Reinforcement Learning

Master IASD, Université PSL

https://www.di.ens.fr/olivier.cappe/Courses/IASD-FoRL/

October 2023

PSL UNIVERSITÉ PARIS

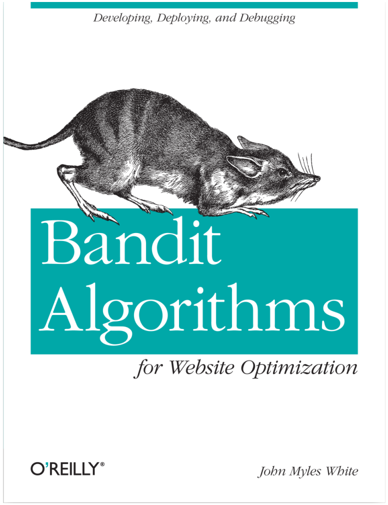
Table of Contents

Reinforcement learning

Compared to other learning paradigms, reinforcement learning (RL) is specific in that

- Observations are not available prior to learning but collected sequentially through successive actions (or decisions)
- Observations come with associated rewards that quantify the relevance of the sequence of actions
- The outcome of an action typically depends on the history of prior actions and observations

Some Noteworthy Applications of RL



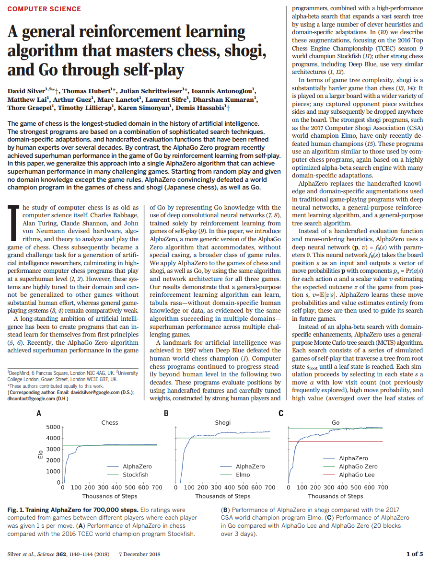
Developing, Deploying, and Debugging

Bandit Algorithms

for Website Optimization

O'REILLY®

John Myles White



COMPUTER SCIENCE

A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play

David Silver[1,2]*†, Thomas Hubert[1]*, Julian Schrittwieser[1]*, Ioannis Antonoglou[1], Matthew Lai[1], Arthur Guez[1], Marc Lanctot[1], Laurent Sifre[1], Dharshan Kumaran[1], Thore Graepel[1], Timothy Lillicrap[1], Karen Simonyan[1], Demis Hassabis[1]†

The game of chess is the longest-studied domain in the history of artificial intelligence. The strongest programs are based on a combination of sophisticated search techniques, domain-specific adaptations, and handcrafted evaluation functions that have been refined by human experts over several decades. By contrast, the AlphaGo Zero program recently achieved superhuman performance in the game of Go by reinforcement learning from self-play. In this paper, we generalize this approach into a single AlphaZero algorithm that can achieve superhuman performance in many challenging games. Starting from random play and given no domain knowledge except the game rules, AlphaZero convincingly defeated a world champion program in the games of chess and shogi (Japanese chess), as well as Go.

The study of computer chess is as old as computer science itself. Charles Babbage, Alan Turing, Claude Shannon, and John von Neumann devised hardware, algorithms, and theory to analyze and play the game of chess. Chess subsequently became a grand challenge task for a generation of artificial intelligence researchers, culminating in high-performance computer chess programs that play at a superhuman level (*1*, *2*). However, these systems are highly tuned to their domain and cannot be generalized to other games without substantial human effort, whereas general game-playing systems (*3*, *4*) remain comparatively weak.

A long-standing ambition of artificial intelligence has been to create programs that can instead learn for themselves from first principles (*5*, *6*). Recently, the AlphaGo Zero algorithm achieved superhuman performance in the game of Go by representing Go knowledge with the use of deep convolutional neural networks (*7*, *8*), trained solely by reinforcement learning from games of self-play (*9*). In this paper, we introduce AlphaZero, a more generic version of the AlphaGo Zero algorithm that accommodates, without special casing, a broader class of game rules. We apply AlphaZero to the games of chess and shogi, as well as Go, by using the same algorithm and network architecture for all three games. Our results demonstrate that a general-purpose reinforcement learning algorithm can learn, tabula rasa—without domain-specific human knowledge or data, as evidenced by the same algorithm succeeding in multiple domains—superhuman performance across multiple challenging games.

A landmark for artificial intelligence was achieved in 1997 when Deep Blue defeated the human world chess champion (*1*). Computer chess programs continued to progress steadily beyond human level in the following two decades. These programs evaluate positions by using handcrafted features and carefully tuned weights, constructed by strong human players and programmers, combined with a high-performance alpha-beta search that expands a vast search tree by using a large number of clever heuristics and domain-specific adaptations. In (*10*) we describe these augmentations, focusing on the 2016 Top Chess Engine Championship (TCEC) season 9 world champion Stockfish (*11*); other strong chess programs, including Deep Blue, use very similar architectures (*1*, *12*).

In terms of game tree complexity, shogi is a substantially harder game than chess (*13*, *14*): It is played on a larger board with a wider variety of pieces; any captured opponent piece switches sides and may subsequently be dropped anywhere on the board. The strongest shogi programs, such as the 2017 Computer Shogi Association (CSA) world champion Elmo, have only recently defeated human champions (*15*). These programs use an algorithm similar to those used by computer chess programs, again based on a highly optimized alpha-beta search engine with many domain-specific adaptations.

AlphaZero replaces the handcrafted knowledge and domain-specific augmentations used in traditional game-playing programs with deep neural networks, a general-purpose reinforcement learning algorithm, and a general-purpose tree search algorithm.

Instead of a handcrafted evaluation function and move-ordering heuristics, AlphaZero uses a deep neural network $(\mathbf{p}, v) = f_\theta(s)$ with parameters θ. This neural network $f_\theta(s)$ takes the board position s as an input and outputs a vector of move probabilities $\mathbf{p}$ with components $p_a = \Pr(a|s)$ for each action a and a scalar value v estimating the expected outcome z of the game from position s, $v \approx \mathbb{E}[z|s]$. AlphaZero learns these move probabilities and value estimates entirely from self-play; these are then used to guide its search in future games.

