

# **Presentation**

**by RL NooB**

성균관대학교 19학번 박찬혁

# CONTENTS

- **Introduction**
  - **About ES**
  - **Conclusion**
- (+ off the record?)

# Introduction

*b..ackground?*

01

02

03

04

## Introduction

## About ES

## Conclusion

arXiv:1703.03864v2 [stat.ML] 7 Sep 2017

### Evolution Strategies as a Scalable Alternative to Reinforcement Learning

---

Tim Salimans   Jonathan Ho   Xi Chen   Szymon Sidor   Ilya Sutskever  
OpenAI

**Abstract**

We explore the use of Evolution Strategies (ES), a class of black box optimization algorithms, as an alternative to popular MDP-based RL techniques such as Q-learning and Policy Gradients. Evolution strategies have been used in a variety of fields as a viable solution strategy that scales extremely well with the number of CPUs available: By using a novel communication strategy based on common random numbers, our ES implementation only needs to communicate scalars, making it possible to scale to over a thousand parallel workers. This allows us to solve 3D humanoid walking in 10 minutes and obtain competitive results on most Atari games after one hour of training. In addition, we highlight several advantages of ES as a black box optimization technique: It is invariant to action frequency and delayed rewards, tolerant of extremely long horizons, and does not need temporal discounting or value function approximation.

**1 Introduction**

Developing agents that can accomplish challenging tasks in complex, uncertain environments is a key goal of artificial intelligence. Recently, the most popular paradigm for analyzing such problems has been using a class of reinforcement learning (RL) algorithms based on the Markov Decision Process (MDP) formalism and the concept of value functions. Successes of this approach include systems that learn to play Atari from pixels [Mnih et al., 2015], perform helicopter acrobatics Ng et al. [2006], or play expert-level Go [Silver et al., 2016].

An alternative approach to solving RL problems is using black-box optimization. This approach is known as direct policy search [Schmidhuber and Zhao, 1998], or neuro-evolution [Risi and Togias, 2015], when applied to neural networks. In this paper, we study Evolution Strategies (ES) [Rechenberg, 1973] as a type of black box optimization. We show that evolution strategies can show that ES can reliably train neural network policies, in a fashion well suited to be scaled up to modern distributed computer systems, for controlling robots in the MuJoCo physics simulator [Todorov et al., 2012] and playing Atari games with pixel inputs [Mnih et al., 2015]. Our key findings are as follows:

1. We found that the use of virtual batch normalization [Salimans et al., 2014] and other reparameterizations of the neural network policy (section 2.2) greatly improve the reliability of evolution strategies. Without these methods ES proved brittle in our experiments, but with these reparameterizations we achieved strong results over a wide variety of environments.
2. We found the evolution strategies method to be highly parallelizable: by introducing a novel communication strategy based on common random numbers, we are able to achieve linear speedups in run time even when using over a thousand workers. In particular, using 1,440 workers, we have been able to solve the MuJoCo 3D humanoid task in under 10 minutes.
3. The data efficiency of evolution strategies was surprisingly good: we were able to match the final performance of A3C [Mnih et al., 2016] on most Atari environments while using between 3x and 10x as much data. The slight decrease in data efficiency is partly offset by a

