Hypothesis Driven Coordinate Ascent for Reinforcement Learning

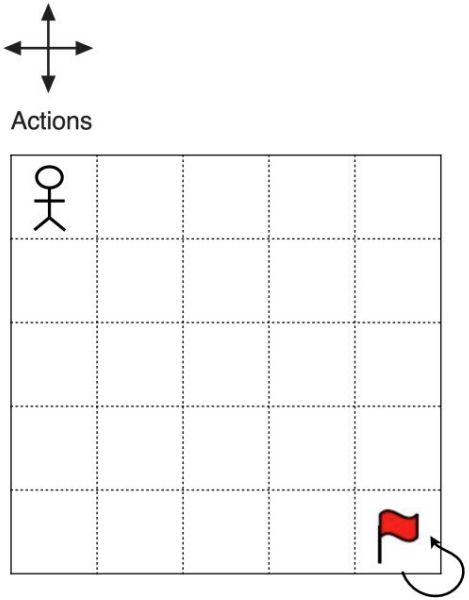
Blaine Hill
March 30, 2023

Contents

Markov Decision Process (MDP)

- Defined by: (***S***, ***A***, $\mathbb{R}$, $\mathbb{P}$, γ)
  - ***S*** : set of possible states
  - ***A*** : set of possible actions
  - $\mathbb{R}$: distribution of reward given a (**s**tate, **a**ction) pair
  - $\mathbb{P}$: transition probabilities (the distribution over next state **s'** given (**s,a**) pair)
  - γ : discount factor
- RL reduces to then searching for a policy π which gives a (deterministic or sampled) action to take given current state
- Goal: find the optimal policy π^* for a given MDP that maximizes the cumulative discounted reward $\sum_{t=0}^{\infty} \gamma^t r_t$

Example: Gridworld

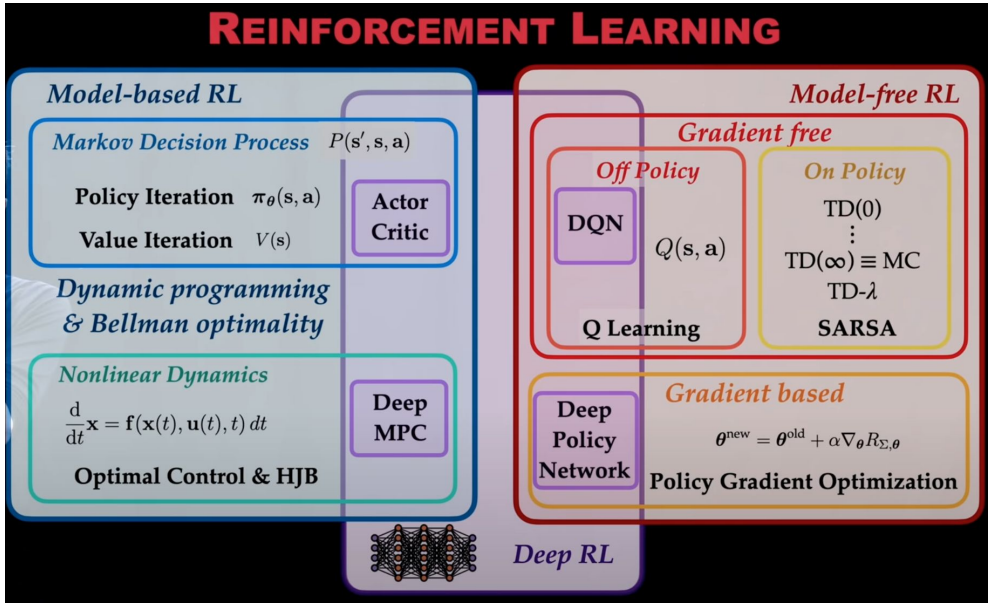


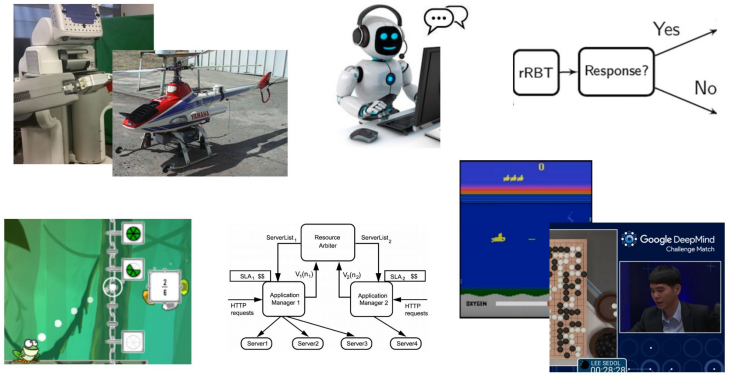
"Absorbing state": emulate the termination of a task in infinite-horizon MDPs

