Gradient free optimization methods

Arjun Rao, Thomas Bohnstingl, Darjan Salaj
Institute of Theoretical Computer Science

Why is this interesting?

- Backpropagating gradient through the environment is not always possible.
- When sampling the gradient of reward using policy gradient, the **variance** of the gradient **increases with the length** of the episode.
- Implementing backpropagation on a **neuromorphic chip** is nontrivial/not possible

ES as stochastic gradient ascent

- The ES update aims to maximize the following fitness function

$$J(\psi) = E_{p_\psi(\theta)}\left[F(\theta)\right]$$

Where $F(\theta)$ is the fitness function that is to be optimized

- This gives the following update rule

$$\begin{aligned}
\nabla_\psi J(\psi) &= \nabla_\psi E_{p_\psi(\theta)}\left[F(\theta)\right] \\
&= E_{p_\psi(\theta)}\left[F(\theta)\nabla_\psi \log p_\psi(\theta)\right] \dashrightarrow \text{[reinforce trick]} \\
&\approx \frac{1}{N}\sum_{i=1}^{N} F(\theta_i)\nabla_\psi \log p_\psi(\theta_i)
\end{aligned}$$

Wierstra et. al. 2014

ES as stochastic gradient descent

- The OpenAI-ES Algorithm is derived by the following

$$\psi \triangleq \boldsymbol{\mu}$$

$$\theta \sim N\left(\boldsymbol{\mu}, \sigma\right)$$

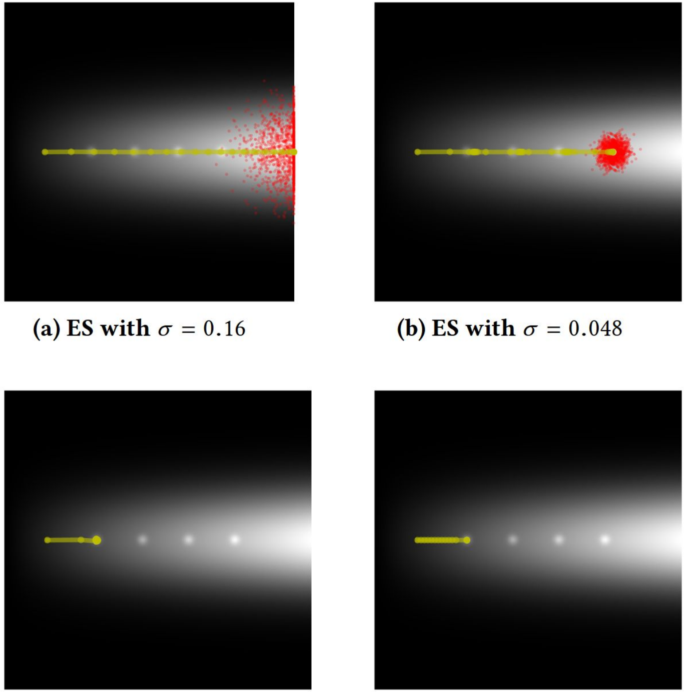
- This leads to the following update:

$$\begin{aligned}\nabla_{\boldsymbol{\mu}} J_{\boldsymbol{\mu}}(\theta) &\approx \frac{1}{N}\sum_{i=1}^{N} F(\theta_i)\nabla_{\boldsymbol{\mu}} \log p_{\boldsymbol{\mu}}(\theta_i) \\ &= \frac{1}{N\sigma^2}\sum_{i=1}^{N} F(\theta_i)\left(\theta_i - \boldsymbol{\mu}\right)\end{aligned}$$

Wierstra et. al. 2014

ES vs Finite Difference

- Finite difference estimates the gradient of $F(\theta)$ instead of $J(\psi)$
- ES with a high enough variance is not caught by local variations



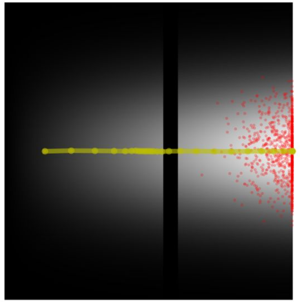
(a) ES with $\sigma = 0.16$ **(b) ES with** $\sigma = 0.048$

(c) ES with $\sigma = 0.002$ **(d) FD with** $\epsilon = 1e - 7$

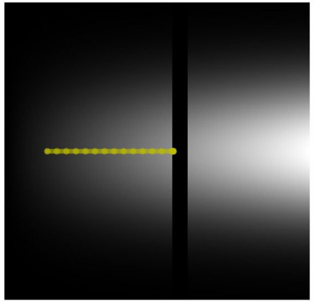
Joel Lehman et. al., 2018

ES vs Finite Difference

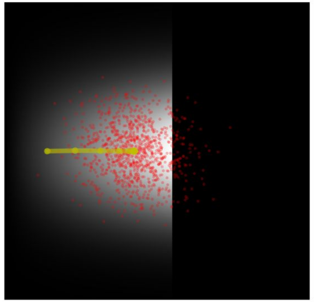
- Finite difference estimates the gradient of $F(\theta)$ instead of $J(\psi)$
- ES with a high enough variance is not caught by local variations



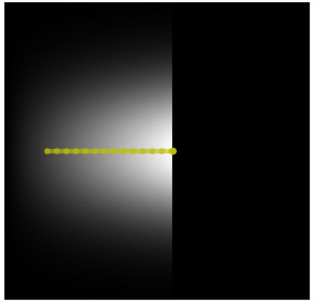
(a) ES with $\sigma = 0.18$



(b) Finite Differences



(d) ES with $\sigma = 0.18$

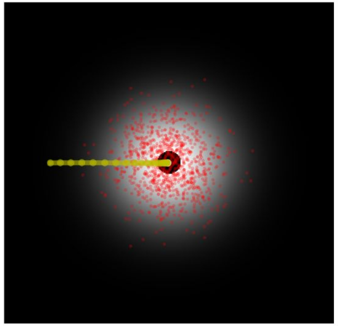


(e) Finite Differences

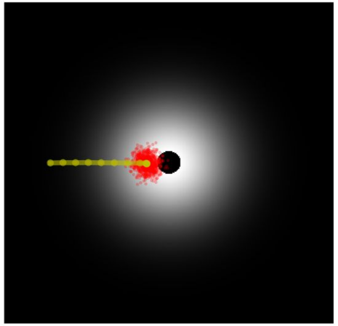
Joel Lehman et. al., 2018

ES vs Finite Difference

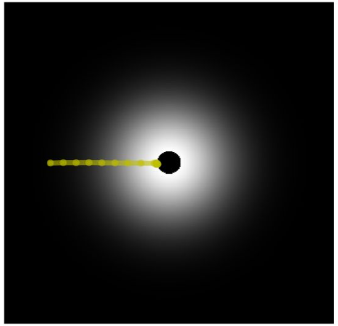
- Finite difference estimates the gradient of $F(\theta)$ instead of $J(\psi)$
- ES with a high enough variance is not caught by local variations



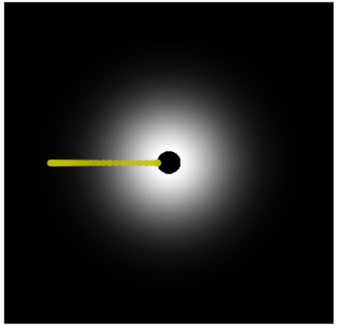
(a) ES with $\sigma = 0.16$



(b) ES with $\sigma = 0.04$



(c) ES with $\sigma = 0.002$

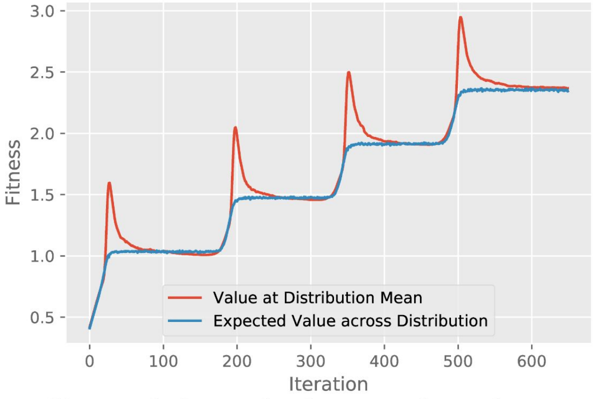


(d) FD with $\epsilon = 1e - 7$

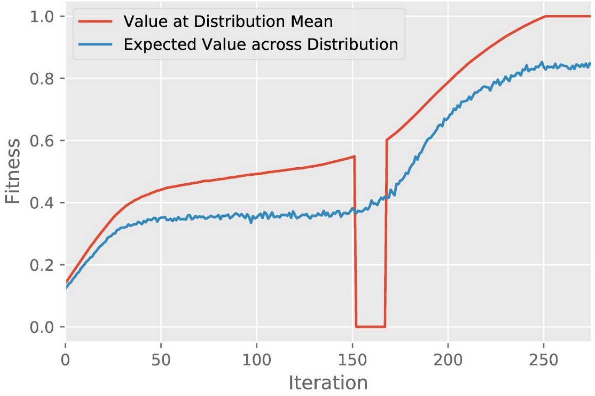
Joel Lehman et. al., 2018

ES vs Finite Difference

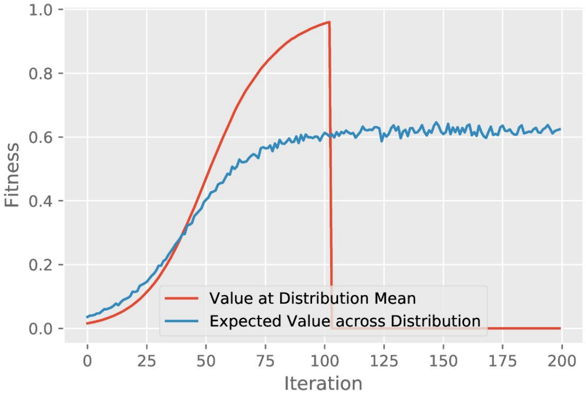
- Finite difference estimates the gradient of $F(\theta)$ instead of $J(\psi)$
- ES with a high enough variance is not caught by local variations



