Carnegie Mellon

School of Computer Science

Deep Reinforcement Learning and Control

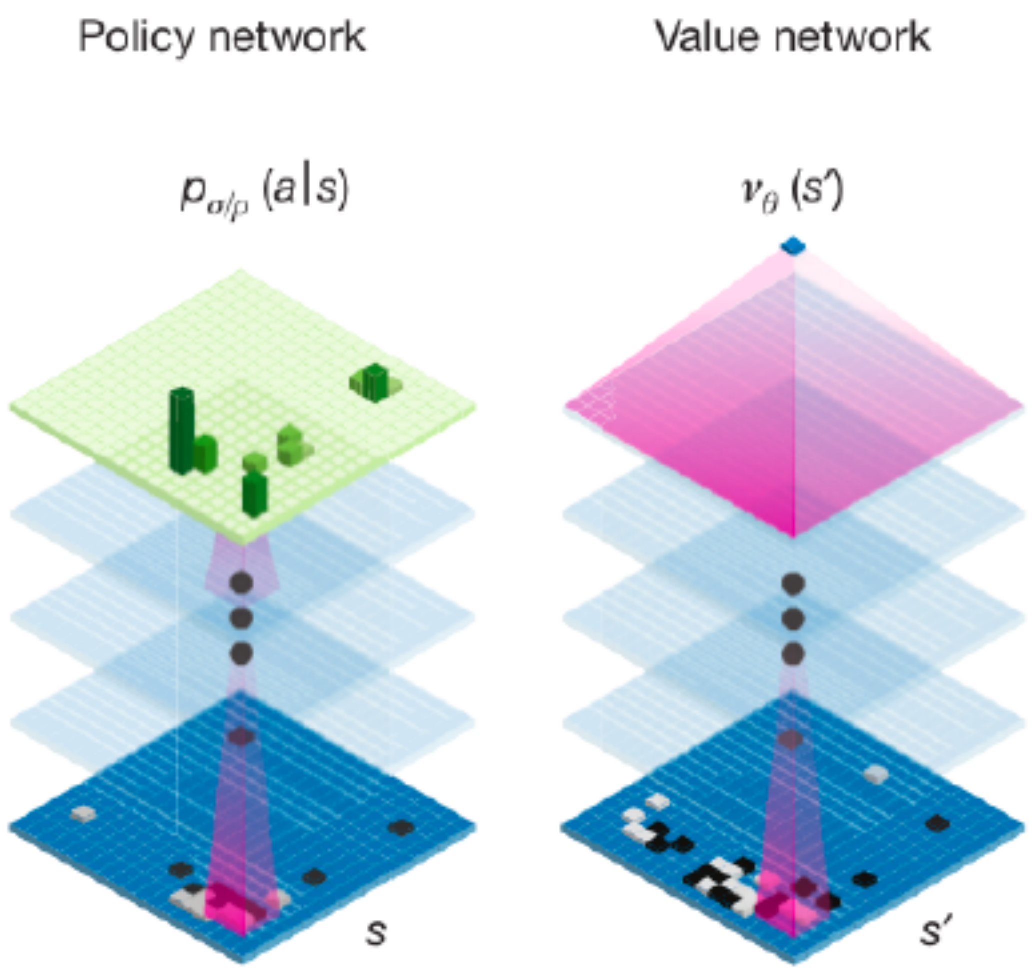
MCTS with neural nets

CMU 10-403

Katerina Fragkiadaki

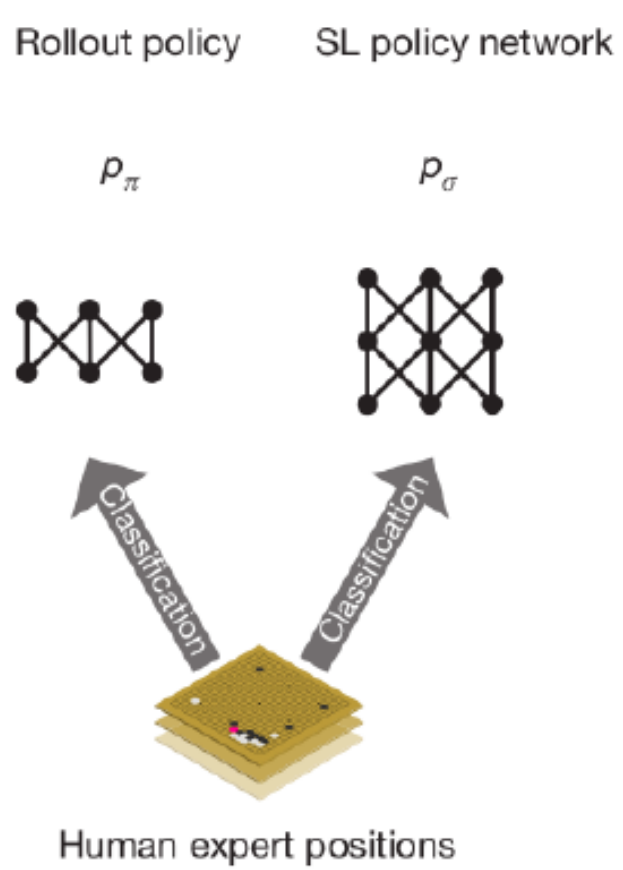
AlphaGo: Learning-guided MCTS

- **Value neural net** to evaluate board positions
- **Policy neural net** to select moves
- Combine those networks with MCTS



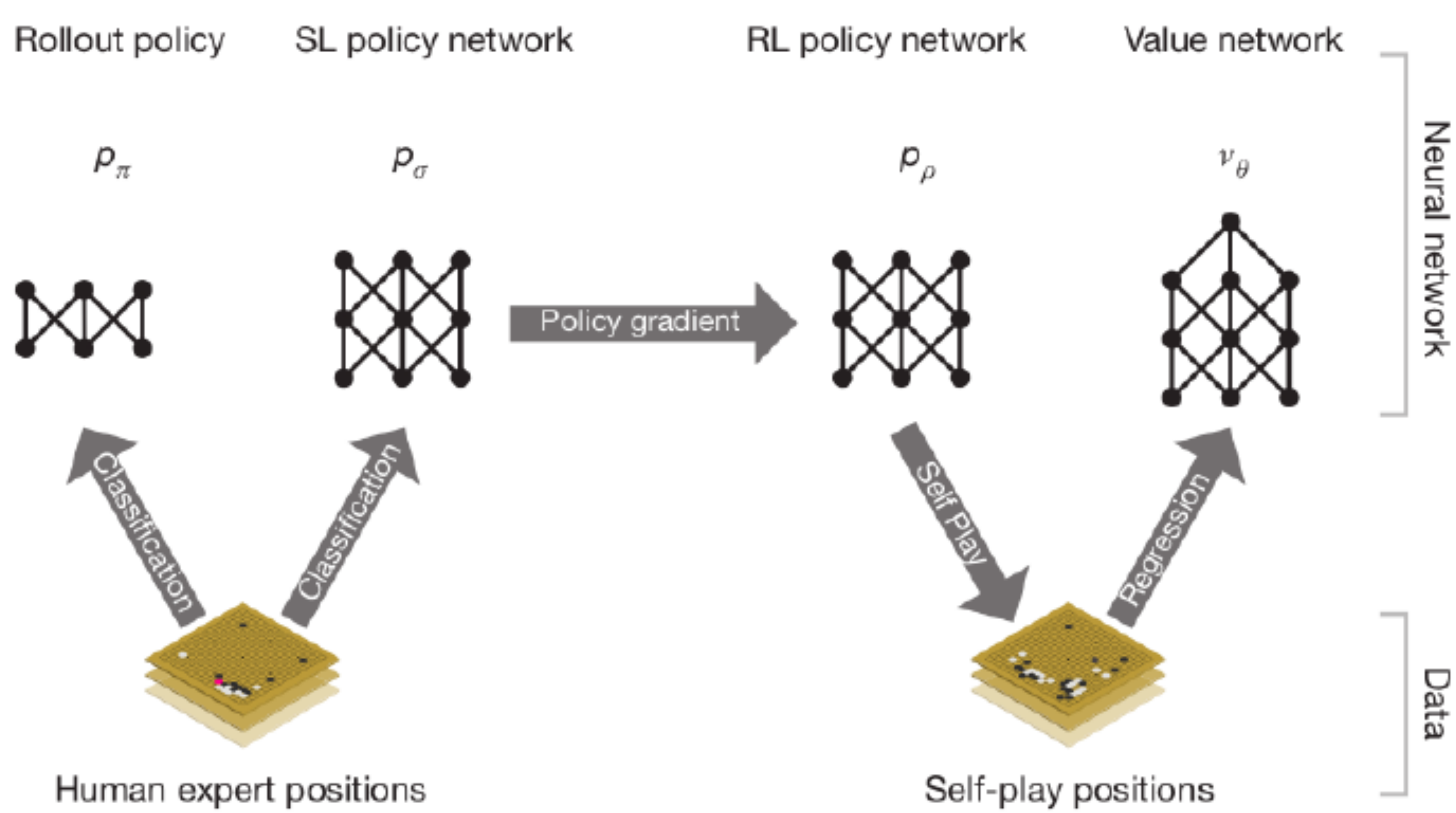
AlphaGo: Learning-guided search

1. Train two action policies by mimicking expert moves (standard supervised learning):
    1. one cheap (rollout) policy
    2. one expensive policy (SL)



AlphaGo: Learning-guided search

1. Train two action policies by mimicking expert moves (standard supervised learning):
    1. one cheap (rollout) policy
    2. one expensive policy (SL)
2. Train a new policy (SLRL) with RL and self-play initialized from SL policy.
3. Train a value network that predicts the winner of games played by SLRL against itself, as well as against previous version of policies



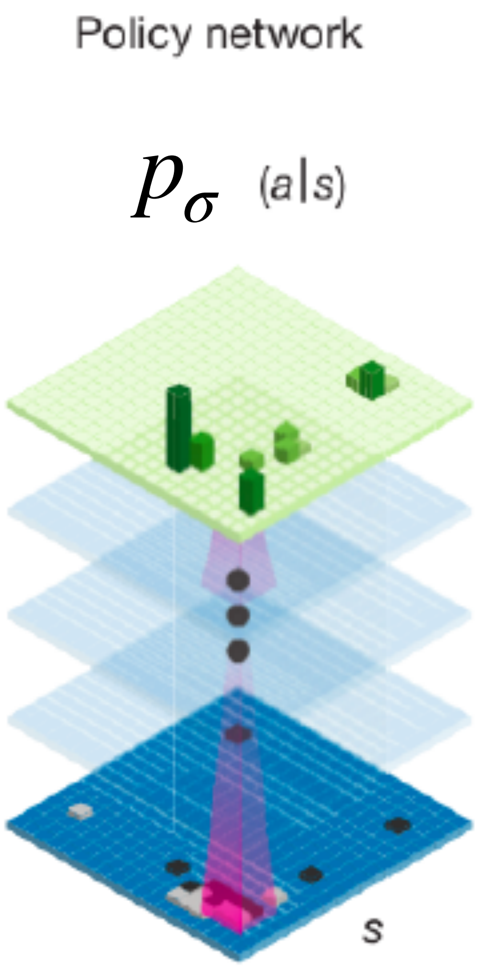
Supervised learning of policy networks

Objective: predicting expert moves

- Input: state (board configuration
- Output: a probability distribution over all **legal** moves a.

SL policy network

- 13-layer policy network trained from 30 million positions.
- accuracy of 57.0% using all input features, 55.7% using only raw board position and move history
- (compared to the state-of-the-art from other research groups of 44.4%).



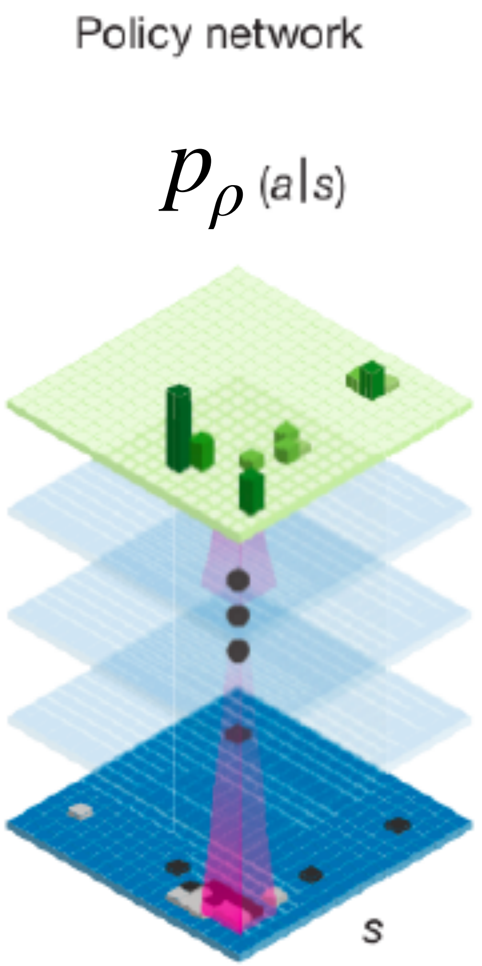
RL with REINFORCE

Objective: improve over SL policy

- Weight initialization from SL network
- Input: Sampled states during self-play
- Output: a probability distribution over all **legal** moves a.

Rewards are provided only at the end of the game, +1 for winning, -1 for loosing

$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t \mid s_t)}{\partial \rho} z_t$$



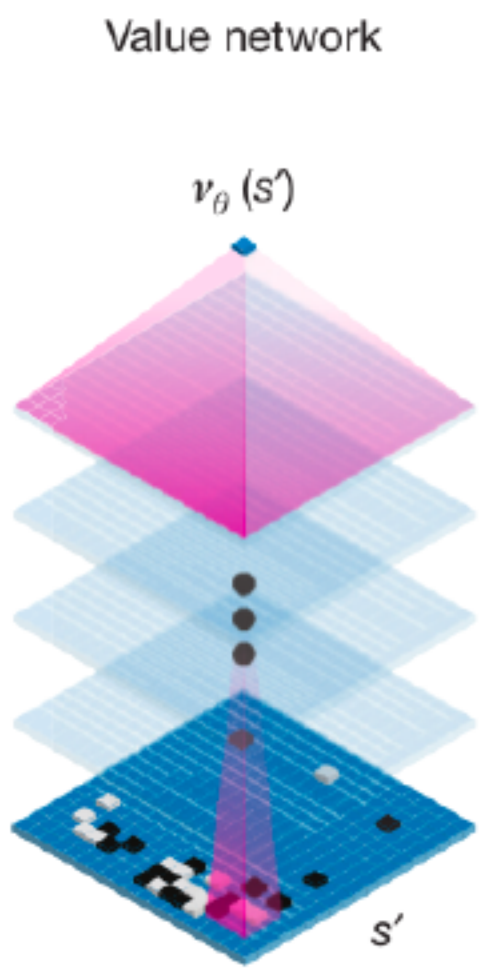
The RL policy won more than 80% of games against the SL policy.

Supervised learning of value networks

Objective: Estimating a value function $v_p(s)$ that predicts the outcome from position (board configuration) s

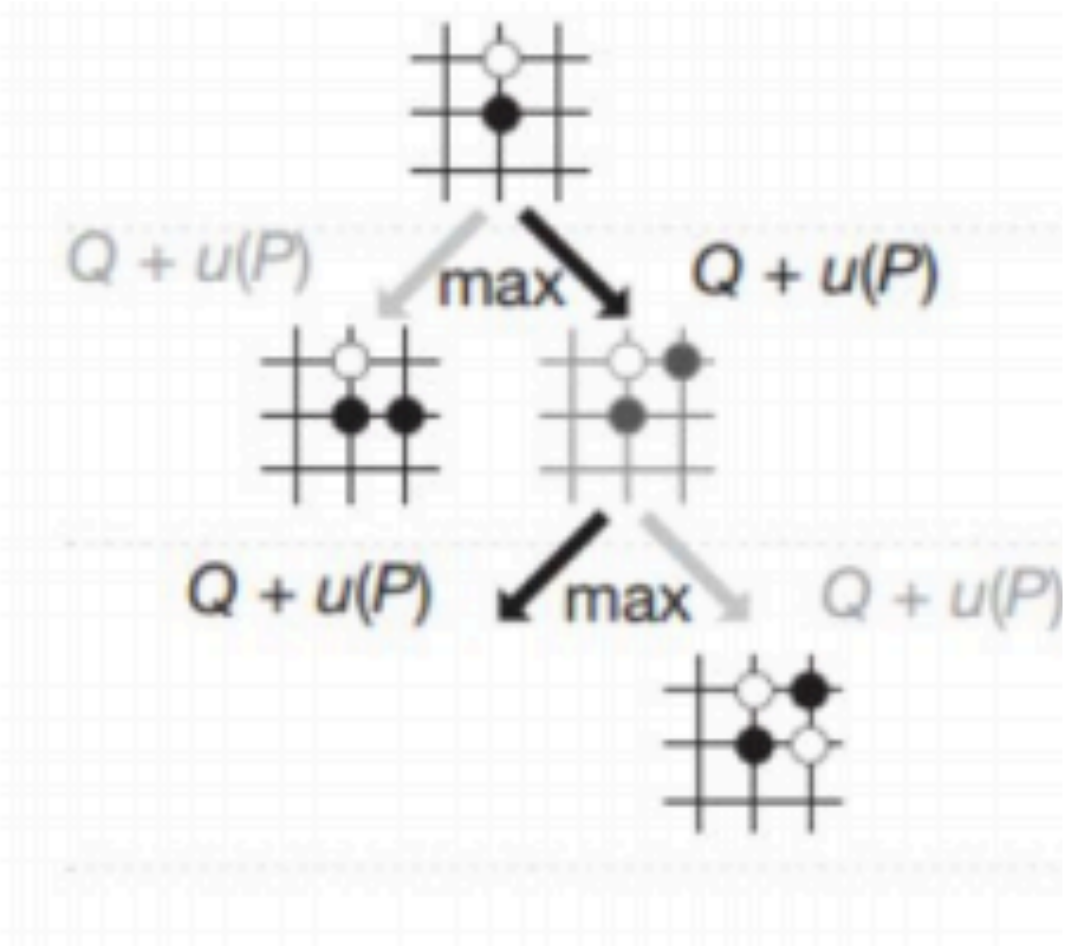
- Input: Sampled states during self-play, 30 million distinct positions, each sampled from a separate game, played by the SLRL policy against itself (and against previous policy versions).
- Output: the board score (a scalar value)

Trained by regression on state-outcome pairs (s, z) to minimize the mean squared error between the predicted value v(s), and the corresponding outcome z.



