EXHIBIT 5

PageVault

Document title:	Evolution strategies as a scalable alternative to reinforcement learning | OpenAI
Capture URL:	https://openai.com/index/evolution-strategies/
Page loaded at (UTC):	Thu, 07 Nov 2024 20:45:41 GMT
Capture timestamp (UTC):	Thu, 07 Nov 2024 20:46:13 GMT
Capture tool:	10.52.0
Collection server IP:	54.145.42.72
Browser engine:	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.6478.234 Safari/537.36
Operating system:	Linux (Node 20.17.0)
PDF length:	11
Capture ID:	idayJsRS419ugc2FPzWcTF
Display Name:	leslie

PDF REFERENCE #: gyLh7dmY2icCxR8Nd5EwzW

March 24, 2017

Evolution strategies as a scalable alternative to reinforcement learning

Read paper ↗ Read documentation ↗



Illustration: Ben Barry

We've discovered that evolution strategies (ES), an optimization technique that's been known for decades, rivals the performance of standard reinforcement learning (RL) techniques on modern RL benchmarks (e.g. Atari/MuJoCo), while overcoming many of RL's inconveniences.

In particular, ES is simpler to implement (there is no need for backpropagation), it is easier to scale in a distributed setting, it does not suffer in settings with sparse rewards, and has fewer hyperparameters. This outcome is surprising because ES resembles simple hill-climbing in a high-dimensional space based only on finite differences along a few random directions at each step.

Read aloud

In particular, ES is simpler to implement (there is no need for backpropagation), it is easier to scale in a distributed setting, it does not suffer in settings with sparse rewards, and has fewer hyperparameters. This outcome is surprising because ES resembles simple hill-climbing in a high-dimensional space based only on finite differences along a few random directions at each step.

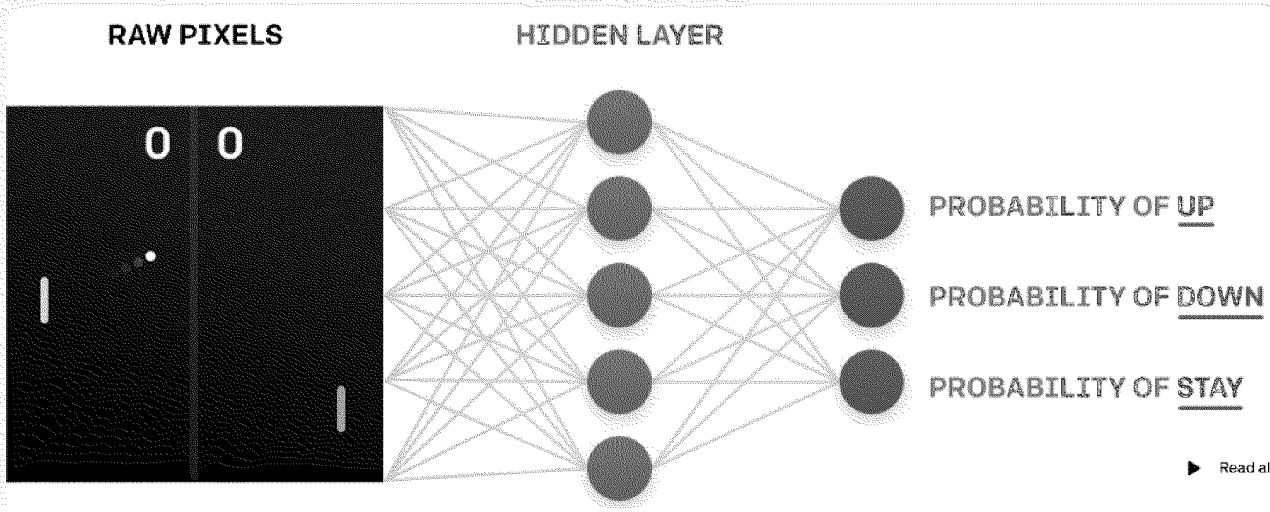
Our finding continues the modern trend of achieving strong results with decades-old ideas. For example, in 2012, the "AlexNet" paper showed how to design, scale and train convolutional neural networks (CNNs) to achieve extremely strong results on image recognition tasks, at a time when most researchers thought that CNNs were not a promising approach to computer vision. Similarly, in 2013, the Deep Q-Learning paper showed how to combine Q-Learning with CNNs to successfully solve Atari games, reinvigorating RL as a research field with exciting experimental (rather than theoretical) results. Likewise, our work demonstrates that ES achieves strong performance on RL benchmarks, dispelling the common belief that ES methods are impossible to apply to high dimensional problems.

