



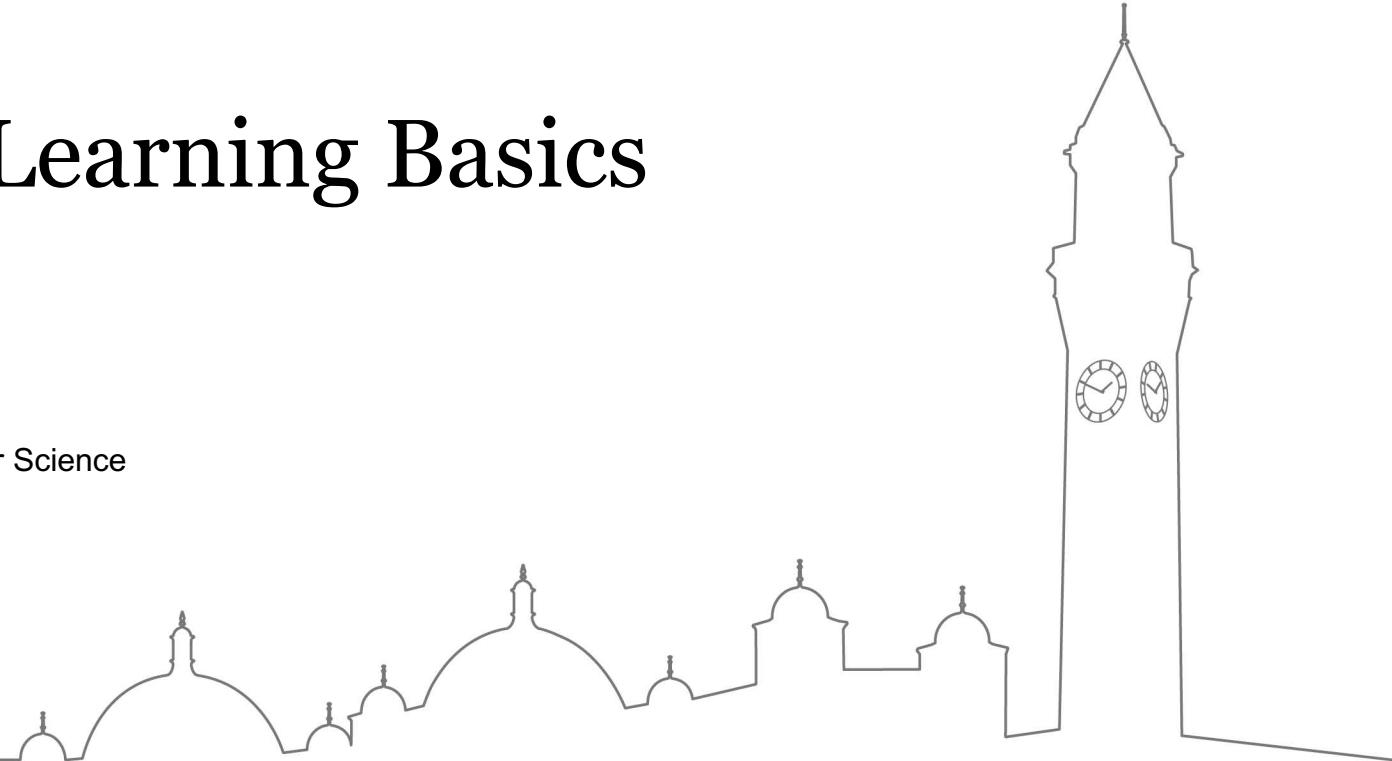
UNIVERSITY OF  
BIRMINGHAM

# Machine Learning Basics

**Dr. Shuo Wang**

Lecturer, School of Computer Science

[s.wang.2@bham.ac.uk](mailto:s.wang.2@bham.ac.uk)



# Lecture Overview

- What is machine learning?
- Categories of machine learning
- How does machine learning work?  
Supervised learning workflow
- Machine learning algorithms
- Model evaluation



# What is machine learning?



# What is machine learning?



# What is machine learning?



# What is machine learning?

"A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

-- Tom Mitchell, Professor at Carnegie Mellon University



Examples of digits from  
the MNIST database

- Task T: classifying handwritten digits from images
- Performance measure P : percentage of digits classified correctly
- Training experience E: dataset of images of handwritten digits



# Applications of machine learning

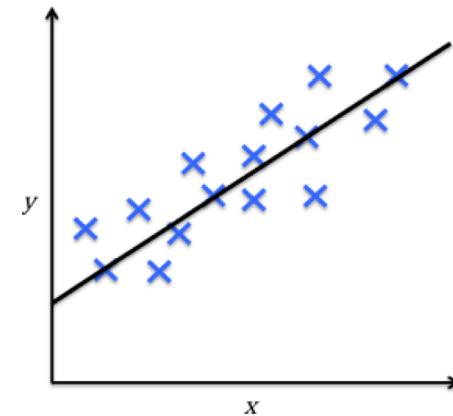
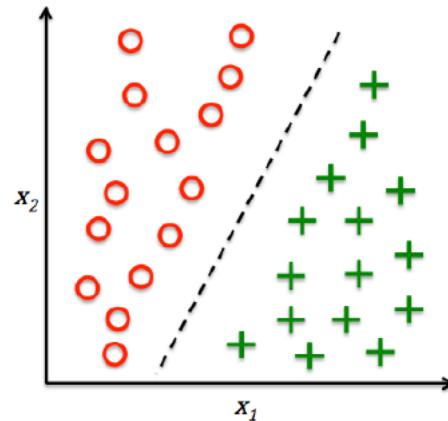
- Email spam detection
- Face detection and matching in smart phones
- Stock predictions
- Product recommendations (e.g. Netflix, amazon)
- Sentiment analysis
- Self-driving cars
- Post office (e.g. sorting letter by post code)
- Medical diagnoses
- Etc.



# Categories of machine learning

- **Supervised learning**

- Labeled data
- Predict outcome/future



Classification  
predict categorical  
class labels  
e.g. the handwritten  
digit (multi-class)

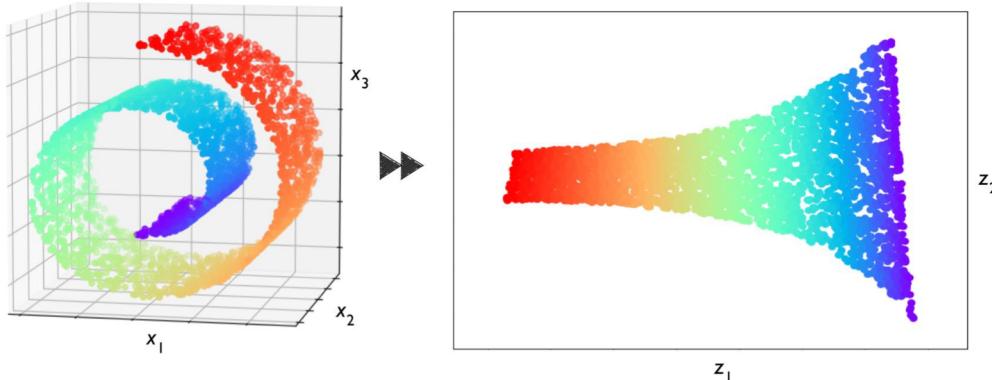
Regression  
Prediction of continuous  
outcomes  
e.g. students' grade scores



# Categories of machine learning

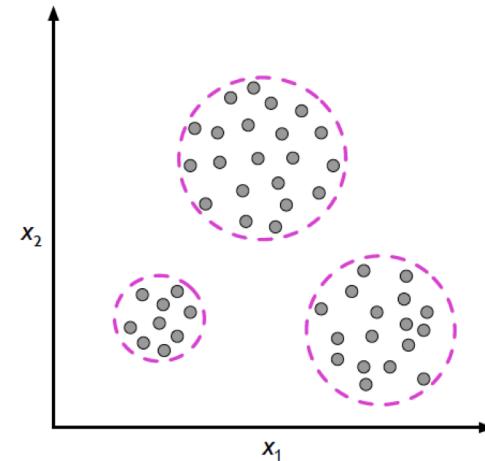
- **Unsupervised learning**

- No labels/targets
- Find hidden structure/insights in data



## Dimensionality Reduction

- reduce data sparsity
- reduce computational cost



## Clustering

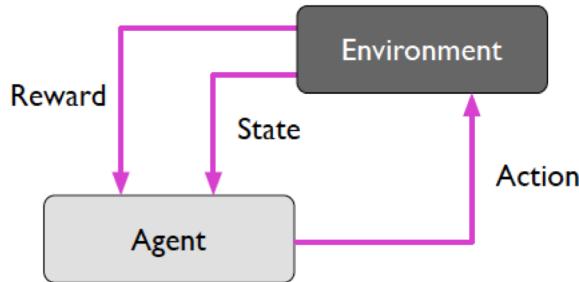
Objectives within a cluster share a degree of similarity.  
e.g. product recommendation



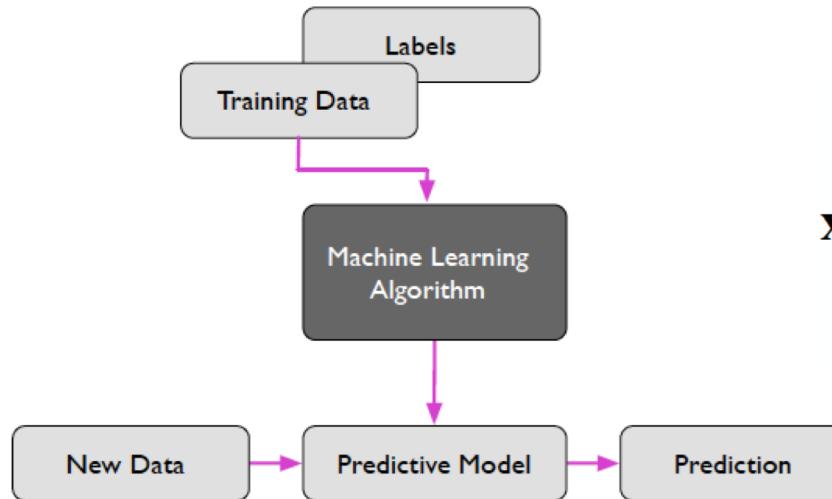
# Categories of machine learning

## ▪ Reinforcement learning

- Decision process
- Reward system
- Learn series of actions
- Applications: chess, video games, some robots, self-driving cars



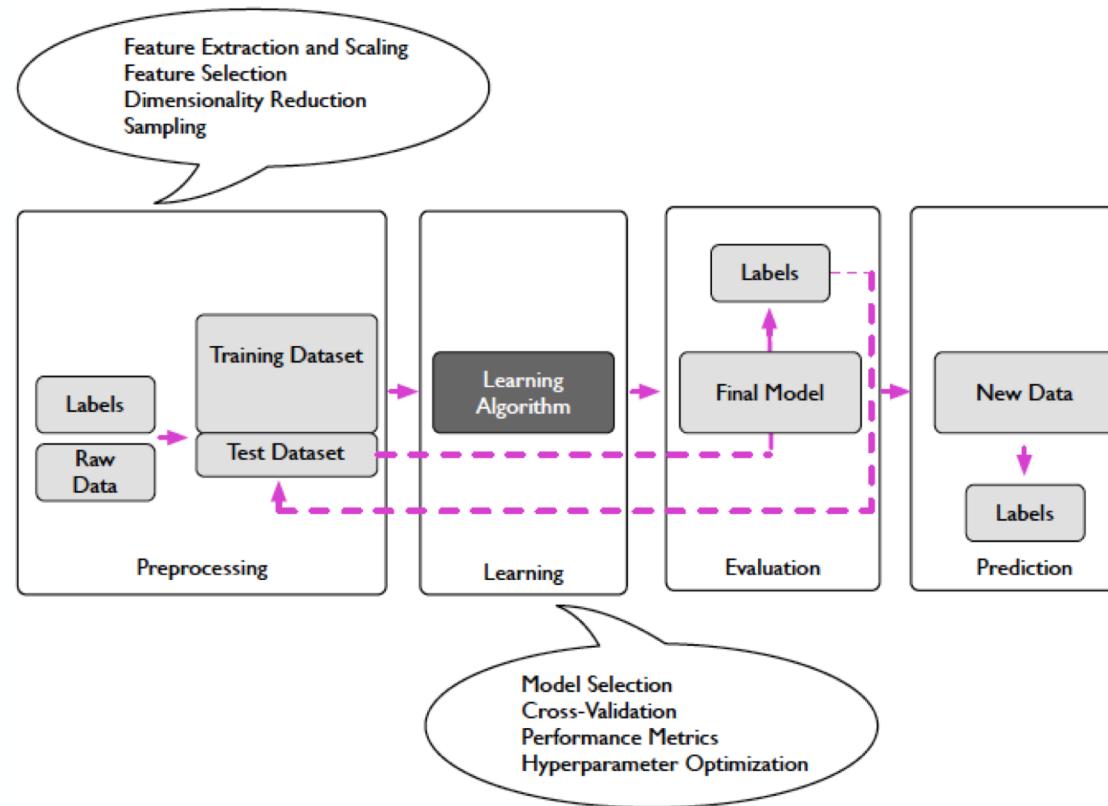
# Supervised learning workflow



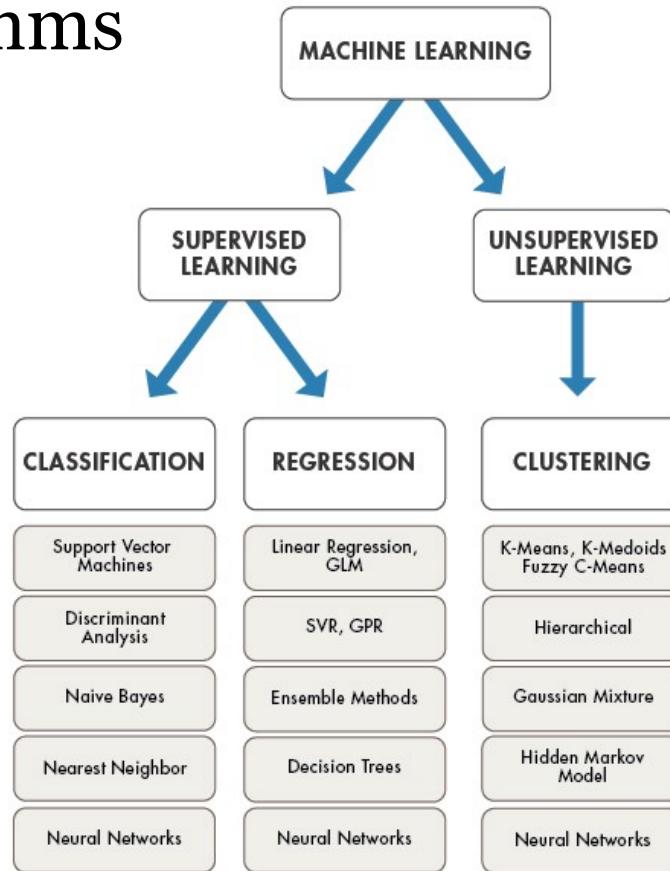
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \xrightarrow{\text{hypothesis function}} \mathbf{y} = \begin{bmatrix} y^{[1]} \\ y^{[2]} \\ \vdots \\ y^{[n]} \end{bmatrix}$$



# Supervised learning workflow



# Some algorithms



# Model evaluation – misclassification error

$$L(\hat{y}, y) = \begin{cases} 0 & \textbf{if } \hat{y} = y \\ 1 & \textbf{if } \hat{y} \neq y \end{cases}$$

$$ERR_{\mathcal{D}_{\text{test}}} = \frac{1}{n} \sum_{i=1}^n L(\hat{y}^{[i]}, y^{[i]})$$



# Model evaluation – other metrics

- Accuracy (1-Error)
- ROC, AUC
- Precision, Recall
- F-measure, G-mean
- (Cross) Entropy
- Likelihood
- Squared Error/MSE
- $R^2$
- etc.



# Summary

- Major concepts of machine learning at a high level.
- Different types of machine learning tasks.
- The major steps of supervised learning: the workflow
- Machine learning algorithms and evaluation



# Learning Resources

- Free machine learning eBooks  
[https://github.com/rasbt/pattern\\_classification/blob/master/resources/machine\\_learning\\_ebooks.md](https://github.com/rasbt/pattern_classification/blob/master/resources/machine_learning_ebooks.md)
- 10-min Video by Kevin Markham:  
<https://www.youtube.com/watch?v=elojMnjn4kk&list=PL5-da3qGB5ICeMbQuqbbCOQWcS6OYBr5A&index=1>  
If you cannot access the video [https://github.com/justmarkham/scikit-learn-videos/blob/master/01\\_machine\\_learning\\_intro.ipynb](https://github.com/justmarkham/scikit-learn-videos/blob/master/01_machine_learning_intro.ipynb)
- Recommended reading:
  - Raschka and Mirjalili: Python Machine Learning, 2nd ed., Ch 1
  - Trevor Hastie, Robert Tibshirani, and Jerome Friedman: The Elements of Statistical Learning, Ch 01  
[https://web.stanford.edu/~hastie/ElemStatLearn/printings/ESLII\\_print12.pdf](https://web.stanford.edu/~hastie/ElemStatLearn/printings/ESLII_print12.pdf)





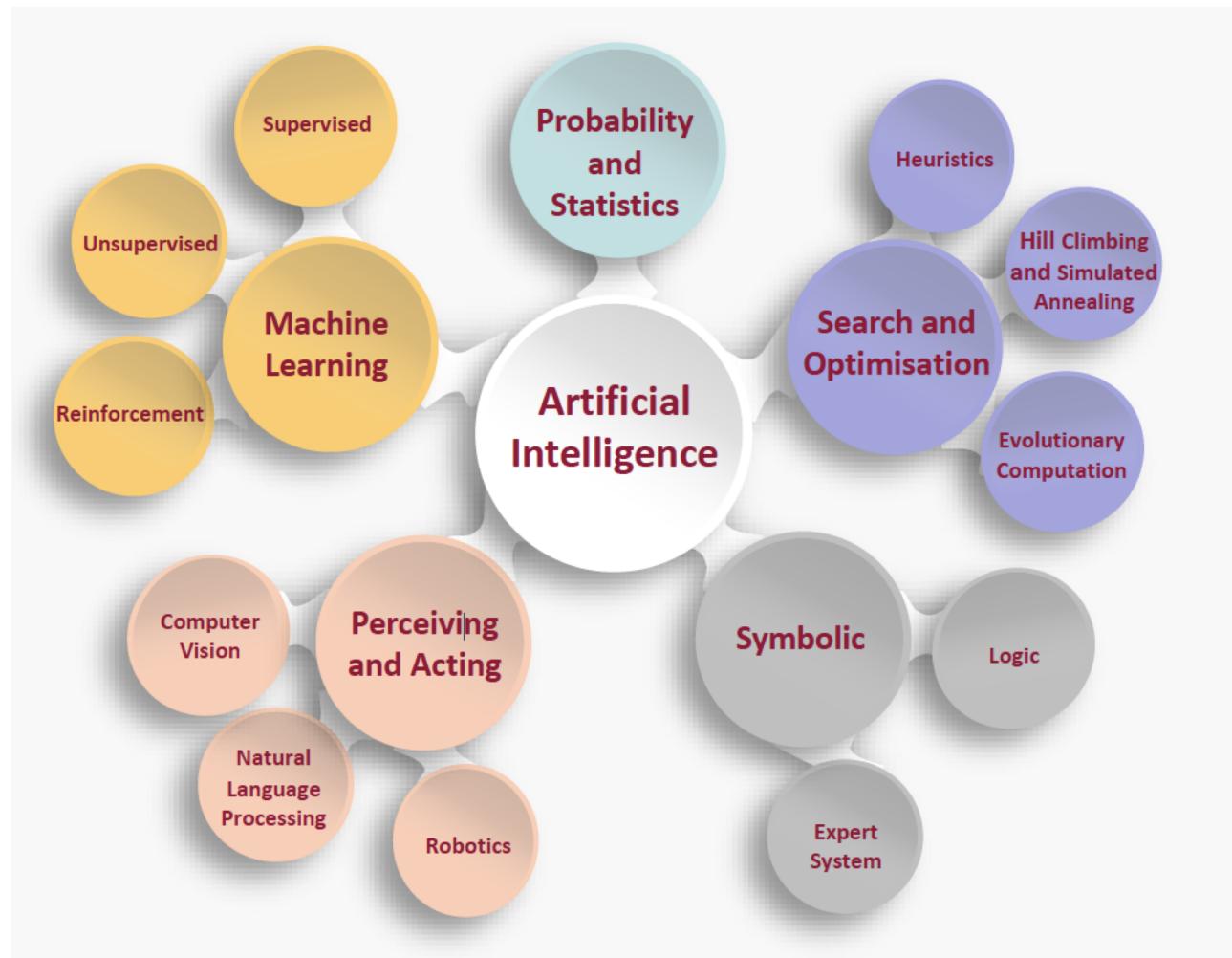
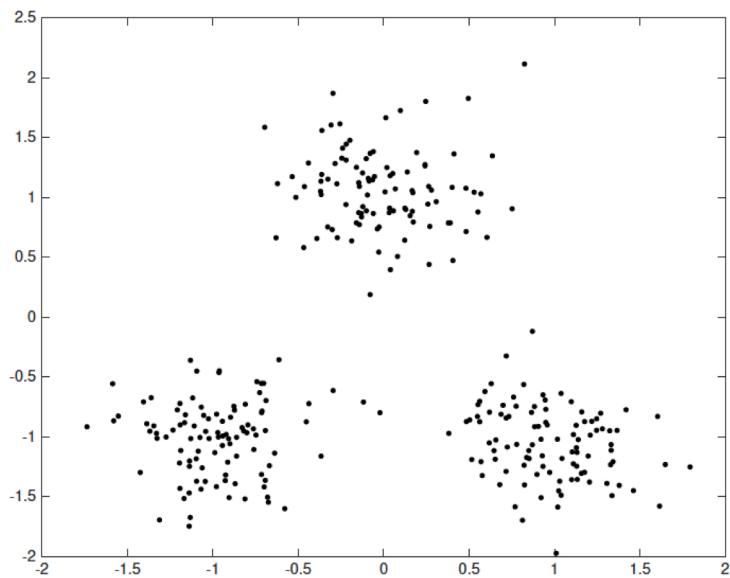
UNIVERSITY OF  
BIRMINGHAM

# Hierarchical Clustering and Expectation-Maximization

**Shuo Wang**



# AI Taxonomy



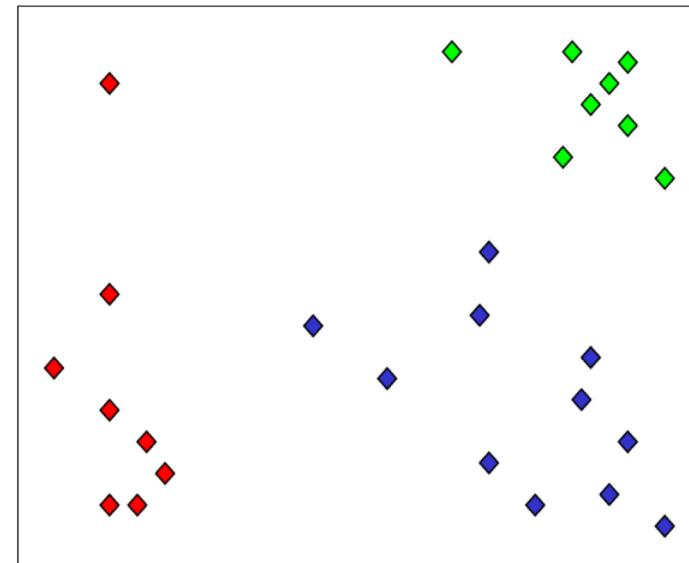
UNIVERSITY OF  
BIRMINGHAM

# Outline

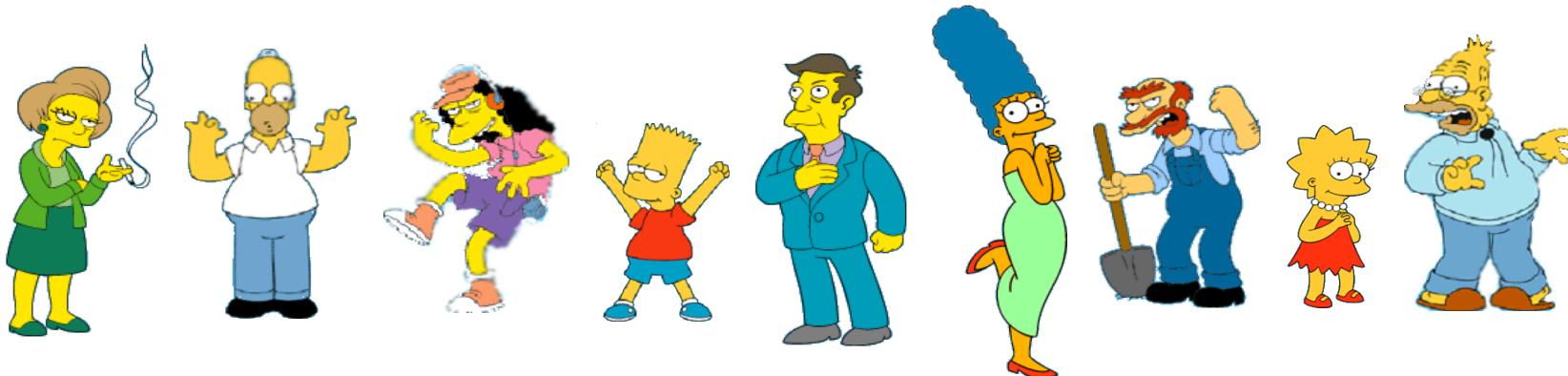
- Clustering concepts
- Hierarchical clustering
- Gaussian mixture models using EM

# Clustering

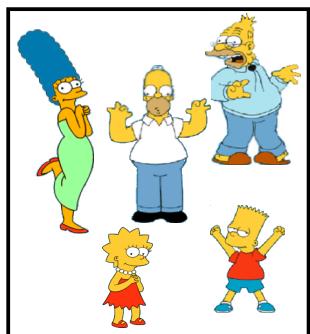
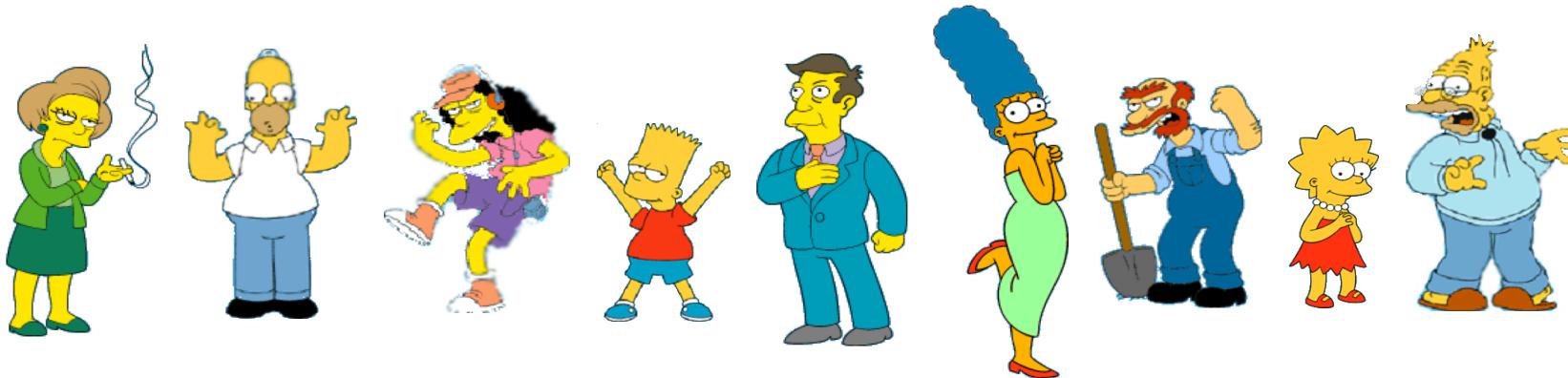
- Segment data into clusters, such that there is
  - high intra-cluster **similarity**
  - low inter-cluster **similarity**
- Informally, finding natural groupings among objects.



# What is the natural grouping of these objects?



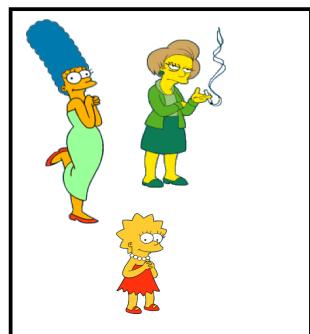
# What is the natural grouping of these objects?



Simpson's family



School employees



Females



Males



UNIVERSITY OF  
BIRMINGHAM

Picture from CMU

# Clustering set-up

- Our data are:  $D = \{x_1, \dots, x_N\}$ .
- Each data point is  $m$ -dimensional, i.e.  
 $x_i = \langle x_{i,1}, \dots, x_{i,m} \rangle$
- Define a *distance function* (*i.e. similarity measures*) between data,  
 $d(x_i, x_j)$
- Goal: segment  $x_n$  into  $k$  groups  
 $\{z_1, \dots, z_N\}$  where  $z_i \in \{1, \dots, K\}$

# Similarity Measures

- Between any two data samples  $p$  and  $q$ , we can calculate their distance  $d(p,q)$  using a number of measurements:

Euclidean

$$d(p, q) = d(q, p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

Manhattan

$$d(p, q) = \sum_{i=1}^n |p_i - q_i|$$

Chebyshev

$$d_{\text{Chebyshev}}(p, q) = \max_i(|p_i - q_i|)$$

Minkowski

$$d(p, q) = \left( \sum_{i=1}^n |p_i - q_i|^b \right)^{1/b}$$



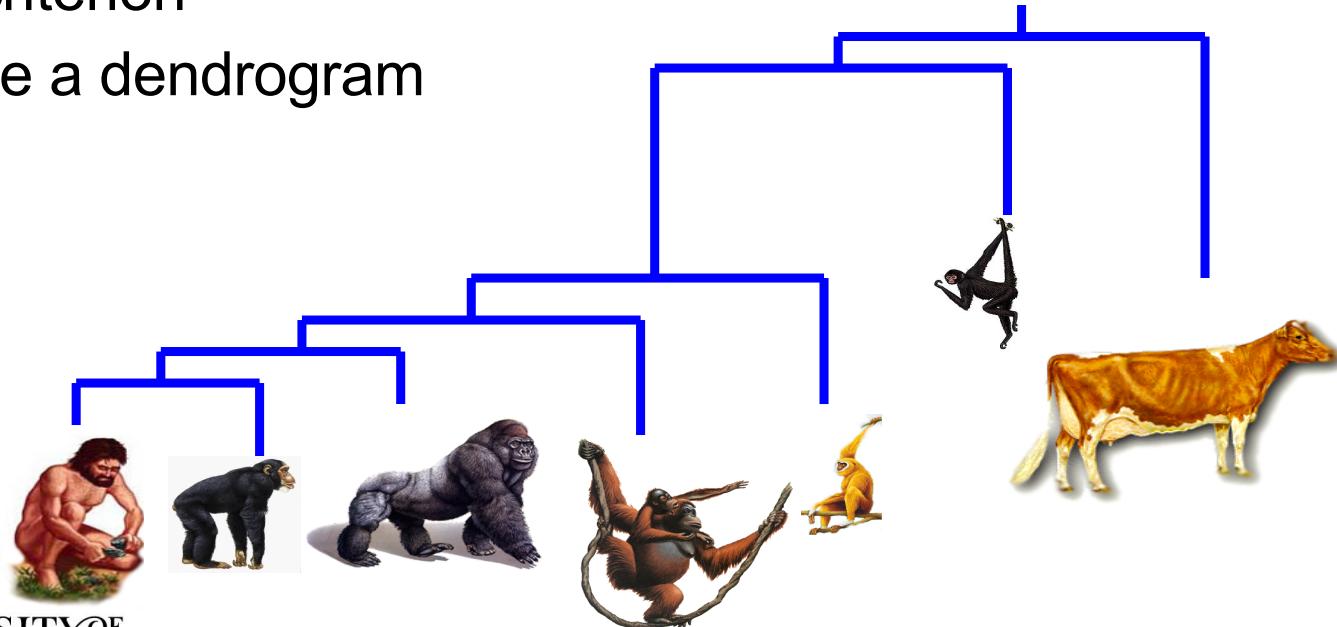
UNIVERS  
BIRMINGHAM

# Types of Clustering Algorithms

- Partitional clustering, e.g. K-means, K-medoids
- Hierarchical clustering
  - Bottom-up (agglomerative)
  - Top-down
- Density-based clustering, e.g. DBScan
- Mixture density based clustering
- Fuzzy theory based, graph theory based, grid based, etc.

# Hierarchical clustering

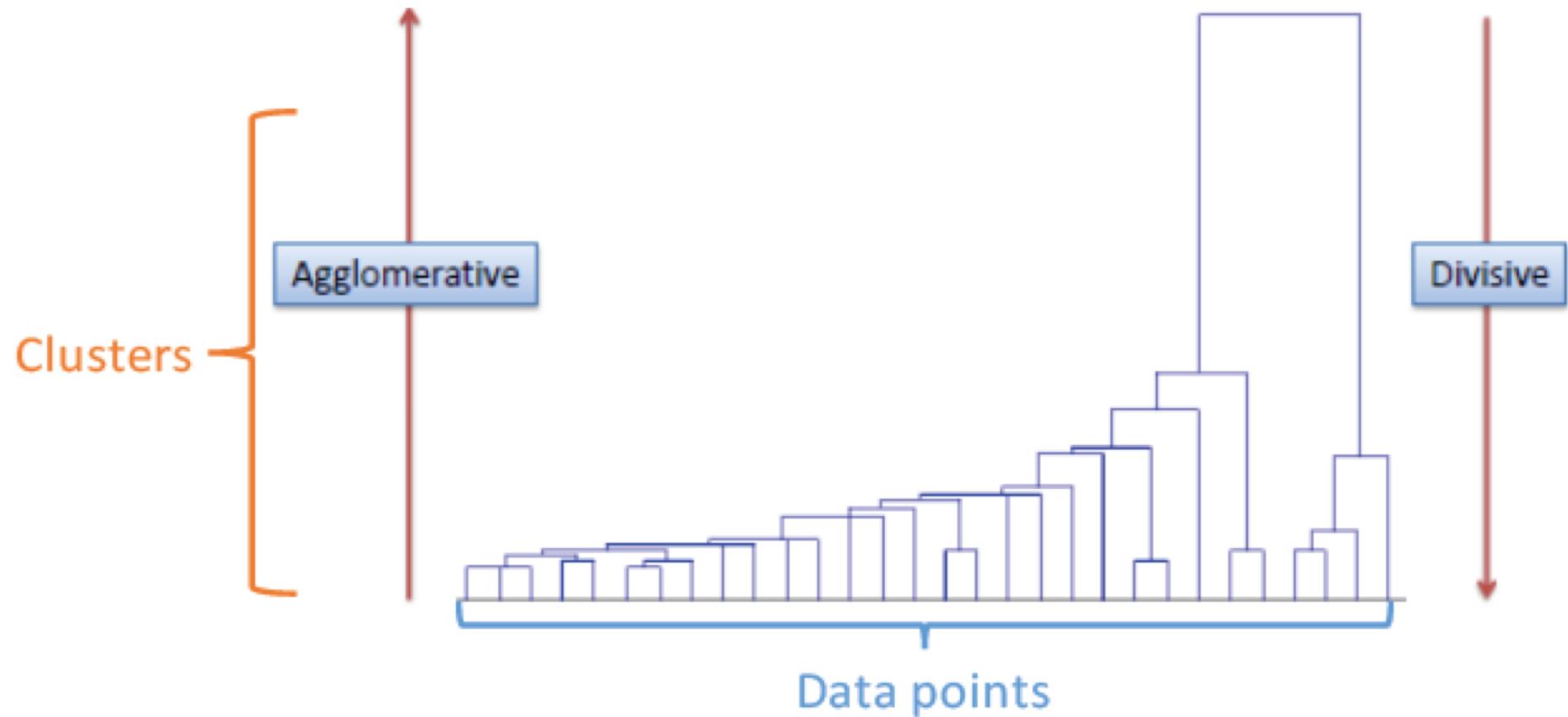
- Create a hierarchical decomposition of the set of objects using some criterion
- Produce a dendrogram



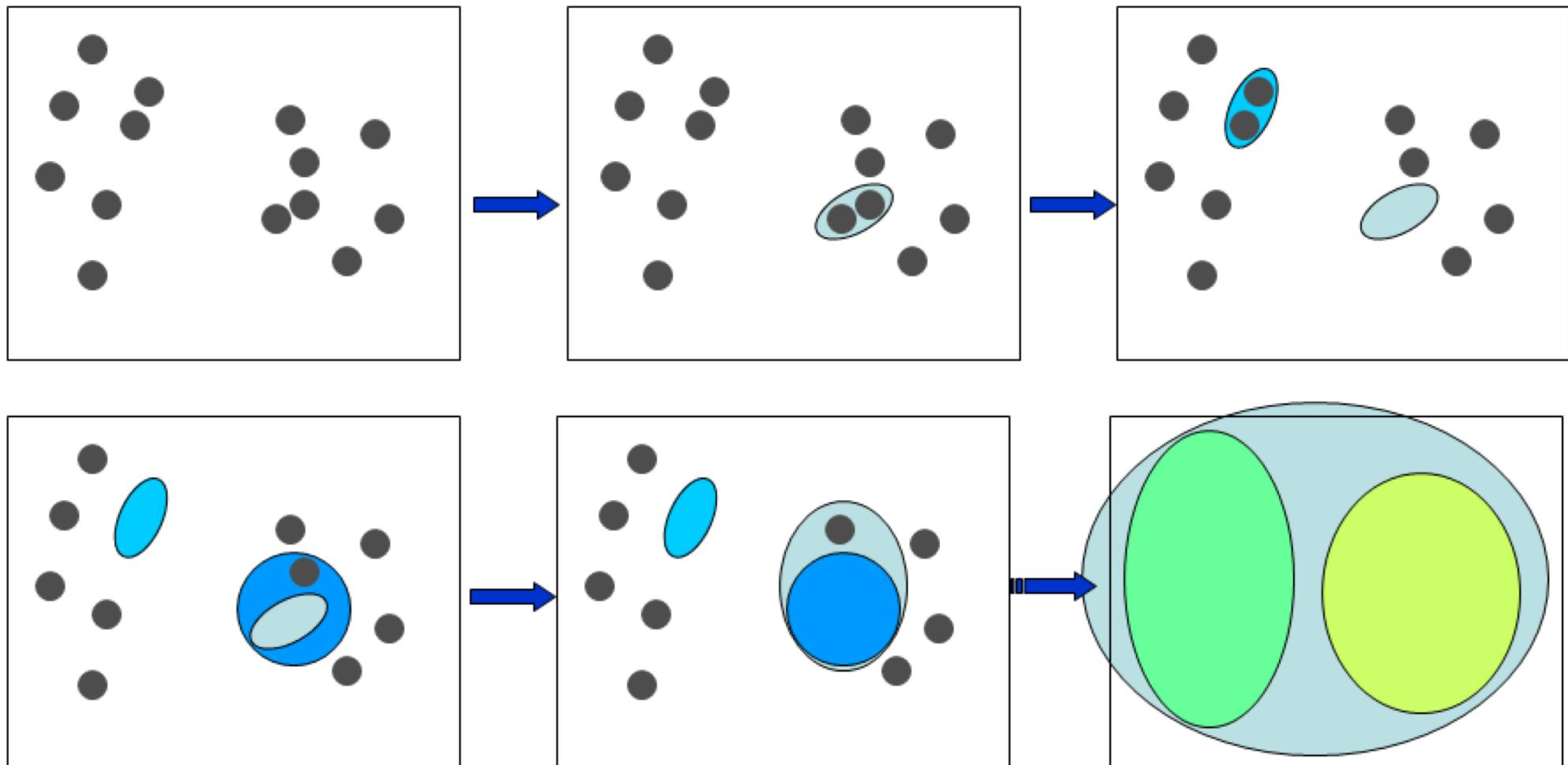
UNIVERSITY OF  
BIRMINGHAM

(Bovine, (Spider Monkey, (Gibbon, (Orang, (Gorilla, (Chimp, Human))))));

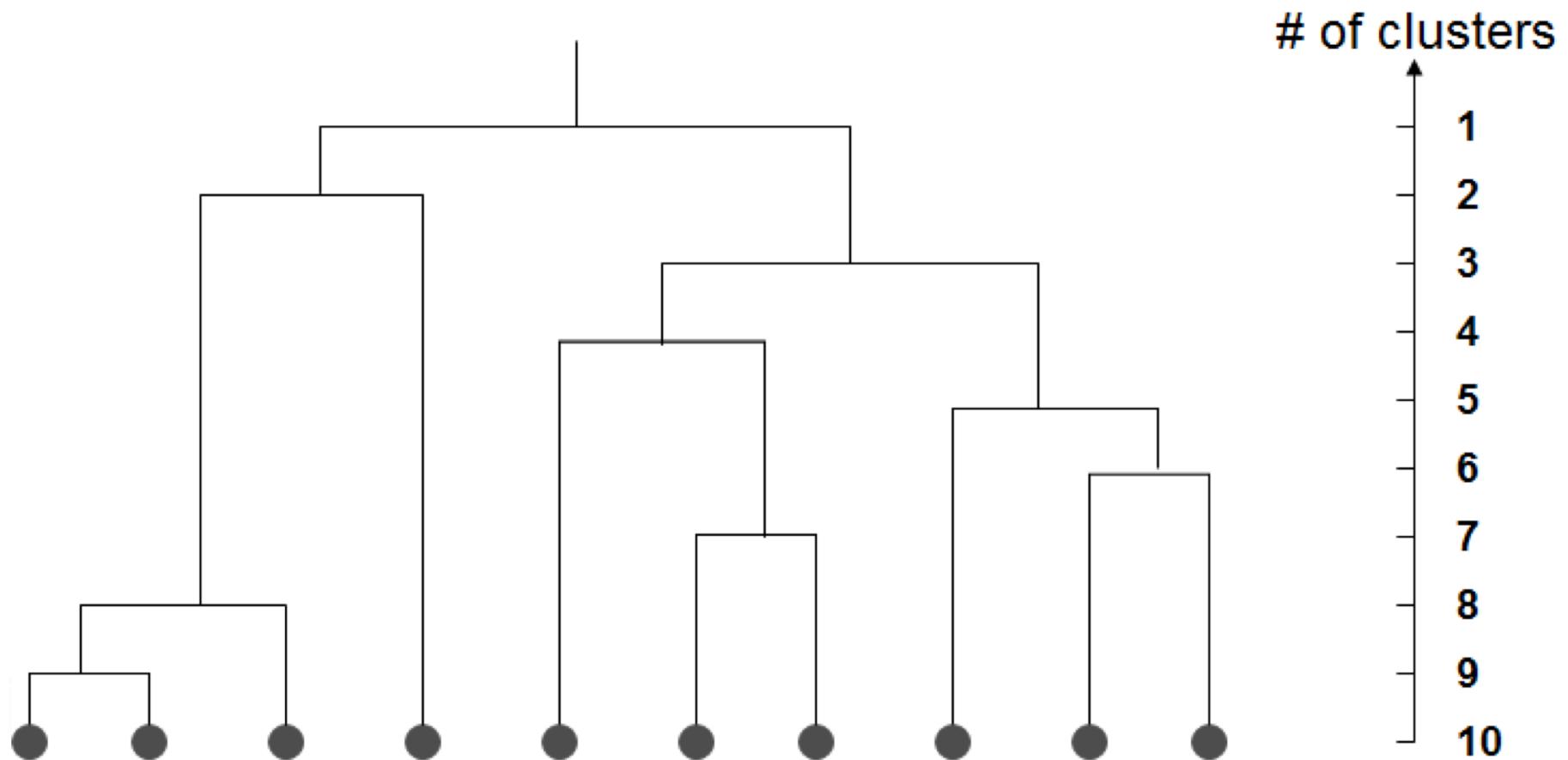
# Hierarchical clustering



# Agglomerative clustering illustration



# Dendrogram: A visual representation of output

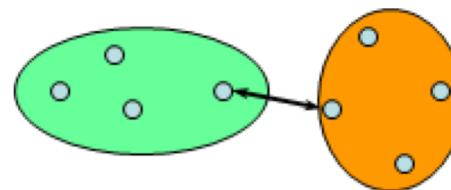


# Agglomerative clustering algorithm

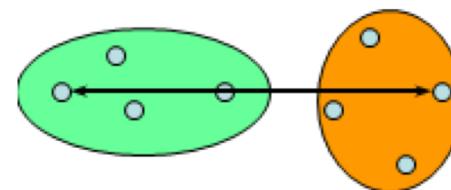
1. Place each data point into its own singleton group
  2. Repeat: iteratively merge the two closest groups
  3. Until: all the data are merged into a single cluster
- 
- Output: a dendrogram
  - Reply on: a *distance metric* between **clusters**

# Measuring distance between clusters

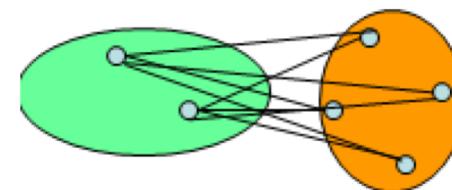
- **Single linkage**  
the similarity of the closest pair
- **Complete linkage**  
the similarity of the furthest pair
- **Group average**  
the average similarity of all pairs  
more widely used  
robust against noise



$$d_{SL}(G, H) = \min_{i \in G, j \in H} d_{i,j}$$



$$d_{CL}(G, H) = \max_{i \in G, j \in H} d_{i,j}$$



$$d_{GA} = \frac{1}{N_G N_H} \sum_{i \in G} \sum_{j \in H} d_{i,j}$$



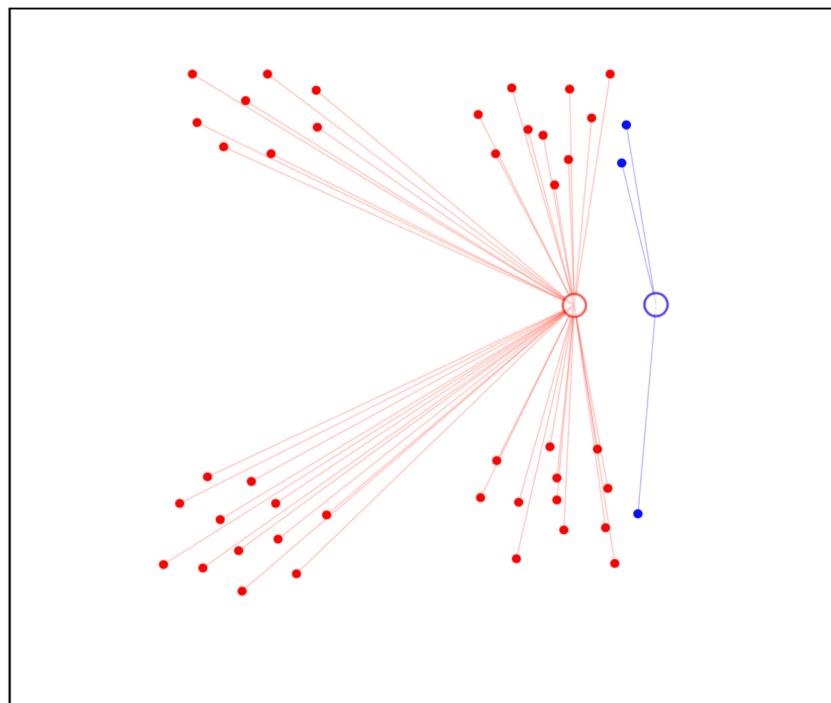
# Strengths, weaknesses, caveats

- Strengths
  - provides deterministic results
  - no need to specify number of clusters beforehand
  - can create clusters of arbitrary shapes
- Weakness
  - does not scale up for large datasets, time complexity at least  $O(n^2)$
- Caveats
  - Different decisions about group similarities can lead to vastly different dendograms.
  - The algorithm imposes a hierarchical structure on the data, even data for which such structure is not appropriate.

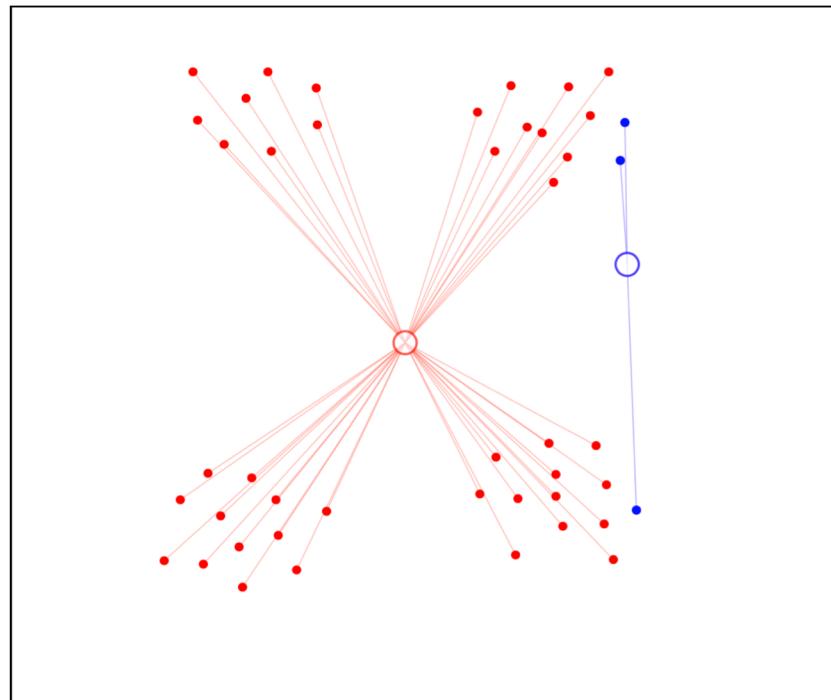
# K-means

- Centroid-based: describe each cluster by its mean
- Goal: assign data to K.
- Algorithm objective: minimize the within-cluster variances of all clusters.

# Initialize 2 clusters and assign points to clusters

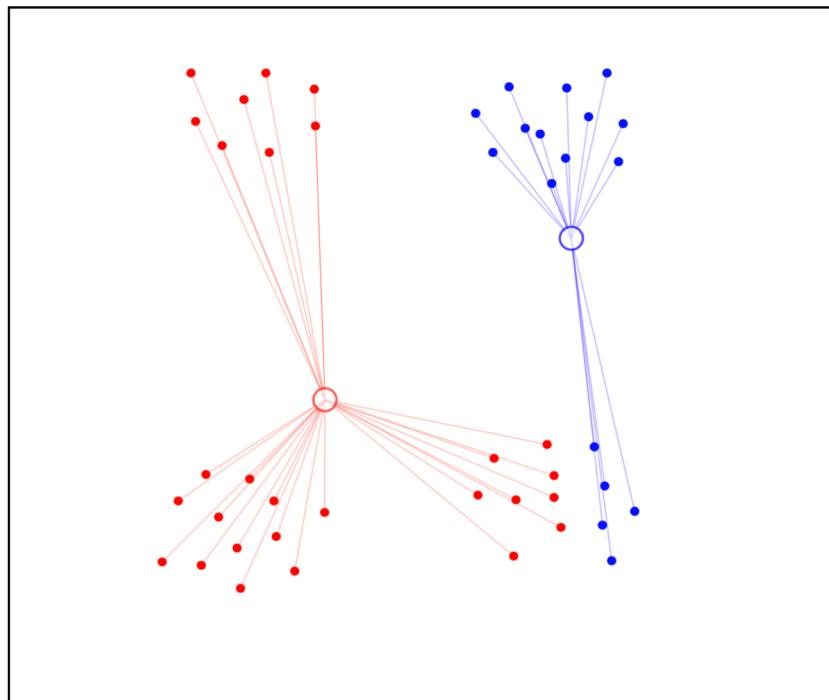


# Adjust mean

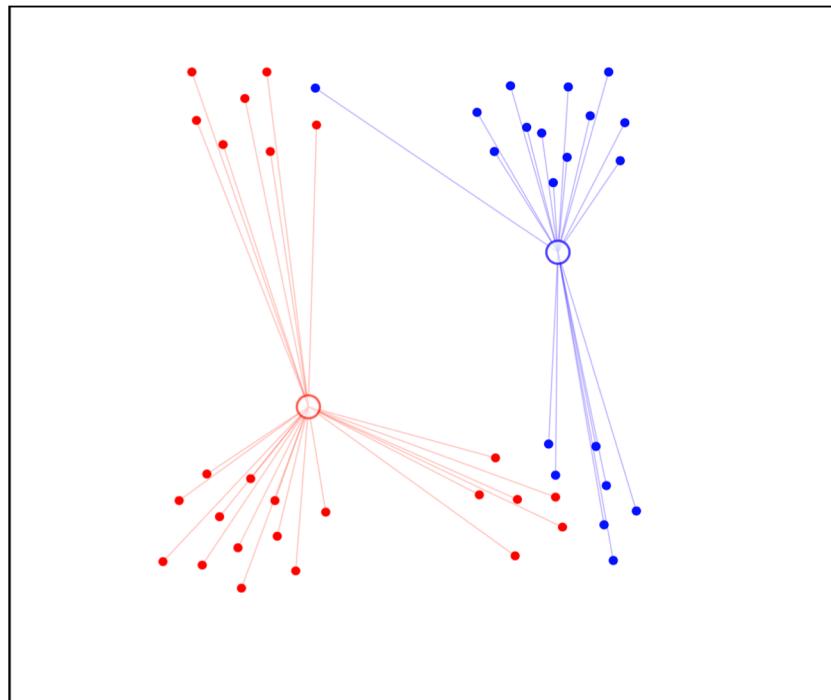


UNIVERSITY OF  
BIRMINGHAM

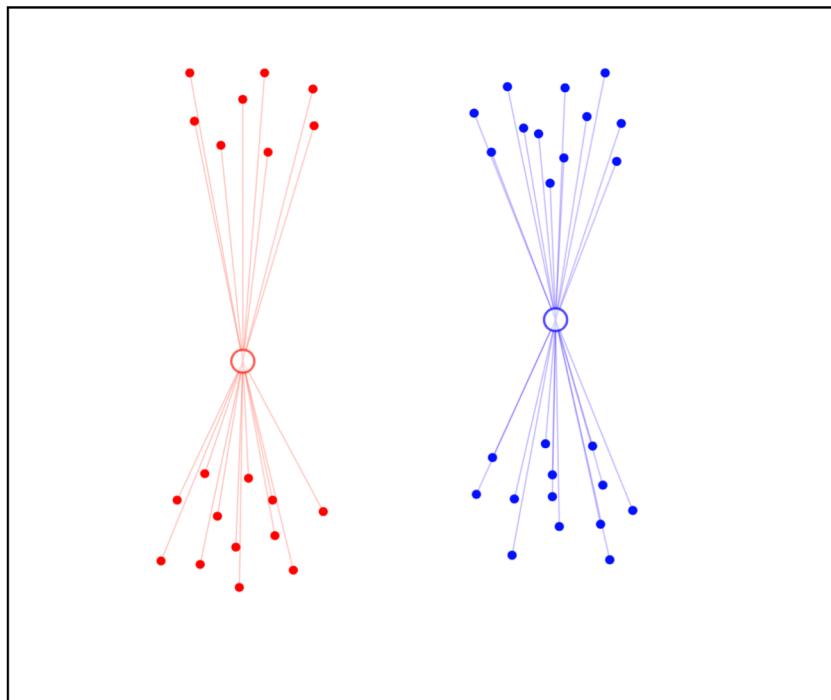
# Reassign points to clusters and adjust mean



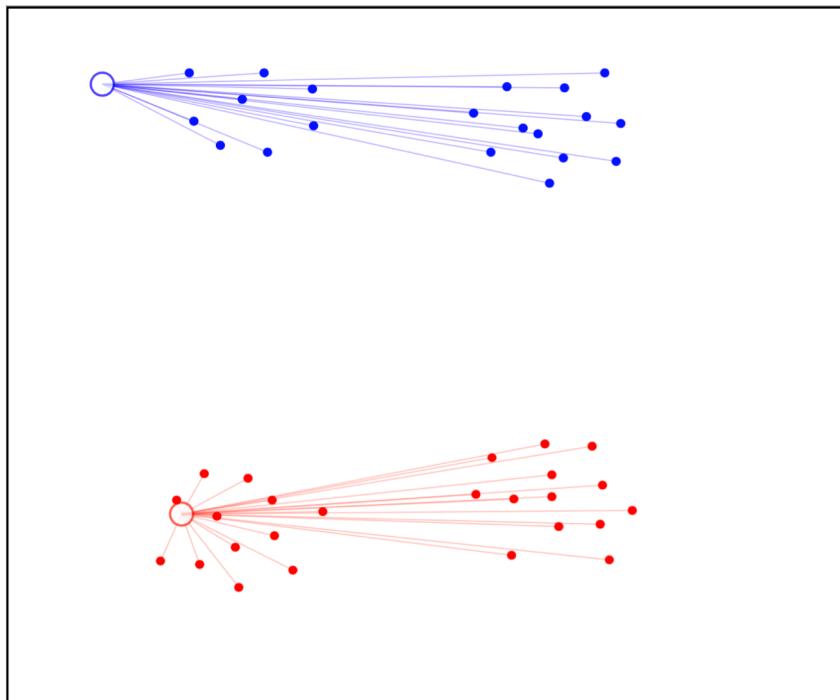
# Reassign points to clusters and adjust mean



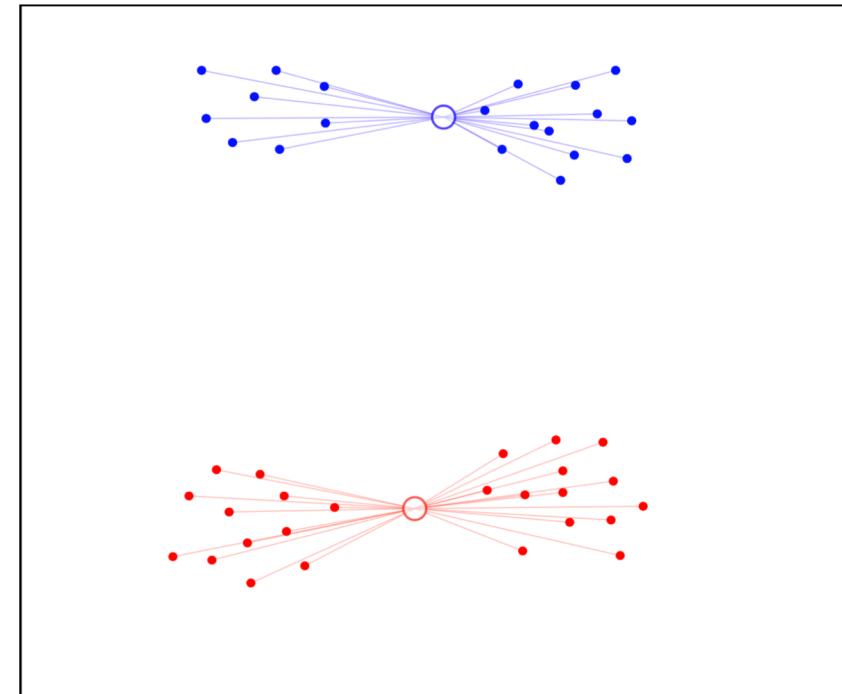
# Repeat this, until no cluster changes



# If we have a different starting point



Initial clusters



Final clusters



UNIVERSITY OF  
BIRMINGHAM

# K-means

- A non-deterministic method
- Finds a local optimal result (multiple restarts are often necessary)

# Algorithm description

## ① Initialization

- Data are  $\mathbf{x}_{1:N}$
- Choose initial cluster means  $\mathbf{m}_{1:k}$  (same dimension as data).

## ② Repeat

### ① Assign each data point to its closest mean

$$z_n = \arg \min_{i \in \{1, \dots, k\}} d(\mathbf{x}_n, \mathbf{m}_i) \quad \longrightarrow \quad d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_i - q_i)^2 + \dots + (p_n - q_n)^2}.$$

Euclidean distance

### ② Compute each cluster mean to be the coordinate-wise average over data points assigned to that cluster,

$$\mathbf{m}_k = \frac{1}{N_k} \sum_{\{n : z_n = k\}} \mathbf{x}_n \quad \longrightarrow \quad \begin{aligned} &\text{For each dimension } j \text{ of } \mathbf{x}_i \text{ in cluster } k: \\ &(\sum_i x_{i,j}) / N_k \end{aligned}$$

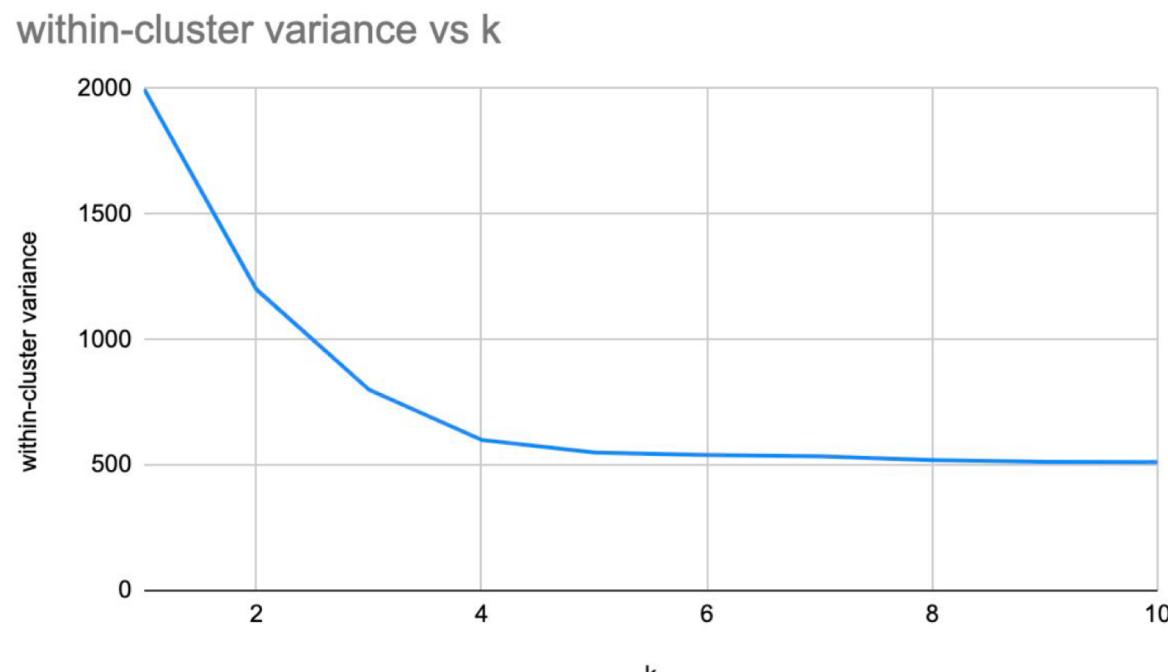
### ③ Until assignments $\mathbf{z}_{1:N}$ do not change



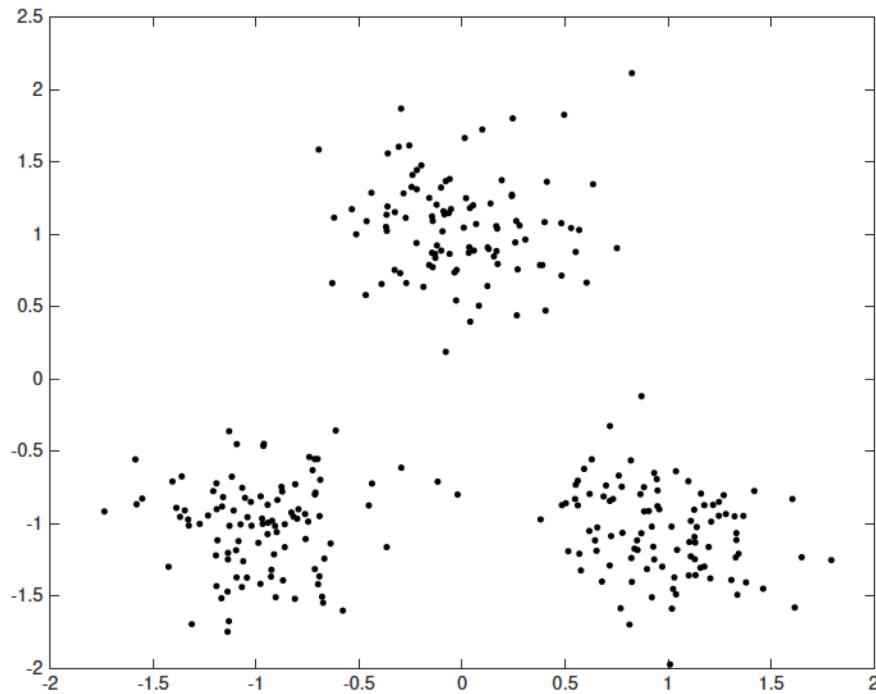
UNIVERSITY OF  
BIRMINGHAM

# K-means: finding optimal k

- Plot the cost for each k and find the “Elbow”



# Probabilistic modelling



How could have this data been generated?

Can you fit models to the data?

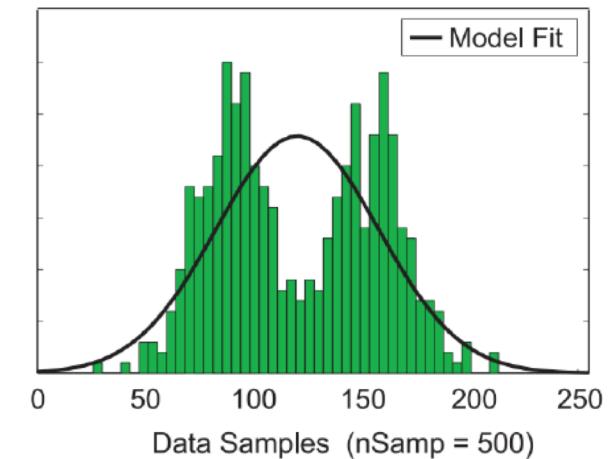
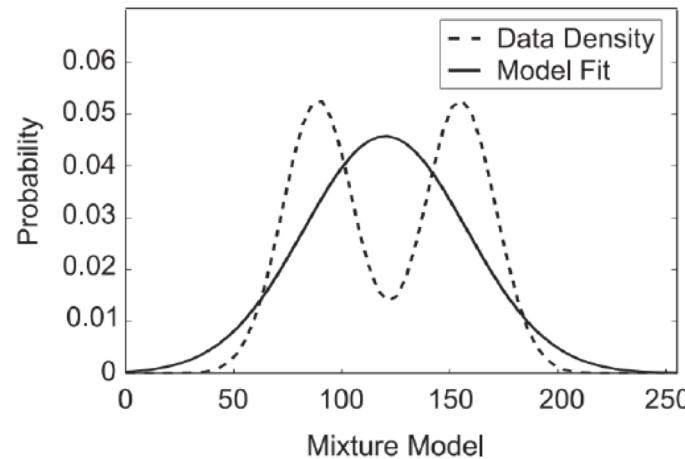


# Gaussian mixture models (GMMs)

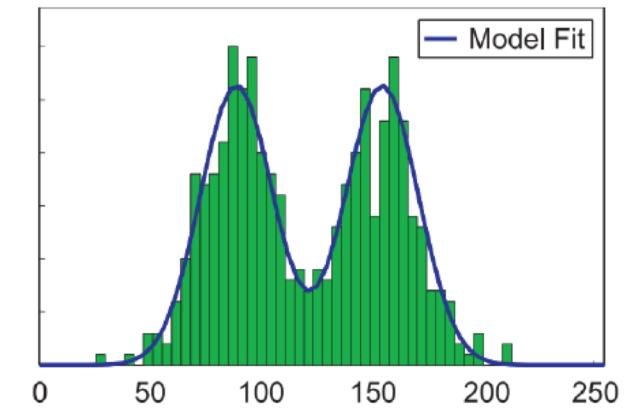
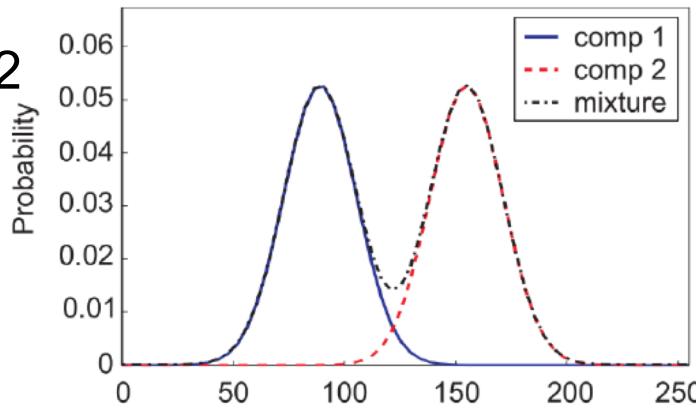
- Assume data was generated by a set of Gaussian distributions.
- The probability density is a mixture of them.
- Find the parameters of the Gaussian distributions and how much each distribution contributes to the data.
- This is a **mixture model of Gaussian**.

# Visualizing GMMs - 1D Gaussians

If you fit one Gaussian



Now we try a GMM with 2 Gaussians  
(each contribute 50%)



UNIVERSITY OF  
BIRMINGHAM

# Generative Models

- In supervised learning, we model the joint distribution

$$p(\mathbf{x}, z) = p(\mathbf{x}|z)p(z)$$

- In unsupervised learning, we do not have labels  $z$ , we model

$$p(\mathbf{x}) = \sum_z p(\mathbf{x}, z) = \sum_z p(\mathbf{x}|z)p(z)$$



Hidden/Latent variables

# Gaussian mixture models (GMMs)

- A GMM represents a distributions as

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x} | \mu_k, \Sigma_k)$$

- with  $\pi_k$  the mixing coefficients, where

$$\sum_{k=1}^K \pi_k = 1 \quad \text{and} \quad \pi_k \geq 0 \quad \forall k$$

- GMM is a density estimator.
- GMM is universal approximators of densities (if you have enough Gaussians)



# Fitting GMMs: Maximum likelihood and EM

- To have a model best fit data, we need to maximize the (log) likelihood

$$\ln p(\mathbf{X}|\pi, \mu, \Sigma) = \sum_{n=1}^N \ln \left( \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}^{(n)} | \mu_k, \Sigma_k) \right)$$

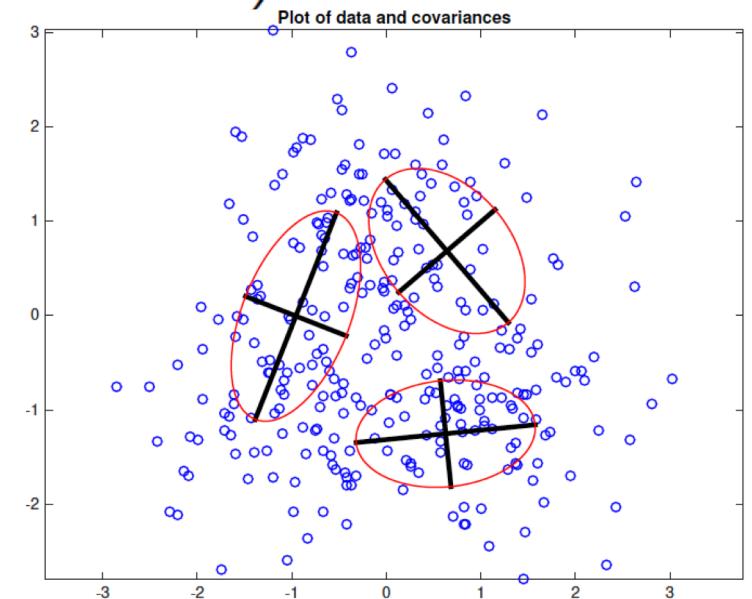
- Expectation**

if we knew  $\pi_k$ ,  $\mu$  and  $\Sigma$ , we can get “soft”  $Z_k$

$P(z^{(n)}|x)$  - **responsibility**

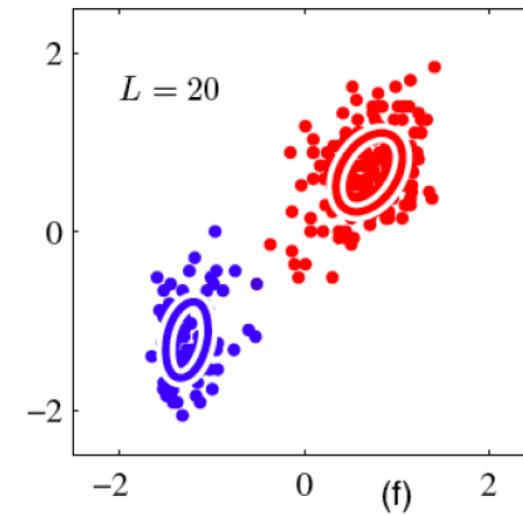
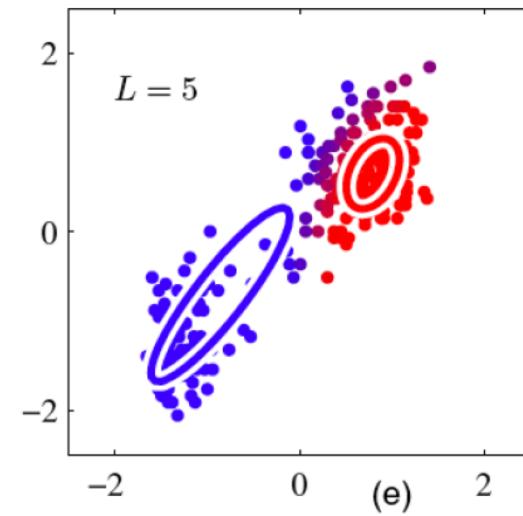
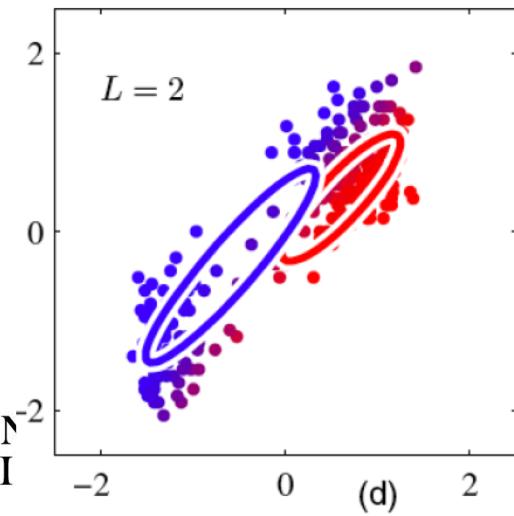
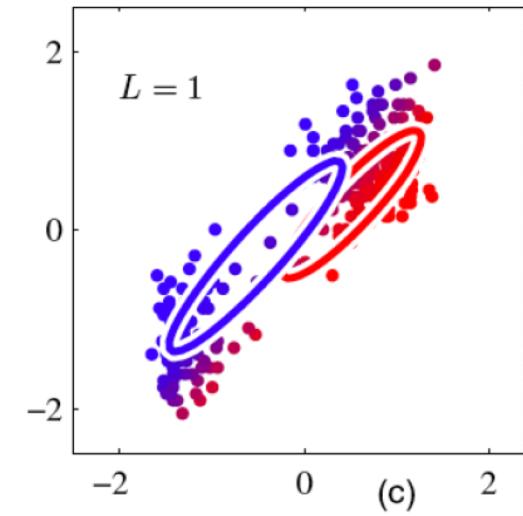
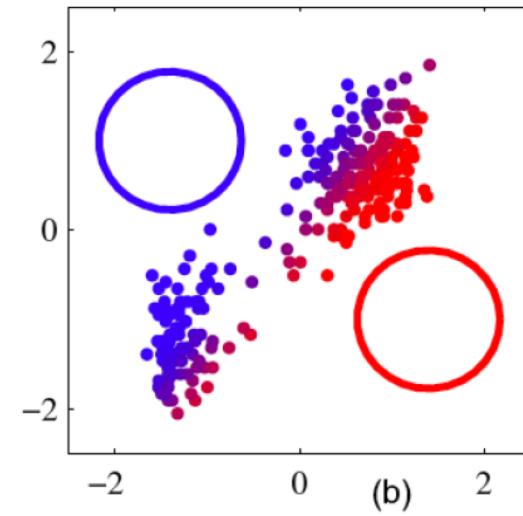
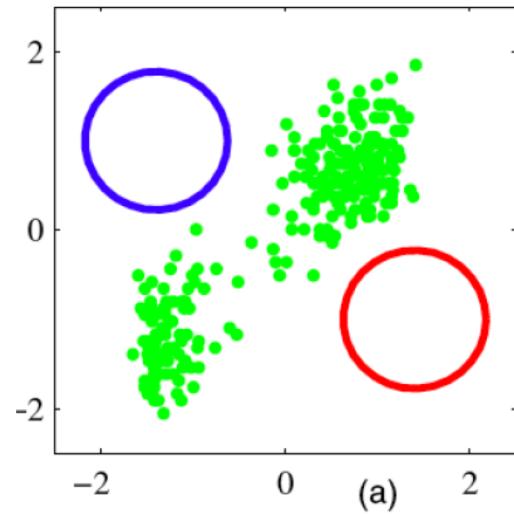
- Maximization**

if we know  $Z_k$ , we can get  $\pi_k$ ,  $\mu$  and  $\Sigma$





UN  
BI



# Expectation-Maximization (EM Algorithm)

An optimization process that alternates between 2 steps:

1. E-step: compute the posterior probability over z given the current model.

$$\gamma_k^{(n)} = p(z^{(n)}|\mathbf{x}) = \frac{\pi_k \mathcal{N}(\mathbf{x}^{(n)}|\mu_k, \Sigma_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}^{(n)}|\mu_j, \Sigma_j)} \quad \text{responsibility}$$

- Which Gaussian generate each data point with how much possibility?

# Expectation-Maximization (EM Algorithm)

2. M-step: Assuming data was really generated this way, change the parameters of each Gaussian to maximize the probability that it would generate the data it is currently responsible for.

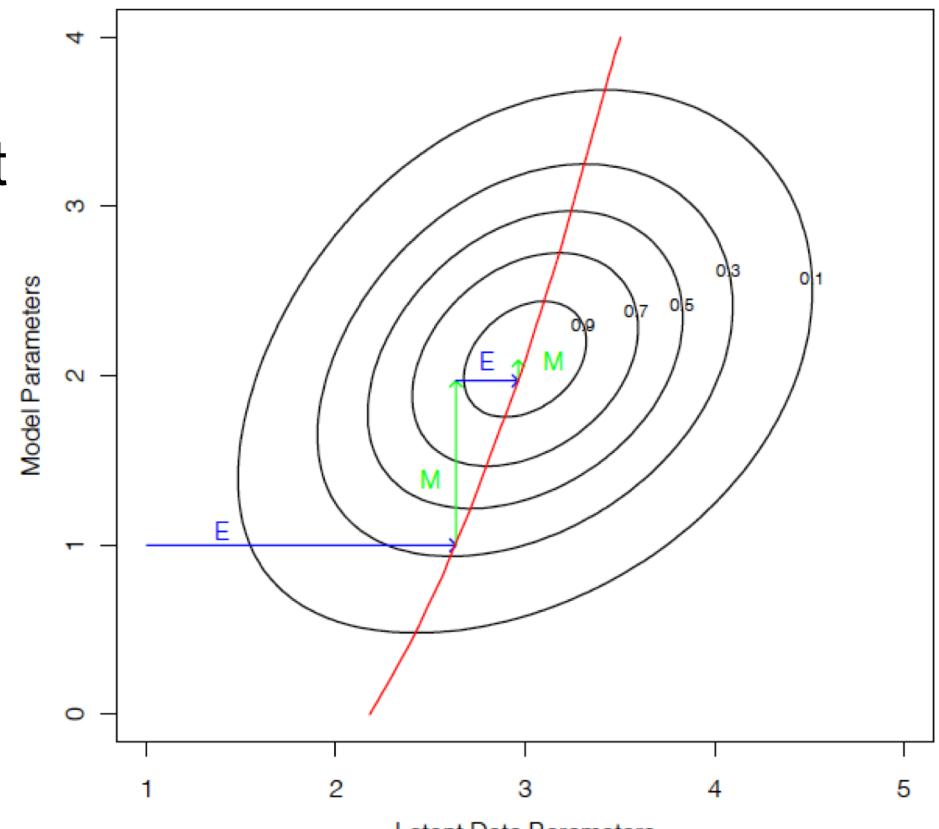
$$\mu_k = \frac{1}{N_k} \sum_{n=1}^N \gamma_k^{(n)} \mathbf{x}^{(n)}$$

$$\Sigma_k = \frac{1}{N_k} \sum_{n=1}^N \gamma_k^{(n)} (\mathbf{x}^{(n)} - \mu_k)(\mathbf{x}^{(n)} - \mu_k)^T$$

$$\pi_k = \frac{N_k}{N} \quad \text{with} \quad N_k = \sum_{n=1}^N \gamma_k^{(n)}$$

# Expectation-Maximization (EM Algorithm)

- A general algorithm for optimizing many latent variable models (not just for GMMs).
- Iteratively computes a lower bound then optimizes it.
- Converges but maybe to a local minima.
- Can use multiple restarts.



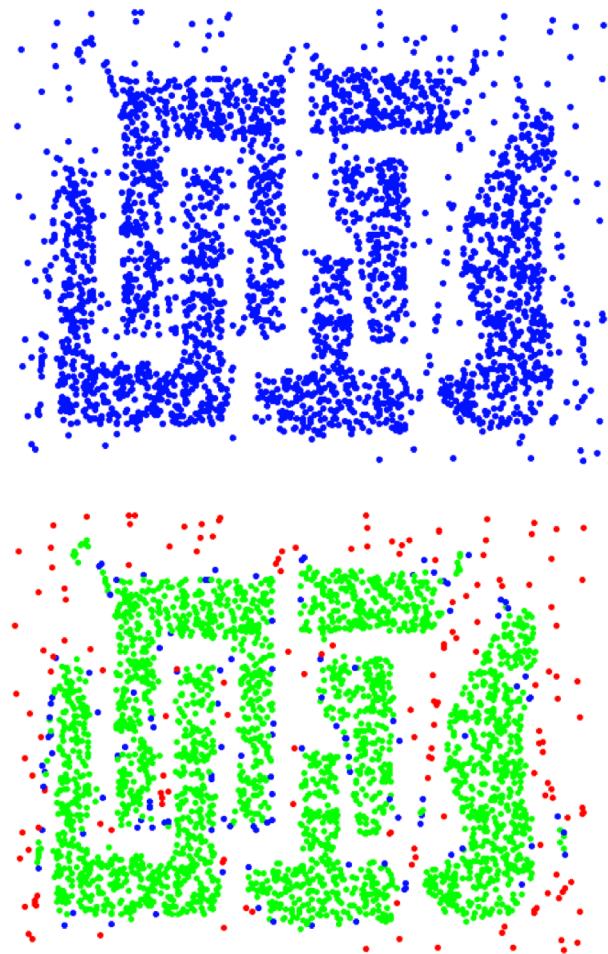
# Summary

- **Clustering**
  - group similar data points
  - need a distance measure
- **Agglomerative hierarchical clustering**
  - successively merges similar groups of points
  - build a dendrogram (binary tree)
  - different ways to measure distance between clusters.
- **GMM using EM**
  - build a generative model based on Gaussian distributions
  - need to pre-define k (number of clusters)
  - Using EM to find the best fit of the model.



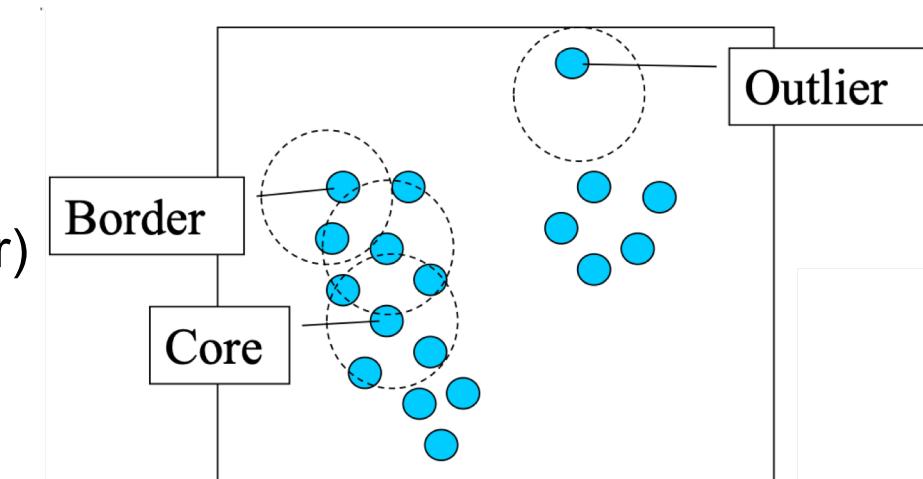
# Density-based Clustering - DBSCAN

- Acronym for: Density-based spatial clustering of applications with noise
- Clusters are dense regions in the data space separated by regions of lower sample density.
- A cluster is defined as a maximal set of density connected points.
- Discover clusters of arbitrary shape.



# Questions

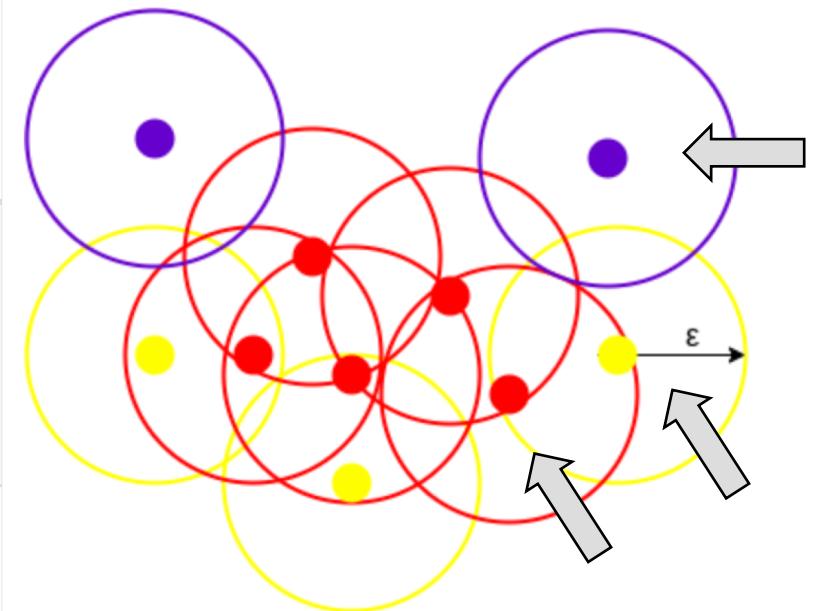
- What is a dense region?
- How do we measure density?
- Define **three exclusive types of points**
  - Core, Border (or Edge) and Noise (or outlier)
  - Core points -- dense region
  - Noise -- sparse region
- Need **two parameters**
  - 1) a circle of epsilon radius
  - 2) a circle containing at least minPts number of points



$$\epsilon = 1 \text{ unit}, \text{MinPts} = 5$$

# Three types of points

core	The point has at least minPts number of points within Eps
border	The point has fewer than minPts within Eps, but is in the neighbourhood (i.e. circle) of a core point.
noise	Any point that is not a core point or a border point.



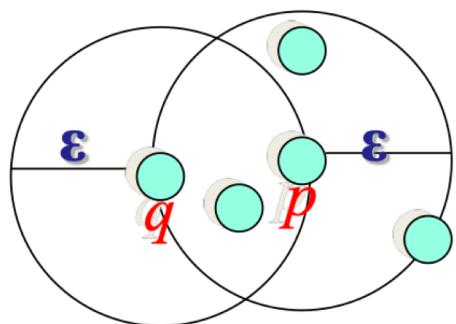
minPts = 3  
red: core  
yellow: border  
purple: noise



# How to form core points into clusters?

## -- Density-reachability

- Directly density-reachable: a point  $q$  is directly density-reachable from point  $p$  if  $p$  is a core point and  $q$  is in  $p$ 's neighbourhood.



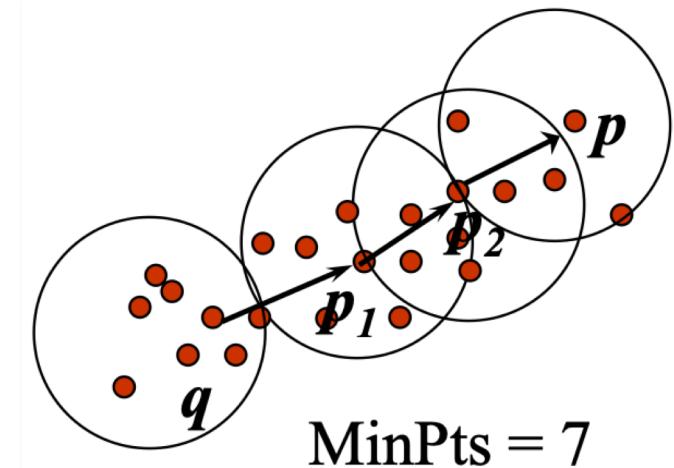
- $q$  is directly density-reachable from  $p$
- $p$  is not necessarily directly density-reachable from  $q$
- Density-reachability is asymmetric.

MinPts = 4

# How to form core points into clusters?

## -- Density-reachability

- Density-Reachable (directly and indirectly)
  - ❖ A point  $p$  is directly density-reachable from  $p_2$
  - ❖  $p_2$  is directly density-reachable from  $p_1$
  - ❖  $p_1$  is directly density-reachable from  $q$
  - ❖  $q \rightarrow p_1 \rightarrow p_2 \rightarrow p$  form a chain  
( $p$  is the border)



- $p$  is indirectly density-reachable from  $q$
- $q$  is not density-reachable from  $p$

# The algorithm

- 1. Label all points as core, border or noise.
- 2. Eliminate noise points.
- 3. For every core point  $p$  that has not been assigned to a cluster:
  - ❖ Create a new cluster with the point  $p$  and all the points that are density-reachable from  $p$
- 4. For border points belonging to more than 1 cluster, assign it to the cluster of the closest core point.

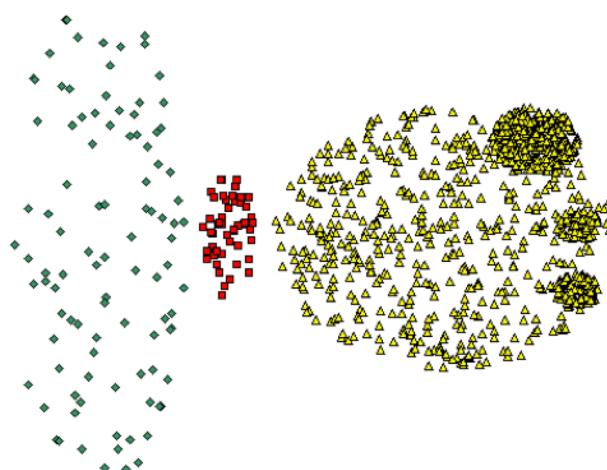
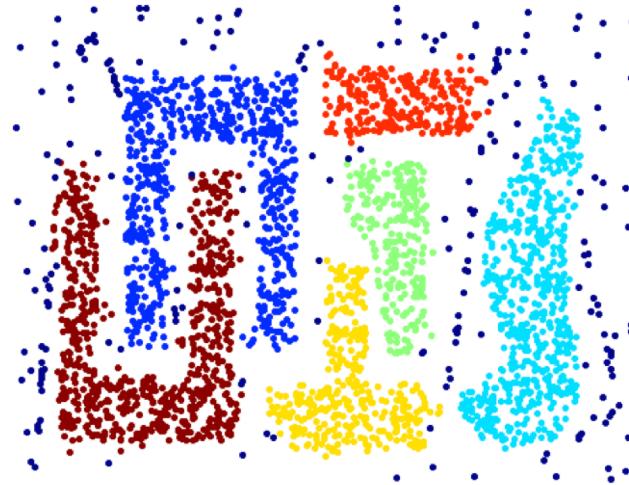


UNIVERSITY OF  
BIRMINGHAM

The distance measure  
could be Euclidean or  
others.

# Some key points

- DBSCAN can find non-linearly separable clusters. (an advantage over K-means and GMM)
- Resistant to noise
- Not entirely deterministic: border points that are reachable from more than one cluster can be part of either cluster, depending on the implementation.

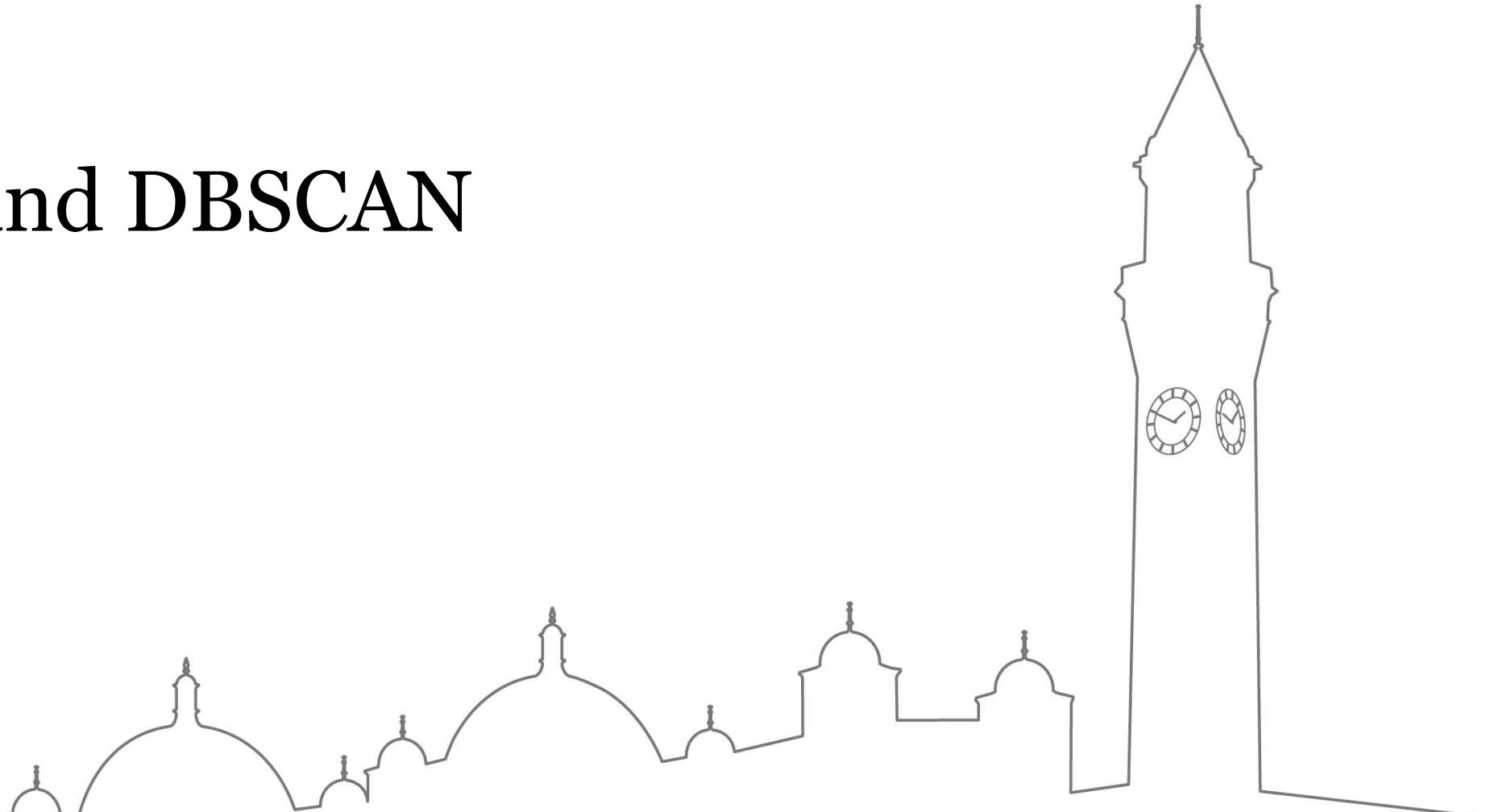




UNIVERSITY OF  
BIRMINGHAM

# Clustering GMM/EM and DBSCAN

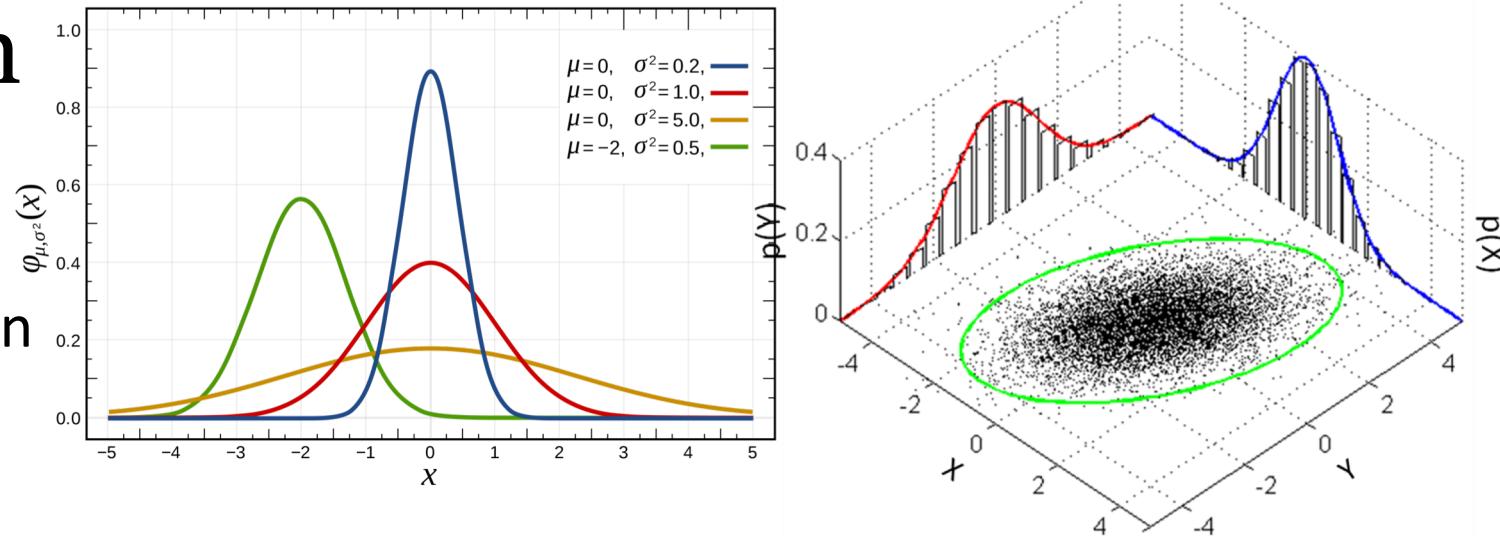
- Shuo Wang



# Gaussian distribution

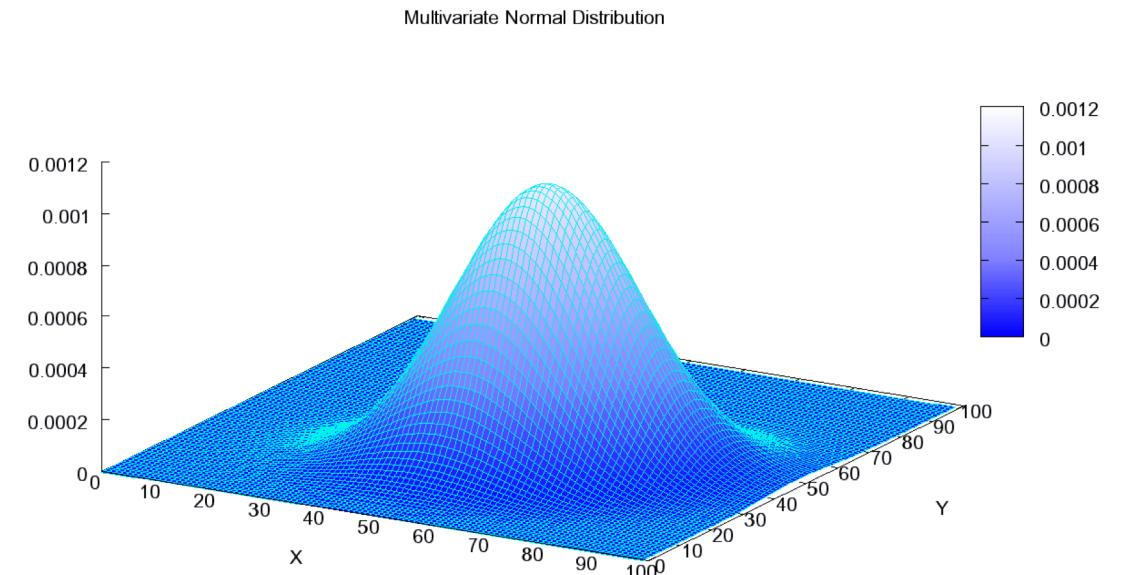
Gaussian (normal) distribution

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$



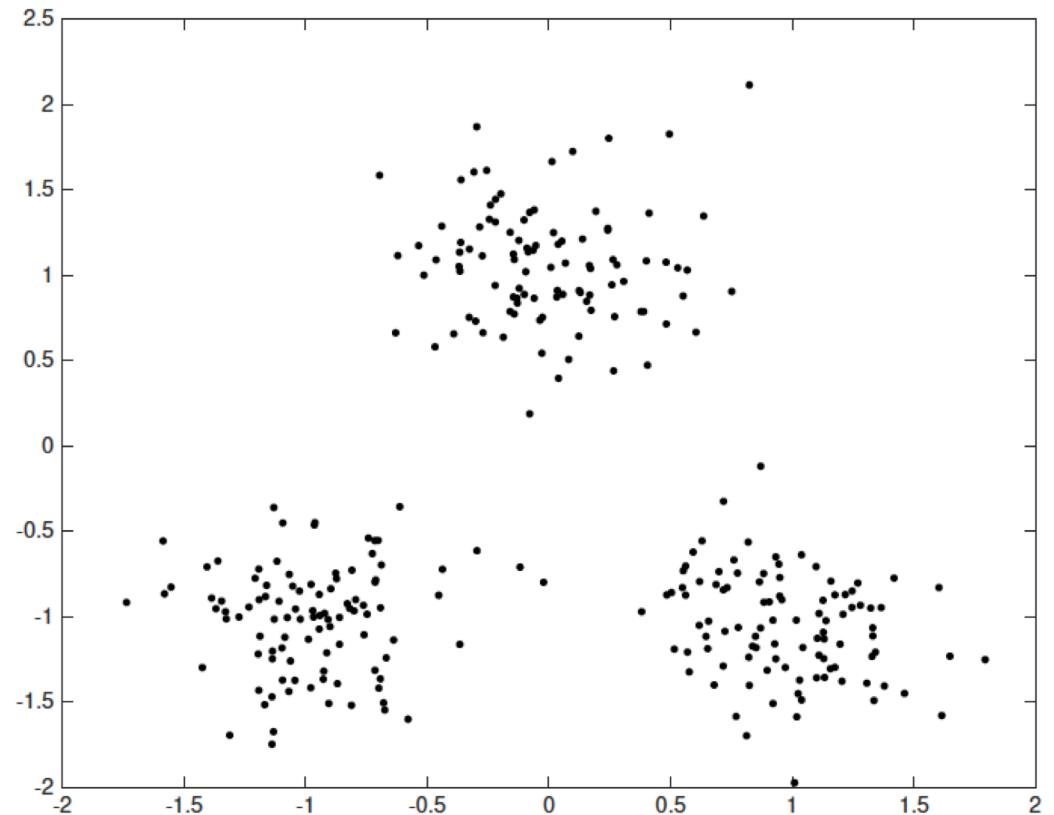
Multivariate Gaussian distribution

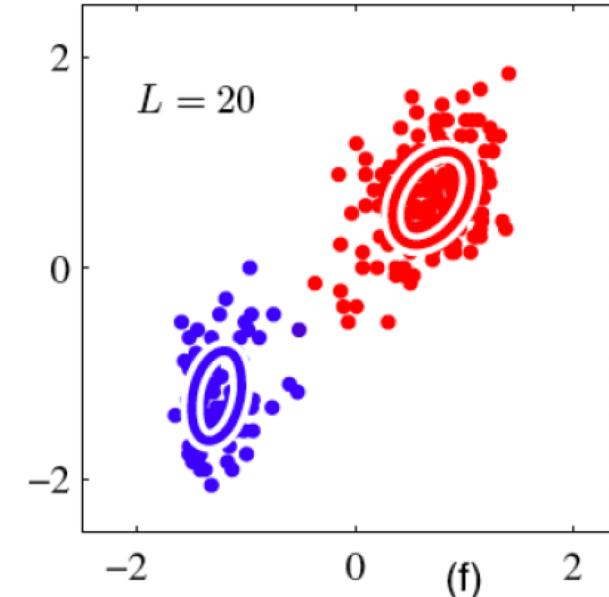
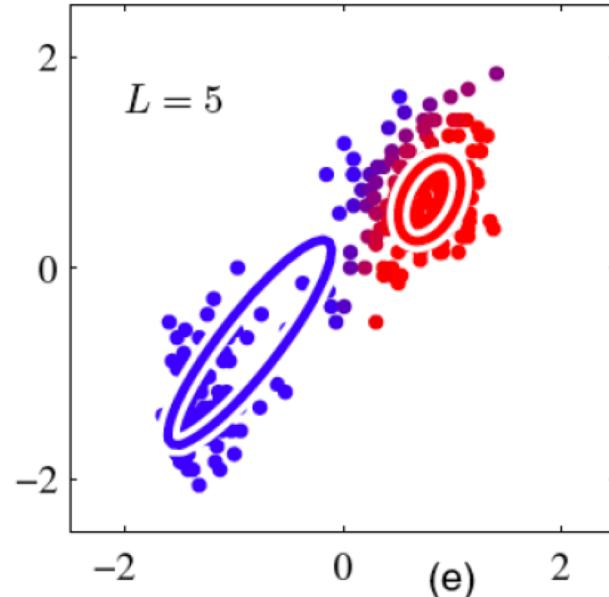
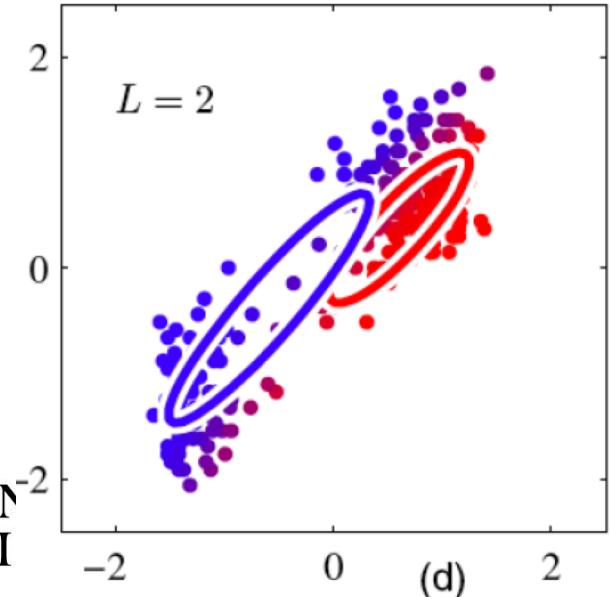
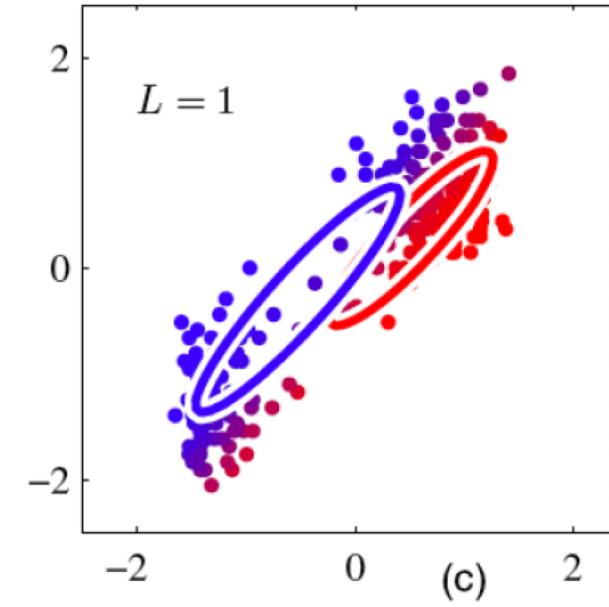
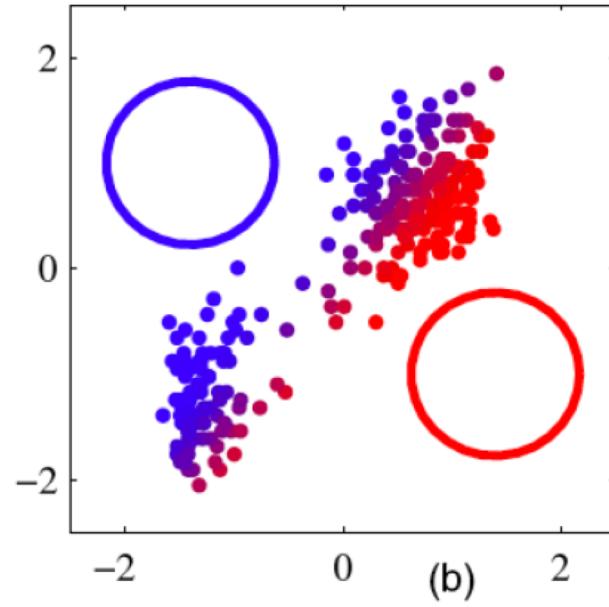
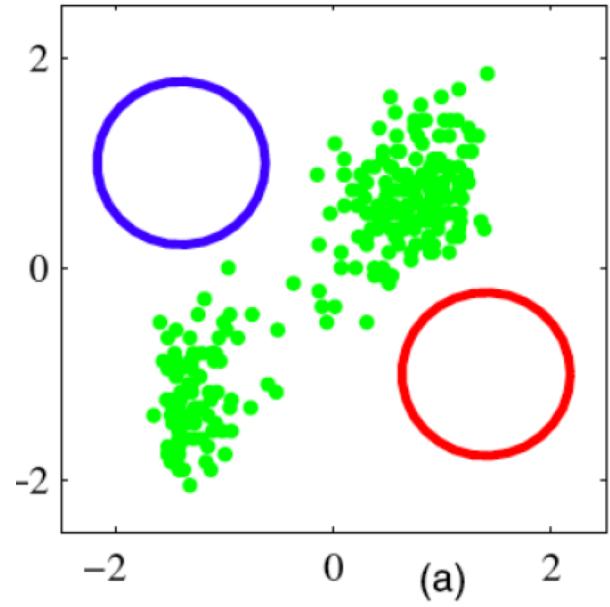
$$f_{\mathbf{X}}(x_1, \dots, x_k) = \frac{\exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right)}{\sqrt{(2\pi)^k |\boldsymbol{\Sigma}|}}$$



# Gaussian mixture models (GMMs)

- Assume data was generated by a set of Gaussian distributions.
- The probability density is a mixture of them.
- Find the parameters of the Gaussian distributions and how much each distribution contributes to the data.
- This is a **mixture model of Gaussian**.





