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”)

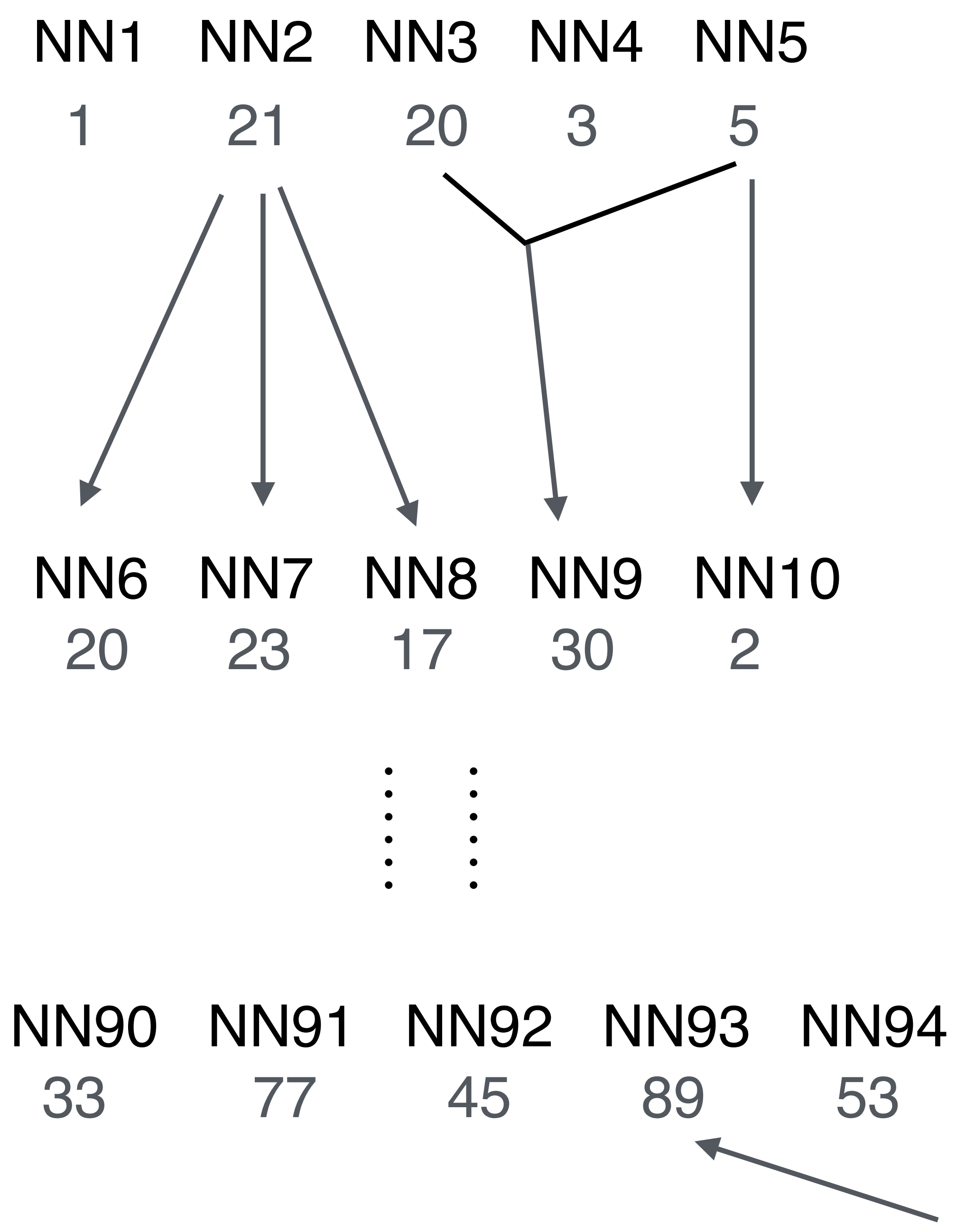
- in RL, one agent modifies its internal structure (policy) to maximise its reward when interacting with the environment
- in supervised/unsupervised learning (e.g. NN, SVM, HMM), there is one set of objective parameters to be optimised

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

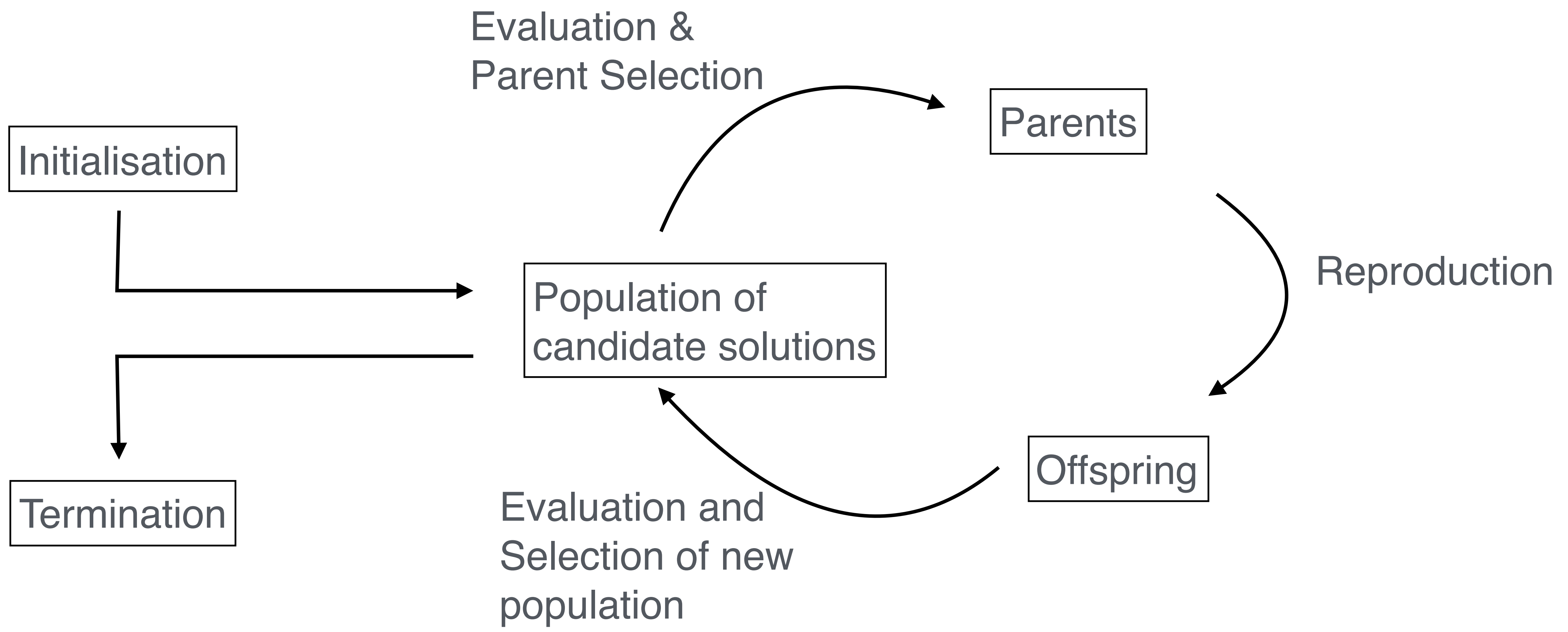
new population of 5 neural networks

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

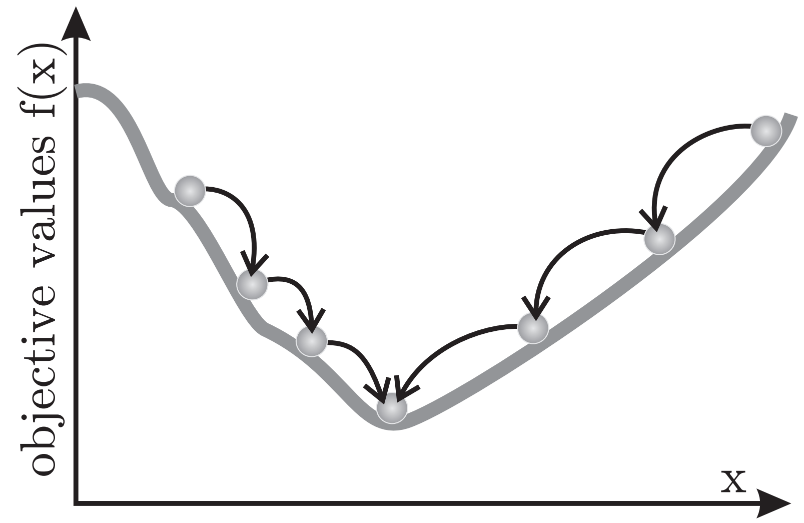


Fig. 1.19.a: Best Case

- 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

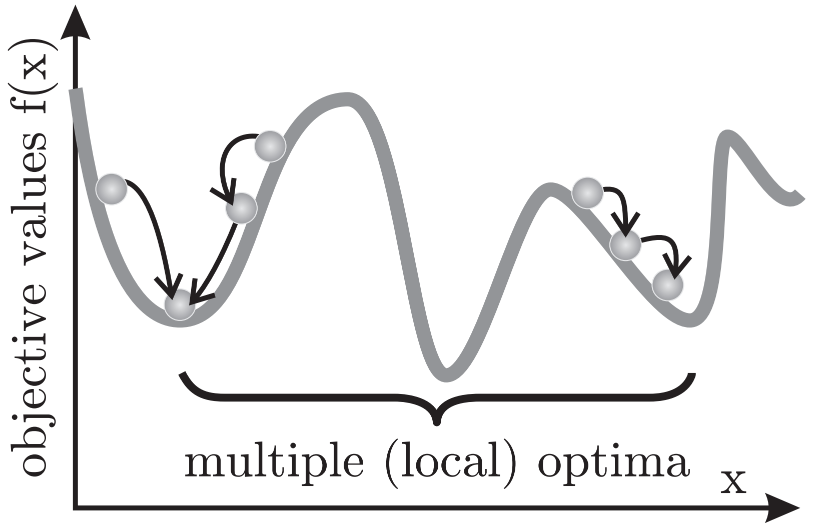


Fig. 1.19.c: Multimodal

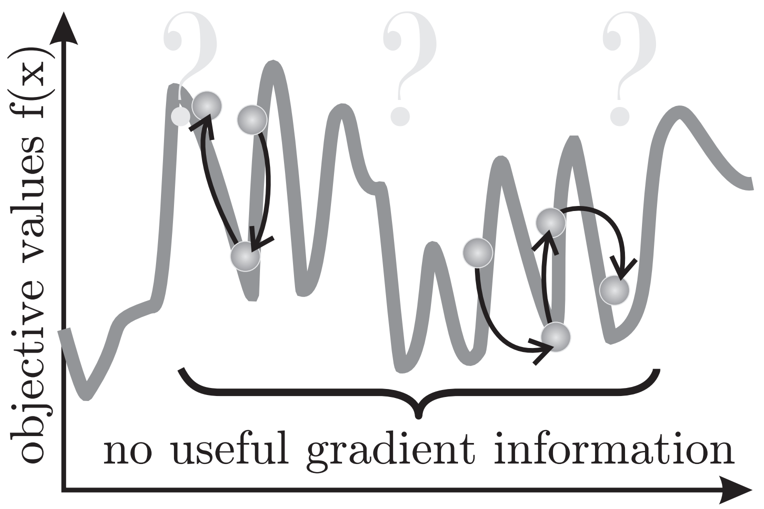


Fig. 1.19.d: Rugged

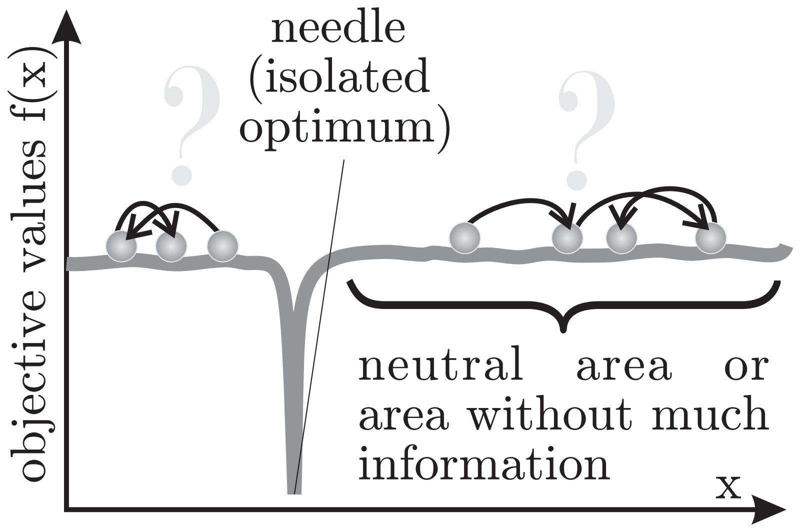

needle (isolated optimum)
objective values f(x)
neutral area or area without much information
x

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

[Weise: Global Optimisation Algorithm - Theory and Application]

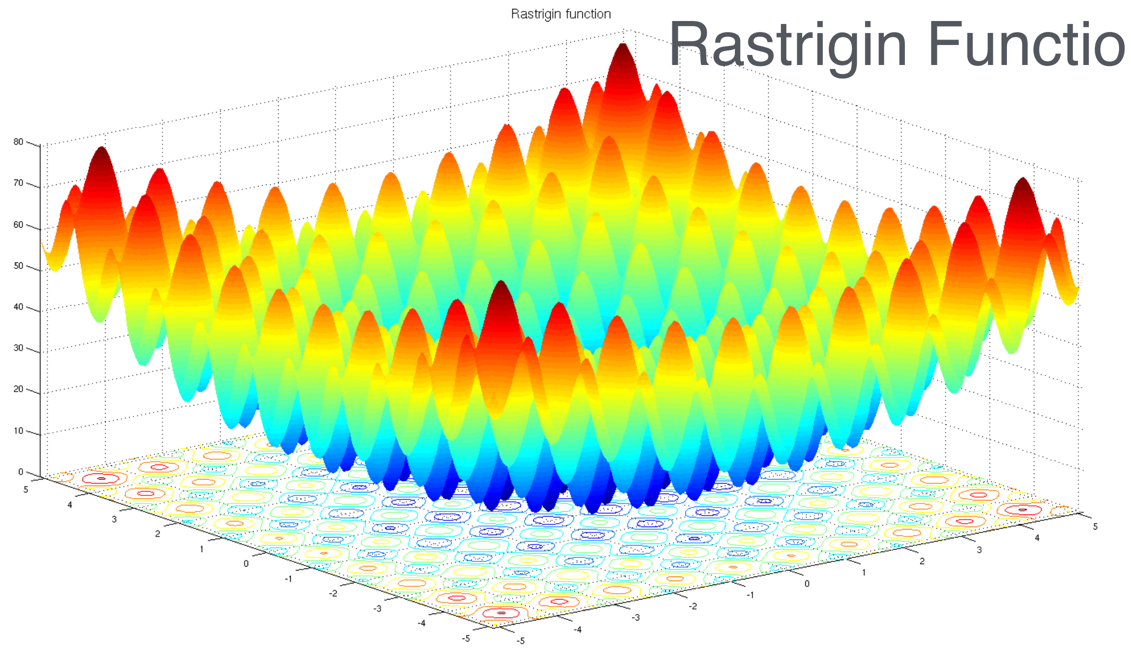
Why EAs ?

- **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

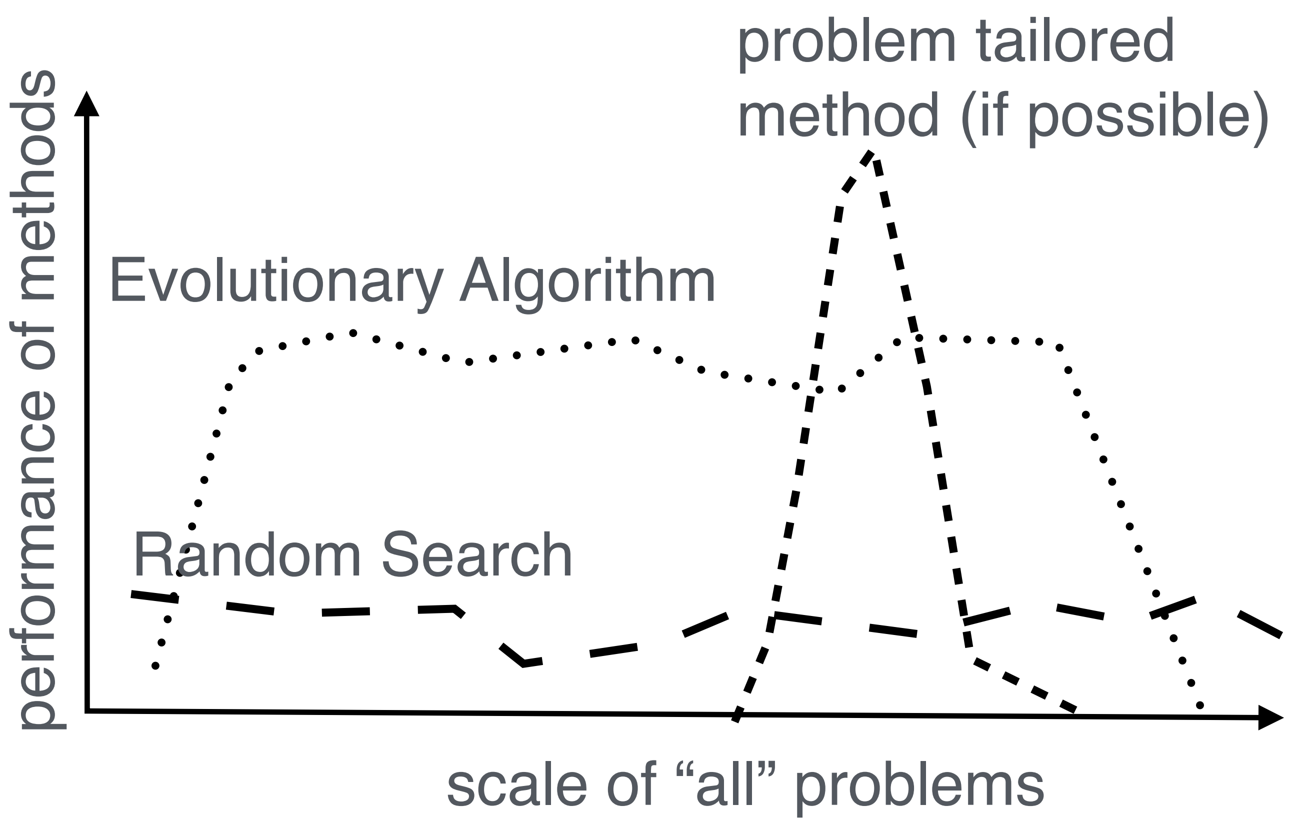
- **if we know little about the objective function (little domain knowledge)**
  - in EA we only need to know how to evaluate the fitness function

- **straightforward parallelisation**

- **conceptual simplicity**



Rastrigin Function



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) (Cramer 1985)
   - “tree (data structure)” representation of candidates
4. Evolutionary Programming (EP) (Fogel 1966)
   - candidates represented by “finite state machines”

Focus of this lecture

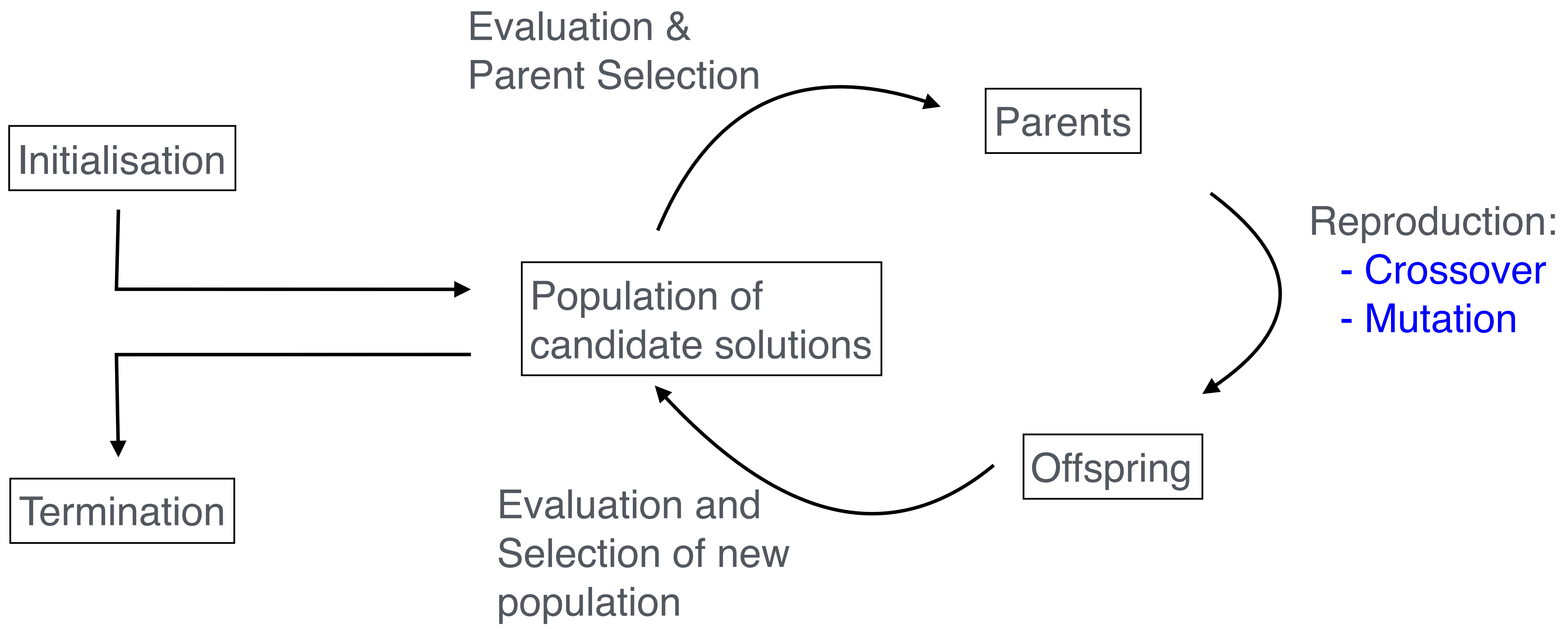
“recent” works:

[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)

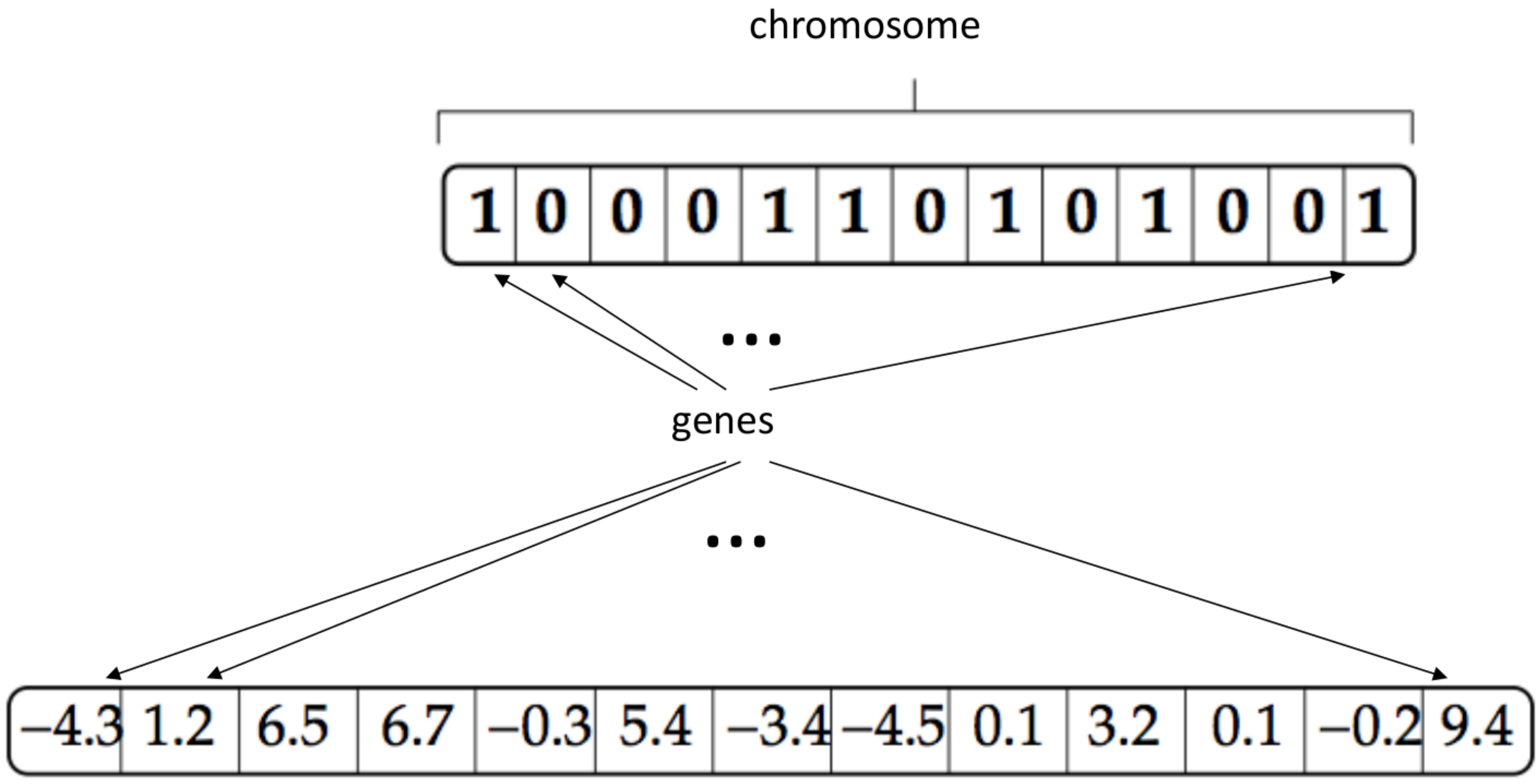


1. Initialise population with random candidate solutions
2. Evaluate each candidate (in parallel) and assign (scalar) “fitness” scores
3. Repeat, until a termination condition is fulfilled
    1. Select fittest candidates in the population for reproduction (parents)
    2. Reproduce new candidates (offspring)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

