Introduction to Data Science and Machine Learning

EPF - P2023
Yassine Idrissi - yassine.idrissi@teads.com
Ismail Chafai - ismail.chafai@teads.com

EPF
ÉCOLE D'INGÉNIEUR·E·S

Summary

Overview

- « Machine Learning is the study of computer **algorithms** that **improve automatically** through **experience** » *Tom Mitchell*

Overview

- « Machine Learning is the study of computer **algorithms** that **improve automatically** through **experience** » *Tom Mitchell*

- How to **design** an algorithm to **learn automatically**, from example **data** or past **experience**, so as to **optimize** a given **criterion**

Overview

- « Machine Learning is the study of computer **algorithms** that **improve automatically** through **experience** » *Tom Mitchell*

- How to **design** an algorithm to **learn automatically**, from example **data** or past **experience**, so as to **optimize** a given **criterion**

- Machine Learning can be seen as a **mixture** of CS and Statistics

Overview

- « Machine Learning is the study of computer **algorithms** that **improve automatically** through **experience** » *Tom Mitchell*

- How to **design** an algorithm to **learn automatically**, from example **data** or past **experience**, so as to **optimize** a given **criterion**

- Machine Learning can be seen as a **mixture** of CS and Statistics

  - **Computer Science**: How to program a computer to solve problems?

Overview

- « Machine Learning is the study of computer **algorithms** that **improve automatically** through **experience** » *Tom Mitchell*

- How to **design** an algorithm to **learn automatically**, from example **data** or past **experience**, so as to **optimize** a given **criterion**

- Machine Learning can be seen as a **mixture** of CS and Statistics

  - **Computer Science**: How to program a computer to solve problems?

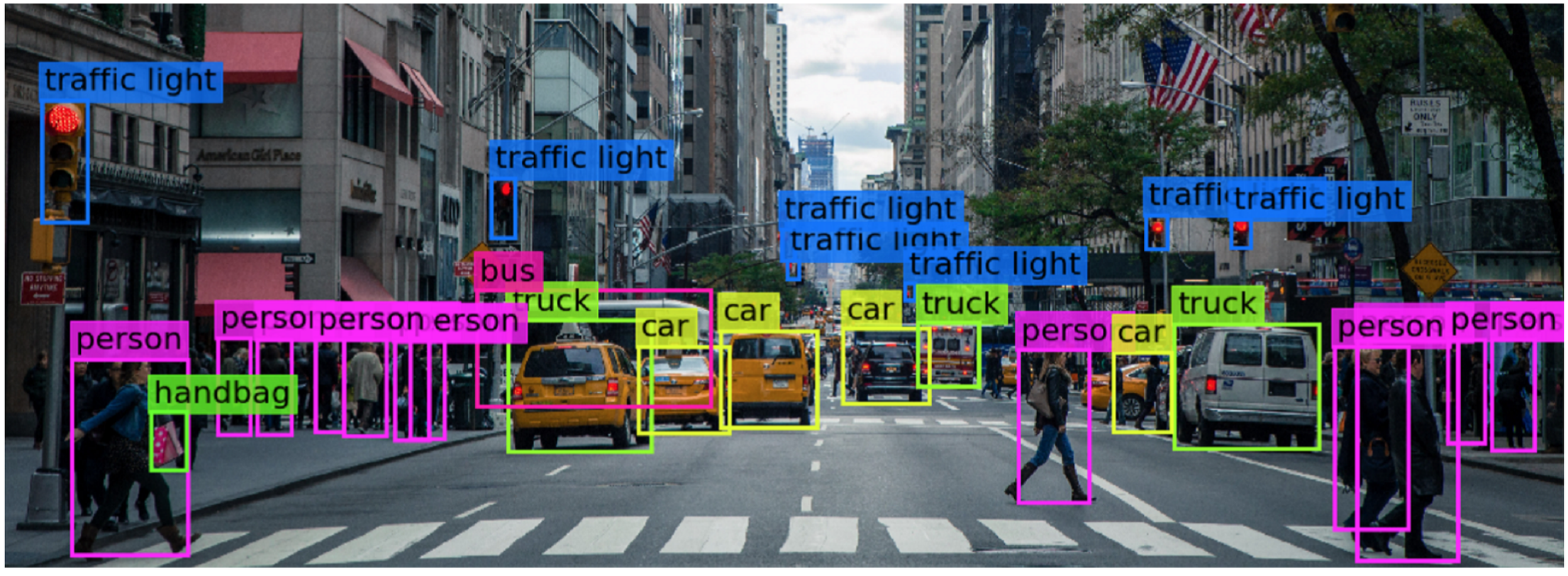
  - **Statistics**: How to extract knowledge from data?

Machine Learning is everywhere

- Computer Vision
- Robotics
- Autonomous vehicles
- Natural language processing, Speech recognition
- Search engines
- Recommender systems
- Medical diagnosis
- Etc.

Computer vision

- Object recognition / scene labeling



Robotics

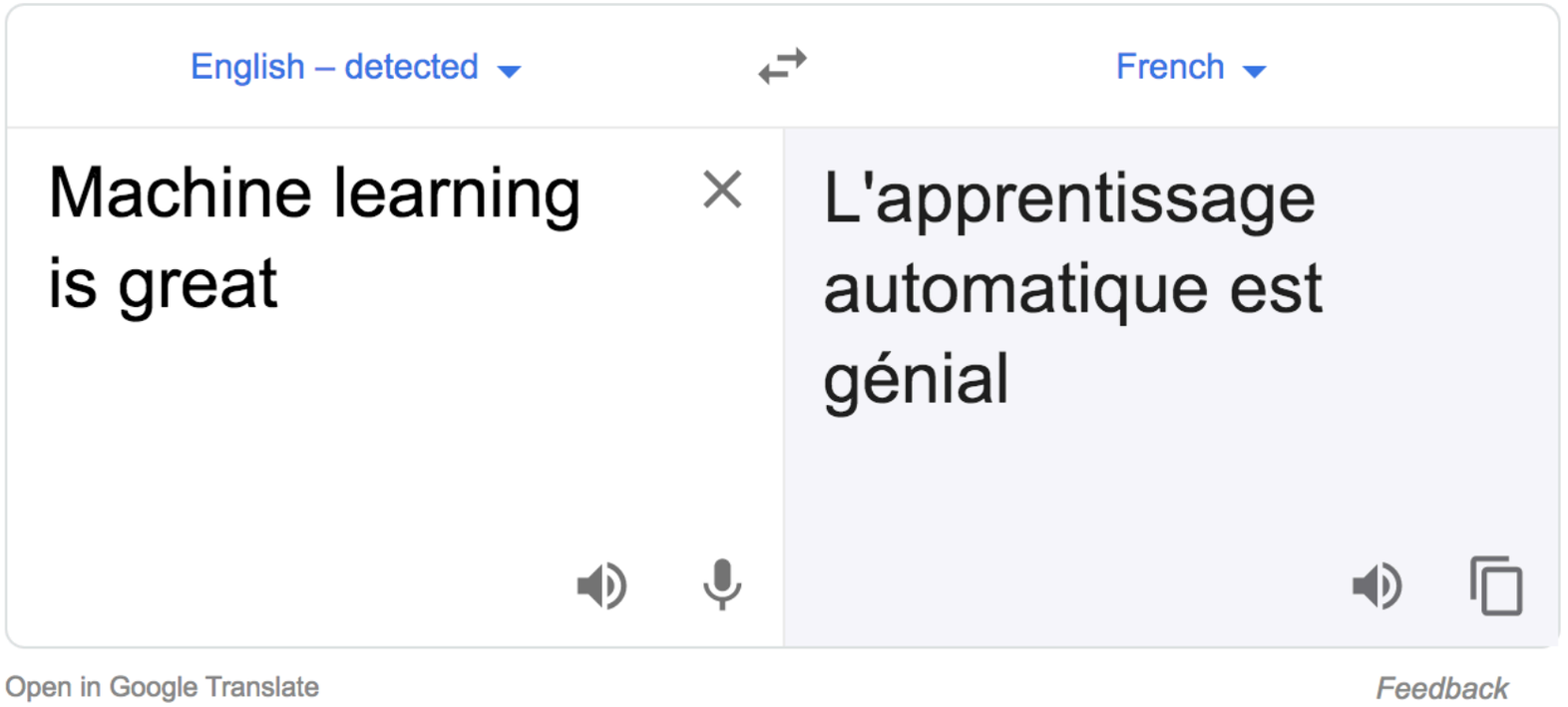
- Tasks automation

Autonomous vehicles

- Autonomous driving

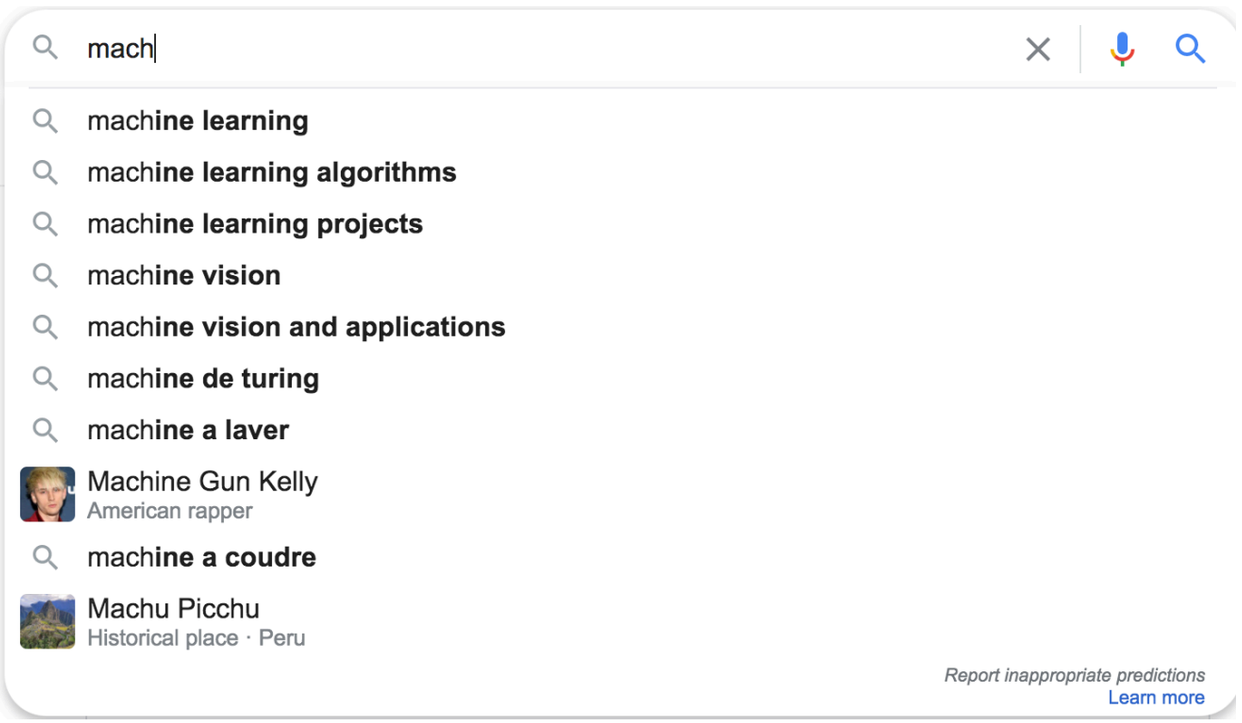
Natural language processing

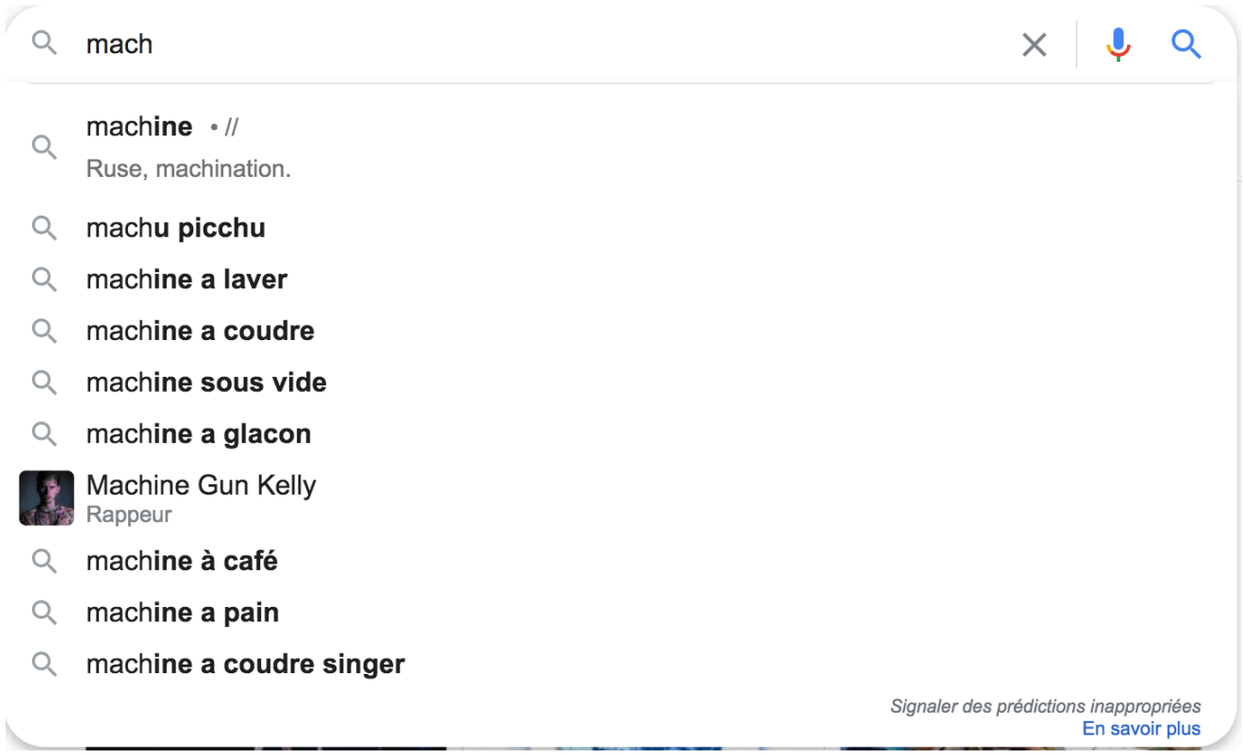
- Machine translation



Search engines

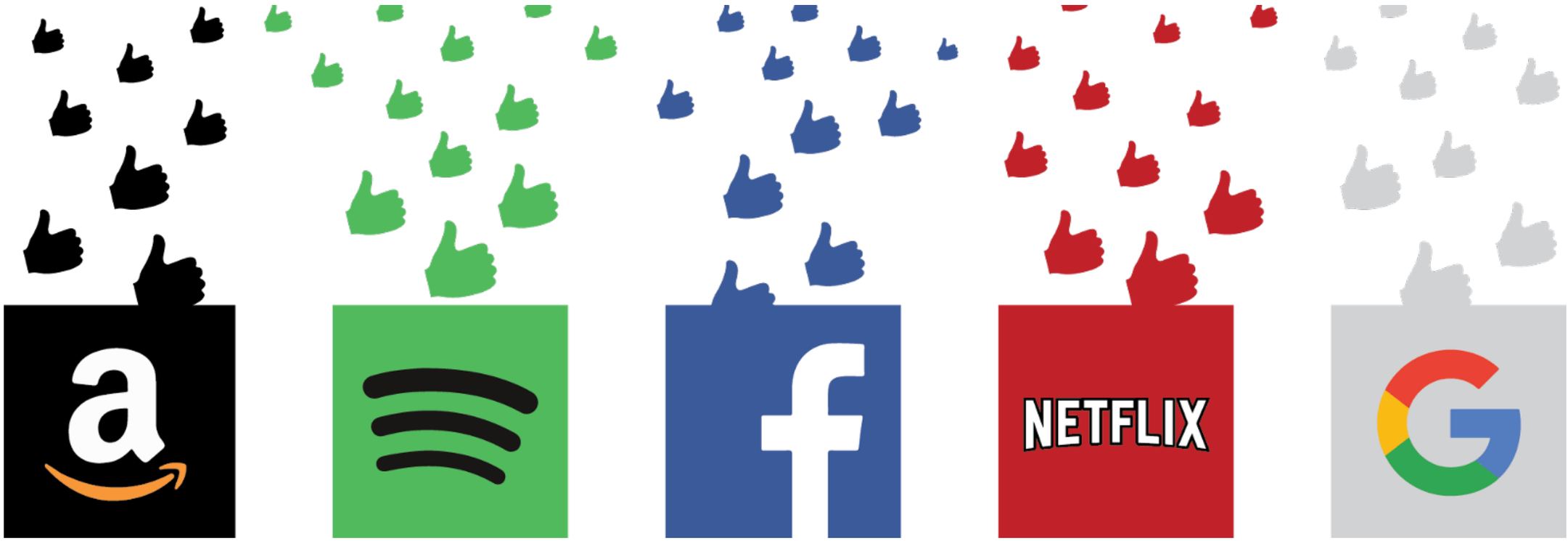
- Search engines optimization





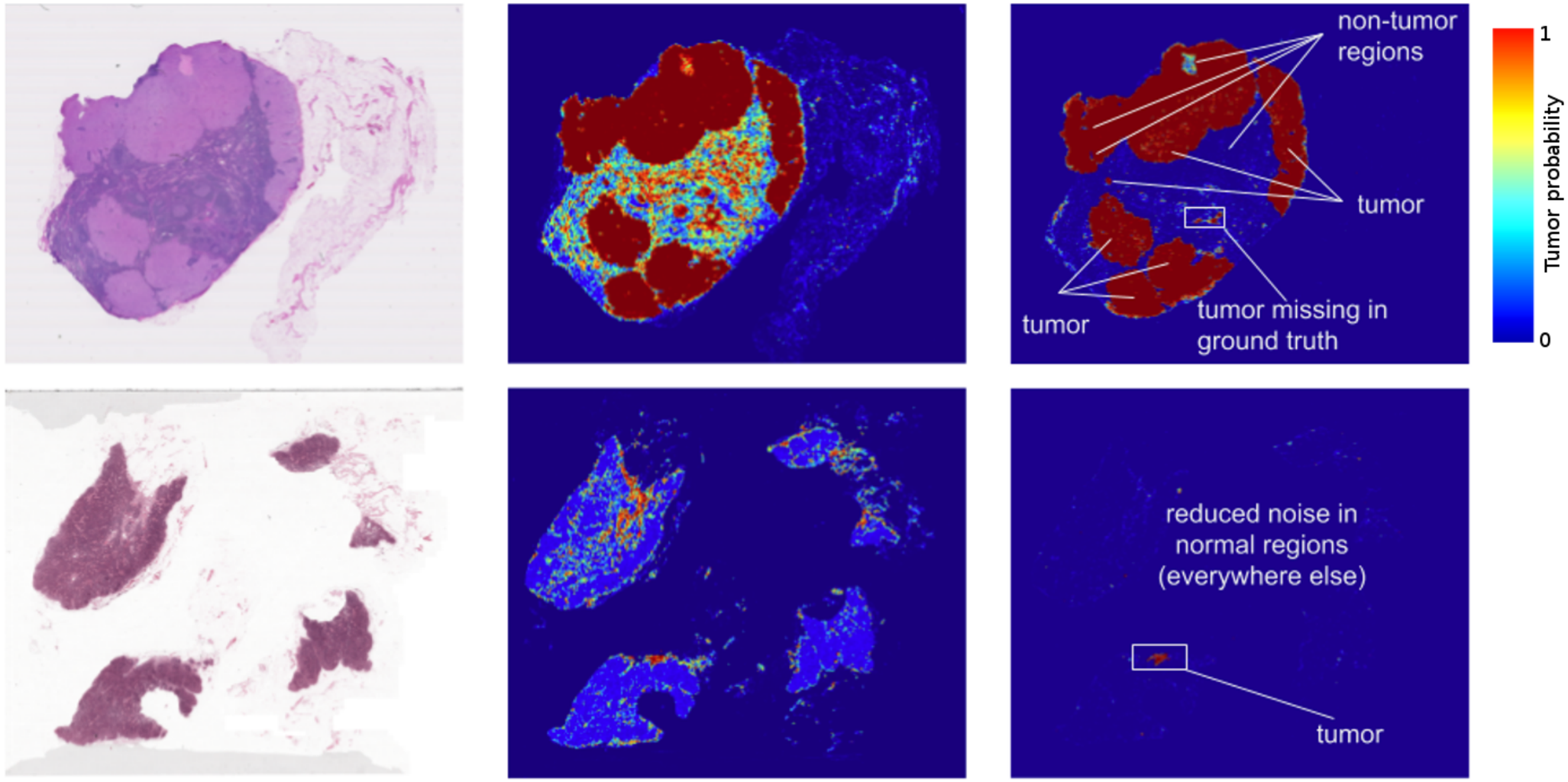
Recommender systems

- Recommender systems optimization



Medical diagnosis

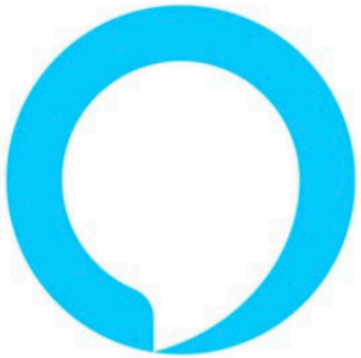
- Cancer detection



Speech recognition

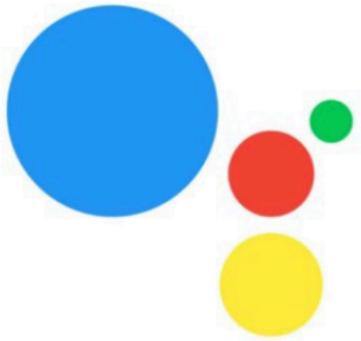
- Voice assistants

"Hey Cortana"



