Evolutionary Algorithms

Machine Learning, Fall 2018

Jürgen Schmidhuber, Michael Wand, **Raoul Malm**

TAs: Robert Csordas, Aleksandar Stanic,
Xingdong Zuo, Francesco Faccio

slides by Raoul Malm

USI/SUPSI
Istituto Dalle Molle di studi sull'intelligenza artificiale
IDSIA

Introduction

So far, we had always one parameter set that was optimised (one “agent”)

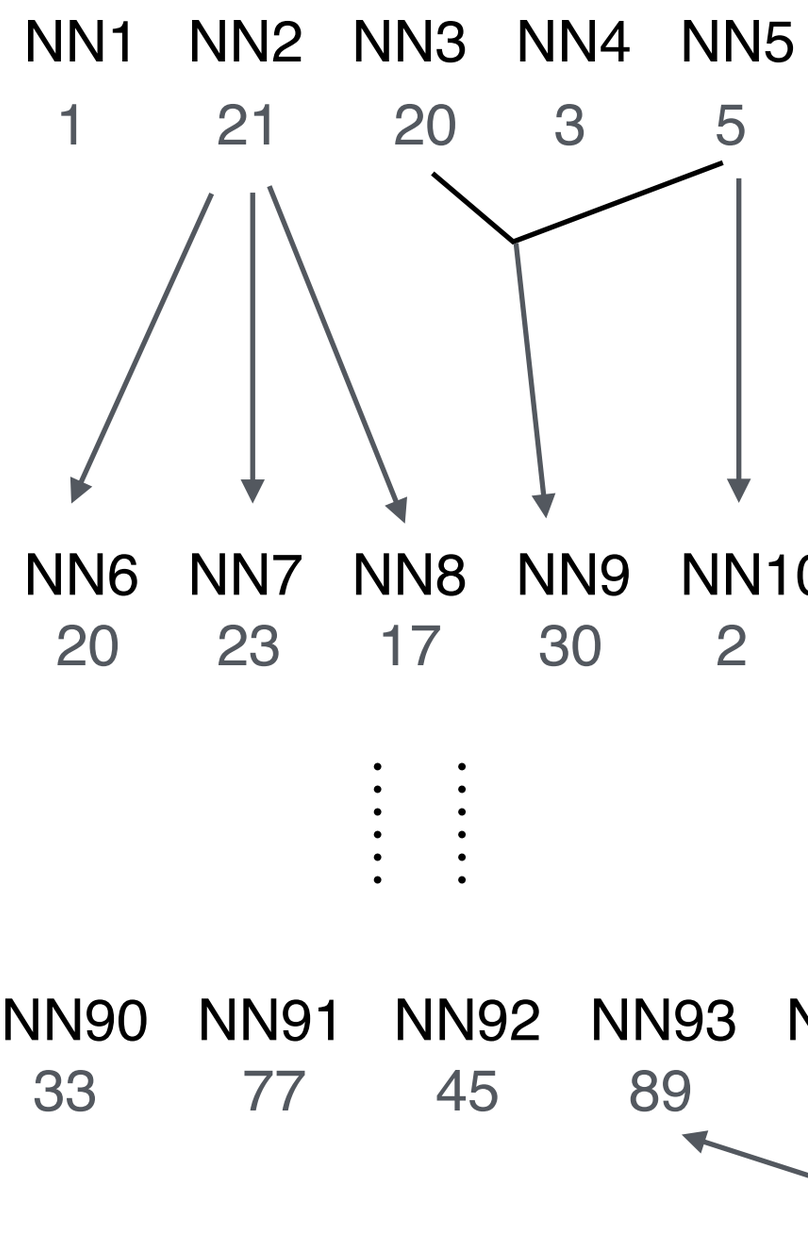
- in RL, one agent modifies its internal structure (policy) to maximise its reward when interacting with the environment
- in supervised/unsupervised learning (e.g. NN, SVM, HMM), there is one set of objective parameters to be optimised

The paradigm of **Evolutionary Algorithms (EA)** is different

- we have a population of many parameter sets (“agents” in RL terminology) that learn collectively
- the parameter sets improve by search techniques based on an abstraction of biological evolution (“natural selection”)
- usually derivative-free optimisation is used: only make use of knowing how to evaluate the objective function

Basic Example

Consider 5 neural networks with random weights, e.g. used for MNIST classification



population of 5 neural networks

evaluate “fitness” scores

select fittest parents and generate 5 offsprings by
- mutation, e.g. add random noise to weights
- crossover, e.g. recombine two parents

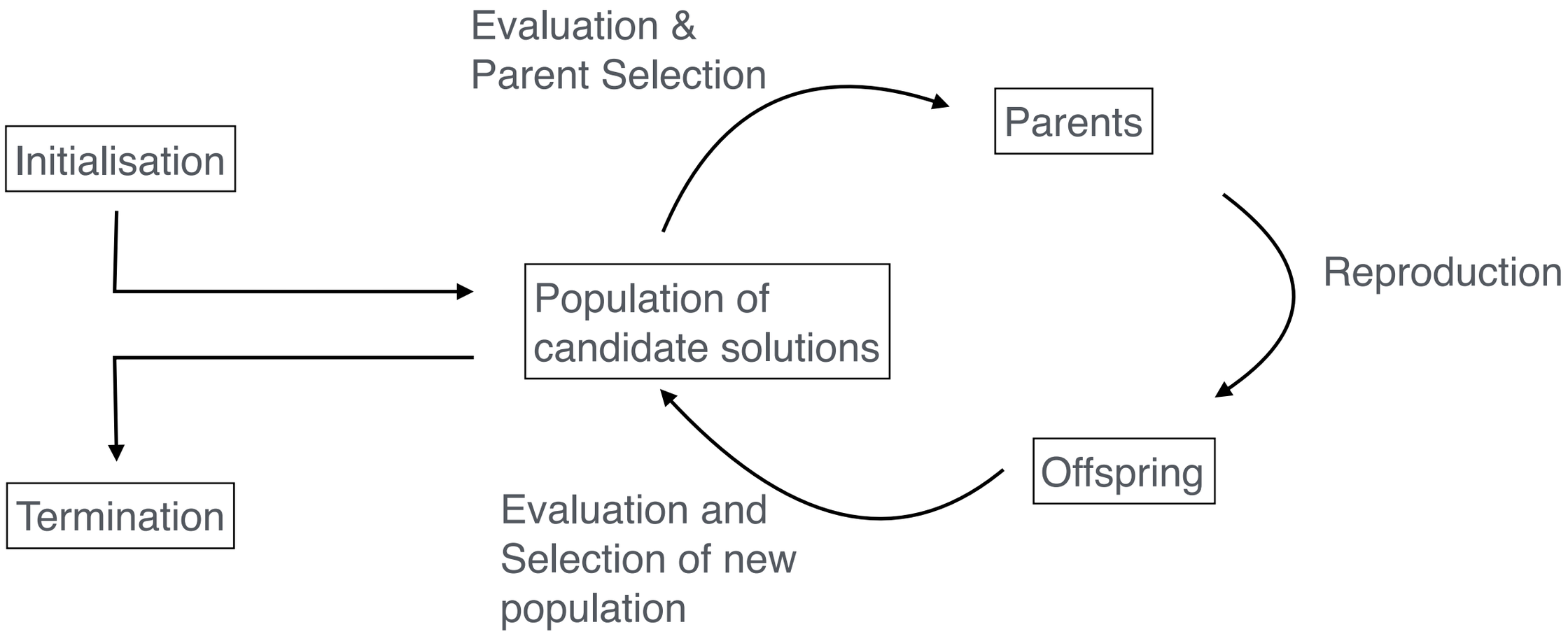
new population of 5 neural networks

evaluate fitness scores

repeat steps

return fittest candidate

Basic Idea



1. Initialise population with random candidate solutions
2. Evaluate each candidate (in parallel) and assign (scalar) “fitness” scores
3. Repeat, until a termination condition is fulfilled
    1. Select fittest candidates in the population for reproduction (parents)
    2. Reproduce new candidates (offspring) from parents
    3. Evaluate the fitness of the offspring and select candidates for the new population
4. Return the fittest candidate

Fitness Landscapes

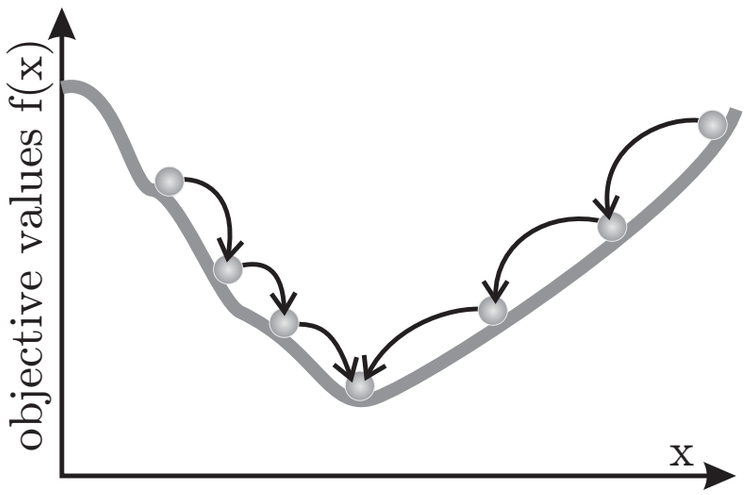


Fig. 1.19.a: Best Case

- optimisation algorithms are guided by objective functions
- A function is “difficult” from a mathematical perspective if it is not continuous, not differentiable, or if it has multiple maxima and minima

