

# Introduction to ML

**The 16th CHPC & 8th NITheCS Coding Summer School  
on Data Science and Computational Thinking  
26 January - 6 February 2026**

# What is Machine Learning?

“Learning is any process by which a system improves performance from experience.”

**Herbert Simon**





Machine Learning is the study of algorithms that

- improve their performance  $P$
- at some task  $T$
- with experience, or “training data”  $\mathcal{D}$

Tom Mitchell (1951-)

A well-defined learning task is given by  $(P, T, \mathcal{D})$

Examples:

$T$ : Classify emails as legitimate or spam

$P$ : Percentage of emails labeled correctly

$\mathcal{D}$ : Repository of emails, some with  
human-specified labels

$T$ : Playing Chess (or Go)

$P$ : Percent games won against an  
opponent

$\mathcal{D}$ : Playing games against itself

**But what does this algorithm *do* with the data?**

# What is Machine Learning?

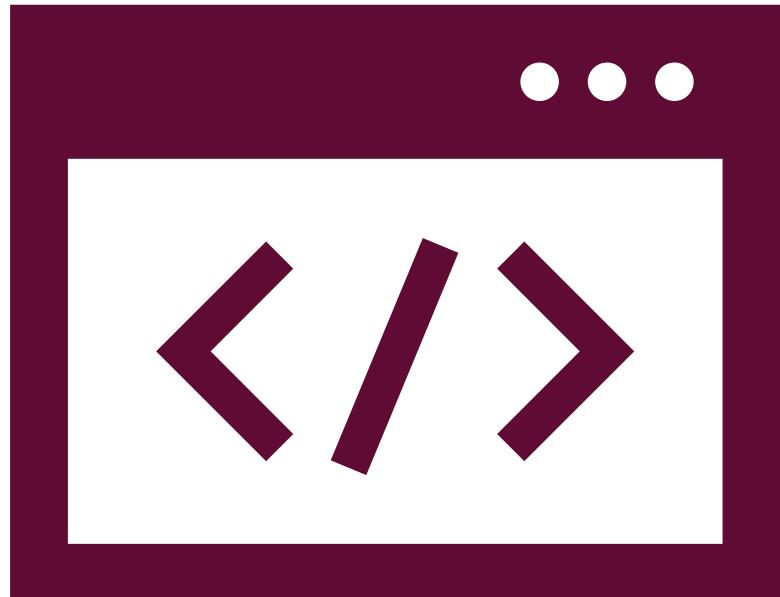
“Machine learning ... gives computers the ability to learn without being explicitly programmed.”

**Arthur Samuel**

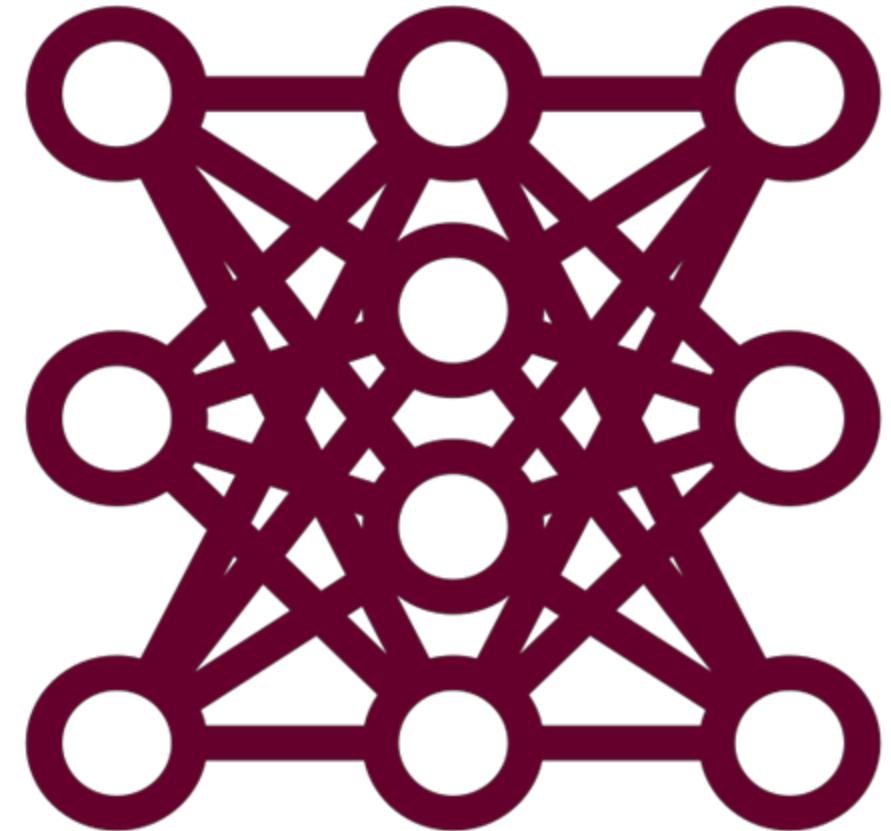


# Machine learning is Programming 2.0

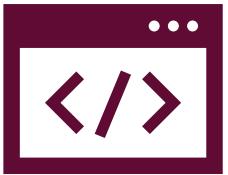
Traditional Programming



Machine learning (ML)

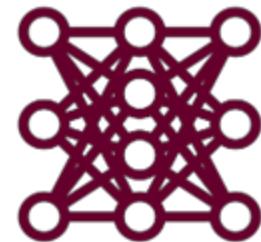


# Task specification in ML: programs → examples



Here is a program to  
recognize an image as  
a cow or a turtle

```
def cow_or_turtle(image):
```



Here are some  
examples. Try to  
imitate them.



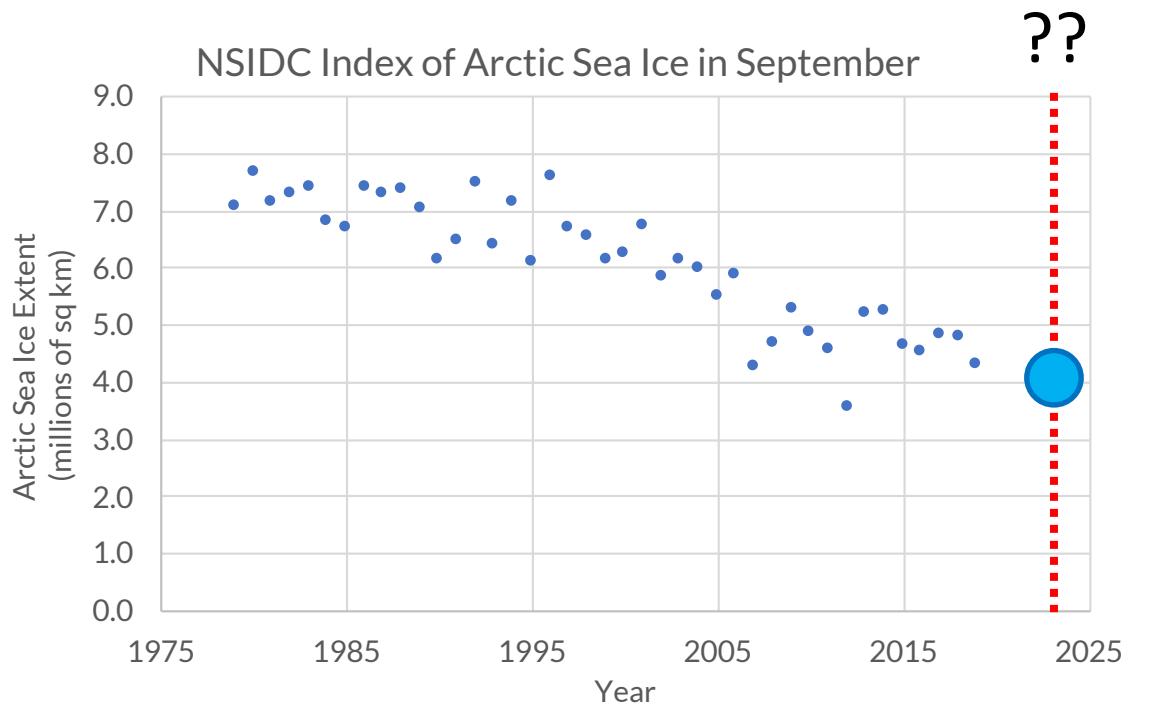
“cows”

“turtles”

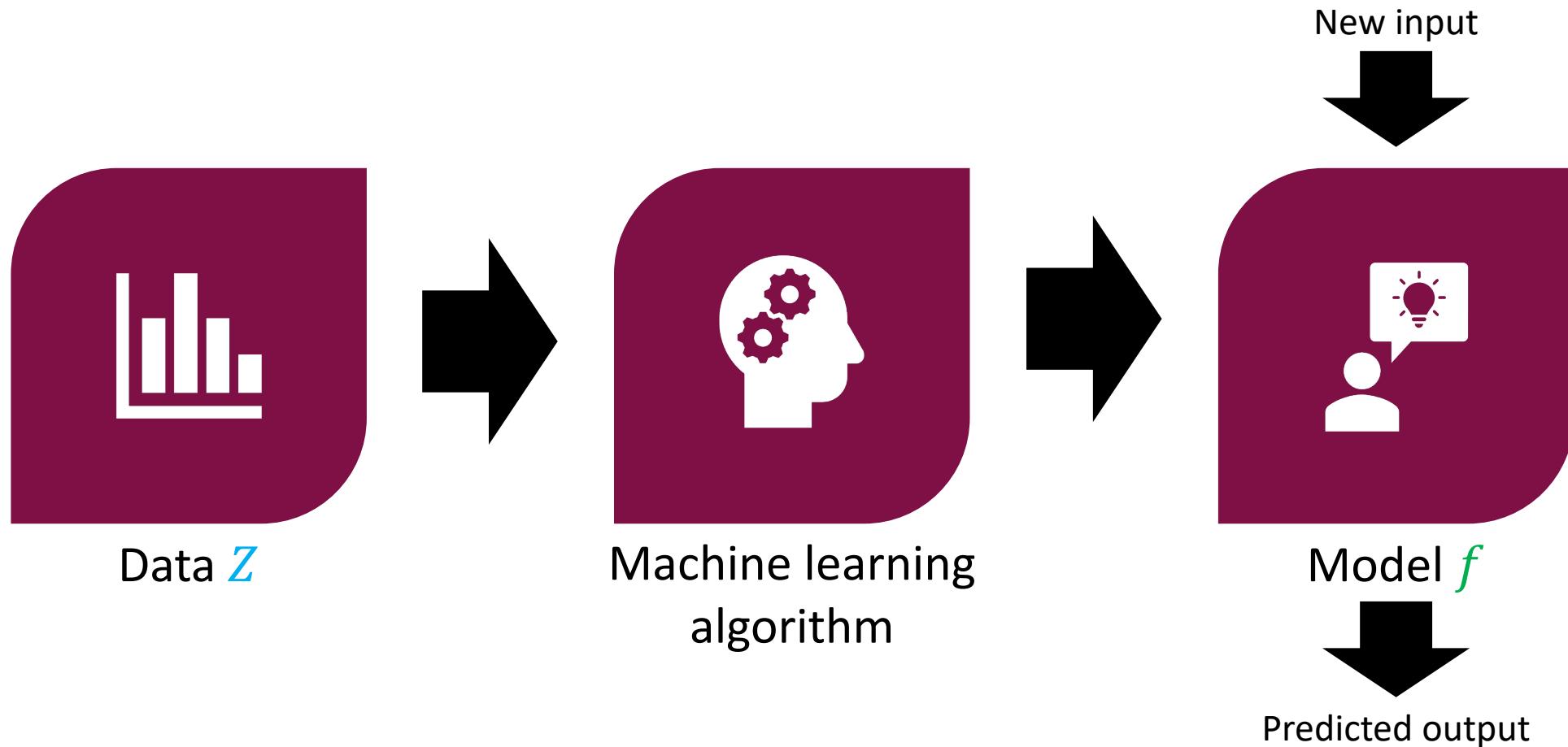
# Example: Prediction



Photo by NASA Goddard



# Machine Learning for Prediction



# Types of Learning

- **Supervised learning**
  - **Input:** Examples of inputs and outputs
  - **Output:** Model that predicts unknown output given a new input
- **Unsupervised learning**
  - **Input:** Examples of some data (no “outputs”)
  - **Output:** Representation of structure in the data
- **Reinforcement learning**
  - **Input:** Sequence of interactions with an environment
  - **Output:** Policy that performs a desired task

# Supervised Learning

- Given  $(x_1, y_1), \dots, (x_n, y_n)$ , learn a function that predicts  $y$  given  $x$
- **Regression:** Labels  $y$  are real-valued



Photo by NASA Goddard

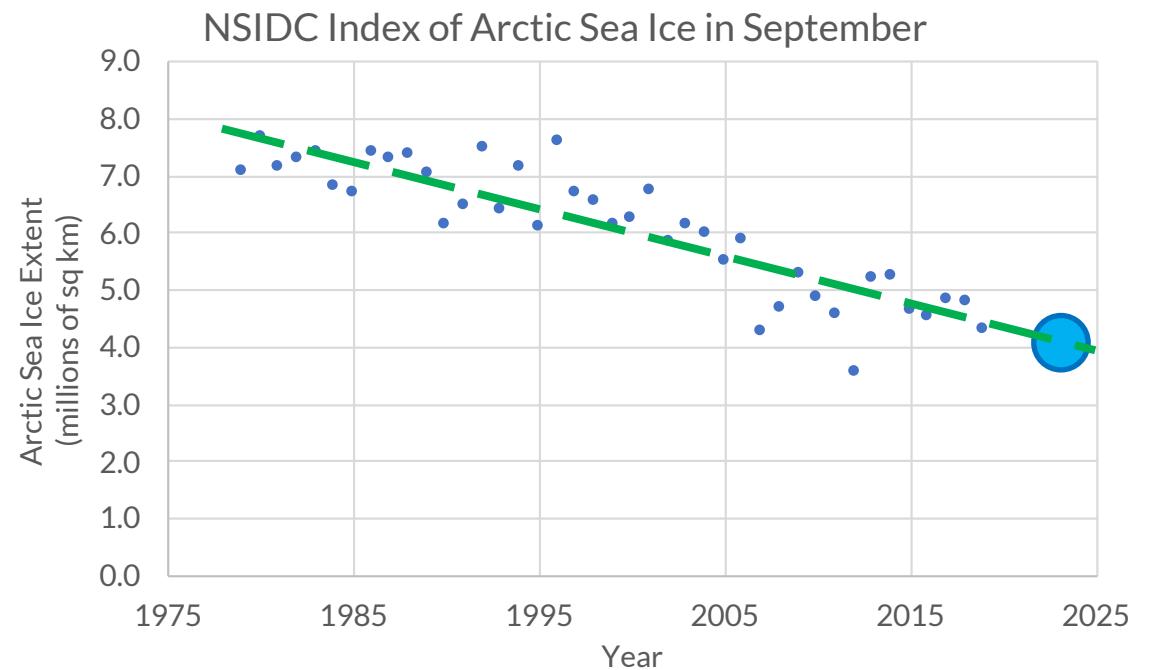
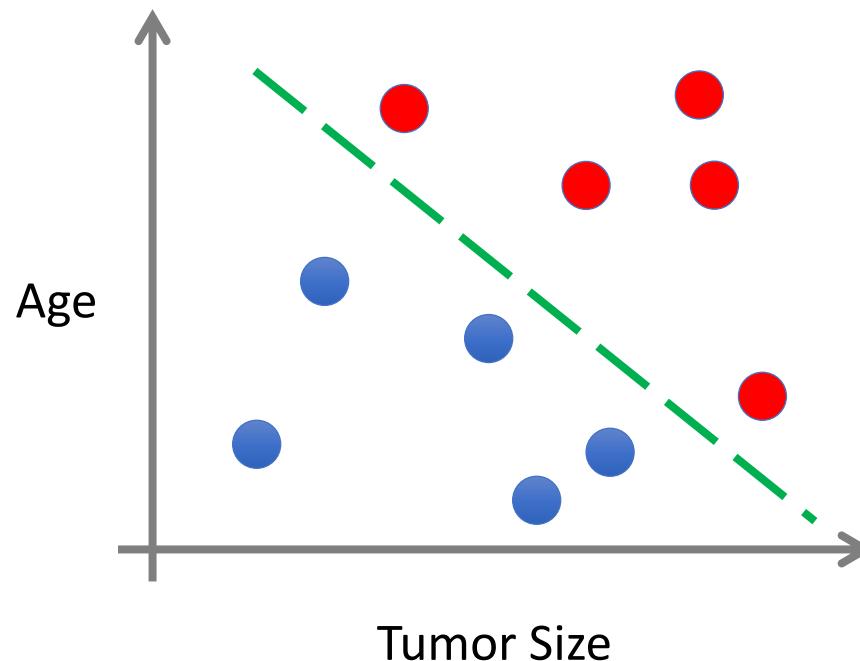


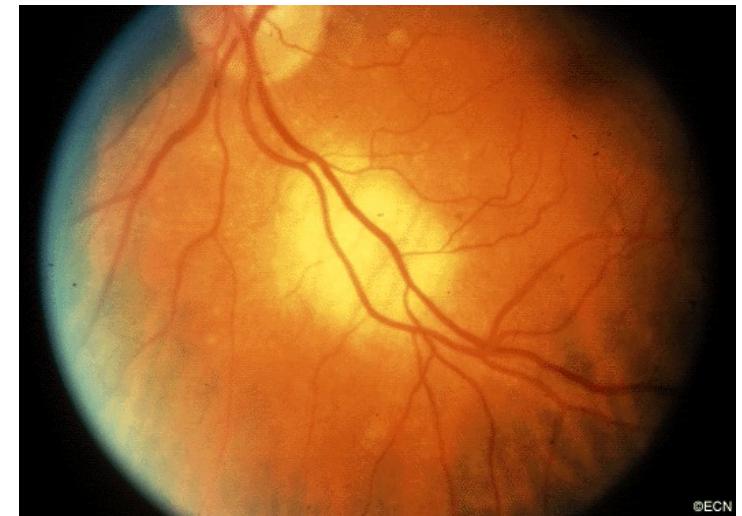
Image: <https://www.flickr.com/photos/gsfc/5937599688/>  
Data from <https://nsidc.org/arcticseainews/sea-ice-tools/>

# Supervised Learning

- Given  $(x_1, y_1), \dots, (x_n, y_n)$ , learn a function that predicts  $y$  given  $x$
- Inputs  $x$  can be multi-dimensional

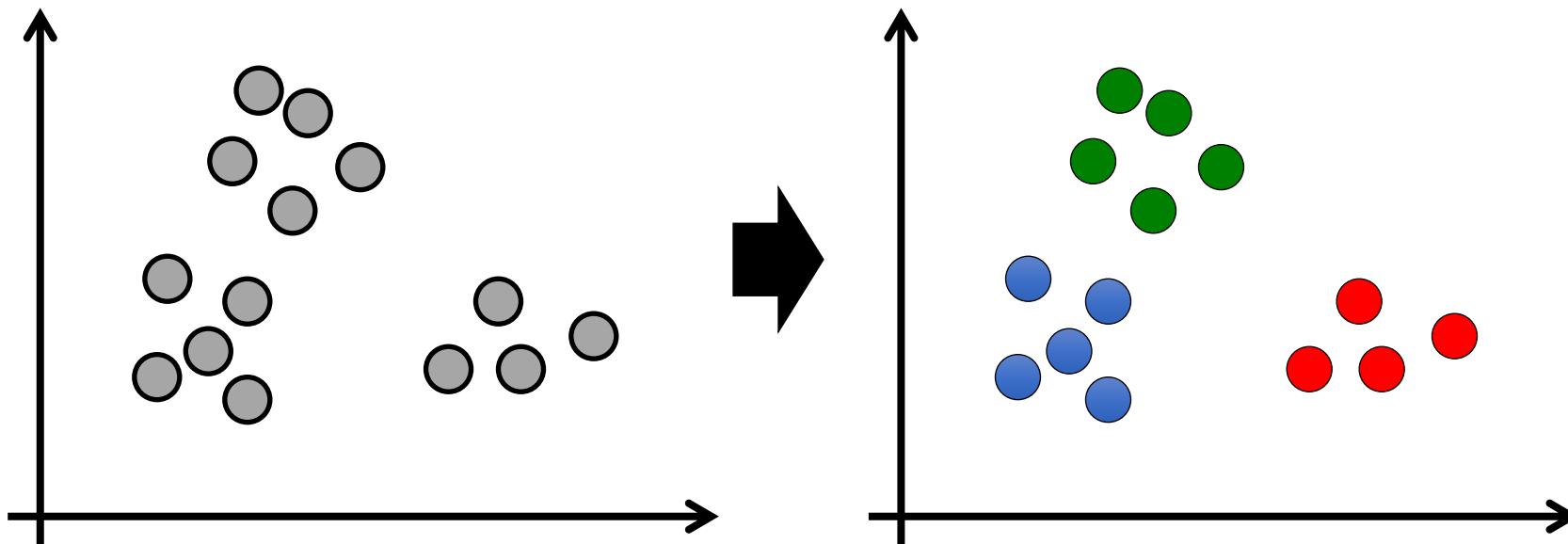


- Patient age
- Clump thickness
- Tumor Color
- Cell type
- ...

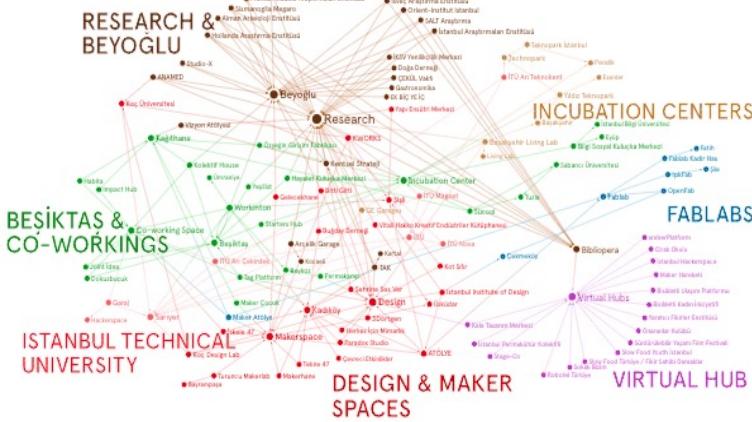


# Unsupervised Learning

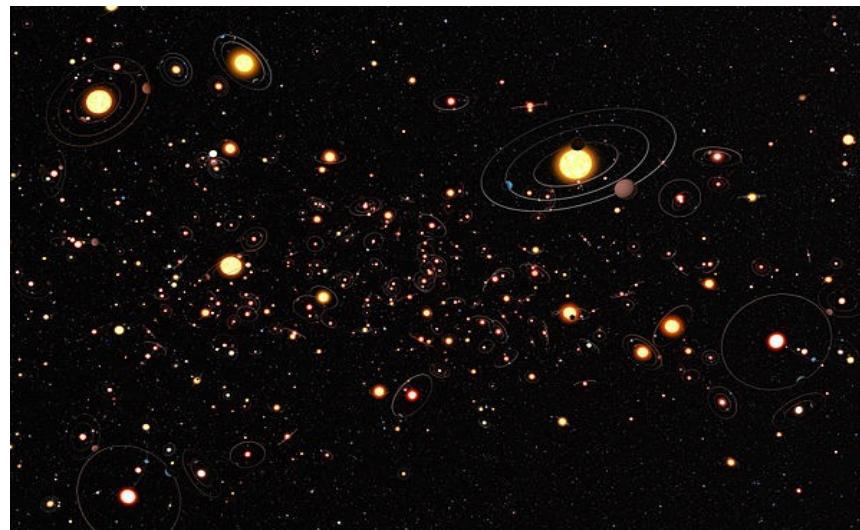
- Given  $x_1, \dots, x_n$  (no labels), output hidden structure in  $x$ 's
  - E.g., clustering



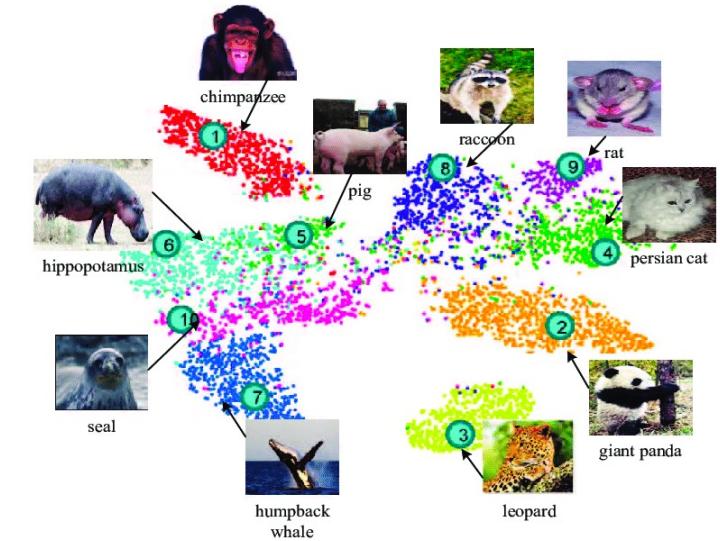
# Unsupervised Learning



Find Subgroups in  
Social Networks



Identify Types of Exoplanets



Visualize Data

Image Credits:

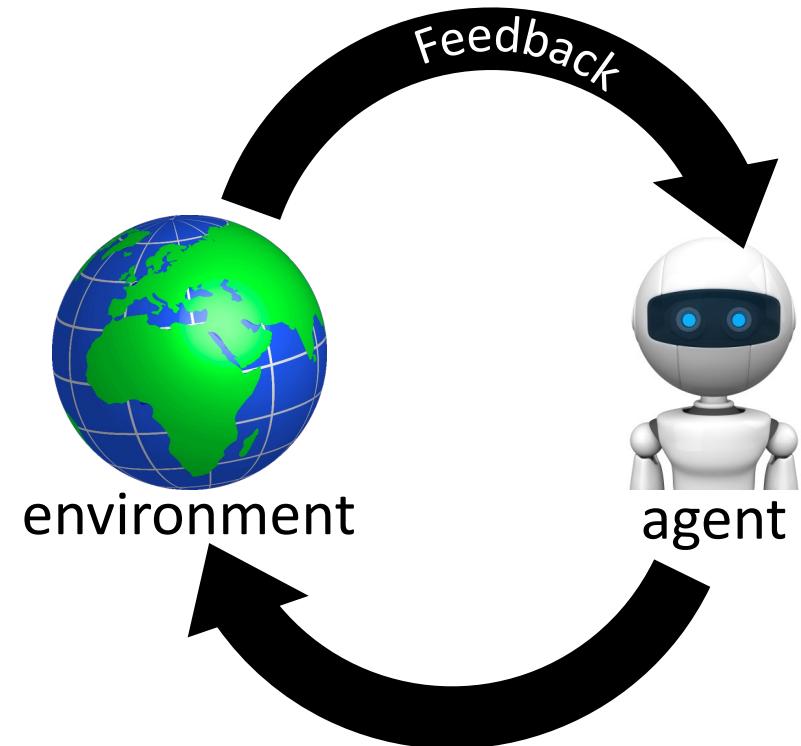
<https://medium.com/graph-commons/finding-organic-clusters-in-your-complex-data-networks-5c27e1d4645d>

