

Chapter 1: Deep Reinforcement Learning

Ernest K. Ryu

University of California, Los Angeles

MDP Basics

Markov decision process

RL considers sequential decision making within a *Markov decision process* (MDP):

- Time $t = 0, 1, \dots, T$

$$s_t \xrightarrow{\pi} a_t \xrightarrow{p} (r_t, s_{t+1})$$

- *Trajectory* $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T)$
- *State* $s_t \in \mathcal{S}$. Define $\mathcal{S}^+ = \mathcal{S} \cup \{\text{<term>}\}$. (Always assume $\mathcal{S} \neq \emptyset$.)
- *Action* $a_t \in \mathcal{A}$. (Always assume $\mathcal{A} \neq \emptyset$.)
- *Reward* $r_t \in \mathbb{R}$. We will choose policy to maximize sum of reward.
- T is *terminal time / stopping time*. Defined by $s_T = \text{<term>} = \text{terminal state}$.
- $T = \infty$ possible. If probability of transition to $s = \text{<term>}$ is zero, then necessarily $T = \infty$. If $T = \infty$, the MDP is said to be a continual task and otherwise an episodic task.
- Initial distribution $s_0 \sim p_0$. Often s_0 is fixed.
- Transition probability $p(r, s' | s, a)$ given by environment (usually not precisely known) and $(r_t, s_{t+1}) \sim p(\cdot, \cdot | s_t, a_t)$.

MDP base definitions

- r_t is sometimes a fully deterministic function of (s_t, a_t) . If so, write $r_t = r(s_t, a_t)$.
- s_{t+1} is sometimes a fully deterministic function of (s_t, a_t) .
- For now, assume the dynamics is *stationary*. In general, $(r_t, s_{t+1}) \sim p_t(\cdot, \cdot | s_t, a_t)$. Stationarity means $p_t(r, s' | s, a) = p(r, s' | s, a)$
- a_t is chosen by the agent via a policy π , given s_t .
- If π is stochastic, then $\pi(a|s)$ is a probability distribution and $a_t \sim \pi(\cdot | s_t)$.
- If π is deterministic, then $a_t = \pi(s_t)$.
- Often, $\pi = \pi_\theta$, i.e., π will be a neural network parameterized by θ .

State vs. observation

In general, an agent may not be able to observe the full state. The system may not be Markovian with respect to the observation. E.g., if you see a tiger hide behind a tree, the past observation is relevant, and the fact that you can no longer see the tiger does not mean you are safe.

This leads to the Partially Observable Markov Decision Process (POMDP) formulation, which is much more challenging than MDPs.

For us, assume the agent has fully observes the state s_t .

In LLMs, this is not an issue since the language model has the full conversational history.



MDP generalizations

There are many generalizations of the standard MDP. We won't cover them in this course.

- Non-stationary dynamics: p_0, p_1, \dots
- The terminal time T may be pre-determined. This is an example of non-stationary dynamics, since $p_{T-1}(s_T = \text{<term>} | s_{T-1}, a_{T-1}) = 1$, while $p_t(s_{t+1} = \text{<term>} | s_t, a_t) = 0$ for any $t = 0, \dots, T - 2$.
- The policy π_t can be time-dependent. (Time-dependent policy is necessary only if dynamics is non-stationary.)
- Set of possible actions may depend on s_t . If so, $a_t \in \mathcal{A}(s_t)$.

In practice, the MDP you work with will incorporate some of these variations. You must understand the *principles*, and adapt the base theory to the particular MDP at hand.

Terminal time notation

Note, terminal time T is a random variable. Therefore, $\sum_{t=0}^T$ requires caution to work with. For example, $\mathbb{E}[\sum_{t=0}^T \cdot] \neq \sum_{t=0}^T \mathbb{E}[\cdot]$ and the RHS makes no sense, since the random variable T is outside of the expectation.

For notational and theoretical convenience, define an equivalent MDP that never stops by making the terminal state an *absorbing state*.

- The MDP nominally never stops, i.e., terminal time is $T = \infty$.
- State $s_t \in \mathcal{S} \cup \{\text{<term>}\}$, and we view <term> as a normal non-terminal state.
- The transition probability $p(r, s' | s, a)$ is defined such that if $s_t = \text{<term>}$, then $r_t = 0$ and $s_{t+1} = \text{<term>}$ with probability 1, regardless of the choice of a_t .
 - I.e., Once $s_t = \text{<term>}$, we no longer collect rewards and never escape <term>.
- Therefore, the policy at terminal state $\pi(\cdot | \text{<term>})$ is irrelevant.
 - No matter what action you take, we never move from $s_t = \text{<term>}$.

Imitation learning and behavior cloning

In imitation learning or behavior cloning, we train a $\pi_\theta \approx \pi_{\text{expert}}$, where π_{expert} is the policy of an expert agent by observing actions made by π_{expert} (usually human demonstrations).

Behavior cloning:

Step 1: Sample trajectories $\tau = (s_0, a_0, s_1, a_1, \dots, s_{T-1}, a_{T-1}, s_T) \sim (p_0, \pi_{\text{expert}}, p)$ and form a dataset of state-action pairs: $\mathcal{D}_{\text{expert}} = \{(s_i, a_i)\}$. (No rewards.)

Step 2: Train the model by solving

$$\underset{\theta}{\text{minimize}} \quad \sum_{(s,a) \in \mathcal{D}_{\text{expert}}} \ell(\pi_\theta(s), a)$$

where ℓ is the cross-entropy loss. This is basically supervised learning.

Imitation learning and behavior cloning

Observation: Next token prediction in LLMs can be thought of as behavior cloning.

Behavior cloning may serve as a good starting point (initialization) as we will see with AlphaGo and LLM pre-training.

In many RL settings, the requirement of expert demonstrations can be onerous. In the LLM setting, however, pre-training data is plentiful.

However, behavior cloning by itself does not work very well.

Distribution shift in behavior cloning

In supervised learning, a model trained on **one distribution** will perform when tested on the **same distribution**, but not on **another distribution**. A change of distribution from training to test is called distribution shift, distribution mismatch, covariate shift, concept drift etc.

Behavior cloning fails due to covariate shift. At test time, the trained model observes trajectories $\tau \sim (p_0, \pi_\theta, p)$. But recall, training was done on trajectories $\tau \sim (p_0, \pi_{\text{expert}}, p)$.

As a result, π_θ will encounter states it was not trained on. Maybe π_{expert} is very good and never drives the MDP to a “bad” state, but π_θ may be imperfect and may drift into such bad states. Although π_{expert} may know how to recover from such a bad state, π_θ was never taught this knowledge.

Off-policy vs. on-policy learning

Key insight: In reinforcement learning, the training data depends on the policy. This is true for both imitation learning or reward-based learning.

In off-policy learning, the RL agent trains on actions taken by another policy (or its old self).
In on-policy learning, the RL agent trains on actions taken by the current policy.

Off-policy RL suffers from distribution shift.

Behavior cloning is off-policy learning.

DAgger

Dataset Aggregation (Dagger) is more on-policy imitation learning. In each round, the current policy is used to generate trajectories and the expert provides labels for the states. By retraining on this aggregated dataset, DAgger mitigates distribution shift.

0. Sample trajectory $\tau \sim (p_0, \pi_{\text{expert}}, p)$ and form $\mathcal{D} \leftarrow \mathcal{D}_{\text{expert}} = \{(s_i, a_i)\}$.
1. Train π_θ from demonstration data \mathcal{D} .
2. Sample trajectory $\tau \sim (p_0, \pi_\theta, p)$ and form $\mathcal{D}_{\pi_\theta} = \{s_i\}$. (No actions, no rewards.)
3. Ask human to label $s_i \in \mathcal{D}_{\pi_\theta}$ with actions a_i . Then let $\mathcal{D}_{\pi_\theta} = \{(s_i, a_i)\}$.
4. Aggregate $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_{\pi_\theta}$. Loop back to Step 1.

Asking human experts to take an action on states generated by π_θ is often very unnatural. Not really something you can do with LLMs anyway.

MDP objective

So far, we have described the dynamics of the MDP. The goal/objective of an MDP is to maximize expected discounted return:

$$\underset{\pi}{\text{maximize}} \quad \mathbb{E}_{s_0 \sim p_0}^{\pi} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right].$$

- Cumulative discounted return $\sum_{t=0}^{T-1} \gamma^t r_t = r_0 + \gamma r_1 + \cdots + \gamma^{T-1} r_{T-1} = G_0$
- G_0 is (cumulative) return, r_t is (instantaneous) reward.
- *Discount factor* $\gamma \in (0, 1]$.
When $T = \infty$ possible and reward is bounded, use $\gamma < 1$ to ensure return is finite.
- \mathbb{E}^{π} means expectation over $a_t \sim \pi(\cdot | s_t)$, $(r_t, s_{t+1}) \sim p(\cdot, \cdot | s_t, a_t)$.
- Define the return from time t as $G_t = r_t + \gamma r_{t+1} + \cdots + \gamma^{T-1-t} r_{T-1} = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$.

State value function V^π and State-action value function Q^π

Define the (state) *value function* (*V-value function*)

$$V^\pi(s) = \mathbb{E}^\pi[G_0 | s_0 = s]$$

$\nearrow r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$
 $\searrow s_0 = s, a_0 \sim \pi(\cdot | s_0), (r_0, s_1) \sim p(\cdot, \cdot | s_0, a_0), \dots$

as the expected return starting at state s following policy π .

Note, $V^\pi(\text{<term>}) = 0$.

Define the *state-action value function* (*Q-value function*)

$$Q^\pi(s, a) = \mathbb{E}^\pi[G_0 | s_0 = s, a_0 = a]$$

$\searrow s_0 = s, a_0 = a, (r_0, s_1) \sim p(\cdot, \cdot | s_0, a_0), a_1 \sim \pi(\cdot | s_1), (r_1, s_2) \sim p(\cdot, \cdot | s_1, a_1), \dots$

as the expected return starting at state s taking action a following policy π thereafter.

Note, $Q^\pi(\text{<term>}, a) = 0$ for all $a \in \mathcal{A}$.

Basic properties of V^π and Q^π

Since \mathbb{E}^π denotes expectation over $a_t \sim \pi(\cdot | s_t)$, $(r_t, s_{t+1}) \sim p(\cdot, \cdot | s_t, a_t)$, we have

$$V^\pi(s_0) = \mathbb{E}_{a_0 \sim \pi(\cdot | s_0)} [Q^\pi(s_0, a_0)]$$

By stationarity,

$$V^\pi(s) = \mathbb{E}^\pi[G_t | s_t = s]$$

and

$$Q^\pi(s, a) = \mathbb{E}^\pi[G_t | s_t = s, a_t = a]$$
$$\nearrow r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

1-step transition property of V^π

$$V^\pi(s) = \mathbb{E}_{\substack{a_0 \sim \pi(\cdot | s) \\ (r_0, s_1) \sim p(\cdot, \cdot | s, a_0)}} [r_0 + \gamma V^\pi(s_1) | s_0 = s].$$

by the Markovian property:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}^\pi \left[r_0 + \gamma \sum_{t=1}^{T-1} \gamma^{t-1} r_t \mid s_0 = s \right] \\ &= \mathbb{E}^\pi [r_0 + \gamma G_1 | s_0 = s] \\ &= \mathbb{E}_{\substack{a_0 \sim \pi(\cdot | s) \\ (r_0, s_1) \sim p(\cdot, \cdot | s, a_0)}} \left[\mathbb{E}^\pi [r_0 + \gamma G_1 | s_0, a_0, r_0, s_1] \mid s_0 = s \right] \\ &= \mathbb{E}_{\substack{a_0 \sim \pi(\cdot | s) \\ (r_0, s_1) \sim p(\cdot, \cdot | s, a_0)}} [r_0 + \gamma \mathbb{E}^\pi [G_1 | s_1] | s_0 = s] \\ &= \mathbb{E}_{\substack{a_0 \sim \pi(\cdot | s) \\ (r_0, s_1) \sim p(\cdot, \cdot | s, a_0)}} [r_0 + \gamma V^\pi(s_1) | s_0 = s]. \end{aligned}$$

1-step transition property of Q^π

$$Q^\pi(s, a) = \mathbb{E}_{\substack{(r, s') \sim p(\cdot, \cdot | s, a) \\ a' \sim \pi(\cdot | s')}} [r + \gamma Q^\pi(s', a') | s, a]$$

by the Markovian property:

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E}^\pi [r_0 + \gamma G_1 | s_0 = s, a_0 = a] \\ &= \mathbb{E}^\pi [\mathbb{E}^\pi [r_0 + \gamma G_1 | s_0, a_0, r_0, s_1, a_1] | s_0 = s, a_0 = a] \\ &= \mathbb{E}_{\substack{(r_0, s_1) \sim p(\cdot, \cdot | s, a) \\ a_1 \sim \pi(\cdot | s_1)}} [r_0 + \gamma \mathbb{E}^\pi [G_1 | s_0, a_0, r_0, s_1, a_1] | s_0 = s, a_0 = a] \\ &= \mathbb{E}_{\substack{(r_0, s_1) \sim p(\cdot, \cdot | s, a) \\ a_1 \sim \pi(\cdot | s_1)}} [r_0 + \gamma \mathbb{E}^\pi [G_1 | s_1, a_1] | s_0 = s, a_0 = a] \\ &= \mathbb{E}_{\substack{(r_0, s_1) \sim p(\cdot, \cdot | s, a) \\ a_1 \sim \pi(\cdot | s_1)}} [r_0 + \gamma Q^\pi(s_1, a_1) | s_0 = s, a_0 = a] \end{aligned}$$

Prelim: Banach fixed point theorem

Let \mathcal{X} be a metric space with metric d . Then, we say $\mathcal{T} : \mathcal{X} \rightarrow \mathcal{X}$ is γ -contractive if

$$d(\mathcal{T}(x), \mathcal{T}(y)) \leq \gamma d(x, y) \quad \forall x, y \in \mathcal{X}$$

Theorem) Let \mathcal{X} be a complete metric space with metric d . Let $\mathcal{T} : \mathcal{X} \rightarrow \mathcal{X}$ be a γ -contractive mapping with $\gamma < 1$. Then \mathcal{T} has a fixed point and the fixed point is unique. Furthermore, for any $x \in \mathcal{X}$,

$$\mathcal{T}^k(x) \rightarrow x^*$$

as $k \rightarrow \infty$, where x^* is the unique fixed point.

(\mathbb{R}^n is a complete metric space with any norm.)

Bellman equation for V^π

Theorem) Let π be a policy. Assume $\gamma \in (0,1)$, $|\mathcal{S}| < \infty$, and $|r| \leq R < \infty$ almost surely.[#] Then $V^\pi : \mathcal{S}^+ \rightarrow \mathbb{R}$, the value function of π , exists, and it satisfies the Bellman equation:

$$V(s) = \mathbb{E}_{\substack{a \sim \pi(\cdot | s) \\ (r, s') \sim p(\cdot, \cdot | s, a)}} [r + \gamma V(s') | s]$$

Conversely, if a function $V : \mathcal{S}^+ \rightarrow \mathbb{R}$ satisfies the Bellman equation, then $V = V^\pi$.

[#]These assumptions can be relaxed.

Proof) Existence follows from

$$\left| \sum_{t=1}^{T-1} \gamma^t r_t \right| \leq \sum_{t=1}^{T-1} \gamma^t |r_t| \leq \sum_{t=1}^{T-1} \gamma^t R \leq \frac{R}{1-\gamma} < \infty$$

so the expectation is well defined.

Let \mathcal{B}^π be the *Bellman operator* (for V) mapping from a function to a function:

$$(\mathcal{B}^\pi[V])(s) = \mathbb{E}_{\substack{a \sim \pi(\cdot | s) \\ (r, s') \sim p(\cdot, \cdot | s, a)}} [r + \gamma V(s') | s]$$

So, $\mathcal{B}^\pi[V^\pi] = V^\pi$ by the 1-step transition property, i.e., V^π is a fixed point of \mathcal{B}^π .

Let $\|V\|_\infty = \max_{s \in \mathcal{S}^+} |V(s)|$. Then $\|\cdot\|_\infty$ is a norm on the space of functions from \mathcal{S}^+ to \mathbb{R} .

If \mathcal{B}^π is a strict contraction with respect to $\|\cdot\|_\infty$, then V^π is the unique fixed-point by Banach.

Finally, we show \mathcal{B}^π is a γ -contraction in the $\|\cdot\|_\infty$ -norm:

$$\begin{aligned}
\|\mathcal{B}^\pi[V_1] - \mathcal{B}^\pi[V_2]\|_\infty &= \max_{s \in \mathcal{S}^+} |(\mathcal{B}^\pi[V_1])(s) - (\mathcal{B}^\pi[V_2])(s)| \\
&= \max_{s \in \mathcal{S}^+} \left| \mathbb{E}_{\substack{a \sim \pi(\cdot | s) \\ (r, s') \sim p(\cdot, \cdot | s, a)}} [r + \gamma V_1(s') | s] - \mathbb{E}_{\substack{a \sim \pi(\cdot | s) \\ (r, s') \sim p(\cdot, \cdot | s, a)}} [r + \gamma V_2(s') | s] \right| \\
&= \max_{s \in \mathcal{S}^+} \gamma \left| \mathbb{E}_{\substack{a \sim \pi(\cdot | s) \\ (r, s') \sim p(\cdot, \cdot | s, a)}} [V_1(s') - V_2(s') | s] \right| \\
&\leq \max_{s \in \mathcal{S}^+} \gamma \mathbb{E}_{\substack{a \sim \pi(\cdot | s) \\ (r, s') \sim p(\cdot, \cdot | s, a)}} \left[|V_1(s') - V_2(s')| \mid s \right] \\
&\leq \max_{s \in \mathcal{S}^+} \gamma \max_{s' \in \mathcal{S}^+} |V_1(s') - V_2(s')| = \gamma \|V_1 - V_2\|_\infty
\end{aligned}$$

■

Bellman equation for Q^π

Theorem) Let π be a policy. Assume $\gamma \in (0,1)$, $|\mathcal{S}| < \infty$, $|\mathcal{A}| < \infty$, and $|r| \leq R < \infty$ almost surely. Then the state-action value function Q^π exists, and it satisfies the Bellman equation

$$Q(s, a) = \mathbb{E}_{\substack{(r, s') \sim p(\cdot, \cdot | s, a) \\ a' \sim \pi(\cdot | s')}} [r + \gamma Q(s', a') | s, a]$$

Conversely, if a function $Q : \mathcal{S}^+ \times \mathcal{A} \rightarrow \mathbb{R}$ satisfies the Bellman equation, then $Q = Q^\pi$.

Proof) First, Q^π is well defined since r is almost surely bounded.

Let \mathcal{B}^π be the *Bellman operator* (for Q) mapping from a function to a function:

$$(\mathcal{B}^\pi[Q])(s, a) = \mathbb{E}_{\substack{(r, s') \sim p(\cdot, \cdot \mid s, a) \\ a' \sim \pi(\cdot \mid s')}} [r + \gamma Q(s', a') \mid s, a]$$

Clearly, $\mathcal{B}^\pi[Q^\pi] = Q^\pi$ by the 1-step transition property, i.e., Q^π is a fixed point of \mathcal{B}^π .

Let $\|Q\|_\infty = \max_{s \in \mathcal{S}^+, a \in \mathcal{A}} |Q(s, a)|$.

Then, $\|\cdot\|_\infty$ is a norm on the space of functions from $\mathcal{S}^+ \times \mathcal{A}$ to \mathbb{R} .

If \mathcal{B}^π is a strict contraction with respect to $\|\cdot\|_\infty$, then Q^π is the unique fixed-point by Banach. In the hw, you will prove that \mathcal{B}^π is a strict contraction. ■

Optimal policy and value functions

We say a policy π^* is optimal if

$$V^{\pi^*}(s) \geq V^{\pi}(s) \quad \forall s \in \mathcal{S}^+, \forall \text{ policy } \pi.$$

(Optimal policy does not depend on starting state s .)

Write $V^* = V^{\pi^*}$ for the optimal value function.

Write $Q^* = Q^{\pi^*}$ for the optimal Q-value function.

As we soon establish, the optimal value functions V^* and Q^* are unique, but optimal policy π^* is not unique. However, all optimal policies yield the same value functions.

Bellman optimality equation for V^*

Theorem) Assume $\gamma \in (0,1)$, $|\mathcal{S}| < \infty$, $|\mathcal{A}| < \infty$, and $|r| \leq R < \infty$ almost surely. Then the optimal value function $V^* : \mathcal{S}^+ \rightarrow \mathbb{R}$ exists, and it satisfies the Bellman optimality equation:

$$V(s) = \max_{a \in \mathcal{A}} \mathbb{E}_{(r,s') \sim p(\cdot, \cdot | s, a)} [r + \gamma V(s') | s, a]$$

Conversely, if a function $V : \mathcal{S}^+ \rightarrow \mathbb{R}$ satisfies the Bellman optimality equation, then $V = V^*$.

Finally,

$$\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}} \mathbb{E}_{(r,s') \sim p(\cdot, \cdot | s, a)} [r + \gamma V^*(s') | s, a] = \operatorname{argmax}_{a \in \mathcal{A}} Q^*(s, a)$$

is an optimal deterministic policy.

Quick lemma

Lemma) For any $u(s)$ and $v(s)$,

$$\left| \max_s v(s) - \max_s u(s) \right| \leq \max_s |u(s) - v(s)|$$

Proof)

$$\max_s u(s) - \max_s v(s) \leq \max_s \{u(s) - v(s)\} \leq \max_s |u(s) - v(s)|$$

and symmetrically,

$$\max_s v(s) - \max_s u(s) \leq \max_s |u(s) - v(s)|$$

We conclude with

$$\max \left\{ \max_s v(s) - \max_s u(s), \max_s u(s) - \max_s v(s) \right\} = \left| \max_s v(s) - \max_s u(s) \right| \quad \blacksquare$$

Proof) Let \mathcal{B}^* be the *Bellman optimality operator* (for V) mapping from a function to a function:

$$(\mathcal{B}^*[V])(s) = \max_{a \in \mathcal{A}} \mathbb{E}_{(r,s') \sim p(\cdot, \cdot | s, a)} [r + \gamma V(s') | s, a]$$

By the following reasoning, \mathcal{B}^* is a strict contraction with respect to $\|\cdot\|_\infty$.

$$\begin{aligned} \|\mathcal{B}^*[V_1] - \mathcal{B}^*[V_2]\|_\infty &= \max_{s \in \mathcal{S}^+} |(\mathcal{B}^*[V_1])(s) - (\mathcal{B}^*[V_2])(s)| \\ &= \max_{s \in \mathcal{S}^+} \left| \max_{a \in \mathcal{A}} \mathbb{E}_{(r,s') \sim p(\cdot, \cdot | s, a)} [r + \gamma V_1(s') | s, a] - \max_{a \in \mathcal{A}} \mathbb{E}_{(r,s') \sim p(\cdot, \cdot | s, a)} [r + \gamma V_2(s') | s, a] \right| \\ &\leq \max_{s \in \mathcal{S}^+} \max_{a \in \mathcal{A}} \gamma \left| \mathbb{E}_{(r,s') \sim p(\cdot, \cdot | s, a)} [V_1(s') - V_2(s') | s, a] \right| \\ &\leq \max_{s \in \mathcal{S}^+} \max_{a \in \mathcal{A}} \gamma \mathbb{E}_{(r,s') \sim p(\cdot, \cdot | s, a)} \left[|V_1(s') - V_2(s')| \mid s, a \right] \\ &\leq \max_{s \in \mathcal{S}^+} \max_{a \in \mathcal{A}} \gamma \max_{s' \in \mathcal{S}^+} |V_1(s') - V_2(s')| = \gamma \|V_1 - V_2\|_\infty \end{aligned}$$

So \mathcal{B}^* has a unique fixed-point that we denote as V^* .
(We don't yet know if V^* is the optimal value function.)

Next, define π^* to be a deterministic policy defined by

$$\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}} \mathbb{E}_{(r,s') \sim p(\cdot, \cdot | s, a)} [r + \gamma V^*(s') | s, a]$$

where ties in the argmax are broken arbitrarily. (We don't yet know if π^* is the optimal policy.)

Then, $V^* = V^{\pi^*}$ by

$$\begin{aligned} V^*(s) &= \max_{a \in \mathcal{A}} \mathbb{E}_{(r,s') \sim p(\cdot, \cdot | s, a)} [r + \gamma V^*(s') | s, a] \\ &= \mathbb{E}_{\substack{a \sim \pi^*(\cdot | s) \\ (r,s') \sim p(\cdot, \cdot | s, a)}} [r + \gamma V^*(s') | s] \\ &= \mathbb{E}^{\pi^*} [r_0 + \gamma V^*(s_1) | s_0 = s] \\ &= \mathbb{E}^{\pi^*} [r_0 + \gamma (\mathbb{E}^{\pi^*} [r_1 + \gamma V^*(s_2) | s_1]) | s_0 = s] \\ &= \mathbb{E}^{\pi^*} [r_0 + \gamma (\mathbb{E}^{\pi^*} [r_1 + \gamma V^*(s_2) | r_0, s_1]) | s_0 = s] \\ &= \mathbb{E}^{\pi^*} [\mathbb{E}^{\pi^*} [r_0 + \gamma (r_1 + \gamma V^*(s_2)) | r_0, s_1] | s_0 = s] \\ &= \mathbb{E}^{\pi^*} [r_0 + \gamma r_1 + \gamma^2 V^*(s_2) | s_0 = s] \\ &= \mathbb{E}^{\pi^*} [r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots | s_0 = s] \\ &= V^{\pi^*}(s) \end{aligned}$$

Lemma) Let π be a policy. Let \mathcal{B}^π be the Bellman operator and \mathcal{B}^* the Bellman optimality operator. For any $V : \mathcal{S}^+ \rightarrow \mathbb{R}$, we have

$$\mathcal{B}^\pi[V] \leq \mathcal{B}^*[V]$$

Moreover, for any $U : \mathcal{S}^+ \rightarrow \mathbb{R}$ and $V : \mathcal{S}^+ \rightarrow \mathbb{R}$ satisfying $U \leq V$, we have

$$\mathcal{B}^*[U] \leq \mathcal{B}^*[V]$$

(Here, \leq denotes pointwise inequality. So $U \leq V$ means $U(s) \leq V(s)$ for all $s \in \mathcal{S}^+$.)

