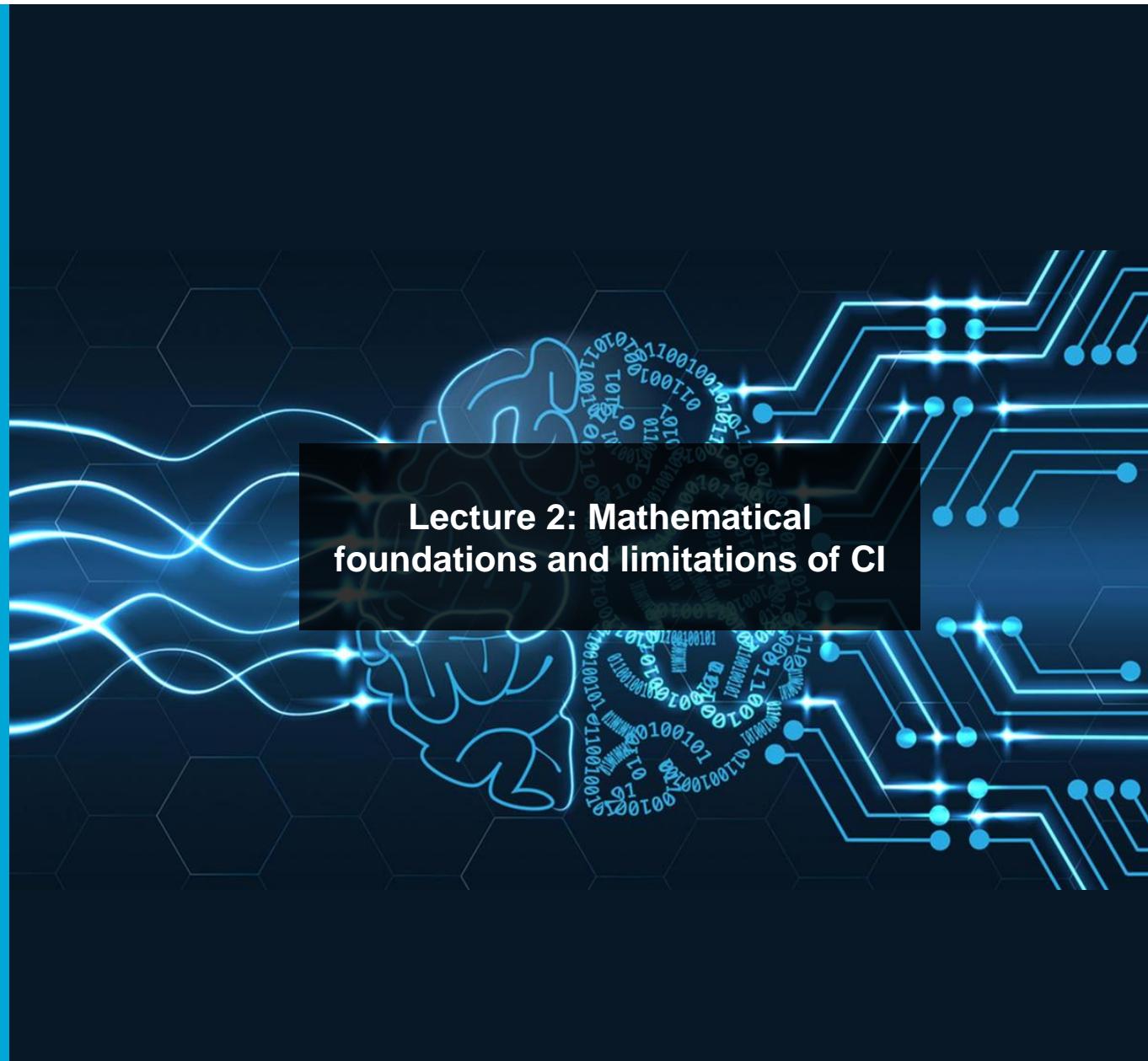


# CSE2530 Computation al Intelligence

Julia Olkhovskaya

(Partial credits to Luciano Cavalcante  
Siebert, Frans Oliehoek, Joost Broekens,  
Judith Redi)



# **What's next? Mathematical foundations (and limitations) of Computational Intelligence**

- CI as optimization
- Fitness functions and fitness landscapes
- Limitations
  
- Problems and learning
- Formalization

# Mathematical basis of CI

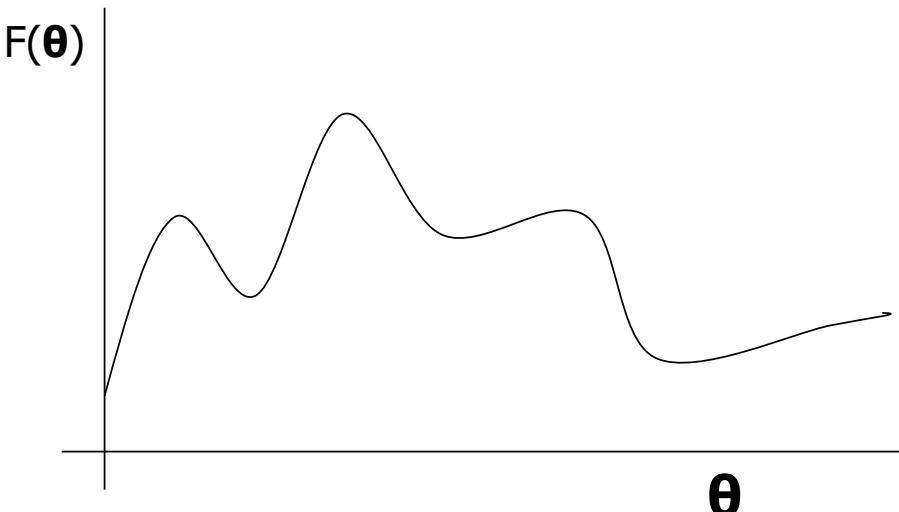
- We focus on a common mathematical basis for CI techniques...
  - ...specifically: the idea that we are somehow **optimizing a function** which corresponds to intelligent behavior
- For instance, things that you will see later in this course:
  - Learning with MLP's by backpropagation
  - Search and optimization with a Genetic Algorithm
  - Search and optimization with an ant colony or a swarm
  - Optimization of action selection with Reinforcement Learning

# CI as optimization

- Both learning and evolution can be thought of as a form of 'optimization':
  - we try to improve something (behavior or organisms) in order to get "better"
- What is 'optimization' in a mathematical sense...?
  - Maximize  $F(\theta)$

Alternatively we could also call it:

- Minimize  $-F(\theta)$



# What is the function we optimize?

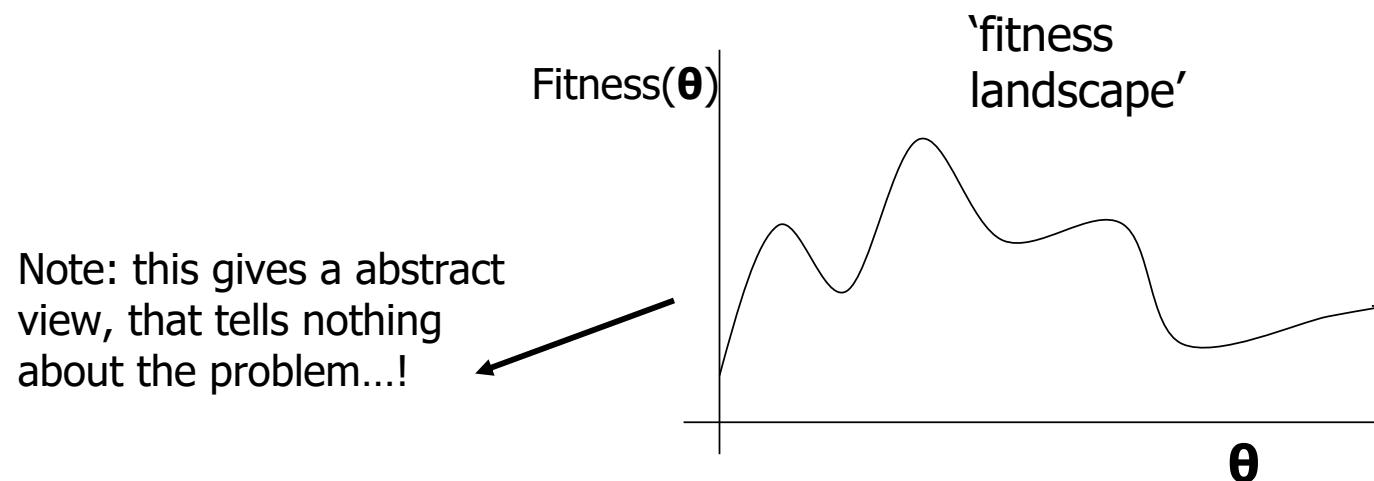
- Idea:  
we may use optimization techniques if we know what function we are trying to optimize...

..but what is this function...?



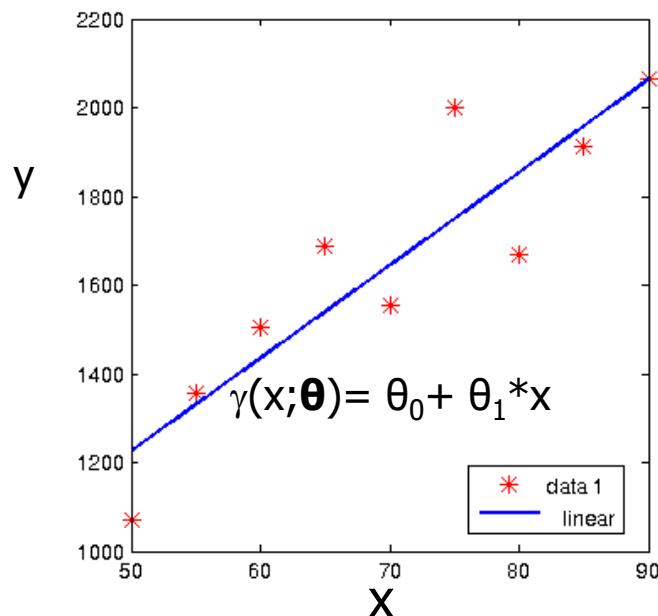
# What is good? (Fitness/error)

- Per definition the **fitness of a candidate**,  $\theta$ , defines:
  - How “good” that state is in solving the “problem”

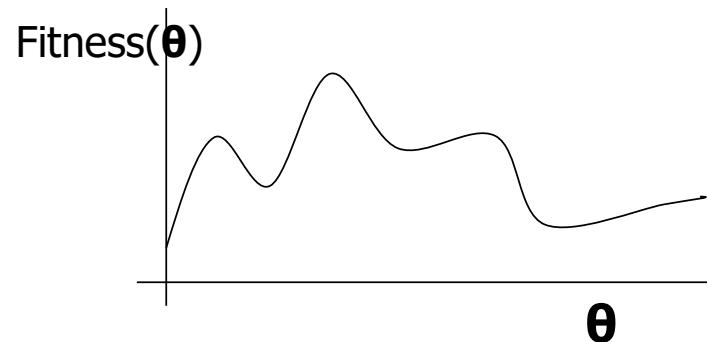


# What is good? (Fitness/error)

- For instance... what is good in a regression problem?
  - So... perhaps define fitness = SSE (sum of the squared differences), which in this case you want to minimize

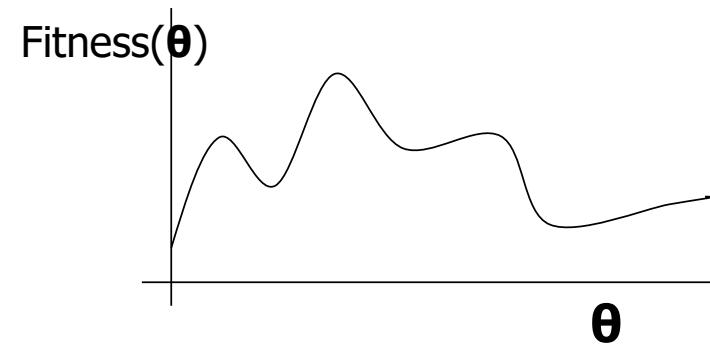
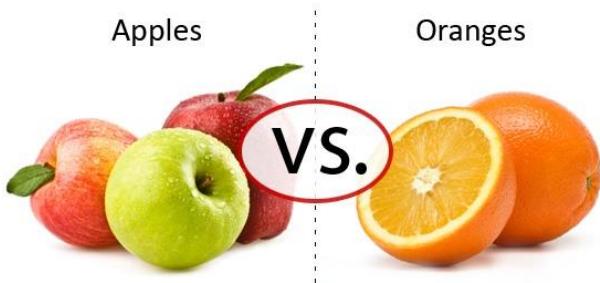


$$SSE(\gamma) = \sum_{i=0}^n [\gamma(x_i) - y_i]^2$$



# What is good? (Fitness/error)

- What is good in a classification problem?
  - So... perhaps define fitness = Cross-entropy (difference between two probability distributions)  
*..measures the performance of a classification model whose output is a probability value between 0 and 1.*

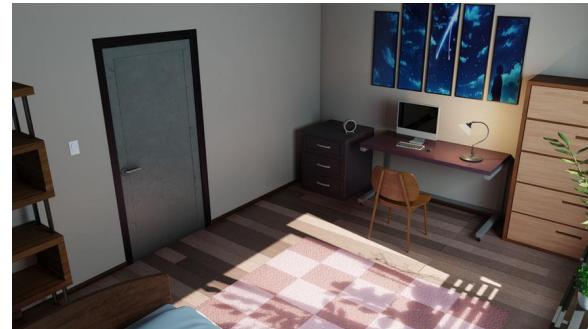


# What is good? (Fitness/error)

- What is good in a search problem?
  - Highest mountain among a set of mountains?

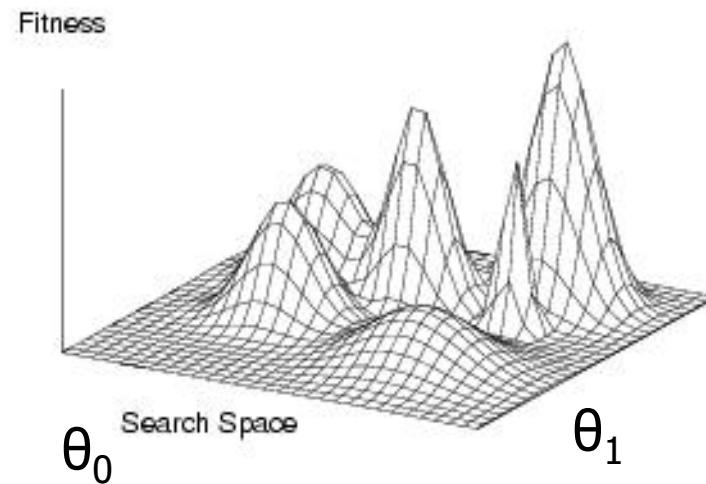


- Room with most space in a building?  
m<sup>2</sup>? m<sup>3</sup>?



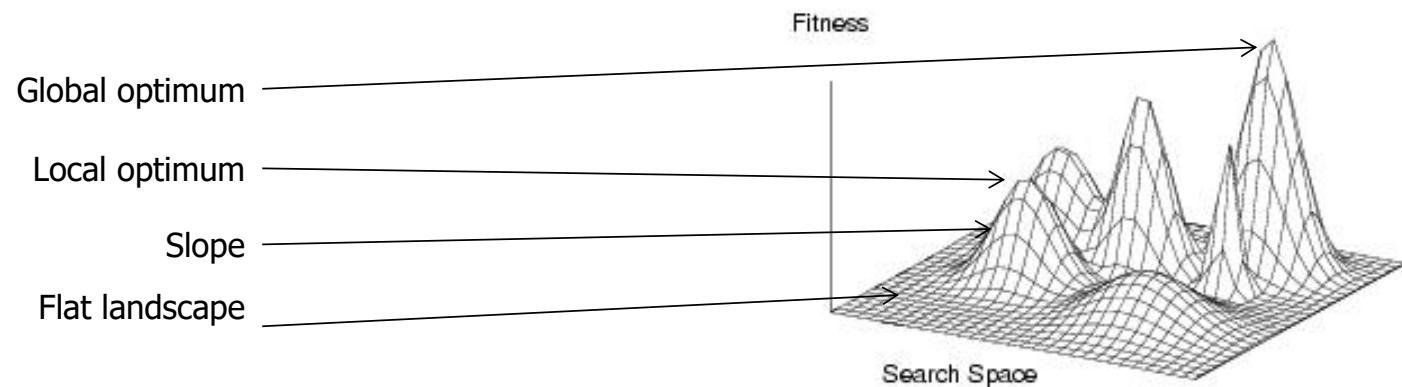
# Fitness in multiple dimensions

- The fitness landscape generalizes to multiple dimensions...
- In general, think of this (fitness landscape):

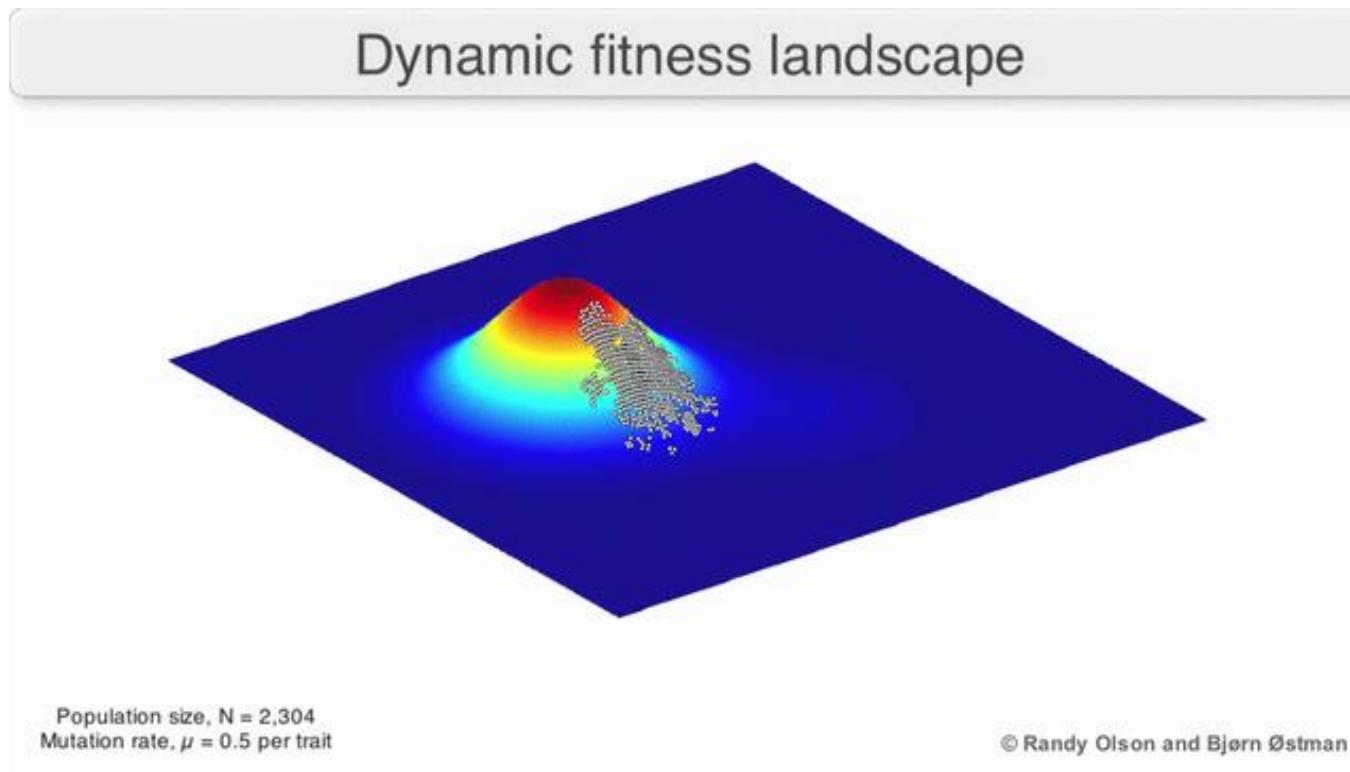


# Fitness in multiple dimensions

- Some concepts in optimization



# Fitness might change through time and context

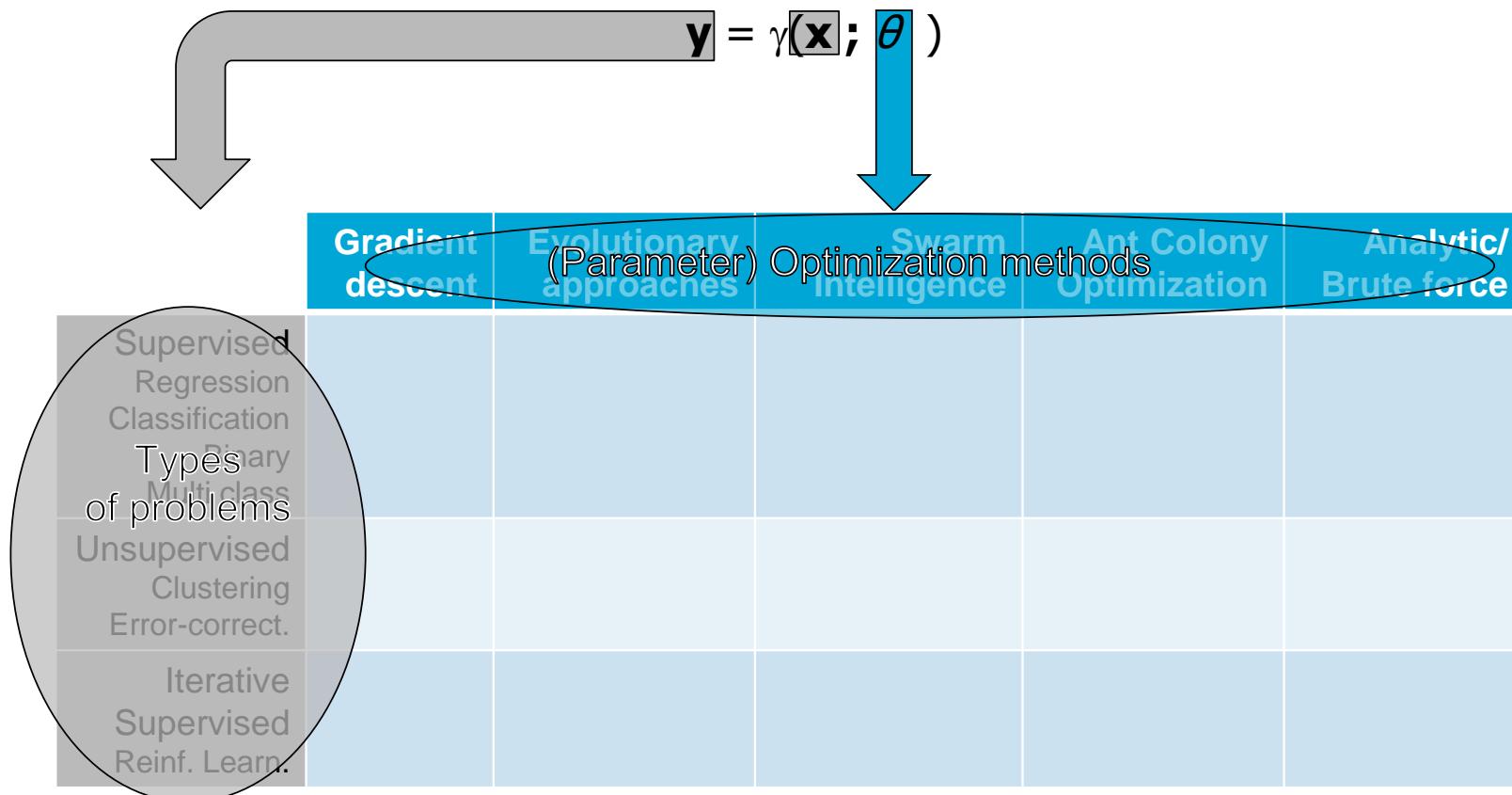


# What is good? (Fitness/error)

- What is the fitness function for selecting a bicycle?
- What is the fitness function for playing chess?
- What should be the fitness function for an automated vehicle to drive safely?
- What is the fitness function of a good employee?
- Many of these questions don't have straightforward answers!
  - Different people will come with different answers
  - AI can solve many problems, but we do need to know what the question is...



# Problems and learning....

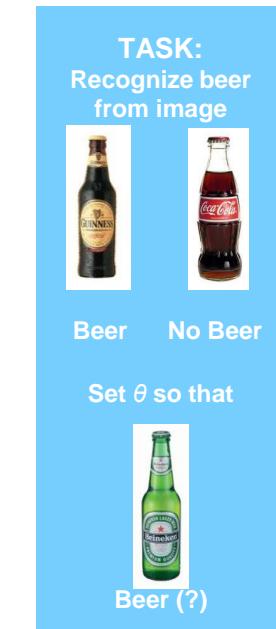


# Supervised learning

- You have a task  $T$  to perform, i.e., link inputs  $\mathbf{x}$  to outputs  $\mathbf{y}$  in some (unknown) domain  $\mathcal{E}$  through  $\gamma: \mathbf{x} \rightarrow \mathbf{y}$
- All you know about  $\mathcal{E}$  is a bunch of examples  $E$  (experience)  
$$E = \{(\mathbf{x}_i, \mathbf{y}_i), i = 1, \dots, p\} \subset \mathcal{E}$$
- A supervised learning technique implements some form of

$$\hat{\mathbf{y}} = \gamma(\mathbf{x}; \theta)$$

Sometimes we also say that  $\gamma$  is a model parametrized by  $\theta$



And learns from the examples in  $E$  (*training*) how to set the parameter vector  $\theta$  so that task  $T$  is performed with a performance  $P$ .

Usually, the larger (and more diverse) is  $E$ , the better is  $P$

# Supervised Learning Problems

- **Binary Classification:** binary target
  - Is it beer or not?  $\gamma : \mathbf{x} \rightarrow y, \quad y \in \{0,1\}$



- **Multi-class Classification:** discrete target
  - Which of  $m$  types of beer is it?

$$\gamma : \mathbf{x} \rightarrow y, \quad y \in \{1,2,\dots,m\}$$



- **Regression:** continuous (n-dimensional) target
  - How many ml of liquid does the bottle contain?

$$\gamma : \mathbf{x} \rightarrow y, \quad y \in \Re$$



given that a set of examples  $E = \{(\mathbf{x}_i, \mathbf{y}_i), i = 1, \dots, p\} \subset \mathcal{E}$  exists

# Unsupervised Learning

- Unsupervised learning: The goal of the machine is to build a model  $\gamma$  that can be used for reasoning, decision making, predicting things, communicating etc.  
However,  $y$  is not given!



**The point is that we want to explore the data to find a more compact representation of latent structure in that data**

Latent → existing but not yet developed or manifest; hidden or concealed

# Unsupervised Learning

- **Clustering:** Finding groups of similar data based on their distribution in the input space



- **Error-correction:** Retrieving specific patterns giving incomplete or noisy ones (association)



given that a set of examples  $E_u = \{(x_i), i = 1, \dots, p\} \subset E_u$  exists

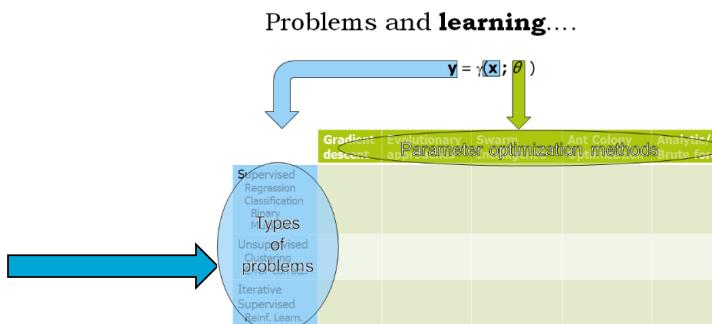
## “Iterative supervised learning problems”

- We don't have the correct  $x \rightarrow y$ , but we do have  $x \rightarrow \tilde{y}$  to approximate  $y$  over batches of learning.
- **Reinforcement Learning:** if  $x$  is the current state and  $y$  is the action to take in that state, then the correct  $y$  is knowable, but not known upfront and will be approximated through exploration.

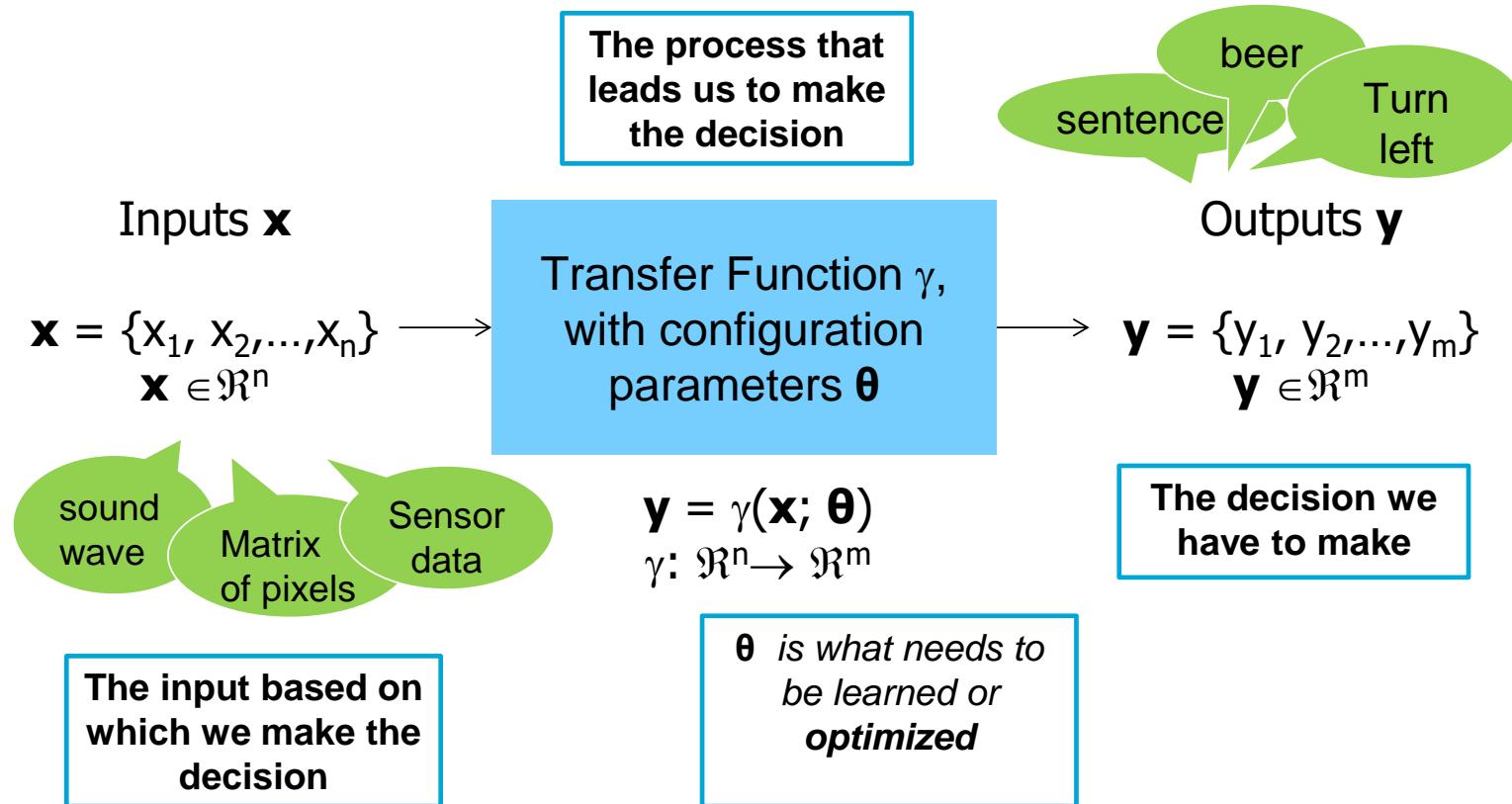


# Warning!

- Textbooks (e.g. Negnevitsky) typically approach Machine Learning and Computational Intelligence as a set of different methods:
  - Artificial Neural Networks
  - Genetic Algorithms
  - Swarm Intelligence
  - Reinforcement Learning
  - Support Vector Machines
  - Bayesian Networks
  - Clustering
  - Deep learning
  - ...
  - And not like this



# The 'transfer function'...

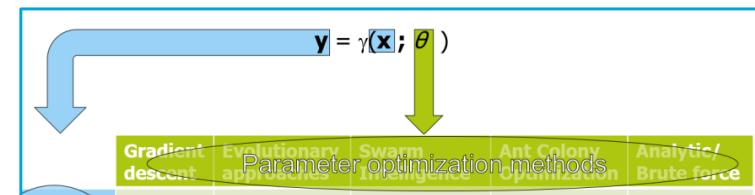


# Formalizing the learning problem

- Want to find a good transfer function:  $y = \gamma(x; \theta)$ 
  - that maps  $x \rightarrow y$
- Each  $\theta$  is a **candidate solution** or “**candidate**”
  - find a  $\theta$  from a set  $T$  of possible  $\theta$ 's
  - $T$  is the (typically large high-dimensional) search space
- What is ‘good’?
  - Find a  $(\theta)$  that induces a)  $\gamma$  that has high “**fitness**” or low **error**

# Finding $\theta$ with high fitness

- How to find a fit candidate  $\theta$  ?
  - Optimization techniques...!
- What optimization techniques do you know?
  - In CI: Gradient descent, random/local/exhaustive search, genetic algorithms, evolutionary optimization, etc.
  - Not covered: linear optimization, convex optimization, nonlinear optimization,...
- Generally: a **guided search process** through a **small subset**



# A general framework for finding $\theta$ with high fitness

- Iterative solution approximation through sampling, goes like:
  - **Build initial sample**  $S$  containing one or more candidate  $\theta$
  - **Decide fitness** for each  $\theta$  in  $S$  based on the
    - **fitness function**, or,
    - **error measure**
  - **Adapt**  $S$  using by changing, adding and removing  $\theta$  's in  $S$  based on :
    - the “**fitness function's derivative**”, or
    - **Probabilities** based on the candidate(s) fitness.
  - GOTO **Decide fitness** until “**stopping criterion**”

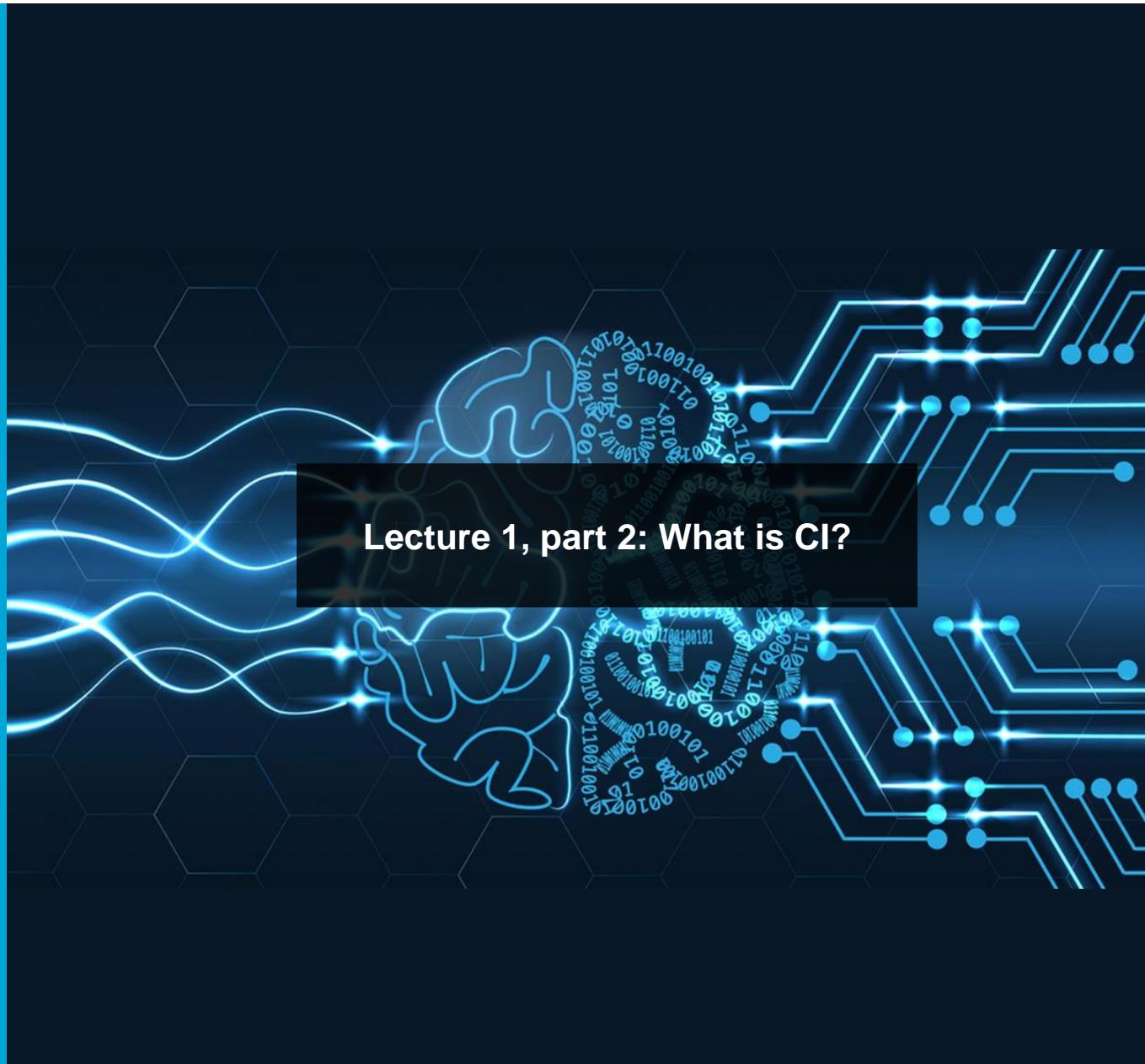
## Next lecture: swarm intelligence

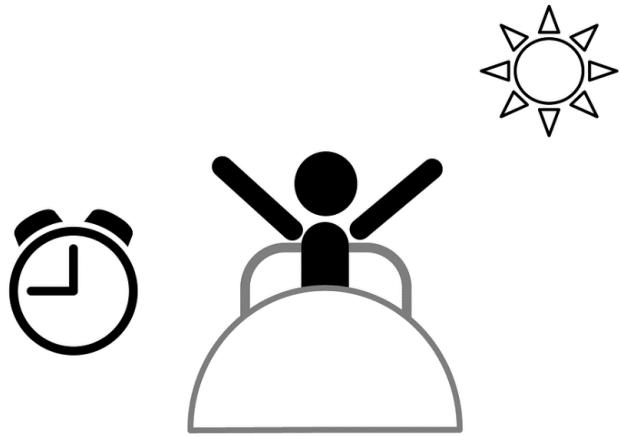
- TODO: find your group!
- Questions?

# CSE2530

## Computational Intelligence

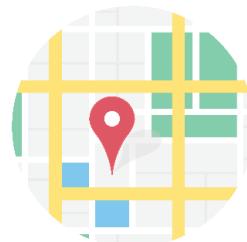
Julia Olkhovskaya, Luciano Cavalcante  
Siebert, Enrico Liscio, Pradeep  
Murukannaiah, and Frans Oliehoek





Unlock your phone with facial recognition +  
Check your social media

AI



Search for the best route to the event you want  
to go

AI



Pay back your share of a restaurant bill from  
last night (e.g. tikkie)

AI



Search for a nice gift to give to your partner

AI

# AI applications



## Healthcare

Social robots, medical diagnosis

## Mobility

Automated vehicles, airplanes

## Retail

Chatbots, virtual assistants, recommender systems

## Agriculture

Soil and crop monitoring, precision farming

## Banking

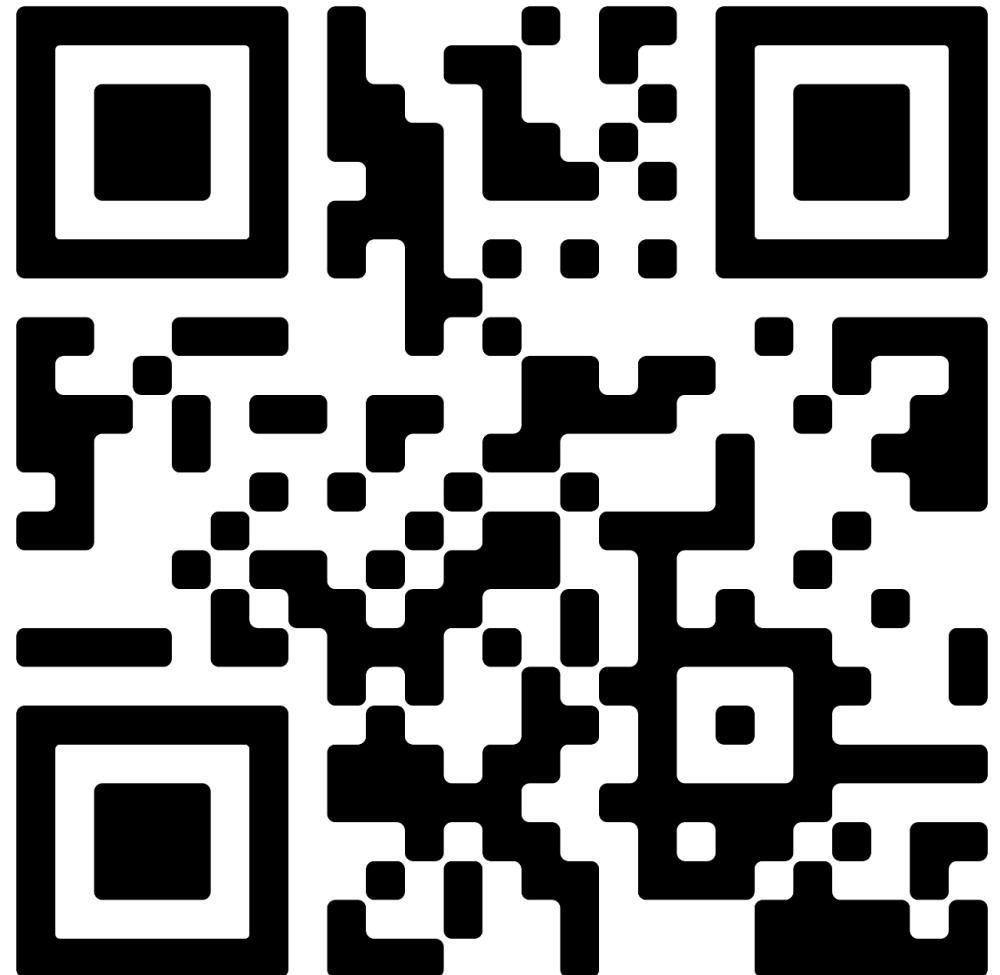
Fraud detection, scan and process documents

AI

AI EVERYWHERE

**Go to vevox.app and use  
the ID: 121-386-518**

**OR Scan the QR code**



# Artificial Intelligence

1 The study of the computations that make it possible to perceive, reason, and act (Winston, 1992)

The study of mental faculties through the use of computational models (Charniak and McDermott, 1985)

**Systems that think rationally**

3 A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes (Schalkoff, 1990)

The branch of computer science that is concerned with the automation of intelligent behavior (Luger and Stubblefield, 1993)

**Systems that act rationally**

2 [The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning..." (Bellman, 1978)

**Systems that think like humans**

4 The art of creating machines that perform functions that require intelligence when performed by people" (Kurzweil, 1990)

The study of how to make computers do things at which, at the moment, people are better (Rich and Knight, 1991)

**Systems that act like humans**

# Artificial Intelligence

- There are multiple definitions of artificial intelligence! No clear consensus...

Some other definitions:

- John McCarthy, who coined the term in 1955, defines it as ‘the science and engineering of making **intelligent machines**’
- Negnevitsky (2005): ‘Make **machines** do things that would **require intelligence** if done by **humans**.’
- Russell and Norvig (1995): `AI strives to **build** intelligent entities as well as understand them.’

## Now, in the 2nd part of this lecture...

Position Computational Intelligence (CI) as a field in Artificial Intelligence (AI)

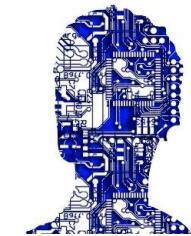
How?

- Define AI
- Define CI as a subclass of AI
- Discuss possible CI techniques

# Reasons for Artificial Intelligence?

**Curiosity:** It could help us understand...

- How the brain works...
- What intelligence is.



**“Laziness” and convenience:** Help us do tasks that we don’t want to do (e.g. they are unsafe) or that makes us bored



**Solve complex problems** that we cannot solve (at least not easily)...

e.g. planning, optimization and search problems

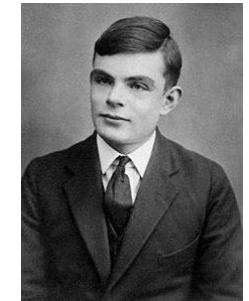
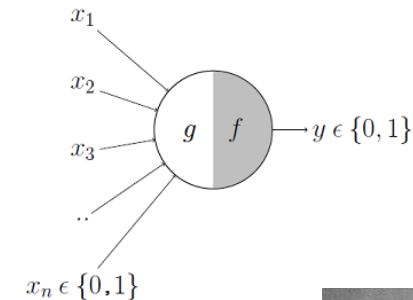
- 1) Search space is too big
- 2) Solutions that are not trivial (e.g. Alpha Go’s Move 37)

etc...

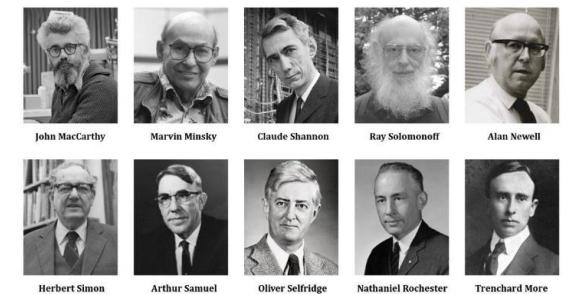


# Origins of AI

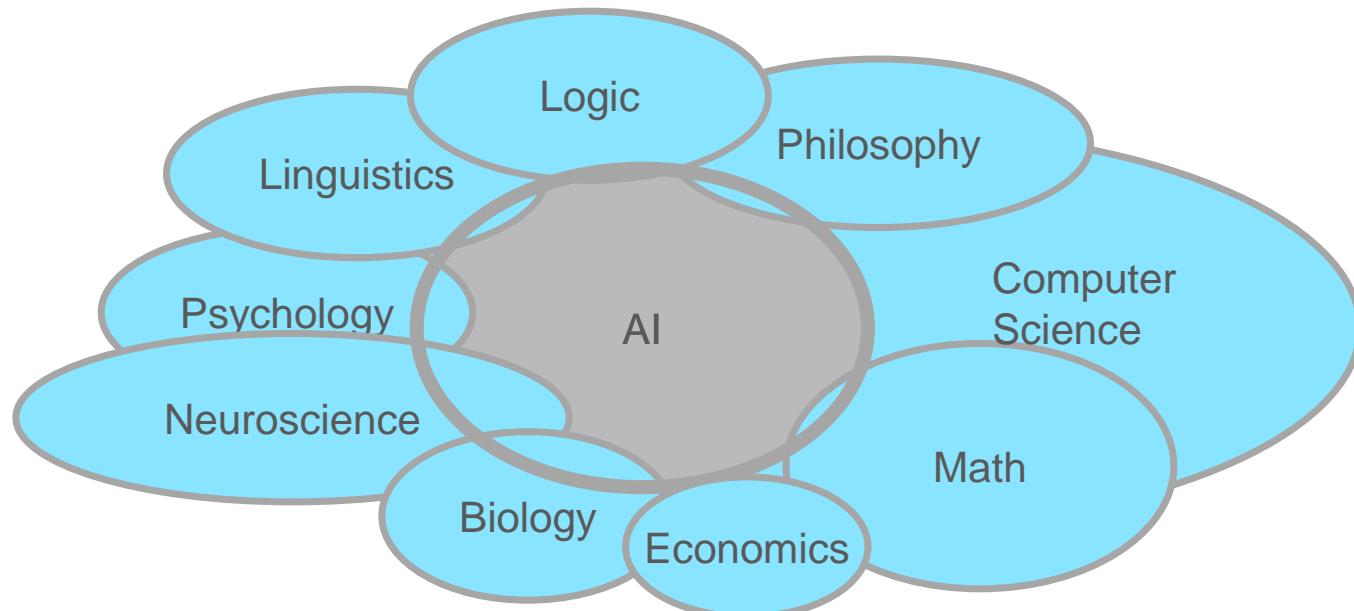
- The birth of artificial intelligence (1943–56)
  - McCulloch and Pitts Neuron (1943)
  - Turing test (1950)
  - The Dartmouth College summer workshop on machine intelligence, artificial neural nets and automata theory (1956).
- Great expectations (1956–late 1960s)
  - LISP – McCarthy
  - Rosenblatt's Perceptron
  - General problem solver (formal logic) - Newell and Simon
- Disillusionment (late 1960s–early 1970s)
- Expert systems (early 1970s–mid-1980s)
- The rebirth of artificial neural networks (1965 – onwards)
- Evolutionary computation (early 1970s–onwards)
- Deep learning revolution (~2012 – onwards)
- What is next???



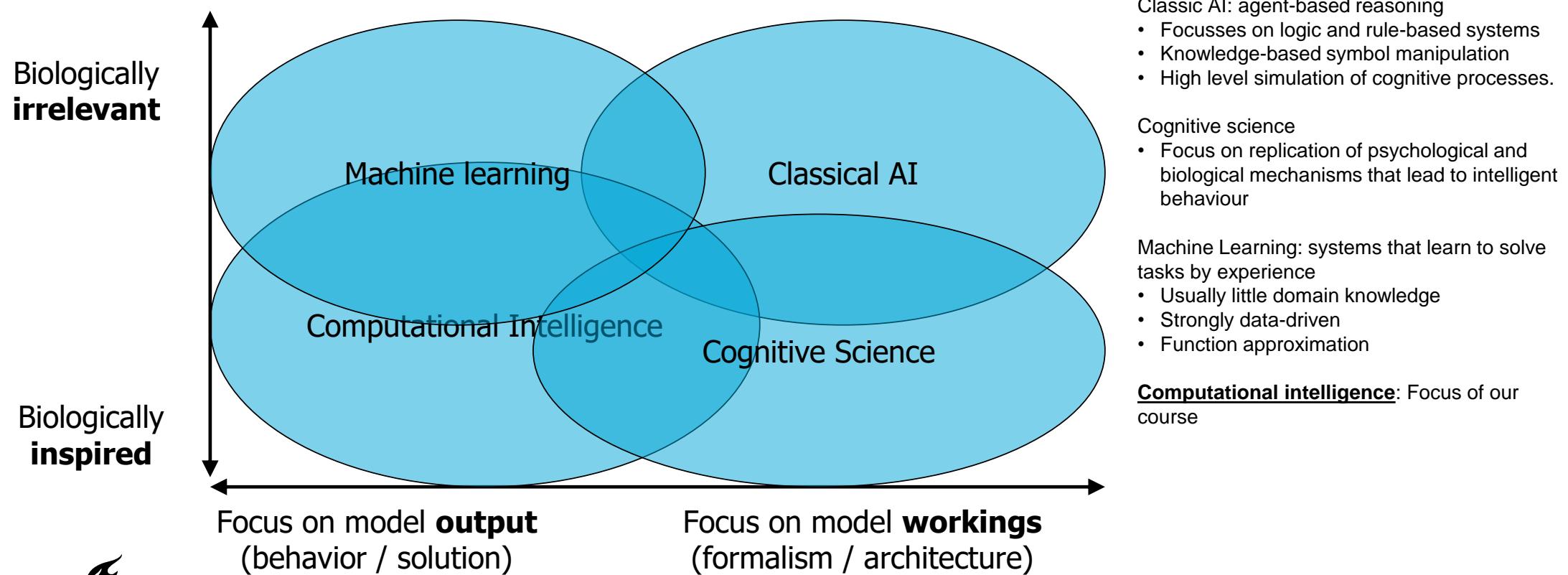
1956 Dartmouth Conference:  
The Founding Fathers of AI



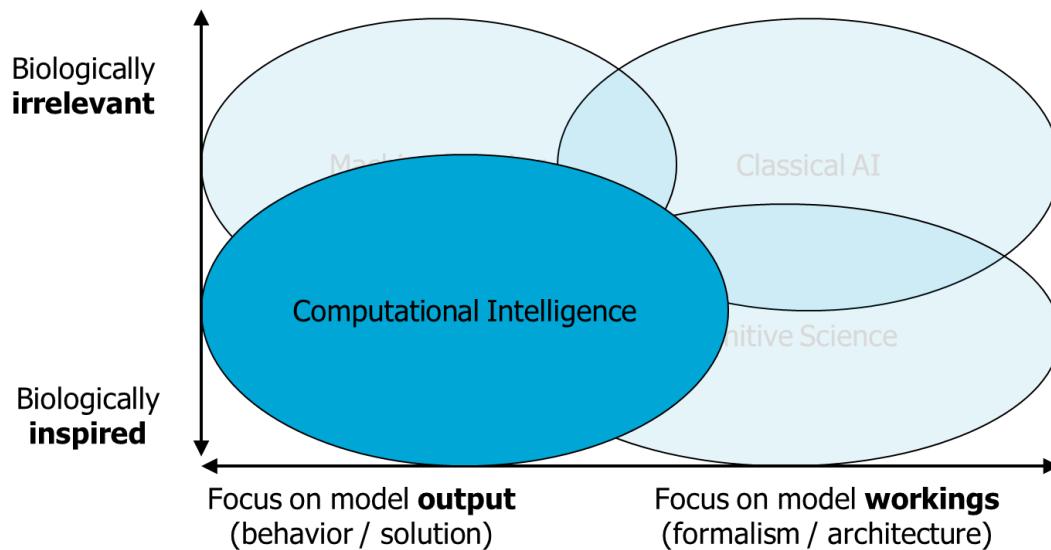
# AI is multidisciplinary



# Different approaches to AI



# Computational Intelligence

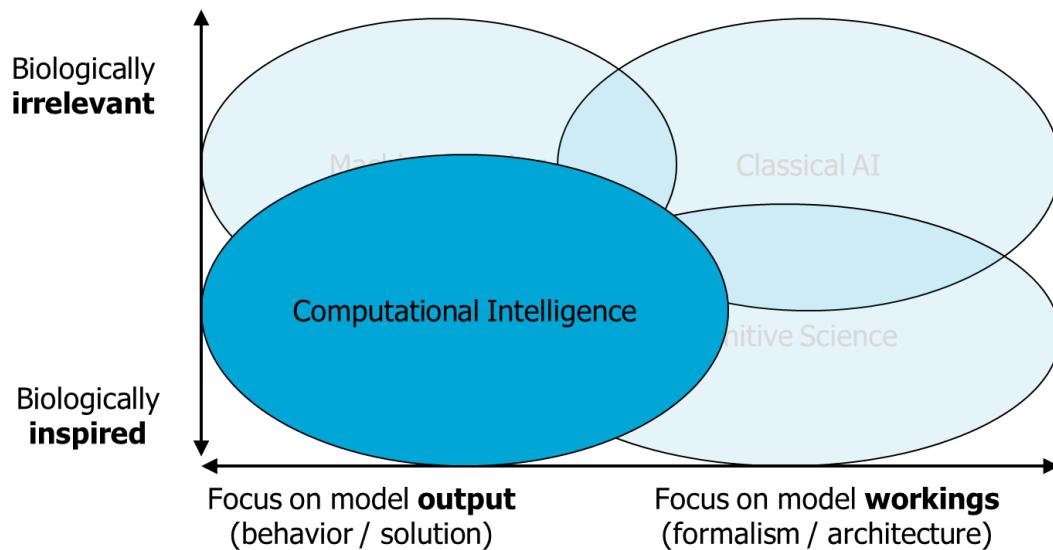


- Biologically inspired:
  - It does not mean that we want to develop systems that are “identical” to its inspiration

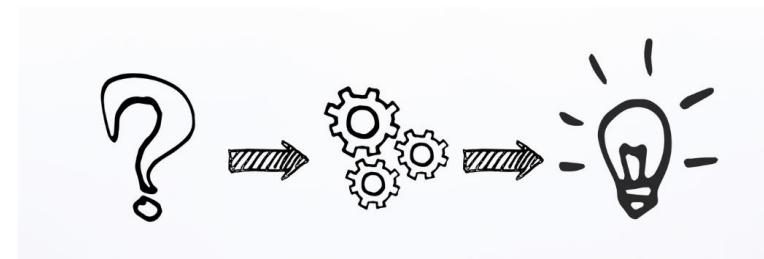


- Usually the algorithms are extremely simplified version of the biological system or concept

# Computational Intelligence



- Focus on model output (behavior / solution):
  - CI focuses on achieving solutions to problems
  - It is not about replicating inner mechanisms
  - For example, it is not about precisely describing how the brain works, but to use a simplified model to solve a problem



# Four perspectives on Computational Intelligence

- The simulation of the workings of the **brain**
- The simulation of **evolution**
- The simulation of **(group) animal behavior**
- The simulation of **animal learning**

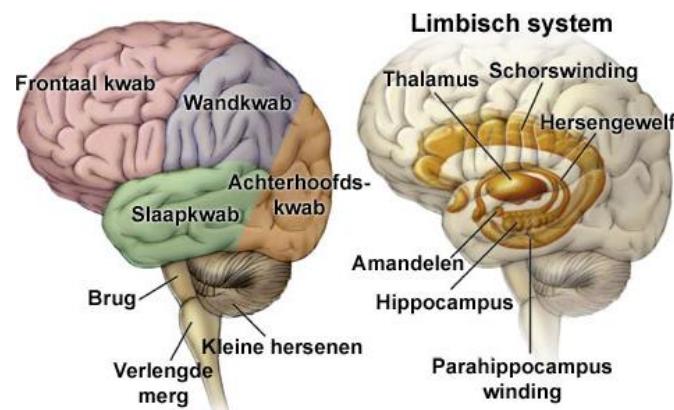
# The simulation of the workings of the brain

- Neural Networks that learn from examples and are able to predict the correct output for input never seen before → **Artificial neural networks**.

## Brains

- Neural networks

Anatomie van de hersenen

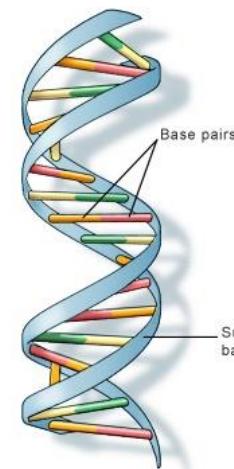
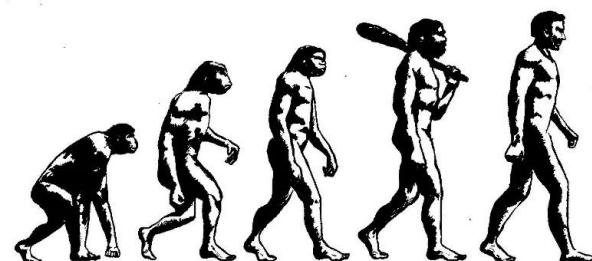


# The simulation of evolution

- Techniques that are inspired by evolution and can find solutions to complex problems, adaptively learning how to deal with them → **Evolutionary computing**.

## Evolution

- Evolutionary computation



# The simulation of (group) animal behavior

- Ants, bees, birds and other animals as inspiration for systems that can solve complex problems based on simple behavior of many agents → **Swarm intelligence**.



# The simulation of animal learning

- The combination of learning by trial and error originated in the psychology of animal learning with the problem of optimal control and its solution using value functions and dynamic programming → **Reinforcement learning**.



<https://www.cs.upc.edu/~mmartin/url-RL.html>

## Activity – Wheel of CI (1/2)

- You will work on this activity with your “neighbor” (if needed you can make groups of 3)
- Everyone take their campus card (student card) and look at the two last digits of your student number (in the example, **67** and **21**). The smaller two-digit number among all students of the group will be the group’s number (in the example, **21**)
  - First digit (in the example, **2**) → Select the indicated row in Table 1  
Second digit (in the example, **1**) → Select the indicated row in Table 2.

Table 1	
Number	Description
0-1	Dog(s)
2-3	Bat(s)
4-5	SARS-Cov-2
6-7	Immune system
8-9	Big bang

Table 2	
Number	Description
0-1	Find the best route from city A to city B.
2-3	Learn to play tic tac toe.
4-5	Recommend a person which youtube tutorials to watch given a simple task.
6-7	Determine if a given photo is of an airplane or a bus.
8-9	Determine when to automatically adjust the radiators in a house.

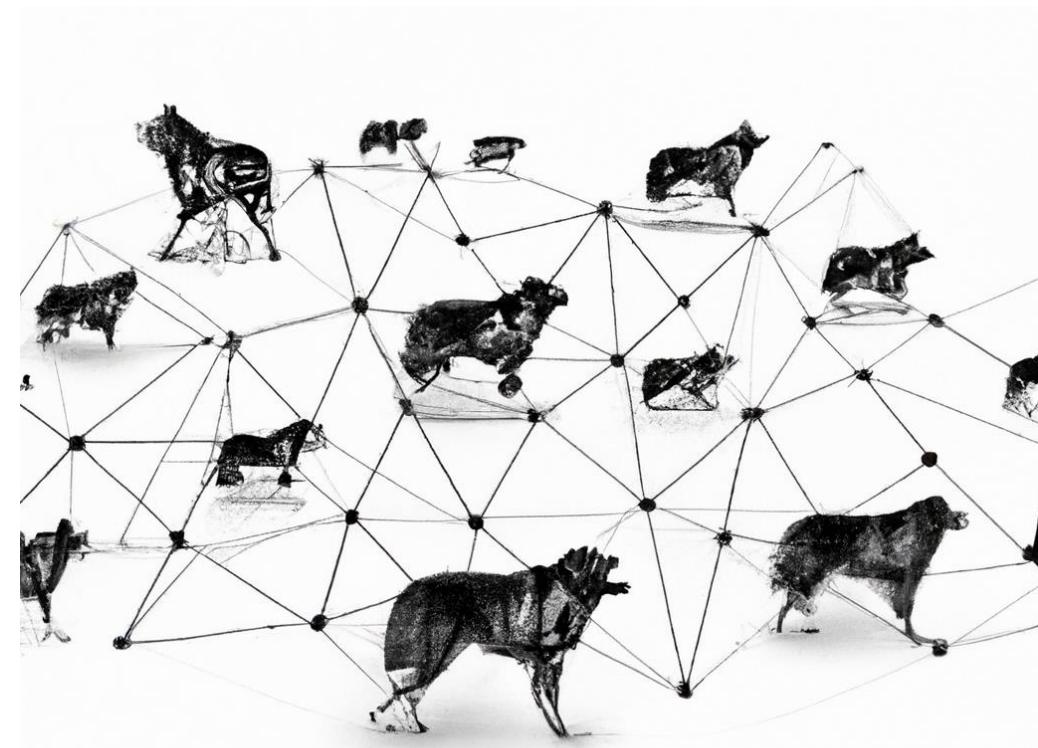
- Propose a CI technique that is inspired by “Table 1” and solves the problem on “Table 2”. Discuss in the group and write a short description. How does it work? What kind of problems can it solve? What are the strong points? What are possible weaknesses?



## Activity – Wheel of CI (2/2)

Example:

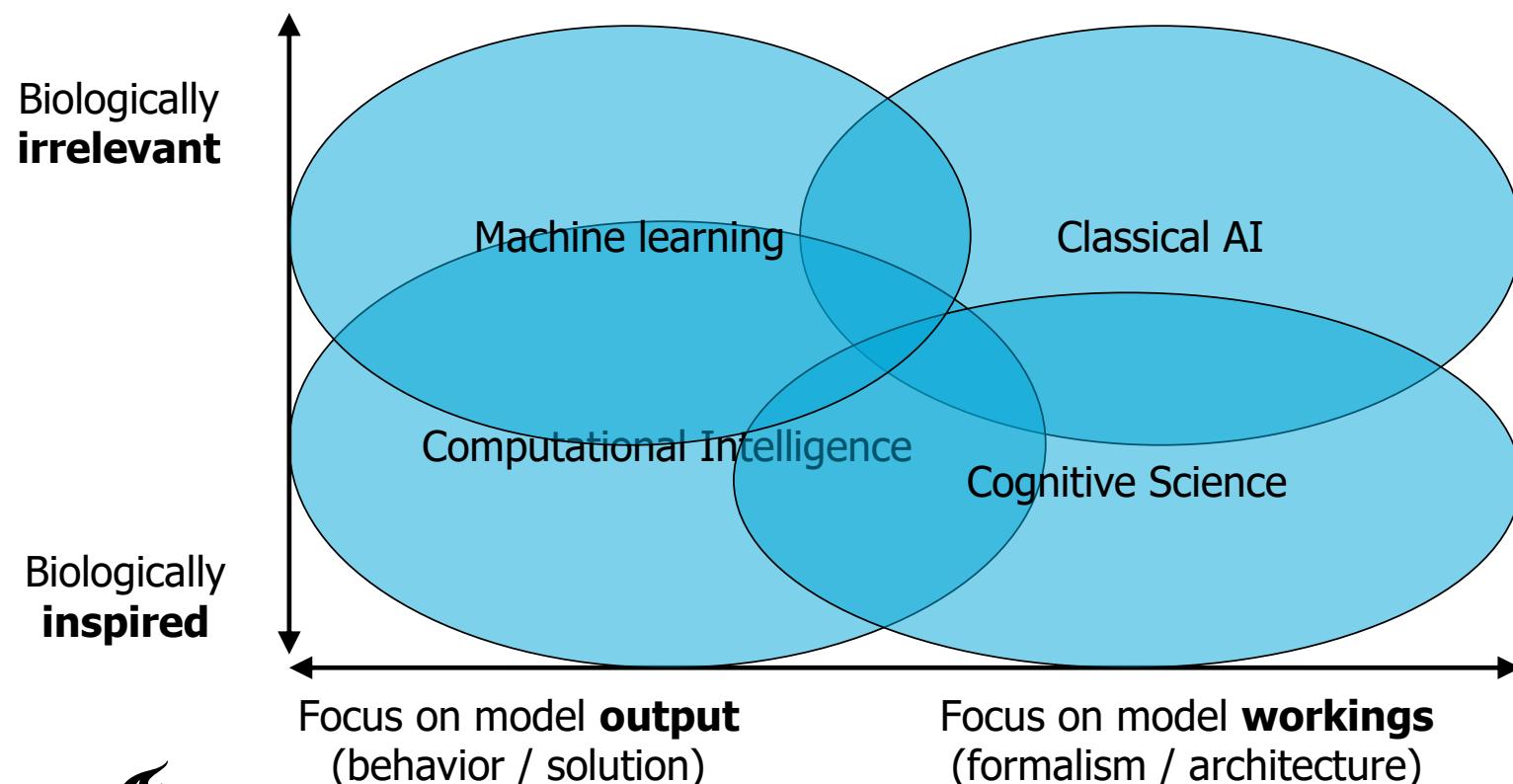
*"This CI technique, called "Wolf Pack Algorithm" is inspired by how **wolves** search for food in the wild, by following the leader of the pack and howling to each other. It can be used to **determine the best location to build a new wind farm** in a given area, being the wind and another beneficial factors for a wind turbine the "food" that the algorithms is searching for."*



Vock Motf Plack

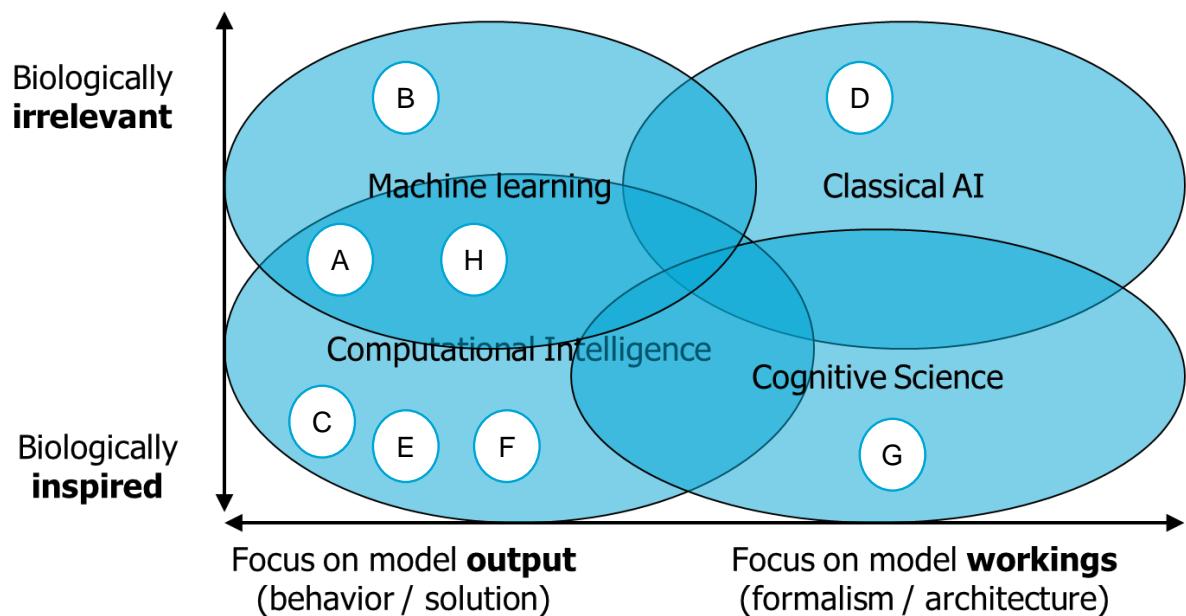
By Dall-E 2, prompt "Wolf pack AI algorithm"

## Quick recap...



# Quick check

- A. Artificial neural networks (ANNs): Computing systems inspired by the biological neural networks that constitute animal brains. They can be used to classify data, regression analysis, and to organize data in similar categories.
- B. Support Vector Machine (SVM): Uses statistical learning frameworks to assign new examples from a dataset to one of two categories. For this, it uses as an input a set of labeled training examples.
- C. Genetic Algorithm (GA): A metaheuristic inspired by the process of natural selection which are commonly used to generate high-quality solutions to optimization and search problems.
- D. Expert systems: Computer system emulating a well-known body of knowledge through if-then rules.
- E. Ant Colony Optimization (ACO): Probabilistic technique inspired by the behavior of real ants for solving computational problems which can be reduced to finding good paths through graphs.
- F. Particle Swarm Optimization (PSO): Computational method that iteratively tests the quality of multiple possible solutions (particles) and, inspired by how bird flocks move, tries to reach better and better solutions.
- G. Dual process theory: Provides an account of how thought can arise in two different ways, namely through fast, intuitive reactions (system 1) or in focused reasoning or analysis (system 2).
- H. Reinforcement Learning (RL): Techniques that aim to learn how agents ought to take actions in an environment in order to maximize the notion of cumulative reward, in a process similar to what appears to occur in animal psychology.



## Recap: today we talked about

- Artificial Intelligence
- Computational Intelligence
- Four perspective on Computational Intelligence
- Next: Mathematical foundations (and limitations) of Computational Intelligence

# CSE2530 Computational Intelligence

Julia Olkhovskaia



# Today we are going to talk about

1st part:

- Course organization

2nd part:

- AI & Computational intelligence
- Math foundations

# Instructors



**Luciano C. Siebert**  
[L.CavalcanteSiebert@tudelft.nl](mailto:L.CavalcanteSiebert@tudelft.nl)



**Sietze Kuilman**  
[s.k.kuilman@tudelft.nl](mailto:s.k.kuilman@tudelft.nl)



**Pradeep Murukannaiah**  
[P.K.Murukannaiah@tudelft.nl](mailto:P.K.Murukannaiah@tudelft.nl)



**Frans Oliehoek**  
[F.A.Oliehoek@tudelft.nl](mailto:F.A.Oliehoek@tudelft.nl)



**Julia Olkhovskaya**  
[I.M.Olkovskaia@tudelft.nl](mailto:I.M.Olkovskaia@tudelft.nl)

## Teaching team:

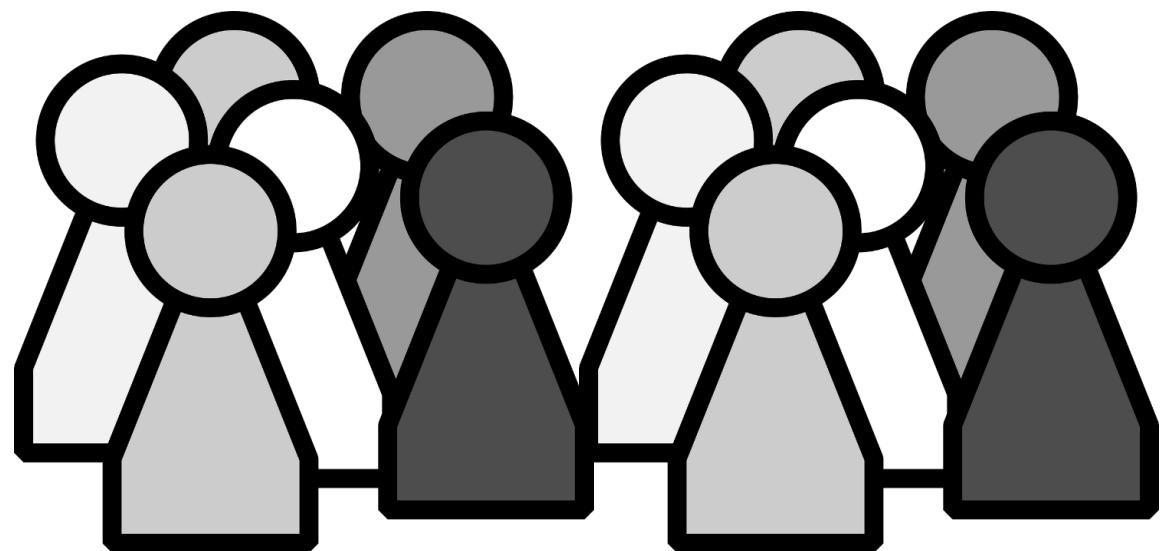


**Thomas Overklift**  
[M.SuaudeCastro@tudelft.nl](mailto:M.SuaudeCastro@tudelft.nl)

# Teaching assistants

- **Teaching assistants:**

- Stefan Minkov
- Marius Frija
- Bernadett Bakos
- Aleksander Buszydlik
- Oana Milchi
- Vasko Dakov
- Dimana Stoyanova
- Valentin Mihăilă
- Daniel Popovici
- Kaj Neumann
- Rareş Biteş



# Course organization

- Brightspace for all important announcements

- 2 lectures per week

- This week:

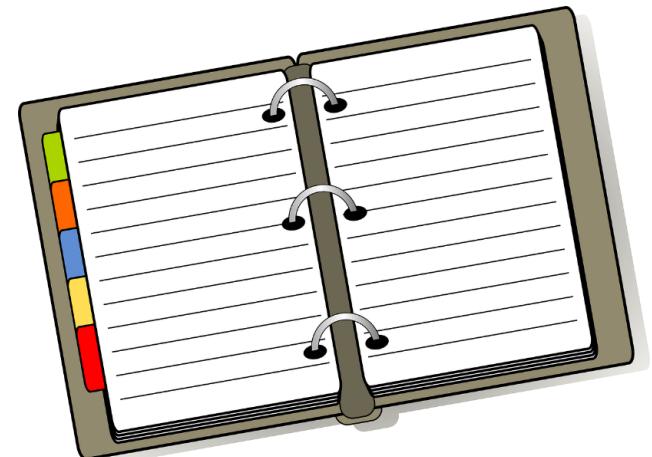
- Monday            8:45 – 10:30
- Wednesday        8:45 – 10:30

- After:

- Monday            8:45 – 10:30
- Wednesday        13:45 – 10:30

- 1 lab session per week (on campus)

- Tuesday          08:45 – 15:45



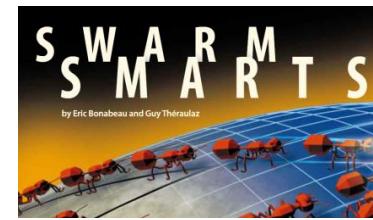
# Learning outcomes

After completing this course, you should be able to:

- LO1: Position Computational Intelligence (CI) as a field in AI
- LO2: Explain concepts such as problem space, fitness, and optimization
- LO3: Understand the working and limitations of CI techniques
- LO4: Select the most appropriate CI technique for a given problem
- LO5: Apply Reinforcement Learning techniques for a given problem
- LO6: Apply different types of neural networks for a given problem
- LO7: Apply evolutionary and swarm techniques to solve given problems

# Study material

- Negnevitsky, M. (2005). Artificial intelligence: a guide to intelligent systems. Third edition. Pearson education.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press. Available at: <https://www.deeplearningbook.org/>
- Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press. Available at: <http://incompleteideas.net/sutton/book/ebook/the-book.html>
- Brightspace : Other readings**
- Brightspace: Lecture slides**
- Collagerama: Recordings**



Nat Comput (2008) 7:109–124  
DOI 10.1007/s11047-007-9050-z

A review of particle swarm optimization. Part II:  
hybridisation, combinatorial, multicriteria and  
constrained optimization, and indicative applications

Alec Banks · Jonathan Vincent · Chukwudi Anyakoha

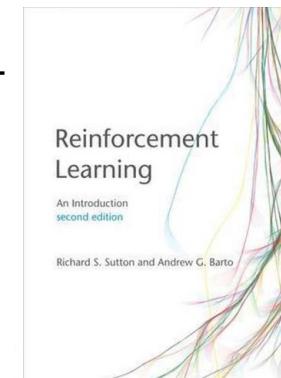
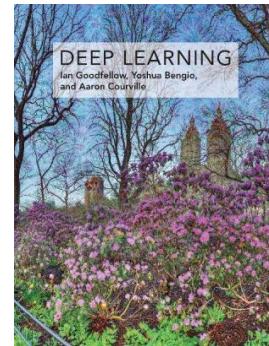
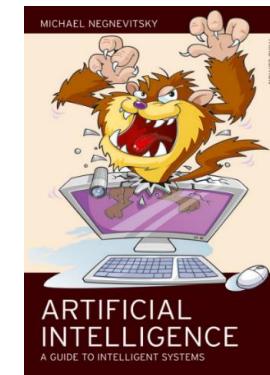
Received: 25 August 2006 / Accepted: 4 June 2007 / Published online: 17 July 2007  
© Springer Science+Business Media B.V. 2007

**Abstract** Particle Swarm Optimization (PSO), in its present form, has been in existence for roughly a decade, with formative research in related domains (such as social modelling, computer graphics, simulation and animation of natural swarms or flocks) for some years before that: a relatively short time compared with some of the other natural computing

Particle Swarm Optimization  
James Kennedy<sup>1</sup> and Russell Eberhart<sup>2</sup>  
<sup>1</sup>Washington, DC 20212  
kennedy\_jim@blu.gov  
<sup>2</sup>Purdue School of Engineering and Technology  
Indianapolis, IN 46202-5160  
eberhart@engr.psu.edu

**ABSTRACT**  
A concept for the optimization of nonlinear functions using particle swarm methodology is introduced. The evolution of several paradigms is outlined, and an implementation of one of the paradigms is discussed. Benchmark testing of the paradigm is described, and applications, including nonlinear function optimization and neural network training, are proposed. The relationships between particle swarm optimization and both artificial life and genetic algorithms are described.

**1 INTRODUCTION**  
This paper introduces a method for optimization of continuous nonlinear functions. The method was discovered through simulation of a simplified social model; thus the social metaphor is discussed, though the algorithm stands without metaphorical support. This paper describes the particle swarm



## Neural Networks and Deep Learning

*Neural Networks and Deep Learning* is a free online book. The book will teach you about:

- Neural networks, a beautiful biologically-inspired programming paradigm which enables a computer to learn from observational data
- Deep learning, a powerful set of techniques for learning in neural networks

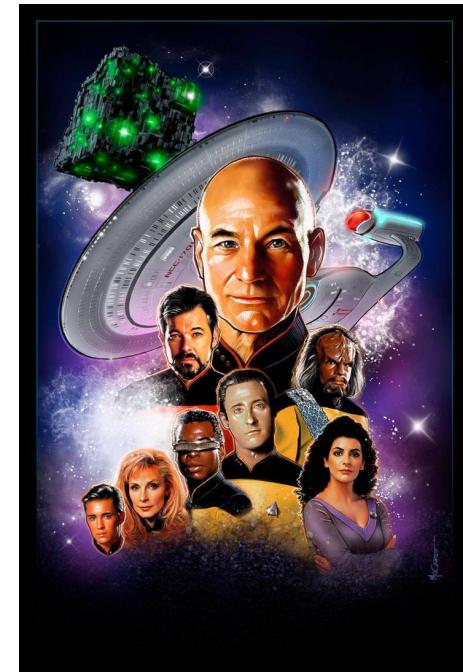
Neural networks and deep learning currently provide the best solutions to many problems in image recognition, speech recognition, and natural language processing. This book will teach you many of the core concepts behind neural networks and deep learning.

**Neural Networks and Deep Learning**  
What this book is about  
On the exercises and problems  
• Using neural nets to recognise handwritten digits  
► How the backpropagation algorithm works  
► Improving the way neural networks learn  
► A visual proof that neural nets can compute any function  
► Why are deep neural networks hard to train?  
► Deep learning  
    Appendix: Is there a simple algorithm for intelligence?  
    Acknowledgements  
    Frequently Asked Questions

If you benefit from the book, please

# Communication channels and general guidelines

- Answers Q&A platform - [answers.ewi.tudelft.nl](http://answers.ewi.tudelft.nl)
  - All content-related questions and discussions
  - First check if someone else did not ask the same question (incl. previous years)
  - Present your questions clearly (providing background information if needed)
  - Tag your questions (cse2530)
  - Do not share (partial) solutions to graded work
  - Upvote / accept questions and answer that have helped you ☺
- E-mail [ci-cs-ewi@tudelft.nl](mailto:ci-cs-ewi@tudelft.nl):
  - For any questions concerning individual circumstances
- Brightspace Discussions:
  - Only first week (for team formation)



**The prime directive: Please be polite ☺**

# Assessment

- Grading:
  - 50% Exam
  - 50% Lab (Groupwork): 3 exercises
    - Assignment 1 (Swarm intelligence and genetic algorithms): 0.34
    - Assignment 2 (Neural networks): 0.33
    - Assignment 3 (Reinforcement learning): 0.33
- Requirements:
  - Both exam and lab needs to be  $\geq 5.0$
  - The total should be  $\geq 5.8$  to pass
- Repair possibilities:
  - Written exam has one resit
  - Practical assignment: If the final lab grade is between 4.0 and 5.7, the group can choose one of the three assignments for resubmission
    - The group can get any grade for the repair assignment, but the final lab grade will not exceed 6.0.
    - Group members can decide if they want to join the repair attempt or not
    - The deadline for resubmitting the chosen assignment will be about two weeks after the final exam

# Practical assignments



Getting your  
robot butler  
to do  
groceries

- **Assignment 1: Swarm Intelligence (ACO) + Genetic Algorithm** – *Finding the shortest route to all products*
- **Assignment 2: MLP (Neural Networks)** - *Locating the product based on product features*
- **Assignment 3: Reinforcement Learning** – *Finding the route to a goal by trial and error*

# Practical assignments

- 50% of the final grade
- 3 assignments, ~2-3 weeks per assignment
- Groups of 4 students
  - Self-enrol for a group on Brightspace
  - **Deadline: Feb. 18 (Monday), 23:59**
- Lab sessions:
  - Enrol via queue.tudelft.nl
  - TA's will be there to answer your questions
- How to find a group?
  - Join tomorrow's (**February 12**) lab session!
  - If you cannot find a group during the lab session, you can use the Brightspace discussion "Looking for a group" to find a group or other students to join your group
  - PS: When finding/defining a group, consider the benefits of meeting new people and working in a diverse group ☺



# Practical assignments

- Guidelines for teamwork and labs:
  - Listen actively to others in your group
  - Try to express your ideas clearly
  - Make agreement on task division, how often you meet, how you communicate
  - Discuss an “aimed grade” and your main goals during the assignments
  - Respect each other. If conflicts arise, feel free to contact the TAs or lecturers
  - For any undesirable behavior that violates your personal integrity contact the confidential advisors
  - Use the gitlab repository that will be provided to you
  - Have fun! ☺
- Recommended coding (Python) skills for the assignments:
  - Notebooks, git, numpy, data visualization (matplotlib, seaborn), etc.
  - If needed, you can use the first lab to recap these topics. Useful resource: [AI Skills for Engineers: Data Engineering and Data Pipelines](#)





A photograph of a massive flock of birds, likely starlings, captured in flight against a clear sky. The birds are concentrated in the upper half of the frame, appearing as a dense, dark mass. Below them, a field of tall, dry grass stretches towards a line of trees and hills in the distance.