<https://arxiv.org/pdf/1703.08893.pdf>

<https://en.wikipedia.org/wiki/Exoplanet>

# Reinforcement Learning

- Learn how to perform a task from interactions with the **environment**
- **Examples:**
  - Playing chess (interact with the game)
  - Robot grasping an object (interact with the object/real world)
  - Optimize inventory allocations (interact with the inventory system)



# Everyday Applications

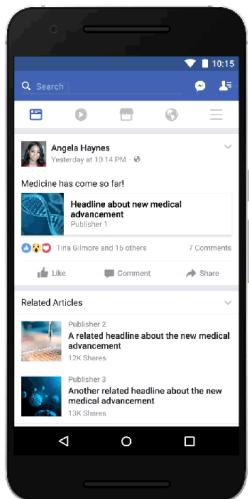
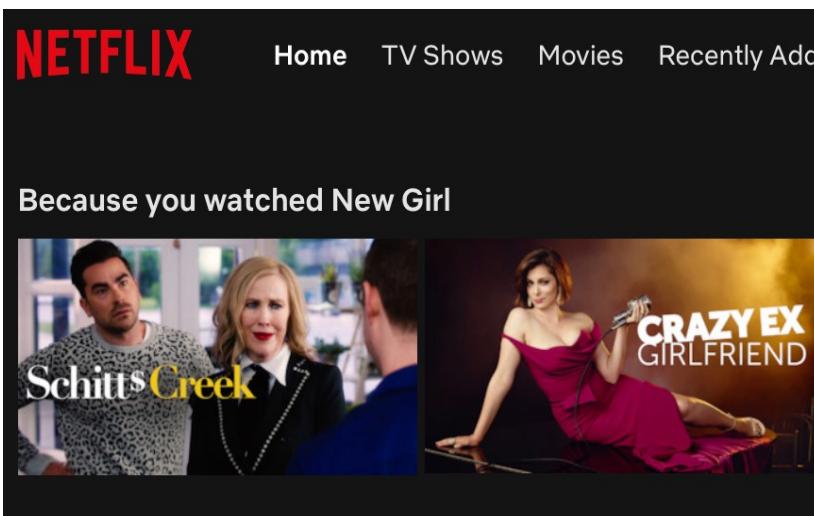
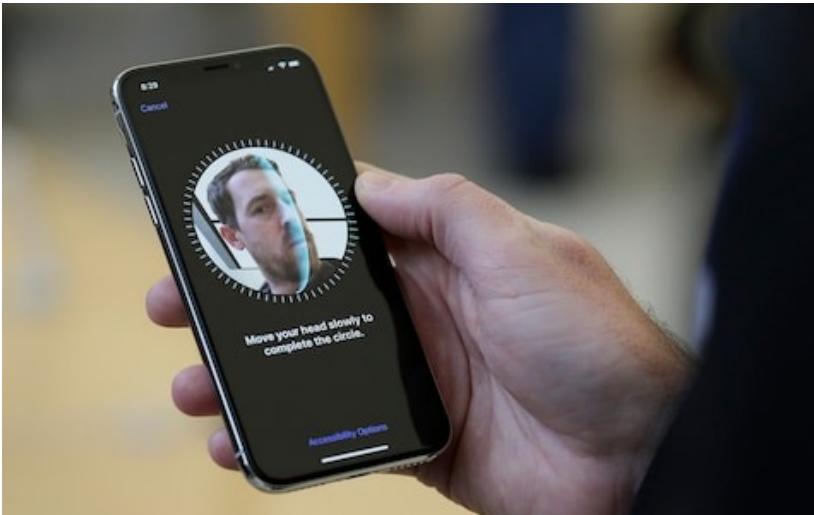
COVID-19 PAYMENT ➔ Spam ✎



Miller, Jane  
to me ▾

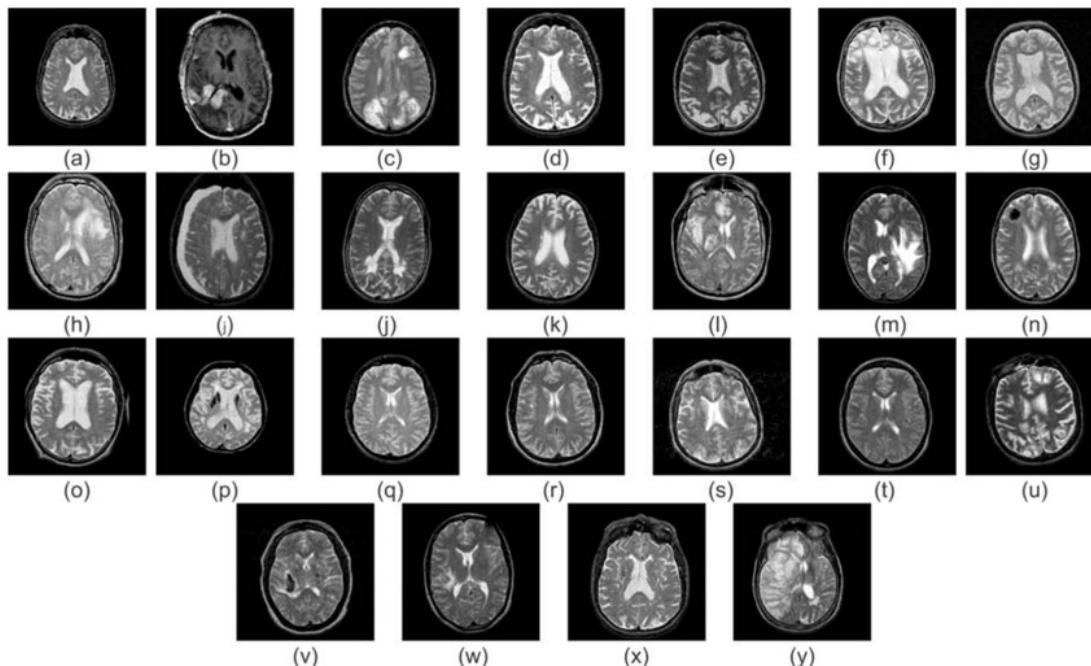


Good morning,  
You are advised to download the attached invoice for your review. Please get back to us as soon as possible.  
Thanks,  
Jane



# Radiology and Medicine

**Input:** Brain scans



**Output:** Neurological disease labels

Machine learning studies on major brain diseases: 5-year trends  
of 2014–2018

**Applications of machine learning in drug discovery  
and development**

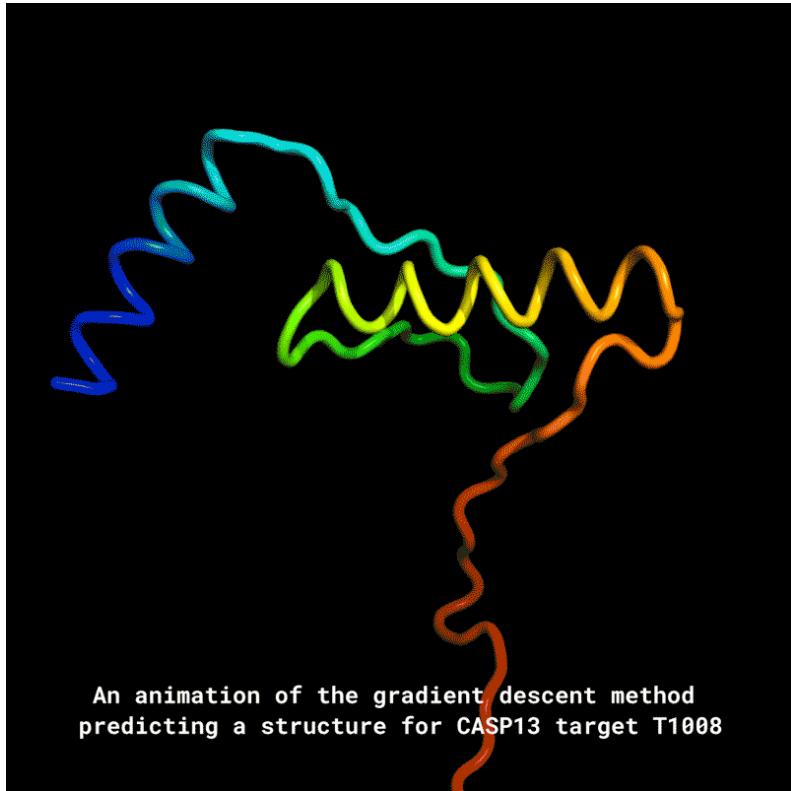
<https://www.nature.com/articles/s41573-019-0024-5>

**Deep learning-enabled medical computer vision**

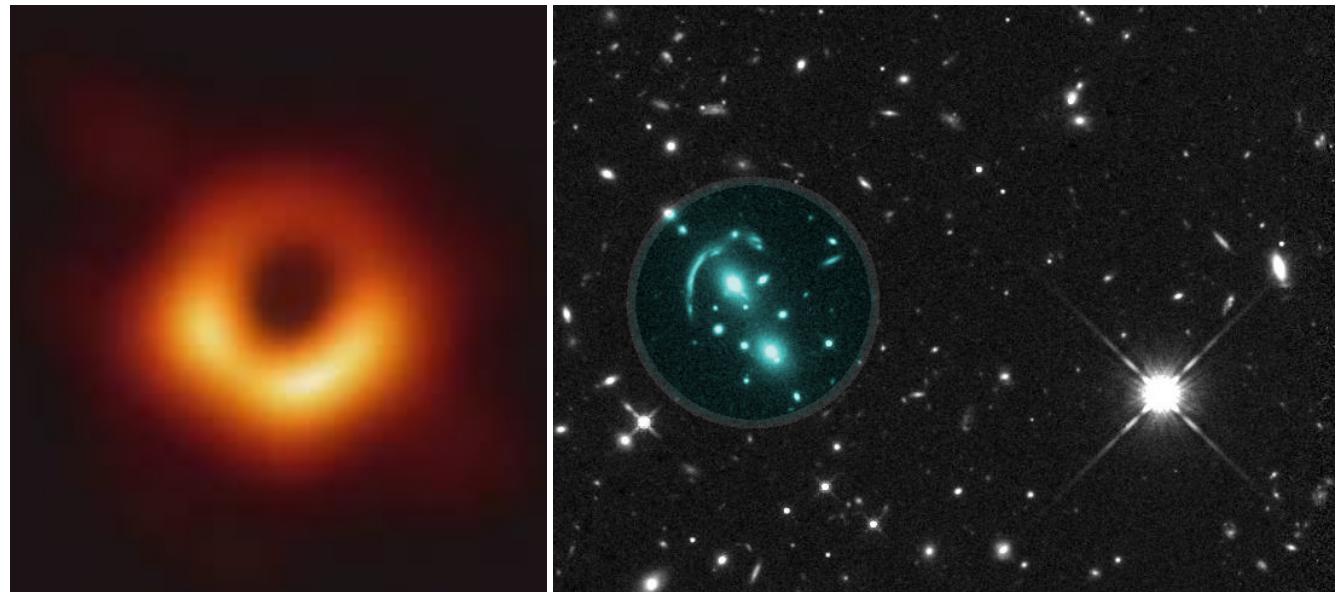
Andre Esteva , Katherine Chou, Serena Yeung, Nikhil Naik, Ali Madani, Ali Mottaghi, Yun Liu, Eric Topol, Jeff Dean & Richard Socher

<https://www.nature.com/articles/s41746-020-00376-2>

# Scientific Discovery



<https://deepmind.com/blog/article/AlphaFold-Using-AI-for-scientific-discovery>



<https://www.jpl.nasa.gov/edu/news/2019/4/19/how-scientists-captured-the-first-image-of-a-black-hole/>

# Creating Images & Text



<https://thispersondoesnotexist.com/>

## SYSTEM PROMPT (HUMAN-WRITTEN)

*Recycling is good for the world.*

***NO! YOU COULD NOT BE MORE WRONG!!***

## MODEL COMPLETION (MACHINE-WRITTEN, 25 TRIES)

Recycling is NOT good for the world. It is bad for the environment, it is bad for our health, and it is bad for our economy. I'm not kidding. Recycling is not good for the environment. It is destructive to the earth and it is a major contributor to global warming. Recycling is not good for our health. It contributes to obesity and diseases like heart disease and cancer. Recycling is bad for our economy. It increases the cost of a product, and in turn, the price of everything that is made with that product. Recycling is not good for our nation. We pay a

- **GPT-2 Small:** 124M or 117M parameters (12 layers, 768 hidden dimension)
- **GPT-2 Medium:** 355M or 345M parameters (24 layers, 1024 hidden dimension)
- **GPT-2 Large:** 774M or 762M parameters (36 layers, 1280 hidden dimension)
- **GPT-2 XL (Full):** 1.5B or 1558M parameters (48 layers, 1600 hidden dimension) ☕

<https://transformer.huggingface.co/doc/gpt2-large>

# GPT2 - Feb 2019; GPT-5.2 Early 2026

ChatGPT 5.2 Thinking ▾

↑ Share

Recycling is good for the world. NO! YOU COULD NOT BE MORE WRONG!

I get the impulse behind that "NO!" — recycling gets marketed like a magic fix, and it absolutely can be **oversold**. But "recycling is good for the world" isn't *wrong*; it's conditional.

What's true at the same time

- Recycling can be genuinely beneficial when it replaces virgin material production (especially aluminum and steel, often paper), because making new material usually costs a lot of energy, mining, water, and pollution.
- Recycling can also be ineffective or even counterproductive when collection/sorting/transport + low yield + contamination means you burn lots of energy to produce low-grade material that still doesn't displace much virgin production.

Why people say "recycling is a scam" (and sometimes they're right)

- Contamination (food/oil/dirty plastics) can ruin batches and force landfilling/incineration.
- Many plastics are hard to recycle economically; a lot get "downcycled" into lower-value products and then landfilled later.
- If recycling becomes a moral "permission slip" to keep over-consuming single-use stuff, the net effect can be bad.

The hierarchy that's hard to argue with

If your goal is actually helping the planet, the order is usually:

Reduce → Reuse/Repair → Refill/Return → Recycle (last)

Recycling is not the hero: it's the cleanup crew.

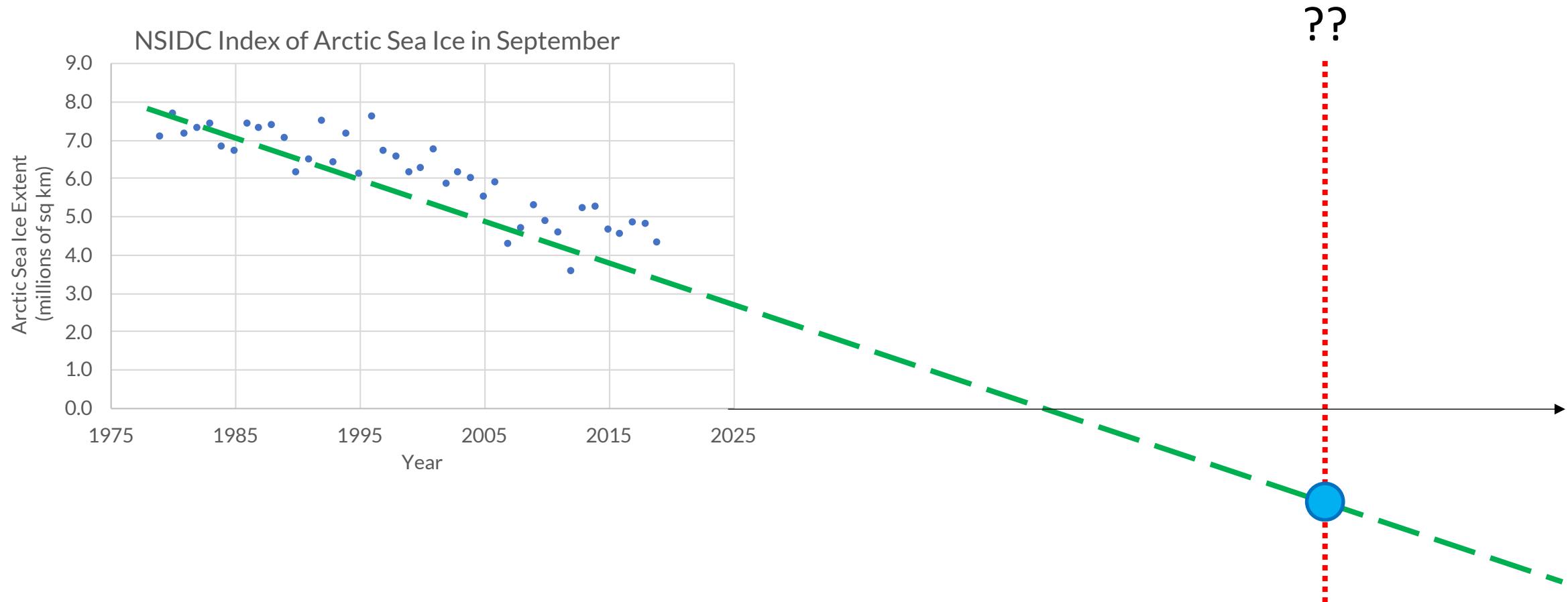
As of early 2026, OpenAI has not officially released the exact parameter count for GPT-5.2. [🔗](#)

However, based on industry analyses and model comparisons from late 2025, here is the current understanding:

- **Model Type:** GPT-5.2 is part of the GPT-5 series, which uses a **Mixture of Experts (MoE)** architecture, meaning not all parameters are active during every inference pass.
- **Estimated Size:** Predictions for models in the GPT-5 family range from **125 billion to over 600 billion parameters**, with some estimations suggesting total parameters for advanced versions could be in the trillions.
- **Performance vs. Size:** Instead of just increasing parameters, GPT-5.2 focuses on **inference-time compute** and advanced reasoning, which allows smaller parameter models (like "mini" or "instant" versions) to perform exceptionally well. [🔗](#)

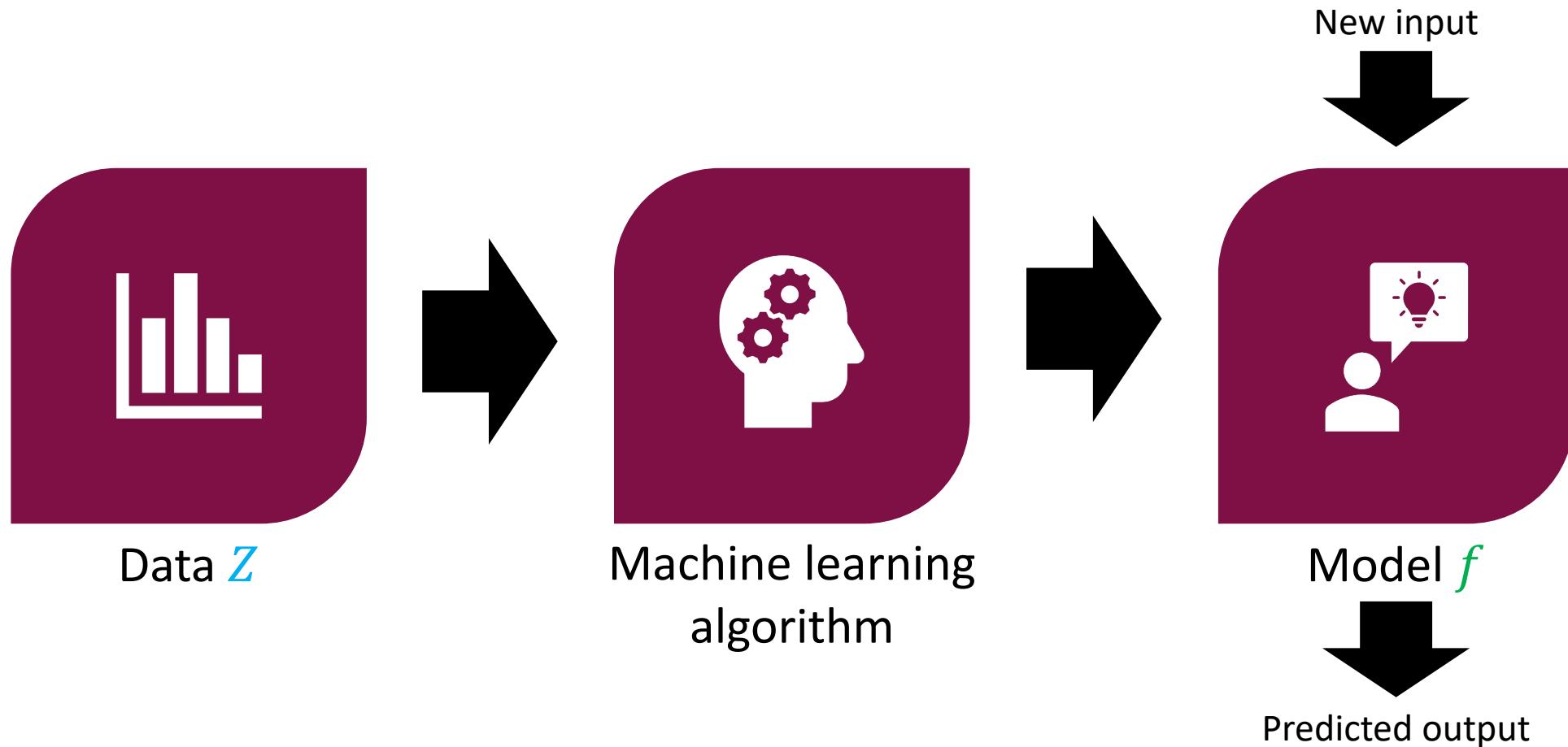
GPT-5.2 is designed for superior agentic tasks, long-context understanding, and multi-step reasoning. [🔗](#)

# Danger of Out-of-Domain Machine Learning



Any time you are evaluating on data “far” from your training data, beware!

# Machine Learning for Prediction



**Question:** What model family (a.k.a. hypothesis class) to consider?

# Linear Functions

- Consider the space of linear functions  $f_{\beta}(x)$  defined by

$$f_{\beta}(x) = \beta^T x$$

# Linear Functions

- Consider the space of linear functions  $f_{\beta}(x)$  defined by

$$f_{\beta}(x) = \beta^T x = [\beta_1 \quad \cdots \quad \beta_d] \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} = \beta_1 x_1 + \cdots + \beta_d x_d$$

- $x \in \mathbb{R}^d$  is called an **input** (a.k.a. **features** or **covariates**)
- $\beta \in \mathbb{R}^d$  is called the **parameters** (a.k.a. **parameter vector**)
- $y = f_{\beta}(x)$  is called the **label** (a.k.a. **output** or **response**)

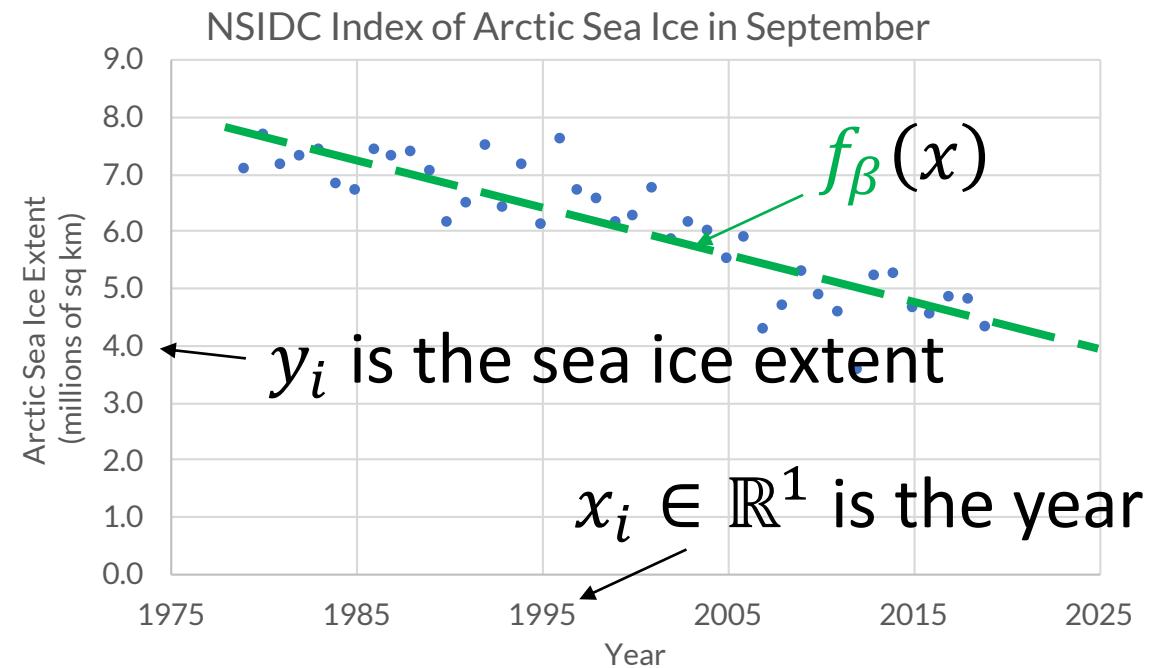
# Linear Regression Problem

- **Input:** Dataset  $Z = \{(x_1, y_1), \dots, (x_n, y_n)\}$ , where  $x_i \in \mathbb{R}^d$  and  $y_i \in \mathbb{R}$
- **Output:** A linear function  $f_{\beta}(x) = \beta^T x$  such that  $y_i \approx \beta^T x_i$
- **Typical notation**
  - Use  $i$  to index examples  $(x_i, y_i)$  in data  $Z$
  - Use  $j$  to index components  $x_j$  of  $x \in \mathbb{R}^d$
  - $x_{ij}$  is component  $j$  of input example  $i$
- **Goal:** Estimate  $\beta \in \mathbb{R}^d$

# Linear Regression Problem

What does this mean?