Instead of an alpha-beta search with domain-specific enhancements, AlphaZero uses a general-purpose Monte Carlo tree search (MCTS) algorithm. Each search consists of a series of simulated games of self-play that traverse a tree from root state s_{root} until a leaf state is reached. Each simulation proceeds by selecting in each state s a move a with low visit count (not previously frequently explored), high move probability, and high value (averaged over the leaf states of

[1]DeepMind, 6 Pancras Square, London N1C 4AG, UK. [2]University College London, Gower Street, London WC1E 6BT, UK.
*These authors contributed equally to this work.
†Corresponding author. Email: davidsilver@google.com (D.S.); dhcontact@google.com (D.H.)

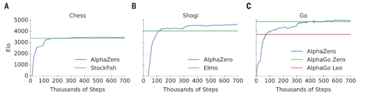


Fig. 1. Training AlphaZero for 700,000 steps. Elo ratings were computed from games between different players where each player was given 1 s per move. (**A**) Performance of AlphaZero in chess compared with the 2016 TCEC world champion program Stockfish. (**B**) Performance of AlphaZero in shogi compared with the 2017 CSA world champion program Elmo. (**C**) Performance of AlphaZero in Go compared with AlphaGo Lee and AlphaGo Zero (20 blocks over 3 days).

Silver *et al.*, *Science* **362**, 1140–1144 (2018) 7 December 2018 **1 of 5**

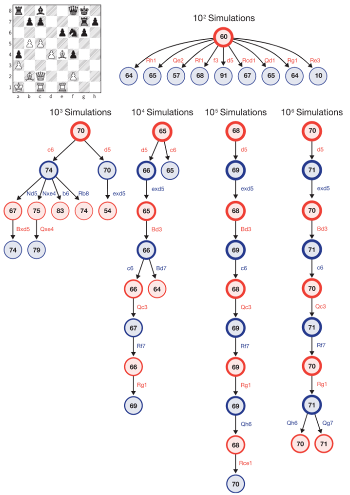


Fig. 4. AlphaZero's search procedure. The search is illustrated for a position (**inset**) from game 1 (table S6) between AlphaZero (white) and Stockfish (black) after 29. ... Qf8. The internal state of AlphaZero's MCTS is summarized after 10^2, ..., 10^6 simulations. Each summary shows the 10 most visited states. The estimated value is shown in each state, from white's perspective, scaled to the range [0, 100]. The visit count of each state, relative to the root state of that tree, is proportional to the thickness of the border circle. AlphaZero considers 30. c6 but eventually plays 30. d5.

Training language models to follow instructions with human feedback

Long Ouyang[*] Jeff Wu[*] Xu Jiang[*] Diogo Almeida[*] Carroll L. Wainwright[*]

Pamela Mishkin[*] Chong Zhang Sandhini Agarwal Katarina Slama Alex Ray

John Schulman Jacob Hilton Fraser Kelton Luke Miller Maddie Simens

Amanda Askell[†] Peter Welinder Paul Christiano[*†]

Jan Leike[*] Ryan Lowe[*]

OpenAI

Abstract

Making language models bigger does not inherently make them better at following a user's intent. For example, large language models can generate outputs that are untruthful, toxic, or simply not helpful to the user. In other words, these models are not *aligned* with their users. In this paper, we show an avenue for aligning language models with user intent on a wide range of tasks by fine-tuning with human feedback. Starting with a set of labeler-written prompts and prompts submitted through the OpenAI API, we collect a dataset of labeler demonstrations of the desired model behavior, which we use to fine-tune GPT-3 using supervised learning. We then collect a dataset of rankings of model outputs, which we use to further fine-tune this supervised model using reinforcement learning from human feedback. We call the resulting models *InstructGPT*. In human evaluations on our prompt distribution, outputs from the 1.3B parameter InstructGPT model are preferred to outputs from the 175B GPT-3, despite having 100x fewer parameters. Moreover, InstructGPT models show improvements in truthfulness and reductions in toxic output generation while having minimal performance regressions on public NLP datasets. Even though InstructGPT still makes simple mistakes, our results show that fine-tuning with human feedback is a promising direction for aligning language models with human intent.

1 Introduction

Large language models (LMs) can be "prompted" to perform a range of natural language processing (NLP) tasks, given some examples of the task as input. However, these models often express unintended behaviors such as making up facts, generating biased or toxic text, or simply not following user instructions (Bender et al., 2021; Bommasani et al., 2021; Kenton et al., 2021; Weidinger et al., 2021; Tamkin et al., 2021; Gehman et al., 2020). This is because the language modeling objective

[*]Primary authors. This was a joint project of the OpenAI Alignment team. RL and JL are the team leads. Corresponding author: lowe@openai.com.
[†]Work done while at OpenAI. Current affiliations: AA: Anthropic; PC: Alignment Research Center.

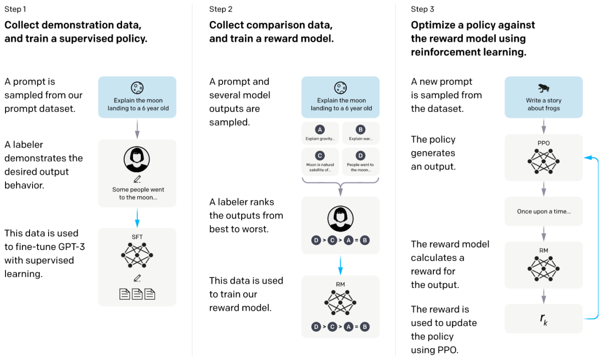
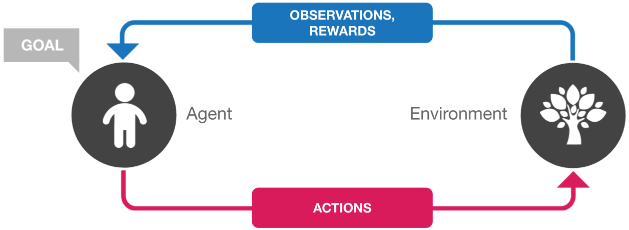

Step 1
Collect demonstration data, and train a supervised policy.
A prompt is sampled from our prompt dataset.
Explain the moon landing to a 6 year old
A labeler demonstrates the desired output behavior.
Some people went to the moon...
This data is used to fine-tune GPT-3 with supervised learning.
SFT
Step 2
Collect comparison data, and train a reward model.
A prompt and several model outputs are sampled.
Explain the moon landing to a 6 year old
A Explain gravity...
B Explain war...
C Moon is natural satellite of...
D People went to the moon...
A labeler ranks the outputs from best to worst.
D > C > A = B
This data is used to train our reward model.
RM
D > C > A = B
Step 3
Optimize a policy against the reward model using reinforcement learning.
A new prompt is sampled from the dataset.
Write a story about frogs
The policy generates an output.
PPO
Once upon a time...
The reward model calculates a reward for the output.
RM
The reward is used to update the policy using PPO.
r_k

