Reinforcement Learning

Aayush Borkar

July 28, 2024

Contents

1 Muti-armed Bandit Problem

1.1 Definitions

- **Arm** a: associated with a Bernoulli distribution with mean p_a.
- A: set of arms. The bandit instance.
- p^*: highest mean of the arms in A.
- **Reward** r^t: Reward obtained from the distribution of arm a.
- **History** h^t: $(a^0, r^0, a^1, r^1, \ldots, a^{t-1}, r^{t-1})$
- **Horizon** T: Total sampling budget.
- **Deterministic algorithm**: a mapping from the set of all histories to the set of all arms.
- **Randomised algorithm**: a mapping from the set of all histories to the set of all probability distributions over arms.

1.2 Bandits and Algorithms

An algorithm, bandit instance pair can generate many possible T-length histories. Observe that,

$$\mathbb{P}\{h^T\} = \prod_{t=0}^{T-1} \mathbb{P}\{a^t|h^t\}\mathbb{P}\{r^t|a^t\}, \text{ where}$$

$$\mathbb{P}\{a^t|h^t\} \text{ is decided by the algorithm}$$

$$\mathbb{P}\{r^t|a^t\} \text{ comes from the bandit instance}$$

The number of possible histories if the algorithm is deterministic and the reward distribution is bernoulli is $|A|^T$.

1.3 $\epsilon-$Greedy Algorithms

1.3.1 ϵG1

- If $t < \epsilon T$, sample an arm uniformly at random.
- At $\lfloor \epsilon T \rfloor$, identify a^{best}, an arm with the highest empirical mean.
- If $t > \epsilon T$, play a^{best}.

Algorithm ϵG1

$t \leftarrow 0$
$N_a \leftarrow 0$ for all $a \in A$
$S_a \leftarrow 0$ for all $a \in A$
$a^{\text{best}} \leftarrow 0$
while $t < T$ **do**
  if $t < \epsilon T$ **then**
    $a^t \sim \text{Uniform}(A)$
  else if $t = \lfloor \epsilon T \rfloor$ **then**
    $a^{\text{best}} \leftarrow \arg\max_{a \in A} \frac{S_a}{N_a}$
  else
    $a^t \leftarrow a^{\text{best}}$
  end if
  Observe r^t
  $N_{a^t} \leftarrow N_{a^t} + 1$
  $S_{a^t} \leftarrow S_{a^t} + r^t$
  $t \leftarrow t + 1$
end while

1.3.2 ϵG2

- If $t \leq \epsilon T$, sample an arm uniformly at random.
- If $t > \epsilon T$, play an arm with the highest empirical mean (upto t).

Algorithm ϵG2

$t \leftarrow 0$
$N_a \leftarrow 0$ for all $a \in A$
$S_a \leftarrow 0$ for all $a \in A$
while $t < T$ **do**
  if $t \leq \epsilon T$ **then**
    $a^t \sim \text{Uniform}(A)$
  else
    $a^t \leftarrow \arg\max_{a \in A} \frac{S_a}{N_a}$
  end if
  Observe r^t
  $N_{a^t} \leftarrow N_{a^t} + 1$
  $S_{a^t} \leftarrow S_{a^t} + r^t$
  $t \leftarrow t + 1$
end while

1.3.3 ϵG3

- With probability ϵ, sample an arm uniformly at random.
- With probability $1 - \epsilon$, play an arm with the highest empirical mean (upto t).

Algorithm ϵG3

$t \leftarrow 0$
$N_a \leftarrow 0$ for all $a \in A$
$S_a \leftarrow 0$ for all $a \in A$
while $t < T$ **do**
  $p \sim \text{Uniform}(0, 1)$
  if $p < \epsilon$ **then**
    $a^t \sim \text{Uniform}(A)$
  else
    $a^t \leftarrow \arg\max_{a \in A} \frac{S_a}{N_a}$
  end if
  Observe r^t
  $N_{a^t} \leftarrow N_{a^t} + 1$
  $S_{a^t} \leftarrow S_{a^t} + r^t$
  $t \leftarrow t + 1$
end while

1.4 Regret

- The maximum achievable expected reward in T steps is Tp^*.
- The actual expected **reward** for an algorithm is $\sum_{t=0}^{T-1} \mathbb{E}[r^t]$.
- The (expected cumulative) **regret** of the algorithm for horizon T is the difference

$$R_T = Tp^* - \sum_{t=0}^{T-1} \mathbb{E}[r^t]$$

- An algorithm with a sub-linear regret (i.e. $R_T = o(T)$) is desirable.
- Let **exploit(T)** denote the number of pulls of the best arm till time T.

1.4.1 Regret Analysis for ϵG1 and ϵG2

$$\begin{aligned} R_T &= Tp^* - \sum_{t=0}^{T-1} \mathbb{E}[r^t] = Tp^* - \sum_{t=0}^{\epsilon T-1} \mathbb{E}[r^t] - \sum_{t=\epsilon T}^{T-1} \mathbb{E}[r^t] \\ &= Tp^* - \epsilon T p_{\text{avg}} - \sum_{t=\epsilon T}^{T-1} \mathbb{E}[r^t] \geq Tp^* - \epsilon T p_{\text{avg}} - (T - \epsilon T)p^* \\ &= \epsilon T(p^* - p_{\text{avg}}) = \Omega(T) \end{aligned}$$

1.4.2 Regret Analysis for ϵG3

$$\begin{aligned} R_T &= Tp^* - \sum_{t=0}^{T-1} \mathbb{E}[r^t] \geq Tp^* - \sum_{t=0}^{T-1} (\epsilon p_{\text{avg}} + (1-\epsilon)p^*) \\ &= \epsilon T(p^* - p_{\text{avg}}) = \Omega(T) \end{aligned}$$