MCTS + Policy/ Value networks

Selection: selecting actions within the expanded tree



Tree policy

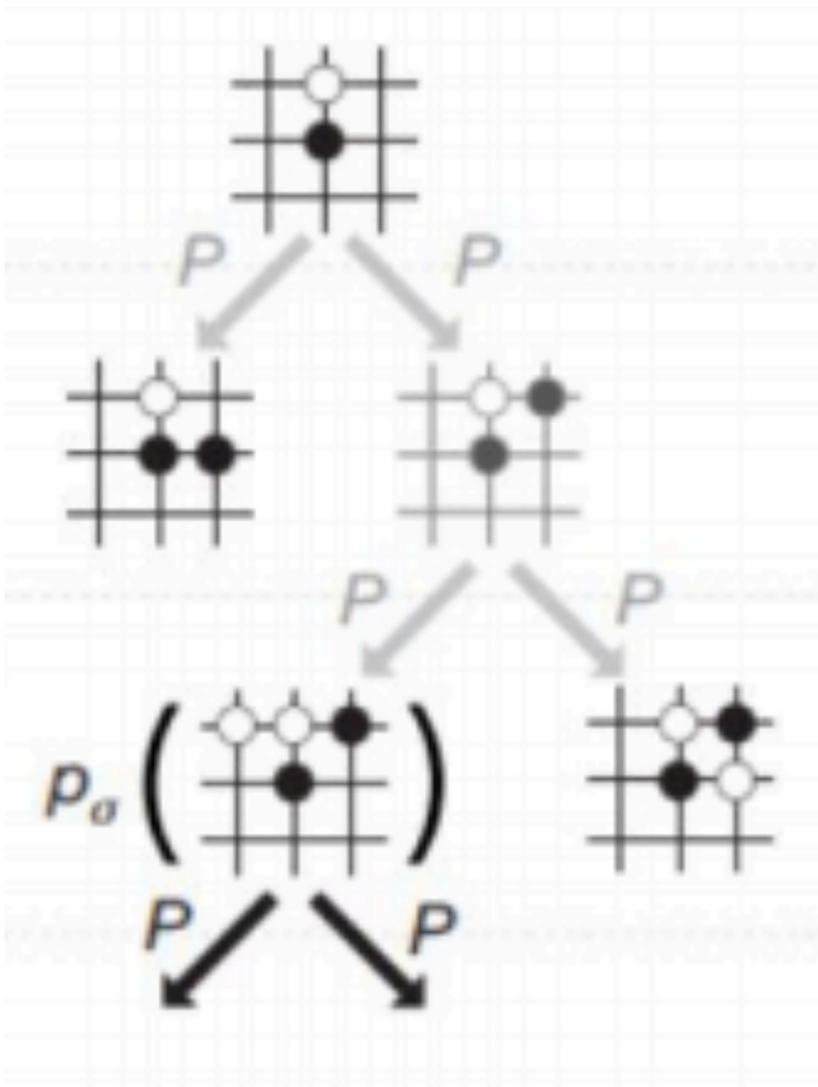
$$a_t = \operatorname{argmax}_a (Q(s_t, a) + u(s_t, a))$$

$$u(s, a) \propto \frac{P(s, a)}{1 + N(s, a)}$$

- a_t - action selected at time step t from board s_t
- $Q(s_t, a)$ - average reward collected so far from MC simulations
- $P(s, a)$ - prior expert probability of playing moving a provided by the SL policy
- $N(s, a)$ - number of times we have visited parent node
- u acts as a bonus value
  - Decays with repeated visits

MCTS + Policy/ Value networks

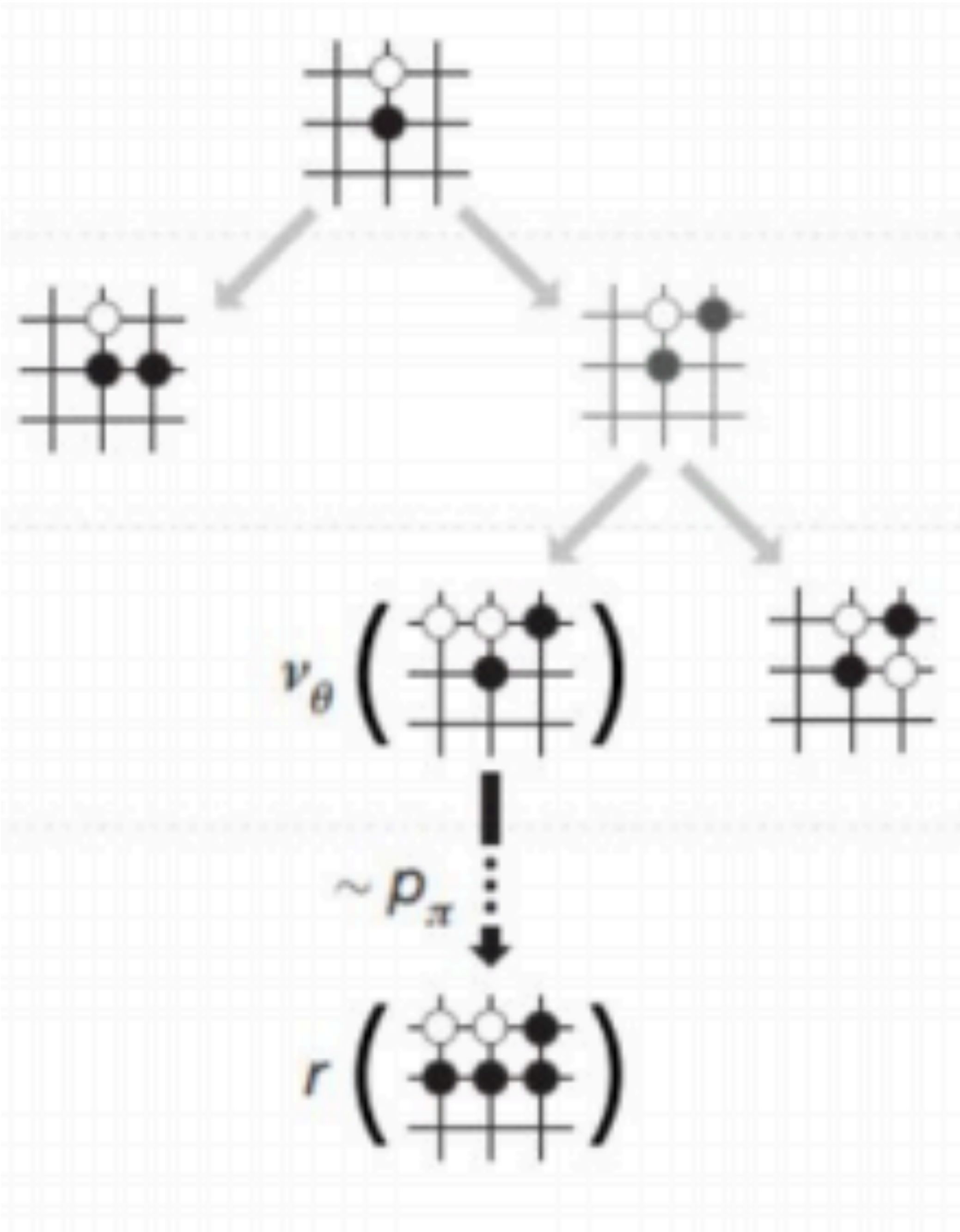
Expansion: when reaching a leaf, play the action with highest score from p_σ



- When leaf node is reached, it has a chance to be expanded
- Processed once by SL policy network (p_σ) and stored as prior probs *P(s, a)*
- Pick child node with highest prior prob

MCTS + Policy/ Value networks

Simulation/Evaluation: use the rollout policy to reach to the end of the game



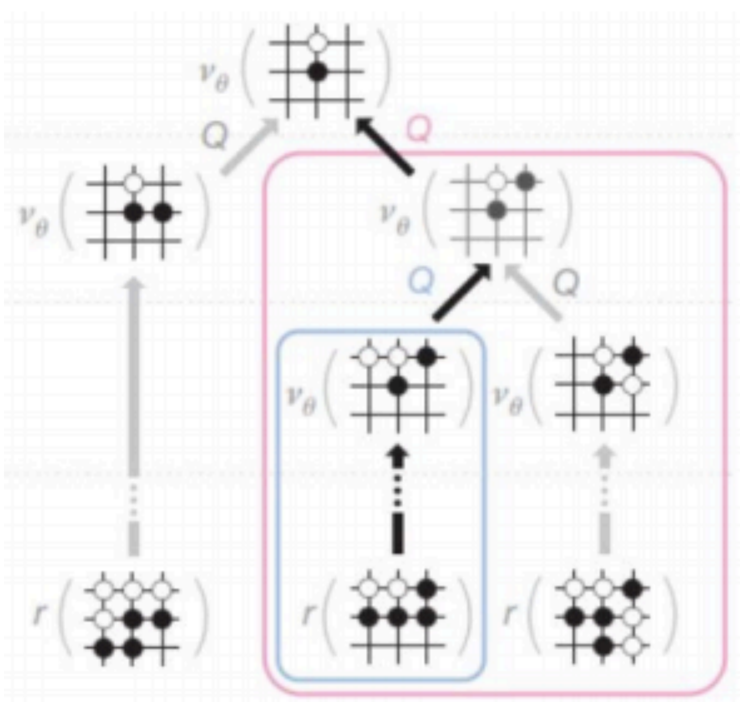
- From the selected leaf node, run multiple simulations in parallel using the rollout policy
- Evaluate the leaf node as:

$$V(s_L) = (1 - \lambda)v_\theta(s_L) + \lambda z_L$$

- v_θ - value from value function of board position s_L
- z_L - Reward from fast rollout p_π
  - Played until terminal step
- λ - mixing parameter
  - Empirical

MCTS + Policy/ Value networks

Backup: update visitation counts and recorded rewards for the chosen path inside the tree:



$$N(s,a) = \sum_{i=1}^{n} 1(s,a,i)$$

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} 1(s,a,i) V(s_L^i)$$

- Extra index *i* is to denote the *i*th simulation, *n* total simulations
- Update visit count and mean reward of simulations passing through node
- Once search completes:
  - Algorithm chooses the most visited move from the root position

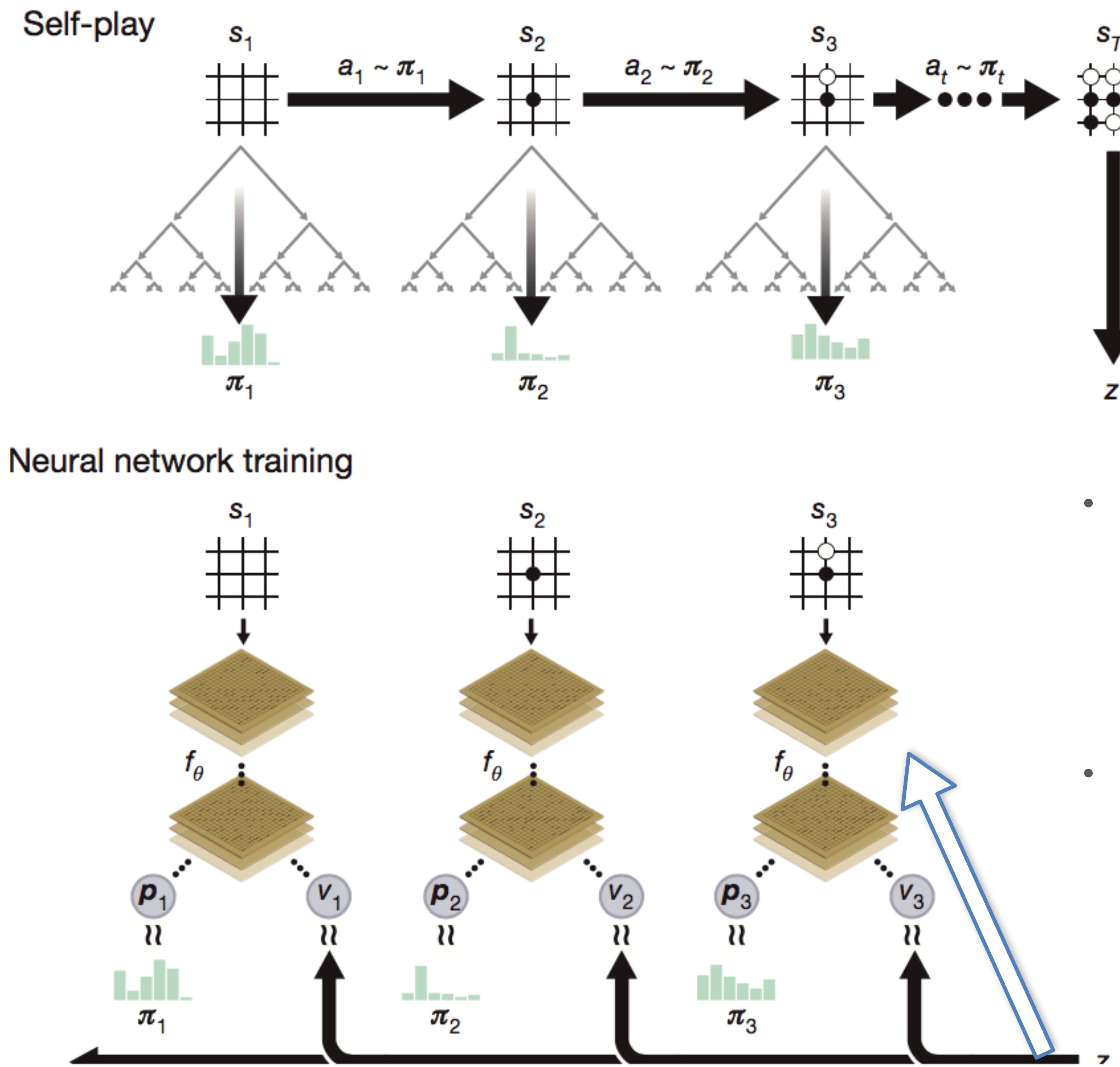
AlphaGoZero: Lookahead search during training!

- So far, MCTS was used for online planning to select moves at test time
- AlphaGoZero uses it during training instead.

AlphaGoZero: Lookahead search during training!

- Given any policy, a MCTS guided by this policy will produce an improved policy (policy improvement operator)
- Train a policy to iteratively mimic such improved policy
- Policy iteration

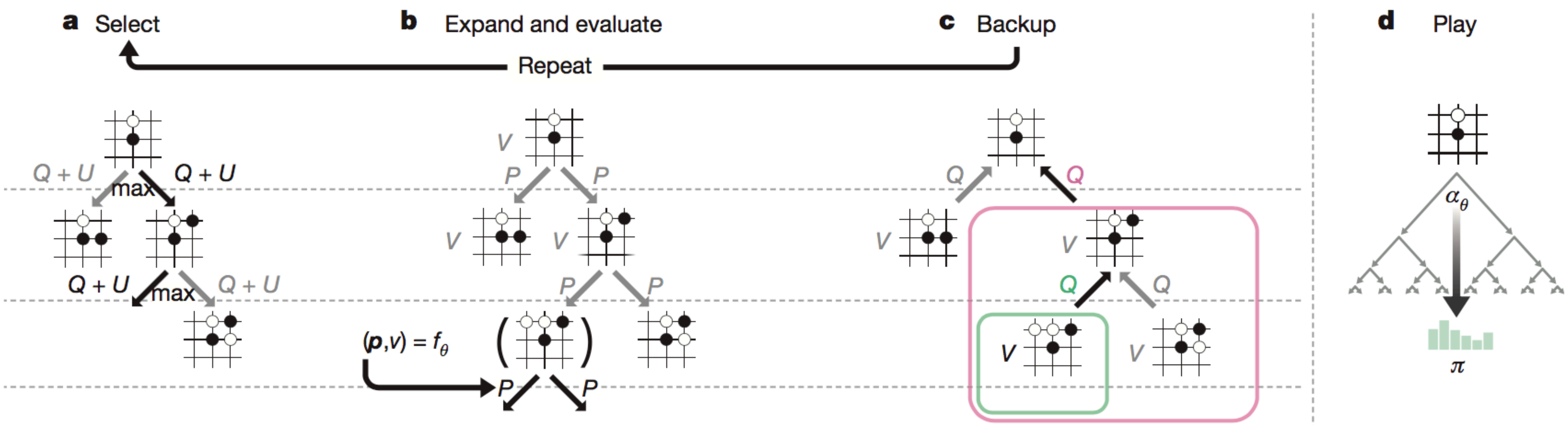
MCTS as policy improvement operator



- Supervised training so that the policy network mimics the output of the MCTS (supervision from a planner!)
- Train so that the value network matches the outcome (same as in AlphaGo)

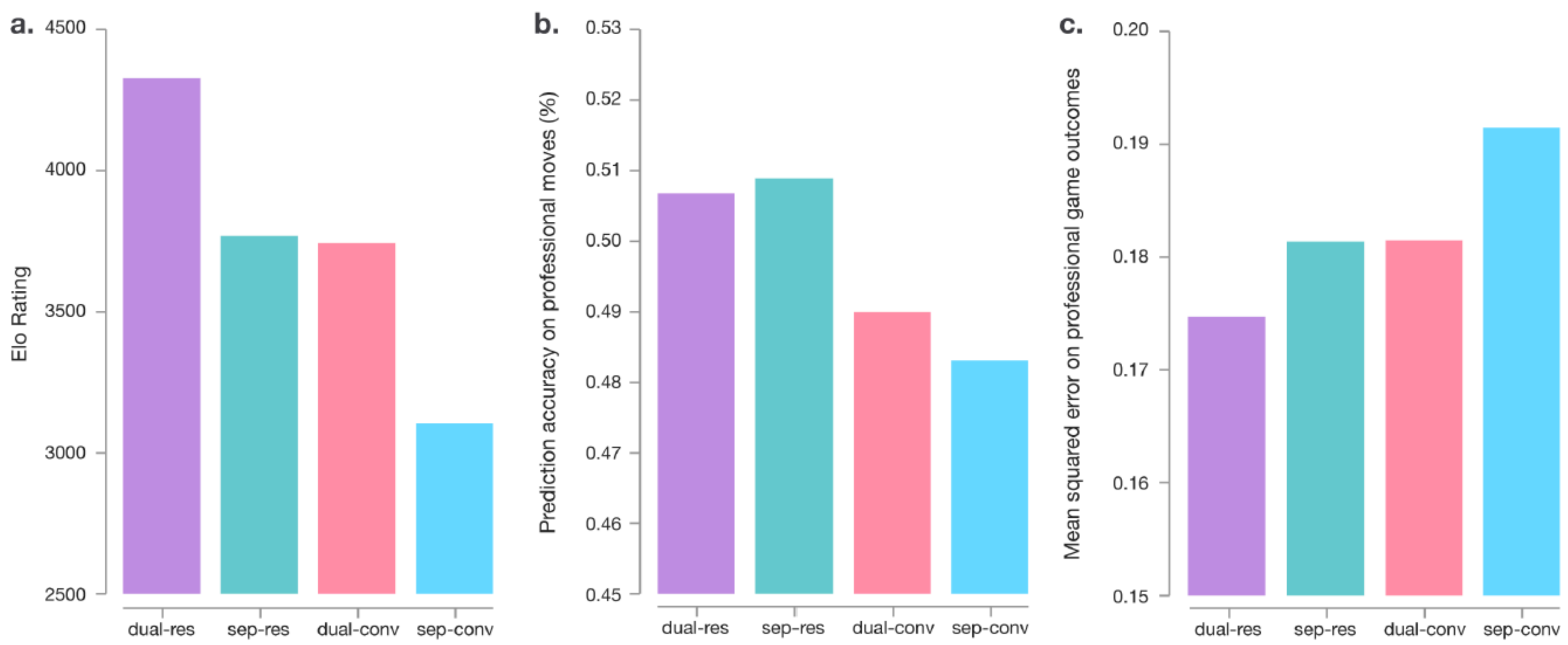
Note that policy and value networks share the backbone!

