Policy-based Reinforcement Learning

Christopher Mutschler

Vapnik's rule

"Never solve a more general problem as an intermediate step."

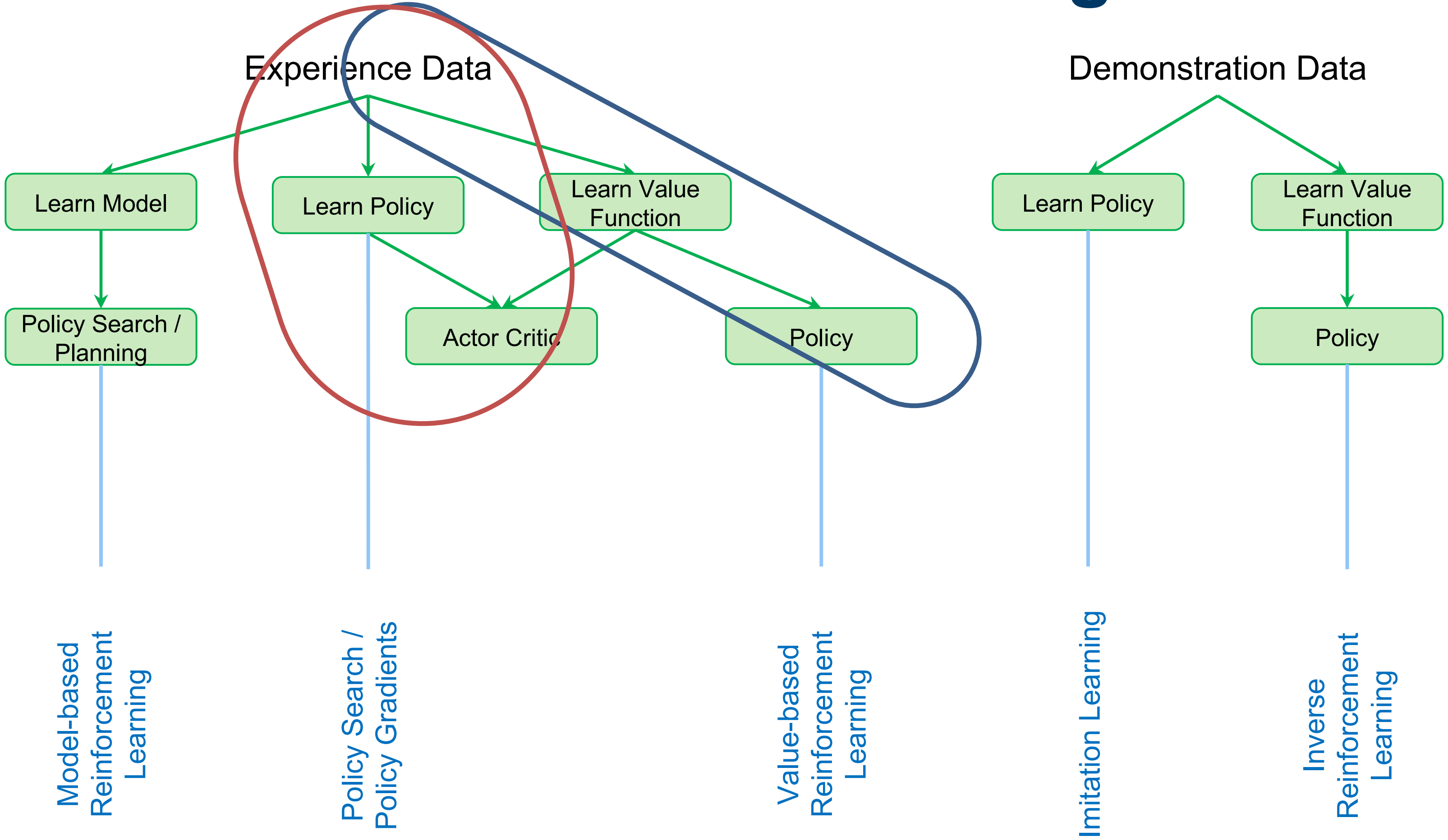
\- *Vladimir Vapnik, 1998*

- Remember:

"New goal: find a policy that maximizes the expected return!"

- If we care about optimal behavior: why not learn a policy directly?