1.4.3 Conditions for Sub-linear Regret

There are two conditions for sub-linear regret:

1. **Infinite exploration**: In the limit as $T \to \infty$, the algorithm must explore all arms infinitely often.
2. **Greed in the Limit**: We need

$$\lim_{T \to \infty} \frac{\mathbb{E}[\text{exploit}(T)]}{T} = 1$$

1.4.4 Some modified ϵ-Greedy algorithms with sub-linear regret

- ϵ_T-first with $\epsilon_T = \frac{1}{\sqrt{T}}$. Explote for $\epsilon_T \cdot T = \sqrt{T}$ pulls, thereafter exploit.
- ϵ_t-greedy with $\epsilon_t = \frac{1}{t+1}$. On the t^{th} pull, explore with probability ϵ_t, exploit with probability $1 - \epsilon_t$.

Both the above algorithms satisy the conditions for sub-linear regret.

1.5 Upper Confidence Bound (UCB) Algorithms

At time t, for every arm a, we have an upper confidence bound ucb_a^t, which we define as:

$$\text{ucb}_a^t = \hat{p}_a^t + \sqrt{\frac{2 \log t}{u_a^t}}$$

$\hat{p}_t^a$ is the empirical mean of rewards from arm a upto time t, and u_a^t is the number of times arm a has been played upto time t. Pull an arm a for whcich ucb_a^t is maximised.

UCB Algorithm

$t \leftarrow 0$
$\hat{p}_a \leftarrow 0$ for all $a \in A$
$u_a \leftarrow 0$ for all $a \in A$
while $t < T$ **do**
  $a^t \leftarrow \arg\max_{a \in A} \left(\hat{p}_a^t + \sqrt{\frac{2 \log t}{u_a^t}} \right)$
  Observe r^t
  $\hat{p}_{a^t} \leftarrow \frac{u_{a^t} \hat{p}_{a^t} + r^t}{u_{a^t} + 1}$
  $u_{a^t} \leftarrow u_{a^t} + 1$
  $t \leftarrow t + 1$
end while

1.5.1 KL-UCB Algorithm

The $KL(x, y)$ divergence is given by:

$$KL(x, y) = x \log \frac{x}{y} + (1 - x) \log \frac{1 - x}{1 - y}$$

We define $ucb - kl_a^t$ as the solution $q \in [\hat{p}_a^t, 1]$ to the equation:

$$KL(\hat{p}_a^t, q) = \frac{\log t + c \log (\log t)}{u_a^t}$$

Since $KL(\hat{p}_a^t, q)$ is a monotonically increasing function in q, we can find the solution using binary search.

Calculating $ucb - kl_a^t$

$l \leftarrow 0$
$r \leftarrow 1$
while $r - l > \epsilon$ **do**
  $q \leftarrow \frac{l+r}{2}$
  if $KL(\hat{p}_a^t, q) > \frac{\log t + c \log(\log t)}{u_a^t}$ **then**
    $r \leftarrow q$
  else
    $l \leftarrow q$
  end if
end while

1.6 Thompson Sampling

At time t, arm a has s_a^t successes and f_a^t failures. We can model the probability of success of arm a as a beta distribution $\text{Beta}(s_a^t + 1, f_a^t + 1)$. For this distribution:

$$\text{Mean} = \frac{s_a^t + 1}{s_a^t + f_a^t + 2}$$
$$\text{Variance} = \frac{(s_a^t + 1)(f_a^t + 1)}{(s_a^t + f_a^t + 2)^2(s_a^t + f_a^t + 3)}$$

Thompson Sampling

$t \leftarrow 0$
$s_a \leftarrow 0$ for all $a \in A$
$f_a \leftarrow 0$ for all $a \in A$
while $t < T$ **do**
  Sample $\theta_a^t \sim \text{Beta}(s_a^t + 1, f_a^t + 1)$ for all $a \in A$
  $a^t \leftarrow \arg\max_{a \in A} \theta_a^t$
  Observe r^t
  if $r^t = 1$ **then**
    $s_{a^t} \leftarrow s_{a^t} + 1$
  else
    $f_{a^t} \leftarrow f_{a^t} + 1$
  end if
end while

Thompson Sampling achieves **optimal regret** and is **excellent** in practice.

1.7 Hoeffding's Inequality

Let X be a random variable bounded in $[0, 1]$. And, let $x_1, x_2, \ldots, x_u$ be samples of X. Then, for any fixed $\epsilon > 0$, we have

$$\mathbb{P}\{\bar{x} \geq \mu + \epsilon\} \leq e^{-2u\epsilon^2}$$
$$\mathbb{P}\{\bar{x} \leq \mu - \epsilon\} \leq e^{-2u\epsilon^2}$$

1.7.1 Hoeffding's Inequality for arbitrary bounded random variable

Suppose X is a random variable bounded in $[a, b]$. We consider $Y = \frac{X-a}{b-a}$

$$\bar{y} = \frac{\bar{x}}{b-a} - \frac{a}{b-a}$$
$$\bar{x} = (b-a)\bar{y} + a$$
$$\mathbb{P}\{\bar{x} \geq \mu + \epsilon\} = \mathbb{P}\{\bar{y} \geq \frac{\mu - a}{b-a} + \frac{\epsilon}{b-a}\} \leq e^{-\frac{2u\epsilon^2}{(b-a)^2}}$$
$$\mathbb{P}\{\bar{x} \leq \mu - \epsilon\} = \mathbb{P}\{\bar{y} \leq \frac{\mu - a}{b-a} - \frac{\epsilon}{b-a}\} \leq e^{-\frac{2u\epsilon^2}{(b-a)^2}}$$

