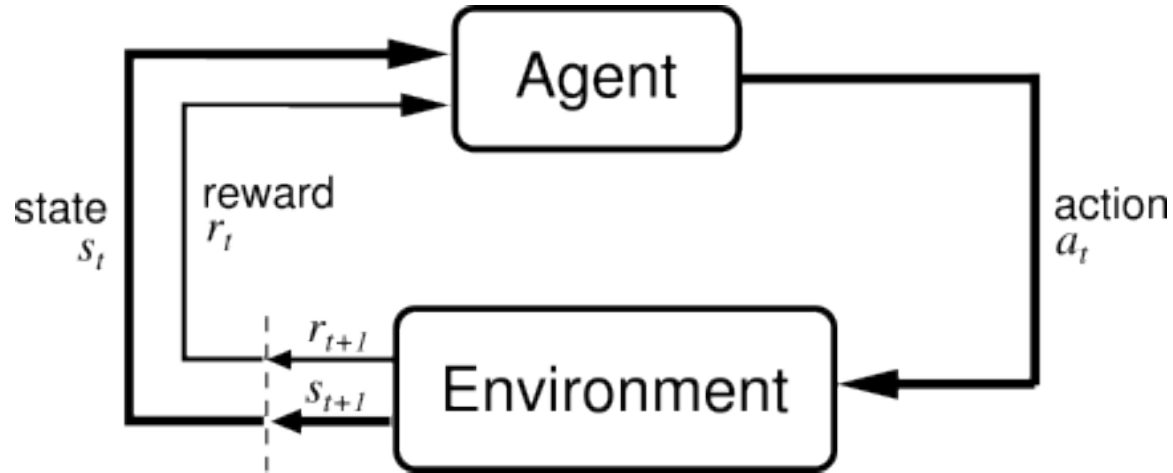


# **Derivative-free and Evolutionary Methods**

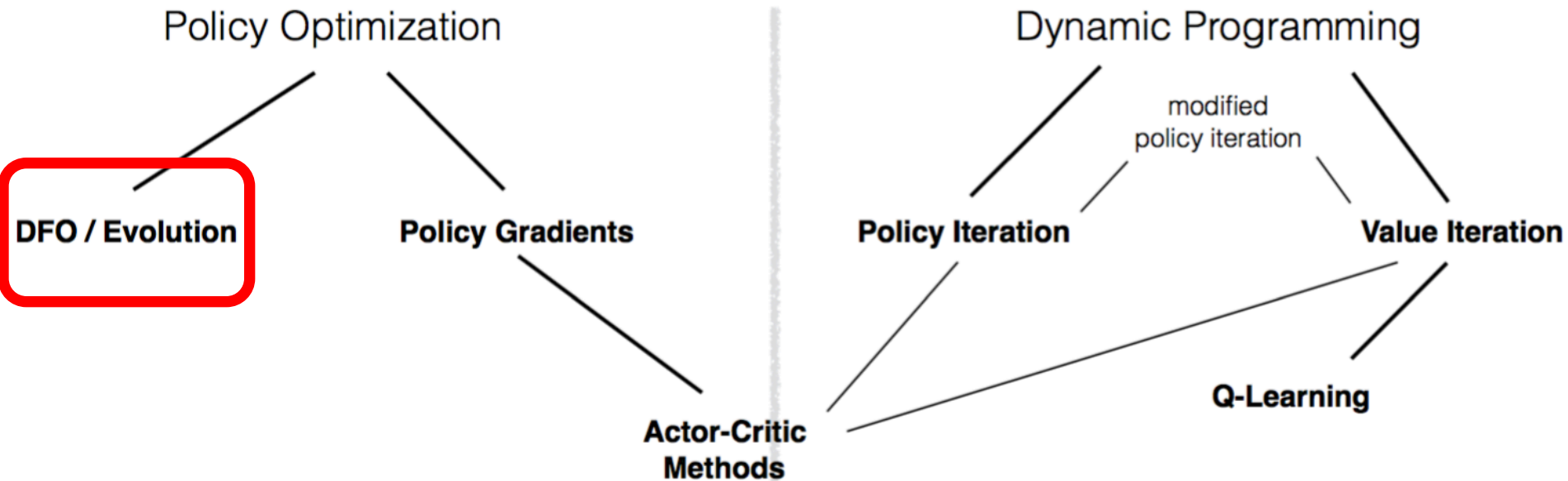
Xi (Peter) Chen – UC Berkeley

Slides authored with John Schulman (OpenAI) and Pieter Abbeel (OpenAI / UC Berkeley)

# Reinforcement Learning



# Policy Optimization in the RL Landscape



$$\max_{\theta} U(\theta) = \max_{\theta} \mathbb{E}\left[\sum_{t=0}^H R(s_t) | \pi_{\theta}\right]$$