ES is easy to implement and scale. Running on a computing cluster of 80 machines and 1,440 CPU cores, our implementation is able to train a 3D MuJoCo humanoid walker in only 10 minutes (A3C on 32 cores takes about 10 hours). Using 720 cores we can also obtain comparable performance to A3C on Atari while cutting down the training time from 1 day to 1 hour.

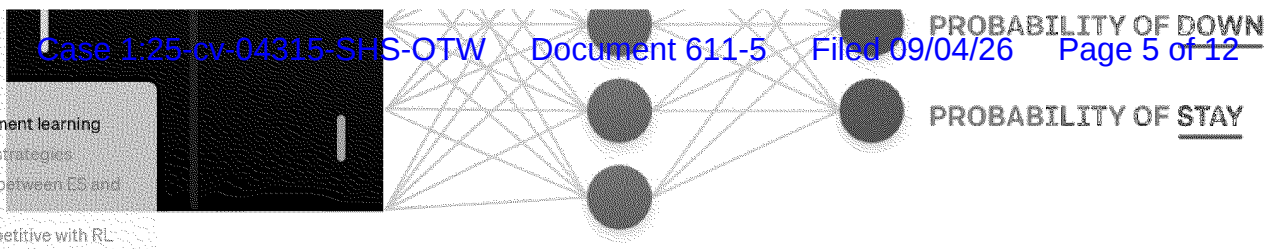
In what follows, we'll first briefly describe the conventional RL approach, contrast that with our ES approach, discuss the tradeoffs between ES and RL, and finally highlight some of our experiments.

Reinforcement learning

Let's briefly look at how RL works. Suppose we are given some environment (e.g. a game) that we'd like to train an agent on. To describe the behavior of the agent, we define a policy function (the brain of the agent), which computes how the agent should act in any given situation. In practice, the policy is usually a neural network that takes the current state of the game as an input and calculates the probability of taking any of the allowed actions. A typical policy function might have about 1,000,000 parameters, so our task comes down to finding the precise setting of these parameters such that the policy plays well (i.e. wins a lot of games).

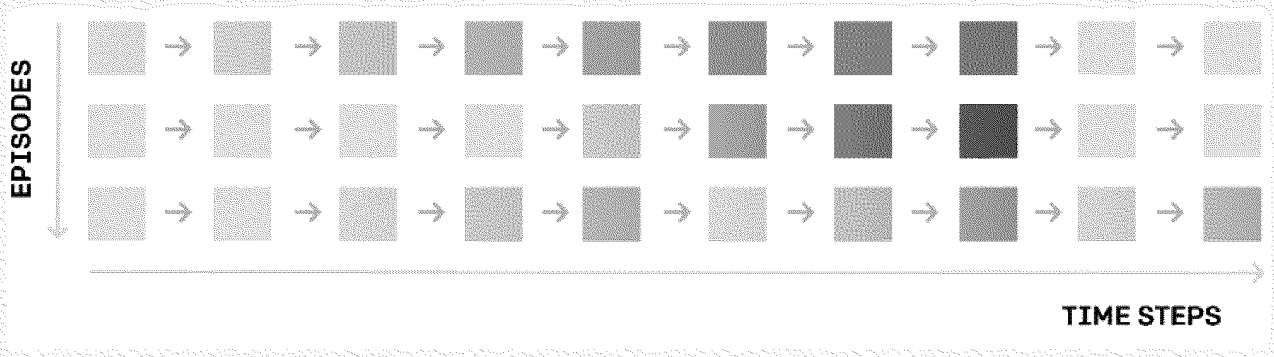


Reinforcement learning
Evolution strategies
Tradeoffs between ES and RL
ES is competitive with RL
Related work
Conclusion



Above: In the game of Pong, the policy could take the pixels of the screen and compute the probability of moving the player's paddle (in green, on right) Up, Down, or neither.