MCTS: no MC rollouts till termination

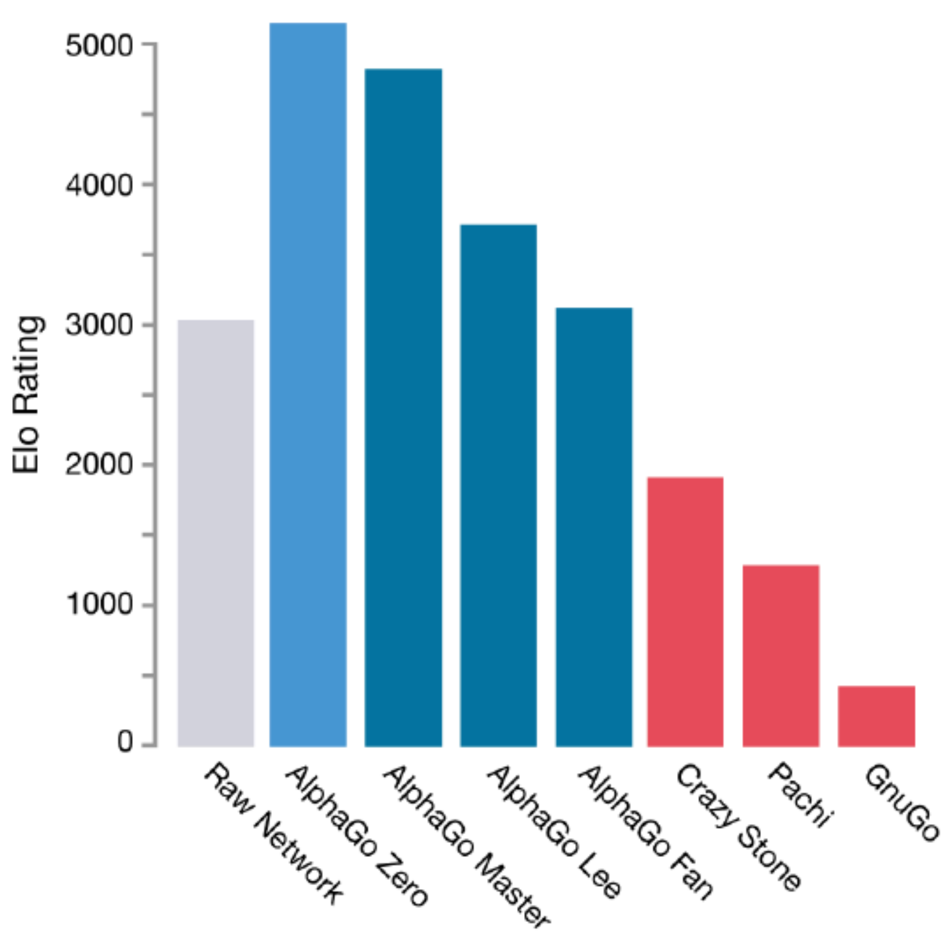


MCTS: using always value net evaluations of leaf nodes, no rollouts!

Architectures

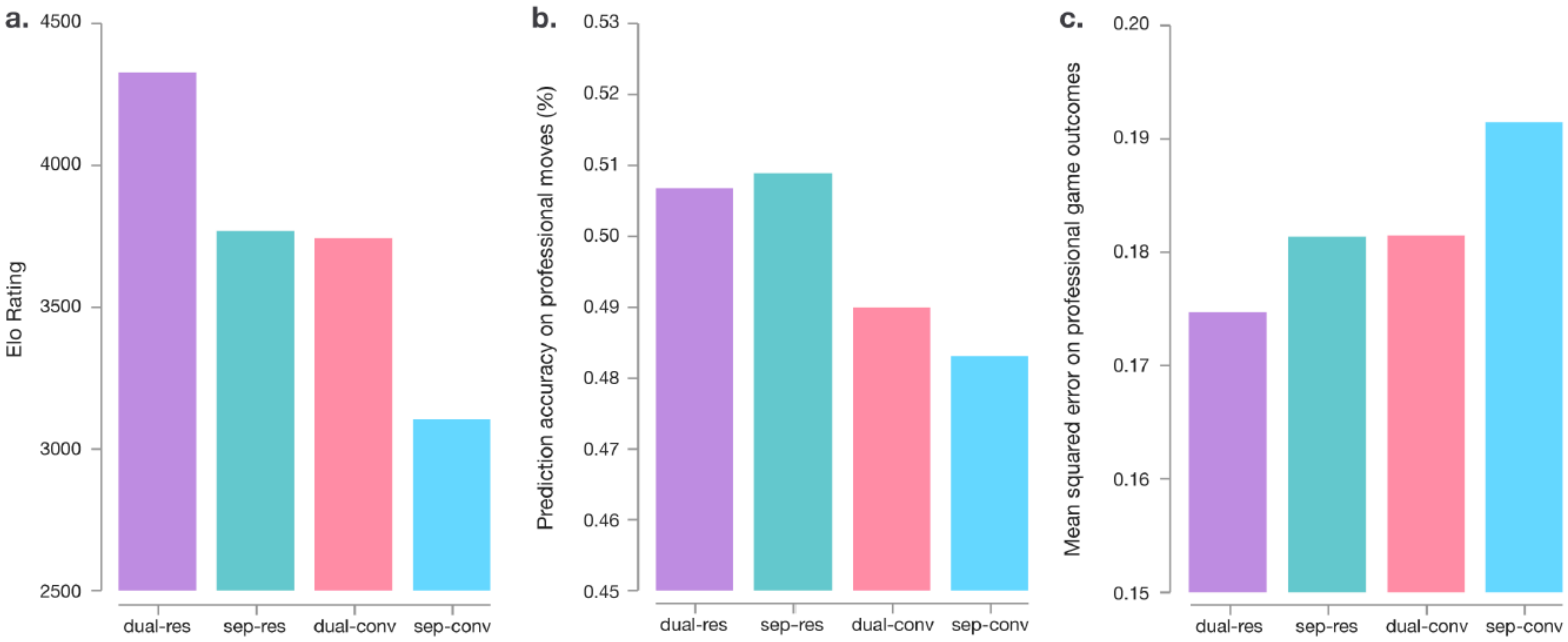


- Resnets help
- Jointly training the policy and value function using the same main feature extractor helps



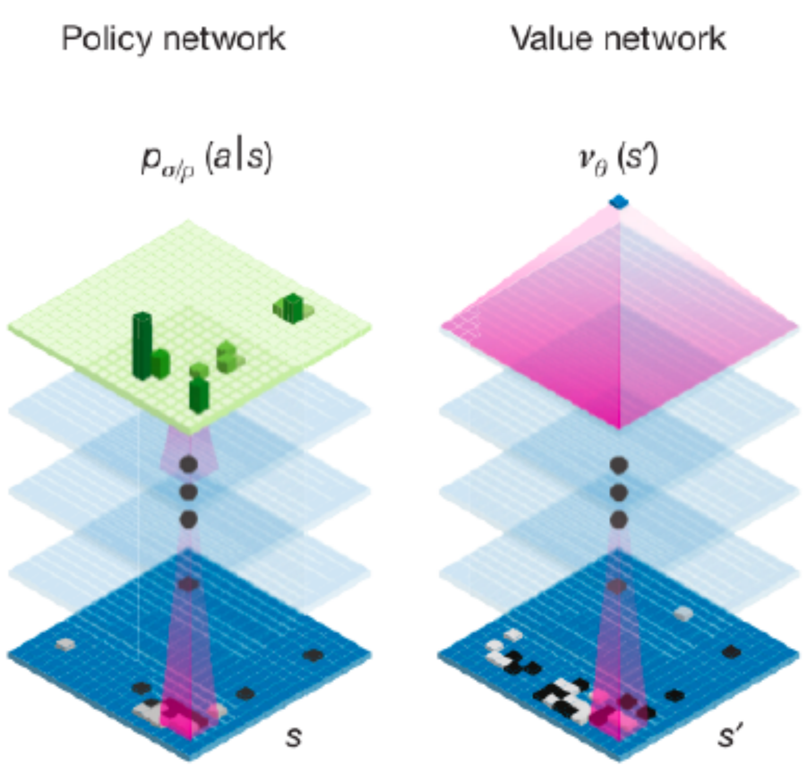
- MCTS improves the basic policy

Architectures

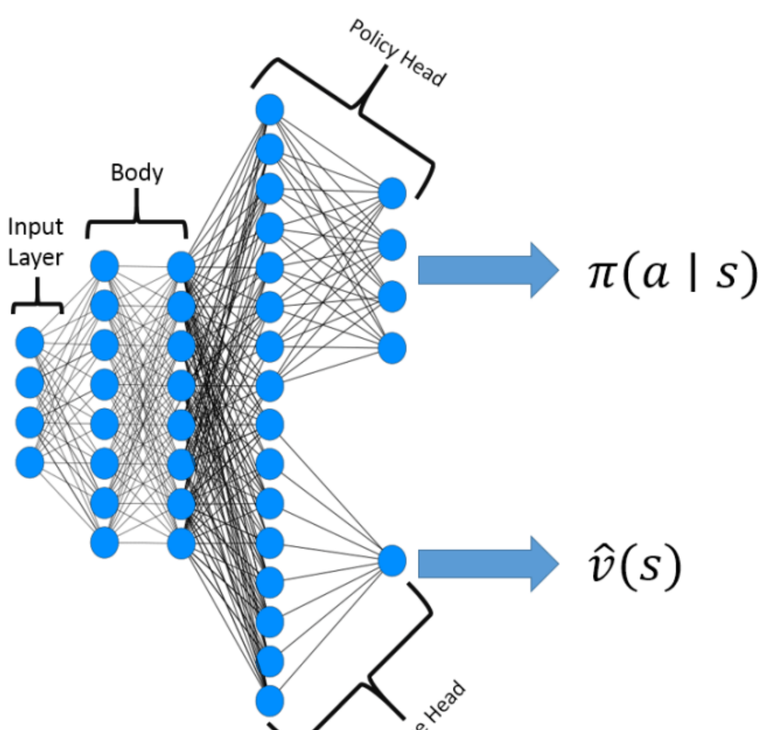


- Resnets help
- Jointly training the policy and value function using the same main feature extractor helps

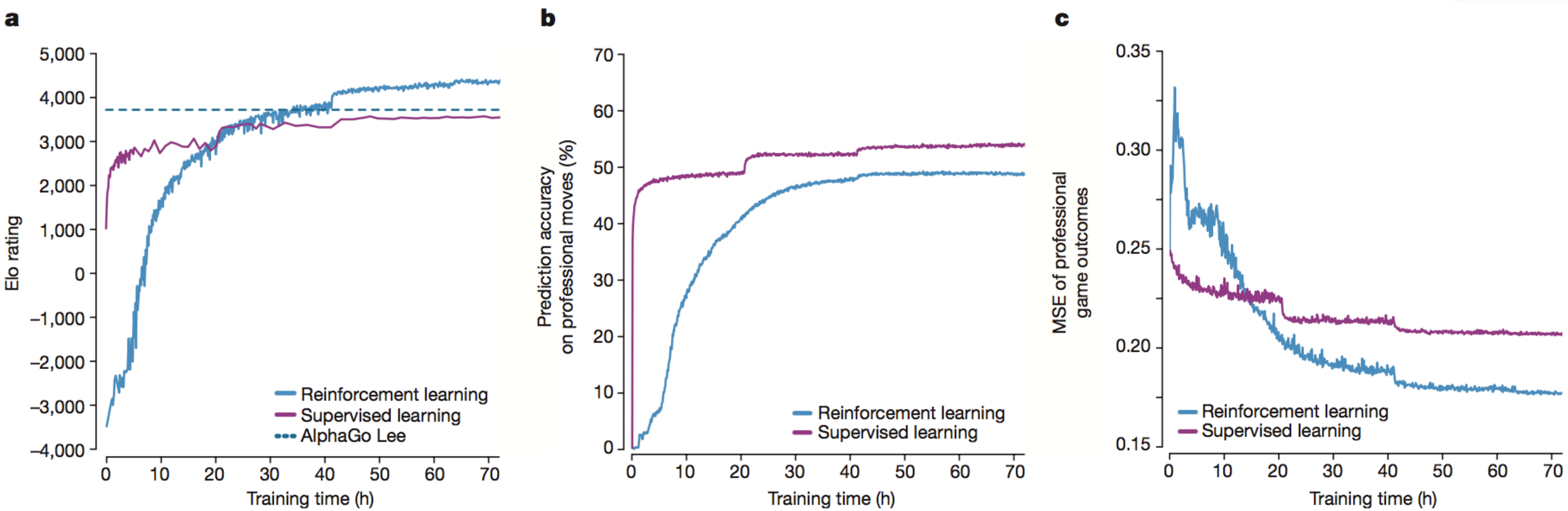
Separate policy/value nets



Joint policy/value nets



RL VS SL


a
Elo rating
5,000
4,000
3,000
2,000
1,000
0
–1,000
–2,000
–3,000
–4,000
0 10 20 30 40 50 60 70
Training time (h)
Reinforcement learning
Supervised learning
AlphaGo Lee
b
Prediction accuracy on professional moves (%)
70
60
50
40
30
20
10
0
0 10 20 30 40 50 60 70
Training time (h)
Reinforcement learning
Supervised learning
c
MSE of professional game outcomes
0.35
0.30
0.25
0.20
0.15
0 10 20 30 40 50 60 70
Training time (h)
Reinforcement learning
Supervised learning

Carnegie Mellon

School of Computer Science

Deep Reinforcement Learning and Control

Evolutionary Methods

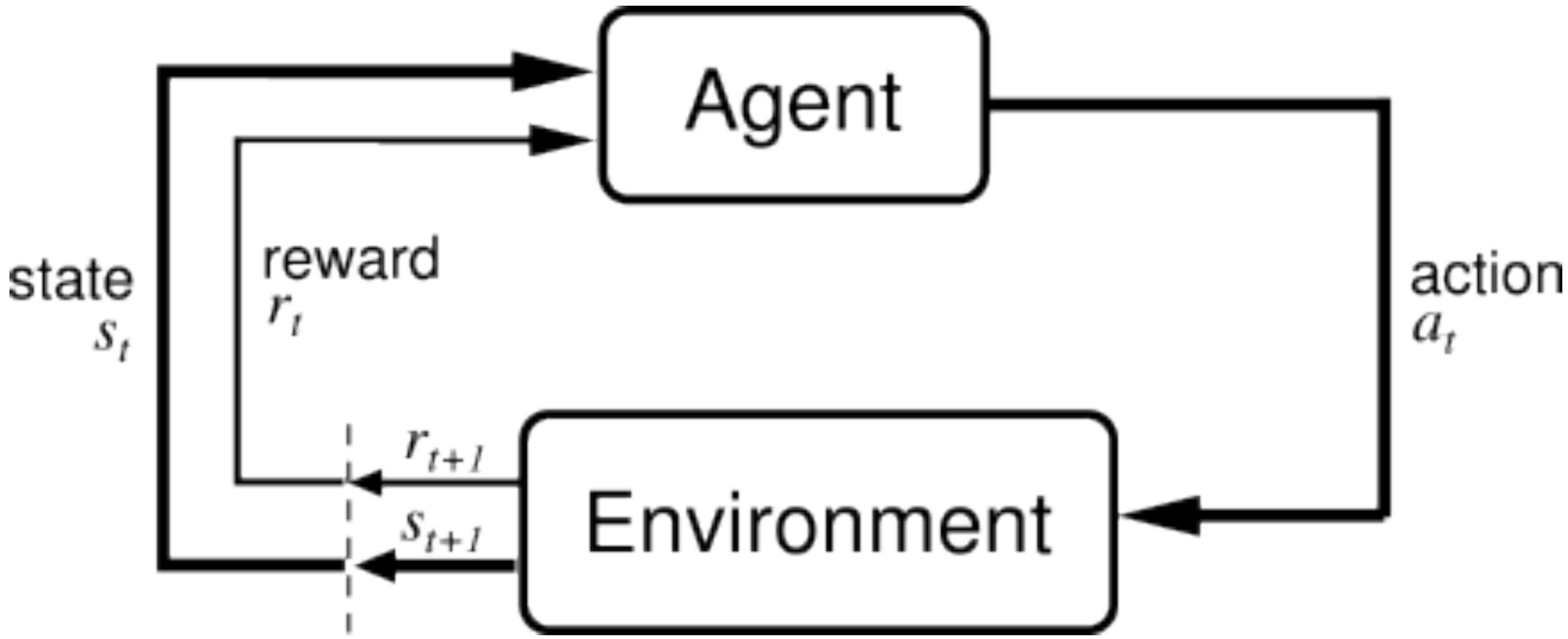
CMU 10-403

Katerina Fragkiadaki

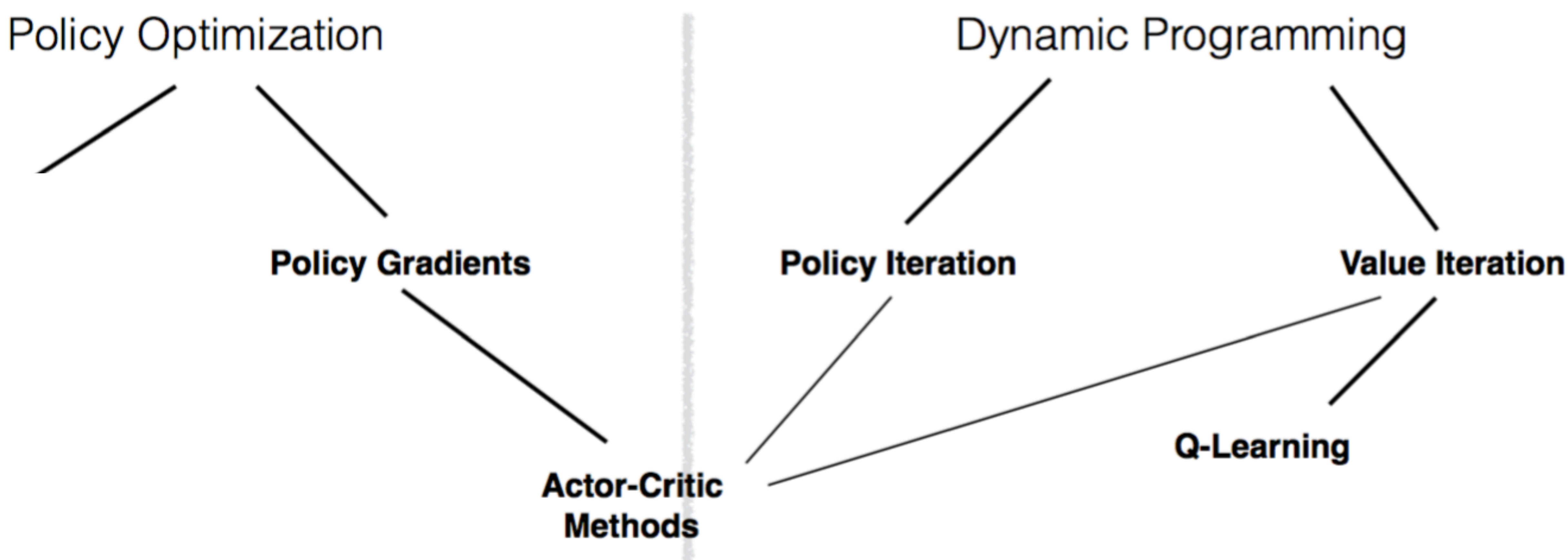
Part of the slides borrowed by Xi Chen, Pieter Abbeel, John Schulman

