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
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**
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

\[
\Delta \rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial \rho} z_t
\]

The RL policy won more than 80% of games against the SL policy.
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 $z$. 
Selection: selecting actions within the expanded tree

Tree policy

\[ a_t = \arg\max_a (Q(s_t, a) + u(s_t, a)) \]

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

- \( a_t \) - 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
Expansion: when reaching a leaf, play the action with highest score from $p_o$

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

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

- \(v_\theta\) - value from value function of board position \(s_L\)
- \(z_L\) - Reward from fast rollout \(p_\pi\)
  - Played until terminal step
- \(\lambda\) - mixing parameter
  - Empirical
**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
• So far, MCTS was used for online planning to select moves at test time.
• AlphaGoZero uses it during training instead.
• 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

\[ \pi(a | s) \]

\[ \hat{v}(s) \]
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

\[
\max_{\theta} U(\theta) = \max_{\theta} \mathbb{E} \left[ R(\tau) \mid \pi_{\theta}, \mu_0(s_0) \right] = \max_{\theta} \mathbb{E} \left[ \sum_{t=0}^{T} R(s_t) \mid \pi_{\theta}, \mu_0(s) \right]
\]
\[
\max_{\theta} U(\theta) = \mathbb{E} \left[ R(\tau) \mid \pi_\theta, \mu_0(s_0) \right]
\]
Policy Optimization in the RL Landscape

\[
\max_{\theta} U(\theta) = \mathbb{E} \left[ R(\tau) \mid \pi_\theta, \mu_0(s_0) \right]
\]
No information regarding the structure of the state space or the reward
Evolutionary methods

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

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…
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 $\theta_i \sim N(\mu, \text{diag}(\sigma^2))$

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

Select the top k% of $\theta$, and fit a new diagonal Gaussian to those samples. Update $\mu, \sigma$

**endfor**
Covariance Matrix Adaptation

Let’s consider our parameters to be sampled from a multivariate Gaussian. We will evolve this Gaussian towards samples 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

\[ \mu_{i+1}, C_{i+1} \]
CMA-ES, CEM

Work embarrassingly well in low-dimensions

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

Approximate Dynamic Programming Finally Performs Well in the Game of Tetris


\[ \mu \in \mathbb{R}^{22} \]

[NIPS 2013]
Question

• Evolutionary methods work well on relatively low-dimensional problems

• Can they be used to optimize deep network policies?
We are sampling in both cases…

- **PG**: sampling in action space
- **ES**: sampling in parameter space
\[ \max_\theta U(\theta) = \mathbb{E}_{\tau \sim P_\theta(\tau)} [R(\tau)] \]
\[
\nabla_\theta U(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim P_\theta(\tau)} [R(\tau)] \\
= \nabla_\theta \sum_\tau P_\theta(\tau) R(\tau) \\
= \sum_\tau \nabla_\theta P_\theta(\tau) R(\tau) \\
= \sum_\tau P_\theta(\tau) \frac{\nabla_\theta P_\theta(\tau)}{P_\theta(\tau)} R(\tau) \\
= \sum_\tau P_\theta(\tau) \nabla_\theta \log P_\theta(\tau) R(\tau) \\
= \mathbb{E}_{\tau \sim P_\theta(\tau)} [ \nabla_\theta \log P_\theta(\tau) R(\tau)]
\]

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

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

\[
\nabla_{\mu} U(\mu) = \nabla_{\mu} \mathbb{E}_{\theta \sim \mu(\theta)} \left[ F(\theta) \right]
\]

\[
= \nabla_{\mu} \int P_{\mu}(\theta) F(\theta) d\theta
\]

\[
= \int \nabla_{\mu} P_{\mu}(\theta) F(\theta) d\theta
\]

\[
= \int P_{\mu}(\theta) \frac{\nabla_{\mu} P_{\mu}(\theta)}{P_{\mu}(\theta)} F(\theta) d\theta
\]

\[
= \int P_{\mu}(\theta) \nabla_{\mu} \log P_{\mu}(\theta) F(\theta) d\theta
\]

\[
= \mathbb{E}_{\theta \sim \mu(\theta)} \left[ \nabla_{\mu} \log P_{\mu}(\theta) F(\theta) \right]
\]