Policy-based Reinforcement Learning

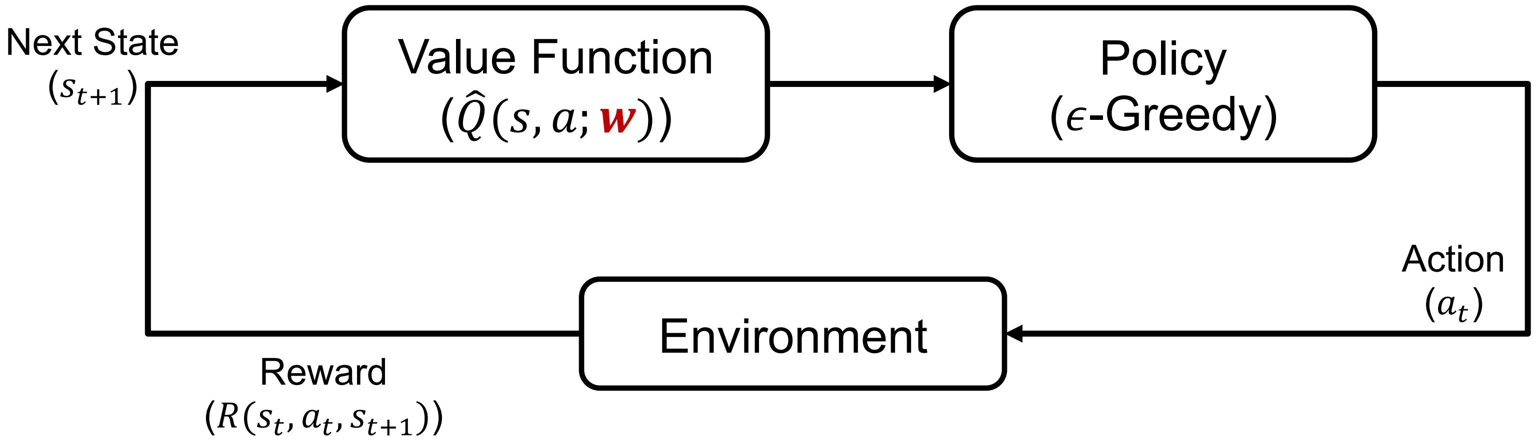


Policy-based Reinforcement Learning

- Previously we approximated parametric value functions:

$$v_w(s) \approx v_\pi(s)$$
$$q_w(s, a) \approx q_\pi(s, a)$$

- A policy can be generated from these values
  - e.g., greedy or ϵ-greedy



Goal: find w that approximates the true Q-function

Policy-based Reinforcement Learning

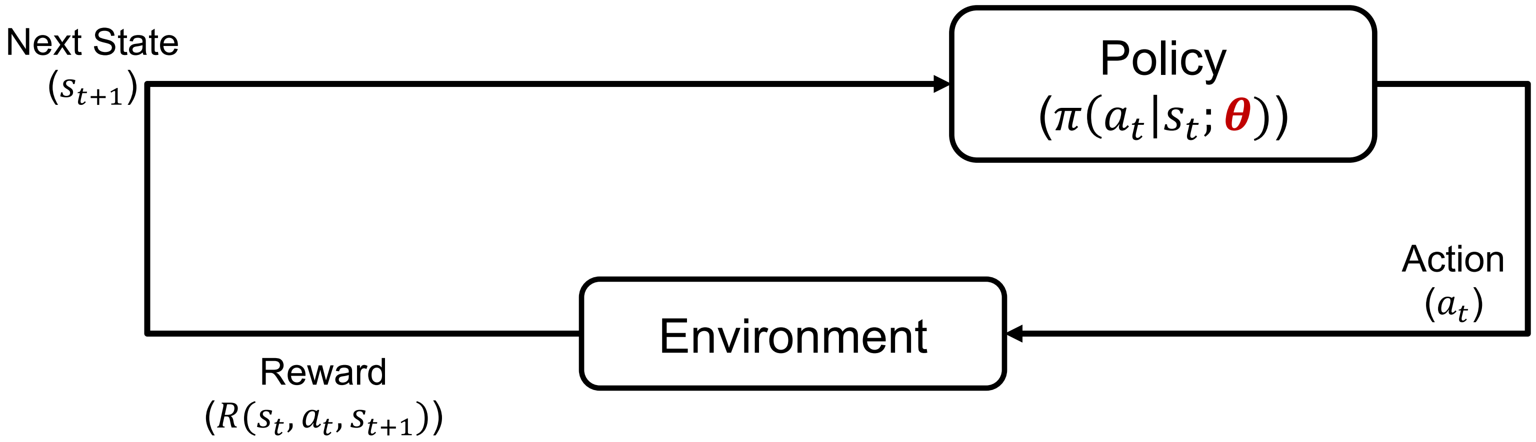
- Previously we approximated parametric value functions:

$$v_w(s) \approx v_\pi(s)$$
$$q_w(s,a) \approx q_\pi(s,a)$$

- A policy can be generated from these values
- In this lesson we will directly parameterize the policy:

