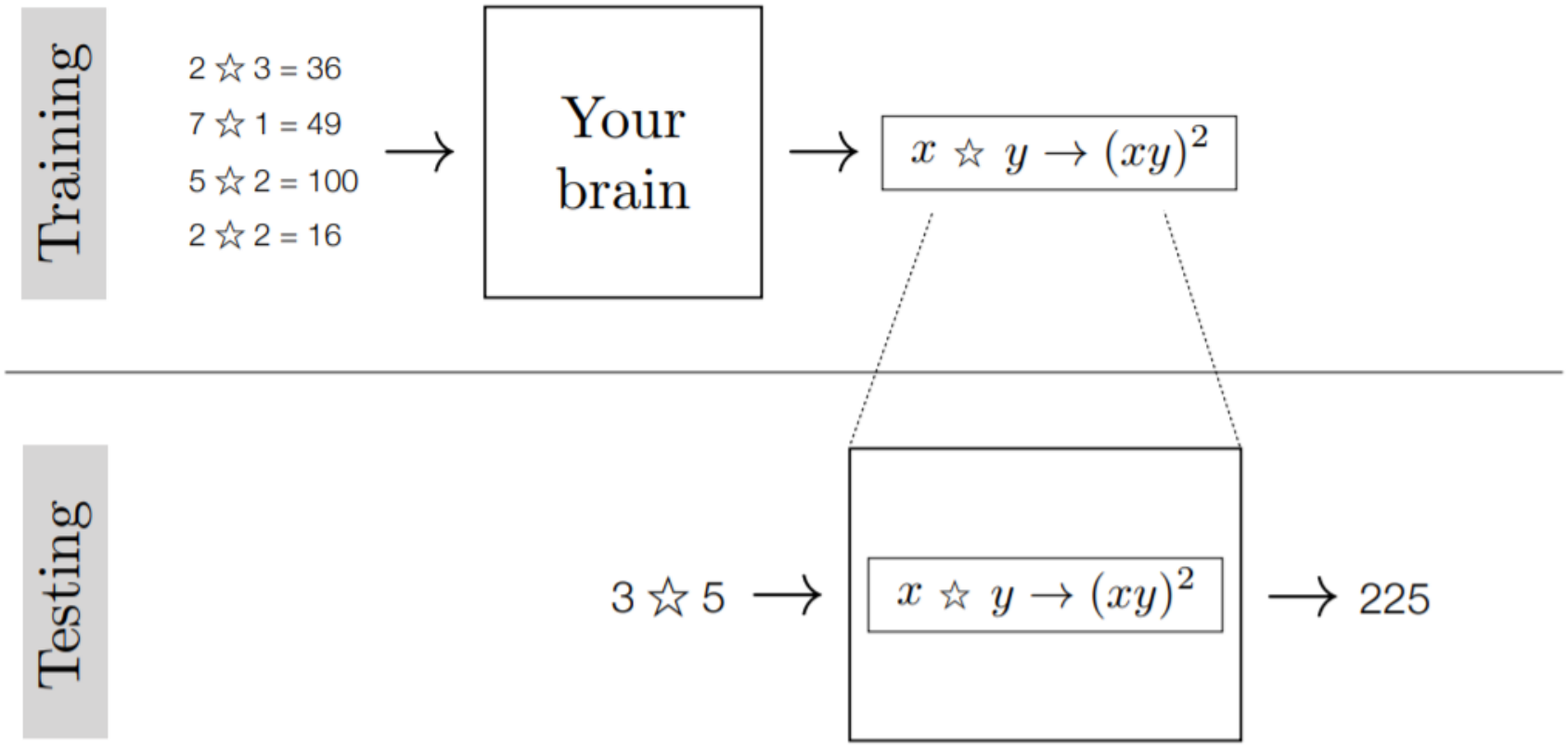
$7 \star 1 = 49$

$5 \star 2 = 100$

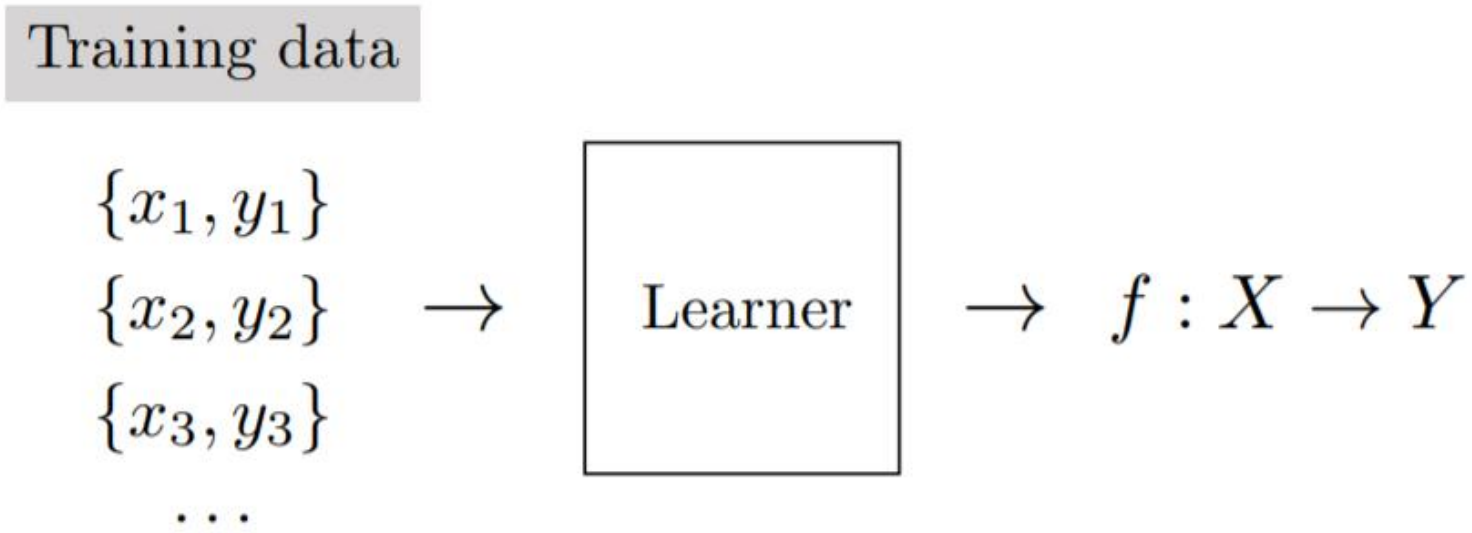
$2 \star 2 = 16$

$3 \star 5$??

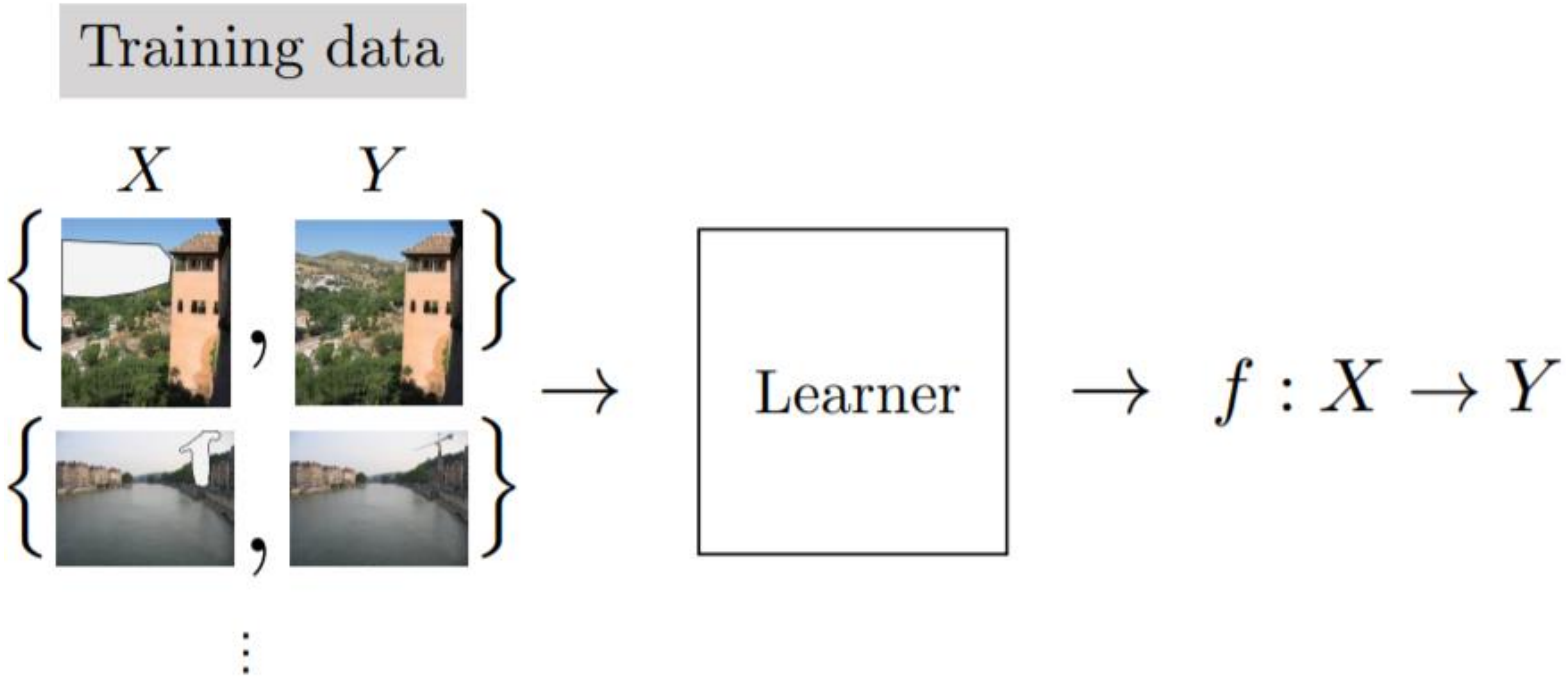
Learning from Examples



Learning from Examples

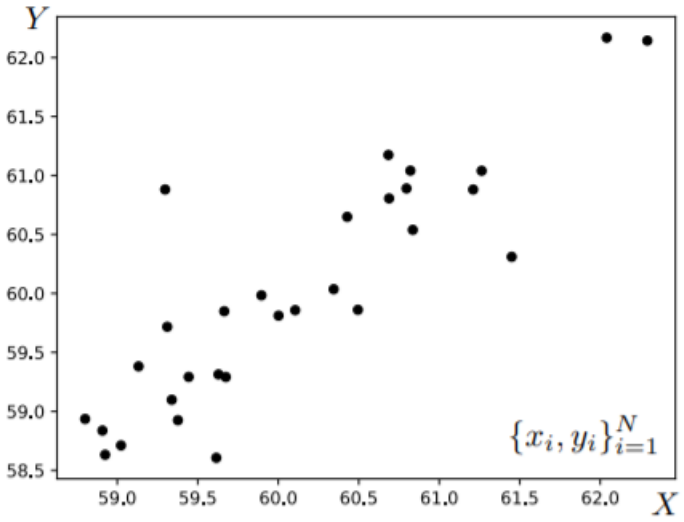


Using Training data for Image Inpainting



Learning from Data

Training data



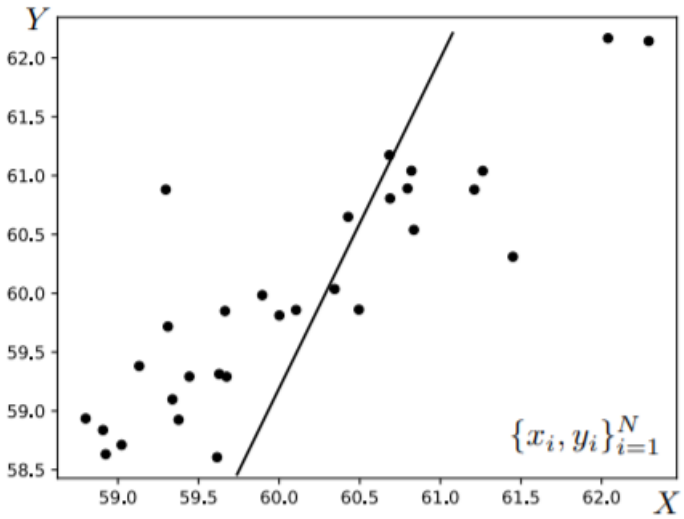
$\rightarrow$ Learner $\rightarrow$ $f_\theta(x) = \theta_1 x + \theta_0$

Hypothesis space

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

Learning from Data

Training data

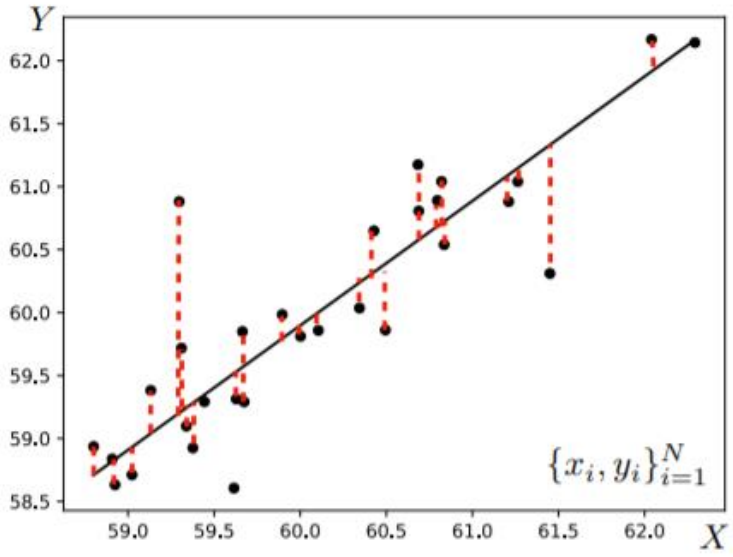


Search for the **parameters**, $\theta = \{\theta_0, \theta_1\}$, that best fit the data.