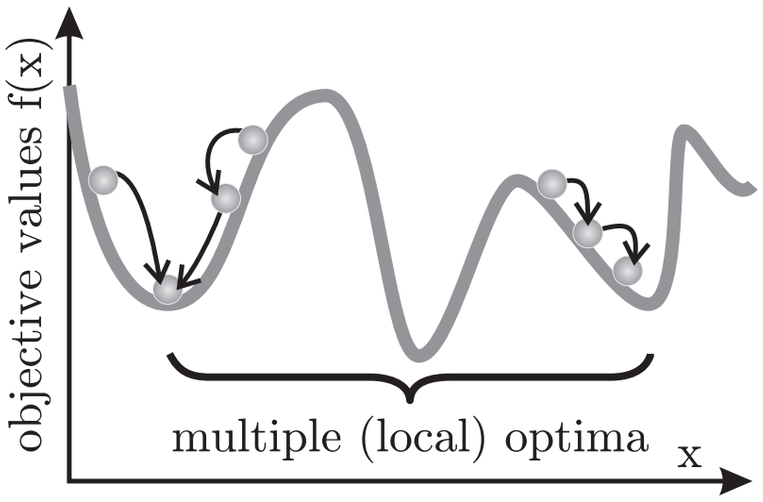


Fig. 1.19.c: Multimodal

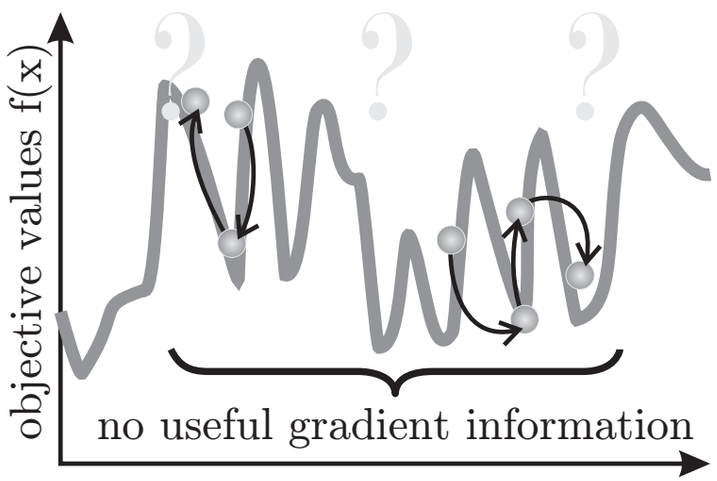


Fig. 1.19.d: Rugged

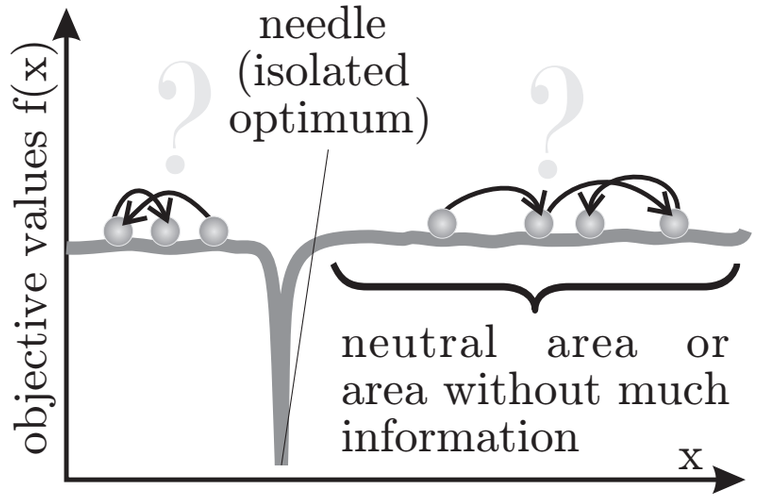
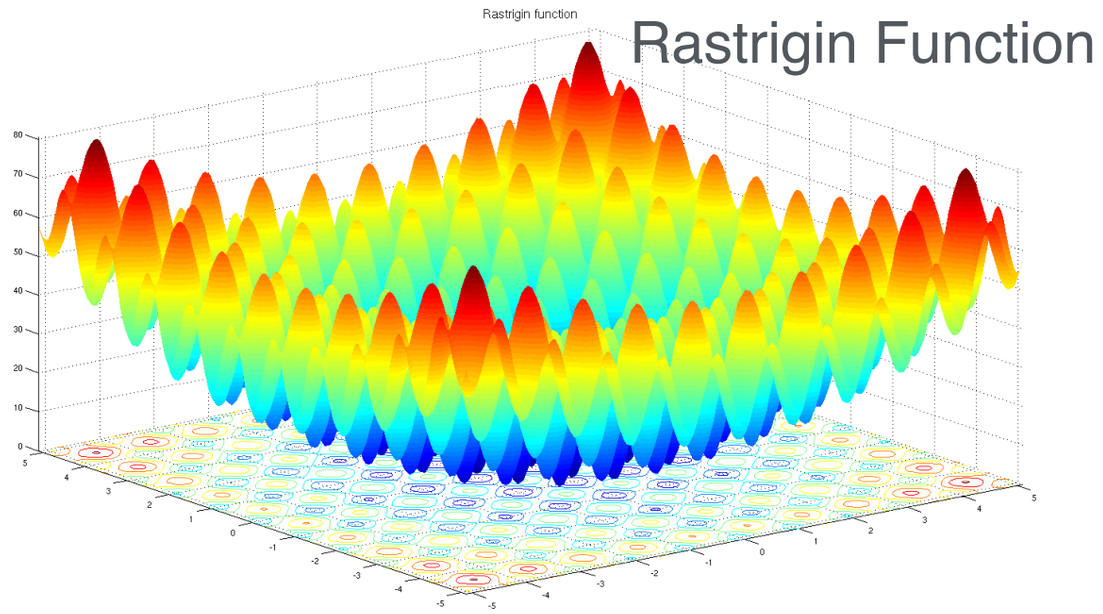


Fig. 1.19.g: Needle-In-A-Haystack

[Weise: Global Optimisation Algorithm - Theory and Application]

Why EAs ?

- **if the objective (fitness) function is multi-modal/noisy/discontinuous**
  - a population of “agents” increases the probability to find the global optimum
  - derivative-free optimisation methods are more robust to local optima (saddle points) than gradient-based methods

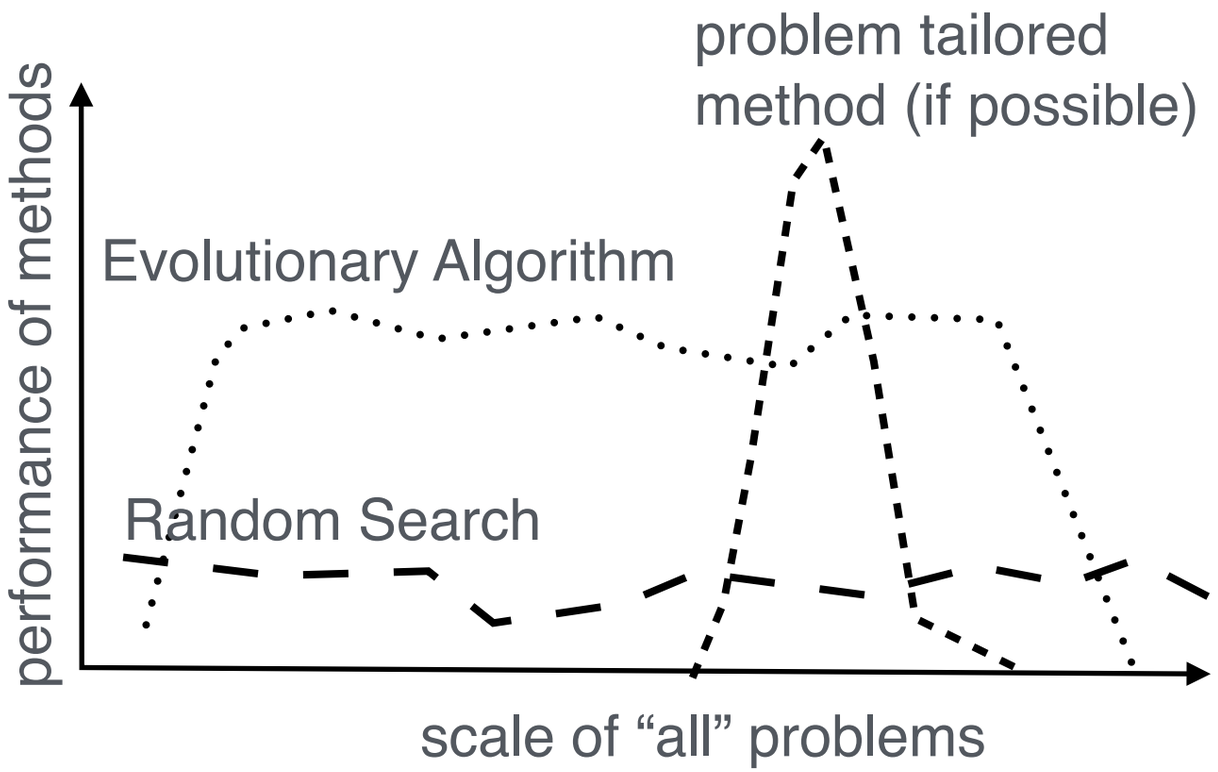


Rastrigin Function

- **if we know little about the objective function (little domain knowledge)**
  - in EA we only need to know how to evaluate the fitness function

- **straightforward parallelisation**

- **conceptual simplicity**



Branches of Evolutionary Algorithms

1. Genetic Algorithms (GA) (Holland 1975)
   - “binary” representation of candidate solutions
2. Evolution Strategies (ES) (Schwefel 1973)
   - “real-valued” representation of candidate solutions
3. Genetic Programming (GP) (Koza 1989)
   - “tree (data structure)” representation of candidates
4. Evolutionary Programming (EP) (Fogel 1966)
   - candidates represented by “finite state machines”