- **Input:** Data  $Z = \{(x_1, y_1), \dots, (x_n, y_n)\}$ , where  $x_i \in \mathbb{R}^d$  and  $y_i \in \mathbb{R}$
- **Output:** A linear function  $f_{\beta}(x) = \beta^T x$  such that  $y_i \approx \beta^T x_i$

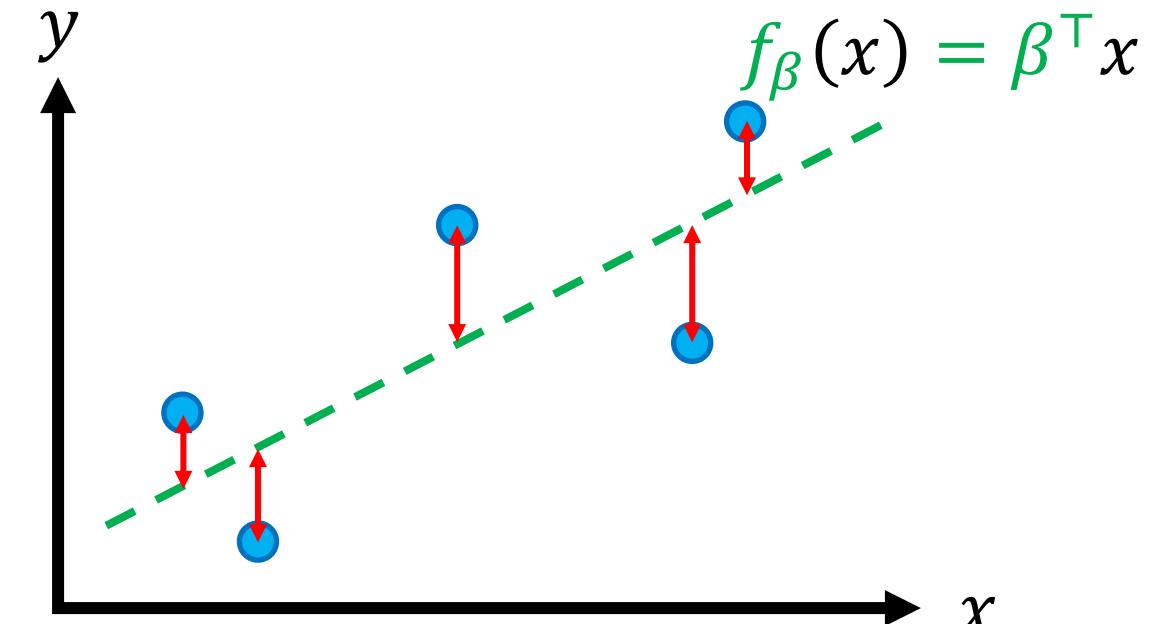


# Choice of Loss Function

- $y_i \approx \beta^\top x_i$  if  $(y_i - \beta^\top x_i)^2$  small
- Mean squared error (MSE):

$$L(\beta; Z) = \frac{1}{n} \sum_{i=1}^n (y_i - \beta^\top x_i)^2$$

- Computationally convenient and works well in practice



$$L(\beta; Z) = \frac{\text{↑}^2 + \text{↑}^2 + \text{↑}^2 + \text{↑}^2 + \text{↑}^2}{n}$$

# Linear Regression Problem

- **Input:** Data  $Z = \{(x_1, y_1), \dots, (x_n, y_n)\}$ , where  $x_i \in \mathbb{R}^d$  and  $y_i \in \mathbb{R}$
- **Output:** A linear function  $f_{\beta}(x) = \beta^T x$  that minimizes the MSE:

$$L(\beta; Z) = \frac{1}{n} \sum_{i=1}^n (y_i - \beta^T x_i)^2$$

# Linear Regression Algorithm

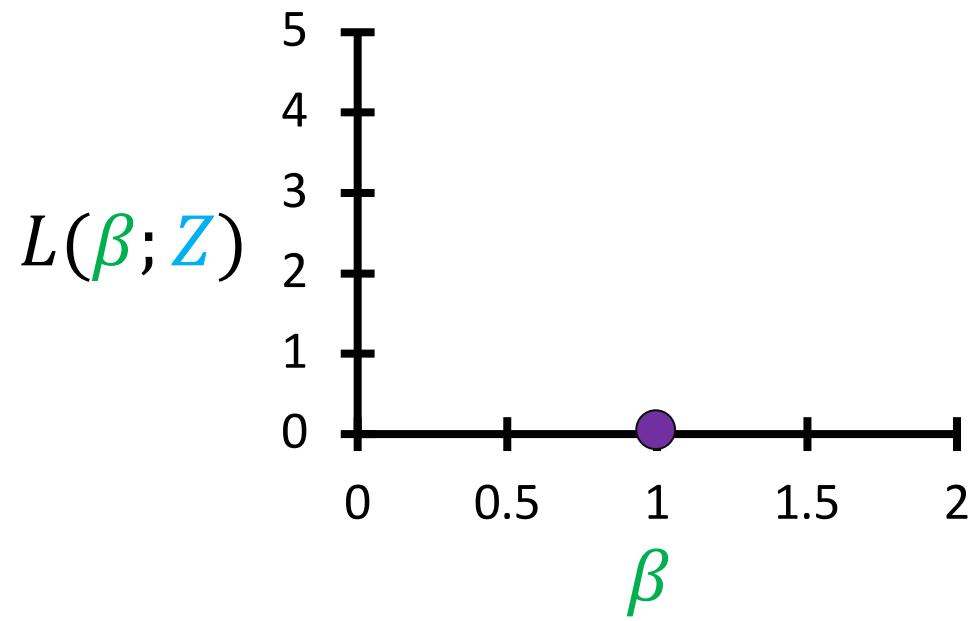
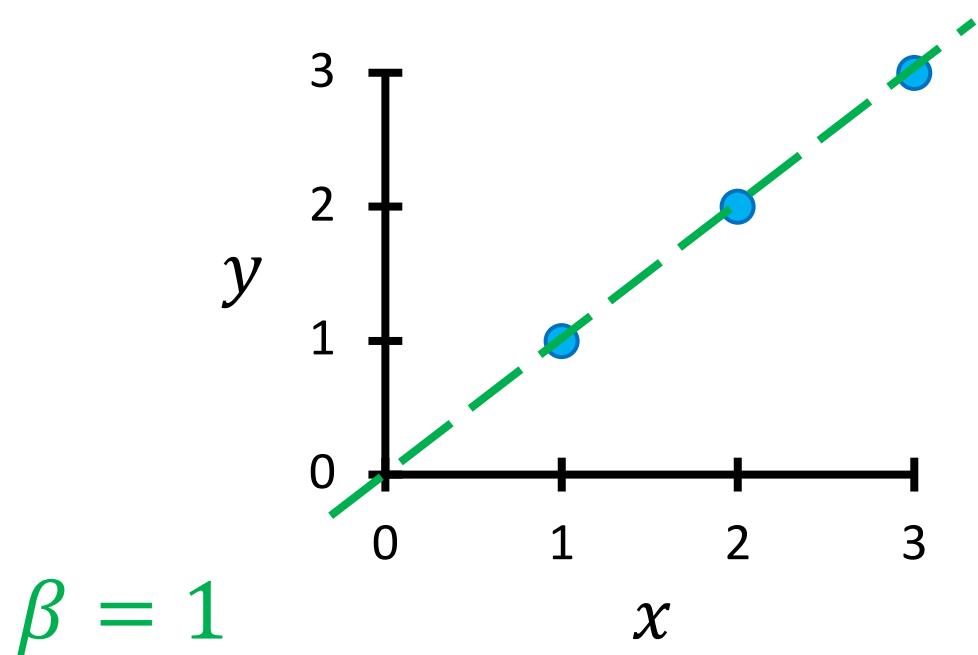
- **Input:** Dataset  $Z = \{(x_1, y_1), \dots, (x_n, y_n)\}$
- Compute

$$\begin{aligned}\hat{\beta}(Z) &= \arg \min_{\beta \in \mathbb{R}^d} L(\beta; Z) \\ &= \arg \min_{\beta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n (y_i - \beta^\top x_i)^2\end{aligned}$$

- **Output:**  $f_{\hat{\beta}(Z)}(x) = \hat{\beta}(Z)^\top x$
- Discuss algorithm for computing the minimal  $\beta$  later

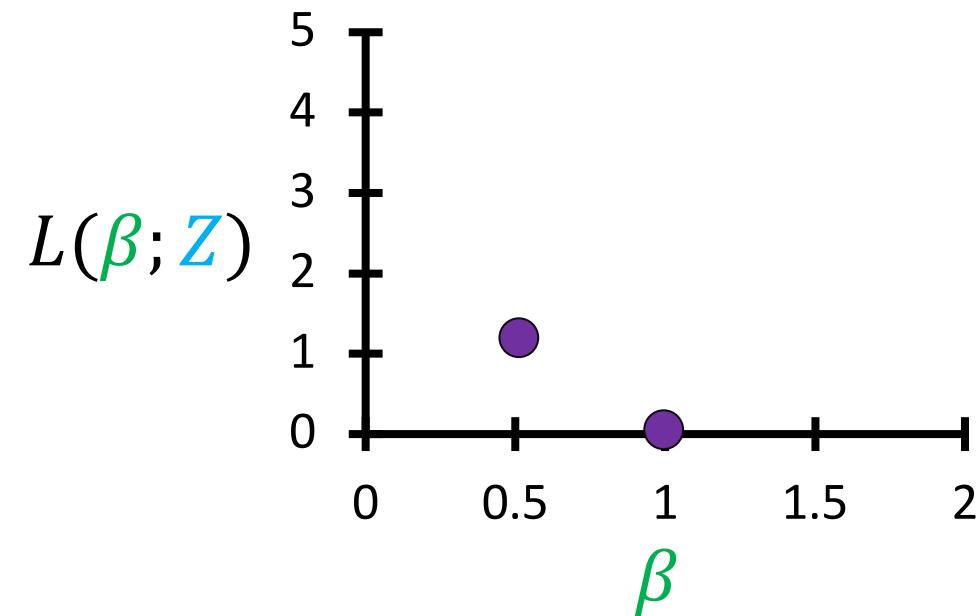
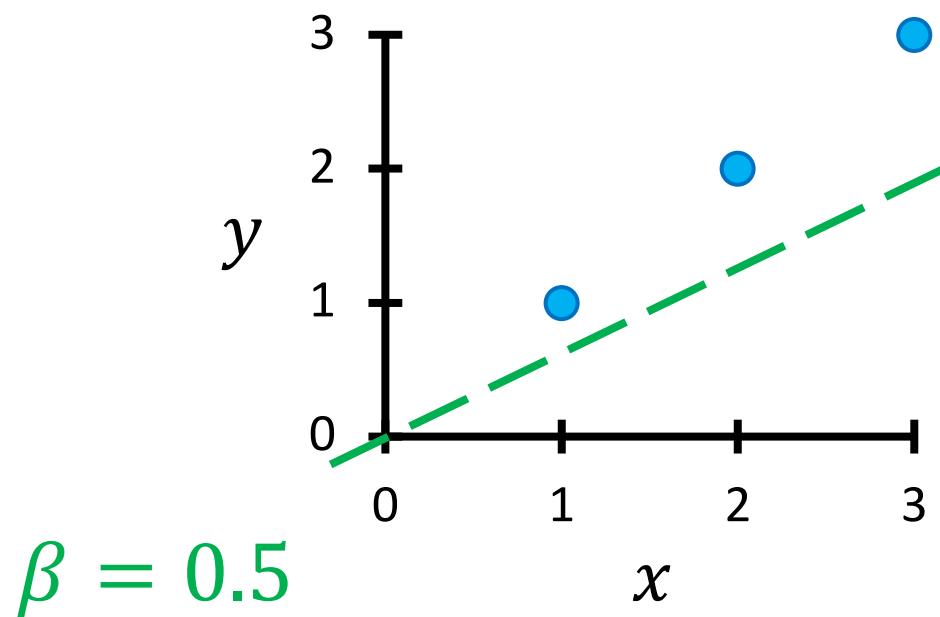
# Intuition on Minimizing MSE Loss

- Consider  $x \in \mathbb{R}$  and  $\beta \in \mathbb{R}$



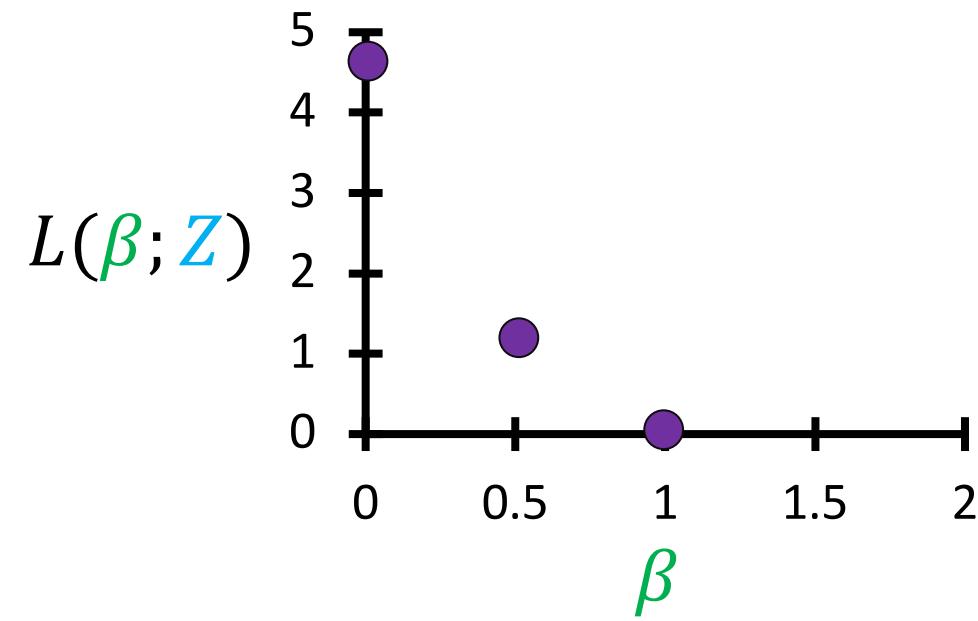
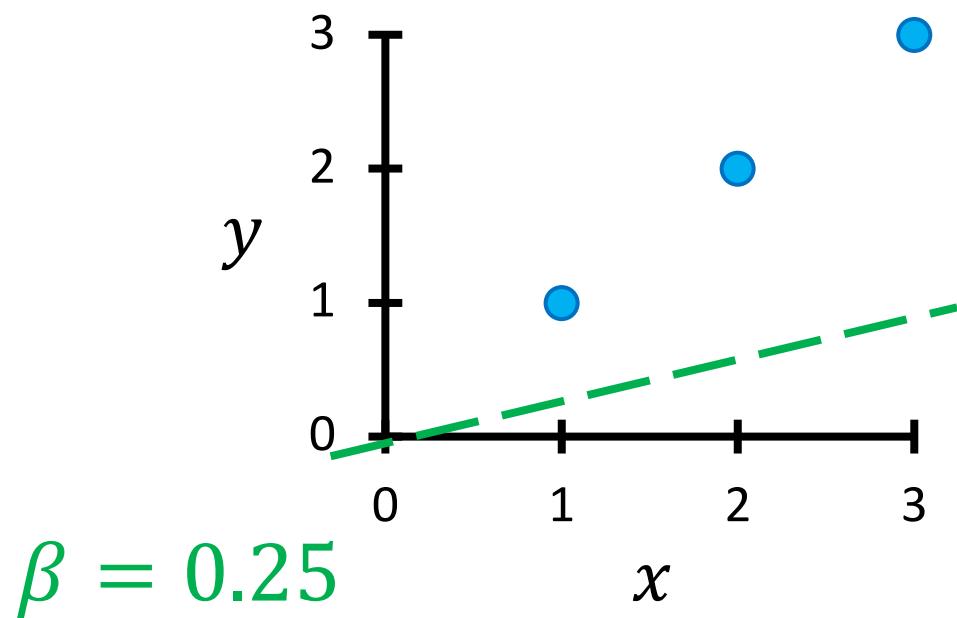
# Intuition on Minimizing MSE Loss

- Consider  $x \in \mathbb{R}$  and  $\beta \in \mathbb{R}$



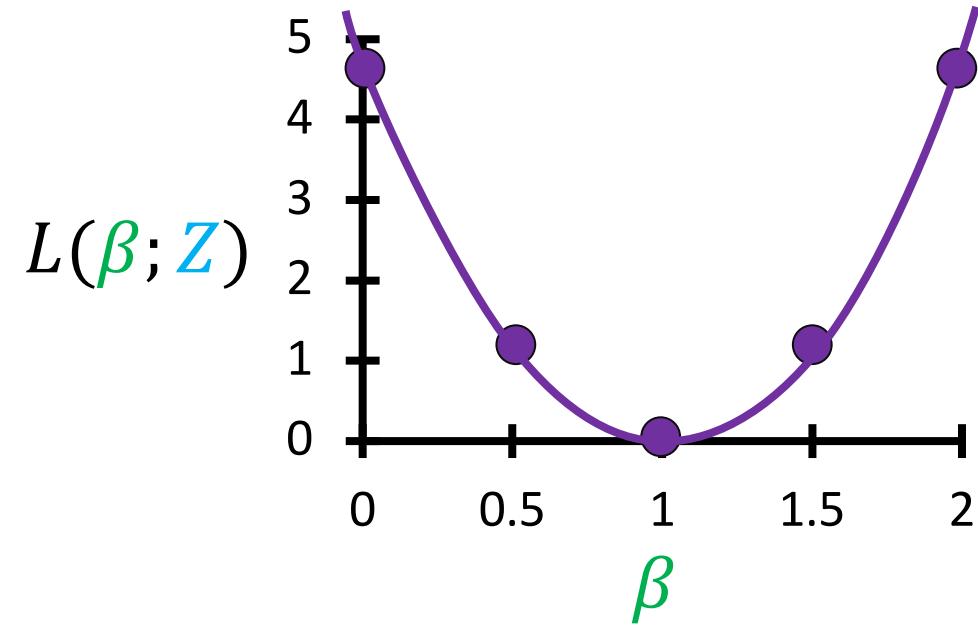
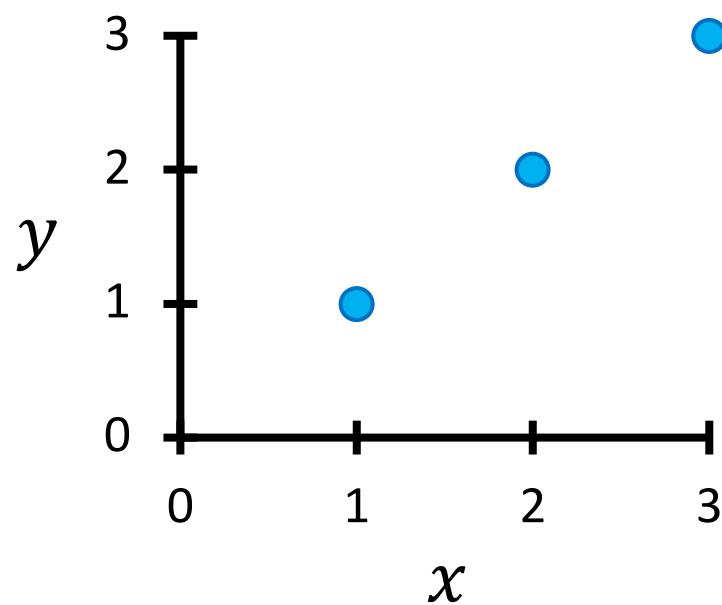
# Intuition on Minimizing MSE Loss

- Consider  $x \in \mathbb{R}$  and  $\beta \in \mathbb{R}$



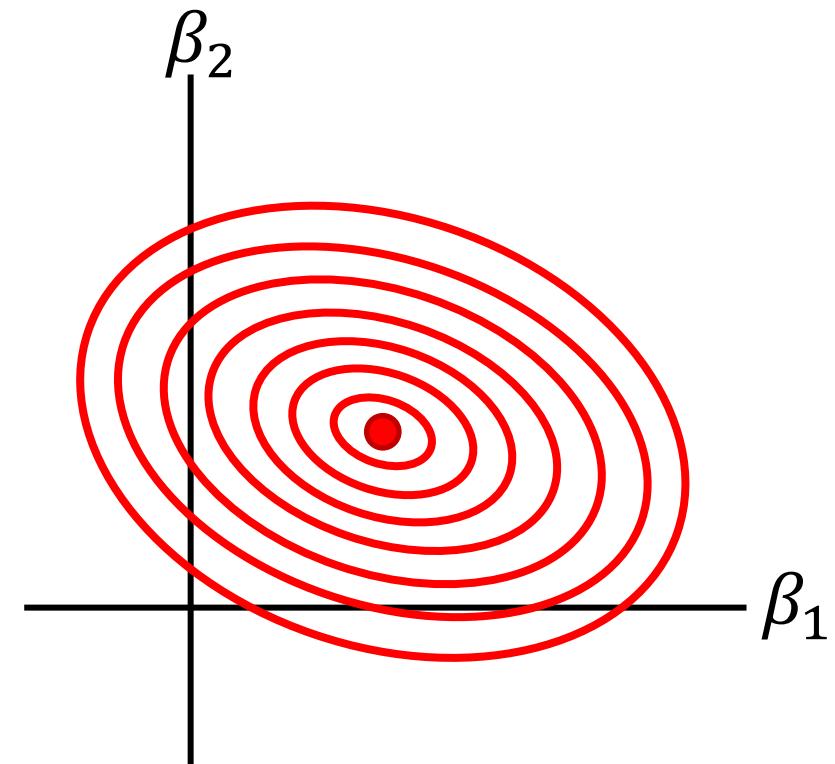
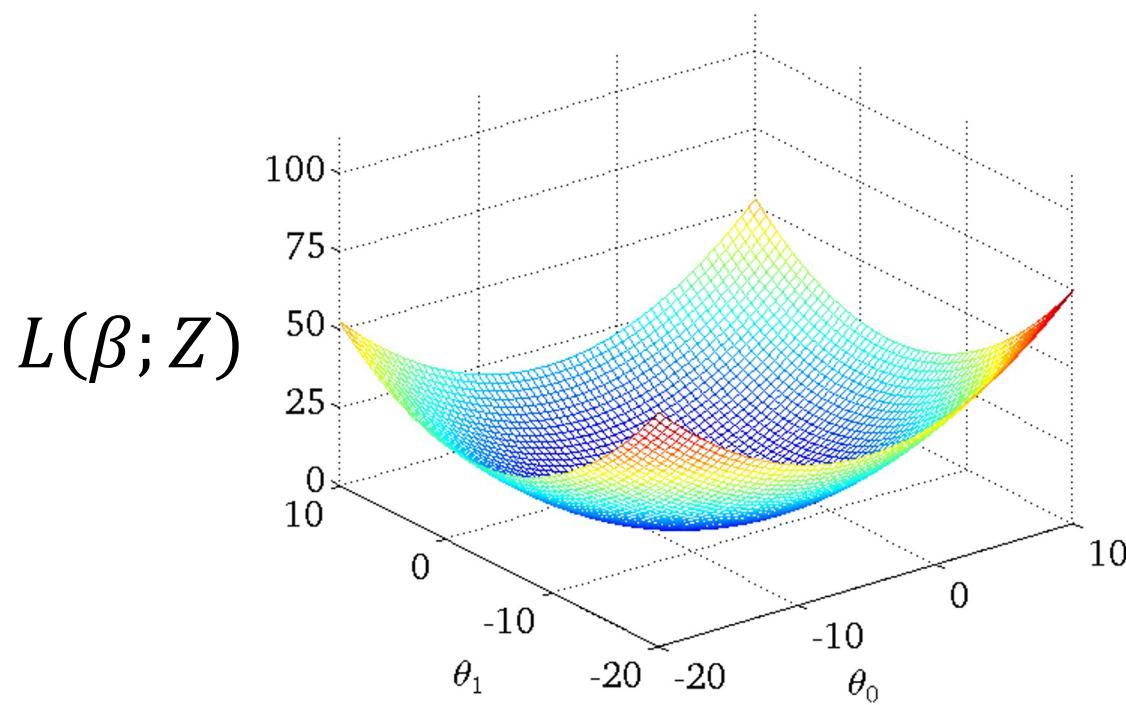
# Intuition on Minimizing MSE Loss

- Consider  $x \in \mathbb{R}$  and  $\beta \in \mathbb{R}$



# Intuition on Minimizing MSE Loss

- **Convex** (“bowl shaped”) in general



# Alternative Loss Functions

- **Mean absolute error:**

$$\frac{1}{n} \sum_{i=1}^n |\hat{y}_i - y_i|$$

- **Mean relative error:**

$$\frac{1}{n} \sum_{i=1}^n \frac{|\hat{y}_i - y_i|}{|y_i|}$$

- **$R^2$  score:**

$$1 - \frac{\text{MSE}}{\text{Variance}}$$

- “Coefficient of determination”
- Higher is better,  $R^2 = 1$  is perfect

# Linear Regression

## General strategy

- Model family  $F = \{f_{\beta}\}_{\beta}$
- Loss function  $L(\beta; Z)$

## Linear regression strategy

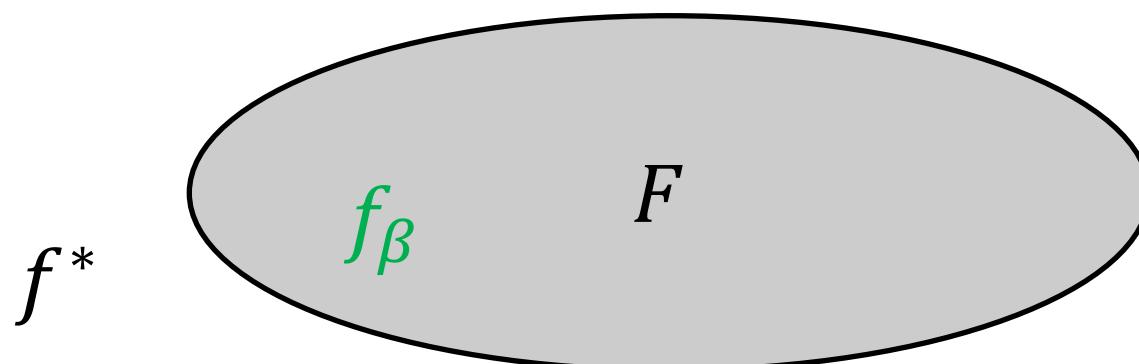
- Linear functions  $F = \{f_{\beta}(x) = \beta^T x\}$
- MSE  $L(\beta; Z) = \frac{1}{n} \sum_{i=1}^n (y_i - \beta^T x_i)^2$