Figure 2: A diagram illustrating the three steps of our method: (1) supervised fine-tuning (SFT), (2) reward model (RM) training, and (3) reinforcement learning via proximal policy optimization (PPO)

The Agent Playing Against a Partially Unknown Environment

For $t = 1, \dots$

In picture



In code

```
for _ in range(1000):
    action = policy(observation)  # User-defined policy function
    observation, reward, terminated, truncated, info = env.step(action)
```

Gym/Gymnasium API `https://gymnasium.farama.org/`

Model of the Environment

RL requires assumptions on the environment

- Deterministic Environment $\longrightarrow$ Search and constraint satisfaction
- Stochastic Environment $\longrightarrow$ RL (this course)
- Adversarial Environment $\longrightarrow$ Game theory

RL combines elements from

- Statistics, Learning to estimate unknown parameters of the environment (in "model-based RL") or to determine action rules (in "model-free RL")
- Planning, Control to optimize sequence actions, when assuming that the environment parameters are known

(Discrete) Markov Chain

A sequence of random variables $(S_t)_{t \geq 0} \in \mathscr{S}$ such that for any $t \geq 0$ and $s \in \mathscr{S}$

$$\mathbb{P}[S_{t+1} = s | H_t] = \mathbb{P}[S_{t+1} = s | S_t]$$

where $H_t = (S_0, \ldots, S_t)$ denotes the history of the process up to time t ⚠

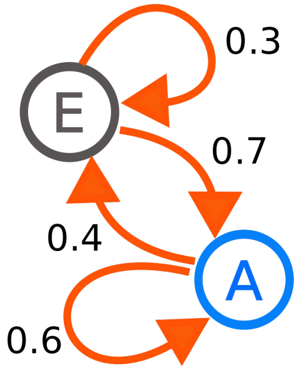
When the chain is time-homogeneous, its distribution is fully determined by

1. The initial distribution $(\mathbb{P}(S_0 = s))_{s \in \mathscr{S}}$
2. The transition probabilities $\big(p(s, s')\big)_{(s,s') \in \mathscr{S}^2}$ where

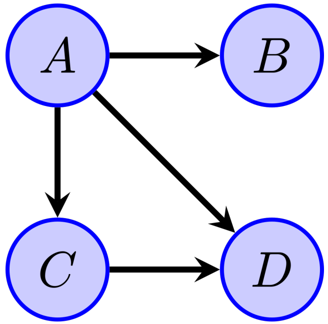
$$p(s, s') = \mathbb{P}(S_{t+1} = s' | S_t = s)$$

Graphical Representation

Stochastic Automaton Representation



Not to be confused with Bayesian Network (Graphical Model) representation



Transition Matrix

When $\mathscr{S}$ is finite, say $\mathscr{S} = \{1, \ldots, k\}$, it convenient to store the transition probabilities in a transition matrix

$$P = \begin{pmatrix} p(1,1) & \cdots & p(1,k) \\ \vdots & & \vdots \\ p(k,1) & \cdots & p(k,k) \end{pmatrix}$$

where it is easily checked that

$$\mathbb{P}(S_{t+i} = s' | S_t = s) = \left(P^i\right)_{s,s'}$$

Example: (Three States) River Swim [Strehl & Litman, 2008]

$$\begin{tikzcd}
1 \arrow[loop above, "0.4"] \arrow[r, bend left, "0.6"] & 2 \arrow[loop above, "0.6"] \arrow[l, bend left, "0.05"] \arrow[r, bend left, "0.35"] & 3 \arrow[loop above, "0.6"] \arrow[l, bend left, "0.4"]
\end{tikzcd}$$

$$P = \begin{pmatrix} 0.4 & 0.6 & 0 \\ 0.05 & 0.6 & 0.35 \\ 0 & 0.4 & 0.6 \end{pmatrix}$$

And it is easily checked numerically that

$$P^i \to \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \underbrace{\begin{pmatrix} 0.042 & 0.511 & 0.447 \end{pmatrix}}_{\text{stationary distribution}}$$

Markov Reward Process

A joint process $(S_t, X_t)_{t\geq 0}$, where $S_t \in \mathcal{S}$ is a discrete state and $X_t \in \mathbb{R}$ is a reward such that, given the history $H_t = (S_0, X_0), \ldots, (S_{t-1}, X_{t-1}), S_t$

- X_t and S_{t+1} are conditionally independent
- their conditional distributions depend only on S_t

Notations

- $p(s, s') = \mathbb{P}(S_{t+1} = s' | S_t = s)$ are the state transition probabilities
- as we will mostly consider expected rewards[*], we introduce a notation for the (expected) reward function:

$$r(s) = \mathbb{E}[X_t | S_t = s]$$

[*] "Distributional RL" refers to approaches that address other features of the reward distribution

Value Function

For a sequence $w = (w_t)_{t\geq 0}$ of summable weights, the value function is defined as

$$v_w(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} w_t X_t \middle| S_0 = s \right]$$

The value function measures the expected weighted sum of rewards for each possible starting state s. It is easily checked that

- for any t, $v_w(s) = \mathbb{E}\left[\sum_{i=0}^{\infty} w_{t+i} X_{t+i} \middle| S_t = s \right]$
- $$v_w(s) = \left(\sum_{i=0}^{\infty} w_i P^i r \right)_s \qquad \text{where} \quad r = \begin{pmatrix} r(1) \\ \vdots \\ r(k) \end{pmatrix}$$

