Fraunhofer IIS

Fraunhofer-Institut für Integrierte Schaltungen IIS

Reinforcement Learning

—

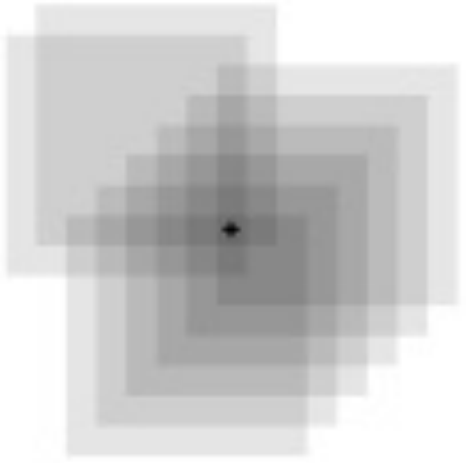
Lecture 6: Policy-based RL 1

Christopher Mutschler

Recap: Value Function Approximation

Tile Coding – what makes more sense?

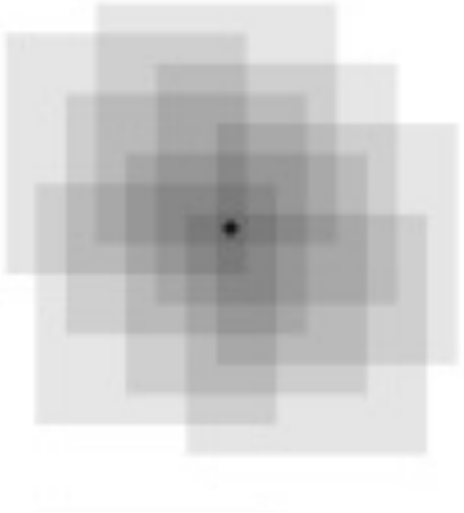
vs.



Recap: Value Function Approximation

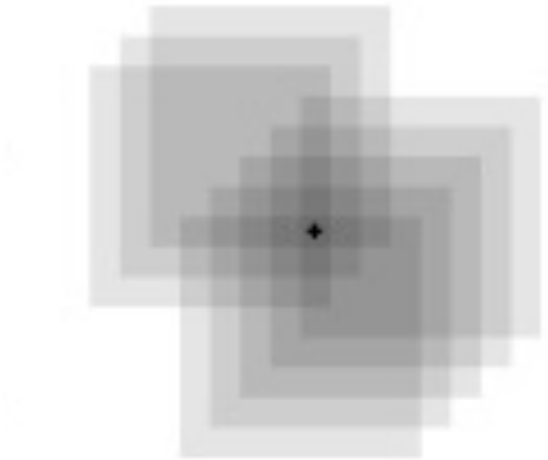
Tile Coding – what makes more sense?

vs.

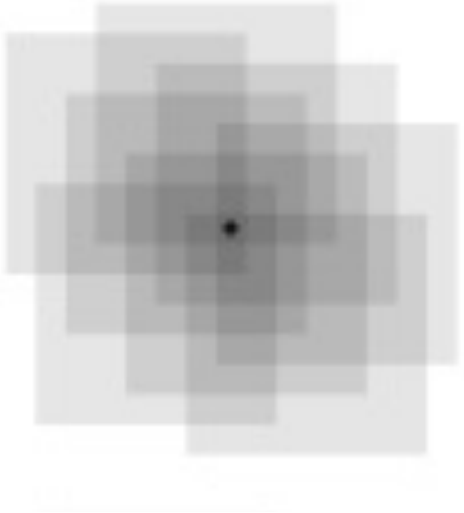


Recap: Value Function Approximation

Tile Coding – what makes more sense?



vs.



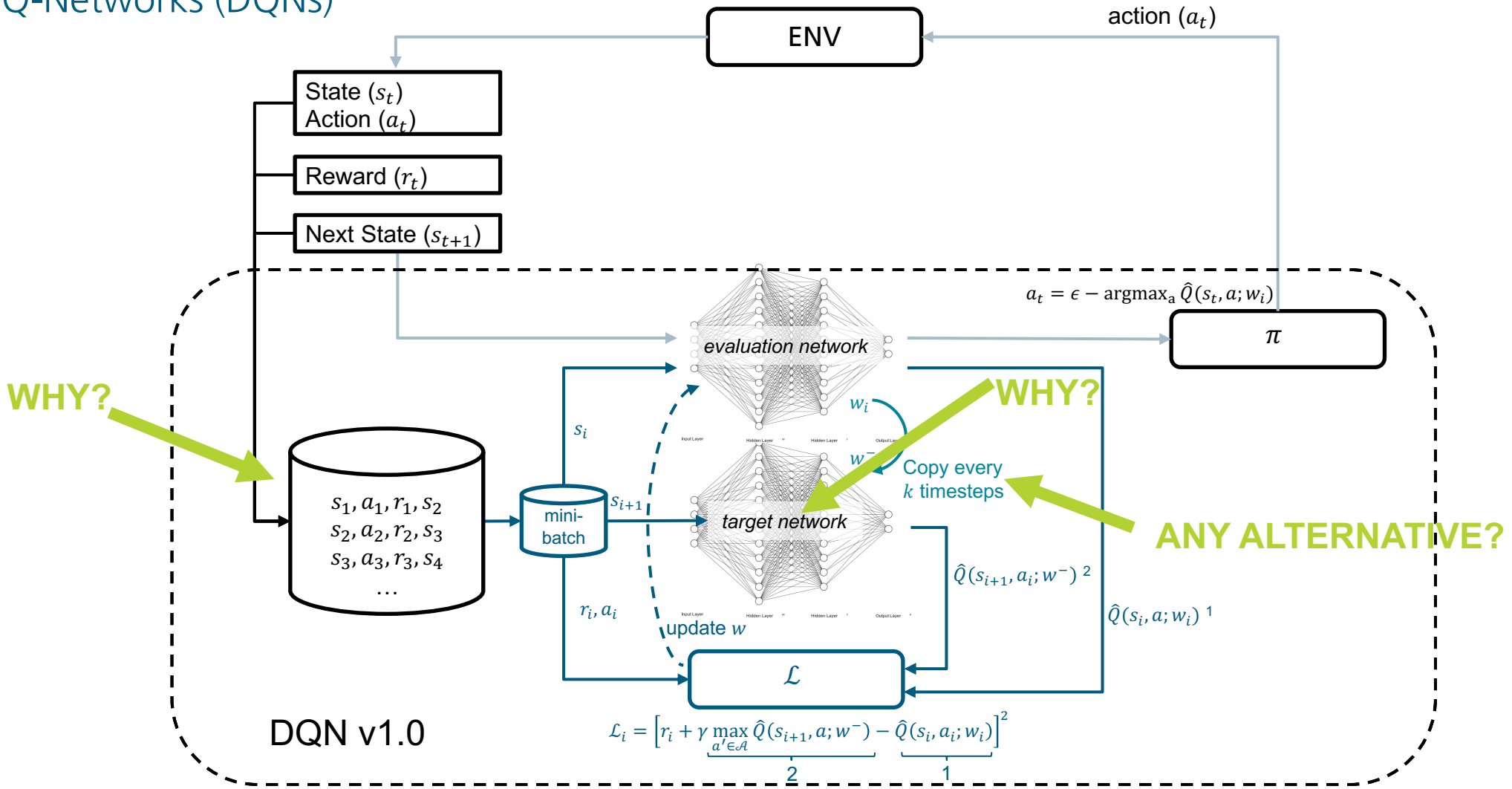
Recap: Value Function Approximation

Tile Coding

Exercise 9.4 Suppose we believe that one of two state dimensions is more likely to have an effect on the value function than is the other, that generalization should be primarily across this dimension rather than along it. What kind of tilings could be used to take advantage of this prior knowledge? □

Recap: Value Function Approximation

Deep Q-Networks (DQNs)



Vapnik’s rule

“Never solve a more general problem as an intermediate step.”