# Generative Models

- In supervised learning, we model the joint distribution

$$p(\mathbf{x}, z) = p(\mathbf{x}|z)p(z)$$

- In unsupervised learning, we do not have labels  $z$ , we model

$$p(\mathbf{x}) = \sum_z p(\mathbf{x}, z) = \sum_z p(\mathbf{x}|z)p(z)$$



z is Hidden/Latent variables  
 $p(z)$  is  $\pi_k$  in later slides.



# Fitting GMMs: Maximum likelihood and EM

- To have a model best fit data, we need to maximize the (log) likelihood

$$\ln p(\mathbf{X}|\pi, \mu, \Sigma) = \sum_{n=1}^N \ln \left( \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}^{(n)} | \mu_k, \Sigma_k) \right)$$

- **Expectation**
  - if we knew  $\pi_k$ ,  $\mu$  and  $\Sigma$ , we can get “soft”  $Z_k$   
 $P(z_k^{(n)} | x^{(n)})$  - **responsibility**
- **Maximization**
  - if we know  $Z_k$ , we can get the best  $\pi_k$ ,  $\mu$  and  $\Sigma$

K: number of clusters,  $k = 1 \dots K$ .  
N: number of samples,  $n = 1 \dots N$ .

# Expectation-Maximization (EM Algorithm)

An optimization process that alternates between 2 steps:

1. E-step: compute the posterior probability over  $z$  given the current model.

$$\gamma_k^{(n)} = p(z_k^{(n)} | \mathbf{x}) = \frac{\pi_k \mathcal{N}(\mathbf{x}^{(n)} | \mu_k, \Sigma_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}^{(n)} | \mu_j, \Sigma_j)} \quad \text{responsibility}$$

# Expectation-Maximization (EM Algorithm)

2. M-step: Assuming data was really generated this way, change the parameters of each Gaussian to maximize the probability that it would generate the data it is currently responsible for.

$$\mu_k = \frac{1}{N_k} \sum_{n=1}^N \gamma_k^{(n)} \mathbf{x}^{(n)}$$

$$\Sigma_k = \frac{1}{N_k} \sum_{n=1}^N \gamma_k^{(n)} (\mathbf{x}^{(n)} - \mu_k)(\mathbf{x}^{(n)} - \mu_k)^T$$

$$\pi_k = \frac{N_k}{N} \quad \text{with} \quad N_k = \sum_{n=1}^N \gamma_k^{(n)}$$



# Online Demo

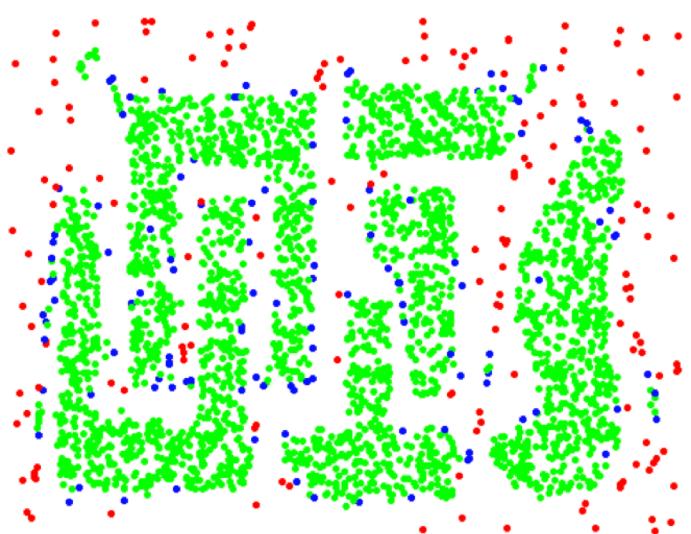
- <https://lukapopijac.github.io/gaussian-mixture-model/>



UNIVERSITY OF  
BIRMINGHAM

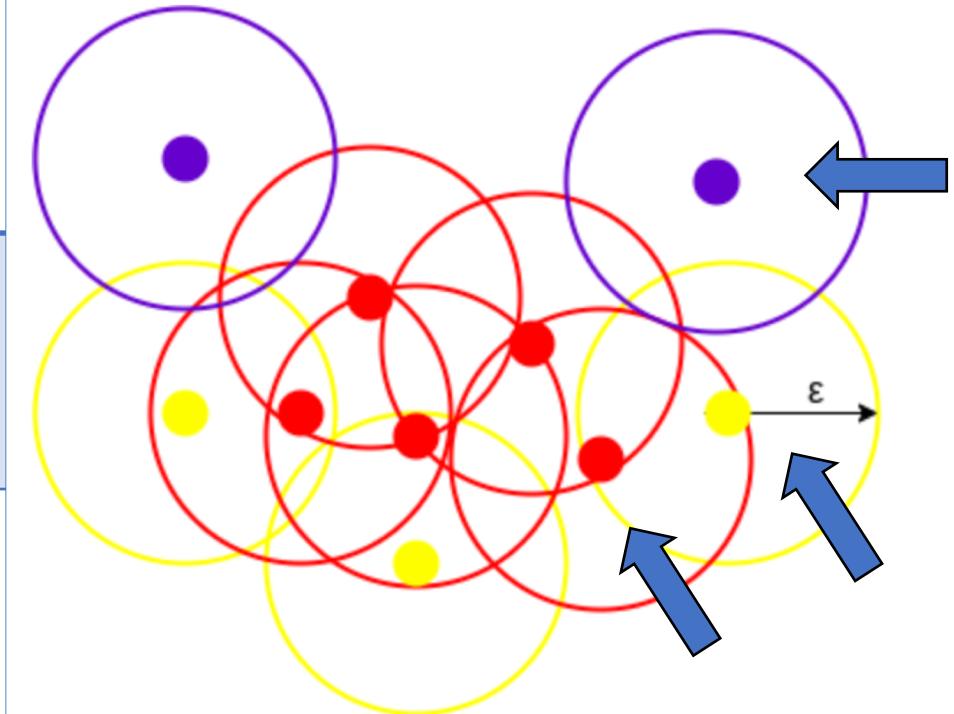
# Density-based Clustering - DBSCAN

- Acronym for: Density-based spatial clustering of applications with noise
- Clusters are dense regions in the data space separated by regions of lower sample density.
- A cluster is defined as a maximal set of density connected points.
- Discover clusters of arbitrary shape.



# Three types of points

core	The point has at least minPts number of points within Eps
border	The point has fewer than minPts within Eps, but is in the neighbourhood (i.e. circle) of a core point.
noise	Any point that is not a core point or a border point.

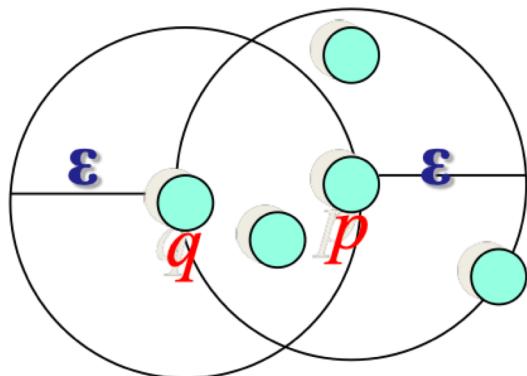


$\text{minPts} = 3$   
red: core  
yellow: border  
purple: noise

# How to form core points into clusters?

## -- Density-reachability

- Directly density-reachable: a point  $q$  is directly density-reachable from point  $p$  if  $p$  is a core point and  $q$  is in  $p$ 's neighbourhood.



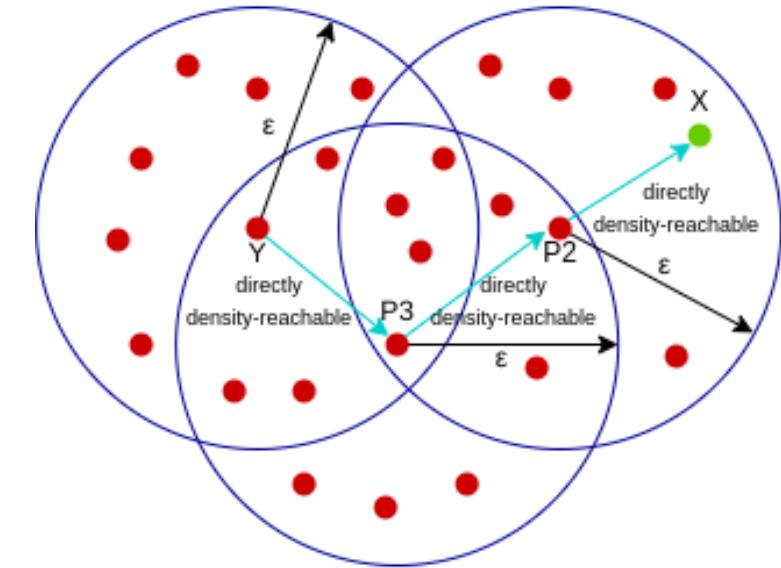
- $q$  is directly density-reachable from  $p$
- $p$  is not necessarily directly density-reachable from  $q$
- Density-reachability is asymmetric.

MinPts = 4

# How to form core points into clusters?

## -- Density-reachability

- Density-Reachable (directly and indirectly)
  - ❖ A point X is directly density-reachable from P2
  - ❖ P2 is directly density-reachable from P3
  - ❖ P3 is directly density-reachable from Y
  - ❖ Y -> P3 -> P2 -> X form a chain  
(X is a border point)



- X is indirectly density-reachable from Y
- Y is not density-reachable from X

# The algorithm

- 1. Label all points as core, border or noise.
- 2. Eliminate noise points.
- 3. For every core point  $p$  that has not been assigned to a cluster:
  - ❖ Create a new cluster with the point  $p$  and all the points that are density-reachable from  $p$
- 4. For border points belonging to more than 1 cluster, assign it to the cluster of the closest core point.



# Online Demo

- <https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/>



UNIVERSITY OF  
BIRMINGHAM

# Introduction to Supervised Learning

Ata Kaban

# What Is Supervised Learning?

- One of the most prevalent forms of ML
  - Teach a computer to do something, then let it use its knowledge to do it
  - Also called “learning with a teacher”
- Other forms of ML
  - Unsupervised learning (“learning without a teacher”)
  - Reinforcement learning (“learning with (delayed) feedback”)

# Example: Spam detection

- Input: Emails received

The screenshot shows a web-based spam detection interface. At the top, a message says "4 broken emails has been found and recovered". Below it is an email from "Users Support <deeniem1988@pro-pakindustries.com>" received on "Jun 24 (12 days ago)". A red warning bar at the bottom of the email preview states: "Be careful with this message. It contains content that's typically used to steal personal information. [Learn more](#)". Below the email is a "Google Notification Service" report for "Recovery Service Report" dated "6/24/2016" which "4 broken emails has been found and recovered". A blue button labeled "(4) emails" is visible. At the bottom, a note says: "We hope you found this message to be useful. However, if you'd rather not receive future e-mails of this sort, please opt-out [here](#)".

- Output: “Spam”, or “No spam”

# Example: Stock price prediction

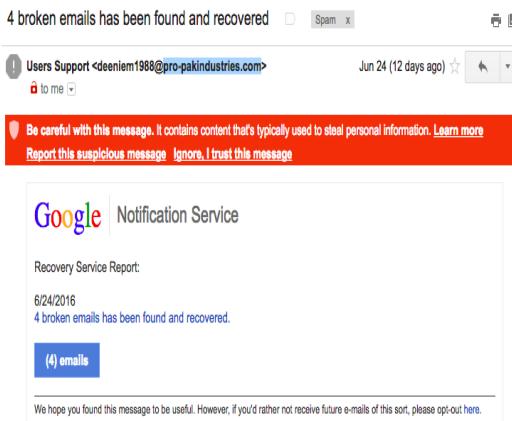
- Input: Historical records of stock prices



- Output: Next day's stock price

# Spam detection

- Input: Emails received



- Output: “Spam”, or “No spam”
- This is a **classification** problem.  
The output has 2 possible values

# Stock price prediction

- Input: Historical records of stock prices



- Output: Next day’s stock price
- This is a **regression** problem.  
The output is a real value.

# Types of supervised learning

- Regression
- Classification
  - Binary
  - Multi-class
  - ...

# Supervised learning

Task:

- Given some **input**  $x$ ,
- Predict an appropriate **output**  $y$ .

*Want: a **function**  $f$  such that  $f(x)=y$*

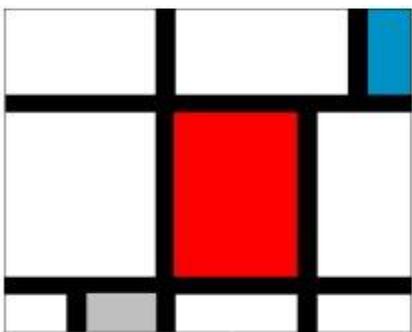
*Have: examples of input-output pairs  $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) , \dots, (x^{(n)}, y^{(n)})$*

*Supervised learning helps find a good  $f$ .*

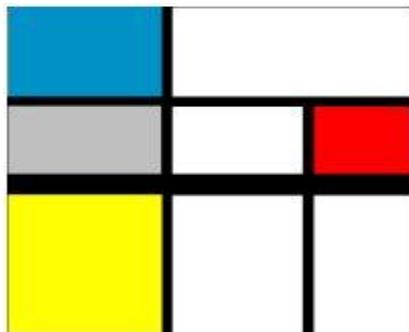
# Training data

- Supervised learning needs annotated data for training:  
in the form of examples of (Input, Output) pairs
- After training completed,
  - you present it with new Input that it hasn't seen before
  - It needs to predict the appropriate Output

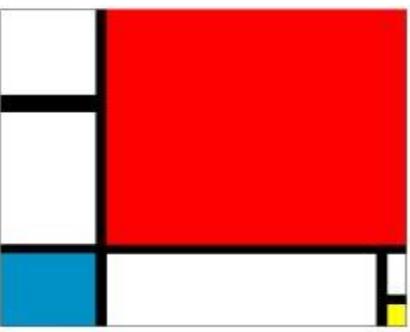
# Is painting 8 a genuine Mondrian?



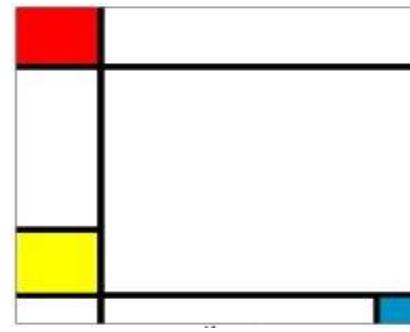
1. No



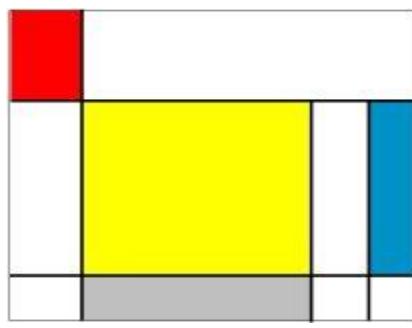
2. No



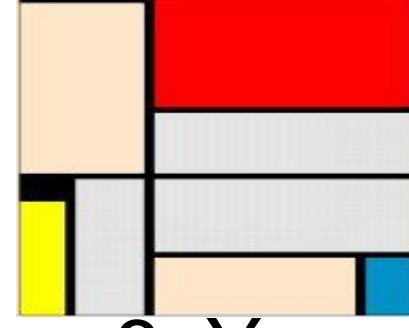
3. Yes



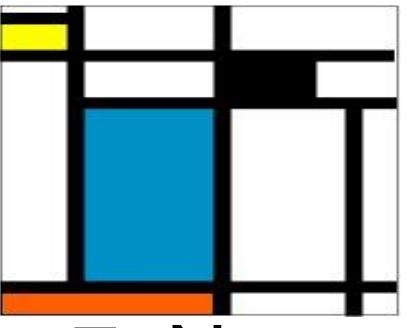
4. Yes



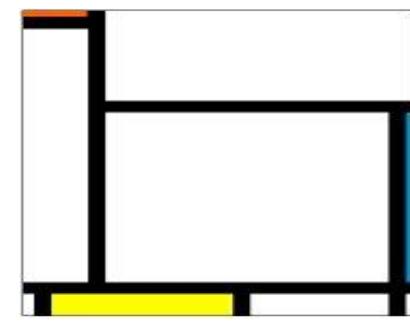
5. No



6. Yes



7. No



8. ?

Annotated  
training data

Examples

Painting 8

Attributes

Labels

Number	Lines	Line types	Rectangles	Colours	Mondrian?
1	6	1	10	4	No
2	4	2	8	5	No
3	5	2	7	4	Yes
4	5	1	8	4	Yes
5	5	1	10	5	No
6	6	1	8	6	Yes
7	7	1	14	5	No

Number	Lines	Line types	Rectangles	Colours	Mondrian?
8	7	2	9	4	???

# How quick will your team complete a project?

(programming language)	(team expertise)	(estimated size)	...	(required effort)
Java	low	1000	...	10 p-month
C++	medium	2000	...	20 p-month
Java	high	2000	...	8 p-month
...	...	...	...	...

# General notation we will use

(programming language)	(team expertise)	(estimated size)	...	(required effort)
		$x^{(1)}$		$y^{(1)}$
		$x^{(2)}$		$y^{(2)}$
		$x^{(3)}$		$y^{(3)}$
...	...	...	...	...

Vector notation

$$x^{(i)} = \left( x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, \dots, x_d^{(i)} \right)$$

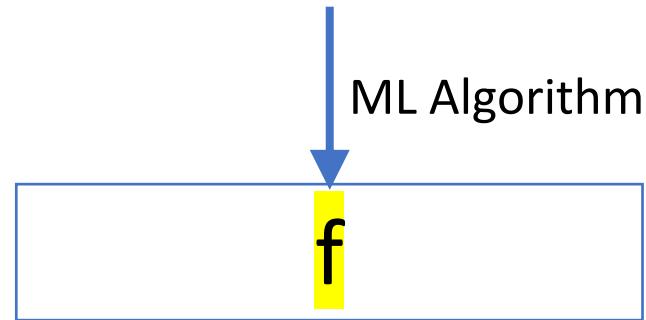
The input of the  $i$ -th example

Attributes

# Workflow of supervised learning:

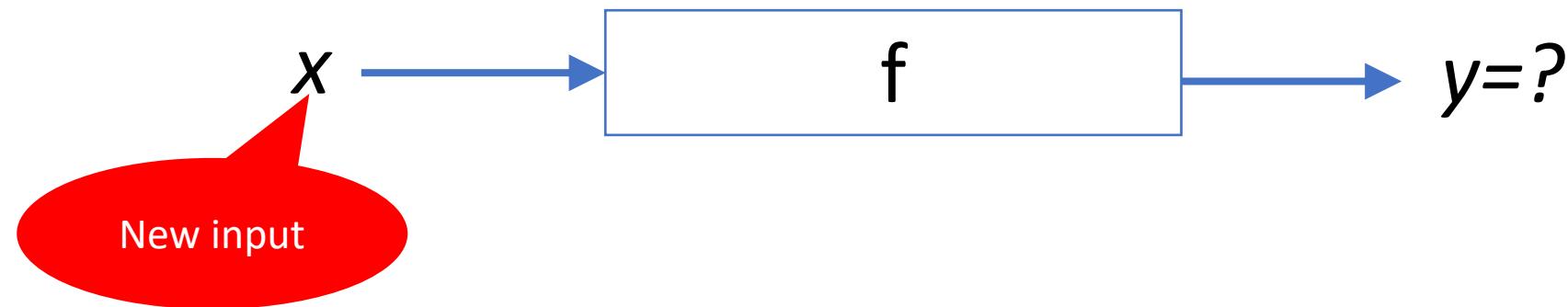
## 1. Training phase

$$(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \dots, (\mathbf{x}^{(n)}, y^{(n)})$$



# Workflow of supervised learning:

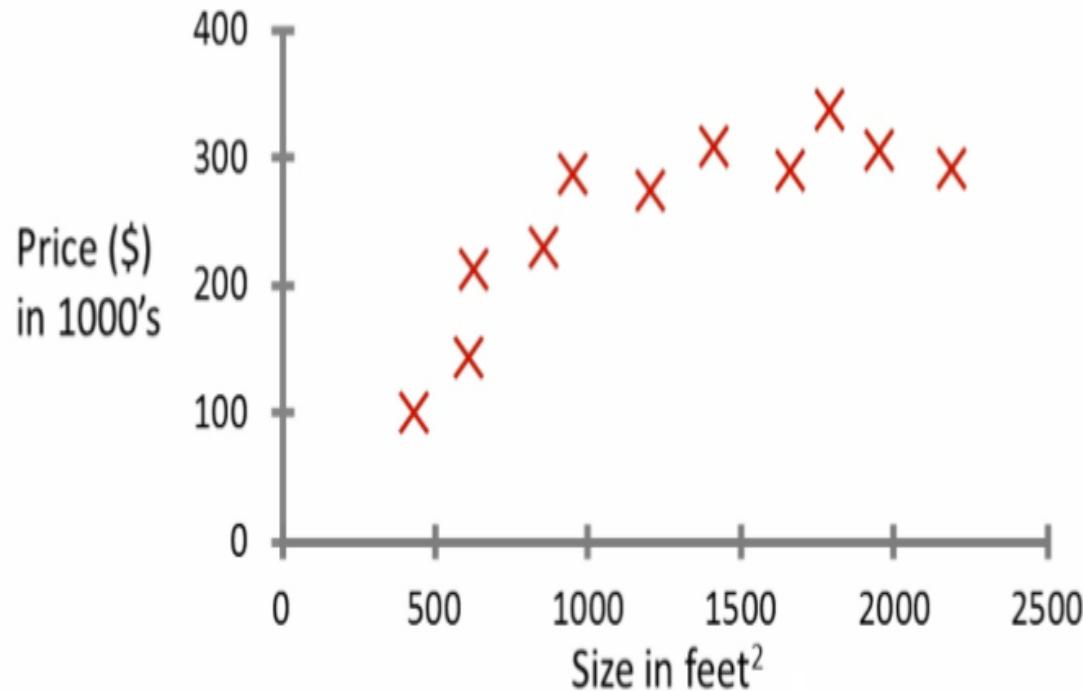
## 2. Test phase & use



# Pictorially

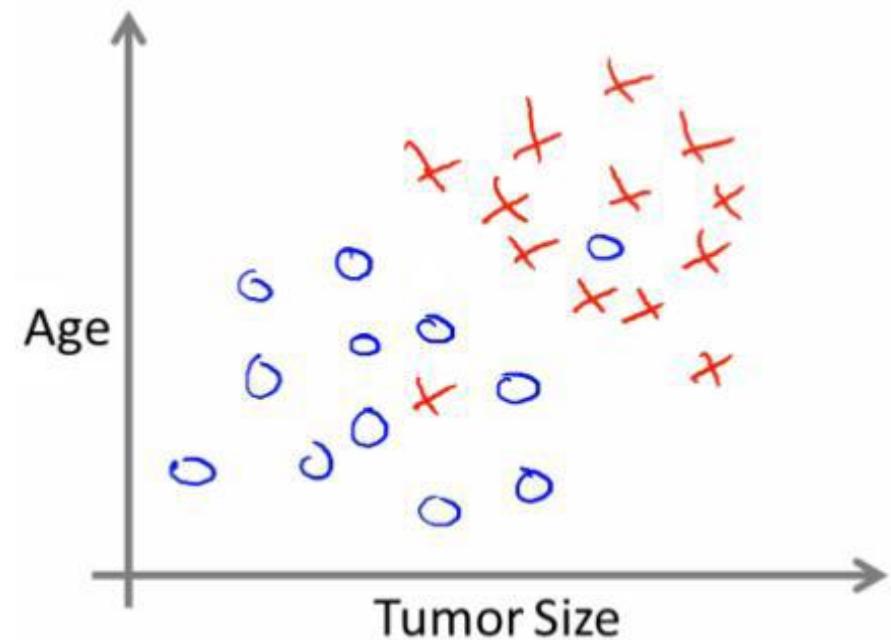
- Regression problem

Housing price prediction.



- Classification problem

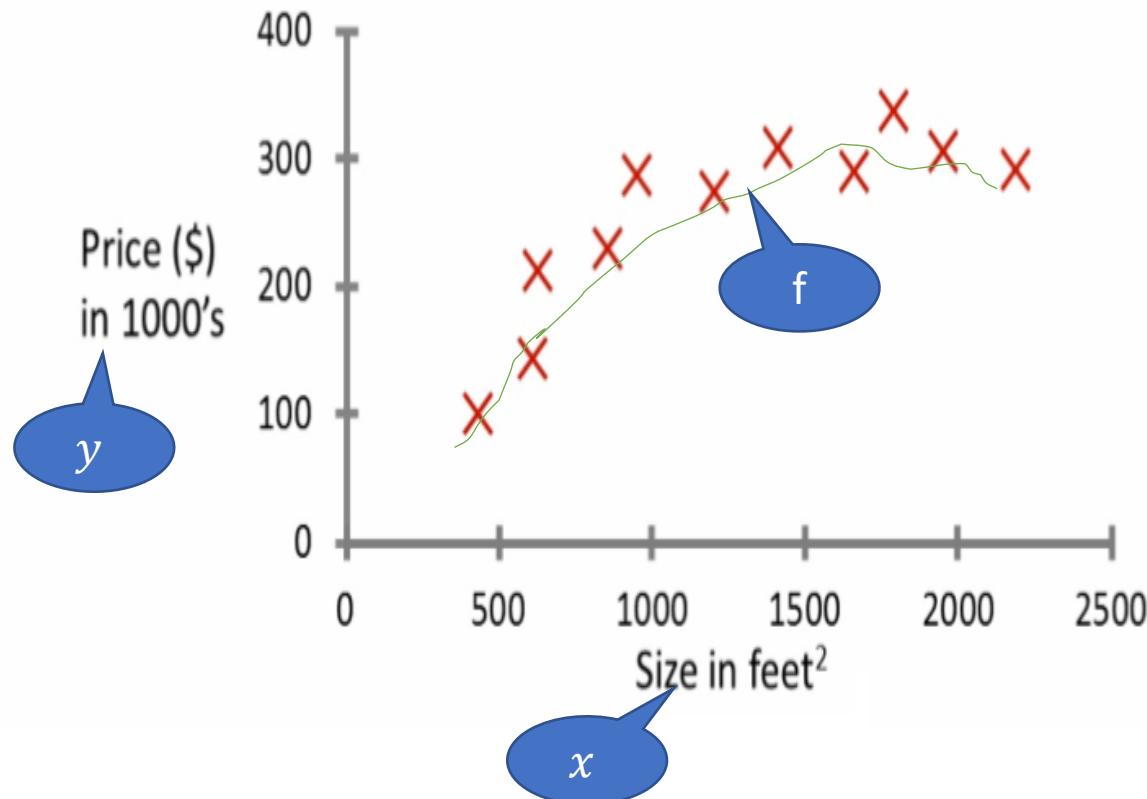
Breast cancer prediction



# Pictorially

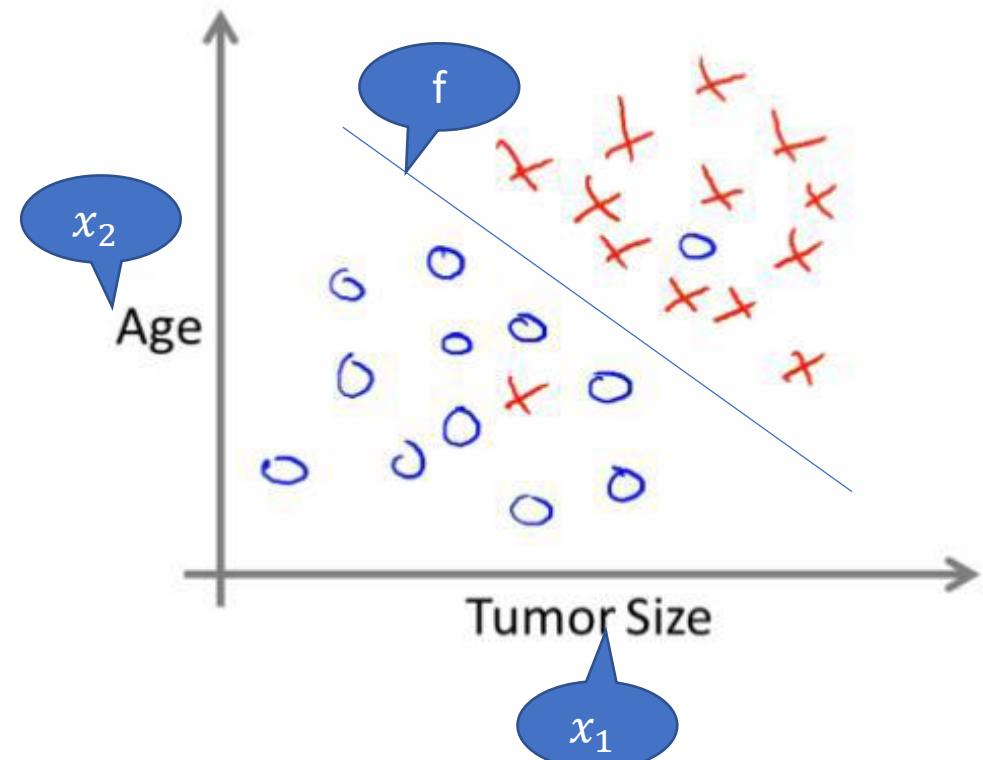
- Regression problem

Housing price prediction.



- Classification problem

Breast cancer prediction



# Terminology in Supervised Learning

- Input = attribute(s) = feature(s) = independent variable
- Output = target = response = dependent variable
- function = hypothesis = predictor

# Pause. Is this some magic?

So...

- there is this unknown function we're after
- we are given the function values at  $n$  specific points only (training set)
- is it really possible to find out the function values at other points?
- No!
- Not unless we make the right assumptions about the unknown function
- Each ML algorithm, implicitly or explicitly, makes assumptions.
- There is a zoo of ML algorithms, there is no best ML algorithm
- Our goal is to focus on few of them, and understand how they work

# Applications of supervised learning

- Handwriting recognition
  - When you write an envelope, algorithms can automatically route envelopes through the post
- Computer vision & graphics
  - When you go out during lockdown, object detection & visual tracking algorithms can automatically detect compliance with the rules
- Bioinformatics
  - Algorithms can predict protein function from sequence
- Human-computer interaction
  - Intrusion detection algorithms can recognise speech, gestures, intention

# Why is ML so Prevalent?

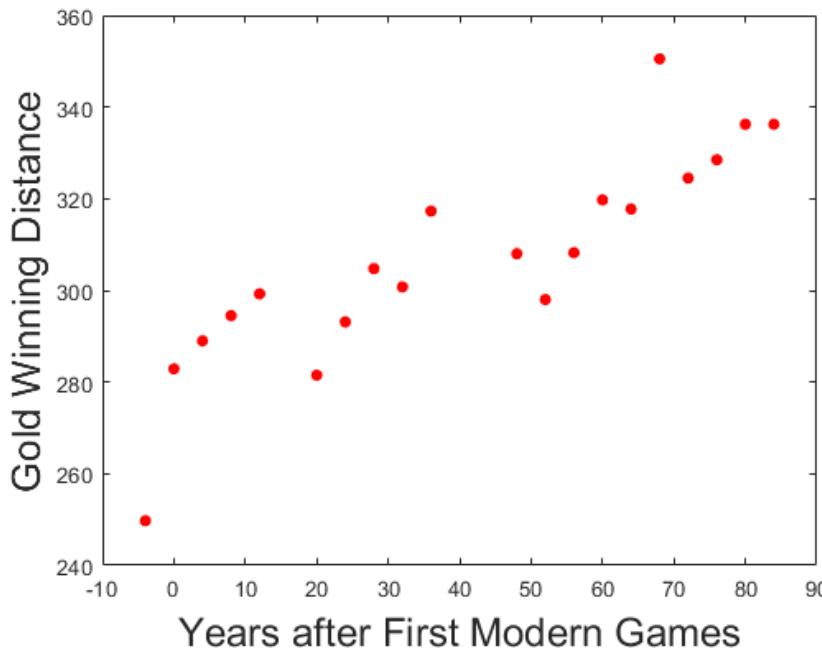
- Generality
  - E.g. a robot learning to navigate mazes must be able to learn the layout of the maze it encounters
- Adaptability
  - E.g. a program designed to predict tomorrow's stock market must learn to adapt when conditions change from boom to bust
- Applicability
  - Often the human programmer has no idea how to program a solution to the problem (think of how you recognise your friend's face)

# Linear Regression

Ata Kaban

# Example of a regression problem

- Let's look at some fun data. Can we predict the long-jump Gold winning distance 125 years after the first modern Olympic games?

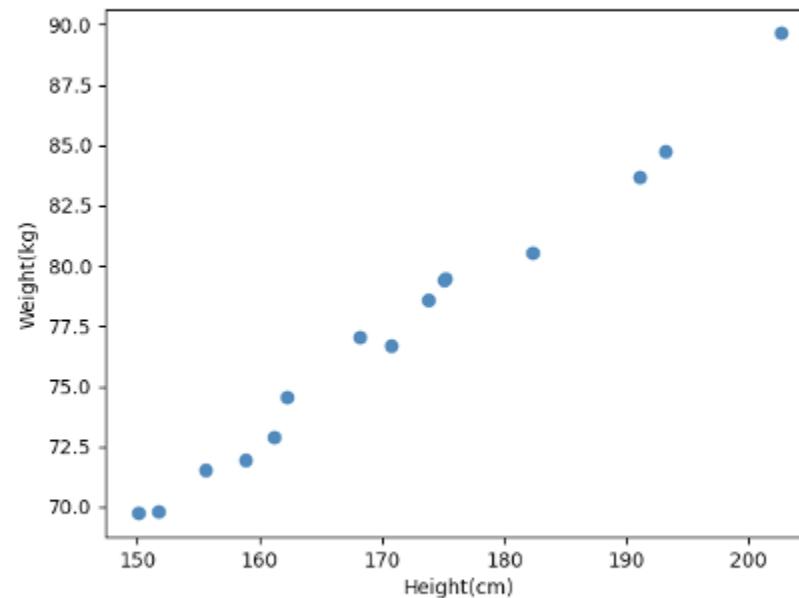


Data from: Rogers & Girolami. A First Course in Machine Learning. 2<sup>nd</sup> edition.  
Chapman & Hall/CRC, 2017

# Example of a regression problem

- Let's look at more fun data. Can we predict people's weight from their height?

Height(cm)	Weight(kg)
150.00686	69.73347
151.64326	69.83261
155.54032	71.55730
158.80535	71.92875
161.17561	72.92118
⋮	
175.15167	79.48533
182.32900	80.52182
191.11317	83.67998
193.21947	84.72086
202.68705	89.64049



# Regression

- Regression means learning a function that captures the “trend” between input and output
- We then use this function to predict target values for new inputs

# Univariate linear regression

- Visually, there appears to be a trend
- A reasonable **model** seems to be the **class of linear functions (lines)**
- We have one input attribute (year) - hence the name **univariate**

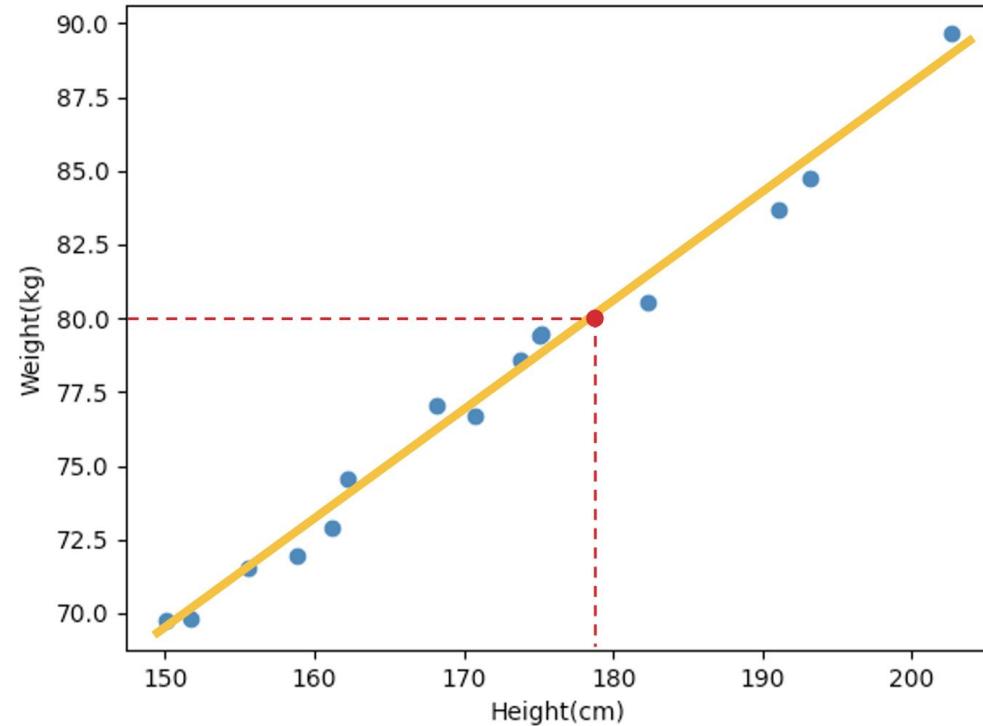
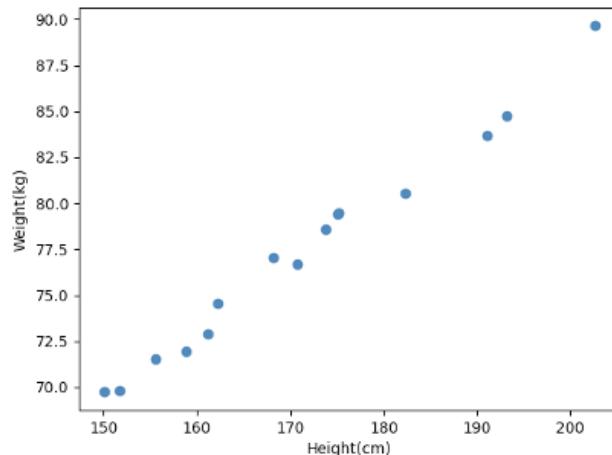
$$y = f(x; w_0, w_1) = w_1 x + w_0$$

dependent variable                          free parameters                          Independent variable

- Any line is described by this equation by specifying values for  $w_1, w_0$ .

# Check your understanding

Height(cm)	Weight(kg)
150.00686	69.73347
151.64326	69.83261
155.54032	71.55730
158.80535	71.92875
161.17561	72.92118
⋮	
175.15167	79.48533
182.32900	80.52182
191.11317	83.67998
193.21947	84.72086
202.68705	89.64049



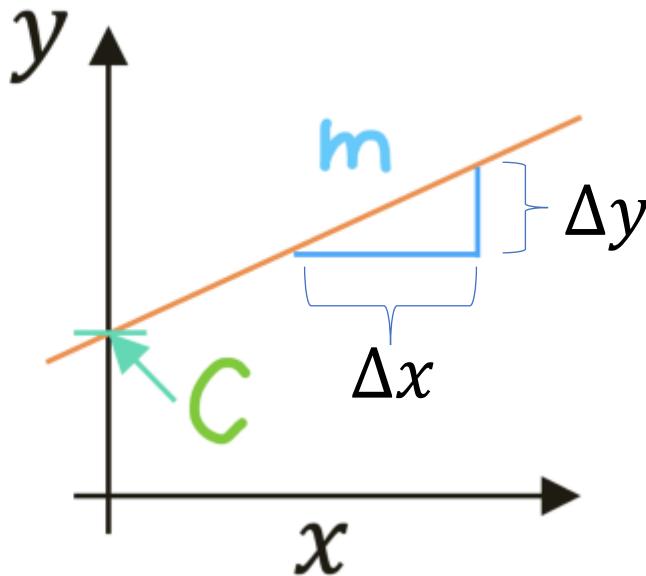
Suppose that from historical data someone already calculated the parameters of our linear model are  $w_0 = 1.68$ ,  $w_1 = 0.44$ . A new person (James) has height  $x=178\text{cm}$ . Using our model, we can predict James' weight is  $0.44 * 178 + 1.68 = 80\text{kg}$ .

# Play around with linear functions

- Go to <https://www.desmos.com/calculator>
- Type:  $y = w_1x + w_0$
- Plug in some values for the free parameters, or use the slider to see their effect
- *What is the role of the free parameters?*
  - $w_1$  is the slope of the line
  - $w_0$  is the intercept with the y-axis
- Fixing concrete numbers for these parameters gives you specific lines

Equation of a straight line with slope  $m$  and intercept  $c$ .

$$f(x) = y = mx + c$$

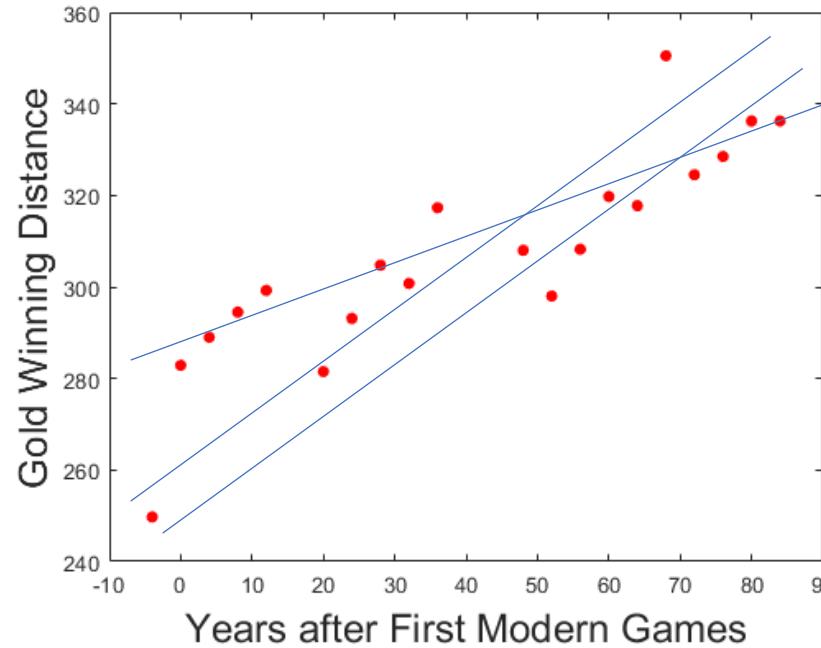


$$m = \frac{\text{change in } y}{\text{change in } x} = \frac{\Delta y}{\Delta x},$$

This is why:

$$\begin{aligned} f(x + \Delta x) &= m(x + \Delta x) + c = mx + m \cdot \Delta x + c = f(x) + m \cdot \Delta x \\ \Rightarrow m &= \frac{f(x + \Delta x) - f(x)}{\Delta x} = \frac{\Delta y}{\Delta x} \end{aligned}$$

# Our goal: Find the “best” line



- Which is the “best” line? That captures the trend in the data.
- Determine the “best” values for  $w_1, w_0$ .

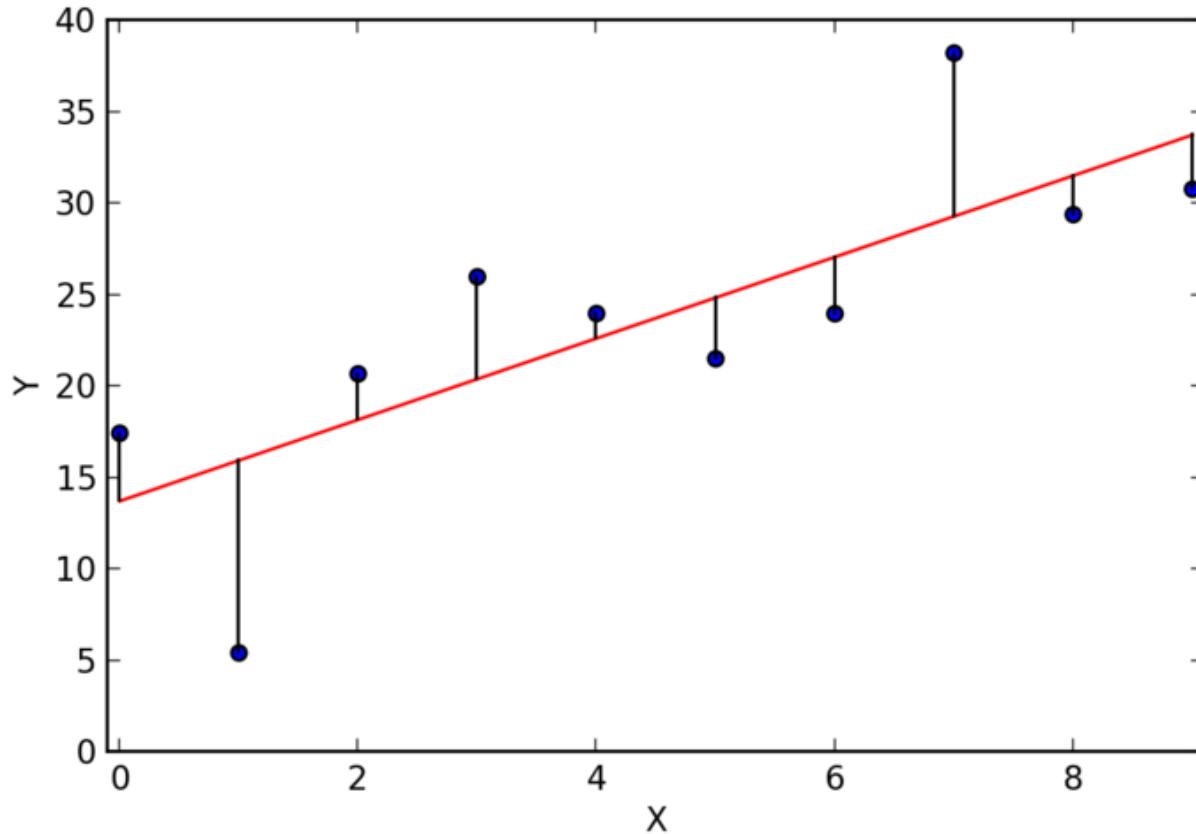
# Loss functions (or cost functions)

- We need a criterion that, given the data, for any given line will tell us how bad is that line.
- Such criterion is called a loss function. It is a function of the free parameters!

## Terminology

- Loss function = cost function = loss = cost = error function

# We average the losses on all training examples



For each training example (point)  
 $n = 1, \dots, N$ ,

The loss on the  $n$ -th point is the  
mismatch between the output of  
the model for this point  
 $f(x^{(n)}; w_0, w_1)$  and the observed  
target  $y^{(n)}$ .

Average these losses.

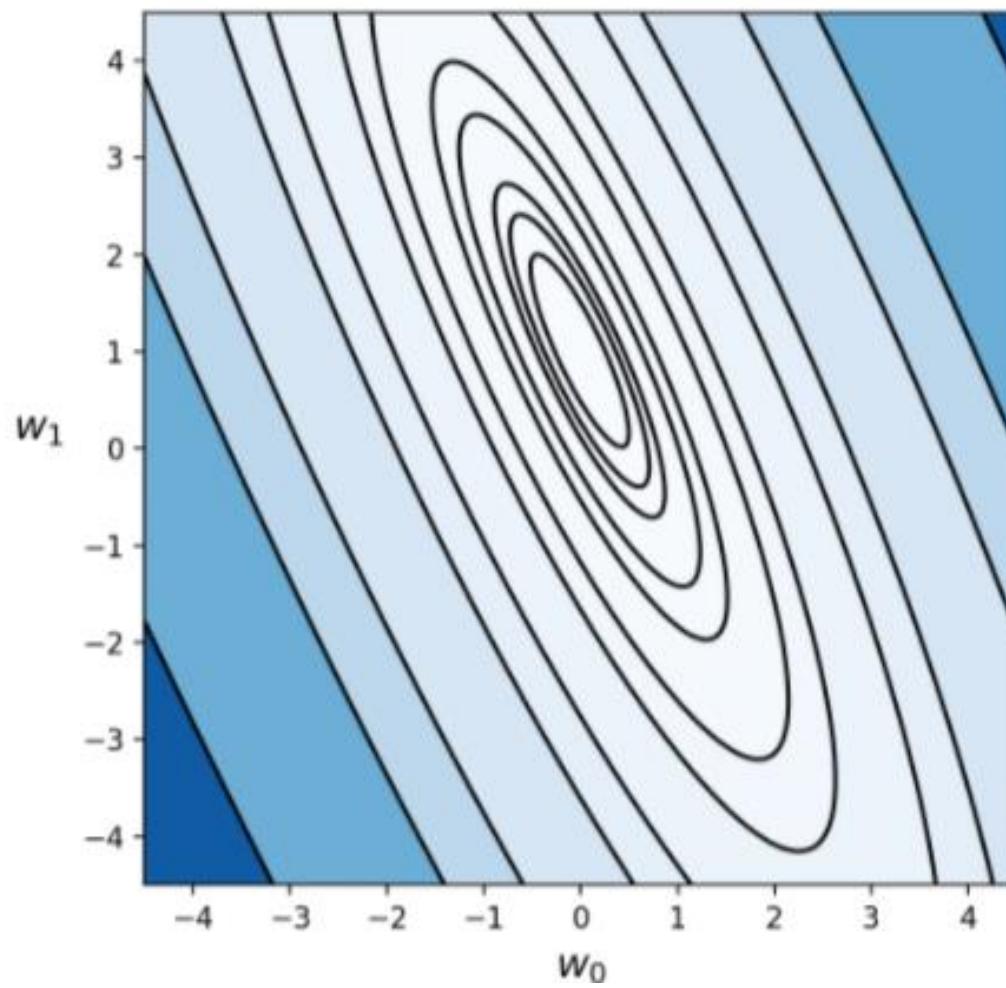
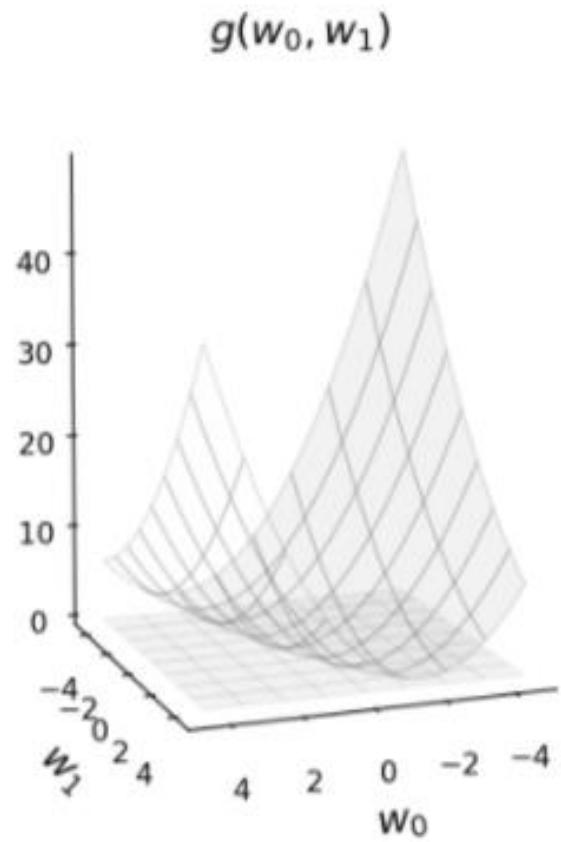
# Square loss (L2 loss)

- The loss expresses an error, so it must be always non-negative
- Square loss is a sensible choice to measure mismatch for regression
- Mean Square Error (MSE)

$$g(w_0, w_1) = \frac{1}{N} \sum_{n=1}^N \underbrace{(f(x^{(n)}; w_0, w_1) - y^{(n)})^2}_{\text{loss for the } n\text{-th training example}}$$

and recall that, for any  $x$ , we have  $f(x; w_0, w_1) = w_1 x + w_0$

# Cost function depends on the free parameters



# Check your understanding

- Suppose a linear function with parameters  $w_0 = 0.5$ ,  $w_1 = 0.5$
- Compute the loss function value for this line at the training example:  $(1,3)$ .
- $f(x^{(1)}; 0.5, 0.5) = 0.5 * 1 + 0.5 = 1$  (output of the model)
- $y^{(1)} = 3$  (actual target)
- Square loss for this point:  $(1-3)^2 = 4$ .
- Cost = 4.

# Univariate linear regression – what we want to do

- Given training data

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(N)}, y^{(N)})$$

- Fit the model

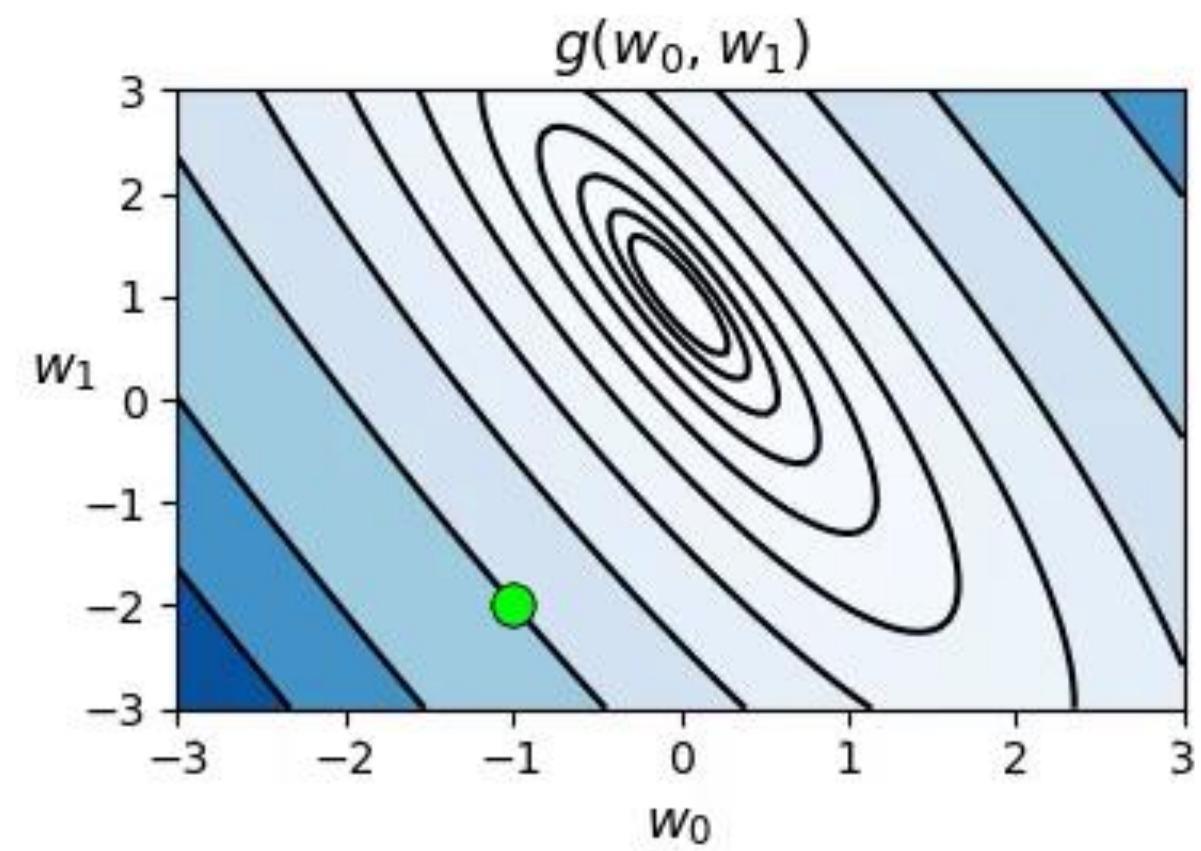
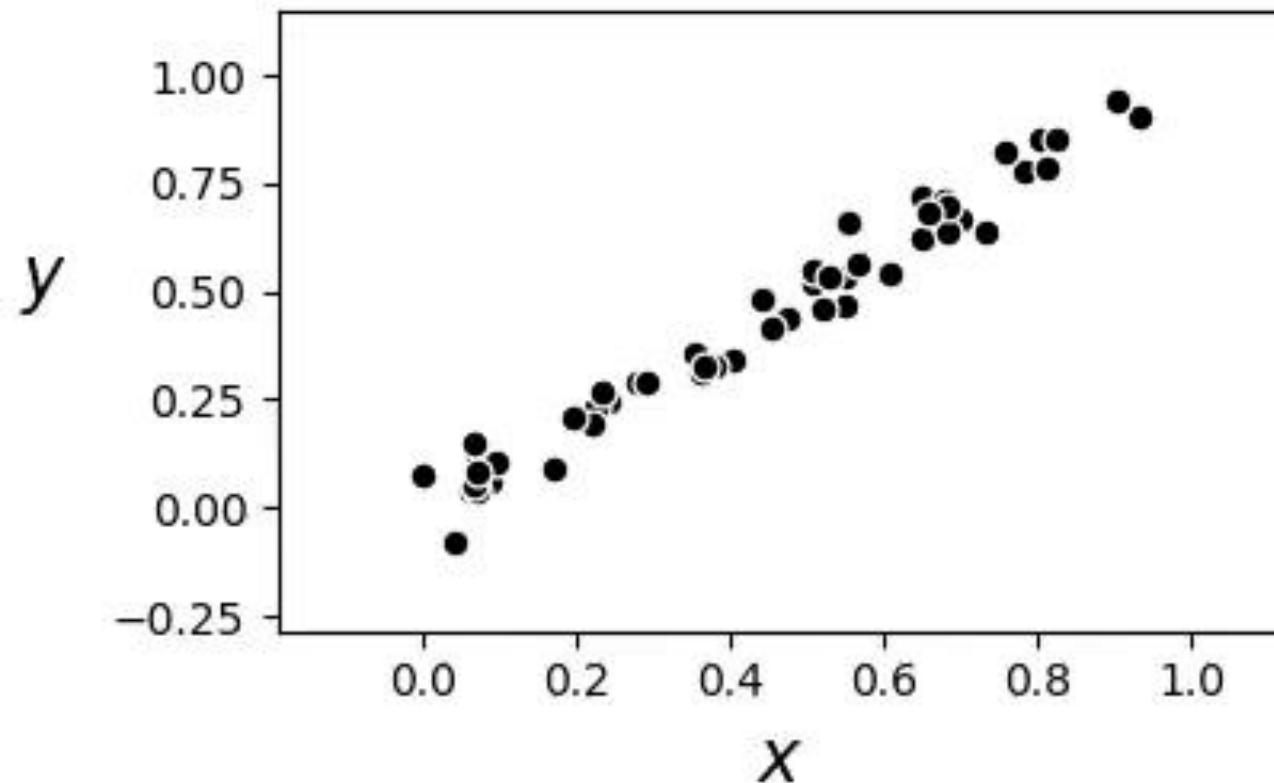
$$y = f(x; w_0, w_1) = w_1 x + w_0$$

- By minimising the cost function

$$g(w_0, w_1) = \frac{1}{N} \sum_{n=1}^N ((w_1 x^{(n)} + w_0) - y^{(n)})^2$$

# Univariate linear regression – what we want to do

- Every combination of  $w_0$  and  $w_1$  has an associated cost.
- To find the ‘best fit’ we need to find values for  $w_0$  and  $w_1$  such that the cost is minimum.



# Gradient Descent

# Gradient Descent

- A general strategy to minimise cost functions.

Goal: Minimise cost function  $g(w_0, w_1)$

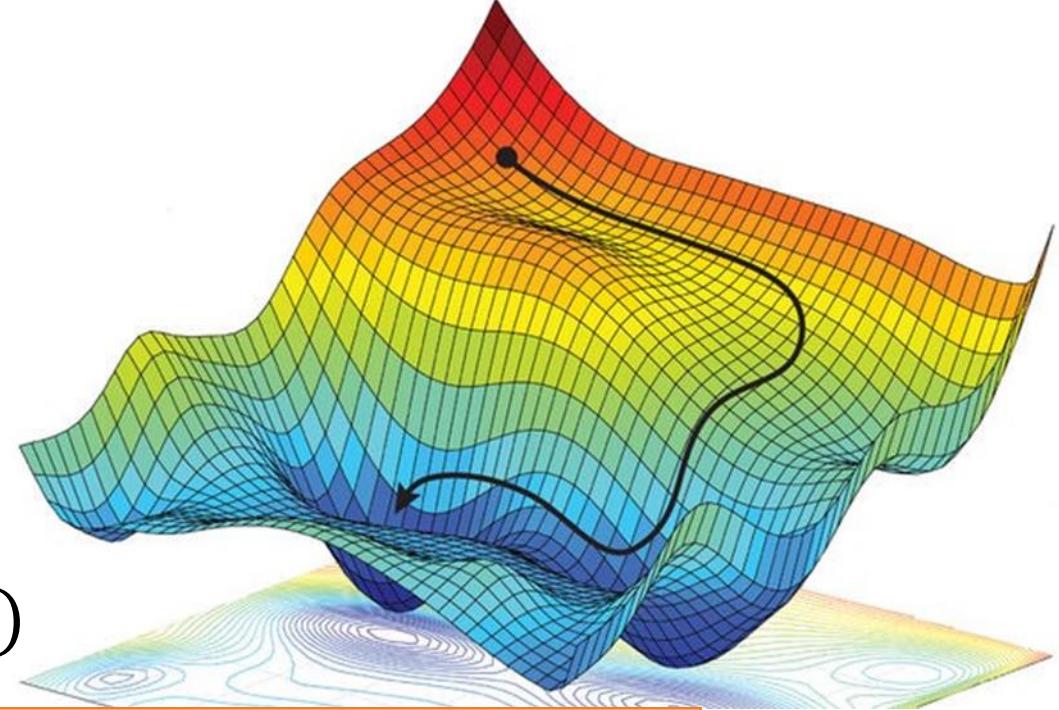
Start at say  $w_0 := 0, w_1 := 0$

Repeat until no change occurs

    Update  $w_0, w_1$  by taking

    a small step in the direction of the steepest descent

Return  $w_0, w_1$



# Gradient descent – the general algorithm

- Goal: Minimise cost function  $g(\mathbf{w})$ , where  $\mathbf{w} = (w_0, w_1, \dots)$

Input:  $\alpha > 0$

Initialise  $\mathbf{w}$  // at 0 or some random value

Repeat until convergence

$$\mathbf{w} := \mathbf{w} - \alpha \nabla g(\mathbf{w})$$

Return  $\mathbf{w}$

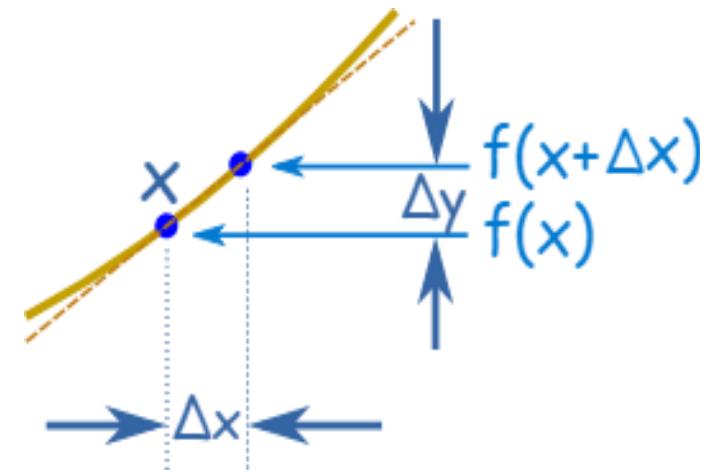
step size

direction

$\alpha$  is called “learning rate”= “step size”, for instance 0.01

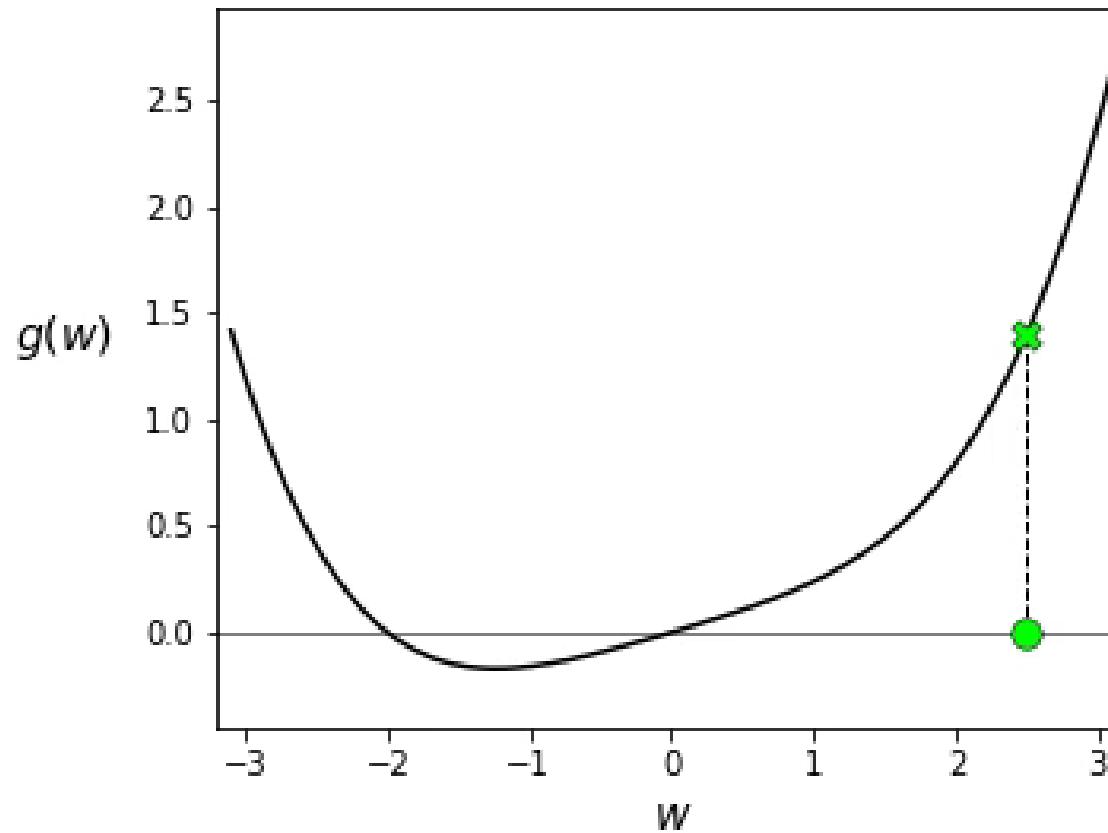
# How to find the best direction?

- First, recall from calculus that the derivative of a function is the change in function value as the argument of the function changes by a minimal amount.
- The derivative evaluated at a given location gives us the slope of the tangent line at that point.
- The negative of the slope points towards the minimum point. Check!



$$\frac{\Delta y}{\Delta x} = \frac{f(x+\Delta x) - f(x)}{\Delta x}$$
$$\Delta x \rightarrow 0$$

# Demo example for gradient descent algorithm



# Gradient

- **Partial derivative** with respect to  $w_0$  is  $\frac{\delta g(w_0, w_1)}{\delta w_0}$ . It means the derivative function of  $g(w_0, w_1)$  when  $w_1$  is treated as constant.
- **Partial derivative** with respect to  $w_1$  is  $\frac{\delta g(w_0, w_1)}{\delta w_1}$ . It means the derivative function of  $g(w_0, w_1)$  when  $w_0$  is treated as constant.
- The vector of partial derivatives is called the **gradient**.

$$\nabla g(w) = \begin{pmatrix} \frac{\delta g(w_0, w_1)}{\delta w_0} \\ \frac{\delta g(w_0, w_1)}{\delta w_1} \end{pmatrix} \quad \text{where } w = \begin{pmatrix} w_0 \\ w_1 \end{pmatrix}$$

- The negative of the gradient evaluated at a location  $(\widehat{w}_0, \widehat{w}_1)$  gives us the direction of the **steepest descent** from that location.
- We take a small step in that direction.

Gradient Descent applied to solving  
Univariate Linear Regression

# Computing the gradient for our L2 loss

- Recall the cost function  $g(w_0, w_1) = \frac{1}{N} \sum_{n=1}^N ((w_1 x^{(n)} + w_0) - y^{(n)})^2$
- *Using the chain rule, we have\*:*

$$\frac{\delta g(w_0, w_1)}{\delta w_0} = \frac{2}{N} \sum_{n=1}^N (w_1 x^{(n)} + w_0) - y^{(n)}$$

$$\frac{\delta g(w_0, w_1)}{\delta w_1} = \frac{2}{N} \sum_{n=1}^N ((w_1 x^{(n)} + w_0) - y^{(n)}) x^{(n)}$$

---

\*For a very detailed explanations of all steps watch: <https://www.youtube.com/watch?v=sDv4f4s2SB8>

# Algorithm for univariate linear regression using GD

- Goal: Minimise  $g(w_0, w_1) = \frac{1}{N} \sum_{n=1}^N ((w_1 x^{(n)} + w_0) - y^{(n)})^2$

Input:  $\alpha > 0$ , training set  $\{(x^{(n)}, y^{(n)}): n=1, \dots, N\}$

Initialise  $w_0 := 0, w_1 := 0$

Repeat

    For  $n=1, \dots, N$  // more efficient to update after each data point

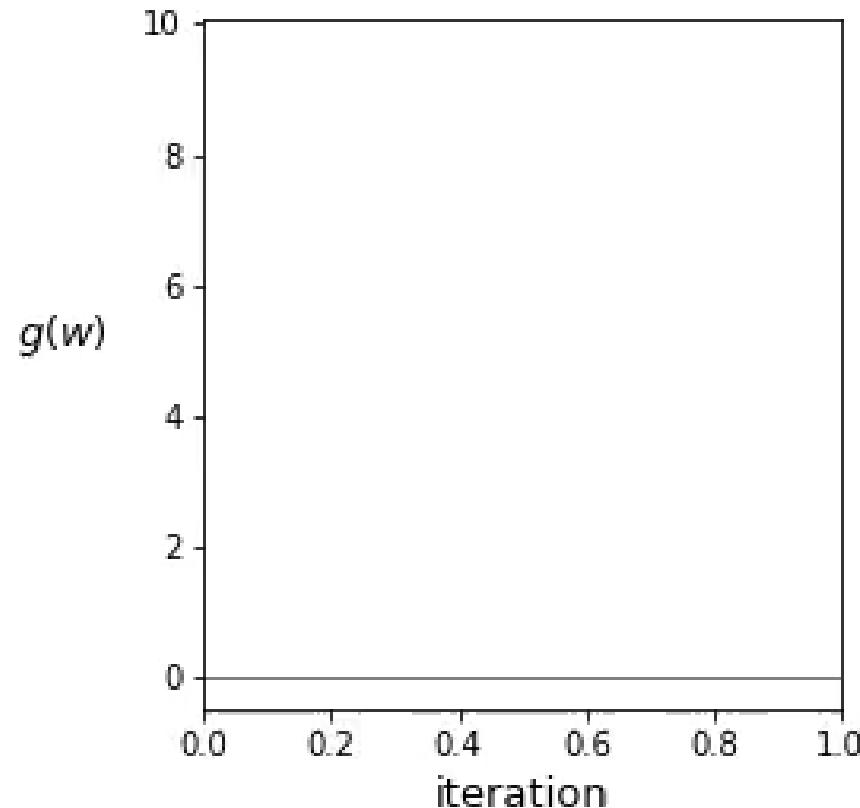
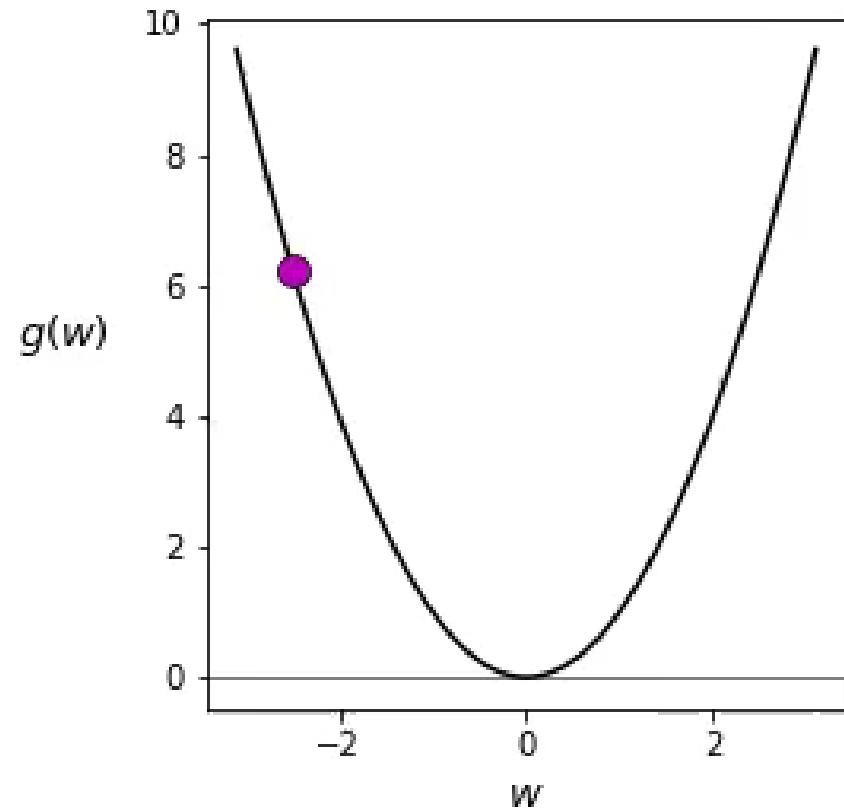
$$w_0 := w_0 - \alpha \cdot ((w_1 x^{(n)} + w_0) - y^{(n)})$$

$$w_1 := w_1 - \alpha \cdot ((w_1 x^{(n)} + w_0) - y^{(n)}) x^{(n)}$$

Until change remains below a very small threshold

Return  $w_0, w_1$

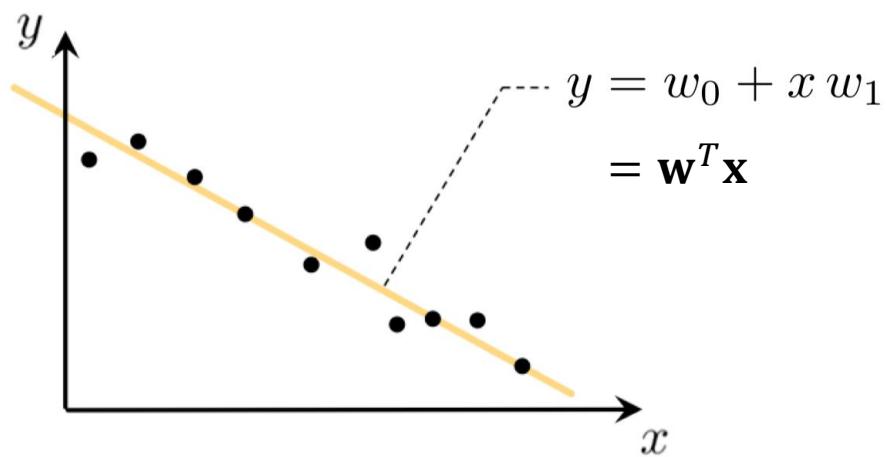
# Effect of the learning rate



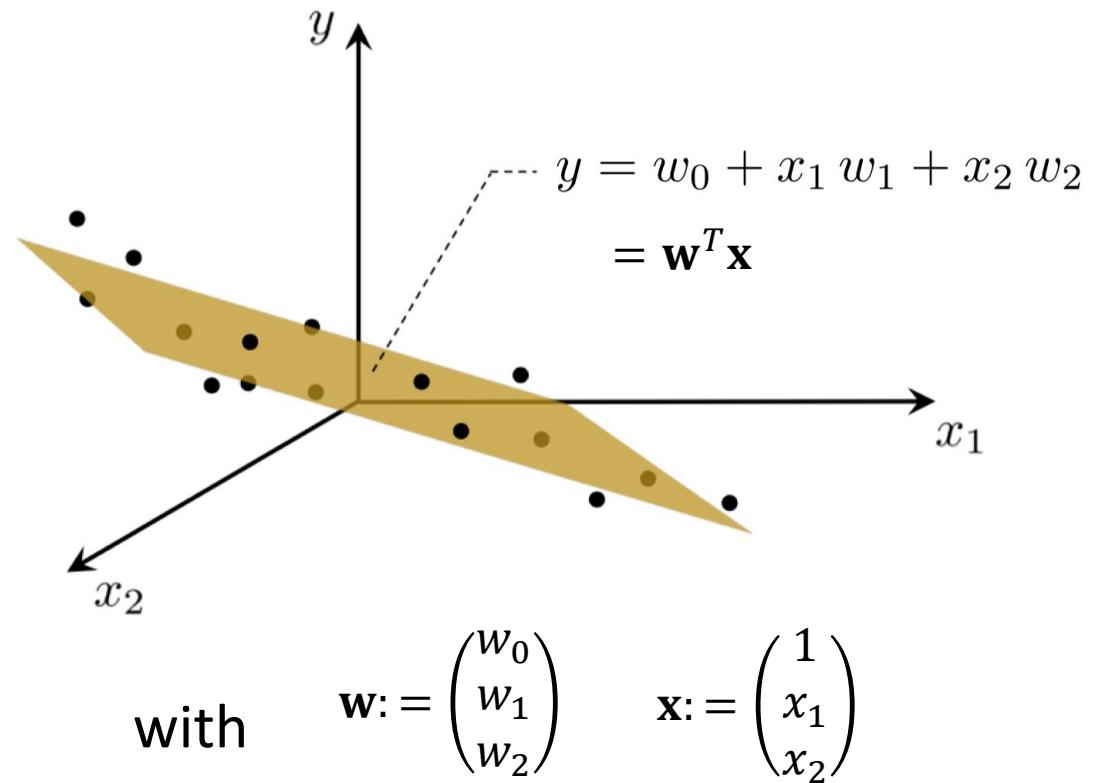
# Extensions & variants of regression problems

- We change the model
- The loss and cost function remains the same

# Multivariate linear regression



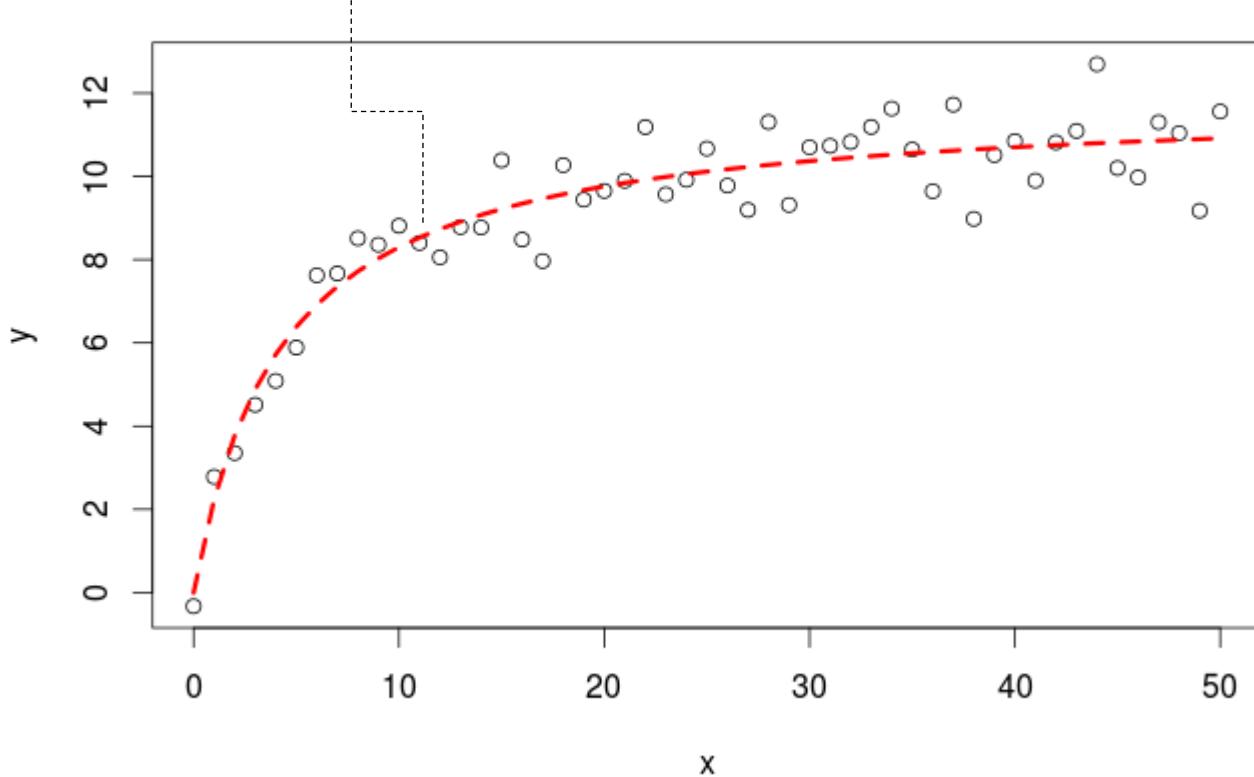
with  $\mathbf{w} := \begin{pmatrix} w_0 \\ w_1 \end{pmatrix}$        $\mathbf{x} := \begin{pmatrix} 1 \\ x \end{pmatrix}$



with  $\mathbf{w} := \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}$        $\mathbf{x} := \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix}$

# Univariate nonlinear regression

$$y = w_0 + w_1x + w_2x^2 + w_3x^3 + \dots + w_mx^m = \mathbf{w}^T \mathbf{x} \quad \text{with}$$



$$\mathbf{w} := \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \dots \\ w_m \end{pmatrix} \quad \mathbf{x} := \begin{pmatrix} 1 \\ x \\ x^2 \\ \dots \\ x^m \end{pmatrix}$$

This is an  $m$ -th order polynomial regression model.

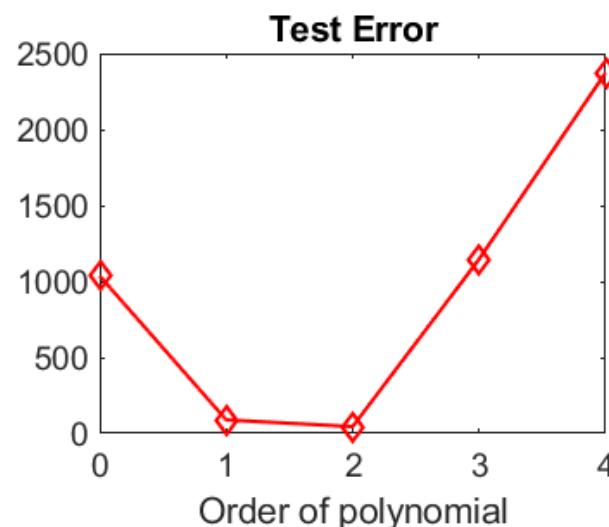
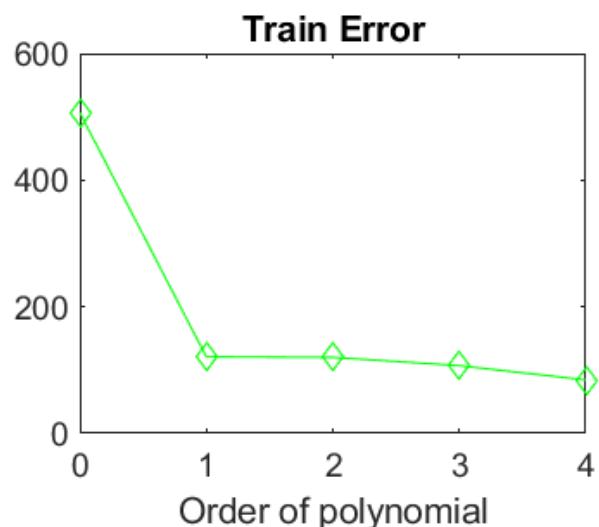
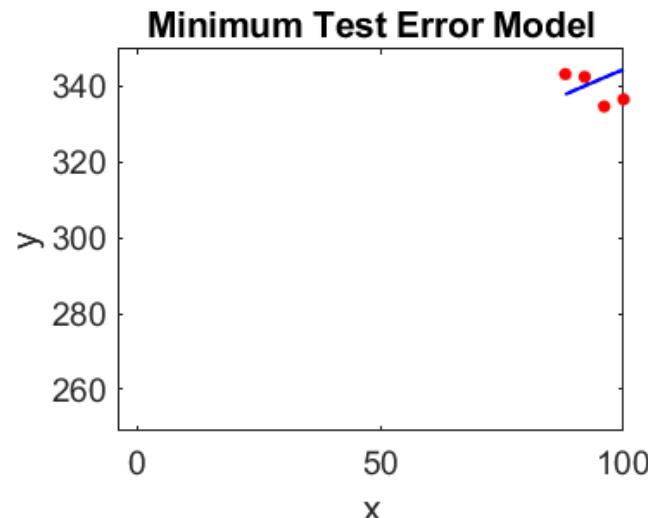
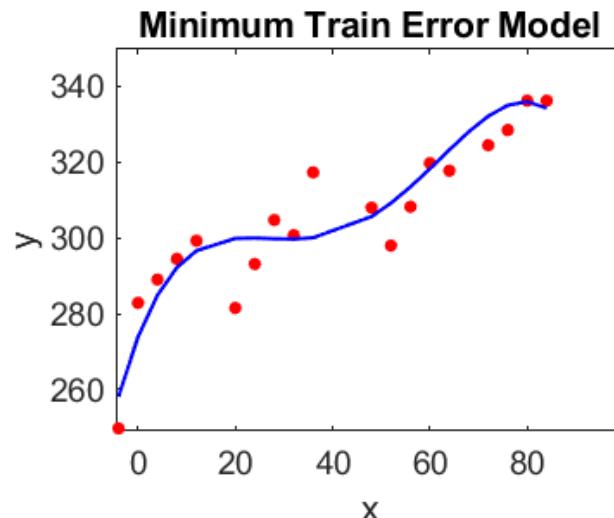
# Advantages of vector notation

- Vector notation is concise
- With the vectors  $\mathbf{w}$  and  $\mathbf{x}$  populated appropriately (and differently in each case, as on the previous 2 slides), these models are still linear in the parameter vector.
- The cost function is the L2 as before
- So the gradient in both cases is:

$$\nabla g(\mathbf{w}) = 2(\mathbf{w}^T \mathbf{x}^{(n)} - y^{(n)})\mathbf{x}^{(n)}$$

- Ready to be plugged into the general gradient descent algorithm

# Don't get too carried away with nonlinearity



# Reference Acknowledgement

Several figures and animations on these slides are taken from:

- Jeremy Watt et al. Machine Learning Refined. Cambridge University Press, 2020.

[https://github.com/jermwatt/machine\\_learning\\_refined](https://github.com/jermwatt/machine_learning_refined)

# Logistic Regression

Ata Kaban

# What is logistic regression?

- It is a linear model **for classification** (contrary to its name!)

Recall the difference:

- In **regression**, the targets are real values
- In **classification**, the targets are categories, and they are called **labels**

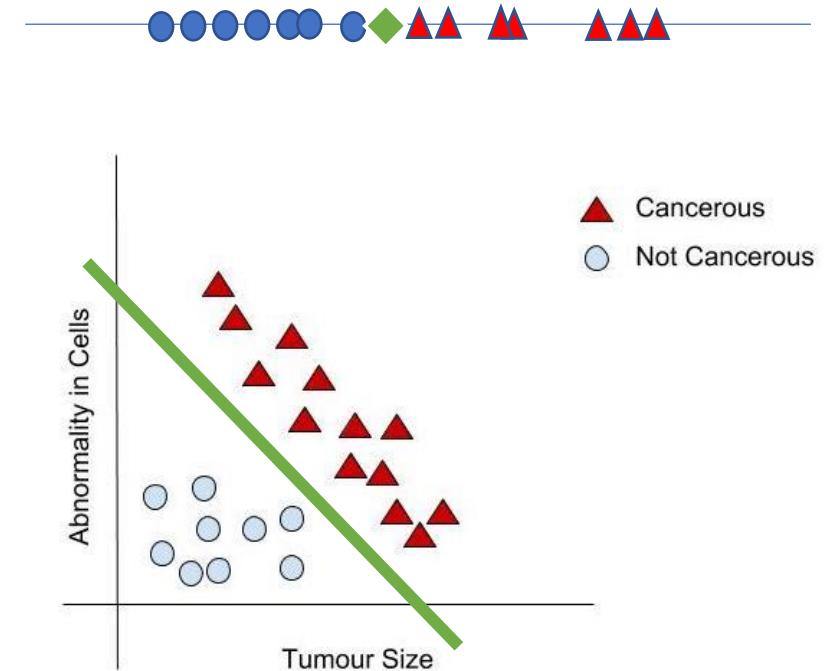
# Logistic regression - outline

We will go through the same conceptual journey as before:

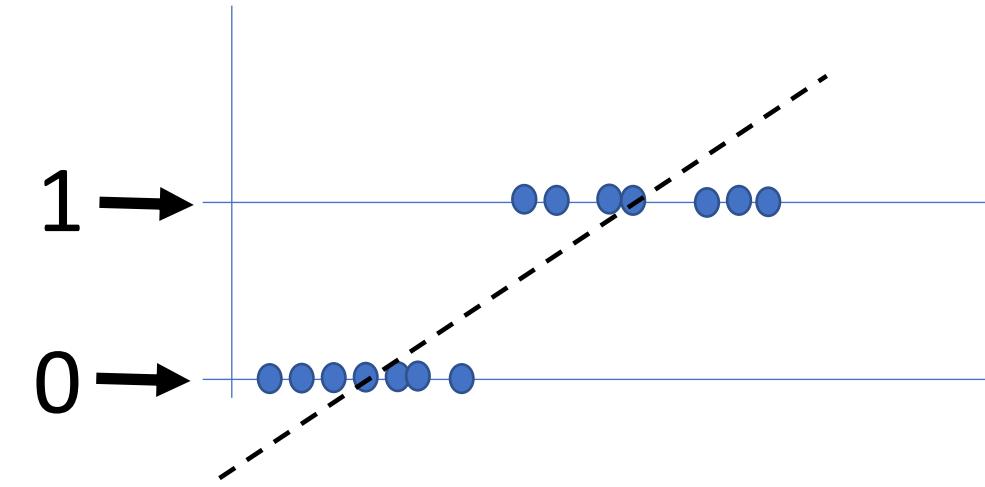
- 1) Model formulation
- 2) Cost function
- 3) Learning algorithm by gradient descent

# 1) Model

- We want to put a boundary between 2 classes
- If  $x$  has a single attribute, we can do it with a point
- If  $x$  has 2 attributes, we can do it with a line
- If  $x$  has 3 attributes, we can do it with a plane
- If  $x$  has more than 3 attributes, we can do it with a hyperplane (can't draw it anymore)
- If the classes are linearly separable, the training error will be 0.



Q: Can you plug classification data into linear regression?



