PA230: Reinforcement Learning

Petr Novotný

"Good and evil, reward and punishment, are the only motives to a rational creature; these are the spur and reins whereby all mankind are set on work and guided."

John Locke, *Some Thoughts Concerning Education* (1693)

Organizational Information

General

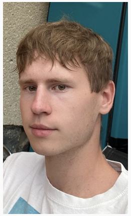
- Lecture: Thursdays 2-3:40p.m.
- Homework: see the interactive syllabus in IS
    - mainly binary classification (accepted/not accepted)
    - all your homeworks need to be marked as passed to proceed to exam
    - can (but do not have to) be done in pairs (pairs can differ across the individual assignments)
    - for those who passed, the teacher will receive feedback on the general quality of the solutions for each student - can be taken into account when determining the final grade (typically in students' favor)
- Exam:
    - oral
    - each attempt counts ? (unlike the Brázdil system)
    - in general, knowledge of anything mentioned on the slides can be required, unless explicitly marked with “nex” (like the Brázdil system)

Team

- Lecturer: Petr Novotný

- HW team:

Martin Kurečka



Václav Nevyhoštěný

Vít Unčovský

Communication

Official discord server:

`https://discord.gg/9mxTgYhcdB`

- Official communication forum of the course: falls under the university ethical guidelines.
- Use your real name for posting (you can set-up an account under your IS email if necessary).

Reading

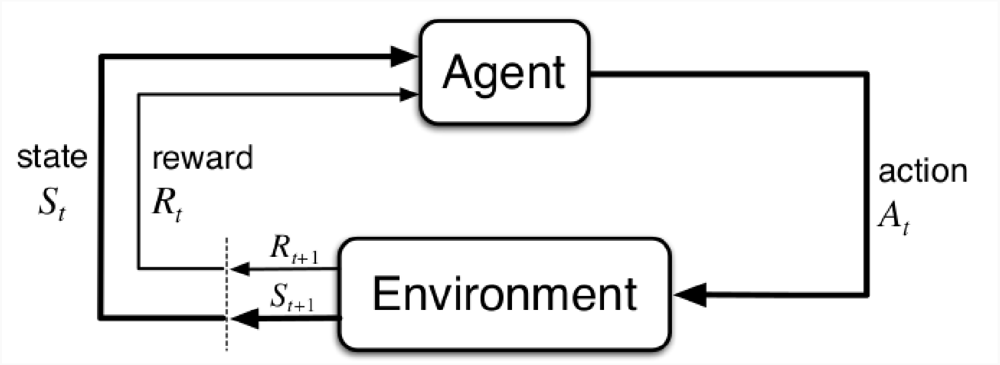
- Compulsory:
  - these slides,
  - material explicitly prescribed by these slides (not much).
- Recommended:
  - Sutton & Barto: Reinforcement Learning: An Introduction (2nd ed.), available at `http://incompleteideas.net/book/RLbook2020.pdf`
    - henceforth referenced as "S&B"
  - slides by David Silver `https://www.davidsilver.uk/teaching/`
  - CMU slides `https://www.andrew.cmu.edu/course/10-703/`
  - more specific literature recommendations will be given for each topic later

Reinforcement Learning: What, Why, When, How, & Other Questions

Types of machine learning

- unsupervised
  - spot ”useful” patterns in data
- supervised
  - given labeled data, predict labels on unlabeled data
- reinforcement
  - agents and decision-making
  - agency = “the ability to take action or to choose what action to take” (Cambridge Dictionary)

General RL scheme



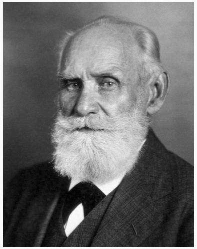
source: Sutton&Barto, p. 48

Keywords: sequential, dynamic, subject to uncertainty

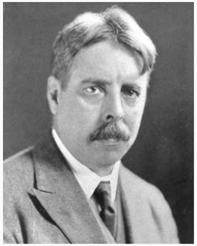
RL: Objective and approach

- Objective: Design a decision policy (= agent behavior) which prescribes to the agent how to act in different situations (states), typically so as to achieve some goal.
- Approach: Start with ($\pm$ random) behavior and adapt it based on past experience via the law of effect:
    - actions with good/bad consequences for the agent are more/less likely to be repeated by the agent (within the same context)

RL in psychology (nex)

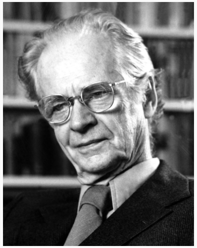


I.P. Pavlov
(1849-1936)
classical conditioning



E. Thorndike
(1874-1949)
law of effect

J.B. Watson
(1878-1958)
behaviorist manifesto



B.F. Skinner
(1904-1990)
radical behaviorism,
reinforcement,
rewards

Learning by trying & XX dilemma

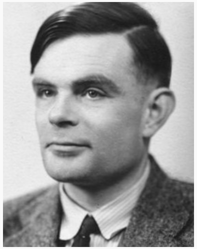
Underlying the RL approach is the idea of learning by trying:

- first, act more or less randomly (exploration)
    - integral part of early human development
- continually adapt behavior according to experience and feedback from the environment (exploitation)
    - strength of feedback $\approx$ strength of behavior adaptation

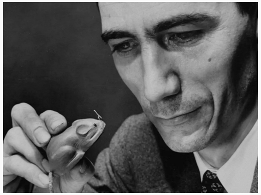
Balancing exploration and exploitation (XX) is a recurring theme in RL.

Incomplete history of RL in computer science I

“Learning by trying” machines and software, ad hoc approaches:



A. Turing
(1912-1954)
1948: theoretical
“pleasure & pain” system
to train computers



C. Shannon
(1916-2001)
1950: Theseus
maze-solving mouse

M. Minsky
(1927-2016)
1950s: analog neural net
machines (SNARCS)

And many more. . .
Recommended: S&B: Sec. 1.7.

Incomplete history of RL in computer science II

Mathematical foundations of sequential decision making:

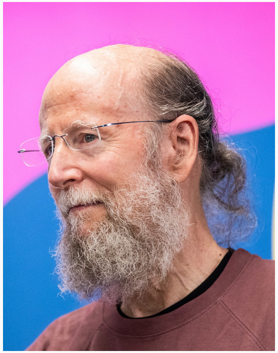
R. Bellman
(1920-1984)

R. Howard
(b. 1934)

- Formalization via Markov decision processes (MDPs)
- value iteration
  (attributed to Bellman, 1957)
- policy iteration
  (attributed to Howard, 1960)

Incomplete history of RL in computer science III

Since late 1980's: synthesis – learning by trial in MDPs


R. Sutton

A. Barto

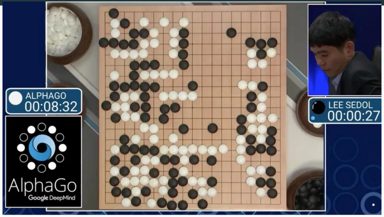
C. Watkins

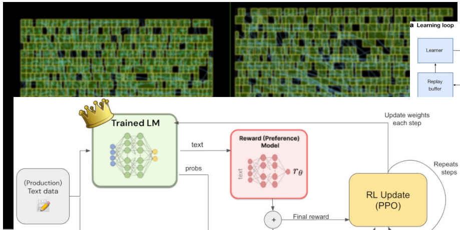
Temporal difference learning

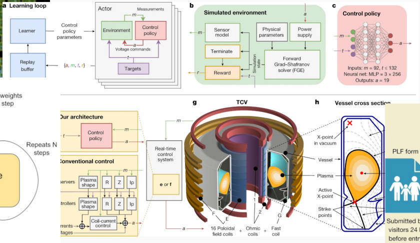
Q-learning

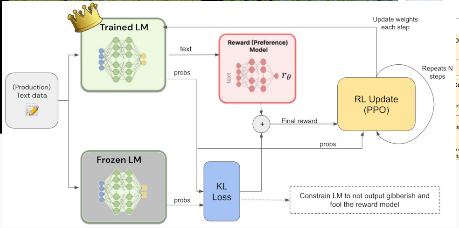
. . . and many more.

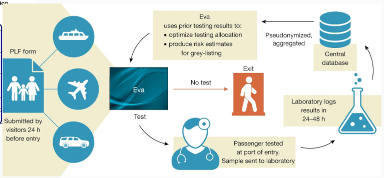
Successes of RL (nex)



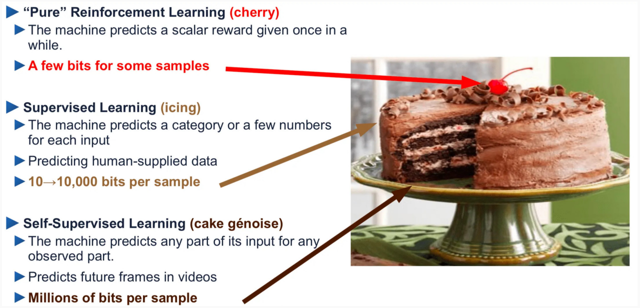







PLF form
Submitted by visitors 24 h before entry
Eva
uses prior testing results to:
• optimize testing allocation
• produce risk estimates for grey-listing
Pseudonymized, aggregated
Central database
Eva
No test
Exit
Test
Passenger tested at port of entry. Sample sent to laboratory
Laboratory logs results in 24–48 h

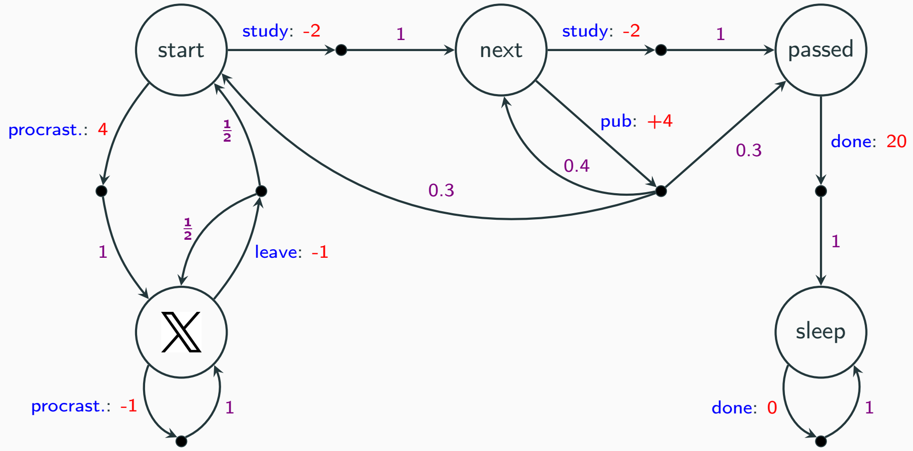
Words of caution (and controversy) (nex)


- **"Pure" Reinforcement Learning (cherry)**
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**
- **Supervised Learning (icing)**
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - **10→10,000 bits per sample**
- **Self-Supervised Learning (cake génoise)**
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**

Mathematical Foundations of Sequential Decision-Making

MDP Example

MDP with actions, rewards and transition probabilities.



Markov Decision Process

Given a set X, we denote $\mathcal{D}(X)$ the set of all probability distributions over X.

Definition 1

A Markov decision process (MDP) is a tuple $(\mathcal{S}, \mathcal{A}, p, r)$ where

- $\mathcal{S}$ is a set of states,
- $\mathcal{A}$ is a set of actions,
- $p\colon \mathcal{S} \times \mathcal{A} \to \mathcal{D}(\mathcal{S})$ is a probabilistic transition function,
- $r\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function.

We will shorten $p(s, a)(s')$ to $p(s' \mid s, a)$.

The p, r can be partial functions: action a is enabled in state s if both $p(s, a)$ and $r(s, a)$ are defined. We denote by $\mathcal{A}(s)$ the set of all actions enabled in s.

Dynamics of MDPs

- start in some initial state s_0
- MDP evolves in discrete time steps $t = 0, 1, 2, 3, \ldots$
- in each time step t, let s_t be the current state; then:
  - agent selects action $a_t \in \mathcal{A}(s_t)$
  - the environment responds with next state $s_{t+1} \sim p(s_t, a_t)$ and with immediate reward $r_{t+1} = r(s_t, a_t)$
  - t is incremented and the process repeats in the same fashion forever

Thus, the agent produces a trajectory $\tau = s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \ldots$.

τ is produced randomly (due to p and possibly also agent choices being probabilistic): it is a random variable and so are its components: we define random variables

- S_t = state at time step t
- A_t = action at time step t
- R_t = reward received just before entering S_t

Policies

Definition 2

A history is a finite prefix of a trajectory ending in a state, i.e., an object of type

$$s_0, a_0, r_1, s_1, a_1, r_2, \ldots, a_{t-1}, r_t, s_t \in (\mathcal{S} \cdot \mathcal{A} \cdot \mathbb{R})^* \mathcal{S}.$$

we denote by $last(h)$ the last state of a history h.

Definition 3

A policy is a function $\pi \colon (\mathcal{S} \cdot \mathcal{A} \cdot \mathbb{R})^* \mathcal{S} \to \mathcal{D}(\mathcal{A})$ which to each history h assigns a probability distribution over $\mathcal{A}(last(h))$.

A policy is by definition an infinite object!

MD policies

Definition 4

A policy π is:

- memoryless if $\pi(h) = \pi(h')$ whenever $\mathit{last}(h) = \mathit{last}(h')$ (we can view memoryless policies as objects of type $\pi \colon \mathcal{S} \to \mathcal{D}(\mathcal{A})$);
- deterministic if $\pi(h)$ always assigns probability 1 to one action, and zero to all others (we can view det. policies of objects of type $\pi \colon (\mathcal{S} \cdot \mathcal{A} \cdot \mathbb{R})^* \mathcal{S} \to \mathcal{A}$).

Definition 5

A policy π is MD (memoryless deterministic) if it is both memoryless and deterministic.

Dynamics of MDPs (more precise)

Given a distribution $\mathcal{I}$ of initial states and a policy π

- start in some initial state $s_0 \sim \mathcal{I}$
- MDP evolves in discrete time steps $t = 0, 1, 2, 3, \ldots$
- in each time step t, let h_t be the history produced so far; then:
    - agent selects action $a_t \in \mathcal{A}(s_t)$ according to π, i.e. $a_t \sim \pi(h_t)$
    - the environment responds with next state $s_{t+1} \sim p(s_t, a_t)$ and with immediate reward $r_{t+1} = r(s_t, a_t)$, the history is extended by a_t, r_t, s_{t+1},
    - t is incremented and the process repeats in the same fashion forever

Probability space induced by a policy

In particular, each policy π together with a distribution $\mathcal{I}$ of initial states induce a probability measure $\mathbb{P}^\pi$ over the trajectories of the MDP.[1]

We denote by $\mathbb{E}^\pi$ the associated expected value (expectation) operator.

We denote by $\mathbb{P}^\pi[E \mid S_0 = s]$ the probability of event E provided that the initial state is fixed to s (and similarly for expectations).

Exercise 6

In the "study" MDP, consider an MD policy π s.t. $\pi(\mathit{start}) = \mathit{study}$ and $\pi(\mathit{next}) = \mathit{pub}$. Compute the following quantities:

- $\mathbb{P}^\pi[\text{visit pub at least twice}' \mid S_0 = \mathit{start}'']$
- $\mathbb{P}^\pi[\text{visit pub at exactly twice} \mid S_0 = \mathit{start}'']$
- $\mathbb{E}^\pi[R_1]$
- $\mathbb{E}^\pi[R_3]$

[1] $\mathcal{I}$ is typically known from the context and hence omitted from the notation

Memorylessness

In this course, we will almost exclusively focus on memoryless policies. Hence, from now on, policy = memoryless policy. General policies will be referred to as history-dependent policies should the need arise.

Why memoryless?

Intuition: Markov property of MDPs: next step depends only on the current state and on action performed in the current step. Hence, intuitively there is no need for a policy to remember the past so as to "play well".

The sufficiency of memoryless policies does not extended to more general/complex decision-making settings (not covered in this course), such as:

- partially observable MDPs
- non-stationary environments
- quantile/risk-aware MDPs, etc.

Returns (payoffs)

Definition 7

Let $\gamma \in [0,1)$ be a discount factor.
For a trajectory $\tau = s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \dots$ we define the discounted return (or payoff) of τ to be the quantity

$$G(\tau) = r_1 + \gamma \cdot r_2 + \gamma^2 \cdot r^3 + \cdots \gamma^3 \cdot r^4 = \sum_{i=0}^{\infty} \gamma^i \cdot r_{i+1}.$$

Equivalently

$$G = \sum_{i=0}^{\infty} \gamma^i \cdot R_{i+1}.$$

Returns (variants)

- Finite horizon (FH): additionally, we are given a finite decision horizon $H \in \mathbb{N} \cup \{\infty\}$. The return is that counted only up to step H:

$$G^H = \sum_{i=0}^{H-1} \gamma^i \cdot R_{i+1}$$

  For finite H, the discount factor can be 1. $H = \infty$ corresponds to the original definition.
- Episodic returns: In episodic tasks, there is a distinguished set $Term \subseteq \mathcal{S}$ of terminal states which is guaranteed to be reached with probability 1 under any policy. We denote by T a random variable denoting the first point in time when we hit a terminal state. We count rewards only up to that time:

$$G^T = \sum_{i=1}^{T-1} \gamma^i \cdot R_{i+1}$$

  Can be modeled under original definition by "sink" states.

Types of returns: discussion

- We will typically omit the superscripts since the type of task considered will be known from the context.
- We have $G^H \to G$ (pointwise) as $H \to \infty$. I.e., finite-horizon returns with high enough H approximate the standard (infinite-horizon) case.
- In real world, we typically deal with FH or episodic tasks: we cannot wait infinite time to learn something from a trajectory. However, the infinite-horizon case can be viewed as a neat mathematical abstraction of the FH&episodic tasks, and the classical sequential decision-making theory is most developed for the infinite horizon case.

Policy and state values

Definition 8

Let π be a policy and s a state. The value of π in state s is the quantity

$$v^\pi(s) = \mathbb{E}^\pi[G \mid S_0 = s].$$

Exercise 9

Discuss the values of MD policies in our running example.

Definition 10

The (optimal) value of state s is the quantity

$$v^*(s) = \sup_\pi v^\pi(s).$$

Optimality

Definition 11

Let π be a policy and $\varepsilon > 0$. We say that π is ε-optimal in state s if

$$v^{\pi}(s) \geq v^{*}(s) - \varepsilon.$$

We say that π is optimal in s is it is 0-optimal in s, i.e. if

$$v^{\pi}(s) = v^{*}(s).$$

A policy is (ε-)optimal if it is (ε-)optimal in every state.

Existence of optimal policies

Theorem 12: (Classical result, not formally proven here)

Let $\mathcal{M}$ be a finite MDP (i.e., the state and action sets are finite) with infinite-horizon returns. Then there exists an optimal MD policy. Moreover, an optimal MD policy can be computed in polynomial time.

Agent control solved? NO! “Only” works if you can actually construct the MDP model of your environment and fit it into a computer. Otherwise, we use reinforcement learning.

Exact Planning with Known Model: Value & Policy Iteration

Goal of this lecture

Algorithms that compute the optimal value vector v^* and some optimal MD policy π^* given a full knowledge of an MDP $\mathcal{M}$.

Polynomial-time algorithm

MDPs can be solved by linear programming (LP)

$$\text{maximize } \vec{c} \cdot \vec{x}$$

$$\text{subject to } A \cdot \vec{x} \leq \vec{b}$$

- LP can be solved in polynomial time by so-called interior-point algorithms.
- However, we typically use other, MDP-specific algorithms: value iteration (VI) and policy iteration (PI). These are not polynomial-time in general, but typically faster on practical instances.
- Moreover, most truly RL algorithms can be seen as approximate generalizations of VI or PI (or both).

Example: Policy evaluation

Exercise 13

Consider all four MD policies in our running "pub or study" example that always try to quit when in X state. Compute the values of these policies in the initial state *start*.

Policy evaluation equations

Theorem 14

For any memoryless policy π and any state s it holds:

$$v^{\pi}(s) = \sum_{a \in \mathcal{A}(s)} \pi(a|s) \cdot \underbrace{\left[r(s,a) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s'|s,a) \cdot v^{\pi}(s') \right]}_{\overset{\text{def}}{=} q^{\pi}(s,a)} .$$

Bellman optimality equations

Theorem 15

The following holds for any state s:

$$v^*(s) = \max_{a \in \mathcal{A}(s)} \underbrace{\left[r(s,a) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s'|s,a) \cdot v^*(s') \right]}_{\overset{\textbf{def}}{=} q^*(s,a)}$$

- Note: the policy evaluation equations are a special case of the Bellman ones: given a policy π, we can consider an MDP $\mathcal{M}^\pi$ in which there is a single action $*$ enabled in each state and the probability of transition $s \xrightarrow{*} s'$ equals $\sum_{a \in \mathcal{A}(s)} \pi(a|s) \cdot p(s'|s,a)$. Then $\mathcal{M}^\pi$ mimics the behavior of π in $\mathcal{M}$ and Bellman eq's in $\mathcal{M}^\pi$ = evaluation equations for π in $\mathcal{M}$.
- But these equations are no longer linear! How do we solve them? Is the solution even unique?

