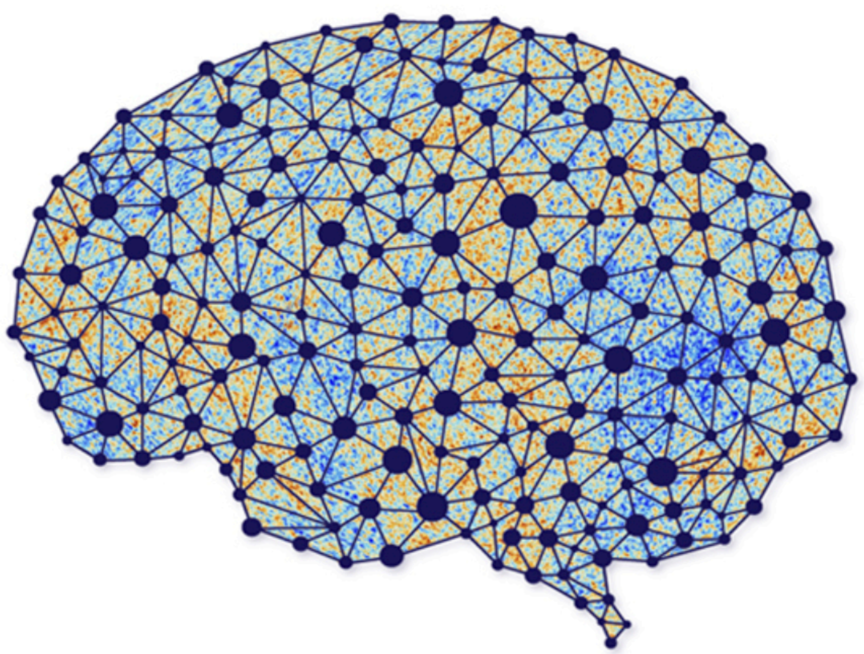
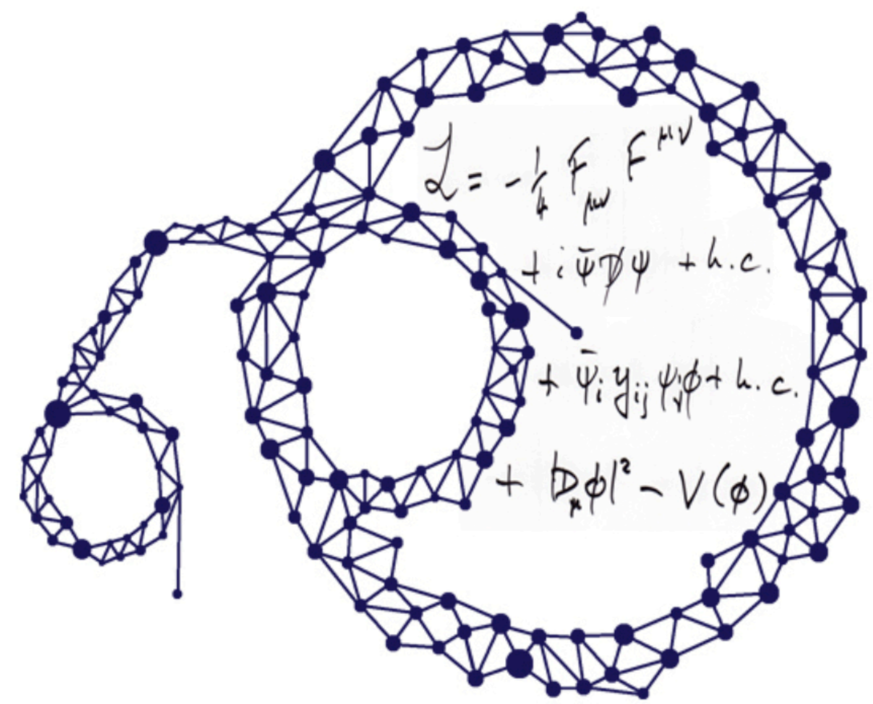
PHY 835: Machine Learning in Physics

Lecture 22: Reinforcement Learning Part 2

April 11, 2024



AI
∩
Universe



Gary Shiu

Fitted Q-learning

- Instead of tabulating the action-values (table size grows as $|S|^2|A|$ $|S|, |A|$ are sizes of state & action spaces), we can learn with a NN.
- Replace the action values $q[\mathbf{s}_t, a_t]$ by a ML model $q[\mathbf{s}_t, a_t, \phi]$.
- Loss function which measures consistency of adjacent action values:

$$L[\boldsymbol{\phi}] = \left(r[\mathbf{s}_t, a_t] + \gamma \cdot \max_a \Big[q[\mathbf{s}_{t+1}, a, \boldsymbol{\phi}] \Big] - q[\mathbf{s}_t, a_t, \boldsymbol{\phi}] \right)^2,$$

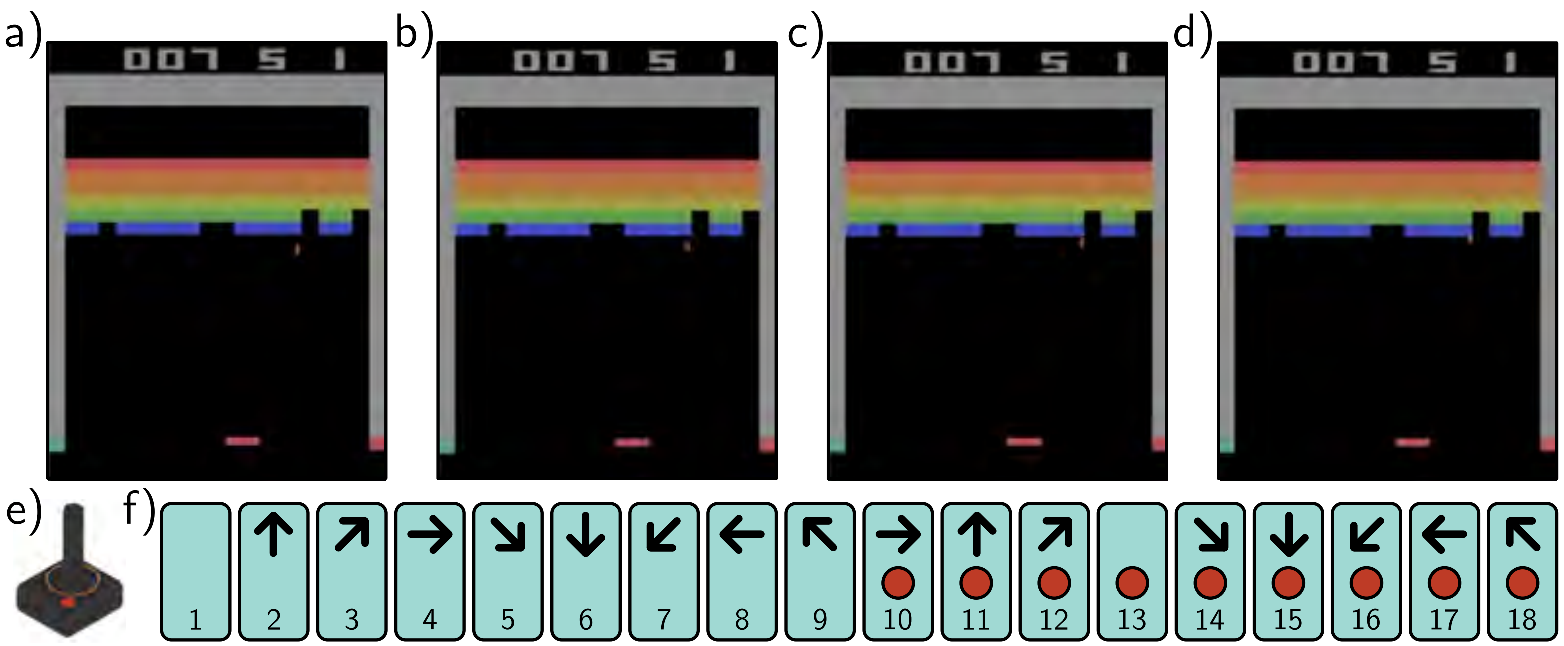
which in turn leads to an update:

$$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \alpha \left(r[\mathbf{s}_t, a_t] + \gamma \cdot \max_a \Big[q[\mathbf{s}_{t+1}, a, \boldsymbol{\phi}] \Big] - q[\mathbf{s}_t, a_t, \boldsymbol{\phi}] \right) \frac{\partial q[\mathbf{s}_t, a_t, \boldsymbol{\phi}]}{\partial \boldsymbol{\phi}}.$$

- Convergence is not guaranteed. A change to the parameters modifies both the target $r[\mathbf{s}_t, a_t] - \gamma \cdot \max_a q[\mathbf{s_{t+1}}, \mathbf{a}, \phi]$ & the prediction $q[\mathbf{s}_t, a_t, \phi]$.

Deep Q-networks

- Use deep NN for fitted Q-learning. Q stands for action-value $q[s_t, a_t, \phi]$.
- **Deep Q-network** was a RL architecture that exploited deep NN to learn to play ATARI 2600 games.

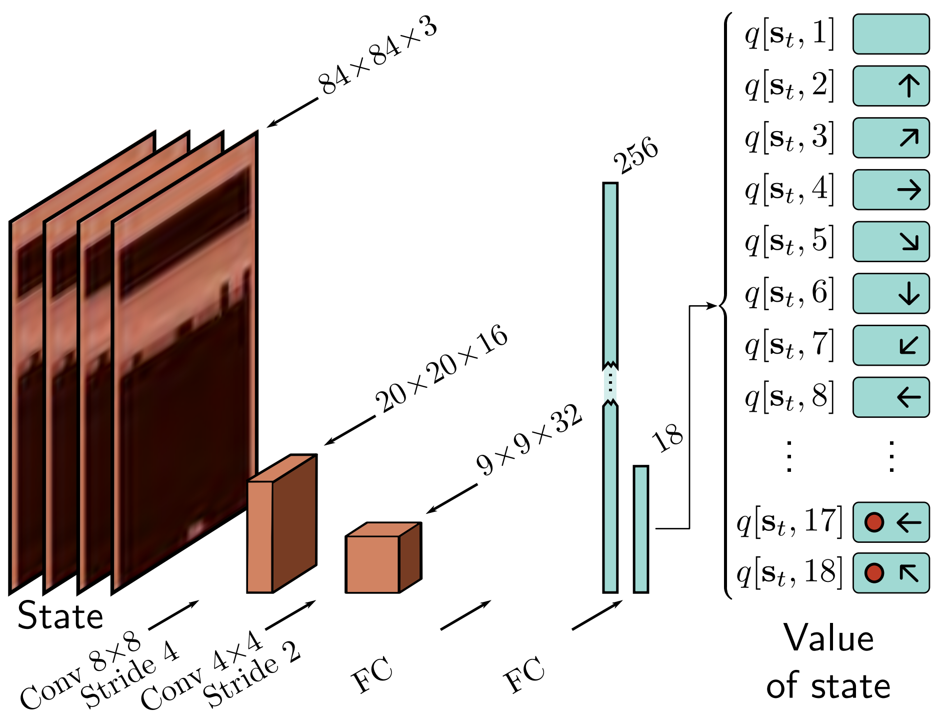


Single frame does not specify velocity ⇒ 4 adjacent frames to represent a state

18 possible actions (9 directions, on/off)

Deep Q-networks

- The data comprises 220 x 160 images with 128 possible colors at each pixel. Reshaped to 84 x 84 and only brightness value was kept.