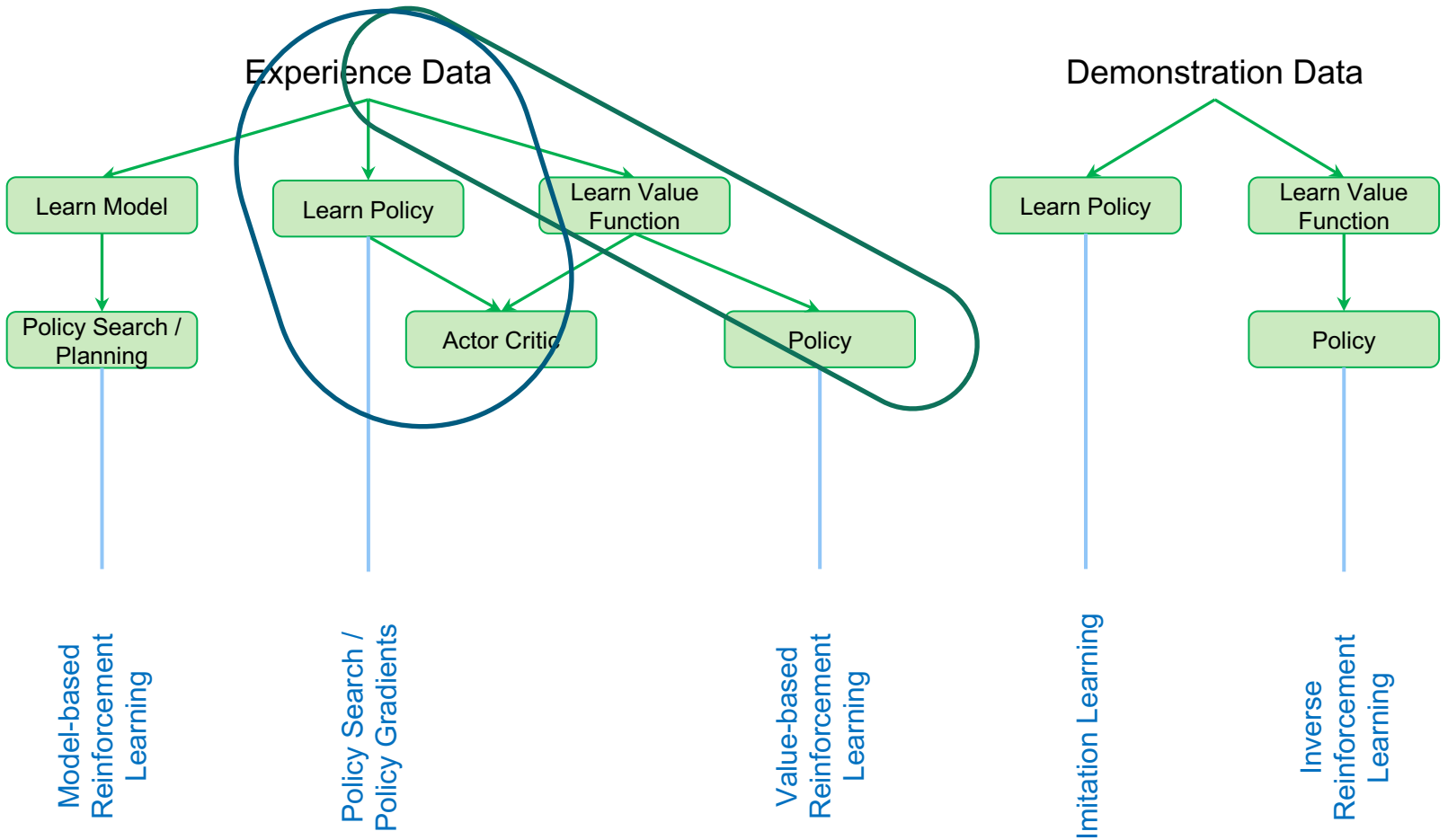
- *Vladimir Vapnik, 1998*

- 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

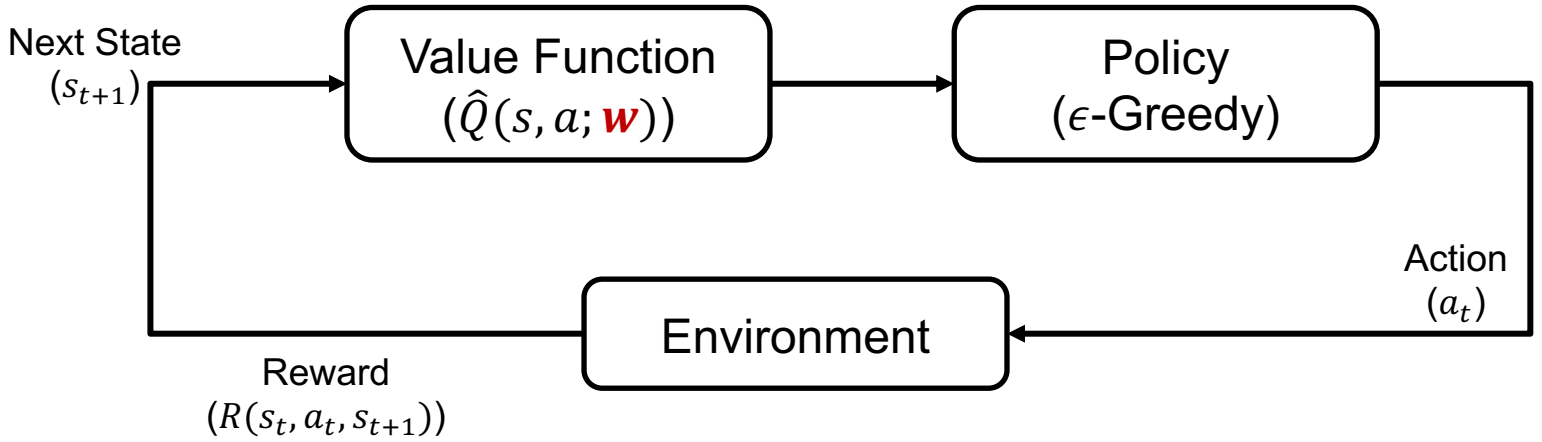


Policy-based Reinforcement Learning

- 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

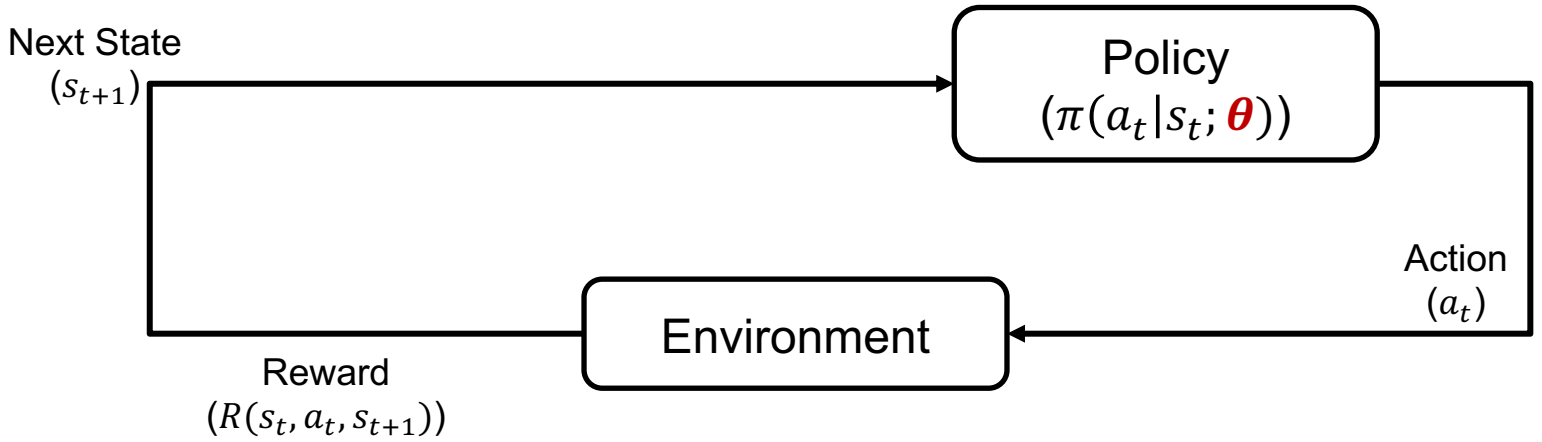
- Previously we approximated parametric value functions:

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

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

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

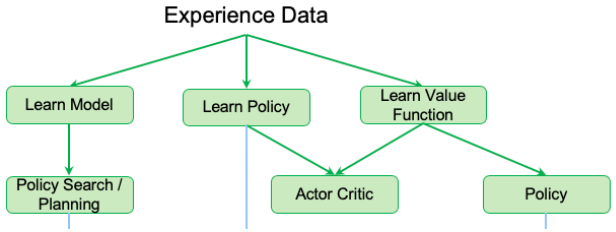
- We still focus on model-free reinforcement learning



Goal: find θ that maximizes long term reward

Policy-based Reinforcement Learning

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

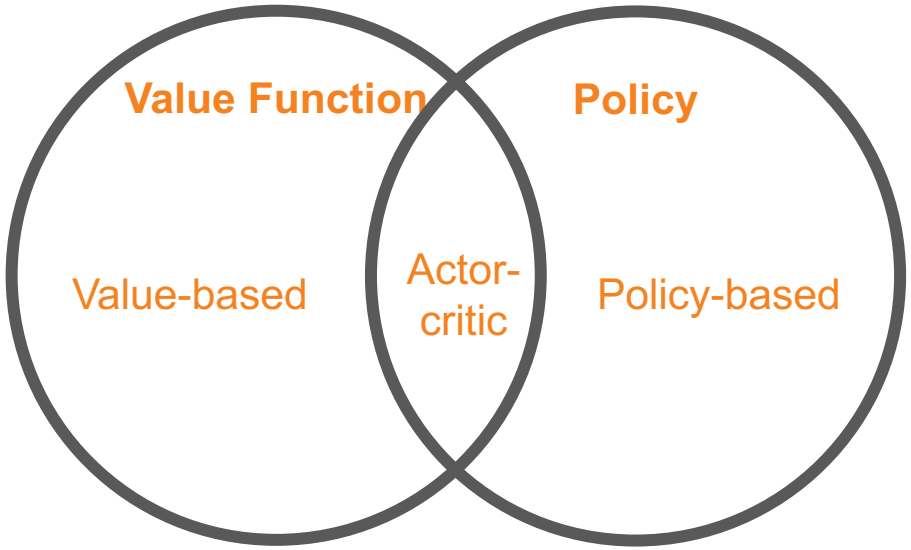
- **Value-based RL:**
  - \+ Closer to true objective
  - \+ Fairly well-understood: somewhat similar to regression
  - \- Still not the true objective: may still focus capacity on less-important details