"Hey Alexa"

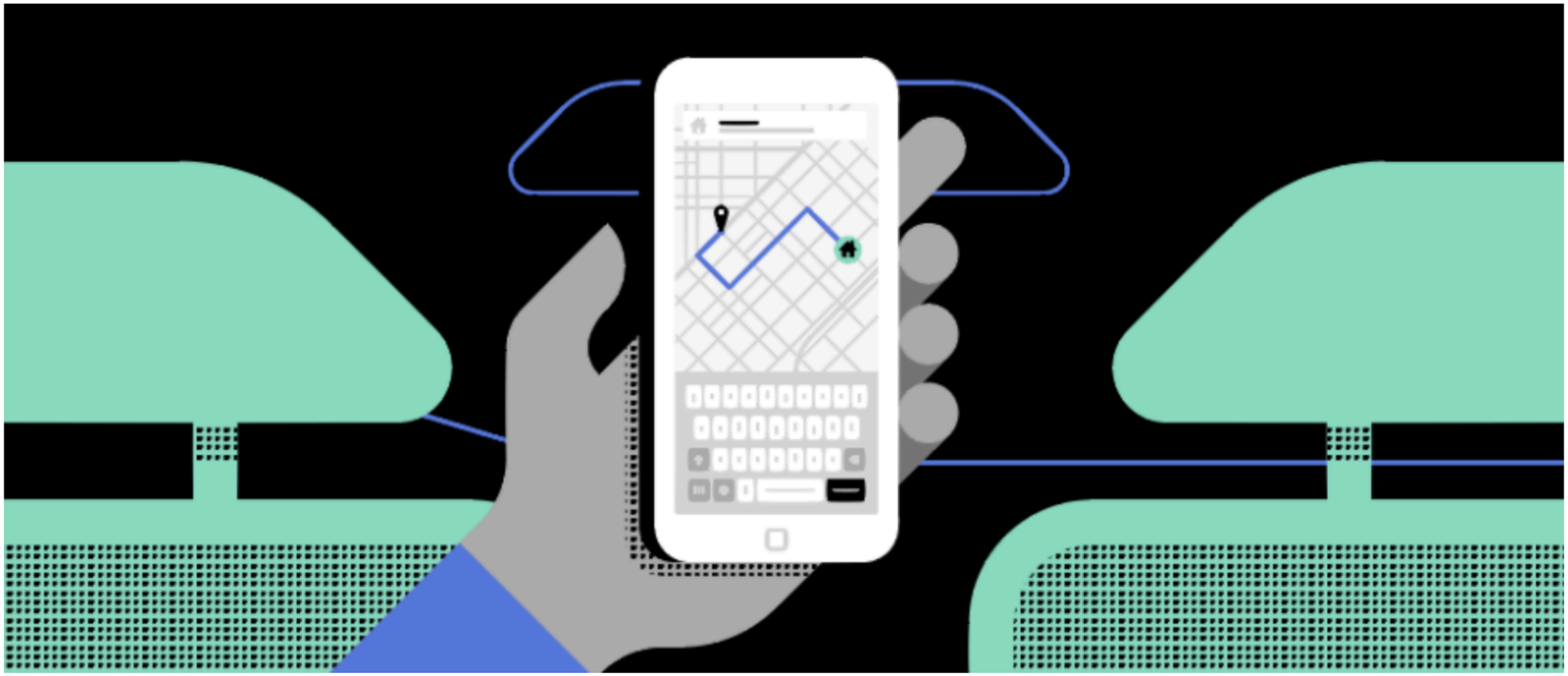
"Hey Siri"



"Hey Google"

Pricing

- Dynamic pricing in Travel



Fraud detection

- Fraudulent banking operations

Main Types of Machine Learning

EPF
ÉCOLE D'INGÉNIEUR·E·S

Main types of Machine Learning

- Supervised Learning
  - Classification
  - Regression
  - Ranking

Main types of Machine Learning

- Supervised Learning
  - Classification
  - Regression
  - Ranking
- Unsupervised Learning

Main types of Machine Learning

- Supervised Learning
  - Classification
  - Regression
  - Ranking
- Unsupervised Learning
- Reinforcement Learning

Supervised Learning

- The algorithm is given a **set of labeled training data** (x_1,y_1), (x_2,y_2), … , (x_N,y_N), i.e observations of (input,output)

Supervised Learning

- The algorithm is given a **set of labeled training data** (x_1,y_1), (x_2,y_2), … , (x_N,y_N), i.e observations of (input,output)

  - **Classification**: Y takes values in a **finite**, **unordered** set (e.g spam or not, object class, …)

Supervised Learning

- The algorithm is given a **set of labeled training data** (x_1,y_1), (x_2,y_2), … , (x_N,y_N), i.e observations of (input,output)

  - **Classification**: Y takes values in a **finite**, **unordered** set (e.g spam or not, object class, …)

  - **Regression**: Y is quantitative (price of a house, value of a stock option,…)

Supervised Learning

- The algorithm is given a **set of labeled training data** (x_1,y_1), (x_2,y_2), … , (x_N,y_N), i.e observations of (input,output)

  - **Classification**: Y takes values in a **finite**, **unordered** set (e.g spam or not, object class, …)

  - **Regression**: Y is quantitative (price of a house, value of a stock option,…)

  - **Ranking**: Y is an ordering between items

Supervised Learning

- The algorithm is given a **set of labeled training data** (x_1,y_1), (x_2,y_2), … , (x_N,y_N), i.e observations of (input,output)

  - **Classification**: Y takes values in a **finite**, **unordered** set (e.g spam or not, object class, …)

  - **Regression**: Y is quantitative (price of a house, value of a stock option,…)

  - **Ranking**: Y is an ordering between items

- The objective is to correctly **predict unseen** test cases and **understand** which inputs **affect** the outcome and how

Unsupervised Learning

- **Unsupervised learning** models learn from datasets **without labels** (output data)

Unsupervised Learning

- **Unsupervised learning** models learn from datasets **without labels** (output data)

- The goal is to find previously **undetected patterns** in the data

Unsupervised Learning

- **Unsupervised learning** models learn from datasets **without labels** (output data)

- The goal is to find previously **undetected patterns** in the data

  - Find groups of samples that exhibit similarity in some sense

Unsupervised Learning

- **Unsupervised learning** models learn from datasets **without labels** (output data)

- The goal is to find previously **undetected patterns** in the data

  - Find groups of samples that exhibit similarity in some sense

  - Find subsets of features that behave similarly

Unsupervised Learning

- **Unsupervised learning** models learn from datasets **without labels** (output data)

- The goal is to find previously **undetected patterns** in the data

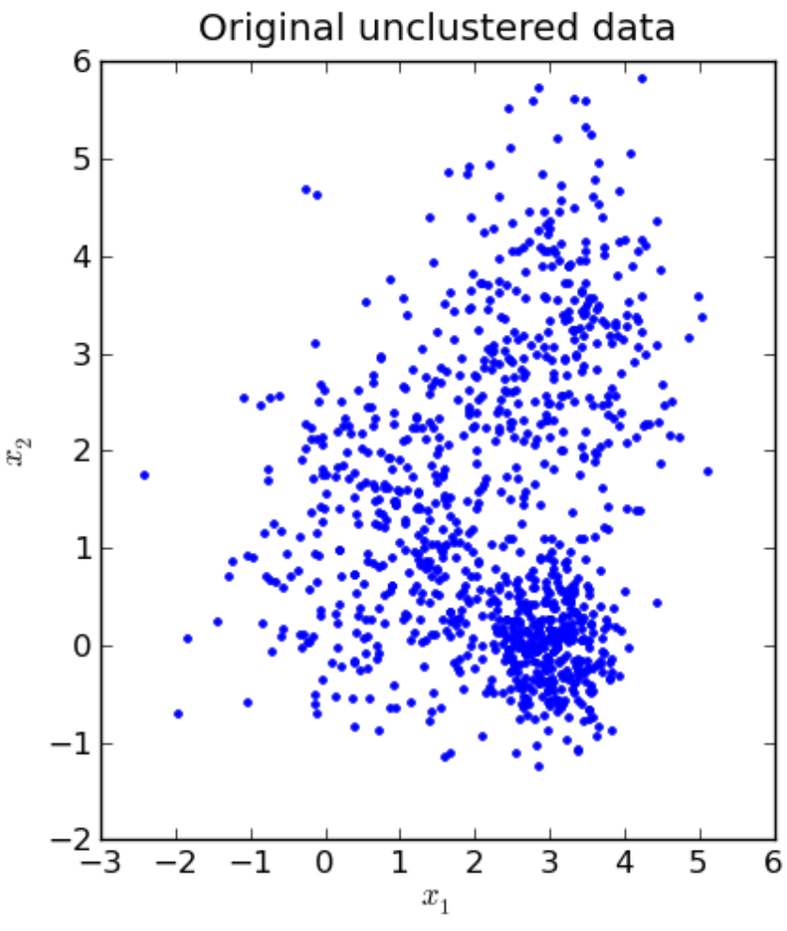
  - Find groups of samples that exhibit similarity in some sense

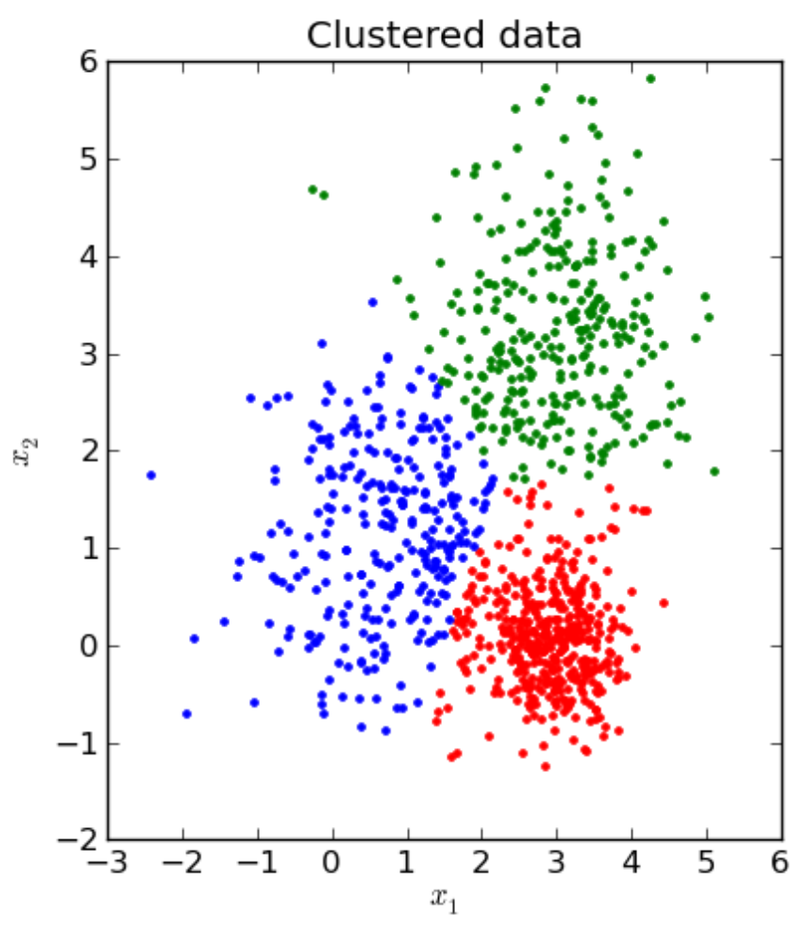
  - Find subsets of features that behave similarly

  - Find combinations of features with the most variation (e.g dimensionality reduction)

Unsupervised Learning

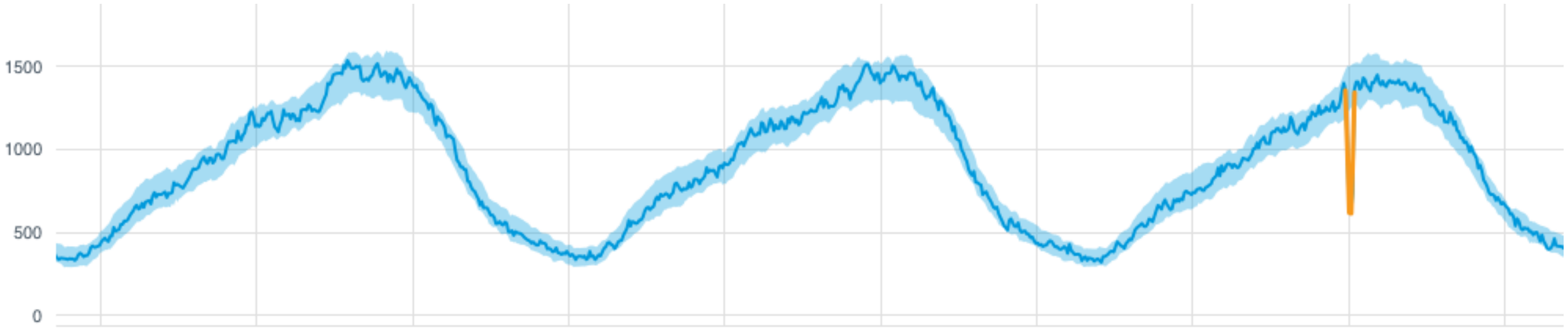
- Clustering





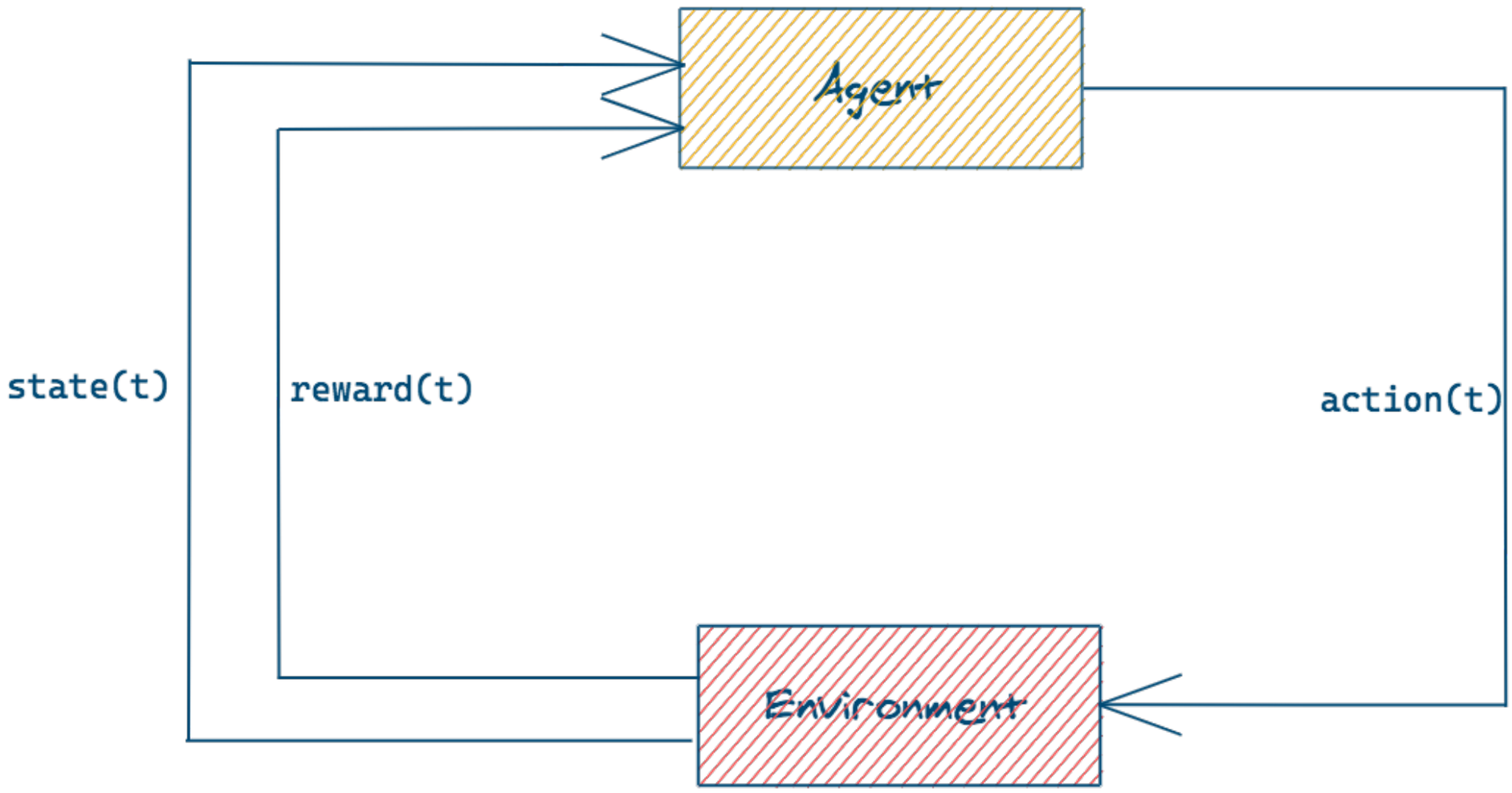
Unsupervised Learning

- Anomaly detection



Reinforcement Learning

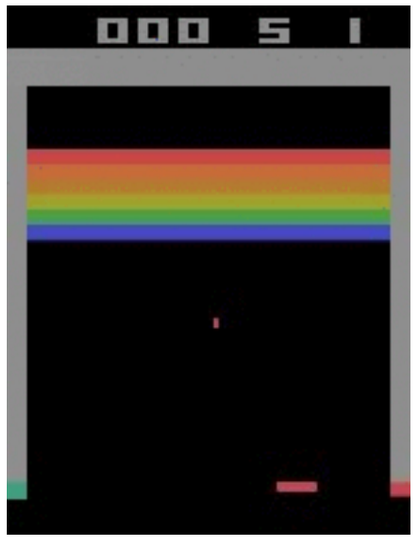
- Algorithms used to train software agents to take actions in an environment in order to maximize a given reward