Rewards ± 1 instead of raw scores of different games, can keep the same learning rate.

Experience replay: store recent states, action, and rewards in a buffer, reuses data samples many times.

- Issue of convergence was alleviated by fixing the target parameters to ϕ^- and only updating them periodically. Only update the prediction:

$$\phi \leftarrow \phi + \alpha\left(r[\mathbf{s}_t, a_t] + \gamma \cdot \max_a \left[q[\mathbf{s}_{t+1}, a, \phi^-]\right] - q[\mathbf{s}_t, a_t, \phi]\right) \frac{\partial q[\mathbf{s}_t, a_t, \phi]}{\partial \phi}.$$

- Not chasing a moving target; less prone to oscillations.

Policy gradient methods

- Recall the notions of value estimation vs policy estimation. Q-learning is an example of value estimation: estimate $q[s_t, a_t, \phi]$ and update π.
- **Policy-based methods** directly learn a stochastic policy $\pi[a_t | s_t, \theta]$.
- For MDP, there is always an optimal deterministic policy.
- There are reasons to use instead a stochastic policy:
    - **Exploration of the action-state space:** not obliged to take the best action at each step.
    - **Loss function changes smoothly:** can use gradient descent.
    - **Knowledge of the state is often incomplete:** two locations may look locally the same but nearby reward structure is different. Stochastic policy: taking different actions until ambiguity resolved.

Gradient update

- Consider a trajectory $\tau = [\mathbf{s}_1, a_1, \mathbf{s}_2, a_2, \ldots, \mathbf{s}_T, a_T]$ through an MDP.
- The probability of this trajectory depends on the current policy:

$$Pr(\boldsymbol{\tau}|\boldsymbol{\theta}) \quad = \quad Pr(\mathbf{s}_1)\prod_{t=1}^{T}\pi[a_t|\mathbf{s}_t,\boldsymbol{\theta}]Pr(\mathbf{s}_{t+1}|\mathbf{s}_t,a_t).$$

- Policy gradient algorithms aim to maximize the expected return $r[\tau]$ over many such trajectories:

$$\boldsymbol{\theta} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\Big[\mathbb{E}_{\boldsymbol{\tau}}\big[r[\boldsymbol{\tau}]\big]\Big] = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\Big[\int Pr(\boldsymbol{\tau}|\boldsymbol{\theta})r[\boldsymbol{\tau}]d\boldsymbol{\tau}\Big],$$

- The return is the sum of all the rewards received along the trajectory.
- To maximize the return, we use the gradient ascent update:

$$\begin{aligned}\boldsymbol{\theta} \quad &\leftarrow \quad \boldsymbol{\theta} + \alpha \cdot \frac{\partial}{\partial \boldsymbol{\theta}}\int Pr(\boldsymbol{\tau}|\boldsymbol{\theta})r[\boldsymbol{\tau}]d\boldsymbol{\tau} \\ &= \quad \boldsymbol{\theta} + \alpha \cdot \int \frac{\partial Pr(\boldsymbol{\tau}|\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} r[\boldsymbol{\tau}]d\boldsymbol{\tau}.\end{aligned}$$

α = learning rate

Gradient update

- $\approx$ this integral with a sum over empirically observed trajectories

$$\begin{aligned}\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta}+\alpha\cdot\int \frac{\partial Pr(\boldsymbol{\tau}|\boldsymbol{\theta})}{\partial\boldsymbol{\theta}} r[\boldsymbol{\tau}]d\boldsymbol{\tau}\\ &= \boldsymbol{\theta}+\alpha\cdot\int Pr(\boldsymbol{\tau}|\boldsymbol{\theta})\frac{1}{Pr(\boldsymbol{\tau}|\boldsymbol{\theta})}\frac{\partial Pr(\boldsymbol{\tau}|\boldsymbol{\theta})}{\partial\boldsymbol{\theta}} r[\boldsymbol{\tau}]d\boldsymbol{\tau}\\ &\approx \boldsymbol{\theta}+\alpha\cdot\frac{1}{I}\sum_{i=1}^{I}\frac{1}{Pr(\boldsymbol{\tau}_i|\boldsymbol{\theta})}\frac{\partial Pr(\boldsymbol{\tau}_i|\boldsymbol{\theta})}{\partial\boldsymbol{\theta}} r[\boldsymbol{\tau}_i].\end{aligned}$$

- Using identity involving log, we can simplify the update on θ:

$$\boldsymbol{\theta}\leftarrow\boldsymbol{\theta}+\alpha\cdot\frac{1}{I}\sum_{i=1}^{I}\frac{\partial\log\big[Pr(\boldsymbol{\tau}_i|\boldsymbol{\theta})\big]}{\partial\boldsymbol{\theta}} r[\boldsymbol{\tau}_i].$$

- The log probability is given by the sum of logs:

$$\begin{aligned}\log[Pr(\boldsymbol{\tau}|\boldsymbol{\theta})] &= \log\Big[Pr(\mathbf{s}_1)\prod_{t=1}^{T}\pi[a_t|\mathbf{s}_t,\boldsymbol{\theta}]Pr(\mathbf{s}_{t+1}|\mathbf{s}_t,a_t)\Big]\\ &= \log\big[Pr(\mathbf{s}_1)\big]+\sum_{t=1}^{T}\log\big[\pi[a_t|\mathbf{s}_t,\boldsymbol{\theta}]\big]+\sum_{t=1}^{T}\log\big[Pr(\mathbf{s}_{t+1}|\mathbf{s}_t,a_t)\big],\end{aligned}$$

Gradient update

- Only the policy $\pi[a_t \,|\, s_t, \theta]$ term depends on θ:

$$\boldsymbol{\theta} \;\leftarrow\; \boldsymbol{\theta} + \alpha \cdot \frac{1}{I}\sum_{i=1}^{I}\sum_{t=1}^{T}\frac{\partial \log\left[\pi[a_{it}|\mathbf{s}_{it},\boldsymbol{\theta}]\right]}{\partial \boldsymbol{\theta}} r[\boldsymbol{\tau}_i],$$

- Since the state evolution $Pr(\mathbf{s}_{t+1} \,|\, \mathbf{s}_t, a_t)$, parameter update does not assume Markov time evolution process.

- The total reward can be expressed as a sum of two contributions:

$$r[\boldsymbol{\tau}_i] = \sum_{t=1}^{T} r_{it} = \sum_{k=1}^{t-1} r_{ik} + \sum_{k=t}^{T} r_{ik},$$

- The first term does not affect the update, thus:

$$\boldsymbol{\theta} \;\leftarrow\; \boldsymbol{\theta} + \alpha \cdot \frac{1}{I}\sum_{i=1}^{I}\sum_{t=1}^{T}\frac{\partial \log\left[\pi[a_{it}|\mathbf{s}_{it},\boldsymbol{\theta}]\right]}{\partial \boldsymbol{\theta}} \sum_{k=t}^{T} r_{ik}.$$

REINFORCE Algorithm

- A policy gradient algorithm that incorporates discounting.
- Use Monte Carlo to generate episodes $[\mathbf{s}_{i1}, a_{i1}, r_{i2}, \mathbf{s}_{i2}, a_{i2}, r_{i3}, \ldots, r_{iT}]$ based on the current policy $\pi[a \mid s, \theta]$.
- $\pi[a \mid s, \theta]$ takes the current state & returns one output for each action.
- The outputs ($|A|$ dim.) are passed through a softmax function to create a distribution over actions, which is sampled at each time step.
- For each episode i, calculate the empirical discounted return for each trajectory τ_{it} that starts at time t:

$$r[\boldsymbol{\tau}_{it}] = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_{ik},$$

and then we update the parameters for each step t in each trajectory:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \cdot \gamma^{t} \frac{\partial \log\left[\pi_{a_{it}}[\mathbf{s}_{it}, \boldsymbol{\theta}]\right]}{\partial \boldsymbol{\theta}} r[\boldsymbol{\tau}_{it}] \qquad \forall\, i, t$$

Baselines

- Drawback of policy gradient methods: high variance; many episodes may be needed to get stable updates of the derivatives.
- To reduce the variance, we subtract the returns from a baseline:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \cdot \frac{1}{I}\sum_{i=1}^{I}\sum_{t=1}^{T}\frac{\partial \log\bigl[\pi_{a_{it}}[\mathbf{s}_{it},\boldsymbol{\theta}]\bigr]}{\partial \boldsymbol{\theta}}\left(r[\boldsymbol{\tau}_{it}] - b\right).$$

- The baseline is often taken to be:

$$b = \frac{1}{I}\sum_{i} r[\boldsymbol{\tau}_i].$$

- Subtracting this baseline factors out variance that might occur when the trajectories happen to pass through states with higher than average returns.

Actor-critic methods

- Actor-critic algorithms: temporal difference policy gradient algorithms.
- Parameters of the policy network are updated at each time step, in contrast with Monte Carlo REINFORCE algorithms.
- We do not have access to the future rewards along the trajectory.
- Approximate the sum over all the future rewards with:

$$\sum_{k=1}^{T} r[\boldsymbol{\tau}_{ik}] \approx r_{it} + \gamma \cdot v[\mathbf{s}_{i,t+1}, \boldsymbol{\phi}].$$

- The value $v[s_{i,t+1}, \phi]$ is estimated by a second NN with parameter ϕ.
- This gives the update:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \cdot \frac{1}{I} \sum_{i=1}^{I} \sum_{t=1}^{T} \frac{\partial \log\big[Pr(a_{it}|\mathbf{s}_{it}, \boldsymbol{\theta})\big]}{\partial \boldsymbol{\theta}} \Big(r_{it} + \gamma \cdot v[\mathbf{s}_{i,t+1}, \boldsymbol{\phi}] - v[\mathbf{s}_{i,t}, \boldsymbol{\phi}] \Big).$$