(b) Reward of ES on the Fleeting Peaks Landscape



(c) Reward of ES on the Gradient Gap Landscape

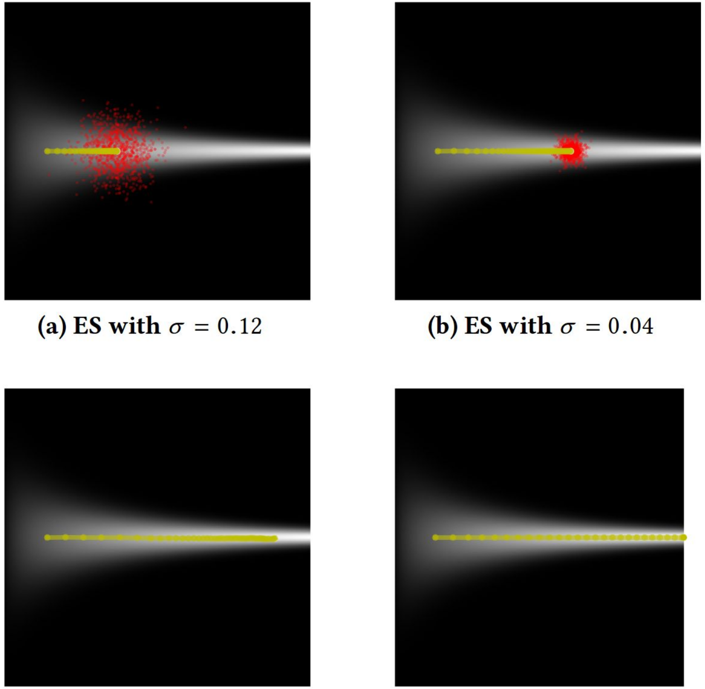


(a) Reward of ES on the Donut Landscape

Joel Lehman et. al., 2018

ES vs Finite Difference

- Finite difference estimates the gradient of $F(\theta)$ instead of $J(\psi)$
- ES with a high enough variance is not caught by local variations
- ES ends up selecting parameter regions with lower parameter sensitivity



(a) ES with $\sigma = 0.12$

(b) ES with $\sigma = 0.04$

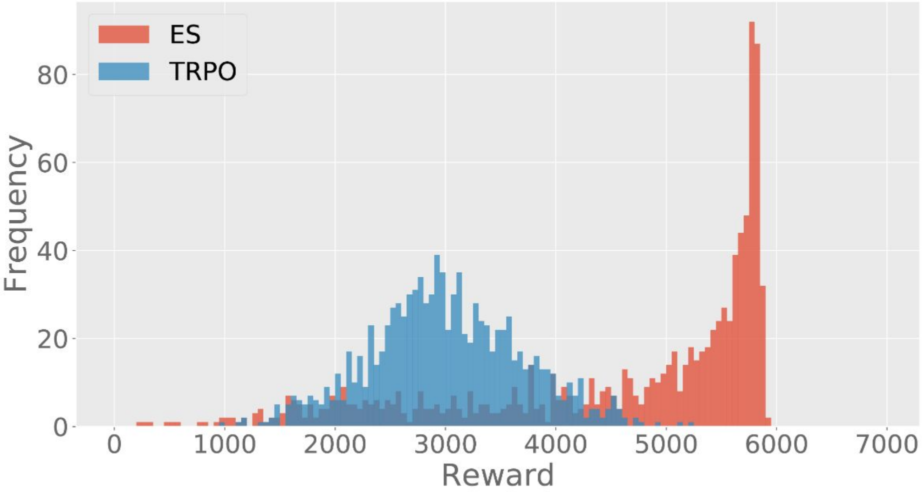
(c) ES with $\sigma = 0.0005$

(d) FD with $\epsilon = 1e - 7$

Joel Lehman et. al., 2018

ES vs Finite Difference

- Finite difference estimates the gradient of $F(\theta)$ instead of $J(\psi)$
- ES with a high enough variance is not caught by local variations.
- ES ends up selecting parameter regions with lower parameter sensitivity



(c) ES ($\sigma = 0.02$) vs TRPO ($\sigma = 0.02$)

Joel Lehman et. al., 2018

Variants of ES

Changing the distribution parameterization

- Covariance Matrix Adaptation - ES (Hansen and Ostermeier, 2001)

Using the natural gradient

- Exponential Natural Evolution Strategies (xNES) (Wierstra et.al. 2014)

Changing distribution family

- Using heavy tailed cauchy distribution for multi-modal objective functions (Wierstra et.al. 2014)

Parallelizability

- OpenAI-ES is highly parallelizable
- Each worker generates own copy of individuals
- Consistent random generator ensures coherence
- **Each worker then simulates one of those individuals and returns the fitness**.
- The fitness is communicated across all workers (all-to-all)
- Each worker then determines the next individual based on the communicated fitnesses

Salimans et. al. 2017

In Neuromorphic Hardware

Pros:

- No backpropagation implies that **most computation** is spent on **calculating the fitness function**
- Neuromorphic hardware will enable **very efficient parallel fitness evaluation** of spiking neural networks.

In Neuromorphic Hardware

Potential Pitfalls:

- Serialization involved in communication with hardware
- Limits on parallel computation on Host Processor

Some Solutions:

- Limit data communicated by only perturbing subset of parameters
- Implementation tricks of ES serve to reduce Host processor computation.

Canonical ES

Back to Basics: Benchmarking Canonical Evolution Strategies for Playing Atari

Patryk Chrabaszcz, Ilya Loshchilov, Frank Hutter
University of Freiburg, Freiburg, Germany

arXiv:1802.08842, 2018

- Simpler algorithm then OpenAI version of NES
- Outperforms OpenAI ES on some Atari games
- **Qualitatively different solutions**
  - Exploits game design, finds bugs

Comparison of OpenAI ES and Canonical ES

Algorithm 1: OpenAI ES

Input:
$optimizer$ - Optimizer function
σ - Mutation step-size
λ - Population size
θ_0 - Initial policy parameters
F - Policy evaluation function

for t = *0, 1, ...* **do**
  for i = *1, 2, ...* $\frac{\lambda}{2}$ **do**
    Sample noise vector: $\epsilon_i \sim \mathcal{N}(0, I)$
    Evaluate score in the game: $s_i^+ \leftarrow F(\theta_t + \sigma * \epsilon_i)$
    Evaluate score in the game: $s_i^- \leftarrow F(\theta_t - \sigma * \epsilon_i)$
  Compute normalized ranks: $r = ranks(s), r_i \in [0, 1)$
  Estimate gradient: $g \leftarrow \frac{1}{\sigma * \lambda} \sum_{i=1}^{\lambda}(r_i * \epsilon_i)$
  Update policy network: $\theta_{t+1} \leftarrow \theta_t + optimizer(g)$

Algorithm 2: Canonical ES Algorithm

Input:
σ - Mutation step-size
θ_0 - Initial policy parameters
F - Policy evaluation function
λ - Offspring population size
μ - Parent population size
Initialize :
$w_i = \frac{\log(\mu+0.5)-\log(i)}{\sum_{j=1}^{\mu} \log(\mu+0.5)-\log(j)}$

for $t = 0, 1, ...$ **do**
  for $i = 1...\lambda$ **do**
    Sample noise: $\epsilon_i \sim \mathcal{N}(0, I)$
    Evaluate score in the game: $s_i \leftarrow F(\theta_t + \sigma * \epsilon_i)$
  Sort $(\epsilon_1, \ldots, \epsilon_\lambda)$ according to s (ϵ_i with best s_i first)
  Update policy: $\theta_{t+1} \leftarrow \theta_t + \sigma * \sum_{j=1}^{\mu} w_j * \epsilon_j$
  Optionally, update step size σ (see text)

Comparison of OpenAI ES and Canonical ES

Algorithm 1: OpenAI ES

Input:
$optimizer$ - Optimizer function
σ - Mutation step-size
λ - Population size
θ_0 - Initial policy parameters
F - Policy evaluation function

mirrored sampling to reduce the variance of estimate

for t = 0, 1, ... **do**
    for i = 1, 2, ... $\frac{\lambda}{2}$ **do**
        Sample noise vector: $\epsilon_i \sim \mathcal{N}(0, I)$
        Evaluate score in the game: $s_i^+ \leftarrow F(\theta_t + \sigma * \epsilon_i)$
        Evaluate score in the game: $s_i^- \leftarrow F(\theta_t - \sigma * \epsilon_i)$
    Compute normalized ranks: $r = ranks(s), r_i \in [0, 1)$
    Estimate gradient: $g \leftarrow \frac{1}{\sigma * \lambda} \sum_{i=1}^{\lambda}(r_i * \epsilon_i)$
    Update policy network: $\theta_{t+1} \leftarrow \theta_t + optimizer(g)$