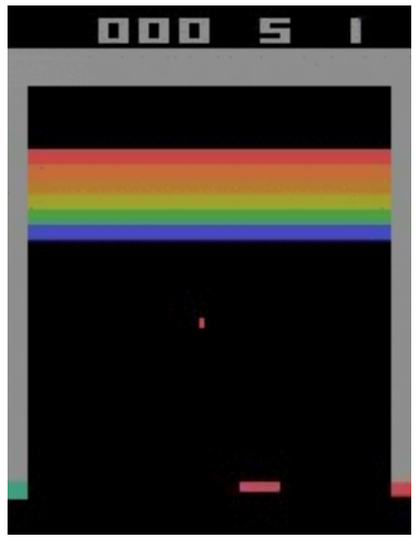
Reinforcement Learning examples

- The ATARI benchmark

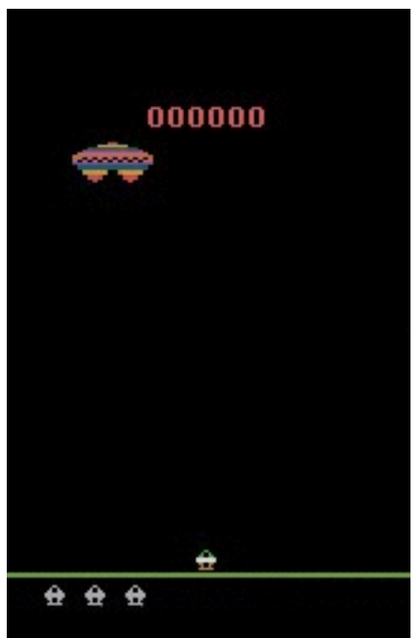
Initial attempt



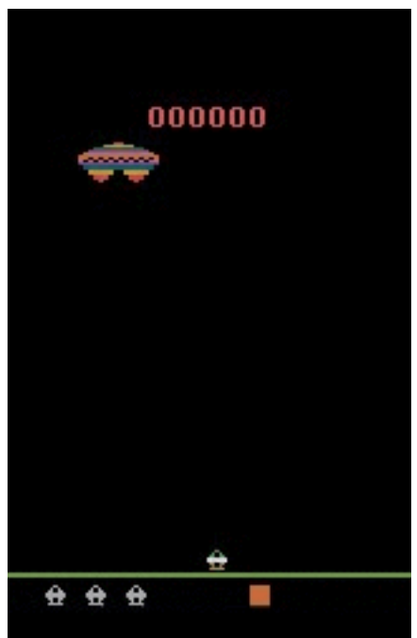
After 30 mins



Initial attempt



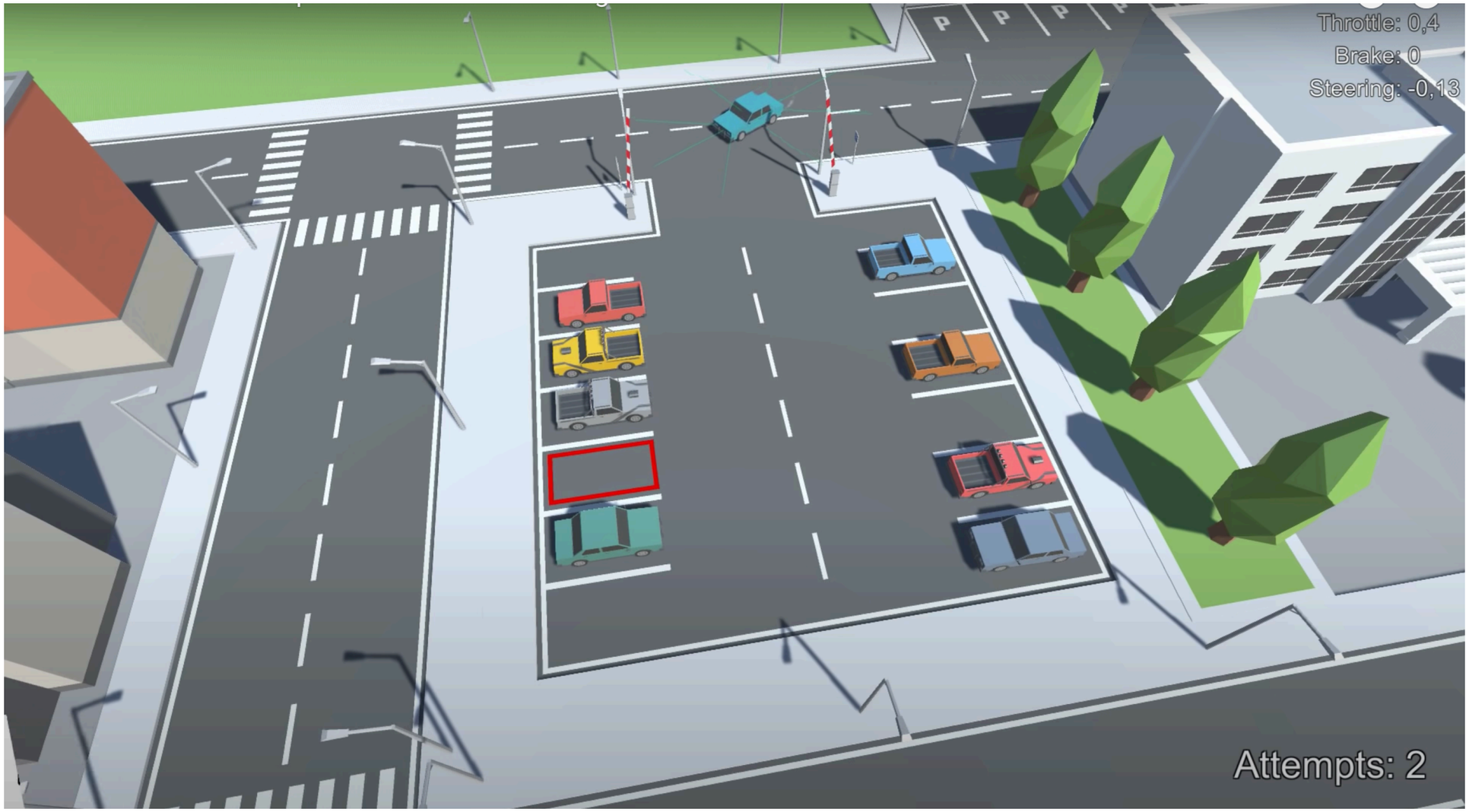
After 30 mins



EPF
ÉCOLE D'INGÉNIEUR·E·S

Reinforcement Learning examples

- Learning to park



Generalized Linear Models

ÉCOLE D'INGÉNIEUR·E·S

Supervised Learning Terminology

- The goal is to use a set of **features** (inputs, predictors, independent variables) to predict the **outputs** (responses, dependent variables)

Supervised Learning Terminology

- The goal is to use a set of **features** (inputs, predictors, independent variables) to predict the **outputs** (responses, dependent variables)

- The output can be **quantitative** (continuous) or **qualitative** (discontinuous, classes)

Supervised Learning Terminology

- The goal is to use a set of **features** (inputs, predictors, independent variables) to predict the **outputs** (responses, dependent variables

- The output can be **quantitative** (continuous) or **qualitative** (discontinuous, classes)

- The distinction in output types led to a **naming convention** for the different prediction tasks

Supervised Learning Terminology

- The goal is to use a set of **features** (inputs, predictors, independent variables) to predict the **outputs** (responses, dependent variables)

- The output can be **quantitative** (continuous) or **qualitative** (discontinuous, classes)

- The distinction in output types led to a **naming convention** for the different prediction tasks

  - **Regression**: predicting quantitative outputs

  - **Classification**: predicting qualitative outputs

Supervised Learning Terminology

- The goal is to use a set of **features** (inputs, predictors, independent variables) to predict the **outputs** (responses, dependent variables)

- The output can be **quantitative** (continuous) or **qualitative** (discontinuous, classes)

- The distinction in output types led to a **naming convention** for the different prediction tasks

  - **Regression**: predicting quantitative outputs

  - **Classification**: predicting qualitative outputs

- Both can be viewed as a **function approximation task**

First approach: Ordinary Least Squares

- Given a vector of inputs $X^T = (X_1, X_2, \ldots, X_n)$ we predict the output Y via the model

$$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{N} X_j \hat{\beta}_j$$

First approach: Ordinary Least Squares

- Given a vector of inputs $X^T = (X_1, X_2, \ldots, X_n)$ we predict the output Y via the model

$$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{N} X_j \hat{\beta}_j$$

- The term $\hat{\beta}_0$ is known as the **intercept** or **bias**, we can include it in the vector of **coefficients** $\hat{\beta}$ and add the **constant** 1 to X

$$\hat{Y} = X^T \hat{\beta}$$

First approach: Ordinary Least Squares

- Given a vector of inputs $X^T = (X_1, X_2, \ldots, X_N)$ we predict the output Y via the model

$$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{N} X_j \hat{\beta}_j$$

- The term $\hat{\beta}_0$ is known as the **intercept** or **bias**, we can include it in the vector of **coefficients** $\hat{\beta}$ and add the **constant** 1 to X

$$\hat{Y} = X^T \hat{\beta}$$

- **Least Squares** approach: pick $\hat{\beta}$ so as the **minimize** the residual sum of squares

$$RSS(\beta) = \sum_{i=1}^{N} (y_i - x_i^T \beta)^2 = (y - X\beta)^T (y - X\beta)$$

First approach: Ordinary Least Squares

- Least square is a **cost function**, we want to **minimize** it

First approach: Ordinary Least Squares

- Least square is a **cost function**, we want to **minimize** it
- RSS is a **quadratic function** in β , let's differentiate it w.r.t β :

$$\frac{\partial RSS}{\partial \beta} = -2X^T(y - X\beta) \qquad \frac{\partial^2 RSS}{\partial \beta \partial \beta^T} = 2X^T X$$

First approach: Ordinary Least Squares

- Least square is a **cost function**, we want to **minimize** it
- RSS is a **quadratic function** in β , let's differentiate it w.r.t β :

$$\frac{\partial RSS}{\partial \beta} = -2X^T(y - X\beta) \qquad \frac{\partial^2 RSS}{\partial \beta \partial \beta^T} = 2X^T X$$

- We set the first derivative to 0 to obtain the **unique solution**, (assuming X is full-rank)

$$X^T(y - X\beta) = 0 \quad \rightarrow \quad \hat{\beta} = (X^T X)^{-1} X^T y$$

First approach: Ordinary Least Squares

- Least square is a **cost function**, we want to **minimize** it
- RSS is a **quadratic function** in β , let's differentiate it w.r.t β :

$$\frac{\partial RSS}{\partial \beta} = -2X^T(y - X\beta) \qquad \frac{\partial^2 RSS}{\partial \beta \partial \beta^T} = 2X^T X$$

- We set the first derivative to 0 to obtain the **unique solution**, (assuming X is full-rank)

$$X^T(y - X\beta) = 0 \quad \rightarrow \quad \hat{\beta} = (X^T X)^{-1} X^T y$$

- This is the **analytical** solution to the minimum least squares

First approach: Ordinary Least Squares

- **OLS** works great for **small** datasets, but

First approach: Ordinary Least Squares

- **OLS** works great for **small** datasets, but
- It's definitely **not scalable**

First approach: Ordinary Least Squares

- **OLS** works great for **small** datasets, but
- It's definitely **not scalable**
- Computing $(X^T X)$ can be **very expensive** for big datasets

First approach: Ordinary Least Squares

- **OLS** works great for **small** datasets, but
- It's definitely **not scalable**
- Computing $(X^T X)$ can be **very expensive** for big datasets
- Matrix inversion to get $(X^T X)^{-1}$ is a **very expensive** and **numerically unstable** operation (e.g: in case of collinearity)

First approach: Ordinary Least Squares

- **OLS** works great for **small** datasets, but
- It's definitely **not scalable**
  - Computing $(X^T X)$ can be **very expensive** for big datasets
  - Matrix inversion to get $(X^T X)^{-1}$ is a **very expensive** and **numerically unstable** operation (e.g: in case of collinearity)
- We need to find a **numerical alternative** solution

Gradient Descent

- Remember: The **goal** is to find the **set of parameters** β to **minimize** a given **cost function** (numerical optimization task)

Gradient Descent

- Remember: The **goal** is to find the **set of parameters** β to **minimize** a given **cost function** (numerical optimization task)

- Training a ML model **often** boils down to an **optimization operation**

Gradient Descent

- Remember: The **goal** is to find the **set of parameters** β to **minimize** a given **cost function** (numerical optimization task)
- Training a ML model **often** boils down to an **optimization operation**
- One of the most common optimization algorithm in ML is called **Gradient Descent**

Gradient Descent

- Remember: The **goal** is to find the **set of parameters** β to **minimize** a given **cost function** (numerical optimization task)

- Training a ML model **often** boils down to an **optimization operation**

- One of the most common optimization algorithm in ML is called **Gradient Descent**

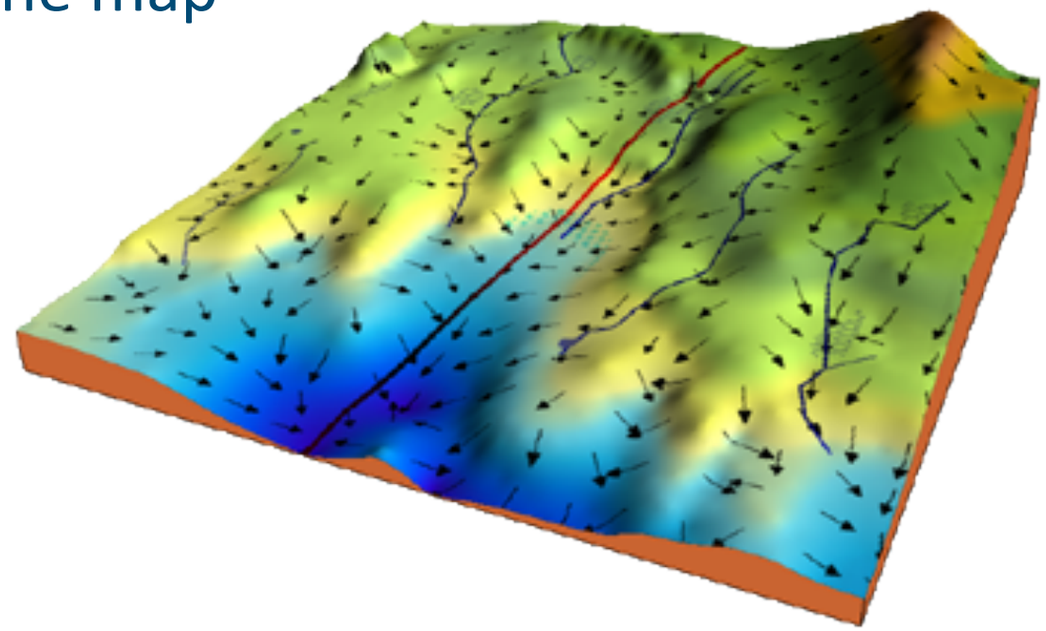
- It is a first-order **iterative optimization algorithm** for finding a local **minimum** of a **differentiable** function

Gradient Descent

- Remember: The **goal** is to find the **set of parameters** β to **minimize** a given **cost function** (numerical optimization task)
- Training a ML model **often** boils down to an **optimization operation**
- One of the most common optimization algorithm in ML is called **Gradient Descent**
- It is a first-order **iterative optimization algorithm** for finding a local **minimum** of a **differentiable** function
- At each iteration, we take a step proportional to the negative of the gradient of the function

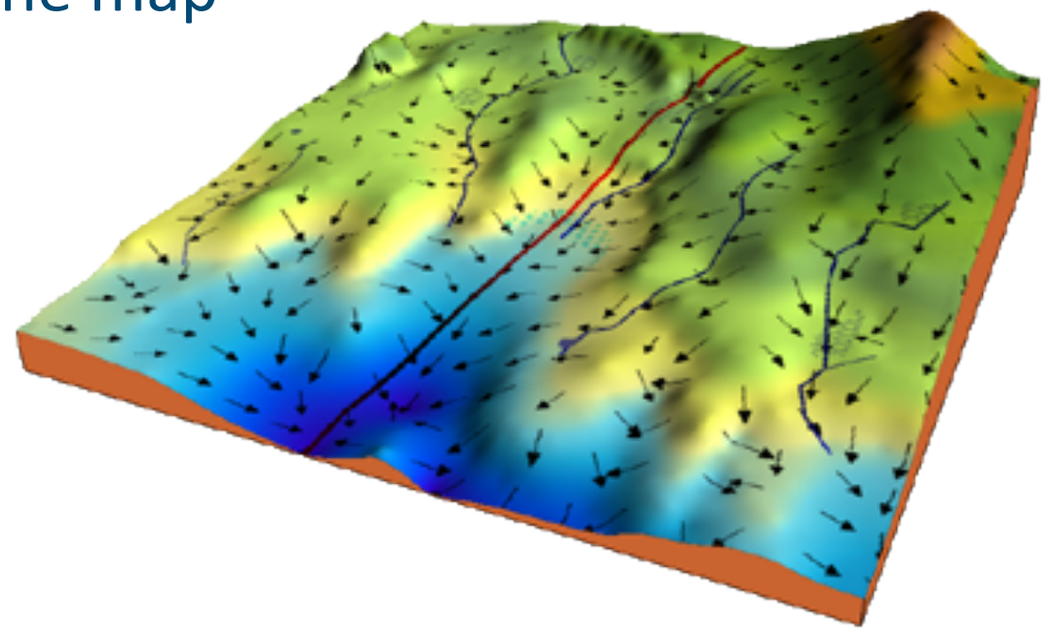
Gradient Descent analogy

- Imagine you're on the **top** of a mountain and you want to reach the **lowest** point of the map



Gradient Descent analogy

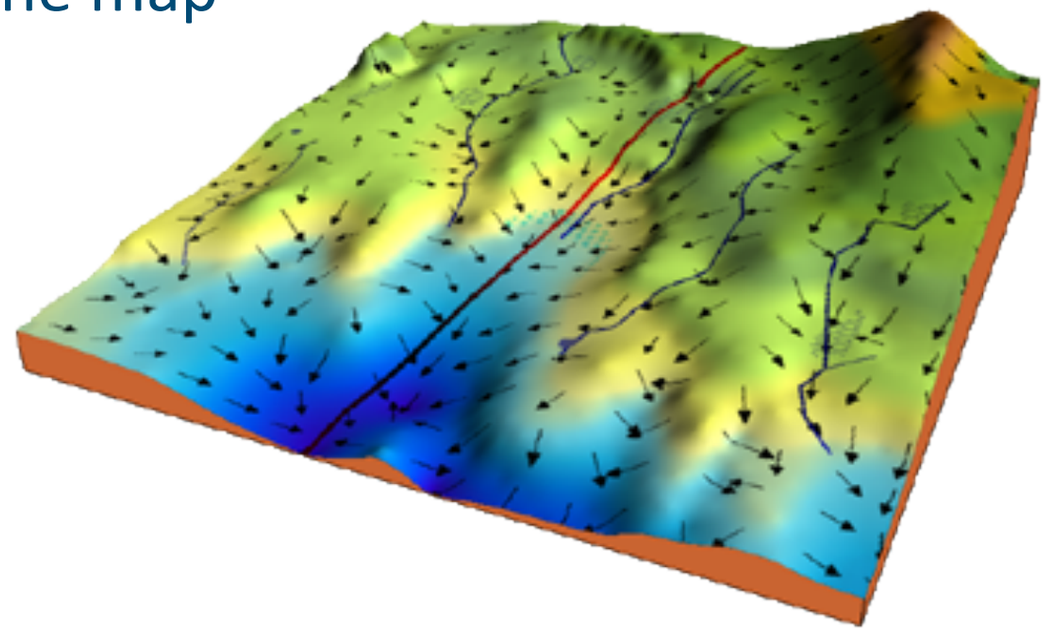
- Imagine you're on the **top** of a mountain and you want to reach the **lowest** point of the map



- The arrows represent the **direction of the steepest descent** (negative gradient)

Gradient Descent analogy

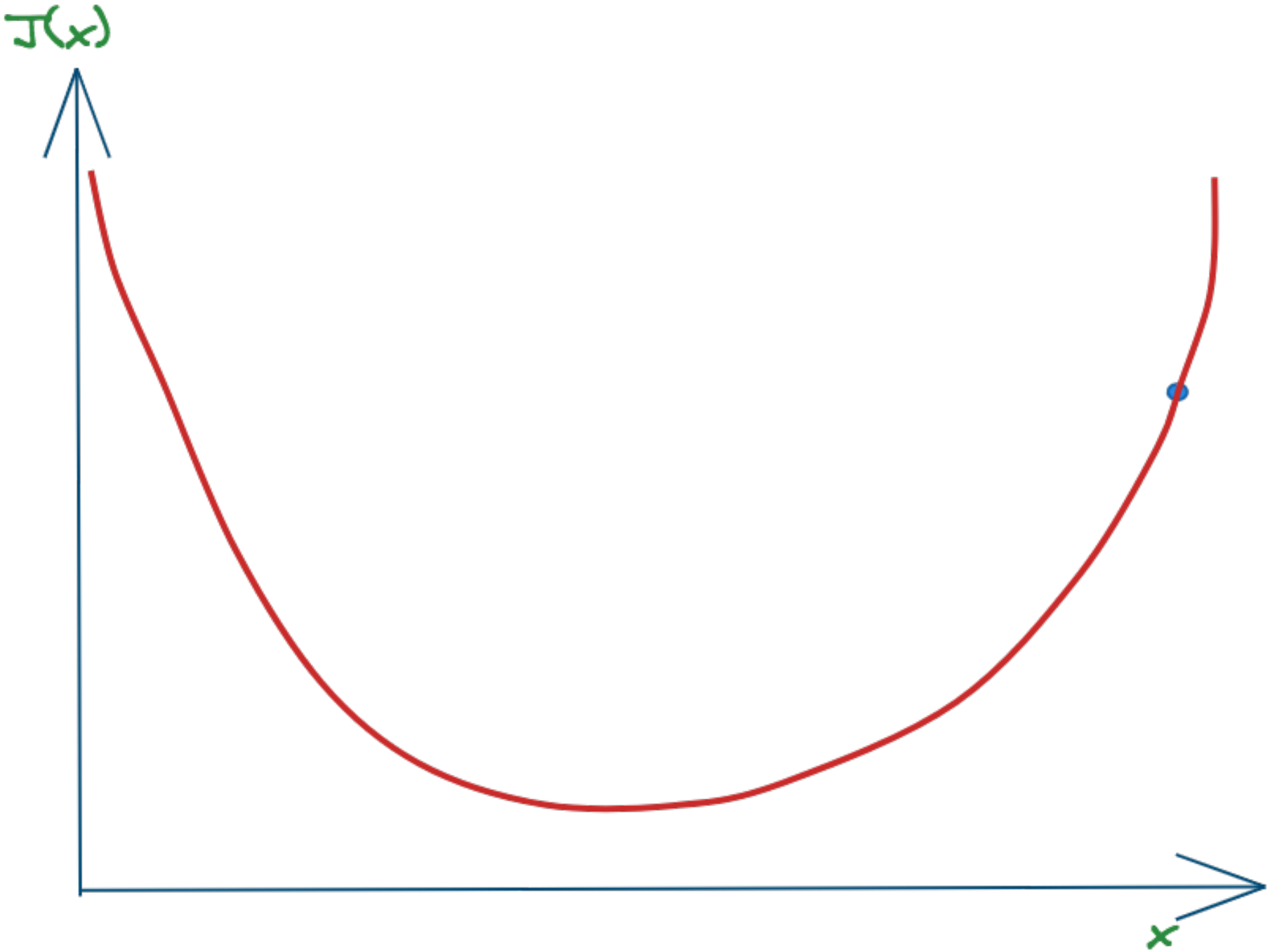
- Imagine you're on the **top** of a mountain and you want to reach the **lowest** point of the map



