Programming, numerics and optimization

Lecture C-5: Heuristic methods

Łukasz Jankowski
ljank@ippt.pan.pl

Institute of Fundamental Technological Research
Room 4.32, Phone +22.8261281 ext. 428

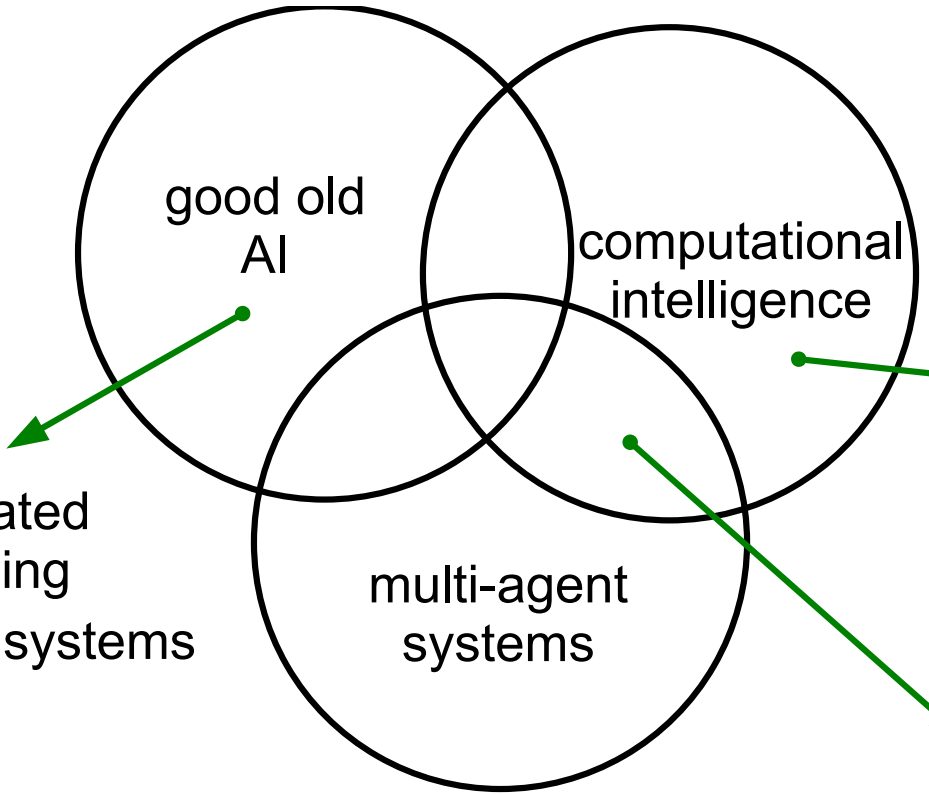
June 8, 2021[1]

[1] Current version is available at http://info.ippt.pan.pl/~ljank.

Outline

Outline

Branches of AI



- automated reasoning
- expert systems
- ...

- evolutionary algorithms
- fuzzy systems
- neural networks
- ...

- swarm intelligence
- ...

Continuous and discrete problems

Two basic types of optimization problems

continuous continuous domain (search space)

discrete discrete domain (combinatorial problems)

The linear programming lies somewhere in the middle:

- this is obviously a continuous problem
- but the potential solutions form a discrete network (or graph) of vertices connected by edges, which are traversed by the simplex method.

Hard and easy problems

The discrete optimization problems are often hard in general, which is usually measured in the terms of their

time complexity

the time[a] necessary to solve the problem in dependence on its size. Problems with

- polynomial time complexities are considered easy,
- exponential (factorial, etc.) time complexities are considered hard.

[a] That is, the number of steps of a Turing machine, which is an abstract symbol-manipulating device that basically models a computer.

Another measure is the *space complexity*.

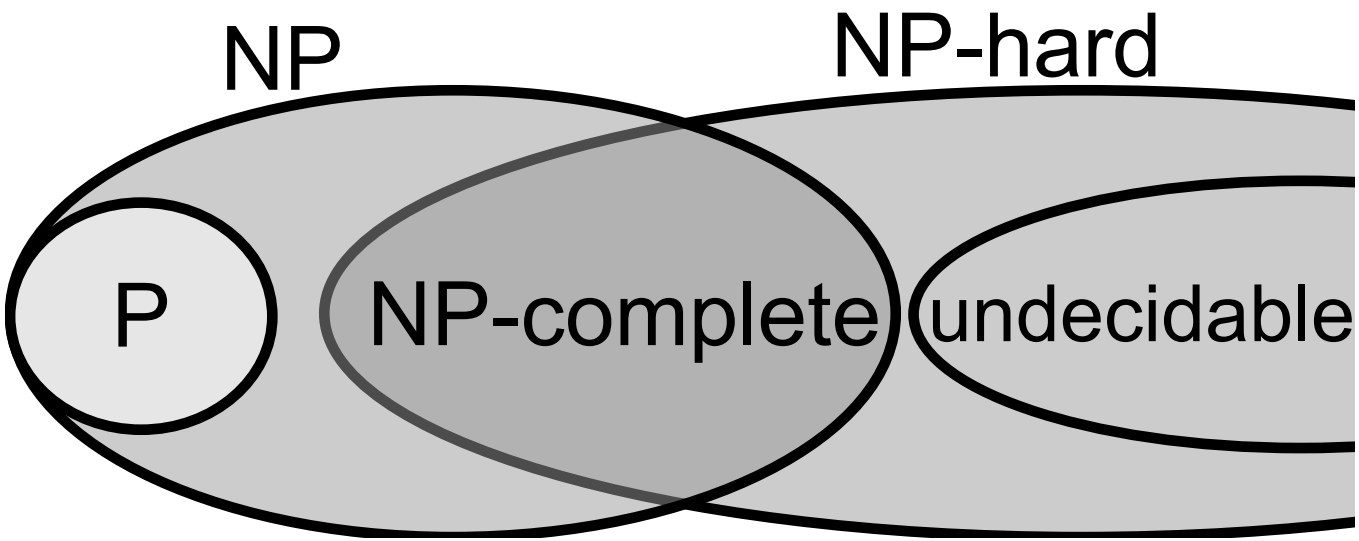
Hard and easy problems

Important classes of problems

P	polynomial-time hard
NP	nondeterministic polynomial-time hard problems, which can be solved in a polynomial time by a nondeterministic generalization of the Turing machine (with a deterministic Turing machine they might take a longer time to be solved)
NP-complete	problems are "the hardest problems in NP" (the problems to which any other problem in NP can be reduced)
NP-hard	problems are "at least as hard as the NP-complete problems"
undecidable	For an undecidable problem, there is no algorithm that always gives a correct answer in a finite time.

Hard and easy problems

For example, the Travelling Salesman Problem (find the shortest path through all nodes of a fully-connected weighted graph) is NP-complete with respect to the number of graph vertices.



The question whether P=NP is an important open problem.

The heuristic approach

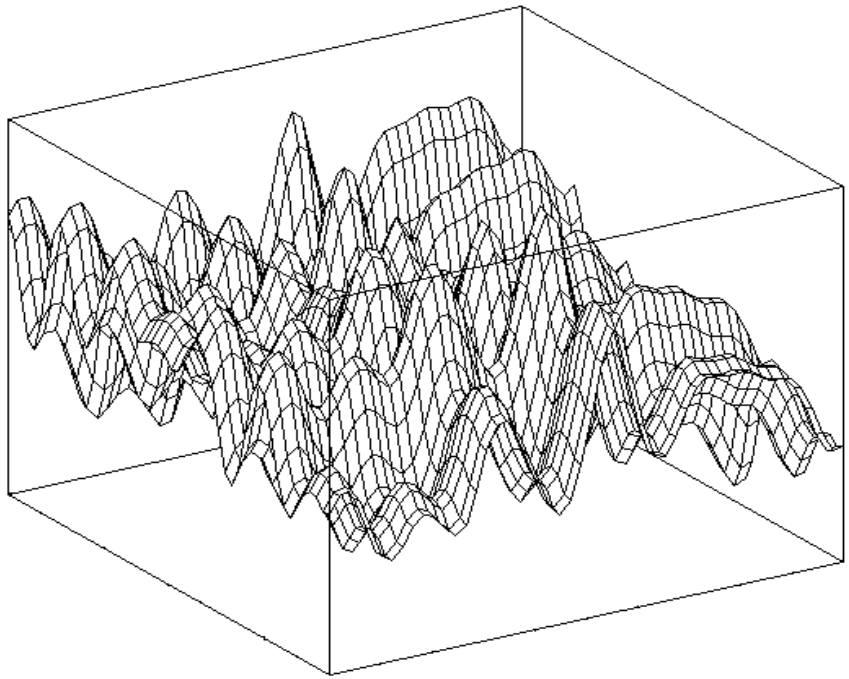
If an optimization problem is too hard to be solved

- in reasonable time
- using classical methods

one can use

the heuristic approach

sacrifice the exact optimality to reduce the run time.



A heuristic optimization algorithm usually:

- runs reasonably fast, but there is no guarantee it will be always the case, and
- finds quite good solutions, but there is no proof they could not get arbitrarily bad.

No Free Lunch Theorem (NFLT)

There's no such thing as a free lunch...

