Evolutionary Computation

Machine Learning 2019

Part 1

Michael Wand, Jürgen Schmidhuber, Cesare Alippi
TAs: Robert Csordas, Krsto Prorokovic, Xingdong Zou, Francesco Faccio, Louis Kirsch

Slides credit: Faustino Gomez, Raoul Malm

Evolutionary Computation

So far, we had always *one* learning “agent”

- conventional supervised/unsupervised learning (NN, HMM, SVM): one parameter set to optimize
- RL: agent modifies its internal structure (policy) to adapt to observation of/interaction with environment

The paradigma of Evolutionary Computation is radically different:

- *population* of many agents learn collectively
- agents improve their parameters by processes modeled according to “natural selection”

Evolutionary Computation

Basic idea:

- Always have a pool of candidate solutions (population)
- search the solution space in parallel
- evaluate candidates and assign a (scalar) “fitness” score
- in each step, generate new population from fittest candidates
  - hope that the new population is better than the old one (in terms of fitness)
  - or at least, that some members are better

Advantages

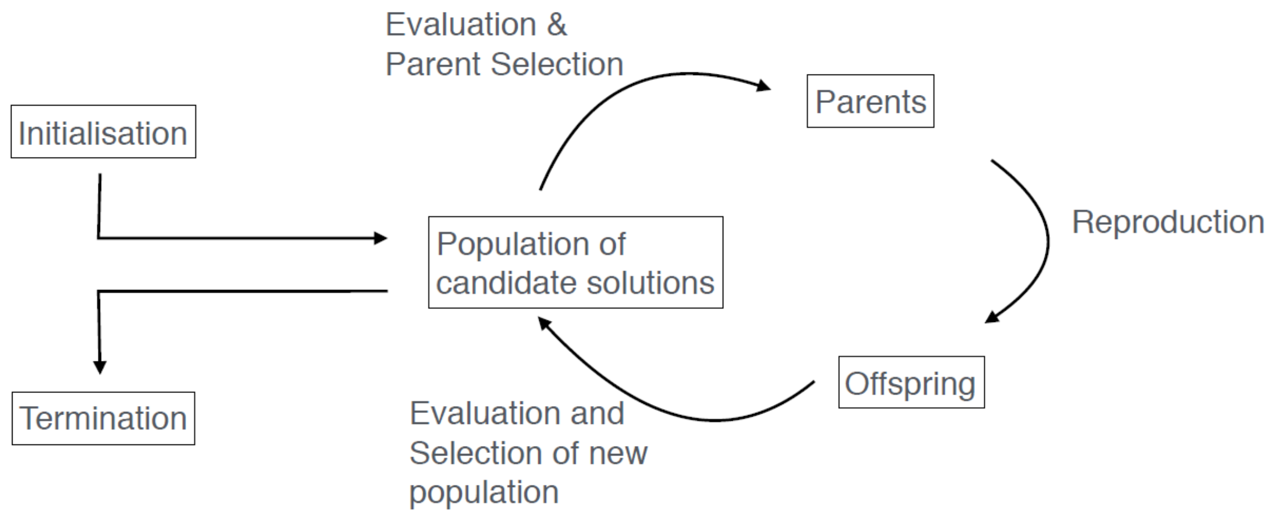
Very general fitness function

- – little training supervision required
- – usually one uses derivative-free optimization: we do not need how to compute a derivative of the objective function

- Powerful way to search solutions in complex/unexplored problem spaces
  - – no need for full mathematical description of problem space
  - – some heuristics required
- Can cope with high dimensionality, local minima, etc.
- Straightforward parallelization

we'll also get to know limitations...

Basic Idea



1. Initialize population with random candidate solutions
2. Repeat, until a termination condition is fulfilled
   a) Evaluate each candidate solution, assign scalar fitness score
   b) Select fittest candidates in the population for reproduction (parents)
   c) Reproduce new candidates (offspring) from parents
3. Return the fittest candidate

Fitness Landscapes

Optimization, machine learning, etc. are guided by objective functions

- A function is “difficult” if it is not continuous, not differentiable, or if it has multiple maxima and minima
- In particular, gradient-based optimization often fails for such functions

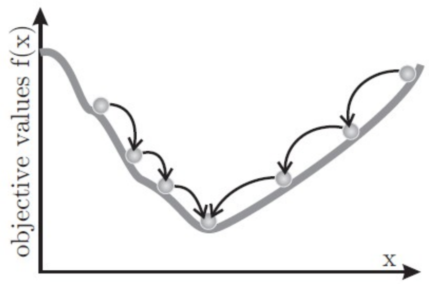


Fig. 1.19.a: Best Case

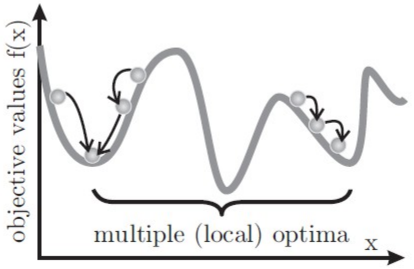


Fig. 1.19.c: Multimodal

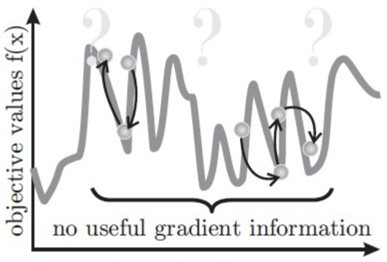
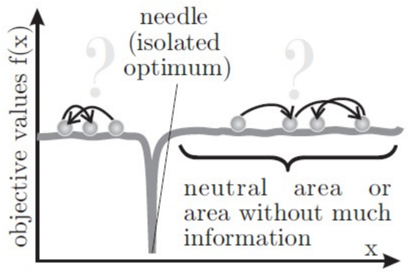


Fig. 1.19.d: Rugged


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 Algorithms - Theory and Application]

Advantages

- if the objective (fitness) function is multimodal/noisy/discontinuous
  - a population of “agents” increases the probability to find the global optimum
  - derivative-free optimisation methods are more robust to local optima and saddle points than gradient-based methods
- if we know little about the objective function (little domain knowledge)
  - we only need to know how to evaluate the fitness function
  - straightforward parallelisation
  - conceptual simplicity

Branches of Evolutionary Computation

- Genetic Algorithms (Holland 1975)
- Evolution Strategies (Schwefel 1973)
- Evolutionary Programming (Fogel 1966)
- Genetic Programming (Cramer 1985)

Branches of Evolutionary Computation

- Genetic Algorithms (Holland 1975)
- Evolution Strategies (Schwefel 1973)
- Evolutionary Programming (Fogel 1966)
- Genetic Programming (Cramer 1985)

Focus of this lecture

Genetic Algorithms

Basic definitions and algorithms

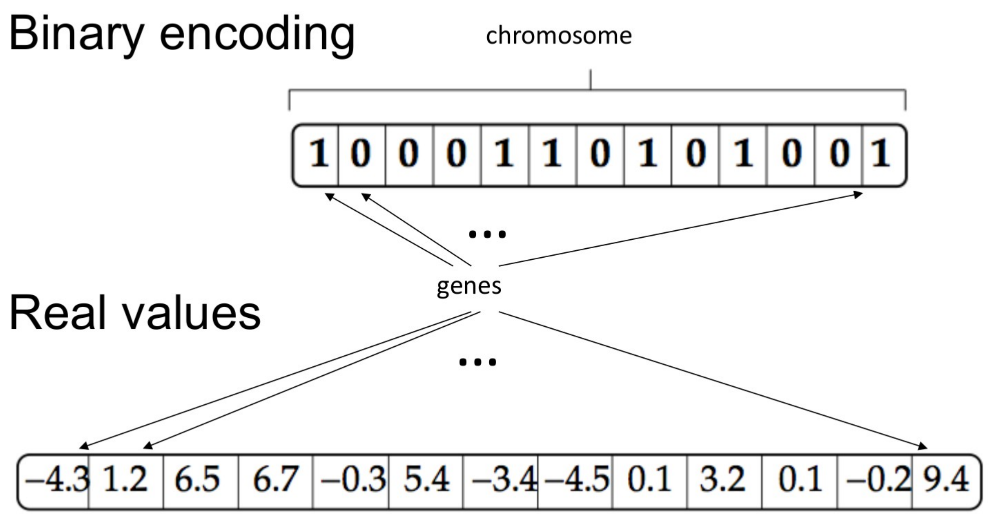
GA: Basic Procedure

1) **Initialize** random population of candidate solutions
2) **Evaluate** solutions on problem and assign a (scalar) fitness score
3) **Select** some solutions for “mating”
4) **Recombine**: create new solutions from existing ones by exchanging structure
5) If best (fittest) solution is not good enough: go to step 2

