SDU

4) Model-Free Reinforcement Learning

Melih Kandemir

University of Southern Denmark
Department of Mathematics and Computer Science (IMADA)

Discriminative versus generative models

Consider the supervised learning problem $(x, y) \sim p(x, y)$ where x are inputs and y are labels. Given a data set $\mathcal{D} = \{(x_i, y_i) : i = 1, \ldots, N\}$

- a **generative model** uses factorization $p(y)p(x|y)$ and learns both: $\arg\max_{\phi,\theta} \frac{1}{N} \sum_{i=1}^{N} p_\phi(y_i)p_\theta(x_i|y_i)$. **Value-based methods** learn $p(\pi)p(V|\pi)$, hence they try to explain the value generation process.
- a **discriminative model** uses the factorization $p(x)p(y|x)$, integrates out the first one $\mathbb{E}_{p(x)}[p(y|x)] \approx \frac{1}{N} \sum_{i=1}^{N} p(y_i|x_i)$ and learns only the second, i.e. only the label predictor

$$\arg\max_\theta \frac{1}{N} \sum_{i=1}^{N} p_\theta(y_i|x_i).$$

Policy-based methods learn $p(\pi|V)$ by doing

$$\mathbb{E}_{V\sim p(V)}[p(\pi|V)] \approx \frac{1}{N} \sum_{i=1}^{N} p(\pi|V_i) \Rightarrow \arg\max_\theta \frac{1}{N} \sum_{i=1}^{N} p_\theta(\pi|V_i)$$

where V_i are observed samples of reward-to-go values.

Value-based methods

The standard recipe of dynamic programming algorithms

$$V(s) := r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, \pi(s)) V(s')$$

is not feasible when:

i) The state space is too large: $|\mathcal{S}| >> 0$. Then approximate by **simulation**:

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

ii) Transition probabilities are not known: $p(s'|s, \pi(s))$. Then collect samples from the environment

$$r_t, s_{t+1} := \texttt{env.step}(\pi(s_t))$$

and approximate $V(s)$. This is called **value-based** approach to model-based reinforcement learning.

Monte Carlo (MC) simulation

The empirical mean of a sample $(v_i \sim p(V))_{i=1}^N$ with a fixed sample size N is

$$M_N = \frac{1}{N}\sum_{i=1}^{N} v_i \approx \mathbb{E}[V] = \sum_{v \in S} P(V = v)v.$$

This quantity can also be computed **online**:

$$M_{N+1} = M_N + \frac{1}{N+1}(v_{N+1} - M_N).$$

If $\mathbb{E}[V] = m$, then

$$\mathbb{E}[M_N|N] = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}[v_i] = m.$$

Hence M_N is an **unbiased** estimator of m. We also have

$$\mathrm{Var}(M_N) = \frac{1}{N^2}\sum_{i=1}^{N} \mathrm{Var}(v_i) = \frac{\sigma^2}{N}.$$

$\lim_{N\to\infty} \mathrm{Var}(M_N) = 0 \Rightarrow M_1, M_2, \ldots \to m$ w.p. 1 (law of large #s).

Policy evaluation with MC simulation

Given N trajectories $\{(s_t^i, a_t^i, r_t^i, s'^i_t)_{t=1}^T : 1 \leq i \leq N\}$ collected by a policy π from an MDP with episode length T. The MC estimate of the reward-to-go for a state s is

$$\widehat{V}(s) = \frac{1}{N} \sum_{i=1}^{N} \underbrace{\sum_{t=\tau_i}^{T} r_t^i}_{:=\widehat{V}_i(s)}, \qquad s \in \mathcal{S}$$

where τ_i is the first time step in trajectory i such that $s_{\tau_i} = s$. This scheme is called **first-visit Monte Carlo**. Such a choice allows us to fix the number of observations of $\widehat{V}_i(s)$ to N, hence makes $\widehat{V}_i(s)$ an unbiased estimate of its true value. In formal terms, $V^\pi(s) = \mathbb{E}[\widehat{V}_i(s)]$.