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

Learning from Data

Training data



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

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

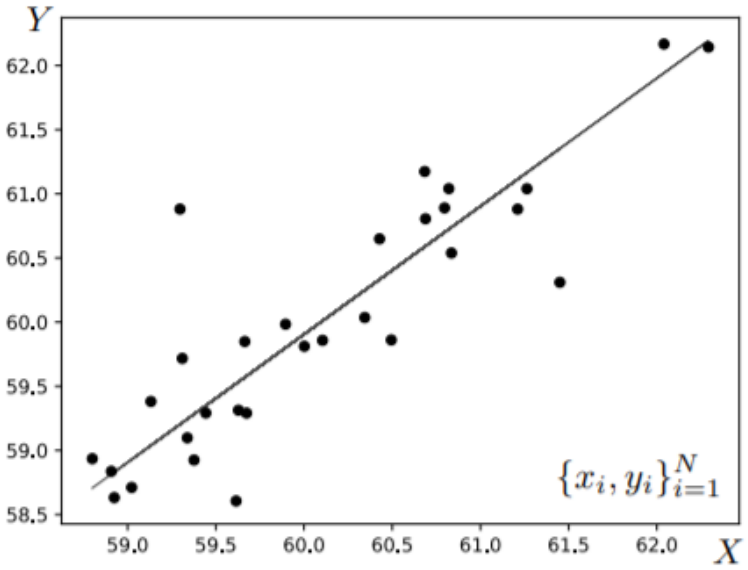
Best fit in what sense?

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

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

Learning from Data

Training data



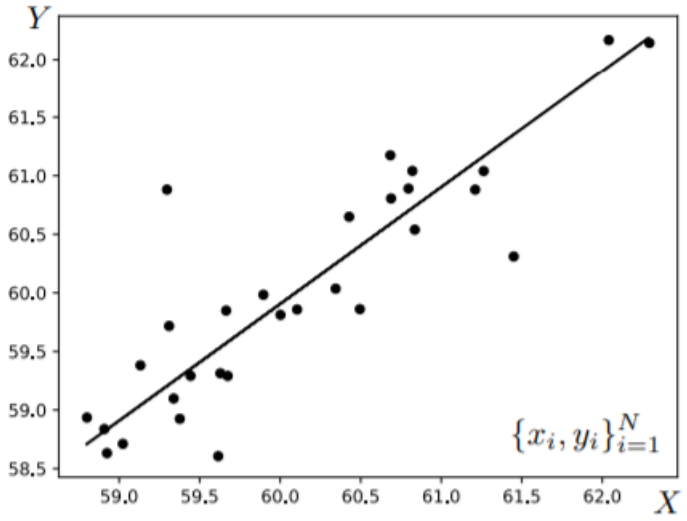
Complete learning problem:

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

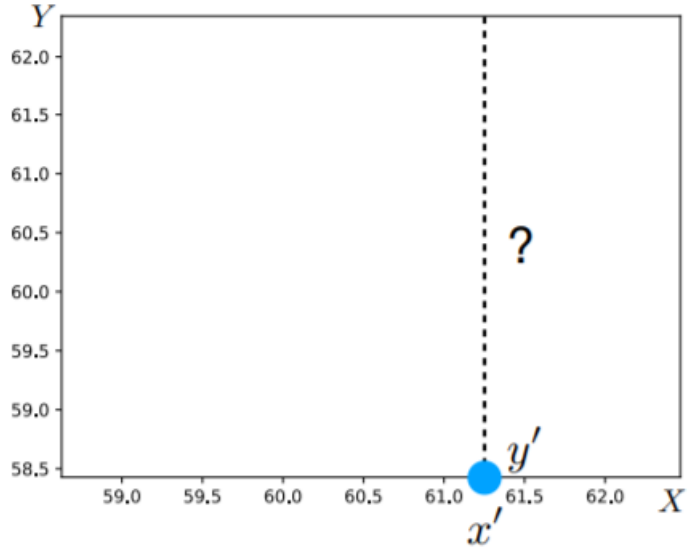
$$= \arg\min_\theta \sum_{i=1}^{N} (\theta_1 x_i + \theta_0 - y_i)^2$$

Learning from Data

Training data

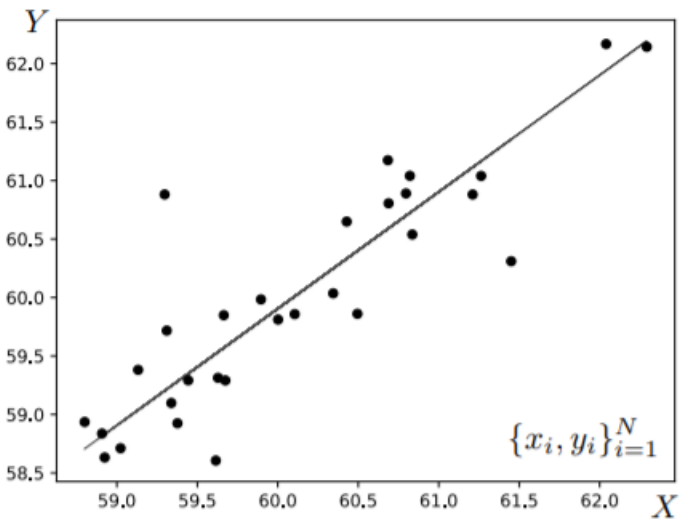


Test query

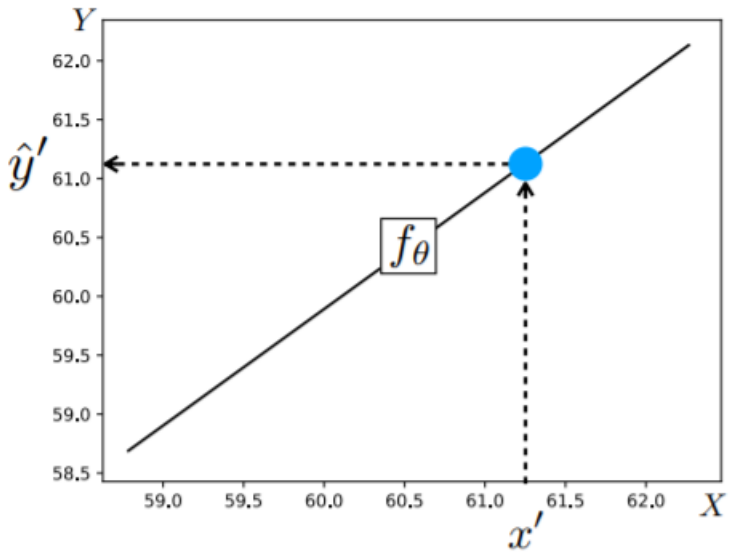


Learning from Data

Training data

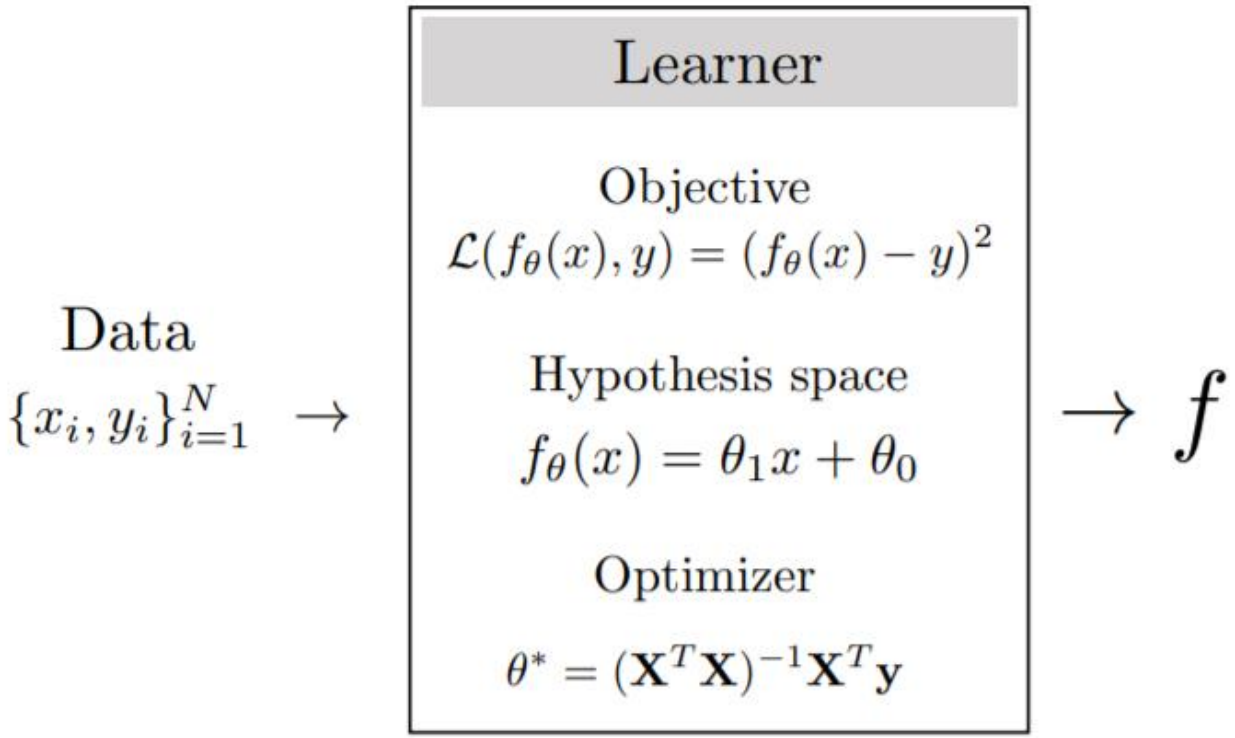


Test query



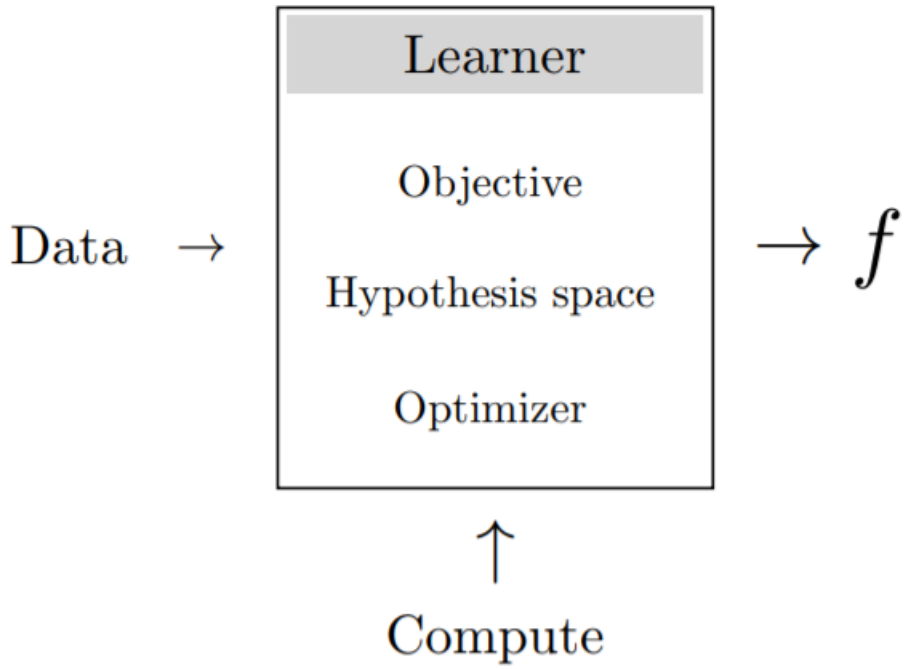
$$x' \dashrightarrow \boxed{f_\theta} \dashrightarrow \hat{y}'$$

Learning from Data



Objective function and hypothesis space in this example are simple.
A closed form solution is available and is given by the optimizer.