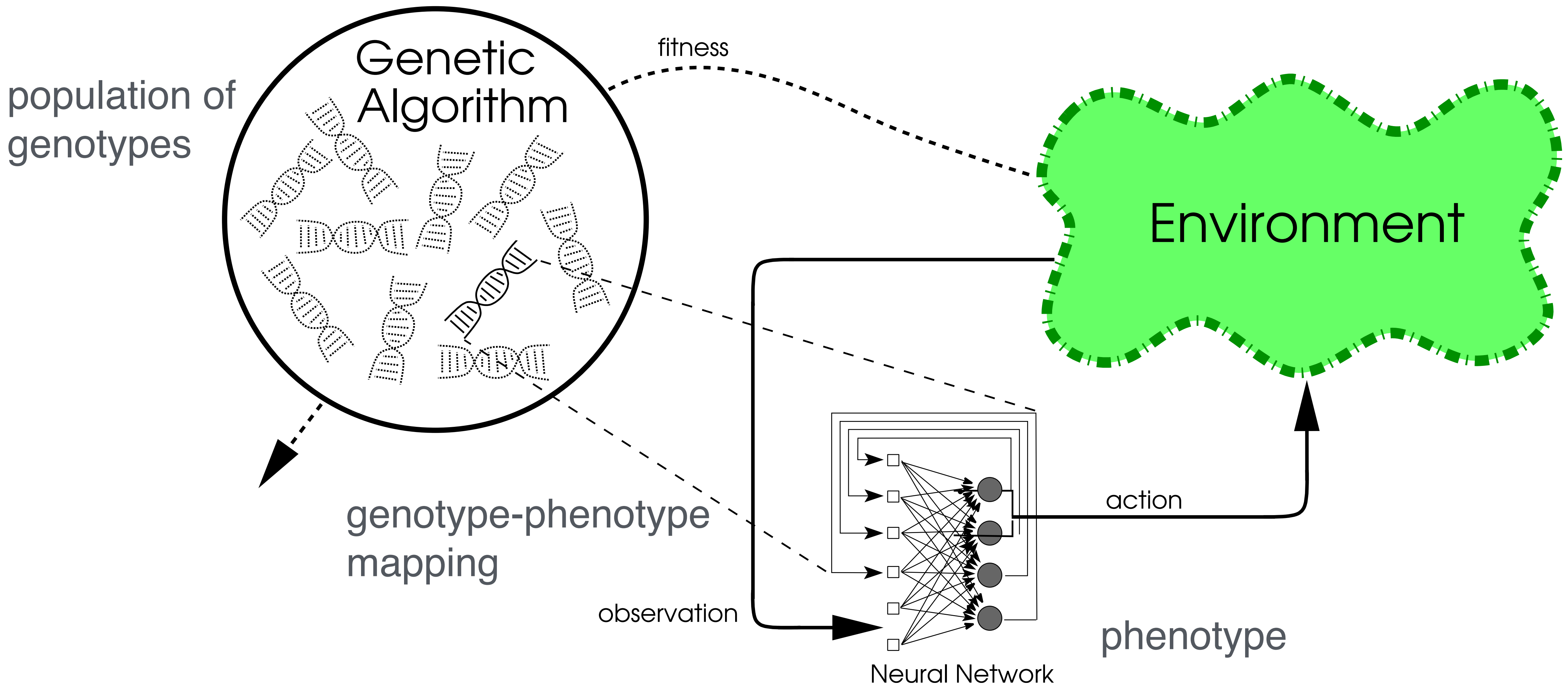
Note on terminology usage in Machine Learning

- 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

- **Main 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
- **Problem**
  - 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

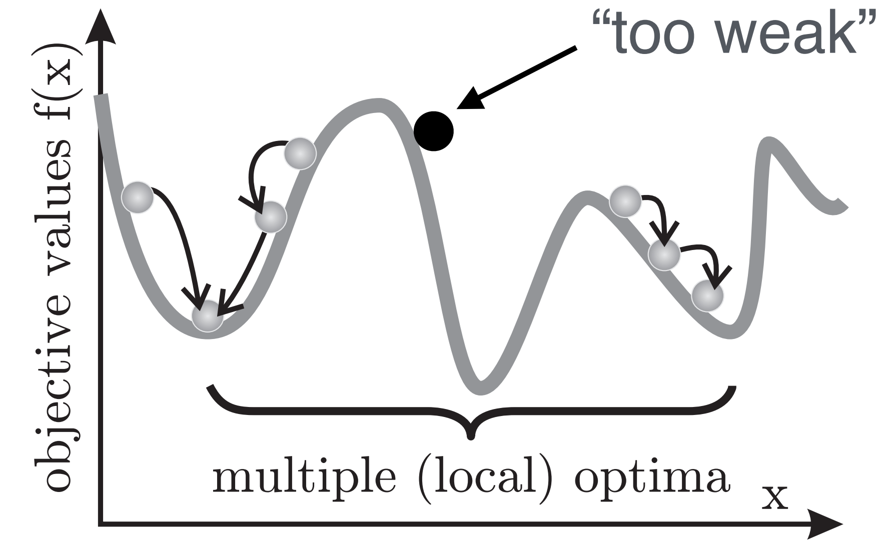


Fig. 1.19.c: Multimodal

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 $f : \mathbb{R}^d \rightarrow \mathbb{R}$
- 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

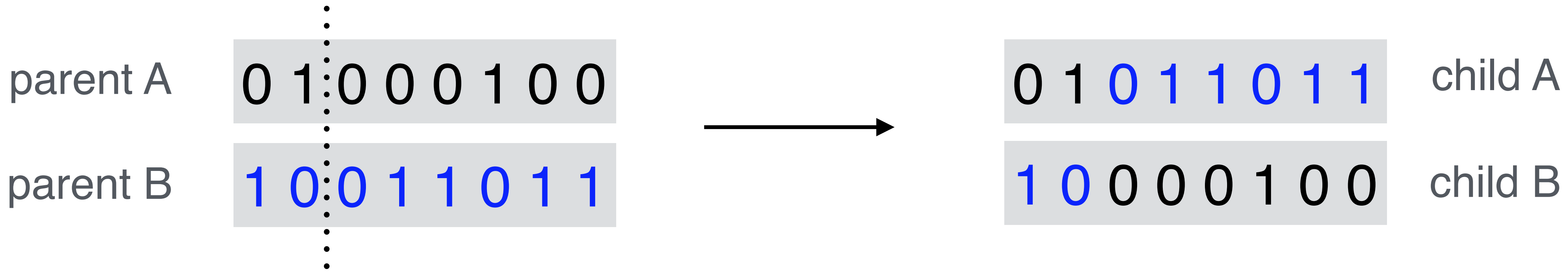
Reproduction

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

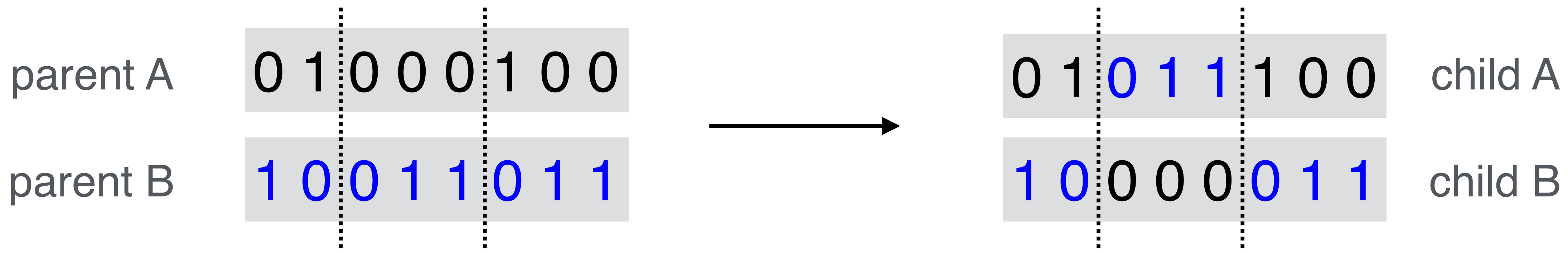
- **Crossover**: a form of recombination, select alleles from two parent chromosomes to form two new offspring chromosomes
- **Mutation**: randomly perturb some of the alleles of a parent chromosome to form a new offspring chromosome

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

$\downarrow$ 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

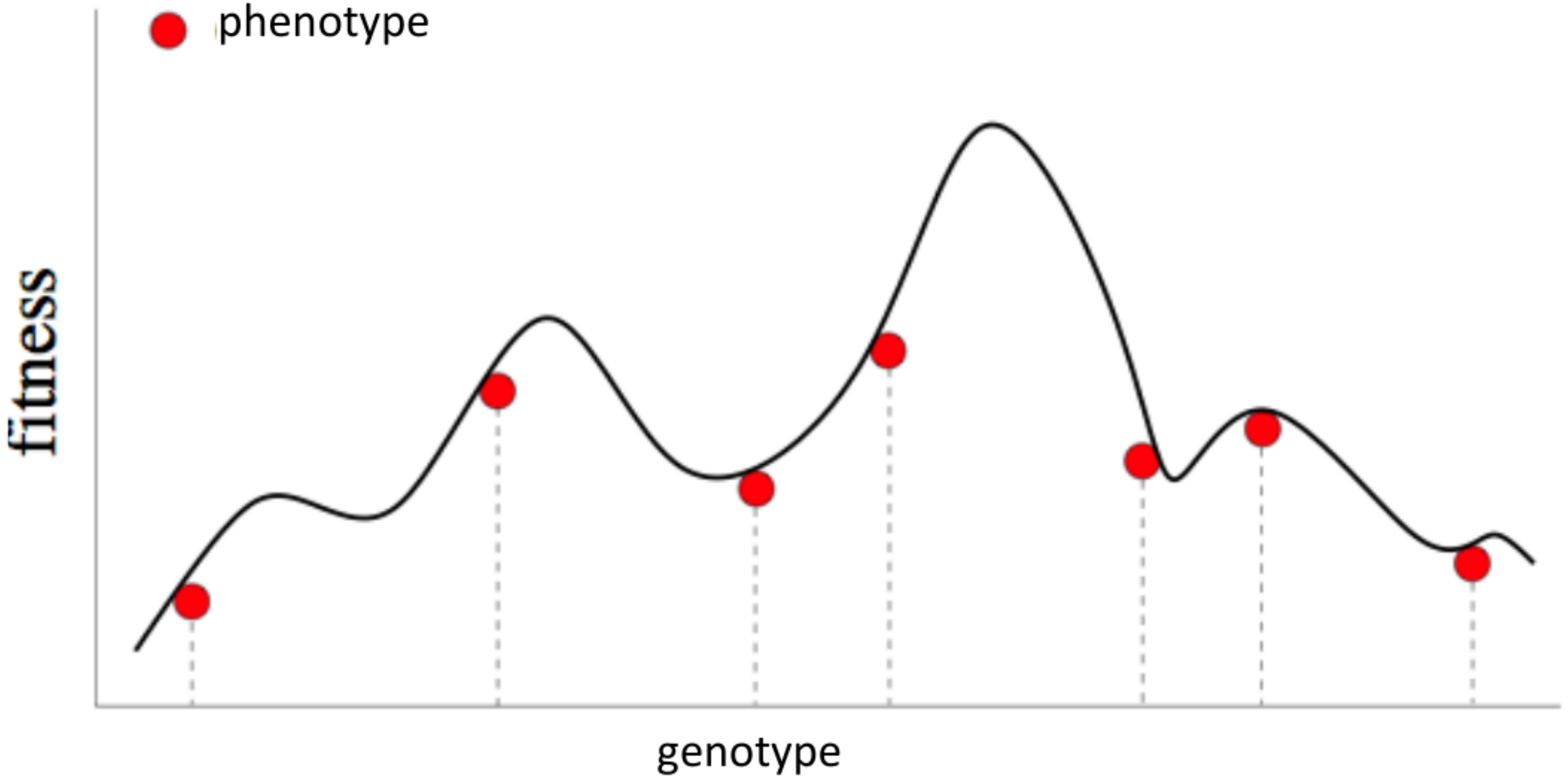
$\downarrow$ add Gaussian noise, e.g. from N(0,1)