A: Yes. But it might not perform very well. No ordering between categories, like there is between real numbers. We need a better model

# Model

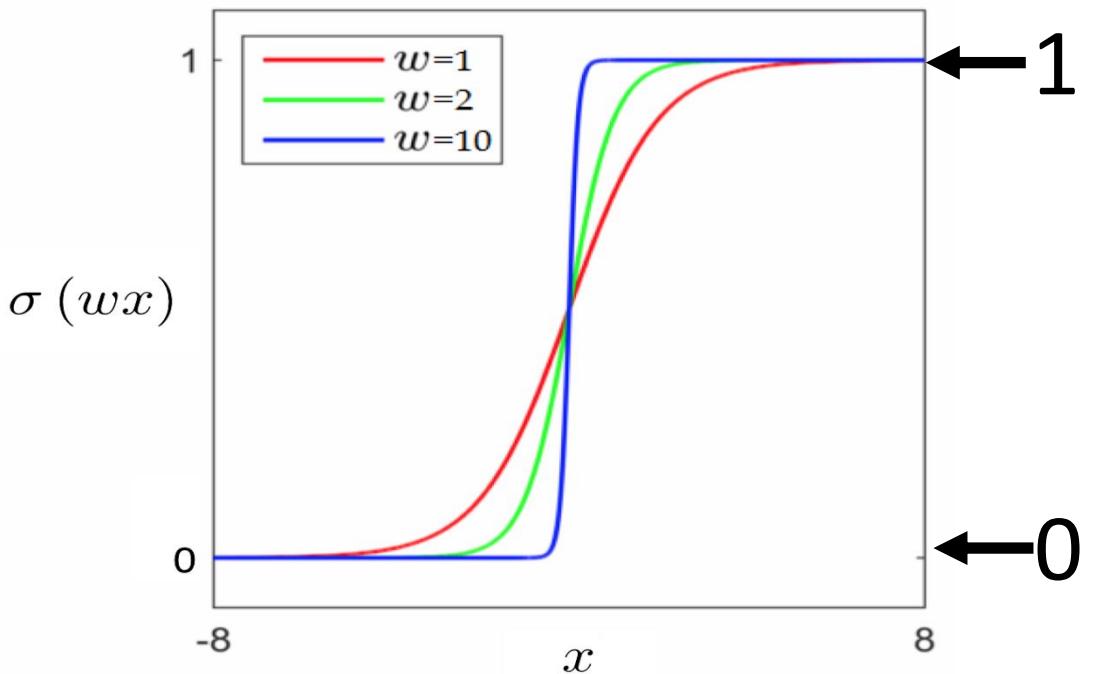
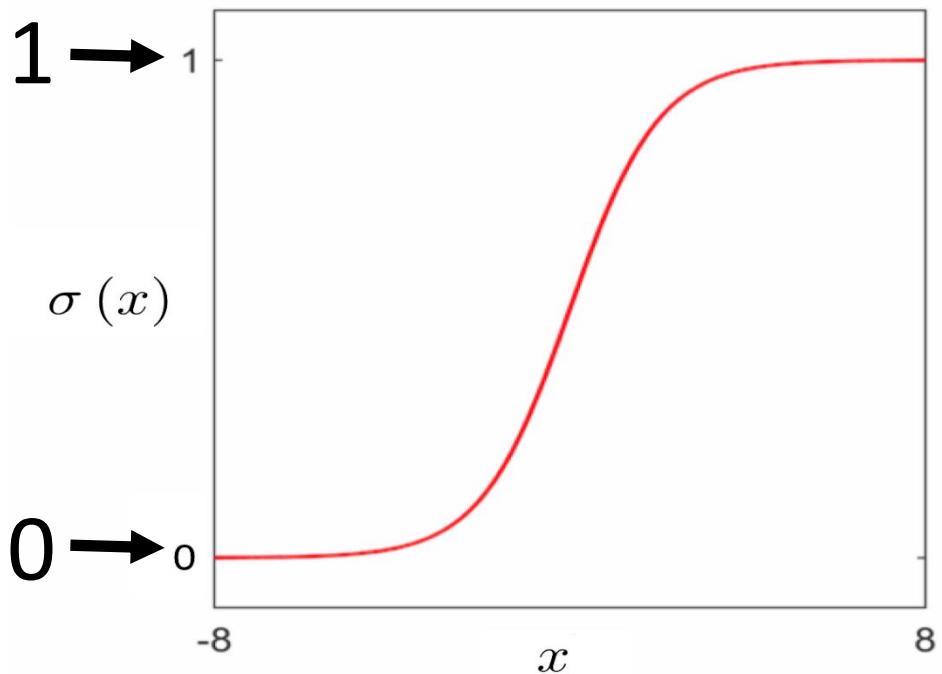
We change the linear model slightly by passing it through a nonlinearity

- If  $x$  has 1 attribute, we will have

$$h(x; \mathbf{w}) = \sigma(w_0 + w_1 x) = \frac{1}{1+e^{-(w_0+w_1x)}}$$

The function  $\sigma(u) = \frac{1}{1+e^{-u}}$  is called the sigmoid function or logistic function

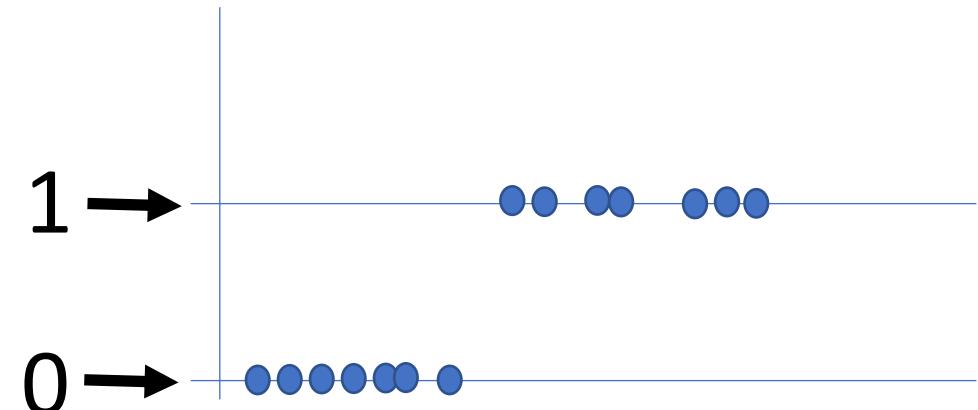
# Sigmoid function



It is a smoothed version of a step function – note the step function would make optimisation difficult.

# Play around with the logistic model

- Go to <https://www.desmos.com/calculator>
- Type:  $y = \frac{1}{1+\exp(-(w_0+w_1x))}$
- Change the values of the free parameters to see their effect
- Imagine how this function could fit this data better than a line did.
- What if your data happens to have class 1 on the left, and class 0 on the right?
  - $w_1$  can be negative, so the same model works.



# Model

- If  $\mathbf{x}$  has  $d$  attributes, that is  $\mathbf{x} = (x_1, x_2, \dots, x_d)$ , we will write

$$h(\mathbf{x}; \mathbf{w}) = \sigma(w_0 + w_1 x_1 + \dots + w_d x_d) = \frac{1}{1+e^{-(\mathbf{w}^T \mathbf{x})}}, \text{ where:}$$

all components of  $\mathbf{w}$  are free parameters

$$\mathbf{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \dots \\ w_d \end{pmatrix} \quad \mathbf{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \dots \\ x_d \end{pmatrix} \in R^d$$

# Meaning of the sigmoid function

- The sigmoid function takes a single argument (note,  $\mathbf{w}^T \mathbf{x}$  is one number).
- It always returns a value between 0 and 1. The meaning of this value is the **probability that the label is 1**.

$$\sigma(\mathbf{w}^T \mathbf{x}) = P(y = 1 | \mathbf{x}; \mathbf{w})$$

- If this is smaller than 0.5 then we predict label 0.
- if this is larger than 0.5 then we predict label 1.
- There is a slim chance that the sigmoid outputs exactly 0.5. The set of all possible inputs for which this happens is called the **decision boundary**.

# Check your understanding

- Can you express the probability that the label is 0 using sigmoid?

$$\sigma(\mathbf{w}^T \mathbf{x}) = P(y = 1 | \mathbf{x}; \mathbf{w})$$

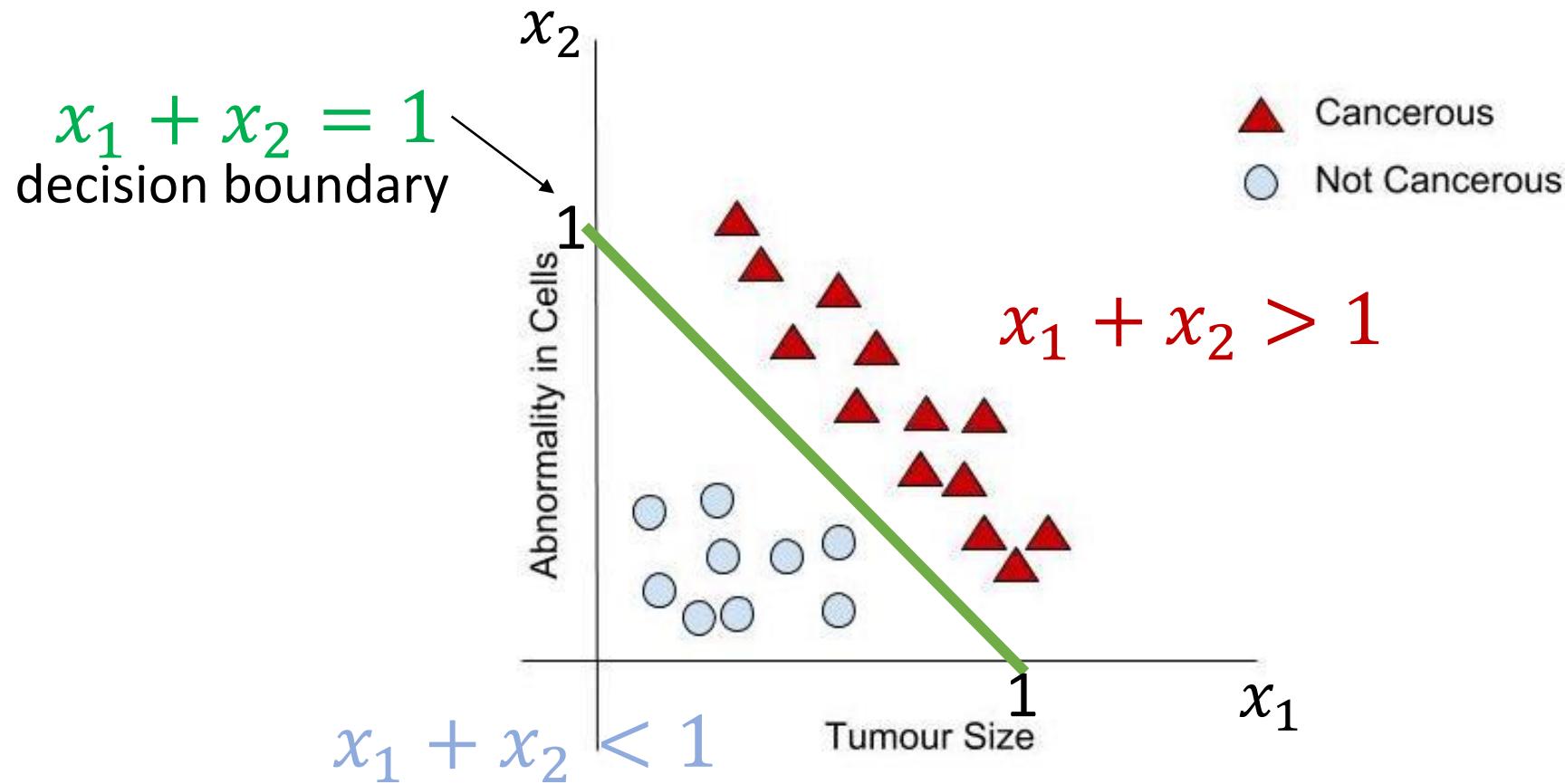
$$\Rightarrow 1 - \sigma(\mathbf{w}^T \mathbf{x}) = 1 - P(y = 1 | \mathbf{x}; \mathbf{w}) = P(y = 0 | \mathbf{x}; \mathbf{w})$$

- In fact we can write both in 1 line as:

$$P(y | \mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^T \mathbf{x})^y (1 - \sigma(\mathbf{w}^T \mathbf{x}))^{1-y} // y \text{ given } \mathbf{x} \text{ has a Bernoulli distribution}$$

# Worked example

- Suppose we have 2 input attributes, so our model is  
$$h(\mathbf{x}; \mathbf{w}) = \sigma(w_0 + w_1 x_1 + w_2 x_2).$$
- Suppose we know that  $w_0 = -1, w_1 = 1, w_2 = 1$ .
- When do we predict 1? What is the decision boundary?
  - We predict 1 precisely when  $P(y = 1 | \mathbf{x}; \mathbf{w}) > 0.5$ . That is, when  $h(\mathbf{x}; \mathbf{w}) > 0.5$ .
  - This happens precisely when the argument of the sigmoid is positive!
  - Decision boundary:  $-1 + x_1 + x_2 = 0$  This is a line
- Q: Is the decision boundary of logistic regression always linear?  
A: Yes.



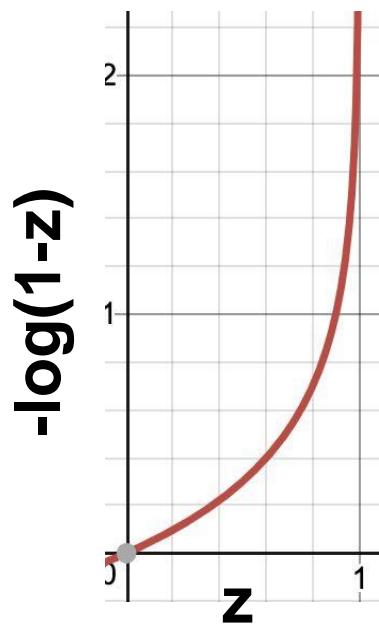
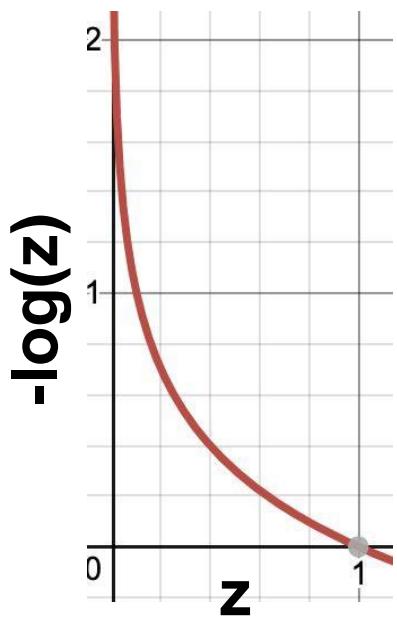
## 2) Cost function

- We need a new cost function, because the Mean Square Error used in linear regression produces a very wiggly function with the new hypothesis function, which would be difficult to optimise.
- But as before we will still have that:
  - each data point contributes a cost, and the overall cost function is the average of these
  - the cost is a function of the free parameters of the model

# Logistic cost function

For each  $(x, y)$  pair,  $Cost(h(x; w), y) =$

$$z \begin{cases} -\log(h(x; w)), & \text{if } y = 1 \\ -\log(1 - h(x; w)), & \text{if } y = 0 \end{cases}$$



Overall cost:  $g(w) = \frac{1}{N} \sum_{n=0}^N Cost(h(x^{(n)}; w), y^{(n)})$  convex (easy to minimise)

# Writing the cost function in a single line

$$g(\mathbf{w}) = \frac{1}{N} \sum_{n=0}^N Cost(h(\mathbf{x}^{(n)}; \mathbf{w}), y^{(n)})$$

$$Cost(h(\mathbf{x}; \mathbf{w}), y) = \begin{cases} -\log(h(\mathbf{x}; \mathbf{w})), & \text{if } y = 1 \\ -\log(1 - h(\mathbf{x}; \mathbf{w})), & \text{if } y = 0 \end{cases}$$

$$g(\mathbf{w}) = -\frac{1}{N} \sum_{n=1}^N (y^{(n)} \log h(\mathbf{x}^{(n)}; \mathbf{w}) + (1 - y^{(n)}) \log(1 - h(\mathbf{x}^{(n)}; \mathbf{w})))$$

This is also called the **cross-entropy**.

# Logistic regression – what we want to do

- Given training data

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(N)}, y^{(N)})$$

- Fit the model

$$y = h(x; \mathbf{w}) = \sigma(\mathbf{w}^T \mathbf{x})$$

- By minimising the cross-entropy cost function

$$g(\mathbf{w}) = -\frac{1}{N} \sum_{n=1}^N (y^{(n)} \log h(x^{(n)}; \mathbf{w}) + (1 - y^{(n)}) \log(1 - h(x^{(n)}; \mathbf{w})))$$

### 3) Learning algorithm by gradient descent

- We use gradient descent (again!) to minimise the cost function, i.e. to find the best weight values.
- The gradient vector is\*:

$$\nabla g(\mathbf{w}) = -(y^{(n)} - h(\mathbf{x}^{(n)}; \mathbf{w})) \cdot \mathbf{x}^{(n)}$$

$$\mathbf{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \dots \\ w_d \end{pmatrix} \quad \mathbf{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \dots \\ x_d \end{pmatrix} \in R^d$$

We plug this into the general gradient descent algorithm given last week.

--

\* This follows after differentiating the cost function w.r.t. weights – we omit the lengthy math!

# Learning algorithm for logistic regression

While not converged

For  $n = 1, \dots, N$  // each example in the training set

$$\mathbf{w} = \mathbf{w} + \alpha(y^{(n)} - h(\mathbf{x}^{(n)}; \mathbf{w})) \cdot \mathbf{x}^{(n)}$$

# Learning algorithm for logistic regression

The same, written component-wise:

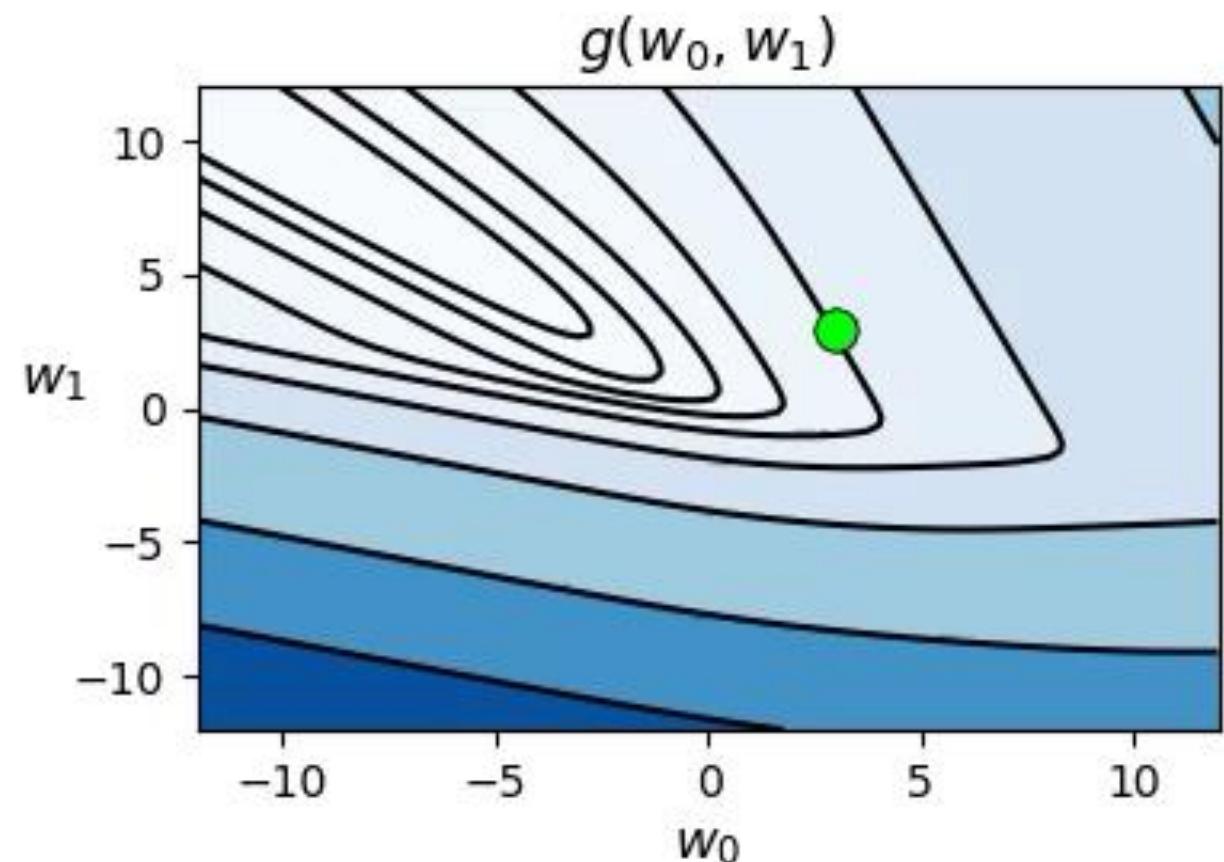
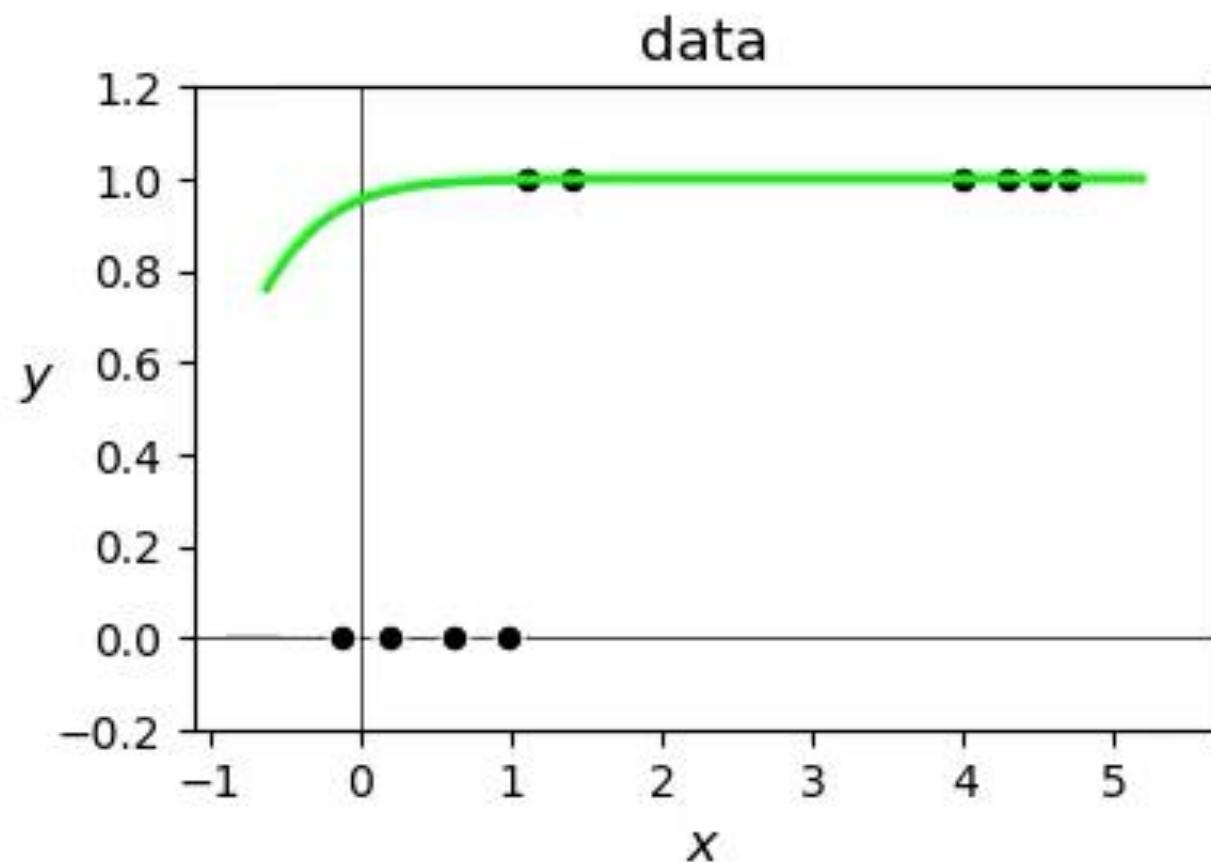
While not converged

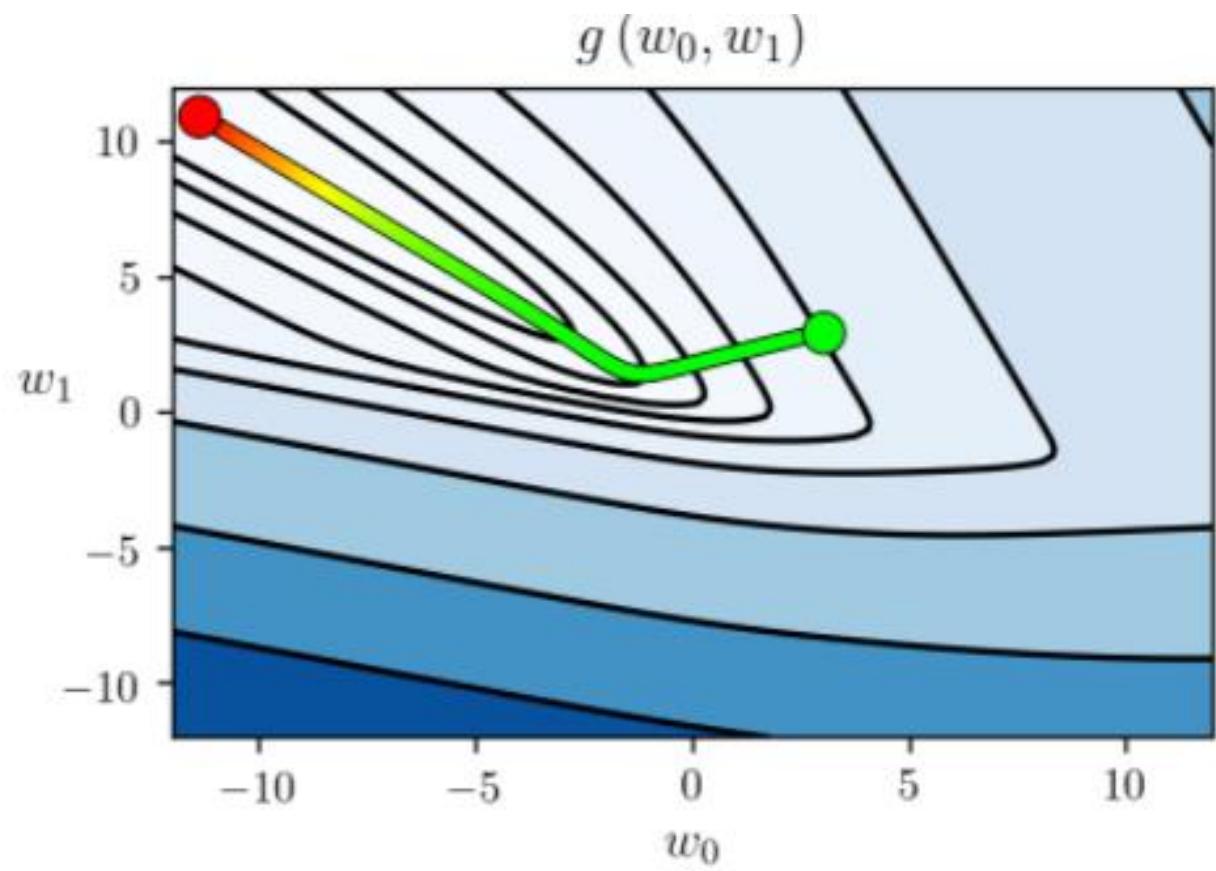
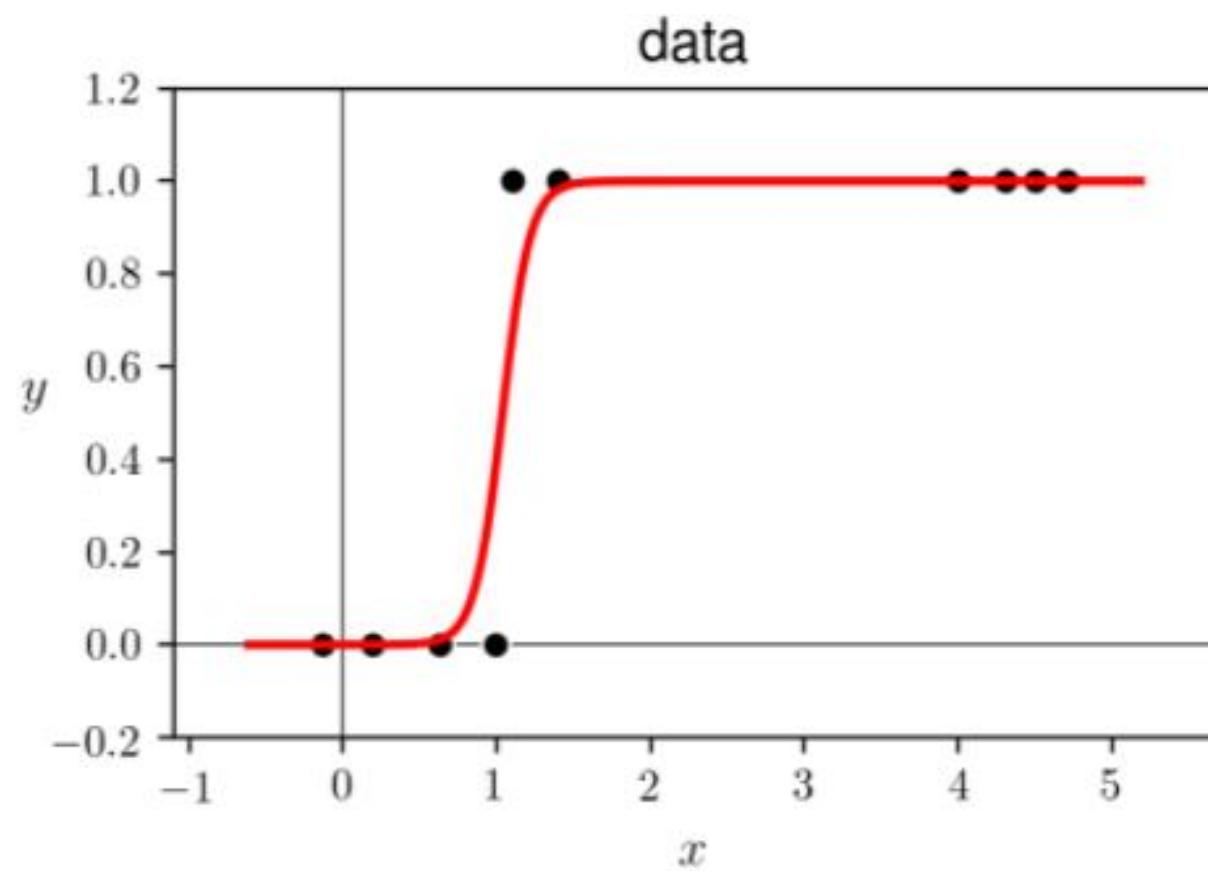
For  $n = 1, \dots, N$  // each example in the training set

$$w_0 = w_0 + \alpha(y^{(n)} - h(\mathbf{x}^{(n)}; \mathbf{w}))$$

For  $i=1, \dots, d$

$$w_i = w_i + \alpha(y^{(n)} - h(\mathbf{x}^{(n)}; \mathbf{w}))x_i^{(n)}$$





# Extensions

- We studied logistic regression for linear binary classification
- There are extensions, such as:
  - Nonlinear logistic regression: instead of linear function inside the exp in the sigmoid, we can use polynomial functions of the input attributes
  - Multi-class logistic regression: uses a multi-valued version of sigmoid
- Details of these extensions are beyond of our scope in this module

# Examples of application of logistic regression

- Face detection: classes consist of images that contain a face and images without a face
- Sentiment analysis: classes consist of written product-reviews expressing a positive or a negative opinion
- Automatic diagnosis of medical conditions: classes consist of medical data of patients who either do or do not have a specific disease

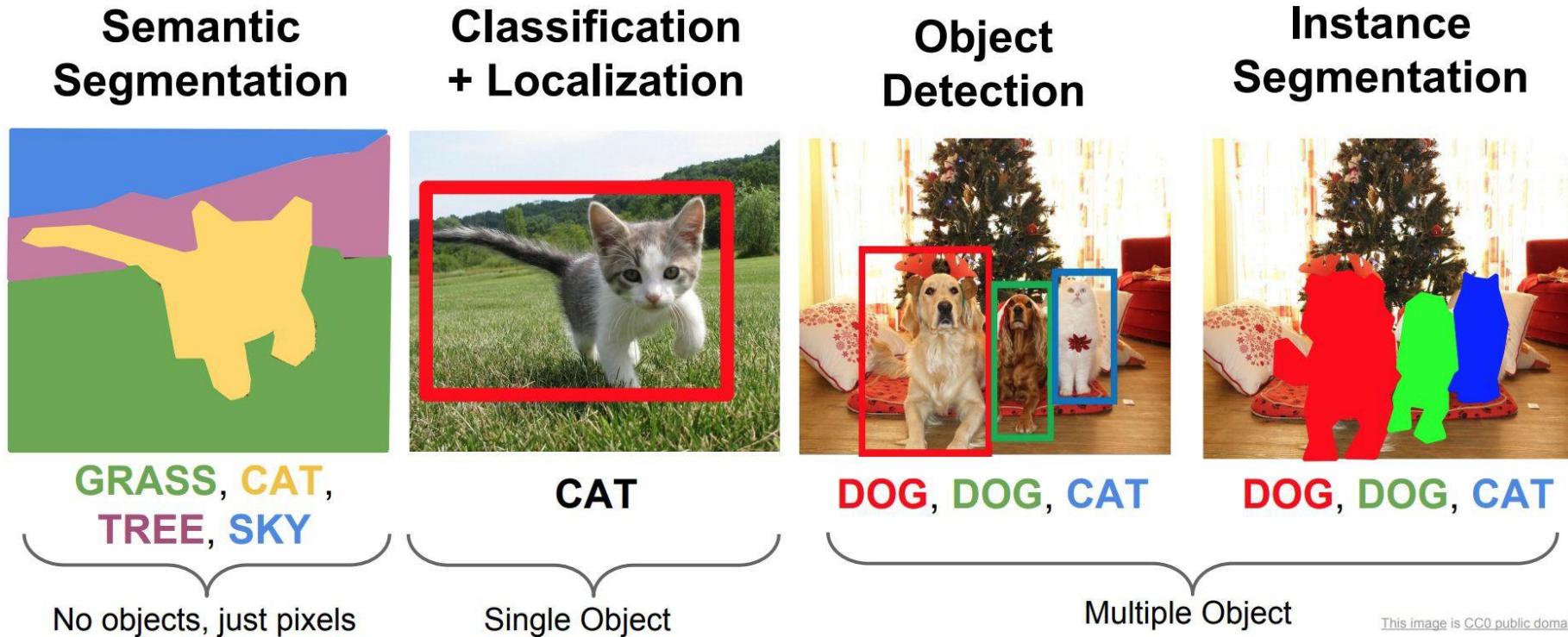
# Neural Networks

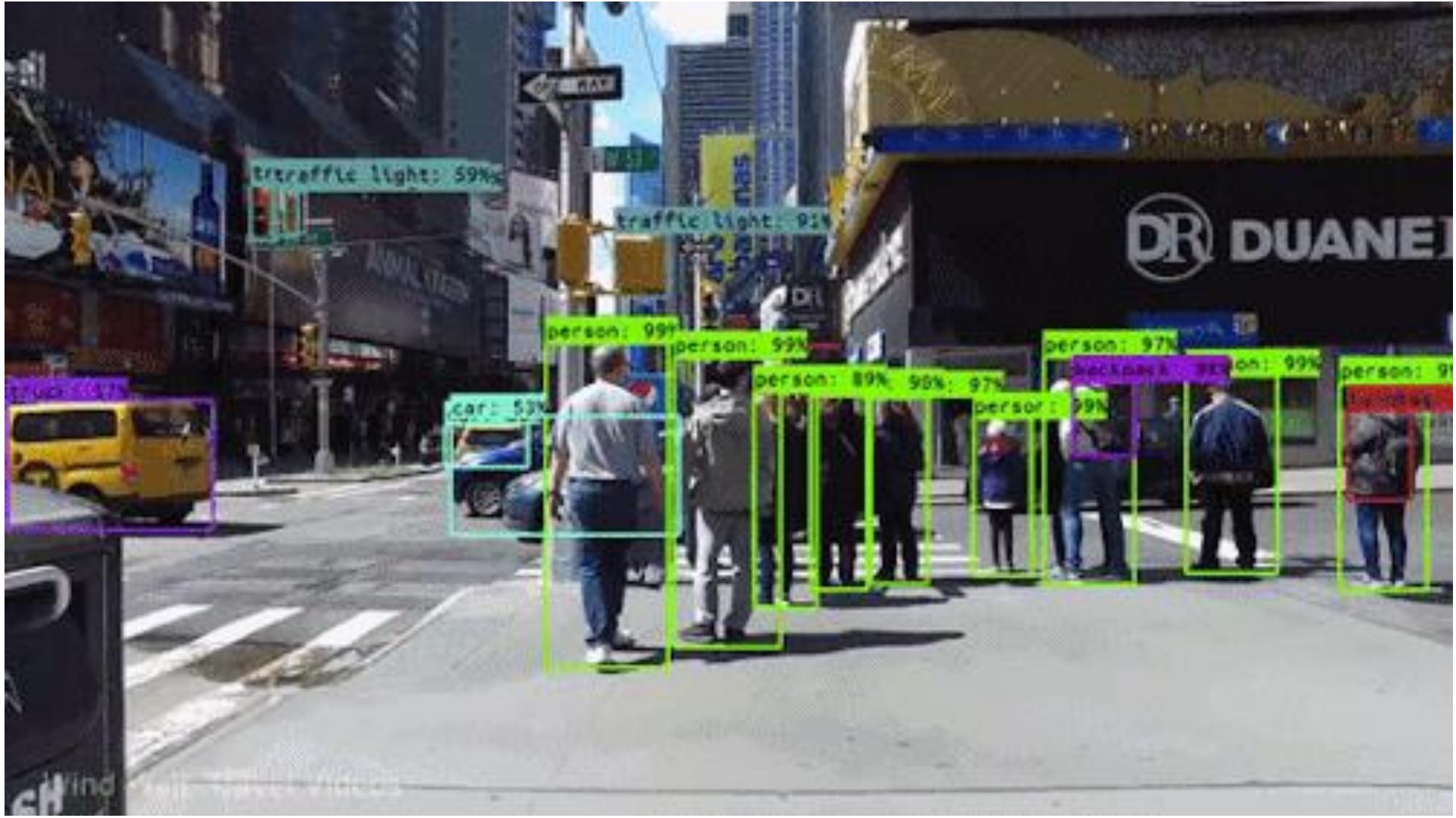
Ata Kaban

# What are neural networks?

- Highly nonlinear models having many free parameters
- Can be used for either regression and classification depending on the choice of loss function.
- Can replace nonlinear regression and nonlinear logistic regression which are less practical

# Motivation





# Classification



255, 255, 255 255	255, 255, 255 255	255, 255, 255 255	255, 255, 255 255	255, 255, 255 255
255, 255, 255 255	255, 255, 255 255	255, 255, 255 255	255, 255, 255 255	255, 255, 255 255
255, 255, 255 255	255, 255, 255 255	255, 255, 255 255	255, 255, 255 255	255, 255, 255 255

**Image**

**Computer Input**  
**255 255 255 for each pixel**

# Neural networks - outline

Using neural networks requires the same conceptual journey as before:

- 1) Model formulation
- 2) Cost function
- 3) Learning algorithm by gradient descent

# Neural networks - outline

Using neural networks requires the same conceptual journey as before:

- 1) Model formulation
- 2) Cost function -- nothing new here
  - for regression: Mean square error between predictions and observed targets
  - for classification: Logistic loss (also called cross-entropy)
- 3) Learning algorithm by gradient descent

# Neural networks - outline

Using neural networks requires the same conceptual journey as before:

1) Model formulation

2) Cost function

3) Learning algorithm by gradient descent

- The update rules are non-trivial, because the models are much more complex
- It is performed by an algorithm called “Backpropagation”
- Conceptually, each iteration of Backprop takes a gradient descent step
- Implementations exist that are able to compute the gradient automatically

# Neural networks - outline

Using neural networks requires the same conceptual journey as before:

## 1) Model formulation

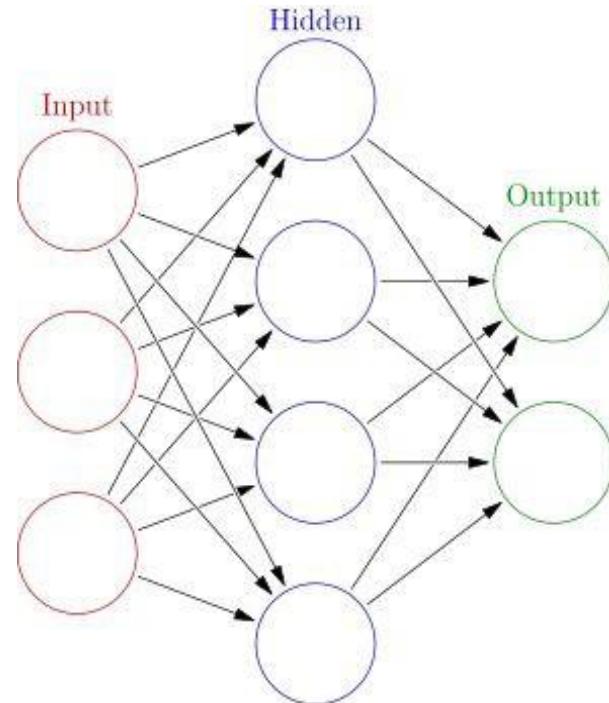
- Sometimes called “architecture”
- Designing this for the problem at hand is the main challenge

## 2) Cost function

## 3) Learning algorithm by gradient descent

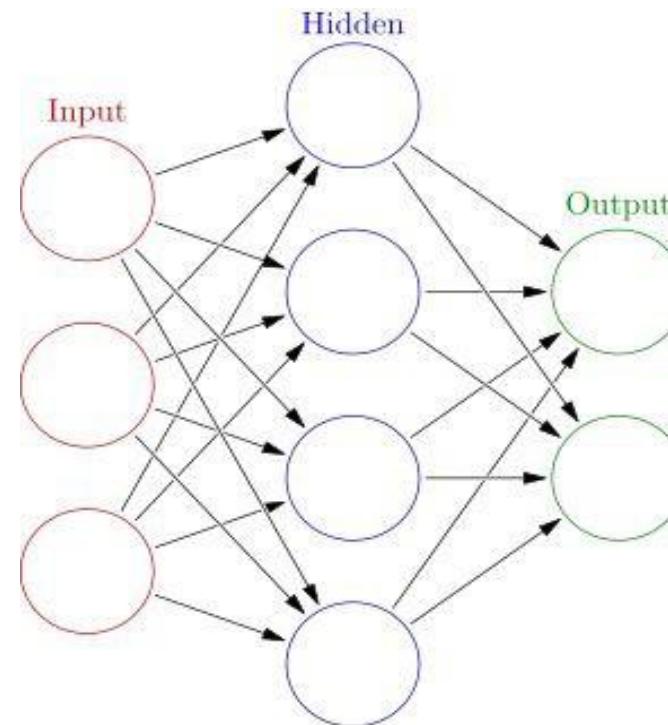
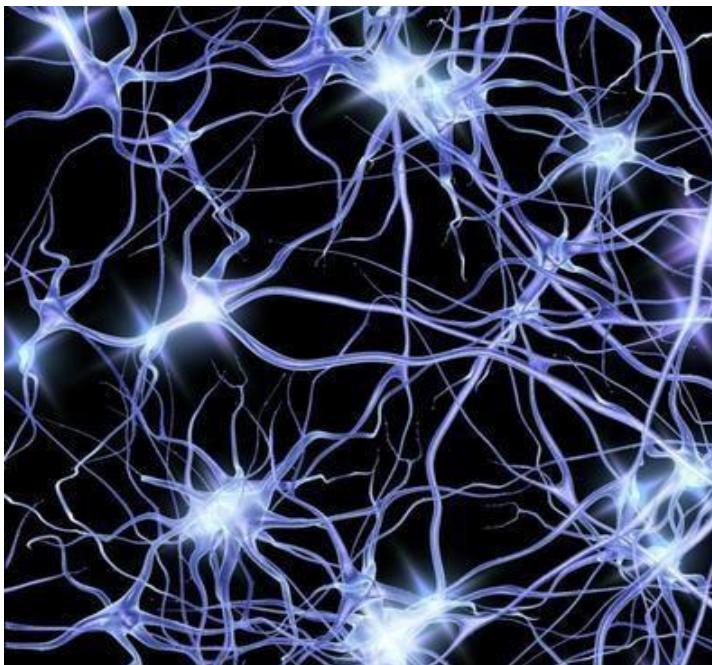
# Neural network model

- Roughly, we can think of a neural network model as being composed of several logistic regression units

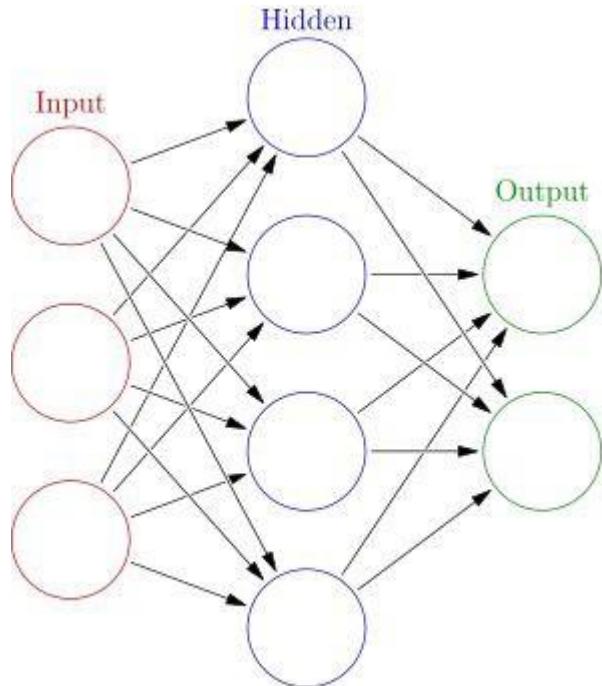


# Neural networks – the name

Inspiration from networks of neurons in the brain – each unit is analogous to a neuron.



# Building blocks of a feedforward neural net

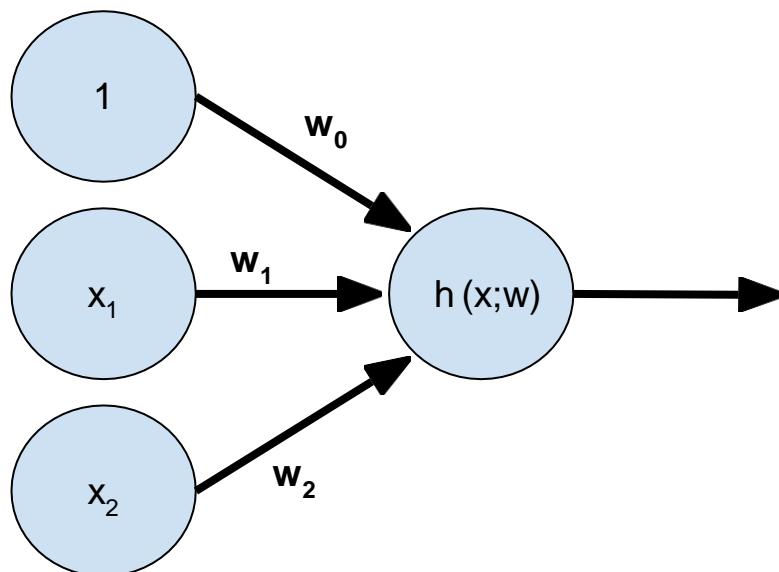


- Each node is one **unit or neuron**
- Each arrow is a connection with a **weight**
- Nodes are arranged in **layers**
  - One input layer
  - One output layer
  - Any number of hidden layers (0,1,2,...)
- Hidden & output nodes typically apply a sigmoid, or other **activation function**

# Simplest neural net

- A neural net with 0 hidden layers is called a **perceptron**
- If the activation function is the sigmoid, then this model is equivalent to a logistic regression

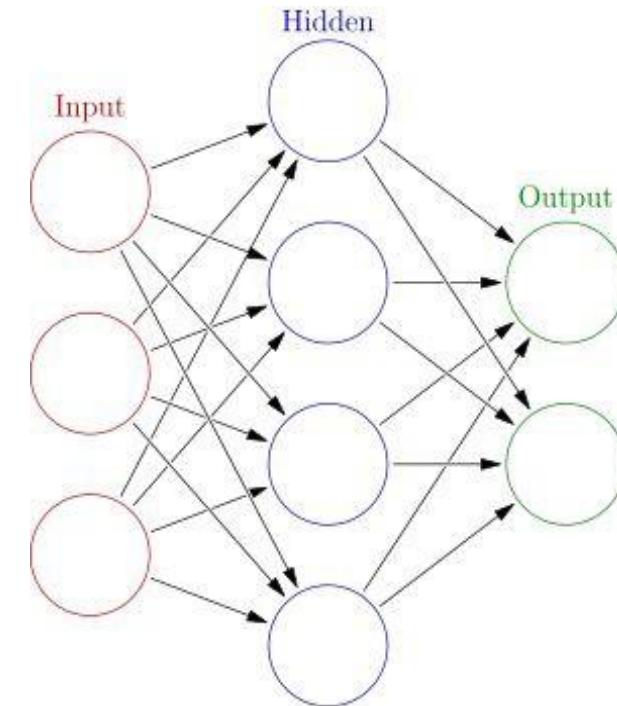
$$h(x, w) = \sigma(w_0 + w_1 x_1 + w_2 x_2)$$



- The type of computation performed by each non-input node is the same in multi-layer networks too.
- The choice of activation function can be different

# Multi-layer perceptron

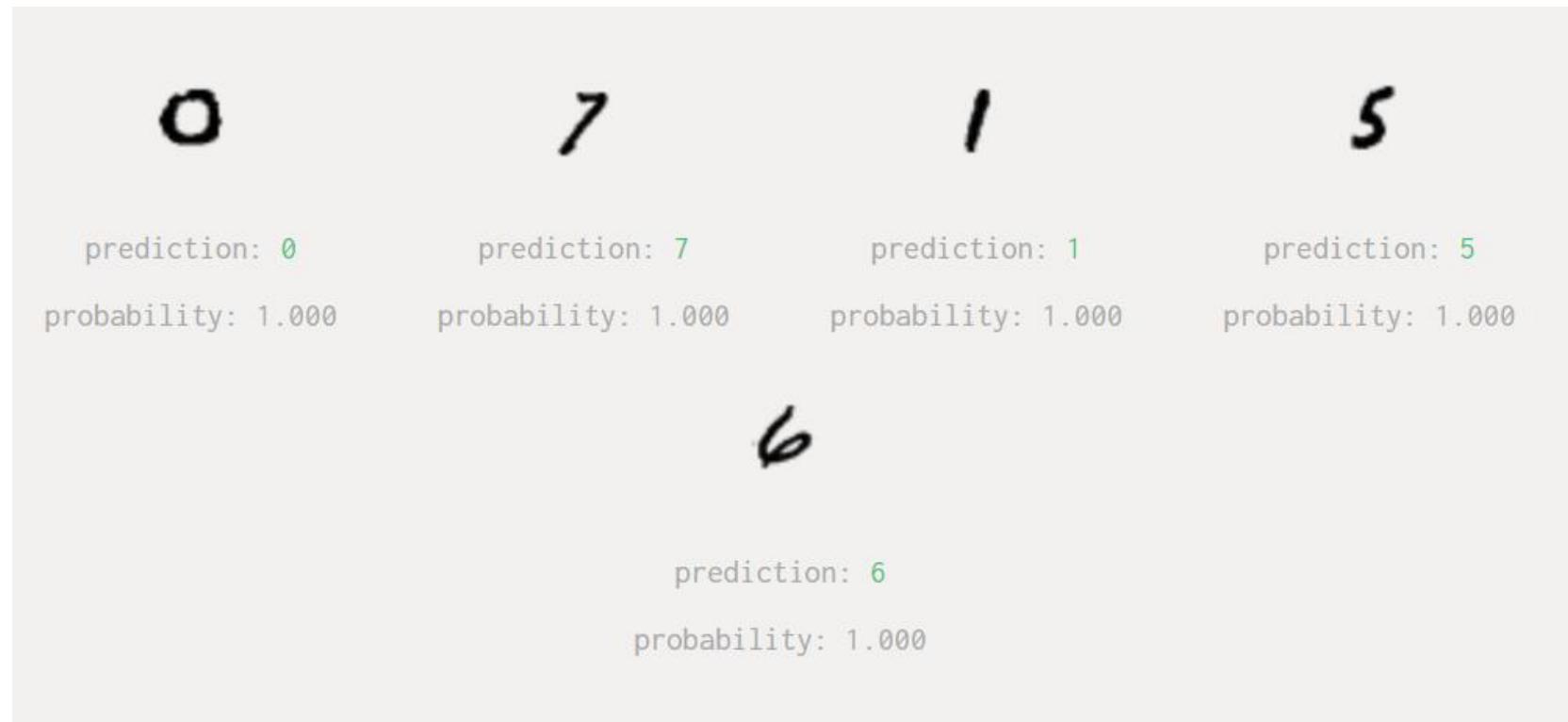
- When we have one hidden layer, the model is called multi-layer perceptron
- It is a truly non-linear model.
- How many free parameters does it have?
  - Weights = parameters
  - Number of hidden units, choice of activation function = hyperparameters
- Number of output nodes = number of targets or labels we want to predict



In theory, with sufficient number of hidden units this type of net is able to approximate any function (cf. Cybenko 1989)

# Handwriting recognition

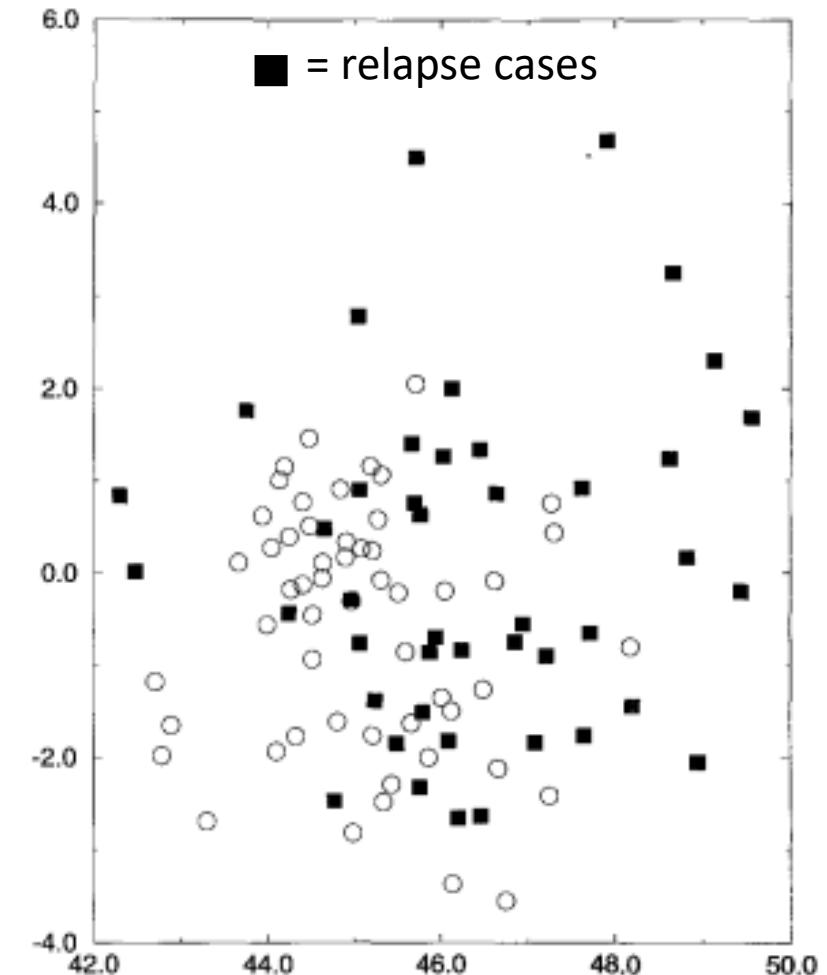
- MLP is well suited to learning from sensor data
- MLP can achieve approximately 98% accuracy in handwriting recognition on the MNIST data set



# Multilayer perceptron for clinical data

Neural network prediction of relapse in breast cancer patients (paper by Tarassenko et al., 1996)

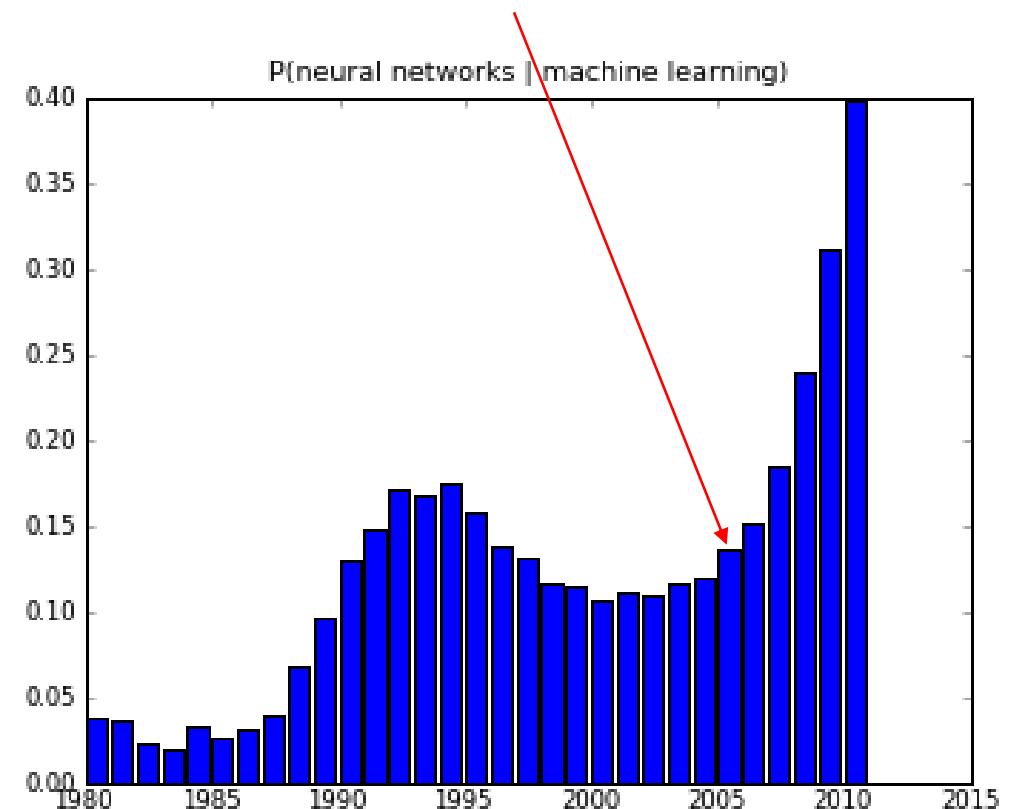
- **Target:** relapse/no within 3 years
- **Attributes:** Age, Tumour size, nr.Nodes, Log er, Log egfr
- **Data:** 350 patients
- **Architecture:** 1 hidden layer
- **Results:** 72% classification accuracy



# Deep neural networks – Deep Learning

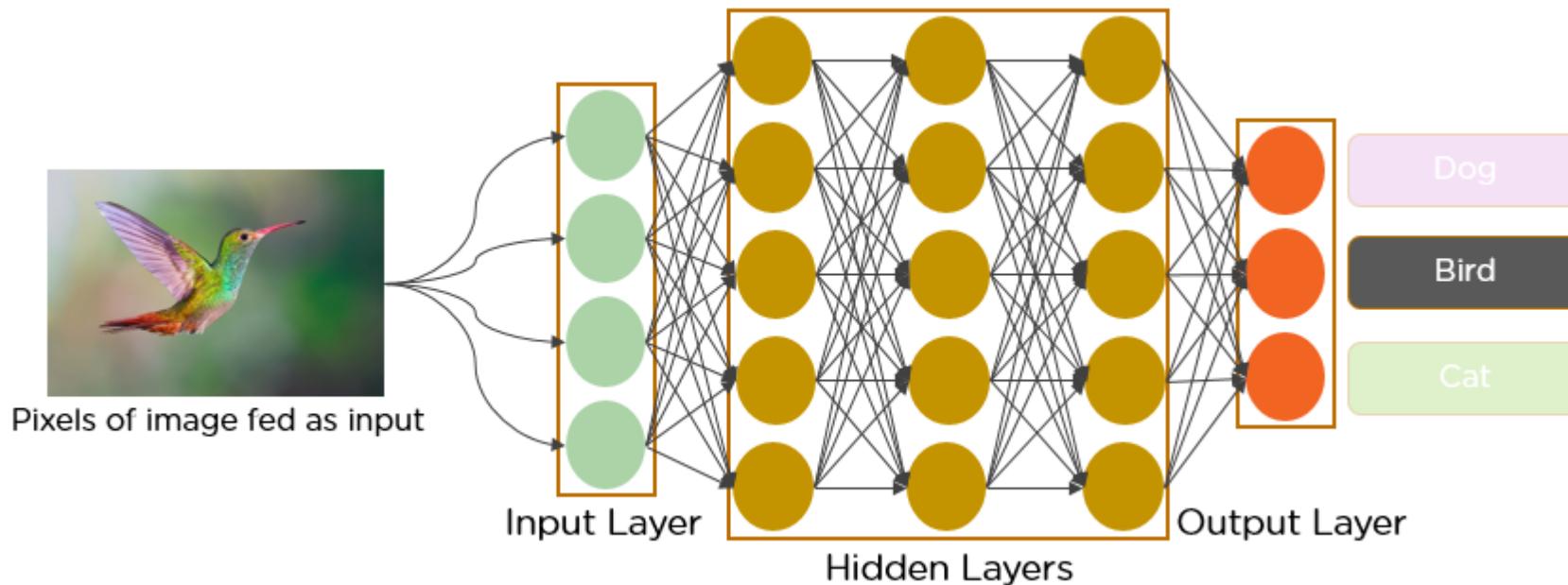
- Proportion of machine learning papers that contain the term ‘neural networks’
- Popular in the 90s
- Dip in the 2000s
- The problem was that training fully connected neural nets is difficult
- Re-emergence of the topic of neural networks with the advent of deep learning

A Fast Learning Algorithm for Deep Belief Nets, Hinton, 2006



# Deep neural networks

- Very simply, deep learning is machine learning using neural networks that have multiple hidden layers.
- Number of hidden layers is another hyperparameter.



# Activation functions

- Recall the perceptron where neurons use the sigmoid function on the weighted sum of their input.
- This function used on the weighted sum of the input is called the activation function.
  - Sigmoid is classic, but it is not the only activation function in use
  - Others include: tanh, ReLu (“rectified linear unit”)
  - A full list can be found at  
[https://en.wikipedia.org/wiki/Activation\\_function#Comparison\\_of\\_activation\\_functions](https://en.wikipedia.org/wiki/Activation_function#Comparison_of_activation_functions)

# Why “Deep”?

- In theory, any classification function can be learned with 1 hidden layer if enough hidden units, why do we need deep learning?
  - With just one hidden layer, you might need very large number of neurons, and training is time-consuming
  - More hidden layer work better in practice
  - Analogy with human vision - early layers correspond to primitive features (e.g. straight lines), late layers correspond to higher-level components (e.g. eye, nose, mouth).
  - No need to hand-craft the input attributes that describe the data

# Deep Learning & the ImageNet Challenge

- ImageNet - a large visual database for visual object recognition
- Classify 150K images into 1,000 categories (e.g. Egyptian cat, gazelle, wok, photocopier)
- 5 guesses allowed per picture
- In 2012, AlexNet, a ‘deep’ Neural Network won by a huge margin, achieving 12% error
- By now, the best is around 3% error
- You can learn about ImageNet here:  
<https://thephotographersgallery.org.uk/whats-on/imagenet-10th-birthday-party>

# Extensions

- Many variants of neural networks – very active area of research
- Generative Adversarial Neural Networks (GAN)
  - Generator network can produce new samples (e.g. new faces)
  - Neural Network discriminator classifies face as ‘real’ or ‘fake’
  - Generator weights are updated based on how well it fools the discriminator
- See if you can do better than a GAN:  
<http://www.whichfaceisreal.com/>

# Overfitting

# Training data and test data

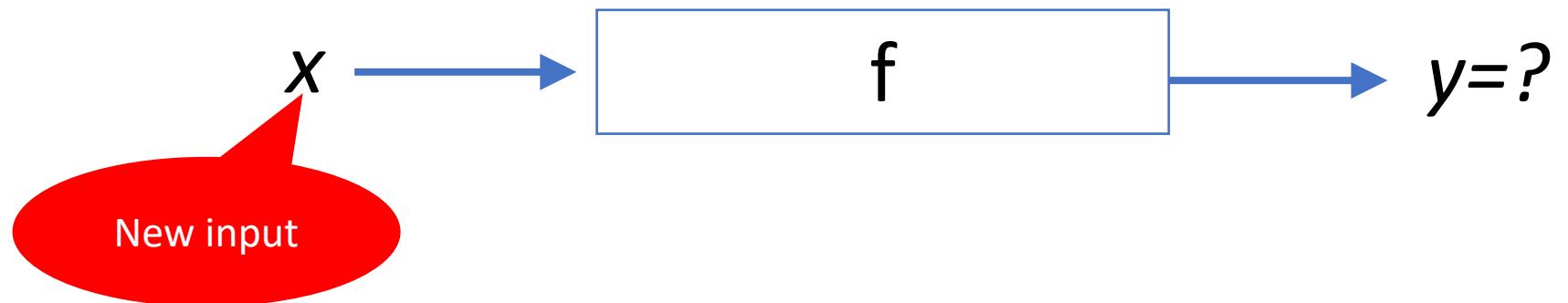
$$(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \dots, (\mathbf{x}^{(n)}, y^{(n)})$$

ML Algorithm

f

- Recall the workflow of supervised learning
- This applies to all supervised learning, including deep learning
  - Training data (input, output) is used to train the model (the neural net)
  - Test data is new inputs to feed the obtained neural net, which then must predict appropriate outputs

# Training data and test data

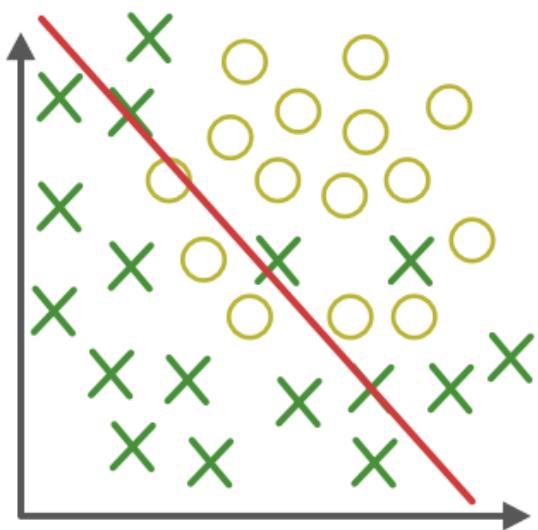


- Recall the workflow of supervised learning
- This applies to all supervised learning, including deep learning
  - Training data (input, output) is used to train the model (the neural net)
  - Test data is new inputs to feed the obtained neural net, which then must predict appropriate outputs

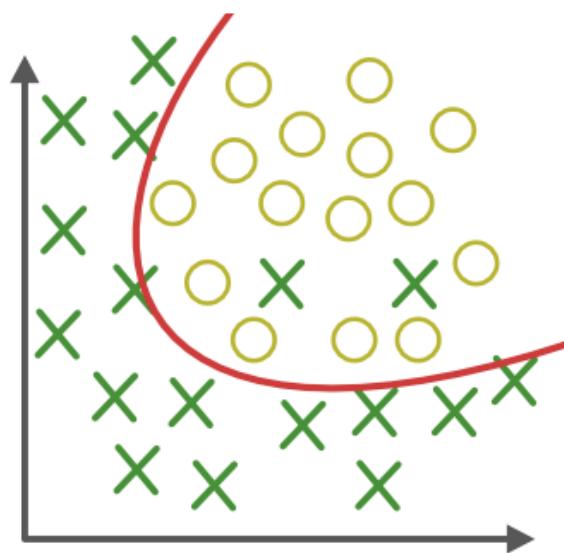
# Overfitting

- Fitting the training data too well is BAD! Why?
- Remember the data you actually want to classify, or predict for, is not the same as the training data – so learning every irrelevant detail (noise) in a training data set will not help
- Overfitting happens when the model is more complex than required

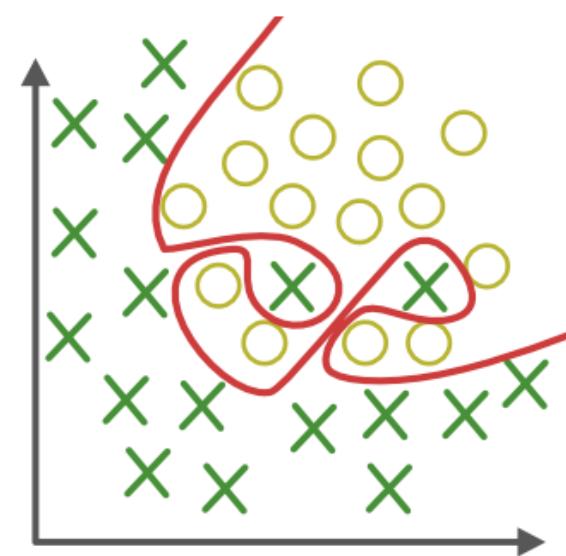
## Classification



**Under-fitting**

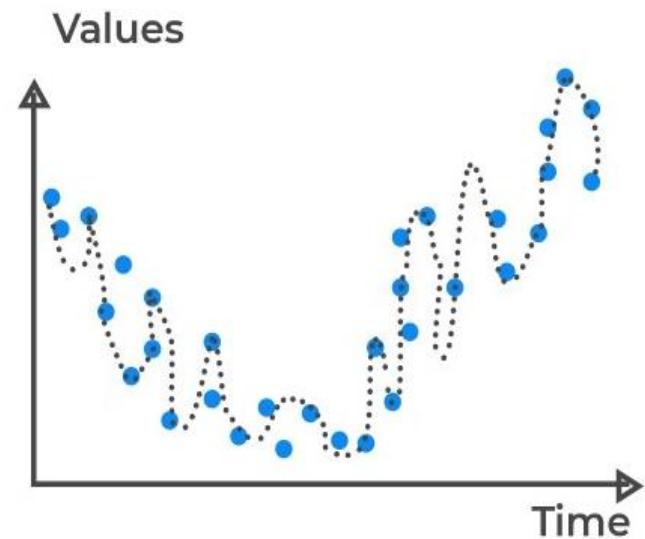
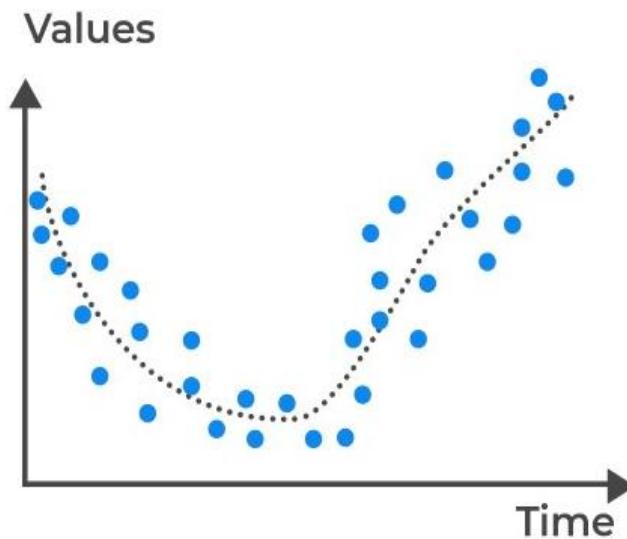
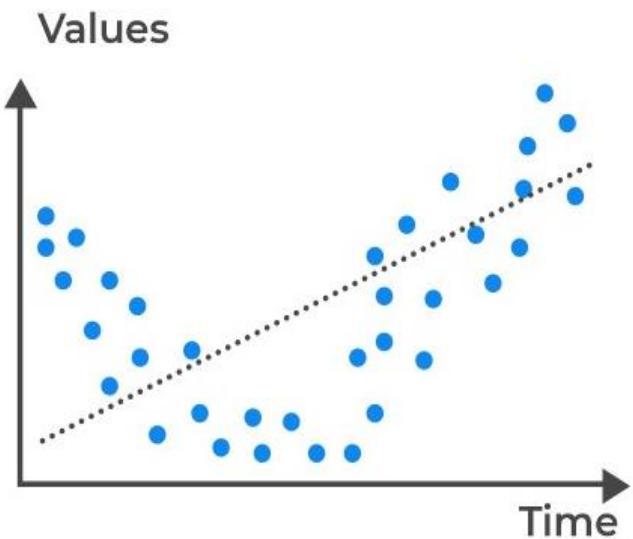


**Appropriate-fitting**



**Over-fitting**

## Regression



# Regularisation

- Neural networks, like all nonlinear models, can overfit the data
- One way to guard against overfitting is regularisation
- Sometimes we need to regularise even linear models (e.g. if there is too much noise in the data)

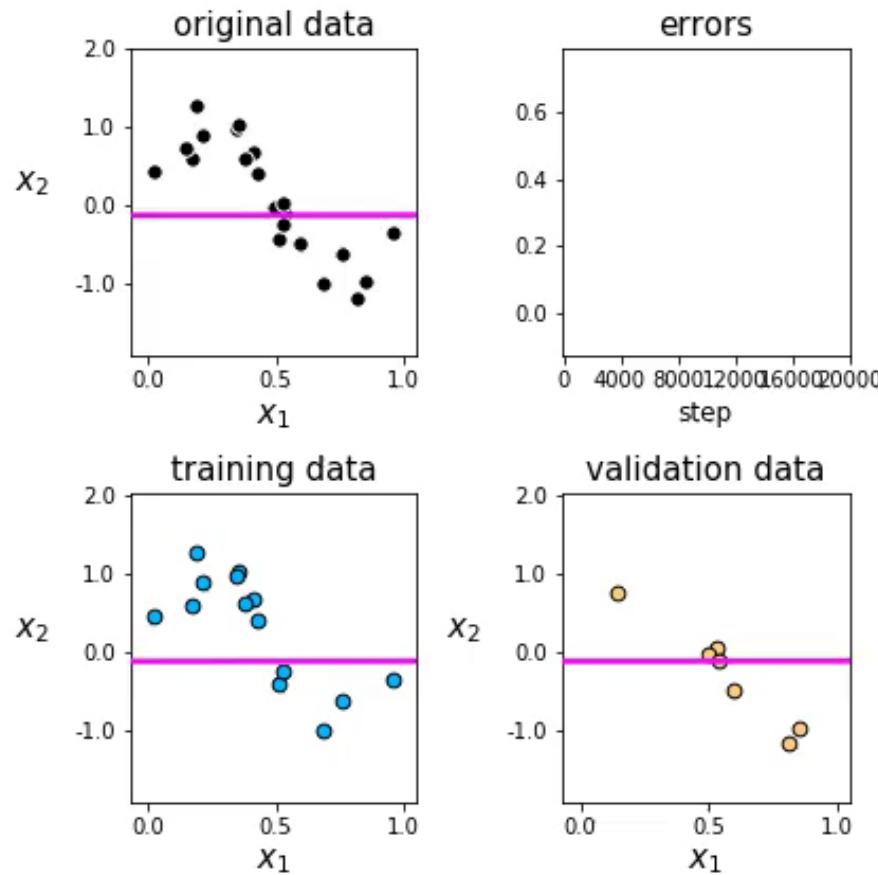
Ways to regularise:

- Add a penalty to the cost function to penalise more complex models
  - e.g. the number of free parameters, or the magnitude of weights
- Prune the model
  - “Dropout” is the process of “leaving out” a proportion of nodes (typically half) when training a deep network.

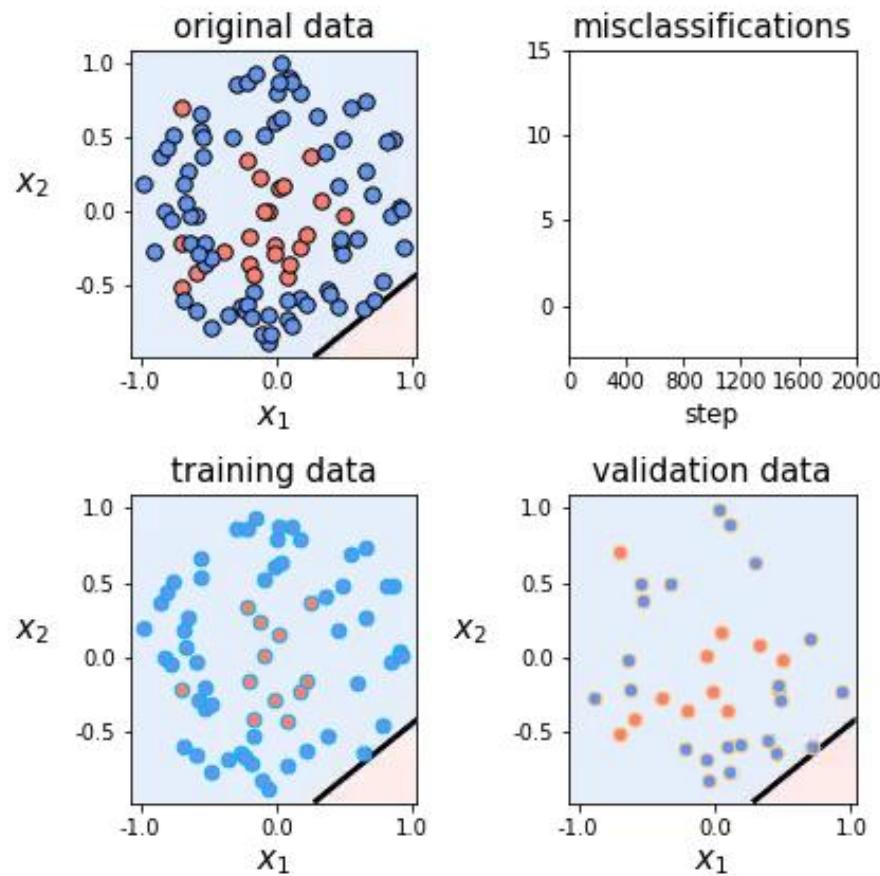
# Early stopping

- Stopping the training early is another effective way to guard against overfitting
- After each gradient update (or Backprop cycle), the training cost will decrease until it reaches 0.
- Set aside a subset of the data (called hold-out set) to use only for monitoring the cost on previously unseen data.
- The error on hold-out set will decrease at first, but as training continues, it can start increasing
- Stop training when the error on hold-out set starts increasing.

# Regression example



# Classification example



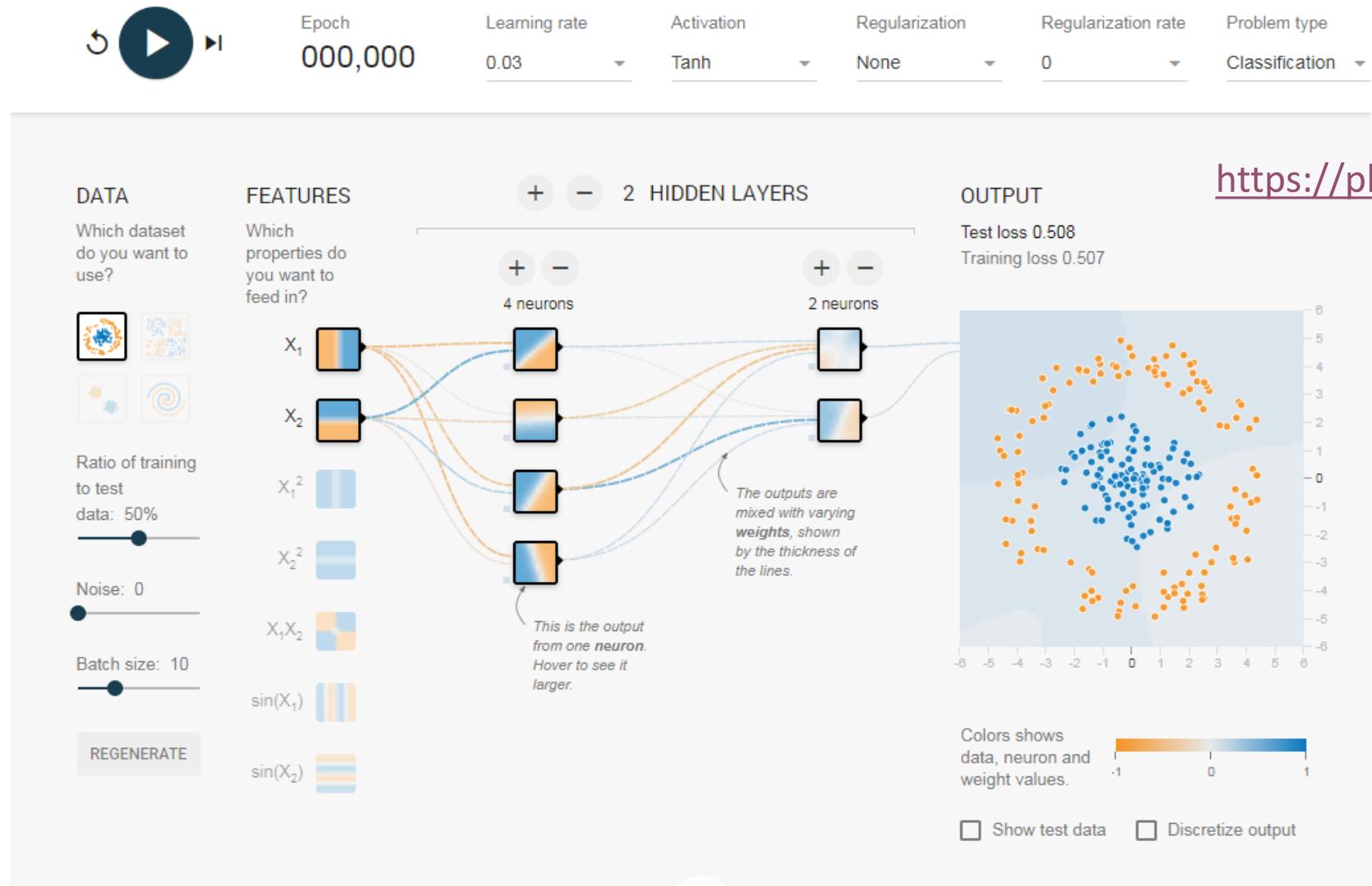
# Next

As we finish our study of supervised learning, one last video lecture next week will cover:

- How to tune hyperparameters
- How to evaluate / validate supervised learning models

Then you will be ready to have a go at running supervised learning methods in practice!

# Playground: Try out neural networks!



# Evaluation & Hyperparameter Tuning

Ata Kaban

With worked examples adapted from Andrew Moore's tutorial slides  
<https://sites.astro.caltech.edu/~george/aybi199/AMooreTutorials/>

# Recap (1)

Each supervised learning method consists of **3 ingredients**:

- Model: form of function we want to learn (has free parameters)
- Cost function: given a training set, it measures the misfit of any particular function from the model
- Training algorithm: gradient descent minimisation of the cost function

Running the training algorithm on some training data learns “best” values of the free parameters, giving us a predictor.

# Recap (2)

**Hyperparameters** are “higher-level” free parameters

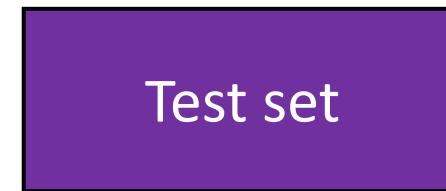
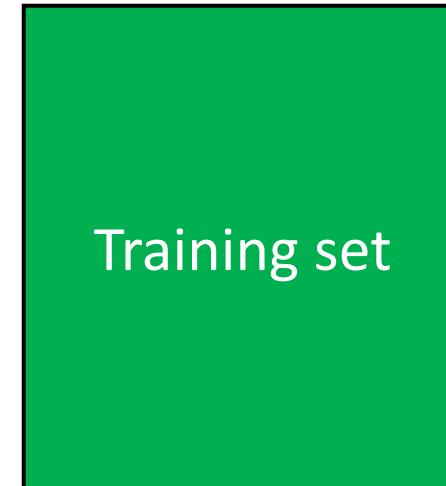
- In Neural Networks:
  - Depth (number of hidden layers)
  - Width (number of hidden neurons in a hidden layer)
  - Activation function (choice of nonlinearity in non-input nodes)
  - Regularisation parameter (way to trade off simplicity vs. fit to the data)
- In polynomial regression
  - Order of the polynomial (use of  $x, x^2, x^3, \dots, x^m$ )
- In general
  - Model choice

# Evaluation of a predictor before deployment

Always split the available annotated data randomly into:

- A **training set** - to be used for training – i.e. estimating all the free parameters
- A **test set** - to be used to evaluate the trained predictor before deploying it

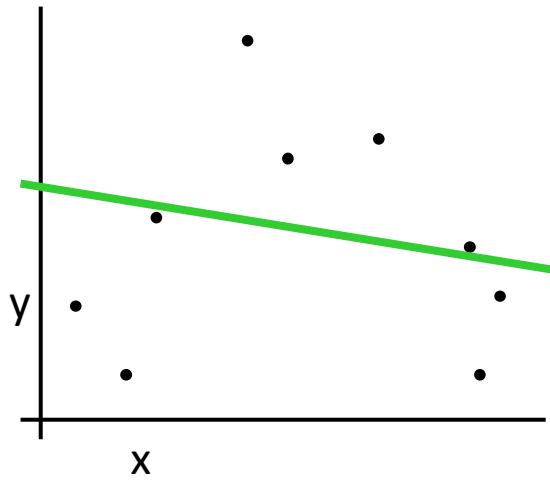
Evaluation of a predictor serves to estimate its future performance, before deploying it in the real world.



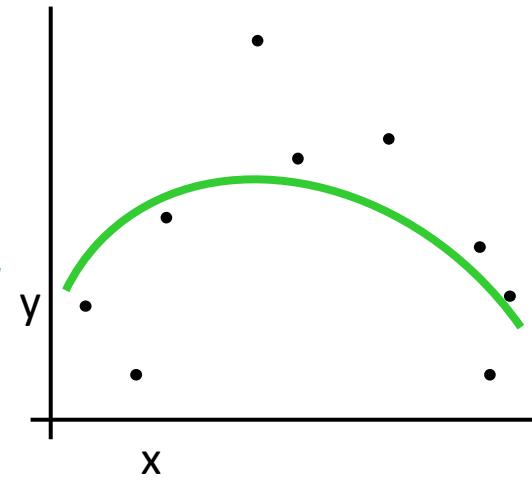
# Which model? How to set hyperparameters?

- Each hyperparameter value corresponds to a different model
- We need methods that evaluate each candidate model
- For this evaluation we can no longer use our cost function computed on training set – why?
  - The more complex (flexible) the model, the better it will fit the training data
  - But the goal is to predict well on future data
  - A model that has capacity to fit any training data will overfit.
- To choose between models (including hyperparameters) we need a criterion to estimate future performance

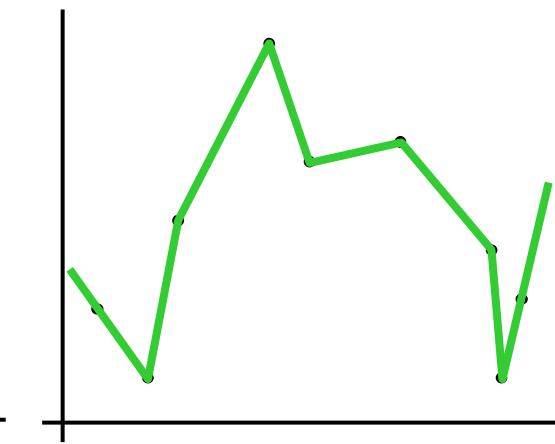
# Which model to choose?



Fit with Model 1



Fit with Model 2



Fit with Model 3

**Remember:** Even if the models only differ by one hyperparameter, they are different models.  
Choosing a particular value of a hyperparameter requires evaluating each model.

# Evaluating models for model choice

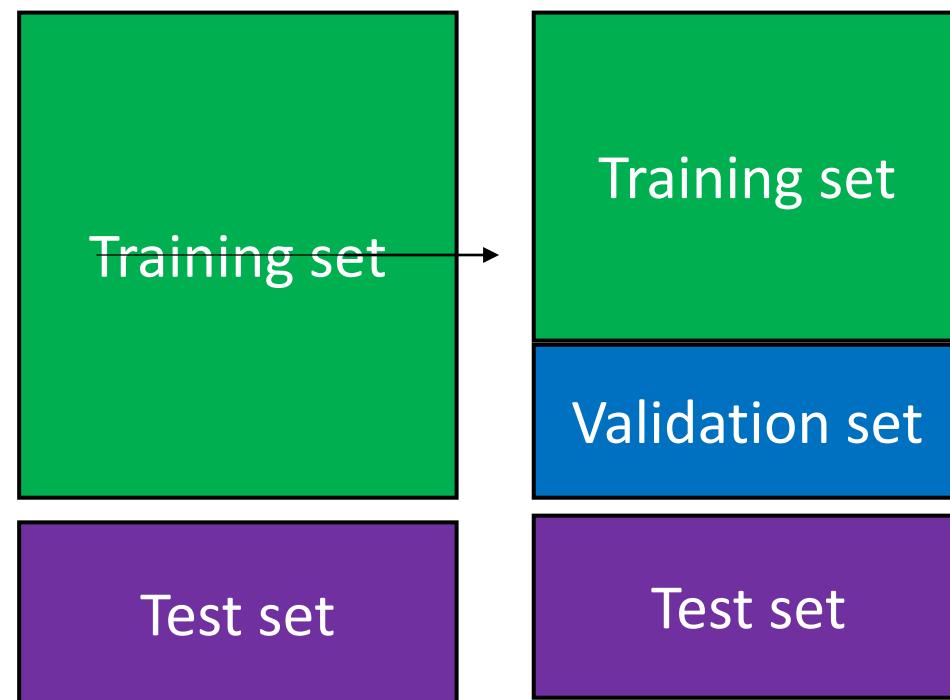
- Don't confuse this with evaluating a predictor (model already chosen)
- The **training set** is annotated data (input, output) – use for training within a chosen model
- The **test set** is also annotated data (input output) – use for evaluating the performance of the trained predictor before deploying it
- None of these can be used to choose the model!
  - Tempting to use the test set, but if we do, we no longer have an independent data set to evaluate the final predictor before deployment!

# Evaluating models for model choice

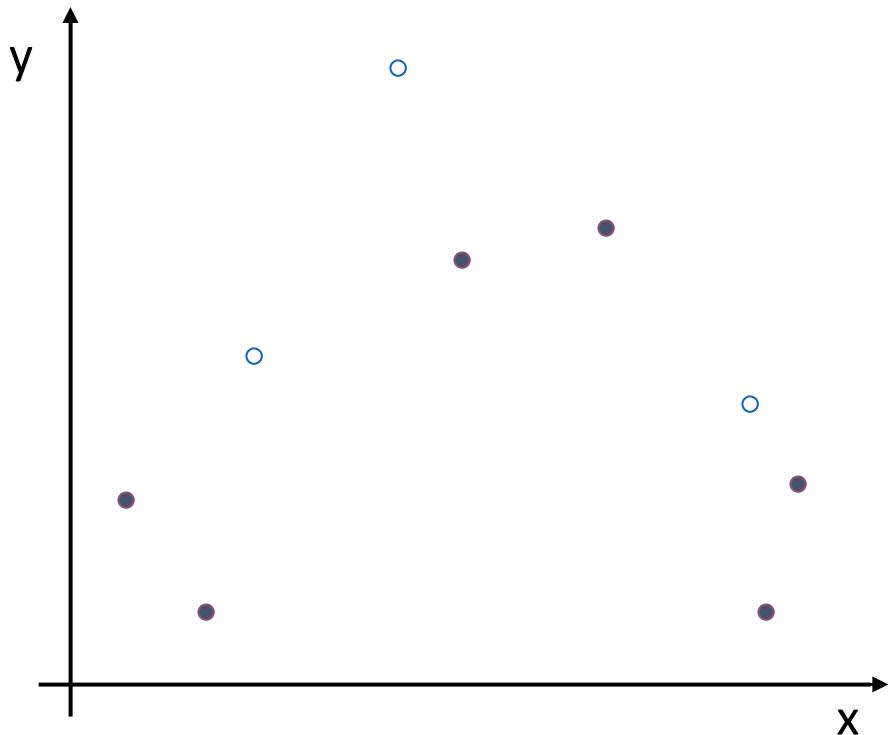
Idea: To choose between models or hyperparameters, split out a subset from the training set = validation set

Methods:

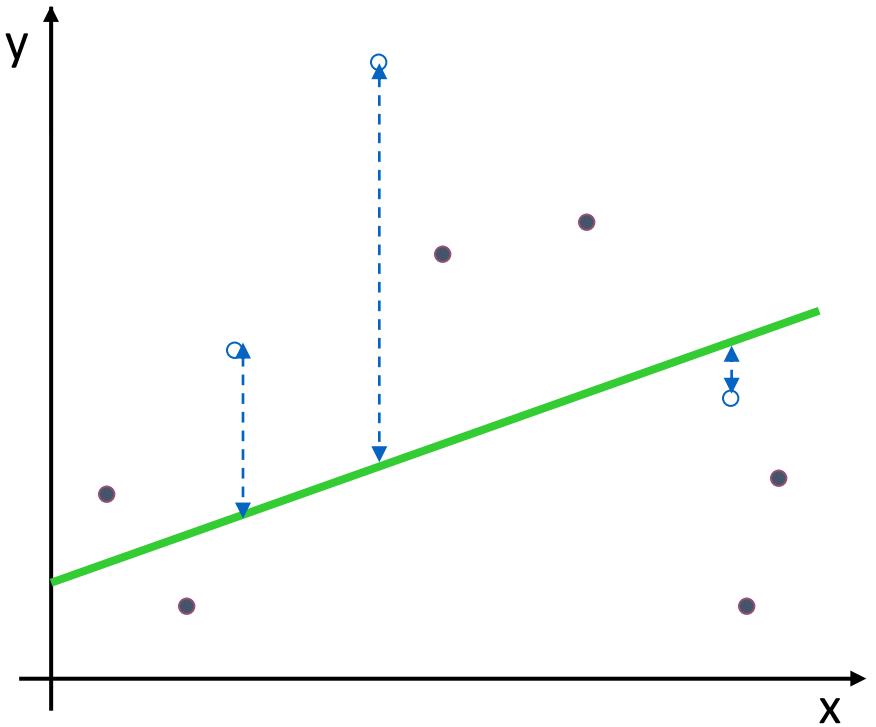
- Holdout validation
- Cross-validation
- Leave-one-out validation



# Method 1: The holdout validation method



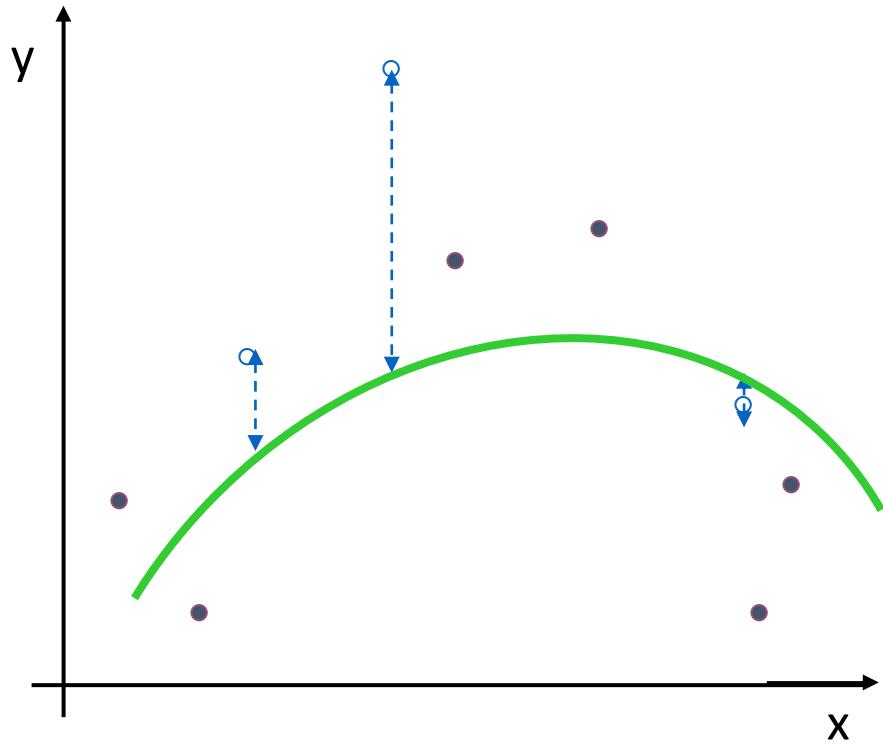
1. Randomly choose 30% of the data to form a **validation set**
2. The remainder is a **training set**
3. Train your model on the training set
4. Estimate the test performance on the validation set
5. Choose the model with lowest validation error
6. Re-train with the chosen model on joined train & validation set to obtain predictor
7. Estimate future performance of obtained predictor on the test set
8. Ready to deploy the predictor



Model 1

Mean Squared Validation Error = 2.4

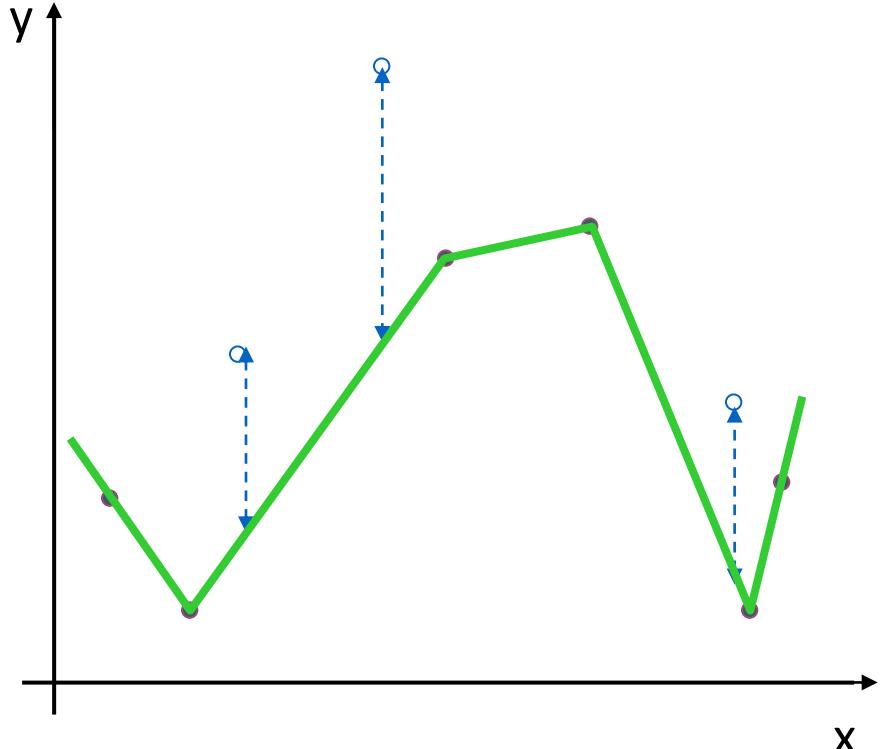
1. Randomly choose 30% of the data to form a validation set
2. The remainder is a training set
3. Train your model on the training set
4. Estimate the test performance on the validation set
5. Choose the model with lowest validation error
6. Re-train with the chosen model on joined train & validation set to obtain predictor
7. Estimate future performance of obtained predictor on the test set
8. Ready to deploy the predictor



Model 2

Mean Squared Validation Error = 0.9

1. Randomly choose 30% of the data to form a **validation set**
2. The remainder is a **training set**
3. **Train your model** on the training set
4. Estimate the test performance on the **validation set**
5. Choose the model with lowest **validation error**
6. Re-train with the chosen model on joined train & validation set to obtain predictor
7. Estimate future performance of obtained predictor on the test set
8. Ready to deploy the predictor



Model 3

Mean Squared Validation Error = 2.2

1. Randomly choose 30% of the data to form a **validation set**
2. The remainder is a **training set**
3. **Train your model** on the training set
4. Estimate the test performance on the **validation set**
5. Choose the model with lowest **validation error**
6. Re-train with the chosen model on joined train & validation set to obtain predictor
7. Estimate future performance of obtained predictor on the test set
8. Ready to deploy the predictor

Choose the model with the **lowest validation error**

Model 1

Mean Squared Validation Error = 2.4

Model 2

Mean Squared Validation Error = 0.9

Model 3

Mean Squared Validation Error = 2.2

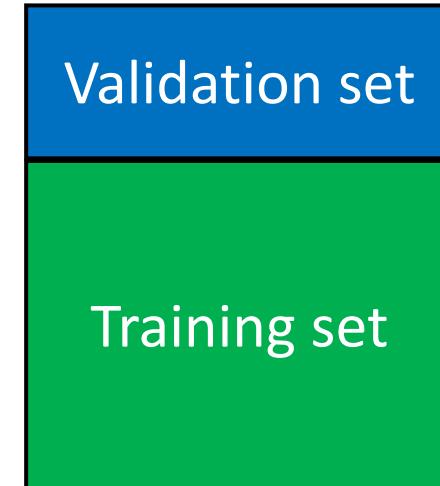
# A practical detail on point 4

## “4. Estimate the test performance on the validation set”

This is done differently in regression and in classification:

- In regression, we compute the cost function (mean square error) on the examples of the validation set (instead of the training set)
- In classification, we don't compute the cross-entropy cost on the validation set, instead on validation set we compute the 0-1 error metric: 
$$\frac{\text{number of wrong predictions}}{\text{number of predictions}} = 1 - \text{Accuracy}$$
  - There are also other metrics, besides Accuracy, that take account of the 2 types of error specific to classification (false positives and false negatives)

# Method 2: k-fold Cross-validation



Split the training set randomly into  $k$  (equal sized) disjoint sets.

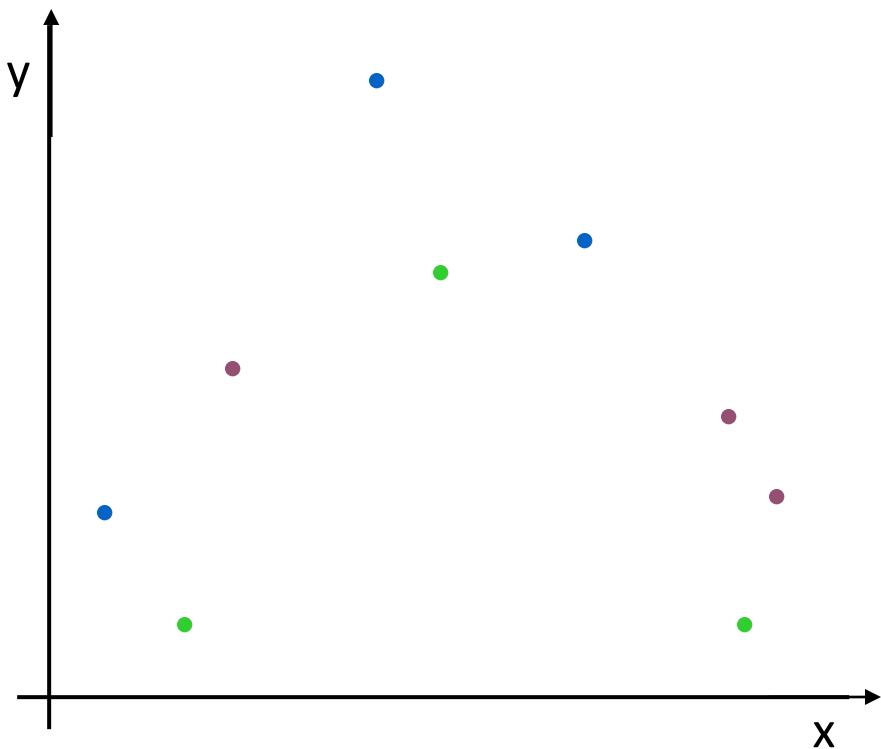
(In this example,  $k=3$ )

Use  $k-1$  of those together for training

Use the remaining one for validation.