1.7.2 KL inequality

$$\mathbb{P}\{\bar{x} \geq \mu + \epsilon\} \leq e^{-uKL(\mu+\epsilon,\mu)} \, \forall \epsilon \in [0, 1-\mu]$$
$$\mathbb{P}\{\bar{x} \leq \mu - \epsilon\} \leq e^{-uKL(\mu-\epsilon,\mu)} \, \forall \epsilon \in [0, \mu]$$

1.7.3 Chernoff Bounds

The KL inequality can be used to derive Chernoff bounds. For $p, q \in [0, 1]$:

$$KL(p, q) \geq 2(p-q)^2$$
$$\implies e^{-uKL(p,q)} \leq e^{-2u(p-q)^2}$$

2 Markov Decision Processes (MDPs)

2.1 Definitions

A **Markov Decision Process (MDP)** is a tuple (S, A, T, R, γ) where:

- **S**: a finite set of states. $|S| = n$.
- **A**: a finite set of actions. $|A| = k$.
- **T**: a transition function $T : S \times A \times S \to [0, 1]$. $T(s, a, s')$ is the probability of transitioning to state s' on taking action a in state s. $\sum_{s'} T(s, a, s') = 1$.
- **R**: a reward function $R : S \times A \times S \to \mathbb{R}$. $R(s, a, s')$ is the reward on transitioning to state s' on taking action a in state s. The reward is immediate.
- γ: a discount factor $\gamma \in [0, 1)$.

A **Policy** $\pi : S \to A$ is a mapping from states to actions. Π denotes the set of all policies. $|\Pi| = k^n$. The **value function** $V^\pi : S \to \mathbb{R}$ is the expected discounted cumulative reward on following policy π from state s.

$$V^\pi(s) = \mathbb{E}_\pi[r^0 + \gamma r^1 + \gamma^2 r^2 + \gamma^3 r^3 + \ldots | s^0 = s]$$

2.2 MDP Planning

Given an MDP, the goal is to find the optimal policy π^* that maximises the value function. Every MDP is guranteed to have a deterministic, markovian, stationary optimal policy. An MDP can have more than one optimal policy, but all optimal policies have the same value function.

2.3 Bellman Equations

For $\pi \in \Pi$, $s \in S$, the **Bellman Equation** is given by:

$$V^{\pi}(s) = \sum_{s' \in S} T(s, \pi(s), s') \left\{ R(s, \pi(s), s') + \gamma V^{\pi}(s') \right\}$$

These are n linear equations in n variables. They are guranteed to have a unique solution if $\gamma < 1$, and can be solved using linear algebra.

2.4 Banach's Fixed Point Theorem

A **Contraction Mapping** is a function $Z : X \to X$ such that:

$$||Zv - Zu|| \leq l||v - u|| \ \forall u \in X, \forall v \in X$$

A point $x^* \in X$ is called a **Fixed Point** if $Zx^* = x^*$.

Bananach's Fixed Point Theorem states that if Z is a contraction mapping on a complete metric space X, with contraction factor $l \in [0, 1)$, then:

1. Z has a unique fixed point $x^* \in X$.
2. For any $x \in X$, the sequence $x_{n+1} = Zx_n$ converges to x^*. Or equivalently, $||Z^m x - x^*|| \leq l^m |x - x^*||$ for $m \geq 0$.

2.5 Bellman Optimality Equations

The **Bellman Optimality Operator** $B^* : \mathbb{R}^n \to \mathbb{R}^n$ for an MDP is given by:

$$(B^*(F))(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s') \left\{ R(s, a, s') + \gamma F(s') \right\}$$

Note that B^* is a contraction mapping. Banach's Fixed Point Theorem implies that B^* has a unique fixed point V^*. This fixed point is the optimal value function, and is given by the **Bellman Optimality Equation**:

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s') \left\{ R(s, a, s') + \gamma V^*(s') \right\}$$

These are n equations in n variables, but are non-linear.

2.6 Value Iteration

Iteratively apply the Bellman Optimality Operator to find the optimal value function.

Value Iteration

$V(s) \leftarrow 0$ for all $s \in S$
while not converged **do**
  for $s \in S$ **do**
    $V'(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s, a, s') \{R(s, a, s') + \gamma V(s')\}$
  end for
  $V(s) \leftarrow V'(s)$ for all $s \in S$
end while

2.7 Linear Programming for Bellman Optimality Equations

Although the Bellman Optimality Equations are non-linear, they can be converted to a linear program. For $s \in S, a \in A$:

$$V(s) \geq \sum_{s' \in S} T(s, a, s') \left\{R(s, a, s') + \gamma V(s')\right\}$$

We get nk linear constraints in n variables, and V^* is in the feasible region of this LP. So, we need more constraints to get the optimal value function. Since, B^* operator preserves the order, and since $V \succeq B^*(V)$ for all V in the feasible set, we can observe that:

$$\begin{aligned} V &\succeq B^*(V) \\ \implies B^*(V) &\succeq (B^*)^2(V) \\ \implies (B^*)^2(V) &\succeq (B^*)^3(V) \\ &\vdots \\ V \succeq \lim_{l \to \infty} (B^*)^l(V) &= V^* \end{aligned}$$

We get an additional constraint, i.e., to **minimise** $V(s)$ for all $s \in S$. These can be solved using any LP solver.