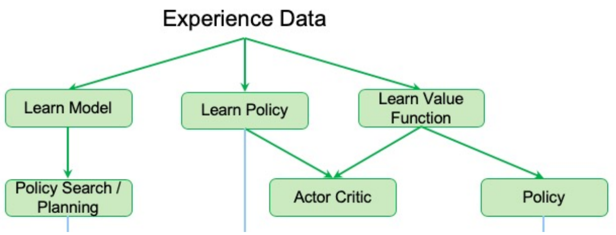
$$\pi_\theta\,(a|s) = p(a|s;\theta)$$

- We still focus on model-free reinforcement learning



Goal: find θ that maximizes long term reward

General overview



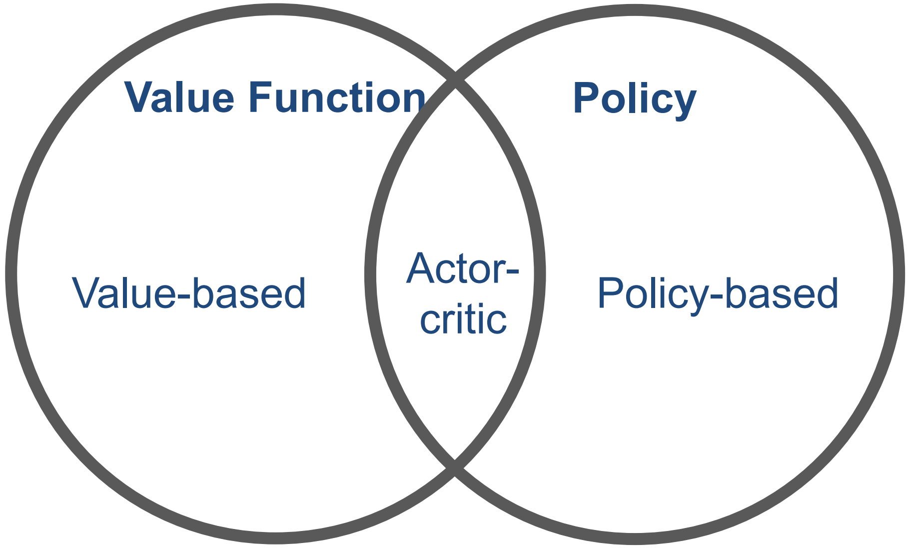
- **Model-based RL:**
  - \+ "Easy" to learn a model (supervised learning)
  - \+ Learns *all there is to know* from the data
  - \- Objective captures irrelevant information
  - \- May focus computations/capacity on irrelevant details
  - \- Computing policy (planning) is non-trivial and can be computationally expensive

- **Value-based RL:**
  - \+ Closer to true objective
  - \+ Fairly well-understood: somewhat similar to regression
  - \- Still not the true objective: may still focus capacity on less-important details

- **Policy-based RL:**
  - \+ Right objective!
  - \- Ignores other learnable knowledge (potentially not the most efficient use of data)

Value-based vs. Policy-based RL

- **Value-based**
  - Learn value function
  - Implicit policy
    (e.g., ϵ-greedy)
- **Policy-based**
  - No value function
  - Learn policy
- **Actor-critic**
  - Learn value function
  - Learn policy