Focus of this lecture

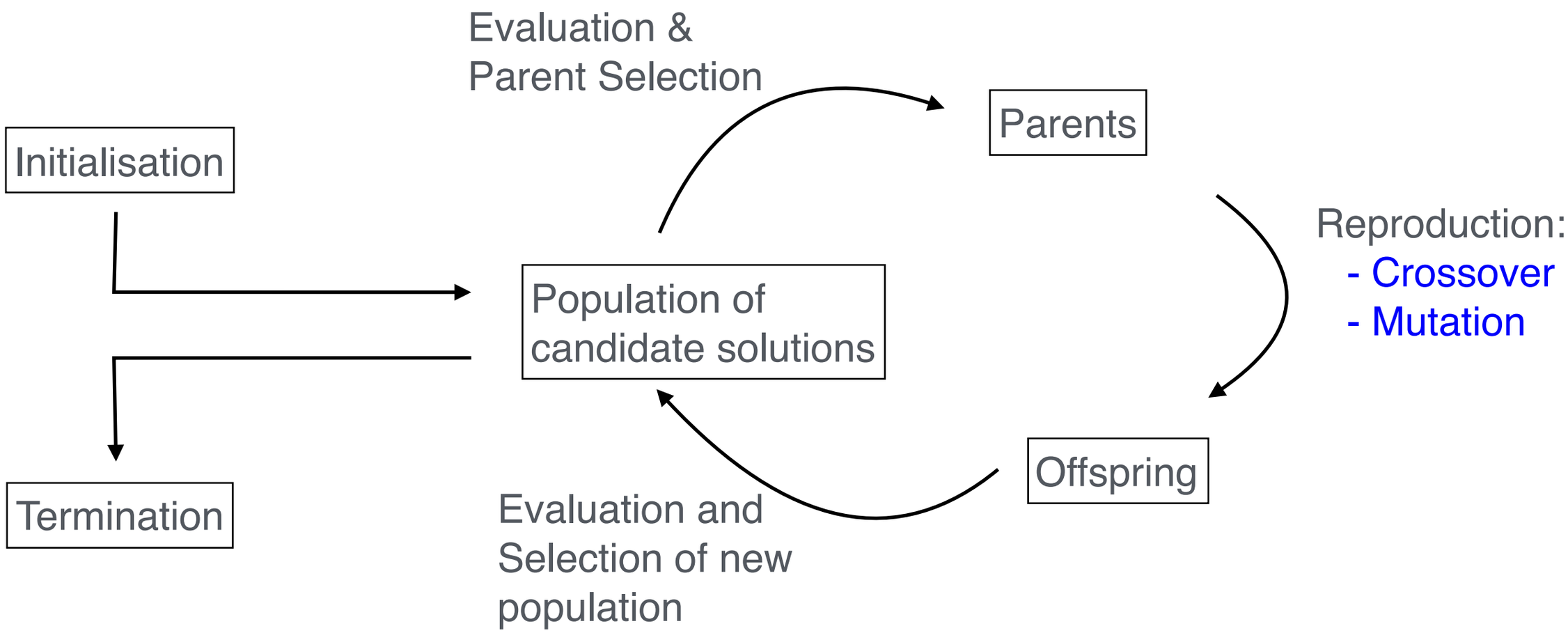
“recent” works:

[arXiv: 1712.06567] Uber AI Lab: apply GA on policies with ~1 million weights

[arXiv: 1703.03864] Open AI Lab: apply ES on policies with ~1 million weights

Genetic Algorithms (GA)

Genetic Algorithms (GA)

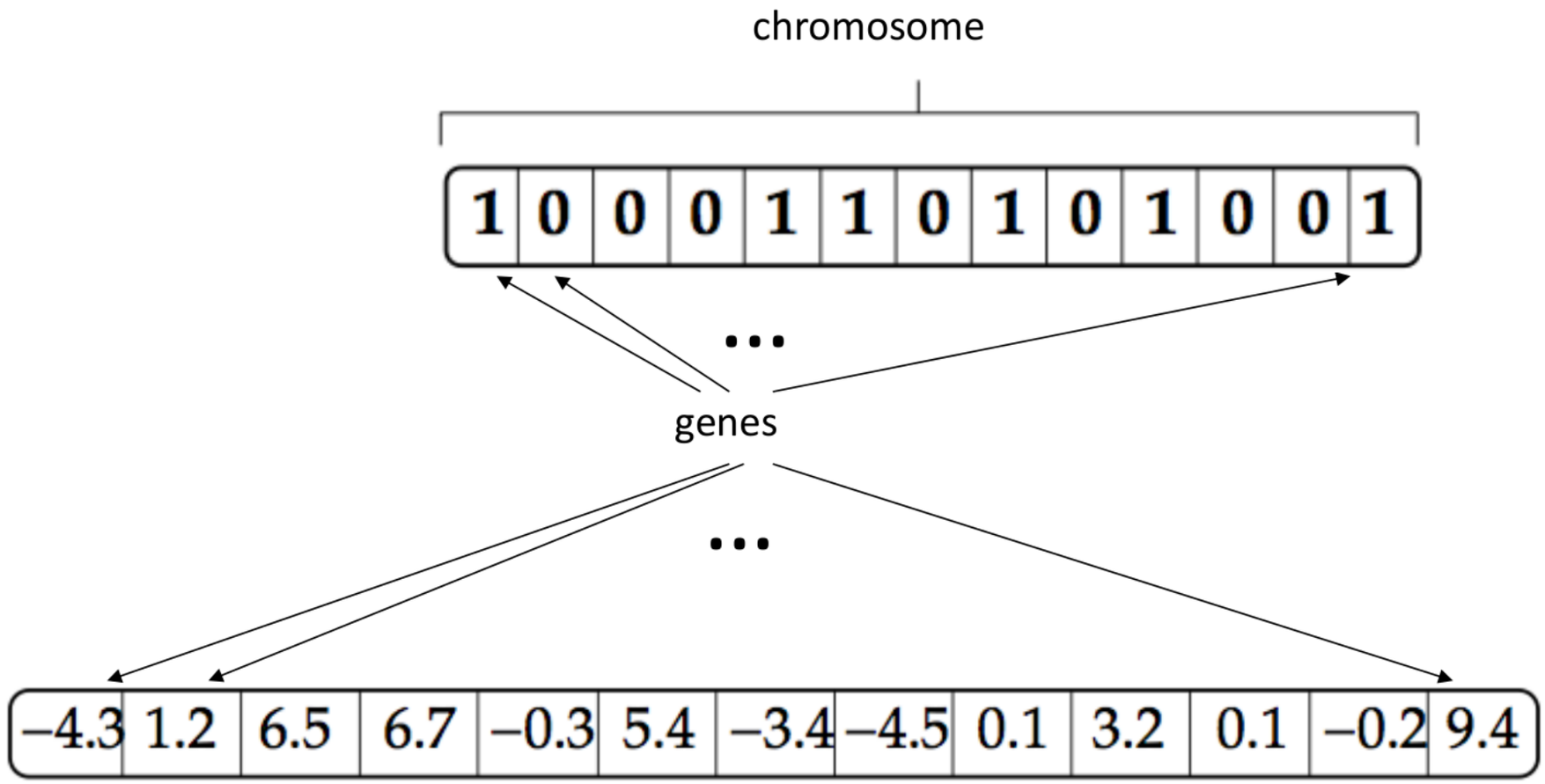


1. Initialise population with random candidate solutions
2. Evaluate each candidate (in parallel) and assign (scalar) “fitness” scores
3. Repeat, until a termination condition is fulfilled
    1. Select fittest candidates in the population for reproduction (parents)
    2. Reproduce new candidates (offspring) from parents: Crossover, Mutation
    3. Evaluate the fitness of the offspring and select candidates for the new population
4. Return the fittest candidate

Relationship with Biology

Representation

- a genome contains the complete genetic information of an individual
- a genome is a collection of chromosomes
- a chromosome is a string of genes (of fixed or variable length)
- a gene can take on a value, also called allele, from some specified alphabet
  - binary alphabet: 0,1
  - (infinite) alphabet of real numbers
- binary example: Lactase, the gene that splits milk sugar into simpler sugars for digestion is a gene. Lactose tolerance and intolerance are alleles of that gene.



Relationship with Biology

- the **genotype** is the set of genes in our genome which is responsible for a particular trait
- the **phenotype** is the physical expression, or characteristics, of that trait
- in general a genotype is a lower-level representation of the phenotype
- **encoding**: the mapping from phenotype to genotype
- **decoding**: the mapping from genotype to phenotype
- the **genotype-phenotype mapping** depends on (unknown) environmental factors and therefore does not represent a one-to-one correspondence (i.e. not bijective)
  - “oversimplified” example: two individuals with the same genotype for ‘blue eye colour’ can have actually two different developed phenotypes, one individual might develop a “blue eye colour” and the other a “green eye colour”