Proof) Exercise.

We now show that π^* is optimal. Let π be any policy. Then,

$$V^\pi = \mathcal{B}^\pi[V^\pi] \leq \mathcal{B}^*[V^\pi] \leq (\mathcal{B}^*)^2[V^\pi] \leq \dots \leq (\mathcal{B}^*)^k[V^\pi] \rightarrow V^* = V^{\pi^*}$$

($V^\pi = \mathcal{B}^\pi[V^\pi]$ follows from the Bellman equation theorem. $\mathcal{B}^\pi[V^\pi] \leq \mathcal{B}^*[V^\pi]$ follows from the previous lemma. Since $V^\pi \leq \mathcal{B}^*[V^\pi]$, we can apply \mathcal{B}^* to both sides to get $\mathcal{B}^*[V^\pi] \leq (\mathcal{B}^*)^2[V^\pi]$. Convergence is due to Banach.)

So $V^\pi \leq V^{\pi^*}$ for any policy π . So π^* is optimal and V^* is the optimal value function.

Finally, since $V^* = V^{\pi^*}$, we have

$$\mathbb{E}_{(r,s') \sim p(\cdot, \cdot | s, a)} [r + \gamma V^*(s') | s, a] = \mathbb{E}_{(r,s') \sim p(\cdot, \cdot | s, a)} [r + \gamma V^{\pi^*}(s') | s, a] = Q^{\pi^*}(s, a) = Q^*(s, a)$$

■

Bellman optimality equation for Q^*

Theorem) Assume $\gamma \in (0,1)$, $|S| < \infty$, $|\mathcal{A}| < \infty$, and $|r| \leq R < \infty$ almost surely. Then the optimal state-action value function $Q^* : S^+ \times \mathcal{A} \rightarrow \mathbb{R}$ exists, and it satisfies the Bellman optimality equation:

$$Q(s, a) = \mathbb{E}_{(r, s') \sim p(\cdot, \cdot | s, a)} [r + \gamma \max_{a' \in \mathcal{A}} Q(s', a') | s, a]$$

Conversely, if a function $Q : S^+ \times \mathcal{A} \rightarrow \mathbb{R}$ satisfies the Bellman optimality equation, then $Q = Q^*$. Finally,

$$\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q^*(s, a)$$

is an optimal deterministic policy.

Proof) Let \mathcal{B}^* be the *Bellman optimality operator* (for Q) mapping from a function to a function:

$$(\mathcal{B}^*[Q])(s, a) = \mathbb{E}_{(r, s') \sim p(\cdot, \cdot | s, a)} \left[r + \gamma \max_{a' \in \mathcal{A}} Q(s', a') \mid s, a \right]$$

I will leave it as an exercise to show \mathcal{B}^* is a strict contraction with respect to $\|\cdot\|_\infty$. So \mathcal{B}^* has a unique fixed-point that we denote as Q^* .

It remains to show that Q^* is the optimal state-value function.

We have $\max_{a \in \mathcal{A}} Q^*(s, a) = V^*(s)$ since

$$\begin{aligned} Q^*(s, a) &= \mathbb{E}_{(r, s') \sim p(\cdot, \cdot | s, a)} \left[r + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a') \mid s, a \right] \\ \max_{a \in \mathcal{A}} Q^*(s, a) &= \max_{a \in \mathcal{A}} \mathbb{E}_{(r, s') \sim p(\cdot, \cdot | s, a)} \left[r + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a') \mid s, a \right] \\ \underbrace{\max_{a \in \mathcal{A}} Q^*(\cdot, a)}_{=V^*} &= \mathcal{B}^* \left[\underbrace{\max_{a \in \mathcal{A}} Q^*(\cdot, a)}_{=V^*} \right] \quad (\mathcal{B}^* \text{ for } V) \end{aligned}$$

By our previous theorem, an optimal policy π^* exists and $V^* = V^{\pi^*}$.

We conclude $Q^* = Q^{\pi^*}$ with

$$\begin{aligned} Q^*(s, a) &= \mathbb{E}_{(r, s') \sim p(\cdot, \cdot \mid s, a)} \left[r + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a') \mid s, a \right] \\ &= \mathbb{E}_{(r, s') \sim p(\cdot, \cdot \mid s, a)} \left[r + \gamma V^*(s') \mid s, a \right] \\ &= \mathbb{E}_{(r, s') \sim p(\cdot, \cdot \mid s, a)} \left[r + \gamma V^{\pi^*}(s') \mid s, a \right] \\ &= Q^{\pi^*}(s, a) \end{aligned}$$

■

Value iteration (VI)

V-value iteration:

$$V^{k+1} = \mathcal{B}^*[V^k]$$

$$V^{k+1}(s) = \max_{a \in \mathcal{A}} \mathbb{E}_{(r, s') \sim p(\cdot, \cdot | s, a)} [r + \gamma V^k(s') | s, a]$$

$$\pi^{k+1}(s) = \operatorname{argmax}_{a \in \mathcal{A}} \mathbb{E}_{(r, s') \sim p(\cdot, \cdot | s, a)} [r + \gamma V^k(s') | s, a]$$

Q-value iteration:

$$Q^{k+1} = \mathcal{B}^*[Q^k]$$

$$Q^{k+1}(s, a) = \mathbb{E}_{(r, s') \sim p(\cdot, \cdot | s, a)} [r + \gamma \max_{a' \in \mathcal{A}} Q^k(s', a') | s, a]$$

$$\pi^k(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q^k(s, a)$$

Value iteration (VI)

V-value iteration:

$$V^{k+1} = \mathcal{B}^*[V^k]$$

Q-value iteration:

$$Q^{k+1} = \mathcal{B}^*[Q^k]$$

Theorem) For $\gamma \in (0,1)$, both iterations converge with rate

$$\|V^k - V^*\|_\infty \leq \gamma^k \|V^0 - V^*\|_\infty \quad \|Q^k - Q^*\|_\infty \leq \gamma^k \|Q^0 - Q^*\|_\infty$$

for $k = 0, 1, \dots$ (For $\gamma = 1$, convergence is not guaranteed.)

Proof) This follows from γ -contractiveness of \mathcal{B}^* . ■

VI is usually not implemented in its exact form. But VI serves as a conceptual framework for designing and understanding many practical deep RL algorithms such as DQN.

Accelerated value iteration

Aside) VI is suboptimal, and it can be accelerated with *anchored value iteration* (AncVI):

$$U^k = \beta_k U^0 + (1 - \beta_k) \mathcal{B}^*[U^{k-1}]$$

for $k = 0, 1, \dots$, where $\beta_k = \frac{1}{\sum_{i=0}^k \gamma^{-2i}}$ and U^0 is an initial point. $U^k = V^k$ or $U^k = Q^k$.

Theorem) If $U^0 \leq \mathcal{B}^*[U^0]$, then AncVI exhibits the rate

$$\begin{aligned} \|\mathcal{B}^*[U^k] - U^k\|_\infty &\leq \frac{(\gamma^{-1} - \gamma)(1 + \gamma - \gamma^{k+1})}{(\gamma^{k+1})^{-1} - \gamma^{k+1}} \|U^0 - U^*\|_\infty \\ &= \left(\frac{1}{k+1} + \frac{k}{k+1} (1 - \gamma) + \mathcal{O}((1 - \gamma)^2) \right) \|U^0 - U^*\|_\infty \end{aligned}$$

for $k = 0, 1, \dots$. This rate is faster than that of VI, and it matches a complexity lower bound up to a constant factor of 4 (and thus is optimal).

This result suggests that methods designed as approximations of vanilla VI may be theoretically suboptimal (in the sense of worst-case guarantees).

Policy iteration

Policy iteration (PI) is a classical algorithm that alternates policy evaluation and policy improvement steps.

Start with π_0

for $k = 0, 1, \dots$

- Compute V^{π_k} and/or Q^{π_k} (Policy Evaluation)
- Compute $\pi_{k+1}(s) = \operatorname{argmax}_{a \in \mathcal{A}} \mathbb{E}_{(r, s') \sim p(\cdot, \cdot | s, a)} [r + \gamma V^{\pi_k}(s') | s, a]$ (Policy Improvement)
$$= \operatorname{argmax}_{a \in \mathcal{A}} Q^{\pi_k}(s, a) = \operatorname{argmax}_{a \in \mathcal{A}} A^{\pi_k}(s, a)$$

PI is usually not implemented in its exact form. But PI serves as a conceptual framework for designing and understanding many practical deep RL algorithms such as PPO.

Policy improvement theorem

Theorem) Consider the PI iteration. Assume $\gamma \in (0,1)$, $|\mathcal{S}| < \infty$, $|\mathcal{A}| < \infty$, and $|r| \leq R < \infty$ almost surely. Then,

$$V^{\pi_{k+1}} \geq V^{\pi_k}$$

for $k = 0, 1, \dots$. Furthermore, there is a $K \in \mathbb{N}$ such that $V^{\pi_k} = V^*$ and π_k is an optimal policy for all $k \geq K$. (Policies become optimal after finitely many steps.)

What can go wrong at $\gamma = 1$?

It is instructive to understand when undiscounted ($\gamma = 1$) MDPs lead to pathologies.

Scenario 1: There is no problem. (There is an optimal policy π^* and $V^* = \mathcal{B}^*[V^*] = V^{\pi^*}$.)

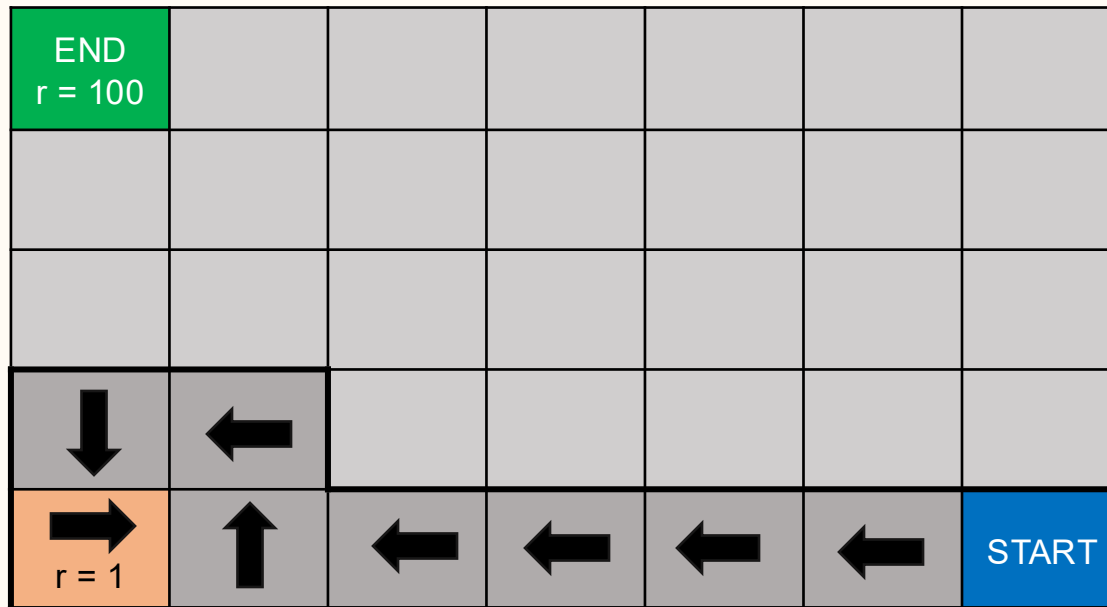
This is the case when \mathcal{S} and \mathcal{A} are finite and V^π is well defined and finite for all policy π .

$$V^\pi(s) = \mathbb{E}^\pi \left[\sum_{t=0}^{T-1} r_t \mid s_0 = s \right] \neq \pm\infty$$

E.g. If you only get a reward at the end of the episode (and never get any reward if the episode never terminates), then everything is well defined. The RL-LLM setting without KL-penalty falls under this case.

What can go wrong at $\gamma = 1$?

Scenario 2: You can get $+\infty$ reward by exploiting a cycle with positive reward.



This is not a problem with discounted MDPs; when $\gamma < 1$, even if the optimal policy exploits a positive-reward cycle, the optimal cumulative discounted return will be finite.

What can go wrong at $\gamma = 1$?

Scenario 2: You can get $+\infty$ reward with a cycle with positive reward.

One resolution is to consider the average-reward MDP formulation, instead of the undiscounted total return MDP. An average-reward MDP maximizes R^π :

$$R^\pi(s) = \liminf_{M \rightarrow \infty} \frac{1}{M} \mathbb{E}^\pi \left[\sum_{t=0}^{M \wedge T - 1} r_t \mid s_0 = s \right]$$

Analyzing average-reward MDPs tends to be more technical than the total return MDPs, and it is an active area of research.[#]

[#]M. Zurek and Y. Chen, Span-based optimal sample complexity for weakly communicating and general average reward MDPs, *NeurIPS*, 2024.

[#]J. Lee and E. K. Ryu, Optimal non-asymptotic rates of value iteration for average-reward Markov decision processes, *ICLR*, 2025.

What can go wrong at $\gamma = 1$?

Scenario 3: Total return (infinite sum) may not be well defined.

For example, if $(r_0, r_1, r_2, r_3, r_4, r_5, \dots) = (+1, -1, +1, -1, +1, -1, \dots)$, then the total return is not summable.

$$V^\pi(s) = \mathbb{E}^\pi \left[\sum_{t=0}^{T-1} r_t \mid s_0 = s \right] = ?$$

A reasonable remedy would be to consider a Cesàro (Abel) sum or the liminf of the partial sums, but this is a complication we will not go into.

Deep Policy Evaluation

Policy evaluation vs. optimization

The goal of RL is to find a good policy π .

We start by studying how to *evaluate* a given policy π .

- Called policy evaluation.

We later talk about how to *optimize/improve* the policy policy π .

- Called policy optimization, policy improvement, control.

Policy evaluation: Monte Carlo

First consider approximating the value function V^π of a given policy π .

For a given $s \in \mathcal{S}$, Monte Carlo (MC) method is

$$V^\pi(s) = \mathbb{E}^\pi \left[\sum_{t=0}^{T-1} \gamma^t r_t \mid s_0 = s \right] \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^{T^{(i)}-1} \gamma^t r_t^{(i)}$$

where

$$\left. \begin{array}{l} \tau^{(1)} = \left(s_0^{(1)}, a_0^{(1)}, r_0^{(1)}, \dots, s_{T^{(1)}-1}^{(1)}, a_{T^{(1)}-1}^{(1)}, r_{T^{(1)}-1}^{(1)}, s_{T^{(1)}}^{(1)} \right) \\ \vdots \\ \tau^{(N)} = \left(s_0^{(N)}, a_0^{(N)}, r_0^{(N)}, \dots, s_{T^{(N)}-1}^{(N)}, a_{T^{(N)}-1}^{(N)}, r_{T^{(N)}-1}^{(N)}, s_{T^{(N)}}^{(N)} \right) \end{array} \right\} \begin{array}{l} N \text{ independent trajectories} \\ \text{with } s_0^{(i)} = s \end{array}$$

If the size of \mathcal{S} is small, we could do this for all $s \in \mathcal{S}$ to approximate $V^\pi(\cdot)$.

$V_\phi \approx V^\pi$ with MC

When the size of \mathcal{S} is large or infinite, we must approximate V^π via a neural network V_ϕ :

$$\underset{\phi \in \mathbb{R}^p}{\text{minimize}} \quad \underbrace{\mathbb{E}_{s \sim p_0} \left[\frac{1}{2} (V_\phi(s) - V^\pi(s))^2 \right]}_{=\mathcal{L}(\phi)}$$

Gradient of \mathcal{L} :

$$\begin{aligned} \nabla_\phi \mathcal{L}(\phi) &= \mathbb{E}_{s \sim p_0} [(V_\phi(s) - V^\pi(s)) \nabla_\phi V_\phi(s)] \\ &= \mathbb{E}_{s \sim p_0} \left[\left(V_\phi(s) - \mathbb{E}^\pi \left[\sum_{t=0}^{T-1} \gamma^t r_t \mid s_0 = s \right] \right) \nabla_\phi V_\phi(s) \right] \\ &= \mathbb{E}_{s \sim p_0} \left[\mathbb{E}^\pi \left[\left(V_\phi(s) - \sum_{t=0}^{T-1} \gamma^t r_t \right) \nabla_\phi V_\phi(s) \mid s_0 = s \right] \right] \\ &= \mathbb{E}_{s_0 \sim p_0}^\pi \left[\underbrace{\left(V_\phi(s_0) - \sum_{t=0}^{T-1} \gamma^t r_t \right) \nabla_\phi V_\phi(s_0)}_g \right]. \end{aligned}$$

Since $V^\pi(\text{<term>}) = 0$, we usually let
 $V_\phi : \mathcal{S} \rightarrow \mathbb{R}$
i.e., we don't allow $V_\phi(\text{<term>})$ and
hard-code the case when
 $V^\pi(\text{<term>}) = 0$ is needed.

$V_\phi \approx V^\pi$ with MC + SGD

while (not converged)

 sample trajectory $\tau \sim (p_0, \pi, p)$

$$g = \left(V_\phi(s_0) - \sum_{t=0}^{T-1} \gamma^t r_t \right) \nabla_\phi V_\phi(s_0)$$

 update ϕ using g with an optimizer

end

$V_\phi \approx V^\pi$ with MC + SGD

while (not converged)

$s_0 \sim p_0, t = 0$

while ($s_t \neq \text{<term>}$)

$a_t \sim \pi(\cdot | s_t)$

$(r_t, s_{t+1}) \sim p(\cdot, \cdot | s_t, a_t)$

$t = t + 1$

end

$T = t$

} sample trajectory τ

$$g = \left(V_\phi(s_0) - \sum_{t=0}^{T-1} \gamma^t r_t \right) \nabla_\phi V_\phi(s_0)$$

update ϕ using g with an optimizer

end

Policy evaluation: Temporal difference

In *Temporal difference* (TD) learning, we use the reward of one transition.

$$V^\pi(s) = \mathbb{E}^\pi [r_0 + \gamma V^\pi(s_1) \mid s_0 = s]$$
$$\approx \frac{1}{N} \sum_{i=1}^N \left(r_0^i + \gamma V^\pi(s_1^{(i)}) \right)$$
$$\left. \begin{array}{c} (s_0^{(1)}, a_0^{(1)}, r_0^{(1)}, s_1^{(1)}) \\ \vdots \\ (s_0^{(N)}, a_0^{(N)}, r_0^{(N)}, s_1^{(N)}) \end{array} \right\} \begin{array}{l} N \text{ independent 1-step transitions} \\ \text{with } s_0^{(i)} = s \end{array}$$

Exactly actionable only if V^π is known.

$V_\phi \approx V^\pi$ with TD + (apx) SGD

$$\underset{\phi \in \mathbb{R}^p}{\text{minimize}} \quad \underbrace{\mathbb{E}_{s \sim p_0} \left[\frac{1}{2} (V_\phi(s) - V^\pi(s))^2 \right]}_{=\mathcal{L}(\phi)}$$

Gradient of \mathcal{L} :

$$\begin{aligned} \nabla_\phi \mathcal{L}(\phi) &= \mathbb{E}_{s \sim p_0}^\pi \left[\left(V_\phi(s) - \sum_{t=0}^{T-1} \gamma^t r_t \right) \nabla_\phi V_\phi(s) \right] \\ &= \mathbb{E}_{s_0 \sim p_0}^\pi \left[\mathbb{E} \left[\left(V_\phi(s_0) - \sum_{t=0}^{T-1} \gamma^t r_t \right) \nabla_\phi V_\phi(s_0) \mid s_0, a_0, r_0, s_1 \right] \right] \\ &= \mathbb{E}_{s_0 \sim p_0}^\pi \left[\left(V_\phi(s_0) - r_0 - \gamma \mathbb{E} \left[\sum_{t=1}^{T-1} \gamma^{t-1} r_t \mid s_0, a_0, r_0, s_1 \right] \right) \nabla_\phi V_\phi(s_0) \right] \\ &= \mathbb{E}_{s_0 \sim p_0}^\pi \left[\underbrace{(V_\phi(s_0) - r_0 - \gamma V^\pi(s_1))}_{g} \nabla_\phi V_\phi(s_0) \right] \end{aligned}$$

$V_\phi \approx V^\pi$ with TD + (apx) SGD

SGD implementation

```
while (not converged)
   $s_0 \sim p_0, a_0 \sim \pi(\cdot | s_0), (r_0, s_1) \sim p(\cdot, \cdot | s_0, a_0)$ 
   $g = (V_\phi(s_0) - r_0 - \gamma V^\pi(s_1)) \nabla_\phi V_\phi(s_0)$ 
  update  $\phi$  using  $g$  with an optimizer
end
```

not actionable

Approximate SGD implementation (g is no longer an unbiased estimate of $\nabla \mathcal{L}$)

```
while (not converged)
   $s_0 \sim p_0, a_0 \sim \pi(\cdot | s_0), (r_0, s_1) \sim p(\cdot, \cdot | s_0, a_0)$ 
   $g = (V_\phi(s_0) - r_0 - \gamma V_\phi(s_1)) \nabla_\phi V_\phi(s_0)$  # If  $s_1 == \text{<term>}$ , then  $V_\phi(s_1) = 0$ 
  update  $\phi$  using  $g$  with an optimizer
end
```

$V_\phi \approx V^\pi$ with TD + (apx) SGD

More precisely, the implementation should be

```
while (not converged)
   $s_0 \sim p_0, a_0 \sim \pi(\cdot | s_0), (r_0, s_1) \sim p(\cdot, \cdot | s_0, a_0)$ 
   $y = \begin{cases} r_0 + \gamma V_\phi(s_1) & \text{if } s_1 \neq \text{<term>} \\ r_0 & \text{if } s_1 == \text{<term>} \end{cases}$ 
   $g = (V_\phi(s_0) - y) \nabla_\phi V_\phi(s_0)$ 
  update  $\phi$  using  $g$  with an optimizer
end
```


Stop-gradient operator

How should we compute the approximate stochastic gradient?

$$g = (V_\phi(s_0) - r_0 - \gamma V_\phi(s_1)) \nabla_\phi V_\phi(s_0)$$

Option 1. Compute $(V_\phi(s_0) - r_0 - \gamma V_\phi(s_1))$ (a scalar) and $\nabla_\phi V_\phi(s_0)$ and multiply the two. This is correct, but it is cumbersome.

Option 2. Forward-evaluate $\frac{1}{2}(V_\phi(s_0) - r_0 - \gamma V_\phi(s_1))^2$ and backprop to compute

$$\nabla_\phi \frac{1}{2}(V_\phi(s_0) - r_0 - \gamma V_\phi(s_1))^2$$

This is **incorrect**! If you apply the chain rule, you don't get the same g .

Stop-gradient operator

How should we compute the approximate stochastic gradient?

$$g = (V_\phi(s_0) - r_0 - \gamma V_\phi(s_1)) \nabla_\phi V_\phi(s_0)$$

Option 3. Forward-evaluate $V_\phi(s_1)$, forward-evaluate $\frac{1}{2} (V_\phi(s_0) - r_0 - \gamma \llbracket V_\phi(s_1) \rrbracket)^2$ and backprop to compute

$$\nabla_\phi \frac{1}{2} (V_\phi(s_0) - r_0 - \gamma \llbracket V_\phi(s_1) \rrbracket)^2$$

where $\llbracket \cdot \rrbracket$ denotes the *stop-gradient operator*.
(In PyTorch, use `.detach()` to perform stop-gradient.)

The stop-gradient operator treats its input as a constant, and stops backpropagation, i.e., the chain rule is not invoked on the input of $\llbracket \cdot \rrbracket$.

$V_\phi \approx V^\pi$ with TD + (apx) SGD

Using the stop-gradient operator, express TD with approximate SGD as

```
while (not converged)
   $s_0 \sim p_0, a_0 \sim \pi(\cdot | s_0), (r_0, s_1) \sim p(\cdot, \cdot | s_0, a_0)$ 
   $g = \nabla_\phi \frac{1}{2} (V_\phi(s_0) - r_0 - \gamma \llbracket V_\phi(s_1) \rrbracket)^2$  # If  $s_1 == \text{<term>}$ , then  $V_\phi(s_1) = 0$ 
  update  $\phi$  using  $g$  with an optimizer
end
```

Semi-gradient method

TD + apx SGD with $g = \nabla_{\phi} \frac{1}{2} (V_{\phi}(s_0) - r_0 - \gamma \mathbb{I}[V_{\phi}(s_1)])^2$ is provably not an instance of gradient descent.[#]

These methods are called *semi-gradient methods*, in the sense that it kind of resembles a gradient method.

One can replace the gradient computation with $g = \nabla_{\phi} \frac{1}{2} (V_{\phi}(s_0) - r_0 - \gamma V_{\phi}(s_1))^2$, i.e., try to directly minimize the Bellman error.

This has been explored under the name gradient TD (GTD)[%], but GTD tends to perform worse[&] than the semi-gradient TD.

[#]Appendix I, E. Barnard, Temporal-difference methods and Markov models, *IEEE Transactions on Systems, Man, and Cybernetics*, 1993.

[%]R. S. Sutton, H. Maei, C. Szepesvári, A convergent O(n) temporal-difference algorithm for off-policy learning with linear function approximation, *NeurIPS*, 2008.

