Notes on RL

Haosheng Zhou

August, 2022

Contents

Bandit Problem

Basics of Bandit Problems

The bandit problem refers to a situation where there's always K actions to choose from at any time and there's only a single state in the state space. As a result, the reward only depends on which action to take and is assumed to be random (deterministic reward is trivial for bandit problem). Different actions have their respective mean of reward, called their **value**. The mean reward for action a is denoted $q_*(a)$,

$$q_*(a) = \mathbb{E}[R_t|A_t = a] \tag{1}$$

the reward of taking action a, WLOG, is a random number generated from $N(q_*(a), 1)$, and rewards are assumed to be independently generated. With a finite time horizon T provided, the goal is to find a strategy to produce a sequence of actions $A_1, ..., A_T$ that maximizes the aggregated reward $\sum_{i=1}^{T} R_i$.

Remark. *The bandit problem we consider is the stationary bandit problem, i.e. the distribution of R_t does not change w.r.t. time. On the other hand, if the distribution of R_t changes w.r.t. time, it's called an non-stationary bandit problem and is much harder to analyze.*

Unfortunately, we don't know about $q_*(a)$ (otherwise we would always take the action with the highest value). However, it's still not too bad because we might learn from the experience of playing this game and construct estimates for those values. The estimates for $q_*(a)$ at time t is denoted $Q_t(a)$. It seems natural that at time t, we shall look at $Q_t(a)$ for all possible action a to find the action with the highest estimated value, which is called a **greedy action**.

Based on current knowledge, the greedy action is definitely the best, but shall we necessarily take it? The answer is NO because our knowledge evolve with time. If we **exploit** (choose the greedy action) all the time and do not **explore** (try other actions), we would not be able to update our knowledge on other actions. This would result in the consequence that a potentially better action is never discovered. On the other hand, exploring too much is also not what we want since the loss in the aggregated reward would be too much. That means, **the trade-off between exploitation and exploration** is a key point in RL.

Let's think about how we shall construct the estimated value function Q_t to our current knowledge. $Q_t(a)$ should provide an estimate for the mean reward received by taking action a based on the experience until time t. It's natural that sample mean would be a good approximation to population mean,

$$Q_t(a) \overset{def}{=} \frac{\sum_{i=1}^{t-1} R_i \mathbb{I}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{I}_{A_i=a}}, N_t(a) = \sum_{i=1}^{t-1} \mathbb{I}_{A_i=a} \tag{2}$$

where $\mathbb{I}$ stands for the indicator. Such estimate $Q_t(a)$ is formed as the sample average of rewards before time t when action a is taken, called the **sample average** method for estimating action values, $N_t(a)$ is the number of times action a is taken before time t. The greedy action is formulated as:

$$A_t = \arg\max_a Q_t(a) \tag{3}$$

ε-greedy Strategy

To set space for exploration to happen, ε-greedy strategy is introduced. In each turn, we behave greedily with probability $1-\varepsilon$ and explore with probability ε. When exploring, uniformly randomly take one action out of all possible actions to gain knowledge. Refer to Fig. 1 for advantages of exploration in long term.

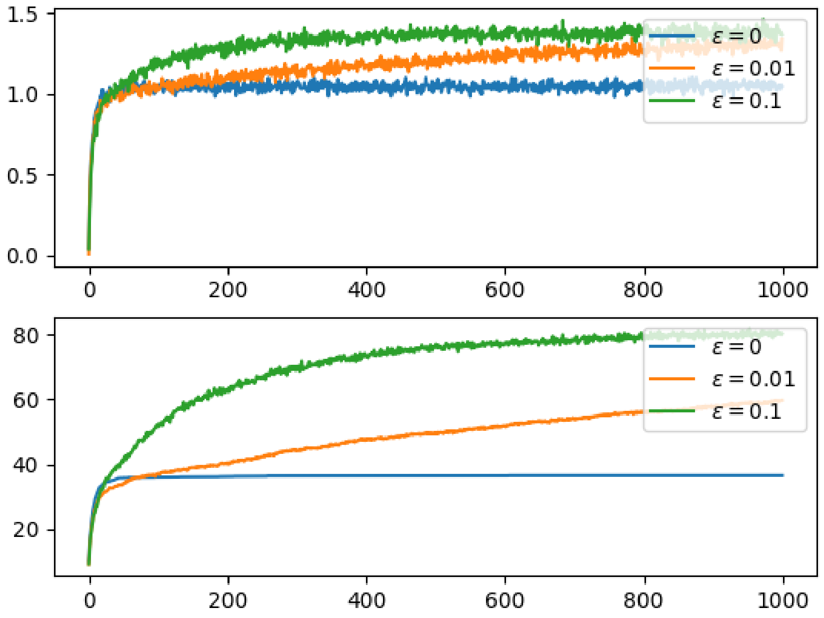


Figure 1: Advantage of ε-greedy strategy over greedy strategy

A K bandit-model with $K=10, T=1000$, action values are generated following $N(0,1)$ and rewards of action a are generated following $N(q_*(a),1)$.
The upper plot shows the average reward derived at each step and the lower plot shows the percentage for the current action to be the best action at each step. All data points are averages among 2000 different bandit problems.
As can be seen, greedy strategy does not explore at all. Although it's converging faster, the reward derived is far from optimal. As ε is slowly increased, more explorations are conducted, resulting in both the average reward and the best action percentage rate being much higher than that of the greedy strategy.

Remark. *$Q_t(a)$ can actually be written in an* ***incremental form****. When action a is taken at time t resulting in* *reward* R_t*,*

$$\forall a' \neq a, Q_{t+1}(a') = Q_t(a') \tag{4}$$

while

$$Q_{t+1}(a) = \frac{\sum_{i=1}^{t-1} R_i \mathbb{I}_{A_i=a} + R_t}{\sum_{i=1}^{t-1} \mathbb{I}_{A_i=a} + 1} = \frac{N_t(a)Q_t(a) + R_t}{N_t(a) + 1} = Q_t(a) + \frac{1}{N_t(a)+1}(R_t - Q_t(a)) \tag{5}$$

which means that we don't have to record and look through all history of actions and rewards each time we have seen a new action a*.*

Note that this incremental form also provides the intuition that we are adjusting the old estimate $Q_t(a)$ in the direction of the difference $R_t - Q_t(a)$ by a **step size factor** $\alpha_{t+1} = \frac{1}{N_t(a)+1}$. As a result, when it comes to non-stationary bandit problems where the reward distribution changes over time, we may replace the step size factor $\frac{1}{N_t(a)+1}$ with a fixed constant α formed as a hyperparameter.

Constant Step Size Methods for Non-stationary Problems

For the stationary bandit problem with sample mean method to estimate the value, use $\alpha_n(a)$ to denote the step size factor when seeing action a for the n-th time, then $\alpha_n = \frac{1}{n}$, satisfying the well-known convergence condition for stochastic optimization that $\sum_n \alpha_n(a) = \infty, \sum_n \alpha_n^2(a) < \infty$ (it's the iff condition for the a.s. convergence of stochastic optimization with varying step size). Although a constant step size factor violates these conditions, it's still adopted for its simplicity and effectiveness. Refer to Fig. 2 for the numerical experiment of **non-stationary bandit problem with constant step size** compared to that with sample mean method.

Optimistic Start

Some tricks for picking up special initial values of the estimated value Q may apply for stationary bandit problems, e.g. the trick of optimistic start. Since in our model the distribution of action values has mean 0. If we could start with an optimistic estimated value (set the initial values of Q as big positive numbers), after picking up an action to take and figuring out the reward of such action, the bandit algorithm would realize that the actual reward is far lower than the optimistic start provided. This results in the disappointment towards such an action so that other actions that still have the optimistic start with them will be explored.

Refer to Fig. 3 for numerical experiments. Although this trick works well for some bandit problems, the performance is not guaranteed so it does not deserve much attention.

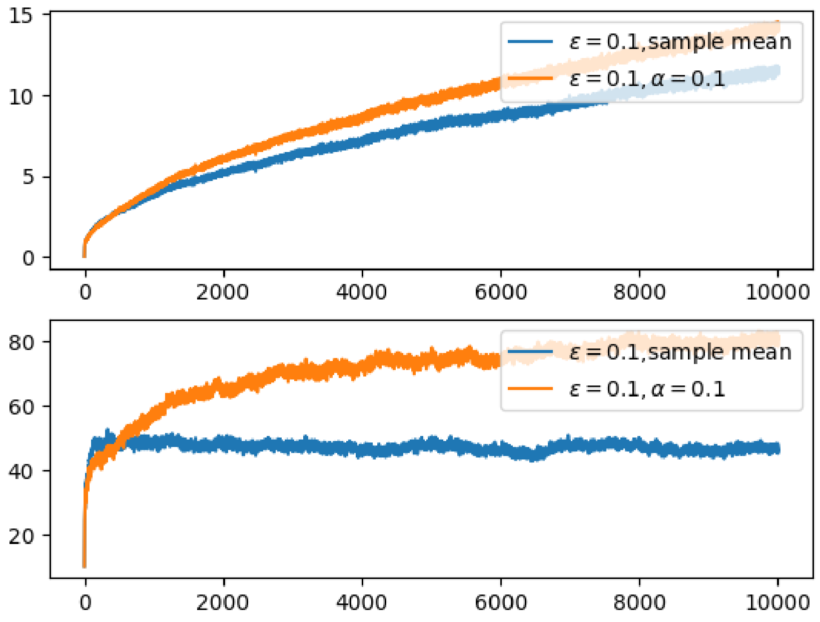


Figure 2: An non-stationary bandit problem

A K bandit-model with $K = 10, T = 10000$, values of actions are generated following $N(0, 1)$ but change over time. Each value has an independent $N(0, 0.01)$ random number added to it on each time step. The rewards of action a are generated following $N(q_*(a), 1)$.