Learning from Data



Neural Networks

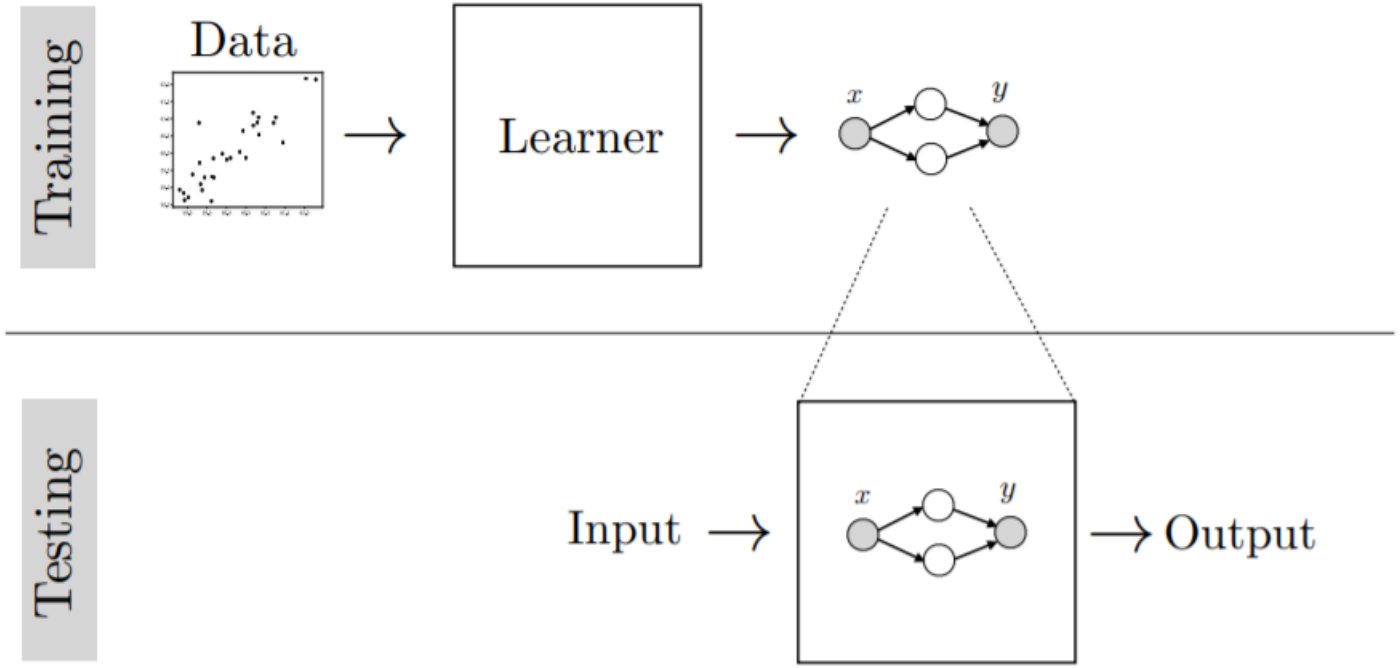


Image Classification

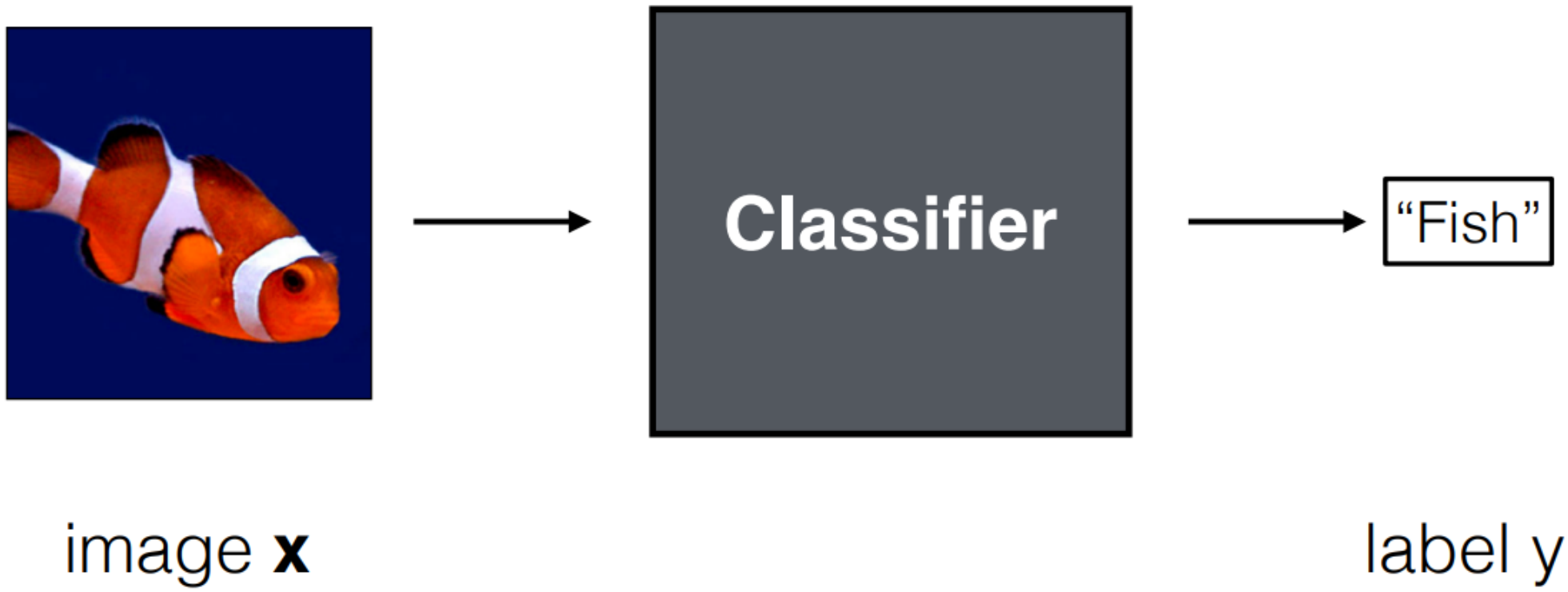


Image Classification

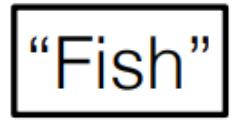
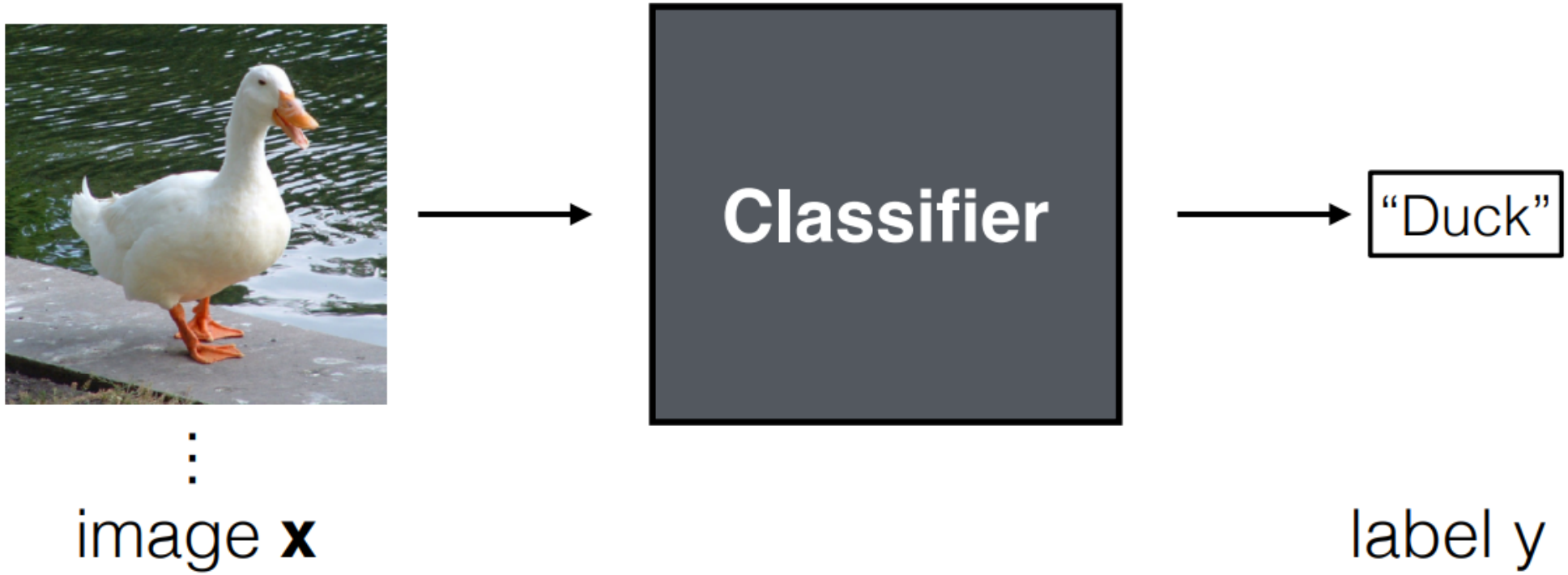


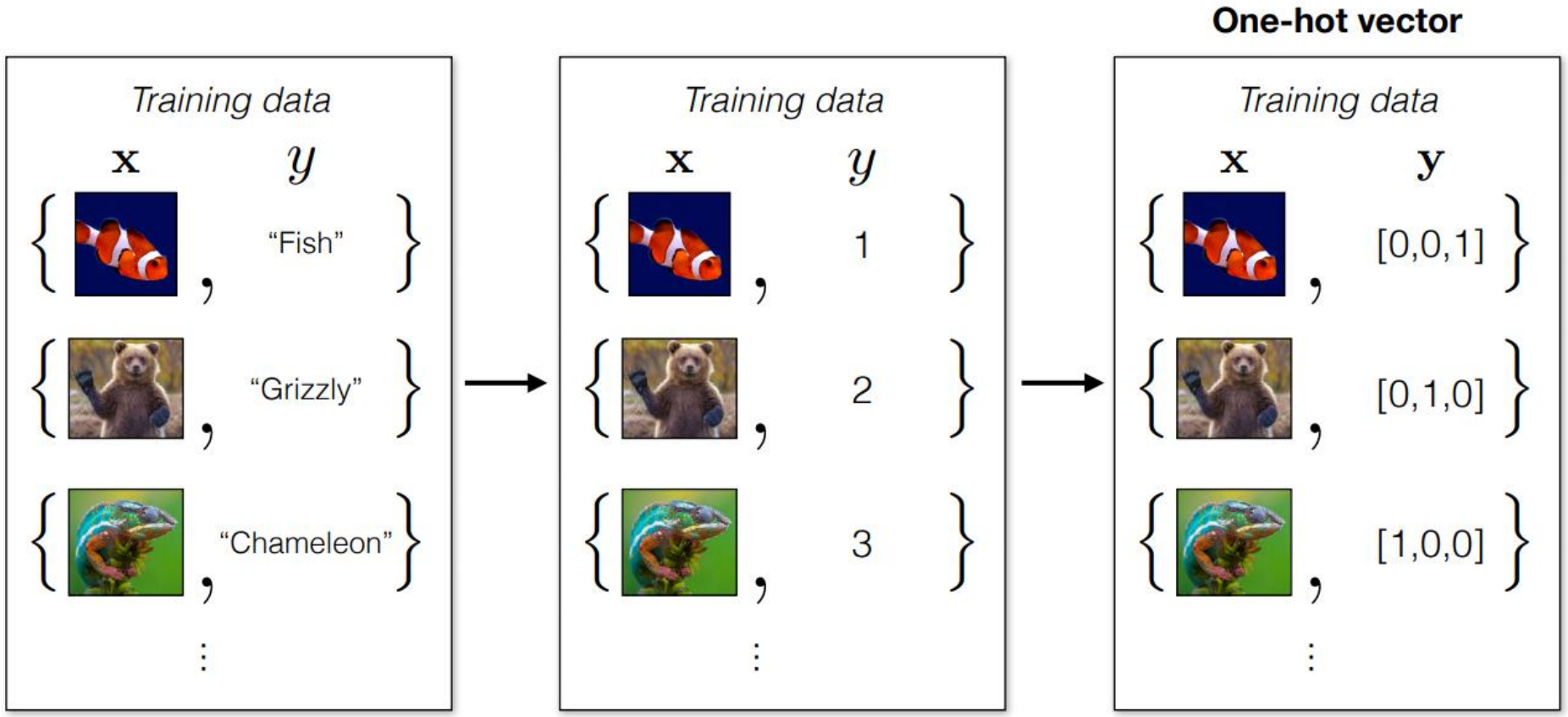
image **x**

label y

Image Classification

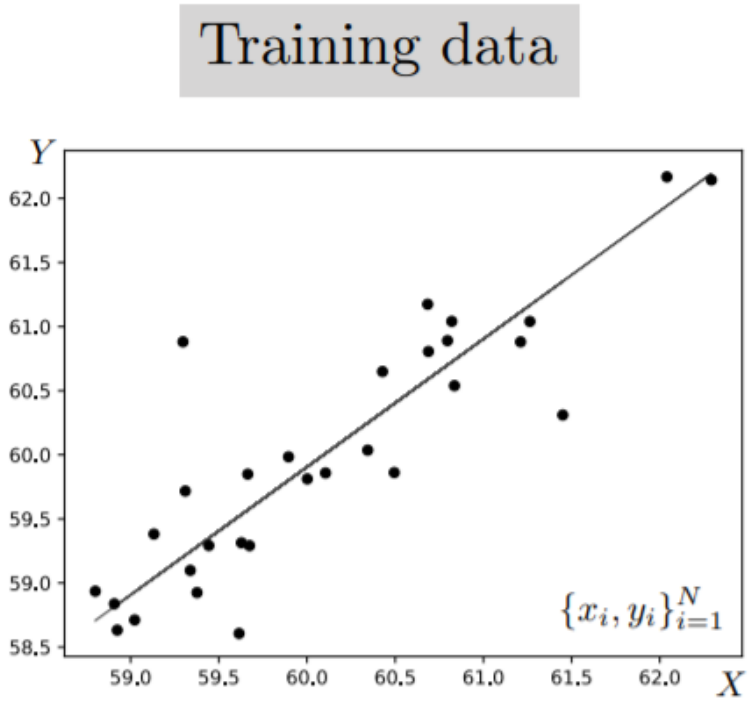


Encoding class labels



Loss Function

What loss function should we use in image classification?



Complete learning problem:

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

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

Least square was used to fit a line to data points (**linear regression**)

Loss Function

- In Image classification, one-hot encoding is used with a **cross-entropy** loss function.