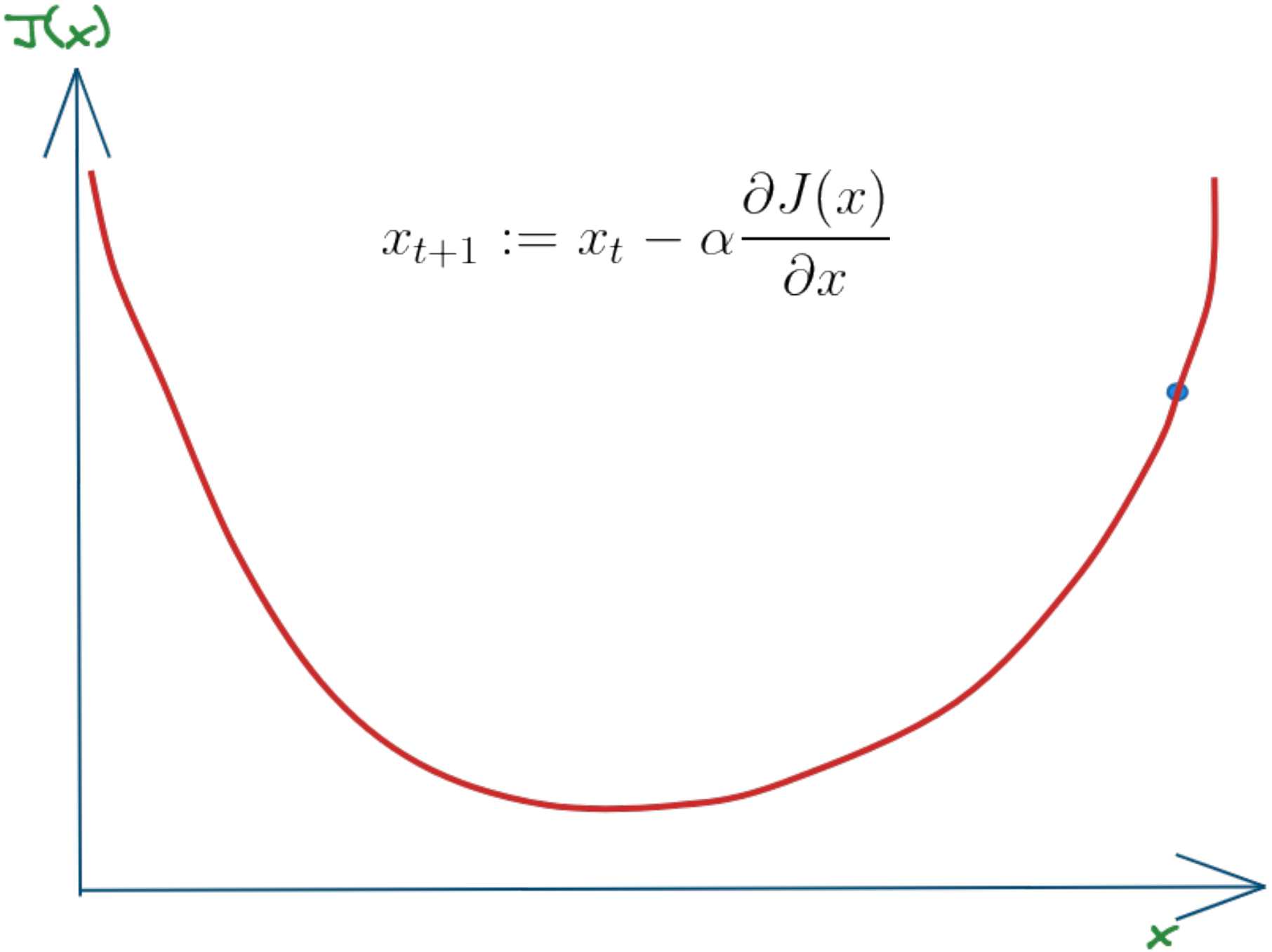
- The arrows represent the **direction of the steepest descent** (negative gradient)

- The idea is to **iteratively** take the direction of the **steepest descent** until you can no longer move downhill (local minimum)

Gradient Descent intuition

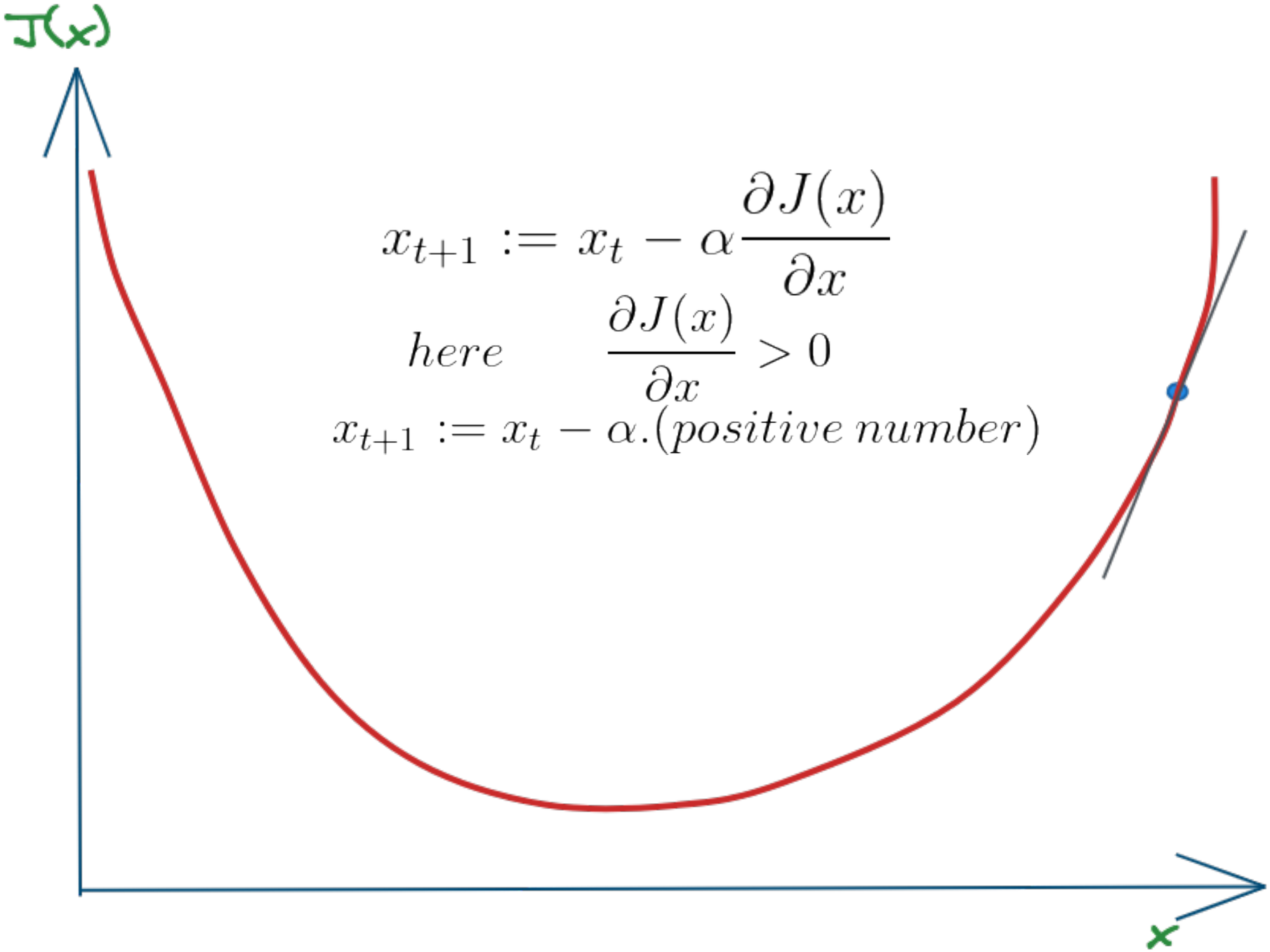


Gradient Descent intuition



$$x_{t+1} := x_t - \alpha \frac{\partial J(x)}{\partial x}$$

Gradient Descent intuition

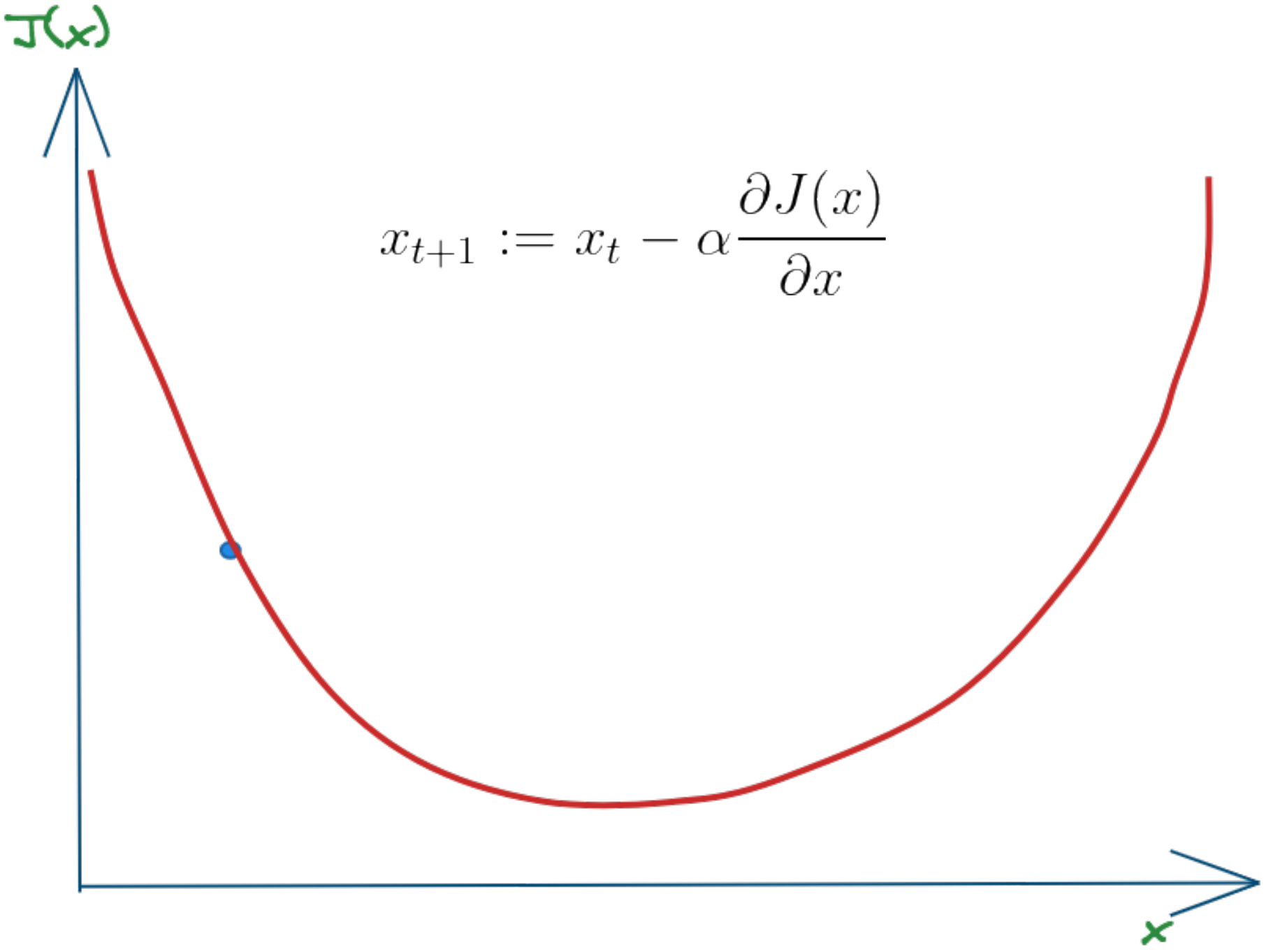


$$x_{t+1} := x_t - \alpha \frac{\partial J(x)}{\partial x}$$

$$here \quad \frac{\partial J(x)}{\partial x} > 0$$

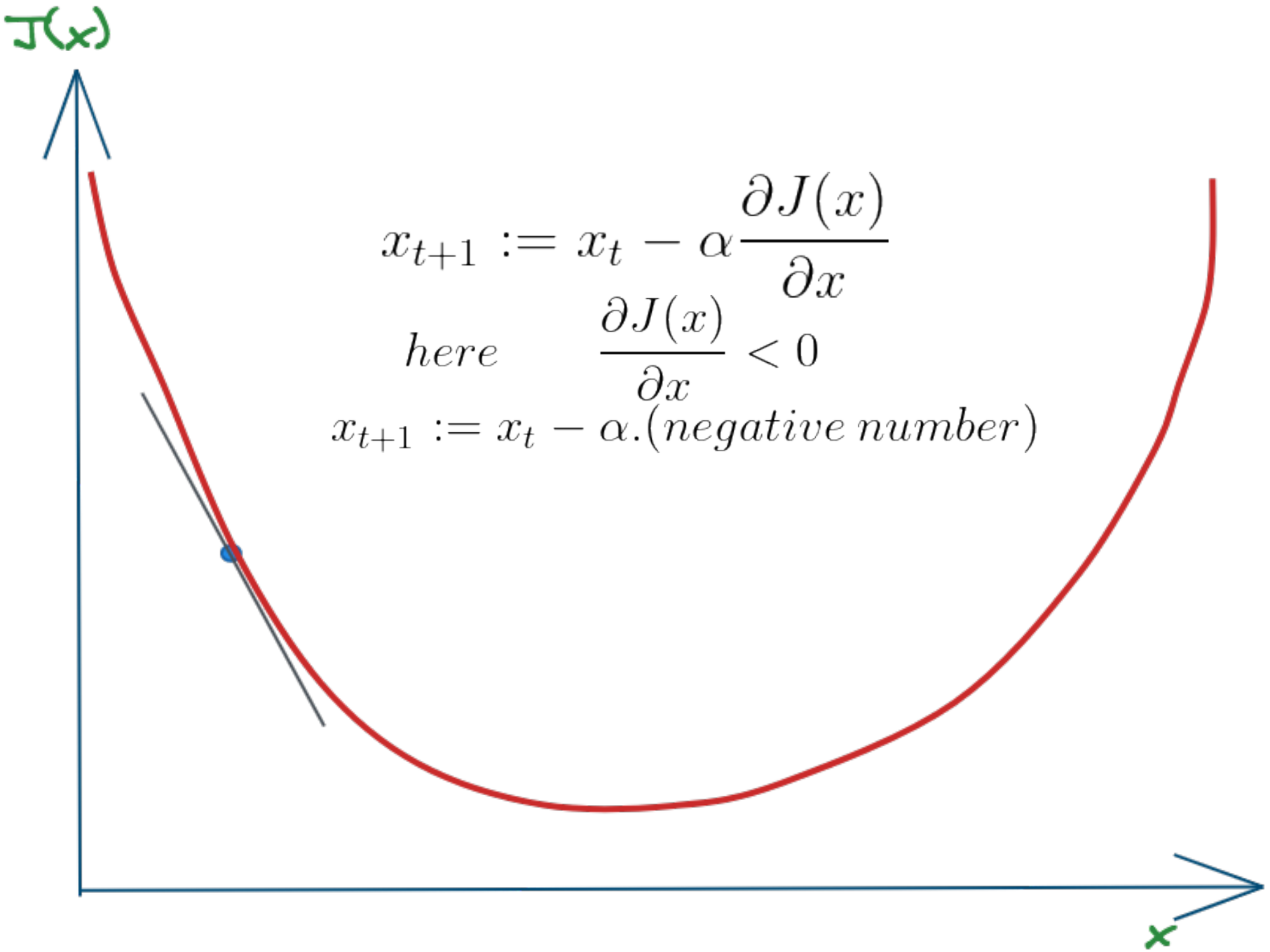
$$x_{t+1} := x_t - \alpha.(positive\ number)$$