2.8 Policy Iteration

2.8.1 Action Value Function

We define $Q^\pi : S \times A \to \mathbb{R}$ as the **Action Value Function** for policy π as the expected discounted cumulative reward on taking action a in state s and then following policy π. It is given by:

$$Q^\pi(s, a) = \mathbb{E}[r^0 + \gamma r^1 + \gamma^2 r^2 + \ldots | s^0 = s; a^0 = a; a^t = \pi(s^t) \text{ for } t \geq 1]$$

$$Q^\pi(s, a) = \sum_{s' \in S} T(s, a, s') \left\{R(s, a, s') + \gamma V^\pi(s')\right\}$$

2.8.2 Policy Improvement

For $\pi \in \Pi, s \in S$, we define two quantities, namely **IA** and **IS** as below.

$$IA(\pi, s) = \{a \in A : Q^\pi(s,a) > V^\pi(s)\} \; and \; IS(\pi) = \{s \in S : |IA(\pi, s) \geq 1|\}$$

Policy Improvement Theorem states that for any $\pi \in \Pi$, if $IS(\pi) = S$, then π is an optimal policy. Else, if π' is obtained by policy improvement on π, then $\pi' \succ \pi$.

2.8.3 Policy Iteration Algorithm

Policy Iteration

$\pi \leftarrow$ random policy
while π has improvable states **do**
  $\pi' \leftarrow$ policy improvement on π
  $\pi \leftarrow \pi'$
end while

2.9 Howard's Policy Iteration for 2-action MDPs

Switch actions in every improvable state. If π has m improvable states and π' is obtained from pi by Howard's PI, then there exist m policies π'' such that $\pi' \succeq \pi'' \succ \pi$. If we take $m^* = \frac{n}{3}$, then the number of policies with m^* or more improvable states visited is at most $\frac{2^n}{m} = 3\frac{2^n}{n}$. The number of policies with fewer than m^* improvable states visited is at most $\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{m^*-1} \leq 3\frac{2^n}{n}$. Therefore, the number of iterations taken by Howaed's PI is at most $O(\frac{2^n}{n})$.

2.10 Batch Switching Policy Iteration

Howard's PI takes at most **3 iterations** for 2-action MDPs. For MDPs with more than 2 actions, we can use **Batch Switching Policy Iteration**. In this, we partition the set of actions into batches of size k ($k = 2$ here), and switch actions within a batch. This reduces the number of policies visited in each iteration. In each iteration, we improve the left-most batch. An illustration is given below.

π_4	0	1	1	0	1	1	1	1	1	0
π_3	0	1	1	0	1	1	1	**0**	1	0
π_2	0	1	1	0	**0**	**0**	1	0	1	0
π_1	0	1	1	0	0	0	1	0	1	0

The complexity of this algorithm can be calculated by the recurrence $T(n) \leq 3 \times T(n-2)$. Hence, the complexity is $O(3^{n/2})$.

3 Reinforcement Learning

3.1 Common Problems

A **learning algorithm** L is a mapping from the set of all histories to the set of all probability distributions over arms. Let h^t denote the history of the first t rounds, and s^t denote the state at time t. The **Control Problem** is to find a learning algorithm L that gives the optimal cumulative reward for all states. In other words, can we construct an algorithm L such that,

$$\lim_{H\to\infty} \frac{1}{H} \left(\sum_{t=0}^{H-1} \mathbb{P}\{a^t \sim L(h^t) \text{ is an optimal action for } s^t\} \right) = 1$$

The **Prediction Problem** asks whether we construct an algorithm L such that,

$$\lim_{t\to\infty} \hat{V}^t = V^\pi$$

3.2 Monte Carlo Methods

We define two functions before proceeding. Let $\mathbf{1}(s,i,j)$ be 1 if s is visited at least j times, and 0 otherwise. Let $G(s,i,j)$ be the discounted long-term reward obtained, starting from jth visit of state s on episode i. Let **times**(s,i) be the number of times state s is visited in episode i.

3.2.1 First-Visit Monte Carlo

Average the G-values obtained from the first visit to each state.

$$\hat{V}^N_{\text{First-visit}}(s) = \frac{\sum_{i=1}^N G(s,i,1)}{\sum_{i=1}^N \mathbf{1}(s,i,1)}$$

3.2.2 Every-Visit Monte Carlo

Average the G-values obtained from every visit to each state.

$$\hat{V}^N_{\text{Every-visit}}(s) = \frac{\sum_{i=1}^N \sum_{j=1}^\infty G(s,i,j)}{\sum_{i=1}^N \sum_{j=1}^\infty \mathbf{1}(s,i,j)}$$

3.2.3 Second-Visit Monte Carlo

Average the G-values obtained from the second visit to each state.

$$\hat{V}^N_{\text{Second-visit}}(s) = \frac{\sum_{i=1}^N G(s,i,2)}{\sum_{i=1}^N \mathbf{1}(s,i,2)}$$

3.2.4 Last-Visit Monte Carlo

Average the G-values obtained from the last visit to each state.

$$\hat{V}^N_{\text{Last-visit}}(s) = \frac{\sum_{i=1}^N G(s,i,\text{times}(s,i))}{\sum_{i=1}^N \mathbf{1}(s,i,\text{times}(s,i))}$$

Observe that $\lim_{N\to\infty} \hat{V}^N_{\text{nth-visit}} = V^\pi$ for $n = 1, 2$,every but not for last.

3.2.5 Online Implementation of Monte Carlo

Assume that every episode starts at the same state s_0. Then, first visit monte carlo simply becomes $\frac{1}{t}\sum_{i=1}^t G(s,i,1)$. We can implement online Monte Carlo as follows.

$$\begin{aligned}
\hat{V}^t(s) &= \frac{1}{t}\sum_{i=1}^{t} G(s,i,1) \\
&= \frac{1}{t}\left(\sum_{i=1}^{t-1} G(s,i,1) + G(s,t,1)\right) \\
&= \frac{1}{t}\left((t-1)\hat{V}^{t-1}(s) + G(s,t,1)\right) \\
&= (1-\alpha_t)\hat{V}^{t-1}(s) + \alpha_t G(s,t,1) \text{ for } \alpha_t = \frac{1}{t}
\end{aligned}$$

3.3 Common Estimates

3.3.1 The Problem

Let $\beta = p_1, p_2, p_3, \ldots$ be the vector of actual probabilities of getting a 1-reward on the arms. Let kth arm be played n_k times and m_k times a 1-reward is obtained. Our goal is to find an estimate of β.