Actor-critic methods

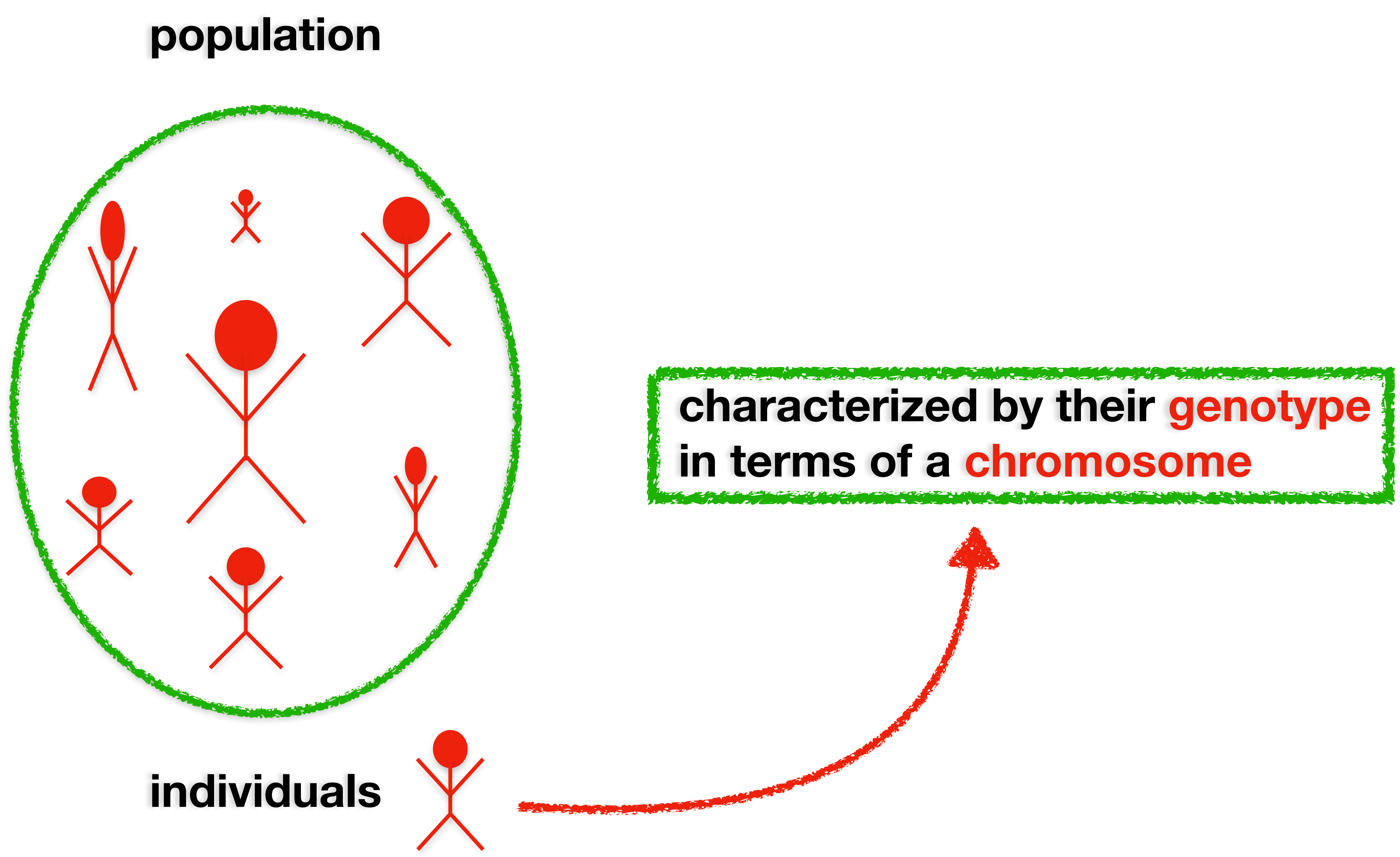
- Concurrently, we update the parameter ϕ using the loss function:

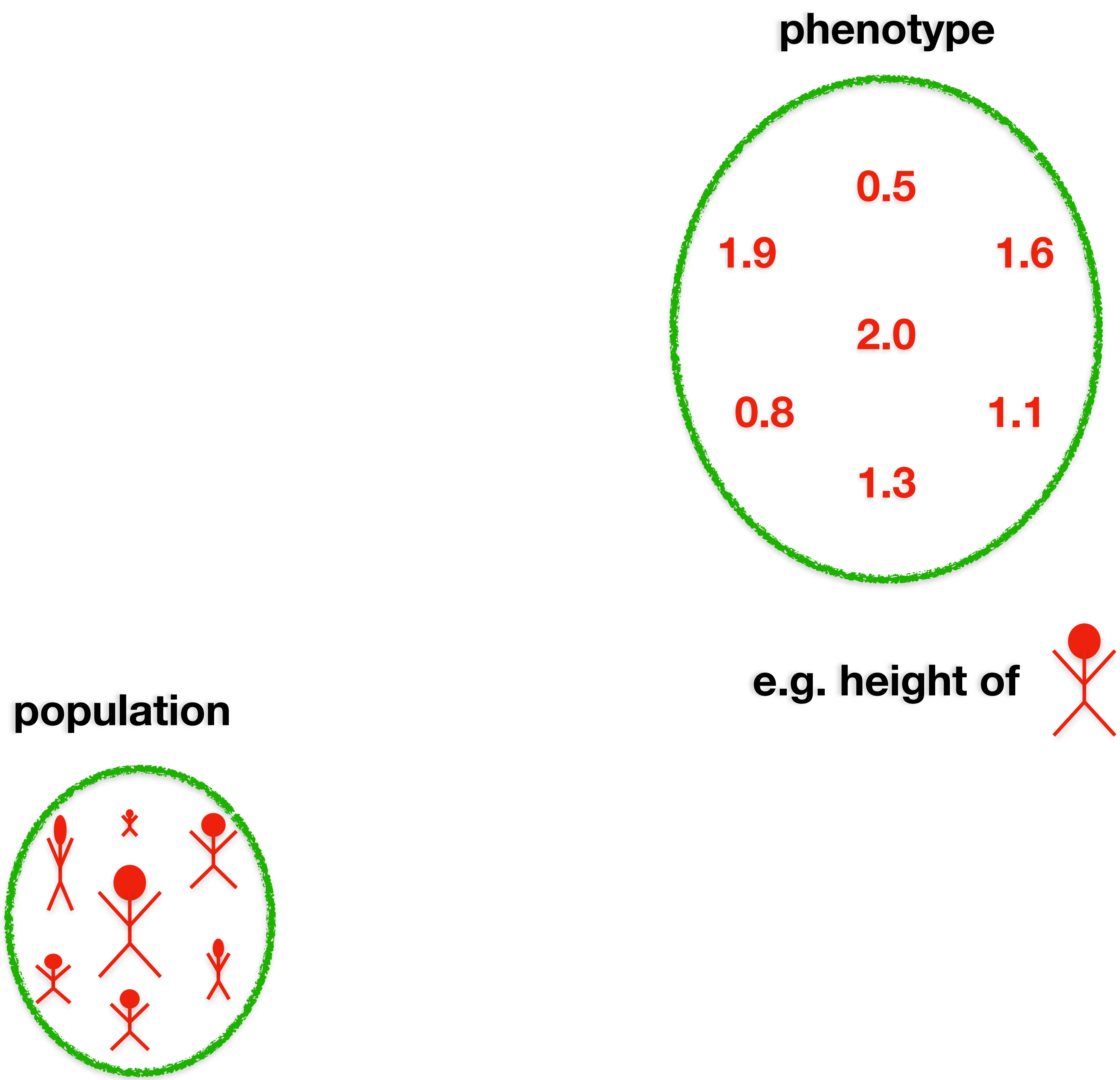
$$L[\phi] = \sum_{i=1}^{I} \sum_{t=1}^{T} \left(r_{it} + \gamma \cdot v[\mathbf{s}_{i,t+1}, \phi] - v[\mathbf{s}_{i,t}, \phi]\right)^2 .$$

- The policy network $\pi[\mathbf{s}_t, \theta]$ that predicts $Pr(a \mid \mathbf{s}_t)$ is term the **actor**.
- The value network $v[\mathbf{s}_t, \phi]$ is termed the **critic**.
- Actor-critic methods can update the policy parameter at each step.
- In practice, the agent typically collects a batch of experience over many time steps before the policy is updated.

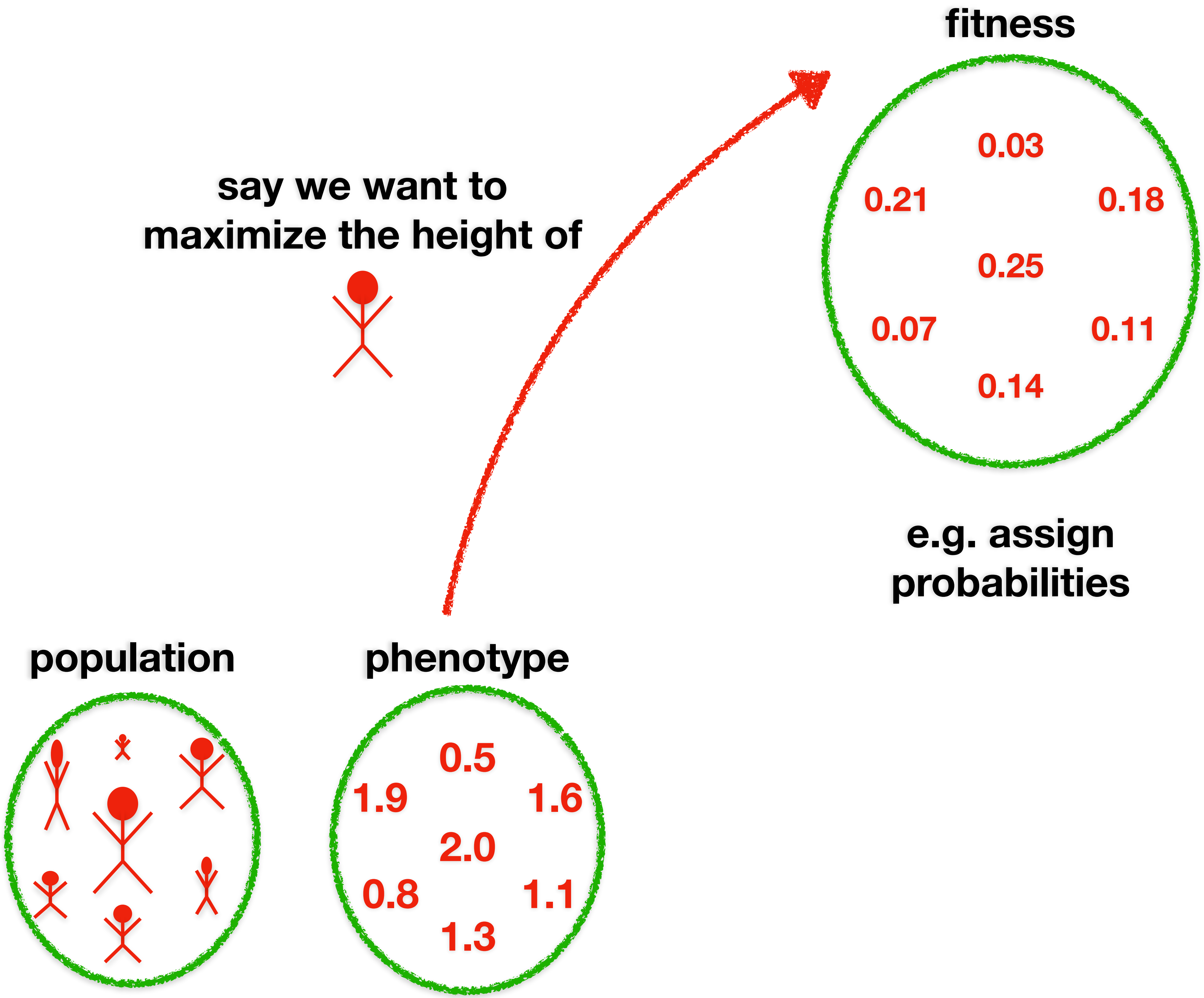
Nature or Nurture?