## Abstract

## 1. Introduction

## 2. Evolution Strategies

## 3. Smoothing in parameter space versus

smoothing in action space

## 4. Experiments

## 5. Related Work

## 6. Conclusion

01

02

03

04

## **Black-Box Optimization**

01

02

03

04

## Black-Box Optimization



Not this black-box

01

02

03

04

## Black-Box Optimization



01

## Black-Box Optimization

02

03

04

입력값에 따라 출력값이 나타나는 장치 or 시스템

INPUT ->



-> OUTPUT

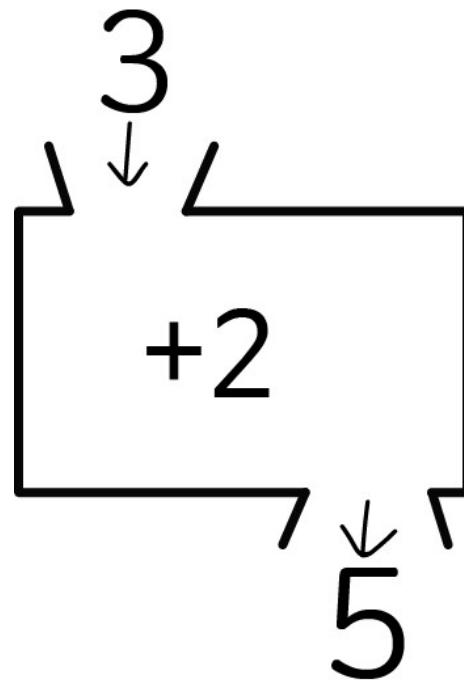
01

## Black-Box Optimization

02

03

04



출처 : 네이버 블로그 오크형제

<https://blog.naver.com/darkhoho2/221710599150>

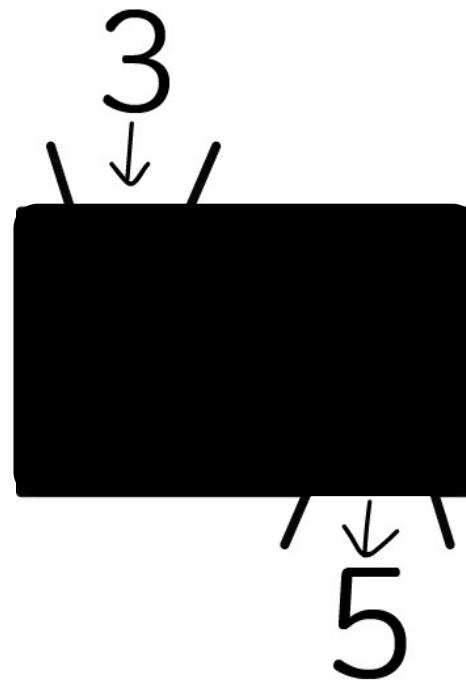
01

## Black-Box Optimization

02

03

04



출처 : 네이버 블로그 오크형제

<https://blog.naver.com/darkhoho2/221710599150>

01

## Black-Box Optimization

02

03

04

입력값에 따라 출력값이 나타나는 장치 or 시스템

INPUT ->



-> OUTPUT

함수와의 차이점은?