3.3.2 Least Squares Estimate

We can calculate $p'_1, p'_2, p'_3, \ldots$ as the observed probabilites of getting a 1-reward for each arm (i.e., $p'_k = \frac{m_k}{n_k}$). The **Squares Estimate** function is given by,

$$SE(\beta) = \sum_i (p_i - p'_i)^2$$

We obtain β by minimising the function $SE(\beta)$.

3.3.3 Maximum Likelihood Estimate

The **Likelihood** function is given by,

$$L(\beta) = \prod_i p_i^{m_i}(1-p_i)^{n_i-m_i}$$

We obtain β by maximising the function $L(\beta)$.

3.4 Temporal Difference Learning: TD(0)

TD(0) Algorithm

$\pi \leftarrow$ arbitrary policy
$\hat{V}_0 \leftarrow$ arbitrary initialisation
for $t = 0, 1, 2, \ldots$ **do**
  Take action $a^t \sim \pi(s^t)$
  Obtain r^t
  $\hat{V}^{t+1}(s^t) \leftarrow \hat{V}^t(s^t) + \alpha_t \left(r^t + \gamma \hat{V}^t(s^{t+1}) - \hat{V}^t(s^t)\right)$
  for all $s \in S \setminus \{s^t\}$ **do**
    $\hat{V}^{t+1}(s) \leftarrow \hat{V}^t(s)$
  end for
end for

First-visit MC, Every-visit MC, and Batch TD(0), all converge to V^π. But, for small N, they are very different. Generally, a middle path works the best.

3.5 Multi-step Returns

For $t \geq 0, n \geq 1$, the **n-step return** $G_{t:t+n}$ is defined as,

$$G_{t:t+n} = r^t + \gamma r^{t+1} + \gamma^2 r^{t+2} + \cdots + \gamma^{n-1} r^{t+n-1} + \gamma^n V^{t+n-1}(s^{t+n})$$

n-step TD makes update of the form:

$$V^{t+n}(s^t) \leftarrow V^{t+n-1}(s^t) + \alpha_t \left(G_{t:t+n} - V^{t+n-1}(s^t)\right)$$

Observe that this is somewhat between first-visit MC and TD(0). Any convex combination of $G_{i:j}$ can be used in place of $G_{t:t+n}$ (called the **Target**).

3.5.1 λ-return

A particular convex combination of n-step returns is called the **λ-return** ($\lambda \in [0, 1]$).

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

Observe that, $G_t^0 = G_{t:t+1}$, corresponds to full bootstrapping and $G_t^1 = G_{t:\infty}$, corresponds to MC estimate. The transition from s^t to s^{t+1} contributes to the λ-returns of $s^0, s^1, ..., s^t$, with a weight of λ.

3.6 Backward View TD Learning: TD(λ)

In the below pseudocode, $z : S \to \mathbb{R}$ is a vector of eligibility traces.

TD(λ) Algorithm

$\hat{V}_0 \leftarrow$ arbitrary initialisation
for each episode **do**
  $z \leftarrow 0$
  for each step **do**
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $z \leftarrow z + 1$
    for all $s \in S$ **do**
      $V(s) \leftarrow V(s) + \alpha\delta z$
      $z \leftarrow \gamma\lambda z$
    end for
  end for
end for

$\lambda = 0$ corresponds to full bootstrapping, and $\lambda = 1$ corresponds to MC estimate.

3.7 Q-Learning

It is an **off-policy model-free value-based** method, which uses a TD approach. The goal is to get the optimal Q-function. We follow ϵ-greedy policy to sample an action at every step.

Q-Learning Algorithm

for $i \leftarrow 1$ to num_episodes **do**
  $\epsilon \leftarrow \epsilon_i$
  Observe S_0
  $t \leftarrow 0$
  while s^t is not terminal **do**
    Sample a^t using ϵ-greedy policy
    Observe r^{t+1}, s^{t+1}
    $Q^{t+1}\left(s^t, a^t\right) \leftarrow Q^t\left(s^t, a^t\right) + \alpha\left\{r^t + \gamma \max_a Q^t\left(s^{t+1}, a\right) - Q^t\left(s^t, a^t\right)\right\}$
    $t \leftarrow t + 1$
  end while
end for

3.7.1 State Action Reward State Action (SARSA)

A general update rule for TD methods is,

$$Q^{t+1}\left(s^t, a^t\right) \leftarrow Q^t\left(s^t, a^t\right) + \alpha_{t+1}\left\{\textbf{Target} - Q^t\left(s^t, a^t\right)\right\}$$

SARSA and Expected SARSA are two **on-policy** methods.
For SARSA, $\textbf{Target} = r^t + \gamma Q^t\left(s^{t+1}, a^{t+1}\right)$.
For Expected SARSA, $\textbf{Target} = r^t + \gamma \sum_a \pi^t\left(s^{t+1}, a\right) Q^t\left(s^{t+1}, a\right)$.

When policy is time-varying, all the three methods QL, SARSA, and Expected SARSA converge to the optimal Q-function. But when the policy is time-invariant, then only QL converges to the optimal Q-function.

3.7.2 Off-policy vs On-policy

- **Off-policy** methods use a different policy for acting (inference) and updating (training).
- **On-policy** methods use the same policy for acting and updating.