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

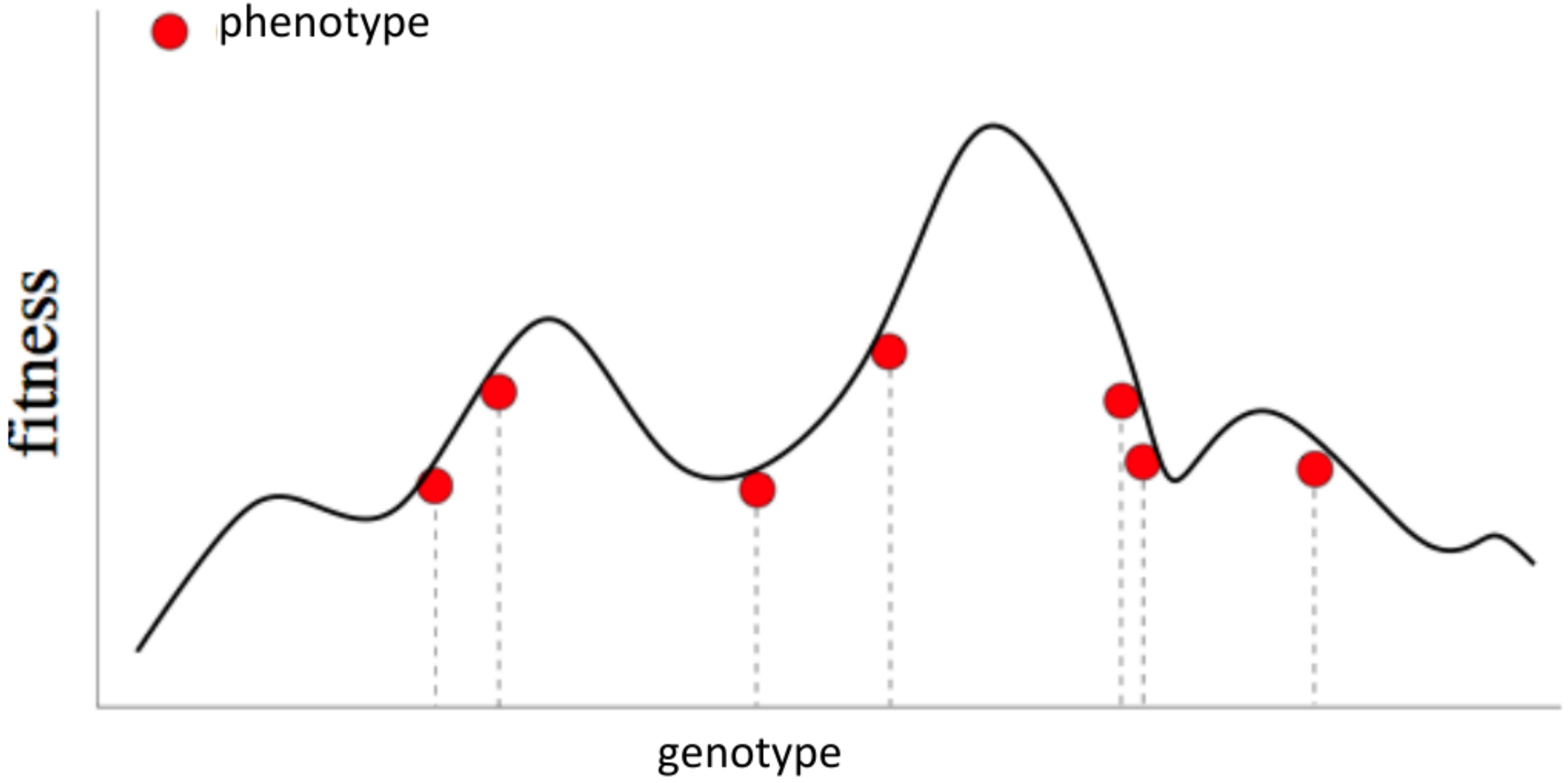
Example Behaviour in GA

Initial population: candidate solutions are distributed throughout search space



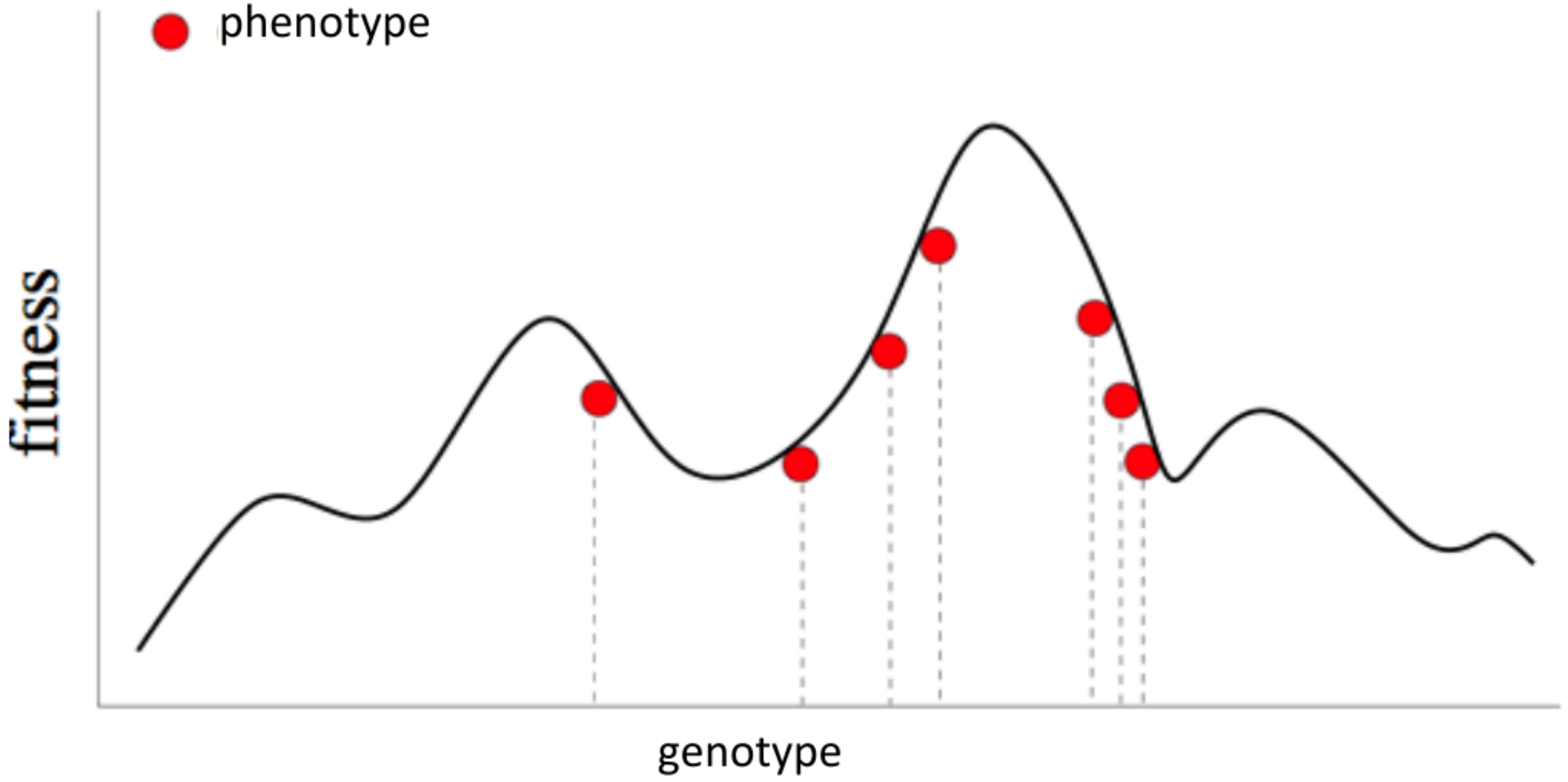
Example Behaviour in GA

After some number of generations …



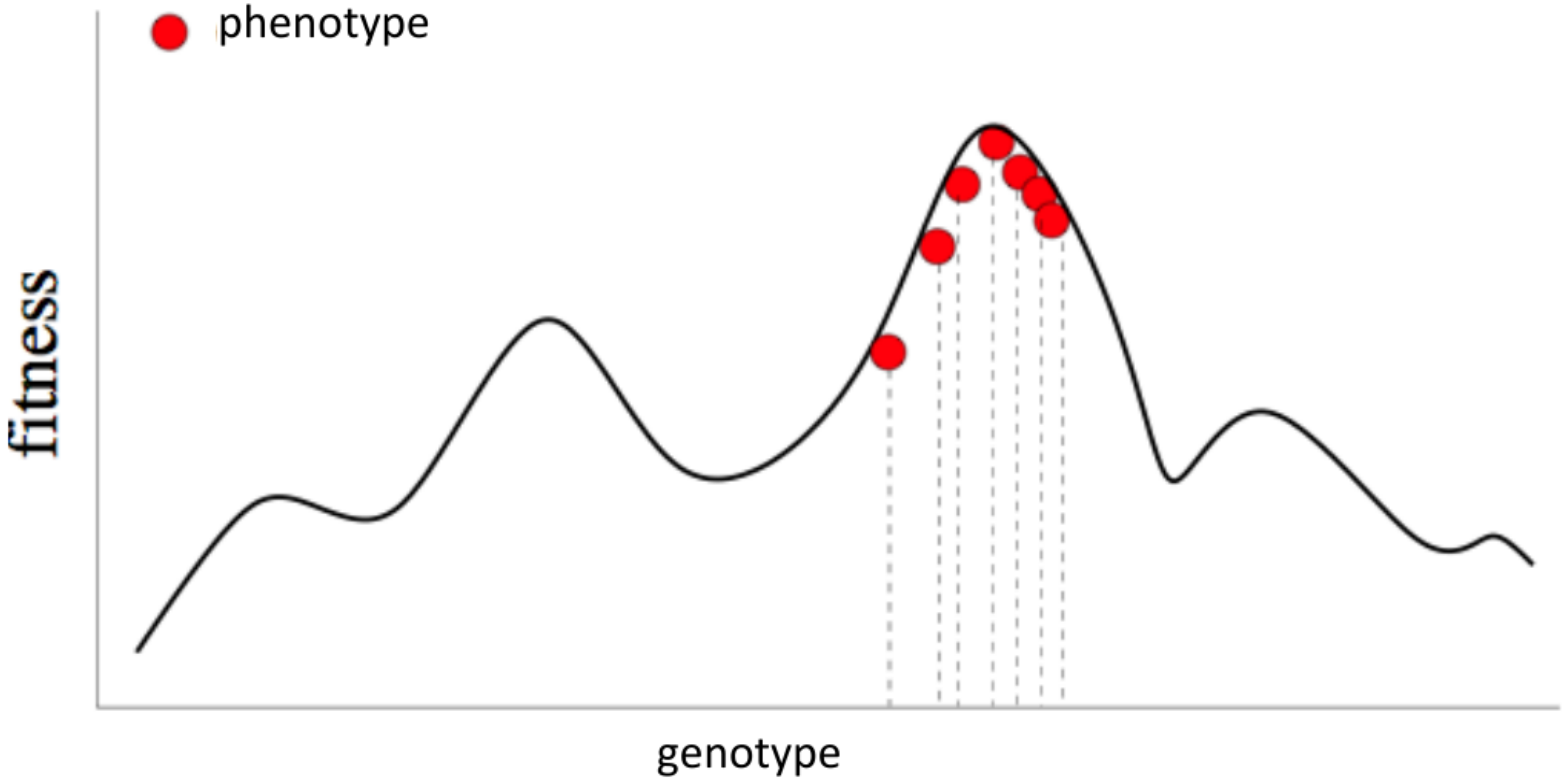
Example Behaviour in GA

After some more generations …



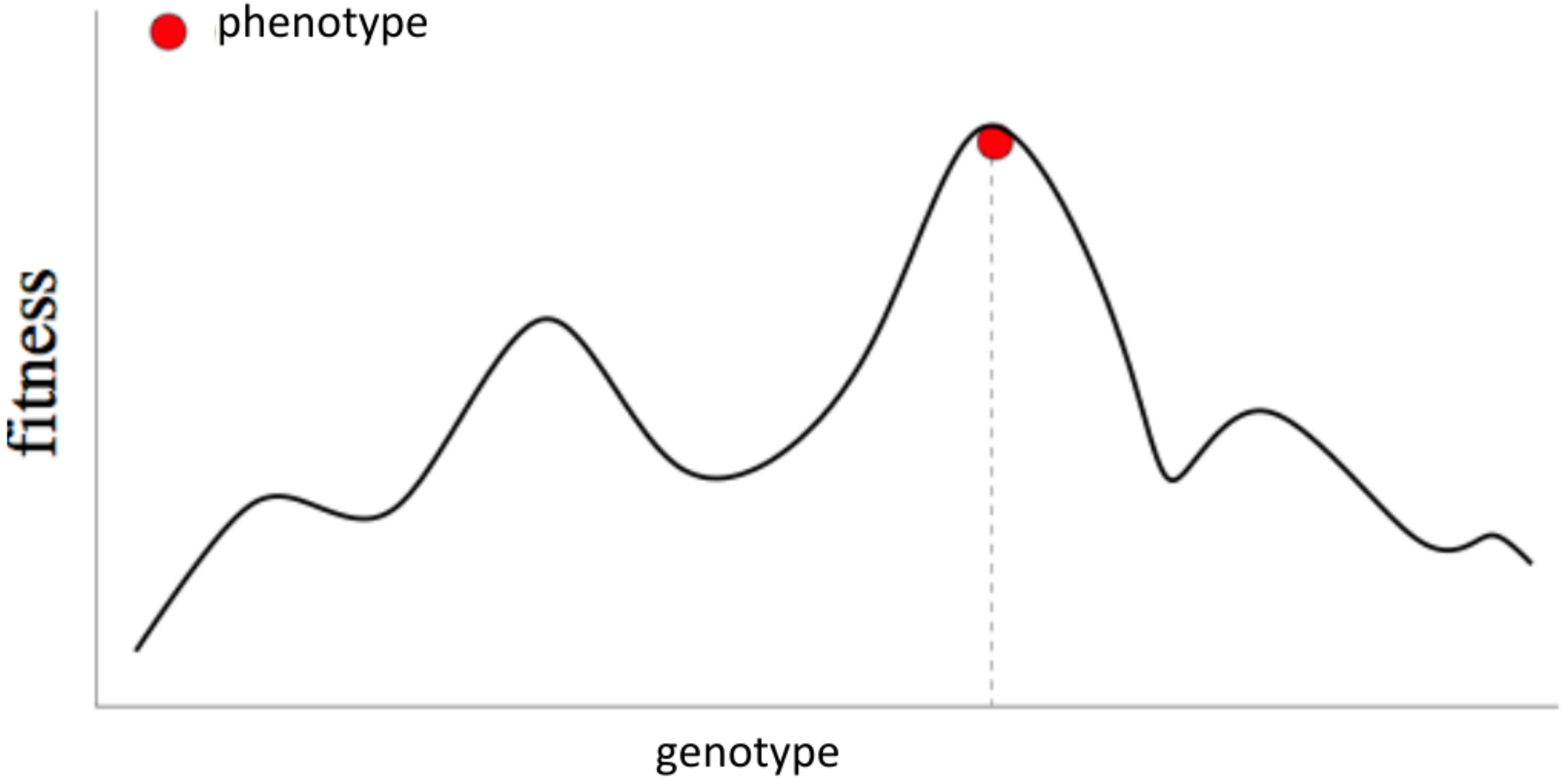
Example Behaviour in GA

Some more generations …



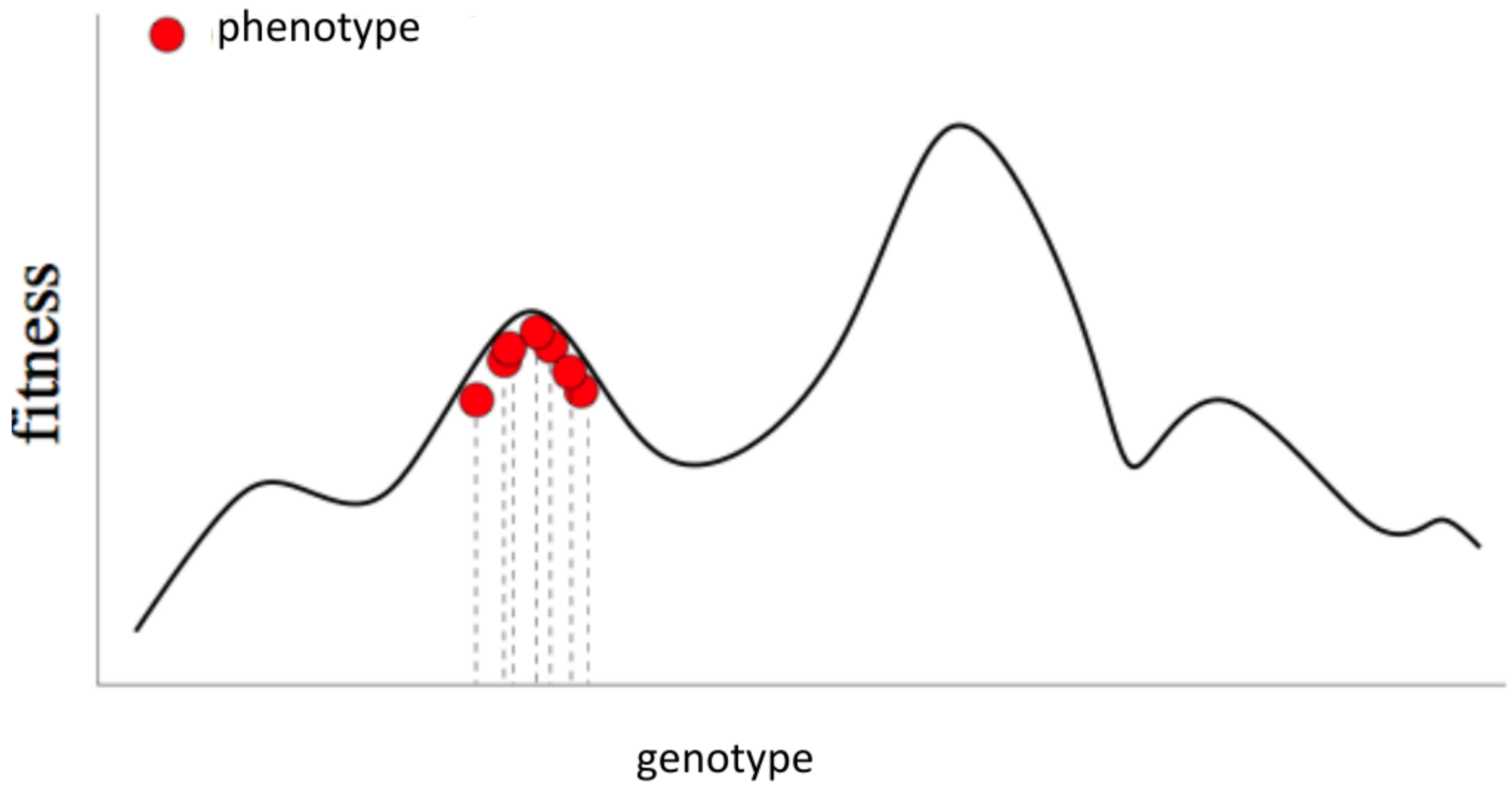
Example Behaviour in GA

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

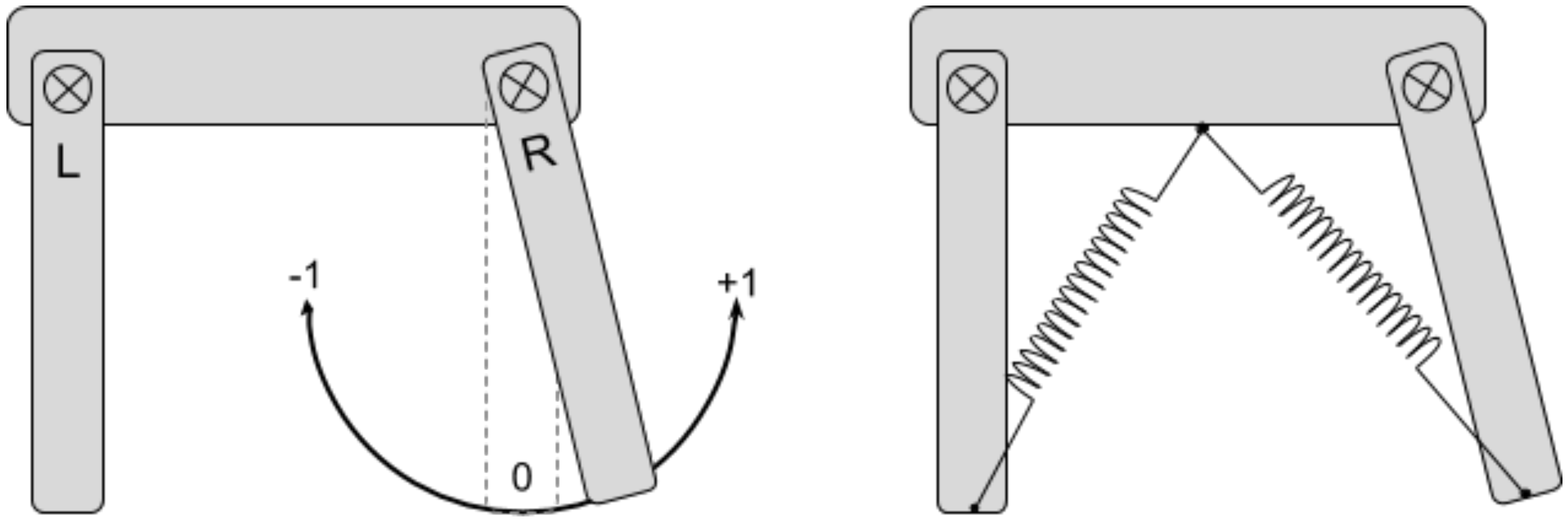
GA Examples

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*

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



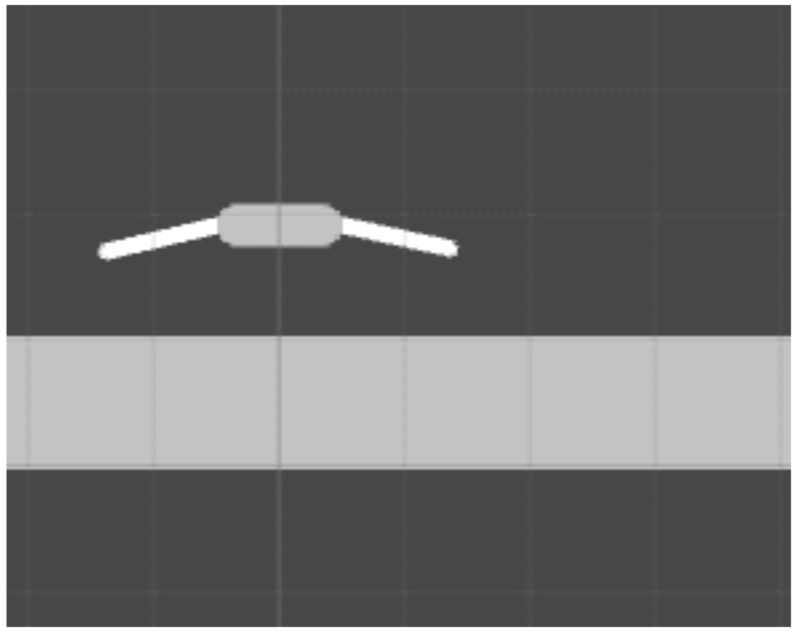
contraction: -1 extension: +1

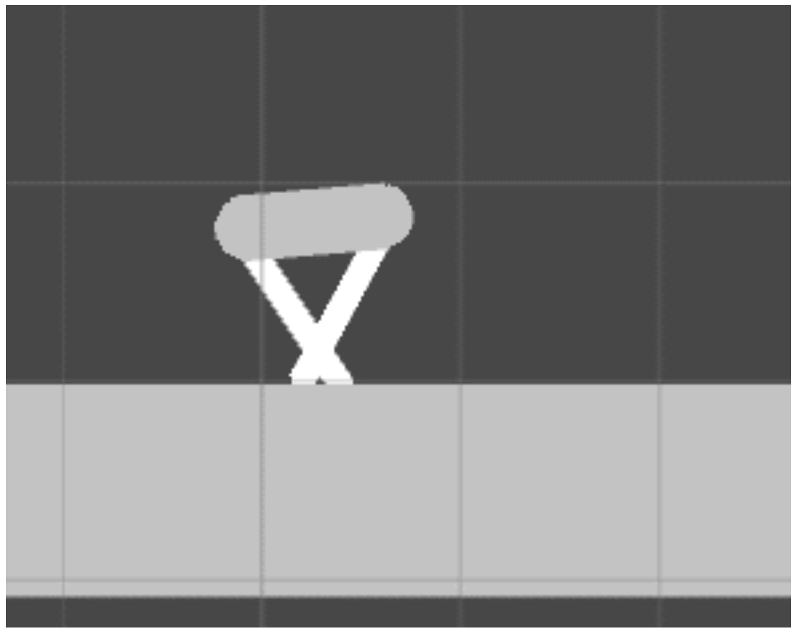
Walking Robot with GA

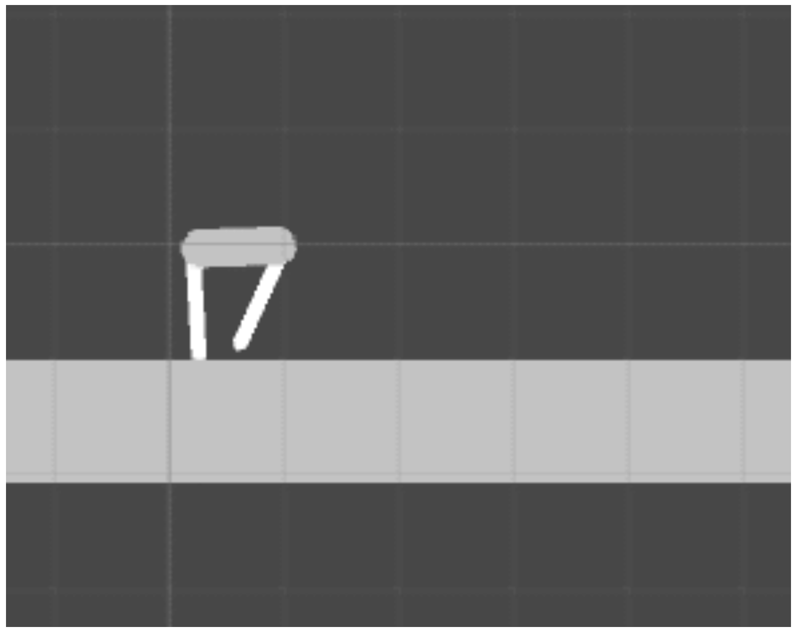
- We have two legs and four parameters per leg. So, one candidate robot (phenotype) can be encoded by the tuple (chromosome):

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

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


reward = the distance how far the robot travelled

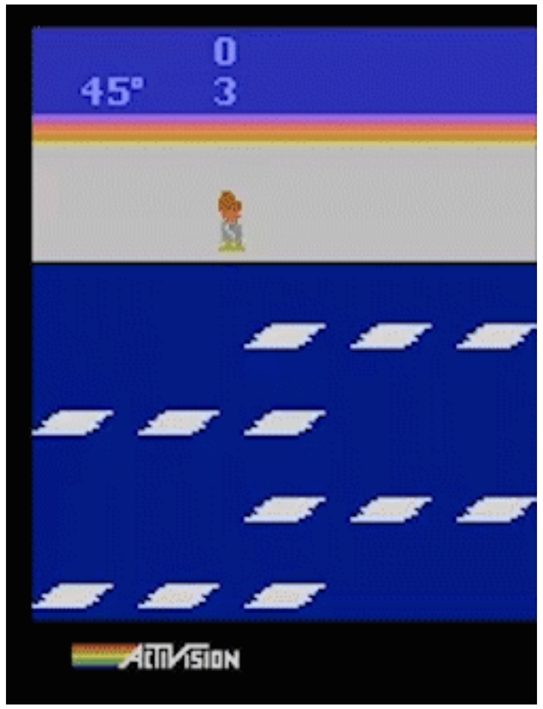

reward = stay up


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]

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

GA in State-of-the Art RL

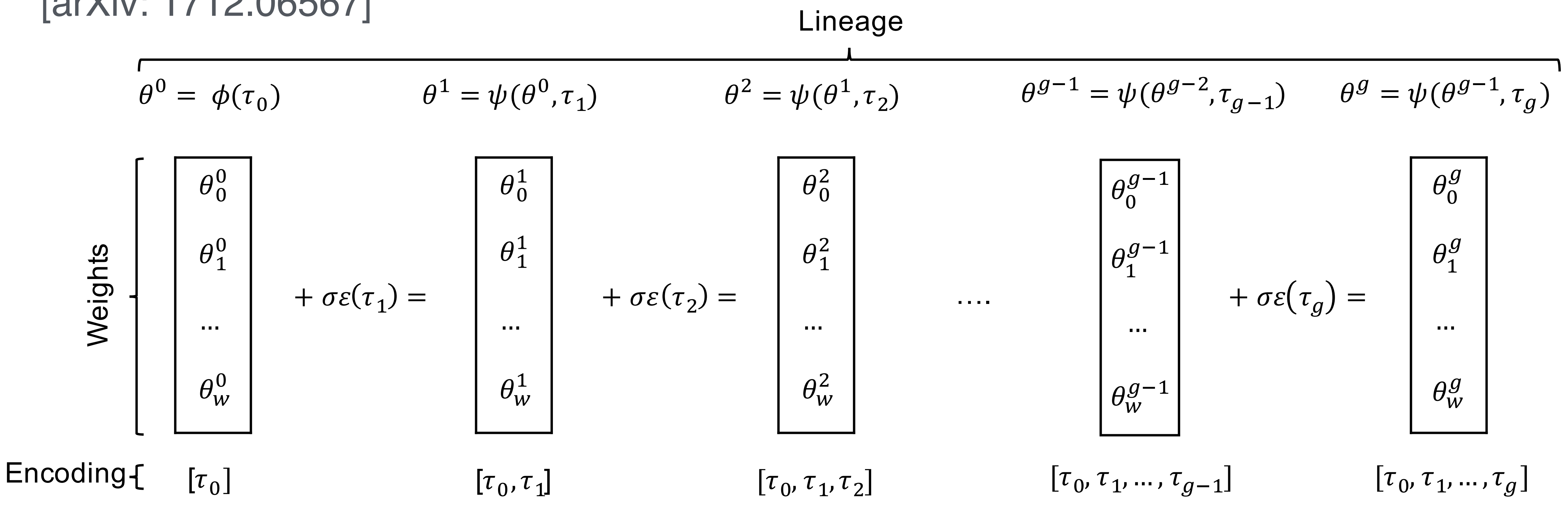
[arXiv: 1712.06567]

	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]


Lineage
$\theta^0 = \phi(\tau_0)$
$\theta^1 = \psi(\theta^0, \tau_1)$
$\theta^2 = \psi(\theta^1, \tau_2)$
$\theta^{g-1} = \psi(\theta^{g-2}, \tau_{g-1})$
$\theta^g = \psi(\theta^{g-1}, \tau_g)$
Weights
θ_0^0 θ_1^0 … θ_w^0
$+\sigma\varepsilon(\tau_1) =$
θ_0^1 θ_1^1 … θ_w^1
$+\sigma\varepsilon(\tau_2) =$
θ_0^2 θ_1^2 … θ_w^2
….
θ_0^{g-1} θ_1^{g-1} … θ_w^{g-1}
$+\sigma\varepsilon(\tau_g) =$
θ_0^g θ_1^g … θ_w^g
Encoding
$[\tau_0]$
$[\tau_0, \tau_1]$
$[\tau_0, \tau_1, \tau_2]$
$[\tau_0, \tau_1, \dots, \tau_{g-1}]$
$[\tau_0, \tau_1, \dots, \tau_g]$

- 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”

Holland’s Schema Theorem for GA

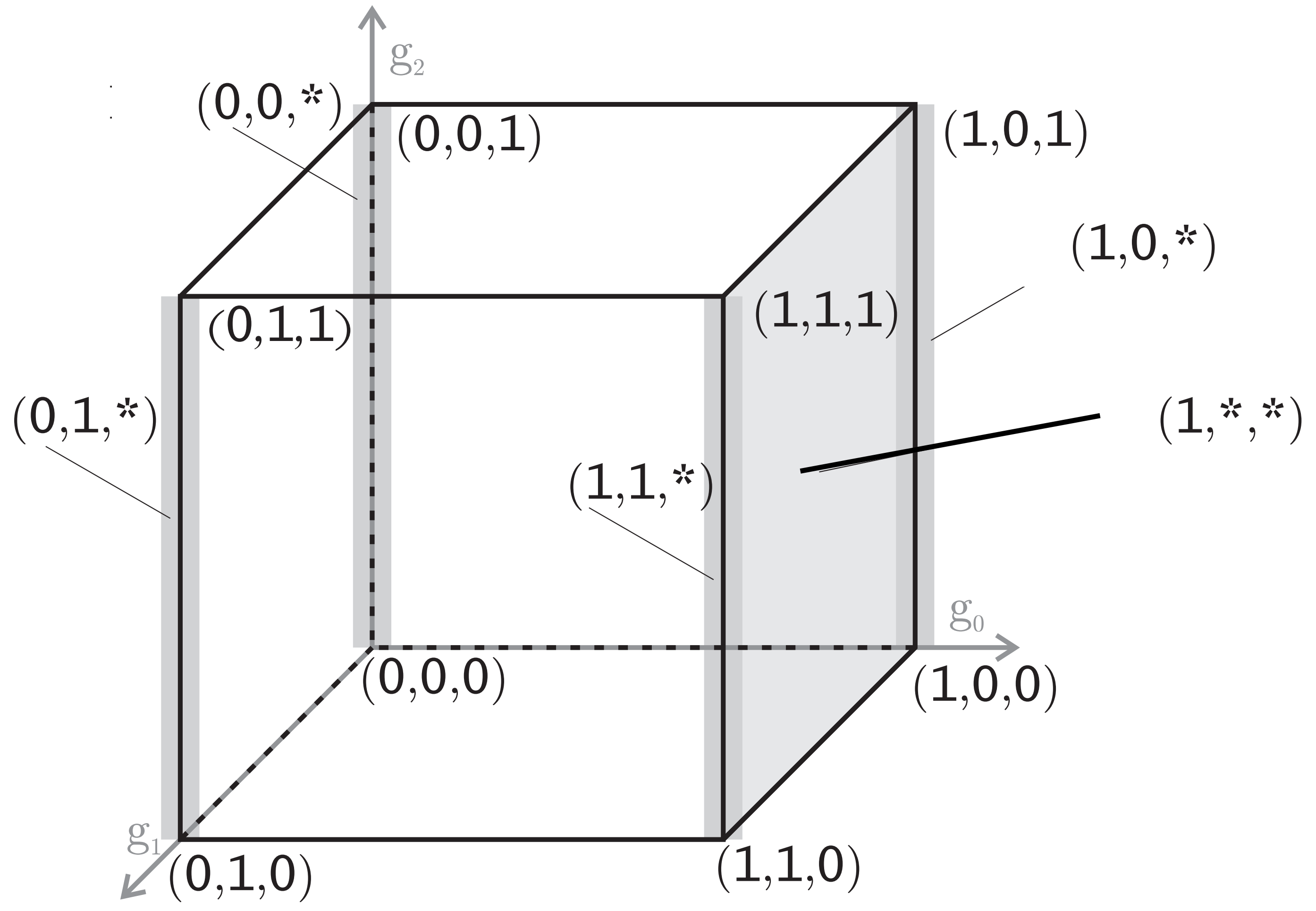
Holland's Schema Theorem

- Fundamental theorem of genetic algorithms
- Goal: provide a formal model for the effectiveness of the GA search process
- Assumptions for the GA
  - binary alphabet: 0,1
  - chromosomes of fixed length (bit strings)
  - fitness proportionate selection
  - recombination by single point crossover
  - gene-wise mutation (bit flipping)

Some details taken from
[T. M. Mitchell: Machine Learning]
[Weise: Global Optimisation Algorithm - Theory and Application]

A Schema

- A schema is a template that identifies a subset of strings with similarities at certain positions
- * is a wildcard symbol, which means that it can take either value 0 or 1 (don't care)
- Example (length 3): schema [1 * *] implies the individuals
  - [1 0 0]
  - [1 1 0]
  - [1 0 1]
  - [1 1 1]
- a schema like [* 0 *] contains less information than [0 0 *]
- there are 3^L possible schemas for bit strings of length L



[Weise: Global Optimisation Algorithm - Theory and Application]

Notation and Definitions

- l, fixed length of each individual
- P_t, population at time t of size n (containing n individuals)
- $o(s)$, the order of s, is the number of non "*" genes in schema s, e.g.

$$o([* * * 1\,1 * 0]) = 3$$

- $d(s)$, schema defining length, is the distance between first and last non "*" genes in schema s, e.g.

$$d([* * * 1\,1 * 0]) = 7 - 4 = 3$$

- $f(h)$, fitness score of some individual $h \in P_t$
- $m(s, t)$, number of instances of schema s in the population at time t

Derivation

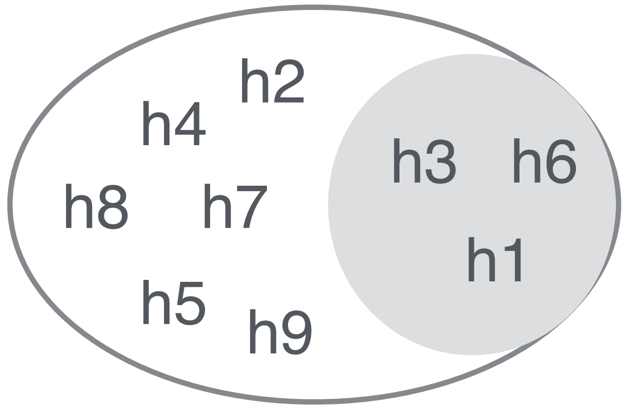
- What is the goal ? Let us consider one particular schema s. We want to know if we start with m instances of schema s in a population at time t, how many instances of schema s survived in the next generation at time $t+1$?
- What is the probability of selecting $h \in P_t$ via fitness proportional selection ? It is

$$p(h) = \frac{f(h)}{\sum_{h' \in P_t} f(h')} = \frac{f(h)}{n\,\bar{f}(t)}\,, \tag{1}$$

  where $\bar{f}(t)$ is the mean fitness of the entire population at time t.
- Suppose we have selected some arbitrary $h \in P_t$ according to (1), what is the probability that h is an instance of some schema s ? It is

$$p(h \in s) = \sum_{h \in s \cap P_t} p(h) = \frac{\sum_{h \in s \cap P_t} f(h)}{n\,\bar{f}(t)} = \frac{\bar{f}(s,t)}{\bar{f}(t)} \frac{m(s,t)}{n}\,,$$

  where $\bar{f}(s,t)$ is the average fitness of schema s at time t, and $m(s,t)$ is the number of instances of schema s in the population at time t.