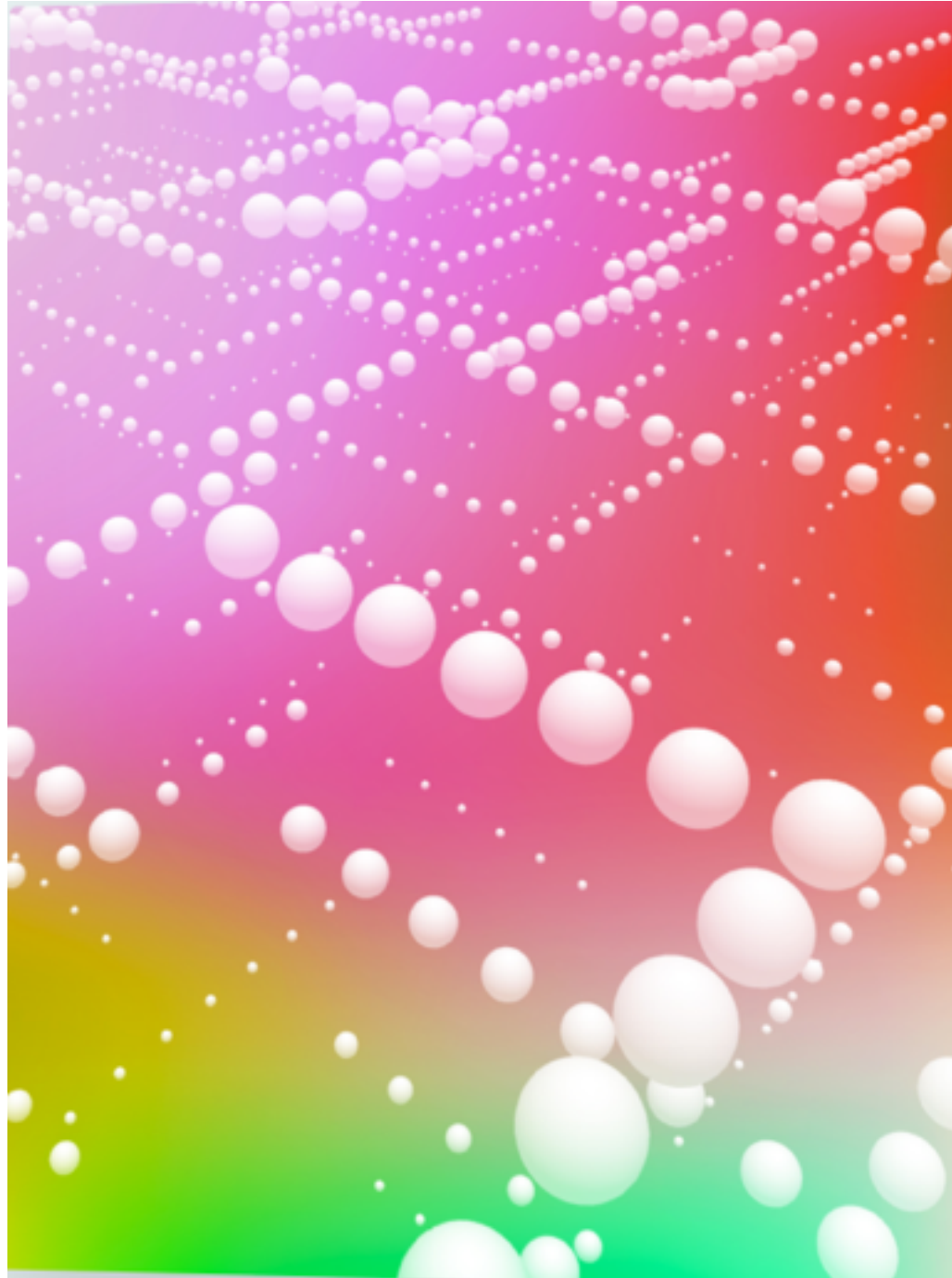
# Evolution Strategies as a Scalable Alternative to Reinforcement Learning

OpenAI

# Notations

- ES: Evolution Strategies
- RL: Reinforcement Learning
- PG: Policy Gradient
- BP: Backpropagation
- NN: Neural Network

# A Strong Alternative



MARCH 24, 2017

## Evolution Strategies as a Scalable Alternative to Reinforcement Learning

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.

# History Repeats Itself

## AlexNet

---

### ImageNet Classification with Deep Convolutional Neural Networks

---

Alex Krizhevsky      Ilya Sutskever      Geoffrey E. Hinton  
University of Toronto      University of Toronto      University of Toronto  
kriz@cs.utoronto.ca      ilya@cs.utoronto.ca      hinton@cs.utoronto.ca

#### Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called “dropout” that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

## DQN

---

### Playing Atari with Deep Reinforcement Learning

---

Volodymyr Mnih    Koray Kavukcuoglu    David Silver    Alex Graves    Ioannis Antonoglou  
Daan Wierstra    Martin Riedmiller

DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

#### Abstract

We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

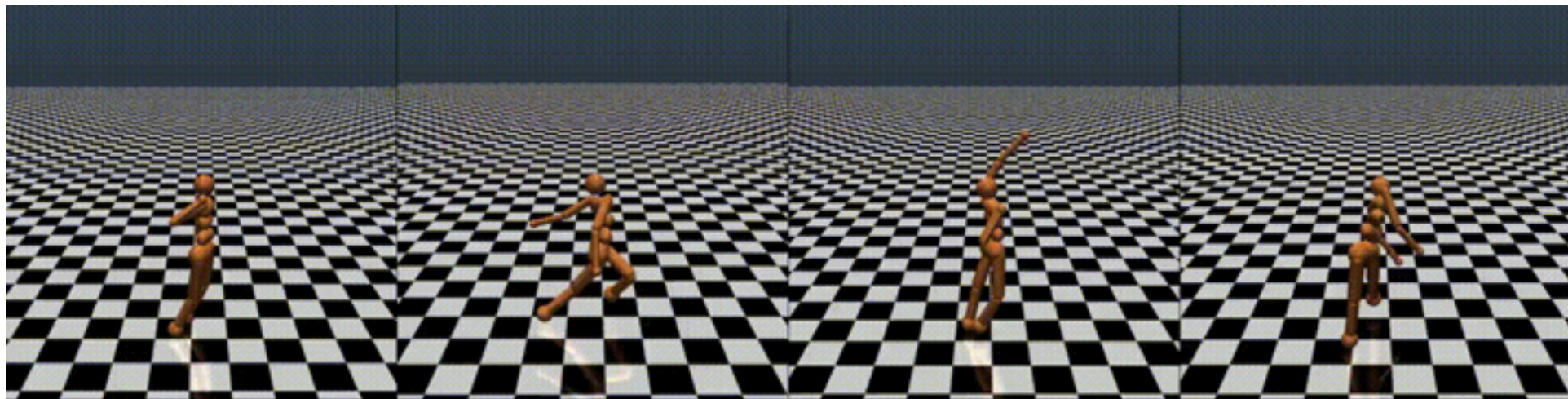
NN, RL are amazing when  
**computing power** is strong.



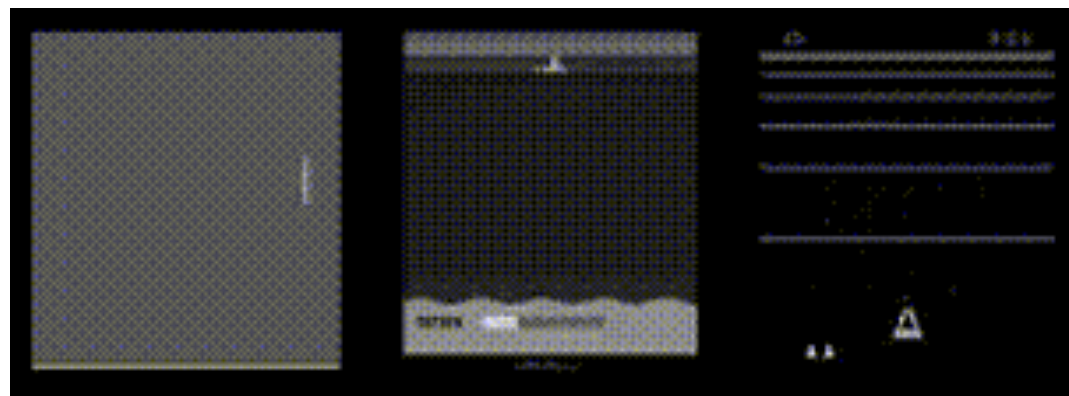
What about ES?