The cycle from 2 to 5 is known as a *generation.*

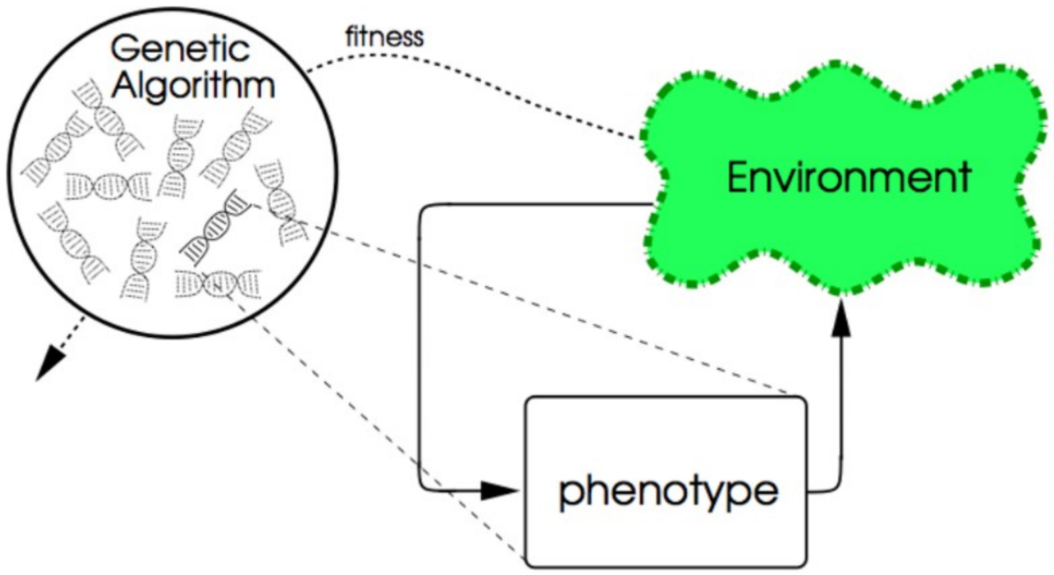
Biology lesson

- Solutions are encoded in strings called *chromosomes*
- Each chromosome consists of some number of *genes*
- Each gene can take an a value or *allele* from some specified alphabet, e.g.
  - Binary {0,1}
  - Real numbers (infinite alphabet)



Biology lesson (2)

- Genotypes can represent any kind of structure of *phenotype* in the problem space (i.e. environment)
- Before evaluating a genotype it must be mapped to its phenotype
- This mapping might be not completely specified, or even nondeterministic!



GA: Selection

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

- 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
  - can lead to low diversity of candidates
  - all candidates under investigation become similar to each other
- low diversity can imply that candidates get stuck in a local optimum and miss the global optimum

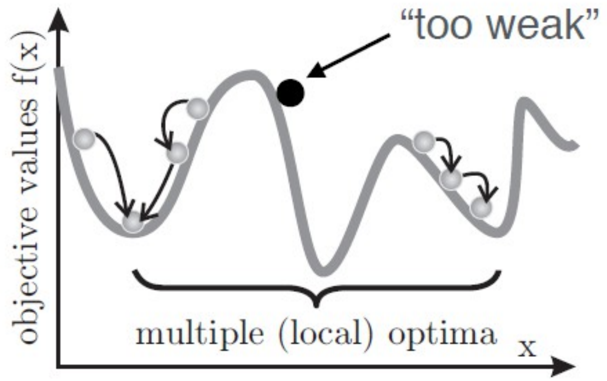


Fig. 1.19.c: Multimodal

Fitness Proportionate Selection

Idea:

- fitter candidates should have a higher chance of being selected (stochastic selection)
- the probability of a candidate θ_i being selected should be proportional to its fitness value:

$$p(\theta_i) = \frac{f(\theta_i)}{\sum_{j=1}^{n} f(\theta_j)}, \quad i = 1, \ldots, n$$

- Each candidate can in principle be selected
- *Scaling problem:* If one individual has very high fitness compared to the others, it will almost certainly be selected → again leads to low diversity

Linear Ranking Selection

- sort the genotypes by fitness: high rank → fit, low rank → less fit
- compute probability of being selected as follows:

$$p_i = 2 - SP + 2 * (SP - 1) * (\text{rank}(i) - 1)/(N - 1)$$

- *1.0 ≤ SP ≤ 2.0* is the *selective pressure*: the higher it is, the stricter is the selection of individuals with high fitness
  - must be carefully tuned
- finally as before, probabilistically select some number of genotypes for recombination

Tournament Selection

- Let *T* be the tournament size (between 2 and *N*, the population size)
- Select T genotypes at random from the population and take the most fit as the tournament "winner"
- Put the winner in the mating pool
- Continue until enough individuals for mating have been found

Larger values of T increase selective pressure

Selective Pressure and Diversity

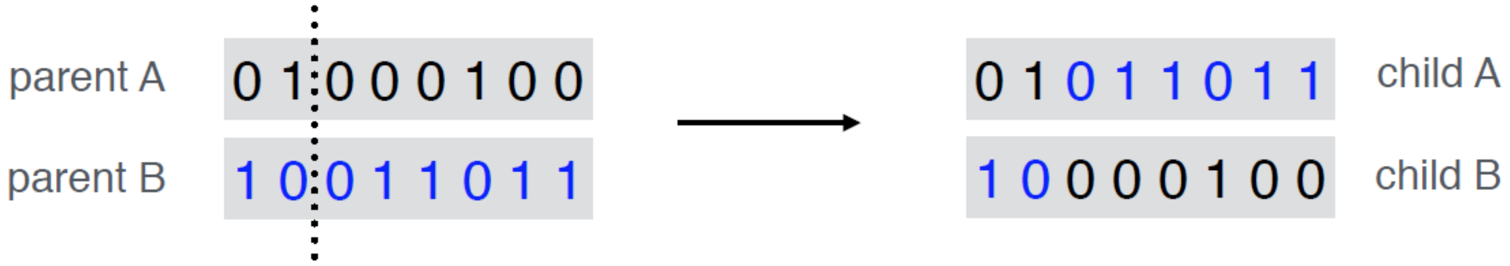
- Similar to exploitation/exploration tradeoff in RL
- We want selective pressure to be high enough to direct search towards a good solution,
  - but not so high that we converge too soon
- Diversity should be high so that we are less likely to “miss” a good solution,
  - but not so high that we don’t converge

Reproduction

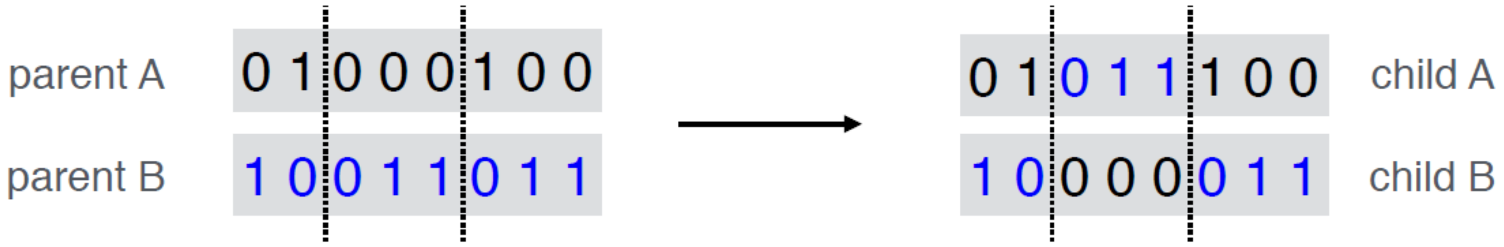
- Generate new individuals by mixing or altering the genotypes of those members of the population that are selected for mating
- **Crossover:** a form of recombination, select alleles from two parent chromosomes to form two children
- **Mutation:** randomly perturb some of the alleles of a parent