Bellman Equation

Particular cases of interest are

- Finite Horizon $w_0 = w_1 = \ldots w_n = 1$ and $w_i = 0$ when $i > n$
- Discounted Rewards $w_i = \gamma^i$ with $\gamma \in (0,1)$, for which, one has the following Bellman equation

$$v_\gamma = \sum_{i=0}^{\infty} \gamma^i P^i r = r + \gamma P v_\gamma$$

and one may also determine v_γ as

$$v_\gamma = (I - \gamma P)^{-1} r$$

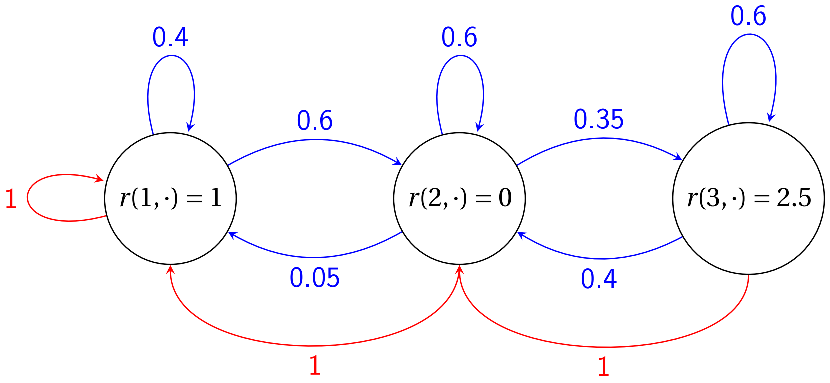
The standard RL model is that of Markov reward model whose transition probabilities and reward function are controlled by the agent's actions

Markov Decision Process

A joint process $(S_t, A_t, X_t)_{t\geq 0}$, with states $S_t \in \mathscr{S}$, actions $A_t \in \mathscr{A}$ and rewards $X_t \in \mathbb{R}$ such that given the history $H_t = (S_0, A_0, X_0), \ldots, (S_{t-1}, A_{t-1}, X_{t-1}), S_t$

- The agent may chose A_t as a function of H_t and, possibly, of an external independent randomization
- Given H_t and A_t, the environment generates X_t and S_{t+1} conditionally independent such that
  - $p(s, a, s') = \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a)$ (action-dependent transition probabilities)
  - $r(s, a) = \mathbb{E}[X_t | S_t = s, A_t = a]$ (action-dependent reward function)

Example: Three States River Swim Continued



$p(\cdot|\cdot,1)$ and $p(\cdot|\cdot,2)$

More Useful Example: Retail Store Management

You owe a bike store. During week t, the demand is D_t units, which we may assume to be $\text{Poisson}(d)$ distributed, independently of the past.

On Monday morning you may choose to command A_t additional units: they are delivered immediately before the shop opens.

For each week

- Maintenance Cost h per unit left in your stock from previous week
- Ordering Cost c per ordered unit
- Sales Profit f per sold unit

Constraints

- Your warehouse has a maximal capacity of m unit (any additional bike gets stolen)
- You cannot sell bikes that you don't have in stock

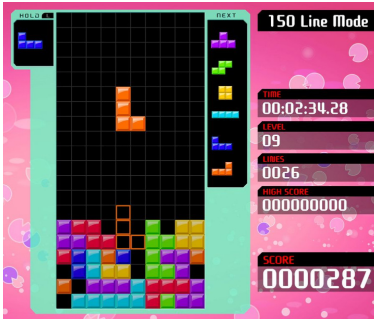
Can you define S_t and X_t such that (S_t, A_t, X_t) is an MDP? Can you simulate the outcome of fixed order policies (i.e. $A_t = a$)?

Episodic tasks

In many applications, it is natural to terminate based on the outcome of a random stopping time T and one would like to consider

$$\mathbb{E}\left[\sum_{t=0}^{T} X_t\right]$$

as a a measure of performance, which is however a more complicated object ⚠



When $\tau = \inf\{t \geq 0 : S_t \in \mathscr{S}_0\}$, this is usually handled by adding a terminal state with zero reward or that deterministically loops to a designated start state

Variants

Structural Variants

- Rewards on Transitions X_{t+1} depends both on S_t, A_t and S_{t+1} (e.g., in Sutton and Barto's book), also usually imply that time indices are shifted for rewards (reward sequence X_t starts at time $t = 1$)
- Additional Observations Choosing action A_t in state S_t not only returns the reward X_t but also some additional observations
- Deterministic rewards (i.e., $X_t = r(S_t, A_t)$) and/or transitions (i.e., $p(s, a, s') = 1$ for some s')

Notation Variants

- t or T (for transition) instead of p
- Conditional ($p(s'|s, a)$) instead of kernel ($p(s, a, s')$) notation (Mnemonic: p and π are kernels and sum to 1 w.r.t. their last argument)
- Sub/super-script notation, e.g., $P^a_{s,s'}$ instead of $p(s, a, s')$
- ... (including upper/lower case variants)

Policy

Although a RL algorithm will typically select actions that depend on the whole history H_t, we will first focus on the case where the parameters of the environment are known and consider only state-dependent stochastic (or randomized) policies such that

$$\mathbb{P}_\pi [A_t = a | H_t] = \pi_t(S_t, a)$$

where $\pi_t(s, a) = \mathbb{P}_\pi(A_t = a | S_t = s)$

Note that while the randomness in $\mathbb{P}$ comes from the environment and cannot be changed, the choice of the policy π is done by the agent (who also generates the randomization used to produce A_t), hence the notation $\mathbb{P}_\pi$

- When $\pi_t(s, a) = \pi(s, a)$ we says that the policy is time-homogeneous
- When $\pi(s, a_s) = 1$, for some a_s such that $a_s = \pi(s)$, the policy is said to be deterministic

⚠

Markov Reward Process Induced by a Policy

When choosing a static randomized policy, the MDP becomes a Markov Reward Process such that

$$\mathbb{E}_\pi[R_t|H_t] = r_\pi(S_t) = \sum_{a\in\mathcal{A}} \pi(S_t,a) r(S_t,a)$$

$$\mathbb{P}_\pi[S_{t+1}=s|H_t] = p_\pi(S_t,s) = \sum_{a\in\mathcal{A}} \pi(S_t,a) p(S_t,a,s)$$

Hence, its γ–discounted value function is given by

$$v_\pi = \left(\sum_{i=0}^{\infty} \gamma^i P_\pi^i r_\pi\right) = r_\pi + \gamma P_\pi v_\pi$$

⚠

and satisfies the Bellman equation

$$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(s,a)\left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s,a,s') v_\pi(s')\right)$$

Determining the Value Function

To compute v_π, one can use matrix inversion, i.e.,

$$v_\pi = (I - \gamma P_\pi)^{-1} r_\pi$$

or use and iterative algorithm

Iterative Policy Evaluation

Initializing by an arbitrary v_0 and running iteratively

$$v_{i+1}(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s,a,s') v_i(s') \right)$$

produces a sequence of iterates that converge to v_π with $\| v_i - v_\pi \|_\infty \leq \gamma^i \| v_0 - v_\pi \|_\infty$

One also has $\| v_i - v_\pi \|_\infty \leq \gamma / (1-\gamma) \| v_i - v_{i-1} \|_\infty$ which may be used as a stopping criterion