Policy Optimization and RL

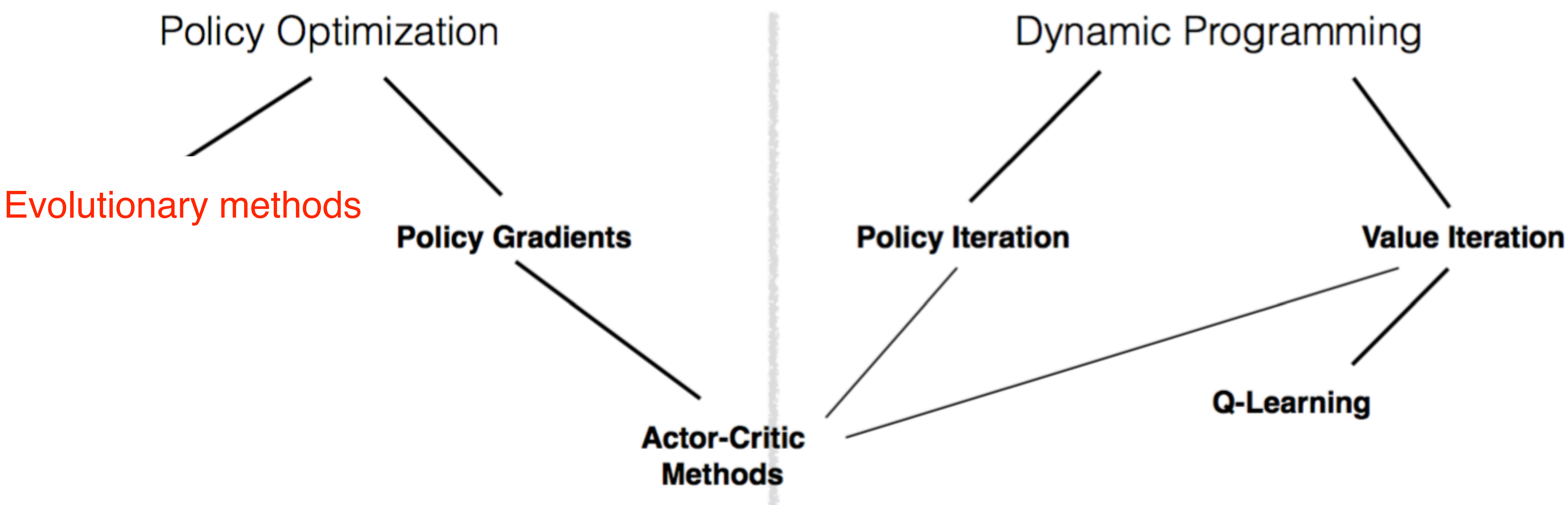
$$\max_\theta U(\theta) = \max_\theta \mathbb{E}\left[R(\tau) \,|\, \pi_\theta, \mu_0(s_0)\right] = \max_\theta \mathbb{E}\left[\sum_{t=0}^{T} R(s_t) \,|\, \pi_\theta, \mu_0(s)\right]$$



$$\max_{\theta} . \quad U(\theta) = \mathbb{E}\left[R(\tau) \,|\, \pi_\theta, \mu_0(s_0)\right]$$



$$\max_{\theta} . \quad U(\theta) = \mathbb{E}\left[R(\tau) \mid \pi_{\theta}, \mu_0(s_0)\right]$$



Black-box Policy Optimization

$$\max_{\theta}. \quad U(\theta) = \mathbb{E}\left[R(\tau) \mid \pi_\theta, \mu_0(s_0)\right]$$

$$\theta$$

$$\mathbb{E}\left[R(\tau)\right]$$

No information regarding the structure of the state space or the reward

Evolutionary methods

$$\max_{\theta}. \quad U(\theta) = \mathbb{E}\left[R(\tau) \,|\, \pi_\theta, \mu_0(s_0)\right]$$

General algorithm:

Initialize a population of parameter vectors (*genotypes*)

1. Make random perturbations (*mutations*) to each parameter vector
2. Evaluate the perturbed parameter vector (*fitness*)
3. Keep the perturbed vector if the result improves (*selection*)
4. GOTO 1

Biologically plausible…

Cross-entropy method

Let's consider our parameters to be sampled from a multivariate isotropic Gaussian
We will evolve this Gaussian towards samples that have highest fitness

CEM:

Initialize $\mu \in \mathbb{R}^d, \sigma \in \mathbb{R}^d_{>0}$

for iteration = 1, 2, …

Sample n parameters $\theta_i \sim N(\mu, \mathrm{diag}(\sigma^2))$

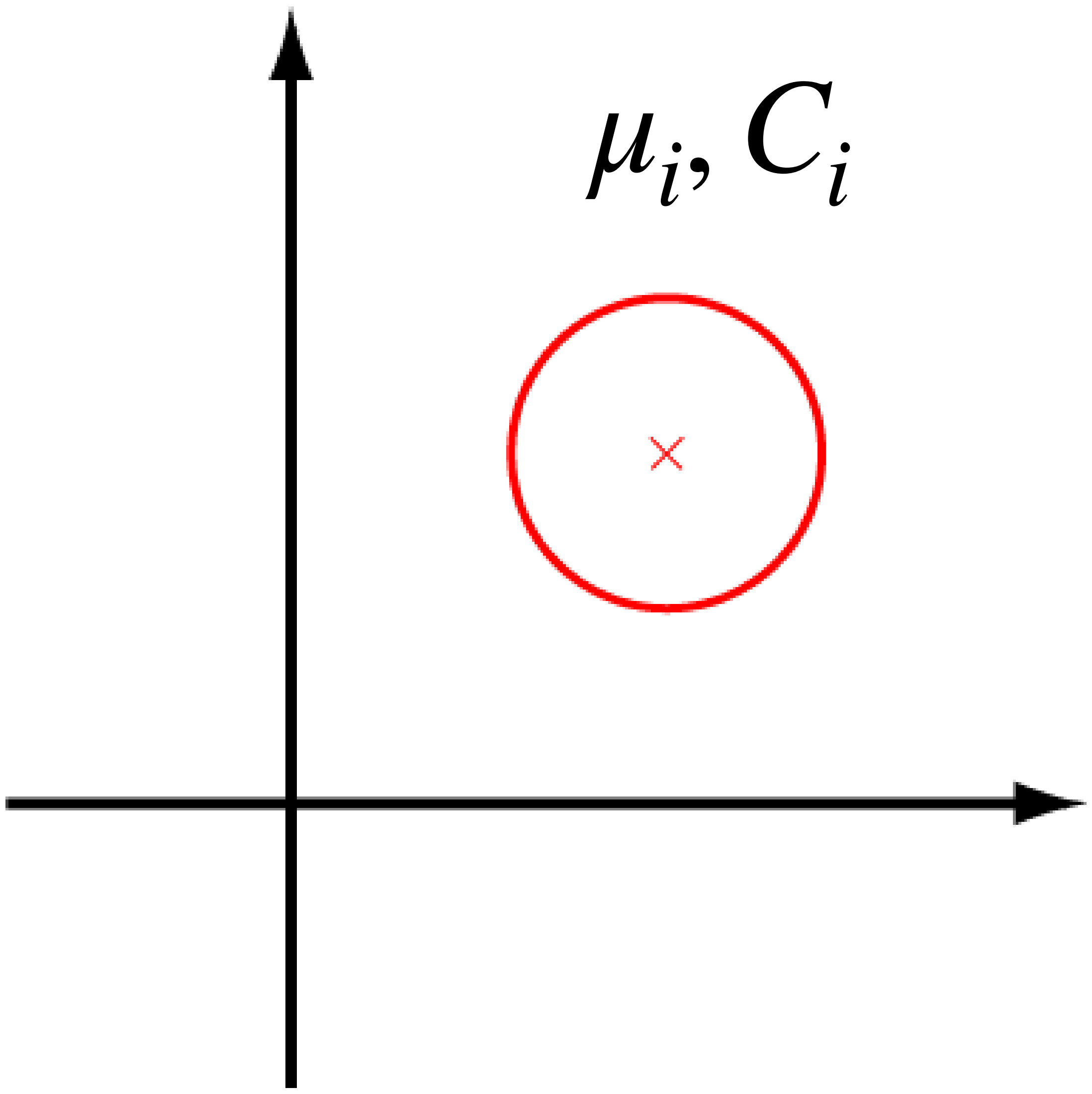
For each θ_i, perform one rollout to get return $R(\tau_i)$

Select the top k% of θ, and fit a new diagonal Gaussian to those samples. Update μ, σ

endfor

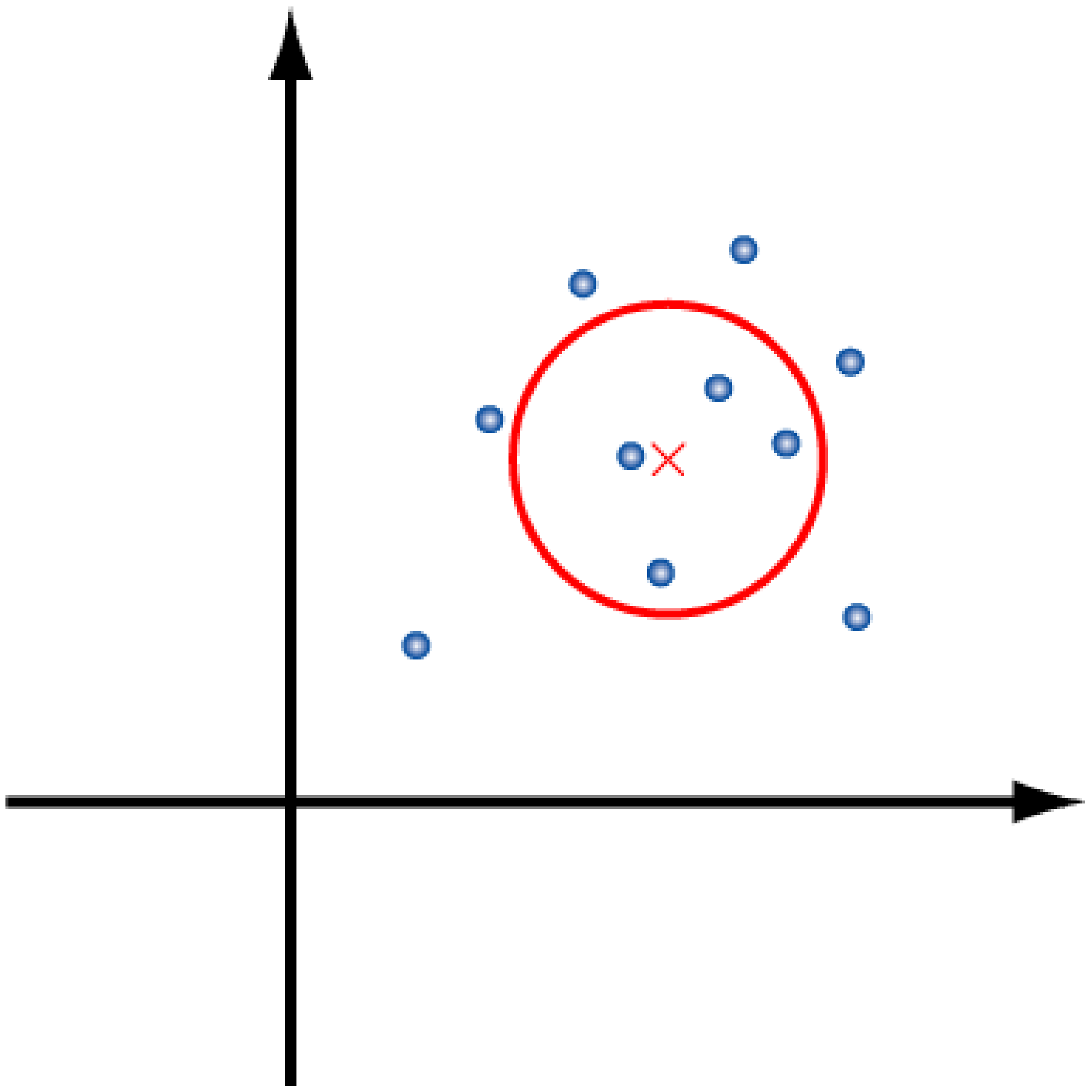
Covariance Matrix Adaptation

Let's consider our parameters to be sampled from a multivariate Gaussian
We will evolve this Gaussian towards sampled that have highest fitness



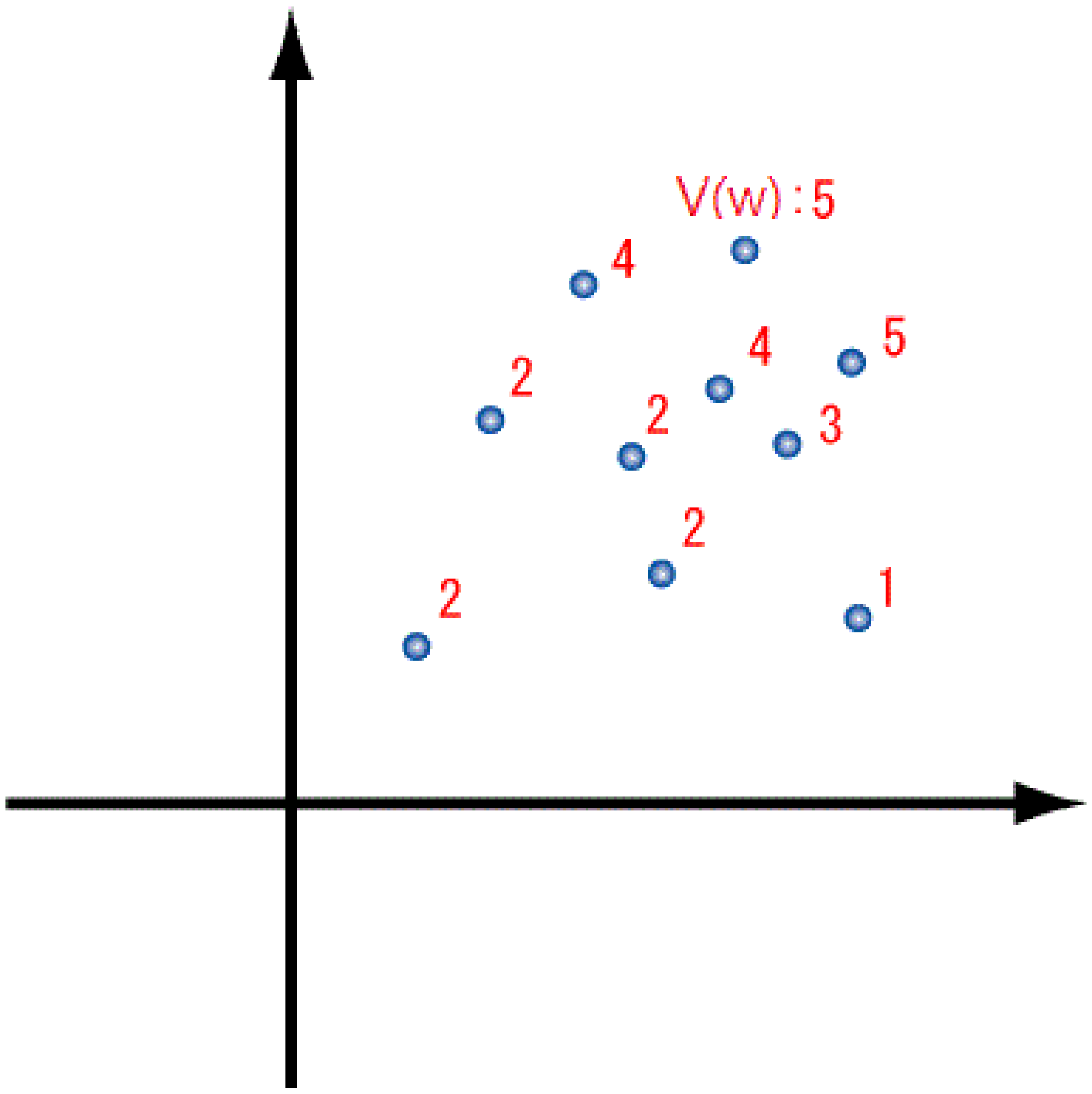
- Sample
- Select elites
- Update mean
- Update covariance
- iterate

Covariance Matrix Adaptation



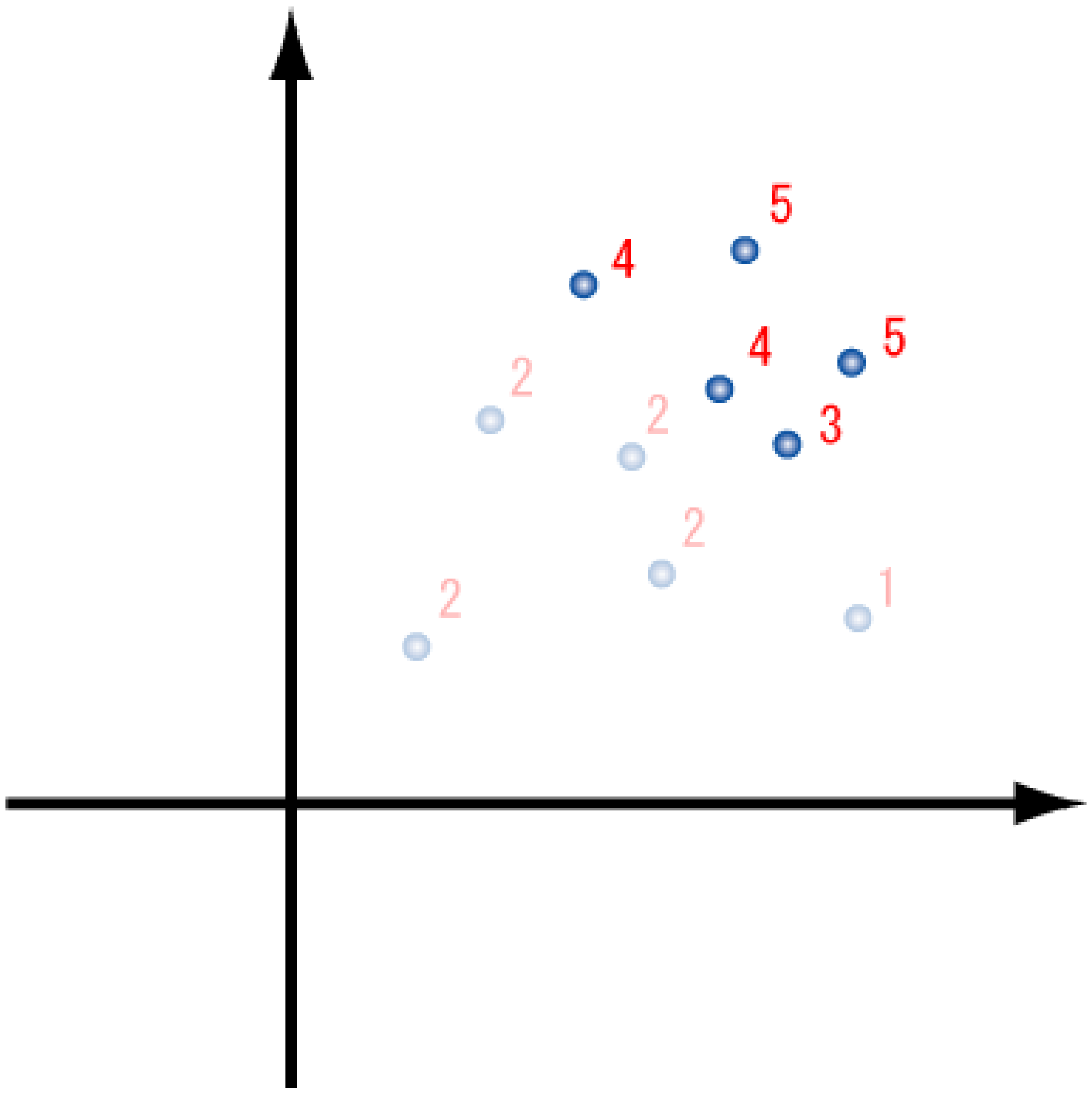
- Sample
- Select elites
- Update mean
- Update covariance
- iterate