A seemingly pessimistic theorem on optimization of *all* problems:

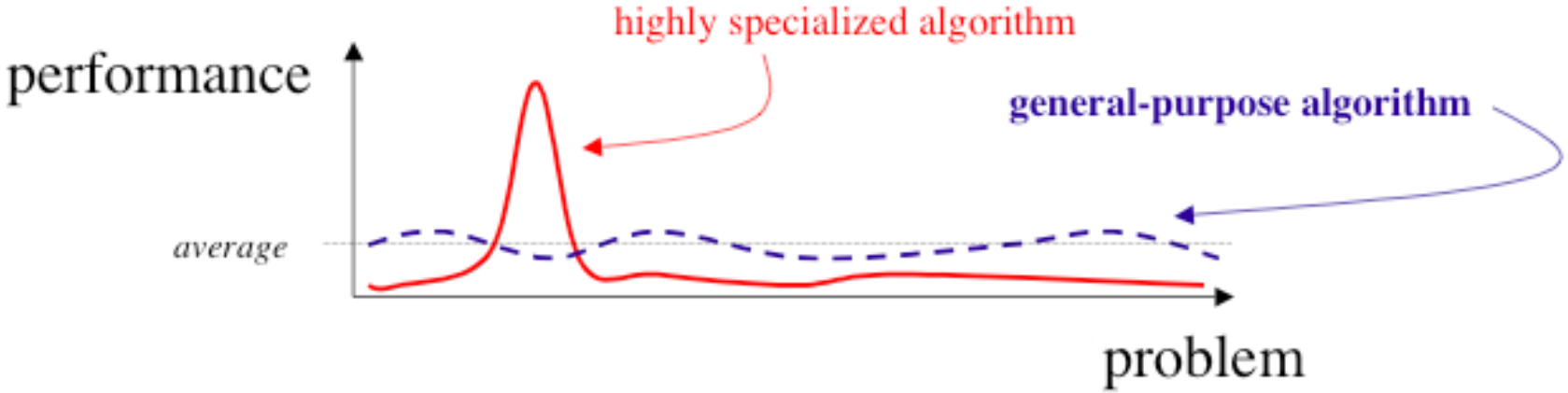
> *All algorithms that search for an extremum of a cost function perform exactly the same, when averaged over all possible cost functions.*
>
> Wolpert and Macready (1995)

> *For any algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class.*
>
> Wolpert and Macready (1997)

No Free Lunch Theorem (NFLT)

There's no such thing as a free lunch...

an illustrative, but not entirely accurate, explanation



The NFLT holds if *all* objective functions are considered, while most of the objective functions met in practice exhibit a degree of regularity (e.g. continuity, smoothness, convexity, etc.). Thus, a prior knowledge of the problem can be used to chose a specialized algorithm, which outperforms an average algorithm.

A general remark

Purely heuristic methods (like evolutionary algorithms) are more or less general all-purpose approaches, hence for specific classes of problems they are almost always slower than classical specialized methods. Thus, it pays off to look beyond the buzzwords like *evolutionary*, *intelligent*, *swarm*, etc.

Use a purely heuristic method for hard problems (discrete NP-complete, non-continuous, no derivative, many local minima), that is when you cannot exploit any specific characteristics of the problem.

Outline

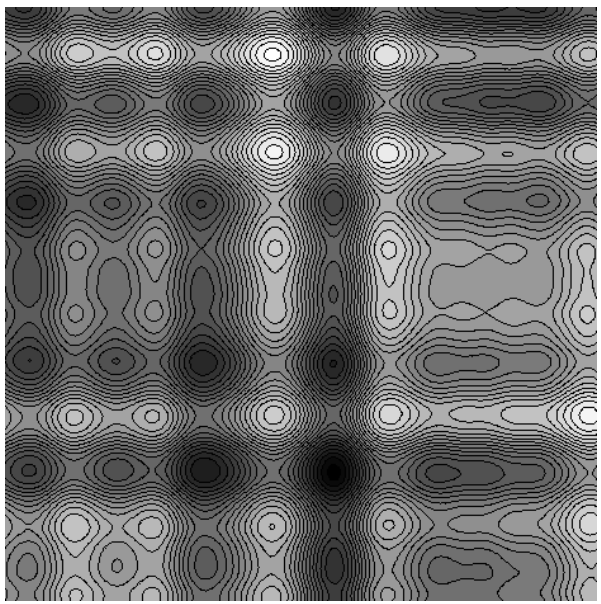
Basic ideas

Classical optimization methods are in fact local search methods:

- Converge only if the objective function is regular enough.
- Strongly depend on the starting point: quickly find a (nearest) local minimum, but there is no guarantee that it is global.
- Find the global optimum only for a very restricted class of objective functions (convex).

General heuristic ideas

- Randomize the starting point(s).
- Try several starting points.
- Randomize the search method.
- Couple the information between different local searches.



Types of algorithms

Several types of algorithms can be used for global optimization:

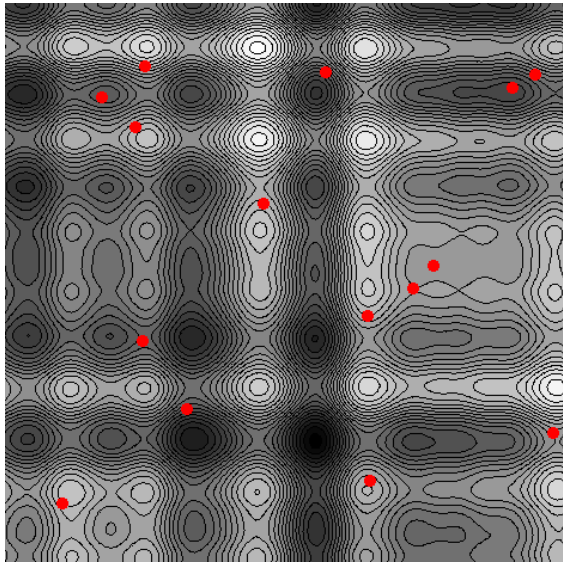
- simple randomization techniques:
  - random search,
  - random-restart hill climbing;
- more complex but still rather classical:
  - Coupled local minimizers,
  - Nelder-Mead (simplex) method;
- heuristics imitating different physical or natural phenomena:
  - Simulated annealing,
  - Evolutionary (genetic) algorithms,
  - Swarm intelligence.

Simple randomization techniques

Random search

- Try randomly different points in the domain.
- Take the best.

is not a very effective technique.



Much better performs its simple improvement:

Random-restart hill climbing

- Choose randomly several points in the domain.
- Perform independent local (classical) searches starting in each of the points.
- Take the best result.

Coupled local minimizers (CLM)

Suykens (2001)

The random restart hill climbing method relies on:

- Randomization of several starting points.
- Independent, fast-convergent local searches.

The local searches can be coupled to "encourage" them to reach the same final position (assumed to be the global minimum).
A composed objective function is used, for example[2]

$$f_{\text{CLM}}(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \frac{1}{n} \sum_{i=1}^{n} f(\mathbf{x}_i) + \alpha \sum_{i=1}^{n-1} \|\mathbf{x}_{i+1} - \mathbf{x}_i\|^2,$$

where α is a positive coefficient:

- a small α results in a wide, near-independent exploration of the search domain (random-restart hill climbing).
- a large α forces the search points to first approach each other and then to search together for the nearest local minimum.

[2] In the original method an augmented Lagrangian is used.

Coupled local minimizers (CLM)

Suykens (2001)

Advantages

- Parallel strategy.
- Information exchange.
- Fast converging gradient-based classical methods.

Disadvantages

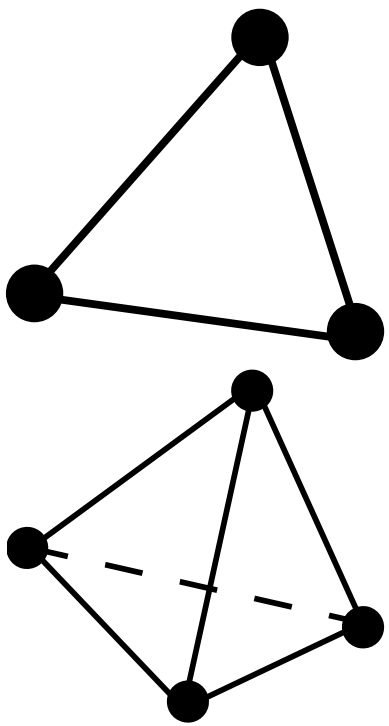
- Increased dimensionality of the objective function.
- Multiple computations of the original objective function (as in all multi-point methods).

Nelder-Mead (simplex) method

The method starts with a polytope, which is called the simplex (2D: triangle, 3D:tetrahedron...).

The search proceeds through recursive updates of the locations of the simplex vertices. In each step, depending on the values of the objective function in the vertices, the simplex is updated through a series of four basic operations:

1. reflection,
2. expansion,
3. contraction,
4. shrinkage.



Nelder-Mead (simplex) method

0. Evaluate the objective function f at each of the vertices $\mathbf{x}_0, \ldots, \mathbf{x}_n$. Let:

   $\mathbf{x}_l$ be the vertex with the minimum value of f,
   $\mathbf{x}_h$ be the vertex with the maximum value of f.

   Try to update "the worst" vertex $\mathbf{x}_h$ by:

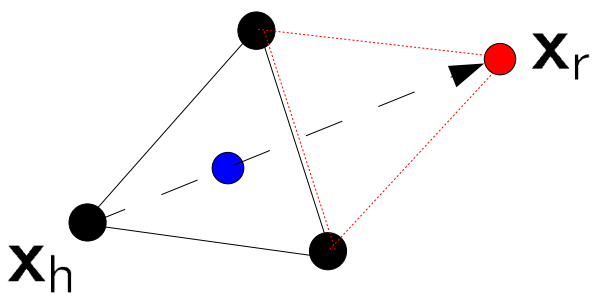
1. *Reflection*,

$$\mathbf{x}_r = \bar{\mathbf{x}} + \alpha(\bar{\mathbf{x}} - \mathbf{x}_h),$$

where $\alpha > 0$ is called the *reflection ratio* and $\bar{\mathbf{x}}$ is the centroid of the vertices,

$$\bar{\mathbf{x}} = \frac{1}{n+1} \sum_{i=0}^{n} \mathbf{x}_i.$$

If $f(\mathbf{x}_l) < f(\mathbf{x}_r) < f(\mathbf{x}_h)$, then $\mathbf{x}_h \leftarrow \mathbf{x}_r$ and return to step (0).