Gradient Descent intuition



$$x_{t+1} := x_t - \alpha \frac{\partial J(x)}{\partial x}$$

Gradient Descent intuition

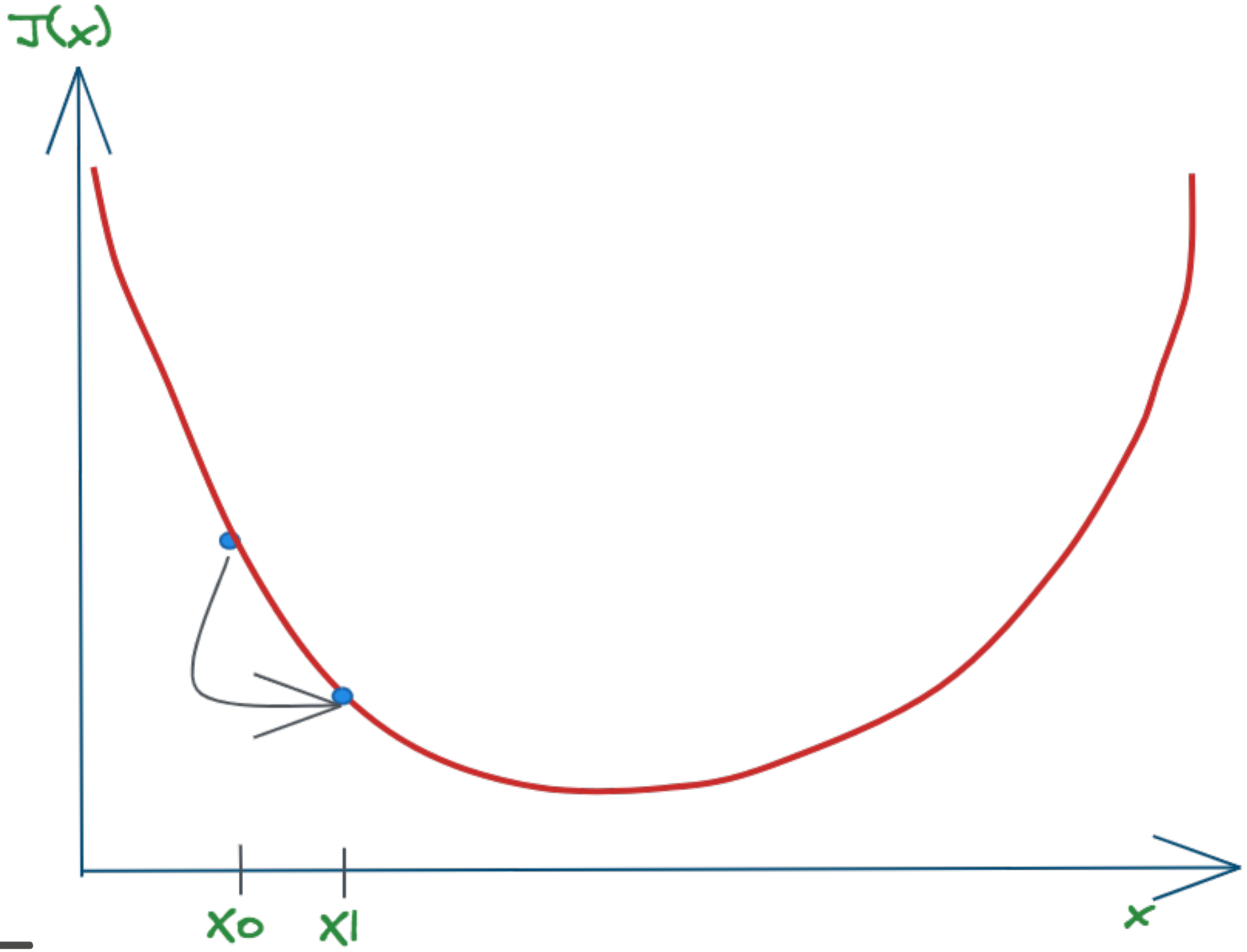


$$x_{t+1} := x_t - \alpha \frac{\partial J(x)}{\partial x}$$

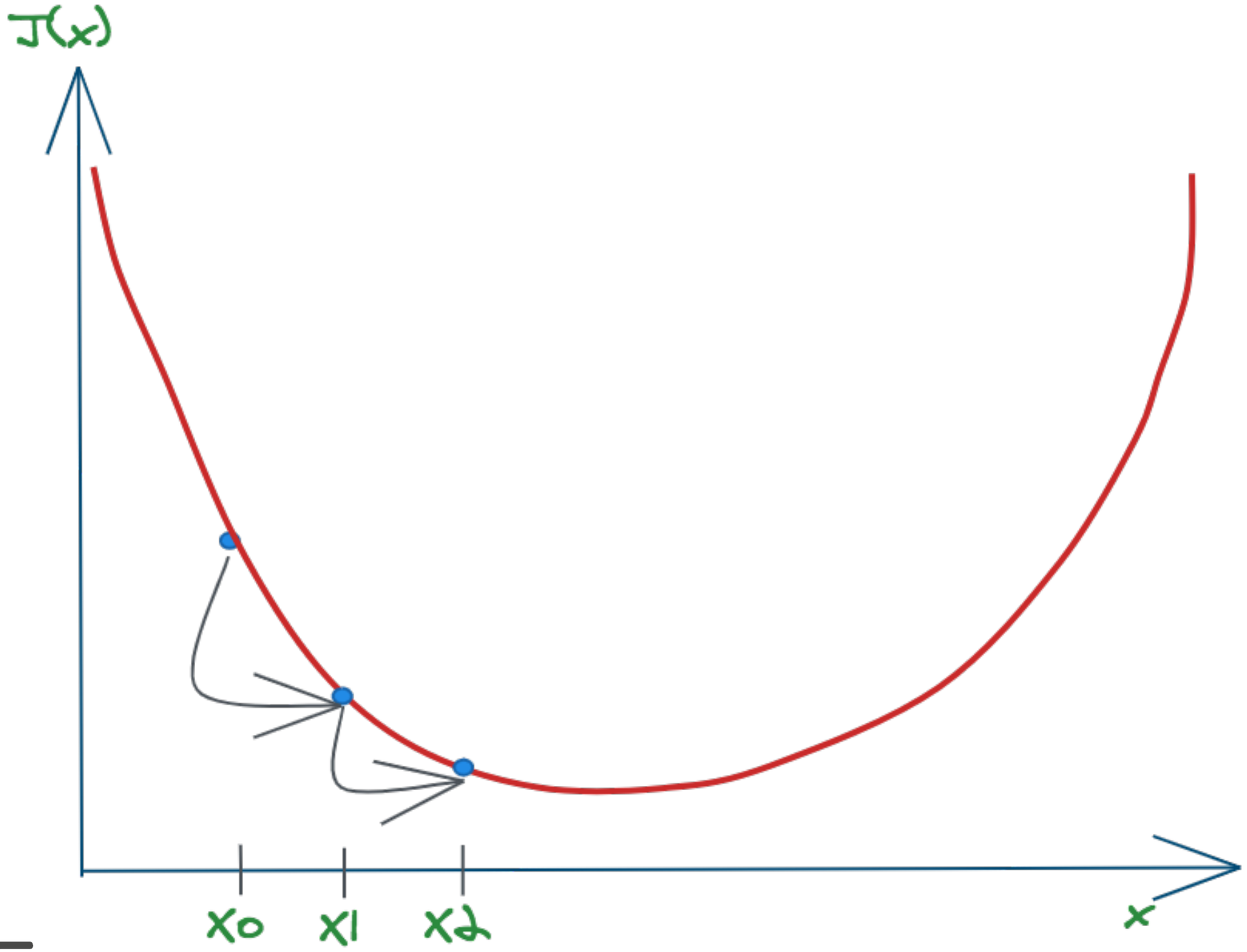
$$here \qquad \frac{\partial J(x)}{\partial x} < 0$$

$$x_{t+1} := x_t - \alpha.(negative\ number)$$

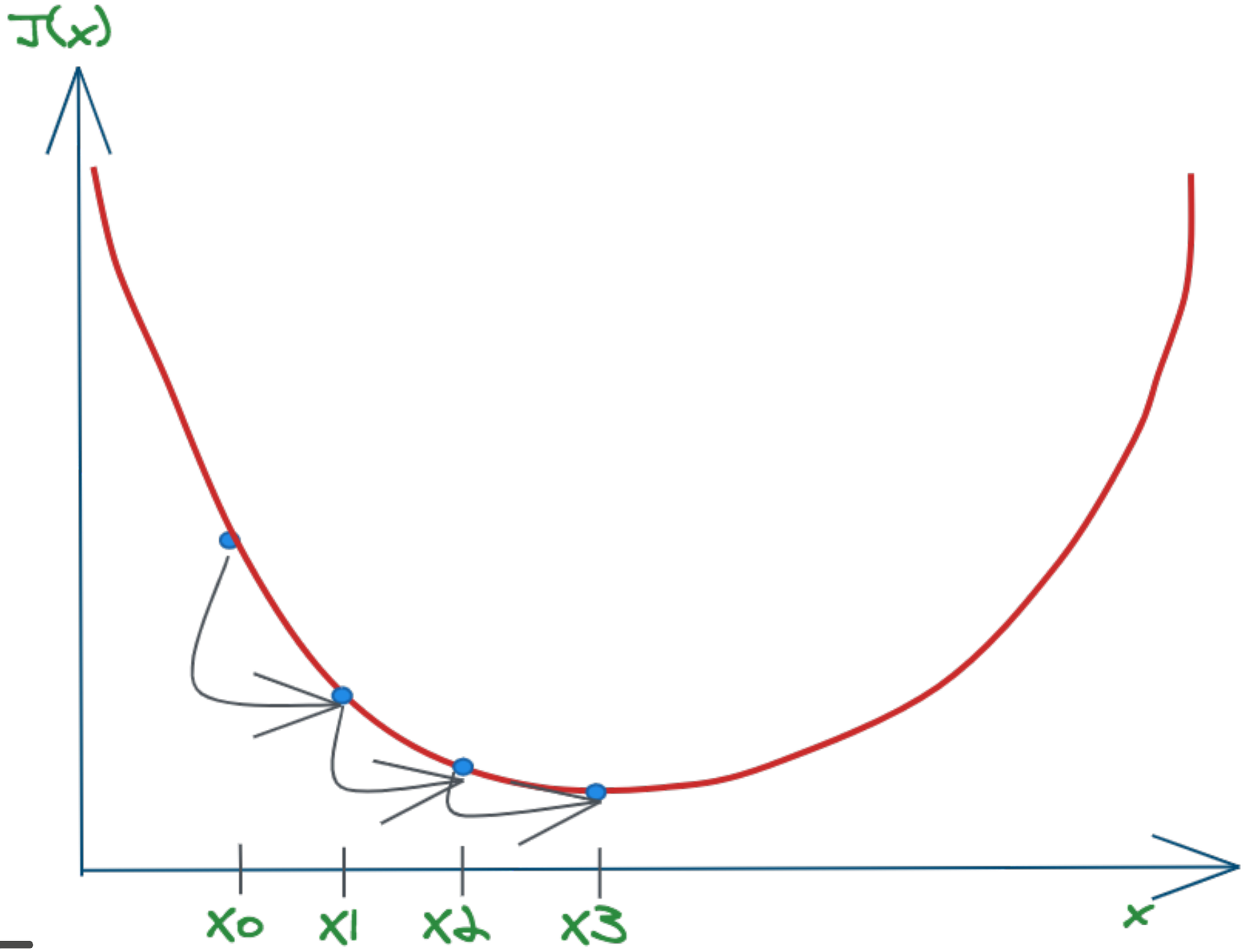
Gradient Descent intuition



Gradient Descent intuition

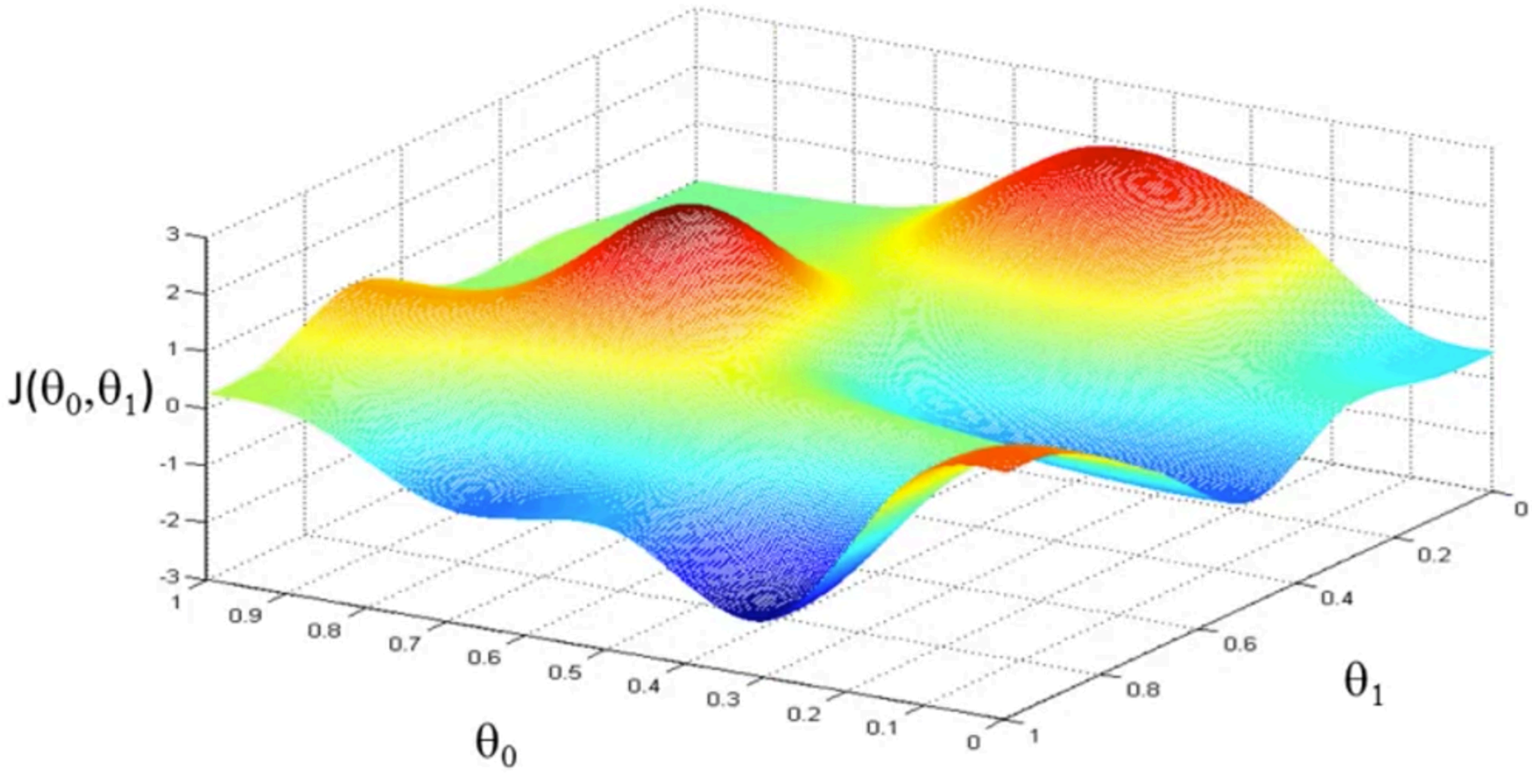


Gradient Descent intuition



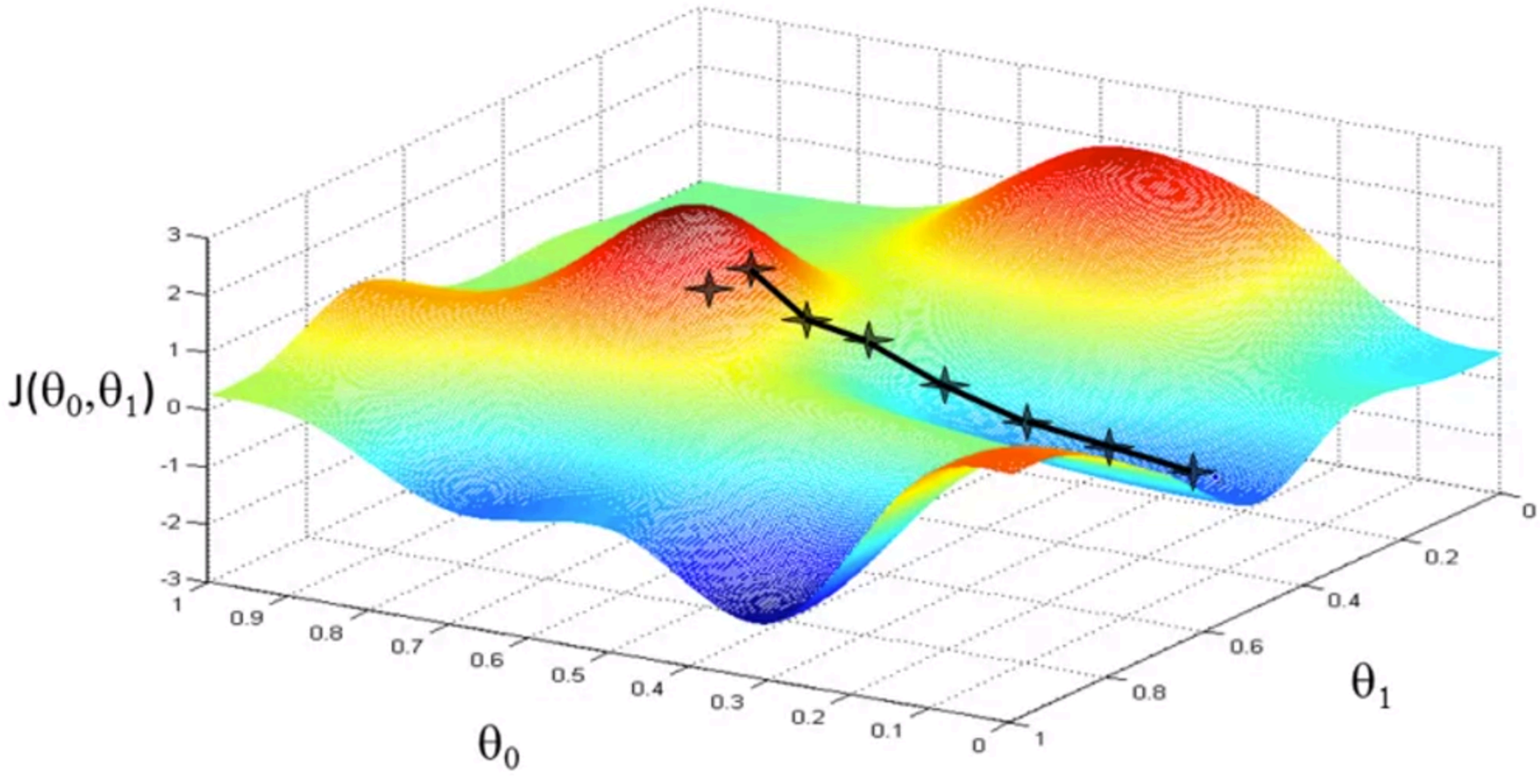
Gradient Descent intuition

- What we've seen on a 1-dimension can be **generalized** to any number of dimensions



Gradient Descent intuition

- What we've seen on a 1-dimension can be generalized to any number of dimensions



Gradient Descent algorithm

- The general algorithm can be described as the following:

$$Repeat\ until\ convergence:$$
$$\left\{ For\ each\ j \in [1, N]: \quad \theta_j := \theta_j - \alpha \frac{\partial J(\theta_1, \theta_2, ..., \theta_N)}{\partial \theta_j} \right\}$$

Step size

Partial derivative

Gradient Descent algorithm

- The general algorithm can be described as the following:

$$Repeat\ until\ convergence:$$
$$\left\{ For\ each\ j \in [1, N]: \quad \theta_j := \theta_j - \alpha \frac{\partial J(\theta_1, \theta_2, ..., \theta_N)}{\partial \theta_j} \right\}$$

Step size

Partial derivative

- The step size is a parameter of the algorithm that needs to be chosen carefully:
  - If it's too small, the convergence will be very slow

Gradient Descent algorithm

- The general algorithm can be described as the following:

$$Repeat\ until\ convergence:$$
$$\left\{ For\ each\ j \in [1, N]: \quad \theta_j := \theta_j - \alpha \frac{\partial J(\theta_1, \theta_2, ..., \theta_N)}{\partial \theta_j} \right\}$$

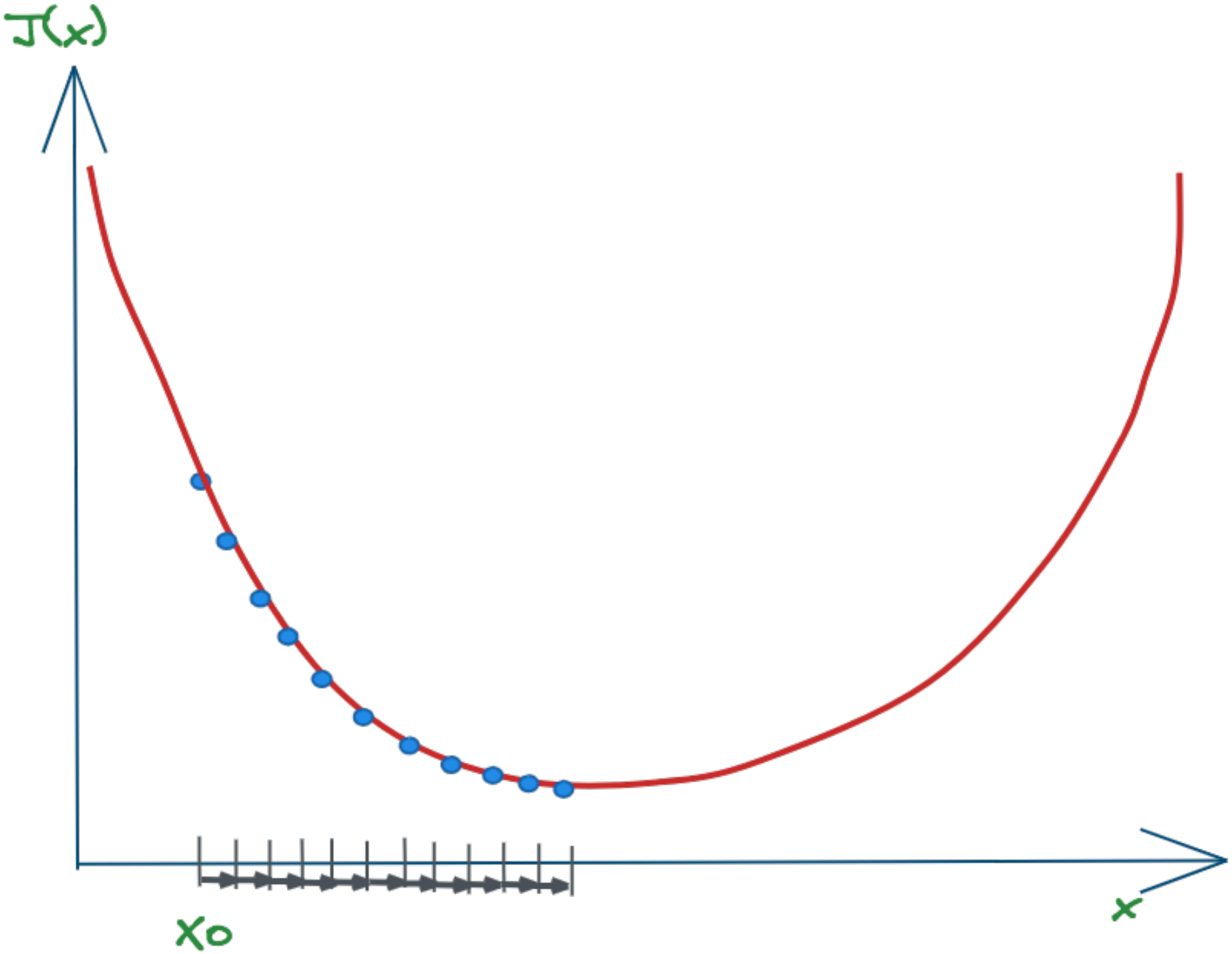
Step size

Partial derivative

- The step size is a parameter of the algorithm that needs to be chosen carefully:
  - If it's too small, the convergence will be very slow
  - If it's too big, we will overshoot the minimum