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

← continuous, differentiable, convex

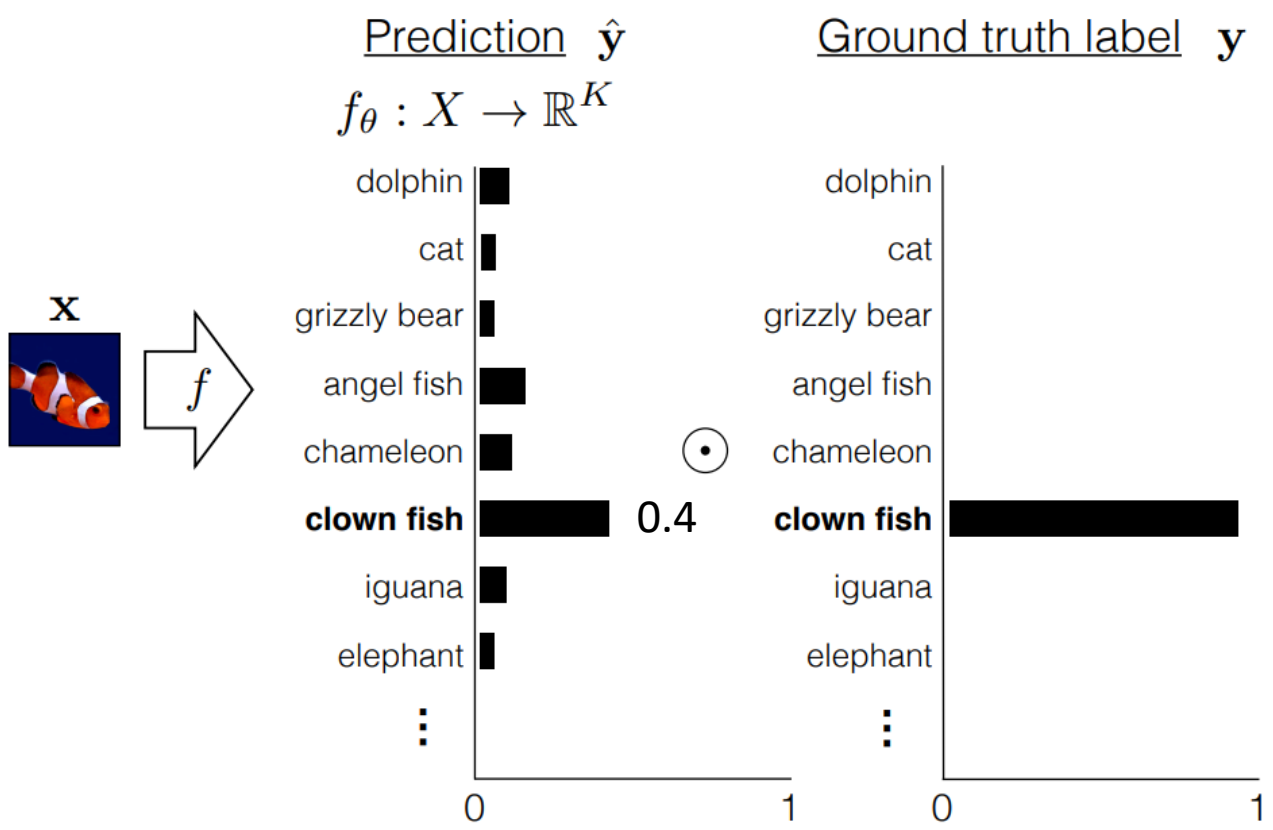
Learning from Data

Ground truth label $\mathbf{y}$

$\mathbf{x}$

[0,0,0,0,0,1,0,0,…]

Learning from Data

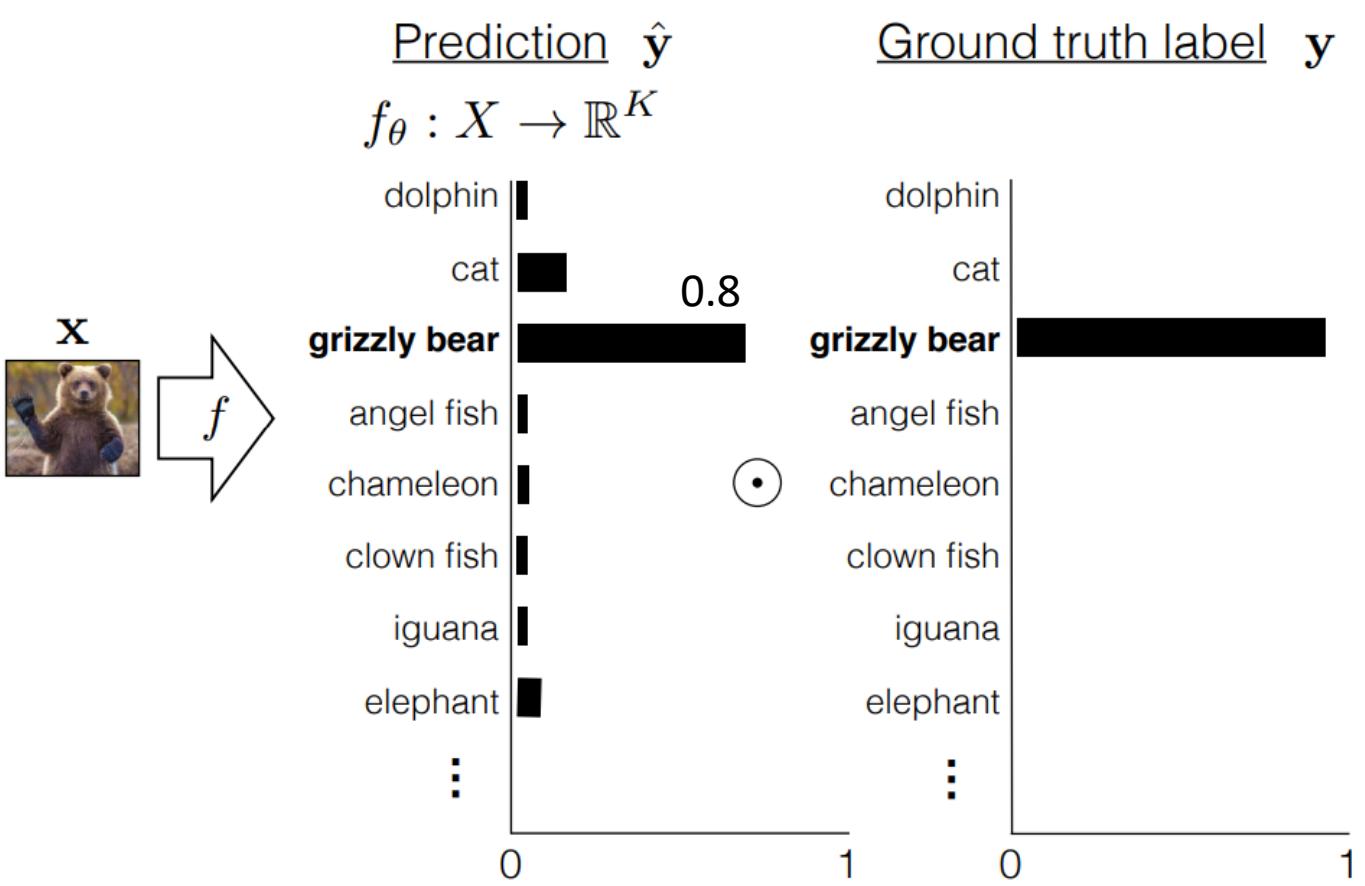


Loss

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

$H(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \log(0.4) = 0.4$ (k='clown fish')

Learning from Data

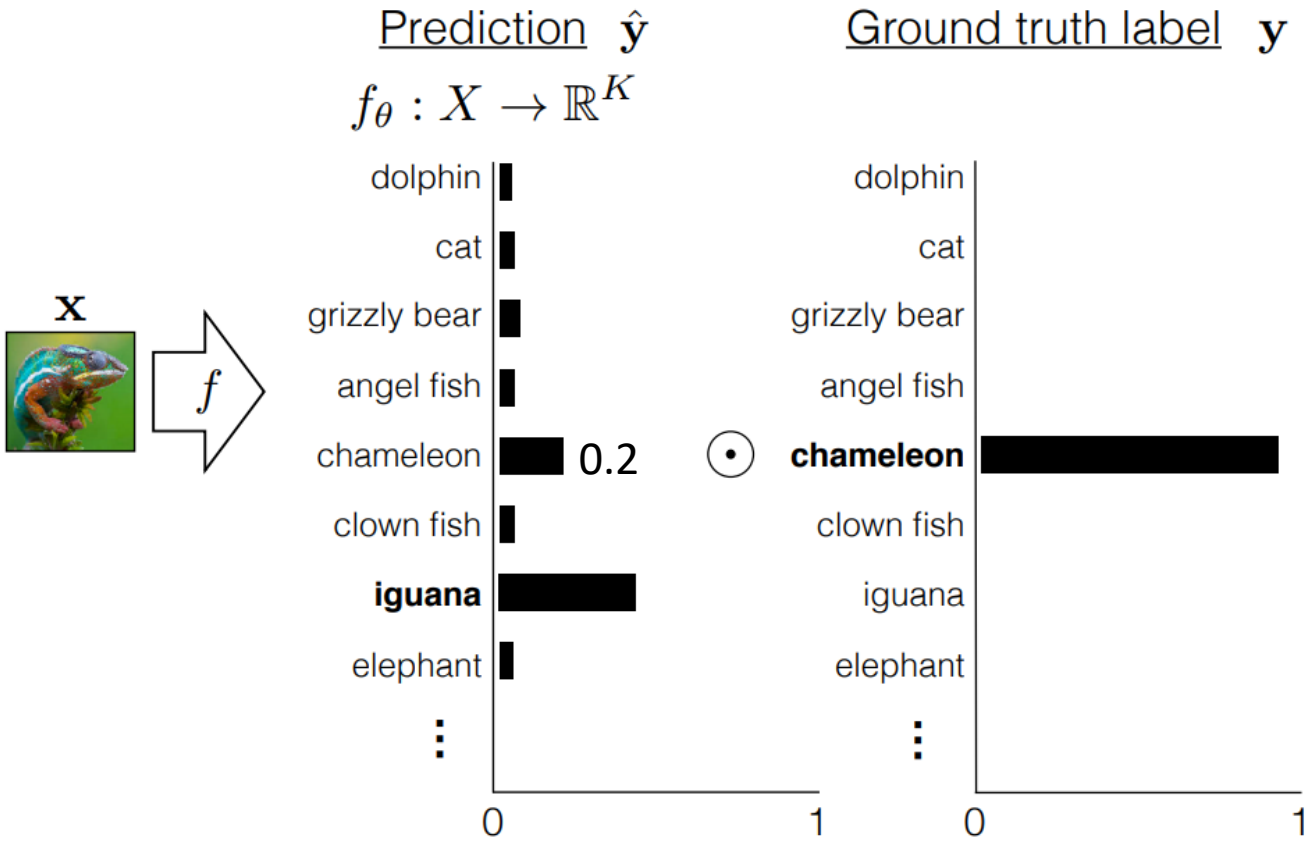


Loss

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

$H(\boldsymbol{y}, \hat{\boldsymbol{y}}) = 0.1$ (k='grizzly bear')

Learning from Data



Loss

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

$H(\boldsymbol{y}, \hat{\boldsymbol{y}}) = 0.7$ (k='chameleon')

Getting probabilities from regular scores

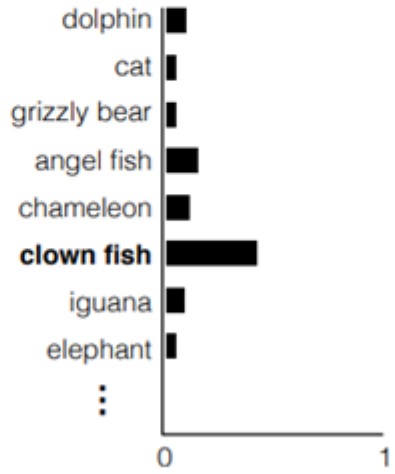
Assume we have a vector **z** of length K that indicates the score computed for each class

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

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

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

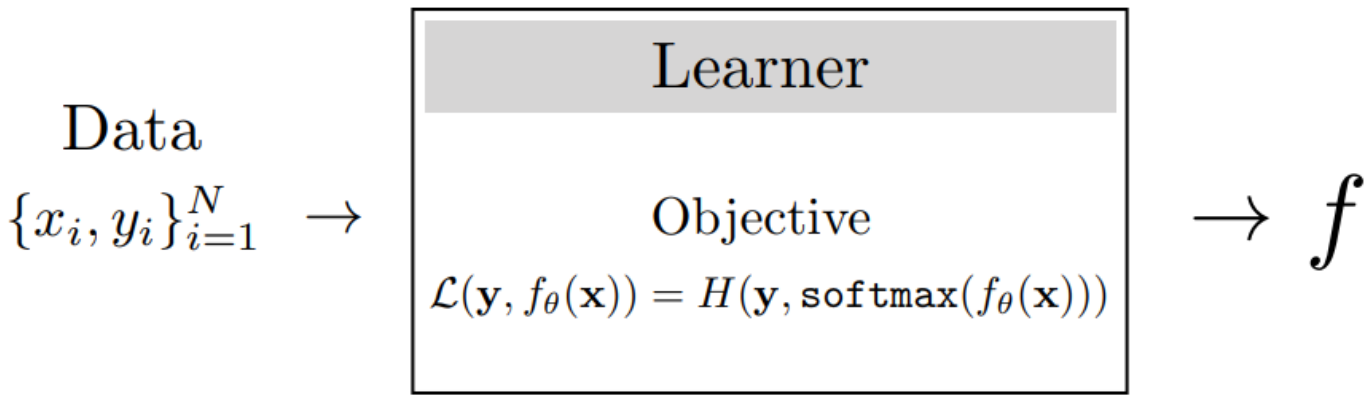


Softmax Regression

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

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

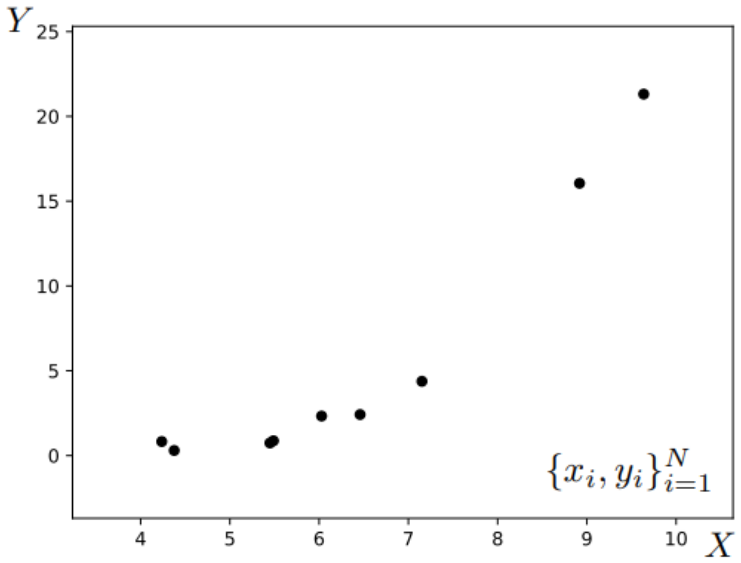
$$\hat{\mathbf{y}} = \texttt{softmax}(\mathbf{z})$$