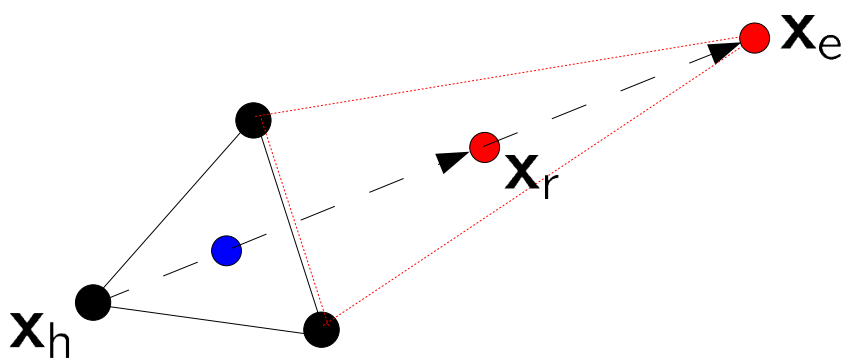


Nelder-Mead (simplex) method

2. If $f(\mathbf{x}_r) < f(\mathbf{x}_l)$, the point $\mathbf{x}_r$ can be tried to be further *expanded* to $\mathbf{x}_e$ by

$$\mathbf{x}_e = \bar{\mathbf{x}} + \beta(\mathbf{x}_r - \bar{\mathbf{x}}),$$

where $\beta > 1$ is called the *expansion ratio*. If $f(\mathbf{x}_e) < f(\mathbf{x}_r)$, then $\mathbf{x}_h \leftarrow \mathbf{x}_e$, otherwise $\mathbf{x}_h \leftarrow \mathbf{x}_r$. Return to (0).

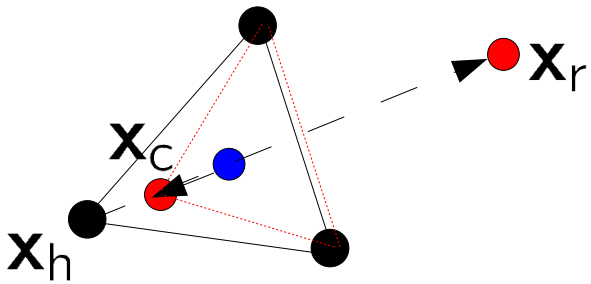


Nelder-Mead (simplex) method

3. If $f(\mathbf{x}_h) < f(\mathbf{x}_r)$, perform the *contraction*:

$$\mathbf{x}_c = \bar{\mathbf{x}} + \gamma(\mathbf{x}_h - \bar{\mathbf{x}}),$$

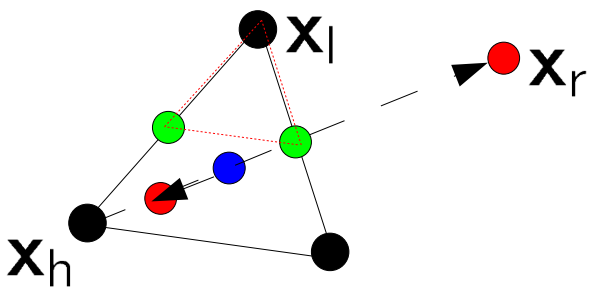
where $0 < \gamma < 1$ is called the *contraction ratio*.



4. If still $f(\mathbf{x}_h) < f(\mathbf{x}_c)$, perform the *shrinkage*:

$$\mathbf{x}_i = \mathbf{x}_i + \frac{1}{2}(\mathbf{x}_l - \mathbf{x}_i)$$

and return to (0).



Nelder-Mead (simplex) method

Have a look at examples on
Zafer Gurdal's homepage, Animation Gallery at
`http://www2.esm.vt.edu/~zgurdal/COURSES/4084/4084-Docs/Animation.html`.

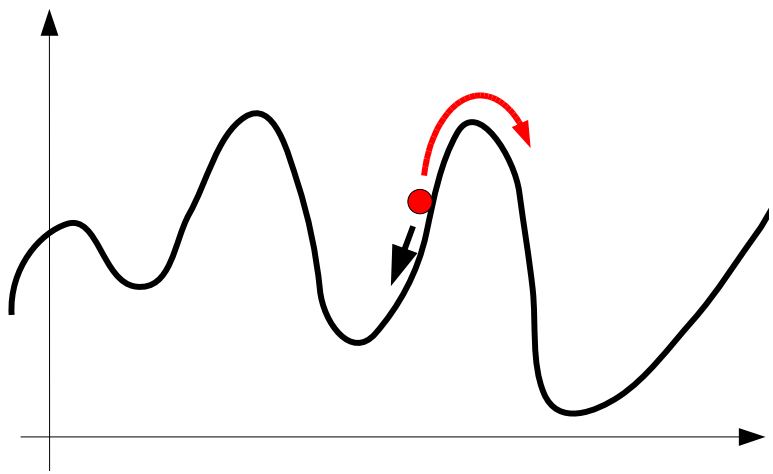
Simulated annealing

Simulated annealing is based on Metropolis' version of the Monte Carlo (probabilistic) system analysis technique.

The principle comes from metallurgy: a piece of metal is heated and then slowly cooled down, which allows the atoms to wander randomly through states of higher energy toward their most stable, minimal energy, positions.

All previously considered minimization methods always try to go downhill:

- it is reasonable, but it can lead to a local minimum only, hence
- sometimes the point should go uphill to get to the next, possibly deeper valley.



Simulated annealing

Simulated annealing can use both one and many starting points:

- the starting points are chosen randomly
- each of them explores the domain wandering in small steps independently and randomly through its neighborhood (no information coupling and no derivative information).
- The "willingness" of a point to go uphill in each step is controlled by a global parameter T, which is called the *system temperature* and gradually decreased during the process.

Possible stop conditions:

1. an optimal enough point is found
2. the system has been cooled down.

Simulated annealing — basic iteration step

Basic iteration step:

1. Repeat for each search point:
   1. Chose randomly a state $\mathbf{x}_1$ from the neighborhood of the current state $\mathbf{x}_0$
      - The notion of neighborhood is application-dependent,
      - e.g. add a vector of random numbers or
      - in the TSP invert two neighboring towns, etc.
   2. Decide randomly whether the new state should be accepted. The transition probability $P(\mathbf{x}_0, \mathbf{x}_1, T)$ is taken from the Maxwell-Boltzmann distribution,

$$P(\mathbf{x}_0, \mathbf{x}_1, T) = \begin{cases} 1 & \text{if } f(\mathbf{x}_1) < f(\mathbf{x}_0) \\ \exp \frac{f(\mathbf{x}_0) - f(\mathbf{x}_1)}{T} & \text{otherwise} \end{cases}$$

2. Repeat (1) until, in general, an equilibrium is attained, that is until the average of $f(\mathbf{x}_0)$ remains stable over several steps.
3. Decrease the system temperature T.

Simulated annealing — transition probability

According to the transition probability

$$P(\mathbf{x}_0, \mathbf{x}_1, T) = \begin{cases} 1 & \text{if } f(\mathbf{x}_1) < f(\mathbf{x}_0) \\ \exp \frac{f(\mathbf{x}_0) - f(\mathbf{x}_1)}{T} & \text{otherwise} \end{cases}$$

- If the new point is better than the old point (a downhill step), then it is always accepted,
- otherwise (an uphill step) it is accepted with a given probability only:
  - high system temperature T — high probability (close to 1),
  - low system temperature T — low probability (close to 0).

High temperature results in a random walk.

Low temperature results in downhill steps only.

Simulated annealing — annealing schedules

Several annealing schedules are used to decrease gradually the system temperature T to zero:

- logarithmic (Boltzmann)

$$T(t) \propto \frac{1}{\log t}$$

- linear (Cauchy)

$$T(t) \propto \frac{1}{t}$$

- exponential

$$T(t) \propto a^{-t}$$

Simulated annealing — parameters

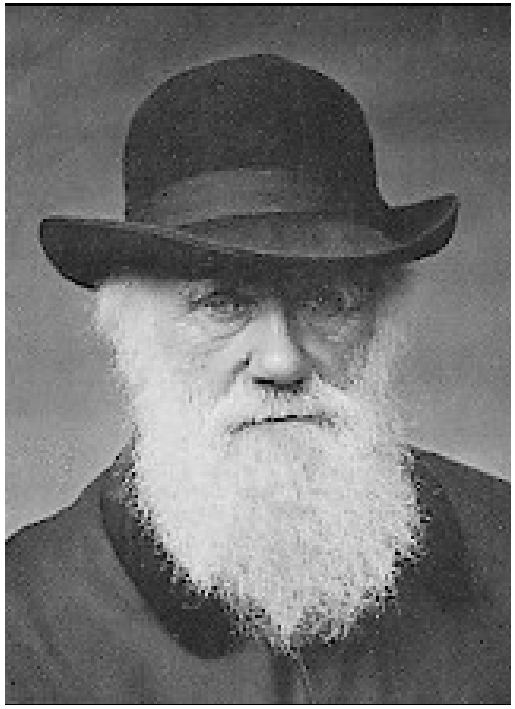
Three parameters govern the optimization process:

1. Neighborhood selection method.
2. Transition probabilities (Maxwell-Boltzmann distribution).
3. Annealing schedule.

Evolutionary algorithms

Genetic algorithms attempt to imitate the mechanisms of biological evolution and survival of the fittest:

- Mutation,
- Cross-over (recombination) and reproduction,
- Competition and environmental selection.