Algorithm 2: Canonical ES Algorithm

Input:
σ - Mutation step-size
θ_0 - Initial policy parameters
F - Policy evaluation function
λ - Offspring population size
μ - Parent population size

Initialize :
$w_i = \frac{\log(\mu+0.5)-\log(i)}{\sum_{j=1}^{\mu}\log(\mu+0.5)-\log(j)}$

for $t = 0, 1, ...$ **do**
    for $i = 1...\lambda$ **do**
        Sample noise: $\epsilon_i \sim \mathcal{N}(0, I)$
        Evaluate score in the game: $s_i \leftarrow F(\theta_t + \sigma * \epsilon_i)$
    Sort $(\epsilon_1, \ldots, \epsilon_\lambda)$ according to s (ϵ_i with best s_i first)
    Update policy: $\theta_{t+1} \leftarrow \theta_t + \sigma * \sum_{j=1}^{\mu} w_j * \epsilon_j$
    Optionally, update step size σ (see text)

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540, 2016*

Comparison of OpenAI ES and Canonical ES

Algorithm 1: OpenAI ES

Input:
$optimizer$ - Optimizer function
σ - Mutation step-size
λ - Population size
θ_0 - Initial policy parameters
F - Policy evaluation function

fitness shaping

for t = 0, 1, ... **do**
  for i = 1, 2, ... $\frac{\lambda}{2}$ **do**
    Sample noise vector: $\epsilon_i \sim \mathcal{N}(0, I)$
    Evaluate score in the game: $s_i^+ \leftarrow F(\theta_t + \sigma * \epsilon_i)$
    Evaluate score in the game: $s_i^- \leftarrow F(\theta_t - \sigma * \epsilon_i)$
  Compute normalized ranks: $r = ranks(s), r_i \in [0, 1)$
  Estimate gradient: $g \leftarrow \frac{1}{\sigma * \lambda} \sum_{i=1}^{\lambda} (r_i * \epsilon_i)$
  Update policy network: $\theta_{t+1} \leftarrow \theta_t + optimizer(g)$

Algorithm 2: Canonical ES Algorithm

Input:
σ - Mutation step-size
θ_0 - Initial policy parameters
F - Policy evaluation function
λ - Offspring population size
μ - Parent population size
Initialize :
$w_i = \frac{\log(\mu+0.5)-\log(i)}{\sum_{j=1}^{\mu} \log(\mu+0.5)-\log(j)}$

weighted recombination

for $t = 0, 1, \ldots$ **do**
  for $i = 1 \ldots \lambda$ **do**
    Sample noise: $\epsilon_i \sim \mathcal{N}(0, I)$
    Evaluate score in the game: $s_i \leftarrow F(\theta_t + \sigma * \epsilon_i)$
  Sort $(\epsilon_1, \ldots, \epsilon_\lambda)$ according to s (ϵ_i with best s_i first)
  Update policy: $\theta_{t+1} \leftarrow \theta_t + \sigma * \sum_{j=1}^{\mu} w_j * \epsilon_j$
  Optionally, update step size σ (see text)

Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jurgen Schmidhuber. Natural evolution strategies. *Journal of Machine Learning Research, 15(1):949–980, 2014*

Comparison of OpenAI ES and Canonical ES

Algorithm 1: OpenAI ES

Input:
$optimizer$ - Optimizer function
σ - Mutation step-size
λ - Population size
θ_0 - Initial policy parameters
F - Policy evaluation function

modern gradient descent
(Adam or SGD with momentum)

for t = $0, 1, \ldots$ **do**
  for i = $1, 2, \ldots \frac{\lambda}{2}$ **do**
    Sample noise vector: $\epsilon_i \sim \mathcal{N}(0, I)$
    Evaluate score in the game: $s_i^{+} \leftarrow F(\theta_t + \sigma * \epsilon_i)$
    Evaluate score in the game: $s_i^{-} \leftarrow F(\theta_t - \sigma * \epsilon_i)$
  Compute normalized ranks: $r = ranks(s), r_i \in [0, 1)$
  Estimate gradient: $g \leftarrow \frac{1}{\sigma * \lambda} \sum_{i=1}^{\lambda} (r_i * \epsilon_i)$
  Update policy network: $\theta_{t+1} \leftarrow \theta_t + optimizer(g)$

Algorithm 2: Canonical ES Algorithm

Input:
σ - Mutation step-size
θ_0 - Initial policy parameters
F - Policy evaluation function
λ - Offspring population size
μ - Parent population size
Initialize :
$w_i = \frac{\log(\mu+0.5)-\log(i)}{\sum_{j=1}^{\mu} \log(\mu+0.5)-\log(j)}$

for $t = 0, 1, \ldots$ **do**
  for $i = 1 \ldots \lambda$ **do**
    Sample noise: $\epsilon_i \sim \mathcal{N}(0, I)$
    Evaluate score in the game: $s_i \leftarrow F(\theta_t + \sigma * \epsilon_i)$
  Sort $(\epsilon_1, \ldots, \epsilon_\lambda)$ according to s (ϵ_i with best s_i first)
  Update policy: $\theta_{t+1} \leftarrow \theta_t + \sigma * \sum_{j=1}^{\mu} w_j * \epsilon_j$
  Optionally, update step size σ (see text)

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization.
arXiv preprint arXiv:1412.6980, 2014

Results: trained on 800 CPUs in parallel

	OpenAI ES 1 hour	OpenAI ES (our) 1 hour	Canonical ES 1 hour	OpenAI ES (our) 5 hours	Canonical ES 5 hours
Alien	994	**3040 ± 276.8**	2679.3 ± 1477.3	4940 ± 0	**5878.7 ± 1724.7**
Alien		**1733.7 ± 493.2**	965.3 ± 229.8	3843.3 ± 228.7	**5331.3 ± 990.1**
Alien		**1522.3 ± 790.3**	885 ± 469.1	2253 ± 769.4	**4581.3 ± 299.1**
BeamRider	744	**792.3 ± 146.6**	774.5 ± 202.7	**4617.1 ± 1173.3**	1591.3 ± 575.5
BeamRider		708.3 ± 194.7	**746.9 ± 197.8**	**1305.9 ± 450.4**	965.3 ± 441.4
BeamRider		690.7 ± 87.7	**719.6 ± 197.4**	**714.3 ± 189.9**	703.5 ± 159.8
Breakout	9.5	14.3 ± 6.5	**17.5 ± 19.4**	26.1 ± 5.8	**105.7 ± 158**
Breakout		11.8 ± 3.3	**13 ± 17.1**	19.4 ± 6.6	**80 ± 143.4**
Breakout		**11.4 ± 3.6**	10.7 ± 15.1	**14.2 ± 2.7**	12.7 ± 17.7
Enduro	95	70.6 ± 17.2	**84.9 ± 22.3**	**115.4 ± 16.6**	86.6 ± 19.1
Enduro		36.4 ± 12.4	**50.5 ± 15.3**	**79.9 ± 18**	76.5 ± 17.7
Enduro		**25.3 ± 9.6**	7.6 ± 5.1	58.2 ± 10.5	**69.4 ± 32.8**
Pong	21	**21.0 ± 0.0**	12.2 ± 16.6	**21.0 ± 0.0**	**21.0 ± 0.0**
Pong		**21.0 ± 0.0**	5.6 ± 20.2	**21 ± 0**	11.2 ± 17.8
Pong		**21.0 ± 0.0**	0.3 ± 20.7	**21 ± 0**	−9.8 ± 18.6
Qbert	147.5	**8275 ± 0**	8000 ± 0	12775 ± 0	**263242 ± 433050**
Qbert		1400 ± 0	**6625 ± 0**	5075 ± 0	**16673.3 ± 6.2**
Qbert		1250 ± 0	**5850 ± 0**	4300 ± 0	**5136.7 ± 4093.9**
Seaquest	1390	1006 ± 20.1	**1306.7 ± 262.7**	1424 ± 26.5	**2849.7 ± 599.4**
Seaquest		898 ± 31.6	**1188 ± 24**	1040 ± 0	**1202.7 ± 27.2**
Seaquest		887.3 ± 20.3	**1170.7 ± 23.5**	**960 ± 0**	946.7 ± 275.1
SpaceInvaders	678.5	**1191.3 ± 84.6**	896.7 ± 123	**2326.5 ± 547.6**	2186 ± 1278.8
SpaceInvaders		**983.7 ± 158.5**	721.5 ± 115	**1889.3 ± 294.3**	1685 ± 648.6
SpaceInvaders		**845.3 ± 69.7**	571.3 ± 98.8	**1706.5 ± 118.3**	1648.3 ± 294.5

Qualitative analysis

Cons:

- In Seaquest and Enduro most of the ES runs converge to local optimum
  - Performance plateaus in both algorithms
  - Easy improvements with reward clipping (like in RL algorithms)
- Solutions not robust to the noise in the environment
  - High variance in score across different initial environment conditions

Pros:

- In Qbert, canonical ES was able to find creative solutions
  - Exploit flaw game design
  - Exploit game implementation bug
- Potential for combining with RL methods

Escaping local optimum

Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents.

Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth O Stanley, and Jeff Clune

Uber AI Labs

arXiv:1712.06560, 2017

Escaping local optimum

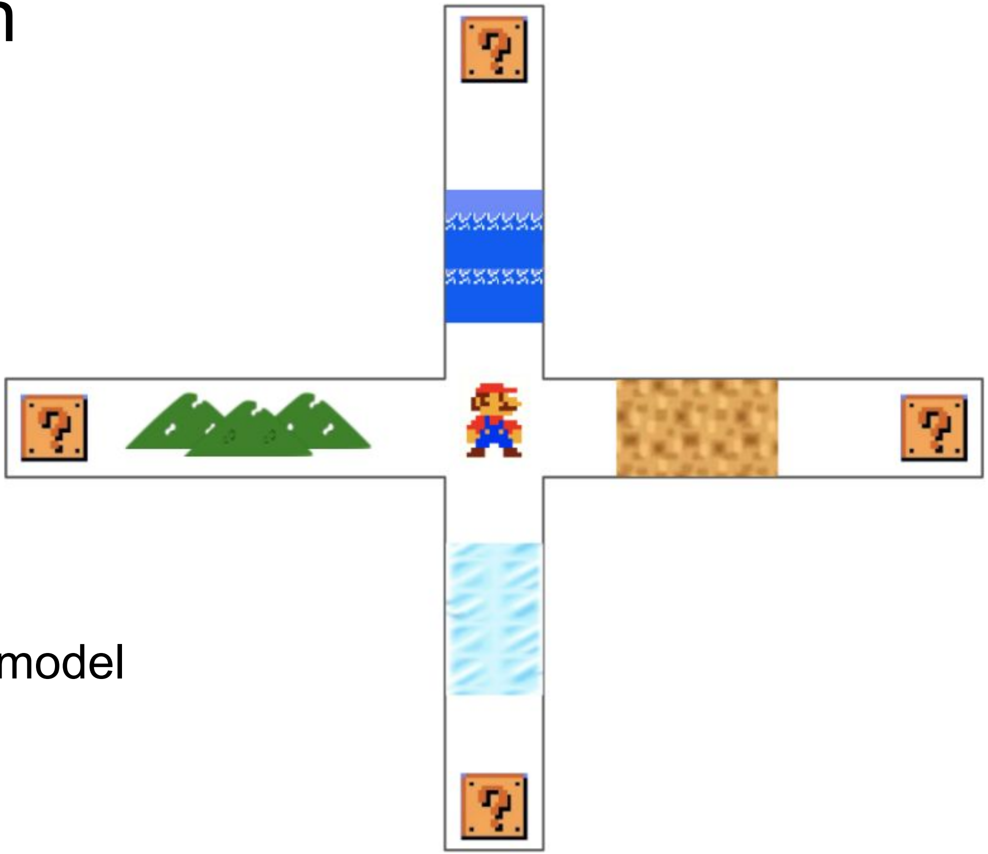
- Deceptive and sparse rewards
  - Need for directed exploration

Different methods for directed exploration:

- Based on state-action pairs
- Based on function of trajectory
  - Novelty search (exploration only)
  - Quality diversity (exploration and exploitation)

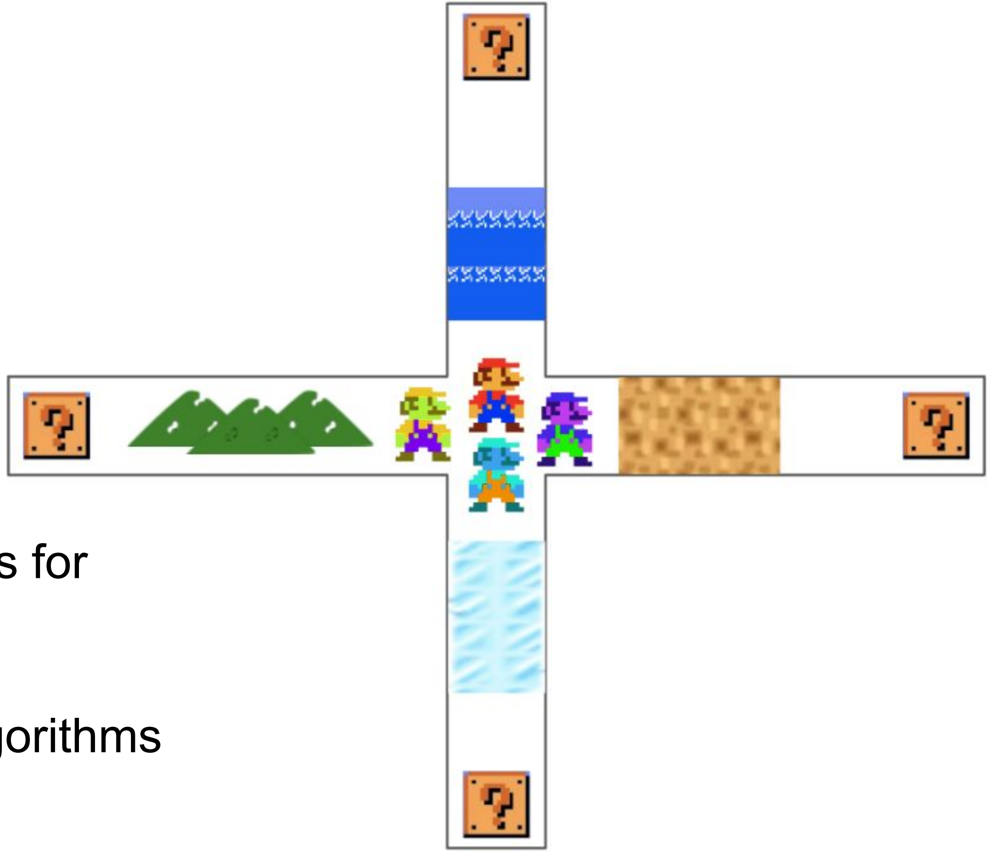
Single agent exploration

- Depth-first search
- Breadth-first search
- Problems
  - Catastrophic forgetting
  - Cognitive capacity of agent/model



Example from Stanton, Christopher and Clune, Jeff. Curiosity search: producing generalists by encouraging individuals to continually explore and acquire skills throughout their lifetime. PloS one, 2016.

Multi agent exploration



- *Meta-population* of *M* agents
- Separate agents become experts for separate tasks
- Population of specialists
  can be exploited by other ML algorithms

Example from Stanton, Christopher and Clune, Jeff. Curiosity search: producing generalists by encouraging individuals to continually explore and acquire skills throughout their lifetime. PloS one, 2016.

Novelty Search

$b(\pi_\theta)$ - *behavior characterization*

A - *archive of past* $b(\pi_\theta)$

$$N(\theta, A) = N(b(\pi_\theta), A) = \frac{1}{|S|} \sum_{j \in S} ||b(\pi_\theta) - b(\pi_j)||_2$$

$$S = kNN(b(\pi_\theta), A)$$
$$= \{b(\pi_1), b(\pi_2), ..., b(\pi_k)\}$$

NS-ES: $$\theta_{t+1}^m \leftarrow \theta_t^m + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} N(\theta_t^{i,m}, A)\epsilon_i$$

Quality diversity

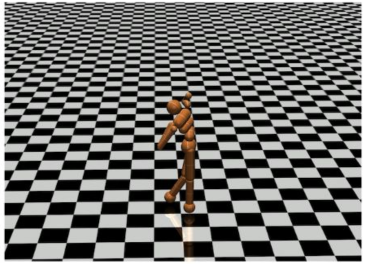
QD-ES / NSR-ES:

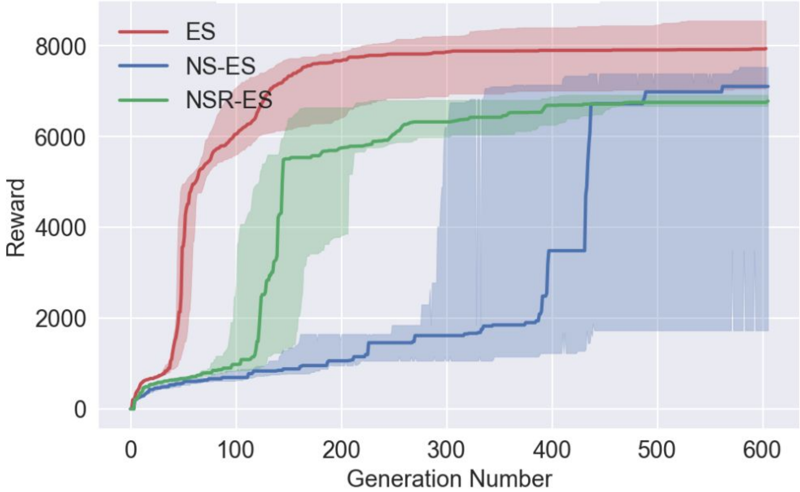
ranked

$$\theta_{t+1}^{m} \leftarrow \theta_t^m + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} \frac{f(\theta_t^{i,m}) + N(\theta_t^{i,m}, A)}{2} \epsilon_i$$