Learning from Data

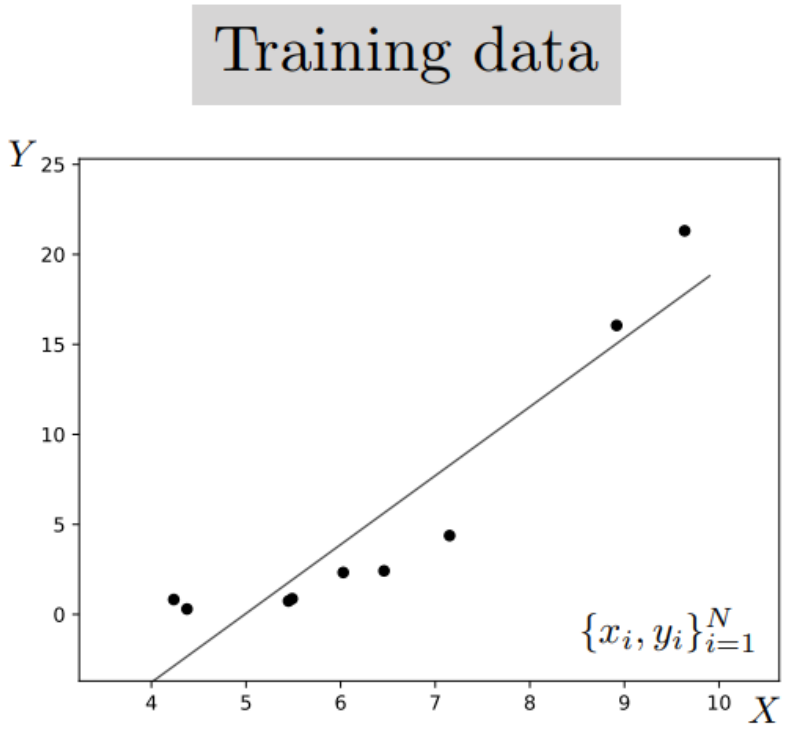
Linear regression

Training data



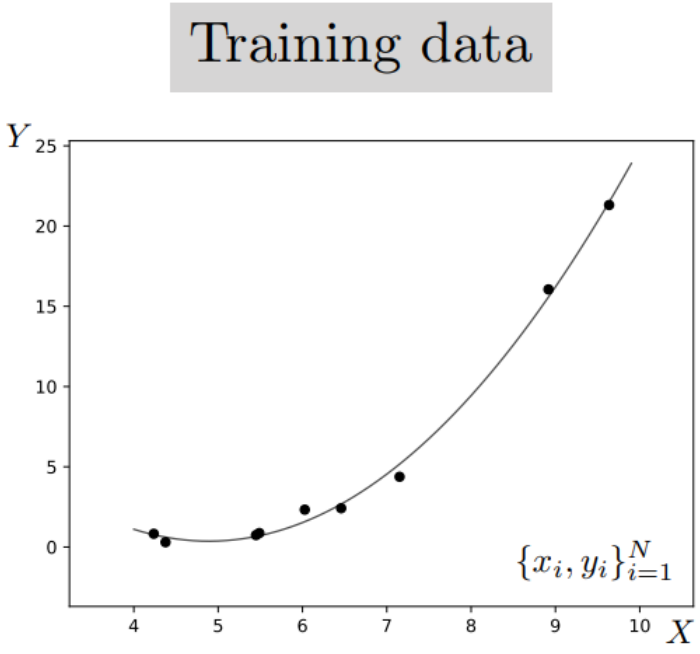
$$f_\theta(x) = \theta_0 + \theta_1 x$$

Linear Regression



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

Polynomial Regression (degree 2)



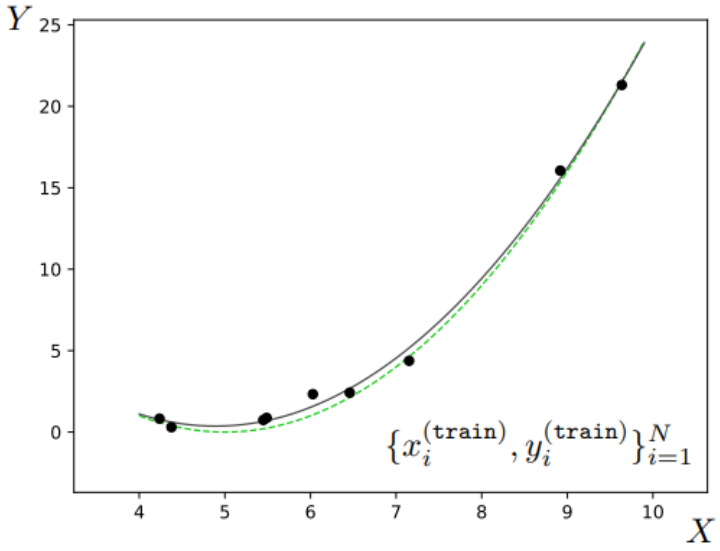
$$f_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

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

K-th degree polynomial regression

Polynomial Regression

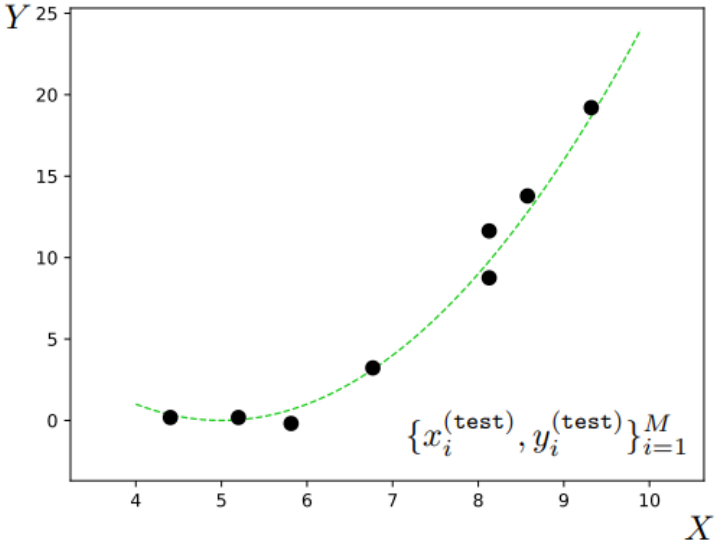
Training data



Training objective:

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

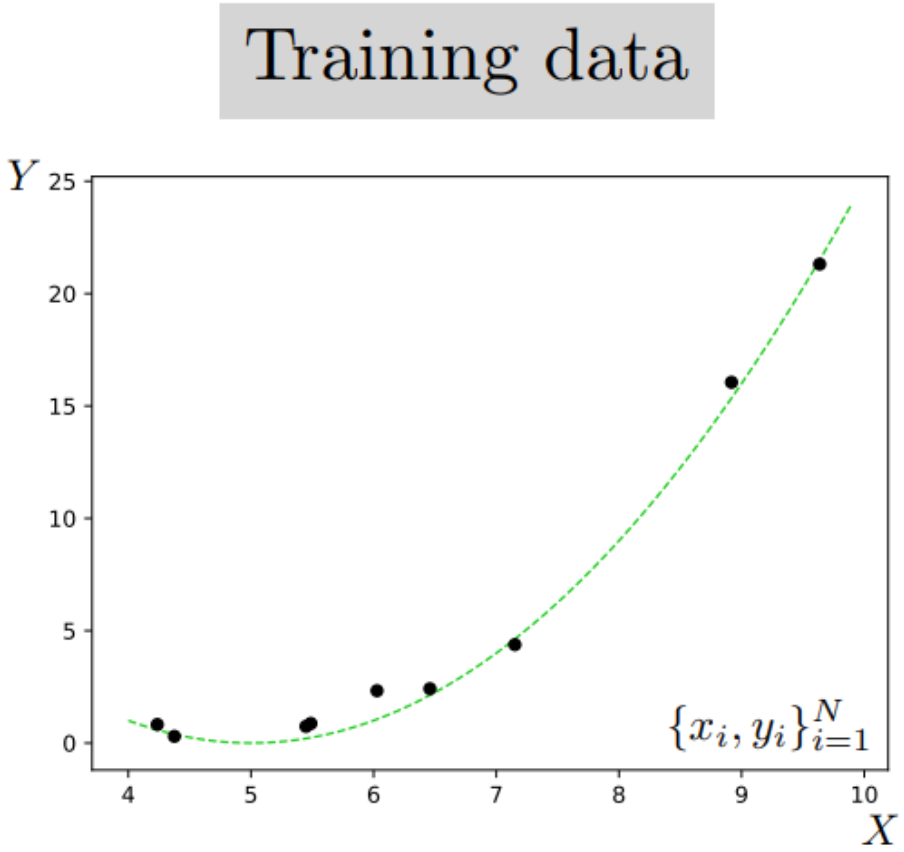
Test data



Test time evaluation:

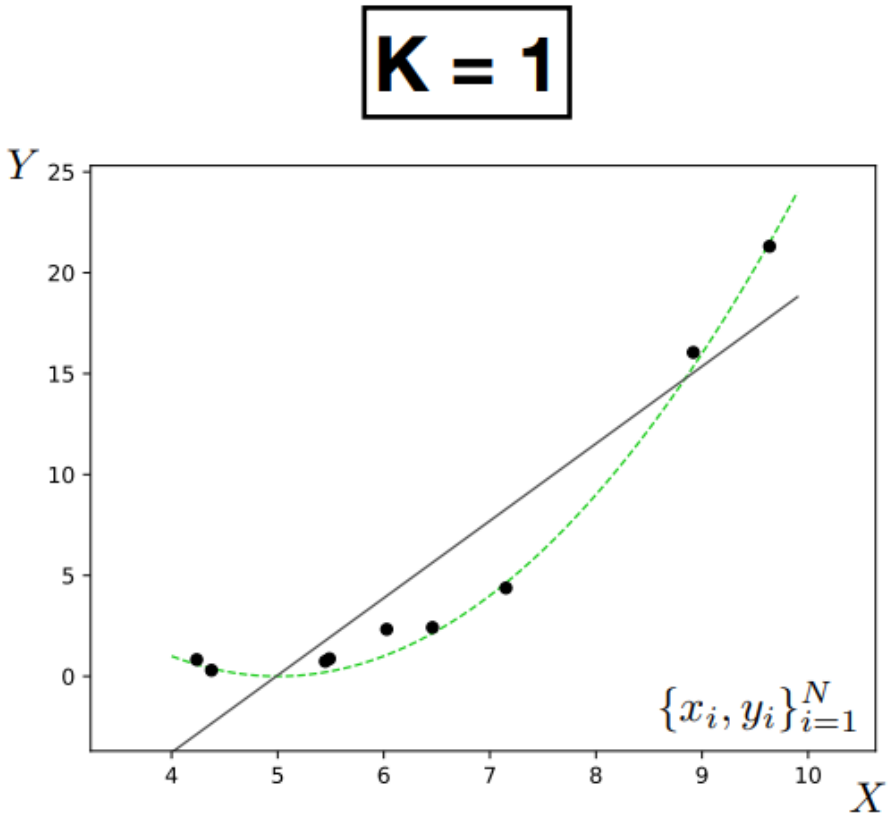
$$\sum_{i=1}^{M}(f_\theta(x_i^{\texttt{test}}) - y_i^{\texttt{test}})^2$$

What happens if we increase the order?



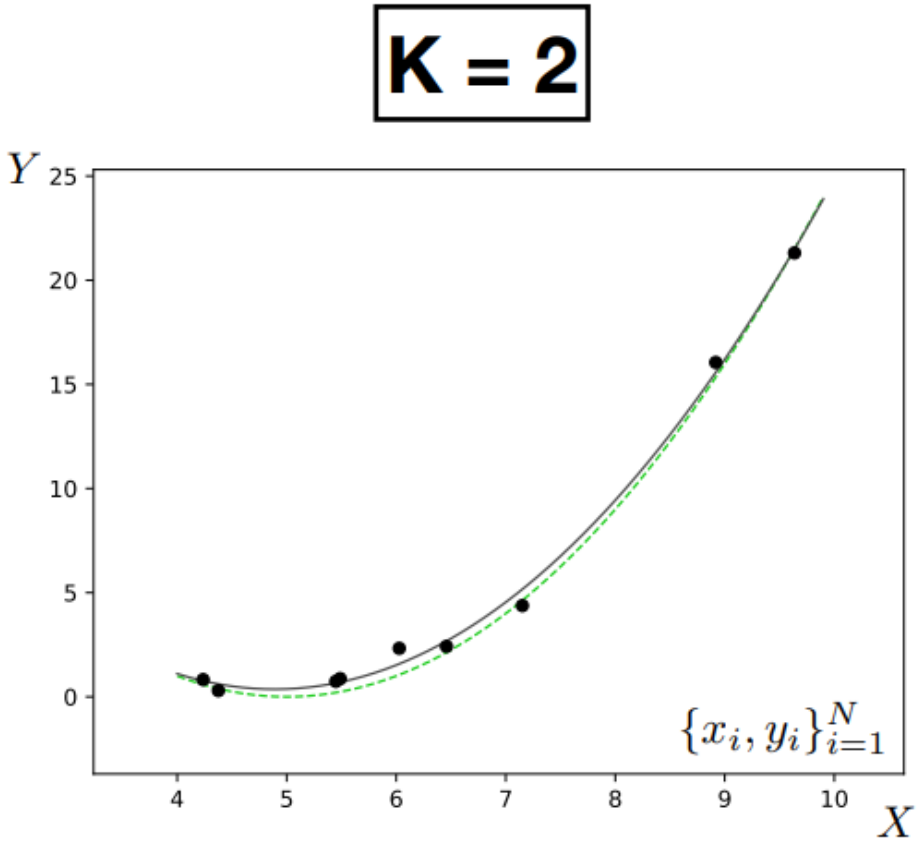
$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