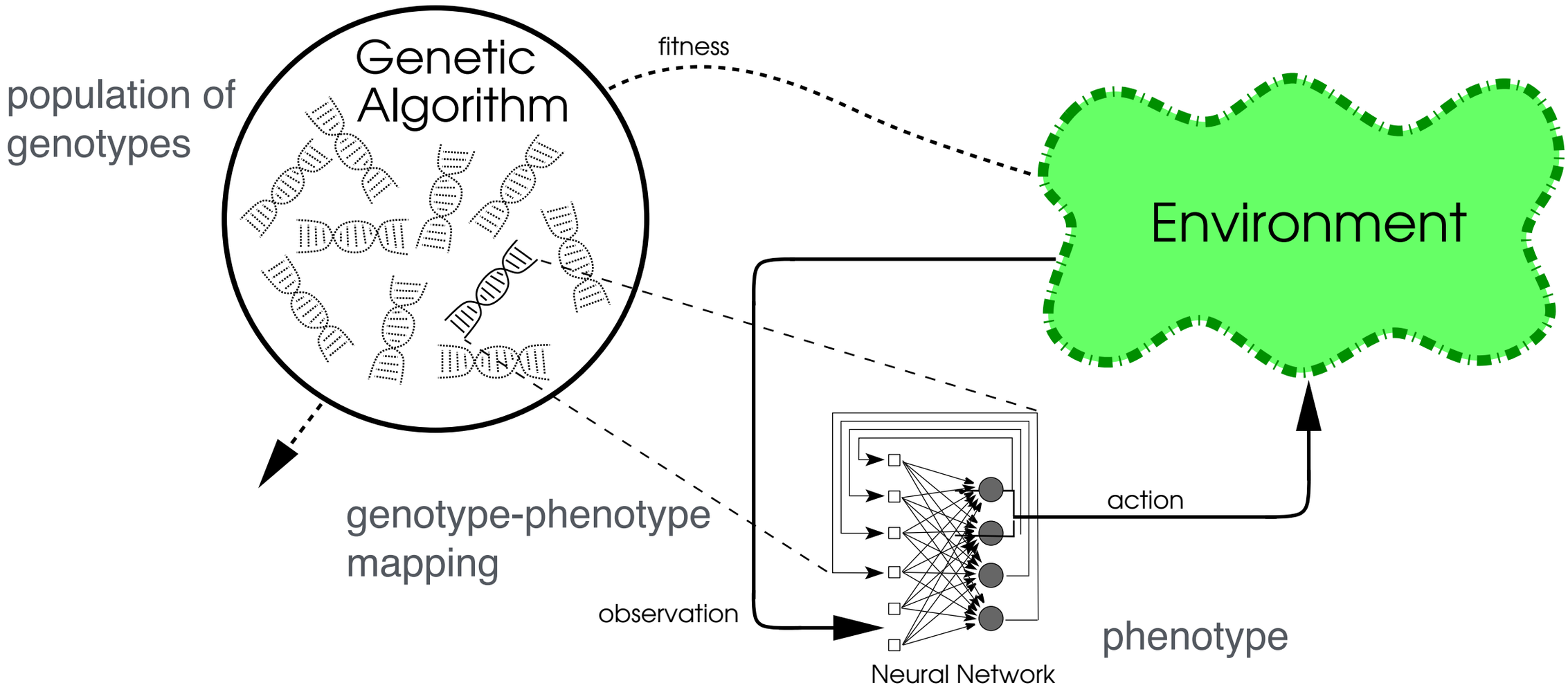
Note on terminology usage in Machine Learning

- sometimes people in ML use the expressions genome, genotype, chromosome interchangeably and just want to refer to the parameter set of the model under consideration (the differences are not as important as in biological systems)

Relationship with Biology

Consider the problem of finding good weights for a neural network for some task

- the set of weights for one neural network ca be regarded as a chromosome
- each single weight can be regarded as a gene
- the value of a weight can be regarded as an allele
- the neural network as a whole is regarded as the phenotype, it can interact with the environment, e.g. a RL agent playing a game



Selection Operator

How should we select parents to create offspring with high fitness ?

- the answer depends on the specific problem one is dealing with (heuristic)
- in general the selection methods are
  - deterministic or stochastic
  - with or without replacement

Truncated Selection

- M**ain Idea**:
  - select candidates based on their fitness ranking (deterministic)
  - candidates are ordered by fitness values
  - some proportion, p, (e.g. $p = 1/2, 1/3$, etc.), of the fittest candidates are then selected for reproduction
- P**roblem**
  - weaker candidates have no chance of getting selected for recombination
  - this can lead to low diversity of candidates
  - **diversity** is the variety and abundance of candidates at a given point in the search space and at a given time (generation)
  - losing diversity means approaching a state where all the candidates under investigation are similar to each other
  - low diversity can imply that the candidates get stuck in a local optimum and miss the global optimum

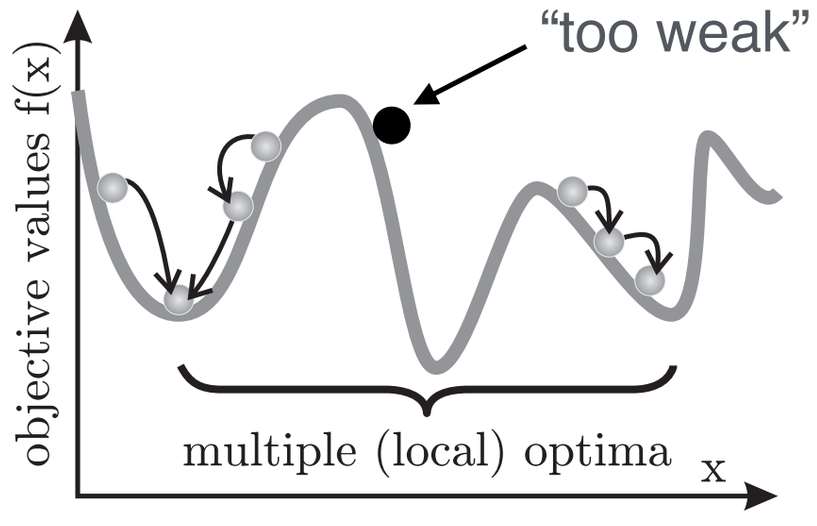


Fig. 1.19.c: Multimodal

Fitness Proportionate Selection

Fitness proportionate selection (also known as roulette wheel selection)

- main idea
  - fitter candidates should have a higher chance of being selected (stochastic)
  - the probability of being selected should be proportional to their fitness value
- given
  - population of n candidates: $P = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, ..., \boldsymbol{\theta}_n\}$
  - each candidate is encoded by a chromosome (parameter set) $\boldsymbol{\theta}_i$ with $\boldsymbol{\theta}_i \in \mathbb{R}^d$
  - scalar fitness function $f : \mathbb{R}^d \to \mathbb{R}$
- the candidate's probability of being selected is then given by

$$p(\boldsymbol{\theta}_i) = \frac{f(\boldsymbol{\theta}_i)}{\sum_{j=1}^{n} f(\boldsymbol{\theta}_j)}, \quad i = 1, ..., n$$

- scaling problem: If one individual has a very high fitness (outsider) compared to the remaining candidates, then the high fitness candidate is almost always selected for reproduction. This leads to low diversity.

Linear Ranking Selection

Linear ranking selection

- main idea: calculate a new fitness value for each individual based on its ranking position and then perform fitness proportionate selection with the new rank-based fitness values
- sort the candidates by fitness: $rank(\boldsymbol{\theta}_{\text{highest fitness}}) = n$ and $rank(\boldsymbol{\theta}_{\text{lowest fitness}}) = 1$
- assign a new fitness value to each candidate

$$\hat{f}(\boldsymbol{\theta}_i) = 2 - sp + 2 * (sp - 1)\frac{rank(\boldsymbol{\theta}_i)}{n - 1}$$

  where $sp \in [1.0, 2.0]$ is called the selective pressure.

- the selection pressure sp is the degree to which the fitter individuals are selected
- the higher the selection pressure, the more the fitter individuals are selected
- the selective pressure should be high enough to converge quickly towards a high-fitness candidate (global optimum), but not so high that we converge too soon (premature convergence to a local optimum).

Tournament Selection

- **pseudocode**
  - let T be the tournament size (between 2 and the population size)
  - select T candidates uniform randomly from the population and take the most fit as the tournament “winner” (deterministic)
  - put the winner in the mating pool (with replacement)
  - continue until enough individuals for mating have been found
- **the larger T, the higher is the selective pressure**
  - individuals with good fitness values create more and more offspring whereas the chance of worse solution candidates to reproduce decreases.
- **advantages**
  - absolute fitness values play no role (no scaling problem)
  - each tournament can be implemented in parallel

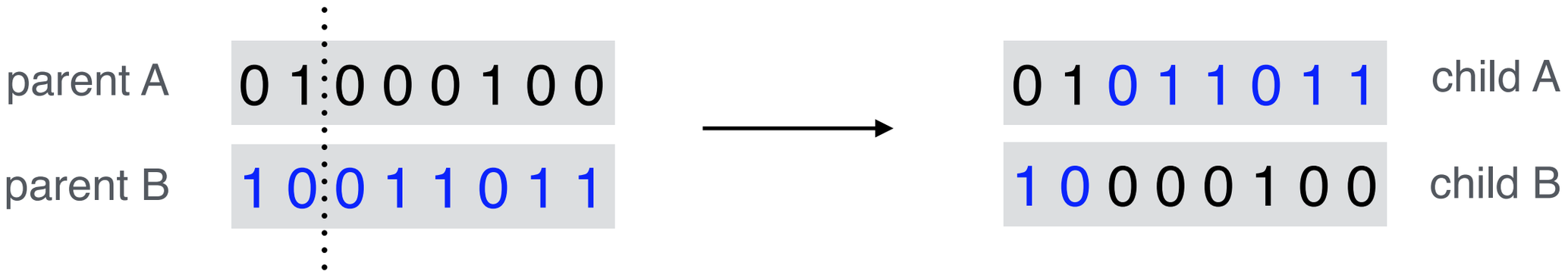
Reproduction

Generate new candidates (offspring) by mixing or altering the chromosomes of those members (parents) of the population that are selected for reproduction

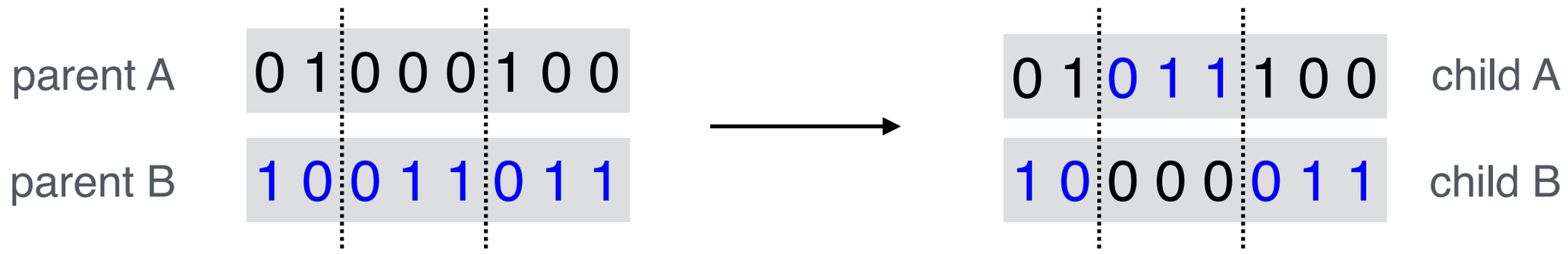
- **Crossover**: a form of recombination, select alleles from two parent chromosomes to form two new offspring chromosomes
- **Mutation**: randomly perturb some of the alleles of a parent chromosome to form a new offspring chromosome