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

Policy-based Reinforcement Learning

Value-based vs. Policy-based RL

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



Policy-based Reinforcement Learning

Advantages of Policy-based RL

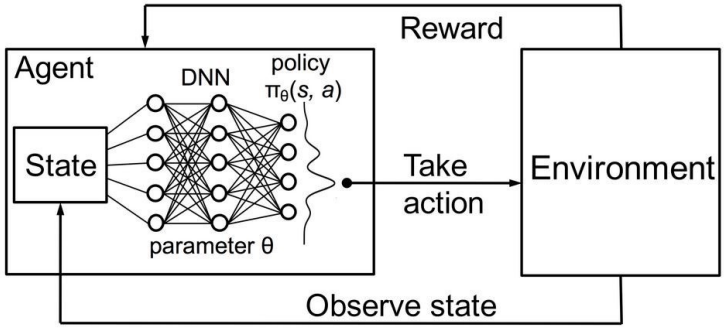
- **Advantages:**
  - Good convergence properties
  - Easily extended to high-dimensional or continuous state and action spaces
  - Can learn *stochastic* policies
  - Sometimes policies are simple while values and models are complex
    - e.g., rich domain, but optimal is always to go left

- **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)

Policy-based Reinforcement Learning

Stochastic Policies

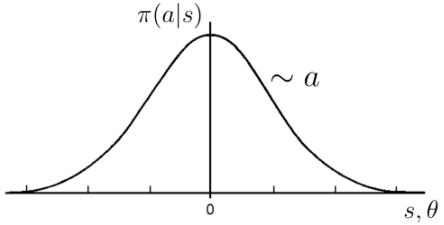
- We have seen deterministic policies like this:
  - State gives $Q(s, a; w)$ and we selected $\pi(a|s)$ by $\text{argmax}_a\, Q(s, a; w)$



- 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

Policy-based Reinforcement Learning

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

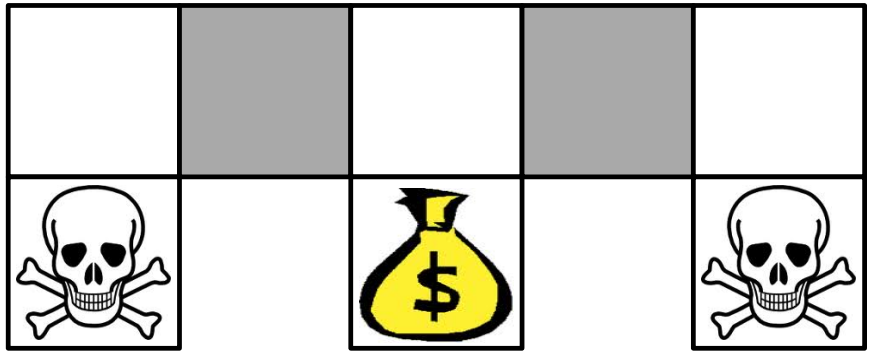
- Consider policies for iterated rock-paper-scissors
  - A deterministic policy (e.g., greedy or even ϵ-greedy) is easily exploited
  - A uniform random policy is the optimal policy (i.e., Nash equilibrium)

David Silver, UCL Lecture on Reinforcement Learning. 2015

Policy-based Reinforcement Learning

Why do we need stochastic policies?

Example #2: Aliased Gridworld



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

$$\phi(s,a) = \underbrace{1}_{N}\ \underbrace{0}_{E}\ \underbrace{1}_{S}\ \underbrace{0}_{W}\ \underbrace{0}_{N}\ \underbrace{1}_{E}\ \underbrace{0}_{S}\ \underbrace{0}_{W}$$

walls actions

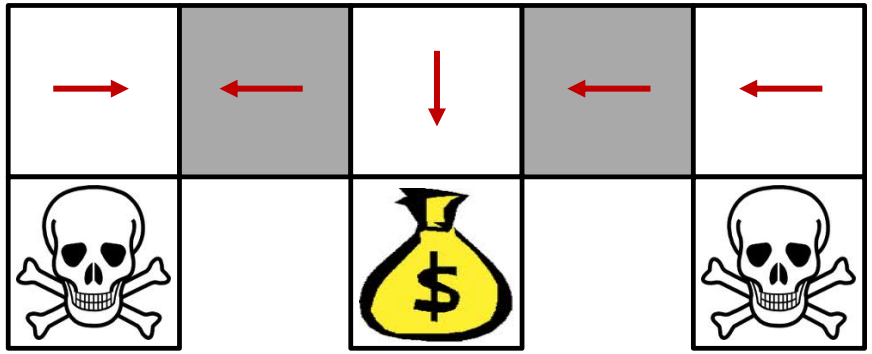
- The agent cannot differentiate the grey states
- Compare *deterministic* and *stochastic* policies

David Silver, UCL Lecture on Reinforcement Learning. 2015

Policy-based Reinforcement Learning

Why do we need stochastic policies?

Example #2: Aliased Gridworld



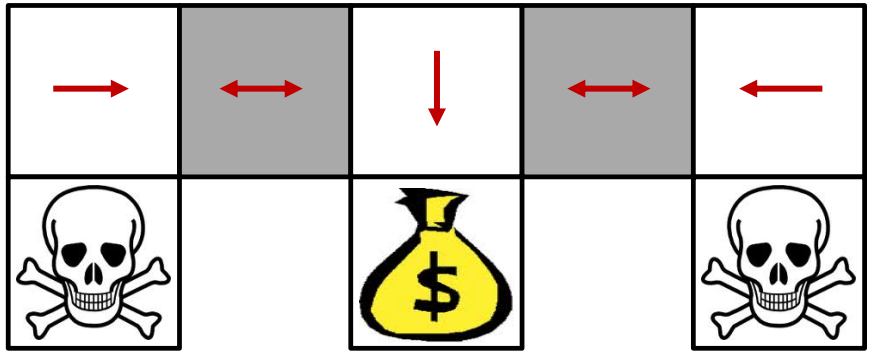
- Value-based RL learns a near-deterministic policy
  - e.g., greedy or ϵ-greedy
- Under aliasing, an optimal *deterministic* policy will either
  - Move W in both grey states (shown by red arrows)
  - Move E in both grey states
- Either way, it can get stuck and never reach the money
- Hence, it will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

David Silver, UCL Lecture on Reinforcement Learning. 2015

Policy-based Reinforcement Learning

Why do we need stochastic policies?

Example #2: Aliased Gridworld



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

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

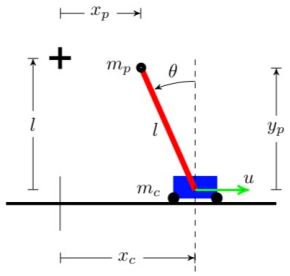
- Will reach the goal state in a few steps with high probability
- **Policy-based RL can learn the optimal stochastic policy!**

David Silver, UCL Lecture on Reinforcement Learning. 2015

Policy-based Reinforcement Learning

Why is it better to learn the policy directly?