Characteristics:

- probabilistic search (common with SA)
- no derivative information (common with SA)
- parallel search with strong information coupling

Evolutionary algorithms — terminology

Biology	Evolutionary algorithms
individual (phenotype)	search point, candidate solution
population	set of candidate solutions
chromosome	encoded search point
adjustment to environment	objective function (fitness function)
generations	iteration steps
mutation	modification of a candidate solution
cross-over, reproduction	combining two candidate solutions

Evolutionary algorithms — an outline

1. Choose the scheme to encode the individuals (search points).
2. Generate randomly an initial population.
3. Proceed iteratively through successive generations. In every generation
   1. Apply randomly the operation of mutation.
   2. Select randomly pairs of search points (perhaps based on the fit function) and produce offsprings.
   3. Calculate the values of the fit function for every of the mutations and offsprings.
   4. Perform a randomized selection (the better fit is the individual, the more survival chances it has).

Evolutionary algorithms — the encoding scheme

The encoding scheme is a *method of representing a solution in a manner that can be manipulated by the algorithm*

- Traditionally binary sequences of constant lengths are used
  - Real numbers: fixed or floating point representations
  - Integers: binary representations
  - Chars: ASCII numbers
  - Sets: membership vectors, etc.
- But any natural representation is basically possible, provided the mutation and cross-over operations are defined (vectors of real numbers, strings, etc.).

consider an objective function of three real variables $f(\mathbf{x}, \mathbf{y}, \mathbf{z})$

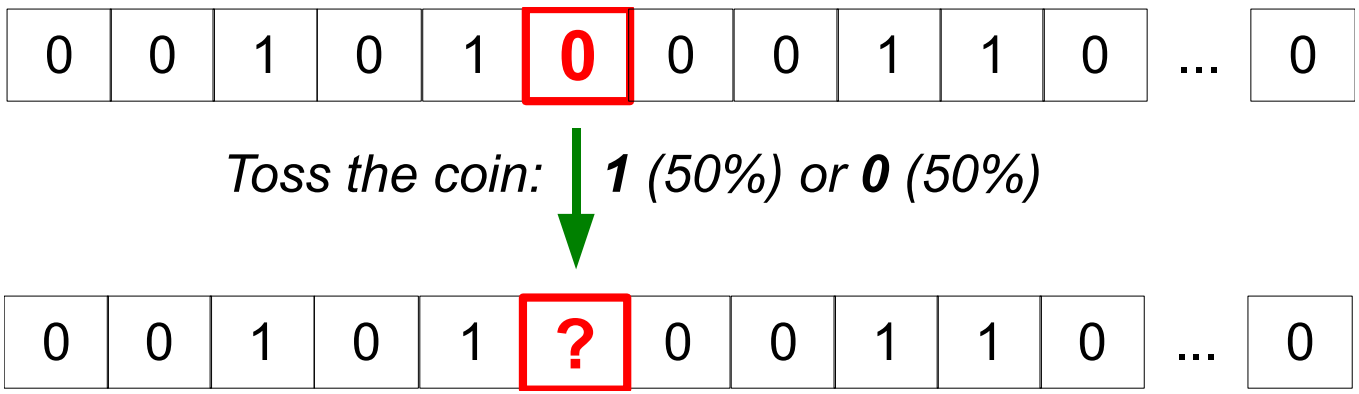
Binary vector 96 bits = 3×32 bit floats:
00100010111010100010111000110100110 1...010010

Real vector (x,y,z) $[17.45, 9.97, -4.28]$

Evolutionary algorithms — mutation

Binary representation

- Should affect in average 0.1% randomly chosen bits of the whole population.
- Modify each randomly chosen bit with 50% probability.



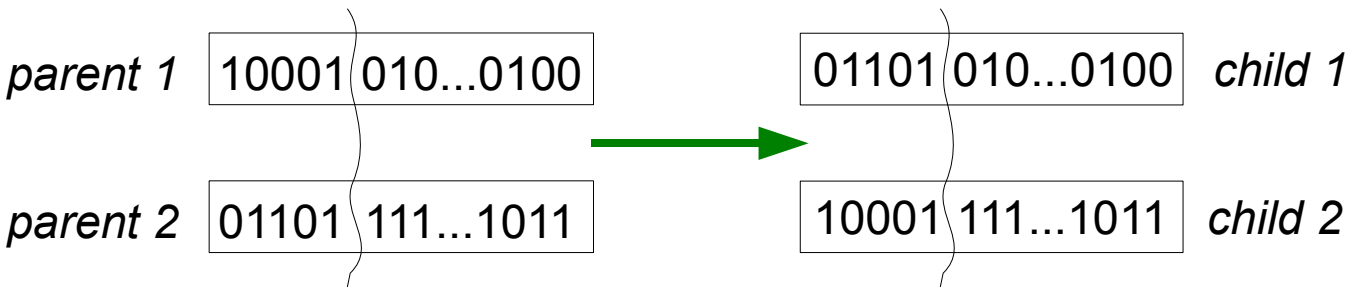
Natural representation (vector of real numbers)

- Randomly choose individuals to mutate
- With some probability modify the chosen individual by adding a vector of random numbers.

Evolutionary algorithms — cross-over

Randomly choose pairs of individuals and for each pair produce offsprings:

- For binary representation exchange random fragments of chromosomes



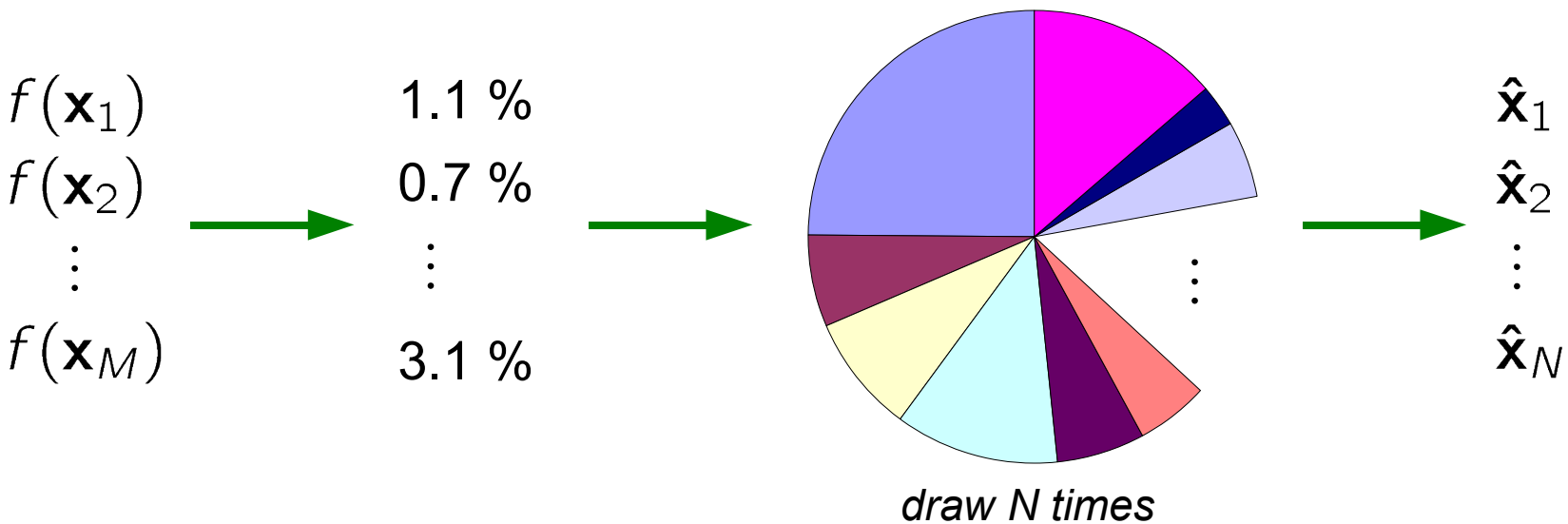
- For a natural representation, for example vectors of real numbers, draw the children from

$$N\left(\frac{\mathbf{x}+\mathbf{y}}{2}, s\frac{\|\mathbf{x}-\mathbf{y}\|}{2}\right).$$

Evolutionary algorithms — selection

The selection can be performed using a roulette wheel procedure

1. Compute the objective function for the new individuals (mutated and offsprings).
2. Distribute the probability of survival among all the individuals (fitter take bigger chances).
3. Draw independently individuals to form the population for the next generation (repetitions are possible).



Swarm intelligence techniques

A swarm is a collection of simple agents

- exploring their environment,
- interacting with one another and with the environment,
- distributed (no centralized control).

Swarm intelligence:

- comes not from the individual intelligence of simple agents
- but from their local interactions, which often lead to emergence of global behavior patterns.

Swarm intelligence techniques:

1. Particle swarm optimization (PSO),
2. Ant colony optimization (ACO).

Particle swarm optimization (PSO)

J. Kennedy and R. C. Eberhart (1995)

In a swarm of insects, if one insect follows a right path to go (e.g. for food), the other will follow it quickly.

The insects are modelled by search points (called particles), which

- have position $\mathbf{x}_i$ and velocity $\mathbf{v}_i$,
- fly through the search space,
- remember the best points
  - individual $\mathbf{x}_i^{\mathrm{I}}$,
  - global (swarm) $\mathbf{x}^{\mathrm{G}}$,
  - local (in the neighborhood) $\mathbf{x}_i^{\mathrm{N}}$.

Particle swarm optimization (PSO)

J. Kennedy and R. C. Eberhart (1995)

In each iteration step the positions and the velocities of all particles are updated:

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i,$$
$$\mathbf{v}_i \leftarrow \omega\mathbf{v}_i + \alpha\mathbf{r}_1(\mathbf{x}_i^{\mathsf{I}} - \mathbf{x}_i) + \beta\mathbf{r}_2(\mathbf{x}^{\mathsf{G}} - \mathbf{x}_i) + \gamma\mathbf{r}_3(\mathbf{x}_i^{\mathsf{N}} - \mathbf{x}_i),$$

where

- ω is an inertia constant (responsible for the *exploration* of the search domain),
- α, β, γ are positive constants (responsible for the *exploitation* of already gathered knowledge),
- r_1, r_2, r_3 are random numbers from $[0, 1]$.