Bellman update operator

The right-hand-side (RHS) of the Bellman equations can be viewed as an operator $\Phi\colon \mathbb{R}^{\mathcal{S}} \to \mathbb{R}^{\mathcal{S}}$: for any $\vec{x} \in \mathbb{R}^{\mathcal{S}}$, $\Phi(\vec{x})$ is a vector such that for any state s:

$$\Phi(\vec{x})(s) \overset{\text{def}}{=} \max_{a \in \mathcal{A}(s)} \left[r(s,a) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s'|s,a) \cdot \vec{x}(s') \right]$$

Exercise 16

In our running example, compute $\Phi(\vec{0})$.

Theorem 15 says that the optimal value vector v^* is a fixed point of Φ:

$$v^* = \Phi(v^*).$$

Mathematical hammers for Bellman

Lemma 17: (not proven here)

For any discount factor $\gamma \in [0, 1)$, the Bellman operator Φ is a contraction, i.e. for any pair of vectors $\vec{x}, \vec{y}$ it holds

$$\|\vec{x} - \vec{y}\|_\infty \leq \gamma \cdot \|\Phi(\vec{x}) - \Phi(\vec{y})\|_\infty.$$

Theorem 18: Banach fixed point theorem (classical calculus, not proven here)

A contraction mapping from a complete metric space (in particular, $\mathbb{R}^n$) to itself has a unique fixed point.

Exact characterization of v^*

Corollary 19

The optimal value vector is a unique solution of the Bellman optimality equations.

In particular, also the policy evaluation equations have a unique solution, equal to v^π. Since the policy evaluation equations are linear, their solution can be computed by Gaussian elimination.

But the general Bellman equations are not linear. How can ve solve them?

Banach fixpoint theorem (full)

Theorem 20: Banach fixed point theorem (full version, not proven here)

A contraction mapping Φ from a complete metric space (in particular, $\mathbb{R}^n$) to itself has a unique fixed point $\vec{z}$.

Moreover, $\vec{z}$ is the limit of iterative applications of Φ on any initial vector. I.e., for any $\vec{x}_0 \in \mathbb{R}^n$, the sequence $\vec{x}_0, \Phi(\vec{x}_0), \Phi(\Phi(\vec{x}_0)), \Phi^{(3)}(\vec{x}_0), \ldots$ converges to $\vec{z}$:

$$z = \lim_{i \to \infty} \Phi^{(i)}(\vec{x}_0)$$

Value iteration (VI; Bellman, 1957)

```
Algorithm 1: Value iteration
Input: MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, r)$
Output: Approximation $\tilde{v}$ of $v^*$
$x \leftarrow$ any vector from $\mathbb{R}^{|\mathcal{S}|}$ ;                // typically $\vec{0}$
$next \leftarrow x$;
repeat
    foreach $s \in \mathcal{S}$ do
        $next(s) \leftarrow \underbrace{\max_{a \in \mathcal{A}(s)} [r(s,a) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s'|s,a) \cdot \vec{x}(s')]}_{\Phi(x)(s)}$;
        $x \leftarrow next$
until termination condition;
```

Typical term. conditions:

- after a fixed no. of iterations (i.e., use for-loop with a fixed bound)
- after each component of x changes less then some given ε

How to use VI

By the Banach fixpoint theorem (and Lemma 17), the value of variable x VI converges to v^*. Can we recognize when is x "close enough" to $\vec{x}$?

In the following couple of theorems, let $\vec{x}_0, \vec{x}_1, \vec{x}_2, \ldots$ be the sequence of vectors computed by VI, i.e. $\vec{x}_0$ is arbitrary and $\vec{x}_{i+1} = \Phi(\vec{x}_i)$ for all $i \geq 0$.

Theorem 21: Stopping condition (not proven here)

For any $\varepsilon > 0$: if

$$\|\vec{x}_{i+1} - \vec{x}_i\|_\infty \leq \varepsilon \cdot \frac{1-\gamma}{\gamma},$$

then

$$\|\vec{x}_{i+1} - v^*\|_\infty \leq \varepsilon$$

How to use VI (2)

How fast can we get to the point where we are close enough?

Theorem 22: Speed of convergence (not proven here)

For all $i \geq 0$ it holds

$$\|\vec{x}^n - v^*\|_\infty \leq \frac{\gamma^n}{1-\gamma} \cdot \|\vec{x}_1 - \vec{x}_0\|_\infty.$$

In particular, if we terminate VI after

$$i = \left\lceil \frac{\log(\varepsilon) + \log\left(\frac{1-\gamma}{\|\vec{x}_1 - \vec{x}_0\|_\infty}\right)}{\log(\gamma)} \right\rceil$$

steps, then its output x_i will be an ε-approximation of v^*.

How to use VI (3)

Can we actually get some optimal values instead of approximations? First, note that VI computes optimal finite-horizon values:

Let $v^i = \sup_\pi \mathbb{E}^\pi[\sum_{i=1}^{H} \gamma^{i-1} \cdot R_i]$. The supremum is over all (i.e., history dependent) policies, since in the FH problem an optimal policy needs to track the number of elapsed (and thus remaining) steps: memory is needed for that.

Theorem 23: (Easy but important exercise)

If $\vec{x}_0 = \vec{0}$, then $\vec{x}_H = v^H$ for all $H \geq 0$.
Moreover, let π^H be a deterministic history-dependent policy such that for all $1 \leq i \leq H$, whenever there are i steps remaining till the horizon, the policy π^H selects in state s an action a s.t.

$$a = \arg\max_{a \in \mathcal{A}(s)} [r(s,a) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s'|s,a) \cdot \vec{x}_{i-1}(s')]$$

(with ties broken arbitrarily). Then π^H is an optimal H-step policy.

Greedy policies

Can we actually get optimal policy for the inf. horizon problem?

Definition 24: $\vec{x}$-greedy policy (very important)

Let $\vec{x} \in \mathbb{R}^{\mathcal{S}}$ be any vector. A $\vec{x}$-greedy policy is an MD policy π such that in any state s:

$$\pi(s) = \arg\max_{a \in \mathcal{A}(s)} [r(s,a) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s'|s,a) \cdot \vec{x}(s')].$$

How to use VI (4)

Theorem 25: Optimal inf.-horizon policy from VI (not proven here)

There is a number N polynomial in size of the MDP and exponential in the binary encoding size of γ such that a policy π that is $\vec{x}_N$-greedy is optimal in every state, i.e. $v^\pi = v^*$.

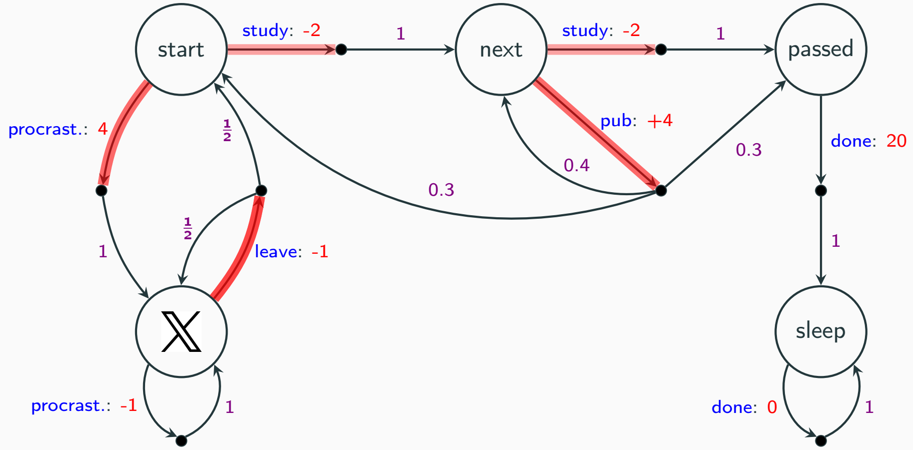
Note that once π is computed, v^π can be computed in polynomial time via policy evaluation equations.

Hence, VI can be said to solve MDPs in exponential time (and in polynomial time if the discount factor is assumed to be a fixed constant instead of an input parameter), though the approximate version is typically used in practice.

Note: the fact that some policy π is $\vec{x}$-greedy does not mean that $v^\pi \geq \vec{x}$! Homework: find a counterexample and post it to Discord.

However, for VI it can be shown that if $\|\vec{x}_{i+1} - \vec{x}_i\|_\infty \leq \varepsilon \cdot \frac{1-\gamma}{\gamma}$ (stopping condition from Theorem 21), then an $\vec{x}_{i+1}$-greedy policy is ε-optimal.

Policy improvement



$$v^{\pi} = (\qquad)$$

let π' be v^{π}-greedy $v^{\pi'} = (\qquad)$

let π'' be $v^{\pi'}$-greedy

Policy improvement theorem

Theorem 26: Policy improvement

Let π be a policy. If $\Phi(v^\pi) \geq v^\pi$, then any v^π-greedy policy π_g is at least as good as π, i.e. $\forall s \in \mathcal{S} : v^{\pi_g}(s) \geq v^\pi(s)$.
Moreover, if $\Phi(v^\pi)(s) > v^\pi(s)$ for some state s, then also $v^{\pi_g}(s') > v^\pi(s')$ for some state s'.

Returns from a given time step

For the proof of PIT and also many times later, we will need the following notation:

Definition 27: Important!

Let $\tau = s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \ldots$ be a trajectory and $t \in \mathbb{N}$ a time step. We define

$$G_t(\tau) = \sum_{i=t}^{H-1} \gamma^{i-t} \cdot r_{i+1} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots,$$

where $H \in \mathbb{N} \cup \{\infty\}$ or $H = T$ for episodic tasks.
We similarly define, for any policy π:

$$G_t^\pi = \mathbb{E}^\pi[G_t] = \mathbb{E}^\pi[\sum_{i=t}^{H-1} \gamma^{i-t} \cdot R_{i+1}].$$

Proof of PIT (setup)

We will define a sequence of policies $\pi_0, \pi_1, \pi_2, \dots$ s.t.:

- $\pi_0 = \pi$
- π_i behaves as π_g (i.e., selects the same actions in same states) for the first i steps, then "switches" back to behave as π:
- we also define $\pi_\infty = \pi_g$

We want: $v^{\pi_\infty}(s) \geq v^{\pi}(s)$ for all s.

Not hard to see: $v^{\pi_i} \rightarrow v^{\pi_\infty}$ as $i \rightarrow \infty$ (π_i behaves as π_∞ for longer and longer as i increases + discounting).
It suffices to show: $v^{\pi_i}(s) \geq v^{\pi}(s)$ for all $i \in \mathbb{N}$ and all $s \in \mathcal{S}$.

Proof of PIT (induction)

$v^{\pi_i}(s) \geq v^{\pi}(s)$ for all $i \in \mathbb{N}$ and all $s \in \mathcal{S}$

- $i = 0$: clear
- $i > 0$:

$$
\begin{aligned}
v^{\pi_i}(s) &= \mathbb{E}^{\pi_i}[R_1 + \gamma R_2 + \cdots + \gamma^{i-1} R_i + \gamma^i R_{i+1} + \cdots \mid S_0 = s] \\
&= \mathbb{E}^{\pi_i}[R_1 + \gamma R_2 + \cdots + \gamma^{i-2} R_{i-1} \mid S_0 = s] + \mathbb{E}^{\pi_i}[\gamma^{i-1} R_i + \gamma^i R_{i+1} \cdots \mid S_0 = s] \\
&= \mathbb{E}^{\pi_{i-1}}[R_1 + \gamma R_2 + \cdots + \gamma^{i-2} R_{i-1} \mid S_0 = s] + \mathbb{E}^{\pi_i}[\gamma^{i-1} R_i + \gamma^i R_{i+1} \cdots \mid S_0 = s] \\
&= \mathbb{E}^{\pi_{i-1}}[R_1 + \gamma R_2 + \cdots + \gamma^{i-2} R_{i-1} \mid S_0 = s] + \underbrace{\mathbb{E}^{\pi_i}[\gamma^{i-1} R_i + \gamma^i R_{i+1} \cdots \mid S_0 = s]}_{\text{Suppose we prove } \geq \mathbb{E}^{\pi_{i-1}}[\gamma^{i-1} R_i + \gamma^i R_{i+1} \cdots \mid S_0 = s]} \\
&\geq \mathbb{E}^{\pi_{i-1}}[R_1 + \gamma R_2 + \cdots + \gamma^{i-2} R_{i-1} + \gamma^{i-1} R_i + \gamma^i R_{i+1} \cdots \mid S_0 = s] \\
&\overset{IH}{\geq} v^{\pi}(s)
\end{aligned}
$$

Proof of PIT (induction, behavior at "reset")

We need: $\mathbb{E}^{\pi_i}[\gamma^{i-1}R_i + \gamma^i R_{i+1}\cdots \mid S_0 = s] \geq \mathbb{E}^{\pi_{i-1}}[\gamma^{i-1}R_i + \gamma^i R_{i+1}\cdots \mid S_0 = s]$.

$$\mathbb{E}^{\pi_i}[\gamma^{i-1}R_i + \gamma^i R_{i+1}\cdots \mid S_0 = s] = \gamma^{i-1}\cdot \mathbb{E}^{\pi_i}[R_i + \gamma R_{i+1}\cdots \mid S_0 = s]$$

$$= \gamma^{i-1}\cdot \sum_{s'\in\mathcal{S}} \mathbb{P}^{\pi_i}[S_{i-1} = s' \mid S_0 = s]\cdot \Big(r(s',\pi_i(s')) + \gamma \cdot \sum_{s''} p(s'' \mid s',\pi_i(s'))\cdot \mathbb{E}^{\pi_i}[G_i \mid S_i = s''] \Big)$$

$$= \gamma^{i-1}\cdot \sum_{s'\in\mathcal{S}} \mathbb{P}^{\pi_g}[S_{i-1} = s' \mid S_0 = s]\cdot \Big(r(s',\pi_g(s')) + \gamma \cdot \sum_{s''} p(s'' \mid s',\pi_g(s'))\cdot \mathbb{E}^{\pi}[G_i \mid S_i = s''] \Big)$$

$$= \gamma^{i-1}\cdot \sum_{s'\in\mathcal{S}} \mathbb{P}^{\pi_g}[S_{i-1} = s' \mid S_0 = s]\cdot \Big(r(s',\pi_g(s')) + \gamma \cdot \sum_{s''} p(s'' \mid s',\pi_g(s'))\cdot \underbrace{\mathbb{E}^{\pi}[G_i \mid S_i = s'']}_{v^\pi(s'')} \Big)$$

$$= \gamma^{i-1}\cdot \sum_{s'\in\mathcal{S}} \mathbb{P}^{\pi_g}[S_{i-1} = s' \mid S_0 = s]\cdot \underbrace{\Big(r(s',\pi_g(s')) + \gamma \cdot \sum_{s''} p(s'' \mid s',\pi_g(s'))\cdot \underbrace{\mathbb{E}^{\pi}[G_i \mid S_i = s'']}_{v^\pi(s'')} \Big)}_{=\Phi(v^\pi),\text{ since } \pi_g \text{ is } v^\pi\text{-greedy}}$$

$$= \gamma^{i-1}\cdot \sum_{s'\in\mathcal{S}} \mathbb{P}^{\pi_g}[S_{i-1} = s' \mid S_0 = s]\cdot \Big(r(s',\pi_g(s')) + \gamma \cdot \sum_{s''} p(s'' \mid s',\pi_g(s'))\cdot \underbrace{\mathbb{E}^{\pi}[G_i \mid S_i = s'']}_{v^\pi(s'')} \Big)$$

Policy iteration (PI; Howard, 1960)

Algorithm 2: Policy iteration

```
Input: MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, r)$
Output: Optimal MD policy $\pi^*$ for $\mathcal{M}$, its value vector $v^*$
$\pi \leftarrow$ arbitrary MD policy ;
$v \leftarrow v^\pi$ ;                // e.g. by solving linear policy evaluation equations
while $\Phi(v) \neq v$ do
    foreach $s \in \mathcal{S}$ do
        $\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}(s)} [r(s, a) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s'|s, a) \cdot v(s')]$
    $v \leftarrow v^\pi$
return $\pi, v$
```

PI correctness & complexity

Theorem 28

Policy iteration terminates after at most exponentially many iterations. Upon termination, it returns an optimal MD policy.

Proof:

- Optimal upon termination: v^π is a fixpoint of Φ when terminating: optimality follows from Corollary 19.
- Terminates: π always stores an MD policy and there are finitely many of these. We will show that no single MD policy appears in more than one iteration of PI.
  Consider any iteration and let v, v' be the contents of variable v before and after the iteration. We will show that unless $\Phi(v) = v$, it holds $v' > v$, i.e. $v' \geq v$ componentwise with strict inequality in some component. Hence, $v = v^\pi$ strictly increases during PI, so no π can appear twice.

PI: correctness proof

$v' \geq v$:

We verify assumptions of PIT: $\Phi(v) \geq v$. Recall $v = v^{\pi}$. For all $s \in \mathcal{S}$:

$$\begin{aligned}
\Phi(v)(s) &= \max_{a \in \mathcal{A}(s)} [r(s,a) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s' \mid s, a) \cdot v(s')] \\
&\geq r(s, \pi(s)) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s' \mid s, \pi(s)) \cdot v^{\pi}(s') \\
&= v^{\pi}(s) = v(s).
\end{aligned}$$

By PIT, $v' = v^{\pi'} \geq v^{\pi} = v$ (here π' is the v-greedy policy).

PI: correctness proof II

It remains to prove that $v' > v$ or PI terminates. Assume that $v' = v$. Then for all $s \in \mathcal{S}$:

$$\begin{aligned}
v'(s) &= r(s, \pi'(s)) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s' \mid s, \pi'(s)) \cdot v^{\pi'}(s') \\
&= r(s, \pi'(s)) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s' \mid s, \pi'(s)) \cdot v^{\pi}(s') \quad \text{(assumption)} \\
&= \max_{a \in \mathcal{A}(s)} [r(s, a) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s' \mid s, a) \cdot v^{\pi}(s')] \quad (\pi' \text{ is } v = v^{\pi}\text{-greedy}) \\
&= \max_{a \in \mathcal{A}(s)} [r(s, a) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s' \mid s, a) \cdot v^{\pi'}(s')] \quad \text{(assumption)} \\
&= \Phi(v')(s),
\end{aligned}$$

so PI terminates at this point. Complexity?

PI&VI: discussion

- We know that MDPs have a linear programming (LP) formulation. PI is basically a variant of a simplex method for this LP, using a special pivoting rule.
- PI typicaly requires less iterations to converge than VI, though each iteration is more expensive (policy eval.)
- Both PI and VI typically work well in practice for MDPs whose explicit transition table fits inside a computer. Which of the two is faster is rather domain-specific.

PI variants

Can we get rid of the expensive policy evaluation by linear system solving?

Yes: we can approximate the value of the current policy π by applying VI on the MDP $\mathcal{M}^{\pi}$, for either fixed number of steps or until v does not change much. Often appearing in RL textbooks:

Policy iteration with approximate evaluation

```
$\pi \leftarrow$ arbitrary MD policy; $v \leftarrow$ arbitrary vector;
repeat
    $v \leftarrow \texttt{Eval}(\pi, v)$;
    foreach $s \in \mathcal{S}$ do
        $\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}(s)}[r(s,a) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s'|s,a) \cdot v(s')]$
until $\pi$ *has not changed*;
return $\pi, v$
Function $\texttt{Eval}(\pi, v)$:
    $v' \leftarrow v$;
    repeat
        foreach $s \in \mathcal{S}$ do
            $v(s) \leftarrow v'(s)$;
            $v'(s) \leftarrow r(s, \pi(s)) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s' \mid s, \pi'(s)) \cdot v(s')$
    until $\|v - v'\|_\infty \leq \varepsilon$;
return $v'$
```

Convergence of PI variants

- The algorithm on previous slide still converges to an optimal policy provided that ε is small enough.
- If we replaced the "π not changed condition" with the original "$\Phi(v) = v$" condition, the algorithm might not terminate, since the VI is only guaranteed to reach a true fixpoint in the limit. However, v would still converge to v^* and thus π would eventually become equal to an optimal policy.
- The previous point holds even in the very degenerate case when we do just one iteration of VI per policy evaluation! See next slide.