Derivation

- Actually, we are interested in the expected number of instances of schema s resulting from n independent selection steps (with replacement) that create the entire generation at time step $t + 1$:

$$\mathbb{E}[m(s, t+1)] = n\, p(h \in s) = \frac{\bar{f}(s,t)}{\bar{f}(t)}\, m(s,t)\,.$$

  Thus, we can expect schemas with above average fitness to be represented with increasing frequency in subsequent generations.

- But wait, what about crossover and mutations ?

Crossover

- Let us denote with p_c the probability that the single-point crossover operator will be applied to an arbitrary individual.
- Note, that even if the single-point crossover is applied on a parent belonging to schema s it does not necessarily mean that the child is not anymore part of the same schema s, e.g.

$$\begin{array}{rcl} s = [\; * \;\; 1 \;\; 0 \;\vdots\; * \;\; * \;] & & [\; * \;\; 1 \;\; 0 \;\; 1 \;\; 1 \;] \\ & \longrightarrow & \\ [\; 0 \;\; 0 \;\; 0 \;\vdots\; 1 \;\; 1 \;] & & [\; 0 \;\; 0 \;\; 0 \;\; * \;\; * \;] \end{array}$$

- In fact, the probability that the single-point crossover operator will be applied and cuts the schema s is given by

$$p_c \, \frac{d(s)}{l-1} \, .$$

e.g. d([* 1 0 * *])/(l-1) = 1/4

Schema with long defining length are more likely to be disrupted by single point crossover than schema using short defining lengths.

- Consequently, the probability that the schema s survives the crossover step is

$$\left(1 - p_c \, \frac{d(s)}{l-1}\right) \, .$$

Mutation

- Mutation is applied gene by gene.
- Let us denote with p_m the probability that an arbitrary bit of an arbitrary individual will be mutated by the mutation operator.
- In order for schema s to survive, all non "*" genes in the schema must remain unchanged
- When the "*" positions of a schema instance are mutated, the instance still belongs to the same schema s
- Thus, the probability that a given instance of schema s will still belong to schema s after the mutation step is given by

$$(1 - p_m)^{o(s)}$$

Holland's Schema Theorem

- Finally, the expected number of instances of schema s resulting from n independent selection steps (with replacement) that create the entire generation at time step $t+1$ is given by

$$\mathbb{E}[m(s,t+1)] \geq \underbrace{\frac{\bar{f}(s,t)}{\bar{f}(t)} \left(1 - p_c \frac{d(s)}{l-1}\right) (1-p_m)^{o(s)}}_{\alpha(s,t)} \; m(s,t)\,. \qquad (1)$$

  This lower bound means that a schema s with $\alpha(s,t) > 1$ will on average increase its instances in the next generation.

- how to achieve $\alpha(s,t) > 1$?
  - s must have above average fitness
  - defined bits in s should be grouped, i.e. small schema-defining length $d(s)$
  - s should contain a large number of "*" symbols, i.e. small order $o(s)$
- "The Schema Theorem says that short, low-order schemata with above-average fitness increase in frequency in successive generations"
- Note, (1) neglects the probability that a string belonging to the schema s will be created "from scratch" by mutation of a single string (or crossover of two strings) that did not belong to s in the previous generation. Therefore, (1) is an inequality.

Holland’s Schema Theorem

Limitations

- The theorem fails to consider the positive effects of crossover and mutation
- The theorem makes no statement about that converging to a global optimum solution
- The theorem is described in terms of expectation, thus it is only valid if you perform an infinite number of “experiments” or if you let the population size increase to infinity. Otherwise the theorem is affected by sampling errors.
- The theorem makes only a statement about one generation step. Note, that the average-fitness of the population and of the schema is time-dependent, i.e. their ratio is not static (population drift). Consequently, the theorem cannot be generalised to multiple time steps.

[some interesting read on the crossover-mutation debate:
https://www-cs.stanford.edu/people/nuwans/docs/GA.pdf]

Evolution Strategies

Evolution Strategies (ES)

- Note: this comparison is historical, nowadays properties can overlap for both types

Properties	ES	GA
objective parameters "genes"	real values	binary values
mutation	Gaussian noise	bit-flip
parent selection	uniform random	various methods
recombination	averaging	crossover
what is evolved	objective parameters and mutation parameters	objective parameters

- Key properties of Evolution Strategies
  - recombination is done by averaging the parameters of parents
  - mutation parameters can evolve (self-adapt their values)
  - mutation is done by adding normally distributed values

Self-Adaptation (μ/ρ+,λ) ES Algorithm

- Each candidate **a** now contains objective (like in GA) and mutation (strategy) parameters (d is the dimensionality of the search space):

$$\mathbf{a} = (\underbrace{\theta_1, \theta_2, ..., \theta_d}_{\text{objective parameters } \boldsymbol{\theta}} , \underbrace{\sigma_1, \sigma_2, ..., \sigma_d}_{\text{mutation parameters } \boldsymbol{\sigma}})$$

  The mutation parameter determines the amount of mutation which is applied to the corresponding objective parameter.

- the meta-parameters of ES are described by the $(\mu/\rho +, \lambda)$ notation
  - μ is the number of parents in the parent population $P_\mu = \{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_\mu\}$
  - ρ is the number of parents that are averaged to create one offspring
  - λ is the number of offspring in the offspring population $P'_\lambda = \{\mathbf{a}'_1, \mathbf{a}'_2, ..., \mathbf{a}'_\lambda\}$
  - ",": select new parents for next generation from offspring (requires $\lambda \geq \mu$)
  - " + ": select new parents for next generation from offspring and previous parents

- simple examples: population with one individual
  - $(1/1, 1)$: next generation always contains the offspring from the parent
  - $(1/1+1)$: next generation contains the more fit candidate (offspring or parent)

Self-Adaptation (μ/ρ+,λ) ES Algorithm

Algorithm 1: $(\mu/\rho \overset{+}{,} \lambda)$ Self-Adaptation Evolution Strategy Algorithm

```
1  initialise the first generation $P_\mu^{g=1} = \{\mathbf{a}_1^{g=1}, \mathbf{a}_2^{g=1}, ..., \mathbf{a}_\mu^{g=1}\}$ and $P_\lambda = \{\}$
2  for each generation $g = 2, 3, ..., G$ do
3      for each offspring $(1, 2, ..., \lambda)$ do
4          select uniform randomly $\rho$ parents from $P_\mu^{g-1}$
5          average the selected parents to form the candidate $\mathbf{a} = (\boldsymbol{\theta}, \boldsymbol{\sigma})$
6          adapt the mutation parameter $\boldsymbol{\sigma}$ yielding the new value $\boldsymbol{\sigma}'$
7          mutate the objective parameter $\boldsymbol{\theta}$ using $\boldsymbol{\sigma}'$ yielding the new value $\boldsymbol{\theta}'$
8          add new offspring $(\boldsymbol{\theta}', \boldsymbol{\sigma}')$ to offspring population $P_\lambda$
9      end
10     select $\mu$ candidates for next generation $P_\mu^g$ via truncated selection from either
11             - the offspring population $P_\lambda$ (“$\mu/\rho, \lambda$” comma selection)
12             - the offsprings $P_\lambda$ and parents $P_\mu^{g-1}$ (“$\mu/\rho + \lambda$” plus selection)
13     reset offspring population $P_\lambda = \{\}$
14 end
```

Uncorrelated Mutation: Case 1

Uncorrelated mutation with 1 mutation parameter

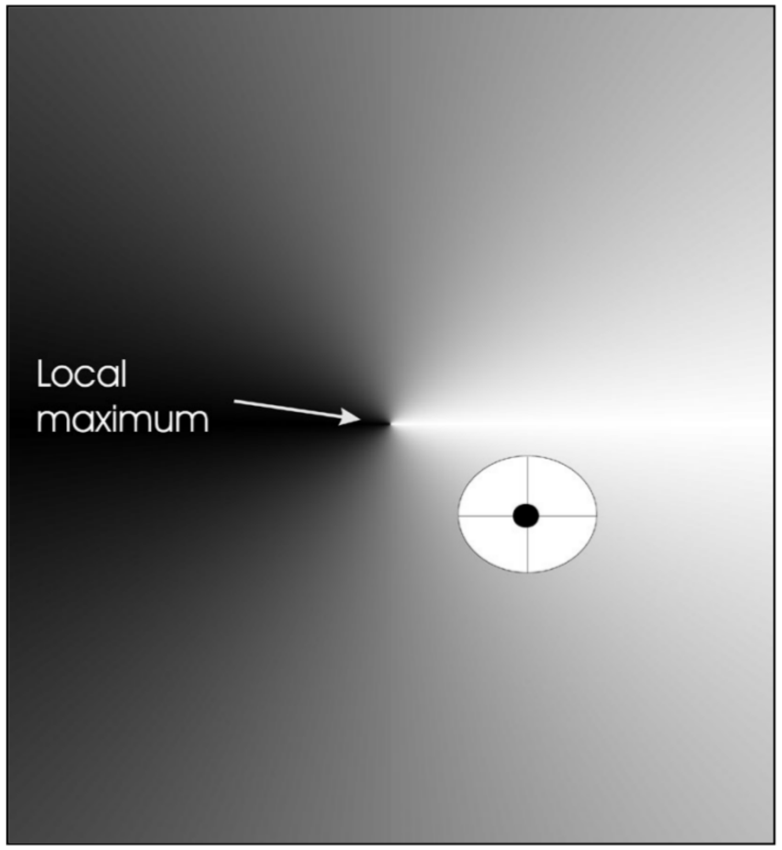
- look at one mutation step of the (parent average) candidate $(\boldsymbol{\theta}, \sigma)$

$$\sigma' = \sigma \exp(\epsilon)\,, \qquad \epsilon \sim N(0, \tau)$$
$$\theta'_i = \theta_i + \epsilon_i\,, \qquad \epsilon_i \sim N(0, \sigma'),\ i = 1, ..., d$$

- τ is the learning rate, typically $\tau \sim 1/d^{1/2}$
- we impose the boundary rule that if $\sigma' < \epsilon$ then $\sigma' = \epsilon$, where ϵ is some fixed threshold
- changing the mutation strength σ allows for a self-tuning of the mutation strength (σ self-adaptation)

isotropically distributed mutations

- mutants on the circle have the same probability of being created from the parent in the centre



[Eiben, Smith: Introduction to Evolutionary Computation]

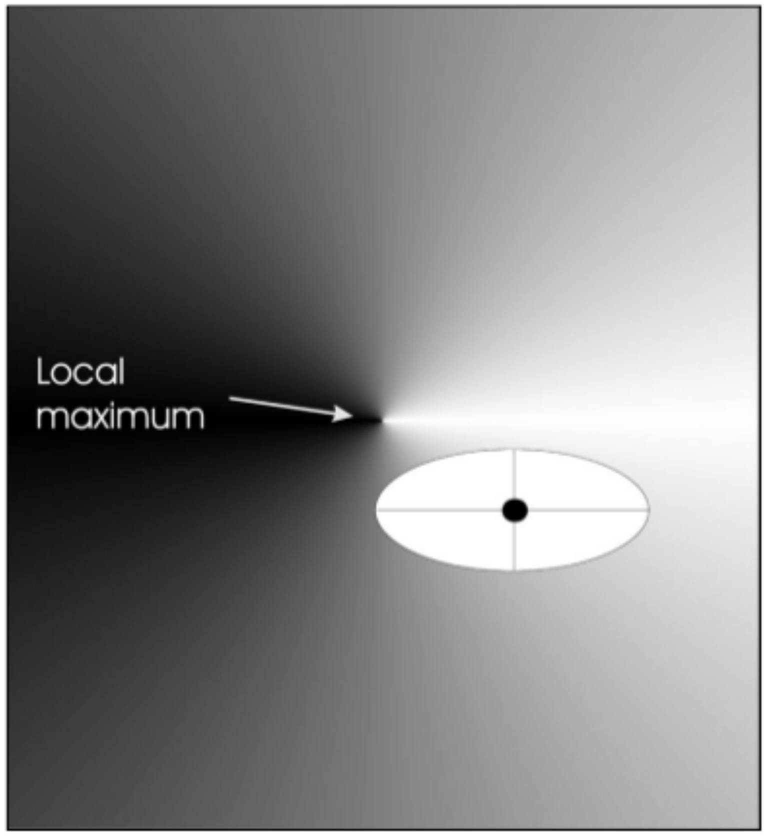
Uncorrelated Mutation: Case 2

Uncorrelated mutation with d mutation parameters

- look at one mutation step of the (parent average) candidate $(\boldsymbol{\theta}, \boldsymbol{\sigma})$

$$\begin{aligned} \sigma_i' &= \sigma_i \exp(\epsilon + \epsilon_i'), \qquad && \epsilon \sim N(0,\tau), \epsilon_i' \sim N(0,\tau'), \\ \theta_i' &= \theta_i + \epsilon_i, && \epsilon_i \sim N(0,\sigma_i'), \; i = 1, ..., d \end{aligned}$$

- two learning rates
  - overall learning rate $\tau \sim 1/d^{1/2}$
  - coordinate-wise learning rate $\tau' \sim 1/d^{1/4}$
- we impose the boundary rule that if $\sigma' < \epsilon$ then $\sigma' = \epsilon$, where ϵ is some fixed threshold
- probability of mutation varies along the coordinates of the $\boldsymbol{\theta}$ space



mutants on the ellipse have the same probability of being created from the parent in the centre

[Eiben, Smith: Introduction to Evolutionary Computation]

Correlated Mutation

Correlated mutation with a $d \times d$ covariance matrix $\mathbf{C}$

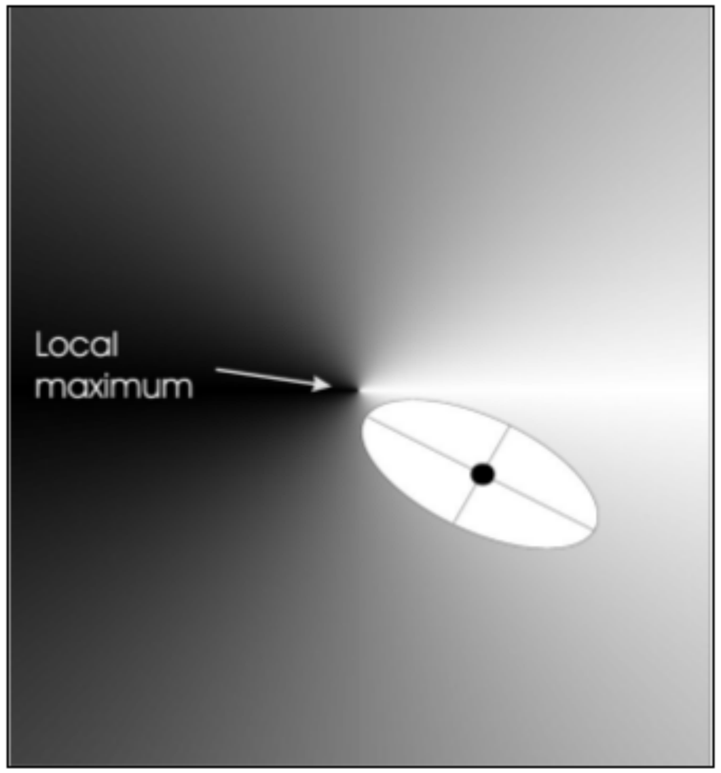
- $\mathbf{C}$ is a real, symmetric, positive semi-definite matrix: $\mathbf{C} = \mathbf{R}\,\mathbf{D}\,\mathbf{R}^T$ with diagonal positive semi-definite matrix $\mathbf{D}$ and orthogonal matrix $\mathbf{R}$
- $\mathbf{C}$ can be parametrised by d standard deviations $\boldsymbol{\sigma} = (\sigma_1, ..., \sigma_d)$ and $k = d(d-1)/2$ rotation angles $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_k)$
- look at one mutation step for the candidate $(\boldsymbol{\theta}, \boldsymbol{\sigma}, \boldsymbol{\alpha})$

$$\sigma_i' = \sigma_i \exp(\epsilon + \epsilon_i')\,, \qquad \epsilon \sim N(0,\tau),\ \epsilon_i' \sim N(0,\tau')$$
$$\alpha_i' = \alpha_i + \epsilon_i\,, \qquad \epsilon_i \sim N(0,\tau''),\ i = 1, ..., d$$

- we impose the same boundary rule as before
- two learning rates as before (τ and τ') and additionally τ''
- create $\mathbf{C}'$ from $\boldsymbol{\sigma}', \boldsymbol{\alpha}'$ and mutate objective parameters

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + \boldsymbol{\epsilon}\,, \qquad \boldsymbol{\epsilon} \sim N(0, \mathbf{C}')$$

- probability of mutation (variance) can vary along any direction in the $\boldsymbol{\theta}$ space



[Eiben, Smith: Introduction to Evolutionary Computation]

ES Examples

Covariance Matrix Adaptation (CMA) - ES

- one of the most popular gradient-free optimisation algorithms
- useful in particular on non-convex, non-separable, ill-conditioned, multi-modal or noisy objective functions
- considers the **mean** and **covariance** of the current candidates in the population and adapts them in direction towards the fittest candidates for the next generation
- belongs to the class of **(μ/μ_w, λ) ES** algorithms where all parents are averaged (weighted) before computing offsprings by mutation
- can be used when the search space dimensionality is less than ~1000 due to O(d^2) behaviour

[https://arxiv.org/abs/1604.00772]

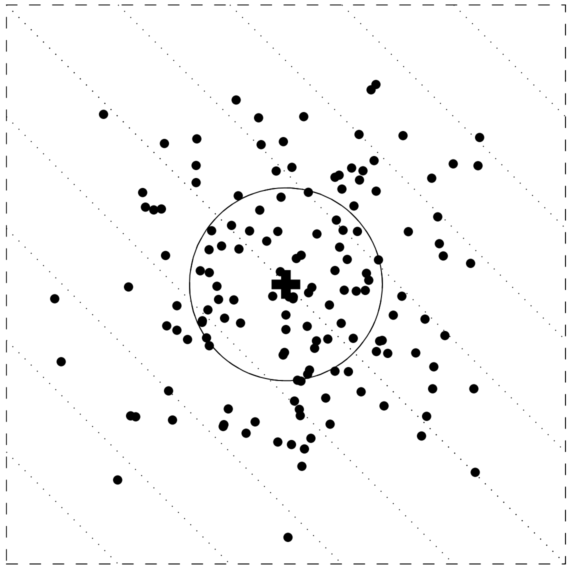
CMA-ES

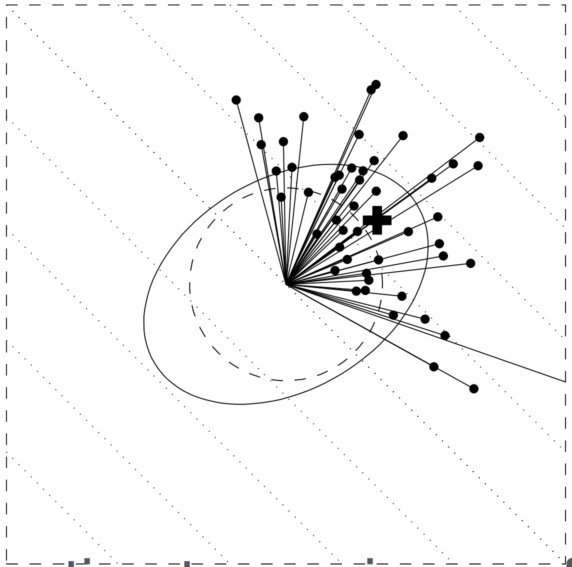
Algorithm 1: Basic idea behind CMA-ES, see [arXiv:1604.00772] for details

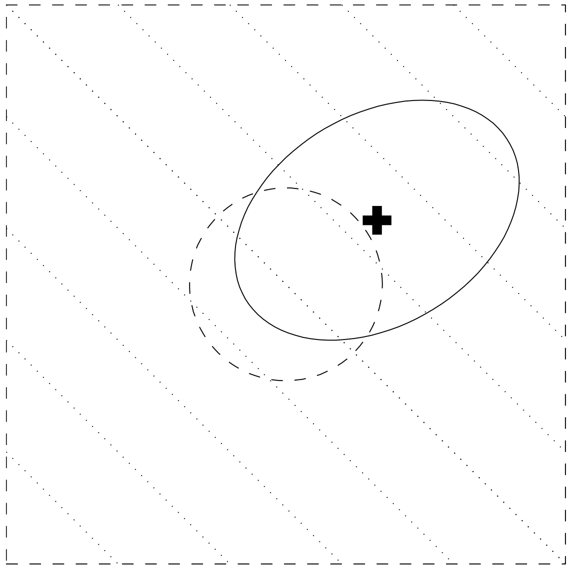
```
1 initialise mean $\mathbf{m}^{(0)}$ and covariance matrix $\mathbf{C}^{(0)} = \sigma\,\mathbf{1}$
2 for each generation $g = 1, 2, ..., G$ do
3     sample $\lambda$ offsprings
          $$\boldsymbol{\theta}_i^{(g)} \sim \mathbf{m}^{(g-1)} + N(0, \mathbf{C}^{(g-1)}) \quad \text{for} \quad i = 1, 2, ..., \lambda$$
4     select parents via truncated selection and average them
          $$\mathbf{m}^{(g)} = \frac{1}{\mu}\sum_{i=1}^{\mu} \boldsymbol{\theta}_{i:\lambda}^{(g)},$$
      where $\boldsymbol{\theta}_{i:\lambda}^{(g)}$ denotes the $i^{\text{th}}$-fittest offspring of generation $g$.
5     adapt the covariance matrix
          $$\mathbf{C}^{(g)} = \frac{1}{\mu}\sum_{i=1}^{\mu}(\boldsymbol{\theta}_{i:\lambda}^{(g)} - \mathbf{m}^{(g-1)})\,(\boldsymbol{\theta}_{i:\lambda}^{(g)} - \mathbf{m}^{(g-1)})^T$$
6 end
```

use true mean !!!