The upper plot shows the average reward derived at each step and the lower plot shows the percentage for the current action to be the best action at each step. All data points are averages among 1000 different bandit problems.

As can be seen, sample mean method as a method with varying step size parameter $\alpha_n(a) = \frac{1}{n}$, behaves not as well as the value estimate with constant step size parameter $\alpha = 0.1$ practically.

Upper Confidence Bound (UCB) Method

In the context above, discussion are organized within the scope of ε-greedy strategy. When the algorithm decides to explore, all available actions are given the same chance of being selected. However, it would have been much more efficient if the actions that are more likely to be the optimal ones could be explored. This leads to the **upper confidence bound (UCB)** method that first visits all possible actions once and then selects action based on

$$A_t = \arg\max_a \left(Q_t(a) + c\sqrt{\frac{\log \frac{2TK}{\delta}}{N_t(a)}} \right) \tag{6}$$

where $\delta > 0$ is a small probability to be specified and $c > 0$ is a hyperparameter for the degree of exploration. If action a is selected, the increase of $N_t(a)$ narrows the confidence bound and it's believed that the updated $Q_t(a)$

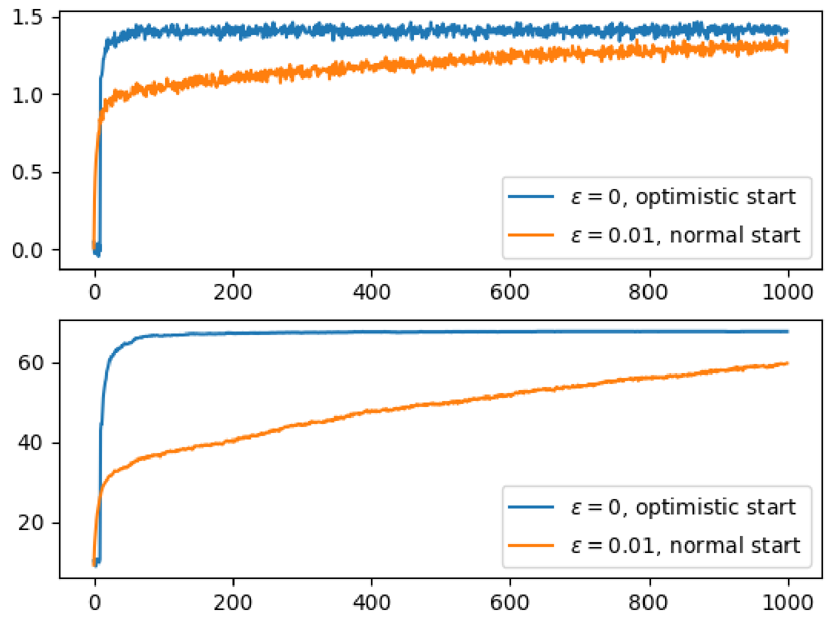


Figure 3: Trick of optimistic start

A stationary K bandit-model with $K = 10, T = 1000$, values of actions are generated following $N(0, 1)$. The rewards of action a are generated following $N(q_*(a), 1)$.
The upper plot shows the average reward derived at each step and the lower plot shows the percentage for the current action to be the best action at each step. All data points are averages among 2000 different bandit problems.
As can be seen, for stationary bandit problem, the bandit algorithm with greedy strategy and optimistic start (for any action a, initial values $Q_1(a) = 5$) converges much faster than that with ε-greedy strategy ($\varepsilon = 0.01$) and normal start (for any action a, initial values $Q_1(a) = 0$).

reflects more accurate information on action a. In contrast, if action a has never been visited again since the start of the game, it has quite a high chance to be explored as time goes by, except that some really good action with a high action value exists.

The algorithm gets its name because $Q_t(a) + c\sqrt{\frac{\log \frac{2TK}{\delta}}{N_t(a)}}$ is the upper confidence bound of $Q_t(a)$, so the strategy is called **optimism towards uncertainty**, i.e. taking actions based on the upper confidence bound to ensure exploration. The reason why such expression provides a confidence bound remains a mystery at this point. Due to the importance of UCB, we will provide a proof for this algorithm in a later context.

Remark. *The UCB algorithm for bandit problem has a lot of different variants built to satisfy different bounds. Here we present one of them which has a nice probabilistic concentration bound shown in a later context. Just don't be surprised when you see slightly different algorithms all with the name UCB.*

Gradient Bandit Methods

So far, the action-value methods we have discussed all focus on picking up the action that maximizes some kind of estimates for action value to ensure exploration. However, in order to maintain a certain amount of exploration, why don't let randomness help us, i.e. why don't we **randomize the action**? That's exactly the motivation of gradient bandit method where a numerical **preference** for each action a at time t, denoted $H_t(a)$ is introduced to generate a probability distribution on the action space.

In order to turn the real numbers into a probability distribution, the softmax function is naturally considered

$$\begin{aligned}\pi_t(a) &\overset{def}{=} \mathbb{P}\left(A_t = a\right) &(7)\\ &\overset{def}{=} \frac{e^{H_t(a)}}{\sum_b e^{H_t(b)}} &(8)\end{aligned}$$

where $\pi_t(a)$ denotes the probability of taking action a at time t.

The only question we have to think about now is how to update the preference H_t based on the experience. Gradient bandit algorithm tells us that it's given by

$$H_{t+1}(a) = H_t(a) + \alpha\left(R_t - \overline{R}_t\right)\left(\mathbb{I}_{A_t=a} - \pi_t(a)\right) \tag{9}$$

where $\alpha > 0$ is the learning rate as a hyperparameter and $\overline{R}_t$ is the mean reward received up to time t with time t included, which works as a **baseline**.

Remark. *A direct explanation is that for action a, if it has been selected as the optimal action at time t, the preference shall rise if the reward is higher than past average (a positive effect). For other actions that have not been selected at time t, the similar logic holds, with the direction of change different from that of the selected action. This algorithm exhibits the process of learning, i.e. prioritizing the actions that bring with benefits and eliminating the actions that bring with loss.*

The update of preference seems intuitively correct but where does it come from? The answer is gradient. Keep in mind that in the context of bandit problem our goal is to maximize the aggregated expected reward received. Viewed in single time step, H_t shall be updated such that it maximized the expected reward received at time t, which is $\mathbb{E}R_t$. From a gradient ascent perspective,

$$H_{t+1}(a) = H_t(a) + \alpha\frac{\partial \mathbb{E}R_t}{\partial H_t(a)} \tag{10}$$

with

$$\begin{aligned}\mathbb{E}R_t &= \sum_a \mathbb{P}\left(A_t = a\right)\mathbb{E}(R_t|A_t = a) &(11)\\ &= \sum_a \pi_t(a) q_*(a) &(12)\end{aligned}$$

However, $q_*(a)$ is unknown so such gradient ascent method can't be implemented practically. Despite this fact, let's still calculate the partial derivative to see what we can get.

$$\begin{aligned}
\frac{\partial \mathbb{E}R_t}{\partial H_t(a)} &= \sum_x q_*(x) \frac{\partial \pi_t(x)}{\partial H_t(a)} && (13)\\
&= \sum_x q_*(x) \frac{\partial \frac{e^{H_t(x)}}{\sum_b e^{H_t(b)}}}{\partial H_t(a)} && (14)\\
&= \sum_x q_*(x) \frac{\mathbb{I}_{a=x} e^{H_t(x)} \sum_b e^{H_t(b)} - e^{H_t(x)} e^{H_t(a)}}{\left(\sum_b e^{H_t(b)}\right)^2} && (15)\\
&= \sum_x q_*(x) \pi_t(x) \left(\mathbb{I}_{a=x} - \pi_t(a)\right) && (16)
\end{aligned}$$

notice that there is a special structure of π_t that

$$\begin{aligned}
\sum_x \pi_t(x) &= 1 && (17)\\
\sum_x \frac{\partial \pi_t(x)}{\partial H_t(a)} &= \sum_x \pi_t(x) \mathbb{I}_{x=a} - \sum_x \pi_t(x) \pi_t(a) && (18)\\
&= \pi_t(a) - \pi_t(a) = 0 && (19)
\end{aligned}$$

use this property to add an arbitrary baseline B_t to be specified later (as a function of $R_1, ..., R_t$ already observed),

$$\begin{aligned}
\frac{\partial \mathbb{E}R_t}{\partial H_t(a)} &= \sum_x \left(q_*(x) - B_t\right) \frac{\partial \pi_t(x)}{\partial H_t(a)} && (20)\\
&= \sum_x \pi_t(x) \left(q_*(x) - B_t\right) \left(\mathbb{I}_{a=x} - \pi_t(a)\right) && (21)
\end{aligned}$$