Covariance Matrix Adaptation



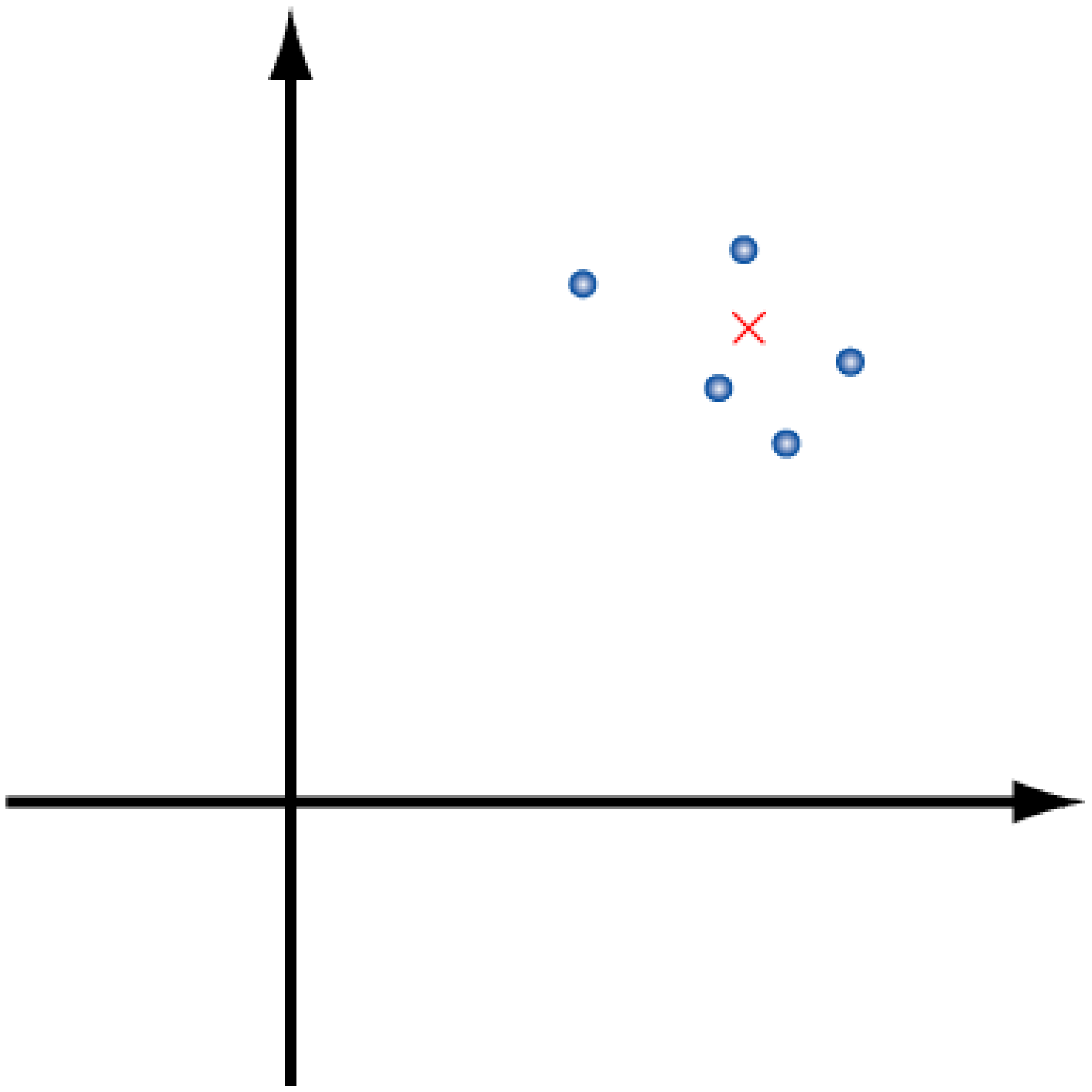
- Sample
- Select elites
- Update mean
- Update covariance
- iterate

Covariance Matrix Adaptation



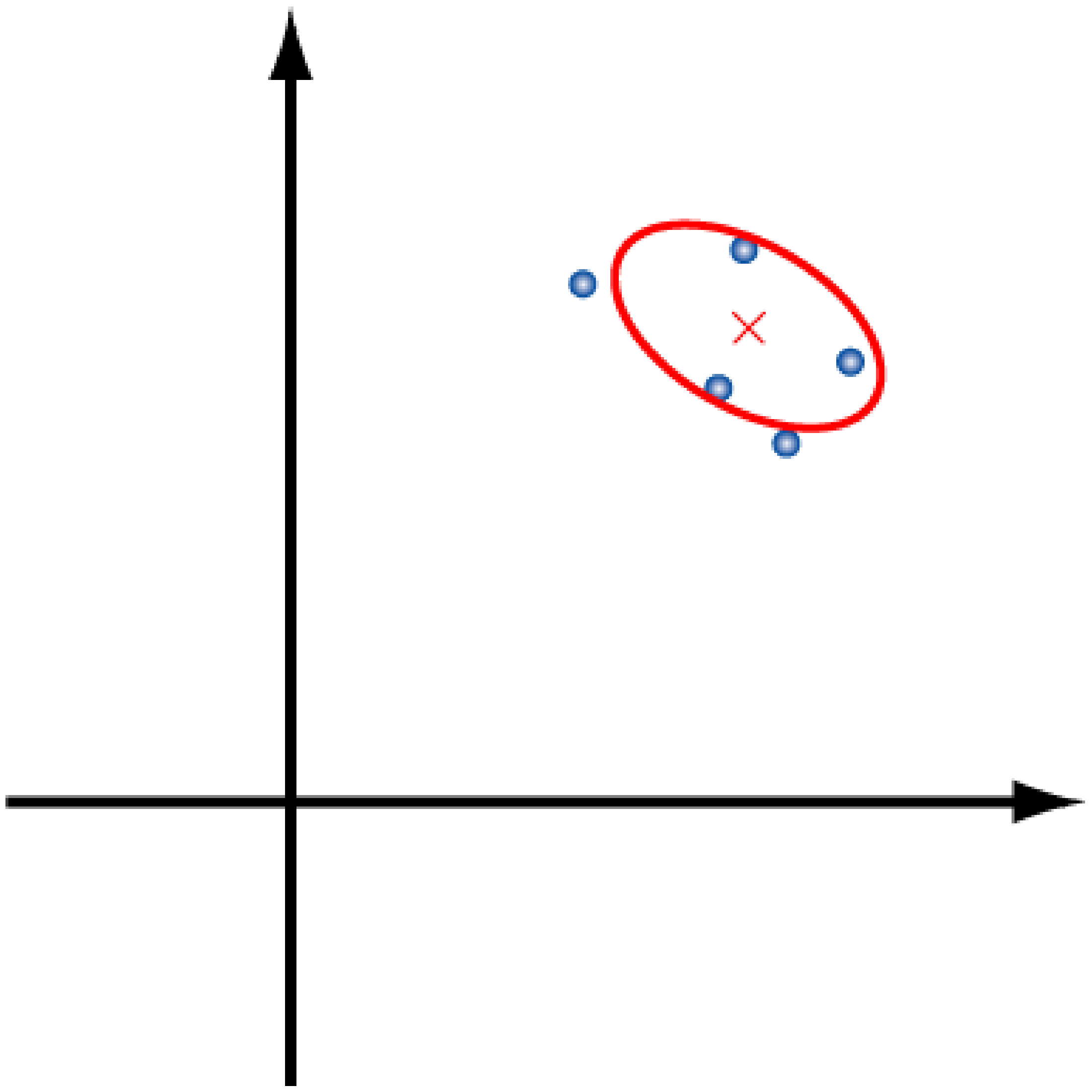
- Sample
- Select elites
- Update mean
- Update covariance
- iterate

Covariance Matrix Adaptation



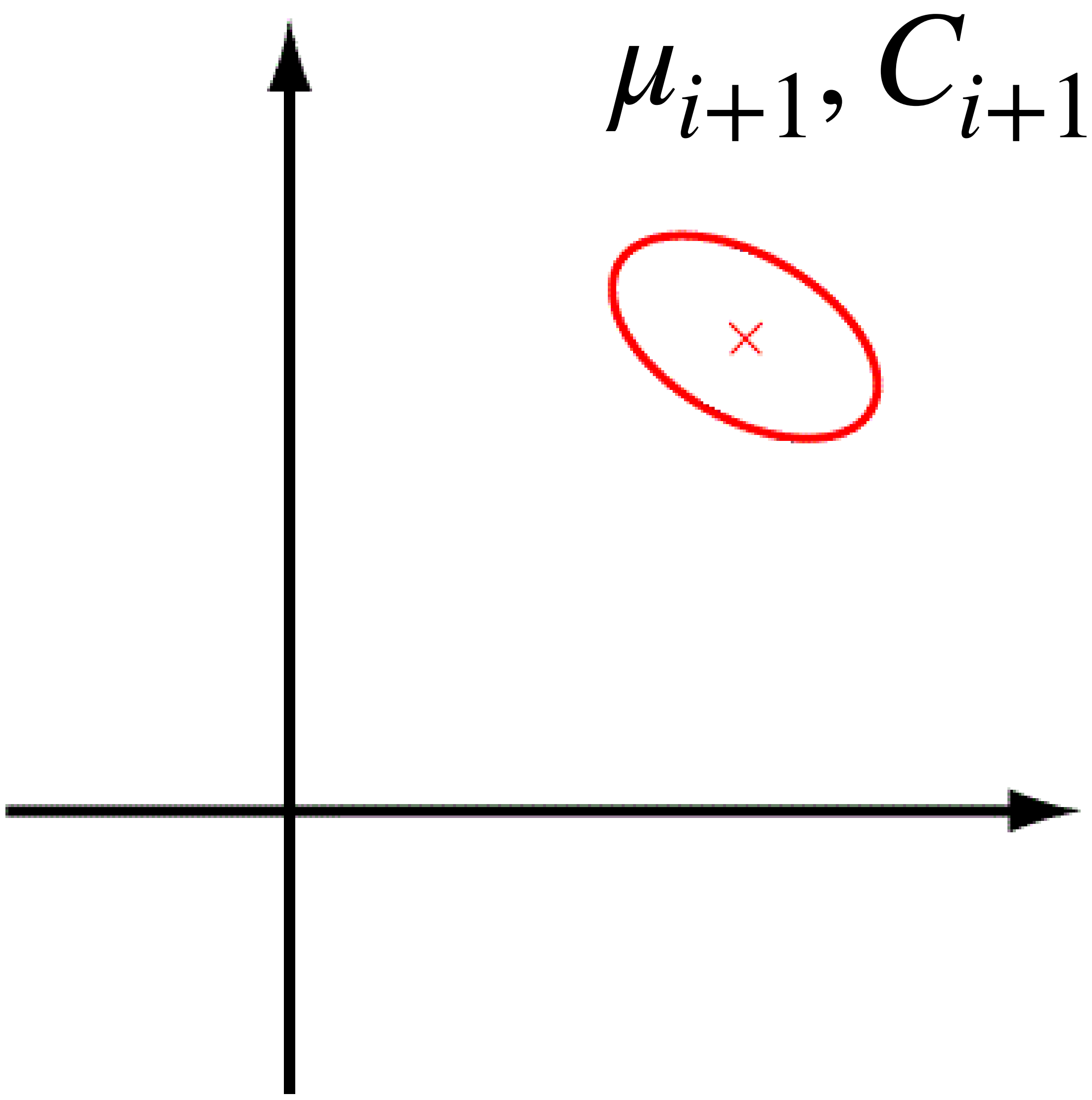
- Sample
- Select elites
- Update mean
- Update covariance
- iterate

Covariance Matrix Adaptation



- Sample
- Select elites
- Update mean
- Update covariance
- iterate

Covariance Matrix Adaptation



- Sample
- Select elites
- Update mean
- Update covariance
- iterate

CMA-ES, CEM

Work embarrassingly well in low-dimensions

Method	Mean Score	Reference
Nonreinforcement learning		
Hand-coded	631,167	Dellacherie (Fahey, 2003)
Genetic algorithm	586,103	(Böhm et al., 2004)
Reinforcement learning		
Relational reinforcement learning+kernel-based regression	≈50	Ramon and Driessens (2004)
Policy iteration	3183	Bertsekas and Tsitsiklis (1996)
Least squares policy iteration	<3000	Lagoudakis, Parr, and Littman (2002)
Linear programming + Bootstrap	4274	Farias and van Roy (2006)
Natural policy gradient	≈6800	Kakade (2001)
CE+RL	21,252	
CE+RL, constant noise	72,705	
CE+RL, decreasing noise	348,895	

István Szita and András Lörincz. "Learning Tetris using the noisy cross-entropy method". In: *Neural computation* 18.12 (2006), pp. 2936–2941

$$\mu \in \mathbb{R}^{22}$$

Approximate Dynamic Programming Finally Performs Well in the Game of Tetris

Victor Gabillon
INRIA Lille - Nord Europe,
Team SequeL, FRANCE
victor.gabillon@inria.fr

Mohammad Ghavamzadeh*
INRIA Lille - Team SequeL
& Adobe Research
mohammad.ghavamzadeh@inria.fr

Bruno Scherrer
INRIA Nancy - Grand Est,
Team Maia, FRANCE
bruno.scherrer@inria.fr

[NIPS 2013]

Question

- Evolutionary methods work well on relatively low-dimensional problems
- Can they be used to optimize deep network policies?

Policy gradients VS Evolutionary methods

We are sampling in both cases…

- PG: sampling in action space
- ES: sampling in parameter space

Policy Gradients

$$\max_{\theta} . \ U(\theta) = \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[R(\tau)\right]$$

$$\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[R(\tau)\right] \\
&= \nabla_\theta \sum_\tau P_\theta(\tau) R(\tau) \\
&= \sum_\tau \nabla_\theta P_\theta(\tau) R(\tau) \\
&= \sum_\tau P_\theta(\tau) \frac{\nabla_\theta P_\theta(\tau)}{P_\theta(\tau)} R(\tau) \\
&= \sum_\tau P_\theta(\tau) \nabla_\theta \log P_\theta(\tau) R(\tau) \\
&= \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[\nabla_\theta \log P_\theta(\tau) R(\tau)\right]
\end{aligned}$$

Sample estimate:

$$\nabla_\theta U(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log P_\theta(\tau^{(i)}) R(\tau^{(i)})$$

Evolutionary Methods

$$\max_{\mu} . \; U(\mu) = \mathbb{E}_{\theta \sim P_{\mu}(\theta)} \left[F(\theta) \right]$$

$$\begin{aligned}
\nabla_{\mu} U(\mu) &= \nabla_{\mu} \mathbb{E}_{\theta \sim P_{\mu}(\theta)} \left[F(\theta) \right] \\
&= \nabla_{\mu} \int P_{\mu}(\theta) F(\theta) d\theta \\
&= \int \nabla_{\mu} P_{\mu}(\theta) F(\theta) d\theta \\
&= \int P_{\mu}(\theta) \frac{\nabla_{\mu} P_{\mu}(\theta)}{P_{\mu}(\theta)} F(\theta) d\theta \\
&= \int P_{\mu}(\theta) \nabla_{\mu} \log P_{\mu}(\theta) F(\theta) d\theta \\
&= \mathbb{E}_{\theta \sim P_{\mu}(\theta)} \left[\nabla_{\mu} \log P_{\mu}(\theta) F(\theta) \right]
\end{aligned}$$

Sample estimate:

$$\nabla_{\mu} U(\mu) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_{\mu} \log P_{\mu}(\theta^{(i)}) F(\theta^{(i)})$$

Policy gradients VS Evolutionary methods

Considers distribution over actions

$$\max_{\theta} .\ U(\theta) = \mathbb{E}_{\tau \sim P_\theta(\tau)}\left[R(\tau)\right]$$

$$\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta \mathbb{E}_{\tau \sim P_\theta(\tau)}\left[R(\tau)\right] \\
&= \nabla_\theta \sum_\tau P_\theta(\tau) R(\tau) \\
&= \sum_\tau \nabla_\theta P_\theta(\tau) R(\tau) \\
&= \sum_\tau P_\theta(\tau) \frac{\nabla_\theta P_\theta(\tau)}{P_\theta(\tau)} R(\tau) \\
&= \sum_\tau P_\theta(\tau) \nabla_\theta \log P_\theta(\tau) R(\tau) \\
&= \mathbb{E}_{\tau \sim P_\theta(\tau)}\left[\nabla_\theta \log P_\theta(\tau) R(\tau)\right]
\end{aligned}$$

Sample estimate:

$$\nabla_\theta U(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log P_\theta(\tau^{(i)}) R(\tau^{(i)})$$

Considers distribution over policy parameters

$$\max_{\mu} .\ U(\mu) = \mathbb{E}_{\theta \sim P_\mu(\theta)}\left[F(\theta)\right]$$

$$\begin{aligned}
\nabla_\mu U(\mu) &= \nabla_\mu \mathbb{E}_{\theta \sim P_\mu(\theta)}\left[F(\theta)\right] \\
&= \nabla_\mu \int P_\mu(\theta) F(\theta) d\theta \\
&= \int \nabla_\mu P_\mu(\theta) F(\theta) d\theta \\
&= \int P_\mu(\theta) \frac{\nabla_\mu P_\mu(\theta)}{P_\mu(\theta)} F(\theta) d\theta \\
&= \int P_\mu(\theta) \nabla_\mu \log P_\mu(\theta) F(\theta) d\theta \\
&= \mathbb{E}_{\theta \sim P_\mu(\theta)}\left[\nabla_\mu \log P_\mu(\theta) F(\theta)\right]
\end{aligned}$$

Sample estimate:

$$\nabla_\mu U(\mu) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\mu \log P_\mu(\theta^{(i)}) F(\theta^{(i)})$$