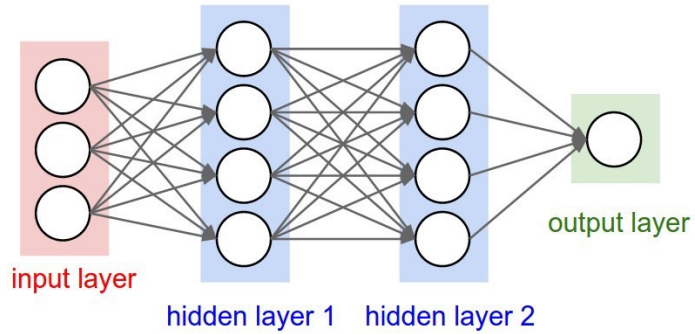
Example #3: Cartpole



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

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

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



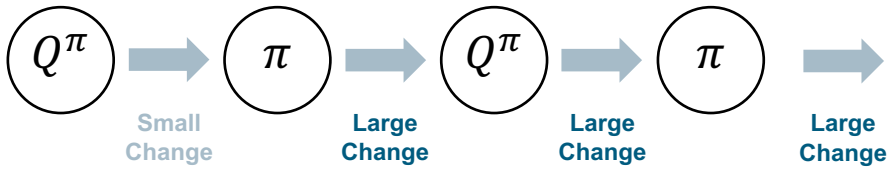
a_t

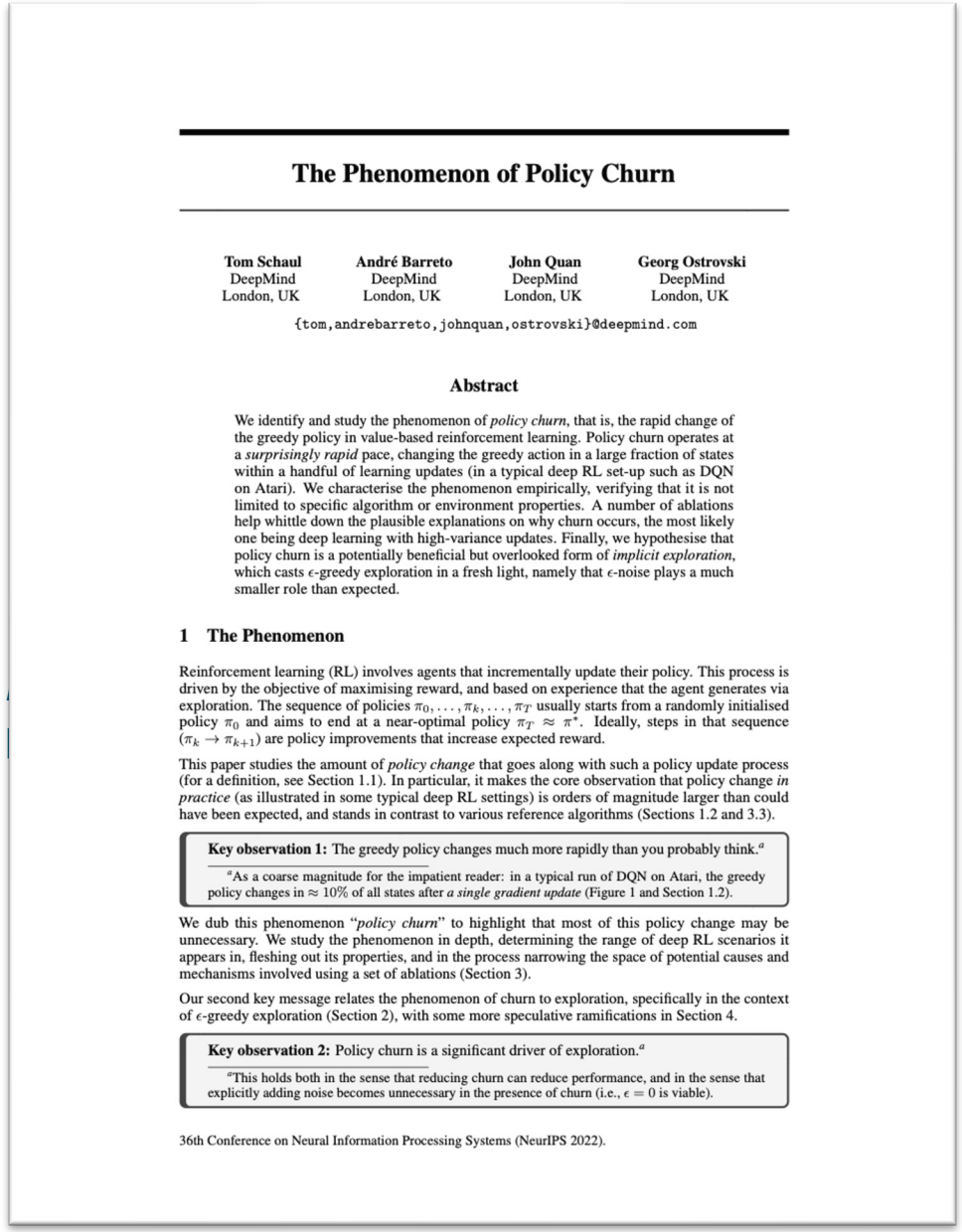
Policy-based Reinforcement Learning

Why is it better to learn the policy directly?

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

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





The Phenomenon of Policy Churn

Tom Schaul DeepMind London, UK
André Barreto DeepMind London, UK
John Quan DeepMind London, UK
Georg Ostrovski DeepMind London, UK
{tom,andrebarreto,johnquan,ostrovski}@deepmind.com

Abstract

We identify and study the phenomenon of *policy churn*, that is, the rapid change of the greedy policy in value-based reinforcement learning. Policy churn operates at a *surprisingly rapid* pace, changing the greedy action in a large fraction of states within a handful of learning updates (in a typical deep RL set-up such as DQN on Atari). We characterise the phenomenon empirically, verifying that it is not limited to specific algorithm or environment properties. A number of ablations help whittle down the plausible explanations on why churn occurs, the most likely one being deep learning with high-variance updates. Finally, we hypothesise that policy churn is a potentially beneficial but overlooked form of *implicit exploration*, which casts ϵ-greedy exploration in a fresh light, namely that ϵ-noise plays a much smaller role than expected.

1 The Phenomenon

Reinforcement learning (RL) involves agents that incrementally update their policy. This process is driven by the objective of maximising reward, and based on experience that the agent generates via exploration. The sequence of policies $\pi_0, \ldots, \pi_k, \ldots, \pi_T$ usually starts from a randomly initialised policy π_0 and aims to end at a near-optimal policy $\pi_T \approx \pi^\star$. Ideally, steps in that sequence $(\pi_k \to \pi_{k+1})$ are policy improvements that increase expected reward.

This paper studies the amount of *policy change* that goes along with such a policy update process (for a definition, see Section 1.1). In particular, it makes the core observation that policy change *in practice* (as illustrated in some typical deep RL settings) is orders of magnitude larger than could have been expected, and stands in contrast to various reference algorithms (Sections 1.2 and 3.3).

Key observation 1: The greedy policy changes much more rapidly than you probably think.[a]

[a] As a coarse magnitude for the impatient reader: in a typical run of DQN on Atari, the greedy policy changes in $\approx 10\%$ of all states after *a single gradient update* (Figure 1 and Section 1.2).

We dub this phenomenon "*policy churn*" to highlight that most of this policy change may be unnecessary. We study the phenomenon in depth, determining the range of deep RL scenarios it appears in, fleshing out its properties, and in the process narrowing the space of potential causes and mechanisms involved using a set of ablations (Section 3).

Our second key message relates the phenomenon of churn to exploration, specifically in the context of ϵ-greedy exploration (Section 2), with some more speculative ramifications in Section 4.

Key observation 2: Policy churn is a significant driver of exploration.[a]

[a] This holds both in the sense that reducing churn can reduce performance, and in the sense that explicitly adding noise becomes unnecessary in the presence of churn (i.e., $\epsilon = 0$ is viable).

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Fraunhofer IIS

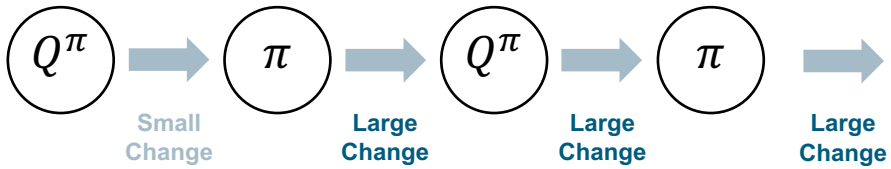
Policy-based Reinforcement Learning

Why is it better to learn the policy directly?

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

  - **Greedy updates**

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



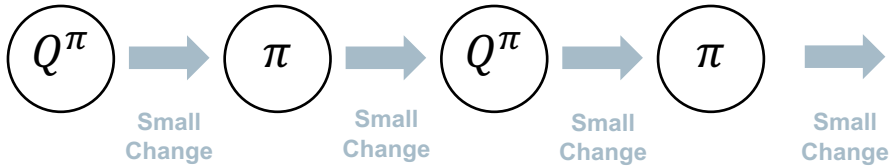
Potentially unstable learning process with large policy "jumps"

  - **Smooth updates**

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

Reminder:

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



Stable learning process with smooth policy improvement

Policy-based Reinforcement Learning

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?**

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

  - 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; $\lim_{j \to \infty} \epsilon = 0$ (j = number of iterations)
  - **In RL literature**: "Finite Difference Gradient Estimator"
  - **Note:** a variation in control literature is called Simultaneous Perturbation Stochastic Approximation (SPSA)

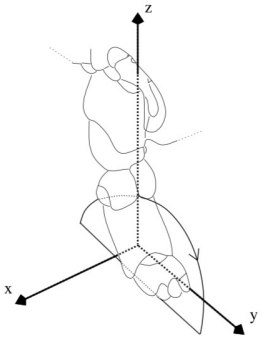
- Simple, noisy, inefficient – but sometimes effective
  → works for arbitrary policies (even if they are not differentiable)!

Policy-based Reinforcement Learning

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.