- State: grid No. (or integer x, y coordinates)
- Action: N, S, E, W
- Dynamics:
  - Deterministic transitions to the adjacent grid in the direction of action in most cases.
  - Keep in the current state if moving towards wall or having reached goal
- Reward: 0 in the goal state and -1 everywhere else
- Discount factor γ: 0.99

Slide Credit: Nan Jiang

Reinforcement Learning

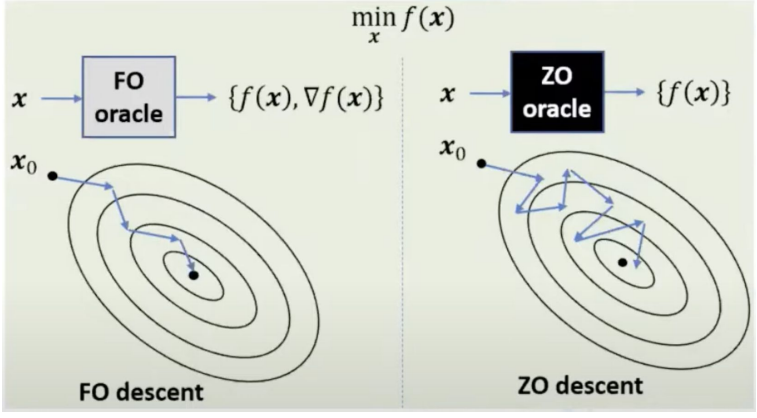




[Levine et al'16] [Ng et al'03] [Singh et al'02] [Lei et al'12]
[Mandel et al'16] [Tesauro et al'07] [Mnih et al'15][Silver et al'16]

Black Box Optimization (BBO) / Zeroth Order Optimization

- What if the model dynamics ($\mathbb{R}$, $\mathbb{P}$) are unknown? Take samples instead!
  - We need to rewrite the problem: $\max_{\theta} \mathbb{E}[F(\pi_\theta)]$
    - F(·) represents the stochastic rewards received from the MDP (can be thought of as an objective function of a non-linear program)
    - θ parametrizes π to give us π_θ (θ and π_θ are sometimes denoted as x)
- Lack of derivatives! Non-smooth environments! What to do?
  - Since there are no guarantees, we must not assume we can naively backpropagate to optimize θ
  - Take Monte Carlo Gradient Descent
    - $$x_{t+1} = x_t - \eta \cdot \frac{1}{n} \sum_{i=1}^{n} [f(x_t + \epsilon_i) - f(x_t)]$$



Evolutionary Strategies (ES)

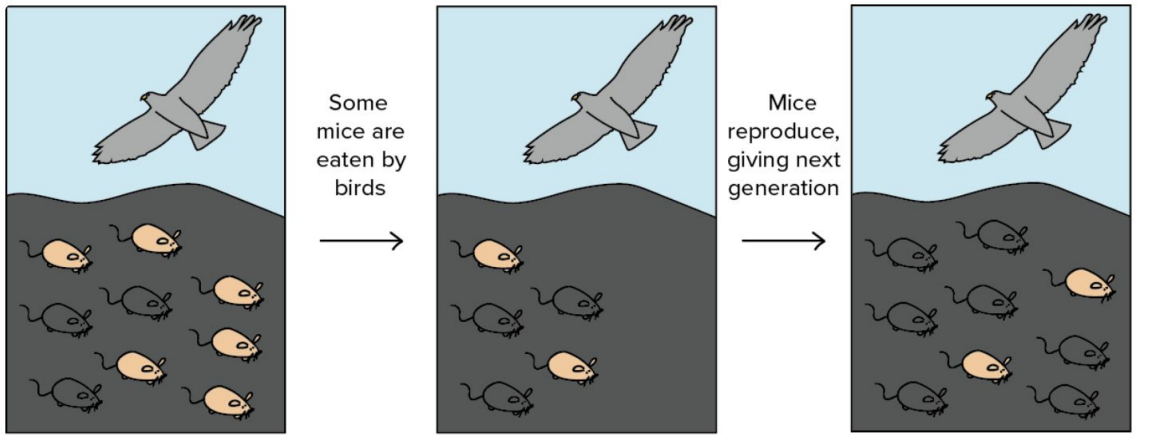


Fig. 1. How natural selection works. (Image source: Khan Academy: Darwin, evolution, & natural selection)

ES Cont.