01

## Black-Box Optimization

02

03

04

입력값에 따라 출력값이 나타나는 장치 or 시스템

INPUT ->



-> OUTPUT

내부 알고리즘이 어떻게 결론을 도출하는지는 확인할 수 없음

01

## Black-Box Optimization

02

03

04

**INPUT ->**

**-> OUTPUT**

**INPUT과 OUTPUT 사이의 관계를 논하기 위해 사용**

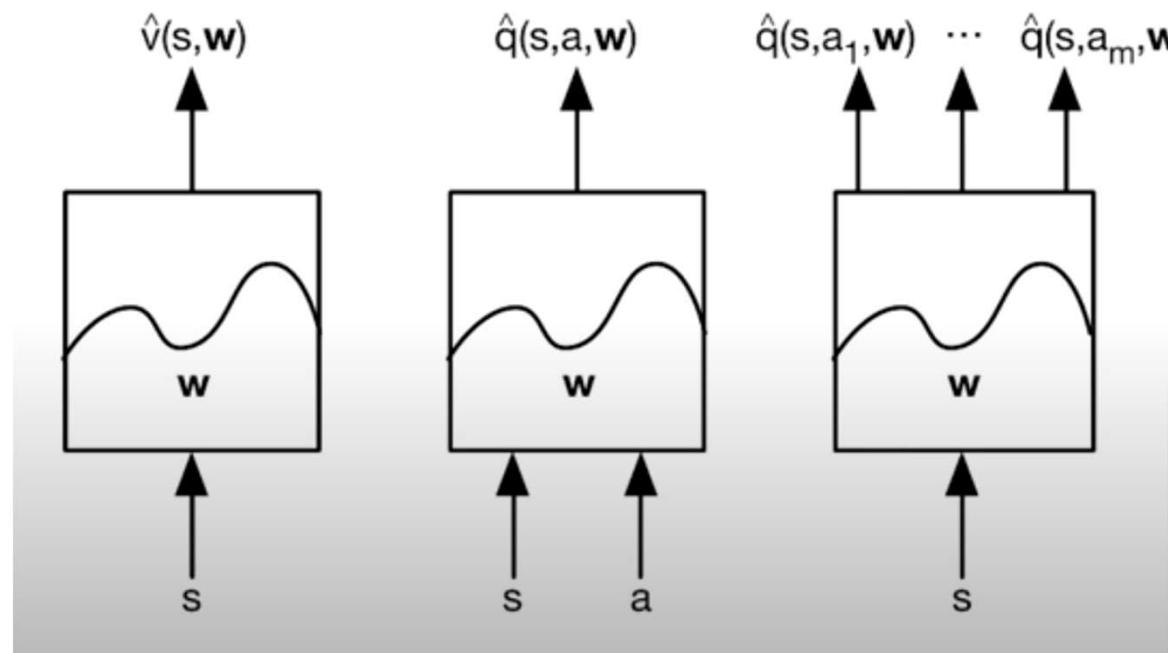
01

## Black-Box Optimization

02

03

04



&gt; Black-Box

출처 : 팡요랩 Pang-Yo Lab Youtube, [강화학습 6강] Value Function Approximation

<https://www.youtube.com/watch?v=71nH1BUjhNw>

01

## MuJoCo

02

03

04

**MuJoCo는 우분투 환경에서 사용가능한 로봇 시뮬레이터 중 하나로,  
코드 기반으로 제공되는 시뮬레이터이다.**