Ant colony optimization (ACO)

M. Dorigo (1992)

Stigmergy (Grassé, 1959)

Indirect communication among social insects via interaction with the environment

Ants

- constantly leave pheromones
- wander randomly, but are more likely to follow the trails marked with pheromones.

Pheromones

- accumulate with the pheromones already laid by other ants on the trail
- evaporate with time.

Ant colony optimization (ACO)

M. Dorigo (1992)

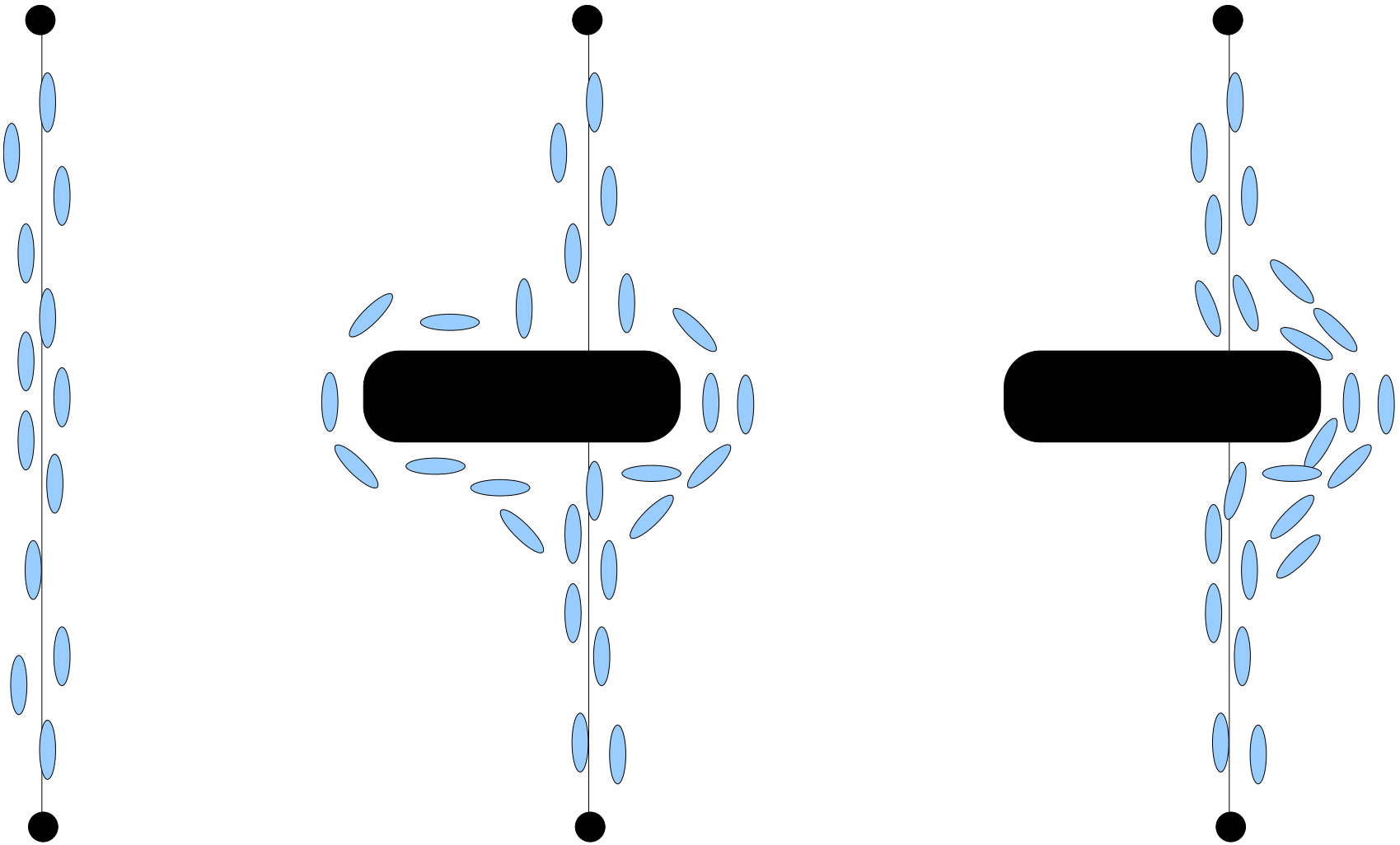
As a result, short paths may emerge.

The ant colony optimization

- is used to solve discrete optimization problems. Dorigo's original problem was finding the shortest path through a graph (the travelling salesman problem).
- thanks to the evaporation of the pheromones, is able to adapt to a dynamically changing environment.

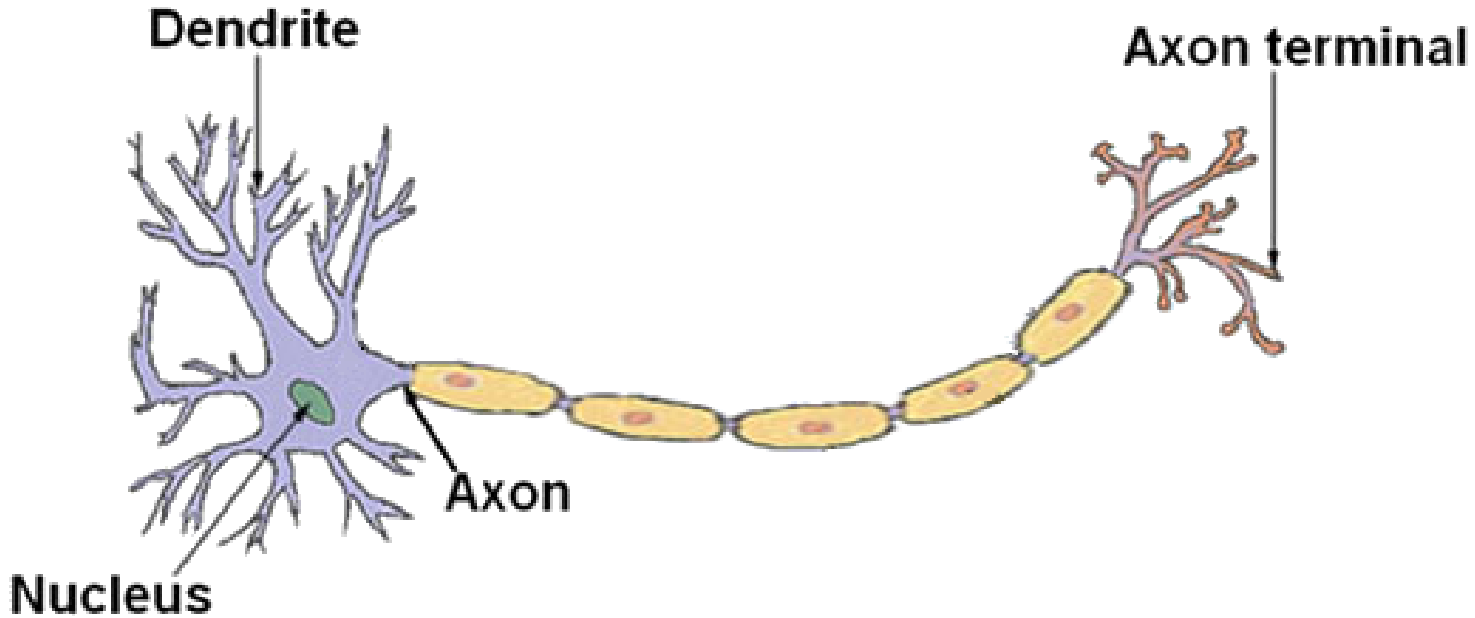
Ant colony optimization (ACO)

M. Dorigo (1992)



Outline

Natural neuron



Dendrites conduct signals into the neuron

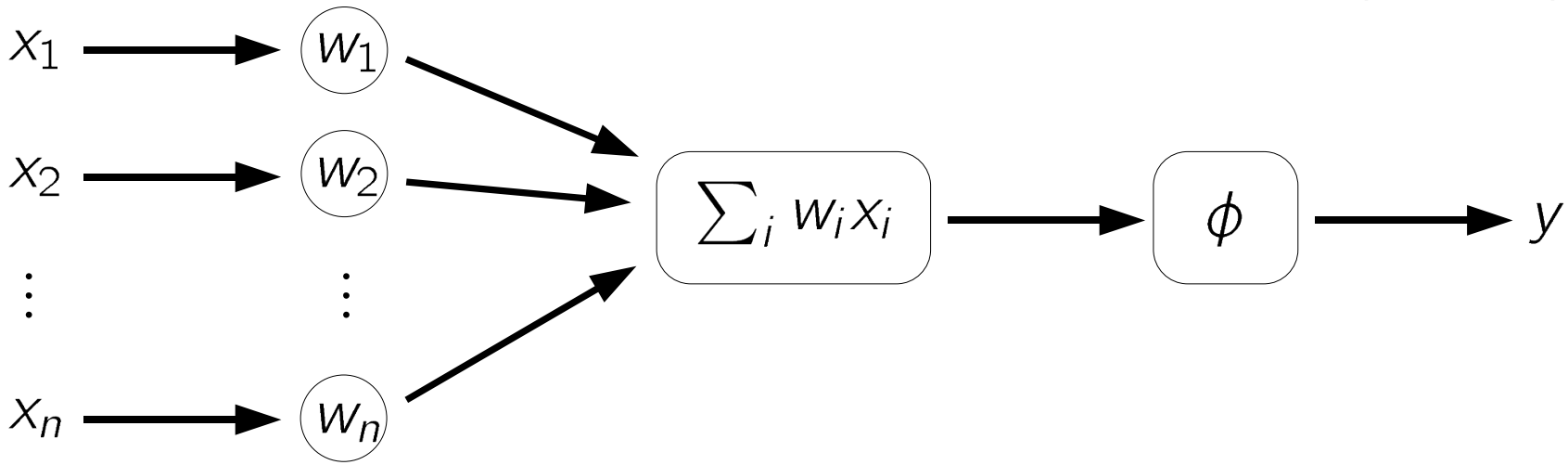
Axon extends 10...1000 times the diameter of the cell body, carries signals away to other neurons

Synapses are communication connections between axon terminals and dendrites

Human brain approximately 10^{11} neurons $\times$ 7 000 synapses

Artificial neuron...