# AMAZING Results of ES

## 3D MuJoCo humanoid walker



## 2D Atari games



# AMAZING Results of ES

## 3D MuJoCo humanoid walker

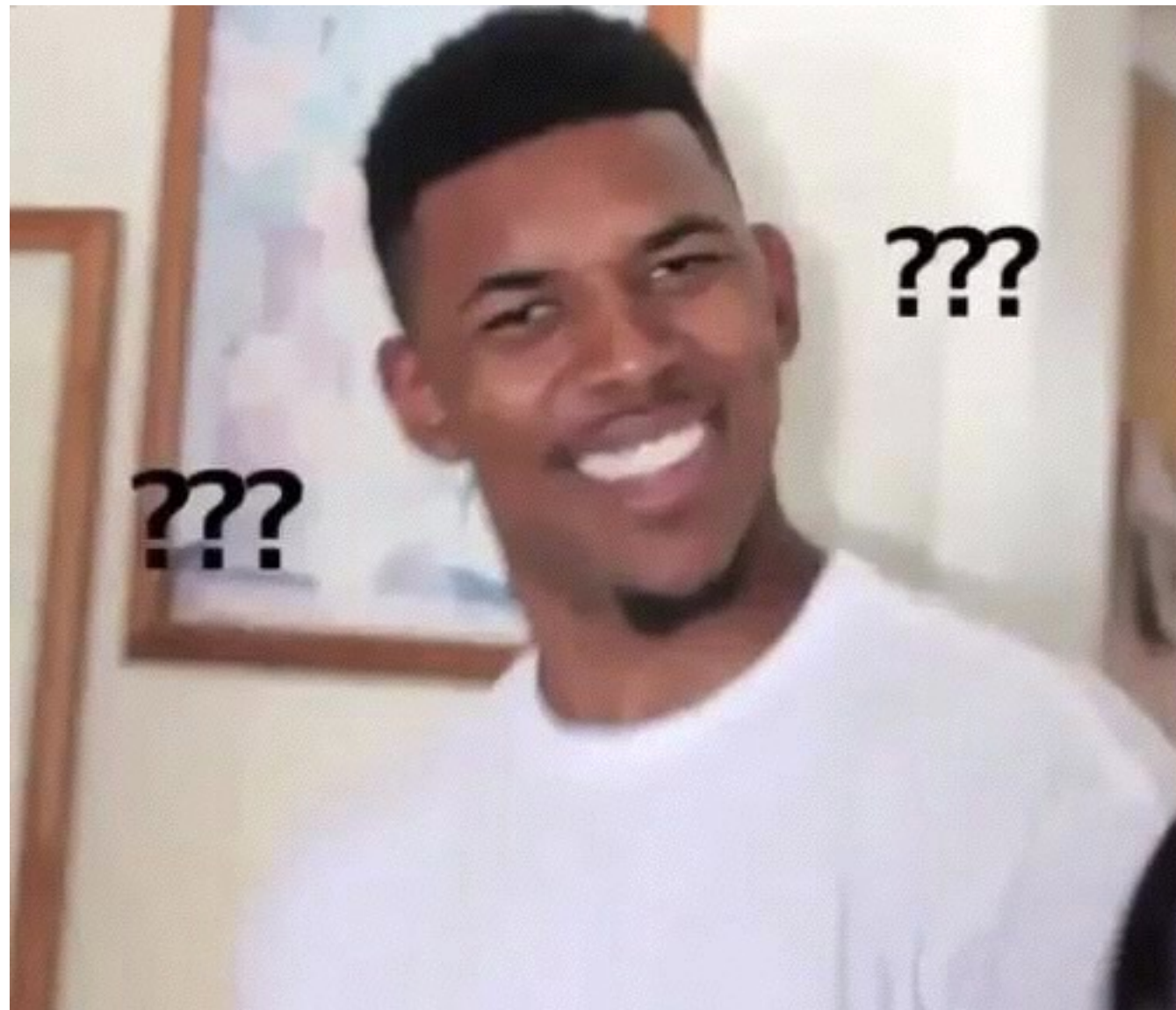
- ES(1440 cores): 10 minutes
- A3C(32 cores): 10 hours