3.7.3 Model-based vs Model-free

- **Model-based** methods (like value iteration) need to know a model of the environment (the transition and reward functions etc.).
- **Model-free** methods (like QL, SARSA) do not need to know a model of the environment.

3.8 Deep Q-Learning

In DQN, we use a neural network to obtain the optimal Q-function. The loss function on which the network is trained (using Stochastic Gradient Descent or Adam) is,

$$r^t + \gamma \max_{a \in A} Q^t\left(s^{t+1}, a^t\right) - Q^t\left(s^t, a^t\right)$$

3.8.1 Experience Replay

We create a **replay memory** to make more efficient use of the experiences during the training.

DQN using Experience Replay

Replay buffer $D \leftarrow$ arbitrary initialisation
$Q \leftarrow$ arbitrary initialisation
for episode$\leftarrow$ 1 to M **do**
  Sequence $s_1 \leftarrow \{x_1\}$
  for $t \leftarrow 1$ to T **do**
    Sample a^t using ϵ-greedy policy
    Observe r^{t+1}, s^{t+1}
    Store $\left(s^t, a^t, r^t, s^{t+1}\right)$ in D
    Sample random minibatch of transitions $\left(s^t, a^t, r^t, s^{t+1}\right)$ from D
    Perform gradient-descent on $\left(r^t + \gamma \max_a Q^t\left(s^{t+1}, a\right) - Q^t\left(s^t, a^t\right)\right)$
    Update Q using the loss
  end for
end for

3.9 Prediction with Linear Approximation

We can choose to approximate our value function by dot product of two vectors, $x : S \rightarrow \mathbb{R}^d$ (d-dimensional feature vector) and $w \in \mathbb{R}^d$ (weight vector).

$$\hat{V}(w, s) = x(s) \cdot w$$

Our goal is that $V^\pi(s)$ and $\hat{V}(w, s)$ be as close as possible. So, we define a loss function,

$$MSVE(w) = \frac{1}{2} \sum_{s \in S} \mu^\pi(s) \left\{V^\pi(s) - \hat{(V)}(w, s)\right\}^2$$

Here, $\mu^\pi : S \rightarrow [0, 1]$ is the stationary distribution of π. And the optimal w is obtained by minimising the loss function.

$$w^* = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}}\, MSVE(w)$$

We can use stochastic gradient descent to find the optimal w. But we don't know $V^\pi\left(s^t\right)$, so we can use n-step returns instead by performing the following update.

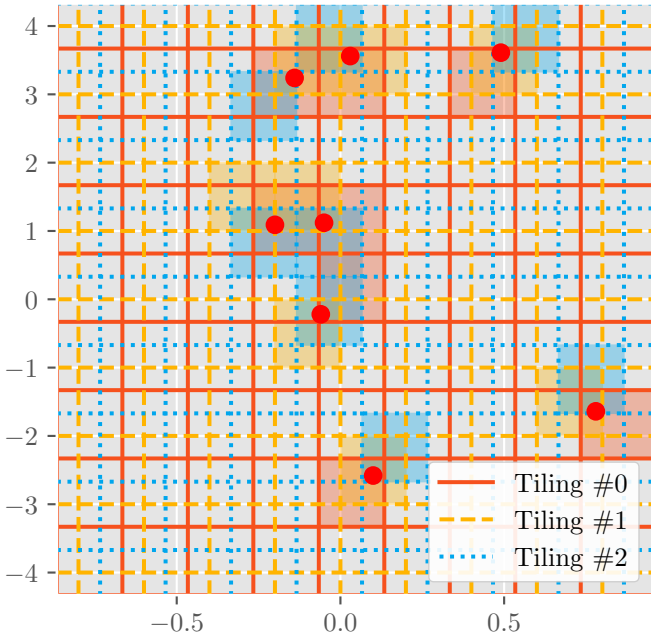
$$w^{t+1} \leftarrow w^t + \alpha_{t+1} \left\{G_t^\lambda - \hat{V}\left(w^t, s^t\right)\right\} \nabla_w \hat{V}\left(w^t, s^t\right)$$

Linear TD(λ) Algorithm

$\hat{V}_0 \leftarrow$ arbitrary initialisation
for each episode **do**
  $z \leftarrow 0$
  for each step **do**
    Take action a, obtain r, s'
    $\delta \leftarrow r + \gamma \hat{V}(w, s') - \hat{V}(w, s)$
    $z \leftarrow \gamma \lambda z + \nabla_w \hat{V}(w, s)$
    $w \leftarrow w + \alpha \delta z$
    $s \leftarrow s'$
  end for
end for

3.10 Tile Coding

Tile coding is a way to discretise a continuous action space. The idea is to create several overlapping grids (**tilings**) and assign a unique tile to each state. Each tile has an associated weight, and the value of a state is the sum of the weights of the tiles associated with that state. Below is an example of the same.



Samples: [(0.03, 3.56), (0.1, -2.58), (0.49, 3.61), (-0.2, 1.09), (-0.05, 1.12), (-0.14, 3.24),