Since π_t is a probability distribution on the action space, this partial derivative has a natural structure as an expectation w.r.t. the action $A_t \sim \pi_t$:

$$\frac{\partial \mathbb{E}R_t}{\partial H_t(a)} = \mathbb{E}_{A_t \sim \pi_t} \left[\left(q_*(A_t) - B_t\right) \left(\mathbb{I}_{A_t=a} - \pi_t(a)\right)\right] \qquad (22)$$

up to this point, the original gradient ascent scheme can be rewritten as

$$\begin{aligned}
H_{t+1}(a) &= H_t(a) + \alpha \frac{\partial \mathbb{E}R_t}{\partial H_t(a)} && (23)\\
&= H_t(a) + \alpha \mathbb{E}_{A_t \sim \pi_t} \left[\left(q_*(A_t) - B_t\right) \left(\mathbb{I}_{A_t=a} - \pi_t(a)\right)\right] && (24)
\end{aligned}$$

with action value $q_*(A_t)$ unknown, which has exactly the form of **stochastic gradient ascent** method, where the ascent direction is taken randomly and it's only ensured that the expectation of the random ascent direction equals the true gradient direction.

Although it's impossible to figure out this expectation (the true gradient), it's completely possible to sample random directions from the distribution with mean $\mathbb{E}_{A_t \sim \pi_t}\left[\left(q_*(A_t) - B_t\right)\left(\mathbb{I}_{A_t=a} - \pi_t(a)\right)\right]$. To show why it's possible, $q_*(A_t) = \mathbb{E}(R_t|A_t)$ results in

$$\mathbb{E}_{A_t \sim \pi_t}\left[\left(q_*(A_t) - B_t\right)\left(\mathbb{I}_{A_t=a} - \pi_t(a)\right)\right] = \mathbb{E}_{A_t \sim \pi_t, R_t}\left[\left(R_t - B_t\right)\left(\mathbb{I}_{A_t=a} - \pi_t(a)\right)\right] \tag{25}$$

and $(R_t - B_t)(\mathbb{I}_{A_t=a} - \pi_t(a))$ is the random direction we want, with R_t observable.

So far, we have proved that **the gradient bandit algorithm is an instance of stochastic gradient ascent**. Its general form for any baseline B_t as a function of $R_1, ..., R_t$ is as follows:

$$H_{t+1}(a) = H_t(a) + \alpha\left(R_t - B_t\right)\left(\mathbb{I}_{A_t=a} - \pi_t(a)\right) \tag{26}$$

in practice, $B_t = \overline{R}_t$ as past averages works pretty well. Although the gradient bandit algorithm is ensured to be an instance of stochastic gradient ascent regardless of the value of B_t, a lack of baseline for which $B_t = 0$ has poor performance empirically.

Comparison of Bandit Algorithms

The comparison can be conducted in different criterion. See Fig. 4 for one comparison criterion. Actually the most useful comparison criterion in the setting of bandit problem is the cumulative regret, see Fig. 5, 6 for the plots and later sections for the definition and theoretical analysis.

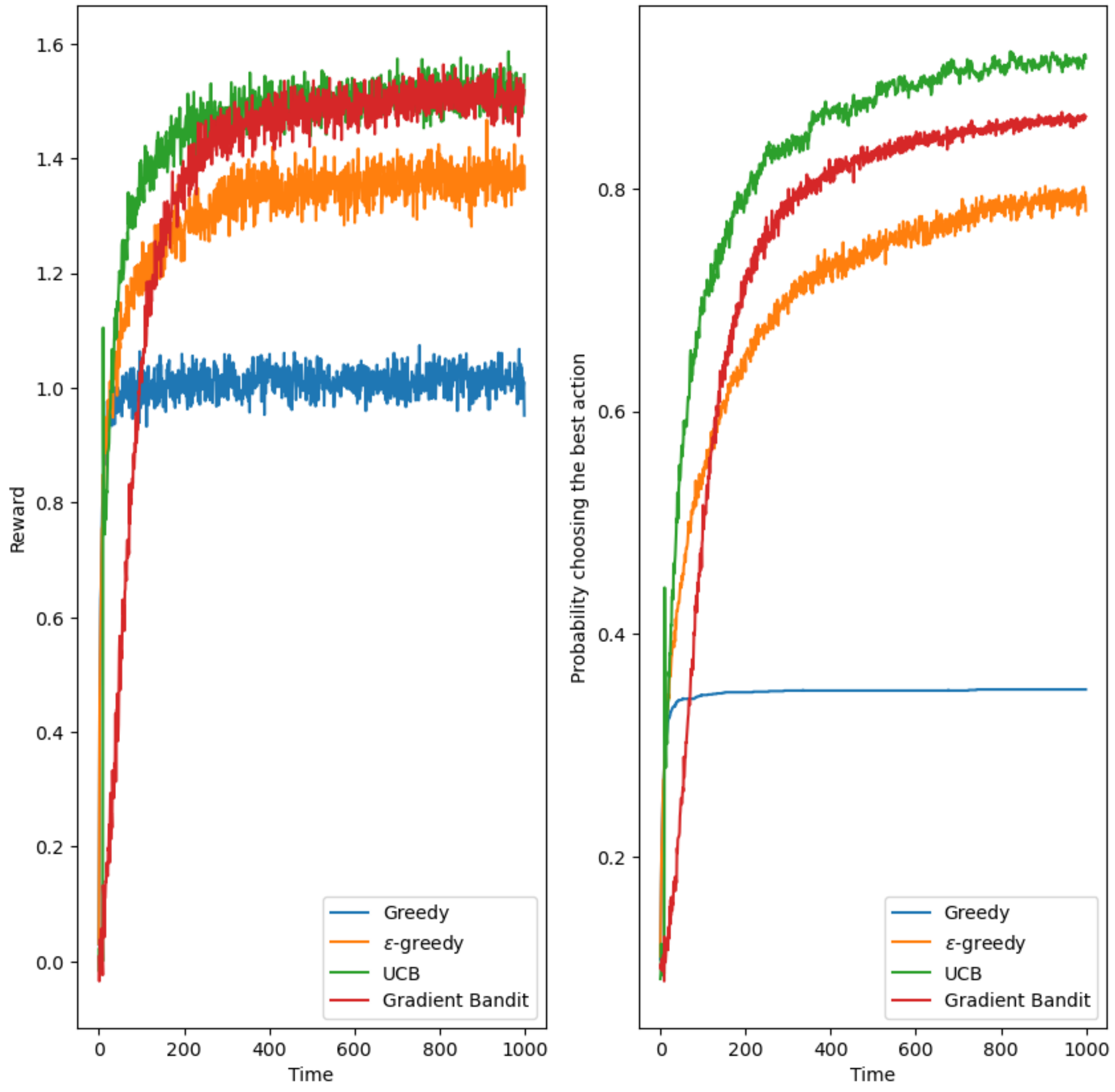


Figure 4: Comparison of Bandit Algorithms
The numerical experiment setting is the same as that in the plots above. Four algorithms are attempted.
The left plot shows the reward we get as time goes by while the right plot shows the probability of choosing the best action as time goes by.