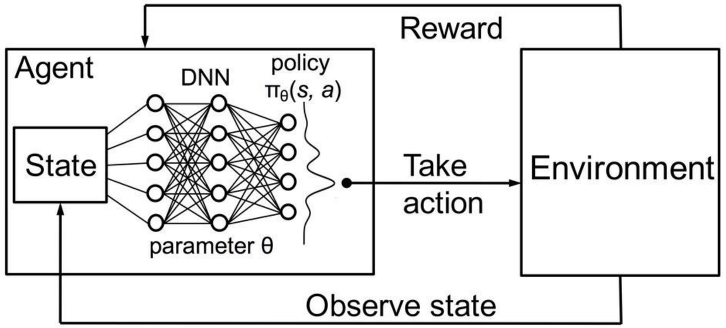
Advantages of Policy-based RL

- Advantages:
  - Good convergence properties
  - Easily extended to high-dimensional or continuous state and action spaces
  - Can learn *stochastic* policies
  - Sometimes policies are simple while values and models are complex
    - e.g., rich domain, but optimal is always to go left

- Disadvantages:
  - Susceptible to local optima (especially with non-linear FA)
  - Obtained knowledge is specific, does not always generalize well
  - Ignores a lot of information in the data (when used in isolation)

Stochastic Policies

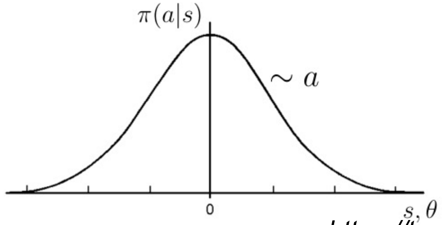
- We have seen deterministic policies like this:
  - State gives $Q(s, a; w)$ and we selected $\pi(a|s)$ by $\operatorname{argmax}_a Q(s, a; w)$



- Instead, stochastic policies do something like this:

$$\pi(a|s) = \mathbb{P}[a|s; \theta]$$

(policy is represented as a probability distribution)



https://towardsdatascience.com/self-learning-ai-agents-iv-stochastic-policy-gradients-b53f088fce20

Why do we need stochastic policies?

Example #1: Rock-Paper-Scissors

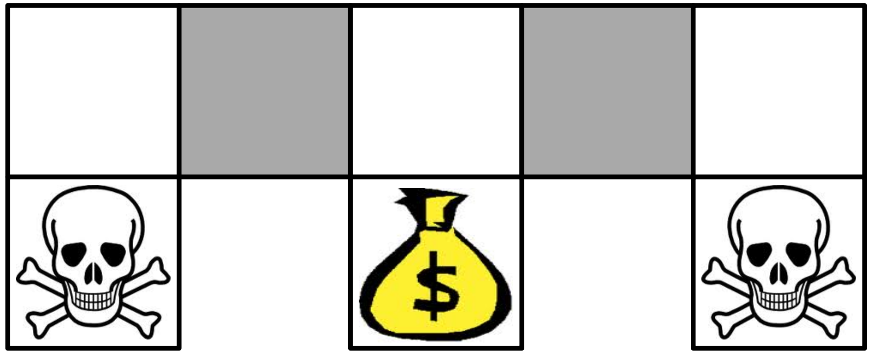
- Two-player game of rock-paper-scissors
  - Scissors beats paper
  - Rock beats scissors
  - Paper beats rock

- Consider policies for iterated rock-paper-scissors
  - A deterministic policy (e.g., greedy or even ϵ-greedy) is easily exploited
  - A uniform random policy is the optimal policy (i.e., Nash equilibrium)

David Silver, UCL Lecture on Reinforcement Learning. 2015

Why do we need stochastic policies?

Example #2: Aliased Gridworld



- Consider features of the following form (for all N, E, S, W):

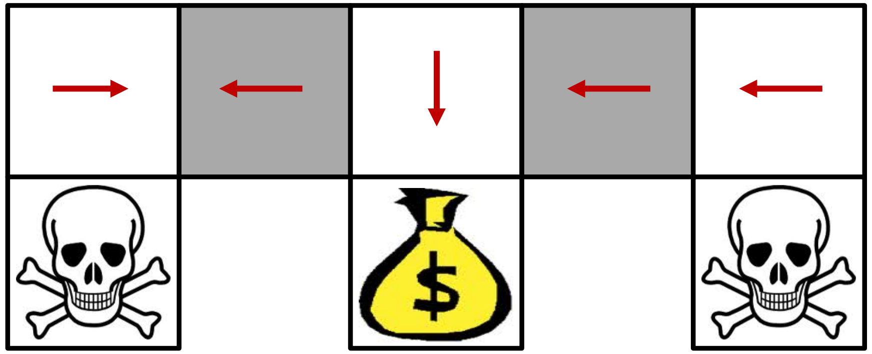
$$\phi(s,a) = \overbrace{\underbrace{1}_{\text{N}}\quad \underbrace{0}_{\text{E}}\quad \underbrace{1}_{\text{S}}\quad \underbrace{0}_{\text{W}}}^{\text{walls}}\quad \overbrace{\underbrace{0}_{\text{N}}\quad \underbrace{1}_{\text{E}}\quad \underbrace{0}_{\text{S}}\quad \underbrace{0}_{\text{W}}}^{\text{actions}}$$