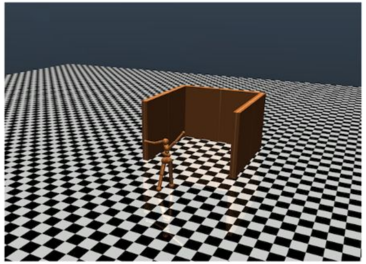
MuJoCo Humanoid-v1

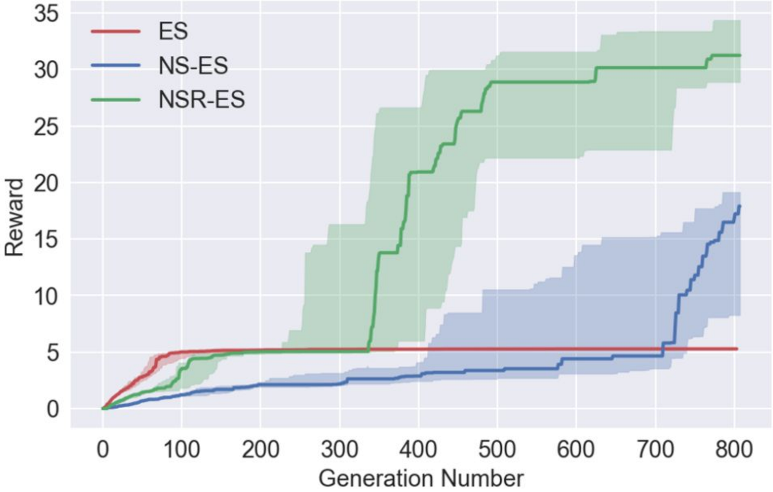
No deceptive reward



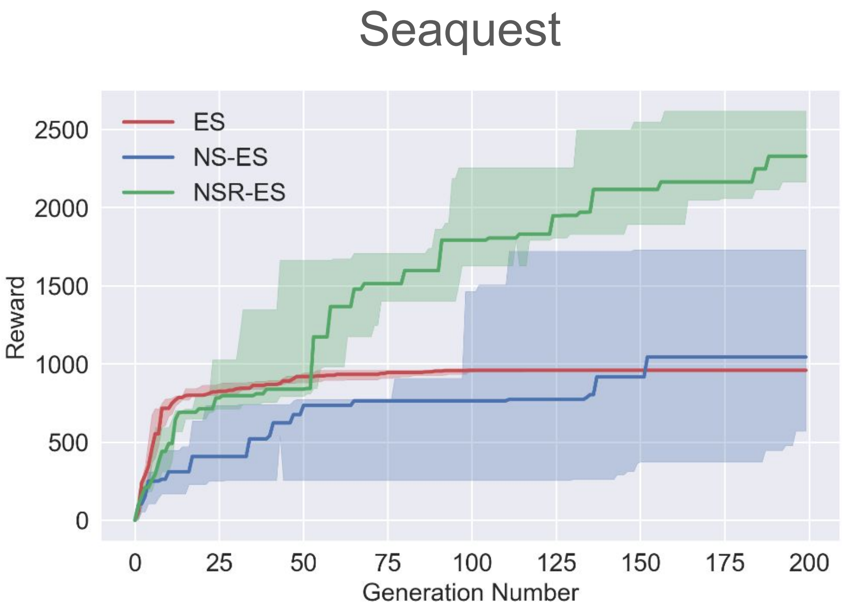


Deceptive reward

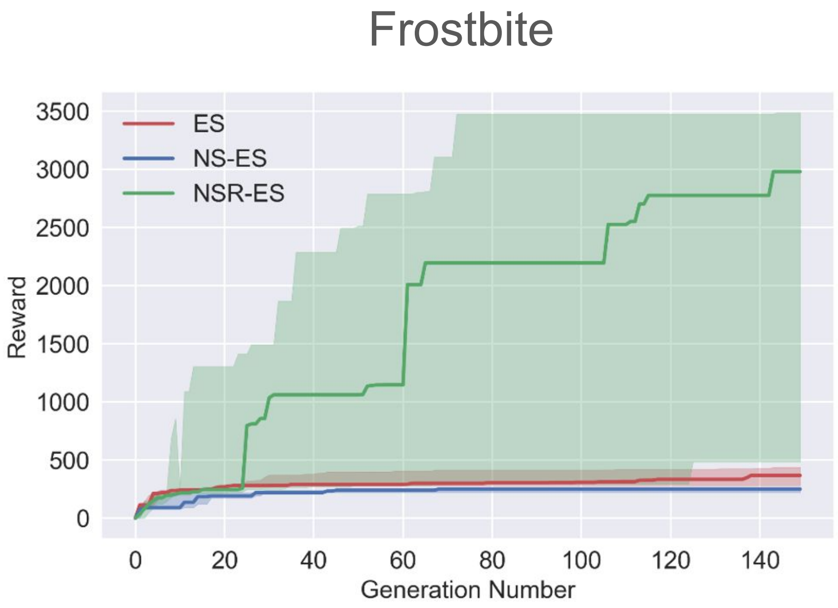




Atari



Seaquest
ES
NS-ES
NSR-ES
Reward
Generation Number
0
500
1000
1500
2000
2500
0
25
50
75
100
125
150
175
200



Frostbite
ES
NS-ES
NSR-ES
Reward
Generation Number
0
500
1000
1500
2000
2500
3000
3500
0
20
40
60
80
100
120
140

Genetic algorithms

Deep Neuroevolution: Genetic Algorithms are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning
Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, Jeff Clune
Uber AI Labs

- Uses a simple population-based genetic algorithm (GA)
- Demonstrates that GA is able to train a large neural networks
- Competitive results to reference algorithms (ES, A3C, DQN) on ATARI games

Algorithm

- Population P of N hyperparameter vectors θ (neural network weights)
- Mutation applied N-1 times to T parents

  $\theta' = \theta + \sigma \epsilon$ where $\epsilon \sim N(0, I)$
  - σ determined empirically

- Elitism applied to get N-th individual

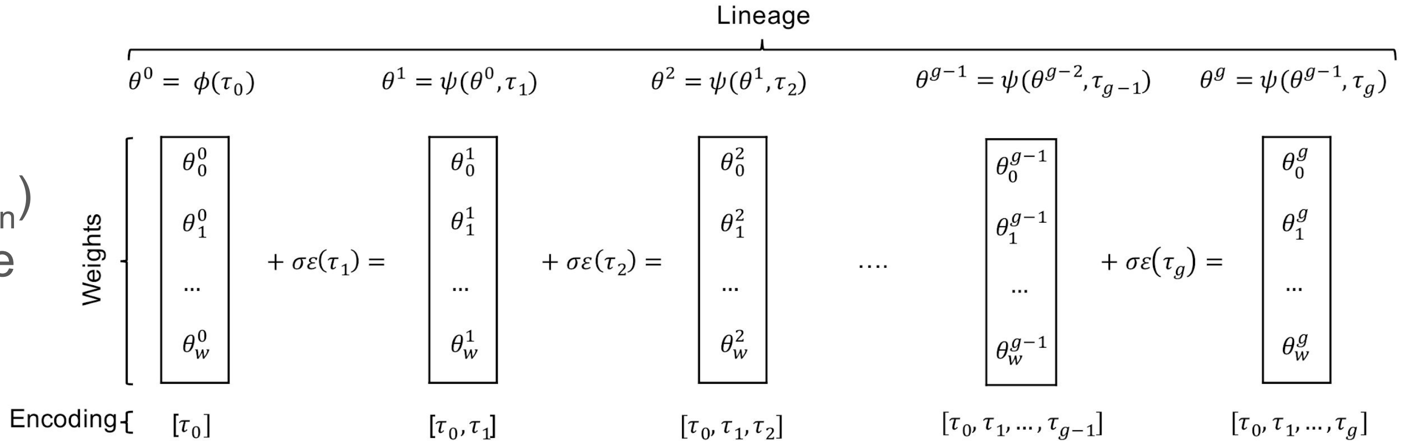
- No crossover performed
  - Can yield improvement in domains where a genomic representation is useful

Data compression

- Storing entire hyperparameter vectors of individuals scales poorly in memory
  - Communication overhead for large networks with high parallelism
- Represent vector as initialization seed and a list of seeds to generate individual
  - Size grows linearly with number of generations, independent of hyperparameter vector length

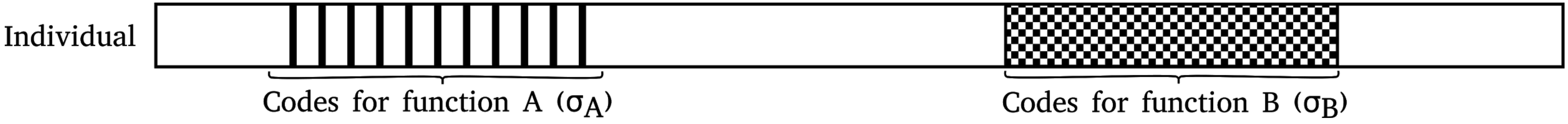
$\Psi(\theta^{n-1}, \tau_n) = \theta^{n-1} + \sigma\, \varepsilon(\tau_n)$