Crossover Operator

1-point crossover: splitting position is chosen uniform randomly



2-point crossover: splitting positions are chosen uniform randomly



crossover rate = probability that the crossover operator will be applied to an arbitrary candidate

Mutation Operator

Binary Mutation: mutation position is chosen uniform randomly

parent: 0 1 **0** 0 0 1 0 0

↓ flip the bit

child: 0 1 **1** 0 0 1 0 0

Real-valued Mutation: mutation position is chosen uniform randomly

parent: -4.3 1.2 6.5 6.7 -0.3 **0.1** 0.8 -0.9

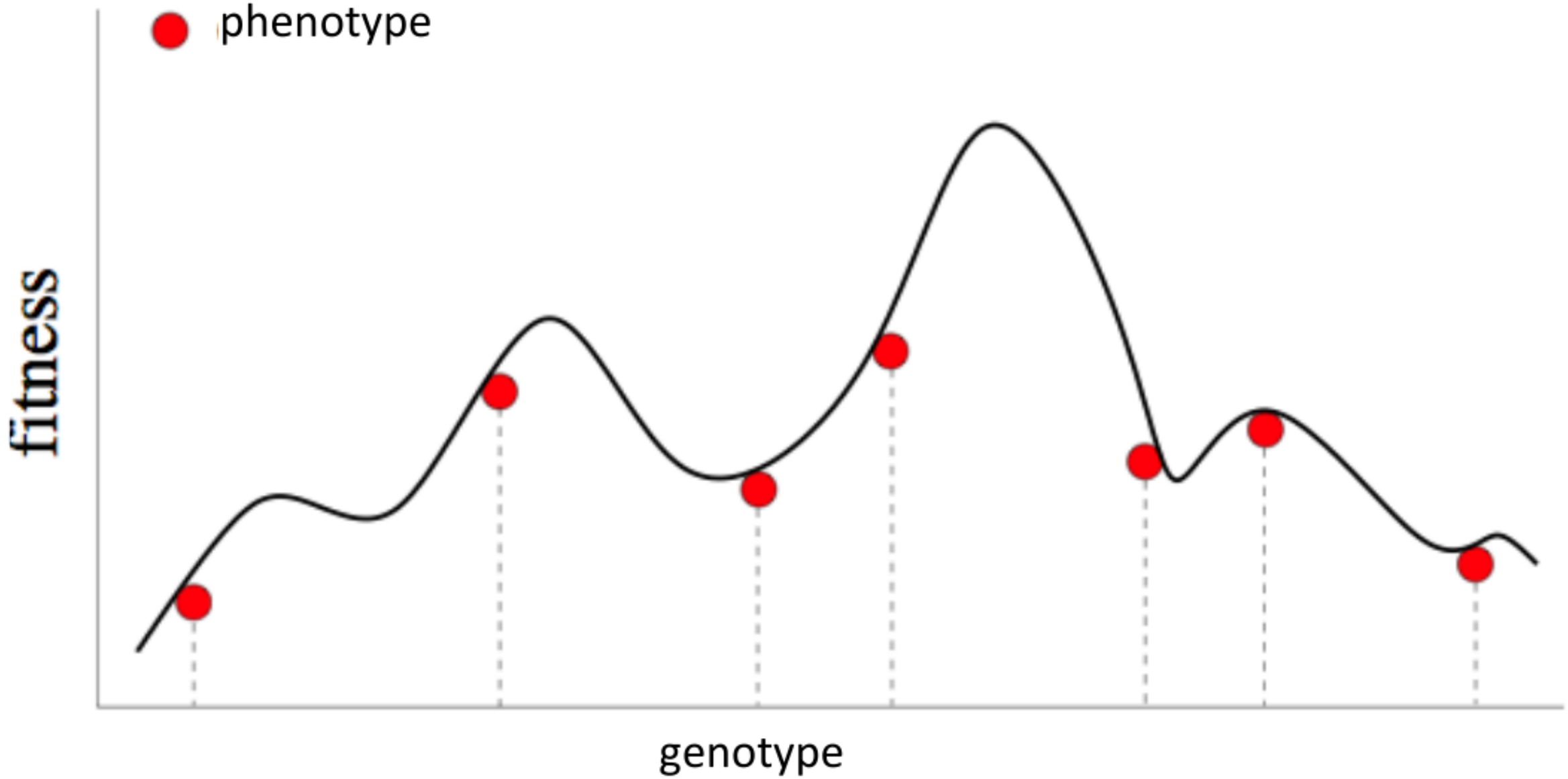
↓ add Gaussian noise, e.g. from N(0,1)

child: -4.3 1.2 6.5 6.7 -0.3 **1.7** 0.8 -0.9

mutation rate = probability that an arbitrary bit of an arbitrary candidate will be mutated