[&]R. S. Sutton, H. R. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvári, and E. Wiewiora, Fast gradient-descent methods for temporal-difference learning with linear function approximation, *ICML*, 2009.

k -step TD policy evaluation

k -step TD interpolates between MC and (1-step) TD using the k -step transition property:

$$V^\pi(s) = \mathbb{E}^\pi \left[\sum_{t=0}^{(k \wedge T)-1} \gamma^t r_t + \gamma^{k \wedge T} V^\pi(s_{k \wedge T}) \mid s_0 = s \right]$$

where $k \wedge T = \min(k, T)$.

k -step transition property

$$\begin{aligned}
 V^\pi(s) &= \mathbb{E}^\pi \left[\sum_{t=0}^{k-1} \gamma^t r_t + \gamma^k \sum_{t=k}^{\infty} \gamma^{t-k} r_t \mid s_0 = s \right] \\
 &= \mathbb{E}^\pi \left[\mathbb{E}^\pi \left[\sum_{t=0}^{k-1} \gamma^t r_t + \gamma^k \sum_{t=k}^{\infty} \gamma^{t-k} r_t \mid s_0, a_0, r_0, s_1, \dots, s_{T(k)-1}, a_{k-1}, r_{k-1}, s_k \right] \mid s_0 = s \right] \\
 &= \mathbb{E}^\pi \left[\sum_{t=0}^{k-1} \gamma^t r_t + \gamma^k \mathbb{E}^\pi \left[\sum_{t=k}^{\infty} \gamma^{t-k} r_t \mid s_0, a_0, r_0, s_1, \dots, s_{k-1}, a_{k-1}, r_{k-1}, s_k \right] \mid s_0 = s \right] \\
 &= \mathbb{E}^\pi \left[\sum_{t=0}^{k-1} \gamma^t r_t + \gamma^k V^\pi(s_k) \mid s_0 = s \right] \\
 &= \mathbb{E}^\pi \left[\sum_{t=0}^{(k \wedge T)-1} \gamma^t r_t + \gamma^{k \wedge T} V^\pi(s_{k \wedge T}) \mid s_0 = s \right] \\
 &\approx \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^{(k \wedge T^{(i)})-1} \gamma^t r_t^{(i)} + \gamma^{k \wedge T^{(i)}} V^\pi(s_{k \wedge T^{(i)}}^{(i)})
 \end{aligned}$$

$$\left. \begin{aligned}
 &(s_0^{(1)}, a_0^{(1)}, r_0^{(1)}, \dots, s_{k \wedge T^{(1)}-1}^{(1)}, a_{k \wedge T^{(1)}-1}^{(1)}, r_{k \wedge T^{(1)}-1}^{(1)}, s_{k \wedge T^{(1)}}^{(1)}) \\
 &\vdots \\
 &(s_0^{(N)}, a_0^{(N)}, r_0^{(N)}, \dots, s_{k \wedge T^{(N)}-1}^{(N)}, a_{k \wedge T^{(N)}-1}^{(N)}, r_{k \wedge T^{(N)}-1}^{(N)}, s_{k \wedge T^{(N)}}^{(N)})
 \end{aligned} \right\}$$

N independent partial trajectories
with $s_0^{(i)} = s$

Exactly actionable only if V^π is known.

$V^\pi \approx V_\phi$ with k -step TD

$$\underset{\phi \in \mathbb{R}^p}{\text{minimize}} \quad \underbrace{\mathbb{E}_{s \sim p_0} \left[\frac{1}{2} (V_\phi(s) - V^\pi(s))^2 \right]}_{=\mathcal{L}(\phi)}$$

Gradient of \mathcal{L} :

$$\begin{aligned} \nabla_\phi \mathcal{L}(\phi) &= \mathbb{E}_{s \sim p_0} \left[(V_\phi(s) - V^\pi(s)) \nabla_\phi V_\phi(s) \right] \\ &= \mathbb{E}_{s_0 \sim p_0}^\pi \left[\underbrace{\left(V_\phi(s_0) - \sum_{t=0}^{(k \wedge T)-1} \gamma^t r_t - \gamma^{k \wedge T} V^\pi(s_{k \wedge T}) \right)}_g \nabla_\phi V_\phi(s_0) \right] \end{aligned}$$

$V^\pi \approx V_\phi$ with k -step TD + SGD

while (not converged)

sample partial trajectory up to $t = k \wedge T$ with (p_0, π, p)

$$g = \left(V_\phi(s_0) - \sum_{t=0}^{(k \wedge T)-1} \gamma^t r_t - \gamma^{k \wedge T} V^\pi(s_{k \wedge T}) \right) \nabla_\phi V_\phi(s_0)$$

update ϕ using g with an optimizer

not actionable

end

$V^\pi \approx V_\phi$ with k -step TD + SGD

while (not converged)

$s_0 \sim p_0, t = 0, T = \infty$

for $t = 0, \dots, k - 1$

if $s_t == \text{term}$

$T = t$

break

end

$a_t \sim \pi(\cdot | s_t)$

$(r_t, s_{t+1}) \sim p(\cdot, \cdot | s_t, a_t)$

end

} sample partial trajectory up to $t = k \wedge T$

$$g = \left(V_\phi(s_0) - \sum_{t=0}^{(k \wedge T) - 1} \gamma^t r_t - \gamma^{k \wedge T} V^\pi(s_{k \wedge T}) \right) \nabla_\phi V_\phi(s_0)$$

update ϕ using g with an optimizer

not actionable

end

$V^\pi \approx V_\phi$ with k -step TD + (apx) SGD

Approximate SGD implementation (g is no longer an unbiased estimate of $\nabla \mathcal{L}$)

```
while (not converged)
```

```
    sample partial trajectory up to  $t = k \wedge T$  with  $(p_0, \pi, p)$ 
```

```
    # If  $s_{k \wedge T} == \text{term}$ , then  $V_\phi(s_{k \wedge T}) = 0$ 
```

$$g = \left(V_\phi(s_0) - \sum_{t=0}^{(k \wedge T)-1} \gamma^t r_t - \gamma^{k \wedge T} V_\phi(s_{k \wedge T}) \right) \nabla_\phi V_\phi(s_0)$$

```
    update  $\phi$  using  $g$  with an optimizer
```

```
end
```

$V^\pi \approx V_\phi$ with k -step TD + (apx) SGD

Using the stop-gradient operator, express k -step TD with approximate SGD as

```
while (not converged)
  sample partial trajectory up to  $t = k \wedge T$  with  $(p_0, \pi, p)$ 
  # If  $s_{k \wedge T} == \text{term}$ , then  $V_\phi(s_{k \wedge T}) = 0$ 
  
$$g = \nabla_\phi \frac{1}{2} \left( V_\phi(s_0) - \sum_{t=0}^{(k \wedge T)-1} \gamma^t r_t - \gamma^{k \wedge T} \llbracket V_\phi(s_{k \wedge T}) \rrbracket \right)^2$$

  update  $\phi$  using  $g$  with an optimizer
end
```

(Here, the random sampling of r_t does not depend on ϕ , so $\partial r_t / \partial \phi = 0$. This will change when we perform policy optimization.)

MC vs. TD(=bootstrapping)

The goal is to train $V_\phi \approx V^\pi$. When we need to use V^π , we replace it with V_ϕ . This is called *bootstrapping*.

- MC evaluation updates V_ϕ by waiting for the episode to terminate.
- TD evaluation updates V_ϕ without waiting for the episode to terminate; they instead bootstrap and use V_ϕ .
- Initially, when V_ϕ is inaccurate, bootstrapping may cause instabilities.
- Later, when V_ϕ is accurate, waiting for the episode to terminate may be wasteful.

As a rule of thumb, an intermediate value of $k = 5$ is a good compromise.

$Q_\phi \approx Q^\pi$ with MC

Policy evaluation for Q^π follows the same principles.

For MC,

$$Q^\pi(s, a) = \mathbb{E}^\pi \left[\sum_{t=0}^{T-1} \gamma^t r_t \mid s_0 = s, a_0 = a \right] \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^{T^{(i)}-1} \gamma^t r_t^{(i)}$$

where

$$\left. \begin{array}{l} \tau^{(1)} = \left(s_0^{(1)}, a_0^{(1)}, r_0^{(1)}, \dots, s_{T^{(1)}-1}^{(1)}, a_{T^{(1)}-1}^{(1)}, r_{T^{(1)}-1}^{(1)}, s_{T^{(1)}}^{(1)} \right) \\ \vdots \\ \tau^{(N)} = \left(s_0^{(N)}, a_0^{(N)}, r_0^{(N)}, \dots, s_{T^{(N)}-1}^{(N)}, a_{T^{(N)}-1}^{(N)}, r_{T^{(N)}-1}^{(N)}, s_{T^{(N)}}^{(N)} \right) \end{array} \right\} \begin{array}{l} N \text{ independent trajectories} \\ \text{with } s_0^{(i)} = s \text{ and } a_0^{(i)} = a \end{array}$$

$Q_\phi \approx Q^\pi$ with MC

$$\underset{\phi \in \mathbb{R}^p}{\text{minimize}} \underbrace{\mathbb{E}_{\substack{s \sim p_0 \\ a \sim \pi(\cdot | s)}} \left[\frac{1}{2} (Q_\phi(s, a) - Q^\pi(s, a))^2 \right]}_{=\mathcal{L}(\phi)}$$

Gradient of \mathcal{L} :

$$\begin{aligned} \nabla_\phi \mathcal{L}(\phi) &= \mathbb{E}_{\substack{s \sim p_0 \\ a \sim \pi(\cdot | s)}} [(Q_\phi(s, a) - Q^\pi(s, a)) \nabla_\phi Q_\phi(s)] \\ &= \mathbb{E}_{\substack{s \sim p_0 \\ a \sim \pi(\cdot | s)}} \left[\left(Q_\phi(s, a) - \mathbb{E}^\pi \left[\sum_{t=0}^{T-1} \gamma^t r_t \mid s_0 = s, a_0 = a \right] \right) \nabla_\phi Q_\phi(s) \right] \\ &= \mathbb{E}_{\substack{s \sim p_0 \\ a \sim \pi(\cdot | s)}} \left[\mathbb{E}^\pi \left[\left(Q_\phi(s, a) - \sum_{t=0}^{T-1} \gamma^t r_t \right) \nabla_\phi Q_\phi(s, a) \mid s_0 = s, a_0 = a \right] \right] \\ &= \mathbb{E}_{s_0 \sim p_0}^\pi \left[\underbrace{\left(Q_\phi(s_0, a_0) - \sum_{t=0}^{T-1} \gamma^t r_t \right) \nabla_\phi Q_\phi(s_0, a_0)}_g \right] \end{aligned}$$

$Q_\phi \approx Q^\pi$ with MC + SGD

```
while (not converged)
```

```
    sample trajectory  $\tau \sim (p_0, \pi, p)$ 
```

$$g = \left(Q_\phi(s_0, a_0) - \sum_{t=0}^{T-1} \gamma^t r_t \right) \nabla_\phi Q_\phi(s_0, a_0)$$

```
    update  $\phi$  using  $g$  with an optimizer
```

```
end
```

$Q_\phi \approx Q^\pi$ with MC + SGD

```
while (not converged)
   $s_0 \sim p_0, a_0 \sim \pi(\cdot | s_0), t = 0$ 
  while ( $s_t \neq \text{<term>}$ )
     $a_t \sim \pi(\cdot | s_t)$ 
     $(r_t, s_{t+1}) \sim p(\cdot, \cdot | s_t, a_t)$ 
     $t = t + 1$ 
  end
   $T = t$ 
} sample trajectory  $\tau$ 


 $g = \left( Q_\phi(s_0, a_0) - \sum_{t=0}^{T-1} \gamma^t r_t \right) \nabla_\phi Q_\phi(s_0, a_0)$ 

update  $\phi$  using  $g$  with an optimizer
end
```


$Q_\phi \approx Q^\pi$ with TD + (apx) SGD

1-step TD works with the same principles.

$$\begin{aligned}
 Q^\pi(s, a) &= \mathbb{E}^\pi \left[r_0 + \gamma \sum_{t=1}^{T-1} \gamma^{t-1} r_t \mid s_0 = s, a_0 = a \right] \\
 &= \mathbb{E}^\pi \left[\mathbb{E}^\pi \left[r_0 + \gamma \sum_{t=1}^{T-1} \gamma^{t-1} r_t \mid s_0, a_0, r_0, s_1, a_1 \right] \mid s_0 = s, a_0 = a \right] \\
 &= \mathbb{E}^\pi \left[r_0 + \gamma Q^\pi(s_1, a_1) \mid s_0 = s, a_0 = a \right] \quad \left. \begin{array}{c} (s_0^{(1)}, a_0^{(1)}, r_0^{(1)}, s_1^{(1)}, a_1^{(1)}) \\ \vdots \\ (s_0^{(N)}, a_0^{(N)}, r_0^{(N)}, s_1^{(N)}, a_1^{(N)}) \end{array} \right\} \begin{array}{l} N \text{ independent 1-step transitions} \\ \text{and action } a_1^{(i)} \\ \text{with } s_0^{(i)} = s \text{ and } a_0^{(i)} = a \end{array} \\
 &\approx \frac{1}{N} \sum_{i=1}^N \left(r_0^{(i)} + \gamma \textcolor{red}{Q}^\pi(\textcolor{red}{s}_1^{(i)}, \textcolor{red}{a}_1^{(i)}) \right)
 \end{aligned}$$


 Exactly actionable only if Q^π is known.

$Q_\phi \approx Q^\pi$ with TD + (apx) SGD

$$\underset{\phi \in \mathbb{R}^p}{\text{minimize}} \underbrace{\mathbb{E}_{\substack{s \sim p_0 \\ a \sim \pi(\cdot | s)}} \left[\frac{1}{2} (Q_\phi(s, a) - Q^\pi(s, a))^2 \right]}_{=\mathcal{L}(\phi)}$$


Gradient of \mathcal{L} :

$$\begin{aligned} \nabla_\phi \mathcal{L}(\phi) &= \mathbb{E}_{\substack{s \sim p_0 \\ a \sim \pi(\cdot | s)}}^\pi \left[\left(Q_\phi(s, a) - \sum_{t=0}^{T-1} \gamma^t r_t \right) \nabla_\phi Q_\phi(s, a) \right] \\ &= \mathbb{E}_{\substack{s_0 \sim p_0 \\ a_0 \sim \pi(\cdot | s)}}^\pi \left[\mathbb{E}^\pi \left[\left(Q_\phi(s_0, a_0) - \sum_{t=0}^{T-1} \gamma^t r_t \right) \nabla_\phi Q_\phi(s_0, a_0) \mid s_0, a_0, r_0, s_1, a_1 \right] \right] \\ &= \mathbb{E}_{\substack{s_0 \sim p_0 \\ a_0 \sim \pi(\cdot | s_0) \\ (r_0, s_1) \sim p(\cdot, \cdot | s_0, a_0) \\ a_1 \sim \pi(\cdot | s_1)}} \left[\left(Q_\phi(s_0, a_0) - r_0 - \gamma \mathbb{E}^\pi \left[\sum_{t=1}^{T-1} \gamma^{t-1} r_t \mid s_0, a_0, r_0, s_1, a_1 \right] \right) \nabla_\phi Q_\phi(s_0, a_0) \right] \\ &= \mathbb{E}_{\substack{s_0 \sim p_0 \\ a_0 \sim \pi(\cdot | s_0) \\ (r_0, s_1) \sim p(\cdot, \cdot | s_0, a_0) \\ a_1 \sim \pi(\cdot | s_1)}} \left[\underbrace{(Q_\phi(s_0, a_0) - r_0 - \gamma Q^\pi(s_1, a_1))}_{g} \nabla_\phi Q_\phi(s_0, a_0) \right] \end{aligned}$$

$Q_\phi \approx Q^\pi$ with TD + (apx) SGD

SGD implementation

```
while (not converged)
   $s_0 \sim p_0, a_0 \sim \pi(\cdot | s_0), (r_0, s_1) \sim p(\cdot, \cdot | s_0, a_0), a_1 \sim \pi(\cdot | s_1)$ 
   $g = (Q_\phi(s_0, a_0) - r_0 - \gamma Q^\pi(s_1, a_1)) \nabla_\phi Q_\phi(s_0, a_0)$ 
  update  $\phi$  using  $g$  with an optimizer
end
```



Approximate SGD implementation (g is no longer an unbiased estimate of $\nabla \mathcal{L}$)

```
while (not converged)
   $s_0 \sim p_0, a_0 \sim \pi(\cdot | s_0), (r_0, s_1) \sim p(\cdot, \cdot | s_0, a_0), a_1 \sim \pi(\cdot | s_1)$ 
   $g = (Q_\phi(s_0, a_0) - r_0 - \gamma Q_\phi(s_1, a_1)) \nabla_\phi Q_\phi(s_0, a_0)$  # If  $s_1 == \text{<term>}$ , then  $Q_\phi(s_1, a_1) = 0$ 
  update  $\phi$  using  $g$  with an optimizer
end
```

$Q_\phi \approx Q^\pi$ with TD + (apx) SGD

Using the stop-gradient operator, express TD with approximate SGD as

```
while (not converged)
```

```
     $s_0 \sim p_0, a_0 \sim \pi(\cdot | s_0), (r_0, s_1) \sim p(\cdot, \cdot | s_0, a_0), a_1 \sim \pi(\cdot | s_1)$ 
```

```
     $g = \nabla_\phi \frac{1}{2} (Q_\phi(s_0, a_0) - r_0 - \gamma \llbracket Q_\phi(s_1, a_1) \rrbracket)^2$  # If  $s_1 == \text{<term>}$ , then  $Q_\phi(s_1, a_1) = 0$ 
```


```
    update  $\phi$  using  $g$  with an optimizer
```

```
end
```

k -step TD policy evaluation

k -step TD works with the same principles.

$$\begin{aligned}
 Q^\pi(s, a) &= \mathbb{E}^\pi \left[\sum_{t=0}^{(k \wedge T)-1} \gamma^t r_t + \gamma^{k \wedge T} Q^\pi(s_{k \wedge T}, a_{k \wedge T}) \mid s_0 = s, a_0 = a \right] \\
 &= \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^{k \wedge T^{(i)}} \gamma^t r_t^{(i)} + \gamma^{k \wedge T^{(i)}} Q^\pi(s_{k \wedge T^{(i)}}^{(i)}, a_{k \wedge T^{(i)}}^{(i)})
 \end{aligned}$$

 Exactly actionable only if Q^π is known.

$$\left. \begin{array}{c}
 (s_0^{(1)}, a_0^{(1)}, r_0^{(1)}, \dots, s_{k \wedge T^{(1)}-1}^{(1)}, a_{k \wedge T^{(1)}-1}^{(1)}, r_{k \wedge T^{(1)}-1}^{(1)}, s_{k \wedge T^{(1)}}^{(1)}) \\
 \vdots \\
 (s_0^{(N)}, a_0^{(N)}, r_0^{(N)}, \dots, s_{k \wedge T^{(N)}-1}^{(N)}, a_{k \wedge T^{(N)}-1}^{(N)}, r_{k \wedge T^{(N)}-1}^{(N)}, s_{k \wedge T^{(N)}}^{(N)})
 \end{array} \right\} \begin{array}{l}
 N \text{ independent partial trajectories} \\
 \text{with } s_0^{(i)} = s \text{ and } a_0^{(i)} = a
 \end{array}$$

$Q_\phi \approx Q^\pi$ with k -step TD + SGD

$$\underset{\phi \in \mathbb{R}^p}{\text{minimize}} \underbrace{\mathbb{E}_{\substack{s \sim p_0 \\ a \sim \pi(\cdot | s)}} \left[\frac{1}{2} (Q_\phi(s, a) - Q^\pi(s, a))^2 \right]}_{=\mathcal{L}(\phi)}$$

Gradient of \mathcal{L} :

$$\begin{aligned} \nabla_\phi \mathcal{L}(\phi) &= \mathbb{E}_{\substack{s \sim p_0 \\ a \sim \pi(\cdot | s)}} [(Q_\phi(s, a) - Q^\pi(s, a)) \nabla_\phi Q_\phi(s, a)] \\ &= \mathbb{E}_{s_0 \sim p_0}^\pi \left[\underbrace{\left(Q_\phi(s_0, a_0) - \sum_{t=0}^{(k \wedge T)-1} \gamma^t r_t - \gamma^{k \wedge T} Q^\pi(s_{k \wedge T}, a_{k \wedge T}) \right)}_g \nabla_\phi Q_\phi(s_0, a_0) \right] \end{aligned}$$

$Q_\phi \approx Q^\pi$ with k -step TD + SGD

while (not converged)

sample partial trajectory up to $s_{k \wedge T}$ and $a_{k \wedge T}$ with (p_0, π, p)

$$g = \left(Q_\phi(s_0, a_0) - \sum_{t=0}^{(k \wedge T)-1} \gamma^t r_t - \gamma^{k \wedge T} Q^\pi(s_{k \wedge T}, a_{k \wedge T}) \right) \nabla_\phi Q_\phi(s_0, a_0)$$

update ϕ using g with an optimizer

not actionable



end

while (not converged)

$s_0 \sim p_0, t = 0, T = \infty$

for $t = 0, \dots, k - 1$

if $s_t == \text{<term>}$

$T = t$

break

end

$a_t \sim \pi(\cdot | s_t)$

$(r_t, s_{t+1}) \sim p(\cdot, \cdot | s_t, a_t)$

end

sample partial trajectory
up to $s_{k \wedge T}$ and $a_{k \wedge T}$

$$y = \sum_{t=0}^{(k \wedge T)-1} \gamma^t r_t$$

if $s_{k \wedge T} != \text{<term>}$ # If $k < T$

$a_k \sim \pi(\cdot | s_k)$

$y \ += \ \gamma^k Q^\pi(s_k, a_k)$ ← not actionable

end

$g = (Q_\phi(s_0, a_0) - y) \nabla_\phi Q_\phi(s_0, a_0)$

update ϕ using g with an optimizer

end

$Q_\phi \approx Q^\pi$ with
 k -step TD + SGD

$Q_\phi \approx Q^\pi$ with k -step TD + (apx) SGD

Approximate SGD implementation (g is no longer an unbiased estimate of $\nabla \mathcal{L}$)

```
while (not converged)
  sample partial trajectory up to  $t = k \wedge T$  with  $(p_0, \pi, p)$ 
   $y = \sum_{t=0}^{(k \wedge T)-1} \gamma^t r_t$ 
  if  $s_{k \wedge T} \neq \text{term}$  # If  $k < T$ 
     $a_k \sim \pi(\cdot | s_k)$ 
     $y += \gamma^k Q_\phi(s_k, a_k)$ 
  end
   $g = (Q_\phi(s_0, a_0) - y) \nabla_\phi Q_\phi(s_0, a_0)$ 
  update  $\phi$  using  $g$  with an optimizer
end
```

$Q_\phi \approx Q^\pi$ with k -step TD + (apx) SGD

Using the stop-gradient operator, express k -step TD with approximate SGD as

```
while (not converged)
  sample partial trajectory up to  $t = k \wedge T$  with  $(p_0, \pi, p)$ 
   $y = \sum_{t=0}^{(k \wedge T)-1} \gamma^t r_t$ 
  if  $s_{k \wedge T} \neq \text{term}$  # If  $k < T$ 
     $a_k \sim \pi(\cdot | s_k)$ 
     $y += \gamma^k Q_\phi(s_k, a_k)$ 
  end
   $g = \nabla_\phi \frac{1}{2} (Q_\phi(s_0, a_0) - \llbracket y \rrbracket)^2$ 
  update  $\phi$  using  $g$  with an optimizer
end
```

Deep Policy Gradient Methods: A3C, TRPO, PPO, GRPO

Policy optimization

So far, we talked about policy evaluation: approximating V^π and Q^π given a policy π .

Next, let's talk about policy optimization: solving

$$\underset{\theta}{\text{maximize}} \quad \mathcal{J}(\theta) = \mathbb{E}_{s_0 \sim p_0} [V^{\pi_\theta}(s_0)] = \mathbb{E}_{s_0 \sim p_0}^\pi \left[\sum_{t=0}^{T-1} \gamma^t r_t \right]$$

where π_θ is represented by a neural network with parameter θ .

Expectation with trajectory τ

For policy gradient methods, Assume $T < \infty$ with probability 1.

Write $\tau = (s_0, a_0, r_0, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T)$ for the trajectory and use the notation

$$\begin{aligned}\mathcal{J}(\theta) &= \mathbb{E}_{s_0 \sim p_0} \left[V^{\pi_\theta}(s_0) \right] = \mathbb{E}_{\substack{s_0 \sim p_0 \\ a_t \sim \pi_\theta(\cdot | s_t) \\ (r_t, s_{t+1}) \sim p(\cdot, \cdot | s_t, a_t)}} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right] \\ &= \mathbb{E}_{\tau \sim (p_0, \pi_\theta, p)} \left[G_0(\tau) \right]\end{aligned}$$

The probability distribution of τ can be written as

$$p_\theta(\tau) = p_0(s_0) \prod_{t=0}^{T-1} p(r_t, s_{t+1} | s_t, a_t) \pi_\theta(a_t | s_t).$$

Policy after stopping

For notational convenience, let $\tilde{a} \in \mathcal{A}$ be a certain action and let

$$\pi_{\theta}(a = \tilde{a} \mid s = \text{<term>}) = 1$$

I.e., once we reach the terminal state $s = \text{<term>}$, always take the action \tilde{a} .

(Remember, actions are irrelevant once we reach <term> .)

Then,

$$p_{\theta}(\tau) = p_0(s_0) \prod_{t=0}^{T-1} p(r_t, s_{t+1} \mid s_t, a_t) \pi_{\theta}(a_t \mid s_t) = p_0(s_0) \prod_{t=0}^{\infty} p(r_t, s_{t+1} \mid s_t, a_t) \pi_{\theta}(a_t \mid s_t)$$

Stochastic gradient of objective $\mathcal{J}(\theta)$

$$\begin{aligned}\nabla \mathcal{J}(\theta) &= \nabla_{\theta} \mathbb{E}_{\tau \sim (p_0, \pi_{\theta}, p)} [G_0(\tau)] \\ &= \sum_{\tau} G_0(\tau) \nabla_{\theta} p_{\theta}(\tau) \\ &= \sum_{\tau} p_{\theta}(\tau) G_0(\tau) \nabla_{\theta} \log p_{\theta}(\tau) \\ &= \mathbb{E}_{\tau \sim (p_0, \pi_{\theta}, p)} \left[\underbrace{G_0(\tau) \nabla_{\theta} \log p_{\theta}(\tau)}_{=g(\tau)} \right]\end{aligned}$$

So $g(\tau)$ with $\tau \sim (p_0, \pi_{\theta}, p)$ is an unbiased estimator of $\nabla \mathcal{J}(\theta)$.

However, current $g(\tau)$ has large variance, so we must reduce it.