- First, rewrite the objective as: $\mathbb{E}_{\theta \sim p_\psi} F(\theta) = \mathbb{E}_{\epsilon \sim N(0,I)} F(\theta + \sigma\epsilon)$
  - Where p_ψ is an isotropic multivariate Gaussian with mean ψ and covariance $\sigma^2 I$
- Then, we can write our score function estimator:
  - $\nabla_\theta \mathbb{E}_{\epsilon \sim N(0,I)} F(\theta + \sigma\epsilon) = \frac{1}{\sigma} \mathbb{E}_{\epsilon \sim N(0,I)} [F(\theta + \sigma\epsilon)\epsilon]$
- Analysis of variance between policy gradient (PG) and ES
  - $$\mathrm{Var}[\nabla_\theta F_{PG}(\theta)] \approx \mathrm{Var}[R(\mathbf{a})] \, \mathrm{Var}[\nabla_\theta \log p(\mathbf{a}; \theta)]$$
    $$\mathrm{Var}[\nabla_\theta F_{ES}(\theta)] \approx \mathrm{Var}[R(\mathbf{a})] \, \mathrm{Var}[\nabla_\theta \log p(\tilde{\theta}; \theta)]$$

ES Cont.

Algorithm 1 Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
for $t = 0, 1, 2, \ldots$ **do**
  Sample $\epsilon_1, \ldots \epsilon_n \sim \mathcal{N}(0, I)$
  Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
  Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \epsilon_i$
end for

Algorithm 2 Parallelized Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
Initialize: n workers with known random seeds, and initial parameters θ_0
for $t = 0, 1, 2, \ldots$ **do**
  for each worker $i = 1, \ldots, n$ **do**
    Sample $\epsilon_i \sim \mathcal{N}(0, I)$
    Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$
  end for
  Send all scalar returns F_i from each worker to every other worker
  for each worker $i = 1, \ldots, n$ **do**
    Reconstruct all perturbations ϵ_j for $j = 1, \ldots, n$ using known random seeds
    Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{j=1}^{n} F_j \epsilon_j$
  end for
end for

Covariance Matrix Adaptation ES (CMA-ES)

- There's a slight problem with ES: $\boldsymbol{\sigma}$ is constant
  - In some implementations, $\boldsymbol{\sigma}$ is parametrized by time ($\boldsymbol{\sigma}_i$) and not constant but highly correlated
  - Because $\boldsymbol{\sigma}$ controls exploration (how much your policy is perturbed), ES can't adjust
- CMA-ES can by tracking pairwise dependencies between samples in a matrix
  - Previously: $\theta = (\mu, \sigma),\ p_\theta(x) \sim \mathcal{N}(\mu, \sigma^2 I) = \mu + \sigma\mathcal{N}(0, I)$
  - Now: $\theta = (\mu, \sigma, C),\ p_\theta(x) \sim \mathcal{N}(\mu, \sigma^2 C) \sim \mu + \sigma\mathcal{N}(0, C)$

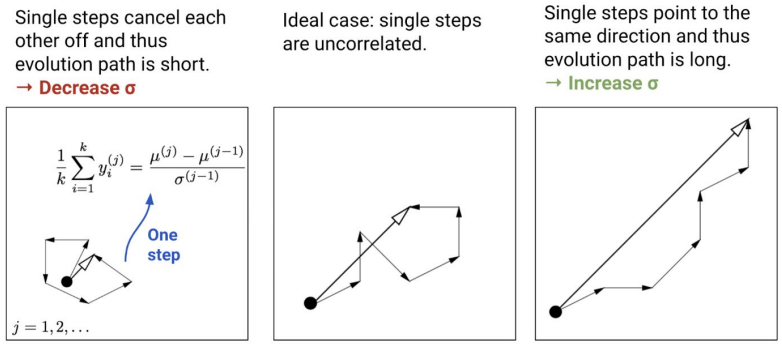


Fig. 2. Three scenarios of how single steps are correlated in different ways and their impacts on step size update. (Image source: additional annotations on Fig 5 in CMA-ES tutorial paper)

CMA-ES Cont.

