Scaling Up RL Using Evolution Strategies

Tim Salimans, Jonathan Ho, Peter Chen, Szymon Sidor, Ilya Sutskever
Reinforcement Learning = AI?

- Definition of “RL” broad enough to capture all that is needed for AGI
- Increased interest and improved algorithms
- Large investments are made
Still a long way to go...
What’s keeping us?

- Credit assignment
- Compute

- Many other things we will not discuss right now
Credit assignment is difficult for general MDPs
Credit assignment is difficult for general MDPs

- At state $s_t$ take action $a_t$. Next get state $s_{t+1}$

- Receive return $R$ after taking $T$ actions

- No precisely timed rewards, no discounting, no value functions

- Currently this seems true for our hardest problems, like meta learning

Wang et al. (2016) "Learning to reinforcement learn."
Vanilla policy gradients

• Stochastic policy $P(a \mid s, \theta)$

• Estimate gradient of expected return $F = E[R]$ using REINFORCE

$$\nabla_{\theta} F_{PG}(\theta) = \mathbb{E}_a \{ R(a) \nabla_{\theta} \log p(a; \theta) \}$$
Vanilla policy gradients

- Correlation between return and individual actions is typically low

\[
\text{Var}[\nabla_{\theta} F_{PG}(\theta)] \approx \text{Var}[R(a)] \text{Var}[\nabla_{\theta} \log p(a; \theta)]
\]

- Gradient of logprob is sum of T uncorrelated terms

\[
\nabla_{\theta} \log p(a; \theta) = \sum_{t=1}^{T} \nabla_{\theta} \log p(a_t; \theta)
\]

- This means the variance grows linearly with T!
We can do only very little sequential computation
CPU clock speed has stopped improving long ago

But increased parallelism keeps us going.

Supercomputer GFLOPS over time. Source: WikiPedia
Communication is the eventual bottleneck

- Clock speed $= \text{constant}$
- Number of cores $\rightarrow \infty$

$\rightarrow$ communication bandwidth between cores becomes bottleneck
Thought experiment:
What’s the optimal algorithm to calculate a policy gradient if…

• Sequence length $T \rightarrow \infty$

• We cannot do credit assignment

• Communication is the only computational bottleneck
Thought experiment:
What’s the optimal algorithm to calculate a policy gradient if...

- Sequence length $T \rightarrow \infty$
- We cannot do credit assignment
- Communication is the only computational bottleneck

Finite differences!
Finite differences and other black box optimizers

\[ \nabla_{\theta_i} F_{PG}(\theta) \approx \frac{E_{a \sim p(a; \theta_i + \epsilon)}[R(a)] - E_{a \sim p(a; \theta_i - \epsilon)}[R(a)]}{2\epsilon} \]

- Each function evaluation only requires communicating a scalar result
- Variance independent of sequence length
- No credit assignment required
Evolution Strategies

- Old technique, known under many other names
- Randomized finite differences:
  - Add noise vector \( \epsilon \) to the parameters
  - If the result improves, keep the change
  - Repeat
Parallelization

- You have a bunch of workers
- They all try on different random noise
- Then they report how good the random noise was
- But they *don’t need to communicate the noise vector*
- Because they know each other’s seeds!
Parallelization

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: **end for**
9: Send all scalar returns $F_i$ from each worker to every other worker
10: **for** each worker $i = 1, \ldots, n$ **do**
11:    Reconstruct all perturbations $\epsilon_j$ for $j = 1, \ldots, n$ using known random seeds
12:    Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n \sigma} \sum_{j=1}^{n} F_j \epsilon_j$
13: **end for**
Distributed Deep Learning

Worker 1
Worker 2
Worker 3
Worker 4
Worker 5
Worker 6
Distributed Deep Learning

Each worker sends big vectors

Worker 1 — Worker 2 — Worker 3 — Worker 4 — Worker 5 — Worker 6
Distributed Evolution Strategies

Each worker broadcasts tiny scalars
Distributed Evolution Strategies

Each worker broadcasts tiny scalars
Distributed Evolution Strategies

Each worker broadcasts tiny scalars
Does it work in practice?

- Surprisingly competitive with popular RL techniques in terms of data efficiency
  - need 3-10x more data than TRPO / A3C on MuJoCo and Atari
- No backward pass, no need to store activations in memory
- Near perfect scaling
MuJoCo results

• ES needs more data, but it achieves nearly the same result

• If we use 1440 cores, we need **10 minutes** to solve the humanoid task, which takes 1 day with TRPO on a single machine
Distributed Evolution Strategies

- Quantitative results on the Humanoid MuJoCo task:
Distributed Evolution Strategies

• Networking requirements very limited

• Cheap! $12 to rent 1440 cores for an hour on Amazon EC2 with spot pricing

• Can run the experiment 6 times for $12!
MuJoCo Results

• Humanoid walker
Atari Results

- We can match one-day A3C on Atari games on average (better on 50%, worse on 50% of games) in 1 hour of our distributed implementation with 720 cores.
Long Horizons

- Long horizons are hard for RL
- RL is sensitive to action frequency
- Higher frequency of actions makes the RL problem more difficult
- Not so for Evolution Strategies
Long Horizons
How can it work in high dimensions?

- Fact: the speed of Evolution Strategies depends on the intrinsic dimensionality of the problem, not on the actual dimensionality of the neural net policy
Evolution strategies automatically discards the irrelevant dimensions — even when they live on a complicated subspace!
Intrinsic Dimensionality

• One explanation for how hill-climbing can succeed in a million-dimensional space!

• Parameterization of policy matters more than number of parameters

  • Virtual batch normalization helps a lot
    Salimans et al. (2016) "Improved techniques for training GANs."

• Future advances to be made?
• Evolution strategies does not use backprop

• So scale of initialization, vanishing gradients, etc, are not important?
Backprop vs Evolution Strategies

• Counterintuitive result: *every* trick that helps backprop, also helps evolution strategies

• scale of random init, batch norm, ResNet…

• Why? Because evolution strategies tries to estimate *the gradient*!

• If the gradient is vanishing, we won’t get much by estimating it!
Conclusion: pros

- Though experiment: black box methods optimal if long horizon, no credit assignment, bandwidth limited
- Scales extremely well
- Competitive with other RL techniques
- Possibility proof for evolution of intelligence: us
Conclusion: cons

- Natural evolution seems much more sophisticated
  - Better parameterization?
  - Evolution of evolvability?
- Assumption that we cannot solve credit assignment / communication may be pessimistic
  - We should not give up on improvements in credit assignment, value functions, hierarchical RL, networking, and communication strategies!