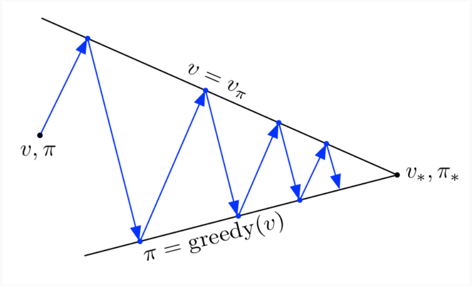
Curiously looking approximate PI

```
$v \leftarrow$ arbitrary vector;
$\pi \leftarrow v$-greedy MD policy ;
repeat
    foreach $s \in \mathcal{S}$ do
        $v'(s) \leftarrow r(s, \pi(s)) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s'|s, \pi(s)) \cdot v(s')$;
    $v \leftarrow v'$;
    foreach $s \in \mathcal{S}$ do
        $\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}(s)} [r(s, a) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s'|s, a) \cdot v(s')]$
until $\Phi(v) = v$;
return $\pi, v$
```

This is just VI in disguise!

Generalized policy iteration

$$\begin{tikzcd}
\boxed{\text{Evaluate}} \arrow[d, bend left, "\text{push } v \text{ towards } v^\pi"] \\
\boxed{\text{Improve}} \arrow[u, bend left, "\text{make } \pi \text{ (more) } v\text{-greedy}"]
\end{tikzcd}$$



Source: Sutton&Barto, p. 87

Tabular Methods for Model-Free Reinforcement Learning

Model-free

We will still be working with MDPs. But for a bunch of the following lectures, we will not (necessarily) have access to, e.g.:

- a table containing explicit enumeration of all states/actions
- a table containing the description of p or r
- the ability to compute the probability vector $\delta(s, a)$ or the reward signal $r(s, a)$ given s and a (having this = gray-box model of the MDP)

Sampling from MDP

But the MDP is still there "behind the scene". In particular, we:

- know how the states of the MDP look like
    - (e.g. robot state = all possible output values of its sensors)
- know how the actions of the MDP look like
    - (e.g. robot = all possible signals that can be sent to the actuators)
- can, for any $s \in \mathcal{S}$, enumerate $\mathcal{A}(s)$
    - could be weakened, but simplifies things
- given $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, we can sample the next state $s' \sim p(s, a)$ and receive the reward $r(s, a)$.

Sampling from a policy

Given an effective representation of a policy π, we can sample a trajectory $s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \ldots$ by performing, for each $t \in \{0, \ldots, T\}$:

- sample $a_t \sim \pi(s_t)$
- query the environment for $s_{t+1} \sim p(s_t, a_t)$ and $r_{t+1} = r(s_t, a_t)$
- increment t

Tabular = value estimates and policies represented as tables (e.g. $Q(s, a)$ for each state s and action a used in s – explicit representation might only be needed for states/actions actually encountered).

Basic classification of (tabular) RL algorithms

Three independent axes:

problem	on/off	updates
policy evaluation (value prediction)	on-policy	Monte Carlo
vs.	vs.	$\updownarrow$
control	off-policy	temporal difference

Assumptions: successor-dependent rewards & episodic tasks

Since we do no longer have the knowledge of the transition dynamics p, we cannot freely interchange MDPs with rewards functions of type $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ and $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$ via the equation $r(s, a) = \sum_{s' \in \mathcal{S}} p(s' \mid s, a) \cdot r(s, a, s')$.
Hence, to maintain generality (and correspondence to e.g. Gymnasium environments) we will assume reward functions of type $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$.

We will assume episodic returns: each trajectory terminates with probability 1 at some (possibly random) time step T. Termination can be defined e.g. by reaching some terminal state or by running out of some fixed decision horizon (in Gymnasium, this is sometimes called truncation):

$$G = \sum_{i=0}^{T-1} \gamma^i \cdot R_{i+1}.$$

Episode = one high-level iteration of an RL algorithm, corresponding of sampling a single trajectory from some policy.

Monte Carlo Methods

MC evaluation

Policy evaluation: given an effective representation of a policy π, estimate v^π (or q^π).

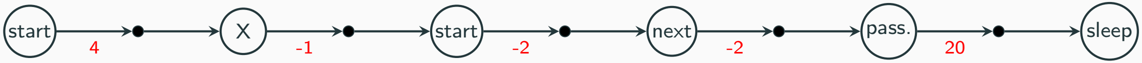
Naive Monte Carlo: Sample from π: if $\{\tau_1, \tau_2, \ldots, \tau_n\}$ are trajectories (episodes) independently sampled under π from the same initial state s, then $\frac{1}{n}\sum_{i=1}^n G(\tau_i) \to v^\pi(s)$ as $n \to \infty$ due to law of large numbers (LLN).

But this throws away a lot of valuable information! E.g. what if we want to estimate the whole v^π?

First-visit MC

For each s, we estimate $v^\pi(s)$ as an average of sample returns $Ret(s)$ which is formed as follows:

- initially, $Ret(s) = \emptyset$ for all s
- we then sample trajectories until timeout:
    - for each sampled trajectory τ and each state s, we identify the first occurrence of s on τ: let this be at timestep t; we add $G_t(\tau)$ to $Ret(s)$

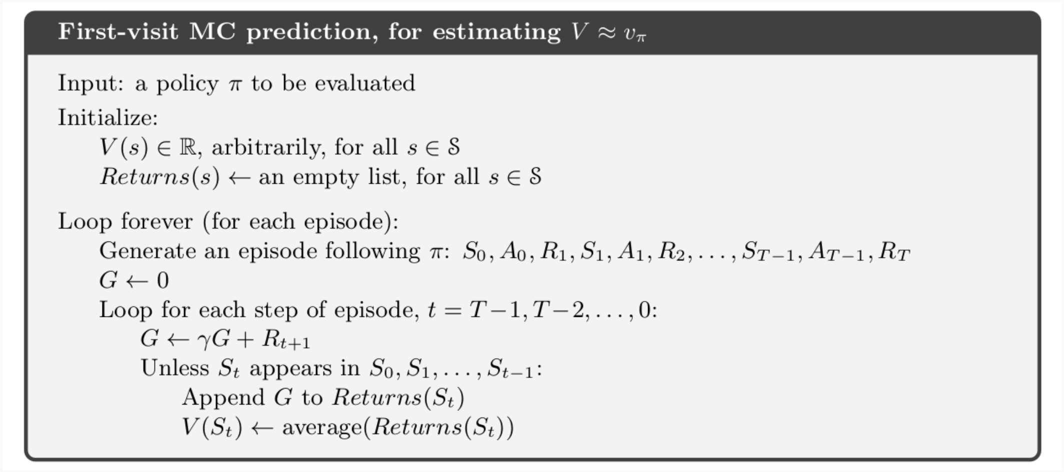


Sub-trajectory starting at the first appearance of s can be seen as a trajectory sampled from π when s is the initial state! (Since we consider memoryless π.)

Theorem 29

As $|Ret(s)| \to \infty$, the average of $Ret(s)$ converges to $v^\pi(s)$. Moreover, the average of $Ret(s)$ is an unbiased estimate of $v^\pi(s)$ (as long as $Ret(s) \neq \emptyset$).

First-visit MC (pseudocode)



First-visit MC prediction, for estimating $V \approx v_\pi$

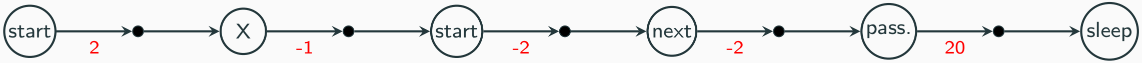
```
Input: a policy $\pi$ to be evaluated
Initialize:
    $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
    Generate an episode following $\pi$: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Unless $S_t$ appears in $S_0, S_1, \ldots, S_{t-1}$:
            Append $G$ to $Returns(S_t)$
            $V(S_t) \leftarrow$ average($Returns(S_t)$)
```

Source: Sutton&Barto, p.92

Every-visit MC

For each s, we estimate $v^\pi(s)$ as an average of sample returns $Ret(s)$ which is formed as follows:

- initially, $Ret(s) = \emptyset$ for all s
- we then sample trajectories until timeout:
    - for each sampled trajectory τ and each state s, and each t such that $S_t(\tau) = s$ we add $G_t(\tau)$ to $Ret(s)$



The sample returns added to $Ret(s)$ within the same episode are not independent! Hence, the estimate is biased, though the bias vanishes in the limit:

Theorem 30

As $|Ret(s)| \to \infty$, the average of $Ret(s)$ converges to $v^\pi(s)$.

MC convergence

Optional reading: More on MC estimate bias, variance, and convergence in:

Singh, S.P. and Sutton, R.S.: Reinforcement Learning with Replacing Eligibility Traces.
In *Machine Learning* 22:123–158. Kluwer, 1996.
(Section 3, particularly 3.3 and onwards, you can skip Theorem 4.)

Towards MC control

Control = computation of “good” policy for a given environment. (Ideally, the policy should get closer to the optimal policy the more episodes we sample.)

We know (PIT): given a policy π a v^π-greedy policy is at least as good as π:

$$\pi_g(s) = \underset{a \in \mathcal{A}(s)}{\arg\max} \left[\sum_{s' \in \mathcal{S}} p(s' \mid s, a) \cdot \big(r(s, a, s') + \gamma \cdot v^\pi(s') \big) \right]$$

Do we have an algo? There is an issue:

MC control with q-values

Recall:

$$q^\pi(s,a) \overset{\text{def}}{=} \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \cdot \left(r(s,a,s') + \gamma \cdot v^\pi(s').\right)$$

Thus, the v^π-greedy policy π_g can be defined as:

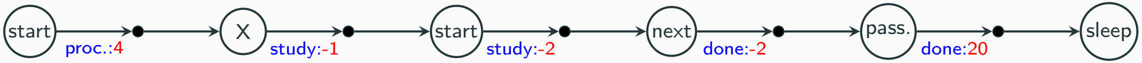
$$\pi_g(s) = \arg\max_{a \in \mathcal{A}(s)} \underbrace{q^\pi(s,a)}_{\text{Estimate by MC.}}$$

MC for q-value estimation

Analogous to value estimation, e.g. first-visit:

For each s, a, we estimate $q^{\pi}(s, a)$ as an average of sample returns $Ret(s, a)$ which is formed as follows:

- initially, $Ret(s, a) = \emptyset$ for all s
- we then sample trajectories until timeout:
    - for each sampled trajectory τ and each state-action pair (s, a), we identify the first t such that $S_t(\tau) = s \wedge A_t(\tau) = a$; we add $G_t(\tau)$ to $Ret(s)$



Similarly for every visit. Convergence guarantees the same as for state values.

Infinite exploration and exploring starts

Issue: MC only estimates $q^\pi(s, a)$ if:

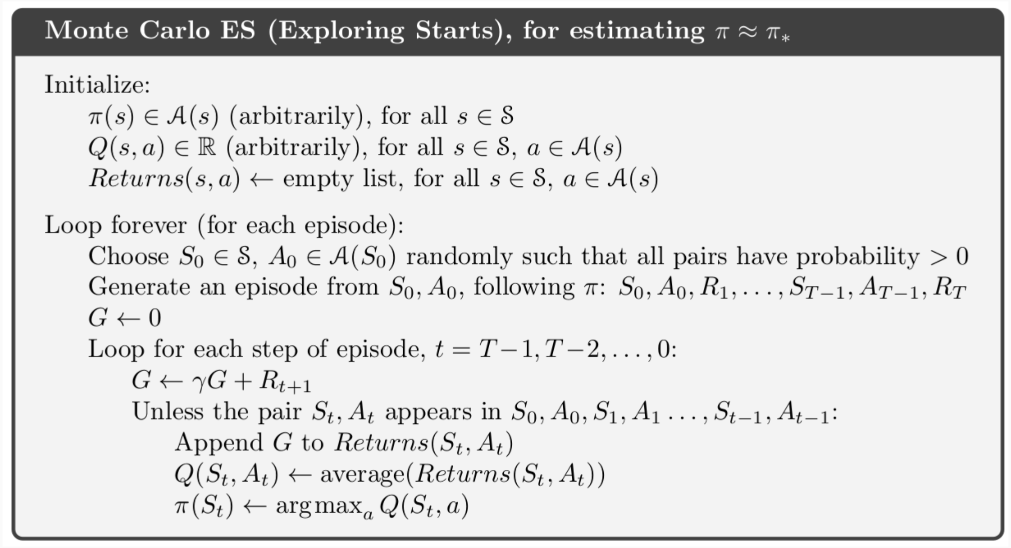
- s guaranteed to be visited with positive probability in each episode
- $\pi(a \mid s) > 0$.

Definition 31: Infinite exploration

A RL algorithm has infinite exploration (IE) if, during the infinite execution of the algorithm, each state-action pair (s, a) is visited infinitely often with probability 1.

One way of achieving IE is through exploring starts (ES): each episode begins with (typically uniformly) randomly selected s_0 and a_0. This is achievable when training, e.g., in simulated environments but might be difficult/impossible in real-world environments.

MC control with exploring starts



Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

```
Initialize:
    $\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
    $Q(s, a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
    $Returns(s, a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):
    Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability $> 0$
    Generate an episode from $S_0, A_0$, following $\pi$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Unless the pair $S_t, A_t$ appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
            Append $G$ to $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
            $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$
```

Source: Sutton&Barto, p.99

IE through ε-soft policies

Exploring starts are not always feasible. Alternative: make the sampled policy itself exploratory.

Definition 32: ε-soft policy

A policy π is ε-soft if for every $s \in \mathcal{S}$ and every $a \in \mathcal{A}(s)$ it holds $\pi(a|s) \geq \frac{\varepsilon}{|\mathcal{A}(s)|}$.

Definition 33: ε-greedy policy

Let $v \in \mathbb{R}^{\mathcal{S}}$ be a value vector. A policy π is v-ε-greedy if for every state $s \in \mathcal{S}$ there is action $a^* = \arg\max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} p(s' \mid s, a) \cdot \big(r(s, a, s') + \gamma \cdot v(s')\big)$ such that for any action $a \in \mathcal{A}(s)$ it holds:

$$\pi(a|s) = \begin{cases} \frac{\varepsilon}{\mathcal{A}(s)} & \text{if } a \neq a^* \\ 1 - \varepsilon + \frac{\varepsilon}{\mathcal{A}(s)} & \text{if } a = a^*. \end{cases}$$

Interpretation: with prob. ε: play uniformly at random; with prob. $1 - \varepsilon$: play greedily.

ε-softing

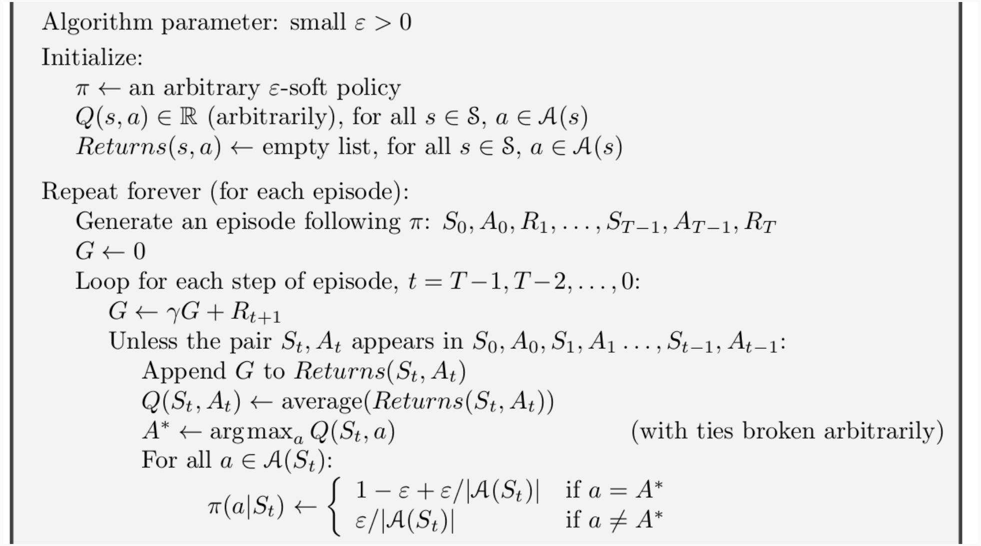
Definition 34

Let π be a policy. An ε-softing of π is a policy π_ε defined as follows: in each state s

- with probability ε, π_ε selects an action uniformly at random;
- with probability $1 - \varepsilon$, π_ε selects $a \sim \pi(s)$.

I.e., an ε-greedy policy can be alternatively defined as ε-softing of a greedy policy.

MC control with ε-greedy policies



```
Algorithm parameter: small $\varepsilon > 0$
Initialize:
    $\pi \leftarrow$ an arbitrary $\varepsilon$-soft policy
    $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
    $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):
    Generate an episode following $\pi$: $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \dots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Unless the pair $S_t, A_t$ appears in $S_0, A_0, S_1, A_1 \dots, S_{t-1}, A_{t-1}$:
            Append $G$ to $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
            $A^* \leftarrow \arg\max_a Q(S_t, a)$                    (with ties broken arbitrarily)
            For all $a \in \mathcal{A}(S_t)$:
                $\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$
```

source: Sutton&Barto, p. 101

Policy iteration for ε-soft policies

Theorem 35

Let π be an ε-soft policy and let π' be a v^π-ε-greedy policy. Than $v^{\pi'} \geq v^\pi$ (componentwise). Moreover, the two value vectors are equal if and only if bot π and π' are optimal among all ε-soft policies; i.e. if, for every state s:

$$v^\pi(s) = \sup_{\bar{\pi} \text{ that is } \varepsilon\text{-soft}} v^{\bar{\pi}}(s).$$

Proof: Required reading: Sutton&Barto, p.101-103.

Incremental computing of averages

Given a sample $\{n_1, n_2, \dots, n_{k+1}\}$ and average $A = avg(\{n_1, n_2, \dots, n_k\})$, how to compute $A' = avg(\{n_1, n_2, \dots, n_k, n_{k+1}\})$ without recomputing the average of the whole sample?

$$A' = \frac{k}{k+1} \cdot A + \frac{n_{k+1}}{k+1}.$$

On-policy vs. off-policy

- On-policy algorithms: track one "policy variable" π; the policy stored in π is used to interact with the environment (i.e., to sample episodes) and at the same time we learn something about it (e.g. its value vector).
    - Corresponds to the generalized policy iteration scheme.
    - All the MC algos we have seen so far.
- Off-policy algorithms: track more (typically two) different policy variables:
    - behavior policy: used to sample episodes
    - target policy: which we want to learn about

Off-policy evaluation

We are given effective representations of:

- a behavior policy β,
- a target policy π.

The task is to estimate v^π by sampling episodes from β . We cannot sample from π! (E.g. π too risky or expensive to sample from.)

Assumptions:

- given (s, a), we can effectively compute $\pi(a|s)$ and $\beta(a|s)$ (or at least estimate via sampling)
- coverage: $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$: if $\pi(a|s) > 0$, then also $\beta(a|s) > 0$

Importance sampling

Definition 36: Importance ratio

Let $\tau = s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \ldots$ be a trajectory. The importance-sampling ratio of τ is the quantity

$$\begin{aligned}\rho(\tau) &\overset{def}{=} \frac{\mathbb{P}^{\pi}[\tau \mid S_0 = s_0]}{\mathbb{P}^{\beta}[\tau \mid S_0 = s_0]} \\ &= \frac{\mathbb{P}^{\pi}[A_0 = a_0, S_1 = s_1, A_1 = a_1, \ldots, A_{T-1} = a_{T-1}, S_T = s_T \mid S_0 = s_0]}{\mathbb{P}^{\beta}[A_0 = a_0, S_1 = s_1, A_1 = a_1, \ldots, A_{T-1} = a_{T-1}, S_T = s_T \mid S_0 = s_0]}.\end{aligned}$$

$\rho(\tau)$ can be computed without the knowledge of MDP transition probabilities!

$$\rho(\tau) = \frac{\pi(a_0 \mid s_0) \cdot p(s_1 \mid s_0, a_0) \cdot \pi(a_1 \mid s_1) \cdot p(s_2 \mid s_1, a_1) \cdots}{\beta(a_0 \mid s_0) \cdot p(s_1 \mid s_0, a_0) \cdot \beta(a_1 \mid s_1) \cdot p(s_2 \mid s_1, a_1) \cdots}$$

Importance ratio from time t

Definition 37

Let $\tau = s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \ldots$ be a trajectory. By $\tau_{i..j}$ we denote the sub-trajectory of τ starting in time step i and ending in timestep j. By $\tau_{i..}$ we denote the suffix of $s_i, a_i, r_{i+1}, s_{i+1}, a_{i+1}, \ldots$.