**다른 시뮬레이터에 비해 환경 구축이 어렵지만 빠른 속도를 낼 수 있다.**

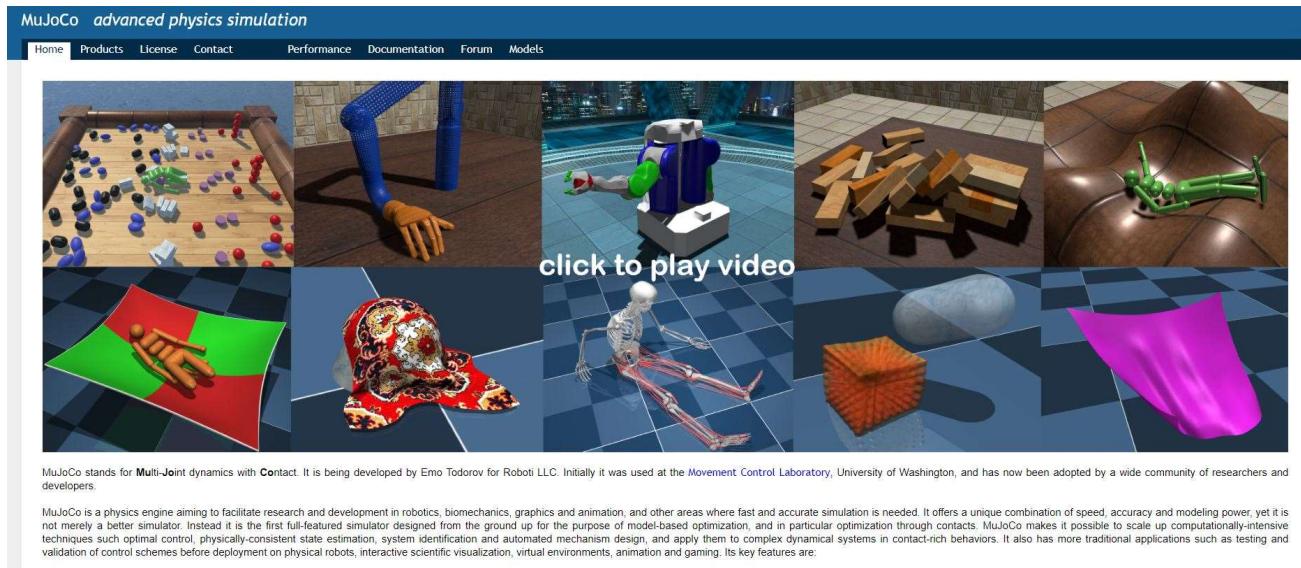
01

02

03

04

## MuJoCo

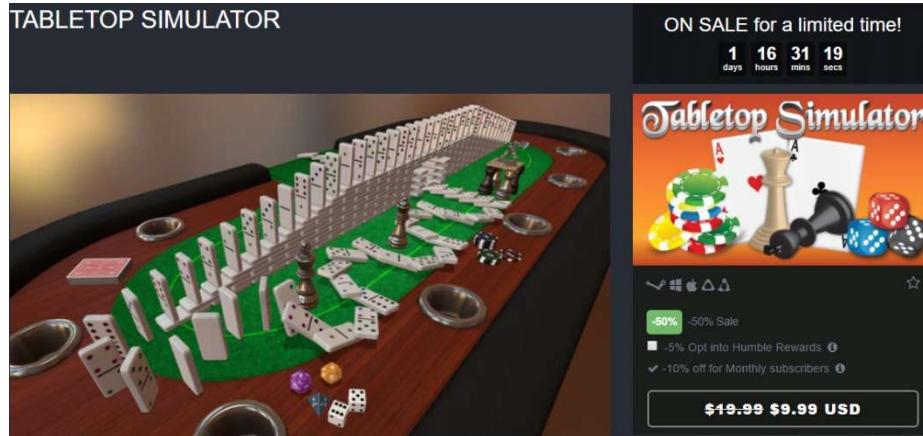


> MuJoCo 홈페이지

01

## MuJoCo

02



03

04



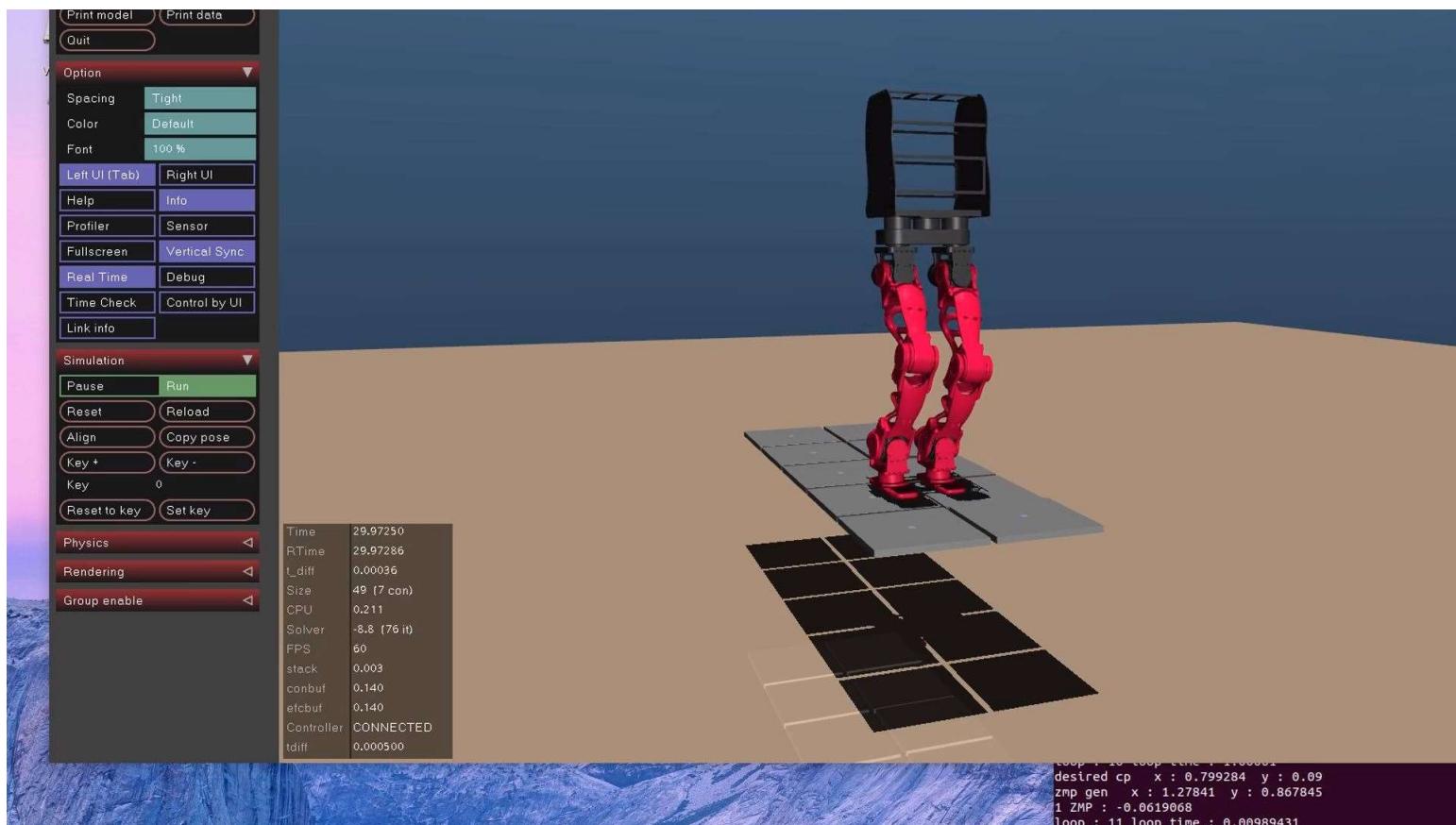
01

02

03

04

## MuJoCo



> MuJoCo 사용 모습

출처 : 네이버 블로그

<https://blog.naver.com/wnsqnl01/221731474073>

01

## Policy Gradient

02

03

04

강화학습의 목표는 Agent를 위한 최적의 행동 전략을 찾는 것이다.

Q-value를 계산하지 않고 직접적으로 policy를 모델링하고 최적화하는 방법이다.

이를 활용하면 value를 계산하지 못하는 action에 대해서도 학습할 수 있다.