## Linear regression algorithm

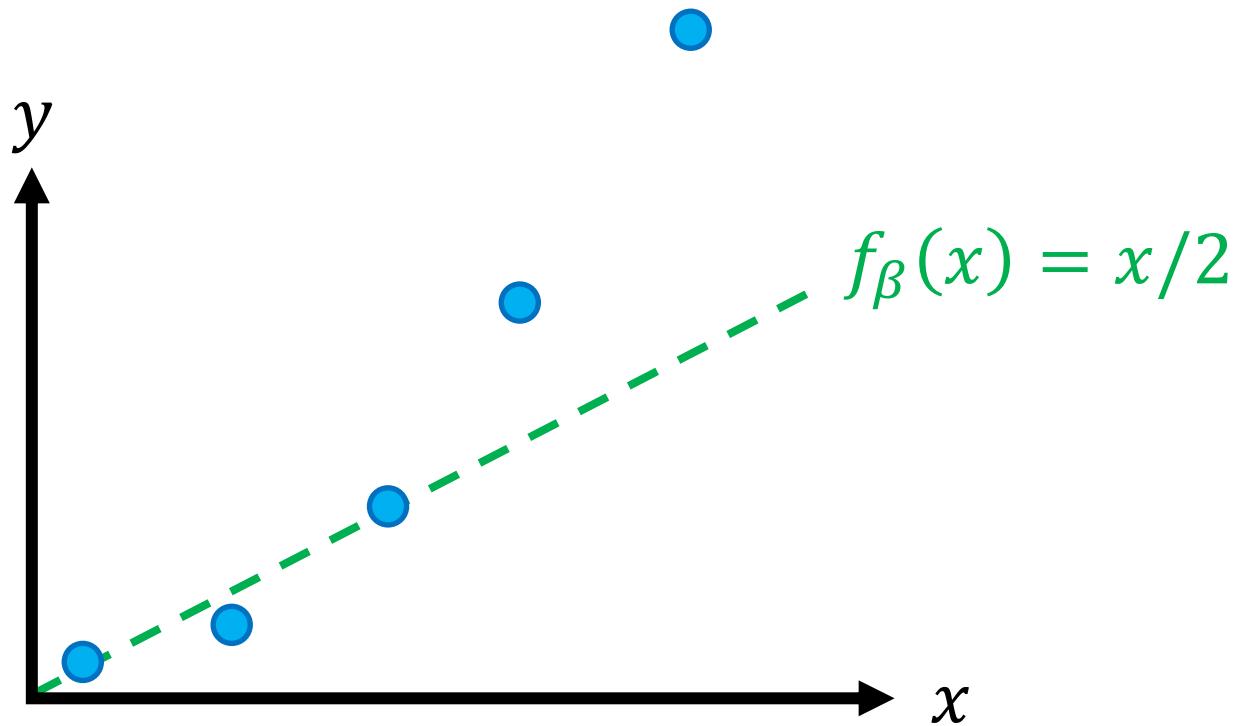
$$\hat{\beta}(Z) = \arg \min_{\beta} L(\beta; Z)$$

# Aside: “True Function”

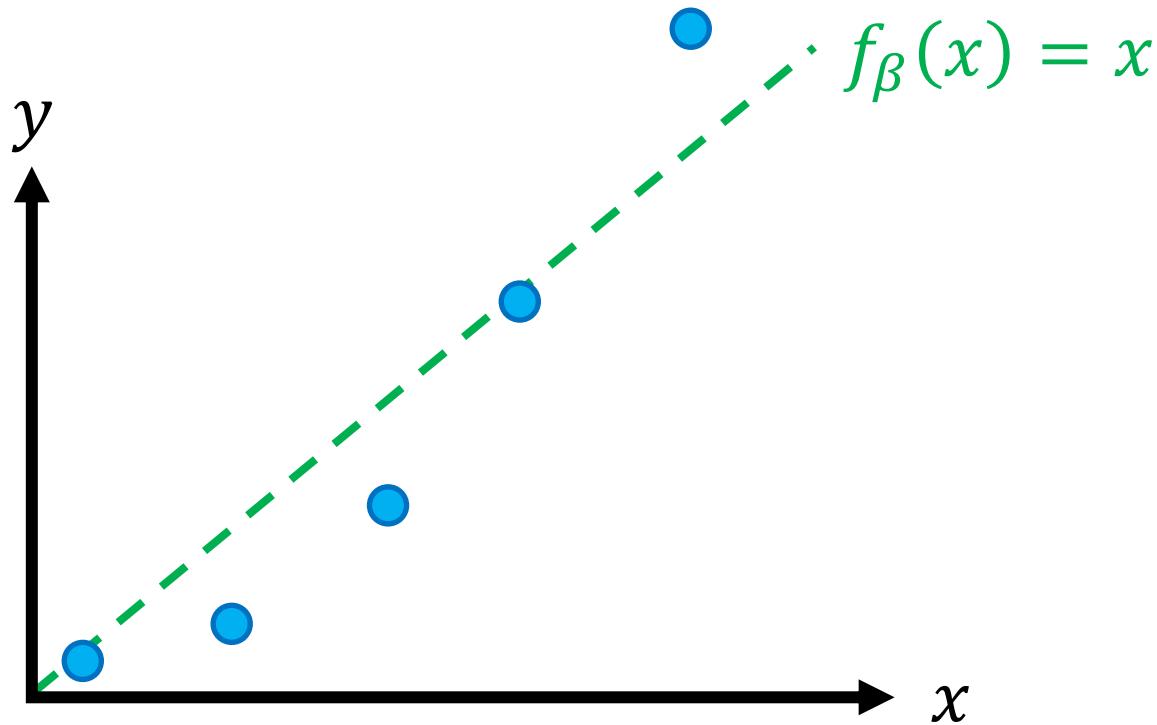
- **Input:** Dataset  $Z$ 
  - Presume there is an unknown function  $f^*$  that **generates**  $Z$
- **Goal:** Find an **approximation**  $f_\beta \approx f^*$  in our model family  $f_\beta \in F$ 
  - Typically,  $f^*$  not in our model family  $F$



# Example: Quadratic Function



# Example: Quadratic Function



Can we get a better fit?

# Feature Maps

## General strategy

- Model family  $F = \{f_{\beta}\}_{\beta}$
- Loss function  $L(\beta; Z)$

## Linear regression with feature map

- Linear functions over a given **feature map**  $\phi: X \rightarrow \mathbb{R}^d$

$$F = \{f_{\beta}(x) = \beta^T \phi(x)\}$$

- MSE  $L(\beta; Z) = \frac{1}{n} \sum_{i=1}^n (\textcolor{blue}{y}_i - \beta^T \phi(\textcolor{blue}{x}_i))^2$

# Quadratic Feature Map

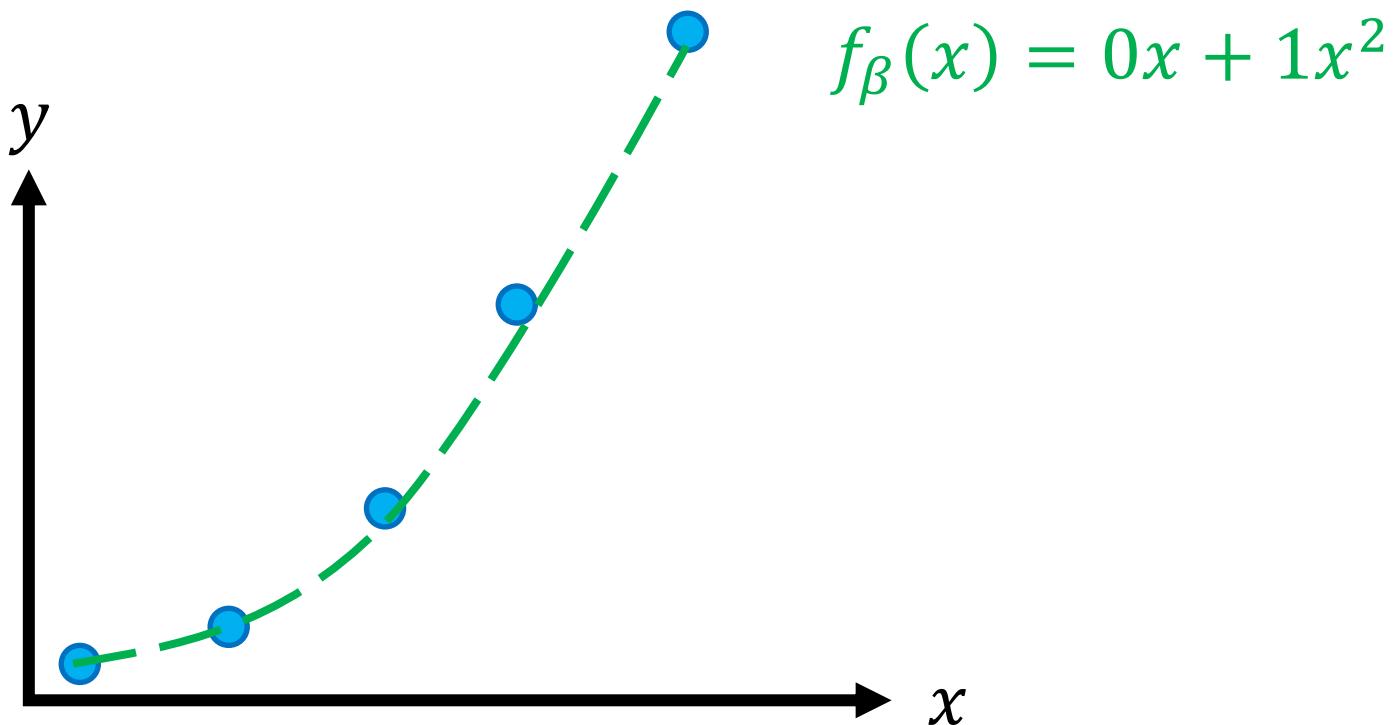
- Consider the feature map  $\phi: \mathbb{R} \rightarrow \mathbb{R}^2$  given by

$$\phi(x) = \begin{bmatrix} x \\ x^2 \end{bmatrix}$$

- Then, the model family is

$$f_{\beta}(x) = \beta_1 x + \beta_2 x^2$$

# Quadratic Feature Map



In our family for  $\beta = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ !

# Feature Maps

- Powerful strategy for encoding prior knowledge
- **Terminology**
  - $x$  is the **input** and  $\phi(x)$  are the **features**
  - Often used interchangeably

# Examples of Feature Maps

- **Polynomial features**

- $\phi(x) = \beta_1 + \beta_2 x_1 + \beta_3 x_2 + \beta_4 x_1^2 + \beta_5 x_1 x_2 + \beta_6 x_2^2 + \dots$
- Quadratic features are very common; capture “feature interactions”
- Can use other nonlinearities (exponential, logarithm, square root, etc.)

- **Intercept term**

- $\phi(x) = [1 \quad x_1 \quad \dots \quad x_d]^T$
- Almost always used; captures constant effect

- **Encoding non-real inputs**

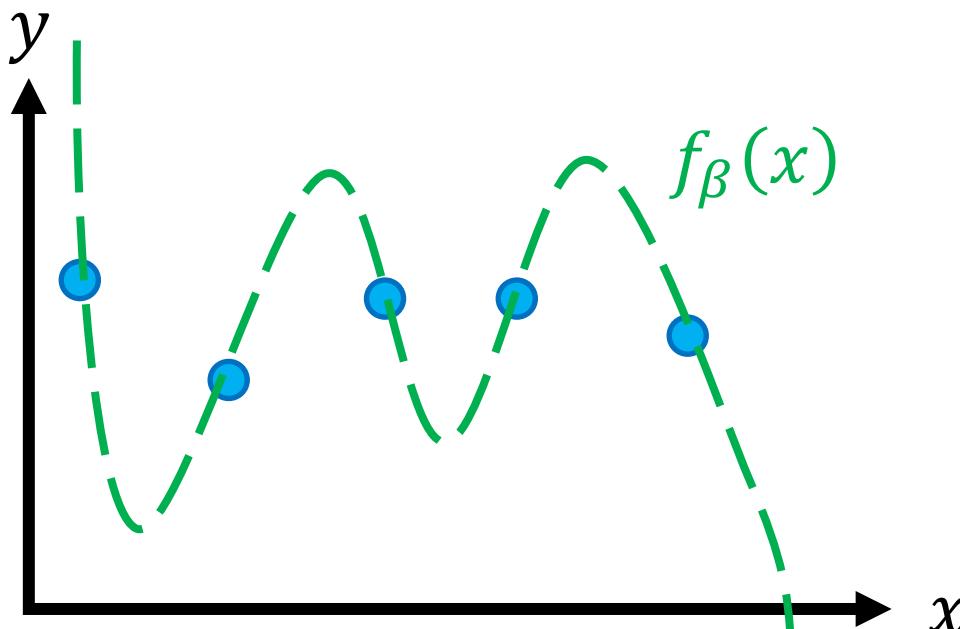
- E.g.,  $x = \text{“the food was good”}$  and  $y = 4$  stars
- $\phi(x) = [1(\text{“good”} \in x) \quad 1(\text{“bad”} \in x) \quad \dots]^T$

# Algorithm

- Reduces to linear regression
- **Step 1:** Compute  $\phi_i = \phi(\textcolor{blue}{x}_i)$  for each  $\textcolor{blue}{x}_i$  in  $Z$
- **Step 2:** Run linear regression with  $Z' = \{(\phi_1, y_1), \dots, (\phi_n, y_n)\}$

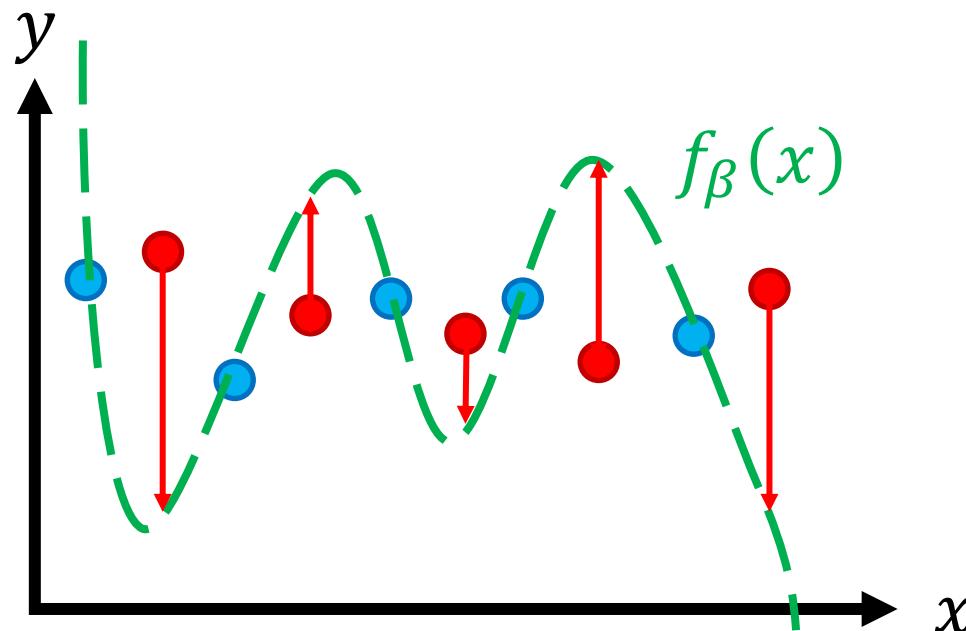
# Question

- **Why not throw in lots of features?**
  - $\phi(x) = \beta_1 + \beta_2 x_1 + \beta_3 x_2 + \beta_4 x_1^2 + \beta_5 x_1 x_2 + \beta_6 x_2^2 + \dots$
  - Can fit any  $n$  points using a polynomial of degree  $n$



# Prediction

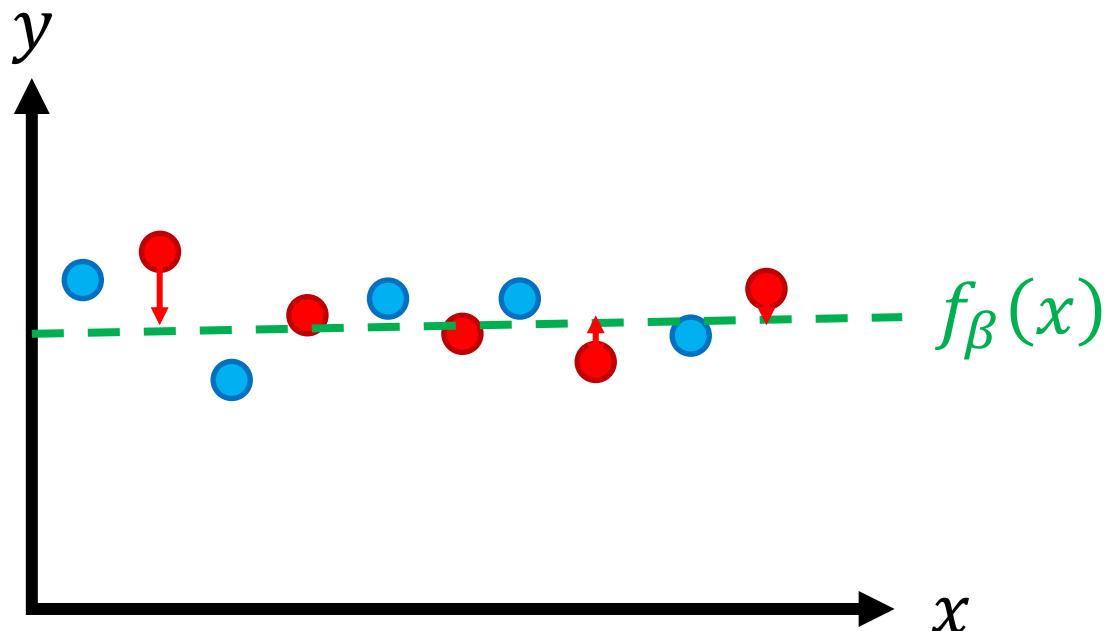
- **Issue:** The goal in machine learning is **prediction**
  - Given a **new** input  $x$ , predict the label  $\hat{y} = f_\beta(x)$



The errors on new inputs is very large!

# Prediction

- **Issue:** The goal in machine learning is **prediction**
  - Given a **new** input  $x$ , predict the label  $\hat{y} = f_\beta(x)$



Vanilla linear regression actually works better!

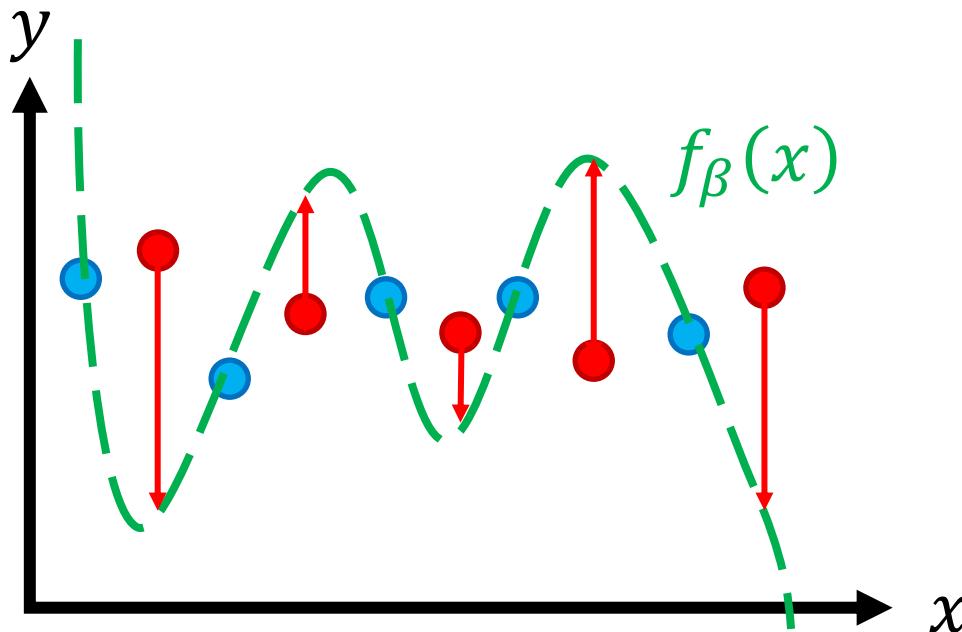
# Training vs. Test Data

- **Training data:** Examples  $Z = \{(x, y)\}$  used to fit our model
- **Test data:** New inputs  $x$  whose labels  $y$  we want to predict

# Overfitting vs. Underfitting

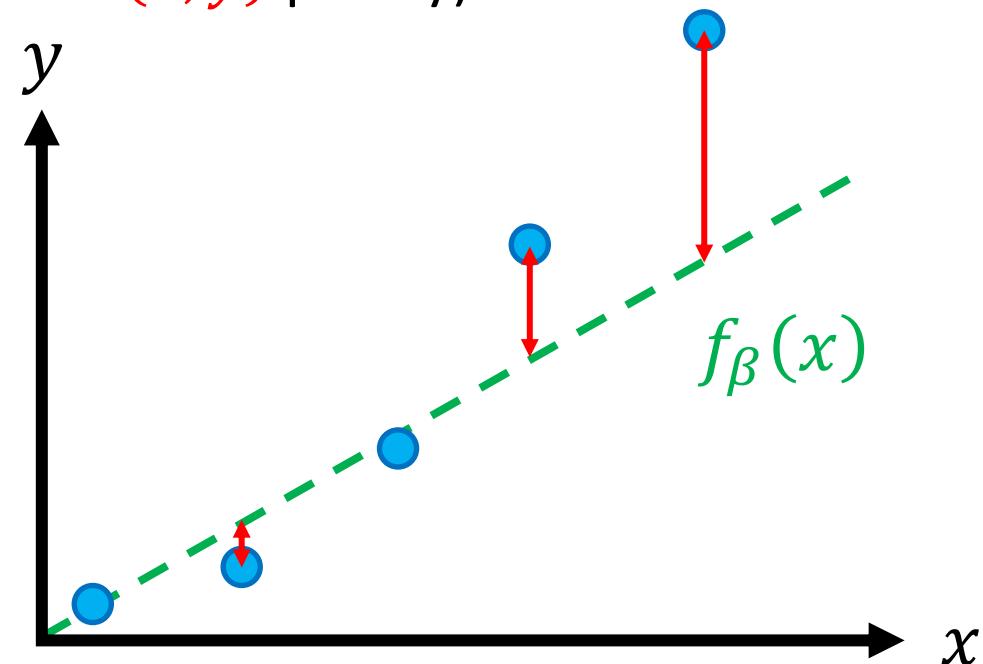
- **Overfitting**

- Fit the **training data**  $Z$  well
- Fit new **test data**  $(x, y)$  poorly



- **Underfitting**

- Fit the **training data**  $Z$  poorly
- (Necessarily fit new **test data**  $(x, y)$  poorly)

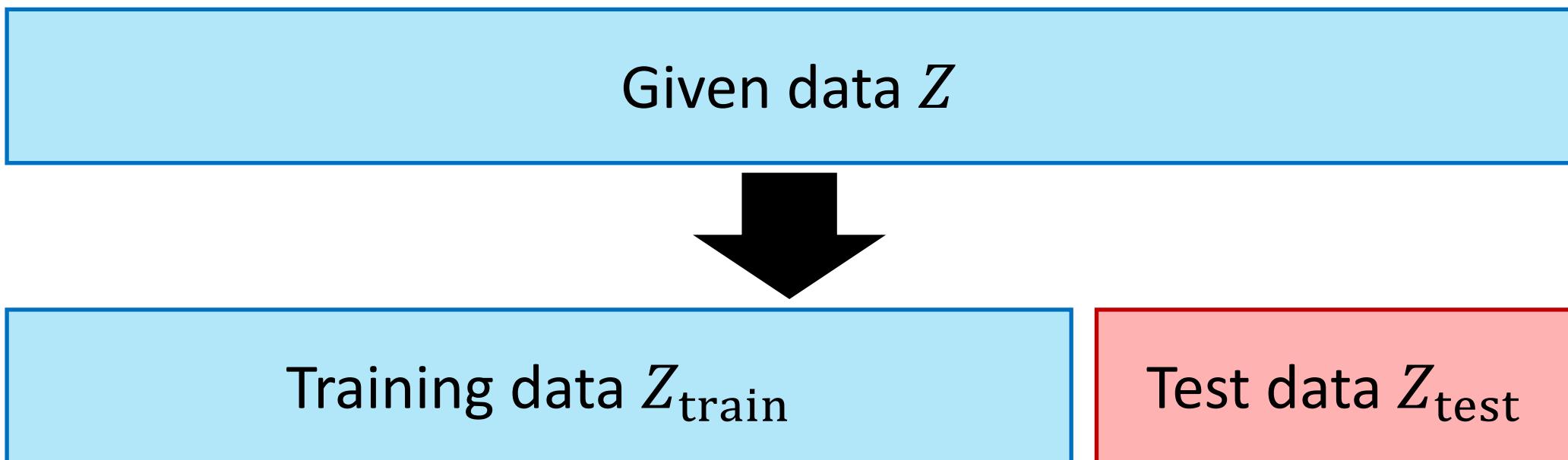


# Aside: Why Does Overfitting Happen?

- Overfitting typically due to fitting noise in the data
- **Noise in labels  $y_i$** 
  - True data generating process is more complex than we can capture
  - May depend on unobserved features
- **Noise in features  $x_i$** 
  - Measurement error in the feature values
  - Errors due to preprocessing
  - Some features might be irrelevant to the decision function

# Training/Test Split

- **Issue:** How to detect overfitting vs. underfitting?
- **Solution:** Use **held-out test data** to estimate loss on new data
  - Typically, randomly shuffle data first



# Training/Test Split Algorithm

- **Step 1:** Split  $Z$  into  $Z_{\text{train}}$  and  $Z_{\text{test}}$

Training data  $Z_{\text{train}}$

Test data  $Z_{\text{test}}$

- **Step 2:** Run linear regression with  $Z_{\text{train}}$  to obtain  $\hat{\beta}(Z_{\text{train}})$