How covariance adaptation works [https://arxiv.org/abs/1604.00772]

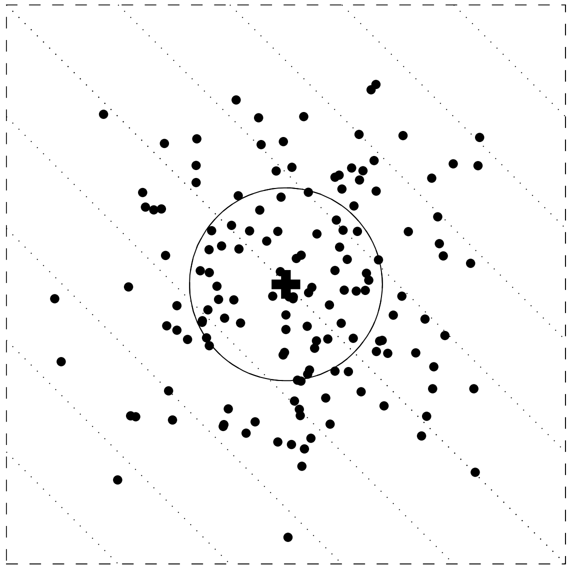

sampling

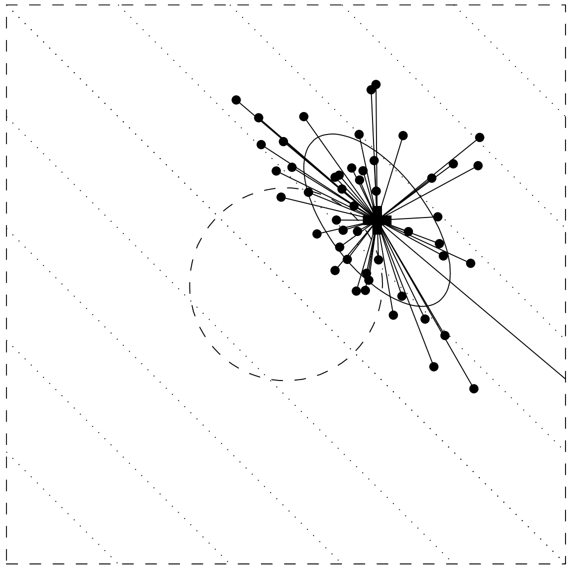

estimates variance of the selected steps

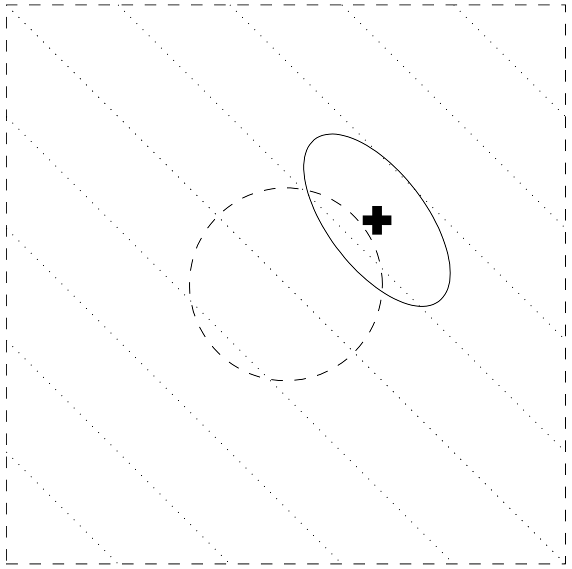

new covariance matrix

$\boldsymbol{C}_{\mu}^{(g+1)}$

(used)


sampling

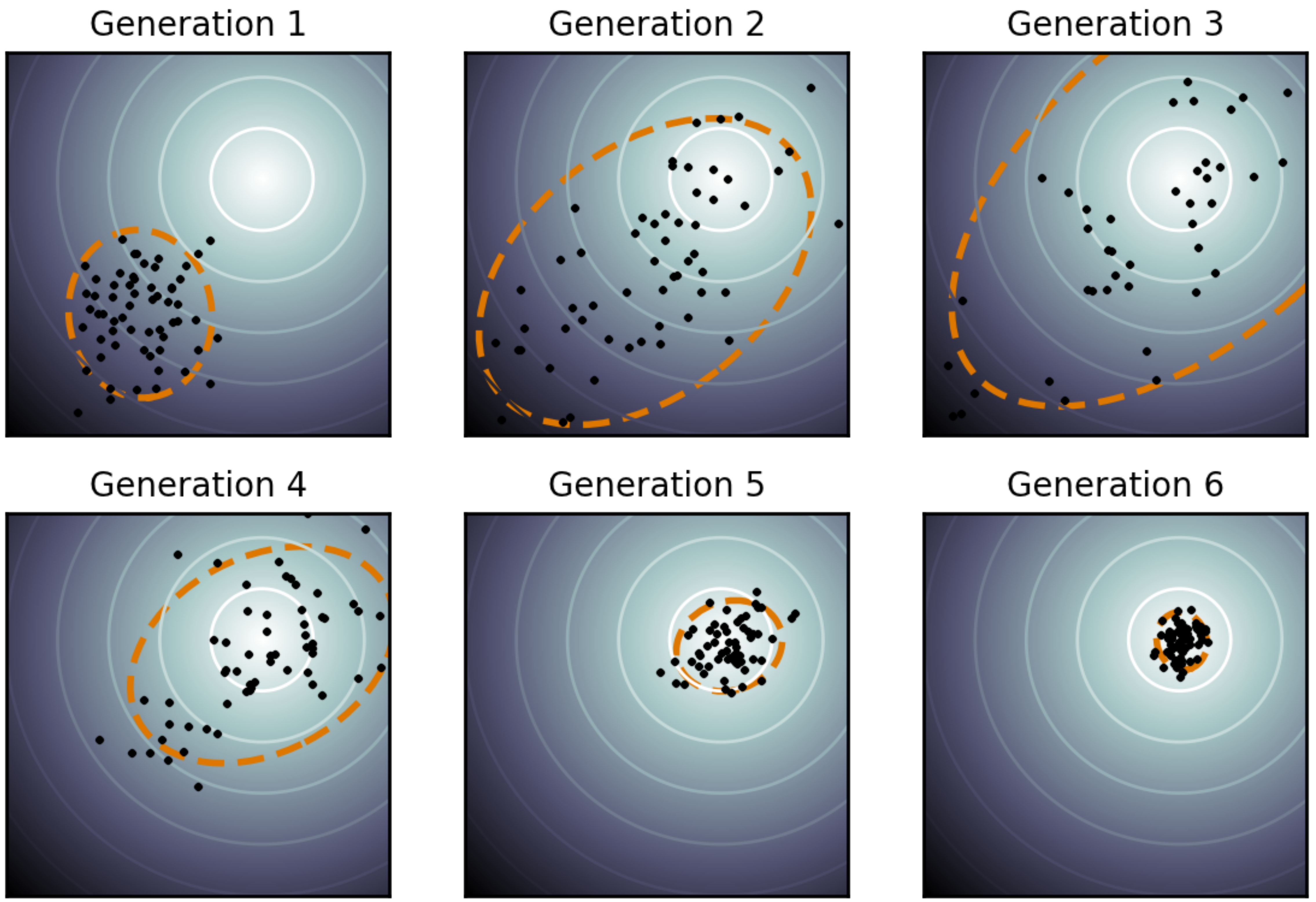

estimates variance within the selected population


new covariance matrix

$C_{\mathrm{EMNA}_{global}}^{(g+1)}$

(not used)

CMA-ES



[https://en.wikipedia.org/wiki/CMA-ES]

ES with a Search Distribution

- So far, we have always discarded some fraction of the offspring, e.g. through truncated selection
- Now, we would like to use the information from all offspring, including those with smaller fitness
- We introduce a search distribution whose parameters θ are updated, e.g. a Gaussian distribution with mean and variance
- The new "objective function" $J(\theta)$ is now the expected fitness under this search distribution $\pi(z|\theta)$

$$J(\theta) = \mathbb{E}_{z\sim\pi(z|\theta)}[f(z)] = \int f(z)\,\pi(z|\theta)\,dz\,,$$

  where $f(z)$ is the original fitness function.
- Now, the core idea is to use search gradients to update the parameters θ of the search distribution $\pi(z|\theta)$

ES with a Search Distribution

If we use θ to denote the parameters of density $\pi(\mathbf{z} \,|\, \theta)$ and $f(\mathbf{z})$ to denote the fitness function for samples $\mathbf{z}$, we can write the expected fitness under the search distribution as

$$J(\theta) = \mathbb{E}_\theta[f(\mathbf{z})] = \int f(\mathbf{z})\, \pi(\mathbf{z} \,|\, \theta)\, d\mathbf{z}. \tag{1}$$

The so-called 'log-likelihood trick' enables us to write

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \int f(\mathbf{z})\, \pi(\mathbf{z} \,|\, \theta)\, d\mathbf{z} \\
&= \int f(\mathbf{z})\, \nabla_\theta \pi(\mathbf{z} \,|\, \theta)\, d\mathbf{z} \\
&= \int f(\mathbf{z})\, \nabla_\theta \pi(\mathbf{z} \,|\, \theta)\, \frac{\pi(\mathbf{z} \,|\, \theta)}{\pi(\mathbf{z} \,|\, \theta)}\, d\mathbf{z} \\
&= \int \Big[f(\mathbf{z})\, \nabla_\theta \log \pi(\mathbf{z} \,|\, \theta) \Big]\, \pi(\mathbf{z} \,|\, \theta)\, d\mathbf{z} \\
&= \mathbb{E}_\theta \left[f(\mathbf{z})\, \nabla_\theta \log \pi(\mathbf{z} \,|\, \theta) \right].
\end{aligned}$$

From this form we obtain the estimate of the search gradient from samples $\mathbf{z}_1 \ldots \mathbf{z}_\lambda$ as

$$\nabla_\theta J(\theta) \approx \frac{1}{\lambda} \sum_{k=1}^{\lambda} f(\mathbf{z}_k)\, \nabla_\theta \log \pi(\mathbf{z}_k \,|\, \theta), \tag{2}$$

where λ is the population size. This gradient on expected fitness provides a search direction in the space of search distributions.

[http://www.jmlr.org/papers/volume15/wierstra14a/wierstra14a.pdf]

ES with a Search Distribution

Algorithm 1: Canonical Search Gradient algorithm

```
input: $f$, $\theta_{init}$
repeat
    for $k = 1 \ldots \lambda$ do
        draw sample $\mathbf{z}_k \sim \pi(\cdot|\theta)$
        evaluate the fitness $f(\mathbf{z}_k)$
        calculate log-derivatives $\nabla_\theta \log \pi(\mathbf{z}_k|\theta)$
    end
    $\nabla_\theta J \leftarrow \frac{1}{\lambda} \sum_{k=1}^{\lambda} \nabla_\theta \log \pi(\mathbf{z}_k|\theta) \cdot f(\mathbf{z}_k)$
    $\theta \leftarrow \theta + \eta \cdot \nabla_\theta J$
until stopping criterion is met
```

[http://www.jmlr.org/papers/volume15/wierstra14a/wierstra14a.pdf]

ES with a Search Distribution

Algorithm 2: Search Gradient algorithm: Multinormal distribution

```
input: $f$, $\boldsymbol{\mu}_{init}$,$\boldsymbol{\Sigma}_{init}$
repeat
    for $k = 1 \ldots \lambda$ do
        draw sample $\mathbf{z}_k \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$
        evaluate the fitness $f(\mathbf{z}_k)$
        calculate log-derivatives:
          $\nabla_{\boldsymbol{\mu}} \log \pi(\mathbf{z}_k|\theta) = \boldsymbol{\Sigma}^{-1}(\mathbf{z}_k - \boldsymbol{\mu})$
          $\nabla_{\boldsymbol{\Sigma}} \log \pi(\mathbf{z}_k|\theta) = -\frac{1}{2}\boldsymbol{\Sigma}^{-1} + \frac{1}{2}\boldsymbol{\Sigma}^{-1}(\mathbf{z}_k - \boldsymbol{\mu})(\mathbf{z}_k - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}$
    end
    $\nabla_{\boldsymbol{\mu}} J \leftarrow \frac{1}{\lambda}\sum_{k=1}^{\lambda} \nabla_{\boldsymbol{\mu}} \log \pi(\mathbf{z}_k|\theta) \cdot f(\mathbf{z}_k)$
    $\nabla_{\boldsymbol{\Sigma}} J \leftarrow \frac{1}{\lambda}\sum_{k=1}^{\lambda} \nabla_{\boldsymbol{\Sigma}} \log \pi(\mathbf{z}_k|\theta) \cdot f(\mathbf{z}_k)$
    $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} + \eta \cdot \nabla_{\boldsymbol{\mu}} J$
    $\boldsymbol{\Sigma} \leftarrow \boldsymbol{\Sigma} + \eta \cdot \nabla_{\boldsymbol{\Sigma}} J$
until stopping criterion is met
```

[http://www.jmlr.org/papers/volume15/wierstra14a/wierstra14a.pdf]

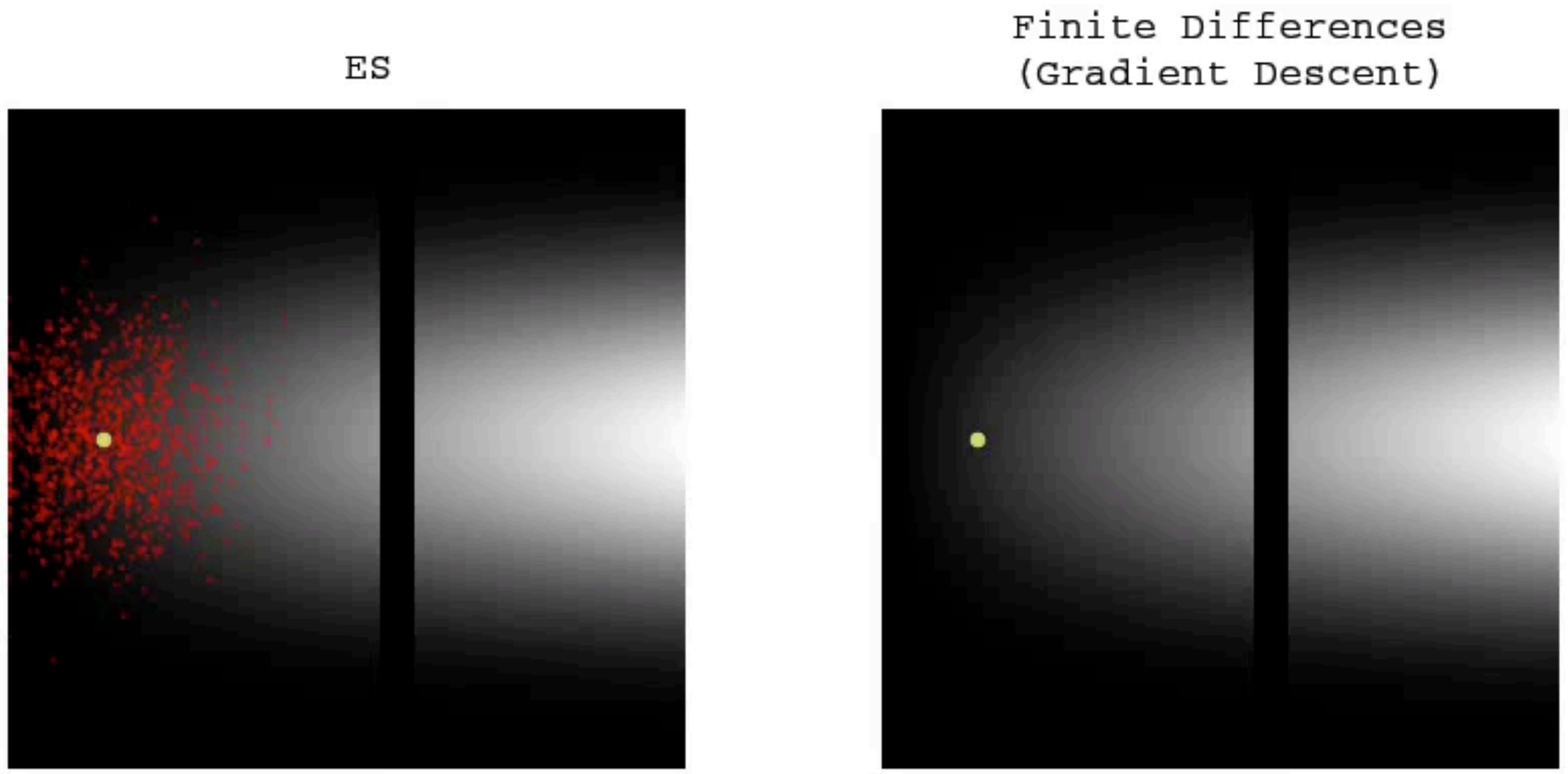
Tradeoff ES and SGD

Note

- ES optimises reward of a population of policies described by a probability distribution (a cloud in the search space)
- SGD optimises reward for a single policy (a point in the search space)

Image

- traditional finite differences (gradient descent) cannot cross a narrow gap of low fitness while ES easily crosses it to find higher fitness on the other side

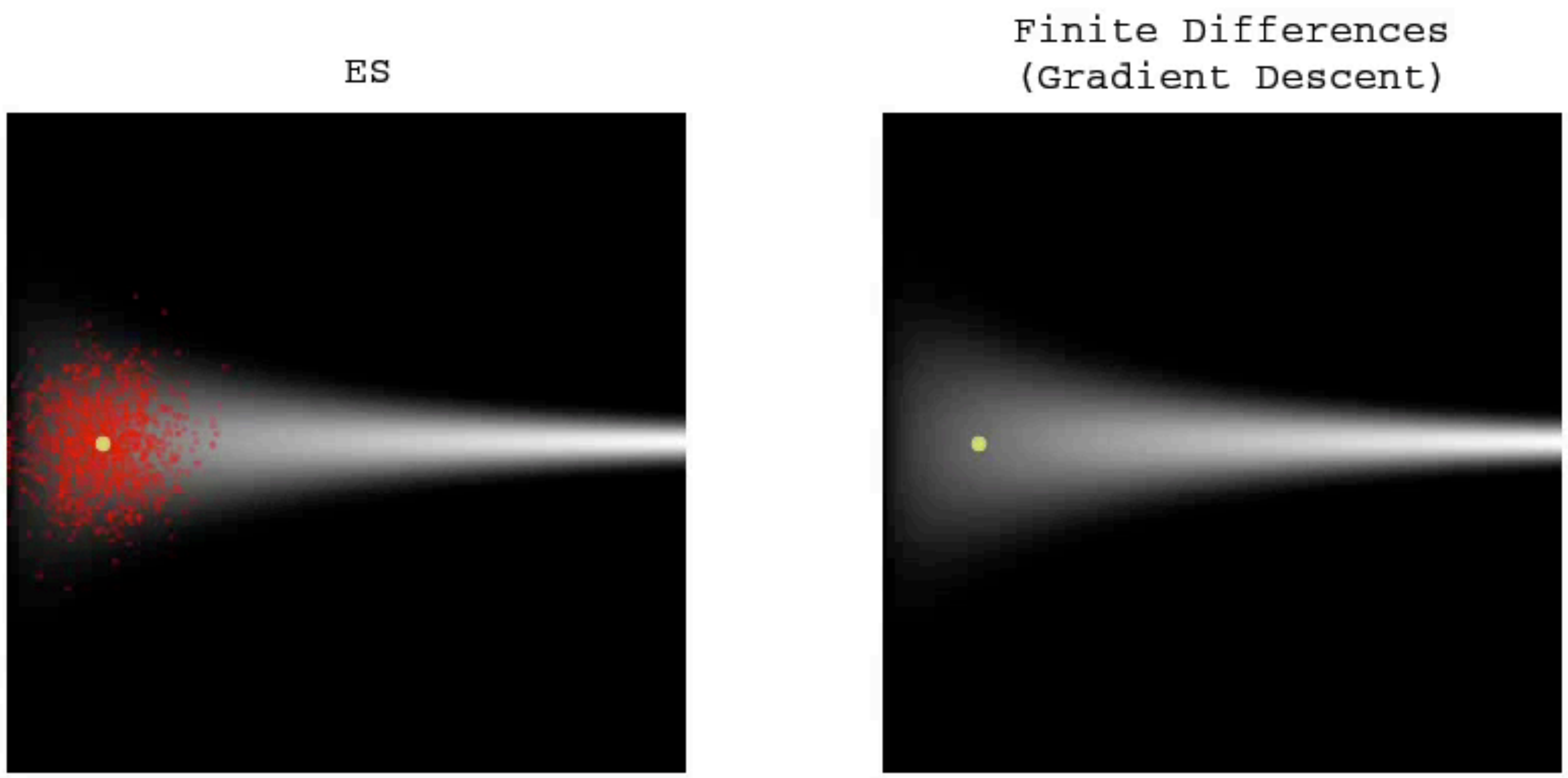


[https://eng.uber.com/deep-neuroevolution/]

Tradeoff ES and SGD

Image

- ES stalls as a path of high fitness narrows, while traditional finite differences (gradient descent) traverses the same path with no problem, illustrating along with the previous video the distinction and trade-offs between the two different approaches.



[https://eng.uber.com/deep-neuroevolution/]

ES in State-of-the Art RL

[arXiv: 1703.03864]

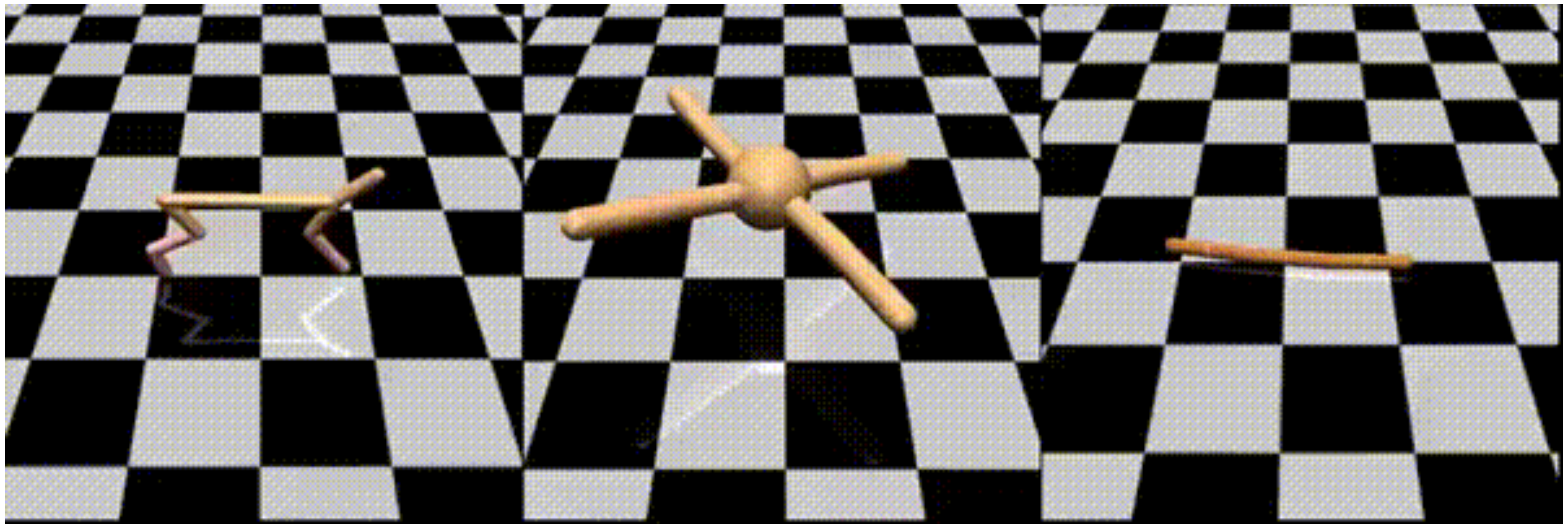
- shows that ES can be competitive with Q-learning/PG methods in RL in certain environments
- parameter space has roughly 1 million dimensions (policy network weights)
- make use of massive parallelisation

Algorithm 2 Parallelized Evolution Strategies

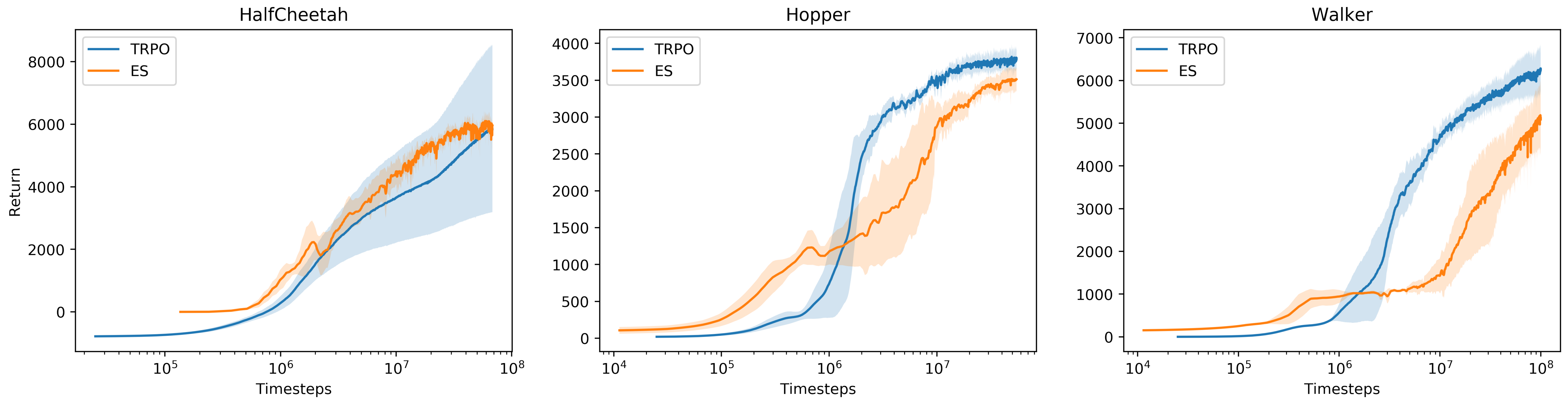
```
 1: **Input:** Learning rate $\alpha$, noise standard deviation $\sigma$, initial policy parameters $\theta_0$
 2: **Initialize:** $n$ workers with known random seeds, and initial parameters $\theta_0$
 3: for $t = 0, 1, 2, \ldots$ do
 4:     for each worker $i = 1, \ldots, n$ do
 5:         Sample $\epsilon_i \sim \mathcal{N}(0, I)$
 6:         Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$
 7:     end for
 8:     Send all scalar returns $F_i$ from each worker to every other worker
 9:     for each worker $i = 1, \ldots, n$ do
10:         Reconstruct all perturbations $\epsilon_j$ for $j = 1, \ldots, n$ using known random seeds
11:         Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{j=1}^{n} F_j \epsilon_j$
12:     end for
13: end for
```

ES in State-of-the Art RL

MuJoCo tasks with objective to move forward



ES (orange) can reach a comparable performance to TRPO (blue)



ES in State-of-the Art RL

Smoothing in parameter space versus smoothing in action space

can only be accessed via sampling. Explicitly, suppose we wish to solve general decision problems that give a return $R(\mathbf{a})$ after we take a sequence of actions $\mathbf{a} = \{a_1, \ldots, a_T\}$, where the actions are determined by a either a deterministic or a stochastic policy function $a_t = \pi(s; \theta)$. The objective we would like to optimize is thus

$$F(\theta) = R(\mathbf{a}(\theta)).$$

Since the actions are allowed to be discrete and the policy is allowed to be deterministic, $F(\theta)$ can be non-smooth in θ. More importantly, because we do not have explicit access to the underlying state transition function of our decision problems, the gradients cannot be computed with a backpropagation-like algorithm. This means we cannot directly use standard gradient-based optimization methods to find a good solution for θ.