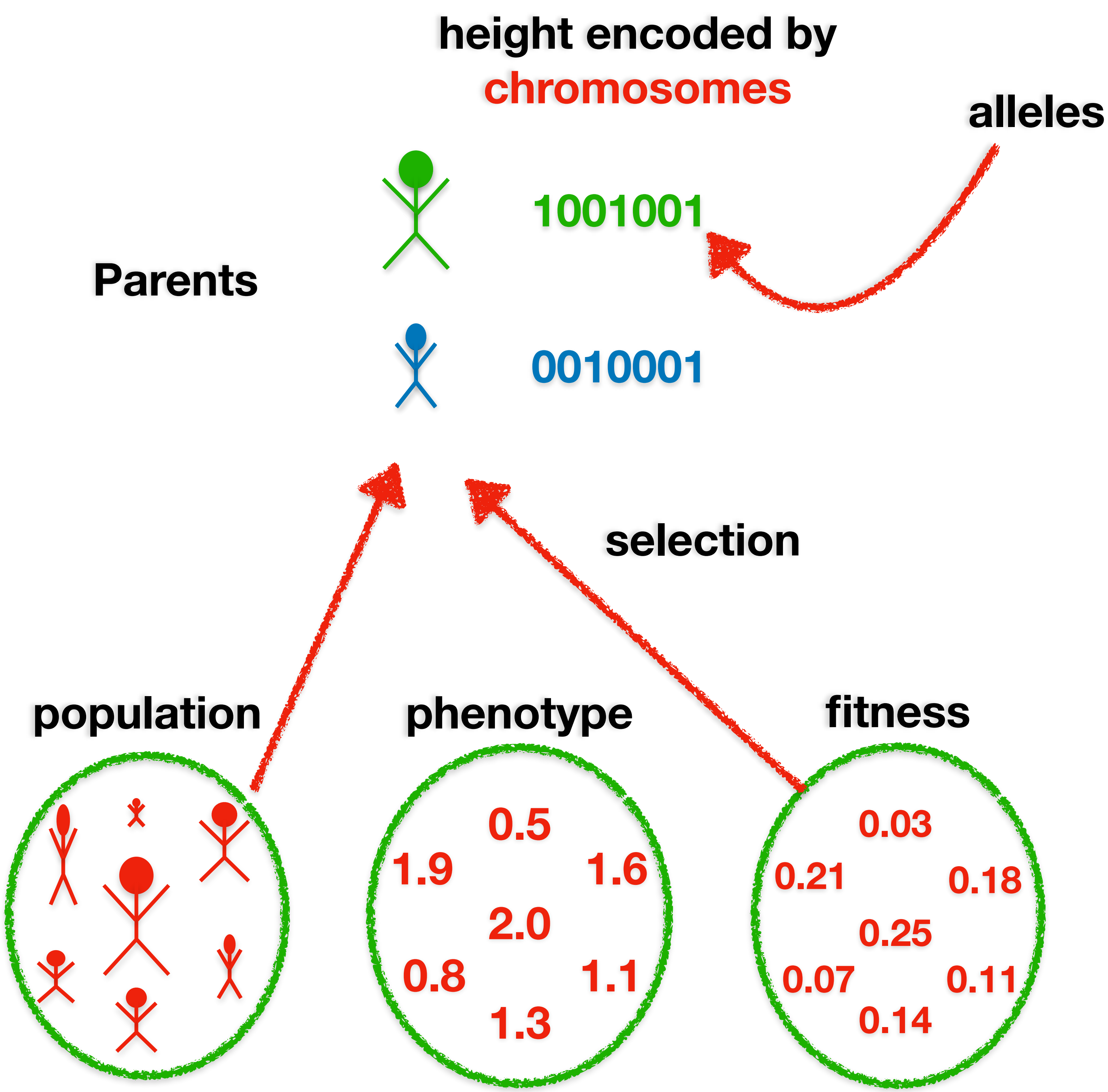
Genetic Algorithm: The basic Idea

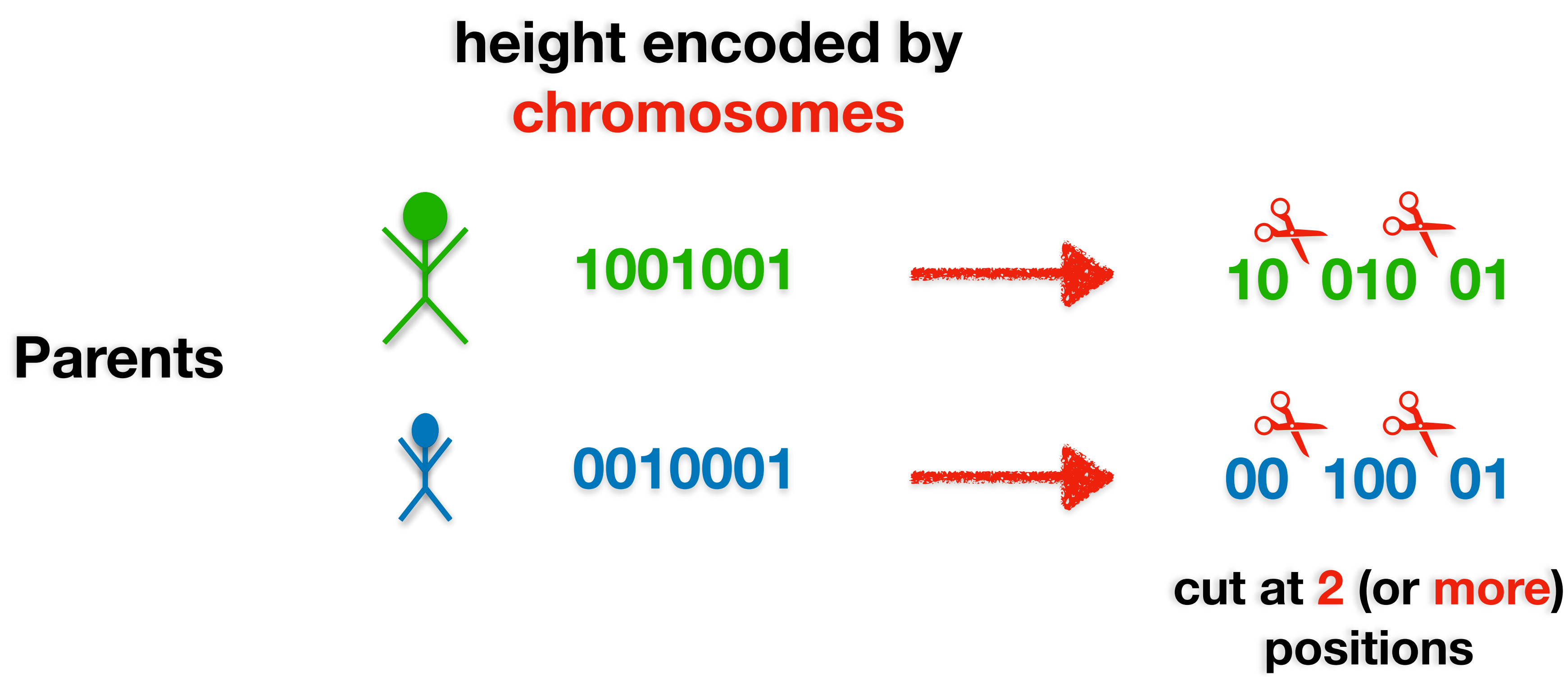


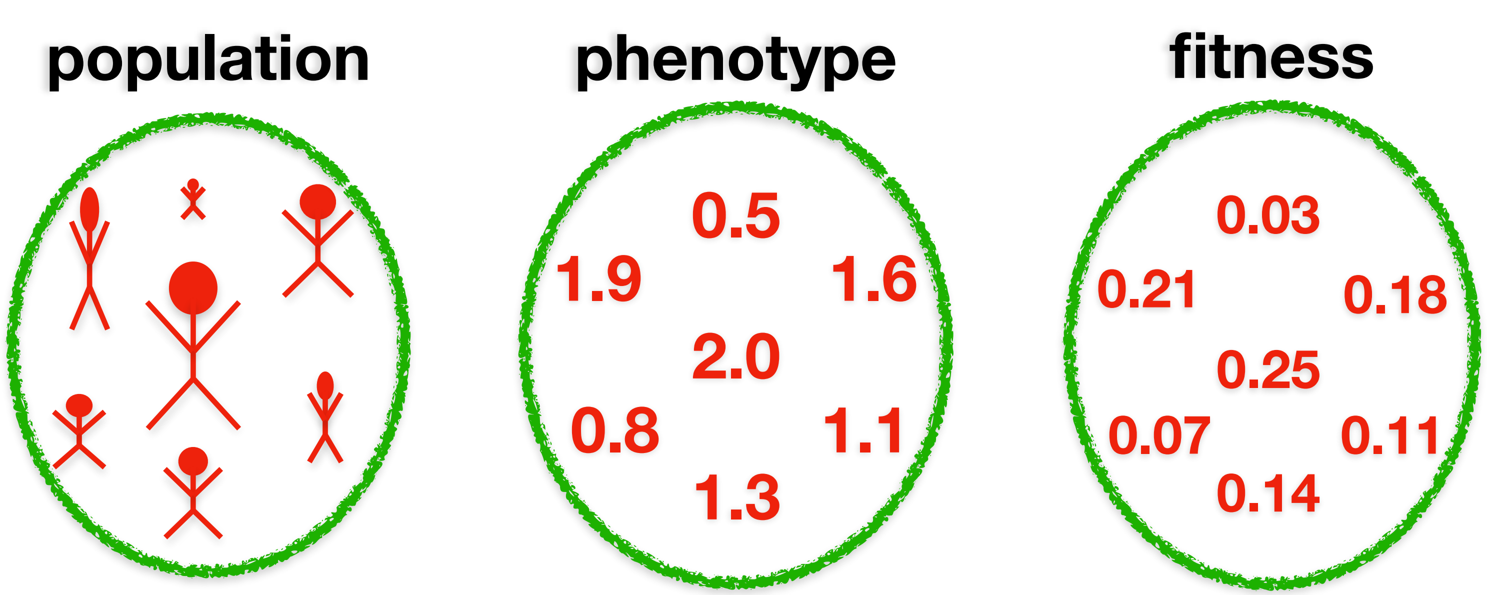


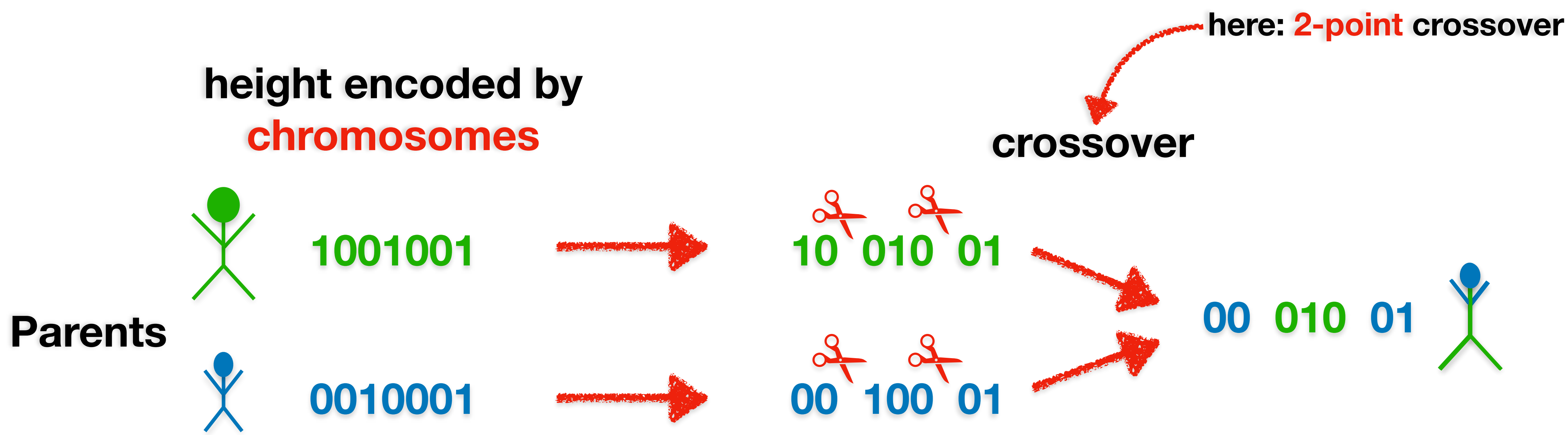
phenotype
0.5
1.9
1.6
2.0
0.8
1.1
1.3
e.g. height of
population