Permute the  $k$  sets and repeat  $k$  times.

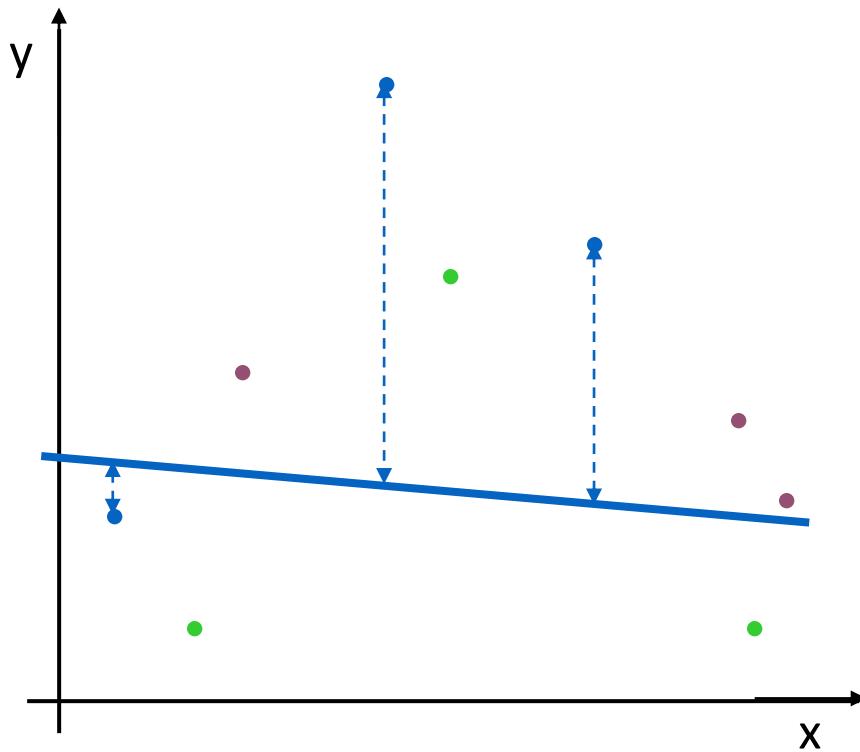
Average the performances on the  $k$  validation sets.



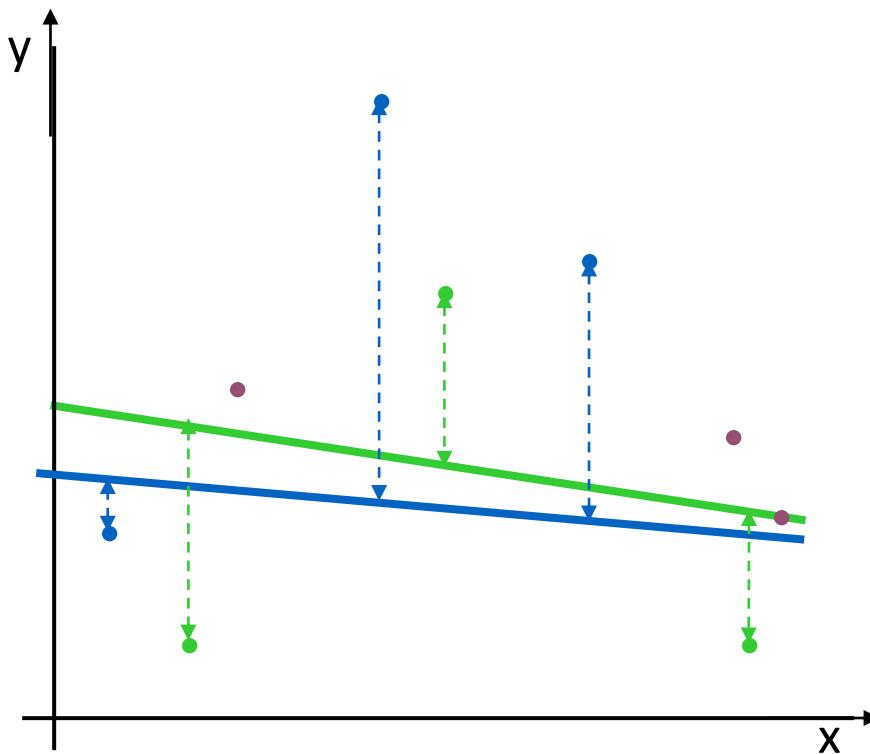
Randomly break the dataset into k partitions (here k=3)

Randomly break the dataset into k partitions (here k=3)

For the blue partition: Train on all the points except the blue partition. Compute the validation error using the points in the blue partition.



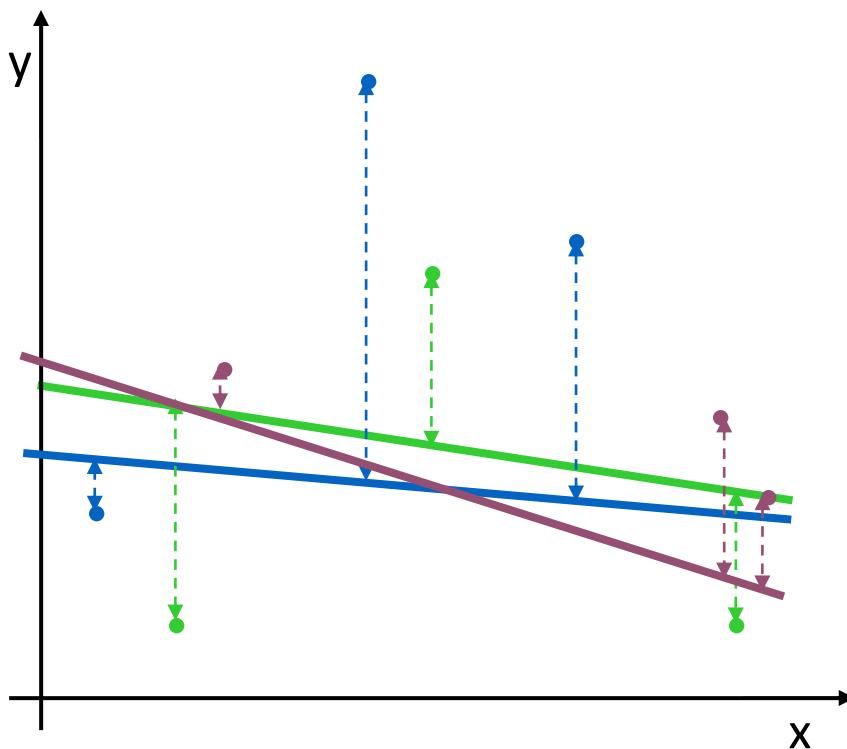
Randomly break the dataset into k partitions (here k=3)



For the blue partition: Train on all the points except the blue partition.  
Compute the validation error using the points in the blue partition.

For the green partition: Train on all the points except the green partition.  
Compute the validation error using the points in the green partition.

Randomly break the dataset into k partitions (here k=3)

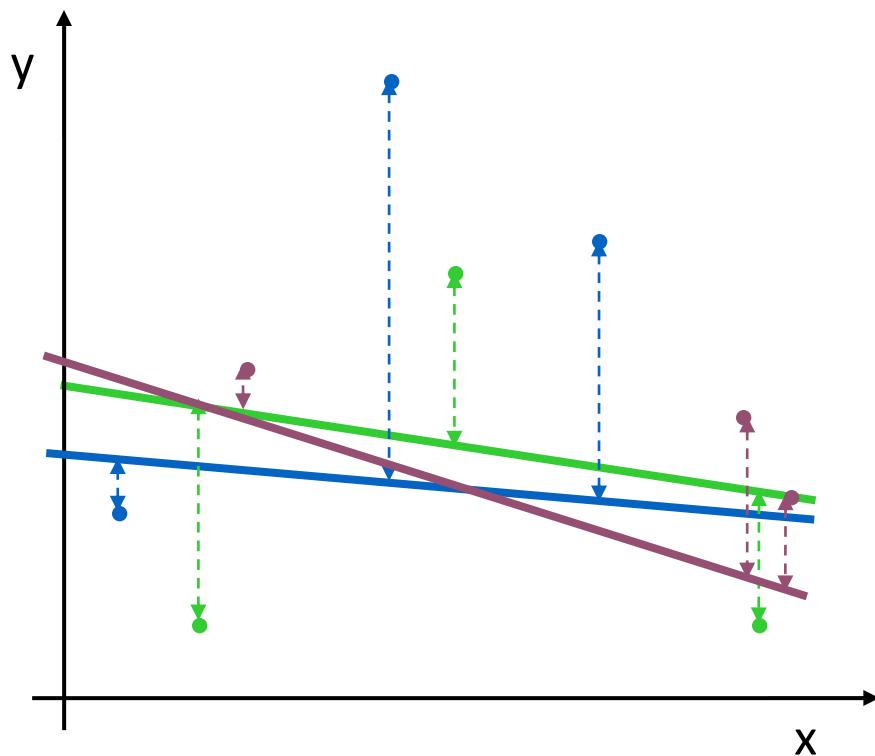


For the blue partition: Train on all the points except the blue partition.  
Compute the validation error using the points in the blue partition.

For the green partition: Train on all the points except the green partition.  
Compute the validation error using the points in the green partition.

For the purple partition: Train on all the points except the purple partition.  
Compute the validation error using the points in the purple partition.

Randomly break the dataset into k partitions (here k=3)



Model 1

$MSE_{3FOLD}=2.05$

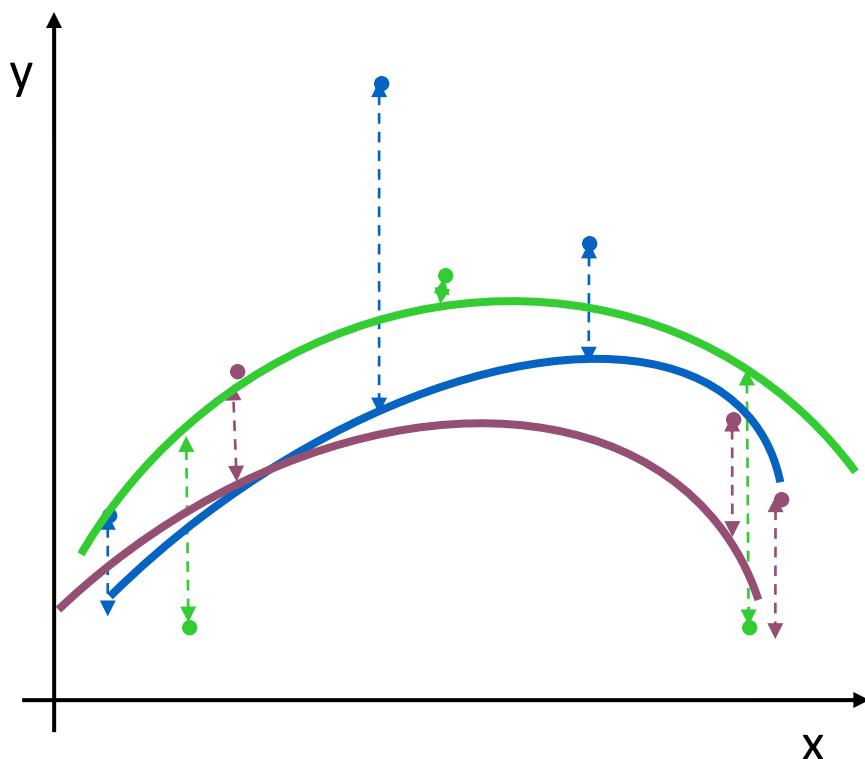
For the blue partition: Train on all the points except the blue partition.  
Compute the validation error using the points in the blue partition.

For the green partition: Train on all the points except the green partition.  
Compute the validation error using the points in the green partition.

For the purple partition: Train on all the points except the purple partition.  
Compute the validation error using the points in the purple partition.

Take the mean of these errors

Randomly break the dataset into k partitions (here k=3)



For the blue partition: Train on all the points except the blue partition.  
Compute the validation error using the points in the blue partition.

For the green partition: Train on all the points except the green partition.  
Compute the validation error using the points in the green partition.

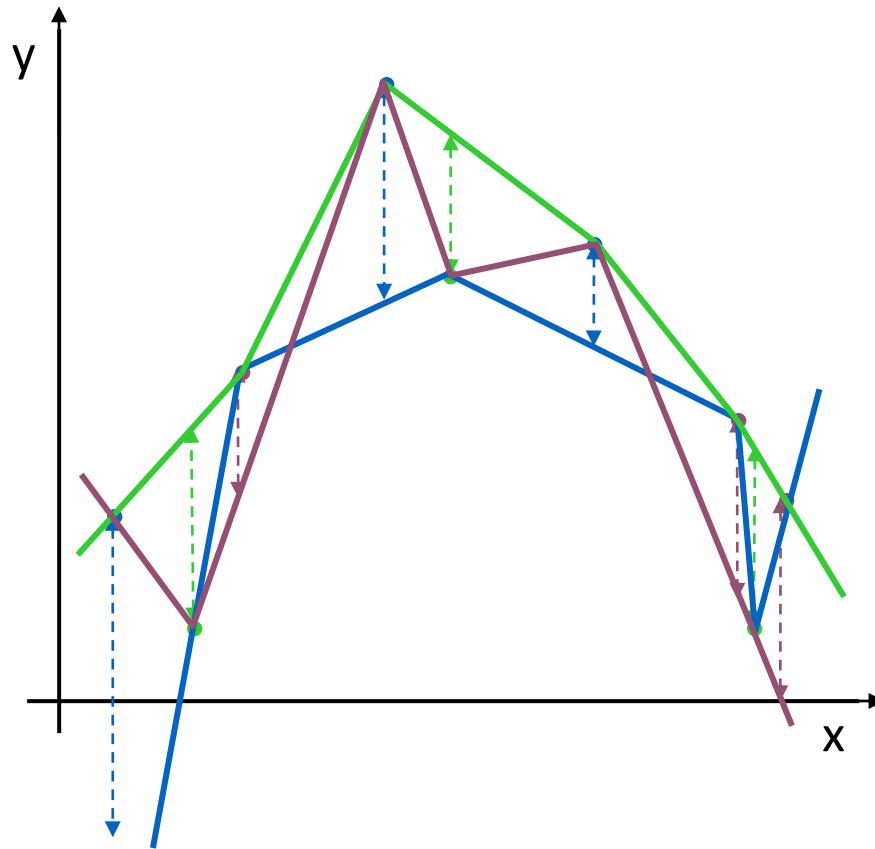
For the purple partition: Train on all the points except the purple partition.  
Compute the validation error using the points in the purple partition.

Take the mean of these errors

Model 2

$MSE_{3FOLD}=1.11$

Randomly break the dataset into k partitions (here k=3)



Model 3

$MSE_{3FOLD}=2.93$

For the blue partition: Train on all the points except the blue partition.  
Compute the validation error using the points in the blue partition.

For the green partition: Train on all the points except the green partition.  
Compute the validation error using the points in the green partition.

For the purple partition: Train on all the points except the purple partition.  
Compute the validation error using the points in the purple partition.

Take the mean of these errors

# Method 3: Leave-one-out validation

- We leave out a single example for validation, and train on all the rest of the annotated data
- For a total of  $N$  examples, we repeat this  $N$  times, each time leaving out a single example
- Take the average of the validation errors as measured on the left-out points
- Same as  $N$ -fold cross-validation where  $N$  is the number of labelled points

# Advantages & Disadvantages

	<b>Advantages</b>	<b>Disadvantages</b>	
<b>Holdout validation</b>	Computationally cheapest	Most unreliable if sample size is not large enough	Large sample
<b>3-fold</b>	Slightly more reliable than holdout	<ul style="list-style-type: none"><li>Wastes 1/3-rd annotated data.</li><li>Computationally 3-times as expensive as holdout</li></ul>	
<b>10-fold</b>	<ul style="list-style-type: none"><li>Only wastes 10%</li><li>Fairly reliable</li></ul>	<ul style="list-style-type: none"><li>Wastes 10% annotated data</li><li>Computationally 10-times as expensive as holdout</li></ul>	
<b>Leave-one-out</b>	Doesn't waste data	Computationally most expensive	Small sample

Using model validation to  
tune hyperparameters

# Example 1: Choosing number of hidden units in a Multi-Layer Perceptron

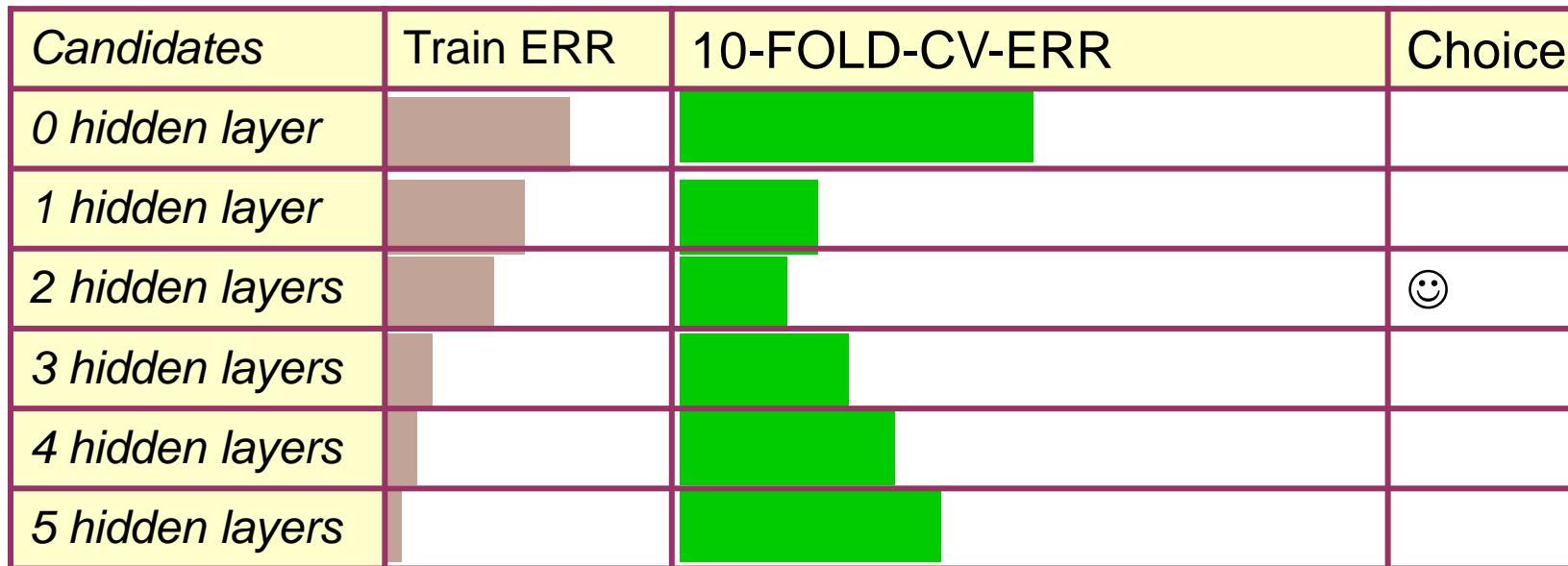
- Step 1: Compute 10-fold CV error for six different model classes:

Candidates	Train ERR	10-FOLD-CV-ERR	Choice
<i>0 hidden units</i>			
<i>1 hidden units</i>			
<i>2 hidden units</i>			😊
<i>3 hidden units</i>			
<i>4 hidden units</i>			
<i>5 hidden units</i>			

- Step 2: Whichever candidate choice gave best CV score: train it with all the data, and that's the predictor you'll use.

# Example 2: Choosing number of hidden layers in a neural nets

- Step 1: Compute 10-fold CV error for six different model classes:



- Step 2: Whichever model class gave best CV score: train it with all the data, and that's the predictor you'll use.

# Example 3: Choosing the activation function is (deep) neural net

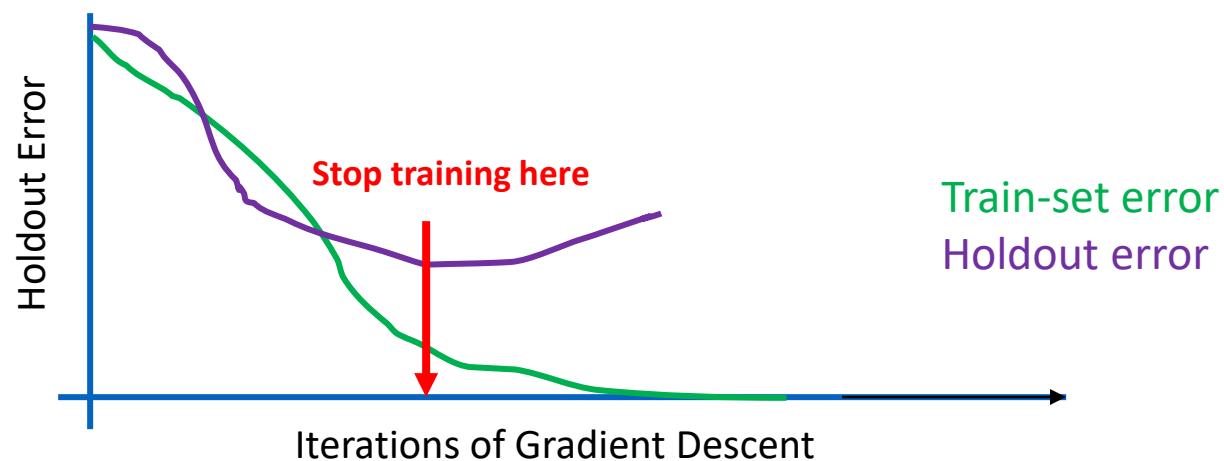
- Step 1: Compute 10-fold CV error for six different model classes:

Candidates	Train ERR	10-FOLD-CV-ERR	Choice
$\sigma_1$			
$\sigma_2$			
$\sigma_3$			😊
$\sigma_4$			
$\sigma_5$			
$\sigma_6$			

- Step 2: Whichever candidate choice gave best CV score: train it with all the data, and that's the predictor you'll use.

# Example 4: Early Stopping using Holdout validation

- Suppose you have a neural net with too many hidden units. It will overfit.
- As Backprop (gradient descent) progresses, monitor the error on a holdout set



# What you should know

- Why you can't use “training-set-error” to choose between models
- Why you need model validation methods to tune hyperparameters
- Methods for model validation and how they work
  - Holdout validation
  - k-fold cross-validation
  - Leave-one-out validation
- Advantages & disadvantages of each model validation method

# ML Applications using Weka

# Sentiment Analysis - A classification task

- Sentiment is feelings
  - Attitudes: positive/negative/neutral?
  - Emotions: happy/sad?
  - Opinions: like/dislike?
- Sentiment analysis: the analysis of feelings behind the words.
- Questions might be asked in sentiment analysis:
  - is this product review positive or negative?
  - how are people responding to this advert campaign?
  - how have bloggers' attitudes about presidential election?

*"I am happy with this water bottle."*



Positive

*"This is a bad investment."*



Negative

*"I am going to walk today."*



Neutral

# Sentiment Analysis - A classification task

1. Get text data with labels
2. Extract features from the data, i.e. convert the text items to feature vectors (NLP techniques)
3. Train a model using a ML algorithm for classification, e.g. Neural networks, SVM.
4. Test the model and use it for future classification.

# Example: a twitter dataset

- 100 positive tweets and 100 negative tweets from Edinburgh twitter corpus.

```
@relation tweets
```

```
@attribute tweet_body string  
@attribute sentiment {pos,neg}
```

```
@data
```

```
'anyone feel motivated the fri afternoon prior to a holiday? wanted to get lots done... but i  
want jammies and judge judy... \"SIR!\" &lt;3 her ',pos  
'seriously, do you have to rub it in maggie!!!! ',pos  
'if i\'m not wrong.. Alt is when image can\'t be displayed.. Tooltip is the \'title\' ',pos  
'I don\'t like social karma much. Would rather skip it, but can\'t afford to piss my friend off  
any more ',pos  
'I\'d be happy to review the Iomega if EMC send me one!!! ',pos  
'can i get a refund if i don\'t like the product? ',pos
```

- Input: tweets, output: positive or negative.
- Try in Weka

# Read Weka Outputs

- Running information
  - how the experiment is run
- Classifier mode
  - what the model looks like, e.g. structure, trained parameters, etc.
  - training time
- Performance results
  - Performance summary
  - Detailed performance
  - Confusion matrix (for classification)

# Confusion Matrix

- Show statistics of how samples from each class are labelled.

		Predicted condition	
		Cancer	Non-cancer
Total		8	4
Actual condition	Cancer	6	2
	Non-cancer	1	3

==== Confusion Matrix ===

a   b   <-- classified as  
55  45 | a = pos  
39  61 | b = neg

- Based on the confusion matrix, we can further calculate other metrics, such as recall, precision, etc.

More info can be found here: [https://en.wikipedia.org/wiki/Confusion\\_matrix](https://en.wikipedia.org/wiki/Confusion_matrix)

# Dominant colour extraction - A clustering task

- Sometimes people want to extract colour palettes in artworks.
- What is the colour theme in this image?



```
@relation house  
@attribute red real  
@attribute green real  
@attribute blue real
```

```
@data  
26 20 45  
28 5 46  
28 12 44  
28 13 46  
31 4 51  
31 5 48  
31 8 44  
31 9 44  
32 5 50  
32 8 47  
32 11 47  
32 12 48
```

# Dominant colour extraction - A clustering task

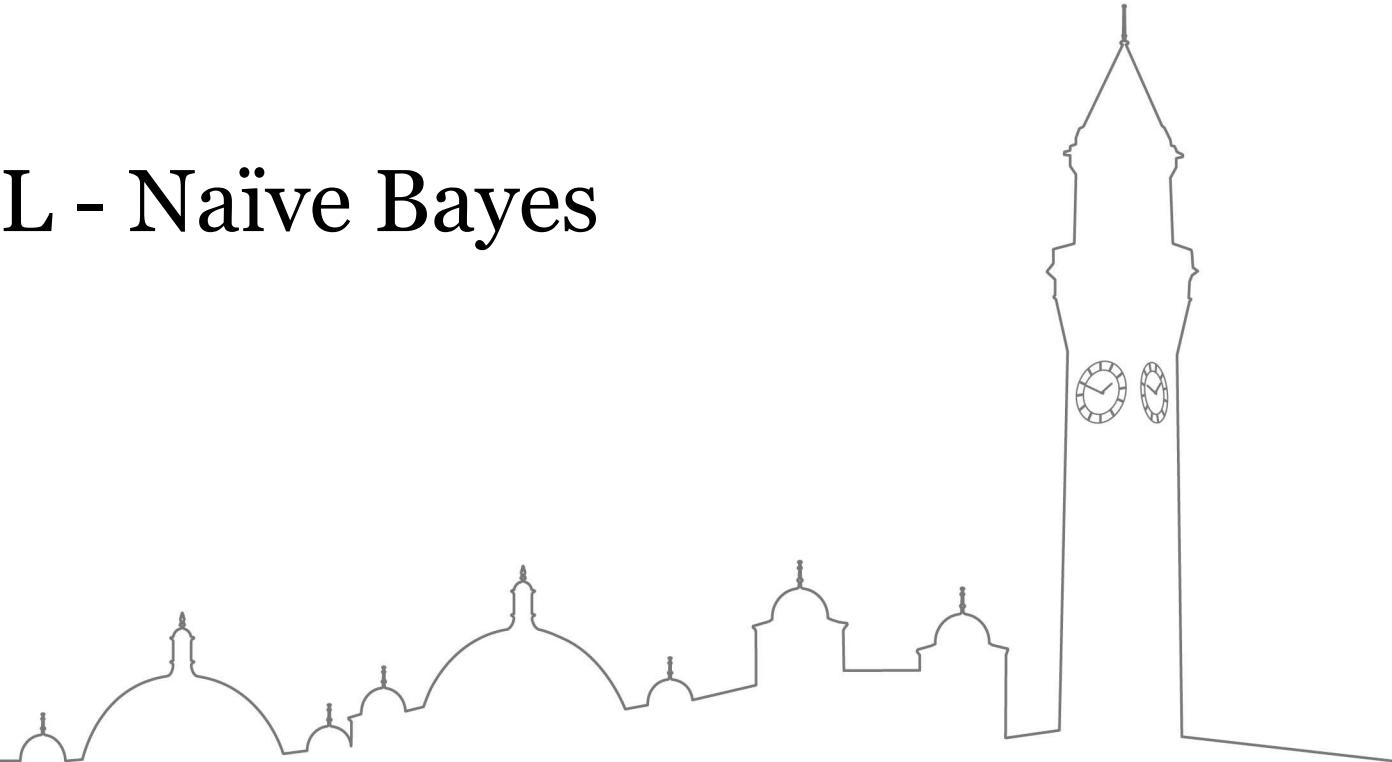
1. Get RGB values of the image as input.
  2. Train a model that cluster RGB vectors into several groups, e.g. K-means.
  3. Obtain clusters and output the RGB value of each cluster centre.
- Try in Weka



UNIVERSITY OF  
BIRMINGHAM

# AI1/AI&ML - Naïve Bayes

Dr Leonardo Stella



# Aims of the Session

This session aims to help you:

- Describe the fundamental concepts in probability theory
- Explain Bayes' Theorem and its application in ML
- Apply Naïve Bayes to classification for categorical and numerical independent variables

# Overview

- **Fundamental concepts in Probability Theory**
- Bayes' Theorem
- Naïve Bayes for Categorical Independent Variables
- Naïve Bayes for Numerical Independent Variables

# Fundamental Concepts in Probability Theory

- **Probabilistic model:** a mathematical description of an uncertain situation. The two main elements of a probabilistic model are:
  - The **sample space**  $\Omega$ , which is the set of all possible outcomes
  - The **probability law**, which assigns to a set  $A$  of possible outcomes (called an **event**) a nonnegative number  $P(A)$  (called the **probability** of  $A$ )
- Every probabilistic model involves an underlying process, called the **experiment**, that produces exactly one of several possible outcomes
- A subset of the sample space  $\Omega$  is called an **event**

# Example: Toss of a Coin

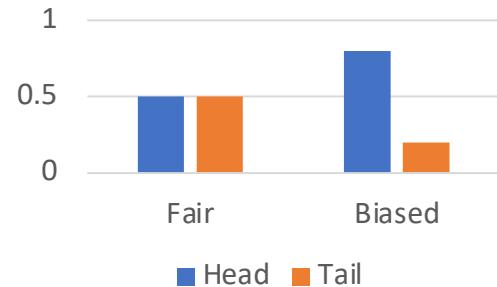
- Consider the following experiment a single toss of a fair coin
  - The **sample space**  $\Omega$ : head (H) or tail (T)
  - The **probability law**:  $P(H) = 0.5$  (called the **probability** of H),  $P(T) = 0.5$

# Example: Toss of a Coin

- Consider the following experiment a single toss of a fair coin
  - The **sample space**  $\Omega$ : head (H) or tail (T)
  - The **probability law**:  $P(H) = 0.5$  (called the **probability** of H),  $P(T) = 0.5$
- Let us now consider the experiment consisting of 3 coin tosses. What is the probability of having exactly 2 heads? What about exactly 1 head?

# Example: Toss of a Coin

- Consider the following experiment a single toss of a fair coin
  - The **sample space**  $\Omega$ : head (H) or tail (T)
  - The **probability law**:  $P(H) = 0.5$  (called the **probability** of H),  $P(T) = 0.5$
- Let us now consider the experiment consisting of 3 coin tosses. What is the probability of having exactly 2 heads? What about exactly 1 head?
- Repeat with the biased coin:  $P(H) = 0.8$



# Probability Axioms

- **Nonnegativity:**  $P(A) \geq 0$ , for every event  $A$
- **Additivity:** If  $A$  and  $B$  are two disjoint events, then the probability of their union satisfies:  $P(A \cup B) = P(A) + P(B)$
- **Normalisation:** The probability of the entire sample space is equal to 1, namely  $P(\Omega) = 1$

# (Discrete) Random Variables

- Given an experiment and the corresponding sample space, a random variable maps a particular number with each outcome
- Mathematically, a random variable  $X$  is a real-valued function of the experimental outcome

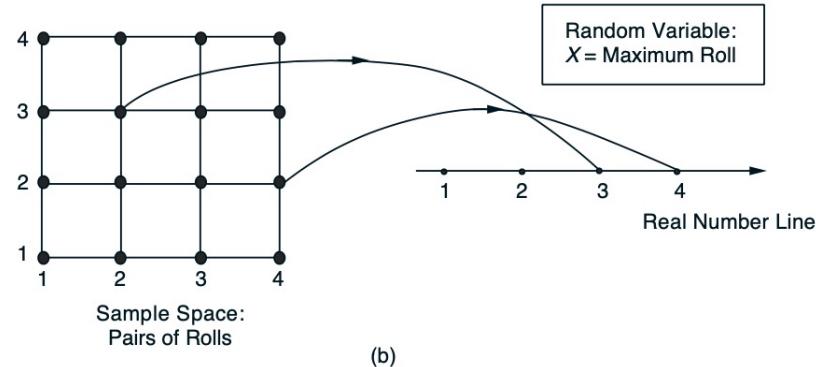


Image: taken from Introduction to Probability (Fig. 2.1 (b))

# Probability Mass Function (PMF)

- The probability mass function (PMF) captures the probabilities of the values that a (discrete) random variable can take
- Let us consider the previous example:  
 $P(X = 1) = 1/16$

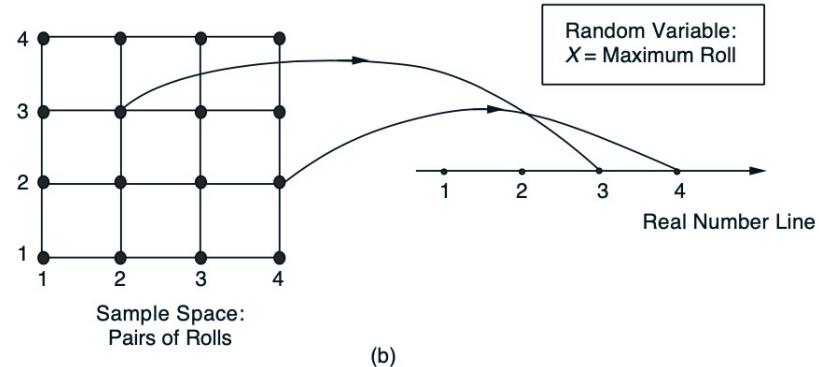


Image: taken from Introduction to Probability (Fig. 2.1 (b))

# Probability Mass Function (PMF)

- The probability mass function (PMF) captures the probabilities of the values that a (discrete) random variable can take
- Let us consider the previous example:

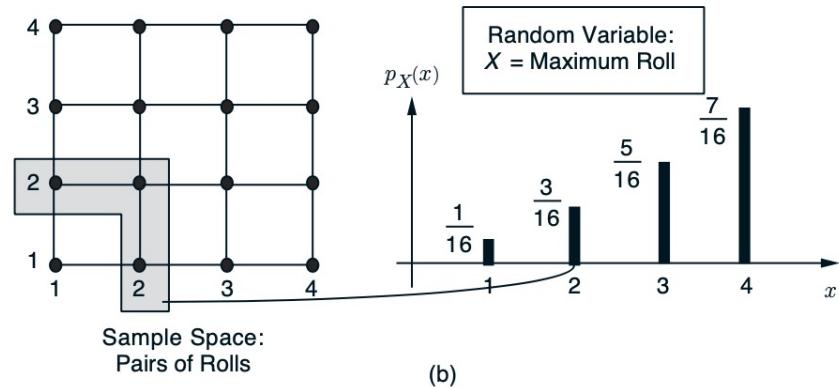
$$P(X = 1) = 1/16$$

$$P(X = 2) = 3/16$$

$$P(X = 3) = 5/16$$

$$P(X = 4) = 7/16$$

Image: taken from Introduction to Probability (Fig. 2.2 (b))



# Notation

- Random variables are usually indicated with uppercase letters, e.g.,  $X$  or *Temperature* or *Infection*
- The values are indicated with lowercase letters, e.g.,  $X \in \{\text{true}, \text{false}\}$  or *Infection*  $\in \{\text{low}, \text{moderate}, \text{high}\}$
- Vectors are usually indicated with bold letters or a small arrow above the letter, e.g.,  $\mathbf{X}$  or  $\vec{X}$
- PMF is usually indicated by the symbol  $p_X(x)$

# Unconditional/Conditional Probability Distributions

- An **unconditional** (or **prior**) probability distribution gives us the probabilities of all possible events without knowing anything else about the problem, e.g., the maximum value of two rolls of a 4-sided die
- $P(X) = \left\{ \frac{1}{15}, \frac{3}{15}, \frac{5}{15}, \frac{7}{15} \right\}$
- A **conditional** (or **posterior**) probability distribution gives us the probability of all possible events with some additional knowledge, e.g., the maximum value of two rolls of a 4-sided die knowing that the first roll is 3
- $P(X | X_1 = 3) = \left\{ 0, 0, \frac{3}{4}, \frac{1}{4} \right\}$

# Joint Probability Distributions

- A **joint probability distribution** is the probability distribution associated to all combinations of the values of two or more random variables
- This is indicated by commas, e.g.,  $P(X, Y)$  or  $P(Toothache, Cavity)$
- We can calculate the joint probability distribution by using the **product rule** as in the following:

$$P(X, Y) = P(X | Y) P(Y) = P(Y | X) P(X)$$

# Mean, Variance and Standard Deviation

- The mean (or expected value or expectation), also indicated by  $\mu$ , of a random variable  $X$  with PMF  $p_X(x)$  represents the centre of gravity of the PMF:

$$E(X) = \sum_x x p_X(x)$$

- E.g., let us consider the random variable  $X$ , i.e., the roll of a 4-sided die. The mean is calculated as:  $E(X) = 1 * \frac{1}{4} + 2 * \frac{1}{4} + 3 * \frac{1}{4} + 4 * \frac{1}{4} = 2.5$

- The variance of a random variable  $X$  provides a measure of the dispersion around the mean:

$$\text{var}(X) = \sum_x (x - E(X))^2 p_X(x)$$

- The standard deviation is another measure of dispersion:  $\sigma_X = \sqrt{\text{var}(X)}$

# Continuous Random Variables

- A random variable  $X$  is called continuous if its probability law can be described in terms of a nonnegative function  $f_X$ . This function is called **probability density function** (PDF) and is the equivalent of the PMF for discrete random variables

$$P(X \in B) = \int_B f_X(x)dx$$

- Since we are dealing with continuous variables, there are an infinite number of values that  $X$  can take
- As for the discrete case, also for continuous random variables we can have unconditional, conditional and joint probability distributions

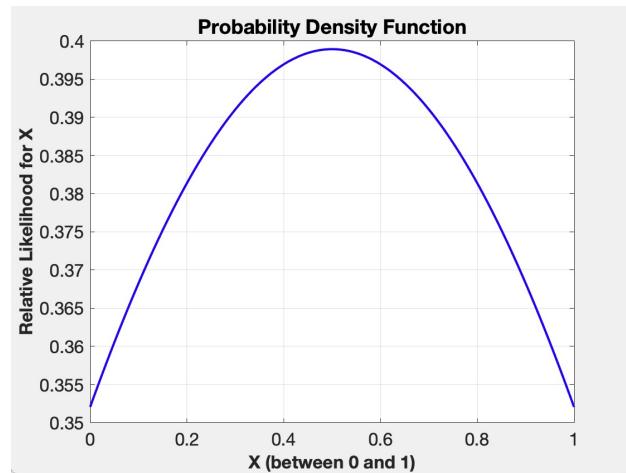
# Example: Random Number Generator

- As an example, let us consider a random number generator that returns a random value between 0 and 1:  $X \in [0,1]$
- And let us model it with a Gaussian (or normal) distribution

$$P(X = a | \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(a-\mu)^2}{2\sigma^2}},$$

where  $\mu$  is the mean and  $\sigma^2$  is the variance

Also, recall that  $\pi = 3.14159$  and  $e = 2.71828$



# Overview

- Fundamental concepts in Probability Theory
- Bayes' Theorem
- Naïve Bayes for Categorical Independent Variables
- Naïve Bayes for Numerical Independent Variables

# Bayes' Theorem

- Recall the product rule for a joint probability distribution of independent variable(s)  $X$  and dependent variable  $Y$ :

$$P(X, Y) = P(X | Y) P(Y) = P(Y | X) P(X)$$

- By taking the second and last term from the above equation and rearranging, we get:

$$P(X | Y) = \frac{P(Y | X) P(X)}{P(Y)}$$

- The above equation is known as **Bayes' Theorem** (also Bayes' rule or Bayes' law)

# ML: Probabilistic Inference

- Our ML task consists in computing the posterior probabilities for query propositions given some observed evidence: this method is **probabilistic inference**
- We use Bayes' Theorem to make predictions about an underlying process given a knowledge base consisting of the data produced by this process

# Equivalent Terminology

- Input attribute, independent variable, input variable
- Output attribute, dependent variable, output variable, label (classification)
- Predictive model, classifier (classification), or hypothesis (statistical learning)
- Learning a model, training a model, building a model
- Training examples, training data
- Example, observation, data point, instance (more frequently used for test examples)
- $P(a, b) = P(a \text{ and } b) = P(a \wedge b)$

# Learning Probabilities

- Consider the **training set**

Days	Sunny ( $X_1$ )	Windy ( $X_2$ )	Tennis ( $Y$ )
Day 1	yes	no	yes
Day 2	yes	no	yes
Day 3	yes	yes	yes
Day 4	no	yes	no
Day 5	no	no	no
Day 6	no	yes	no

# Learning Probabilities

- Consider the **training set**

Days	Sunny ( $X_1$ )	Windy ( $X_2$ )	Tennis ( $Y$ )
Day 1	yes	no	yes
Day 2	yes	no	yes
Day 3	yes	yes	yes
Day 4	no	yes	no
Day 5	no	no	no
Day 6	no	yes	no

- Let us build the **model** for one independent variable, e.g., Windy ( $X_2$ )

Frequency Table	Tennis = yes	Tennis = no	Total
Windy = yes			
Windy = no			
Total			

# Learning Probabilities

- Consider the **training set**

Days	Sunny ( $X_1$ )	Windy ( $X_2$ )	Tennis ( $Y$ )
Day 1	yes	no	yes
Day 2	yes	no	yes
Day 3	yes	yes	yes
Day 4	no	yes	no
Day 5	no	no	no
Day 6	no	yes	no

- Let us build the **model** for one independent variable, e.g., Windy ( $X_2$ )

Frequency Table	Tennis = yes	Tennis = no	Total
Windy = yes	1		
Windy = no			
Total			

# Learning Probabilities

- Consider the **training set**

Days	Sunny ( $X_1$ )	Windy ( $X_2$ )	Tennis ( $Y$ )
Day 1	yes	no	yes
Day 2	yes	no	yes
Day 3	yes	yes	yes
Day 4	no	yes	no
Day 5	no	no	no
Day 6	no	yes	no

- Let us build the **model** for one independent variable, e.g., Windy ( $X_2$ )

Frequency Table	Tennis = yes	Tennis = no	Total
Windy = yes	1		
Windy = no	2		
Total	3		

# Learning Probabilities

- Consider the **training set**

Days	Sunny ( $X_1$ )	Windy ( $X_2$ )	Tennis ( $Y$ )
Day 1	yes	no	yes
Day 2	yes	no	yes
Day 3	yes	yes	yes
Day 4	no	yes	no
Day 5	no	no	no
Day 6	no	yes	no

- Let us build the **model** for one independent variable, e.g., Windy ( $X_2$ )

Frequency Table	Tennis = yes	Tennis = no	Total
Windy = yes	1	2	3
Windy = no	2	1	3
Total	3	3	6

# Learning Probabilities (continued)

$$P(\text{Windy=yes} \mid \text{Tennis=yes}) =$$

$$P(\text{Windy=no} \mid \text{Tennis=yes}) =$$

$$P(\text{Windy=yes} \mid \text{Tennis=no}) =$$

$$P(\text{Windy=no} \mid \text{Tennis=no}) =$$

Frequency Table	Tennis = yes	Tennis = no	Total
Windy = yes	1	2	3
Windy = no	2	1	3
Total	3	3	6

# Learning Probabilities (continued)

$$P(\text{Windy=yes} \mid \text{Tennis=yes}) = 1/3$$

$$P(\text{Windy=no} \mid \text{Tennis=yes}) = 2/3$$

$$P(\text{Windy=yes} \mid \text{Tennis=no}) = 2/3$$

$$P(\text{Windy=no} \mid \text{Tennis=no}) = 1/3$$

Frequency Table	Tennis = yes	Tennis = no	Total
Windy = yes	1	2	3
Windy = no	2	1	3
Total	3	3	6

# Learning Probabilities (continued)

$$P(\text{Windy=yes} \mid \text{Tennis=yes}) = 1/3$$

$$P(\text{Windy=no} \mid \text{Tennis=yes}) = 2/3$$

$$P(\text{Windy=yes} \mid \text{Tennis=no}) = 2/3$$

$$P(\text{Windy=no} \mid \text{Tennis=no}) = 1/3$$

$$P(\text{Windy=yes}) = 3/6 = 1/2$$

$$P(\text{Windy=no}) = 3/6 = 1/2$$

$$P(\text{Tennis=yes}) = 3/6 = 1/2$$

$$P(\text{Tennis=no}) = 3/6 = 1/2$$

Frequency Table	Tennis = yes	Tennis = no	Total
Windy = yes	1	2	3
Windy = no	2	1	3
Total	3	3	6

# Applying Bayes' Theorem

- Let us consider output class  $c$  and input value(s)  $a$ . Bayes' Theorem can be rewritten as

$$P(c | a) = \frac{P(a | c)P(c)}{P(a)}$$

- Now, given input value(s)  $a$ , we calculate the above for every class  $c$ : our prediction is the one with:  $\max_c P(c | a)$

$$P(Tennis = yes | Windy = yes) = \frac{P(Windy = yes | Tennis = yes)P(Tennis = yes)}{P(Windy = yes)}$$

# Applying Bayes' Theorem (continued)

$$\begin{aligned} P(\text{Tennis} = \text{yes} \mid \text{Windy} = \text{yes}) &= \frac{P(\text{Windy} = \text{yes} \mid \text{Tennis} = \text{yes})P(\text{Tennis} = \text{yes})}{P(\text{Windy} = \text{yes})} \\ &= \frac{\frac{1}{3} * \frac{3}{6}}{\frac{3}{6}} = 0.33 \end{aligned}$$

Frequency Table	Tennis = yes	Tennis = no	Total
Windy = yes	1	2	3
Windy = no	2	1	3
Total	3	3	6

# Applying Bayes' Theorem (continued)

$$P(Tennis = yes | Windy = yes) = \frac{P(Windy = yes | Tennis = yes)P(Tennis = yes)}{P(Windy = yes)}$$
$$= \frac{\frac{1}{3} * \frac{3}{6}}{\frac{3}{6}} = 0.33$$

$$P(Tennis = no | Windy = yes) = \frac{P(Windy = yes | Tennis = no)P(Tennis = no)}{P(Windy = yes)}$$

$$= \frac{\frac{2}{3} * \frac{3}{6}}{\frac{3}{6}} = 0.67$$

Frequency Table	Tennis = yes	Tennis = no	Total
Windy = yes	1	2	3
Windy = no	2	1	3
Total	3	3	6

# Applying Bayes' Theorem (continued)

$$P(Tennis = yes \mid Windy = yes) = 0.33$$

$$P(Tennis = no \mid Windy = yes) = 0.67$$

$$\max_c P(c \mid a) = \max \{0.33, 0.67\} = 0.67$$

Frequency Table	Tennis = yes	Tennis = no	Total
Windy = yes	1	2	3
Windy = no	2	1	3
Total	3	3	6

# Normalising Factor

$$P(Tennis = yes | Windy = yes) = \frac{P(Windy = yes | Tennis = yes)P(Tennis = yes)}{P(Windy = yes)}$$
$$= \frac{\frac{1/3 * 3/6}{3/6}}{3/6} = 0.33$$

$$P(Tennis = no | Windy = yes) = \frac{P(Windy = yes | Tennis = no)P(Tennis = no)}{P(Windy = yes)}$$
$$= \frac{\frac{2/3 * 3/6}{3/6}}{3/6} = 0.67$$

- $1/P(Windy = yes)$  can be seen as a normalisation constant for the distribution: we can replace it with the constant parameter  $\alpha = 1/\beta$
- $\beta = \sum_{c \in \mathcal{Y}} P(c)P(a|c)$

# More than 1 Independent Variable

$$P(c|a_1, \dots, a_n) = \frac{P(a_1, \dots, a_n | c)P(c)}{\sum_{c \in \mathcal{Y}} P(c) \prod_{i=1}^n P(a_i | c)} = \alpha P(a_1, \dots, a_n | c)P(c)$$

- $P$  represents the probability calculated based on the frequency tables
- $c$  represents a class
- $a_i$  represents the value of independent variable  $x_i \in \{1, \dots, n\}$
- $n$  is the number of independent variables
- $\alpha$  is the normalisation factor

# Problems: Scaling and Missing Values

	toothache		¬toothache	
	catch	¬catch	catch	¬catch
cavity	0.108	0.012	0.072	0.008
¬cavity	0.016	0.064	0.144	0.576

- In this example (from the book), we have 3 Boolean variables
- For a domain described by  $n$  Boolean variables, we would need an input table of size  $O(2^n)$  and it would take  $O(2^n)$  to process the table
- Also, it is reasonable to think that we will never see values for all possible combinations of the variables
- Naïve Bayes can be used to deal with these issues

# Overview

- Fundamental concepts in Probability Theory
- Bayes' Theorem
- **Naïve Bayes for Categorical Independent Variables**
- Naïve Bayes for Numerical Independent Variables

# Recall: Issues with Bayes' Theorem

	toothache		¬toothache	
	catch	¬catch	catch	¬catch
cavity	0.108	0.012	0.072	0.008
¬cavity	0.016	0.064	0.144	0.576

- For increasing numbers of independent variables, all possible combinations must be considered:  
$$P(c|a_1, \dots, a_n) = \alpha P(c) \prod_{i=1}^n P(a_i | c)$$
- For a domain described by  $n$  Boolean variables, we would need an input table of size  $O(2^n)$  and it would take  $O(2^n)$  to process the table

# Naïve Bayes: Conditional Independence

- Assumption: each input variable is **conditionally independent** of any other input variables given the output
- **Independence:**  $A$  is **independent** of  $B$  when the following equality holds (i.e.,  $B$  does not alter the probability that  $A$  has occurred):  
$$P(A|B) = P(A)$$
- **Conditional independence:**  $x_1$  is **conditionally independent** of  $x_2$  given  $y$  when the following equality holds:  
$$P(x_1|x_2, y) = P(x_1, y)$$

# Naïve Bayes

- **Conditional independence:**  $x_1$  is **conditionally independent** of  $x_2$  given  $y$  when the following equality holds:

$$P(x_1|x_2, y) = P(x_1, y)$$

$$P(c|a_1, \dots, a_n) = \alpha P(c) \textcolor{red}{P(a_1, \dots, a_n|c)}$$

# Naïve Bayes

- **Conditional independence:**  $x_1$  is **conditionally independent** of  $x_2$  given  $y$  when the following equality holds:

$$P(x_1|x_2, y) = P(x_1, y)$$

$$P(c|a_1, \dots, a_n) = \alpha P(c) P(a_1, \dots, a_n|c)$$



$$P(c|a_1, \dots, a_n) = \alpha P(c) P(a_1|c)P(a_2|c) \dots P(a_n|c)$$

# Naïve Bayes

$$P(c|a_1, \dots, a_n) = \alpha P(c) P(a_1|c)P(a_2|c) \dots P(a_n|c)$$

$$\downarrow$$
$$P(c|a_1, \dots, a_n) = \alpha P(c) \prod_{i=1}^n P(a_i|c)$$

where  $\alpha = 1/\beta$  and  $\beta = \sum_{c \in \mathcal{Y}} (P(c) \prod_{i=1}^n P(a_i|c))$

# Example: Naïve Bayes

- Consider again the **training set**

Days	Sunny ( $X_1$ )	Windy ( $X_2$ )	Tennis ( $Y$ )
Day 1	yes	no	yes
Day 2	yes	no	yes
Day 3	yes	yes	yes
Day 4	no	yes	no
Day 5	no	no	no
Day 6	no	yes	no

- Because of conditional independence, we have a table for each variable:

Frequency Table	Tennis = yes	Tennis = no	Total
Windy = yes	1	2	3
Windy = no	2	1	3
Total	3	3	6

Frequency Table	Tennis = yes	Tennis = no	Total
Sunny = yes	3	0	3
Sunny = no	0	3	3
Total	3	3	6

# Example: Naïve Bayes (continued)

- Let us determine the predicted class for the following instance:  
**(Windy = no, Sunny = no, Y = ?)**

Frequency Table	Tennis = yes	Tennis = no	Total
Windy = yes	1	2	3
Windy = no	2	1	3
Total	3	3	6

Frequency Table	Tennis = yes	Tennis = no	Total
Sunny = yes	3	0	3
Sunny = no	0	3	3
Total	3	3	6

- $P(c|a_1, \dots, a_n) = \alpha P(c) \prod_{i=1}^n P(a_i|c)$
- $P(\neg T|\neg W, \neg S) = \alpha P(\neg T)P(\neg W|\neg T)P(\neg S|\neg T) = \alpha \frac{3}{6} * \frac{1}{3} * \frac{3}{3} = \frac{1}{6} \alpha$
- $P(T|\neg W, \neg S) = \alpha P(T)P(\neg W|T)P(\neg S|T) = \alpha \frac{3}{6} * \frac{2}{3} * \frac{0}{3} = 0$

# Example: Naïve Bayes (continued)

- $P(\neg T|\neg W, \neg S) = \alpha P(\neg T)P(\neg W|\neg T)P(\neg S|\neg T) = \alpha \frac{3}{6} * \frac{1}{3} * \frac{3}{3} = \frac{1}{6}\alpha$
- $P(T|\neg W, \neg S) = \alpha P(T)P(\neg W|T)P(\neg S|T) = \alpha \frac{3}{6} * \frac{2}{3} * \frac{0}{3} = 0$
- $\alpha = \frac{1}{\beta} = \frac{1}{\frac{3}{6} * \frac{2}{3} * \frac{0}{3} + \frac{3}{6} * \frac{1}{3} * \frac{3}{3}} = 6$
- $P(\neg T|\neg W, \neg S) = \frac{1}{6} * 6 = 1$
- $P(T|\neg W, \neg S) = 0$
- Problem: in this example, there is no data where Tennis = yes with Sunny = no, so regardless of the value of Windy, we will get inaccuracies in doing predictions

# Laplace Smoothing

- To avoid this problem, we can use Laplace smoothing by adding 1 to the frequency of all elements of our training data

Frequency Table	Tennis = yes	Tennis = no	Total
Windy = yes	1+1	2+1	3+2
Windy = no	2+1	1+1	3+2
Total	3+2	3+2	6+4

Frequency Table	Tennis = yes	Tennis = no	Total
Sunny = yes	3+1	0+1	3+2
Sunny = no	0+1	3+1	3+2
Total	3+2	3+2	6+4

- Then we use the updated tables when calculating  $P(a_i|c)$ , so we do not get values with 0
- When we calculate  $P(c)$ , we use the original tables

# Summary

## Naïve Bayes Learning Algorithm

- Create frequency tables for each independent variable and the corresponding values for the frequency of an event
- Count the number of training examples of each class with each independent variable
- Apply Laplace smoothing

## Naïve Bayes Model

- Consists of the frequency tables obtained from Bayes' Theorem under the conditional independence assumption (with or without Laplace smoothing)

## Naïve Bayes prediction for an instance ( $X=a, Y=?$ )

- We use Bayes' Theorem under the conditional independence assumption

# Overview

- Fundamental concepts in Probability Theory
- Bayes' Theorem
- Naïve Bayes for Categorical Independent Variables
- **Naïve Bayes for Numerical Independent Variables**

# Naïve Bayes for Numerical Independent Variables

$$P(c|a_1, \dots, a_n) = \alpha P(c) P(a_1|c)P(a_2|c) \dots P(a_n|c)$$

$$P(c|a_1, \dots, a_n) = \alpha P(c) \prod_{i=1}^n P(a_i|c)$$



where  $\alpha = 1/\beta$  and  $\beta = \sum_{c \in \mathcal{Y}} (P(c) \prod_{i=1}^n P(a_i|c))$

- We predict the class with  $\max_c [P(c|a_1, \dots, a_n)]$

# Naïve Bayes for Numerical Independent Variables

- For categorical independent variables, we can compute the probability of an event through the probability mass function associated with the training data

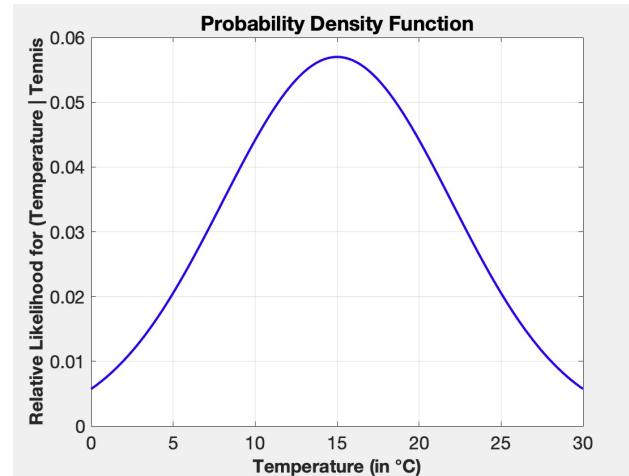
Frequency Table	Tennis = yes	Tennis = no	Total
Windy = yes	1+1	2+1	3+2
Windy = no	2+1	1+1	3+2
Total	3+2	3+2	6+4

# Naïve Bayes for Numerical Independent Variables

- Instead, we assume that examples are drawn from a probability distribution. We can use a Gaussian distribution as we did before
- Gaussian distribution with mean  $\mu = 15$  and variance  $\sigma^2 = 49$

$$P(X = a \mid \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(a-\mu)^2}{2\sigma^2}},$$

Also, recall that  $\pi = 3.14159$  and  $e = 2.71828$



# Naïve Bayes for Numerical Independent Variables

- Let us consider the training data below. We create the PDF for Tennis = yes and for Tennis = no
- So, for Tennis = yes, we calculate mean and variance

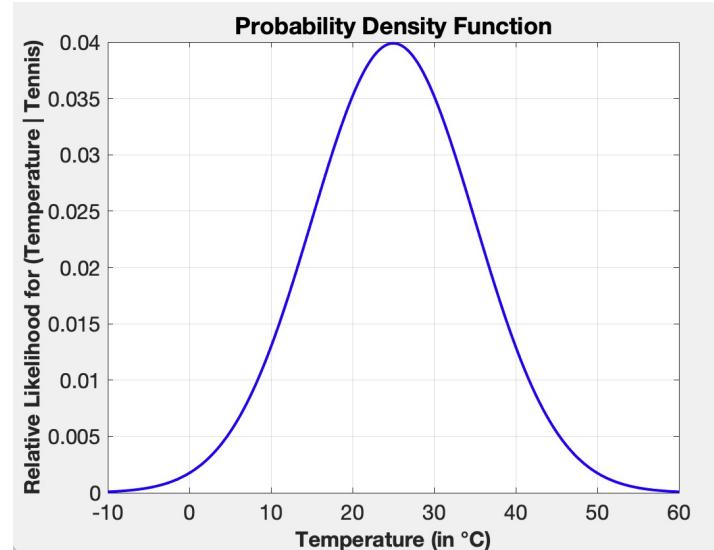
$$\mu = \frac{1}{n} \sum_{i=1}^n x_i = \frac{15 + 25 + 35}{3} = 25$$
$$\sigma^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \mu)^2 = \frac{1}{2} [(15 - 25)^2 + (25 - 25)^2 + (35 - 25)^2] = 100$$

Days	Sunny ( $X_1$ )	Temp. ( $X_2$ )	Tennis ( $Y$ )
Day 1	yes	15	yes
Day 2	yes	25	yes
Day 3	yes	35	yes
Day 4	no	10	no
Day 5	no	20	no
Day 6	no	5	no

# Naïve Bayes for Numerical Independent Variables

- Gaussian distribution with mean  $\mu = 25$  and variance  $\sigma^2 = 100$

$$P(X = a | \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(a-\mu)^2}{2\sigma^2}}$$



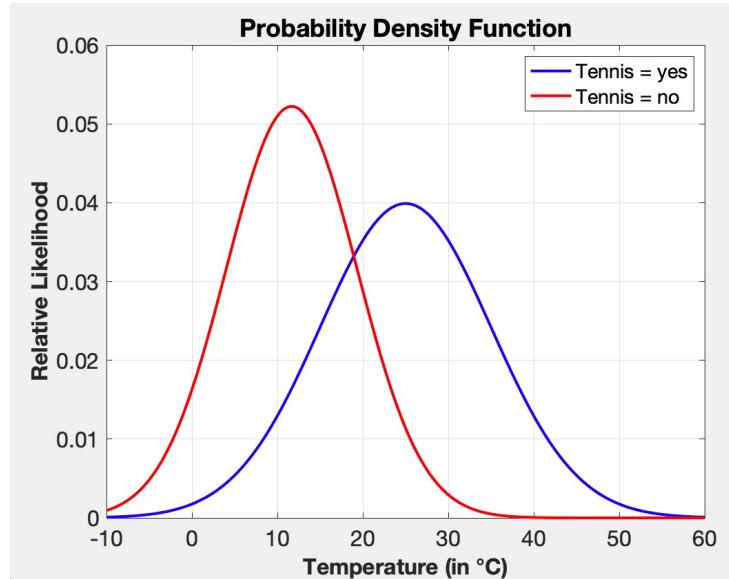
# Naïve Bayes for Numerical Independent Variables

- Gaussian distribution with mean  $\mu = 25$  and variance  $\sigma^2 = 100$

$$P(X = a | \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(a-\mu)^2}{2\sigma^2}}$$

Now, if we repeat for Tennis = no

- $\mu = 11.67$
- $\sigma^2 = 58.34$



# Example

- Let us build the tables for

Days	Sunny ( $X_1$ )	Temp. ( $X_2$ )	Tennis ( $Y$ )
Day 1	yes	15	yes
Day 2	yes	25	yes
Day 3	yes	35	yes
Day 4	no	10	no
Day 5	no	20	no
Day 6	no	5	no

Frequency Table	Tennis = yes	Tennis = no	Total
Sunny = yes	3+1	0+1	3+2
Sunny = no	0+1	3+1	3+2
Total	3+2	3+2	6+4

Parameter Table	Tennis = yes	Tennis = no
$\mu$	25	11.67
$\sigma^2$	100	58.34

# Example

- Now, let us use Naïve Bayes to make a prediction based on the tables:

Frequency Table	Tennis = yes	Tennis = no	Total
Sunny = yes	3+1	0+1	3+2
Sunny = no	0+1	3+1	3+2
Total	3+2	3+2	6+4

Parameter Table	Tennis = yes	Tennis = no
$\mu$	25	11.67
$\sigma^2$	100	58.34

- $P(c|a_1, \dots, a_n) = \alpha P(c) \prod_{i=1}^n P(a_i|c)$
- We use the frequency table for the categorical independent variables
- We use the parameter table for the numerical independent variables

# Example

- Calculate  $P(\text{Tennis=yes} | \text{Sunny=no}, \text{Temperature}=20)$ :

Frequency Table	Tennis = yes	Tennis = no	Total
Sunny = yes	3+1	0+1	3+2
Sunny = no	0+1	3+1	3+2
Total	3+2	3+2	6+4

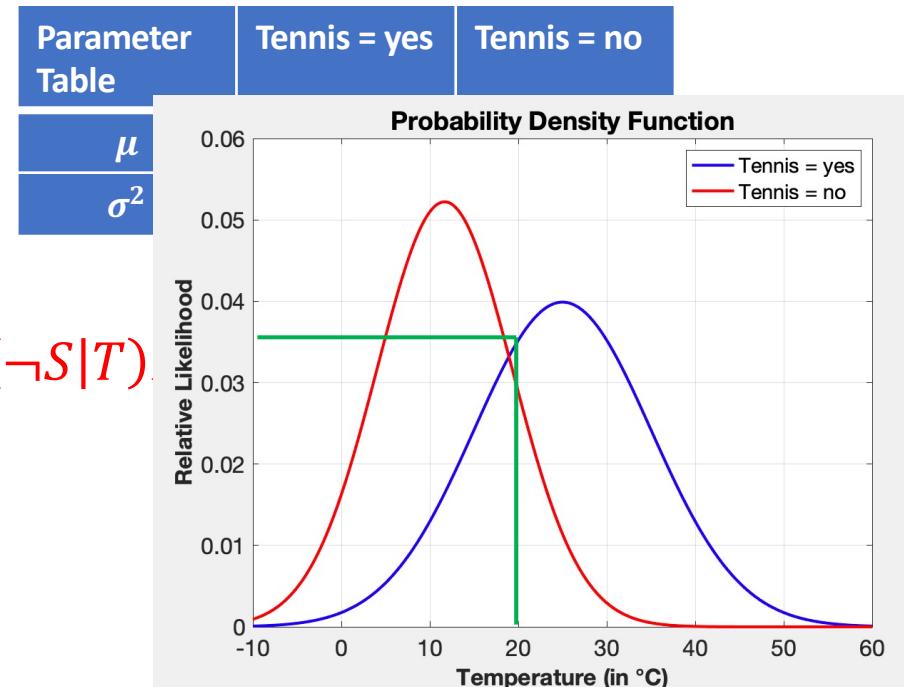
Parameter Table	Tennis = yes	Tennis = no
$\mu$	25	11.67
$\sigma^2$	100	58.34

- $$P(T | \neg S, Temp = 20) = \alpha P(T)P(\neg S|T)P(Temp = 20|T)$$
$$= \alpha \frac{3}{6} * \frac{1}{5} * P(Temp = 20|T)$$

# Example

- Calculate  $P(\text{Tennis=yes} | \text{Sunny=no}, \text{Temperature}=20)$ :

Frequency Table	Tennis = yes	Tennis = no	Total
Sunny = yes	3+1	0+1	3+2
Sunny = no	0+1	3+1	3+2
Total	3+2	3+2	6+4



- $$P(T | \neg S, \text{Temp} = 20) = \alpha P(T) P(\neg S | T)$$

$$= \alpha \frac{3}{6} * \frac{1}{5} * 0.035 = 0.0035\alpha$$

# Example

- Calculate  $P(\text{Tennis}=\text{no} \mid \text{Sunny}=\text{no}, \text{Temperature}=20)$ :

Frequency Table	Tennis = yes	Tennis = no	Total
Sunny = yes	3+1	0+1	3+2
Sunny = no	0+1	3+1	3+2
Total	3+2	3+2	6+4

Parameter Table	Tennis = yes	Tennis = no
$\mu$	25	11.67
$\sigma^2$	100	58.34

- $$P(\neg T \mid \neg S, \text{Temp} = 20) = \alpha P(\neg T)P(\neg S \mid \neg T)P(\text{Temp} = 20 \mid \neg T)$$
$$= \alpha \frac{3}{6} * \frac{4}{5} * 0.029 = 0.0116\alpha$$

# Example

- Calculate  $P(\text{Tennis}=\text{no} | \text{Sunny}=\text{no}, \text{Temperature}=20)$ :
- $$P(T | \neg S, \text{Temp} = 20) = \alpha P(T) P(\neg S | T) P(\text{Temp} = 20 | T)$$
$$= \alpha \frac{3}{6} * \frac{1}{5} * 0.035 = 0.0035\alpha$$
- $$P(\neg T | \neg S, \text{Temp} = 20) = \alpha P(\neg T) P(\neg S | \neg T) P(\text{Temp} = 20 | \neg T)$$
$$= \alpha \frac{3}{6} * \frac{4}{5} * 0.029 = 0.0116\alpha$$
- Predicted class: Tennis = no

# Summary

## Naïve Bayes Learning Algorithm

- Create frequency tables for each categorical independent variable and the corresponding values for the frequency of an event
- Apply Laplace smoothing
- Calculate the parameters of the PDF corresponding to each numerical independent variable

## Naïve Bayes Model

- Consists of the frequency tables obtained from Bayes' Theorem under the conditional independence assumption (with or without Laplace smoothing)

## Naïve Bayes prediction for an instance ( $X=a, Y=?$ )

- We use Bayes' Theorem under the conditional independence assumption

# Pros and Cons of Naïve Bayes

## Pros

- Easy to implement and fast to predict a class from training data (online learning)
- Performs well in multi-class prediction
- Good for categorical variables in general

## Cons

- Data that are not observed require smoothing techniques to be applied
- For numerical variables, Gaussian distribution is assumed (strong assumption)
- Not good for regression problems

# Aims of the Session

You should now be able to:

- Describe the fundamental concepts in probability theory
- Explain Bayes' Theorem and its application in ML
- Apply Naïve Bayes to classification for categorical and numerical independent variables



UNIVERSITY OF  
BIRMINGHAM

# AI1/AI&ML - k-Nearest Neighbours

Dr Leonardo Stella



# Aims of the Session

This session aims to help you:

- Describe the steps of the k-Nearest Neighbours algorithm
- Apply k-Nearest Neighbours to problems involving numeric, ordinal and categorical input attributes

# Overview

- **Notation**
- k-Nearest Neighbour
- k-NN algorithm and pros/cons

# Notation

- Probabilistic models
  - Variables are denoted by uppercase letters, e.g.,  $X$  or  $Y$
  - Values that a variable can take are denoted by lowercase letters, e.g.,  $x_1$
  - Vectors are denoted by letters in bold, e.g.,  $\mathbf{X}$  or  $\mathbf{x}$

# Notation

- Probabilistic models
  - Variables are denoted by uppercase letters, e.g.,  $X$  or  $Y$
  - Values that a variable can take are denoted by lowercase letters, e.g.,  $x_1$
  - Vectors are denoted by letters in bold, e.g.,  $\mathbf{X}$  or  $\mathbf{x}$
- In this lecture, we will be using the following notation:
  - Variables are denoted by lowercase letters, e.g.,  $x$  or  $y$
  - Values are typically stated, and letters are generally not used to represent values
  - Vectors are still denoted by letters in bold, e.g.,  $\mathbf{x}$

# Nonparametric Models

- A nonparametric model is a model that cannot be characterised by a bounded set of parameters
  - For instance, suppose that each prediction we make will consider all training examples, including the one from the previous prediction(s)
  - The set of examples grows over time, thus nonparametric

# Nonparametric Models

- A nonparametric model is a model that cannot be characterised by a bounded set of parameters
  - For instance, suppose that each prediction we make will consider all training examples, including the one from the previous prediction(s)
  - The set of examples grows over time, thus nonparametric
- This approach is also called **instance- or memory-based learning**
  - The simplest method for instance-based learning is **table lookup**
  - For table lookup, we put all training examples in a table, and when looking for a value, we return the corresponding value
  - Problem: if the value does not exist, then a default value is returned

# Overview

- Notation
- **k-Nearest Neighbour**
- k-NN algorithm and pros/cons

# k-Nearest Neighbours (k-NN or KNN)

- Consider the following two-dimensional problem ( $x_1, x_2$ )
  - Two classes: green or red ( $y \in \{g, r\}$ )
  - New example (blue) to classify (majority vote)

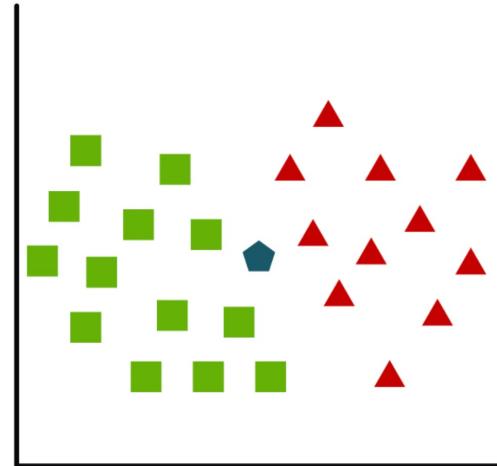


Image: taken from Ashraf et al., "A Review of Intrusion Detection Systems Using Machine and Deep Learning in Internet of Things: Challenges, Solutions and Future Directions", *Electronics*, vol. 9, no. 7, 2020.

# k-Nearest Neighbours (k-NN or KNN)

- Consider the following two-dimensional problem ( $x_1, x_2$ )

- Two classes: green or red ( $y \in \{g, r\}$ )
- New example (blue) to classify (majority vote)
- Look at the k nearest neighbours
- Let  $k = 3$ , to avoid issues
- We want to predict the class of the new example

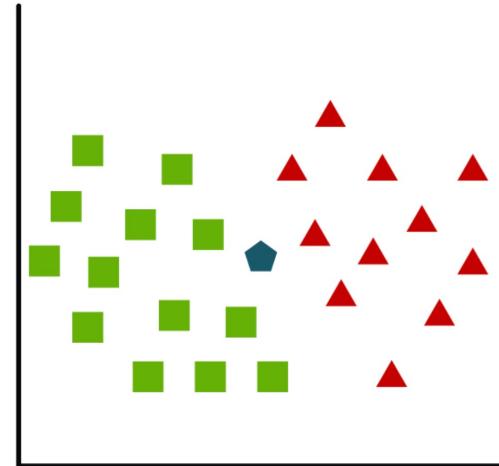


Image: taken from Ashraf et al., "A Review of Intrusion Detection Systems Using Machine and Deep Learning in Internet of Things: Challenges, Solutions and Future Directions", *Electronics*, vol. 9, no. 7, 2020.

# k-Nearest Neighbours (k-NN or KNN)

- Consider the following two-dimensional problem ( $x_1, x_2$ )

- Two classes: green or red ( $y \in \{g, r\}$ )
- New example (blue) to classify (majority vote)
- Look at the k nearest neighbours
- Let  $k = 3$ , to avoid issues
- We want to predict the class of the new example
- In this case, we predict green
- Intuitively, a distance metric on the **input space**

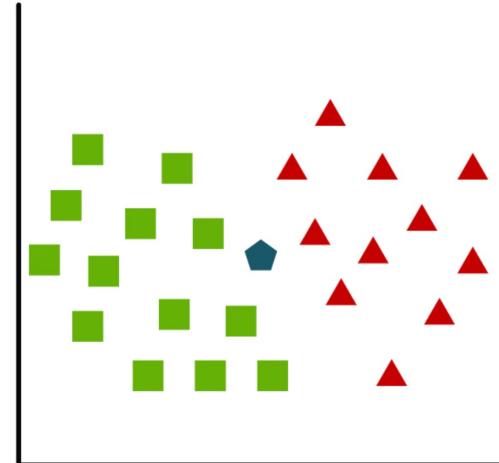


Image: taken from Ashraf et al., "A Review of Intrusion Detection Systems Using Machine and Deep Learning in Internet of Things: Challenges, Solutions and Future Directions", *Electronics*, vol. 9, no. 7, 2020.

# Distance Metrics

- Consider a problem with  $n$  dimensions,  $x^{[q]}$  being the new example
- The Minkowski distance (or  $L^p$  norm) is defined as

$$L^p(x^{[q]}, x^{[i]}) = \sqrt[p]{\sum_{j=1}^n |x_j^{[q]} - x_j^{[i]}|^p}$$

- In general, the Euclidean distance is used, namely when  $p = 2$

$$L^2(x^{[q]}, x^{[i]}) = \sqrt{2} \sum_{j=1}^n (x_j^{[q]} - x_j^{[i]})^2$$

# Example

- Let us consider a problem with 2 dimensions (green and red),  $x^{[4]} = [0.1, 0.6]$  being the new example and the following training examples are given:  $x^{[1]} = [0.4, 0.2] \in \{green\}$ ,  $x^{[2]} = [0.4, 0.1] \in \{green\}$ ,  $x^{[3]} = [0.2, 0.6] \in \{red\}$
- Let us use the Euclidean distance to predict the new example

$$L^2(x^{[4]}, x^{[1]}) = \sqrt{2} \sum_{j=1}^n (x_j^{[4]} - x_j^{[1]})^2 =$$

# Example

- Let us consider a problem with 2 dimensions (green and red),  $x^{[4]} = [0.1, 0.6]$  being the new example and the following training examples are given:  $x^{[1]} = [0.4, 0.2] \in \{green\}$ ,  $x^{[2]} = [0.4, 0.1] \in \{green\}$ ,  $x^{[3]} = [0.2, 0.6] \in \{red\}$
- Let us use the Euclidean distance to predict the new example

$$L^2(x^{[4]}, x^{[1]}) = \sqrt{\sum_{j=1}^n (x_j^{[4]} - x_j^{[1]})^2} = \sqrt{(x_1^{[4]} - x_1^{[1]})^2 + (x_2^{[4]} - x_2^{[1]})^2}$$

# Example

- Let us consider a problem with 2 dimensions (green and red),  $x^{[4]} = [0.1, 0.6]$  being the new example and the following training examples are given:  $x^{[1]} = [0.4, 0.2] \in \{green\}$ ,  $x^{[2]} = [0.4, 0.1] \in \{green\}$ ,  $x^{[3]} = [0.2, 0.6] \in \{red\}$
- Let us use the Euclidean distance to predict the new example

$$\begin{aligned}L^2(x^{[4]}, x^{[1]}) &= \sqrt{2 \sum_{j=1}^n (x_j^{[4]} - x_j^{[1]})^2} = \sqrt{(x_1^{[4]} - x_1^{[1]})^2 + (x_2^{[4]} - x_2^{[1]})^2} \\&= \sqrt{(0.1 - 0.4)^2 + (0.6 - 0.2)^2} = \sqrt{0.09 + 0.16} = 0.5\end{aligned}$$

# Example

- Let us consider a problem with 2 dimensions (green and red),  $\mathbf{x}^{[4]} = [0.1, 0.6]$  being the new example and the following training examples are given:  $\mathbf{x}^{[1]} = [0.4, 0.2] \in \{green\}$ ,  $\mathbf{x}^{[2]} = [0.4, 0.1] \in \{green\}$ ,  $\mathbf{x}^{[3]} = [0.2, 0.6] \in \{red\}$
- We repeat the process for all the points:

$$L^2(\mathbf{x}^{[4]}, \mathbf{x}^{[1]}) = 0.5$$

$$L^2(\mathbf{x}^{[4]}, \mathbf{x}^{[2]}) = 0.583$$

$$L^2(\mathbf{x}^{[4]}, \mathbf{x}^{[3]}) = 0.1$$

# Example

- Let us consider a problem with 2 dimensions (green and red),  $\mathbf{x}^{[4]} = [0.1, 0.6]$  being the new example and the following training examples are given:  $\mathbf{x}^{[1]} = [0.4, 0.2] \in \{green\}$ ,  $\mathbf{x}^{[2]} = [0.4, 0.1] \in \{green\}$ ,  $\mathbf{x}^{[3]} = [0.2, 0.6] \in \{red\}$
- We repeat the process for all the points:

$$L^2(\mathbf{x}^{[4]}, \mathbf{x}^{[1]}) = 0.5$$

$$L^2(\mathbf{x}^{[4]}, \mathbf{x}^{[2]}) = 0.583$$

$$L^2(\mathbf{x}^{[4]}, \mathbf{x}^{[3]}) = 0.1$$

# k-NN for Regression Problems

- Consider the following two-dimensional problem  $(x_1, x_2)$ 
  - Let us consider examples that take a value between 1 and 5 ( $y \in [1, 5]$ )
  - Assume each example has a value, e.g., 3.2 or 4.1
  - Look at the k nearest neighbours
  - We want to predict the value of the new example
  - In this case, we predict the average or median of the values of the k nearest neighbours

# Overview

- Notation
- k-Nearest Neighbour
- **k-NN algorithm and pros/cons**

# k-NN Algorithm

- Input: training examples  $x^{[i]} \in \mathcal{X}$  and their corresponding class  $y^{[i]}$ , a new query example  $x^{[q]}$ , number of neighbours k
- Output: prediction of the new query example  $x^{[q]}$
- For each training example  $x^{[i]} \in \mathcal{X}$ 
  - Calculate the distance between the training example  $x^{[i]}$  and the new query example  $x^{[q]}$
  - Keep the best k distances (the shortest distance) in a data structure T
- Return the majority vote (or average/median) of the class  $y^{[i]}$  for the first k entries of T

# Problem with Numeric Independent Variables

- Different numeric attributes may have different scales
- For example, if  $x_1$  is in  $[0,1]$  and  $x_2$  is in  $[1, 10]$ ,  $x_2$  will affect the distance more

# Problem with Numeric Independent Variables

- Different numeric attributes may have different scales
- For example, if  $x_1$  is in [0,1] and  $x_2$  is in [1, 10],  $x_2$  will affect the distance more
- To avoid this problem, we normalise the numeric input attributes of all data as in the following

$$\text{normalise}(x_j^{[i]}) = \frac{x_j^{[i]} - \min_j}{\max_j - \min_j}$$

- Another approach (see book) is to calculate mean  $\mu_j$  and standard deviation  $\sigma_j$  for each dimension  $j$  as:  $(x_j^{[i]} - \mu_j)/\sigma_j$

# Example: Normalisation

- Consider the following  $\boldsymbol{x}^{[1]} = [170, 50] \in \{yes\}$ , where  $x_1 = Age$  and  $x_2 = Weight$ ,  $y \in \{yes, no\}$
- Let us calculate the normalised values for  $\boldsymbol{x}^{[1]}$ :
- $$\text{normalise}(x_1^{[1]}) = \frac{x_1^{[1]} - \min_1}{\max_1 - \min_1} = \frac{14 - 12}{15 - 12} = 0.667$$
- $$\text{normalise}(x_2^{[1]}) = \frac{x_2^{[1]} - \min_2}{\max_2 - \min_2} = \frac{70 - 66}{90 - 66} = 0.167$$

Days	$x_1$	$x_2$	$y$
$\boldsymbol{x}^{[1]}$	14	70	yes
$\boldsymbol{x}^{[2]}$	12	90	no
$\boldsymbol{x}^{[3]}$	15	66	yes

# k-NN Algorithm with Normalisation

- Input: training examples  $x^{[i]} \in \mathcal{X}$  and their corresponding class  $y^{[i]}$ , a new query example  $x^{[q]}$ , number of neighbours k
- Output: prediction of the new query example  $x^{[q]}$
- For each training example  $x^{[i]} \in \mathcal{X}$ 
  - Calculate the **normalised** distance between the training example  $x^{[i]}$  and the new query example  $x^{[q]}$
  - Keep the best k distances (the shortest distance) in a data structure T
- Return the majority vote (or average/median) of the class  $y^{[i]}$  for the first k entries of T

# Different Input Attributes

- For **numeric** input attributes, e.g., age in  $[0, 100]$ , we calculate the distance as shown in previous examples
- For **ordinal** input attributes, e.g., sunny in  $\{\text{yes}, \text{no}\}$ , we can convert the values to numeric values: yes = 1, no = 0
- For **categorical** input attributes, e.g., phone\_brand in  $\{\text{samsung}, \text{apple}, \text{nokia}\}$ , we can use the following approach:
  - If the value of the query example is the same as the value for example  $i$ , then their difference is 0. Formally, if  $(x_j^{[q]} = x_j^{[i]})$ , then  $(x_j^{[q]} - x_j^{[i]}) = 0$
  - Otherwise, their difference is 1. Formally, if  $(x_j^{[q]} \neq x_j^{[i]})$ , then  $(x_j^{[q]} - x_j^{[i]}) = 1$

# Summary

## k-NN Learning Algorithm

- The algorithm does not have proper training
- We simply store all training data, which increase over time
- We normalise by calculating the minimum and maximum in the training data

## k-NN Model

- All training data, the values of the numeric input attributes

## k-NN prediction for an instance ( $x^{[i]}$ , $y = ?$ )

- Find the k nearest neighbours whose distance to  $x^{[i]}$  is the smallest
- For classification problems, majority vote. For regression problems, average/median

# Pros and Cons of k-NN

## Pros

- Training is simple and fast: just store training data
- Find the class of the new example based on most similar examples present in the training data

## Cons

- It uses large space in memory: we need to store all data
- Running the algorithm can be slow if we have many training examples and many dimensions

# Aims of the Session

You should now be able to:

- Describe the steps of the k-Nearest Neighbours algorithm
- Apply k-Nearest Neighbours to problems involving numeric, ordinal and categorical input attributes



UNIVERSITY OF  
BIRMINGHAM

# AI1/AI&ML - Uninformed Search

Dr Leonardo Stella



# Aims of the Session

This session aims to help you:

- Describe asymptotic analysis and why it is important
- Explain the steps to formulate a search problem
- Apply and compare the performance of Breadth-First Search, Depth-First Search and its variations

# Overview

- **Asymptotic Analysis**
- Search Problem Formulation
- Breadth-First Search
- Depth-First Search
- Variations of Depth-First Search

# Asymptotic Analysis

- Computer scientists are often asked to determine the quality of an algorithm by comparing it with other ones and measure the speed and memory required
- **Benchmarking** is one approach:
  - We run the algorithms and we measure speed (in seconds) and memory consumption (in bytes)
  - Problem: this approach measures the performance of a specific program written in a particular language, on a given computer, with particular input data
- **Asymptotic analysis** is the second approach:
  - It is a mathematical abstraction over both the exact number of operations (by ignoring constant factors) and exact content of the input (by considering the size of the input, only)
  - It is independent of the particular implementation and input

# Asymptotic Analysis

- The first step in the analysis is to abstract over the input. In practice, we characterise the size of the input, which we call  $n$
- The second step is to abstract over the implementation. The idea is to find some measure that reflects the running time of the algorithm
- For asymptotic analysis, we typically use 3 notations:
  - Big O notation:  $O(\cdot)$
  - Big Omega notation:  $\Omega(\cdot)$
  - Big Theta notation:  $\Theta(\cdot)$

# Asymptotic Analysis: Big O

- We say that  $f(n) \in O(g(n))$  when the following condition holds:

$$\exists k > 0 \exists n_0 \forall n > n_0: |f(n)| \leq k \cdot g(n)$$

- The above reads: “There exists a positive constant  $k, n_0$  such that for all  $n > n_0, |f(n)| \leq k \cdot g(n)$ ”
- In simple terms, this is equivalent to saying that  $|f|$  is bounded above by a function  $g$  (up to a constant factor) asymptotically

# Asymptotic Analysis: Big Theta and Big Omega

- We say that  $f(n) \in \Omega(g(n))$  when the following condition holds:

$$\exists k > 0 \exists n_0 \forall n > n_0: |f(n)| \geq k \cdot g(n)$$

- This is equivalent to saying that  $f$  is bounded below by  $g$  asymptotically
- We say that  $f(n) \in \Theta(g(n))$  when the following condition holds:

$$\exists k_1, k_2 > 0 \exists n_0 \forall n > n_0: k_1 \cdot g(n) \leq |f(n)| \leq k_2 \cdot g(n)$$

- Or  $f$  is bounded both above and below by  $g$  asymptotically

# Asymptotic Analysis: Example

- Consider the following algorithm (pseudocode):

**function** SUMMATION(*sequence*) **returns** a number

```
sum ← 0
for i = 1 to LENGTH(sequence) do
    sum ← sum + sequence[i]
return sum
```

- Step 1: abstract over input, e.g., the length of the *sequence*
- Step 2: abstract over the implementation, e.g., total number of steps. If we call this characterisation  $T(n)$  and we count lines of code, we have  
$$T(n) = 2n + 2$$

# Asymptotic Analysis: Example

- Consider the following algorithm (pseudocode):

**function** SUMMATION(*sequence*) **returns** a number

```
sum ← 0
for i = 1 to LENGTH(sequence) do
    sum ← sum + sequence[i]
return sum
```

- We say that the SUMMATION algorithm is  $O(n)$ , meaning that its measure is at most of constant times n with few possible exceptions
- $T(n) \in O(f(n))$  if  $T(n) \leq k \cdot f(n)$  for some  $k$ , for all  $n > n_0$
- For  $T(n) = 2n + 2$ , an example would be:  $k = 3, n_0 = 2$

# Summary

- Asymptotic analysis is a powerful tool to describe the speed and memory consumption of an algorithm
- It is useful as it is independent of a particular implementation and input
- It is an approximation as the input  $n$  approaches infinity and over the number of steps required
- Convenient to compare algorithms, e.g., an  $O(n)$  algorithm is better than an  $O(n^2)$  algorithm
- Other notations exist, such as  $\Omega(n)$  and  $\Theta(n)$

# Overview

- **Asymptotic Analysis**
- **Search Problem Formulation**
- Breadth-First Search
- Depth-First Search
- Variations of Depth-First Search

# Problem-Solving Agents

- In this lecture, we introduce the concept of a goal-based agent called **problem-solving agent**
- An agent is something that perceives and acts in an environment
- A problem-solving agent
  - Uses atomic representations (each state of the world is perceived as indivisible)
  - Requires a precise definition of the problem and its goal/solution

# Search Problem Formulation

- **Problem formulation** is the process of deciding what actions and states to consider, given a goal
- To this end, we make the following assumptions about the environment:
  - **Observable**, i.e., the agent knows the current state
  - **Discrete**, i.e., there are only finitely many actions at any state
  - **Known**, i.e., the agent knows which states are reached by each action
  - **Deterministic**, i.e., each action has exactly one outcome
- Under these assumptions, the solution to any problem is a fixed sequence of actions

# Search Problem Formulation

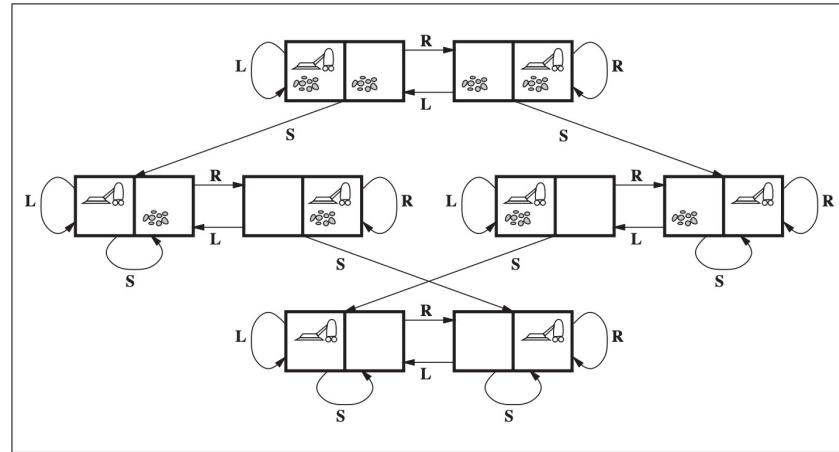
- The agent's task is to find out how to act, now and in the future, in order to reach a goal state: namely to determine a sequence of actions
- The process of looking for a sequence of actions is called **search**
- A solution to a search problem is the sequence of actions from the initial state to the goal state

# Search Problem Formulation

- A problem is defined formally by five components:
  - **Initial state**, i.e., the state that the agent starts in
  - **Actions**, i.e., a description of all possible actions that can be executed in a given state  $s$
  - **Transition model**, i.e., the states resulting from executing each action  $a$  from every state  $s$  (a description of what each action does)
  - **Goal test** to determine if a state is a goal state
  - **Path cost** function that assigns a value (cost) to each path
- The first three components considered together define the **state space** of the problem, in the form of a directed graph or network
- A path in the state space is a sequence of states connected by a sequence of actions

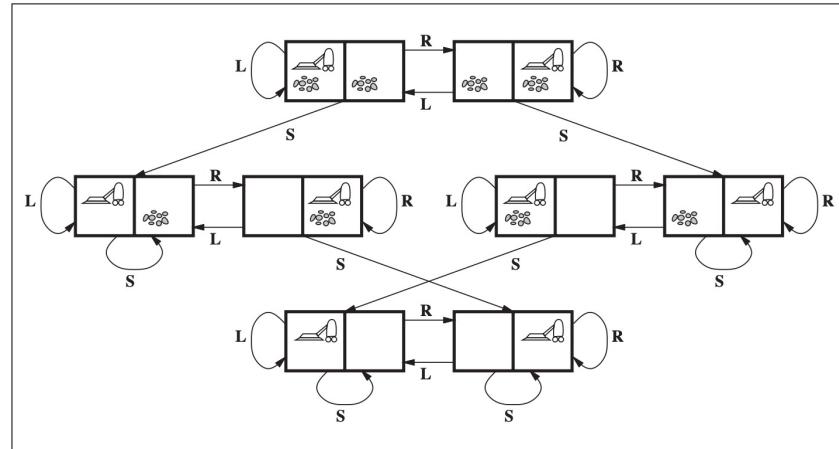
# Example: Vacuum World

- Let us consider the following example where the state is determined by the dirt location and agent location
  - Initial state:** any state
  - Actions:** L (left), R (right) and S (suction)
  - Transition model:** see image
  - Goal test:** checks if all squares are clean
  - Path cost:** each step costs 1



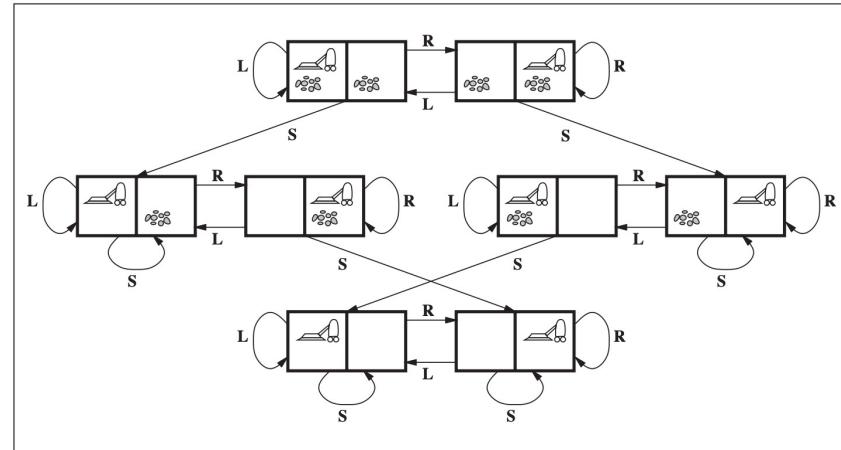
# Example: Vacuum World

- Let's find the solution when the initial state is the top-left state, namely the agent is in the left square, both squares are dirty
- Example of solution:** S (suction), R (right), S (suction)
- Cost of the solution:**  $1 + 1 + 1 = 3$



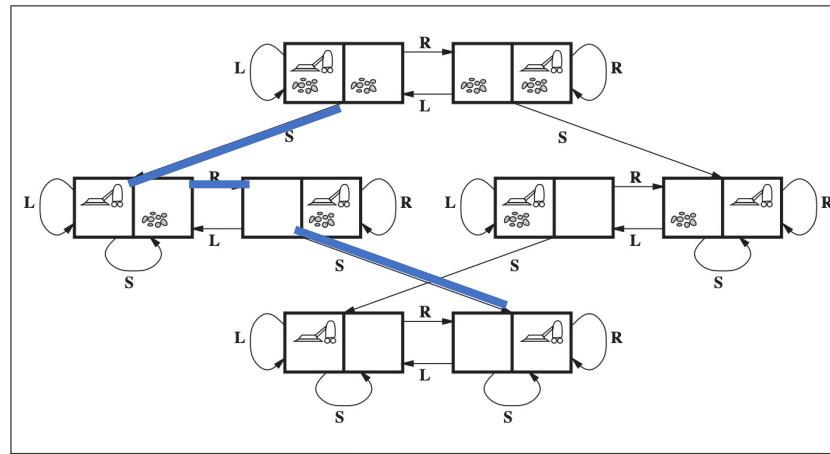
# Discussion

- It is important to note that typical AI problems have a large number of states and it is virtually impossible to draw the state space graph
- For the state space graph for the vacuum world example has a small number of states
- The state space graph for chess would be very large



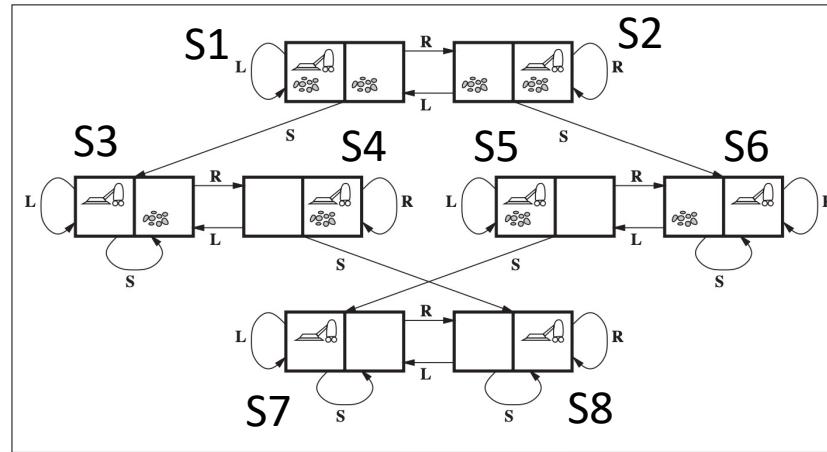
# Notation

- A solution can be seen as a path in the state space graph



# Notation

- A solution can be seen as a path in the state space graph
- Each state corresponds to a node in the state space graph



# Summary

- A **problem-solving agent** is an agent that is able to search for a solution in a given problem
- **Problem formulation**, namely the process of deciding what actions and states to consider, given a goal

# Overview

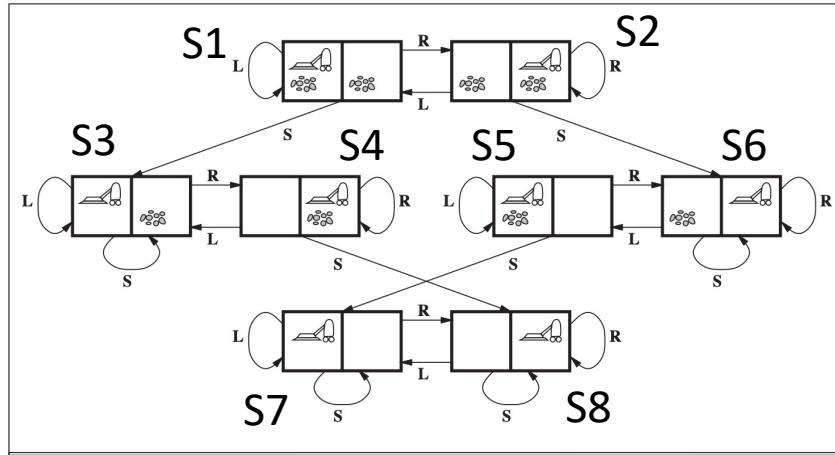
- Asymptotic Analysis
- Search Problem Formulation
- **Breadth-First Search**
- Depth-First Search
- Variations of Depth-First Search

# Searching for Solutions

- A solution is an action sequence from an initial state to a goal state
- Possible action sequences form a **search tree** with initial state at the root; actions are the branches and nodes correspond to the state space
- The idea is to expand the current state by applying each possible action: this generates a new set of states

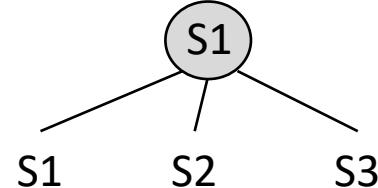
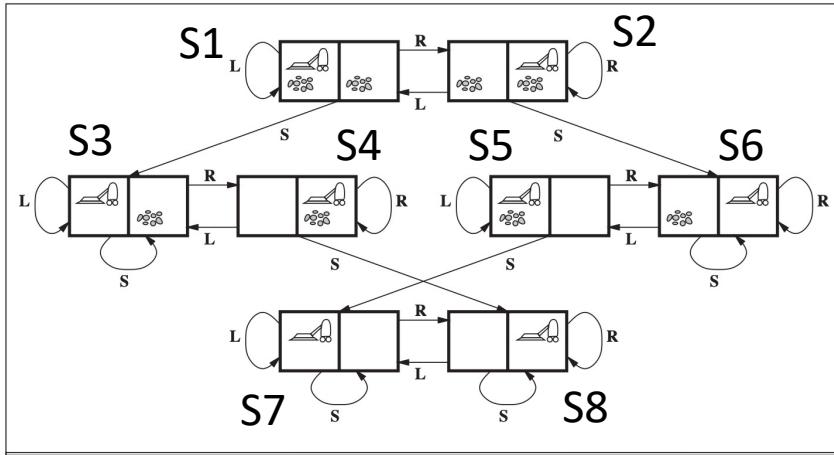
# Searching for Solutions

- Let us consider the example from before



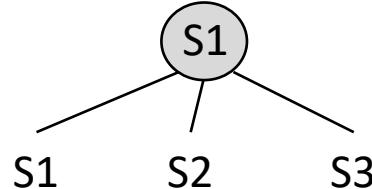
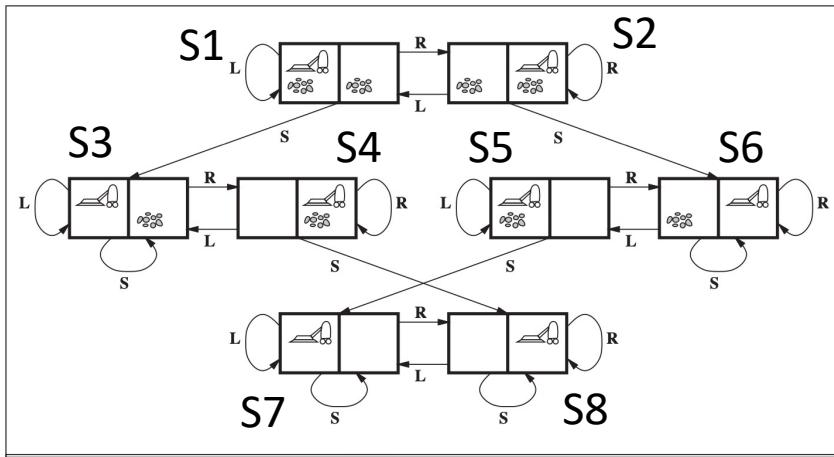
# Searching for Solutions

- Let us consider the example from before
- If  $S_1$  is the initial state and  $\{S_7, S_8\}$  is the set of goal states, the corresponding search tree after expanding the initial state is:



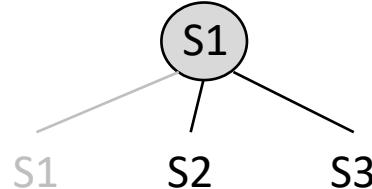
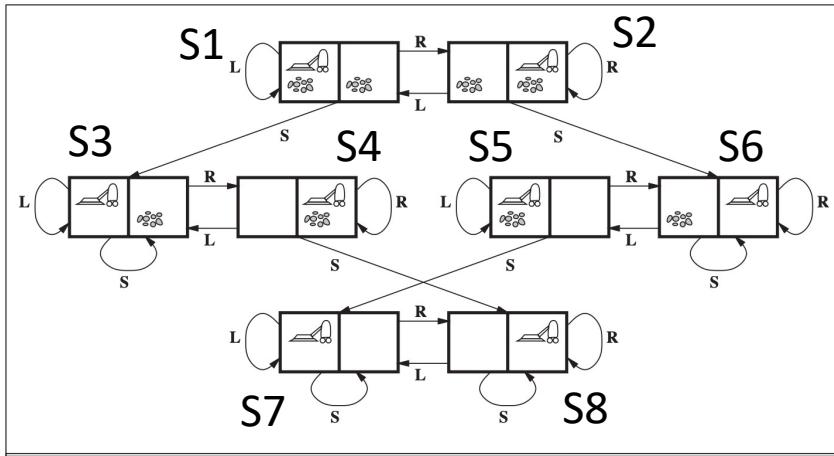
# Searching for Solutions

- Each of the three nodes resulting from the first expansion is a **leaf node**
- The set of all leaf nodes available for expansion at any given time is called the **frontier** (also sometimes called the **open list**)
- The path from S1 to S1 is a **loopy path** and in general is not considered



# Searching for Solutions

- Each of the three nodes resulting from the first expansion is a **leaf node**
- The set of all leaf nodes available for expansion at any given time is called the **frontier** (also sometimes called the **open list**)
- The path from S1 to S1 is a **loopy path** and in general is not considered



# Uninformed Search Strategies

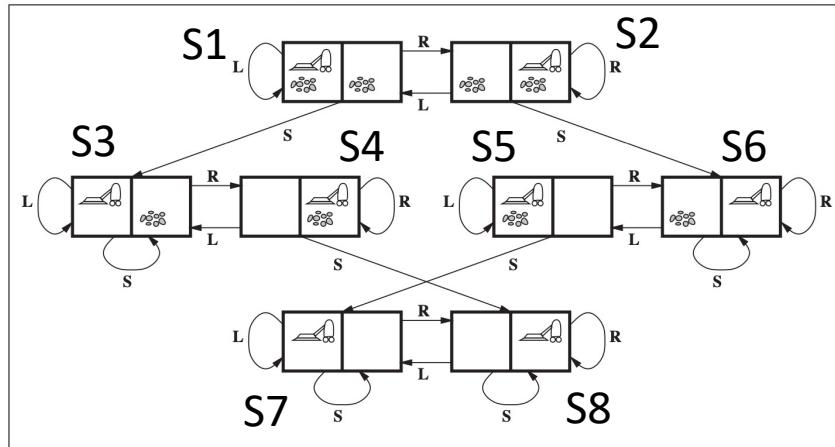
- **Uninformed search** (also called **blind search**) means that the strategies have no additional information about states beyond that provided in the problem definition
- Uninformed search strategies can only generate successors and distinguish a goal state from a non-goal state
- The key difference between two uninformed search strategies is the **order** in which nodes are expanded

# Breadth-First Search

- Breadth-First search is one of the most common search strategies:
  - The root node is expanded first
  - Then, all the successors of the root node are expanded
  - Then, the successors of each of these nodes
- In general, the frontier nodes that are expanded belong to a given depth of the tree
- This is equivalent to expanding the shallowest unexpanded node in the frontier; simply use a **queue (FIFO)** for expansion

# Breadth-First Search

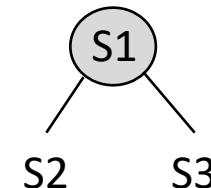
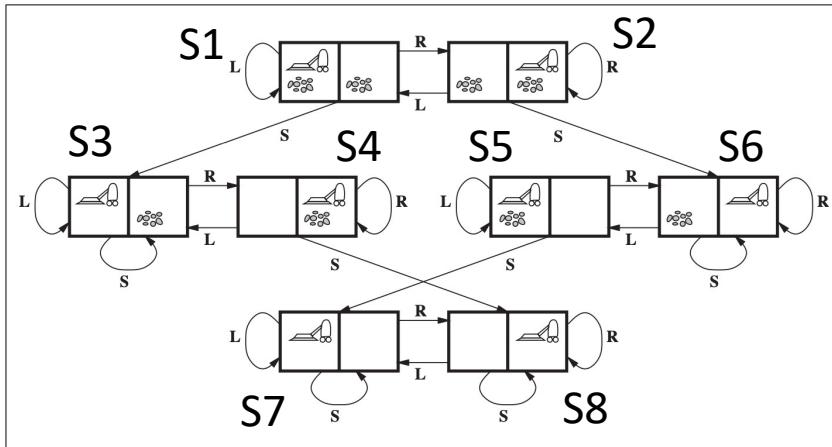
- Breadth-First search algorithm:
  - **Expand** the shallowest node in the frontier
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - **Stop** when a goal node is added to the frontier



S1

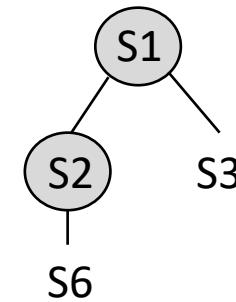
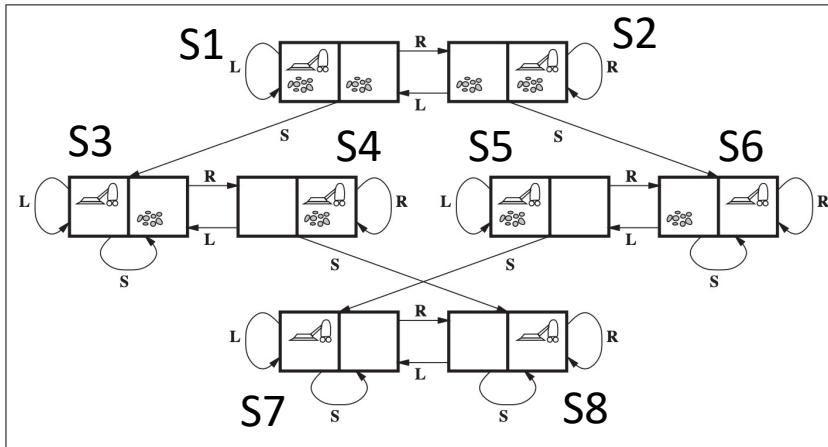
# Breadth-First Search

- Breadth-First search algorithm:
  - **Expand** the shallowest node in the frontier
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - **Stop** when a goal node is added to the frontier



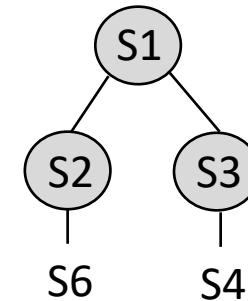
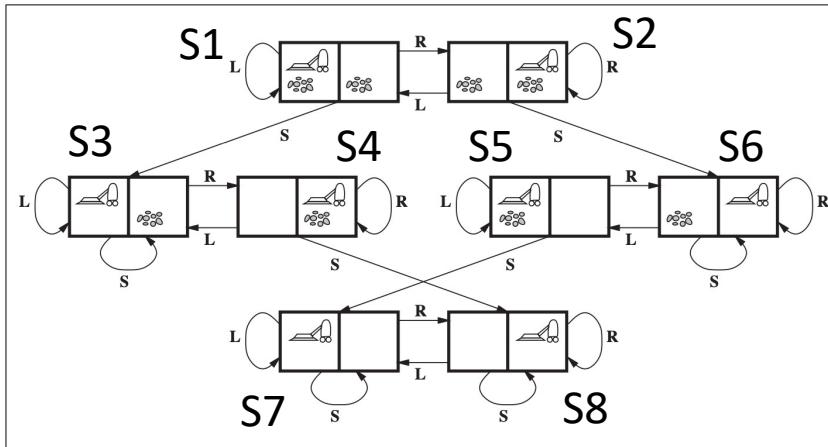
# Breadth-First Search

- Breadth-First search algorithm:
  - **Expand** the shallowest node in the frontier
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - **Stop** when a goal node is added to the frontier



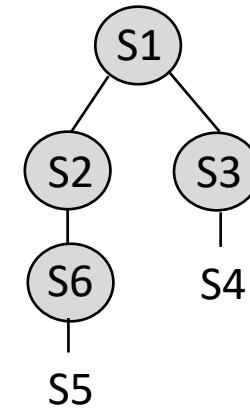
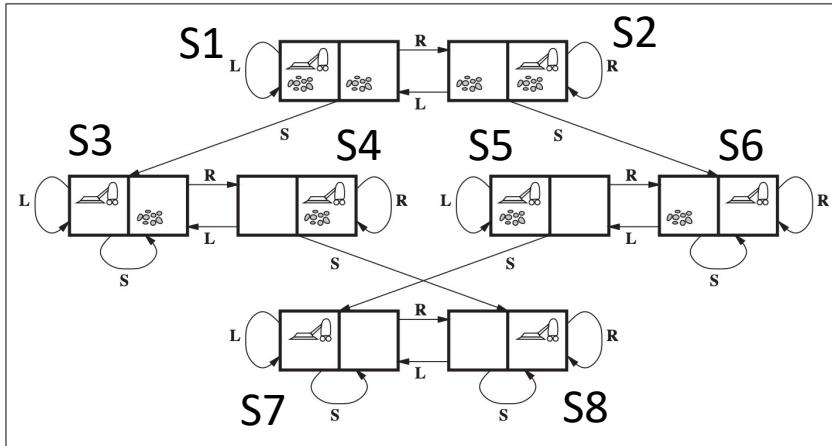
# Breadth-First Search

- Breadth-First search algorithm:
  - **Expand** the shallowest node in the frontier
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - **Stop** when a goal node is added to the frontier



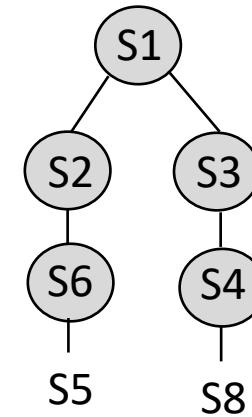
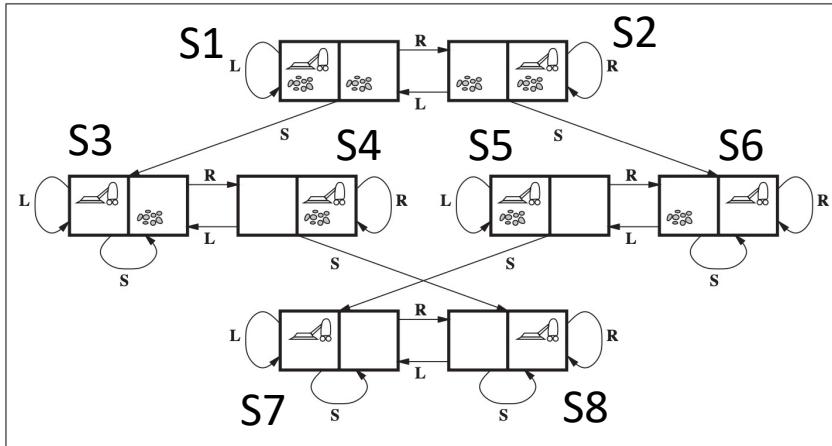
# Breadth-First Search

- Breadth-First search algorithm:
  - **Expand** the shallowest node in the frontier
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - **Stop** when a goal node is added to the frontier



# Breadth-First Search

- Breadth-First search algorithm:
  - **Expand** the shallowest node in the frontier
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - **Stop** when a goal node is added to the frontier



# Breadth-First Search

- Solution:

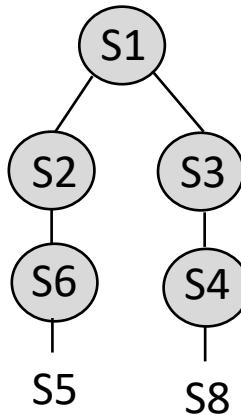
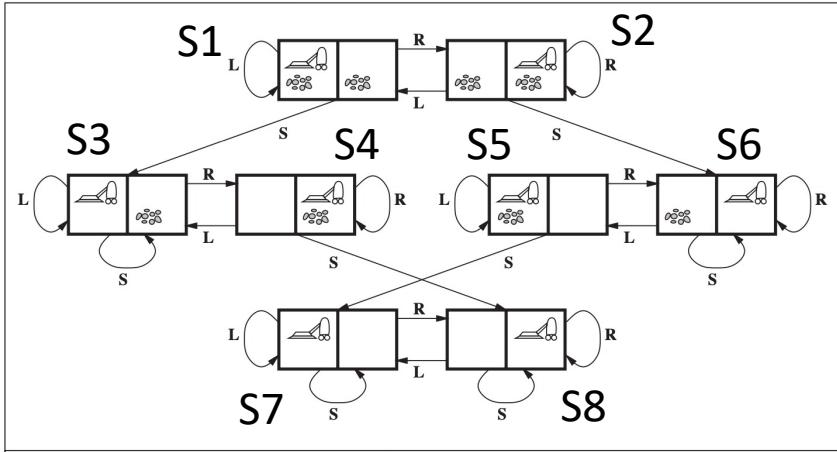
S, R, S

- Cost of the solution:

$$1 + 1 + 1 = 3$$

- Order of nodes visited

S1, S2, S3, S6, S4



# Measuring Performance

We can evaluate the performance of an algorithm based on the following:

- **Completeness**, i.e., whether the algorithm is guaranteed to find a solution if there is one
- **Optimality**, i.e., whether the strategy is able to find the optimal solution
- **Time complexity**, i.e., the time the algorithm takes to find a solution
- **Space complexity**, i.e., the memory used to perform the search

# Measuring Performance

We can evaluate the performance of an algorithm based on the following:

- **Completeness**, i.e., whether the algorithm is guaranteed to find a solution if there is one
- **Optimality**, i.e., whether the strategy is able to find the optimal solution
- **Time complexity**, i.e., the time the algorithm takes to find a solution
- **Space complexity**, i.e., the memory used to perform the search
  
- To measure the performance, the size of the space graph is typically used, i.e.,  $|\mathcal{V}| + |\mathcal{E}|$ , the set of vertices and set of edges, respectively

# Measuring Performance

- In AI, we use an implicit representation of the graph via the initial state, actions and transition model (also the graph could be infinite)
- Therefore, the following three quantities are used
  - **Branching factor**, the maximum number of successors of each node:  $b$
  - **Depth** of the shallowest goal node (number of steps from the root):  $d$
  - The maximum length of any path in the state space:  $m$

# BFS - Performance

Let us evaluate the performance of the breadth-first search algorithm

- **Completeness:** if the goal node is at some finite depth  $d$ , then the BFS algorithm **is complete** as it will find it (given that  $b$  is finite)
- **Optimality:** BFS **is optimal** if the path cost is a nondecreasing function of the depth of the node (e.g., all actions have the same cost)

# BFS - Performance

Let us evaluate the performance of the breadth-first search algorithm

- **Completeness:** if the goal node is at some finite depth  $d$ , then the BFS algorithm **is complete** as it will find it (given that  $b$  is finite)
- **Optimality:** BFS **is optimal** if the path cost is a nondecreasing function of the depth of the node (e.g., all actions have the same cost)
- **Time complexity:**  $O(b^d)$ , assuming a uniform tree where each node has  $b$  successors, we generate  $b + b^2 + \dots + b^d = O(b^d)$

# BFS - Performance

Let us evaluate the performance of the breadth-first search algorithm

- **Completeness:** if the goal node is at some finite depth  $d$ , then the BFS algorithm **is complete** as it will find it (given that  $b$  is finite)
- **Optimality:** BFS **is optimal** if the path cost is a nondecreasing function of the depth of the node (e.g., all actions have the same cost)
- **Time complexity:**  $O(b^d)$ , assuming a uniform tree where each node has  $b$  successors, we generate  $b + b^2 + \dots + b^d = O(b^d)$
- **Space complexity:**  $O(b^d)$ , if we store all expanded nodes, we have  $O(b^{d-1})$  explored nodes in memory and  $O(b^d)$  in the frontier

# Summary

- Uninformed tree search strategies have no additional information
- Breadth-First Search is a search algorithm that expands the nodes in the frontier starting from the shallowest, similar to a queue (FIFO)
- This algorithm is complete (for finite  $b$ ), optimal (if the path cost is nondecreasing), but it has high time and space complexity  $O(b^d)$

# Overview

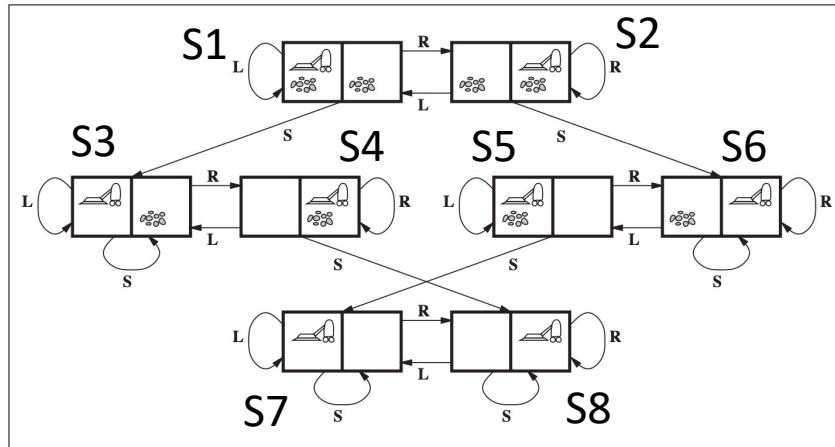
- Asymptotic Analysis
- Search Problem Formulation
- Breadth-First Search
- **Depth-First Search**
- Variations of Depth-First Search

# Depth-First Search

- Depth-First search is another common search strategy:
  - The root node is expanded first
  - Then, the first (or one at random) successor of the root node is expanded
  - Then, the deepest node in the current frontier is expanded
- This is equivalent to expanding the deepest unexpanded node in the frontier; simply use a **stack (LIFO)** for expansion
- Basically, the most recently generated node is chosen for expansion

# Depth-First Search

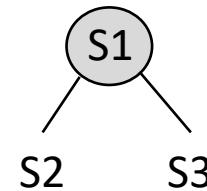
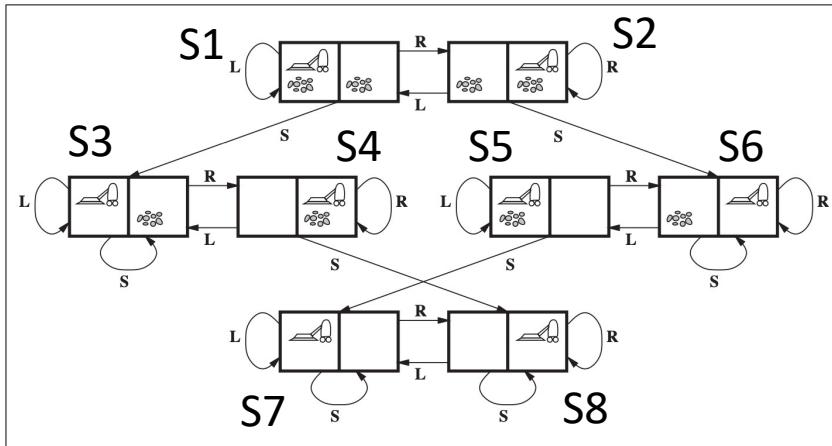
- Depth-First search algorithm:
  - **Expand** the deepest node in the frontier
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - **Stop** when a goal node is visited



S1

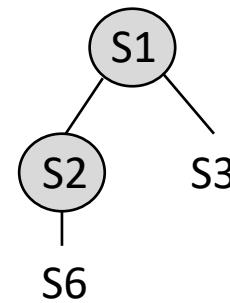
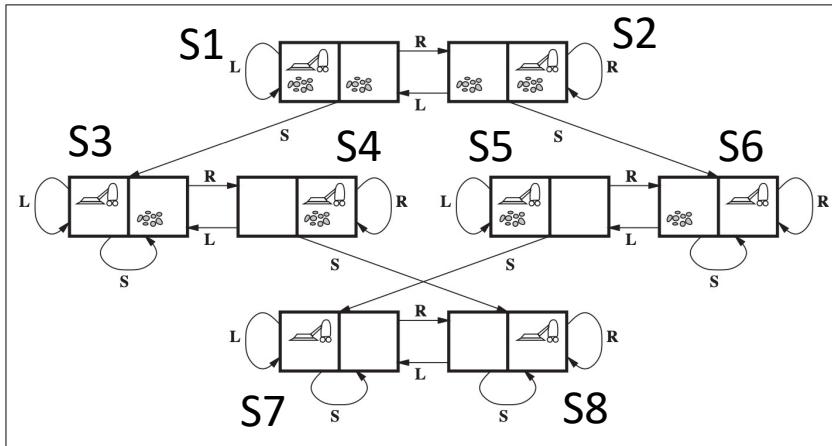
# Depth-First Search

- Depth-First search algorithm:
  - **Expand** the deepest node in the frontier
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - **Stop** when a goal node is visited



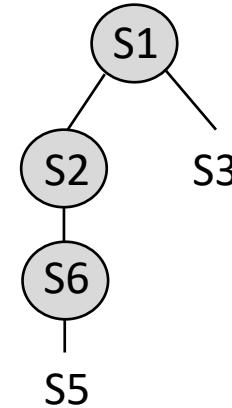
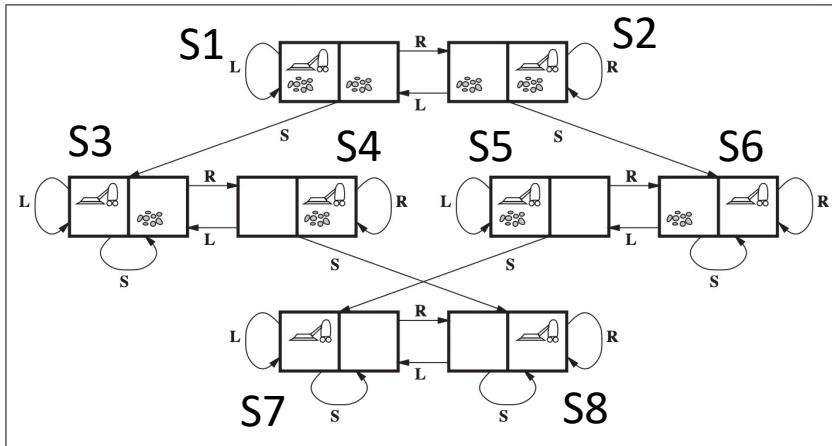
# Depth-First Search

- Depth-First search algorithm:
  - **Expand** the deepest node in the frontier
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - **Stop** when a goal node is visited



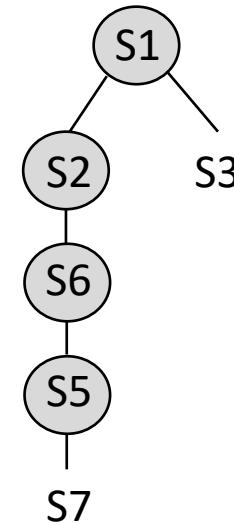
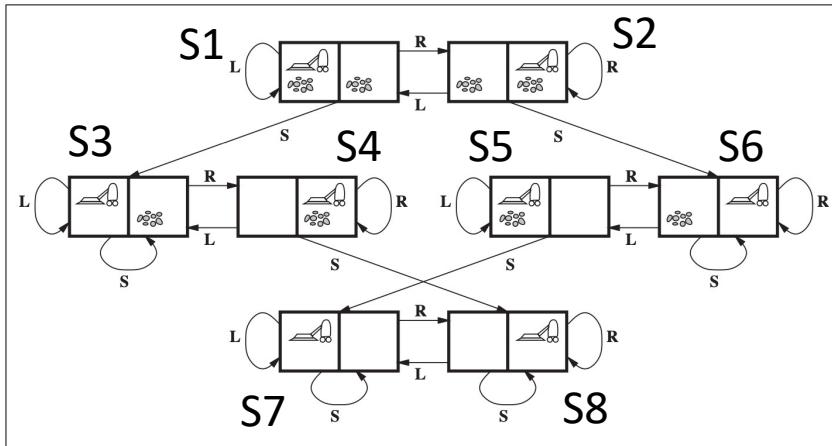
# Depth-First Search

- Depth-First search algorithm:
  - **Expand** the deepest node in the frontier
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - **Stop** when a goal node is visited



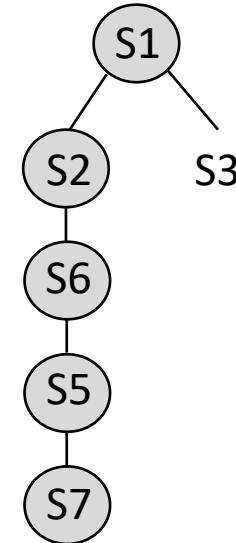
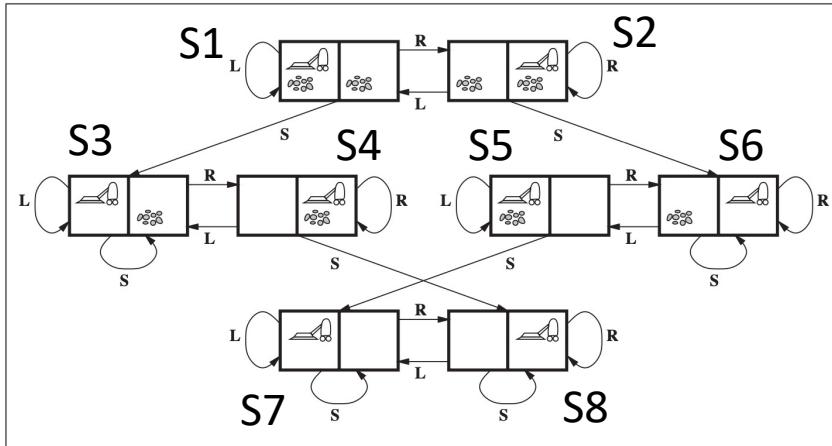
# Depth-First Search

- Depth-First search algorithm:
  - **Expand** the deepest node in the frontier
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - **Stop** when a goal node is visited



# Depth-First Search

- Depth-First search algorithm:
  - **Expand** the deepest node in the frontier
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - **Stop** when a goal node is visited



# Depth-First Search

- Solution:

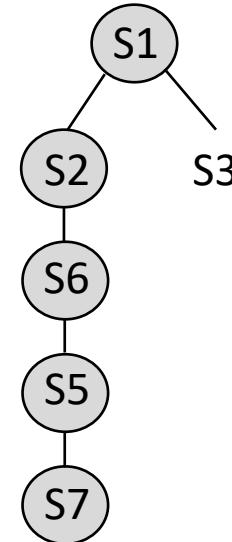
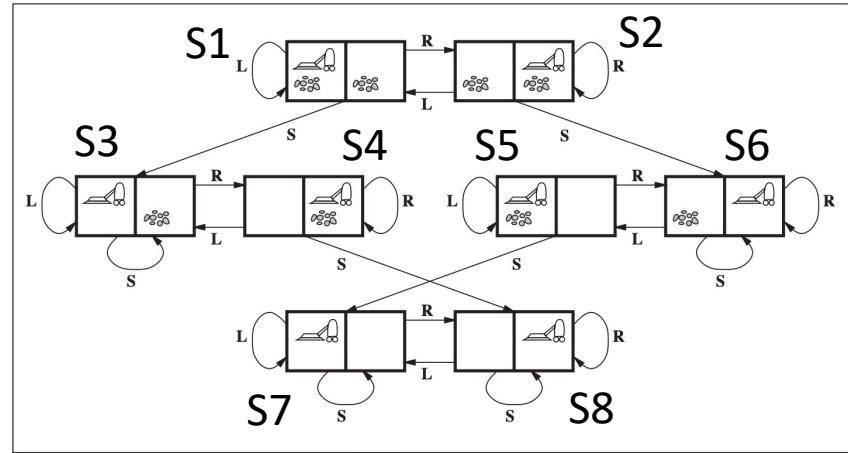
R, S, L, S

- Cost of the solution:

$$1 + 1 + 1 + 1 = 4$$

- Order of nodes visited

S1, S2, S6, S5, S7



# DFS - Performance

Let us evaluate the performance of the depth-first search algorithm

- **Completeness:** DFS **is not complete** if the search space is infinite or if we do not check infinite loops; it **is complete** if the search space is finite
- **Optimality:** DFS **is not optimal** as it can expand a left subtree when the goal node is in the first level of the right subtree

# DFS - Performance

Let us evaluate the performance of the depth-first search algorithm

- **Completeness:** DFS **is not complete** if the search space is infinite or if we do not check infinite loops; it **is complete** if the search space is finite
- **Optimality:** DFS **is not optimal** as it can expand a left subtree when the goal node is in the first level of the right subtree
- **Time complexity:**  $O(b^m)$ , as it depends on the maximum length of the path in the search space (in general  $m$  can be much larger than  $d$ )

# DFS - Performance

Let us evaluate the performance of the depth-first search algorithm

- **Completeness:** DFS **is not complete** if the search space is infinite or if we do not check infinite loops; it **is complete** if the search space is finite
- **Optimality:** DFS **is not optimal** as it can expand a left subtree when the goal node is in the first level of the right subtree
- **Time complexity:**  $O(b^m)$ , as it depends on the maximum length of the path in the search space (in general  $m$  can be much larger than  $d$ )
- **Space complexity:**  $O(b^m)$ , as we store all the nodes from each path from the root node to the leaf node

# Summary

- Depth-First Search is a search algorithm that expands the nodes in the frontier starting from the deepest, similar to a stack (LIFO)
- This algorithm is complete (for finite search space), but not optimal; also it has high time complexity and space complexity  $O(b^m)$

# Overview

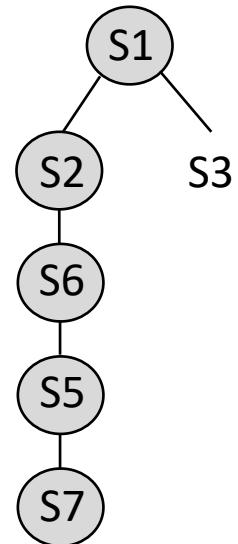
- Asymptotic Analysis
- Search Problem Formulation
- Breadth-First Search
- Depth-First Search
- **Variations of Depth-First Search**

# Depth-First Search - Variations

- Depth-First Search comes with several issues
  - Not optimal
  - High time complexity
  - High space complexity
- DFS with less memory usage (saving space complexity)
- Depth-Limited Search

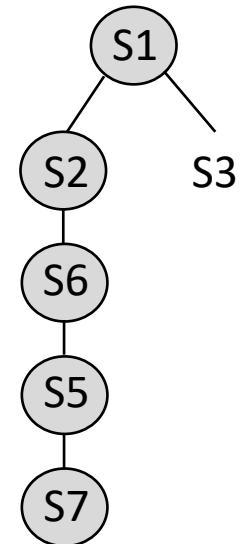
# Depth-First Search – Less Memory Usage

- Imagine we have a tree similar the one in the example
- Now, S7 is not a goal node and it has no children



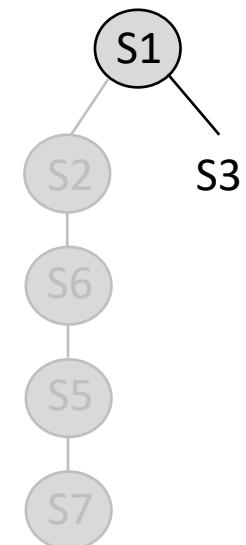
# Depth-First Search – Less Memory Usage

- Imagine we have a tree similar the one in the example
- Now, S7 is not a goal node and it has no children
- The next step of the algorithm would be to expand S3



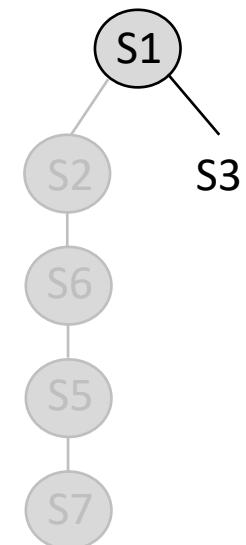
# Depth-First Search – Less Memory Usage

- Imagine we have a tree similar the one in the example
- Now, S7 is not a goal node and it has no children
- The next step of the algorithm would be to expand S3
- Since we explored all the left subtree, we can remove it from memory



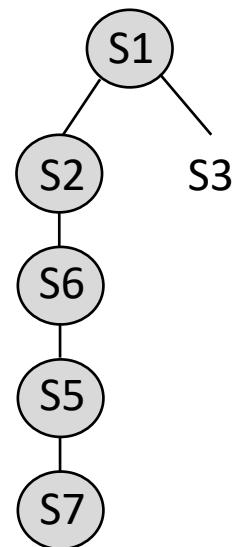
# Depth-First Search – Less Memory Usage

- This would reduce the space complexity to  $O(bm)$
- We need to store a single path along with the siblings for each node on the path
- Recall that  $b$  is the branching factor and  $m$  is the maximum depth of the search tree



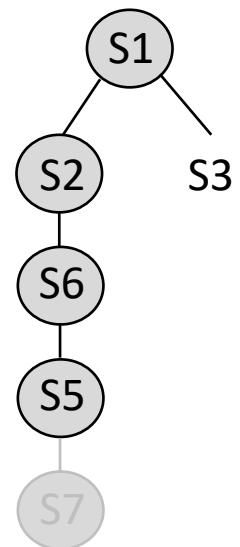
# Depth-Limited Search

- The issue related to depth-first search in infinite state spaces can be mitigated by providing a depth limit  $\ell$
- This approach is called **depth-limited search**



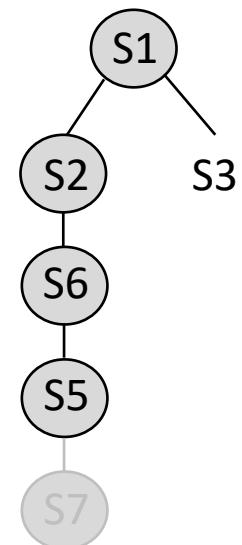
# Depth-Limited Search

- The issue related to depth-first search in infinite state spaces can be mitigated by providing a depth limit  $\ell$
- This approach is called **depth-limited search**
- For  $\ell = 3$ , we would have



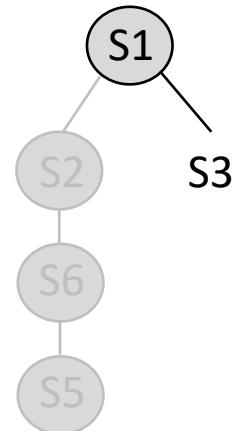
# Depth-Limited Search

- This adds an additional source of incompleteness if we choose  $\ell < d$ , namely the shallowest goal is beyond the depth limit
- This approach is nonoptimal also in the case  $\ell > d$
- Time complexity is  $O(b^\ell)$



# Depth-Limited Search – Less Memory Usage

- As before, we can remove the explored paths from memory after we have reached the depth limit  $\ell$
- Space complexity is  $O(b\ell)$



# Comparing Uninformed Search Strategies

Criterion / Algorithm	Breadth-First	Depth-First	Depth-First (less memory)	Depth-Limited (less memory)
<b>Completeness</b>	Yes*	Yes***	Yes***	Yes if $\ell \geq d$
<b>Optimality</b>	Yes**	No	No	No
<b>Time</b>	$O(b^d)$	$O(b^m)$	$O(b^m)$	$O(b^\ell)$
<b>Space</b>	$O(b^d)$	$O(b^m)$	$O(bm)$	$O(b\ell)$

\* If  $b$  is finite

\*\* If the path cost is a nondecreasing function of the depth of the node (e.g., all actions have the same cost)

\*\*\* If the search space is finite (also, loopy paths are removed)

# Summary

- Depth-First Search can be improved in terms of its time and space complexity through some modifications
- Depth-First Search with less memory usage only keeps in memory the current path and the siblings of the nodes
- Depth-Limited Search is another variation, where a depth limit is specified; this adds an additional source of incompleteness

# Aims of the Session

You should now be able to:

- Describe asymptotic analysis and why it is important
- Explain the steps to formulate a search problem
- Apply and compare the performance of Breadth-First Search, Depth-First Search and its variations



UNIVERSITY OF  
BIRMINGHAM

# AI1/AI&ML - Informed Search

Dr Leonardo Stella



# Aims of the Session

This session aims to help you:

- Describe the difference between uninformed and informed search
- Understand the concept of a heuristic function in informed search
- Analyse the performance of A\* and apply the algorithm to solve search problems

# Overview

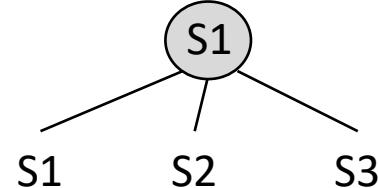
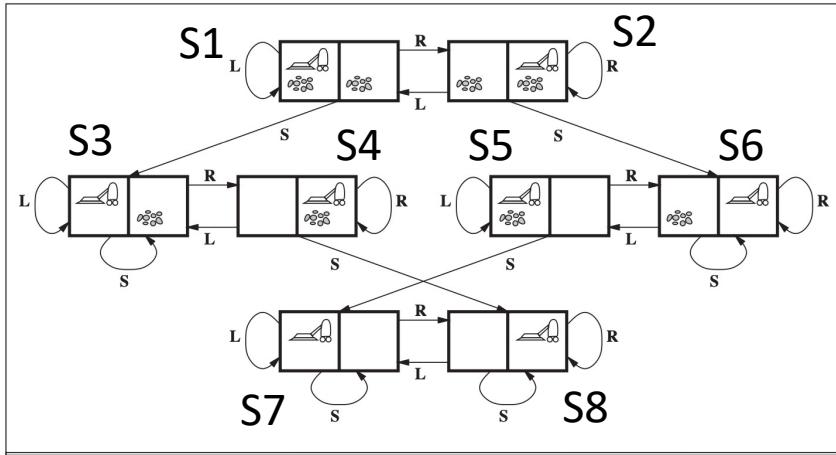
- **Recap – Uninformed Search**
- Informed Search
- A\* Search

# Searching for Solutions

- A solution is an action sequence from an initial state to a goal state
- Possible action sequences form a **search tree** with initial state at the root; actions are the branches and nodes correspond to the state space
- The idea is to expand the current state by applying each possible action: this generates a new set of states

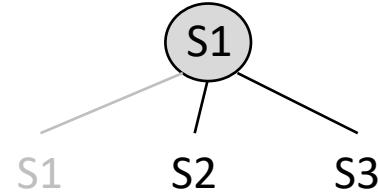
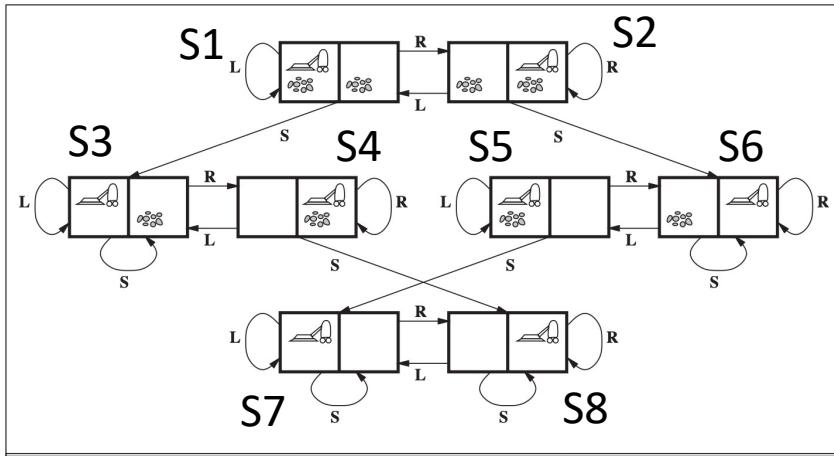
# Searching for Solutions

- Let us consider the example from before
- If  $S_1$  is the initial state and  $\{S_7, S_8\}$  is the set of goal states, the corresponding search tree after expanding the initial state is:



# Searching for Solutions

- Each of the three nodes resulting from the first expansion is a **leaf node**
- The set of all leaf nodes available for expansion at any given time is called the **frontier** (also sometimes called the **open list**)
- The path from S1 to S1 is a **loopy path** and in general is not considered

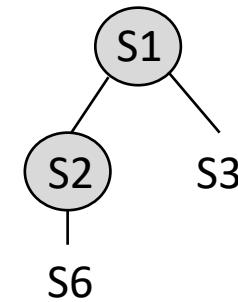
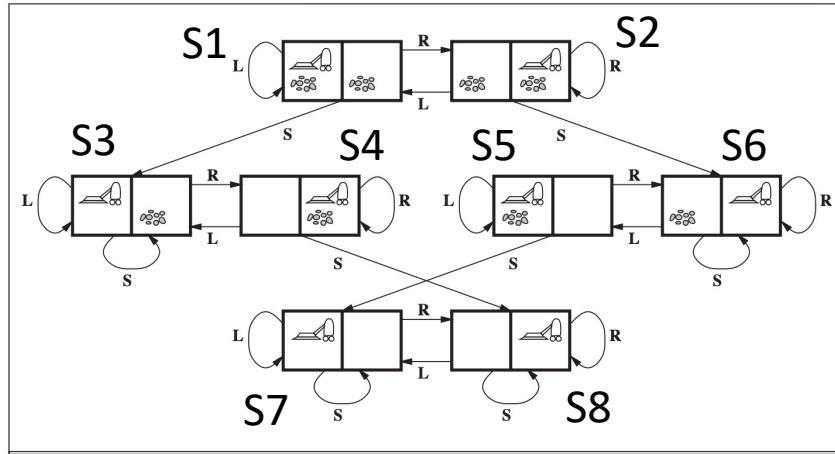


# Uninformed Search Strategies

- **Uninformed search** (also called **blind search**) means that the strategies have no additional information about states beyond that provided in the problem definition
- Uninformed search strategies can only generate successors and distinguish a goal state from a non-goal state
- The key difference between two uninformed search strategies is the **order** in which nodes are expanded

# Breadth-First Search vs Depth-First Search

- BFS would expand the shallowest node, namely S3
- DFS would expend the deepest node, namely S6



# Overview

- Recap – Uninformed Search
- **Informed Search**
- A\* Search

# Informed Search Strategies

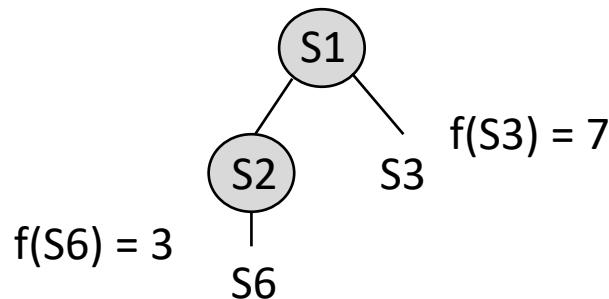
- **Informed search** strategies use problem-specific knowledge beyond the definition of the problem itself
- Informed search strategies can find solutions more efficiently compared to uninformed search

# Informed Search Strategies

- The general approach, called **best-first search**, is to determine which node to expand based on an **evaluation function**

$$f(n): \text{node} \rightarrow \text{cost estimate}$$

- This function acts as a cost estimate: the node with the lowest cost is the one that is expanded next

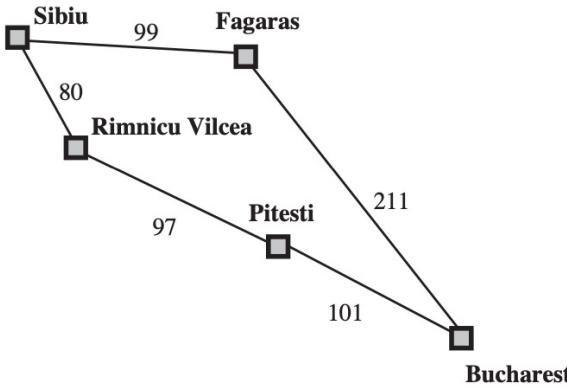


# Informed Search Strategies - Heuristics

- The evaluation function  $f(n)$  for most best-first algorithms includes a **heuristic function** as a component:  
$$h(n) = \text{estimated cost of the cheapest path from node } n \text{ to a goal node}$$
- Heuristic functions are the most common form in which new knowledge is given to the search algorithm. If  $n$  is a goal node, then  $h(n) = 0$
- A heuristic can be a rule of thumb, common knowledge; it is quick to compute, but not guaranteed to work (nor to yield optimal solutions)

# Informed Search Strategies - Heuristics

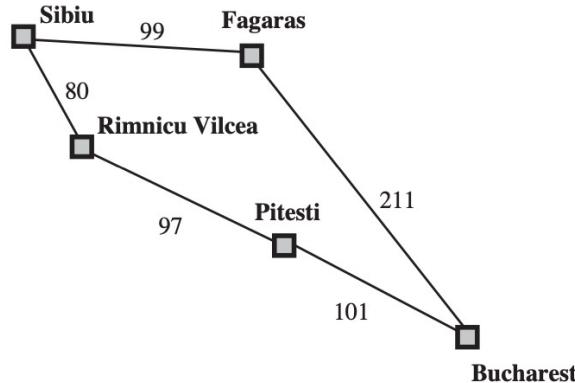
- Consider the problem to find the shortest path to Bucharest in Romania



- We can use the straight-line distance heuristic, denoted by  $h_{SLD}$
- This is a useful heuristic as it is correlated with actual road distances

# Informed Search Strategies - Heuristics

- Consider the problem to find the shortest path to Bucharest in Romania

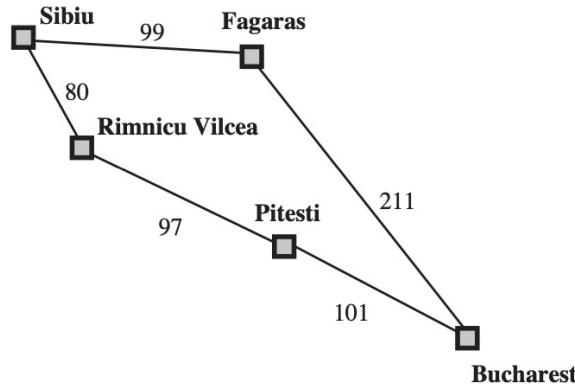


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

- The straight-line distances  $h_{SLD}$  are shown in the table above
- For example, the SLD from Sibiu would be 253

# Informed Search Strategies - Heuristics

- Consider the problem to find the shortest path to Bucharest in Romania

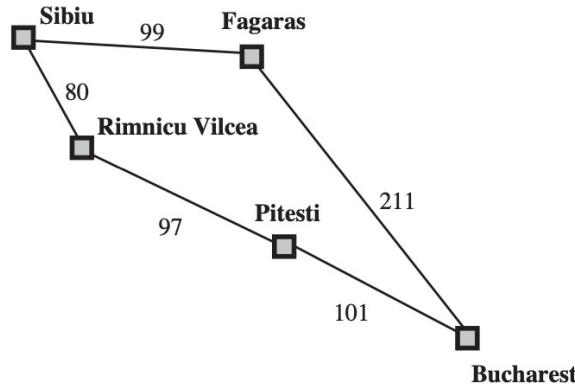


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

- If we use  $f(n) = h_{SLD}(n)$ , then from Sibiu we expand Fagaras
- This is because Fagaras has SLD 176, while Rimnicu Vilcea 193

# Informed Search Strategies - Heuristics

- Consider the problem to find the shortest path to Bucharest in Romania



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

- When  $f(n) = h(n)$ , we call this strategy **Greedy Best-First Search**

# Overview

- Recap – Uninformed Search
- Informed Search
- A\* Search

# A\* Search

- The most widely known informed search strategy is **A\***
- This search strategy evaluates nodes using the following cost function

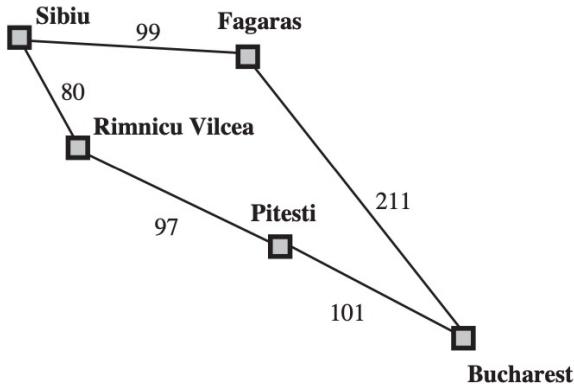
$$f(n) = g(n) + h(n)$$

where  $g(n)$  is the cost to reach the node and  $h(n)$  is the heuristic from the node to the goal

- This is equivalent to *the cost of the cheapest solution through node n*

# A\* Search - Example

- Consider the problem to find the shortest path to Bucharest in Romania



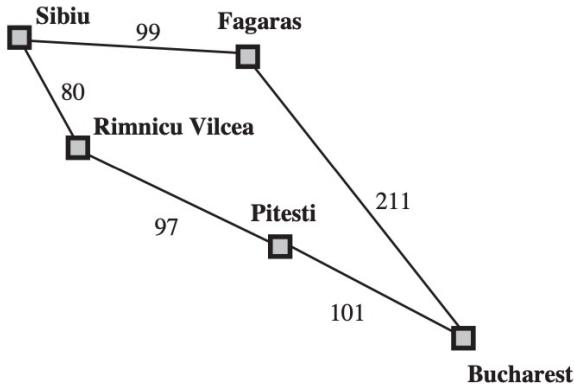
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

- Let us consider Sibiu as the initial state. Calculate  $f(n)$  to choose which node to expand, starting with Fagaras

$$f(Fagaras) = g(Fagaras) + h(Fagaras)$$

# A\* Search - Example

- Consider the problem to find the shortest path to Bucharest in Romania



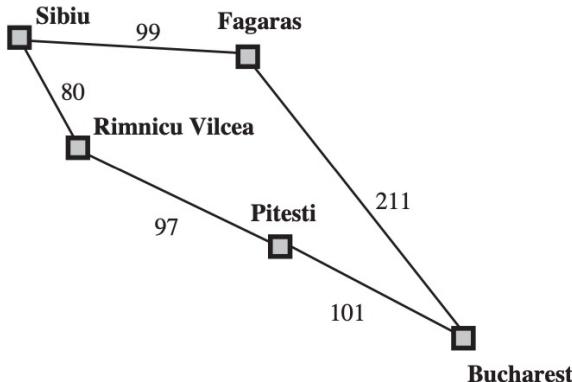
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

- Let us consider Sibiu as the initial state. Calculate  $f(n)$  to choose which node to expand, starting with Fagaras

$$f(Fagaras) = 99 + 176 = 275$$

# A\* Search - Example

- Consider the problem to find the shortest path to Bucharest in Romania



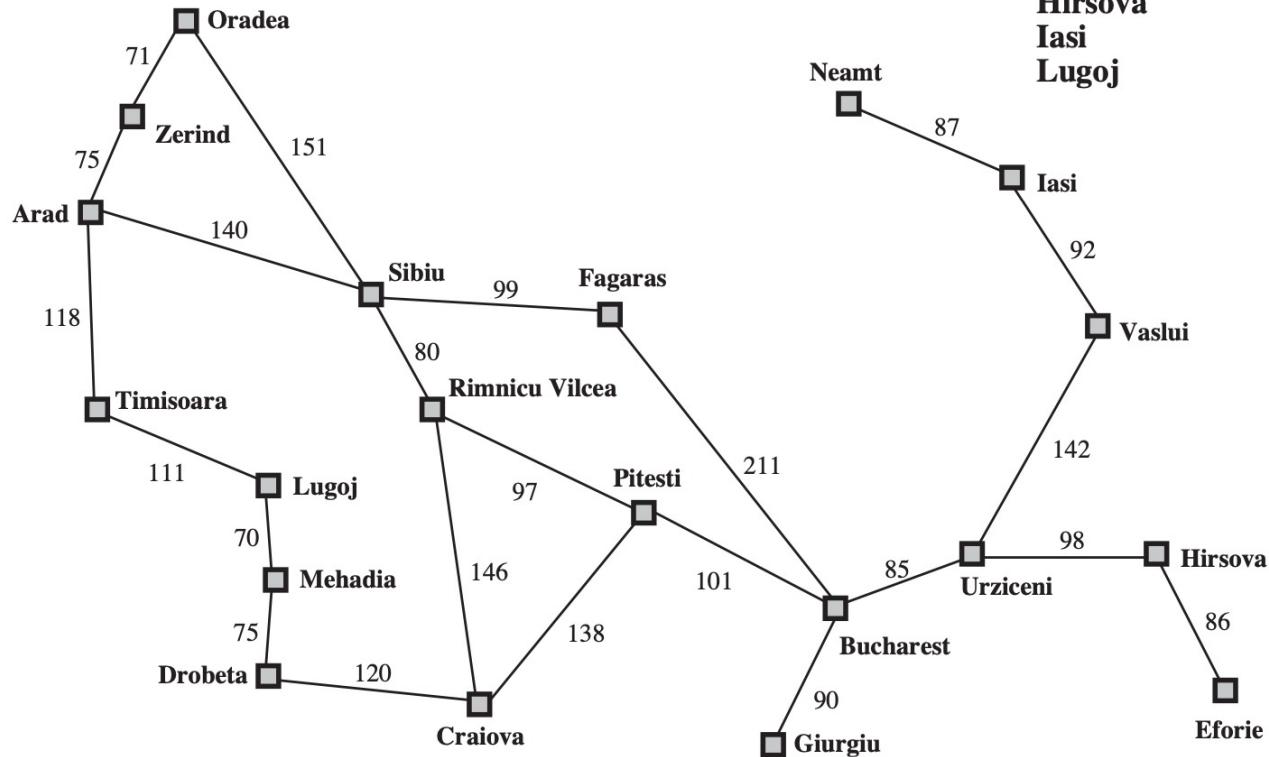
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

- Repeat the calculation for all the children, i.e., for Rimnicu Vilcea  
 $f(Rimnicu\ Vilcea) = 80 + 193 = 273$

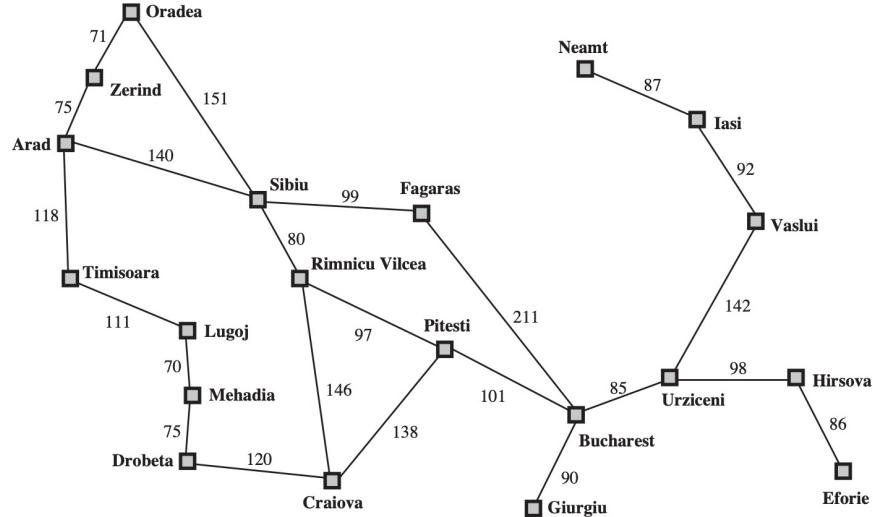
# A\* Search - Algorithm

- A\* search algorithm:
  - **Expand** the node in the frontier with smallest cost  $f(n) = g(n) + h(n)$
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - If the state of a given child is in the frontier
    - If the frontier node has a larger  $g(n)$ , place the child into the frontier and remove the node with larger  $g(n)$  from the frontier
  - **Stop** when a goal node is visited

# A\* Search



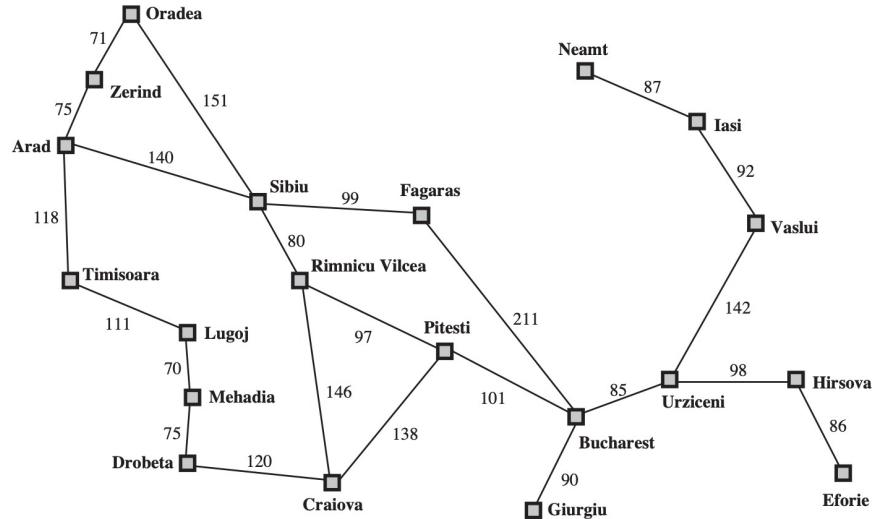
# A\* Search



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

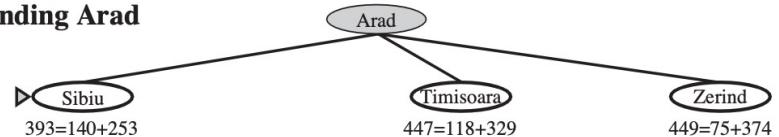
Arad  
366=0+366

# A\* Search

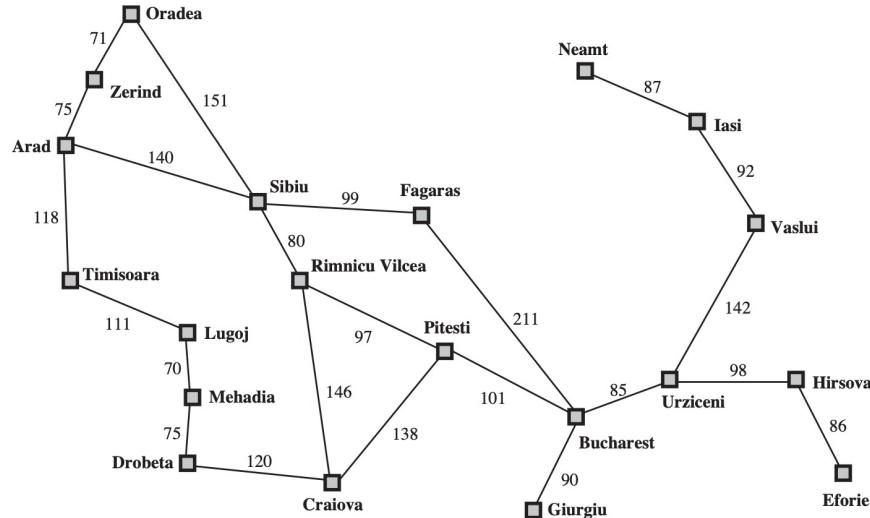


<b>Arad</b>	366	<b>Mehadia</b>	241
<b>Bucharest</b>	0	<b>Neamt</b>	234
<b>Craiova</b>	160	<b>Oradea</b>	380
<b>Drobeta</b>	242	<b>Pitesti</b>	100
<b>Eforie</b>	161	<b>Rimnicu Vilcea</b>	193
<b>Fagaras</b>	176	<b>Sibiu</b>	253
<b>Giurgiu</b>	77	<b>Timisoara</b>	329
<b>Hirsova</b>	151	<b>Urziceni</b>	80
<b>Iasi</b>	226	<b>Vaslui</b>	199
<b>Lugoj</b>	244	<b>Zerind</b>	374

After expanding Arad

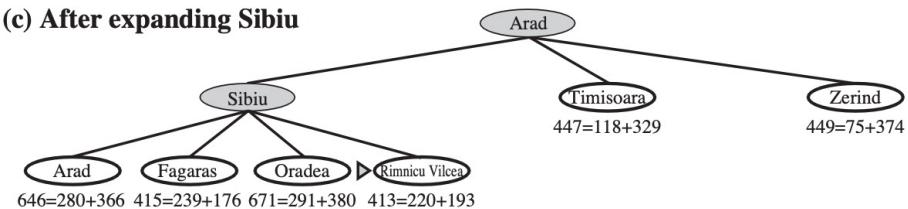


# A\* Search

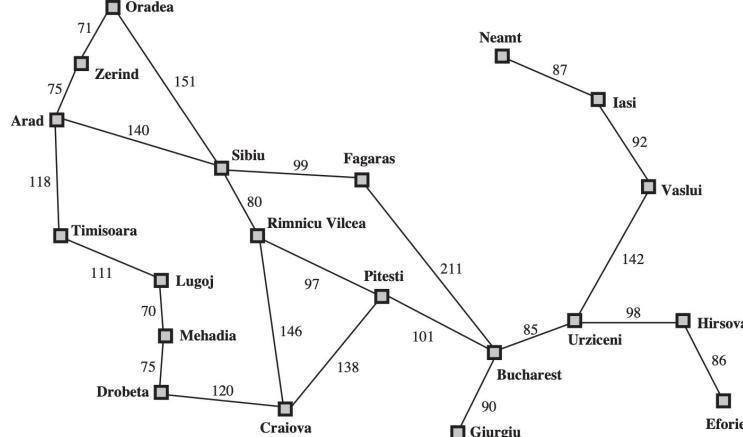


<b>Arad</b>	366	<b>Mehadia</b>	241
<b>Bucharest</b>	0	<b>Neamt</b>	234
<b>Craiova</b>	160	<b>Oradea</b>	380
<b>Drobeta</b>	242	<b>Pitesti</b>	100
<b>Eforie</b>	161	<b>Rimnicu Vilcea</b>	193
<b>Fagaras</b>	176	<b>Sibiu</b>	253
<b>Giurgiu</b>	77	<b>Timisoara</b>	329
<b>Hirsova</b>	151	<b>Urziceni</b>	80
<b>Iasi</b>	226	<b>Vaslui</b>	199
<b>Lugoj</b>	244	<b>Zerind</b>	374

(c) After expanding Sibiu

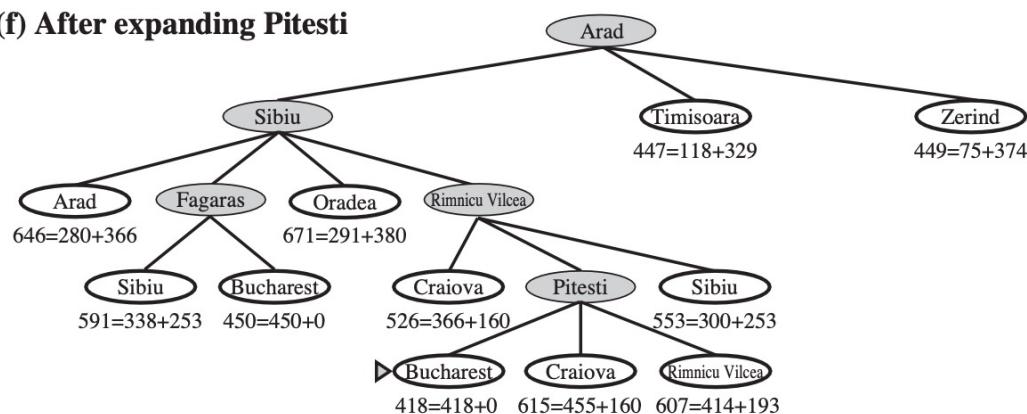


# A\* Search



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

(f) After expanding Pitesti



# A\* Search - Completeness and Optimality

- The A\* search is **complete** and **optimal** if  $h(n)$  is consistent

# A\* Search - Completeness and Optimality

- The A\* search is **complete** and **optimal** if  $h(n)$  is consistent
- A heuristic is said to be consistent (or monotone), if the estimate is always no greater than the estimated distance from any neighbouring vertex to the goal, plus the cost of reaching that neighbour

$$h(n) \leq \text{cost}(n, n') + h(n')$$

# A\* Search - Time and Space Complexity

- The number of states for the A\* search is **exponential** in the length of the solution, namely for constant step costs:  $O(b^{\epsilon d})$
- When  $h^*$  is the actual cost from root node to goal node,  $\epsilon = \frac{(h^* - h)}{h^*}$  is the relative error

# A\* Search - Time and Space Complexity

- The number of states for the A\* search is **exponential** in the length of the solution, namely for constant step costs:  $O(b^{\epsilon d})$
- When  $h^*$  is the actual cost from root node to goal node,  $\epsilon = \frac{(h^* - h)}{h^*}$  is the relative error
- Space is the main issue with A\*, as it keeps all generated nodes in memory, therefore A\* is not suitable for many large-scale problems

# A\* Search - Summary

Let us summarise the performance of the A\* search algorithm

- **Completeness:** if the heuristic  $h(n)$  is consistent, then the A\* algorithm **is complete**
- **Optimality:** if the heuristic  $h(n)$  is consistent, A\* **is optimal**

# A\* Search - Summary

Let us summarise the performance of the A\* search algorithm

- **Completeness:** if the heuristic  $h(n)$  is consistent, then the A\* algorithm **is complete**
- **Optimality:** if the heuristic  $h(n)$  is consistent, A\* **is optimal**
- **Time complexity:**  $O(b^{\epsilon d})$ , where  $\epsilon$  is the relative error of the heuristic

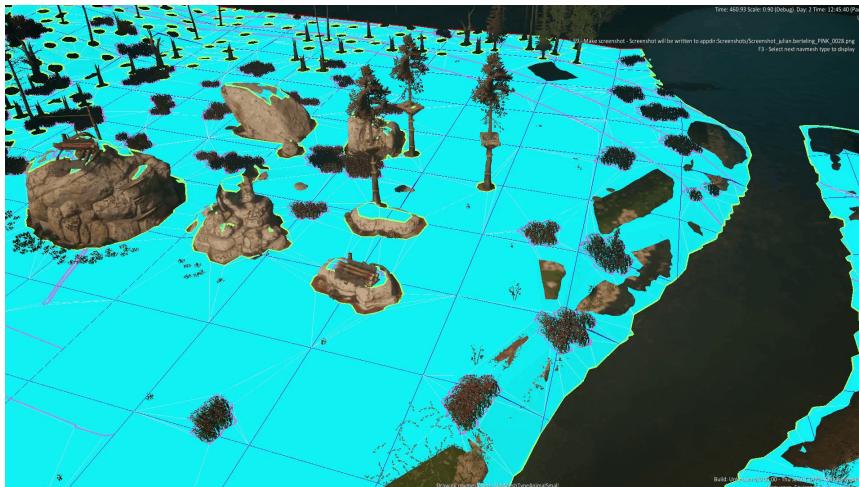
# A\* Search - Summary

Let us summarise the performance of the A\* search algorithm

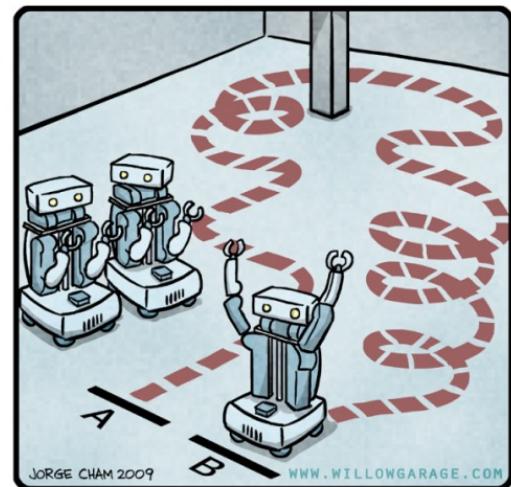
- **Completeness:** if the heuristic  $h(n)$  is consistent, then the A\* algorithm **is complete**
- **Optimality:** if the heuristic  $h(n)$  is consistent, A\* **is optimal**
- **Time complexity:**  $O(b^{\epsilon d})$ , where  $\epsilon$  is the relative error of the heuristic
- **Space complexity:**  $O(b^d)$ , since we keep in memory all expanded nodes and all nodes in the frontier

# A\* - Applications

- A\* has a large number of applications
- In practice, the most common ones are in games and in robotics



R.O.B.O.T. Comics



"HIS PATH-PLANNING MAY BE  
SUB-OPTIMAL, BUT IT'S GOT FLAIR."

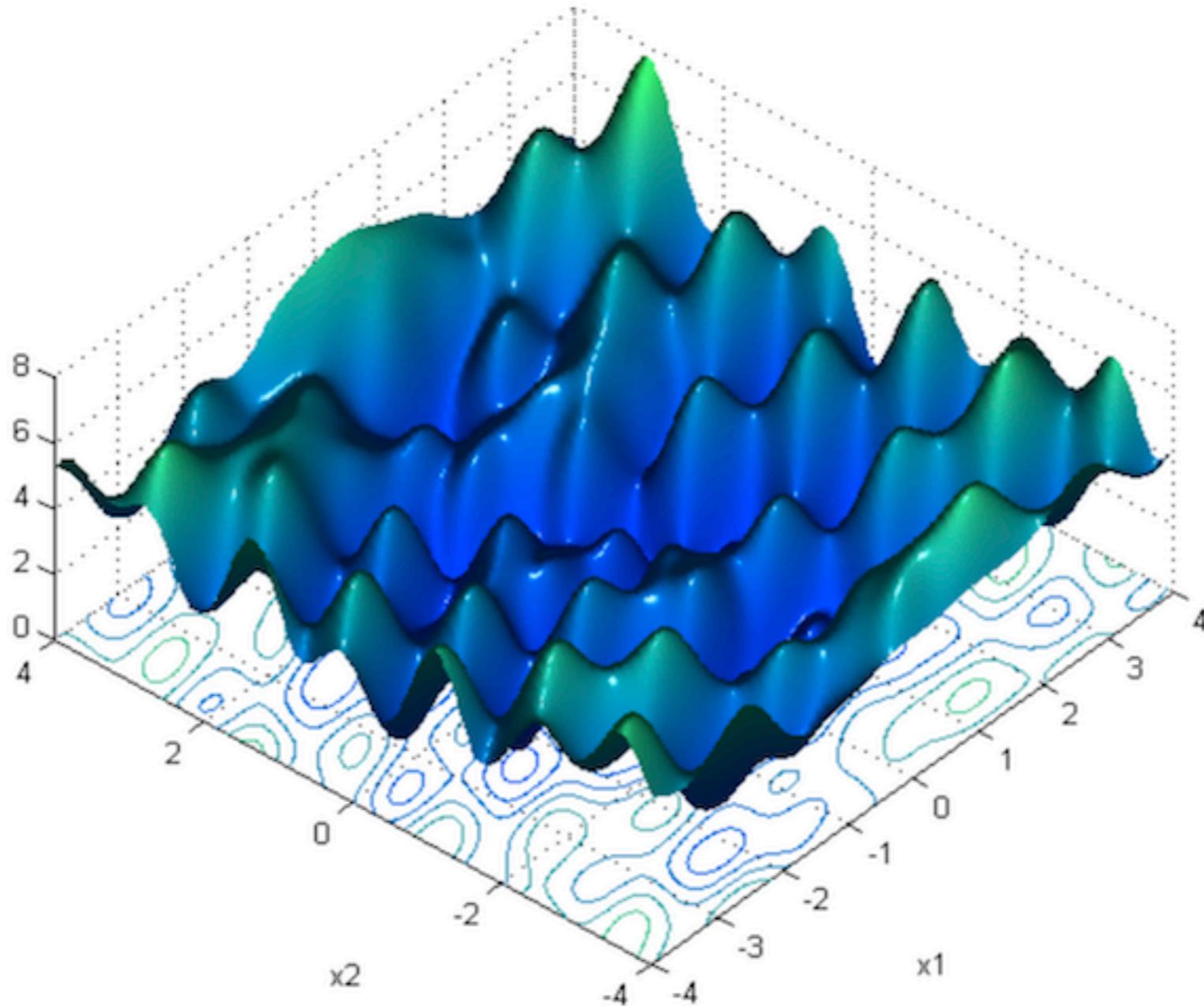
# Summary

- A\* is complete and optimal, given a consistent heuristic
- However, A\* has typically high time/space complexity, regardless of the heuristic chosen
- Heuristics have a considerable impact on the performance of informed search algorithms, and they can drastically reduce the time and space complexity in comparison to uninformed search algorithms

# Aims of the Session

You should now be able to:

- Describe the difference between uninformed and informed search
- Understand the concept of a heuristic function in informed search
- Analyse the performance of A\* and apply the algorithm to solve search problems



# Introduction to Optimisation

Leandro L. Minku

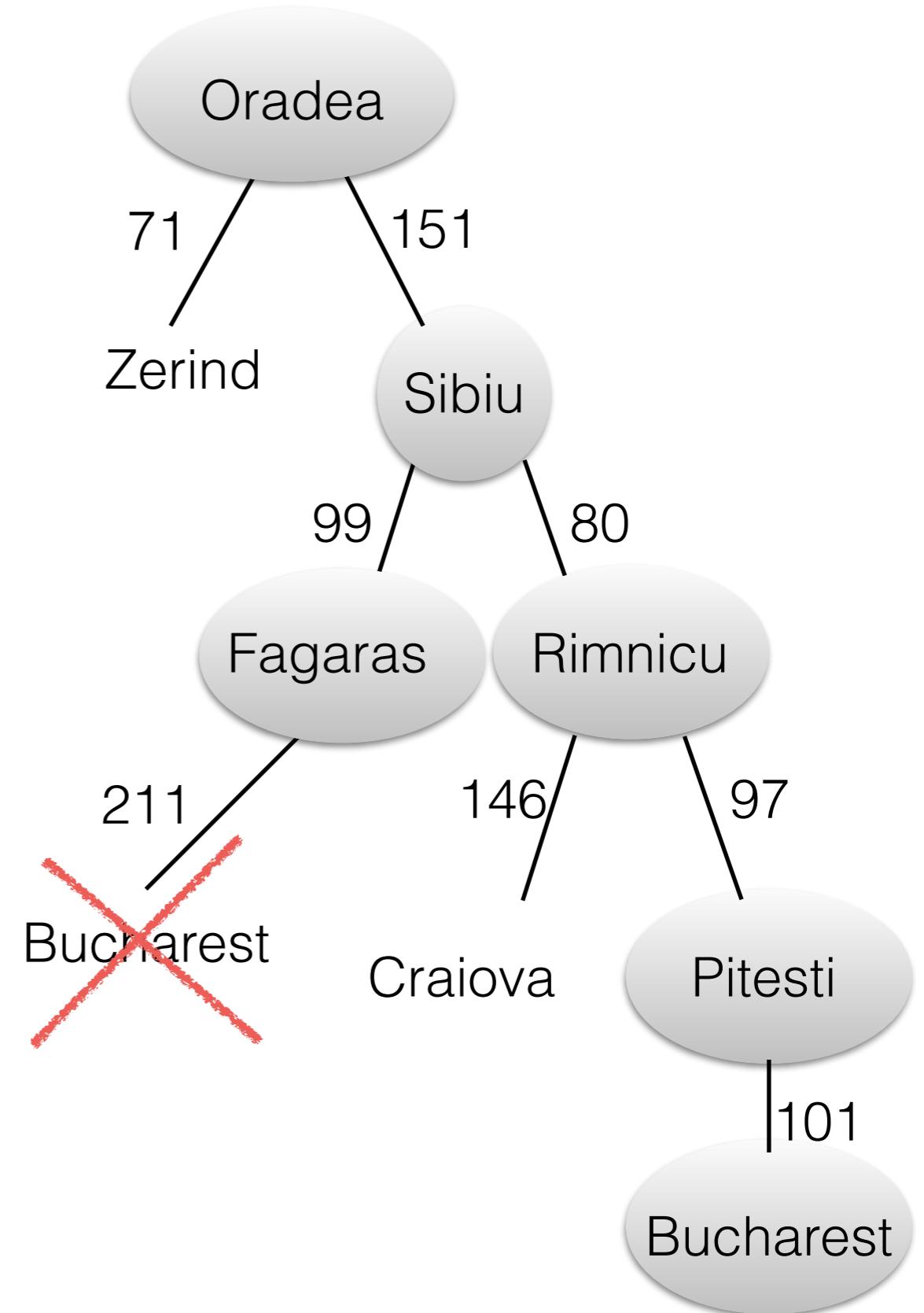
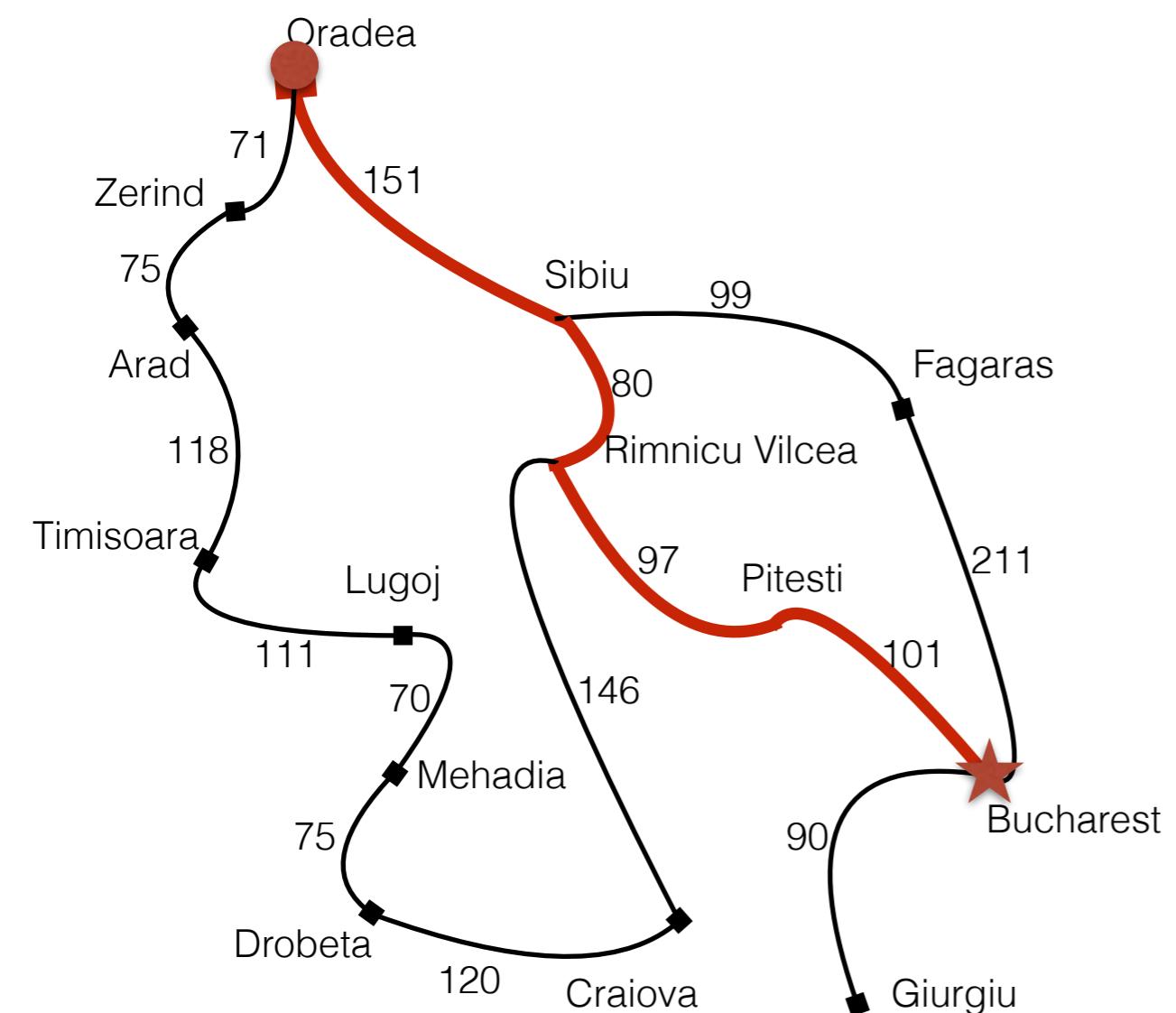
# Optimisation Problems

- Optimisation problems: to find a solution that minimises/maximises one or more pre-defined objective functions.
- Maximisation / minimisation problems.
- There may be some constraints that must or should be satisfied for a given solution to be feasible.

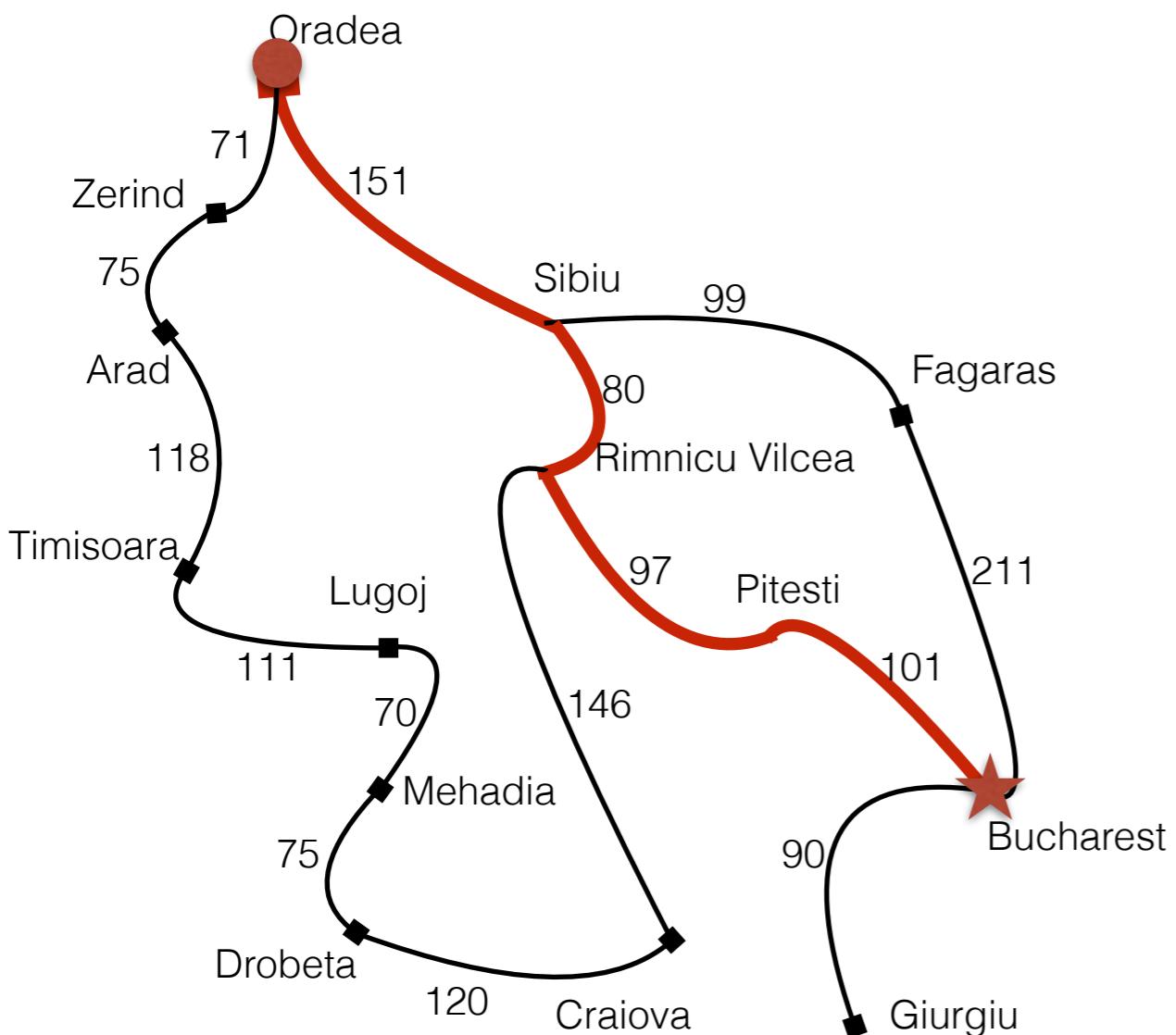
# Optimisation Algorithms from Artificial Intelligence

- Solutions do not correspond to paths built step by step from an **initial** to a **goal** state.
- Instead, the algorithms typically maintain **whole candidate** solutions from the beginning.
- **Candidate** solutions may be feasible or infeasible.

# Routing Problem



# Routing Problem



Example of infeasible candidate solution:  
[Oradea, Zerind, Arad, Timisoara]  
(does not reach the destination)

Example of infeasible candidate solution:  
[Oradea, Rimnicu, Mehadia, Bucharest]  
(moves to non-neighbouring cities)

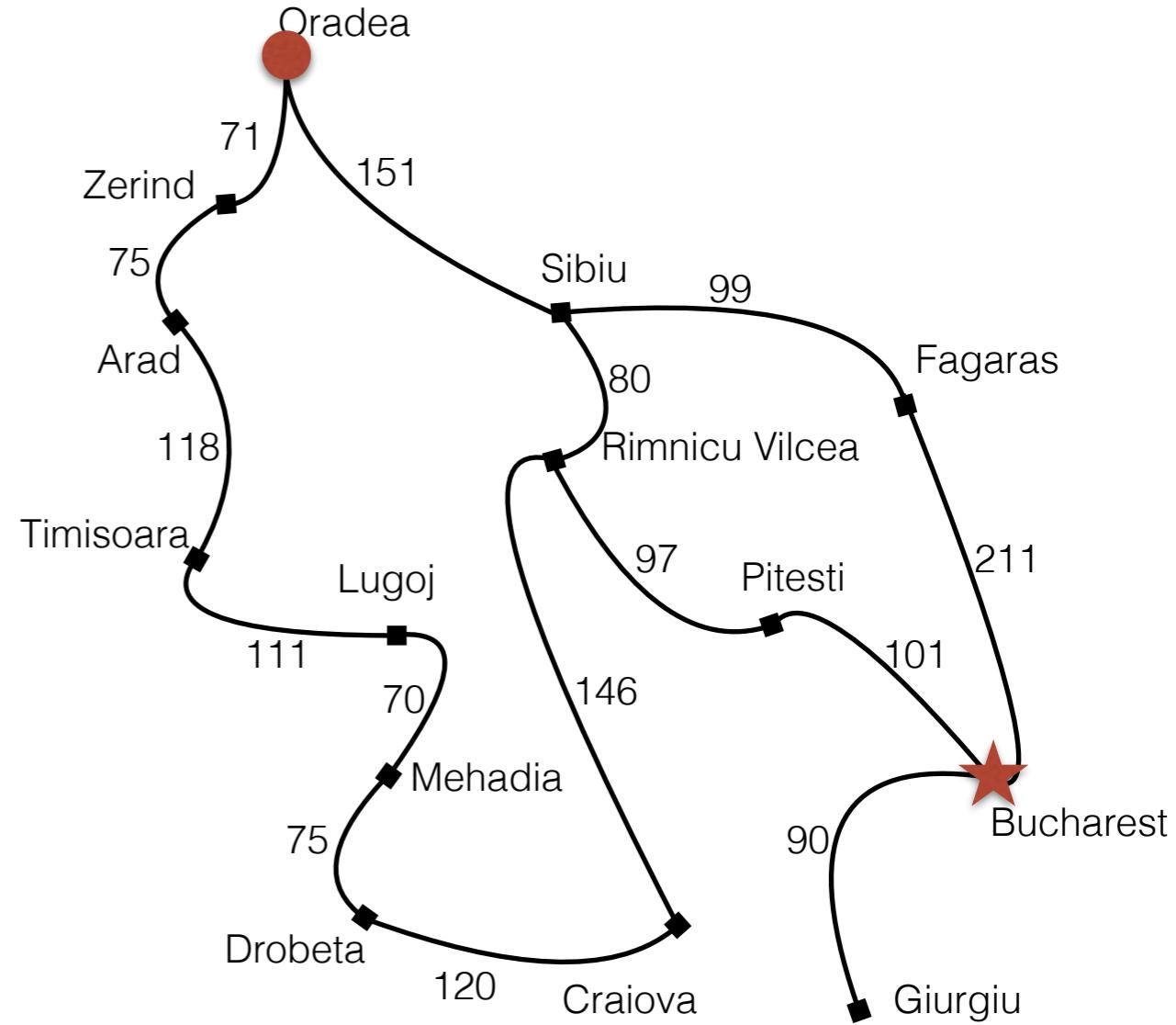
Example of feasible candidate solution:  
[Oradea, Sibiu, Fagaras, Bucharest]  
(non-optimal solution that takes the agent  
from the origin to the destination)

Optimal solution:  
[Oradea, Sibiu, Rimnicu, Pitesti, Bucharest]

# Examples of Optimisation Problems

- Routing problem:
  - Given a motorway map containing  $N$  cities.
  - The map shows the distance between connected cities.
  - We have a city of origin and a city of destination.

Problem: find a path from the origin to the destination that minimises the distance travelled, while ensuring that direct paths between non-neighbouring cities are not used.



# Search and Optimisation

- In search, we are interested in searching for a [goal state](#).
- In optimisation, we are interested in searching for an [optimal solution](#).
- As many search problems have a cost associated to actions, they can also be formulated as optimisation problems.
- Similarly, optimisation problems can frequently be formulated as search problems associated to a cost function.
- Many search algorithms will “search” for optimal solutions (see A\* as an example).
- Optimisation algorithms may also be used to solve search problems if they can be associated to an appropriate function to be optimised.

# Artificial Intelligence Optimisation Algorithms

- Advantages:
  - Usually more space efficient, frequently requiring the same amount of space from the beginning to the end of the optimisation process.
    - They do not maintain alternative paths to solutions.
    - Frequently able to find reasonable solutions for problems with large state spaces, for which the tree-based search algorithms are unsuitable.
  - Can potentially be more time efficient, depending on the algorithm.
  - Do not necessarily require problem-specific heuristics.
- Weaknesses:
  - Not guaranteed to retrieve the optimal solution in a reasonable amount of time.
  - Depending on the problem formulation and operators, not guaranteed to be complete either.
- Applicability:
  - Can be used for any problem that can be formulated as an optimisation problem.

# Examples of Optimisation Problems

- Bin packing problem:
  - Given bins with maximum volume  $V$ , which cannot be exceeded.
  - We have  $n$  items to pack, each with a volume  $v$ .
  - We must pack all items.

Problem: **find** an assignment of items to bins that **minimises** the number of bins used, **ensuring** that all items are **packed** and the volume of the bins is **not exceeded**.



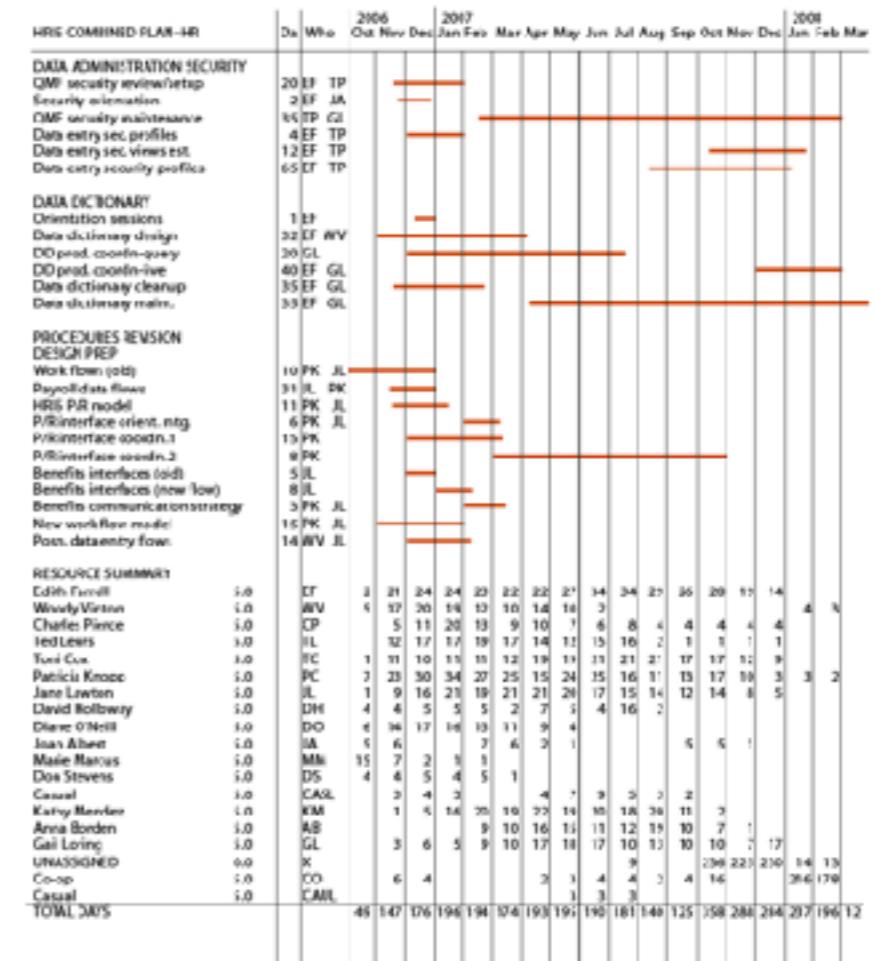
Photo from: <http://maritime-connector.com/images/container-ship-16-wiki-19057.jpg>



Photo from: <http://www.tscargo.ca/images/cargo1.jpg>

# Examples of Software Engineering Optimisation Problems

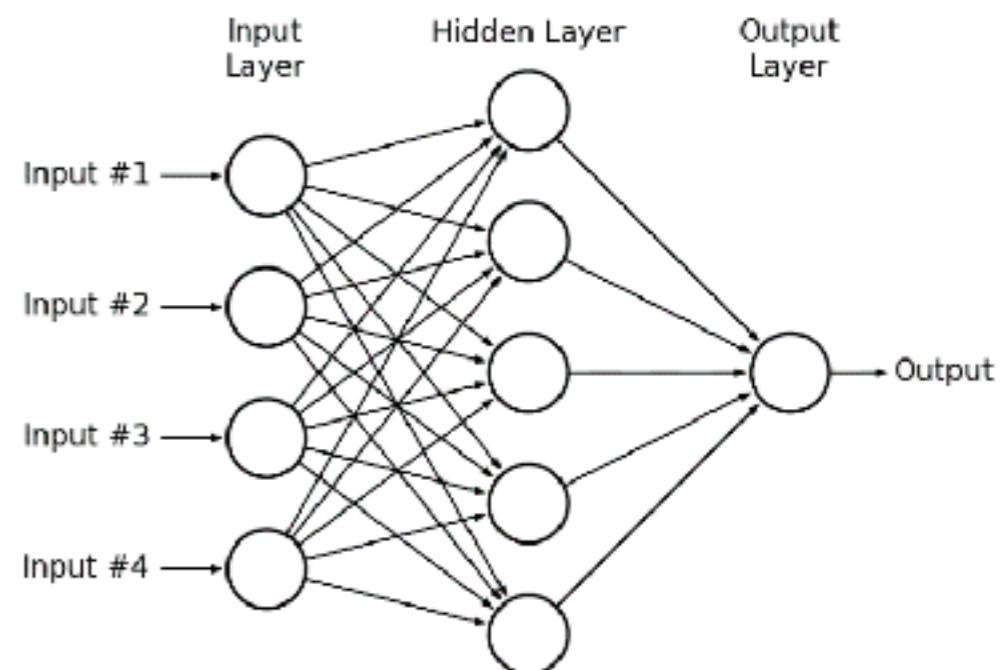
- Software Project Scheduling:
  - Given  $E$  employees and  $T$  tasks.
  - Each task requires an effort in hours and certain skills.
  - Each employee has a salary, a set of skills and can work a maximum number of hours.
  - Tasks have precedence relationships.



Problem: find an allocation of employees to tasks that minimises the cost and the duration of the software project, while ensuring that employees are only assigned to tasks for which they have the required skills, that they work only up to a maximum number of hours, and that the task precedences are respected.

# Examples of Optimisation Problems

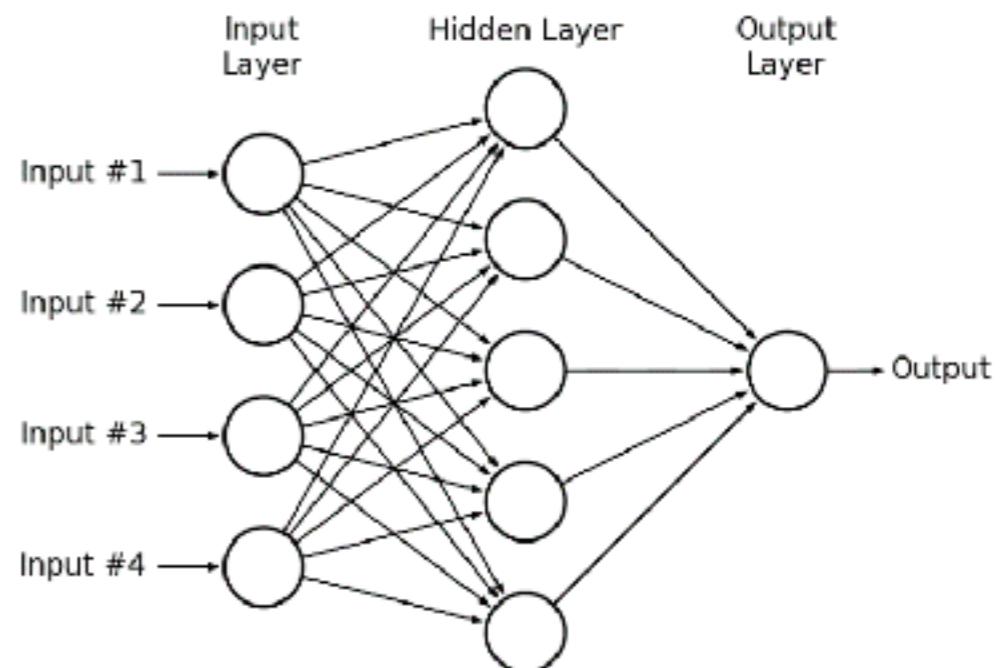
- Hyperparameter optimisation:
  - Consider the hyperparameters of a machine learning algorithm.
  - Some hyperparameters may be continuous, e.g., learning rate.
  - Some hyperparameters may be categorical or ordinal, e.g., activation function.



Problem: find the hyperparameter values that minimise the error on the validation set.

# Examples of Optimisation Problems

- Learning?
  - Some machine learning algorithms have models that take the form of functions of a pre-defined form.
  - These functions are described by parameters, e.g., the weights of a neural networks.



Problem: find the parameter values that minimise the loss calculated based on the training set.

# Learning vs Optimisation

- From an **algorithmic** perspective, learning can be seen as finding parameters that minimise a loss function.
- We can compute the loss based on the training set.

# Learning vs Optimisation

- From a **problem** perspective, the **goal** of machine learning is to create models able to **generalise** to unseen data.
  - In supervised learning, we want to minimise the expected loss, i.e., the loss considering all possible examples, including those that we have not observed yet.
  - We cannot calculate the loss based on unseen data during training time!
  - So, learning can be essentially seen as trying to optimise a function that cannot be computed.
  - Therefore, our algorithms may calculate the loss based on the training set, and design a loss function that includes, e.g., a regularisation term, in an attempt to generalise well to unseen data.

# Learning vs Optimisation

- From a **problem** perspective, optimisation usually really wants to minimise (or maximise) the value of a given (known) objective function.
- In that sense, learning and optimisation are different.
- However, there will be some optimisation problems where we can't compute the exact function to be optimised, causing the distinction between learning and optimisation to become more blurry.

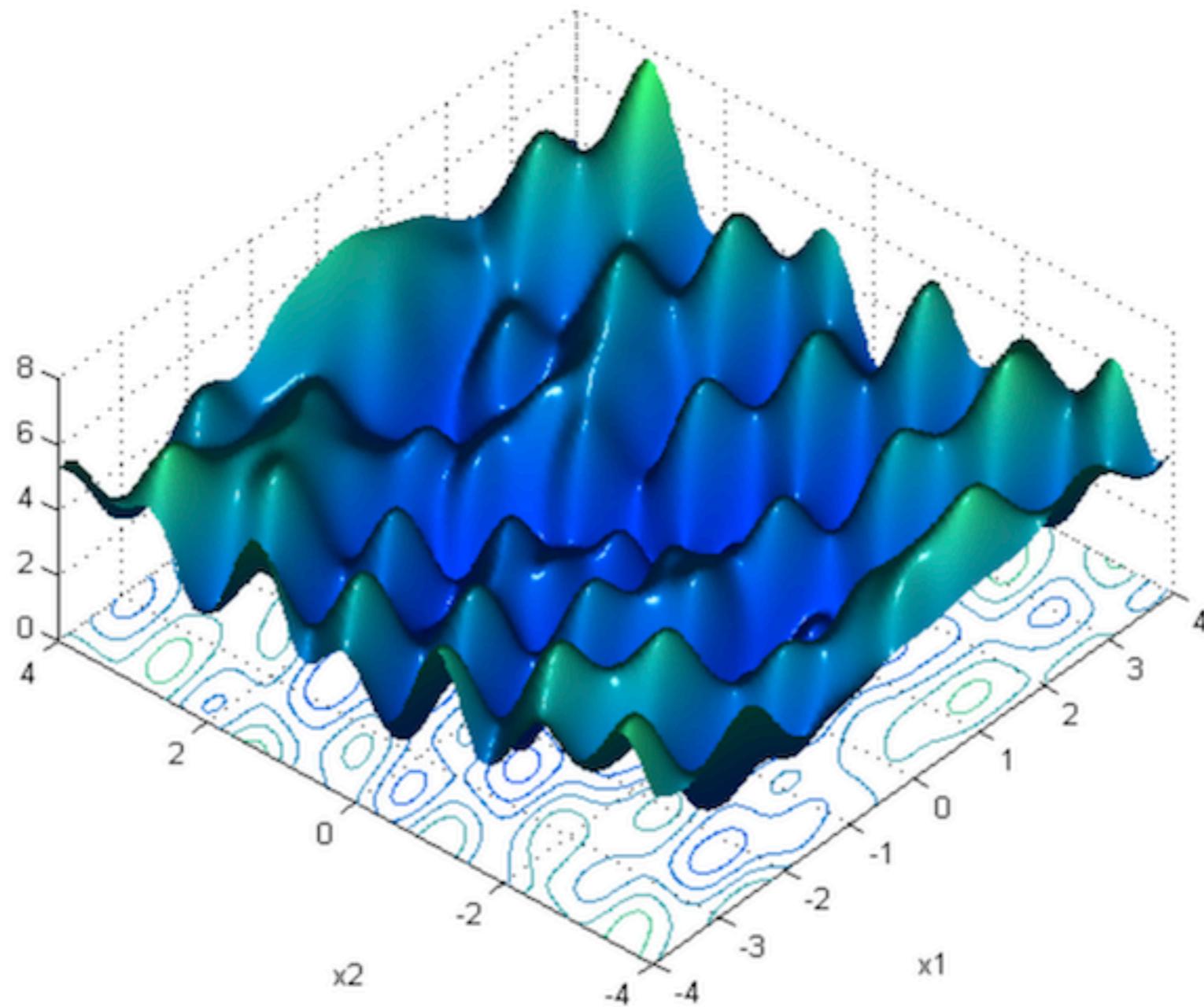
# Summary

- Optimisation problems are problems where we want to minimise (or maximise) one or more objective functions, possibly subject to certain constraints.
- Optimisation algorithms can often find good solutions in a reasonable amount of time, but are typically not guaranteed to find optimal solutions in a reasonable amount of time.