Definition 38: Importance ratio from time t

Let $\tau = s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \ldots$ be a trajectory and t a time step. The importance-sampling ratio of τ from t is the quantity

$$\begin{aligned}\rho_t(\tau) &\overset{def}{=} \frac{\mathbb{P}^{\pi}[\tau_{t..} \mid S_0 = s_t]}{\mathbb{P}^{\beta}[\tau_{t..} \mid S_0 = s_t]}\\ &= \frac{\mathbb{P}^{\pi}[A_0 = a_t, S_1 = s_{t+1}, A_1 = a_{t+1}, \ldots, A_{T-1-t} = a_{T-1}, S_{T-t} = s_T \mid S_0 = s_t]}{\mathbb{P}^{\beta}[A_0 = a_t, S_1 = s_{t+1}, A_1 = a_{t+1}, \ldots, A_{T-1-t} = a_{T-1}, S_{T-t} = s_T \mid S_0 = s_t]}.\end{aligned}$$

Off-policy evaluation with importance sampling

Theorem 39

For any $s \in \mathcal{S}$ it holds:

$$\mathbb{E}^{\beta}[\rho \cdot G \mid S_0 = s] = v^{\pi}(s).$$

Proof:

$$\begin{aligned}\mathbb{E}^{\beta}[\rho \cdot G \mid S_0 = s] &= \sum_{\tau} \mathbb{P}^{\beta}[\tau \mid S_0 = s] \cdot \rho(\tau) \cdot G(\tau) \\ &= \sum_{\tau} \mathbb{P}^{\beta}[\tau \mid S_0 = s] \cdot \frac{\mathbb{P}^{\pi}[\tau \mid S_0 = s]}{\mathbb{P}^{\beta}[\tau \mid S_0 = s]} \cdot G(\tau) \\ &= \sum_{\tau} \mathbb{P}^{\pi}[\tau \mid S_0 = s] \cdot G(\tau) = \mathbb{E}^{\pi}[G \mid S_0 = s] = v^{\pi}(s).\end{aligned}$$

Easily integrates into both first-visit and every visit MC: sample from β and store $\rho_t(\tau) \cdot G_t(\tau)$ in $Ret(s_t)$.

Weighted importance sampling

First-visit variant: for each state s, we keep a set of samples $Sam(s)$. Each sample is a tuple (τ, t) – trajectory and time step.

- initially, $Sam(s) = \emptyset$ for all s
- we then sample trajectories until timeout:
    - for each sampled trajectory τ and each state s, and the smallest t such that $S_t(\tau) = s$ we add (τ, t) to $Sam(s)$

Throughout the algorithm, the value of state s is estimated as

$$WIS(s) = \frac{\sum\limits_{(\tau,t)\in Sam(s)} \rho_t(\tau) \cdot G_t(\tau)}{\sum\limits_{(\tau,t)\in Sam(s)} \rho_t(\tau)}$$

Exercise 40

Compare ordinary/weighted importance sampling after single sample.

Weighted importance sampling – correctness

The weighted sampling is clearly a biased estimator. However, the bias vanishes in the limit:

Theorem 41

With probability 1: as $|Sam(s)| \to \infty$, we have that $WIS(s) \to v^{\pi}(s)$.

Proof:

Ordinary vs. weighted sampling

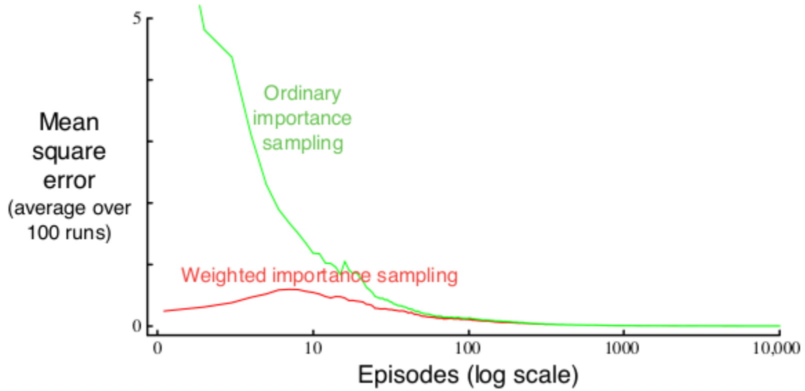


Figure 5.3: Weighted importance sampling produces lower error estimates of the value of a single blackjack state from off-policy episodes. ■

source: Sutton&Barto, p. 106

Importance sampling: summary

But ordinary and weighted importance sampling can be adapted to every-visit MC.

Bias & Convergence:

- First visit:
    - ordinary IS: unbiased, i.e. also converges
    - weighted IS: biased, but converges in the limit
- Every visit:
    - both ordinary and weighted: biased (due to EV), but converges in the limit

Weighted IS: incremental implementation

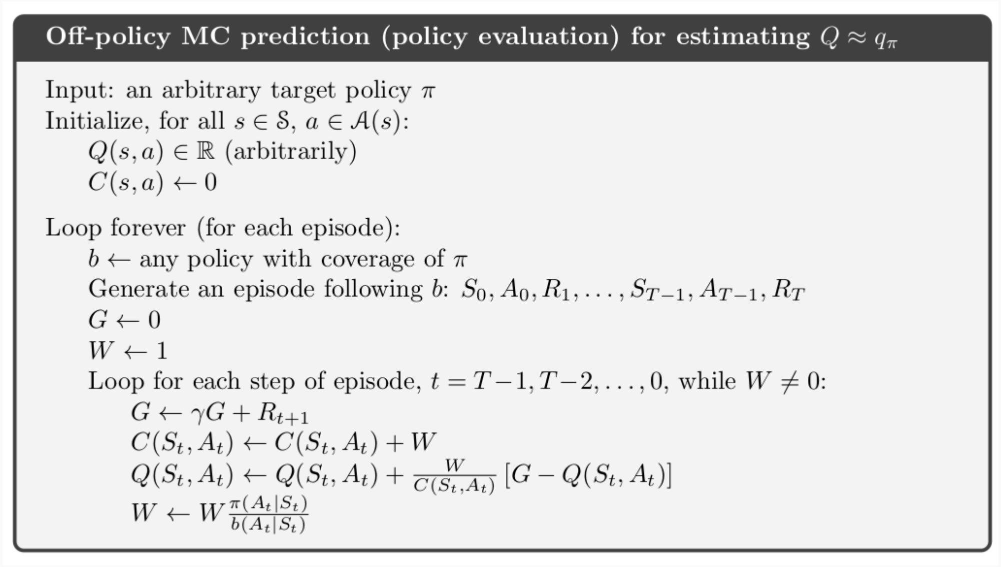
Instead of recomputing the weighted average for each new sample, $WIS(s)$ can be updated by keeping keep just two variables:

- V – current value of $WIS(s)$, initially arbitrary
- C – the sum of importance ratios, initially 0

Upon arrival of new sample (τ', t'), we update V, C into new values V', C' by setting:

$$C' = C + \rho_{t'}(\tau')$$
$$V' = V + \frac{\rho_{t'}(\tau')}{C'} \cdot (G_{t'}(\tau') - V)\,.$$

Off-policy evaluation with weighted IS



Off-policy MC prediction (policy evaluation) for estimating $Q \approx q_\pi$

```
Input: an arbitrary target policy $\pi$
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \in \mathbb{R}$ (arbitrarily)
    $C(s,a) \leftarrow 0$

Loop forever (for each episode):
    $b \leftarrow$ any policy with coverage of $\pi$
    Generate an episode following $b$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    $W \leftarrow 1$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$, while $W \neq 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
        $W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$
```

Off-policy control with weighted IS

Required reading: Sutton&Barto, Section 5.7.

Temporal Difference Methods

TD: Motivation

Let us first focus on policy evaluation.

MC: zero bias (at least in the limit), but potentially high variance: many samples needed to converge. Also, to update estimates, it must wait till the end of each episode.

TD methods retain the focus on sampling but combine it with bootstrapping.

Incremental update notation

Definition 42: Notation for updates

In the context of RL algorithms will denote by $V^n(s)$ (resp. $Q^n(s, a)$) the algorithm's estimate of $v^\pi(s)$ (resp. $q^\pi(s, a)$) after n-th update of this estimate.

MC vs. TD(0) update

On-policy MC (incremental) update using sampled trajectory $\tau = s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \ldots$:

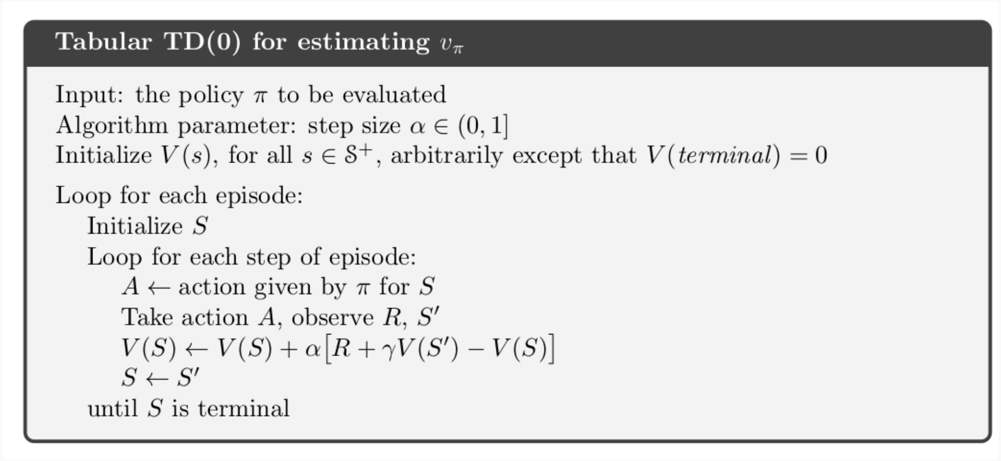
$$V^{n+1}(s_t) \leftarrow (1 - \alpha_n)V^n(s_t) + \alpha_n G_t(\tau) = V^n(s_t) + \alpha_n \cdot \Big[\underbrace{\underbrace{G_t(\tau)}_{\text{update target}} - V^n(s_t)}_{\text{update error}} \Big],$$

where $\alpha_n = n/(n+1)$.

TD(0) update in the same situation, with α_n "suitably chosen" (possibly constant):

$$V^{n+1}(s_t) \leftarrow V^n(s_t) + \alpha_n \cdot \big[R_{t+1}(\tau) + \gamma \cdot \underbrace{V^n(S_{t+1}(\tau))}_{\text{bootstrap}} - V^n(s_t)\big]$$

Policy evaluation with TD(0)



Tabular TD(0) for estimating v_π

```
Input: the policy $\pi$ to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\textit{terminal}) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$, observe $R$, $S'$
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

source: Sutton&Barto, p. 120

How can it even work?

Really "just" a very asynchronous, sample-based, and "α-dampened" version of value iteration.

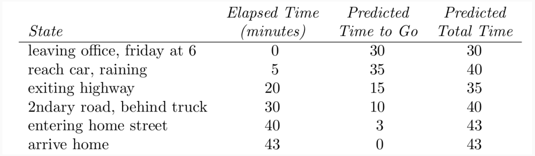
$$\mathbb{E}^\pi[G_t | S_t = s] = \mathbb{E}^\pi[R_{t+1} + \gamma \cdot G_{t+1} \mid S_t = s] = \mathbb{E}^\pi[R_{t+1} \mid S_t = s] + \gamma \cdot \underbrace{\mathbb{E}^\pi[G_{t+1} \mid S_t = s]}_{v^\pi(S_{t+1})}.$$

In expectation, the TD(0) update is the same as VI update in $\mathcal{M}^\pi$. Thanks to the contractivity of the Bellman operator, VI possesses an error reduction property: after each update, the error of the estimate decreases. Hence, in expectation, the same is true for the TD(0) update.

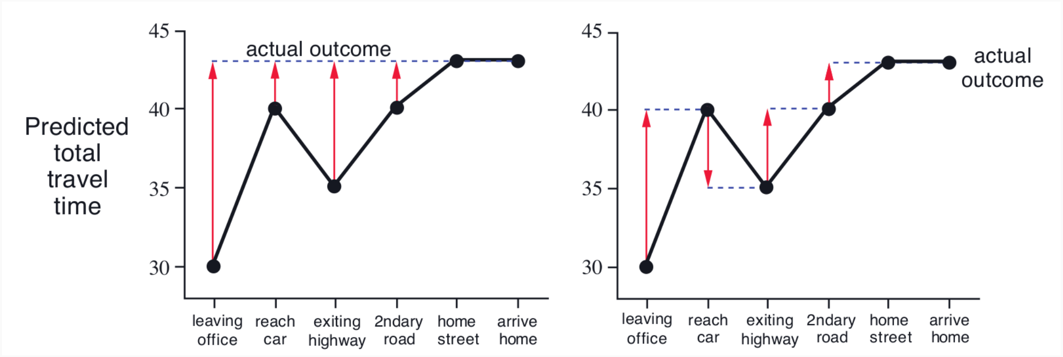
Formal proof of correctness in optional reading:

Sutton, R.S.: Learning to Predict by Methods of Temporal Differences. In *Machine Learning* 3:9–44. Kluwer, 1988. (For MDPs with function approximation.)

Why TD is natural (Sutton&Barto, p. 122-123)



State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43



Left: MC. Right TD(0).

On-policy TD control

Recall:

- In control setting, we need to estimate q-values of a policy.
- On-policy: we sample trajectories according to some policy π and then push value estimates towards q^π.

To maintain exploration, the policy π will typically be the ε-Q-greedy policy for some $\varepsilon > 0$, where Q are the current Q values estimates. I.e., throughout the algorithm

$$\pi(a|s) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{\mathcal{A}(s)} & \text{if } a = \arg\max_{a' \in \mathcal{A}(s)} Q(s, a') \\ \frac{\varepsilon}{\mathcal{A}(s)} & \text{otherwise.} \end{cases}$$

SARSA

State-Action-Reward-State-Action. Introduced in Rummery, Niranjan: *On-Line Q-Learning Using Connectionist Systems* (1994).

In each episode, sample a trajectory $\tau = s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \ldots, s_T$ according to current policy π; for each time step $0 \leq t \leq T-1$, perform the following update:

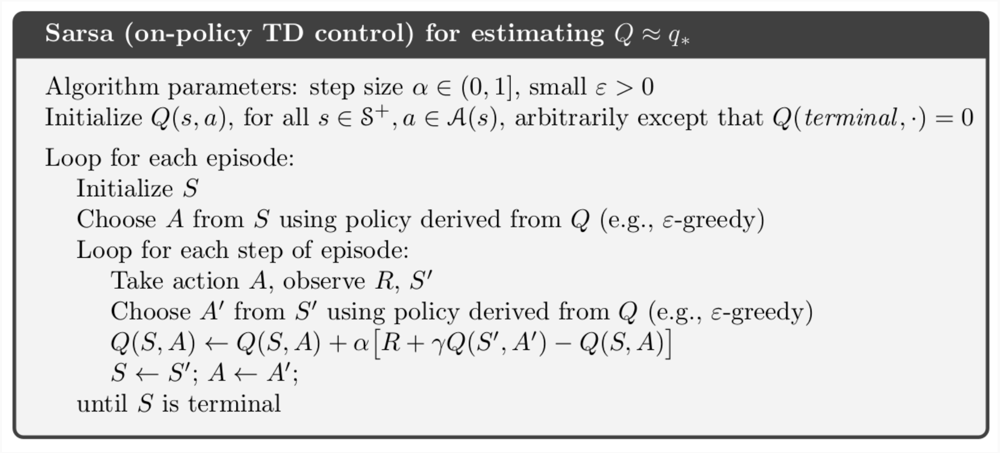
$$Q^{n+1}(s_t, a_t) = Q^n(s_t, a_t) + \alpha_n \cdot \left[r_{t+1} + \gamma Q^n(s_{t+1}, a_{t+1}) - Q^n(s_t, a_t) \right]$$

The update can be performed immediately when s_{t+1} and a_{t+1} is known (no need to wait for the episode to terminate).

After the episode ends, make π ε-Q-greedy.

Conforms to the GVI scheme.

SARSA pseudocode



Sarsa (on-policy TD control) for estimating $Q \approx q_*$

```
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Loop for each step of episode:
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until $S$ is terminal
```

source: Sutton&Barto, p. 130

GLIE policies

Definition 43: GLIE condition

A RL algorithm is greedy in the limit (GL) if its behavior policy (=target policy in on-policy algorithms) converges to a 0-greedy policy with increasing number of episodes.
A RL algorithm is GLIE if it is GL and IE (infinitely exploring).

Typical ways of ensuring GLIE:

- Dynamically adjust ε in ε-greedy policy selection" When selecting action in state s, behave ε-greedily with $\varepsilon = \frac{c}{n(s)}$, where $0 < c < 1$ is a constant and $n(s)$ is a number of visits to state s over all the episodes so far.
- Use Boltzmann (softmax) exploration:

$$\pi(a \mid s) = \frac{e^{\frac{Q(s,a)}{\eta(s)}}}{\sum_{b \in \mathcal{A}(s)} e^{\frac{Q(s,b)}{\eta(s)}}},$$

where η is a state-dependent and time-varying temperature parameter. We need η to converge to 0 over time, but not too fast (often, $\eta(s)$ proportional to $\frac{1}{\log(n(s))}$).

Convergence of SARSA

Theorem 44

Consider a GLIE instantiation of SARSA. Moreover, assume that the sequence of learning rates $(\alpha_n)_{n\in\mathbb{N}}$ satisfies $\sum_n \alpha_n = \infty$ and $\sum_n \alpha_n^2 < \infty$. In this setting, Q converges to q^* and the behavior policy of SARSA converges to some optimal policy π^*.

For the proof, see optional reading: Singh, Jaakkola, Littman, Szepesvári: Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms. In *Machine Learning* 39:287-308. Kluwer, 2000.

Note: learning rate can itself be state/action dependent (omitted for conciseness, constant learning rates preferred in practice).

Off-policy TD control

Surprise: no importance sampling!

Recall the SARSA update:

$$Q^{n+1}(s_t, a_t) = Q^n(s_t, a_t) + \alpha_n \cdot \left[r_{t+1} + \gamma Q^n(s_{t+1}, a_{t+1}) - Q^n(s_t, a_t) \right]$$

It pushes Q towards q^π, where π is the current policy.

Idea: push Q directly towards q^*.

We could do this e.g. by a VI-like update:

$$Q^{n+1}(s_t, a_t) = Q^n(s_t, a_t) + \alpha_n \cdot \left[\max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} p(s' \mid s, a)\big(r(s, a, s') + \gamma V(s')\big) - Q^n(s_t, a_t) \right].$$

Two problems:

- We do not calculate v-estimates. (Must somehow replace with Q)
- We must get rid of transition probabilities and instead use the sampled a_t and r_{t+1}.

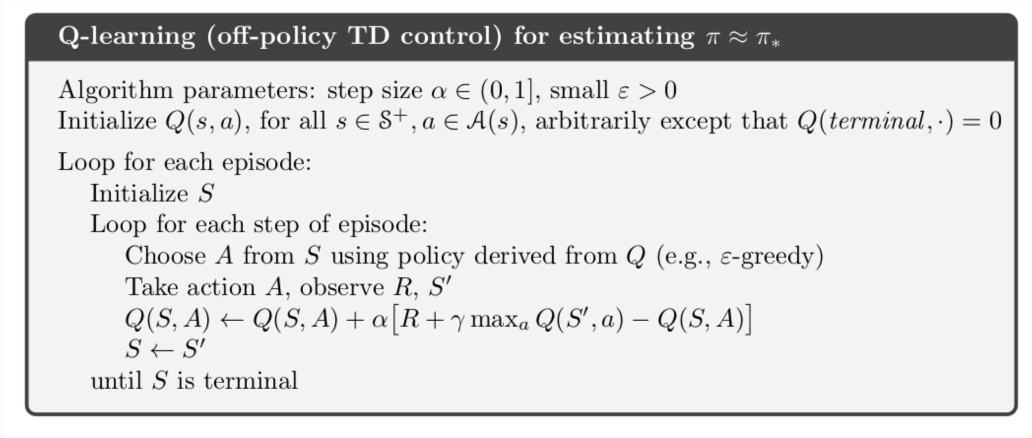
Q-learning update

Solution: push the max towards the bootstrap.

Q-learning (Watkins, 1989): given a sampled trajectory $s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \ldots$, for every t we update:

$$Q^{n+1}(s_t, a_t) = Q^n(s_t, a_t) + \alpha_n \cdot \left[r_{t+1} + \gamma \cdot \Big(\max_{a \in \mathcal{A}(s_{t+\mathbf{1}})} Q^n(s_{t+1}, a) \Big) - Q^n(s_t, a_t) \right]$$

Q-learning pseudocode



Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

```
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

source: Sutton&Bato, p. 131

Off-policy: where is the second policy?