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

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

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

$$= \nabla_\theta \sum_{\tau} P_\theta(\tau) R(\tau)$$

$$= \sum_{\tau} \nabla_\theta P_\theta(\tau) R(\tau)$$

$$= \sum_{\tau} P_\theta(\tau) \frac{\nabla_\theta P_\theta(\tau)}{P_\theta(\tau)} R(\tau)$$

$$= \sum_{\tau} P_\theta(\tau) \nabla_\theta \log P_\theta(\tau) R(\tau)$$

$$= \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[ \nabla_\theta \log P_\theta(\tau) R(\tau) \right]$$

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

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

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

$$= \nabla_\mu \int P_\mu(\theta) F(\theta) d\theta$$

$$= \int \nabla_\mu P_\mu(\theta) F(\theta) d\theta$$

$$= \int P_\mu(\theta) \frac{\nabla_\mu P_\mu(\theta)}{P_\mu(\theta)} F(\theta) d\theta$$

$$= \int P_\mu(\theta) \nabla_\mu \log P_\mu(\theta) F(\theta) d\theta$$

$$= \mathbb{E}_{\theta \sim P_\mu(\theta)} \left[ \nabla_\mu \log P_\mu(\theta) F(\theta) \right]$$

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

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

\[
\nabla_{\theta} U(\theta) = \nabla_{\theta} \mathbb{E}_{\tau \sim P_{\theta}(\tau)} [R(\tau)]
\]

\[
= \nabla_{\theta} \sum_{\tau} P_{\theta}(\tau) R(\tau)
\]

\[
= \sum_{\tau} \nabla_{\theta} P_{\theta}(\tau) R(\tau)
\]

\[
= \sum_{\tau} P_{\theta}(\tau) \frac{\nabla_{\theta} P_{\theta}(\tau)}{P_{\theta}(\tau)} R(\tau)
\]

\[
= \sum_{\tau} P_{\theta}(\tau) \nabla_{\theta} \log P_{\theta}(\tau) R(\tau)
\]

\[
= \mathbb{E}_{\tau \sim P_{\theta}(\tau)} [\nabla_{\theta} \log P_{\theta}(\tau) R(\tau)]
\]

Sample estimate:

\[
\nabla_{\theta} U(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}(a^{(i)}_t | s^{(i)}_t) R(s^{(i)}_t, a^{(i)}_t)
\]

Considers distribution over policy parameters

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

\[
\nabla_{\mu} U(\mu) = \nabla_{\mu} \mathbb{E}_{\theta \sim P_{\mu}(\theta)} [F(\theta)]
\]

\[
= \nabla_{\mu} \int P_{\mu}(\theta) F(\theta) d\theta
\]

\[
= \int \nabla_{\mu} P_{\mu}(\theta) F(\theta) d\theta
\]

\[
= \int P_{\mu}(\theta) \frac{\nabla_{\mu} P_{\mu}(\theta)}{P_{\mu}(\theta)} F(\theta) d\theta
\]

\[
= \int P_{\mu}(\theta) \nabla_{\mu} \log P_{\mu}(\theta) F(\theta) d\theta
\]

\[
= \mathbb{E}_{\theta \sim P_{\mu}(\theta)} [\nabla_{\mu} \log P_{\mu}(\theta) F(\theta)]
\]

Sample estimate:

\[
\nabla_{\mu} U(\mu) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_{\mu} \log P_{\mu}(\theta^{(i)}) F(\theta^{(i)})
\]
Suppose $\theta \sim P_\mu(\theta)$ is a Gaussian distribution with mean $\mu$, and covariance matrix $\sigma^2 I$.

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

\[
\nabla_\mu \log P_\mu(\theta) = \frac{\theta - \mu}{\sigma^2}
\]
Suppose $\theta \sim P_\mu(\theta)$ is a Gaussian distribution with mean $\mu$, 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 $\theta_1, \theta_2$, and obtain two trajectories $\tau_1, \tau_2$:

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

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

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

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

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

The theta samples have the desired mean and variance
Suppose $\theta \sim P_\mu(\theta)$ is a Gaussian distribution with mean $\mu$, 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 $\mu$, 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 $\theta_1, \theta_2$, and obtain two trajectories $\tau_1, \tau_2$:

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

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

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

Approximately:

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

1: Input: Learning rate $\alpha$, noise standard deviation $\sigma$, initial policy parameters $\theta_0$
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
## Antithetic sampling