Table of Contents

Fixed-Horizon Bellman Optimality Equations

A sequence of deterministic policies $\pi_0^n, \dots \pi_n^n$ that maximize the fixed horizon reward $\mathbb{E}_\pi[\sum_{t=0}^n X_t]$ may be found using the following backward recursion

Dynamic Programming Algorithm

Initialization

$$v_n^n(s) = \max_{a \in \mathscr{A}} r(s,a)$$

$$\pi_n^n(s) \in \arg\max_{a \in \mathscr{A}} r(s,a)$$

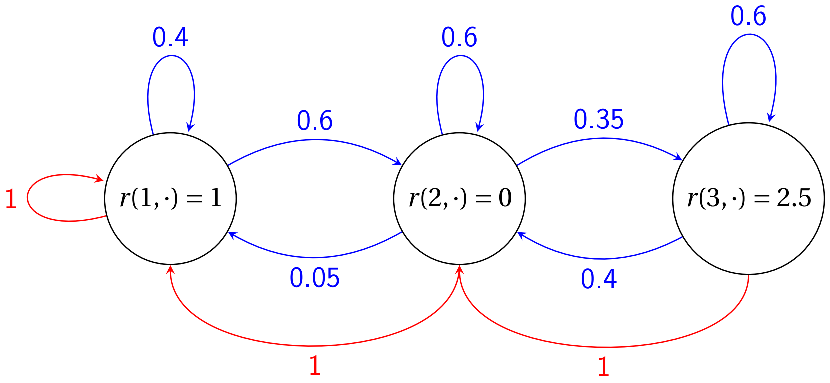
For $t = n-1, \dots, 0$

$$v_t^n(s) = \max_{a \in \mathscr{A}} \left(r(s,a) + \sum_{s' \in \mathscr{S}} p(s,a,s') v_{t+1}^n(s') \right)$$

$$\pi_t^n(s) \in \arg\max_{a \in \mathscr{A}} \left(r(s,a) + \sum_{s' \in \mathscr{S}} p(s,a,s') v_{t+1}^n(s') \right)$$

Example

Let's try this on the three states river-swim example...



We now consider maximizing $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t X_t]$, with $\gamma \in (0,1)$

Bellman Optimality Equation

Let $v_\star$ be the solution to

$$v_\star(s) = \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s,a,s') v_\star(s') \right)$$

$v_\star$ is the optimal value function in the sense that

1. it is unique
2. $v_\star(s) \geq \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t X_t | S_0 = s]$ for any sequence of, possibly time-dependent, policies $(\pi_t)_{t \geq 0}$
3. it can be achieved by a (time-homogeneous) deterministic policy such that

$$\pi_\star(s) \in \arg\max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s,a,s') v_\star(s') \right)$$

Properties of Bellman Operators

Bellman Operator Associated With a Policy

Let $B_\pi : v \mapsto v'$ defined by

$$v'(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s,a,s') v(s') \right)$$

- It is an affine mapping
- We have already seen that it a γ-contraction (in $\|\cdot\|_\infty$ norm)
- It is also isotonic: if $v_1 \succeq v_2$ (component-wise), then $B_\pi(v_1) \succeq B_\pi(v_2)$

Interpretation: $B_{\pi'}(v_\pi)$ is the value function resulting from choosing A_0 according to π' and subsequent actions $A_1, A_2, \ldots$ from π

Bellman Improvement Operator

Let $B_+ : v \mapsto v'$ befined by

$$v'(s) = \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s,a,s') v(s') \right)$$

- It is a γ-contraction (in $\|\cdot\|_\infty$ norm)
- It is isotonic: $(v_1 \succeq v_2 \Rightarrow B_+(v_1) \succeq B_+(v_2))$
- It satisfies $B_+(v) \succeq B_\pi(v)$ (for all π and v)

B_+ is not an affine operator (contrary to B_π), it is continous but usually not differentiable (because of the max)

Policy Improvement

Greedy Policy

Let $g_+ : v \mapsto \pi'$, where π' is a deterministic policy such that

$$\pi'(s) \in \operatorname*{arg\,max}_{a \in \mathscr{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathscr{S}} p(s,a,s')\, v \right)$$

such that $B_+(v) = B_{g_+(v)}(v)$ ⚠

$B_+(v_\pi)$ is the value function resulting from choosing A_0 according to $g_+(v_\pi)$ and subsequent actions $A_1, A_2, \ldots$ from π

Policy Improvement Lemma

$$v_{g_+(v_\pi)} \succeq v_\pi \text{ and } v_{g_+(v_\pi)} = v_\pi \Rightarrow v_\pi = v_\star$$

Value and Policy Iteration

Value Iteration

Starting with an arbitrary v_0 and iterating

$$v_{i+1} = B_+(v_i)$$

until $\|v_{i+1} - v_i\|_\infty \le (1-\gamma)/\gamma\epsilon$ yields an ϵ-approximation of $v_\star$

Policy Iteration

Starting with an arbitrary π_0 and iterating

$$\pi_{i+1} = g_+(v_{\pi_i})$$

returns $\pi_\star$ in, at most, $|\mathscr{A}|^{|\mathscr{S}|}$ iterations

Each iteration requires $O(|\mathscr{A}| \times |\mathscr{S}|^2)$ operations, as well as computation of v_{π_i} for policy iteration

Table of Contents

Estimating the Model Parameters is (Usually) Not the Way to Go

We consider learning scenarios where the model parameters (r and p) are unknown and one wants to learn good policies by observation of the behavior of the system

A first easy approach is model-based Monte Carlo learning in which we observe m independent trajectories $(S_t^i, A_t^i, X_t^i)_{t\geq 0}^{i=1,\dots,m}$ of the MDP under a logging policy π. By the law of large number and Markov property

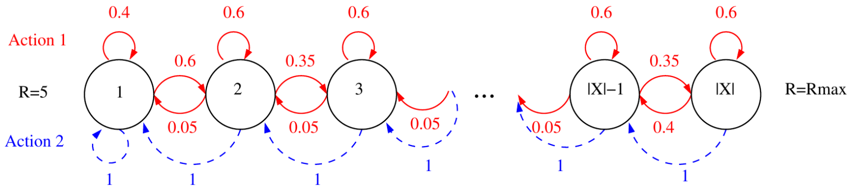
$$\frac{\sum_{i=1}^m \sum_{t=0}^n X_t^i \mathbb{1}\{S_t^i = s, A_t^i = a\}}{\sum_{i=1}^m \sum_{t=0}^n \mathbb{1}\{S_t^i = s, A_t^i = a\}} \text{ and } \frac{\sum_{i=1}^m \sum_{t=0}^n \mathbb{1}\{S_t^i = s, A_t^i = a, S_{t+1}^i = s'\}}{\sum_{i=1}^m \sum_{t=0}^n \mathbb{1}\{S_t^i = s, A_t^i = a\}}$$

are consistent estimators of $r(s,a)$ and $p(s,a,s')$ respectively (when $m \to \infty$)*, as soon as π is such that $\sum_{t=0}^n \mathbb{P}_\pi(S_t^i = s, A_t^i = a) > 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$ ✎