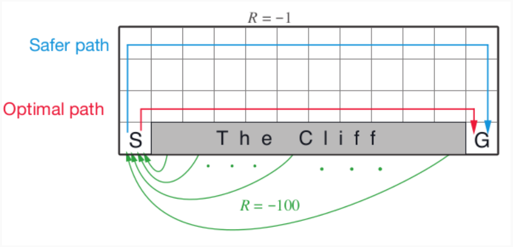
Q-learning convergence

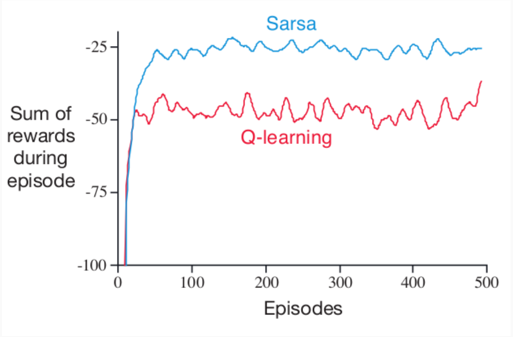
Theorem 45

Consider any Q-learning instantiation with infinite exploration. Assume that the sequence of learning rates $(\alpha_n)_{n\in\mathbb{N}}$ satisfies $\sum_n \alpha_n = \infty$ and $\sum_n \alpha_n^2 < \infty$. In this setting, Q converges to q^*. Moreover, if the behavior policy is GL, then it converges to an optimal policy π^*.

Proof in optional reading: Watkins, Dayan: Q-Learning. In *Machine Learning* 8:279-292. Kluwer, 1992.

SARSA vs. Q-learning (SB: p. 132)





Left: greedy policies learned by SARSA and Q-learning.

Right: in-training performance with a 0.1-greedy behavior policy.

(Rough) takeaway: Q-learning more aggressive in finding optimal policy, can lead to risky behavior. Possibly advantageous when environment not too stochastic or if in-training performance has less importance (simulator vs. real world).

Maximization bias in Q-learning

Q-learning is “risky” not only due to exploration, but also because it is optimistic in the face of uncertainty. TBD

The positive bias only disappears in the limit.

Double Q-learning

Idea: use two independent value estimates Q_1, Q_2 and decouple action selection from evaluation in the bootstrap. During each update, we randomly select one of these for update, which is also used to select the maximizing action in bootstrap. The other is used as the bootstrap estimate.

I.e., in each time step t we perform one of these updates, each with probability $\frac{1}{2}$: either

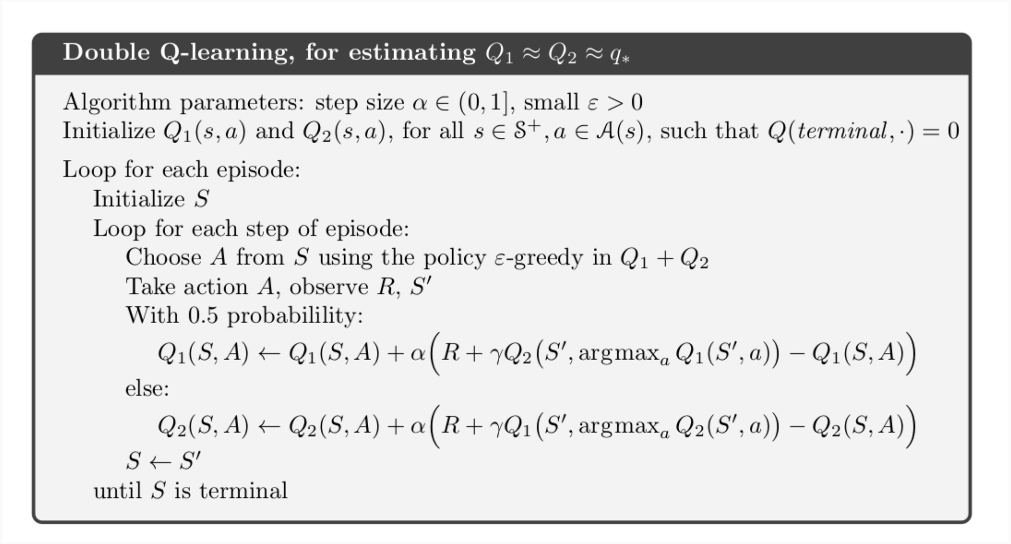
$$Q_1(s_t, a_t) = Q_1(s_t, a_t) + \alpha_n \cdot \left[r_{t+1} + \gamma \cdot Q_2\left(s_{t+1}, (\max_{a \in \mathcal{A}(s_{t+1})} Q_1(s_{t+1}, a))\right) - Q_1(s_t, a_t)\right]$$

or

$$Q_2(s_t, a_t) = Q_2(s_t, a_t) + \alpha_n \cdot \left[r_{t+1} + \gamma \cdot Q_1\left(s_{t+1}, (\max_{a \in \mathcal{A}(s_{t+1})} Q_2(s_{t+1}, a))\right) - Q_2(s_t, a_t)\right].$$

Behavior policy = e.g. ε-greedy w.r.t. $Q_1 + Q_2$.

Double Q-learning pseudocode



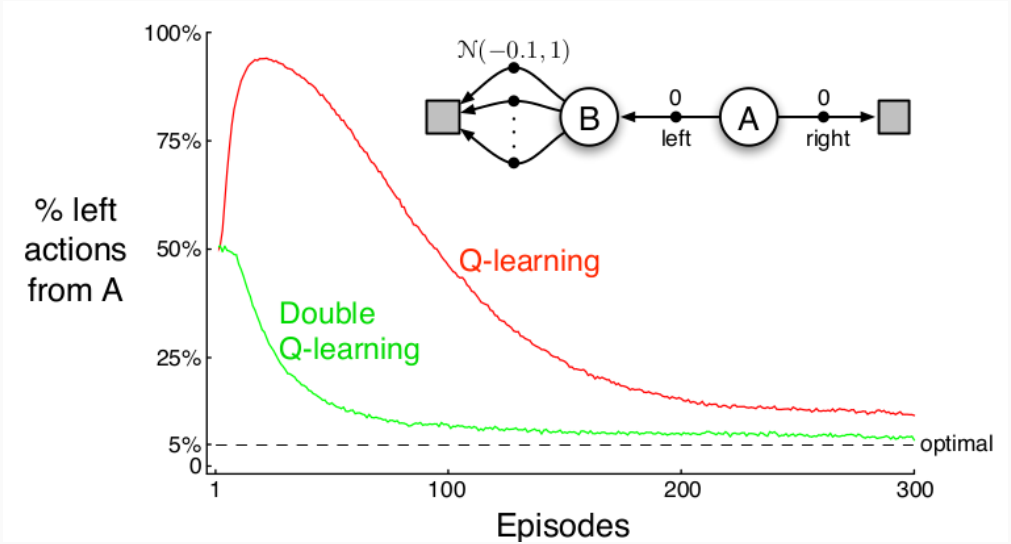
```
Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q_1(s,a)$ and $Q_2(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A$ from $S$ using the policy $\varepsilon$-greedy in $Q_1 + Q_2$
        Take action $A$, observe $R$, $S'$
        With 0.5 probabilility:
            $Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \operatorname{argmax}_a Q_1(S',a)\big) - Q_1(S,A)\Big)$
        else:
            $Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \operatorname{argmax}_a Q_2(S',a)\big) - Q_2(S,A)\Big)$
        $S \leftarrow S'$
    until $S$ is terminal
```

source: Sutton&Barto, p. 136

Why Double Q-learning helps

TBD

Double Q-learning: experiment

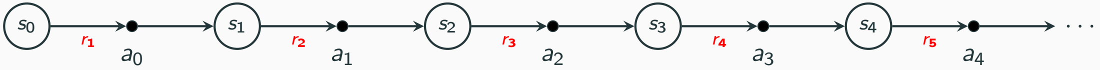


source: Sutton&Barto, p. 135

Between Monte Carlo and TD: n-Step and λ-Returns

MC vs TD(0) update targets

Given a trajectory $s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \ldots$:



Update for time step t in:

- MC = discounted return from t till the end of trajectory, e.g. for $t = 1$:

$$r_2 + \gamma r_3 + \gamma^2 r_4 + \cdots + \gamma^{T-2} r_T$$

  unbiased, but high variance + need the whole trajectory
- TD(0) = 1-step reward and then (discounted) bootstrap:

$$r_2 + \gamma V(s_2)$$

n-step return

Idea: use n-step discounted return and then bootstrap

Definition 46

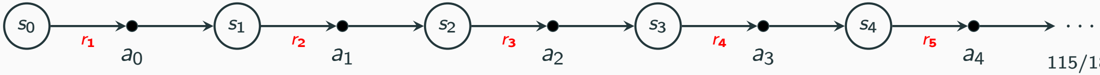
Let $\tau = s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \ldots$ be a trajectory and $n \in \mathbb{N} \setminus \{0\}$.
An *n-step return* of τ from time step t is the quantity

$$G_{t:t+n}(\tau) = r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + \cdots \gamma^{n-1} r_{t+n} + \gamma^n \cdot V(s_{t+n}).$$

We also define a Q-estimate-based version:

$$G_{t:t+n}(\tau) = r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + \cdots \gamma^{n-1} r_{t+n} + \gamma^n \cdot Q(s_{t+n}, a_{t+n}).$$

(Which of the two is used will be clear from the context.)



n-step TD policy evaluation

Similar to TD(0), but using *n*-step return targets:

given a trajectory $\tau = s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \ldots$, for each $0 \leq t < T$ we perform an update

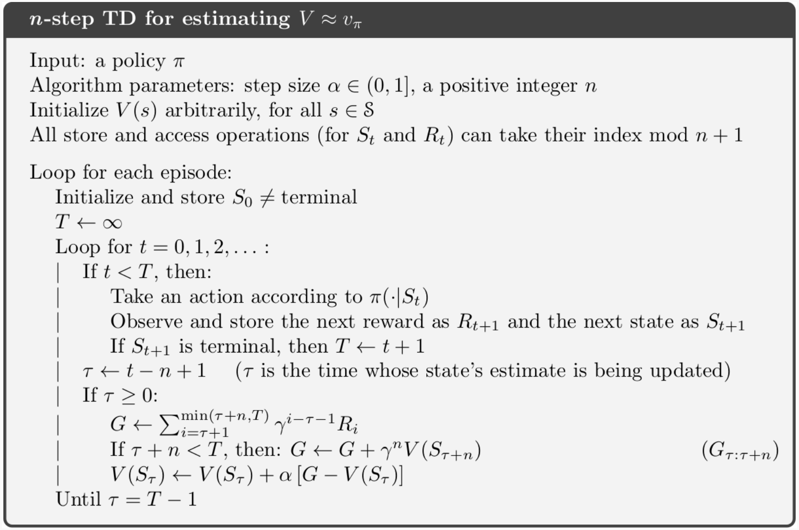
$$V(s_t) \leftarrow V(s_t) + \alpha[G_{t:t+n}(\tau) - V(s_t)].$$

Note 1: for $n = 1$ we get exactly TD(0).

Note 2: for $n > 1$, we cannot update $V(s_t)$ directly at step $t + 1$. We need to obtain $r_{t+1}, \ldots, r_{t+n}, s_{t+n}$ first, i.e. we can perform the update after step $t + n$.

Note 3: if $t + n > T$, we truncate the sum in $G_{t:t+n}$ at r_T, i.e. in such a case $G_{t:t+n} = G_t$.

n-step TD policy evaluation: pseudocode



```
n-step TD for estimating $V \approx v_\pi$

Input: a policy $\pi$
Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer $n$
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
All store and access operations (for $S_t$ and $R_t$) can take their index mod $n + 1$

Loop for each episode:
    Initialize and store $S_0 \neq$ terminal
    $T \leftarrow \infty$
    Loop for $t = 0, 1, 2, \ldots$ :
    |   If $t < T$, then:
    |       Take an action according to $\pi(\cdot|S_t)$
    |       Observe and store the next reward as $R_{t+1}$ and the next state as $S_{t+1}$
    |       If $S_{t+1}$ is terminal, then $T \leftarrow t + 1$
    |   $\tau \leftarrow t - n + 1$    ($\tau$ is the time whose state's estimate is being updated)
    |   If $\tau \geq 0$:
    |       $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
    |       If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$          $(G_{\tau:\tau+n})$
    |       $V(S_\tau) \leftarrow V(S_\tau) + \alpha [G - V(S_\tau)]$
    Until $\tau = T - 1$
```

source: Sutton&Barto, p.144

n-step TD policy evaluation: performance

19-state symmetric random walk:

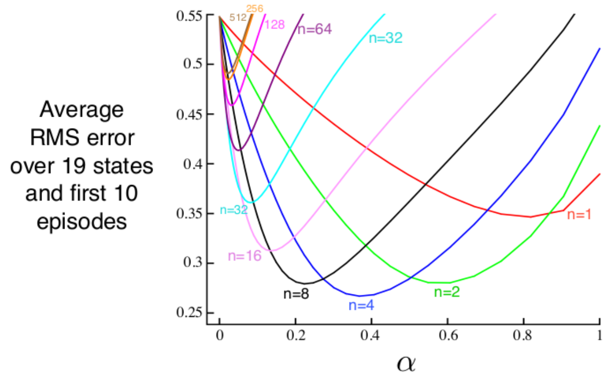


Figure 7.2: Performance of n-step TD methods as a function of α, for various values of n, on a 19-state random walk task (Example 7.1). ■

source: Sutton&Barto, p.145

n-step SARSA (on-policy control)

Uses Q-value-bootstrapped *n*-step returns.

For a sampled trajectory $\tau = s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \ldots$ and for all time steps $0 \leq t < T$ we perform an update

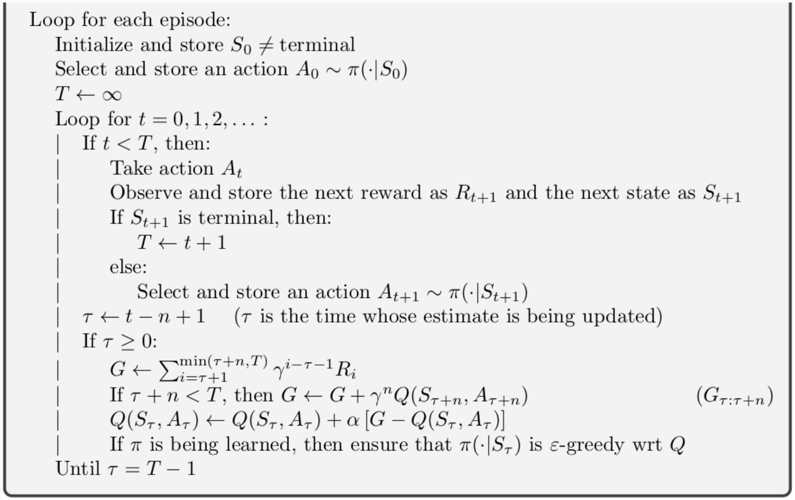
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[G_{t:t+n} - Q(s_t, a_t)].$$

We sample trajectories according to a policy π that is ε-greedy w.r.t. current Q-estimates:

$$\pi(a|s) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}(s)|} & \text{if } a = \arg\max_{a' \in \mathcal{A}(s)} Q(s, a') \text{ (ties broken in principled way)} \\ \frac{\varepsilon}{|\mathcal{A}(s)|} & \text{otherwise.} \end{cases}$$

π is redefined in this way after each episode

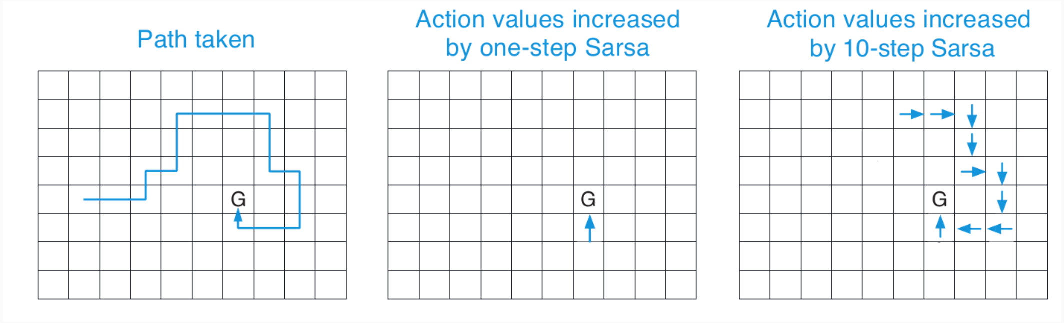
n-step SARSA (pseudocode)



```
Loop for each episode:
    Initialize and store $S_0 \neq$ terminal
    Select and store an action $A_0 \sim \pi(\cdot|S_0)$
    $T \leftarrow \infty$
    Loop for $t = 0, 1, 2, \dots$ :
    |   If $t < T$, then:
    |       Take action $A_t$
    |       Observe and store the next reward as $R_{t+1}$ and the next state as $S_{t+1}$
    |       If $S_{t+1}$ is terminal, then:
    |           $T \leftarrow t + 1$
    |       else:
    |           Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
    |   $\tau \leftarrow t - n + 1$    ($\tau$ is the time whose estimate is being updated)
    |   If $\tau \geq 0$:
    |       $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
    |       If $\tau + n < T$, then $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$          $(G_{\tau:\tau+n})$
    |       $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha [G - Q(S_\tau, A_\tau)]$
    |       If $\pi$ is being learned, then ensure that $\pi(\cdot|S_\tau)$ is $\varepsilon$-greedy wrt $Q$
    Until $\tau = T - 1$
```

source: Sutton&Barto, p. 147

n-step SARSA (speed of signal propagation)


Path taken
Action values increased by one-step Sarsa
Action values increased by 10-step Sarsa
G
G
G

source: Sutton&Barto, p. 147

n-step Q-learning

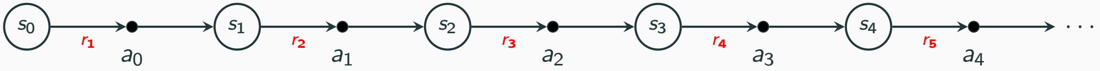
The Q-learning update for a trajectory $\tau = s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \ldots$ and time step t:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot [r_{t+1} + \gamma \cdot \max_{a \in \mathcal{A}(s_{t+1})} Q(s_{t+1}, a) - Q(s_t, a_t)].$$

Naive extension to n-step returns

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot [r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{t+n-1} \cdot r_{t+n} + \gamma^{t+n} \cdot \max_{a \in \mathcal{A}(s_{t+1})} Q(s_{t+n+1}, a) - Q(s_t, a_t)]$$

does not really correspond to Q-learning, since some of the actions $a_{t+1}, \ldots, a_{t+n}$ might not be Q-greedy (the behavior policy is ε-greedy, so some actions might be exploratory). Hence, we are no longer pushing Q towards the Q-value of an optimal policy.



n-step Q-learning: correct

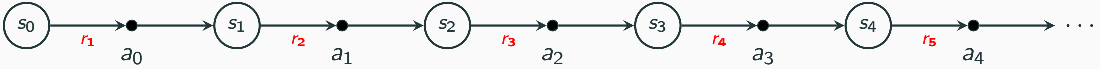
Idea: apply the Q-learning bootstrap at the first occurrence of a non-Q-greedy action.

I.e., for each episode:

- make π an ε-Q-greedy policy
- sample a trajectory $\tau = s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \ldots$ from π
- for each time step $0 \leq t < T$:
    - identify the smallest $n' \in \{t+1, t+2, \ldots, t+n\}$ such that $a_{n'}$ is not a Q-greedy action
    - perform the update

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot [r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n'-1} r_{n'} + \gamma^{n'} \cdot \max_{a \in \mathcal{A}(s_{n'})} Q(s_{n'}, a) - Q(s_t, a_t)]$$

(if $n' > T$, do the standard MC update).



λ-returns: idea

By varying n, the n-step returns provide a nice tradeoff between bias and variance (and update speed). But the choice of optimal n is mostly a guesswork.

Idea: find a notion of return which combines n-step returns for multiple n's. E.g., a suitable convex combination of individual n-step returns. This leads to the notion of λ-returns.

We will focus only on policy evaluation, though λ-returns can be used also in control.

λ-returns: definition

Recall: $G_{t:t+n}$ is the n-step return from timestep t.