Improved efficiency with every-visit Monte Carlo

Collecting data from the real environment dynamics is often expensive. Let us then reason about how we can extract multiple samples from a single trajectory. Denote $\tau(s,i)$ as the list of the time steps when a state s is encountered in the ith episode. Then we can estimate the reward-to-go by

$$\widehat{V}(s) = \frac{\sum_{i=1}^{N}\sum_{j\in\tau(s,i)}\sum_{t=j}^{T} r_t^i}{\sum_{i=1}^{N}|\tau(s,i)|} = \frac{\sum_{i=1}^{N}\sum_{j=1}^{|\tau(s,i)|}\widehat{V}_{ij}(s)}{\sum_{i=1}^{N}|\tau(s,i)|}.$$

This approach is called the **every-visit** method. Here,

- $\widehat{V}_{ij}(s)$ is the MC estimate of the jth encounter of state s in the ith episode. As this is a sequence conditioned on s, we obviously have unbiased individual samples $\mathbb{E}[\widehat{V}_{ij}(s)] = V^{\pi}(s)$.
- The denominator $\sum_{i=1}^{N}|\tau(s,i)|$ counts the total number of appearances of state s in the data set. The problem is that this number is now a random variable. We need to check how this situation affect the bias of our estimator.

The random data size case

When N is a random variable and because we have $\mathbb{E}[v_i|N \geq i] = \mathbb{E}[v_1]$ due to i.i.d.ness, we attain

$$\begin{aligned}\mathbb{E}\left[\sum_{i=1}^{N} v_i\right] &= \sum_{i=1}^{\infty} P(N \geq i)\mathbb{E}[v_i|N \geq i] = \mathbb{E}[v_1]\sum_{i=1}^{\infty} P(N \geq i) \\ &= \mathbb{E}[v_1]\sum_{i=1}^{\infty}\sum_{n=i}^{\infty} P(N = n) = \mathbb{E}[v_1]\sum_{n=1}^{\infty} nP(N = n) \\ &= \mathbb{E}[v_1]\mathbb{E}[N]\end{aligned}$$

This is the **Wald's identity**. The conclusion is that $\mathbb{E}[M_N] = m/\mathbb{E}[N] \neq m$, hence M_N is a **biased** estimator of $\mathbb{E}[V]$. Note the use of the following property in the second equality

$$\sum_{i=1}^{\infty}\sum_{n=i}^{\infty} a_n = \sum_{n=1}^{\infty} na_n.$$

Every-visit estimator is biased but consistent

The bias reduces as more data are collected and the estimator converges the true value.

$$
\begin{aligned}
\lim_{N\to\infty} \frac{\sum_{i=1}^{N}\sum_{j=1}^{|\tau(s,i)|}\widehat{V}_{ij}(s)}{\sum_{i=1}^{N}|\tau(s,i)|} &= \lim_{N\to\infty} \frac{\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|\tau(s,i)|}\widehat{V}_{ij}(s)}{\frac{1}{N}\sum_{i=1}^{N}|\tau(s,i)|} \\
&= \frac{\lim_{N\to\infty}\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|\tau(s,i)|}\widehat{V}_{ij}(s)}{\lim_{N\to\infty}\frac{1}{N}\sum_{i=1}^{N}|\tau(s,i)|} \\
&= \frac{\mathbb{E}\left[\mathbb{E}\left[\widehat{V}_{ij}(s)\middle||\tau(s,i)|\right]\right]}{\mathbb{E}[|\tau(s,i)|]} \\
&= \frac{\mathbb{E}[\widehat{V}_{ij}(s)]\cancel{\mathbb{E}[|\tau(s,i)|]}}{\cancel{\mathbb{E}[|\tau(s,i)|]}} \\
&= V^{\pi}(s).
\end{aligned}
$$

Telescoping series and their convergence

A series $\sum_{t=0}^{\infty} u_t$ is defined as a **telescoping series** if $u_t = v_{t+1} - v_t$ for some sequence $(v_t)_{t \geq 0}$.

Lemma

For any convergent series $\sum_{t=0}^{\infty} a_t$, the related telescoping series $\sum_{t=0}^{\infty}(a_{t+1} - a_t)$ converges to $-a_0$.

Proof. By the assumption of convergence we have $\sum_{t=0}^{\infty} a_t = C$ for some $C \in \mathbb{R}$. Then

$$\begin{aligned}
\sum_{t=0}^{\infty}(a_{t+1} - a_t) &= \sum_{t=0}^{\infty} a_{t+1} - \sum_{t=0}^{\infty} a_t \\
&= -a_0 + a_0 + \sum_{t=0}^{\infty} a_{t+1} - C \\
&= -a_0 + \sum_{t=0}^{\infty} a_t - C = -a_0 \qquad \square
\end{aligned}$$