- The agent cannot differentiate the grey states
- Compare *deterministic* and *stochastic* policies

David Silver, UCL Lecture on Reinforcement Learning. 2015

Why do we need stochastic policies?

Example #2: Aliased Gridworld

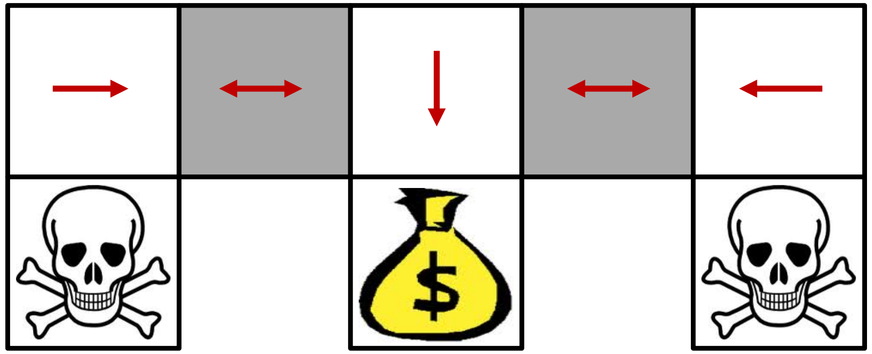


- Value-based RL learns a near-deterministic policy
  - e.g., greedy or ϵ-greedy
- Under aliasing, an optimal *deterministic* policy will either
  - Move W in both grey states (shown by red arrows)
  - Move E in both grey states
- Either way, it can get stuck and never reach the money
- Hence, it will traverse the corridor for a long time

David Silver, UCL Lecture on Reinforcement Learning. 2015

Why do we need stochastic policies?

Example #2: Aliased Gridworld



- Instead,
  an optimal *stochastic* policy moves randomly E or W in grey states:

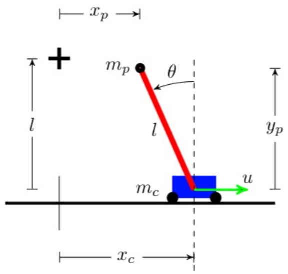
$$\pi_\theta(\text{wall to N and S, move E}) = 0.5$$
$$\pi_\theta(\text{wall to N and S, move W}) = 0.5$$

- Will reach the goal state in a few steps with high probability
- Policy-based RL can learn the optimal stochastic policy

David Silver, UCL Lecture on Reinforcement Learning. 2015

Why is it better to learn the policy directly?

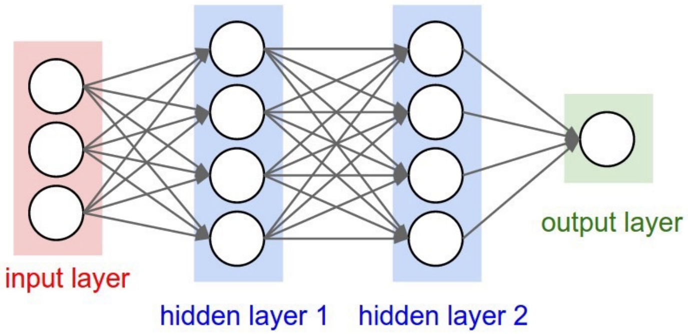
- Example: Cartpole



$(x_t, \dot{x_t}, \vartheta, \dot{\vartheta}_t)$

$$a_t = \boldsymbol{\theta_0} + \boldsymbol{\theta_1} x_t + \boldsymbol{\theta_2} \dot{x_t} + \boldsymbol{\theta_3} \vartheta + \boldsymbol{\theta_4} \dot{\vartheta}$$

$(x_t, \dot{x_t}, \vartheta, \dot{\vartheta}_t)$



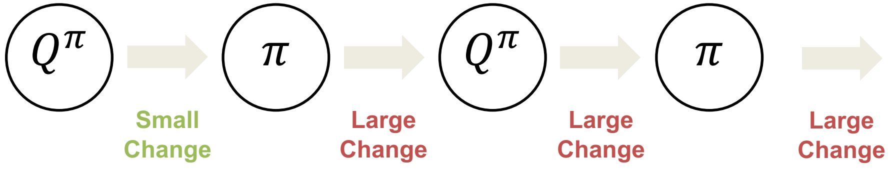
a_t

Why is it better to learn the policy directly?