Enhancement #1: Removing past rewards

We can reduce the variance by removing past rewards until time $t - 1$:

$$\begin{aligned}
 \nabla_{\theta} \mathcal{J}(\theta) &= \mathbb{E}_{\tau \sim (p_0, \pi_{\theta}, p)} [\nabla_{\theta} \log p_{\theta}(\tau) G_0(\tau)] \\
 &= \mathbb{E}_{\tau \sim (p_0, \pi_{\theta}, p)} \left[\left(\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right) G_0(\tau) \right] \\
 &= \mathbb{E}_{\tau \sim (p_0, \pi_{\theta}, p)} \left[\sum_{t=0}^{T-1} \left(\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) G_0(\tau) \right) \right] = \mathbb{E}_{\tau \sim (p_0, \pi_{\theta}, p)} \left[\sum_{t=0}^{T-1} \left(\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \sum_{t'=0}^{T-1} \gamma^{t'} r_{t'} \right) \right] \\
 &= \mathbb{E}_{\tau \sim (p_0, \pi_{\theta}, p)} \left[\sum_{t=0}^{T-1} \left(\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \left(\sum_{t'=0}^{t-1} \gamma^{t'} r_{t'} + \gamma^t \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} \right) \right) \right] \\
 &\stackrel{(*)}{=} \mathbb{E}_{\tau \sim (p_0, \pi_{\theta}, p)} \left[\sum_{t=0}^{T-1} \left(\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \gamma^t \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} \right) \right] \quad \left(p_{\theta}(\tau) = p_0(s_0) \prod_{t=0}^{\infty} p(r_t, s_{t+1} | s_t, a_t) \pi_{\theta}(a_t | s_t) \right) \\
 &= \mathbb{E}_{\tau \sim (p_0, \pi_{\theta}, p)} \left[\underbrace{\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \gamma^t G_t}_{=g(\tau)} \right]
 \end{aligned}$$

Justification of (*)

Intuitively, ∇_{θ} here examines how the probabilities for a_t should change. However, a change of a_t does not affect the past rewards r_1, \dots, r_{t-1} . Past rewards only contribute to unnecessary variance.

Let $\tau^{(t)} = (s_0, a_0, r_0, \dots, s_{t-1}, a_{t-1}, r_{t-1}, s_t)$. Then

$$\begin{aligned} \mathbb{E}_{\tau \sim (p_0, \pi_{\theta}, p)} \left[\underbrace{\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \sum_{t'=0}^{t-1} \gamma^{t'} r_{t'}}_{\stackrel{\text{def}}{=} Y_t} \middle| \tau^{(t)} \right] &= \mathbb{E}_{a_t \sim \pi_{\theta}(\cdot | s_t)} \left[\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \sum_{t'=0}^{t-1} \gamma^{t'} r_{t'} \middle| \tau^{(t)} \right] \\ &= \sum_{a_t} \nabla_{\theta} \pi_{\theta}(a_t | s_t) \sum_{t'=0}^{t-1} \gamma^{t'} r_{t'} \\ &= \left(\sum_{t'=0}^{t-1} \gamma^{t'} r_{t'} \right) \nabla_{\theta} \sum_{a_t} \pi_{\theta}(a_t | s_t) \\ &= \left(\sum_{t'=0}^{t-1} \gamma^{t'} r_{t'} \right) \nabla_{\theta} 1 = 0 \end{aligned}$$

Here, a_t is the only random term since s_t, r_0, \dots, r_{t-1} is deterministic conditioned on $\tau^{(t)}$.

Justification of $(*)$

By the tower property,

$$Y_t = \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \sum_{t'=0}^{t-1} \gamma^{t'} r_{t'}$$

has expectation 0 (with respect to $\mathbb{E}_{\tau \sim (p_0, \pi_{\theta}, p)} [\dots]$), so

$$\mathbb{E}_{\tau \sim (p_0, \pi_{\theta}, p)} \left[\sum_{t=1}^{\infty} Y_t \right] = 0$$

■

Let us further reduce the variance.

Enhancement #2: State-dependent baseline

Let $b : \mathcal{S} \rightarrow \mathbb{R}$ be a state-dependent baseline.

Then,

$$\begin{aligned} g(\tau) &= \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \gamma^t (G_t - b(s_t)) \\ &= \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \gamma^t \left(\sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} - b(s_t) \right) \end{aligned}$$

is an unbiased estimator of $\nabla J(\theta)$.

Why unbiased?

Let $\tau^{(t)} = (s_0, a_0, r_0, \dots, s_{t-1}, a_{t-1}, r_{t-1}, s_t)$. Then

$$\begin{aligned}\mathbb{E}_{\tau \sim (p_0, \pi_\theta, p)} \left[\nabla_\theta \log \pi_\theta(a_t | s_t) b(s_t) \mid \tau^{(t)} \right] &= \mathbb{E}_{\tau \sim (p_0, \pi_\theta, p)} \left[\nabla_\theta \log \pi_\theta(a_t | s_t) \mid \tau^{(t)} \right] b(s_t) \quad (s_t \text{ is fixed conditioned on } \tau^{(t)}) \\ &= \mathbb{E}_{a_t \sim \pi_\theta(\cdot | s_t)} \left[\nabla_\theta \log \pi_\theta(a_t | s_t) \mid s_t \right] b(s_t) \quad (a_t \text{ is the only random term}) \\ &= \left(\nabla_\theta \underbrace{\sum_{a_t \in \mathcal{A}} \pi_\theta(a_t | s_t)}_{=1} \right) b(s_t) \quad (\text{sum over all } a_t \text{ has probability 1}) \\ &= 0\end{aligned}$$

By the tower property, the following is a sum of zero-mean random variables

$$\mathbb{E}_{\tau \sim (p_0, \pi_\theta, p)} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) b(s_t) \right] = \mathbb{E}_{\tau \sim (p_0, \pi_\theta, p)} \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t | s_t) b(s_t) \right] = 0$$

Enhancement #3: Q-estimates

Theorem) Let $\{\hat{Q}_t\}_{t=0}^{T-1}$ be a random variable such that

$$\mathbb{E}^{\pi_\theta} \left[\hat{Q}_t \mid \tau^{(t)}, a_t \right] = Q^{\pi_\theta}(s_t, a_t)$$

Let $b(s)$ be any (measurable) deterministic function of $s \in \mathcal{S}$. Then

$$\nabla \mathcal{J}(\theta) = \mathbb{E}_{\tau \sim (p_0, \pi_\theta, p)} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \gamma^t (\hat{Q}_t - b(s_t)) \right]$$

(Note. Previous enhancements are all instances of this theorem.)

Proof) We already established

$$\mathbb{E}_{\tau \sim (p_0, \pi_\theta, p)} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) \gamma^t b(s_t) \right] = 0$$

Next,

$$\nabla \mathcal{J}(\theta) = \mathbb{E}_{\tau \sim (p_0, \pi_\theta, p)} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) \gamma^t G_t \right] = \mathbb{E}_{\tau \sim (p_0, \pi_\theta, p)} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) \gamma^t \hat{Q}_t \right]$$

follows from

$$\begin{aligned} \mathbb{E} \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t | s_t) \gamma^t (G_t - \hat{Q}_t) \right] &= \sum_{t=0}^{\infty} \mathbb{E} \left[\nabla_\theta \log \pi_\theta(a_t | s_t) \gamma^t (G_t - \hat{Q}_t) \right] \\ &= \sum_{t=0}^{\infty} \mathbb{E} \left[\mathbb{E} \left[\nabla_\theta \log \pi_\theta(a_t | s_t) \gamma^t (G_t - \hat{Q}_t) \mid \tau^{(t)}, a_t \right] \right] \\ &= \sum_{t=0}^{\infty} \mathbb{E} \left[\nabla_\theta \log \pi_\theta(a_t | s_t) \gamma^t \mathbb{E} \left[G_t - \hat{Q}_t \mid \tau^{(t)}, a_t \right] \right] \\ &= \sum_{t=0}^{\infty} \mathbb{E} \left[\nabla_\theta \log \pi_\theta(a_t | s_t) \gamma^t (Q^{\pi_\theta}(s_t, a_t) - Q^{\pi_\theta}(s_t, a_t)) \right] \\ &= 0 \end{aligned}$$

Remember,

$$\tau^{(t)} = (s_0, a_0, r_0, \dots, s_{t-1}, a_{t-1}, r_{t-1}, s_t)$$

Enhancement #3: Q-estimates

With the choice $\hat{Q}_t = Q^{\pi_\theta}$ and $b = V_\phi$

$$g(\tau) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) \gamma^t (Q^{\pi_\theta}(s_t, a_t) - V_\phi(s_t))$$

Exactly actionable only if Q^{π_θ} is known.

is an unbiased estimator of $\nabla J(\theta)$ by the policy gradient theorem. Here, $\tau \sim (p_0, \pi_\theta, p)$ and V_ϕ is a neural network approximating V^{π_θ} .

(This is an unbiased estimate regardless of the accuracy of $V_\phi \approx V^{\pi_\theta}$. However, we must use the exact Q^{π_θ} to have exact unbiasedness.)

This choice $\hat{Q}_t = Q^{\pi_\theta}$ and $b = V_\phi$ leads to small (but not optimally small) variance. Why?

Rao–Blackwell Theorem

Theorem) Let X and Y be random variables. Let $\hat{I}_1(X, Y)$ be an unbiased estimator of I , i.e.,

$$I = \mathbb{E}_{X,Y} [\hat{I}_1(X, Y)]$$

Let $\hat{I}_2(Y) = \mathbb{E}_{X|Y} [\hat{I}_1(X, Y) | Y]$. Then \hat{I}_2 is also an unbiased estimator of I and

$$\text{Var}(\hat{I}_2) \leq \text{Var}(\hat{I}_1).$$

$\hat{I}_2(Y)$ is called a *Rao–Blackwellized* estimator of $\hat{I}_1(X, Y)$.

This motivates $\hat{Q}_t = Q^{\pi_\theta}$ with $Y = (\tau^{(t)}, a_t)$ and $X = (r_{t+1}, s_{t+1}, a_{t+1}, \dots)$, i.e., $Q^{\pi_\theta}(s_t, a_t)$ is a good choice as it has all variance beyond s_t, a_t removed through conditional expectation and therefore has low variance.

Proof) Unbiasedness of \hat{I}_2 follows from the tower property of expectation.

Variance bound follows from Jensen's inequality:

$$\begin{aligned}\text{Var}(\hat{I}_2) &= \mathbb{E}_Y \left[(I - \hat{I}_2(Y))^2 \right] \\ &= \mathbb{E}_Y \left[\left(\mathbb{E}_{X|Y} \left[I - \hat{I}_1(X, Y) \mid Y \right] \right)^2 \right] \\ &\leq \mathbb{E}_Y \left[\mathbb{E}_{X|Y} \left[(I - \hat{I}_1(X, Y))^2 \mid Y \right] \right] \\ &= \mathbb{E}_{X,Y} \left[(I - \hat{I}_1(X, Y))^2 \right] \\ &= \text{Var}(\hat{I}_1)\end{aligned}$$

■

Minimum-variance conditional estimator lemma

Lemma) Let s and a be random variables. Let $w(s, a)$, $Q(s, a)$, and $b(s)$ be functions. Then

$$\operatorname{argmin}_{b(\cdot)} \mathbb{E}_{a,s} [w^2(s, a)(Q(s, a) - b(s))^2] = b^*$$

with

$$b^*(s) = \frac{\mathbb{E}_{a|s} [w^2(s, a)Q(s, a) | s]}{\mathbb{E}_{a|s} [w^2(s, a) | s]}$$

(When $w(s, a) = 1$, this lemma reduces to a standard result in Bayesian statistics: Conditional mean is the minimum mean squared error estimator. In fact, if $\rho(s, a)$ is a probability density function of (s, a) , then $b^*(s)$ is the conditional mean of $Q(s, a)$ with respect to the probability distribution proportional to $w^2\rho$.)

Proof)

$$\begin{aligned}
\mathbb{E}_{a,s} [w^2(s, a)(Q(s, a) - b(s))^2] &= \mathbb{E}_{a,s} [w^2(s, a)(Q(s, a) - b^*(s) + b^*(s) - b(s))^2] \\
&= \mathbb{E}_{a,s} [w^2(s, a)(Q(s, a) - b^*(s))^2] + \mathbb{E}_{a,s} [w^2(s, a)(b^*(s) - b(s))^2] \\
&\quad + 2 \mathbb{E}_s \left[\mathbb{E}_{a|s} [w^2(s, a)(Q(s, a) - b^*(s))(b^*(s) - b(s)) | s] \right] \\
&\stackrel{(*)}{=} \mathbb{E}_{a,s} [w^2(s, a)(Q(s, a) - b^*(s))^2] + \mathbb{E}_{a,s} [w^2(s, a)(b^*(s) - b(s))^2] \\
&\geq \mathbb{E}_{a,s} [w^2(s, a)(Q(s, a) - b^*(s))^2]
\end{aligned}$$

with equality attained at $b(s) = b^*(s)$. The step $(*)$ follows from the definition of $b^*(s)$:

$$\begin{aligned}
&\mathbb{E}_{a|s} [w^2(s, a)(Q(s, a) - b^*(s))(b^*(s) - b(s)) | s] \\
&= \left(\mathbb{E}_{a|s} [w^2(s, a)Q(s, a) | s] - \mathbb{E}_{a|s} [w^2(s, a) | s] b^*(s) \right) (b^*(s) - b(s)) \\
&= 0
\end{aligned}$$

■

Why $b = V_\phi$?

The minimum-variance conditional estimator lemma suggests the choice

$$b^*(s) = \frac{\mathbb{E}_{a \sim \pi_\theta(\cdot | s)} \left[(\nabla_\theta \log \pi_\theta(a_t | s_t))^2 Q^{\pi_\theta}(s, a) \mid s \right]}{\mathbb{E}_{a \sim \pi_\theta(\cdot | s)} \left[(\nabla_\theta \log \pi_\theta(a_t | s_t))^2 \mid s \right]}$$

However, this choice is cumbersome as it is an entirely new quantity not used later in the estimation of $Q^{\pi_\theta}(s_t, a_t)$. Therefore, use the simplified surrogate

$$\frac{\mathbb{E}_{a \sim \pi_\theta(\cdot | s)} \left[\cancel{(\nabla_\theta \log \pi_\theta(a_t | s_t))^2} Q^{\pi_\theta}(s, a) \mid s \right]}{\mathbb{E}_{a \sim \pi_\theta(\cdot | s)} \left[\cancel{(\nabla_\theta \log \pi_\theta(a_t | s_t))^2} \mid s \right]} \Rightarrow b(s) = \frac{\mathbb{E}_{a \sim \pi_\theta(\cdot | s)} [Q^{\pi_\theta}(s, a) \mid s]}{\mathbb{E}_{a \sim \pi_\theta(\cdot | s)} [1 \mid s]} = V^{\pi_\theta}(s)$$

The choice $b = V_\phi \approx V^{\pi_\theta}$ is not optimal, but it is a reasonable proxy of the optimal choice.

Interpretation via advantage estimation

So, we have

$$\nabla \mathcal{J}(\theta) = \mathbb{E}_{\tau \sim (p_0, \pi_\theta, p)} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) \gamma^t (Q^{\pi_\theta}(s_t, a_t) - V_\phi(s_t)) \right]$$

$A^{\pi_\theta}(s_t, a_t) = Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)$ is called the *advantage* of a_t at s_t .

- If $A^{\pi_\theta}(s_t, a_t) > 0$, then a_t is a good action, better than the average action of π_θ .
- If $A^{\pi_\theta}(s_t, a_t) < 0$, then a_t is a bad action, worse than the average action of π_θ .

This gradient estimate uses $Q^{\pi_\theta}(s_t, a_t) - V_\phi(s_t)$, an approximation of the advantage.

- If a_t is good, ∇_θ points in a direction to make a_t more likely.
- If a_t is bad, ∇_θ points in a direction to make a_t less likely

Interpretation via advantage estimation

Aside: For an optimal policy π^* ,

$$A^{\pi^*}(s_t, a_t) = Q^{\pi^*}(s_t, a_t) - V^{\pi^*}(s_t) \leq 0$$

Proof) Hw assignment. ■

Without the baseline,

$$\nabla \mathcal{J}(\theta) = \mathbb{E}_{\tau \sim (p_0, \pi_\theta, p)} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) \gamma^t Q^{\pi_\theta}(s_t, a_t) \right]$$

the sign of $Q^{\pi_\theta}(s_t, a_t)$ is not directly correlated with whether a_t is good or bad. (In fact, many MDPs only have positive rewards and thus have $Q^{\pi_\theta}(s_t, a_t) \geq 0$.) Rather, ∇_θ will adjust the probability of a_t , and a_t becomes more likely by being pushed up harder than the other actions due to the normalization of $\pi_\theta(\cdot | s_t)$.

With the baseline, a_t should be push up only when a_t is good.

Interpretation as an actor-critic method

Loosely speaking *actor-critic* methods have an “actor” (i.e., policy π_θ) and a separate “critic” that evaluates the actor’s action.

In policy gradient methods, the learned state-value function V_ϕ serves as the critic.

$$\nabla \mathcal{J}(\theta) = \mathbb{E}_{\tau \sim (p_0, \pi_\theta, p)} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) \gamma^t (Q^{\pi_\theta}(s_t, a_t) - V_\phi(s_t)) \right]$$

Replacing Q^{π_θ} with Q_ϕ ?

Should we replace Q^{π_θ} with $Q_\phi \approx Q^{\pi_\theta}$?

Exactly actionable only if Q^{π_θ} is known.

$$\begin{aligned}\nabla \mathcal{J}(\theta) &= \mathbb{E}_\tau \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) \gamma^t \left(\overset{\text{red}}{Q^{\pi_\theta}}(s_t, a_t) - V_\phi(s_t) \right) \right] \\ &\approx \underbrace{\mathbb{E}_\tau \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) \gamma^t \left(Q_\phi(s_t, a_t) - V_\phi(s_t) \right) \right]}_{=g}\end{aligned}$$

Then g is a biased estimator of $\nabla_\theta \mathcal{L}(\theta)$. This is possible, but a few problems:

- We must learn *both* Q_ϕ and V_ϕ .
- Replacing G_t with $Q_\phi(s_t, a_t)$ will reduce the variance, but the bias will initially be large since $Q_\phi \neq Q^{\pi_\theta}$ initially.

Enhancement #4: k -step TD

Use k -step TD

$$\begin{aligned}\hat{Q}_t &= r_t + \gamma r_{t+1} + \cdots + \gamma^{k \wedge (T-t)-1} r_{(t+k) \wedge T-1} + \gamma^{k \wedge (T-t)} \mathbf{V}^{\pi_\theta}(\mathbf{s}_{(t+k) \wedge T}) \quad \# \text{ unbiased estimate of } Q^{\pi_\theta}(a_t, s_t) \\ &\approx r_t + \gamma r_{t+1} + \cdots + \gamma^{k \wedge (T-t)-1} r_{(t+k) \wedge T-1} + \gamma^{k \wedge (T-t)} V_\phi(\mathbf{s}_{(t+k) \wedge T}) \quad \# \text{ biased estimate of } Q^{\pi_\theta}(a_t, s_t)\end{aligned}$$

So with $\tau \sim (p_0, \pi_\theta, p)$

$$g(\tau) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) \gamma^t \left(\left(\sum_{t'=0}^{k \wedge (T-t)-1} \gamma^{t'} r_{t'+t} \right) + \gamma^{k \wedge (T-t)} \mathbf{V}^{\pi_\theta}(\mathbf{s}_{(t+k) \wedge T}) - V_\phi(s_t) \right)$$

is an unbiased estimator of $\nabla J(\theta)$, but is not implementable without V^{π_θ} .

$$\mathbf{g}(\tau) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) \gamma^t \left(\left(\sum_{t'=0}^{k \wedge (T-t)-1} \gamma^{t'} r_{t'+t} \right) + \gamma^{k \wedge (T-t)} V_\phi(\mathbf{s}_{(t+k) \wedge T}) - V_\phi(s_t) \right)$$

is a **biased** estimator of $\nabla J(\theta)$.

Enhancement #4: k -step TD

Now, we only need to learn $V_\phi \approx V^{\pi_\theta}$. No need to separately learn Q_ϕ .

Even when V_ϕ is initially wrong, then

$$\hat{Q}_t = r_t + \gamma r_{t+1} + \dots + \gamma^{k-1} r_{t+k-1} + \gamma^k V_\phi(s_{t+k})$$

is still reasonably informative since the rewards r_t, \dots, r_{t+k-1} provide informative unbiased information of the quality of the action a_t , even if $V_\phi(s_{t+k})$ has large bias and is completely non-informative. (Especially so when $\gamma < 1$.)

Policy Gradient Algorithm #1

```
while (not converged)
   $g_\theta = 0, g_\phi = 0$ 
  sample trajectory  $\tau \sim (p_0, \pi_\theta, p)$ 
  for  $t = 0, 1, \dots, T - 1$ 
    # If  $s_{(t+k) \wedge T} == \text{<term>}$ , then  $V_\phi(s_{(t+k) \wedge T}) = 0$ 
     $\hat{Q} = \sum_{t'=0}^{k \wedge (T-t)-1} \gamma^{t'} r_{t'+t} + \gamma^{k \wedge (T-t)} V_\phi(s_{(t+k) \wedge T})$ 
     $g_\theta += -(\nabla_\theta \log \pi_\theta(a_t | s_t)) \gamma^t (\hat{Q} - V_\phi(s_t))$  # ascent for  $\theta$  and descent for  $\phi$ 
     $g_\phi += \nabla_\phi \frac{1}{2} (\mathbb{E}[\hat{Q}] - V_\phi(s_t))^2$  # do not backprop  $\hat{Q}$  with respect to  $\phi$ 
  end
  update  $\theta$  and  $\phi$  using  $g_\theta$  and  $g_\phi$  with an optimizer
end
```

Stop-gradient operator for g_θ

For ease of practical implementation, use the stop-gradient operator for g_θ :

$$\begin{aligned}\hat{Q} &= \sum_{t'=0}^{k \wedge (T-t)-1} \gamma^{t'} r_{t'+t} + \gamma^{k \wedge (T-t)} V_\phi(s_{(t+k) \wedge T}) \\ g_\theta &\leftarrow g_\theta + (\nabla_\theta \log \pi_\theta(a_t | s_t)) \gamma^t (\hat{Q} - V_\phi(s_t)) \\ &= g_\theta + \nabla_\theta \left(\gamma^t [\hat{Q} - V_\phi(s_t)] \log \pi_\theta(a_t | s_t) \right)\end{aligned}$$

Note that sampling of r_0, \dots, r_{T-1} and s_1, \dots, s_T does depend on π_θ . Therefore, it is dubious to claim $\partial \hat{Q} / \partial \theta = 0$ and $\partial V_\phi(s_t) / \partial \theta = 0$. It is best to avoid these derivatives by through the stop-gradient operator.

The γ -trick

When $\gamma = 1$,

$$\nabla \mathcal{J}(\theta) = \mathbb{E}_{\tau \sim (p_0, \pi_\theta, p)} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) (\hat{Q}_t - b(s_t)) \right]$$

where $\mathbb{E}^{\pi_\theta} [\hat{Q}_t | s_t, a_t] = Q^{\pi_\theta}(s_t, a_t) = \mathbb{E}^{\pi_\theta} \left[\sum_{t'=t}^{T-1} r_{t'} \mid s_t, a_t \right]$

Let $\tilde{\gamma} < 1$ but $\tilde{\gamma} \approx 1$. Let $\hat{Q}_t^{\tilde{\gamma}}$ be a random variable such that

$$\mathbb{E}^{\pi_\theta} [\hat{Q}_t^{\tilde{\gamma}} | s_t, a_t] = Q^{\pi_\theta, \tilde{\gamma}}(s_t, a_t) \stackrel{\text{def}}{=} \mathbb{E}^{\pi_\theta, \tilde{\gamma}} \left[\sum_{t'=t}^{T-1} \tilde{\gamma}^{t'} r_{t'} \mid s_t, a_t \right]$$

Then

$$\nabla \mathcal{J}(\theta) \approx \mathbb{E}_{\tau \sim (p_0, \pi_\theta, p)} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) (\hat{Q}_t^{\tilde{\gamma}} - b(s_t)) \right]$$

The γ -trick

Introducing the artificial discount factor $\tilde{\gamma}$ (not part of the MDP) introduces bias in the gradient estimates but can reduce the variance.

Most deep RL setups consider the undiscounted problem (so $\gamma = 1$) but introduces the artificial discount factor (so $\tilde{\gamma} < 1$).

For notational simplicity, we will not distinguish γ , the discount factor of the MDP, from $\tilde{\gamma}$, the artificial discount factor, and write $\gamma = \tilde{\gamma}$.

Policy Gradient Algorithm #2

while (not converged)


$g_\theta = 0, g_\phi = 0$

sample trajectory $\tau \sim (p_0, \pi_\theta, p)$

for $t = 0, 1, \dots, T - 1$

If $s_{(t+k) \wedge T} == \text{term}$, then $V_\phi(s_{(t+k) \wedge T}) = 0$

$$\hat{Q} = \sum_{t'=0}^{k \wedge (T-t)-1} \gamma^{t'} r_{t'+t} + \gamma^{k \wedge (T-t)} V_\phi(s_{(t+k) \wedge T})$$

$g_\theta += -(\nabla_\theta \log \pi_\theta(a_t | s_t)) (\hat{Q} - V_\phi(s_t))$  no γ factor here

$g_\phi += \nabla_\phi \frac{1}{2} (\hat{Q} - V_\phi(s_t))^2$ # do not backprop \hat{Q} with respect to ϕ

end

update θ and ϕ using g_θ and g_ϕ with an optimizer

end

SGD with non-uniform selection rules

Let

$$\nabla F(\theta) = \frac{1}{N} \sum_{i=1}^N g_i$$

where N is fixed (non-random). If $i(k) \sim \text{Uniform}(\{1, \dots, N\})$, then $g_{i(k)}$ is a stochastic gradient (up to a factor of $1/N$) and

$$\begin{aligned} i(k) &\sim \text{Uniform}(\{1, \dots, N\}) \\ \theta^{k+1} &= \theta^k - \alpha_k g_{i(k)} \end{aligned}$$

is an instance of SGD with unbiased stochastic gradients.

However, *cyclic SGD*

$$\theta^{k+1} = \theta^k - \alpha_k g_{\text{mod}(k,N)+1}$$

which gradient selection $g_1, \dots, g_N, g_1, \dots, g_N, \dots$ is *not* an instance SGD with unbiased stochastic gradients. Nevertheless, cyclic SGD is commonly used in practice.