# **Swarm Intelligence**

## **CSE-2530: Computational Intelligence**

**Luciano Cavalcante Siebert**

**(Slides adapted from Joost Broekens; and credits to  
Judith Redi and Pradeep K. Murukannaiah)**

# Last week...

- Introduction to Computational Intelligence

# Next

- Swarm intelligence
  - A form of distributed optimization
  - ... more “collective” than Genetic Algorithms
- Ant Colony Optimization (ACO)
- Evolutionary Algorithms (next week)
  - Inspired by biological evolution
  - Population-based random search from generation to generation

# Learning Objectives (for this lecture)

By the end of these two lectures, you should be able to:

- Understand **swarm intelligence** as a form of distributed optimization
- Explain the **Ant Colony Optimization (ACO)** algorithm
- Apply ACO to the **Traveling Salesman Problem**

# Study Material

- Not in the text book(s)!
- Lecture slides
- Ant Colony Optimization:
  - Reading material:
    - Concept: Bonabeau, E., & Théraulaz, G. (2000). **Swarm smarts**. Scientific American, 282(3), 72-79.
    - Algorithm: Colorni, A., Dorigo, M., & Maniezzo, V. (1991, December). **Distributed optimization by ant colonies**. In Proceedings of the first European conference on artificial life (Vol. 142, pp. 134-142).
  - Optional:
    - Book: Dorigo, M., & Stutzle, T. (2004). **Ant colony optimization**. The MIT Press. 321p. <https://tudelft.on.worldcat.org/oclc/57182707> (eBook available through TU Delft student account)

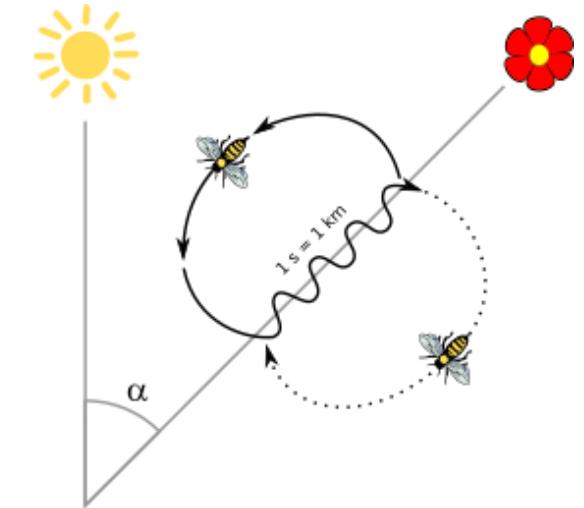
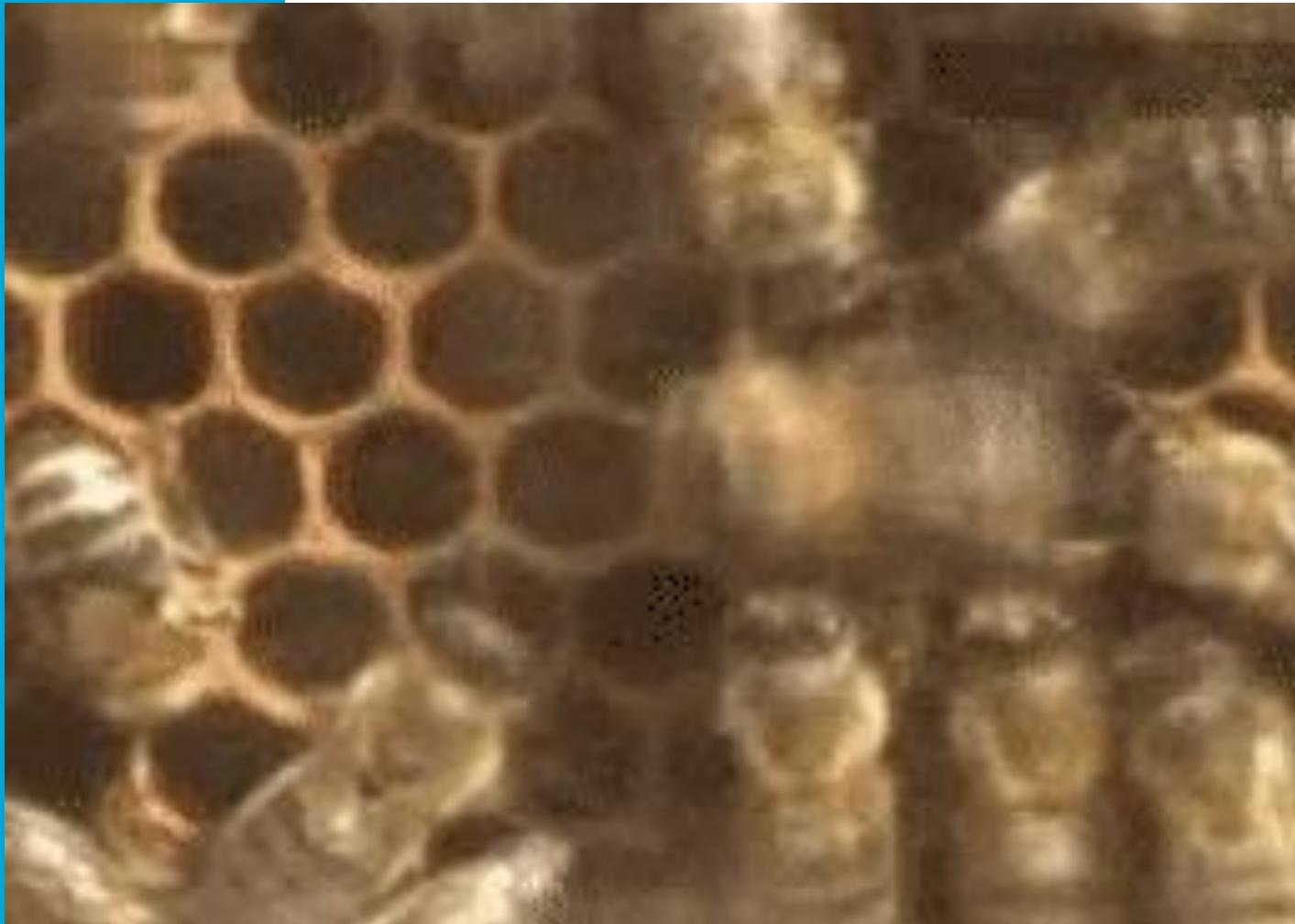
# Swarm Intelligence



# Swarm Intelligence



# Swarm Intelligence



# Swarm Intelligence

Science

Current Issue First release papers Archive About

Submit manuscript

GET OUR E-ALERTS

HOME > SCIENCE > VOL. 335, NO. 6064 > STOP SIGNALS PROVIDE CROSS INHIBITION IN COLLECTIVE DECISION-MAKING BY HONEYBEE SWARMS

REPORT

## Stop Signals Provide Cross Inhibition in Collective Decision-Making by Honeybee Swarms

THOMAS D. SEELEY, P. KIRK VISSCHER, THOMAS SCHLEGEL, PATRICK M. HOGAN, NIGEL R. FRANKS, AND JAMES A. R. MARSHALL [Authors Info & Affiliations](#)

SCIENCE • 8 Dec 2011 • Vol 335, Issue 6064 • pp. 108-111 • DOI: 10.1126/science.1210361

561 194

CHECK ACCESS

### Stop, Don't Go There

Decision-making in complex brains requires the reaching of a consensus based on information relayed by multiple neurons. A similar situation occurs in the decision-making process of swarming bees, where multiple individuals relay information about suitable hive sites, but a single site is chosen by the swarm. Seeley *et al.* (p. 108, published online 8 December; see the Perspective by Niven) show that consensus in this system is reached as information from scouting bees accumulates within the hive. Stop signals were given by scout bees from a particular site to



### CURRENT ISSUE



The connectome of an insect brain

BY MICHAEL WINDING, BENJAMIN D. PEDIGO, ET AL.

*“Decision-making in complex brains requires the reaching of a consensus based on information relayed by multiple neurons. A similar situation occurs in the decision-making process of swarming bees, where multiple individuals relay information about suitable hive sites, but a single site is chosen by the swarm.”*

<https://www.science.org/doi/10.1126/science.1210361>

# Swarm Intelligence



# Swarm Intelligence



# Swarm Intelligence

- The whole > the sum of the parts
- Complex tasks can be tackled only through cooperation of the multitude
  - **The intelligence is in the group**
- The outcome of the effort comes at a group level
  - The single parts may not be aware of the bigger goal
  - There is no leader or centralized coordination
  - Interaction/coordination happens at a microlevel
    - Direct: through senses
    - Indirect: through changes in the environment
- Complex systems:
  - Self-organization
  - Emergent properties
  - No central planning

# Swarm Intelligence

Can we mimic the emergent behavior of swarms of animals to create intelligent algorithms?

- Computational swarm intelligence:
  - Algorithms that define simple individual behavior modulated by simple interactions between individuals
  - Emergent group behavior
- Focus on the individual actors, not on the whole system (which might be too complex to model)
- We will focus on one illustrative method: Ant Colony Optimization (ACO), but there are many others!

# Ant Colony Optimization (ACO)



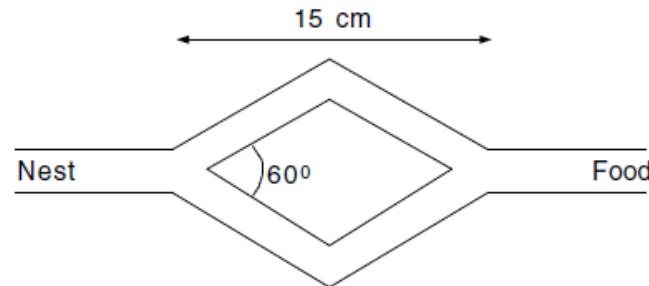
# Ant Colonies

- Foraging behavior (act of gathering wild food resources) is based on indirect communication mediated by pheromones
- A pheromone is a secreted or excreted chemical factor which enables ants to effectively explore the environment

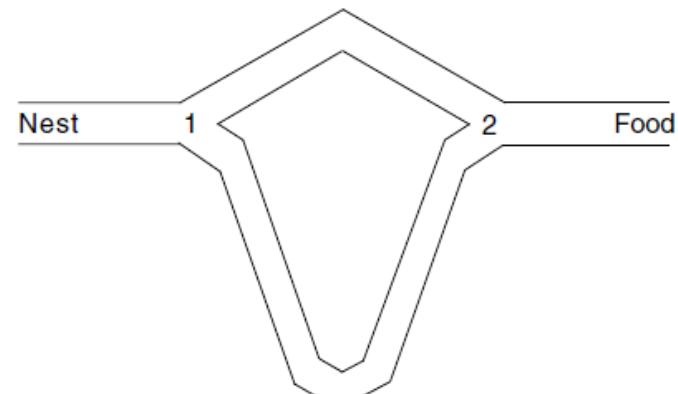


# Ant Colonies

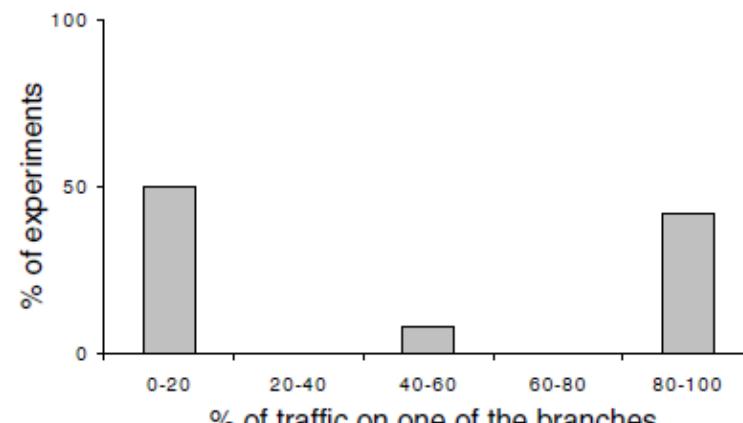
- Double bridge experiment (Goss et al, 1989):



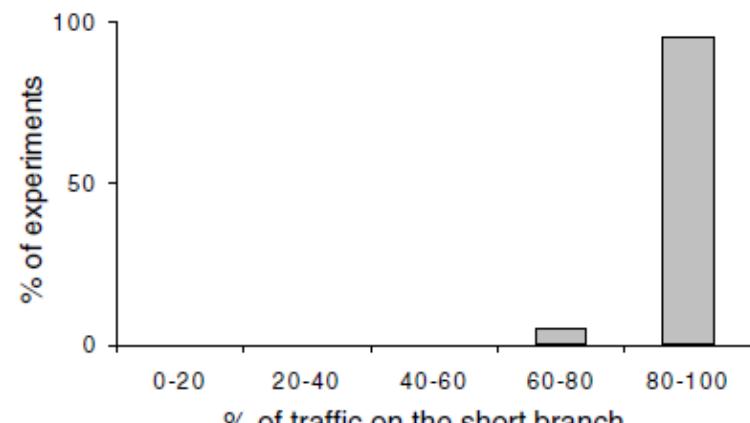
(a)



(b)



(a)



(b)

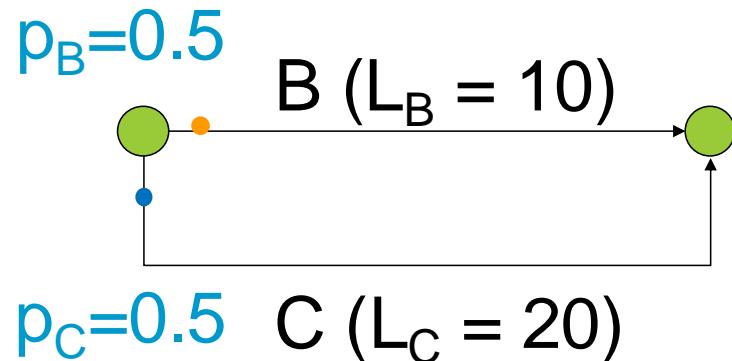
# Ant Colonies



# Ant Colony Optimization

- **Artificial ants** solve optimization problems, inspired by how real ants communicate (indirectly) via pheromone
- For example, the task for “artificial ants” can be to **find the shortest way to some goal**
  - Initially all links have the same (non-zero) probability to be explored
  - When exploring different links, ants release a quantity of pheromone  $\tau$ , proportional to the length of the link (calculated at the end of course)
  - The probability that an ant chooses a specific path depends on the amount of pheromone that was deposited on it
  - The process is **stochastic** → Near-optimal solutions

# Ant Colony Optimization: How?



Two paths to the goal: B and C

Ant 1 chooses B: distance travelled:  $L_B = 10$

Ant 2 chooses C: distance travelled  $L_C = 20$

The shorter the distance, the more pheromone is released on the link

Assuming  $Q = 100$

$$\Delta\tau_B^1 = \frac{100}{10} = 10$$

$$\Delta\tau_C^2 = \frac{100}{20} = 5$$

Pheromone left on the i-th link by ant k

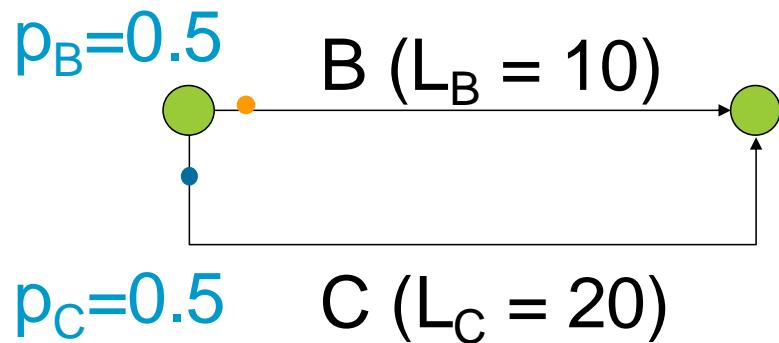
Constant

link length

$$\Delta\tau_i^k = \frac{Q}{L_i}$$

# Ant Colony Optimization

Probabilities to chose the path are updated according to the amount of pheromone

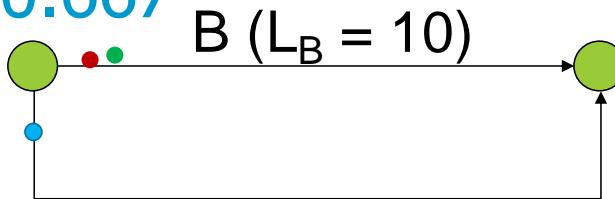


$$p_B = \frac{\tau_B}{\tau_B + \tau_C} = \frac{10}{15} = \frac{2}{3}$$

$$p_C = \frac{\tau_C}{\tau_B + \tau_C} = \frac{5}{15} = \frac{1}{3}$$

# Ant Colony Optimization

$$p_B = 0.667$$



$$p_C = 0.333 \quad C (L_C = 20)$$

Ant 3 chooses B:  $L_B = 10$ ,  $\Delta\tau_B^3 = 100 / 10 = 10$

Ant 4 chooses C:  $L_c = 20$ ,  $\Delta\tau_C^4 = 100 / 20 = 5$

Ant 5 chooses B:  $L_B = 10$ ,  $\Delta\tau_B^5 = 100 / 10 = 10$

Assuming total pheromone  
on link i

$$\tau_i = \sum_k \Delta\tau_i^k$$

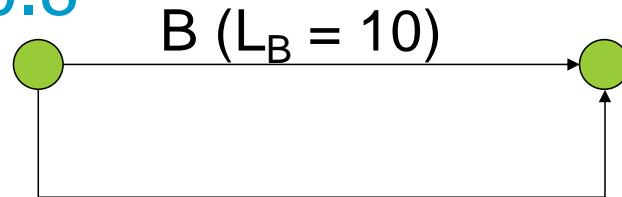
This is a  
simplification!  
We will see more  
elaborate methods  
later in the lecture

$$\tau_B = \tau_B^3 + \tau_B^5 = 10 + 10 = 20$$

$$\tau_C = \tau_C^4 = 5$$

# Ant Colony Optimization

$$p_B = 0.8$$



$$p_C = 0.2$$

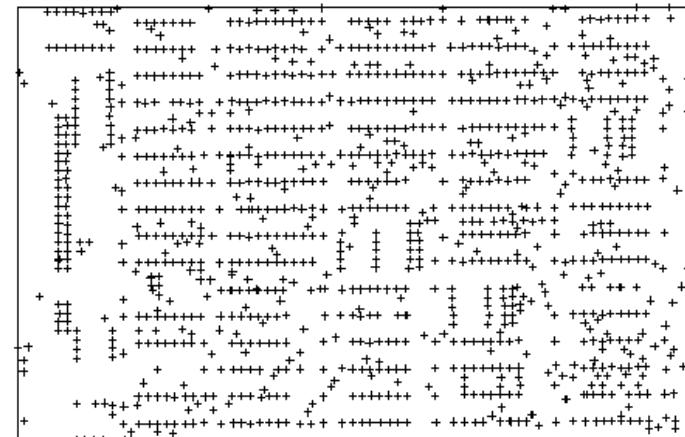
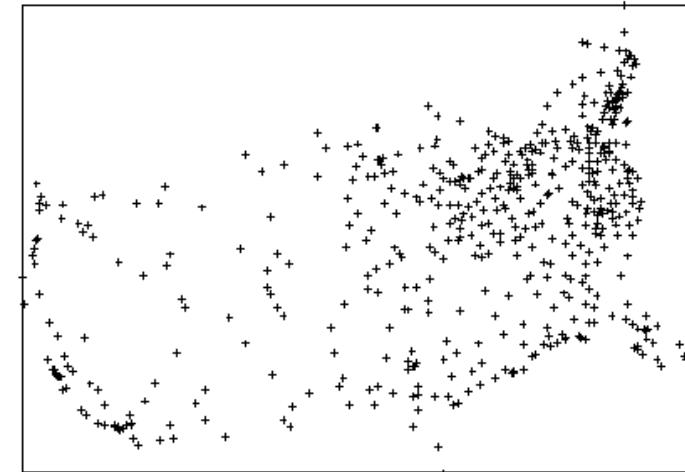
$$p_B = \frac{\tau_B}{\tau_B + \tau_C} = \frac{20}{25} = \frac{4}{5}$$

$$p_C = \frac{\tau_C}{\tau_B + \tau_C} = \frac{5}{25} = \frac{1}{5}$$

# The Travelling Salesman Problem (TSP)

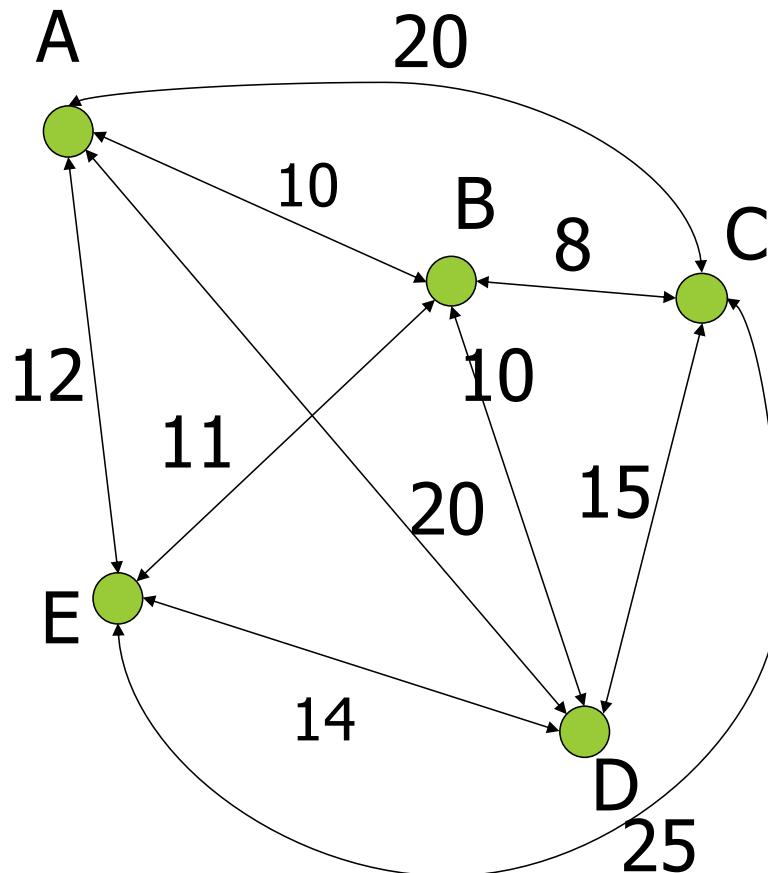
Find the shortest path that allows visiting each city exactly once  
(many applications, e.g. logistics)

- The solution requires a number of computational steps that grows faster than the number of cities
- 15 cities =  $\frac{(n-1)!}{2} = 43\ 589\ 145\ 600$  combinations



# Travelling Sales Ants

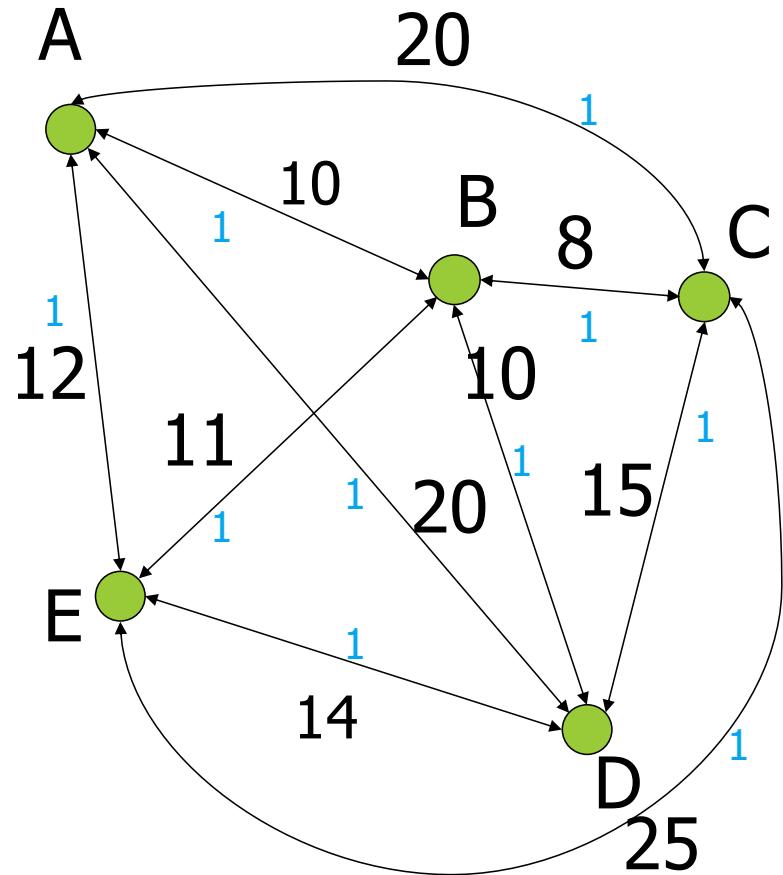
Employ artificial ants to solve TSP



## Assumptions:

- Ants have some memory (they know which cities were visited before)
- Ants are not completely blind (they know to which city a given link goes)
- Time is discrete
- Starting points are randomly assigned

# Ant Colony Optimization for TSP

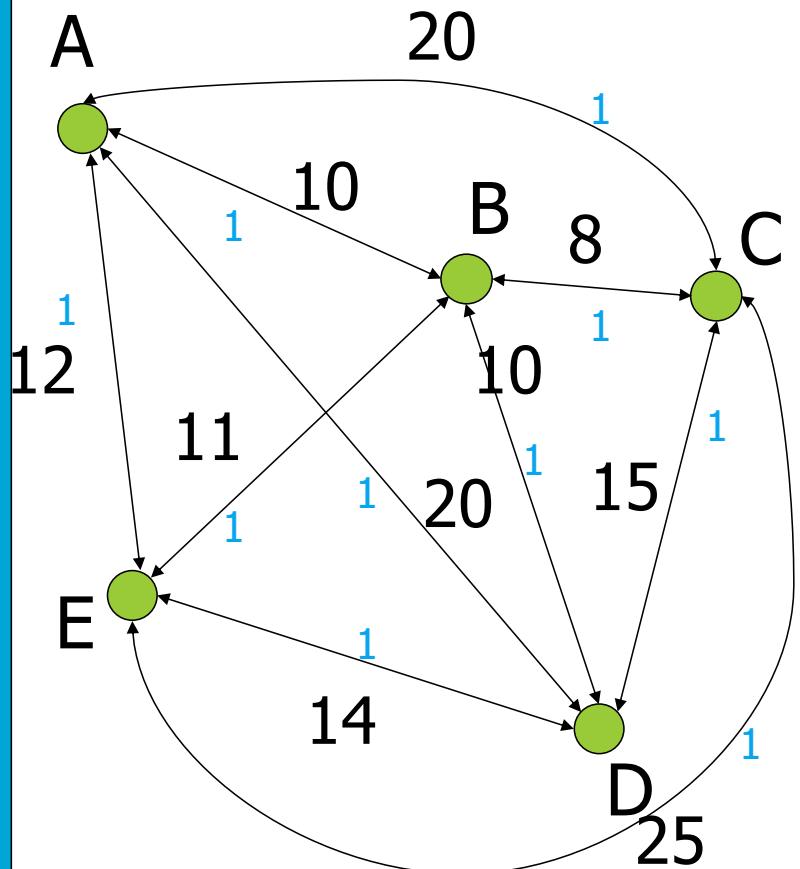


Ant 1: Starting in A  
A → B → C → E → D → A

$$L^1 = \frac{10 + 8 + 25 + 14 + 20}{77}$$

\*little blue numbers on links represent the quantity of pheromone

# Ant Colony Optimization for TSP



Ant 1: Starting in A  
A→B→C→E→D→A

Ant 2: Starting in C  
C→B→E→A→D→C

$$L^1 = \frac{10 + 8 + 25 + 14 + 20}{77}$$
$$L^2 = \frac{8 + 11 + 12 + 20 + 15}{66}$$

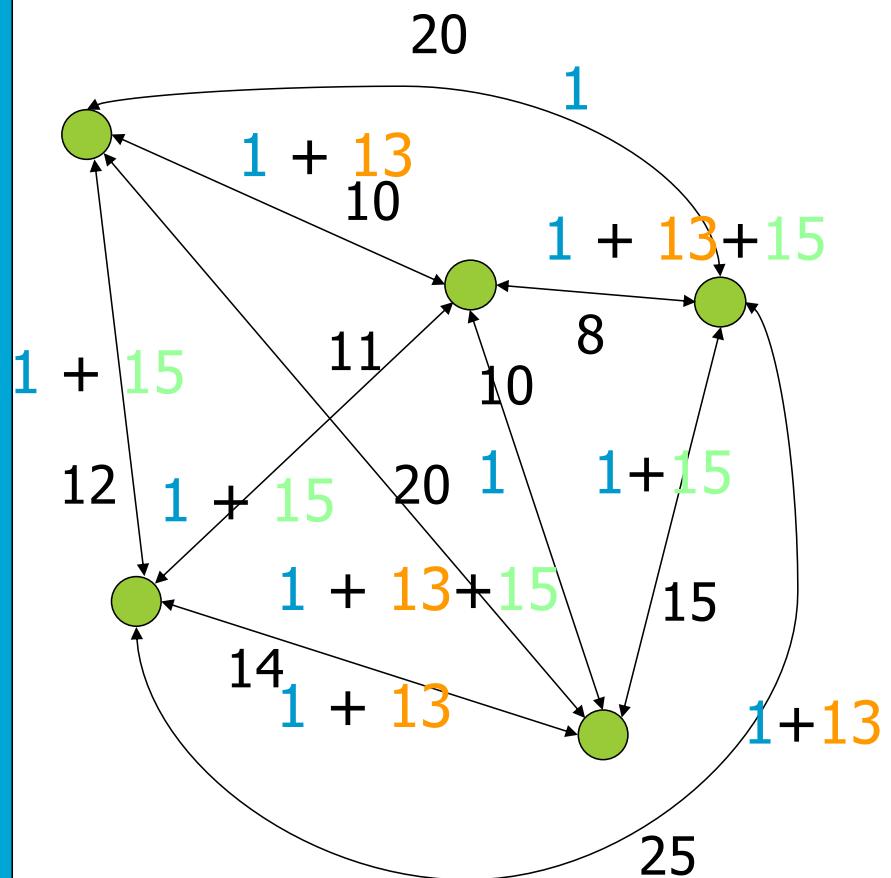
Shortest route gets more pheromone:

Assuming: Q = 1000

$$\Delta\tau^k = Q \times \frac{1}{L^k}$$

$$\Delta\tau^1 = \frac{1000}{77} = 12.99 \quad \Delta\tau^2 = \frac{1000}{66} = 15.15$$

# Ant Colony Optimization for TSP



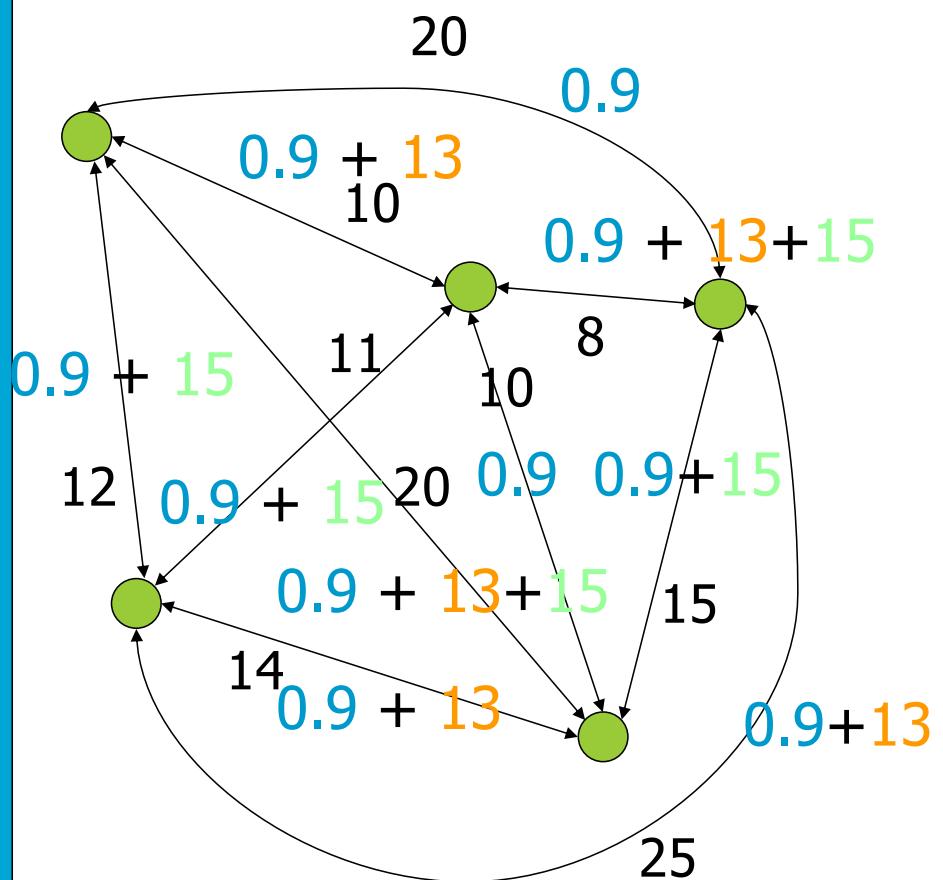
Ant 1:  $12.99 \cong 13$

Ant 2:  $15.15 \cong 15$

$$\tau_{ij} = \tau_{ij} + \sum_{k=1}^m \Delta\tau_{ij}^k$$

- What if many routes use a link which is not part of a shorter tour?
- What if a short route contains a very long link that initially is less likely to be used?

# Ant Colony Optimization for TSP



Ant 1:  $12.99 \cong 13$

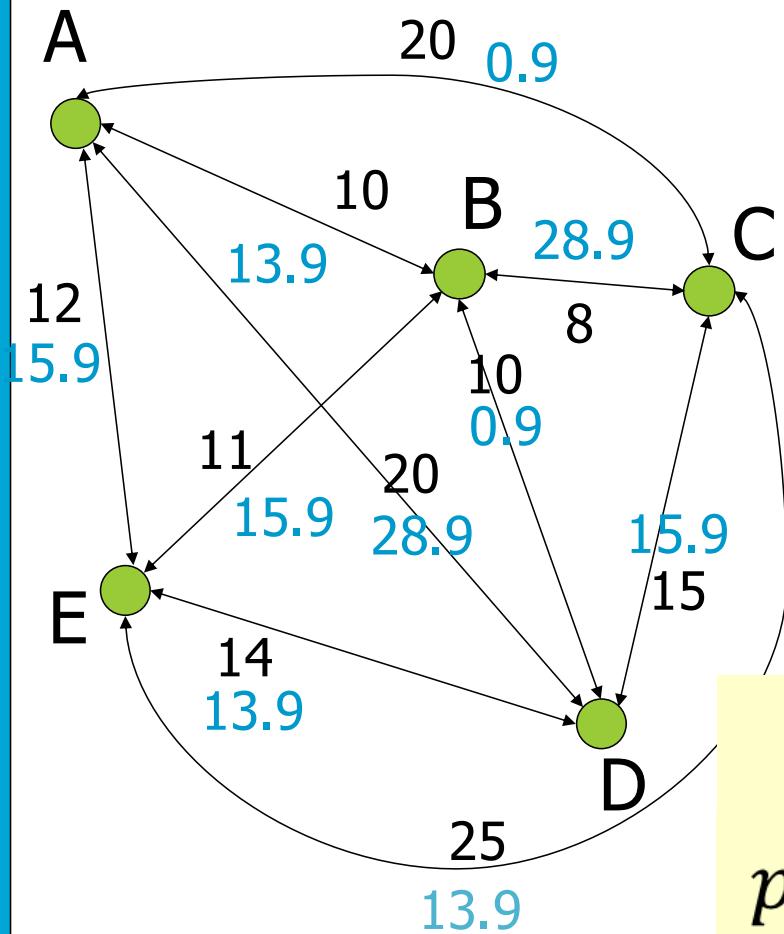
Ant 2:  $15.15 \cong 15$

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \sum_{k=1}^m \Delta\tau_{ij}^k$$

$\rho$ : evaporation constant (here 0.1)

The amount of pheromone currently on path  $ij$  corresponds to the amount of pheromone existing on it, decreased by a quantity proportional to the evaporation constant, plus the sum of all the pheromone released on that link by all the ants that passed by it this round.

# Ant Colony Optimization for TSP



The probability of choosing for a (previously unseen) path is proportional to the amount of pheromone and inversely proportional to the length of the path

Pheromone on path  $ij$

Heuristic (can be another), e.g. Euclidean distance

$$\eta_{ij} = \frac{1}{L_{ij}}$$

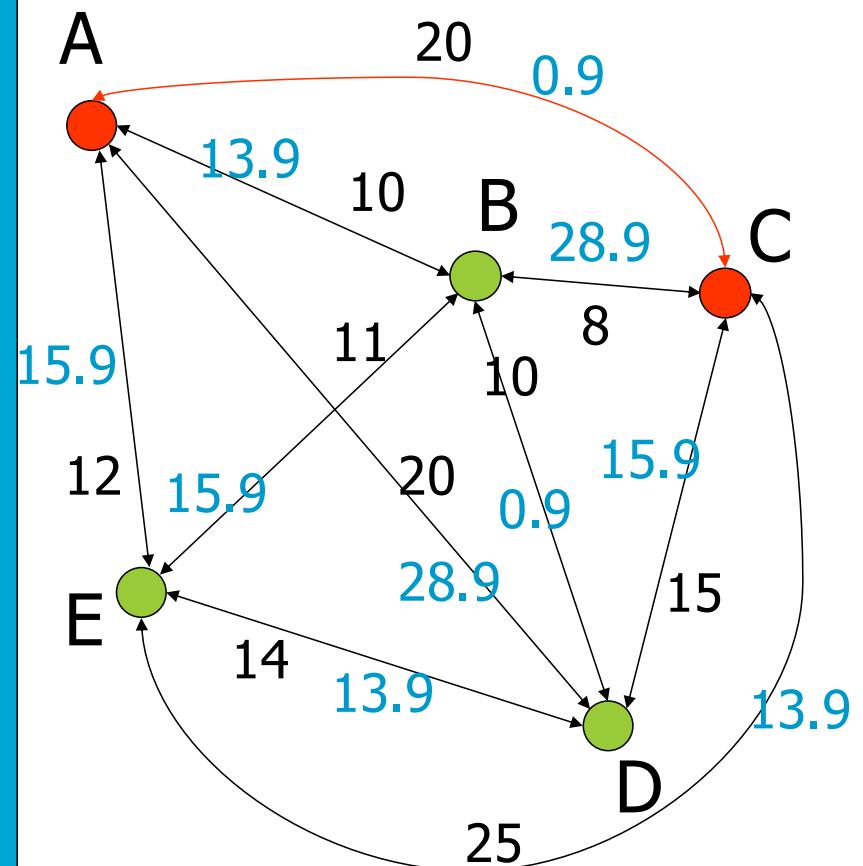
Set of all unvisited cities that can be reached from  $i$

$$p_{ij}^k(t) = \begin{cases} \frac{\tau_{ij}(t)^\alpha \eta_{ij}^\beta}{\sum_{m \in C_i^k} \tau_{im}(t)^\alpha \eta_{im}^\beta}, & j \in C_i^k \\ 0, & \text{otherwise} \end{cases}$$

Pheromone on all paths departing from city  $i$

$\alpha$  and  $\beta$  are pre-specified parameters (trail x visibility)

# Ant Colony Optimization for TSP



Ant 1 in C (coming from A):

$$p_{CB} = \frac{28.9 * (1/8)^{0.5} (= 10.2177)}{10.2177 + 4.105 + 2.78}$$

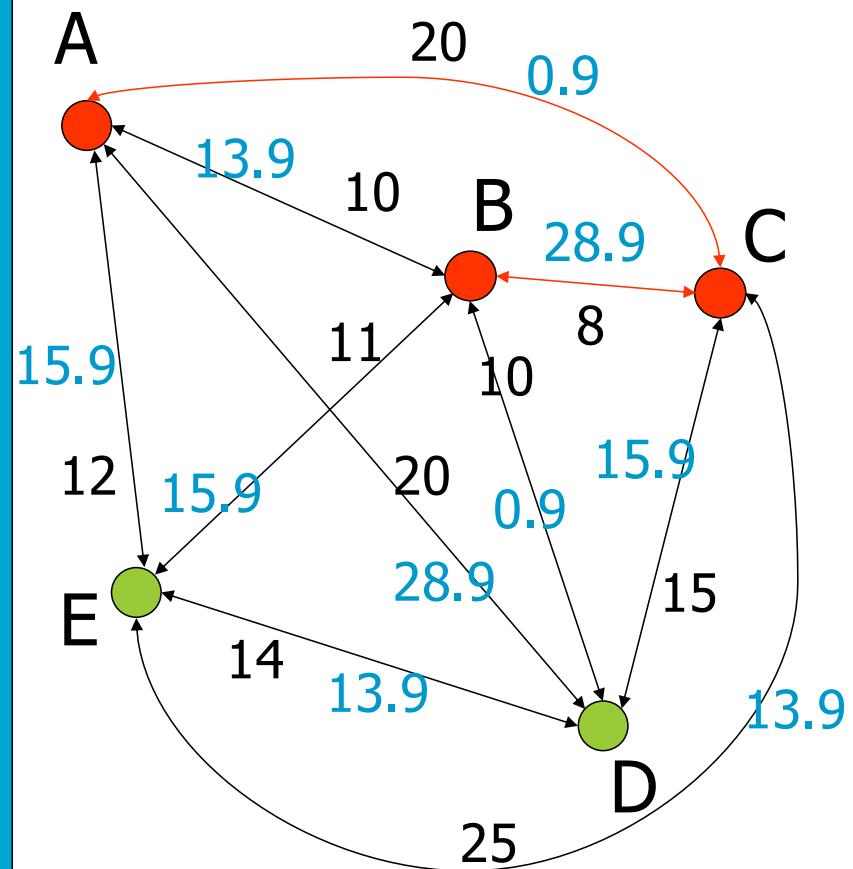
$$p_{CD} = \frac{15.9 * (1/15)^{0.5} (= 4.105)}{10.2177 + 4.105 + 2.78}$$

$$p_{CE} = \frac{13.9 * (1/25)^{0.5} (= 2.78)}{10.2177 + 4.105 + 2.78}$$

$$p_{ij}^k(t) = \begin{cases} \frac{\tau_{ij}(t)^\alpha \eta_{ij}^\beta}{\sum_{m \in C_i^k} \tau_{im}(t)^\alpha \eta_{im}^\beta}, & j \in C_i^k \\ 0, & \text{otherwise} \end{cases}$$

Assuming:  
 $\alpha = 1, \beta = 0.5$

# Ant Colony Optimization for TSP



$$p_{CB} = 0.597$$

Let us assume this path was chosen

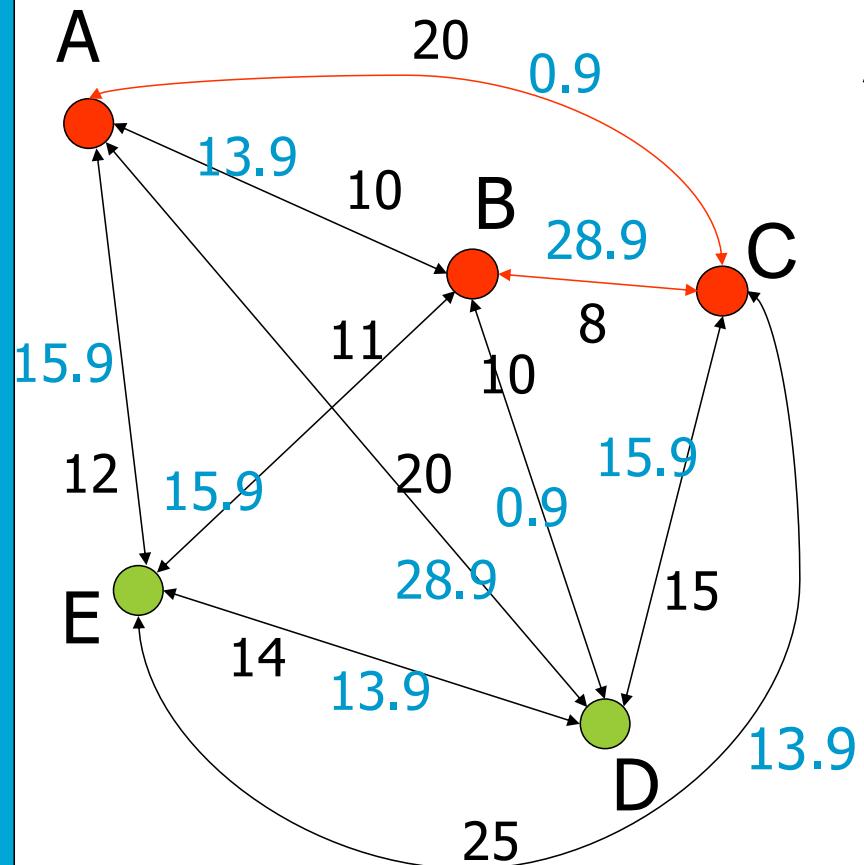
$$p_{CD} = 0.240$$

$$p_{CE} = 0.163$$

$$p_{ij}^k(t) = \begin{cases} \frac{\tau_{ij}(t)^\alpha \eta_{ij}^\beta}{\sum_{m \in C_i^k} \tau_{im}(t)^\alpha \eta_{im}^\beta}, & j \in C_i^k \\ 0, & \text{otherwise} \end{cases}$$

$$\alpha = 1, \beta = 0.5$$

# Ant Colony Optimization for TSP



Ant 1 in B (coming from C):

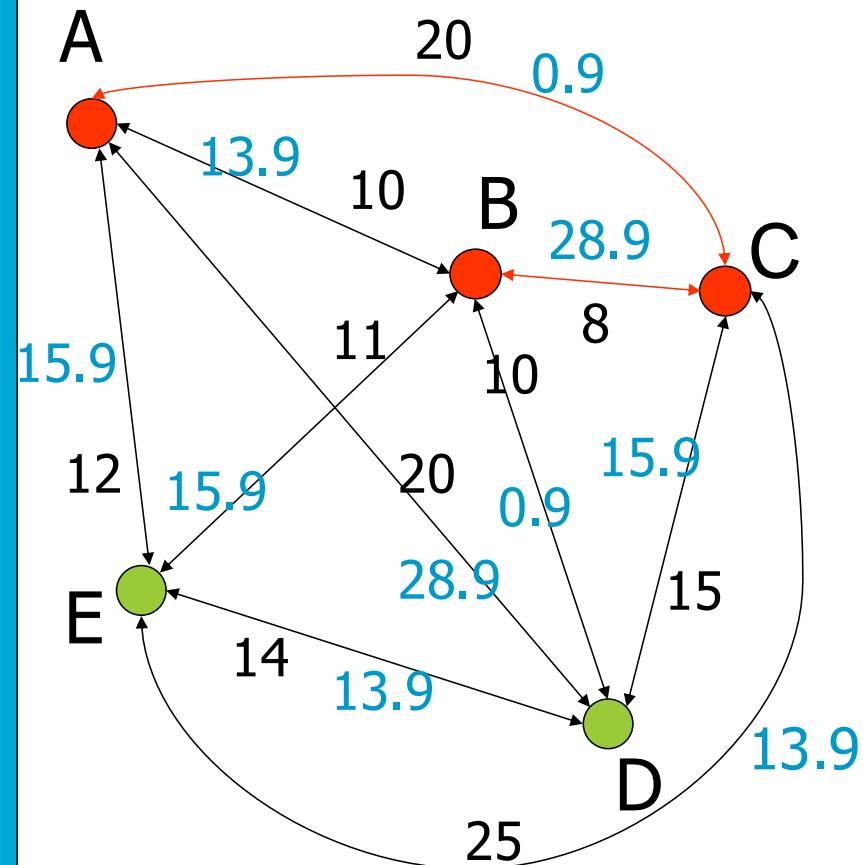
$$p_{BD} = \frac{0.9 * (1/10)^{0.5}}{0.9 * (1/10)^{0.5} + 15.9 * (1/14)^{0.5}} = 0.056$$

$$p_{BE} = \frac{15.9 * (1/11)^{0.5}}{0.9 * (1/10)^{0.5} + 15.9 * (1/14)^{0.5}} = 0.944$$

$$p_{ij}^k(t) = \begin{cases} \frac{\tau_{ij}(t)^\alpha \eta_{ij}^\beta}{\sum_{m \in C_i^k} \tau_{im}(t)^\alpha \eta_{im}^\beta}, & j \in C_i^k \\ 0, & \text{otherwise} \end{cases}$$

$$\alpha = 1, \beta = 0.5$$

# Ant Colony Optimization for TSP



Ant 1 in B (coming from C):

$$p_{BD} = 0.056$$

Let us assume that, even being more unlikely, Ant 1 still chooses to go to node D (stochasticity)

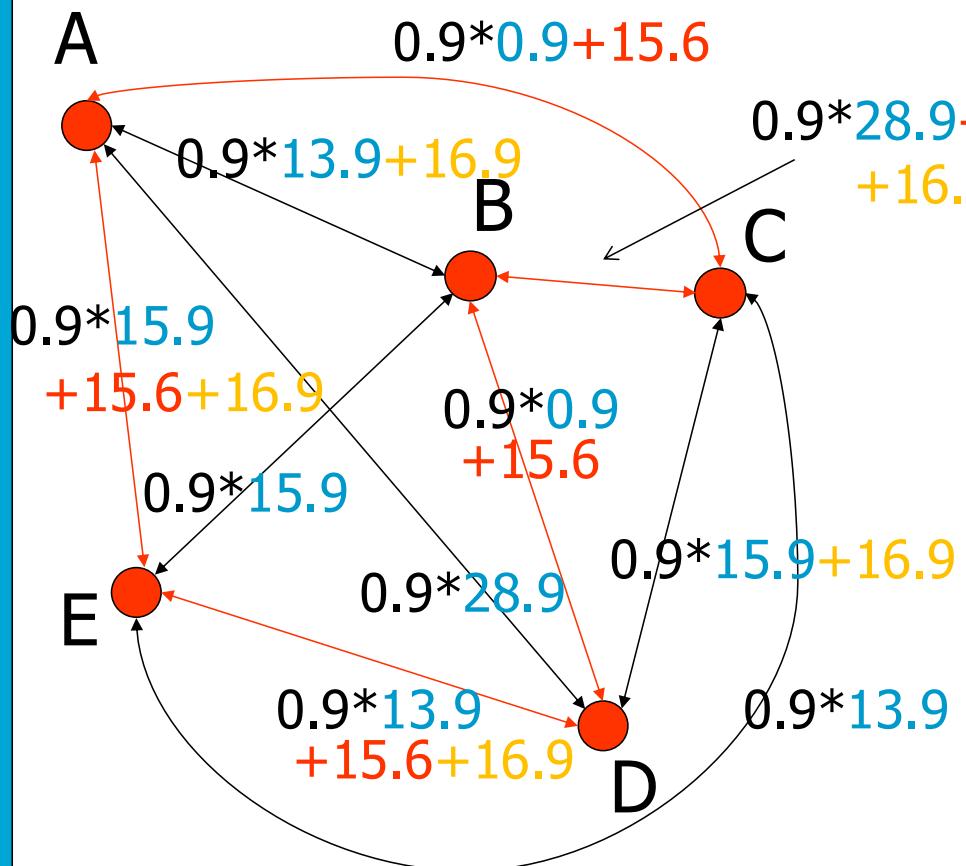
$$p_{BE} = 0.944$$

Next step would then be E and then back to A

$$p_{ij}^k(t) = \begin{cases} \frac{\tau_{ij}(t)^\alpha \eta_{ij}^\beta}{\sum_{m \in C_i^k} \tau_{im}(t)^\alpha \eta_{im}^\beta}, & j \in C_i^k \\ 0, & \text{otherwise} \end{cases}$$

$$\alpha = 1, \beta = 0.5$$

# Ant Colony Optimization for TSP



20	8
8	10
10	12
14	14
12	15
<hr/>	
64	59

$$\Delta\tau^k = Q \times \frac{1}{L^k}$$

$$\Delta\tau^1 = \frac{1000}{64} = 15.6$$

$$\Delta\tau^2 = \frac{1000}{59} = 16.9$$

$$\alpha = 1, \beta = 0.5$$

# Travelling Sales Ants

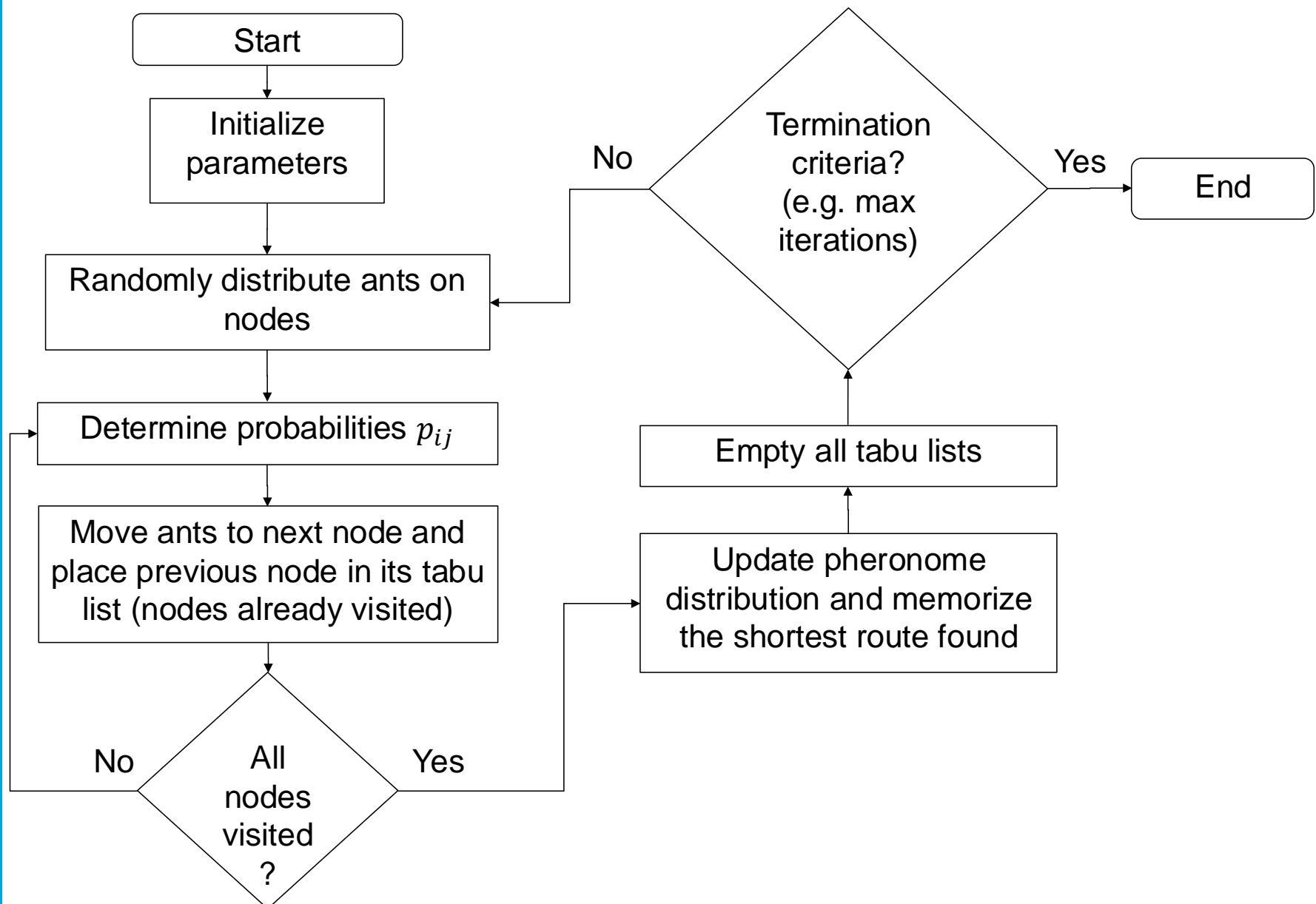
- Ant after ant, shorter and shorter routes are found
- Even if a long link (which is not good for a short route) is very popular, at the end it will be neglected, because of evaporation
- Same happens for long links that are part of a short route
  - Even if in the beginning they are less likely to be selected
  - Since selection is stochastic, at some point some an ant will pass by and make the link more appealing with pheromone
- Once again, the method finds a short route, but not necessarily the shortest!

# ACO for TSP: Parameters

Parameters must be **tuned to the specific problem**

- Number of ants?
  - Typically, same as the number of cities
- Initial distribution of ants over cities
  - Uniform distribution?
- Q value
  - Large enough to not lose precision during updates
- $\alpha$  and  $\beta$  values?
- Evaporation rate?

# ACO Algorithm (Ant-Cycle)



# Variations on the ACO algorithm

- The Ant-Cycle algorithm achieved encouraging initial results in the TSP, but was found to be inferior to other state-of-the-art algorithms for the TSP
- Other extensions achieved better results, and also led to other applications

ACO algorithm	TSP	Main references
Ant System (AS)	yes	Dorigo (1992); Dorigo, Maniezzo, & Colorni (1991a,b, 1996)
Elitist AS	yes	Dorigo (1992); Dorigo, Maniezzo, & Colorni (1991a,b, 1996)
Ant-Q	yes	Gambardella & Dorigo (1995); Dorigo & Gambardella (1996)
Ant Colony System	yes	Dorigo & Gambardella (1997a,b)
$\mathcal{MAX}\text{-}\mathcal{MIN}$ AS	yes	Stützle & Hoos (1996, 2000); Stützle (1999)
Rank-based AS	yes	Bullnheimer, Hartl, & Strauss (1997, 1999c)
ANTS	no	Maniezzo (1999)
Hyper-cube AS	no	Blum, Roli, & Dorigo (2001); Blum & Dorigo (2004)

In the column TSP we indicate whether this ACO algorithm has already been applied to the traveling salesman problem.

- In these extensions, the main differences are the way the pheromone update is performed, and some details in the management of the pheromone trails

# Variations on the ACO algorithm

- Elitist Ant Systems: Dorigo (1992) and Dorigo et al., (1991a, 1996)
  - Provide additional reinforcement to the best routers found since the start of the algorithm

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^m \Delta\tau_{ij}^k + e\Delta\tau_{ij}^{bs},$$

$$\Delta\tau_{ij}^{bs} = \begin{cases} 1/C^{bs}, & \text{if arc } (i, j) \text{ belongs to } T^{bs}; \\ 0, & \text{otherwise.} \end{cases}$$

length

# Variations on the ACO algorithm

- Extending the ACO with human intelligence
  - “Interactive machine learning: experimental evidence for the human in the algorithmic loop - A case study on Ant Colony Optimization”
  - Each time the human travels over an ant, the current pheromone value of the edge is multiplied by 5.
    - Recommendation: Link with highest probability



Holzinger, A., Plass, M., Kickmeier-Rust, M., Holzinger, K., Crișan, G. C., Pintea, C. M., & Palade, V. (2019). Interactive machine learning: experimental evidence for the human in the algorithmic loop. *Applied Intelligence*, 49(7), 2401-2414.  
Game (Firefox): <https://iml.hci-kdd.org/TravellingSnakesman/>

# Other applications

- Scheduling
  - Assembly lines
  - Airline crew scheduling
- Routing
  - Vehicle routing
  - Circuit boards
- Image processing
  - Edge detection and edge linking
- Network design
  - Energy, gas, internet
- Biology
  - Protein folding

# Other applications

- Airline crew scheduling

Taipei–Tokyo sample timetable – eight flights.

Flight num	Dep. city	Days of service	Dep. time	Des. city	Arr. time	Flying time
1	Taipei	Mon	10:55	Tokyo	13:50	2:55
2	Taipei	Mon	12:55	Tokyo	15:50	2:55
3	Taipei	Tue	14:55	Tokyo	17:50	2:55
4	Taipei	Wed	12:55	Tokyo	15:50	2:55
5	Tokyo	Mon	16:55	Taipei	19:50	2:55
6	Tokyo	Tue	14:55	Taipei	17:50	2:55
7	Tokyo	Wed	14:55	Taipei	17:50	2:55
8	Tokyo	Wed	18:15	Taipei	21:40	2:55

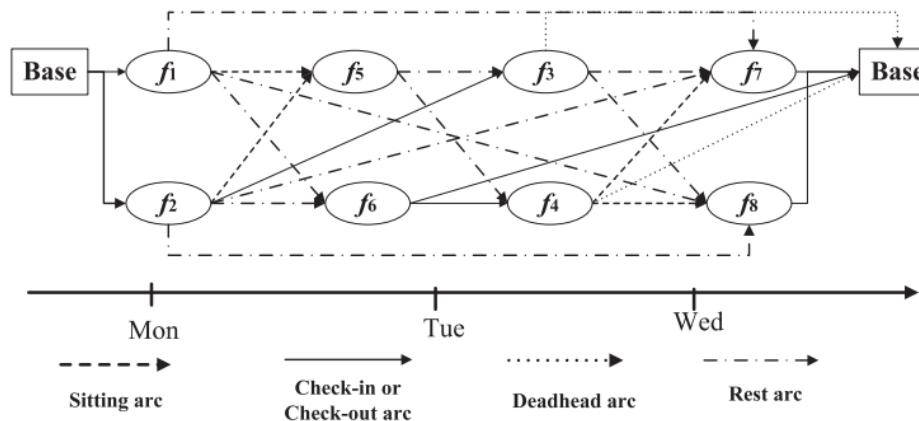


Fig. 2. Sample timetable as the search for the shortest path.

# A bit more on two examples on other applications

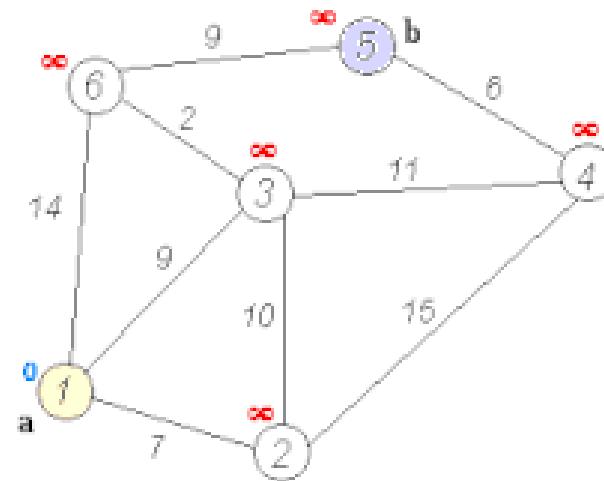
- Image processing for magnetic resonance
  - Brain segmentation to calculate brain's volume changes



Fig. 2 Results of brain segmentation. **a** Original image. **b** Divided to WM. **c** GM. **d** CSF using the proposed ACO algorithm

# Comparison between ACO and other algorithms

- Dijkstra usually requires less computational power
- ACO is usually better suited than Dijkstra for:
  - Scenarios where you have a dynamic cost function (such as traffic in a city)
  - Very Large scenarios



Baeza, D., Ihle, C. F., & Ortiz, J. M. (2017). A comparison between ACO and Dijkstra algorithms for optimal ore concentrate pipeline routing. *Journal of Cleaner Production*, 144, 149-160.

Shojaie, A. A., & Seyed Bariran, S. E. (2021). A Comparison of Extended Dijkstra and ACO Algorithm for Shortest Path Problem in Dynamic Networks with Delay Times. *Advances in Industrial Engineering*, 55(1), 1-26.

# Ant Colony Optimization: Summary

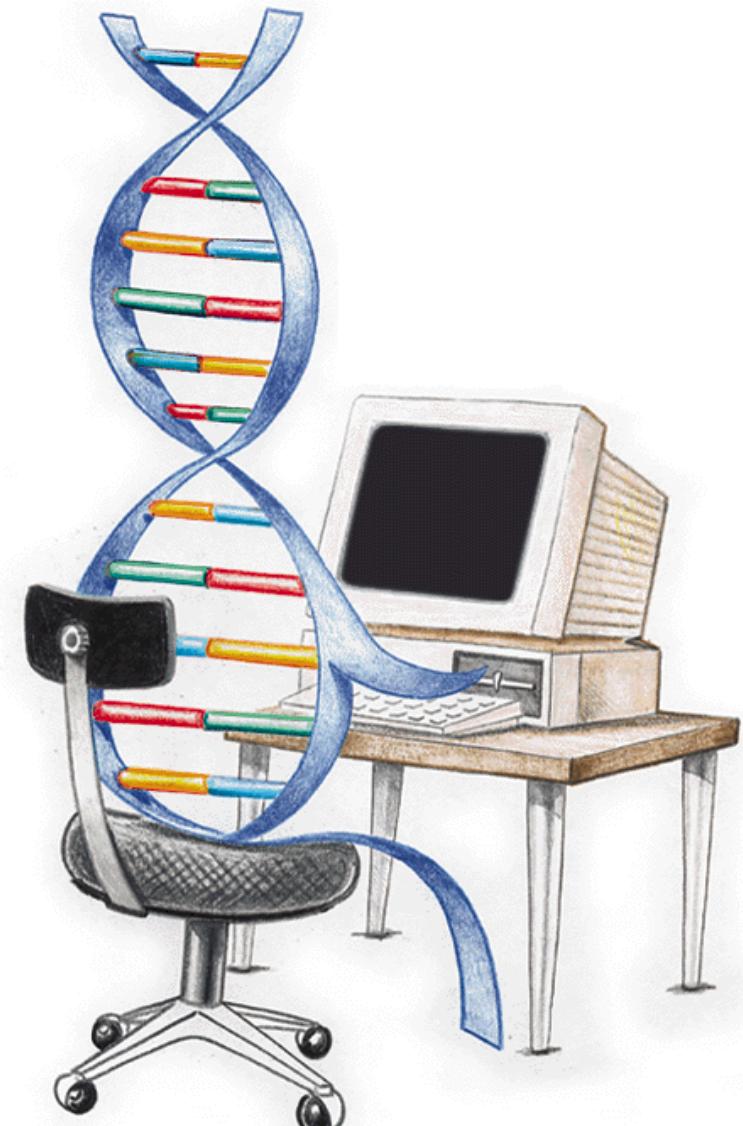
- A big advantage of ACO is its inherent flexibility
- Environment and conditions might be unknown and/or dynamic (changing over time)
  - Internet: node breaks down
  - Highway: closed for accident
- Ants keep exploring: they adapt fast
- Pheromone gives backup options
  - If a link is closed, the second shortest route is easy to reconstruct

# Genetic Algorithms

**CSE-2530:**  
**Computational Intelligence**

Pradeep K. Murukannaiah

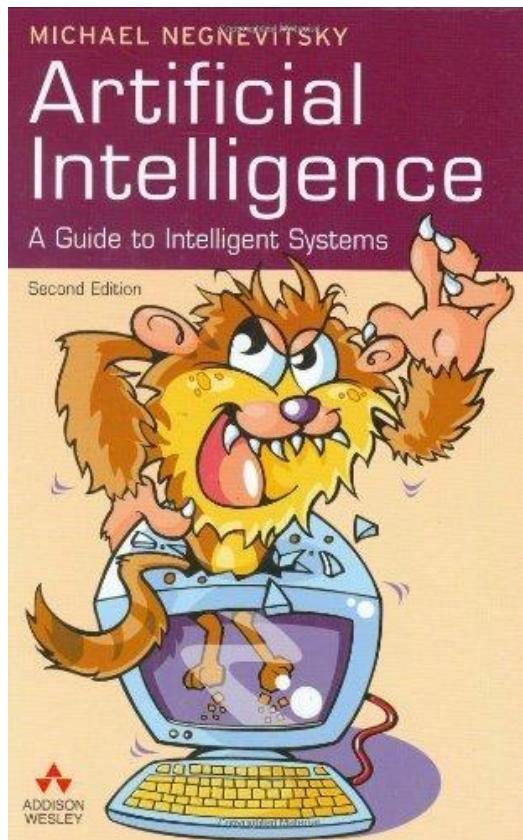
(Slides adapted from Joost Broekens)



# Learning Objectives

- Explain the intuition behing **evolutionary computing**
- Recognize **problems** that can be modeled via evolutionary computing methods
- Enumerate the steps of a **genetic algorithm** (GA)
- Describe the key GA operations: **selection, cross-over, and mutation**
- Apply GA to **optimization problems**

# Study Material



Chapter 7: Sections 7.1–7.4

Lecture slides

# Evolutionary Computing (EC)

EC is most commonly used in **optimization problems**, with a given *objective function* and a set of *constraints*

For example:

Maximize:  $f(x_1, x_2) = 2x_1 + x_2$  Objective Function

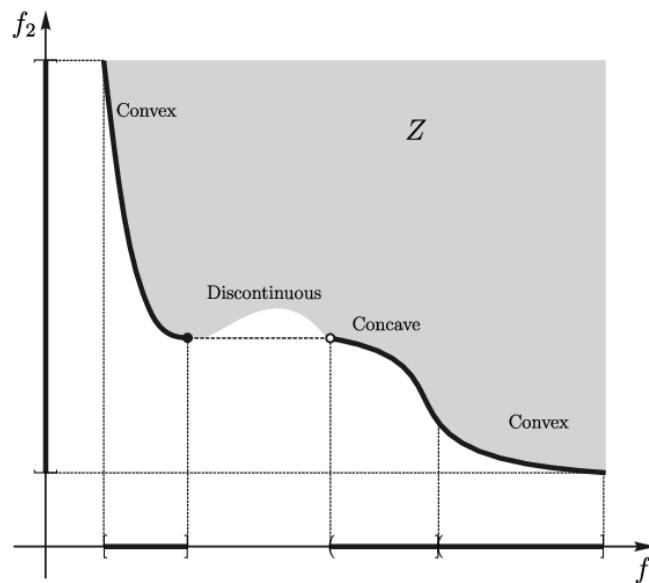
Subject to:  $x_1 + x_2 \leq 3$  Constraints

$$5x_1 + 2x_2 \leq 10$$

and:  $x_1, x_2 \geq 0$  Variable Bounds

# Why EC for Optimization?

EC algorithms are **flexible** optimization algorithms



- No constraints in the **type of variables**
- Works well with **non-linear** objective functions
- Insensitive to **discontinuities or shape**
- Can be adapted to **multi-objective optimization** problems
- In general, requires **few assumptions** about the problem ...

# EC: Application Areas

- Economics
  - For example, to characterize economic models, asset pricing, etc.
- Logistics
  - For example, in scheduling and traveling salesman problem (TSP)
- Engineering
  - For example, in identifying the optimal design or an spacecraft
- Machine learning
  - For example, to search optimal values of (hyper)parameters

# Evolutionary Process

# Rabbits and Foxes



Smarter Rabbits

Faster foxes

Faster rabbits

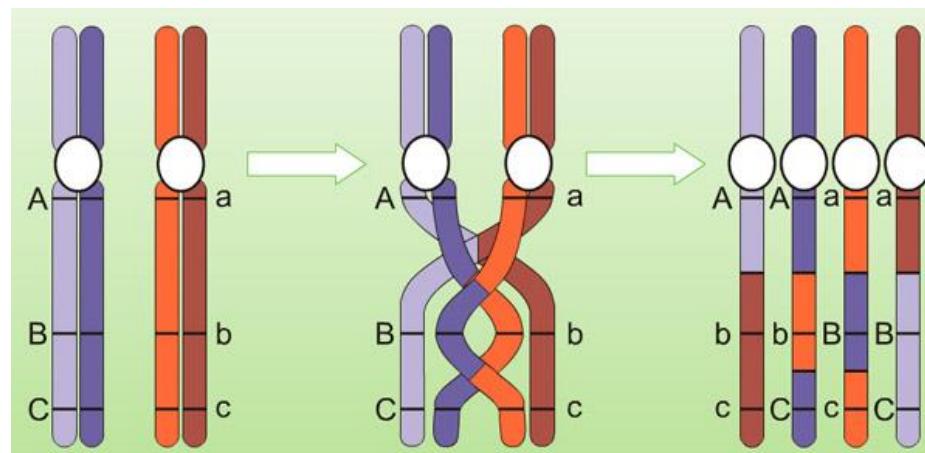
# Evolution: General Principles

- **Heredity:** There must be a process by which children receive the properties of their parents
- **Variation:** There must be a variety of traits present in the population or a means to introduce variation
- **Selection:** There must be a mechanism by which some individuals in the population have an opportunity to be parents and others do not. This is typically referred to “*the survival of the fittest*”

# Phenotypes and Genotypes

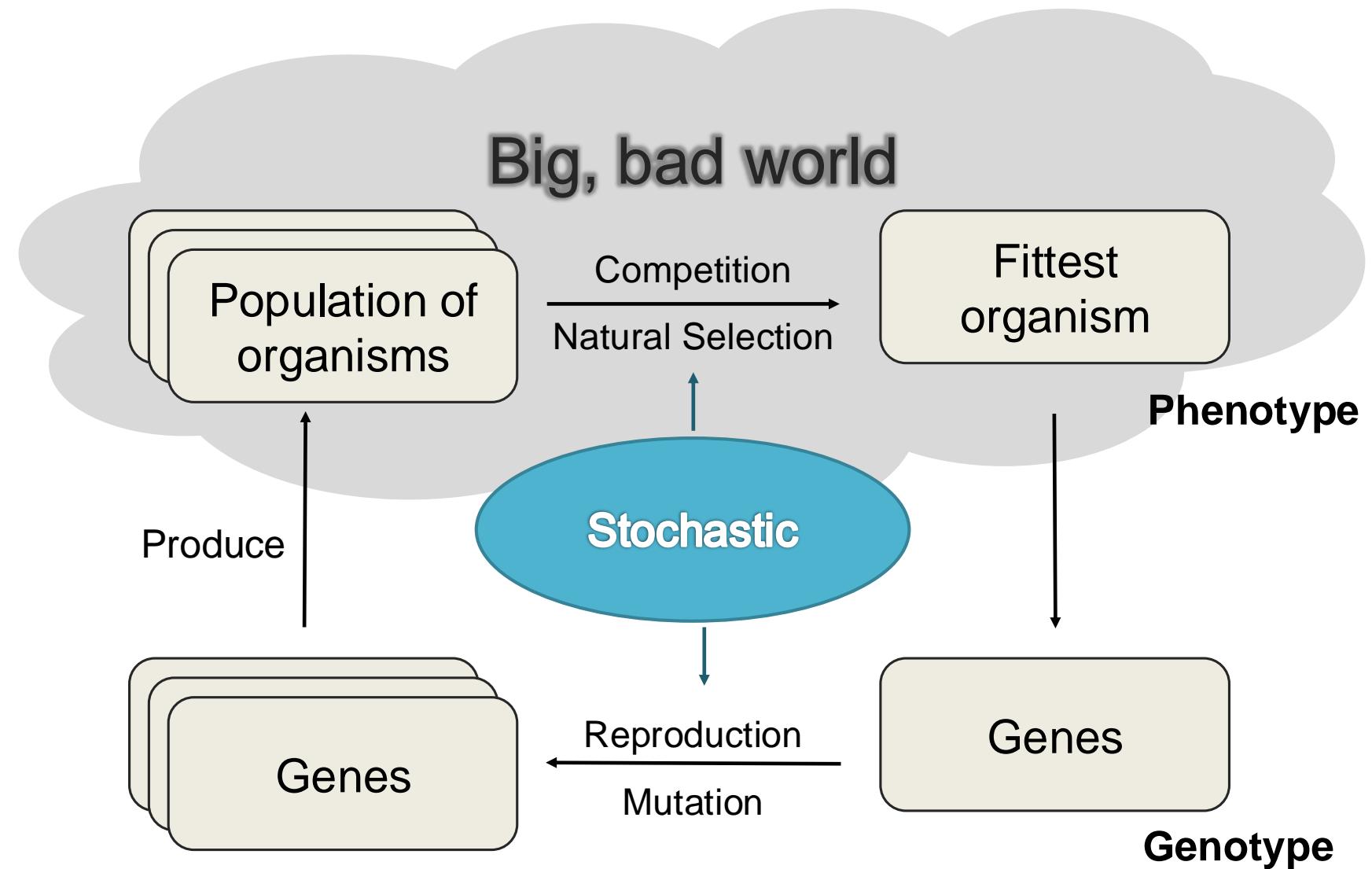


Phenotype



Genotype

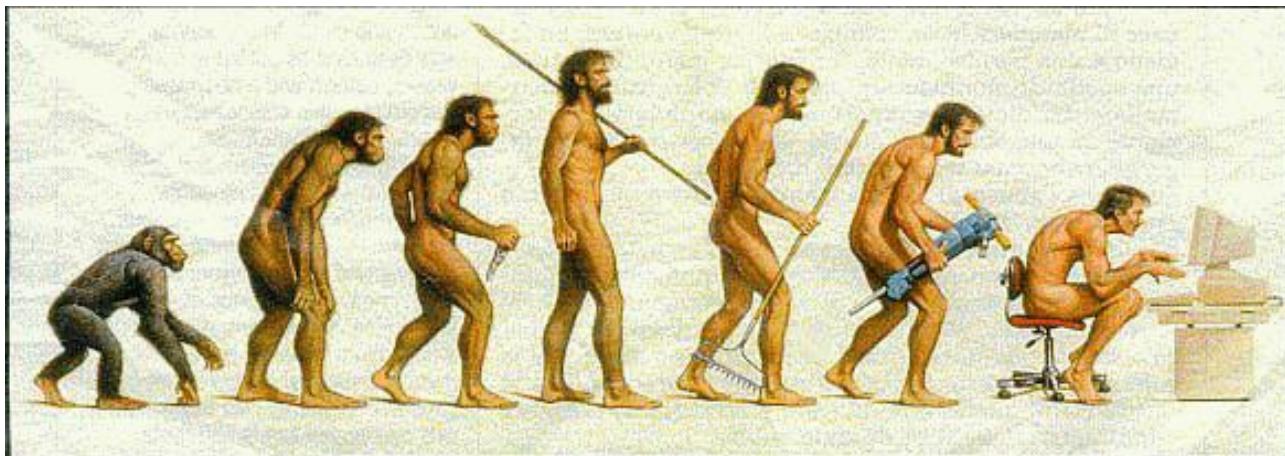
# Evolutionary Cycle



# A Broad View on Intelligence

- Intelligence is the capability of a system to adapt its behaviour to an ever changing environment.

## The Theory of Evolution



- Organisms change over time as a result of changes in heritable physical or behavioural traits. Changes that allow an organism to better adapt to its environment will help it survive and have more off-springs.



# Simulating Evolution on a Computer



# Evolution as an Algorithm

# Evolution as an Algorithm

Can we **mimic evolution** to find a solution (model, program, design, parameter setting, etc.) that is

**‘the fittest to the domain’?**

Fitness = Goodness of a  
candidate solution

## **Assumptions:**

- Every candidate in the solution space has its “*fitness*”
- Similar solutions have similar fitness

- **Evolutionary Computing (EC)** is the umbrella term used for algorithms that are inspired by evolution. Multiple flavours:
  - Genetic algorithms, genetic programming, evolutionary strategies

# EC: Basic Concepts

Borrows concepts from evolution to solve optimization problems

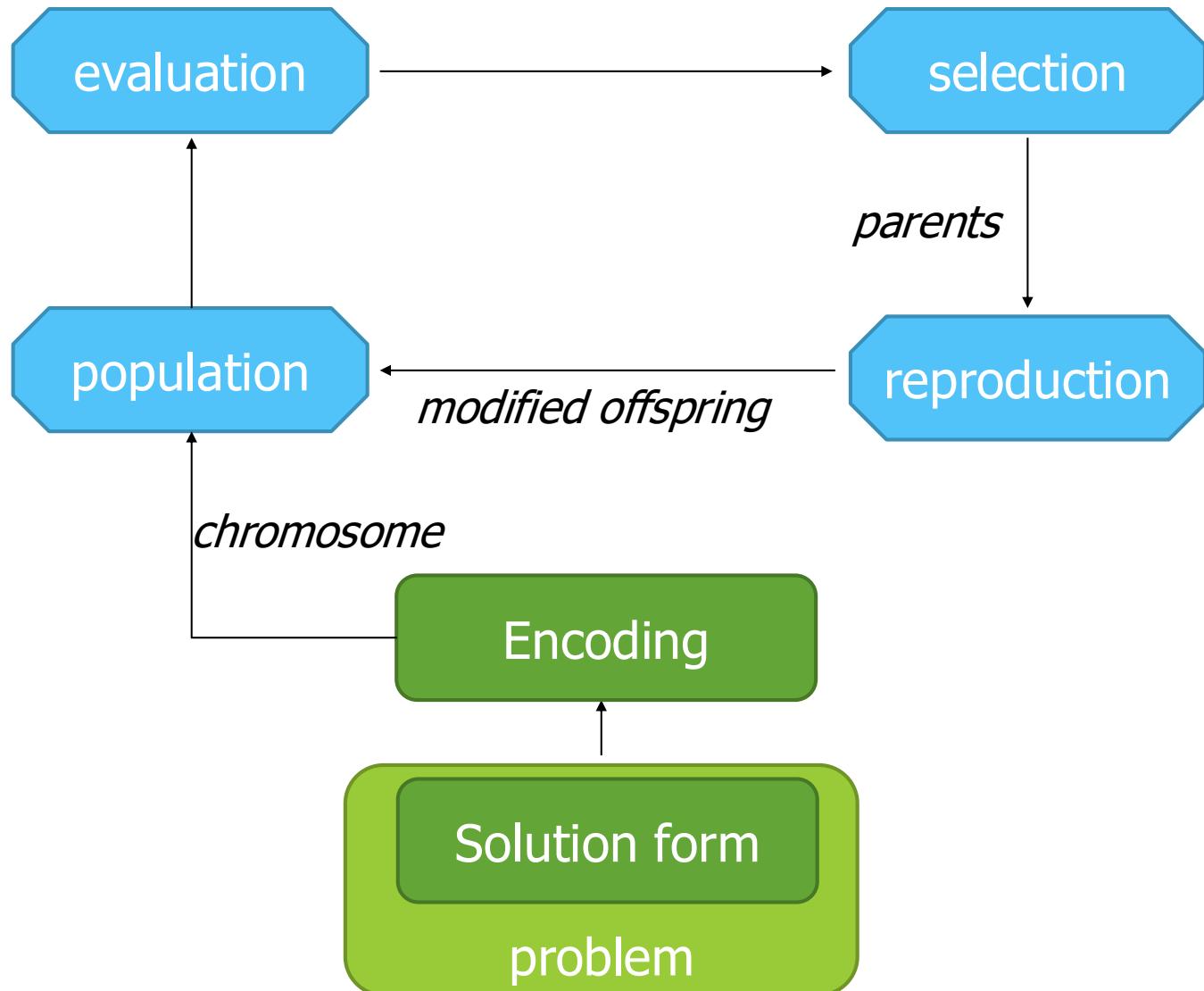
- **Population:** A set of candidate solutions
- **Fitness:** Criterion for evaluating the goodness of candidates
- **Selection:** Process through which solutions are picked for further refinement
- **Reproduction:** Process through which new solutions are produced from existing solutions under the assumptions that
  - Good solutions, when combined, will likely yield even better solutions (exploitation)
  - Unexpected changes in solutions (e.g., via mutation) can yield better solutions (exploration)

# Genetic Algorithm



Proposed in  
the 1970s by  
John Holland

# Genetic Algorithm: Overview



# An Example Optimization Problem

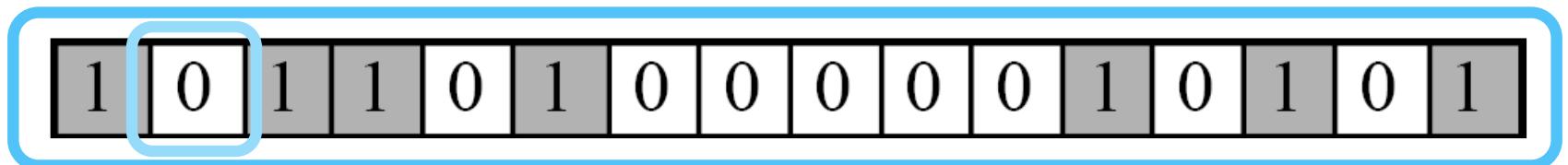
- **Problem:** Find the value of  $x$  (say,  $x^*$ ) that maximizes the function

$$(16x - x^2)$$

- **Constraint:**  $x$  is an integer between 0 and 15

# Solution Form

- **Chromosome:** Bit-string encoded candidate solution (nowadays other encoding forms are possible)
- **Gene:** Each bit of the solution



A chromosome could represent the value of one or more parameters in your algorithm/system, e.g. the length of a robot arm, the value of weights in your neural network, the ids of operations you need to execute in a sequence...

# Encoding Candidate Solutions

- Step 1: Encode candidate solutions ( $x$ ) into chromosomes
  - Since  $x$  can only take 16 different values, 4 bits are sufficient to encode any candidate solution

<i>Integer</i>	<i>Binary code</i>	<i>Integer</i>	<i>Binary code</i>	<i>Integer</i>	<i>Binary code</i>
1	0 0 0 1	6	0 1 1 0	11	1 0 1 1
2	0 0 1 0	7	0 1 1 1	12	1 1 0 0
3	0 0 1 1	8	1 0 0 0	13	1 1 0 1
4	0 1 0 0	9	1 0 0 1	14	1 1 1 0
5	0 1 0 1	10	1 0 1 0	15	1 1 1 1

That's our whole search space encoded!