Policy-based Reinforcement Learning

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 ϵ_n it is really hard to distinguish the difference between R^+ and R^-

Policy-based Reinforcement Learning

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

Algorithm 1 Basic Random Search (BRS)

1: **Hyperparameters:** step-size α, number of directions sampled per iteration N, standard deviation of the exploration noise ν
2: **Initialize:** $\theta_0 = \mathbf{0}$, and $j = 0$.
3: **while** ending condition not satisfied **do**
4: Sample $\delta_1, \delta_2, \ldots, \delta_N$ of the same size as θ_j, with i.i.d. standard normal entries.
5: 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 $k \in \{1, 2, \ldots, N\}$.
6: Make the update step:

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

7: $j \leftarrow j + 1$.
8: **end while**

Policy-based Reinforcement Learning

Better: 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**

Sample N different variations for the policy parameters (δ_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

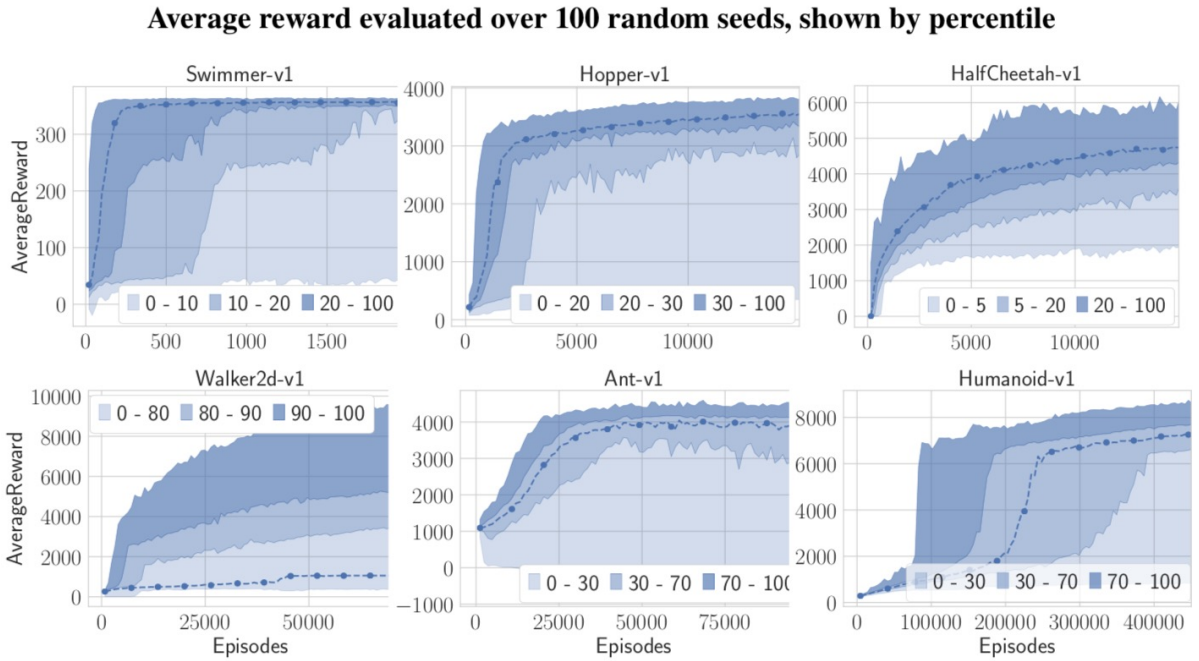
To avoid tuning the learning rate, scale the update by the standard deviation (σ_R) of the $2b$ returns used for the update

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

Policy-based Reinforcement Learning

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.

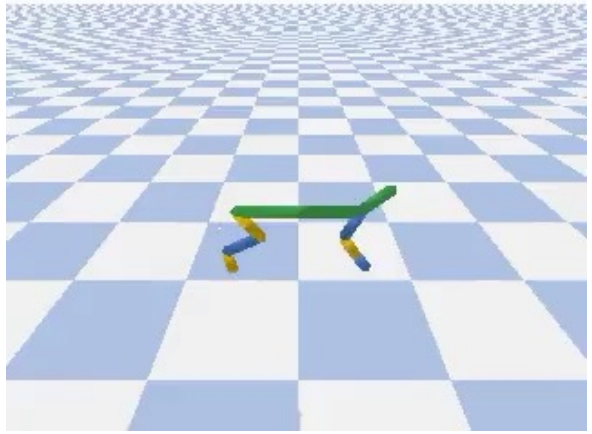
see also: https://towardsdatascience.com/introduction-to-augmented-random-search-d8d7b55309bd

Policy-based Reinforcement Learning

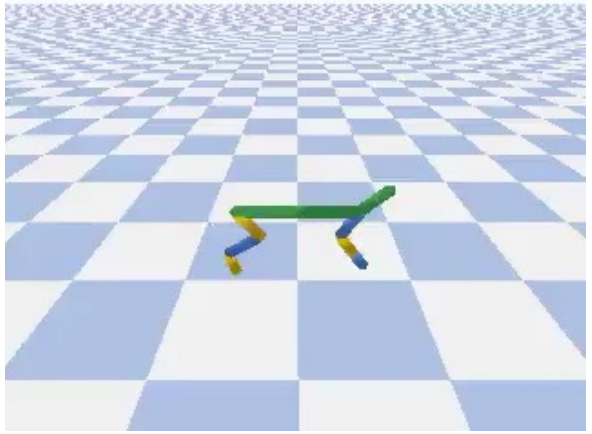
Better: Augmented Random Search (ARS)

- 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!

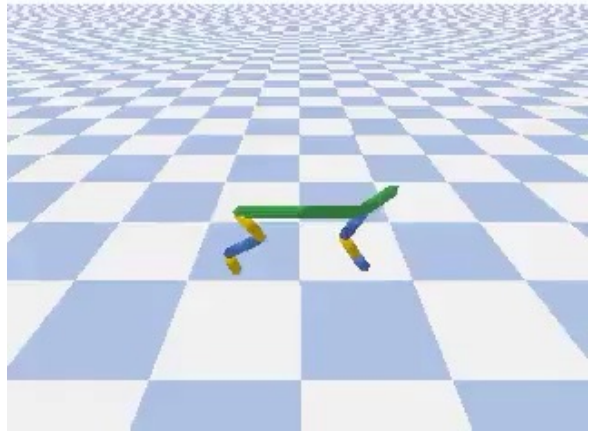
at the beginning…



after 100 iterations:



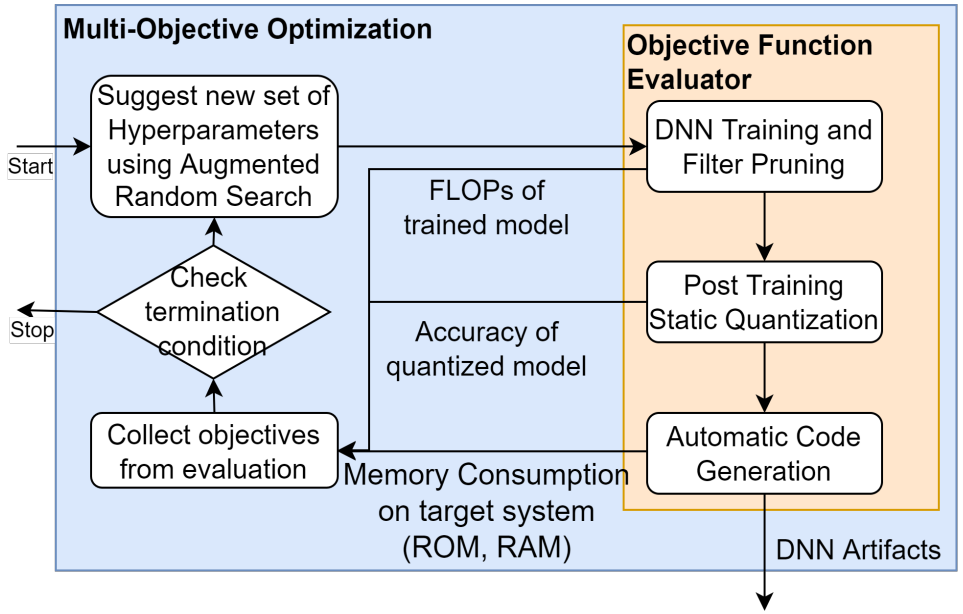
after 300 iterations:



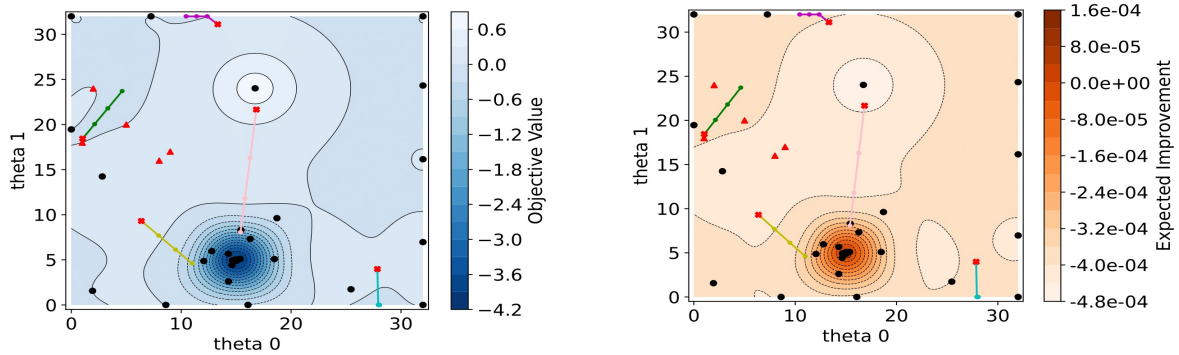
see also: https://towardsdatascience.com/introduction-to-augmented-random-search-d8d7b55309bd