...is a function $y = \phi(\sum_i w_i x_i)$



Inputs $x_1, \ldots, x_n$

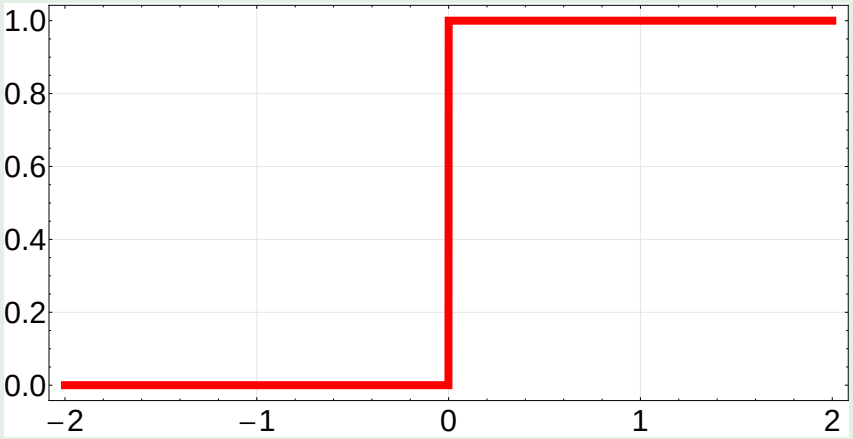
Weights $w_1, \ldots, w_n$

Transfer function ϕ

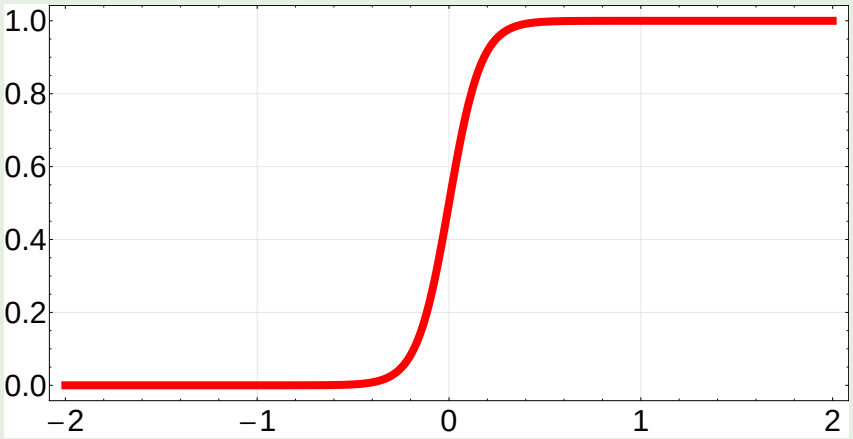
- standard $\phi(\sum_i w_i x_i)$ (step, sigmoid, linear)
- radial basis $\phi(\mathbf{x})$ (RBF)

Artificial neuron — transfer functions

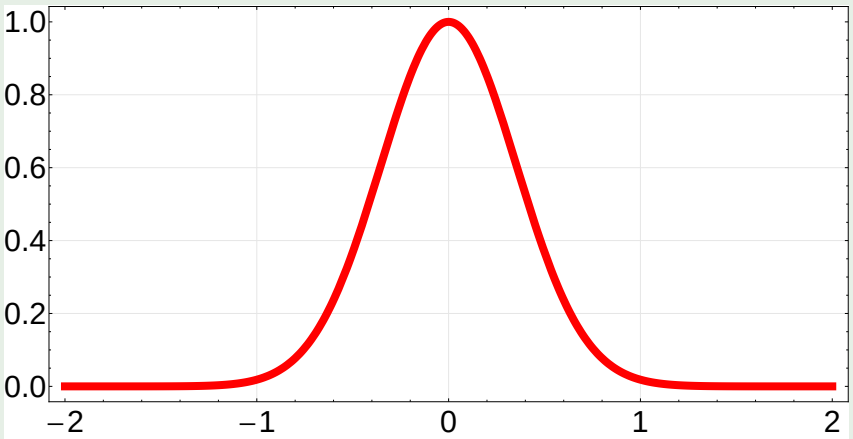
Step function



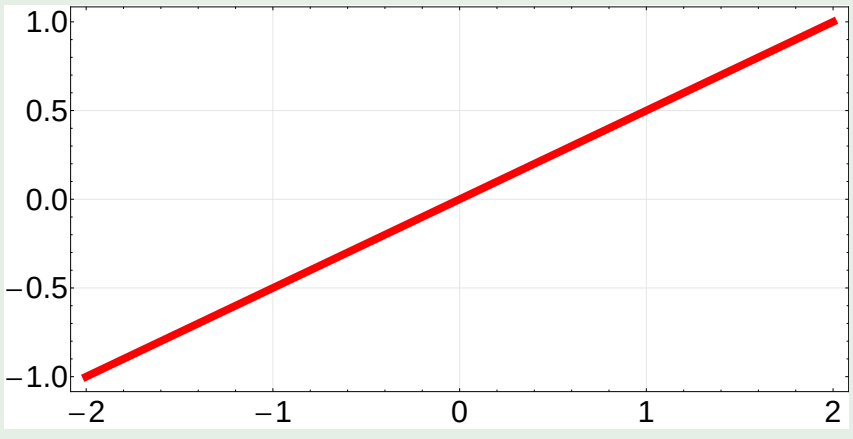
Sigmoid



RBF (Gaussian)