Overfitting



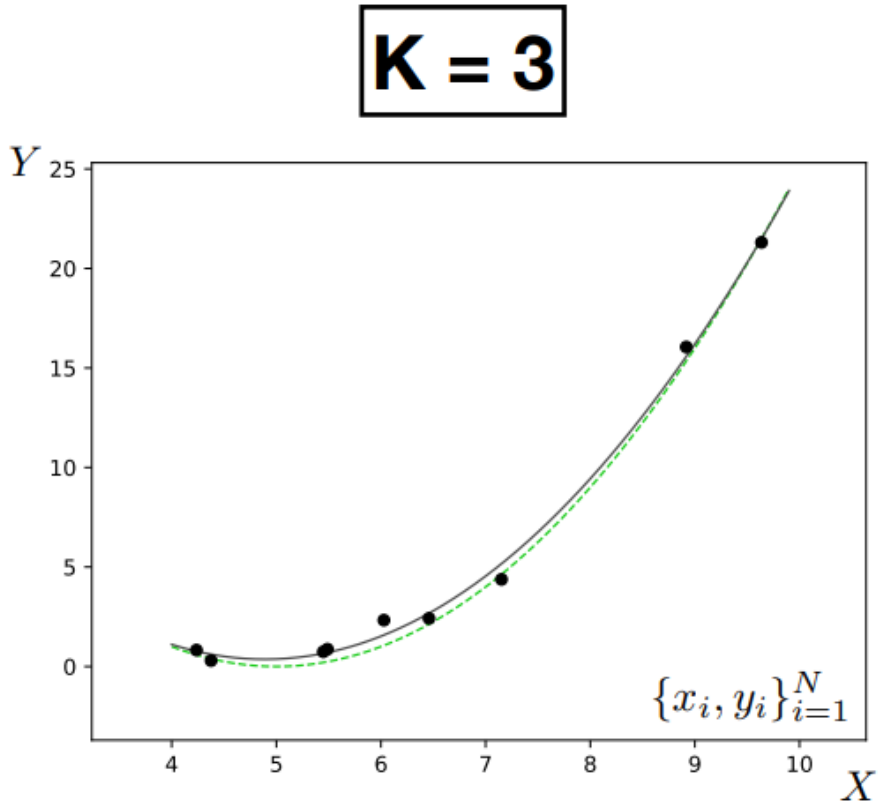
$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

Overfitting



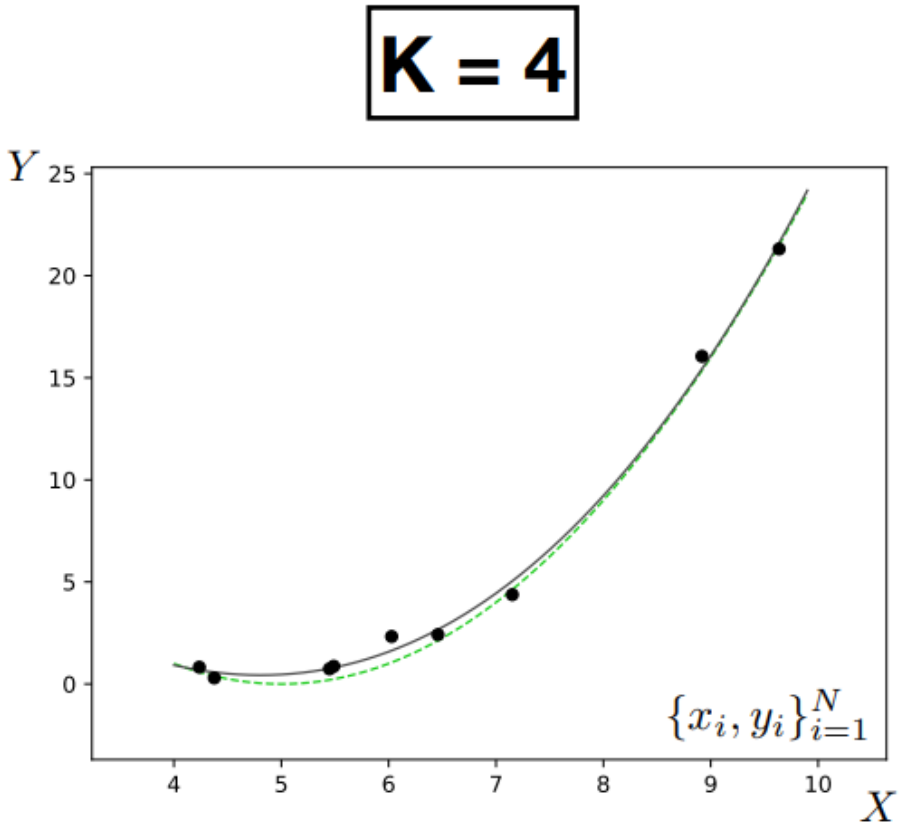
$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

Overfitting



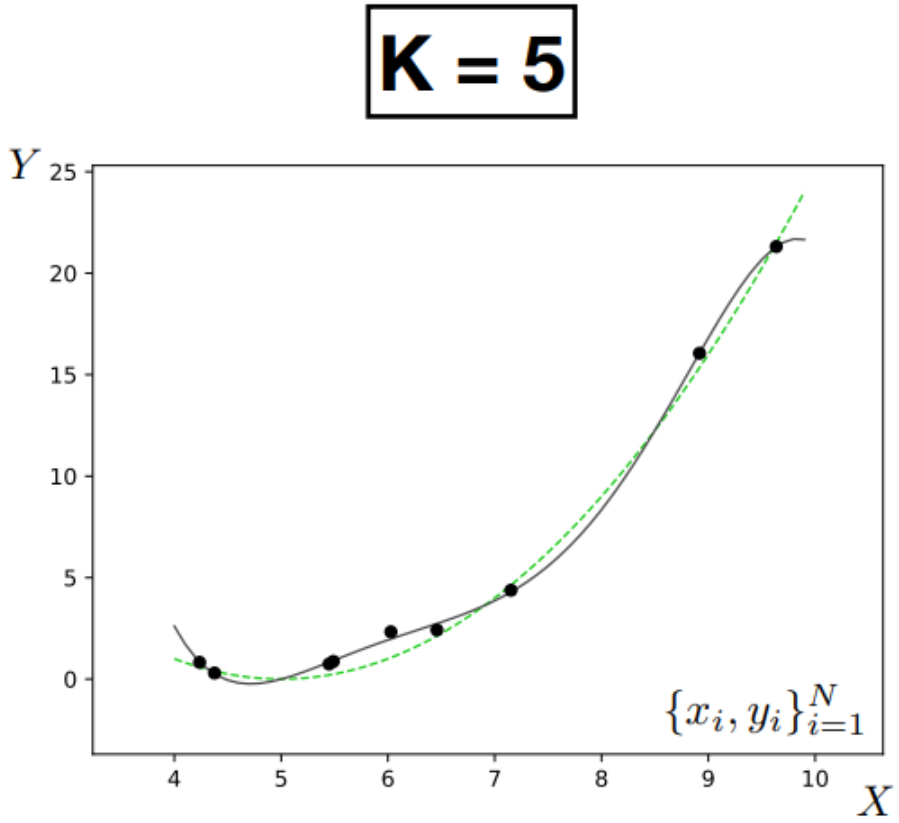
$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

Overfitting



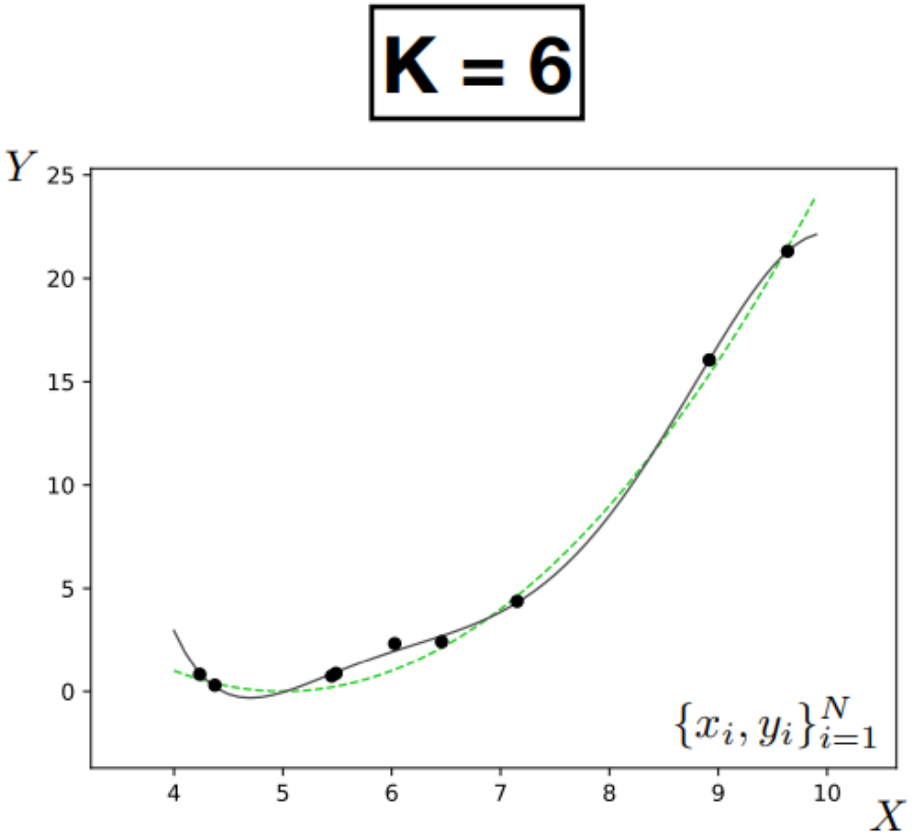
$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

Overfitting



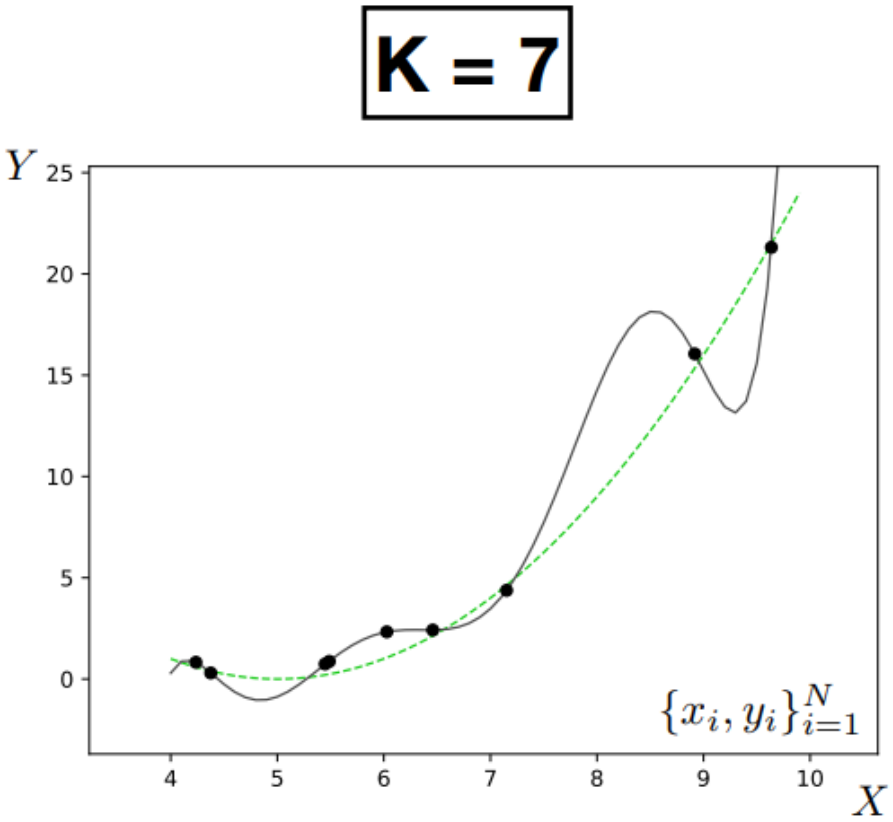
$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

Overfitting



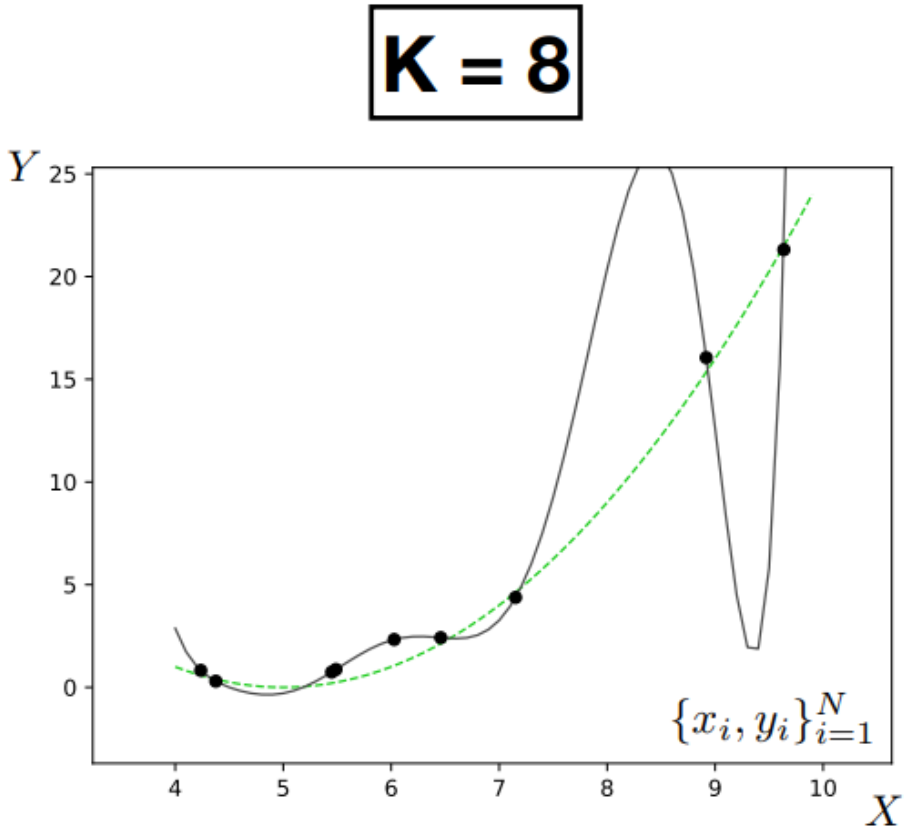
$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