Policy gradients VS Evolutionary methods

Considers distribution over actions

$$\max_{\theta} .\ U(\theta) = \mathbb{E}_{\tau \sim P_\theta(\tau)}\left[R(\tau)\right]$$

$$\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta \mathbb{E}_{\tau \sim P_\theta(\tau)}\left[R(\tau)\right] \\
&= \nabla_\theta \sum_\tau P_\theta(\tau) R(\tau) \\
&= \sum_\tau \nabla_\theta P_\theta(\tau) R(\tau) \\
&= \sum_\tau P_\theta(\tau) \frac{\nabla_\theta P_\theta(\tau)}{P_\theta(\tau)} R(\tau) \\
&= \sum_\tau P_\theta(\tau) \nabla_\theta \log P_\theta(\tau) R(\tau) \\
&= \mathbb{E}_{\tau \sim P_\theta(\tau)}\left[\nabla_\theta \log P_\theta(\tau) R(\tau)\right]
\end{aligned}$$

Sample estimate:

$$\nabla_\theta U(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\alpha_t^{(i)} \mid s_t^{(i)}) R(s_t^{(i)}, a_t^{(i)})$$

Considers distribution over policy parameters

$$\max_{\mu} .\ U(\mu) = \mathbb{E}_{\theta \sim P_\mu(\theta)}\left[F(\theta)\right]$$

$$\begin{aligned}
\nabla_\mu U(\mu) &= \nabla_\mu \mathbb{E}_{\theta \sim P_\mu(\theta)}\left[F(\theta)\right] \\
&= \nabla_\mu \int P_\mu(\theta) F(\theta) d\theta \\
&= \int \nabla_\mu P_\mu(\theta) F(\theta) d\theta \\
&= \int P_\mu(\theta) \frac{\nabla_\mu P_\mu(\theta)}{P_\mu(\theta)} F(\theta) d\theta \\
&= \int P_\mu(\theta) \nabla_\mu \log P_\mu(\theta) F(\theta) d\theta \\
&= \mathbb{E}_{\theta \sim P_\mu(\theta)}\left[\nabla_\mu \log P_\mu(\theta) F(\theta)\right]
\end{aligned}$$

Sample estimate:

$$\boxed{\nabla_\mu U(\mu) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\mu \log P_\mu(\theta^{(i)}) F(\theta^{(i)})}$$

A concrete example

- Suppose $\theta \sim P_\mu(\theta)$ is a Gaussian distribution with mean μ, and covariance matrix $\sigma^2 I$

$$\log P_\mu(\theta) = -\frac{||\theta - \mu||^2}{2\sigma^2} + \text{const}$$

$$\nabla_\mu \log P_\mu(\theta) = \frac{\theta - \mu}{\sigma^2}$$

A concrete example

Suppose $\theta \sim P_\mu(\theta)$ is a Gaussian distribution with mean μ, and covariance matrix $\sigma^2 I$

$$\log P_\mu(\theta) = -\frac{||\theta - \mu||^2}{2\sigma^2} + \mathrm{const}$$

$$\nabla_\mu \log P_\mu(\theta) = \frac{\theta - \mu}{\sigma^2}$$

If we draw two parameter samples θ_1, θ_2, and obtain two trajectories τ_1, τ_2:

$$\mathbb{E}_{\theta \sim P_\mu(\theta)}\left[\nabla_\mu \log P_\mu(\theta) R(\tau)\right] \approx \frac{1}{2}\left[R(\tau_1)\frac{\theta_1 - \mu}{\sigma^2} + R(\tau_2)\frac{\theta_2 - \mu}{\sigma^2}\right]$$

Sampling parameter vectors

- Suppose $\theta \sim P_\mu(\theta)$ is a Gaussian distribution with mean μ, and covariance matrix $\sigma^2 I$

Imagine we have access to random vectors $\epsilon \sim \mathcal{N}(0,I)$

$$\theta_1 = \mu + \sigma * \epsilon_1,\ \epsilon_1 \sim \mathcal{N}(0,I)$$

$$\theta_2 = \mu + \sigma * \epsilon_2,\ \epsilon_2 \sim \mathcal{N}(0,I)$$

The theta samples have the desired mean and variance

A concrete example

- Suppose $\theta \sim P_\mu(\theta)$ is a Gaussian distribution with mean μ, and covariance matrix $\sigma^2 I$

$$\log P_\mu(\theta) = -\frac{||\theta - \mu||^2}{2\sigma^2} + \text{const}$$

$$\nabla_\mu \log P_\mu(\theta) = \frac{\theta - \mu}{\sigma^2}$$

A concrete example

- Suppose $\theta \sim P_\mu(\theta)$ is a Gaussian distribution with mean μ, and covariance matrix $\sigma^2 I$

$$\log P_\mu(\theta) = -\frac{||\theta - \mu||^2}{2\sigma^2} + \text{const}$$

$$\nabla_\mu \log P_\mu(\theta) = \frac{\theta - \mu}{\sigma^2}$$

- If we draw two parameter samples θ_1, θ_2, and obtain two trajectories τ_1, τ_2:

$$\mathbb{E}_{\theta \sim P_\mu(\theta)}\left[\nabla_\mu \log P_\mu(\theta) R(\tau)\right] \approx \frac{1}{2}\left[R(\tau_1)\frac{\theta_1 - \mu}{\sigma^2} + R(\tau_2)\frac{\theta_2 - \mu}{\sigma^2}\right]$$

$$\approx \frac{1}{2\sigma}\left[R(\tau_1)\epsilon_1 + R(\tau_2)\epsilon_2\right]$$

$$\theta_1 = \mu + \sigma * \epsilon_1,\ \epsilon_1 \sim \mathcal{N}(0,I)$$

$$\theta_2 = \mu + \sigma * \epsilon_2,\ \epsilon_2 \sim \mathcal{N}(0,I)$$

Natural Evolutionary Strategies

Algorithm 1 Evolution Strategies

1: **Input:** Learning rate α, noise standard deviation σ, initial policy parameters θ_0
2: **for** $t = 0, 1, 2, \ldots$ **do**
3: Sample $\epsilon_1, \ldots \epsilon_n \sim \mathcal{N}(0, I)$
4: Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
5: Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \epsilon_i$
6: **end for**

Connection to Finite Differences

- Antithetic sampling
  - Sample a pair of policies with mirror noise $(\theta_+ = \mu + \sigma\epsilon, \theta_- = \mu - \sigma\epsilon)$

Connection to Finite Differences

- Antithetic sampling
  - Sample a pair of policies with mirror noise $(\theta_+ = \mu + \sigma\epsilon, \theta_- = \mu - \sigma\epsilon)$
  - Get a pair of rollouts from environment (τ_+, τ_-)

Connection to Finite Differences

- Antithetic sampling
  - Sample a pair of policies with mirror noise $(\theta_+ = \mu + \sigma\epsilon, \theta_- = \mu - \sigma\epsilon)$
  - Get a pair of rollouts from environment (τ_+, τ_-)
  - SPSA: Finite Difference with random direction

$$\begin{aligned}
\nabla_\mu \mathbb{E}\left[R(\tau)\right] &\approx \frac{1}{2}\left[R(\tau_+)\frac{\theta_+ - \mu}{\sigma^2} + R(\tau_-)\frac{\theta_- - \mu}{\sigma^2}\right] \\
&= \frac{1}{2}\left[R(\tau_+)\frac{\sigma\epsilon}{\sigma^2} + R(\tau_-)\frac{-\sigma\epsilon}{\sigma^2}\right] \\
&= \frac{\epsilon}{2\sigma}\left[R(\tau_+) - R(\tau_-)\right]
\end{aligned}$$

Connection to Finite Differences

- Antithetic sampling
  - Sample a pair of policies with mirror noise $(\theta_+ = \mu + \sigma\epsilon, \theta_- = \mu - \sigma\epsilon)$
  - Get a pair of rollouts from environment (τ_+, τ_-)
  - SPSA: Finite Difference with random direction

$$
\begin{aligned}
\nabla_\mu \mathbb{E}\left[R(\tau)\right] &\approx \frac{1}{2}\left[R(\tau_+)\frac{\theta_+ - \mu}{\sigma^2} + R(\tau_-)\frac{\theta_- - \mu}{\sigma^2}\right] \\
&= \frac{1}{2}\left[R(\tau_+)\frac{\sigma\epsilon}{\sigma^2} + R(\tau_-)\frac{-\sigma\epsilon}{\sigma^2}\right] \\
&= \frac{\epsilon}{2\sigma}\left[R(\tau_+) - R(\tau_-)\right]
\end{aligned}
$$

vs

Finite Difference

$$
\frac{\partial U}{\partial \theta_j}(\theta) = \frac{U(\theta + \epsilon e_j) - U(\theta - \epsilon e_j)}{2\epsilon}
$$

Finite Differences

We can compute the gradient g using standard finite difference methods, as follows:

$$\frac{\partial U}{\partial \theta_j}(\theta) = \frac{U(\theta + \epsilon e_j) - U(\theta - \epsilon e_j)}{2\epsilon}$$

Where:

$$e_j = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \begin{array}{l} \\ \\ \\ \\ \leftarrow \quad j\text{'th entry} \\ \\ \\ \\ \end{array}$$

Evolution Strategies as a Scalable Alternative to Reinforcement Learning

Tim Salimans **Jonathan Ho** **Xi Chen** **Szymon Sidor** **Ilya Sutskever**
OpenAI

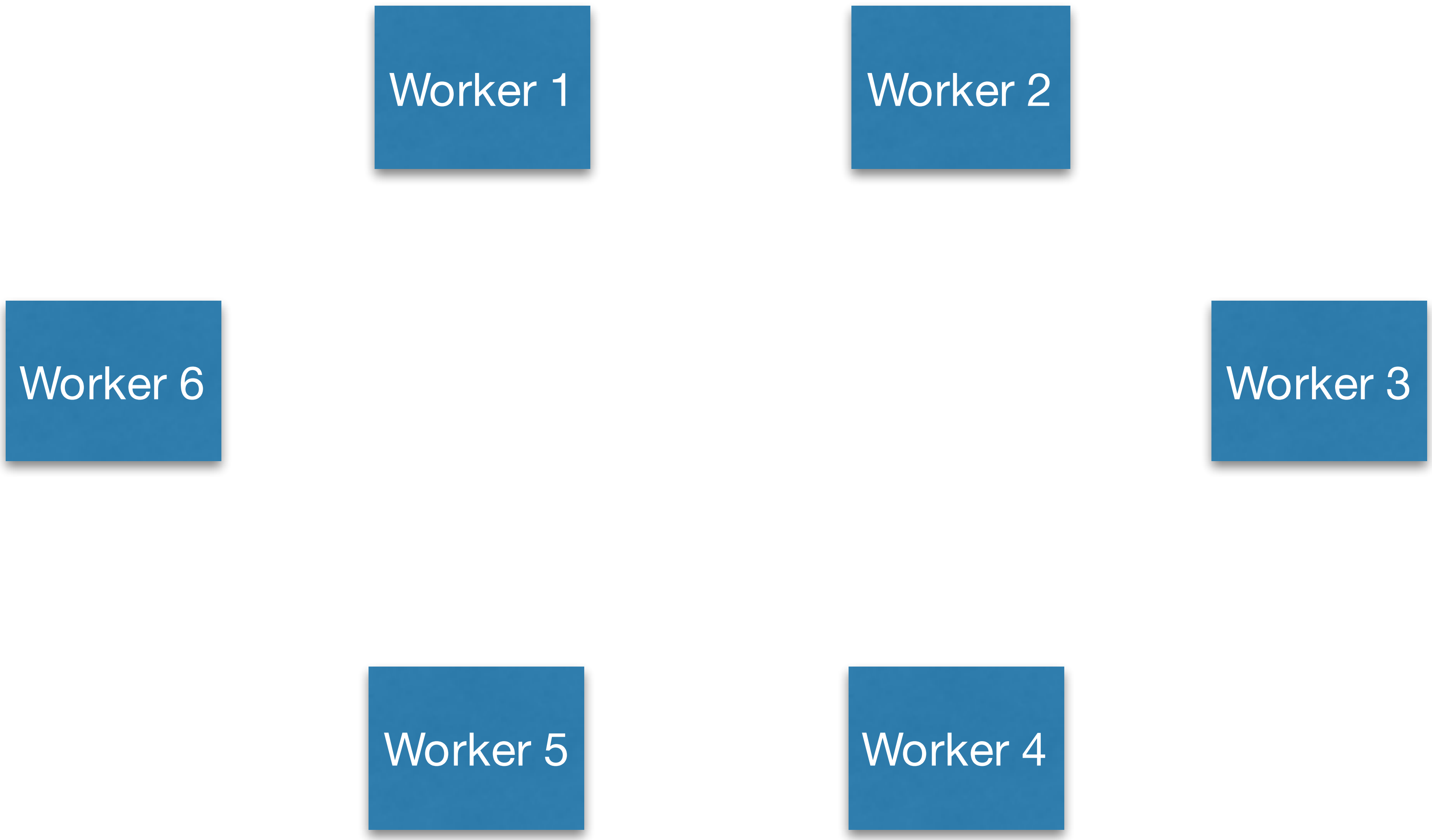
Algorithm 1 Evolution Strategies

1: **Input:** Learning rate α, noise standard deviation σ, initial policy parameters θ_0
2: **for** $t = 0, 1, 2, \ldots$ **do**
3: Sample $\epsilon_1, \ldots \epsilon_n \sim \mathcal{N}(0, I)$
4: Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
5: Set $\theta_{t+1} \leftarrow \theta_t + \alpha\frac{1}{n\sigma}\sum_{i=1}^{n} F_i\epsilon_i$
6: **end for**

Main contribution:

- Parallelization with a need for tiny only cross-worker communication

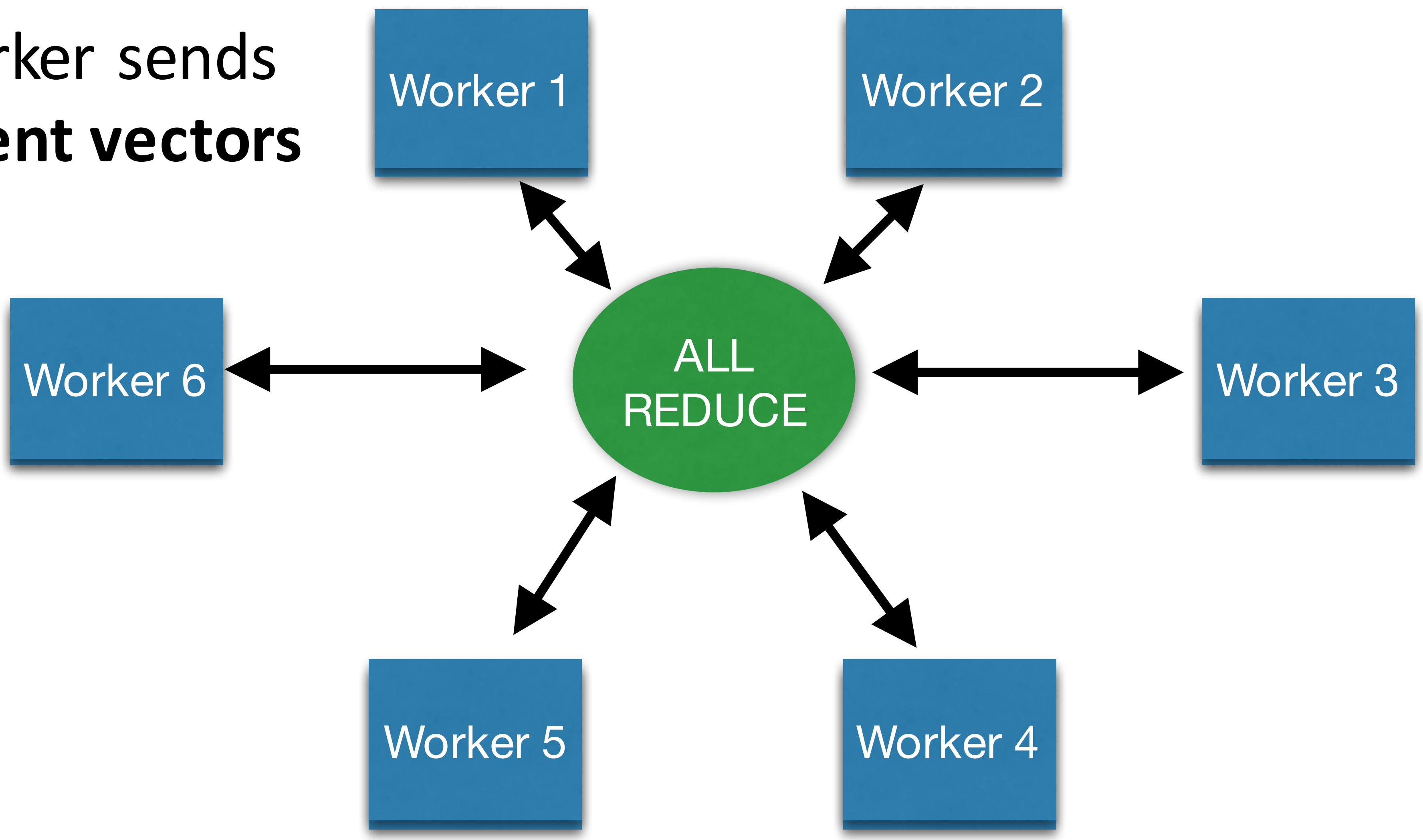
Distributed SGD



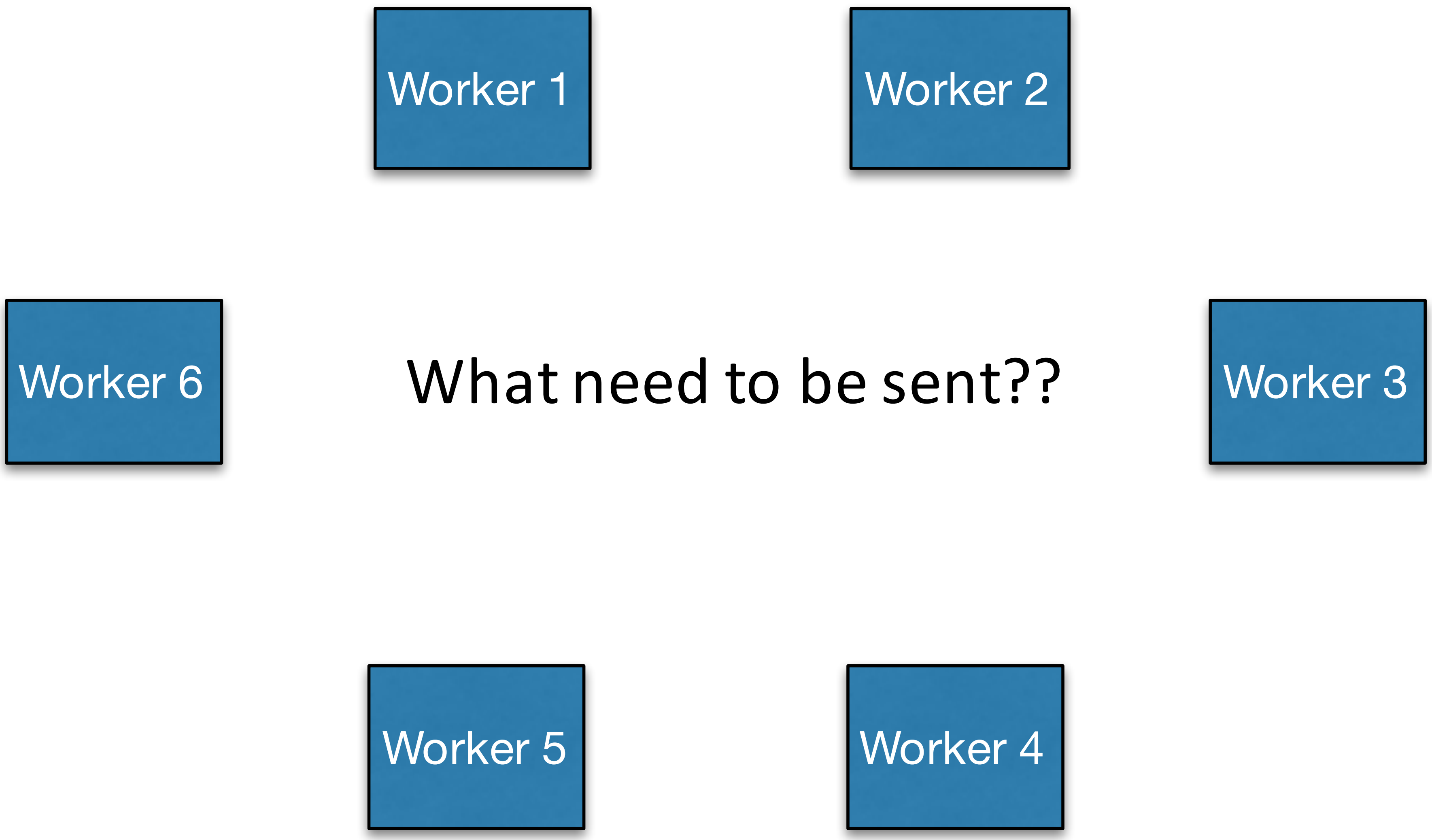
Used in Asynchronous RL!