ARS in Practice

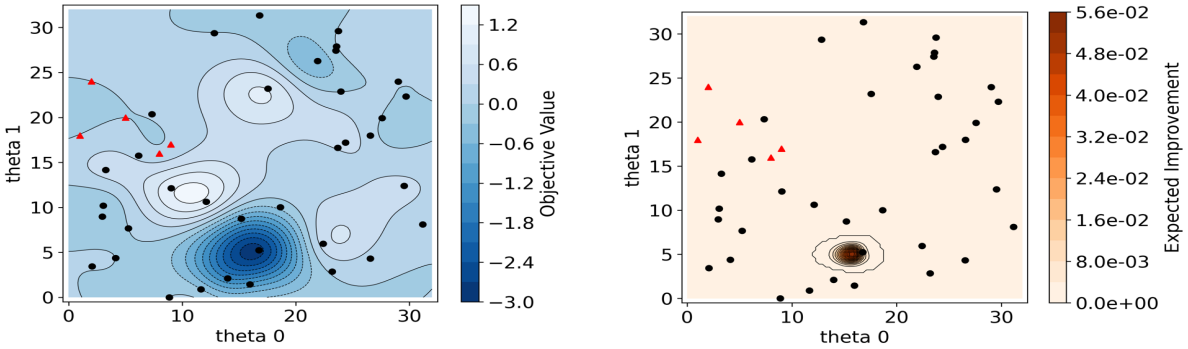
Augmented Random Search for Multi-Objective Bayesian Optimization of Neural Networks



MOBOpt-ARS:



ParEGO:



Ground Truth:

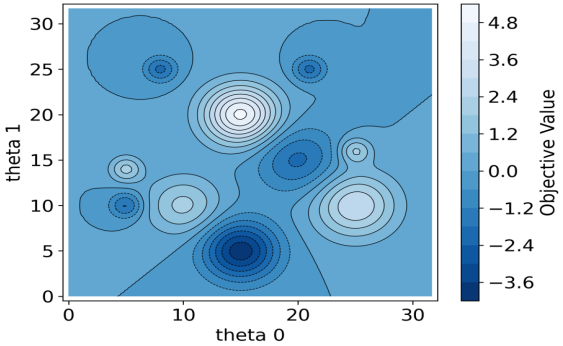


Figure 3. Topography of the objective value and the expected improvement for MOBOpt-ARS and ParEGO after a total of 40 samples, given an initial prior of 5 samples, marked as red triangles.
The global minimum can be found at $\theta_0 = 15$ and $\theta_1 = 5$. For MOBOpt-ARS, the rollouts for all competing trained policies are shown as lines with their starting points marked by red crosses.

Fraunhofer IIS

ARS in Practice

Augmented Random Search for Multi-Objective Bayesian Optimization of Neural Networks

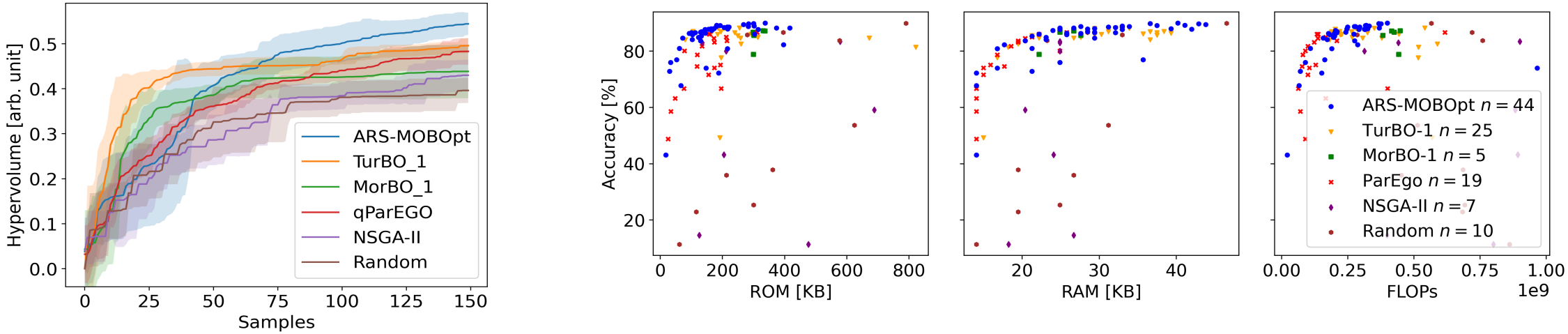


Figure 2. Hypervolume and feasible Pareto fronts of optimizations performed for two problems (ResNet18, CIFAR10 and MobileNetV3, DaLiAc) with a search budget of 150 samples and 5 seeds each.

Policy-based Reinforcement Learning

Better: Augmented Random Search (ARS)

Pros:

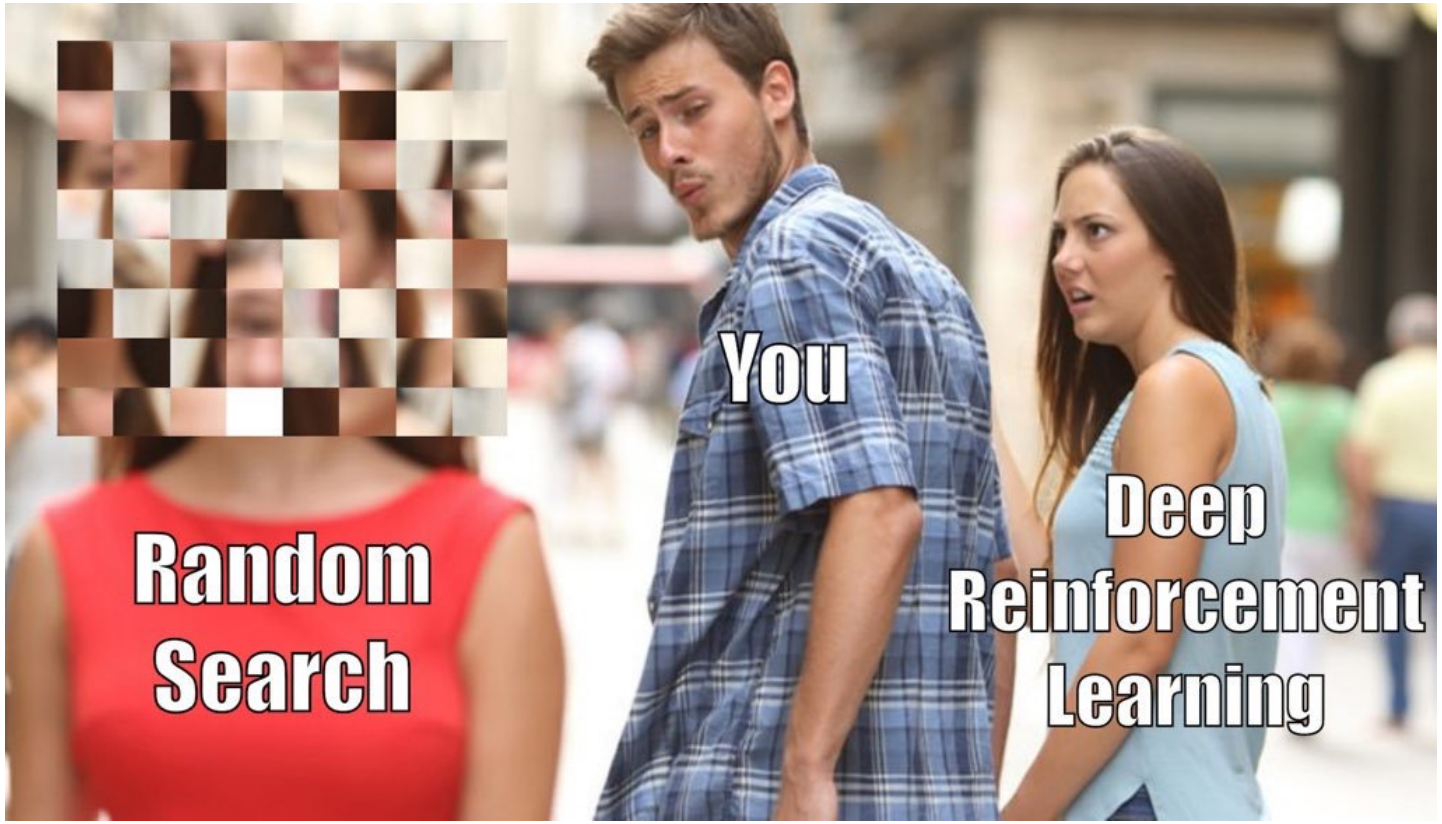
- Simple to understand and implement
- Less parameter tuning and robust to hyper-parameters
- Embarrassingly easy to parallelize

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

Policy-based Reinforcement Learning

But…wait…uh… What are we doing here?



Policy-based Reinforcement Learning

Policy Gradients

- Assume a state-action sequence in a complete trajectory with T steps (with s_T being a terminal state):

$$\tau = (s_0, a_0, \dots, s_{T-1}, a_{T-1}, s_T)$$

- As usual,
  - $R(s_t, a_t)$ is the reward received after observing s_t and performing action a_t
  - $G(\tau) \coloneqq \sum_{t=0}^{T-1} \gamma^t R(s_t, a_t)$ is the (discounted) sum of rewards (return)

- **Our goal is to maximize the expected reward:**

$$\max_{\theta} \mathbb{E}_{\pi_\theta} G(\tau)$$

(where π_θ is a parameterized policy, e.g., a neural network)

Policy-based Reinforcement Learning

Policy Gradients

- But how do we maximize this?