Overfitting



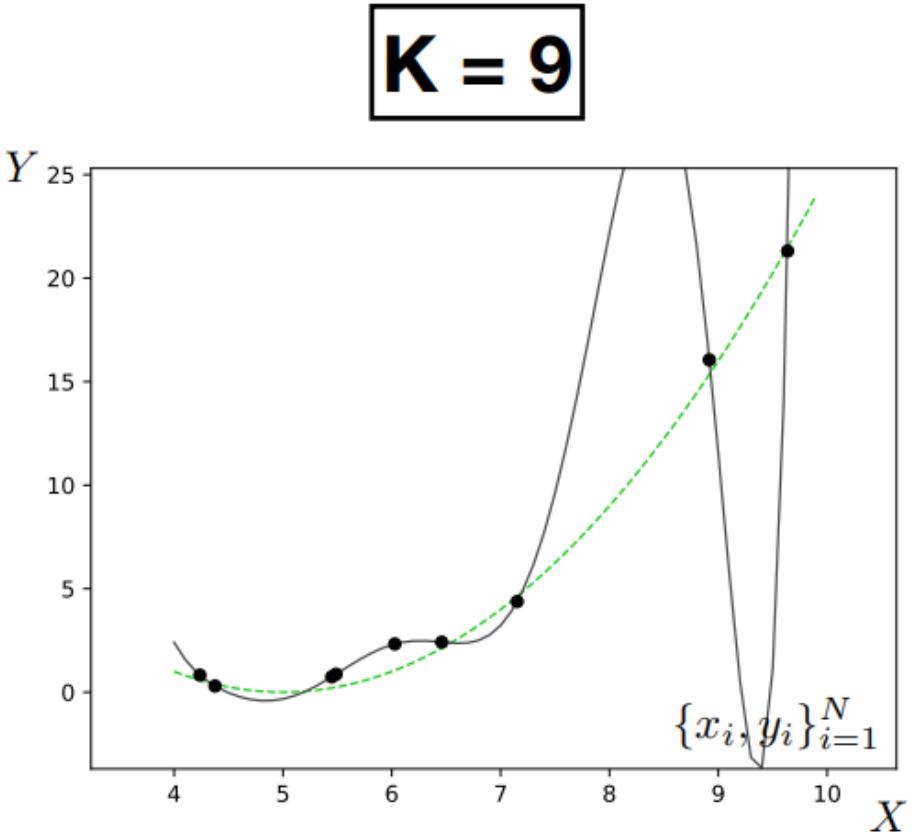
$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

Overfitting



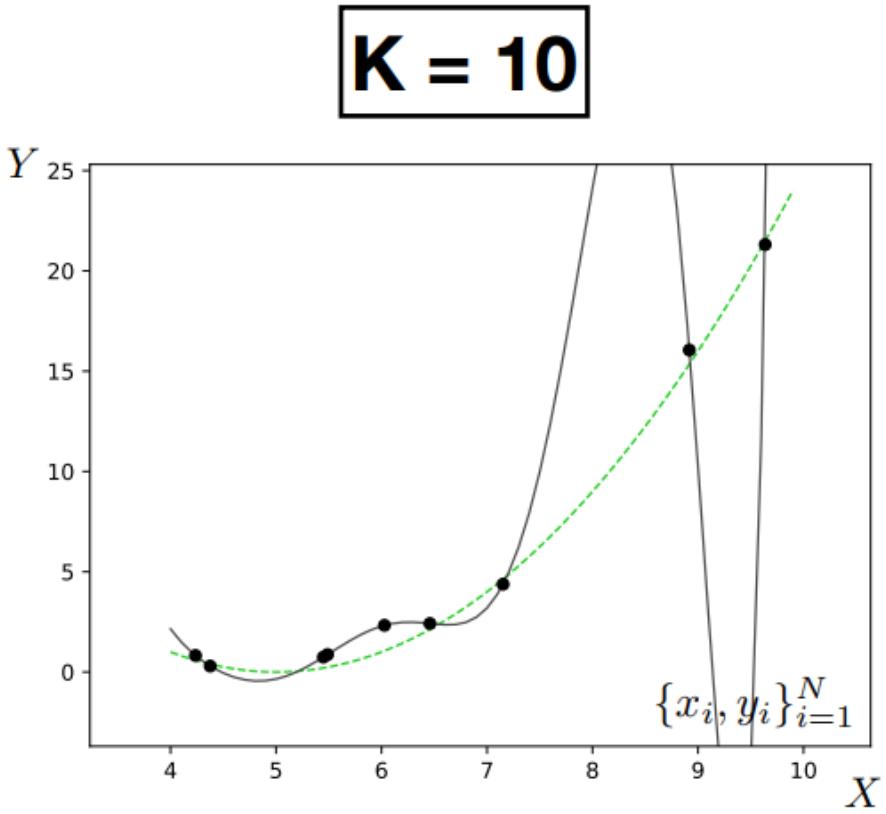
$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

Overfitting



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

Overfitting

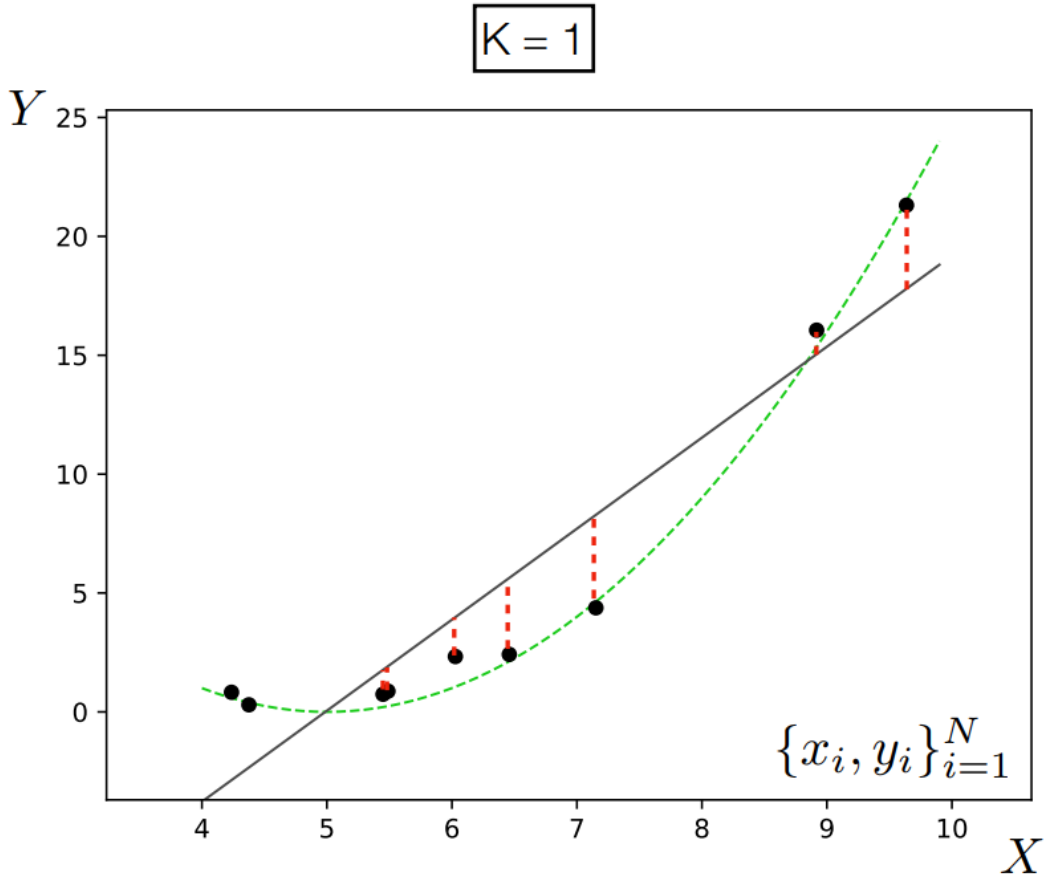


$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

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.

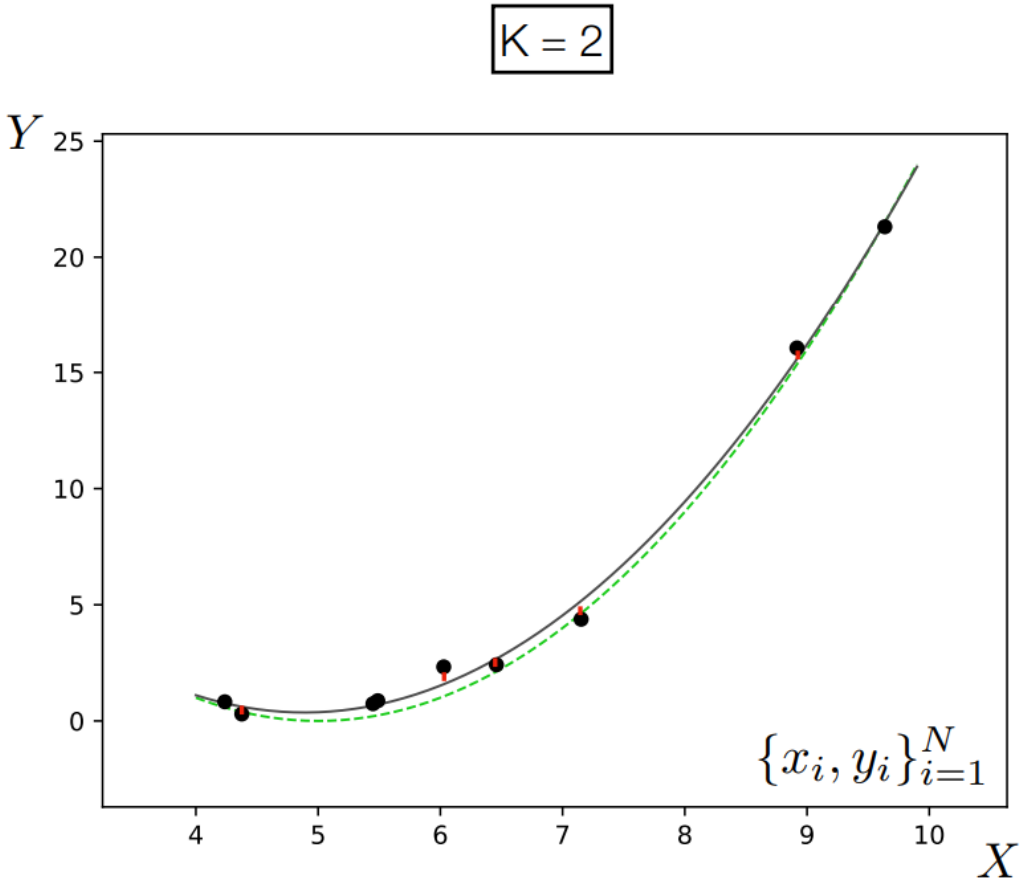
Underfitting



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

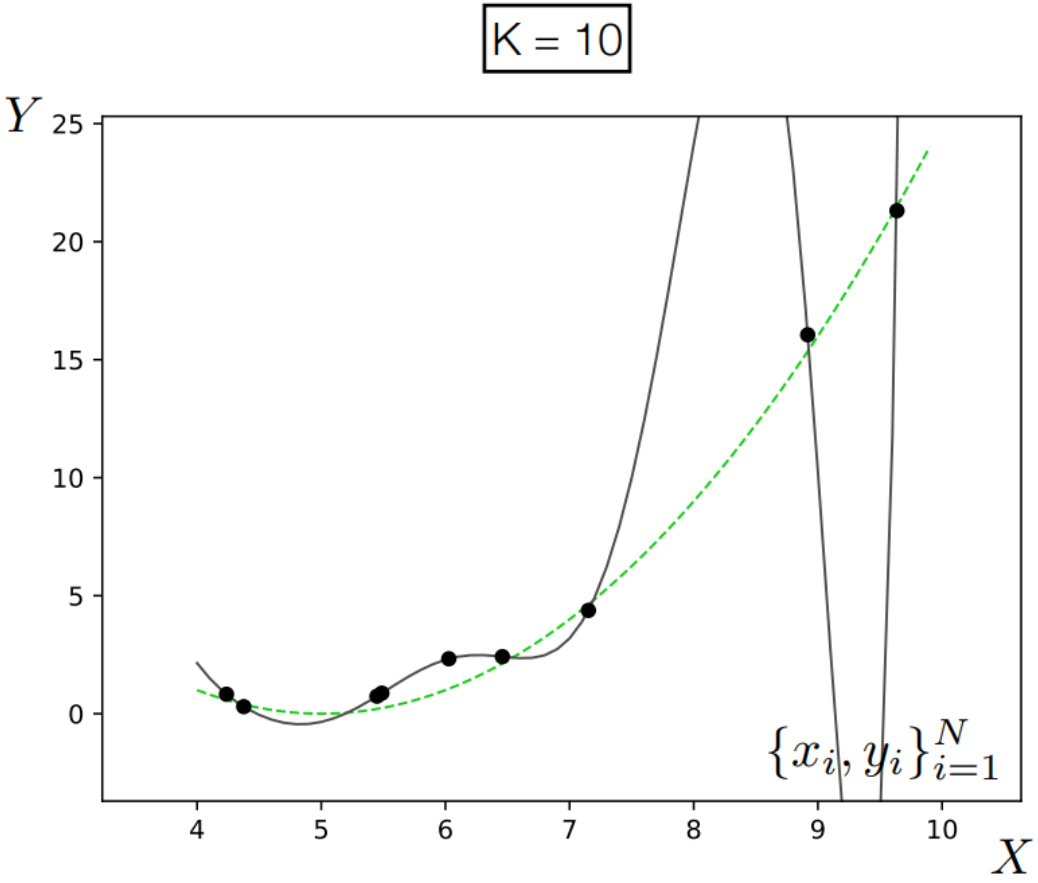
An underfit model will have high error on the training points. This error is known as **approximation error**.