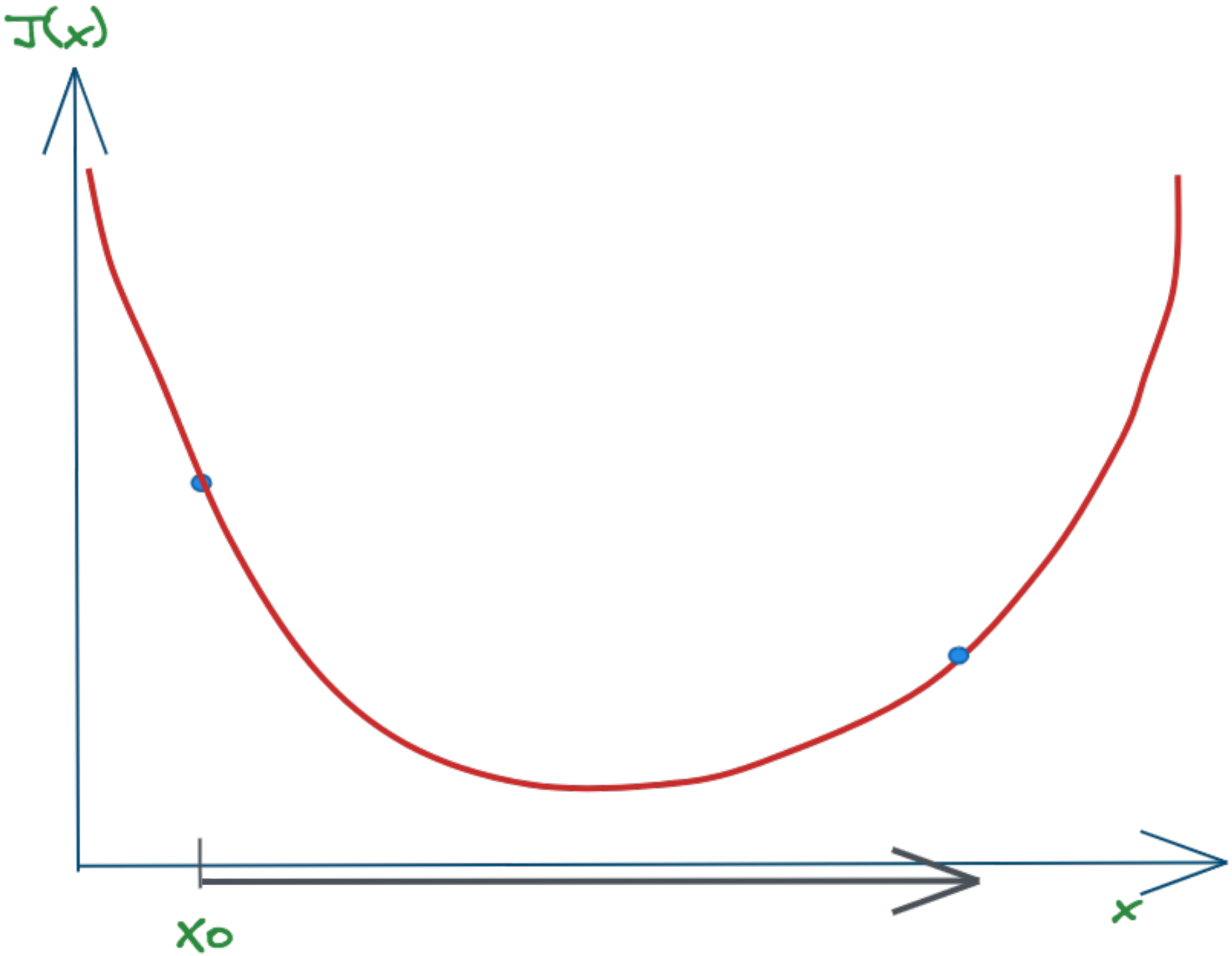
Gradient Descent algorithm

- If the step size is too small, the convergence will be **slow**



Gradient Descent algorithm

- If the step size is too big, we may **overshoot** the minimum



Gradient Descent for Linear Regression

- Remember our **hypothesis function** for N-dim: $h_\theta(x) = \theta^T X = \sum_{j=0}^{N} \theta_j x_j$

Gradient Descent for Linear Regression

- Remember our **hypothesis function** for N-dim: $h_\theta(x) = \theta^T X = \sum_{j=0}^{N} \theta_j x_j$
- We define our **cost function**: $J_\theta(x) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

Gradient Descent for Linear Regression

- Remember our **hypothesis function** for N-dim: $h_\theta(x) = \theta^T X = \sum_{j=0}^{N} \theta_j x_j$
- We define our **cost function**: $J_\theta(x) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$
- The **gradient descent** iterations are then defined for **each parameter:**

$$\theta_0 := \theta_0 - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\dots$$

$$\theta_N := \theta_N - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_N^{(i)}$$

Gradient Descent for Linear Regression

- We should define the **step size** (also called learning rate)

Gradient Descent for Linear Regression

- We should define the **step size** (also called learning rate)
- We should also define a **convergence** (stopping) **criterion**, either:

Gradient Descent for Linear Regression

- We should define the **step size** (also called learning rate)
- We should also define a **convergence** (stopping) **criterion**, either:
  - A maximum number of iterations
  - A minimum decrease of the cost function

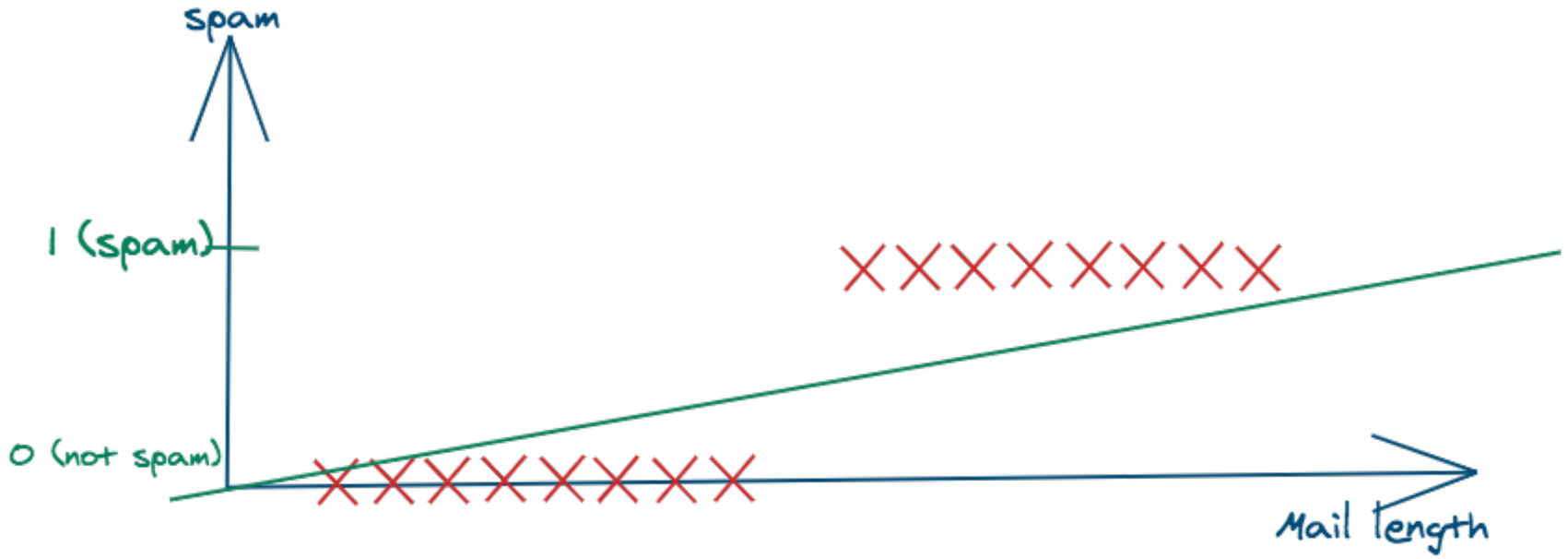
Gradient Descent for Linear Regression

- We should define the **step size** (also called learning rate)
- We should also define a **convergence** (stopping) **criterion**, either:
  - A maximum number of iterations
  - A minimum decrease of the cost function
- In the linear regression case, we have the **guarantee** that we will reach a **local minimum**

Classification

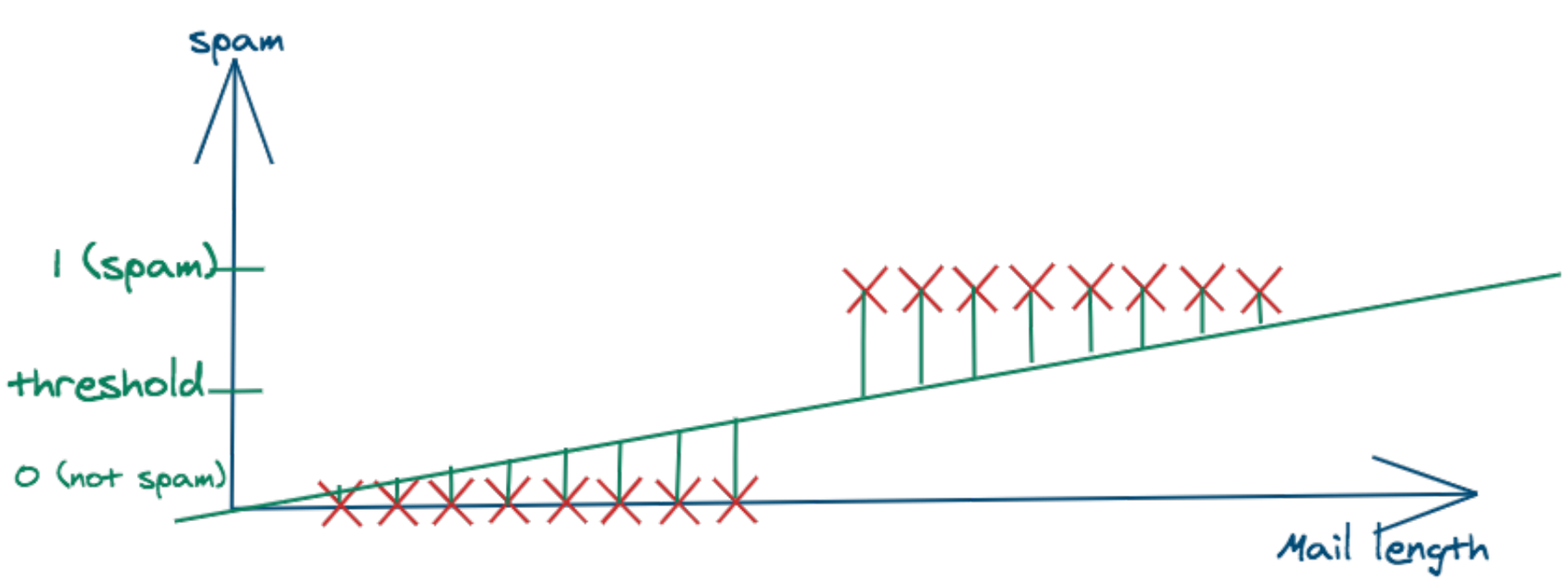
- In **classification**, the output takes values in a **finite unordered** set (spam/not spam, fraudulent/not fraudulent …)

- Let's try fitting a **linear model** to a **classification** task



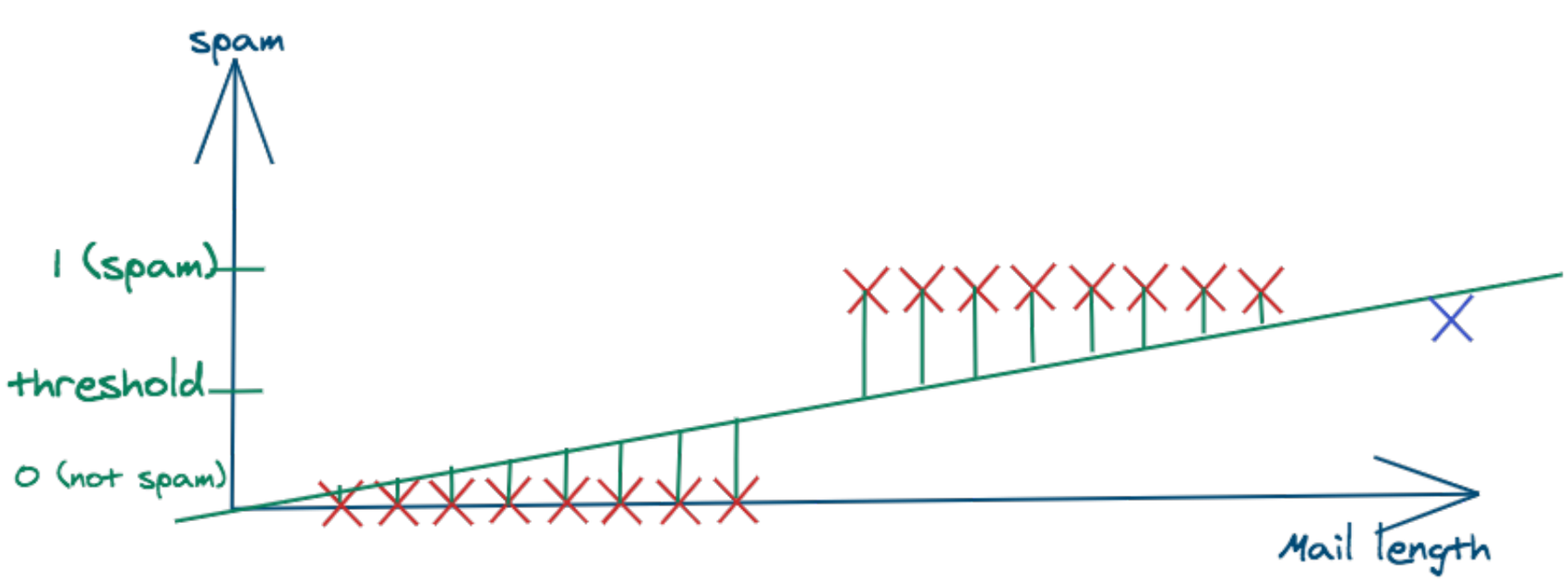
Classification

- The linear model is defined as $h_\theta(x) = \theta^T x$
- We also need to define a threshold: $if \quad h_\theta(x) > 0.5, \quad predict \quad 1$, $else \quad predict \quad 0$



Classification

- The linear model is defined as $h_\theta(x) = \theta^T x$
- We also need to define a threshold: $if \quad h_\theta(x) > 0.5, \quad predict \quad 1$, $\quad else \quad predict \quad 0$



Classification

- The linear model is **not adapted** for classification:

Classification

- The linear model is **not adapted** for classification:
  - It is very **sensitive** to **outliers**, it doesn't provide a clear **separation** either

Classification

- The linear model is **not adapted** for classification:
  - It is very **sensitive** to **outliers**, it doesn't provide a clear **separation** either
  - It takes values in R, not in [0,1]

Classification

- The linear model is **not adapted** for classification:
  - It is very **sensitive** to **outliers**, it doesn't provide a clear **separation** either
  - It takes values in R, not in [0,1]
- We want the values of our **classifier** to be in the [0,1] range to be interpreted as **probabilities**

Logistic Regression

- In **linear regression** the hypothesis is $h_\theta(x) = \theta^T x$

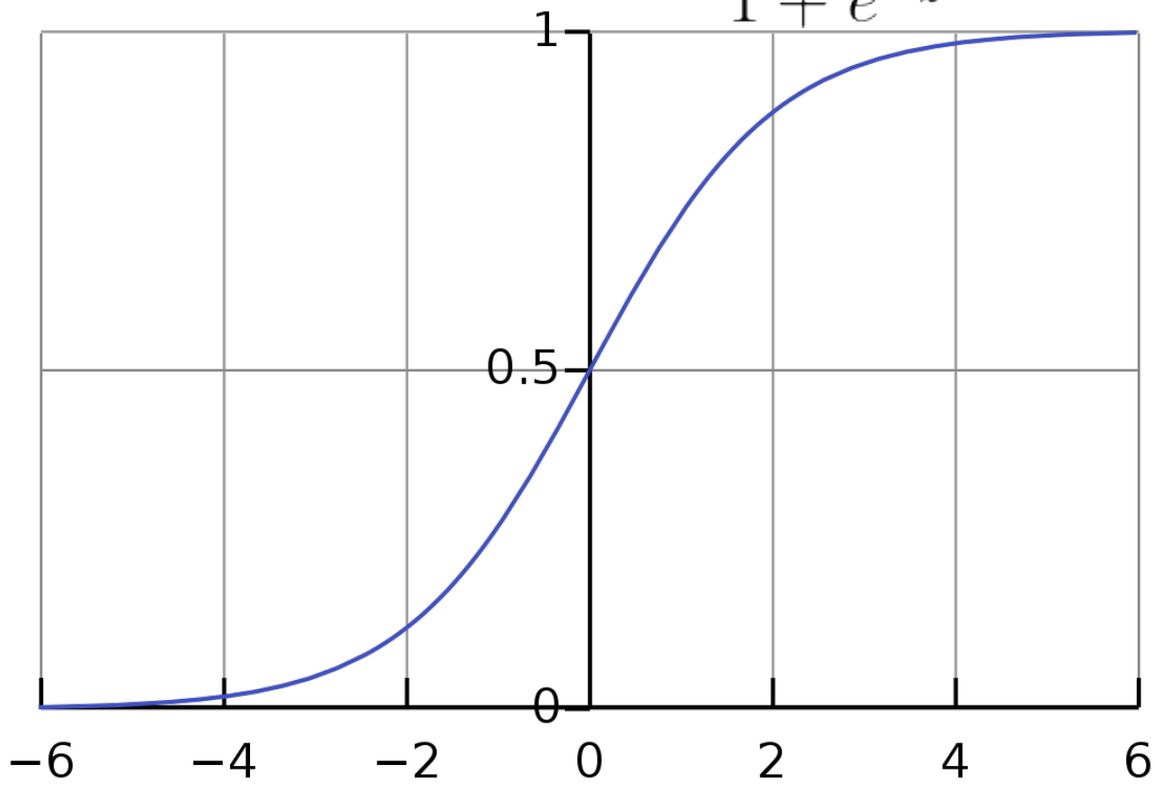
Logistic Regression

- In **linear regression** the hypothesis is $h_\theta(x) = \theta^T x$

- In **logistic regression**, we add a **sigmoid** function

$$h_\theta(x) = \sigma(\theta^T x)$$

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



Logistic Regression

- Thanks to the sigmoid function, the hypothesis function **verifies**

$$h_\theta(x) \in [0, 1]$$

Logistic Regression

- Thanks to the sigmoid function, the hypothesis function **verifies**

$$h_\theta(x) \in [0, 1]$$

- And can be **interpreted** as a **probability**

$$h_\theta(x) = \mathbb{P}(y = 1 \mid x)$$

Logistic Regression

- Thanks to the sigmoid function, the hypothesis function **verifies**

$$h_\theta(x) \in [0, 1]$$

- And can be **interpreted** as a **probability**

$$h_\theta(x) = \mathbb{P}(y = 1 \mid x)$$

- We can also define

$$\mathbb{P}(y = 0 \mid x) = 1 - \mathbb{P}(y = 1 \mid x) = 1 - h_\theta(x)$$

Logistic Regression

- Thanks to the sigmoid function, the hypothesis function **verifies**

$$h_\theta(x) \in [0, 1]$$

- And can be **interpreted** as a **probability**

$$h_\theta(x) = \mathbb{P}(y = 1 \mid x)$$

- We can also define

$$\mathbb{P}(y = 0 \mid x) = 1 - \mathbb{P}(y = 1 \mid x) = 1 - h_\theta(x)$$

- Now to find the **best set of parameters**, we need to define a **cost function**, just like we did for linear regression

Logistic Regression

- For **Linear Regression**, we defined our cost function as:

$$J_\theta(x) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Logistic Regression

- For **Linear Regression**, we defined our cost function as:

$$J_\theta(x) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

- In logistic regression, the **hypothesis** function $h_\theta(x)$ is **not linear**

Logistic Regression

- For **Linear Regression**, we defined our cost function as:

$$J_\theta(x) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

- In logistic regression, the **hypothesis** function $h_\theta(x)$ is **not linear**

- This means that the **cost function** would not be **convex**, and we have no **guarantees** that the gradient descent would **converge**

Logistic Regression

- For **Linear Regression**, we defined our cost function as:

$$J_\theta(x) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

- In logistic regression, the **hypothesis** function $h_\theta(x)$ is **not linear**