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

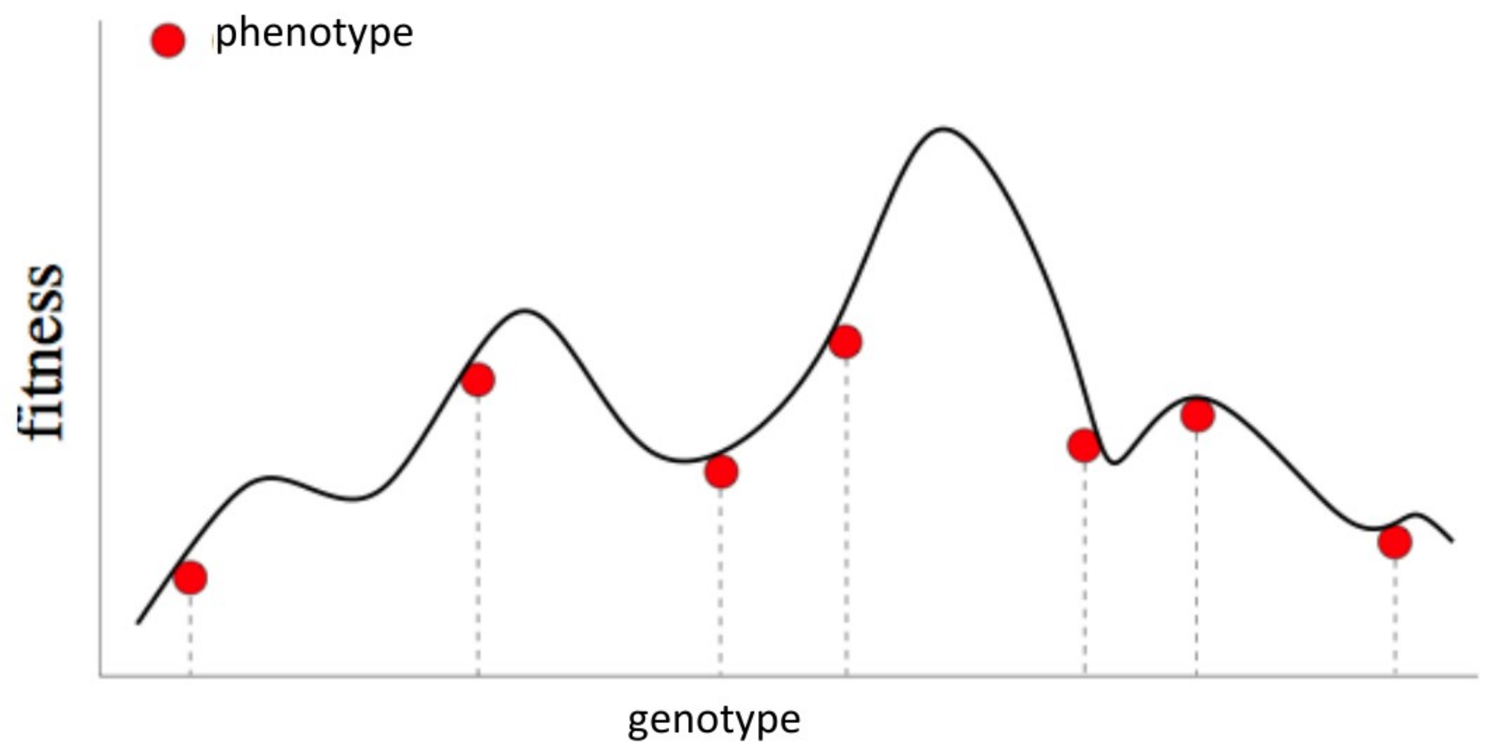
parent	-4.3	1.2	6.5	6.7	-0.3	0.1	0.8	-0.9

↓ 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

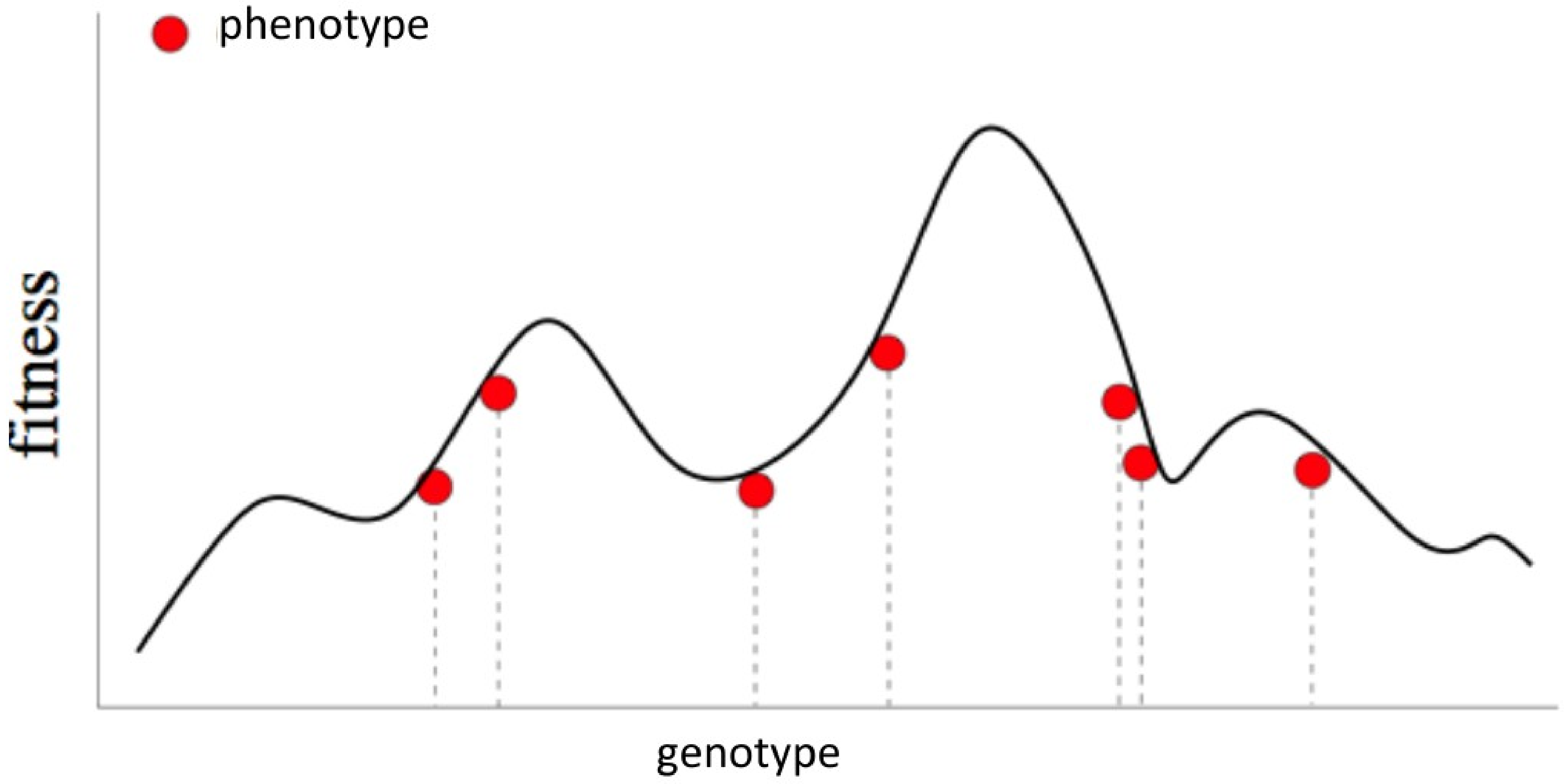
mutation rate = probability that an arbitrary bit of an arbitrary candidate will be mutated