*Asymptotic behavior when $n \to \infty$ depend on the properties of P_π

But

- Estimating $|\mathscr{S}|^2 \times |\mathscr{A}|$ parameters can be very long
- Parameters for which $\sum_{t=0}^{n} \mathbb{P}_\pi(S_t^i = s, A_t^i = a)$ is very small will be poorly estimated
- The frequency with which a policy visits some regions of the state–action space typically decreases exponentially fast in the size of the state–action space



E.g. in (Bigger) River Swim Environment

- The corresponding plug-in estimators (e.g. of $v_\star$) may be unreliable

Estimating the Value Function Is Not Sufficient

When considering γ-discounting, estimating directly $v_\pi(s)$ can be done by*

$$\frac{\sum_{i=1}^{m}\sum_{t=0}^{n}\left(\sum_{r\geq t}\gamma^{r-t}X_r\right)\mathbb{1}\{S_t^i = s\}}{\sum_{i=1}^{m}\sum_{t=0}^{n}\mathbb{1}\{S_t^i = s\}}$$

However, estimating v_π is not sufficient for determining ways of improving π (as $g_+(\pi)$ for instance also depends on the unknown model parameters)

*In practice a small bias in the numerator is unavoidable due to the truncation effect when observing finite-length trajectories

State–Action Value Functions

State–Action (or Q) Value Function

Let $q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s,a,s') v_\pi(s')$ so that $v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(s,a) q_\pi(s,a)$. The Bellman equation may be written as

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s,a,s') \underbrace{\sum_{a' \in \mathcal{A}} \pi(s',a') q_\pi(s',a')}_{v_\pi(s')}$$

Optimal State-Action Value Function

$q_\star(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s,a,s') v_\star(s')$ which is such that $v_\star(s) = \max_{a \in \mathcal{A}} q_\star(s,a)$ satisfies the Bellman optimality equation:

$$q_\star(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s,a,s') \underbrace{\max_{a' \in \mathcal{A}} q_\star(s',a')}_{v_\star(s')}$$

Like previously, if one define the operators

- $T_\pi : q \mapsto q'$ such that

$$q'(s,a) = r(s,a) + \gamma \sum_{s' \in \mathscr{S}} p(s,a,s') \sum_{a' \in \mathscr{A}} \pi(s',a') q(s',a')$$

- $T_+ : q \mapsto q'$ where

$$q'(s,a) = r(s,a) + \gamma \sum_{s' \in \mathscr{S}} p(s,a,s') \max_{a' \in \mathscr{A}} q_\star(s',a')$$

- Both T_π and T_+ are isotonic γ–contractions
- $T_+(q) \succeq T_\pi(q)$
- q_π and $q_\star$ are the unique solutions to

$$q_\pi = T_\pi(q_\pi)$$
$$q_\star = T_+(q_\star)$$

State–Action Value Functions May Be Used To Improve the Policy

- The greedy policy w.r.t. q_π

$$\arg\max_{a\in\mathscr{A}} q_\pi(s,a) = \arg\max_{a\in\mathscr{A}} \left(r(s,a) + \gamma \sum_{s'\in\mathscr{S}} p(s,a,s')\, v_\pi(s') \right) \ni g_+(v_\pi)$$

  which improves over π (Policy Improvement Lemma)

- Likewise, $\pi_\star(s) \in \arg\max_{a\in\mathscr{A}} q_\star(s,a)$ is an optimal policy

- Most RL algorithms use estimates of q_π or $q_\star$ $(|\mathscr{S}| \times |\mathscr{A}|$ parameters)

Monte Carlo Policy Improvement ("Hello World" RL)

Given m independent trajectories $(S_t^i, A_t^i, X_t^i)_{t\geq 0}^{i=1,\dots,m}$ of the MDP under policy π,

1. Estimate

$$\widehat{q}_\pi(s,a) = \frac{\sum_{i=1}^m \sum_{t=0}^n \left(\sum_{j\geq t} \gamma^{j-t} X_j\right) \mathbb{1}\{S_t^i = s, A_t^i = a\}}{\sum_{i=1}^m \sum_{t\geq 0}^n \mathbb{1}\{S_t^i = s, A_t^i = a\}}$$

2. Update the policy to

$$\pi_+(s) \in \operatorname*{arg\,max}_{a\in\mathscr{A}} \widehat{q}_\pi(s,a)$$

Works but cannot be iterated, as π_+ does not try anymore all possible actions!

Typically one would instead choose

- ϵ–greedy policy $\pi_+(s,a) = 1-\epsilon+\frac{\epsilon}{|\mathscr{A}|}$ for $a = \operatorname{arg\,max}_{a'\in\mathscr{A}} \widehat{q}_\pi(s,a')$ and $\pi_+(s,a) = \frac{\epsilon}{|\mathscr{A}|}$ otherwise (assuming unique maxima)
- Boltzmann (softmax) policy

$$\pi_+(s,a) = \frac{\exp(\beta\widehat{q}_\pi(s,a))}{\sum_{a'\in\mathscr{A}} \exp(\beta\widehat{q}_\pi(s,a'))}$$

with low ϵ or large β

Temporal Difference (TD)[*] learning algorithms use a common recursive updating principle

Q-Learning [Watkins, 1989]

Q-Learning performs off-policy learning of the optimal Q-Value by behaving according to π and recursively updating an estimate Q_t of $q_\star$

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha_t \left[X_t + \gamma \max_a Q_t(S_{t+1}, a) - Q_t(S_t, A_t) \right]$$

and $Q_{t+1}(s,a) = Q_t(s,a)$ for all other state–action pairs

SARSA

SARSA performs on-policy learning of the Q-Value by behaving according to π and recursively updating an estimate Q_t of q_π