Distributed SGD

Each worker sends
big gradient vectors



Distributed Evolution



Distributed Evolution

Algorithm 1 Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
for $t = 0, 1, 2, \ldots$ **do**
    Sample $\epsilon_1, \ldots \epsilon_n \sim \mathcal{N}(0, I)$
    Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
    Set $\theta_{t+1} \leftarrow \theta_t + \alpha\frac{1}{n\sigma}\sum_{i=1}^{n} F_i\epsilon_i$
end for

Worker 5

Worker 4

Distributed Evolution

Worker 1

Worker 2

Worker 6

θ and R(τ)?

Worker 3

θ is big!

but $\theta = \mu + \sigma\epsilon$

Same for all workers

Only need seed of random number generator!

Distributed Evolution

Algorithm 2 Parallelized Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
Initialize: n workers with known random seeds, and initial parameters θ_0
for $t = 0, 1, 2, \ldots$ **do**
  for each worker $i = 1, \ldots, n$ **do**
    Sample $\epsilon_i \sim \mathcal{N}(0, I)$
    Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$
  end for
  Send all scalar returns F_i from each worker to every other worker
  for each worker $i = 1, \ldots, n$ **do**
    Reconstruct all perturbations ϵ_j for $j = 1, \ldots, n$
    Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{j=1}^{n} F_j \epsilon_j$
  end for
end for

[Salimans, Ho, Chen, Sutskever, 2017]

Distributed Evolution

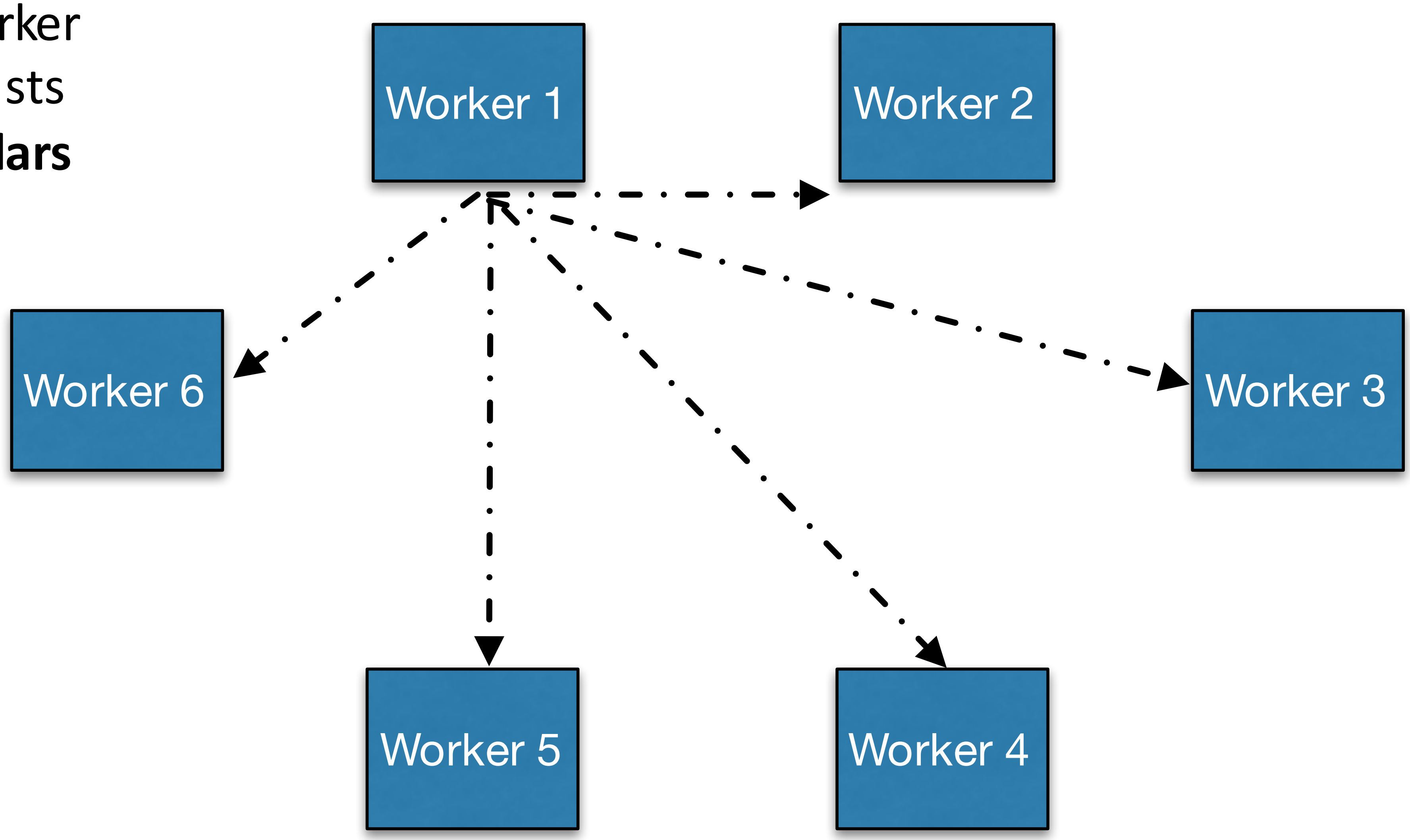
Algorithm 2 Parallelized Evolution Strategies

```
Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
Initialize: n workers with known random seeds, and initial parameters θ_0
for t = 0, 1, 2, . . . do
    for each worker i = 1, . . . , n do
        Sample ε_i ~ N(0, I)
        Compute returns F_i = F(θ_t + σε_i)
    end for
    Send all scalar returns F_i from each worker to every other worker
    for each worker i = 1, . . . , n do
        Reconstruct all perturbations ε_j for j = 1, . . . , n
        Set θ_{t+1} ← θ_t + α (1/(nσ)) Σ_{j=1}^{n} F_j ε_j
    end for
end for
```

[Salimans, Ho, Chen, Sutskever, 2017]

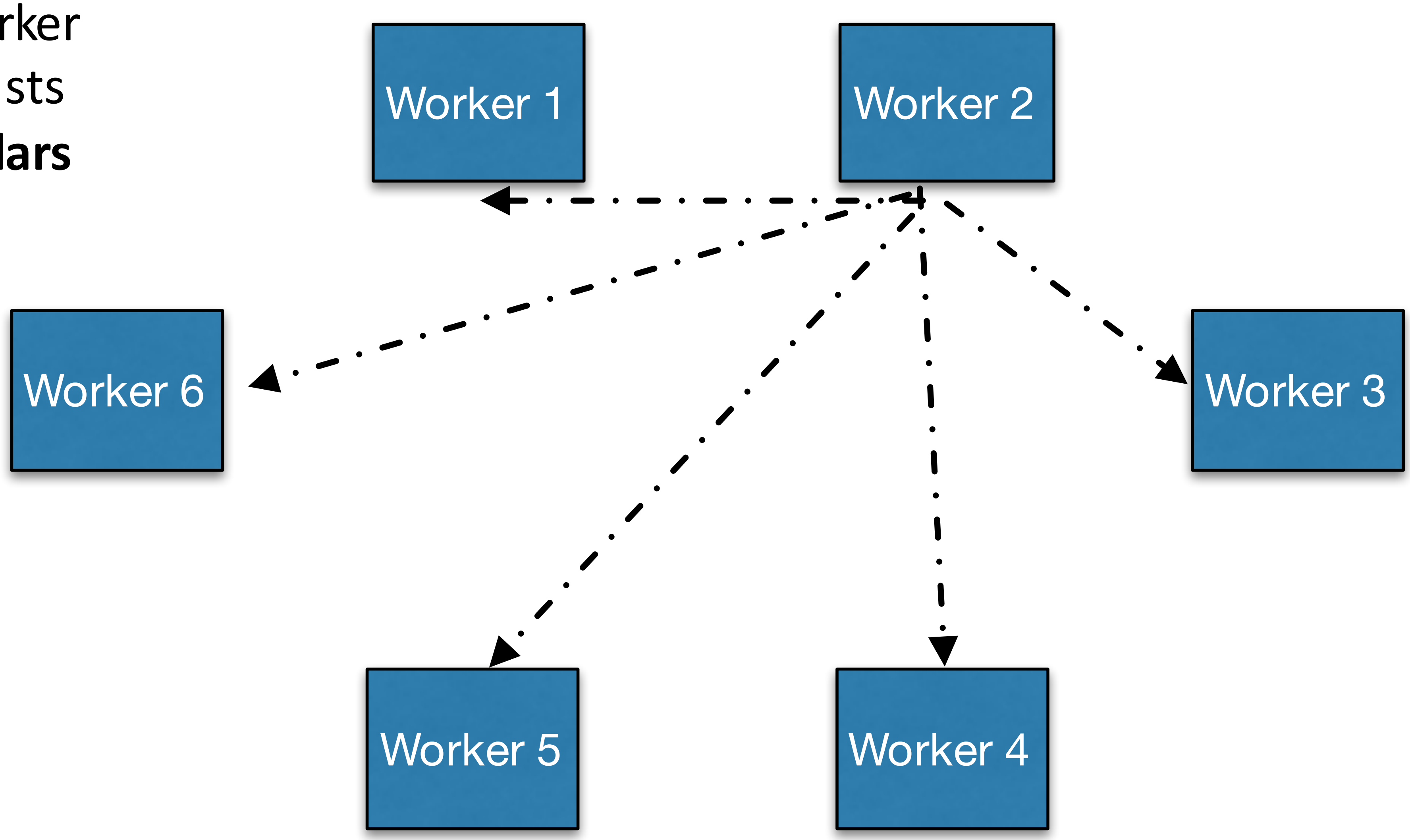
Distributed Evolution

Each worker
broadcasts
tiny scalars



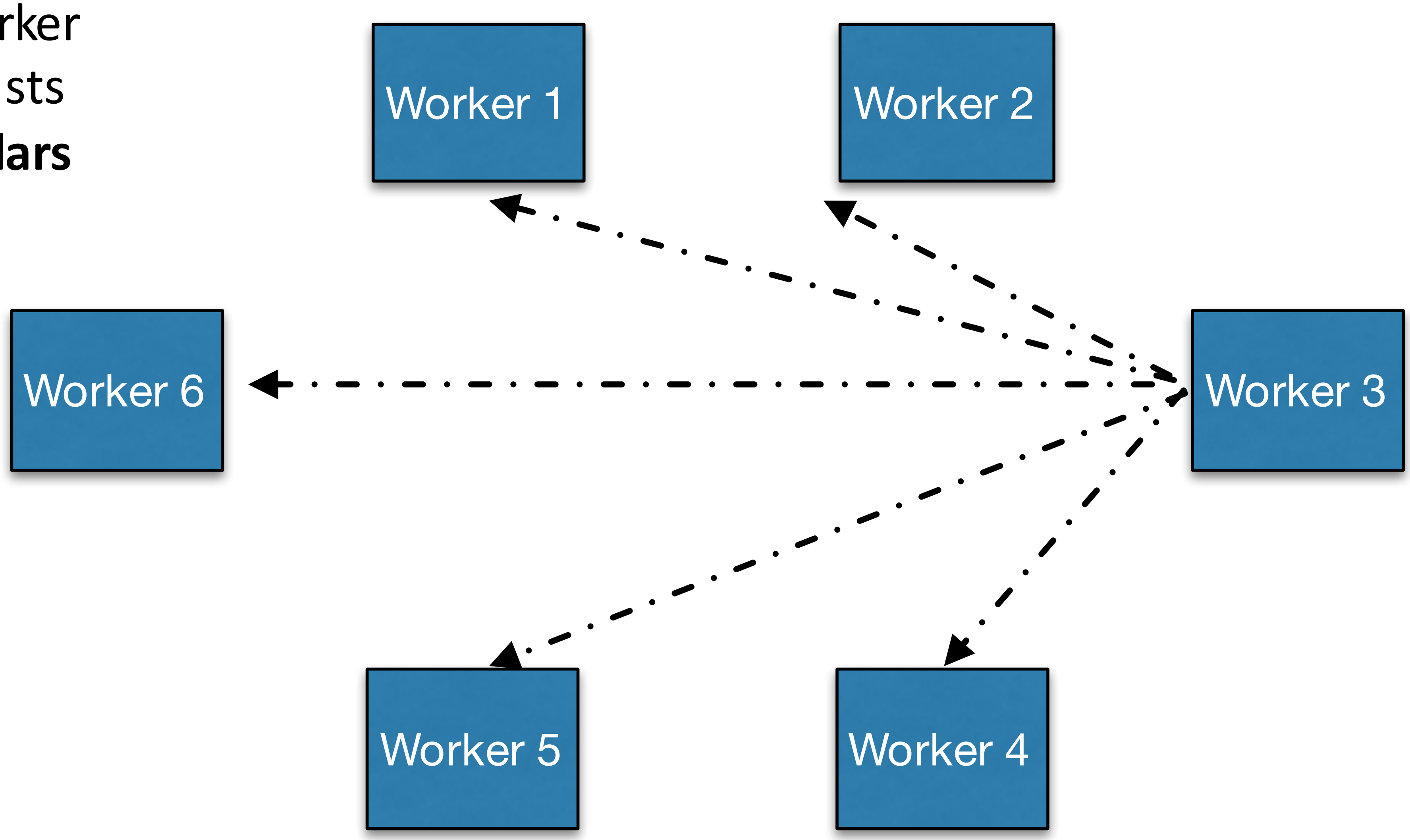
Distributed Evolution

Each worker
broadcasts
tiny scalars



Distributed Evolution

Each worker
broadcasts
tiny scalars



Distributed Evolution Scales Very Well :-)

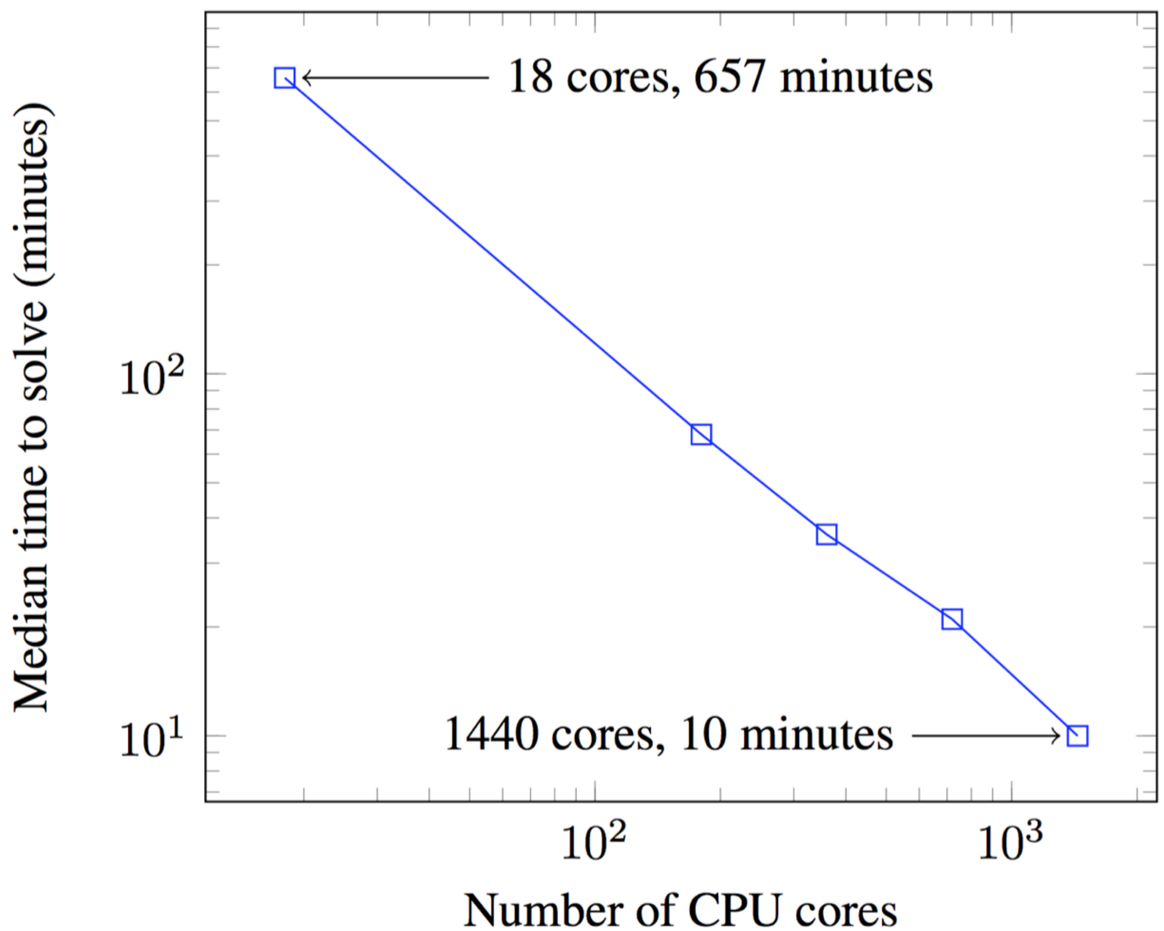
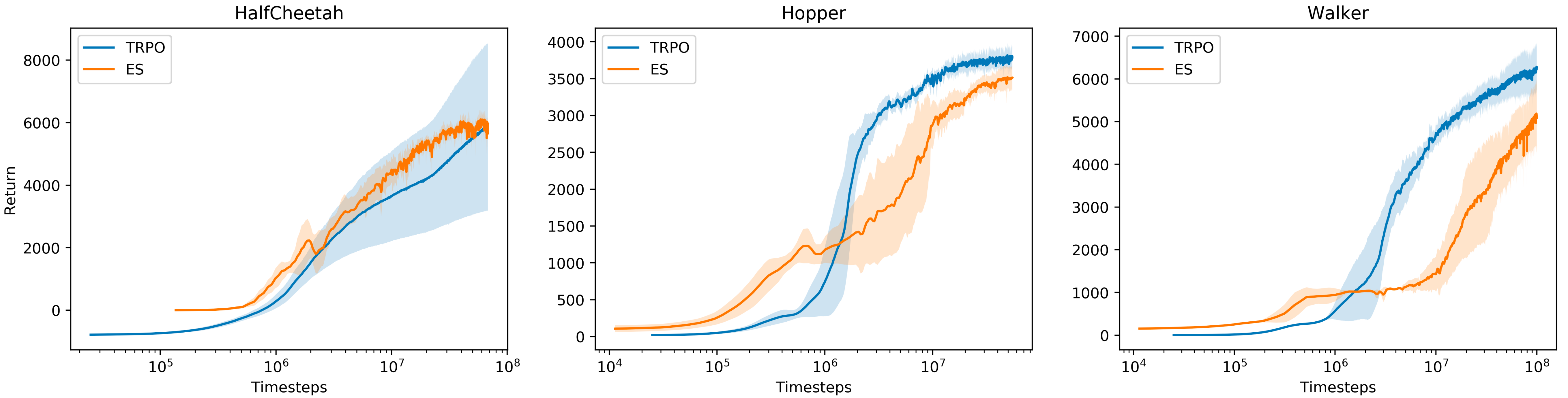


Figure 1. Time to reach a score of 6000 on 3D Humanoid with different number of CPU cores. Experiments are repeated 7 times and median time is reported.