# Defining the Fitness

- Step 2: Define how the fitness of a candidate solution is measured
- The fitness is computed on the decoded chromosome; in this example:

$$f(x) = 16x - x^2$$

# The Population

- Step 3:  
Randomly select **an initial population of N chromosomes**

Label	Chromosome (genotype)	Decoded (phenotype)
X1	0001	1
X2	0010	2
X3	0100	4
X4	1001	9
X5	1100	12
X6	1111	15

# Using the Fitness Function

- Step 4: Compute the fitness of all the chromosomes in the population

$$f(x) = 16x - x^2$$

Label	Chromosome	Decoded	Fitness
X1	0001	1	15
X2	0010	2	28
X3	0100	4	48
X4	1001	9	63
X5	1100	12	48
X6	1111	15	15
			217

# Fitness Ratio

$$r(x_i) = \frac{f(x_i)}{\sum_j f(x_j)} \times 100\%$$

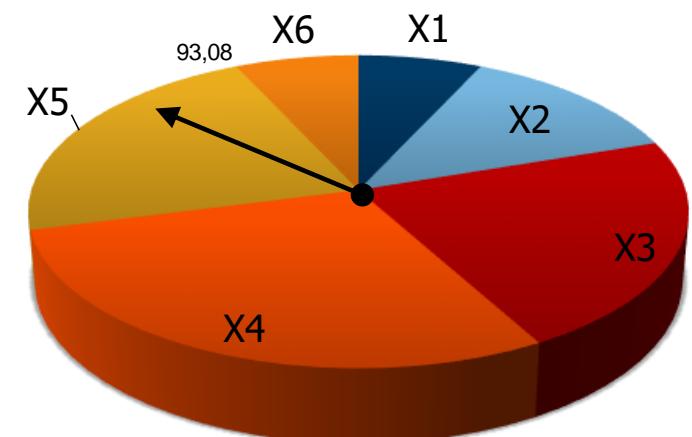
$$r(X_1) = 15 / (15+28+48+63+48+15) * 100\% = 15 / (217) * 100\% = 6.91$$

Label	Chromosome	Decoded	Fitness	Fitness ratio
X1	0001	1	15	6.91
X2	0010	2	28	12.90
X3	0100	4	48	22.11
X4	1001	9	63	29.03
X5	1100	12	48	22.11
X6	1111	15	15	6.912
			217	100

# Selection

- Step 5: Pick the chromosomes for reproduction
- Example: **Roulette Wheel Selection**

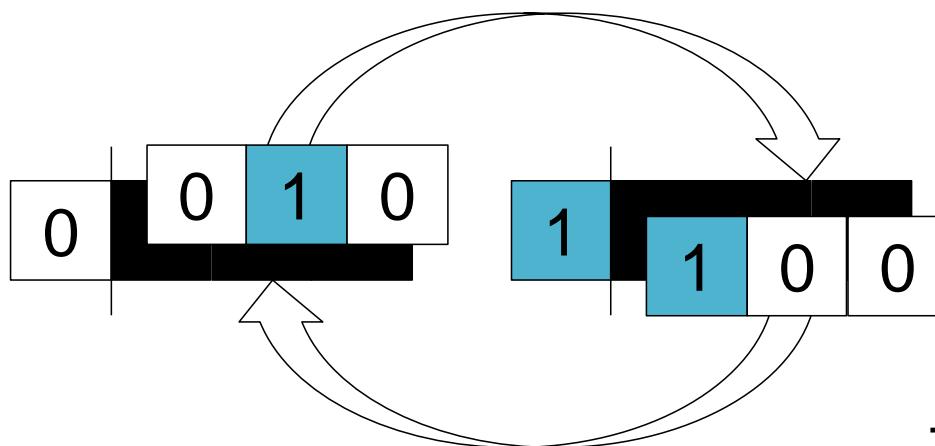
Label	Chromosome	Fitness ratio	Cumulative
X1	0001	6.91	6.91
X2	0010	12.90	19.81
X3	0100	22.11	41.93
X4	1001	29.03	70.96
X5	1100	22.11	93.08
X6	1111	6.912	100
		100	



- Pick a random number between 0 and 1
- Check the “slice” it falls in
- Select the chromosome corresponding to the “slice”
- With this procedure, select two chromosomes, e.g. X2 and X5

# Modification: Crossover

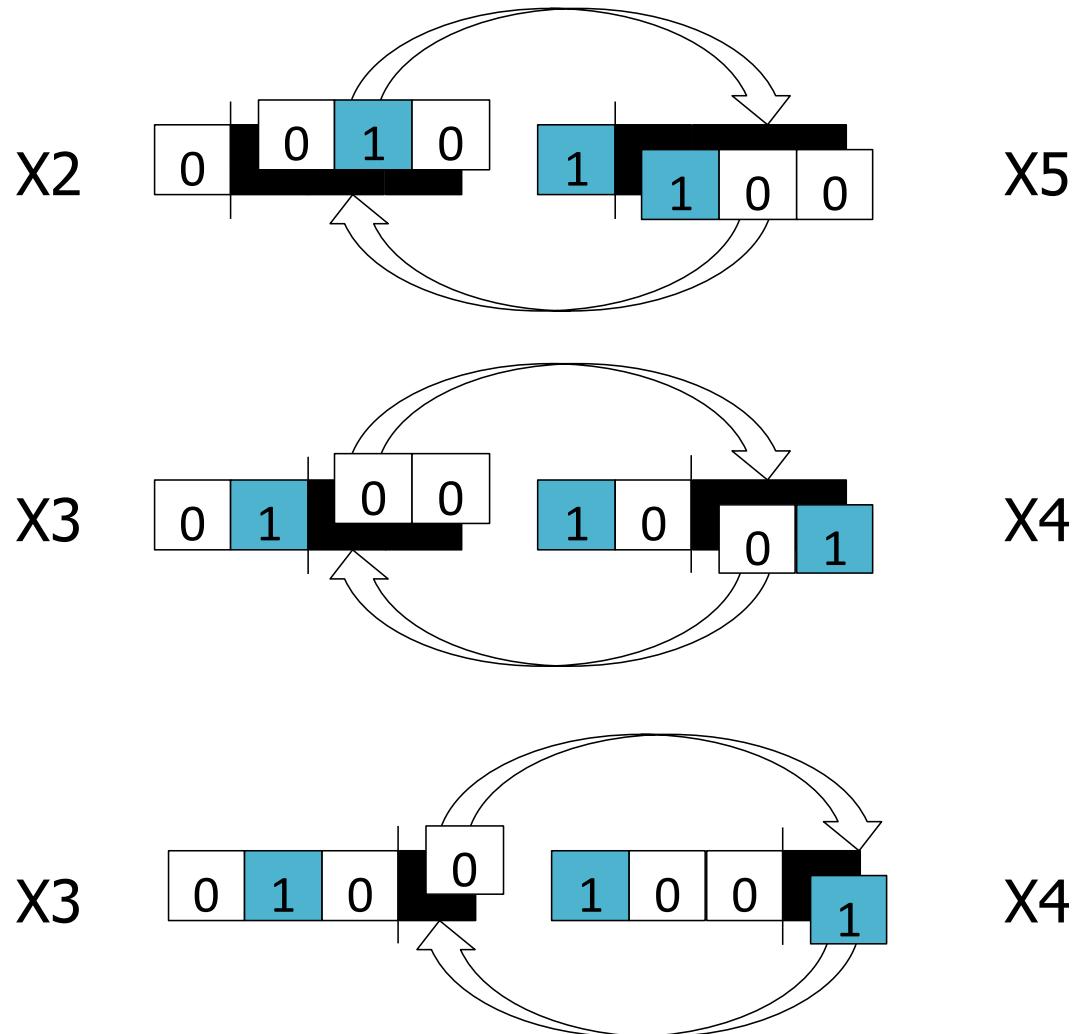
- Step 6a: With (predefined) **probability  $p_c$** , use the two selected chromosomes to generate offspring
  - If no offspring is generated → **cloning**
- Pick a random **cross-over point** and swap the bits after that point among the two chromosomes
- X2: 0010, **X5**: 1100, cross-over point 1



Typically,  $p_c \approx 0.7$

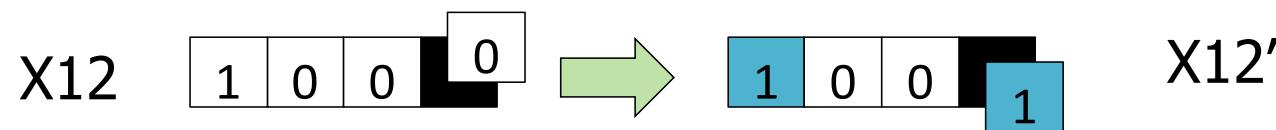
# Crossover

- Pick through selection as many chromosomes as you have in your population
- Recombine them in pairs through crossover with chance  $p_c$



# Modification: Mutation

- Step 6b: With (predefined) **probability  $p_m$** , mutate (flip) the bits in the new chromosomes



- Mutation probability is typically small, otherwise leads to instability (and longer search times)
- Typical values of  $p_m$  range between 0.001 and 0.01

# The Next Generation



- Step 7: Put the offspring in the new population

X7	0100
X8	1010
X9	0101
X10	1000
X11	0101
X12'	1001

- Step 8: Substitute the old population with the new one
- Step 9: Repeat from step 4 (fitness computation)

# Second Generation: Fitness

- Step 4: Compute the fitness of all the chromosomes in the population

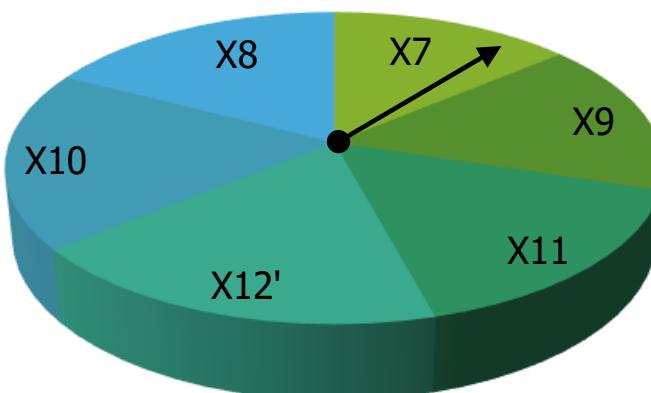
$$f(x) = 16x - x^2$$

Label	Chromosome	Decoded	Fitness
X7	0100	4	48
X9	0101	5	55
X11	0101	5	55
X12'	1001	9	63
X10	1000	8	64
X8	1010	10	60
			345

# Second Generation: Selection

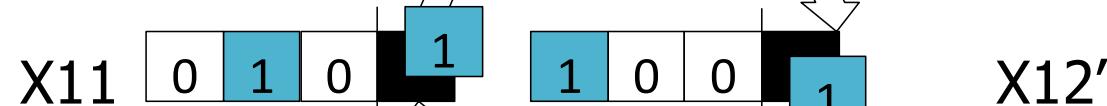
- Step 5: Picking the chromosomes for reproduction

Label	Chromosome	Decoded	Fitness	Fitness ratio	Cumulative
X7	0100	4	48	13.91	13.91
X9	0101	5	55	15.94	29.85
X11	0101	5	55	15.94	45.79
X12'	1001	9	63	18.26	64.05
X10	1000	8	64	18.55	82.60
X8	1010	10	60	17.39	100
			345	100	



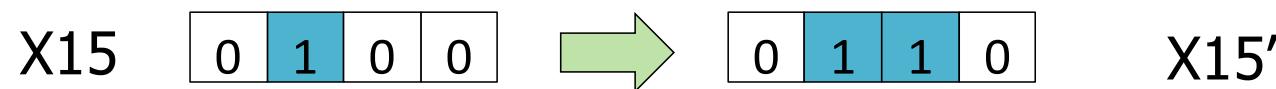
# Second Generation: Crossover

- Step 6



# Second Generation: Mutation

- Step 6



- And of course step 7 and 8 (replace old population with offspring)

# Third Generation: Fitness

- Step 4: Compute the fitness of all the chromosomes in the population

$$f(x) = 16x - x^2$$

Label	Chromosome	Decoded	Fitness
X13	0110	5	55
X15'	0110	6	48
X17	1000	8	64
X18	1001	9	63
X16	1001	9	63
X14	1001	9	63
			368

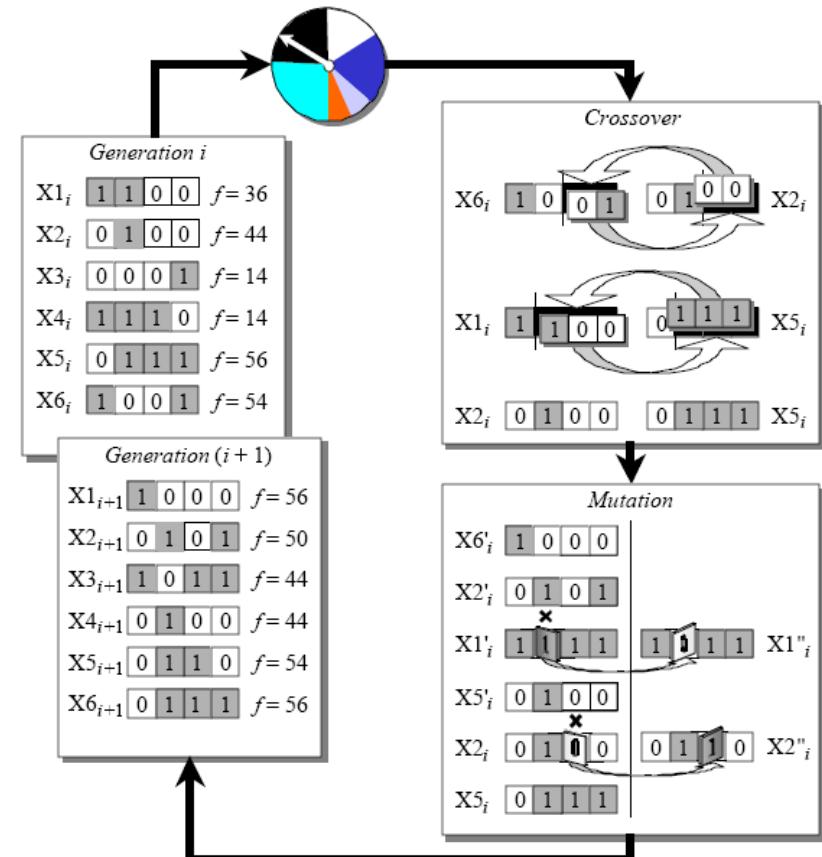
- Step 5, 6, 7, and 8: You get the idea...

# Genetic Algorithm: Termination

- The process stops if:
  - A sufficiently good solution is found, or
  - A predefined number of iterations (generations) has been executed

# Genetic Algorithm: Recap

1. Define a coding to transform solutions into chromosomes
2. Define the fitness function
3. Select a random initial population
4. Compute fitness for all candidates
5. Select chromosomes for reproduction
6. Cross-over, Mutation
7. Put offspring into new population
8. Switch old population with the new one
9. Repeat 4-8 until you find a sufficiently good solution or for a defined number of iterations



# Genetic Algorithm: Variations

- **Coding:**

- Binary coding
- Gray coding (avoids Hamming cliffs)
- ...

- **Selection:**

- Roulette-wheel selection
- Ranking selection
- Tournament selection

- **Cross-over**

- Single-point cross-over
- Multi-point cross-over
- Uniform cross-over

# Genetic Algorithm: (Hyper)Parameters

- Length of the chromosome,  $L$ 
  - Depending in the precision required for the problem
- Population Size,  $N$ 
  - Larger the search space, larger the  $N$  should be
- Cross-over probability
  - Generally high, e.g., 0.7
- Mutation probability
  - Generally low, e.g.,  $1/L$  Or  $0.1/L$
- Number of generations,  $G$ 
  - Depends on the computational resources available

Optimal values of  $N$ ,  $P_c$ ,  $P_m$ ,  $G$  can be chosen **empirically**

# Multiple Variables

# An Example with Two Variables

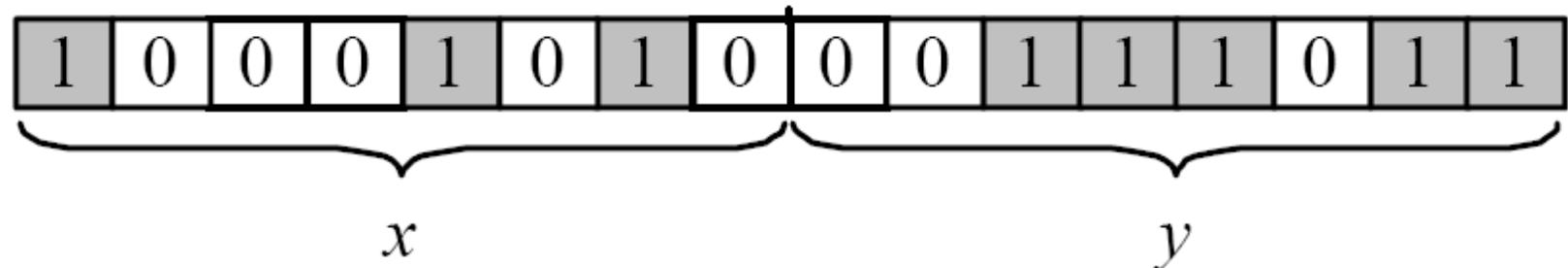
- Suppose that we want to find the maxima of a function of two parameters  $x$  and  $y$ , each a continuous variable with values between -3 and 3:

$$f(x, y) = (1-x)^2 e^{-x^2-(y+1)^2} - (x - x^3 - y^3) e^{-x^2-y^2}$$

How do we encode this? Ideas?

# A slightly more complicated example

We use 8 bits (binary) encoding for x and y



To decode them, we first convert them into decimal values

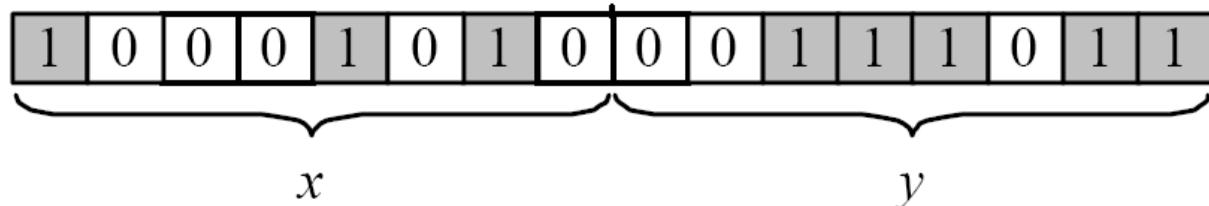
1	0	0	0	1	0	1	0
---	---	---	---	---	---	---	---

$$1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 138$$

0	0	1	1	1	0	1	1
---	---	---	---	---	---	---	---

$$0 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 59$$

## Two Variables: Encoding



We end up with values in [0,255], and we need to re-scale them into [-3, 3]:

$$x_s = \frac{(x_u - x_{u,min})(x_{s,max} - x_{s,min})}{x_{u,max} - x_{u,min}} + x_{s,min}$$

$$x = (138 - 0) (3 - -3) / (255 - 0) + -3 = 138 \times 6 / 255 - 3 = 0.247$$

$$y = (59 - 0) (3 - -3) / (255 - 0) + -3 = 59 \times 6 / 255 - 3 = -1.612$$

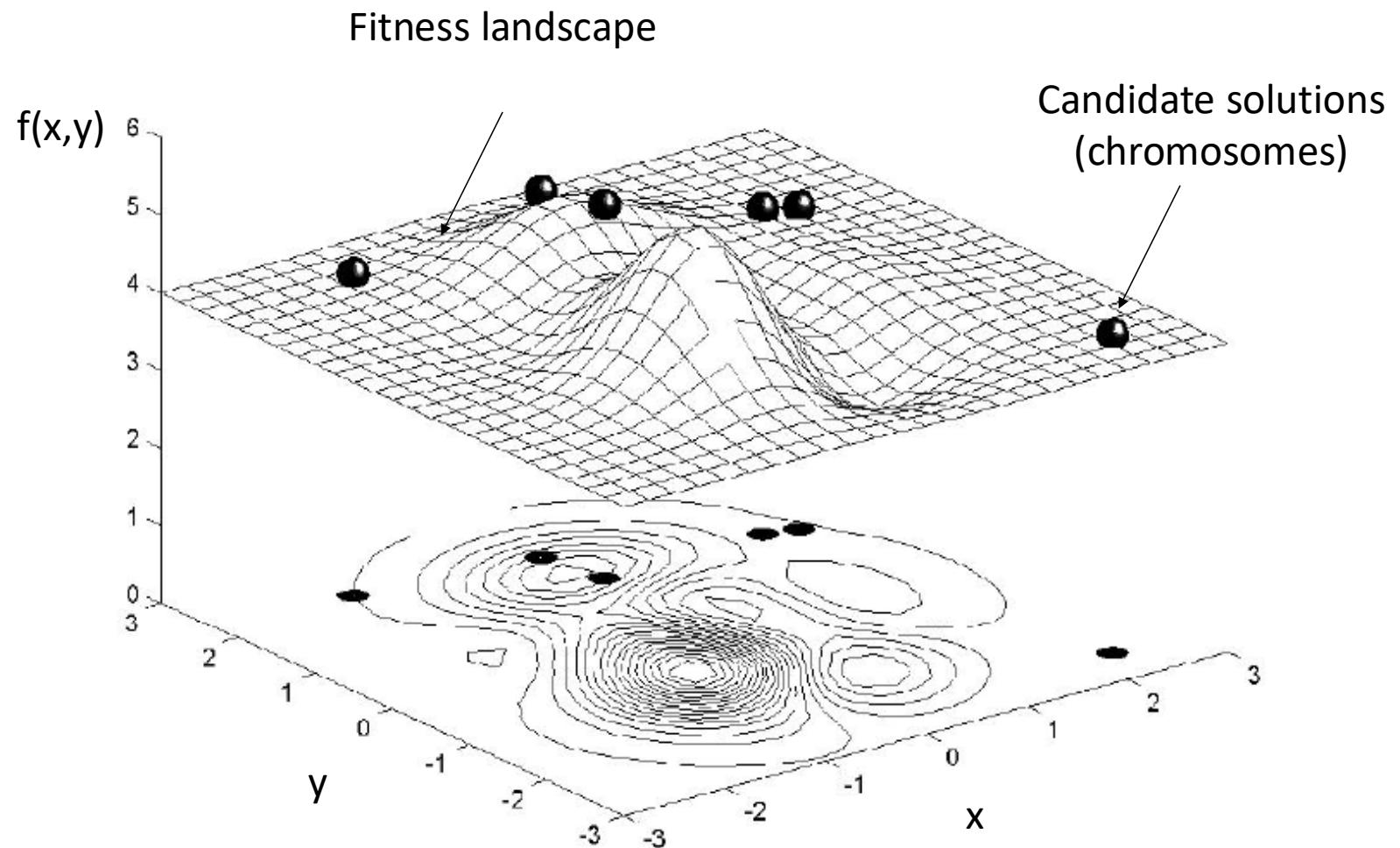
# Two Variables: Fitness

- Fitness function:

$$f(x, y) = (1 - x)^2 e^{-x^2 - (y+1)^2} - (x - x^3 - y^3) e^{-x^2 - y^2}$$

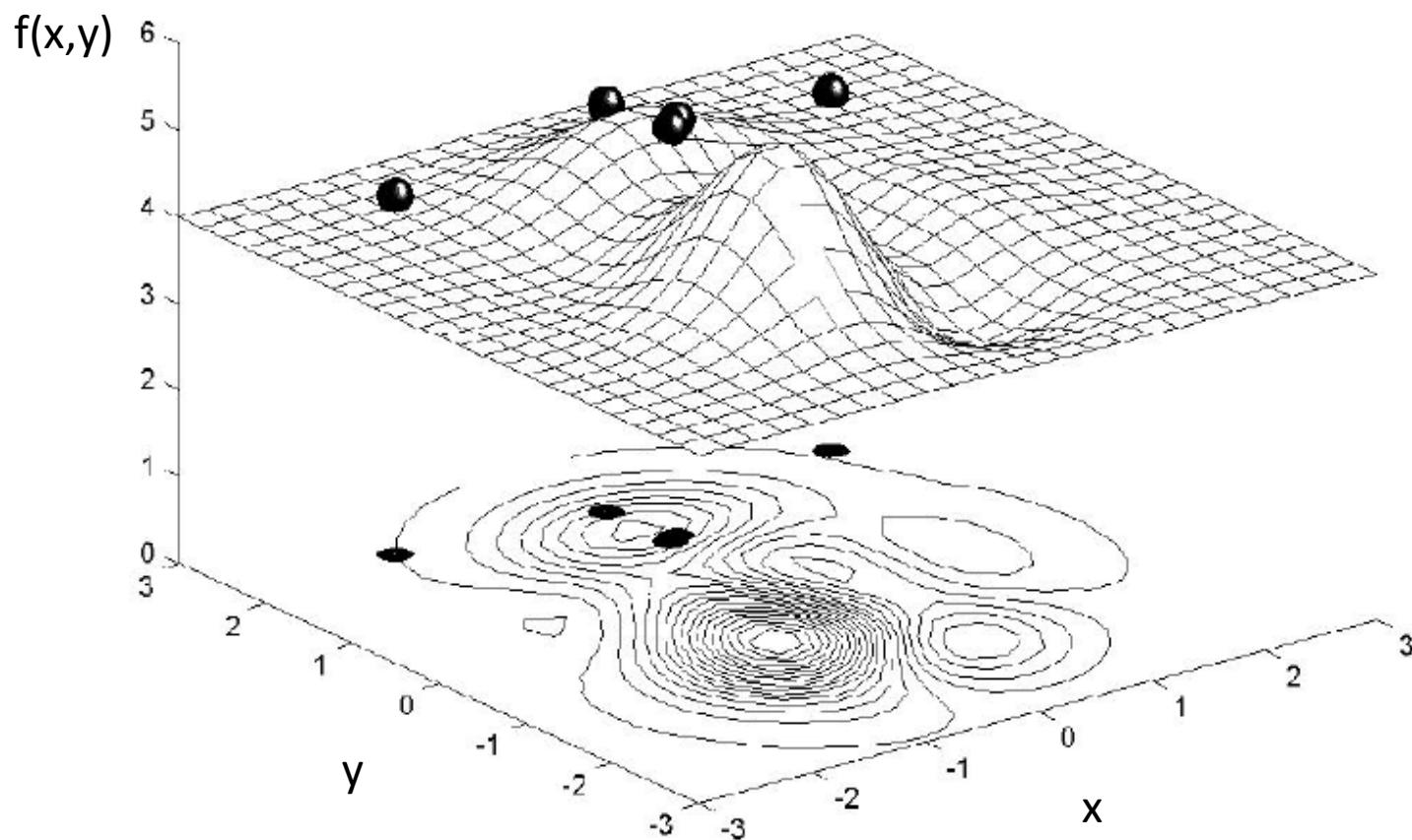
- We choose a population with  $N = 6$ 
  - Random bit-strings
- Crossover probability  $p_c = 0.7$
- Mutation probability  $p_m = 0.001$

# Two Variables: Progress



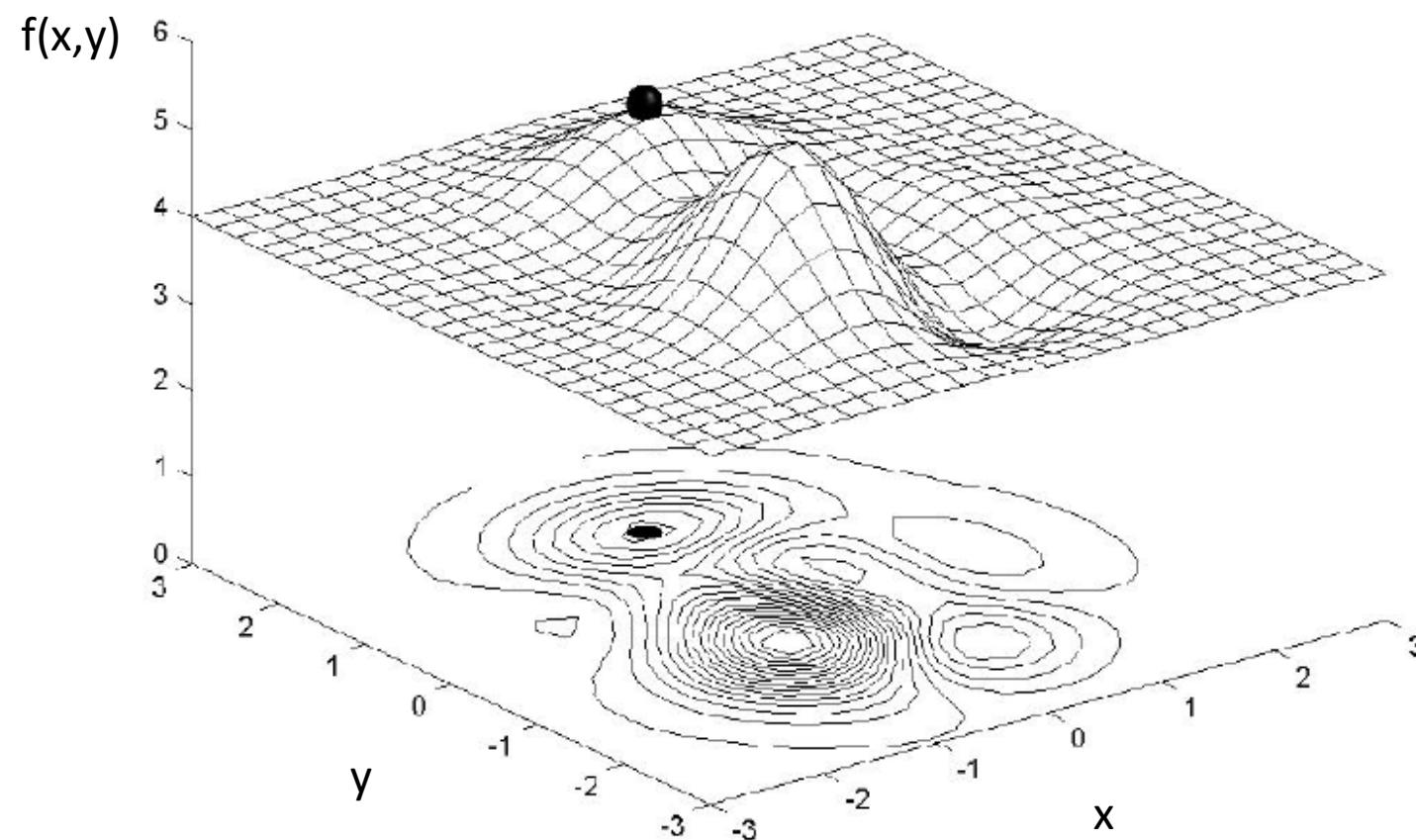
# Two Variables: Progress

First generation



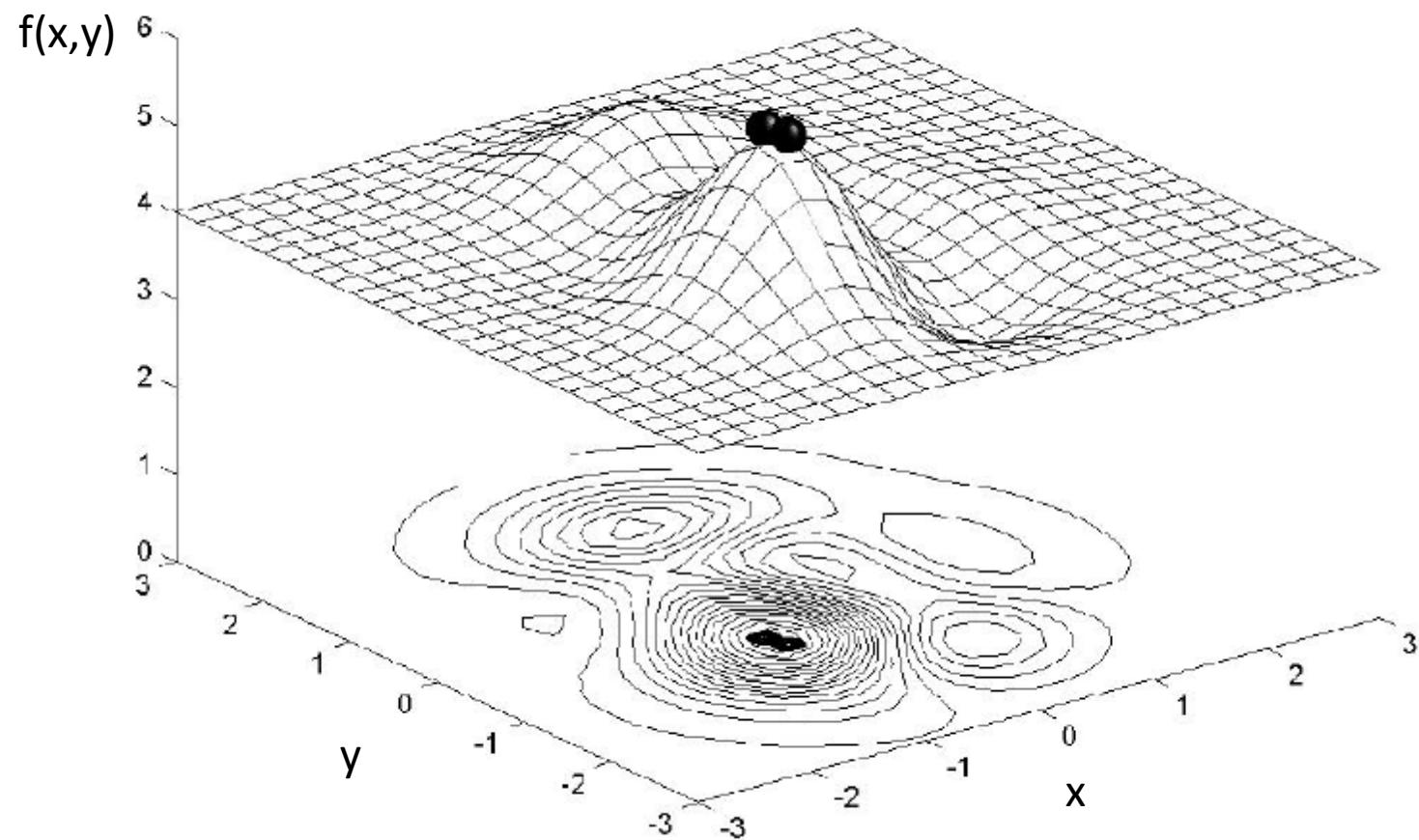
# Two Variables: Progress

A local maximum



# Two Variables: Progress

The global maximum



# Why does GA Work?

# Why do GA work?

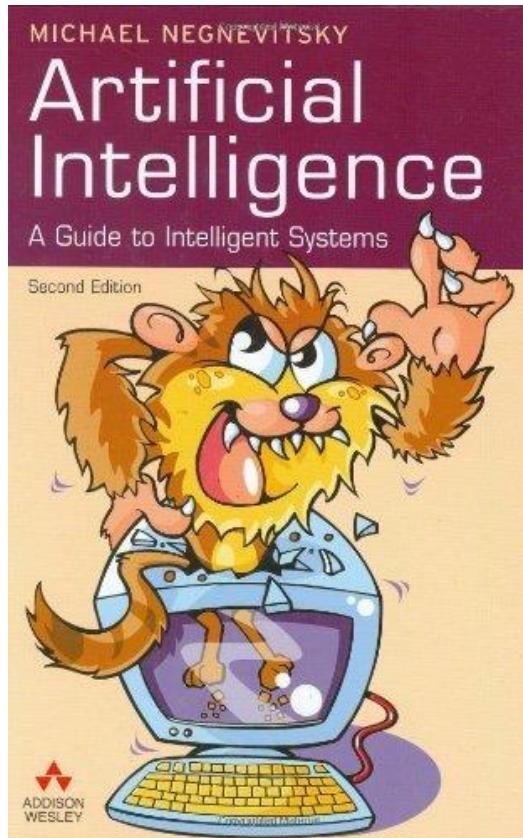
- **Holland's schema theorem:**

The presence of

- *short*,
- *low-order* schema with
- *above-average fitness*

increases in successive generations of GA.

# Schema Theorem: Details and Proof



## Self Study: Section 7.4

You will **not** be asked to prove the theorem in the exam.

# Schema Theorem: Criticism

- Almost a tautology describing only the proportional selection, but not the benefits of cross-over and mutation
- Excludes the re-emergence of a schema after recombination
- ...holds under the assumption of an infinitely large population
  - small populations can introduce sampling erroring, resulting in convergence on schema that have no selective advantage
- Schema theorem does NOT imply that
  - GA finds global optima
  - GA is better than other optimization techniques
  - ...

# More (Visual) Examples

# THE NATURE OF CODE

DANIEL SHIFFMAN

## Chapter 9. The Evolution of Code

*“The fact that life evolved out of nearly nothing, some 10 billion years after the universe evolved out of literally nothing, is a fact so staggering that I would be mad to attempt words to do it justice.”*

— Richard Dawkins

Let's take a moment to think back to a simpler time, when you wrote your first Processing sketches and life was free and easy. What is one of programming's fundamental concepts that you likely used in those first sketches and continue to use over and over again? *Variables*. Variables allow you to save data and reuse that data while a program runs. This, of course, is nothing new to us. In fact, we have moved far beyond a sketch with just one or two variables and on to more complex data structures—variables made from custom types (objects) that include both data and functionality. We've made our own little worlds of movers and particles and vehicles and cells and trees.

<https://github.com/shiffman/The-Nature-of-Code-Examples-p5.js>

# To Summarize

## Evolutionary computation

- Solve optimization problems by mimicking natural evolution

## Genetic algorithm

- Problem encoding in chromosomes
- Definition of fitness
- Selection
- Reproduction: Crossover and mutation
- Termination

## Next time...

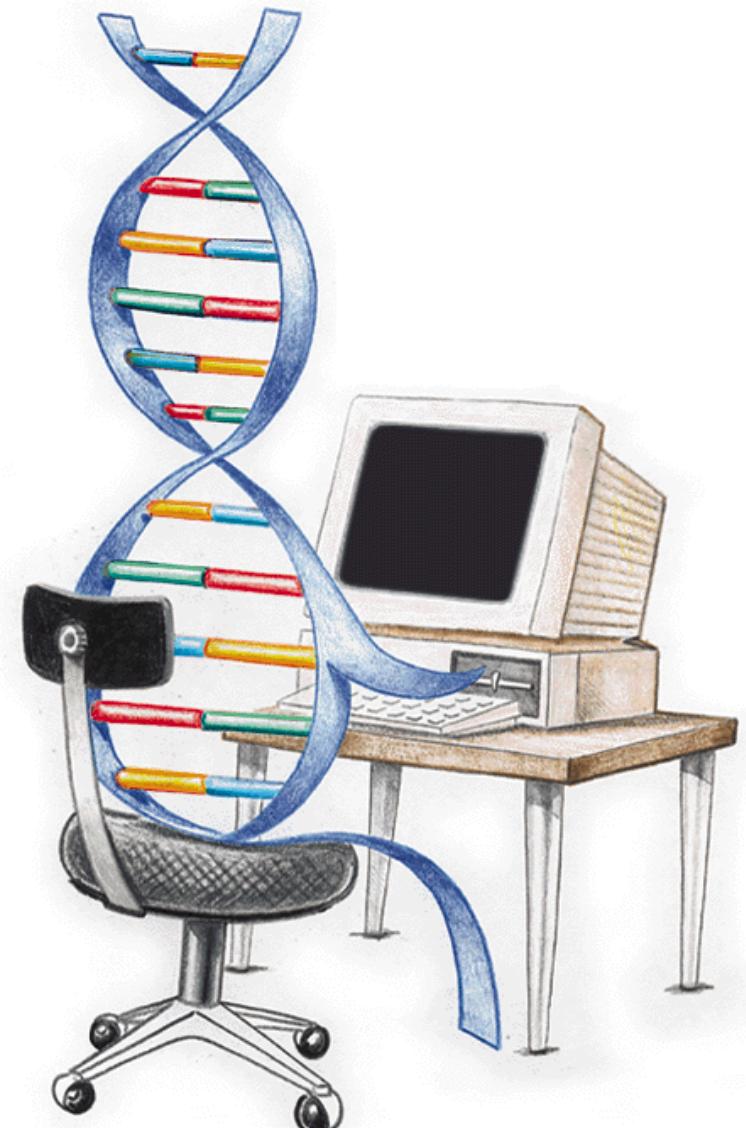
- Performance Graphs
- A practical problem (scheduling)
- Multiobjective optimization

# Genetic Algorithm

**CSE-2530:**  
**Computational Intelligence**

Pradeep K. Murukannaiah

(Slides adapted from Joost Broekens  
and Jazmin Salazar)



# Recap

After the previous lecture, you should be able to

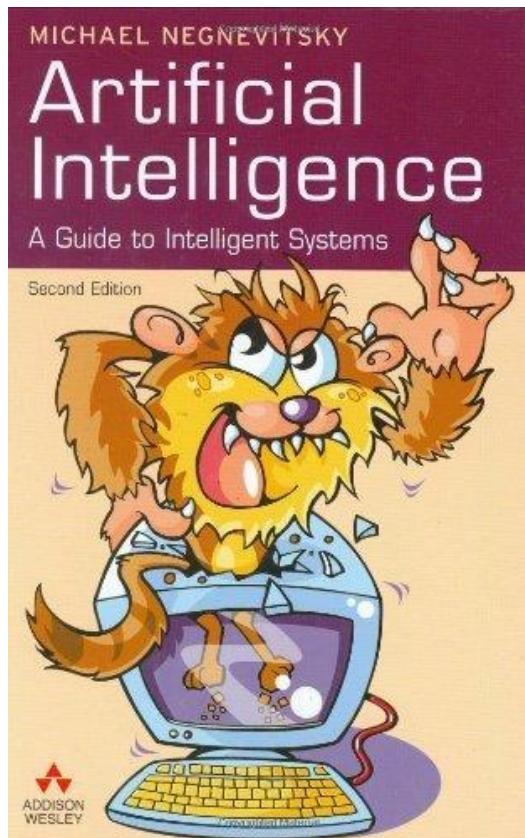
- Explain the intuition behind **evolutionary computing**
- Recognize **problems** that can be modeled via evolutionary computing methods
- Enumerate the steps of a **genetic algorithm** (GA)
- Describe the key GA operations: **selection**, **cross-over**, and **mutation**
- Apply GA to **optimization problems**

# Learning Objectives

After today's lecture you should be able to:

- Analyze GA via **peformance graphs**
- Apply GA to a practical **scheduling problem**
- Recognize **multi-objective optimization** problems
- Describe **NSGA-II**, a GA for multi-objective optimization

# Study Material



Chapter 7: Sections 7.5

Lecture slides

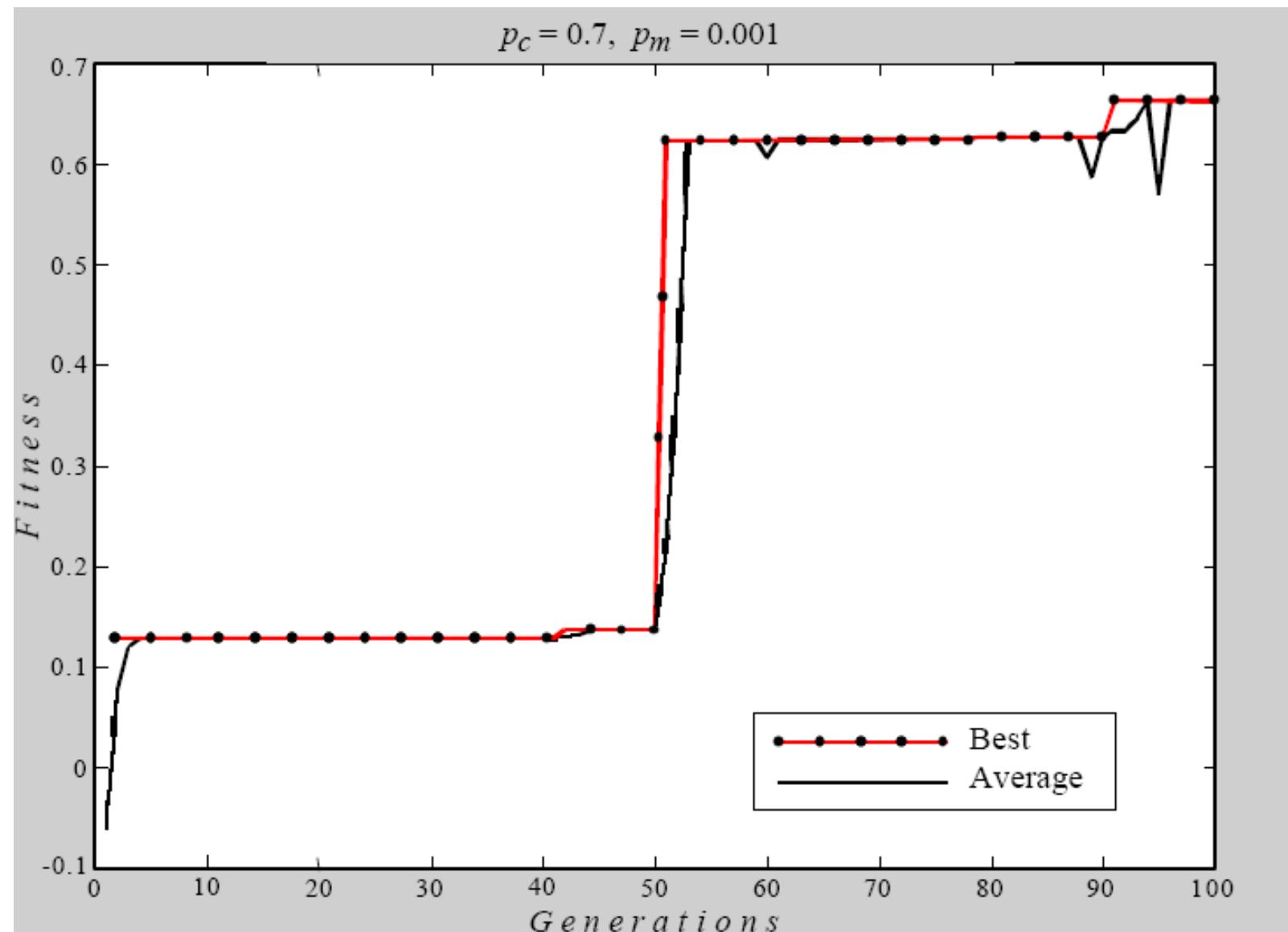
# Performance Graphs

# Performance Graphs

- How do we know whether a solution we found is good?
- Performance graph:
  - Average fitness of the population over time
  - Fitness of the fittest individual per population

$P_c = 0.7$   
 $P_m = 0.001$

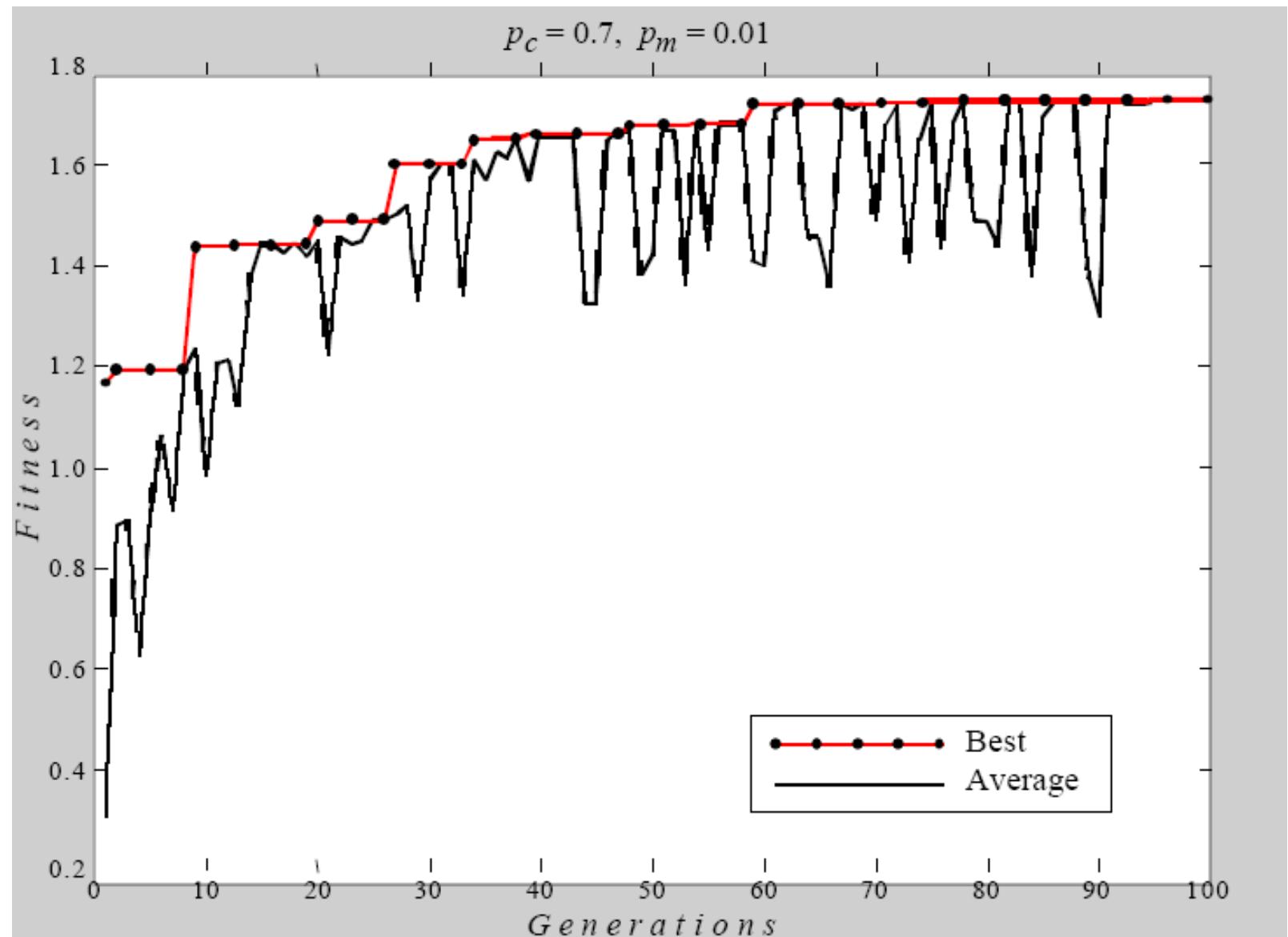
# Performance Graphs



100 generations, 6 chromosomes, local maximum

$P_c = 0.7$   
 $P_m = 0.01$

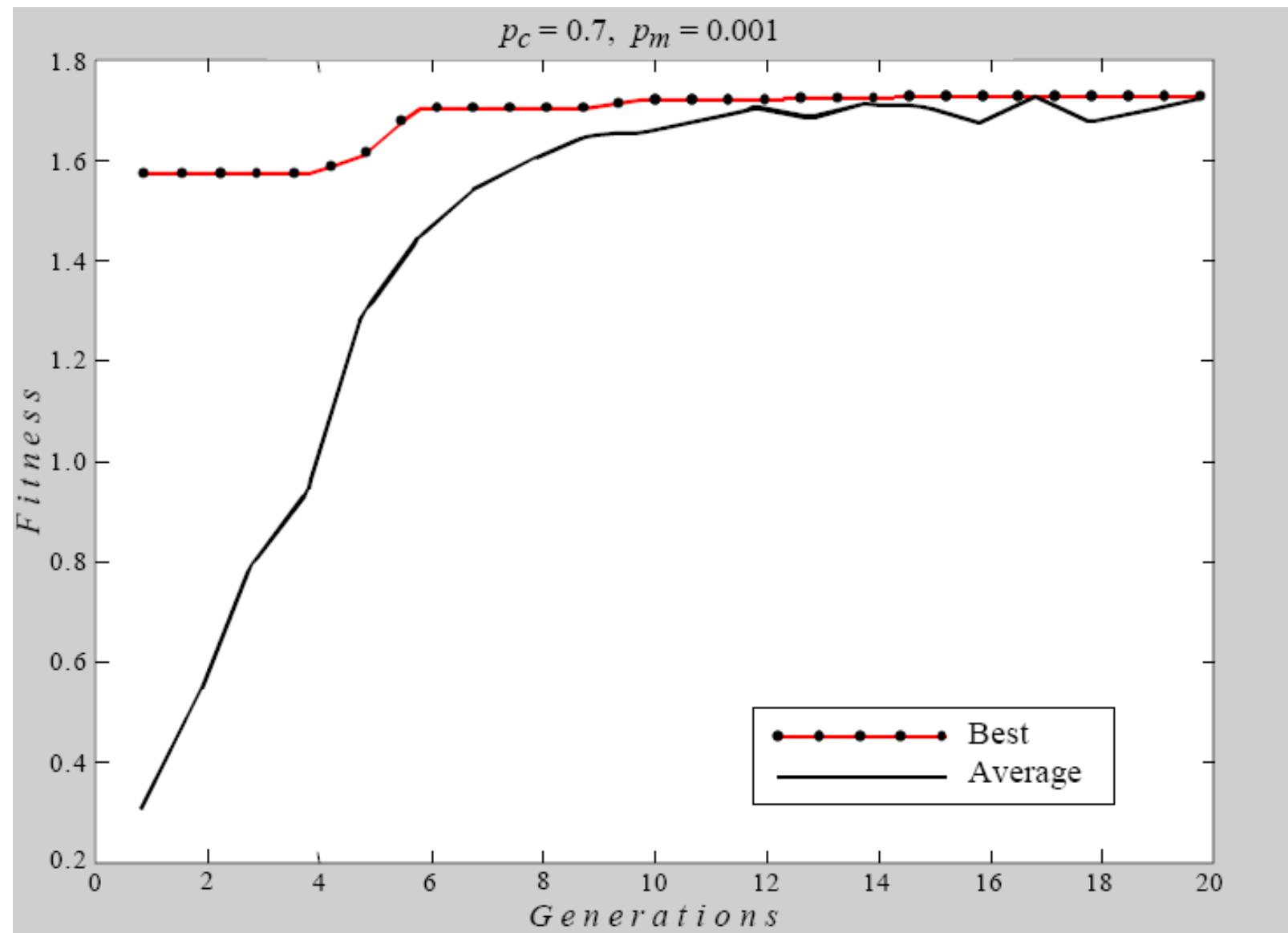
# Performance Graphs



100 generations, 6 chromosomes, global maximum

# Performance Graphs

$P_c = 0.7$   
 $P_m = 0.001$



20 generations, 60 chromosomes, global maximum

# A Case Study

# GA Case Study: Maintenance Schedule for Power systems

- Scheduling is complex
  - A lot of requirements and constraints
  - NP complete (and not all solutions satisfy the requirements – remember the TUDelft schedule?)
- Scenario: **Yearly** maintenance of power units in a power system
  - Should be done to prevent unexpected breakdowns
  - A unit cannot generate power during its maintenance → shortage
  - Units should be maintained based on the capacity of the system to generate the required power, nevertheless

# Case Study: Maintenance Schedule

- Expected loads per quarter: 80 MW, 90 MW, 65 MW, 70 MW
- Total capacity: 150 MW

- Unit capacities:

<i>Unit number</i>	<i>Unit capacity, MW</i>	<i>Number of intervals required for unit maintenance</i>
1	20	2
2	15	2
3	35	1
4	40	1
5	15	1
6	15	1
7	10	1

- Maintenance takes 1 or 2 quarters (no interruption)

# Case Study: Maintenance Schedule

## Problem:

Generate a maintenance schedule for the whole year

## Activity:

- How would you encode solutions to this problem?
- How large is the search space?
- What fitness function would you use?
- How would you perform cross-over and mutation?

# Maintenance Schedule: Encoding

How does a candidate solution look like in this scenario?

- Candidate solution = Schedule
- Multiple units can be maintained at the same time

How do we encode the candidate solutions into chromosomes?

- Encode each unit into a 4 bit string
- Each bit indicates the status of the unit in the quarter
  - 1 = in maintenance for that quarter
  - 0 = otherwise
- Chromosome of 28 bits

Unit 1	Unit 2	Unit 3	Unit 4	Unit 5	Unit 6	Unit 7
0 1 1 0	0 0 1 1	0 0 0 1	1 0 0 0	0 1 0 0	0 0 1 0	1 0 0 0

# Maintenance Schedule: Encoding

Redefine the syntax:  
A gene is an **indivisible** 4-bit string

Gene pools:      Unit 1 is in maintenance for the first two quarters

Unit 1:	1 1 0 0	0 1 1 0	0 0 1 1	
Unit 2:	1 1 0 0	0 1 1 0	0 0 1 1	
Unit 3:	1 0 0 0	0 1 0 0	0 0 1 0	0 0 0 1
Unit 4:	1 0 0 0	0 1 0 0	0 0 1 0	0 0 0 1
Unit 5:	1 0 0 0	0 1 0 0	0 0 1 0	0 0 0 1
Unit 6:	1 0 0 0	0 1 0 0	0 0 1 0	0 0 0 1
Unit 7:	1 0 0 0	0 1 0 0	0 0 1 0	0 0 0 1

Unit 1	Unit 2	Unit 3	Unit 4	Unit 5	Unit 6	Unit 7
0 1 1 0	0 0 1 1	0 0 0 1	1 0 0 0	0 1 0 0	0 0 1 0	1 0 0 0

# Maintenance Schedule: Crossover

Crossover operator does not cut across genes

*Parent 1*

0	1	1	0	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	1	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

*Parent 2*

1	1	0	0	0	1	1	0	0	1	0	0	0	1	0	0	1	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

*Child 1*

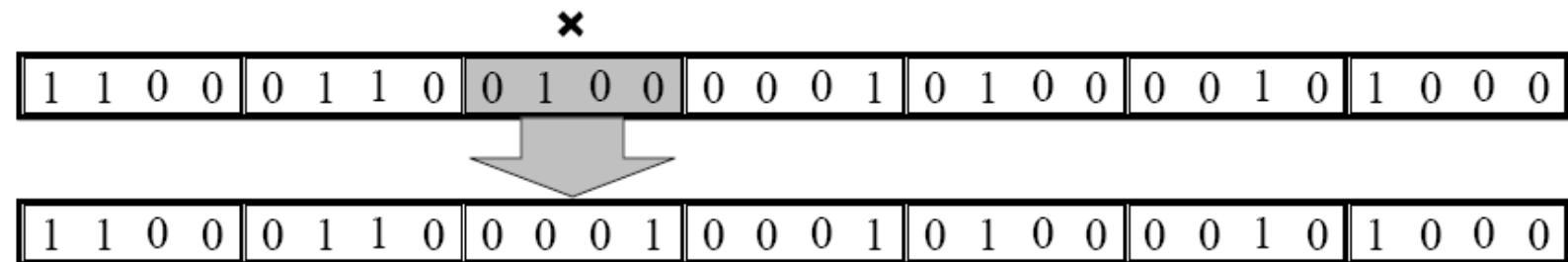
0	1	1	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

*Child 2*

1	1	0	0	0	1	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

# Maintenance Schedule: Mutation

- Mutation selects another gene from the allowed ‘gene pool’
  - No individual bit flipping



Unit 1:	1 1 0 0	0 1 1 0	0 0 1 1
Unit 2:	1 1 0 0	0 1 1 0	0 0 1 1
Unit 3:	1 0 0 0	0 1 0 0	0 0 1 0
Unit 4:	1 0 0 0	0 1 0 0	0 0 1 0
Unit 5:	1 0 0 0	0 1 0 0	0 0 1 0
Unit 6:	1 0 0 0	0 1 0 0	0 0 1 0
Unit 7:	1 0 0 0	0 1 0 0	0 0 1 0

# Maintenance Schedule: Fitness

## *Fitness function*

A solution is a schedule for the whole year, for all units

Even if some units are in maintenance, some **reserve** of power is desirable:

$$\text{Net reserve} \geq 0$$

- Compute the reserve per quarter:  
**Net reserve** =  
= total capacity – maintained unit capacity – expected load
- Take the **minimum** value of Net reserve over the 4 quarters (worst case scenario)
- If Net reserve < 0 : fitness = 0

# Maintenance Schedule: Fitness

Unit number	Unit capacity, MW
1	20
2	15
3	35
4	40
5	15
6	15
7	10

Compute the reserve per quarter:

**Net reserve =**

= total capacity – maintained unit capacity – expected load

Unit 1	Unit 2	Unit 3	Unit 4	Unit 5	Unit 6	Unit 7
0 1 1 0	0 0 1 1	0 0 0 1	1 0 0 0	0 1 0 0	0 0 1 0	1 0 0 0

Quarter 1: Net reserve = 150MW – U4 – U7 – 80MW = 20MW

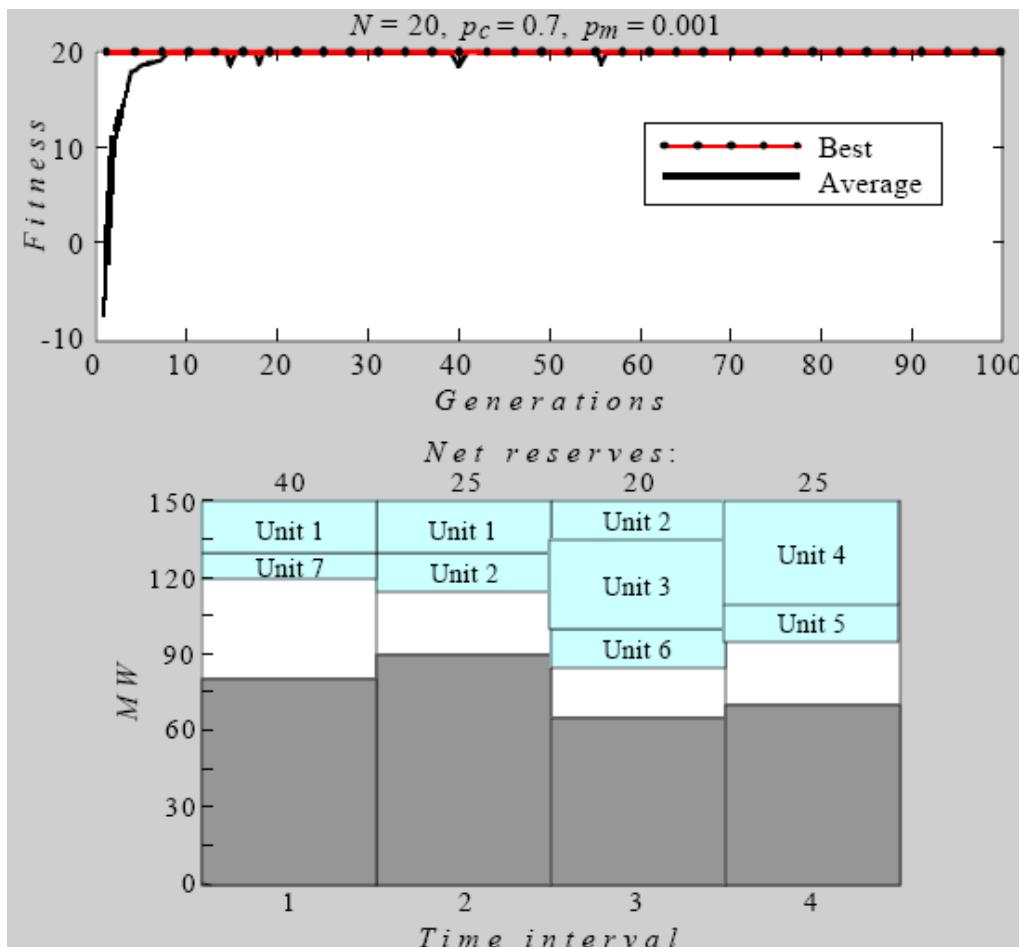
Quarter 2: Net reserve = 150MW – U1 – U5 – 90MW = 25MW

Quarter 3: Net reserve = 150MW – U1 – U2 – U6 – 65MW = 35MW

Quarter 4: Net reserve = 150MW – U2 – U3 – 70MW = 30MW

→ Fitness = 20 (minimum across the quarters)

# Maintenance Schedule



Population:  
 $N = 20$

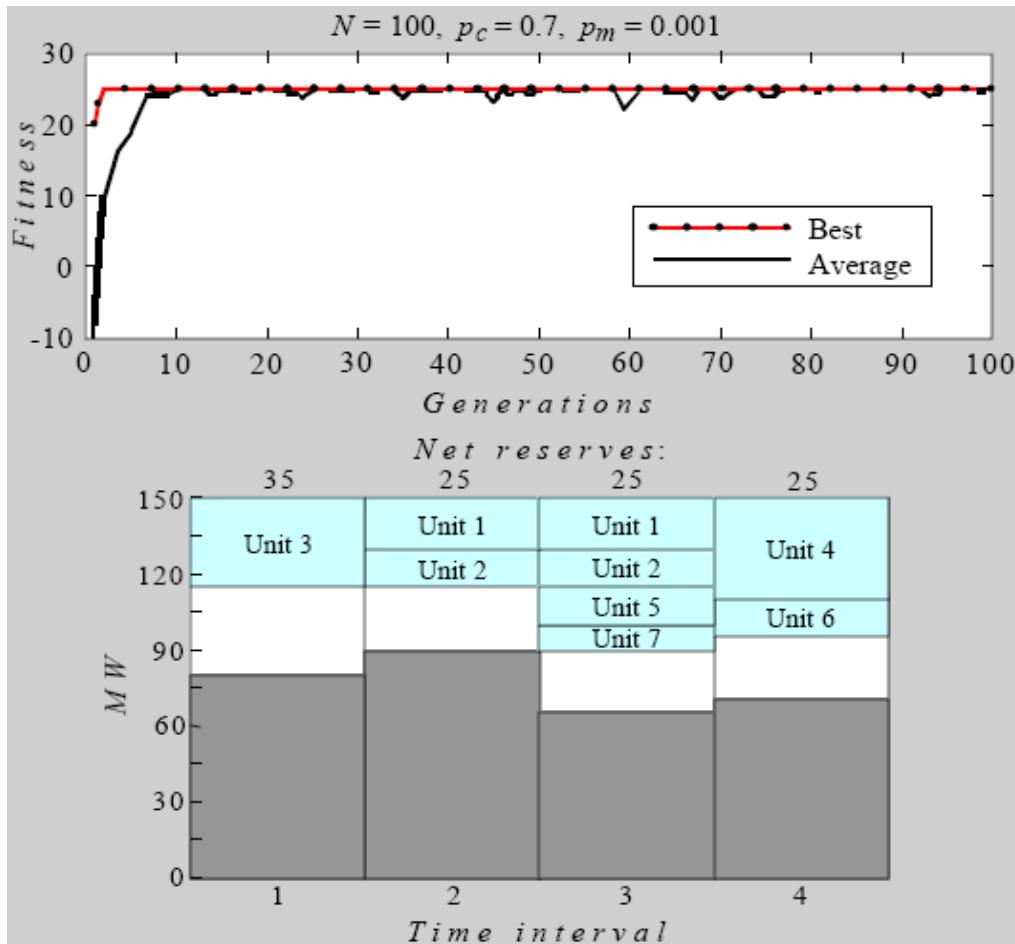
Crossover prob:  
 $P_c = 0.7$

Mutation prob:  
 $P_m = 0.001$

Generations:  
 $i = 100$

**Correction:** Given the fitness function Slide 18, we cannot have negative average fitness as the performance graph above shows in the initial iterations.

# Maintenance Schedule: Performance



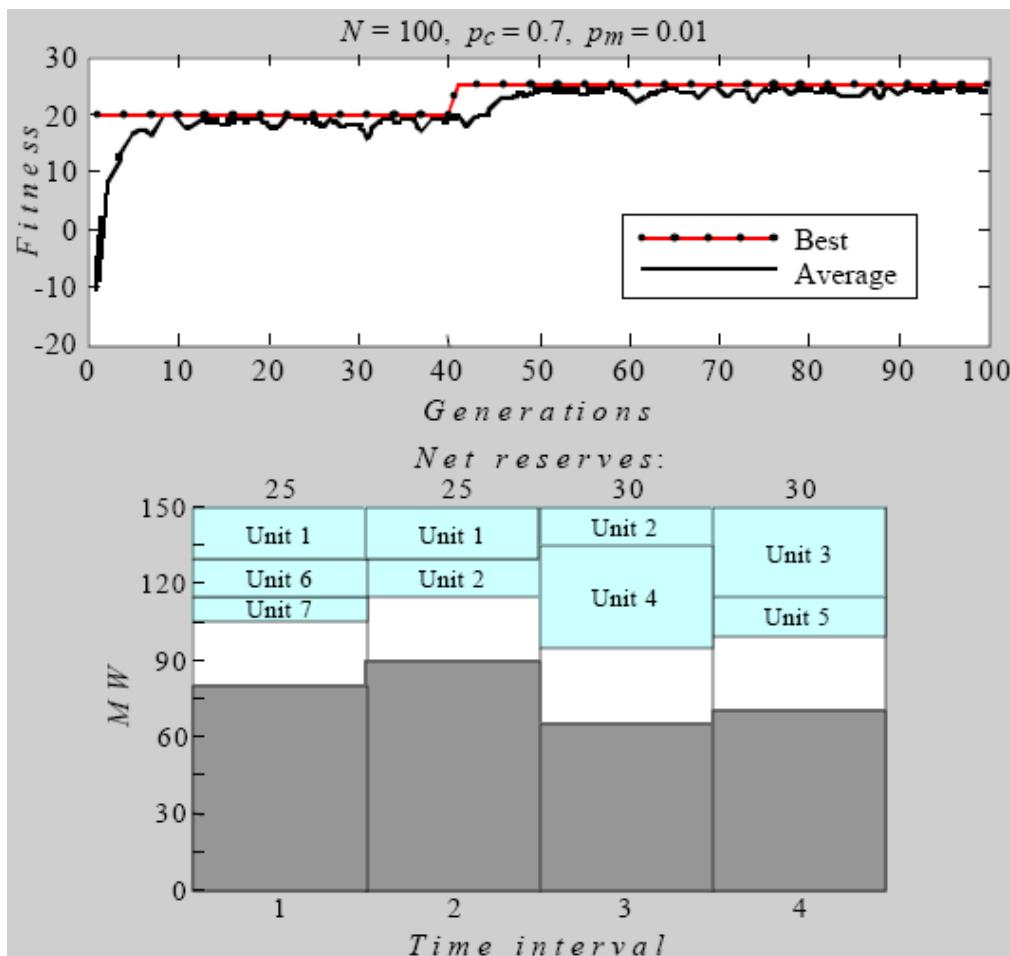
Population:  
 $N = 100$

Crossover prob:  
 $P_c = 0.7$

Mutation prob:  
 $P_m = 0.001$

Generations:  
 $i = 100$

# Maintenance Schedule: Performance



Population:  
 $N = 100$

Crossover prob:  
 $P_c = 0.7$

Mutation prob:  
 $P_m = 0.01$

Generations:  
 $i = 100$

# Multi-Objective Optimization with GA

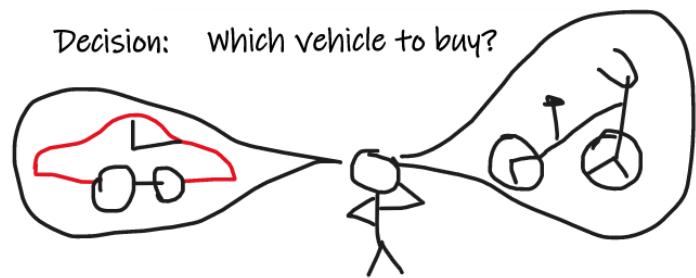
# Why Multi-Objective Optimization?



Decisions often entail **multiple actors** with **conflicting interests**

# An Example Multi-Objective Problem

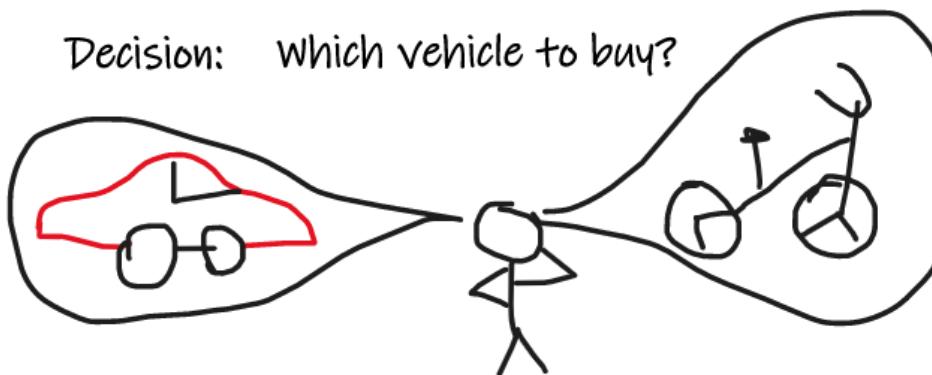
You are tasked to buy a vehicle as a group.



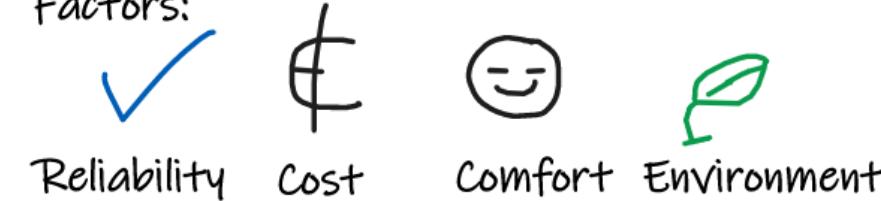
To help guide your choices, answer the following questions:

1. What factors or goals do you consider when buying a vehicle?
2. How would you measure each of the factors or goals that you defined? In other words, how would you quantify how well does your decision meet those goals?
3. What constraints do you encounter when making that decision?

# Defining a Multi-Objective Problem



Factors:



Reliability

Cost

Comfort

Environment

Constraint: The budget



A multi-objective problem consists of **two or more** objectives that conflict with each other

Alternative strategies that decision makers want to explore

Decisions

Objectives

Constraints

Measures to rank the desirability of a certain decision

Condition that needs to be satisfied often non-negotiable

# General Form of a Multi-Objective Optimization Problem

min/max  $f_m(\mathbf{x}), \quad m=1, 2, \dots, M$

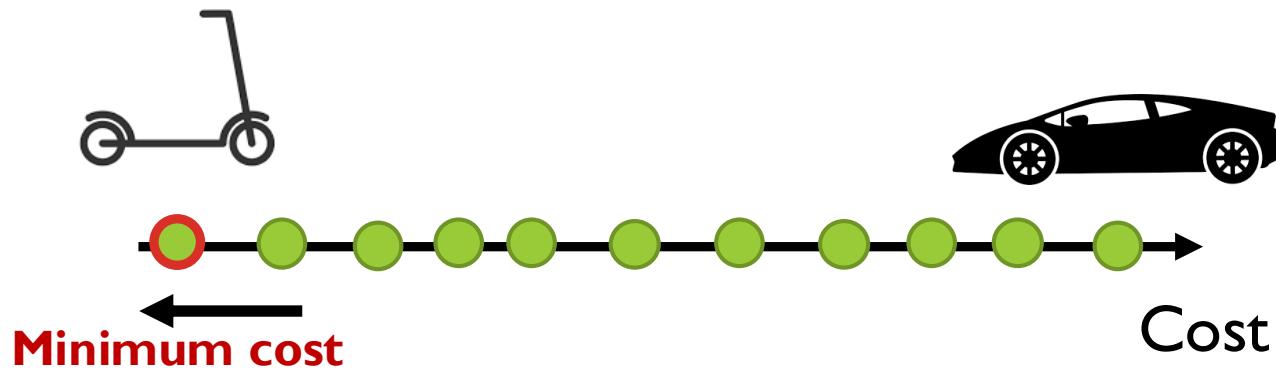
subject to  $g_j(\mathbf{x}) \geq 0, \quad j=1, 2, \dots, J$

$h_k(\mathbf{x}) = 0, \quad k=1, 2, \dots, K$

$\underset{\text{lower bound}}{x_i^{(L)}} \leq x_i \leq \underset{\text{upper bound}}{x_i^{(U)}}, \quad i=1, 2, \dots, n$

# Solving Single Objective vs. Multi-Objective Problems

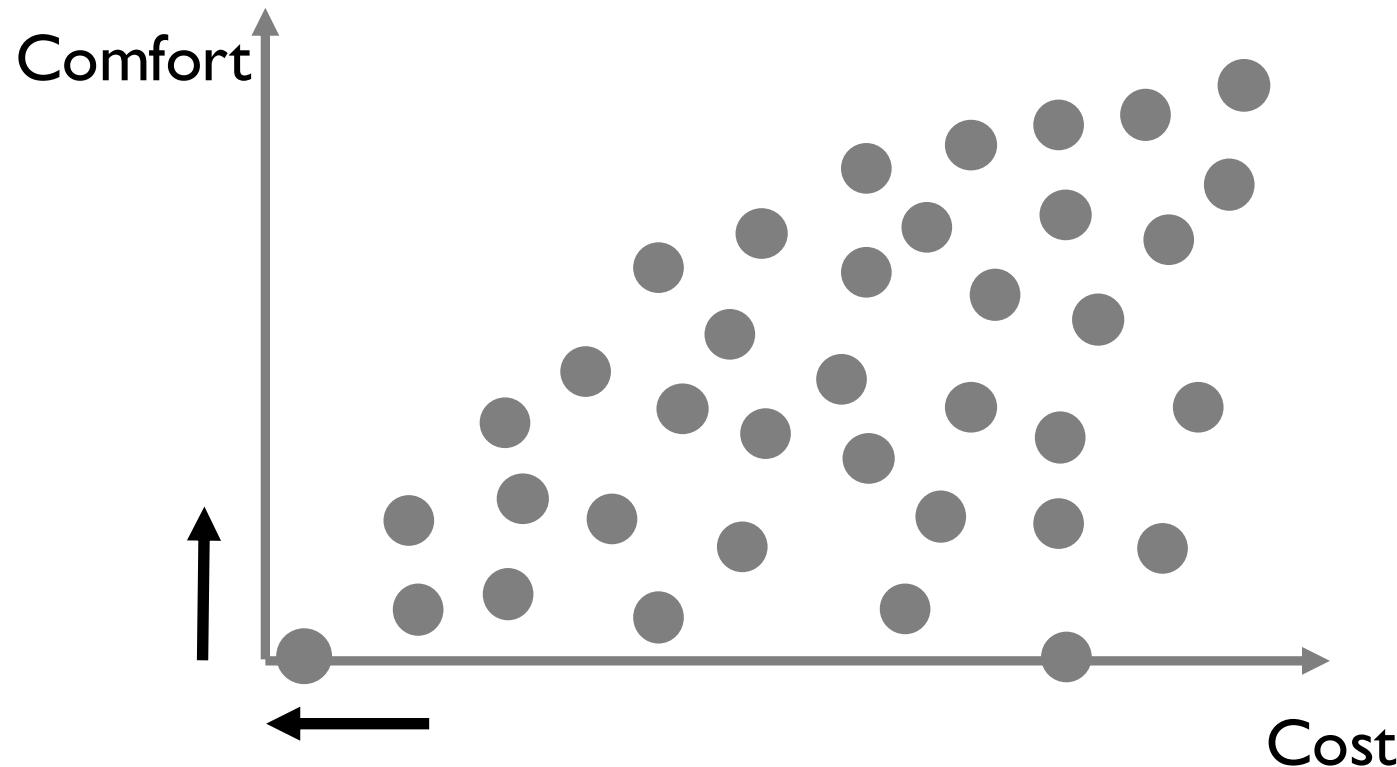
- **Single objective problem:**



- Find a **solution** that optimizes the given objective

# Solving Single Objective vs. Multi-Objective Problems

- **Multi-objective problem:**



- Find **the set of solutions** that define the best possible **tradeoffs** between competing objectives

# Pareto Optimality

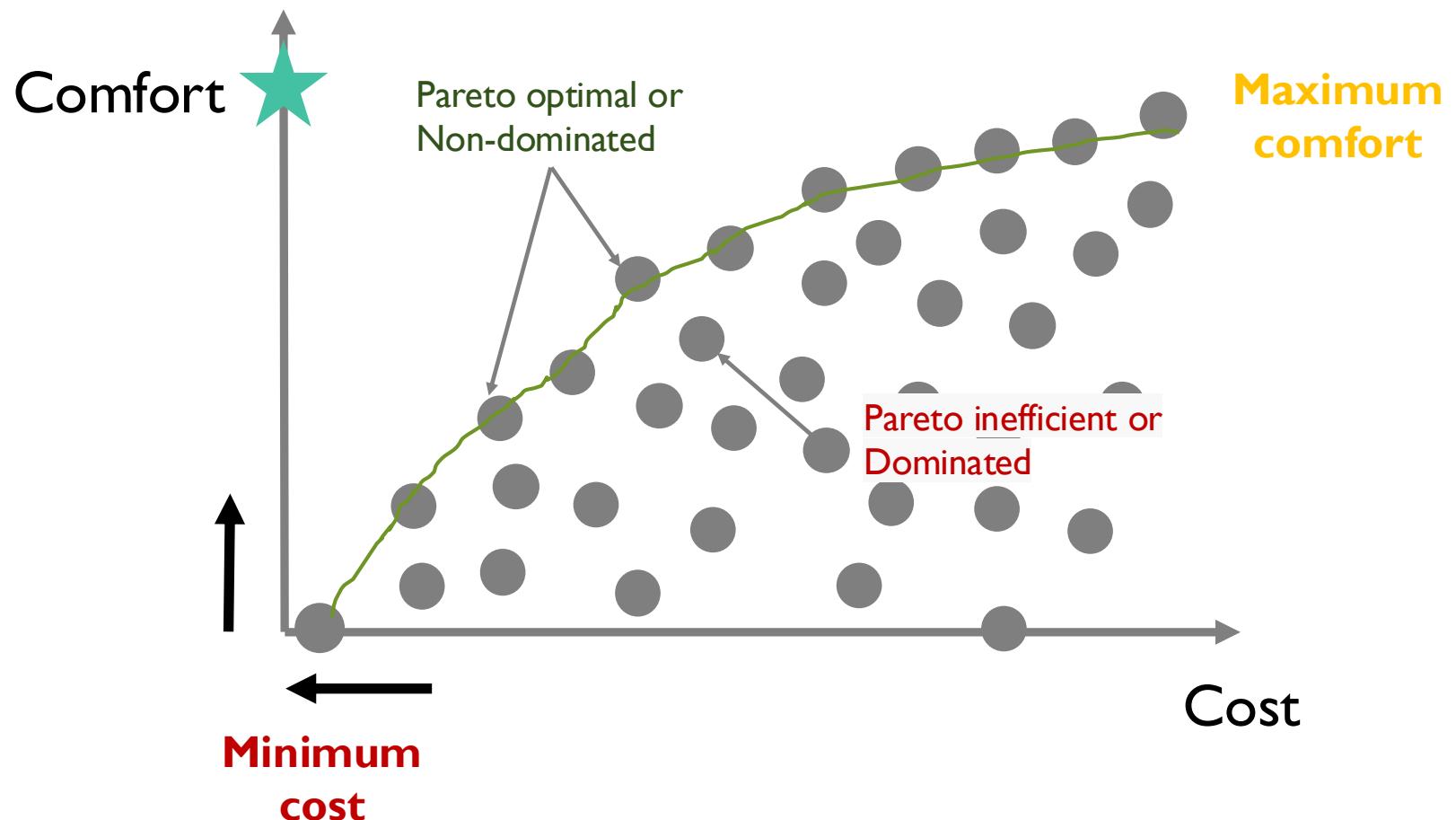
- **Premise:**

A solution is Pareto optimal when there exists no other solution that improves at least one objective without degrading the performance in any another objective

- **Purpose:**

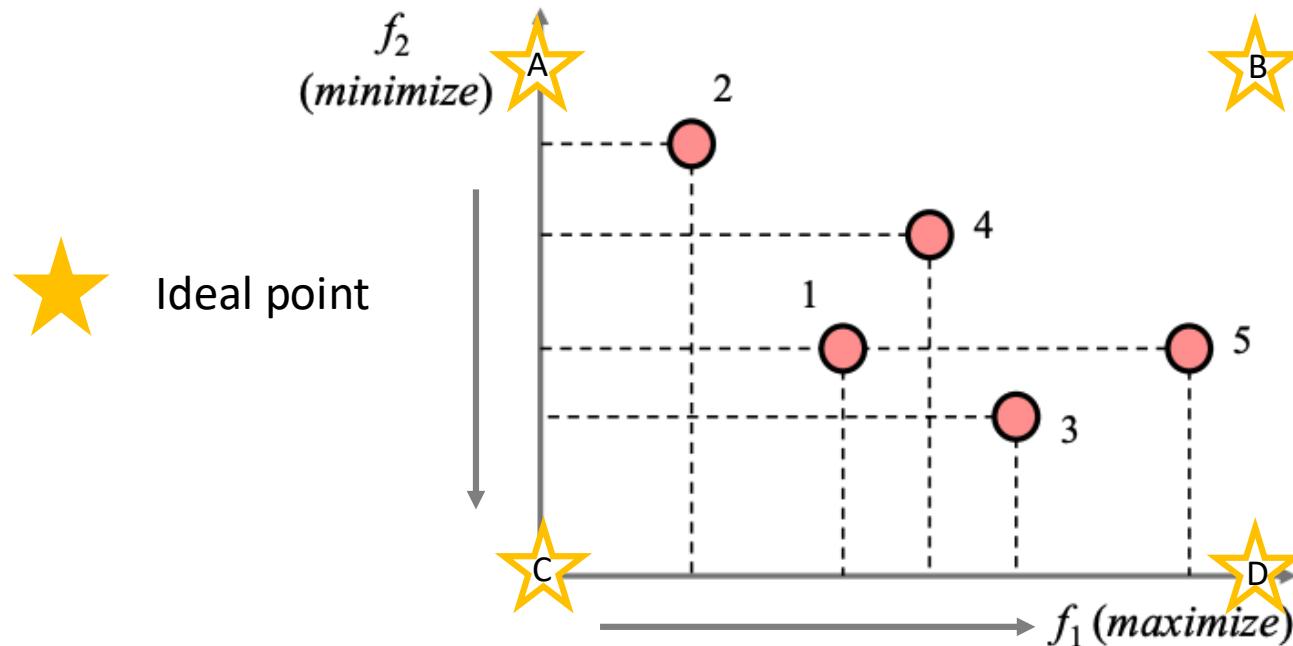
To rank alternatives in multi-objective problems

# Pareto Optimal Front



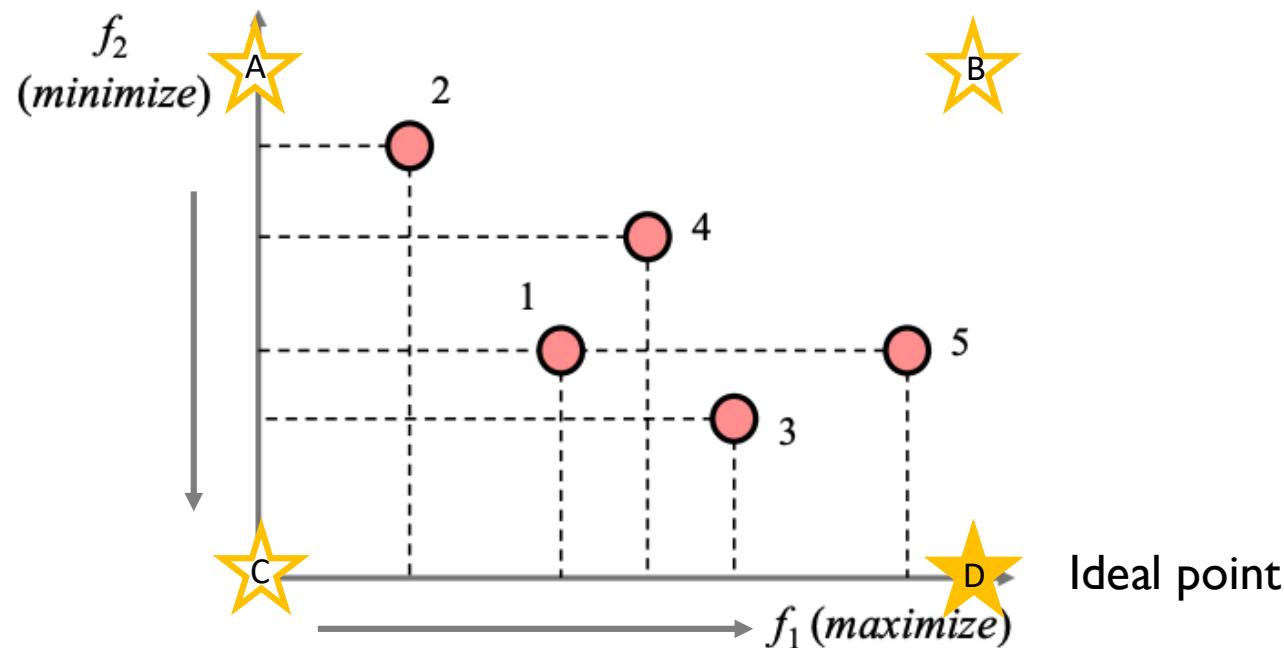
# Pareto Dominance

Which of the following stars would be closer to the **ideal point**?