## 2D Atari games

- ES(720 cores): 1 hour
- A3C(32 cores): 1 day

Yet another example of  
**achieving strong results with  
decades-old ideas.**

# Why?




# How ES Works?

---

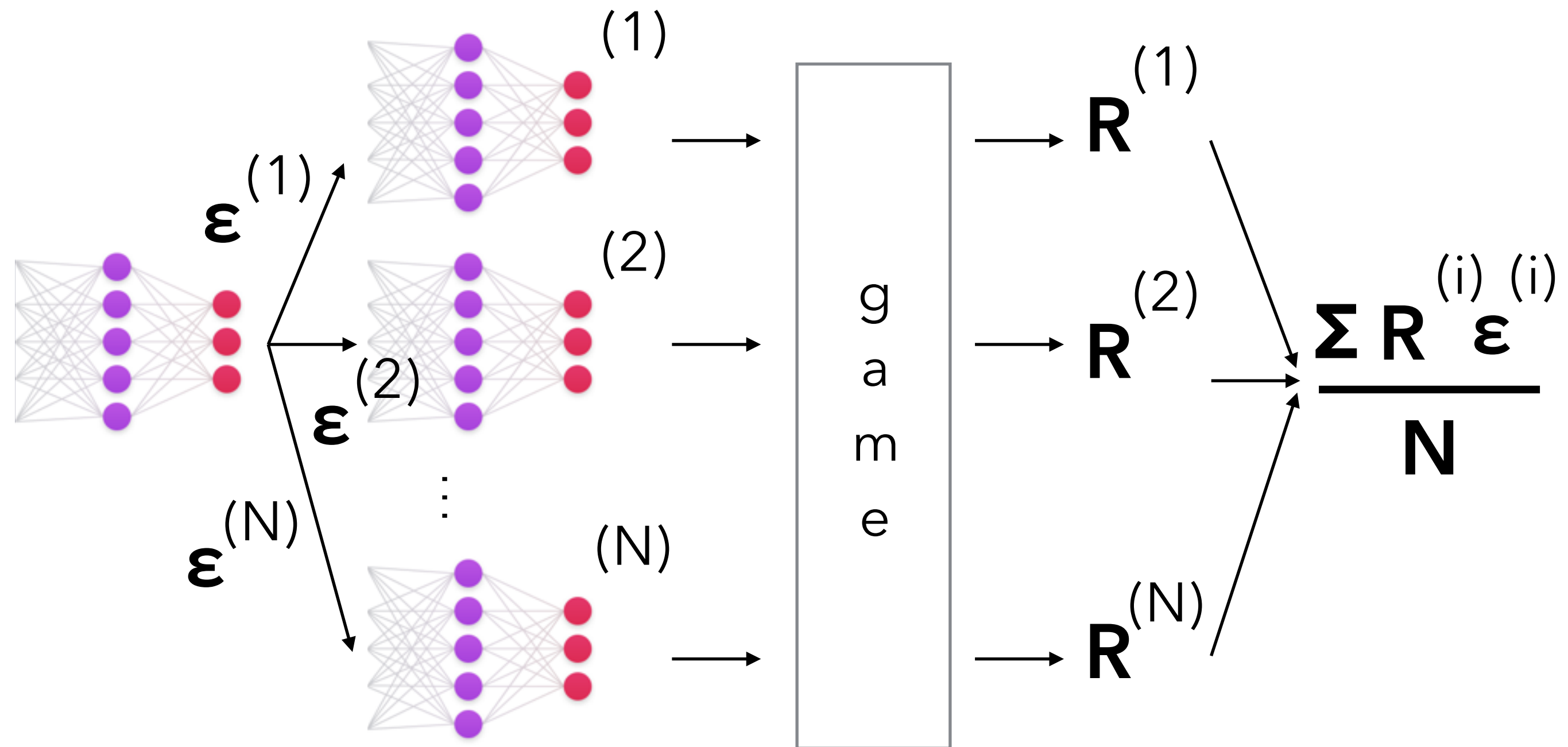
**Algorithm 1** Evolution Strategies

---


- 1: **Input:** Learning rate  $\alpha$ , noise standard deviation  $\sigma$ , initial policy parameters  $\theta_0$
  - 2: **for**  $t = 0, 1, 2, \dots$  **do**
  - 3:     Sample  $\epsilon_1, \dots, \epsilon_n \sim \mathcal{N}(0, I)$
  - 4:     Compute returns  $F_i = F(\theta_t + \sigma\epsilon_i)$  for  $i = 1, \dots, n$
  - 5:     Set  $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^n F_i \epsilon_i$  
  - 6: **end for**
-



# ES in a Nutshell



# Another Cool View Point

- Agent: parameter tuner 
- Policy: gaussian policy
- Action: parameter perturbation
- Environment: RL agent's policy parameter + game



# Advantages of ES

- No need for BP
- Easier to scale
- Immune to sparse rewards
- Robust to hyperparameters

# Advantages of ES

- No need for BP
- Easier to scale
- Immune to sparse rewards
- Fewer hyperparameters

# No Need for BP

- Write code faster / easier
- Code runs faster (only 1/3 computation)
- Use CPU instead of GPU
- No gradient explosion (common in RNN)
- Memory efficient

# Advantages of ES

- No need for BP
- Easier to scale
- Immune to sparse rewards
- Fewer hyperparameters

# Easier to Scale

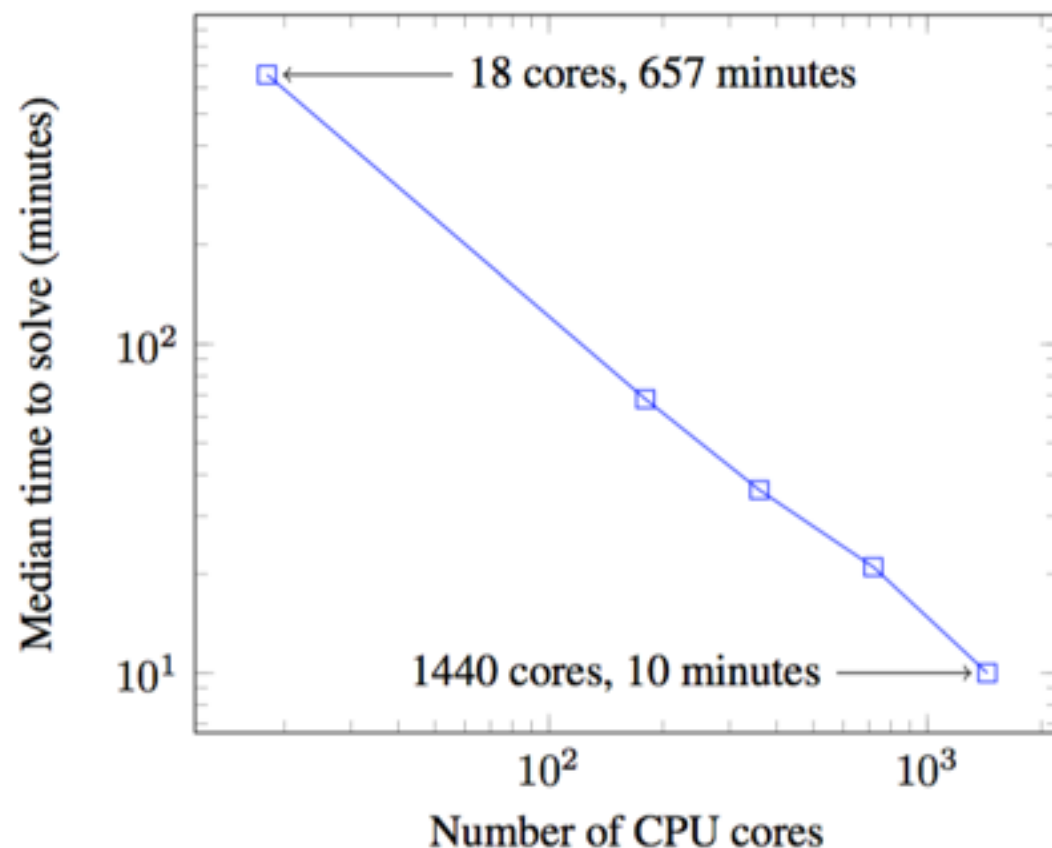



Figure 1. Time to reach a score of 6000 on 3D Humanoid with different number of CPU cores. Experiments are repeated 7 times and median time is reported.