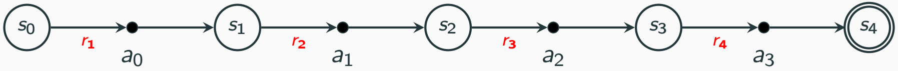
Definition 47: λ-return

Let $\lambda \in [0,1]$. A λ-return from timestep t is the random variable

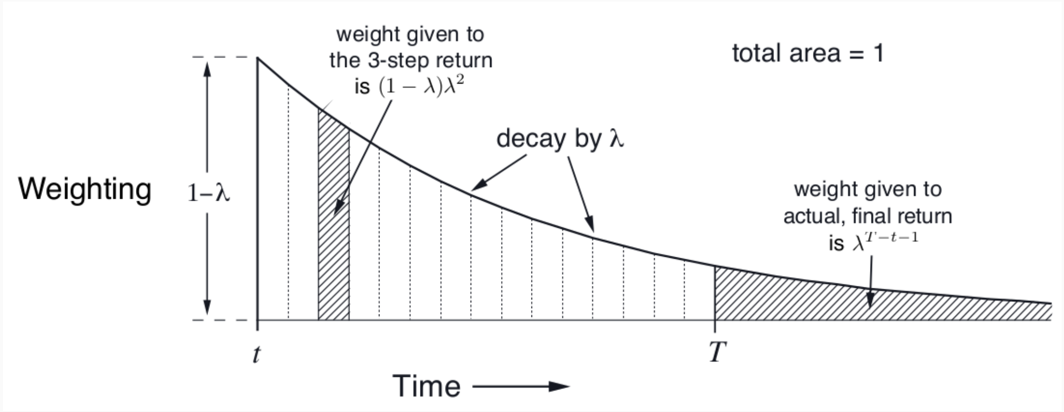
$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n}.$$

Note that due to truncation at $t+n \geq T$, the λ-return can be more explicitly written as

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} \cdot G_{t:t+n} + \lambda^{T-t-1} \cdot G_t.$$



λ-return as discounting of n-step returns


weight given to the 3-step return is $(1-\lambda)\lambda^2$
total area = 1
decay by λ
Weighting
$1-\lambda$
weight given to actual, final return is λ^{T-t-1}
t
T
Time

source: Sutton&Barto, p. 290

(Forward-view) TD(λ)

Like TD(0), but uses λ-returns.

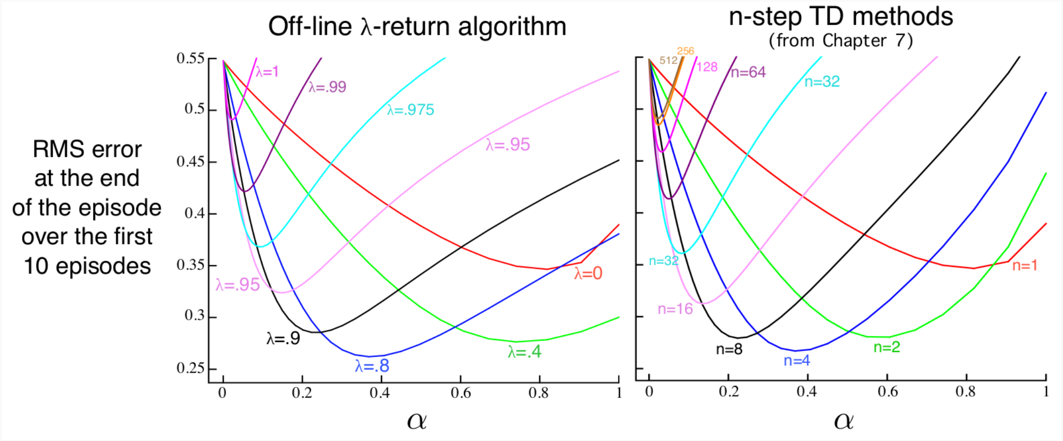
Given a trajectory $\tau = s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \ldots$ sampled from the evaluated policy π, we perform, for each time step t an update:

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t^\lambda(\tau) - V(s_t)).$$

Note:

- for $\lambda = 0$, this is exactly the TD(0) update,
- for $\lambda = 1$, this is exactly the MC update,
- $G_t^\lambda(\tau)$ depends on the whole suffix of $\tau_{t..}$, hence the update can be only performed at the end of the episode. (We will show a workaround later.)

TD(λ) vs. n-step TD on 19-state random walk



source: Sutton&Barto, p. 291

Forward vs. backward view TD(λ)

Backward-view TD(λ) $=$ an algorithm performing roughly the same updates as Forward-view TD(λ) in an online fashion ($V(s_t)$ can be updated by time $t+1$).

Implemented using eligibility traces: state-wise signals that indicate how much is the current state eligible for an update (sort of state-wise modulation of the learning rate).

We are more keen to update states that:

- appear often along the trajectory (frequency heuristic)
- were visited in the recent past (recency heuristic)

Accumulating eligibility trace

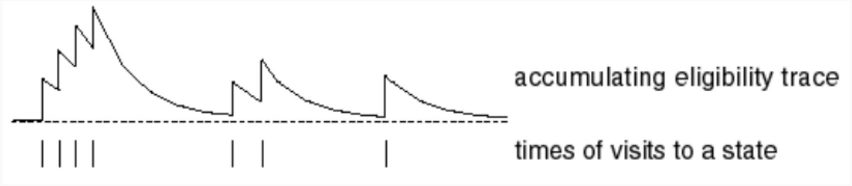
Definition 48: (Accumulating) eligibility trace

For a trajectory $\tau = s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \ldots$, $\lambda \in [0,1]$, and a state $s \in \mathcal{S}$, an (accumulating) eligibility trace is a sequence of values $E_0(s), E_1(s), E_2(s), \ldots$ defined inductively as follows:

$$E_0(s) = 0$$

$$\text{and for } t > 0 \qquad E_t(s) = \gamma \cdot \lambda \cdot E_{t-1}(s) + \mathbb{I}(S_t(\tau) = s),$$

where $\mathbb{I}(S_t(\tau) = s)$ is the indicator of the t-th state of τ being s, i.e. $\mathbb{I}(S_t(\tau) = s) = 1$ if $s_t = s$ and $\mathbb{I}(S_t(\tau) = s) = 0$ otherwise.



source: slides of D. Silver (Model-free prediction)

Backward-view TD(λ): idea

- $E_t(s)$ denotes how much is s eligible for an update after playing t-th action along the run (i.e., action a_{t-1}).
- In time step t, all states with non-zero eligibility signal will have their estimates updated in proportion to the learning rate and the strength of the eligibility signal.
- The update target is the standard TD(0) target for time t. I.e., for each timestep t and each state q, we perform the update

$$V(q) \leftarrow V(q) + \alpha \cdot E_t(q) \cdot \left[r_{t+1} + \gamma \cdot V(s_{t+1}) - V(s_t)\right].$$

Backward-view TD(λ): pseudocode

```
Input: policy $\pi$ to evaluate
Output: Estimate $V$ of $v^\pi$
initialize $V$ arbitrarily
repeat
    $s \leftarrow$ sample uniformly (ES) or according to init. distr.
    initialize $E$ to be uniformly zero
    while $s$ not terminal do
        $a \leftarrow$ sample from $\pi(s)$
        $s' \leftarrow$ sample from $p(s, a)$
        $r \leftarrow r(s, a, s')$
        foreach $q \in \mathcal{S}$ (Only $q$'s visited so far) do
            $E(q) = \gamma \cdot \lambda \cdot E(q) + \mathbb{I}(q = s)$
            $V(q) \leftarrow V(q) + \alpha \cdot E(q) \cdot \left[r + \gamma \cdot V(s') - V(s)\right]$
        $s \leftarrow s'$
until timeout
```

Forward vs. backward view

If $\lambda = 0$, then $E_t(q) = \mathbb{I}(S_t(\tau) = q)$, i.e. the backward-view update at time point t is

$$V(s_t) \leftarrow V(s_t) + \alpha \cdot \left[r + \gamma \cdot V(s_{t+1}) - V(s_t) \right],$$

while for all states other than s_t, no update is performed. I.e., backward TD(0) is exactly the same thing as forward TD(0).

For general λ the correspondence is more subtle:

Theorem 49: Forward-backward view correspondence

Assume that in the backward view, all the updates along the trajectory are performed offline, i.e. only after the end of the episode, and in a batch, i.e. concurrently, using the pre-episode estimates in right-hand sides.
Then, for any $\lambda \in (0, 1)$, this offline backward TD(λ) performs the same updates as forward TD(λ).

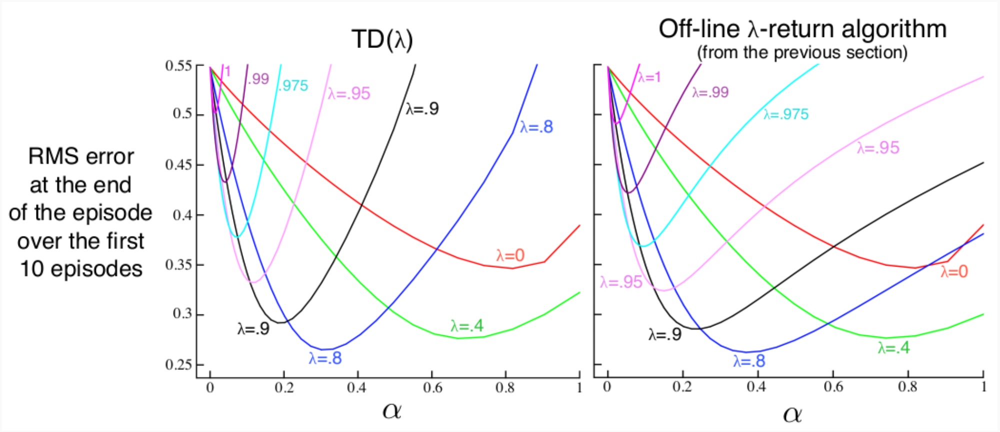
Offline backward and forward view (source: D. Silver slides)

Given the batch nature of updates, it suffices to show that the forward update target at time t equals the sum of all updates "triggered" by a visit to state s_t.

$$
\begin{aligned}
G_t^\lambda - V(S_t) = -V(S_t) \quad & + \quad (1-\lambda)\lambda^0 \left(R_{t+1} + \gamma V(S_{t+1})\right) \\
& + \quad (1-\lambda)\lambda^1 \left(R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})\right) \\
& + \quad (1-\lambda)\lambda^2 \left(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 V(S_{t+3})\right) \\
& + \quad \ldots \\
= -V(S_t) \quad & + \quad (\gamma\lambda)^0 \left(R_{t+1} + \gamma V(S_{t+1}) - \gamma\lambda V(S_{t+1})\right) \\
& + \quad (\gamma\lambda)^1 \left(R_{t+2} + \gamma V(S_{t+2}) - \gamma\lambda V(S_{t+2})\right) \\
& + \quad (\gamma\lambda)^2 \left(R_{t+3} + \gamma V(S_{t+3}) - \gamma\lambda V(S_{t+3})\right) \\
& + \quad \ldots \\
= \quad & \quad\quad (\gamma\lambda)^0 \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right) \\
& + \quad (\gamma\lambda)^1 \left(R_{t+2} + \gamma V(S_{t+2}) - V(S_{t+1})\right) \\
& + \quad (\gamma\lambda)^2 \left(R_{t+3} + \gamma V(S_{t+3}) - V(S_{t+2})\right) \\
& + \quad \ldots \\
= \delta_t + \gamma\lambda\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \ldots &
\end{aligned}
$$

Online vs. offline backward view on 19-state RW

In practice, we want to use the online backward algorithm, which only approximates the forward view. Nevertheless, it performs acceptably:



source: Sutton&Barto, p. 295, LEFT: online backward TD(λ), RIGHT: offline forward TD(λ)

Concluding remarks on λ-returns

- There are other types of eligibility traces (replacing, dutch, ...), yielding different algorithms.
- Eligibility traces neatly generalize to deep learning, where they are not state-wise but parameter-wise signals; optional reading: Sutton&Barto, Sec. 12.1-12.2
- λ-returns and eligibility traces can be generalized to control setting SARSA(λ), Q(λ); optional reading: Sutton&Barto, Sec. 12.7-12.10.
- There is a true online backward TD(λ) version. Here, true online=having perfect equivalence with the forward view. However, the equivalence is w.r.t. a more complex notion of λ-return (truncated λ-return) and uses more complex version of eligibility traces than presented here. Outperforms both forward and backward algorithms presented here. Optional reading: van Seijen, Sutton: *True Online TD(λ)*. In *Proceedings of ICML'14*.

First Steps Towards Deep RL: Value-Based On-Policy Methods

Working with huge MDPs

E.g. original Atari games have 160x192 resolution with 128 colors: observable state space of size $2^{7\cdot 160\cdot 192} = 2^{215040}$ (though only a fraction reachable and resolution typically scaled down in benchmarks – however, state typically encompass last 3 frames so as to provide some info on movement).

State space can be even continuous (position, velocity,...).

Most states will not be seen - we need the ability to generalize from experience to unseen/rarely seen states.

Huge MDP representation

From now on, states of the MDP will be represented by vectors from $\mathbb{R}^n$. The vectorized representation is chosen in a domain-specific manner, e.g.:

- Atari = one component per pixel per frame
- continuous navigation = agent coordinates, velocity, etc.
- small discrete MDPs can be represented by one-hot encoding

For simplicity, we will still assume that the action space is discrete, and reasonably small, though many algorithms can be adapted for continuous actions (acceleration, etc).

Function approximators

The value functions have types:

$$v^\pi, v^* : \mathbb{R}^n \to \mathbb{R} \qquad q^\pi, q^* : \mathbb{R}^n \times \mathcal{A} \to \mathbb{R}.$$

In RL, we need to approximate these functions.

Definition 50

A function approximator (FA) for functions of type $X \to Y$ is a class of functions $f \subseteq Y^X$ parameterized by a some set of parameter vectors $\Theta \subseteq \mathbb{R}^n$.

Each concrete parameter vector $\theta \in \Theta$ defines a concrete function $f_\theta \in f$, i.e. $f = \{f_\theta \mid \theta \in \Theta\}$.

For FA f, we often write $f_\theta(x) = f(x, \theta)$ to stress the fact that the output of f_θ depends on both the input x and on θ. Hence, FA for type $X \to y$ can be itself seen as a function of type $X \times \Theta \to Y$.

Function approximators in RL

Our algorithms will use mainly these types of function approximators:

- $V\colon \mathbb{R}^n \times \Theta \to \mathbb{R}$ to approximate v^π or v^θ
- $Q\colon (\mathbb{R}^n \times \mathcal{A}) \times \Theta \to \mathbb{R}$ to approximate q^π or q^θ

The typical task is to find $\theta \in \Theta$ such that $V_\theta = V(\cdot, \theta)$ is a “good” approximation for v^π or v^θ, and similarly for Q_θ.

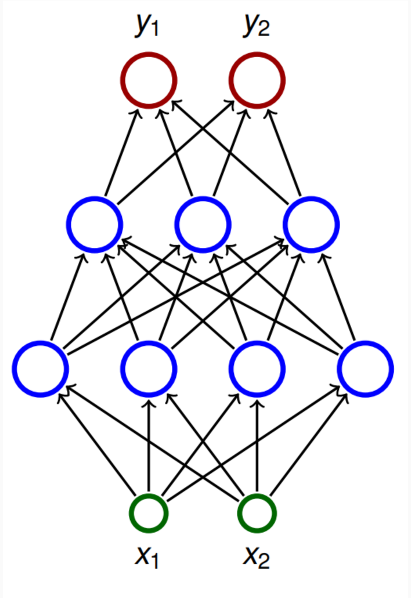
The parametrization Θ will depend on the concrete form of function approximator used.

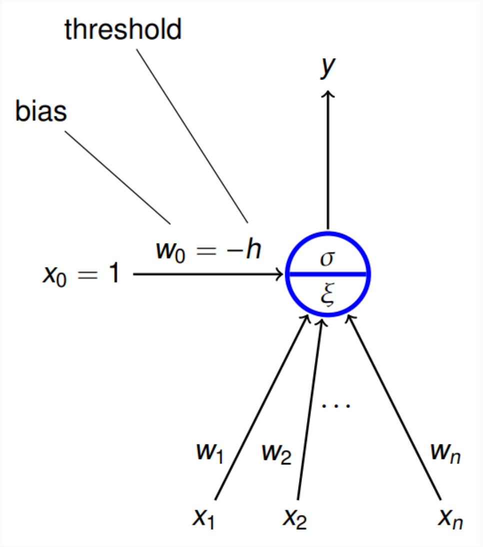
Forms of function approximators

- tabular
  - $\theta =$ the vector containing the contents of the table
- linear
  - $\Theta = \mathcal{S} = \mathbb{R}^n$ and e.g. $V_\theta(s) = \theta^\top \cdot s$
- neural nets
  - $\theta =$ NN weights and biases
- decision trees
- ...

We require the approximators to be differentiable and to admit a training method suitable for non-stationary data.

Neural nets (source: slides by T. Brázdil)





Policy evaluation with FAs

Task: given a policy π and FA $V\colon \mathbb{R}^n \times \Theta \to \mathbb{R}$, find θ s.t. V_θ is "close" to v^π.

"Closeness" can be expressed using various loss functions. Typically, we want to minimize the mean squared error (MSE):

$$MSE(v^\pi, V_\theta) = \frac{1}{2}\mathbb{E}_{s\sim\mu}\left[(v^\pi(s) - V_\theta(s))^2\right] = \frac{1}{2}\sum_{s\in\mathcal{S}} \mu(s)\cdot\left[(v^\pi(s) - V_\theta(s))^2\right],$$

where μ is some distribution over states expressing how much do we care about errors in particular states.

A local minimum of MSE can be found gradient descent: making successive step in the direction opposite to the gradient of MSE.

Recall: gradients

Definition 51

Given a scalar function $f(x_1, \dots, x_n, \theta_1, \dots, \theta_m) \colon \mathbb{R}^n \times \Theta \to \mathbb{R}$ (where $\Theta \subseteq \mathbb{R}^m$), the gradient of f w.r.t. parameters $\theta = (\theta_1, \dots, \theta_m)$ is the vector function

$$\nabla_\theta f = (\frac{\partial f}{\partial \theta_1}, \dots, \frac{\partial f}{\partial \theta_m}) \text{ of type } \mathbb{R}^n \times \Theta \to \mathbb{R}^m$$

When f is a function approximator defined by a neural net, the value of the gradient $\nabla_\theta f(x, \theta)$ at a given point $(x, \theta) = (x_1, \dots, x_n, \theta_1, \dots, \theta_m)$ can be computed by backpropagation (under some usual conditions like smoothness, etc.).

Gradient descent for policy evaluation

To (locally) minimize $MSE(v^\pi, V_\theta)$, it suffices to perform (sufficiently small) steps in the negative direction of the current gradient, i.e., repeatedly perform updates:

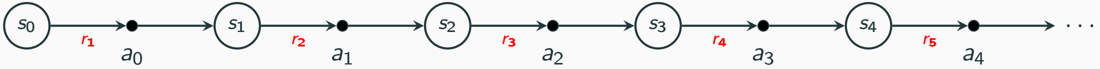
$$\begin{aligned}\theta \leftarrow \theta - \alpha \cdot \nabla_\theta MSE(v^\pi, V_\theta) &= \theta - \alpha \cdot \nabla_\theta \frac{1}{2} \cdot \mathbb{E}_{s\sim\mu}\left[\left(v^\pi(s) - V_\theta(s)\right)^2\right] \\ &= \theta - \frac{\alpha}{2} \cdot \mathbb{E}_{s\sim\mu}\left[\nabla_\theta\left(v^\pi(s) - V_\theta(s)\right)^2\right] \\ &= \theta + \alpha \cdot \mathbb{E}_{s\sim\mu}\left[\left(v^\pi(s) - V_\theta(s)\right) \cdot \nabla_\theta V_\theta(s)\right]\end{aligned}$$

The expected value above is typically impossible to evaluate in practice. Instead we estimate it by samples $\Rightarrow$ stochastic gradient descent.

We typically take $\mu(s)$ representing the overall fraction of time spent in s when behaving according to μ. Hence, $\mathbb{E}_{s\sim\mu}$ can be estimated by sampling a trajectory from μ and performing the update for each s on the trajectory in an every-visit fashion.

Stochastic gradient policy evaluation + MC instantiation

We keep sampling trajectories τ from π:



For each timestep t we perform the update of parameters

$$\theta \leftarrow \theta + \alpha \cdot \left[\left(v^{\pi}(s_t) - V_{\theta}(s_t)\right) \cdot \nabla_{\theta} V_{\theta}(s_t)\right].$$

Problem: in policy evaluation setting, we do not know $v^{\pi}(s_t)$. Hence, we estimate it using RL targets.