The telescoping lemma

Lemma

For any state sequence $(s_t)_{t=0}^{\infty}$ with $s_t \in \mathcal{S}$ and any bounded function V with domain $\mathcal{S}$, we have

$$\sum_{t=0}^{\infty} \gamma^t (r_t + \gamma V(s_{t+1}) - V(s_t)) = -V(s_0) + \sum_{t=0}^{\infty} \gamma^t r_t$$

Proof.

$$\sum_{t=0}^{\infty} \gamma^t (r_t + \gamma V(s_{t+1}) - V(s_t))$$

$$= \sum_{t=0}^{\infty} \gamma^t r_t + \sum_{t=0}^{\infty} (\gamma^{t+1} V(s_{t+1}) - \gamma^t V(s_t))$$

Define $a_t := \gamma^t V(s_t)$ and apply the telescoping series convergence result $\sum_{t=0}^{\infty} (\gamma^{t+1} V(s_{t+1}) - \gamma^t V(s_t)) = -V(s_0)$ $\square$

MC policy evaluation with Temporal Difference

$$V(s_t) := V(s_t) + \left[\left(\sum_{j=t}^{T-1} \gamma^{j-t} r_j \right) - V(s_t) \right]$$

$$= V(s_t) + \left[\left(\sum_{j=t}^{T-1} \underbrace{r_j + \gamma V(s_{j+1}) - V(s_j)}_{d_j} \right) \right]$$

- The term below is defined as the **Temporal Difference (TD)**

$$\delta_t = r_j + \gamma V(s_{j+1}) - V(s_j).$$

  This term is referred also as the **Bellman error.**
- $r_j + \gamma V(s_{j+1})$ and $V(s_j)$ estimate the same quantity. Treat the mismatch as error and minimize it, hence the name.
- TD makes it possible to do online updates

$$V(s_t) := V(s_t) + \delta_t, \qquad t = 1, \ldots, T-1.$$

Multi-step TD

One can also do bootstrapping with n-**step look-ahead Bellman targets**:

$$G_t^{(n)} := \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n V(s_{t+n})$$

This idea is plausible because $G_t^n = (T^\pi)^n(s)$, hence when viewed as an operator

$$\|G_t^{(n)}(V_1) - G_t^{(n)}(V_2)\|_\infty \leq \gamma^n \|V_1 - V_2\|_\infty$$

we see that it exponentially speeds up contraction! Define the $n-$step TD as

$$\delta_t^{(n)} := G_t^{(n)} - V(s_t).$$

We can relate this term to one-step TD via the telescoping lemma

$$\delta_t^{(n)} := \sum_{i=0}^{n-1} \gamma^i \delta_{t+i}.$$

λ-returns

The look-ahead horizon n is a hyper-parameter. Instead of being bothered with tuning it, we can take an exponential average of all possible values. The below term is called a λ-**return**

$$G_t^\lambda := (1-\lambda)\sum_{n=0}^{\infty}\lambda^n\delta_t^{(n+1)}.$$

Then we can do policy evaluation by

$$V(s_t) := V(s_t) + G_t^\lambda, \qquad t = 1,\dots,T-1.$$

There is a major bottleneck here. An update with a λ-return is possible only at an episode end and is not possible at all in a non-episodic environment. We will address this issue next by algebraic manipulations. The sum order swap identity below will be the key:

$$\sum_{i=0}^{\infty}\sum_{j=0}^{i} a_{ij} = \sum_{j=0}^{\infty}\sum_{i=j}^{\infty} a_{ij}$$

A nicer form of the λ-return

$$
\begin{aligned}
G_t^\lambda &:= (1-\lambda)\sum_{n=0}^{\infty}\lambda^n\delta_t^{(n+1)} \\
&= (1-\lambda)\sum_{n=0}^{\infty}\lambda^n\sum_{i=0}^{n}\gamma^i\delta_{t+i} && \text{telescoping lemma} \\
&= (1-\lambda)\sum_{i=0}^{\infty}\sum_{n=i}^{\infty}\lambda^n\gamma^i\delta_{t+i} && \text{sum order swap} \\
&= (1-\lambda)\sum_{i=0}^{\infty}\sum_{n=i}^{\infty}\lambda^{n-i}\lambda^i\gamma^i\delta_{t+i} && \text{split powers of } \lambda^n \\
&= \sum_{i=0}^{\infty}(\lambda\gamma)^i\delta_{t+i} && \text{as } \sum_{n=i}^{\infty}\lambda^{n-i} = 1/(1-\lambda)
\end{aligned}
$$