The training process for the policy works as follows. Starting from a random initialization, we let the agent interact with the environment for a while and collect episodes of interaction (e.g. each episode is one game of Pong). We thus obtain a complete recording of what happened: what sequence of states we encountered, what actions we took in each state, and what the reward was at each step. As an example, below is a diagram of three episodes that each took 10 time steps in a hypothetical environment. Each rectangle is a state, and rectangles are colored green if the reward was positive (e.g. we just got the ball past our opponent) and red if the reward was negative (e.g. we missed the ball):



This diagram suggests a recipe for how we can improve the policy; whatever we happened to do leading up to the green states was good, and whatever we happened to do in the states leading up to the red areas was bad. We can then use backpropagation to compute a small update on the network's parameters that would make the green actions more likely in those states in the future, and the red actions less likely in those states in the future. We expect that the updated policy works a bit better as a result. We then iterate the process: collect another batch of episodes, do another update, etc.

Exploration by injecting noise in the actions. The policies we usually use in RL are stochastic, in that they only compute probabilities of taking any action. This way, during the course of training, the agent may find itself in a particular state many times, and at different times it will take different actions due to the sampling. This provides the signal needed for learning; some of those actions will lead to good outcomes, and get encouraged, and some of them will not work out, and get discouraged. We therefore say that we introduce exploration into the learning process by injecting noise into the agent's actions, which we do by sampling from the action distribution at each time step. This will be in contrast to ES, which we describe next.

Evolution strategies

On "Evolution". Before we dive into the ES approach, it is important to note that despite the word "evolution", ES has very little to do with biological evolution. Early versions of these

Read aloud

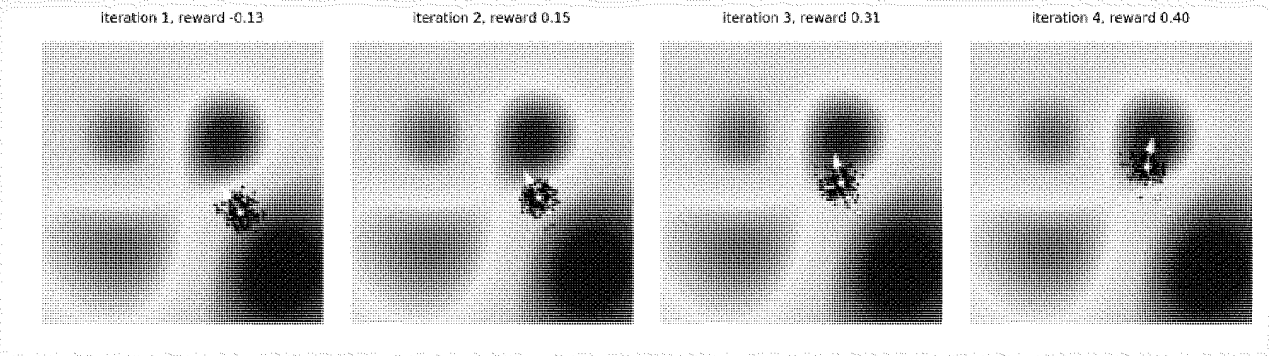
Reinforcement learning
Evolution strategies
Tradeoffs between ES and RL
ES is competitive with RL
Related work
Conclusion

Evolution strategies

On "Evolution". Before we dive into the ES approach, it is important to note that despite the word "evolution", ES has very little to do with biological evolution. Early versions of these techniques may have been inspired by biological evolution and the approach can, on an abstract level, be seen as sampling a population of individuals and allowing the successful individuals to dictate the distribution of future generations. However, the mathematical details are so heavily abstracted away from biological evolution that it is best to think of ES as simply a class of black-box stochastic optimization techniques.

Black-box optimization. In ES, we forget entirely that there is an agent, an environment, that there are neural networks involved, or that interactions take place over time, etc. The whole setup is that 1,000,000 numbers (which happen to describe the parameters of the policy network) go in, 1 number comes out (the total reward), and we want to find the best setting of the 1,000,000 numbers. Mathematically, we would say that we are optimizing a function f(w) with respect to the input vector w (the parameters / weights of the network), but we make no assumptions about the structure of f, except that we can evaluate it (hence "black box").