In order to both make the problem smooth and to have a means of to estimate its gradients, we need to add noise. Policy gradient methods add the noise in action space, which is done by sampling the actions from an appropriate distribution. For example, if the actions are discrete and $\pi(s; \theta)$ calculates a score for each action before selecting the best one, then we would sample an action $\mathbf{a}(\epsilon, \theta)$ (here ϵ is the noise source) from a categorical distribution over actions at each time period, applying a softmax to the scores of each action. Doing so yields the objective $F_{PG}(\theta) = \mathbb{E}_\epsilon \, R(\mathbf{a}(\epsilon, \theta))$, with gradients

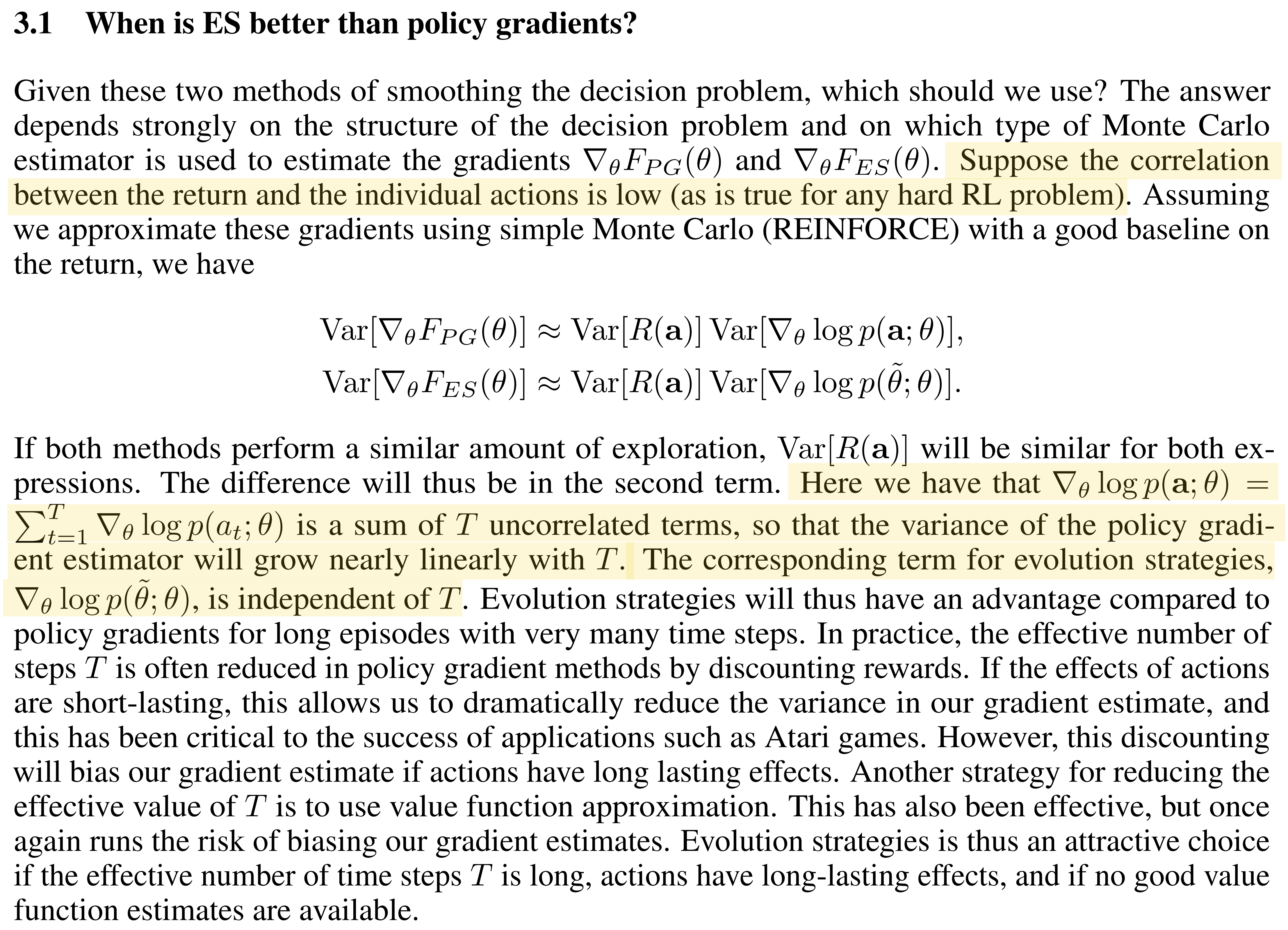
$$\nabla_\theta F_{PG}(\theta) = \mathbb{E}_\epsilon \left\{ R(\mathbf{a}(\epsilon, \theta)) \nabla_\theta \log p(\mathbf{a}(\epsilon, \theta); \theta) \right\}.$$

Evolution strategies, on the other hand, add the noise in parameter space. That is, they perturb the parameters as $\tilde{\theta} = \theta + \xi$, with ξ from a multivariate Gaussian distribution, and then pick actions as $a_t = \mathbf{a}(\xi, \theta) = \pi(s; \tilde{\theta})$. It can be interpreted as adding a Gaussian blur to the original objective, which results in a smooth, differentiable cost $F_{ES}(\theta) = \mathbb{E}_\xi \, R(\mathbf{a}(\xi, \theta))$, this time with gradients

$$\nabla_\theta F_{ES}(\theta) = \mathbb{E}_\xi \left\{ R(\mathbf{a}(\xi, \theta)) \nabla_\theta \log p(\tilde{\theta}(\xi, \theta); \theta) \right\}.$$

[arXiv: 1703.03864]

ES in State-of-the Art RL



3.1 When is ES better than policy gradients?

Given these two methods of smoothing the decision problem, which should we use? The answer depends strongly on the structure of the decision problem and on which type of Monte Carlo estimator is used to estimate the gradients $\nabla_\theta F_{PG}(\theta)$ and $\nabla_\theta F_{ES}(\theta)$. Suppose the correlation between the return and the individual actions is low (as is true for any hard RL problem). Assuming we approximate these gradients using simple Monte Carlo (REINFORCE) with a good baseline on the return, we have

$$\mathrm{Var}[\nabla_\theta F_{PG}(\theta)] \approx \mathrm{Var}[R(\mathbf{a})]\,\mathrm{Var}[\nabla_\theta \log p(\mathbf{a};\theta)],$$
$$\mathrm{Var}[\nabla_\theta F_{ES}(\theta)] \approx \mathrm{Var}[R(\mathbf{a})]\,\mathrm{Var}[\nabla_\theta \log p(\tilde{\theta};\theta)].$$

If both methods perform a similar amount of exploration, $\mathrm{Var}[R(\mathbf{a})]$ will be similar for both expressions. The difference will thus be in the second term. Here we have that $\nabla_\theta \log p(\mathbf{a};\theta) = \sum_{t=1}^{T} \nabla_\theta \log p(a_t;\theta)$ is a sum of T uncorrelated terms, so that the variance of the policy gradient estimator will grow nearly linearly with T. The corresponding term for evolution strategies, $\nabla_\theta \log p(\tilde{\theta};\theta)$, is independent of T. Evolution strategies will thus have an advantage compared to policy gradients for long episodes with very many time steps. In practice, the effective number of steps T is often reduced in policy gradient methods by discounting rewards. If the effects of actions are short-lasting, this allows us to dramatically reduce the variance in our gradient estimate, and this has been critical to the success of applications such as Atari games. However, this discounting will bias our gradient estimate if actions have long lasting effects. Another strategy for reducing the effective value of T is to use value function approximation. This has also been effective, but once again runs the risk of biasing our gradient estimates. Evolution strategies is thus an attractive choice if the effective number of time steps T is long, actions have long-lasting effects, and if no good value function estimates are available.

[arXiv: 1703.03864]

Recap: Evolutionary Algorithms

Advantages

- derivative-free optimisation method (no gradient required)
  - the fitness function can be noisy and non-smooth
  - optimisation is more robust to local minima (saddle points) than gradient-based methods
- only requires to know how to evaluate the objective function (fitness)
  - required for problems with little domain knowledge
- straightforward parallelisation of computing the fitness score for each candidate
- conceptual simplicity (flexible to adapt to specific problems)

Limitations

- if we know more about the objective function, than just how to compute it, then we can make use of this information, and construct an algorithm that performs better, e.g. using gradient information

Topics

- Natural Search Gradient
- Compressed Network Search
- Evolution of Neural Network Topologies

Stochastic Variational Optimisation

Variational Optimization is based on the simple observation

$$\min_x f(x) \leq \mathbb{E}[f(x)]_{p(x|\theta)} \tag{1}$$

where θ is a set of continuous parameters of the variational distribution p. That is, the minimum of a collection of values is always less than their average. By defining

$$U(\theta) = \mathbb{E}[f(x)]_{p(x|\theta)} \tag{2}$$

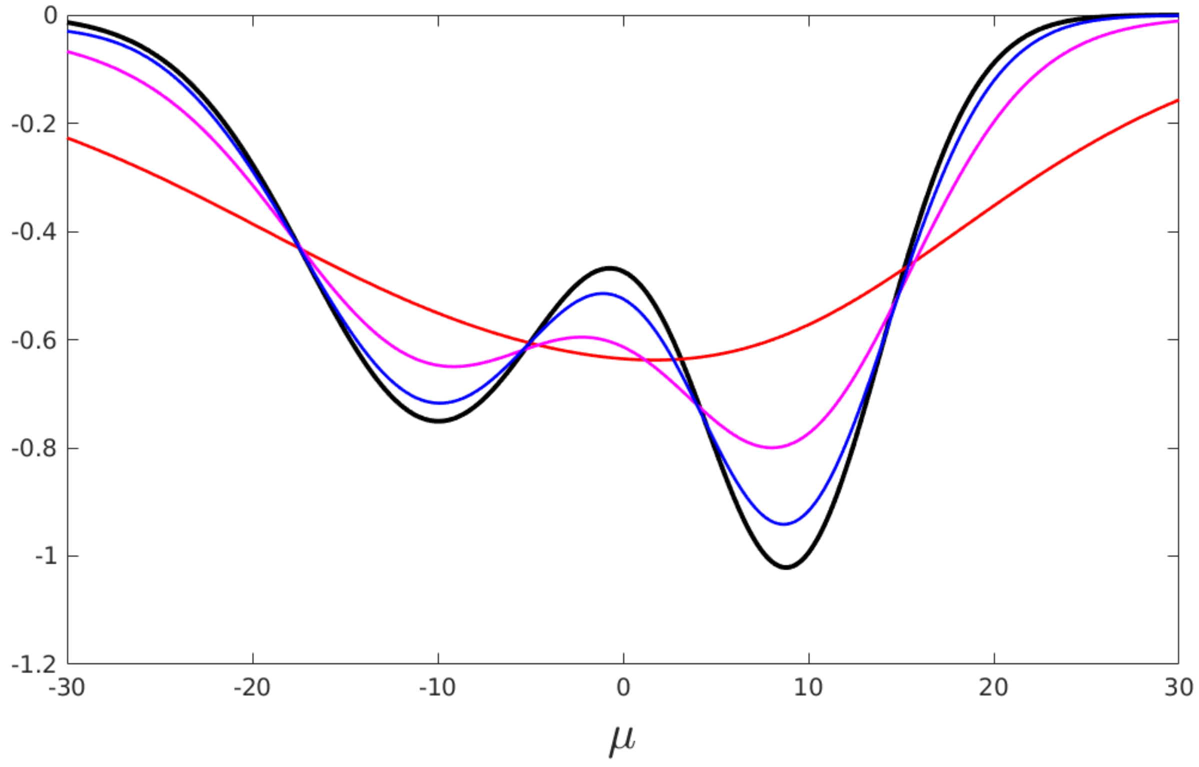
Instead of minimizing f with respect to x, we can minimize the upper bound U with respect to θ. In the original VO work [5] the focus was on forming a differentiable upper bound for non-differentiable f or discrete x.

The gradient of the upper bound can be computed by any standard means. However, it is interesting to express it as

$$\frac{\partial U}{\partial \theta} = \mathbb{E}\left[f(x)\frac{\partial}{\partial \theta}\log p(x|\theta)\right]_{p(x|\theta)}$$

which is reminiscent of the 'reinforce' algorithm [6].

[http://arxiv.org/abs/1809.04855]



Plain Search Gradient

- The new “objective function” $J(\theta)$ is the expectation of the original fitness function $f(z)$ under the search distribution $\pi(z|\theta)$. The gradient is given by:

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{z\sim\pi(z|\theta)}[f(z)] = \int f(z)\, \nabla_\theta \pi(z|\theta)\, dz\ .$$

Algorithm 2: Search Gradient algorithm: Multinormal distribution

```
input: $f$, $\boldsymbol{\mu}_{init}$,$\boldsymbol{\Sigma}_{init}$
repeat
    for $k = 1 \ldots \lambda$ do
        draw sample $\mathbf{z}_k \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$
        evaluate the fitness $f(\mathbf{z}_k)$
        calculate log-derivatives:
          $\nabla_{\boldsymbol{\mu}} \log \pi\left(\mathbf{z}_k|\theta\right) = \boldsymbol{\Sigma}^{-1}\left(\mathbf{z}_k - \boldsymbol{\mu}\right)$
          $\nabla_{\boldsymbol{\Sigma}} \log \pi\left(\mathbf{z}_k|\theta\right) = -\frac{1}{2}\boldsymbol{\Sigma}^{-1} + \frac{1}{2}\boldsymbol{\Sigma}^{-1}\left(\mathbf{z}_k - \boldsymbol{\mu}\right)\left(\mathbf{z}_k - \boldsymbol{\mu}\right)^\top \boldsymbol{\Sigma}^{-1}$
    end
    $\nabla_{\boldsymbol{\mu}} J \leftarrow \frac{1}{\lambda}\sum_{k=1}^{\lambda} \nabla_{\boldsymbol{\mu}} \log \pi(\mathbf{z}_k|\theta) \cdot f(\mathbf{z}_k)$
    $\nabla_{\boldsymbol{\Sigma}} J \leftarrow \frac{1}{\lambda}\sum_{k=1}^{\lambda} \nabla_{\boldsymbol{\Sigma}} \log \pi(\mathbf{z}_k|\theta) \cdot f(\mathbf{z}_k)$
    $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} + \eta \cdot \nabla_{\boldsymbol{\mu}} J$
    $\boldsymbol{\Sigma} \leftarrow \boldsymbol{\Sigma} + \eta \cdot \nabla_{\boldsymbol{\Sigma}} J$
until stopping criterion is met
```

[http://www.jmlr.org/papers/volume15/wierstra14a/wierstra14a.pdf]

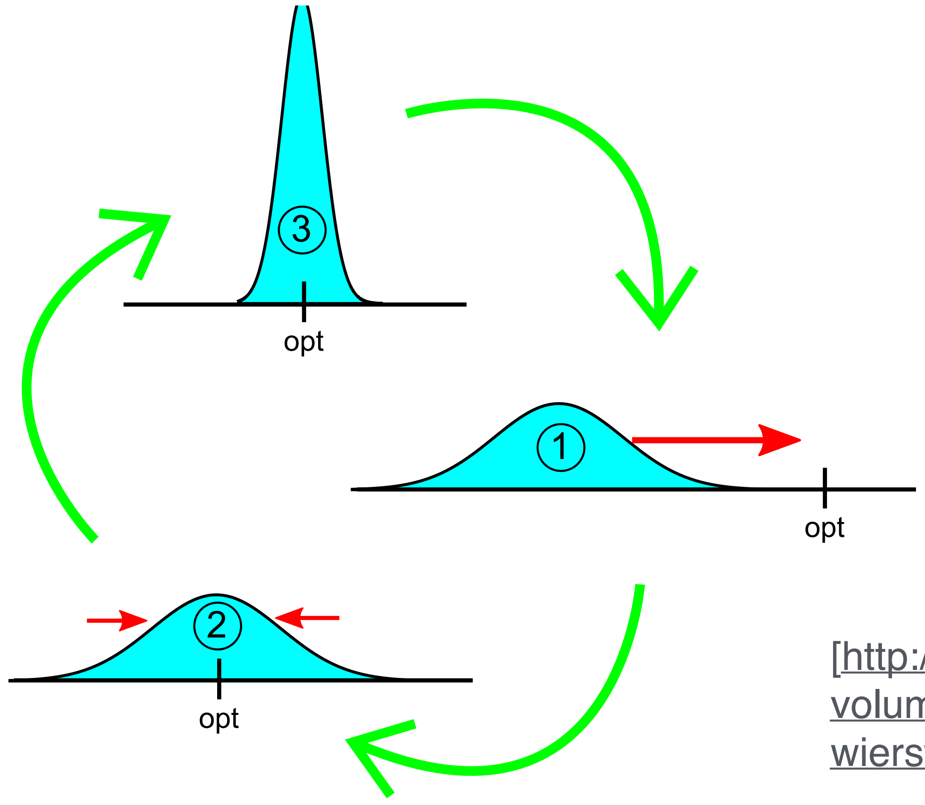
Problem of Plain Search Gradient

- Consider the 1-dimensional Gaussian case, with $\theta = (\mu, \sigma)$, and samples $z \sim N(\mu, \sigma)$. The gradients on μ and σ become

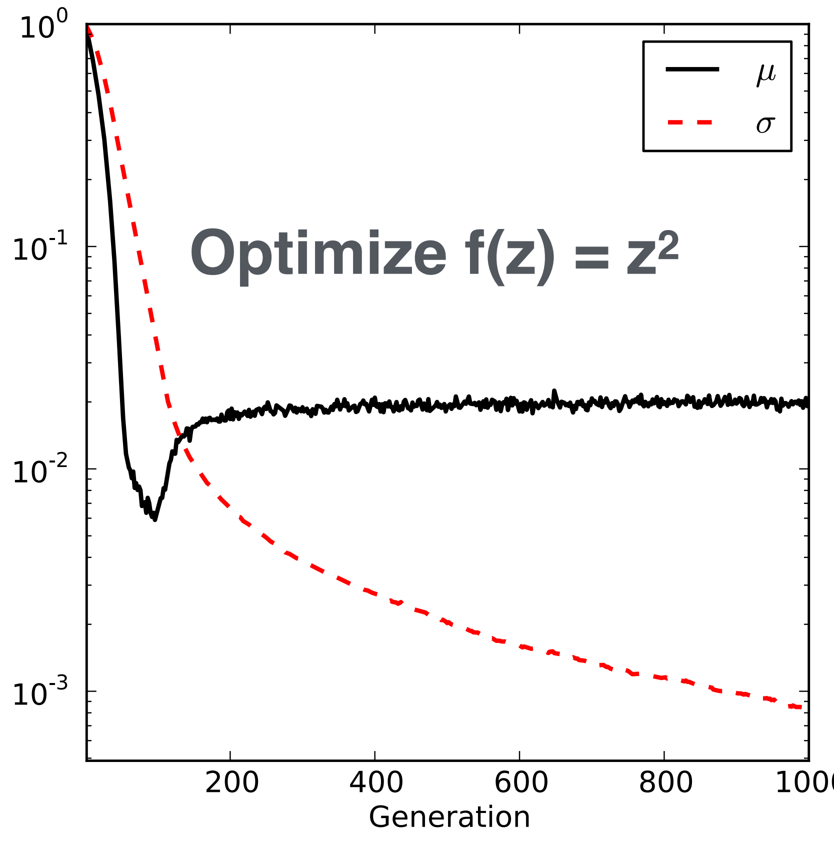
$$\nabla_\mu \log \pi(z|\theta) = \frac{z-\mu}{\sigma^2}, \qquad \nabla_\sigma \log \pi(z|\theta) = \frac{(z-\mu)^2 - \sigma^2}{\sigma^3}.$$

- The gradient updates, assuming simple hill-climbing (population size $\lambda = 1$), read

$$\mu \leftarrow \mu + \eta \frac{z-\mu}{\sigma^2} f(z), \qquad \sigma \leftarrow \sigma + \eta \frac{(z-\mu)^2 - \sigma^2}{\sigma^3} f(z).$$



[http://www.jmlr.org/papers/volume15/wierstra14a/wierstra14a.pdf]

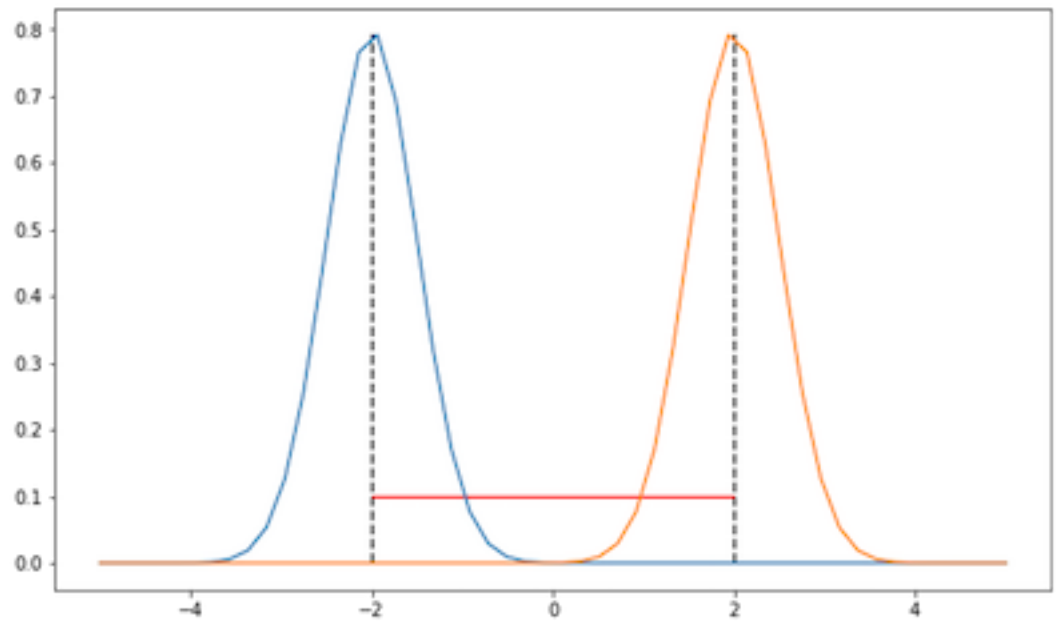


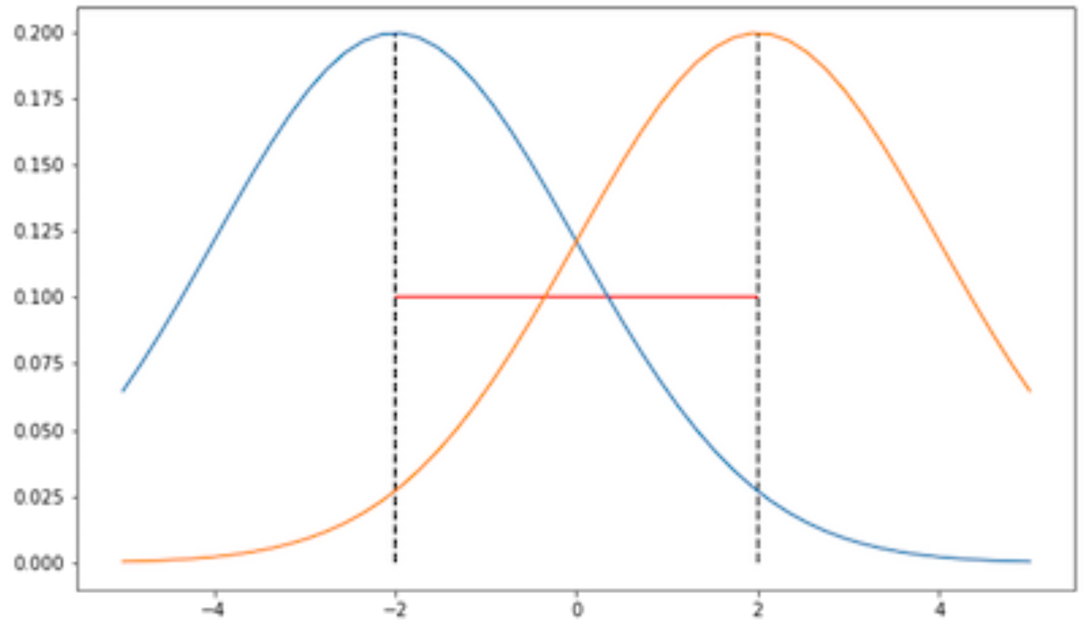
Problem of Plain Search Gradient

- The plain gradient $\nabla_\theta J(\theta)$ simply follows the steepest ascent in the space of the parameters θ (search space):

$$\max_{\delta\theta} J(\theta + \delta\theta) \approx J(\theta) + \delta\theta^T \nabla_\theta J(\theta) \quad s.t. \, \|\delta\theta\| = \epsilon \tag{1}$$

- For a small step-size $\epsilon > 0$, following $\nabla_\theta J(\theta)$ yields a new distribution with parameters chosen from a hypersphere of radius ϵ and centre θ that maximises $J(\theta)$.
- The Euclidean distance in parameter space is used to measure the distance between subsequent distributions.
- This makes the update dependent on the parameterisation of the distribution, a change of parameterisation leads to different gradients and different updates.