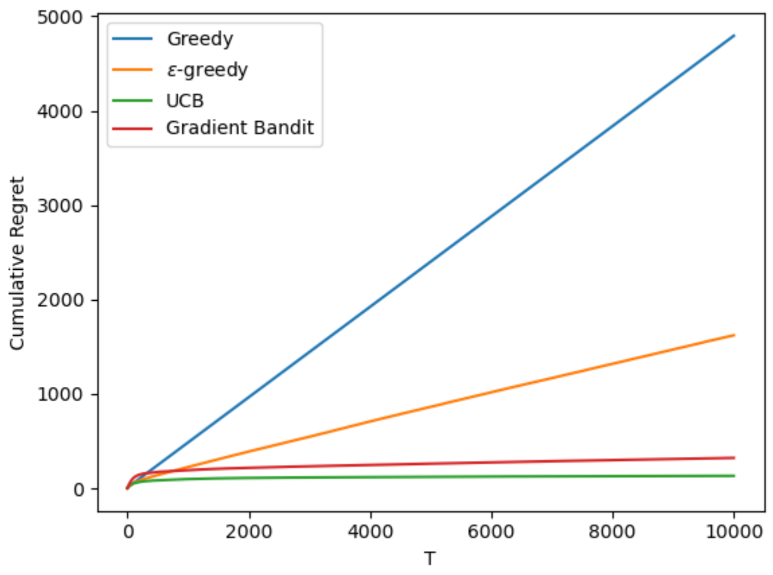


Figure 5: Cumulative Regret of Bandit Algorithms

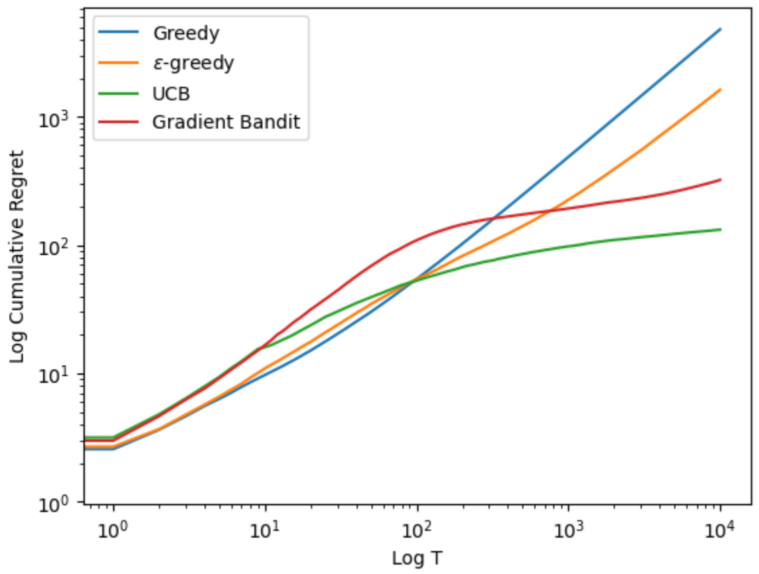


Figure 6: Cumulative Regret of Bandit Algorithms, Log-Log Plot

UCB, LinUCB and Regret Analysis

One of the algorithms that performs pretty well for bandit problems is the UCB method. Recall that UCB algorithm tells us to first explore through all K possible actions once and then select action based on

$$A_t = \arg\max_a \left(Q_t(a) + c\sqrt{\frac{\log \frac{2TK}{\delta}}{N_t(a)}} \right) \tag{27}$$

for some small probability $\delta \in (0,1)$, some constant $c > 0$ and $N_t(a) = \sum_{i=1}^{t-1} \mathbb{I}_{A_t=a}$. In this section, we illustrate why this bound is called the upper confidence bound, what's the interpretation of hyperparameters c, δ and why this algorithm works that well in practice. Before doing any of those, we want to build a measurement for different algorithms in bandit problem for the purpose of comparison.

Cumulative Regret

The **cumulative regret** up to time T is defined as

$$R_T = T \max_a q_*(a) - \sum_{i=1}^{T} q_*(A_i) \tag{28}$$

the term regret means that at time i on taking action A_i, we would regret on not taking the optimal action and get the maximum possible expected reward $\max_a q_*(a)$. As a result

$$R_T = \sum_{i=1}^{T} \left(\max_a q_*(a) - q_*(A_i) \right) \tag{29}$$

is just the sum of the opportunity cost at all time steps resulting from not taking the best action.

In the following context we always focus on estimating the cumulative regret of the action sequence generated by a certain algorithm. For simplicity, we always assume that the rewards are random but almost surely bounded taking value in $[0,1]$. This assumption enables us to use simpler concentration bounds for a.s. bounded random variables. WLOG, all those conclusions also work for sub-Gaussian random variables, the connection provided by Hoeffding's lemma.

Analysis for UCB

Let's provide an estimate for the cumulative regret of UCB. Assume that $T >> K$, the first K actions are devoted in exploring all available actions which result in $O(K)$ contribution to the cumulative regret. Notice that adding this exploration step provides empirically better performance but in the theoretical analysis it's not crucial thus neglected.

The analysis of UCB algorithm starts from an observation on $Q_t(a)$ that it's an unbiased estimator for $q_*(a)$ for every action a given action sequence $A_1, ..., A_{t-1}$. To see this,

$$\mathbb{E}(Q_t(a)|A_1, ..., A_{t-1}) = \frac{\sum_{i=1}^{t-1} \mathbb{E}(R_i|A_1, ..., A_{t-1})\mathbb{I}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{I}_{A_i=a}} = \frac{\sum_{i=1}^{t-1} q_*(a)\mathbb{I}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{I}_{A_i=a}} = q_*(a) \tag{30}$$

hence we have the reason to believe that $Q_t(a)$ would concentrate near $q_*(a)$. If we fix action a,

$$X_i = \mathbb{I}_{A_i=a}(R_i - q_*(a)) \tag{31}$$

defines a sequence of random variables adapted to the filtration $\{\mathscr{F}_t\}$ where $\mathscr{F}_t$ denotes the collection of information in the game up to time t. Since UCB tells us the way to generate A_t based on all the information before time t, it's clear that $A_t \in \mathscr{F}_{t-1}$ is predictable. Simple calculation tells us

$$\mathbb{E}(X_t|\mathscr{F}_{t-1}) = \mathbb{I}_{A_t=a}[\mathbb{E}(R_t|\mathscr{F}_{t-1}) - q_*(a)] = 0 \tag{32}$$

as a result, the sequence of r.v. $\{X_t\}$ can be seen as a sequence of MG increments. Thinking of concentration inequality for bounded martingale increments, Azuma's inequality is the right tool to use.

Lemma 1 (Azuma's Inequality)**.** *Let $\{S_n\}$ be a martingale with almost surely bounded increments $\forall k, A_k \leq S_k - S_{k-1} \leq B_k$ a.s. for deterministic A_k, B_k, then*

$$\forall a > 0, \mathbb{P}\left(S_n - S_0 \geq a\right) \leq e^{-\frac{2a^2}{\sum_{i=1}^{n}(B_i - A_i)^2}} \tag{33}$$

Apply Azuma's inequality for $S_t = \sum_{i=1}^{t} X_i$ as a MG to see that

$$\mathbb{P}\left(\sum_{i=1}^{t} X_i \geq c\sqrt{\log\frac{2}{\delta}\cdot N_t(a)}\Big| N_t(a)\right) \leq e^{-\frac{2c^2 \log\frac{2}{\delta} N_t(a)}{4N_t(a)}} = e^{-\log\frac{2}{\delta}} = \frac{\delta}{2} \tag{34}$$

since X_i takes value zero if $A_i \neq a$ and $-1 \leq X_i \leq 1$ *a.s.* if $A_i = a$. Since action a has been taken for $N_t(a)$ times given the value of $N_t(a)$, $\sum_{i=1}^{t}(B_i - A_i)^2 \leq 4N_t(a)$. Taking $c^2 = 2$, we get the concentration inequality above. Written in double-sided form

$$\mathbb{P}\left(\left|\sum_{i=1}^{t} X_i\right| \geq c\sqrt{\log\frac{2}{\delta}\cdot N_t(a)}\right) \leq \delta \tag{35}$$

with high probability (at least $1-\delta$), $\left|\sum_{i=1}^{t} X_i\right| \leq c\sqrt{\log\frac{2}{\delta}\cdot N_t(a)}$ happens.

Notice that

$$\sum_{i=1}^{t} X_i = \sum_{i=1}^{t} \mathbb{I}_{A_i=a}(R_i - q_*(a)) = N_t(a)Q_t(a) - N_t(a)q_*(a) \tag{36}$$

this tells us on fixing action a and time t, with high probability (at least $1-\delta$) the following concentration bound holds

$$|Q_t(a) - q_*(a)| \le c\sqrt{\frac{\log \frac{2}{\delta}}{N_t(a)}} \tag{37}$$

taking union bound w.r.t. $\forall t \in \{1, 2, ..., T\}, a \in \{1, 2, ..., K\}$, with probability at least $1 - KT\delta$ the same bound holds. Set δ' as $\frac{\delta}{KT}$, we have proved the following result.

Lemma 2 (Concentration of $Q_t(a)$)**.** *With probability at least* $1 - \delta$*,*

$$\forall t \in \{1, 2, ..., T\}, \forall a \in \{1, 2, ..., K\}, |Q_t(a) - q_*(a)| \le c\sqrt{\frac{\log \frac{2TK}{\delta}}{N_t(a)}} \tag{38}$$

Now it's time to prove the regret bound for UCB.

Theorem 1 (UCB Regret Bound)**.** *With probability at least* $1 - \delta$*,*

$$R_T = O\left(\min\left\{\sqrt{KT\log\frac{2KT}{\delta}}, \sum_{a \ne a^*} \frac{\log \frac{2KT}{\delta}}{\Delta_a}\right\} + K\right) \tag{39}$$

where a^* *denotes the best action and* $\Delta_a = q_*(a^*) - q_*(a)$ *denotes the optimality gap of action* a*.*

Remark. *There are three parts in the regret bound. The first one is* K*, which results from the exploration stage and is typically negligible.*

The second part is $\sqrt{KT\log\frac{KT}{\delta}}$*, which is denoted* $\tilde{O}(\sqrt{KT})$*.* $\tilde{O}$ *means the time complexity ignoring logarithmic factors compared to the main factor.*

The third part is $\sum_{a \ne a^*} \frac{\log \frac{KT}{\delta}}{\Delta_a}$*, saying that if the expected rewards of different bandits are very different (large* Δ_a*), UCB works well and vice versa. This term represents the intrinsic complexity of the bandit problem from the perspective of the reward distribution.*

Proof. We neglect the $O(K)$ regret from the exploration stage and work on proving the remaining terms.