The ES algorithm. Intuitively, the optimization is a "guess and check" process, where we start with some random parameters and then repeatedly 1) tweak the guess a bit randomly, and 2) move our guess slightly towards whatever tweaks worked better. Concretely, at each step we take a parameter vector w and generate a population of, say, 100 slightly different parameter vectors w1 ... w100 by jittering w with gaussian noise. We then evaluate each one of the 100 candidates independently by running the corresponding policy network in the environment for a while, and add up all the rewards in each case. The updated parameter vector then becomes the weighted sum of the 100 vectors, where each weight is proportional to the total reward (i.e. we want the more successful candidates to have a higher weight). Mathematically, you'll notice that this is also equivalent to estimating the gradient of the expected reward in the parameter space using finite differences, except we only do it along 100 random directions. Yet another way to see it is that we're still doing RL (Policy Gradients, or REINFORCE specifically), where the agent's actions are to emit entire parameter vectors using a gaussian policy.



Above: ES optimization process, in a setting with only two parameters and a reward function (red = high, blue = low). At each iteration we show the current parameter value (in white), a population of jittered samples (in...

Code sample. To make the core algorithm concrete and to highlight its simplicity, here is a short example of optimizing a quadratic function using ES (or see this longer version with more comments):

Read aloud

Reinforcement learning
Evolution strategies
Tradeoffs between ES and RL
ES is competitive with RL
Related work
Conclusion

Code sample. To make the core algorithm concrete and to highlight its simplicity, here is a short example of optimizing a quadratic function using ES (or see this longer version with more comments):

Python

```
import numpy as np
solution = np.array([0.5, 0.1, -0.3])
def f(w): return -np.sum((w - solution)**2)

npop = 50      # population size
sigma = 0.1    # noise standard deviation
alpha = 0.001  # learning rate
w = np.random.randn(3) # initial guess
for i in range(300):
  N = np.random.randn(npop, 3)
  R = np.zeros(npop)
  for j in range(npop):
    w_try = w + sigma*N[j]
    R[j] = f(w_try)
  A = (R - np.mean(R)) / np.std(R)
  w = w + alpha/(npop*sigma) * np.dot(N.T, A)
```

Simple example: minimize a quadratic around some solution point

Injecting noise in the parameters. Notice that the objective is identical to the one that RL optimizes: the expected reward. However, RL injects noise in the action space and uses backpropagation to compute the parameter updates, while ES injects noise directly in the parameter space. Another way to describe this is that RL is a "guess and check" on actions, while ES is a "guess and check" on parameters. Since we're injecting noise in the parameters, it is possible to use deterministic policies (and we do, in our experiments). It is also possible to add noise in both actions and parameters to potentially combine the two approaches.

Tradeoffs between ES and RL

ES enjoys multiple advantages over RL algorithms (some of them are a little technical):

- **No need for backpropagation**. ES only requires the forward pass of the policy and does not require backpropagation (or value function estimation), which makes the code shorter and between 2-3 times faster in practice. On memory-constrained systems, it is also not necessary to keep a record of the episodes for a later update. There is also no need to worry about exploding gradients in RNNs. Lastly, we can explore a much larger function class of policies, including networks that are not differentiable (such as in binary networks), or ones that include complex modules (e.g. pathfinding, or various optimization layers).
- **Highly parallelizable**. ES only requires workers to communicate a few scalars between each other, while in RL it is necessary to synchronize entire parameter vectors (which can be millions of numbers). Intuitively, this is because we control the random seeds on each worker, so each worker can locally reconstruct the perturbations of the other workers. Thus, all that we need to communicate between workers is the reward of each perturbation. As a result, we observed linear speedups in our experiments as we added on the order of thousands of CPU cores to the optimization.
- **Higher robustness**. Several hyperparameters that are difficult to set in RL implementations are side-stepped in ES. For example, RL is not "scale-free", so one can achieve very different learning outcomes (including a complete failure) with different settings of the frame-skip hyperparameter in Atari. As we show in our work, ES works about equally well with any frame-skip.

Read aloud

perturbation. As a result, we observed linear speedups in our experiments as we added more cores, all the way to thousands of CPU cores in our optimization.