- Learn directly a policy without calculating value functions in between
- Why?

  - **Greedy updates**

$$\theta_{n+1} = \underset{\theta}{\operatorname{argmax}}\, E_{\pi_\theta}\{Q^\pi(s,a)\}$$



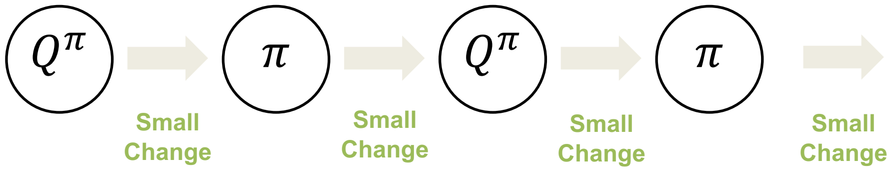
Potentially unstable learning process with large policy “jumps”

  - **Smooth updates**

$$\theta_{n+1} = \theta_n + \boxed{\alpha_n \nabla G_{\theta_n}}$$

Reminder:

$$\mathrm{G} = r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \cdots = \sum_{t=0}^{\infty} \gamma^t r_t$$



Stable learning process with smooth policy improvement

Why is it better to learn the policy directly?

- Learn directly a policy without calculating value functions in between
- How to calculate the gradient term?

$$\theta_{n+1} = \theta_n + \boxed{\alpha_n \nabla G_{\theta_n}}$$

  - Simple optimization: **Finite Difference Stochastic Approximation (FDSA)**
  - Idea: to evaluate the gradient, for each dimension $k \in [1, n]$:
    - Estimate k-th partial derivative of objective function w.r.t. θ by perturbation θ by a small amount ϵ in k-th dimension:

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon},$$

    where u_k is a unit vector with 1 in k-th component, 0 elsewhere; $\lim_{n \to \infty} \epsilon = 0$

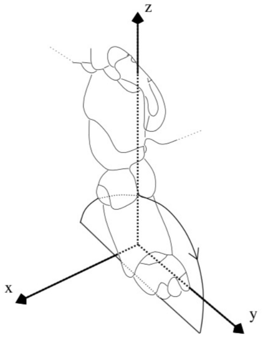
  - **In RL literature**: "Finite Difference Gradient Estimator"
  - **Note:** a variation in control literature is called Simultaneous Perturbation Stochastic Approximation (SPSA)

- Simple, noisy, inefficient – but sometimes effective

  → works for arbitrary policies (even if they are not differentiable)!

Example: AIBO with FDSA

- Learn fast walking patterns for RoboCup (speed decides on win/lose)
- Policy parametrized as an ellipsoid (12 parameters)
- Adapt parameters by (sampled) FDSA
- Policy evaluated by field traversal time



http://www.cs.utexas.edu/users/AustinVilla/?p=research/learned_walk

	π_1	$\pi_2 - \pi_N$	Score	
$-\epsilon_1$	$\theta_1 - \epsilon_1$	...	207	
	$\theta_1 - \epsilon_1$	...	214	$\Rightarrow$ Average: 210
		...		
$+0$	$\theta_1 + 0$	...	225	
	$\theta_1 + 0$	...	220	$\Rightarrow$ Average: 220
		...		
$+\epsilon_1$	$\theta_1 + \epsilon_1$	...	239	
	$\theta_1 + \epsilon_1$	...	244	$\Rightarrow$ Average: 240
		...		

Kohl et al.: Policy gradient reinforcement learning for fast quadrupedal locomotion. ICRA' 2004.