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha_t \left[X_t + \gamma Q_t(S_{t+1}, A_{t+1}) - Q_t(S_t, A_t) \right]$$

and $Q_{t+1}(s,a) = Q_t(s,a)$ for all other state–action pairs

[*] More precisely TD(0) algorithms

These updating rules are based on the observation that

For Q-Learning

$$\mathbb{E}_\pi \left[Q_{t+1}(S_t, A_t) | H_t, A_t \right] = (1 - \alpha_t) Q_t(S_t, A_t) + \alpha_t \left(T_+(Q_t) \right) (S_t, A_t)$$

For SARSA

$$\mathbb{E}_\pi \left[Q_{t+1}(S_t, A_t) | H_t, A_t \right] = (1 - \alpha_t) Q_t(S_t, A_t) + \alpha_t \left(T_\pi(Q_t) \right) (S_t, A_t)$$

✎

Stochastic Approximation

More generally,

Stochastic Approximation (a.k.a. Robbins–Monro) Algorithm

$$Q_{t+1} = Q_t + \alpha_t \left[T(Q_t) - Q_t + \epsilon_{t+1} \right] = (1 - \alpha_t) Q_t + \alpha_t \left[T(Q_t) + \epsilon_{t+1} \right]$$

with $\mathbb{E}[\epsilon_{t+1} | H_t] = 0$

is a general purpose scheme for finding the root of the equation of $q = T(q)$.

In the particular case where $T(q) - q$ may be interpreted as the gradient ∇f of a function f, one recovers the stochastic gradient algorithm (for *maximizing* f). This is not however the case for TD learning algorithms

Convergence of SA

[Bertsekas & Tsitsiklis, 1996] study the SA scheme under the assumptions required for Q-learning and other TD algorithms

[Bertsekas & Tsitsiklis, 1996] Proposition 4.4 (Simplified)

Assuming

- $\|T(q) - T(q')\|_\infty \leq \gamma \|q - q'\|_\infty$, with $\gamma < 1$
- $\mathbb{E}[\epsilon_{t+1} | H_t] = 0$, $\mathbb{E}[\|\epsilon_{t+1}\|_\infty^2 | H_t] \leq A + B\|Q_t\|_\infty^2$
- $\sum_t \alpha_t = \infty$, $\sum_t \alpha_t^2 < \infty$

implies that $Q_t \to q_*$ a. s., where q_* is the unique solution to $q_* = T(q_*)$

For actual application to Q-learning, one still needs to show that if the exploration policy π is such that each state–action pair (s, a) is visited infinitely often, it implies that Q_t (produced by Q-Learning) converges (a.s.) to $q_\star$

Convergence of SA $\longrightarrow$ L^2 result, written in the scalar case (for simplicity)

$$Q_{t+1} - q_* = (1-\alpha_t)(Q_t - q_*) + \alpha_t[T(Q_t) - q_*] + \alpha_t \varepsilon_{t+1}$$

$$\mathbb{E}\left[(Q_{t+1}-q_*)^2 \mid \mathcal{H}_t\right] = \underbrace{(1-\alpha_t)^2(Q_t-q_*)^2 + \alpha_t^2(T(Q_t)-q_*)^2 + 2\alpha_t(1-\alpha_t)(Q_t-q_*)(T(Q_t)-q_*)}_{\leq \left[(1-\alpha_t)^2 + \gamma^2\alpha_t^2 + 2\alpha_t(1-\alpha_t)\gamma\right](Q_t-q_*)^2} + \alpha_t^2 \underbrace{\mathbb{E}\left[\varepsilon_{t+1}^2 \mid \mathcal{H}_t\right]}_{\substack{\leq A + BQ_t^2 \\ \leq \tilde{A} + \tilde{B}(Q_t - q^*)^2}}$$

$$\leq \Big[\underbrace{(1-\alpha_t)^2 + (\gamma^2 + \tilde{B})\alpha_t^2 + 2\alpha_t(1-\alpha_t)\gamma}_{\mu_t = 1 - 2\alpha_t(1-\gamma) + O(\alpha_t^2)}\Big](Q_t - q_*)^2 + \tilde{A}\alpha_t^2$$

(recall that due to our assumptions $\alpha_t \to 0$)

Taking expectation on both sides and iterating yields

$$\mathbb{E}[Q_{t+1} - q_*]^2 \leq \underbrace{\prod_{i=0}^{t} \mu_i \, \mathbb{E}[Q_0 - q_*]^2}_{\substack{\text{decay of initial error} \\ \longrightarrow 0 \text{ as } \sum_i \alpha_i = \infty}} + \underbrace{\tilde{A} \sum_{i=0}^{t} \left(\prod_{j=i+1}^{t} \mu_j\right) \alpha_i^2}_{\substack{\text{cumulated random fluctuations} \\ \longrightarrow 0 \text{ when } \sum_i \alpha_i^2 < \infty}}$$

$\square$

Asynchronous Q Learning Algorithm

If a simulator of the environment is available (e.g. in game playing) one can choose the state-action pair (S_t, A_t) for which the $q_\star$ table approximation will be updated

Otherwise, the updating scheme of the different cells of the $q_\star$ table approximation is random and driven by the environment and exploration policy π (so-called asynchronous updates)

Asynchronous Q Learning

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha(N_t(S_t, A_t)) \left[X_t + \gamma \max_a Q_t(S_{t+1}, a) - Q_t(S_t, A_t) \right]$$

where

$$N_t(s, a) = \sum_{i=0}^{t} \mathbb{1}\{S_i = s, A_i = a\}$$

is the number of visit to (s, a) up to time t and $\alpha(n) = n^{-\beta}$ with $\beta \in (0.5, 1]$

Table of Contents

Importance Sampling

To estimate $\mathbb{E}_\nu[f(X)]$ the basic Monte Carlo approach requires random draws under ν but one may also use weighted draws under $\pi \neq \nu$.