$\varepsilon(\tau_n)$ precomputed table



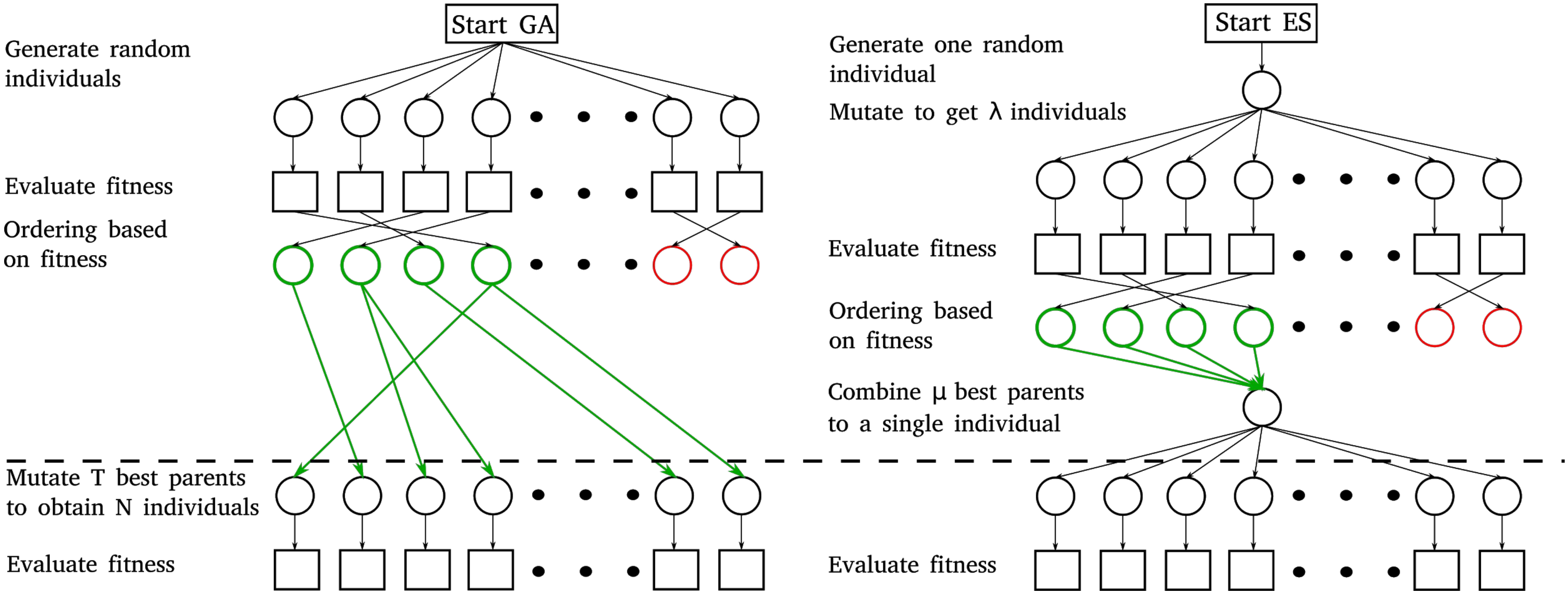
Exploit structure in hyperparameter vector

- Hyperparameter vector is often more than just bunch of numbers
  - Different components may need different values of σ



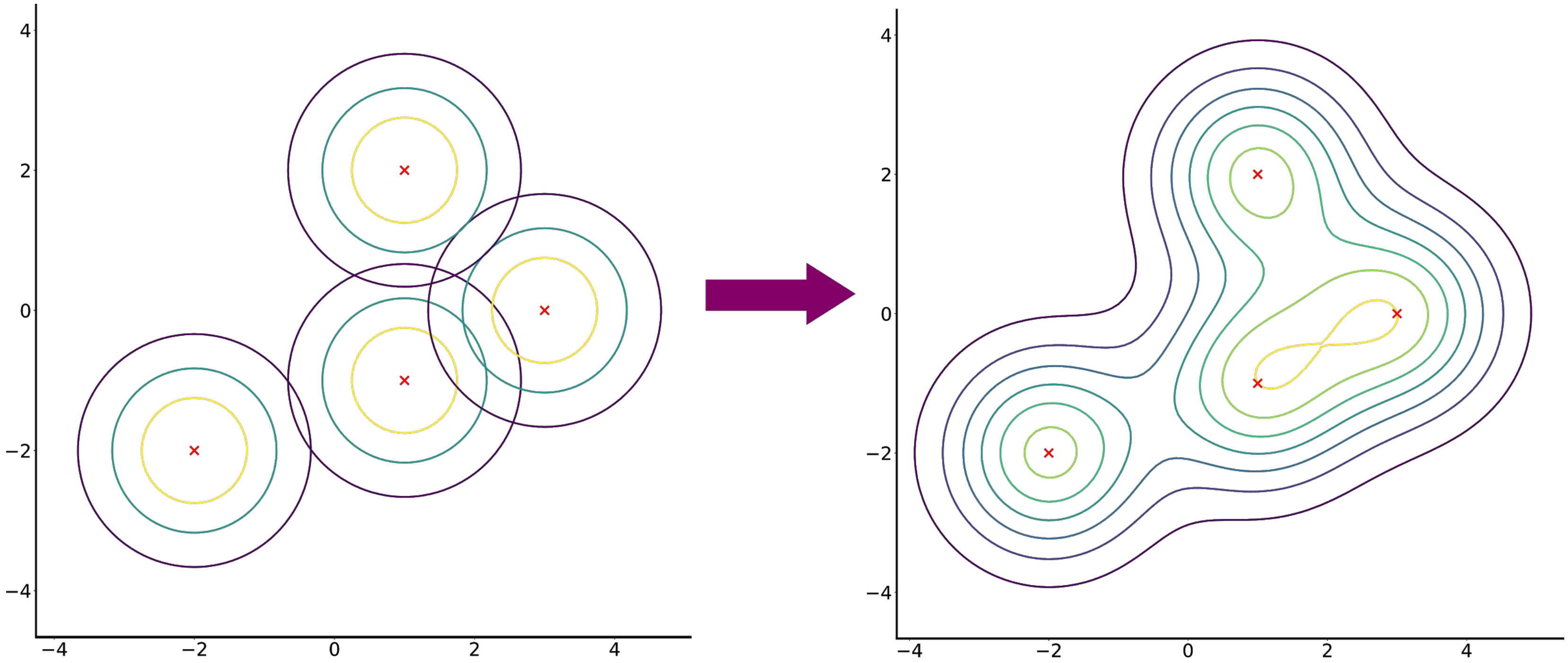
- Crossover allows efficient transfer of modular functions

Comparison between GA and ES



Comparison between GA and CE

- Parents of generation can be viewed as centers of Gaussian distribution
  - Offsprings can be viewed as samples from multimodal Gaussian distribution



Conclusion

- Simple vanilla population-based genetic algorithm
- Improvements for GA's from literature can also be included (e.g.: individual σ)
- Motivates the usage of hybrid optimization algorithms
- During progress of paper authors realize that sampling the local neighbourhood yields also good results for some domains
    - Random search

Random Search

Simple random search provides a competitive approach to reinforcement learning
Horia Mania, Aurelia Guy, Benjamin Recht
University of California, Berkeley

- Uses a simple random search algorithm to solve continuous control problems
    - Modifications to increase performance (Augmented Random Search ARS)

- Uses linear policies to solve MuJoCo locomotion tasks
- Demonstrate high robustness to optimizer parameter choices
    - Relevant for practical applications?

Algorithm

- Sample N random directions
- Evaluate fitness for steps ν and -ν along directions (2*N evaluations)
- Weight directions with fitness difference and linearly recombine them

Improvements:

- Scale update-step by standard deviation of collected rewards (**ARS V1**)
- State normalization (similar to whitening) (**ARS V2**)
- Discard directions which have low rewards (**ARS V1-t** / **ARS V2-t**)

Differences between ARS and ES

Algorithm 1: OpenAI ES

Input:
$optimizer$ - Optimizer function
σ - Mutation step-size
λ - Population size
θ_0 - Initial policy parameters
F - Policy evaluation function

for $t = 0, 1, \ldots$ **do**
  for $i = 1, 2, \ldots \frac{\lambda}{2}$ **do**
    Sample noise vector: $\epsilon_i \sim \mathcal{N}(0, I)$
    Evaluate score in the game: $s_i^+ \leftarrow F(\theta_t + \sigma * \epsilon_i)$
    Evaluate score in the game: $s_i^- \leftarrow F(\theta_t - \sigma * \epsilon_i)$
  Compute normalized ranks: $r = ranks(s), r_i \in [0, 1)$
  Estimate gradient: $g \leftarrow \frac{1}{\sigma * \lambda} \sum_{i=1}^{\lambda} (r_i * \epsilon_i)$
  Update policy network: $\theta_{t+1} \leftarrow \theta_t + optimizer(g)$

- No additional optimizer
- No ranking mechanism
- No virtual batch normalization

Algorithm 2: Canonical ES Algorithm

Input:
σ - Mutation step-size
θ_0 - Initial policy parameters
F - Policy evaluation function
λ - Offspring population size
μ - Parent population size
Initialize :

$$w_i = \frac{\log(\mu + 0.5) - \log(i)}{\sum_{j=1}^{\mu} \log(\mu + 0.5) - \log(j)}$$