> 실제 세상과 가까운 환경을 학습할 수 있다 (DQN과의 차별점)

01

02

03

04

## 현재까지 주로 연구되었던 Issue

**MDP (Markov Decision Process) 기반 RL 기법**

> Q-learning, Policy Gradient and etc.

**보상 구조가 복잡하거나 오랜 시간 연구해야 하는 분야에서는 적합성이 떨어진다**

**파라서, 새로운 알고리즘의 도입이 필요하다고 느껴짐.**

# About ES

*Quality of ES, Experiments*

01

## What is ES?

02

03

04

05

06

07

**Evolution Strategy (ES), 진화 전략 (또는 진화 알고리즘)**

**부모 개체와 자식 개체를 확인하여 어떻게 진화해 나갈 것이지 선택하는 알고리즘**

**랜덤 교배, 돌연변이 및 선택 등에 대한 가중치를 부여함으로써 자연계 모방**

**다수 개체로 구성된 집단까지 확장시키면 거의 모든 종류의 최적화 문제에 적용 가능**

01

## What is ES?

02

03

$(\mu + \lambda)$  or  $(\mu, \lambda)$  진화전략

04

$(\mu :$ 집단의 크기,  $\lambda :$ 매 세대에서 발생하는 자식 개체 수 $)$

초기  $(1 + 1)$  진화전략에서 시작하여 목적함수를 도출한다.

05

여기서 목적함수는 원하는 결과일 수도 있고, 단순 시뮬레이션 결과일 수도 있다.

06

$(\mu + \lambda)$  진화전략은 뛰어난 개체를 선발하는데 사용

$(\mu, \lambda)$  진화전략은 결정론적 선택으로 자연적인 결과를 알아보는데 사용

출처 : 컴퓨터에 의한 진화계산 및 진화디자인, 황희수 (내하출판사, 2002)

[http://www.aistudy.com/biology/genetic/evolution\\_strategy\\_hwang.htm](http://www.aistudy.com/biology/genetic/evolution_strategy_hwang.htm)

01

## What is ES?

02

03

04

05

06

07

```

// 시간을 초기화하고 시작
t = 0;
// 임의의 값으로 개체 집단을 초기화
initPopulation P(t), P(t) = { $\alpha_1(t), \alpha_2(t), \dots, \alpha_\mu(t)$ }
 $\alpha_k = (x_{1k}, \dots, x_{nk}, \sigma_{1k}, \dots, \sigma_{nk}, \alpha'_{1k}, \dots, \alpha'_{nk})$ ;
// 집단내 모든 개체의 적합도를 평가
evalPopulation P(t) : { $\Phi(\alpha_1(t)), \dots, \Phi(\alpha_\mu(t))$ }
 $\Phi(\alpha_k(t)) = F(x_{1k}, \dots, x_{nk})$ 

```

```

// 종료 조건 (시간 또는 적합도) 을 만족하지 않으면 계속 수행
while (not 종료조건) do {
    // 개체 집단에 재결합 적용
    recombine :  $\alpha'_{ik} = r(P(t))$ ,  $r$  는 재결합 연산자
    // 개체 집단에 돌연변이 적용
    mutate :  $\alpha''_{ik} = m'(a)(x'_{1k}, \dots, x'_{nk}, \sigma'_{1k}, \dots, \sigma'_{nk}, \alpha'_{1k}, \dots, \alpha'_{nk})$ 
            k = 1, 2, ...,  $m'(a)$  돌연변이 연산자
    // 새로운 개체 집단의 적합도를 평가
    evaluate  $P''(t) : \{\Phi(\alpha''_1(t)), \dots, \Phi(\alpha''_\mu(t))\}$ 
 $\Phi(\alpha''_k(t)) = F(x''_{1k}, \dots, x''_{nk})$ 
    // 실제 적합도로부터 확률적으로 생존 개체의 선택
     $P(t+1) = select\{s_{(\mu,\lambda)}(P''(t)) or s_{(\mu+\lambda)}(P(t) \cup P''(t))\}$ 
             $s_{(\mu,\lambda)}$  와  $s_{(\mu+\lambda)}$  는 각각  $(\mu, \lambda)$  와  $(\mu + \lambda)$  선택
    // 세대 수의 증가
    t = t + 1;
}
end;

```

출처 : 컴퓨터에 의한 진화계산 및 진화디자인, 황희수 (내하출판사, 2002)

[http://www.aistudy.com/biology/genetic/evolution\\_strategy\\_hwang.htm](http://www.aistudy.com/biology/genetic/evolution_strategy_hwang.htm)