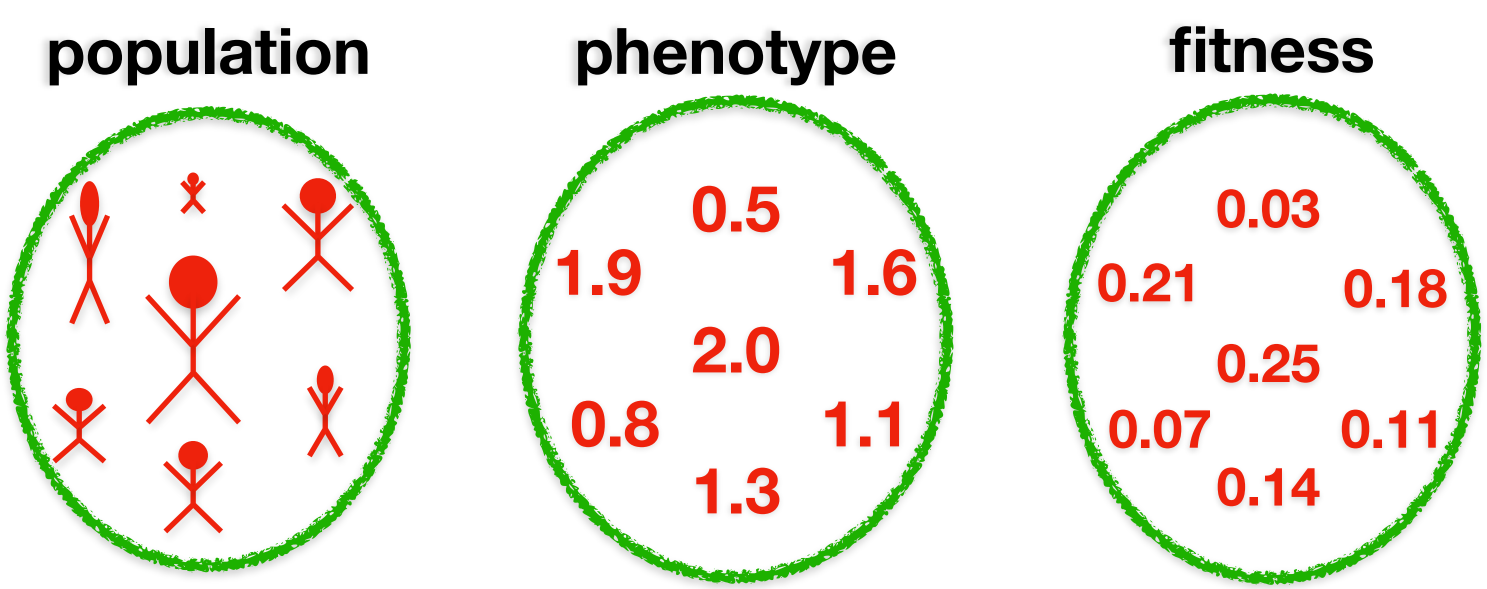

fitness
0.03
0.21
0.18
0.25
0.07
0.11
0.14
say we want to
maximize the height of
e.g. assign
probabilities
population
phenotype
0.5
1.9
1.6
2.0
0.8
1.1
1.3

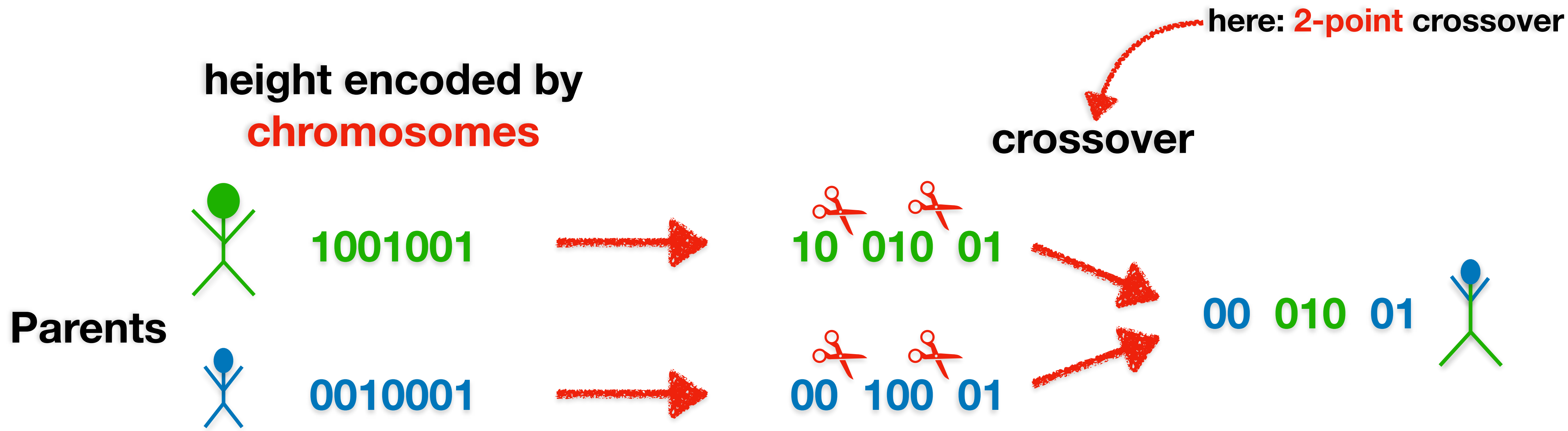

height encoded by
chromosomes
alleles
1001001
Parents
0010001
selection
population
phenotype
fitness
0.5
1.9
1.6
2.0
0.8
1.1
1.3
0.03
0.21
0.18
0.25
0.07
0.11
0.14

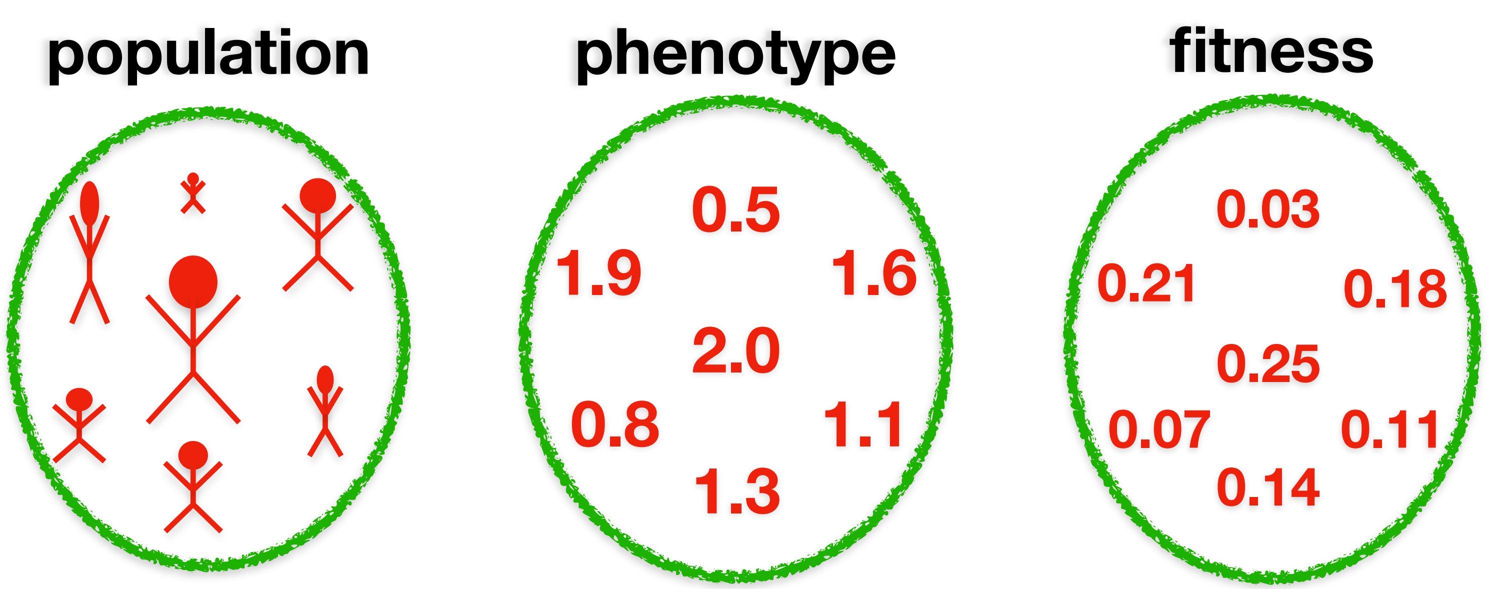

height encoded by chromosomes
Parents
1001001
0010001
10 010 01
00 100 01
cut at 2 (or more) positions

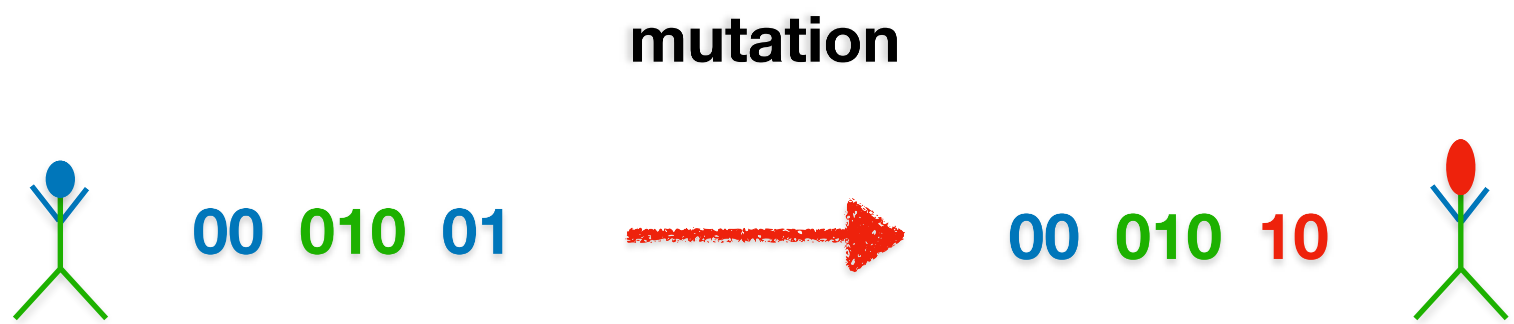

population
phenotype
0.5
1.9
1.6
2.0
0.8
1.1
1.3
fitness
0.03
0.21
0.18
0.25
0.07
0.11
0.14