Example: AIBO with FDSA

- Learn fast walking patterns for RoboCup (speed decides on win/lose)
- Policy parametrized as an ellipsoid (12 parameters)
- Adapt parameters by (sampled) FDSA
- Policy evaluated by field traversal time

Problems:

- Requires **A LOT** of samples/trajectories
- In stochastic environments, with small c_n it is really hard to distinguish the difference between R^+ and R^-

Better: Augmented Random Search (ARS)

- Builds on the **Basic Random Search (BRS)** Algorithm:
  1. Pick a policy π_θ, perturb the parameters θ by applying $+v\delta$ and $-v\delta$ ($v < 1$ is constant noise and δ is a random number sampled from a normal distribution)
  2. Run the policies and apply actions based on $\pi(\theta + v\delta)$ and $\pi(\theta - v\delta)$ and collect the rewards $r(\theta + v\delta)$ and $r(\theta - v\delta)$
  3. For all δ compute the average $\Delta = \frac{1}{N} \cdot \Sigma[r(\theta + v\delta) - r(\theta - v\delta]\delta$ and update the parameters θ using Δ and a learning rate α:

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

- Augmented Random Search (ARS) adds 3 improvements:
  1. Divide by the rewards by their standard deviation σ_r
  2. Normalize the states
  3. Only use the top-k best rollouts to compute the average

Better: Augmented Random Search (ARS)