SGD with non-uniform selection rules

In the context of MDPs, we have $\nabla \mathcal{L}(\theta) = \mathbb{E} \left[\sum_{t=0}^{T-1} g_t \right]$

with a random T . Then

sample τ # gets us T

$$t(k) \sim \text{Uniform}(\{0, \dots, T-1\})$$

$$\theta^{k+1} = \theta^k - \alpha_k T g_{t(k)}$$

is an instance of SGD with unbiased stochastic gradients, but it is an inefficient algorithm as only one of g_0, \dots, g_{T-1} is used in the update.

SGD with non-uniform selection rules

We can use all of g_1, \dots, g_T in the update:

sample τ # gets us T

$$\theta^{k+1} = \theta^k - \alpha_k \sum_{t=0}^{T-1} g_t$$

This is an instance of SGD with unbiased stochastic gradients, but it is inefficient as it updates infrequently. (In supervised learning, full-batch GD is less efficient than SGD.)

Finally,

sample (s_t, a_t, r_t, s_{t+1}) and compute g_t # (sample k steps if k -step TD)

$$\theta^{k+1} = \theta^k - \alpha_k g_t$$

is *not* an instance of SGD, but it updates the policy frequently and therefore is efficient despite using biased gradients.

Advantage actor-critic (A2C)

```
while (not converged)
   $g_\theta = 0, g_\phi = 0$ 
  if  $s_t == \text{<term>}$ 
     $t = 0, s_0 \sim p_0$ 
  end
   $t_{\text{start}} = t, T = \infty$ 
  for  $_ = 1, \dots, k$ 
     $a_t \sim \pi_\theta(\cdot | s_t)$ 
     $(r_t, s_{t+1}) \sim p(\cdot | s_t, a_t)$ 
     $t = t + 1$ 
    if  $s_t == \text{<term>}$ 
       $T = t$ 
      break
    end
  end
end
```

} Sample up to k steps ($k \approx 5$)

```
 $\hat{Q} = \begin{cases} V_\phi(s_t) & \text{if } s_t \neq \text{<term>} \\ 0 & \text{if } s_t == \text{<term>} \end{cases}$ 
for  $i = (t_{\text{start}} + k) \wedge T - 1, \dots, t_{\text{start}}$ 
   $\hat{Q} = r_i + \gamma \hat{Q}$ 
   $g_\theta += -(\nabla_\theta \log \pi_\theta(a_i | s_i))(\hat{Q} - V_\phi(s_i))$ 
   $g_\phi += \nabla_\phi \frac{1}{2}([\hat{Q}] - V_\phi(s_i))^2$ 
end
# accumulating grad using 1,2,...,k-step TD
update  $\theta$  and  $\phi$  using  $g_\theta$  and  $g_\phi$  with an optimizer
end
```

A2C

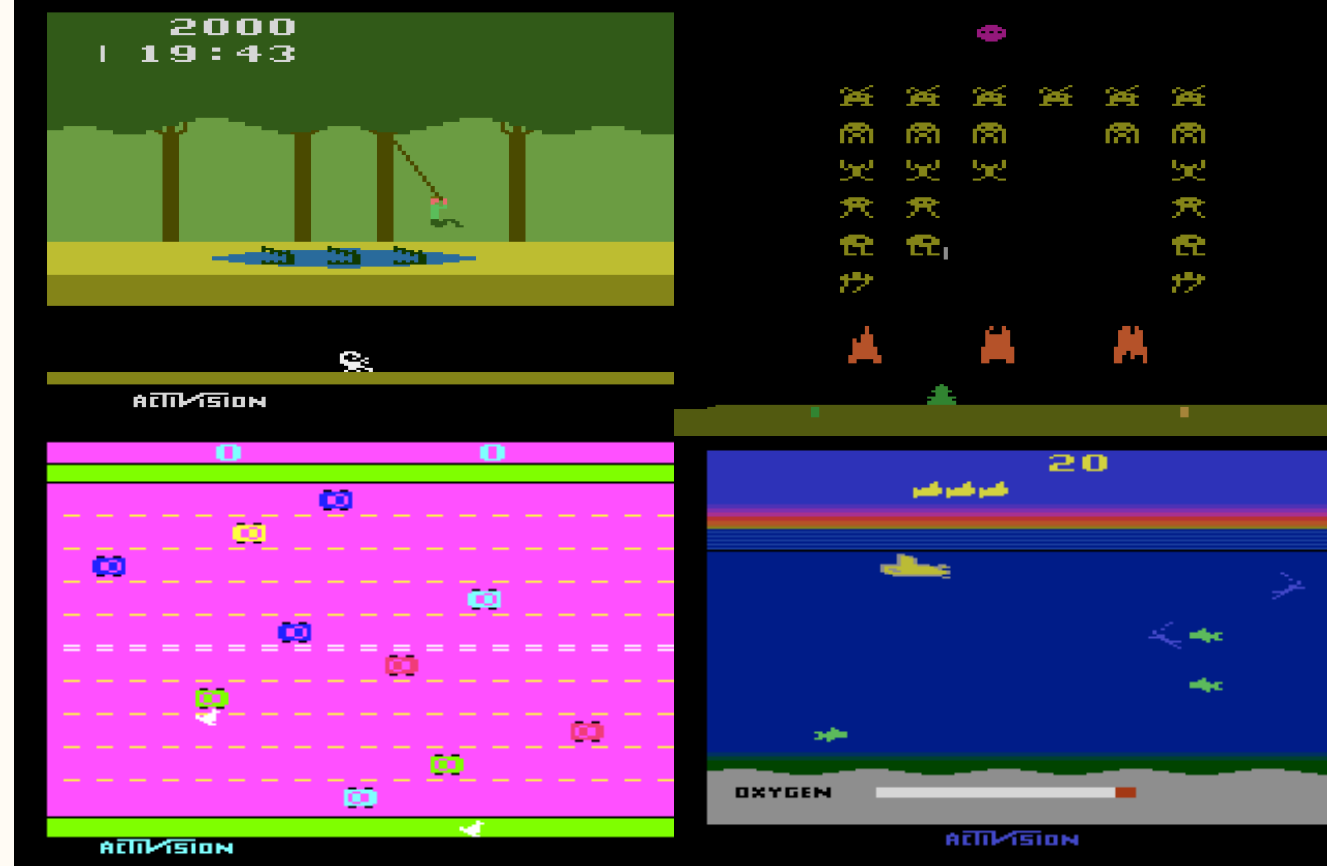
Mnih et al. published the *asynchronous advantage actor-critic* (A3C) method. A2C is a version of A3C without the multiple CPU cores performing gradient updates asynchronously.

Action space \mathcal{A} may be discrete or continuous. Further details through examples.

Atari 2600 game

Atari 2600 is a video game console released in 1977.

Bellemare et al. created an emulator environment to train and evaluate RL models on 55 different Atari games.



(Technically, using these Atari games for RL research is against copyright law as US Code Title 17, Chapter 3, Sec. 302, stipulates that the copyright of these games apply 70 years after the author's death.)

Atari 2600 action space

18 possible actions

↖ ↗ ↘

← · → × {no press, button press}

↙ ↓ ↘

(Not all games use all 18 actions.)

60 frames per second, so up to 60 actions per second.

Standard trick: Agent repeats the same action for 4 frames, i.e., 15 actions per second.



Atari 2600 preprocessing

Remove flickering (artifact caused by the limited number of sprites Atari 2600 can display at once), make black and white, and down-scale the image resolution.

Stack 4 most recent frames. This allows the information to contain current velocity.

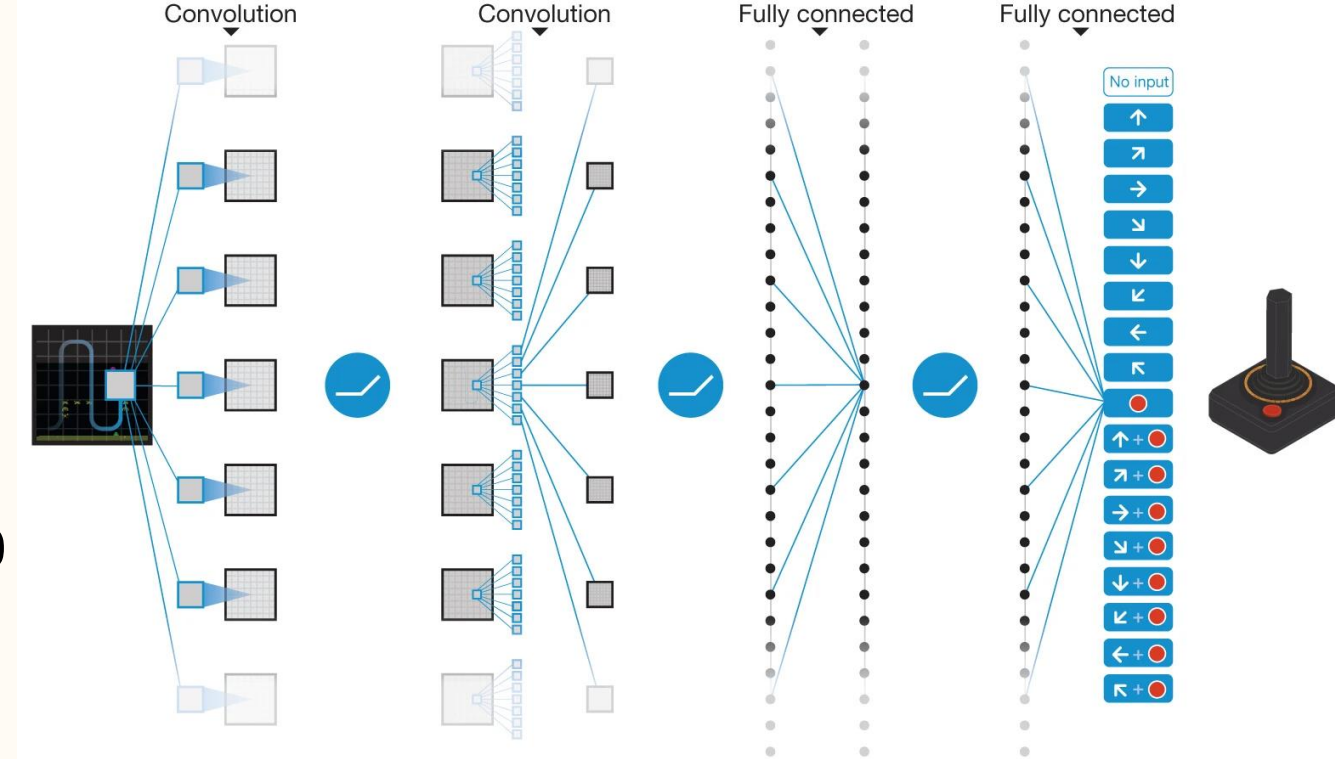
Note that Atari games are actually partially observable Markov decision processes (POMDP) and the game screen does not fully capture the game state. Moreover, the screen preprocessing means the agent (policy) receives only partial information.

These issues are not addressed (ignored) in the initial A3C or DQN papers.

Architecture

Architecture of Mnih et al.:

- Frames preprocessed into: 4x84x84
- 16 channel, 8x8 conv, stride 4, ReLU: 16x20x20
- 23 channel, 4x4 conv, stride 2, ReLU: 23x9x9
- FC 256, ReLU
- FC $|\mathcal{A}|$, softmax
 - \mathcal{A} is the set of valid actions. Between 4 and 18 for the games.
 - Softmax makes the output probabilities over actions.



In finite-action-space deep RL, deep NN usually has an output for each $a \in \mathcal{A}$ (as opposed to action $a \in \mathcal{A}$ being an input to the NN).

A2C for discrete action

Form a neural network $f_\theta : \mathcal{S} \rightarrow \Delta_{|\mathcal{A}|}$, where

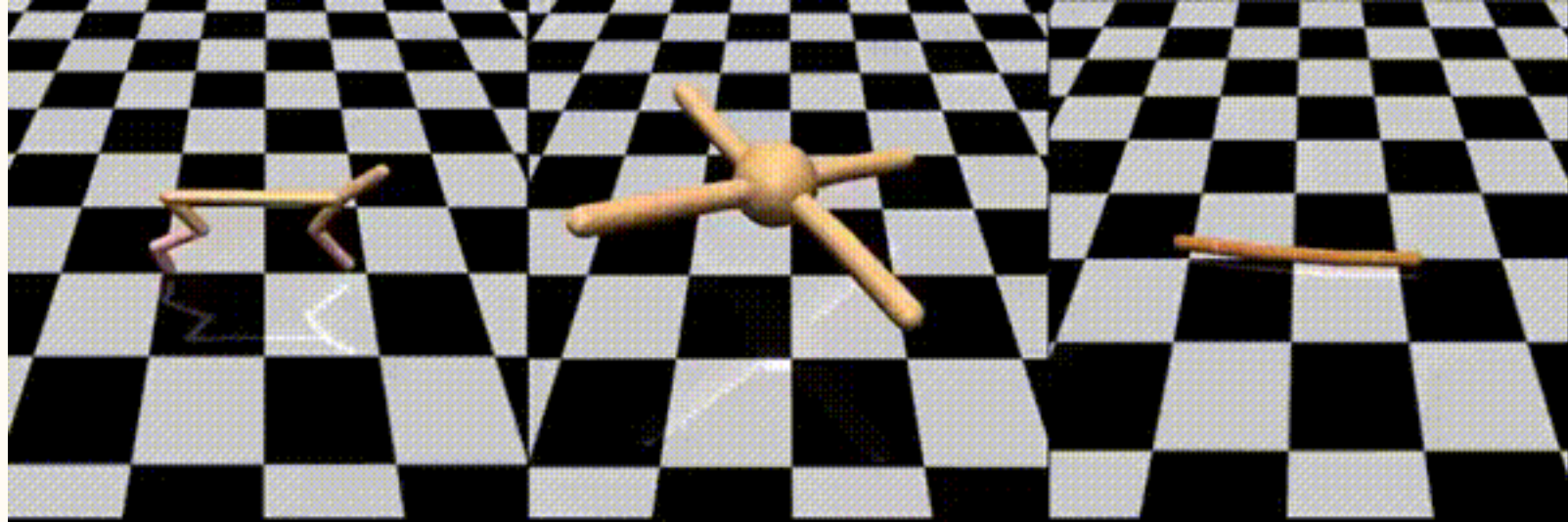
$$\Delta_d = \{p \in \mathbb{R}^d \mid p_1, \dots, p_d \geq 0, p_1 + \dots + p_d = 1\}$$

is the probability simplex of dimension d . For notational simplicity, assume actions $a_1, \dots, a_{|\mathcal{A}|} \in \mathcal{A}$ are all integers. There are 2 steps in A2C to clarify.

Step $a_t \sim \pi_\theta(\cdot \mid s_t)$: Evaluate $f_\theta(s_t)$ and sample a_j with probability $(f_\theta(s_t))_j$ for $j = 1, \dots, |\mathcal{A}|$.

Step $\nabla_\theta \log \pi_\theta(a_t \mid s_t)$: Backprop on $\log \left((f_\theta(s_t))_{a_t} \right)$.

MuJoCo



MuJoCo (**M**ulti-**J**oint dynamics with **C**ontact) is a general purpose physics engine that simulates articulated structures.

- State is the generalized coordinates and velocity of the joints.
- Action represents forces exerted on the joints and is continuous.
 - In robotics applications, it is common for an action to correspond to a controller, and the controller has maximum and minimum inputs bounds.

A2C for continuous action

A2C applies to the setup with continuous actions. There are 2 steps in A2C to clarify.

As a concrete instance, let $\mathcal{A} \subseteq [-1, +1]$ be a continuous action space. Form a neural network $f_\theta : \mathcal{S} \rightarrow \mathbb{R}^2$ and write

$$f_\theta(s) = (\mu_\theta(s), \tau_\theta(s))$$

Step $a_t \sim \pi_\theta(\cdot | s_t)$: Sample $a_t = \tanh(z_t)$ with $z_t \sim \mathcal{N}(\mu_\theta(s_t), \text{variance} = e^{2\tau_\theta(s_t)})$,

Step $\nabla_\theta \log \pi_\theta(a_i | s_i)$: Backpropagate on

$$\log \pi_\theta(a_t | s_t) = -\tau_\theta(s) - \frac{(\tanh^{-1}(a_t) - \mu_\theta(s_t))^2}{2e^{2\tau_\theta(s)}} + C$$

(Derivation in Hw. Follows from change of variable formula for probability density functions.)

Architecture

For a MuJoCo environment, $\mathcal{S} \subseteq \mathbb{R}^m$ with m ranging from 4 to 100 and $\mathcal{A} \subseteq \mathbb{R}^n$ with n being in a similar range.

The neural network

$$f_{\theta} : \mathcal{S} \rightarrow \mathbb{R}^{2n}$$

can be a simple ReLU MLP with ~ 4 layers.

Sample efficiency

In ML, sample efficiency refers to the method's ability to learn with fewer data points.

For RL in simulated environments (e.g. Atari 2600) sample efficiency is not a concern and only compute efficiency matters.

In many RL setups, however, samples (full trajectories or (s_t, a_t) pairs) are obtained by interacting with the environment, and this can be expensive. In such cases, we prefer methods with good sample efficiency.

E.g. having a physical robot take certain actions.

E.g. an LLM writing a completion and having a human provide feedback on whether the completion is good.

Can we learn more from an episode τ ?

$$\begin{aligned}\mathcal{J}(\theta) - \mathcal{J}(\theta_0) &= \mathcal{J}(\theta) - \mathbb{E}_{s_0 \sim p_0} [V^{\pi_{\theta_0}}(s_0)] \\ &= \mathcal{J}(\theta) - \mathbb{E}_{\tau \sim (p_0, \pi_{\theta}, p)} [V^{\pi_{\theta_0}}(s_0)]\end{aligned}$$

A2C uses an episode τ to perform T SGD updates. Can we learn more from τ ?
Can we be more sample efficient?

$$\begin{aligned}&= \mathcal{J}(\theta) - \mathbb{E}_{\tau \sim (p_0, \pi_{\theta}, p)} \left[\sum_{t=0}^{T-1} \gamma^t V^{\pi_{\theta_0}}(s_t) - \sum_{t=1}^{T-1} \gamma^t V^{\pi_{\theta_0}}(s_t) \right] \\ &= \mathbb{E}_{\tau \sim (p_0, \pi_{\theta}, p)} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right] + \mathbb{E}_{\tau \sim (p_0, \pi_{\theta}, p)} \left[\sum_{t=0}^{T-1} \gamma^t (\gamma V^{\pi_{\theta_0}}(s_{t+1}) - V^{\pi_{\theta_0}}(s_t)) \right] \\ &= \mathbb{E}_{\tau \sim (p_0, \pi_{\theta}, p)} \left[\sum_{t=0}^{T-1} \gamma^t (r_t + \gamma V^{\pi_{\theta_0}}(s_{t+1}) - V^{\pi_{\theta_0}}(s_t)) \right] \\ &= \mathbb{E}_{\tau \sim (p_0, \pi_{\theta}, p)} \left[\sum_{t=0}^{T-1} \gamma^t (Q^{\pi_{\theta_0}}(s_t, a_t) - V^{\pi_{\theta_0}}(s_t)) \right] \\ &= \mathbb{E}_{\tau \sim (p_0, \pi_{\theta}, p)} \left[\sum_{t=0}^{T-1} \gamma^t A^{\pi_{\theta_0}}(s_t, a_t) \right]\end{aligned}$$

We do so via the *surrogate objective*.

Let θ and θ_0 be the policy parameters.
Consider $\mathcal{J}(\theta)$ relative to $\mathcal{J}(\theta_0)$:

Surrogate objective derivation

Then,

$$\begin{aligned}
 \mathcal{J}(\theta) - \mathcal{J}(\theta_0) &= \mathbb{E}_{\tau \sim (p_0, \pi_{\theta}, p)} \left[\sum_{t=0}^{T-1} \gamma^t A^{\pi_{\theta_0}}(s_t, a_t) \right] \\
 &= \mathbb{E}_{\substack{s_0 \sim p_0 \\ a_0 \sim \pi_{\theta}(\cdot | s_0)}} \left[A^{\pi_{\theta_0}}(s_0, a_0) + \mathbb{E}^{\pi_{\theta}} \left[\sum_{t=1}^{T-1} \gamma^t A^{\pi_{\theta_0}}(s_t, a_t) \mid s_0, a_0 \right] \right] \\
 &= \mathbb{E}_{\substack{s_0 \sim p_0 \\ a_0 \sim \pi_{\theta}(\cdot | s_0) \\ a'_0 \sim \pi_{\theta_0}(\cdot | s_0)}} \left[\frac{\pi_{\theta}(a'_0 | s_0)}{\pi_{\theta_0}(a'_0 | s_0)} A^{\pi_{\theta_0}}(s_0, a'_0) + \mathbb{E}^{\pi_{\theta}} \left[\sum_{t=1}^{T-1} \gamma^t A^{\pi_{\theta_0}}(s_t, a_t) \mid s_0, a_0 \right] \right] \\
 &= \mathbb{E}_{\substack{\tau \sim (p_0, \pi_{\theta}, p) \\ a'_t \sim \pi_{\theta_0}(\cdot | s_t)}} \left[\sum_{t=0}^{T-1} \gamma^t \frac{\pi_{\theta}(a'_t | s_t)}{\pi_{\theta_0}(a'_t | s_t)} A^{\pi_{\theta_0}}(s_t, a'_t) \right]
 \end{aligned}$$

Surrogate objective derivation

So,

$$\mathcal{J}(\theta) = \mathbb{E}_{\substack{\tau \sim (p_0, \pi_\theta, p) \\ a'_t \sim \pi_{\theta_0}(\cdot | s_t)}} \left[\sum_{t=0}^{T-1} \gamma^t \frac{\pi_\theta(a'_t | s_t)}{\pi_{\theta_0}(a'_t | s_t)} A^{\pi_{\theta_0}}(s_t, a'_t) \right] + C$$

The expectation \mathbb{E} depends on θ and this is inconvenient. Let

$$\mathcal{K}(\theta; \theta_0) = \mathbb{E}_{\substack{\tau \sim (p_0, \pi_{\theta_0}, p) \\ a'_t \sim \pi_{\theta_0}(\cdot | s_t)}} \left[\sum_{t=0}^{T-1} \gamma^t \frac{\pi_\theta(a'_t | s_t)}{\pi_{\theta_0}(a'_t | s_t)} A^{\pi_{\theta_0}}(s_t, a'_t) \right] + C$$

Then, $\mathcal{J}(\theta) \approx \mathcal{K}(\theta; \theta_0)$ for $\theta \approx \theta_0$.[#] Here, C represents constants independent of θ .

In fact, $\nabla \mathcal{J}(\theta) \Big|_{\theta=\theta_0} = \nabla \mathcal{K}(\theta; \theta_0) \Big|_{\theta=\theta_0}$, although $\nabla^2 \mathcal{J}(\theta) \Big|_{\theta=\theta_0} \neq \nabla^2 \mathcal{K}(\theta; \theta_0) \Big|_{\theta=\theta_0}$.

[#]The \approx does not take C into account since constants are irrelevant when optimizing with respect to θ . The argument for $\mathcal{J} \approx \mathcal{K}$ is that \mathcal{J} and \mathcal{K} are equal up to 1st order in a neighborhood of θ_0 .

Surrogate objective derivation

For $\theta \approx \theta_0$, if we sample IID trajectories $\tau^{(1)}, \dots, \tau^{(N)} \sim (p_0, \pi_{\theta_0}, p)$, then

$$\begin{aligned} \mathcal{K}(\theta; \theta_0) &= \mathbb{E}_{\tau \sim (p_0, \pi_{\theta_0}, p)} \left[\sum_{t=0}^{T-1} \gamma^t \frac{\pi_{\theta}(s_t, a_t)}{\pi_{\theta_0}(s_t, a_t)} A^{\pi_{\theta_0}}(s_t, a_t) \right] \\ &\stackrel{(*)}{\approx} \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^{T^{(i)}-1} \gamma^t \frac{\pi_{\theta}(s_t^{(i)}, a_t^{(i)})}{\pi_{\theta_0}(s_t^{(i)}, a_t^{(i)})} \hat{A}_t^{(i)} \end{aligned}$$

where $\hat{A}_t \approx A^{\pi_{\theta_0}}(s_t, a_t)$ is an advantage estimate. The approximation $(*)$ is accurate when N is large and when $\pi_{\theta} \approx \pi_{\theta_0}$. (In general, importance sampling estimators become inaccurate when the sampling distribution π_{θ_0} is too far from the nominal distribution π_{θ} .)

We therefore maximize the *surrogate objective* subject to a certain trust-region constraint:

$$\begin{aligned} &\underset{\theta \in \mathbb{R}^p}{\text{maximize}} && \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^{T^{(i)}-1} \gamma^t \frac{\pi_{\theta}(s_t^{(i)}, a_t^{(i)})}{\pi_{\theta_0}(s_t^{(i)}, a_t^{(i)})} \hat{A}_t \\ &\text{subject to} && \theta \text{ and } \theta_0 \text{ close} \end{aligned}$$

(The trust-region constraint is needed for 2 reasons: $\mathcal{K}(\theta; \theta_0) \approx J(\theta)$ and $(*)$.)

TRPO

Trust-region policy optimization (TRPO) solves a sequence of trust-region optimization problems to improve the policy.

The trust region is defined by the KL-divergence of the policies.

Note that $\hat{A}_t = \hat{Q}_t - V_\phi(s_t)$ depends on θ_{curr} through \hat{Q}_t and ϕ through V_ϕ . (We discuss the choice of \hat{A}_t soon.)