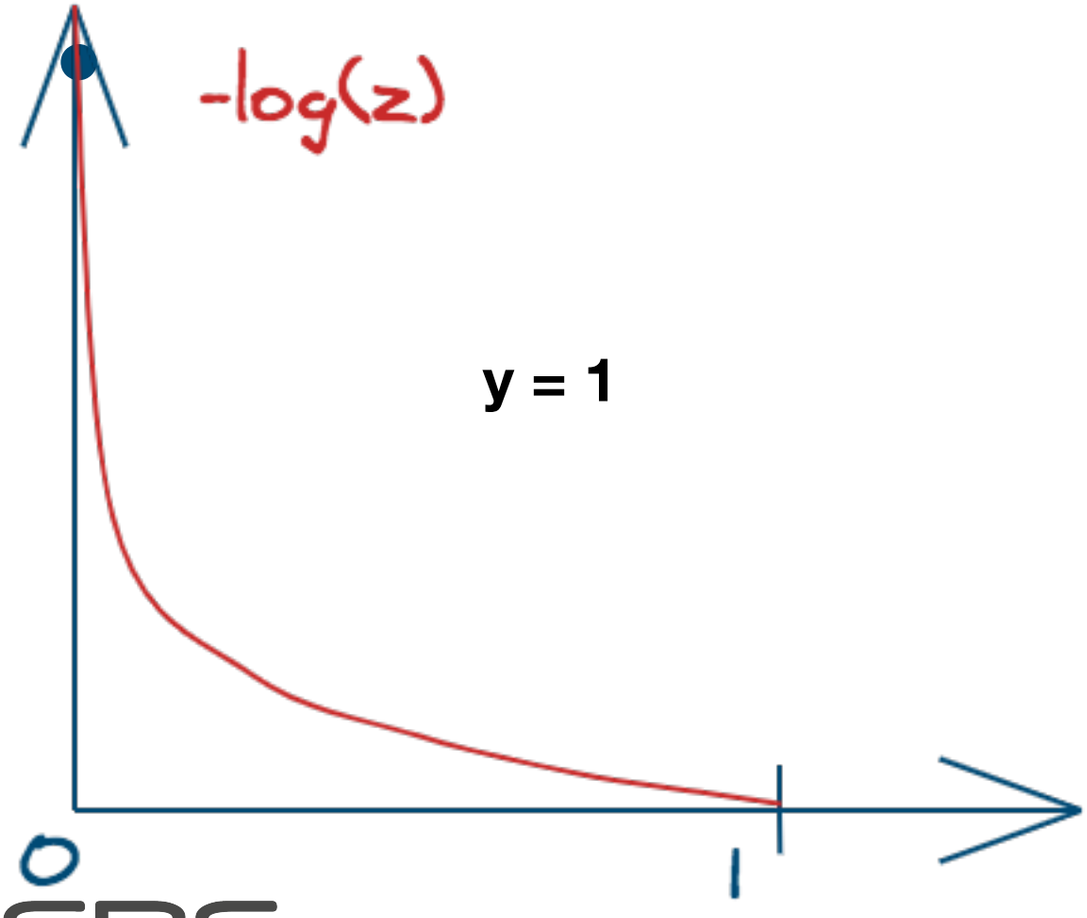
- This means that the **cost function** would not be **convex**, and we have no **guarantees** that the gradient descent would **converge**

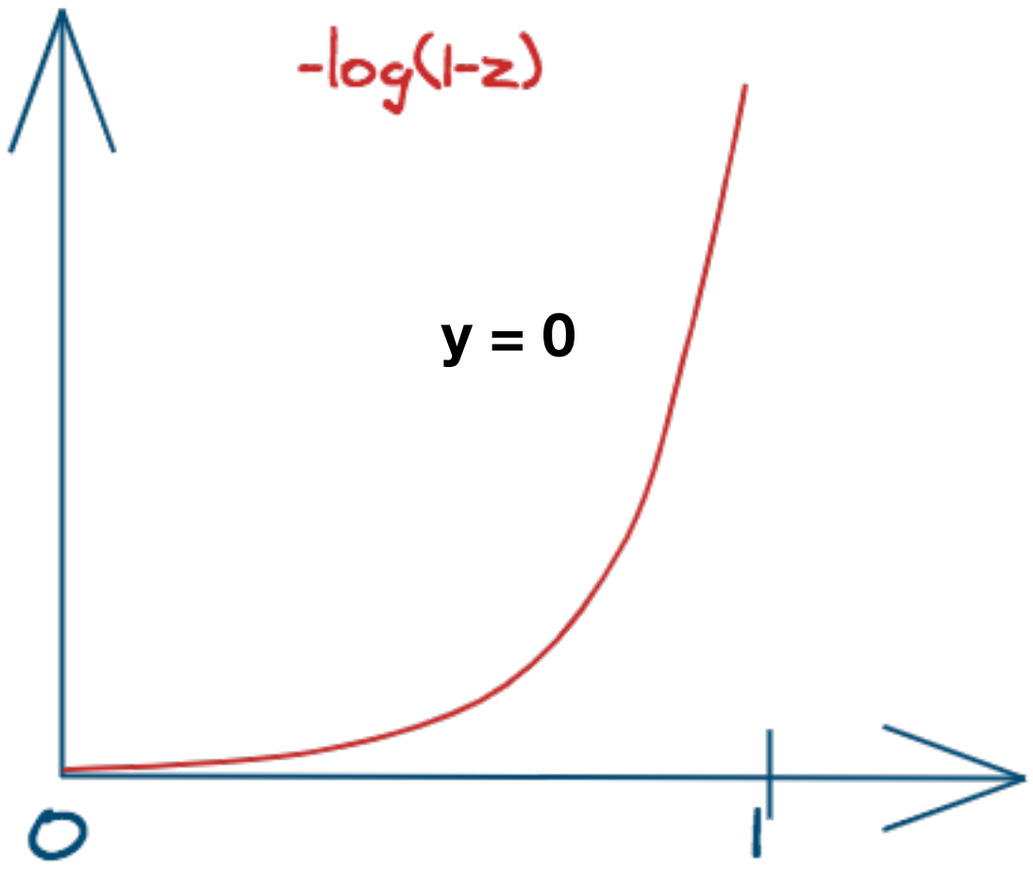
- We need to define a **convex cost function**

Logistic Regression

- We define the following **cost function** for logistic regression

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)), & \text{if } y = 1 \\ -log(1 - h_\theta(x)), & \text{if } y = 0 \end{cases}$$





Logistic Regression

- This cost function is **convex**, and presents some **interesting characteristics**

Logistic Regression

- This cost function is **convex**, and presents some **interesting characteristics**

$$\begin{cases} \text{if } y = 1 & \lim_{h_\theta(x)\to 1} cost = 0 \text{ and } \lim_{h_\theta(x)\to 0} cost = \infty \\ \text{if } y = 0 & \lim_{h_\theta(x)\to 1} cost = \infty \text{ and } \lim_{h_\theta(x)\to 0} cost = 0 \end{cases}$$

Logistic Regression

- This cost function is **convex**, and presents some **interesting characteristics**

$$\begin{cases} \text{if } y = 1 & \lim_{h_\theta(x)\to 1} cost = 0 \text{ and } \lim_{h_\theta(x)\to 0} cost = \infty \\ \text{if } y = 0 & \lim_{h_\theta(x)\to 1} cost = \infty \text{ and } \lim_{h_\theta(x)\to 0} cost = 0 \end{cases}$$

- In other words, it will **not penalize correct predictions, and highly penalize bad predictions**

Logistic Regression

- This cost function is **convex**, and presents some **interesting characteristics**

$$\begin{cases} \text{if } y = 1 & \lim_{h_\theta(x)\to 1} cost = 0 \text{ and } \lim_{h_\theta(x)\to 0} cost = \infty \\ \text{if } y = 0 & \lim_{h_\theta(x)\to 1} cost = \infty \text{ and } \lim_{h_\theta(x)\to 0} cost = 0 \end{cases}$$

- In other words, it will **not penalize correct predictions, and highly penalize bad predictions**

- We can **rewrite** $Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)), & \text{if } y = 1 \\ -log(1 - h_\theta(x)), & \text{if } y = 0 \end{cases}$

as $Cost(h_\theta(x), y) = -y * log(h_\theta(x)) - (1 - y) * log(1 - h_\theta(x))$

Logistic Regression

- We can **rewrite** our cost function for a given **dataset**

$$J_\theta(x) = -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}log(h_\theta(x^{(i)})) + (1-y^{(i)})log(1-(h_\theta(x^{(i)}))))$$

Logistic Regression

- We can **rewrite** our cost function for a given **dataset**

$$J_\theta(x) = -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}log(h_\theta(x^{(i)})) + (1-y^{(i)})log(1-(h_\theta(x^{(i)}))))$$

- Side note: The **same cost function** can be derived by using the **Maximum Log-Likelihood** method

Logistic Regression

- We can **rewrite** our cost function for a given **dataset**

$$J_\theta(x) = -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}log(h_\theta(x^{(i)})) + (1-y^{(i)})log(1-(h_\theta(x^{(i)}))))$$

- Side note: The **same cost function** can be derived by using the **Maximum Log-Likelihood** method

- Once we have defined our **cost function**, we just need to use the **gradient descent** to find the **best set of parameters**, identically to the linear regression model

Linear models

- Linear models are a **simple** yet **powerful** tool

Linear models

- Linear models are a **simple** yet **powerful** tool
- They are **easy to train** and are **highly interpretable**

Linear models

- Linear models are a **simple** yet **powerful** tool
- They are **easy to train** and are **highly interpretable**
- Ex: Bike rental prediction

Feature	Weight
Intercept	2399.4
isSeasonSummer	899.3
isHoliday	-686.1
isWorkingDay	124.9
isItRainingOrSnowing	-1901.5
temperature	110.7
humidity	-17.4
windSpeed	-42.5

Linear models

- Linear models are a **simple** yet **powerful** tool
- They are **easy to train** and are **highly interpretable**
- Ex: Bike rental prediction

Feature	Weight
Intercept	2399.4
isSeasonSummer	899.3
isHoliday	-686.1
isWorkingDay	124.9
isItRainingOrSnowing	-1901.5
temperature	110.7
humidity	-17.4
windSpeed	-42.5

- But what if our target is **not linear**, or if the decision space is **not linearly separable**?

Non-Linear Models: Randomized Trees & Forests

ÉCOLE D'INGÉNIEUR·E·S

Non-linear models

- It is **extremely unlikely** that the function we're trying to fit is **linear**, a linear representation is often done for **convenience**

Non-linear models

- It is **extremely unlikely** that the function we're trying to fit is **linear**, a linear representation is often done for **convenience**

- To **move beyond linearity**, the core idea of non-linear models is to **augment/replace** the vector of inputs *X* with **additional** variables, which are **transformations** of *X*

Non-linear models

- It is **extremely unlikely** that the function we're trying to fit is **linear**, a linear representation is often done for **convenience**

- To **move beyond linearity**, the core idea of non-linear models is to **augment/replace** the vector of inputs *X* with **additional** variables, which are **transformations** of *X*

- We denote by $h_m(X) : \mathbb{R}^p \mapsto \mathbb{R}$ the *m*-th **transformation** of *X*

Non-linear models

- It is **extremely unlikely** that the function we're trying to fit is **linear**, a linear representation is often done for **convenience**

- To **move beyond linearity**, the core idea of non-linear models is to **augment/replace** the vector of inputs *X* with **additional** variables, which are **transformations** of *X*

- We denote by $h_m(X) : \mathbb{R}^p \mapsto \mathbb{R}$ the *m*-th **transformation** of *X*

- We then model $$f(X) = \sum_{m=1}^{M} \beta_m h_m(X)$$

Non-linear models

- Examples of transformations

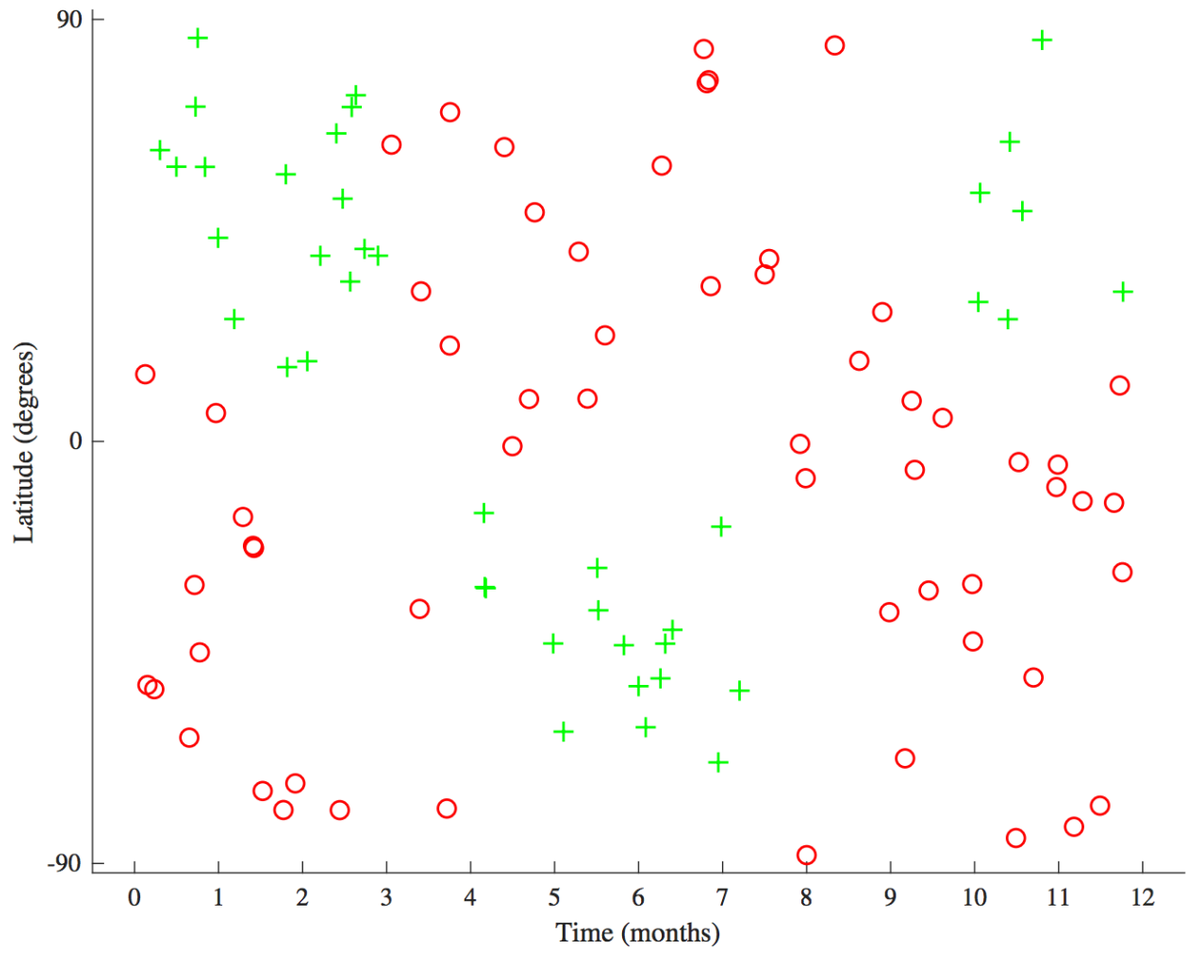
Polynomial transformations: $h_m(X) = X_j^2 \text{ or } X_j X_k$

Non-linear transformations: $h_m(X) = log(X_j) \text{ or } \sqrt{X_j}$

An indicator for a region of X_k $h_m(X) = \mathbb{I}(L_m \leq X_k < U_m)$

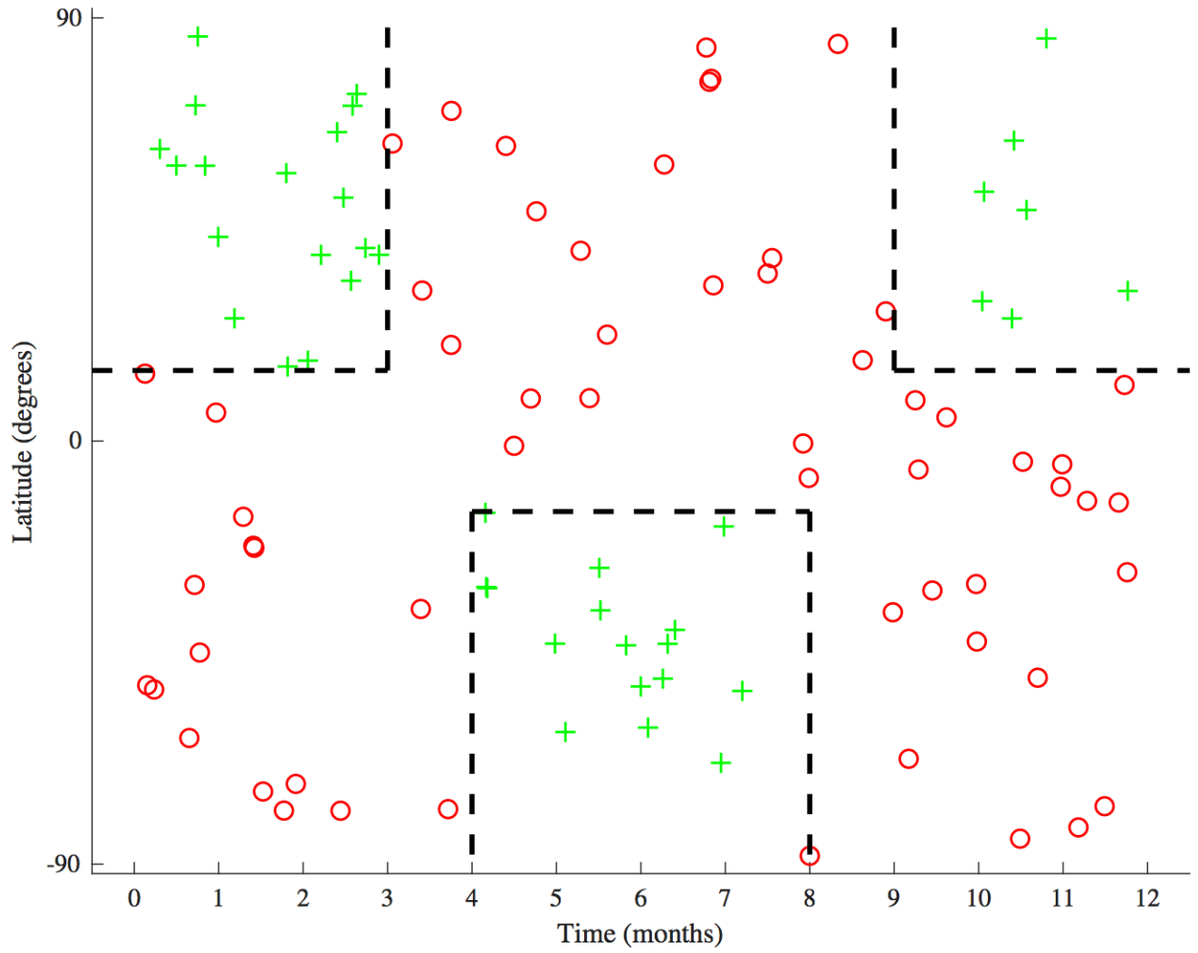
Non-linear models

- Example of a non-linearly separable decision space: a very simplistic snow prediction model (based on latitude and month)



Non-linear models

- Example of a non-linearly separable decision space: a very simplistic snow prediction model (based on latitude and month)



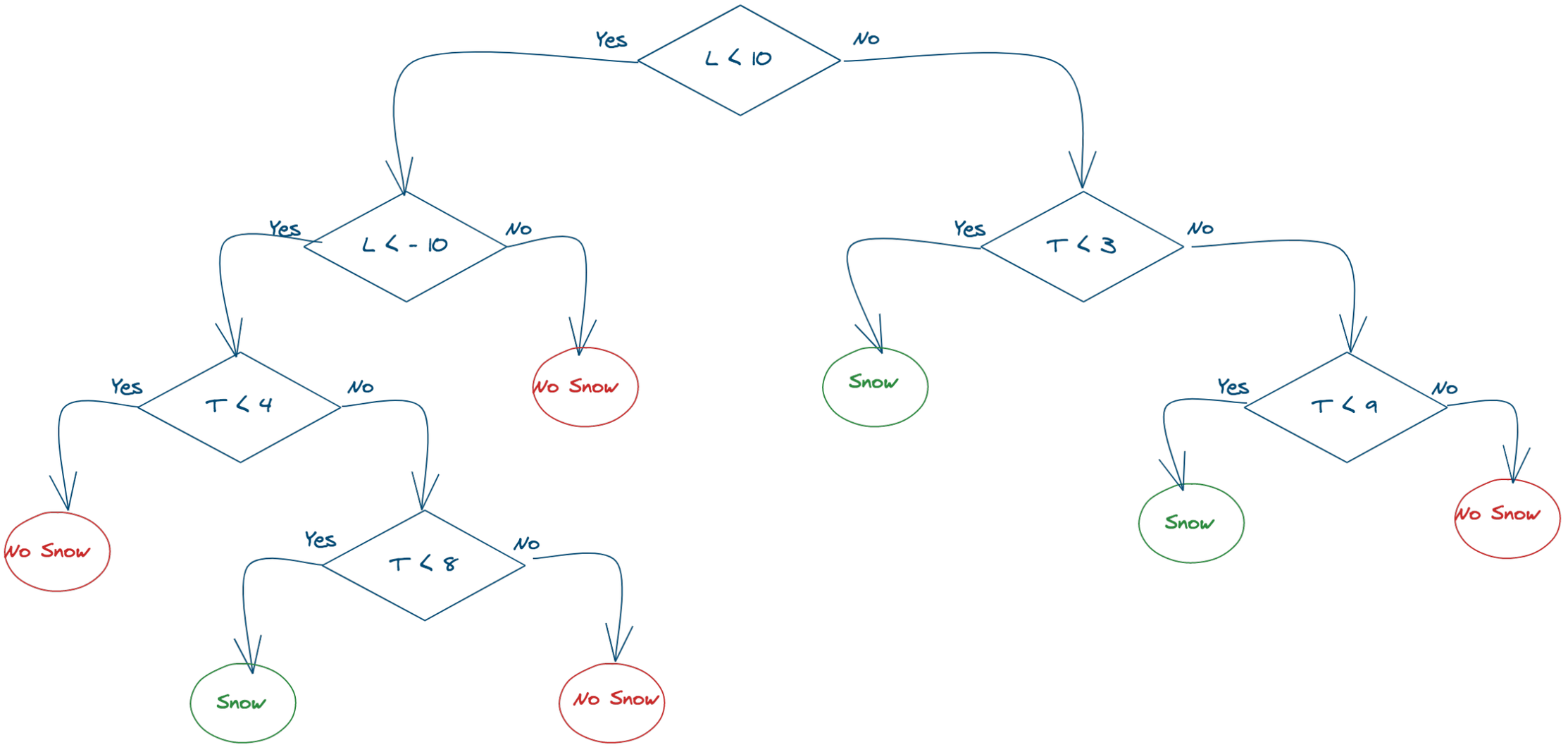
Non-linear models

- Snow prediction model: let's try to model what we see:

$$\begin{aligned}
&\text{If Latitude} \geq 10 \text{ and Month} < 3 \text{ then Snow} \\
&\text{If Latitude} \leq -10 \text{ and } 4 \leq \text{Month} < 8 \text{ then Snow} \\
&\text{If Latitude} \geq 10 \text{ and Month} \geq 9 \text{ then Snow} \\
&\text{Else No Snow}
\end{aligned}$$

Non-linear models

- Snow prediction model: this is what we call a **decision tree**



Classification And Regression Trees (CART)

- What we've seen in the last example is a **classification decision tree**. How can we **train** such models? Precisely, how does the algorithm **decide how to split the data**?