GA: Behavior



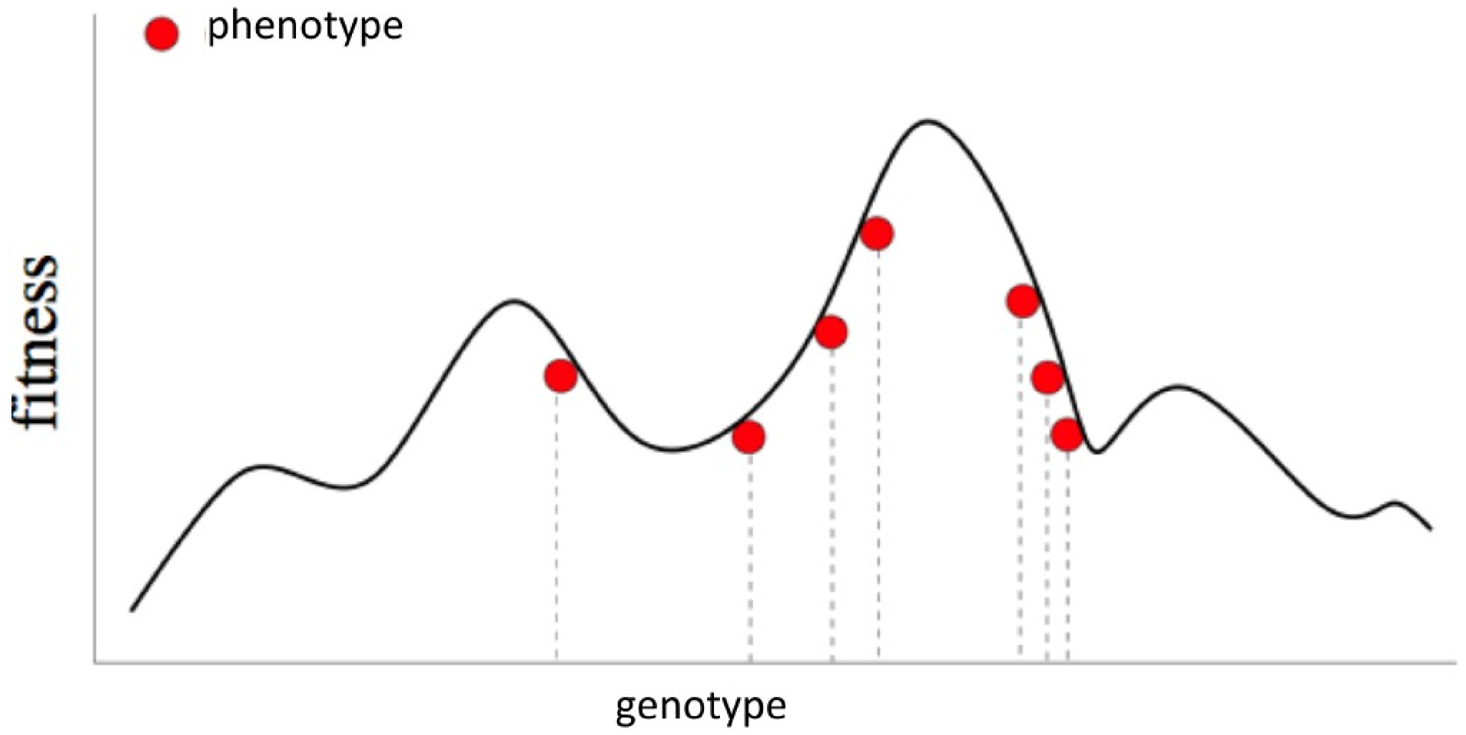
Initial population: candidate solutions are distributed throughout search space