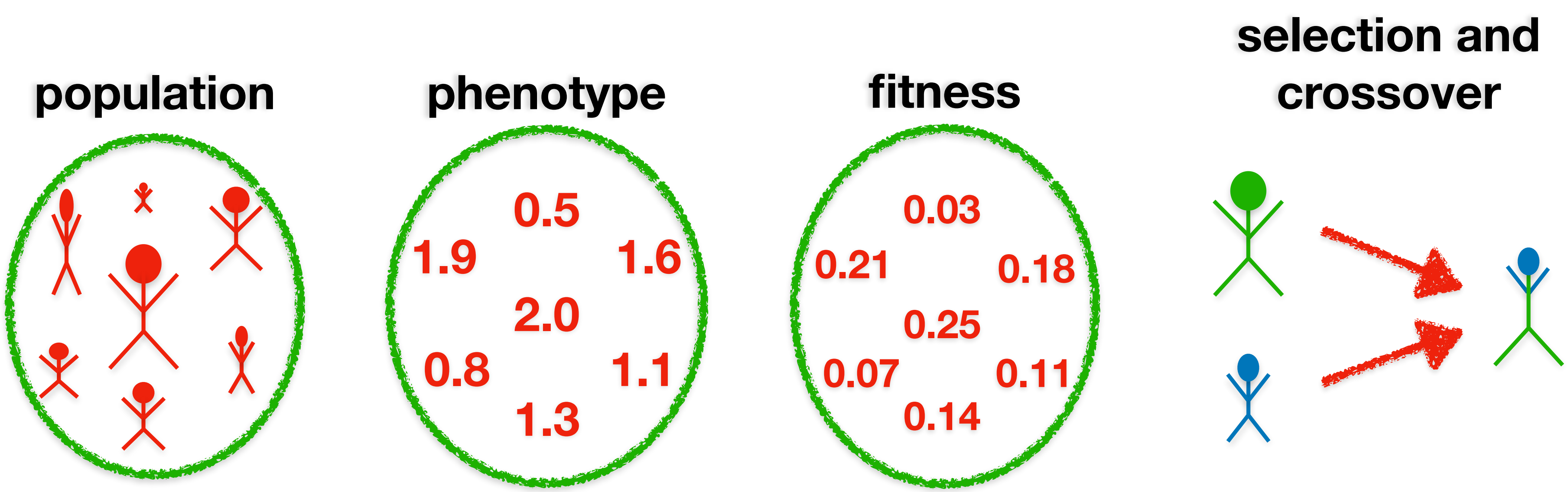

here: 2-point crossover
height encoded by chromosomes
crossover
Parents
1001001
0010001
10 010 01
00 100 01
00 010 01


population
phenotype
0.5
1.9
1.6
2.0
0.8
1.1
1.3
fitness
0.03
0.21
0.18
0.25
0.07
0.11
0.14

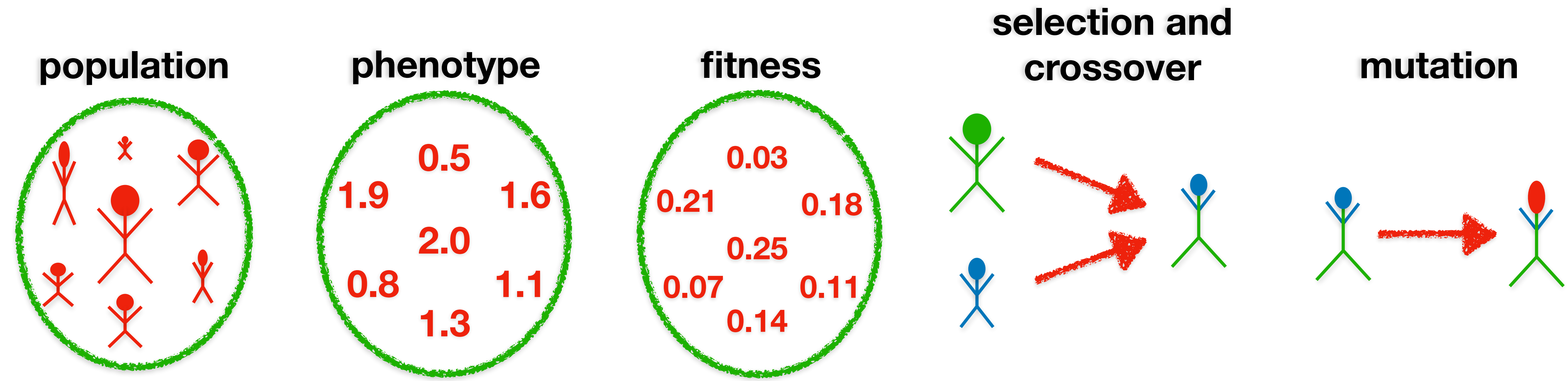

here: 2-point crossover
height encoded by chromosomes
crossover
Parents
1001001
0010001
10 010 01
00 100 01
00 010 01

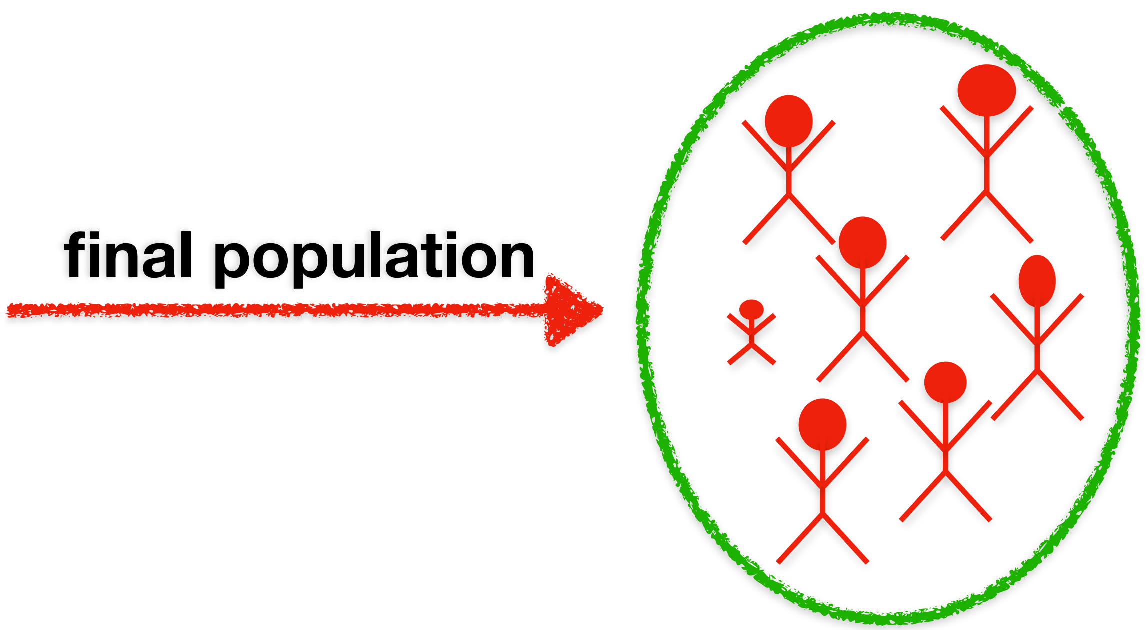

population
phenotype
0.5
1.9
1.6
2.0
0.8
1.1
1.3
fitness
0.03
0.21
0.18
0.25
0.07
0.11
0.14


mutation
00 010 01
00 010 10


population
phenotype
0.5
1.9
1.6
2.0
0.8
1.1
1.3
fitness
0.03
0.21
0.18
0.25
0.07
0.11
0.14
selection and crossover


define new population and repeat
population
phenotype
0.5
1.9
1.6
2.0
0.8
1.1
1.3
fitness
0.03
0.21
0.18
0.25
0.07
0.11
0.14
selection and crossover
mutation

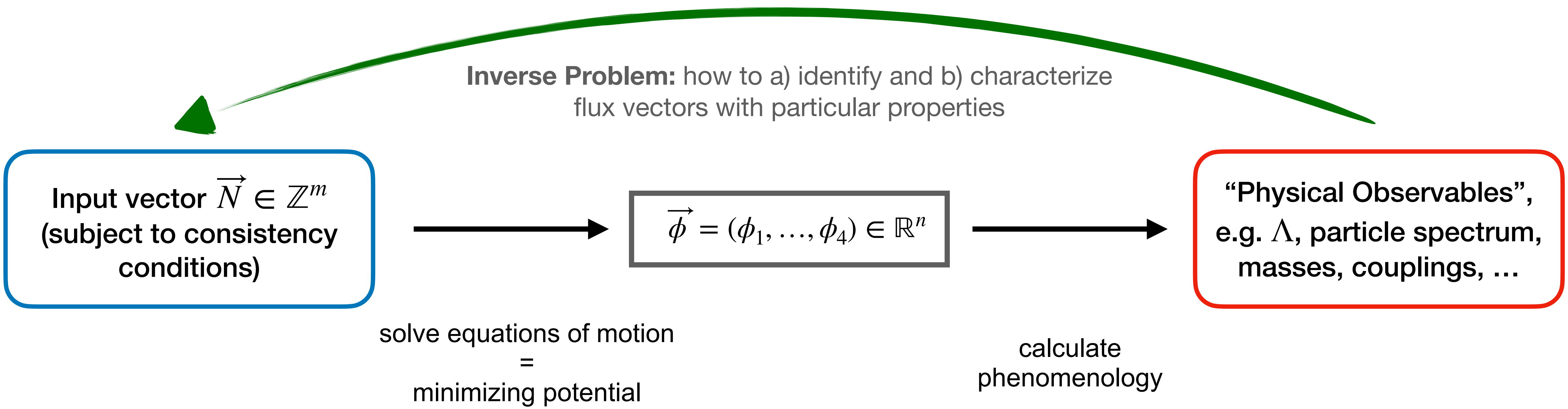




Various choices to be made:

- definition of fitness function
- selection method
- crossover procedure
- mutation rate
- further attributes

The String Landscape Inverse Problem



$\vec{N} =$ (topologies of compactification, number of branes and wrapping numbers, quantized fluxes, ….)