## Next

- How to formulate optimisation problems.

Wolfram Global Problem



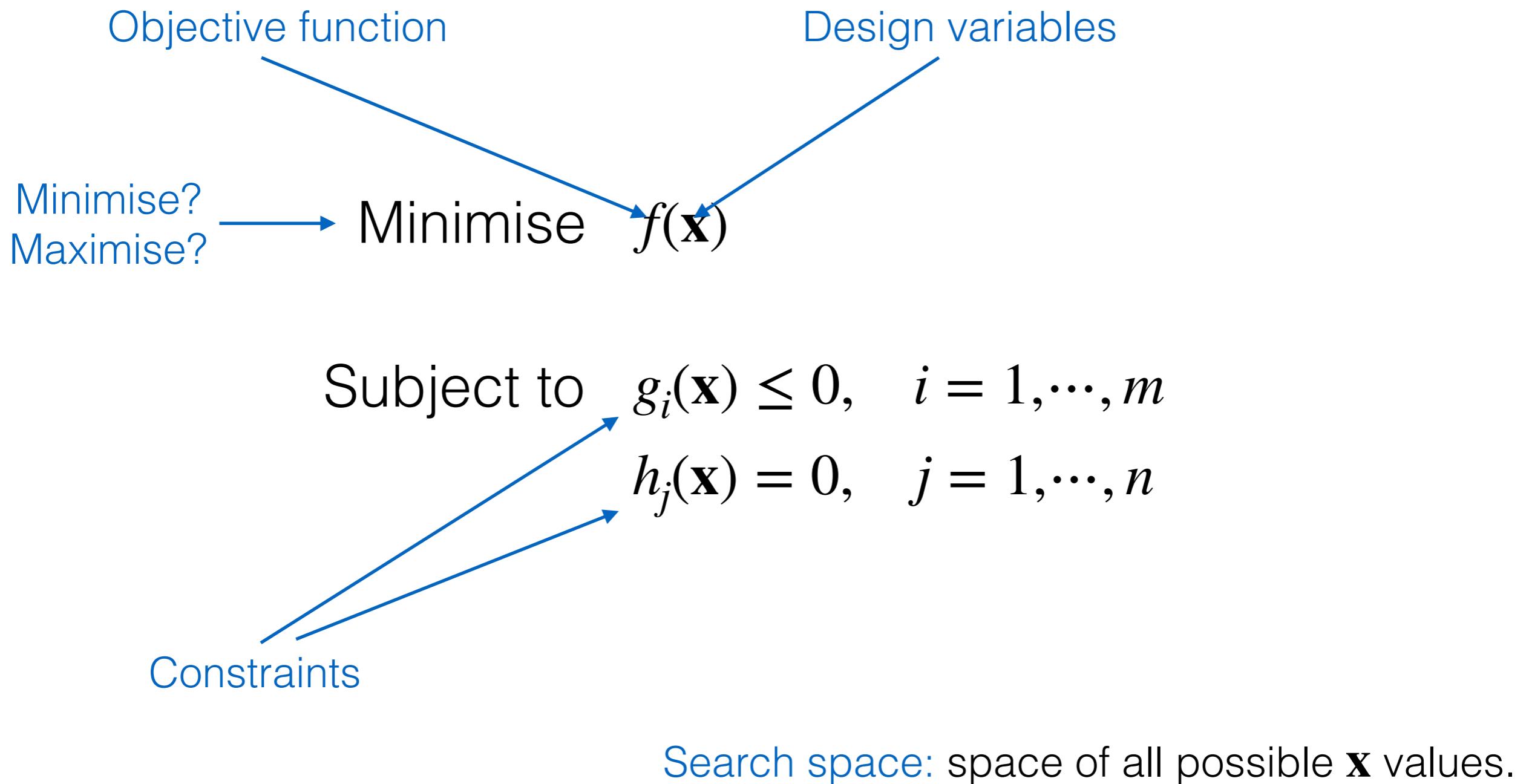
# Optimisation Problem Formulation

Leandro L. Minku

# Optimisation Problems

- Optimisation problems: to find a solution that minimises/maximises one or more pre-defined objective functions.
- Maximisation / minimisation problems.
- What constitutes a solution depends on the problem in hands.

# Optimisation Problems



# Multi-Objective Optimisation Problems

Minimise  $f_k(\mathbf{x}), k = 1 \dots, p$

Subject to  $g_i(\mathbf{x}) \leq 0, i = 1, \dots, m$

$h_j(\mathbf{x}) = 0, j = 1, \dots, n$

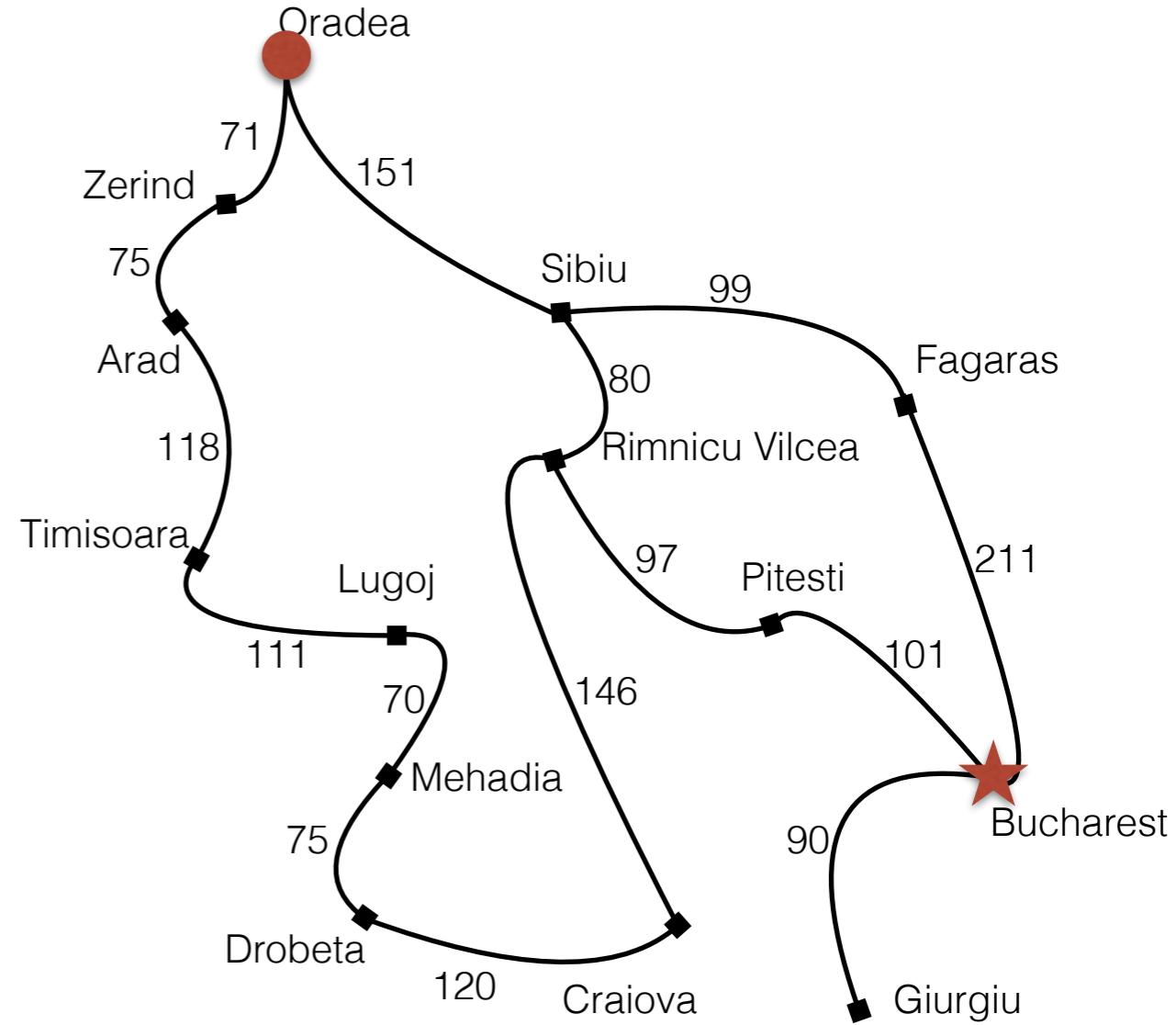
# Formulating Optimisation Problems

- Design variables represent a candidate solution.
  - Design variables define the search space of candidate solutions.
- Objective function defines the quality (or cost) of a solution.
  - Function to be optimised (maximised or minimised).
- [Optional] Solutions must satisfy certain constraints, which define solution feasibility.
  - Candidate solutions may be feasible or infeasible.

# Examples of Optimisation Problems

- Routing problem:
  - Given a motorway map containing  $N$  cities.
  - The map shows the distance between connected cities.
  - We have a city of origin and a city of destination.

Problem: find a path from the origin to the destination that minimises the distance travelled, while ensuring that direct paths between non-neighbouring cities are not used.



# Routing Problem: Formulation as an Optimisation Problem

- Design variables represent a candidate solution.
  - Sequence  $\mathbf{x}$  containing the cities to be visited, where  $x_i \in C$ ,  $C$  is the set of available cities, and  $\mathbf{x}$  can be of any size.
  - The search space consists of all possible sequences of cities.

<u>Oradea</u>	<u>Sibiu</u>	<u>Fagaras</u>	<u>Bucharest</u>
$x_1$	$x_2$	$x_3$	$x_4$

# Routing Problem: Formulation as an Optimisation Problem

- Objective function defines the quality (or cost) of a solution.  
Minimise the sum of the distances between consecutive cities in  $\mathbf{x}$ .

Oradea      Sibiu      Fagaras      Bucharest  
 $x_1$              $x_2$              $x_3$              $x_4$

# Routing Problem: Formulation as an Optimisation Problem

- [Optional] Solutions must satisfy certain constraints, which define solution feasibility.
  - (Inexistent) direct paths between non-neighbouring cities must not be used (**explicit constraint**).
  - We must start at the city of origin and end at the city of destination (**explicit constraint**).
  - Only cities in  $C$  can be used (**implicit constraint**).

Oradea	Sibiu	Fagaras	Bucharest
$x_1$	$x_2$	$x_3$	$x_4$

# Routing Problem: Formulation as an Optimisation Problem

- Design variables represent a candidate solution.
  - Sequence  $\mathbf{x}$  containing the cities to be visited, where  $x_i \in C$ ,  $C$  is the set of available cities, and  $\mathbf{x}$  can be of any size.
  - The search space consists of all possible sequences of cities.
- Objective function defines the quality (or cost) of a solution.

Minimise the sum of the distances between consecutive cities in  $\mathbf{x}$ .
- [Optional] Solutions must satisfy certain constraints, which define solution feasibility.
  - (Inexistent) direct paths between non-neighbouring cities must not be used (**explicit constraint**).
  - We must start at the city of origin and end at the city of destination (**explicit constraint**).
  - Only cities in  $C$  can be used (**implicit constraint**).

# Routing Problem: Making it More Formal

- Design variables represent a candidate solution.
  - Sequence  $\mathbf{x}$  containing the cities to be visited, where  $x_i \in \{1, \dots, N\}$  and  $\mathbf{x}$  can be of any size.
  - The search space consists of all possible sequences of cities.

<u>Oradea</u>	<u>Sibiu</u>	<u>Fagaras</u>	<u>Bucharest</u>
$x_1$	$x_2$	$x_3$	$x_4$
1	10	2	14
$x_1$	$x_2$	$x_3$	$x_4$

# Routing Problem: Making it More Formal

- Design variables represent a candidate solution.
  - Sequence  $\mathbf{x}$  containing the cities to be visited, where  $x_i \in \{1, \dots, N\}$  and  $\mathbf{x}$  can be of any size.
  - The search space consists of all possible sequences of cities.
- Objective function defines the quality (or cost) of a solution.

$$\text{Minimise } f(\mathbf{x}) = \sum_{i=1}^{\text{size}(\mathbf{x})-1} D_{x_i, x_{i+1}}$$

where  $D$  is a matrix of distances, with each position  $D_{i,j}$  containing:

- the distance in km to travel directly between city  $i$  and  $j$ , or
- -1 if such direct path does not exist.

Note that this objective function doesn't work well when an nonexistent direct path is used, but this is ok because constraints will be defined next.

# Routing Problem: Making it More Formal

- Design variables represent a candidate solution.
  - Sequence  $\mathbf{x}$  containing the cities to be visited, where  $x_i \in \{1, \dots, N\}$  and  $\mathbf{x}$  can be of any size.
  - The search space consists of all possible sequences of cities.
- Objective function defines the quality (or cost) of a solution.

$$\text{Minimise } f(\mathbf{x}) = \sum_{i=1}^{\text{size}(\mathbf{x})-1} D_{x_i, x_{i+1}}$$

- [Optional] Solutions must satisfy certain constraints, which define solution feasibility.
  - (Inexistent) direct paths between non-neighbouring cities must not be used (**explicit constraint**).
  - We must start at the city of origin and end at the city of destination (**explicit constraint**).
  - Only cities in  $\{1, \dots, N\}$  can be used (**implicit constraint**).

# Routing Problem: Making it More Formal

Minimise  $f(\mathbf{x})$

Subject to  $g_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, m$

$h_j(\mathbf{x}) = 0, \quad j = 1, \dots, n$

- (Inexistent) direct paths between non-neighbouring cities must not be used (**explicit constraint**).

Assume that we have a matrix  $D$  where each position  $D_{i,j}$  contains

- the distance to travel directly between city  $i$  and  $j$ , or
- -1 if such direct path does not exist.

$$h_1 : \mathbf{x} \rightarrow \{0,1\} \quad h_1(\mathbf{x}) = \begin{cases} 0 & \text{if } D_{x_i, x_{i+1}} \neq -1, \quad \forall i \in \{1, \dots, \text{size}(\mathbf{x}) - 1\} \\ 1 & \text{otherwise} \end{cases}$$

# Routing Problem: Making it More Formal

- Design variables represent a candidate solution.
  - Sequence  $\mathbf{x}$  containing the cities to be visited, where  $x_i \in \{1, \dots, N\}$  and  $\mathbf{x}$  can be of any size.
  - The search space consists of all possible sequences of cities.

- Objective function defines the quality (or cost) of a solution.

$$\text{Minimise } f(\mathbf{x}) = \sum_{i=1}^{\text{size}(\mathbf{x})-1} D_{x_i, x_{i+1}}$$

- [Optional] Solutions must satisfy certain constraints, which define solution feasibility.
  - $h_1(\mathbf{x}) = 0$
  - We must start at the city of origin and end at the city of destination (explicit constraint).
  - Only cities in  $\{1, \dots, N\}$  can be used (implicit constraint).

# Routing Problem: Making it More Formal

Minimise  $f(\mathbf{x})$

Subject to  $g_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, m$

$h_j(\mathbf{x}) = 0, \quad j = 1, \dots, n$

- We must start at the city of origin and end at the city of destination (**explicit constraint**).

$$h_2 : \mathbf{x} \rightarrow \{0,1\}$$

$$h_2(\mathbf{x}) = \begin{cases} 0 & \text{if } x_1 = \text{OriginCity and } x_{\text{size}(\mathbf{x})} = \text{DestinationCity} \\ 1 & \text{otherwise} \end{cases}$$

# Routing Problem: Making it More Formal

- Design variables represent a candidate solution.
  - Sequence  $\mathbf{x}$  containing the cities to be visited, where  $x_i \in \{1, \dots, N\}$  and  $\mathbf{x}$  can be of any size.
  - The search space consists of all possible sequences of cities.
- Objective function defines the quality (or cost) of a solution.
$$\text{Minimise } f(\mathbf{x}) = \sum_{i=1}^{\text{size}(\mathbf{x})-1} D_{x_i, x_{i+1}}$$
- [Optional] Solutions must satisfy certain constraints, which define solution feasibility.
  - $h_1(\mathbf{x}) = 0$
  - $h_2(\mathbf{x}) = 0$
  - Only cities in  $\{1, \dots, N\}$  can be used (implicit constraint).

# Routing Problem: Making it More Formal

- Design variables represent a candidate solution.
  - Sequence  $\mathbf{x}$  containing the cities to be visited, where  $x_i \in \{1, \dots, N\}$  and  $\mathbf{x}$  can be of any size.
  - The search space consists of all possible sequences of cities.
- Objective function defines the quality (or cost) of a solution.

$$\text{Minimise } f(\mathbf{x}) = \sum_{i=1}^{\text{size}(\mathbf{x})-1} D_{x_i, x_{i+1}}$$

- [Optional] Solutions must satisfy certain constraints, which define solution feasibility.
  - $h_1(\mathbf{x}) = 0$
  - $h_2(\mathbf{x}) = 0$

# Routing Problem: Making it More Formal

$$\text{Minimise } f(\mathbf{x}) = \sum_{i=1}^{\text{size}(\mathbf{x})-1} D_{x_i, x_{i+1}}$$

Subject to  $\mathbf{h}_1(\mathbf{x}) = 0$  and  $\mathbf{h}_2(\mathbf{x}) = 0$

Where  $x_i \in \{1, \dots, N\}$ ;  $\{1, \dots, N\}$  are the cities in the map;  $\mathbf{x}$  has any size;

$D$  is a matrix of distances, with each position  $D_{i,j}$  containing:

- the distance in km to travel directly between city  $i$  and  $j$ , or
- -1 if such direct path does not exist.

$$h_1(\mathbf{x}) = \begin{cases} 0 & \text{if } D_{x_i, x_{i+1}} \neq -1, \quad \forall i \in \{1, \dots, \text{size}(\mathbf{x}) - 1\} \\ 1 & \text{otherwise} \end{cases}$$

$$h_2(\mathbf{x}) = \begin{cases} 0 & \text{if } x_1 = \text{OriginCity and } x_{\text{size}(\mathbf{x})} = \text{DestinationCity} \\ 1 & \text{otherwise} \end{cases}$$

# Summary

- We can formulate an optimisation problem by specifying:
  - Design variables.
  - Objective functions.
  - Constraints.

# Next

- How to solve optimisation problems?



Photo from: <http://cdn.motocross.transworld.net/files/2009/11/img-7532.jpg>

# Hill Climbing

Leandro L. Minku

# Hill-Climbing

## Hill-Climbing (assuming maximisation)

1. `current_solution` = generate initial candidate solution randomly
2. Repeat:
  - 2.1 generate neighbour solutions (differ from current solution by a single element)
  - 2.2 `best_neighbour` = get highest quality neighbour of `current_solution`
  - 2.3 If  $\text{quality}(\text{best\_neighbour}) \leq \text{quality}(\text{current\_solution})$ 
    - 2.3.1 Return `current_solution`
  - 2.4 `current_solution` = `best_neighbour`

# Designing Representation, Initialisation and Neighbourhood Operators

## Hill-Climbing (assuming maximisation)

1. `current_solution = generate initial solution randomly`
2. Repeat:
  - 2.1 `generate neighbour solutions` (differ from current solution by a single element)
  - 2.2 `best_neighbour = get highest quality neighbour of current_solution`
  - 2.3 If `quality(best_neighbour) <= quality(current_solution)`
    - 2.3.1 Return `current_solution`
  - 2.4 `current_solution = best_neighbour`

- **Representation:**
  - How to store the design variable.
  - E.g., boolean, integer or float variable or array.
- **Initialisation procedure:**
  - Usually involve randomness.
- **Neighbourhood operator:**
  - How to generate neighbour solutions.

# Illustrative Example

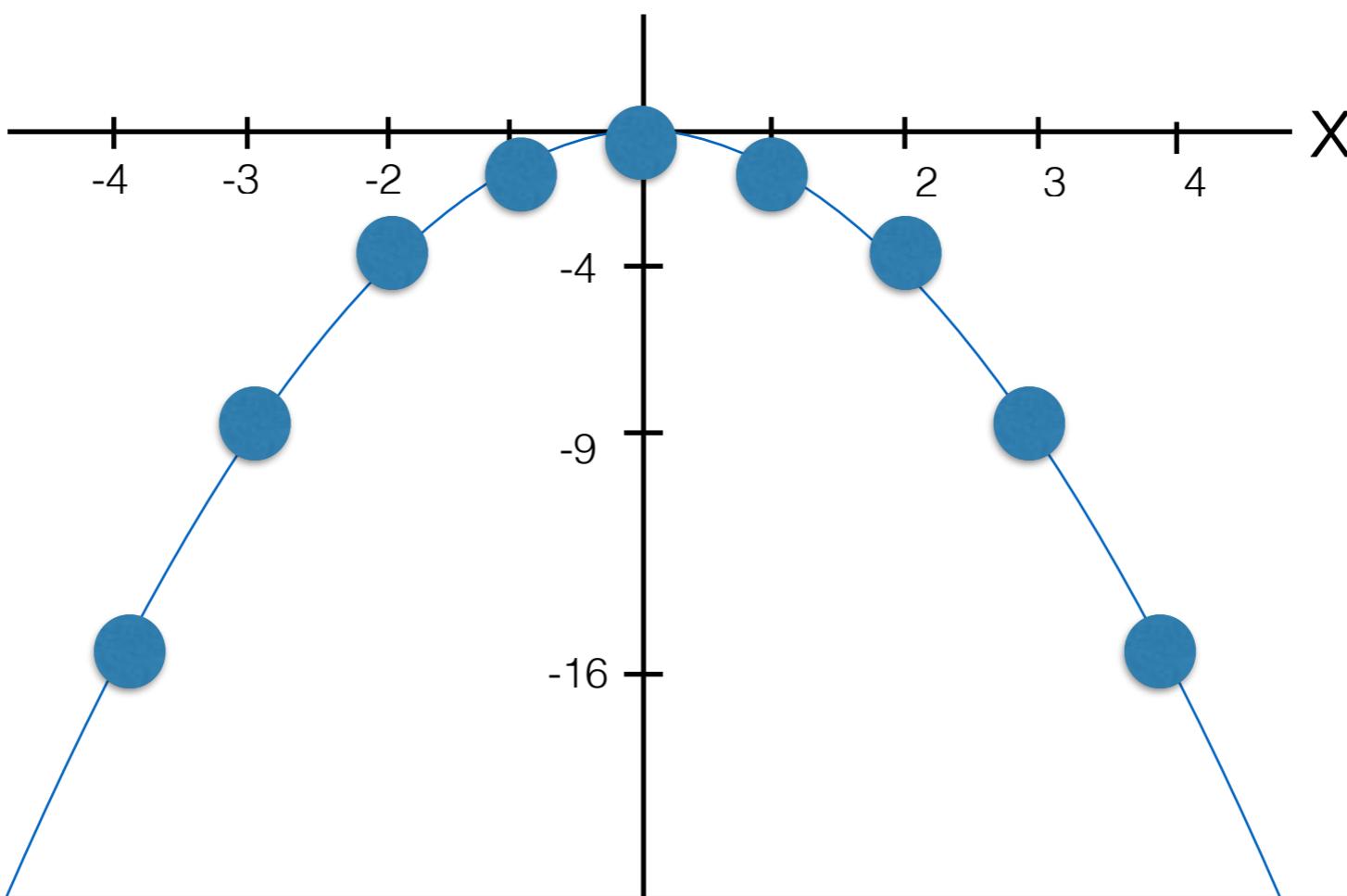
- Design variables represent a candidate solution.
  - $x \in \mathbb{Z}$
  - Our search space are all integer numbers.
- [Optional] Solutions must satisfy certain constraints, which define solution feasibility.
  - None
- Objective function defines the quality (or cost) of a solution.
  - $f(x) = -x^2$ , to be maximised

# Illustrative Example

- Representation:
  - Integer variable.
- Initialisation procedure:
  - Initialise with an integer value picked uniformly at random.
- Neighbourhood operator:
  - Add or subtract 1.

# Illustrative Example

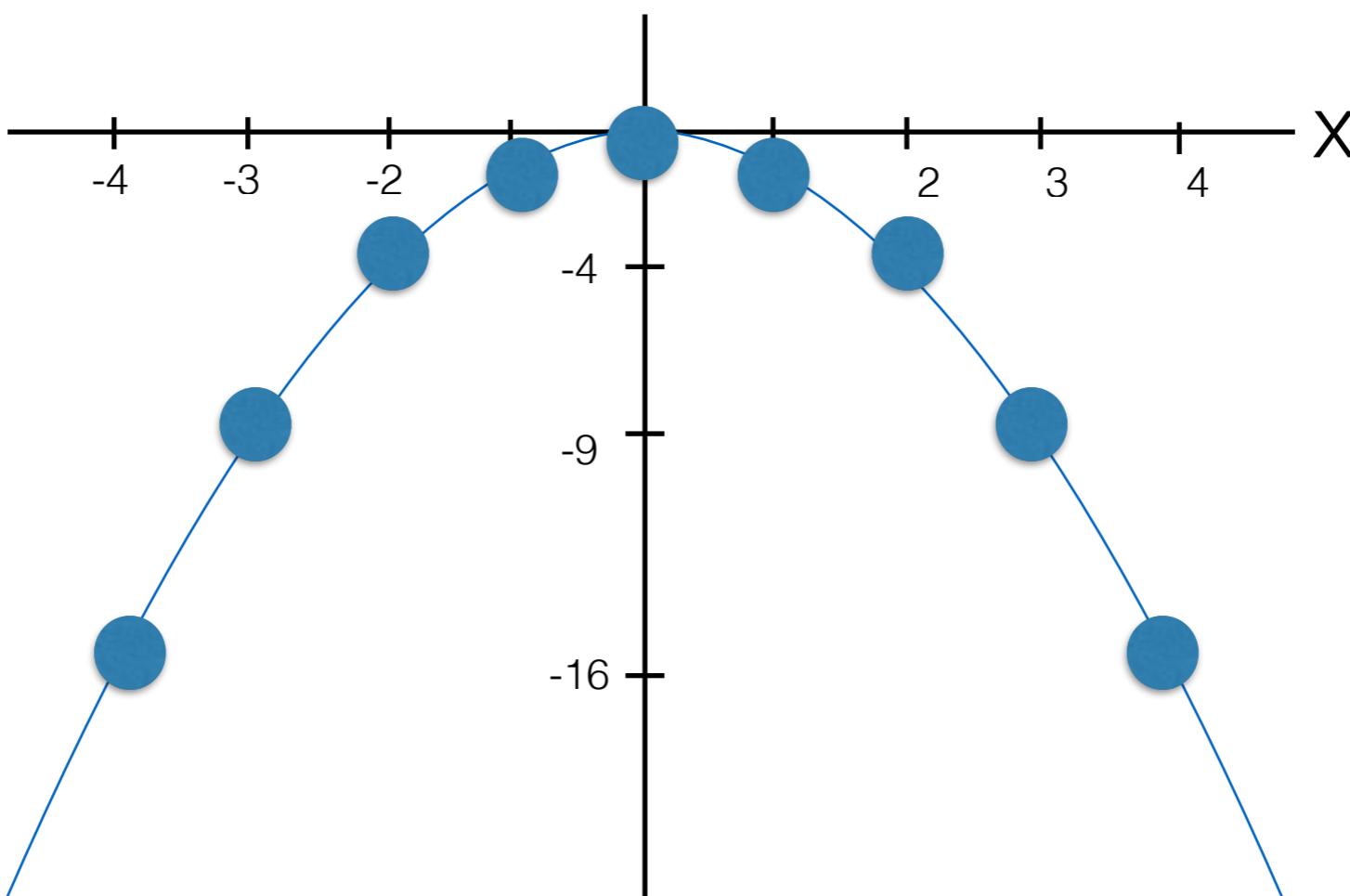
$$f(x) = -x^2$$



This is an illustrative example to understand the behaviour of the algorithm. In reality, we are unlikely to know the shape of our function to be optimised beforehand.

# Illustrative Example

$$f(x) = -x^2$$

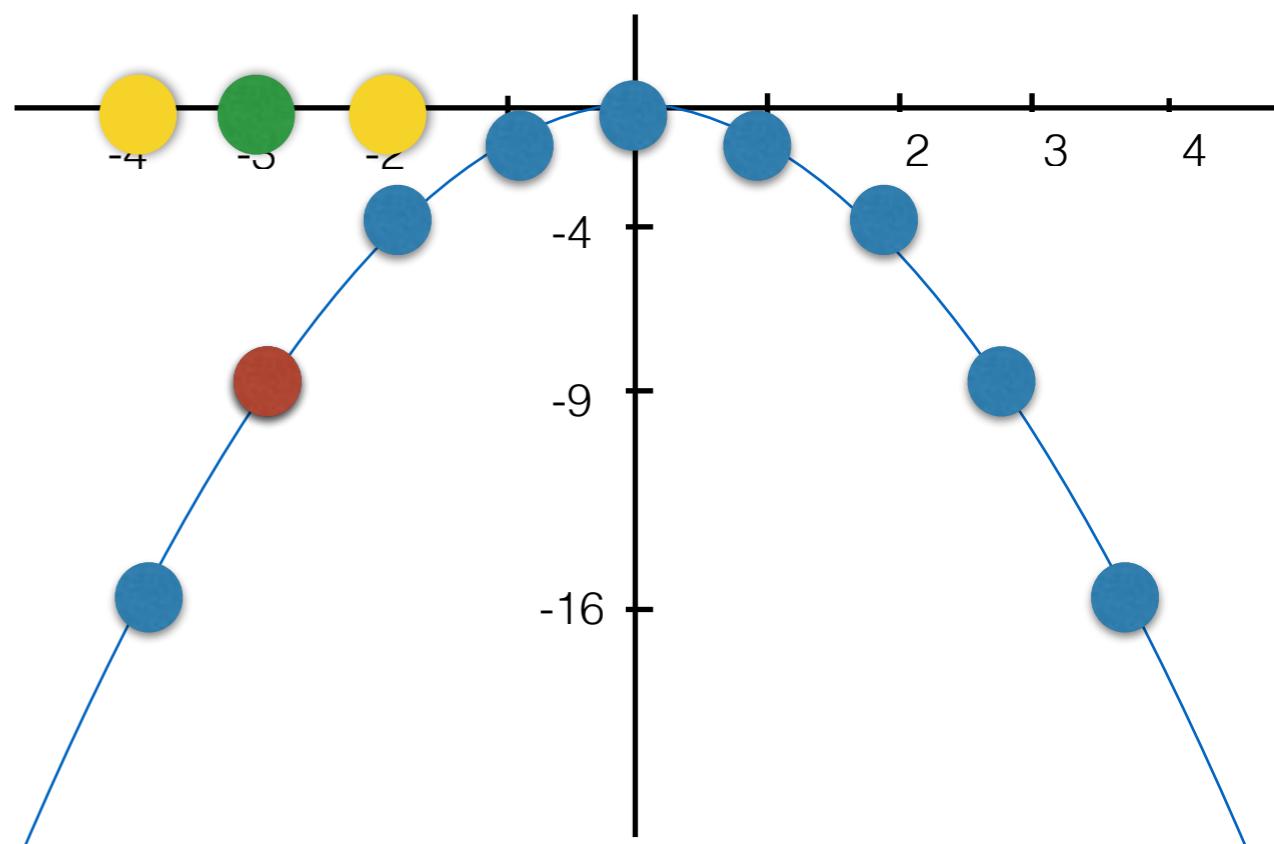


Other problems may have more dimensions,  
and more neighbours for each candidate solution.

# Illustrative Example

## Hill-Climbing (assuming maximisation)

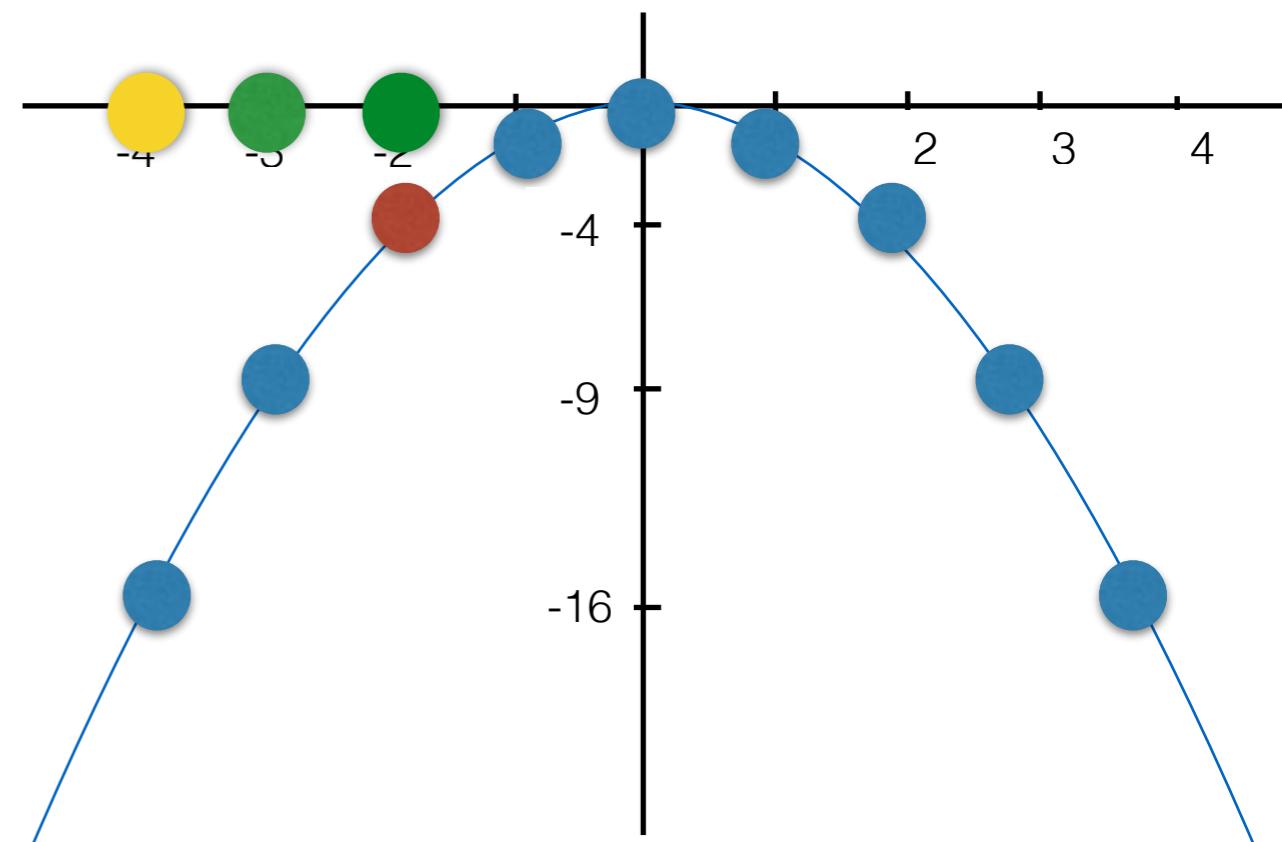
1. `current_solution` = generate initial solution randomly
2. Repeat:
  - 2.1 generate neighbour solutions (differ from current solution by a single element)
  - 2.2 `best_neighbour` = get highest quality neighbour of `current_solution`
  - 2.3 If `quality(best_neighbour) <= quality(current_solution)`
    - 2.3.1 Return `current_solution`
  - 2.4 `current_solution` = `best_neighbour`



# Illustrative Example

## Hill-Climbing (assuming maximisation)

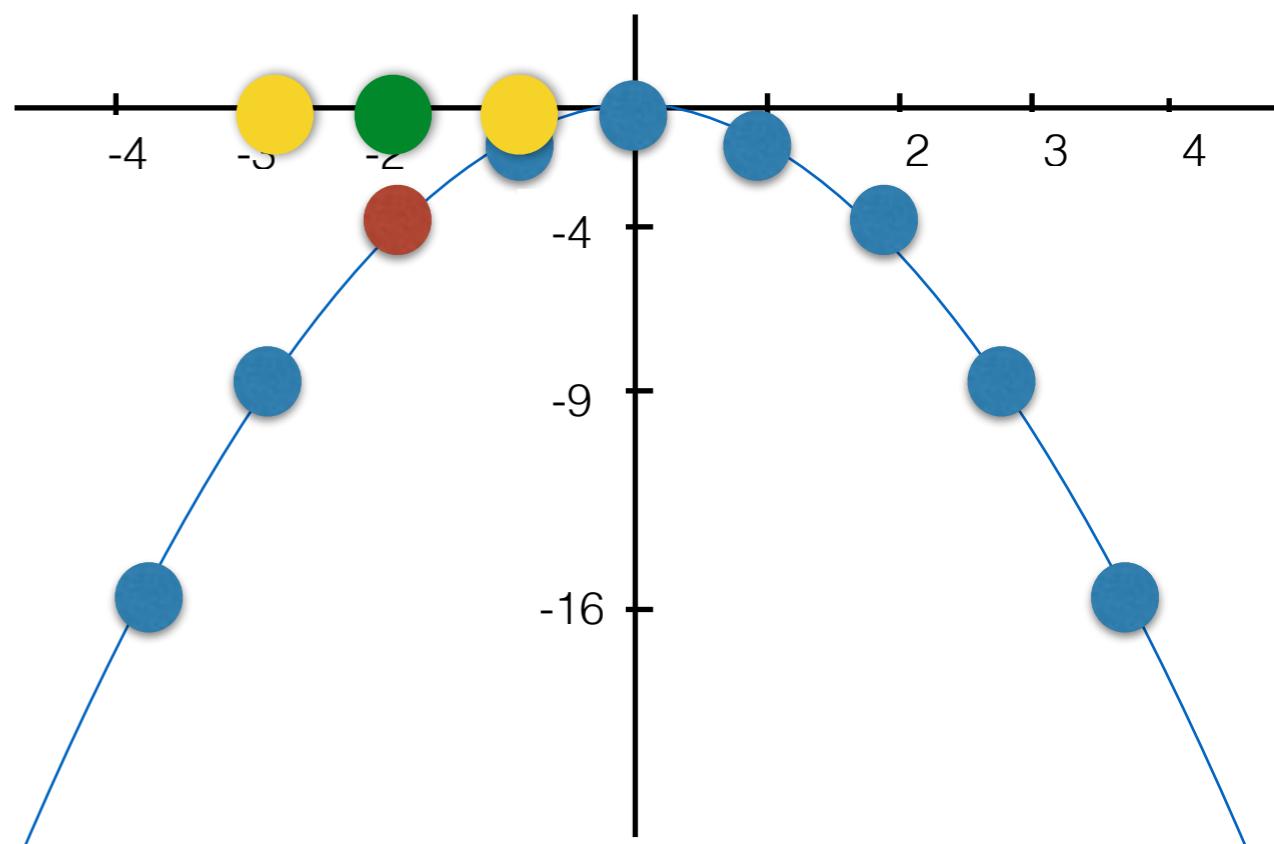
1. `current_solution` = generate initial solution randomly
2. Repeat:
  - 2.1 generate neighbour solutions (differ from current solution by a single element)
  - 2.2 `best_neighbour` = get highest quality neighbour of `current_solution`
  - 2.3 If `quality(best_neighbour) <= quality(current_solution)`
    - 2.3.1 Return `current_solution`
  - 2.4 `current_solution` = `best_neighbour`



# Illustrative Example

## Hill-Climbing (assuming maximisation)

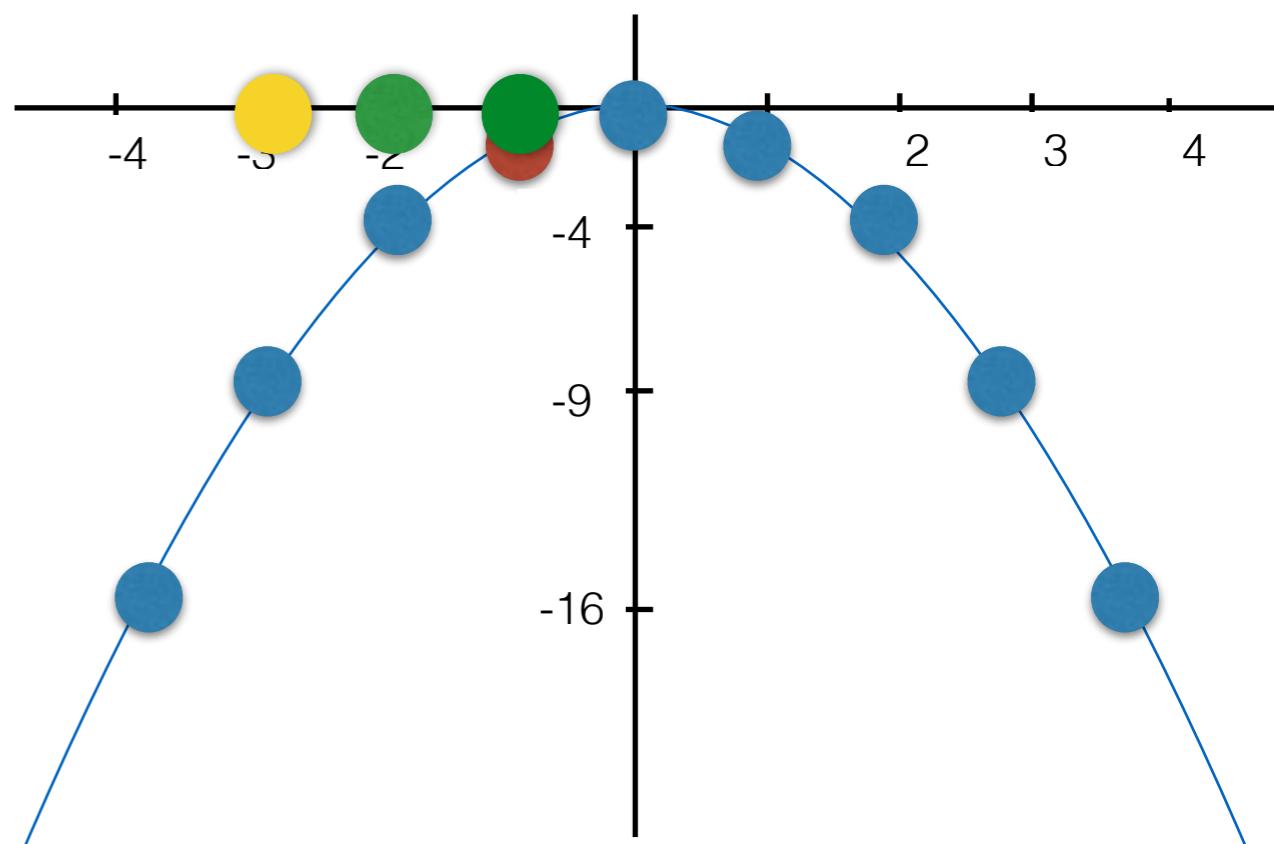
1. `current_solution` = generate initial solution randomly
2. Repeat:
  - 2.1 generate neighbour solutions (differ from current solution by a single element)
  - 2.2 `best_neighbour` = get highest quality neighbour of `current_solution`
  - 2.3 If `quality(best_neighbour) <= quality(current_solution)`
    - 2.3.1 Return `current_solution`
  - 2.4 `current_solution` = `best_neighbour`



# Illustrative Example

## Hill-Climbing (assuming maximisation)

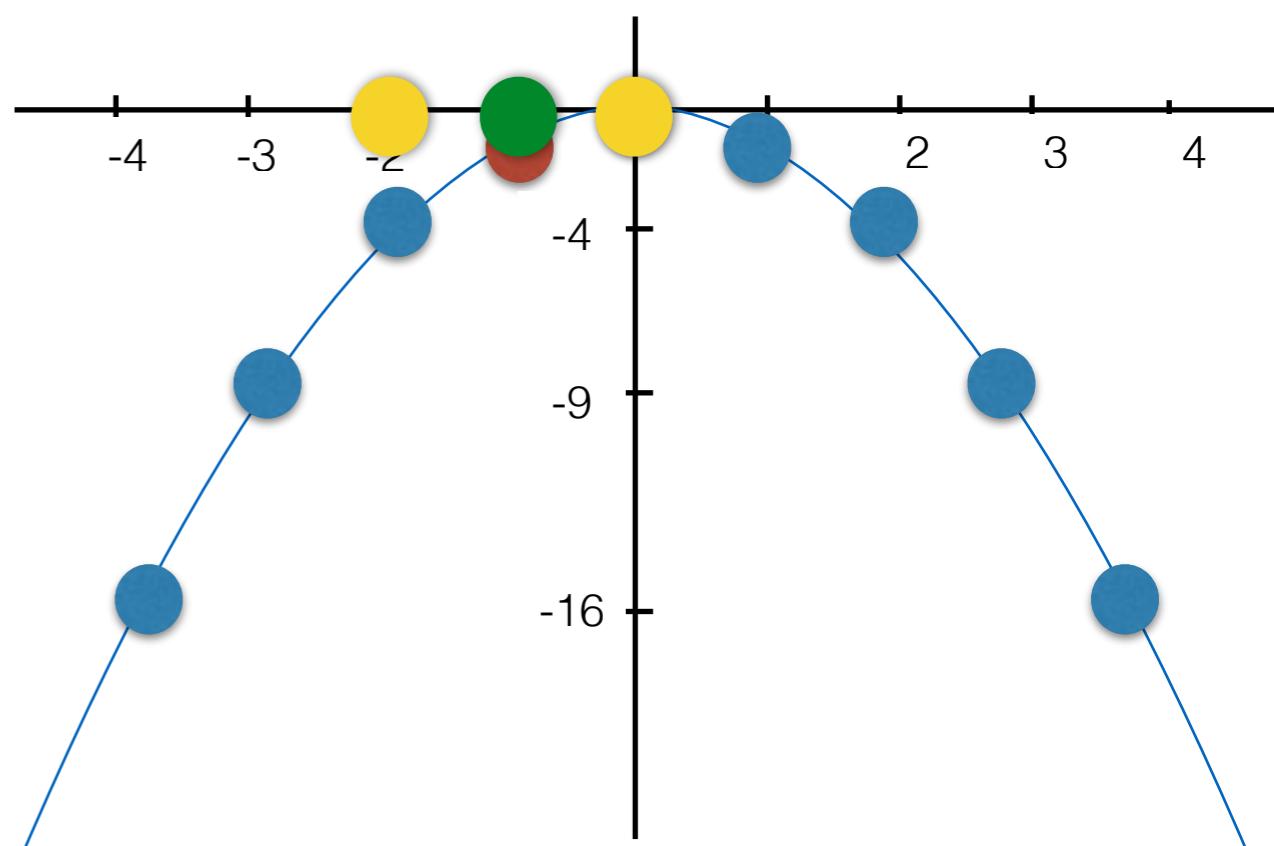
1. `current_solution` = generate initial solution randomly
2. Repeat:
  - 2.1 generate neighbour solutions (differ from current solution by a single element)
  - 2.2 `best_neighbour` = get highest quality neighbour of `current_solution`
  - 2.3 If `quality(best_neighbour) <= quality(current_solution)`
    - 2.3.1 Return `current_solution`
  - 2.4 `current_solution` = `best_neighbour`



# Illustrative Example

## Hill-Climbing (assuming maximisation)

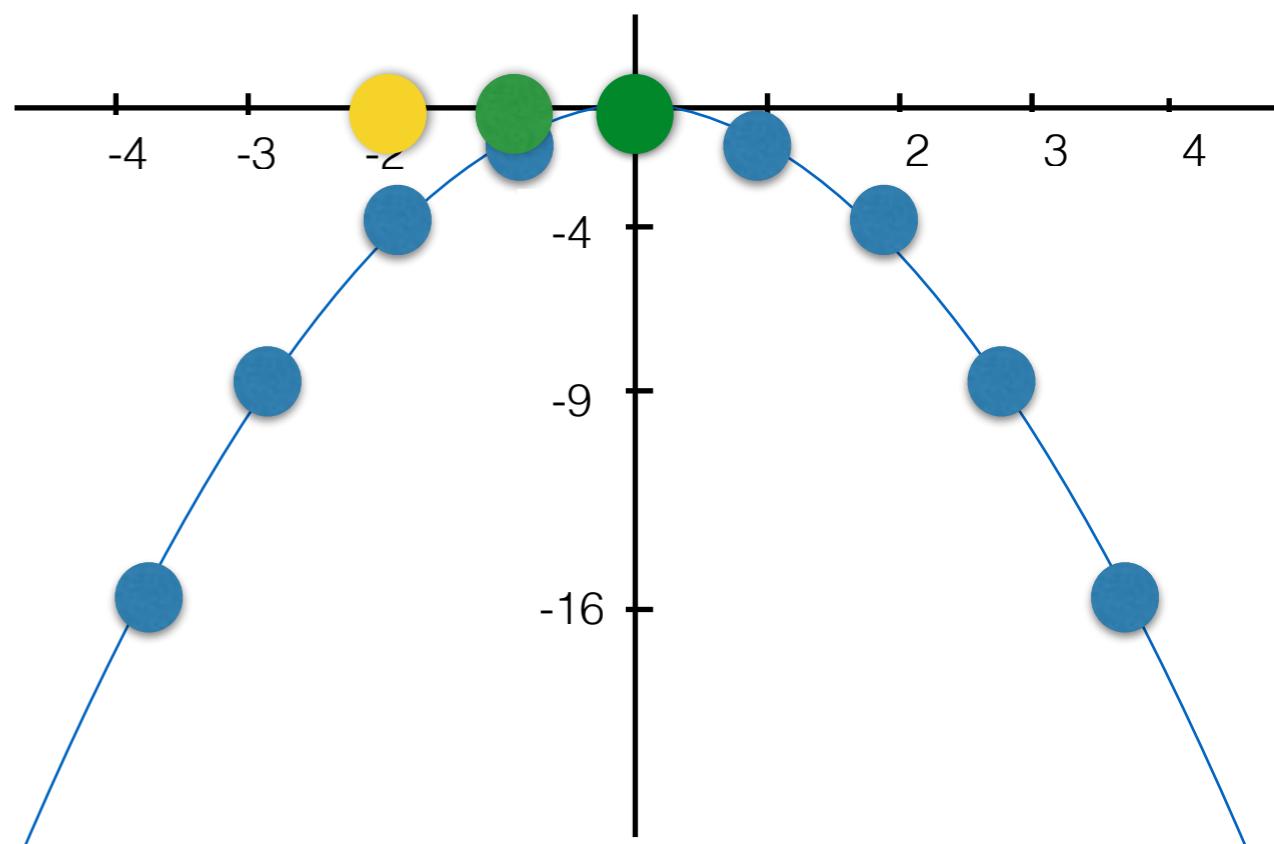
1. `current_solution` = generate initial solution randomly
2. Repeat:
  - 2.1 generate neighbour solutions (differ from current solution by a single element)
  - 2.2 `best_neighbour` = get highest quality neighbour of `current_solution`
  - 2.3 If `quality(best_neighbour) <= quality(current_solution)`
    - 2.3.1 Return `current_solution`
  - 2.4 `current_solution` = `best_neighbour`



# Illustrative Example

## Hill-Climbing (assuming maximisation)

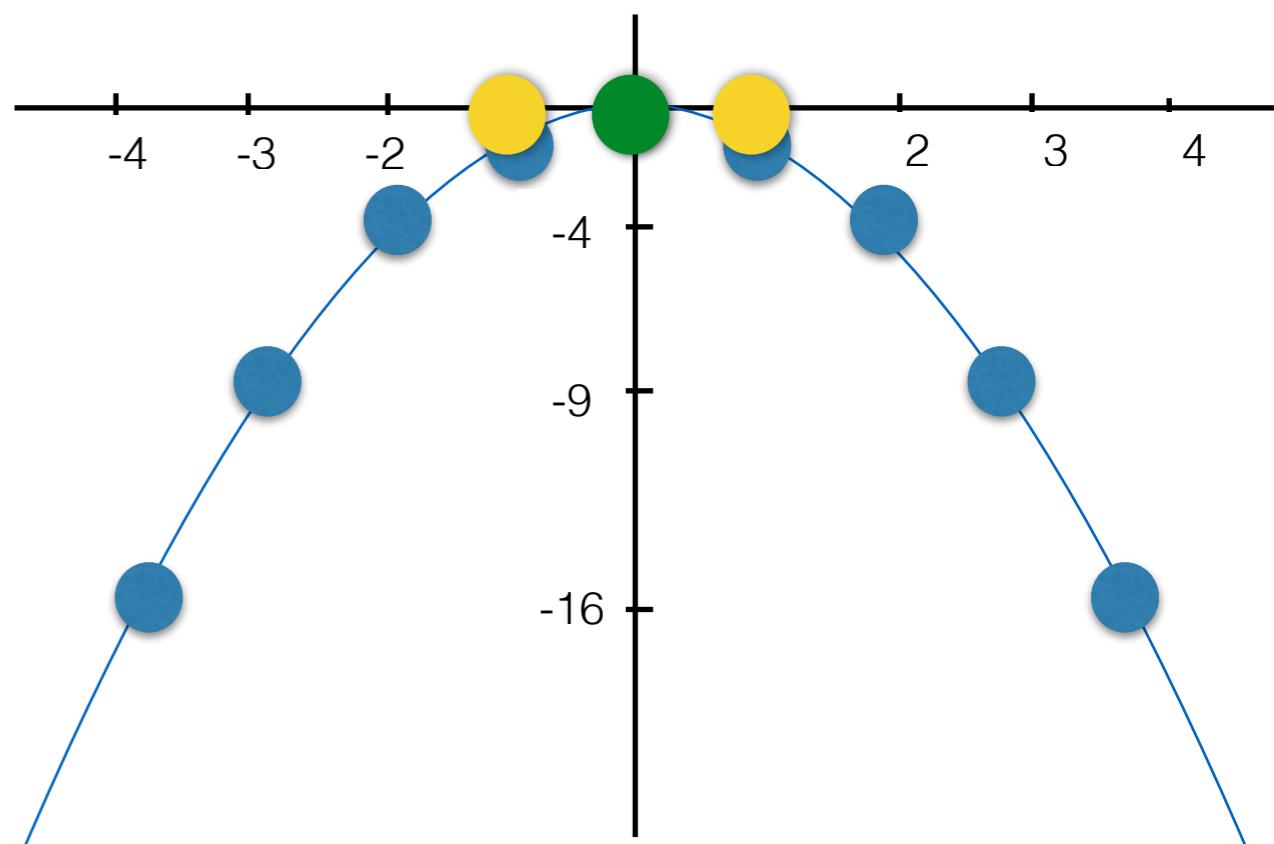
1. `current_solution` = generate initial solution randomly
2. Repeat:
  - 2.1 generate neighbour solutions (differ from current solution by a single element)
  - 2.2 `best_neighbour` = get highest quality neighbour of `current_solution`
  - 2.3 If `quality(best_neighbour) <= quality(current_solution)`
    - 2.3.1 Return `current_solution`
  - 2.4 `current_solution` = `best_neighbour`



# Illustrative Example

## Hill-Climbing (assuming maximisation)

1. `current_solution` = generate initial solution randomly
2. Repeat:
  - 2.1 generate neighbour solutions (differ from current solution by a single element)
  - 2.2 `best_neighbour` = get highest quality neighbour of `current_solution`
  - 2.3 If `quality(best_neighbour) <= quality(current_solution)`
    - 2.3.1 Return `current_solution`
  - 2.4 `current_solution` = `best_neighbour`

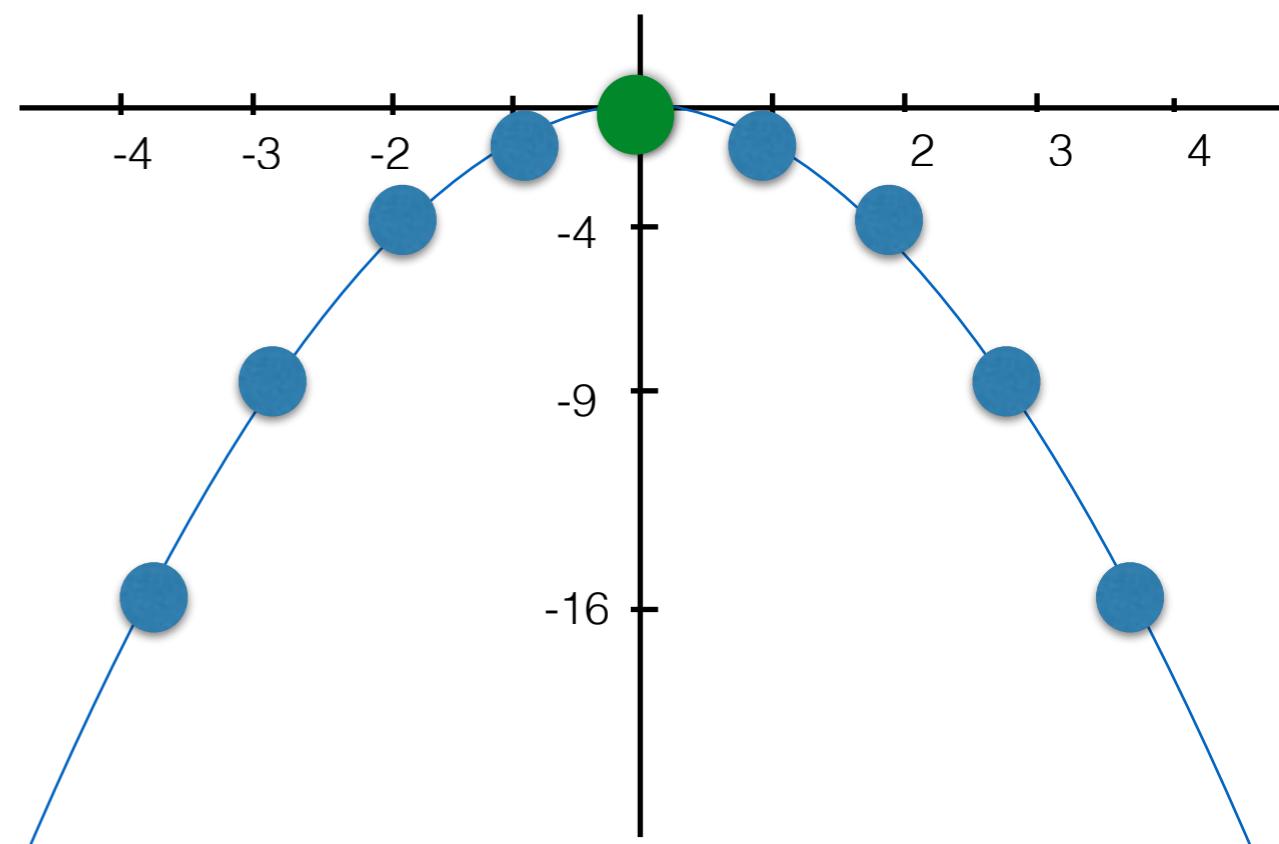


# Illustrative Example

## Hill-Climbing (assuming maximisation)

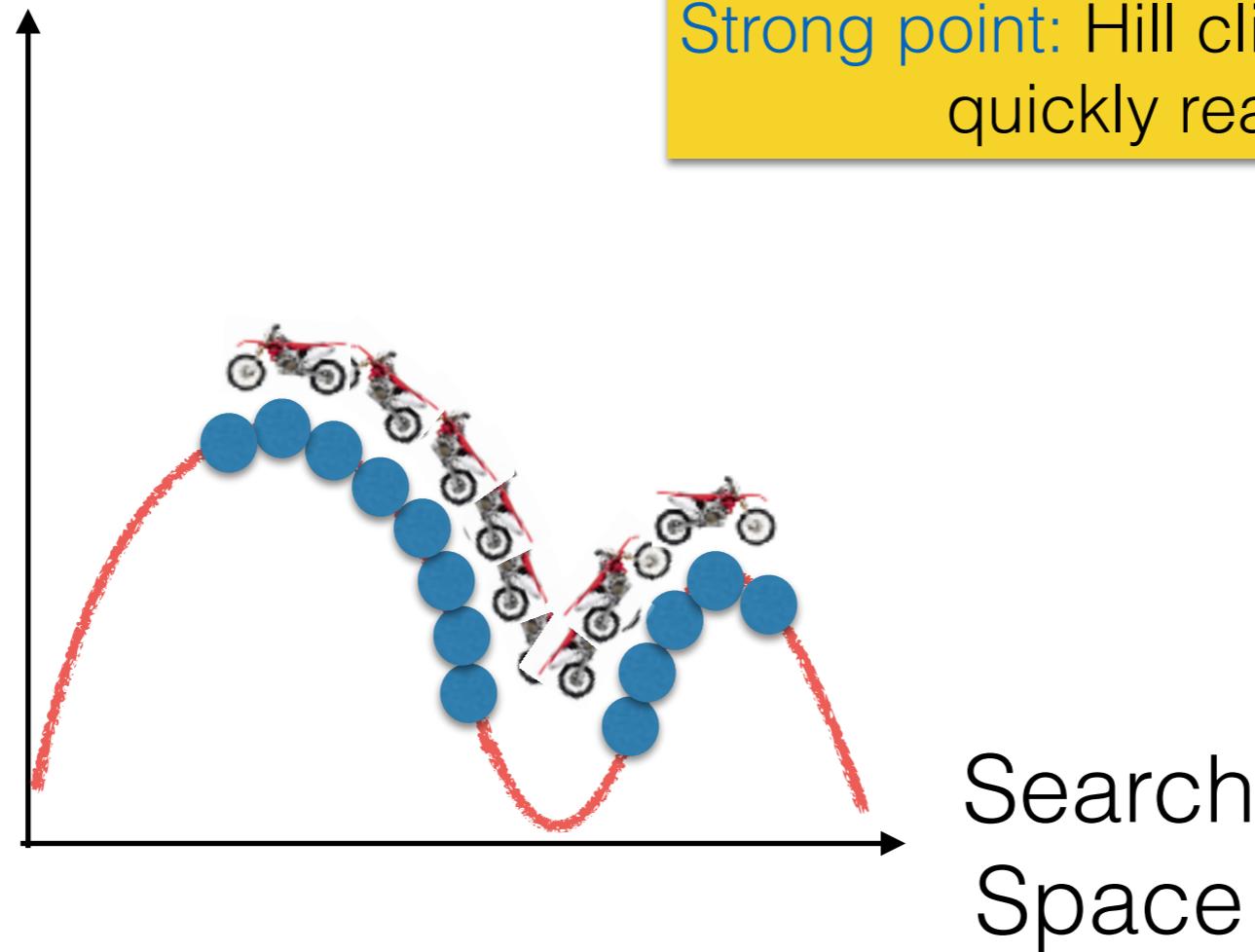
1. `current_solution` = generate initial solution randomly
2. Repeat:
  - 2.1 generate neighbour solutions (differ from current solution by a single element)
  - 2.2 `best_neighbour` = get highest quality neighbour of `current_solution`
  - 2.3 If `quality(best_neighbour) <= quality(current_solution)`
    - 2.3.1 Return `current_solution`
  - 2.4 `current_solution` = `best_neighbour`

Until a maximum number of iterations



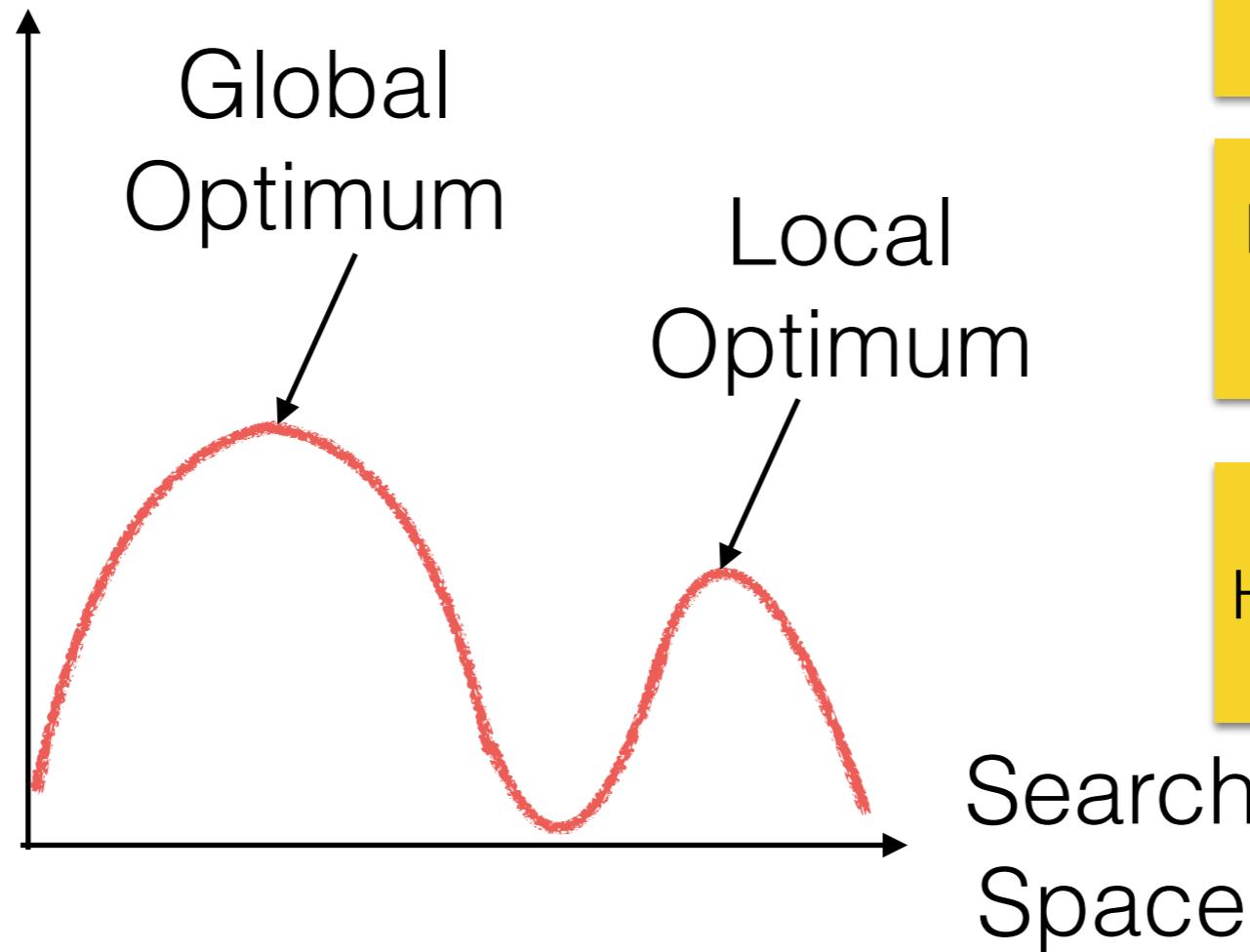
# General Idea

Objective  
Function



# Greedy Local Search

Objective  
Function



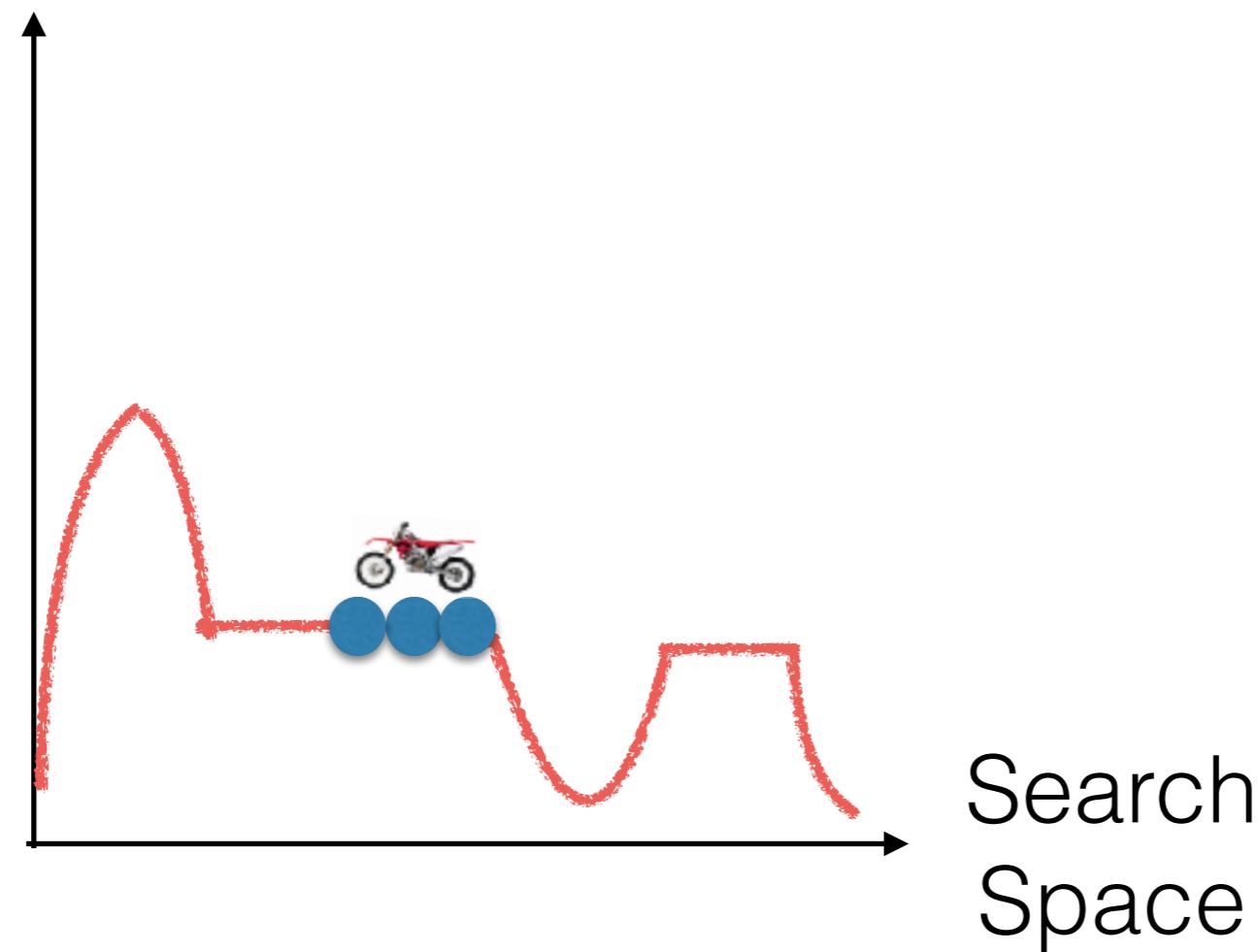
**Weakness:** Hill-climbing may get trapped in a local optimum.

Hill-climbing is a local search method.

Hill-climbing is greedy.

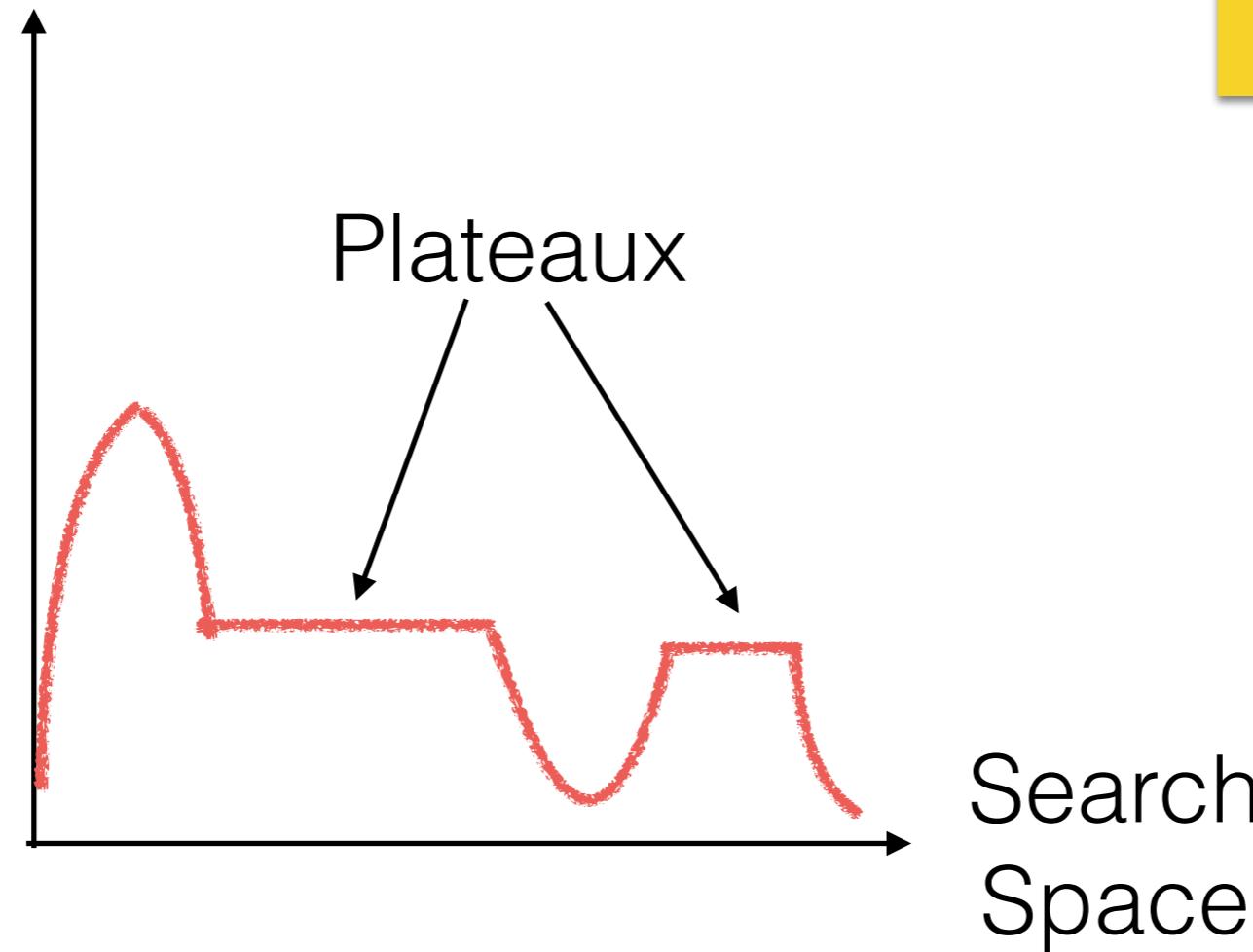
# Greedy Local Search

Objective  
Function



# Greedy Local Search

Objective  
Function



Weakness: Hill-climbing may get trapped in plateaux.

# Optimality, Time and Space Complexity

- Optimality:
  - Hill Climbing is not guaranteed to find optimal solutions.
- Time complexity (worst case scenario):
  - We will run until the maximum number of iterations  $m$  is reached.
  - Within each iteration, we will generate a maximum number of neighbours  $n$ , each of which may take  $O(p)$  each to generate.
  - Worst case scenario:  $O(mnp)$ .
- Space complexity (worst case scenario):
  - Assume that the design variable is represented by  $O(q)$ .
  - Within each iteration, we will generate a maximum number of neighbours  $n$ .
  - Space complexity:  $O(nq)$

# Summary

- Hill-climbing is an example of greedy local search optimisation algorithm.
- It can quickly find a local optimum, but may not find optimal solutions.
- Its success depends on the shape of the objective function. As it is a relatively simple algorithm, it can be attempted before more complex algorithms are investigated.

# Next

- How to avoid getting stuck in local optima and plateaux?



UNIVERSITY OF  
BIRMINGHAM

# AI1/AI&ML - Informed Search

Dr Leonardo Stella



# Aims of the Session

This session aims to help you:

- Describe the difference between uninformed and informed search
- Understand the concept of a heuristic function in informed search
- Analyse the performance of A\* and apply the algorithm to solve search problems

# Overview

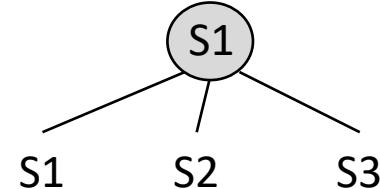
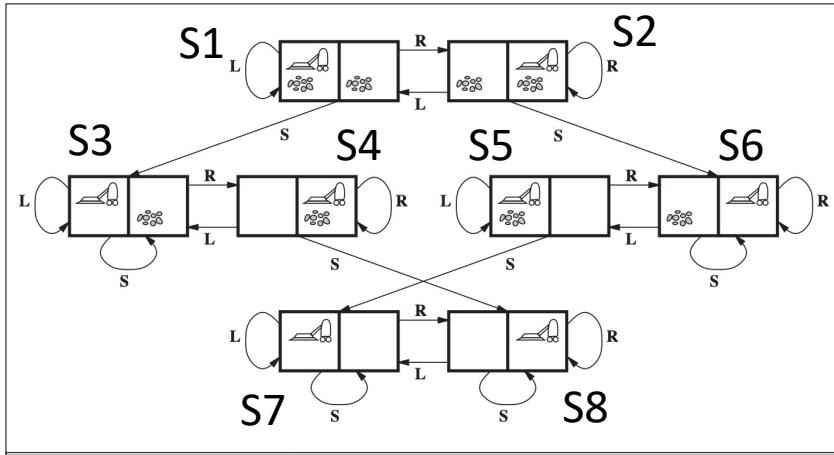
- **Recap – Uninformed Search**
- Informed Search
- A\* Search

# Searching for Solutions

- A solution is an action sequence from an initial state to a goal state
- Possible action sequences form a **search tree** with initial state at the root; actions are the branches and nodes correspond to the state space
- The idea is to expand the current state by applying each possible action: this generates a new set of states

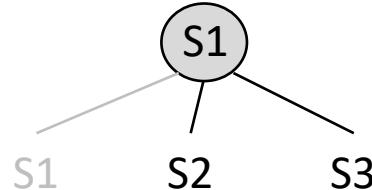
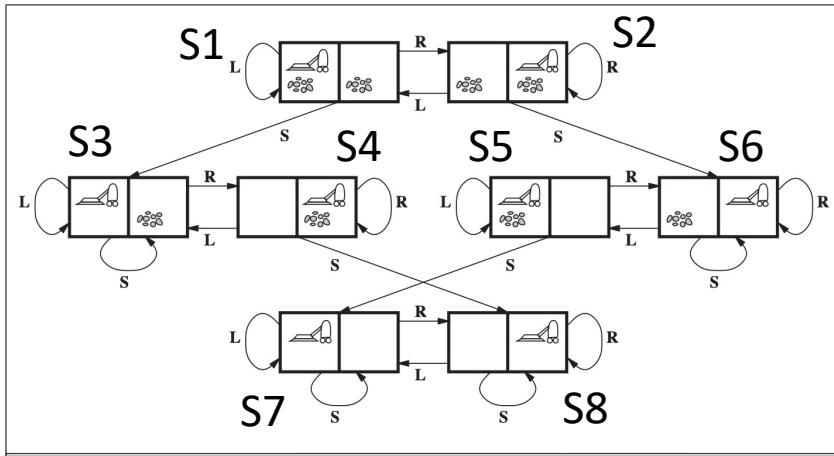
# Searching for Solutions

- Let us consider the example from before
- If  $S_1$  is the initial state and  $\{S_7, S_8\}$  is the set of goal states, the corresponding search tree after expanding the initial state is:



# Searching for Solutions

- Each of the three nodes resulting from the first expansion is a **leaf node**
- The set of all leaf nodes available for expansion at any given time is called the **frontier** (also sometimes called the **open list**)
- The path from S1 to S1 is a **loopy path** and in general is not considered

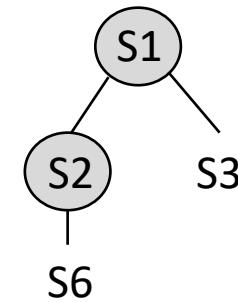
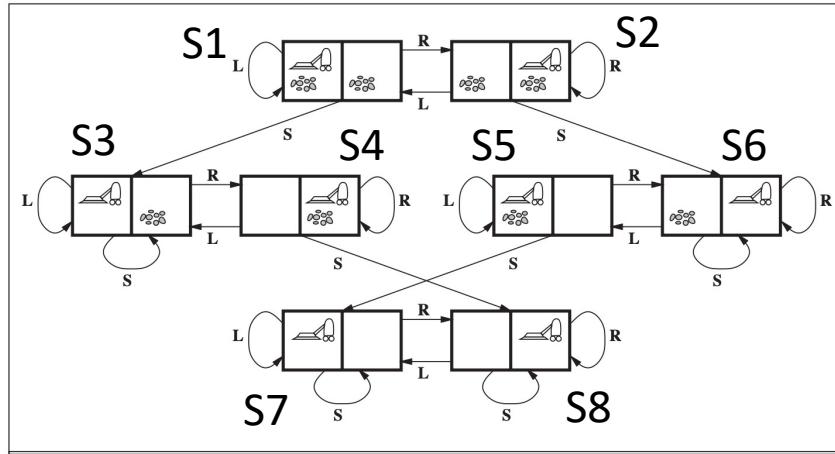


# Uninformed Search Strategies

- **Uninformed search** (also called **blind search**) means that the strategies have no additional information about states beyond that provided in the problem definition
- Uninformed search strategies can only generate successors and distinguish a goal state from a non-goal state
- The key difference between two uninformed search strategies is the **order** in which nodes are expanded

# Breadth-First Search vs Depth-First Search

- BFS would expand the shallowest node, namely S3
- DFS would expend the deepest node, namely S6



# Overview

- Recap – Uninformed Search
- **Informed Search**
- A\* Search

# Informed Search Strategies

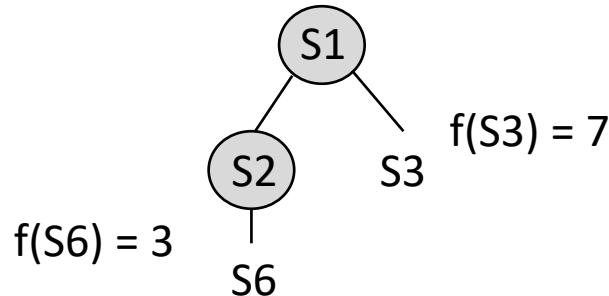
- **Informed search** strategies use problem-specific knowledge beyond the definition of the problem itself
- Informed search strategies can find solutions more efficiently compared to uninformed search

# Informed Search Strategies

- The general approach, called **best-first search**, is to determine which node to expand based on an **evaluation function**

$$f(n): \text{node} \rightarrow \text{cost estimate}$$

- This function acts as a cost estimate: the node with the lowest cost is the one that is expanded next

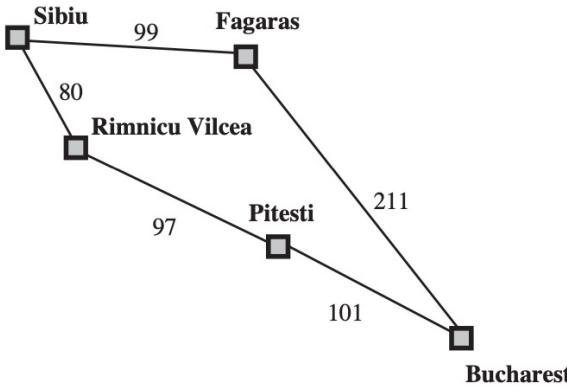


# Informed Search Strategies - Heuristics

- The evaluation function  $f(n)$  for most best-first algorithms includes a **heuristic function** as a component:  
$$h(n) = \text{estimated cost of the cheapest path from node } n \text{ to a goal node}$$
- Heuristic functions are the most common form in which new knowledge is given to the search algorithm. If  $n$  is a goal node, then  $h(n) = 0$
- A heuristic can be a rule of thumb, common knowledge; it is quick to compute, but not guaranteed to work (nor to yield optimal solutions)

# Informed Search Strategies - Heuristics

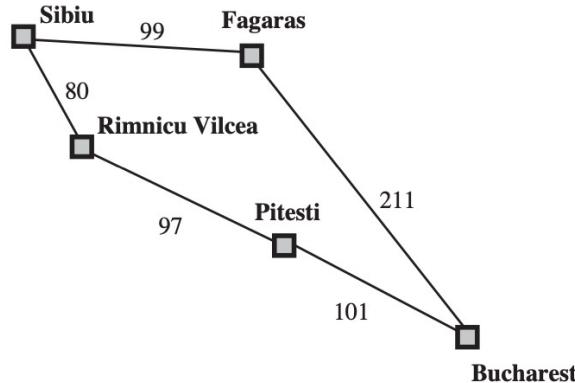
- Consider the problem to find the shortest path to Bucharest in Romania



- We can use the straight-line distance heuristic, denoted by  $h_{SLD}$
- This is a useful heuristic as it is correlated with actual road distances

# Informed Search Strategies - Heuristics

- Consider the problem to find the shortest path to Bucharest in Romania

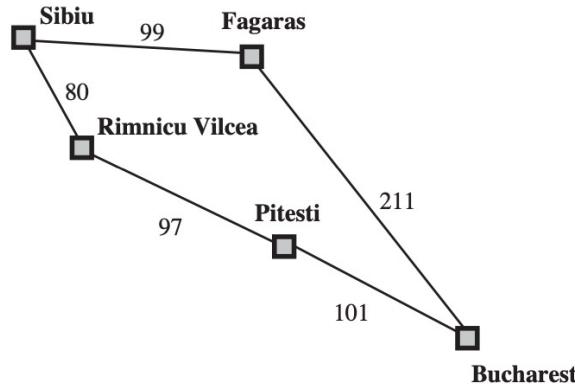


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

- The straight-line distances  $h_{SLD}$  are shown in the table above
- For example, the SLD from Sibiu would be 253

# Informed Search Strategies - Heuristics

- Consider the problem to find the shortest path to Bucharest in Romania

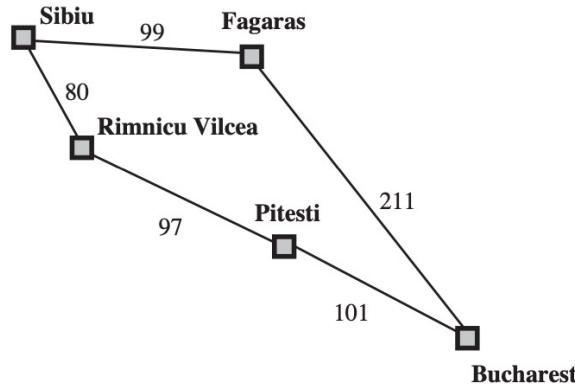


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

- If we use  $f(n) = h_{SLD}(n)$ , then from Sibiu we expand Fagaras
- This is because Fagaras has SLD 176, while Rimnicu Vilcea 193

# Informed Search Strategies - Heuristics

- Consider the problem to find the shortest path to Bucharest in Romania



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

- When  $f(n) = h(n)$ , we call this strategy **Greedy Best-First Search**

# Overview

- Recap – Uninformed Search
- Informed Search
- A\* Search

# A\* Search

- The most widely known informed search strategy is **A\***
- This search strategy evaluates nodes using the following cost function

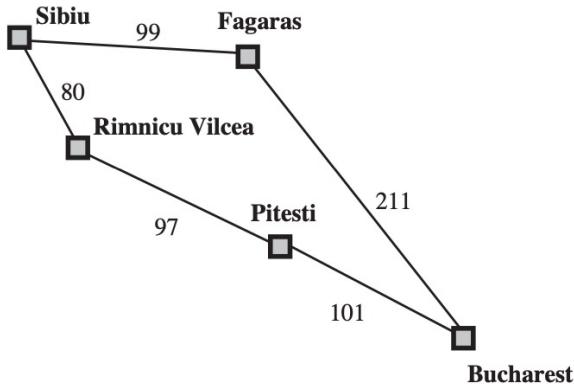
$$f(n) = g(n) + h(n)$$

where  $g(n)$  is the cost to reach the node and  $h(n)$  is the heuristic from the node to the goal

- This is equivalent to *the cost of the cheapest solution through node n*

# A\* Search - Example

- Consider the problem to find the shortest path to Bucharest in Romania



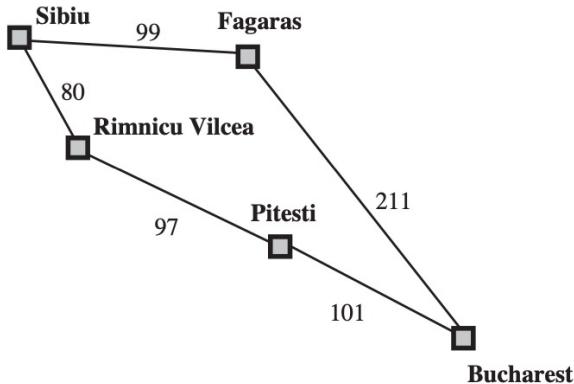
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

- Let us consider Sibiu as the initial state. Calculate  $f(n)$  to choose which node to expand, starting with Fagaras

$$f(Fagaras) = g(Fagaras) + h(Fagaras)$$

# A\* Search - Example

- Consider the problem to find the shortest path to Bucharest in Romania



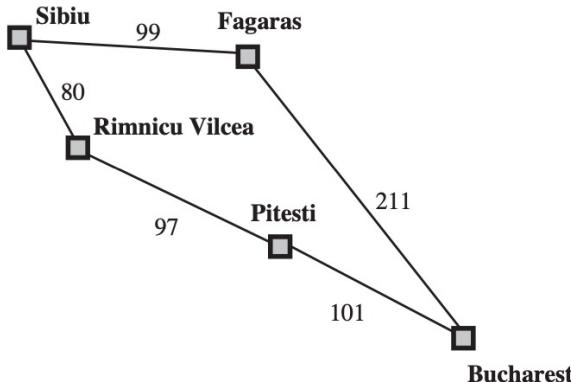
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

- Let us consider Sibiu as the initial state. Calculate  $f(n)$  to choose which node to expand, starting with Fagaras

$$f(Fagaras) = 99 + 176 = 275$$

# A\* Search - Example

- Consider the problem to find the shortest path to Bucharest in Romania



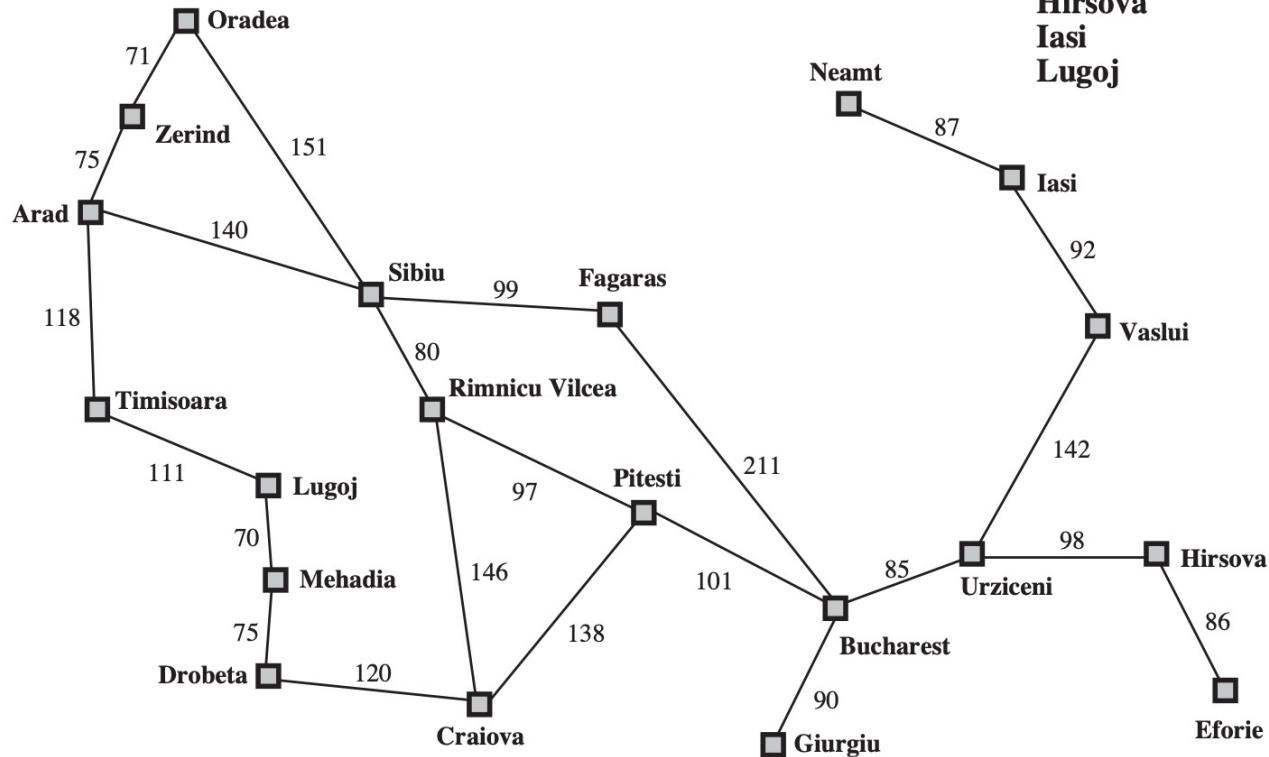
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

- Repeat the calculation for all the children, i.e., for Rimnicu Vilcea  
 $f(Rimnicu\ Vilcea) = 80 + 193 = 273$

# A\* Search - Algorithm

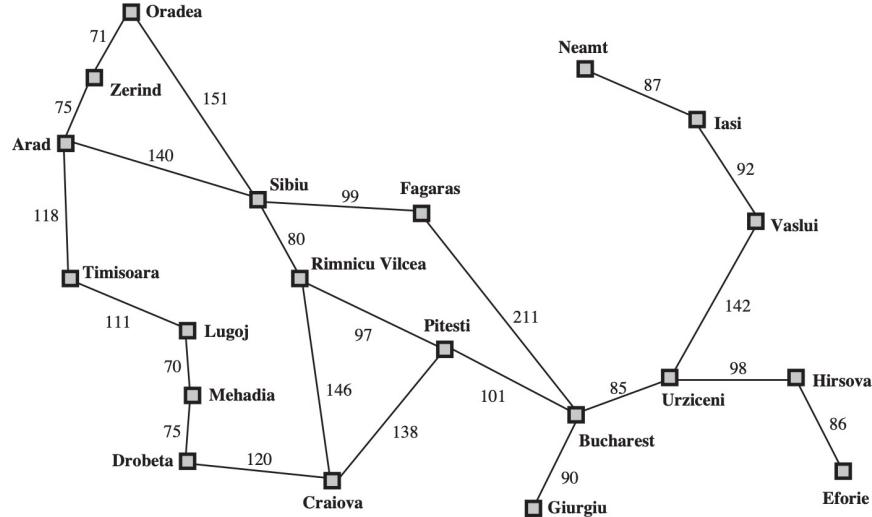
- A\* search algorithm:
  - **Expand** the node in the frontier with smallest cost  $f(n) = g(n) + h(n)$
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - If the state of a given child is in the frontier
    - If the frontier node has a larger  $g(n)$ , place the child into the frontier and remove the node with larger  $g(n)$  from the frontier
  - **Stop** when a goal node is visited

# A\* Search



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

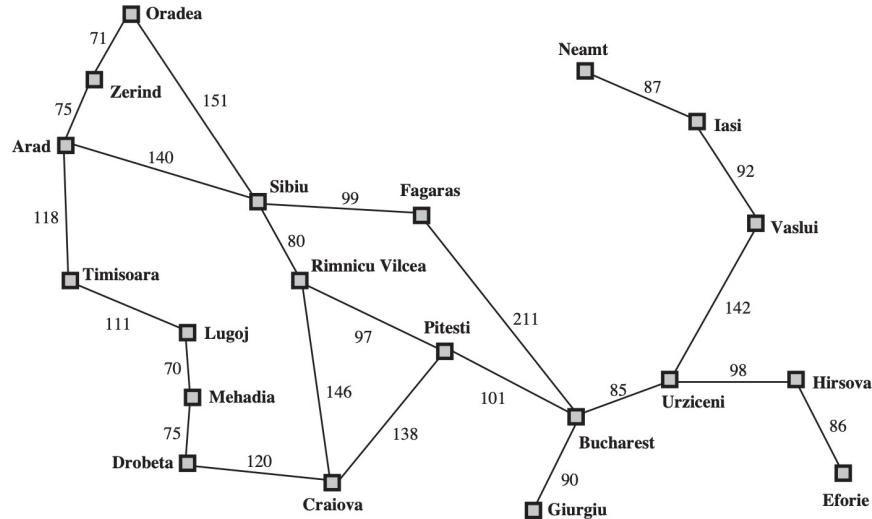
# A\* Search



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

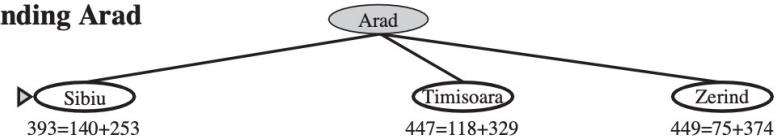
► Arad  
366=0+366

# A\* Search

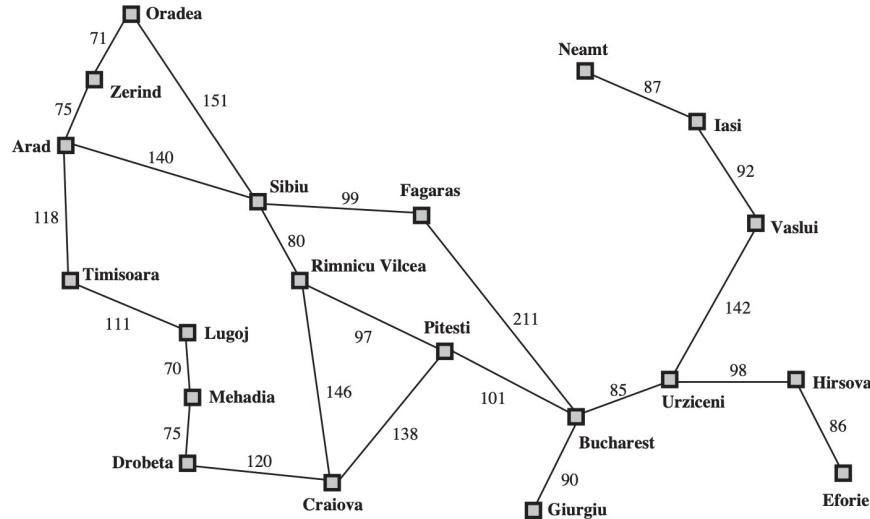


<b>Arad</b>	366	<b>Mehadia</b>	241
<b>Bucharest</b>	0	<b>Neamt</b>	234
<b>Craiova</b>	160	<b>Oradea</b>	380
<b>Drobeta</b>	242	<b>Pitesti</b>	100
<b>Eforie</b>	161	<b>Rimnicu Vilcea</b>	193
<b>Fagaras</b>	176	<b>Sibiu</b>	253
<b>Giurgiu</b>	77	<b>Timisoara</b>	329
<b>Hirsova</b>	151	<b>Urziceni</b>	80
<b>Iasi</b>	226	<b>Vaslui</b>	199
<b>Lugoj</b>	244	<b>Zerind</b>	374

After expanding Arad

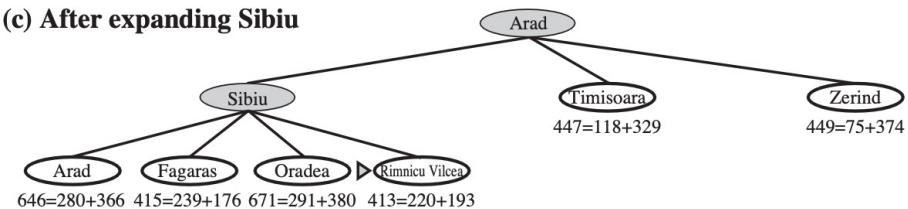


# A\* Search

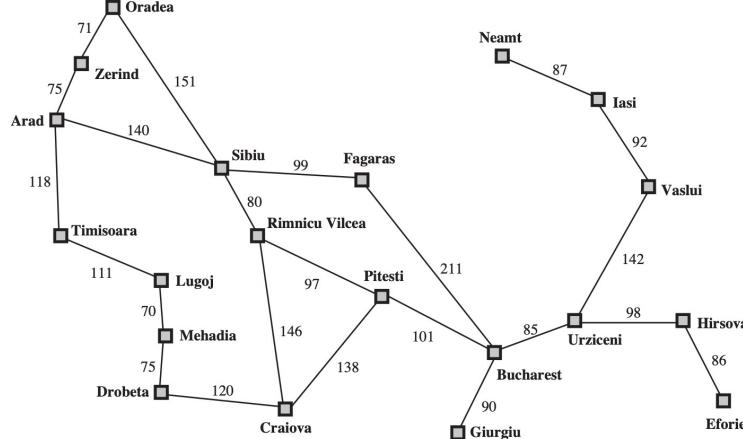


<b>Arad</b>	366	<b>Mehadia</b>	241
<b>Bucharest</b>	0	<b>Neamt</b>	234
<b>Craiova</b>	160	<b>Oradea</b>	380
<b>Drobeta</b>	242	<b>Pitesti</b>	100
<b>Eforie</b>	161	<b>Rimnicu Vilcea</b>	193
<b>Fagaras</b>	176	<b>Sibiu</b>	253
<b>Giurgiu</b>	77	<b>Timisoara</b>	329
<b>Hirsova</b>	151	<b>Urziceni</b>	80
<b>Iasi</b>	226	<b>Vaslui</b>	199
<b>Lugoj</b>	244	<b>Zerind</b>	374

(c) After expanding Sibiu

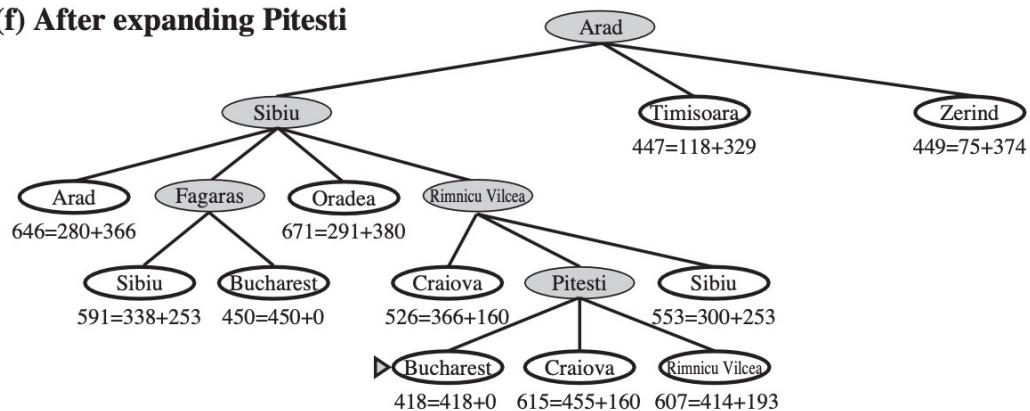


# A\* Search



<b>Arad</b>	366	<b>Mehadia</b>	241
<b>Bucharest</b>	0	<b>Neamt</b>	234
<b>Craiova</b>	160	<b>Oradea</b>	380
<b>Drobeta</b>	242	<b>Pitesti</b>	100
<b>Eforie</b>	161	<b>Rimnicu Vilcea</b>	193
<b>Fagaras</b>	176	<b>Sibiu</b>	253
<b>Giurgiu</b>	77	<b>Timisoara</b>	329
<b>Hirsova</b>	151	<b>Urziceni</b>	80
<b>Iasi</b>	226	<b>Vaslui</b>	199
<b>Lugoj</b>	244	<b>Zerind</b>	374

(f) After expanding Pitesti



# A\* Search - Completeness and Optimality

- The A\* search is **complete** and **optimal** if  $h(n)$  is consistent

# A\* Search - Completeness and Optimality

- The A\* search is **complete** and **optimal** if  $h(n)$  is consistent
- A heuristic is said to be consistent (or monotone), if the estimate is always no greater than the estimated distance from any neighbouring vertex to the goal, plus the cost of reaching that neighbour

$$h(n) \leq \text{cost}(n, n') + h(n')$$

# A\* Search - Time and Space Complexity

- The number of states for the A\* search is **exponential** in the length of the solution, namely for constant step costs:  $O(b^{\epsilon d})$
- When  $h^*$  is the actual cost from root node to goal node,  $\epsilon = \frac{(h^* - h)}{h^*}$  is the relative error

# A\* Search - Time and Space Complexity

- The number of states for the A\* search is **exponential** in the length of the solution, namely for constant step costs:  $O(b^{\epsilon d})$
- When  $h^*$  is the actual cost from root node to goal node,  $\epsilon = \frac{(h^* - h)}{h^*}$  is the relative error
- Space is the main issue with A\*, as it keeps all generated nodes in memory, therefore A\* is not suitable for many large-scale problems

# A\* Search - Summary

Let us summarise the performance of the A\* search algorithm

- **Completeness:** if the heuristic  $h(n)$  is consistent, then the A\* algorithm **is complete**
- **Optimality:** if the heuristic  $h(n)$  is consistent, A\* **is optimal**

# A\* Search - Summary

Let us summarise the performance of the A\* search algorithm

- **Completeness:** if the heuristic  $h(n)$  is consistent, then the A\* algorithm **is complete**
- **Optimality:** if the heuristic  $h(n)$  is consistent, A\* **is optimal**
- **Time complexity:**  $O(b^{\epsilon d})$ , where  $\epsilon$  is the relative error of the heuristic

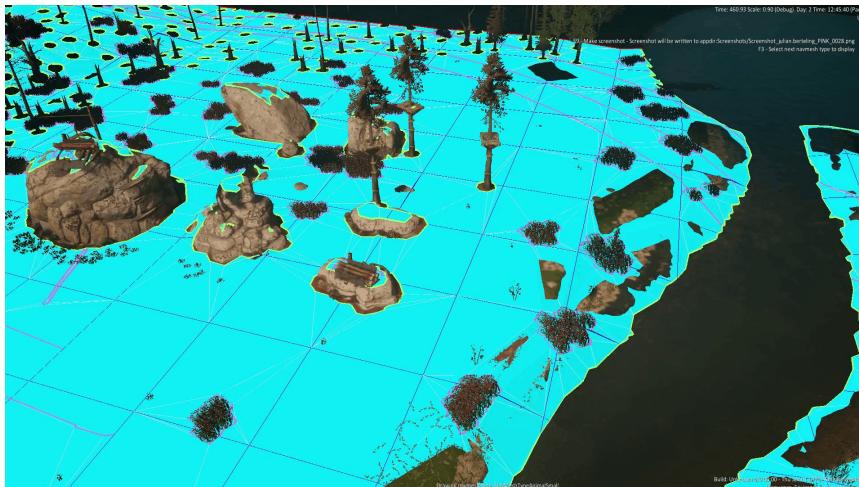
# A\* Search - Summary

Let us summarise the performance of the A\* search algorithm

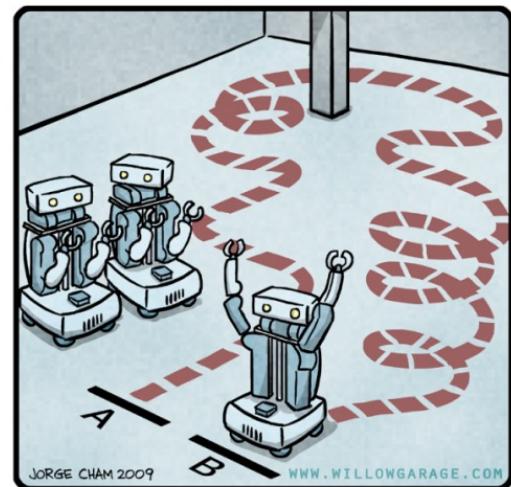
- **Completeness:** if the heuristic  $h(n)$  is consistent, then the A\* algorithm **is complete**
- **Optimality:** if the heuristic  $h(n)$  is consistent, A\* **is optimal**
- **Time complexity:**  $O(b^{\epsilon d})$ , where  $\epsilon$  is the relative error of the heuristic
- **Space complexity:**  $O(b^d)$ , since we keep in memory all expanded nodes and all nodes in the frontier

# A\* - Applications

- A\* has a large number of applications
- In practice, the most common ones are in games and in robotics



R.O.B.O.T. Comics



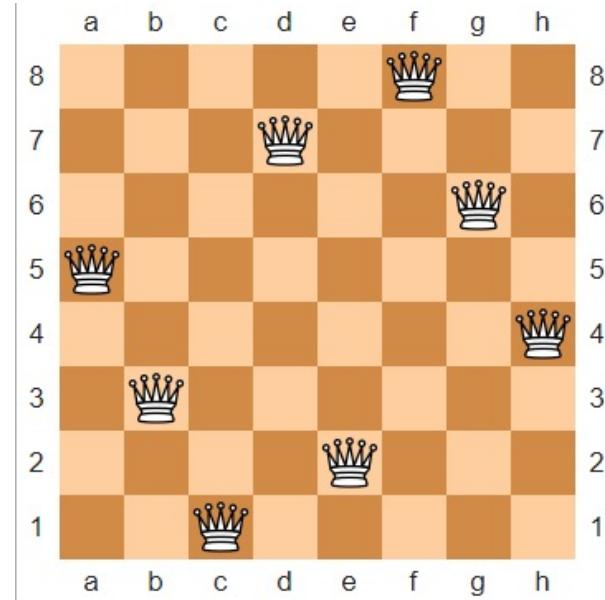
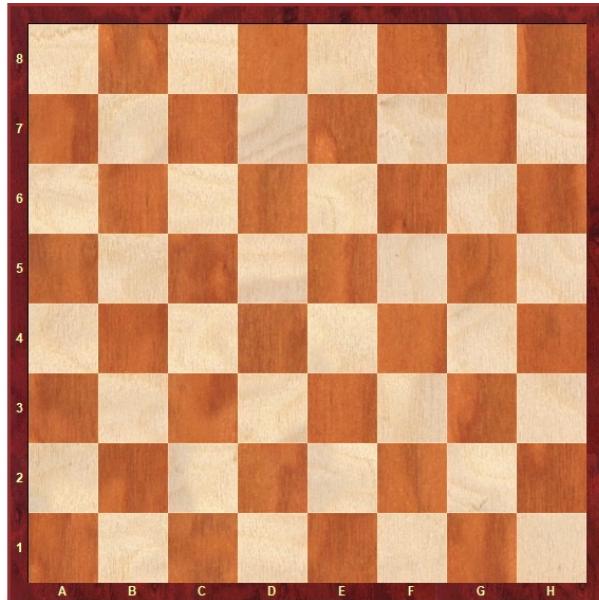
"HIS PATH-PLANNING MAY BE  
SUB-OPTIMAL, BUT IT'S GOT FLAIR."