Value updates with λ-returns

Assume we update the value function for a whole episode with the $\lambda-$returns:

$$
\begin{aligned}
V &:= \sum_{t=0}^{\infty} \sum_{i=0}^{\infty} (\lambda\gamma)^i \delta_{t+i} \mathbb{1}_{s_t} && \text{forward view} \\
&= \sum_{t=0}^{\infty} \sum_{i=t}^{\infty} (\lambda\gamma)^{i-t} \delta_i \mathbb{1}_{s_t} \\
&= \sum_{i=0}^{\infty} \sum_{t=0}^{i} (\lambda\gamma)^{i-t} \delta_i \mathbb{1}_{s_t} \\
&= \sum_{i=0}^{\infty} \delta_i \sum_{t=0}^{i} (\lambda\gamma)^{i-t} \mathbb{1}_{s_t} && \text{backward view}
\end{aligned}
$$

where $\mathbb{1}_{s_t}$ is a one-hot vector with a 1 at the entry for state s_t.

Eligibility traces

How can we reuse intermediate computations in such a way that we get rid of one of the two sums? Build recursion. Define for $i > 1$:

$$\begin{aligned} e^{(i)} &:= \sum_{t=0}^{i} (\lambda\gamma)^{i-t} \mathbb{1}_{s_t} \\ &= \mathbb{1}_{s_i} + \sum_{t=0}^{i-1} (\lambda\gamma)^{i-t} \mathbb{1}_{s_t} \\ &= \mathbb{1}_{s_i} + \sum_{t=0}^{i-1} (\lambda\gamma)^{i-t+1-1} \mathbb{1}_{s_t} \\ &= \mathbb{1}_{s_i} + \lambda\gamma \sum_{t=0}^{i-1} (\lambda\gamma)^{i-1+t} \mathbb{1}_{s_t} \\ &= \mathbb{1}_{s_i} + \lambda\gamma e^{(i-1)}. \end{aligned}$$

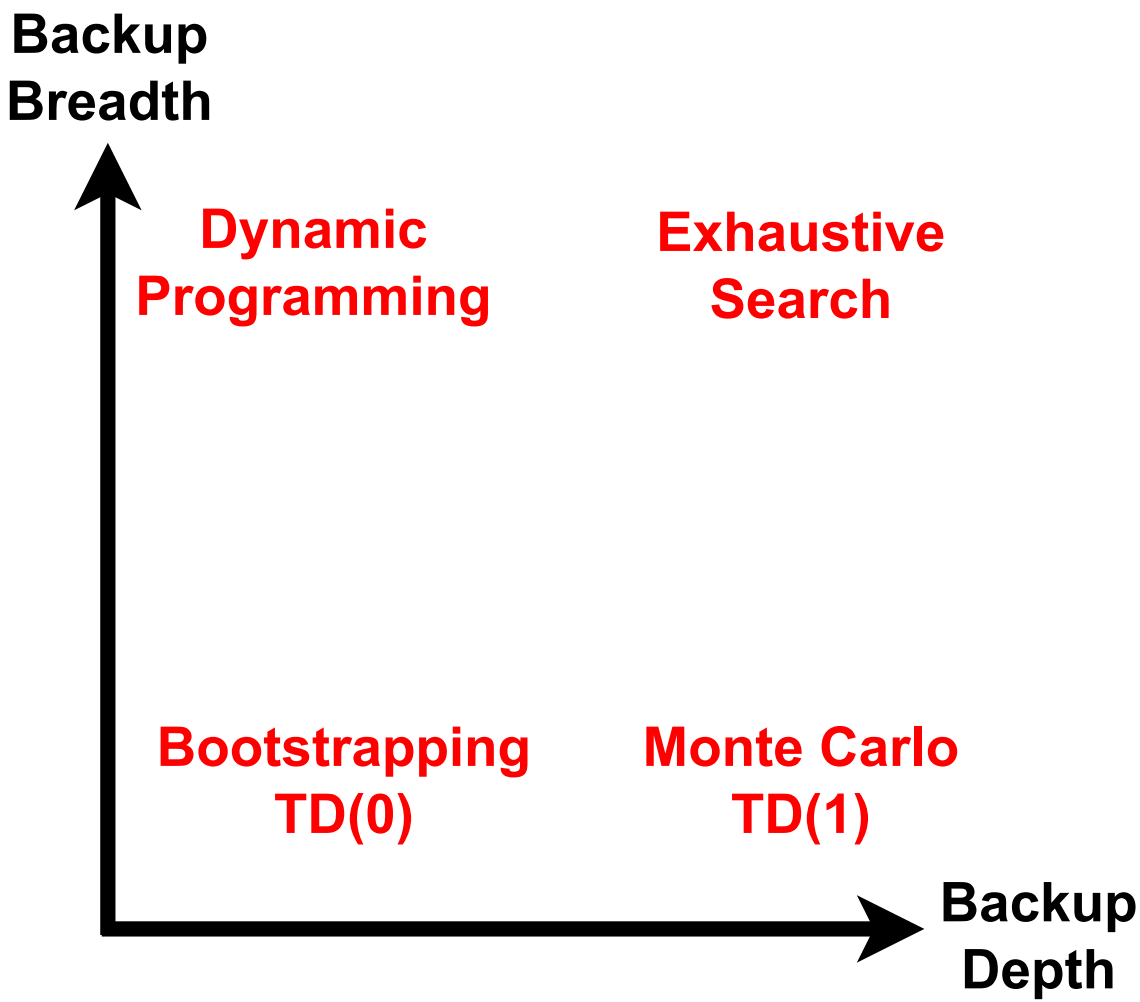
The vector $e^{(i)}$ is called an **eligibility trace**. It keeps track of the backward effect of each new state sample.