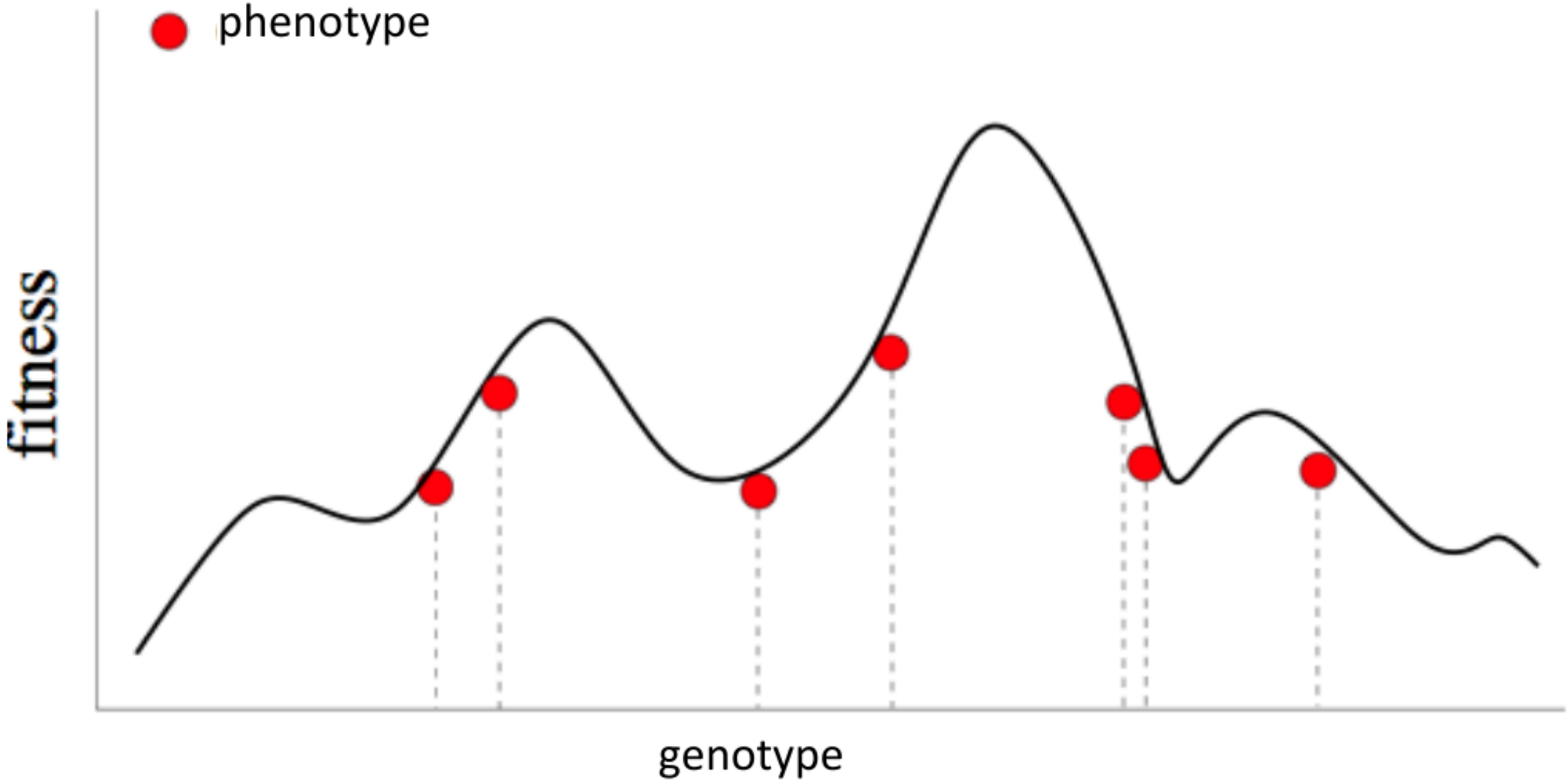
Example Behaviour in GA

Initial population: candidate solutions are distributed throughout search space



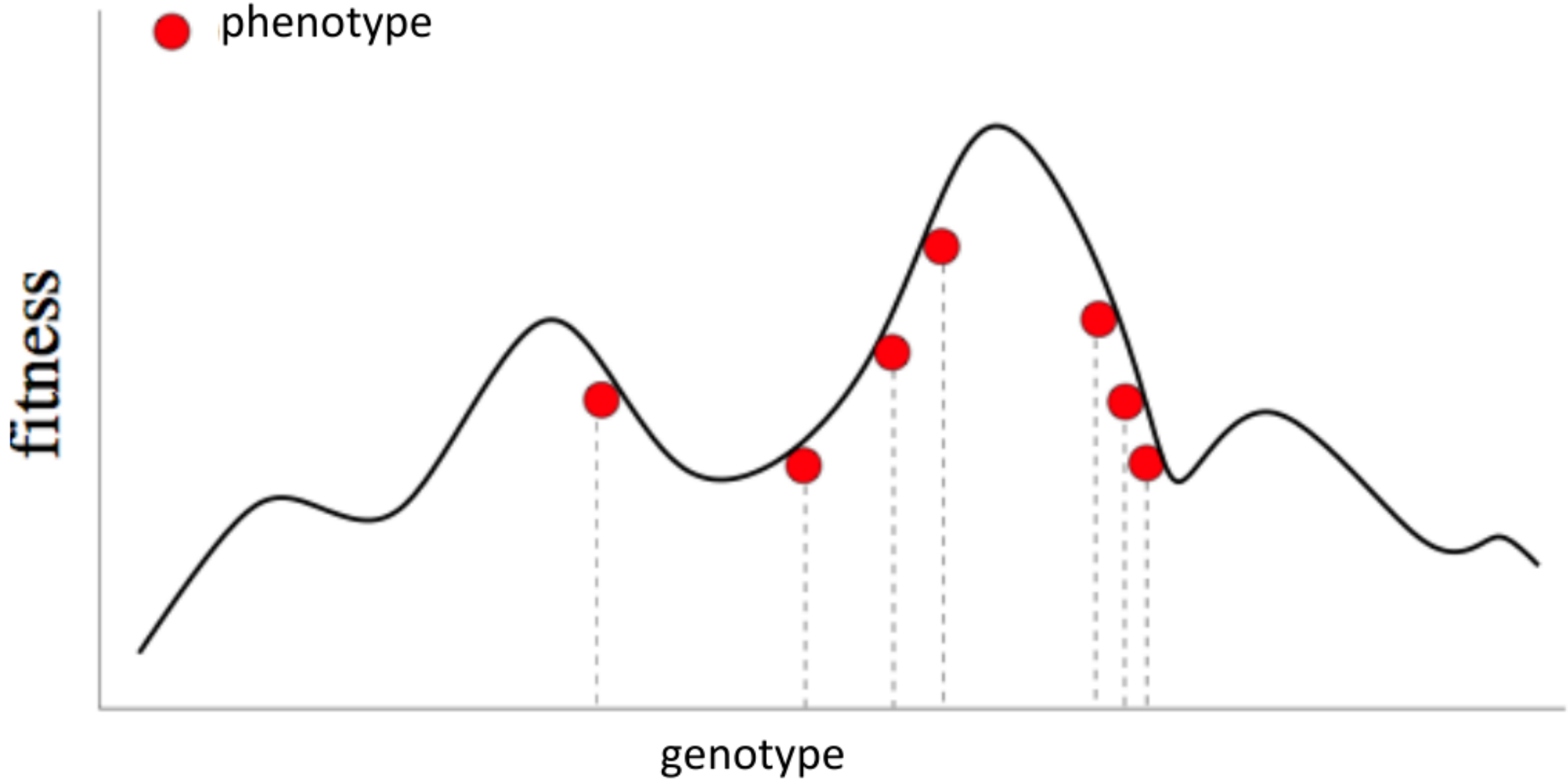
Example Behaviour in GA

After some number of generations …



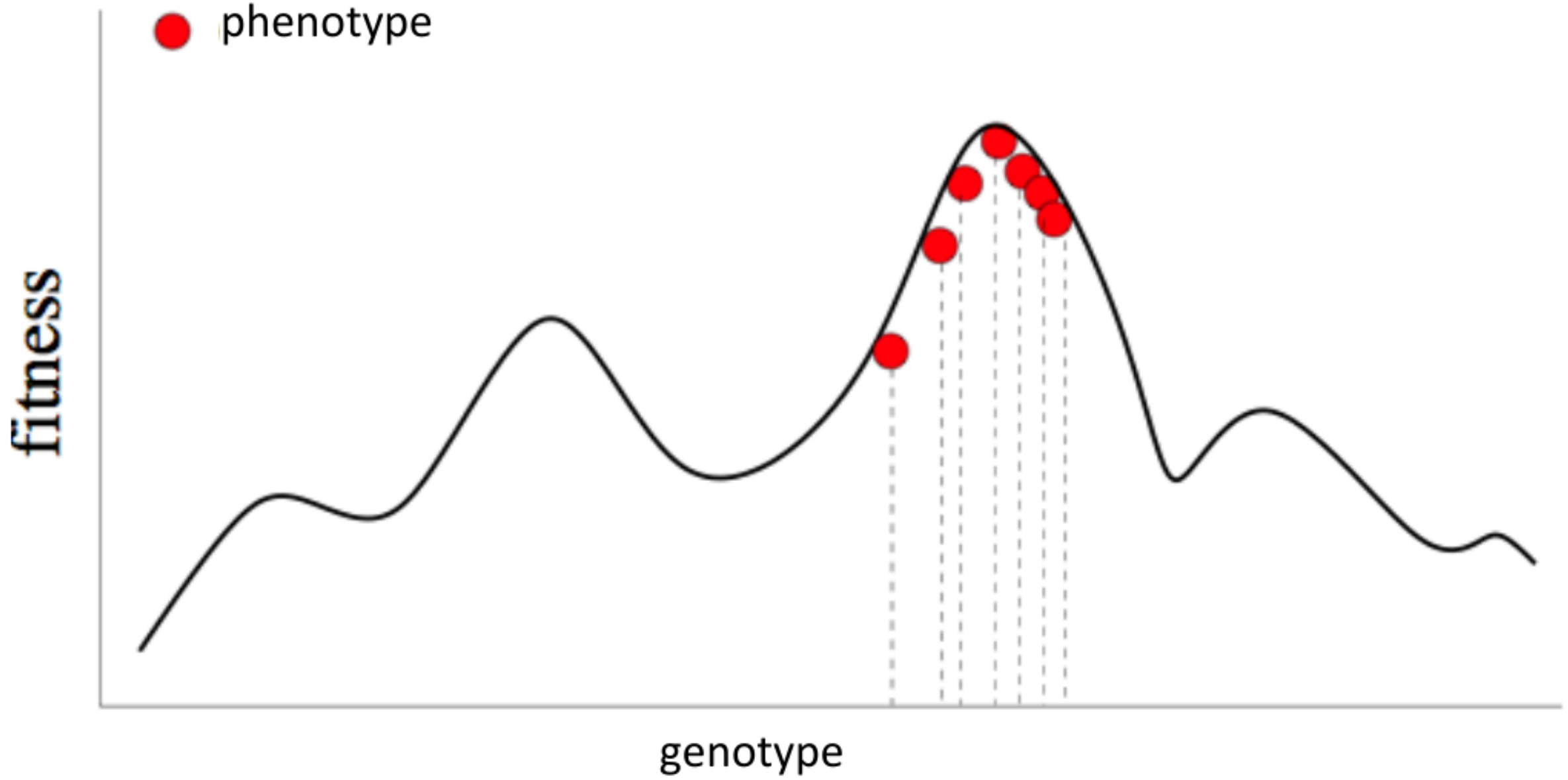
Example Behaviour in GA

After some more generations …



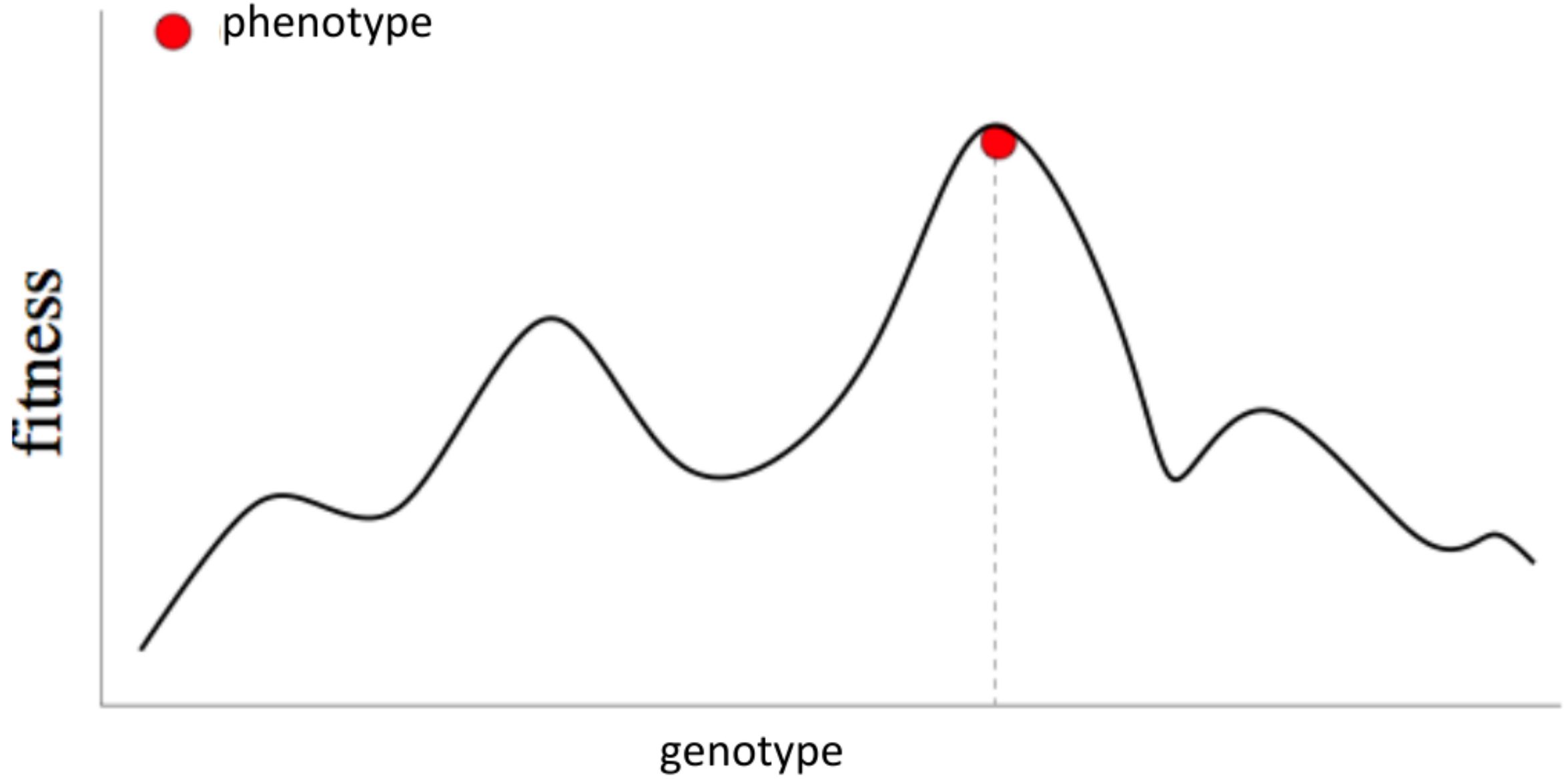
Example Behaviour in GA

Some more generations …



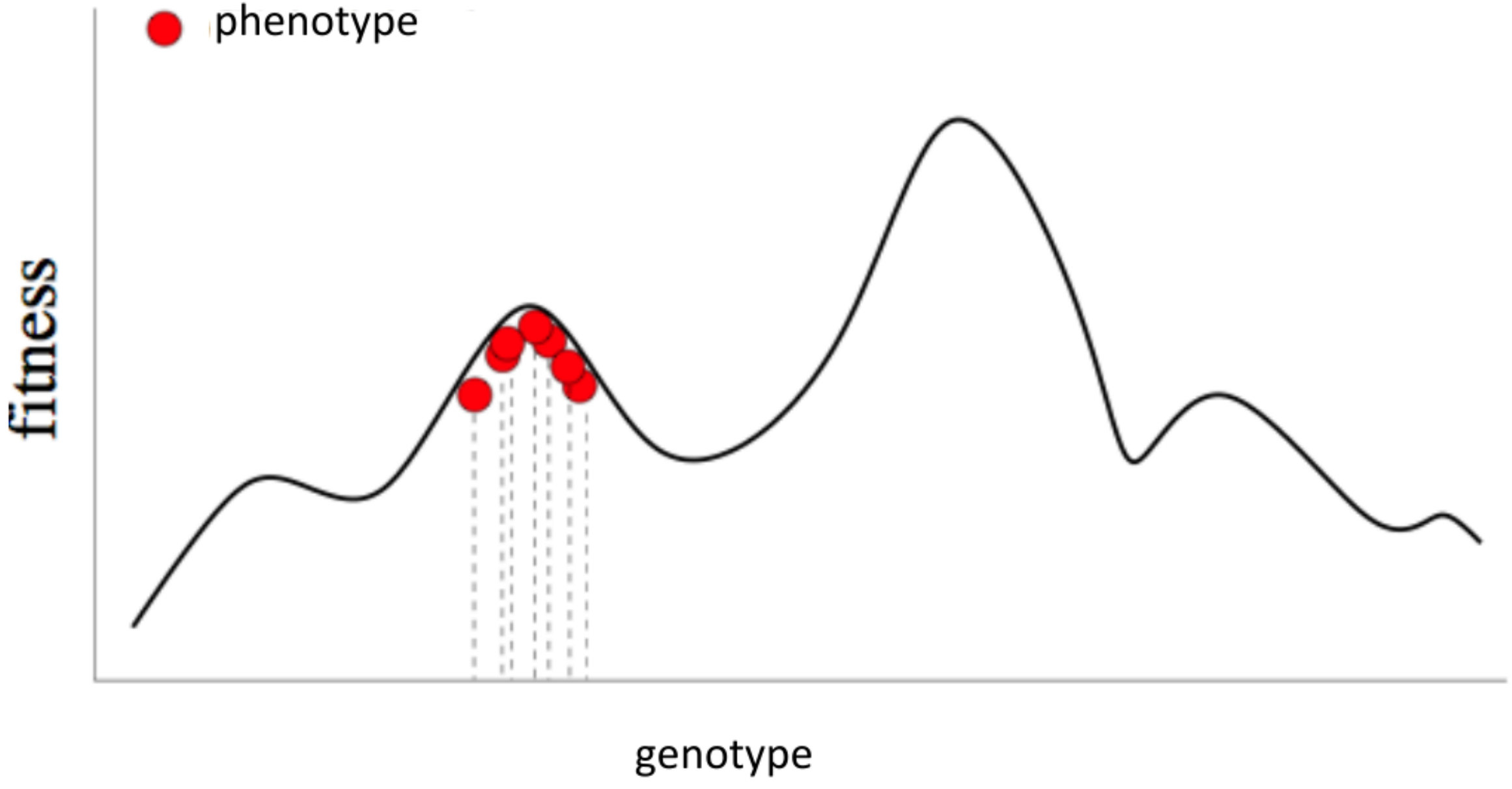
Example Behaviour in GA

All individuals look almost the same after many generations of reproduction (convergence to global optimum)



Premature Convergence in GA

Here: Population converges too quickly and misses global optimum



Premature Convergence

How can we avoid premature convergence to local optima ?

- use larger population
  - requires more evaluations and therefore takes longer to converge
- reduce selective pressure, i.e. make the search less greedy
  - it could take much longer to converge
- increase the crossover/mutation rate
  - this adds diversity but could also disrupt building blocks

There is a trade-off between selective pressure and diversity