# Summary

- A\* is complete and optimal, given a consistent heuristic
- However, A\* has typically high time/space complexity, regardless of the heuristic chosen
- Heuristics have a considerable impact on the performance of informed search algorithms, and they can drastically reduce the time and space complexity in comparison to uninformed search algorithms

# In-Class Exercise

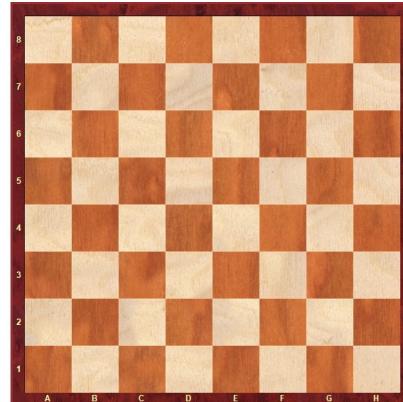
**ACTIVITY:** Consider the 8-queen puzzle (initial state and goal state below)



# In-Class Exercise

**ACTIVITY:** Problem formulation.

- **Initial state:**
- **Actions:**
- **Transition model:**
- **Goal test:**
- **Path cost:**

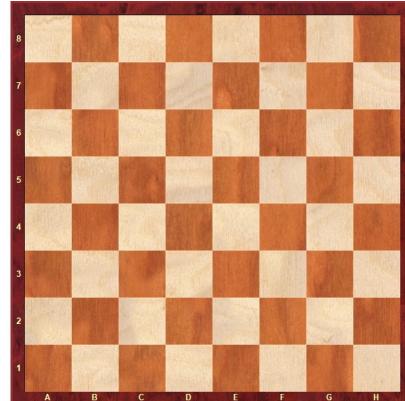


	a	b	c	d	e	f	g	h	
8									8
7									7
6									6
5	👑								5
4									4
3	👑								3
2					👑				2
1						👑			1
	a	b	c	d	e	f	g	h	

# In-Class Exercise

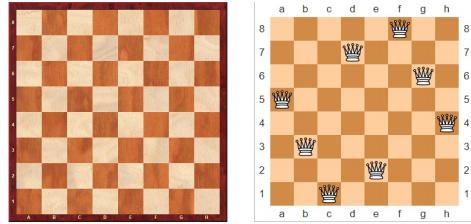
**ACTIVITY:** Problem formulation.

- **Initial state:** empty board
- **Actions:** place a queen on an empty square
- **Transition model:** the corresponding board after taking the action
- **Goal test:** all 8 queens placed on the board; no conflicts
- **Path cost:** each step costs 1



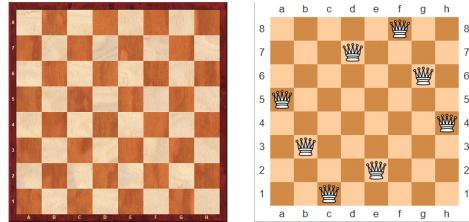
	a	b	c	d	e	f	g	h	
8							♕		8
7					♕				7
6							♕		6
5	♕								5
4								♕	4
3		♕							3
2					♕				2
1						♕			1
	a	b	c	d	e	f	g	h	

# In-Class Exercise



**ACTIVITY:** In small groups, do some research on possible heuristic functions that can be used for an implementation of A\* (10 minutes) and evaluate their performance in terms of speed to calculate and quality of the result

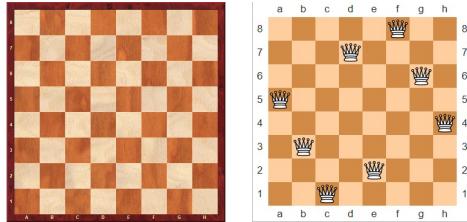
# In-Class Exercise



**ACTIVITY:** In small groups, do some research on possible heuristic functions that can be used for an implementation of A\* (10 minutes) and evaluate their performance in terms of speed to calculate and quality of the result

- **H1:** Different column (row) from previously placed queen(s)
  - Quick, but not very informed

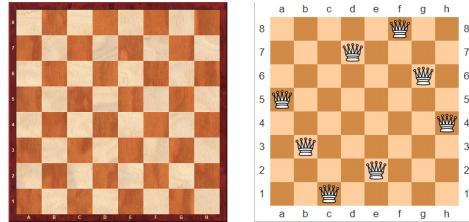
# In-Class Exercise



**ACTIVITY:** In small groups, do some research on possible heuristic functions that can be used for an implementation of A\* (10 minutes) and evaluate their performance in terms of speed to calculate and quality of the result

- **H1:** Different column (row) from previously placed queen(s)
  - Quick, but not very informed
- **H2:** Distance from previously placed queen
  - Quick, but not very informed

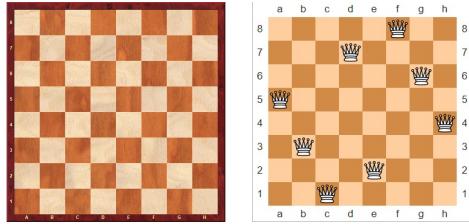
# In-Class Exercise



**ACTIVITY:** In small groups, do some research on possible heuristic functions that can be used for an implementation of A\* (10 minutes) and evaluate their performance in terms of speed to calculate and quality of the result

- **H1:** Different column (row) from previously placed queen(s)
  - Quick, but not very informed
- **H2:** Distance from previously placed queen
  - Quick, but not very informed
- **H3:** Number of future feasible slots
  - Slower than previous ones, but more informed

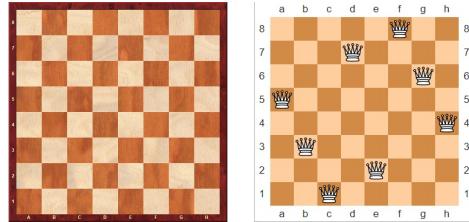
# In-Class Exercise



**ACTIVITY:** In small groups, do some research on possible heuristic functions that can be used for an implementation of A\* (10 minutes) and evaluate their performance in terms of speed to calculate and quality of the result

- **H4:** Mean distance between previously placed queens
  - Mean of H2, a reasonable compromise between speed and quality

# In-Class Exercise

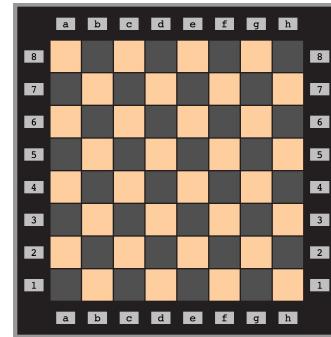


**ACTIVITY:** In small groups, do some research on possible heuristic functions that can be used for an implementation of A\* (10 minutes) and evaluate their performance in terms of speed to calculate and quality of the result

- **H4:** Mean distance between previously placed queens
  - Mean of H2, a reasonable compromise between speed and quality
- **H5:** More...

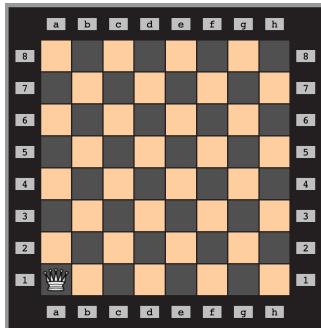
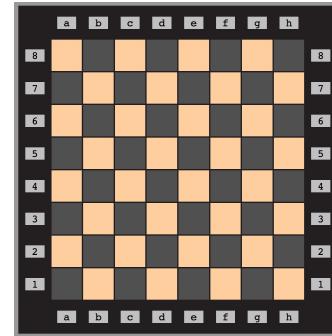
# In-Class Exercise

- **H1:** Different row from previously placed queens



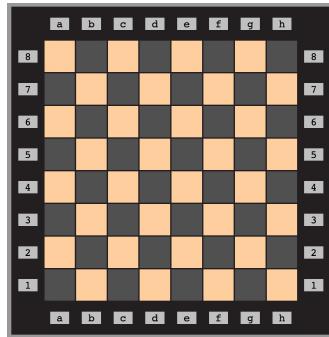
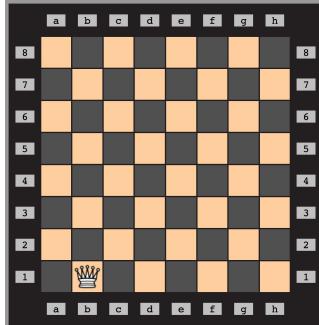
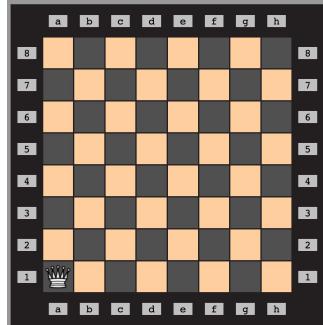
# In-Class Exercise

- H1: Different row from previously placed queens

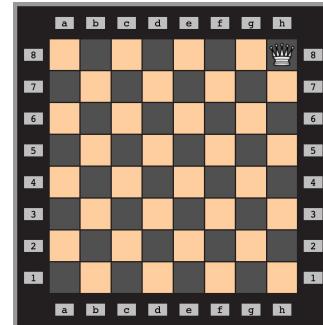


# In-Class Exercise

- H1: Different row from previously placed queens



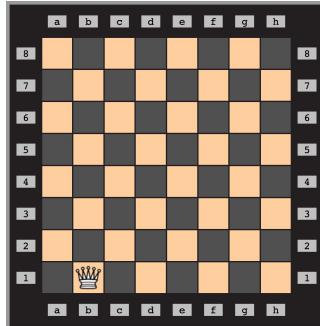
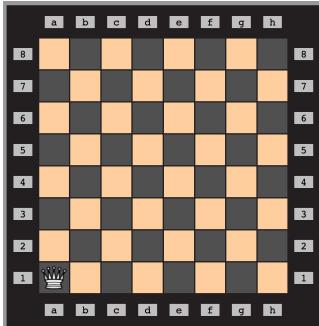
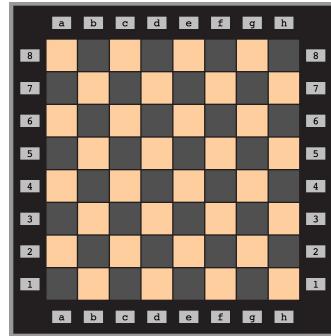
...



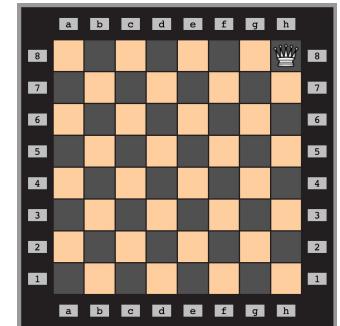
# In-Class Exercise

- H1: Different row from previously placed queens

$$f(n) = 1 + 0 = 1$$



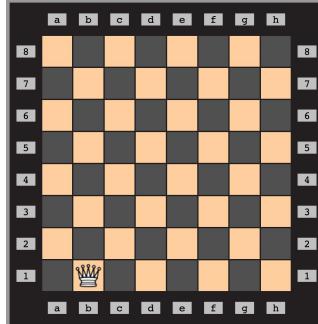
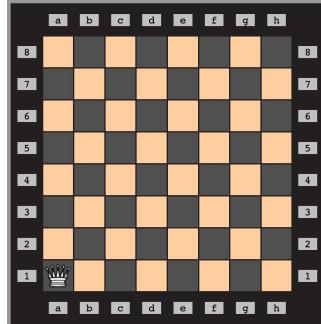
...



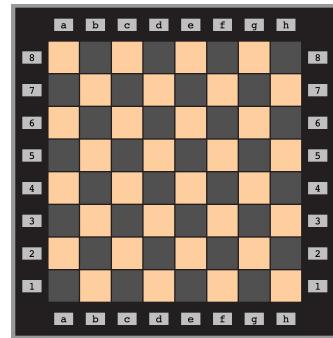
# In-Class Exercise

- H1: Different row from previously placed queens

$$f(n) = 1 + 0 = 1$$

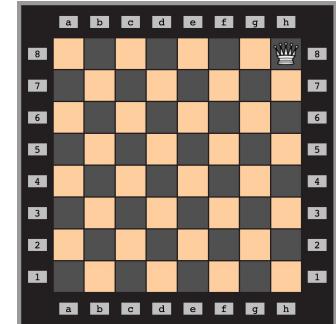


$$f(n) = 1 + 0 = 1$$

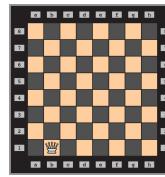
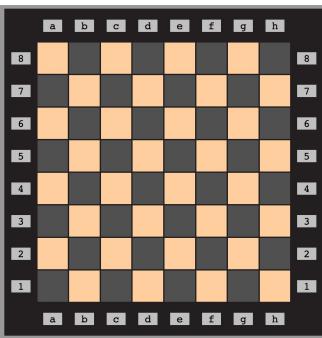
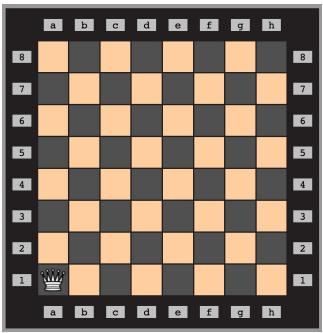


$$f(n) = 1 + 0 = 1$$

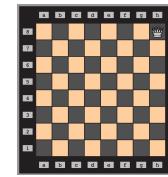
...



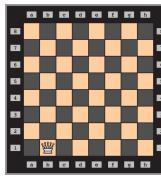
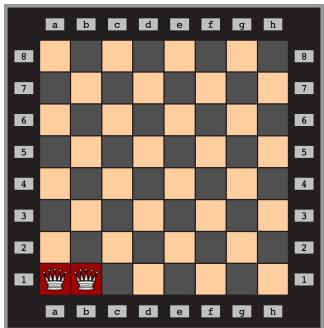
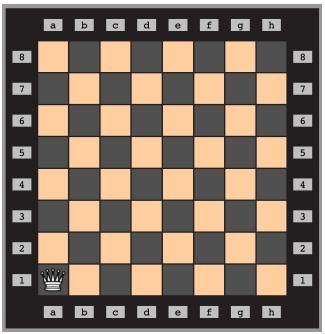
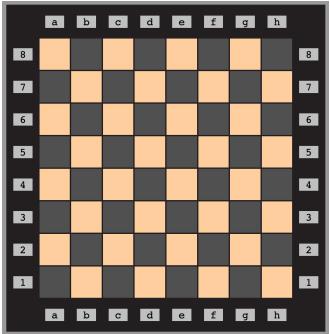
$$f(n) = 1 + 0 = 1$$



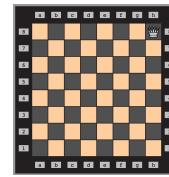
...



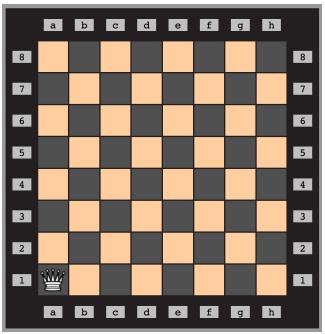
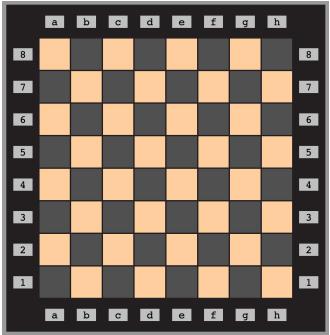
$$f(n) = 1 + 0 = 1$$



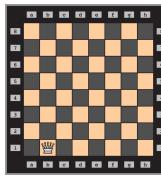
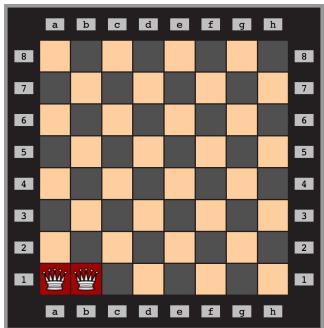
...



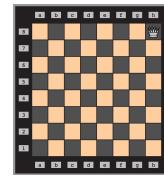
$$f(n) = 1 + 0 = 1$$



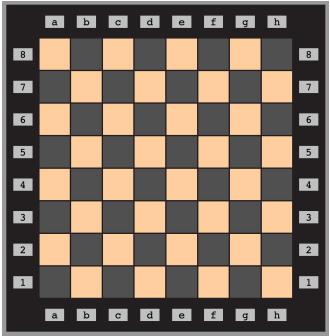
$$f(n) = 2 + 1 = 3$$



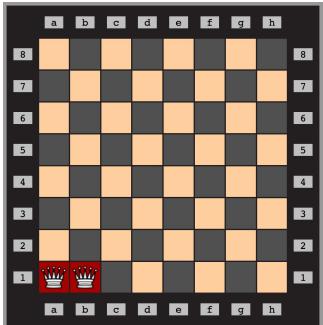
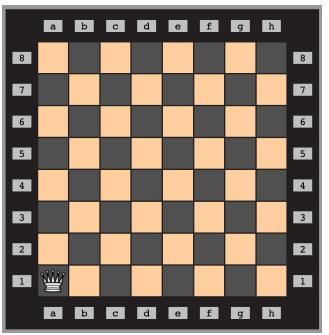
...



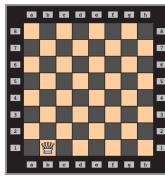
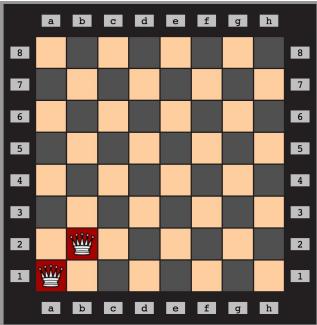
$$f(n) = 1 + 0 = 1$$



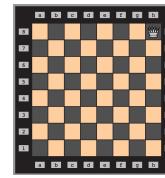
$$f(n) = 2 + 1 = 3$$



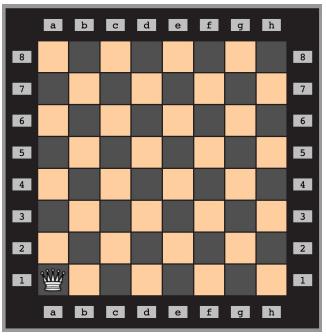
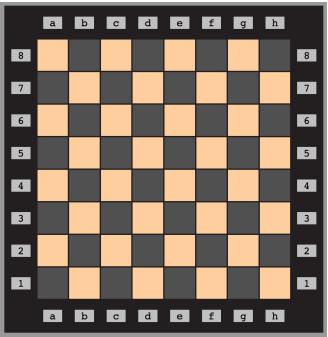
...



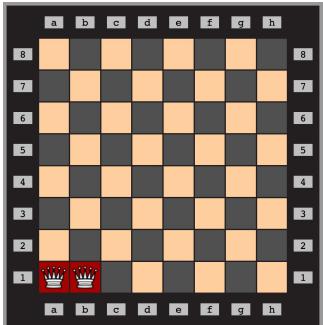
...



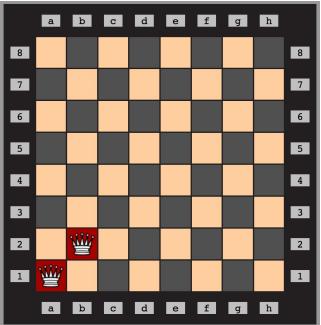
$$f(n) = 1 + 0 = 1$$



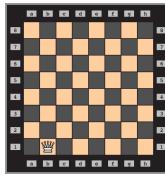
$$f(n) = 2 + 1 = 3$$



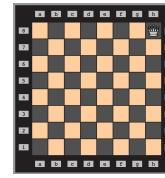
$$f(n) = 2 + 0 = 2$$



...

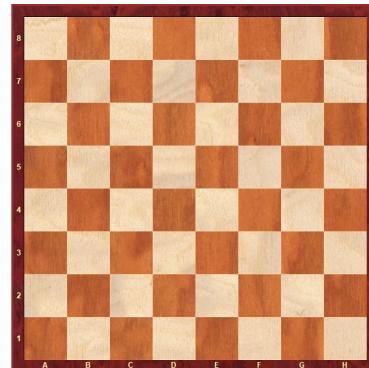


...



# In-Class Exercise: Final Remarks

- Finding a good heuristic is not easy, many candidate heuristics can exist
- Different heuristics lead to different results in terms of performance
- Trade-off between computational expense and level of information (path cost in general)
- <https://datagenetics.com/blog/august42012/index.html>



	a	b	c	d	e	f	g	h	
8									8
7									7
6									6
5	♛								5
4		♛							4
3			♛						3
2				♛					2
1					♛				1
	a	b	c	d	e	f	g	h	

# Aims of the Session

You should now be able to:

- Describe the difference between uninformed and informed search
- Understand the concept of a heuristic function in informed search
- Analyse the performance of A\* and apply the algorithm to solve search problems

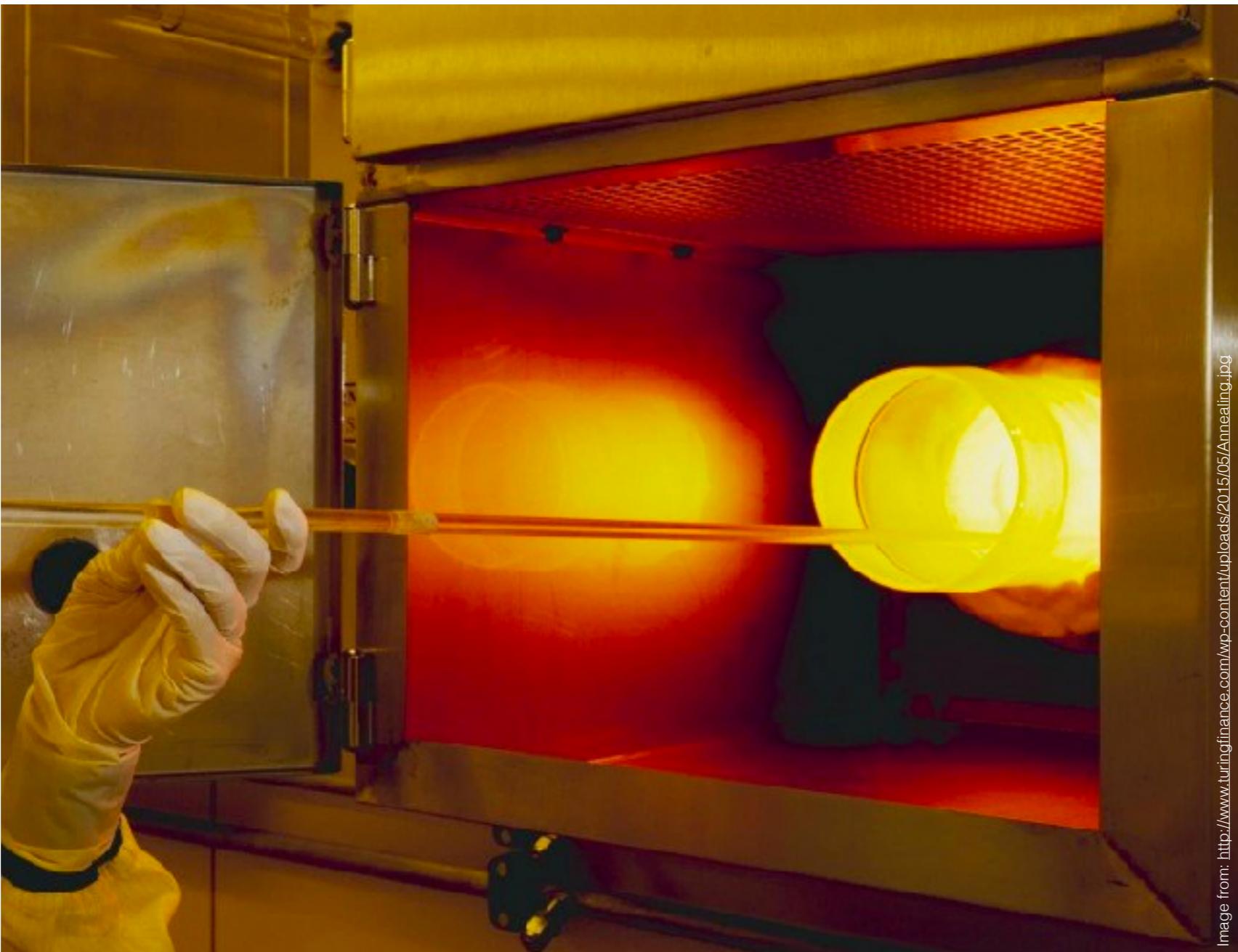


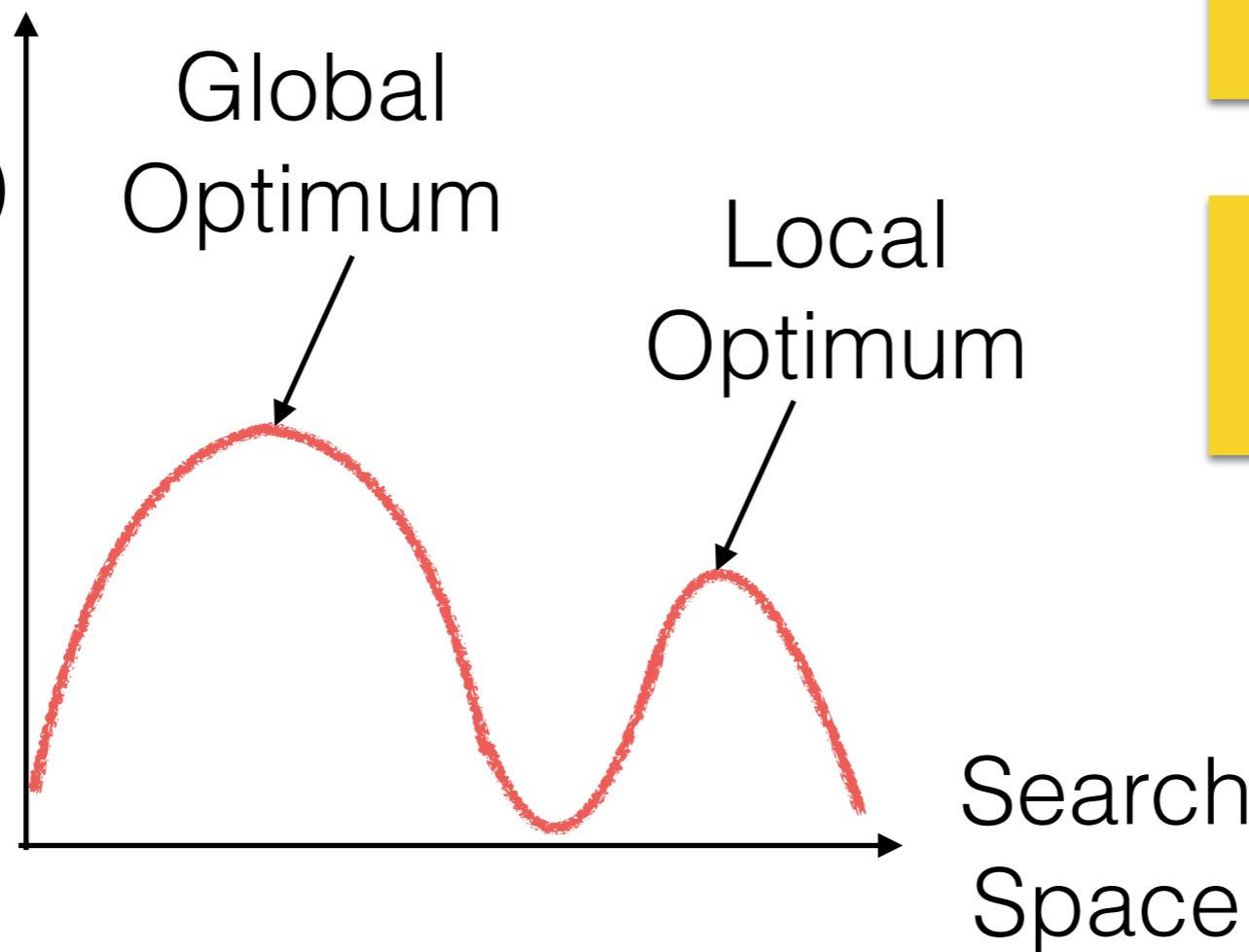
Image from: <http://www.turingfinance.com/wp-content/uploads/2015/05/Annealing.jpg>

# Simulated Annealing - Part 1

Leandro L. Minku

# Motivation

Objective  
Function  
(to be  
maximised)

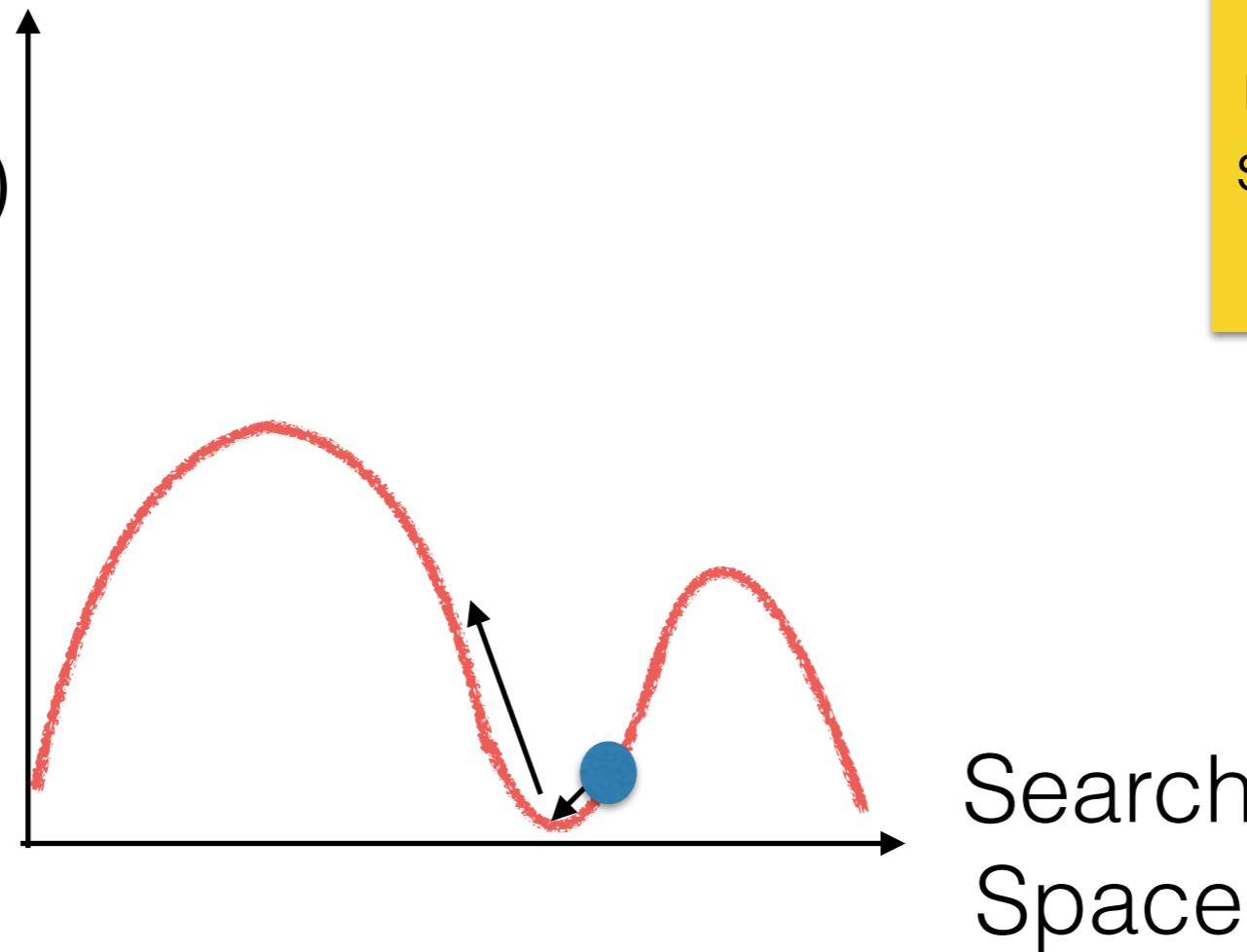


Hill-climbing may get trapped in a local optimum.

Heuristic = informed guess

# Motivation

Objective  
Function  
(to be  
maximised)



If we could sometimes accept a downward move, we would have some chance to move to another hill.

# Hill-Climbing

Hill-Climbing (assuming maximisation)

1. `current_solution` = generate initial solution randomly
2. Repeat:
  - 2.1 generate neighbour solutions (differ from current solution by a single element)
  - 2.2 `best_neighbour` = get highest quality neighbour of `current_solution`
  - 2.3 If  $\text{quality}(\text{best\_neighbour}) \leq \text{quality}(\text{current\_solution})$ 
    - 2.3.1 Return `current_solution`
  - 2.4 `current_solution` = `best_neighbour`

Until a maximum number of iterations

In simulated annealing, instead of taking the best neighbour, we pick a random neighbour.

# Hill-Climbing

Hill-Climbing (assuming maximisation)

1. `current_solution` = generate initial solution randomly
2. Repeat:
  - 2.1 generate neighbour solutions (differ from current solution by a single element)
  - 2.2 `best_neighbour` = get highest quality neighbour of `current_solution`
  - 2.3 If  $\text{quality}(\text{best\_neighbour}) \leq \text{quality}(\text{current\_solution})$ 
    - 2.3.1 Return `current_solution`
  - 2.4 `current_solution` = `best_neighbour`

Until a maximum number of iterations

Simulated annealing will give some chance to accept a bad neighbour.

# Simulated Annealing

Simulated Annealing (assuming maximisation)

1. current\_solution = generate initial solution randomly
2. Repeat:
  - 2.1 generate neighbour solutions (differ from current solution by a single element)
  - 2.2 rand\_neighbour = get random neighbour of current\_solution
  - 2.3 If quality(rand\_neighbour) <= quality(current\_solution) {
    - 2.3.1 With some probability,  
current\_solution = rand\_neighbour
  - } Else current\_solution = rand\_neighbour
- Until a maximum number of iterations

# Simulated Annealing

Simulated Annealing (assuming maximisation)

1. current\_solution = generate initial solution randomly
2. Repeat:
  - 2.1 ~~generate neighbour solutions (differ from current solution by a single element)~~
  - 2.2 rand\_neighbour = get random neighbour of current\_solution
  - 2.3 If quality(rand\_neighbour) <= quality(current\_solution) {
    - 2.3.1 With some probability,  
current\_solution = rand\_neighbour
  - } Else current\_solution = rand\_neighbour
- Until a maximum number of iterations

# Simulated Annealing

Simulated Annealing (assuming maximisation)

1. current\_solution = generate initial solution randomly

2. Repeat:

    2.1 rand\_neighbour = generate random neighbour of current\_solution

    2.2 If quality(rand\_neighbour) <= quality(current\_solution) {

        2.2.1 With some probability,

            current\_solution = rand\_neighbour

    } Else current\_solution = rand\_neighbour

Until a maximum number of iterations

# Simulated Annealing

Simulated Annealing (assuming maximisation)

1. current\_solution = generate initial solution randomly

2. Repeat:

    2.1 rand\_neighbour = generate random neighbour of current\_solution

    2.2 If quality(rand\_neighbour) <= quality(current\_solution) {

**2.2.1 With some probability,**

            current\_solution = rand\_neighbour

    } Else current\_solution = rand\_neighbour

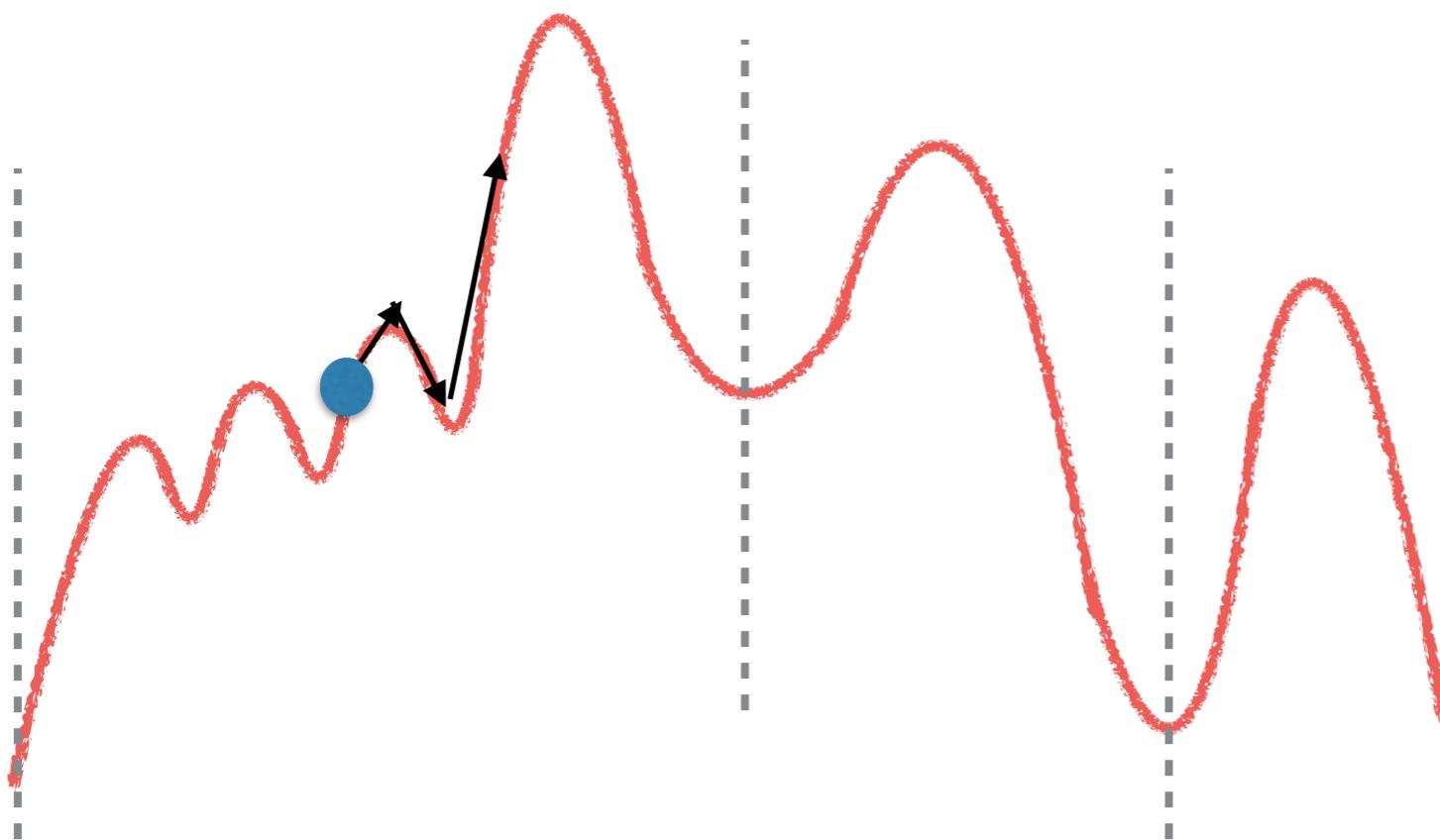
Until a maximum number of iterations

# How Should the Probability be Set?

- Probability to accept solutions with much worse quality should be lower.
  - We don't want to be dislodged from the optimum.
- High probability in the beginning.
  - More similar effect to random search.
  - Allows us to **explore** the search space.
- Lower probability as time goes by.
  - More similar effect to hill-climbing.
  - Allows us to **exploit** a hill.

# How to Decrease the Probability?

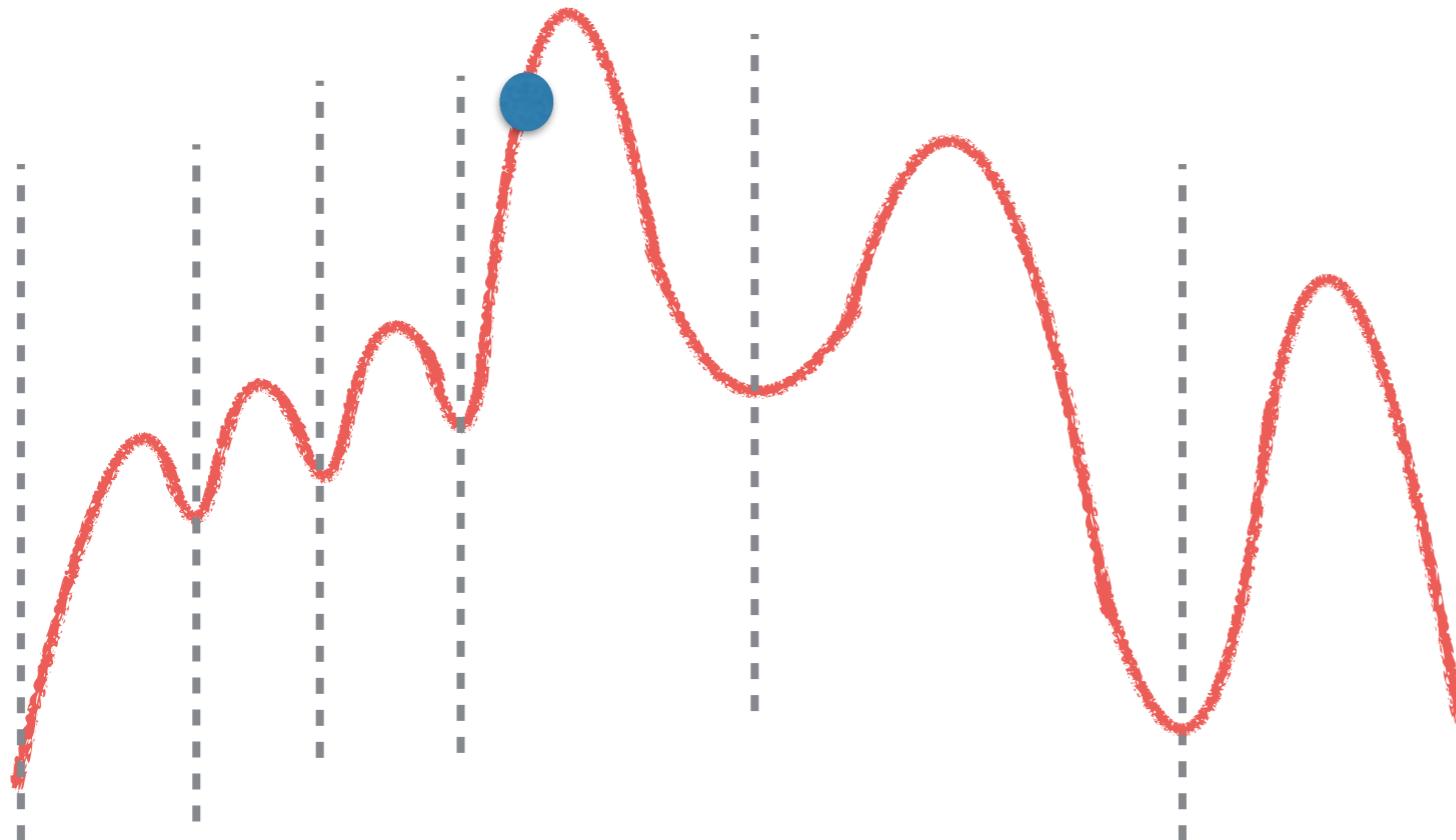
- We would like to decrease the probability slowly.



If you decrease the probability slowly, you start to form basis of attraction, but you can still walk over small hills initially.

# How to Decrease the Probability?

- We would like to decrease the probability slowly.



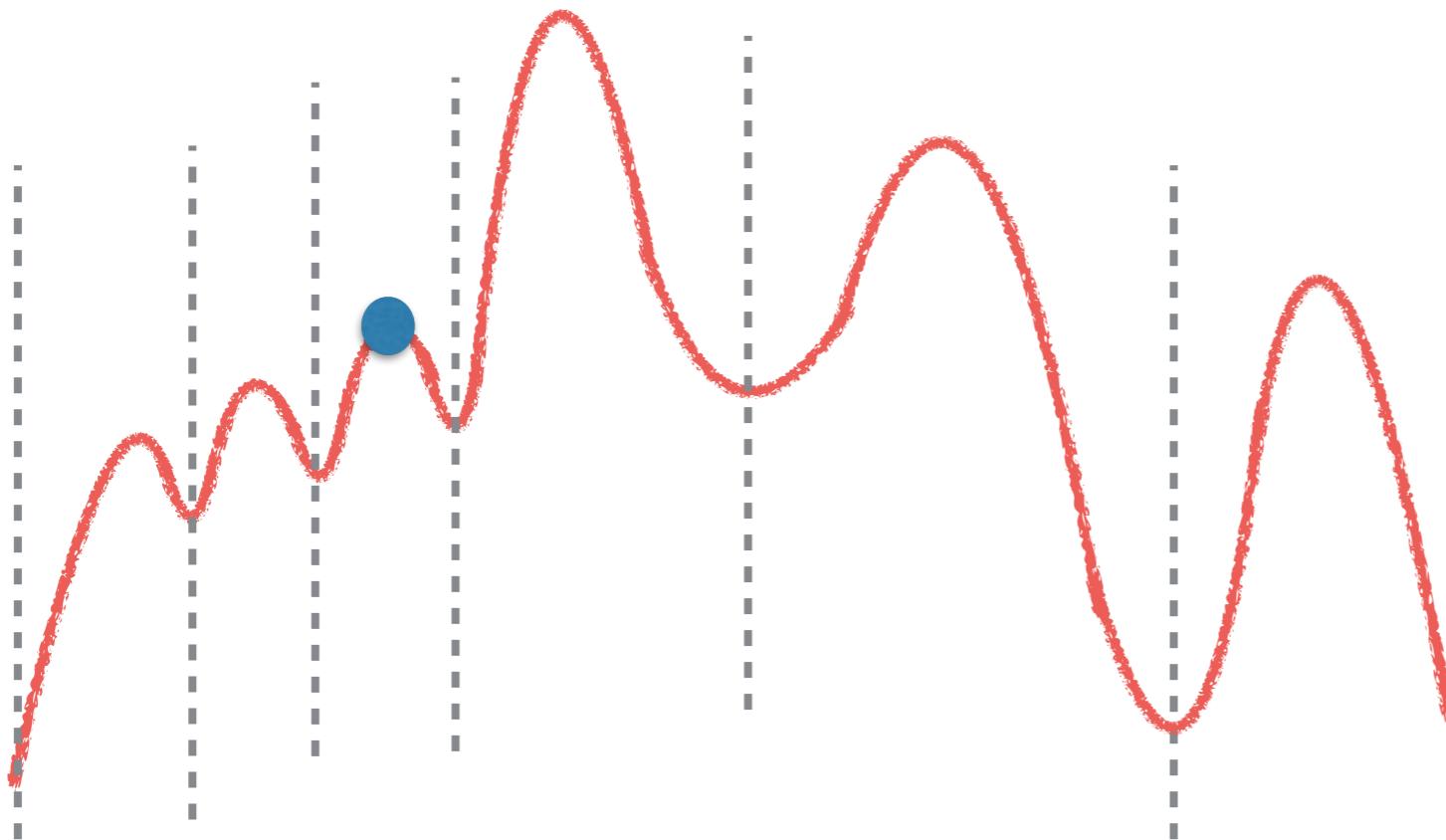
As the probability decreases further, the small hills start to form basis of attraction too.

But if you do so slowly enough, you give time to wander to the higher value hills before starting to exploit.

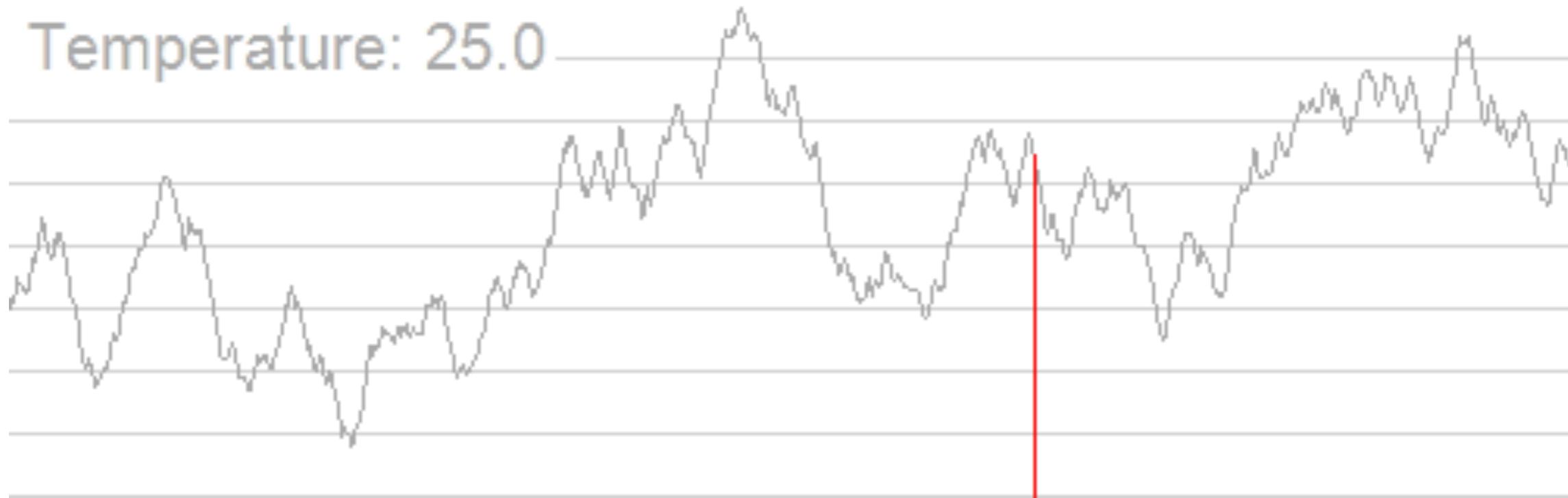
So, you can find the global optimum!

# How to Decrease the Probability?

- We would like to decrease the probability slowly.



If you decrease too quickly, you can get trapped in local optima.

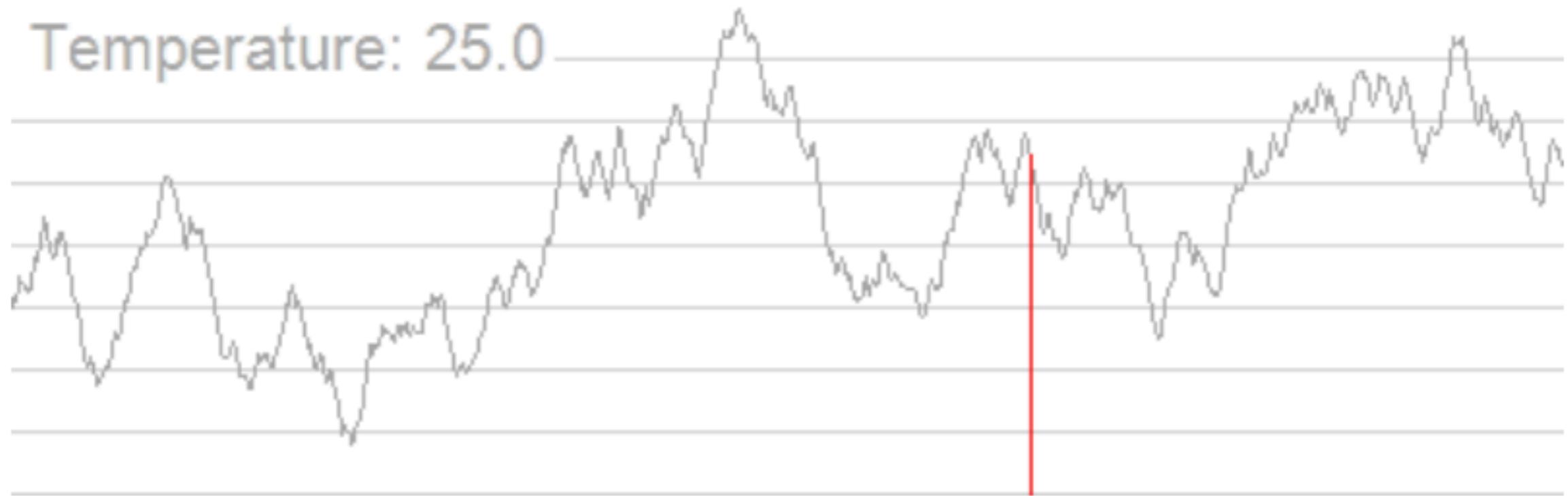


[By Kingpin13 - Own work, CC0, <https://commons.wikimedia.org/w/index.php?curid=25010763>]

**Note 1:** the line is not jumping to non-neighbouring positions, it's just moving very fast!

**Note 2:** in this example we have only 2 neighbours for each solution, but in real world problems we may have many neighbours for each solution.

**Note 3:** real world problems may have much more dimensions.



[By Kingpin13 - Own work, CC0, <https://commons.wikimedia.org/w/index.php?curid=25010763>]

**Note 4:** it may be very difficult to visualise the quality function in real world problems.

**Note 5:** the algorithm itself does not know the shape of the function beforehand.

# Simulated Annealing

Simulated Annealing (assuming maximisation)

1. current\_solution = generate initial solution randomly
2. Repeat:
  - 2.1 rand\_neighbour = generate random neighbour of current\_solution
  - 2.2 If quality(rand\_neighbour) <= quality(current\_solution) {
    - 2.2.1 With some probability,**  
current\_solution = rand\_neighbour
    - } Else current\_solution = rand\_neighbour
  - 2.3 Reduce probability**
- Until a maximum number of iterations

# Summary

- Simulated annealing gives some chance to accept a neighbour that has equal or worse quality than the current solution, in order to avoid getting trapped in local optima.
- The probability of accepting such bad neighbours should reduce over time.
- The worse the quality of this neighbour, the smaller the probability of accepting it should be.

# Next

- How to calculate this probability?
- How to reduce it over time?

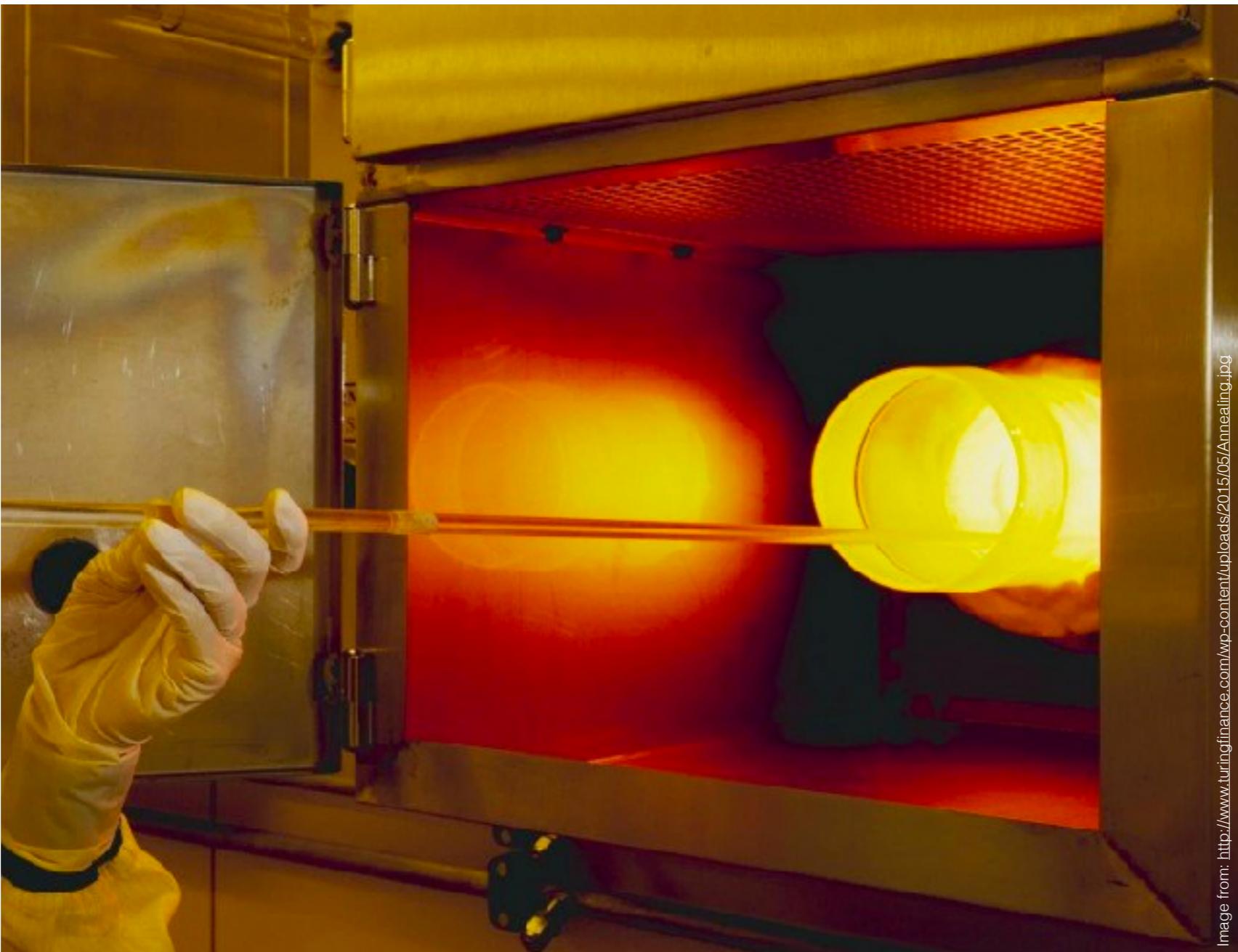


Image from: <http://www.turingfinance.com/wp-content/uploads/2015/05/Annealing.jpg>

# Simulated Annealing — Part 2

Leandro L. Minku

# Simulated Annealing

Simulated Annealing (assuming maximisation)

1. current\_solution = generate initial solution randomly
2. Repeat:
  - 2.1 rand\_neighbour = generate random neighbour of current\_solution
  - 2.2 If quality(rand\_neighbour) <= quality(current\_solution) {
    - 2.2.1 With some probability,**  
current\_solution = rand\_neighbour
    - } Else current\_solution = rand\_neighbour
  - 2.3 Reduce probability**
- Until a maximum number of iterations

# Metallurgy Annealing

- A blacksmith heats the metal to a very high temperature.
- When heated, the steel's atoms can move fast and randomly.



Image from: <http://www.stormthecastle.com/indeximages/sting-steel-thumb.jpg>

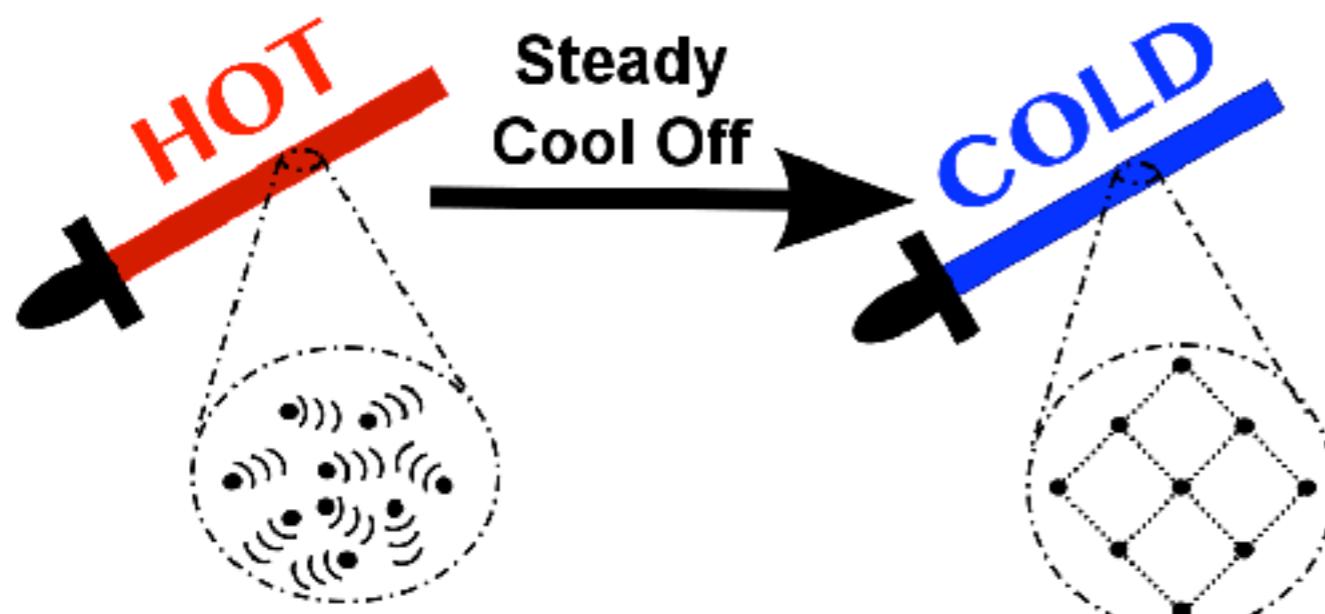


Image from: [http://2.bp.blogspot.com/-kOlrodykkg/UbfVZO\\_l5HI/AAAAAAAJJ4/0rQ98g6tDDA/s1600/annealingAtoms.png](http://2.bp.blogspot.com/-kOlrodykkg/UbfVZO_l5HI/AAAAAAAJJ4/0rQ98g6tDDA/s1600/annealingAtoms.png)

- The blacksmith then lets it cool down slowly.
- If cooled down at the right speed, the atoms will settle in nicely.
- This makes the sword stronger than the untreated steel.



# Probability Function

Probability of accepting a solution of equal or worse quality,  
inspired by thermodynamics:

$$e^{\Delta E/T}$$

$$\Delta E = \text{quality}(\text{rand\_neighbour}) - \text{quality}(\text{current\_solution})$$

( $\leq 0$ )

Assuming maximisation...

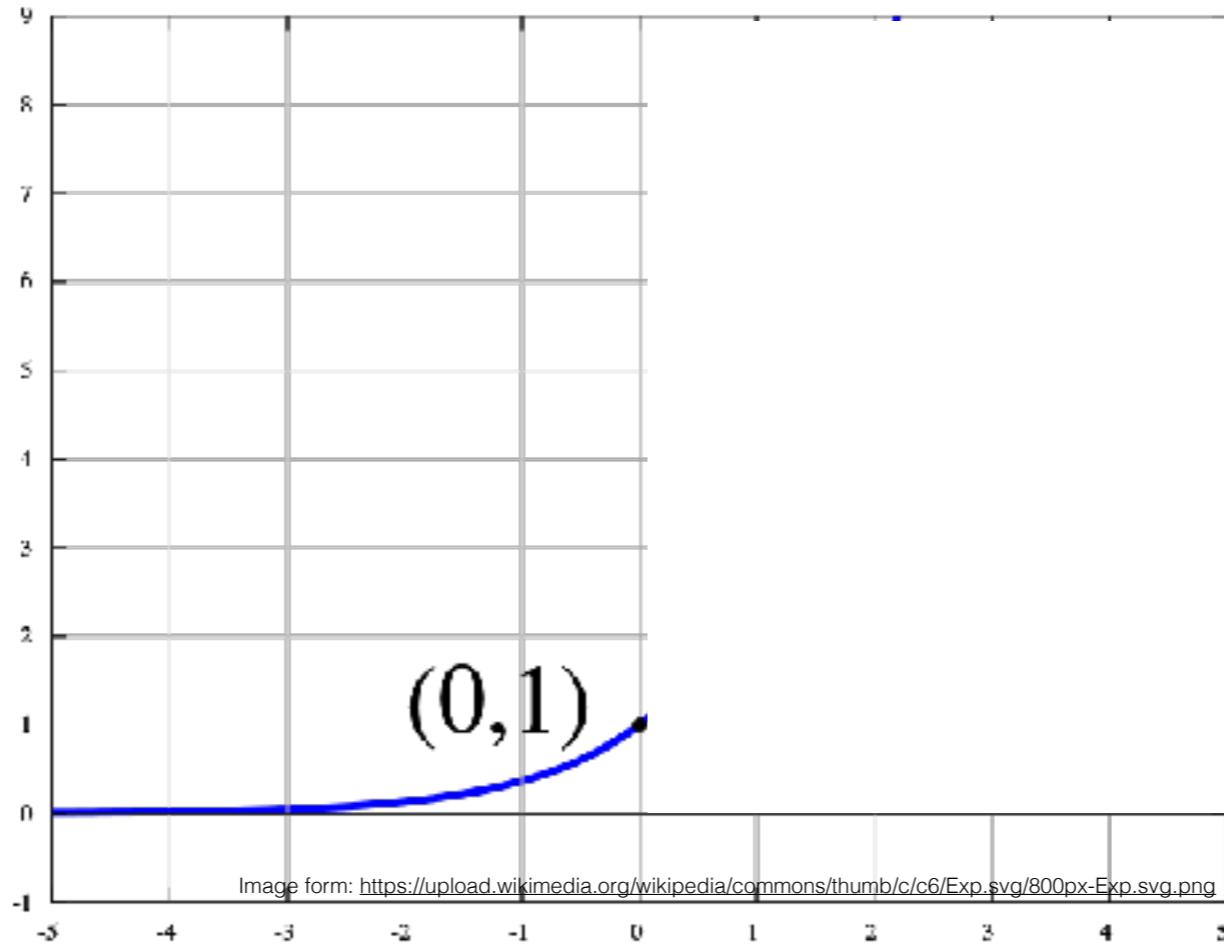
$$T = \text{temperature}$$

( $> 0$ )

$$e = 2.71828\dots$$

# Exponential Function

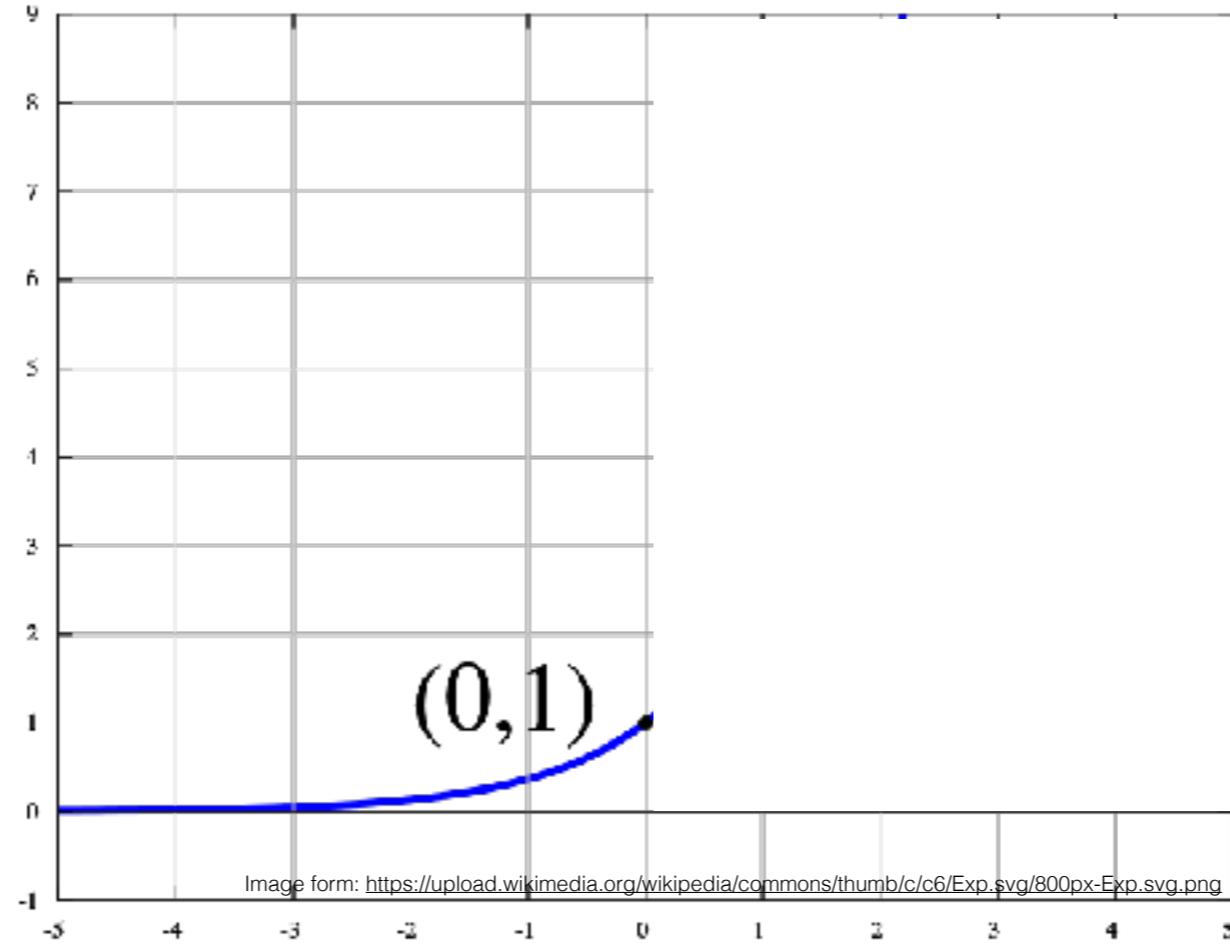
$$e^{\Delta E/T}$$



# Exponential Function

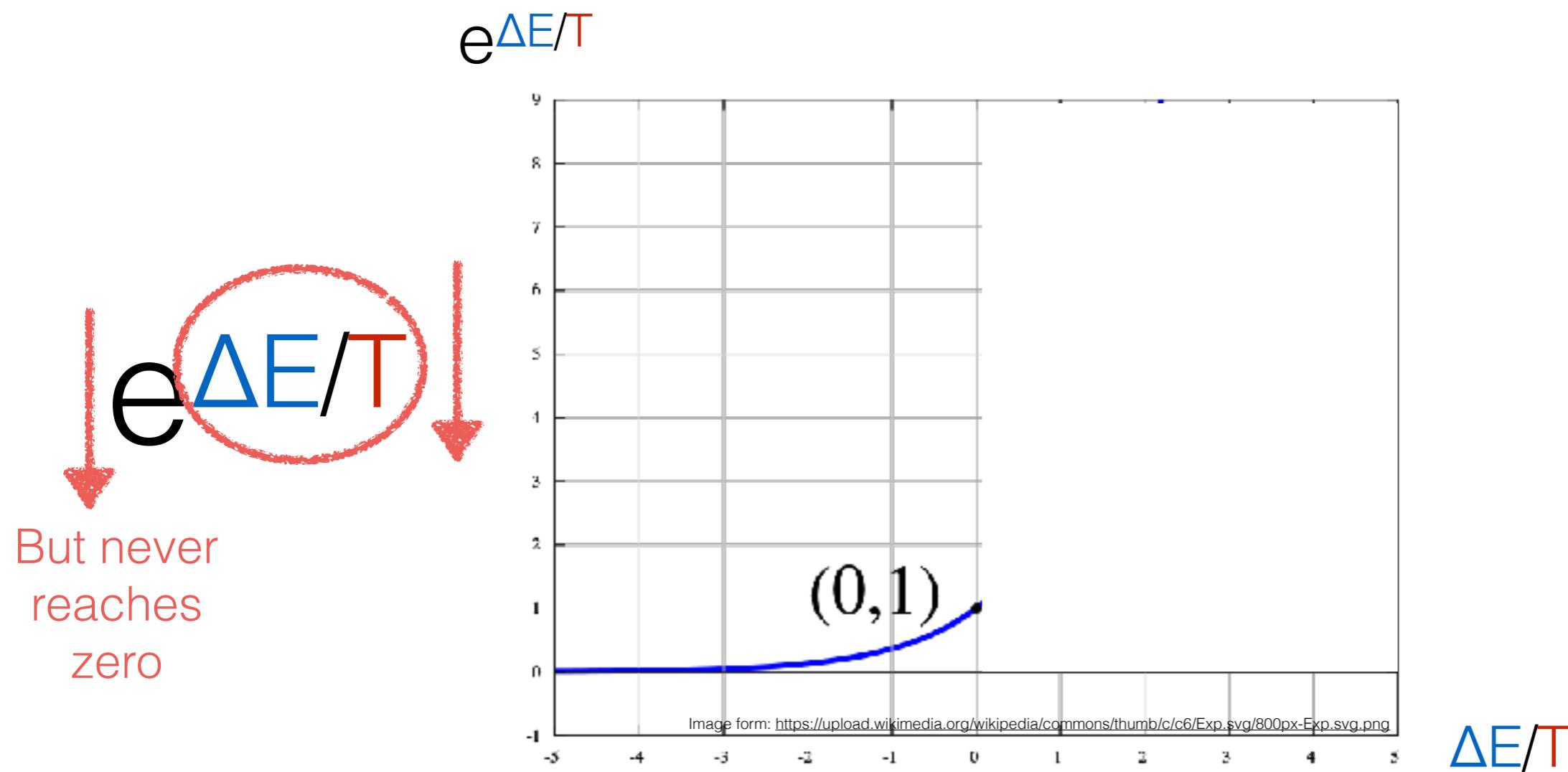
$$e^{\Delta E/T}$$

$e^{\Delta E/T}$



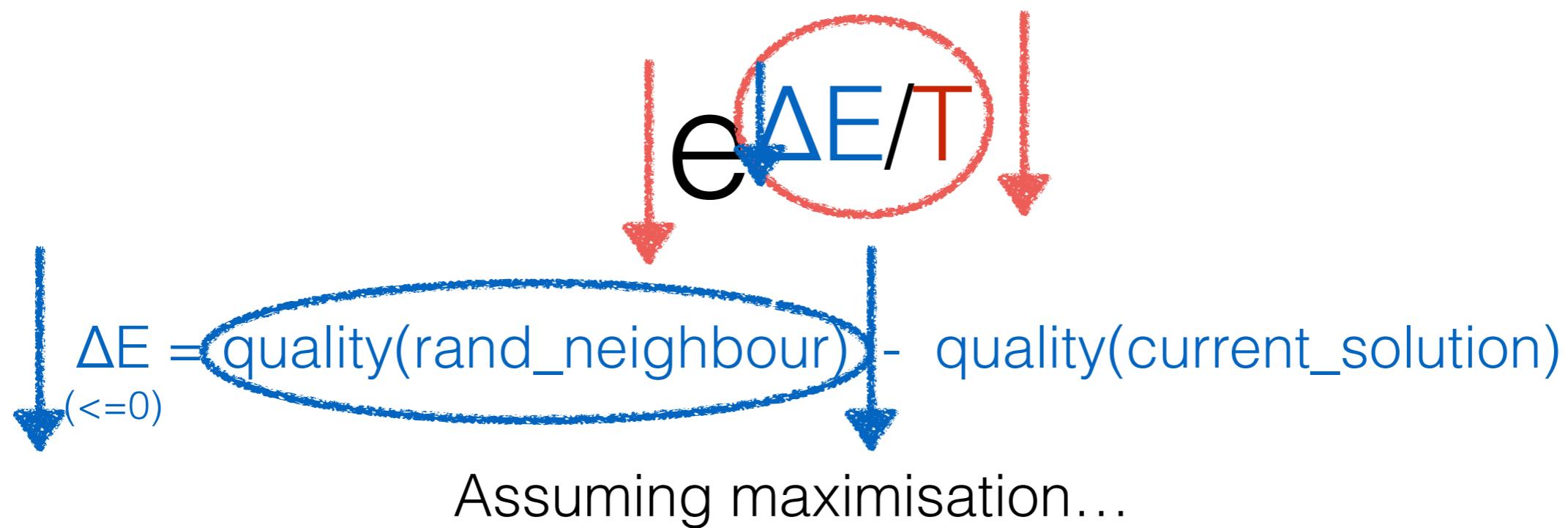
$$\Delta E/T$$

# Exponential Function



# How Does $\Delta E$ Affect the Probability?

Probability of accepting a solution of equal or worse quality:

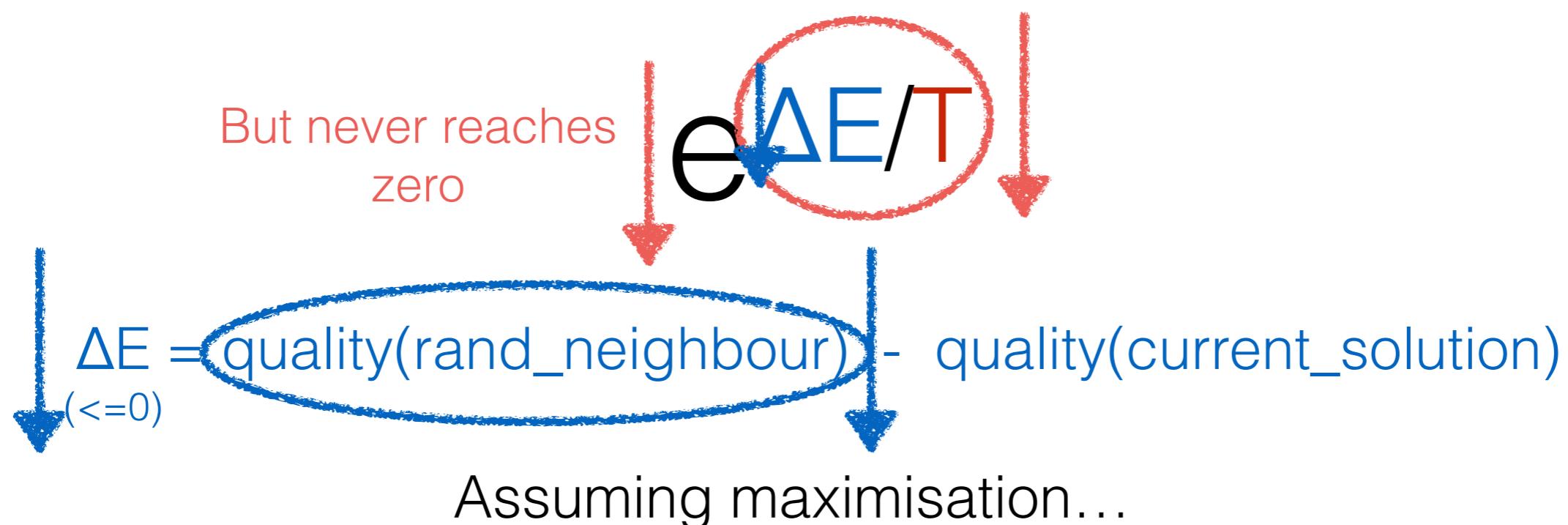


$T = \text{temperature}$   
( $> 0$ )

The worse the neighbour is in comparison to the current solution,  
the less likely to accept it.

# How Does $\Delta E$ Affect the Probability?

Probability of accepting a solution of equal or worse quality:

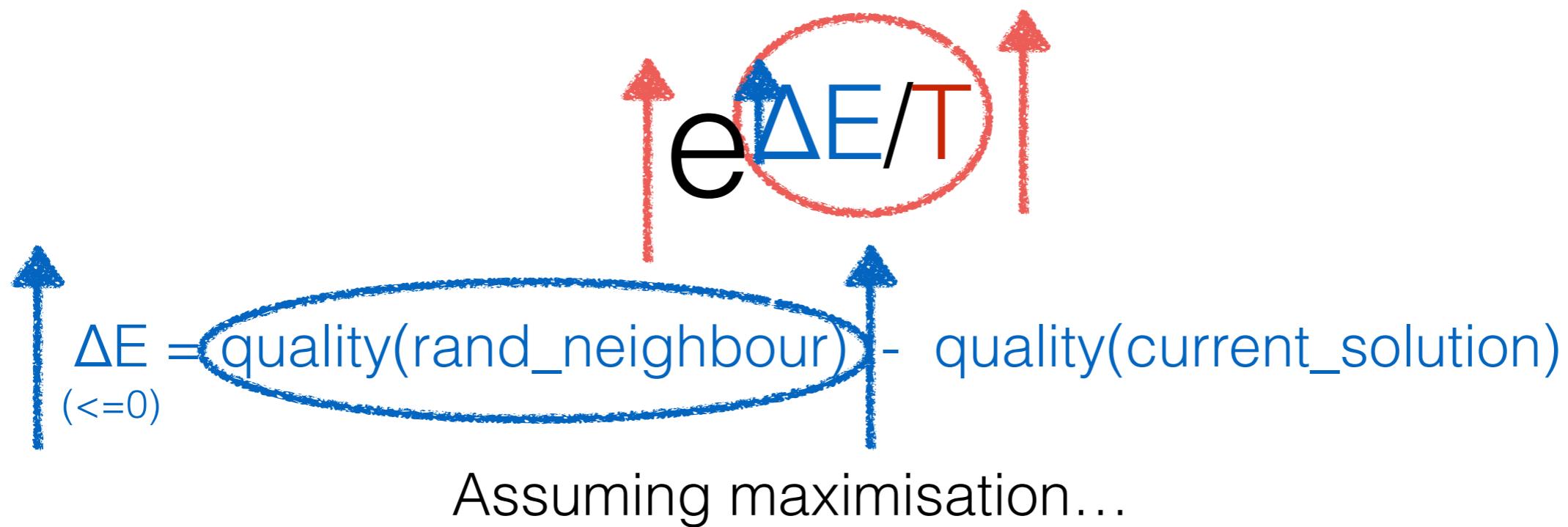


$T = \text{temperature}$   
( $> 0$ )

We always have some probability to accept a bad neighbour,  
no matter how bad it is.

# How Does $\Delta E$ Affect the Probability?

Probability of accepting a solution of equal or worse quality:



$T = \text{temperature}$   
( $>0$ )

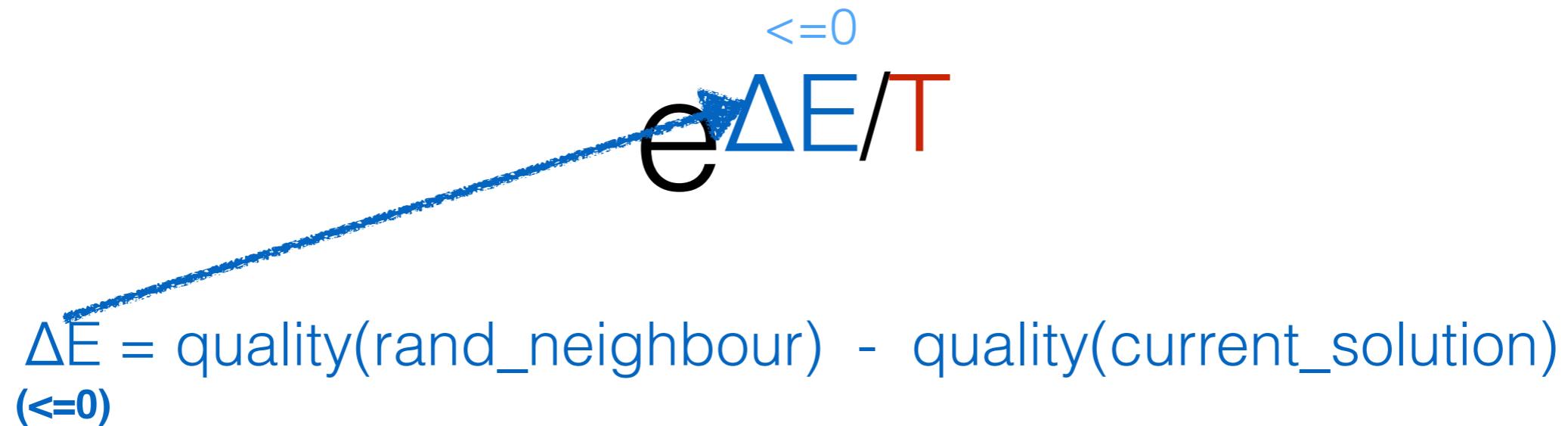
The better the neighbour is, the more likely to accept it.

# How Should the Probability be Set?

- **Probability to accept solutions with much worse quality should be lower.**
  - We don't want to be dislodged from the optimum.
- High probability in the beginning.
  - More similar effect to random search.
  - Allows us to explore the search space.
- Lower probability as time goes by.
  - More similar effect to hill-climbing.
  - Allows us to exploit a hill.

# How Does T Affect the Probability?

Probability of accepting a solution of **equal or worse quality**:

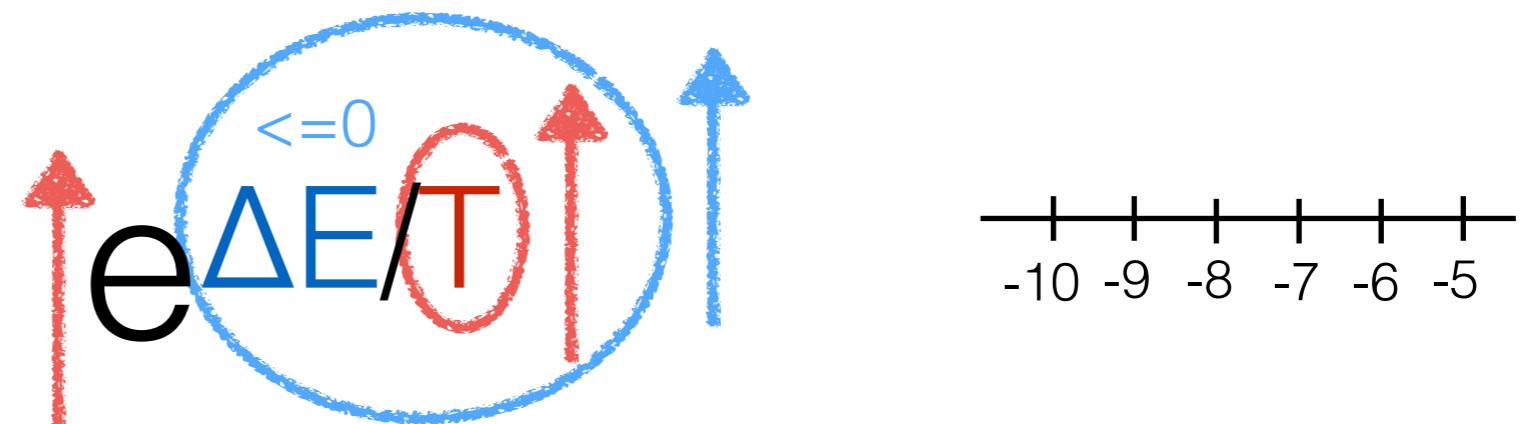


Assuming maximisation...

T = temperature  
(>0)

# How Does T Affect the Probability?

Probability of accepting a solution of equal or worse quality:



$$\Delta E = \text{quality}(\text{rand\_neighbour}) - \text{quality}(\text{current\_solution})$$

( $<=0$ )

Assuming maximisation...

T = temperature  
( $>0$ )

If T is higher, the probability of accepting the neighbour is higher.

# How Does T Affect the Probability?

Probability of accepting a solution of equal or worse quality:

$$e^{\Delta E / T}$$

A hand-drawn style diagram shows the expression  $e^{\Delta E / T}$  inside a blue circle. Above the circle, the condition  $\Delta E \leq 0$  is written in blue. Red arrows point from the left and right sides of the circle towards the center, indicating the magnitude of  $\Delta E$ .

$$\Delta E = \text{quality}(\text{rand\_neighbour}) - \text{quality}(\text{current\_solution})$$

$(\leq 0)$

Assuming maximisation...

T = temperature  
 $(> 0)$

If T is lower, the probability of accepting the neighbour is lower.

# How Does T Affect the Probability?

Probability of accepting a solution of equal or worse quality:

$$e^{\Delta E / T}$$

A hand-drawn style diagram shows the expression  $e^{\Delta E / T}$  inside a blue circle. Above the circle, the condition  $\Delta E \leq 0$  is written in blue. Red arrows point from the left and right sides of the circle towards the center, indicating the magnitude of  $\Delta E$ .

$$\Delta E = \text{quality}(\text{rand\_neighbour}) - \text{quality}(\text{current\_solution})$$

$(\leq 0)$

Assuming maximisation...

T = temperature  
 $(> 0)$

So, reducing the temperature over time would reduce the probability of accepting the neighbour.

# How Should the Temperature be Set?

- High probability in the beginning.
  - More similar effect to random search.
  - Allows us to **explore** the search space.
- Lower probability as time goes by.
  - More similar effect to hill-climbing.
  - Allows us to **exploit** a hill.

T should start high.  
T should reduce slowly over time.



Image from: <http://static.comicvine.com/uploads/original/13130470/2931473-151295.jpg>

# How to Set and Reduce T?

- T starts with an initially high pre-defined value (parameter of the algorithm).
- There are different update rules (schedules)...
- Update rule:
  - $T = aT$ ,
  - a is close to, but smaller than, 1
  - e.g.,  $a = 0.95$

# Simulated Annealing

Simulated Annealing (assuming maximisation)

Input: initial temperature  $T_i$

1. current\_solution = generate initial solution randomly
  - 2.  $T = T_i$**
  3. Repeat:
    - 3.1 rand\_neighbour = generate random neighbour of current\_solution
    - 3.2 If quality(rand\_neighbour) <= quality(current\_solution) {
      - 3.2.1 With probability  $e^{\Delta E/T}$ ,**  
current\_solution = rand\_neighbour
    - } Else current\_solution = rand\_neighbour
    - 3.3  $T = \text{schedule}(T)$**
- Until a maximum number of iterations

# Simulated Annealing

Simulated Annealing (assuming maximisation)

Input: initial temperature  $T_i$ , minimum temperature  $T_f$

1. `current_solution = generate initial solution randomly`

**2.  $T = T_i$**

**3. Repeat:**

    3.1 `rand_neighbour = generate random neighbour of current_solution`

    3.2 If `quality(rand_neighbour) <= quality(current_solution)` {

**3.2.1 With probability  $e^{\Delta E/T}$ ,**

`current_solution = rand_neighbour`

        } Else `current_solution = rand_neighbour`

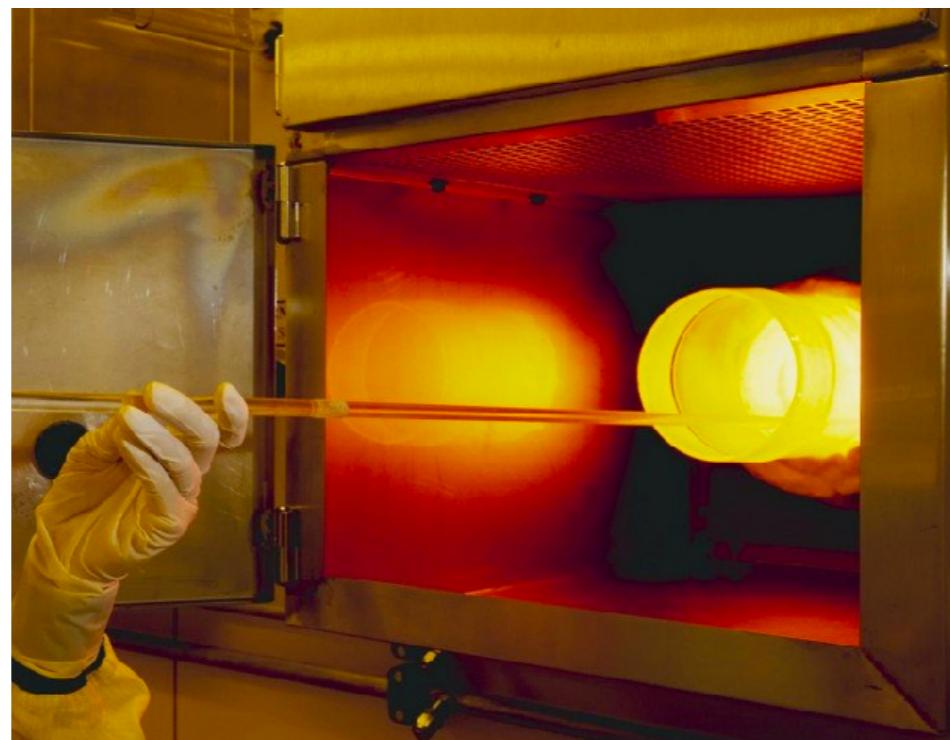
    3.3  **$T = schedule(T)$**

**until a minimum temperature  $T_f$  is reached or**

**until the current solution “stops changing”**

# Local Search

- Simulated annealing can also be considered as a local search, as it allows to move only to neighbour solutions.
- However, it has mechanisms to try to escape from local optima.



# Optimality

Is simulated annealing guaranteed to find the optimum?

- Simulated annealing is not guaranteed to find the optimum **in a reasonable amount of time**.
- Whether or not it will find the optimum depends on the termination criteria and the schedule.
- If we leave simulated annealing to run indefinitely, it is guaranteed to find an optimal solution, depending on the schedule used.
- However the time required for that can be prohibitive — even more than the time to enumerate all possible solutions using brute force.
- Therefore, the advantage of simulated annealing is that it can frequently obtain good (near optimal) solutions, by escaping from several poor local optima in a reasonable amount of time.