GA: Behavior



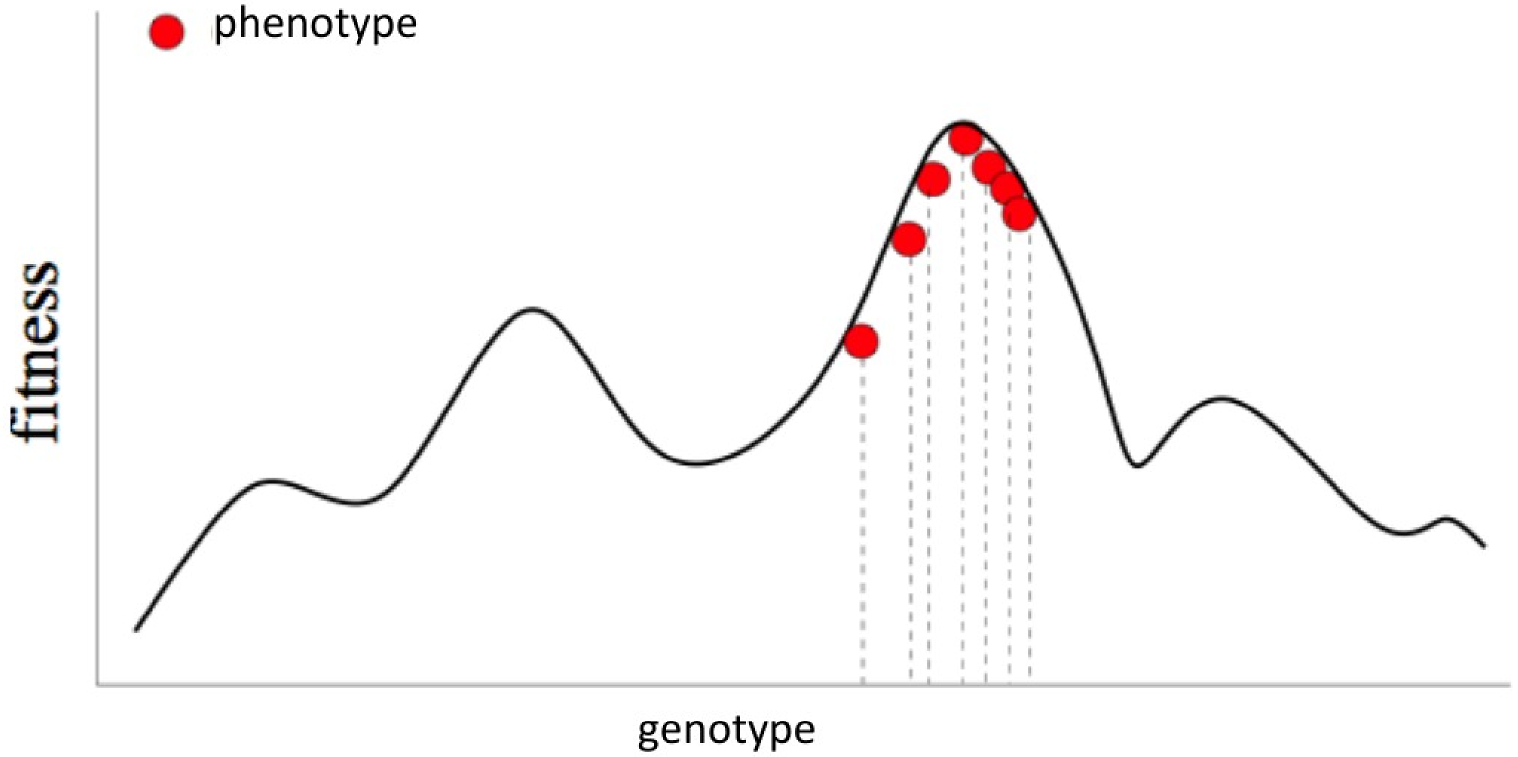
After some number of generations…

GA: Behavior



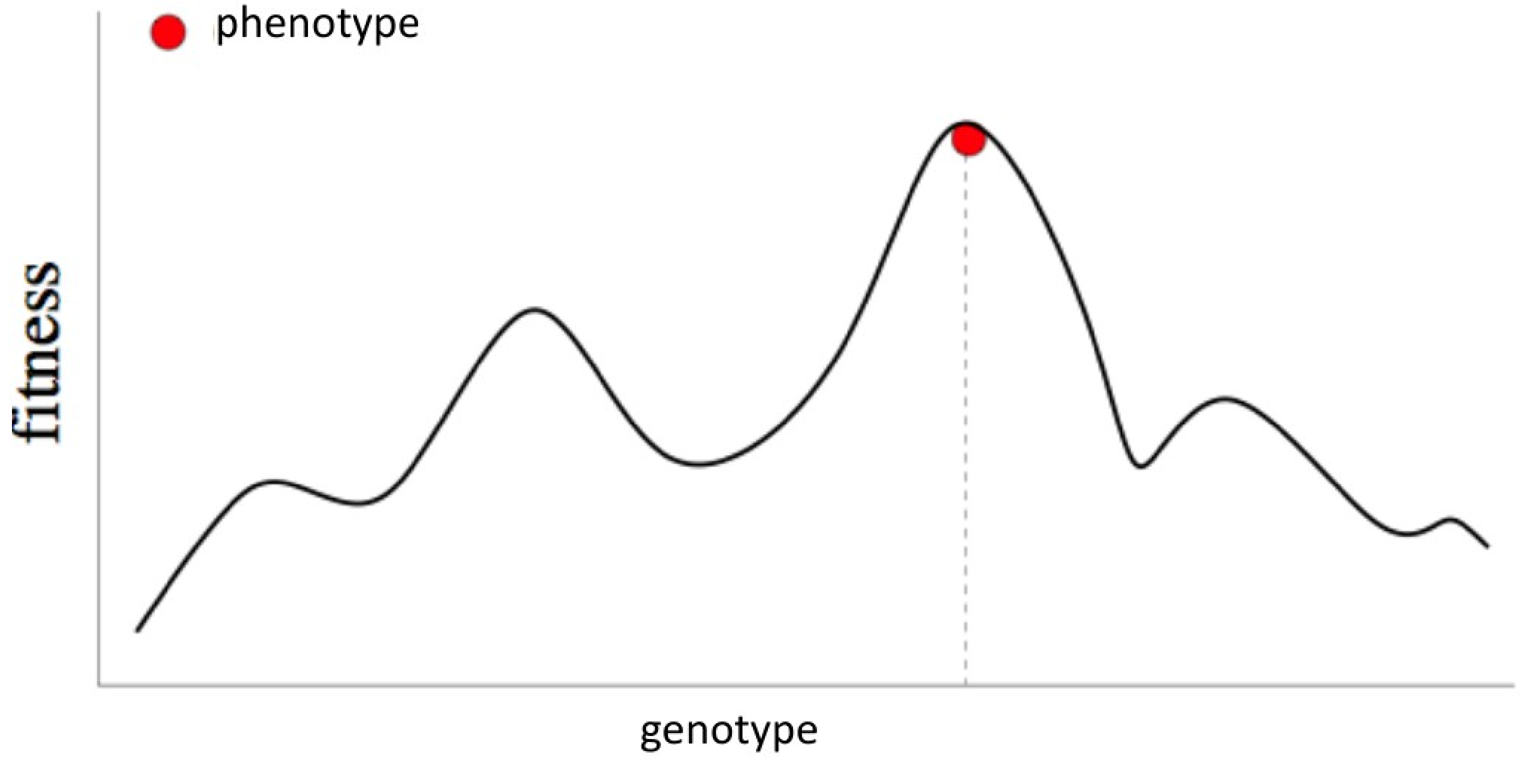
After some more generations…

GA: Behavior



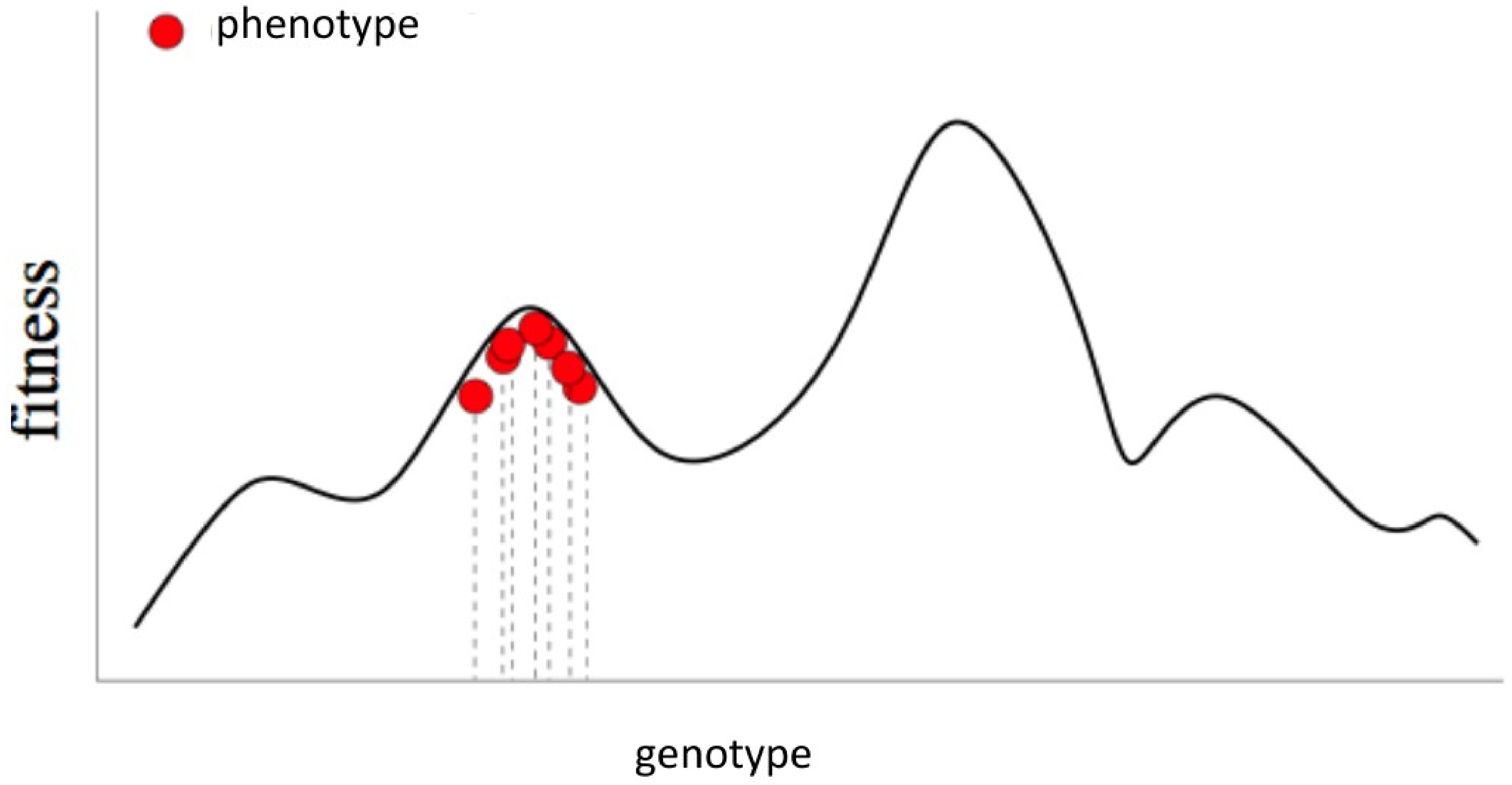
Some more generations…

GA: Behavior



All individuals look almost the same after many generations of recombination (crossover)

Premature Convergence



Population converges too quickly and misses global peak

How to avoid premature convergence?

- Use larger population
  - More genetic material
  - Takes longer to converge, requires more evaluations
- Reduce selective pressure
  - Make the search less greedy
  - Could take much longer
- Increase mutation
  - Adds diversity but could disrupt building blocks
- Niching
  - Force genotypes stay spread out into niches by penalizing fitness when bunched together

Recap: Advantages

Very general fitness function

- little training supervision required
- usually one uses derivative-free optimization: we do not need how to compute a derivative of the objective function