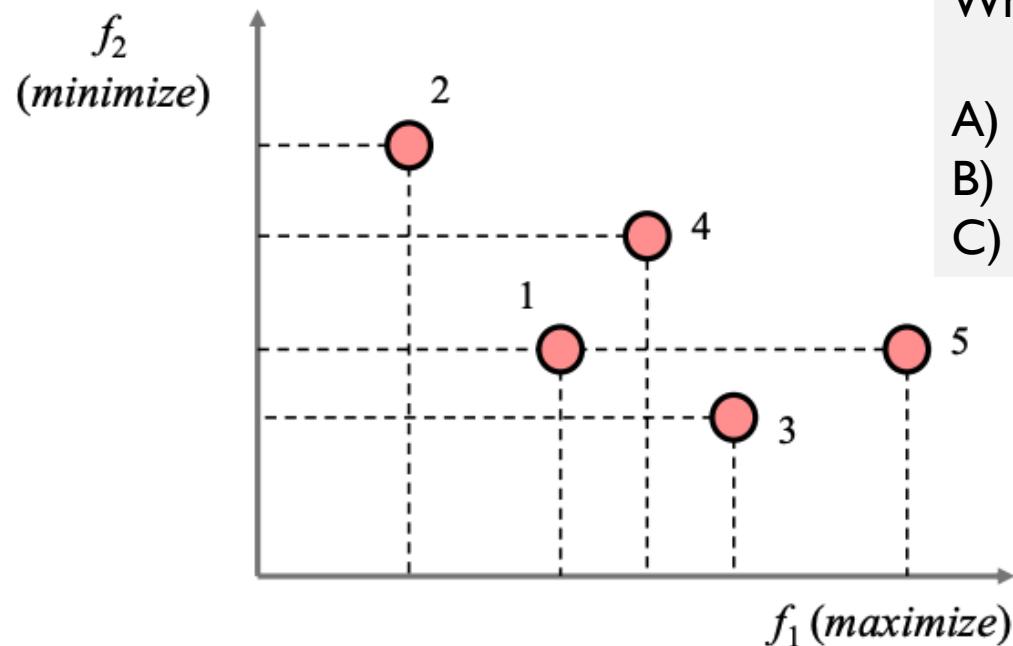


# Pareto Dominance

Which of the following stars would be closer to the **ideal point**?



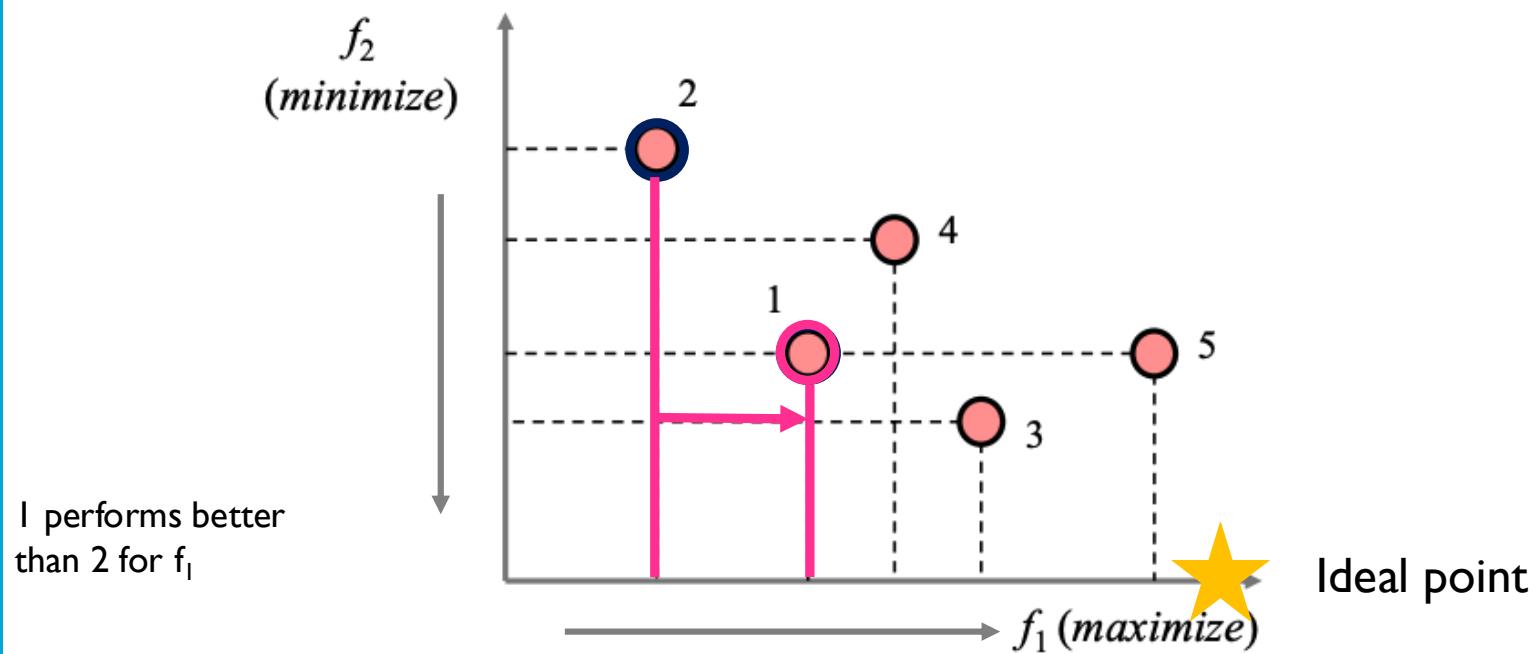
# Dominance Test: 1 vs. 2



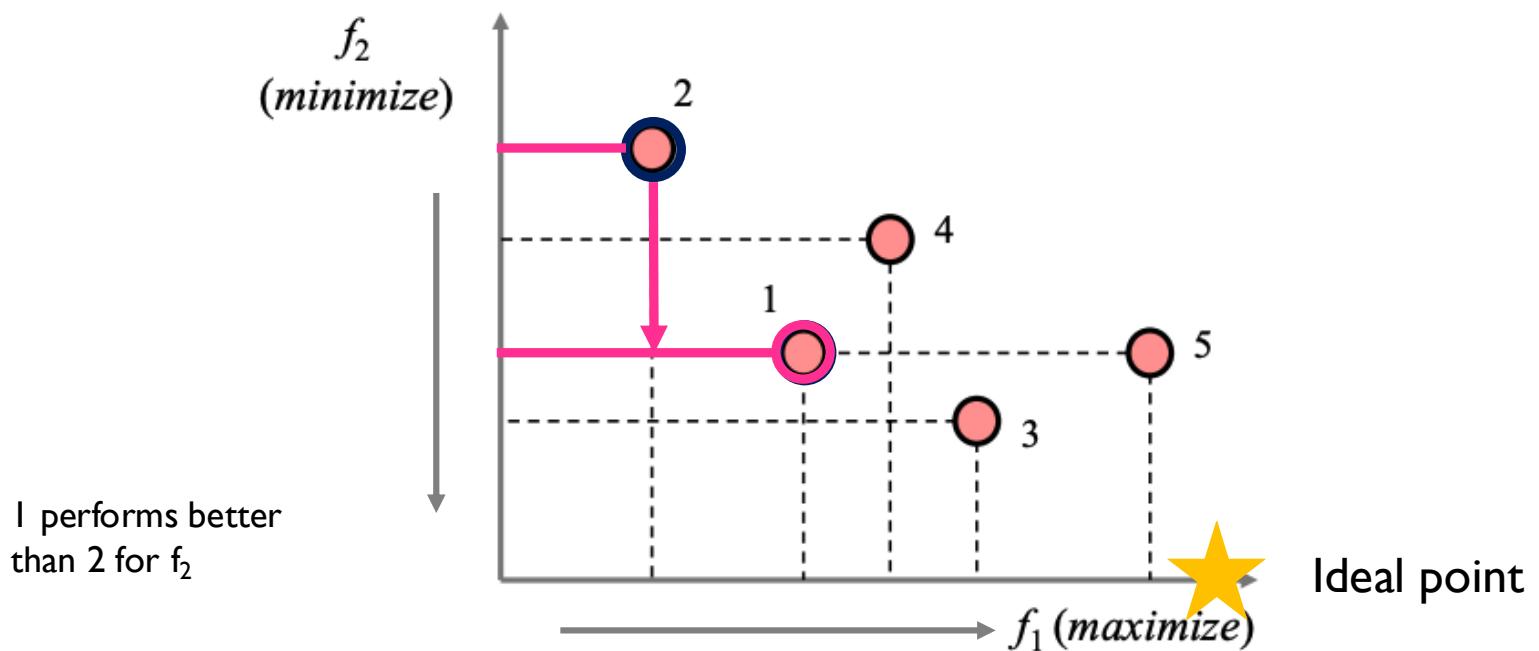
Which one dominates?

- A) 1 dominates 2
- B) 2 dominates 1
- C) Neither dominates

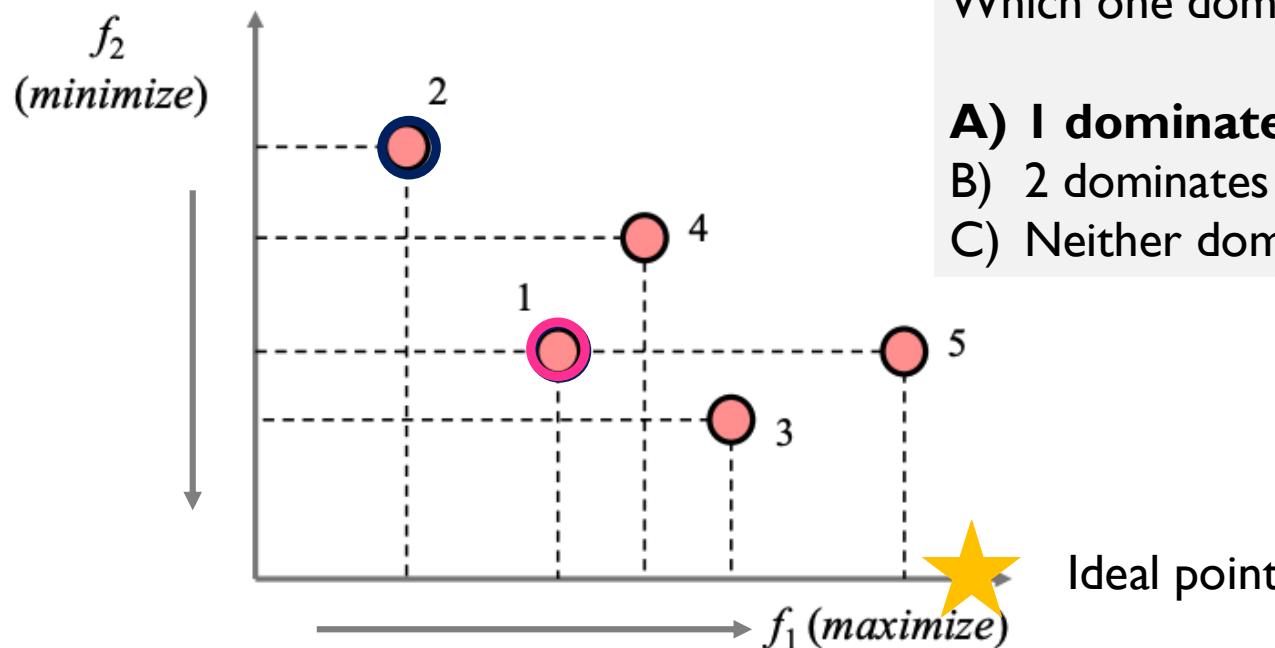
# Dominance Test: 1 vs. 2



# Dominance Test: 1 vs. 2

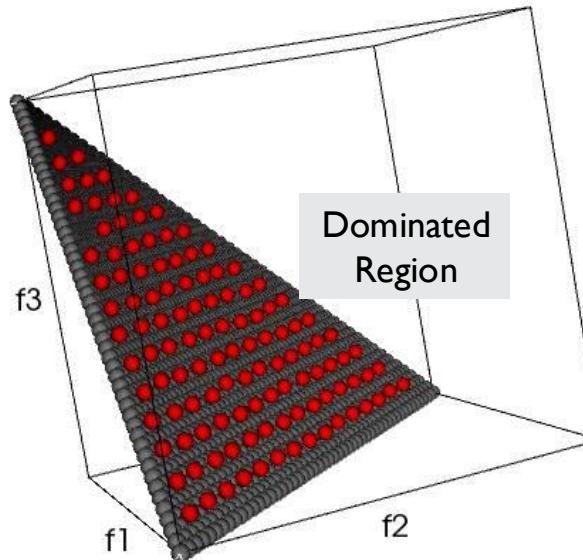


# Dominance Test: 1 vs. 2



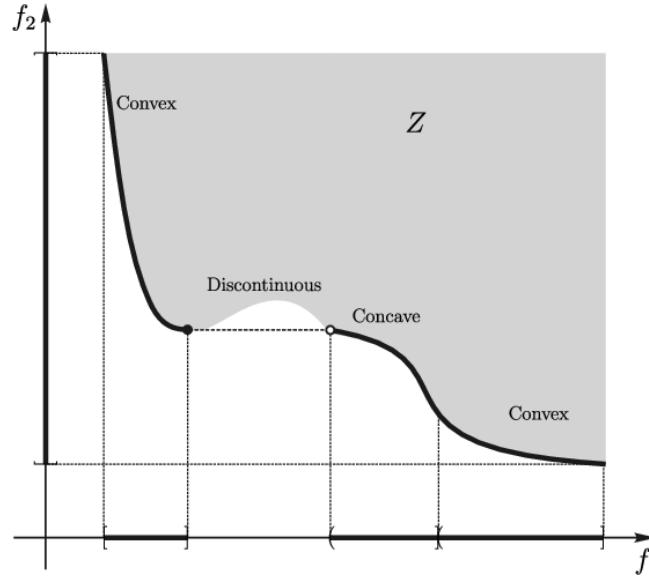
# Pareto Optimal Surface

- Pareto optimality applies to more than two objectives as well



Three-objective test problem example

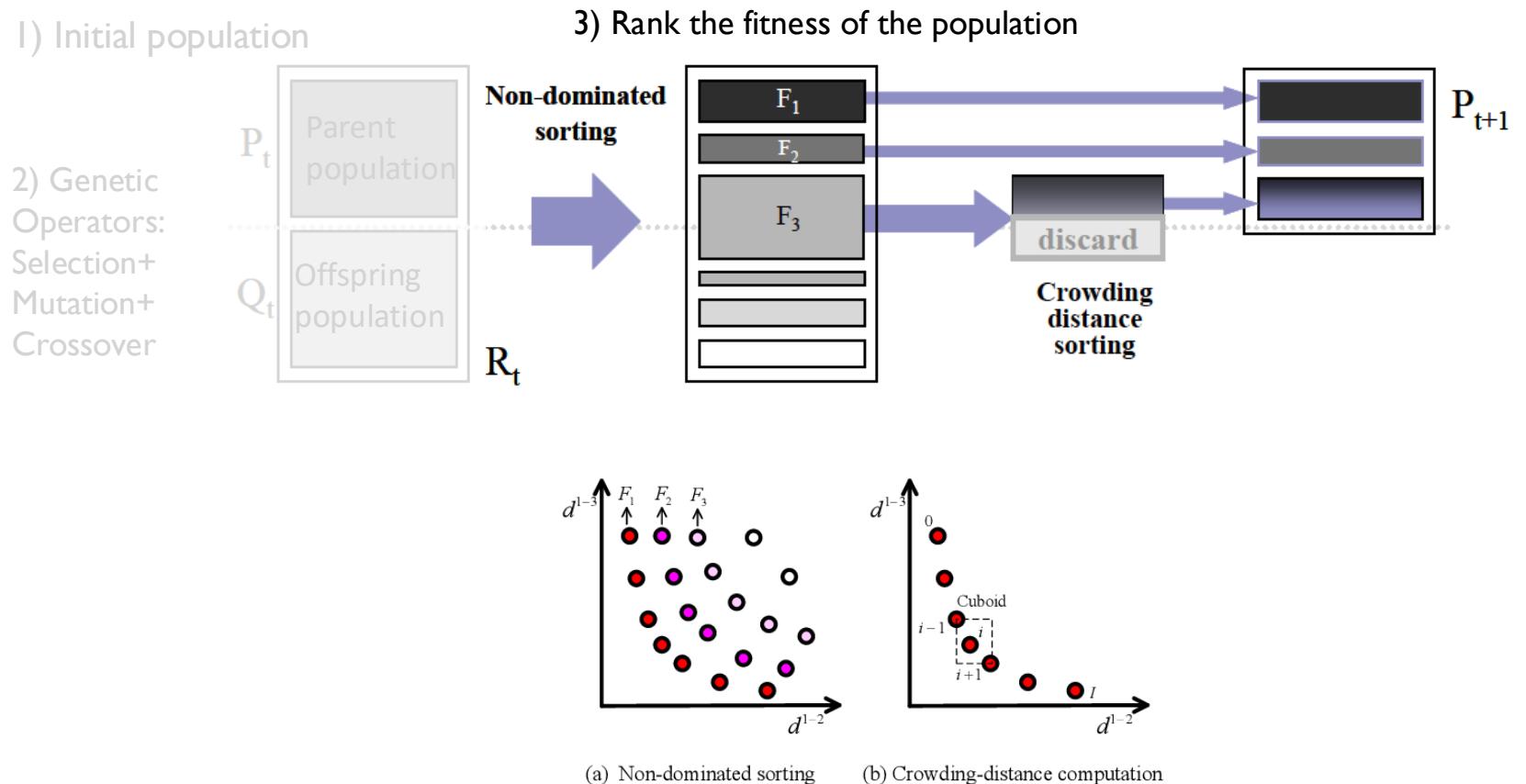
# Why Use Genetic Algorithm for Multi-Objective Optimization



- **Non-linear** objective functions
- Insensitive to **discontinuities** or **shape** of the Pareto front.
- **Flexible**, can find solutions to problems for which we have little knowledge
- The Pareto set is generated in a **single run**

# NSGA II

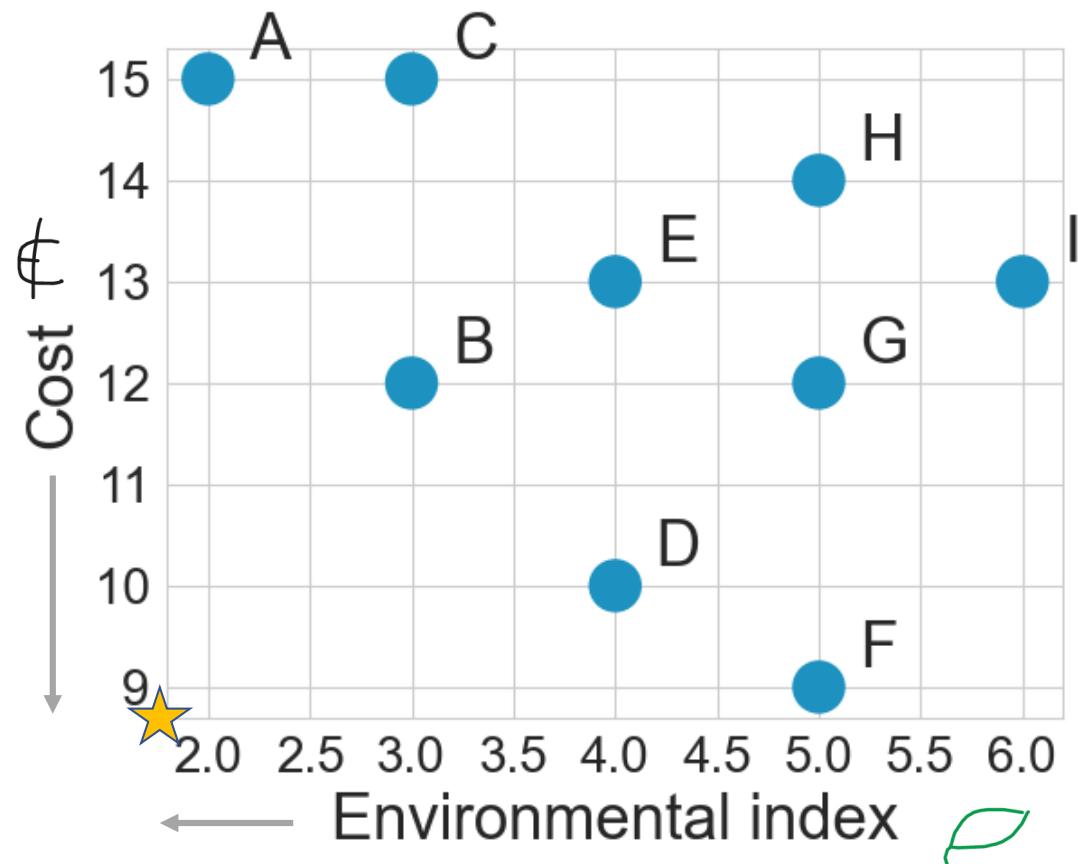
# NSGA II: Non-dominated Sorting Genetic Algorithm (Deb et al. 2000)



K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," in *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182-197, April 2002, doi: 10.1109/4235.996017.

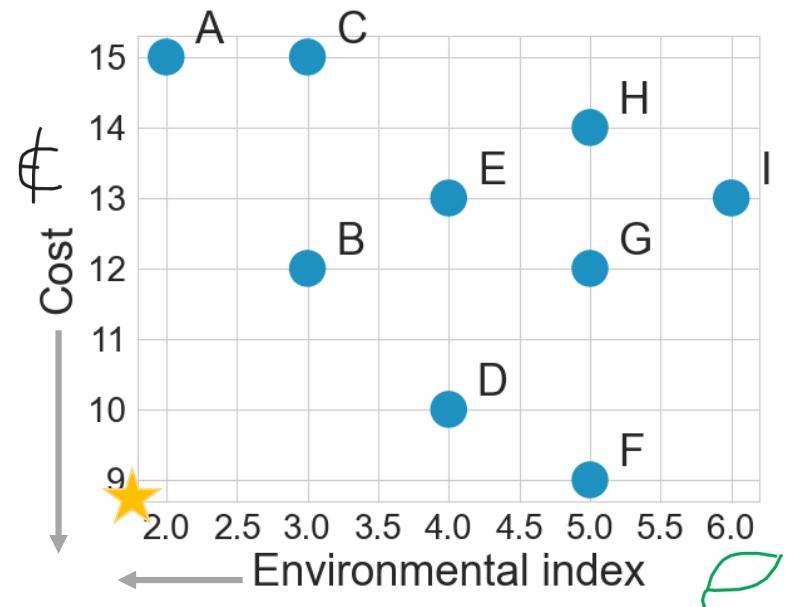
# NSGA II: Example

Car type	Environmental index (minimize)	Cost (minimize)
A	2	15K
B	3	12K
C	3	15K
D	4	10K
E	4	13K
F	5	9K
G	5	12K
H	5	14K
I	6	13K



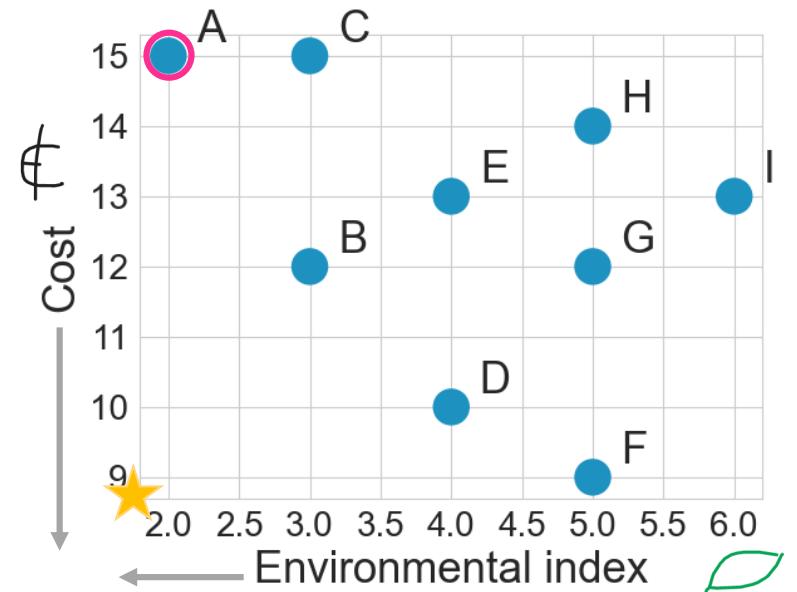
# NSGA II Example: Non-dominated Sorting

Car type	Env index (min)	Cost (min)	Points dominated	No. points that dominate
A	2	15K		
B	3	12K		
C	3	15K		
D	4	10K		
E	4	13K		
F	5	9K		
G	5	12K		
H	5	14K		
I	6	13K		



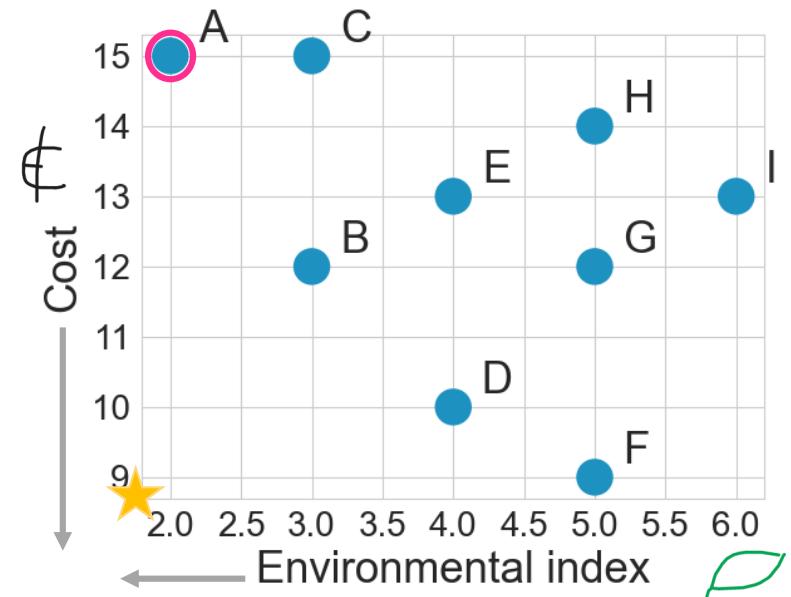
# NSGA II Example: Non-dominated Sorting

Car type	Env index (min)	Cost (min)	Points dominated	No. points that dominate
A	2	15K		
B	3	12K		
C	3	15K		
D	4	10K		
E	4	13K		
F	5	9K		
G	5	12K		
H	5	14K		
I	6	13K		



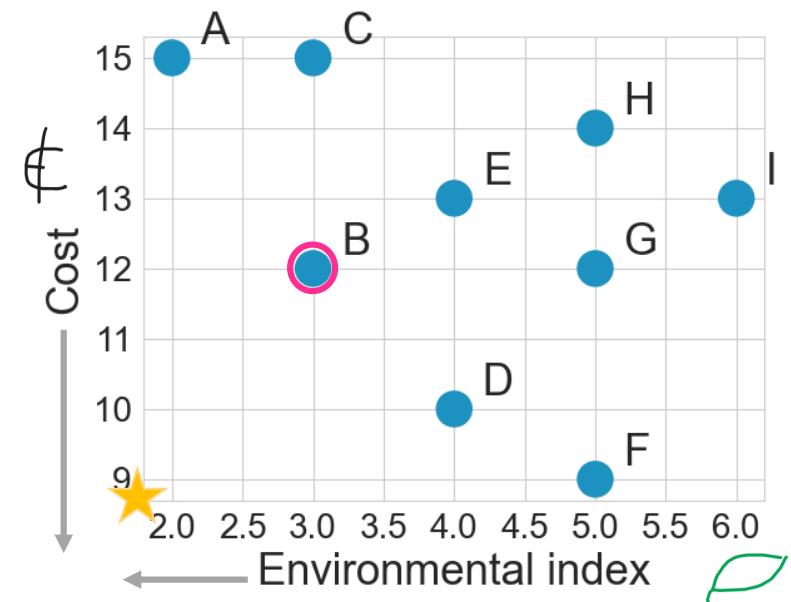
# NSGA II Example: Non-dominated Sorting

Car type	Env index (min)	Cost (min)	Points dominated	No. points that dominate
A	2	15K	C	0
B	3	12K		
C	3	15K		
D	4	10K		
E	4	13K		
F	5	9K		
G	5	12K		
H	5	14K		
I	6	13K		



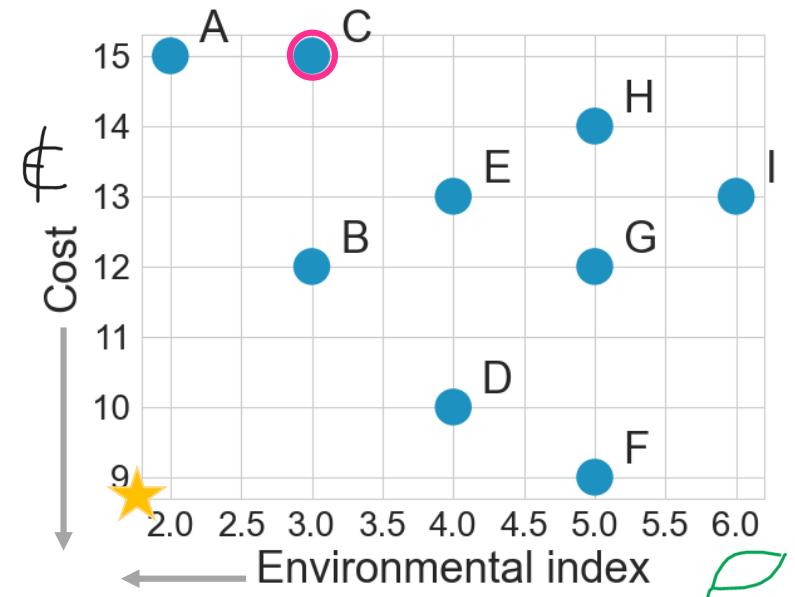
# NSGA II Example: Non-dominated Sorting

Car type	Env index (min)	Cost (min)	Points dominated	No. points that dominate
A	2	15K	C	0
B	3	12K	C,E,G,H,I	0
C	3	15K		
D	4	10K		
E	4	13K		
F	5	9K		
G	5	12K		
H	5	14K		
I	6	13K		



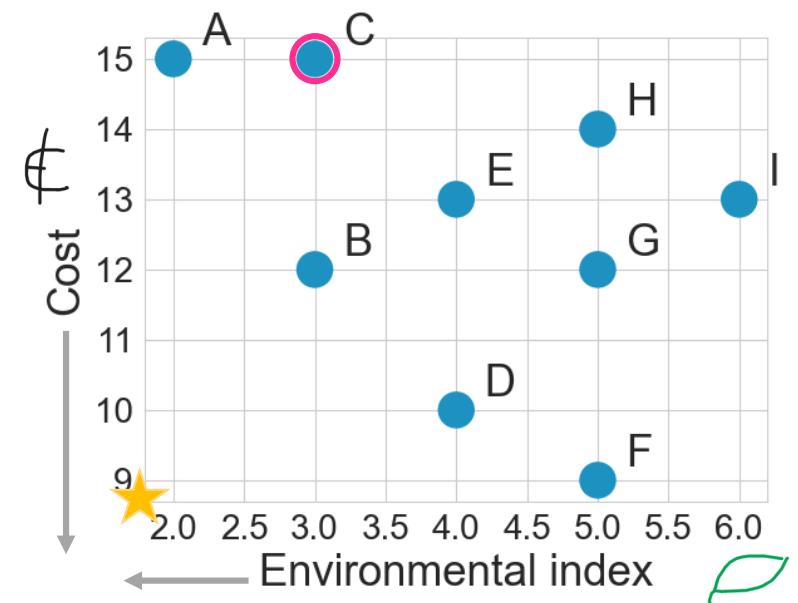
# NSGA II Example: Non-dominated Sorting

Car type	Env index (min)	Cost (min)	Points dominated	No. points that dominate
A	2	15K	C	0
B	3	12K	C,E,G,H,I	0
<b>C</b>	<b>3</b>	<b>15K</b>		
D	4	10K		
E	4	13K		
F	5	9K		
G	5	12K		
H	5	14K		
I	6	13K		



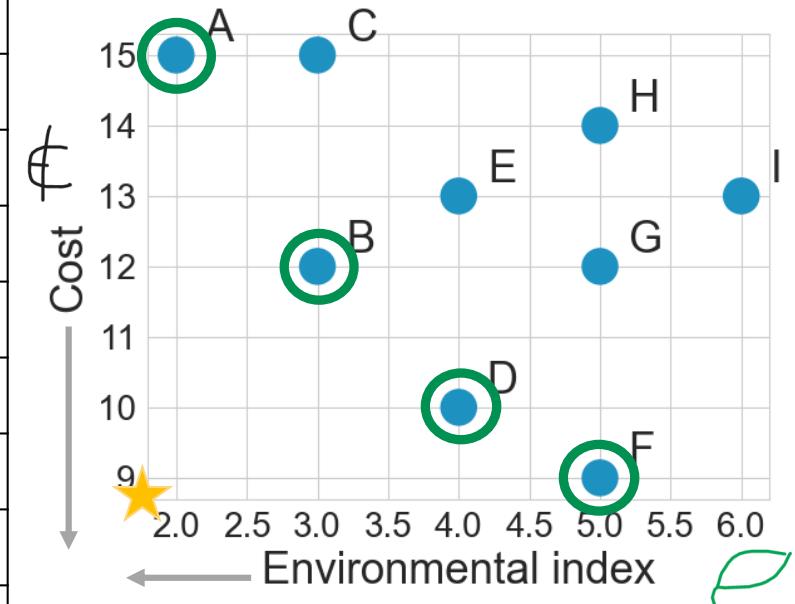
# NSGA II Example: Non-dominated Sorting

Car type	Env index (min)	Cost (min)	Points dominated	No. points that dominate
A	2	15K	C	0
B	3	12K	C,E,G,H,I	0
C	3	15K	-	2
D	4	10K		
E	4	13K		
F	5	9K		
G	5	12K		
H	5	14K		
I	6	13K		



# NSGA II Example: Pareto Fronts

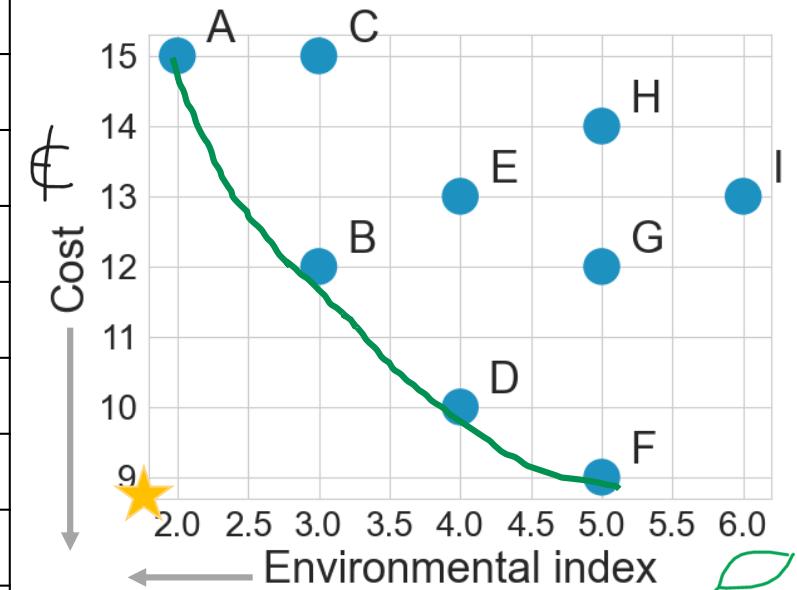
Car type	Env index (min)	Cost (min)	Points dominated	No. points that dominate	Pareto rank
A	2	15K	C	0	
B	3	12K	C,E,G,H,I	0	
C	3	15K	-	2	
D	4	10K	E,G,H,I	0	
E	4	13K	H,I	2	
F	5	9K	G,H,I	0	
G	5	12K	H,I	3	
H	5	14K	-	5	
I	6	13K	-	5	



# NSGA II Example: Pareto Fronts

Car type	Env index (min)	Cost (min)	Points dominated	No. points that dominate	Pareto rank
A	2	15K	C	0	I
B	3	12K	C,E,G,H,I	0	I
C	3	15K	-	2	
D	4	10K	E,G,H,I	0	I
E	4	13K	H,I	2	
F	5	9K	G,H,I	0	I
G	5	12K	H,I	3	
H	5	14K	-	5	
I	6	13K	-	5	

Non-dominated front 1

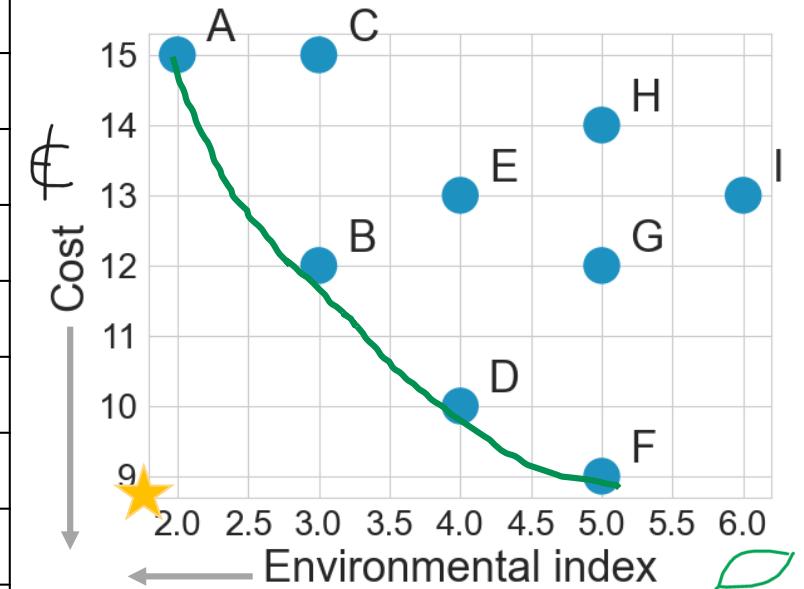


# NSGA II Example: Pareto Fronts

Update “No. points that dominate” by taking out the identified non-dominated front

Car type	Env index (min)	Cost (min)	Points dominated	No. points that dominate	Pareto rank
A	2	15K	C	0	I
B	3	12K	C,E,G,H,I	0	I
C	3	15K	-	2-I-I	
D	4	10K	E,G,H,I	0	I
E	4	13K	H,I	2-I-I	
F	5	9K	G,H,I	0	I
G	5	12K	H,I	3-I-I-I	
H	5	14K	-	5-I-I-I	
I	6	13K	-	5-I-I-I	

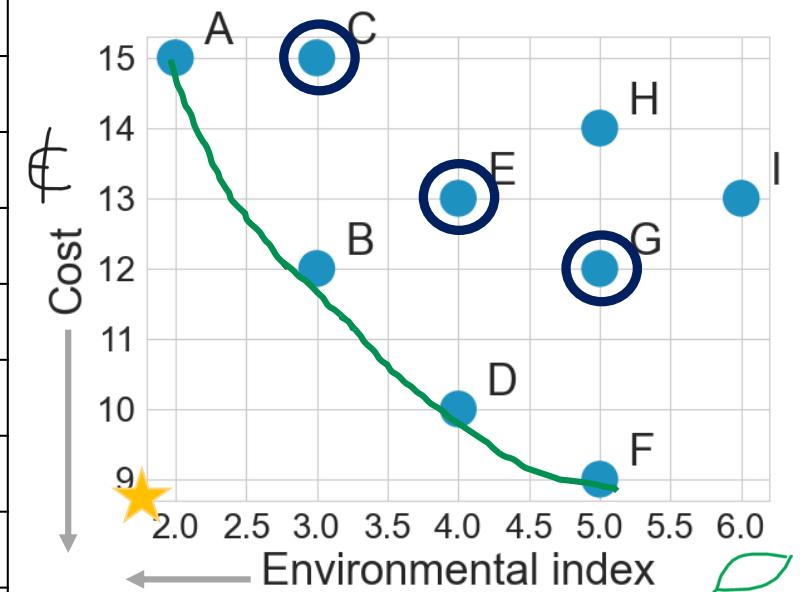
Non-dominated front 1



# NSGA II Example: Pareto Fronts

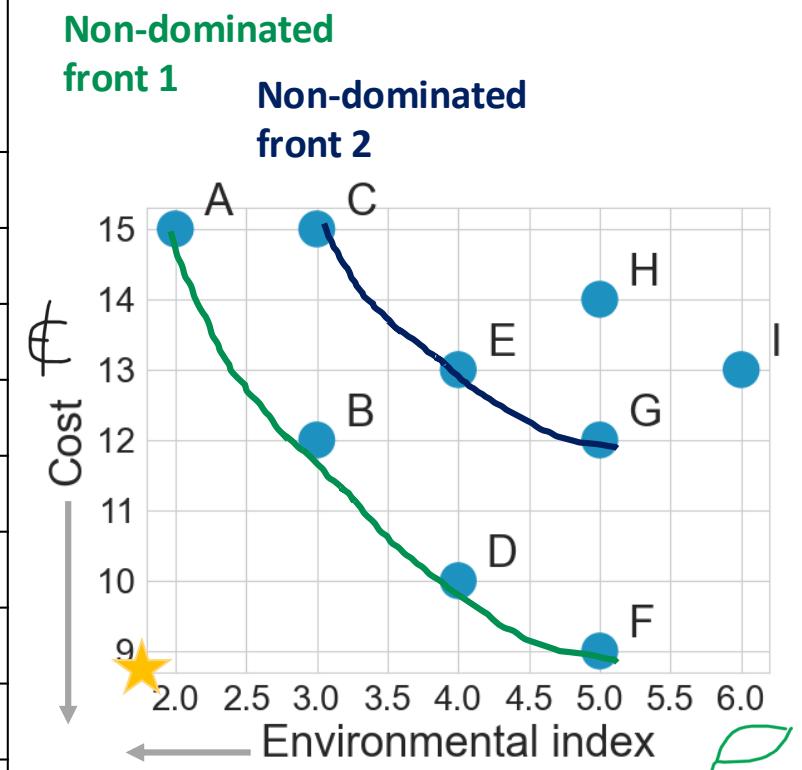
Car type	Env index (min)	Cost (min)	Points dominated	No. points that dominate	Pareto rank
A	2	15K	C	0	I
B	3	12K	C,E,G,H,I	0	I
C	3	15K	-	0	
D	4	10K	E,G,H,I	0	I
E	4	13K	H,I	0	
F	5	9K	G,H,I	0	I
G	5	12K	H,I	0	
H	5	14K	-	2	
I	6	13K	-	2	

Non-dominated front 1



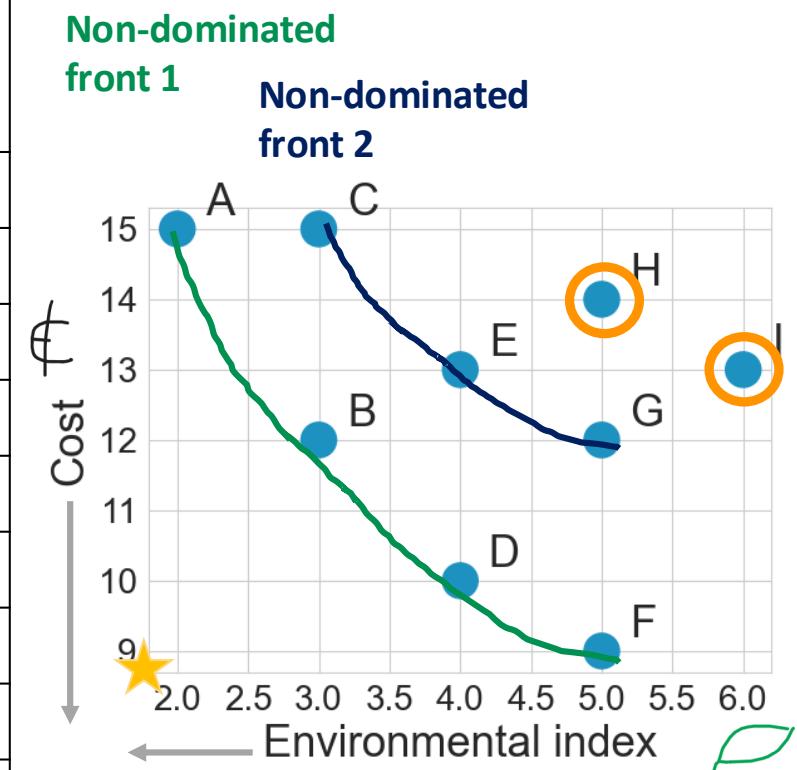
# NSGA II Example: Pareto Fronts

Car type	Env index (min)	Cost (min)	Points dominated	No. points that dominate	Pareto rank
A	2	15K	C	0	1
B	3	12K	C,E,G,H,I	0	1
C	3	15K	-	0	2
D	4	10K	E,G,H,I	0	1
E	4	13K	H,I	0	2
F	5	9K	G,H,I	0	1
G	5	12K	H,I	0	2
H	5	14K	-	2-1-1	
I	6	13K	-	2-1-1	



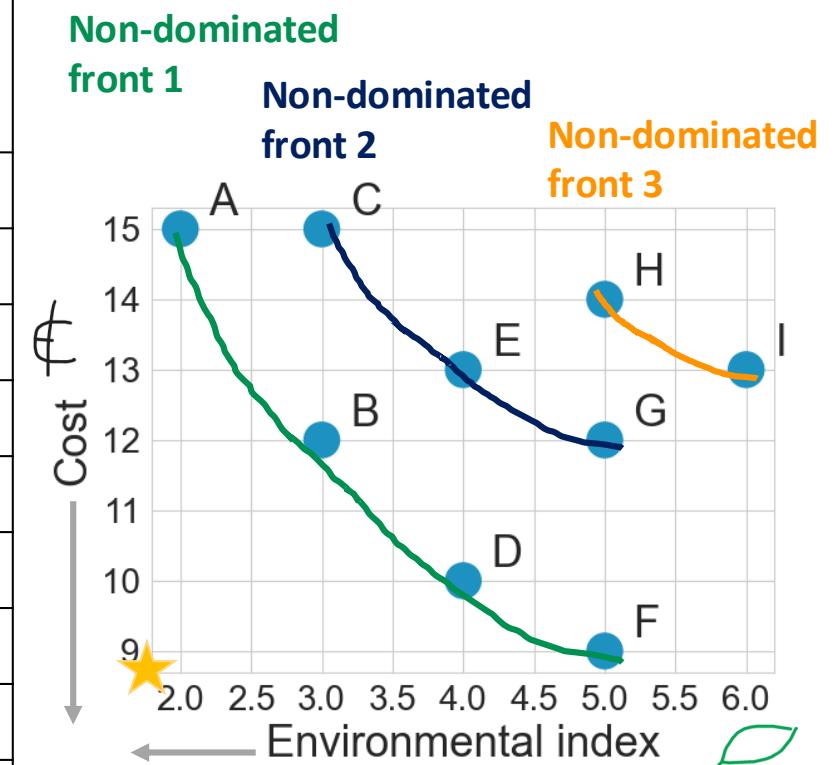
# NSGA II Example: Pareto Fronts

Car type	Env index (min)	Cost (min)	Points dominated	No. points that dominate	Pareto rank
A	2	15K	C	0	1
B	3	12K	C,E,G,H,I	0	1
C	3	15K	-	0	2
D	4	10K	E,G,H,I	0	1
E	4	13K	H,I	0	2
F	5	9K	G,H,I	0	1
G	5	12K	H,I	0	2
H	5	14K	-	0	
I	6	13K	-	0	

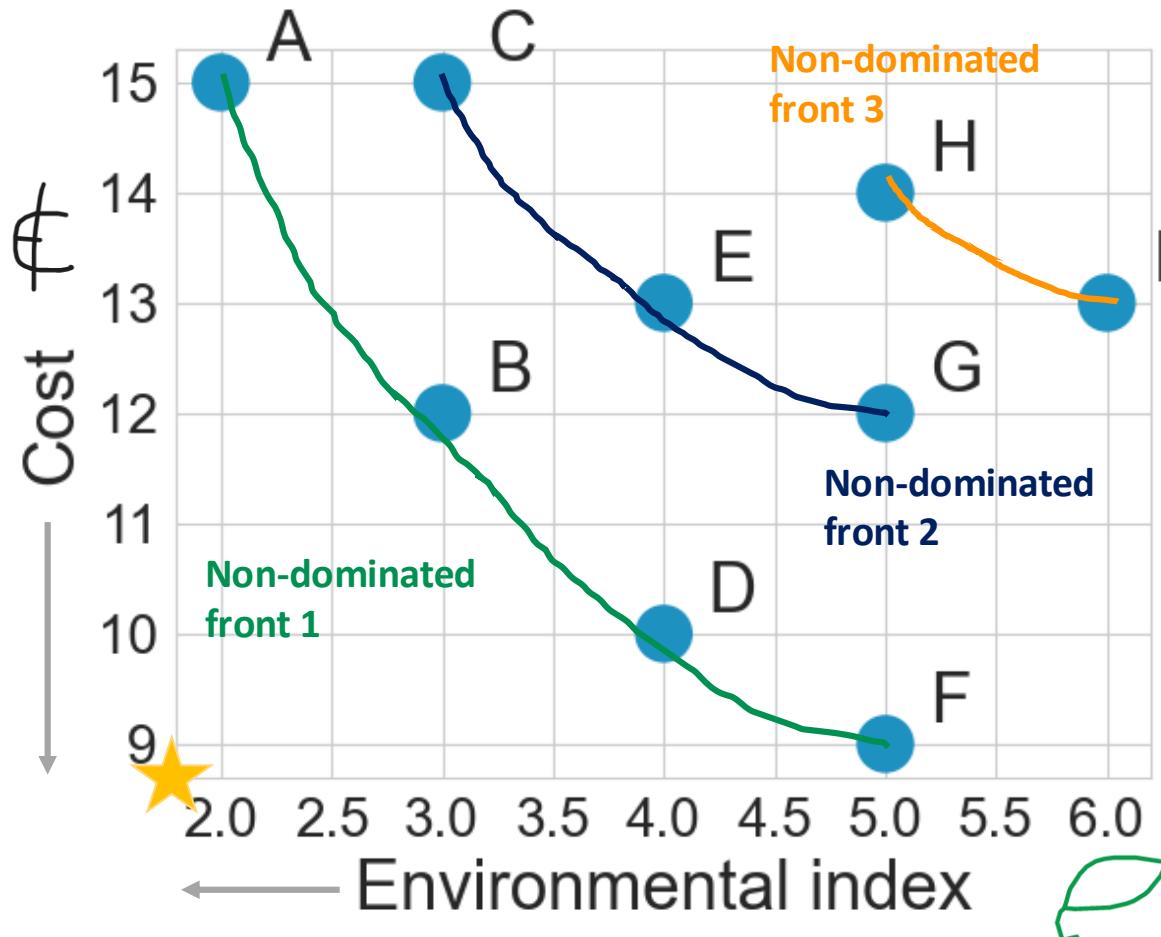


# NSGA II Example: Pareto Fronts

Car type	Env index (min)	Cost (min)	Points dominated	No. points that dominate	Pareto rank
A	2	15K	C	0	1
B	3	12K	C,E,G,H,I	0	1
C	3	15K	-	0	2
D	4	10K	E,G,H,I	0	1
E	4	13K	H,I	0	2
F	5	9K	G,H,I	0	1
G	5	12K	H,I	0	2
H	5	14K	-	0	3
I	6	13K	-	0	3



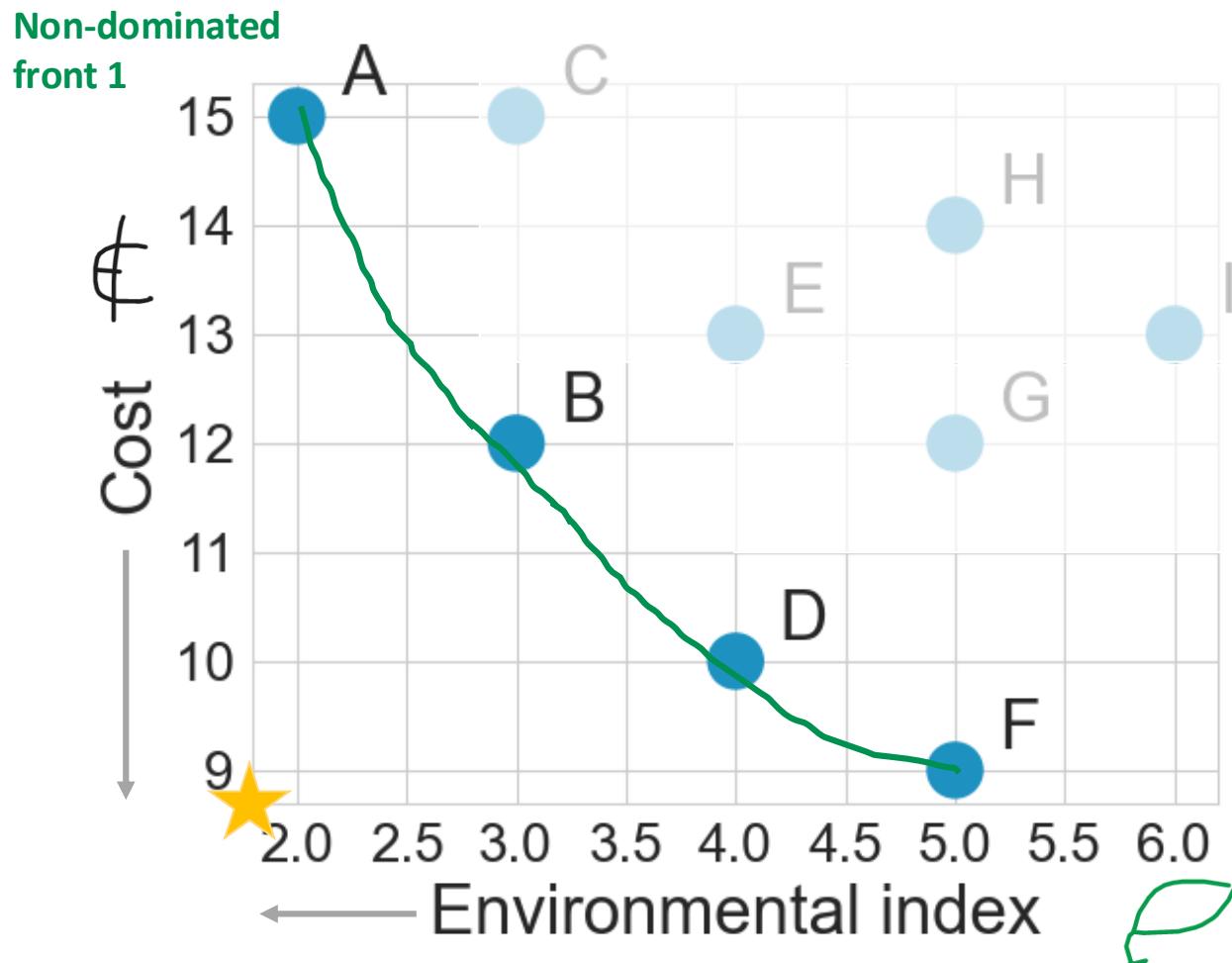
# NSGA II Example: Pareto Fronts



Fitness assignment:

- Non-dominated front 1 has the highest fitness
- Same front has the same fitness
- Lower non-dominated rank is preferred over others

# NSGA II Example: Preserving Diversity

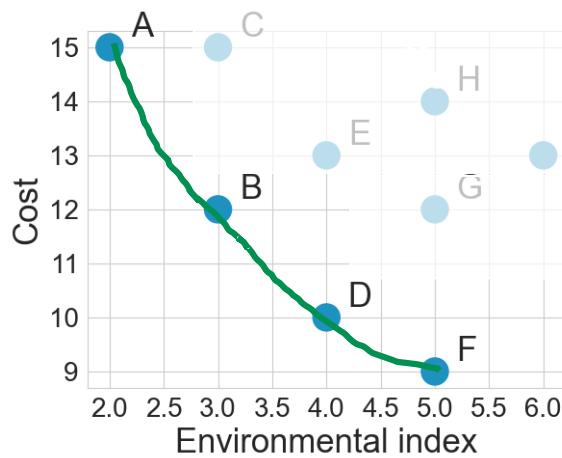


If we could only  
keep 3 solutions  
which ones  
should we keep?

# NSGA II Example: Crowding Distance (CD)

$$CD_{im} = \frac{f_m(x_{i+1}) - f_m(x_{i-1})}{f_m(x_{max}) - f_m(x_{min})}$$

$$CD_i = \sum_{m=1}^M CD_{im}$$



Car type	Env index (min)	Cost (min)	CD
A	2	15K	inf
B	3	12K	1.33
D	4	10K	1
F	5	9K	inf

$$CD_B = \frac{4 - 2}{4} + \frac{15 - 10}{6} \approx 1.33$$

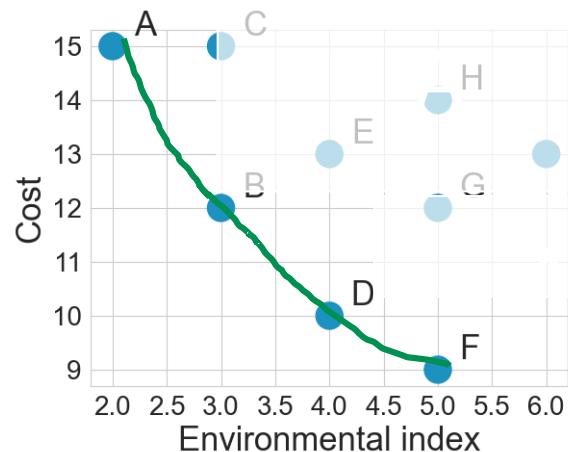
$$CD_D = \frac{5 - 3}{4} + \frac{12 - 9}{6} = 1$$

Car type	Env index (min)	Cost (min)
A	2	15K
B	3	12K
C	3	15K
D	4	10K
E	4	13K
F	5	9K
G	5	12K
H	5	14K
I	6	13K

max	6	15
min	2	9
diff	4	6

# NSGA II Example: Preserving Diversity

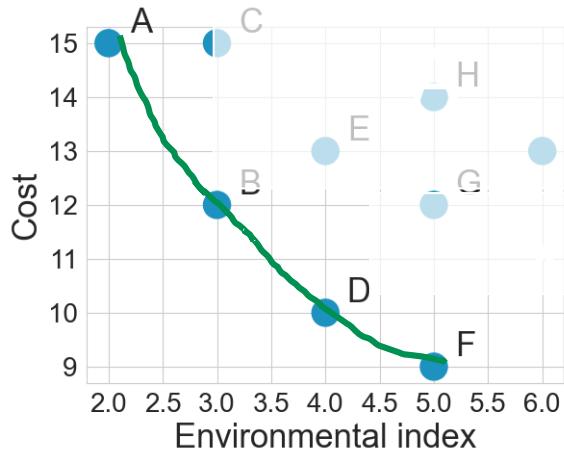
If you could only keep 3 solutions which point would you eliminate?



Options:	Car type	Env index (min)	Cost (min)	CD
A)	A	2	15K	inf
B)	B	3	12K	1.33
C)	D	4	10K	1
D)	F	5	9K	inf

# NSGA II Example: Preserving Diversity

If you could only keep 3 solutions which point would you eliminate?



Options:	Car type	Env index (min)	Cost (min)	CD
A)	A	2	15K	inf
B)	B	3	12K	1.33
<b>C)</b>	<b>D</b>	<b>4</b>	<b>10K</b>	<b>I</b>
D)	F	5	9K	inf

A larger crowding distance is preferred to preserve diversity

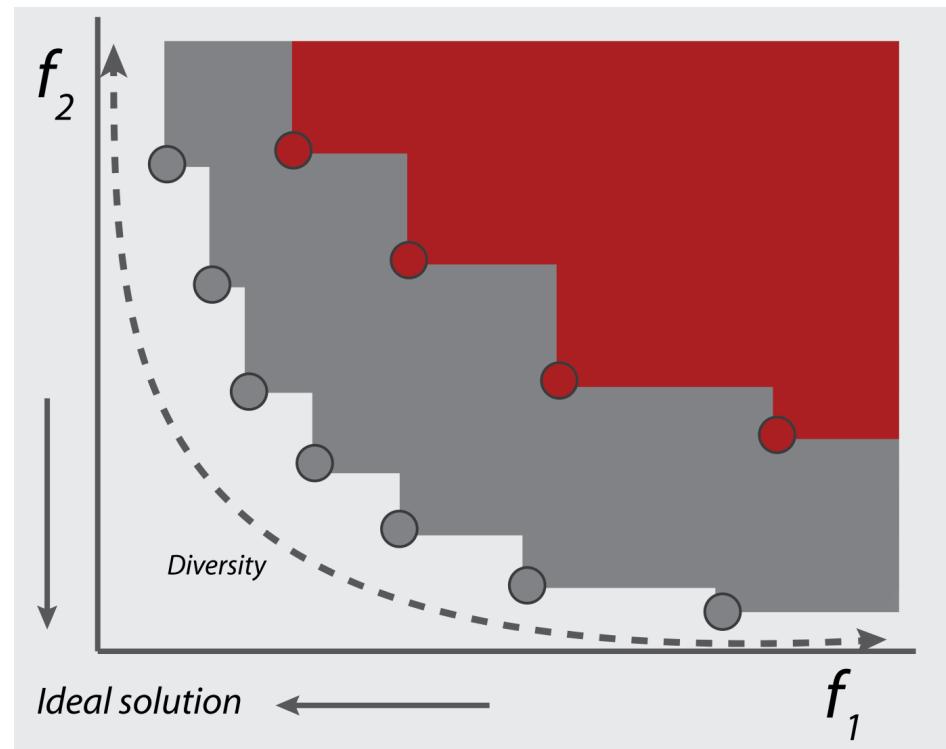
# NSGA II: Fitness Strategies

Selection pressure mechanism:

**Non-dominated sorting**

Diversity preservation:

**Crowding distance**



# Multi-Objective Optimization: Wrap Up

- ✓ Multiobjective optimization problem consists of a problem with 2 or more objectives.
- ✓ Concept of Pareto optimality and non-dominance, allows to rank solutions in a multiobjective optimization problem
- ✓ Goals of multi-objective evolutionary optimization:
  - ✓ Preserve the best individuals (non-dominated sorting)
  - ✓ Preserve diversity (crowding distance)

# Summary

- Steps of Genetic Algorithm
- Applications of Genetic Algorithm
- Multi-Objective Evolutionary Optimization
- NSGA II Algorithm
- **Optional Reading from Chapter 7:** Evolutionary Strategies and Genetic Programming

# Artificial Neural Networks

## The Multilayer Perceptron

CSE2530 – Julia Olkhovskaya



# Why do we need Artificial Neural Networks?

# We need something more

Say that we want an algorithm capable of distinguishing dogs from wolves in pictures.

We can manually specify what distinguishes a dog from a wolf (e.g., fur and ears).

This is called feature engineering and **good old-fashioned AI (GOFAI)**.

But it is very difficult and prone to mistakes.

Alternatively, we can show the algorithm photos of dogs and photos of wolves and ask it to figure out the difference on its own (**supervised machine learning**).

Though, the **input dimension** of even a small RGB photo (256x256 pixels) is extremely large (256x256x3=196608).

Traditional ML algorithms (e.g., Logistic Regression or SVM) can handle low-dimensional problems with small amounts of data. However, to **scale-up** to larger input dimensions and scale of data, we need something that can be:

- Stacked in a scalable way;
- Trained efficiently.



# Content Overview

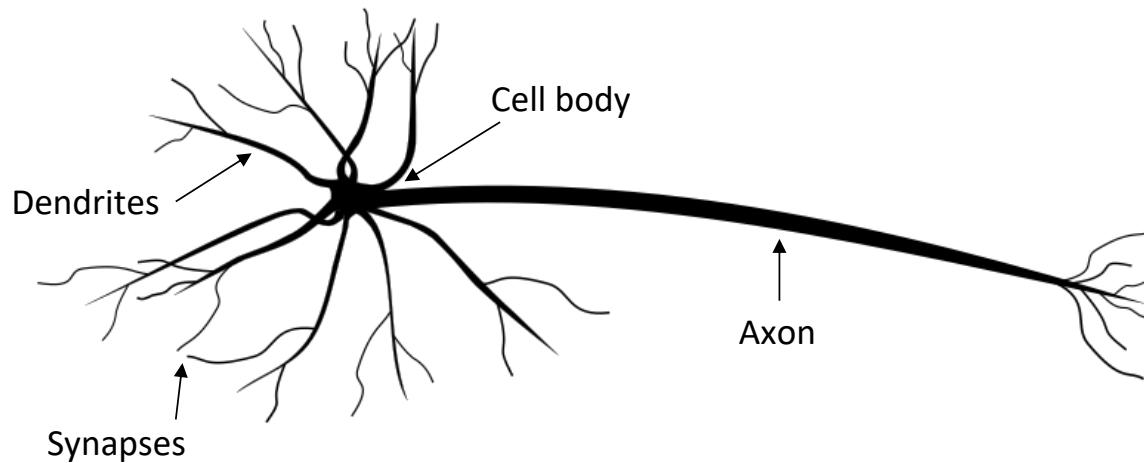
- How the brain **neuron** inspired the first artificial neuron (the **perceptron**);
- How to **train** the perceptron;
- **Limitations** of the perceptron;
- The **multilayer perceptron**;
- **Feedforward** inference;
- Training the multilayer perceptron through **backpropagation**.

# Material

- CSE2510 Machine Learning course – week 6.2 (this lesson is a recap);
- Michael Negnevitsky – Artificial Intelligence: sections 6.1 to 6.4 (recommended);
- Michael Nielsen - Neural Networks and Deep Learning (further reading):  
<http://neuralnetworksanddeeplearning.com/>
- Grant Sanderson - (3blue1brown) (further reading):  
Intro to Artificial Neural Networks: <https://youtu.be/aircAruvnKk>  
Gradient Descent: <https://youtu.be/IHZwWFHWa-w>

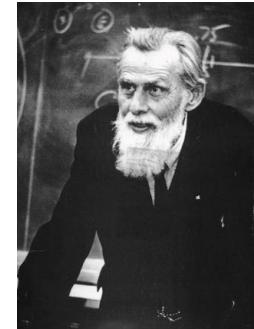
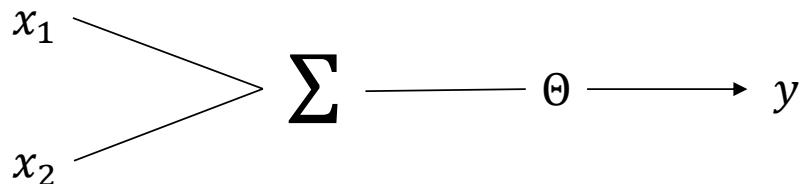
# Neural Networks

Our brain is composed of a network of **neurons**.



# Artificial Neuron

In 1943, McCulloch and Pitts, inspired by the neuron, devised the first mathematical model of the artificial neuron:



Warren S. McCulloch

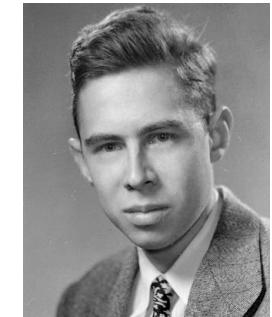
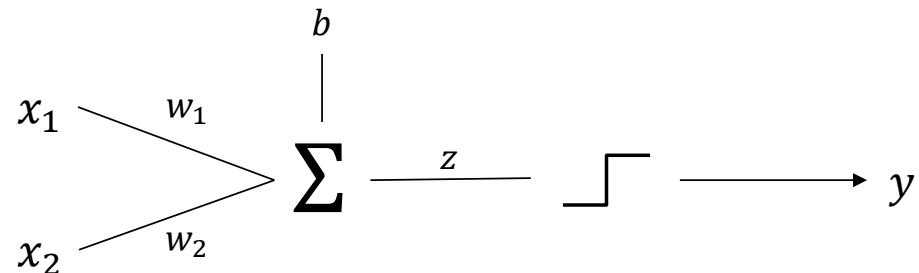
$$y = \begin{cases} 1, & \text{if } \sum_{k=1}^n x_k > \Theta \\ 0, & \text{otherwise} \end{cases}$$



Walter H. Pitts Jr.

# Perceptron

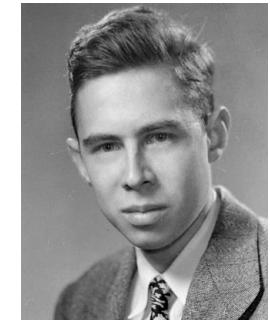
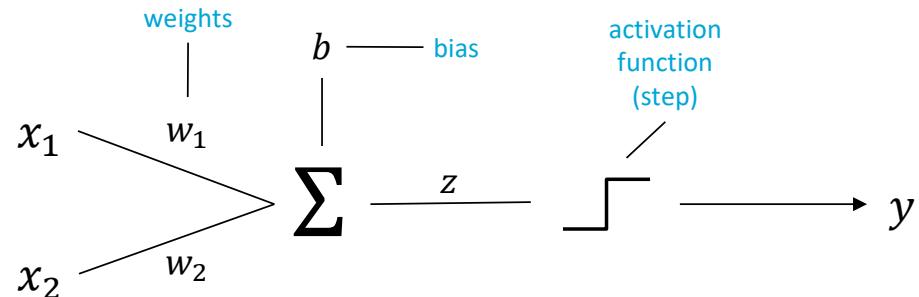
The neuron from McCulloch&Pitts had to be hard-coded and could not learn complex functions. In 1957, Rosenblatt proposes the modern version of the artificial neuron, named **perceptron**.



Frank Rosenblatt

# Perceptron

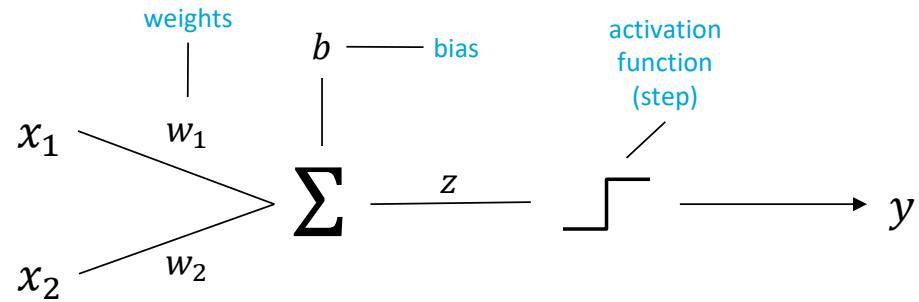
The neuron from McCulloch&Pitts had to be hard-coded and could not learn complex functions. In 1957, Rosenblatt proposes the modern version of the artificial neuron, named **perceptron**.



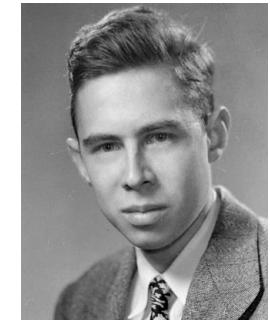
Frank Rosenblatt

# Perceptron

The neuron from McCulloch&Pitts had to be hard-coded and could not learn complex functions. In 1957, Rosenblatt proposes the modern version of the artificial neuron, named **perceptron**.



$$z = w^T x + b$$
$$\text{step}(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{otherwise} \end{cases}$$



Frank Rosenblatt

$$y = \begin{cases} 1, & \text{if } w^T x + b \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

# Training the Perceptron

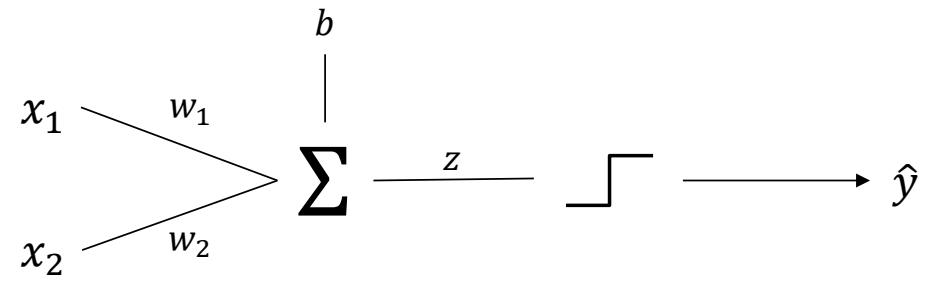
We want a perceptron to be able to predict whether a property is a house or an apartment ( $y$ ), based on the size in  $\text{m}^2$  ( $x_1$ ) and the number of rooms ( $x_2$ ).

We have some examples available:

Size ( $x_1$ )	Rooms ( $x_2$ )	Type ( $y$ )
100	3	0
120	3	1
65	2	0

$y = 0$  apartment

$y = 1$  house



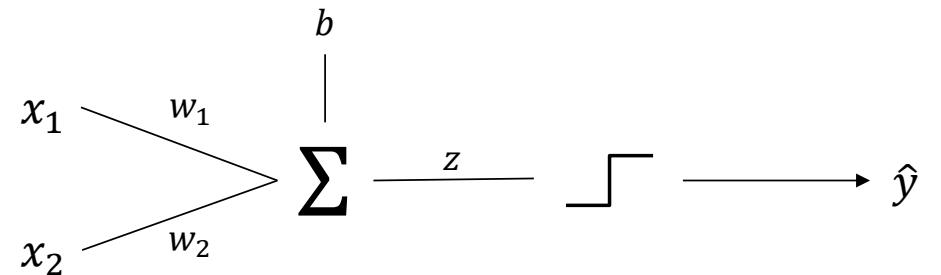
$$\text{step}(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

# Training the Perceptron

We want a perceptron to be able to predict whether a property is a house or an apartment ( $y$ ), based on the size in  $\text{m}^2$  ( $x_1$ ) and the number of rooms ( $x_2$ ).