Policy evaluation with the TD(λ) algorithm

Input: λ, γ, π
$V := 0, e := 0$
repeat
  $s :=$ `env.reset()`
  repeat
    $a \sim \pi(\cdot|s)$
    $r, s' :=$ `env.step`(a)
    $e := \lambda\gamma e + \mathbb{1}_s$
    $\delta := r + \gamma V(s') - V(s)$
    $V(s) := V(s) + e(s)\delta$
    $s = s'$
  until episode end
until convergence

- $\lambda = 0 \Rightarrow$ 1-step TD, a.k.a. TD(0).
- $\lambda \to 1 \Rightarrow$ Monte Carlo, a.k.a. TD(1).

RL methods in the backup spectrum



Policy iteration as a learning algorithm

Extracting a greedy policy from a value function requires

$$\pi(s) \in \arg\max_a \left\{ \underbrace{r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^\pi(s')}_{T^\pi(V^\pi)(s,a)} \right\}.$$

but we don't have $r(s,a)$ and $p(s'|s,a)$. We can instead maintain an action-value function Q^π that satisfies $V^\pi(s) = Q^\pi(s, \pi(s))$, Then using the Bellman equation

$$V^\pi(s) = T^\pi(V^\pi)(s, \pi(s)) = Q^\pi(s, \pi(s))$$

we get a new action-value based recipe for calculating the greedy policy

$$\pi(s) \in \arg\max_a Q^\pi(s,a).$$

Optimistic policy iteration

Use the action-value based greedy policy calculation and finish policy evaluation before convergence (be optimistic about its precision).

$Q := 0$ and π arbitrary
repeat
  for $i = 1, \ldots, N$ **do** ▷ Policy evaluation for N episodes
    $s :=$ `env.reset()`
    repeat
      $r, s' :=$ `env.step`$(\pi(s))$
      $Q(s, a) := r + \gamma Q(s', \pi(s'))$
      $s' := s$
    until episode end
  end for
  $\pi(s) := \arg\max_a Q(s, a), \qquad \forall s \in \mathcal{S}$
until convergence

Optimism (and compute speed) at the extremes

Update the policy whenever the Q function changes.

$Q := 0$ and set π arbitrarily
repeat
  for $i = 1, \ldots, N$ **do** ▷ Policy evaluation for N episodes
    $s :=$ `env.reset()`
    repeat
      $r, s' :=$ `env.step`$(\pi(s))$
      $Q(s, a) := r + \gamma Q(s', \pi(s'))$
      $\pi(s) := \arg\max_a Q(s, a)$
      $s' := s$
    until episode end
  end for
until convergence

We can also compress the two steps in red into $Q(s', a') := r + \max_{a'} \gamma Q(s', a')$ which fast-forwards to the time when the policy update affects the Q calculation.

The exploration-exploitation dilemma once again!

This setup has some major differences from the vanilla form of policy iteration.

- The value functions are approximated based on MC samples. Hence the policy improvement step does not guarantee an improvement.
- The MC samples are conditioned on the policy used in the step $r, s' := \texttt{env.step}(\pi(s))$. Hence, the online learning pattern gives no guarantees for the coverage of the whole state-action space.