From the concentration inequality, with probability at least $1 - \delta$,

$$\forall a, t, q_*(a) \le Q_t(a) + c\sqrt{\frac{\log \frac{2TK}{\delta}}{N_t(a)}} \tag{40}$$

denote $\{A_t\}$ as the sequence of actions generated by UCB, by definition of UCB,

$$\forall t, q_*(a^*) \le Q_t(a^*) + c\sqrt{\frac{\log \frac{2TK}{\delta}}{N_t(a^*)}} \le Q_t(A_t) + c\sqrt{\frac{\log \frac{2TK}{\delta}}{N_t(A_t)}} \tag{41}$$

so UCB generated action value $q_*(A_t)$ is always not too far away from the best action value $q_*(a^*)$

$$\forall t, q_*(a^*) - q_*(A_t) \leq Q_t(A_t) - q_*(A_t) + c\sqrt{\frac{\log \frac{2TK}{\delta}}{N_t(A_t)}} \leq 2c\sqrt{\frac{\log \frac{2TK}{\delta}}{N_t(A_t)}} \tag{42}$$

sum over $t \in \{1, 2, ..., T\}$ to get

$$R_T = \sum_{t=1}^{T} [q_*(a^*) - q_*(A_t)] \tag{43}$$

$$\leq 2c \sum_{t=1}^{T} \sqrt{\frac{\log \frac{2TK}{\delta}}{N_t(A_t)}} \tag{44}$$

$$= 2c\sqrt{\log \frac{2TK}{\delta}} \sum_{t=1}^{T} \frac{1}{\sqrt{N_t(A_t)}} \tag{45}$$

the summation can be simplified here since A_t can only take values in $\{1, 2, ..., K\}$

$$\sum_{t=1}^{T} \frac{1}{\sqrt{N_t(A_t)}} = \sum_{a=1}^{K} \sum_{t: A_t = a} \frac{1}{\sqrt{N_t(a)}} = \sum_{a=1}^{K} \sum_{i=1}^{N_T(a)} \frac{1}{\sqrt{i}} \tag{46}$$

the last equation is from re-ordering the terms w.r.t. the second summation index. Action a must have appeared for $N_T(a)$ times so the i-th time it's appearing, it contributes $\frac{1}{\sqrt{i}}$ to the summation. An easy integral estimation now gives

$$\sum_{a=1}^{K} \sum_{i=1}^{N_T(a)} \frac{1}{\sqrt{i}} \leq \sum_{a=1}^{K} \left(1 + \int_1^{N_T(a)} \frac{1}{\sqrt{x}}\, dx\right) \leq 2 \sum_{a=1}^{K} \sqrt{N_T(a)} \tag{47}$$

since $\sum_{a=1}^{K} N_T(a) = T$, we want to see this summation, apply Cauchy-Schwarz

$$\sum_{a=1}^{K} \sqrt{N_T(a)} \cdot 1 \leq \sqrt{\sum_{a=1}^{K} 1 \cdot \sum_{a=1}^{K} N_T(a)} = \sqrt{KT} \tag{48}$$

combine all inequalities together to see

$$R_T \leq 2c\sqrt{\log \frac{2TK}{\delta}}\sqrt{KT} \tag{49}$$

this proves one bound in the minimum.

For the other bound, the proof is simpler

$$\begin{aligned} R_T &= \sum_{t=1}^{T}[q_*(a^*) - q_*(A_t)] && (50)\\ &\le \sum_{a\neq a^*} \sum_{1\le t\le T, A_t=a} [q_*(a^*) - q_*(a)] && (51)\\ &\le \sum_{a\neq a^*} N_T(a)\cdot \Delta_a && (52) \end{aligned}$$

for $\forall a \neq a^*$, consider the last time action a is selected by UCB (e.g. at time t), then the upper confidence bound of a must exceed that of a^*

$$Q_t(a) + c\sqrt{\frac{\log \frac{2tK}{\delta}}{N_t(a)}} \ge Q_t(a^*) + c\sqrt{\frac{\log \frac{2tK}{\delta}}{N_t(a^*)}} \ge q_*(a^*) \tag{53}$$

the second inequality happens with probability at least $1-\delta$. With high probability, we also have

$$|Q_t(a) - q_*(a)| \le c\sqrt{\frac{\log \frac{2TK}{\delta}}{N_t(a)}} \tag{54}$$

since it's the last time action a is selected, $N_t(a) = N_T(a)$ so

$$\forall a \neq a^*, 0 < \Delta_a = q_*(a^*) - q_*(a) \le 2c\sqrt{\frac{\log \frac{2TK}{\delta}}{N_T(a)}}, N_T(a) \le 4c^2\frac{\log \frac{2TK}{\delta}}{\Delta_a^2} \tag{55}$$

as a result,

$$R_T \le \sum_{a\neq a^*} 4c^2\frac{\log \frac{2TK}{\delta}}{\Delta_a} \tag{56}$$

concludes the proof.

□

Remark. *This theorem shows the* ***sublinear*** $\tilde{O}(\sqrt{KT})$ *cumulative regret of UCB. In contrast, ε-greedy strategy still has linear cumulative regret $O(T)$ (try to argue this).*

From the proof, it's clear that $1-\delta$ stands for the confidence of the bound, c has something to do with the sub-Gaussian constant or the width of concentration region of random variables.

See Fig. 5, 6 that numerically shows what we have proved above.

Linear Bandit Problem and LinUCB

Linear bandit problem is an extension of the normal bandit problem discussed above. In linear bandit problem, there are uncountably many bandits available labelled by $a \in D$ where D is a compact subset of $\mathbb{R}^d$. Similarly, the action value is stationary and deterministic but unknown, formed as

$$\mathbb{E}(R_t|A_t = a) = \mu^* \cdot a \tag{57}$$

so that the mean of the reward received is linear in action a taken, but $\mu^* \in \mathbb{R}^d$ is unknown. Assume that in linear bandit problem the reward R_t is still almost surely bounded on $[-1, 1]$ so that

$$\forall a \in D, |\mu^* \cdot a| \leq 1 \tag{58}$$

clearly, for fixed μ^* together with $|\mu^* \cdot a| \leq 1$ for any action a on compact set D, there exists $a^* \in D$ as the maximizer of $\mu^* \cdot a$, i.e. the best action available. Ideally, one wants to keep taking the action a^* to achieve the maximum aggregated reward but unluckily there's no way to find a^* given that μ^* is unknown. As a result, we expect to see again a trade-off between exploration and exploitation in the linear bandit problem. In short, linear bandit problem is a bandit problem with a **large action space** but it's still not as general as the setting of RL problem since state transition is not yet involved.

A famous algorithm solving the linear bandit problem is the LinUCB algorithm as a natural extension of the UCB algorithm adopting the spirit of optimism towards uncertainty. See the following Alg. 1 for the details.

Algorithm 1 LinUCB

```
Input: Confidence constant $\delta \in (0,1)$
Output: A sequence of actions selected $\{A_t\}$
1: for t=1,...,T do
2:     $A_t = \arg\max_{a \in D} \max_{\mu \in B_t} \mu \cdot a$
3:     Take action $A_t$ which results in reward $R_t$
4:     Update the confidence region $B_{t+1}$
5: end for
```

The confidence region in LinUCB is provided as

$$B_t = \left\{\mu \in \mathbb{R}^d : (\hat{\mu}_t - \mu)^T \Sigma_t (\hat{\mu}_t - \mu) \leq \beta_t\right\} \tag{59}$$

where

$$\Sigma_{t+1} = \Sigma_t + A_t A_t^T, \Sigma_1 = \lambda I \tag{60}$$

and $\hat{\mu}_t$ as the center of B_t is constructed as the optimizer of a ridge regression to match $\mu \cdot A_i$ with the observed reward R_i (without specification, all vector norms are ℓ_2 norms)

$$\hat{\mu}_t = \arg\min_{\mu} \sum_{i=1}^{t-1} \|\mu \cdot A_i - R_i\|^2 + \lambda \|\mu\|^2 \tag{61}$$

Remark. *Simple calculation shows*

$$\frac{\partial}{\partial \mu} \sum_{i=1}^{t-1} \|\mu \cdot A_i - R_i\|^2 + \lambda \|\mu\|^2 = \sum_{i=1}^{t-1} 2A_i(\mu \cdot A_i - R_i) + 2\lambda\mu \tag{62}$$

set it as zero to see that

$$\hat{\mu}_t = \left(\sum_{i=1}^{t-1} A_i A_i^T + \lambda I\right)^{-1} \sum_{i=1}^{t-1} R_i A_i = \Sigma_t^{-1} \sum_{i=1}^{t-1} R_i A_i \tag{63}$$

keep in mind that A_i is vector-valued while R_i is scalar-valued.

Remark. *Optimism towards uncertainty shall be understood in terms of the two maximum in LinUCB. As what happens in UCB, it selects the action with the largest reward in the optimistic case (the largest possible reward to get under some confidence level). Here the double maximum has the same reasoning. We take the action A_t that generates the largest reward in the optimistic case (consider only the good case when $\mu \in B_t$ also maximizes $\mu \cdot x$).*

λ and β_t seems to be the hyperparameters in this algorithm, but we will provide a nice set of values for them in a later context to achieve the theoretical bound.