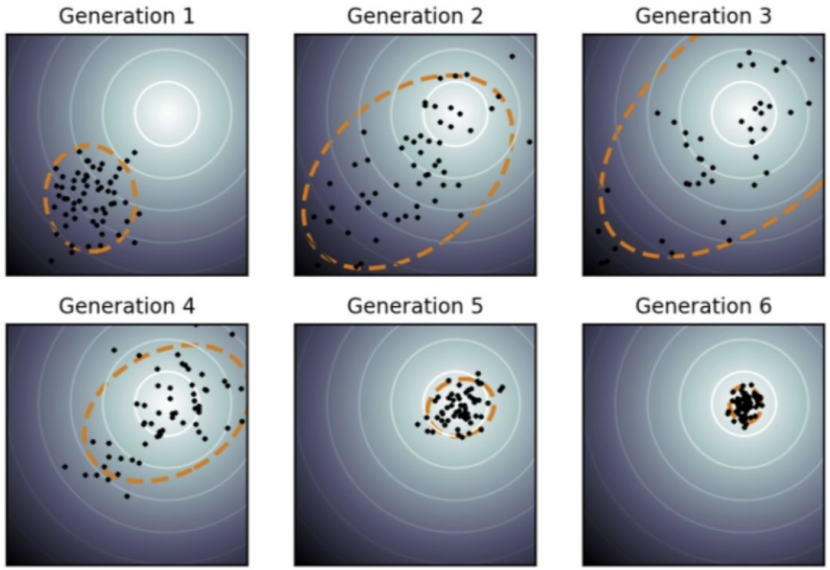


Fig. 3. Illustration of how CMA-ES works on a 2D optimization problem (the lighter color the better). Black dots are samples in one generation. The samples are more spread out initially but when the model has higher confidence in finding a good solution in the late stage, the samples become very concentrated over the global optimum. (Image source: Wikipedia CMA-ES)

Augmented Random Search (ARS)

Algorithm 1 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

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

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

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

6: **V1-t, V2-t:** 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.[1]

9: $j \leftarrow j + 1$

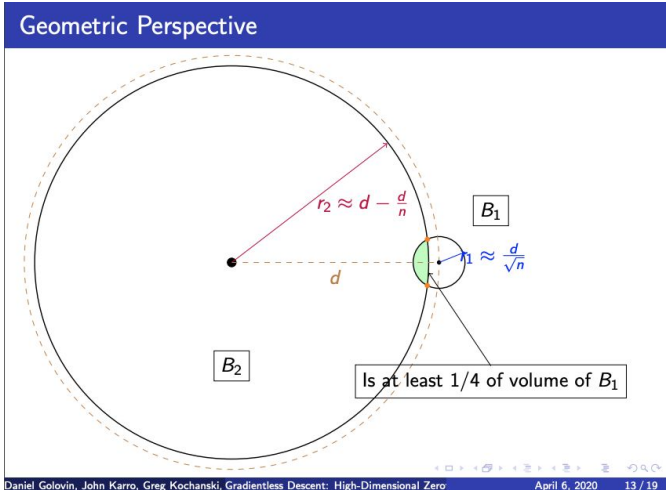
10: **end while**

GradientLess Descent (GLD)

Algorithm 1: Gradientless Descent with Binary Search (GLD-Search)

Input: function: $f : \mathbb{R}^n \to \mathbb{R}$, $T \in \mathbb{Z}_+$: number of iterations, x_0: starting point,
$\mathcal{D}$: sampling distribution, R: maximum search radius, r: minimum search radius

Set $K = \log(R/r)$
for $t = 0, \ldots, T$ **do**
  Ball Sampling Trial:
  for $k = 0, \ldots, K$ **do**
    Set $r_k = 2^{-k}R$.
    Sample $v_k \sim r_k\mathcal{D}$.
  end
  Update: $x_{t+1} = \arg\min_k \left\{ f(y) \middle| y = x_t, y = x_t + v_k \right\}$
end
return x_T