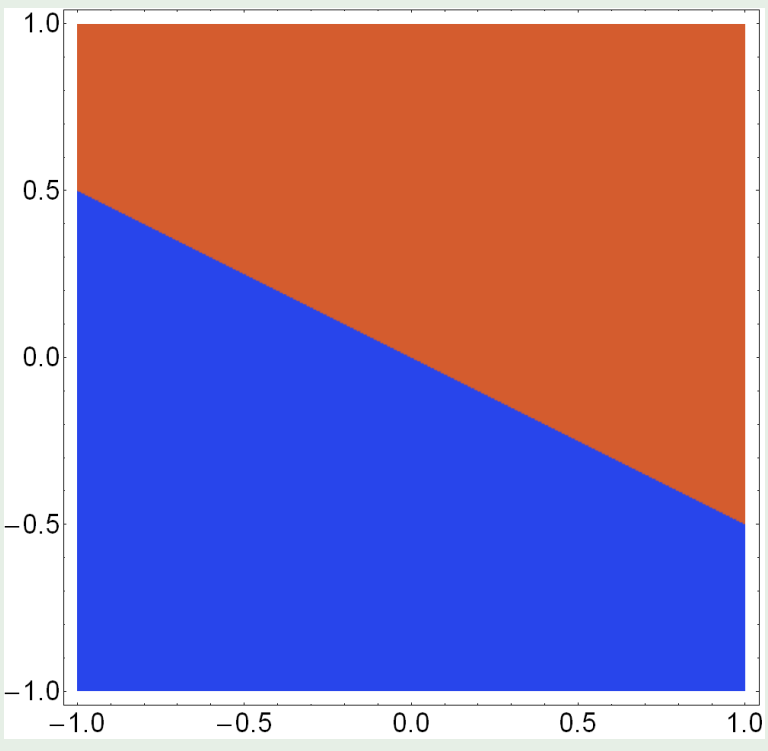
Linear



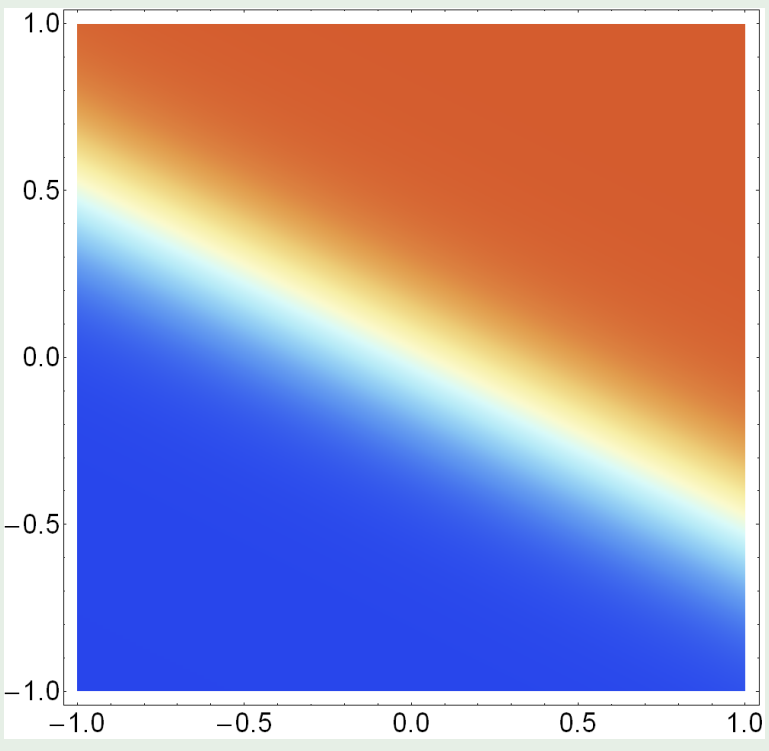
Artificial neuron — transfer functions

$$y(x_1, x_2) = \phi(x_1 + 2x_2)$$

Step function



Sigmoid

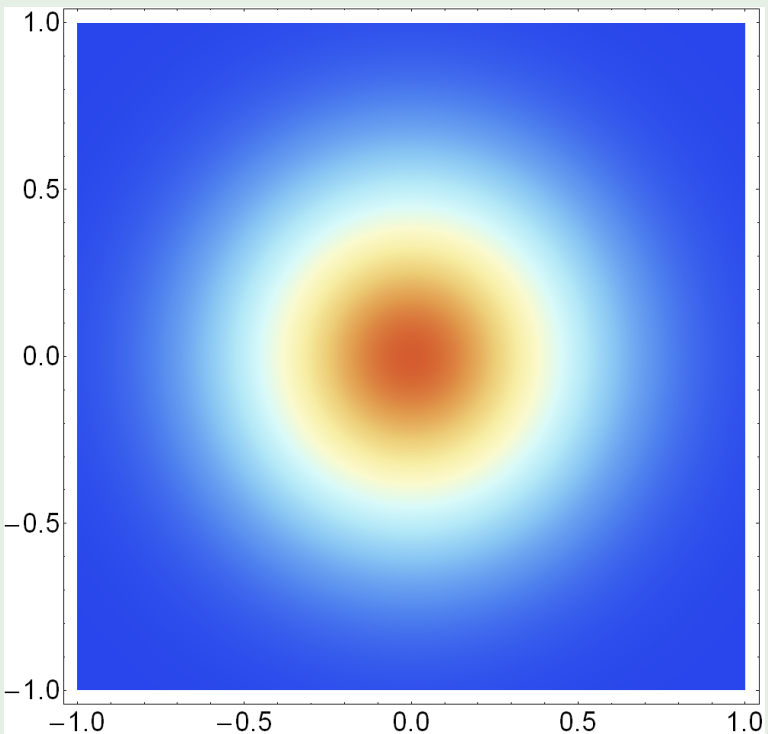


Artificial neuron — transfer functions

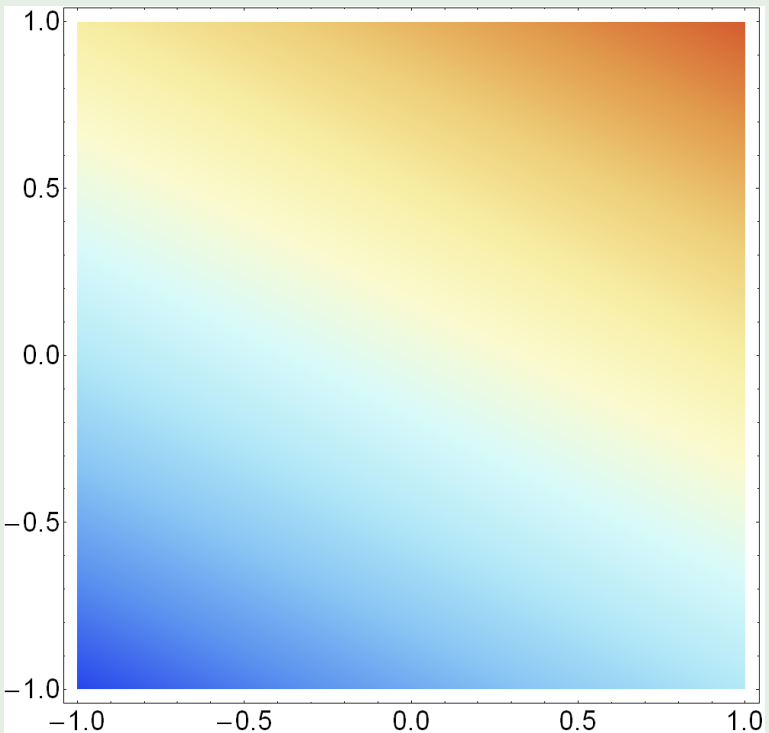
$$y(x_1, x_2) = \phi(x_1, x_2)$$

$$y(x_1, x_2) = \phi(x_1 + 2x_2)$$

RBF (Gaussian)



Linear



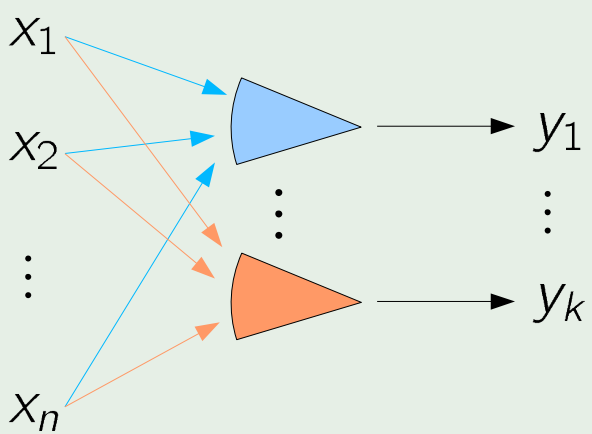
Feed-forward ANN

A single artificial neuron is just a function of its input variables.
More artificial neurons can be used

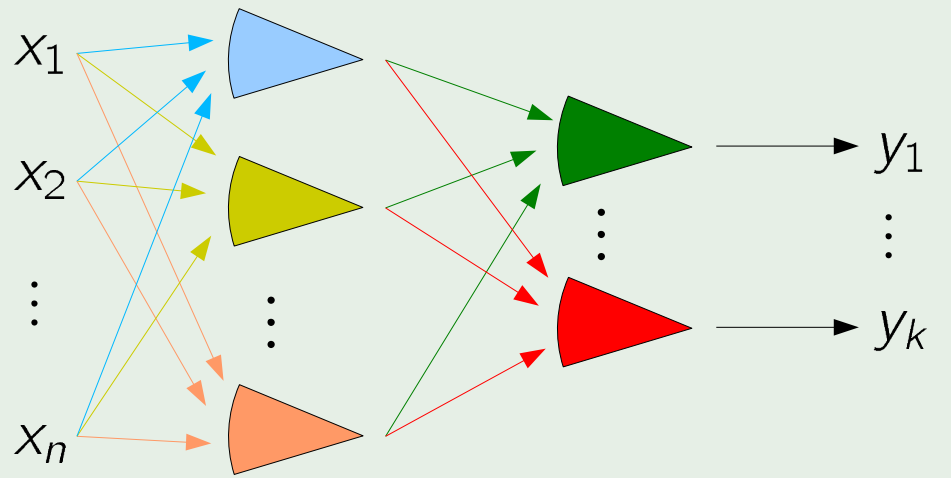
- in parallel and/or
- in series

to form an artificial neural network (ANN).

single layer ANN

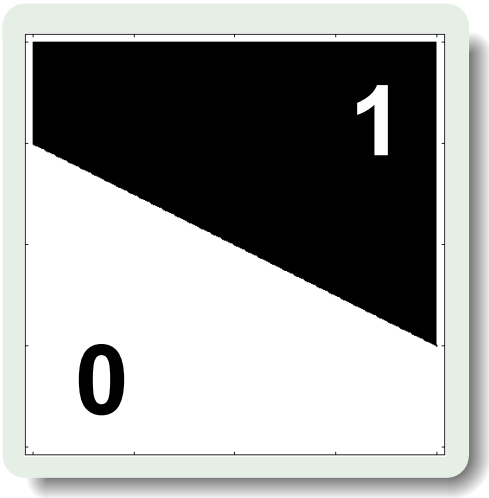


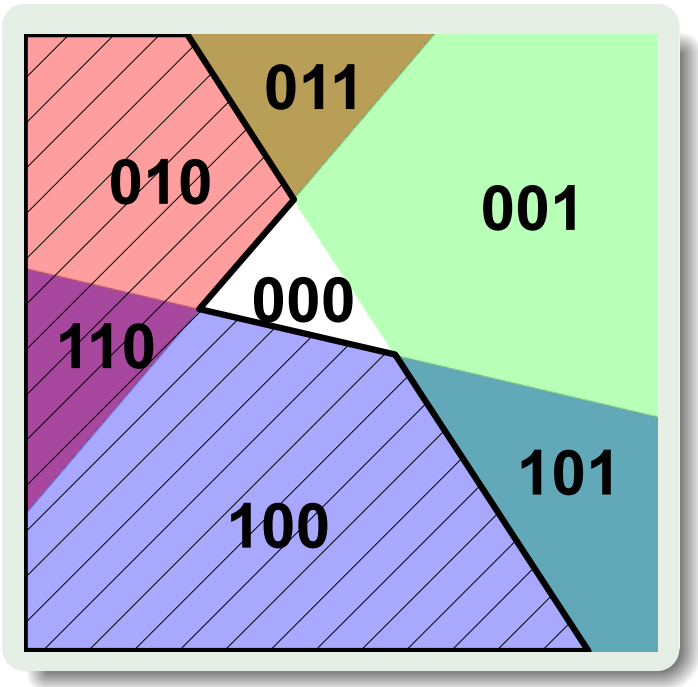
two layer ANN

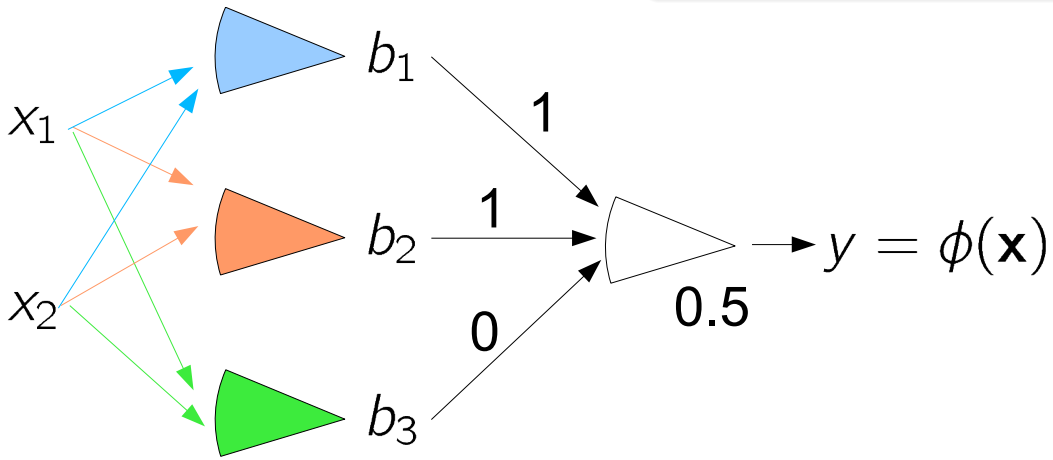


Feed-forward ANN

A single artificial neuron with a step transfer function differentiates between two sides of a hyperplane defined by the weight vector $w_1, \ldots, w_n$ and the threshold. If more artificial neurons are used, more line-based discriminations can be performed simultaneously.