When it comes to the main idea of LinUCB, the guideline of action selection is the optimism towards uncertainty, $\hat{\mu}_t$ as an estimator for μ^* is derived as an optimizer of a ridge regression to match the action A_i with the reward R_i (don't forget that there is randomness in R_i given action A_i!). After that, a confidence region B_t is built as an ellipsoid centered at $\hat{\mu}_t$ with it's shape determined by the matrix Σ_t. This confidence region gets its name since it's believed that with high probability B_t contains the unknown μ^*. Finally, Σ_t is constructed initially as a positive definite matrix added by the outer product of the action as time goes by.

Remark. *Notice that the optimization step with double maximum is NP-hard in LinUCB. Despite the existence of nice theoretical bounds for LinUCB, it's not practically useful. In practice, we often enlarge the confidence region, which means that we keep track of an ℓ_1 ball entered at $\hat{\mu}_t$ that covers B_t to be the confidence region, i.e. being more optimistic towards the reward we are going to receive. In this case, the optimization step in LinUCB can be carried out efficiently in polynomial time.*

Refer to ***Online Least Squares Estimation with Self-Normalized Processes: An Application to Bandit Problems by Abbasi-Yadkori et al.*** *for another more efficient variant of LinUCB and a concentration analysis of LinUCB.*

Analysis for LinUCB

There are a lot of works done on the theoretical bounds for LinUCB, and a lot of variants of the LinUCB algorithm. Here we present a probabilistic upper bound for the cumulative regret of the LinUCB algorithm.

Remark. *There is also a lower bound on the LinUCB algorithm showing that one cannot do much better than LinUCB asymptotically in linear bandit problem, i.e. the optimality of LinUCB. We shall be able to cover only the idea of the proof of the lower bound in this note. Refer to **Stochastic Linear Optimization under Bandit Feedback by Dani et al.** if you are interesteed in the details.*

Similar to UCB, LinUCB is evaluated through the cumulative regret (we abuse notation here to use R_T for cumulative regret up to time T, when R_T stands for the reward received at time T we shall mention it explicitly) where $\{A_t\}$ is a sequence of actions generated by LinUCB

$$R_T = T\mu^* \cdot a^* - \sum_{t=1}^{T} \mu^* \cdot A_t \tag{64}$$

The following theorem provides an upper bound for the cumulative regret of LinUCB and also tells us how we shall choose λ and β_t in LinUCB.

Theorem 2 (LinUCB Regret Upper Bound)**.** *Under the assumption that*

$$\forall a \in D, |\mu^* \cdot a| \leq 1 \tag{65}$$

and that

$$\exists B, W, \|\mu^*\| \leq W, \forall a \in D, \|a\| \leq B \tag{66}$$

together with the assumption that the noise sequence (here R_t stands for reward at time t)

$$\eta_t = R_t - \mu^* \cdot A_t \tag{67}$$

is σ^2 sub-Gaussian, set

$$\lambda = \frac{\sigma^2}{W^2}, \beta_t = \sigma^2 \left(2 + 4d \log\left(1 + \frac{tB^2W^2}{d\sigma^2}\right) + 8 \log \frac{4}{\delta}\right) \tag{68}$$

with probability at least $1 - \delta$,

$$\forall T, R_T \leq C\sigma\sqrt{T} \left(d \log\left(1 + \frac{TB^2W^2}{d\sigma^2}\right) + \log \frac{4}{\delta}\right) \tag{69}$$

for some uniform constant $C > 0$, so with high probability $R_T = \tilde{O}(d\sqrt{T})$.

This analysis for LinUCB is important and can be replicated in many situations in RL. The idea of the proof is that we first stick to the belief that with high probability all confidence regions contain the unknown μ^*, i.e. $\forall t, \mu^* \in B_t$. Eventually, we come back and verify this belief through a concentration statement.

From now on we act as if we believe in the fact that $\forall t, \mu^* \in B_t$. Let's start at the cumulative regret and apply Cauchy-Schwarz to introduce a square that matches the structure of ℓ_2 norm in LinUCB

$$R_T = \sum_{t=1}^{T} (\mu^* \cdot a^* - \mu^* \cdot A_t) \leq \sqrt{T \sum_{t=1}^{T} (\mu^* \cdot a^* - \mu^* \cdot A_t)^2} \tag{70}$$

then we think about bounding the regret $\mu^* \cdot a^* - \mu^* \cdot A_t$ at time t. This bound comes from the fact that since μ^* is always in the confidence region, the regret at time t shall be bounded by the width of the ellipsoid in the direction of the action taken. This is formed as a lemma below.

Lemma 3. *If $\forall t, \mu^* \in B_t, \beta_t \geq 1$, then*

$$\mu^* \cdot a^* - \mu^* \cdot A_t \leq 2\sqrt{\beta_T} \min\{w_t, 1\} \tag{71}$$

where

$$w_t = \sqrt{A_t^T \Sigma_t^{-1} A_t} \tag{72}$$

Proof. If $\tilde{\mu} = \arg\max_{\mu \in B_t} \mu \cdot A_t$, from the action selection rule of LinUCB (consider how A_t is selected),

$$\tilde{\mu} \cdot A_t = \max_{a \in D} \max_{\mu \in B_t} \mu \cdot a \geq (\mu^*) \cdot a^* \tag{73}$$

naturally,

$$\mu^* \cdot a^* - \mu^* \cdot A_t \leq (\tilde{\mu} - \mu^*) \cdot A_t \tag{74}$$

notice that $\tilde{\mu}, \mu^* \in B_t$ are both close enough to $\hat{\mu}_t$, so this quantity shall have an upper bound given by

$$\mu^* \cdot a^* - \mu^* \cdot A_t \leq (\tilde{\mu} - \hat{\mu}_t) \cdot A_t + (\hat{\mu}_t - \mu^*) \cdot A_t \tag{75}$$

Now the question turns into bounding $(\alpha - \hat{\mu}_t) \cdot A_t$ for $\forall \alpha \in B_t$, which is a simple linear algebra question, one just have to create connection with Σ_t that determines the shape of B_t

$$\forall \alpha \in B_t, |(\alpha - \hat{\mu}_t) \cdot A_t| = |(\alpha - \hat{\mu}_t)^T \Sigma_t^{\frac{1}{2}} \Sigma_t^{-\frac{1}{2}} A_t| \tag{76}$$

$$\leq \left\| \Sigma_t^{\frac{1}{2}} (\alpha - \hat{\mu}_t) \right\| \left\| \Sigma_t^{-\frac{1}{2}} A_t \right\| \leq \sqrt{\beta_t} \sqrt{A_t^T \Sigma_t^{-1} A_t} = \sqrt{\beta_t} w_t \tag{77}$$

Back on our estimation above, now we proved

$$\mu^* \cdot a^* - \mu^* \cdot A_t \leq 2\sqrt{\beta_t} w_t \leq 2\sqrt{\beta_T} w_t \tag{78}$$

since β_t is increasing in t and exceeds 1. The other bound $\mu^* \cdot a^* - \mu^* \cdot A_t \le 2 \le 2\sqrt{\beta_T}$ is just trivial since the action value (mean of reward) has magnitude no larger than 1 in our assumption. □

Remark. *Recall that positive definite matrix Σ induces a norm*

$$\|x\|_{\Sigma^{-1}} = \sqrt{x^T\Sigma^{-1}x} = \left\|\Sigma^{-\frac{1}{2}}x\right\|_2 \tag{79}$$

which is the normalized radius in the ellipsoid $\left\{y : y^T\Sigma y = 1\right\}$ in the direction of x. This provides intuitive interpretation of our bound $2\sqrt{\beta_t}w_t$ with scaling factor $2\sqrt{\beta_t}$.

Now that we have made at least some progress, saying, cumulative regret satisfies

$$R_T \le \sqrt{4\beta_T T\sum_{t=1}^T \min\left\{w_t^2, 1\right\}} \le \sqrt{8\beta_T T\sum_{t=1}^T \log(1+w_t^2)} \tag{80}$$

with a logarithm bounds applies on $[0, 1]$, but where does the product $\prod_{t=1}^T(1 + w_t^2)$ come from? If one recalls the construction of Σ_t in LinUCB with increments as outer products, here is where it works.

$$\begin{aligned}
\det\Sigma_{t+1} &= \det(\Sigma_t + A_tA_t^T) &(81)\\
&= \det\Sigma_t^{\frac{1}{2}} \det(I + \Sigma_t^{-\frac{1}{2}}A_tA_t^T\Sigma_t^{-\frac{1}{2}}) \det\Sigma_t^{\frac{1}{2}} &(82)\\
&= \det\Sigma_t \det(I + A_t^T\Sigma_t^{-\frac{1}{2}}\Sigma_t^{-\frac{1}{2}}A_t) &(83)\\
&= \det\Sigma_t \cdot (1 + w_t^2) &(84)
\end{aligned}$$

where we used the well-known fact in linear algebra proved by the technique of blocked matrix that

$$\det(I + v^Tv) = \det(I + vv^T) \tag{85}$$

as a result

$$\frac{\det\Sigma_{T+1}}{\det\Sigma_1} = \prod_{t=1}^T(1 + w_t^2) \tag{86}$$

so the problem finally turns into bounding

$$\log\frac{\det\Sigma_{T+1}}{\det\Sigma_1} \tag{87}$$

formed as another lemma below.