- selective pressure should be high enough to converge quickly towards good solutions, but not so high that we converge too soon
- diversity should be high enough so that we are less likely to miss a good solution, but not so high that we don’t converge
- similar to exploitation/exploration tradeoff in reinforcement learning

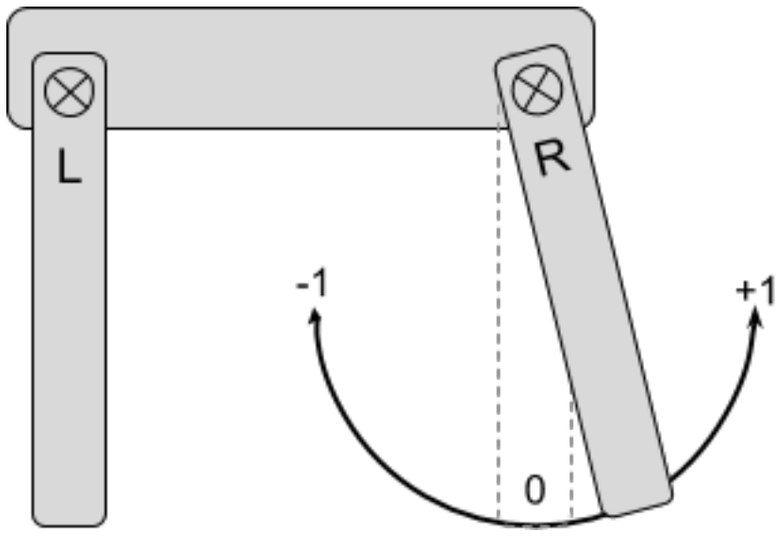
GA Examples

Walking Robot with GA

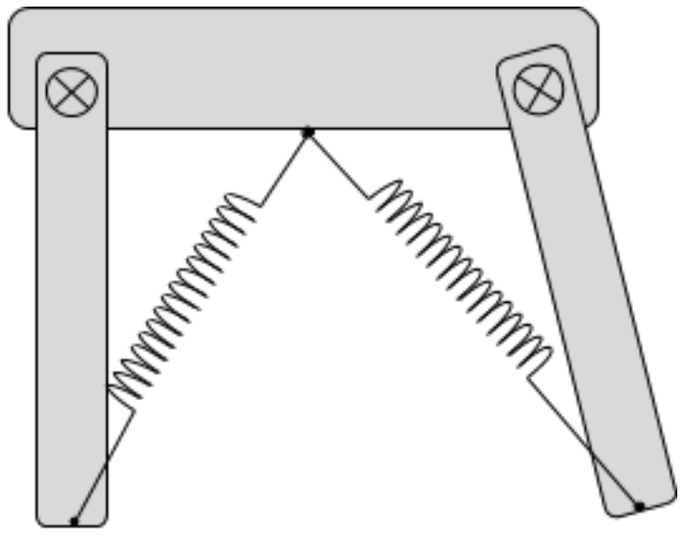
[https://www.alanzucconi.com/2016/04/06/evolutionary-coputation-1/]

- goal: create a simple walking robot
- movement is performed by extending and contracting springs
- for simplicity: the rotation angle *s(t)* (extension/contraction) of a leg follows a sinus wave with
  - wave length *p*
  - amplitude ranging from *m* to *M*
  - shifted by a phase *o*

$$s(t) = \frac{M - m}{2}\left(1 + \sin\left((t + o)\frac{2\pi}{p}\right)\right)$$



contraction: -1



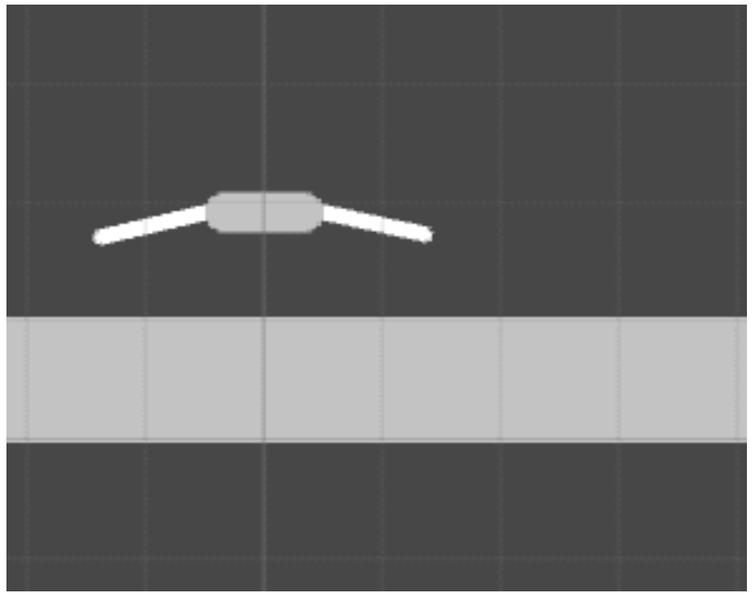
extension: +1

Walking Robot with GA

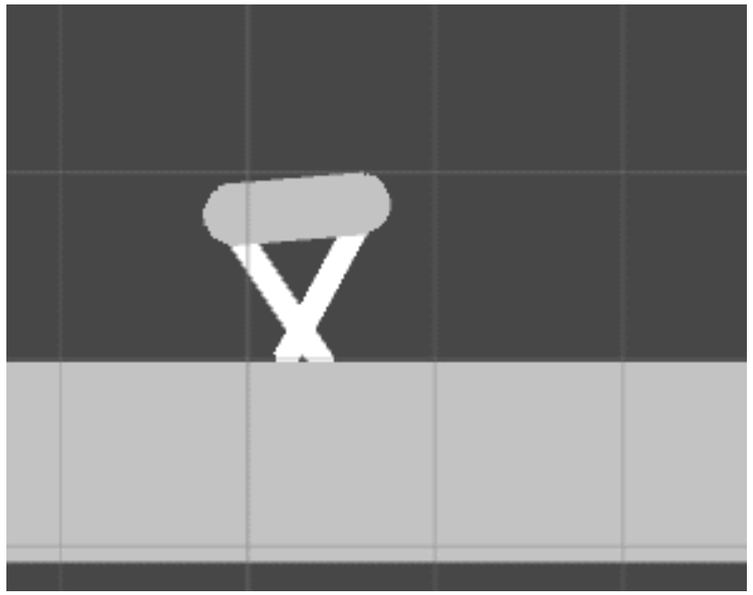
- We have two legs and four parameters per leg. So, one candidate robot (phenotype) can be encoded by the tuple (chromosome):