- Powerful way to search solutions in complex/unexplored problem spaces
  - no need for full mathematical description of problem space
  - some heuristics required
- Can cope with high dimensionality, local minima, etc.
- Straightforward parallelization

we'll do limitations later

Weaknesses of Evolutionary Computation

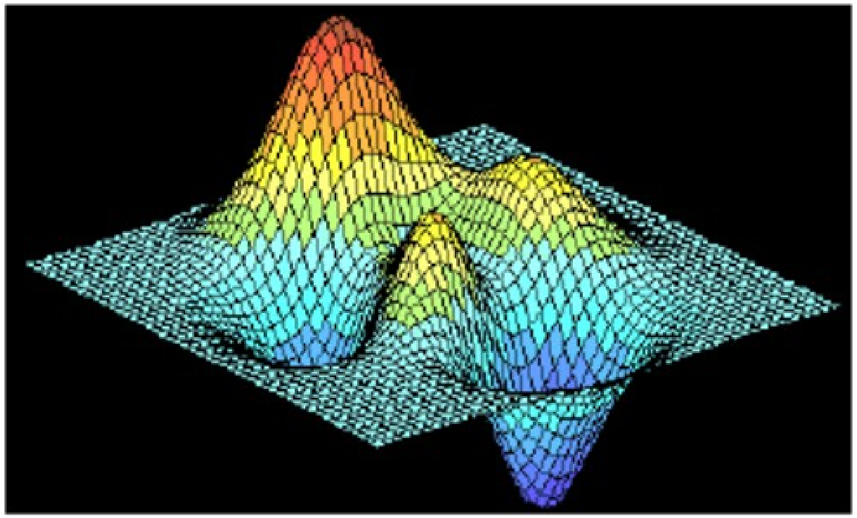
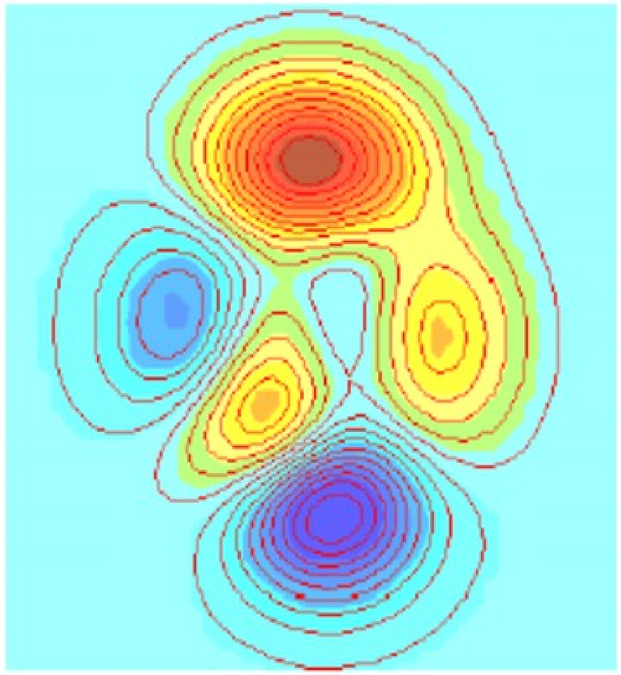
We have learned enough to cover the weaknesses of the method:

- Repeated evaluation of fitness function is costly
- EC does not scale well with complexity
- Avoiding local minima possible, but needs good tuning
- Operating on dynamic data sets is difficult
- Cannot effectively solve problems where no fitness ranking exists

Examples!

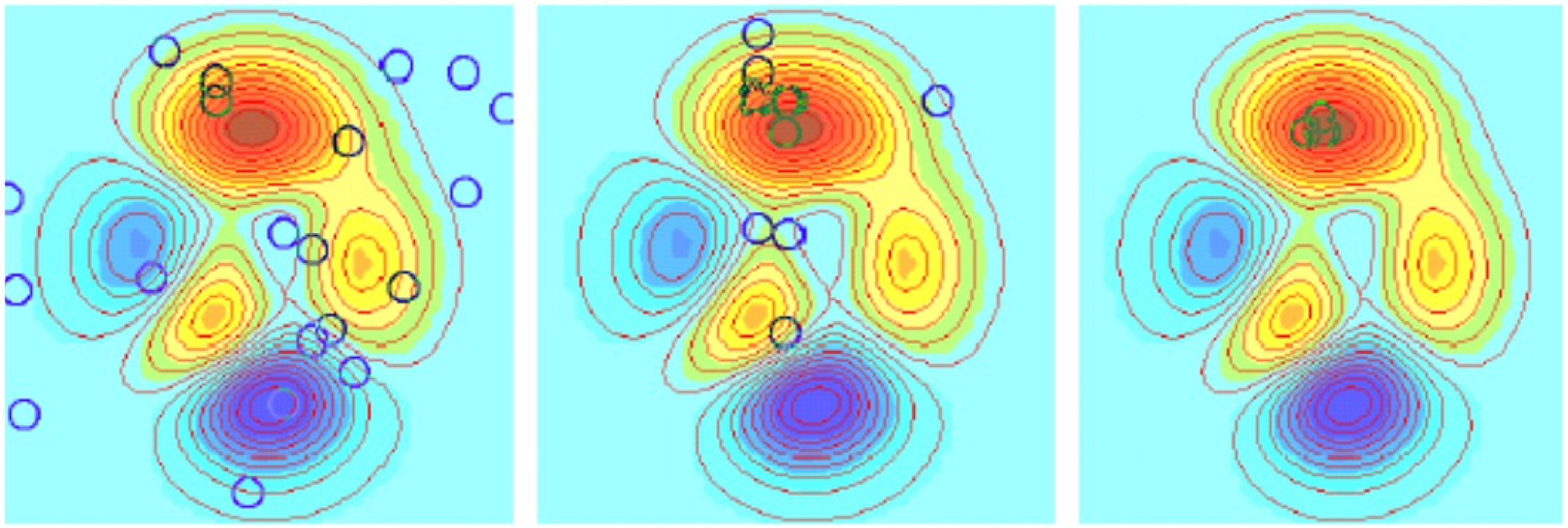
Function optimization

- z = f(x, y) = 3*(1-x)^2*exp(-(x^2) - (y+1)^2) - 10*(x/5 - x^3 - y^5)*exp(-x^2-y^2) -1/3*exp(-(x+1)^2 - y^2).

Function optimization

- GA process:



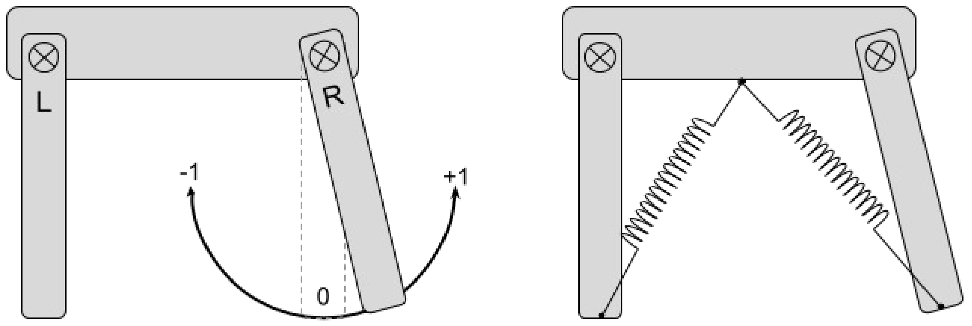
Initial population 5th generation 10th generation

see tutorials for details...

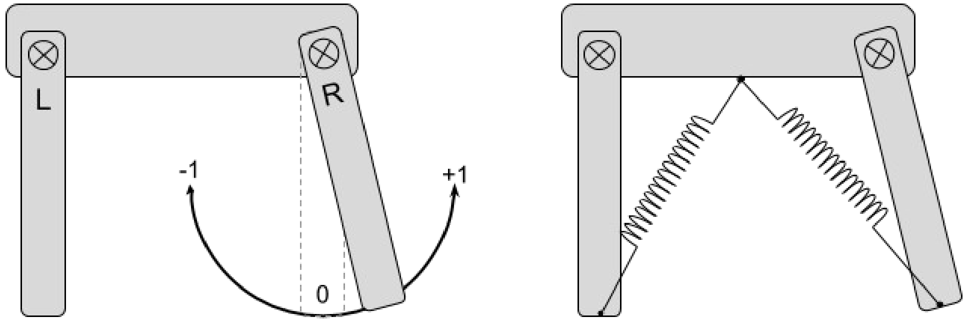
A Walking Robot

Example from:
https://www.alanzucconi.com/2016/04/06/evolutionary-coputation-1/

Goal: Create a simple walking robot



A Walking Robot



movement is performed by extending and contracting the springs

for simplicity, extension/contraction follow sinus wave

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

(a sine wave ranging from *m* to *M*, shifted by *o*, with period *p*)

A Walking Robot

How do we define a chromosome?

We have two legs and four parameters per leg, so a chromosome (completely encoding one candidate robot) has the form

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

Here we have a great example of the genotype being *very* different from the phenotype!

A Walking Robot

How to define the fitness function?

- just reward movement → robot drags

combined reward (i.e. extra reward for staying on feet?)