Block Coordinate Ascent

Choose $\theta^0 \in \mathbb{R}^d$;
Set $i \leftarrow 0$;
repeat
  Given θ^i, choose a block of coordinates b in the set $\{1, ..., d\}$ and compute new iterate θ^{i+1} that satisfies;

$$\theta_b^{i+1} \in \arg\max_{\theta_b} f(\theta_p, \theta_b) \tag{1}$$

$$\theta_p^{i+1} = \theta_p^i, \forall p \notin b \tag{2}$$

  $i \leftarrow i + 1$;
until *termination test satisfied*;

Algorithm 1: Block Coordinate Ascent

Approximate Block Coordinate Ascent (ABCA)

Input: Initial policy parameters $\theta^0 \in \mathbb{R}^d$;
Hyperparameters: Search range s, number of noise samples n, Number of coordinates per block c, number of rollouts in mean calculation M;
Set $i \leftarrow 0$;
repeat

Let coordinate block b consist of c randomly selected coordinates from the set $\{1, ..., d\}$;
Sample $\epsilon_{j,k} \sim \mathrm{U}(s)$ **for** $j = 1, ..., n$, and $k \in b$;
Let $\hat{\theta}^{(j)}$ be the j^{th} candidate optimizer (perturbation) over block b with values:

$$\forall q \in \{1, \ldots, d\} \setminus b,\ \hat{\theta}^{(j)}_q \leftarrow \theta^i_q, \quad \forall k \in b,\ \hat{\theta}^{(j)}_k \leftarrow \theta^i_k + \epsilon_{j,k} \tag{3}$$

Using the policy samples $a_i \sim \pi_{\hat{\theta}^{(j)}}(s_i)$ and environment samples $s_{i+1} \sim p(s_{i+1} \mid s_i, a_i)$, generate M rollouts $t^{(j)}_m$, and compute respective cumulative rewards $\mathbf{R}(t^{(j)}_m)$, $\forall m \in \{1, \ldots, M\}$:

$$t^{(j)}_m \leftarrow \left\{(s^{(j)}_{m,0}, a^{(j)}_{m,0}), (s^{(j)}_{m,1}, a^{(j)}_{m,1}), \ldots\right\} \quad \mathbf{R}(t^{(j)}_m) = \sum_{(s,a) \in t^{(j)}_m} R(s, a) \tag{4}$$

Update θ based on the empirical average cumulative rewards $\bar{\mathbf{R}}(\hat{\theta}^{(j)}) \leftarrow \frac{1}{M} \sum_{m=1}^{M} \mathbf{R}(t^{(j)}_m)$:

$$\mathbf{j} \leftarrow \arg\max_j \bar{\mathbf{R}}(\hat{\theta}^{(j)}), \quad \theta^{i+1} \leftarrow \hat{\theta}^{(\mathbf{j})} \tag{5}$$

$i \leftarrow i + 1$;
until *termination test satisfied*;

Algorithm 2: Approximate Block Coordinate Ascent (ABCA)

Hypothesis Driven… Coordinate Ascent?

- Need to carefully examine variance over $\mathbb{E}[\mathbf{R}(\hat{\theta}^{(j)})]$
  - Why? Because it may be due to chance rather than an improvement in parameters that the return is better (other BBO algorithms incorrectly take the mean of rollouts as ground truth)
    - Mean: $\bar{\mathbf{R}}(\hat{\theta}^{(j)}) \leftarrow \frac{1}{M}\sum_{m=1}^{M}\mathbf{R}(t_m^{(j)})$
    - Variance: $\sigma^2_{\mathbf{R}(\hat{\theta}^{(j)})} \leftarrow \frac{1}{M}\sum_{m=1}^{M}(\mathbf{R}(t_m^{(j)}) - \bar{\mathbf{R}}(\hat{\theta}^{(j)}))^2$
- Enter: one-sided two sample t-test
  - Null Hypothesis: $H_0^{(i)} \leftarrow \mathbb{E}[\pi_{\theta^{(i)}}] > \mathbb{E}[\pi_{\bar{\theta}^{(i)}}]$
  - Need to estimate our test statistic
    - Recall:
      - z statistic: $\frac{\bar{x}-\mu_{H_0}}{\sigma_{\bar{x}}}$ or $\frac{\bar{x}-\mu_{H_0}}{\frac{\sigma_{pop}}{\sqrt{n}}}$
      - t statistic: $\frac{\bar{x}-\mu_{H_0}}{\frac{\sigma_x}{\sqrt{n}}}$
  - For our t-test: $T_j = \frac{\bar{\mathbf{R}}(\hat{\theta^{(j)}}) - \bar{\mathbf{R}}(\hat{\theta^{i}})}{\sqrt{\frac{\sigma^2_{\mathbf{R}(\hat{\theta}^{(j)})}}{M} + \frac{\sigma^2_{\mathbf{R}(\hat{\theta}^{i})}}{M}}}$