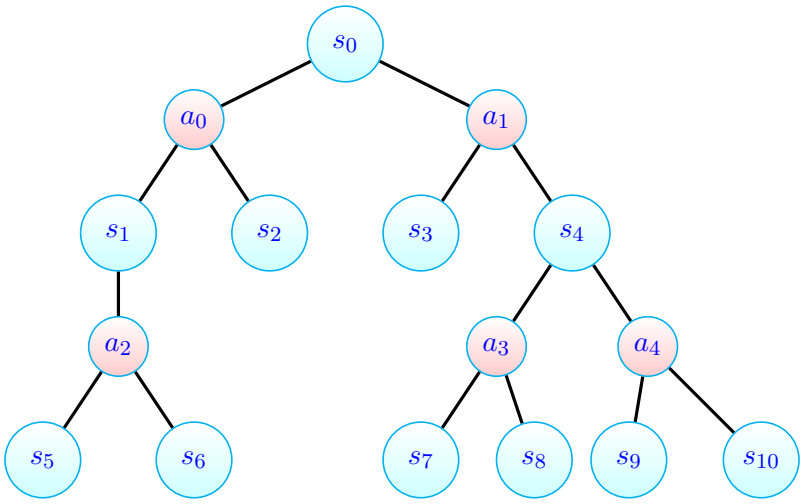
(0.78, -1.64), (-0.06, -0.22)]
Encoded samples: [[(5, 8), (5, 8), (4, 8)], [(5, 2), (5, 2), (5, 2)], [(7, 8), (7, 8), (7, 8)], [(4, 6), (3, 6), (3, 5)], [(5, 6), (4, 6), (4, 5)], [(4, 8), (4, 8), (3, 7)], [(9, 3), (8, 3), (8, 3)], [(5, 5), (4, 4), (4, 4)]]

3.11 ExpectiMax Algorithm

MDPs can be represented as trees where the nodes represent states and the edges represent actions. These are called **ExpectiMax trees**. V and Q values are computed using a bottom-up approach.

$$V^d(s) \leftarrow \max_{a \in A} Q^{d+1}(s,a)$$
$$Q^d(s,a) \leftarrow \sum_{s' \in S} T(s,a,s') \left\{ R(s,a,s') + \gamma V^d(s') \right\}$$

For sufficient accuracy, episode length must be $h = \Theta\left(\frac{1}{1-\gamma}\right)$ And the number of nodes in the tree is $\Theta\left(b^h\right)$ where b is branching factor. This becomes computationally very expensive.



3.12 Monte Carlo Tree Search (MCTS)

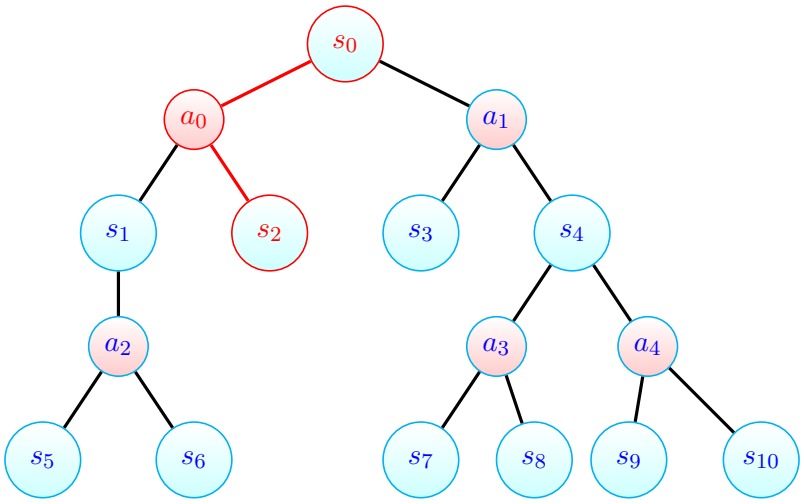
In this, a search tree is built node-by-node. The algorithm has four steps.

3.12.1 Selection

Starting at root node R, recursively select child nodes until a leaf node L is reached. Analogous to the MAB problem, we can use Upper Confidence Bounds (UCB) to select the child nodes.

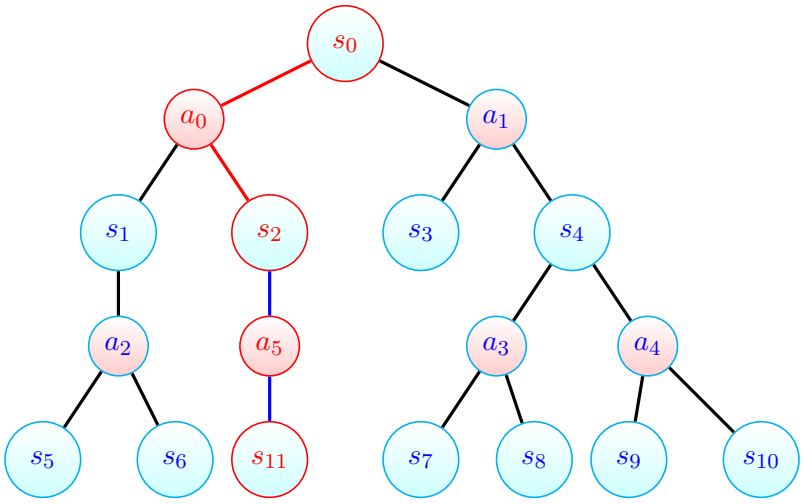
$$\text{ucb}(s,a) = Q(s,a) + C_p \sqrt{\frac{\ln t}{\text{visits}(s,a)}}$$

This is done in order to balance exploration with exploitation. This variant of MCTS is also called **UCT** (Upper Confidence Bounds for Trees). C_p in the UCB formula needs to be large to deal with non-stationarity.