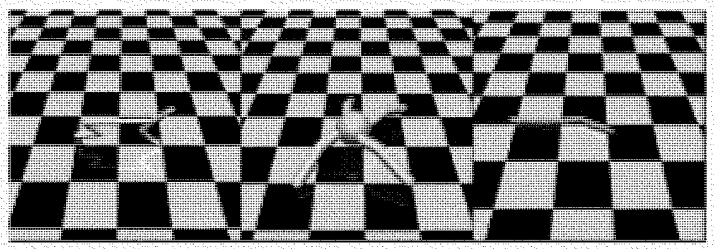
Reinforcement learning
Evolution strategies
Tradeoffs between ES and RL
ES is competitive with RL
Related work
Conclusion

- **Higher robustness.** Several hyperparameters that are difficult to set in RL implementations are side-stepped in ES. For example, RL is not "scale-free", so one can achieve very different learning outcomes (including a complete failure) with different settings of the frame-skip hyperparameter in Atari. As we show in our work, ES works about equally well with any frame-skip.
- **Structured exploration.** Some RL algorithms (especially policy gradients) initialize with random policies, which often manifests as random jitter on spot for a long time. This effect is mitigated in Q-Learning due to epsilon-greedy policies, where the max operation can cause the agents to perform some consistent action for a while (e.g. holding down a left arrow). This is more likely to do something in a game than if the agent jitters on spot, as is the case with policy gradients. Similar to Q-learning, ES does not suffer from these problems because we can use deterministic policies and achieve consistent exploration.
- **Credit assignment over long time scales.** By studying both ES and RL gradient estimators mathematically we can see that ES is an attractive choice especially when the number of time steps in an episode is long, where actions have longlasting effects, or if no good value function estimates are available.

Conversely, we also found some challenges to applying ES in practice. One core problem is that in order for ES to work, adding noise in parameters must lead to different outcomes to obtain some gradient signal. As we elaborate on in our paper, we found that the use of virtual batchnorm can help alleviate this problem, but further work on effectively parameterizing neural networks to have variable behaviors as a function of noise is necessary. As an example of a related difficulty, we found that in Montezuma's Revenge, one is very unlikely to get the key in the first level with a random network, while this is occasionally possible with random actions.

ES is competitive with RL

We compared the performance of ES and RL on two standard RL benchmarks: MuJoCo control tasks and Atari game playing. Each MuJoCo task (see examples below) contains a physically-simulated articulated figure, where the policy receives the positions of all joints and has to output the torques to apply at each joint in order to move forward. Below are some example agents trained on three MuJoCo control tasks, where the objective is to move forward:

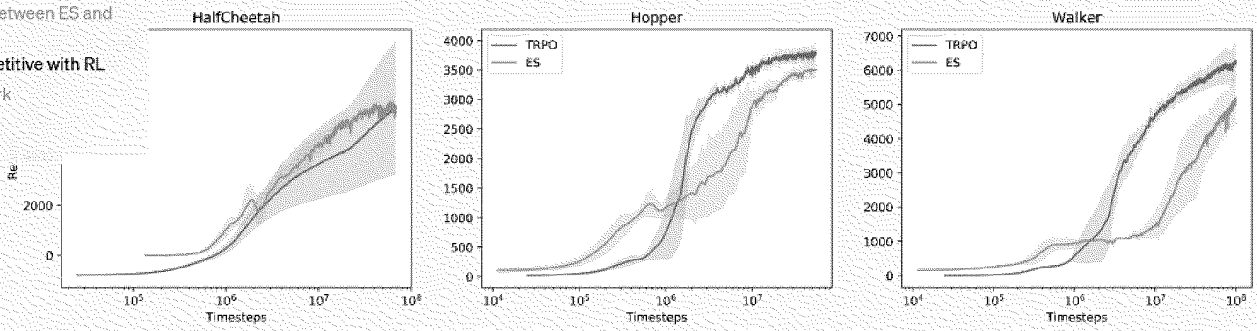


We usually compare the performance of algorithms by looking at their efficiency of learning from data; as a function of how many states we've seen, what is our average reward? Here are the example learning curves that we obtain, in comparison to RL (the TRPO algorithm in this case):



from data; as a function of how many states we've seen, what is our average reward? Here are the example learning curves that we obtain in comparison to RL (the TRPO algorithm in this case):