- **Step 3:** Evaluate

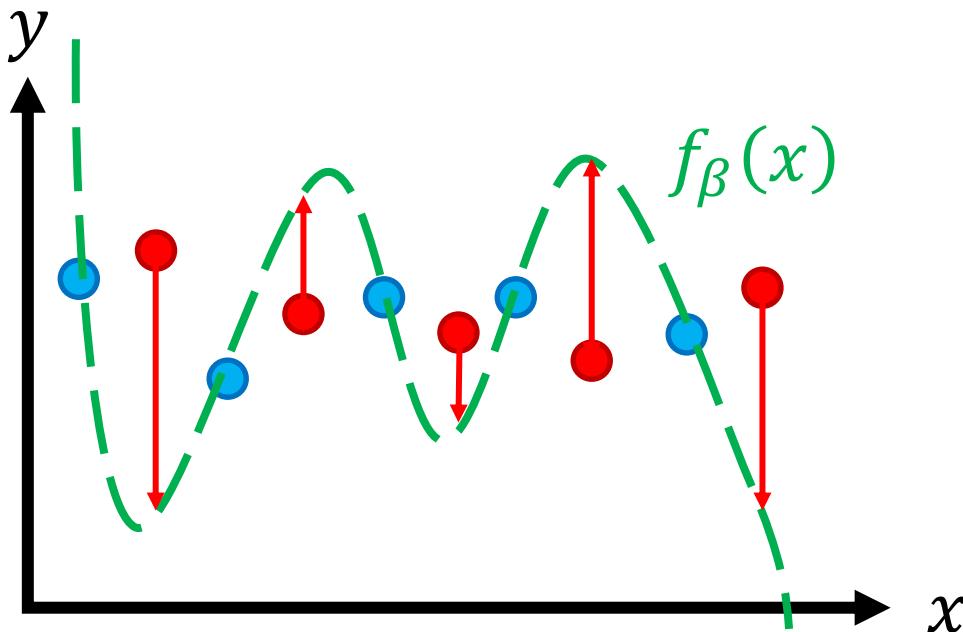
- **Training loss:**  $L_{\text{train}} = L(\hat{\beta}(Z_{\text{train}}); Z_{\text{train}})$

- **Test (or generalization) loss:**  $L_{\text{test}} = L(\hat{\beta}(Z_{\text{train}}); Z_{\text{test}})$

# Training/Test Split Algorithm

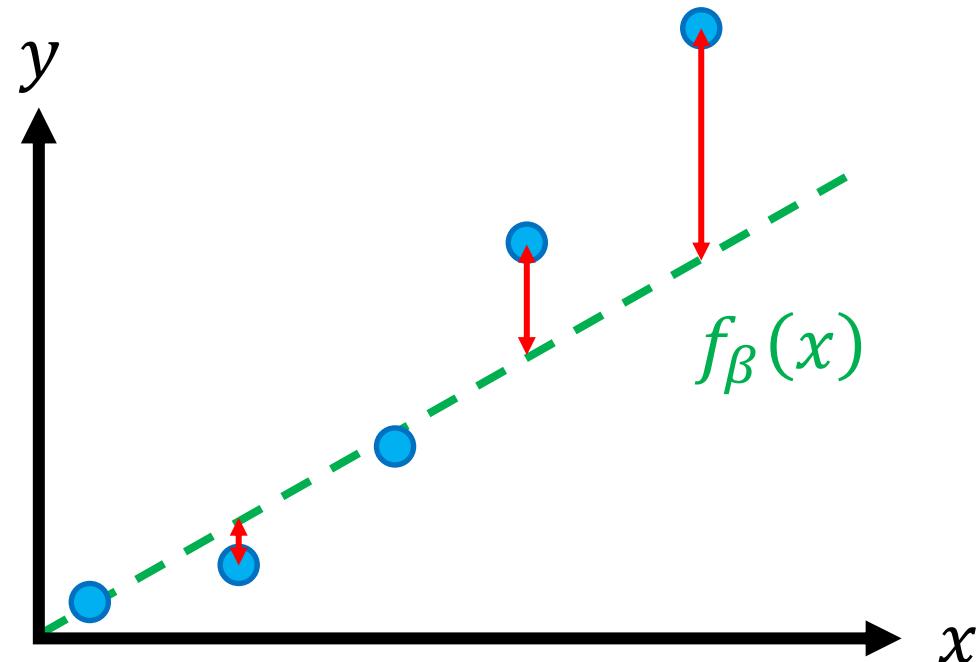
- **Overfitting**

- $L_{\text{train}}$  is small
- $L_{\text{test}}$  is large



- **Underfitting**

- $L_{\text{train}}$  is large
- $L_{\text{test}}$  is large



# How to Fix Underfitting/Overfitting?

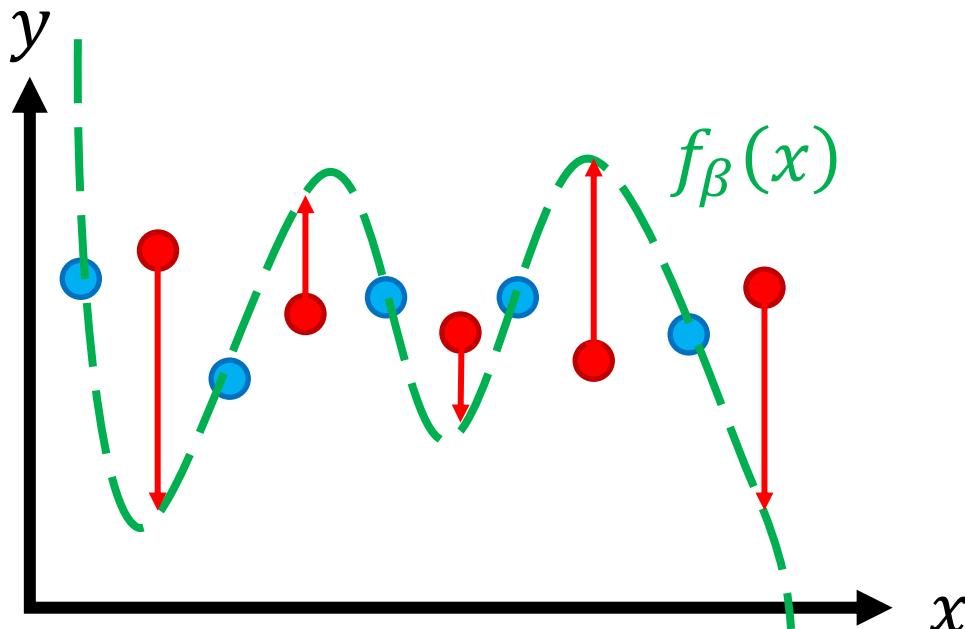
- Choose the right model family!

# Role of Capacity

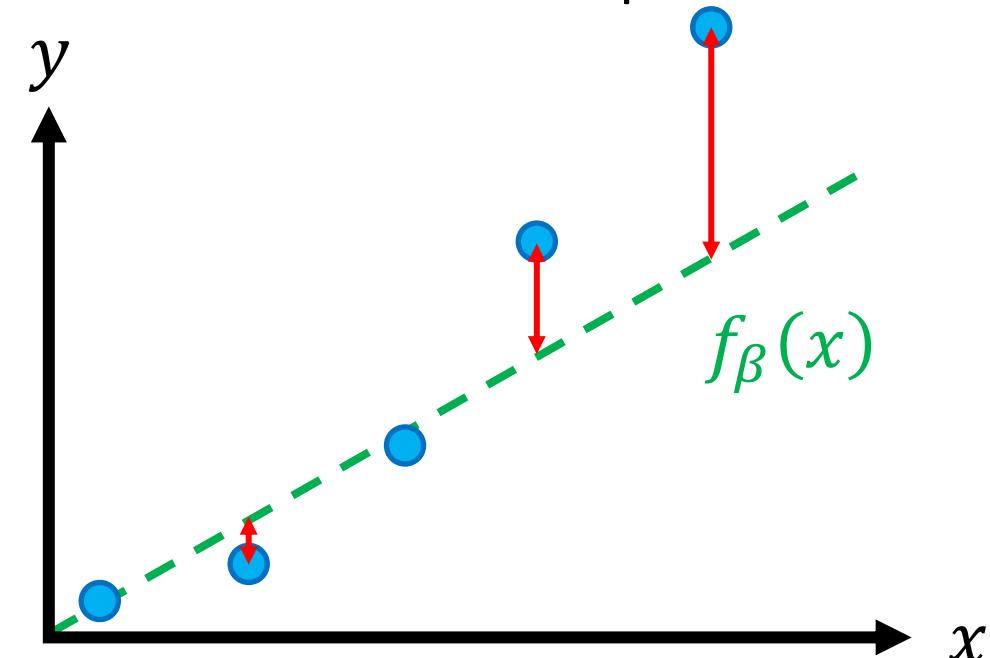
- **Capacity** of a model family captures “complexity” of data it can fit
  - Higher capacity → more likely to overfit (model family has high **variance**)
  - Lower capacity → more likely to underfit (model family has high **bias**)
- For linear regression, capacity corresponds to feature dimension  $d$ 
  - I.e., number of features in  $\phi(x)$

# Bias-Variance Tradeoff

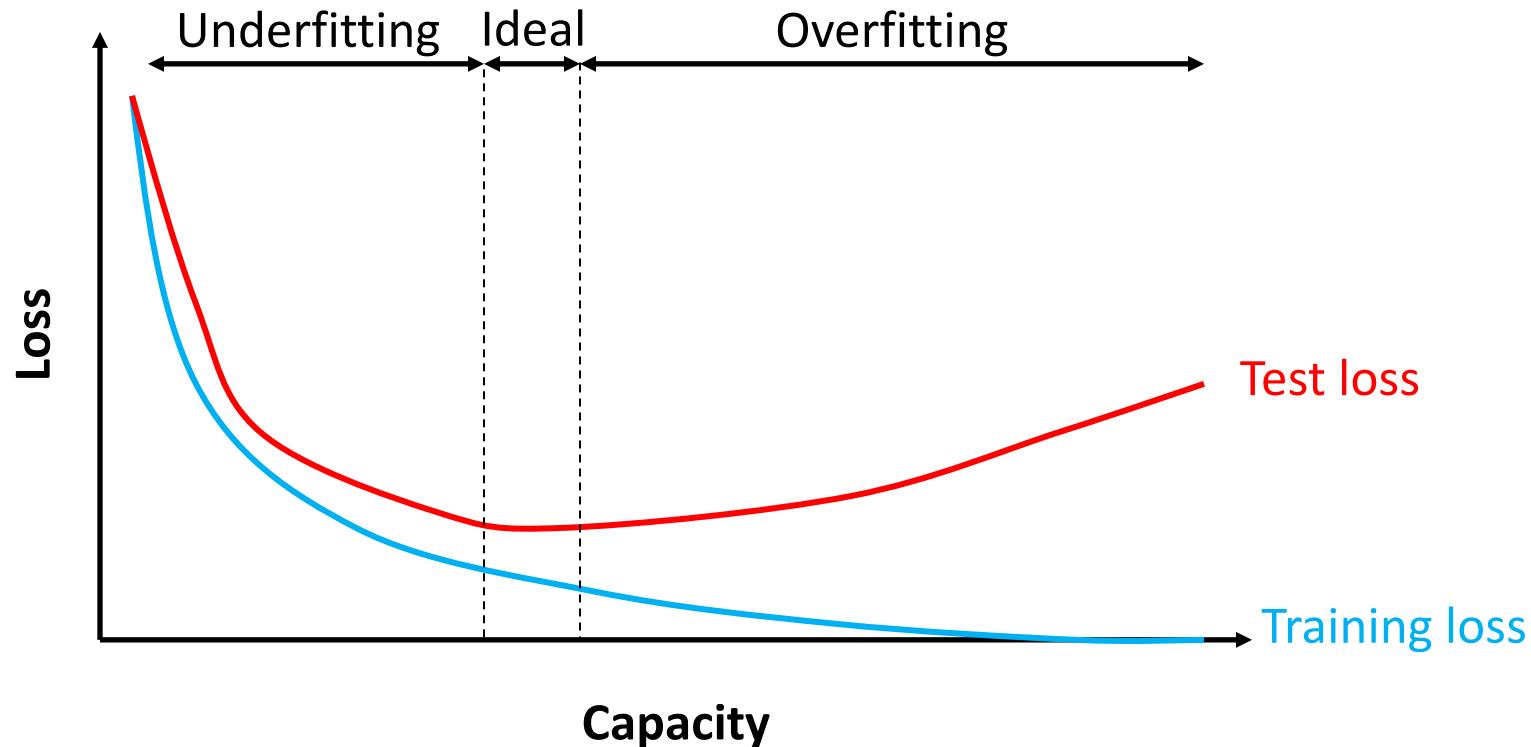
- **Overfitting (high variance)**
  - High capacity model capable of fitting complex data
  - Insufficient data to constrain it



- **Underfitting (high bias)**
  - Low capacity model that can only fit simple data
  - Sufficient data but poor fit



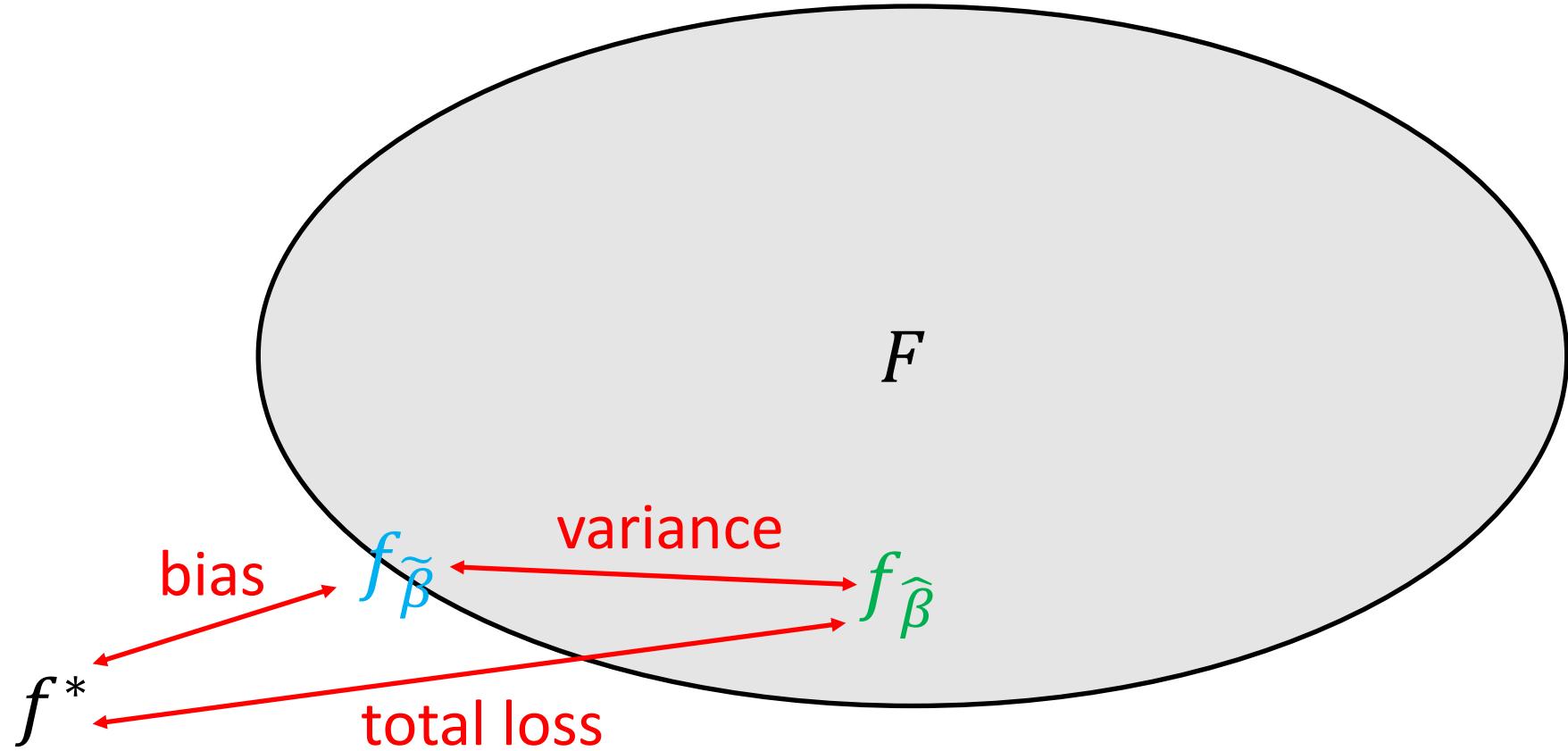
# Bias-Variance Tradeoff



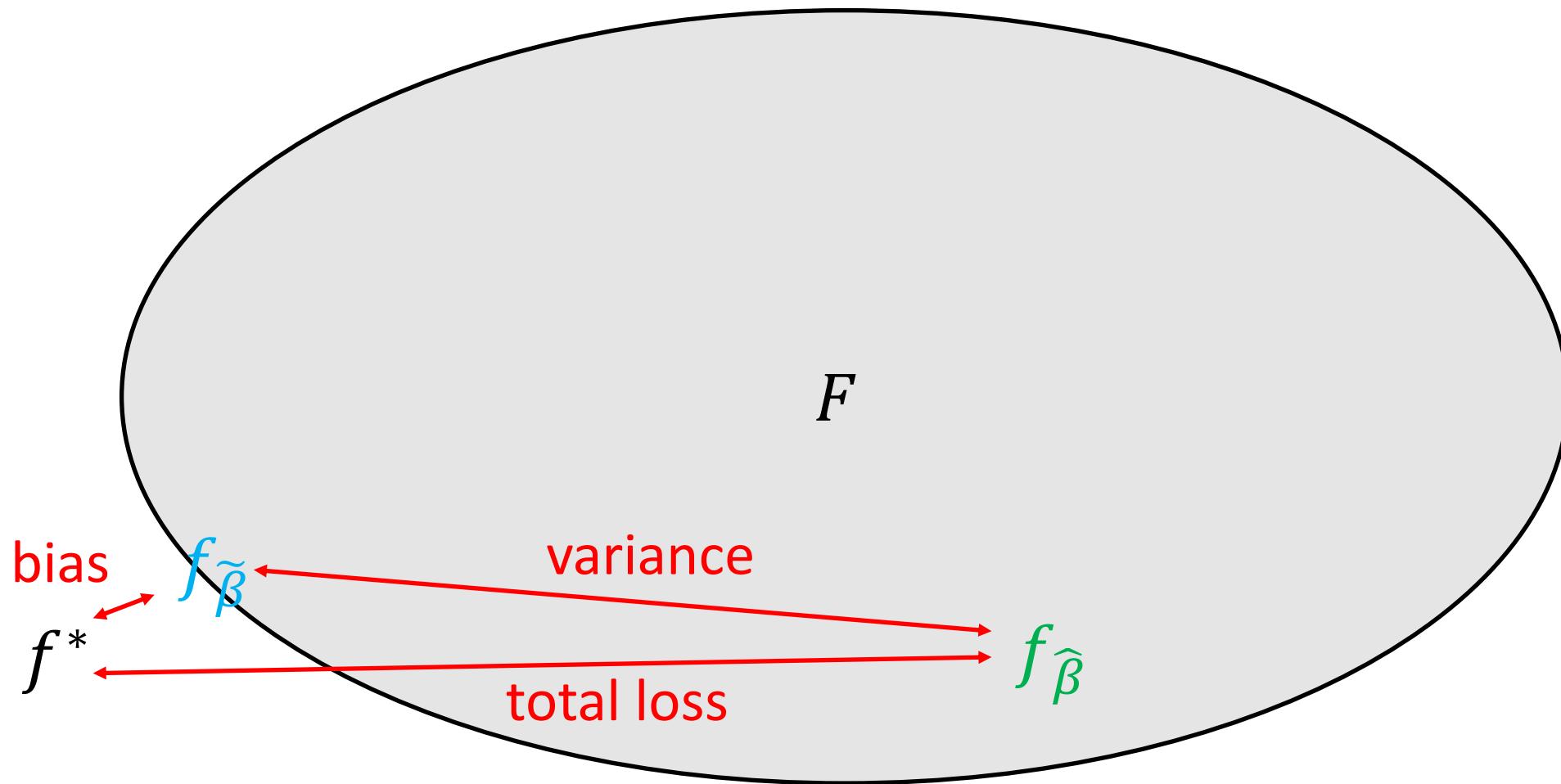
# Bias-Variance Tradeoff

- For linear regression, increasing feature dimension  $d$ ...
  - Tends to **increase capacity**
  - Tends to **decrease bias** but **increase variance**
- Need to construct  $\phi$  to balance tradeoff between bias and variance
  - **Rule of thumb:**  $n \approx d \log d$
  - Large fraction of data science work is data cleaning + feature engineering

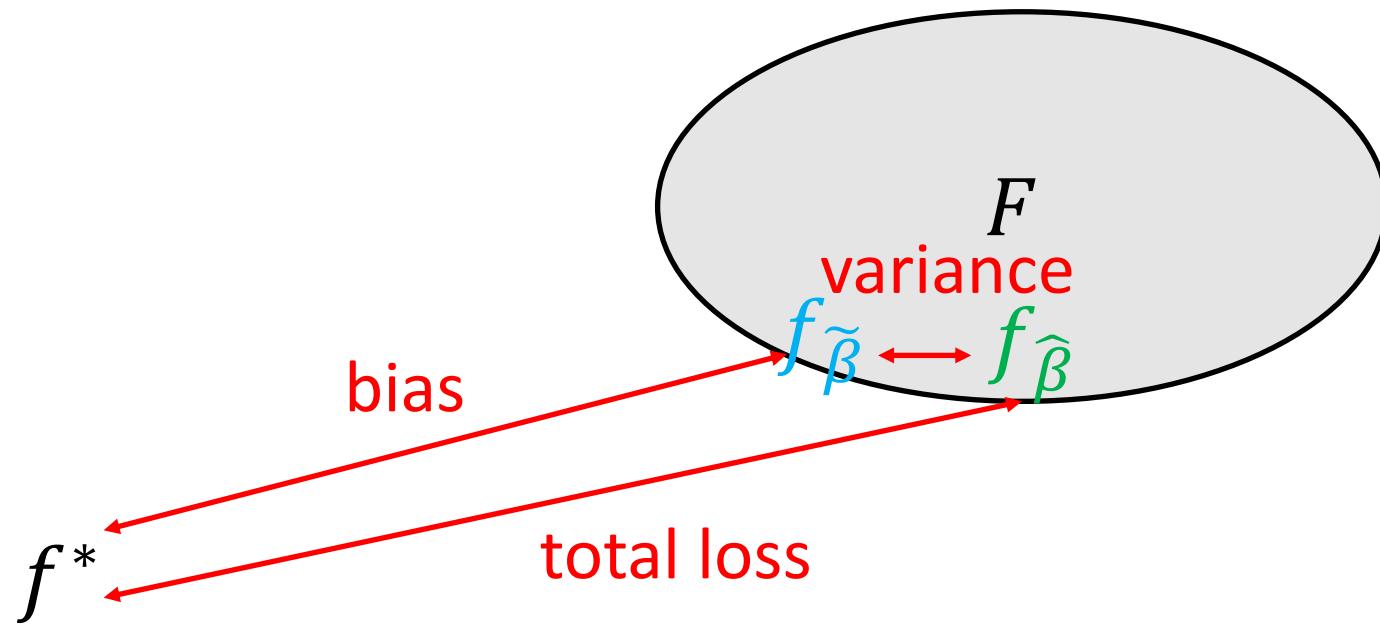
# Recap: Bias-Variance Tradeoff



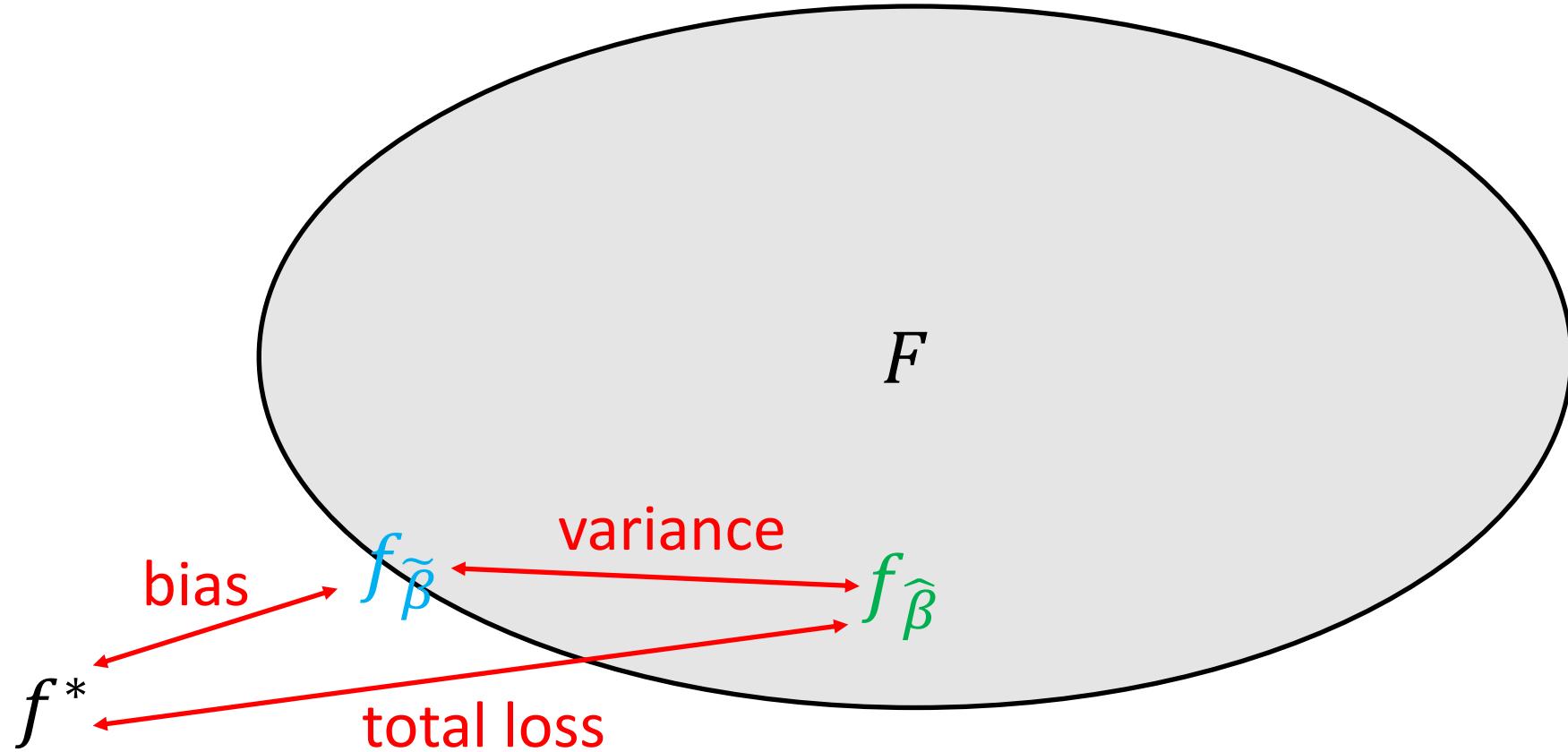
# Recap: Bias-Variance Tradeoff (Overfitting)