The “solve” involves performing an approximate Newton method (which resembles a natural gradient method) with $H^{-1}g$ solved via a conjugate gradient (CG) solver. (We skip the details.)

while (not converged)

sample N trajectories $\tau^{(1)}, \dots, \tau^{(N)} \sim (p_0, \pi_{\theta_{\text{curr}}}, p)$

solve: (γ^t can be removed when using γ -trick)

$$\begin{aligned} & \underset{\theta_{\text{next}} \in \mathbb{R}^p}{\text{maximize}} && \sum_{i=1}^N \sum_{t=0}^{T^{(i)}-1} \gamma^t \frac{\pi_{\theta_{\text{next}}}(s_t^{(i)}, a_t^{(i)})}{\pi_{\theta_{\text{curr}}}(s_t^{(i)}, a_t^{(i)})} \hat{A}_t \\ & \text{subject to} && \max_{s \in \mathcal{S}} D_{\text{KL}}(\pi_{\theta_{\text{next}}}(\cdot | s) \| \pi_{\theta_{\text{curr}}}(\cdot | s)) \leq \delta \end{aligned}$$

$$\theta_{\text{curr}} = \theta_{\text{next}}$$

$$\hat{R}_t^{(i)} = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}^{(i)} \quad \text{for } i = 1, \dots, N, t = 1, \dots, T^{(i)} - 1$$

solve:

$$\underset{\phi \in \mathbb{R}^q}{\text{minimize}} \quad \frac{1}{N} \sum_{i=1}^N \frac{1}{T^{(i)}} \sum_{t=0}^{T^{(i)}-1} \frac{1}{2} (V_\phi(s_t^{(i)}) - \hat{R}_t^{(i)})^2$$

end

Proximal policy optimization (PPO)

Implementing TRPO hard work due to the trust-region formulation. Also, it is unclear whether the 2nd-order optimization of TRPO is efficient, since most deep learning formulations are optimized via 1st-order optimization algorithms (SGD).

Proximal policy optimization (PPO) returns to a 1st-order formulation while keeping the trust-region idea.

The PPO paper[#] presents “PPO-Penalty” and “PPO-Clip”. We will talk about the simpler PPO-Clip.

(PPO-Penalty uses a penalty, rather than a constrained, version of TRPO with the KL-divergence added to the objective as a regularizer.)

[#]J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, Proximal policy optimization algorithms, *arXiv*, 2017.

Clipped surrogate objective

The clipped surrogate objective in PPO is

$$\mathcal{C}_\varepsilon \left(\frac{\pi_{\theta}(a | s)}{\pi_{\theta_k}(a | s)}, \hat{A} \right) = \begin{cases} \min \left(\frac{\pi_{\theta}(a | s)}{\pi_{\theta_k}(a | s)}, 1 + \varepsilon \right) \hat{A} & \text{if } \hat{A} \geq 0 \\ \max \left(\frac{\pi_{\theta}(a | s)}{\pi_{\theta_k}(a | s)}, 1 - \varepsilon \right) \hat{A} & \text{if } \hat{A} < 0 \end{cases}$$

Interpretation: We increase/maximize $\frac{\pi_{\theta}(a | s)}{\pi_{\theta_k}(a | s)} \hat{A}$ only by a small factor.

This removes the incentive to move θ far away from θ_k .

The loss is equivalent to

$$\mathcal{C}_\varepsilon \left(\frac{\pi_{\theta}(a | s)}{\pi_{\theta_k}(a | s)}, \hat{A} \right) = \min \left(\frac{\pi_{\theta}(a | s)}{\pi_{\theta_k}(a | s)} \hat{A}, \text{clip}_{1-\varepsilon}^{1+\varepsilon} \left(\frac{\pi_{\theta}(a | s)}{\pi_{\theta_k}(a | s)} \right) \hat{A} \right)$$

while (not converged)

sample N trajectories $\tau^{(1)}, \dots, \tau^{(N)} \sim (p_0, \pi_{\theta_{\text{curr}}}, p)$

solve: $(\gamma^t \text{ can be removed when using } \gamma\text{-trick})$

$$\underset{\theta_{\text{next}} \in \mathbb{R}^p}{\text{maximize}} \quad \sum_{i=1}^N \sum_{t=0}^{T^{(i)}-1} \gamma^t \mathcal{C}_\varepsilon \left(\frac{\pi_{\theta_{\text{next}}}(a_t^{(i)} | s_t^{(i)})}{\pi_{\theta_{\text{curr}}}(a_t^{(i)} | s_t^{(i)})}, \hat{A}_t^{(i)} \right)$$

$$\theta_{\text{curr}} = \theta_{\text{next}}$$

$$\hat{R}_t^{(i)} = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}^{(i)} \quad \text{for } i = 1, \dots, N, t = 1, \dots, T^{(i)} - 1$$

solve:

$$\underset{\phi \in \mathbb{R}^q}{\text{minimize}} \quad \frac{1}{N} \sum_{i=1}^N \frac{1}{T^{(i)}} \sum_{t=0}^{T^{(i)}-1} \frac{1}{2} (V_\phi(s_t^{(i)}) - \hat{R}_t^{(i)})^2$$

end

PPO

Use the clipped loss

$$\mathcal{C}_\varepsilon(\ell, A) = \min(\ell A, \text{clip}_{1-\varepsilon}^{1+\varepsilon}(\ell) A)$$

The optimization subproblem is solved with first-order methods like SGD or Adam with early stopping.

The trust-region constraint is implicitly enforced by the clipping (no motivation to improve too much) and by performing few SGD iterations.

(We discuss the choice of \hat{A}_t soon.)

PPO: Discussion

In Schulman et al., $\varepsilon = 0.2$ is used.

In the maximization objective, $\sum_{i=1}^N \sum_{t=0}^{T^{(i)}-1}$ is viewed as loss formed with NT data points, where T is the average of $T^{(i)}$. The maximization performs ≈ 10 epochs over the NT data points.

Strictly speaking, TRPO and PPO are not policy *gradient* methods, but they can be viewed as variants/enhancements of deep policy gradient methods.

Bias-variance tradeoff of $\hat{A}_t^{\text{TD}(k)}$

Assume, for the sake of argument, we have access to $V^{\pi_{\text{curr}}}$. You will show in hw that

$$\hat{A}_t^{\text{TD}(1)} = r_t + \gamma V^{\pi_{\text{curr}}}(s_{t+1}) - V^{\pi_{\text{curr}}}(s_t)$$

$$\hat{A}_t^{\text{TD}(k)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{k-1} r_{t+k-1} + \gamma^k V^{\pi_{\text{curr}}}(s_{t+k}) - V^{\pi_{\text{curr}}}(s_t)$$

$$\hat{A}_t^{\text{TD}(\infty)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots - V^{\pi_{\text{curr}}}(s_t)$$

all have the same mean, but the variance reduces (Rao–Blackwell) with smaller k .

$$\mathbb{E}[\hat{A}_t^{\text{TD}(k)} \mid s_t, a_t] = A^{\pi_{\text{curr}}}(s_t, a_t) = Q^{\pi_{\text{curr}}}(s_t, a_t) - V^{\pi_{\text{curr}}}(s_t)$$

$$\text{Var}(\hat{A}_t^{\text{TD}(1)}) \leq \text{Var}(\hat{A}_t^{\text{TD}(k)}) \leq \text{Var}(\hat{A}_t^{\text{TD}(\infty)})$$

Bias-variance tradeoff of $\hat{A}_t^{\text{TD}}(k)$

In practice, we replace $V^{\pi_{\text{curr}}}$ with V_ϕ

$$\hat{A}_t^{\text{TD}(1)} = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$$

$$\hat{A}_t^{\text{TD}(k)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{k-1} r_{t+k-1} + \gamma^k V_\phi(s_{t+k}) - V_\phi(s_t)$$

$$\hat{A}_t^{\text{TD}(\infty)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots - V_\phi(s_t)$$

The estimators with $k < \infty$ are no longer unbiased.

$$\mathbb{E}[\hat{A}_t^{\text{TD}(1)} \mid s_t, a_t] = Q^{\pi_{\text{curr}}}(s_t, a_t) - V_\phi(s_t) + \underbrace{\gamma \mathbb{E}[V_\phi(s_{t+1}) - V^{\pi_{\text{curr}}}(s_{t+1}) \mid s_t, a_t]}_{\text{bias}}$$

$$\mathbb{E}[\hat{A}_t^{\text{TD}(k)} \mid s_t, a_t] = Q^{\pi_{\text{curr}}}(s_t, a_t) - V_\phi(s_t) + \underbrace{\gamma^k \mathbb{E}[V_\phi(s_{t+k}) - V^{\pi_{\text{curr}}}(s_{t+k}) \mid s_t, a_t]}_{\text{smaller bias}}$$

$$\mathbb{E}[\hat{A}_t^{\text{TD}(\infty)} \mid s_t, a_t] = Q^{\pi_{\text{curr}}}(s_t, a_t) - V_\phi(s_t) \underbrace{\hspace{1cm}}_{\text{no bias}}$$

Bias-variance tradeoff of $\hat{A}_t^{\text{TD}(k)}$

Although there is no precise analysis, we still expect the variance to reduce with smaller k .

$$\text{Var}(\hat{A}_t^{\text{TD}(1)} \mid s_t, a_t) \leq \text{Var}(\hat{A}_t^{\text{TD}(k)} \mid s_t, a_t) \leq \text{Var}(\hat{A}_t^{\text{TD}(\infty)} \mid s_t, a_t)$$

We now have a bias-variance tradeoff with k :

- Small k : Large bias but small variance.
- Large k : Small bias but large variance.

Bias-variance tradeoff of $\hat{A}_t^{\text{TD}(k)}$

What about the dependency on γ ? Consider using an artificial discount factor γ for an undiscounted MDP:

$$\hat{A}_t^{\text{TD}(k)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{k-1} r_{t+k-1} + \gamma^k V_\phi(s_{t+k}) - V_\phi(s_t)$$

In terms of expectations, $\gamma = 1$ is the correct choice (if $k = \infty$ and $\gamma = 1$, then unbiased) and using $\gamma < 1$ introduces bias. With $\gamma < 1$, however, the later rewards contribute less and they contribute less towards the variance.

We now have a bias-variance tradeoff with γ :

- Small γ : Large bias but small variance.
- Large γ : Small bias but large variance.

Generalized Advantage Estimation (GAE)

One challenge with tuning k is that it cannot be continuously tuned. (What if you want to use k larger than 4 and smaller than 5?)

Generalized Advantage Estimation (GAE) uses an exponentially weighted average of all $\hat{A}_t^{\text{TD}(k)}$ estimators based on an approach analogous to a classical technique called TD(λ).

Generalized Advantage Estimation (GAE)

Given a $V(s)$ (meant to approximate $V^\pi(s)$), let

$$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$$

Then, by a telescoping-sum argument, we have

$$\hat{A}_t^{\text{TD}(k)} = \sum_{l=0}^{k-1} (\gamma)^l \delta_{t+l}^V$$

Next, define the GAE estimator with $\lambda \in [0,1]$ as

$$\begin{aligned} \hat{A}_t^{\text{GAE}(\gamma, \lambda)} &= (1 - \lambda) \left(\hat{A}_t^{\text{TD}(1)} + \lambda \hat{A}_t^{\text{TD}(2)} + \lambda^2 \hat{A}_t^{\text{TD}(3)} + \dots \right) \\ &\stackrel{(*)}{=} \sum_{l=0}^{\infty} (\gamma \lambda)^l \delta_{t+l}^V \end{aligned}$$

where $(*)$ is due to a geometric-sum argument. (Exercise)

Generalized Advantage Estimation (GAE)

$$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t) \quad \hat{A}_t^{\text{TD}(k)} = \sum_{l=0}^{k-1} (\gamma)^l \delta_{t+l}^V$$

$$\hat{A}_t^{\text{GAE}(\gamma, \lambda)} = (1 - \lambda) \left(\hat{A}_t^{\text{TD}(1)} + \lambda \hat{A}_t^{\text{TD}(2)} + \lambda^2 \hat{A}_t^{\text{TD}(3)} + \dots \right) = \sum_{l=0}^{\infty} (\gamma \lambda)^l \delta_{t+l}^V$$

Note, if $\lambda = 0$, we recover the TD(1) estimator. If $\lambda = 1$, we recover the TD(∞) estimator

$$\hat{A}_t^{\text{GAE}(\gamma, 0)} = \delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t) = \hat{A}_t^{\text{TD}(1)}$$

$$\hat{A}_t^{\text{GAE}(\gamma, 1)} = \sum_{l=0}^{\infty} \gamma^l \delta_{t+l}^V = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots - V(s_t) = \hat{A}_t^{\text{TD}(\infty)}$$

If $V = V^\pi$, then GAE is an unbiased estimator. I will leave it as an exercise to show that

$$\mathbb{E}[\delta_t^V \mid s_t, a_t] = A^\pi(s_t, a_t)$$

$$\mathbb{E}[\delta_{t+l}^V \mid s_t, a_t] = 0 \quad \text{for } l \geq 1$$

So

$$\mathbb{E}[\hat{A}_t^{\text{GAE}(\gamma, \lambda)} \mid s_t, a_t] = \underbrace{\mathbb{E}[\delta_t^V \mid s_t, a_t]}_{=A^\pi(s_t, a_t)} + \sum_{l=1}^{\infty} (\gamma \lambda)^l \underbrace{\mathbb{E}[\delta_{t+l}^V \mid s_t, a_t]}_{=0} = A^\pi(s_t, a_t)$$

Bias-variance tradeoff of $\hat{A}_t^{\text{GAE}(\gamma, \lambda)}$

$$\hat{A}_t^{\text{GAE}(\gamma, \lambda)} = (1 - \lambda) \left(\hat{A}_t^{\text{TD}(1)} + \lambda \hat{A}_t^{\text{TD}(2)} + \lambda^2 \hat{A}_t^{\text{TD}(3)} + \dots \right) = \sum_{l=0}^{\infty} (\gamma \lambda)^l \delta_{t+l}^V$$

The λ of GAE is a continuous parameter serving a similar role to the k of TD.

When $V_\phi = V^\pi$, any value of λ incurs no bias. (Unlike γ , which always causes a bias when $\gamma < 1$.) But when $V_\phi \neq V^\pi$, then large λ allows the telescoping structure to reduce bias (when $\lambda = \gamma = 1$ there is no bias) while for small λ the bias is larger.

We now have a bias-variance tradeoff with γ and λ :

- Small γ : Large bias but small variance.
- Large γ : Small bias but large variance.
- Small λ : Large bias but small variance.
- Large λ : Small bias but large variance.

TRPO and PPO uses GAE for the advantage estimator \hat{A}_t .

A common choice of values:
 $\gamma = 0.995$ and $\lambda = 0.96$.

Policy advantage

Define the policy advantage of π over π_k as

$$\mathcal{I}(\pi; \pi_k) = \mathbb{E}_{\substack{\tau \sim (p_0, \pi_k, p) \\ a'_t \sim \pi(\cdot | s_t)}} \left[\sum_{t=0}^{T-1} \gamma^t A^{\pi_k}(s_t, a'_t) \right]$$

Note that

$$\mathcal{I}(\pi_k; \pi_k) = \mathbb{E}_{\tau \sim (p_0, \pi_k, p)} \left[\sum_{t=0}^{T-1} \gamma^t A^{\pi_k}(s_t, a_t) \right] = 0$$

Interpretation: Follow a trajectory τ generated π_k , and we ask what actions π would have chosen. $\mathcal{I}(\pi; \pi_k)$ quantifies how better, as measured by A^{π_k} , these actions are compared to the actions chosen by π_k .

Theorem) A policy π_k is optimal if and only if $\max_{\pi} \mathcal{I}(\pi; \pi_k) = 0$.

PI as policy advantage maximization

We can equivalently express PI as policy advantage maximization.

$$\mathcal{I}(\pi; \pi_k) = \mathbb{E}_{\substack{\tau \sim (p_0, \pi_k, p) \\ a'_t \sim \pi(\cdot | s_t)}} \left[\sum_{t=0}^{T-1} \gamma^t A^{\pi_k}(s_t, a'_t) \right]$$

Policy iteration (PI): $\pi_{k+1} = \operatorname{argmax}_{\pi} \mathcal{I}(\pi; \pi_k), \quad \text{for } k = 0, 1, \dots$

(Strictly speaking, the equivalence requires that $\tau \sim (p_0, \pi_k, p)$ has positive probability to visit every state $s \in \mathcal{S}$.)

Proof) $\mathcal{I}(\pi; \pi_k)$ is maximized when $\pi(s) = \operatorname{argmax}_{a \in \mathcal{A}} A^{\pi_k}(s, a)$. This is exactly PI. ■

Implementing PI with IS estimates

In practice, the expectation of $\mathcal{I}(\pi; \pi_k)$ has no closed form formula, so we approximate it with importance sampling (IS) estimates. We sample $\tau^{(1)}, \dots, \tau^{(N)} \sim (p_0, \pi_k, p)$ IID trajectories and form the estimator $\hat{\mathcal{I}}(\pi; \pi_k)$:

$$\mathcal{I}(\pi; \pi_k) = \mathbb{E}_{\substack{\tau \sim (p_0, \pi_k, p) \\ a'_t \sim \pi(\cdot | s_t)}} \left[\sum_{t=0}^{T-1} \gamma^t A^{\pi_k}(s_t, a'_t) \right] = \mathbb{E}_{\substack{\tau \sim (p_0, \pi_k, p) \\ a_t \sim \pi_k(\cdot | s_t)}} \left[\sum_{t=0}^{T-1} \gamma^t \frac{\pi(a_t | s_t)}{\pi_k(a_t | s_t)} A^{\pi_k}(s_t, a_t) \right]$$

$$\hat{\mathcal{I}}(\pi; \pi_k) = \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^{T^{(i)}-1} \gamma^t \frac{\pi(s_t^{(i)}, a_t^{(i)})}{\pi_k(s_t^{(i)}, a_t^{(i)})} \hat{A}_t^{(i)}$$

using advantage estimators $\hat{A}_t^{(i)}$ satisfying

$$\mathbb{E}^{\pi_k}[\hat{A}_t^{(i)} | s_t^{(i)}, a_t^{(i)}] = A^{\pi_k}(s_t^{(i)}, a_t^{(i)})$$

When N is large, we expect $\mathcal{I}(\pi; \pi_k) \approx \hat{\mathcal{I}}(\pi; \pi_k)$?

TRPO and PPO as approximate PI

We can interpret TRPO and PPO as an approximate policy iteration (PI). Consider the following approximate PI:

$$\hat{\mathcal{I}}(\pi_{\theta}; \pi_{\theta_k}) = \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^{T^{(i)}-1} \gamma^t \frac{\pi_{\theta}(s_t^{(i)}, a_t^{(i)})}{\pi_{\theta_k}(s_t^{(i)}, a_t^{(i)})} \hat{A}_t^{(i)}$$
$$\theta_{k+1} \approx \underset{\theta}{\operatorname{argmax}} \hat{\mathcal{I}}(\pi_{\theta}; \pi_{\theta_k}), \quad \text{for } k = 0, 1, \dots$$

where the argmax is computed approximately.

This fails because $\mathcal{I}(\pi_{\theta_{k+1}}; \pi_{\theta_k}) \approx \hat{\mathcal{I}}(\pi_{\theta_{k+1}}; \pi_{\theta_k})$ requires $\pi_{\theta_{k+1}} \approx \pi_{\theta_k}$. In general, importance sampling estimators become inaccurate when the sampling distribution π_{θ_k} is not too far from the nominal distribution $\pi_{\theta_{k+1}}$. Therefore, a trust-region or proximal mechanism is needed, and this leads to TRPO as PPO.

(The first derivation covered earlier is the derivation from the TRPO and PPO papers. This is an alternate interpretation of the methods.)

Avoiding the baseline function?

In TRPO and PPO, we must learn two neural networks: π_θ and V_ϕ .

Learning the baseline function (also called the critic function) V_ϕ is an additional layer of complexity, and it would be better if we can avoid it.

But let us recall why V_ϕ is needed: We want to assign appropriate sign of the gradient signals based on whether the action is better than what the current policy prescribes.

PPO without baseline

Consider the undiscounted MDP with no γ -trick. Assume a non-zero reward is collected only at the end of the episode. In this case, $\hat{Q}_t^{\text{TD}(\infty)} = r$ for $t = 0, 1, \dots, T - 1$, where $r = r_{T-1}$ is the reward obtained at the end, as the episode terminates.

Then, PPO without a baseline function would be:

while (not converged)

sample N trajectories $\tau^{(1)}, \dots, \tau^{(N)} \sim (p_0, \pi_{\theta_{\text{curr}}}, p)$

solve:

$$\mathcal{C}_\varepsilon(\ell, A) = \min(\ell A, \text{clip}_{1-\varepsilon}^{1+\varepsilon}(\ell) A)$$

$$\underset{\theta_{\text{next}} \in \mathbb{R}^p}{\text{maximize}} \quad \sum_{i=1}^N \sum_{t=0}^{T^{(i)}-1} \mathcal{C}_\varepsilon \left(\frac{\pi_{\theta_{\text{next}}}(a_t^{(i)} | s_t^{(i)})}{\pi_{\theta_{\text{curr}}}(a_t^{(i)} | s_t^{(i)})}, r^{(i)} \right)$$

$$\theta_{\text{curr}} = \theta_{\text{next}}$$

end

Without the baseline, this will not work!
Variance of the objective will be too large.

Group Relative Policy Optimization

Again consider the undiscounted MDP with no γ -trick. Assume a non-zero reward is collected only at the end of the episode.

Group Relative Policy Optimization (GRPO) replaces the baseline function with a BatchNorm-style normalization of the rewards.

while (not converged)

sample N trajectories $\tau^{(1)}, \dots, \tau^{(N)} \sim (p_0, \pi_{\theta_{\text{curr}}}, p)$

$\mathbf{r} = (r^{(1)}, \dots, r^{(N)})$

solve:

$$\mathcal{C}_\varepsilon(\ell, A) = \min(\ell A, \text{clip}_{1-\varepsilon}^{1+\varepsilon}(\ell) A)$$

$$\underset{\theta_{\text{next}} \in \mathbb{R}^p}{\text{maximize}} \quad \sum_{i=1}^N \sum_{t=0}^{T^{(i)}-1} \mathcal{C}_\varepsilon \left(\frac{\pi_{\theta_{\text{next}}}(a_t^{(i)} | s_t^{(i)})}{\pi_{\theta_{\text{curr}}}(a_t^{(i)} | s_t^{(i)})}, \frac{r^{(i)} - \text{mean}(\mathbf{r})}{\text{std}(\mathbf{r}) + \varepsilon} \right)$$

$$\theta_{\text{curr}} = \theta_{\text{next}}$$

end

Advantage estimate interpretation

The advantage $A^\pi(s_t, a_t)$ measures how good is action a_t compared to the average action selected by π .

$$\hat{A}_{(i)}^{\text{GRPO}} = \frac{r^{(i)} - \text{mean}(\mathbf{r})}{\text{std}(\mathbf{r}) + \varepsilon}$$

The GRPO advantage estimate is not an unbiased estimate of $A^\pi(s_t, a_t)$.

Qualitatively, however, $\hat{A}_{(i)}^{\text{GRPO}}$ measures similar information: The trajectory with action $a_t^{(i)}$ yielded reward $r^{(i)}$, and how good is this compared to the other actions selected by π , which yielded rewards \mathbf{r} ?

DeepSeek's GitHub uses $\varepsilon = 10^{-4}$. If $\text{std}(\mathbf{r}) = 0$, then $\hat{A}_{(i)}^{\text{GRPO}} = 0$ and no update happens.

GRPO: Caveats

GRPO should not yet be considered a general deep RL method. Its effectiveness has not been tested outside of the LLM applications

GRPO is not the first modern deep RL method that forgoes a baseline or critic function.[#]

It is not obvious that GRPO is the best (or a good enough) algorithm to do RL without a baseline or critic model, even restricted to LLMs. GRPO got a lot of attention due to its use in training DeepSeek-R1, but better approaches may replace GRPO in future work. Improving upon GRPO will likely be an active area of research.

[#]Z. Li, T. Xu, Y. Zhang, Z. Lin, Y. Yu, R. Sun, Z.-Q. Luo, ReMax: A simple, effective, and efficient reinforcement learning method for aligning large language models, *ICML*, 2024.

Why not other deep RL algorithms?

Most RL-LLM methods use PPO or variants of PPO like GRPO. Why not other choices?

Key properties of the RL-LLM setup:

- LLMs have finite action space (# of possible tokens). Action space is not continuous.
- A strong pre-trained large language model π_θ , which serves as a suboptimal but reasonably good initial policy for the RL, is absolutely crucial. Tabula rasa RL methods (randomly initialized π_θ) do not work.
- We expect an RL method for RL-LLM to work only if it can effectively utilize the pre-trained LLM π_θ .

Why not other deep RL algorithms?

DQN and Rainbow DQN are deep RL methods parameterizing Q_ϕ as a neural network and learning $Q_\phi \approx Q^*$ through the Bellman optimality equation.

The policy π_ϕ is implicitly derived from the greedy rule $\pi_\phi(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q_\phi(s, a)$. There is no explicitly parameterized policy. The strength of π_ϕ hinges on the accuracy of $Q_\phi \approx Q^*$.

There does not seem to be a good way for DQN to utilize a pre-trained policy.

Why not other deep RL algorithms?

DDPG, TD3, and SAC are methods with dual interpretations as deterministic policy gradient methods (qualitatively quite different from the stochastic policy gradient methods) and Q-learning methods. These methods are designed for continuous action spaces, although the variant ‘SAC discrete’ applies to discrete action spaces.

These methods train both π_θ and Q_ϕ . Different from PPO, π_θ is not trained directly to maximize reward. Rather π_θ is trained to satisfy the relation $\pi_\theta(s) \approx \operatorname{argmax}_{a \in \mathcal{A}} Q_\phi(s, a)$.

In PPO, rewards directly influence the updates of π_θ . In DDPG, TD3, and SAC, rewards influence Q_ϕ which in turns influences π_θ . So, the updates on π_θ depend on the rewards only through Q_ϕ .

Without a good initialization for $Q_\phi \approx Q^*$, a pre-trained π_θ alone seems difficult to utilize.

S. Fujimoto, H. van Hoof, and D. Meger, Addressing function approximation error in actor-critic methods, *ICML*, 2018.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, Continuous control with deep reinforcement learning, *ICLR*, 2016.

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, *ICML (and arXiv v2)*, 2018.

P. Christodoulou, Soft actor-critic for discrete action settings, *arXiv*, 2019.

AlphaGo, Test-Time Compute, and Expert Iteration

2-player zero-sum games

In a 2-player zero-sum game, two players compete against each other and one player's reward is precisely the loss of the other player.

In a simultaneous move game, the 2 players make their moves simultaneously.
E.g., rock paper scissors.