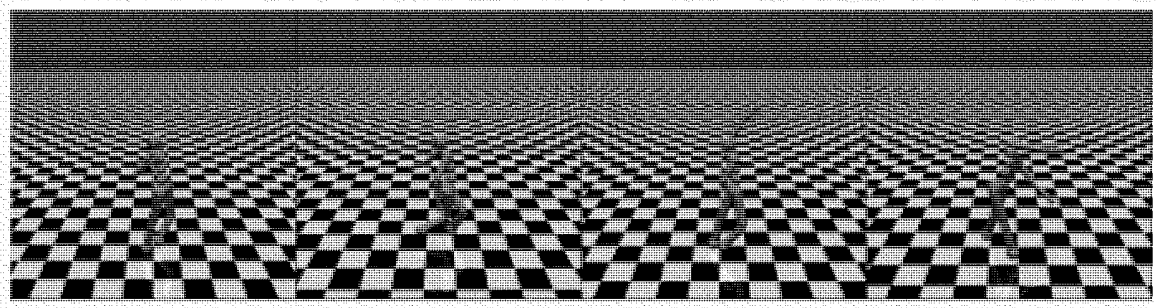
Reinforcement learning
Evolution strategies
Tradeoffs between ES and RL
ES is competitive with RL
Related work
Conclusion



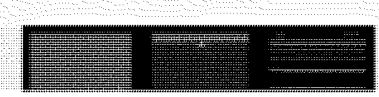
Data efficiency comparison. The comparisons above show that ES (orange) can reach a comparable performance to TRPO (blue), although it doesn't quite match or surpass it in all cases. Moreover, by scanning horizontally we can see that ES is less efficient, but no worse than about a factor of 10 (note the x-axis is in log scale).

Wall clock comparison. Instead of looking at the raw number of states seen, one can argue that the most important metric to look at is the wall clock time: how long (in number of seconds) does it take to solve a given problem? This quantity ultimately dictates the achievable speed of iteration for a researcher. Since ES requires negligible communication between workers, we were able to solve one of the hardest MuJoCo tasks (a 3D humanoid) using 1,440 CPUs across 80 machines in only 10 minutes. As a comparison, in a typical setting 32 A3C workers on one machine would solve this task in about 10 hours. It is also possible that the performance of RL could also improve with more algorithmic and engineering effort, but we found that naively scaling A3C in a standard cloud CPU setting is challenging due to high communication bandwidth requirements.

Below are a few videos of 3D humanoid walkers trained with ES. As we can see, the results have quite a bit of variety, based on which local minimum the optimization ends up converging into.



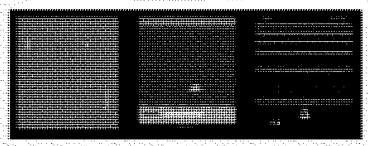
On Atari, ES trained on 720 cores in 1 hour achieves comparable performance to A3C trained on 32 cores in 1 day. Below are some result snippets on Pong, Seaquest and Beamrider. These videos show the preprocessed frames, which is exactly what the agent sees when it is playing:



Read aloud

trained on 32 cores in 1 day. Below are some result snippets on Pong, Seaquest and Beamrider. These videos show the preprocessed frames, which is exactly what the agent sees when it is playing:

Reinforcement learning
Evolution strategies
Tradeoffs between ES and RL
ES is competitive with RL
Related work
Conclusion



In particular, note that the submarine in Seaquest correctly learns to go up when its oxygen reaches low levels.

Related work

ES is an algorithm from the neuroevolution literature, which has a long history in AI and a complete literature review is beyond the scope of this post. However, we encourage an interested reader to look at Wikipedia, Scholarpedia, and Jürgen Schmidhuber's review article (Section 6.6). The work that most closely informed our approach is Natural Evolution Strategies by Wierstra et al. 2014. Compared to this work and much of the work it has inspired, our focus is specifically on scaling these algorithms to large-scale, distributed settings, finding components that make the algorithms work better with deep neural networks (e.g. virtual batch norm), and evaluating them on modern RL benchmarks.

It is also worth noting that neuroevolution-related approaches have seen some recent resurgence in the machine learning literature, for example with HyperNetworks, "Large-Scale Evolution of Image Classifiers" and "Convolution by Evolution".