The simplest is the Monte Carlo target: estimate s_t by the discounted return of the sampled trajectory from s_t, i.e. perform updates of the form

$$\theta \leftarrow \theta + \alpha \cdot \left[\left(G_t(s_t) - V_{\theta}(s_t)\right) \cdot \nabla_{\theta} V_{\theta}(s_t)\right].$$

Gradient Monte Carlo policy evaluation: pseudocode

```
Algorithm 3: Gradient MC evaluation
Input: Policy $\pi$, FA $V : \mathcal{S} \times \Theta \to \mathbb{R}$, step size $\alpha$
Output: Approximation $V_\theta$ of $v^\pi$
initialize $\theta$ arbitrarily;
repeat
    sample trajectory $\tau = s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \ldots$ from $\pi$;
    foreach $t \in \{0, \ldots, T-1\}$ do
        $\theta \leftarrow \theta + \alpha \cdot [G_t(\tau) - V(s_t, \theta)] \cdot \nabla_\theta V(s_t, \theta)$
until timeout;
```

Semi-gradient TD(0)

In the gradient update formula

$$\theta \leftarrow \theta + \alpha \cdot \left[\left(v^\pi(s_t) - V_\theta(s_t)\right) \cdot \nabla_\theta V_\theta(s_t)\right].$$

we can also estimate $v^\pi(s_t)$ with the TD(0) target:

$$\theta \leftarrow \theta + \alpha \cdot \left[\left(r_{t+1} + \gamma \cdot V_\theta(s_{t+1}) - V_\theta(s_t)\right) \cdot \nabla_\theta V_\theta(s_t)\right].$$

This yield the semi-gradient TD(0) policy evaluation algorithm.

Why semi-gradient?

Gradient vs. semi-gradient TD(0)

Recall that our ultimate goal is to minimize

$$MSE(v^\pi, V_\theta) = \frac{1}{2}\mathbb{E}_{s\sim\mu}\left[(v^\pi(s) - V_\theta(s))^2\right].$$

The gradient of this loss is

$$\nabla_\theta \frac{1}{2}\mathbb{E}_{s\sim\mu}\left[(v^\pi(s) - V_\theta(s))^2\right] = \frac{1}{2}\mathbb{E}_{s\sim\mu}\left[\nabla_\theta(v^\pi(s) - V_\theta(s))^2\right],$$

Estimation with sample trajectory $\tau = s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \ldots$ and substituting $v^\pi(s)$ with the TD(0) target would yield

$$\nabla_\theta MSE(v^\pi, V_\theta) \approx \frac{1}{2}\nabla_\theta\big(r_{t+1}+\gamma V_\theta(s_{t+1})-V_\theta(s_t)\big)^2 = (r_{t+1}+\gamma V_\theta(s_{t+1})-V_\theta(s_t))\cdot(\gamma\nabla_\theta V_\theta(s_{t+1})-\nabla_\theta V_\theta(s_t))$$

different update then semi-gradient TD(0)! However, this full gradient:

- is more expensive to compute (2 backpropagations per update);
- does not really express TD(0) idea (the update target is not fixed).

Semi-gradient TD(0): pseudocode

Algorithm 4: Semi-gradient TD(0) evaluation

```
Input: Policy $\pi$, FA $V : \mathcal{S} \times \Theta \to \mathbb{R}$, step size $\alpha$
Output: Approximation $V_\theta$ of $v^\pi$
initialize $\theta$ arbitrarily;
repeat
    $s \leftarrow$ initial state;
    $a \sim \pi(s)$;
    while $s$ not terminal do
        $s' \sim p(s, a)$;
        $r \leftarrow r(s, a, s')$;
        $a' \sim \pi(s')$;
        $\theta \leftarrow \theta + \alpha \cdot [r + \gamma \cdot V(s', \theta) - V(s, \theta)] \cdot \nabla_\theta V(s, \theta)$;
        $s \leftarrow s'$; $a \leftarrow a'$
until timeout;
```

On-policy control with function approximation

Semi-gradient SARSA uses the same idea as TD(0), but with Q-approximator, i.e. $Q \colon \mathbb{R}^n \times \mathcal{A} \times \Theta \to \mathbb{R}$.

Behavior policy = e.g. ε-greedy with respect to the current Q. For a sampled trajectory $\tau =$

$$\begin{tikzcd} s_0 \arrow[r, "r_1"'] & a_0 \arrow[r] & s_1 \arrow[r, "r_2"'] & a_1 \arrow[r] & s_2 \arrow[r, "r_3"'] & a_2 \arrow[r] & s_3 \arrow[r, "r_4"'] & a_3 \arrow[r] & s_4 \arrow[r, "r_5"'] & a_4 \arrow[r] & \cdots \end{tikzcd}$$

we perform, in each timestep t, an update

$$\theta \leftarrow \theta + \alpha \cdot \Big[\Big(r_{t+1} + \gamma \cdot Q_\theta(s_{t+1}, a_{t+1}) - Q_\theta(s_t, a_t)\Big) \cdot \nabla_\theta Q_\theta(s_t, a_t)\Big].$$

Semi-gradient SARSA: pseudocode

```
Algorithm 5: Semi-gradient SARSA
Input: FA $Q \colon \mathcal{S} \times \mathcal{A} \times \Theta \to \mathbb{R}$, step size $\alpha$
Output: Approximation $Q_\theta$ of $q^*$
initialize $\theta$ arbitrarily;
repeat
    $s \leftarrow$ initial state;
    $\pi \leftarrow$ policy $\varepsilon$-greedy w.r.t. $Q_\theta$;
    $a \sim \pi(s)$;
    while $s$ not terminal do
        $s' \sim p(s, a)$;
        $r \leftarrow r(s, a, s')$;
        $a' \sim \pi(s')$;
        $\theta \leftarrow \theta + \alpha \cdot [r + \gamma \cdot Q_\theta(s', a') - Q_\theta(s, a)] \cdot \nabla_\theta Q_\theta(s, a)$;
        $s \leftarrow s'$; $a \leftarrow a'$
until timeout;
```

Representing actions in DNNs

How to represent actions in the (say, DNN) function approximator Q is largely a domain-dependent engineering choice.

If the set of actions $\mathcal{A} = \{a^1, \ldots, a^k\}$ is discrete and reasonably small, we can consider a net which inputs a state (i.e., n input neurons when $\mathcal{S} = \mathbb{R}^n$) and outputs an $|\mathcal{A}|$-dimensional vector (i.e., one output neuron per action), so that the output of the i-th neuron on input s is interpreted as $Q(s, a^i)$.

I.e., in such a case we consider Q to be function of type $Q\colon \mathcal{S} \times \Theta \to \mathbb{R}^{|\mathcal{A}|}$.

On-policy semi-gradient methods: concluding remarks

The presented algorithms can be instantiated also with other types of returns, such as:

- n-step returns
  - n-step SARSA update:
    $\theta \leftarrow \theta + \alpha \cdot \left[\left(\underbrace{r_{t+1} + \gamma \cdot r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^{n} \cdot Q_\theta(s_{t+n}, a_{t+n})}_{G_{t:t+n,\theta}} - Q_\theta(s_t, a_t)\right) \cdot \nabla_\theta Q_\theta(s_t, a_t)\right]$
- forward-view λ-returns
  - SARSA(λ) update: $\theta \leftarrow \theta + \alpha \cdot \left[\left((1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n,\theta} - Q_\theta(s_t, a_t)\right) \cdot \nabla_\theta Q_\theta(s_t, a_t)\right]$
- backward-view λ-returns (optional reading: Sutton&Barto, sections 12.2 and 12.7)

Value-Based Off-Policy Control with Approximators: DQNs and Friends

Off-policy methods with function approximation

...are tricky to get right, already in the case of policy evaluation. The training can become very unstable.

For on-policy (semi)-gradient methods, one can typically prove convergence to correct/optimal values at least in the case of linear function approximation (though not in the more general case of NN approximators).

Off-policy semi-gradient methods, such as:

- TD with importance sampling (not covered here), or
- Q-learning with function approximators (will be covered a bit later),

can diverge already with linear function approximators.

Divergence examples (high-level)

- Baird's counterexample: semi-gradient TD with importance sampling can diverge in presence of linear FAs
- Moreover, the divergence is not due to the instability of (semi)-gradient descent. Tsitsiklis and Van Roy's counterexample shows divergence even in the case where each update completely replaces the current θ with the optimal θ^* which minimizes the MSE between V_θ and the TD(0) update target. The problem lies in the off-policy distribution of updates.
- Counterexamples explained in optional reading: Sutton&Barto, Sec. 11.2.

Deadly triad

Identified by Sutton&Barto: risk of training instability and divergence steeply rises when combining:

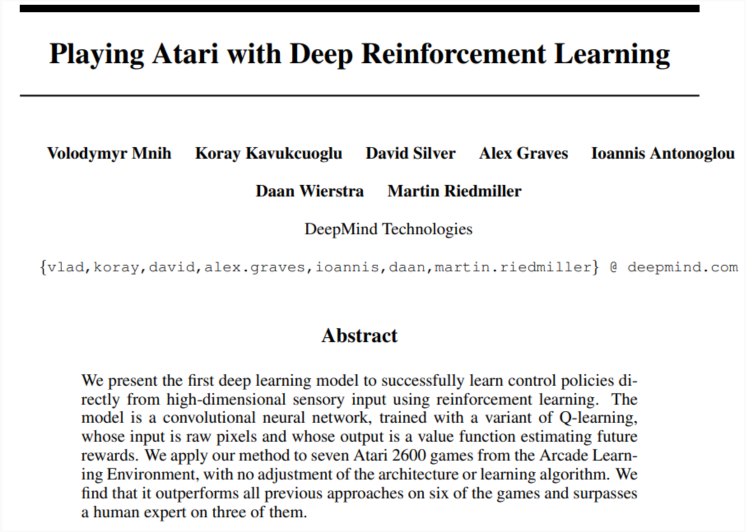
- function approximation,
- bootstrapping, and
- off-policy training.

But often we want to do just that. :)

Practical solution: Happily do the deadly triad, but use insights from supervised learning to develop additional techniques that help stabilize the training.

Deep Q-Networks (DQN)

2013 arXiv tech. report, there is also follow-up 2015 Nature paper



Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou

Daan Wierstra Martin Riedmiller

DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

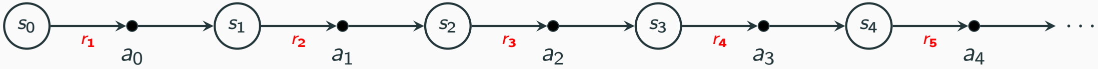
Abstract

We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

Q-learning with function approximators

Same semi-gradient idea as in TD(0), SARSA: adjust θ to bring $Q_\theta(s, a)$ closer to the fixed Q-learning update target.

I.e., for a sampled trajectory



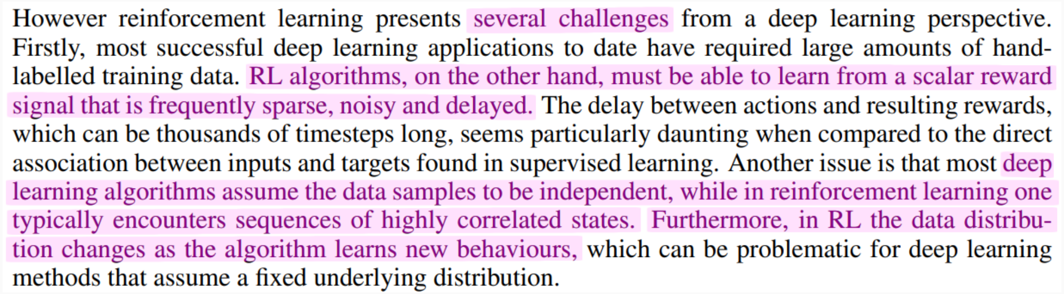
and its timestep t, the update is

$$\theta \leftarrow \theta + \alpha \cdot \left[\left(r_{t+1} + \gamma \cdot \max_{a \in \mathcal{A}(s_{t+1})} Q_\theta(s_{t+1}, a) - Q_\theta(s_t, a_t)\right) \cdot \nabla_\theta Q_\theta(s_t, a_t)\right].$$

But performing the updates based solely on the current step would be susceptible to instability due to the presence of the deadly triad.

Deep Q-learning challenges

From Mnih et al. *Playing Atari with Deep Reinforcement Learning*:



However reinforcement learning presents several challenges from a deep learning perspective. Firstly, most successful deep learning applications to date have required large amounts of hand-labelled training data. RL algorithms, on the other hand, must be able to learn from a scalar reward signal that is frequently sparse, noisy and delayed. The delay between actions and resulting rewards, which can be thousands of timesteps long, seems particularly daunting when compared to the direct association between inputs and targets found in supervised learning. Another issue is that most deep learning algorithms assume the data samples to be independent, while in reinforcement learning one typically encounters sequences of highly correlated states. Furthermore, in RL the data distribution changes as the algorithm learns new behaviours, which can be problematic for deep learning methods that assume a fixed underlying distribution.

Experience replay

Originated in the work of Long-Ji Lin, e.g.: *Reinforcement Learning for Robots Using Neural Networks*, dissertation, 1993.

Definition 52: Experience

An experience is a 4-tuple $(s, a, r, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathbb{R}$ interpreted as ("state", "action played in it", "reward obtained", "next state observed").

- DQN does not perform update based only on the current step. Instead, for each sampled trajectory $\tau = s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \ldots$ and each timestep t it:
    - first stores the one-step experience $(s_t, a_t, r_{t+1}, s_{t+1})$ in a data structure $\mathcal{B}$ called replay buffer;
    - then, sample a random minibatch of experiences $B \subseteq \mathcal{B}$ of a given minibatch size *Bsize*
    - perform a minibatch-gradient-descent update w.r.t. B: compute the gradient of the Q-learning loss for each $e \in B$ and then update θ in the direction of an average gradient over the whole B.

Minibatch update

Fix a minibatch B.

For each $e = (s, a, r, s') \in B$ we compute the gradient of the Q-learning loss $\nabla_\theta \mathcal{L}(\theta, e)$ at point e:

$$\nabla_\theta \mathcal{L}(\theta, e) = \nabla_\theta \frac{1}{2}\Big(\underbrace{r + \gamma \cdot \max_{b \in \mathcal{A}(s')} Q_\theta(s', b)}_{\text{fixed target}} - Q_\theta(s, a)\Big)^2$$

$$= \Big[\Big(\underbrace{r + \gamma \cdot \max_{b \in \mathcal{A}(s')} Q_\theta(s', b)}_{=0 \text{ if } s \text{ terminal}} - Q_\theta(s, a)\Big) \cdot \nabla_\theta Q_\theta(s, a)\Big]$$

We then perform an update in the direction of average gradient:

$$\theta \leftarrow \theta + \alpha \cdot \frac{1}{|B|} \sum_{e \in B} \nabla_\theta \mathcal{L}(\theta, e).$$

Experience replay rationale

- helps decorrelate the DNN training data
- helps to prevent catastrophic forgetting
- improves data efficiency via experience re-use

Why good match for deep Q-learning? Experience replay is by design off-policy since we train on old data, which were sampled from different policy than the current one.

Replay buffer implementation

The replay buffer $\mathcal{B}$ is typically not unbounded, but has a fixed capacity $\mathcal{B}size$. Replacement is eventually needed. If $\mathcal{B}$ is full, the oldest experience if removed ($\mathcal{B} =$ queue).

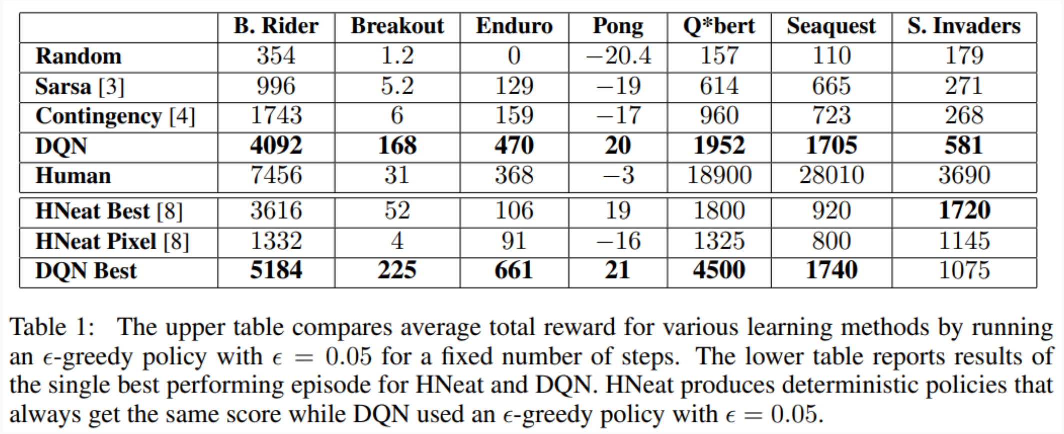
How to sample the minibatches?

- Original DQN: uniformly from $\mathcal{B}$.
- Alternative: prioritized experience replay: each experience is assigned a priority (several heuristics exist). An experience is sampled into a minibatch with probability proportional to its priority.

DQN: 2013 pseudocode

```
Algorithm 6: DQN with replay buffer
Input: Black-box MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, r)$, approximator $Q$; hyperparam's
       $\varepsilon, \mathcal{B}size, Bsize, \ldots$
Output: Approximation $Q_\theta$ of $q^*$
initialize $\theta$ arbitrarily; initialize empty replay buffer $\mathcal{B}$ of capacity $\mathcal{B}size$;
repeat
    $s \leftarrow$ initial state;
    while $s$ not terminal do
        $\pi \leftarrow$ policy $\varepsilon$-greedy w.r.t. $Q_\theta$;
        $a \sim \pi(s)$;
        $s' \sim p(s, a)$;
        $r \leftarrow r(s, a, s')$;
        store $(s, a, r, s')$ in $\mathcal{B}$;
        sample a minibatch $B$ of size $Bsize$ from replay buffer $\mathcal{B}$;
        perform the minibatch update $\theta \leftarrow \theta + \alpha \cdot \frac{1}{Bsize} \sum_{e \in B} \nabla_\theta \mathcal{L}(\theta, e)$ (see this slide);
        $s \leftarrow s'$;
until timeout;
```

DQN: 2013 results



	B. Rider	**Breakout**	**Enduro**	**Pong**	**Q*bert**	**Seaquest**	**S. Invaders**
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

Table 1: The upper table compares average total reward for various learning methods by running an ϵ-greedy policy with $\epsilon = 0.05$ for a fixed number of steps. The lower table reports results of the single best performing episode for HNeat and DQN. HNeat produces deterministic policies that always get the same score while DQN used an ϵ-greedy policy with $\epsilon = 0.05$.

Mnih et al. (2013, arXiv)

Target networks: another stabilizing factor in DQNs

Introduced in the reviewed version of DQN paper:

Mnih et al.: Human-level control through deep reinforcement learning. *Nature*, 518 (2015).

Performing the (minibatch) Q-update looks like supervised learning:

change θ so that $Q_\theta(s,a)$ gets closer to the fixed target $r + \gamma \cdot \max_{b \in \mathcal{A}(s')} Q_\theta(s', b)$,

where (s, a, r, s') is the processed experience.

Performing the (minibatch) Q-update looks like supervised learning, but:

change θ so that $Q_\theta(s,a)$ gets closer to the fixed target $r + \gamma \cdot \max_{b \in \mathcal{A}(s')} \underbrace{Q_\theta(s', b)}_{\text{the ``label'' changes with each update!}}$,

Target networks: idea

To stabilize learning, we use two networks: the main network, and the target network. They have the same architecture (denote it by Q), but their weights may differ during the execution of the algorithm.

We denote:

- θ - weights of main network
- $\hat{\theta}$ - weights of target network

Usage:

- The target network is used only to compute TD targets when computing losses:

$$\nabla_\theta \mathcal{L}(\theta, e) = \big(r + \gamma \cdot \max_{b \in \mathcal{A}(s')} Q(s', b, \hat{\theta}) - Q(s, a, \theta)\big) \cdot \nabla_\theta Q(s, a, \theta)$$

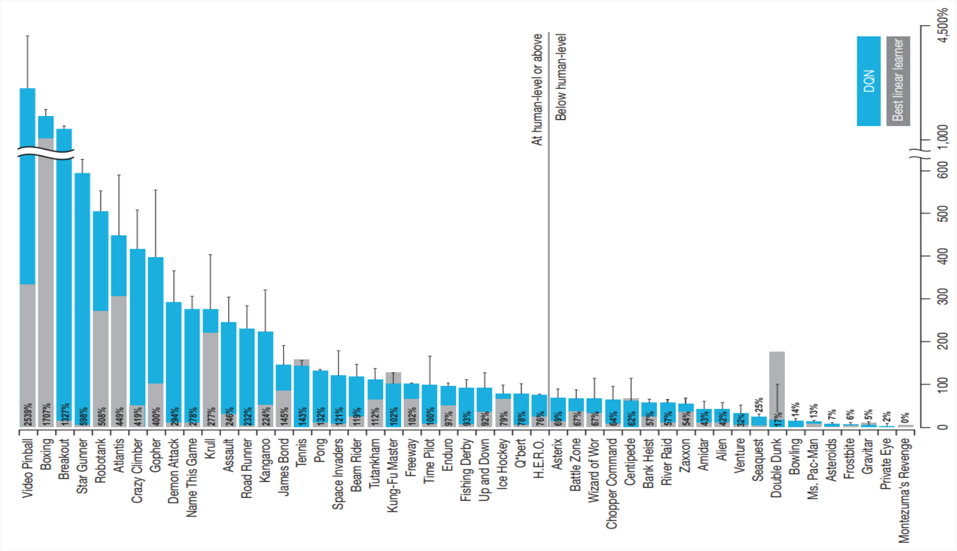
- At the start, and also in periodic intervals (but not after each update!) the two networks are synchronized by performing $\hat{\theta} \leftarrow \theta$. Other than this, $\hat{\theta}$ stays fixed, the gradient steps are only used to update θ (i.e., the main network).