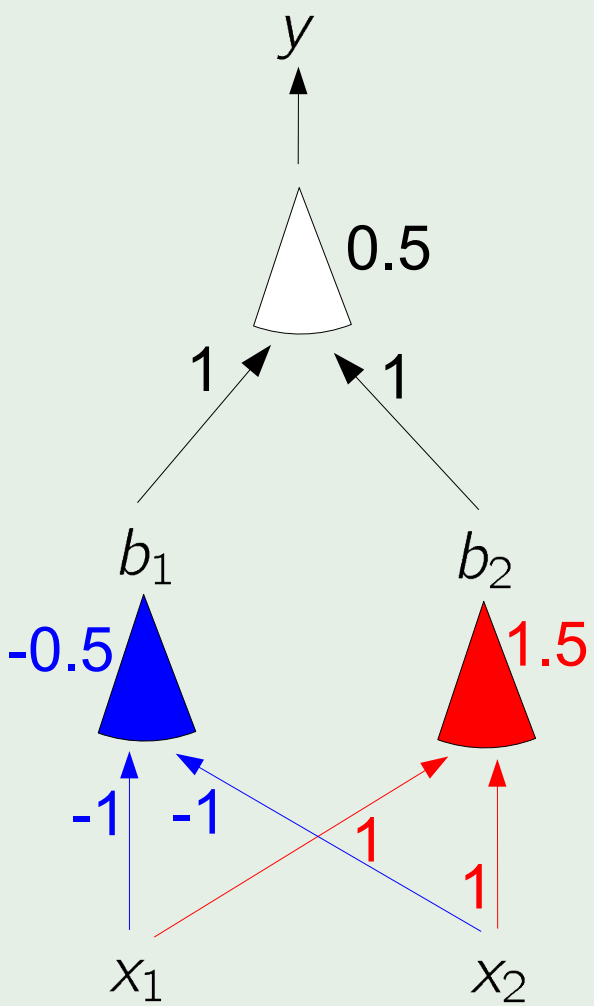


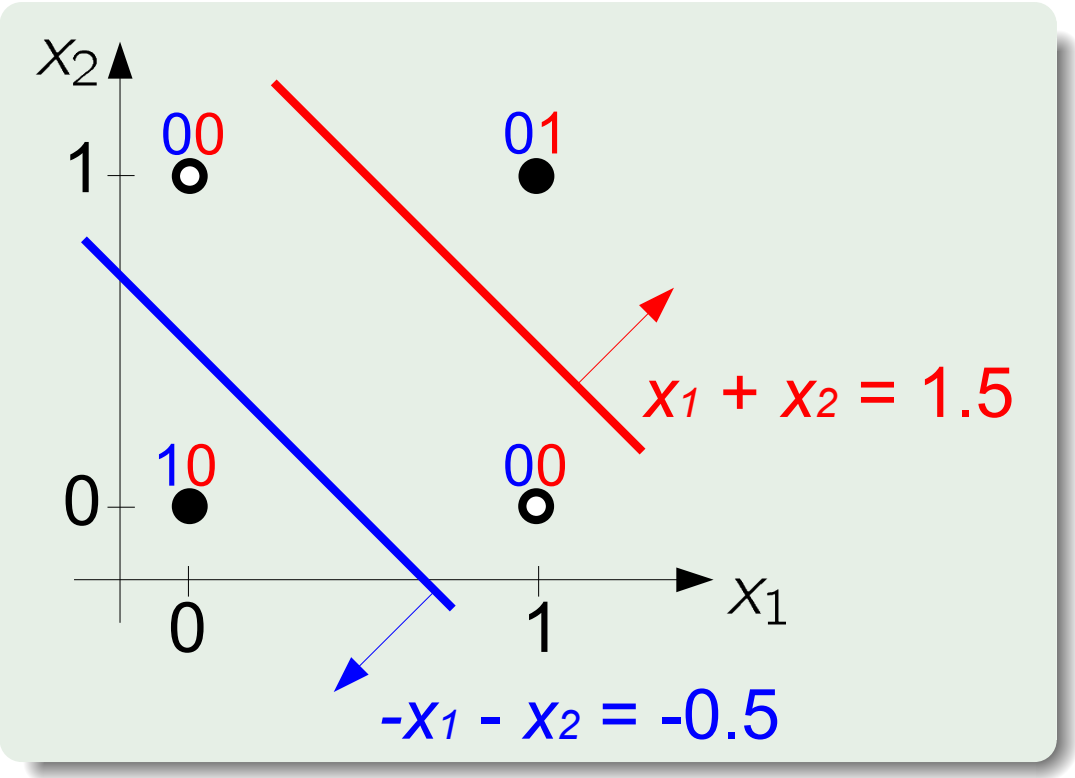


The weights are assigned to the RGB neurons according to the inclinations of the respective lines.

Feed-forward ANN

logical XOR





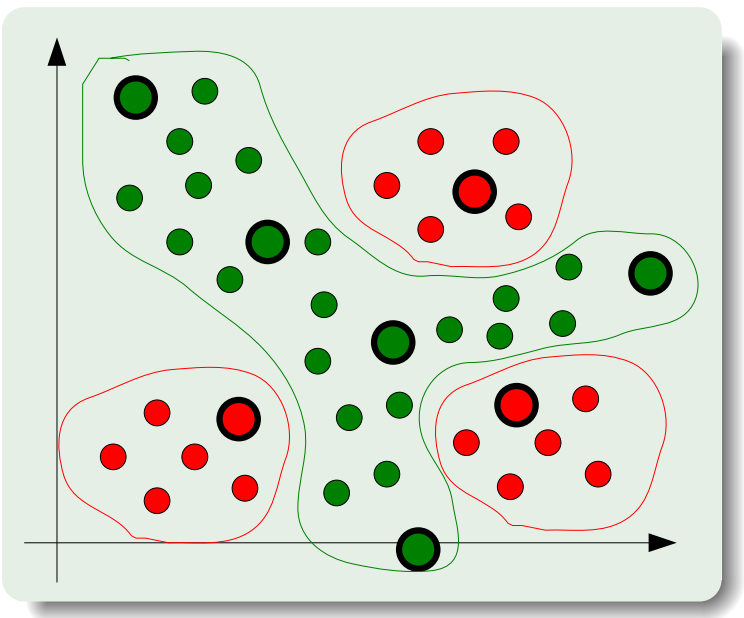
Hence, an ANN can be *predesigned* (by setting the weights and thresholds) to recognize given, known patterns. But it can be also *trained* to recognize and classify unknown patterns.

ANN learning

An ANN can be also trained using sample patterns to recognize and classify similar but previously unknown patterns.

Behavior of an ANN is fully determined by the weights and thresholds of its neurons, hence

ANN training = optimization of weights and thresholds of neurons



Input: *co-ordinates*; output: *color*

The colors of the points are unknown, but they are supposed to group around the known bigger dots. If an ANN is trained to recognize properly the bigger dots (the *training set*), it will probably recognize properly most of the dots.

ANN learning

Training set is a sequence of pairs of an input and the corresponding proper output $(\mathbf{x}_i, \mathbf{y}_i)$.

Objective function is a measure of difference between the computed and the proper outputs, for example

$$\sum_i \|\mathbf{y}_i - \phi(\mathbf{x}_i)\|^2.$$

Optimization Basically any algorithm can be used. The classical method is the *backpropagation* algorithm:

- simple steepest-descent approach (gradient-based, thus requires a differentiable, non-step neuron transfer function like sigmoid);
- backward propagation of error done layer-by-layer, starting with the output layer and proceeding backwards.

ANN learning — three paradigms

supervised learning the training set (the co-ordinates of the bigger dots and their real colors) is known in advance: *pattern recognition*, *approximation*

unsupervised learning no training set is provided, the objective function (the colors of the dots) has to be figured out based on the input data (the co-ordinates): *clustering*

reinforcement learning no training set and no data provided a priori, they are generated sequentially in interactions of an agent with its environment (data is the current state of the agent; computed output is the action taken; objective function is based on the effect of the action).

ANN learning — over-optimization

Take care not to over-optimize the network:

- An ANN with 5 layers of 10 artificial neurons has 550 unknowns (weights and thresholds) to optimize, and hence can perfectly interpolate a 1 year history of 2 stock indices (fewer than 550 values).
- But the network would be worth nothing when tested on next year's data.
- Divide the dataset into subsets: training, validation, test

Other network structures

There are many different ANN structures:

- Feed-forward ANN
- Kohonen self-organizing map (SOM)
- Recurrent networks (dynamic systems)
- Stochastic ANN
- etc.

An ANN can be also used as a subsystem of a larger hybrid system.

There is also a domain of deep neural networks & deep learning...