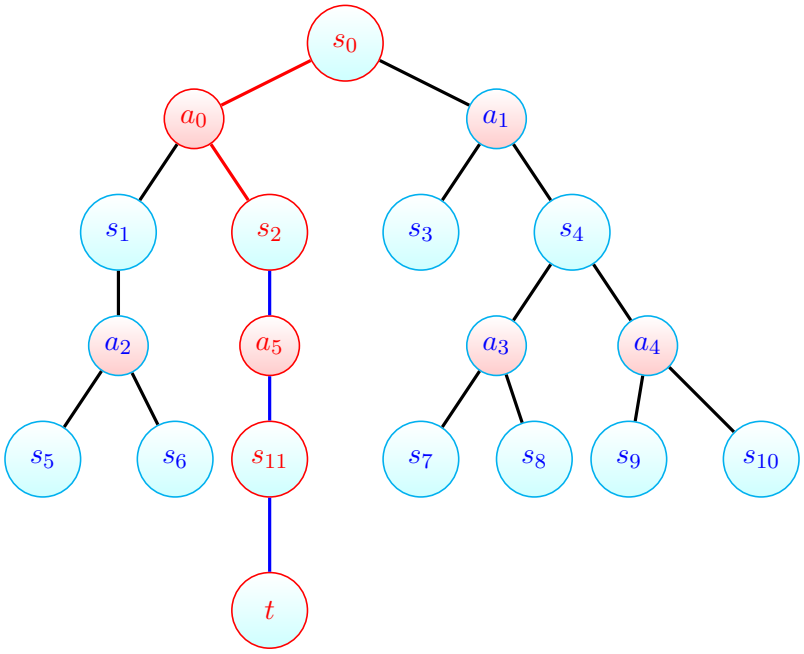
3.12.2 Expansion

If the lead node L is not terminal, then create one or more child nodes and select one, called C.



3.12.3 Simulation

Run a random simulation of the MDP from C until a result (terminating state, t) is reached.



3.12.4 Backpropagation

Obtain the reward r from the simulation till t and backpropagate it to update the statistics of the nodes in the path till C

4 Policy Gradient Methods

4.1 Stochastic Policy

A policy that, given a state, outputs a probability distribution over actions at that state.

$$\pi_\theta(s) = \mathbb{P}\left[A|s;\theta\right]$$

4.2 Objective Function

The expected value of a policy π_θ with parameters θ is given by the objective function.

$$J(\theta) = V^{\pi_\theta}\left(s_0\right)$$

The objective function is computationally infeasible to compute, so we search for $J(\theta)$ by **ascending the gradient**.

4.3 Policy Gradient

Given a policy objective $J(\theta)$, the **policy gradient** of J with respect to θ is given by:

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

We follow the gradient to maximize the objective function $J(\theta)$, and update the weights as:

$$\theta \leftarrow \theta + \alpha J(\theta)$$

Here, α is the learning rate.

4.4 Policy Gradient Theorem

The **policy gradient theorem** says that for any differentiable policy π_θ, state s, and action a, the objective function is given by:

$$J(\theta) = \mathbb{E}_{\pi_\theta} \left[Q(s,a) \nabla_\theta \log \pi_\theta(s,a) \right]$$

4.5 REINFORCE Algorithm

REINFORCE algorithm

$\theta \leftarrow$ arbitrary initialization
while True **do**
  $\theta_{\text{new}} \leftarrow \theta$
  Generate episode $s^0, a^0, r^0, \ldots, s^T = s_T$, following π_θ
  for $t = 0$ to $T-1$ **do**
    $G \leftarrow \sum_{k=t}^{T-1} r^k$
    $\theta_{\text{new}} \leftarrow \theta_{\text{new}} + \alpha \gamma^t G \nabla_\theta \log \pi_\theta(s^t, a^t)$
  end for
  $\theta \leftarrow \theta_{\text{new}}$
end while

4.6 Actor-Critic Methods

In REINFORCE algorithm, the high variance in the cumulative rewards of G can lead to slow convergence. Actor-Critic methods reduce this variance by introducing a **critic** that estimates the value function. The policy is called the **actor**. The main idea here is to **bootstrap**, i.e., to use $r^t + \hat{V}\left(s^{t+1}\right)$ instead of $G_{t:T}$. So the updation step becomes:

$$\theta_{\text{new}} \leftarrow \theta_{\text{new}} + \alpha \sum_{t=0}^{T-1} \left(r^t + \hat{V}\left(s^{t+1}\right) - \hat{V}\left(s^t\right) \right) \nabla_\theta \log \pi_\theta(s^t, a^t)$$

This reduces the variance in the policy gradient estimate. It is not always convergent, but widely used.

References

[1] Shivaram Kalyanakrishnan. *CS 747: Foundations of Intelligent and Learning Agents*. URL: https://www.cse.iitb.ac.in/~shivaram/teaching/old/cs747-a2022/index.html.

[2] Emma Brunskill. *CS234: Reinforcement Learning Spring 2024*. URL: https://web.stanford.edu/class/cs234/index.html.

[3] Sutton and Barto. *Reinforcement Learning: An Introduction*. URL: http://incompleteideas.net/book/the-book-2nd.html.

[4] David Silver. *Reinforcement Learning Course*. URL: https://www.davidsilver.uk/teaching/.

[5] Stuart Mitchell. *PuLP Documentation*. URL: https://coin-or.github.io/pulp/.

[6] NumPy Community. *NumPy Documentation*. URL: https://numpy.org/doc/stable/.

[7] Hugging Face. *Deep RL Course*. URL: https://huggingface.co/learn/deep-rl-course/.

[8] MCTS Research Hub. *Monte Carlo Tree Search*. URL: https://mcts.ai/about/index.html.

[9] Surag Nair. *A Simple Alpha(Go) Zero Tutorial*. URL: https://suragnair.github.io/posts/alphazero.html.

[10] David Silver, Demis Hassabis. *AlphaGo Zero: Starting from scratch*. URL: https://deepmind.google/discover/blog/alphago-zero-starting-from-scratch/.

[11] Udacity. *Deep Reinforcement Learning Nanodegree*. URL: https://github.com/udacity/deep-reinforcement-learning.

[12] Tim Miller. *Mastering Reinforcement Learning*. URL: https://gibberblot.github.io/rl-notes/intro.html.