# Recap: Bias-Variance Tradeoff (Underfitting)



# Recap: Bias-Variance Tradeoff (Ideal)



# Linear Regression with $L_2$ Regularization

- Original loss + regularization:

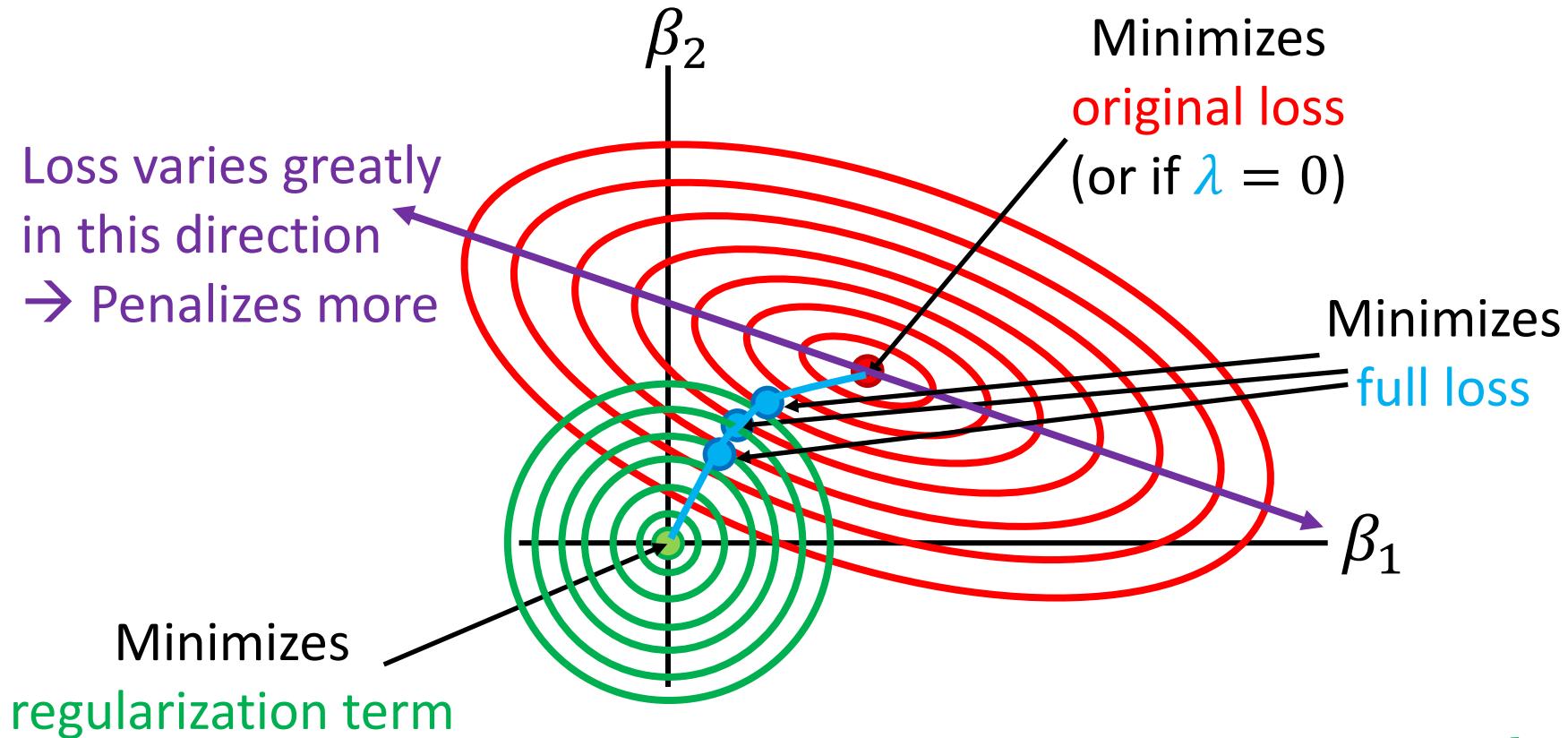
$$\begin{aligned} L(\beta; Z) &= \frac{1}{n} \sum_{i=1}^n (y_i - \beta^\top x_i)^2 + \lambda \cdot \|\beta\|_2^2 \\ &= \frac{1}{n} \sum_{i=1}^n (y_i - \beta^\top x_i)^2 + \lambda \sum_{j=1}^d \beta_j^2 \end{aligned}$$

- $\lambda$  is a **hyperparameter** that must be tuned (satisfies  $\lambda \geq 0$ )

# Intuition on $L_2$ Regularization

- **Why does it help?**
  - Encourages “simple” functions
  - E.g., as  $\lambda \rightarrow \infty$ , obtain  $\beta = 0$
  - Use  $\lambda$  to tune bias-variance tradeoff

# Intuition on $L_2$ Regularization



$$L(\beta; Z) = \frac{1}{n} \sum_{i=1}^n (y_i - \beta^\top x_i)^2 + \lambda \sum_{j=1}^d \beta_j^2$$

- At this point, the gradients are **equal** (with opposite sign)
- Tradeoff depends on choice of  $\lambda$

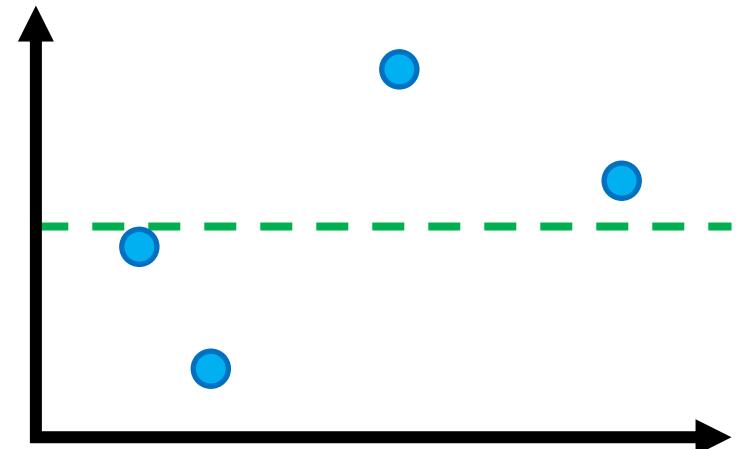
# Regularization and Intercept Term

- If using intercept term ( $\phi(x) = [1 \quad x_1 \quad \dots \quad x_d]^\top$ ), no penalty on  $\beta_1$ :

$$L(\beta; Z) = \frac{1}{n} \sum_{i=1}^n (y_i - \beta^\top x_i)^2 + \lambda \sum_{j=2}^d \beta_j^2$$

Sum from  $j = 2$

- As  $\lambda \rightarrow \infty$ , we have  $\beta_2 = \dots = \beta_d = 0$ 
  - I.e., only fit  $\beta_1$  (which yields  $\hat{\beta}_1(Z) = \text{mean}(\{y_i\}_{i=1}^n)$ )



# Feature Standardization

- Unregularized linear regression is invariant to feature scaling
  - Suppose we scale  $x_{ij} \leftarrow 2x_{ij}$  for all examples  $x_i$
  - Without regularization, simply use  $\beta_j \leftarrow \beta_j/2$  to obtain equivalent solution
  - In particular,  $\frac{\beta_j}{2} \cdot 2x_{ij} = \beta_j \cdot x_{ij}$
- Not true for regularized regression!
  - Penalty  $(\beta_j/2)^2$  is scaled by 1/4 (not cancelled out!)

$$L(\beta; Z) = \frac{1}{n} \sum_{i=1}^n (y_i - \beta^\top x_i)^2 + \lambda \sum_{j=2}^d \beta_j^2$$

# Feature Standardization

- **Solution:** Rescale features to zero mean and unit variance

$$x_{i,j} \leftarrow \frac{x_{i,j} - \mu_j}{\sigma_j} \quad \mu_j = \frac{1}{N} \sum_{i=1}^N x_{i,j} \quad \sigma_j = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_{i,j} - \mu_j)^2}$$

- **Note:** When using intercept term, do not rescale  $x_1 = 1$
- Can be sensitive to outliers (fix by dropping outliers)
- **Must use same transformation during training and for prediction**
  - Compute standardization on training data and use on test data

# General Regularization Strategy

- **Original loss + regularization:**

$$L_{\text{new}}(\beta; Z) = L(\beta; Z) + \lambda \cdot R(\beta)$$

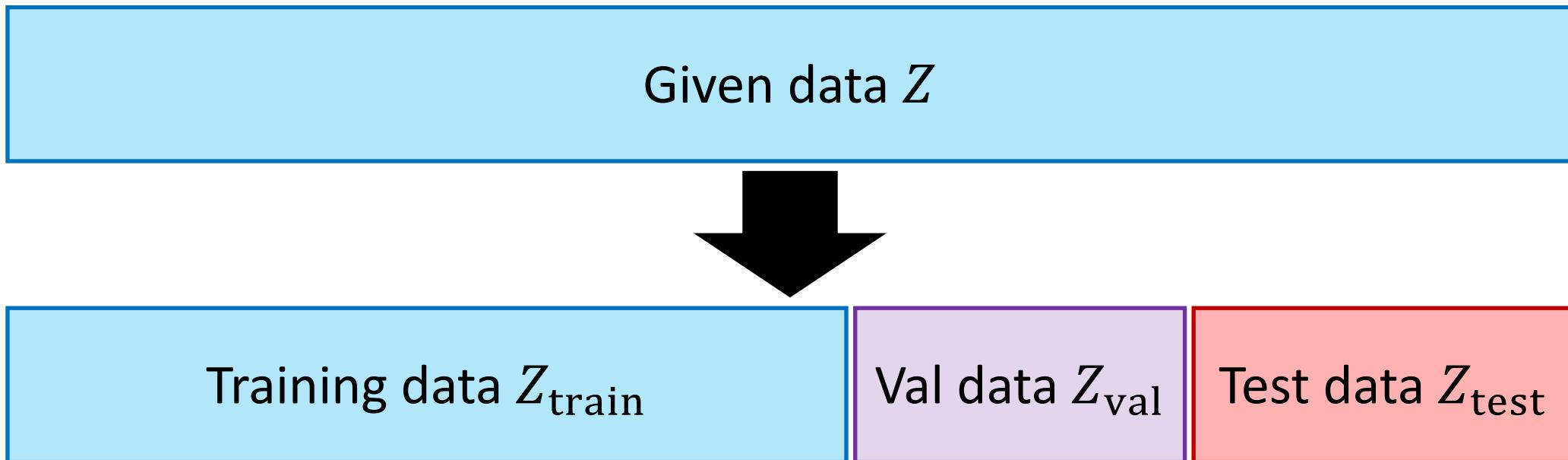
- Offers a way to express a preference “simpler” functions in family
- Typically, regularization is independent of data

# Hyperparameter Tuning

- $\lambda$  is a **hyperparameter** that must be tuned (satisfies  $\lambda \geq 0$ )
- **Naïve strategy:** Try a few different candidates  $\lambda_t$  and choose the one that minimizes the test loss
- **Problem:** We may overfit the test set!
  - Major problem if we have more hyperparameters

# Training/Val/Test Split

- **Goal:** Choose best hyperparameter  $\lambda$ 
  - Can also compare different model families, feature maps, etc.
- **Solution:** Optimize  $\lambda$  on a **held-out validation data**
  - **Rule of thumb:** 60/20/20 split



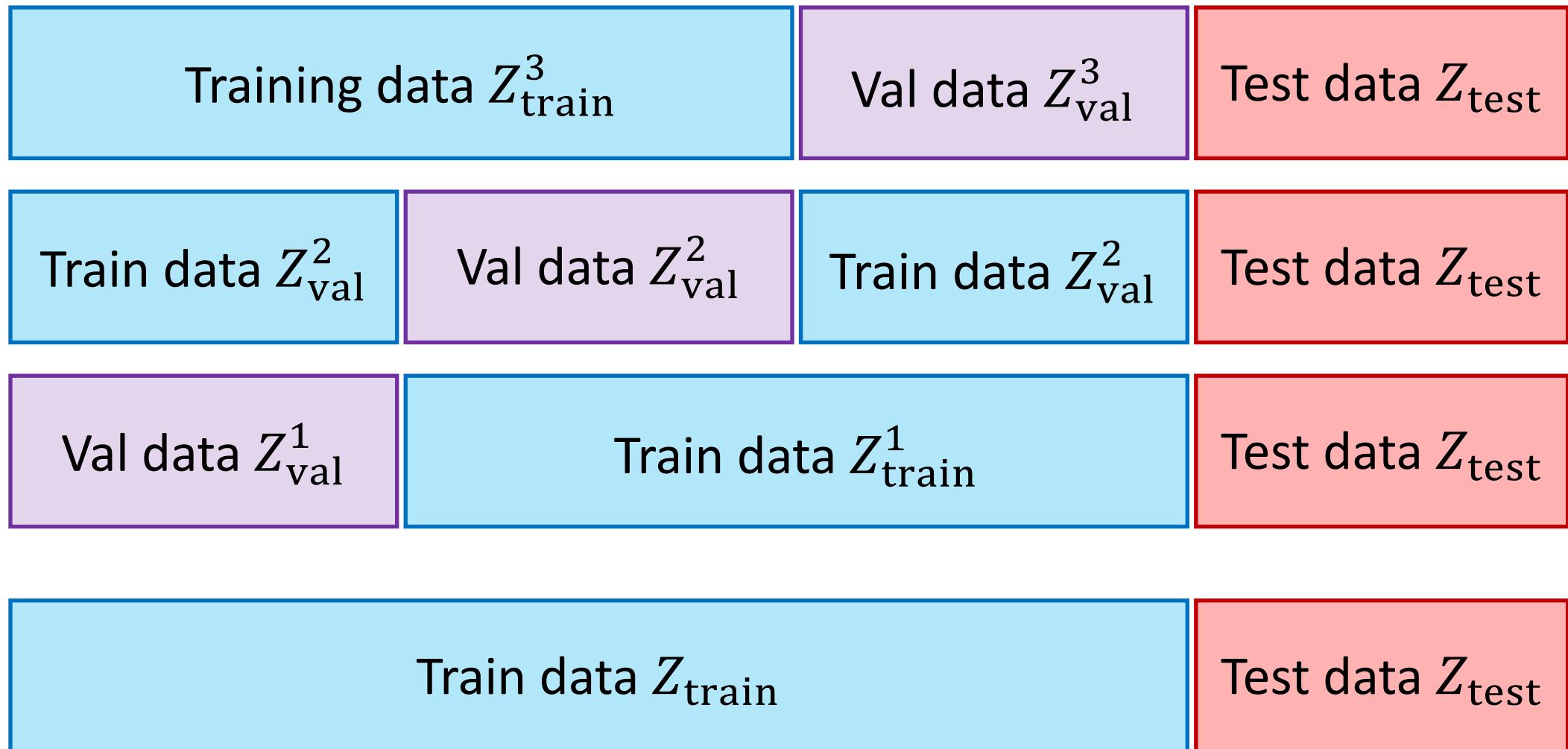
# Basic Cross Validation Algorithm

- **Step 1:** Split  $Z$  into  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ , and  $Z_{\text{test}}$



- **Step 2:** For  $t \in \{1, \dots, h\}$ :
  - **Step 2a:** Run linear regression with  $Z_{\text{train}}$  and  $\lambda_t$  to obtain  $\hat{\beta}(Z_{\text{train}}, \lambda_t)$
  - **Step 2b:** Evaluate validation loss  $L_{\text{val}}^t = L(\hat{\beta}(Z_{\text{train}}, \lambda_t); Z_{\text{val}})$
- **Step 3:** Use best  $\lambda_t$ 
  - Choose  $t' = \arg \min_t L_{\text{val}}^t$  with lowest validation loss
  - Re-run linear regression with  $Z_{\text{train}}$  and  $\lambda_{t'}$  to obtain  $\hat{\beta}(Z_{\text{train}}, \lambda_{t'})$

# Example: 3-Fold Cross Validation



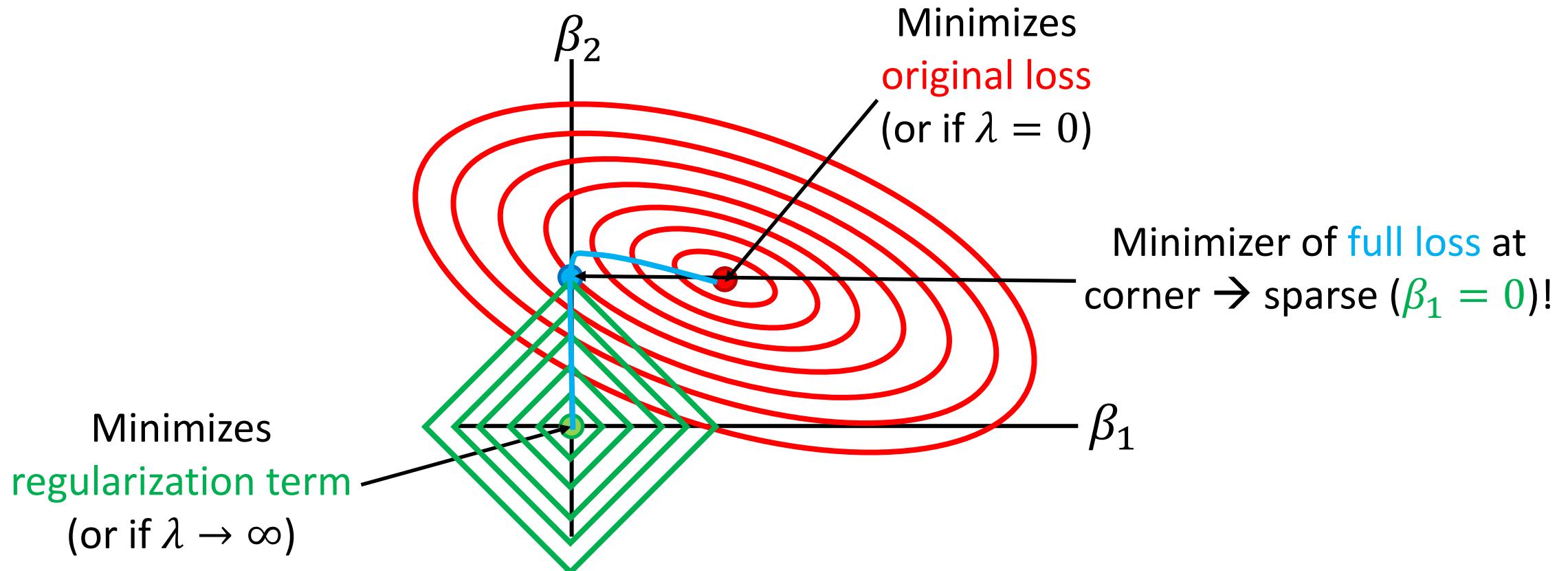
# $k$ -Fold Cross-Validation

- If  $Z$  is small, then splitting it can reduce performance
  - Can use  $Z_{\text{train}} \cup Z_{\text{val}}$  in Step 3
- **Alternative:**  $k$ -fold cross-validation (e.g.,  $k = 3$ )
  - Split  $Z$  into  $Z_{\text{train}}$  and  $Z_{\text{test}}$
  - Split  $Z_{\text{train}}$  into  $k$  disjoint sets  $Z_{\text{val}}^s$ , and let  $Z_{\text{train}}^s = \bigcup_{s' \neq s} Z_{\text{val}}^{s'}$
  - Use  $\lambda'$  that works best on average across  $s \in \{1, \dots, k\}$  with  $Z_{\text{train}}$
  - Chooses better  $\lambda'$  than above strategy
- **Compute vs. accuracy tradeoff**
  - As  $k \rightarrow N$ , the model becomes more accurate
  - But algorithm becomes more computationally expensive

# $L_1$ Regularization

- **Sparsity:** Can we minimize  $\|\beta\|_0 = |\{j \mid \beta_j \neq 0\}|$ ?
  - That is, the number of nonzero components of  $\beta$
  - Improves interpretability (automatic **feature selection!**)
  - Also serves as a **strong** regularizer ( $n \sim s \log d$ , where  $s = \|\beta\|_0$ )
- **Challenge:**  $\|\beta\|_0$  is not differentiable, making it hard to optimize
- **Solution**
  - We can instead use an  $L_1$  norm as the regularizer!
  - Still harder to optimize than  $L_2$  norm, but at least it is convex

# Intuition on $L_1$ Regularization



$$L(\beta; Z) = \frac{1}{n} \sum_{i=1}^n (y_i - \beta^\top x_i)^2 + \lambda \sum_{j=1}^d |\beta_j|$$

# $L_1$ Regularization for Feature Selection

- **Step 1:** Construct a lot of features and add to feature map
- **Step 2:** Use  $L_1$  regularized regression to “select” subset of features
  - I.e., coefficient  $\beta_j \neq 0 \rightarrow$  feature  $j$  is selected)
- **Optional:** Remove unselected features from the feature map and run vanilla linear regression (a.k.a. ordinary least squares)

# Minimizing the MSE Loss

- Recall that linear regression minimizes the loss

$$L(\beta; Z) = \frac{1}{n} \sum_{i=1}^n (y_i - \beta^\top x_i)^2$$

- **Closed-form solution:** Compute using matrix operations
- **Optimization-based solution:** Search over candidate  $\beta$

# Vectorizing Linear Regression

$$\begin{bmatrix} f_{\beta}(x_1) \\ \vdots \\ f_{\beta}(x_n) \end{bmatrix} = \begin{bmatrix} \beta^T x_1 \\ \vdots \\ \beta^T x_n \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^d \beta_j x_{1,j} \\ \vdots \\ \sum_{j=1}^d \beta_j x_{n,j} \end{bmatrix} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,d} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,d} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_d \end{bmatrix} = X\beta$$

||

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = Y$$

**Summary:**  $Y \approx X\beta$

# Vectorizing Mean Squared Error

$$L(\beta; Z) = \frac{1}{n} \sum_{i=1}^n (y_i - \beta^\top x_i)^2 = \frac{1}{n} \|Y - X\beta\|_2^2$$
$$\|z\|_2^2 = \sum_{i=1}^n z_i^2$$

The diagram illustrates the vectorization of the mean squared error loss function. It shows two vectors:  $y$  (blue) and  $f_\beta(x)$  (green). Arrows point from these vectors to the terms in the sum of squares formula. One arrow points from  $y$  to the term  $(y_i - \beta^\top x_i)^2$ , and another arrow points from  $f_\beta(x)$  to the same term. A third arrow points from the sum of squares formula up to the final result.

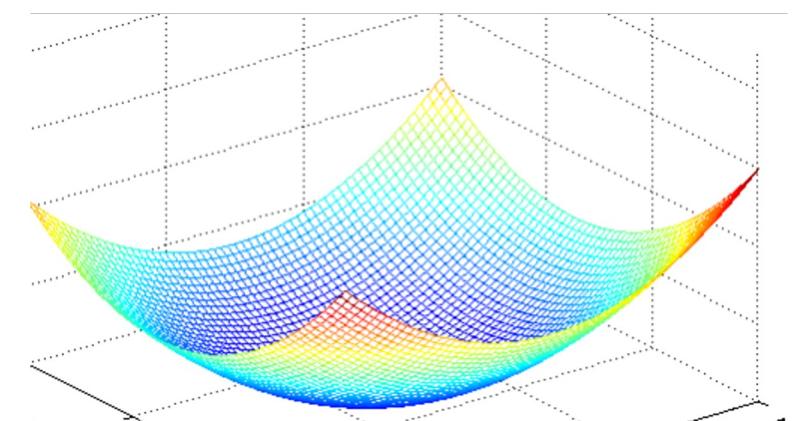
# Strategy 1: Closed-Form Solution

- Recall that linear regression minimizes the loss

$$L(\beta; Z) = \frac{1}{n} \|Y - X\beta\|_2^2$$

- Minimum solution has gradient equal to zero:

$$\nabla_{\beta} L(\hat{\beta}(Z); Z) = 0$$



# Strategy 1: Closed-Form Solution

- The gradient is

$$\nabla_{\beta} L(\beta; Z) = \nabla_{\beta} \frac{1}{n} \|Y - X\beta\|_2^2 = -\frac{2}{n} X^T Y + \frac{2}{n} X^T X \beta$$

- Setting  $\nabla_{\beta} L(\hat{\beta}; Z) = 0$ , we have  $X^T X \hat{\beta} = X^T Y$

# Strategy 1: Closed-Form Solution

- Setting  $\nabla_{\beta} L(\hat{\beta}; Z) = 0$ , we have  $X^T X \hat{\beta} = X^T Y$
- Assuming  $X^T X$  is invertible, we have

$$\hat{\beta}(Z) = (X^T X)^{-1} X^T Y$$

# Note on Invertibility

- Closed-form solution only **unique** if  $X^T X$  is invertible
  - Otherwise, **multiple solutions exist** to  $X^T X \hat{\beta} = X^T Y$
  - **Intuition:** Underconstrained system of linear equations
- **Example:**

$$\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} \hat{\beta}_1 \\ \hat{\beta}_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

- In this case, any  $\hat{\beta}_2 = 2 - \hat{\beta}_1$  is a solution

# Shortcomings of Closed-Form Solution

- Computing  $\hat{\beta}(Z) = (X^\top X)^{-1} X^\top Y$  can be challenging
- **Computing  $(X^\top X)^{-1}$  is  $O(d^3)$** 
  - $d = 10^4$  features  $\rightarrow O(10^{12})$
  - Even storing  $X^\top X$  requires a lot of memory
- **Numerical accuracy issues due to “ill-conditioning”**
  - $X^\top X$  is “barely” invertible
  - Then,  $(X^\top X)^{-1}$  has large variance along some dimension
  - Regularization helps (more on this later)

# Strategy 2: Gradient Descent

- **Gradient descent:** Update  $\beta$  based on gradient  $\nabla_{\beta} L(\beta; Z)$  of  $L(\beta; Z)$ :

$$\beta_{t+1} \leftarrow \beta_t - \alpha \cdot \nabla_{\beta} L(\beta_t; Z)$$

- **Intuition:** The gradient is the direction along which  $L(\beta; Z)$  changes most quickly as a function of  $\beta$
- $\alpha \in \mathbb{R}$  is a hyperparameter called the **learning rate**
  - More on this later

# Strategy 2: Gradient Descent

- Choose initial value for  $\beta$
- Until we reach a minimum:
  - Choose a new value for  $\beta$  to reduce  $L(\beta; Z)$