Algorithm 1 Augmented Random Search (ARS): four versions **V1**, **V1-t**, **V2** and **V2-t**

```
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)
Initialize: M_0 = 0 ∈ R^{p×n}, μ_0 = 0 ∈ R^n, and Σ_0 = I_n ∈ R^{n×n}, j = 0.
while ending condition not satisfied do
    Sample δ_1, δ_2, ..., δ_N in R^{p×n} with i.i.d. standard normal entries.
    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 $k \in \{1, 2, \ldots, N\}$.

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.

Make the update step:

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

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

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]

$j \leftarrow j + 1$

end while

Sample N different variations for the policy parameters $(\boldsymbol{\delta_i})$

Run $2N$ simulations/rollouts for the positive and negative directions

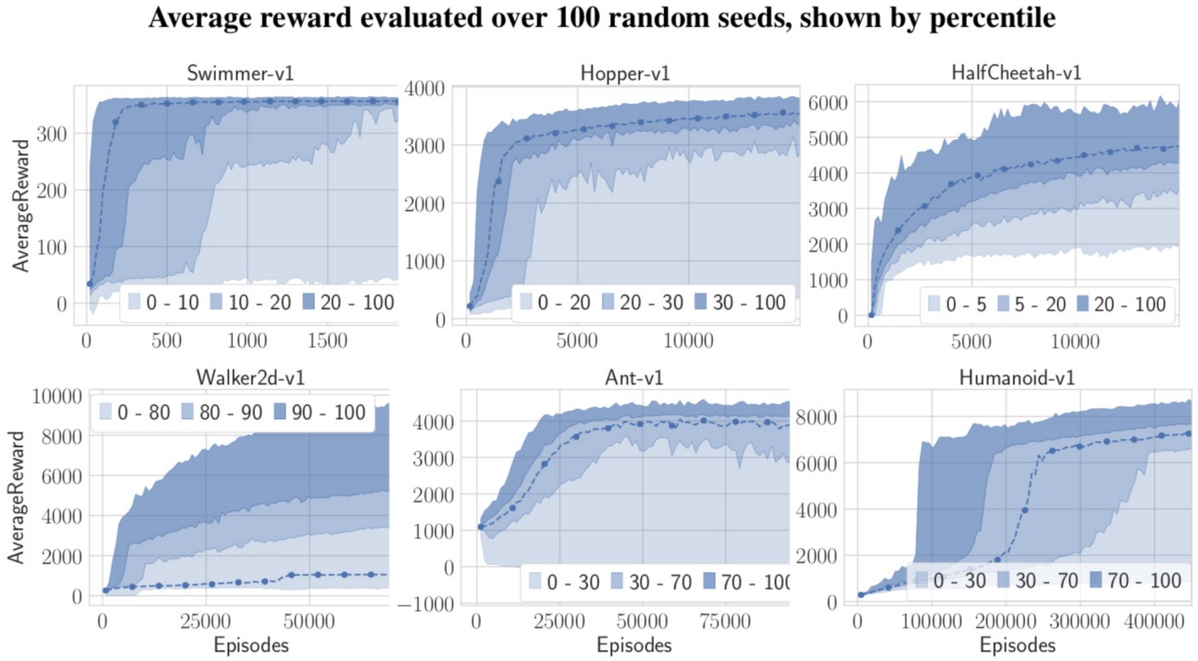
In Vx-t version of the algorithm, select only the best **b** rollouts for the parameter update

To avoid tuning the learning rate, scale the update by the standard deviation $(\boldsymbol{\sigma_R})$ of the $\boldsymbol{2b}$ returns used for the update

In V2 of the algorithm, do not use the state observed as input but normalize states using the running mean and variance of all states observed so far

Better: Augmented Random Search (ARS)

- State-of-the Art algorithm extending classical random search method
- Comparable performance to modern Deep RL algorithms
- Robust to hyper-parameters and minimum tuning required
- Developed by the Control Engineering Community!



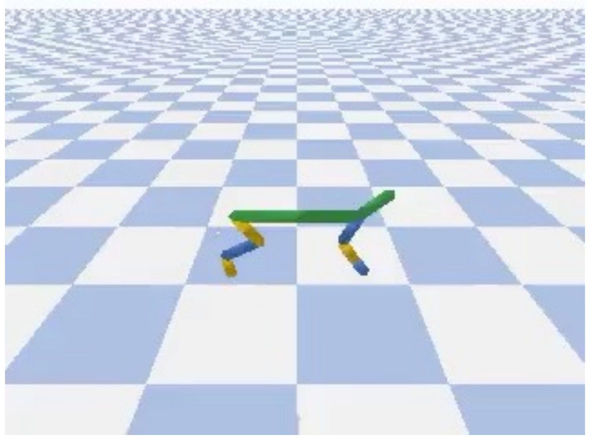
Mania et al.: Simple random search of static linear policies is competitive for reinforcement learning. NeurIPS 2018.

see also: https://towardsdatascience.com/introduction-to-augmented-random-search-d8d7b55309bd

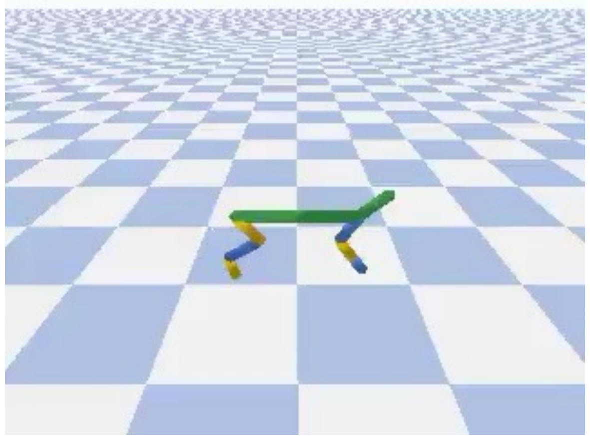
Better: Augmented Random Search (ARS)

- State-of-the Art algorithm extending classical random search method
- Comparable performance to modern Deep RL algorithms
- Robust to hyper-parameters and minimum tuning required
- Developed by the Control Engineering Community!

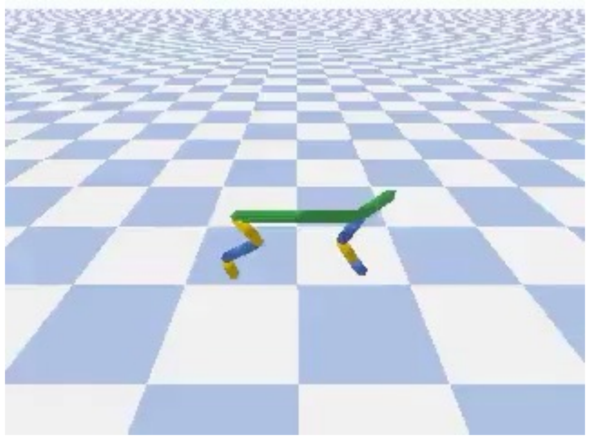
at the beginning…



after 100 iterations:



after 300 iterations:



see also: https://towardsdatascience.com/introduction-to-augmented-random-search-d8d7b55309bd

Better: Augmented Random Search (ARS)

Pros:

- Simple to understand and implement
- Less parameter tuning and robust to hyper-parameters
- Embarrassingly easy to parallelize

Cons:

- They tend to favor “lucky” rollouts
- In stochastic environments it is not easy to distinguish if good performance is due to parameter variation or environment noise
- They do not exploit the sequential structure of the problem
- They require 10x more samples (approx.) compared to properly tuned Deep RL algorithms

But…wait…uh… What are we doing here?