Lemma 4. *For action sequence $A_1, ..., A_T$ such that $\forall t, \|A_t\| \leq B$,*

$$\log \frac{\det \Sigma_{T+1}}{\det \Sigma_1} \leq d \log \left(1 + \frac{TB^2}{d\lambda}\right) \tag{88}$$

Proof. It's again simple linear algebra

$$\log \frac{\det \Sigma_{T+1}}{\det \Sigma_1} = \log \det \Sigma_1^{-\frac{1}{2}} \Sigma_{T+1} \Sigma_1^{-\frac{1}{2}} = \log \det \left(I + \sum_{t=1}^{T} \Sigma_1^{-\frac{1}{2}} A_t A_t^T \Sigma_1^{-\frac{1}{2}}\right) \tag{89}$$

$$= \log \det \left(I + \frac{1}{\lambda} \sum_{t=1}^{T} A_t A_t^T\right) = \log \prod_{i=1}^{d} \left(1 + \frac{1}{\lambda} p_i\right) \tag{90}$$

where $p_1, ..., p_d$ are non-negative eigenvalues of $\sum_{t=1}^T A_t A_t^T$, the product can be formed as a geometric mean, bounded by arithmetic mean

$$\log \prod_{i=1}^{d} \left(1 + \frac{1}{\lambda} p_i\right) = d \log \left[\prod_{i=1}^{d} \left(1 + \frac{1}{\lambda} p_i\right)\right]^{\frac{1}{d}} \tag{91}$$

$$\leq d \log \frac{\sum_{i=1}^d \left(1 + \frac{1}{\lambda} p_i\right)}{d} \tag{92}$$

$$= d \log \left(1 + \frac{\sum_{i=1}^d p_i}{\lambda d}\right) \tag{93}$$

$$\leq d \log \left(1 + \frac{TB^2}{\lambda d}\right) \tag{94}$$

since the sum of eigenvalues is the trace and

$$tr\left(\sum_{t=1}^{T} A_t A_t^T\right) = \sum_{t=1}^{T} tr(A_t^T A_t) \leq TB^2 \tag{95}$$

□

Let's provide the remaining part of the proof of the theorem that cumulative regret

$$R_T \leq \sqrt{8\beta_T T \sum_{t=1}^{T} \log(1 + w_t^2)} \leq \sqrt{8\beta_T T d \log\left(1 + \frac{TB^2}{d\lambda}\right)} \tag{96}$$

$$\leq \sqrt{8Td\sigma^2 \left(2 + 4d \log\left(1 + \frac{TB^2W^2}{d\sigma^2}\right) + 8 \log \frac{4}{\delta}\right) \log\left(1 + \frac{W^2TB^2}{d\sigma^2}\right)} \tag{97}$$

$$\leq C\sigma\sqrt{T}\left(d \log\left(1 + \frac{TB^2W^2}{d\sigma^2}\right) + \log \frac{4}{\delta}\right) = \tilde{O}(d\sqrt{T}) \tag{98}$$

with the formula for λ, β_t plugged in, which provides the same bound as stated in the theorem above.

Remark. *The theorem is proved now on assuming that $\forall t, \mu^* \in B_t$, i.e. all confidence regions contain the underlying unknown μ^*. As a result, to truly conclude the proof, we need to make a concentration argument to show that with probability at least $1-\delta$, the event $\forall t, \mu^* \in B_t$ really happens.*

However, the main part of the proof has already been presented in a clear way. The connection with w_t and the log-determinant "potential function" selected to bound the sum of width in the ellipsoid are the key takeaways. It can be seen that all the objects in LinUCB are designed very carefully and subtly such that each object plays its own crucial role and they work together perfectly.

Finally, it comes to proving the concentration statement that we have made at the very beginning. Here we directly use a concentration inequality stated as a lemma below without providing its proof. Keep in mind that concentration inequalities are merely tools for us to use, we just need to understand the place where concentration inequalities help us, the proof itself is not of much concern, on the other hand.

Lemma 5 (Self-normalized Bound for Vector-valued Martingale (Abbasi-Yadkori et al. 2011))**.** *Consider the natural filtration $\{\mathscr{F}_n\}$ generated by stochastic process $\{\varepsilon_n\}$ such that each ε_n is a martingale difference, i.e. $\mathbb{E}(\varepsilon_n|\mathscr{F}_{n-1}) = 0$ with ε_n to be conditionally σ sub-Gaussian. Let $\{X_n\}$ be a vector-valued predictable stochastic process , i.e. $X_n \in \mathscr{F}_{n-1}$, and Σ_1 as a positive definite matrix with*

$$\Sigma_t = \Sigma_1 + \sum_{i=1}^{t} X_i X_i^T \tag{99}$$

then with probability at least $1-\delta$,

$$\forall t \geq 1, \left\| \sum_{i=1}^{t} X_i \varepsilon_i \right\|_{\Sigma_t^{-1}}^2 \leq \sigma^2 \log \frac{\det \Sigma_t \det \Sigma_1^{-1}}{\delta^2} \tag{100}$$

We provide the final part of the proof of the theorem above

Proof of the theorem above. Clearly since the noise sequence $\{\eta_t\}$ is σ sub-Gaussian, we shall make it the $\{\varepsilon_t\}$ in the lemma above. Since the action sequence is derived in LinUCB from all the information until the previous time step, $A_t \in \mathscr{F}_{t-1}$ holds, with probability at least $1-\delta$, we have

$$\forall t \geq 1, \left\| \sum_{i=1}^{t} \eta_i A_i \right\|_{\Sigma_t^{-1}}^2 \leq \sigma^2 \log \frac{\det \Sigma_t \det \Sigma_1^{-1}}{\delta^2} \tag{101}$$

Our goal is to argue that the confidence region always contains μ^*, thus it's natural to consider the following quantity

$$\begin{aligned}
\hat{\mu}_t - \mu^* &= \Sigma_t^{-1}\sum_{i=1}^{t-1} R_i A_i - \mu^* = \Sigma_t^{-1}\sum_{i=1}^{t-1}(\eta_i + \mu^*\cdot A_i)A_i - \mu^* && (102)\\
&= \Sigma_t^{-1}\sum_{i=1}^{t-1}\eta_i A_i + \Sigma_t^{-1}\sum_{i=1}^{t-1} A_i A_i^T \mu^* - \mu^* && (103)\\
&= \Sigma_t^{-1}\sum_{i=1}^{t-1}\eta_i A_i - \lambda\Sigma_t^{-1}\mu^* && (104)
\end{aligned}$$

as a result, with probability at least $1-\delta$,

$$\begin{aligned}
\forall t\geq 1, \sqrt{(\hat{\mu}_t-\mu^*)^T\Sigma_t(\hat{\mu}_t-\mu^*)} &= \left\|\Sigma_t^{\frac{1}{2}}(\hat{\mu}_t-\mu^*)\right\| && (105)\\
&\leq \left\|\Sigma_t^{-\frac{1}{2}}\sum_{i=1}^{t-1}\eta_i A_i\right\| + \lambda\left\|\Sigma_t^{-\frac{1}{2}}\mu^*\right\| && (106)\\
&\leq \sqrt{\sigma^2\log\frac{\det\Sigma_{t-1}\det\Sigma_1^{-1}}{\delta^2}} + \lambda\left\|\Sigma_t^{-\frac{1}{2}}\right\|\|\mu^*\| && (107)\\
&\leq \sqrt{\sigma^2\log\frac{\det\Sigma_{t-1}\det\Sigma_1^{-1}}{\delta^2}} + \sqrt{\lambda}W && (108)\\
&\leq \sqrt{\sigma^2\log\frac{\det\Sigma_{t-1}\det\Sigma_1^{-1}}{\delta^2}} + \sigma && (109)\\
&\leq \sqrt{\sigma^2\left(2+4d\log\left(1+\frac{tB^2W^2}{d\sigma^2}\right)+8\log\frac{4}{\delta}\right)} = \beta_t && (110)
\end{aligned}$$

concludes the proof. □

References for Bandit Problem

The theory of bandit problem:

- Online Least Squares Estimation with Self-Normalized Processes: An Application to Bandit Problems by Abbasi-Yadkori et al.
- Stochastic Linear Optimization under Bandit Feedback by Dani et al.
- Improved Algorithms for Linear Stochastic Bandits by Abbasi-Yadkori et al.
- Contextual Bandits with Linear Payoff Functions by Wei Chu et al.

The application of bandit algorithm:

- (Recommendation System) A Contextual-Bandit Approach to Personalized News Article Recommendation by Lihong Li et al.
- (Recommendation System) Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms by Lihong Li et al.
- (Hyperparameter Optimization) Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization by Lisha Li et al.
- (Hyperparameter Optimization) Provably Efficient Online Hyperparameter Optimization with Population-Based Bandits by Jack Parker-Holder et al.