- careful balancing required

(follow the link & have a look at the videos)

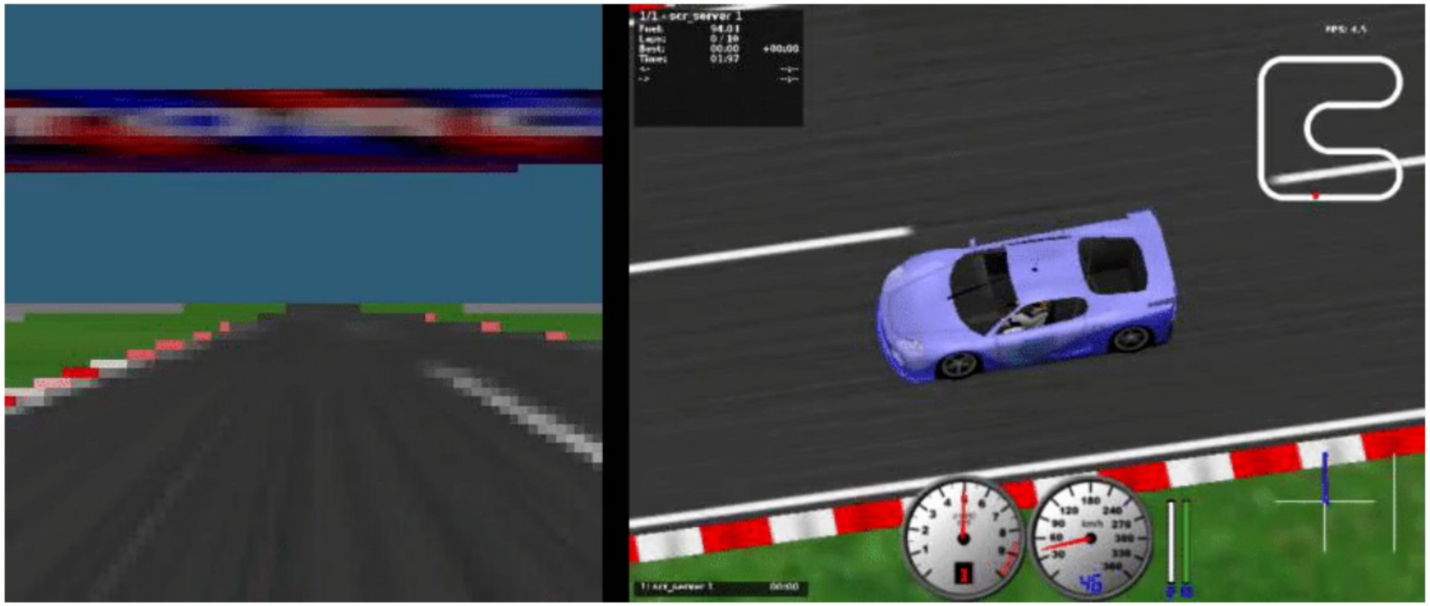
Mutation:

- by randomly perturbing one of the parameters of the sine function which controls the springs

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

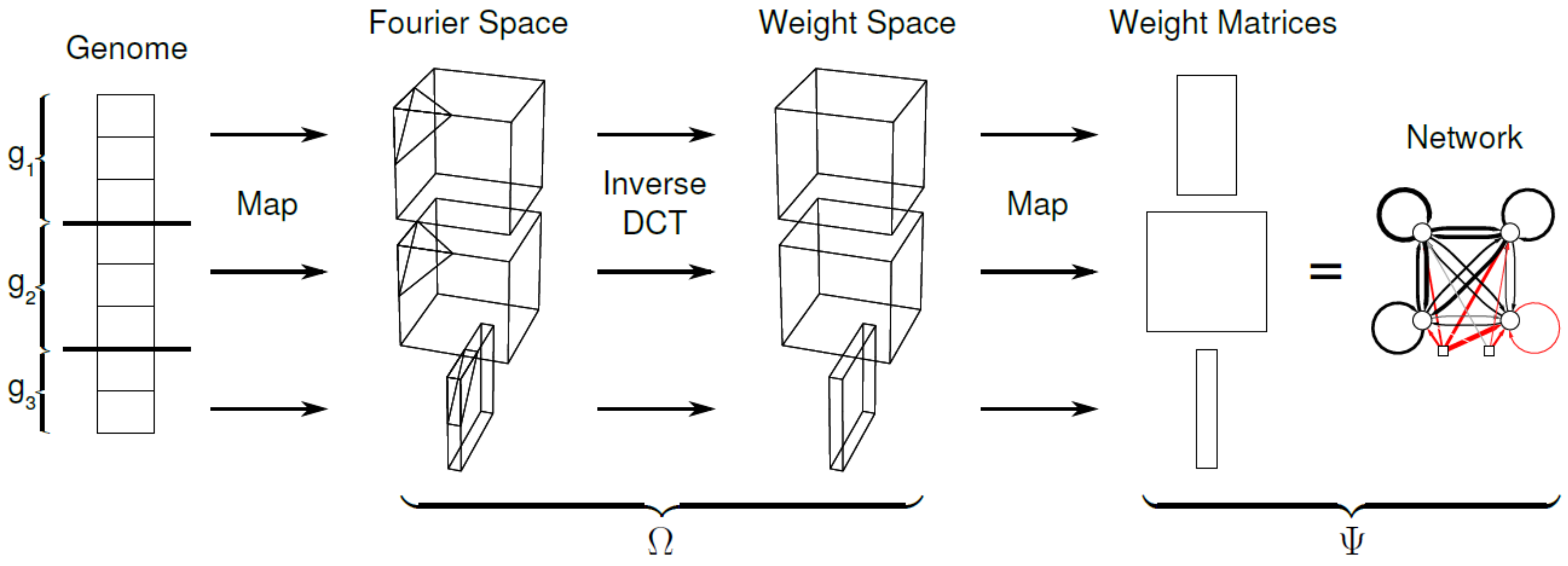

input images

[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.

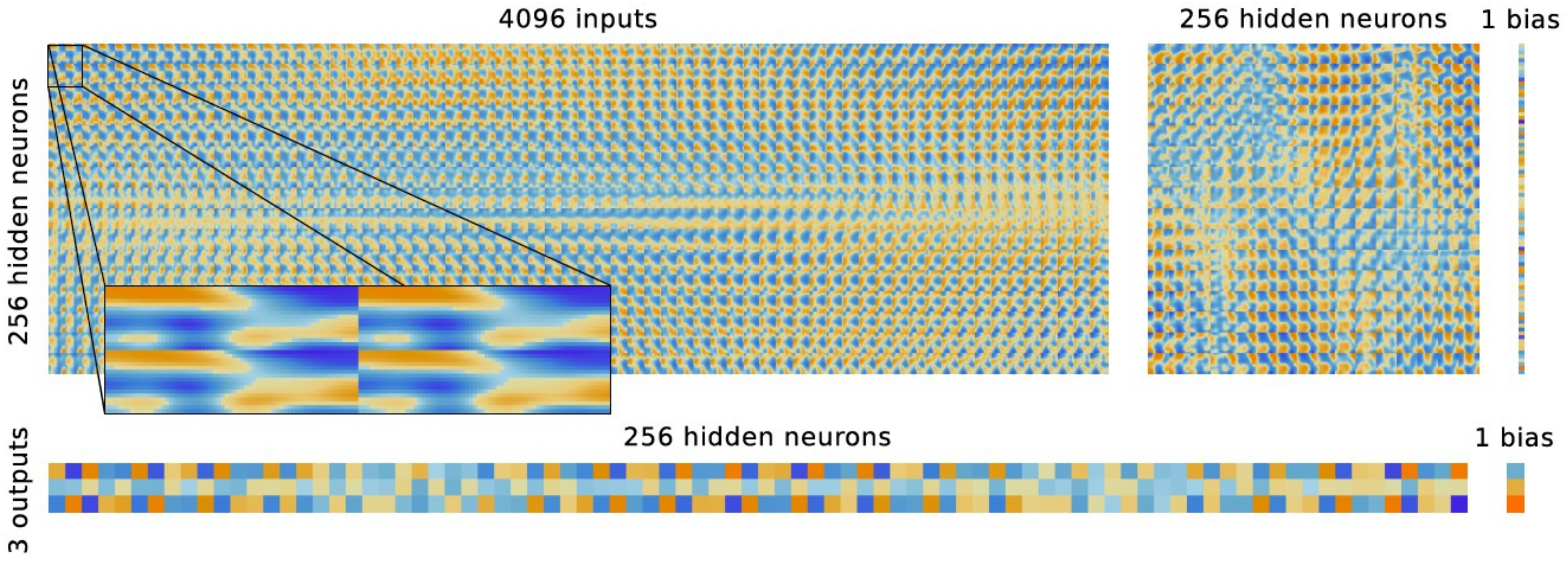


[Koutnik et al: Evolving Large-Scale Neural Networks for Vision-Based TORCS (2013)]