We need to address these issues by introducing:

- A behavior policy π_b that guarantees infinite visits to each state-action pair when run for an infinitely long time.
- A learning rate α that changes the Q-table entries only partially.

With these modifications, we arrive at the most practised algorithm of the RL field: **Q-Learning**.

Q-Learning

$Q := 0$
repeat
  $s :=$ `env.reset()`
  repeat
    $\pi_b(\cdot|s) :=$ `eps-greedy`$(Q(s,.),\varepsilon)$
    $a \sim \pi_b(\cdot|s)$
    $r, s' :=$ `env.step`(a)
    $\delta := r + \gamma \max_{a'} Q(s',a') - Q(s,a)$
    $Q(s,a) := Q(s,a) + \alpha\delta$
    $s' := s$
  until episode end
until convergence

The **behavior policy** $\pi_b(s)$ can be chosen ε-**greedily.** The function above called `eps-greedy` returns:

$$\pi(a|s) = \mathbb{1}_{a=a^*}\left(1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}|}\right) + \mathbb{1}_{a\neq a^*}\frac{\varepsilon}{|\mathcal{A}|} \qquad \text{for} \quad a^* = \arg\max_{a'} Q(s,a').$$

On-policy TD learning: Sarsa

$Q := 0$
repeat
  $s :=$ `env.reset()`
  repeat
    $\pi(\cdot|s) :=$ `eps-greedy`$(Q(s, .), \varepsilon)$
    $a \sim \pi(\cdot|s)$
    $r, s' :=$ `env.step`(a)
    $a' \sim \pi(\cdot|s')$
    $\delta := r + \gamma Q(s', a') - Q(s, a)$
    $Q(s, a) := Q(s, a) + \alpha\delta$
    $s' := s$
  until episode end
until convergence

Expected Sarsa

We can reduce the variance caused by $a' \sim \pi(\cdot|s')$ if we can compute the integral or calculate it analytically.

$Q := 0$
repeat
  $s :=$ `env.reset()`
  repeat
    $\pi(\cdot|s) :=$ `eps-greedy`$(Q(s,.),\varepsilon)$
    $a \sim \pi(\cdot|s)$
    $r, s' :=$ `env.step`(a)
    $\delta := r + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') Q(s',a') - Q(s,a)$
    $Q(s,a) := Q(s,a) + \alpha\delta$
    $s' := s$
  until episode end
until convergence

Sarsa(λ) algorithm

It is possible to do on-policy learning with λ-returns.

$Q := 0, e := 0$
repeat
  $s :=$ `env.reset()`
  repeat
    $\pi(\cdot|s) :=$ `eps-greedy`$(Q(s,.), \varepsilon)$
    $a \sim \pi(\cdot|s)$
    $r, s' :=$ `env.step`(a)
    $e := \lambda\gamma e + \mathbb{1}_{s,a}$
    $a' \sim \pi(\cdot|s')$
    $\delta := r + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') Q(s', a') - Q(s, a)$
    $Q(s, a) := Q(s, a) + \alpha\delta e(s, a)$
    $s' := s$
  until episode end
until convergence

Importance Sampling (IS)

Intuition: Sample from a different distribution from the one being integrated.

$$\mathbb{E}_{p(z)}\Big[f(z)\Big] = \sum_z f(z)p(z) = \sum_z f(z)\frac{p(z)}{q(z)}q(z)$$

then do Monte Carlo integration

$$\mathbb{E}_{p(z)}\Big[f(z)\Big] \approx \frac{1}{N}\sum_{i=1}^{N} f(z^{(i)}) \times \underbrace{\frac{p(z^{(i)})}{q(z^{(i)})}}_{w_i:\ \text{importance weight}}$$

for a set of $z^{(i)} \sim q(z)$. Importance weight can be arbitrarily large (unbounded), hence cause instability. Instead do **Weighted Importance Sampling (WIS)**:

$$\mathbb{E}_{p(z)}\Big[f(z)\Big] \approx \frac{(1/N)\sum_{i=1}^{N} w_i f(z^{(i)})}{(1/N)\sum_{i=1}^{N} w_i}$$

WIS is a biased estimator, but the bias converges to zero as $N \to \infty$.