We have some examples available:

Size ( $x_1$ )	Rooms ( $x_2$ )	Type ( $y$ )	
100	3	0	$y = 0$ apartment
120	3	1	$y = 1$ house
65	2	0	



$$\text{step}(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

Step 0: Initialize weights and learning rate ( $\alpha$ )

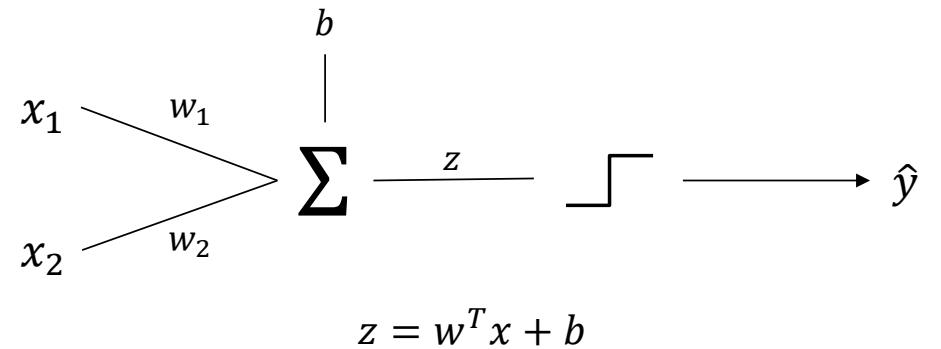
$$w_1 = 0.5 \quad w_2 = 0.5 \quad b = -50 \quad \alpha = 0.1$$

# Training the Perceptron

Step 1: Perform inference

$$z = w_1 x_1 + w_2 x_2 + b = 50 + 1.5 - 50 = 1.5$$

$$\hat{y} = \text{step}(z) = 1$$



$$\text{step}(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

Size ( $x_1$ )	Rooms ( $x_2$ )	Type ( $y$ )	
100	3	0	$y = 0$ apartment
120	3	1	$y = 1$ house
65	2	0	

# Training the Perceptron

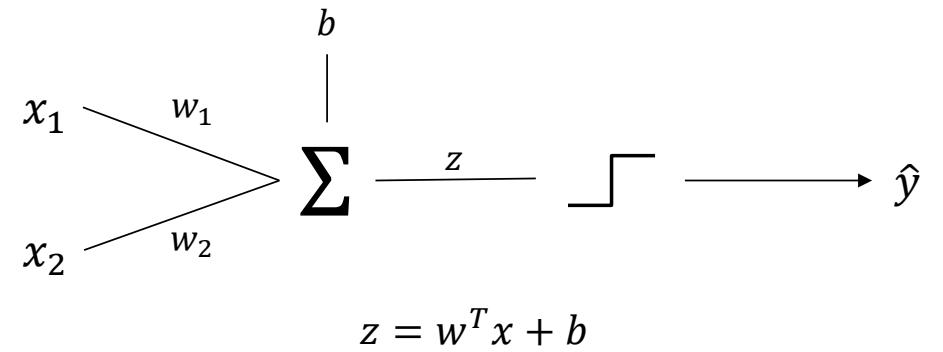
Step 1: Perform inference

$$z = w_1 x_1 + w_2 x_2 + b = 50 + 1.5 - 50 = 1.5$$

$$\hat{y} = \text{step}(z) = 1$$

Step 2: Compute loss

$$\mathcal{L} = y - \hat{y} = 0 - 1 = -1$$



$$\text{step}(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

Size ( $x_1$ )	Rooms ( $x_2$ )	Type ( $y$ )	
100	3	0	$y = 0$ apartment
120	3	1	$y = 1$ house
65	2	0	

# Training the Perceptron

## Step 1: Perform inference

$$z = w_1 x_1 + w_2 x_2 + b = 50 + 1.5 - 50 = 1.5$$

$$\hat{y} = \text{step}(z) = 1$$

## Step 2: Compute loss

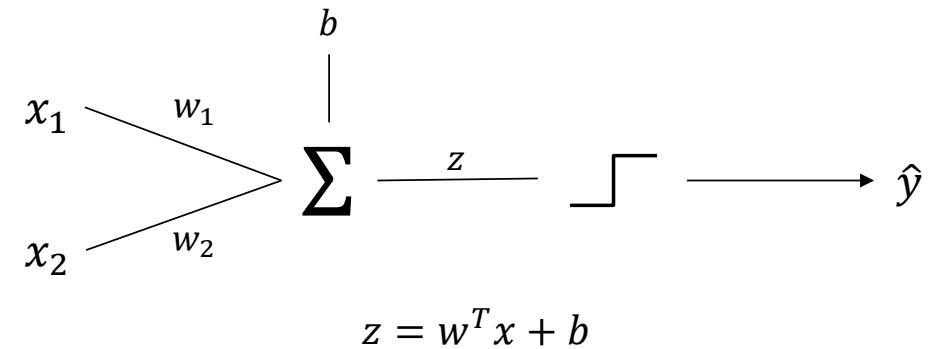
$$\mathcal{L} = y - \hat{y} = 0 - 1 = -1$$

## Step 3: Update weights

$$w_1 = w_1 + \alpha x_1 \mathcal{L} = 0.5 - 10 = -9.5$$

$$w_2 = w_2 + \alpha x_2 \mathcal{L} = 0.5 - 0.3 = 0.2$$

$$b = b + \alpha \mathcal{L} = -50 - 0.1 = -50.1$$



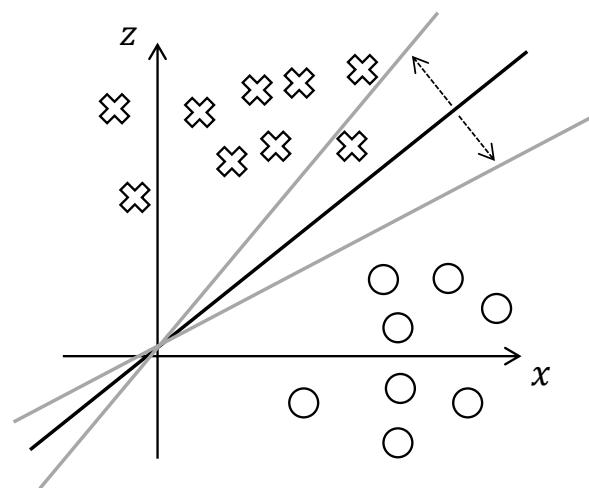
$$\text{step}(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

Size ( $x_1$ )	Rooms ( $x_2$ )	Type ( $y$ )	
100	3	0	$y = 0$ apartment
120	3	1	$y = 1$ house
65	2	0	

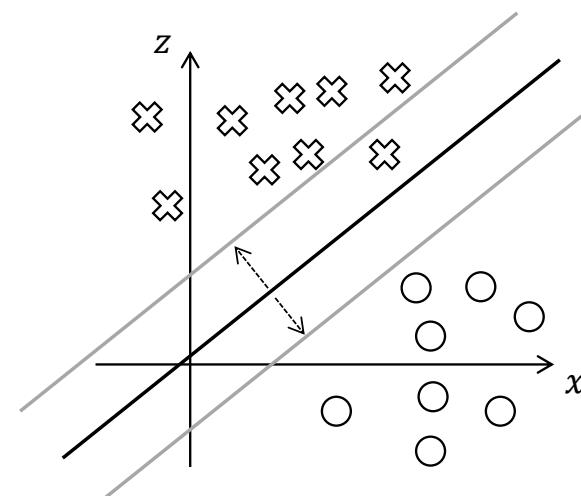
# Training the Perceptron: Decision Boundary

$$z = w_1x_1 + w_2x_2 + b$$

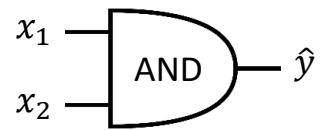
$w$  determines the slope



$b$  determines the intercept



## Example: Logic Gates

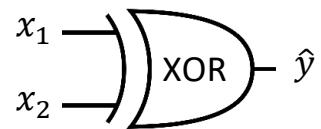


$$w_1 = ?$$

$$w_2 = ?$$

$$\mathbf{b} = ?$$

$x_1$	$x_2$	$y$
0	0	0
0	1	0
1	0	0
1	1	1

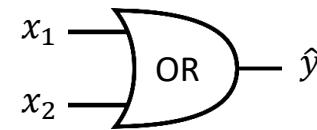


$$w_1 = ?$$

$$w_2 = ?$$

$$\mathbf{b} = ?$$

$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	0



$$w_1 = ?$$

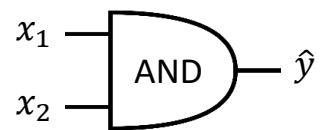
$$w_2 = ?$$

$$\mathbf{b} = ?$$

$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	1

$$\hat{y} = \begin{cases} 1, & \text{if } w^T x + b \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

## Example: Logic Gates

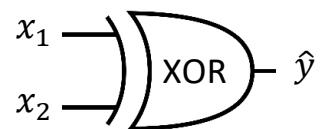


$$w_1 = 1$$

$$w_2 = 1$$

$$b = -1.5$$

$x_1$	$x_2$	$y$
0	0	0
0	1	0
1	0	0
1	1	1

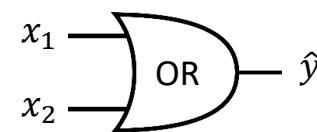


$$w_1 = ?$$

$$w_2 = ?$$

$$b = ?$$

$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	0

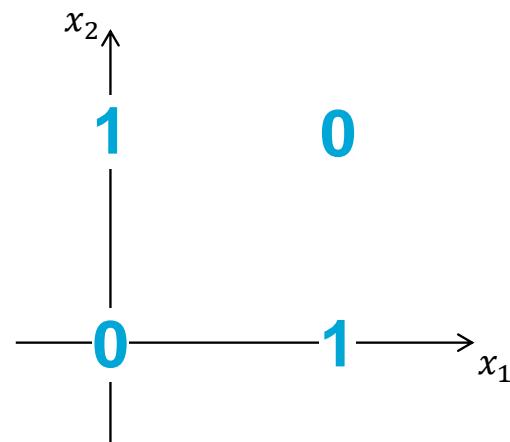
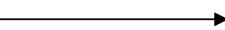


$$w_1 = 1$$

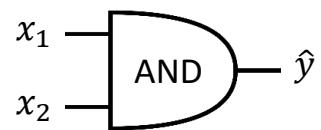
$$w_2 = 1$$

$$b = -0.5$$

$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	1



## Example: Logic Gates

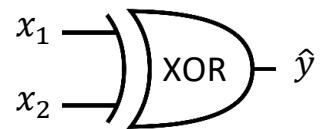


$$w_1 = 1$$

$$w_2 = 1$$

$$b = -1.5$$

$x_1$	$x_2$	$y$
0	0	0
0	1	0
1	0	0
1	1	1

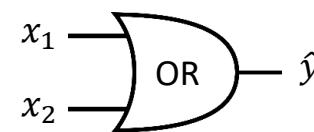


$$w_1 = ?$$

$$w_2 = ?$$

$$b = ?$$

$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	0

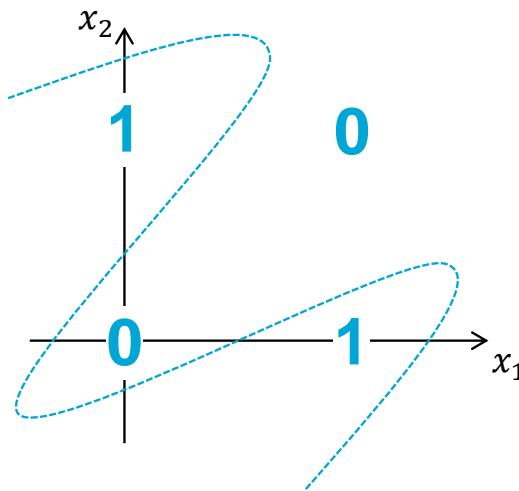


$$w_1 = 1$$

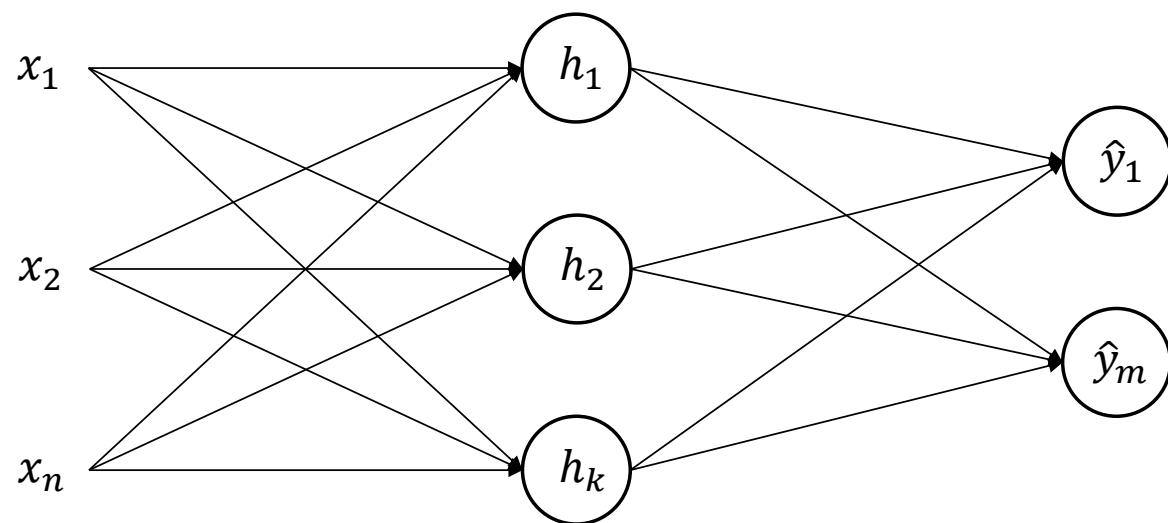
$$w_2 = 1$$

$$b = -0.5$$

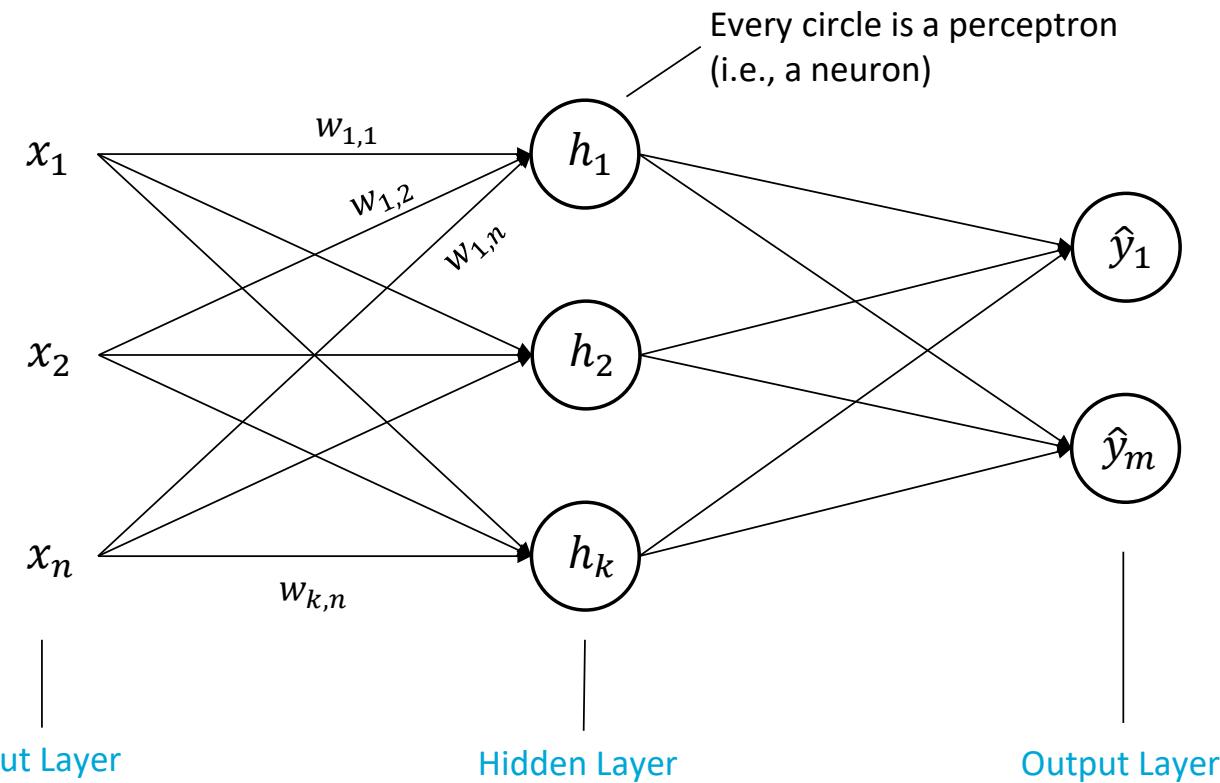
$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	1



# The Multilayer Perceptron



# The Multilayer Perceptron

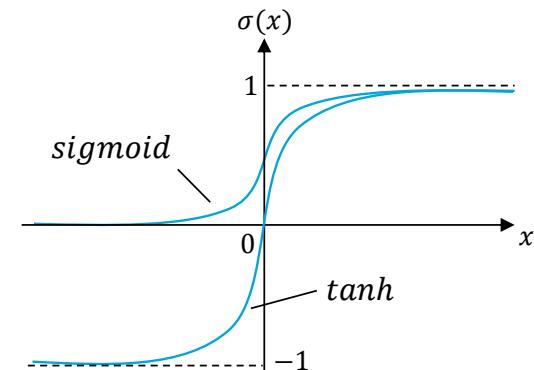


# Activation Functions

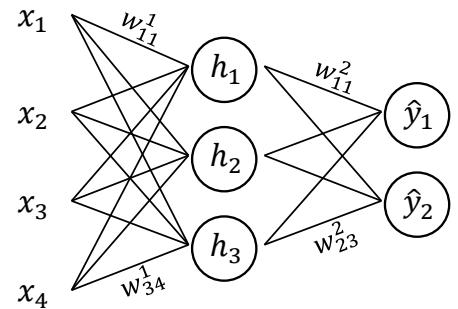
- In a multilayer perceptron, the activation function is referred to as  $\sigma$  (sigma).
- The activation functions are necessary to introduce **nonlinearities** in the network.
- Smooth (**differentiable**) activation functions are suited to perform gradient descent.
- **Sigmoid** (logistic) and hyperbolic tangent (**tanh**) are common choices.

$$\sigma_{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

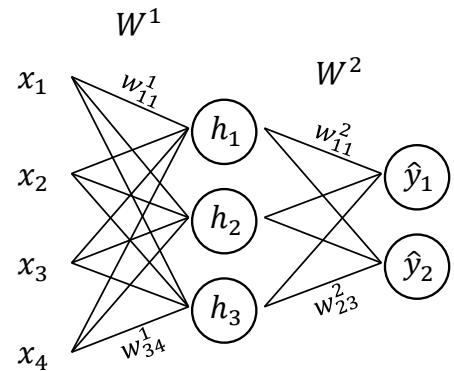
$$\sigma_{tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



# Forward Propagation



# Forward Propagation



$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

$$h = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

$$\hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix}$$

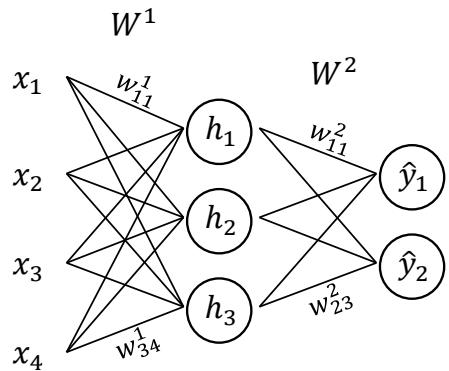
$$W^1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 & w_{14}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 & w_{24}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 & w_{34}^1 \end{bmatrix}$$

$$b^1 = \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \end{bmatrix}$$

$$W^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 \end{bmatrix}$$

$$b^2 = \begin{bmatrix} b_1^2 \\ b_2^2 \end{bmatrix}$$

## Forward Propagation



$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \quad h = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} \quad \hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix}$$

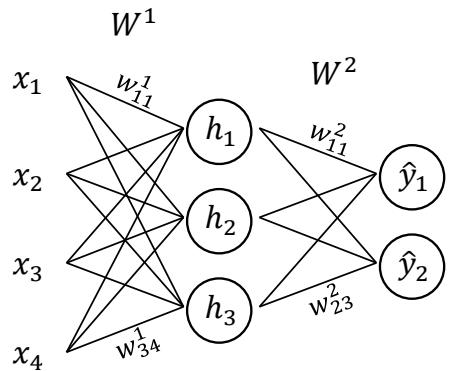
$$W^1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 & w_{14}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 & w_{24}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 & w_{34}^1 \end{bmatrix} \quad b^1 = \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \end{bmatrix}$$

$$W^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 \end{bmatrix} \quad b^2 = \begin{bmatrix} b_1^2 \\ b_2^2 \end{bmatrix}$$

$$z^1 = W^1 x + b^1 \quad h = \sigma(z^1) = \begin{bmatrix} \sigma(z_1^1) \\ \sigma(z_2^1) \\ \sigma(z_3^1) \end{bmatrix}$$

$$z^2 = W^2 h + b^2 \quad \hat{y} = \sigma(z^2) = \begin{bmatrix} \sigma(z_1^2) \\ \sigma(z_2^2) \end{bmatrix}$$

# Forward Propagation



$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \quad h = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} \quad \hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix}$$

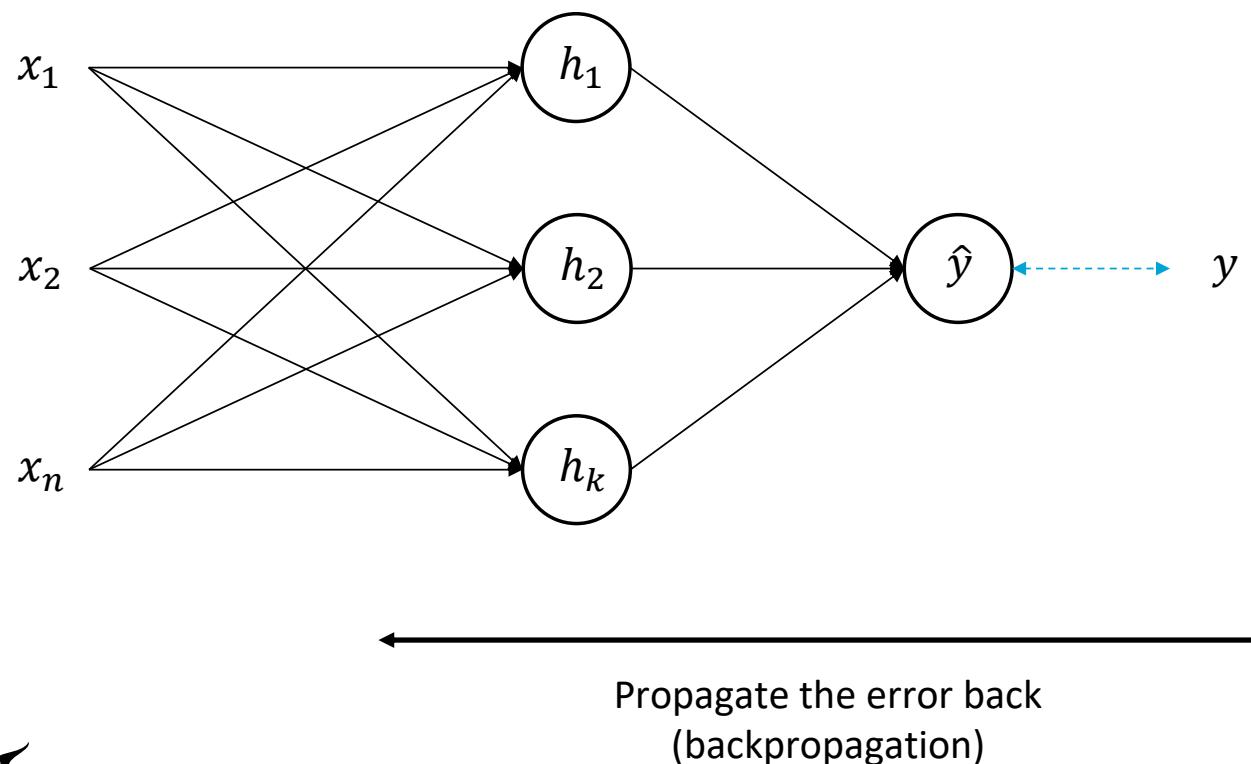
$$W^1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 & w_{14}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 & w_{24}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 & w_{34}^1 \end{bmatrix} \quad b^1 = \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \end{bmatrix}$$

$$W^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 \end{bmatrix} \quad b^2 = \begin{bmatrix} b_1^2 \\ b_2^2 \end{bmatrix}$$

$$z^1 = W^1 x + b^1 \quad h = \sigma(z^1) = \begin{bmatrix} \sigma(z_1^1) \\ \sigma(z_2^1) \\ \sigma(z_3^1) \end{bmatrix} \quad \longrightarrow \quad \hat{y} = \sigma(W^2 h + b^2) = \sigma(W^2 \sigma(W^1 x + b^1) + b^2)$$

$$z^2 = W^2 h + b^2 \quad \hat{y} = \sigma(z^2) = \begin{bmatrix} \sigma(z_1^2) \\ \sigma(z_2^2) \end{bmatrix}$$

## Minimizing a Fitness Function (or Loss Function)



Compare using a loss function:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N (y - \hat{y})^2$$

# Gradient Descent

We want to **minimize the loss function** for all samples in the training set:

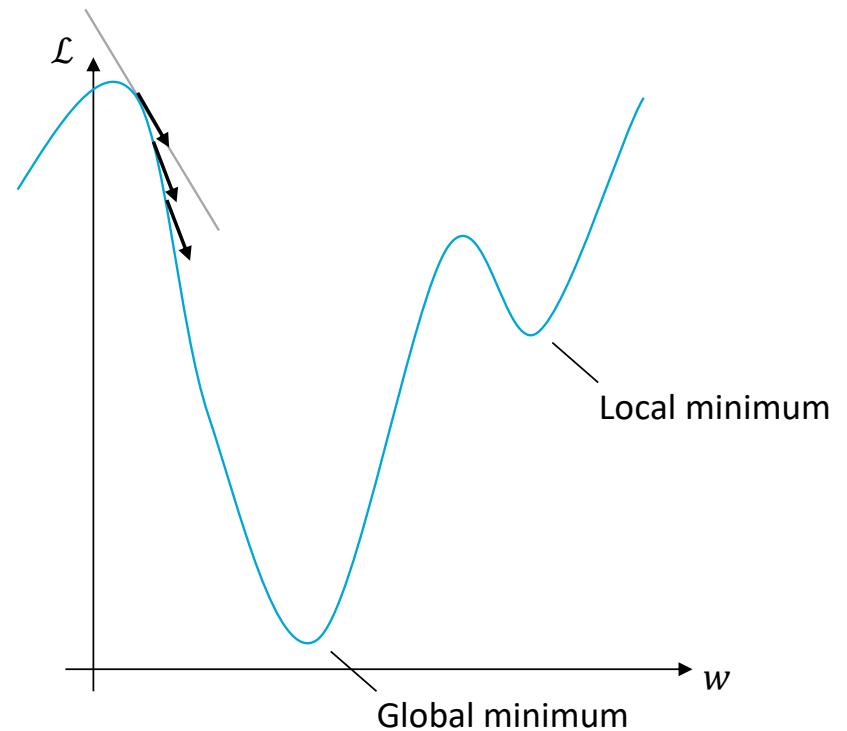
$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N (y - \hat{y})^2$$

The loss function is dependent on correct outputs ( $y$ ) and predicted outputs ( $\hat{y}$ ), which in turn depend on inputs ( $x$ ) and weights ( $w$  and  $b$ ).

**The only element we can control are the weights.** Thus, we search for the set of weights that minimizes  $\mathcal{L}$  for all training samples.

Finding the global minimum analytically is computationally unfeasible. Instead, we update the weights step by step until we find a set of weights that minimizes  $\mathcal{L}$ , using the **gradient descent update**:

$$w^{t+1} = w^t - \alpha \frac{\partial \mathcal{L}(w)}{\partial w} \Big|_{w=w^t}$$



# Gradient Descent Update

$$w^{t+1} = w^t - \alpha \frac{\partial \mathcal{L}}{\partial w} \Big|_{w=w^t}$$

Gradient Update

$$\frac{\partial \mathcal{L}}{\partial w_{jk}^{(l)}} = h_k^{(l-1)} \sigma'(z_j^{(l)}) \frac{\partial \mathcal{L}}{\partial h_j^{(l)}}$$

Derivative of the weights

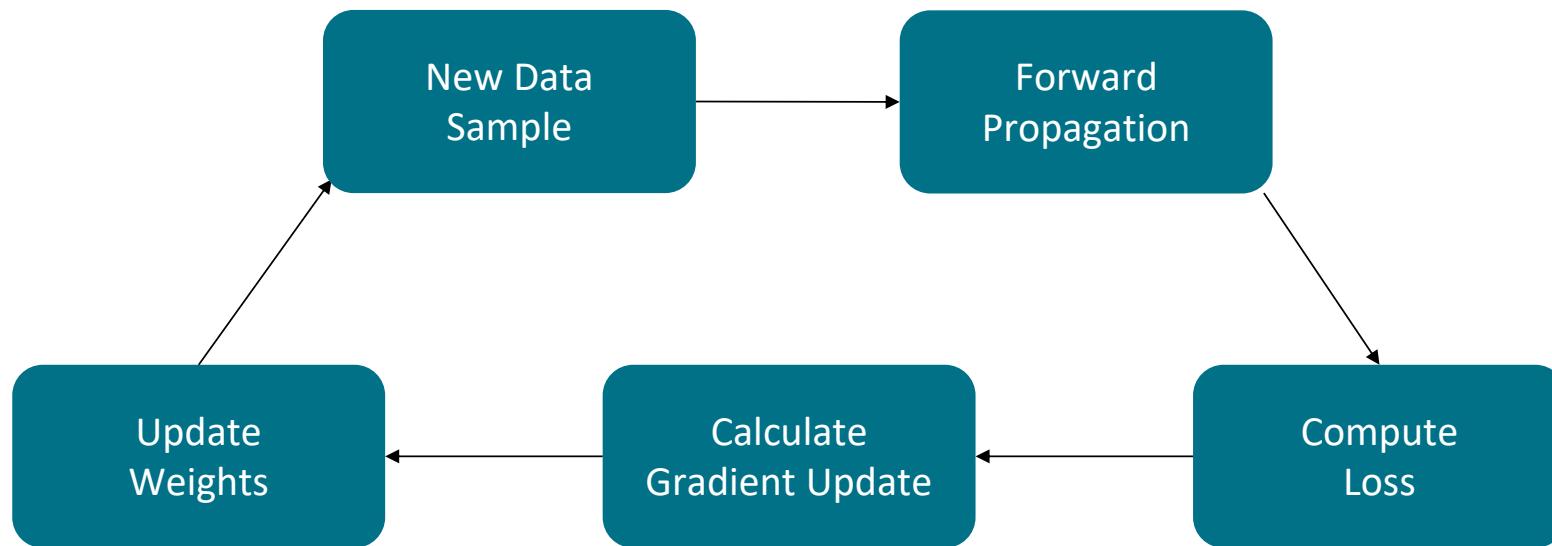
$$\frac{\partial \mathcal{L}}{\partial h_j^{(l)}} = \sum_{i=0}^{n_{l+1}-1} w_{ij}^{(l+1)} \sigma'(z_i^{(l+1)}) \frac{\partial \mathcal{L}}{\partial h_i^{(l+1)}}$$

Derivative of  $\mathcal{L}$  with respect to  $h_j^{(l)}$   
for a layer  $(l)$

$$\frac{\partial \mathcal{L}}{\partial h_j^{(L)}} = 2(y_j - h_j^{(L)})$$

Derivative of  $\mathcal{L}$  with respect to  $h_j^{(L)}$   
for the final layer  $(L)$   
(Example with Mean Squared Error loss)

# Training Procedure



# Lecture Recap

- Introduction to the **perceptron**;
- How to **train** the perceptron;
- **Limitations** of the perceptron;
- The **multilayer perceptron**;
- **Feedforward** inference;
- **Backpropagation** through **gradient descent**;
- **Gradient descent update** equations.

## Next Lecture

- **Gradient descent update** equations deep dive (lecture 1.5);
- Decisions to be taken when training a multilayer perceptron (**hyperparameters tuning**):
  - Activation functions;
  - Number of layers and number of neurons;
  - Regularization;
  - Learning rate;
  - Batch size.



# **Artificial Neural Networks**

## Training a Multilayer Perceptron

CSE2530 - Enrico Liscio



# Content Overview

Decisions to be taken when training a multilayer perceptron (**hyperparameters**):

- Activation functions and loss functions;
- Number of layers and number of neurons;
- Regularization;
- Learning rate;
- Batch size.

# Material

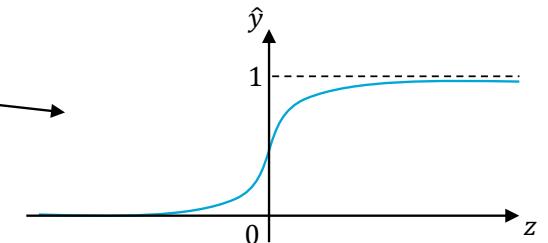
- Ian Goodfellow - Deep Learning (<https://www.deeplearningbook.org>) (further reading):
  - Chapter 5: sections 5.2 to 5.3.1 and 5.5 to 5.5.1
  - Chapter 6: sections 6.2, 6.3, and 6.4
  - Chapter 7: section 7.1
  - Chapter 8: section 8.1.3, and 8.4
- TensorFlow Playground: <http://playground.tensorflow.org>

# Activation Functions – Output Layer

The activation function ( $\sigma$ ) for the output layer's neurons depends on task and number of classes ( $K$ ).

- **Binary classification ( $K = 1$ ):**

$$\hat{y} = \text{sigmoid}(z) = \frac{1}{1+e^{-z}}$$

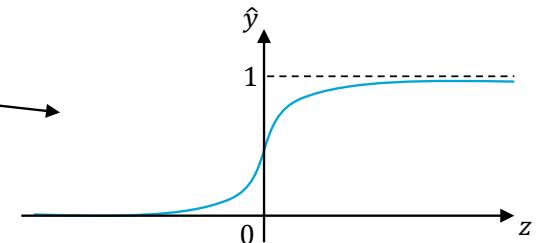


# Activation Functions – Output Layer

The activation function ( $\sigma$ ) for the output layer's neurons depends on task and number of classes ( $K$ ).

- **Binary classification ( $K = 1$ ):**

$$\hat{y} = \text{sigmoid}(z) = \frac{1}{1+e^{-z}}$$



- **Multiclass classification ( $K > 1$ ):**

- With non mutually exclusive labels (multilabel):

$$\hat{y}_i = \text{sigmoid}(z_i) = \frac{1}{1+e^{-z_i}}$$

- With mutually exclusive labels:

$$\hat{y}_i = \text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

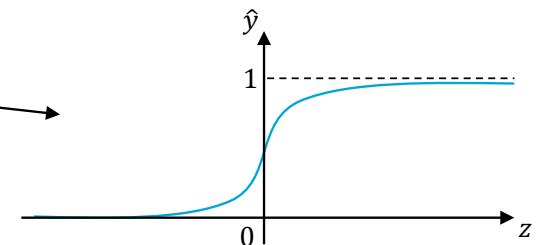
The output is normalized,  
the network outputs  
probability scores:  
$$\sum_{j=1}^K \hat{y}_i = 1$$

# Activation Functions – Output Layer

The activation function ( $\sigma$ ) for the output layer's neurons depends on task and number of classes ( $K$ ).

- **Binary classification ( $K = 1$ ):**

$$\hat{y} = \text{sigmoid}(z) = \frac{1}{1+e^{-z}}$$



- **Multiclass classification ( $K > 1$ ):**

- With non mutually exclusive labels (multilabel):  $\hat{y}_i = \text{sigmoid}(z_i) = \frac{1}{1+e^{-z_i}}$

- With mutually exclusive labels:  $\hat{y}_i = \text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$

The output is normalized,  
the network outputs  
probability scores:  
$$\sum_{j=1}^K \hat{y}_i = 1$$

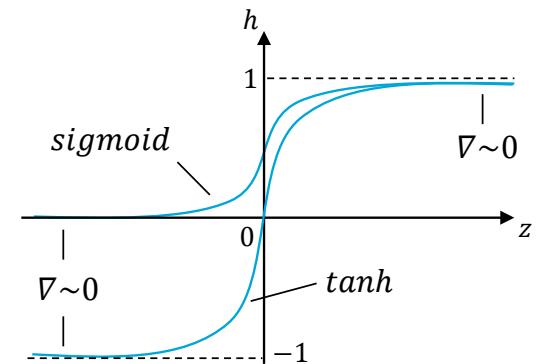
- **Regression (for any  $K$ ):**

$$\hat{y}_i = z_i$$

# Activation Functions – Hidden Layers

The activation function ( $\sigma$ ) for the hidden layers' neurons is necessary to introduce **non-linearities** in the network. Multiple options are available:

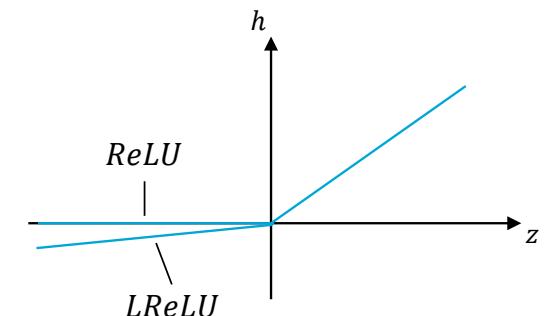
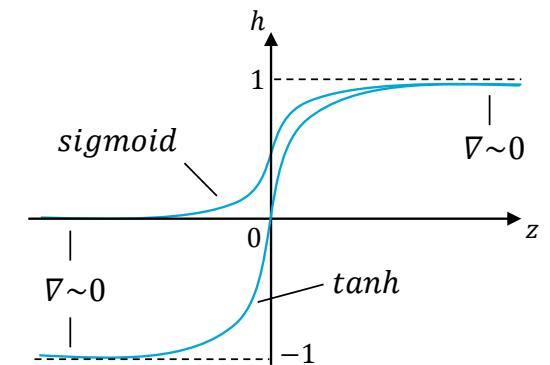
- **Sigmoid (sigmoid)**  
Pro: interpretable output (it's a probability).  
Con: vanishing gradients (small and large values of  $z$  cause the gradients to be close to 0).
- **Hyperbolic Tangent ( $tanh$ )**  
Pro: allows for positive and negative output.



# Activation Functions – Hidden Layers

The activation function ( $\sigma$ ) for the hidden layers' neurons is necessary to introduce **non-linearities** in the network. Multiple options are available:

- **Sigmoid (sigmoid)**  
Pro: interpretable output (it's a probability).  
Con: vanishing gradients (small and large values of  $z$  cause the gradients to be close to 0).
- **Hyperbolic Tangent ( $tanh$ )**  
Pro: allows for positive and negative output.
- **Rectified Linear Unit ( $ReLU$ )**  $\rightarrow h = \max(0, z)$   
Pro: no vanishing gradients. Con:  $\nabla = 0$  for negative values of  $z$ .
- **Leaky Rectified Linear Unit ( $LReLU$ )**  
Pro: nonlinear but piecewise linear, thus easy to differentiate. Always  $\nabla \neq 0$



# Loss Function

The loss function  $\mathcal{L}$  depends on the output layer's activation function (and thus, on the task). Common choices are:

- Multiclass classification (softmax activation): **categorical cross-entropy**

$$\mathcal{L} = - \sum_{i=1}^K y_i \log(\hat{y}_i)$$

- Binary classification (sigmoid activation): **binary cross-entropy**

$$\mathcal{L} = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

- Multiclass multilabel classification (sigmoid activation):  
**binary cross-entropy on each output**

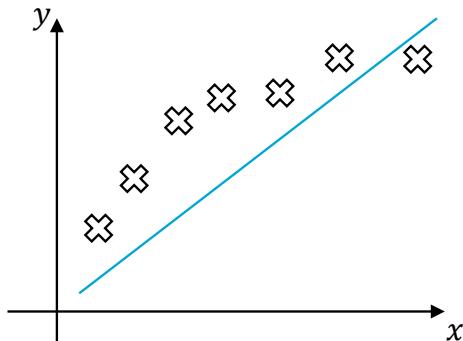
$$\mathcal{L} = - \sum_{i=1}^K y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

- Regression (no activation): **Mean Squared Error (MSE)**

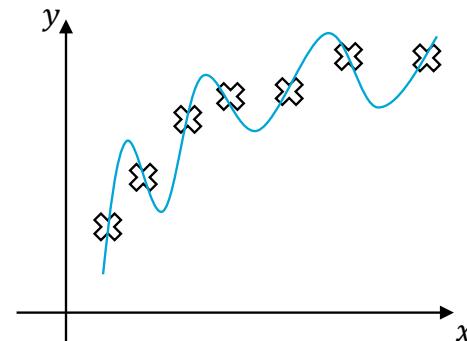
$$\mathcal{L} = \frac{1}{K} \sum_{i=1}^K (y_i - \hat{y}_i)^2$$

# Underfitting vs. Overfitting

**Underfitting** happens when the network has **too few parameters** to be able to model the complexity of the desired input-output mapping. The network does a poor job.



**Overfitting** happens when the network has **too many parameters** for the desired input-output mapping. The network does an unrealistically good job, which does not generalize well to unseen examples.

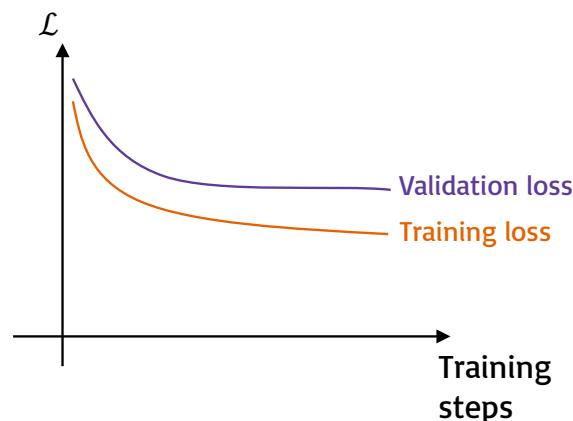


# Underfitting vs. Overfitting: How to Detect

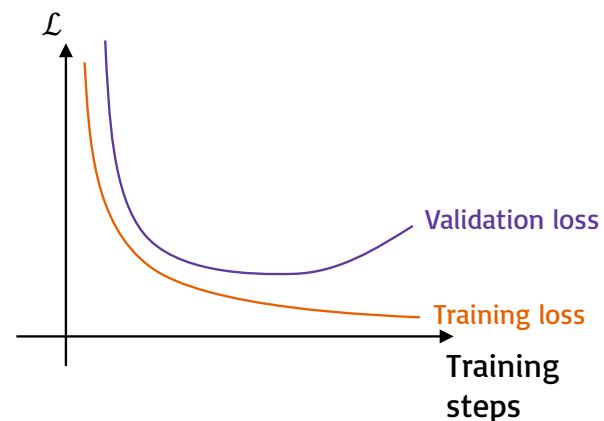
Split the dataset in **training set** (typically 80% of the data, used to update the weights and train the model) and **validation set** (typically 20% of the data, used to test the generalizability of the model).

At every training step, plot the **training loss** (the value of the loss function evaluated on the training set) and the **validation loss** (the value of the loss function evaluated on the validation set).

**Underfitting:** both training loss and validation loss are high.



**Overfitting:** the training loss keeps decreasing, the validation loss starts increasing.



# Underfitting vs. Overfitting: How to Prevent

How to prevent underfitting:

- **Increase complexity** of the network (increase the number of neurons and number of layers).  
**Universal Approximation Theorem:** a MLP with a linear output layer, at least one hidden layer with any squashing activation function (e.g., *sigmoid* or *tanh*) can approximate any function, provided that the network is given enough hidden neurons.

How to prevent overfitting:

- Ensure that training and validation sets are **Independent and Identically Distributed (IID)**;
- **Gather more data** (when possible);
- **Augment data** (e.g., crop/zoom/rotate images in the training set) (when possible);
- **Decrease complexity** of the network (decrease the number of neurons and number of layers);
- **Apply regularization.**

# Regularization

By constraining the weights to have a **small magnitude**, we prevent the model from overfitting.

We constrain the magnitude of the weights by adding a **regularization term** to the loss function:

$$\mathcal{L} = \mathcal{L}_{error} + \lambda \mathcal{L}_{reg}$$

where  $\mathcal{L}_{error}$  is the **error loss** caused by the difference between desired output and model prediction (e.g., cross-entropy loss),  $\mathcal{L}_{reg}$  is the **regularization loss**, and  $\lambda$  is the **regularization rate** (which determines how impactful the regularization loss is).

Examples of regularization loss are:

- L1 regularization:  $\mathcal{L}_{reg} = \sum_i |w_i|$
- L2 regularization:  $\mathcal{L}_{reg} = \sum_i w_i^2$

# Weights Initialization

Biases are typically initialized to 0.

However, initializing all weights to 0 (or to the same constant value) would lead all neurons to learn the same features during training. **Random initialization** is necessary to **break the symmetry** of the MLP.

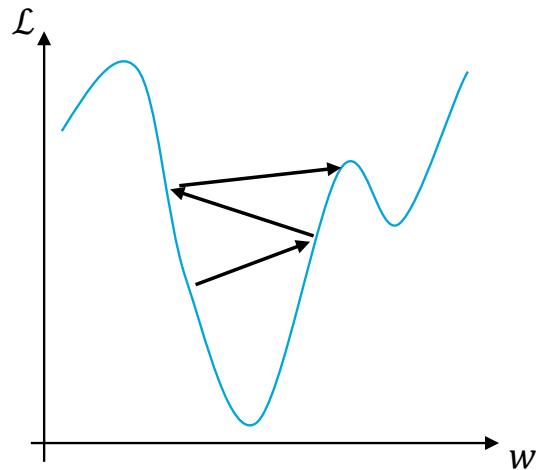
Common initialization choices for the weights are:

- A **normal (Gaussian) distribution** with mean 0 and variance 1:  $w_{jk}^{(l)} = \mathcal{N}(0,1)$
- For *sigmoid* and *tanh*, **Xavier initialization** (with  $n$  referring to the number of neurons):  $w_{jk}^{(l)} = \mathcal{N}\left(0, \frac{1}{n_{l-1}}\right)$
- For *ReLU* and *LReLU*, **He initialization** (with  $n$  referring to the number of neurons):  $w_{jk}^{(l)} = \mathcal{N}\left(0, \frac{2}{n_{l-1}}\right)$

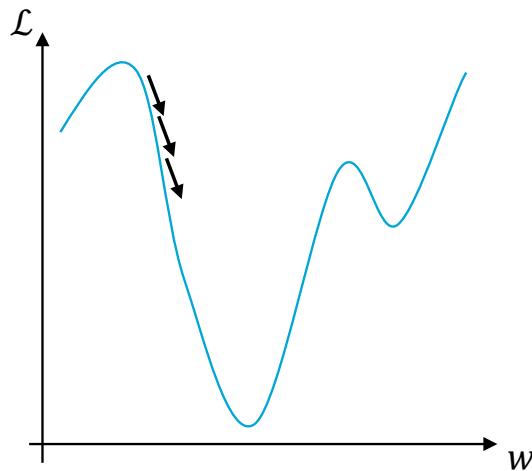
# Learning Rate

The learning rate  $\alpha$  controls the speed in which the weights are updated:  $w^{t+1} = w^t - \alpha \frac{\partial \mathcal{L}(w)}{\partial w} \Big|_{w=w^t}$

A higher  $\alpha$  leads to faster convergence, but with the risk of missing a minimum and not reaching convergence.



A lower  $\alpha$  leads to slower convergence, with more certainty of finding a minimum, but with the risk of getting stuck in a local minimum.

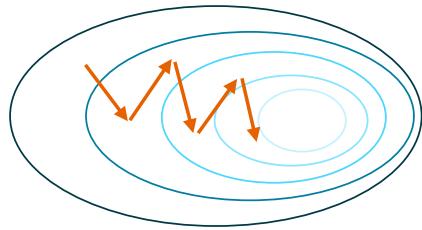


# Mini-Batch Gradient Descent

## Stochastic Gradient Descent:

One training example at a time

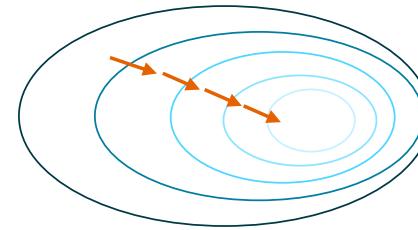
- Fast computation
- High variance



## Batch Gradient Descent:

All training examples at a time

- Slow computation
- Low variance

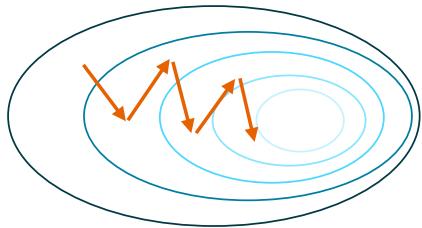


# Mini-Batch Gradient Descent

## Stochastic Gradient Descent:

One training example at a time

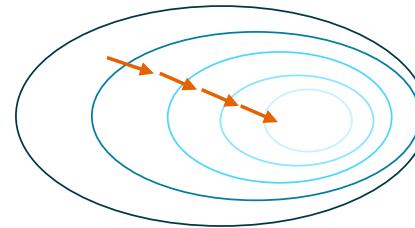
- Fast computation
- High variance



## Batch Gradient Descent:

All training examples at a time

- Slow computation
- Low variance



## Mini-batch Gradient Descent:

Some training examples at a time  
(common batch sizes: 4, 8, 16, 32)

- Fast computation
- Low variance

# Training Procedure

- Implement a **modular multilayer perceptron** model where hyperparameters can be easily changed. Initialize the MLP weights and biases.
- Shuffle the dataset and split it **training set** (~70% of the data), **validation set** (~15%) and **test set** (~15%), ensuring that the three sets are IID.
- Split the training set in  $s = \text{train\_set\_size}/\text{batch\_size}$  mini-batches. At every **training step**, perform the weights update based on the training examples in the mini-batch. After all mini-batches in the training set are used for training, one training **epoch** is terminated. Shuffle the data, re-create  $s$  mini-batches, and train again. It is common to repeat the training procedure for several epochs (3-5). Save the model weights after every epoch (or even more often).
- Select multiple checkpoints per epoch (e.g., 3-5 per epoch). At every checkpoint, compute the loss on all training set examples (**training loss**) and all validation set examples (**validation loss**). Plot the two losses (during or after the training procedure) to evaluate underfitting and overfitting.
- Use the validation set to **tune the hyperparameters** (with the goal of maximizing performance on the validation set): choice of activation function, number of hidden layers, number of neurons per layer, learning rate, type of regularization, regularization rate, batch size.
- Report the performance **results on the test set**.

# Lecture Recap

- Backpropagation equations;
- Hyperparameters tuning:
  - Activation functions and loss functions;
  - Number of layers and number of neurons;
  - Regularization;
  - Learning rate;
  - Batch size.



# Next lecture

- Introduction to Convolutional Neural Networks (CNN) and their applications;
- Introduction to Recurrent Neural Networks (RNN) and their applications;
- Intuition behind DALL·E.

# **Artificial Neural Networks**

## Gradient Descent Update

CSE2530 - Enrico Liscio



# Content Overview

Gradient descent update equations.

## Material:

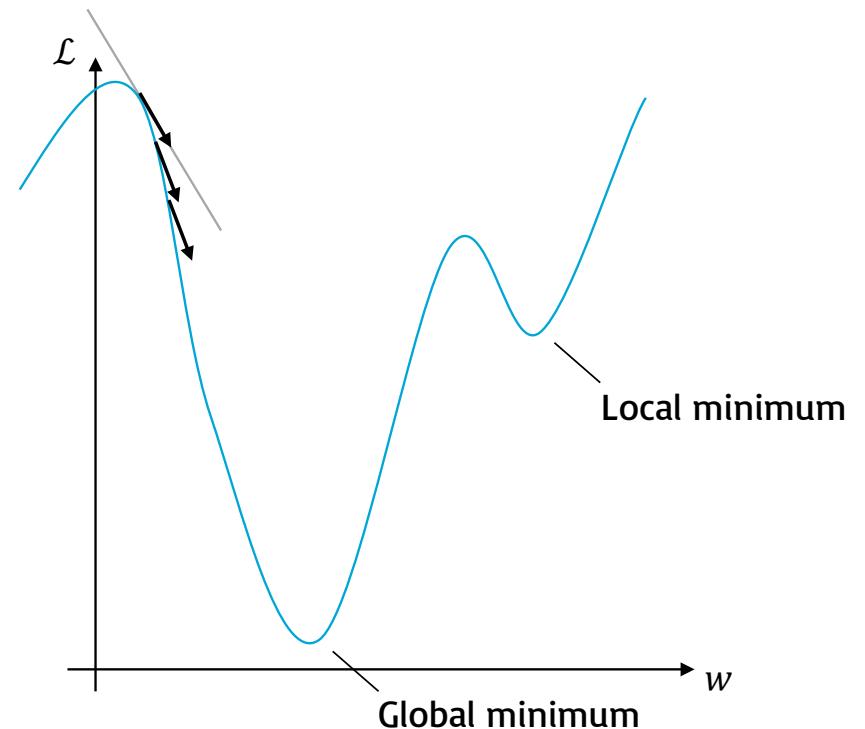
- Michael Negnevitsky – Artificial Intelligence: section 6.4 (recommended);
- Michael Nielsen - Neural Networks and Deep Learning (further reading):  
<http://neuralnetworksanddeeplearning.com/>
- Ian Goodfellow - Deep Learning (<https://www.deeplearningbook.org>) (further reading):
  - Chapter 6: section 6.5
- Grant Sanderson - (3blue1brown) (further reading):  
Intuition behind backpropagation: <https://youtu.be/Ilg3gGewQ5U>  
Math behind backpropagation: <https://youtu.be/tleHLnjs5U8>



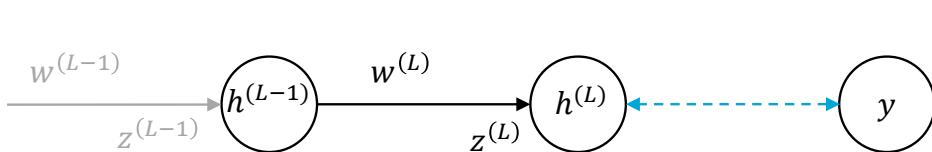
# Gradient Descent (recap)

- Computing the optimum for all samples and weights is computationally unfeasible even for relatively small networks.
- Instead, we attempt to compute the global minimum with a numerical approach, while being aware that we may end up in a local minimum.
- For every  $w_{jk}^{(l)}$ , we compute the gradient of the loss function and take a step in its direction.

$$w^{t+1} = w^t - \alpha \frac{\partial \mathcal{L}(w)}{\partial w} \Big|_{w=w^t}$$



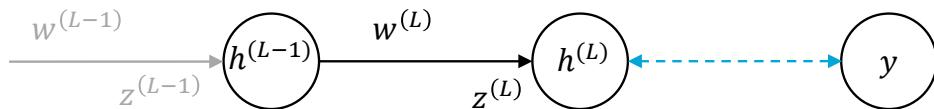
# Loss Gradient for a Simple Network



Loss for one training example

$$\mathcal{L}_0 = (y - h^{(L)})^2$$
$$h^{(L)} = \hat{y}$$
$$z^{(L)} = w^{(L)}h^{(L-1)} + b^{(L)}$$
$$h^{(L)} = \sigma(z^{(L)})$$
$$\sigma'_{sigmoid}(z^{(L)}) = \sigma(z^{(L)})\left(1 - \sigma(z^{(L)})\right) = h^{(L)}(1 - h^{(L)})$$

# Loss Gradient for a Simple Network

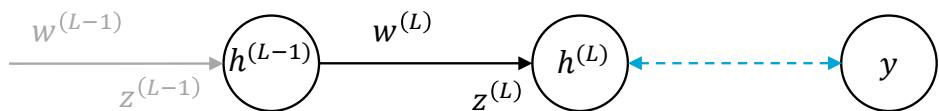


$$\frac{\partial \mathcal{L}_0}{\partial w^{(L)}} = \frac{\partial \mathcal{L}_0}{\partial h^{(L)}} \frac{\partial h^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial w^{(L)}}$$

Loss for one training example

$$\mathcal{L}_0 = (y - h^{(L)})^2$$
$$h^{(L)} = \hat{y}$$
$$z^{(L)} = w^{(L)}h^{(L-1)} + b^{(L)}$$
$$h^{(L)} = \sigma(z^{(L)})$$
$$\sigma'_{sigmoid}(z^{(L)}) = \sigma(z^{(L)}) (1 - \sigma(z^{(L)})) = h^{(L)}(1 - h^{(L)})$$

# Loss Gradient for a Simple Network



$$\frac{\partial \mathcal{L}_0}{\partial w^{(L)}} = \frac{\partial \mathcal{L}_0}{\partial h^{(L)}} \frac{\partial h^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial w^{(L)}} = 2(y - h^{(L)})\sigma'(z^{(L)})h^{(L-1)}$$

$$\frac{\partial \mathcal{L}_0}{\partial h^{(L)}} = 2(y - h^{(L)})$$

$$\frac{\partial h^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)})$$

$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = h^{(L-1)}$$

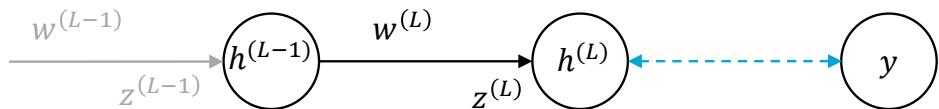
Loss for one training example

$\mathcal{L}_0 = (y - h^{(L)})^2$   
 $z^{(L)} = w^{(L)}h^{(L-1)} + b^{(L)}$   
 $\sigma'_{sigmoid}(z^{(L)}) = \sigma(z^{(L)})(1 - \sigma(z^{(L)})) = h^{(L)}(1 - h^{(L)})$

$h^{(L)} = \hat{y}$

---

# Loss Gradient for a Simple Network



$$\frac{\partial \mathcal{L}_0}{\partial \mathbf{w}^{(L)}} = \frac{\partial \mathcal{L}_0}{\partial h^{(L)}} \frac{\partial h^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial w^{(L)}} = 2(y - h^{(L)}) \sigma'(z^{(L)}) h^{(L-1)}$$

$$\frac{\partial \mathcal{L}_0}{\partial h^{(L)}} = 2(y - h^{(L)})$$

$$\frac{\partial h^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)})$$

$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = h^{(L-1)}$$

Loss for one training example

$$\mathcal{L}_0 = (y - h^{(L)})^2$$

$$h^{(L)} = \hat{y}$$

$$z^{(L)} = w^{(L)}h^{(L-1)} + b^{(L)}$$

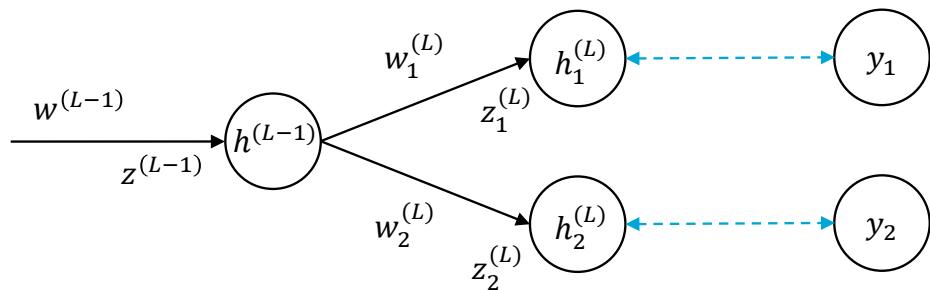
$$h^{(L)} = \sigma(z^{(L)})$$

$$\sigma'_{sigmoid}(z^{(L)}) = \sigma(z^{(L)}) (1 - \sigma(z^{(L)})) = h^{(L)}(1 - h^{(L)})$$

$$\frac{\partial \mathcal{L}_0}{\partial \mathbf{b}^{(L)}} = \frac{\partial \mathcal{L}_0}{\partial h^{(L)}} \frac{\partial h^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial b^{(L)}} = 2(y - h^{(L)}) \sigma'(z^{(L)})$$

$$\frac{\partial z^{(L)}}{\partial b^{(L)}} = 1$$

# Loss Gradient for a Less Simple Network



Loss for one training example

$$\mathcal{L}_0 = \sum_{i=1}^2 (y_i - h_i^{(L)})^2 = (y_1 - h_1^{(L)})^2 + (y_2 - h_2^{(L)})^2$$

$$z_1^{(L)} = w_1^{(L)} h^{(L-1)} + b_1^{(L)} \quad h_1^{(L)} = \sigma(z_1^{(L)})$$

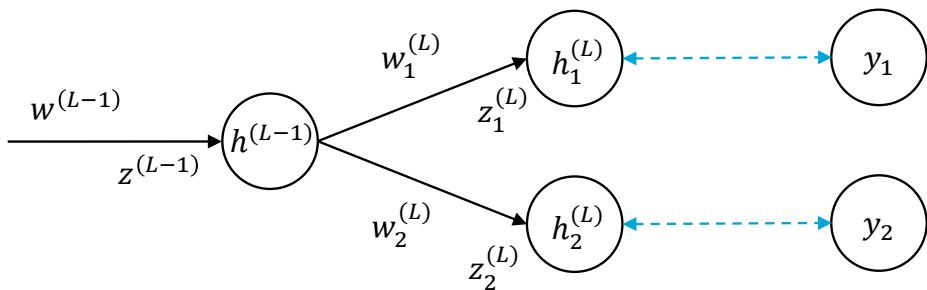
$$z_2^{(L)} = w_2^{(L)} h^{(L-1)} + b_2^{(L)} \quad h_2^{(L)} = \sigma(z_2^{(L)})$$

$$z^{(L-1)} = w^{(L-1)} h^{(L-2)} + b^{(L-1)} \quad h^{(L-1)} = \sigma(z^{(L-1)})$$

$$\frac{\partial \mathcal{L}_0}{\partial w_1^{(L)}} = \frac{\partial \mathcal{L}_0}{\partial h_1^{(L)}} \frac{\partial h_1^{(L)}}{\partial z_1^{(L)}} \frac{\partial z_1^{(L)}}{\partial w_1^{(L)}} = 2(y_1 - h_1^{(L)})\sigma'(z_1^{(L)})h^{(L-1)}$$

$$\frac{\partial \mathcal{L}_0}{\partial w_2^{(L)}} = \frac{\partial \mathcal{L}_0}{\partial h_2^{(L)}} \frac{\partial h_2^{(L)}}{\partial z_2^{(L)}} \frac{\partial z_2^{(L)}}{\partial w_2^{(L)}} = 2(y_2 - h_2^{(L)})\sigma'(z_2^{(L)})h^{(L-1)}$$

# Loss Gradient for a Less Simple Network

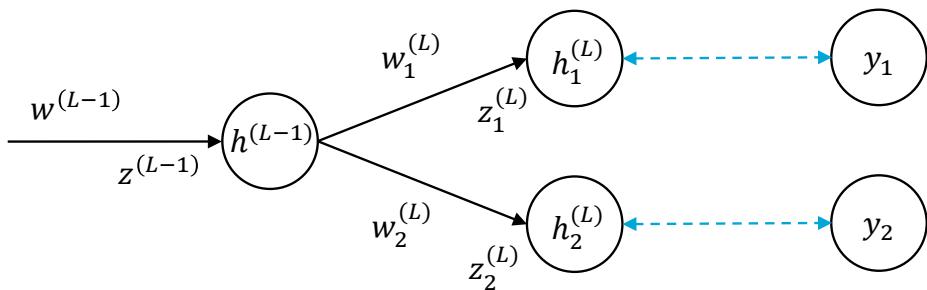


$$\frac{\partial \mathcal{L}_0}{\partial \mathbf{w}^{(L-1)}} = \frac{\partial \mathcal{L}_0}{\partial h^{(L-1)}} \frac{\partial h^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial \mathbf{w}^{(L-1)}}$$

Loss for one training example

$$\begin{aligned}\mathcal{L}_0 &= \sum_{i=1}^2 (y_i - h_i^{(L)})^2 = (y_1 - h_1^{(L)})^2 + (y_2 - h_2^{(L)})^2 \\ z_1^{(L)} &= w_1^{(L)} h^{(L-1)} + b_1^{(L)} & h_1^{(L)} &= \sigma(z_1^{(L)}) \\ z_2^{(L)} &= w_2^{(L)} h^{(L-1)} + b_2^{(L)} & h_2^{(L)} &= \sigma(z_2^{(L)}) \\ z^{(L-1)} &= w^{(L-1)} h^{(L-2)} + b^{(L-1)} & h^{(L-1)} &= \sigma(z^{(L-1)}) \\ \frac{\partial \mathcal{L}_0}{\partial w_1^{(L)}} &= \frac{\partial \mathcal{L}_0}{\partial h_1^{(L)}} \frac{\partial h_1^{(L)}}{\partial z_1^{(L)}} \frac{\partial z_1^{(L)}}{\partial w_1^{(L)}} = 2(y_1 - h_1^{(L)})\sigma'(z_1^{(L)})h^{(L-1)} \\ \frac{\partial \mathcal{L}_0}{\partial w_2^{(L)}} &= \frac{\partial \mathcal{L}_0}{\partial h_2^{(L)}} \frac{\partial h_2^{(L)}}{\partial z_2^{(L)}} \frac{\partial z_2^{(L)}}{\partial w_2^{(L)}} = 2(y_2 - h_2^{(L)})\sigma'(z_2^{(L)})h^{(L-1)}\end{aligned}$$

# Loss Gradient for a Less Simple Network

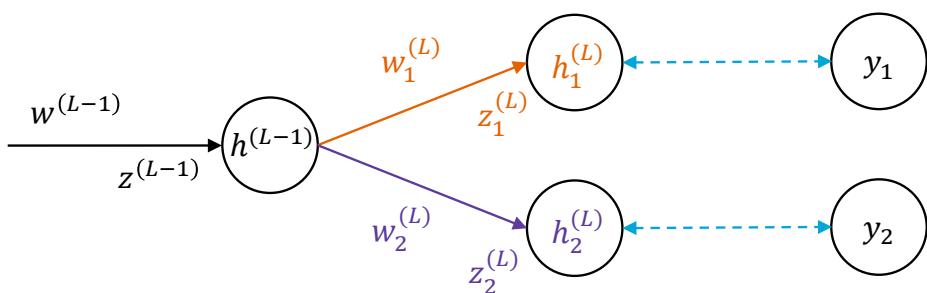


$$\frac{\partial \mathcal{L}_0}{\partial \mathbf{w}^{(L-1)}} = \frac{\partial \mathcal{L}_0}{\partial h^{(L-1)}} \frac{\partial h^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial \mathbf{w}^{(L-1)}}$$

Loss for one training example

$$\begin{aligned}\mathcal{L}_0 &= \sum_{i=1}^2 (y_i - h_i^{(L)})^2 = (y_1 - h_1^{(L)})^2 + (y_2 - h_2^{(L)})^2 \\ z_1^{(L)} &= w_1^{(L)} h^{(L-1)} + b_1^{(L)} & h_1^{(L)} &= \sigma(z_1^{(L)}) \\ z_2^{(L)} &= w_2^{(L)} h^{(L-1)} + b_2^{(L)} & h_2^{(L)} &= \sigma(z_2^{(L)}) \\ z^{(L-1)} &= w^{(L-1)} h^{(L-2)} + b^{(L-1)} & h^{(L-1)} &= \sigma(z^{(L-1)}) \\ \frac{\partial \mathcal{L}_0}{\partial w_1^{(L)}} &= \frac{\partial \mathcal{L}_0}{\partial h_1^{(L)}} \frac{\partial h_1^{(L)}}{\partial z_1^{(L)}} \frac{\partial z_1^{(L)}}{\partial w_1^{(L)}} = 2(y_1 - h_1^{(L)})\sigma'(z_1^{(L)})h^{(L-1)} \\ \frac{\partial \mathcal{L}_0}{\partial w_2^{(L)}} &= \frac{\partial \mathcal{L}_0}{\partial h_2^{(L)}} \frac{\partial h_2^{(L)}}{\partial z_2^{(L)}} \frac{\partial z_2^{(L)}}{\partial w_2^{(L)}} = 2(y_2 - h_2^{(L)})\sigma'(z_2^{(L)})h^{(L-1)}\end{aligned}$$

# Loss Gradient for a Less Simple Network

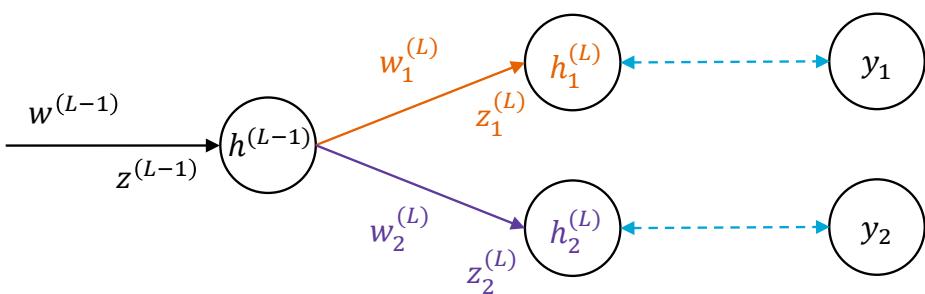


$$\begin{aligned}\frac{\partial \mathcal{L}_0}{\partial \mathbf{w}^{(L-1)}} &= \frac{\partial \mathcal{L}_0}{\partial h^{(L-1)}} \frac{\partial h^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial \mathbf{w}^{(L-1)}} = \\ &= \left( \frac{\partial \mathcal{L}_0}{\partial h_1^{(L)}} \frac{\partial h_1^{(L)}}{\partial z_1^{(L)}} \frac{\partial z_1^{(L)}}{\partial h^{(L-1)}} + \frac{\partial \mathcal{L}_0}{\partial h_2^{(L)}} \frac{\partial h_2^{(L)}}{\partial z_2^{(L)}} \frac{\partial z_2^{(L)}}{\partial h^{(L-1)}} \right) \frac{\partial h^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial \mathbf{w}^{(L-1)}}\end{aligned}$$

Loss for one training example

$$\begin{aligned}\mathcal{L}_0 &= \sum_{i=1}^2 (y_i - h_i^{(L)})^2 = (y_1 - h_1^{(L)})^2 + (y_2 - h_2^{(L)})^2 \\ z_1^{(L)} &= w_1^{(L)} h^{(L-1)} + b_1^{(L)} & h_1^{(L)} &= \sigma(z_1^{(L)}) \\ z_2^{(L)} &= w_2^{(L)} h^{(L-1)} + b_2^{(L)} & h_2^{(L)} &= \sigma(z_2^{(L)}) \\ z^{(L-1)} &= w^{(L-1)} h^{(L-2)} + b^{(L-1)} & h^{(L-1)} &= \sigma(z^{(L-1)}) \\ \frac{\partial \mathcal{L}_0}{\partial w_1^{(L)}} &= \frac{\partial \mathcal{L}_0}{\partial h_1^{(L)}} \frac{\partial h_1^{(L)}}{\partial z_1^{(L)}} \frac{\partial z_1^{(L)}}{\partial w_1^{(L)}} = 2(y_1 - h_1^{(L)}) \sigma'(z_1^{(L)}) h^{(L-1)} \\ \frac{\partial \mathcal{L}_0}{\partial w_2^{(L)}} &= \frac{\partial \mathcal{L}_0}{\partial h_2^{(L)}} \frac{\partial h_2^{(L)}}{\partial z_2^{(L)}} \frac{\partial z_2^{(L)}}{\partial w_2^{(L)}} = 2(y_2 - h_2^{(L)}) \sigma'(z_2^{(L)}) h^{(L-1)}\end{aligned}$$

# Loss Gradient for a Less Simple Network



$$\begin{aligned}
 \frac{\partial \mathcal{L}_0}{\partial \mathbf{w}^{(L-1)}} &= \frac{\partial \mathcal{L}_0}{\partial h^{(L-1)}} \frac{\partial h^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial \mathbf{w}^{(L-1)}} = \\
 &= \left( \frac{\partial \mathcal{L}_0}{\partial h_1^{(L)}} \frac{\partial h_1^{(L)}}{\partial z_1^{(L)}} \frac{\partial z_1^{(L)}}{\partial h^{(L-1)}} + \frac{\partial \mathcal{L}_0}{\partial h_2^{(L)}} \frac{\partial h_2^{(L)}}{\partial z_2^{(L)}} \frac{\partial z_2^{(L)}}{\partial h^{(L-1)}} \right) \frac{\partial h^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial \mathbf{w}^{(L-1)}} = \\
 &= \left( 2(y_1 - h_1^{(L)}) \sigma'(z_1^{(L)}) w_1^{(L)} + 2(y_2 - h_2^{(L)}) \sigma'(z_2^{(L)}) w_2^{(L)} \right) \sigma'(z^{(L-1)}) h^{(L-2)}
 \end{aligned}$$

Loss for one training example

$$\begin{aligned}
 \mathcal{L}_0 &= \sum_{i=1}^2 (y_i - h_i^{(L)})^2 = (y_1 - h_1^{(L)})^2 + (y_2 - h_2^{(L)})^2 \\
 z_1^{(L)} &= w_1^{(L)} h^{(L-1)} + b_1^{(L)} & h_1^{(L)} &= \sigma(z_1^{(L)}) \\
 z_2^{(L)} &= w_2^{(L)} h^{(L-1)} + b_2^{(L)} & h_2^{(L)} &= \sigma(z_2^{(L)}) \\
 z^{(L-1)} &= w^{(L-1)} h^{(L-2)} + b^{(L-1)} & h^{(L-1)} &= \sigma(z^{(L-1)}) \\
 \frac{\partial \mathcal{L}_0}{\partial w_1^{(L)}} &= \frac{\partial \mathcal{L}_0}{\partial h_1^{(L)}} \frac{\partial h_1^{(L)}}{\partial z_1^{(L)}} \frac{\partial z_1^{(L)}}{\partial w_1^{(L)}} = 2(y_1 - h_1^{(L)}) \sigma'(z_1^{(L)}) h^{(L-1)} \\
 \frac{\partial \mathcal{L}_0}{\partial w_2^{(L)}} &= \frac{\partial \mathcal{L}_0}{\partial h_2^{(L)}} \frac{\partial h_2^{(L)}}{\partial z_2^{(L)}} \frac{\partial z_2^{(L)}}{\partial w_2^{(L)}} = 2(y_2 - h_2^{(L)}) \sigma'(z_2^{(L)}) h^{(L-1)}
 \end{aligned}$$

# Gradient Update – General Formula

$$w^{t+1} = w^t - \alpha \frac{\partial \mathcal{L}}{\partial w} \Big|_{w=w^t}$$

Gradient Update

$$\frac{\partial \mathcal{L}}{\partial w_{jk}^{(l)}} = h_k^{(l-1)} \sigma' \left( z_j^{(l)} \right) \frac{\partial \mathcal{L}}{\partial h_j^{(l)}}$$

Derivative of the weights

$$\frac{\partial \mathcal{L}}{\partial h_j^{(l)}} = \sum_{i=0}^{n_{l+1}-1} w_{ij}^{(l+1)} \sigma' \left( z_i^{(l+1)} \right) \frac{\partial \mathcal{L}}{\partial h_i^{(l+1)}}$$

Derivative of  $\mathcal{L}$  with respect to  $h_j^{(l)}$  for a layer  $(l)$

$$\frac{\partial \mathcal{L}}{\partial h_j^{(L)}} = 2(y_j - h_j^{(L)})$$

Derivative of  $\mathcal{L}$  with respect to  $h_j^{(L)}$  for the final layer  $(L)$   
(Example with Mean Squared Error loss)

# **Artificial Neural Networks**

Convolutional and Recurrent Neural Networks

CSE2530 – Julia Olkhovskaya

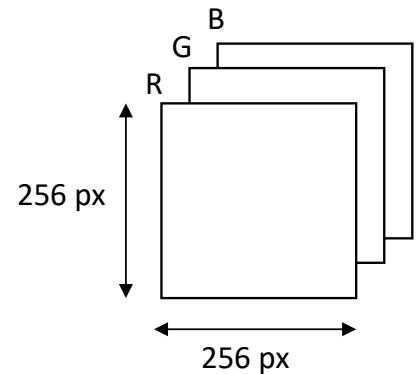
# Content Overview

- Introduction to Convolutional Neural Networks (CNN) and their applications;
- Introduction to Recurrent Neural Networks (RNN) and their applications;
- Intuition behind DALL·E.

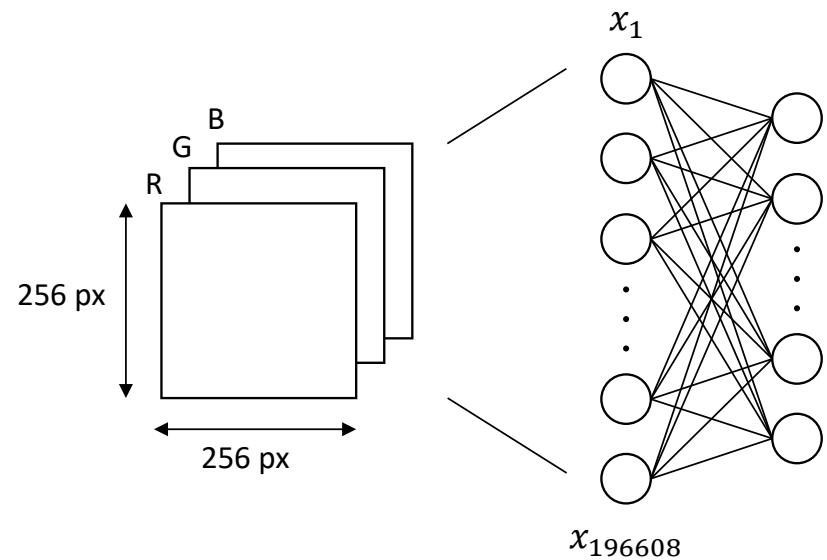
# Material

- Ian Goodfellow - Deep Learning (<https://www.deeplearningbook.org>) (further reading):
  - Chapter 9: sections 9.1 to 9.3
  - Chapter 10: sections 10.1 to 10.2.2
- Anything from Andrew Ng

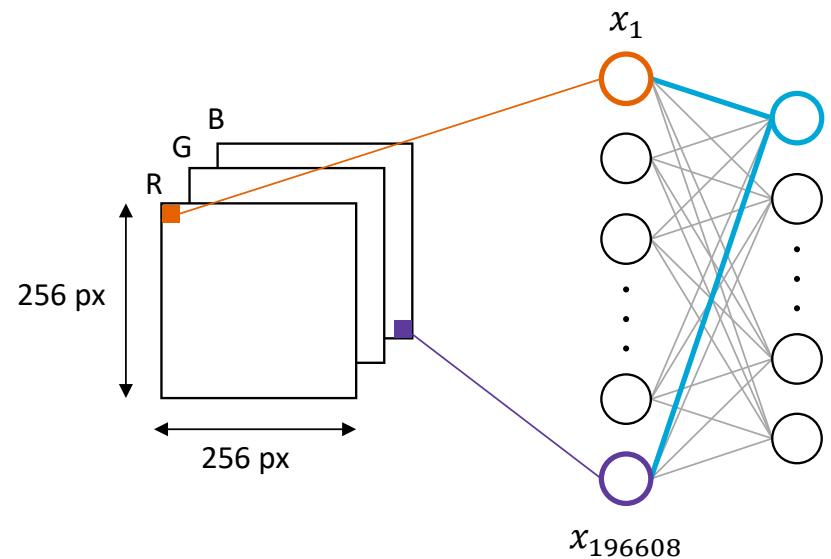
# Convolutional Neural Networks (CNN)



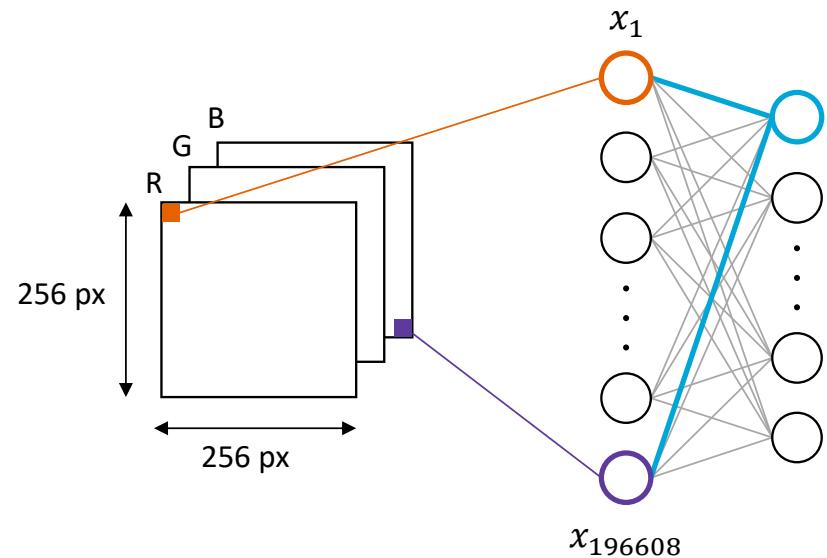
# Convolutional Neural Networks (CNN)



# Convolutional Neural Networks (CNN)



# Convolutional Neural Networks (CNN)



When used on RGB images, the MLP:

- **Scales badly** (increasing the dimensions of the image leads to a quadratic increase of inputs);
- **Ignores the spatial structure** (pixels in opposite parts of the image are connected in the same way as adjacent pixels);
- **Cannot handle translation** (if the network is shown a dog always in a specific corner of the image, it won't recognize it when shown in another part of the image).

# The Convolution Filter

Input image

1	1	1	0	0
0	1	1	0	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

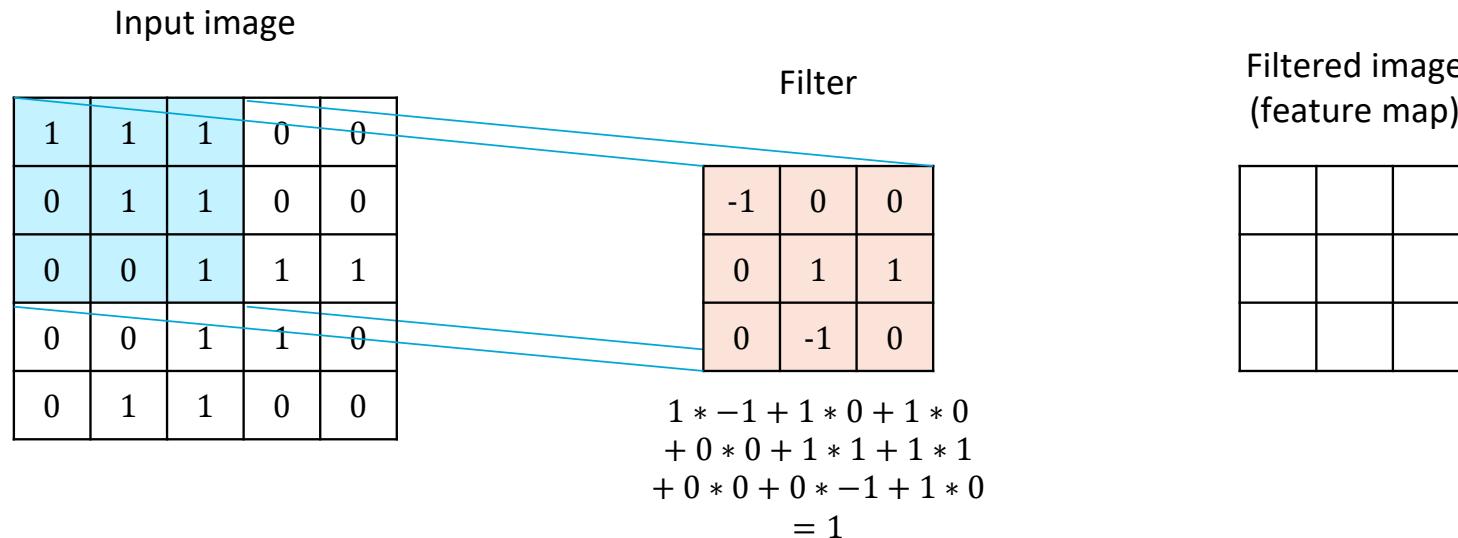
Filter

-1	0	0
0	1	1
0	-1	0

Filtered image  
(feature map)

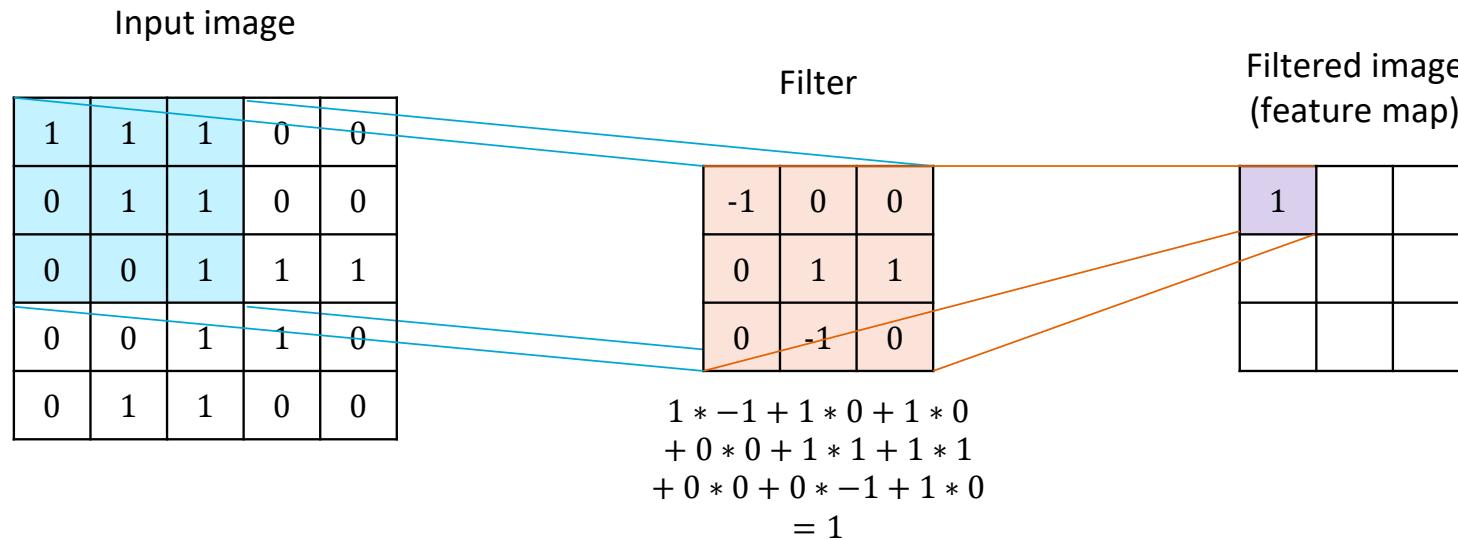

We perform element-wise multiplication between the elements of the filter and the intensity values of the image (e.g., in the case of RGB, the color intensity between 0 and 255) and sum the result, scanning the filter over the whole image.

# The Convolution Filter



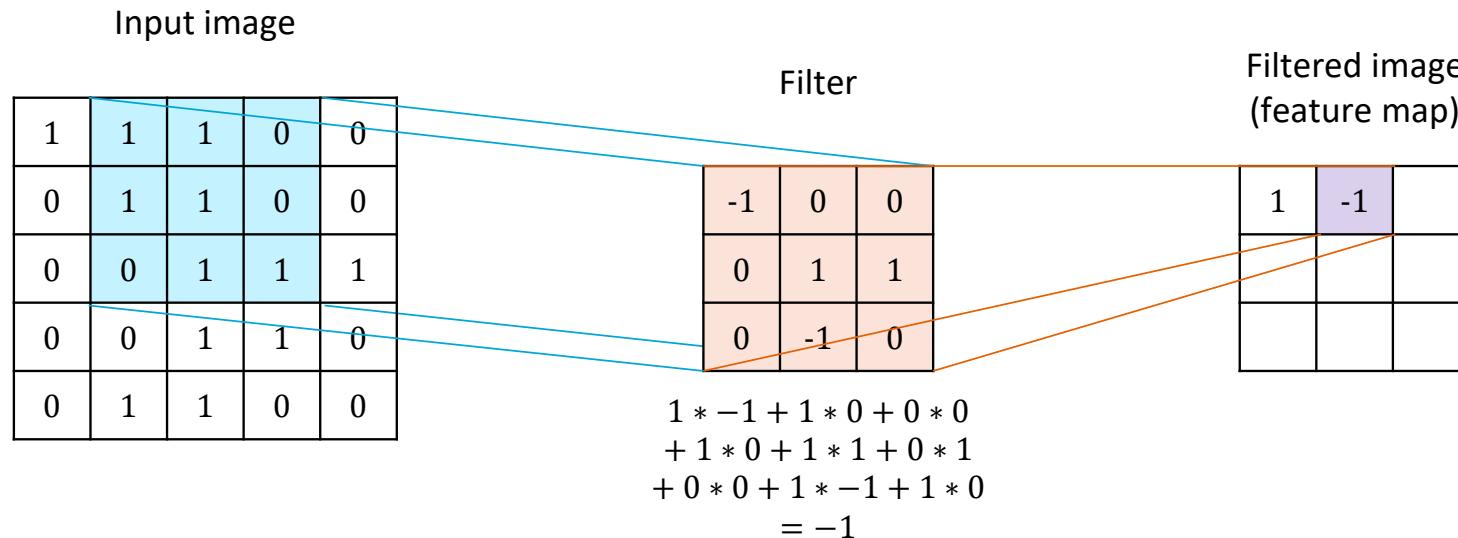
We perform element-wise multiplication between the elements of the filter and the intensity values of the image (e.g., in the case of RGB, the color intensity between 0 and 255) and sum the result, scanning the filter over the whole image.

# The Convolution Filter



We perform element-wise multiplication between the elements of the filter and the intensity values of the image (e.g., in the case of RGB, the color intensity between 0 and 255) and sum the result, scanning the filter over the whole image.

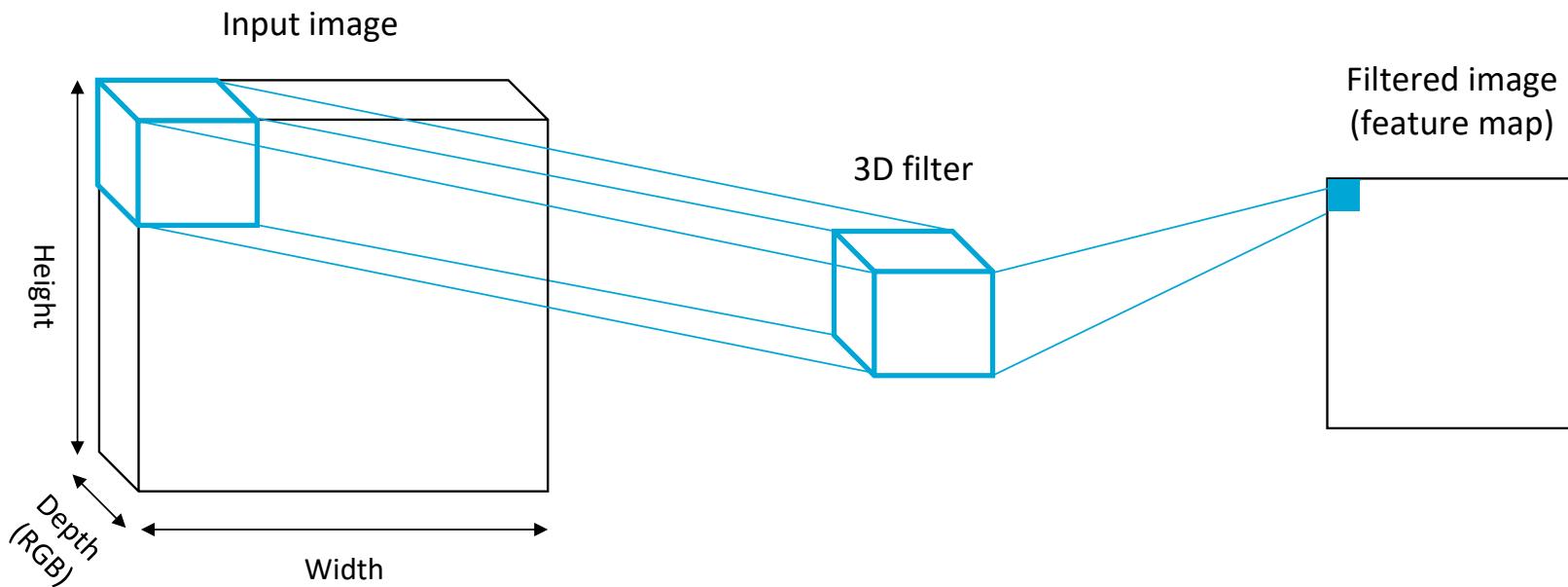
# The Convolution Filter



We perform element-wise multiplication between the elements of the filter and the intensity values of the image (e.g., in the case of RGB, the color intensity between 0 and 255) and sum the result, scanning the filter over the whole image.