Natural Search Gradient

The natural gradient can then be formalized as the solution to the constrained optimization problem

$$\max_{\delta\theta} J(\theta + \delta\theta) \approx J(\theta) + \delta\theta^\top \nabla_\theta J,$$
$$s.t.\ D(\theta + \delta\theta || \theta) = \varepsilon, \tag{5}$$

where $J(\theta)$ is the expected fitness of Equation (1), and ε is a small increment size. Now, we have for $\lim \delta\theta \to 0$,

$$D(\theta + \delta\theta || \theta) = \frac{1}{2} \delta\theta^\top \mathbf{F}(\theta)\, \delta\theta,$$

where

$$\begin{aligned} \mathbf{F} &= \int \pi(\mathbf{z}|\theta)\, \nabla_\theta \log \pi(\mathbf{z}|\theta)\, \nabla_\theta \log \pi(\mathbf{z}|\theta)^\top d\mathbf{z} \\ &= \mathbb{E}\left[\nabla_\theta \log \pi(\mathbf{z}|\theta)\, \nabla_\theta \log \pi(\mathbf{z}|\theta)^\top\right] \end{aligned}$$

is the *Fisher information matrix* of the given parametric family of search distributions. The solution to the constrained optimization problem in Equation (5) can be found using a Lagrangian multiplier (Peters, 2007), yielding the necessary condition

$$\mathbf{F}\delta\theta = \beta \nabla_\theta J,$$

for some constant $\beta > 0$. The direction of the natural gradient $\widetilde{\nabla}_\theta J$ is given by $\delta\theta$ thus defined. If $\mathbf{F}$ is invertible,[1] the natural gradient amounts to

$$\widetilde{\nabla}_\theta J = \mathbf{F}^{-1} \nabla_\theta J(\theta).$$

[http://www.jmlr.org/papers/volume15/wierstra14a/wierstra14a.pdf]

Natural Evolution Strategies

Algorithm 3: Canonical Natural Evolution Strategies

```
input: $f$, $\theta_{init}$
repeat
    for $k = 1 \dots \lambda$ do
        draw sample $\mathbf{z}_k \sim \pi(\cdot|\theta)$
        evaluate the fitness $f(\mathbf{z}_k)$
        calculate log-derivatives $\nabla_\theta \log \pi(\mathbf{z}_k|\theta)$
    end
    $\nabla_\theta J \leftarrow \frac{1}{\lambda} \sum_{k=1}^{\lambda} \nabla_\theta \log \pi(\mathbf{z}_k|\theta) \cdot f(\mathbf{z}_k)$
    $\mathbf{F} \leftarrow \frac{1}{\lambda} \sum_{k=1}^{\lambda} \nabla_\theta \log \pi\left(\mathbf{z}_k|\theta\right) \nabla_\theta \log \pi\left(\mathbf{z}_k|\theta\right)^\top$
    $\theta \leftarrow \theta + \eta \cdot \mathbf{F}^{-1} \nabla_\theta J$
until stopping criterion is met
```

Limitations

- computational cost of computing F is $O(d^2)$
- F is subject to sampling errors for finite λ

Compressed Network Search

Problem

- search space is too high-dimensional to apply GA on large neural networks, e.g. greater than 1 million weights for a candidate in state of the art policy networks in RL

Idea

- view network's weights as images and images can be compressed, e.g. jpg
- encode weight matrices in the frequency domain by sets of Fourier coefficients using the Discrete Cosine Transformation (DCT)
- compression is done by erasing high-frequency coefficients (like in lossy image compression)
- now, perform search in the lower-dimensional (compressed) search space (Fourier coefficients) to solve specific tasks, e.g. pole-balancing
- this method can be seen as an inductive bias that the weights admit some intrinsic structure / pattern

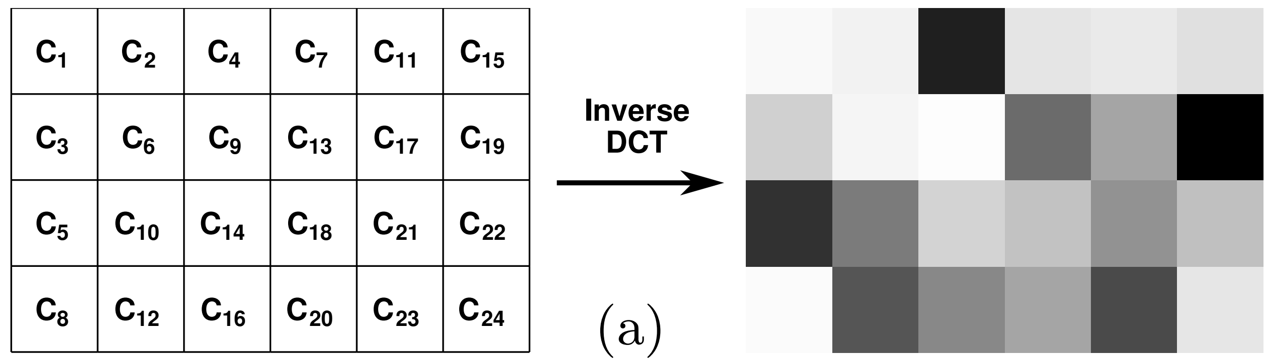
see details in
[Koutnik, Gomez, Schmidhuber: Evolving Neural Networks in Compressed Weight Space (2010)]
[Srivastava, Schmidhuber, Gomez: Generalized Compressed Network Search (2012)]

Compressed Network Search

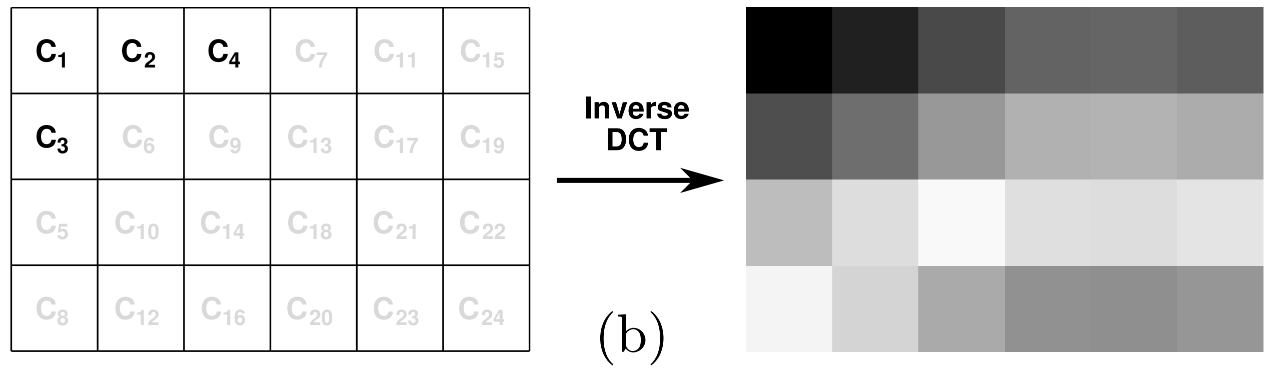
ℓ

Fourier coefficient index	index	4	78	0	12	7	12	45	0	97	5
Fourier coefficient value	value	5.65	−2.32	6.52	−12.1	2.10	3.46	−5.	−7.38	−3.98	1.87

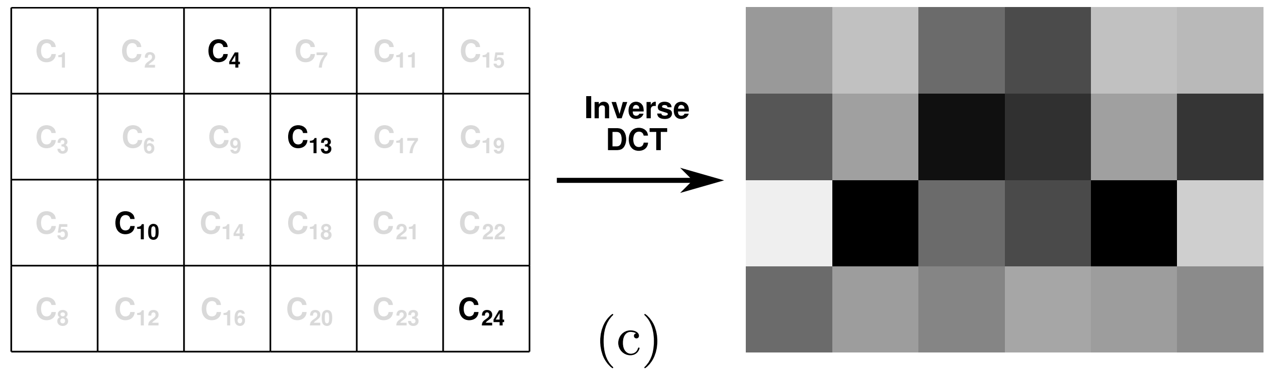
“chromosome”



all frequencies are present, so that all possible weight matrices can be represented



only the first four Fourier coefficients are used, weights are more spatially correlated than in (a)



shows a weight matrix encoded by a subset of frequencies from (a)

[Srivastava, Schmidhuber, Gomez: Generalized Compressed Network Search (2012)]

Compressed Network Search

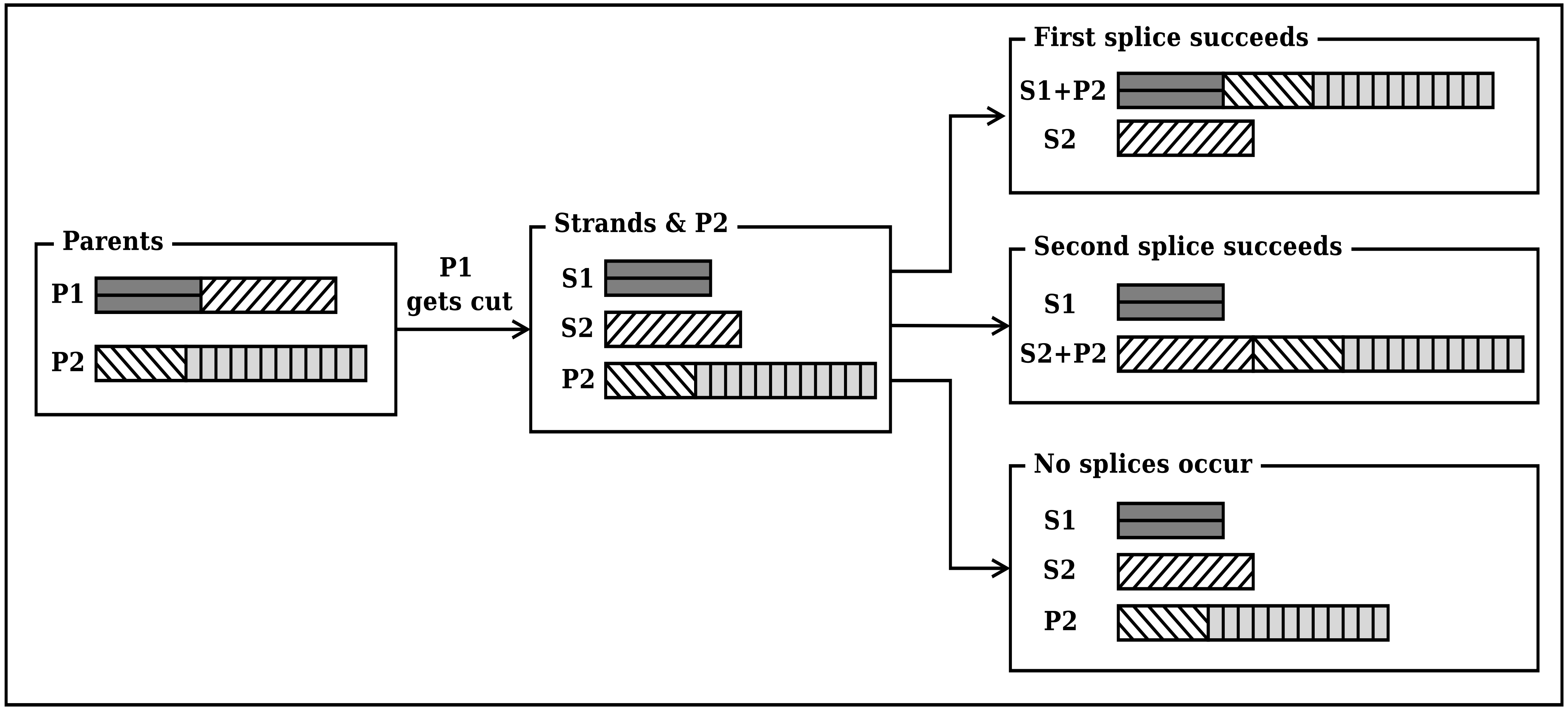


Fig. 3. Cut and Splice. This schematic shows the effect of application of the cut and splice operators on a set of two parent genomes. In the case shown, only P1 gets cut resulting in three chromosomes (strands S1, S2 and parent P2). Then splice is applied with probability p_s. If the first splice succeeds, then S1 gets spliced with P2, leaving S2 as a separate genome. If first splice does not occur, another splice between S2 and P2 can lead to the two children shown if it succeeds. If both splices do not succeed, S1, S2 and P2 become the final children as shown. Similar possibilities exist for other cases of the parents getting cut.

[Srivastava, Schmidhuber, Gomez: Generalized Compressed Network Search (2012)]

Compressed Network Search

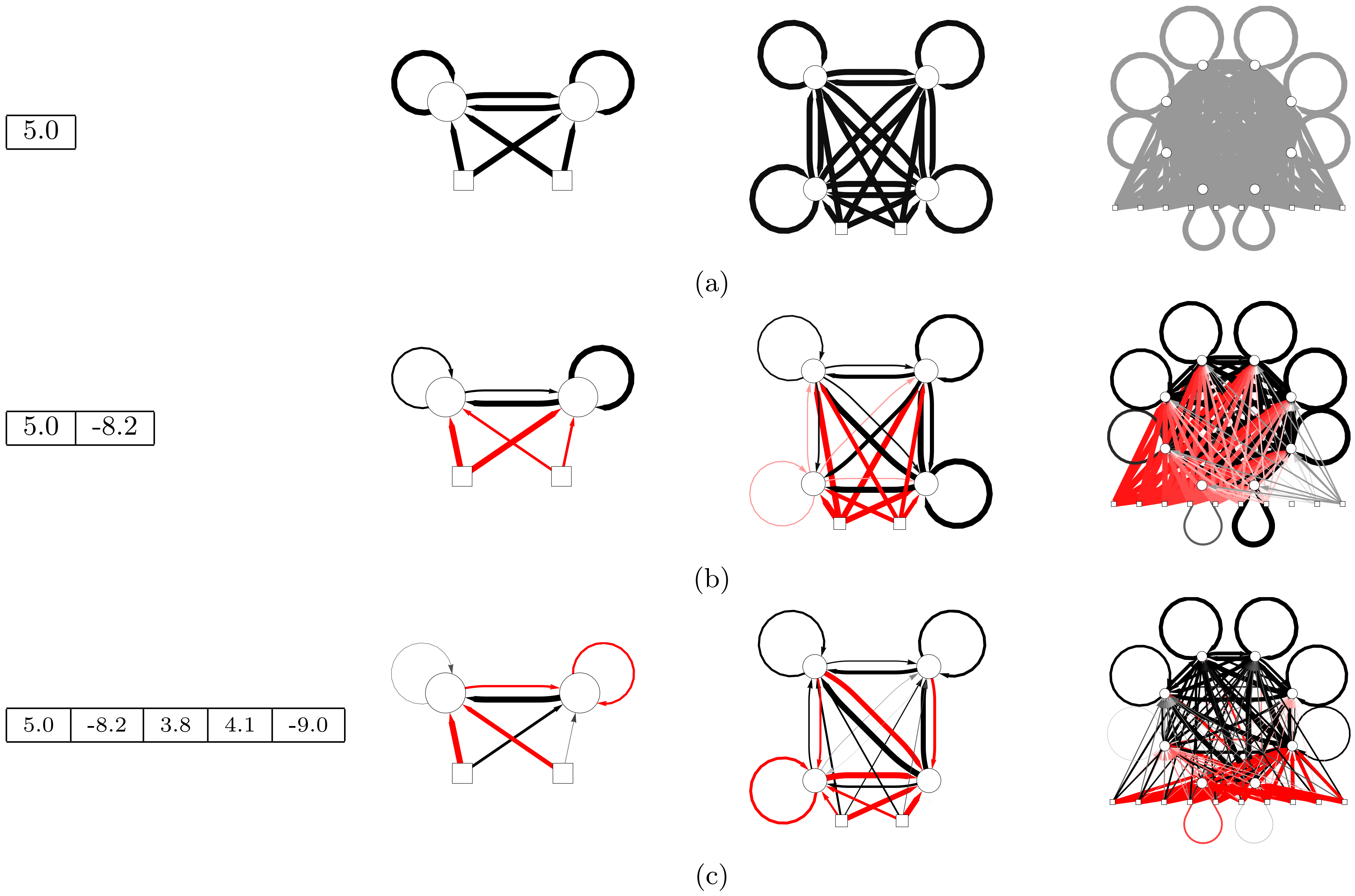


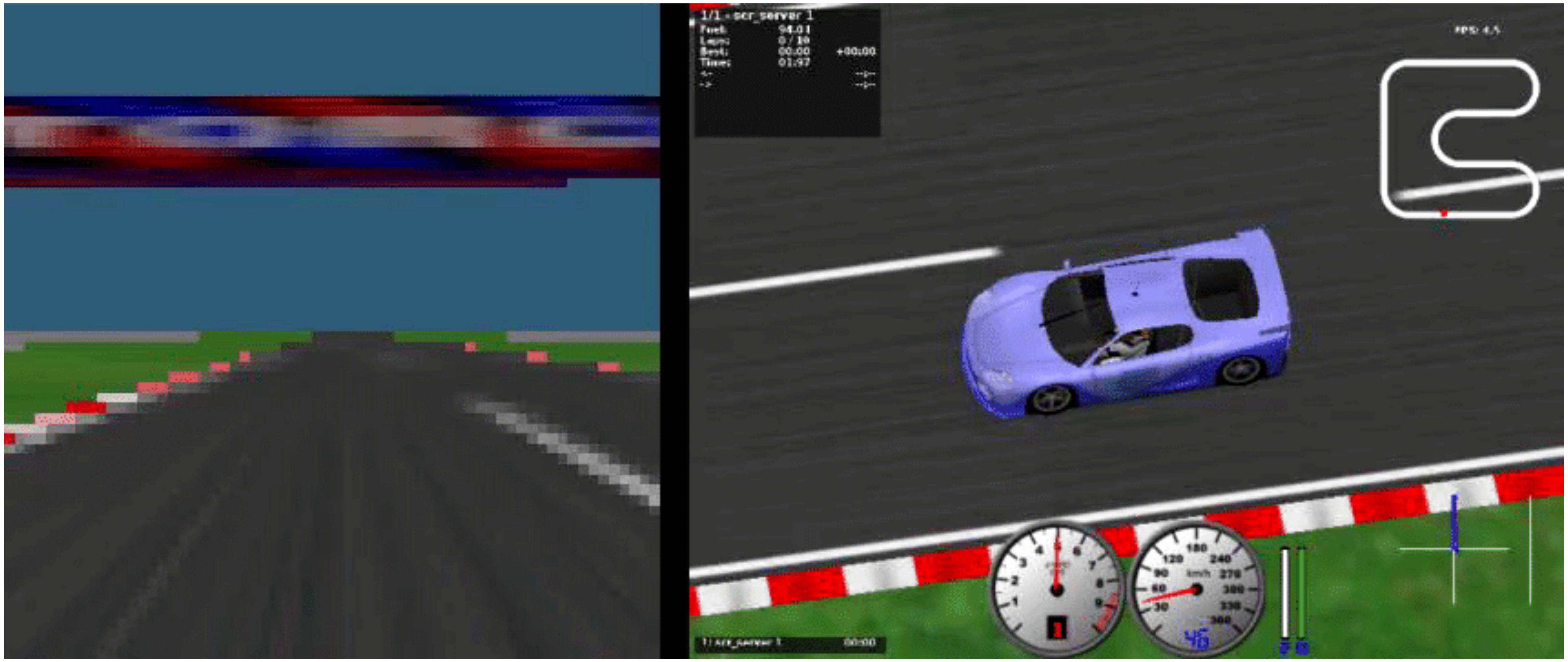
Figure 2: Mapping from DCT genotype to FRNN phenotypes. For each chromosome in the left column, three different networks with different architectures are shown, instantiated with weights generated by applying the inverse DCT. The potential complexity of the weight matrices increases with the number of DCT coefficient genes. The small squares in the networks denote input units, the circles are neurons. The thickness of a connection (arrow) corresponds to the relative magnitude of its weight, and the weight is positive if the arrow is dark (black) and negative if light (red).

[Koutnik, Gomez, Schmidhuber: Evolving Neural Networks in Compressed Weight Space (2010)]

Compressed Network Search

Example: TORCS (racing simulator)

- control race car to drive along a track using a visual stream from the driver's perspective
- policy has > 1 million weights, and reduces to 200 DCT coefficients (indirect encoding)
- searching the smaller space of DCT coefficients using evolution strategies is tractable
- evolve a recurrent neural network controller that can drive the car around a race track
- coefficients are evolved using Cooperative Synapse NeuroEvolution (64 population size)
- fitness scores roughly correspond to the distance traveled along the race track axis



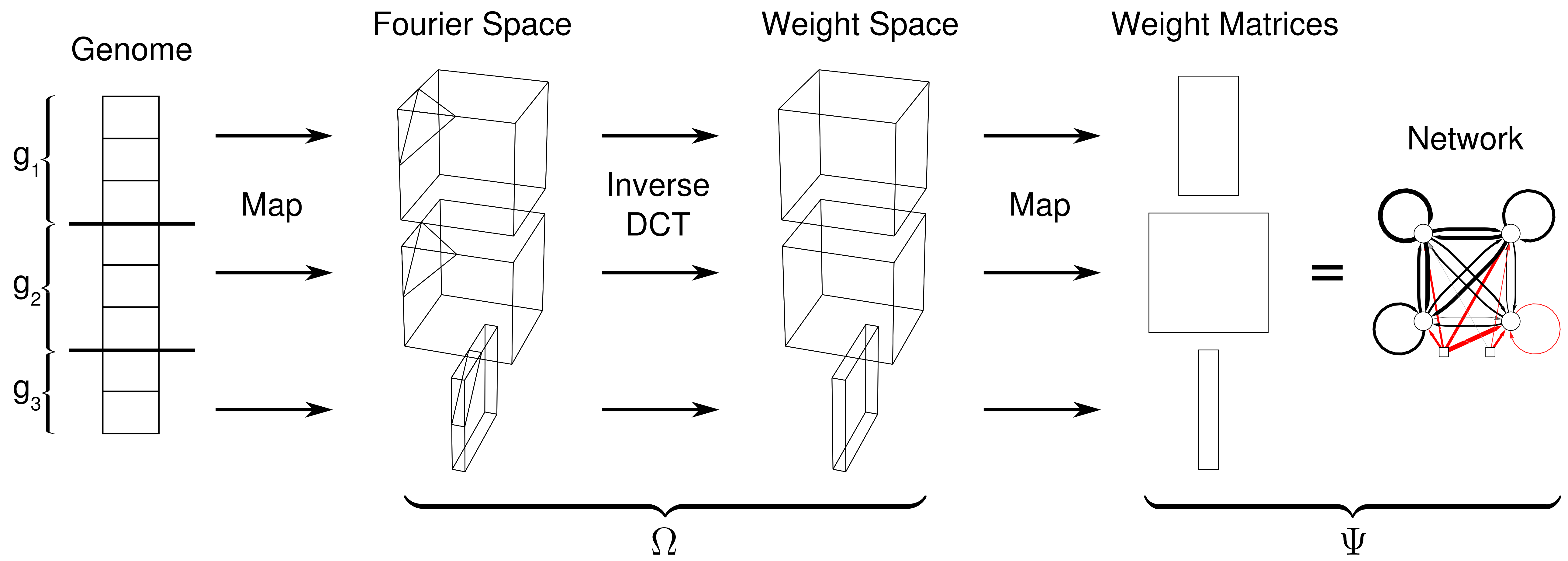
input images

[Koutnik et al: Evolving Large-Scale Neural Networks for Vision-Based TORCS (2013)]

Compressed Network Search

Decoding the compressed networks

- The genome is divided into k chromosomes, one for each of the weight matrices specified by the network architecture.
- Each chromosome is mapped into a coefficient array of a specified dimensionality. In this example, an RNN with two inputs and four neurons is encoded as 8 coefficients.
- The next step is to apply the inverse DCT to each array to generate the weight values, which are mapped into the weight matrices in the last step.