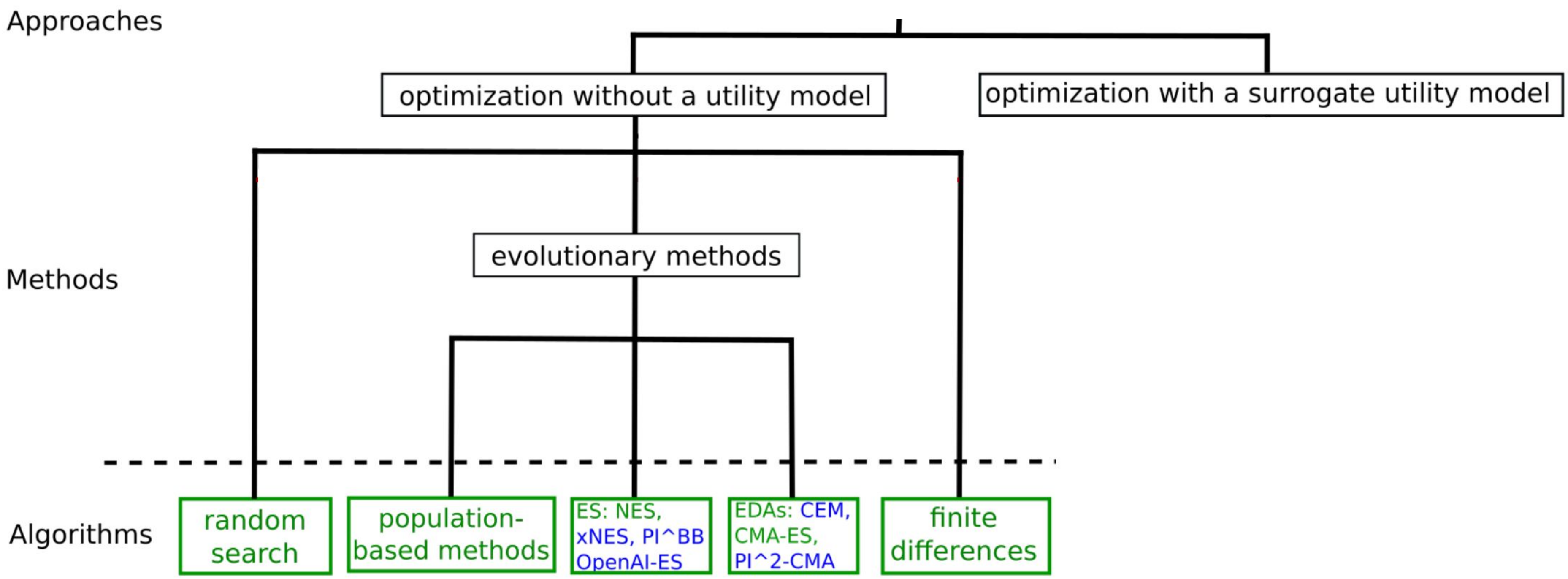
for $t = 0, 1, \ldots$ **do**
  for $i = 1 \ldots \lambda$ **do**
    Sample noise: $\epsilon_i \sim \mathcal{N}(0, I)$
    Evaluate score in the game: $s_i \leftarrow F(\theta_t + \sigma * \epsilon_i)$
  Sort $(\epsilon_1, \ldots, \epsilon_\lambda)$ according to s (ϵ_i with best s_i first)
  Update policy: $\theta_{t+1} \leftarrow \theta_t + \sigma * \sum_{j=1}^{\mu} w_j * \epsilon_j$
  Optionally, update step size σ (see text)

- No virtual batch normalization

Conclusion

- Simple random search algorithm yields competitive results on some domains
  - Robust to optimizer parameter choices
- Linear policy might not be sufficient for all domains
  - They show that linear policies can solve MuJoCo locomotion tasks
- Can be compared to ES with mirror sampling

Summary



Policy Search in Continuous Action Domains: an Overview
Olivier Sigaud, Freek Stulp

Questions?

GA Algorithm

Algorithm 1 Simple Genetic Algorithm

Input: mutation function ψ, population size N, number of selected individuals T, policy initialization routine ϕ, fitness function F.

for $g = 1, 2..., G$ generations **do**
    for $i = 1, ..., N - 1$ in next generation's population **do**
        if $g = 1$ **then**
            $\mathcal{P}_i^{g=1} = \phi(\mathcal{N}(0, I))$ {initialize random DNN}
        else
            $k = \text{uniformRandom}(1, T)$ {select parent}
            $\mathcal{P}_i^g = \psi(\mathcal{P}_k^{g-1})$ {mutate parent}
        end if
        Evaluate $F_i = F(\mathcal{P}_i^g)$
    end for
    Sort $\mathcal{P}_i^g$ with descending order by F_i
    if $g = 1$ **then**
        Set Elite Candidates $C \leftarrow \mathcal{P}_{1...10}^{g=1}$
    else
        Set Elite Candidates $C \leftarrow \mathcal{P}_{1...9}^{g} \cup \{\text{Elite}\}$
    end if
    Set Elite $\leftarrow \arg\max_{\theta \in C} \frac{1}{30} \sum_{j=1}^{30} F(\theta)$
    $\mathcal{P}^g \leftarrow [\text{Elite}, \mathcal{P}^g - \{\text{Elite}\}]$ {only include elite once}
end for
Return: Elite

Basic Random Search (BRS) as starting point

Algorithm 1 Basic Random Search (BRS)

Hyperparameters: step-size α, number of directions sampled per iteration N, standard deviation of the exploration noise ν

Initialize: $\theta_0 = \mathbf{0}$, and $j = 0$.

while ending condition not satisfied **do**

  Sample $\delta_1, \delta_2, \ldots, \delta_N$ of the same size as θ_j, with i.i.d. standard normal entries.

  Collect $2N$ rollouts of horizon H and their corresponding rewards using the policies

$$\pi_{j,k,+}(x) = \pi_{\theta_j + \nu\delta_k}(x) \quad \text{and} \quad \pi_{j,k,-}(x) = \pi_{\theta_j - \nu\delta_k}(x),$$

  with $k \in \{1, 2, \ldots, N\}$.

  Make the update step:

$$\theta_{j+1} = \theta_j + \tfrac{\alpha}{N} \sum_{k=1}^{N} \left[r(\pi_{j,k,+}) - r(\pi_{j,k,-}) \right] \delta_k \, .$$

  $j \leftarrow j + 1$.

end while

.

Variants of BRS

- Modifications to increase performance of BRS
  - Four different versions grouped under: Augmented Random Search (ARS)

- Scale update-step by variance of collected rewards (**ARS V1**)
- Apply state rescaling (similar to whitening) (**ARS V2**)
  - Crucial to solve the Humanoid locomotion task

- Discard perturbations which have low rewards compared to others (**ARS V1-t** / **ARS V2-t**)
  - (**ARS V1** / **ARS V2**) Limit where all perturbations are combined

ARS V1

n … state space dimensionality
p … action space dimensionality

- V1: BRS + scaling of update step
- Variation of reward increases over the course of training

- Circumvents issue of finding a suitable α or a schedule for it

- ES addresses this issue by ranking followed by an adaptive optimization algorithm

---

Algorithm 2 Augmented Random Search (ARS): four versions **V1**, **V1-t**, **V2** and **V2-t**

1: **Hyperparameters:** step-size α, number of directions sampled per iteration N, standard deviation of the exploration noise ν, number of top-performing directions to use b ($b < N$ is allowed only for **V1-t** and **V2-t**)

2: **Initialize:** $M_0 = \mathbf{0} \in \mathbb{R}^{p\times n}$, $\mu_0 = \mathbf{0} \in \mathbb{R}^n$, and $\Sigma_0 = \mathbf{I}_n \in \mathbb{R}^{n\times n}$, $j = 0$.

3: **while** ending condition not satisfied **do**

4: Sample $\delta_1, \delta_2, \dots, \delta_N$ in $\mathbb{R}^{p\times n}$ with i.i.d. standard normal entries.

5: Collect $2N$ rollouts of horizon H and their corresponding rewards using the $2N$ policies

$$\mathbf{V1:} \begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k)x \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k)x \end{cases}$$

$$\mathbf{V2:} \begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k)\,\mathrm{diag}\,(\Sigma_j)^{-1/2}\,(x - \mu_j) \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k)\,\mathrm{diag}(\Sigma_j)^{-1/2}(x - \mu_j) \end{cases}$$

for $k \in \{1, 2, \dots, N\}$.

6: Sort the directions δ_k by $\max\{r(\pi_{j,k,+}), r(\pi_{j,k,-})\}$, denote by $\delta_{(k)}$ the k-th largest direction, and by $\pi_{j,(k),+}$ and $\pi_{j,(k),-}$ the corresponding policies.

7: Make the update step: Scaling update step by standard deviation of collected results

$$M_{j+1} = M_j + \frac{\alpha}{b\sigma_R} \sum_{k=1}^{b} \left[r(\pi_{j,(k),+}) - r(\pi_{j,(k),-})\right]\delta_{(k)},$$

where σ_R is the standard deviation of the $2b$ rewards used in the update step.

8: **V2** : Set μ_{j+1}, Σ_{j+1} to be the mean and covariance of the $2NH(j+1)$ states encountered from the start of training.[2]

9: $j \leftarrow j + 1$

10: **end while**

.

ARS V2

n … state space dimensionality
p … action space dimensionality

- V2: BRS + modified states
- Similar to whitening in regression
  - Put equal weight on different components of the state
- Mean and Covariance computed over all states encountered so far
- Without this trick, Humanoid locomotion task is unsolvable
- Similar normalization also done by ES

Algorithm 2 Augmented Random Search (ARS): four versions **V1**, **V1-t**, **V2** and **V2-t**

1: **Hyperparameters:** step-size α, number of directions sampled per iteration N, standard deviation of the exploration noise ν, number of top-performing directions to use b ($b < N$ is allowed only for **V1-t** and **V2-t**)

2: **Initialize:** $M_0 = \mathbf{0} \in \mathbb{R}^{p \times n}$, $\mu_0 = \mathbf{0} \in \mathbb{R}^n$, and $\Sigma_0 = \mathbf{I}_n \in \mathbb{R}^{n \times n}$, $j = 0$.