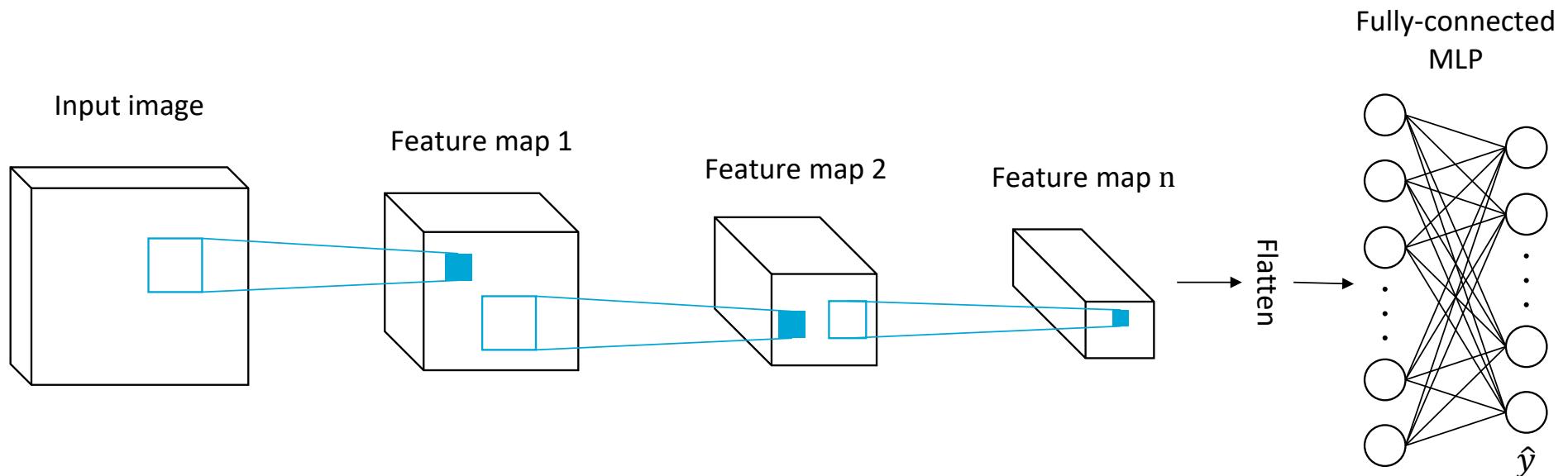
During training, we update the elements of the filter (i.e., the weights of the CNN).

# The 3D Convolution Filter



Convolution filters can be applied on 3D images (e.g., considering the RGB channels as depth of 3).  
The filter still performs element by element multiplication and sums the result to a scalar value.

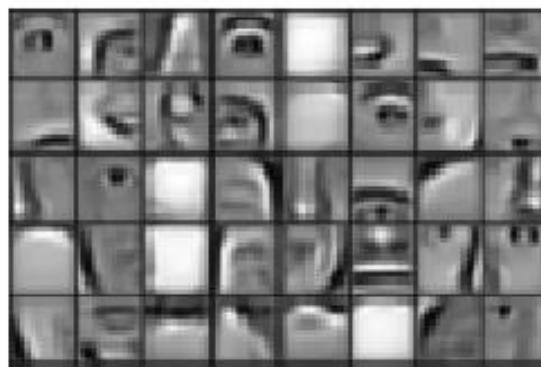
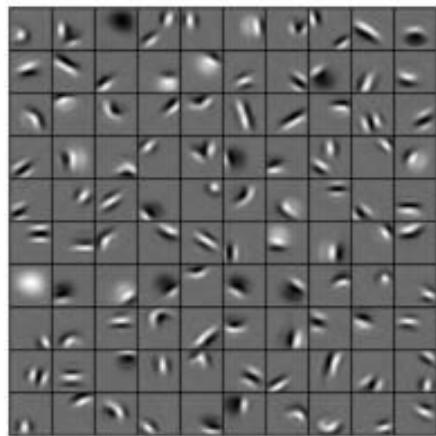
# CNN Architecture



At every layer, an array of multidimensional filters are applied, to reduce the width and height of the feature map while increasing the depth. After the last layer of filters, the feature map is flattened to a 1-dimensional array which is used as input to a fully-connected MLP, whose weights are also learned during training. The output of the MLP is then compared to the desired output.

# Convolution Filters are Feature Detectors

Convolution filters in the first layers detect low level features (e.g., vertical and horizontal lines). In deeper layers, the feature maps represent higher levels concepts (e.g., parts of the face), and in the deepest layers they reconstruct the full image, highlighting the most important features needed for classification (e.g., a moustache).



# CNN Applications

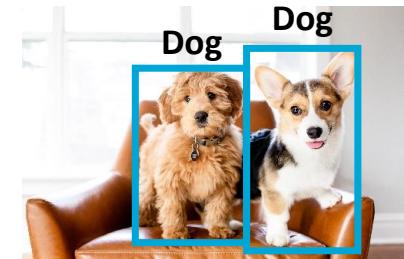
CNN are mostly used for image processing. The main applications are:

- **Image classification:** indicate whether an image contains a class or not. Used for cancer identification, quality control.
- **Object detection:** find an object in an image. Used for foot traffic analysis, inventory management.
- **Object segmentation:** find the pixels corresponding to an object in an image. Used for face recognition, automated object picking, environment scanning in autonomous vehicles.

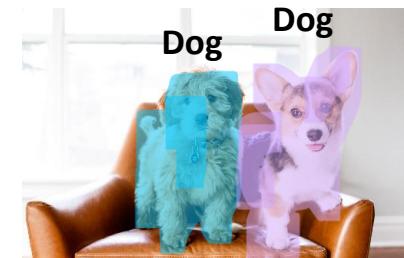
Dog



Dog Dog

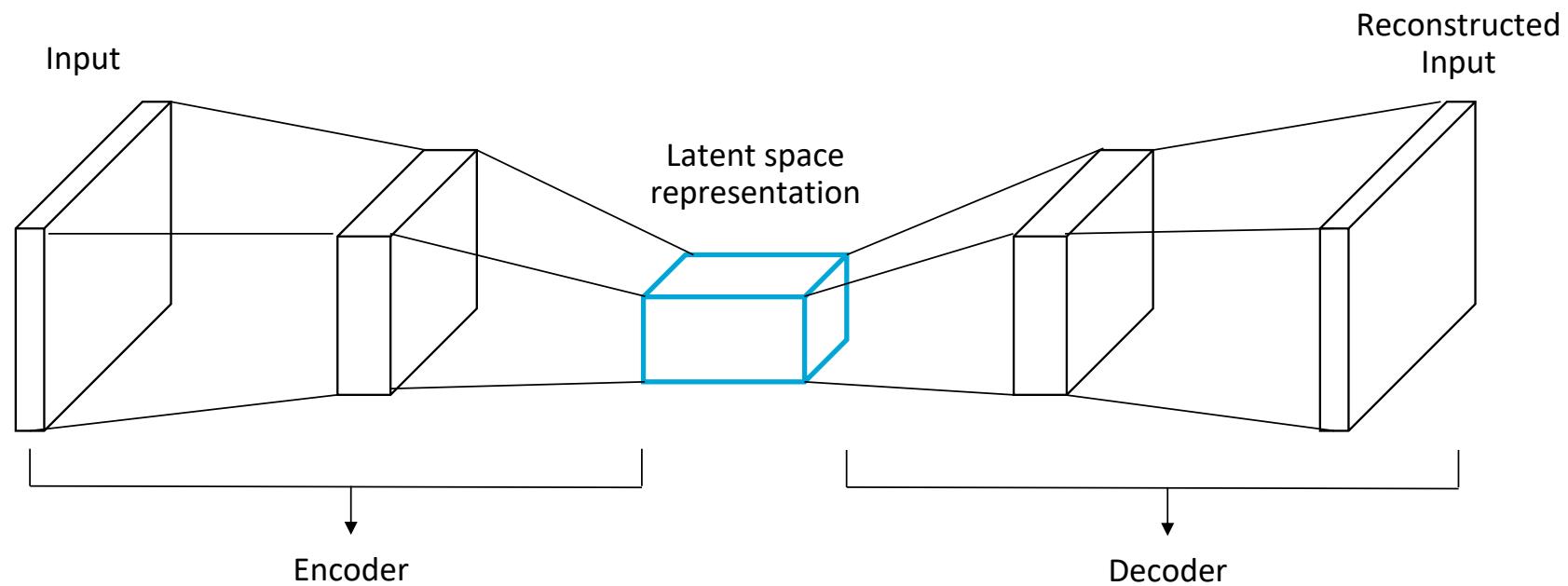


Dog Dog



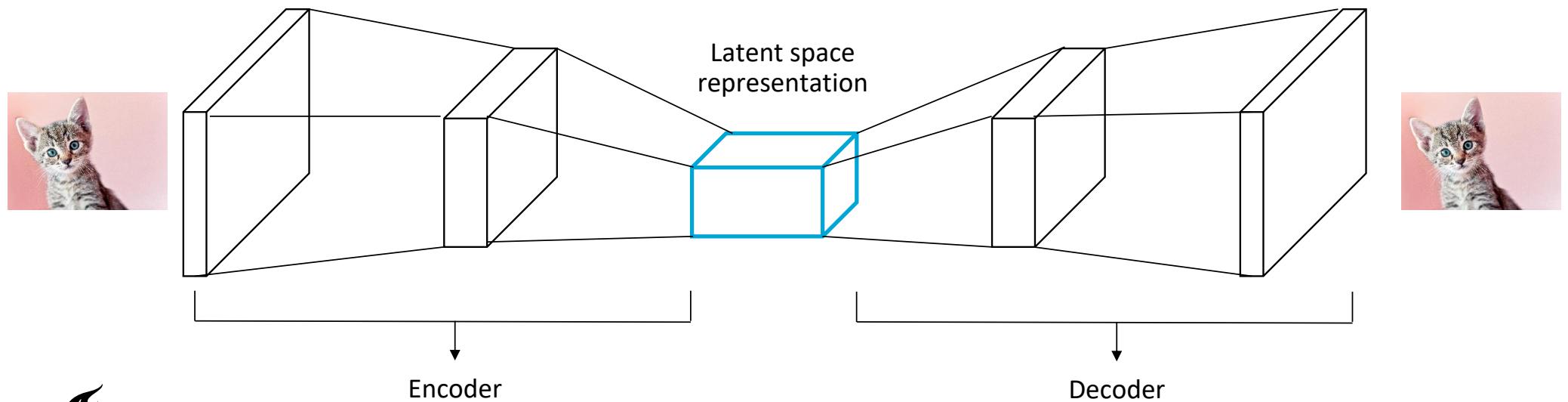
# Encoder-Decoder Architecture

An encoder-decoder architecture (also referred to as autoencoder) is an ANN (any type, including CNN but not only) with a **symmetric structure** and a **bottleneck** in the middle. The encoder-decoder is trained to compress and reconstruct the input information, using the input itself as desired output. In this way, the bottleneck consists of a compressed (encoded) version of the input information, referred to as **latent representation**.



# CNN Applications: Image Reconstruction

A CNN autoencoder is trained with the same image as input and desired output. In this way, the decoder learns to **reconstruct that specific type of image**. When fed with a different encoded image, the decoder will attempt to reconstruct the image as it was trained. For example, it will complete missing parts of an image. Or, if trained with photos of Tom Cruise's face, the decoder will attempt to reconstruct Tom Cruise's face on top of the latent representation of somebody else's face (the technology at the base of deepfakes).



# Recurrent Neural Networks (RNN)

When handling **sequential data** (e.g., text or temperature), the MLP cannot be applied because:

- it requires a fixed input length;
- all inputs are treated equally (there is no notion of proximity in the sequence).

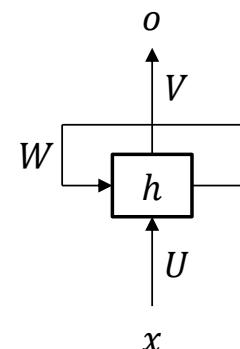
# Recurrent Neural Networks (RNN)

When handling **sequential data** (e.g., text or temperature), the MLP cannot be applied because:

- it requires a fixed input length;
- all inputs are treated equally (there is no notion of proximity in the sequence).

**Recurrent Neural Networks (RNN)** are preferred because:

- Information flows from one step to the following;
- Parameters are shared;
- Internal memory is passed to the next step in the sequence.



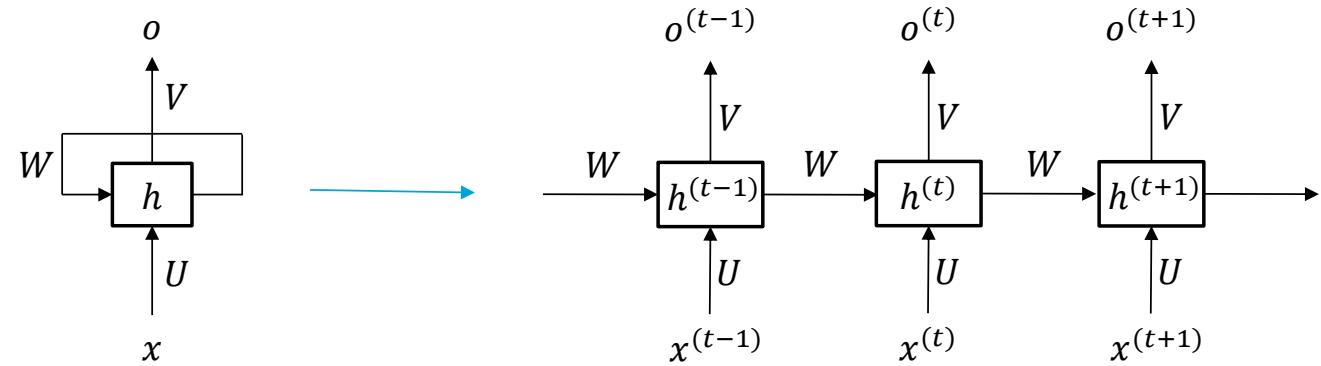
# Recurrent Neural Networks (RNN)

When handling **sequential data** (e.g., text or temperature), the MLP cannot be applied because:

- it requires a fixed input length;
- all inputs are treated equally (there is no notion of proximity in the sequence).

**Recurrent Neural Networks (RNN)** are preferred because:

- Information flows from one step to the following;
- Parameters are shared;
- Internal memory is passed to the next step in the sequence.



# Recurrent Neural Networks (RNN)

When handling **sequential data** (e.g., text or temperature), the MLP cannot be applied because:

- it requires a fixed input length;
- all inputs are treated equally (there is no notion of proximity in the sequence).

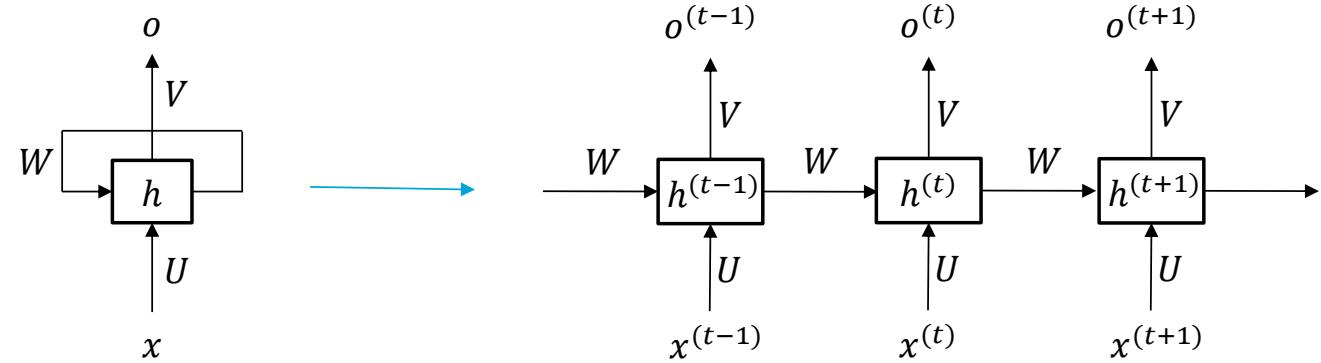
**Recurrent Neural Networks (RNN)** are preferred because:

- Information flows from one step to the following;
- Parameters are shared;
- Internal memory is passed to the next step in the sequence.

$$z^{(t)} = Ux^{(t)} + Wh^{(t-1)} + b$$

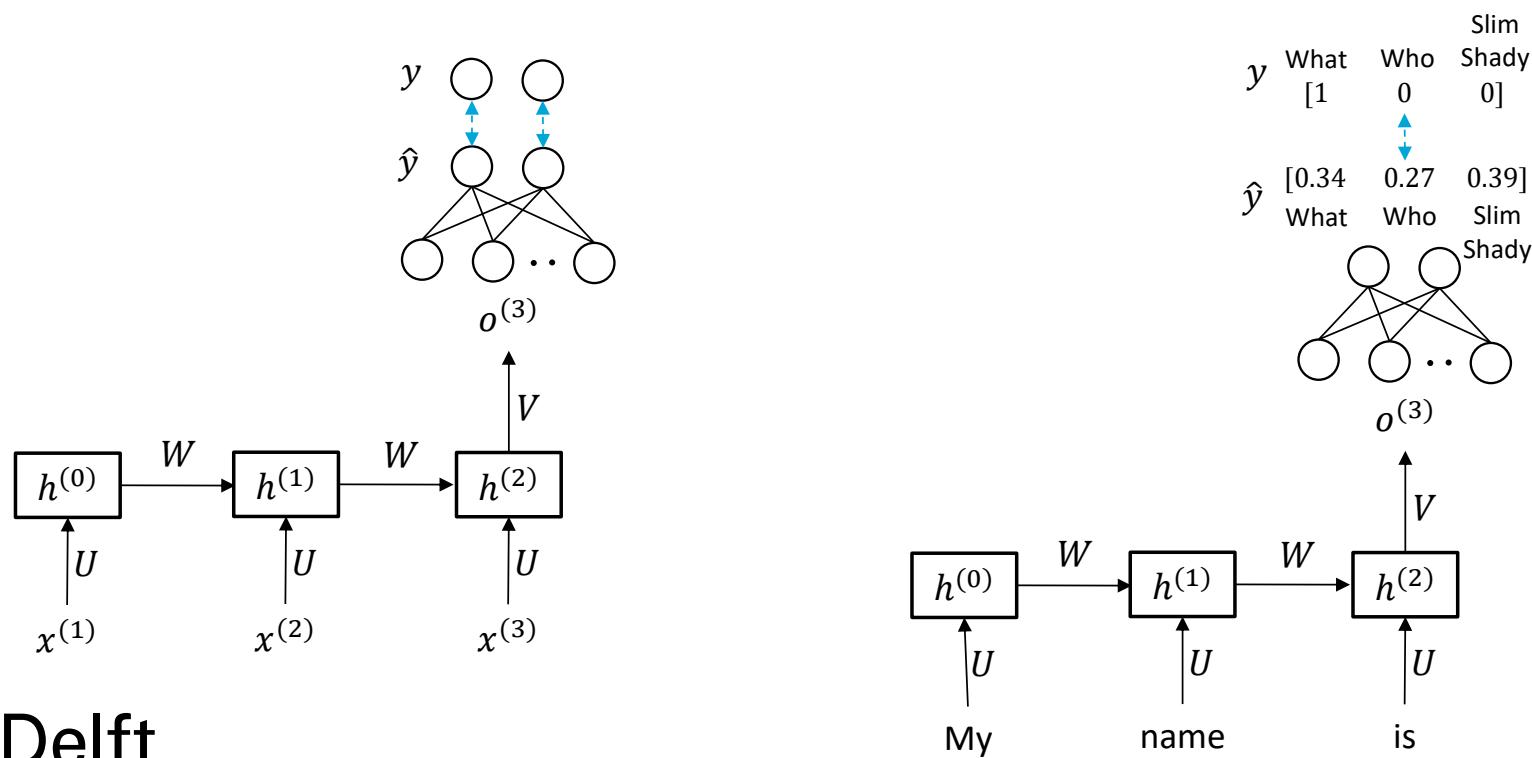
$$h^{(t)} = \sigma(z^{(t)})$$

$$o^{(t)} = Vh^{(t)} + c$$



## Example: Next Word Prediction

The input (e.g., a sentence) is passed sequentially through the RNN. The final output  $o^{(T)}$  contains processed information on the whole sequence.  $o^{(T)}$  is used as the input to a fully-connected MLP, whose output is compared to the desired output  $y$ . For text processing, the output is a probability score for each word available in the dictionary.



# Self-Supervised Learning

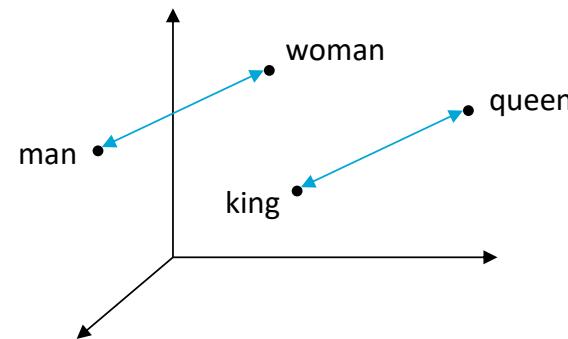
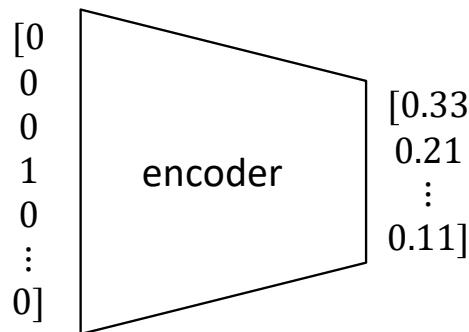
Self-supervised learning consists of supervised learning where no human intervention is required to indicate which is the desired output. Examples of tasks with self-supervised labels are:

- **Image reconstruction:** use the input image as desired output in encoder-decoder architectures (often referred to as autoencoders);
- **Next word prediction:** use the next word in the sentence as desired output for the next word prediction task. This is how Large Language Models are trained on massive datasets;
- **Word embeddings generation:** use word proximity in the sentence to train word embeddings.

# Word Embeddings

The simplest way of representing a word in a numerical way is to create a vector long as the dictionary length, with a 1 in the element index corresponding to the word and all other vector's elements set to 0. However, this generates **sparse vectors** with the length in the order of the tens of thousands.

Word embeddings are instead preferred. An autoencoder learns to map words to an **n-dimensional embedding space** (with n typically between 100 and 1000). The encoder is trained so that co-occurring words (e.g., "Statue" and "Liberty") and words that appear in consistent part of the sentence (e.g., adjectives always preceding nouns) are mapped closely in the embedding space. The nature of the labels (words co-occurrence) makes the process self-supervised. This generates n-dimensional **dense vectors** (with all vector elements ranging between 0 and 1) that **preserve word similarity** in a Euclidian space. The embedded representation of the word is used as input to the RNN.



# RNN Applications

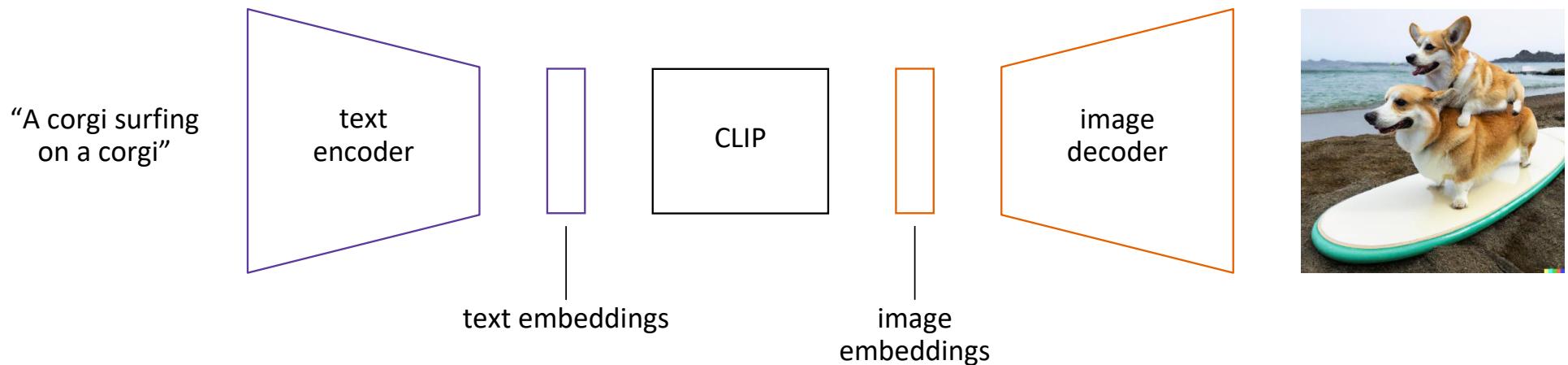
RNN are mostly applied to Natural Language Processing (NLP). The main applications are:

- **Text classification:** indicate whether a piece of text contains a class or not. Used for spam detection, sentiment analysis (e.g., product reviews analysis).
- **Machine translation:** An encoder-decoder is trained with the text in the source language as input, and the translation in the target language as desired output.
- **Chatbot:** the conversation is fed as input to a language generation model, which outputs a reply. Can be trained with supervised learning (trained with examples of other chats) or reinforcement learning (e.g., chatGPT).

# DALL·E

DALL·E is a deep learning model that generates an image based on an arbitrary text description.

It has three components: a **text encoder**, a model that matches **text embeddings to image embeddings** (CLIP), and an **image decoder**. The text encoder and image decoder are independently trained as autoencoders through self-supervised learning. CLIP is trained with pairs of images and associated captions as inputs and desired outputs.

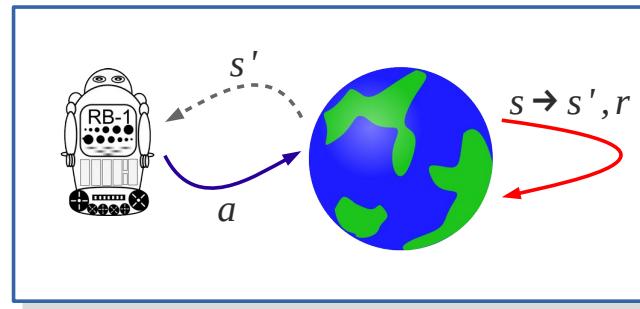


# Ethical Reflections

- ANN and Deep Learning are not magic: they are based on the **data** they are trained with. They learn to map input to output by extrapolating relationships and knowledge from the training data. Thus, they can only be as good as the training data.
- ANN like DALL·E construct enormous internal databases that they use to generate images. However, we do not have access to those databases, and we cannot fully understand the decisions taken by them. **Explainable AI (XAI)** is the field of AI that attempts to interpret the rationale of large-scale ANNs.
- Large-scale ANNs are trained on immense amounts of data, requiring huge computational power and causing a **serious impact on the environment**. However, we have not yet found the limit: it seems that the larger the model and training set sizes, the better the performance.
- We should reflect on our definition of **progress**. Should we keep on training black-box models on more and more data? Or is it time to find an alternative, interpretable, and lighter paradigm?

# Reinforcement Learning

## Part 1



Dr. Frans A. Oliehoek

# Why RL?

- E.g., for tasks that are too complex to program

---

**Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots**

Zhongyu Li, Xuxin Cheng, Xue Bin Peng, Pieter Abbeel, Sergey Levine, Glen Berseth, Koushil Sreenath



Berkeley

hybrid-robotics.berkeley.edu rail.eecs.berkeley.edu rll.berkeley.edu



AlphaGo playing against Lee Se-dol

- <https://www.technologyreview.com/2021/04/08/1022176/boston-dynamics-cassie-robot-walk-reinforcement-learning-ai/>

# Overview of the “RL Topic”

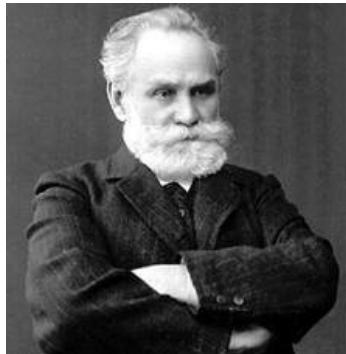
- Intro to **sequential decision making (SDM) & reinforcement learning (RL)**
- Part 1 (today)
  - A formal description of RL: Markov Decision Processes (MDPs)
  - Using MDPs: planning.
- Part 2:
  - Unknown MDPs: reinforcement learning
  - Challenges / Issues that limit RL applicability
- Material:
  - Sutton and Barto (2018) book “Reinforcement Learning: An introduction second edition”:
  - Ch. 1, Ch. 3, 4.1 – 4.4, 5.1 - 5.3, 5.7, 6.1, 6.2, 6.4, 6.5

## Learning Outcomes P1

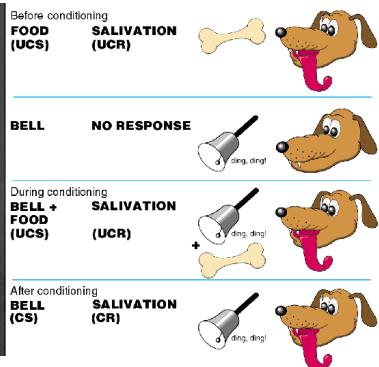
- Explain when sequential decision making is applicable/required for a robot/agent.
- Explain the temporal credit assignment problem
- Given a certain problem, define an MDP that represents it in terms of:
  - States, actions, transitions, rewards, discount
- Explain concepts like: Markov property, absorbing states, value function, etc.
- Apply policy iteration and value iteration to compute a solution to an MDP

# The idea of 'reinforcement'

# Roots: Behavioral psychology & conditioning



Ivan Pavlov

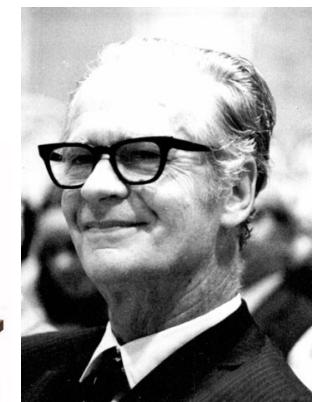
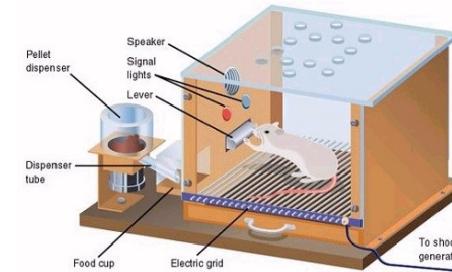


## classical conditioning

- *involuntary* behavior
- pairing of stimuli with biologically significant events.
- Reflexes: automatically elicited by the appropriate stimuli.
- Example: sight of sweets may cause a child to salivate - not voluntarily "chosen".

## operator conditioning

- *voluntary* behavior
- Reinforced by reward/punishment
- Example: a child may learn to open a box to get the sweets inside

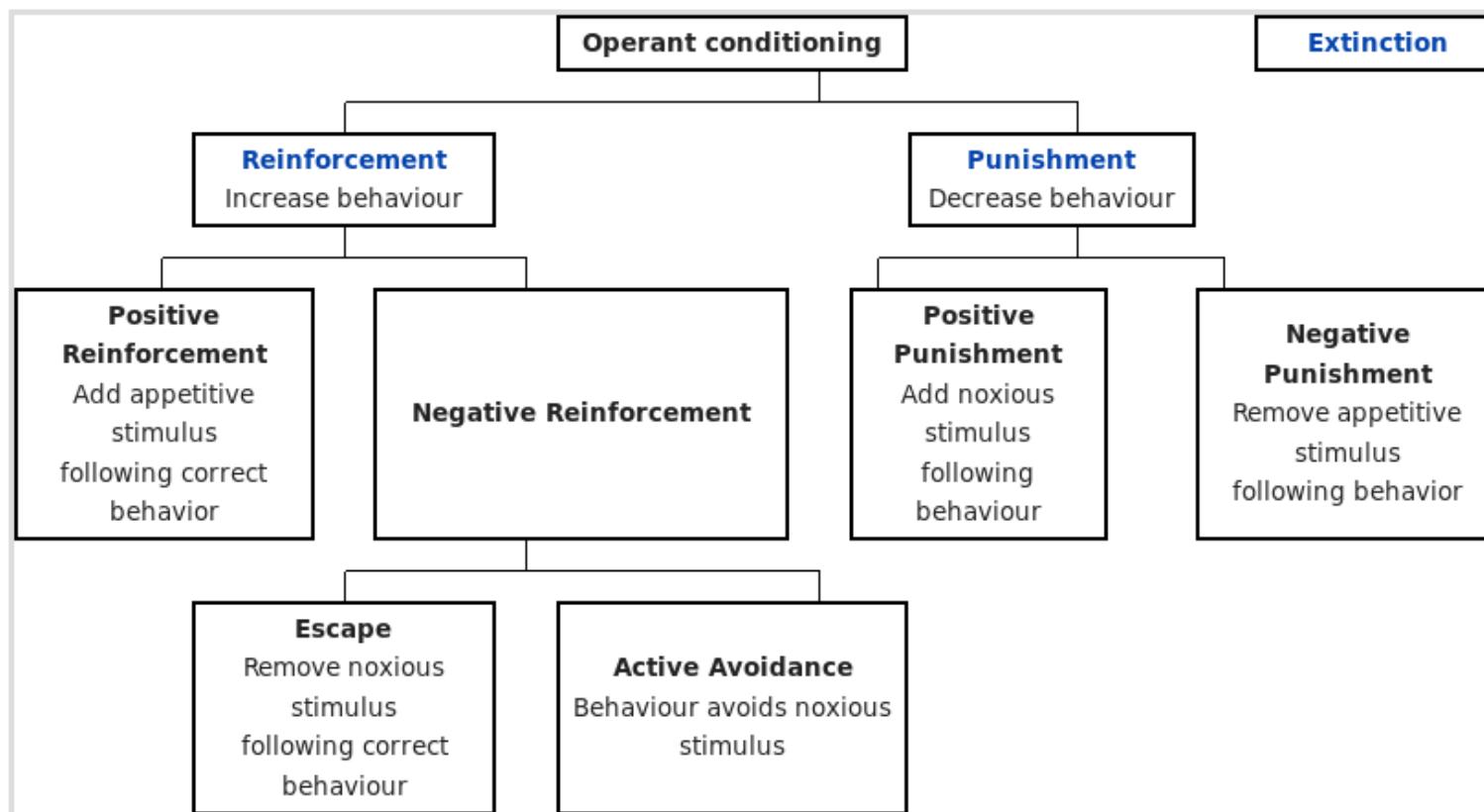


B.F. Skinner

# Types of Reinforcement

## Operant conditioning

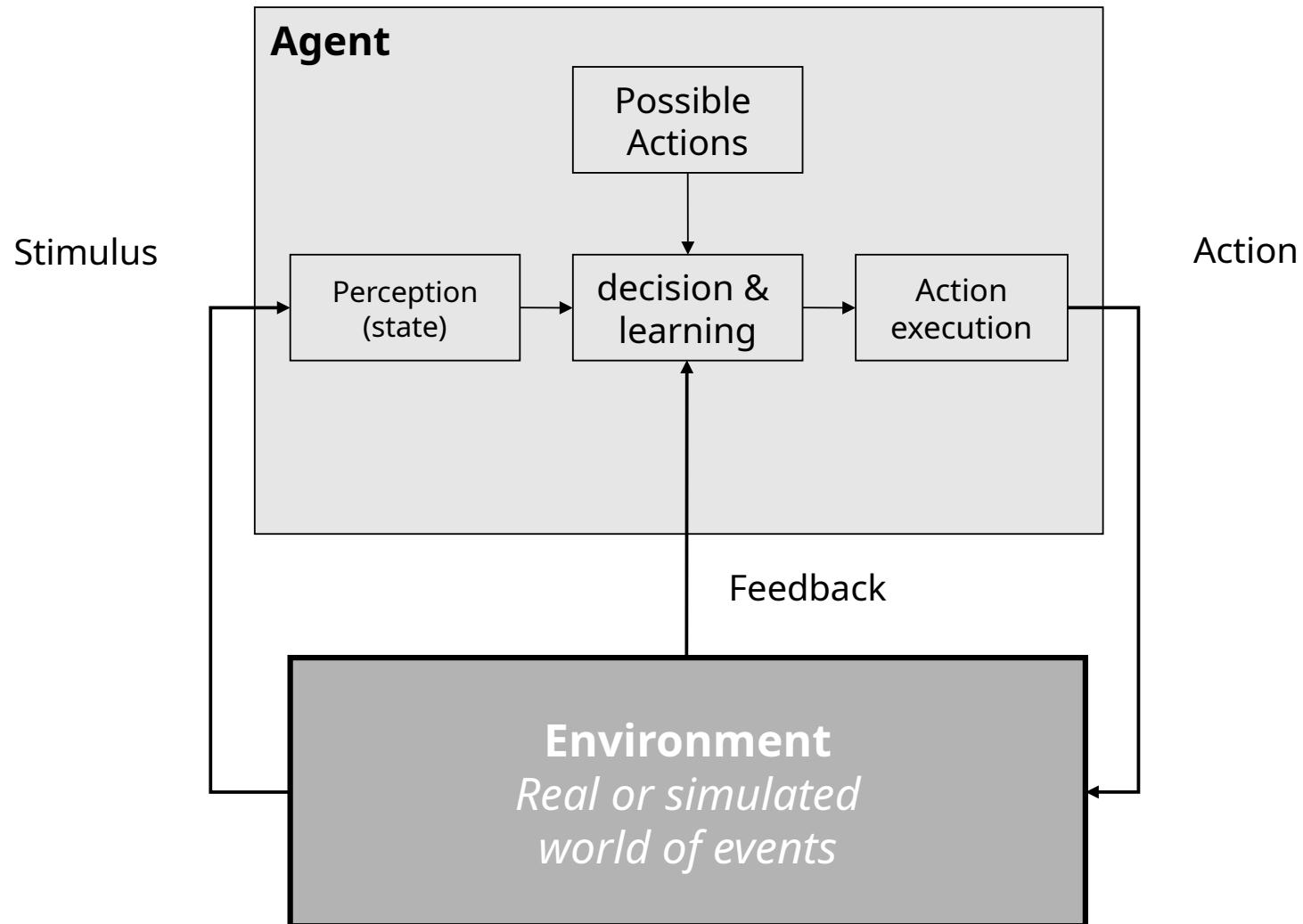
From Wikipedia, the free encyclopedia



We will use  
'reinforcement' to  
mean both  
reward and  
punishment

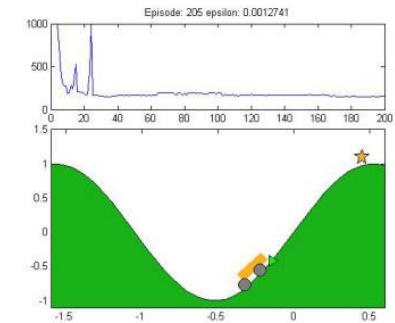
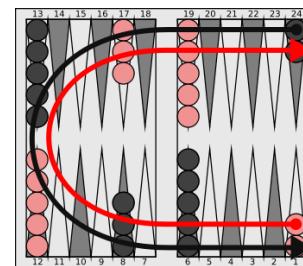
# Sequential Decision Making

# Agents & environment interact over time



# Temporal credit assignment problems

- Robot navigation
  - Car driving
  - Chess /backgammon / etc. playing
  - Mountain car
- 
- Any problem that:
    - needs sequential action selection to be solved
    - has state transitions triggered by agent actions.
    - involves delayed rewards defining task success/failure



# Temporal credit assignment problems

- Robot navigation
  - Car driving
  - Chess /backgammon / etc. playing
  - Mountain car
- 
- Any problem that:
    - needs sequential action selection to be successful
    - has state transitions triggered by agent's actions
    - involves delayed rewards defining task success/failure



“Temporal credit assignment”

→ **which action at what time step** was responsible for the rewards?

E.g.,

- Was the “brake action” to blame when crashing into a tree? Or was it the “overtake action” 10 seconds before on the narrow road?
- Did you pass your exam due to studying hard the day before? Or due to efforts during the entire quarter?

# Quiz

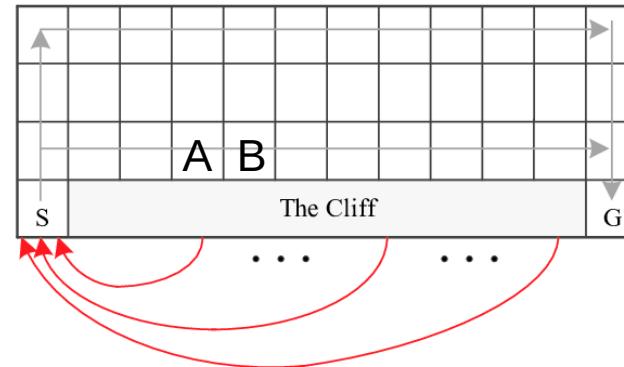
<https://vevox.app>  
session ID: 184-748-254

- Which are temporal credit assignment problems?
  - medical treatment of a patient over time
  - inspection of manufactured parts on a conveyor belt
  - determining the 'best' slot machine (out of 5 in a casino)
  - controlling traffic lights at a busy intersection

- ▶ needs sequential action selection to be solved
- ▶ has state transitions triggered by agent actions.
- ▶ involves delayed rewards defining task success/failure

# Walking along a cliff

- We want to get from S to G...
  - ...don't want to fall in the cliff
  - ...want to get there fast
  - ...but our movements are noisy



E.g., action 'right' may succeed with probability 0.9.  
Rest of time: effect is up or down.

Then:

$$p(B \mid \text{Right}, A) = .9$$

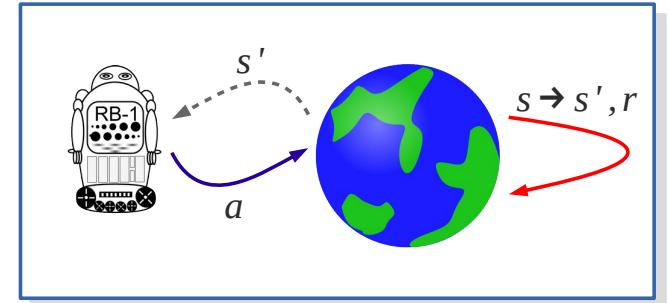
$$p(\text{Cliff} \mid \text{Right}, A) = .05$$

So...  
what route  
should we  
take?

# A Formal Model: The Markov Decision Process

# The Markov decision process (MDP)

- An MDP  $M = \langle S, A, T, R \rangle$
- $S$  – set of world states  $s$
- $A$  – set of actions  $a$
- $T$  – transition function
  - specifies  $p(s'|s, a)$
  - enables: outcome uncertainty
- $R$  – reward function
  - Task encoded by rewards  $R(s, a, s')$ 
    - also  $R(s, a)$  or  $R(s')$
- Agent wants to optimize **sum** of rewards



# Some notation and comments

- Reward function  $R(s,a,s')$ 
  - The definition of good/bad for all actions in all states.
  - In many problems reward is **delayed** and/or **sparse**:
    - only once you get to the goal / win a match, etc.
    - sparse: many  $R(s,a,s')=0$
- State transition function  $T(s,a,s')$ 
  - Defines the world's reactions to the agent's actions
  - **Markov assumption**:
    - Effect of actions depends only on *current* state
    - $P(s_t | s_{t-1} a_{t-1} s_{t-2} a_{t-2} \dots) = P(s_t | s_{t-1} a_{t-1})$

notation:  
 $R(s, a, s') = r$   
 $T(s,a,s') = P(s'|a,s)$

# Some notation and comments

- Reward function  $R(s,a,s')$ 
  - The definition of good/bad for all actions in all states.
  - In many problems reward is **delayed** and/or **sparse**:
    - only once you get to the goal / win a match, etc.
    - sparse: many  $R(s,a,s')=0$
- State transition function  $T(s,a,s')$ 
  - Defines the world's reactions to the agent's actions
  - **Markov assumption:**
    - Effect of actions depends only on *current* state
    - $P(s_t | s_{t-1} a_{t-1} s_{t-2} a_{t-2} \dots) = P(s_t | s_{t-1} a_{t-1})$

notation:  
 $R(s, a, s') = r$   
 $T(s,a,s') = P(s'|a,s)$

It is a very strong assumption!

→ state has enough information to predict the future

i.e.,

- everything I need to know can be observed
- do not need to remember anything...

How about this?

# Some notation and comments

- Reward function  $R(s,a,s')$ 
  - The definition of good/bad for all actions in all states.
  - In many problems reward is **delayed** and/or **sparse**:
    - only once you get to the goal / win a match, etc.
    - sparse: many  $R(s,a,s')=0$
- State transition function  $T(s,a,s')$ 
  - Defines the world's reactions to the agent's actions
  - **Markov assumption**:
    - Effect of actions depends only on *current state*
    - $P(s_t | s_{t-1} a_{t-1} s_{t-2} a_{t-2} \dots) = P(s_t | s_{t-1} a_{t-1})$

notation:  
 $R(s, a, s') = r$   
 $T(s,a,s') = P(s'|a,s)$

Sutton&Barto V2:

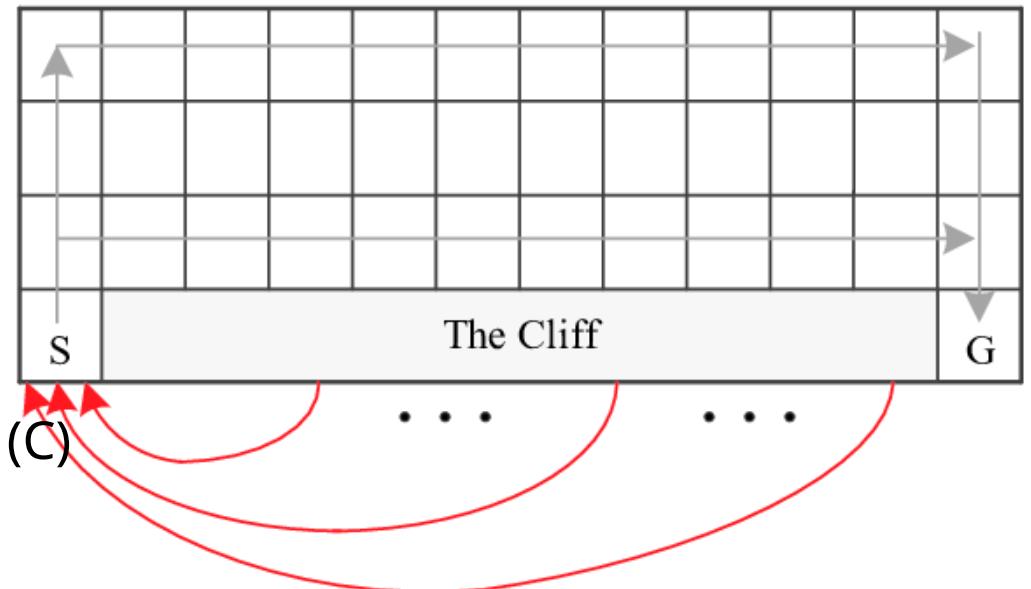
combine both into a single  
'dynamics function'

$$p(s',r|s,a)$$

- makes explicit stochastic rewards
- can convert this to deterministic reward function: take expectation (cf. p49 SBv2)

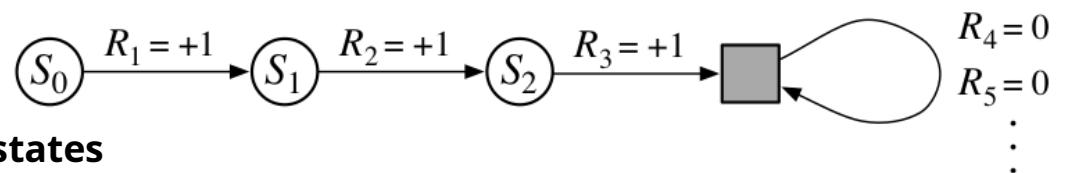
## Example: walk the cliff

- Actions: Up, Down, Left, Right
- States: Location
- Rewards:
  - 10 for getting to the goal (G)
  - -100 for walking/falling into the cliff (C)
- Starting location (S)
- Terminal States
  - G: Agent is 'reset' at S
  - C: Agent is 'reset' at S



# Episodic vs Continuing Tasks

- Smart traffic light should continue forever  
→ ‘continuing’
- But Cliff Walking has a goal  
→ naturally splits in **episodes** ('episodic')
- Why discriminate?
  - next episode is independent of how previous ended
  - how good the last action of an episode was does not depend on the first actions and rewards of the next episode
- Want treat in a unified way...
  - sometimes convenient to think of **absorbing states**



# Using a \*given\* MDP: Planning

# Planning to walk the cliff

- Possible action sequences:
  - URRRRRRRRRRRD
  - UUDUDUDRRRRRRRRRRD
  - UUUURRRRRRRRRRRRDDDD
  - R
  - URD
- What's the optimal one?
- What gives the highest reward?
- Here we first talk about "planning"
  - using a given MDP to **compute** a policy
  - S&B (chap.4) says "dynamic programming"



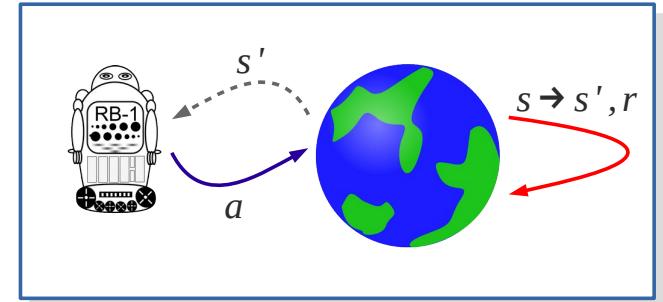
Figure 3: Cliff Walking scenario. Slippery path marked in grey (see text). The red box indicates the distance 4,3,2 and 1 to the Cliff depicted on the horizontal axis of figure 4.

# Goal

- Goal: optimize **long-term** (sum of) rewards
- Formally “optimize sum of reward in an episode” called **return**:  
$$G_t = R_{t+1} + R_{t+2} + \dots + R_T$$
- But what for a continuing task? → return is infinite
  - fix using discount factor  $\gamma$  in  $[0,1)$
  - **discounted return**:  
$$G_t = R_t + \gamma R_{t+2} + \gamma^2 R_{t+3} \dots$$
- But the world can be stochastic...?
  - yes... need to optimize **expected (discounted) return**:  
$$v_\pi(s) = E_\pi [ G_t | S_t=s ]$$
  
when acting according to policy  $\pi$
  - $v_\pi(s)$  is called the **value function** of  $\pi$

# Policies

- So how should such policies  $\pi$  look like...?
  - they could depend on histories...?
  - they could randomize...?
- For a discounted reward, infinite horizon MDP\*\*  
it is a **mapping from state to action**:  $\pi(s) = a$
- This means a policy  
is a **feedback plan**
  - not an 'open loop' plan



\*\*under some technical conditions

# Quiz

<https://vevox.app>  
session ID: 184-748-254

- Which are correct?
  - Sometimes an open loop plan can get higher value than a feedback plan
  - The best feedback plan can always achieve higher value than the best open loop plan
  - Good feedback plans are more difficult to compute than open loop plans
  - A feedback plan is only more useful when the measurements (i.e. observations) are sufficiently informative

## Computing the Optimal Policy

- Generally 2 steps:
  - policy evaluation: compute  $v_\pi(s)$
  - policy improvement: update  $\pi \rightarrow \pi'$
- By alternating these, converge to optimal policy  $\pi^*$

## So what is the value of a particular policy?

- How can we compute  $v_\pi(s) = E_\pi [ G_t \mid S_t=s ]$  ?
- It satisfies the **Bellman equation**:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) [r + \gamma v_\pi(s')]$$

- How to solve? (do “policy evaluation”)
  - system of  $|S|$  equations, with  $|S|$  unknowns.
  - iterative policy iteration

# So what is the value of a particular policy?

- How can we compute  $v_{\pi}(s) = E_{\pi} [ G_t \mid S_t=s ]$  ?

**Different forms of the Bellman equation.**

- It says

$$v_{\pi}(s) = \sum_a \pi(a|s) \sum_{s', r} p(s'|s, a)[r + \gamma v_{\pi}(s')] \quad \text{SBv2}$$

$$v_{\pi}(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s, a)[r(s, a, s') + \gamma v_{\pi}(s')] \quad r(s, a, s')$$

- How

$$v_{\pi}(s) = \sum_a \pi(a|s)[r(s, a) + \sum_{s'} p(s'|s, a) \gamma v_{\pi}(s')] \quad r(s, a)$$

- so

$$v_{\pi}(s) = r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_{\pi}(s') \quad \text{determ. policy}$$

- if

all of these lead to the same result.

# Break

- break time

## Computing the Optimal Policy

- Generally 2 steps:
  - policy evaluation: compute  $v_\pi(s)$
  - policy improvement: update  $\pi \rightarrow \pi'$
- By alternating these, converge to optimal policy  $\pi^*$

# Iterative Policy Evaluation (IPE)

- Given that we know what the value function *is*,  
**how can we compute it?**

# Iterative Policy Evaluation (IPE)

- Given that we know what the value function *is*,  
**how can we compute it?**
- Steps:
  - 1. fix the policy  $\pi$  to evaluate (let's assume deterministic)
  - 2. Take the Bellman equation...

$$v_\pi(s) = r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_\pi(s')$$

... and turn it into an update equation:

$$v_{k+1}(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_k(s')$$

← use variant  
that is most  
convenient

- 3. Initialize,  $v_0(s)=0$ , for all  $s$
- 4. Apply the update equation, in iterations, to all states

# Policy Evaluation Example

- A little maze:

- transitions:

N,E,S,W, deterministic movements

$$P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$$

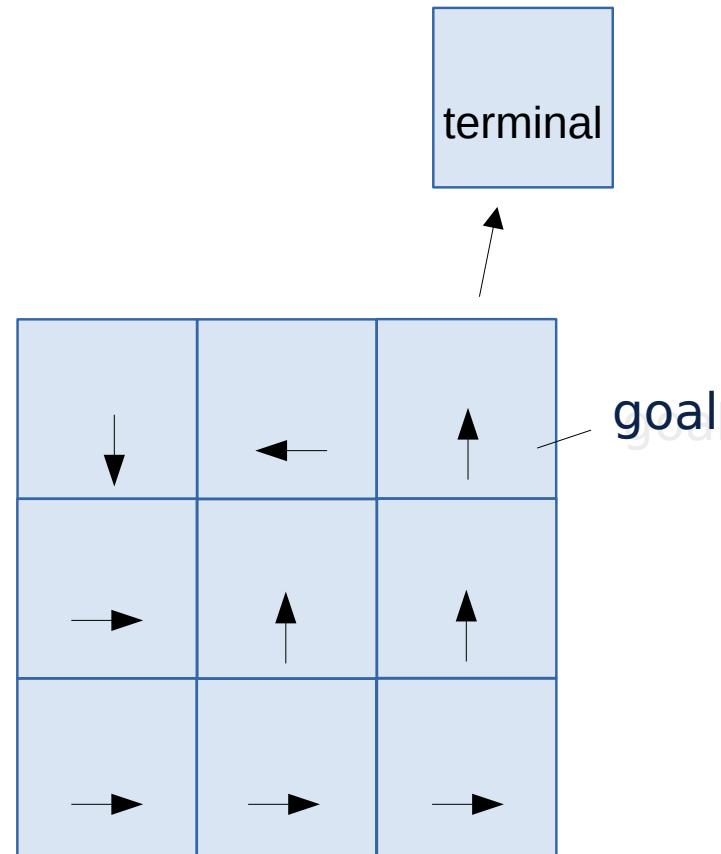
- reward:

$$R(s=\text{Goal}, a=*) = +10$$

$$R(s=\text{terminal}, a=*) = 0$$

$$R(s=*, a=*) = -1 \quad (\text{otherwise})$$

- discount  $\gamma=0.9$



# Policy Evaluation Example

- A little maze:

- transitions:

N,E,S,W, deterministic movements

$$P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$$

- reward:

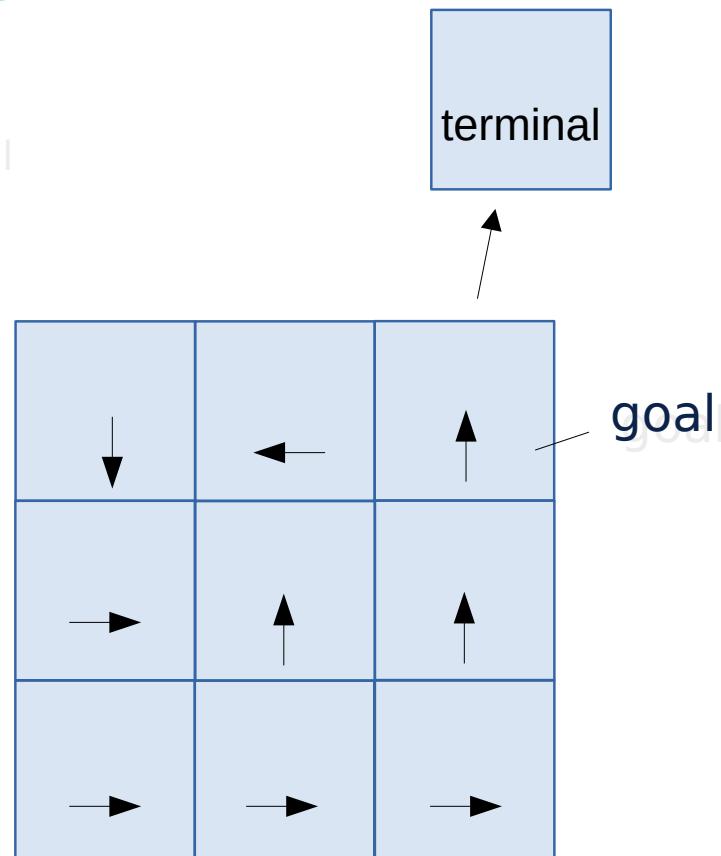
$$R(s=\text{Goal}, a=*) = +10$$

$$R(s=\text{terminal}, a=*) = 0$$

$$R(s=*, a=*) = -1 \quad (\text{otherwise})$$

- discount  $\gamma=0.9$

when hitting a wall  
we stay in place



# Policy Evaluation Example

- A little maze:

- transitions:

N,E,S,W, deterministic movements

$$P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$$

- reward:

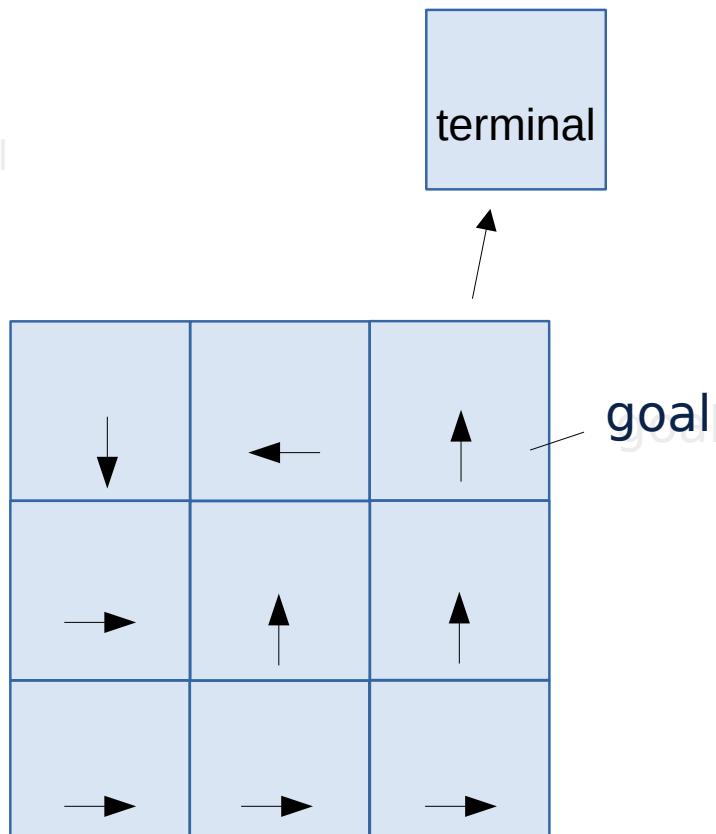
$$R(s=\text{Goal}, a=*) = +10$$

$$R(s=\text{terminal}, a=*) = 0$$

$$R(s=*, a=*) = -1 \quad (\text{otherwise})$$

- discount  $\gamma=0.9$

when hitting a wall  
we stay in place



First step...?

# Policy Evaluation Example

- A little maze:

- transitions:

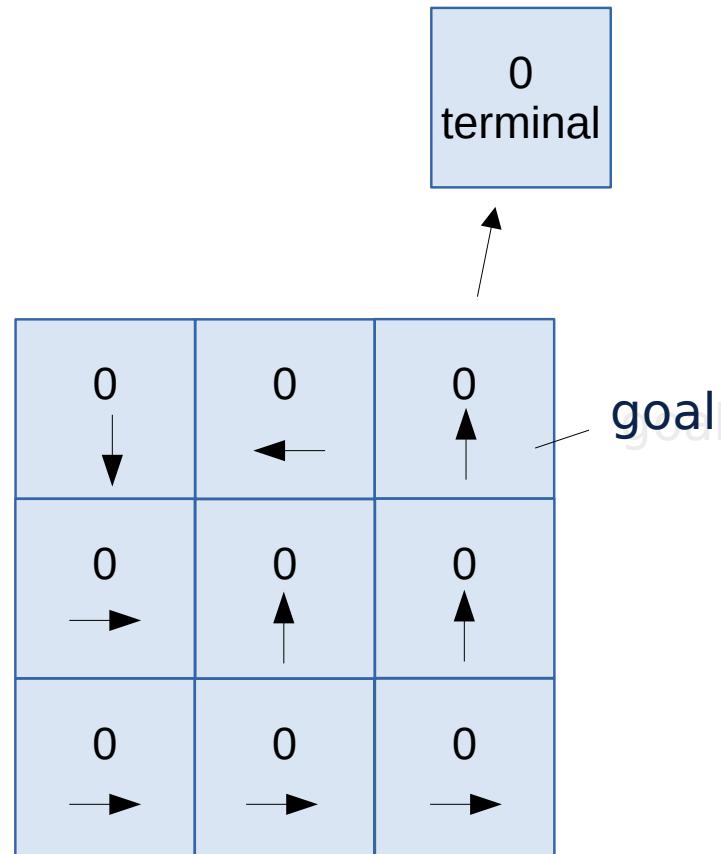
N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$

- reward:

$R(s=\text{Goal}, a=*) = +10$   
 $R(s=\text{terminal}, a=*) = 0$   
 $R(s=*, a=*) = -1$  (otherwise)

- discount  $\gamma=0.9$

Initialize  $v_0(s) = 0$



# Policy Evaluation Example

- A little maze:

- transitions:

N,E,S,W, deterministic movements

$P(s' = \text{terminal} | s=\text{goal}, a=*)=1$

- reward:

$R(s=\text{Goal}, a=*) = +10$

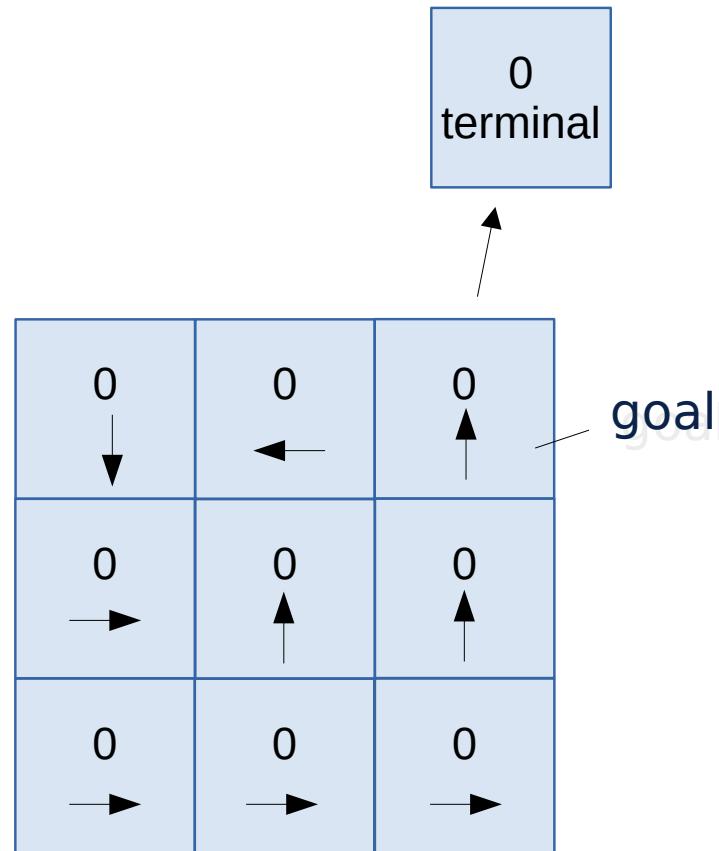
$R(s=\text{terminal}, a=*) = 0$

$R(s=*, a=*) = -1$  (otherwise)

- discount  $\gamma=0.9$

Let's start  
updating...

$$v_1(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_0(s')$$



# Policy Evaluation Example

$$v_1(\text{terminal}) = \dots ?$$

- A little maze:

- transitions:

N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} | s=\text{goal}, a=*)=1$

- reward:

$$R(s=\text{Goal}, a=*) = +10$$

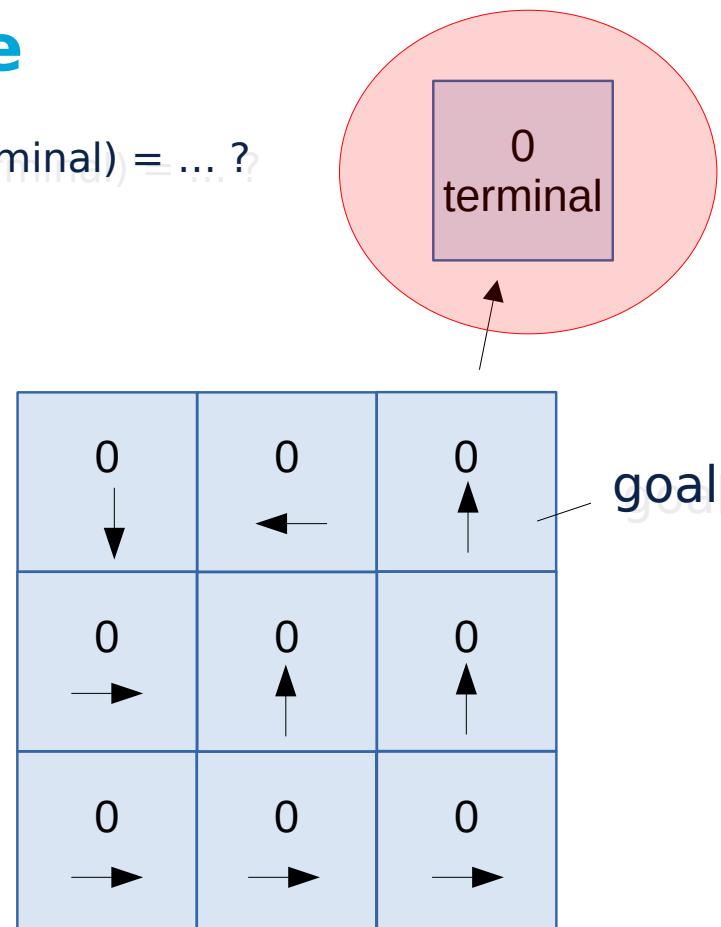
$$R(s=\text{terminal}, a=*) = 0$$

$$R(s=*, a=*) = -1 \quad (\text{otherwise})$$

- discount  $\gamma=0.9$

Let's start  
updating...

$$v_1(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_0(s')$$



# Policy Evaluation Example

$$v_1(\text{terminal}) = 0$$

- A little maze:

- transitions:

N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} | s=\text{goal}, a=*)=1$

- reward:

$R(s=\text{Goal}, a=*) = +10$

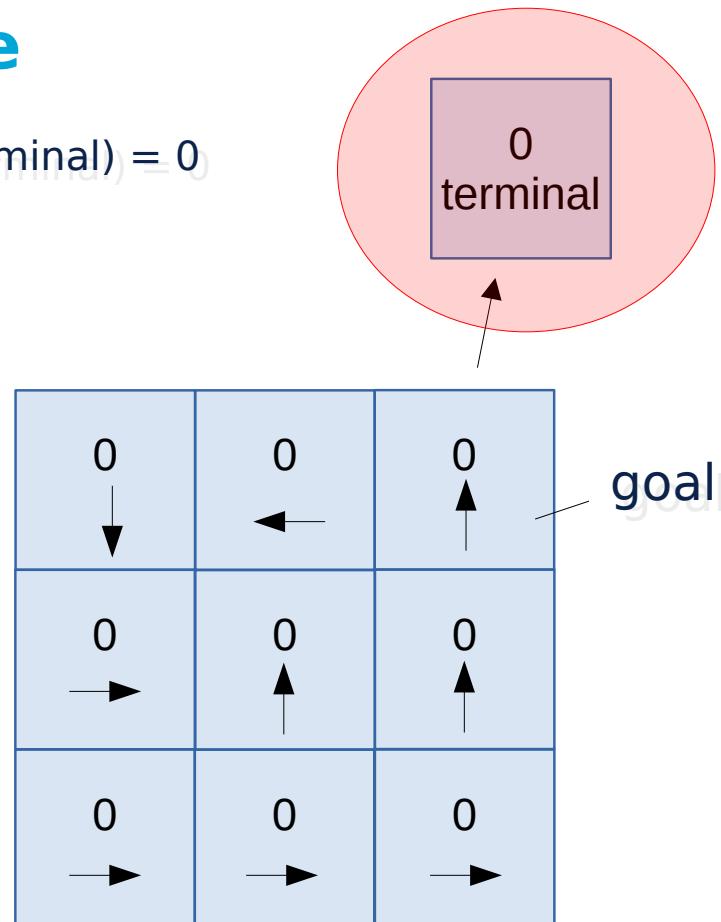
$R(s=\text{terminal}, a=*) = 0$

$R(s=*, a=*) = -1$  (otherwise)

- discount  $\gamma=0.9$

Let's start  
updating...

$$v_1(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_0(s')$$



# Policy Evaluation Example

$$v_1(\text{goal}) = \dots?$$

- A little maze:

- transitions:

N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$

- reward:

$R(s=\text{Goal}, a=*) = +10$

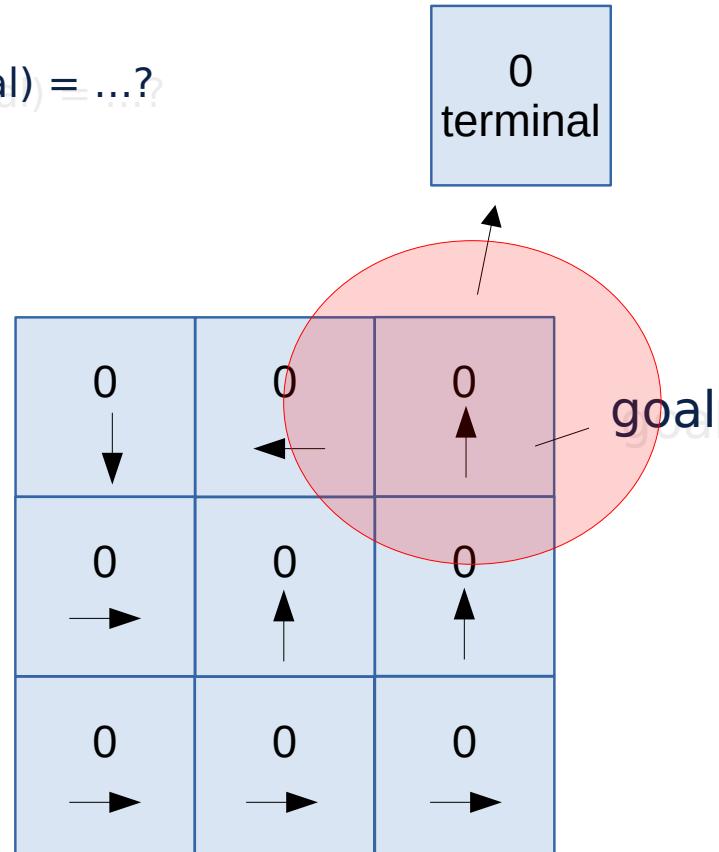
$R(s=\text{terminal}, a=*) = 0$

$R(s=*, a=*) = -1$  (otherwise)

- discount  $\gamma=0.9$

Let's start  
updating...

$$v_1(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_0(s')$$



# Policy Evaluation Example

$$v_1(\text{goal}) = 10$$

- A little maze:

- transitions:

N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} | s=\text{goal}, a=*)=1$

- reward:

$R(s=\text{Goal}, a=*) = +10$

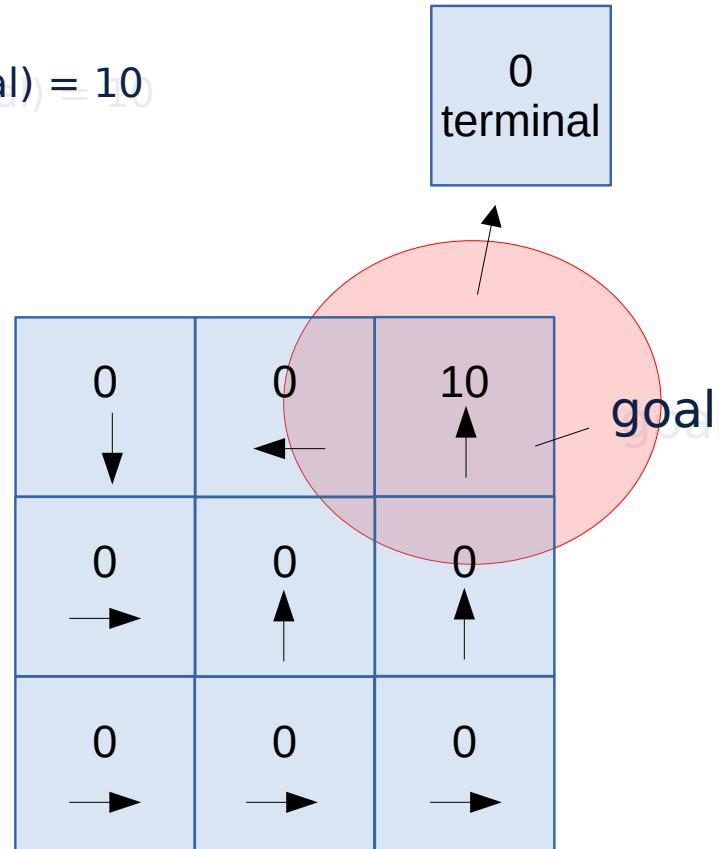
$R(s=\text{terminal}, a=*) = 0$

$R(s=*, a=*) = -1$  (otherwise)

- discount  $\gamma=0.9$

Let's start  
updating...

$$v_1(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_0(s')$$



# Policy Evaluation Example

- A little maze:

- transitions:

N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$

- reward:

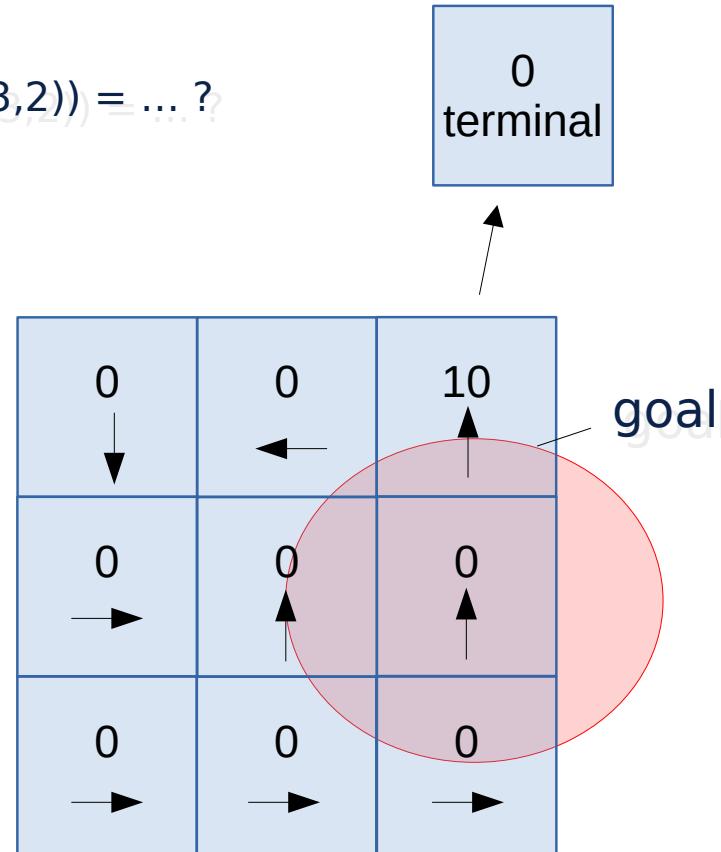
$R(s=\text{Goal}, a=*) = +10$   
 $R(s=\text{terminal}, a=*) = 0$   
 $R(s=*, a=*) = -1$  (otherwise)

- discount  $\gamma=0.9$

Let's start  
updating...

$$v_1(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_0(s')$$

$$v_1((3,2)) = \dots ?$$



# Policy Evaluation Example

- A little maze:

- transitions:

N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$

- reward:

$R(s=\text{Goal}, a=*) = +10$   
 $R(s=\text{terminal}, a=*) = 0$   
 $R(s=*, a=*) = -1 \quad (\text{otherwise})$

- discount  $\gamma=0.9$

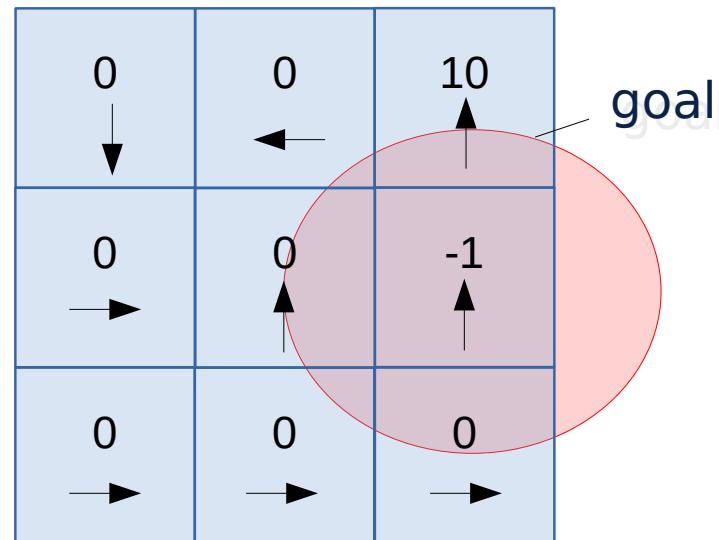
Let's start  
updating...

$$v_1(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_0(s')$$

$$v_1((3,2)) = -1$$

0  
terminal

**NOTE: the updates  
are 'in parallel'**



# Policy Evaluation Example

- A little maze:

- transitions:

N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$

- reward:

$R(s=\text{Goal}, a=*) = +10$

$R(s=\text{terminal}, a=*) = 0$

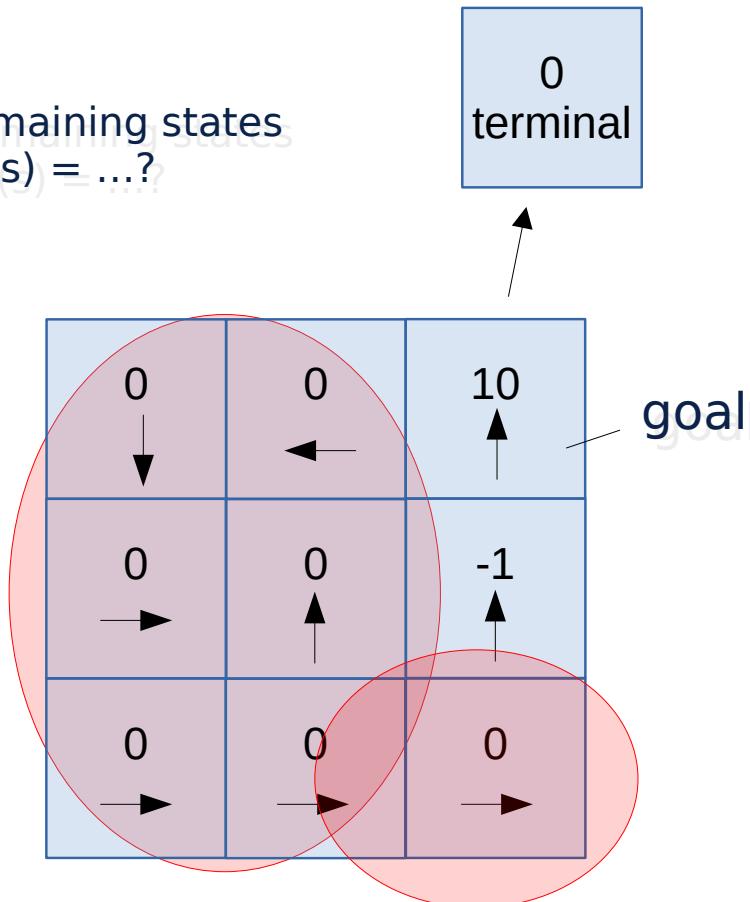
$R(s=*, a=*) = -1$  (otherwise)

- discount  $\gamma=0.9$

Let's start  
updating...

$$v_1(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_0(s')$$

remaining states  
 $v_1(s) = \dots ?$



# Policy Evaluation Example

- A little maze:

- transitions:

N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$

- reward:

$$R(s=\text{Goal}, a=*) = +10$$

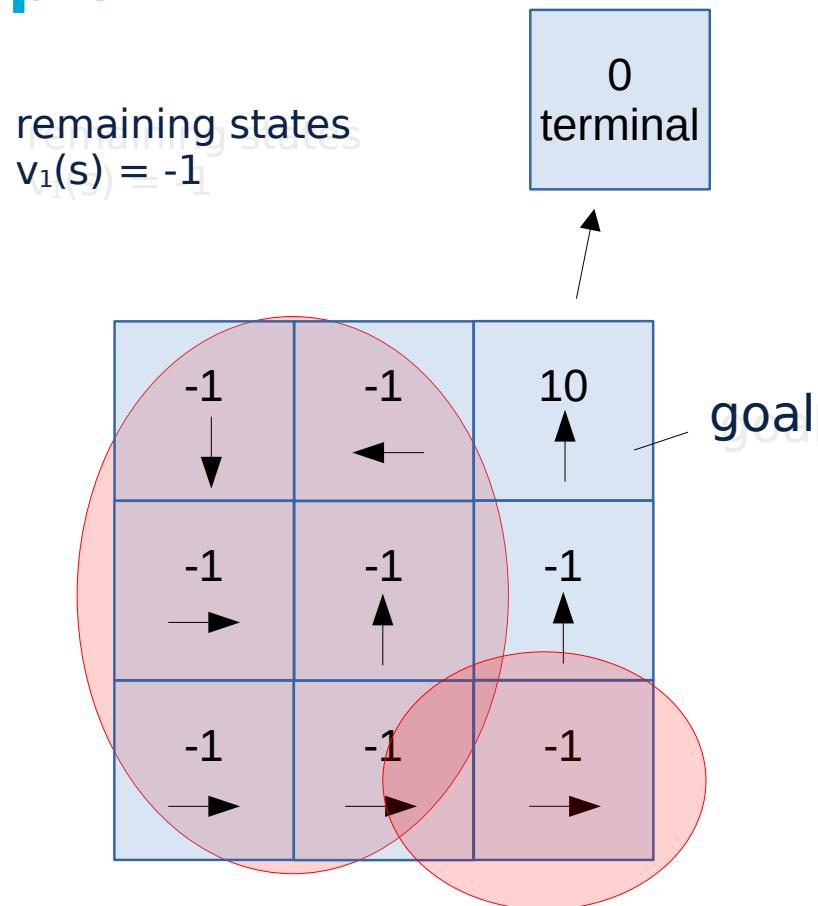
$$R(s=\text{terminal}, a=*) = 0$$

$$R(s=*, a=*) = -1 \quad (\text{otherwise})$$

- discount  $\gamma=0.9$

Let's start  
updating...

$$v_1(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_0(s')$$



# Policy Evaluation Example

- A little maze:

- transitions:

N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$

- reward:

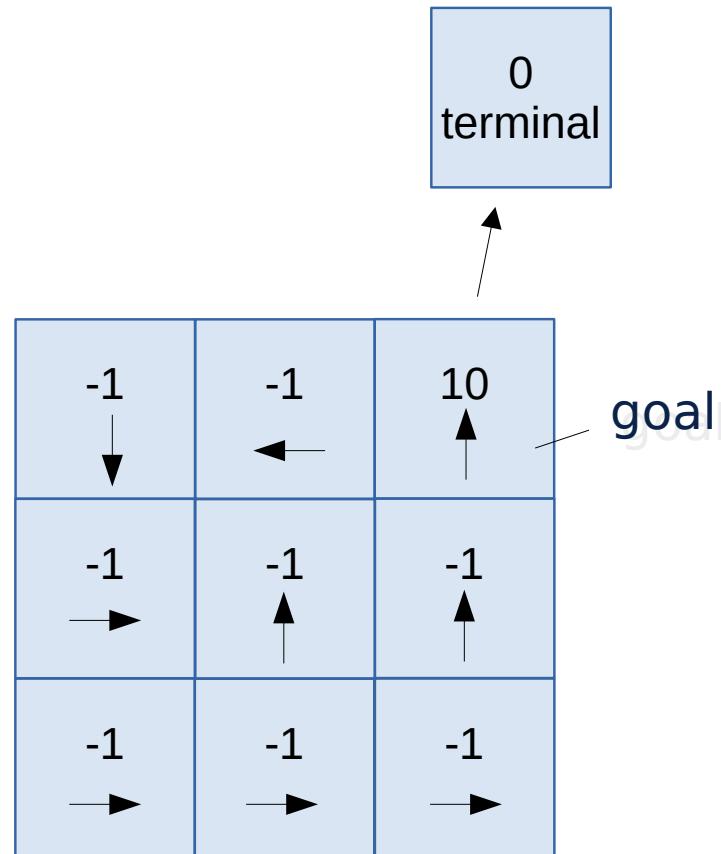
$$R(s=\text{Goal}, a=*) = +10$$

$$R(s=\text{terminal}, a=*) = 0$$

$$R(s=*, a=*) = -1 \quad (\text{otherwise})$$

- discount  $\gamma=0.9$

Have now computed  $v_1$ ,  
lets move to  $v_2\dots$



# Policy Evaluation Example

- A little maze:

- transitions:

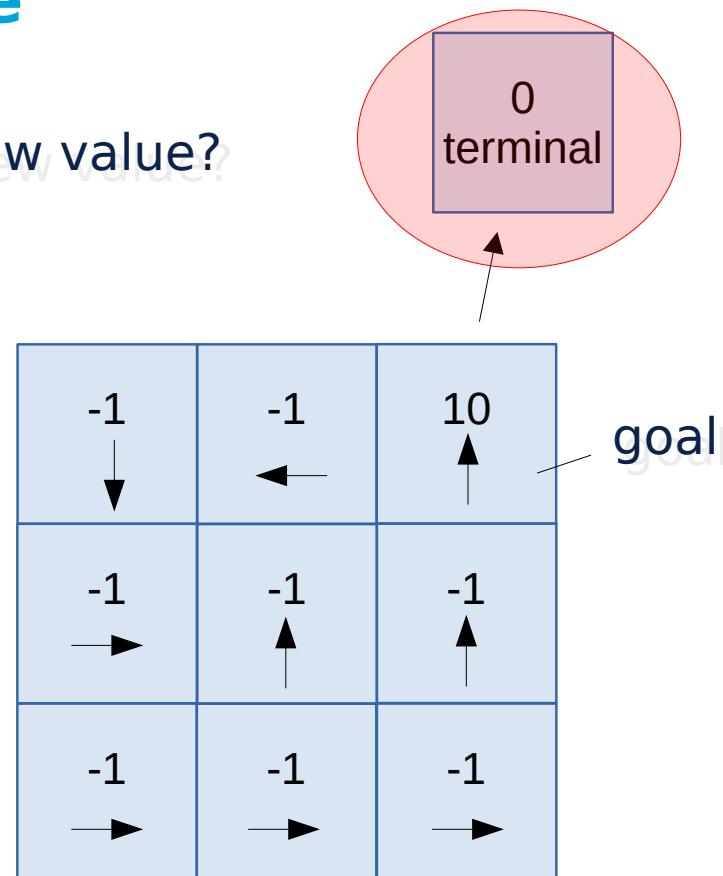
N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$

- reward:

$R(s=\text{Goal}, a=*) = +10$   
 $R(s=\text{terminal}, a=*) = 0$   
 $R(s=*, a=*) = -1 \quad (\text{otherwise})$

- discount  $\gamma=0.9$

new value?



$$v_2(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_1(s')$$

# Policy Evaluation Example

- A little maze:

- transitions:

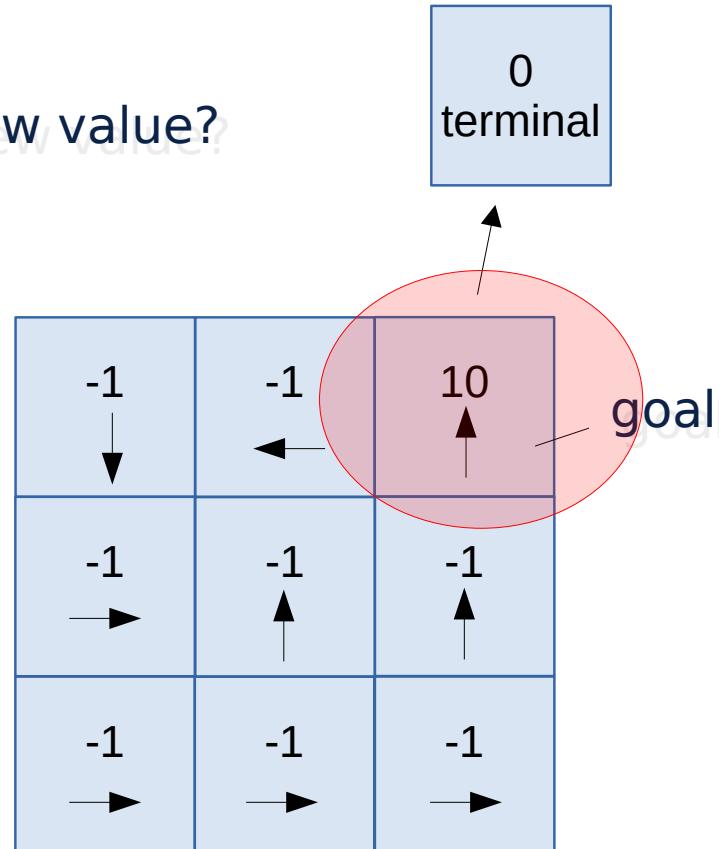
N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$

- reward:

$R(s=\text{Goal}, a=*) = +10$   
 $R(s=\text{terminal}, a=*) = 0$   
 $R(s=*, a=*) = -1 \quad (\text{otherwise})$

- discount  $\gamma=0.9$

new value?



$$v_2(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_1(s')$$

# Policy Evaluation Example

- A little maze:

- transitions:

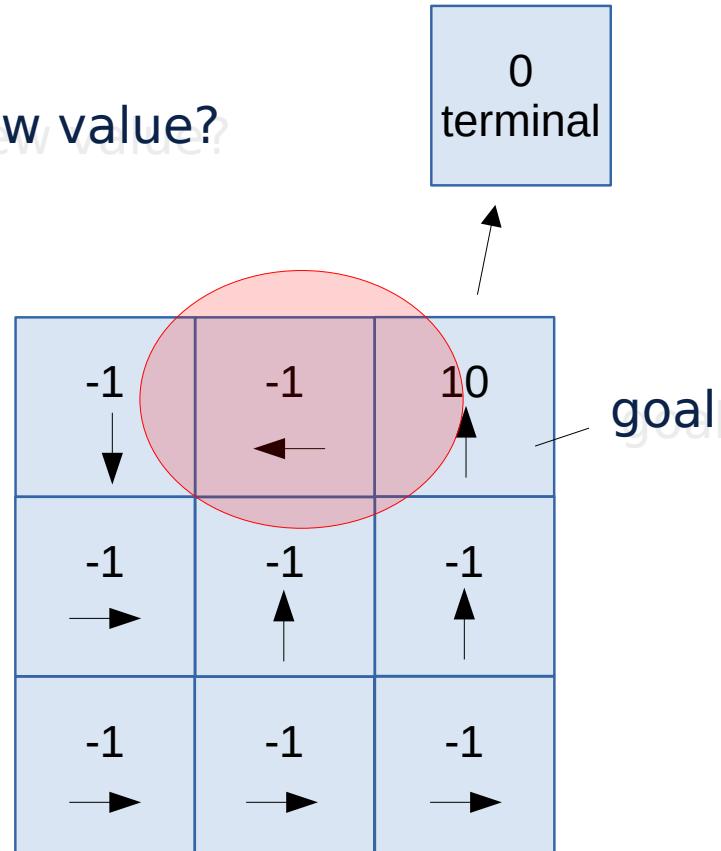
N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$

- reward:

$R(s=\text{Goal}, a=*) = +10$   
 $R(s=\text{terminal}, a=*) = 0$   
 $R(s=*, a=*) = -1 \quad (\text{otherwise})$

- discount  $\gamma=0.9$

new value?



$$v_2(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_1(s')$$

# Policy Evaluation Example

- A little maze:

- transitions:

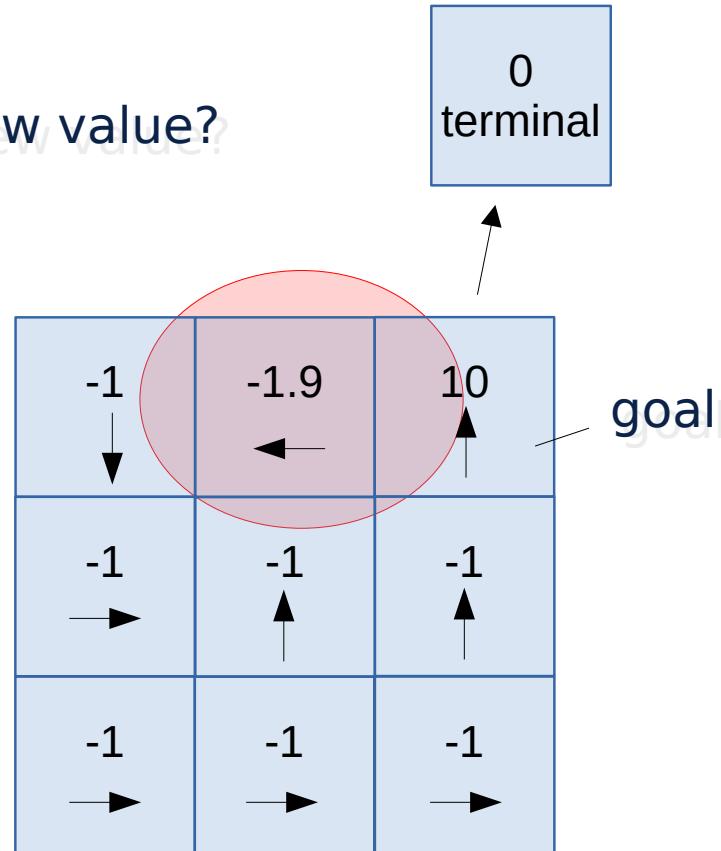
N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$

- reward:

$R(s=\text{Goal}, a=*) = +10$   
 $R(s=\text{terminal}, a=*) = 0$   
 $R(s=*, a=*) = -1 \quad (\text{otherwise})$

- discount  $\gamma=0.9$

new value?



$$v_2(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_1(s')$$

# Policy Evaluation Example

- A little maze:

- transitions:

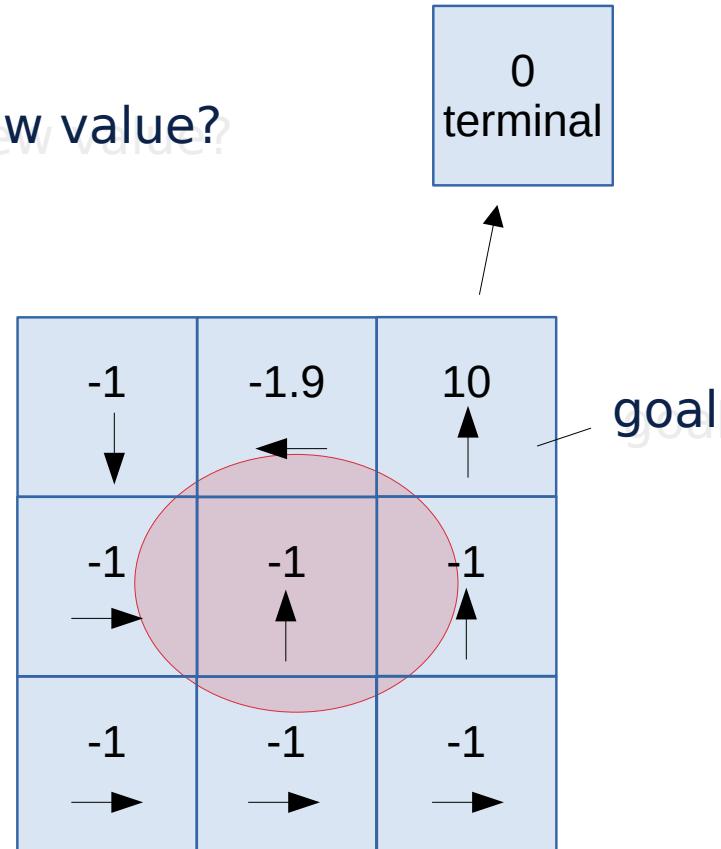
N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$

- reward:

$R(s=\text{Goal}, a=*) = +10$   
 $R(s=\text{terminal}, a=*) = 0$   
 $R(s=*, a=*) = -1 \quad (\text{otherwise})$

- discount  $\gamma=0.9$

new value?



$$v_2(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_1(s')$$

# Policy Evaluation Example

- A little maze:

- transitions:

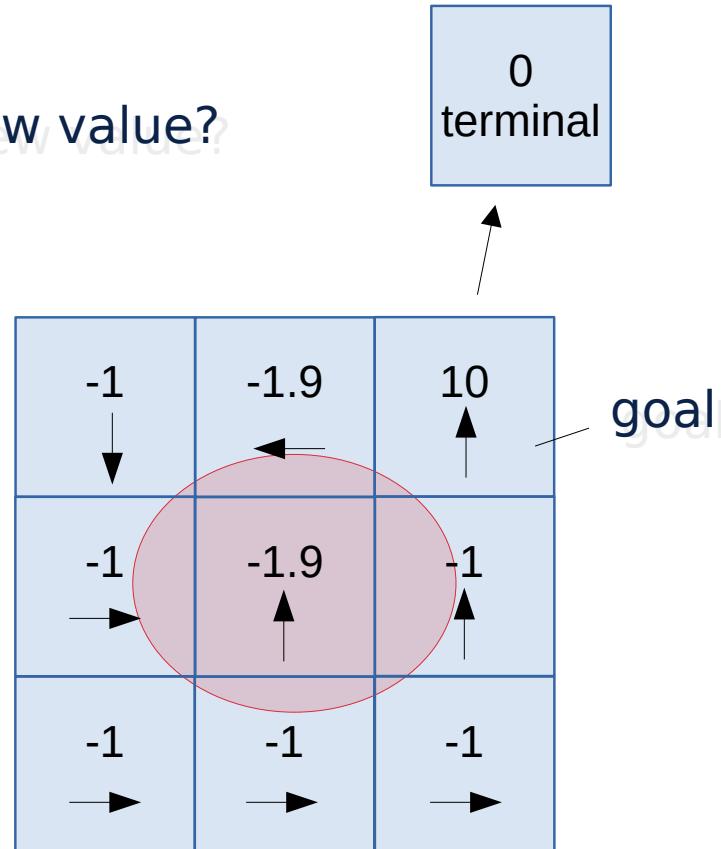
N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$

- reward:

$R(s=\text{Goal}, a=*) = +10$   
 $R(s=\text{terminal}, a=*) = 0$   
 $R(s=*, a=*) = -1$  (otherwise)

- discount  $\gamma=0.9$

new value?



$$v_2(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_1(s')$$

# Policy Evaluation Example

- A little maze:

- transitions:

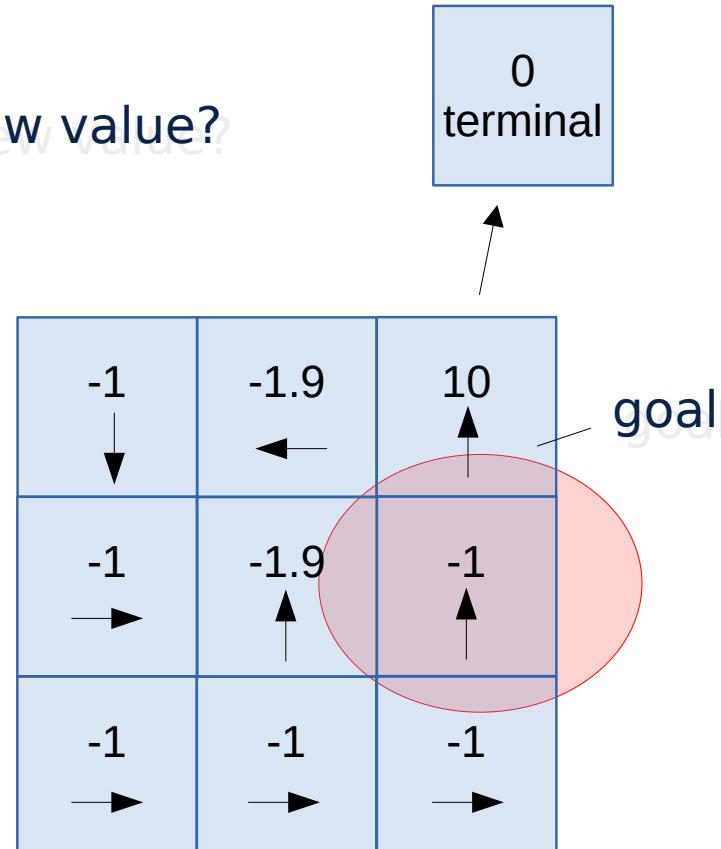
N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$

- reward:

$R(s=\text{Goal}, a=*) = +10$   
 $R(s=\text{terminal}, a=*) = 0$   
 $R(s=*, a=*) = -1 \quad (\text{otherwise})$

- discount  $\gamma=0.9$

new value?



$$v_2(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_1(s')$$

# Policy Evaluation Example

- A little maze:

- transitions:

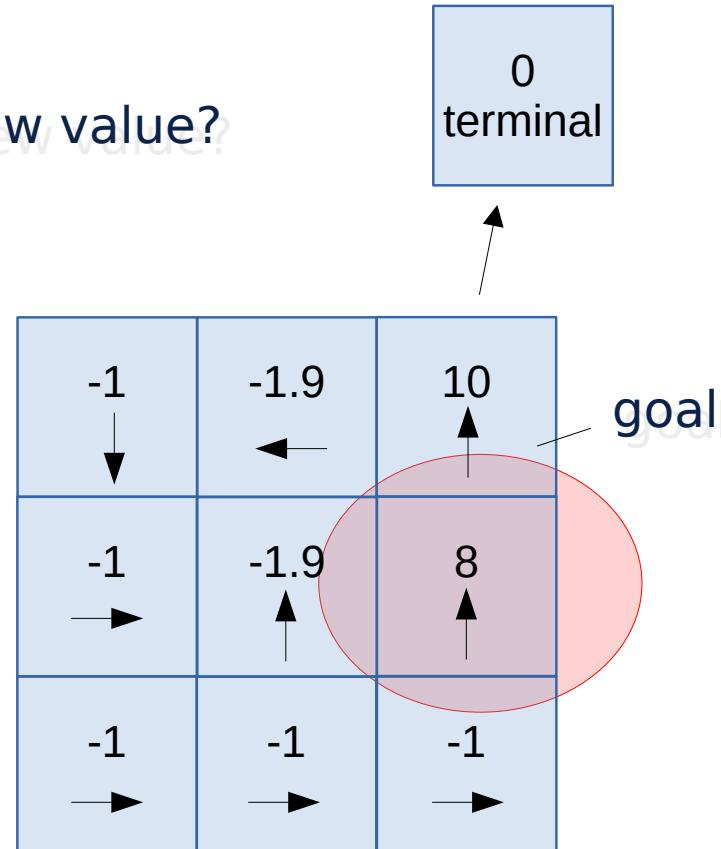
N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$

- reward:

$R(s=\text{Goal}, a=*) = +10$   
 $R(s=\text{terminal}, a=*) = 0$   
 $R(s=*, a=*) = -1 \quad (\text{otherwise})$

- discount  $\gamma=0.9$

new value?



$$v_2(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_1(s')$$

# Policy Evaluation Example

- A little maze:

- transitions:

N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$

- reward:

$$R(s=\text{Goal}, a=*) = +10$$

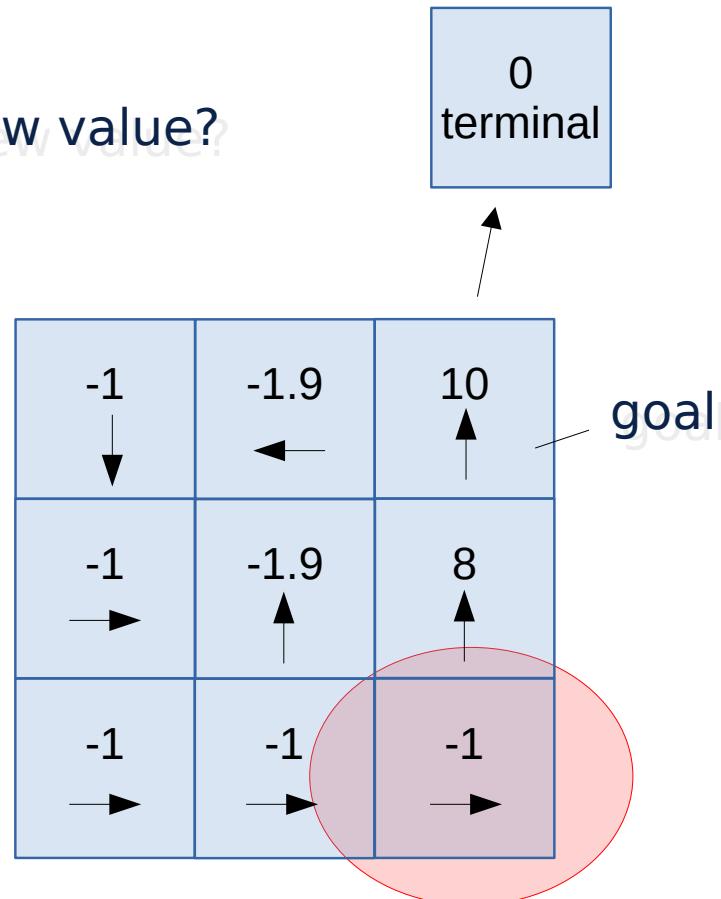
$$R(s=\text{terminal}, a=*) = 0$$

$$R(s=*, a=*) = -1 \quad (\text{otherwise})$$

- discount  $\gamma=0.9$

$$v_2(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_1(s')$$

new value?



# Policy Evaluation Example

- A little maze:

- transitions:

N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} | s=\text{goal}, a=*)=1$

- reward:

$$R(s=\text{Goal}, a=*) = +10$$

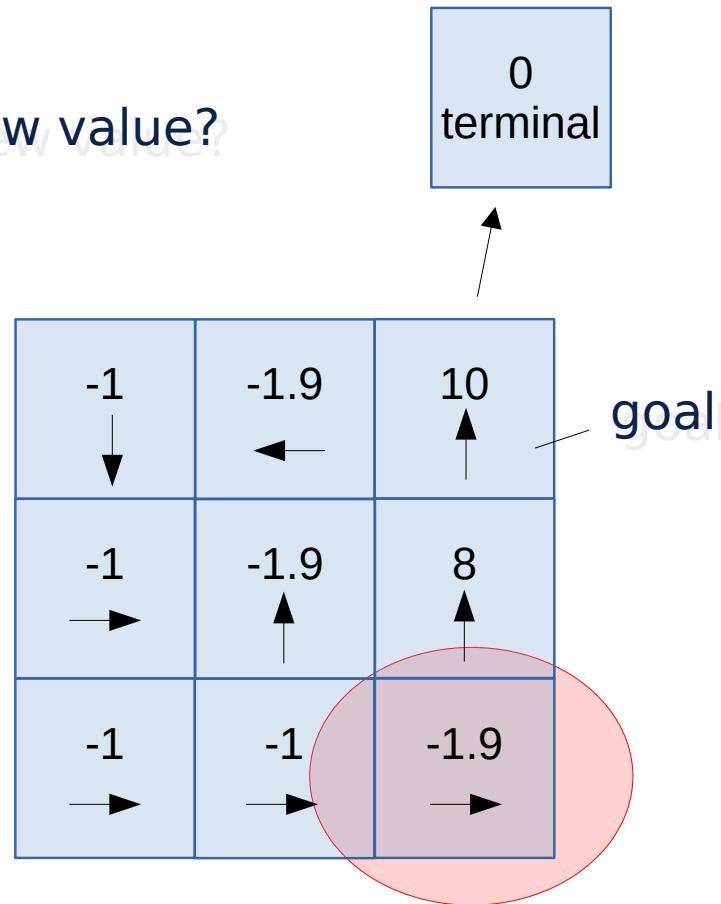
$$R(s=\text{terminal}, a=*) = 0$$

$$R(s=*, a=*) = -1 \quad (\text{otherwise})$$

- discount  $\gamma=0.9$

$$v_2(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_1(s')$$

new value?



# Policy Evaluation Example

- A little maze:

- transitions:

N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} | s=\text{goal}, a=*)=1$

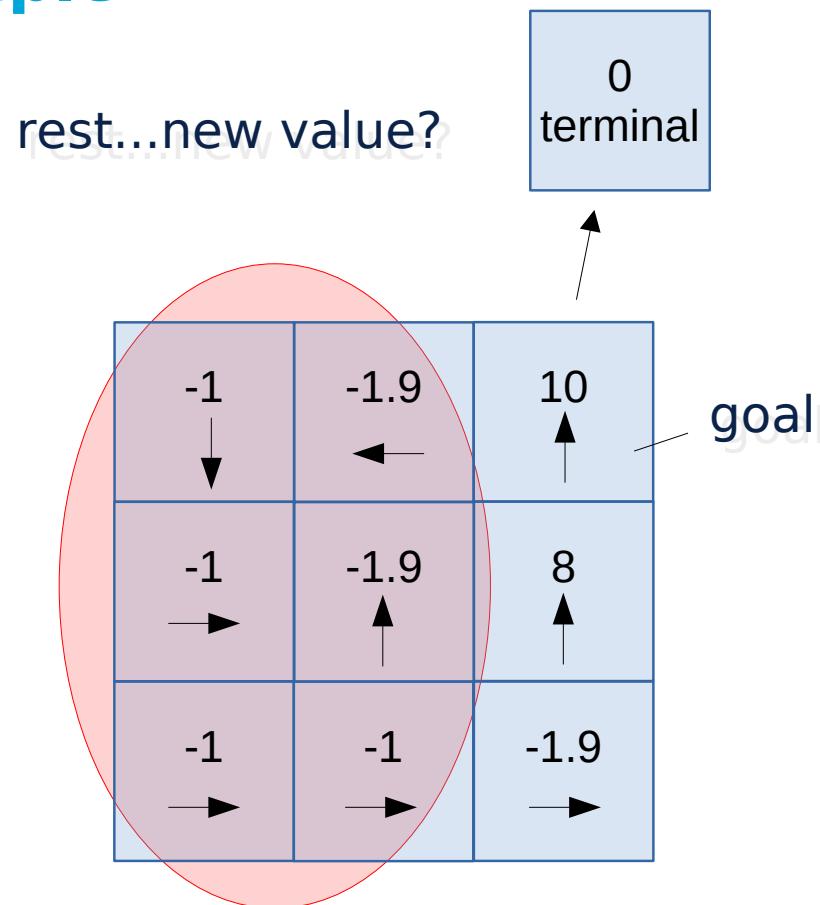
- reward:

$$R(s=\text{Goal}, a=*) = +10$$

$$R(s=\text{terminal}, a=*) = 0$$

$$R(s=*, a=*) = -1 \quad (\text{otherwise})$$

- discount  $\gamma=0.9$



$$v_2(s) := r(s, \pi(s)) + \sum_{s'} p(s'|s, \pi(s)) \gamma v_1(s')$$

# Policy Evaluation Example

- A little maze:

- transitions:

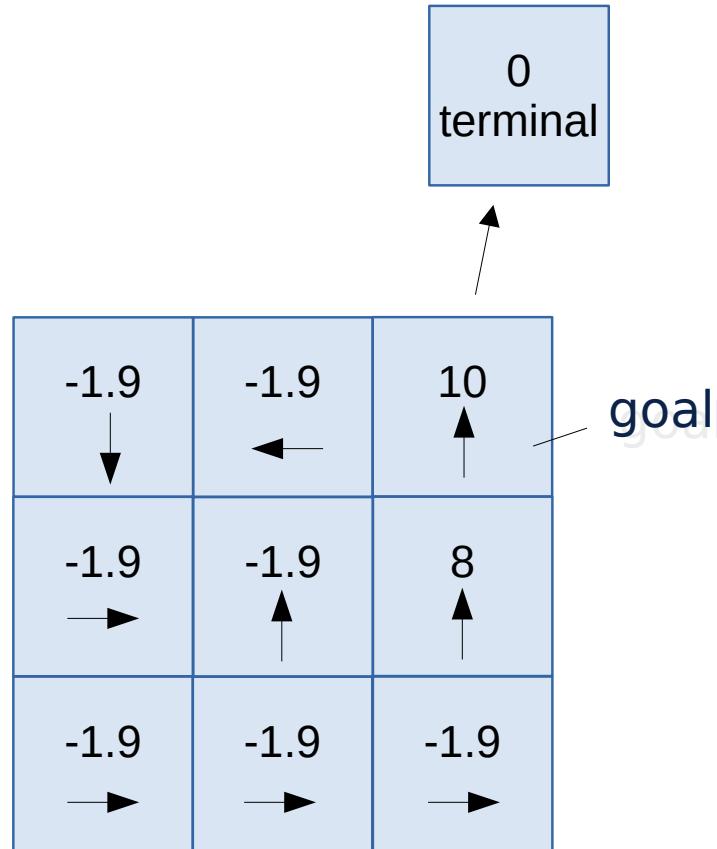
N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$

- reward:

$R(s=\text{Goal}, a=*) = +10$   
 $R(s=\text{terminal}, a=*) = 0$   
 $R(s=*, a=*) = -1$  (otherwise)

- discount  $\gamma=0.9$

have now computed  $v_2$



# Policy Evaluation Example

- A little maze:

- transitions:

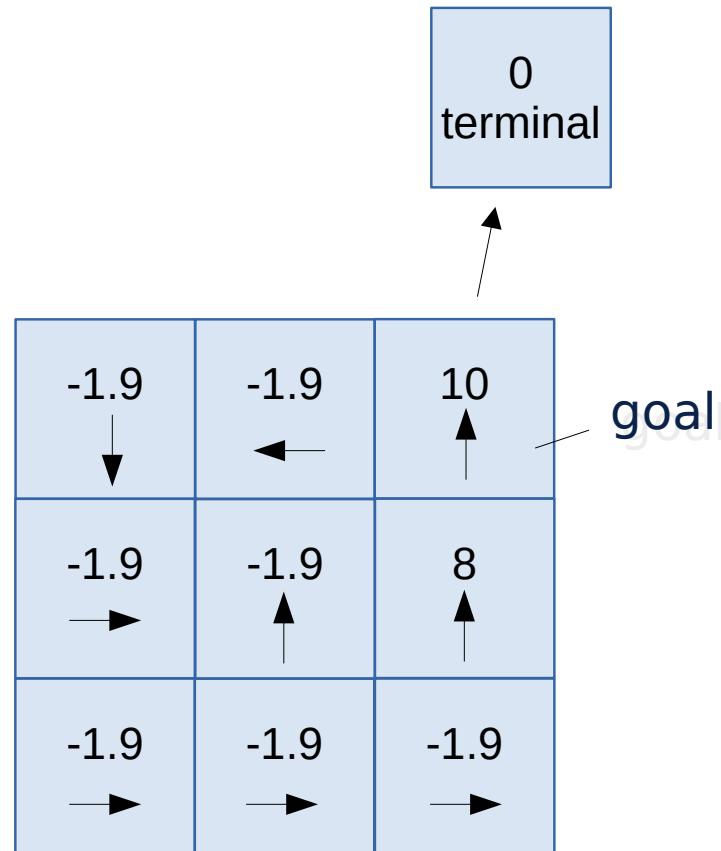
N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$

- reward:

$R(s=\text{Goal}, a=*) = +10$   
 $R(s=\text{terminal}, a=*) = 0$   
 $R(s=*, a=*) = -1$  (otherwise)

- discount  $\gamma=0.9$

how about  $v_3\dots?$



# Policy Evaluation Example

- A little maze:

- transitions:

N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$

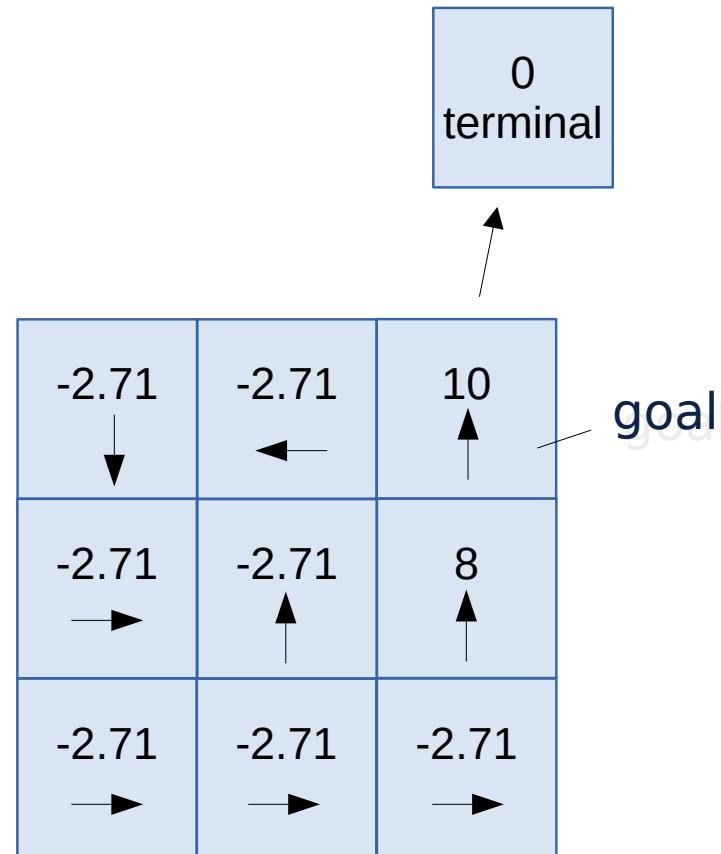
- reward:

$$R(s=\text{Goal}, a=*) = +10$$

$$R(s=\text{terminal}, a=*) = 0$$

$$R(s=*, a=*) = -1 \quad (\text{otherwise})$$

- discount  $\gamma=0.9$        $v_3$  is this:



# Policy Evaluation Example

- A little maze:

- transitions:

N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$

- reward:

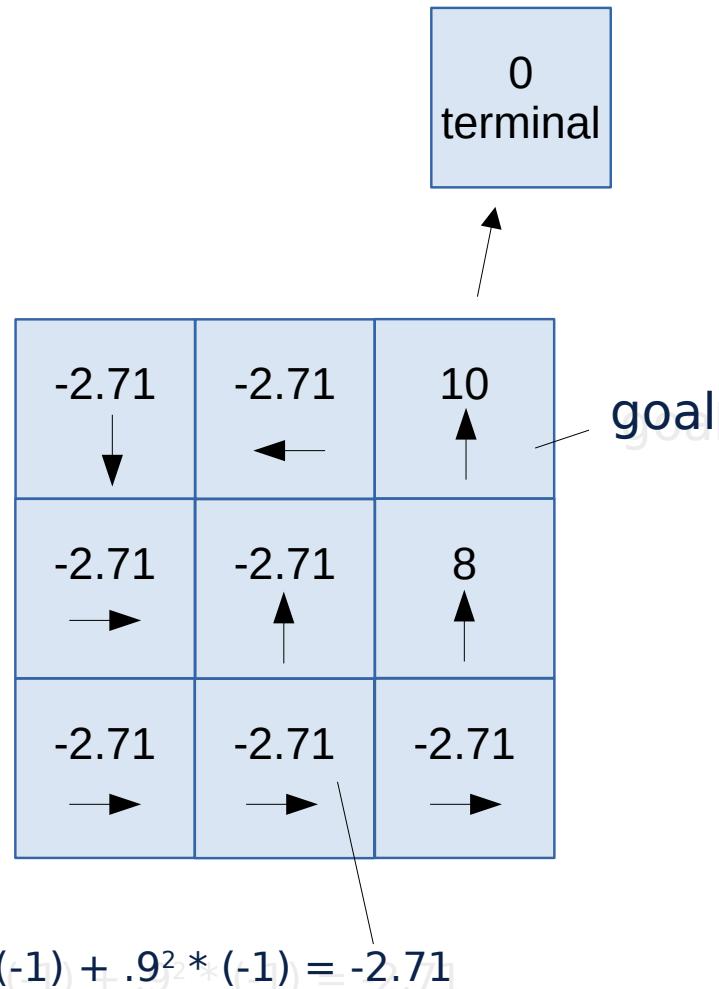
$$R(s=\text{Goal}, a=*) = +10$$

$$R(s=\text{terminal}, a=*) = 0$$

$$R(s=*, a=*) = -1 \quad (\text{otherwise})$$

- discount  $\gamma=0.9$

$v_3$  is this:



# Policy Evaluation Example

- A little maze:

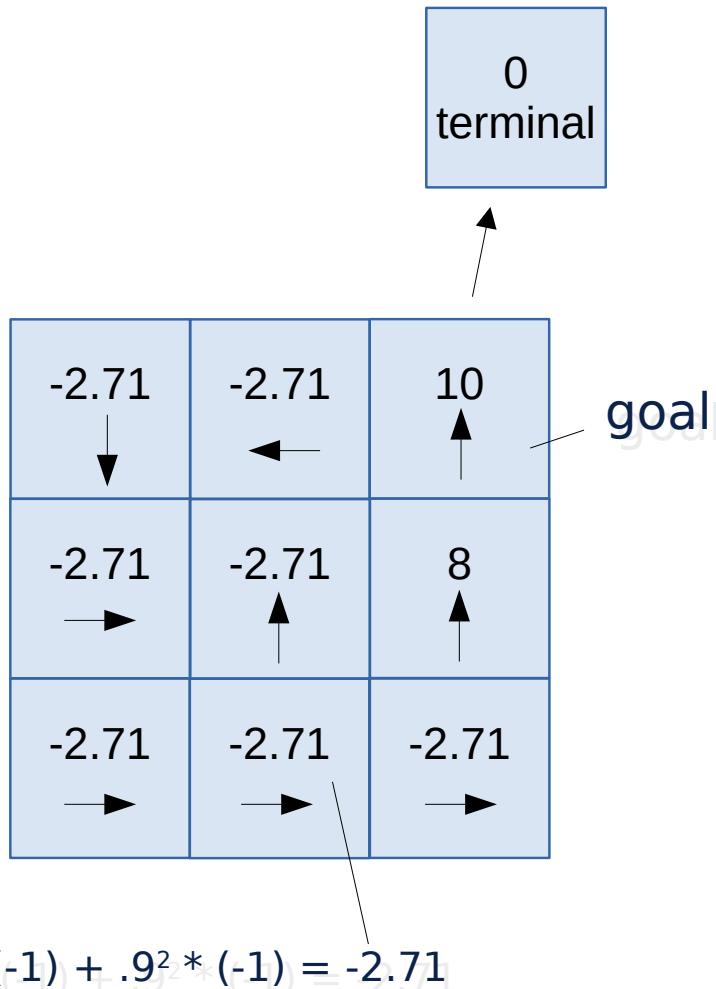
- transitions:

N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} | s=\text{goal}, a=*)=1$

- reward:

$R(s=\text{Goal}, a=*) = +10$   
 $R(s=\text{terminal}, a=*) = 0$

What does this converge to...?



$$-1 + .9 * (-1) + .9^2 * (-1) = -2.71$$

# Policy Evaluation Example

- A little maze:

- transitions:

N,E,S,W, deterministic movements  
 $P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$

- reward:

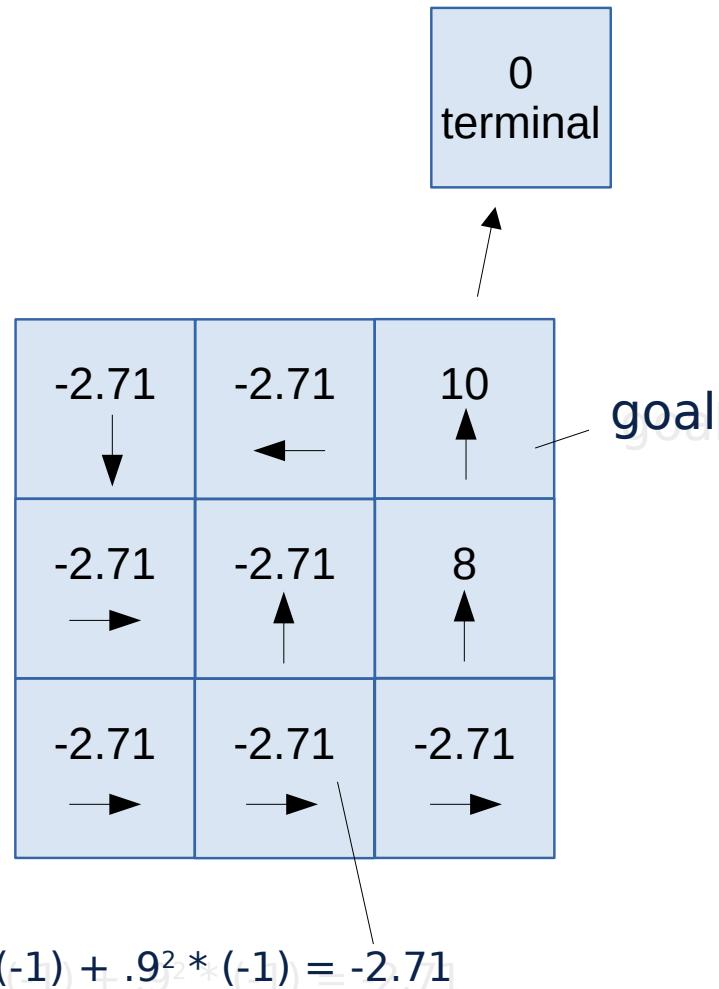
$$R(s=\text{Goal}, a=*) = +10$$

$$R(s=\text{terminal}, a=*) = 0$$

What does this converge to...?

$$\sum_{t=0}^{\infty} \gamma^t r = \frac{r}{1-\gamma} = \frac{-1}{1-0.9} = -10$$

$$-1 + .9 * (-1) + .9^2 * (-1) = -2.71$$



# Policy Evaluation Example

- A little maze:

- transitions:

N,E,S,W, deterministic movements

$P(s' = \text{terminal} \mid s=\text{goal}, a=*)=1$

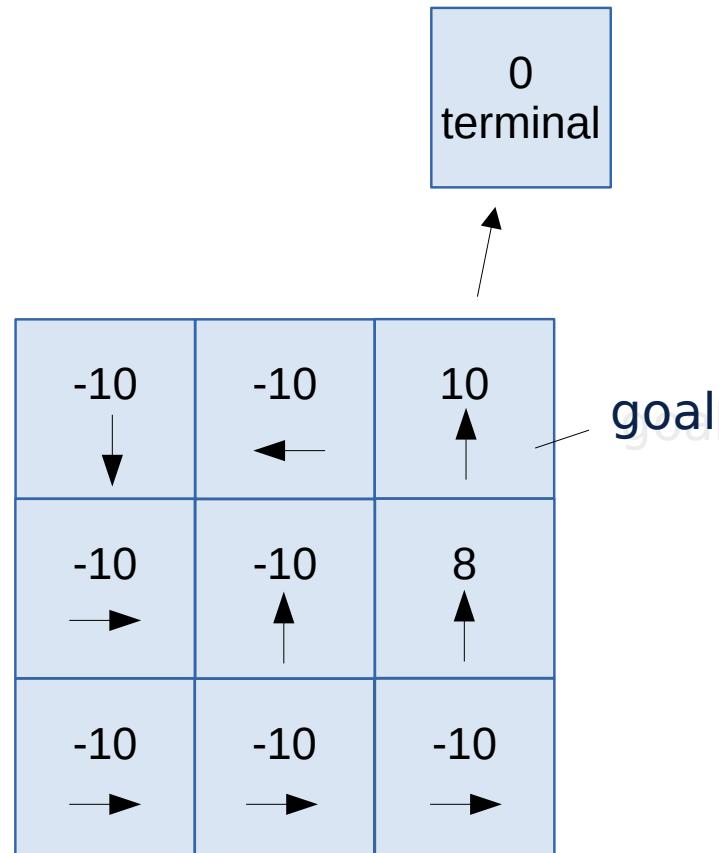
- reward:

$R(s=\text{Goal}, a=*) = +10$

$R(s=\text{terminal}, a=*) = 0$

$R(s=*, a=*) = -1$  (otherwise)

- discount  $\gamma=0.9$        $V_\pi = V_\infty$  is this:



## Implementational issues

- As you saw, we did “parallel updates”
  - $v_k$  is the ***k*-steps-to-go value function**  
(for following  $\pi$ )
  - but requires 2 arrays to implement
- Can also implement in 1 array
  - do updates “in place”
  - also converges, and can be faster!
  - but  $v_k$  will no longer correspond to *k*-steps-to-go value

## Policy Improvement - 1

- When we have computed  $v_\pi(s)$ ...  
...we want to use that to **improve** the policy!
- Let's define the **action-value function**:

$$q_\pi(s, a) = \sum_{s', r} p(s', r | s, a) [r + \gamma v_\pi(s')]$$

- expected value when selecting  $a$  at  $s$ , and following  $\pi$  afterwards

## Policy Improvement

1

different forms of  $v_\pi \rightarrow$  different forms of  $q_\pi$  !

- When we have computed  $v_\pi$  ...we want to use that
- Let's define the **action value function**

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) [r + \gamma v_\pi(s')]$$

$$q_\pi(s, a) = \sum_{s', r} p(s', r|s, a) [r + \gamma v_\pi(s')]$$

- expected value when selecting  $a$  at  $s$ , and following  $\pi$  afterwards

## Policy Improvement - 1

- When we have computed  $v_\pi(s)$ ...  
...we want to use that to **improve** the policy!
- Let's define the **action-value function**:

$$q_\pi(s, a) = \sum_{s', r} p(s', r | s, a) [r + \gamma v_\pi(s')]$$

- expected value when selecting  $a$  at  $s$ , and following  $\pi$

Notation:

- S&B use lower case 'v' and 'q'
- many other people use capitals 'V' and 'Q'

→ later in the slides I also use capitals.

## Policy Improvement - 2

- Now, given  $q_{\pi}(s,a)$ , we can improve the policy...  
...by being **greedy**:

forall  $s$ :  $\pi'(s) \leftarrow \max_a q_{\pi}(s,a)$

- Then repeat:
  - policy evaluation
  - policy improvement

→ called **policy iteration**

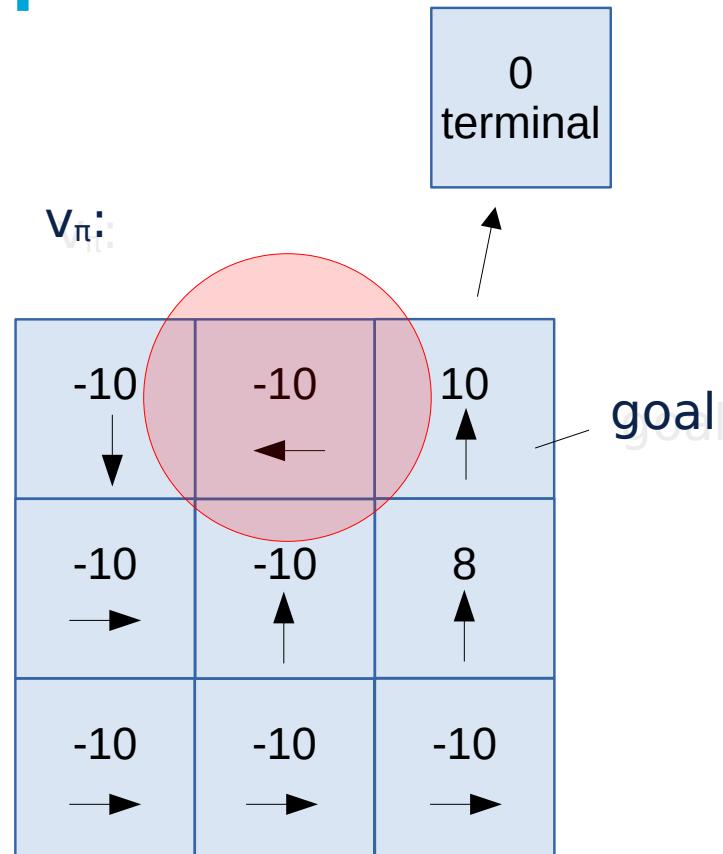
# Policy Improvement Example

- Continuing with our little maze:

- convenient Q-value function:

$$q_{\pi}(s, a) = r(s, a) + \gamma \sum_{s'} p(s' | s, a) v_{\pi}(s')$$

$$\begin{aligned} q(s(2,1), N) &= -1 + .9 * -10 = -10 \\ q(s(2,1), E) &= -1 + .9 * +10 = +8 \\ q(s(2,1), S) &= -1 + .9 * -10 = -10 \\ q(s(2,1), W) &= -1 + .9 * -10 = -10 \end{aligned}$$



# Policy Improvement Example

- Continuing with our little maze:

- convenient Q-value function:

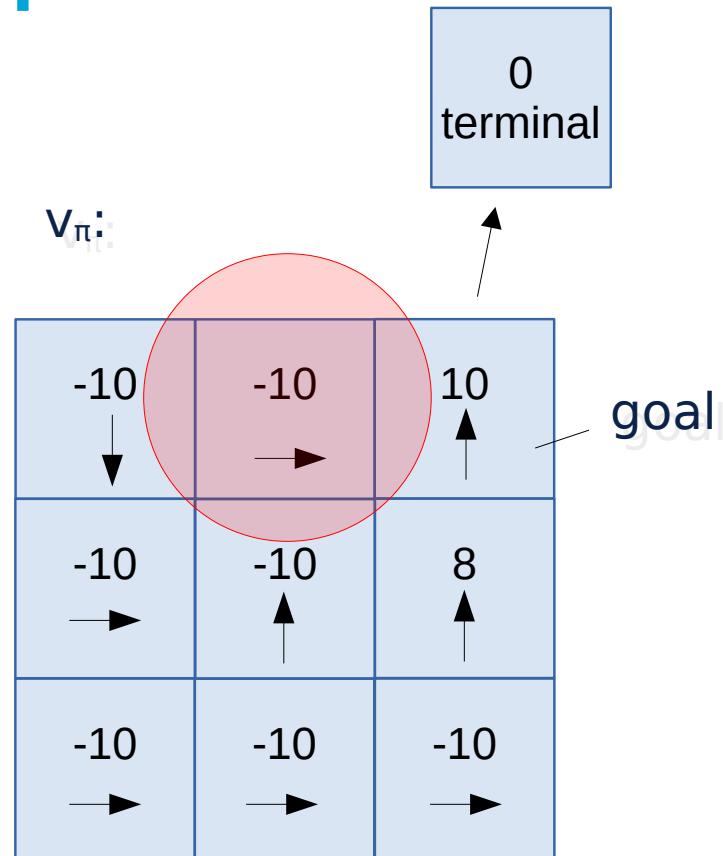
$$q_{\pi}(s, a) = r(s, a) + \gamma \sum_{s'} p(s'|s, a) v_{\pi}(s')$$

$$q(s(2,1), N) = -1 + .9 * -10 = -10$$

$$q(s(2,1), E) = -1 + .9 * +10 = +8$$

$$q(s(2,1), S) = -1 + .9 * -10 = -10$$

$$q(s(2,1), W) = -1 + .9 * -10 = -10$$



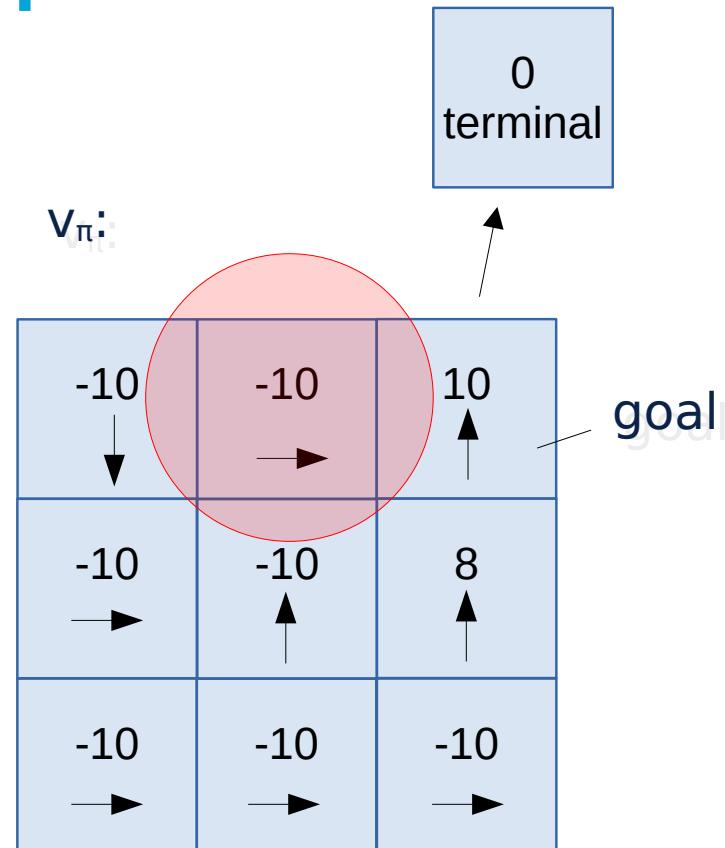
# Policy Improvement Example

- Continuing with our little maze:

- convenient Q-value function:

$$q_{\pi}(s, a) = r(s, a) + \gamma \sum_{s'} p(s' | s, a) v_{\pi}(s')$$

Other updates...?



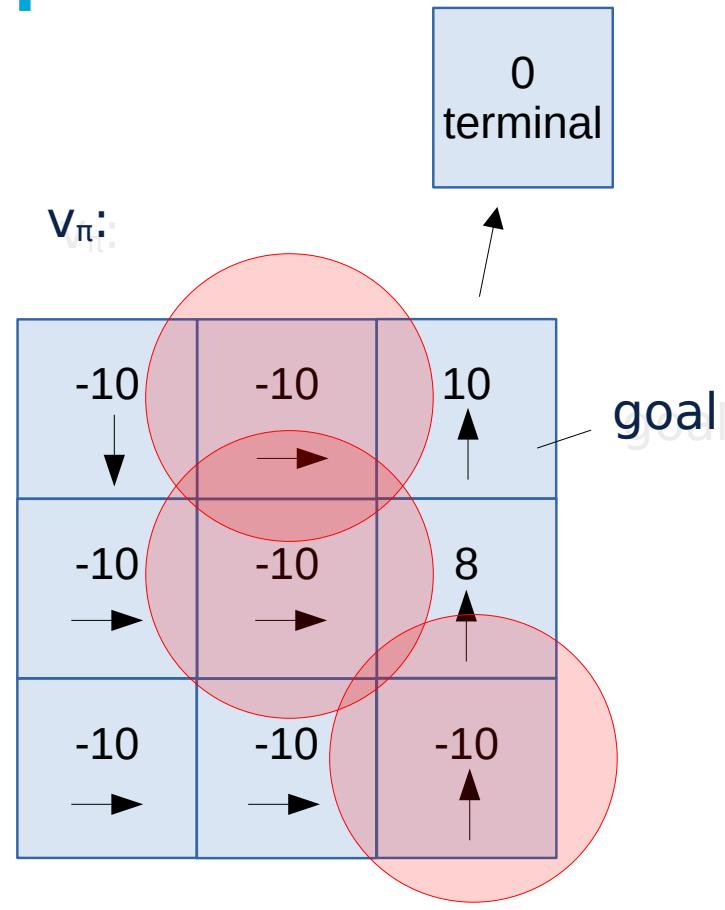
# Policy Improvement Example

- Continuing with our little maze:

- convenient Q-value function:

$$q_{\pi}(s, a) = r(s, a) + \gamma \sum_{s'} p(s' | s, a) v_{\pi}(s')$$

Other updates...?



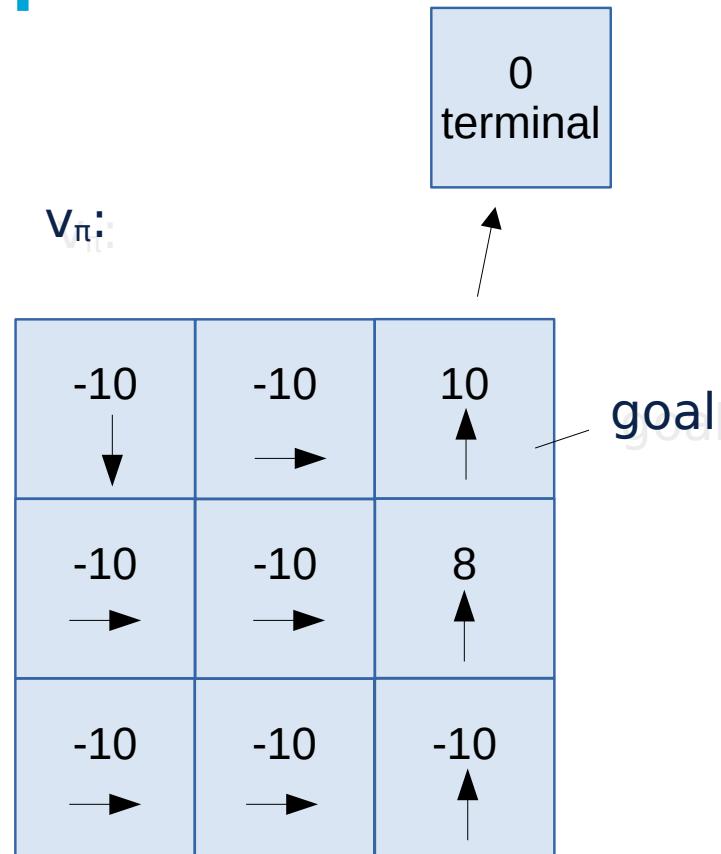
# Policy Improvement Example

- Continuing with our little maze:

- convenient Q-value function:

$$q_{\pi}(s, a) = r(s, a) + \gamma \sum_{s'} p(s' | s, a) v_{\pi}(s')$$

value  $v_{\pi}$  of this resulting policy?



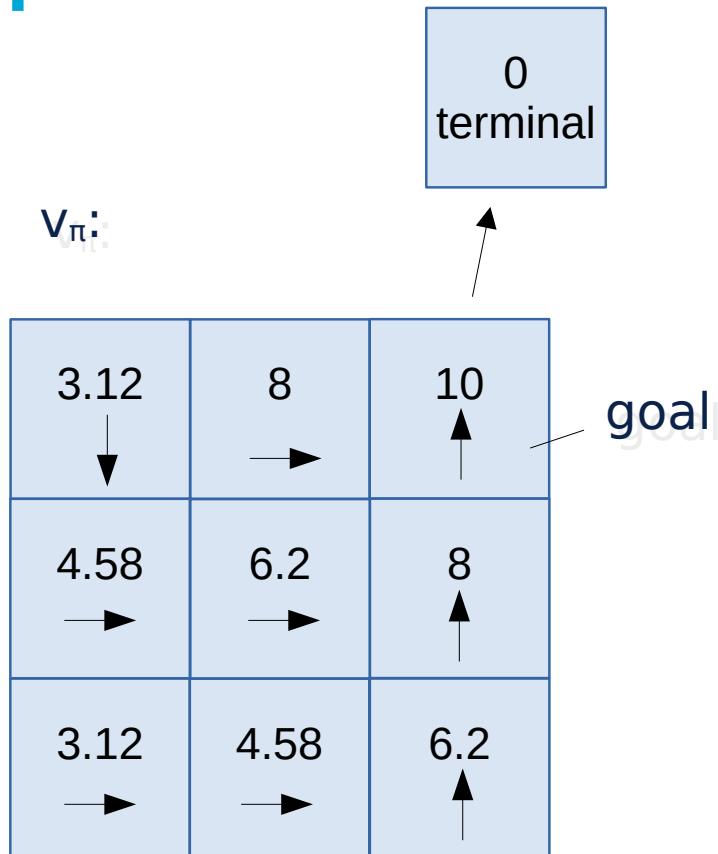
# Policy Improvement Example

- Continuing with our little maze:

- convenient Q-value function:

$$q_{\pi}(s, a) = r(s, a) + \gamma \sum_{s'} p(s' | s, a) v_{\pi}(s')$$

value  $v_{\pi}$  of this resulting policy?



# Optimal policies & (action-) value functions

- So does this converge...?

# Optimal policies & (action-) value functions

- So does this converge...?
- Yes! Converges to unique optimal value functions
  - given by **Bellman optimality equations:**

$$v_*(s) = \max_a q_*(s, a)$$

$$q_*(s, a) = \sum_{s', r} p(s', r | s, a) [r + \gamma v_*(s')]$$

- There can be multiple optimal policies
  - they share the same optimal value function

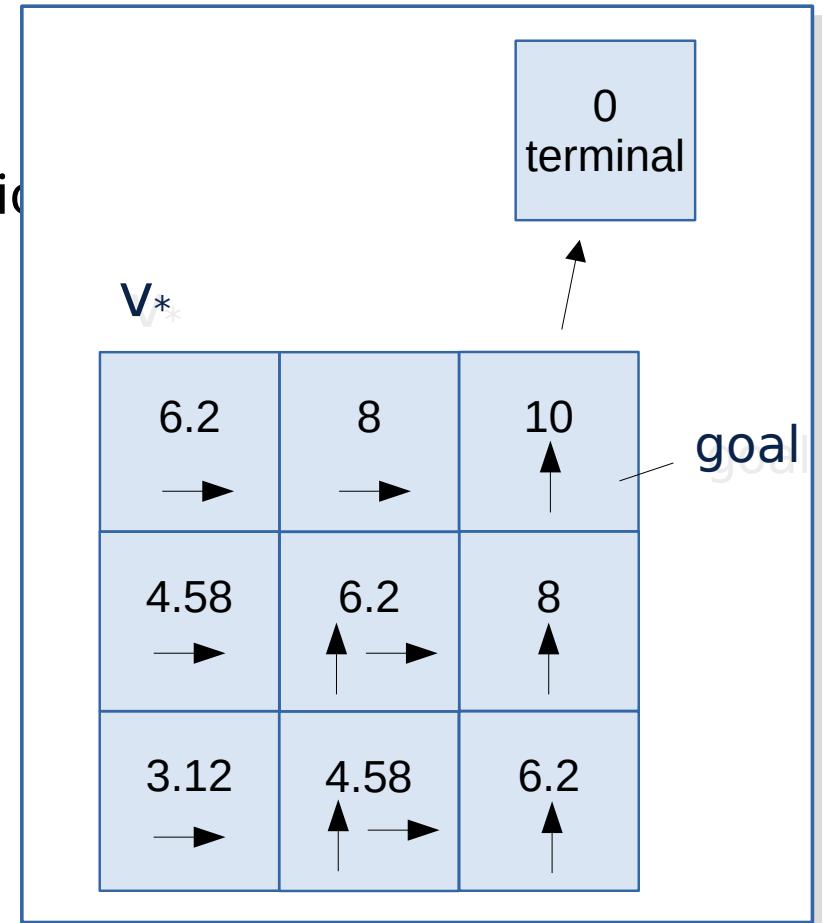
# Optimal policies & (action-) value functions

- So does this converge...?
- Yes! Converges to unique optimal value function
  - given by **Bellman optimality equations:**

$$v_*(s) = \max_a q_*(s, a)$$

$$q_*(s, a) = \sum_{s', r} p(s', r | s, a) [r + \gamma v_*(s')]$$

- There can be multiple optimal policies
  - they share the same optimal value function



## Generalized Policy Iteration & Value Iteration

- Is it needed to run policy evaluation until convergence...?  
→ No..! Can do a few iterations of IPE, and then do policy improvement. Still works.
- In the extreme: value iteration
  - does only 1 IPE iteration
  - It combines IPE and policy improvement in a single update rule:

$$v_{k+1}(s) \leftarrow \max_a \left\{ \sum_{s', r} p(s'|s, a)[r(s, a, s') + \gamma v_k(s')] \right\}$$

- repeatedly sweep through state space, until convergence

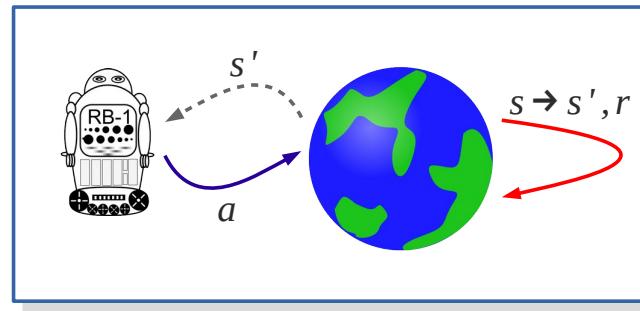
# Conclusions

- **Sequential decision making:** When applicable?
- An SDM problem can be defined using the **MDP framework**:
  - **States** (and how they are represented by the agent)
  - **Actions** (and their effect in terms of state transitions)
  - **reward** function (payoff for certain actions in certain states)
  - **transition** function
- Task concepts:
  - Markov property, episodes, absorbing states
- **Planning vs reinforcement learning**
- Planning concepts:
  - policy, value function, state-action value function
  - different notations/formulations of them
- Planning methods: policy iteration, value iteration



# Reinforcement Learning

## Part 2



Dr. Frans A. Oliehoek

## Refresher... we saw:

- **Sequential decision making:** When applicable?
- An SDM problem can be defined using the **MDP framework**:
  - **States** (and how they are represented by the agent)
  - **Actions** (and their effect in terms of state transitions)
  - **reward** function (payoff for certain actions in certain states)
  - **transition** function
- Task concepts: Markov property, episodes, absorbing states
- **Planning vs reinforcement learning**
- Planning concepts:
  - policy, value function, state-action value function
  - different notations/formulations of them
- Planning methods: policy iteration, value iteration

## Refresher... we saw:

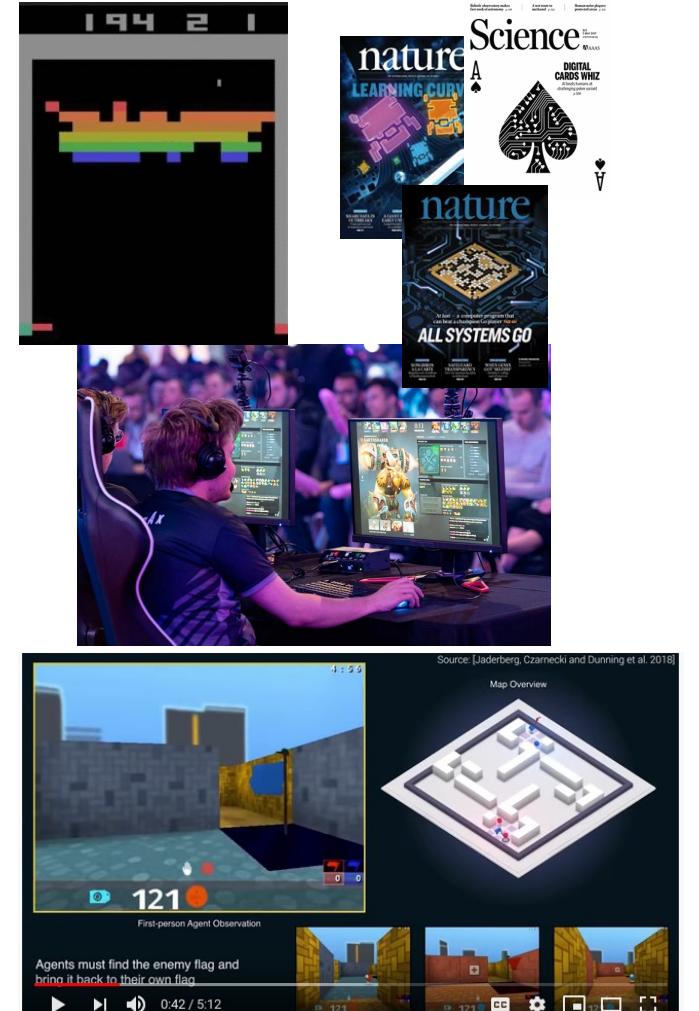
- **Sequential decision making:** When applicable?
- An SDM problem can be defined using the **MDP framework**:
  - **States** (and how they are represented by the agent)
  - **Actions** (and their effect in terms of state transitions)
  - **reward** function (payoff for certain actions in certain states)
  - **transition** function
- Task concepts: Markov property, episodes, absorbing states
- **Planning vs reinforcement learning**
- Planning concepts:
  - policy, value function, state-action value function
  - different notations/formulations of them
- Planning methods: policy iteration, value iteration

# Learning objectives

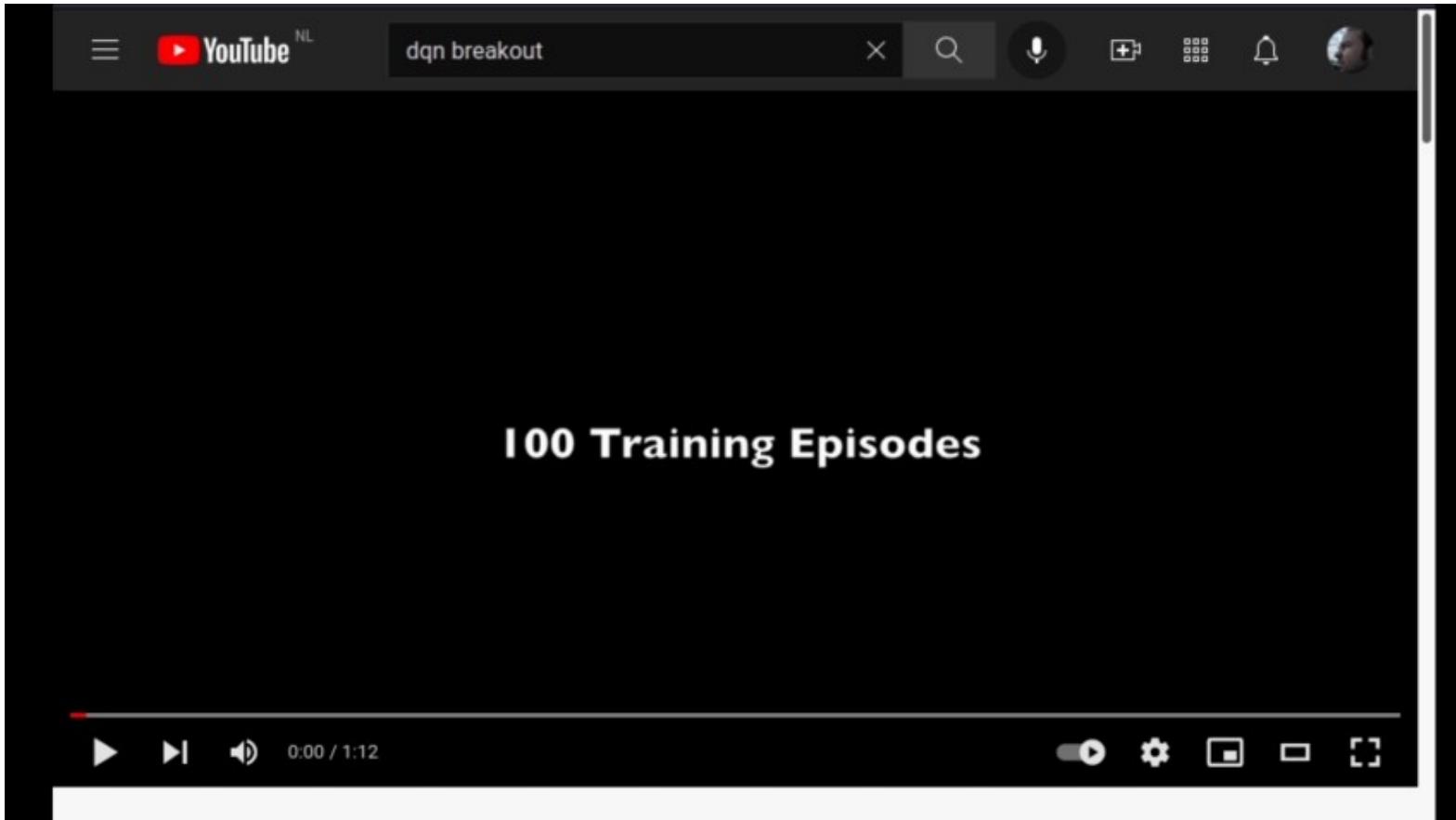
- Explain difference between RL and planning
- Types of RL methods:
  - **policy evaluation** versus **control**
  - **model-based RL** and **model-free RL**
  - **on-policy** versus **off-policy learning**
  - **on-line** versus **off-line learning**
- Explain the **exploration/exploitation** trade-off.
- Apply TD learning, SARSA and Q-learning
- Understand the basics of and challenges for **deep RL**

# Glimpse of the State of the Art...

- “Deep RL”: Combination of RL techniques with deep neural networks
- Many recent results:
  - Atari Breakout
  - Go, Poker
  - Dota 2 / Starcraft
  - Simulated Robotics/Locomotion
  - Hide and Seek
  - Capture the flag
  - Chip Design
  - Summarizing books
  - ‘Generally capable agents’



# Breakout



# How do we *learn*?

# The problem of \*learning\*

- **Planning** requires access to the model ( $T, R$ )
  - In many cases, we don't have this...
- Reinforcement learning: learn optimal policy from (interaction) data.  
Roughly:
  - act using an 'exploration policy'  $\pi$
  - collect the generated transitions:  $\{ (s, a, s', r) \}$
  - then improve the policy
- Two main approaches:
  - **model-based RL**: learn a model, than plan
  - **model-free RL**: e.g., try to learn  $q_*$  directly

# Model-based RL

- Classic approach:
  - by counting!
  - $N(s,a,s')$  counts how often we saw  $(s,a) \rightarrow s'$
  - estimate  $P(s'|s,a) = N(s,a,s') / N(s,a)$
  - (and similar for rewards)
- Given model... use planning!
- Current research...  
how to combine this with  
deep learning?  
→ I do some research on this topic.



'world models'  
[Ha&Schmidhuber'18 NeurIPS]

## Model free RL

- Core idea: estimate the
    - value function,
    - q-value function,
    - or policy
- directly, **without first learning a model**

## Model free RL

- Core idea: estimate the
  - value function,
  - q-value function,
  - or policy

directly, **without first learning a model**

- Next up:
  - **policy evaluation** to learn  $V_\pi(s)$
  - then extend to 'control'

A bit like before:

Policy iteration combined

- policy evaluation
- policy improvement

→ also in **learning** policy evaluation is important

## Model-free Policy Evaluation: Temporal Difference (TD) learning

- Main idea: take the Bellman equation

$$V_\pi(s) = R(s, \pi(s)) + \sum_{s'} P(s'|s, \pi(s)) \gamma V_\pi(s')$$

- ...turn it into a update equation that uses **sampled experience**.
- Specifically, when following  $\pi$ , after observing  $(s, a, r, s')$ , we update:

$$\hat{V}_\pi(s) := (1 - \alpha) \hat{V}_\pi(s) + \alpha [r + \gamma \hat{V}_\pi(s')]$$

Update target:  
if last time step: just  $R(s, a)$   
(or add terminal state...)

## Model-free Policy Evaluation: Temporal Difference (TD) learning

- Main idea: take the expectation of the return

$$V_\pi(s) = R(s, \pi(s)) + \gamma V_\pi(s')$$

- ...turn it into a TD update rule

$$\hat{V}_\pi(s) := (1 - \alpha)\hat{V}_\pi(s) + \alpha[R(s, a) + \gamma \hat{V}_\pi(s')]$$

- Specifically, when  $\pi(a|s) = 1$

$$= \hat{V}_\pi(s) + \alpha[R(s, a) + \gamma \hat{V}_\pi(s') - \hat{V}_\pi(s)]$$

$$\hat{V}_\pi(s) := (1 - \alpha)\hat{V}_\pi(s) + \alpha[R(s, a) + \gamma \hat{V}_\pi(s') - \hat{V}_\pi(s)]$$

# TD Learning - Procedure

So, overall it works as follows...

- fix: policy  $\pi$ , learning rate  $\alpha$ , value estimates  $V(s)=0$
- //start executing in the environment:
- For each episode
  - $s = \text{initial state}$
  - For each step (while  $s$  is not terminal)
    - $a = \pi(s)$
    - $s',r = \text{execute}(a)$
    - $V(s) := V(s) + \alpha [r + \gamma V(s') - V(s)]$
    - $s = s'$

## Model-free Control: SARSA

- If we want to learn how to take actions...  
learn  $Q(s,a)$  instead!
- updates based on sampled transitions
- every step:
  - receive last transition  $\langle s,a,r,s' \rangle$
  - determine action  $a' \sim \pi_{\text{exploration}}(\cdot | s')$   $\leftarrow$  now we have  
 $\langle s,a,r,s',a \rangle$
  - update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$$

- After **convergence**, we have  $Q_\pi$  and can use policy improvement as before.

# Model-free Control: SARSA

- If we want to learn  $Q(s,a)$  instead of  $\pi_{\text{exploration}}$ ...  
learn  $Q(s,a)$  instead of  $\pi_{\text{exploration}}$ .  
 $\pi_{\text{exploration}}$  can actually depend on our estimate  $Q$ .
  - updates based on  $(s, a, r, s')$
  - every step:
    - receive last observation  $s'$
    - determine reward  $r$
    - update:  $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a))$
- Alternatively...
- E.g., we can use “epsilon-greedy”:
- with prob  $1-\varepsilon$ , take action  $\arg \max_a Q(s, a)$
  - with prob  $\varepsilon$ , take a uniform random action
- this means that  $\pi_{\text{exploration}}$  changes over time: it gets better.  
(instantiation of the idea of generalized policy iteration)

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a))$$

- After **convergence**, we have  $Q_\pi$  and can use policy improvement as before.

# On-policy and off-policy

- **On policy**
  - Learn the value function of the policy that selects the actions
  - i.e., we learn about the exploration policy
    - behavior generation is coupled to learning V or Q
- **Off policy**
  - Convergence of value function is NOT influenced by actions selected
  - Behavior generation is decoupled from learning V or Q

## Example: Sarsa.

- It learns about the action actually selected
- converges to  $v_\pi$  for a fixed  $\pi$
- so to converge to  $\pi^*$ , we need to gradually improve  $\pi$
- Typically: act by being greedy w.r.t. current estimate of Q

# Example off-policy: Q-learning

- Q-learning is an off policy method
- every step:
  - receive last transition  $\langle s, a, s', r \rangle$
  - update:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_b Q(s', b) - Q(s, a))$$

- update of value function is **NOT** influenced by actually selected action  $a'$  at this step
- cf. SARSA:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a))$$

**And how does learning influence our policy (and vice versa)?**

# Exploration vs. exploitation

- **Exploration** vs. **exploitation**
  - When to try something new vs when to use  $V(s)$  or  $Q(s, a)$  for action selection?
- When learning is **offline** (before deployment)
  - E.g., we have a simulator to do this
  - We could try this:
    - Purely random behavior to learn  $Q(s, a)$
    - Then use  $Q(s,a)$  to perform
- When learning **online**  
(in the actual environment):  
too costly!

← an active research topic

# Balancing Exploration vs. exploitation

- How to do action selection ('exploration policy') ?

- uniform random:  $p(a) = \frac{1}{|A|}$

- Selection of actions based on  $Q(s,a)$  values:

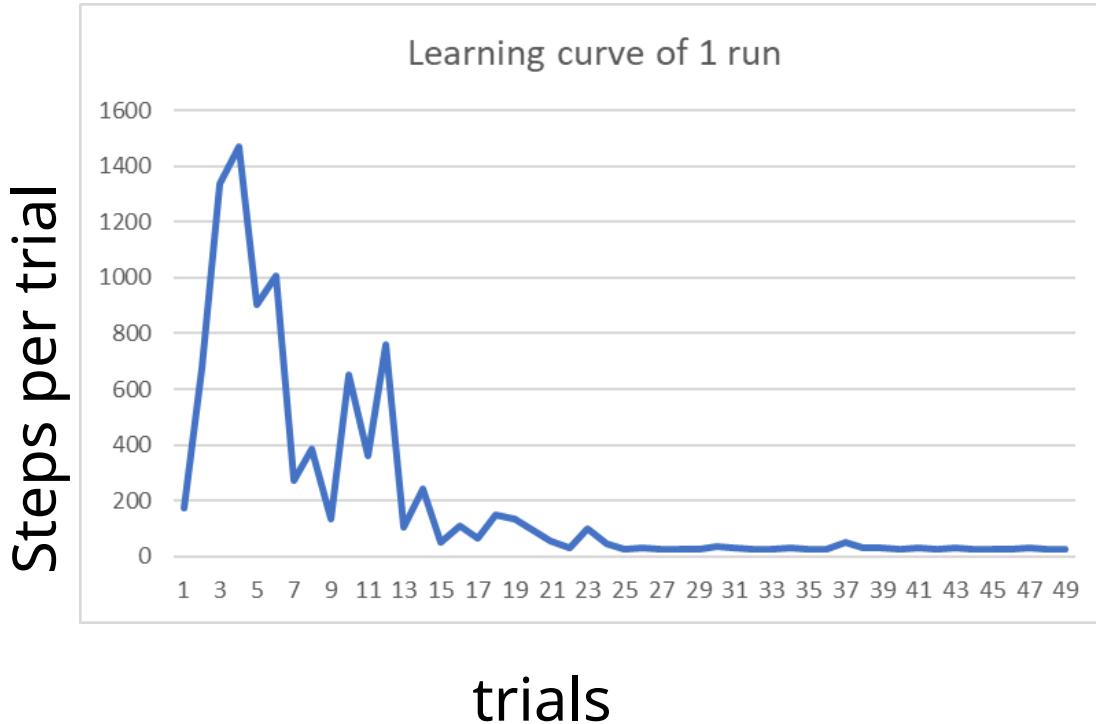
- Greedy:  $\arg \max_a Q(s,a)$
  - Epsilon greedy:
    - do uniform random with  $\epsilon$  probability
    - do greedy with  $(1-\epsilon)$

- Boltzmann exploration:  $p(a) = \frac{\exp[\beta \times Q(s,a)]}{\sum_{i=1}^{|A|} \exp[\beta \times Q(s,a_i)]}$

Remember on-policy versus off-policy value propagation!

So....?

# A typical learning curve



Step= one step taken

Trial/episode= from start to goal

Run= set of trials

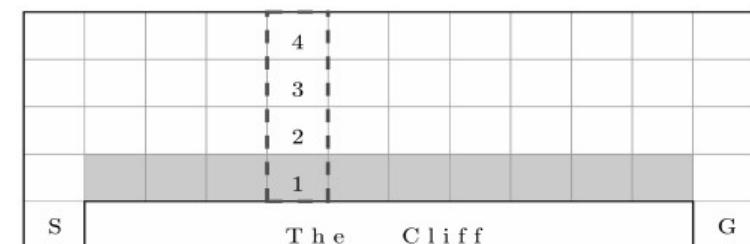


Figure 3: Cliff Walking scenario. Slippery path marked in grey (see text). The red box indicates the distance 4,3,2 and 1 to the Cliff depicted on the horizontal axis of figure 4.

# Scaling: Generalization & Value Function Approximation

# Value function representations

- So far: 'tabular' → represent  $V(s)$  as **value table**
  - **Each state** has a value that is learned
  - E.g., using Q-learning, or dynamic programming.

$$\hat{v}(s) \leftarrow \sum_{s', r} p(s', r | s, a) [r + \gamma \hat{v}(s')]$$

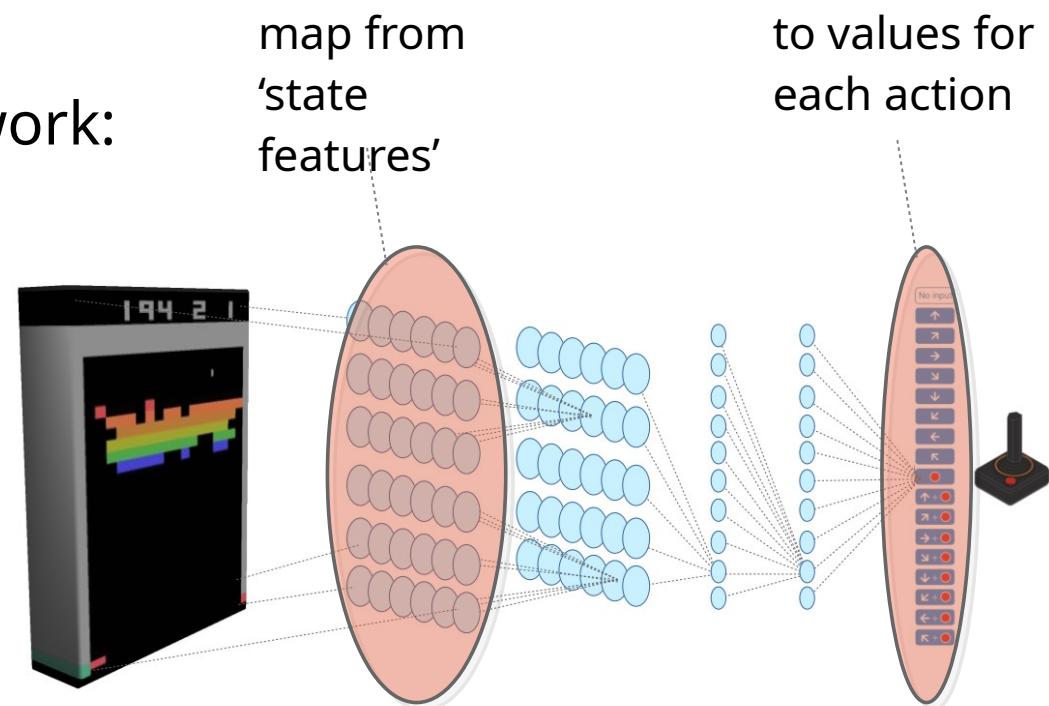
- but... MDPs are big!
  - (e.g., number of possible screens in Atari?)
  - how can we ever learn to play Atari games?