- Performance boosts linearly with #CPUs
- Different workers only need to pass reward to each other, not gradient
- $\$CPU \ll \$GPU$

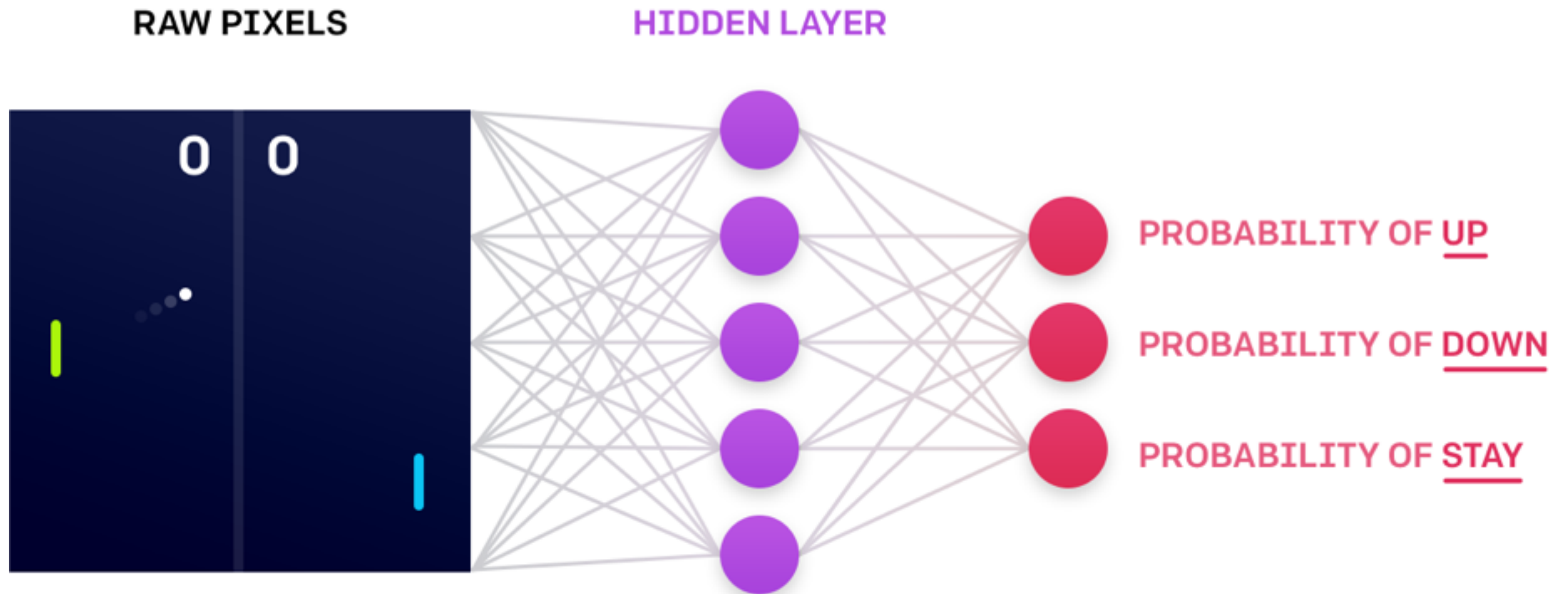
# The Killer Feature of ES

- Every worker's random seed are shared 
- Can directly reconstruct other worker's parameter
- Only need to pass scalar **R**

# Advantages of ES

- No need for BP
- Easier to scale
- Immune to sparse rewards
- Fewer hyperparameters