Importance Sampling

Draw $X_1, \ldots, X_n$ i.i.d $\sim \pi$ and approximate $\mu = \mathbb{E}_\nu[f(X)]$ by

$$\widehat{\mu}_n^{\mathrm{IS}} = \frac{1}{n} \sum_{i=1}^{n} \frac{\nu(X_i)}{\pi(X_i)} f(X_i)$$

or

$$\widehat{\mu}_n^{\mathrm{SN\text{-}IS}} = \frac{\sum_{i=1}^{n} \frac{\nu(X_i)}{\pi(X_i)} f(X_i)}{\sum_{i=1}^{n} \frac{\nu(X_i)}{\pi(X_i)}}$$

The latter also works if ν is known up to a constant only and is called self-normalized or Bayesian importance sampling

$W_i = \nu(X_i)/\pi(X_i)$ are called importance weights

Importance Sampling

- It is easily checked that $\mathbb{E}_\pi[\widehat{\mu}_n^{\text{IS}}] = \mu$ (unbiased estimator) and that both $\widehat{\mu}_m^{\text{IS}}$ and $\widehat{\mu}_m^{\text{SN-IS}}$ are consistent estimators of μ

- However, the variance of the estimator may be high and is usually measured by the effective sample size

$$N_n^{\text{ESS}} = \frac{\left(\sum_{i=1}^n W_i\right)^2}{\sum_{i=1}^n W_i^2}$$

  - $1 \le N_n^{\text{ESS}} \le n$
  - n/N_n^{ESS} is a consistent estimator of $\mathbb{E}_\pi[\nu^2(X)/\pi^2(X)] \ge 1$
  - $\text{Var}\left(\widehat{\mu}_n^{\text{IS}}\right) \le \frac{\mathbb{E}_\pi[\nu^2(X)/\pi^2(X)]\|f-\mu\|_\infty^2}{n}$

Importance Sampling in Reinforcement Learning

One can estimate the value of a policy π using a different exploration policy π_0 by importance sampling based on

$$v_\pi(s) = \mathbb{E}_\pi\left(\sum_{t\geq 0}\gamma^t X_t \middle| S_0 = s\right) = \mathbb{E}_{\pi_0}\left(\sum_{t\geq 0}\gamma^t \prod_{i=0}^{t}\frac{\pi(S_i,A_i)}{\pi_0(S_i,A_i)} X_t \middle| S_0 = s\right)$$

But as the variance under $\mathbb{P}_{\pi_0}$ of the importance weigths $W_t = \prod_{i=0}^t \pi(S_i,A_i)/\pi_0(S_i,A_i)$ typically diverges exponentially in t, the estimator may be unreliable when π isn't very close to π_0

Parameterized Policies

A more robust idea, which can be traced back to the likelihood ratio method of [Glynn, 1990], uses importance sampling to estimate the gradient of the value function

- This requires smoothly parameterized policies

Softmax policy

$$\pi_\theta(s,a) = \frac{\exp(\theta_{s,a})}{\sum_{a'} \exp(\theta_{s,a'})}$$

Outside of the finite state (or “tabular”) case this implies the use of features to restrict the space of investigated policies:

$$\pi_\theta(s,a) = \frac{\exp(f_\theta(s,a))}{\sum_{a'} \exp(f_\theta(s,a'))}$$

The choice $f_\theta(s,a) = \theta^T \phi(s,a)$ corresponds to log-linear policies

The Likelihood Ratio Method

The gradient of $\mu_\theta = \mathbb{E}_\theta[f(X)]$ w.r.t. θ may be obtained as

$$\nabla_\theta \mu_\theta = \mathbb{E}_\theta\left[f(X) \nabla \log \pi_\theta(X)\right]$$

where π_θ is the p.d.f. of X

Note that one also has

$$\nabla_\theta \mu_\theta = \mathbb{E}_\theta\left[(f(X) - b) \nabla \log \pi_\theta(X)\right]$$

for all baseline b suggesting estimators of the form

$$\frac{1}{n} \sum_{i=1}^{n} (f(X_i) - b) \nabla \log \pi_\theta(X_i)$$

where choosing a proper b can significantly reduce the variance of the estimator (e.g. using $b = \mu_\theta$ is a good rule of thumb)

Policy Gradient

The identity

$$\nabla_\theta v_\theta(s) = \mathbb{E}_\theta\left[\sum_{t\geq 0}\gamma^t\left(\sum_{i\geq 0}\gamma^i X_{t+i}\right)\nabla_\theta \log\pi_\theta(S_t,A_t)\,\middle|\, S_0 = s\right]$$

suggests the following gradient estimator

Policy Gradient (REINFORCE)

Given m independent trajectories $(S_t^i, A_t^i, X_t^i)_{t\geq 0}^{i=1,\dots,m}$ of the MDP under policy π_θ started from some initial distribution ν

$$\frac{1}{m}\sum_{i=1}^{m}\sum_{t\geq 0}\left(\sum_{j\geq t}\gamma^j X_j\right)\nabla_\theta \log\pi_\theta(S_t,A_t)$$

Provides an unbiased gradient estimate that can typically be used to perform one step of SGD on the parameter θ

Policy Gradient With a Baseline

One also has

$$\nabla_\theta v_\theta(s) = \mathbb{E}_\theta \left[\sum_{t\geq 0} \gamma^t \left(\sum_{i\geq 0} \gamma^i X_{t+i} - v_\theta(S_t) \right) \nabla_\theta \log \pi_\theta(S_t, A_t) \middle| S_0 = s \right]$$

Which may be used to reduce the variance of the gradient approximation as

$$\mathbb{E}_\theta \left[\sum_{i\geq 0} \gamma^i X_{t+i} - v_\theta(S_t) \middle| H_t, A_t \right] = q_\theta(S_t, A_t) - v_\theta(S_t)$$

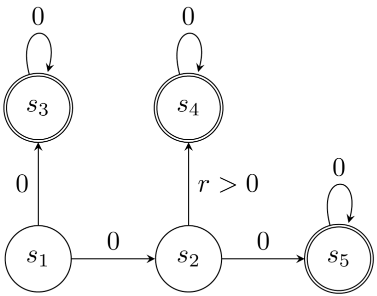
which is known as the advantage function and is centered under π_θ, i.e.

$$\mathbb{E}_\theta \left[\sum_{i\geq 0} \gamma^i X_{t+i} - v_\theta(S_t) \middle| H_t \right] = 0$$

This approach requires to maintain also an approximation of the value function v_θ

Non Concavity

Previous ideas lead to efficient stochastic gradient schemes, but the value function is in general non concave (even for log-linear policies)



From [Agarwal *et al.*, 2021]

Hint: Consider $\theta^{(1)}$ such that $\theta^{(1)}_{s_1,U} = \log 1, \theta^{(1)}_{s_1,R} = \log 3, \theta^{(1)}_{s_2,U} = \log 3, \theta^{(1)}_{s_2,R} = \log 1$ and $\theta^{(2)} = -\theta^{(1)}$ and check that $v_{\theta^{(1)}}(s_1) + v_{\theta^{(2)}}(s_1) > 2v_{(\theta^{(1)}+\theta^{(2)})/2}(s_1)$: note that it also holds in direct parameterization.