Hypothesis Driven?

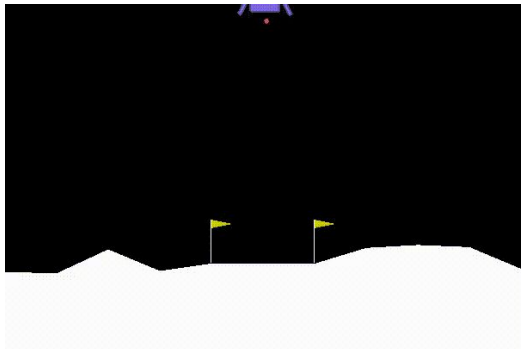
- We reject H_0 if: $T_j > t_{1-\alpha, v_j}$
  - With significance threshold ***α***
    - Controls policy exploration!
  - And degrees of freedom: $$v_j = \frac{(\frac{\sigma^2_{\mathbf{R}(\hat{\theta}^{(j)})}}{M} + \frac{\sigma^2_{\mathbf{R}(\hat{\theta}^{i})}}{M})^2}{\frac{(\frac{\sigma^2_{\mathbf{R}(\hat{\theta}^{(j)})}}{M})^2}{M-1} + \frac{(\frac{\sigma^2_{\mathbf{R}(\hat{\theta}^{i})}}{M})^2}{M-1}}$$
- We now replace Eq 5 in ABCA with:
  - Where $$\mathbf{j} \leftarrow \arg\max_j T_j - t_{1-\alpha, v_j}$$

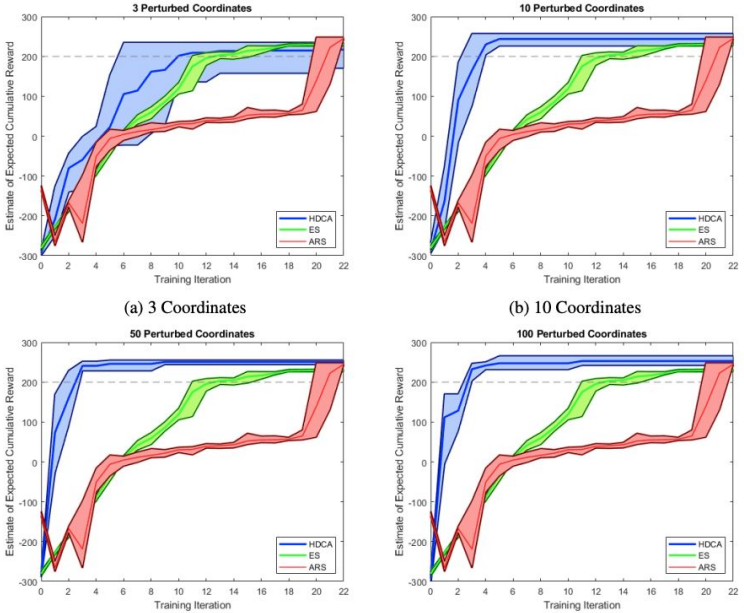
$$\theta^{i+1} \leftarrow \hat{\theta^{(\mathbf{j})}} \textbf{ if } T_{\mathbf{j}} - t_{1-\alpha, v_j} > 0, \textbf{ else } \theta^i$$

Effects of Sub-Problem Dimensionality

Table 1: Average number of training iterations before solving LunarLanderContinuous-v2

Method	Average Training Iterations
HDCA-3	14.8 ± 10.5
HDCA-10	4.4 ± 0.80
HDCA-50	4.6 ± 2.33
HDCA-100	5.2 ± 2.99
HDCA (Average)	7.25 ± 4.37