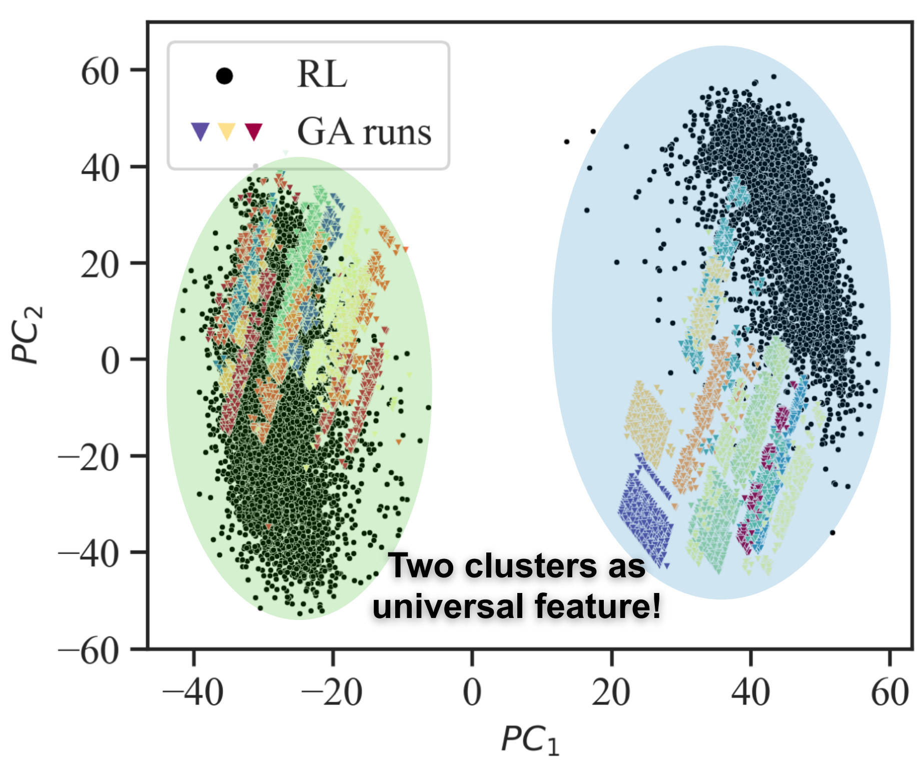
Understanding the structure of the Landscape

Task

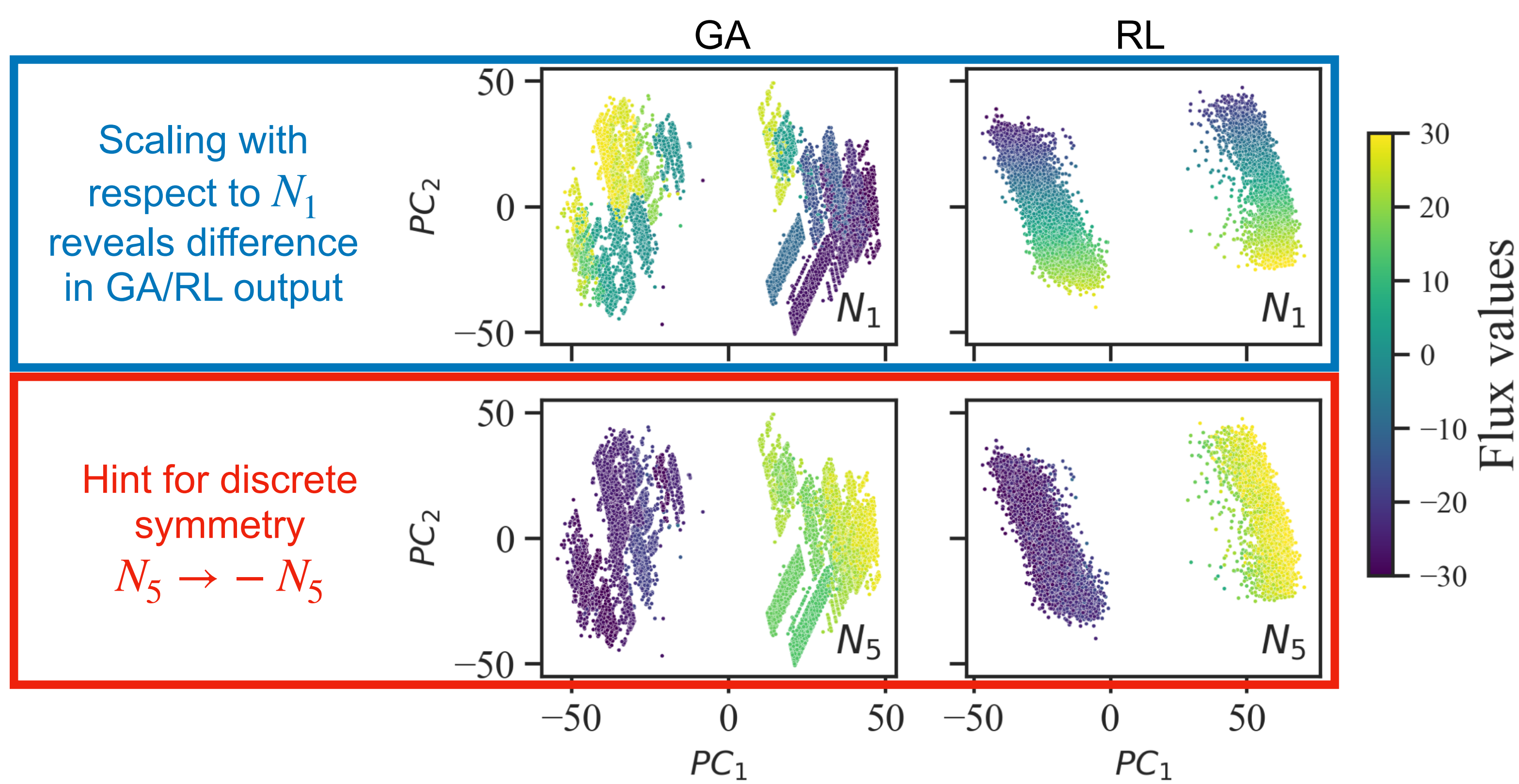
Apply GA+RL to find string vacua with $|W_0| = 50{,}000 \pm 1000$

We performed a **Principal Component Analysis (PCA)** on the output of flux vectors in $\mathbb{Z}^8$

PCA on combined output



PCA on individual output

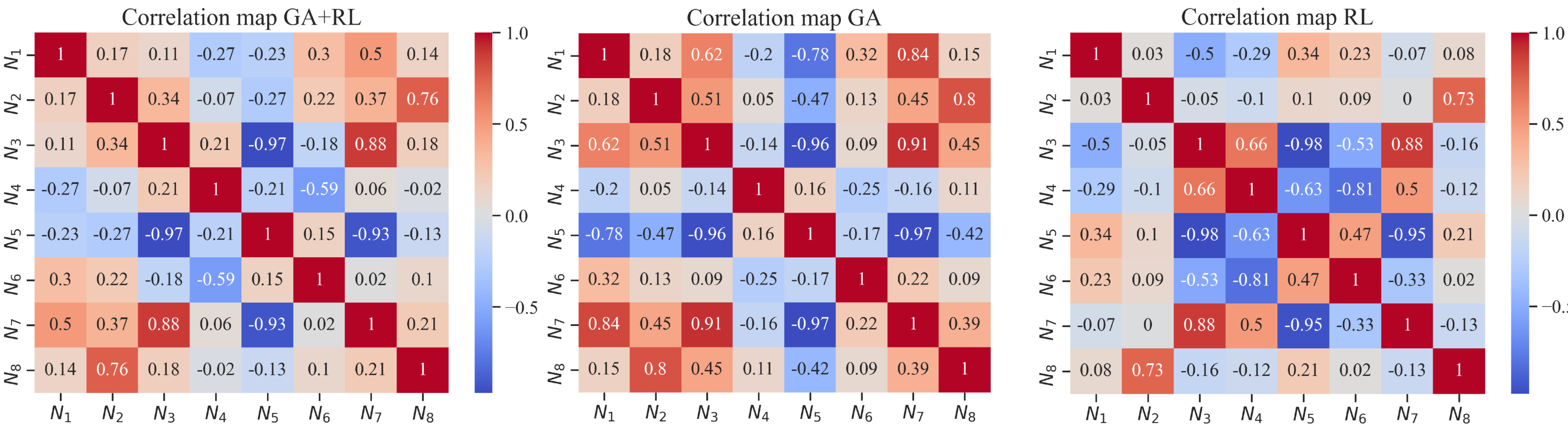


https://arxiv.org/abs/1907.10072

https://arxiv.org/abs/2111.11466

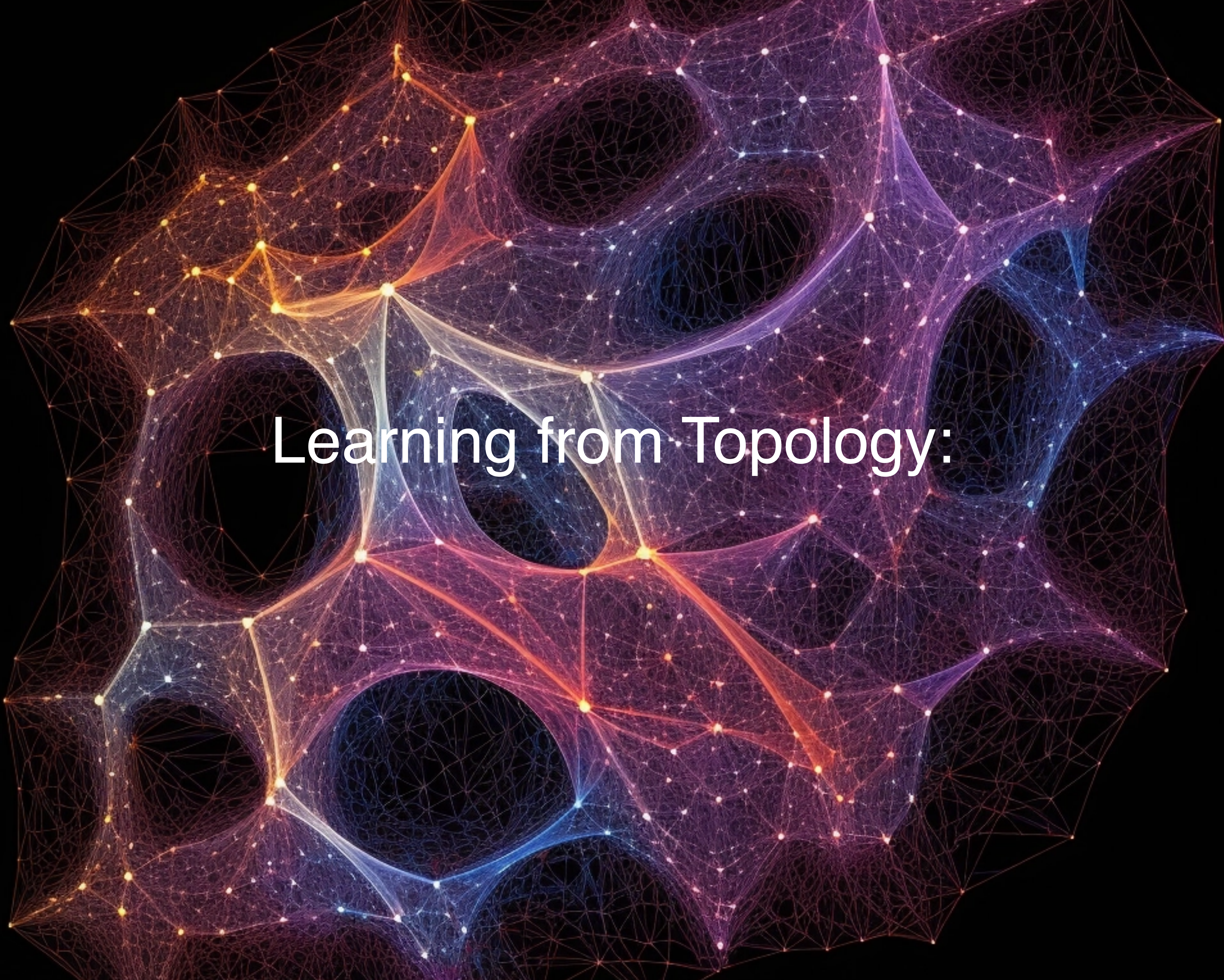
Genetic Algorithms + RL

- Correlations:



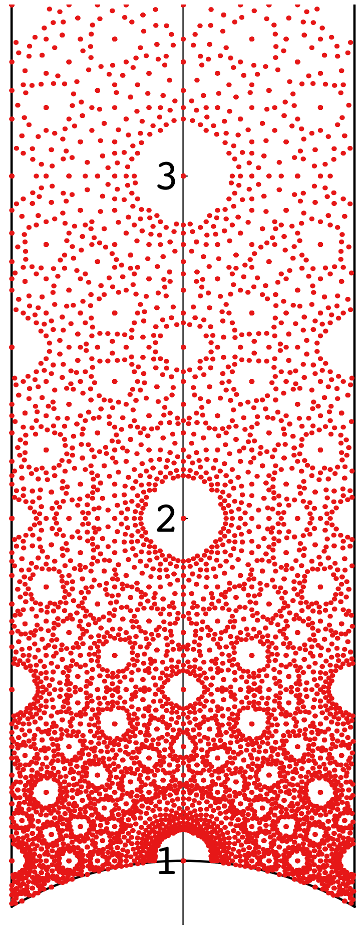
- GAs and RL have also been used to optimize the search for realistic particle physics models: https://arxiv.org/abs/2112.08391
- Using dynamic programming, we can even count the exact number of solutions: https://arxiv.org/abs/2206.03506

Learning from Topology:

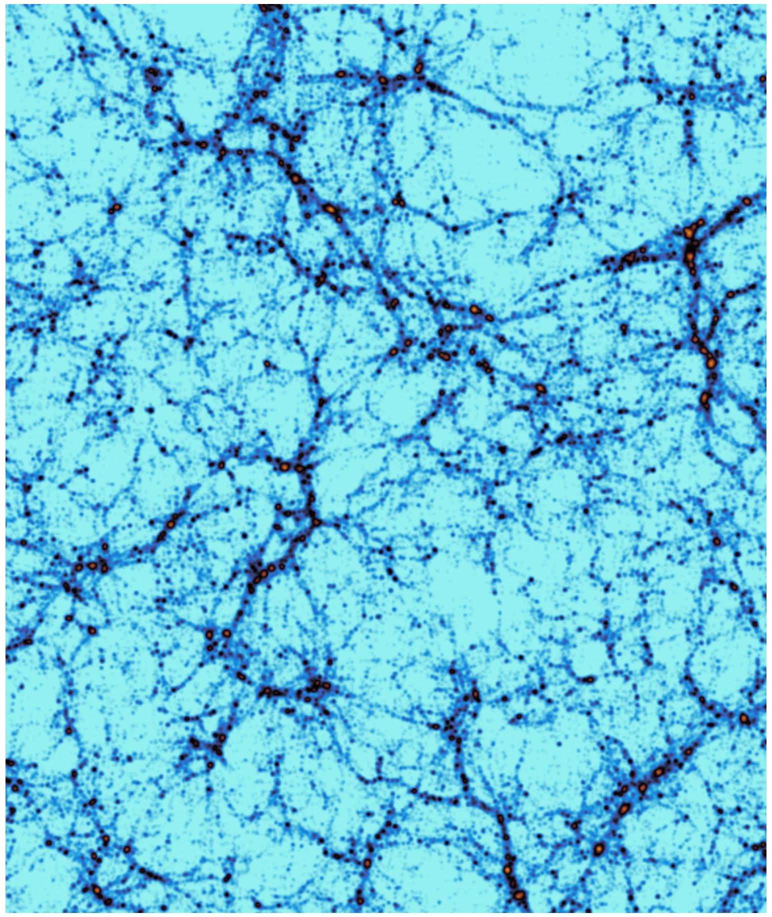


Topological Data Analysis:

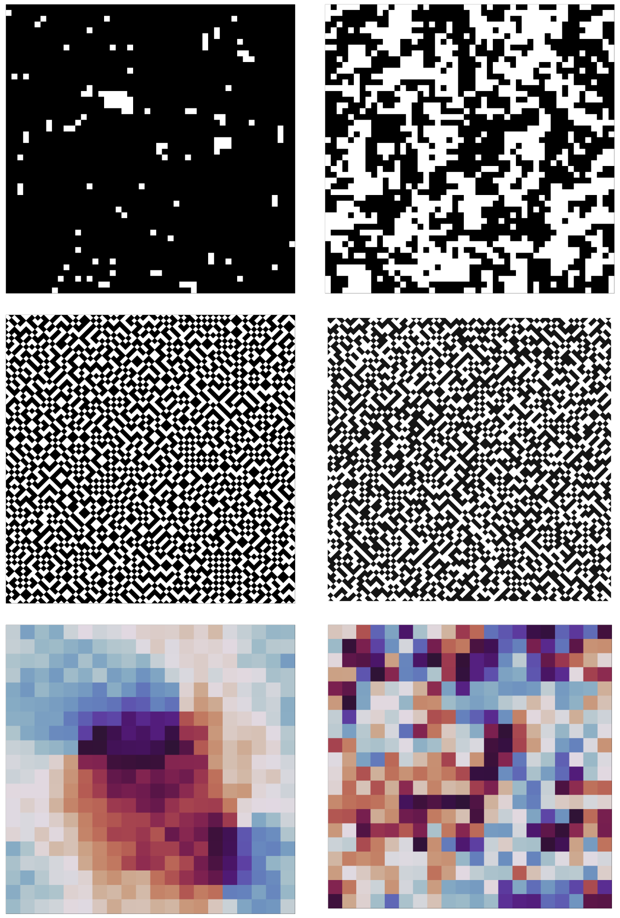
From String Theory to Cosmology to Phases of Matter



String Landscape
https://arxiv.org/abs/1812.06960
https://arxiv.org/abs/1907.10072



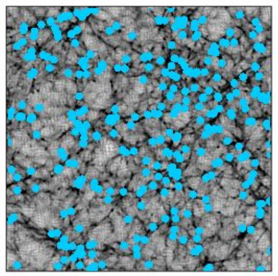
Cosmology
https://arxiv.org/abs/1710.04737
https://arxiv.org/abs/2009.04819
https://arxiv.org/abs/2308.02636
https://arxiv.org/abs/2403.13985



Phases of Matter
https://arxiv.org/abs/2009.14231

Learning from Topology

DM Simulation

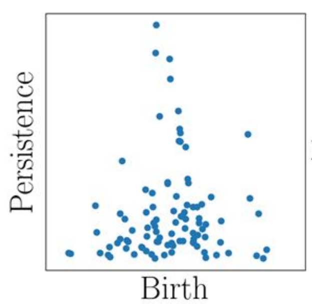


Halo Map

Parameters

(Ω_m, σ_8)

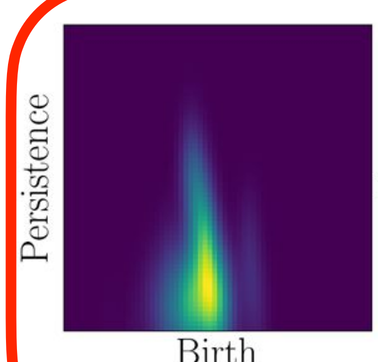
- FlowPM [Modi, ‘21]
- 256 $(\mathrm{Mpc}/h)^3$; 160^3 particles
- 36000+5000+10000 =51000 total simulations
- Rockstar halo finder



x3

Persistence Diagrams

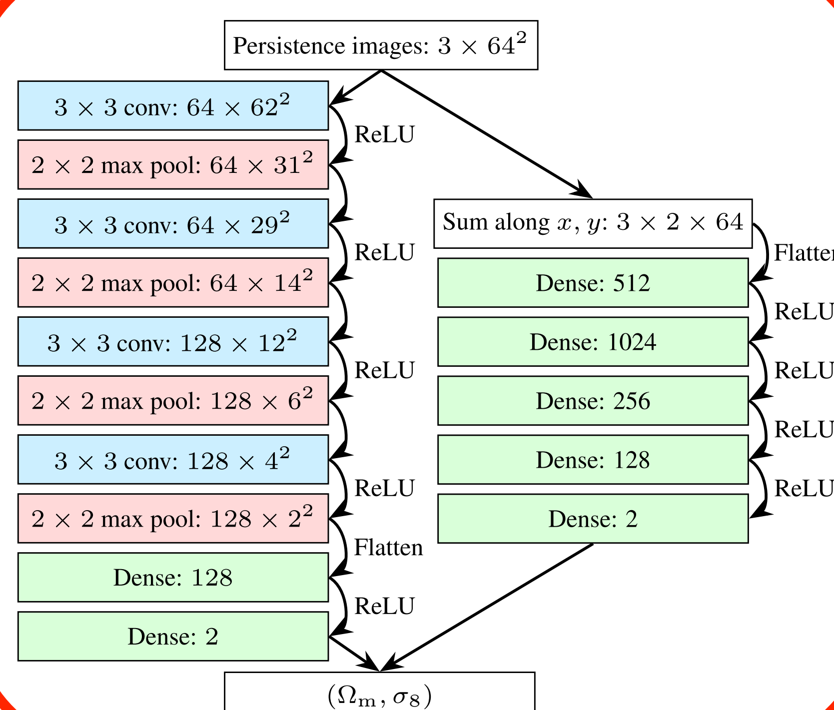
Summary Statistics

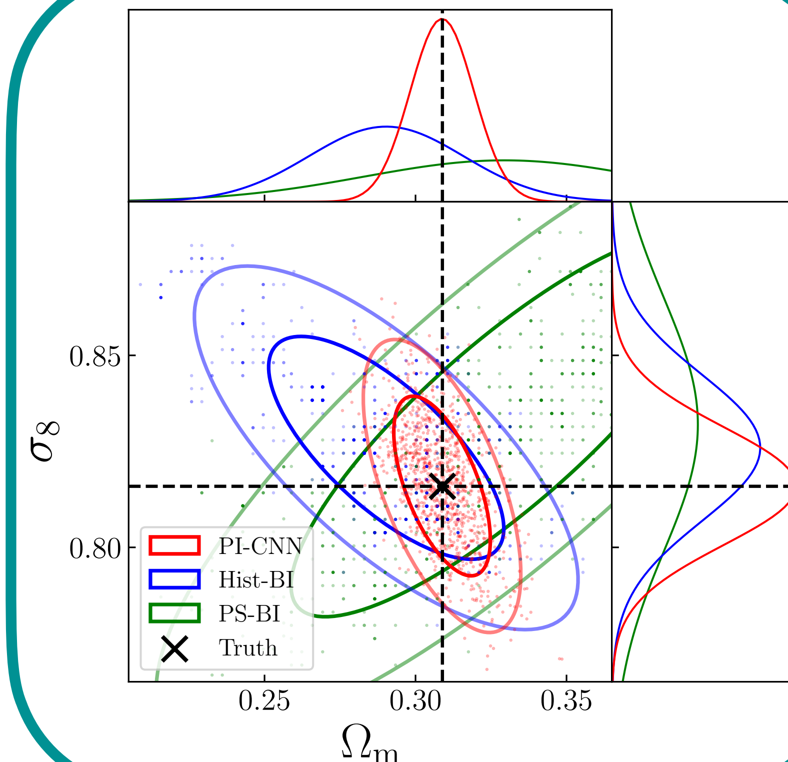


x3

Persistence Images

Machine Learning





Parameter Recovery Test

- CNN + dense branch
- +10% in the errors w/o dense branch

https://arxiv.org/abs/2308.02636