Conclusion

Our work suggests that neuroevolution approaches can be competitive with reinforcement learning methods on modern agent-environment benchmarks, while offering significant benefits related to code complexity and ease of scaling to large-scale distributed settings. We also expect that more exciting work can be done by revisiting other ideas from this line of work, such as indirect encoding methods, or evolving the network structure in addition to the parameters.

Note on supervised learning. It is also important to note that supervised learning problems (e.g. image classification, speech recognition, or most other tasks in the industry), where one can compute the exact gradient of the loss function with backpropagation, are not directly impacted by these findings. For example, in our preliminary experiments we found that using ES to estimate the gradient on the MNIST digit recognition task can be as much as 1,000 times slower than using backpropagation. It is only in RL settings, where one has to estimate the gradient of the expected reward by sampling, where ES becomes competitive.

Code release. Finally, if you'd like to try running ES yourself, we encourage you to dive into the full details by reading our paper or looking at our code on this GitHub repo.

Read aloud

the full details by reading our paper or looking at our code on this GitHub repo.

Reinforcement learning
Evolution strategies
Tradeoffs between ES and RL
ES is competitive with RL
Related work
Conclusion

Milestone | Publication | Release | Reinforcement Learning

Authors

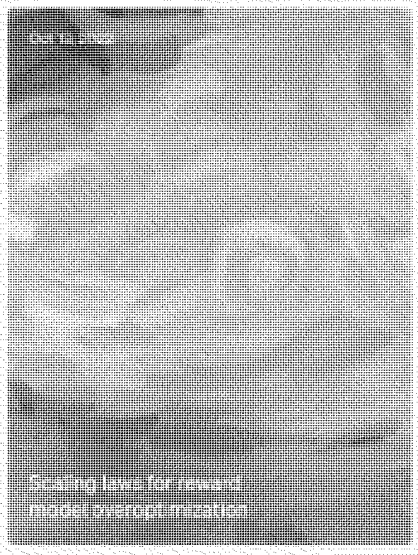
Andrej Karpathy, Tim Salimans, Jonathan Ho, Peter Chen, Ilya Sutskever, Greg Brockman, Szymon Sidor

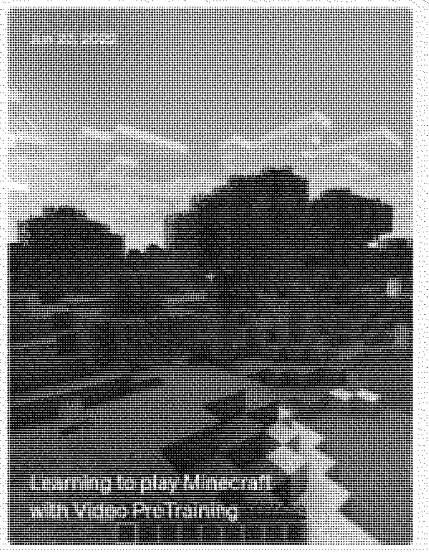
Cover Artwork

Ben Barry

Related articles

View all research articles ›

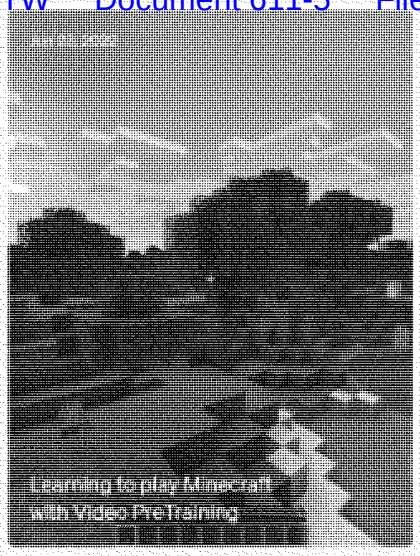






Read aloud







Read aloud

Our research
Overview
Index

Latest advancements
OpenAI o1
OpenAI o1-mini
GPT-4
GPT-4o mini
DALL·E 3
Sora

ChatGPT
For Everyone
For Teams
For Enterprises
ChatGPT login ↗
Download

API
Platform overview
Pricing
Documentation ↗
API login ↗

Explore more
OpenAI for business
Stories

Safety overview
Safety overview

Company
About us
News
Our Charter
Security
Residency
Careers

Terms & policies
Terms of use
Privacy policy
Brand guidelines
Other policies

English (US)

OpenAI © 2015–2024