# Function Approximation, e.g., neural networks

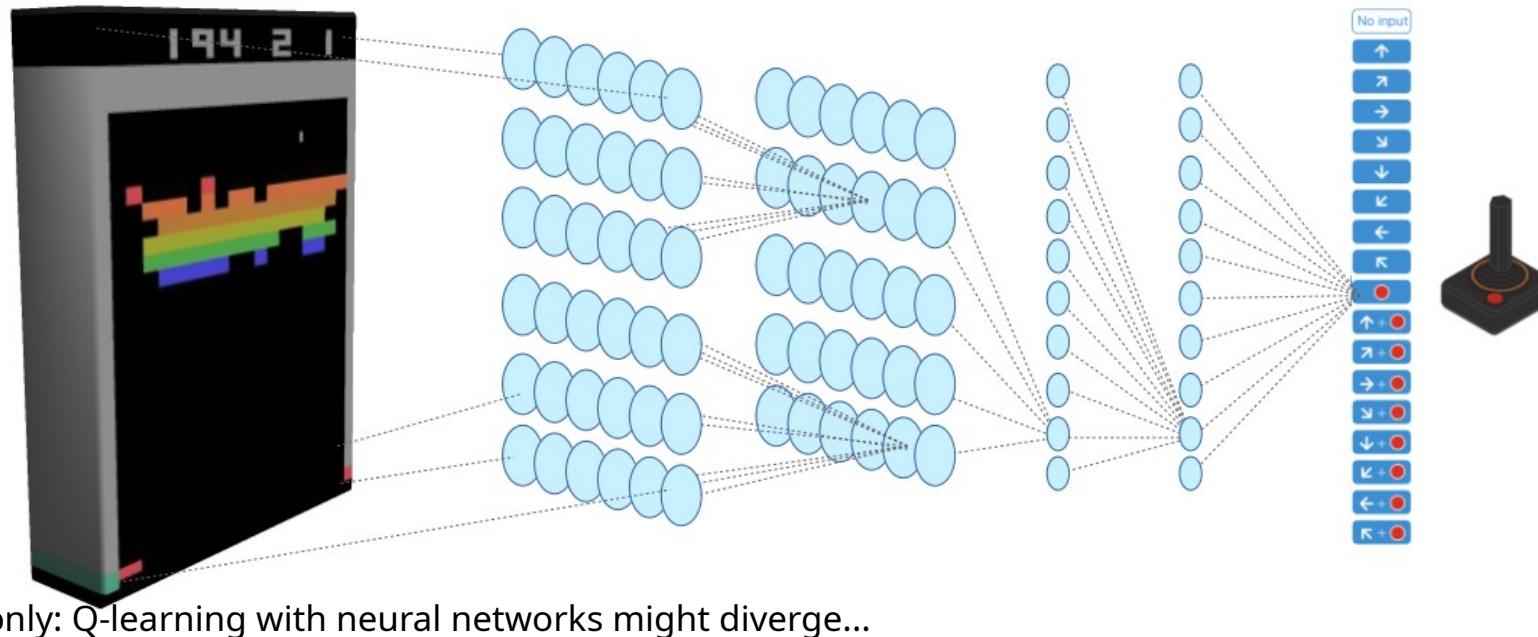
- In the end  $Q(s,a)$  is a function → approximate it

- E.g., with a deep neural network:



# Deep Q-Networks

- Prototypical example: DQN [Mnih et al. 2015]
- does precisely this:
  - Q-network: 84x84 image → 'action values'
  - Train with Q-learning



- only: Q-learning with neural networks might diverge...



Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." *Nature* 518.7540 (2015): 529.

# Tricks for Convergence

- Only: Q-learning with neural networks might diverge...  
→ so need to stabilize...
- DQN uses a number of techniques:
  - experience replay
  - ‘target network’
  - gradient clipping
- together, they lead to sufficient stability

## Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory  $D$  to capacity  $N$

Initialize action-value function  $Q$  with random weights  $\theta$

Initialize target action-value function  $\hat{Q}$  with weights  $\theta^- = \theta$

**For** episode = 1,  $M$  **do**

    Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequence  $\phi_1 = \phi(s_1)$

**For**  $t = 1, T$  **do**

        With probability  $\epsilon$  select a random action  $a_t$

        otherwise select  $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$

        Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$

        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $D$

        Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $D$

        Set  $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

        Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  with respect to the network parameters  $\theta$

        Every  $C$  steps reset  $\hat{Q} = Q$

**End For**

**End For**



# Limitation / Assumptions

# Non-stationary environments

- The transition function changes over time
  - the world or action effects in the world change
  - i.e., the transition function  $T(s,a,s)$  changes
- The reward function changes over time
  - The goal(s) change(s):  
the reward function  $R(s,a,s')$  changes
- Well... one big advantage of learning methods: they are adaptive
  - but learning rate can't go down to 0
  - need to keep exploring

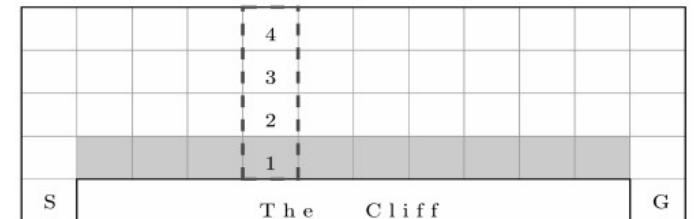
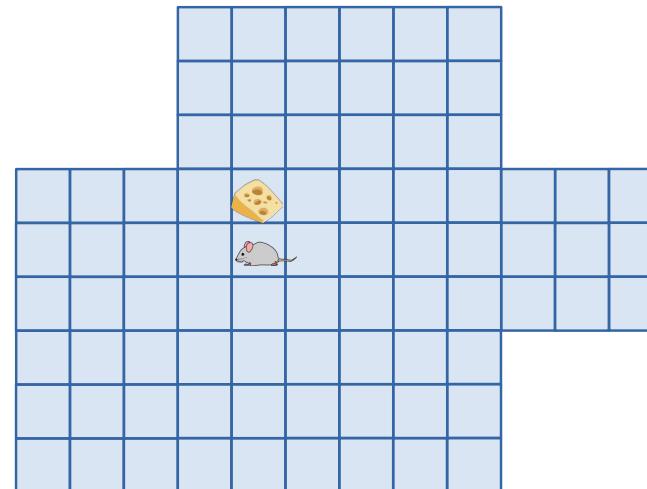
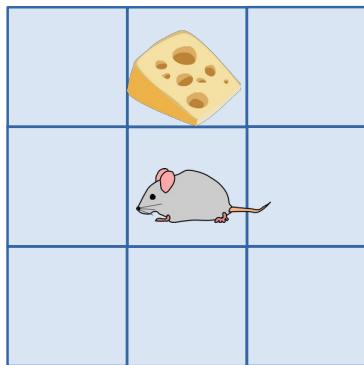


Figure 3: Cliff Walking scenario. Slippery path marked in grey (see text). The red box indicates the distance 4,3,2 and 1 to the Cliff depicted on the horizontal axis of figure 4.

## Markov assumption

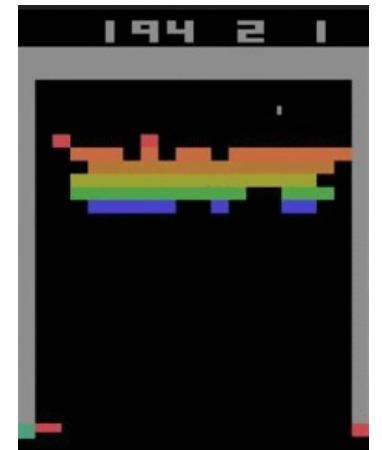
- What if the last state is not the only info needed to predict what happens after action  $a$  is executed?
- What if the world is partially observable?  
(non-markovian)



# Conclusions

## Conclusions: Part 2

- Learning from experience: Reinforcement learning (RL)
- Types of RL methods:
  - **policy evaluation** versus **control**
  - **model-based RL** and **model-free RL**
  - **on-policy** versus **off-policy learning**
  - **on-line** versus **off-line learning**
- Methods:
  - TD-learning, SARSA, Q-learning, Deep Q-networks
- Challenges:
  - the **exploration/exploitation** trade-off.
  - scaling up and **generalization**
  - **partial observability**



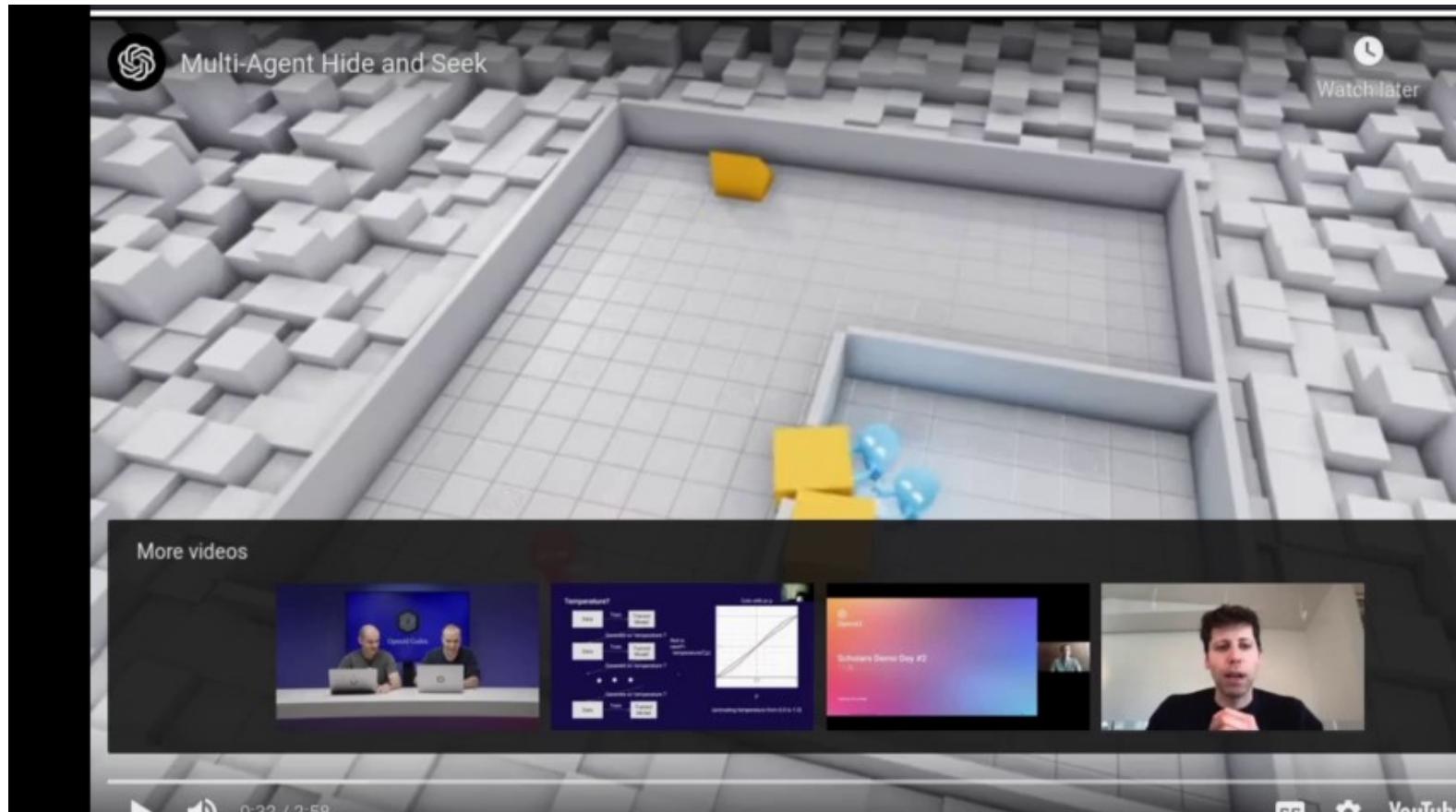
# Conclusions: RL

- What is sequential decision making? When applicable?
  - An RL problem can be defined using the **MDP framework**:
    - States, actions, transitions and rewards
  - **Planning vs reinforcement learning**
    - Planning methods (policy iteration, value iteration)
    - RL methods (TD-learning, SARSA, Q-learning, ...)
  - Challenges:
    - scalability: also for planning... often tackled *with* RL
    - RL challenges just mentioned
- **One of the most active field of machine learning currently.**
- Further reading / viewing:
    - RL Course by David Silver on youtube



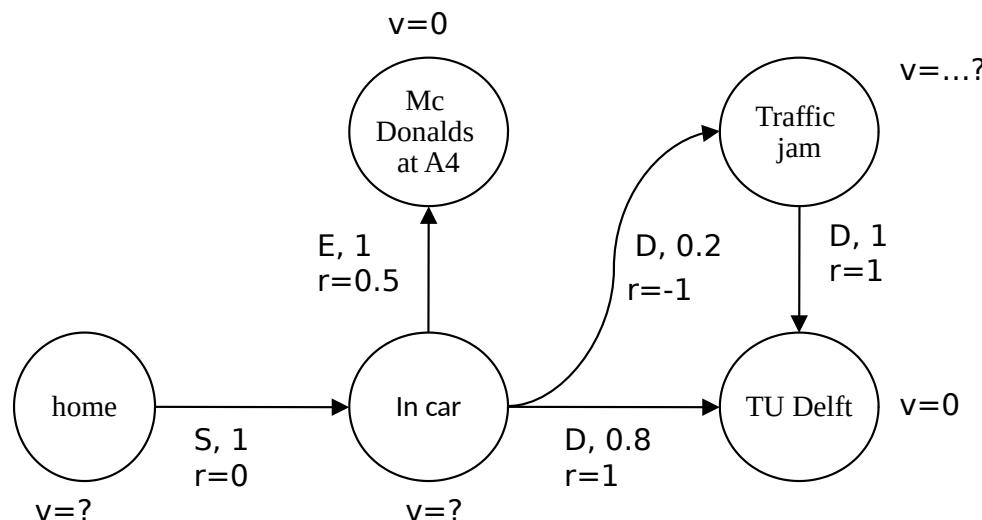
# Open AI's tool use

<https://openai.com/blog/emergent-tool-use/>



# Exercise: policy evaluation

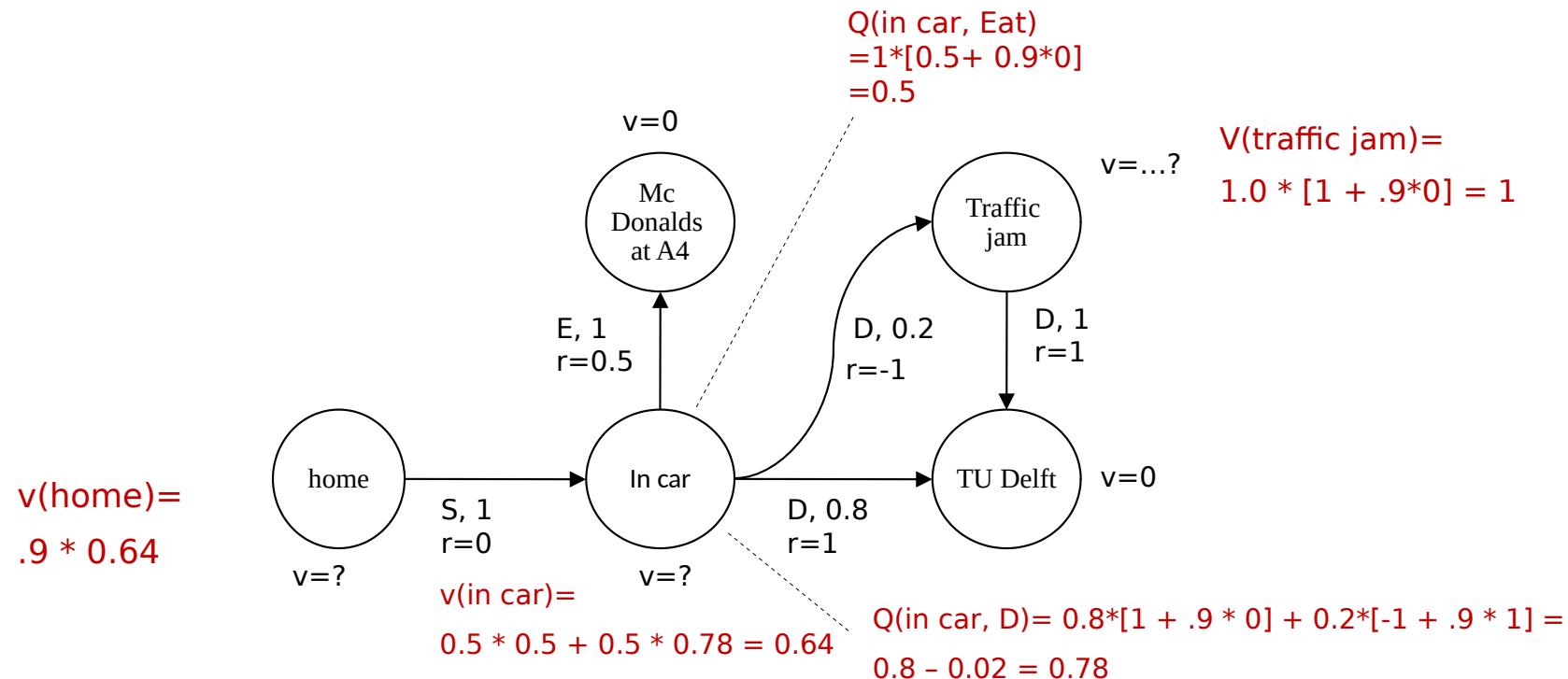
- 3 actions: S(tart), E(at), D(rive to work)
- Assume “random policy”  
(when multiple actions available, select one at random)
- nodes without outgoing arrows are terminal:  $v=0$
- Use Bellman for  $V(s)$ , assume  $\gamma=0.9$ .



$$v_{\pi}(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s, a) [r(s, a, s') + \gamma v_{\pi}(s')]$$

# Exercise: policy evaluation

- 3 actions: S(tart), E(at), D(rive to work)
- Assume “random policy”  
(when multiple actions available, select one at random)
- nodes without outgoing arrows are terminal:  $v=0$
- Use Bellman for  $V(s)$ , assume  $\gamma=0.9$ .

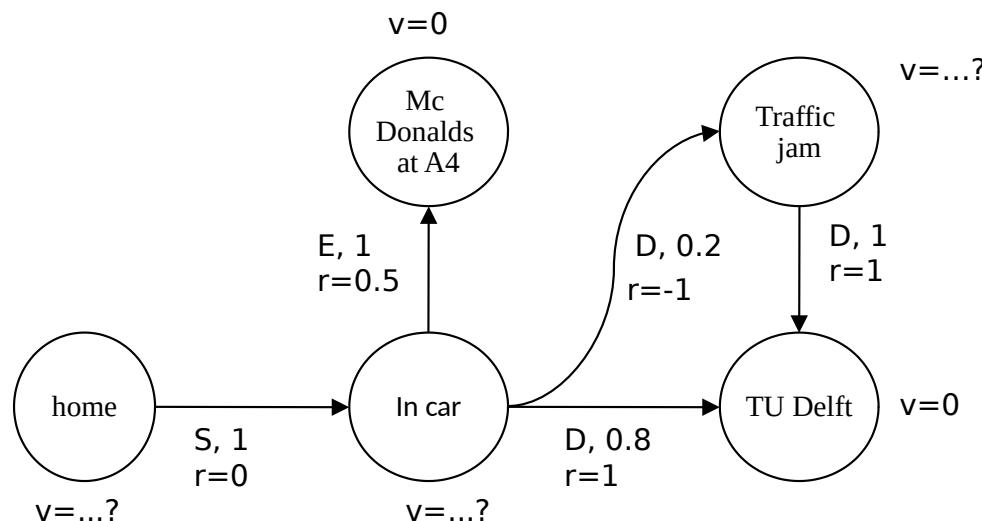


2

$$v_{\pi}(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s, a) [r(s, a, s') + \gamma v_{\pi}(s')]$$

# Exercise: optimal value function

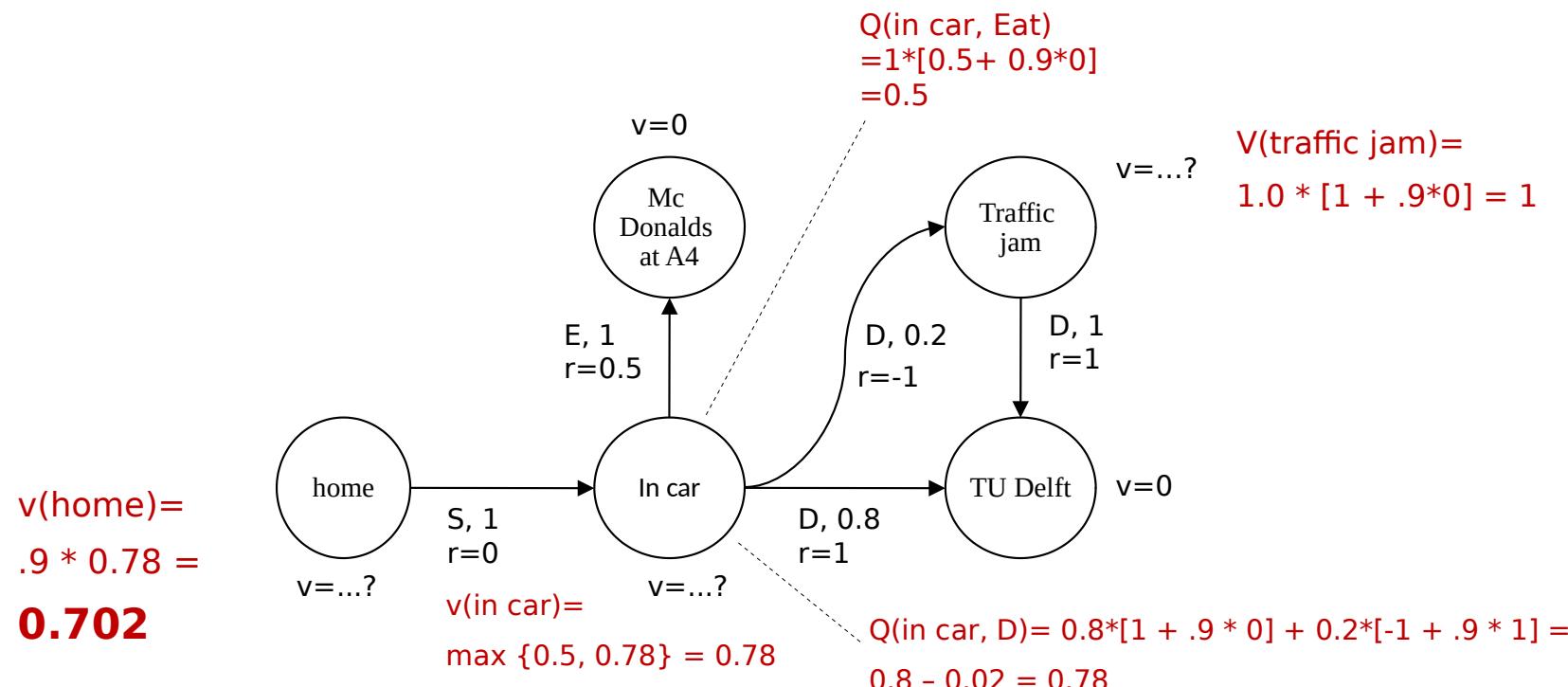
- 3 actions: S(tart), E(at), D(rive to work)
- nodes without outgoing arrows are terminal:  $v=0$
- Assume discount is 0.9.
- What is  $v_*(\text{home})$  ?



$$v_*(s) = \max_a \sum_{s'} p(s'|s, a) [r(s, a, s') + \gamma v_*(s')]$$

# Exercise: optimal value function

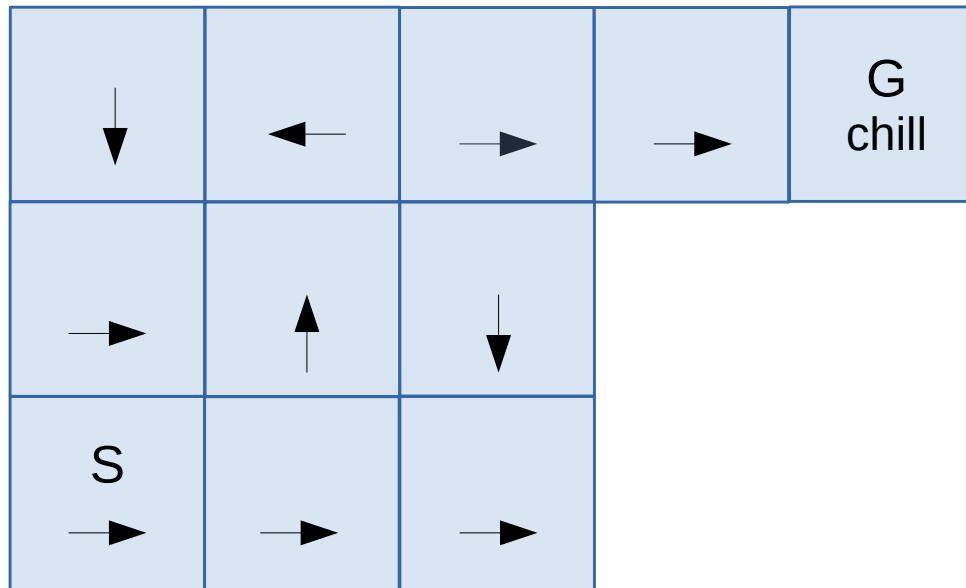
- 3 actions: S(tart), E(at), D(rive to work)
- nodes without outgoing arrows are terminal:  $v=0$
- Use Bellman for  $V(s)$ , assume  $\gamma=0.9$ .
- What is  $v_*(\text{home})$  ?



$$v_*(s) = \max_a \sum_{s'} p(s'|s, a)[r(s, a, s') + \gamma v_*(s')]$$

# Iterative Policy evaluation

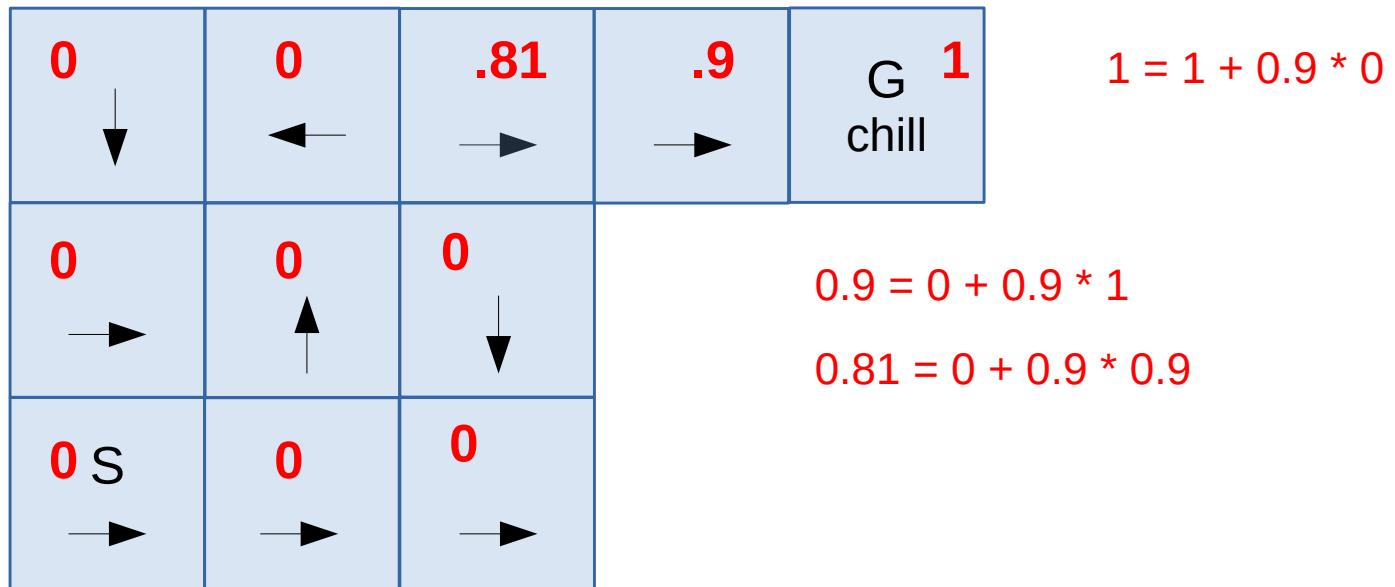
- Transitions are deterministic. At goal, the ‘chill’ action gives +1 reward. No other rewards. Discount is 0.9
- Use iterative policy evaluation (p.74 in S&B) on the following policy  $\pi$



- How many iterations ‘k’ do you need to compute  $v_\pi$ ?

# Iterative Policy evaluation

- Transitions are deterministic. At goal, the ‘chill’ action gives +1 reward. No other rewards. Discount is 0.9
- Use iterative policy evaluation (p.74 in S&B) on the following policy  $\pi$



- How many iterations ‘k’ do you need to compute  $v_\pi$ ? **3 iterations**

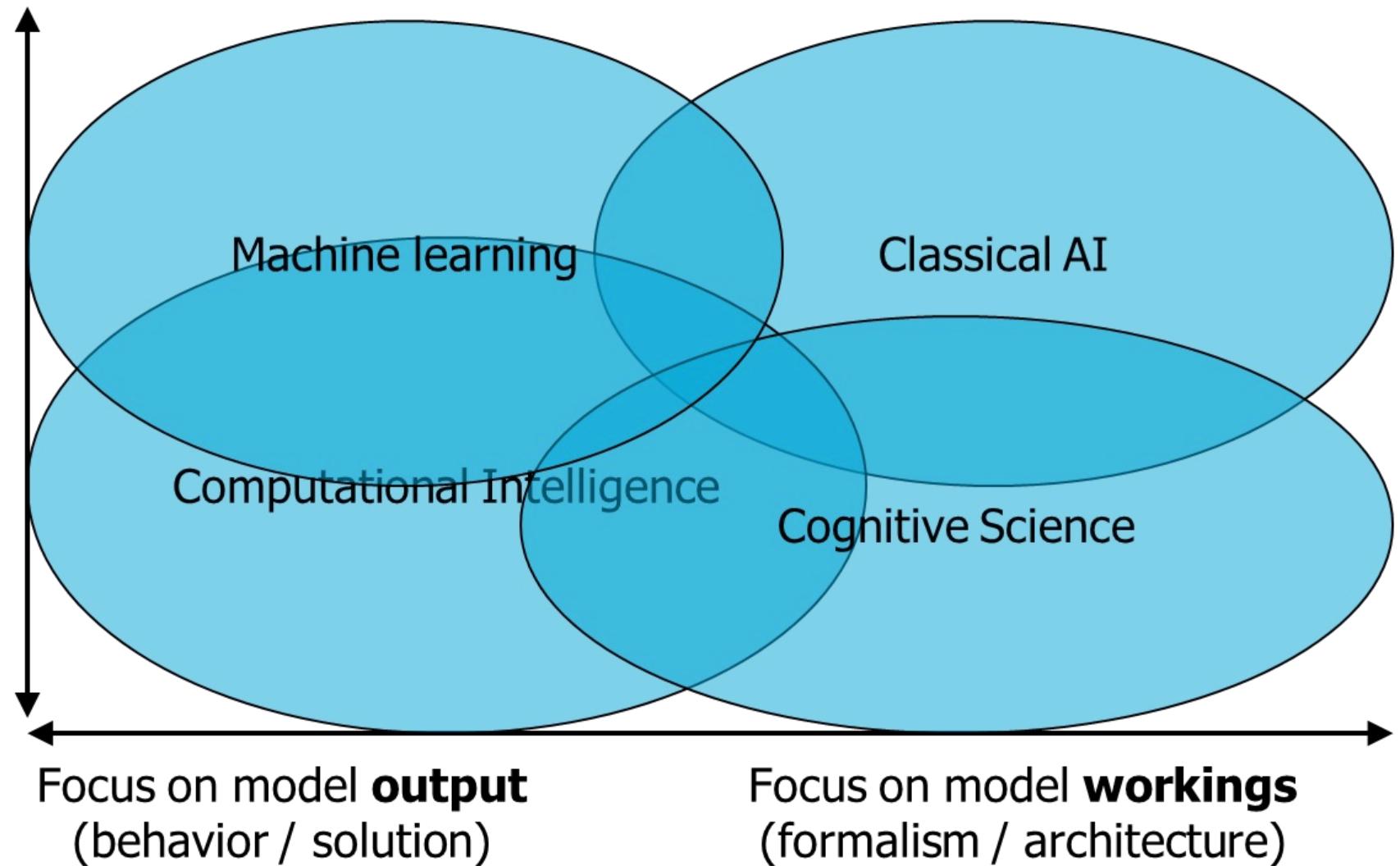
# CSE2530 Computational Intelligence

Luciano Cavalcante Siebert

(Partial credits to Joost Broekens, Judith Redi)



**Biologically  
irrelevant**



# We saw four perspectives on Computational Intelligence

The simulation of  
the workings of  
the **brain** (**MLP**)

The simulation of  
**evolution**  
(**Genetic  
Algorithms**)

The simulation of  
**(group) animal  
behavior** (**PSO  
and ACO**)

The simulation of  
**animal learning**  
(**Reinforcement  
Learning**)

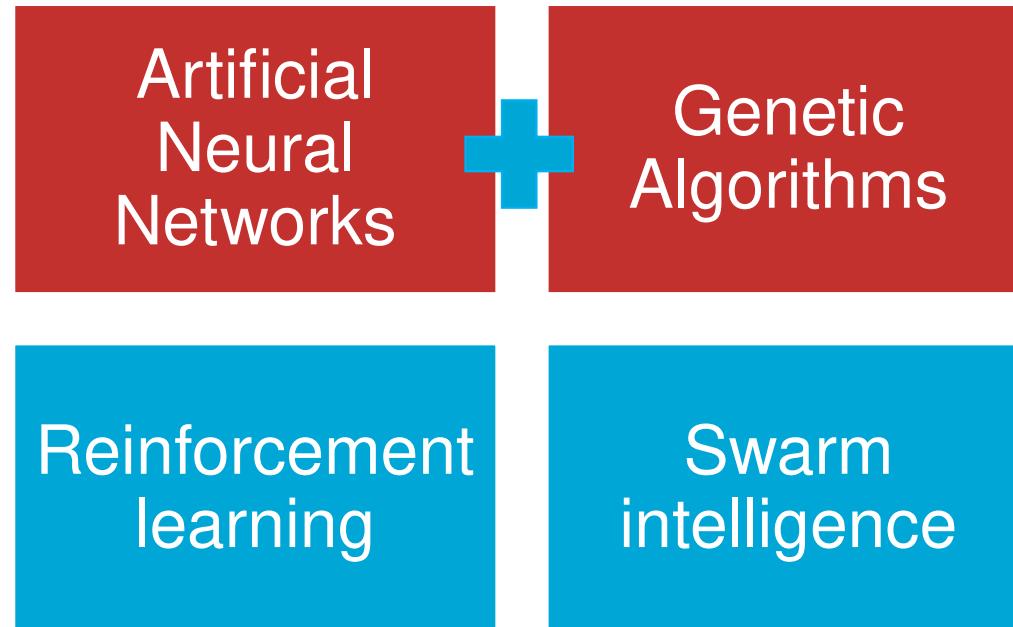
- These and other perspective can be combined!

# Mixing strategies

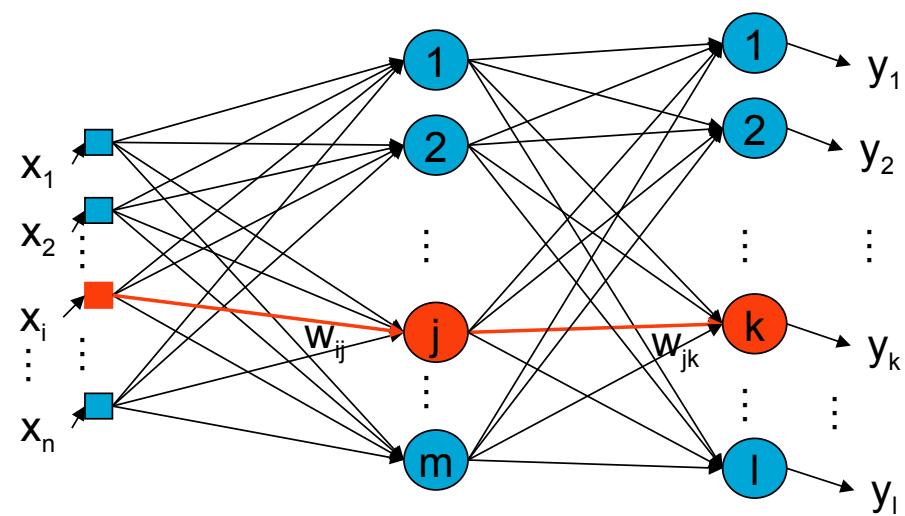
- Alleviate a shortcoming of one CI using another
  - Convergence, e.g.,
    - a method does not converge quickly or converges to a local optimum
  - State representation, e.g.,
    - states are continuous, or ,  
states share features
  - Fitness representation, e.g.,
    - fitness might be too locally defined, or,  
fitness can not be locally defined

These techniques can be combined in many different ways. In this lecture, we will see a few examples

# Mixing strategies

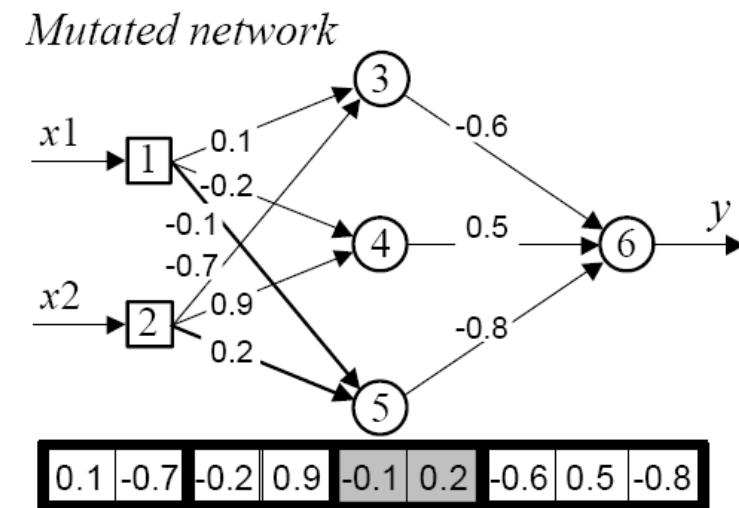
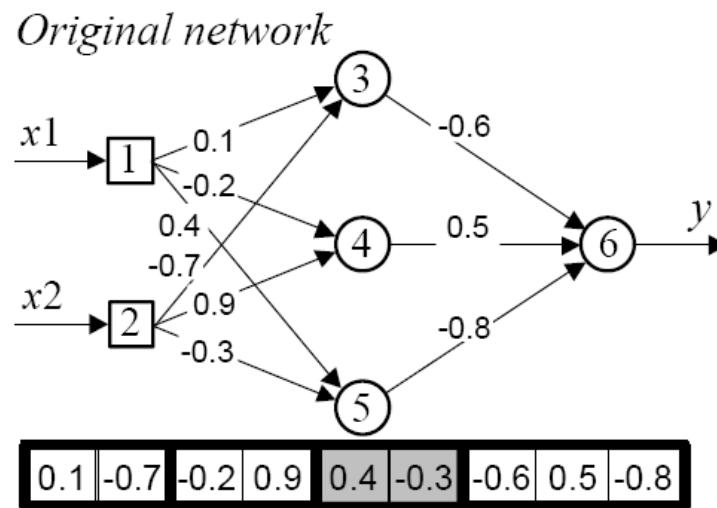


# MLP and genetic algorithms



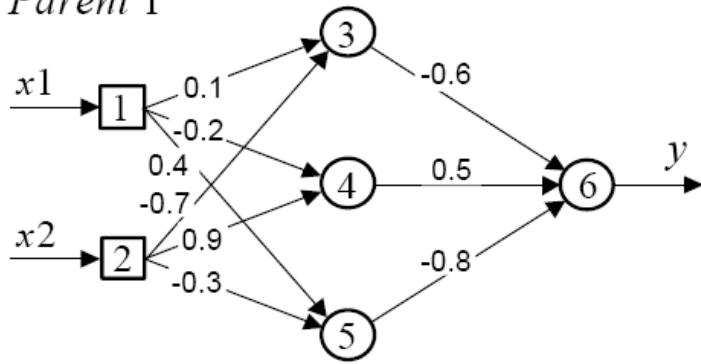
# Evolutionary neural networks

Evolving weights

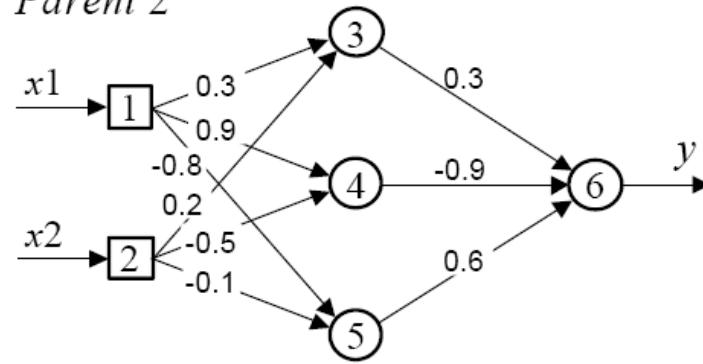


**Mutation:** for each gene, add gaussian noise to each weight

*Parent 1*



*Parent 2*

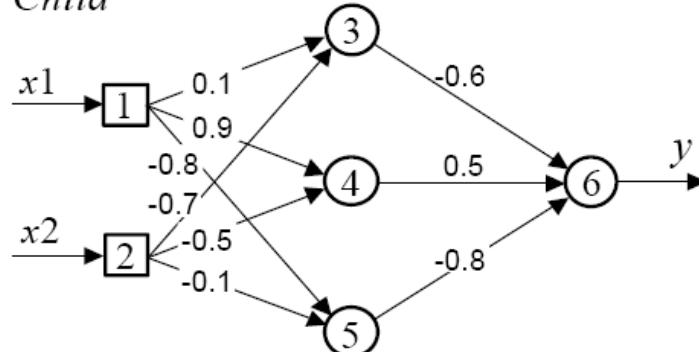


0.1	-0.7	-0.2	0.9	0.4	-0.3	-0.6	0.5	-0.8
-----	------	------	-----	-----	------	------	-----	------

0.3	0.2	0.9	-0.5	-0.8	-0.1	0.3	-0.9	0.6
-----	-----	-----	------	------	------	-----	------	-----

0.1	-0.7	0.9	-0.5	-0.8	0.1	-0.6	0.5	-0.8
-----	------	-----	------	------	-----	------	-----	------

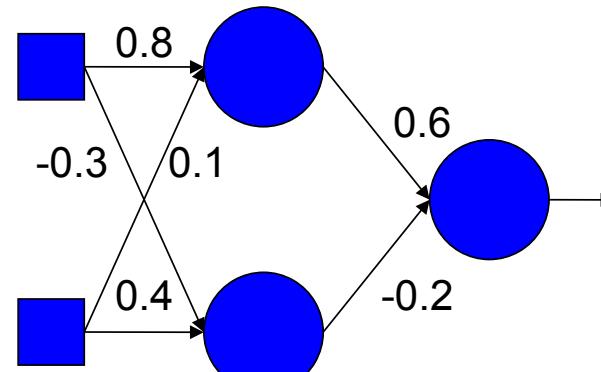
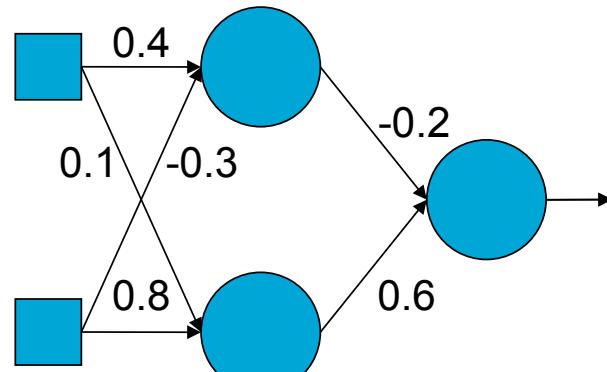
*Child*



## Crossover

# Cross-over: permutation problem

But, cross-over is usually inefficient for neural network weights...



0.4	-0.3	0.1	0.8	-0.2	0.6
-----	------	-----	-----	------	-----

0.1	0.8	0.4	-0.3	0.6	-0.2
-----	-----	-----	------	-----	------

0.4	-0.3	0.4	-0.3	0.6	-0.2
-----	------	-----	------	-----	------

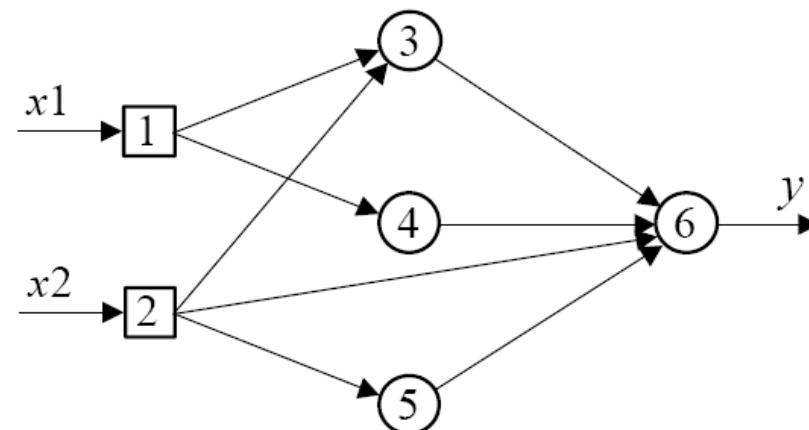
# Evolutionary Neural Networks

Can we do something also for the structure of the network?

# Evolutionary Neural Networks

Finding the optimal Network Structure

<i>From neuron:</i>	1	2	3	4	5	6
<i>To neuron:</i>	1	0	0	0	0	0
2	0	0	0	0	0	0
3	1	1	0	0	0	0
4	1	0	0	0	0	0
5	0	1	0	0	0	0
6	0	1	1	1	1	0



*Chromosome:*

0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	1	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

# Mixing strategies

Artificial  
Neural  
Networks

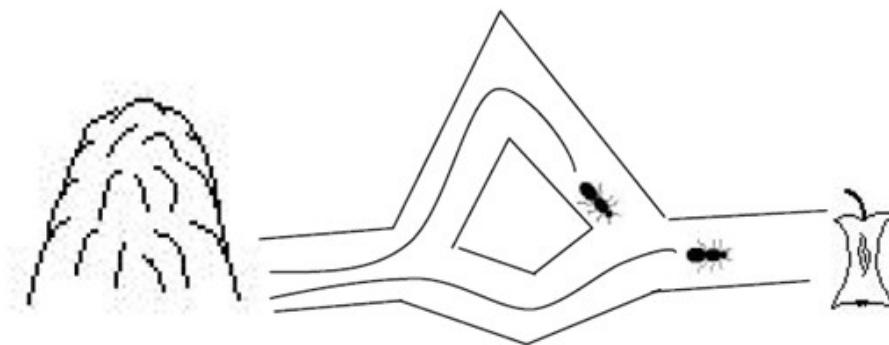
Genetic  
Algorithms

Reinforcement  
learning

Swarm  
intelligence  
(ACO)



# Ant colony optimization and reinforcement learning



# RL-based pheromone dropping

- We could combine Q-learning

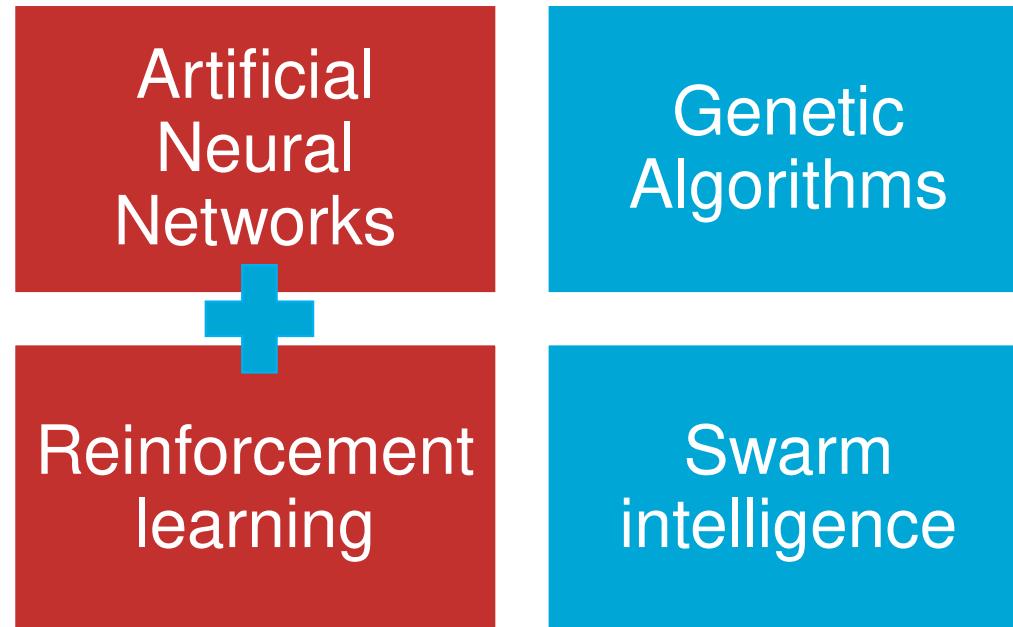
$$Q(s, a) \leftarrow Q(s, a) + \alpha (R(s, a) + \gamma \max_{a_{\max}} Q(s', a_{\max}) - Q(s, a))$$

- with pheromone dropping, we get:

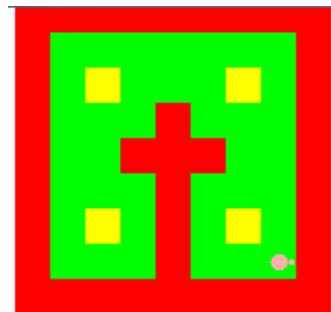
$$\tau_{ij} \leftarrow (1 - \alpha) \cdot \tau_{ij} + \alpha \cdot \left( \Delta \tau_{ij} + \gamma \cdot \max_{l \in N_j^k} \tau_{jl} \right) \quad \forall (i, j) \in E$$

- Effectively adding an anticipation of future “reward”, and introduces a discount and learning rate.
- Applied during solution construction based on MAX transition

# Mixing strategies



# Reinforcement learning and Artificial Neural Networks

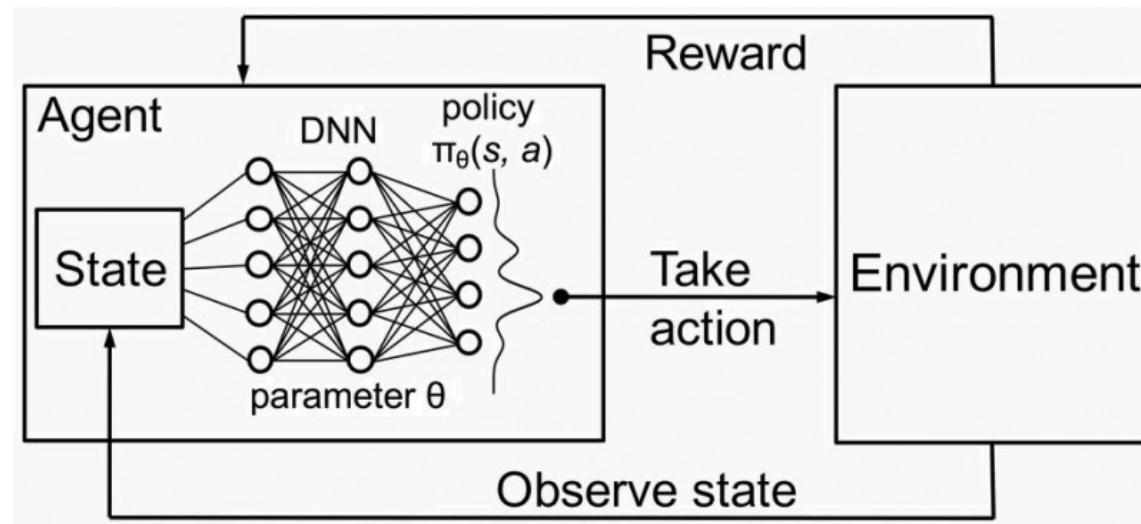


# Real world state spaces are large

- States and action spaces can be very large
- States are composed out of features
  - What is “road” other than a *hard, dark, flat, outside* surface
- States resemble each other->**generalization over states**
  - A *floor* is like a *road* (hard, flat, color independent, *inside*)
- Assumption: states that **resemble each other (share features)** are **like each other in terms of consequences**
  - I can drive on a *road* and a *floor*.
- However, be careful!! This might be **very wrong in some cases!**

# Reinforcement learning and Artificial Neural Networks

- Can we use an MLP to help RL large set of states, in situation where Q-learning does not perform adequately?
- In particular, how can we train an MLP to predict the:
  - $Q(s, a)$  function.



# Reinforcement learning and Artificial Neural Networks

- For more details:
  - Sutton and Barto, section 9.7 “Nonlinear Function Approximation: Artificial Neural Networks”
- Some algorithms:
  - Deep Q-learning (DQL) <https://proceedings.mlr.press/v120/yang20a.html>
  - Proximal Policy Optimization (PPO) <https://arxiv.org/pdf/1707.06347.pdf>
  - Trust Region Policy Optimization (TRPO)  
<https://arxiv.org/pdf/1502.05477.pdf>

Fan, J., Wang, Z., Xie, Y., & Yang, Z. (2020, July). A theoretical analysis of deep Q-learning. In *Learning for Dynamics and Control* (pp. 486-489). PMLR.

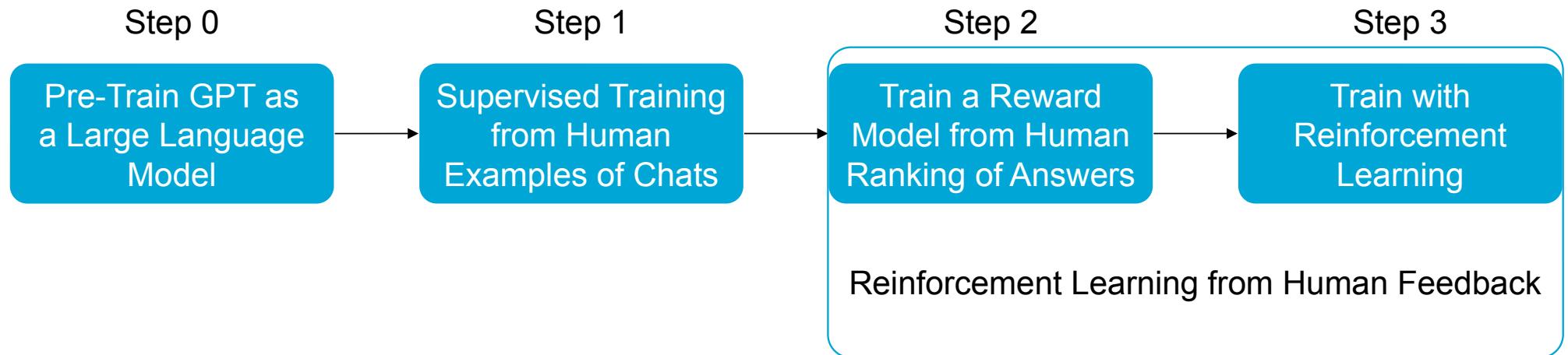
Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015, June). Trust region policy optimization. In International conference on machine learning (pp. 1889-1897). PMLR.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

ChatGPT



# ChatGPT Training Procedure



## Step 0: GPT as a Large Language Model (LLM)

GPT-3 is a **Large Language Model (LLM)**, trained with self-supervised learning with the task of predicting the next word in a sentence. Thus, no human labels are required for this training step.

GPT-3 has 175 billion parameters and it is trained on 45TB of text data, including Wikipedia, Reddit, and news articles.

# Step 1: Supervised Learning from Human Examples

## STEP 1

### Collect demonstration data and train a supervised policy

A prompt is sampled from our prompt dataset

A labeler demonstrates the desired output behavior

This data is used to fine-tune GPT-3.5 with supervised learning.

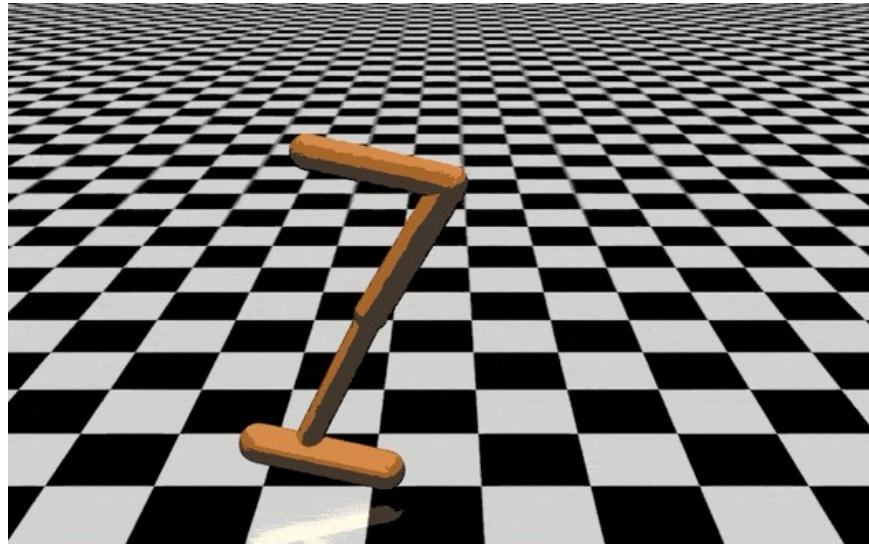


<https://www.aivo.co/blog/chatgpt-training-process-advantages-and-limitations>

# Reinforcement Learning from Human Feedback

Reinforcement Learning from Human Feedback (RLHF) is used when it is too difficult to mathematically model a reward function. Instead, human feedback is used as reward signal.

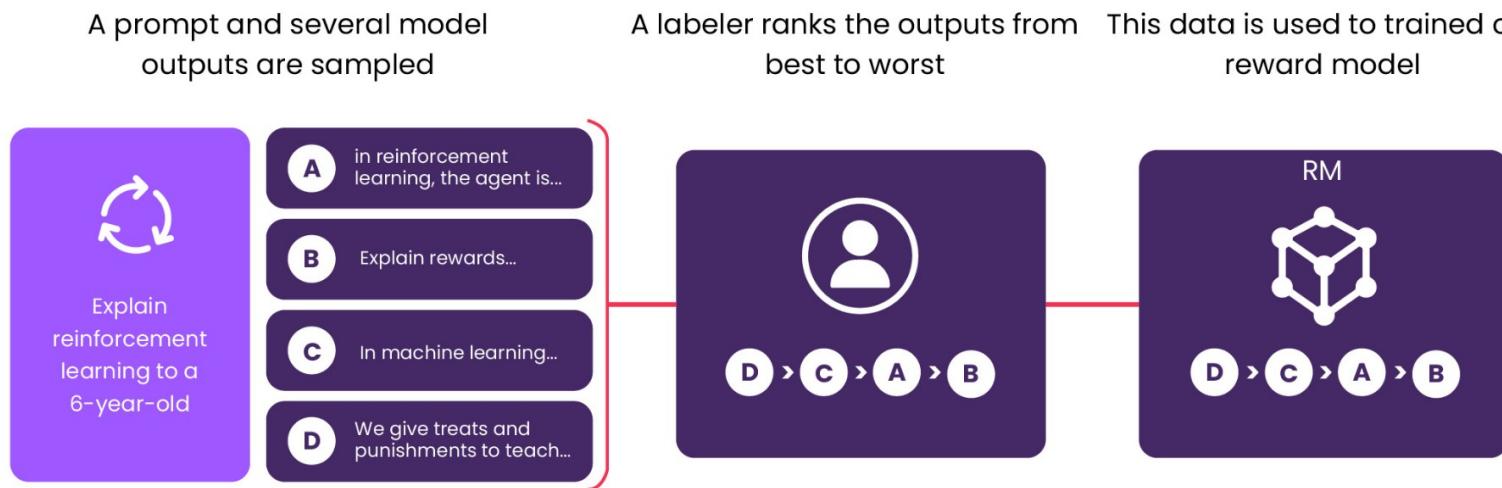
This method was introduced to teach to a virtual stickman to do a back-flip.



# Step 2: Train a Reward Model

## STEP 2

### Collect comparison data and train a reward model

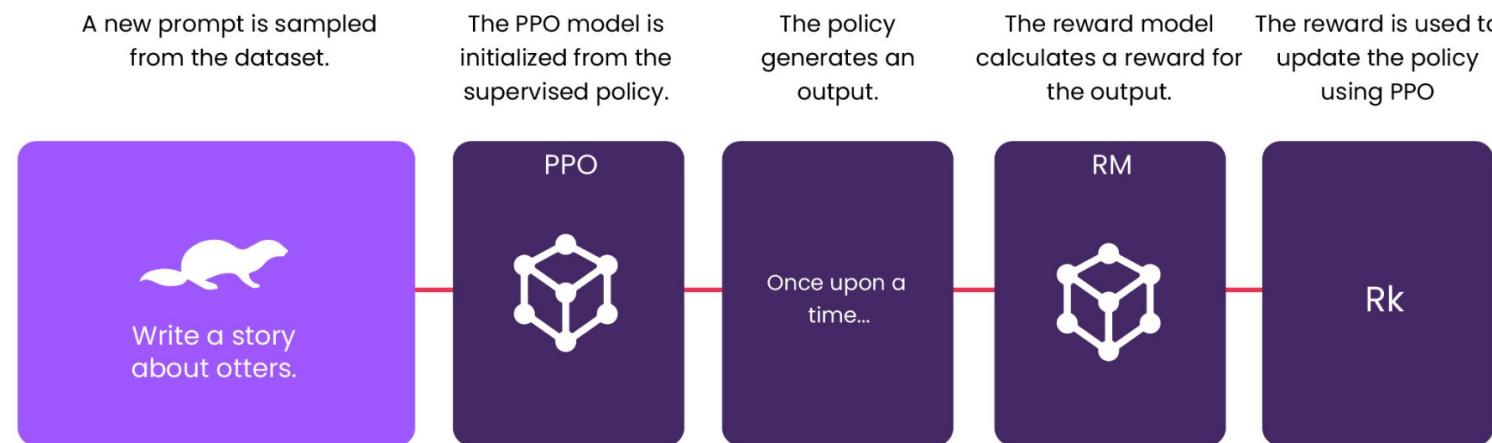


<https://www.aivo.co/blog/chatgpt-training-process-advantages-and-limitations>

# Step 3: Train with Reinforcement Learning

## STEP 3

**Optimize a policy against the reward model  
using the PPO reinforcement learning algorithm**



<https://www.aivo.co/blog/chatgpt-training-process-advantages-and-limitations>

# Why is ChatGPT so good?

The technology behind ChatGPT is **not new** (it was invented in 2017).

What makes it special is:

- The **size** (175 billion parameters);
- The amount of **human annotations** (which is not disclosed);
- The **computing power**: GPT-3 was trained on 1000 parallel GPUs for 34 days (for the estimated cost of \$5M). The training time of ChatGPT is not disclosed.

# What is ChatGPT (not) optimized for?

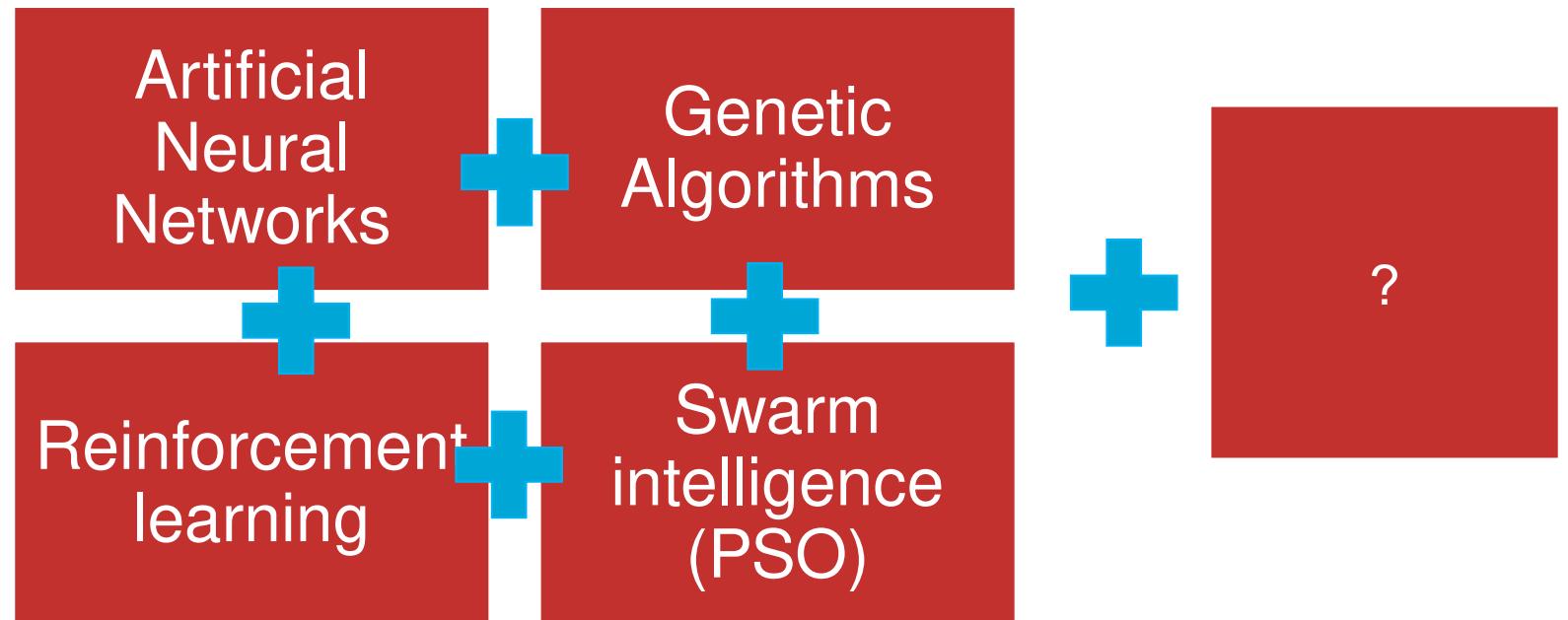
Due to its training, ChatGPT is optimized for:

- Maintaining a **good conversation**;
- Being **human-like**, kind and friendly.

It is **not** optimized for:

- **Being correct.** It does not index any existing source, it simply completes a sentence (text auto-completion on steroids).

# Mixing strategies



# Learning outcomes

After completing this course, you should be able to:

- LO1: Position Computational Intelligence (CI) as a field in AI
- LO2: Explain concepts such as problem space, fitness, and optimization
- LO3: Understand the working and limitations of CI techniques
- LO4: Select the most appropriate CI technique for a given problem
- LO5: Apply Reinforcement Learning techniques for a given problem
- LO6: Apply different types of neural networks for a given problem
- LO7: Apply evolutionary and swarm techniques to solve given problems

# CSE2530 Computational Intelligence

Luciano Cavalcante Siebert

(Partial credits to Joost Broekens, Judith Redi)



# CSE2530 Computational Intelligence

Sietze Kai Kuilman



Lecture: The Frame Problem

# Today's topic

- A small history lesson
- Some philosophy
- A bit of critical thinking
- A short game

# Main takeaway

- At the end of this lesson you'll be able to explain the frame problem.

# A small history lesson

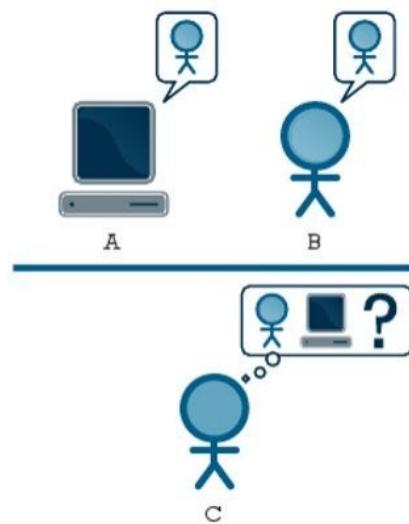
- Artificial intelligence was coined as a term at the Dartmouth conference of 1956
  - It was discussed under the following premise: Any aspect of intelligence can in principle be simulated in a machine.
  - This premise is still with us today.

# Taking a step back

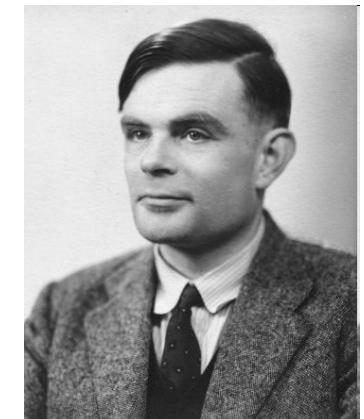
- Well, what is required for a system to be intelligent?

# Can machines think? A practical approach

- Turing, 1950: that's an ill-posed problem
- Let's rather frame it in terms of an **imitation game**:



A computer or a human is interrogated by a participant through a teletype (chat). The interrogator guesses if there is a computer or a human at the other end of the terminal.



Alan Turing  
1912-1954

# The Turing test

- To pass it, a machine needs to be able to
  - Process natural language (recognize and interpret it)
  - Represent knowledge (to store and access info)
  - Reason (to draw conclusions)
  - Learn (to adapt to new circumstances by examining its previous experiences)
- Or not? Is just a subset of the above enough?

# Does passing the Turing test make a system intelligent?

- The Turing test can promote:
  - Mistakes
  - Subterfuge
  - Behaviour over intention
- The Turing test was passed by a robot who pretend to be a Ukrainian 13-year old boy.
  - He spoke in broken English and misunderstood most questions
  - The test is designed to mislead the interrogators, rather than be intelligent

# When is behaviour correct?

- In computational intelligence we focus on the output of the model.
  - Is that sufficient?
- *Remember lecture 1:*
- In CI we are somehow **optimizing a function** which corresponds to intelligent behavior.

# Cats and tigers

- Let's train a very simple neural network
  - Such that it can classify cats and tigers
  - It only has these (kind of) pictures



# Cats and tigers

- Classified as: ?



# Cats and tigers

- Classified as: Cat?
  - Of course depends on a variety of things, but snow may be a problem



# Cats and tigers

- See?



# Cats and tigers

- See?

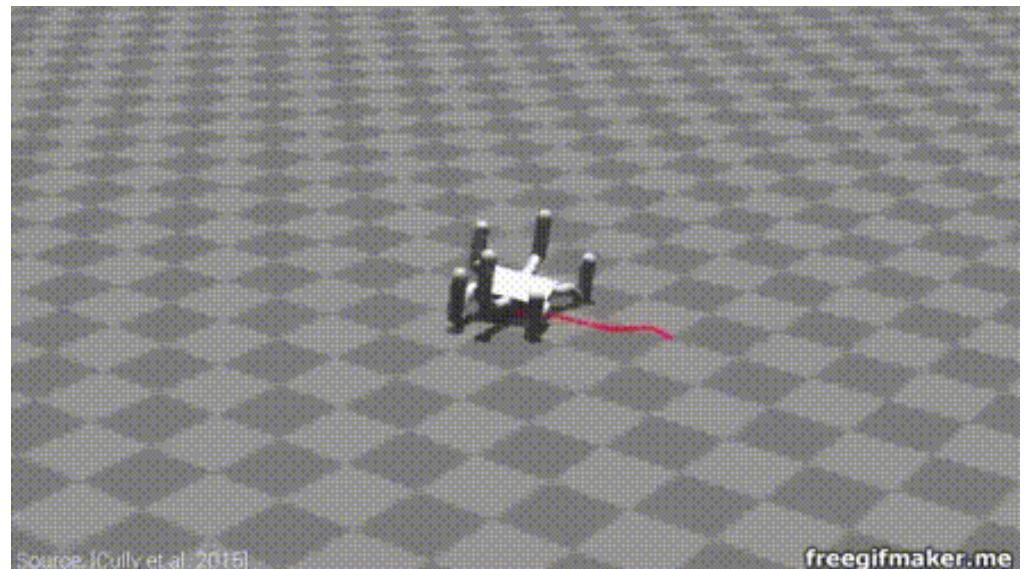


Snow  
No snow

## Another example

- Let's make a bot which we want to walk well
  - We implement a reward function that awards the robot for each second its feet are not on the ground
  - What goes wrong?

# Walking upside down



Source: [Cully et al. 2015]

freegifmaker.me

# Dang, why does this happen?

- Wait wait wait. Behaviour is important.
  - But are mistakes okay?
    - And if they are, what mistakes are okay?
  - Why is subterfuge okay?
    - And if it is, under which circumstances?

# Reward hacking

- Without looking at intention we open ourselves up to issues of reward hacking
  - Reward hacking: finding a different means towards the intended goal.
  - (Remember: Fitness function hacking)
- *Remember lecture 1:*
- Implementation is not as easy as it seems
  - You will not notice or disregard relevant features, data, inferences, connections

# Relevancy

- Our data set lacked the relevant materials to let a neural network correctly classify A and B
- Our bot didn't understand it was relevant to walk on its feet
- It (and perhaps I) didn't understand all the *relevant* things
  - What is relevant in a situation?

# What is relevant for relevancy?

- A simple definition:
  - *Something is relevant to a task if it increases the likelihood of accomplishing the goal which is implied by the task*

# What is relevant?

- We need a **something**
- A **goal**
- Some set of **tasks**



- The **goal**:
  - ?
- **Tasks**:
  - ?
- **Somethings**:
  - ?

# What is relevant?

- Do we need additional contextual information?
- How well specified does it need to be?
- What about additional common knowledge?

# You've got to be kidding right?

- Well what is important to know?
  - Is it okay to not even consider things?
  - Are there things you don't consider that you should?
    - Say a peanut allergy?

# Hold up!

- Don't we just need to model the stuff that matters?
  - Well: which stuff?



# I don't want your boring philosophy

- In CI we are specifically looking at output of a model
- How do you know your output is correct?
  - You may have to look at what is inside.
  - The output may be achieved through reward hacking without you even knowing it.





# Now what?

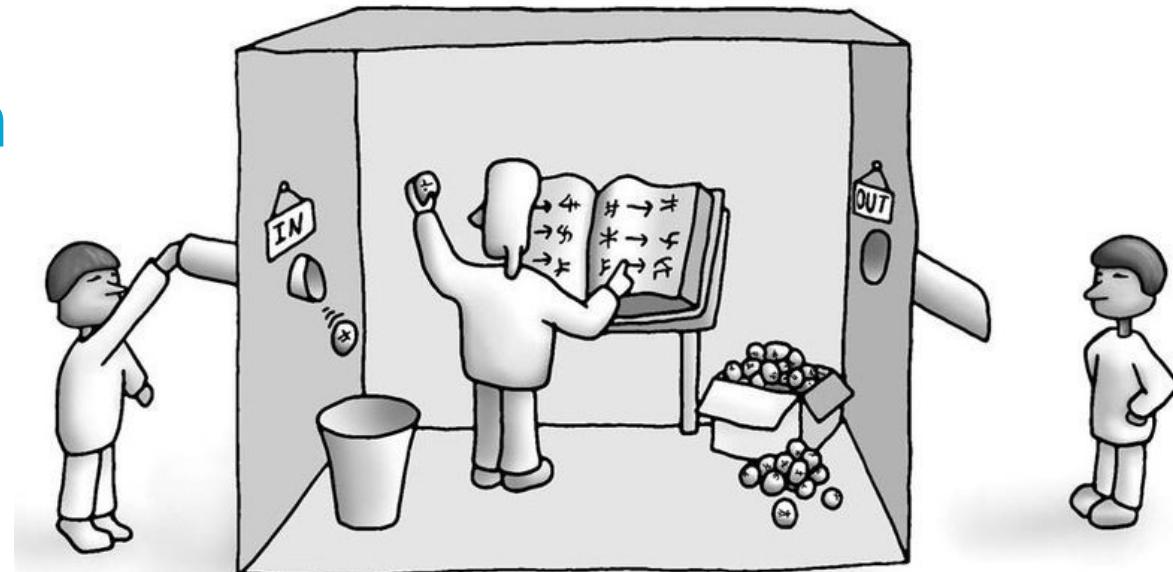
- Before the break
  - We talked about output and relevance
- Now:
  - What can we learn if we look beyond output
  - And what is relevant about this relevance?

# Inside the machine

- Why does the inside matter if you have good results?
- Let me pose a thought experiment to you

# Inside The Chinese Room

- Say I don't read Mandarin Character but I do have a big book of rules and I am in a closed room.
- Those rules make me adapt symbols which leads to a particular output.
- That output seems to be correct.
- I do it at such a speed that any onlooker is going to argue there is a Chinese speaker inside.
- Do I understand Mandarin?

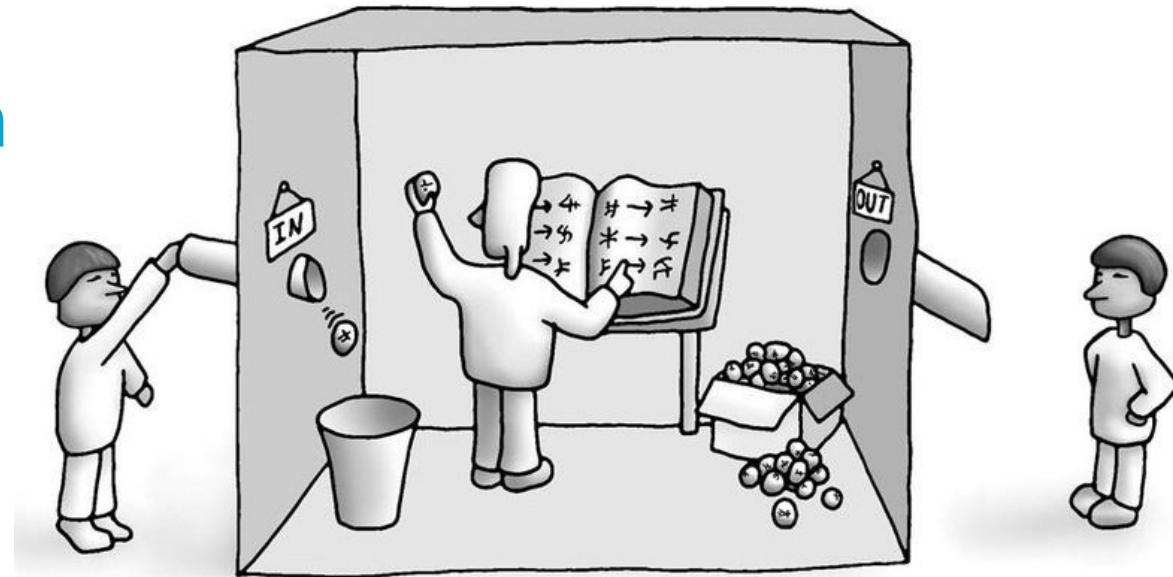


<https://analyticsindiamag.com/chinese-room-experiment-genereal-ai/>



# Inside The Chinese Room

- “Intelligent” **behavior** does not imply the machine is **conscious** and **understanding** in a human sense
- But if I don’t know the difference
  - Does it matter?



<https://analyticsindiamag.com/chinese-room-experiment-genereal-ai/>



# Understanding?

- Remember Lecture 1:
- If we choose a poor error function and obtain unsatisfactory results, **the fault is ours** for badly specifying the goal of the search.
- If it doesn't understand in a *human sense*, then what does it understand?
- And when are we not at fault?
  - What if, after ten years, it is revealed that our designed Chinese Room did contain mistakes?
  - And that implementation had impact on a whole host of things?

# Subterfuge

- If intelligence and behaviour are linked:
  - We need to know when is our behaviour correct.
- Does looking inside the machine help?
  - Well it can help understand what goes wrong.
  - But when are we satisfied?

# The Frame problem

- The Frame Problem: Hamlet's problem for engineers
  - When do we stop thinking?
- The Frame Problem: When have we reasonably gone over enough situations and data such that the machine acts according to our intended design?
- What's the problem?
  - Well, when is it sufficient?

# The Frame problem

- One answer may be: when we have taken into account all the *relevant* things.

# The Frame problem

- Remember:
  - *Something is relevant to a task if it increases the likelihood of accomplishing the goal which is implied by the task*
- So if we want to stop thinking we first need to know the goal and the tasks, such that we can take into account all the relevant things.

# The Frame problem

- It is actually immensely difficult to solve technically
  - Reason being, we can basically resort to two strategies:
    - Overreacting
    - Overthinking

# Overreacting

- Consider we have a small robot R1
  - Who wishes to save his precious life source: a battery on a cart
  - There is a bomb in the room.



# Overreacting

- Consider we have a small robot R1
  - PULLOUT (cart, room, t)



# Overreacting

- The bomb was also on the cart
  - OOPS!



# Overreacting

- The robot R1 overreacted by not taking into account all the relevant details of the situation
- It acted on limited knowledge and failed

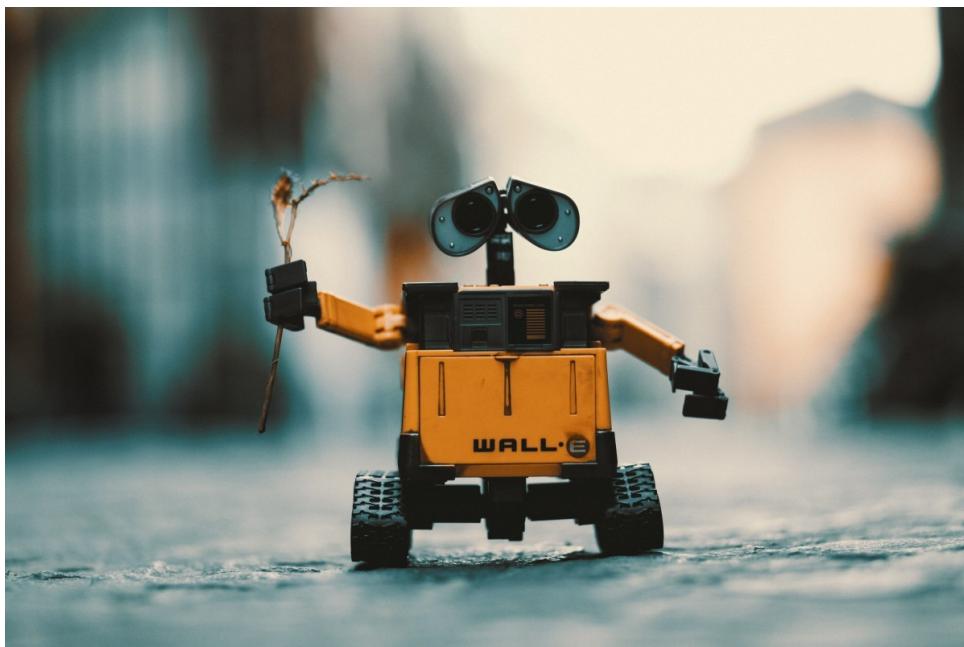
# Overthinking

- Consider the next robot R1D1
  - The bomb is still on the cart
  - R1D1 “thought” about it.



# Overthinking

- Consider the next robot R1D1
  - But at that moment the robot needed to know
    - What would change if I pulled the cart out?



# Boom.

- The bomb exploded
  - While R1D1 was still deducing



# Overreacting and Overthinking

- Overthinking is unlikely
- But overreacting is rather the norm

# The Frame problem

- The Frame problem is not a problem with data or logical inferences
- Our models just don't necessarily align with the world
  - And why would they?
  - We need to cut corners

# Next steps?

- What should R2D2 do?

# The Frame Game

- Talk to one of your neighbors
- Each take turns playing the robot – you say what you do(e.g. I pullout the cart)
- The other party tells you why you still didn't save your precious battery (e.g. the bomb was actually on the cart!)
- Then swap rolls, however you must now take into account what has happened before (e.g. the bomb may be on the cart. So check for it!)



# The Frame Game

- What was the weirdest thing that happened?

# The Frame Game

- Models cut corners in favor of tackling a problem
  - This is good for general cases
  - But it can blow up in your face (literally)
- Food for thought:
  - Under what condition is your implementation satisfactory?



# Questions?

But then there comes a time when the machine begins to dictate to you. For example; I have bought this wonderful machine – a computer. Now I am rather an authority on gods, so I identified the machine – it seems to me to be an Old Testament god with a lot of rules and no mercy.

– Joseph Campbell, The power of myth p.36



# Optional Slides

# Intelligence. Again?

- So maybe output is limited to talk about intelligence
- Yet it is also hard to know when all the relevant things are in the model.
  - It seems rather impossible.
- What's the actual solution, Mr Philosopher?
- For starters acknowledge that we are limited
  - Maybe we did not take someone into account
  - Maybe we overlooked some aspect
  - Its just a model, not ground truth

# Relevancy. Again?

- Because we create these agents that act on relevant things, and we are limited, we should actually understand relevancy for agents as follows:
- Something is deemed **relevant** to a task, if a set of designated individuals think - during the initial design process or feedback - it impacts the accomplishing of their conceptualization of the goal.
- As a potential solution, we should be able to disagree with the individuals and their conception of the goal, maybe even the somethings or the task.

# Contestability

- What can we do? Provide:
  - means of adjusting the models output (goal)
  - means of updating factors (somethings)
  - Means of altering the way it approaches said goal (tasks)

# Contestability

- We need to give users options to contest design
  - To disagree with a result and to make sure the machine does not do this again.
  - Call it: the right to alteration
- And you want to think about this!

# Childcare benefit scandal

- Caused in part by
  - No contestation
  - Discriminatory algorithms



mystic\_mabel -  
<https://www.flickr.com/photos/parkris/3672434417/>

# Boing 737 crash

- In part caused by
  - Overcorrection by AI
  - Pilots could not effectively intervene