Incremental (W)IS

For IS, use a straightforward extension of the running average formula

$$V(s_t) := V(s_t) + \frac{1}{i}\Big[w_i \widehat{V}_t^i - V(s_t)\Big].$$

For WIS define $\beta_i = \beta_{i-1} + w_i$ with $\beta_0 = 0$ and do

$$\begin{aligned}
\beta_i V_i(s_t) &= \widehat{V}_t^i w_i + \beta_{i-1} V_{i-1}(s_t) \\
&= \widehat{V}_t^i w_i + (\beta_i - w_i) V_{i-1}(s_t) \\
&= \widehat{V}_t^i w_i + \beta_i V_{i-1}(s_t) - w_i V_{i-1}(s_t) \\
\therefore \qquad V_i(s_t) &= \frac{\widehat{V}_t^i w_i + \beta_i V_{i-1}(s_t) - w_i V_{i-1}(s_t)}{\beta_i} \\
\therefore \qquad V_i(s_t) &:= V(s) + \frac{w_i}{\beta_i}\Big[\widehat{V}_t^i - V_{i-1}(s_t)\Big]
\end{aligned}$$

Off-policy Monte Carlo policy evaluation with IS

We may be interested in evaluating a policy π based on data collected by a behavior policy π_b. This way, we can explore the success of policies alternative to π_b and decide which policy to use next to collect data or act for optimal performance. We can then compute a first-visit Monte Carlo estimate for policy π:

$$\widehat{V}^{\pi}_{IS}(s_t) = \frac{1}{N}\sum_{i=1}^{N} \underbrace{\left(\prod_{j=t}^{T-1} \frac{\pi(a^i_j|s^i_j)}{\pi_b(a^i_j|s^i_j)} \frac{\cancel{p(s'^i_j|s^i_j, a^i_j)}}{\cancel{p(s'^i_j|s^i_j, a^i_j)}}\right)}_{:=w_i} \underbrace{\sum_{j=t}^{T-1} r^i_j}_{:=\widehat{V}^i_t}.$$

Note that we require

$$\pi(a|s) > 0 \Rightarrow \pi_b(a|s) > 0, \quad \forall (s,a)$$

which is called the **coverage** assumption. We can also apply the same to WIS

$$\widehat{V}^{\pi}_{WIS}(s_t) = \frac{\frac{1}{N}\sum_{i=1}^{N} w_i \widehat{V}^i_t}{\frac{1}{N}\sum_{i=1}^{N} w_i}.$$

Pros and cons of off-policy RL

- Off-policy methods incur higher variance, hence converge slower than on-policy methods.
- Off-policy methods have on-policy methods as their special case, hence they are more general and powerful.
- Off-policy methods can learn from a non-learning controller (e.g. a human expert), on-policy methods cannot.

RL in large and/or continuous state spaces

What if the state space is too large to allocate memory for every state or, the state description is very high-dimensional (e.g. a Go table, or the scene image)? A solution would be :

- to project states to a feature space $\phi(s)$. Do feature extraction, use kernels, learn an **embedding** separately or **end-to-end**, i.e. as part of the RL algorithm.
- to represent value function as a neural network $V_\theta(s) = w_2^T \sigma(W_1^T \phi(s))$ where $\theta = \{W_1, w_2\}$ and σ is an activation function (e.g. ReLU). Given a sample (s, r, s'), solve

$$\arg\min_\theta \left(\underbrace{r + \gamma V_\theta(s')}_{\text{target}} - \underbrace{V_\theta(s)}_{\text{prediction}} \right)^2$$

Since θ describe the value of all states, unlike the tabular approach, updating parameters for a single state affects the values of many other states! We **bootstrap** if V_θ is used both in prediction and target calculation.

MC versus TD on a single episode

Reorganize the sampled episode $s_1, a_1, \ldots, s_{T-1}, a_{T-1}, s_T$ as $\{(s_1, a_1, s_2), \ldots, (s_{T-1}, a_{T-1}, s_T)\}$, generate a labeled data set and do gradient-descent

- **MC:** $\mathcal{D} = \{(s_t, \sum_{k=t}^{T} r_k)\}$

$$\theta := \theta - \alpha \left[\sum_{k=t}^{T-1} \gamma^{k-t} r_k - V_\theta(s_t) \right] \nabla_\theta V_\theta(s_t)$$

- **TD**(0):

$$\mathcal{D} = \left\{ \begin{array}{c} (s_1, r_1 + \gamma V_\theta(s_2)) \\ (s_2, r_2 + \gamma V_\theta(s_3)) \\ \vdots \\ (s_T, r_t) \end{array} \right\}$$

$$\theta := \theta - \alpha \Big[r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t) \Big] \nabla_\theta V_\theta(s_t), \quad t = 1, \ldots, T-1$$