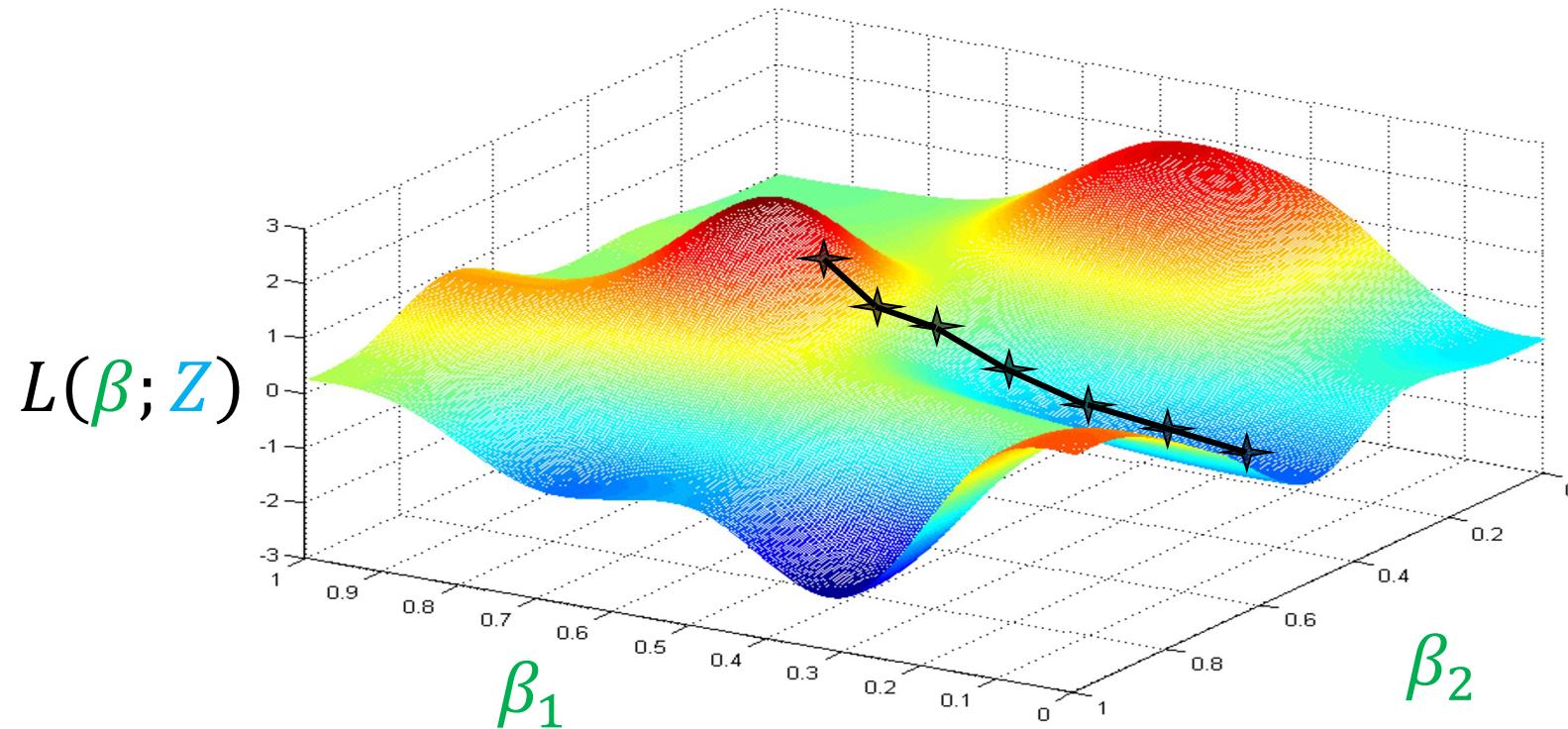


Figure by Andrew Ng

# Strategy 2: Gradient Descent

- Choose initial value for  $\beta$
- Until we reach a minimum:
  - Choose a new value for  $\beta$  to reduce  $L(\beta; Z)$

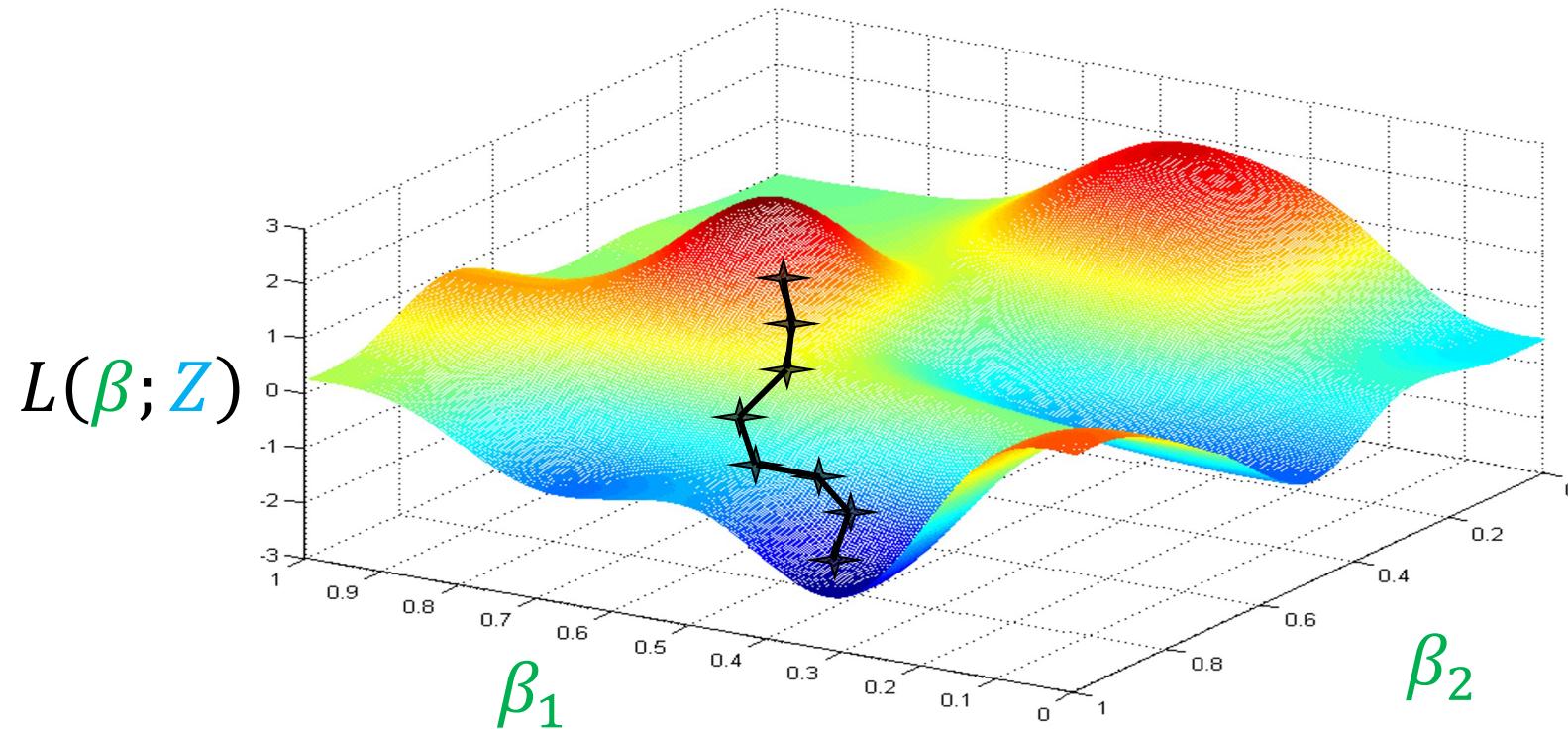
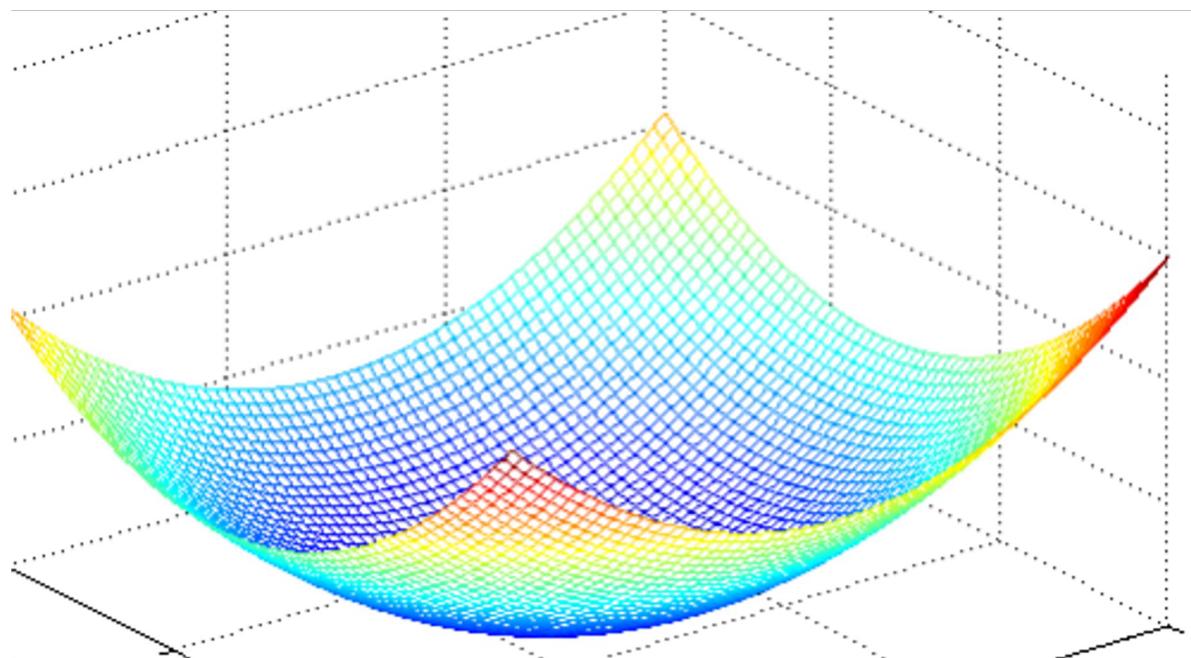


Figure by Andrew Ng

# Strategy 2: Gradient Descent

- Choose initial value for  $\beta$
- Until we reach a minimum:
  - Choose a new value for  $\beta$  to reduce  $L(\beta; Z)$



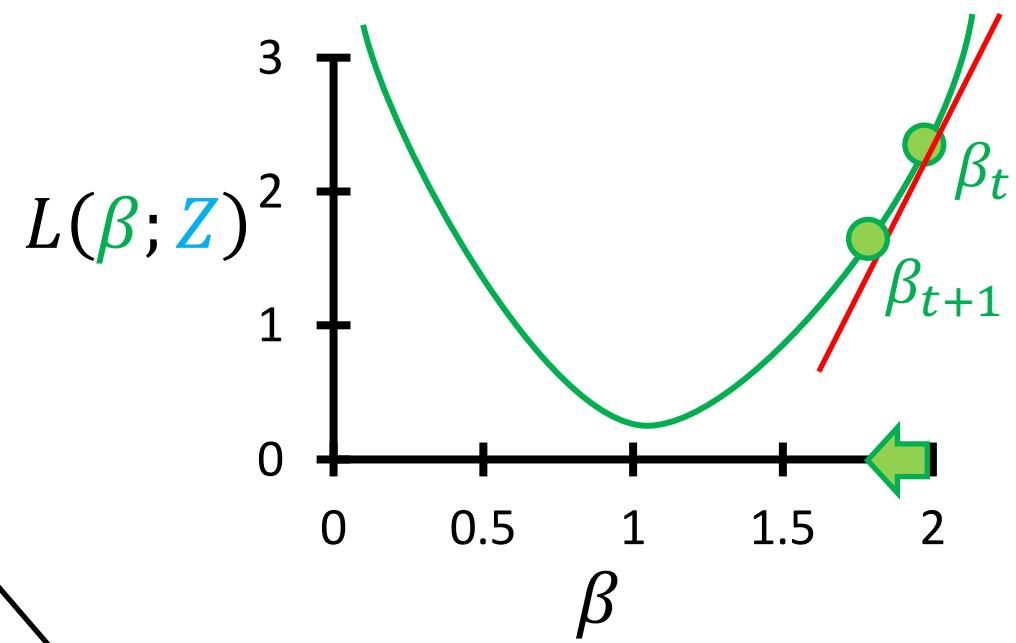
Linear regression loss is convex, so no local minima

# Strategy 2: Gradient Descent

- Initialize  $\beta_1 = 0$
- Repeat until convergence:

$$\beta_{t+1} \leftarrow \beta_t - \alpha \cdot \nabla_{\beta} L(\beta_t; Z)$$

- For linear regression, know the gradient from strategy 1



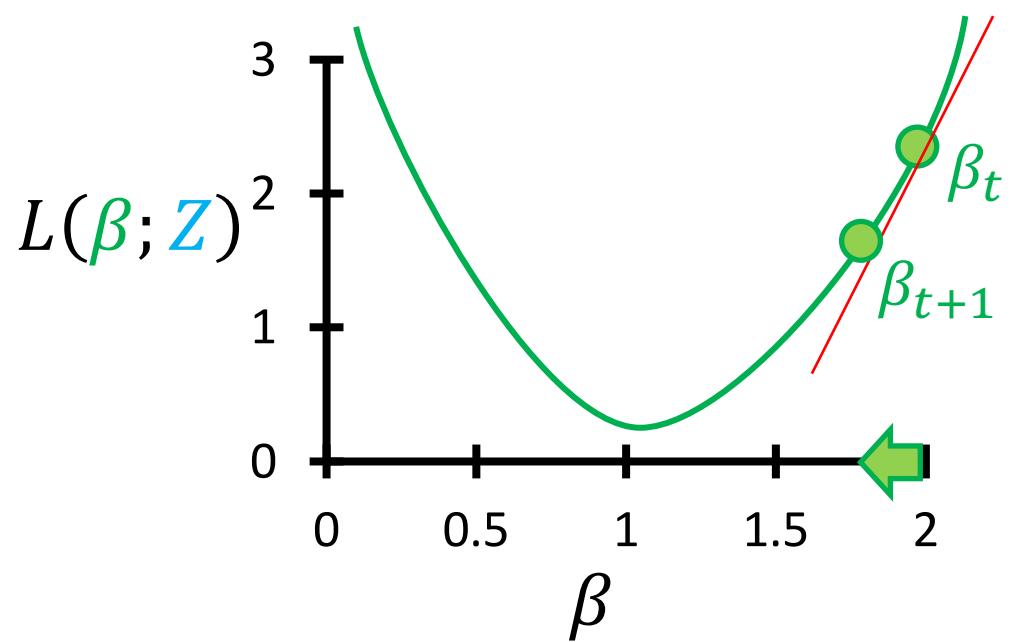
For in-place updates  $\beta \leftarrow \beta - \alpha \cdot \nabla_{\beta} L(\beta; Z)$ , compute all components of  $\nabla_{\beta} L(\beta; Z)$  before modifying  $\beta$

# Strategy 2: Gradient Descent

- Initialize  $\beta_1 = 0$
- Repeat until **convergence**:

$$\beta_{t+1} \leftarrow \beta_t - \alpha \cdot \nabla_{\beta} L(\beta_t; Z)$$

- For linear regression, know the gradient from strategy 1



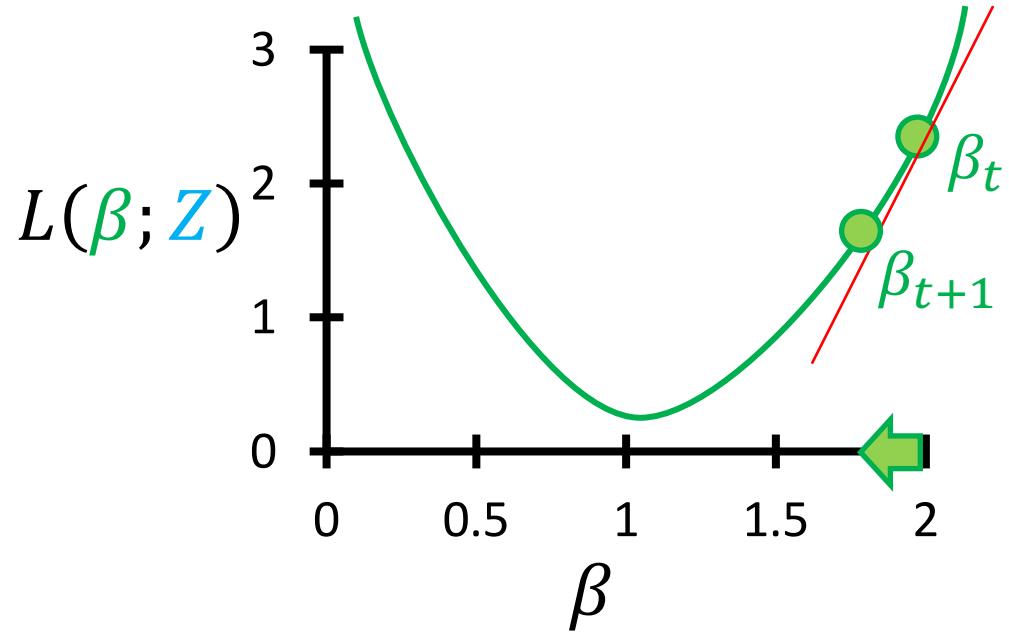
# Strategy 2: Gradient Descent

- Initialize  $\beta_1 = \vec{0}$
- Repeat until  $\|\beta_t - \beta_{t+1}\|_2 \leq \epsilon$ :

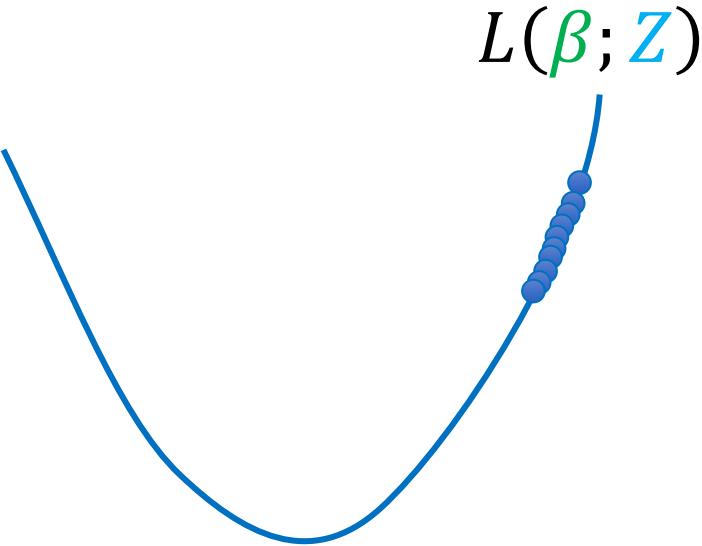
$$\beta_{t+1} \leftarrow \beta_t - \alpha \cdot \nabla_{\beta} L(\beta_t; Z)$$

- For linear regression, know the gradient from strategy 1

Hyperparameter defining convergence

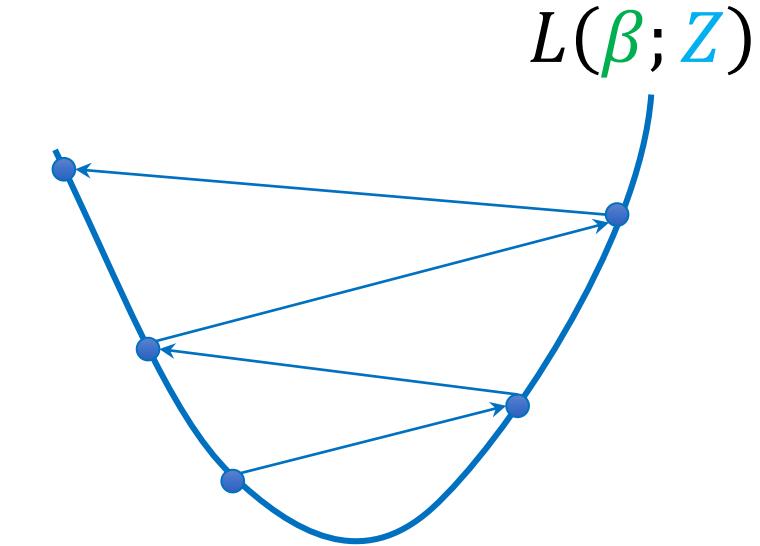


# Choice of Learning Rate $\alpha$



**Problem:**  $\alpha$  too small

- $L(\beta; Z)$  decreases slowly



**Problem:**  $\alpha$  too large

- $L(\beta; Z)$  increases!

Plot  $L(\beta_t; Z_{\text{train}})$  vs.  $t$  to diagnose these problems

# Choice of Learning Rate $\alpha$

- $\alpha$  is a hyperparameter for gradient descent that we need to choose
  - Can set just based on training data
- **Rule of thumb**
  - **$\alpha$  too small:** Loss decreases slowly
  - **$\alpha$  too large:** Loss increases!
- Try rates  $\alpha \in \{1.0, 0.1, 0.01, \dots\}$  (can tune further once one works)

# Comparison of Strategies

- **Closed-form solution**
  - No hyperparameters
  - Slow if  $n$  or  $d$  are large
- **Gradient descent**
  - Need to tune  $\alpha$
  - Scales to large  $n$  and  $d$
- For linear regression, there are better optimization algorithms, but gradient descent is very general
  - Accelerated gradient descent is an important tweak that improves performance in practice (and in theory)

# $L_2$ Regularized Linear Regression

- Recall that linear regression with  $L_2$  regularization minimizes the loss

$$L(\beta; Z) = \frac{1}{n} \sum_{i=1}^n (y_i - \beta^\top x_i)^2 + \lambda \sum_{j=1}^d \beta_j^2$$

# $L_2$ Regularized Linear Regression

- Recall that linear regression with  $L_2$  regularization minimizes the loss

$$L(\beta; Z) = \frac{1}{n} \sum_{i=1}^n (y_i - \beta^\top x_i)^2 + \lambda \sum_{j=1}^d \beta_j^2 = \frac{1}{n} \|Y - X\beta\|_2^2 + \lambda \|\beta\|_2^2$$

- Gradient is

$$\nabla_\beta L(\beta; Z) = -\frac{2}{n} X^\top Y + \frac{2}{n} X^\top X\beta + 2\lambda\beta$$

# Strategy 1: Closed-Form Solution

- Gradient is

$$\nabla_{\beta} L(\beta; Z) = -\frac{2}{n} X^T Y + \frac{2}{n} X^T X \beta + 2\lambda \beta$$

- Setting  $\nabla_{\beta} L(\hat{\beta}; Z) = 0$ , we have  $(X^T X + n\lambda I) \hat{\beta} = X^T Y$
- Always invertible if  $\lambda > 0$ , so we have

$$\hat{\beta}(Z) = (X^T X + n\lambda I)^{-1} X^T Y$$

# Strategy 2: Gradient Descent

- Gradient is

$$\nabla_{\beta} L(\beta; Z) = -\frac{2}{n} X^T Y + \frac{2}{n} X^T X \beta + 2\lambda \beta$$

- Same algorithm as vanilla linear regression (a.k.a. OLS)
- **Intuition:** The extra term  $\lambda \beta$  in the gradient is **weight decay** that encourages  $\beta$  to be small

# Supervised Learning



Data  $Z = \{(x_i, y_i)\}_{i=1}^n$        $\hat{\beta}(Z) = \arg \min_{\beta} L(\beta; Z)$   
 $L$  encodes  $y_i \approx f_{\beta}(x_i)$

Model  $f_{\hat{\beta}(Z)}$

# Regression



Data  $Z = \{(x_i, y_i)\}_{i=1}^n$

$$\hat{\beta}(Z) = \arg \min_{\beta} L(\beta; Z)$$

$L$  encodes  $y_i \approx f_{\beta}(x_i)$

Model  $f_{\hat{\beta}(Z)}$

Label is a **real value**  $y_i \in \mathbb{R}$

# Classification



Data  $Z = \{(x_i, y_i)\}_{i=1}^n$

$\hat{\beta}(Z) = \arg \min_{\beta} L(\beta; Z)$   
 $L$  encodes  $y_i \approx f_{\beta}(x_i)$

Model  $f_{\hat{\beta}(Z)}$

Label is a **discrete value**  $y_i \in \mathcal{Y} = \{1, \dots, k\}$

# (Binary) Classification

- **Input:** Dataset  $Z = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$
- **Output:** Model  $y_i \approx f_{\beta}(x_i)$

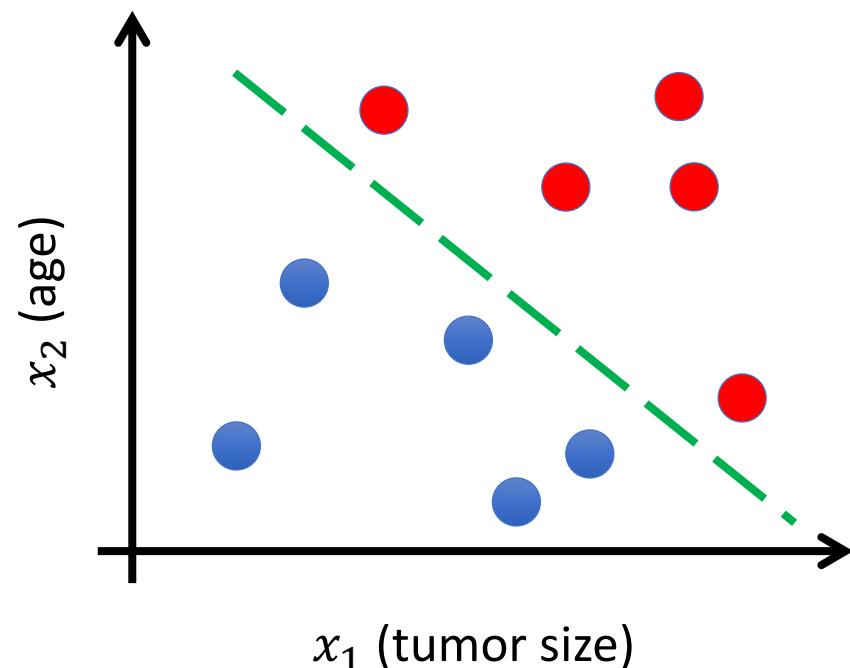


Image: <https://eyecancer.com/uncategorized/choroidal-metastasis-test/>

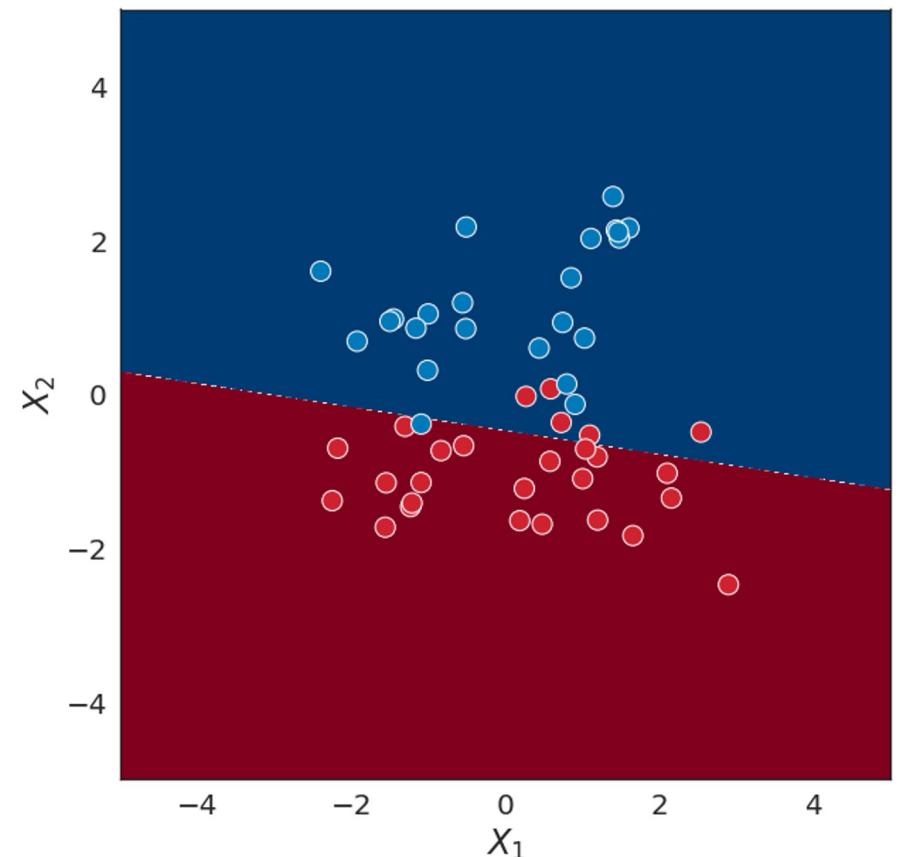
Example: Malignant vs. Benign Ocular Tumor

# Linear Functions for (Binary) Classification

- **Input:** Dataset  $Z = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$

- **Classification:**

- Labels  $y_i \in \{0, 1\}$
- Predict  $y_i \approx 1(\beta^T x_i \geq 0)$
- $1(C)$  equals 1 if  $C$  is true and 0 if  $C$  is false
- How to learn  $\beta$ ? Need a loss function!