# Recap: Deep RL



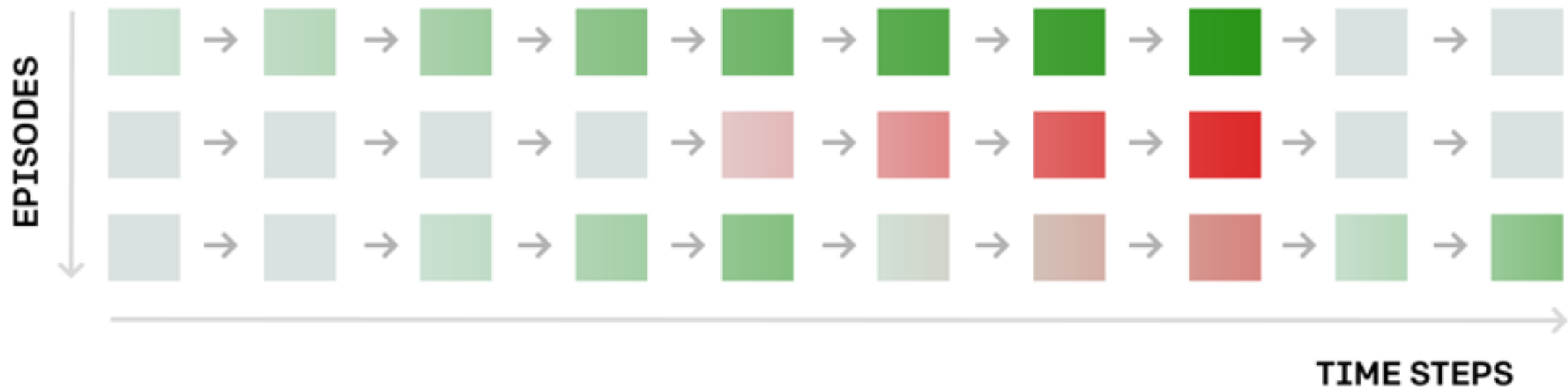


# Recap: Deep RL Training Process

■: + reward

■: - reward

Sampled Minibatch:



# Gradient of RL & ES

- PG adds noise in **action space**

$$\nabla_{\theta} F_{PG}(\theta) = \mathbb{E}_{\epsilon} \{ R(\mathbf{a}(\epsilon, \theta)) \nabla_{\theta} \log p(\mathbf{a}(\epsilon, \theta); \theta) \} .$$

- ES adds noise in parameter space

$$\nabla_{\theta} F_{ES}(\theta) = \mathbb{E}_{\xi} \left\{ R(\mathbf{a}(\xi, \theta)) \nabla_{\theta} \log p(\tilde{\theta}(\xi, \theta); \theta) \right\} .$$

# Immune to Sparse Rewards

- Policy Gradient:

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

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

- ES:

$$\text{Var}[\nabla_{\theta} F_{ES}(\theta)] \approx \text{Var}[R(\mathbf{a})] \text{Var}[\nabla_{\theta} \log p(\tilde{\theta}; \theta)].$$