- Sample a pair of policies with mirror noise \((\theta_+ = \mu + \sigma \epsilon, \theta_- = \mu - \sigma \epsilon)\)
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 \( (\tau_+, \tau_-) \)
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 \((\tau_+, \tau_-)\)
- SPSA: Finite Difference with random direction

\[
\nabla_\mu \mathbb{E}[R(\tau)] \approx \frac{1}{2} \left[ R(\tau_+) \frac{\theta_+ - \mu}{\sigma^2} + R(\tau_-) \frac{\theta_- - \mu}{\sigma^2} \right] \\
= \frac{1}{2} \left[ R(\tau_+) \frac{\sigma \epsilon}{\sigma^2} + R(\tau_-) \frac{-\sigma \epsilon}{\sigma^2} \right] \\
= \frac{\epsilon}{2\sigma} \left[ R(\tau_+) - R(\tau_-) \right]
\]
### 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 \((\tau_+, \tau_-)\)
  - SPSA: Finite Difference with random direction

\[
\nabla_{\mu} \mathbb{E}[R(\tau)] \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} [R(\tau_+) - R(\tau_-)] \\

\text{Finite Difference} \quad \frac{\partial U}{\partial \theta_j}(\theta) = \frac{U(\theta + \epsilon \epsilon_j) - U(\theta - \epsilon \epsilon_j)}{2\epsilon}
\]
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:

$$e_j = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

← $j^{th}$ entry
Main contribution:
• Parallelization with a need for tiny only cross-worker communication
Distributed SGD

Worker 1
Worker 2
Worker 3
Worker 4
Worker 5
Worker 6

Used in Asynchronous RL!
Distributed SGD

Each worker sends big gradient vectors
Distributed Evolution

Worker 1
Worker 2
Worker 3
Worker 4
Worker 5
Worker 6

What need to be sent??
Algorithm 1 Evolution Strategies

1: **Input:** Learning rate $\alpha$, noise standard deviation $\sigma$, initial policy parameters $\theta_0$
2:  
3:  
4:  
5:  
6:  

Worker 5

Worker 4
Distributed Evolution

Worker 1

Worker 2

Worker 6

Worker 3

\( \theta \) and \( R(\tau) \)?

\( \theta \) 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

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: \quad Sample $\epsilon_i \sim \mathcal{N}(0, I)$
6: \quad 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: \quad Reconstruct all perturbations $\epsilon_j$ for $j = 1, \ldots, n$
11: \quad 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**

[Salimans, Ho, Chen, Sutskever, 2017]
Distributed Evolution

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 $j = 1, \ldots, n$
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

[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

(a) Sequential Optimisation

Hyperparameters → Performance

Training → Weights
Searching for Hyperparameters
Algorithm 1 Population Based Training (PBT)

1: procedure TRAIN($\mathcal{P}$)  \hfill $\triangleright$ initial population $\mathcal{P}$
2:  \hspace{1em} for $(\theta, h, p, t) \in \mathcal{P}$ (asynchronously in parallel) do
3:      \hspace{2em} while not end of training do \hfill $\triangleright$ one step of optimisation using hyperparameters $h$
4:         \hspace{3em} $\theta \leftarrow \text{step}(\theta|h)$ \hfill $\triangleright$ current model evaluation
5:         \hspace{3em} $p \leftarrow \text{eval}(\theta)$
6:         \hspace{3em} if ready($p, t, \mathcal{P}$) then \hfill $\triangleright$ use the rest of population to find better solution
7:             \hspace{4em} $h', \theta' \leftarrow \text{exploit}(h, \theta, p, \mathcal{P})$
8:             \hspace{4em} if $\theta \neq \theta'$ then \hfill $\triangleright$ produce new hyperparameters $h$
9:                 \hspace{5em} $h, \theta \leftarrow \text{explore}(h', \theta', \mathcal{P})$ \hfill $\triangleright$ new model evaluation
10:                \hspace{5em} $p \leftarrow \text{eval}(\theta)$
11:         \hspace{3em} end if
12:      \hspace{2em} end if
13:  \hspace{1em} update $\mathcal{P}$ with new $(\theta, h, p, t + 1)$ \hfill $\triangleright$ update population
14:  \hspace{1em} end while
15: \hspace{1em} end for
16: \hspace{1em} return $\theta$ with the highest $p$ in $\mathcal{P}$
17: end procedure
Searching for Hyperparameters

(a) Sequential Optimisation

Hyperparameters → Performance → Weights → Training

(b) Parallel Random/Grid Search

(c) Population Based Training

Hyperparameters → Performance → Weights

exploit

explore