In a sequential (turn-based) move game, the 2 players make their moves in turns.
E.g., chess and go. However, we will think of sequential move games as simultaneous move games where the two players each offer policies $\pi^{(1)}$ and $\pi^{(2)}$ at the start of the game. (In real life game competitions, a human offers their brain at the start of the game.) These policies $\pi^{(1)}$ and $\pi^{(2)}$ can strategize and adapt to the opponents' moves. They can also make randomized decisions.

Minimax optimization

In a *minimax optimization problem*, we minimize with respect to one variable and maximize with respect to another:

$$\underset{\theta^{(2)}}{\text{minimize}} \quad \underset{\theta^{(1)}}{\text{maximize}} \quad R(\theta^{(1)}, \theta^{(2)})$$

We say $(\theta_{\star}^{(1)}, \theta_{\star}^{(2)})$ is a *solution*^{*} to the minimax problem if it is a Nash equilibrium:

$$R(\theta^{(1)}, \theta_{\star}^{(2)}) \leq R(\theta_{\star}^{(1)}, \theta_{\star}^{(2)}) \leq R(\theta_{\star}^{(1)}, \theta^{(2)}), \quad \forall \theta^{(1)}, \theta^{(2)}.$$

In other words, unilaterally deviating from $\theta_{\star}^{(1)}$ decreases the value of R while unilaterally deviating from $\theta_{\star}^{(2)}$ increases the value of R .

Standard RL is posed as a maximization problem. However, adversarial training and two-player zero-sum games are posed as minimax optimization problems.

^{*}There are other broader definitions of a “solution” in minimax optimization problems. Our definition is, in a sense, the strictest definition.

Example: Rock paper scissors

Consider the game of rock paper scissors with randomized strategies $p_{\theta^{(1)}}$ and $p_{\theta^{(2)}}$ and expected payoff $R(\theta^{(1)}, \theta^{(2)})$:

$$\begin{aligned} & \underset{\theta^{(2)} \in \mathbb{R}^3}{\text{minimize}} \quad \underset{\theta^{(1)} \in \mathbb{R}^3}{\text{maximize}} \quad \underbrace{p_{\theta^{(1)}}^\top \begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix} p_{\theta^{(2)}}}_{=R(\theta^{(1)}, \theta^{(2)})} \\ p_{\theta^{(1)}} &= \frac{1}{e^{\theta_1^{(1)}} + e^{\theta_2^{(1)}} + e^{\theta_3^{(1)}}} \begin{bmatrix} e^{\theta_1^{(1)}} \\ e^{\theta_2^{(1)}} \\ e^{\theta_3^{(1)}} \end{bmatrix} = \mu(\theta^{(1)}), \quad p_{\theta^{(2)}} = \frac{1}{e^{\theta_1^{(2)}} + e^{\theta_2^{(2)}} + e^{\theta_3^{(2)}}} \begin{bmatrix} e^{\theta_1^{(2)}} \\ e^{\theta_2^{(2)}} \\ e^{\theta_3^{(2)}} \end{bmatrix} = \mu(\theta^{(2)}) \end{aligned}$$

where μ is the softmax function. Of course, the Nash equilibrium occurs at

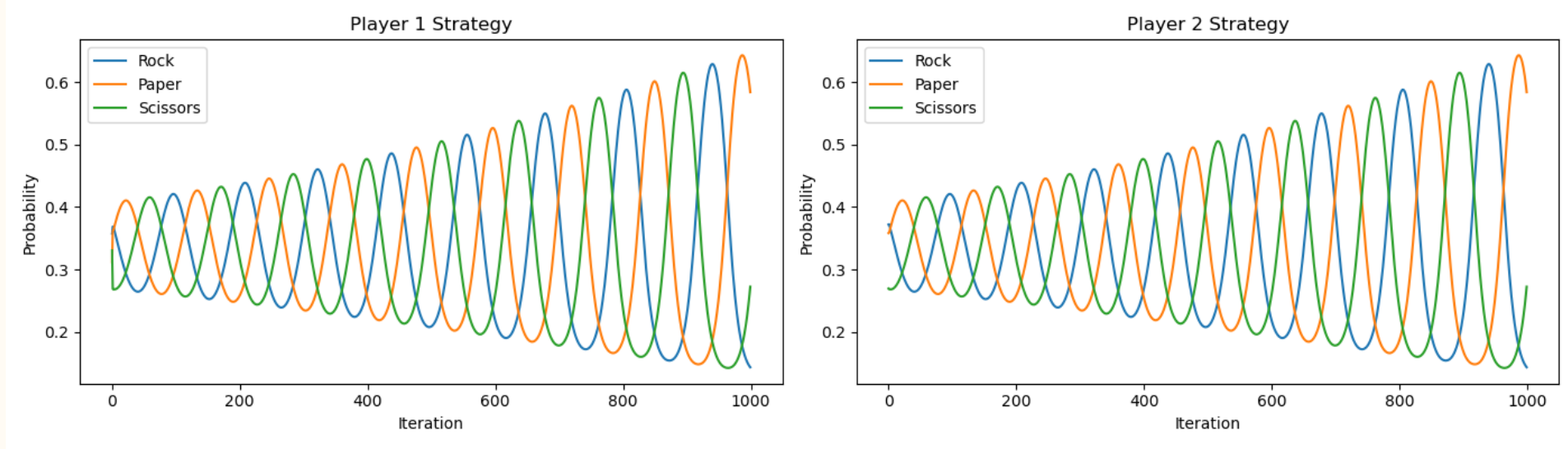
$$\theta^{(1)} = c_1 \mathbf{1}, \quad p_{\theta^{(1)}} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}, \quad \theta^{(2)} = c_2 \mathbf{1}, \quad p_{\theta^{(2)}} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

Minimax optimization with gradients

In deep learning, we solve minimax optimization algorithms with first-order methods using stochastic gradients.

However, convergence of minimax optimization should not be taken for granted, and much more delicate care is needed than min optimization. The training of GANs is famously tricky. We will also see that the minimax training of the rock paper scissors examples is a highly unstable process.

RL training is already more unstable than supervised learning. Multi-agent RL (2-agent in our case) is even more delicate than non-RL minimax training.



Simultaneous gradient ascent-descent

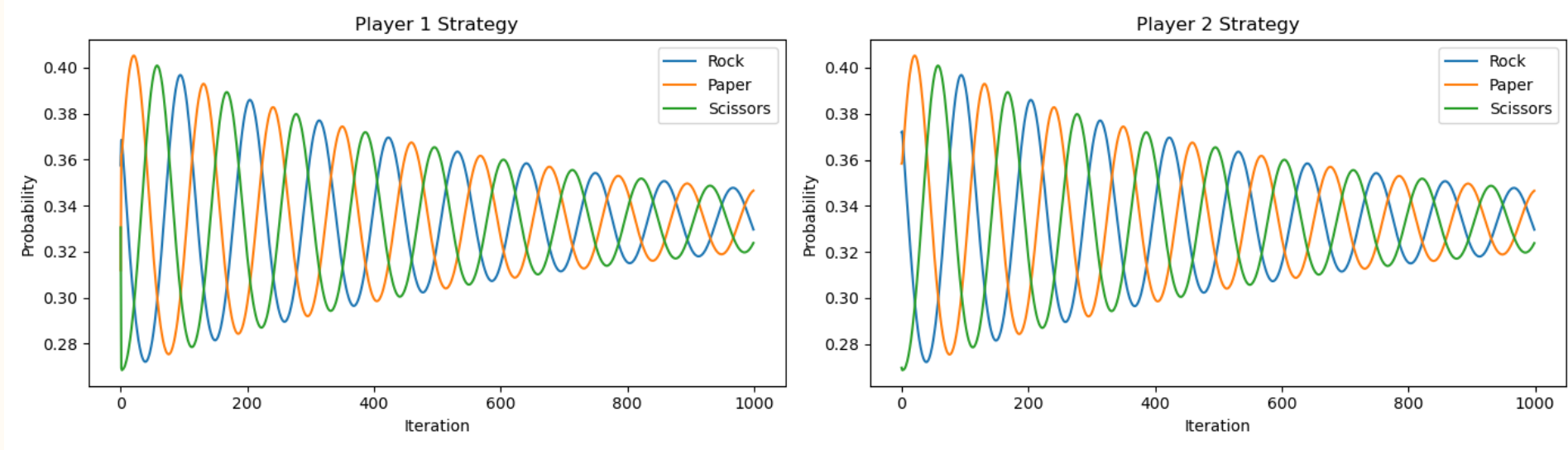
Using the notation $\nabla_i = \nabla_{\theta^{(i)}}:$

$$\theta_{\text{next}}^{(1)} = \theta^{(1)} + \alpha \nabla_1 R(\theta^{(1)}, \theta^{(2)})$$

$$\theta_{\text{next}}^{(2)} = \theta^{(2)} - \alpha \nabla_2 R(\theta^{(1)}, \theta^{(2)})$$

Simultaneous gradient ascent-descent (SimGAD) is one of the simplest minimax optimization algorithms, but it fails to converge on rock paper scissors. In fact, SimGAD is expected to diverge on any zero-sum game.

Cycling dynamics where players counter the counter: Player 1 plays rock \rightarrow Player 2 plays paper \rightarrow Player 1 plays scissors \rightarrow Player 2 plays rock \rightarrow Player 1 plays paper $\rightarrow \dots$



Extragradient method

$$\theta_{\text{temp}}^{(1)} \leftarrow \theta^{(1)} + \alpha \nabla_1 R(\theta^{(1)}, \theta^{(2)})$$

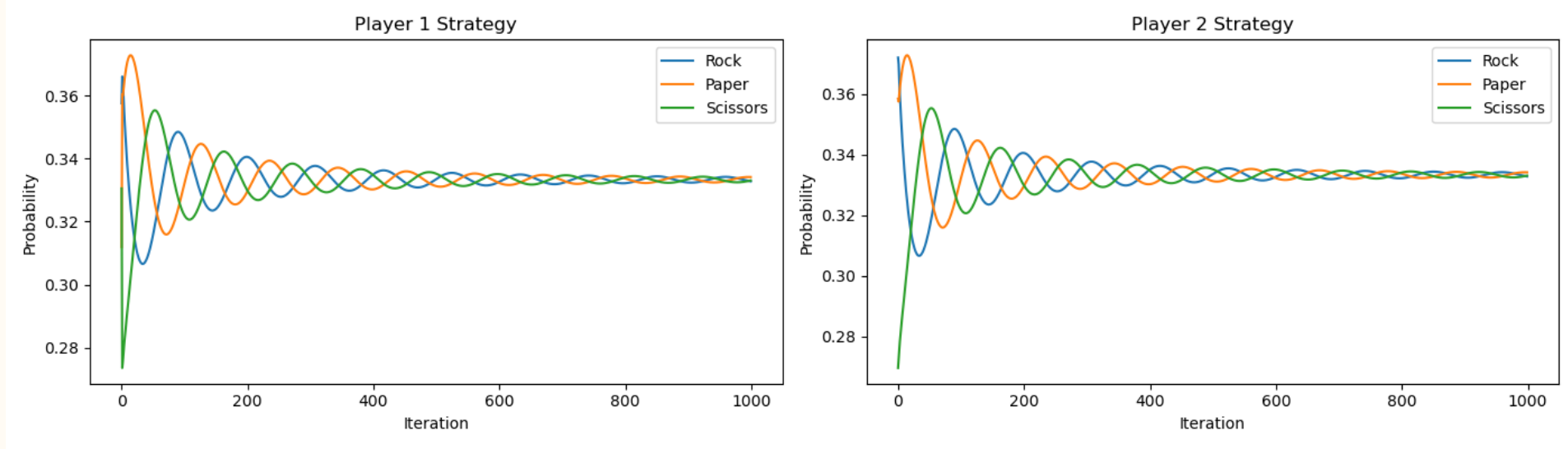
$$\theta_{\text{temp}}^{(2)} \leftarrow \theta^{(2)} - \alpha \nabla_2 R(\theta^{(1)}, \theta^{(2)})$$

$$\theta_{\text{next}}^{(1)} \leftarrow \theta^{(1)} + \alpha \nabla_1 R(\theta_{\text{temp}}^{(1)}, \theta_{\text{temp}}^{(2)})$$

$$\theta_{\text{next}}^{(2)} \leftarrow \theta^{(2)} - \alpha \nabla_2 R(\theta_{\text{temp}}^{(1)}, \theta_{\text{temp}}^{(2)})$$

For the extragradient (EG) method, imagine you and the opponent make regular 1-step updates, evaluate the gradient from there, and use that gradient to commit to the update. Closely related to learning with opponent-level awareness (LOLA).

EG does converge for 2-player zero-sum games.



Anchoring and weight decay

$$\theta_{\text{next}}^{(1)} = (1 - \lambda)\theta^{(1)} + \alpha \nabla_1 R(\theta^{(1)}, \theta^{(2)})$$

$$\theta_{\text{next}}^{(2)} = (1 - \lambda)\theta^{(2)} - \alpha \nabla_2 R(\theta^{(1)}, \theta^{(2)})$$

Anchored simultaneous gradient ascent-descent adds the “anchor” mechanism that can also be understood as weight decay. (Anchor doesn’t have to be at 0, but weight decay shrinks toward 0.) This does converge on 2-player zero-sum games.

Antisymmetric payoff games

We say a 2-player zero-sum game has *antisymmetric payoff* if

$$R(\theta^{(2)}, \theta^{(1)}) = -R(\theta^{(1)}, \theta^{(2)}), \quad \forall \theta^{(1)}, \theta^{(2)}$$

In this case, the Nash equilibrium at (θ^*, θ^*) for some θ^* and

$$R(\theta, \theta^*) \leq \underbrace{R(\theta^*, \theta^*)}_{=0} \leq R(\theta^*, \theta), \quad \forall \theta$$

Also,

$$\nabla_1 R(\theta, \theta) = -\nabla_2 R(\theta, \theta)$$

Therefore, SimGAD simplifies to

$$\begin{aligned} \theta_{\text{next}}^{(1)} &= \theta^{(1)} + \alpha \nabla_1 R(\theta^{(1)}, \theta^{(2)}) \\ \theta_{\text{next}}^{(2)} &= \theta^{(2)} - \alpha \nabla_2 R(\theta^{(1)}, \theta^{(2)}) \end{aligned} \quad \Rightarrow \quad \theta_{\text{next}} = \theta + \alpha \nabla_1 R(\theta, \theta)$$

Antisymmetric payoff games

SimGAD with weight decay simplifies to

$$\begin{aligned}\theta_{\text{next}}^{(1)} &= (1 - \lambda)\theta^{(1)} + \alpha \nabla_1 R(\theta^{(1)}, \theta^{(2)}) \\ \theta_{\text{next}}^{(2)} &= (1 - \lambda)\theta^{(2)} - \alpha \nabla_2 R(\theta^{(1)}, \theta^{(2)})\end{aligned} \quad \Rightarrow \quad \theta_{\text{next}} = (1 - \lambda)\theta + \alpha \nabla_1 R(\theta, \theta)$$

So a gradient ascent is done on player 1 with player 1 is playing against itself.
(Player 2 is a copy of player 1.)

Chess and go

Chess and go are 2-player zero-sum perfect information games with antisymmetric payoff.

Technically, the games are not perfectly (anti)symmetric, since one player moves first. (We can make game symmetric if first move is given to the players randomly.)

However, because the game is mostly symmetric, we will train one agent to play both players. So, there is only one policy π_θ .

(In a highly asymmetric 2-player game, you would train 2 policies for each player.)

Chess, shogi, and go

2-player perfect-information zero-sum turn-based games.

The top human chess player was defeated in 1997 by Deep Blue.

The top human shogi player was defeated in 2017 by Elmo.

The top human go player was defeated in 2016 by AlphaGo.



AlphaGo and AlphaGo Zero

AlphaGo[#] and AlphaGo Zero[%] combine search and learning to achieve super-human play.

These methods represent landmark scientific accomplishments in AI, and contain insights generalizable beyond the domains of games.

However, the exact techniques of MCTS has not yet been successfully adapted to the setup of LLM reasoning.

[#]D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, Mastering the game of Go with deep neural networks and tree search, *Nature*, 2016.

[%]D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, Mastering the game of Go without human knowledge, *Nature* 2017.

AlphaGo training step 1: π_{θ}^{IL}

Train policy π_{θ}^{IL} with imitation learning. From expert play record, construct dataset

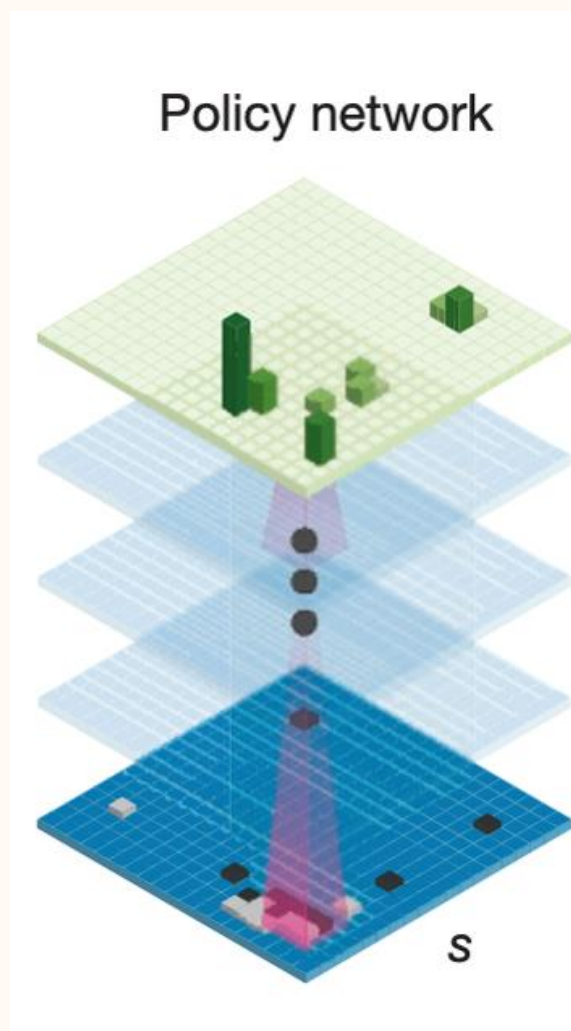
$$\mathcal{D} = \{(s, a) \mid \text{board position } s \text{ and next move } a \text{ from expert games}\}$$

Then, train the policy to predict the expert players' next move with loss

$$\mathcal{L}(\theta) = \sum_{(s,a) \in \mathcal{D}} \ell^{\text{CE}}(\pi_{\theta}^{\text{IL}}(\cdot \mid s), a)$$

Neural network architecture is a 13-layer conv net. (Architecture is later improved in AlphaGo Zero.)

π_{θ}^{IL} cannot yet beat expert humans.



AlphaGo training step 2: π_{θ}^{RL}

Train policy π_{θ}^{RL} through self-play. (i) Initialize $\pi_{\theta}^{\text{IL}} = \pi_{\theta}^{\text{RL}}$. (ii) Play

$$\pi_{\theta}^{\text{RL}} \text{ vs. } \pi_{\theta^-}^{\text{RL}}$$

where θ^- is an earlier version of the parameter θ . Choosing θ^- from a collection of past values stabilize training. Let $z = \pm 1$ indicate whether π_{θ}^{RL} won ($z = +1$ means π_{θ}^{RL} won). (iii) Perform undiscounted ($\gamma = 1$) policy gradient update on

$$\mathcal{R}(\pi_{\theta}^{\text{RL}}, \pi_{\theta^-}^{\text{RL}}) = \mathbb{E}_{\pi_{\theta}^{\text{RL}} \text{ vs. } \pi_{\theta^-}^{\text{RL}}} [z] = \mathbb{P}[\pi_{\theta}^{\text{RL}} \text{ wins}] - \mathbb{P}[\pi_{\theta^-}^{\text{RL}} \text{ wins}]$$

where first move (black or white stone) is randomized. Obtain an unbiased estimate of $\nabla_{\theta} \mathcal{R}(\pi_{\theta}^{\text{RL}}, \pi_{\theta^-}^{\text{RL}})$ with the deep policy gradient method gradient (without a baseline function)

$$g = \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) z$$

Here, the $\{(s_t, a_t)\}_{t=0}^{T-1}$ is the board states provided to and actions taken by π_{θ}^{RL} . (Of course, π_{θ}^{RL} should not be penalized or rewarded by actions taken by $\pi_{\theta^-}^{\text{RL}}$.)

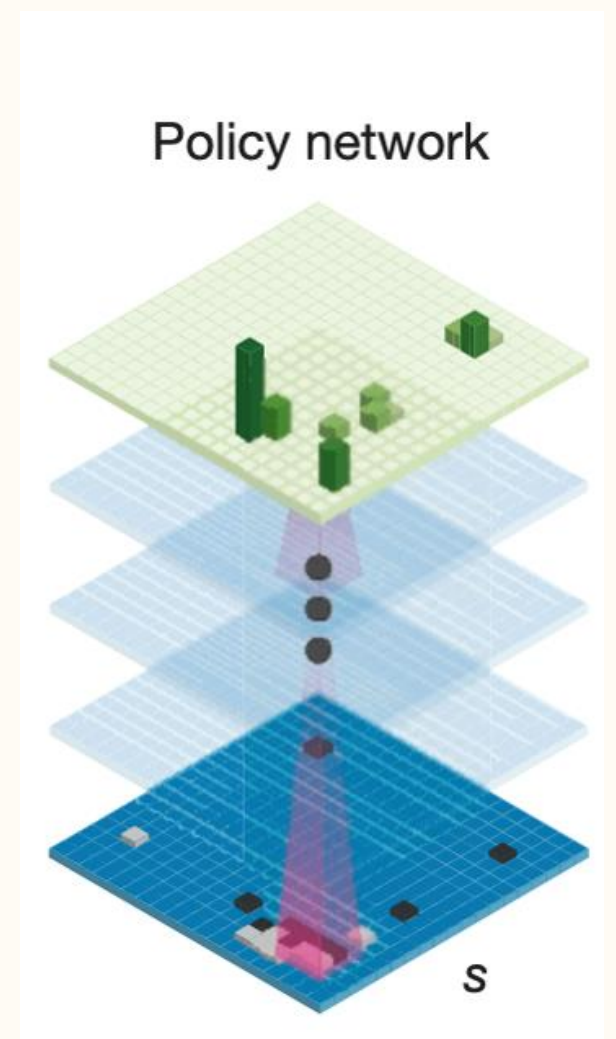
AlphaGo training step 2: π_{θ}^{RL}

Same neural network architecture as π_{θ}^{IL} .

π_{θ}^{RL} vs π_{θ}^{IL} wins 80%, but still cannot defeat human experts.

In principle, if NN is very large and self-play is done for very long, then π_{θ}^{RL} should converge to perfection (and beat all humans).

Under practical compute constraints, we need something more.



AlphaGo training step 3: V_ϕ

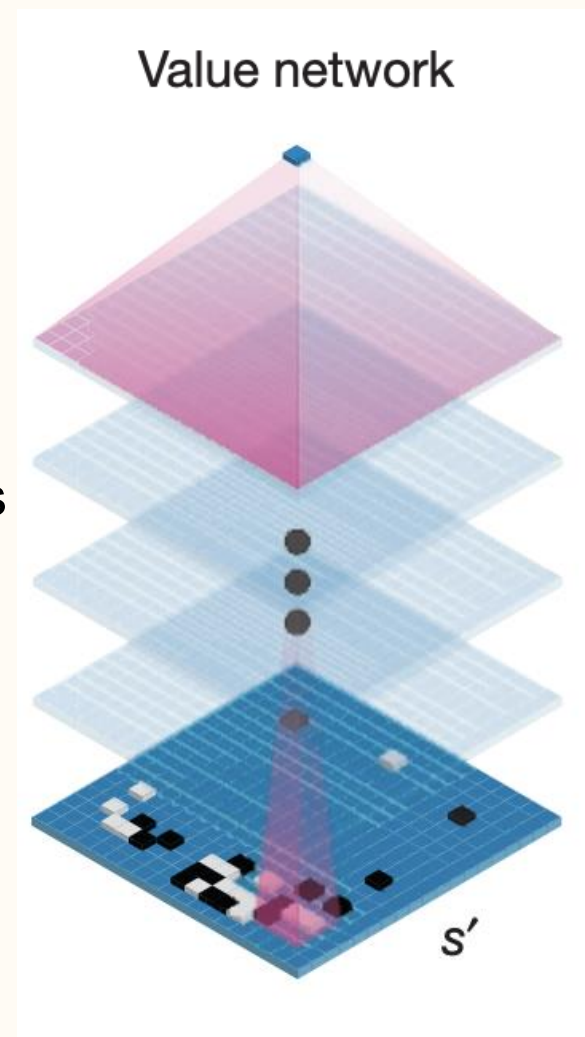
Train value network

$$V_\phi \approx V^{\pi_\theta^{\text{RL}}} \approx V^*$$

use standard Monte Carlo policy evaluation method.

Self-play with the strongest policy π_θ^{RL} to collect (s, z) pairs, where s is a board state and $z = \pm 1$ is whether player 1 goes on to eventually win or lose. Then, stochastic gradient descent with

$$g = \nabla_\phi \frac{1}{2} (V_\phi(s) - z)^2$$



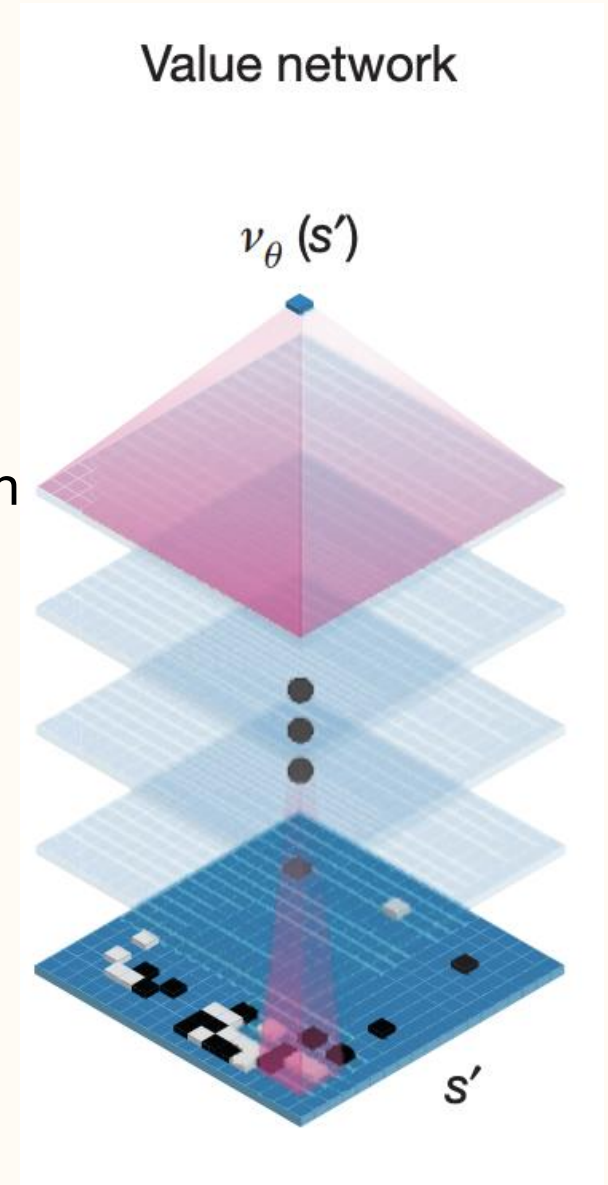
AlphaGo training step 3: V_ϕ

What about Q ? Should we also learn $Q_\phi \approx Q^{\pi_\theta^{\text{RL}}} \approx Q^*$?