  → **Gradient Ascent!** *Suppose* we know how to calculate the gradient w.r.t. the parameters:

$$\nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta} G(\tau)$$

- Then we can update our parameters θ with a learning rate α in the direction of the gradient:

$$\theta \leftarrow \theta + \alpha \underline{\nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta} G(\tau)}$$

Policy Gradient

often in literature referred to as $\nabla_\theta J(\pi_\theta)$

Policy-based Reinforcement Learning

Policy Gradients

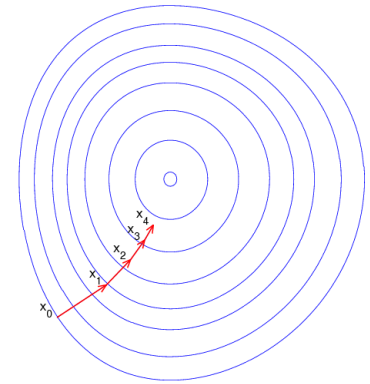
- Let $J(\theta)$ be any policy objective function
- Policy gradient algorithms search for a local maximum in $J(\theta)$ by ascending the gradient of the policy w.r.t. the parameters θ:

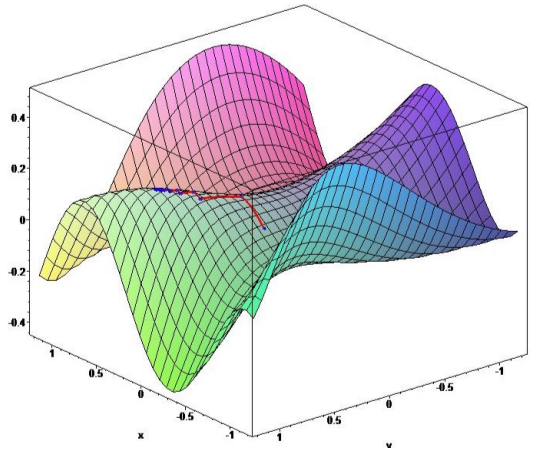
$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

where $\nabla_\theta J(\theta)$ is the **policy gradient**

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

and α is a step-size parameter.





David Silver. 2016.

Policy-based Reinforcement Learning

Policy Gradients

Let $P(\tau|\theta)$ be the probability of a trajectory τ under policy π_θ, then

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta} G(\tau) = \nabla_\theta \sum_\tau P(\tau|\theta) G(\tau)$$

definition of expectation

$$= \sum_\tau \nabla_\theta P(\tau|\theta) G(\tau)$$

swap sum/integral and gradient

$$= \sum_\tau \frac{P(\tau|\theta)}{P(\tau|\theta)} \nabla_\theta P(\tau|\theta) G(\tau)$$

multiply and divide by $P(\tau|\theta)$

$$= \sum_\tau P(\tau|\theta) \nabla_\theta \, log \, P(\tau|\theta) G(\tau)$$

$$\nabla_x \log f(x) = \frac{\nabla_x f(x)}{f(x)}$$

$$= \mathbb{E}_{\tau\sim\pi_\theta}(\nabla_\theta \, log \, P(\tau|\theta) G(\tau))$$

definition of expectation

https://www.janisklaise.com/post/rl-policy-gradients/

Policy-based Reinforcement Learning

Policy Gradients

- The probability of a trajectory τ can be formulated as (note: MDP):

$$P(\tau|\theta) = p(s_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

argh…we do not like this!

- But wait, let's take the gradient of the log-prob first:

$$\nabla_\theta \log P(\tau|\theta) = \nabla_\theta \left(\log p(s_0) + \sum_{t=0}^{T} (\log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)) \right)$$

$$= \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

dynamics model disappears!

Policy-based Reinforcement Learning

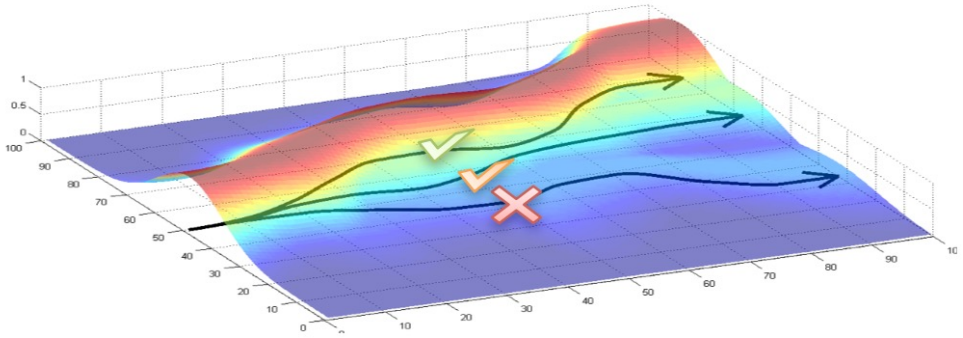
Policy Gradients

- Plugging grad-log-prob into the gradient update gives:

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta} G(\tau) = \mathbb{E}_{\tau\sim\pi_\theta}(\nabla_\theta \log P(\tau|\theta) G(\tau))$$
$$= \mathbb{E}_{\tau\sim\pi_\theta}\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)\right)$$

- But what is the intuition behind this gradient?

- The gradient tries to
  - Increase probability of paths with positive G
  - Decrease probability of paths with negative G



Pieter Abbeel. DeepRL Bootcamp 4A Policy Gradients.

Policy-based Reinforcement Learning

Policy Gradients

- Plugging grad-log-prob into the gradient update gives:

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta} G(\tau) = \mathbb{E}_{\tau\sim\pi_\theta}(\nabla_\theta \log P(\tau|\theta) G(\tau))$$
$$= \mathbb{E}_{\tau\sim\pi_\theta}\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)\right)$$

- As this is an expectation, we can estimate it with a sample mean using Monte-Carlo sampling of $|\tau| = L$ trajectories:

each action in the episode in influenced by the reward of the whole episode???
→ *reward-to-go* policy gradient

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta} G(\tau) \approx \frac{1}{L}\sum_{\tau}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)$$

Policy-based Reinforcement Learning

Policy Gradients

- Plugging grad-log-prob into the gradient update gives:

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta} G(\tau) = \mathbb{E}_{\tau\sim\pi_\theta}(\nabla_\theta \log P(\tau|\theta) G(\tau))$$
$$= \mathbb{E}_{\tau\sim\pi_\theta}\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)\right)$$

- **Reduce variance:** as this is an expectation, we can estimate it with a sample mean using Monte-Carlo sampling of L trajectories:

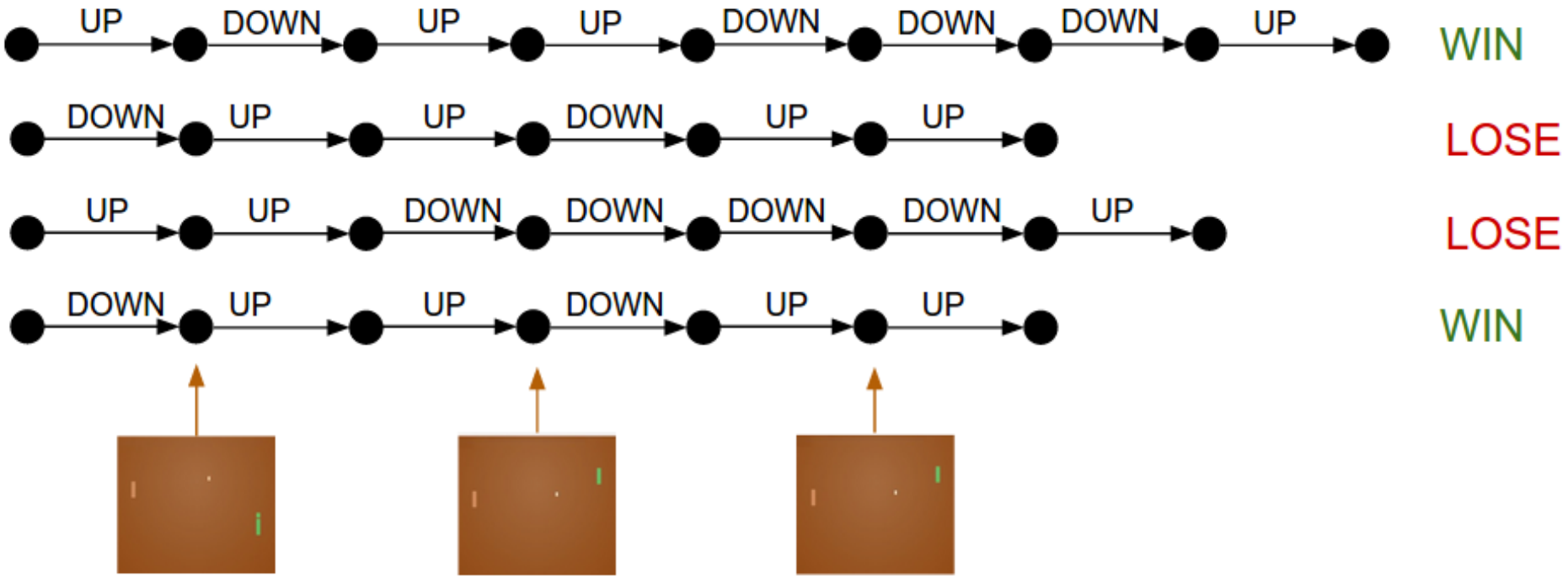
each action in the episode in influenced by the reward of the whole episode???
→ *reward-to-go* policy gradient