01

## How to use?

02

**자연 진화 전략 (NES, Natural Evolution Strategies)에 속함**

03

> 선형연구에서 모집단에 대한 평균 목표값을 최대화하는 알고리즘

04

05

06

07

$$\nabla_{\psi} \mathbb{E}_{\theta \sim p_{\psi}} F(\theta) = \mathbb{E}_{\theta \sim p_{\psi}} \{F(\theta) \nabla_{\psi} \log p_{\psi}(\theta)\}$$

> 선형연구에서 사용한 점수 함수 추정기

F : 매개변수 theta에 작용하는 목적 함수

$p_{\psi}$  : 매개변수화 된 모집단

$E_{\theta \sim p_{\psi}}$  : 모집단에 대한 평균 목표값

01

## How to use?

02

03

### 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**
- 

04

05

06

07

1) 주어진 환경에서 episode(action)를 취하고 결과값을 평가한다.

이 결과를 바탕으로 확률적인 가중치를 매긴다.

2) 여러 episode의 결과를 바탕으로 확률적 추정치를 계산하고 업데이트한다.

위 2가지 방법을 반복하는 것이 ES의 기본 전략이다.

01

## What's the difference?

02

03

본 논문에서 제시한 parallel version of ES이다.

04

알고리즘이 랜덤 시드를 공유하는 모습을 볼 수 있다.

05

> Workers 간 통신 대역폭을 획기적으로 줄임

06

07

---

### 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**  $t = 0, 1, 2, \dots$  **do**
  - 4:   **for** each worker  $i = 1, \dots, n$  **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  $i = 1, \dots, n$  **do**
  - 10:     Reconstruct all perturbations  $\epsilon_j$  for  $j = 1, \dots, n$  using known random seeds
  - 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**
-

01

## What's the difference?

02

03

본 논문에서 제시한 parallel version of ES이다.

04

Each workers가 시간에 따라 병렬적으로 작업하도록 구현했다.