$$(m_1, M_1, o_1, p_1, m_2, M_2, o_2, p_2)$$

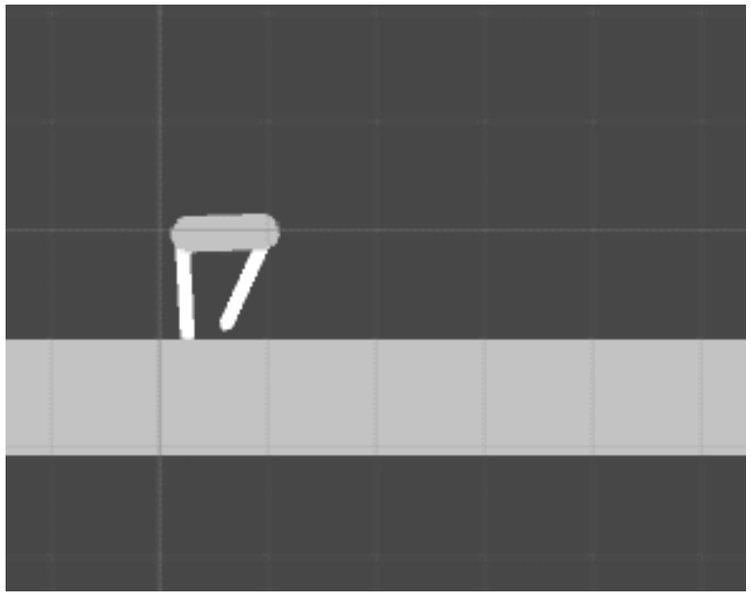
- Mutation: randomly perturb one of the above parameters
- How to define the fitness function (reward) ?



reward = the distance how far the robot travelled



reward = stay up

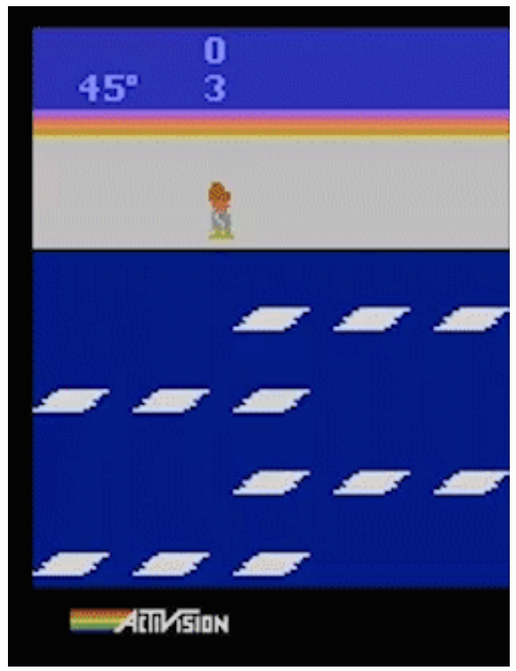


reward = balance between distance and stay up

GA in State-of-the Art RL

[arXiv: 1712.06567]

- goal: create an agent to play Atari games
- “genotype” = weights of the policy neural network (~ 4 million)
- “phenotype” = agent playing an Atari game
- GA can make use of massive parallelisation (next slide)
- GA can be competitive on certain games with DQN, A3C especially in games with sparse rewards



Algorithm 1: Conceptual description of GA used in [arXiv:1712.06567]

initialise the first generation $P^{(1)} = \{\boldsymbol{\theta}_1^{(1)}, \boldsymbol{\theta}_2^{(1)}, ..., \boldsymbol{\theta}_n^{(1)}\}$ of size n
for *each generation* $(g = 2, 3, ..., G)$ **do**
    evaluate the fitness of each candidate in $P^{(g-1)}$ and sort in descending order
    for *each candidate* $(1, 2, ..., n)$ **do**
        select one of the t $(t < n)$ fittest candidates from $P^{(g-1)}$, call it $\boldsymbol{\theta}^{(g-1)}$
        create offspring with Gaussian noise: $\theta^{(g)} = \theta^{(g-1)} + \epsilon, \quad \epsilon \sim N(0, \sigma)$
        add offspring $\boldsymbol{\theta}^{(g)}$ to new generation $P^{(g)}$
    end
end

GA in State-of-the Art RL

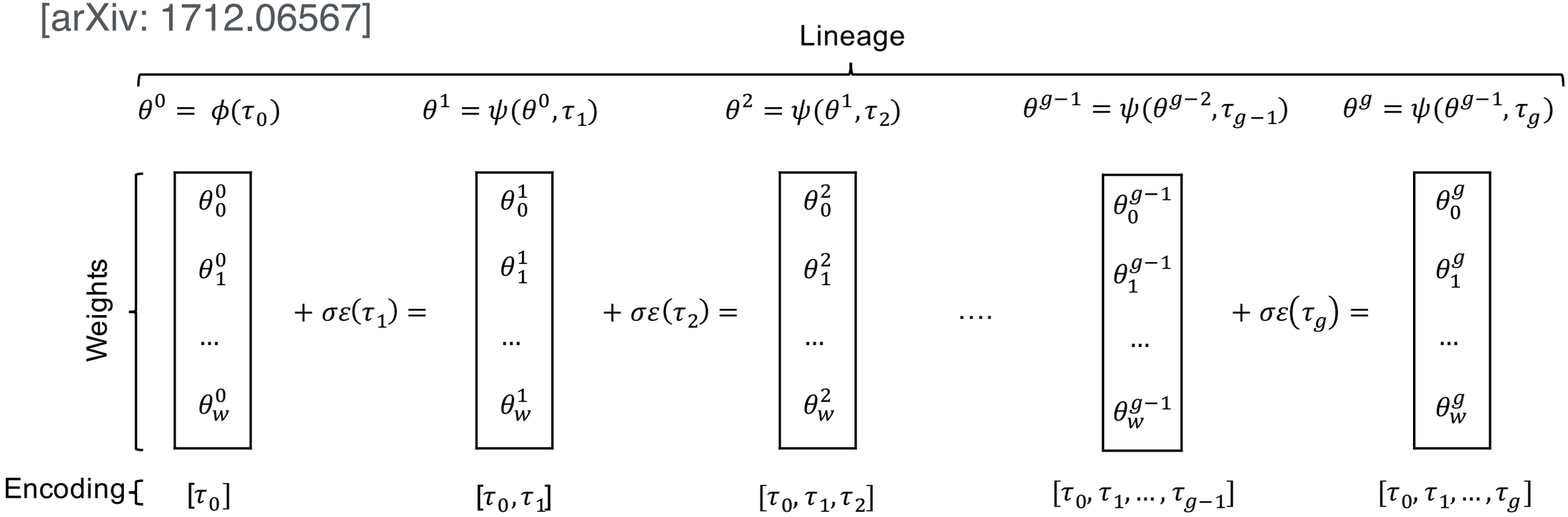
[arXiv: 1712.06567]

	DQN	ES	A3C	RS	GA	GA
Frames	200M	1B	1B	1B	1B	6B
Time	∼7-10d	∼ 1h	∼ 4d	∼ 1h or 4h	∼ 1h or 4h	∼ 6h or 24h
Forward Passes	450M	250M	250M	250M	250M	1.5B
Backward Passes	400M	0	250M	0	0	0
Operations	1.25B U	250M U	1B U	250M U	250M U	1.5B U
amidar	**978**	112	264	143	263	377
assault	4,280	1,674	**5,475**	649	714	814
asterix	4,359	1,440	**22,140**	1,197	1,850	2,255
asteroids	1,365	1,562	**4,475**	1,307	1,661	2,700
atlantis	279,987	**1,267,410**	911,091	26,371	76,273	129,167
enduro	**729**	95	-82	36	60	80
frostbite	797	370	191	1,164	**4,536**	**6,220**
gravitar	473	**805**	304	431	476	764
kangaroo	7,259	**11,200**	94	1,099	3,790	**11,254**
seaquest	**5,861**	1,390	2,355	503	798	850
skiing	-13,062	-15,443	-10,911	-7,679	†**-6,502**	†**-5,541**
venture	163	760	23	488	**969**	†**1,422**
zaxxon	5,363	6,380	**24,622**	2,538	6,180	7,864

sometimes it is worse to follow the gradient than sample locally in the parameter space

Parallelisation in GA

[arXiv: 1712.06567]



- instead of transmitting the whole one-million dimensional parameter vector from one worker to the next worker, we only need to exchange the encoding seeds
- consider 1000 generations with each 100 candidates
  - exchange 100 x 1000 numbers (seeds) instead of 100 x 1 million numbers
  - speed improvement of roughly O(1000)
- “With our GPU-enabled implementation, on one modern desktop we can train Atari in ~4 hours or ~1 hour distributed on 720 cores”