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta} G(\tau) \approx \frac{1}{L}\sum_{\tau}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)$$
$$\approx \frac{1}{L}\sum_{\tau}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} \gamma^{t'-t} R(s_{t'}, a_{t'})$$

Policy-based Reinforcement Learning

Policy Gradients

- How does the reward-to-go help?



http://karpathy.github.io/2016/05/31/rl/
Andrej Karpathy. DeepRL Bootcamp 4B

Policy-based Reinforcement Learning

Policy Gradient: REINFORCE

- Monte-Carle Policy Gradient Control
  - Update parameters by stochastic gradient ascent
  - Using policy gradient theorem
  - Using return-to-go $Q^{\pi_\theta}(s_t, a_t)$ as an unbiased sample of G:

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(a_t|s_t)\, \gamma G_t$$

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

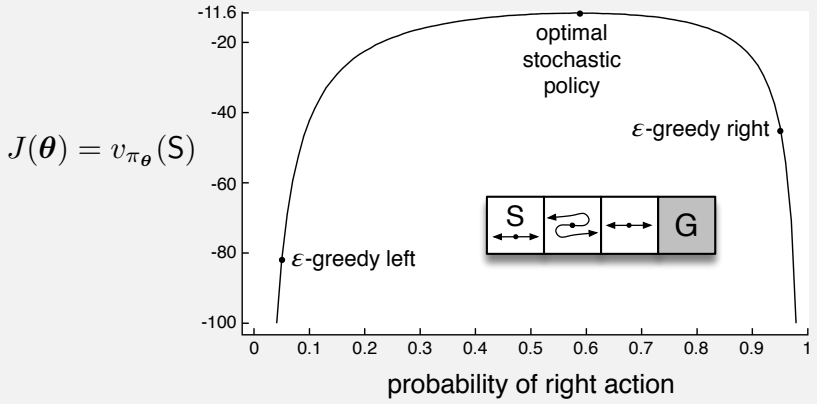
Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Policy-based Reinforcement Learning

Policy Gradient: REINFORCE

Example 13.1 Short corridor with switched actions

Consider the small corridor gridworld shown inset in the graph below. The reward is -1 per step, as usual. In each of the three nonterminal states there are only two actions, right and left. These actions have their usual consequences in the first and third states (left causes no movement in the first state), but in the second state they are reversed, so that right moves to the left and left moves to the right. The problem is difficult because all the states appear identical under the function approximation. In particular, we define $\mathbf{x}(s, \text{right}) = [1, 0]^\top$ and $\mathbf{x}(s, \text{left}) = [0, 1]^\top$, for all s. An action-value method with ε-greedy action selection is forced to choose between just two policies: choosing right with high probability $1 - \varepsilon/2$ on all steps or choosing left with the same high probability on all time steps. If $\varepsilon = 0.1$, then these two policies achieve a value (at the start state) of less than -44 and -82, respectively, as shown in the graph. A method can do significantly better if it can learn a specific probability with which to select right. The best probability is about 0.59, which achieves a value of about -11.6.



Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Policy-based Reinforcement Learning

Policy Gradient: REINFORCE

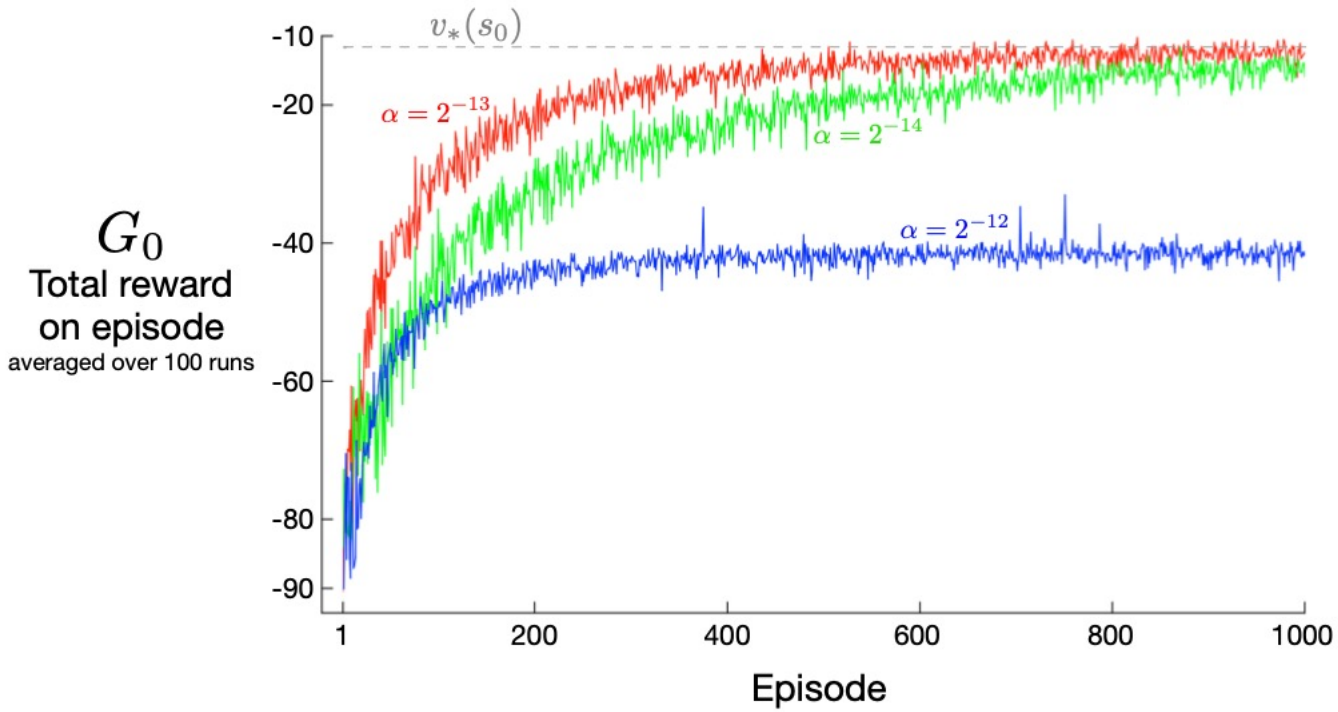


Figure 13.1: REINFORCE on the short-corridor gridworld (Example 13.1). With a good step size, the total reward per episode approaches the optimal value of the start state.

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Policy-based Reinforcement Learning

Policy Gradients

- Policy Gradient:
  - On-policy algorithm that also works w/ continuous action-spaces

```
Initialize a policy network randomly.
Repeat forever:
        collect a bunch of rollouts with the policy.
        Increase the probability of actions that worked well.
???
Profit.
```

\+ Good & easy-to-follow implementations available[1]
\+ Intuitive algorithm
\+ Easy to parallelize
\- High variance
\- Not capable of solving modern continuous action control problems

[1] *https://www.janisklaise.com/post/rl-policy-gradients*

Policy-based Reinforcement Learning

References

- Deisenroth, M. P., Neumann, G., & Peters, J. (2013). A survey on policy search for robotics. Foundations and Trends® in Robotics, 2(1–2), 1-142: https://spiral.imperial.ac.uk/bitstream/10044/1/12051/7/fnt_corrected_2014-8-22.pdf
- Sigaud, O., & Stulp, F. (2019). Policy search in continuous action domains: an overview. Neural Networks. ArXiv: https://arxiv.org/pdf/1803.04706.pdf
- Sutton, R. S., McAllester, D. A., Singh, S. P., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In Advances in neural information processing systems (pp. 1057-1063). Link: http://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf
- Kakade, S. M. (2002). A natural policy gradient. In Advances in neural information processing systems (pp. 1531-1538). Link: http://papers.nips.cc/paper/2073-a-natural-policy-gradient.pdf
- Duan, Y., Chen, X., Houthooft, R., Schulman, J., & Abbeel, P. (2016, June). Benchmarking deep reinforcement learning for continuous control. In International Conference on Machine Learning (pp. 1329-1338): http://proceedings.mlr.press/v48/duan16.pdf
- Riedmiller, M., Peters, J., & Schaal, S. (2007, April). Evaluation of policy gradient methods and variants on the cart-pole benchmark. In 2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (pp. 254-261). IEEE. link: http://is.tuebingen.mpg.de/fileadmin/user_upload/files/publications/ADPRL2007-Peters2_[0].pdf
- Kober, J., & Peters, J. R. (2009). Policy search for motor primitives in robotics. In Advances in neural information processing systems (pp. 849-856). Link: https://papers.nips.cc/paper/3545-policy-search-for-motor-primitives-in-robotics
- Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning, 8(3-4), 229-256.
- Vlassis, N., Toussaint, M., Kontes, G., & Piperidis, S. (2009). Learning model-free robot control by a Monte Carlo EM algorithm. Autonomous Robots, 27(2), 123-130.