05

06

07

---

### 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  $t = 0, 1, 2, \dots$  do
4:   for each worker  $i = 1, \dots, n$  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  $i = 1, \dots, n$  do
10:    Reconstruct all perturbations  $\epsilon_j$  for  $j = 1, \dots, n$  using known random seeds
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
```

---

01

## What's the difference?

02

03

알고리즘이 랜덤 시드를 공유하는 모습을 볼 수 있다.

04

> Workers 간 통신 대역폭을 획기적으로 줄임

05

06

07

---

### 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**  $t = 0, 1, 2, \dots$  **do**
  - 4:   **for** each worker  $i = 1, \dots, n$  **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  $i = 1, \dots, n$  **do**
  - 10:     Reconstruct all perturbations  $\epsilon_j$  for  $j = 1, \dots, n$  using known random seeds
  - 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**
-

01

## What's the difference?

02

03

소음 변수(sigma)에 Gaussian noise(백색잡음)를 도입하여

04

변화시킴으로써 샘플링을 구현했다.

05

06

07

학습하는 부분인 9~12줄에서 유의미한 시간 감소를 보였다.

---

### 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**  $t = 0, 1, 2, \dots$  **do**
  - 4:   **for** each worker  $i = 1, \dots, n$  **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  $i = 1, \dots, n$  **do**
  - 10:     Reconstruct all perturbations  $\epsilon_j$  for  $j = 1, \dots, n$  using known random seeds
  - 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**
-

01

**What's the difference?**

02

03

04

**Q-learning과 Policy Gradient는 확률적 정책에서 얻은 샘플을 통해 학습하는데**

05

06

**ES는 매개변수를 sampling instantiation하여 학습한다.**

07

**때문에 ES는 매개변수와 노이즈를 고려하여****다음 매개변수를 추출하는 것이 중요하다.**

01

## With Atari

02

03

**Atari 환경의 경우, Virtual Batch Normalization(VBN)을 사용하여**

04

**대부분 게임에서 정책을 모델링하는데 있어 경쟁력을 가진다.**

05

06

07

**그러나 정책의 가중치가 랜덤일 경우 초기 단계에서**

**작은 변화에도 민감하게 받아들이기에 학습 과정을 expensive하게 한다.**

**그러나 본 논문에서는 정상화된 통계를 계산하기 위해**

**Episode 단계를 줄여 간접비용을 줄일 수 있었다.**

출처 : 네이버 블로그

<https://blog.naver.com/kakm12/221746144449>

01

## With MuJoCo

02

03

04

거의 모든 환경에서 사용가능한 수준의 성능을 달성했다.

05

그러나 일부 환경에서 행동의 변별을 주어서

06

다른 행동을 고려하는 모습을 보였다.

07

> 앞으로 걷던 로봇이 옆이나 뒤로 걷는 모습을 보임

더욱 다양한 환경 제공의 필요성을 느낌

01

## More valuable

02

03

04

05

각각의 action  $a_i$ (parameter  $\theta$ )에 대해서

값을 반환하는  $R$  함수를 가정하면

06

아래와 같은  $F$  함수를 최적화하는 것을 목표로 할 수 있다.

07

(action은 확률적 정책 함수에 의해 결정)

$$F(\theta) = R(a(\theta))$$

01

**More valuable**

02

03

04

05

각각의 action  $a_i$ (parameter  $\theta$ )에 대해서

값을 반환하는  $R$  함수를 가정하면

06

아래와 같은  $F$  함수를 최적화하는 것을 목표로 할 수 있다.

07

(action은 확률적 정책 함수에 의해 결정)

$$F(\theta) = R(a(\theta))$$

01

**More valuable**

02

03

04

05

06

07

F(theta)는 theta에 대해서 non-smooth 할 수 있다.

따라서 gradient 기반 최적화 방법을 사용할 수는 없다.

이를 smooth하게 만들기 위해 필요한 것은?

$$F(\theta) = R(\mathbf{a}(\theta))$$

01

## More valuable

02

03

04

05

06

07

$F(\theta)$ 는  $\theta$ 에 대해서 non-smooth 할 수 있다.

따라서 gradient 기반 최적화 방법을 사용할 수는 없다.

이를 smooth하게 만들기 위해 필요한 것은?

수행되는 작업 공간의 noise를 추가한다.

ES는 매개변수 공간에 noise를 추가해 준다.

따라서 action과 매개변수 둘 모두에게 noise를 추가해 줄 수 있다는 점에서

ES를 잘 활용할 수 있다.

01

## When it is better than?

02

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

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

05

## Using simple Monte Carlo(REINFORCE)

06

07 
$$\nabla_{\theta} \log p(a; \theta) = \sum_{t=1}^T \nabla_{\theta} \log p(a_t; \theta)$$

Policy Gradient 부분에서 T에 관한 함수로 표현된다.

반면에 ES도 이와 같이 표현되는데, T와는 무관한 함수이다.

파라서 시간이 오래 걸리는 긴 episode일수록,

좋은 value function이 없을수록 ES는 매력적인 선택지이다.

01

## Experiments

02

03

04

05

06

07

MuJoCo 환경에서 균형감각의 보정과 같은 고전적인 문제와  
2D hopping and walking과 같은 난해한 문제를 학습시킴