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

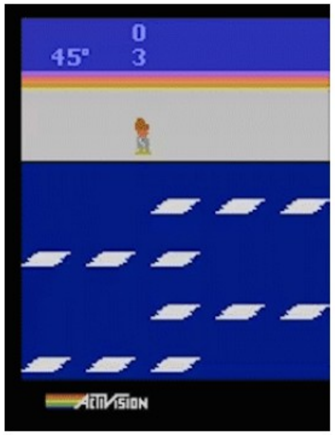


[Koutnik et al: Evolving Large-Scale Neural Networks for Vision-Based TORCS (2013)]

Genetic Algorithms: State-of-the-art example

[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

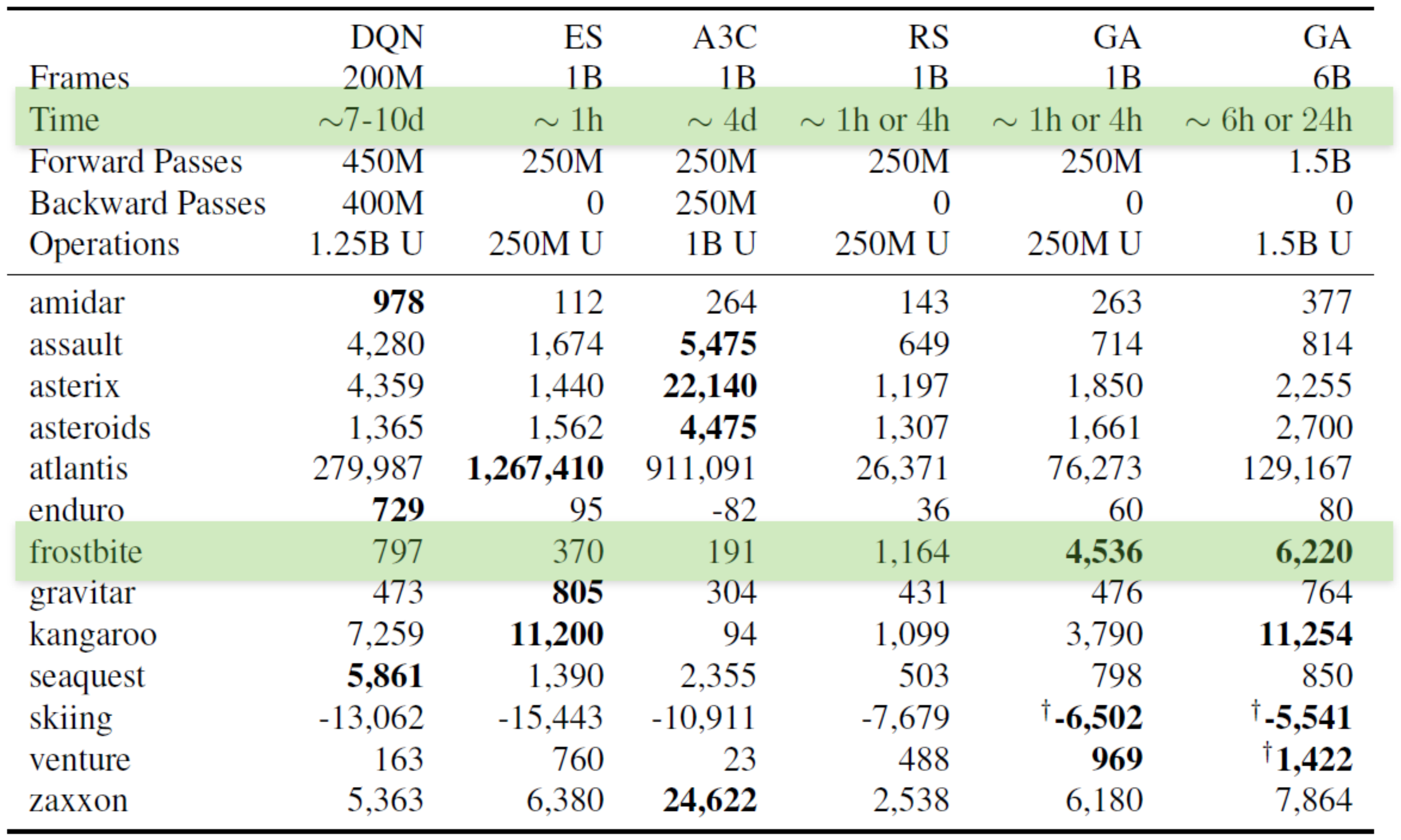


Algorithm 1: Conceptual description of GA used in [arXiv:1712.06567]

```
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
```

Genetic Algorithms: State-of-the-art example

[arXiv: 1712.06567]

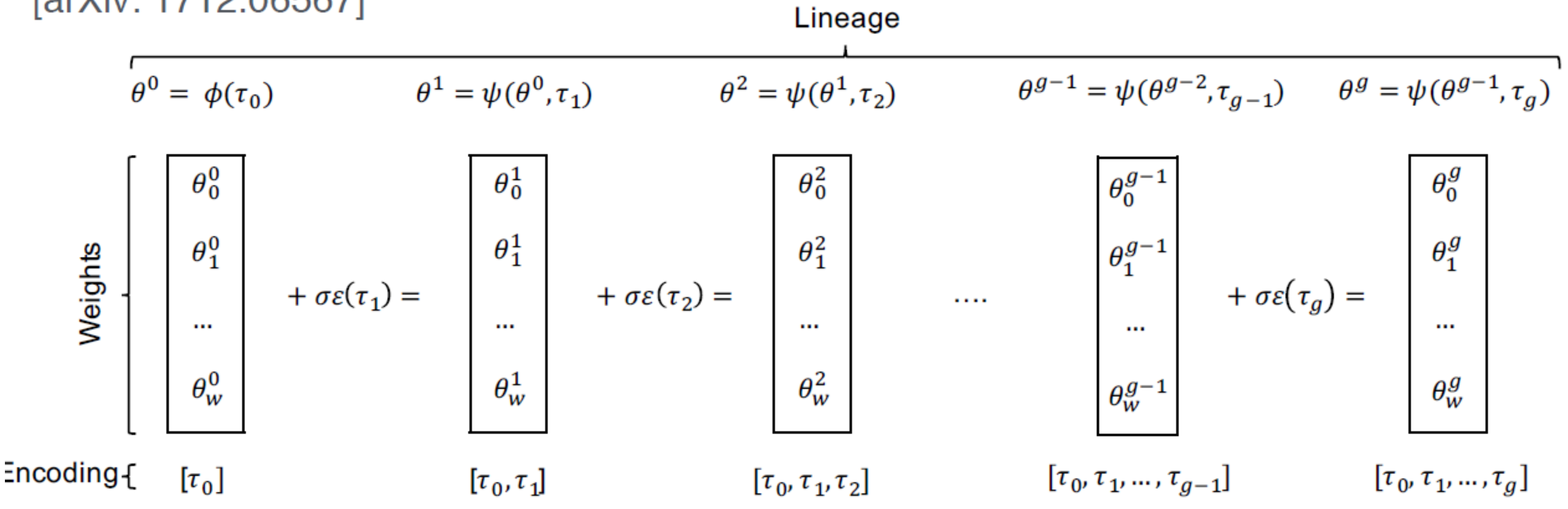


	DQN	ES	A3C	RS	GA	GA
Frames	200M	1B	1B	1B	1B	6B
Time	~7-10d	~ 1h	~ 4d	~ 1h or 4h	~ 1h or 4h	~ 6h or 24h
Forward Passes	450M	250M	250M	250M	250M	1.5B
Backward Passes	400M	0	250M	0	0	0
Operations	1.25B U	250M U	1B U	250M U	250M U	1.5B U
amidar	**978**	112	264	143	263	377
assault	4,280	1,674	**5,475**	649	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

Genetic Algorithms: State-of-the-art example

[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”