MC versus TD

- TD can learn *before* the final outcome is observed
  - TD learns online from every state transition
  - MC has to wait the episode end to calculate the return
- TD can learn *without* the final outcome
  - TD can learn from incomplete sequences (i.e. works in continuing environments)
  - MC can only learn from complete sequences (i.e. works only in episodic environments)
- TD exploits Markov property, MC does not
  - TD works better if the environment is Markov
  - MC can better handle non-stationarity

Semi-gradient off-policy control: Deep Q-learning

$\mathcal{D} := \emptyset$ ▷ Initialize the replay buffer
repeat
  $s :=$ `env.reset()`
  repeat
    $\pi(\cdot|s) :=$ `eps-greedy`(Q_θ, ε)
    $a \sim \pi(\cdot|s)$
    $r, s' ::=$ `env.step`(s, a)
    $\mathcal{D} := \mathcal{D} \cup (s, a, r, s')$,
    $\widetilde{\mathcal{D}} \overset{iid}{\sim} \mathcal{D}$ ▷ Sample minibatch from the replay buffer
    $\theta := \theta - \frac{\alpha}{|\widetilde{\mathcal{D}}|} \sum_{\widetilde{\mathcal{D}}} \left[r + \gamma \max_{a'} Q_\theta(s', a') - Q_\theta(s, a) \right] \nabla_\theta Q_\theta(s, a)$
    $s := s'$
  until episode end
until convergence

Replay buffer is a queue (FIFO container) of fixed size.

Semi-gradient on-policy control: Deep Sarsa

$\mathcal{D} := \emptyset$
repeat
  $s :=$ `env.reset()`
  repeat
    $\pi :=$ `eps-greedy`(Q_θ, ε)
    $a \sim \pi(\cdot|s)$
    $r, s' :=$ `env.step`(s, a)
    $a' \sim \pi(\cdot|s')$
    $\mathcal{D} := \mathcal{D} \cup (s, a, r, s', a')$,
    $\widetilde{\mathcal{D}} \overset{iid}{\sim} \mathcal{D}$
    $\theta := \theta - \frac{\alpha}{|\widetilde{\mathcal{D}}|} \sum_{\widetilde{\mathcal{D}}} \Big[r + \gamma Q_\theta(s', a') - Q_\theta(s, a) \Big] \nabla_\theta Q_\theta(s, a)$
    $s := s'$
  until episode end
until convergence

Remark: Incurs high variance. Requires $\gamma << 1$.

Deep Expected Sarsa

$\mathcal{D} := \emptyset$

repeat

  $s :=$ `env.reset()`

  repeat

    $\pi :=$ `eps-greedy`(Q_θ, ε)

    $a \sim \pi(\cdot|s)$

    $r, s' :=$ `env.step`(s, a)

    $\mathcal{D} := \mathcal{D} \cup (s, a, r, s'), \qquad \widetilde{\mathcal{D}} \overset{iid}{\sim} \mathcal{D}$

    $\theta := \theta - \frac{\alpha}{|\widetilde{\mathcal{D}}|} \sum_{\widetilde{\mathcal{D}}} \Big[r + \gamma \sum_{a'} \pi(a'|s') Q_\theta(s', a') - Q_\theta(s, a) \Big] \nabla_\theta Q_\theta(s, a)$

    $s := s'$

  until episode end

until convergence

Remark: Less variance, but more computation cost. No longer requires $\gamma << 1$.

N-step Deep Q-Learning (DQL)

$\mathcal{D} := \emptyset$

repeat

  $s :=$ `env.reset()`

  repeat

    $r_N := 0$

    for do $n = 0, \ldots, N-1$

      $\pi(\cdot|s) :=$ `eps-greedy`$(Q_\theta(s,\cdot), \varepsilon)$ and $a \sim \pi(\cdot|s)$

      if $n = 0$ **then** $s_0 := s, a_0 := a$

      $r, s' :=$ `env.step(s,a)`

      $r_N := r_N + \gamma^n r$

      $s = s'$

    end for

    $\mathcal{D} := \mathcal{D} \cup (s_0, a_0, r_N, s')$ and $\widetilde{\mathcal{D}} \overset{iid}{\sim} \mathcal{D}$

    $\theta := \theta - \frac{\alpha}{|\widetilde{\mathcal{D}}|} \sum_{\widetilde{\mathcal{D}}} \Big[r_N + \gamma^N \max_a Q_\theta(s', a') - Q_\theta(s, a) \Big] \nabla Q_\theta(s, a)$

  until episode end

until convergence