Classification And Regression Trees (CART)

- What we've seen in the last example is a **classification decision tree**. How can we **train** such models? Precisely, how does the algorithm **decide how to split the data**?

- A **decision tree** generates an approximate solution via **greedy**, **top-down**, **recursive** partitioning

Classification And Regression Trees (CART)

- What we've seen in the last example is a **classification decision tree**. How can we **train** such models? Precisely, how does the algorithm **decide how to split the data**?

- A **decision tree** generates an approximate solution via **greedy**, **top-down**, **recursive** partitioning

- We start with the **original** input space and **split** it into **two child regions** by **threshold** on a single variable

Classification And Regression Trees (CART)

- What we've seen in the last example is a **classification decision tree**. How can we **train** such models? Precisely, how does the algorithm **decide how to split the data**?

- A **decision tree** generates an approximate solution via **greedy**, **top-down**, **recursive** partitioning

- We start with the **original** input space and **split** it into **two child regions** by **threshold** on a single variable

- We then take one of these child regions partition it **again** on a single variable

Classification And Regression Trees (CART)

- We do the training of our model in a **recursive** manner. At each step, we select a **leaf node**, a **feature** and a **threshold** to form a new **split**

Classification And Regression Trees (CART)

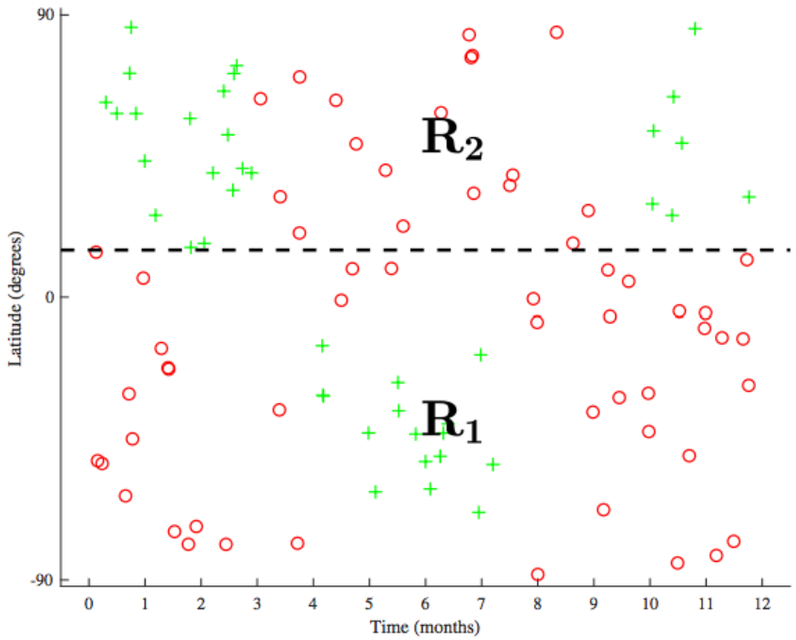
- We do the training of our model in a **recursive** manner. At each step, we select a **leaf node**, a **feature** and a **threshold** to form a new **split**

- Formally, given a parent region R_p , a feature index j and a threshold t, we obtain two child regions R_1 and R_2 as follows:

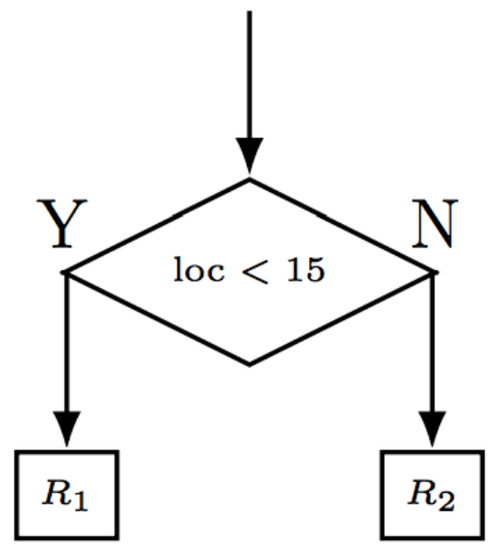
$$R_1 = \{X | X_j < t, X \in R_p\}$$
$$R_2 = \{X | X_j \geq t, X \in R_p\}$$

Classification And Regression Trees (CART)

- Let's see it in action for the snow example

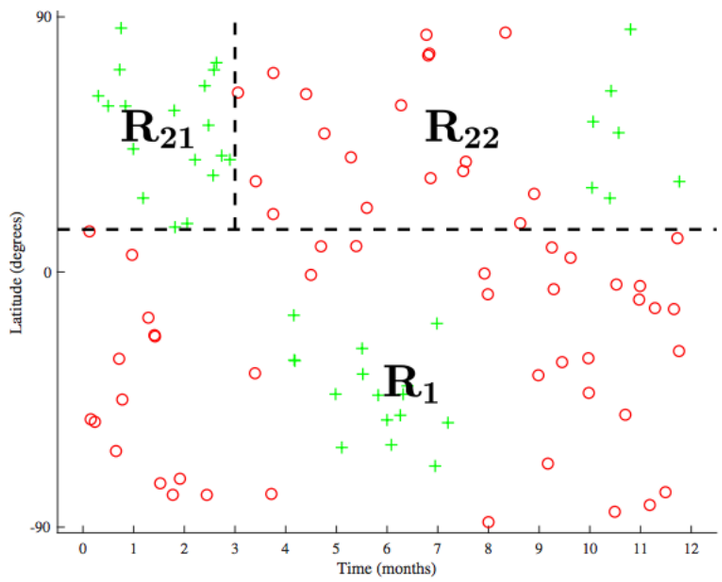


(a)

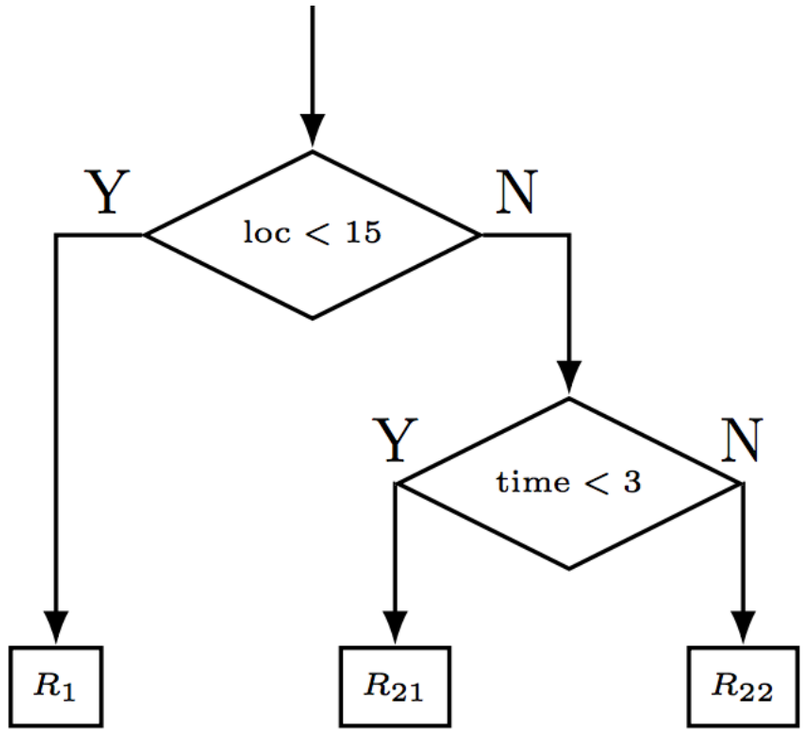


Classification And Regression Trees (CART)

- Let's see it in action for the snow example

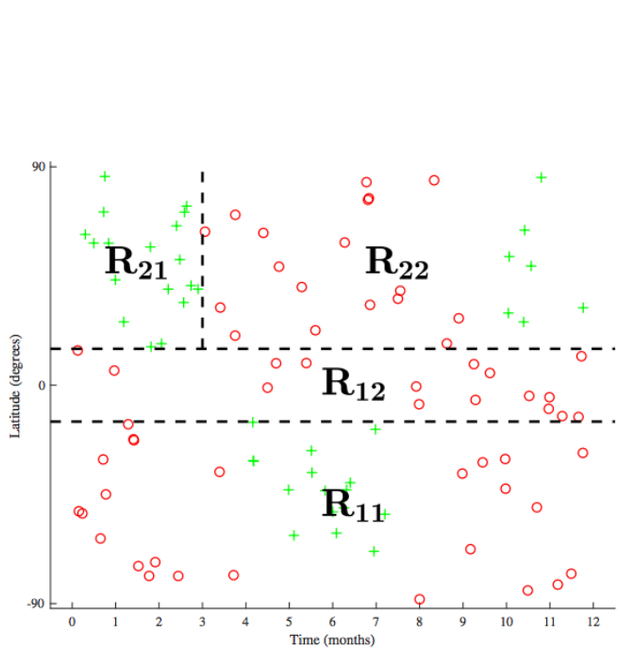


(b)

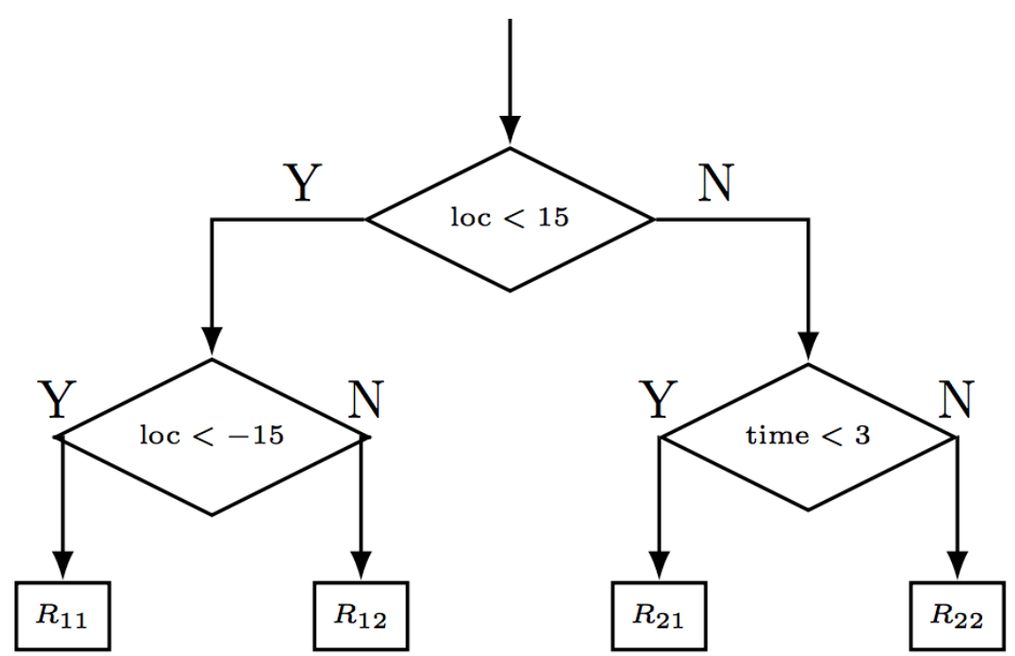


Classification And Regression Trees (CART)

- Let's see it in action for the snow example



(c)



Classification And Regression Trees (CART)

- How to **choose** the splits?

Classification And Regression Trees (CART)

- How to **choose** the splits?

- We need to define a **loss** L, given a split of a parent region R_p into two child regions R_1 and R_2, we compute the loss of the parent region and the weighted loss of the children

Classification And Regression Trees (CART)

- How to **choose** the splits?

- We need to define a **loss** L, given a split of a parent region R_p into two child regions R_1 and R_2, we compute the loss of the parent region and the weighted loss of the children

- We select the leaf region, feature, and threshold that will **maximize** our **decrease** in loss

$$L(R_P) - \frac{|R_1|L(R_1) + |R_2|L(R_2)}{|R_1| + |R_2|}$$

Classification And Regression Trees (CART)

- How to **choose** the splits?

- We need to define a **loss** L, given a split of a parent region R_p into two child regions R_1 and R_2, we compute the loss of the parent region and the weighted loss of the children

- We select the leaf region, feature, and threshold that will **maximize** our **decrease** in loss

$$L(R_P) - \frac{|R_1|L(R_1) + |R_2|L(R_2)}{|R_1| + |R_2|}$$

- We continue until we meet a **stopping criterion**

Classification And Regression Trees (CART)

- We need to define a **specific loss** *L* for each task

Classification And Regression Trees (CART)

- We need to define a **specific loss** *L* for each task
  - For **classification**:
    - **Misclassification loss**

Classification And Regression Trees (CART)

- We need to define a **specific loss** *L* for each task
  - For **classification**:
    - **Misclassification loss**

$$L_{misclass}(R) = 1 - max_c(\hat{p}_c)$$

where $\hat{p}_c$ is the proportion of examples in R that are of class c

Classification And Regression Trees (CART)

- We need to define a specific loss *L* for each task
  - For **classification**:
    - **Cross entropy loss**

$$L_{cross}(R) = -\sum_{c} \hat{p}_c log_2 \hat{p}_c$$

Classification And Regression Trees (CART)

- We need to define a specific loss *L* for each task
  - For **classification**:
    - **Cross entropy loss**

$$L_{cross}(R) = -\sum_{c} \hat{p}_c log_2 \hat{p}_c$$

- From an **information-theoretic** perspective, **cross-entropy** measure the number of bits needed to specify the outcome (or class) given that the distribution is known. The decrease of loss from parent to child is known as **information gain**

Classification And Regression Trees (CART)

- We need to define a specific loss L for each task
  - For **regression**, for each data point x_i we have an associated value y_i
  - The final **prediction** for a region R is the mean of all the values in the region

$$\hat{y} = \frac{\sum_{i \in R} y_i}{|R|}$$

- We define the **squared loss** as

$$L_{squared}(R) = \frac{\sum_{i \in R} (y_i - \hat{y})^2}{|R|}$$

Classification And Regression Trees (CART)

- Let's talk about some **stopping criteria**

- The simplest criteria would be to fully grow the tree: continue until each leaf region contains exactly one training data point

- But this would be **expensive** and the model would **overfit**

- We need other stopping **heuristics**

Classification And Regression Trees (CART)

- Some common stopping **heuristics:**

Classification And Regression Trees (CART)

- Some common stopping **heuristics:**
  - **Minimum Leaf Size**: Do not split *R* if its cardinality falls below a threshold

Classification And Regression Trees (CART)

- Some common stopping **heuristics:**
  - **Minimum Leaf Size**: Do not split *R* if its cardinality falls below a threshold
  - **Maximum Depth**: Do not split *R* if more than a fixed threshold of splits were already taken to reach *R*

Classification And Regression Trees (CART)

- Some common stopping **heuristics:**
  - **Minimum Leaf Size**: Do not split *R* if its cardinality falls below a threshold
  - **Maximum Depth**: Do not split *R* if more than a fixed threshold of splits were already taken to reach *R*
  - **Maximum Number of Nodes**: Stop if a tree has more leaf nodes than a fixed threshold

Classification And Regression Trees (CART)

- We can be tempted to define a **minimum decrease** in loss for a split (like we did for gradient descent)

Classification And Regression Trees (CART)

- We can be tempted to define a **minimum decrease** in loss for a split (like we did for gradient descent)

- This is **problematic** as sometimes, we need to **first** threshold on multiple features (with no significant decrease of loss) before **reaching a good split**

Classification And Regression Trees (CART)

- We can be tempted to define a **minimum decrease** in loss for a split (like we did for gradient descent)

- This is **problematic** as sometimes, we need to **first** threshold on multiple features (with no significant decrease of loss) before **reaching a good split**

- This is mainly due to **higher order interactions** between features

Classification And Regression Trees (CART)

- We can be tempted to define a **minimum decrease** in loss for a split (like we did for gradient descent)

- This is **problematic** as sometimes, we need to **first** threshold on multiple features (with no significant decrease of loss) before **reaching a good split**

- This is mainly due to **higher order interactions** between features

- A better approach is called **pruning**: we first fully grow the tree, and then prune away nodes that minimally decrease the loss

Random Forest

- A **Random Forest** is an **ensemble** of **Decision Trees**, trained with the **bagging** method: these trees are **merged** together to get a more **accurate** and **stable** prediction

Random Forest

- A **Random Forest** is an **ensemble** of **Decision Trees**, trained with the **bagging** method: these trees are **merged** together to get a more **accurate** and **stable** prediction

- Each tree is grown as follows (if we have N data points and M variables)

Random Forest

- A **Random Forest** is an **ensemble** of **Decision Trees**, trained with the **bagging** method: these trees are **merged** together to get a more **accurate** and **stable** prediction

- Each tree is grown as follows (if we have N data points and M variables)

1. **Randomly sample** N data points **with replacement:** this our training set for the current tree, we then leave out 1/3 of this data (**out-of-bag data**) to get an **unbiased** estimate of the error

Random Forest

- A **Random Forest** is an **ensemble** of **Decision Trees**, trained with the **bagging** method: these trees are **merged** together to get a more **accurate** and **stable** prediction

- Each tree is grown as follows (if we have N data points and M variables)

1. **Randomly sample** N data points **with replacement:** this our training set for the current tree, we then leave out 1/3 of this data (**out-of-bag data**) to get an **unbiased** estimate of the error

2. At each node **m (m<<M)** variables are selected at **random**, the best split on these m variables is used to split the node

Random Forest

- A **Random Forest** is an **ensemble** of **Decision Trees**, trained with the **bagging** method: these trees are **merged** together to get a more **accurate** and **stable** prediction

- Each tree is grown as follows (if we have N data points and M variables)

1. **Randomly sample** N data points **with replacement:** this our training set for the current tree, we then leave out 1/3 of this data (**out-of-bag data**) to get an **unbiased** estimate of the error

2. At each node **m (m<<M)** variables are selected at **random**, the best split on these m variables is used to split the node

3. Each tree is grown to the largest extent possible (**no pruning**)

Random Forest

- Once all the trees are grown, to make a prediction we can either:
  - Do a **majority vote** for **classification** tasks
  - **Average** all the trees' **estimates** for **regression** tasks

Random Forest

- Random Forest shows very good performance compared to other models

Random Forest

- Random Forest shows very good performance compared to other models
- It’s efficient on large data sets

Random Forest

- Random Forest shows very good performance compared to other models
- It’s efficient on large data sets
- It can handle thousands of input variables

Random Forest

- Random Forest shows very good performance compared to other models
- It’s efficient on large data sets
- It can handle thousands of input variables
- It gives an estimate of what variables are important

Random Forest

- Random Forest shows very good performance compared to other models
- It’s efficient on large data sets
- It can handle thousands of input variables
- It gives an estimate of what variables are important
- It offers an experimental method for detecting variable interactions

Random Forest

- Random Forest shows very good performance compared to other models
- It’s efficient on large data sets
- It can handle thousands of input variables
- It gives an estimate of what variables are important
- It offers an experimental method for detecting variable interactions
- It handles very well missing data