Appropriate Model



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

Overfitting

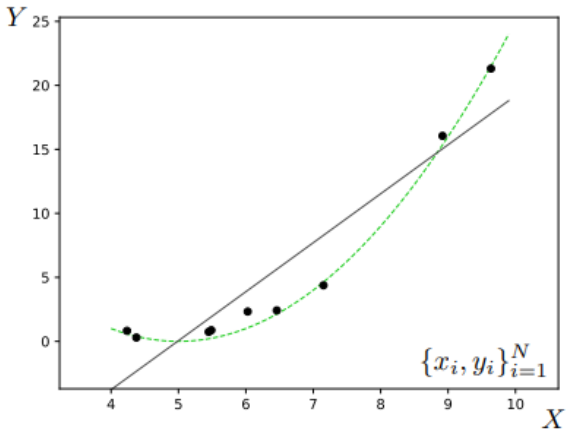


Now we have zero approximation error — the curve passes exactly through each training point.

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

Underfitting

K = 1

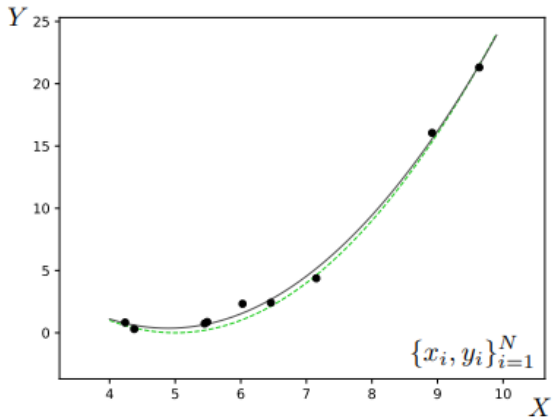


High error on train set

High error on test set

Appropriate model

K = 2

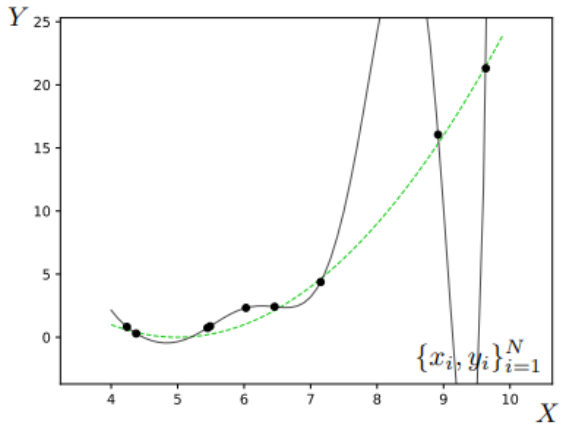


Low error on train set

Low error on test set

Overfitting

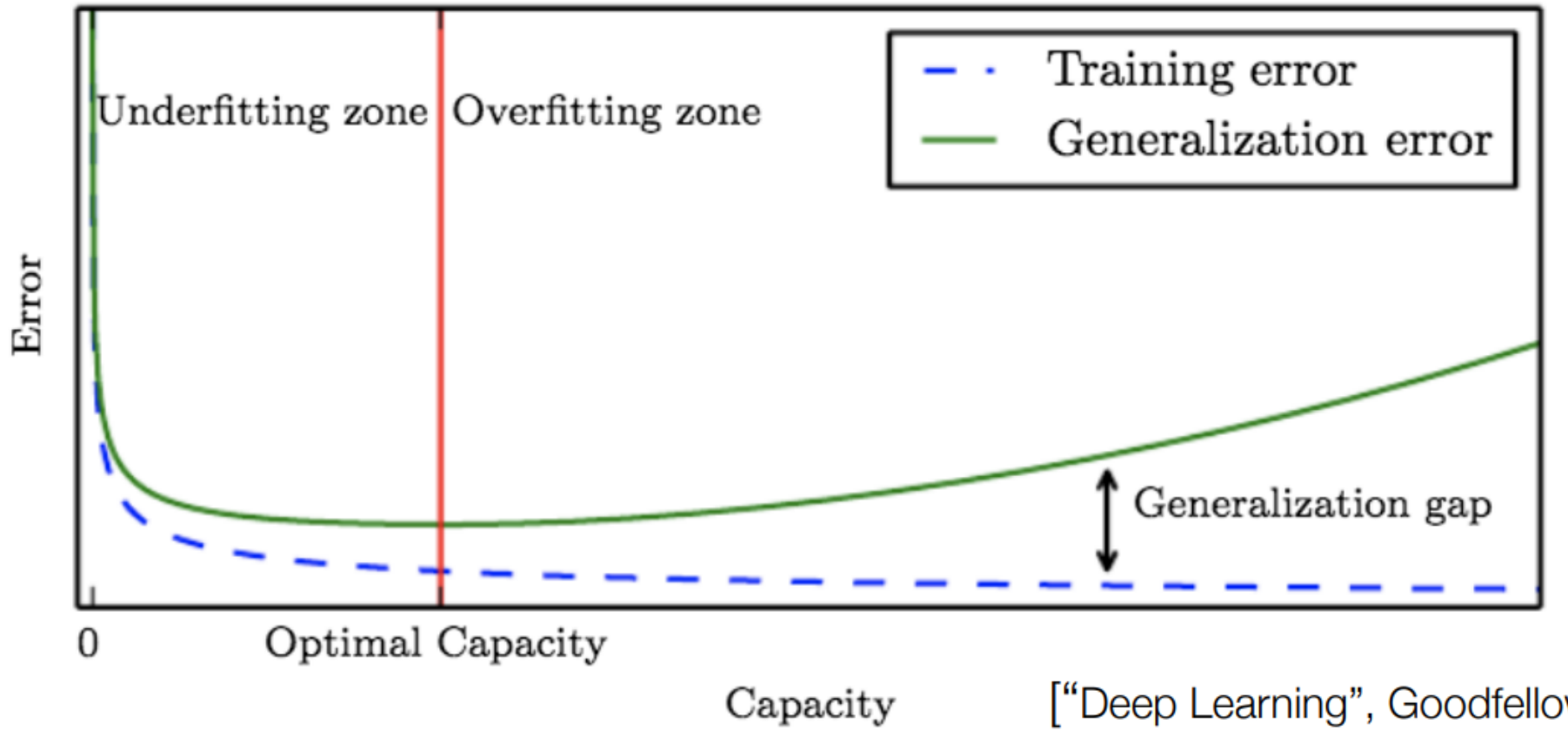
K = 10



Lowest error on train set

High error on test set

Generalization gap



["Deep Learning", Goodfellow et al.]

To reduce overfitting (generalization gap)

- We need to control the **capacity** of the model (e.g., use the appropriate number of free parameters.
- Complex models with many free parameters have high capacity
- Simple models have low capacity.

Fitting just right

- Underfitting?
    1. add more parameters (more layers, etc.)
- Overfitting?
    1. remove parameters
    2. Increasing training data
    3. add regularizes

Regularization

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

- The function $R(\theta)$ is selected to enforce lower magnitude of parameters.
- This is achieved by adding a positive term that penalizes high magnitude parameters.

Regularized least squares

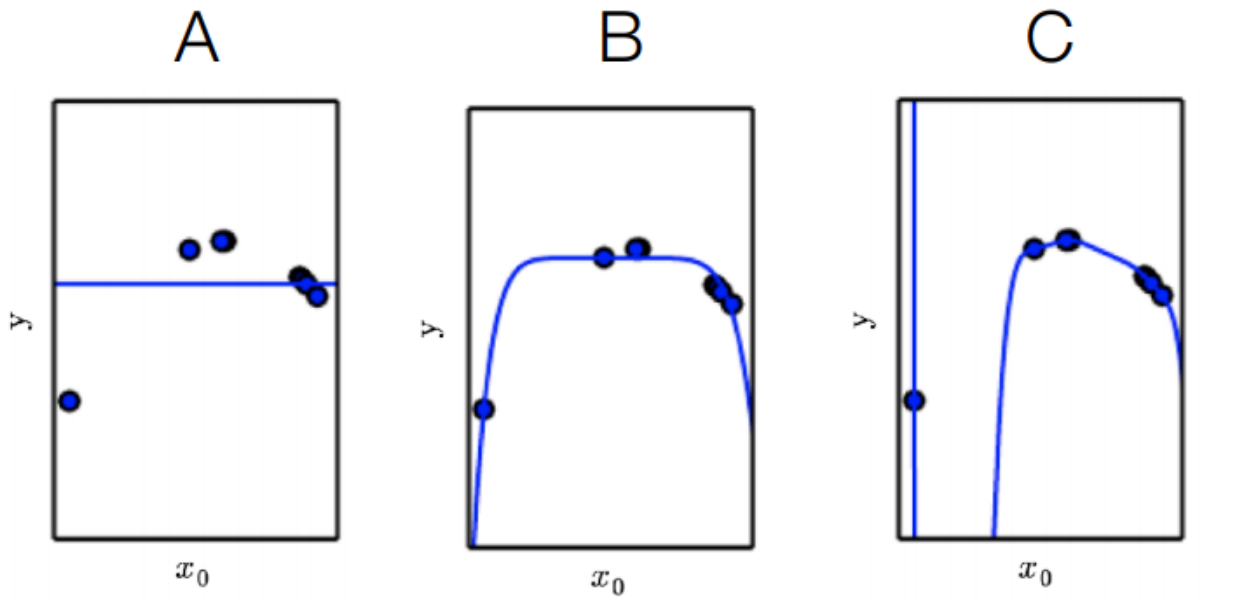
$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

$$R(\theta) = \lambda \left|\left|\theta\right|\right|_2^2$$ ⟵ Only use polynomial terms if you really need them! Most terms should be zero

Regularized least squares

Assume that K (polynomial order) is large compared to the number of points (training data)

$$\theta^* = \arg\min_{\theta} \sum_{I=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i) + \lambda \|\theta\|_2^2$$



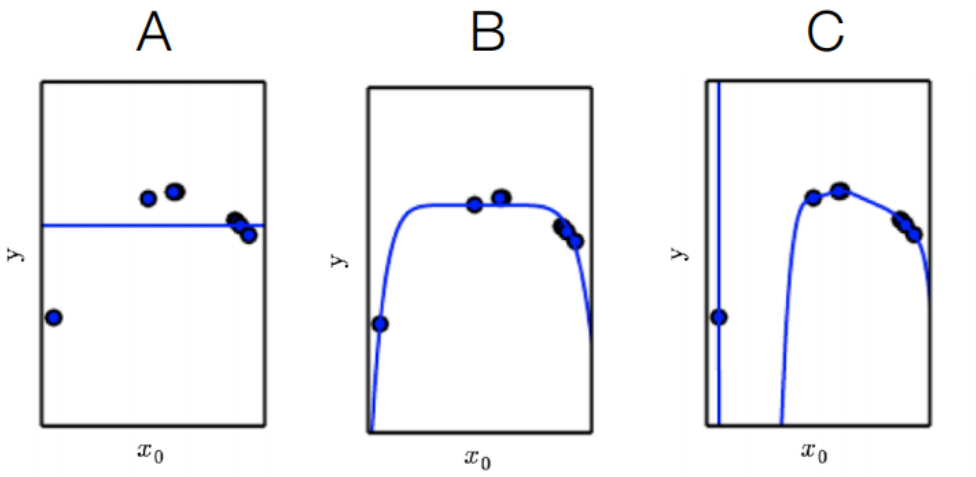
Low λ — ?

Medium λ — ?

High λ — ?

Regularized least squares

$$\theta^* = \arg\min_{\theta} \sum_{I=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i) + \lambda \|\theta\|_2^2$$



Low λ — C

Medium λ — B

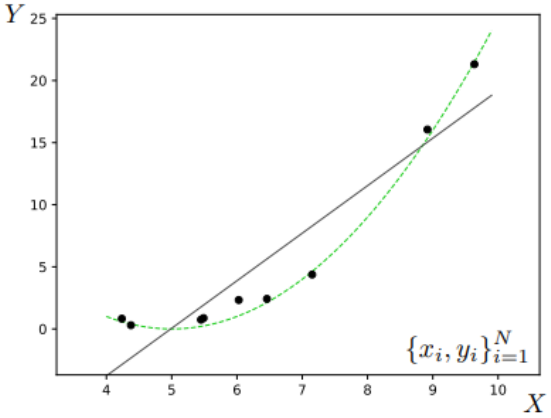
High λ — A

[Adapted from "Deep Learning", Goodfellow et al.]

Regularized least squares

Underfitting

K = 1

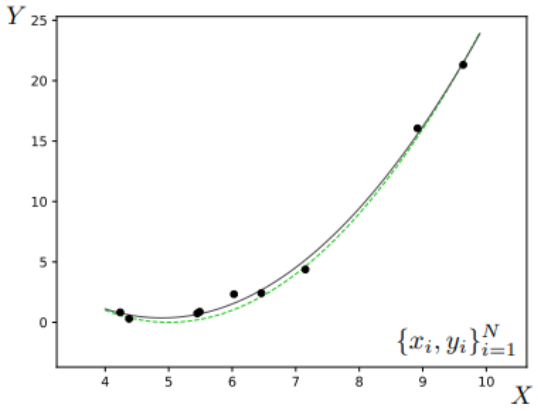


Simple model

Doesn't fit the training data

Appropriate model

K = 2

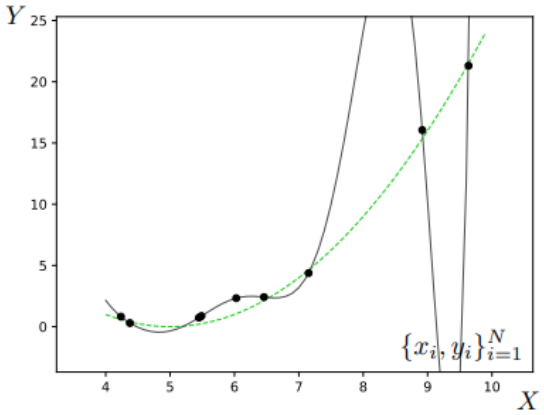


Simple model

Fits the training data

Overfitting

K = 10



Complex model

Fits the training data