Note, the dynamics is deterministic, with the transitioned state s' given by a function $f : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. So with $s' = f(s, a)$, we have

$$Q^*(s, a) = -V^*(f(s, a))$$

where the negative sign reflects the fact that the turn passes to the opponent, and both the Q - and V -value functions are defined with respect to the player whose turn it is to move.



AlphaGo training step 4: π_{ψ}^{fast}

Train a faster rollout policy π_{ψ}^{fast} with imitation learning.

Because π_{ψ}^{fast} uses a lightweight architecture, it has significantly faster inference time:
~2 microseconds for π_{ψ}^{fast} for compared to ~3 milliseconds for π_{θ}^{RL} .

So more than 1000x faster.

Not a strong policy, but it does understand the most basic principles.

Neural-net-only play

We have two reasonable options for neural-net-only play.

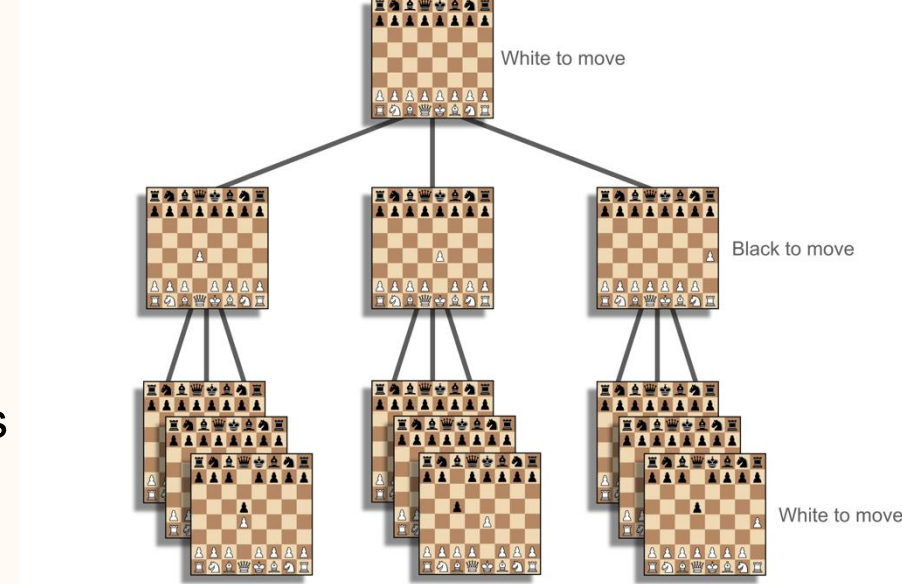
Option 1. Play with π_{θ}^{RL} .

Option 2. Play with the greedy policy $\operatorname{argmax}_{a \in \mathcal{A}} Q_{\phi}(s, a)$.

In principle, raw neural nets could beat humans with an exorbitant amount of compute, and we can estimate how much this would be. (More on this later.) However, under practical compute constraints, pure neural-net-based play is not enough.

Pure tree search

Consider all possible future board states that can be played. This approach is called minimax tree search, since I try to maximize my reward and my opponent tries to minimize my reward.



At every board state, the optimal action is

$$\operatorname{argmax}_{a \in \mathcal{A}} Q^*(s, a) = \operatorname{argmin}_{a \in \mathcal{A}} V^*(f(s, a))$$

Without learning, we don't know $V^*(s)$ for most states. However, if s is terminal, then we know $V^*(s)$ based on who won the game. So, expand the tree until the game ends, and recursively backtrack (dynamic programming) to find moves that take you to a winning board state.

This pure search-based method does not require any learning. However, this strategy is infeasible for moderately sized games because the computation size exponentially blows up.

Humans combine systems 1 and 2

It is instructive to reflect on how we humans play.

Human players have intuition on a set of reasonable moves and how advantageous a board state is. This intuition corresponds to system 1 thinking and is analogous to what neural networks learn.

Humans do not immediately act on these instincts, and instead deliberate through the future ramifications of moves. This deliberation corresponds to system 2 thinking and is analogous to search.

Human deliberation is not exhaustive. We do not consider all possible actions (limited width), and we do not mentally simulate until the end of the game (limited depth).

AlphaGo performs *neural guided search* through MCTS; Use neural networks to guide and focus the search on the relevant region of the game space.

Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search (MCTS) performs a *neural guided search*, selectively exploring parts of the tree based on the guidance of neural networks.

Principle #1: Gradually build up the tree, making it wider and deeper until we exhaust the given computational budget.

Monte Carlo Tree Search (MCTS)

Principle #2: Control the width of the tree by only considering “good” actions, defined as actions $a \in \mathcal{A}$ such that

- (a) $\pi_{\theta}^{\text{IL}}(a|s)$ or $\pi_{\theta}^{\text{RL}}(a|s)$ is high (actions that π_{θ} would play)
- (b) $Q_{\phi}(s, a)$ is high (actions that Q_{ϕ} thinks is good)
- (c) we have not yet deliberated on (consider a set of actions instead of focusing on the presumed top choice).

Solution: At every node s_t , choose the action a_t based on

$$a_t = \operatorname{argmax}_a \left\{ Q_{\phi}(s_t, a) + \rho \frac{\pi(a | s_t)}{1 + N(s_t, a)} \right\}$$

where $\rho > 0$ is a hyperparameter, $\pi = \pi_{\theta}^{\text{IL}}$ or $\pi = \pi_{\theta}^{\text{SL}}$, and $N(s_t, a)$ is the number of times the action has already been considered in the tree search.

Monte Carlo Tree Search (MCTS)

Principle #3: Starting at s_t consider a sequence of states based on the actions selected as previously described:

$$s_t \mapsto \tilde{s}_{t+1} \mapsto \tilde{s}_{t+2} \mapsto \cdots \mapsto \tilde{s}_L$$

We will have multiple leaf nodes \tilde{s}_L , and \tilde{s}_L will likely not reach the end of the game. (Lookahead is ~ 30 moves, i.e., ~ 60 plies, deep.) So, truncate the depth of the search by approximating $V^*(\tilde{s}_L)$ in the following two ways:

(a) Evaluate value network $V^*(\tilde{s}_L) \approx V_\phi(\tilde{s}_L)$

(b) Play N rollouts (until the end of the game) starting from \tilde{s}_L using the fast policy π_ψ^{fast} and form the Monte Carlo estimate :

$$V^*(\tilde{s}_L) \approx V^{\pi_\psi^{\text{fast}} \text{ vs. } \pi_\psi^{\text{fast}}}(\tilde{s}_L) = \mathbb{E}_{\pi_\psi^{\text{fast}} \text{ vs. } \pi_\psi^{\text{fast}}} [z] \approx \frac{1}{N} \sum_{i=1}^N z_i$$

(c) Form a weighted average (50-50 weight) of the estimates of (a) and (b).

Summary of MCTS

- Consider moves based on intuition encoded by π_{θ}^{IL} or π_{θ}^{RL} and Q_{ϕ} and construct a lookahead tree.
- Assess the strength of the leaf node position \tilde{s}_L by evaluating V_{ϕ} and by quickly playing until the end of the game using π_{ψ}^{fast} .
- Based on the estimate of \tilde{V} on the leaf node positions, backtrack and assign strength to each action at root node s_t .
- Commit to the best action a_t after this deliberation.

(Common principle of planning and control: Planning takes into account for many future steps, but only commit to one step. Once you reach the next step, do the planning again.)

Bibliography

The Monte Carlo Tree Search (MCTS) of AlphaGo is really the classical MCTS + additional enhancements

The ideas of MCTS was first described in

- B. Abramson. *The Expected-Outcome Model of Two-Player Games*, Columbia University Ph.D. Thesis, 1987.

and was first applied to the game of go in

- B. Brügmann, *Monte Carlo Go*, 1993.

The name “MCTS” was first coined in

- R. Coulom, Efficient selectivity and backup operators in Monte-Carlo tree search, *Computers and Games*, 2006.

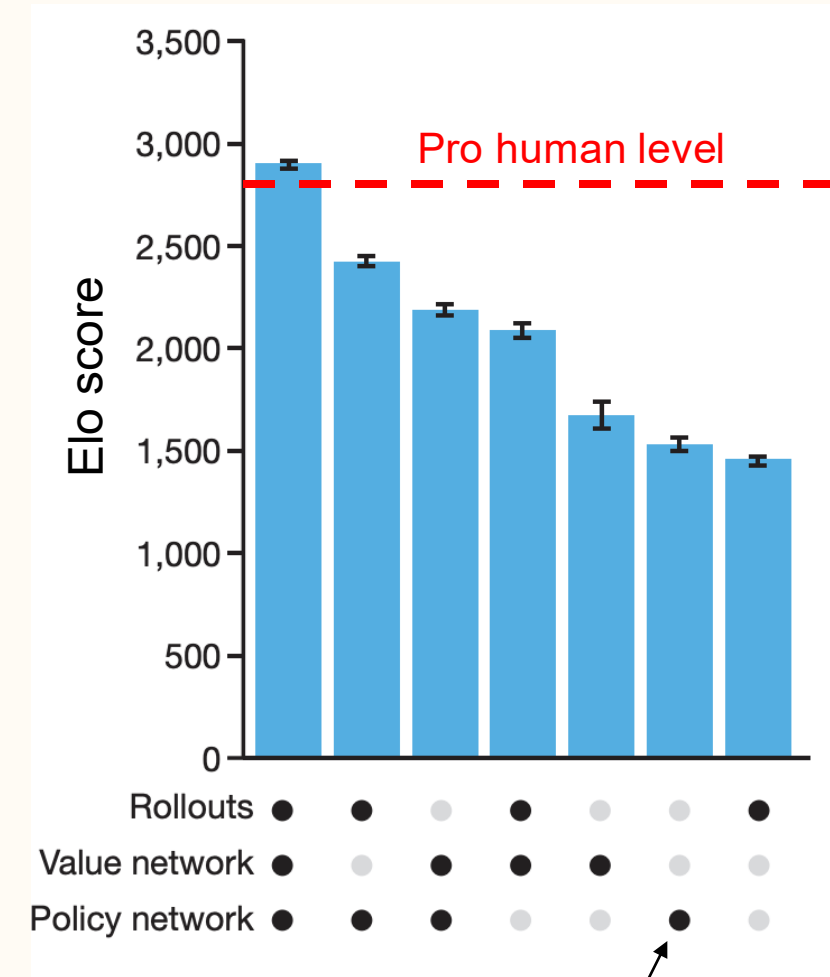
The action selection rule $a_t = \underset{a}{\operatorname{argmax}} \left\{ Q_\phi(s_t, a) + \rho \frac{\pi(a | s_t)}{1 + N(s_t, a)} \right\}$ is called Upper Confidence bounds applied to Trees (UCT) in analogy to the UCB selection rule for the multi-armed bandit setting. UCT was proposed by

- K. Levente and C. Szepesvári, Bandit based Monte-Carlo planning, *European Conference on Machine Learning*, 2006.

Why MCTS?

The MCTS algorithm is complicated. Why not do pure RL and just use π_{θ}^{RL} ?

- Without, π_{θ}^{IL} , the RL training of π_{θ}^{RL} makes no progress.
- Ablation studies show that rollouts (using π_{ψ}^{fast}), value network (V_{ϕ}), and policy network (π_{θ}^{IL} or π_{θ}^{RL}) are all necessary.



Pure RL with π_{θ}^{RL}

Improving AlphaGo to AlphaGo Zero

Problem: Can we eliminate the reliance on imitation learning and human expert play data?

Technical improvements:

- The neural network (NN) architecture of AlphaGo relied on design principles that were outdated by the time the AlphaGo paper was published. Use better architecture?
- AlphaGo training does not use search. MCTS is employed at inference time to produce a stronger agent. Can we use this stronger agent during training?

These improvements led to AlphaGo Zero.

Improved architecture of AlphaGo Zero

AlphaGo used 13-layer convnets for π and V .

AlphaGo Zero used a 40-layer convnet with BatchNorm and residual connections. Also, the policy π_{θ}^{EI} and value function V_{θ} are two heads with a shared base:

$$\left(\pi_{\theta}^{\text{EI}}(s), V_{\theta}(s)\right) = f_{\theta}(s)$$

Sharing the base makes sense because the two networks π_{θ}^{EI} and V_{θ} would use similar features.

Takeaway: Neural network architecture matters.

Training of AlphaGo Zero

Perform MCTS with π_{θ}^{EI} and V_{θ} . Perform self-play and improve $(\pi_{\theta}^{\text{EI}}, V_{\theta}) = f_{\theta}$.
(No rollouts are used, so π_{ψ}^{fast} is not needed.)

Expert iteration:

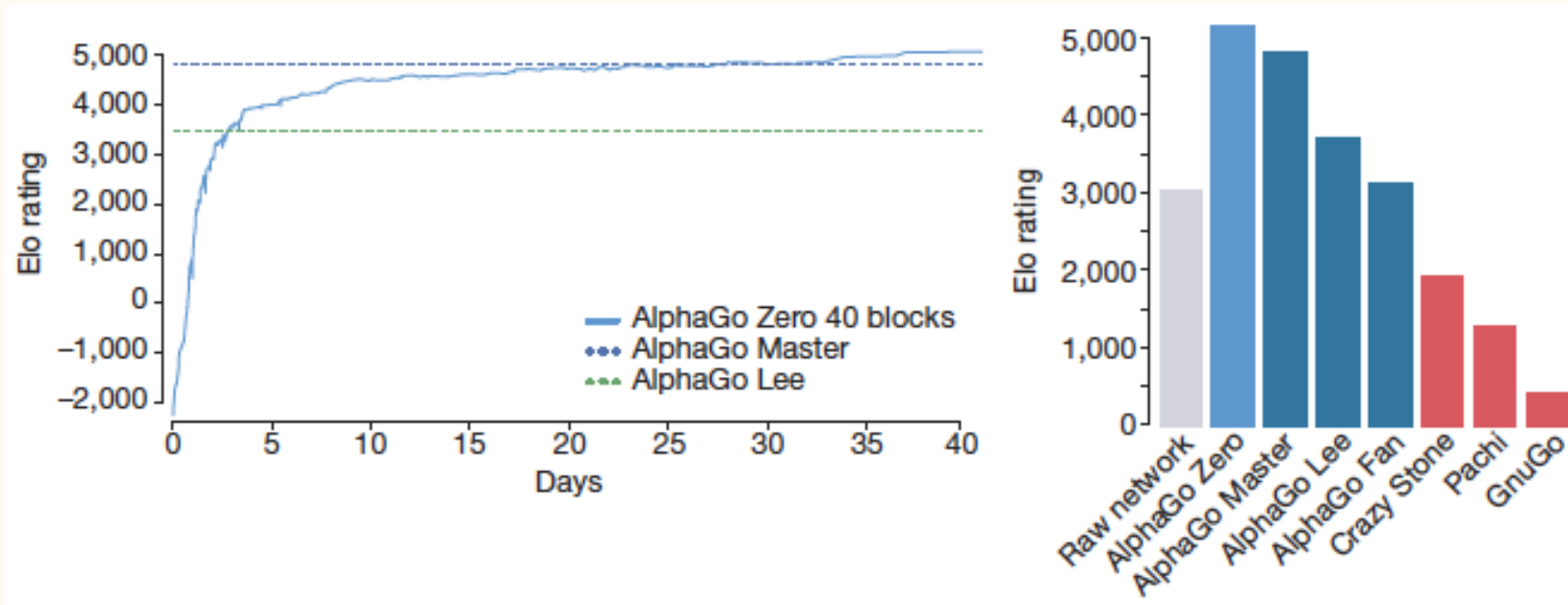
1. Self-play with MCTS($\pi_{\theta}^{\text{EI}}, V_{\theta}$) for many games.
 - Form action dataset $\mathcal{D}_a = \{(s_t, a_t)\}$ (actions from MCTS are stronger than π_{θ}^{EI})
 - Form win/loss dataset $\mathcal{D}_w = \{(s_t, z)\}$
2. Train π_{θ}^{EI} to mimic \mathcal{D}_a and V_{θ} to fit \mathcal{D}_w .

Key insight of expert iteration: Given a NN, improve the policy with NN+search, and train the NN to mimic the improved policy. Can be thought of as extensions of imitation learning and policy iteration. Much faster learning compared to pure RL.

AlphaGo Zero results

The expert iteration of AlphaGo Zero significantly simplifies the approach of AlphaGo.

Results: Much stronger policy trained with no human expert data or any handcrafted human knowledge.



D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. v an den Driessche, T. Graepel, and D. Hassabis, Mastering the game of Go without human knowledge, *Nature* 2017.

The tale of computer poker

Cepheus.

- M. Bowling, N. Burch, M. Johanson, O. and Tammelin, Heads-up limit hold'em poker is solved, *Science*, 2015.

Libratus. Spectacular victory in public match against professional players in 2017.

- N. Brown and T. Sandholm, Superhuman AI for heads-up no-limit poker: Libratus beats top professionals, *Science*, 2018.

Pluribus. Spectacular victory in public match against professional players in 2019.

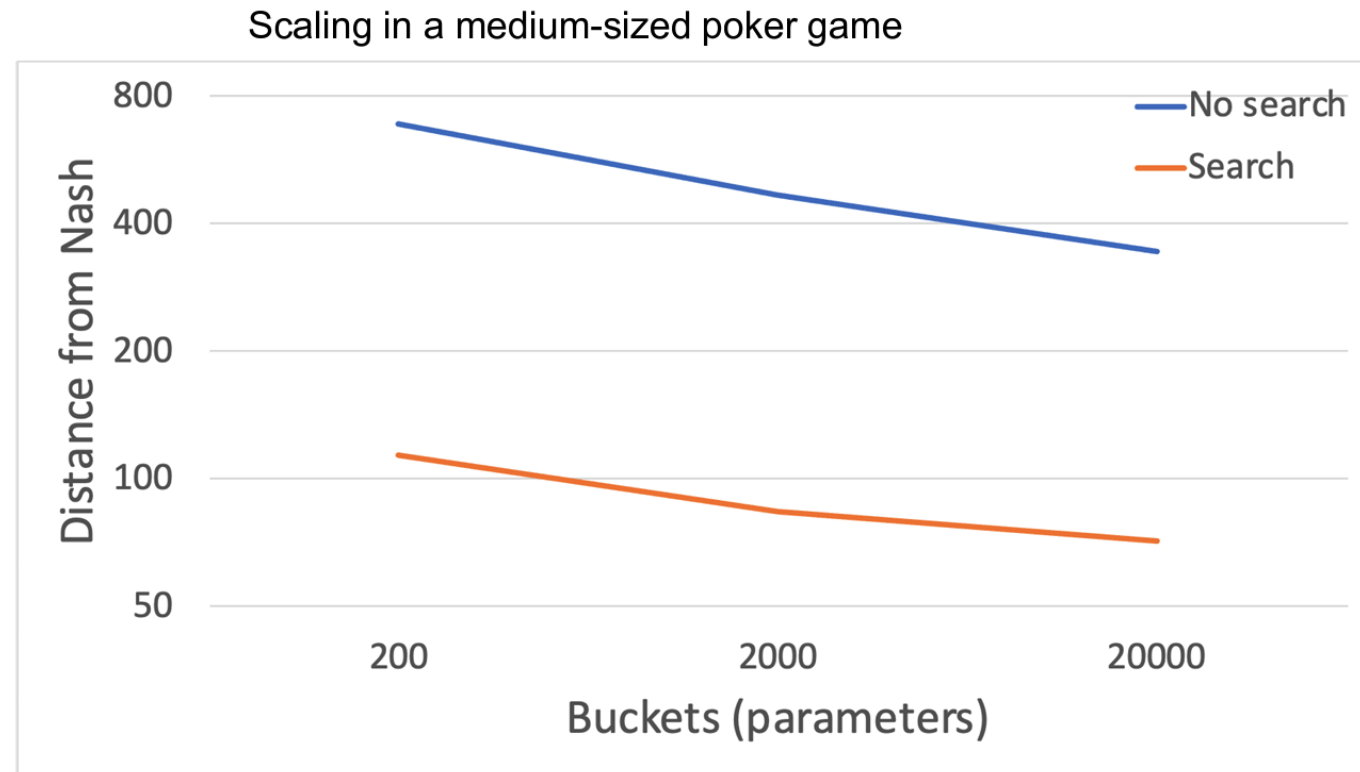
- N. Brown and T. Sandholm, Superhuman AI for multiplayer poker, *Science*, 2019.
- Jason Les: "very hopeless. You don't feel like there's anything you can do to win."
- Chris Ferguson: "Pluribus is a very hard opponent to play against. It's really hard to pin him down on any kind of hand."
- Jimmy Chou: "Whenever playing the bot, I feel like I pick up something new to incorporate into my game."

Test-time scaling with computer poker

While the poker bots utilize learning, the capability is primarily due to search.

The search methodology is quite different from MCTS. How to carry out search is often domain specific.

By leveraging search (test-time compute), one can achieve capabilities that would otherwise require an exorbitant amount of train-time compute.



Train- vs. test-time compute

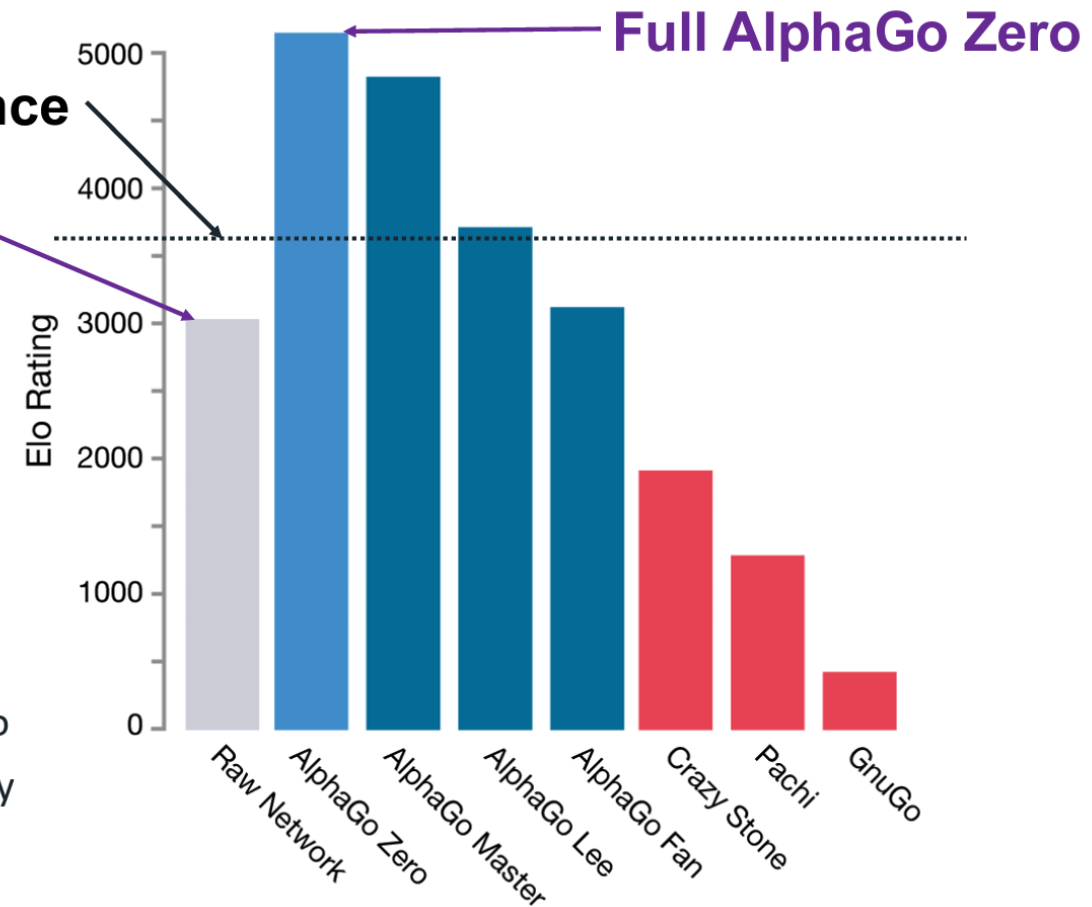
In fact, no raw neural network has yet to defeat top human players in go.

About 1000x train-time compute would be needed based on a back-of-the-envelope computation.

Superhuman performance

No test-time search

- Increasing Elo by 120 points requires either:
 - ~2x model size and training
 - ~2x test-time search
- To get the raw policy net from 3000 Elo to 5200 Elo, you would need to scale by **~100,000x**



Mathematically, why does test-time compute work?

A pre-trained policy must handle all possible game states. Finding a perfect policy, therefore, amounts to solving the entire game upfront.

Tree search considers only the game states that can occur given the current state. I.e., time-time compute solves for a much smaller subset of the overall game.

Takeaways from games

By leveraging test-time compute, one can spend extra compute on the given problem instance and deliberate to find good actions that would be otherwise impossible to find.

In domains with verifiable answers (win or lose in the case of games), expert iteration can significantly accelerate training compared to pure RL.