TRPO(Trust Region Policy Gradient)와 비슷한 성능을 보이기까지  
환경과 500만 timestep의 상호작용을 진행했다.

6개의 랜덤 시드에 대해서 실행한 결과, TRPO학습 진행률에 따른

ES to TRPO의 비율

Environment	25%	50%	75%	100%
HalfCheetah	0.15	0.49	0.42	0.58
Hopper	0.53	3.64	6.05	6.94
InvertedDoublePendulum	0.46	0.48	0.49	1.23
InvertedPendulum	0.28	0.52	0.78	0.88
Swimmer	0.56	0.47	0.53	0.30
Walker2d	0.41	5.69	8.02	7.88

01

## Experiments

02

03

04

51개의 10억 프레임에 대해 연산하는 Atari 2600 게임에 적용시켰다.

05

이전 연구결과는 3억 2천만 프레임에 대해 연산했는데,

06

동일한 양의 신경망 연산이 필요했다.

07

(ES가 backpropagation과 value func를 사용하지 않았기 때문)

23개의 게임에서 더 나은 성능을, 28개의 게임에서 더 나쁜 성능을 보였다.

01

02

03

04

05

06

07

# Experiments

Game	DQN	A3C FF, 1 day	HyperNEAT	ES FF, 1 hour	A2C FF
Amidar	133.4	283.9	184.4	112.0	<b>548.2</b>
Assault	3332.3	<b>3746.1</b>	912.6	1673.9	2026.6
Asterix	124.5	<b>6723.0</b>	2340.0	1440.0	3779.7
Asteroids	697.1	<b>3009.4</b>	1694.0	1562.0	1733.4
Atlantis	76108.0	772392.0	61260.0	1267410.0	<b>2872644.8</b>
Bank Heist	176.3	<b>946.0</b>	214.0	225.0	724.1
Battle Zone	17560.0	11340.0	<b>36200.0</b>	16600.0	8406.2
Beam Rider	8672.4	<b>13235.9</b>	1412.8	744.0	4438.9
Berzerk		<b>1433.4</b>	1394.0	686.0	720.6
Bowling	41.2	36.2	<b>135.8</b>	30.0	28.9
Boxing	25.8	33.7	16.4	49.8	<b>95.8</b>
Breakout	303.9	<b>551.6</b>	2.8	9.5	368.5
Centipede	3773.1	3306.5	<b>25275.2</b>	7783.9	2773.3
Chopper Command	3046.0	<b>4669.0</b>	3960.0	3710.0	1700.0
Crazy Climber	50992.0	<b>101624.0</b>	0.0	26430.0	100034.4
Demon Attack	12835.2	<b>84997.5</b>	14620.0	1166.5	23657.7
Double Dunk	<b>21.6</b>	0.1	2.0	0.2	3.2
Enduro	<b>475.6</b>	82.2	93.6	95.0	0.0
Fishing Derby	2.3	13.6	<b>49.8</b>	49.0	33.9
Freeway	25.8	0.1	29.0	<b>31.0</b>	0.0
Frostbite	157.4	180.1	<b>2260.0</b>	370.0	266.6
Gopher	2731.8	<b>8442.8</b>	364.0	582.0	6266.2
Gravitar	216.5	269.5	370.0	<b>805.0</b>	256.2
Ice Hockey	3.8	4.7	<b>10.6</b>	4.1	4.9
Kangaroo	2696.0	106.0	800.0	<b>11200.0</b>	1357.6

Game	DQN	A3C FF, 1 day	HyperNEAT	ES FF, 1 hour	A2C FF
Krull	3864.0	8066.6	<b>12601.4</b>	8647.2	6411.3
Montezuma's Revenge	50.0	<b>53.0</b>	0.0	0.0	0.0
Name This Game	5439.9	5614.0	<b>6742.0</b>	4503.0	5532.8
Phoenix		<b>28181.8</b>	1762.0	4041.0	14104.7
Pit Fall		<b>123.0</b>	0.0	0.0	8.2
Pong		16.2	11.4	17.4	<b>21.0</b>
Private Eye		298.2	194.4	<b>10747.4</b>	100.0
Q*Bert		4589.8	13752.3	695.0	147.5
River Raid		4065.3	<b>10001.2</b>	2616.0	5009.0
Road Runner		9264.0	31769.0	3220.0	16590.0
Robotank		<b>58.5</b>	2.3	43.8	11.9
Seaquest	<b>2793.9</b>	2300.2	716.0	1390.0	1763.7
Skiing		13700.0	7983.6	<b>15442.5</b>	15245.8
Solaris			1884.8	160.0	2090.0
Space Invaders		1449.7	<b>2214.7</b>	1251.0	678.5
Star Gunner		34081.0	<b>64393.0</b>	2720.0	1470.0
Tennis		2.3	10.2	0.0	4.5
Time Pilot		5640.0	5825.0	<b>7340.0</b>	4970.0
Tutankham		32.4	26.1	23.6	130.3
Up and Down		3311.3	54525.4	43734.0	67974.0
Venture		54.0	19.0	0.0	<b>760.0</b>
Video Pinball		20228.1	<b>185852.6</b>	0.0	22834.8
Wizard of Wor		246.0	<b>5278.0</b>	3360.0	3480.0
Yars Revenge			7270.8	<b>24096.4</b>	16401.7
Zaxxon		831.0	2659.0	3000.0	<b>6380.0</b>

# Conclusion

*With Q&A*

01

Random Seed를 사용함으로써 진화 전략의 신뢰성을 크게 향상시켰다.

02

난수를 활용한 새로운 통신 전략(병렬)을 도입하여  
런타임 속도를 선형적으로 상승시켰다.

ES가 TRPO보다 더 나은 탐색 활동을 보인다는 것을 발견하여  
TRPO와 다른 탐색 과정을 거친다는 것을 알아냈다. (related black-box)

Backpropagation과 value func를 사용하지 않기에  
같은 시간에 적으면 3배, 많으면 10배까지의 데이터를 사용할 수 있다.

모든 Atari 환경에서 고정된 Hyperparameter를 사용하고  
모든 MuJoCo 환경에서 다른 set of Hyperparameter를 사용한다는 점에서  
ES가 견고하다는 것을 알아냈다.

01

02

따라서,

ES는 보상 지연과 행동 빈도에 불변하며  
시간이 오래 걸리는 학습일수록 유리하다는 점에서  
**Q-learning, Policy Gradient보다 경쟁력있는 알고리즈다.**

# Q & A

*With off the record*

**Thank you**