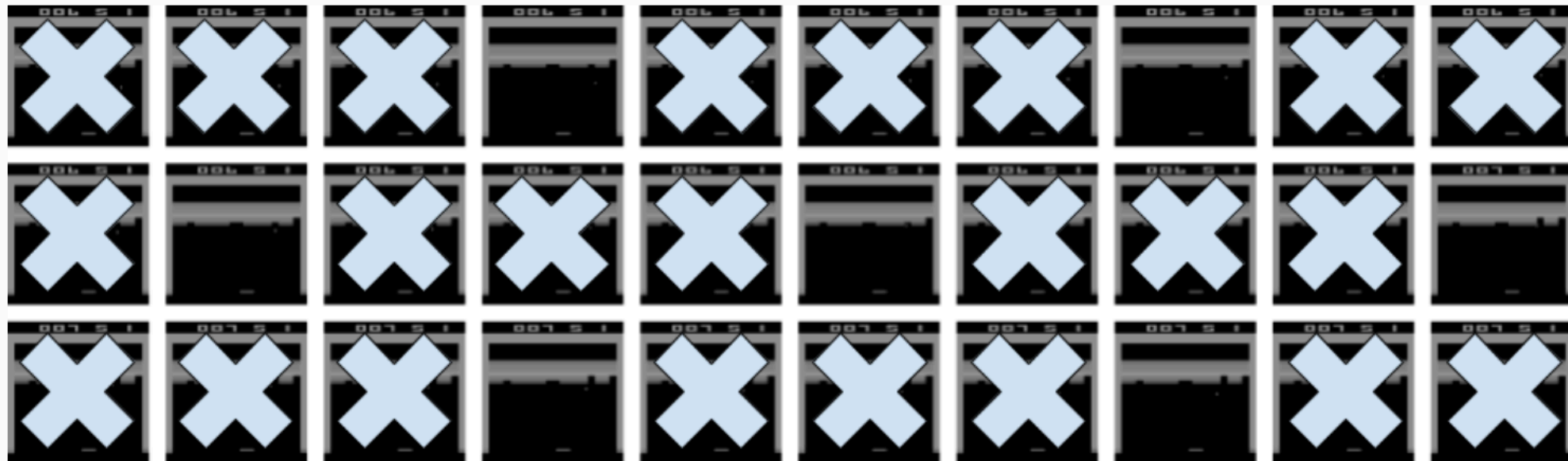


Variance of gradient estimated by ES  
does not grow linearly with T

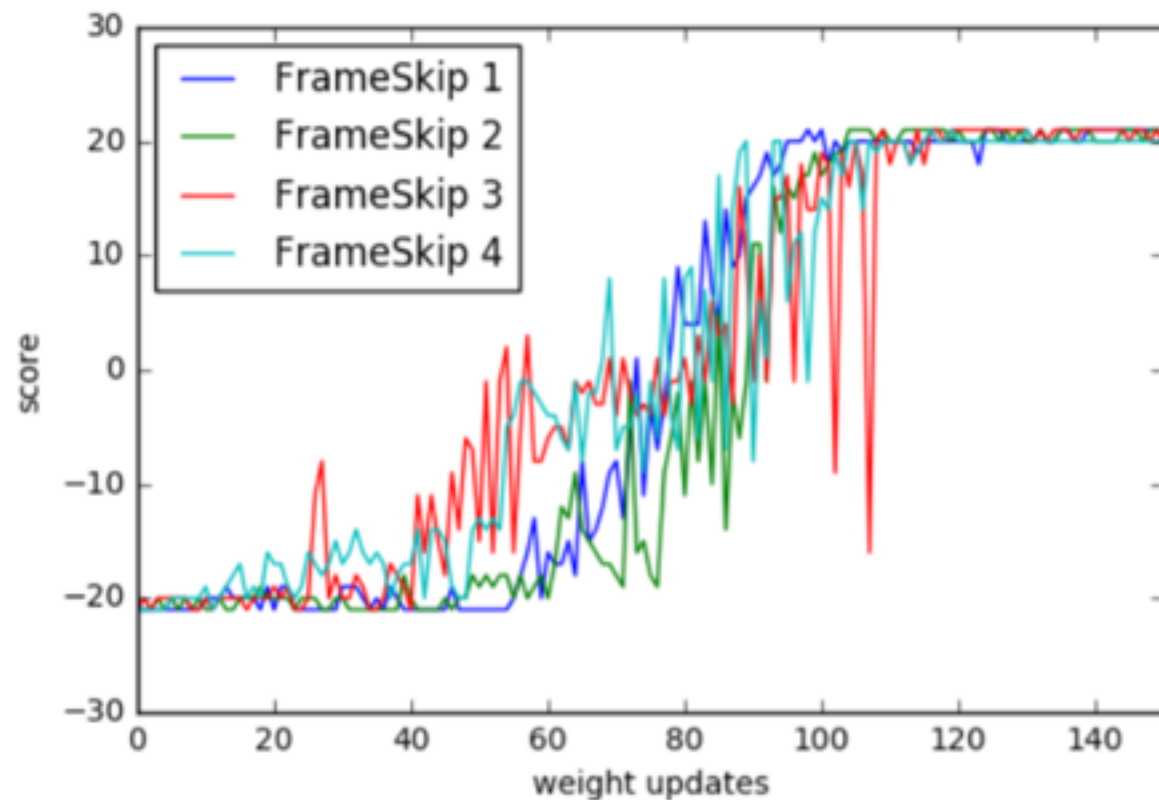
# Advantages of ES

- No need for BP
- Easier to scale
- Immune to sparse rewards
- Robust to hyperparameters

## Frame Skip



# Insensitive to Hyperparameters



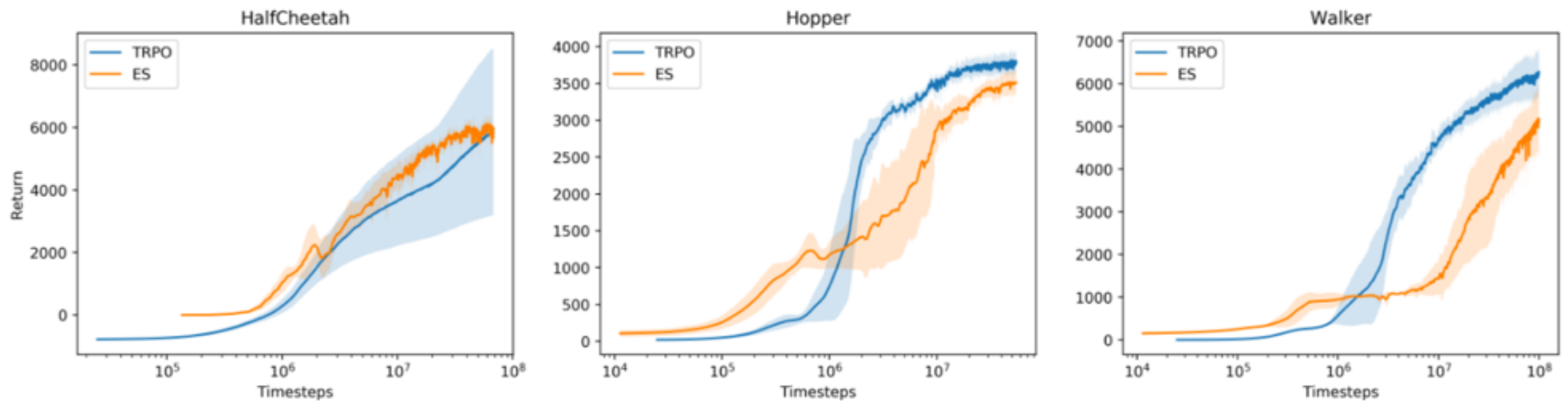
- No need to tune annoying hyperparameters.

# Advantages of ES

- No need for BP
- Easier to scale 👍
- Immune to sparse rewards 🤔
- Robust to hyperparameters

# Disadvantage of ES

- Less data efficient



At most needs 7.8x data on Mujoco, looks acceptable





RL v.s ES

*“There are open-loop domain and closed-loop domain for RL.*

*In open-loop domain such as self-driving car, everything could happen.*

*In closed-loop domain such as electricity control, we can define our own MDP.”*

– Honglak Lee

# My Two Cents

- Open-loop: RL

Cost of data is **high**, e.g., robotics

- Closed-loop: ES

Cost of data is **low**, e.g., simulation