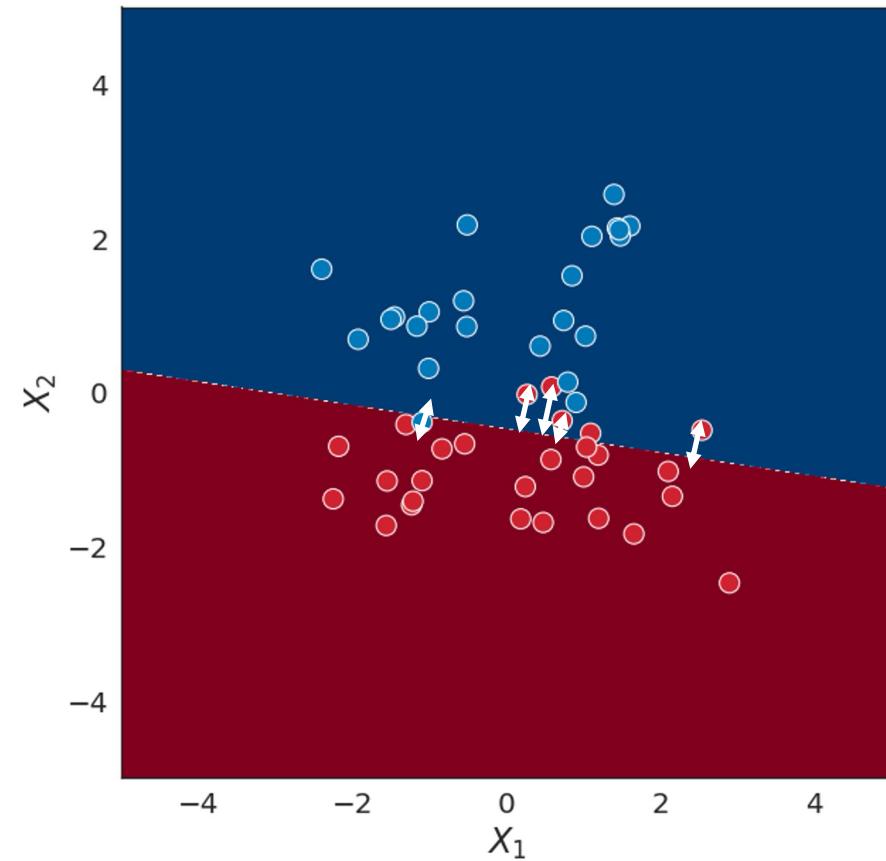


# Loss Functions for Linear Classifiers

- **Distance:**

$$L(\beta; Z) = \frac{1}{n} \sum_{i=1}^n \text{dist}(x_i, f_\beta) \cdot 1(f_\beta(x_i) \neq y_i)$$

- If  $L(\beta; Z) = 0$ , then 100% accuracy
- Variant of this loss results in SVM
- But, we will consider a more general strategy



$$L(\beta; Z) = 1.2$$

# Maximum Likelihood Estimation

- Our first **probabilistic** viewpoint on learning (from statistics)
- Given  $x_i$ , **suppose**  $y_i$  is drawn i.i.d. from distribution  $p_{Y|X}(Y = y | x; \beta)$  with parameters  $\beta$  (or density, if  $y_i$  is continuous):

$$y_i \sim p_{Y|X}(\cdot | x_i; \beta)$$

$Y$  is random variable,  
not vector

- Typically write  $p_\beta(Y = y | x)$  or just  $p_\beta(y | x)$ 
  - Called a **model** (and  $\{p_\beta\}_\beta$  is the **model family**)
  - Will show up convert  $p_\beta$  to  $f_\beta$  later

# Maximum Likelihood Estimation

- Compare to loss function minimization:
  - Before:  $y_i \approx f_\beta(x_i)$
  - Now:  $y_i \sim p_\beta(\cdot | x_i; \beta)$
- Intuition the difference:
  - $f_\beta(x_i)$  just provides a point that  $y_i$  should be close to
  - $p_\beta(\cdot | x_i; \beta)$  provides a score for each possible  $y_i$
- Maximum likelihood estimation combines the **loss function** and **model family** design decisions

# Maximum Likelihood Estimation

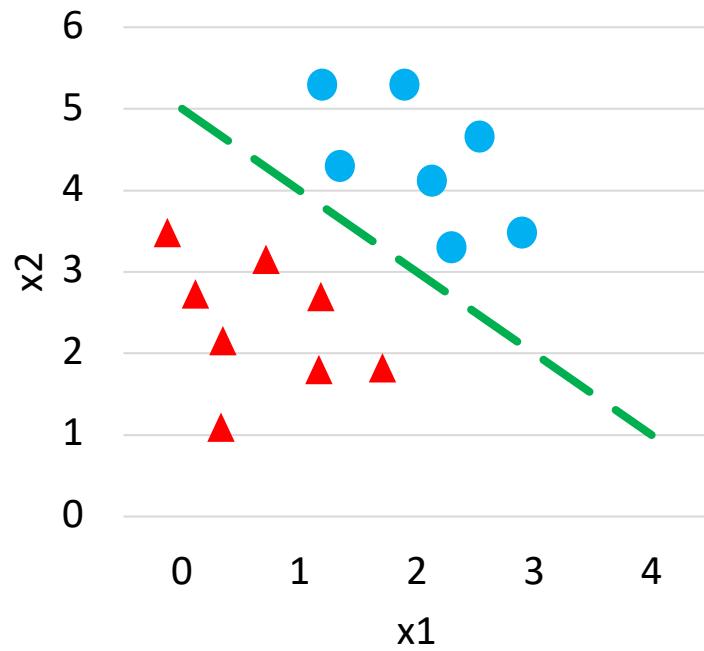
- **Likelihood:** Given model  $p_{\beta}$ , the probability of dataset  $Z$  (replaces loss function in loss minimization view):

$$L(\beta; Z) = p_{\beta}(Y \mid X) = \prod_{i=1}^n p_{\beta}(y_i \mid x_i)$$

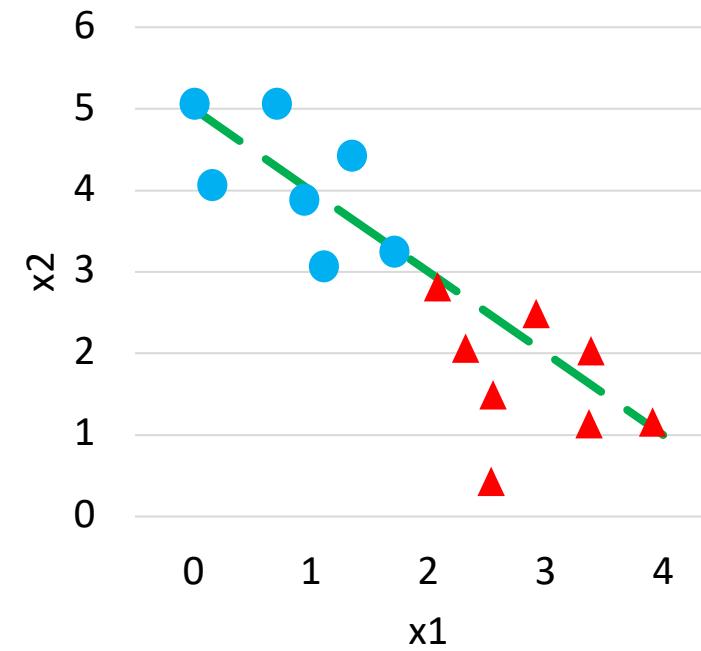
- **Negative Log-likelihood (NLL):** Computationally better behaved form:

$$\ell(\beta; Z) = -\log L(\beta; Z) = -\sum_{i=1}^n \log p_{\beta}(y_i \mid x_i)$$

# Intuition on the Likelihood



High likelihood  
(Low NLL)



Low likelihood  
(High NLL)

# Example: Linear Regression

- Assume that the conditional density is

$$p_{\beta}(y_i | x_i) = N(y_i; \beta^T x_i, 1) = \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{(\beta^T x_i - y_i)^2}{2}}$$

- $N(y; \mu, \sigma^2)$  is the density of the normal (a.k.a. Gaussian) distribution with mean  $\mu$  and variance  $\sigma^2$

# Example: Linear Regression

- Then, the likelihood is

$$L(\beta; Z) = \prod_{i=1}^n p_\beta(y_i | x_i) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{(\beta^\top x_i - y_i)^2}{2}}$$

- The NLL is

$$\ell(\beta; Z) = - \sum_{i=1}^n \log p_\beta(y_i | x_i) = \underbrace{\frac{n \log(2\pi)}{2}}_{\text{constant}} + \underbrace{\sum_{i=1}^n (\beta^\top x_i - y_i)^2}_{\text{MSE!}}$$

# Example: Linear Regression

- Loss minimization for maximum likelihood estimation:

$$\hat{\beta}(Z) = \arg \min_{\beta} \ell(\beta; Z)$$

- **Note:** Called maximum likelihood estimation since maximizing the likelihood equivalent to minimizing the NLL

# Example: Linear Regression

- What about the model family?

$$\begin{aligned} f_{\beta}(\mathbf{x}) &= \arg \max_{\mathbf{y}} p_{\beta}(\mathbf{y} \mid \mathbf{x}) \\ &= \arg \max_{\mathbf{y}} \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{\|\beta^T \mathbf{x} - \mathbf{y}\|_2^2}{2}} \\ &= \beta^T \mathbf{x} \end{aligned}$$

- Recovers linear functions!

# Maximum Likelihood View of ML

- **Two design decisions**
  - **Likelihood:** Probability  $p_{\beta}(y | x)$  of data  $(x, y)$  given parameters  $\beta$
  - **Optimizer:** How do we optimize the NLL? (E.g., gradient descent)
- **Corresponding Loss Minimization View:**
  - **Model family:** Most likely label  $f_{\beta}(x) = \arg \max_y p_{\beta}(y | x)$
  - **Loss function:** Negative log likelihood (NLL)  $\ell(\beta; Z) = -\sum_{i=1}^n \log p_{\beta}(y_i | x_i)$
- Very powerful framework for designing cutting edge ML algorithms
  - Write down the “right” likelihood, form tractable approximation if needed
  - Especially useful for thinking about non-i.i.d. data

# What about classification?

Compare to linear regression:

$$p_{\beta}(y | x_i) \propto e^{-\frac{(\beta^T x_i - y)^2}{2}}$$

- Consider the following choice:

$$p_{\beta}(Y = 0 | x_i) \propto e^{-\frac{\beta^T x_i}{2}} \text{ and } p_{\beta}(Y = 1 | x_i) \propto e^{\frac{\beta^T x_i}{2}}$$

- Then, we have

$$p_{\beta}(Y = 1 | x_i) = \frac{e^{\frac{\beta^T x_i}{2}}}{e^{\frac{\beta^T x_i}{2}} + e^{-\frac{\beta^T x_i}{2}}} = \frac{1}{1 + e^{-\beta^T x_i}}$$

Sigmoid function

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

# What about classification?

Compare to linear regression:

$$p_{\beta}(y | x_i) \propto e^{-\frac{(\beta^T x_i - y)^2}{2}}$$

- Consider the following choice:

$$p_{\beta}(Y = 0 | x_i) \propto e^{-\frac{\beta^T x_i}{2}} \text{ and } p_{\beta}(Y = 1 | x_i) \propto e^{\frac{\beta^T x_i}{2}}$$

- Then, we have

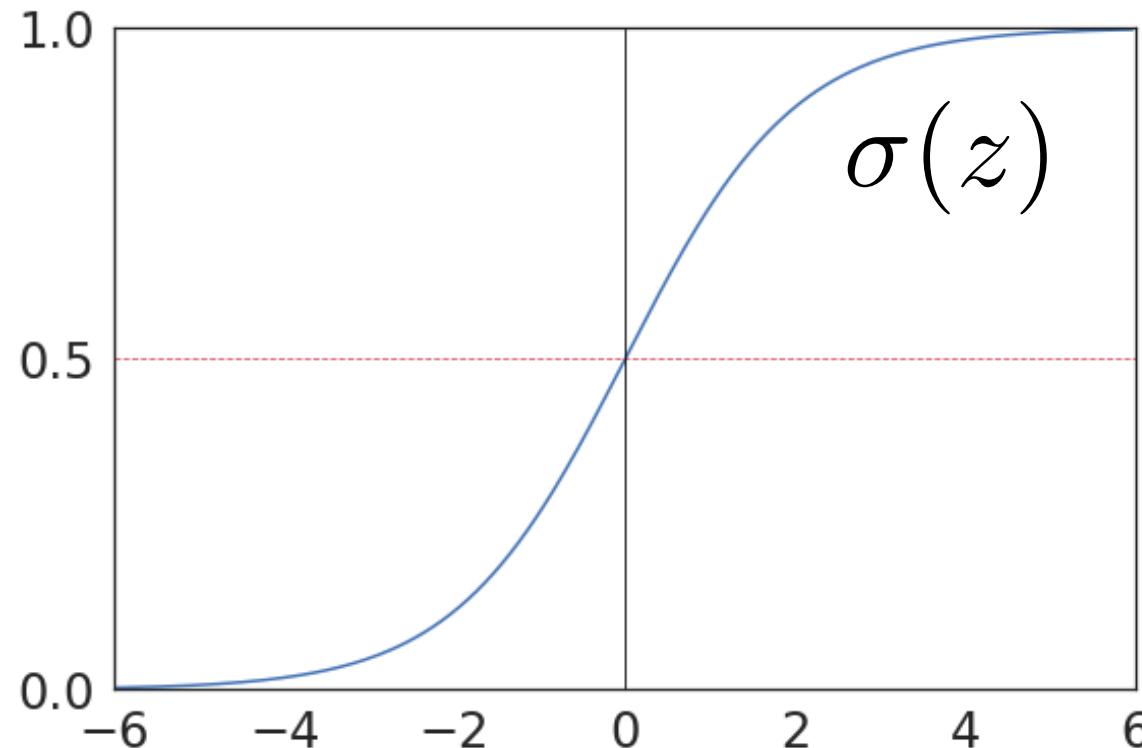
$$p_{\beta}(Y = 1 | x_i) = \frac{e^{\frac{\beta^T x_i}{2}}}{e^{\frac{\beta^T x_i}{2}} + e^{-\frac{\beta^T x_i}{2}}} = \sigma(\beta^T x_i)$$

Sigmoid function

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

- Furthermore,  $p_{\beta}(Y = 0 | x_i) = 1 - \sigma(\beta^T x_i)$

# Logistic/Sigmoid Function



$$p_{\beta}(Y = 1 \mid \textcolor{blue}{x}_i) = \sigma(\boldsymbol{\beta}^T \textcolor{blue}{x}_i)$$

# Logistic Regression Model Family

$$f_{\beta}(x) = \arg \max_{y} p_{\beta}(y | x)$$

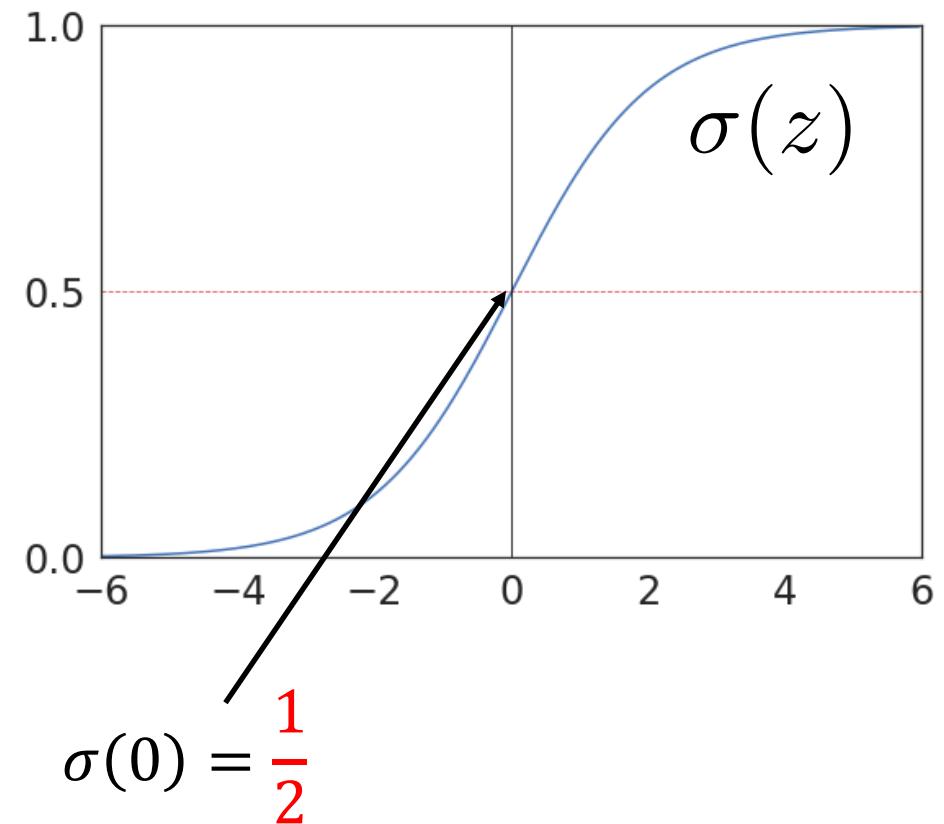
$$= \arg \max_{y} \begin{cases} \sigma(\beta^T x) & \text{if } y = 1 \\ 1 - \sigma(\beta^T x) & \text{if } y = 0 \end{cases}$$

$$= \begin{cases} 1 & \text{if } \sigma(\beta^T x) \geq \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

$$= \begin{cases} 1 & \text{if } \beta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

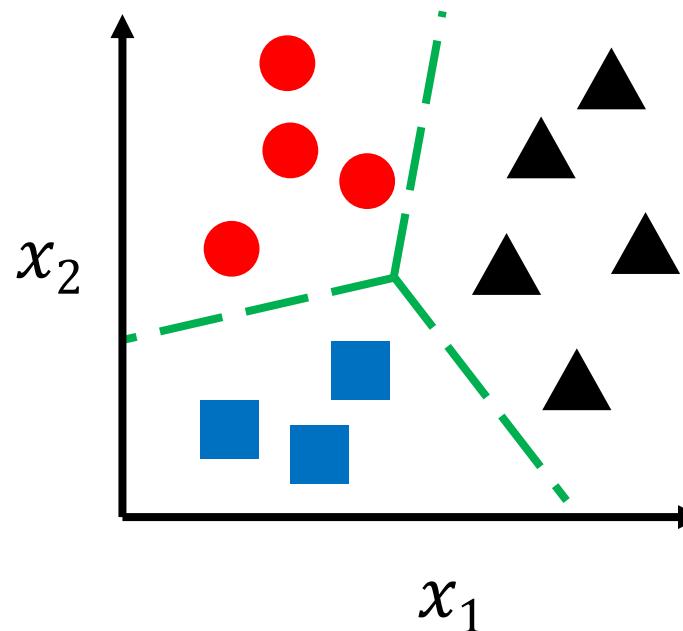
$$= 1(\beta^T x \geq 0)$$

- Recovers linear classifiers!



# Multi-Class Classification

- What about more than two classes?
  - **Disease diagnosis:** healthy, cold, flu, pneumonia
  - **Object classification:** desk, chair, monitor, bookcase
  - In general, consider a finite space of labels  $\mathcal{Y}$



# Multi-Class Logistic Regression

- **Strategy:** Include separate  $\beta_y$  for each label  $y \in \mathcal{Y} = \{1, \dots, k\}$
- Let  $p_\beta(y | x) \propto e^{\beta_y^\top x}$ , i.e.

$$p_\beta(y | x) = \frac{e^{\beta_y^\top x}}{\sum_{y' \in \mathcal{Y}} e^{\beta_{y'}^\top x}}$$

- We define softmax( $z_1, \dots, z_k$ ) =  $\begin{bmatrix} \frac{e^{z_1}}{\sum_{i=1}^k e^{z_i}} & \dots & \frac{e^{z_k}}{\sum_{i=1}^k e^{z_i}} \end{bmatrix}$
- Then,  $p_\beta(y | x) = \text{softmax}(\beta_1^\top x, \dots, \beta_k^\top x)_y$ 
  - Thus, sometimes called **softmax regression**

# Multi-Class Logistic Regression

- **Model family**

- $f_{\beta}(x) = \arg \max_y p_{\beta}(y | x) = \arg \max_y \frac{e^{\beta_y^T x}}{\sum_{y' \in \mathcal{Y}} e^{\beta_{y'}^T x}} = \arg \max_y \beta_y^T x$

- **Optimization**

- Gradient descent on NLL
- Simultaneously update all parameters  $\{\beta_y\}_{y \in \mathcal{Y}}$

# Classification Metrics

- While we minimize the NLL, we often evaluate using **accuracy**
- However, even accuracy isn't necessarily the “right” metric
  - If 99% of labels are negative (i.e.,  $y_i = 0$ ), accuracy of  $f_\beta(x)$  = 0 is 99%!
  - For instance, very few patients test positive for most diseases
  - “Imbalanced data”
- What are alternative metrics for these settings?

# Classification Metrics

- Classify test examples as follows:
  - **True positive (TP)**: Actually positive, predictive positive
  - **False negative (FN)**: Actually positive, predicted negative
  - **True negative (TN)**: Actually negative, predicted negative
  - **False positive (FP)**: Actually negative, predicted positive
- Many metrics expressed in terms of these; for example:

$$\text{accuracy} = \frac{TP + TN}{n} \quad \text{error} = 1 - \text{accuracy} = \frac{FP + FN}{n}$$

# Confusion Matrix

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN

# Confusion Matrix

		Predicted Class	
		Yes	No
Actual Class	Yes	3 TP	4 FN
	No	6 FP	37 TN

Accuracy = 0.8

# Classification Metrics

- For imbalanced metrics, we roughly want to disentangle:
  - Accuracy on “positive examples”
  - Accuracy on “negative examples”
- Different definitions are possible (and lead to different meanings)!

# Sensitivity & Specificity

- **Sensitivity:** What fraction of **actual positives** are **predicted positive**?
  - **Good sensitivity:** If you have the disease, the test correctly detects it
  - Also called **true positive rate**
- **Specificity:** What fraction of **actual negatives** are **predicted negative**?
  - **Good specificity:** If you do not have the disease, the test says so
  - Also called **true negative rate**
- Commonly used in medicine

# Sensitivity & Specificity

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN

$$\text{sensitivity} = \frac{TP}{TP + FN}$$

$$\text{specificity} = \frac{TN}{TN + FP}$$

# Sensitivity & Specificity

		Predicted Class	
		Yes	No
Actual Class	Yes	3 TP    4 FN	
	No	6 FP    37 TN	

$$\text{sensitivity} = \frac{TP}{TP + FN}$$

$$\text{specificity} = \frac{TN}{TN + FP}$$

# Sensitivity & Specificity

		Predicted Class	
		Yes	No
Actual Class	Yes	3 TP    4 FN	sensitivity = 3/7
	No	6 FP    37 TN	specificity = 37/43

# Precision & Recall

- **Recall:** What fraction of **actual positives** are **predicted positive**?
  - **Good recall:** If you have the disease, the test correctly detects it
  - Also called the **true positive rate** (and sensitivity)
- **Precision:** What fraction of **predicted positives** are **actual positives**?
  - **Good precision:** If the test says you have the disease, then you have it
  - Also called **positive predictive value**
- Used in information retrieval, NLP

# Precision & Recall

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN

$$\text{recall} = \frac{TP}{TP + FN}$$

$$\text{precision} = \frac{TP}{TP + FP}$$

# Precision & Recall

		Predicted Class	
		Yes	No
Actual Class	Yes	3 TP	4 FN
	No	6 FP	37 TN

$$\text{recall} = \frac{TP}{TP + FN}$$

$$\text{precision} = \frac{TP}{TP + FP}$$

# Precision & Recall

		Predicted Class	
		Yes	No
Actual Class	Yes	3 TP 4 FN	
	No	6 FP	37 TN

precision = 3/9

recall = 3/7