# Time and Space Complexity

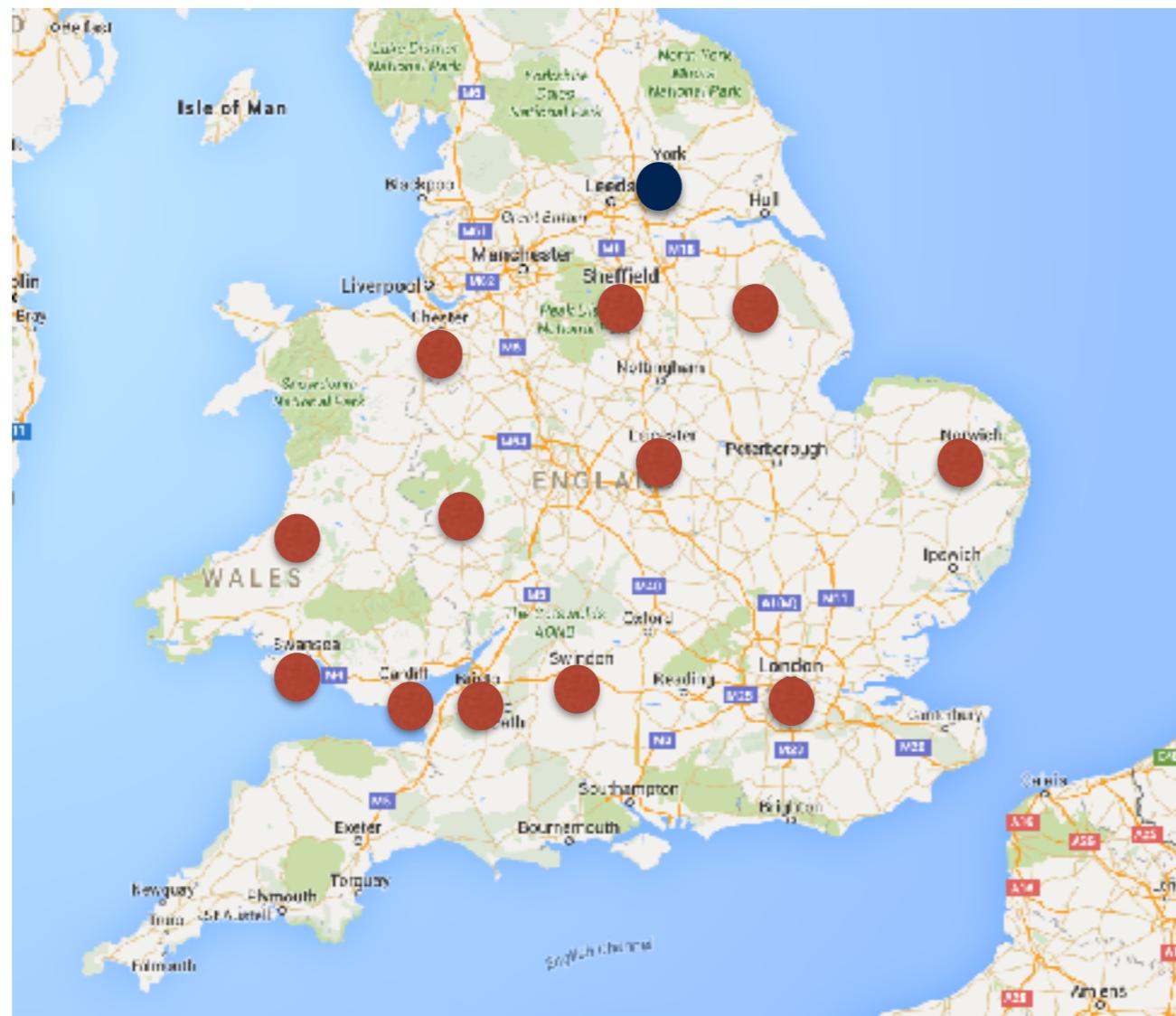
- Time complexity:
  - We will run more or less iterations depending on the schedule and minimum temperature / termination criterion.
  - It is possible to compute the time complexity to reach the optimal solution, but it varies depending on the problem and may be even worse than the brute force time complexity, as mentioned in the previous slide.
- Space complexity:
  - Depends on how the design variable is represented in the algorithm.

# Summary

- The probability of accepting neighbouring solutions of equal or worse quality than the current solution is inspired by metallurgy annealing.
- A “temperature” is used to control how low the probability is.
- A schedule is used to reduce the “temperature” over time.
- The worse a neighbour is, the lower the chances of accepting it.

# Next

- Dealing with constraints.

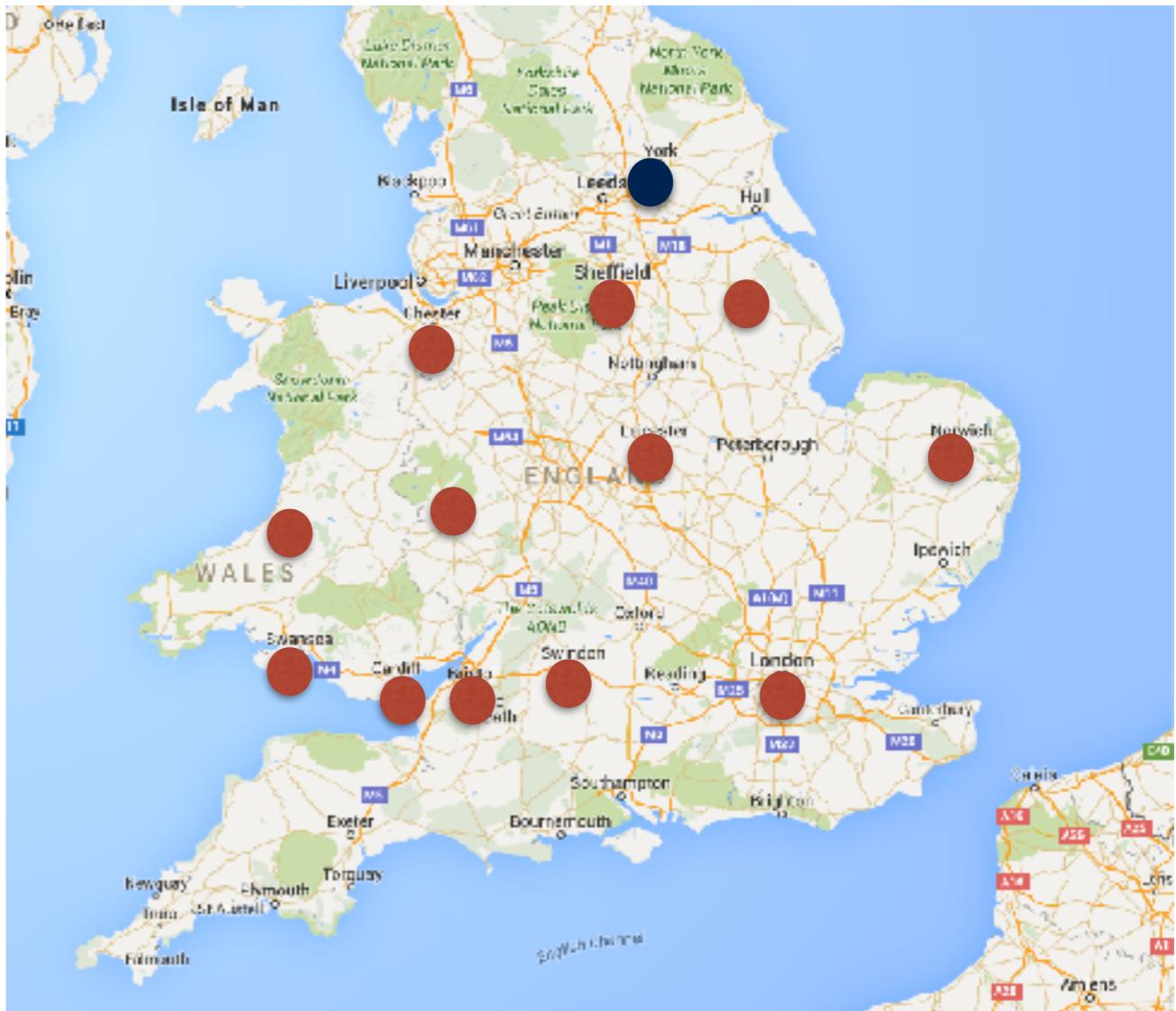


# Traveling Salesman Problem Formulation

# Leandro L. Minku

# Examples of Optimisation Problems

- Traveling Salesman Problem:
  - A salesman must travel passing through  $N$  cities.
  - Each city must be visited once and only once.
  - He/she must finish where he/she was at first.
  - The path between each pair of cities has a distance (or cost).



Problem: find a sequence of cities that minimises traveling distance (or cost), where each city appears once and only once.

# Traveling Salesman Problem Formulation

- Design variables represent a candidate solution.
  - Sequence  $\mathbf{x}$  of  $N$  cities to be visited, where cities are in  $C$ .
  - $C$  is a set containing the  $N$  cities to be visited.
  - The search space is all possible sequences of cities.
- Objective function defines the cost of a solution.
  - $\text{Total\_distance}(\mathbf{x}) =$   
sum of distances between consecutive cities in  $\mathbf{x}$  + distance from last city to the origin.
  - To be minimised.
- [Optional] Solutions must satisfy certain constraints.
  - Each city must appear once and only once in  $\mathbf{x}$  (explicit constraint).
  - Salesman must return to the city of origin (implicit constraint).
  - Only cities in  $C$  must appear in  $\mathbf{x}$  (implicit constraint).

$$\overline{x_1} \quad \overline{x_2} \quad \overline{x_3} \quad \overline{x_4} \quad \overline{x_5}$$

# Traveling Salesman Problem Formulation

- Design variables represent a candidate solution.
  - The design variable is a sequence  $\mathbf{x}$  of  $N$  cities, where  $x_i \in \{1, \dots, N\}$ ,  $\forall i \in \{1, \dots, N\}$ .
  - The  $N$  cities to be visited are represented by values  $\{1, \dots, N\}$ .
  - The search space is all possible sequences of  $N$  cities, where cities are in  $\{1, \dots, N\}$ .
- Objective function defines the cost of a solution.

$$\text{minimise totalDistance}(\mathbf{x}) = \left( \sum_{i=1}^{N-1} D_{x_i, x_{i+1}} \right) + D_{x_N, x_1}$$

$$\frac{1}{x_1} \quad \frac{3}{x_2} \quad \frac{2}{x_3} \quad \frac{4}{x_4} \quad \frac{5}{x_5}$$

where  $D_{j,k}$  is the distance of the path between cities  $j$  and  $k$ .

- [Optional] Solutions must satisfy certain constraints.
  - Each city must appear once and only once in  $\mathbf{x}$  (explicit constraint).

“For each city  $i$  in  $\{1, \dots, N\}$ ”,  
 $\forall i \in \{1, \dots, N\}, \left( \sum_{j=1}^N 1(x_j = i) \right) = 1$

$$1(x_j = i) = \begin{cases} 1, & \text{if } x_j = i \\ 0, & \text{if } x_j \neq i \end{cases}$$

# Traveling Salesman Problem Formulation

$$\forall i \in \{1, \dots, N\}, \left( \sum_{j=1}^N 1(x_j = i) \right) = 1 \quad 1(x_j = i) = \begin{cases} 1, & \text{if } x_j = i \\ 0, & \text{if } x_j \neq i \end{cases}$$

$$\forall i \in \{1, \dots, N\}, h_i(\mathbf{x}) = \left( \sum_{j=1}^N 1(x_j = i) \right) - 1 = 0$$

$\{1, 2, 3, 4, 5\}$

$i$

$$\frac{4}{x_1} \frac{2}{x_2} \frac{1}{x_3} \frac{3}{x_4} \frac{3}{x_5}$$

$j$

$$\text{Sum}_1: 0 + 0 + 1 + 0 + 0 = 1$$

$$\text{Sum}_2: 0 + 1 + 0 + 0 + 0 = 1$$

$$\text{Sum}_3: 0 + 0 + 0 + 1 + 1 = 2$$

# Traveling Salesman Problem Formulation

- Design variables represent a candidate solution.
  - The design variable is a sequence  $\mathbf{x}$  of  $N$  cities, where  $x_i \in \{1, \dots, N\}$ ,  $\forall i \in \{1, \dots, N\}$ .
  - The  $N$  cities to be visited are represented by values  $\{1, \dots, N\}$ .
  - The search space is all possible sequences of  $N$  cities, where cities are in  $\{1, \dots, N\}$ .
- Objective function defines the cost of a solution.

$$\text{minimise } \text{totalDistance}(\mathbf{x}) = \left( \sum_{i=1}^{N-1} D_{x_i, x_{i+1}} \right) + D_{x_N, x_1}$$

where  $D_{j,k}$  is the distance of the path between cities  $j$  and  $k$ .

- [Optional] Solutions must satisfy certain constraints.

For each city  $i$ ,  $h_i(\mathbf{x}) = 0$

# Summary

- Traveling salesman problem formulation, including constraints.

# Next

- How to deal with constraints?



# Constraint Handling — Representation, Initialisation and Neighbourhood Operators

Leandro L. Minku

# How to Deal with Constraints in Optimisation Problems?

- Most real world problems have constraints.
- Optimisation algorithms themselves usually do not contain strategies to deal with constraints.
- Instead, strategies need to be designed for each problem.
- Examples of strategies:
  - Representation, initialisation and neighbourhood operators.
  - Objective function.

# How to Deal with Constraints in Optimisation Problems?

- Most real world problems have constraints.
- Optimisation algorithms themselves usually do not contain strategies to deal with constraints.
- Instead, strategies need to be designed for each problem.
- Examples of strategies:
  - Representation, initialisation and neighbourhood operators.
  - Objective function.

# Traveling Salesman Problem Formulation

- Design variables represent a candidate solution.
  - The design variable is a sequence  $\mathbf{x}$  of  $N$  cities, where  $x_i \in \{1, \dots, N\}$ ,  $\forall i \in \{1, \dots, N\}$ .
  - The  $N$  cities to be visited are represented by values  $\{1, \dots, N\}$ .
  - The search space is all possible sequences of  $N$  cities, where cities are in  $\{1, \dots, N\}$ .
- Objective function defines the cost of a solution.

$$\text{minimise } \text{totalDistance}(\mathbf{x}) = \left( \sum_{i=1}^{N-1} D_{x_i, x_{i+1}} \right) + D_{x_N, x_1}$$

where  $D_{j,k}$  is the distance of the path between cities  $j$  and  $k$ .

- [Optional] Solutions must satisfy certain constraints.

$$\forall i \in \{1, \dots, N\}, \quad h_i(\mathbf{x}) = \left( \sum_{j=1}^N 1(x_j = i) \right) - 1 = 0 \quad 1(x_j = i) = \begin{cases} 1, & \text{if } x_j = i \\ 0, & \text{if } x_j \neq i \end{cases}$$

# Designing Representation, Initialisation and Neighbourhood Operators to Deal with Constraints

- **Representation:**

- 1-dimensional array of size  $N$ , where  $N$  is the number of cities to visit.
- The fact that the return to the initial city is not in the representation helps to deal with the implicit constraint that we must return to the city of origin.
- E.g.: for  $N = 5$

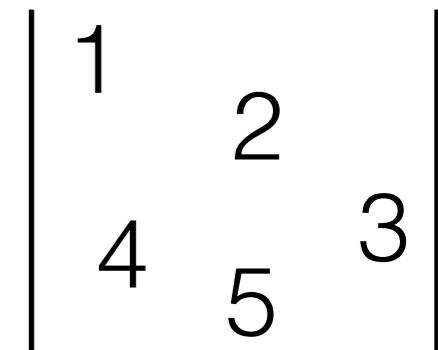
1   3   2   4   5   1

3   1   2   4   5   3

- **Initialisation:**

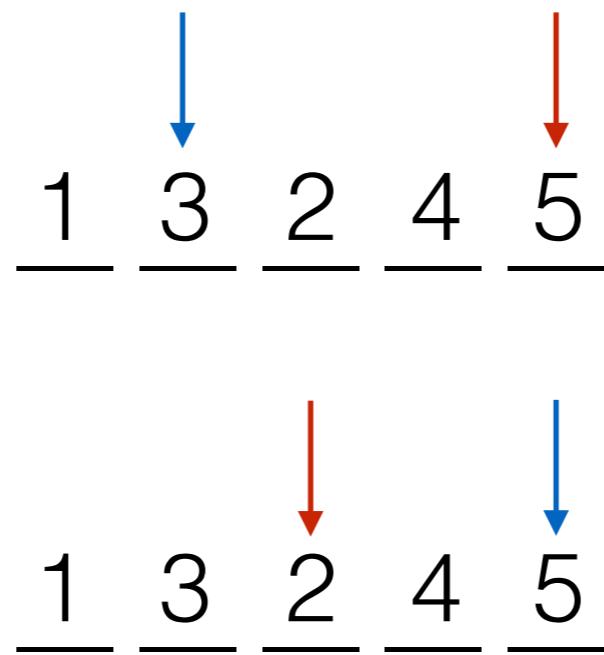
- Draw cities uniformly at random from  $\{1, \dots, N\}$  without replacement,
- This ensures that there will be no missing or duplicated cities (explicit constraint) and that only cities in  $\{1, \dots, N\}$  are used (implicit constraint).

— — — — —

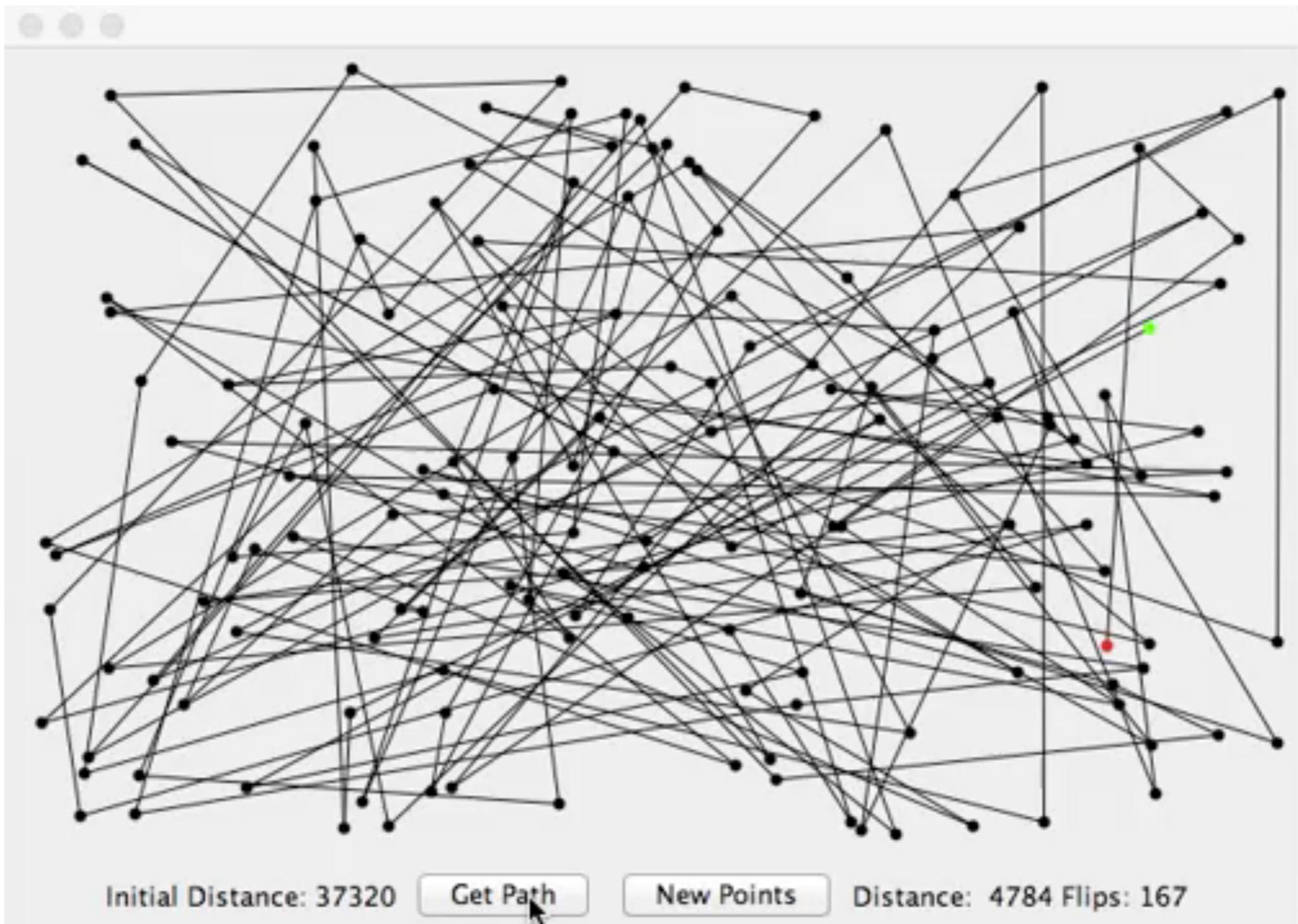


# Designing Representation, Initialisation and Neighbourhood Operators to Deal with Constraints

- Neighbourhood operator:
  - Reverse the path between two randomly picked cities.
  - This ensures that there will be no missing or duplicated cities (explicit constraint) and that only cities in  $\{1, \dots, N\}$  are used (implicit constraint).



- This design ensures that the constraints are satisfied.



[Video posted by sarahbau: <https://youtu.be/3TrnjUKeFg8> ]

# Dealing with Constraints Based on Representation, Initialisation and Neighbourhood Operators

- **Advantage:**
  - Ensure that no infeasible candidate solutions will be generated, facilitating the search for optimal solutions.
- **Disadvantage:**
  - May be difficult to design, and the design is problem-dependent.
  - Sometimes, it could restrict the search space too much, making it difficult to find the optimal solution.

# Summary

- We need to design strategies to deal with the constraints.
- Examples of strategies:
  - Representation, initialisation and neighbourhood operators.
  - Objective function.

# Next

- Examples of strategies:
  - Representation, initialisation and neighbourhood operators.
  - Objective function.



# Constraint Handling — Objective Function

Leandro L. Minku

# Traveling Salesman Problem Formulation

- Design variables represent a candidate solution.
  - The design variable is a sequence  $\mathbf{x}$  of  $N$  cities, where  $x_i \in \{1, \dots, N\}$ ,  $\forall i \in \{1, \dots, N\}$ .
  - The  $N$  cities to be visited are represented by values  $\{1, \dots, N\}$ .
  - The search space is all possible sequences of  $N$  cities, where cities are in  $\{1, \dots, N\}$ .
- Objective function defines the cost of a solution.

$$\text{minimise } \text{totalDistance}(\mathbf{x}) = \left( \sum_{i=1}^{N-1} D_{x_i, x_{i+1}} \right) + D_{x_N, x_1}$$

where  $D_{j,k}$  is the distance of the path between cities  $j$  and  $k$ .

- [Optional] Solutions must satisfy certain constraints.

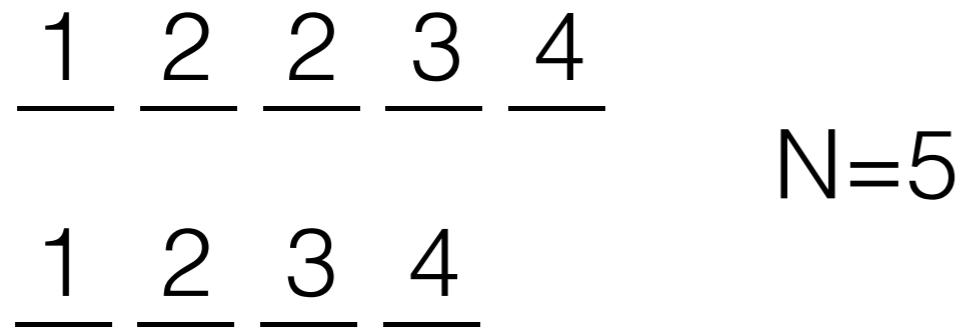
$$\forall i \in \{1, \dots, N\}, \quad h_i(\mathbf{x}) = \left( \sum_{j=1}^N 1(x_j = i) \right) - 1 = 0 \quad 1(x_j = i) = \begin{cases} 1, & \text{if } x_j = i \\ 0, & \text{if } x_j \neq i \end{cases}$$

# Designing Objective Functions to Deal With Constraints

- The original objective function of a problem can be modified to deal with constraints.
- A **penalty** can be added for infeasible solutions, increasing their cost.

# Designing Objective Functions to Deal With Constraints

- E.g.: assume that the representation is a list of any size, and that our initialisation procedure is uniformly at random with replacement.



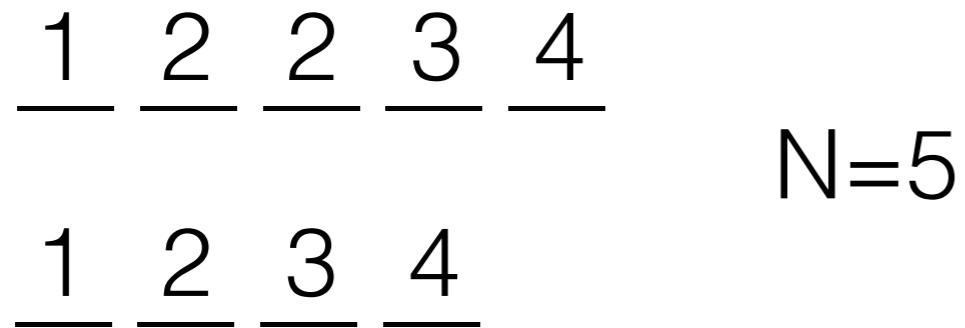
- Objetive function:

$$\text{minimise } \text{totalDistance}(\mathbf{x}) = \left( \sum_{i=1}^{\text{size}(\mathbf{x})-1} D_{x_i, x_{i+1}} \right) + D_{x_{\text{size}(\mathbf{x})}, x_1}$$

How to modify the objective function to deal with the constraint that each city must appear once and only once?

# Designing Objective Functions to Deal With Constraints

- E.g.: assume that the representation is a list of any size, and that our initialisation procedure is uniformly at random with replacement.



- Objetive function:

$$\text{minimise } \text{totalDistance}(\mathbf{x}) = \left( \sum_{i=1}^{\text{size}(\mathbf{x})-1} D_{x_i, x_{i+1}} \right) + D_{x_{\text{size}(\mathbf{x})}, x_1} + n_m P + n_d P$$

where  $n_m$  is the number of cities missing,  $n_d$  is the number of duplications of cities and  $P$  is a large positive constant.

# Generalising The Strategy

Minimise  $f(\mathbf{x})$

Subject to  $g_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, m$

$h_j(\mathbf{x}) = 0, \quad j = 1, \dots, n$

Penalty

Minimise  $f(\mathbf{x}) + Q(\mathbf{x})$

$$Q(\mathbf{x}) = \begin{cases} 0 & \text{if } \mathbf{x} \text{ is feasible} \\ P \times [g_a(\mathbf{x})^2 + g_b(\mathbf{x})^2 + \dots + h_{a'}(\mathbf{x})^2 + h_{b'}(\mathbf{x})^2 + \dots] & \text{otherwise} \end{cases}$$

Only sum here the violated constraints

$\overbrace{g_a(\mathbf{x})^2 + g_b(\mathbf{x})^2 + \dots + h_{a'}(\mathbf{x})^2 + h_{b'}(\mathbf{x})^2 + \dots}$

where  $P$  is a large positive constant.

# Generalising The Strategy

Minimise  $f(\mathbf{x})$

Subject to  $g_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, m$

$h_j(\mathbf{x}) = 0, \quad j = 1, \dots, n$

Minimise  $f(\mathbf{x}) + Q(\mathbf{x})$

Penalty

$$Q(\mathbf{x}) = P \times [v_{g1}g_1(\mathbf{x})^2 + v_{g2}g_2(\mathbf{x})^2 + \dots + v_{gm}g_m(\mathbf{x})^2 + \\ + v_{h1}h_1(\mathbf{x})^2 + v_{h2}h_2(\mathbf{x})^2 + \dots + v_{hn}h_n(\mathbf{x})^2]$$

where  $P$  is a large positive constant, and  $v_{gi}$  and  $v_{hi}$  are 1 if the corresponding constraint is violated and 0 otherwise.

# Dealing with Constraints Based on Objective Functions

- Advantage:
  - Easier to design.
- Disadvantage:
  - Algorithm has to search for feasible solutions.

# Completeness

- If we use a strategy to deal with constraints that never enables any infeasible solution to be generated, algorithms such as Hill Climbing and Simulated Annealing are complete.
- Otherwise:
  - [Hill Climbing](#): not complete if the objective function has local optima.
  - [Simulated Annealing](#): not guaranteed to find a feasible solution within a reasonable amount of time.

# Summary

- We need to design strategies to deal with the constraints.
- Examples of strategies:
  - Representation, initialisation and neighbourhood operators.
  - Objective function.

# Next

- Example applications.

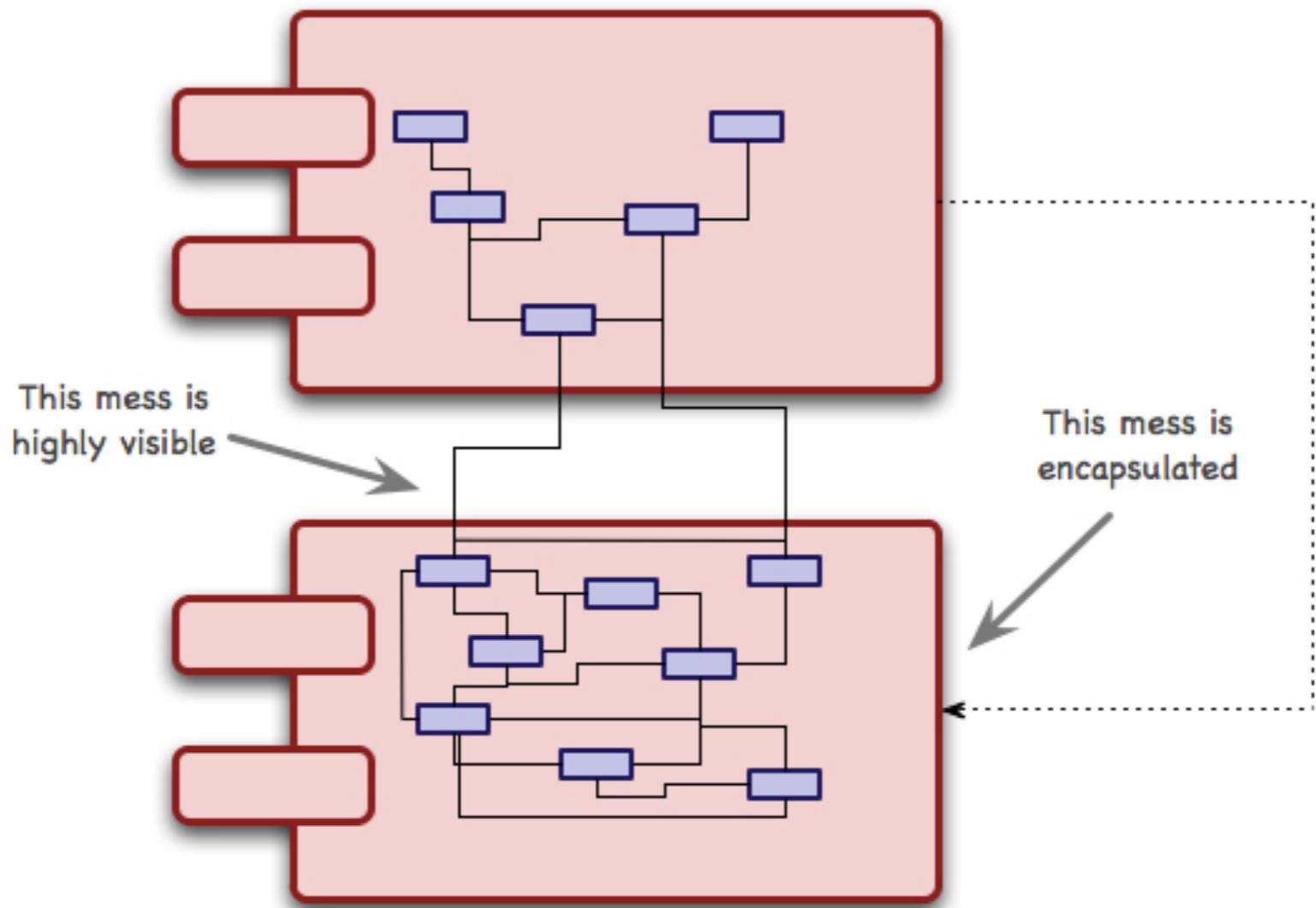


Image from: <http://www.kirkk.com/modularity/wp-content/uploads/2009/12/EncapsulatingDesign1.jpg>

## Example of Hill Climbing Application: Software Module Clustering (Problem Formulation)

Leandro L. Minku

# Hill Climbing Applications

Hill-Climbing is applicable to any optimisation problem, but its success depends on the shape of the objective function for the problem instance in hands.

Simple algorithm — not difficult to implement.  
Could be attempted first to see if the retrieved solutions are good enough, before a more complex algorithm is investigated.

# Applications

- Hill-climbing has been successfully applied to software module clustering.
- Software Module Clustering:
  - Software is composed of several units, which can be organised into modules.
  - Well modularised software is easier to develop and maintain.
  - As software evolves, modularisation tends to degrade.

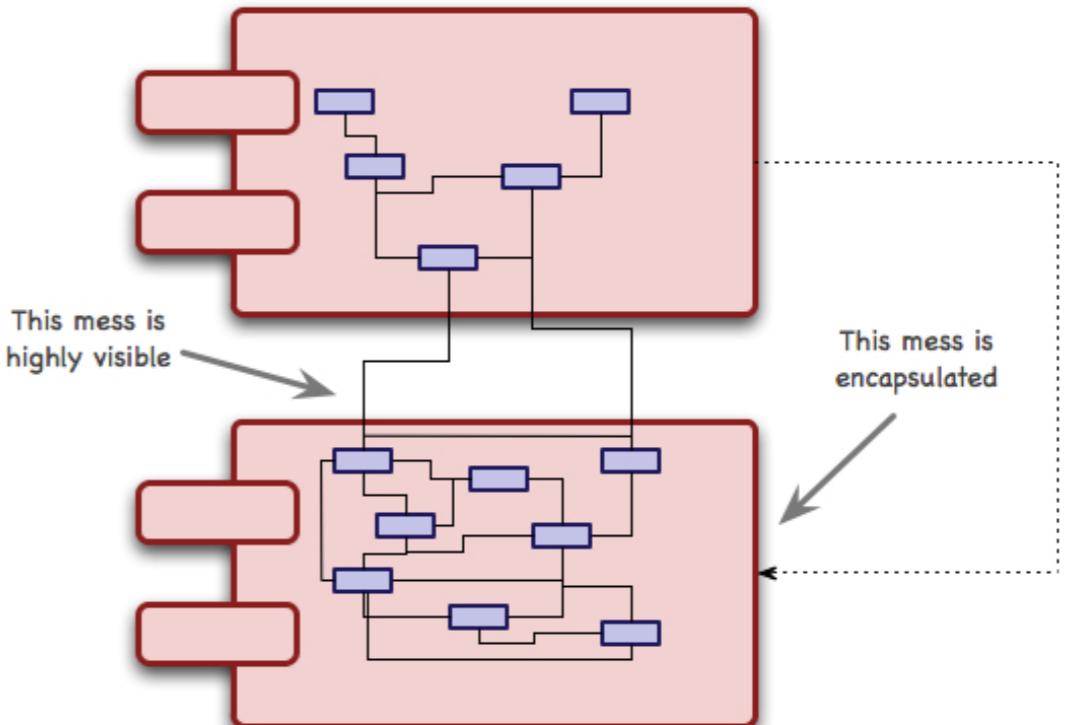


Image from: <http://www.kirkk.com/modularity/wp-content/uploads/2009/12/EncapsulatingDesign1.jpg>

Problem: **find** an allocation of units into modules that **maximises** the quality of modularisation.

# Applying Hill-Climbing (and Simulated Annealing)

- We need to specify:
  - Optimisation problem formulation:
    - Design variable and search space
    - Constraints
    - Objective function
  - Algorithm-specific operators:
    - Representation.
    - Initialisation procedure.
    - Neighbourhood operator.
  - Strategy to deal with constraints, e.g.:
    - Representation, initialisation and neighbourhood operators that ensure only feasible solutions to be generated.
    - Modification in the objective function.

# Formulation Optimisation Problems

- Design variables represent a candidate solution.
  - Design variables define the search space of candidate solutions.
- Objective function defines our goal.
  - Can be used to evaluate the quality of solutions.
  - Function to be optimised (maximised or minimised).
- [Optional] Solutions must satisfy certain constraints.

# Design Variable

Design variable: allocation of units into modules.

- Consider that we have  $N$  units, identified by natural numbers in  $\{1, 2, \dots, N\}$ .
- This means that we have at most  $N$  modules.
- Our design variable is a list  $L$  of  $N$  modules, where each module  $L_i$ ,  $i \in \{1, 2, \dots, N\}$ , is a set containing a minimum of 0 and a maximum of  $N$  units.

$$L_1 = \{\}$$

Module 1

$$L_2 = \{1\}$$

Module 2

unit 1

$$L_3 = \{2, 4, 5\}$$

Module 3

unit 2

unit 4

unit 5

$$L_4 = \{\}$$

Module 4

$$L_5 = \{3\}$$

Module 5

unit 3

# Design Variable

Design variable: allocation of units into modules.

- Consider that we have  $N$  units, identified by natural numbers in  $\{1, 2, \dots, N\}$ .
- This means that we have at most  $N$  modules.
- Our design variable is a list  $L$  of  $N$  modules, where each module  $L_i$ ,  $i \in \{1, 2, \dots, N\}$ , is a set containing a minimum of 0 and a maximum of  $N$  units.

$$L_1 = \{\}$$

Module 1

$$L_2 = \{1\}$$

Module 2

unit 1

$$L_3 = \{2, 4, 5\}$$

Module 3

unit 2

unit 4

unit 5

$$L_4 = \{\}$$

Module 4

$$L_5 = \{3\}$$

Module 5

unit 3

Search space: all possible allocations.

# Constraints and Objective Function

Constraints: N/A

Objective function: quality of modularisation (to be maximised).

How to compute quality?

What does good quality mean?

$$L_1 = \{\}$$

Module 1

$$L_2 = \{1\}$$

Module 2

$$L_3 = \{2, 4, 5\}$$

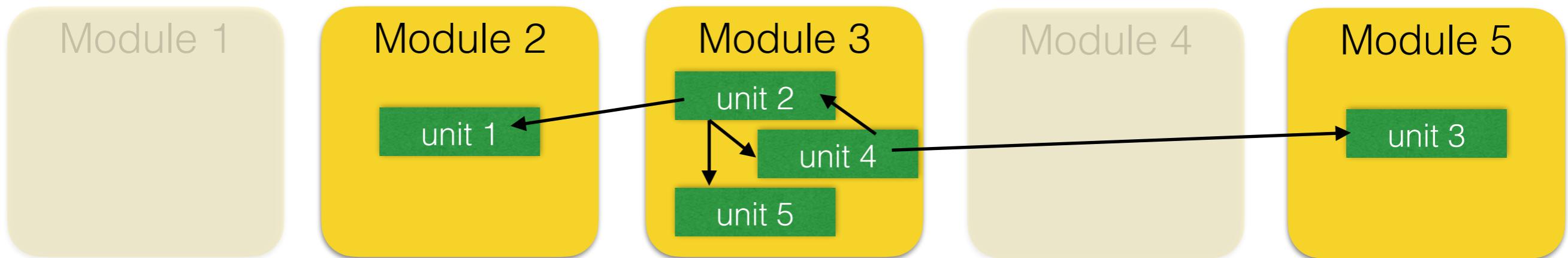
Module 3

$$L_4 = \{\}$$

Module 4

$$L_5 = \{3\}$$

Module 5



A unit can make use of (depend on) another unit — this information can be retrieved from the current source code being refactored.

# Constraints and Objective Function

Constraints: N/A

Objective function: quality of modularisation (to be maximised).

How to compute quality?

What does good quality mean?

$$L_1 = \{\}$$

Module 1

$$L_2 = \{1\}$$

Module 2

$$L_3 = \{2, 4, 5\}$$

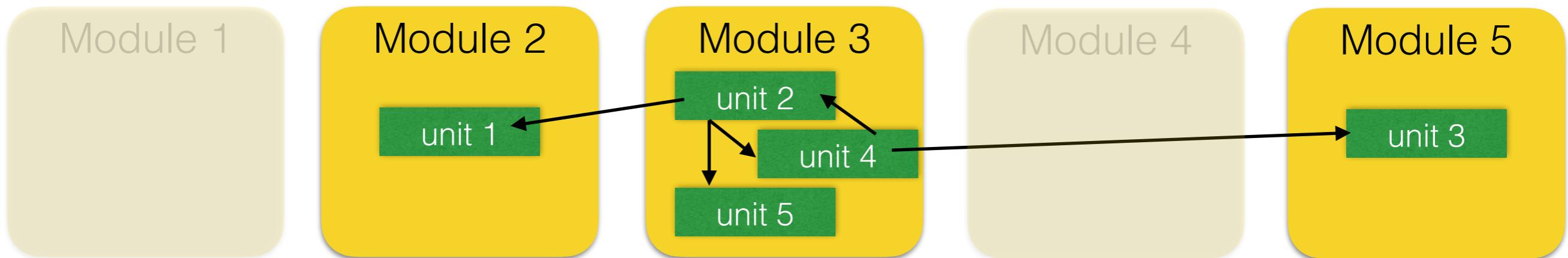
Module 3

$$L_4 = \{\}$$

Module 4

$$L_5 = \{3\}$$

Module 5



Lots of connections inside a module (high cohesion) and few connections between modules (low coupling).

# Quality of a Module $L_i$

Constraints: N/A

Objective function: quality of modularisation (to be maximised).

How to compute quality?

What does good quality mean?

$$L_1 = \{\}$$

Module 1

$$L_2 = \{1\}$$

Module 2

$$L_3 = \{2, 4, 5\}$$

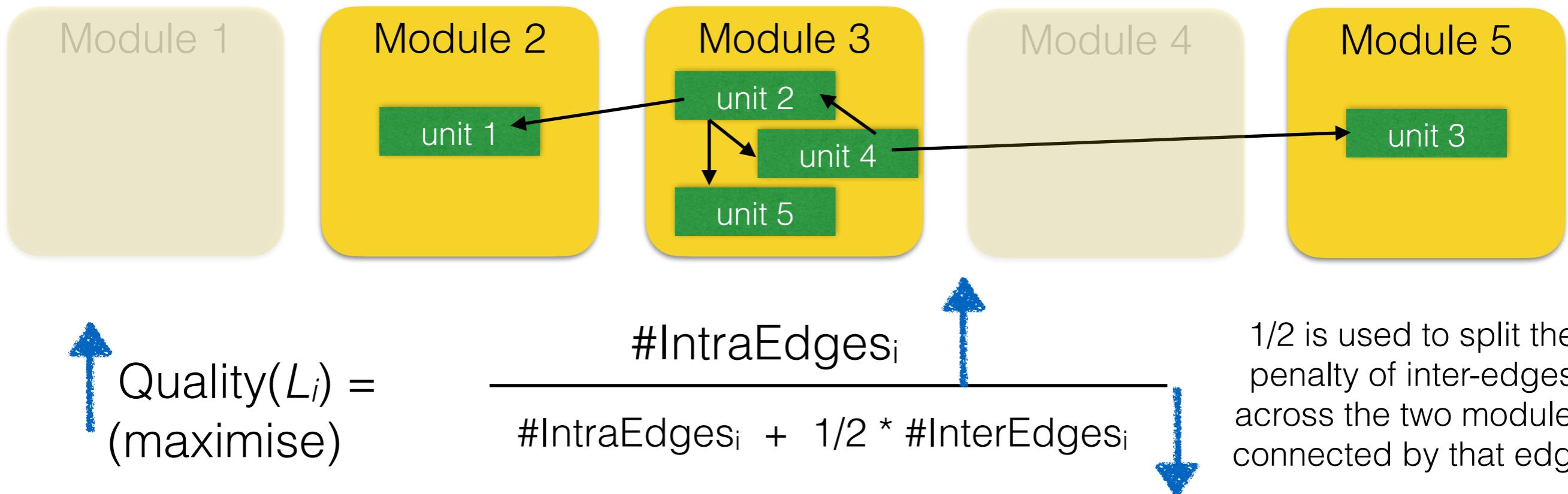
Module 3

$$L_4 = \{\}$$

Module 4

$$L_5 = \{3\}$$

Module 5



# Quality of a Module $L_i$

Constraints: N/A

Objective function: quality of modularisation (to be maximised).

How to compute quality?

What does good quality mean?

$$L_1 = \{\}$$

Module 1

$$L_2 = \{1\}$$

Module 2

$$L_3 = \{2, 4, 5\}$$

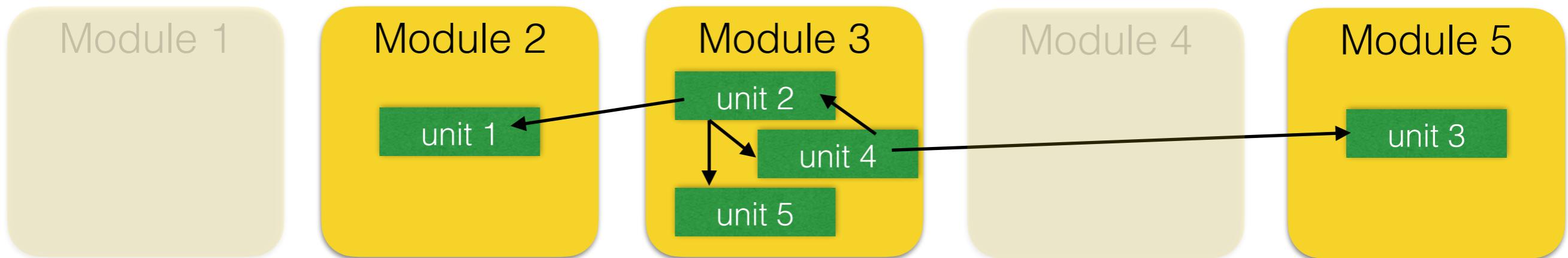
Module 3

$$L_4 = \{\}$$

Module 4

$$L_5 = \{3\}$$

Module 5



↑  
Quality( $L_i$ ) =  
(maximise)

$$\frac{\#IntraEdges_i}{\#IntraEdges_i + 1/2 * \#InterEdges_i}$$

The  $\#intra\_edges_i$  in  
the denominator is a  
normalisation factor.

# Intra Edges

Constraints: N/A

Objective function: quality of modularisation (to be maximised).

How to compute quality?

What does good quality mean?

$$L_1 = \{\}$$

Module 1

$$L_2 = \{1\}$$

Module 2

$$L_3 = \{2, 4, 5\}$$

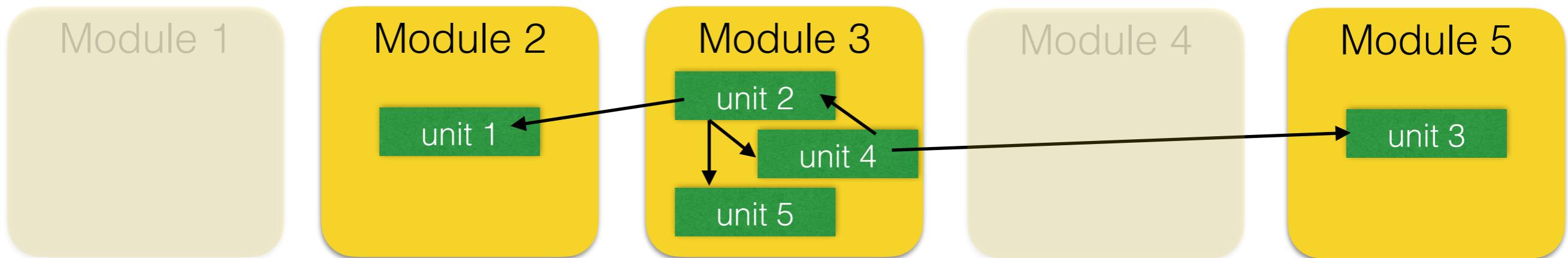
Module 3

$$L_4 = \{\}$$

Module 4

$$L_5 = \{3\}$$

Module 5



$$\#\text{IntraEdges}_i = \sum_{j=1}^{\text{size}(L_i)} \sum_{j'=1}^{\text{size}(L_i)} D_{L_{ij}, L_{ij'}}$$

$$D_{a,b} = \begin{cases} 1, & \text{if unit } a \text{ depends on unit } b \\ 0, & \text{otherwise (incl. diagonal)} \end{cases}$$

# Inter Edges

Constraints: N/A

Objective function: quality of modularisation (to be maximised).

How to compute quality?

What does good quality mean?

$$L_1 = \{\}$$

Module 1

$$L_2 = \{1\}$$

Module 2

$$L_3 = \{2, 4, 5\}$$

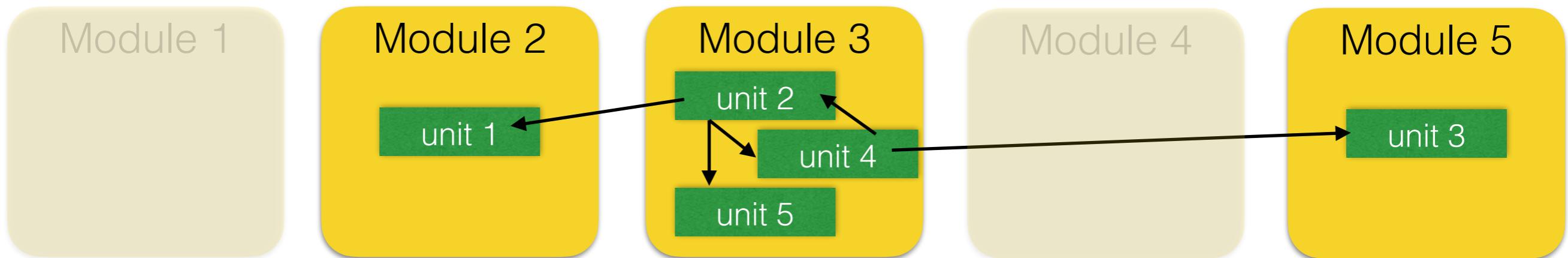
Module 3

$$L_4 = \{\}$$

Module 4

$$L_5 = \{3\}$$

Module 5



$$\# \text{InterEdges}_i = \sum_{j=1}^{\text{size}(L_i)} \sum_{i' \in \{1, 2, \dots, N\} | i' \neq i} \sum_{j'=1}^{\text{size}(L_{i'})} (D_{L_{ij}, L_{i'j'}} + D_{L_{i'j'}, L_{ij}})$$

# Quality of a Module $L_i$

Constraints: N/A

Objective function: quality of modularisation (to be maximised).

How to compute quality?

What does good quality mean?

$$L_1 = \{\}$$

Module 1

$$L_2 = \{1\}$$

Module 2

$$L_3 = \{2, 4, 5\}$$

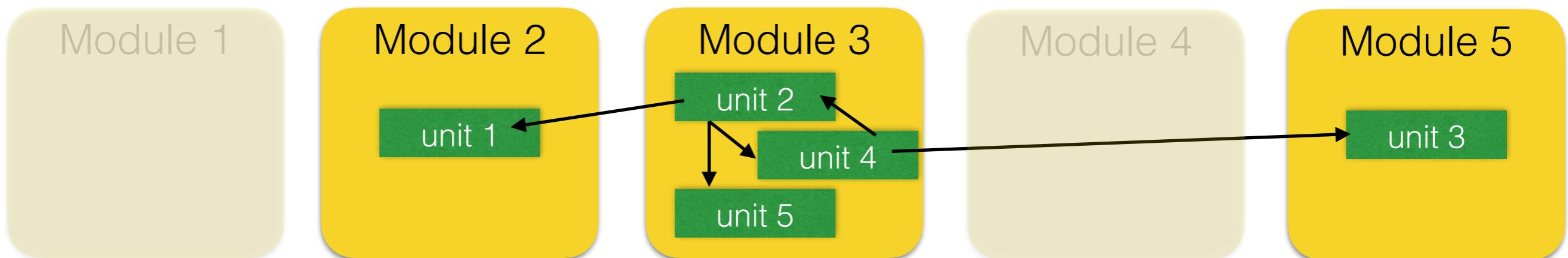
Module 3

$$L_4 = \{\}$$

Module 4

$$L_5 = \{3\}$$

Module 5



$$\text{Quality}(L_i) = \frac{\#\text{IntraEdges}_i}{(\text{maximise}) \quad \#\text{IntraEdges}_i + 1/2 * \#\text{InterEdges}_i}$$

This is the quality of a **single** module.

# Quality of a Solution $L$

Constraints: N/A

Objective function: quality of modularisation (to be maximised).

How to compute quality?

What does good quality mean?

$$L_1 = \{\}$$

Module 1

$$L_2 = \{1\}$$

Module 2

$$L_3 = \{2, 4, 5\}$$

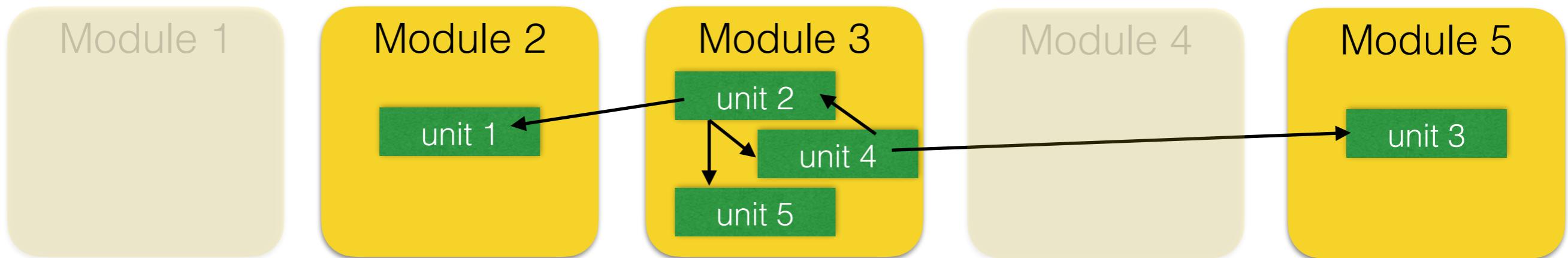
Module 3

$$L_4 = \{\}$$

Module 4

$$L_5 = \{3\}$$

Module 5



$\text{Quality}(L) = \text{sum of the qualities of the non-empty modules}$   
(maximise)

# Quality of a Solution $L$

Constraints: N/A

Objective function: quality of modularisation (to be maximised).

How to compute quality?

What does good quality mean?

$$L_1 = \{\}$$

Module 1

$$L_2 = \{1\}$$

Module 2

$$L_3 = \{2, 4, 5\}$$

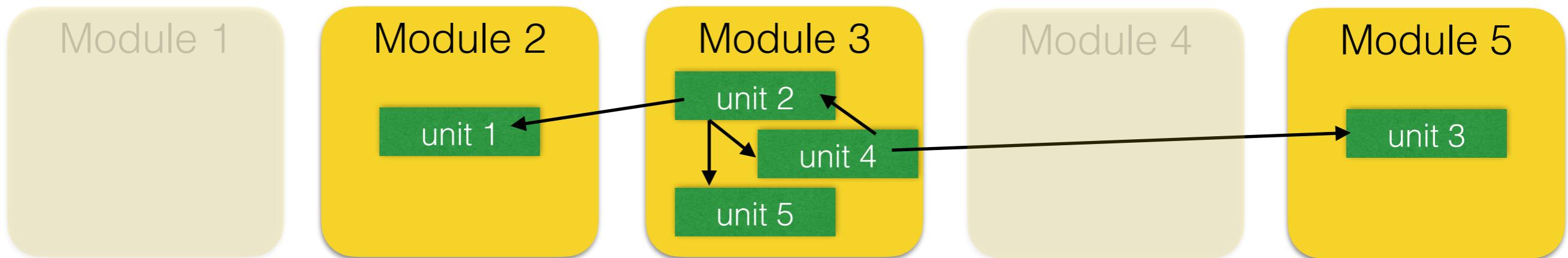
Module 3

$$L_4 = \{\}$$

Module 4

$$L_5 = \{3\}$$

Module 5



$$\text{Quality}(L) = \\ (\text{maximise})$$

$$\sum_{\substack{i \in \{1, 2, \dots, N\} \\ L_i \neq \{\}}} \text{Quality}(L_i)$$

# Problem Formulation

## Hill-Climbing (assuming maximisation)

1. `current_solution` = generate initial solution randomly
2. Repeat:
  - 2.1 generate neighbour solutions (differ from current solution by a single element)
  - 2.2 `best_neighbour` = get highest quality neighbour of `current_solution`
  - 2.3 If `quality(best_neighbour) <= quality(current_solution)`
    - 2.3.1 Return `current_solution`
  - 2.4 `current_solution` = `best_neighbour`
- Until a maximum number of iterations

Design variable —>  
what is a candidate solution for us?

# Problem Formulation

## Hill-Climbing (assuming maximisation)

1. current\_solution = generate initial solution randomly
2. Repeat:
  - 2.1 generate neighbour solutions (differ from current solution by a single element)
  - 2.2 best\_neighbour = get highest quality neighbour of current\_solution
  - 2.3 If quality(best\_neighbour) <= quality(current\_solution)
    - 2.3.1 Return current\_solution
  - 2.4 current\_solution = best\_neighbour
- Until a maximum number of iterations

Design variable →  
what is a candidate solution for us?

Objective →  
what is quality for us?

Are there any constraints that  
need to be satisfied?

Simulated Annealing would also require a problem formulation to be able to solve a problem.

# Summary

- Software Module Clustering problem formulation.

# Next

- Representation, initialisation and neighbourhood operators.

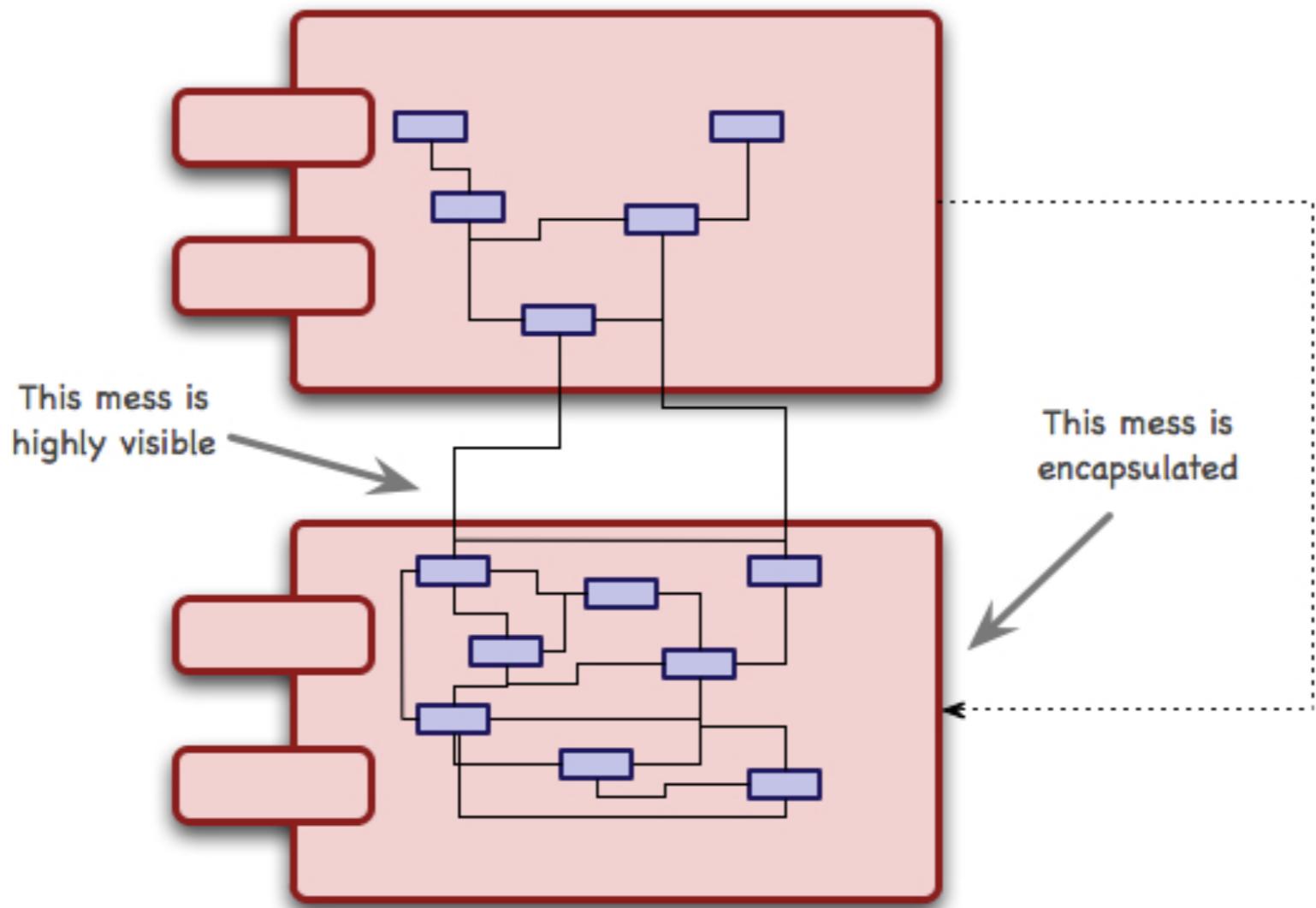


Image from: <http://www.kirkk.com/modularity/wp-content/uploads/2009/12/EncapsulatingDesign1.jpg>

## Example of Hill Climbing Application: Software Module Clustering (Algorithmic Design)

Leandro L. Minku

# Design Variable

Design variable: allocation of units into modules.

- Consider that we have  $N$  units, identified by natural numbers in  $\{1, 2, \dots, N\}$ .
- This means that we have at most  $N$  modules.
- Our design variable is a list  $L$  of  $N$  modules, where each module  $L_i$ ,  $i \in \{1, 2, \dots, N\}$ , is a set containing a minimum of 0 and a maximum of  $N$  units.

$$L_1 = \{\}$$

Module 1

$$L_2 = \{1\}$$

Module 2

unit 1

$$L_3 = \{2, 4, 5\}$$

Module 3

unit 2

unit 4

unit 5

$$L_4 = \{\}$$

Module 4

$$L_5 = \{3\}$$

Module 5

unit 3

# Constraints and Objective Function

Constraints: N/A

Objective function: quality of modularisation (to be maximised).

$$\text{Quality}(L) = \sum_{\substack{i \in \{1, 2, \dots, N\} \\ L_i \neq \{\}}} \text{Quality}(L_i)$$

$$\text{Quality}(L_i) = \frac{\#\text{IntraEdges}_i}{\#\text{IntraEdges}_i + 1/2 * \#\text{InterEdges}_i}$$

# Problem Formulation

## Hill-Climbing (assuming maximisation)

1. current\_solution = generate initial solution randomly
2. Repeat:
  - 2.1 generate neighbour solutions (differ from current solution by a single element)
  - 2.2 best\_neighbour = get highest quality neighbour of current\_solution
  - 2.3 If quality(best\_neighbour) <= quality(current\_solution)
    - 2.3.1 Return current\_solution
  - 2.4 current\_solution = best\_neighbour
- Until a maximum number of iterations

Design variable →  
what is a candidate solution for us?

Objective →  
what is quality for us?

Are there any constraints that  
need to be satisfied?

# Designing Representation, Initialisation and Neighbourhood Operators

## Hill-Climbing (assuming maximisation)

1. current\_solution = generate initial solution randomly
2. Repeat:
  - 2.1 generate neighbour solutions (differ from current solution by a single element)
  - 2.2 best\_neighbour = get highest quality neighbour of current\_solution
  - 2.3 If quality(best\_neighbour) <= quality(current\_solution)
    - 2.3.1 Return current\_solution
  - 2.4 current\_solution = best\_neighbour

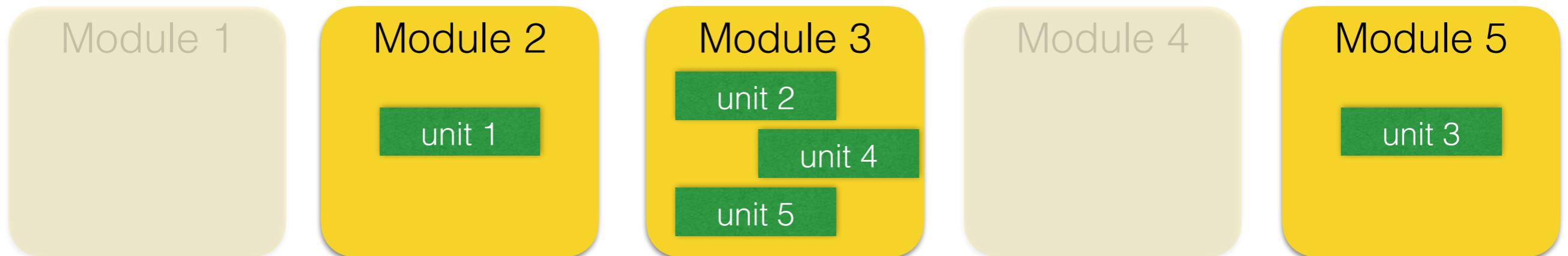
Until a maximum number of iterations

- Representation:
  - How to store the design variable.
  - E.g., boolean, integer or float variable or array.
- Initialisation:
  - Usually involve randomness.
- Neighbourhood operator:
  - How to generate neighbour solutions.

# Representation

How to represent the design variable internally in the implementation?

- E.g., list of  $N$  modules, where each module is a list of integers in  $\{1, 2, \dots, N\}$  identifying the existing units.

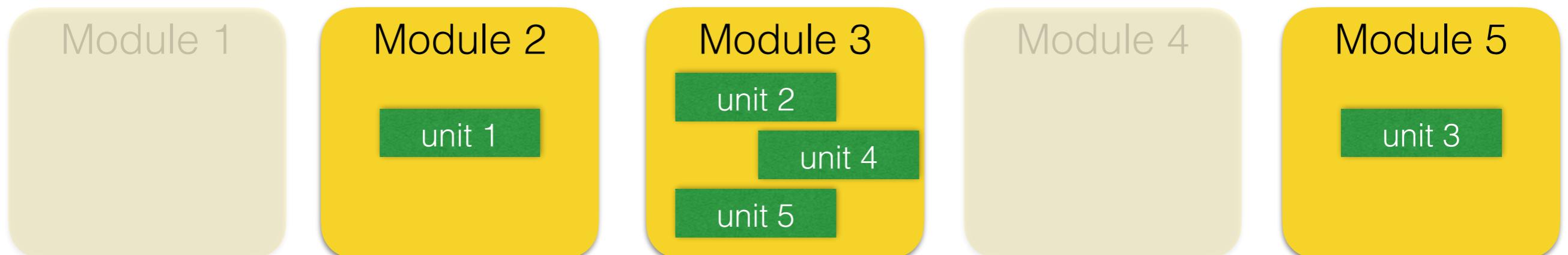


- E.g., if we have  $N=5$ , a possible allocation is  $L = \{\{\}, \{1\}, \{2, 4, 5\}, \{\}, \{3\}\}$ .

# Representation

How to represent the design variable internally in the implementation?

- E.g., matrix  $A_{N \times N}$ , where  $A_{ij} = 1$  if unit j is in module i, and 0 otherwise.



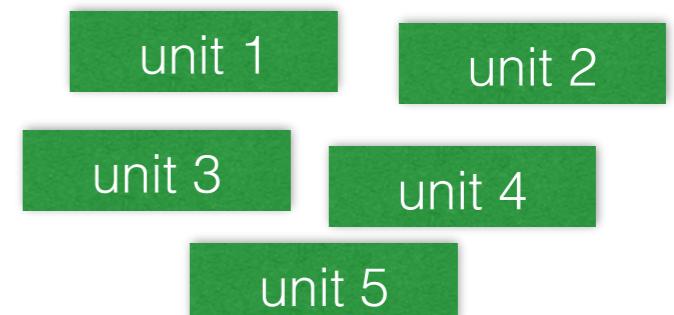
$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

# Initialisation

E.g.: place each unit into a randomly picked module.

For each unit  $u \in \{1, \dots, N\}$

Add  $u$  to a module  $L_i$ , where  $i \sim U\{1, N\}$



Module 1

Module 2

Module 3

Module 4

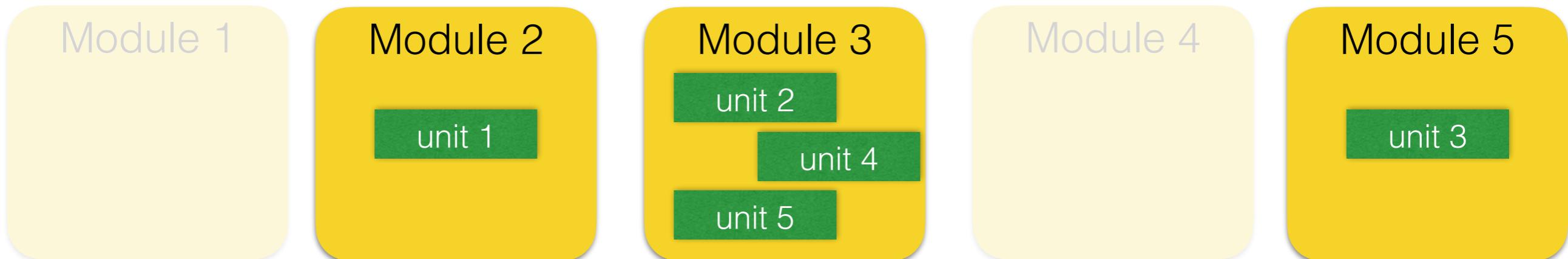
Module 5

$$L = \{\{\}, \{1, \{3, \{2, \{4, \{5\}\}\}\}\}\}$$

# Neighbourhood Operator

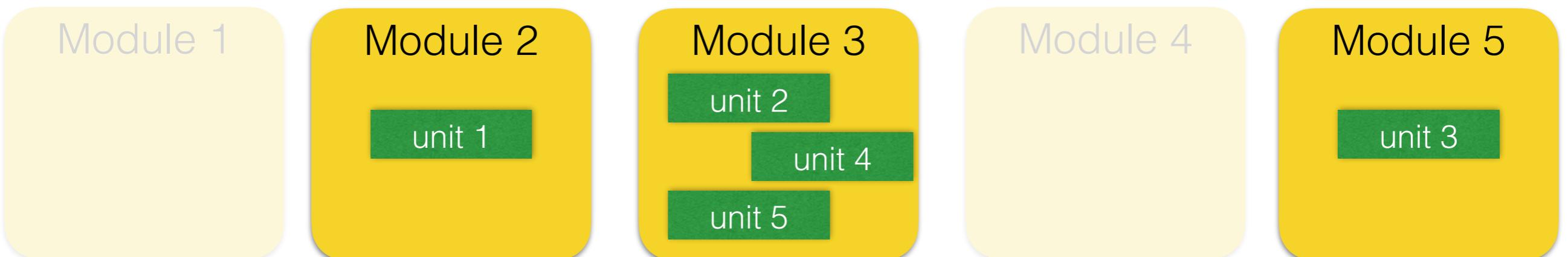
- What would be a possible neighbourhood operator for the software clustering problem?
  - A neighbour in the software module clustering problem would be a solution where a single unit moves from one module to another. E.g.:

$$L = \{\{\}, \{1\}, \{2, 4, 5\}, \{\}, \{3\}\} \longrightarrow L = \{\{\}, \{1, 5\}, \{2, 4\}, \{\}, \{3\}\}$$



# Neighbourhood

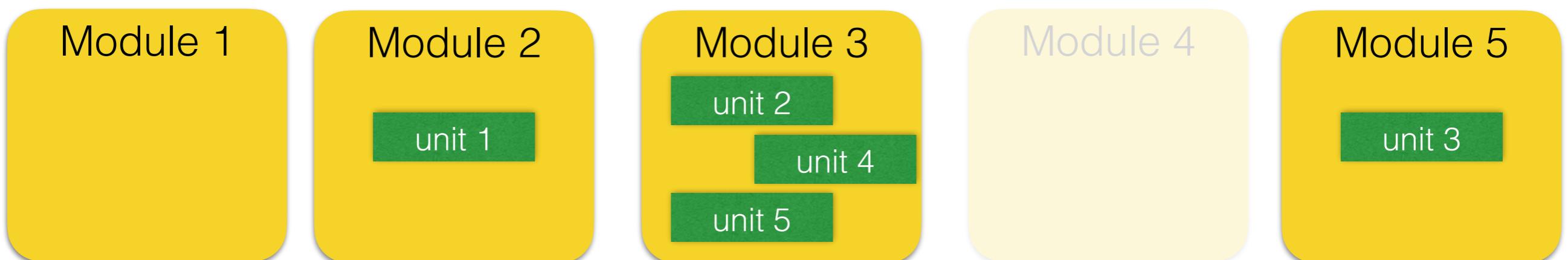
- Real world problems will frequently have more than two neighbours for each candidate solution.
- How many neighbours do we have for the candidate solution below, if we allow for equivalent neighbours?



5 units \* 4 possible modules to move to = 20

# Neighbourhood

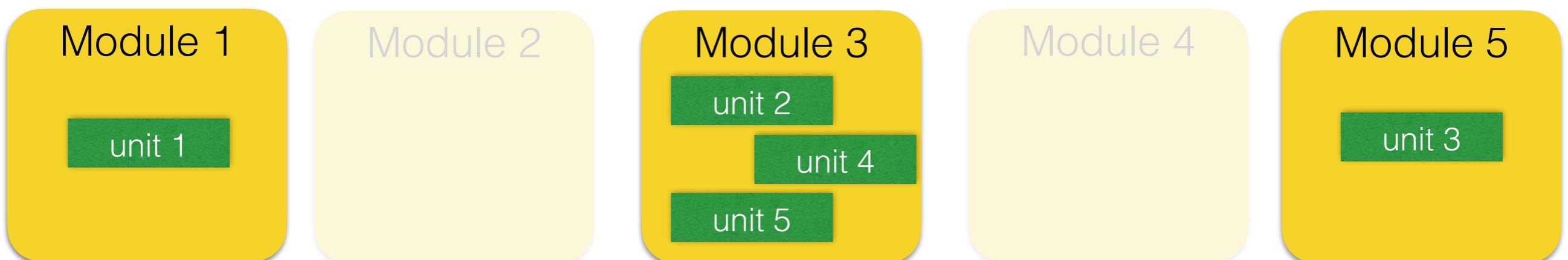
- How many neighbours do we have for the candidate solution below, if we allow for equivalent neighbours?



Some neighbours will be equivalent.  
Duplicates could be eliminated.

# Neighbourhood

- How many neighbours do we have for the candidate solution below, if we allow for equivalent neighbours?



For  $i \in \{1, \dots, N\}$  // module

For  $j \in \{1, \dots, \text{size}(L_i)\}$  // unit within module

For  $i' \in \{1, \dots, N\} \setminus i$  // another module

$L'$  = clone of  $L$

Move unit  $L'_{ij}$  to module  $L'_{i'}$

Yield  $L'$  as a neighbour

# Hill Climbing

Hill-Climbing (assuming maximisation)

1. `current_solution` = generate `initial` solution randomly
2. Repeat:
  - 2.1 generate `neighbour` solutions (differ from current solution by a single element)
  - 2.2 `best_neighbour` = get highest `quality` neighbour of `current_solution`
  - 2.3 If `quality(best_neighbour) <= quality(current_solution)`
    - 2.3.1 Return `current_solution`
  - 2.4 `current_solution` = `best_neighbour`

Until a maximum number of iterations

Simulated Annealing would also require a representation, initialisation procedure, and neighbourhood operator to solve a problem.

# Summary

- Software Module Clustering problem formulation.
- Representation, initialisation and neighbourhood operators.

# Next

- Application of Simulated Annealing.



# Example of Simulated Annealing Application: VLSI Design

Leandro L. Minku

# Simulated Annealing Applications

- Simulated Annealing is applicable to any problem that can be formulated as an [optimisation problem](#).
- Some problems may be easier or more difficult for Simulated Annealing to solve well, and [it may not be possible to know beforehand](#) how well Simulated Annealing will work.
  - This is an issue with many of the Artificial Intelligence algorithms.
- One possibility is to check whether [other similar problems](#) have been well solved well by Simulated Annealing before.
  - This can be helpful, though it [does not guarantee](#) that Simulated Annealing is a good algorithm for the problem in hands.

# Examples of Applications

- Software engineering problems:
    - Component selection and prioritisation for the next release problem.
    - Software quality prediction.
  - Several engineering problems, e.g.: VLSI (Very-Large-Scale Integration).
    - Process of creating an integrated circuit by combining thousands of transistors into a single chip.
    - Decide placement of transistors.
    - Objectives: reduce area, wiring and congestion.
    - Constraints: e.g., maximum area must not be exceeded.



Image from: [https://upload.wikimedia.org/wikipedia/commons/9/94/VLSI\\_Chip.jpg](https://upload.wikimedia.org/wikipedia/commons/9/94/VLSI_Chip.jpg)

# Applying Simulated Annealing (and Hill-Climbing)

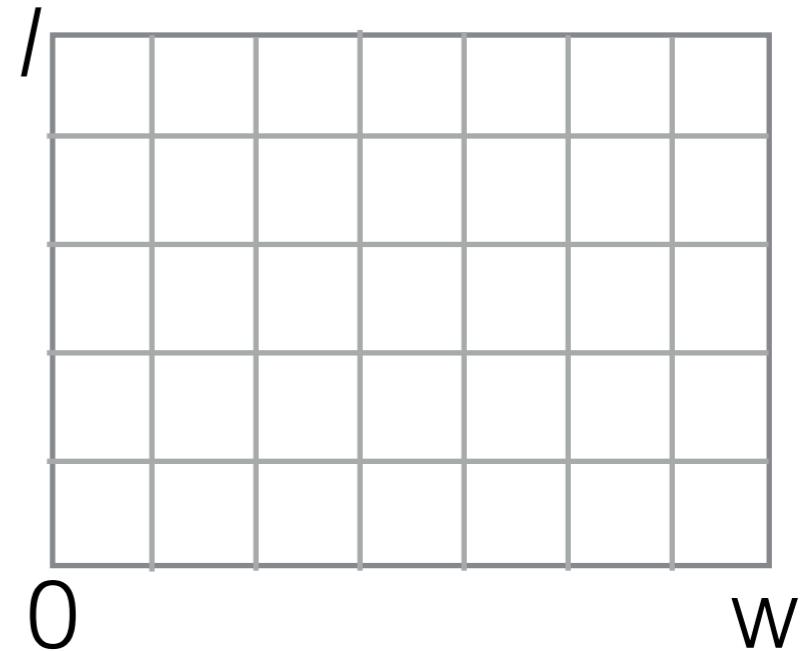
- We need to specify:
  - Optimisation problem formulation:
    - Design variable and search space
    - Constraints
    - Objective function
  - Algorithm-specific operators:
    - Representation.
    - Initialisation procedure.
    - Neighbourhood operator.
  - Strategy to deal with constraints, e.g.:
    - Representation, initialisation and neighbourhood operators that ensure only feasible solutions to be generated.
    - Modification in the objective function.

# Applying Simulated Annealing (and Hill-Climbing)

- We need to specify:
  - **Optimisation problem formulation:**
    - Design variable and search space
    - Constraints
    - Objective function
  - **Algorithm-specific operators:**
    - Representation.
    - Initialisation procedure.
    - Neighbourhood operator.
  - **Strategy to deal with constraints, e.g.:**
    - Representation, initialisation and neighbourhood operators that ensure only feasible solutions to be generated.
    - Modification in the objective function.

# Optimising VLSI Transistor Placement

- Design variable:  
candidate solution  $\mathbf{z} = [(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \dots, (\mathbf{x}_N, \mathbf{y}_N)]$ ,  
where  $N$  is the number of transistors,  
 $\mathbf{x}_i \in \mathbb{Z}$  and  $\mathbf{y}_i \in \mathbb{Z}$  are the coordinates of transistor  $i$ .
- Objectives: minimise area of the chip, length of wiring and congestion.
- Constraints: for all  $i, j \in \{1, \dots, N\}$  where  $i \neq j$ :
  - $x_i \geq 0$
  - $x_i \leq w$
  - $y_i \geq 0$
  - $y_i \leq l$
  - $(x_i, y_i) \neq (x_j, y_j)$



# Applying Simulated Annealing (and Hill-Climbing)

- We need to specify:
  - Optimisation problem formulation:
    - Design variable and search space
    - Constraints
    - Objective function
  - **Algorithm-specific operators:**
    - Representation.
    - Initialisation procedure.
    - Neighbourhood operator.
  - **Strategy to deal with constraints, e.g.:**
    - Representation, initialisation and neighbourhood operators that ensure only feasible solutions to be generated.
    - Modification in the objective function.

# Representation and Initialisation

- **Representation:** direct representation of the design variable candidate solution is a vector of size N of 2-d vectors  $\mathbf{z} = [(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \dots, (\mathbf{x}_N, \mathbf{y}_N)]$ , where N is the number of transistors,  $\mathbf{x}_i \in \mathbb{Z}$  and  $\mathbf{y}_i \in \mathbb{Z}$ ,  $\forall i \in \{1, 2, \dots, N\}$  are the coordinates of transistor i.
- **Initialisation:**
  - Initialise each  $\mathbf{x}_i, \forall i \in \{1, 2, \dots, N\}$ , uniformly at random with values  $0 \leq \mathbf{x}_i \leq w$ .
  - Initialise each  $\mathbf{y}_i, \forall i \in \{1, 2, \dots, N\}$ , uniformly at random with values  $0 \leq \mathbf{y}_i \leq l$ .
- **Neighbourhood operator:** sum or subtract 1 from a given  $\mathbf{x}_i$  or  $\mathbf{y}_i$ .

# Applying Simulated Annealing (and Hill-Climbing)

- We need to specify:
  - Optimisation problem formulation:
    - Design variable and search space
    - Constraints
    - Objective function
  - Algorithm-specific operators:
    - Representation.
    - Initialisation procedure.
    - Neighbourhood operator.
  - **Strategy to deal with constraints, e.g.:**
    - Representation, initialisation and neighbourhood operators that ensure only feasible solutions to be generated.
    - **Modification in the objective function.**

# Dealing With Multiple Objectives

- Design variable:

candidate solution  $\mathbf{z} = [(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \dots, (\mathbf{x}_N, \mathbf{y}_N)]$ ,

where  $N$  is the number of transistors,

$\mathbf{x}_i \in \mathbb{Z}$  and  $\mathbf{y}_i \in \mathbb{Z}$  are the coordinates of transistor  $i$ .

- **Objectives:**

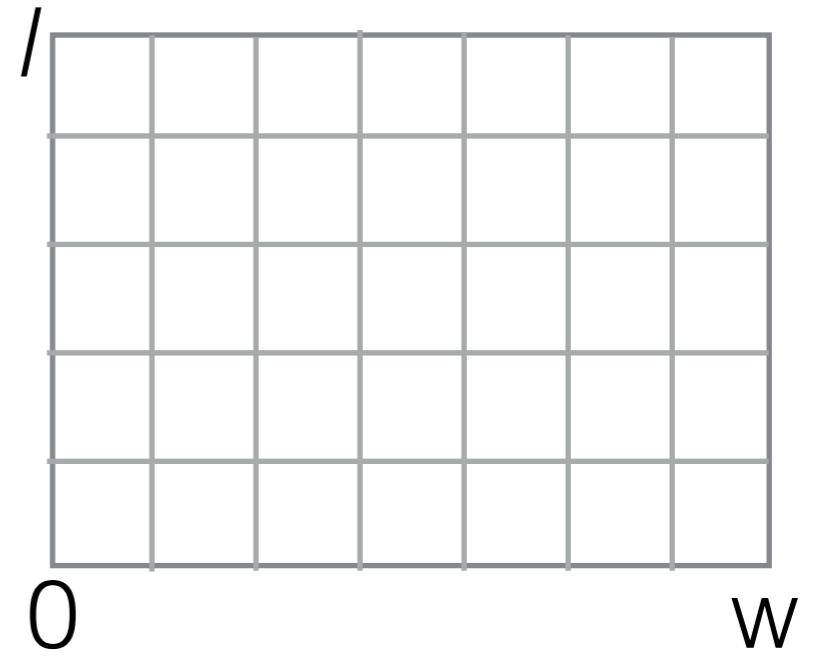
minimise  $f(\mathbf{z}) = (a * \text{area of the chip} + w * \text{length of wiring} + c * \text{congestion})$ ,

where  $a, w, c \in [0, 1]$  and  $a + w + c = 1$ .

- **Constraints:** for all  $i, j \in \{1, \dots, N\}$  where  $i \neq j$ :

- $x_i \geq 0$
- $x_i \leq w$
- $y_i \geq 0$
- $y_i \leq l$
- $(x_i, y_i) \neq (x_j, y_j)$

There are other strategies, including algorithms specifically designed for multiple objectives.



# How to Modify the Objective Function To Deal With Constraints?

- Design variable:

candidate solution  $\mathbf{z} = [(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \dots, (\mathbf{x}_N, \mathbf{y}_N)]$ ,

where  $N$  is the number of transistors,

$\mathbf{x}_i \in \mathbb{Z}$  and  $\mathbf{y}_i \in \mathbb{Z}$  are the coordinates of transistor  $i$ .

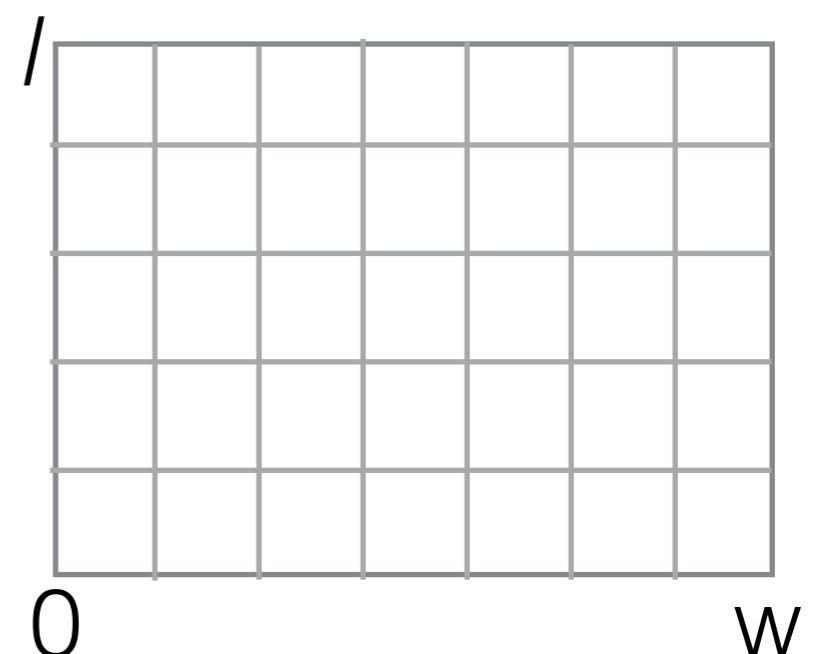
- **Objectives:**

minimise  $f(\mathbf{z}) = a * \text{chip area} + w * \text{wiring length} + c * \text{congestion}$ ,

where  $a, w, c \in [0, 1]$  and  $a + w + c = 1$ .

- **Constraints:** for all  $i, j \in \{1, \dots, N\}$  where  $i \neq j$ :

- $x_i \geq 0$
- $x_i \leq w$
- $y_i \geq 0$
- $y_i \leq l$
- $(x_i, y_i) \neq (x_j, y_j)$



# Generalising The Strategy

Minimise  $f(\mathbf{x})$

Subject to  $g_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, m$

$h_j(\mathbf{x}) = 0, \quad j = 1, \dots, n$

Penalty

Minimise  $f(\mathbf{x}) + Q(\mathbf{x})$

$$Q(\mathbf{x}) = \begin{cases} 0 & \text{if } \mathbf{x} \text{ is feasible} \\ P \times [\underline{g_a(\mathbf{x})^2 + g_b(\mathbf{x})^2 + \dots + h_{a'}(\mathbf{x})^2 + h_{b'}(\mathbf{x})^2 + \dots}] & \text{otherwise} \end{cases}$$

Only sum here the violated constraints

$\underline{\underline{P \times [g_a(\mathbf{x})^2 + g_b(\mathbf{x})^2 + \dots + h_{a'}(\mathbf{x})^2 + h_{b'}(\mathbf{x})^2 + \dots]}}$

where  $P$  is a large positive constant.

# Generalising The Strategy

Minimise  $f(\mathbf{x})$

Subject to  $g_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, m$

$h_j(\mathbf{x}) = 0, \quad j = 1, \dots, n$

Minimise  $f(\mathbf{x}) + Q(\mathbf{x})$

Penalty

$$Q(\mathbf{x}) = P \times [v_{g1}g_1(\mathbf{x})^2 + v_{g2}g_2(\mathbf{x})^2 + \dots + v_{gm}g_m(\mathbf{x})^2 + \\ + v_{h1}h_1(\mathbf{x})^2 + v_{h2}h_2(\mathbf{x})^2 + \dots + v_{hn}h_n(\mathbf{x})^2]$$

where  $P$  is a large positive constant, and  $v_{gi}$  and  $v_{hi}$  are 1 if their corresponding constraint is violated and 0 otherwise.

# Optimising VLSI Transistor Placement

- Design variable:

candidate solution  $\mathbf{z} = [(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \dots, (\mathbf{x}_N, \mathbf{y}_N)]$ ,

where  $N$  is the number of transistors,

$x_i \in \mathbb{Z}$  and  $y_i \in \mathbb{Z}$  are the coordinates of transistor  $i$ .

- Objectives:

minimise  $f(\mathbf{z}) = a * \text{chip area} + w * \text{wiring length} + c * \text{congestion}$ ,

where  $a, w, c \in [0, 1]$  and  $a + w + c = 1$ .

- Constraints: for all  $i, j \in \{1, \dots, N\}$  where  $i \neq j$ :

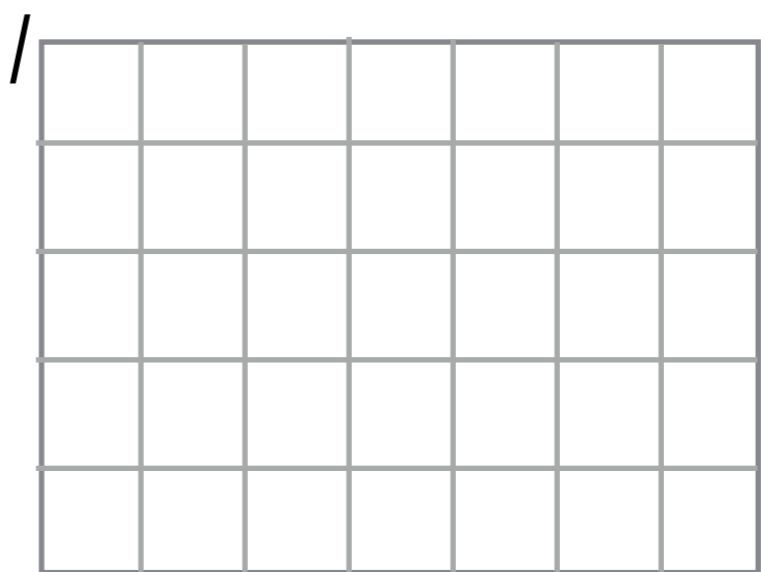
$$\bullet \quad x_i \geq 0 \quad -x_i \leq 0$$

$$\bullet \quad x_i \leq w \quad x_i - w \leq 0$$

$$\bullet \quad y_i \geq 0 \quad -y_i \leq 0$$

$$\bullet \quad y_i \leq l \quad y_i - l \leq 0$$

$$\bullet \quad (x_i, y_i) \neq (x_j, y_j) \quad |x_i - x_j| + |y_i - y_j| \neq 0$$



$$|x_i - x_j| + |y_i - y_j| - 1 \geq 0$$

$$-|x_i - x_j| - |y_i - y_j| + 1 \leq 0$$

# Constraints

Example for N=2 transistors

- $g_1(\mathbf{z}) = -x_1 \leq 0$
- $g_2(\mathbf{z}) = x_1 - w \leq 0$
- $g_3(\mathbf{z}) = -y_1 \leq 0$
- $g_4(\mathbf{z}) = y_1 - l \leq 0$
- $g_5(\mathbf{z}) = -|x_1 - x_2| - |y_1 - y_2| + 1 \leq 0$
- $g_6(\mathbf{z}) = -x_2 \leq 0$
- $g_7(\mathbf{z}) = x_2 - w \leq 0$
- $g_8(\mathbf{z}) = -y_2 \leq 0$
- $g_9(\mathbf{z}) = y_2 - l \leq 0$
- $g_{10}(\mathbf{z}) = -|x_2 - x_1| - |y_2 - y_1| + 1 \leq 0$

# Constraints

Example for N=2 transistors

- $g_1(\mathbf{z}) = -x_1 \leq 0$
- $g_2(\mathbf{z}) = x_1 - w \leq 0$
- $g_3(\mathbf{z}) = -y_1 \leq 0$
- $g_4(\mathbf{z}) = y_1 - l \leq 0$
- $g_5(\mathbf{z}) = -|x_1 - x_2| - |y_1 - y_2| + 1 \leq 0$
- $g_6(\mathbf{z}) = -x_2 \leq 0$
- $g_7(\mathbf{z}) = x_2 - w \leq 0$
- $g_8(\mathbf{z}) = -y_2 \leq 0$
- $g_9(\mathbf{z}) = y_2 - l \leq 0$
- ~~$g_{10}(\mathbf{z}) = -|x_2 - x_1| - |y_2 - y_1| + 1 \leq 0$~~

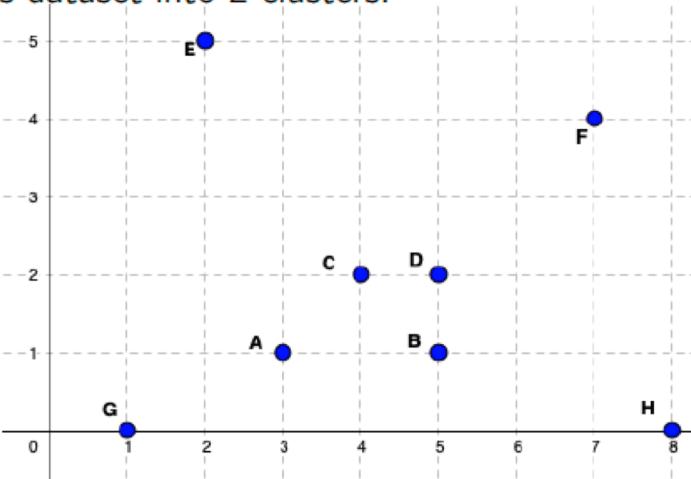
$$Q(\mathbf{z}) = P \times [v_{g1}g_1(\mathbf{z})^2 + v_{g2}g_2(\mathbf{z})^2 + \dots + v_{g9}g_9(\mathbf{z})^2]$$

# Summary

- VLSI transistor placement problem formulation (except for objective function).
- Representation, neighbourhood operator, initialisation procedure.
- Strategy to deal with constraints.

## Question 1 Clustering

We have a dataset with 8 two-dimensional points: A = (3,1), B = (5,1), C = (4,2), D = (5,2), E = (2,5), F = (7,4), G = (1,0), H = (8,0). Use K-means and the Euclidean distance to cluster this dataset into 2 clusters.

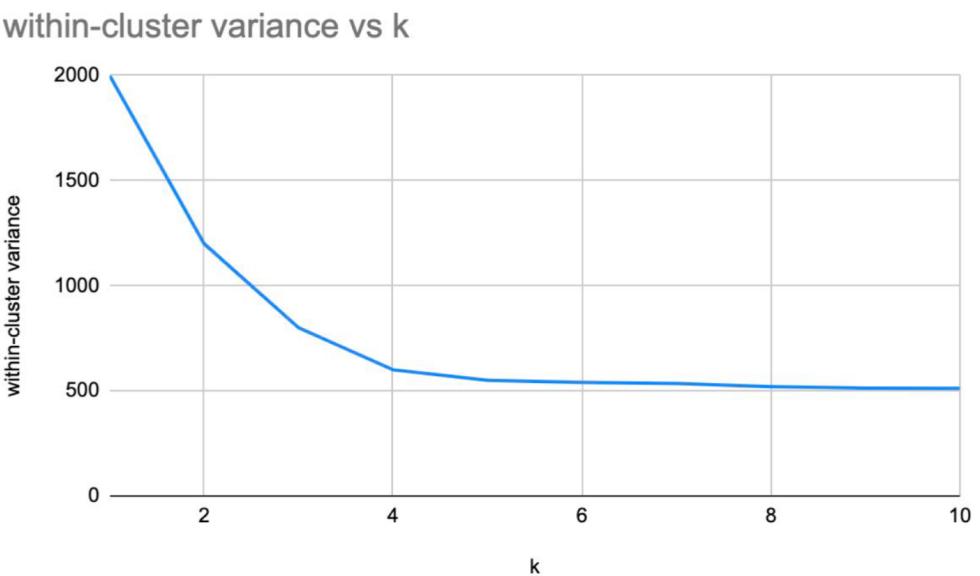


- If the initial cluster centroids are at (1,2) and (1,4), what are the final clusters?  
**Show the step-by-step calculations.** [5 marks]
- If the initial cluster centroids are at (3,4) and (6,4), what are the final clusters?  
**Show the step-by-step calculations.** [5 marks]
- Between the two results from a) and b), which is a better grouping in terms of within-cluster variance? **Justify your answer.**

PS: Within-cluster variance is defined as the sum of squared Euclidean distance between each point and its cluster centroid:  $\sum_{k=1}^K \sum_{x \in C_k} d(x, c_k)^2$ , where  $C_k$  denotes the k-th cluster with centroid  $c_k$ ,  $K$  is the total number of clusters,  $x$  is the data point, and  $d$  is the Euclidean distance between 2 given points. [5 marks]

# Key Concepts -- K-means

- How the algorithm works
  - Assign data points to clusters
  - Calculate new centroids of clusters
- Objective: minimize within-cluster variance



# K-means

## ① Initialization

- Data are  $\mathbf{x}_{1:N}$
- Choose initial cluster means  $\mathbf{m}_{1:k}$  (same dimension as data).

## ② Repeat

### ① Assign each data point to its closest mean

Euclidean distance

$$z_n = \arg \min_{i \in \{1, \dots, k\}} d(\mathbf{x}_n, \mathbf{m}_i) \quad \longrightarrow$$

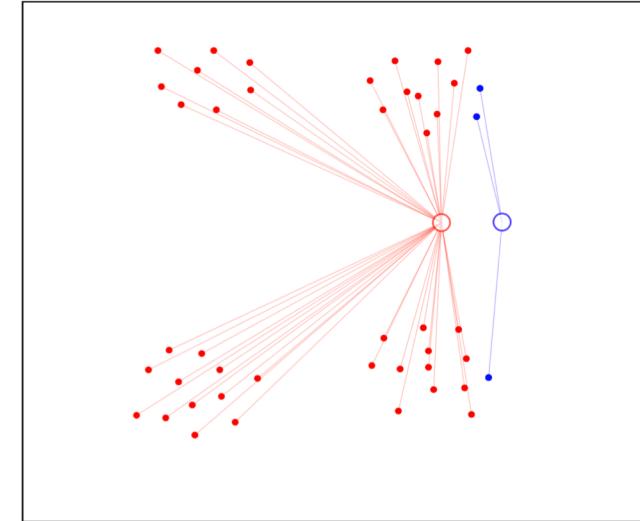
### ② Compute each cluster mean to be the coordinate-wise average over data points assigned to that cluster,

$$\mathbf{m}_k = \frac{1}{N_k} \sum_{\{n : z_n=k\}} \mathbf{x}_n \quad \longrightarrow$$

For each dimension  $j$  of  
 $\mathbf{x}_i$  in cluster  $k$ :

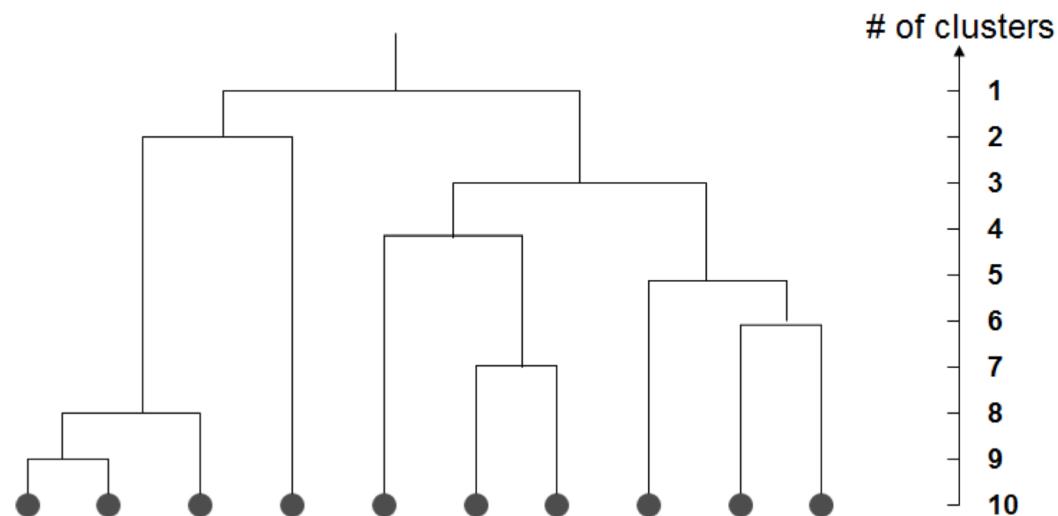
$$(\sum_i x_{i,j}) / N_k$$

### ③ Until assignments $\mathbf{z}_{1:N}$ do not change



# Key Concepts - Agglomerative clustering

- Dissimilarity (distance) measure between clusters:
  - Single linkage, complete linkage, group average
- Dendrogram
  - What is ‘height’
  - How to produce a dendrogram



# Key Concepts - DBSCAN

- 3 types of data points
  - Core, border and noise
- 2 pre-defined parameters
  - epsilon radius and min number of points
- Density reachability directly and indirectly

# DBSCAN

- 1. Label all points as core, border or noise.
- 2. Eliminate noise points.
- 3. For every core point  $p$  that has not been assigned to a cluster:
  - ❖ Create a new cluster with the point  $p$  and all the points that are density-reachable from  $p$
- 4. For border points belonging to more than 1 cluster, assign it to the cluster of the closest core point.