3: **while** ending condition not satisfied **do**

4: Sample $\delta_1, \delta_2, \ldots, \delta_N$ in $\mathbb{R}^{p \times n}$ with i.i.d. standard normal entries.

5: Collect $2N$ rollouts of horizon H and their corresponding rewards using the $2N$ policies

$$\textbf{V1:} \begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k)x \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k)x \end{cases}$$

$$\textbf{V2:} \begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k)\operatorname{diag}(\Sigma_j)^{-1/2}(x - \mu_j) \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k)\operatorname{diag}(\Sigma_j)^{-1/2}(x - \mu_j) \end{cases}$$

States normalized by mean μ and standard deviation Σ

for $k \in \{1, 2, \ldots, N\}$.

6: Sort the directions δ_k by $\max\{r(\pi_{j,k,+}), r(\pi_{j,k,-})\}$, denote by $\delta_{(k)}$ the k-th largest direction, and by $\pi_{j,(k),+}$ and $\pi_{j,(k),-}$ the corresponding policies.

7: Make the update step:

$$M_{j+1} = M_j + \frac{\alpha}{b\sigma_R} \sum_{k=1}^{b} \left[r(\pi_{j,(k),+}) - r(\pi_{j,(k),-}) \right] \delta_{(k)},$$

where σ_R is the standard deviation of the $2b$ rewards used in the update step.

8: **V2** : Set μ_{j+1}, Σ_{j+1} to be the mean and covariance of the $2NH(j+1)$ states encountered from the start of training.[2]

9: $j \leftarrow j + 1$

10: **end while**

ARS V1-t + V2-t

n … state space dimensionality
p … action space dimensionality

- V1-t (V2-t): V1 (V2-t) + drop of perturbations with least improvement

- Discard perturbations if rewards are small
  - Average over directions with higher Reward

- Additional optimizer parameter

- When b = N, V1 (V2) are obtained

Algorithm 2 Augmented Random Search (ARS): four versions **V1**, **V1-t**, **V2** and **V2-t**

1: **Hyperparameters:** step-size α, number of directions sampled per iteration N, standard deviation of the exploration noise ν, number of top-performing directions to use b ($b < N$ is allowed only for **V1-t** and **V2-t**)

2: **Initialize:** $M_0 = \mathbf{0} \in \mathbb{R}^{p \times n}$, $\mu_0 = \mathbf{0} \in \mathbb{R}^n$, and $\Sigma_0 = \mathbf{I}_n \in \mathbb{R}^{n \times n}$, $j = 0$.

3: **while** ending condition not satisfied **do**

4: Sample $\delta_1, \delta_2, \ldots, \delta_N$ in $\mathbb{R}^{p \times n}$ with i.i.d. standard normal entries.

5: Collect $2N$ rollouts of horizon H and their corresponding rewards using the $2N$ policies

$$\textbf{V1:} \quad \begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k)x \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k)x \end{cases}$$

$$\textbf{V2:} \quad \begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k)\,\mathrm{diag}\,(\Sigma_j)^{-1/2}\,(x - \mu_j) \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k)\,\mathrm{diag}(\Sigma_j)^{-1/2}(x - \mu_j) \end{cases}$$

for $k \in \{1, 2, \ldots, N\}$.

6: Sort the directions δ_k by $\max\{r(\pi_{j,k,+}), r(\pi_{j,k,-})\}$, denote by $\delta_{(k)}$ the k-th largest direction, and by $\pi_{j,(k),+}$ and $\pi_{j,(k),-}$ the corresponding policies.

7: Make the update step:

Drop least performing perturbations

$$M_{j+1} = M_j + \frac{\alpha}{b\sigma_R} \sum_{k=1}^{b} \left[r(\pi_{j,(k),+}) - r(\pi_{j,(k),-}) \right] \delta_{(k)},$$

where σ_R is the standard deviation of the $2b$ rewards used in the update step.

8: **V2** : Set μ_{j+1}, Σ_{j+1} to be the mean and covariance of the $2NH(j+1)$ states encountered from the start of training.[2]

9: $j \leftarrow j + 1$

10: **end while**

.

Summary

ES
(Salimans et al. 2017)

- Unimodal distribution sampling
- Uses Adam
- Hard to overcome local optima

Canonical ES
(Chrabaszcz et al. 2018)

- Unimodal distribution sampling
- Neglects suboptimal perturbations
- Hard to overcome local optima

GA
(Petroski Such et al. 2018)

Pros:

- Multimodal individual distribution
- Few HP’s
- Highly parallelizable
- High data compression
- High potential for improvements

Cons:

ARS (Mania et al. 2018)

Pros:

- Unimodal
- Simple algorithm
- Low computational complexity
- Data compression
- Neglects suboptimal perturbations

Cons:

- Proposed for linear policies
- Hard to overcome local optima

ARS result

		Maximum average reward after # timesteps				
Task	**# timesteps**	**ARS**	**PPO**	**A2C**	**CEM**	**TRPO**
Swimmer-v1	10^6	361	≈ 110	≈ 30	≈ 0	≈ 120
Hopper-v1	10^6	3047	≈ 2300	≈ 900	≈ 500	≈ 2000
HalfCheetah-v1	10^6	2345	≈ 1900	≈ 1000	≈ -400	≈ 0
Walker2d-v1	10^6	894	≈ 3500	≈ 900	≈ 800	≈ 1000

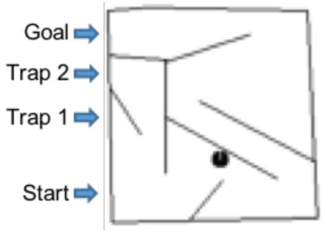
ARS result continued

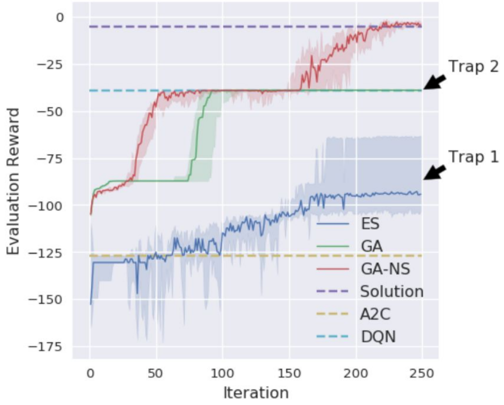
		Average # timesteps to hit Th.		
Task	**Threshold**	**ARS**	**ES**	**TRPO**
Swimmer-v1	128.25	$6.00 \cdot 10^4$	$1.39 \cdot 10^6$	$4.59 \cdot 10^6$
Hopper-v1	3403.46	$2.00 \cdot 10^6$	$3.16 \cdot 10^7$	$4.56 \cdot 10^6$
HalfCheetah-v1	2385.79	$5.86 \cdot 10^5$	$2.88 \cdot 10^6$	$5.00 \cdot 10^6$
Walker2d-v1	3830.03	$8.14 \cdot 10^6$	$3.79 \cdot 10^7$	$4.81 \cdot 10^6$

Experiments GA

- ATARI games
  - Experimental setup similar to Salimans et. al 2017
  - Data preprocessing, Network architecture, Environments same as in Mnih et. al 2015
  - Constant number of frames over GA run for comparison

- Image Hard Maze
  - Deceptive task with many local optima (“Traps”)
  - Novelty search used: reward behavior never seen before





GA results

	DQN	ES	A3C	RS	GA	GA
Frames	200M	1B	1B	1B	1B	6B
Time	∼7-10d	∼ 1h	∼ 4d	∼ 1h or 4h	∼ 1h or 4h	∼ 6h or 24h
Forward Passes	450M	250M	250M	250M	250M	1.5B
Backward Passes	400M	0	250M	0	0	0
Operations	1.25B U	250M U	1B U	250M U	250M U	1.5B U
amidar	**978**	112	264	143	263	377
assault	4,280	1,674	**5,475**	649	714	814
asterix	4,359	1,440	**22,140**	1,197	1,850	2,255
asteroids	1,365	1,562	**4,475**	1,307	1,661	2,700
atlantis	279,987	**1,267,410**	911,091	26,371	76,273	129,167
enduro	**729**	95	-82	36	60	80
frostbite	797	370	191	1,164	**4,536**	**6,220**
gravitar	473	**805**	304	431	476	764
kangaroo	7,259	**11,200**	94	1,099	3,790	**11,254**
seaquest	**5,861**	1,390	2,355	503	798	850
skiing	-13,062	-15,443	-10,911	-7,679	† **-6,502**	† **-5,541**
venture	163	760	23	488	**969**	† **1,422**
zaxxon	5,363	6,380	**24,622**	2,538	6,180	7,864

GA results continued

	DQN	ES	A3C	RS 1B	GA 1B	GA 6B
DQN		6	6	3	6	7
ES	7		7	3	6	8
A3C	7	6		6	6	7
RS 1B	10	10	7		13	13
GA 1B	7	7	7	0		13
GA 6B	6	5	6	0	0	

Parallelizability

- Requires only communication of fitnesses, and can thus scale w.r.t parameter vector size
- Perturbations are pre-generated and randomly sampled from for efficient generation of individuals