DQN: 2015 pseudocode

Algorithm 7: DQN with replay buffer and target network

```
Input: Black-box MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, r)$, approximator $Q$; hyperparam's $\varepsilon, \mathcal{B}size, Bsize, C, \dots$
Output: Approximation $Q_\theta$ of $q^*$
initialize $\theta$ arbitrarily; $\hat{\theta} \leftarrow \theta$; $counter \leftarrow C$;
initialize empty replay buffer $\mathcal{B}$ of capacity $\mathcal{B}size$;
repeat
    $s \leftarrow$ initial state;
    while $s$ not terminal do
        if $counter = 0$ then $\hat{\theta} \leftarrow \theta$; $counter \leftarrow C$ else $counter \leftarrow counter - 1$;
        $\pi \leftarrow$ policy $\varepsilon$-greedy w.r.t. $Q_\theta$;
        $a \sim \pi(s)$;
        $s' \sim p(s, a)$;
        $r \leftarrow r(s, a, s')$;
        store $(s, a, r, s')$ in $\mathcal{B}$;
        sample a minibatch $B$ of size $Bsize$ from replay buffer $\mathcal{B}$;
        perform the minibatch update $\theta \leftarrow \theta + \alpha \cdot \frac{1}{Bsize} \sum_{e \in B} \nabla_\theta \mathcal{L}(\theta, e)$, where
         $\nabla_\theta \mathcal{L}(\theta, e) = \big(r + \gamma \cdot \max_{b \in \mathcal{A}(s')} Q(s', b, \hat{\theta}) - Q(s, a, \theta)\big) \cdot \nabla_\theta Q(s, a, \theta)$;
        $s \leftarrow s'$;
until timeout;
```

DQN: 2015 results



Engineering behind DQN for Atari: partial observability

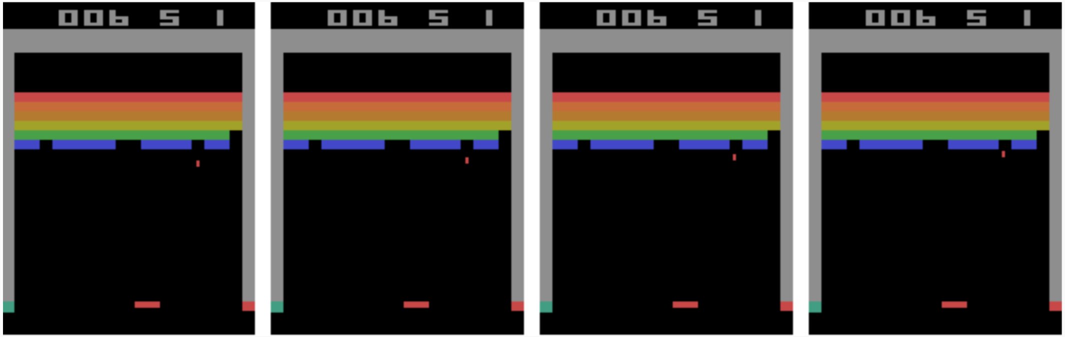
True state = current state (program counter + variable values) of the game program.

We do not see this – only the frames rendered on screen.

Solving partially observable environments requires (per some POMDP theory) making decisions based on the whole history of observations. This is computationally demanding (recurrent NNs...).

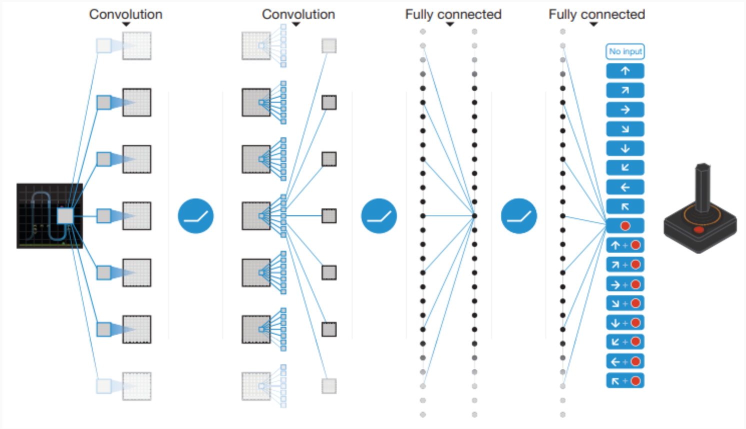
DQN for Atari solves this by feeding the last 4 observed frames into the NN. This is typically enough to deduce the dynamics of the current play.

DQN: dynamics from limited frame history



Engineering behind DQN for Atari: the network

Inputs four 84x84px images, then 3 convolutional layers, then two fully connected layers. all with ReLU activations. Outputs Q-estimate for each action.



source: Mnih et al. (Nature, 2015), details in appendix “Model architecture”

DQN for Atari: dirty engineering tricks

- Preprocessing: 210x160 RGB color images are converted to grayscale and resized to 84x84 resolution.
- Frame skipping: the agent only observes and acts in every K-th frame, for the frames in between, the last selected action is repeated without providing the frame to the agent. (In the paper, $K = 4$.)
- Reward clipping: all positive one-step Atari rewards are clipped to $+1$, all negative ones are clipped to -1 (Atari gives integer rewards).
- TD error clipping: for each update, the Q-learning error $r + \gamma \cdot \max b \in \mathcal{A}(s') Q(s', b) - Q(s, a)$ is clipped to $[-1, 1]$.

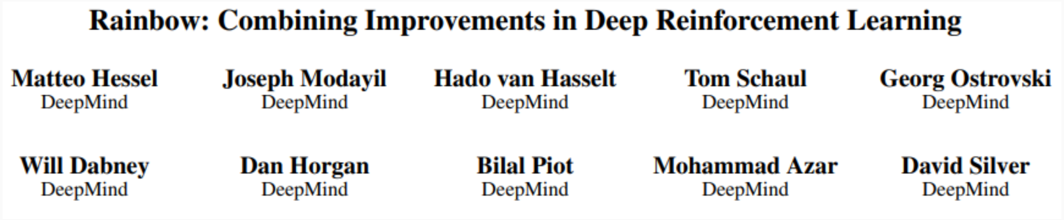
DQN for Atari: selected hyperparameters (nex)

minibatch size *Bsize*	32
replay buffer size $\mathcal{B}size$	1,000,000
target network update freqeuency *C*	10,000
discount factor γ	0.99
update frequency (steps between two minibatch updates)	4
learning rate	0.00025
initial ε	1
final ε (linear decay)	0.1
final decay frame	1,000,000
random policy played for init.	50,000 frames
max. do-nothing actions at episode start	30

RAINBOW of heuristics

Multitude of heuristics for the improvement of DQN were developed over time. Some of them make sense also in the context of other deep RL algorithms.

The RAINBOW agent combines six such heuristics to further improve the DQN performance on Atari games.



Rainbow: Combining Improvements in Deep Reinforcement Learning

Matteo Hessel DeepMind
Joseph Modayil DeepMind
Hado van Hasselt DeepMind
Tom Schaul DeepMind
Georg Ostrovski DeepMind
Will Dabney DeepMind
Dan Horgan DeepMind
Bilal Piot DeepMind
Mohammad Azar DeepMind
David Silver DeepMind

(In proceedings of AAAI 2018.)

Rainbow heuristics

- dueling networks architecture
- double DQN
- prioritized experience replay
- n-step rewards
- distributional learning
- noisy networks

Action advantage

Idea: imagine that for some state s, the Q-values of all actions are high. Then s should be in some sense valuable in itself.

Definition 53: Advantage

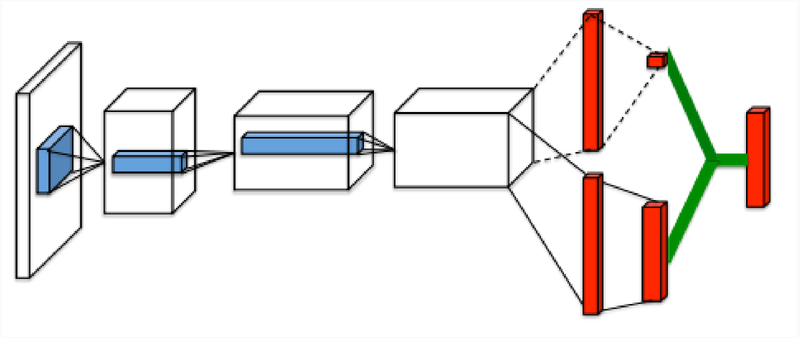
Let π be a policy. An advantage function $adv^{\pi} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is defined as

$$adv^{\pi}(s, a) = q^{\pi}(s, a) - v^{\pi}(s).$$

Dueling architecture splits certain layers of the neural network into two "streams", one estimating (somethign like) $v^{\pi}(s)$ and one estimating (something like) $adv^{\pi}(s, a)$. The final layer combines these estimates to produce an estimate of $q^{\pi}(s, a)$.

Dueling architecture

Wang et al.: *Dueling Network Architectures for Deep Reinforcement Learning*. In proceedings of ICML'16.



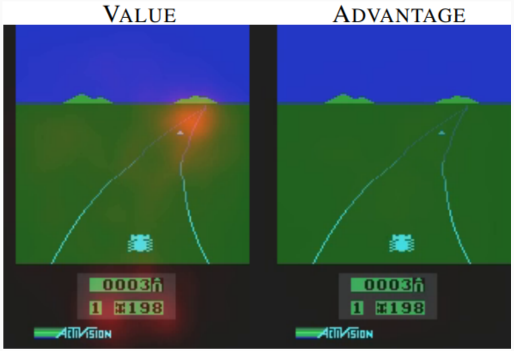
Dueling architecture: theory and training

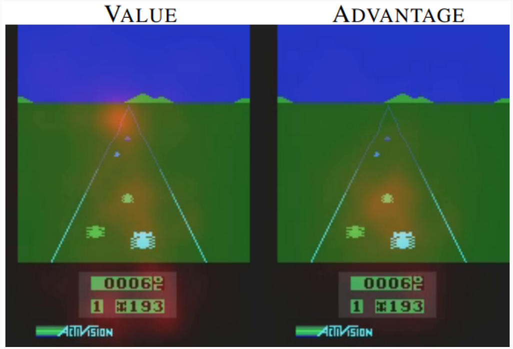
We have $Q_{\theta,\alpha,\beta}(s, a) = aggregate(V_{\theta,\alpha}(s), A_{\theta,\beta}(s, a))$, where

- θ - convolutional (or other feature extraction) layer parameters
- α - value channel parameters
- β - advantage channel parameters

The whole network Q is trained to estimate q^{π} (where π is the target policy) using any deep RL algorithm (e.g. DQN, in which case π is the optimal policy). There is nothing new from RL perspective here, all the novelty is inside the network. The factorization into state value and advantage is supposed to help the network “focus” on features that are important to recognize valuable states and features that help us rank actions.

Dueling architecture: Atari example





From Wang et al.: *Dueling Network Architectures for Deep Reinforcement Learning*. In proceedings of ICML'16.

Learning state values and advantages

- The whole net is trained end-to-end to predict q^π.
- How do we ensure the value/advantage channels are trained to predict state values/advantages?
- By a suitable choice of aggregator:
    - $Q_{\theta,\alpha,\beta}(s,a) = V_{\theta,\alpha}(s) + A_{\theta,\beta}(s,a)$ does not work: e.g. $V_{\theta,\alpha}$ could converge to constant 0 and $A_{\theta,\beta}$ to q^π.
    - $$Q_{\theta,\alpha,\beta}(s,a) = V_{\theta,\alpha}(s) + A_{\theta,\beta}(s,a) - \max_{b \in \mathcal{A}(s)} A_{\theta,\beta}(s,b),$$

      the training then indeed pushes $V_{\theta,\alpha}$ to v^π and $A_{\theta,\beta}$ to $adv^\pi + c$ where c is some constant. Issues: not differentiable, update sensitive to the value of maximizing action changes.

Aggregation in Rainbow

The point: aggregating layer should anchor the sum of the channels to same baseline value derived non-trivially from the advantages (if all advantages shift up/down, so should the baseline). Rainbow uses mean advantage baseline:

$$Q_{\theta,\alpha,\beta}(s,a) = V_{\theta,\alpha}(s) + A_{\theta,\beta}(s,a) - \frac{1}{|\mathcal{A}(s)|}\sum_{b\in\mathcal{A}(s)} A_{\theta,\beta}(s,b)$$

pushes the value channel to predict $\frac{1}{|\mathcal{A}(s)|}\sum_{a\in\mathcal{A}(s)} q^{\pi}(s,a)$.

Double DQN

van Hasselt, Guez, Silver: *Deep Reinforcement Learning with Double Q-learning.* In proceedings of AAAI 2016.

Similar idea to tabular Double Q-learning (use different estimates for selecting maximizing action in bootstrap and for evaluating the bootstrap), but instead of independently updated networks uses main and target networks. I.e., for experience (s, a, r, s'), the update is:

$$\theta \leftarrow \theta + \alpha \cdot \left[r + \gamma \cdot Q_{\hat{\theta}}\left(s', \arg\max_{b \in \mathcal{A}(s')} Q_\theta(s', b)\right) - Q_\theta(s, a)\right] \nabla_\theta Q_\theta(s, a),$$

where $\hat{\theta}$ is the parameter vector of the target network.

Prioritized experience replay (Schaul et al., ICLR'16)

Each experience $e = (s, a, r, s')$ in the replay buffer is assigned a priority according to its TD-error

$$p_e = |r + \gamma \cdot \max_{b \in \mathcal{A}(s')} Q_{\hat{\theta}}(s', b) - Q_\theta(s, a)| + \varepsilon$$

($\varepsilon > 0$ ensures all priorities are positive).

The probability of sampling an experience e from the buffer is set to $\frac{p_e^\alpha}{\sum_{e' \in \mathcal{B}} p_{e'}^\alpha}$, where $\alpha > 0$ is a hyperparameter controlling the degree of prioritization.

Prioritization induces bias: the sampled experiences no longer follow the same distribution as sampled trajectories. We can correct this by using importance sampling during updates:

$$\theta \leftarrow \theta + \alpha \cdot \left(\frac{1}{|\mathcal{B}|} \cdot \frac{1}{p_e^\alpha} \right)^\beta \cdot \left[r + \gamma \cdot Q_{\hat{\theta}}\big(s', \arg\max_{b \in \mathcal{A}(s')} Q_\theta(s', b)\big) - Q_\theta(s, a) \right] \cdot \nabla_\theta Q_\theta(s, a),$$

where $\beta > 0$ determines the degree of IS correction (annealed to 1 during training).

n-step returns

Self-explanatory, use *n*-step return with Q-learning bootstrap when computing TD target.

How to combine with replay buffer? Each experience stores a single step.

Solutions:

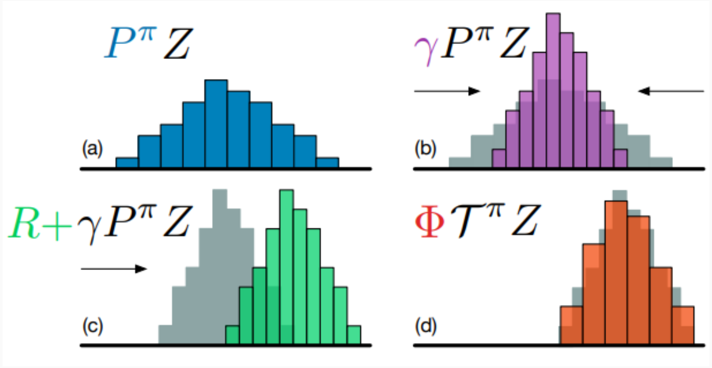
- Store experiences in $\mathcal{B}$ sequentially, with each sampled experience, retrieve also the next $n-1$ ones (up to episode termination). Requires careful implementation.
- Naive: each element of $\mathcal{B}$ consists of n consecutive experiences (space inefficient).

Given consecutive experiences $(s_t, a_t, r_{t+1}, s_{t+1}), (s_{t+1}, a_{t+1}, r_{t+2}, s_{t+2}), \ldots, ((s_{t+n-1}, a_{t+n-1}, r_{t+n}, s_{t+n}))$, perform update

$$\theta \leftarrow \theta + \alpha \cdot \left[r_{t+1} + \gamma r_{t+2} + \cdots + \cdot\gamma^{n-1} r_{t+n} + \gamma^n \max_{b \in \mathcal{A}(s_{t+n})} Q_{\hat{\theta}}(s_{t+n}, b) - Q_\theta(s_t, a_t)\right] \nabla_\theta Q_\theta(s_t, a_t).$$

Distributional learning

Very rough idea: instead of expected returns, predict (discretized) distribution of returns.



Source: Bellemare, Dabney, Munos: *A Distributional Perspective on Reinforcement Learning*. In proceedings of ICML'17.

We still optimize the expected value of the distribution, but the NN processes richer information (neurological inspiration).

Noisy nets

Fortunato et al.: *Noisy Networks for Exploration* . In proceedings of ICRL'17.

An alternative way of achieving exploration (without ε-greedy policies). Replaces linear layers $y = W \cdot x + b$ with noisy layers of the form

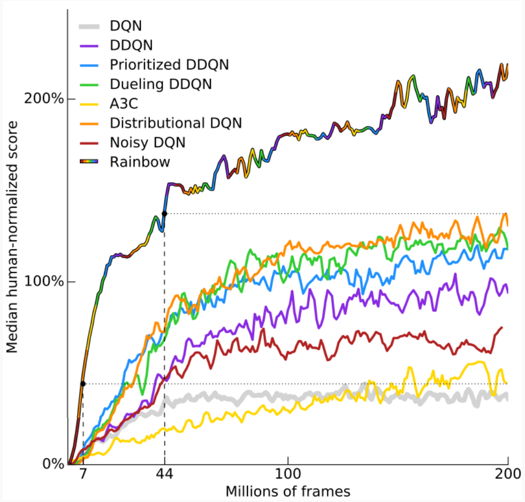
$$y = (\mu_w + \sigma_w \odot \varepsilon_w) \cdot x + \mu_b + \sigma_b \odot \varepsilon_b,$$

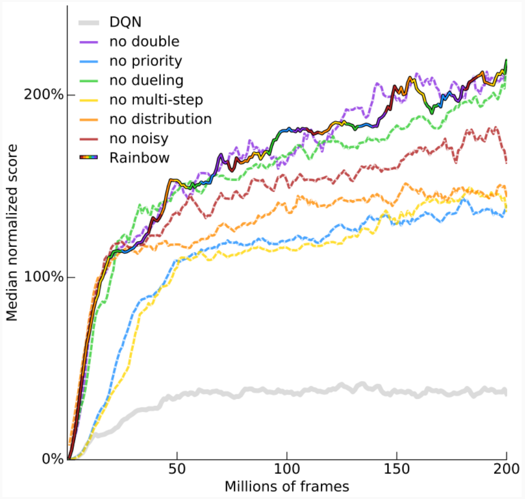
where matrices μ_w, σ_w and vectors μ_b, σ_b are learnable, matrix ε_w and vector ε_b consist of random noise, and $\odot$ represents component-wise multiplication.

The loss function of the DQN training is then encapsulated in expectation over the noise.

Interesting point: the net can learn to adjust σ's and thus the degree of exploration over time.

Rainbow: evaluation and ablations (Hessel et al., 2017)


DQN
DDQN
Prioritized DDQN
Dueling DDQN
A3C
Distributional DQN
Noisy DQN
Rainbow
Median human-normalized score
Millions of frames


DQN
no double
no priority
no dueling
no multi-step
no distribution
no noisy
Rainbow
Median normalized score
Millions of frames