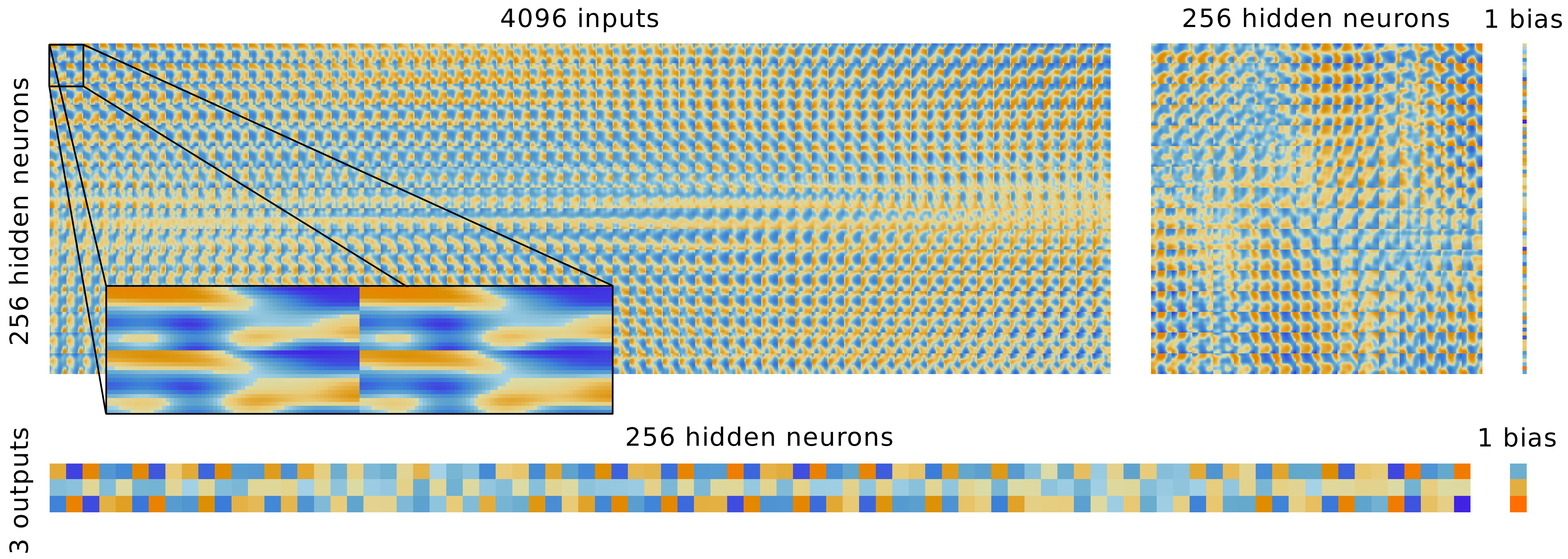


[Koutnik et al: Evolving Large-Scale Neural Networks for Vision-Based TORCS (2013)]

Compressed Network Search

Evolved low-complexity weight matrices of the policy

- colours indicate weight value
- the frequency-domain representation enforces clear regularity on weight matrices that reflects the regularity of the environment



[Koutnik et al: Evolving Large-Scale Neural Networks for Vision-Based TORCS (2013)]

NEAT

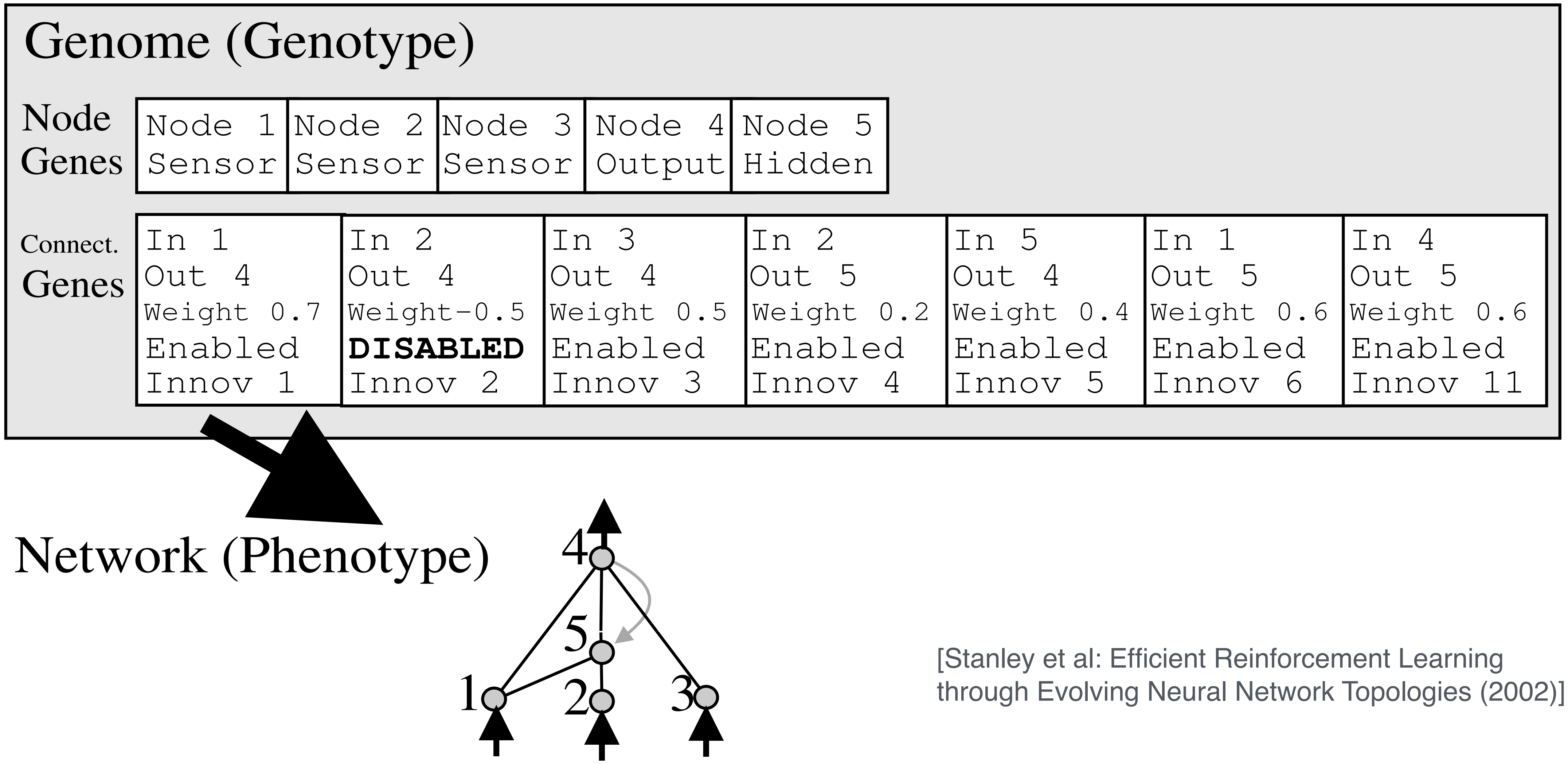
Idea

- evolve neural networks using GA (NeuroEvolution of Augmented Topologies)
- main idea is that it is most effective to start evolution with small, simple networks and allow them to become increasingly complex over generations
- genotype consists of list of nodes (neurons) and connection weights (synapses)
- key aspects
  - genetic encoding with historical markings
  - speciation
  - complexification

[Stanley et al: Efficient Reinforcement Learning through Evolving Neural Network Topologies (2002)]

Genetic Encoding - NEAT

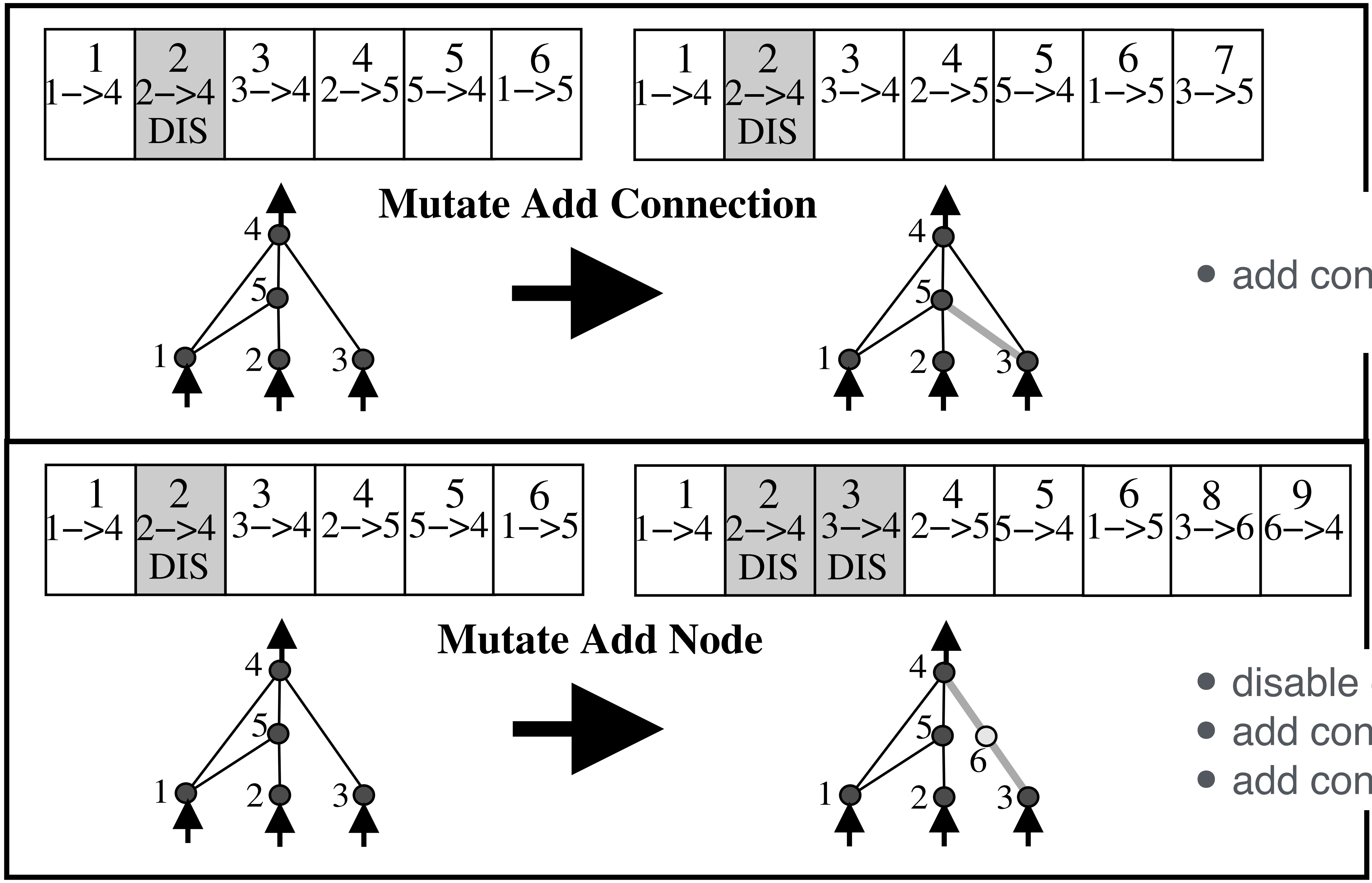
- variable-size chromosomes
- connection gene: in-node, out-node, weight and status
- node gene: input (sensor), output or hidden node
- each gene carries innovation number which allows chromosomes to be lined up
  - it is important to overlay two topologies for crossover
  - each new connection gene has its own innovation number


Genome (Genotype)
Node Genes
Node 1 Sensor
Node 2 Sensor
Node 3 Sensor
Node 4 Output
Node 5 Hidden
Connect. Genes
In 1 Out 4 Weight 0.7 Enabled Innov 1
In 2 Out 4 Weight-0.5 DISABLED Innov 2
In 3 Out 4 Weight 0.5 Enabled Innov 3
In 2 Out 5 Weight 0.2 Enabled Innov 4
In 5 Out 4 Weight 0.4 Enabled Innov 5
In 1 Out 5 Weight 0.6 Enabled Innov 6
In 4 Out 5 Weight 0.6 Enabled Innov 11
Network (Phenotype)
4
5
1
2
3

[Stanley et al: Efficient Reinforcement Learning through Evolving Neural Network Topologies (2002)]

Mutation - NEAT

- mutation operators: add connection, add node, perturb connection weight
  - DIS = “disable”
- maintain mapping of mutations to innovation numbers

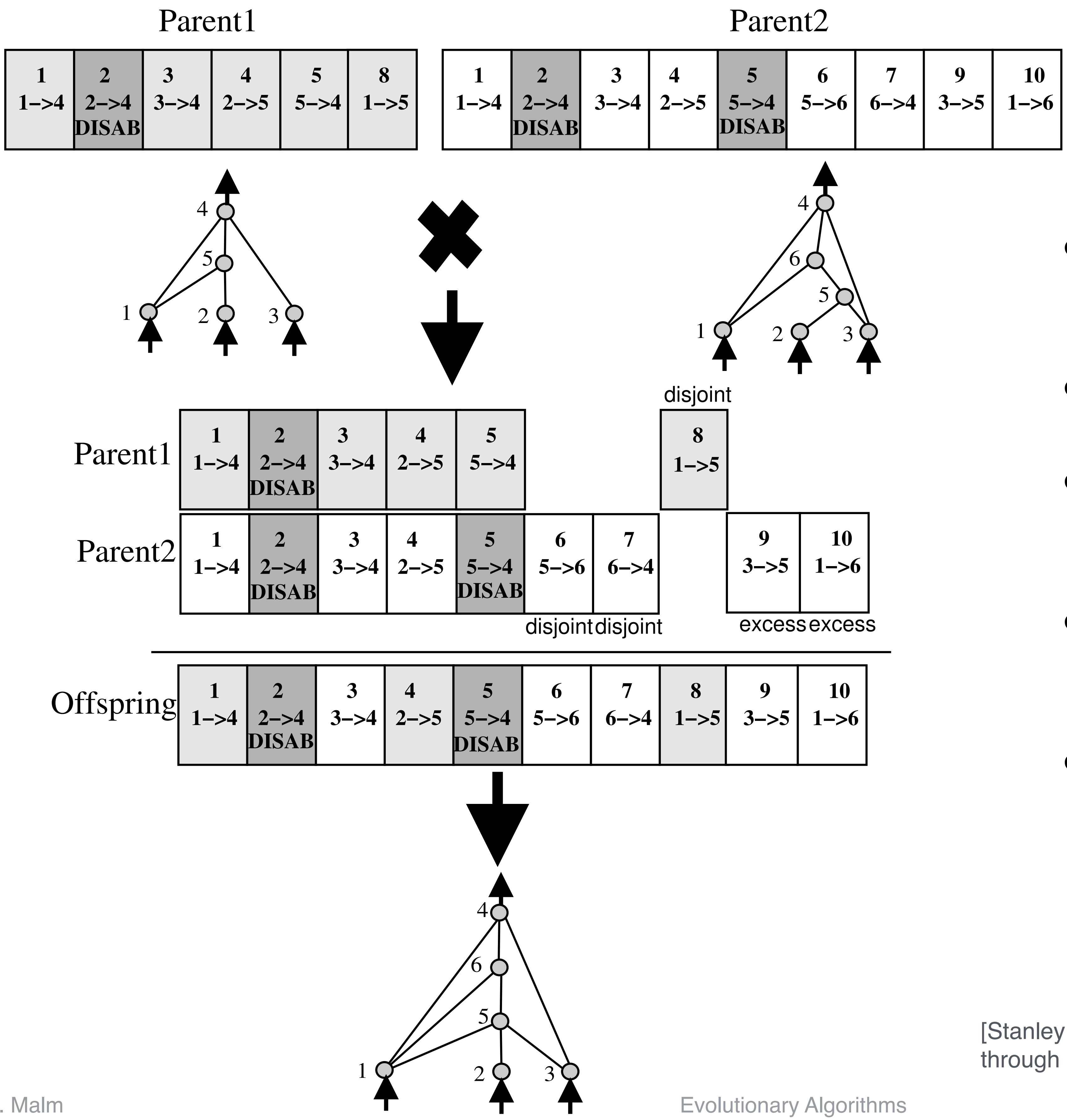


- add connection 3->5

- disable connection 3->4
- add connection 3->6
- add connection 6->4

[Stanley et al: Efficient Reinforcement Learning through Evolving Neural Network Topologies (2002)]

Crossover - NEAT



- genes in both genomes with the same innovation number are lined up
- these genes are called matching genes
- genes that do not match are either disjoint or excess
- matching genes are selected randomly for the offspring
- disjoint/excess genes are always included

[Stanley et al: Efficient Reinforcement Learning through Evolving Neural Network Topologies (2002)]

Speciation - NEAT

Problem

- smaller networks optimise faster than larger networks, i.e. adding nodes and connections initially decrease the fitness, compared to just perturbing weights
- this prevents diversity/innovation of network topologies

Solution: Speciation

- divide population into species such that similar topologies are in the same species, and then perform competition within each species
- measure “compatibility distance” between two chromosomes (topologies) as a linear combination of the number of excess (E), disjoint (D) and the average weight differences of matching genes (W)

$$\delta = \frac{c_1\, E}{N} + \frac{c_2\, D}{N} + c_3\, W$$

- N = number of genes in the larger genome, c1, c2, c3 are adjustable hyperparameters
- form a species if the distance of two networks is below a compatibility threshold

Complexification - NEAT

Idea: start with simplest network and then incrementally grow structure

- first generation starts with population of network with identical topology
  - minimal structure
  - no hidden nodes
  - possibly only one connection
- each network starts with random weights
- complexification is the process of introducing structure through add connection and add node mutations in an incremental process

NEAT

Double Pole Balancing

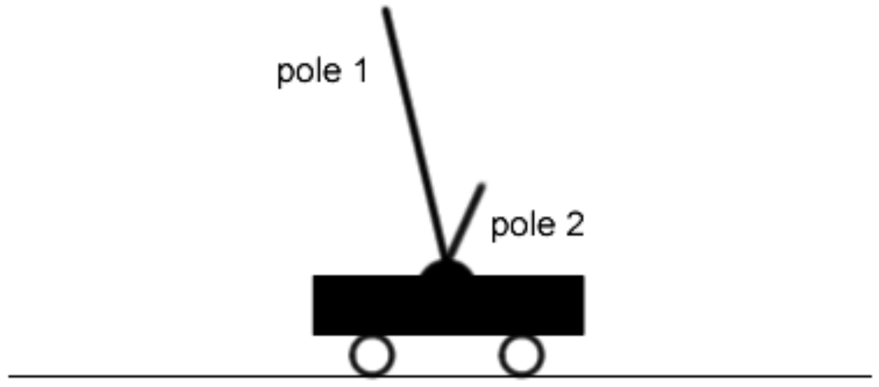
- with velocity information
- 150 NEAT networks

Method	Evaluations	Generations	No. Nets
Ev. Programming	**307,200**	150	2048
Conventional NE	**80,000**	800	100
SANE	**12,600**	63	200
ESP	**3,800**	19	200
NEAT	**3,578**	24	150

- without velocity information
- 1000 NEAT networks

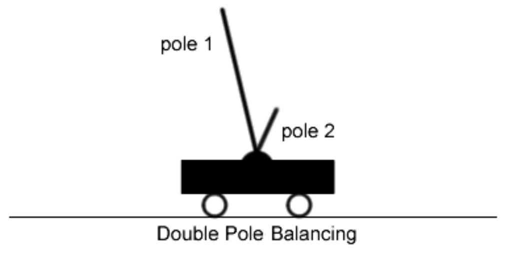
Method	Evaluations	Generalization	No. Nets
CE	**840,000**	300	16,384
ESP	**169,466**	289	1,000
NEAT	**33,184**	286	1,000

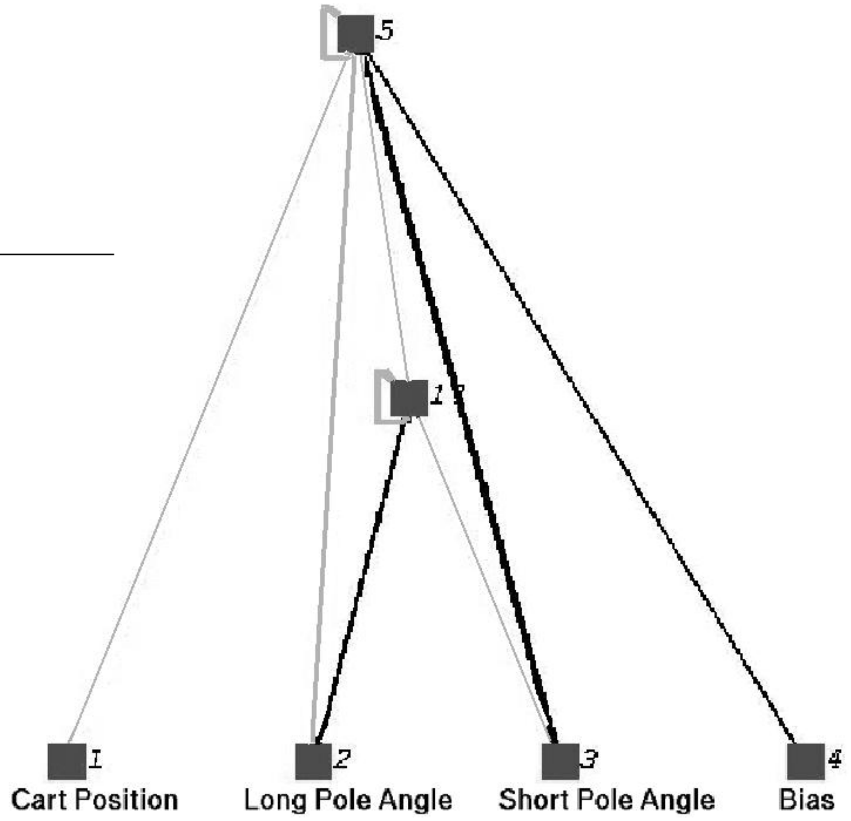
[Stanley et al: Efficient Reinforcement Learning through Evolving Neural Network Topologies (2002)]

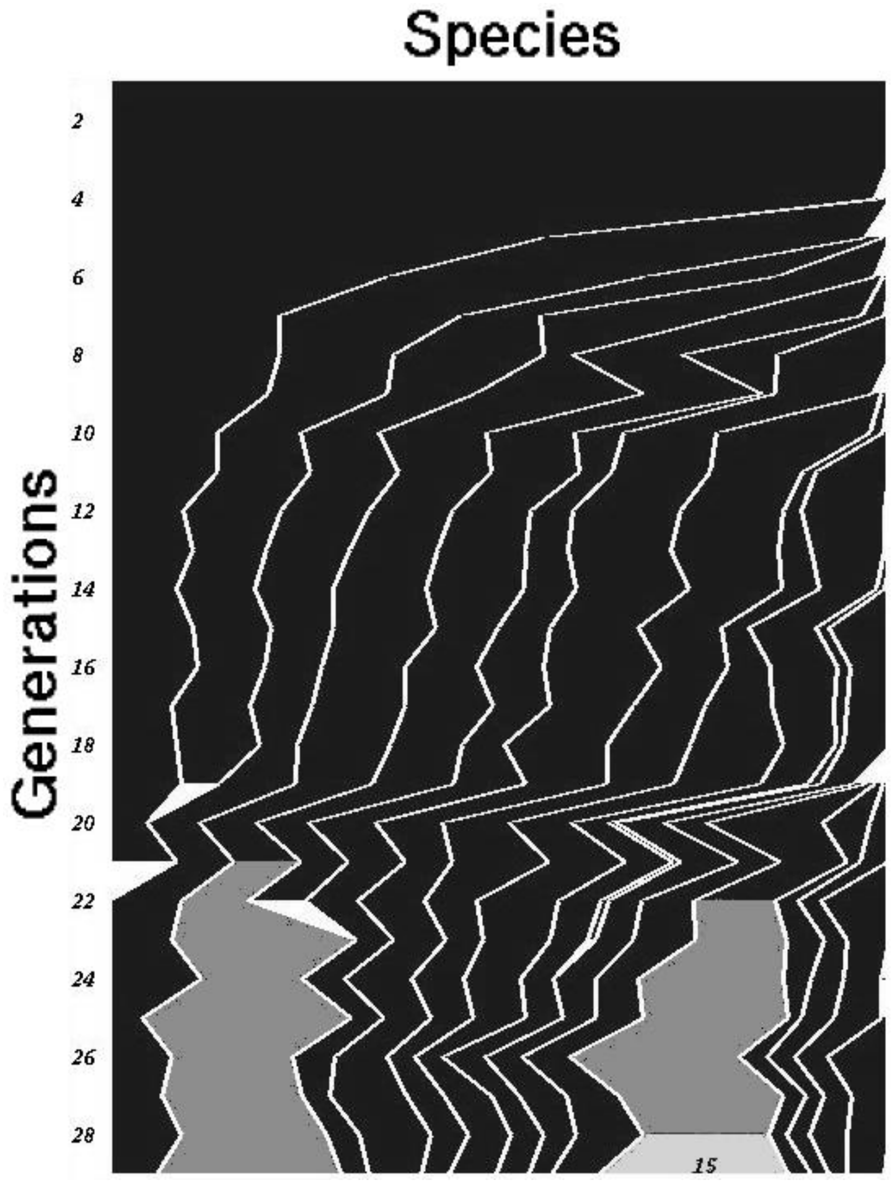


Double Pole Balancing

NEAT







- NEAT found a simple recurrent network
- using the recurrent connection to itself the single hidden node determines whether the poles are falling away or towards each other
- allows controlling the system without computing the velocities of each pole separately

[Stanley et al: Efficient Reinforcement Learning through Evolving Neural Network Topologies (2002)]