[Salimans, Ho, Chen, Sutskever, 2017]

Distributed Evolution Requires More Samples :-(



[Salimans, Ho, Chen, Sutskever, 2017]

Population Based Training of Neural Networks

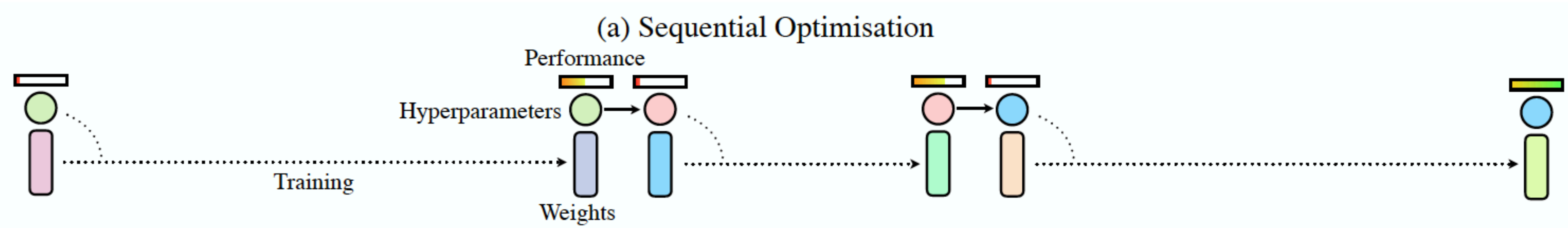
Max Jaderberg **Valentin Dalibard** **Simon Osindero** **Wojciech M. Czarnecki**

Jeff Donahue **Ali Razavi** **Oriol Vinyals** **Tim Green** **Iain Dunning**

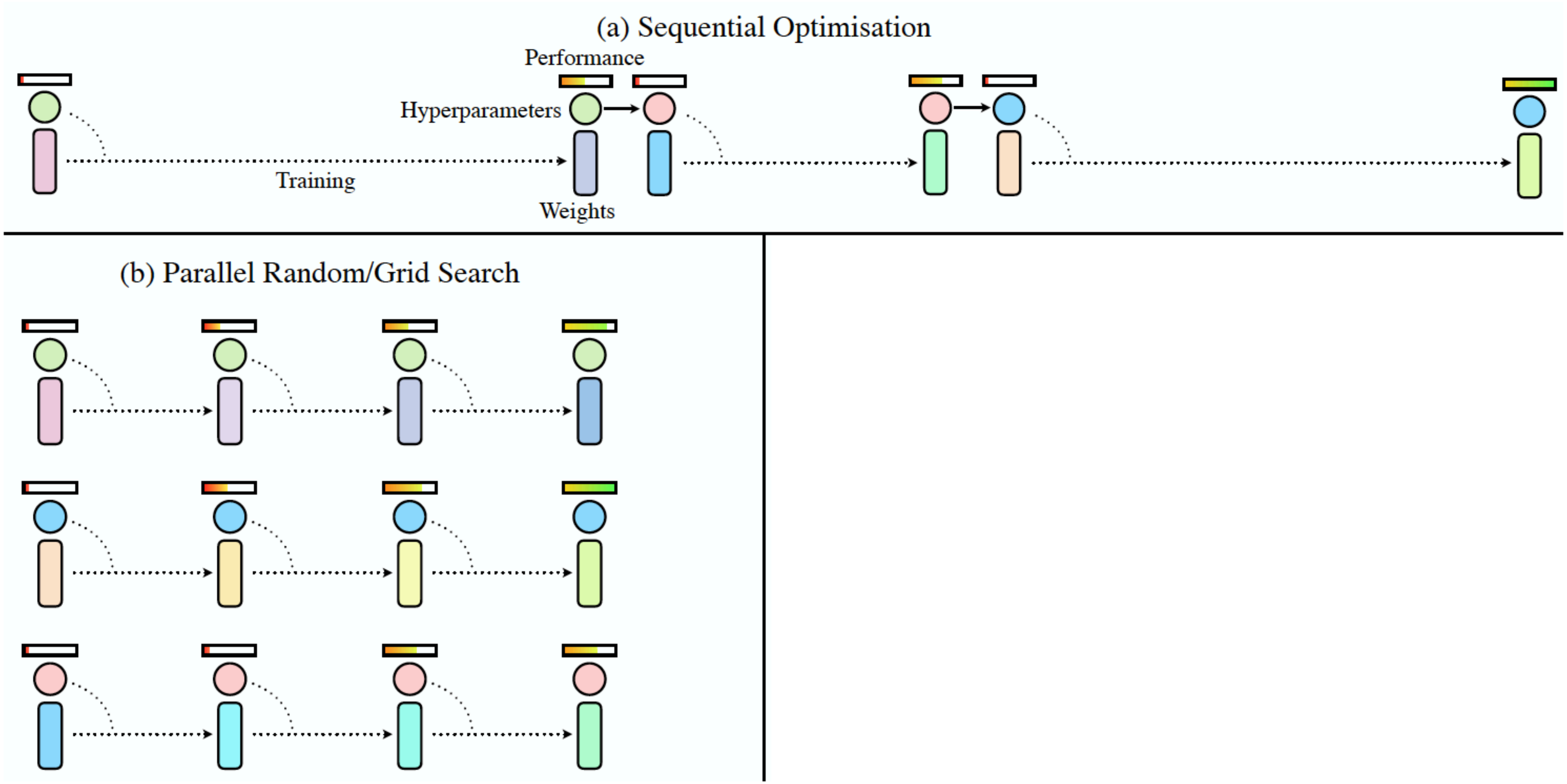
Karen Simonyan **Chrisantha Fernando** **Koray Kavukcuoglu**

DeepMind, London, UK

Searching for Hyperparameters



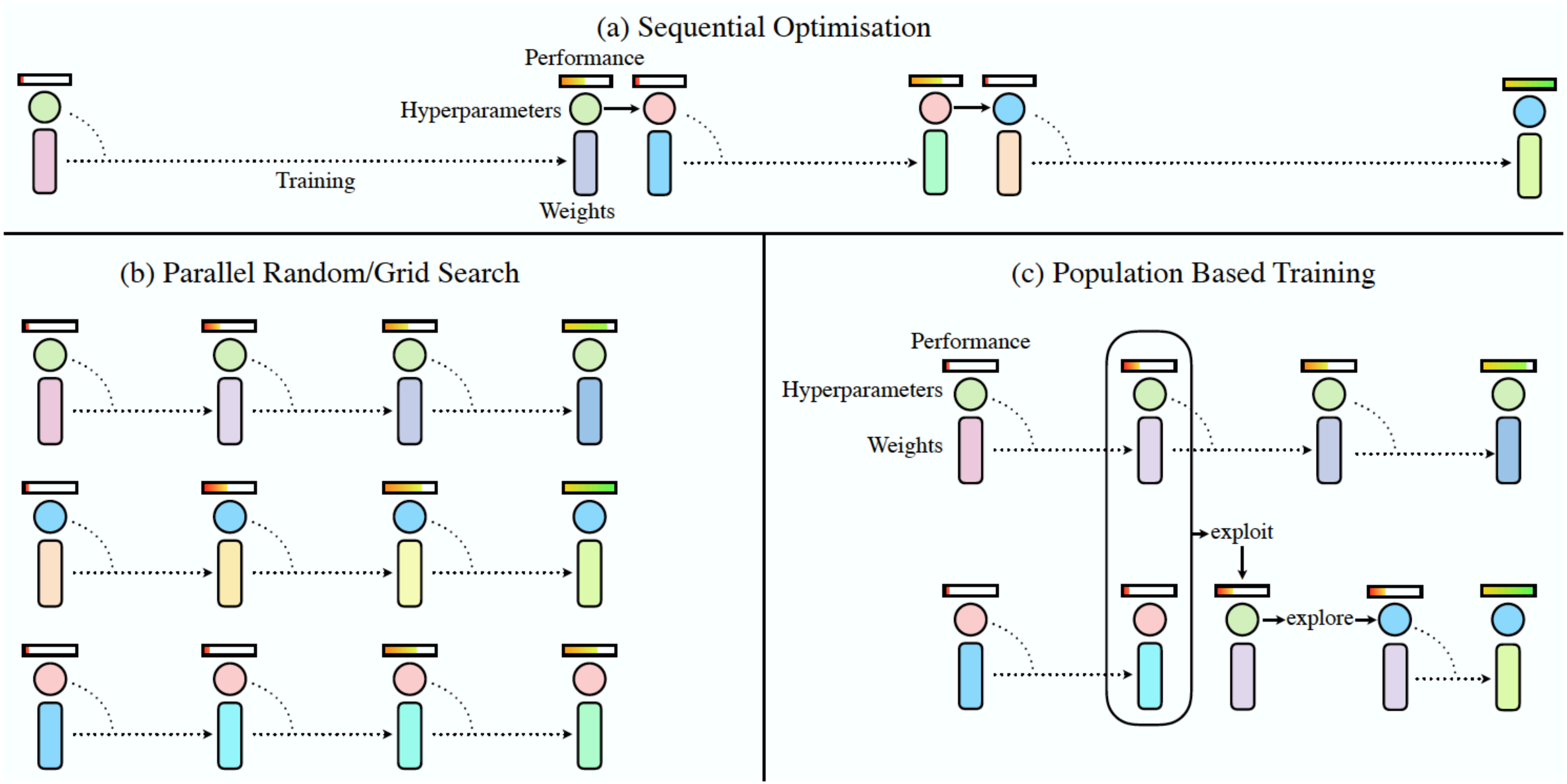
Searching for Hyperparameters

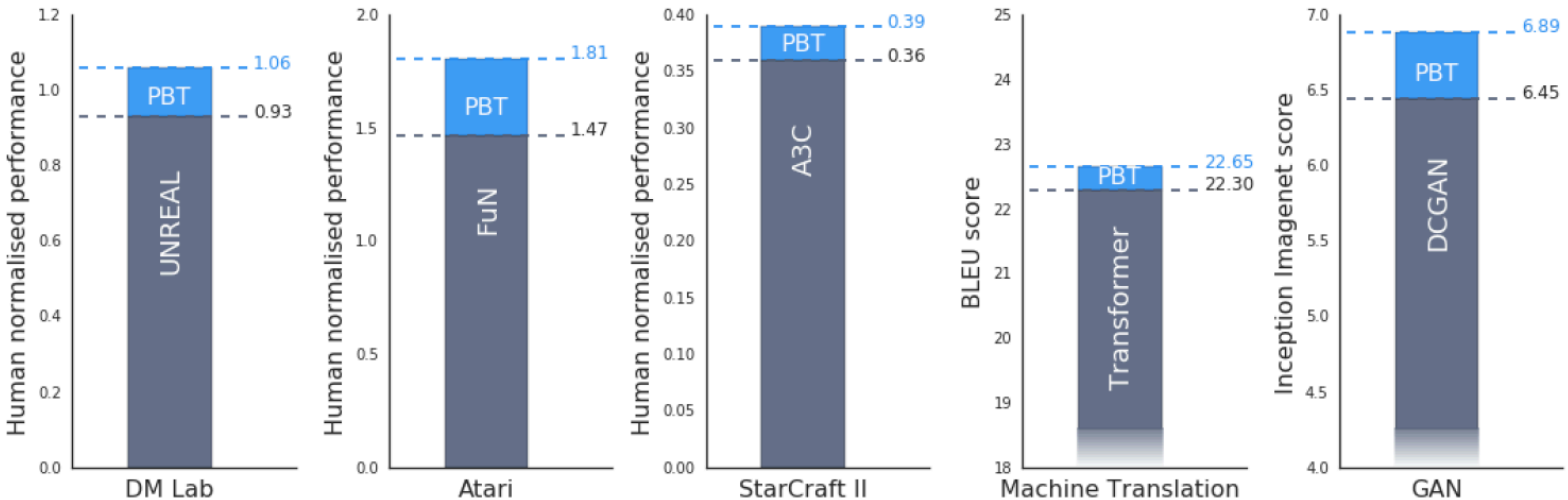


Algorithm 1 Population Based Training (PBT)

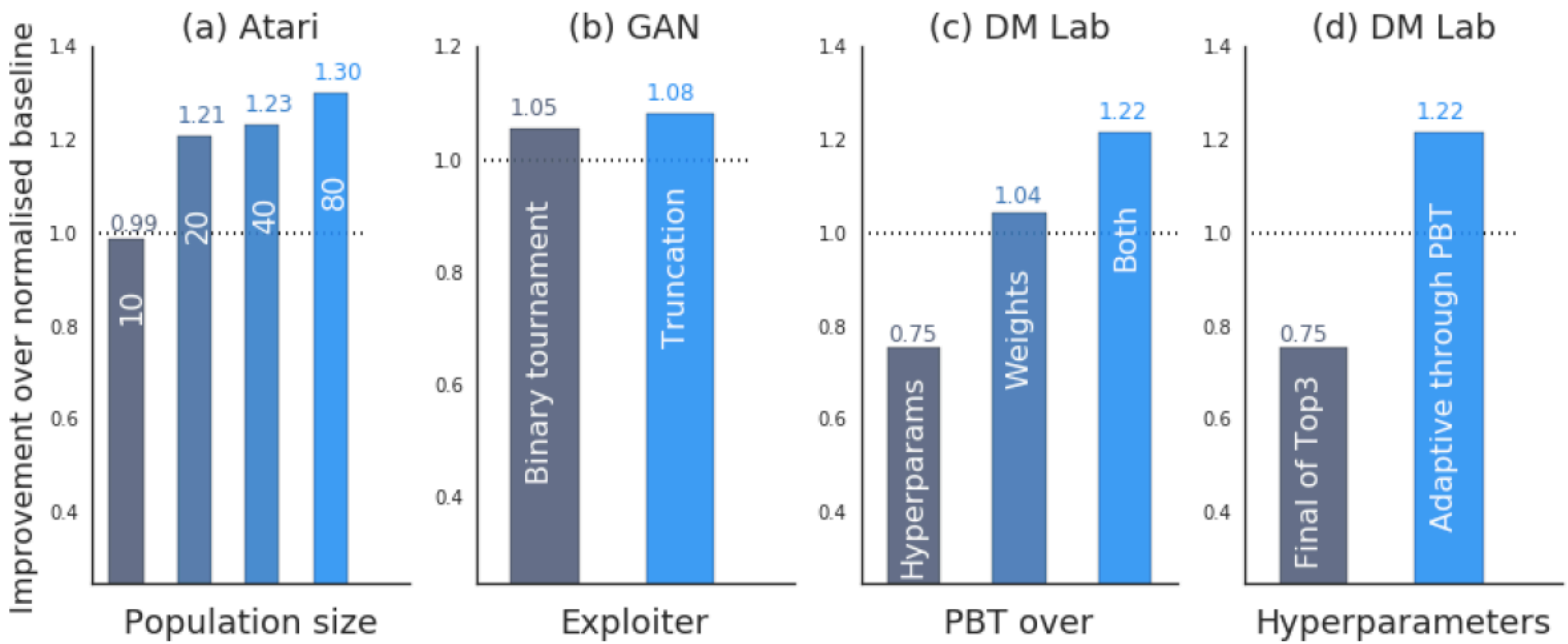
procedure TRAIN($\mathcal{P}$) ▷ initial population $\mathcal{P}$
    for (θ, h, p, t) $\in \mathcal{P}$ (asynchronously in parallel) **do**
        while not end of training **do**
            $\theta \leftarrow \texttt{step}(\theta|h)$ ▷ one step of optimisation using hyperparameters h
            $p \leftarrow \texttt{eval}(\theta)$ ▷ current model evaluation
            if $\texttt{ready}(p, t, \mathcal{P})$ **then**
                $h', \theta' \leftarrow \texttt{exploit}(h, \theta, p, \mathcal{P})$ ▷ use the rest of population to find better solution
                if $\theta \neq \theta'$ **then**
                    $h, \theta \leftarrow \texttt{explore}(h', \theta', \mathcal{P})$ ▷ produce new hyperparameters h
                    $p \leftarrow \texttt{eval}(\theta)$ ▷ new model evaluation
                end if
            end if
            update $\mathcal{P}$ with new $(\theta, h, p, t + 1)$ ▷ update population
        end while
    end for
    return θ with the highest p in $\mathcal{P}$
end procedure

Searching for Hyperparameters





Human normalised performance
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1.06
PBT
0.93
UNREAL
DM Lab
Human normalised performance
2.0
1.5
1.0
0.5
0.0
1.81
PBT
1.47
FuN
Atari
Human normalised performance
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.39
PBT
0.36
A3C
StarCraft II
BLEU score
25
24
23
22
21
20
19
18
22.65
PBT
22.30
Transformer
Machine Translation
Inception Imagenet score
7.0
6.5
6.0
5.5
5.0
4.5
4.0
6.89
PBT
6.45
DCGAN
GAN


(a) Atari
(b) GAN
(c) DM Lab
(d) DM Lab
Improvement over normalised baseline
1.4
1.2
1.0
0.8
0.6
0.4
0.99
1.21
1.23
1.30
10
20
40
80
Population size
1.05
1.08
Binary tournament
Truncation
Exploiter
0.75
1.04
1.22
Hyperparams
Weights
Both
PBT over
0.75
1.22
Final of Top3
Adaptive through PBT
Hyperparameters