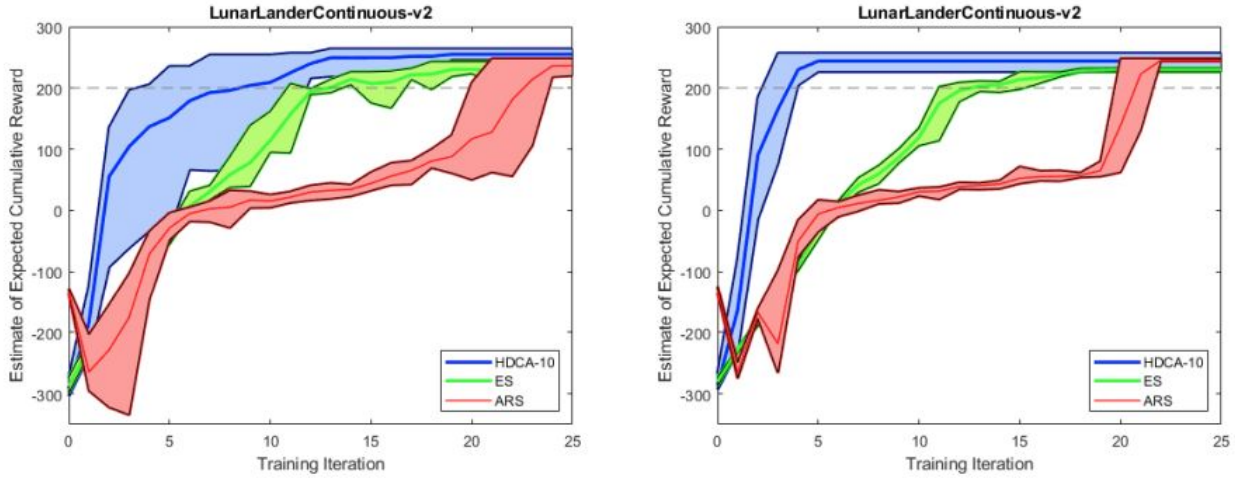


(a) 3 Coordinates (b) 10 Coordinates

(c) 50 Coordinates (d) 100 Coordinates

Figure 1: Training performance for HDCA compared to ARS and ES for the LunarLanderContinuous-v2 environment. The estimated expected cumulative reward is plotted against the training iteration, where training continued until the expected reward minus its standard deviation exceeded the threshold of 200. HDCA and ES were trained on a neural network with one hidden layer consisting of 100 neurons, while ARS trained a linear policy. 5,000 directions were evaluated at each training iteration for all three methods, and the number of coordinates perturbed in each subplot refers to the block size b for HDCA.

Effects of Perturbations per Iteration



(a) 250 Perturbed Policies per Iteration

(b) 5000 Perturbed Policies per Iteration

Figure 2: Training performance for HDCA-10 compared to ARS and ES for the LunarLanderContinuous-v2 environment. The estimated expected cumulative reward is plotted against the training iteration, where training continued until the expected reward minus its standard deviation exceeded the threshold of 200. The left figure shows the expected number of training iterations when 250 samples are taken at each iteration, and the right shows the same relationship for 5000 samples.

Comparison of HDCA

Table 2: Average number of training iterations before solving various MuJoCo locomotion tasks

Task (Threshold)	Method	Average Training Iterations
Swimmer-v2 (325)		
	HDCA	2.2 ± 0.75
	ARS	25.6 ± 2.65
	GLD	$2.2 \pm .4$
Hopper-v2 (3120)		
	HDCA	80.8 ± 30.9
	ARS	121 ± 34.9
	GLD	393 ± 245
HalfCheetah-v2 (3430)		
	HDCA	116 ± 72.8
	ARS	93.6 ± 11.3
	GLD	*

Conclusion

- In Summary…
  - Gradient based (or gradient approximating) BBO algorithms often can't handle complex domains, get stuck in local optima
  - They can't take advantage of the max operator
- HDCA addresses these issues via both avoiding gradient approximation and testing each perturbed policy's performance to ensure updates are statistically significant
  - Additionally, it is very parallelizable (similar to other derivative-free optimization approaches)
  - It is customizable to each application by tuning $\boldsymbol{\alpha}$ (policy exploration)

Q&A