Markov Decision Process (MDP)

In the last section, we have seen bandit problems. For multi-armed (finite) bandit problem, there is no state process, finitely many actions, the reward is generated depending on the action. For linear bandit problem, there is one unknown deterministic state μ^* with infinitely many actions, the reward is generated depending both on the state and the action. For more general contextual bandit problem, there are infinitely many states with infinitely many actions, the reward is generated depending both on the state and the action.

In this sense one might be wondering what are reinforcement learning (RL) problems and why RL problem is more general than the bandit problems. The key difference is the existence of state transition caused by the action. Likewise, the state and action jointly affects the amount of reward the agent receives. In this situation, the agent is required to think in a long-term way instead of maximizing his immediate reward at each time step (the failure of pointwise greedy strategy).

Basic Formulation of MDP

The Markov decision process (MDP) is the theoretical foundation of RL that models the agent-environment interaction. Recall the fundamental of RL problem that on seeing the current state, the agent makes an action that changes the future state and the reward he might get.

The value of the state is denoted $s \in \mathscr{S}$, with $\mathscr{S}$ to be the state space. The action availble might depend on the current state, and is denoted $a \in \mathscr{A}(s)$, with $\mathscr{A}(s)$ to be the set of all possible actions given that the agent observes state s. The value of the reward is denoted $r \in \mathscr{R} \subset \mathbb{R}$. Different from the literature of bandit problems where one assumes a randomized reward, **the reward in RL problem is typically assumed to be non-randomized**, i.e. $r = r(s, a)$ for some deterministic given reward function r. In other words, the randomness in the reward completely comes from the state and action.

MDP can be interpreted as a sequential decision-making process. The simulation of MDP generates a sequence $S_0, A_0, R_1, S_1, A_1, R_2, ...$ of states, actions and rewards. It means that the agent first starts with state S_0, takes action A_0, receives reward R_1 based on S_0, A_0. At the next time step, his action A_0 changes the environment and results in a state transition from S_0 to S_1. Facing the new state S_1, he takes action A_1 and receives reward R_2 based on S_1, A_1 etc. For the notation, we remind the readers that R_{t+1} is actually the immediate reward after taking the action A_t.

Most of our discussion focuses on **finite discrete-time MDP on infinite time horizon**, for which the sets of states, actions are finite, the time is discrete taking value in $\mathbb{N} = \{0, 1, 2, ...\}$. We denote the set of all possible actions as $\mathscr{A} = \bigcup_{s \in \mathscr{S}} \mathscr{A}(s)$, not depending on the state any longer but it remains to be finite. The dynamics of MDP tells us how states, actions and rewards interact with each other, and it's assumed to be **time homogeneous**, i.e. the transition rule does not depend on the current time t,

$$p(s', r|s, a) \overset{def}{=} \mathbb{P}\left(S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\right) \tag{111}$$

The "Markov" in MDP refers to the fact that when $\{A_t\}$ is given, $\{S_t\}$ is a Markov chain. It's clear that the distribution of a Markov chain is determined iff both the initial state S_0 and the transition kernel p is given. We represent some of the useful probability and expectation below in terms of the transition kernel. The state-transition probability is given by

$$p(s'|s,a) \overset{def}{=} \mathbb{P}\left(S_t = s'|S_{t-1} = s, A_{t-1} = a\right) \tag{112}$$

$$= \sum_r \mathbb{P}\left(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\right) \tag{113}$$

$$= \sum_r p(s', r|s, a) \tag{114}$$

the expected reward for a certain state-action pair is given by

$$\mathbb{E}\left(R_t|S_{t-1} = s, A_{t-1} = a\right) = \sum_r r \cdot \mathbb{P}\left(R_t = r|S_{t-1} = s, A_{t-1} = a\right) \tag{115}$$

$$= \sum_r r \cdot \sum_{s'} p(s', r|s, a) \tag{116}$$

the expected rewards for state-action-next-state triple is given by

$$\mathbb{E}\left(R_t|S_{t-1} = s, A_{t-1} = a, S_t = s'\right) = \sum_r r \cdot \mathbb{P}\left(R_t = r|S_{t-1} = s, A_{t-1} = a, S_t = s'\right) \tag{117}$$

$$= \sum_r r \cdot \frac{p(s', r|s, a)}{\mathbb{P}\left(S_t = s'|S_{t-1} = s, A_{t-1} = a\right)} \tag{118}$$

$$= \frac{\sum_r r \cdot p(s', r|s, a)}{\sum_r p(s', r|s, a)} \tag{119}$$

Discounted Construction of Return

What do we want to optimize in RL problem? We want to maximize the expected aggregated reward, similar to that in bandit problem. However, we are in the infinite time horizon setting so it's very easy for $\sum_{t=0}^{\infty} R_t$ to explode. As a result, a discount rate $\gamma \in (0, 1)$ is introduced and our objective function is rewritten in the discounted form

$$\sum_{t=0}^{\infty} \gamma^t R_{t+1} \tag{120}$$

the convergence of this series is ensured if $\{R_t\}$ is bounded. The **return** at a certain time t describes the discounted aggregated reward one gets after time t (include time t), denoted

$$G_t \overset{def}{=} \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{121}$$

the return can be written in the incremental form

$$G_t = R_{t+1} + \gamma G_{t+1} \tag{122}$$

so it can be easily updated after a new reward is observed.

The motivation of the definition of return might be unclear for those readers who don't have a background in control theory. We shall see later that RL problem can be formalized into value functions and solved backward in time, which have the natural appearance of return in its definition. The reader shall notice at this point that reward is immediately paid at each time step, so only focusing on the reward makes one short-sighted. On the other hand, return has the structure of long-term aggregated reward which matches the objective function in RL (G_0).

Policy

After getting the main idea about modelling the environment with MDP and the objective of RL, it remains a problem how the agent takes action based on the current state. One of the ideas is to formulate $A_t = \pi(S_t)$ for some deterministic function π, i.e. the action A_t is known and fixed after the state S_t is observed. However, such formulation results in a high rigidity in the state-action-reward sequence generated by a simulation of MDP, i.e. action never varies if the current state stays the same. Recall the consequence of the lack in exploration we have mentioned in the context of bandit problem, such formulation puts too much restrictions on exploration and is not the ideal one to adopt in RL.

As a result, an ideal formulation of action selection shall be randomized on observing the current state. This is exactly the motivation of the definition of **policy** in RL problem. The policy π is defined as a conditional distribution,

$$\pi(a|s) = \mathbb{P}\left(A_t = a|S_t = s\right) \tag{123}$$

indicating the probability of taking action a given the current state to be s. For each possible state value $s \in \mathscr{S}$, policy π maps it to a probability distribution $\pi(\cdot|s)$ on the action space $\mathscr{A}$. By doing this, we are allowing the agent to take a time-homogeneous feedback strategy with exploration intrinsically integrated. At this point, the law of MDP is completely determined if a policy is specified. As an example, on seeing the current state S_t and sticking to policy π, the expectation of R_{t+1} can be represented

$$\begin{aligned}
\mathbb{E}(R_{t+1}|S_t = s) &= \sum_r r \cdot \mathbb{P}\left(R_{t+1} = r|S_t = s\right) && (124)\\
&= \sum_r r \cdot \sum_a \mathbb{P}\left(A_t = a|S_t = s\right)\mathbb{P}\left(R_{t+1} = r|S_t = s, A_t = a\right) && (125)\\
&= \sum_r r \cdot \sum_a \pi(a|s) \cdot \sum_{s'} p(s', r|s, a) && (126)\\
& && (127)
\end{aligned}$$

Value Function for Policy π

The value function comes from the context of game theory and control problems to solve out the optimal feedback (Markovian) control. The key idea of value function is to think backwardly in time instead of thinking forwardly. If we want to make a sequence of optimal actions during a certain time period, it's always hard to think about what's the first action to take since the first action might have a great impact in the future states and rewards. However, if one thinks backwardly in time, that's a much easier problem since the optimal action at time $t, t+1, ...$ must be consistent with the optimal action at time $t+1, t+2, ...$ if our randomized action A_t has a Markovian feedback form.

Let's explain the spirit of value function in more details. If now we are at time t and we want to figure out an optimal action sequence $A_t, A_{t+1}, ...$ to maximize our return G_t (called the optimization at time t), backward thinking tells us to first consider another problem as if we are at time $t+1$ and we want to figure out an optimal action sequence $A_{t+1}, A_{t+2}, ...$ to maximize our return G_{t+1} (called the optimization at time $t+1$). Assume the optimization at time $t+1$ is solved to get the optimal feedback function $\hat{\pi}^{t+1}$ and the optimization at time t is solved to get the optimal feedback function $\hat{\pi}^t$.

Remark. *Notice that the action A_t changes as the state S_t changes, and their connection is completely through the deterministic feedback function $\pi : \mathscr{S} \to \mathscr{P}(\mathscr{A})$. As a result, what we are optimizing is essentially the feedback function π, not the values of the action! One must well understand this point before proceeding deeper into RL. One can refer to the difference between open-loop and feedback control in control theory.*

An intuitive observation is that the two feedback functions produce action sequences that are as good as each other starting from time $t+1$. The proof is simple in words. If $(\hat{\pi}^t(S_{t+1}), \hat{\pi}^t(S_{t+2}), ...)$ is strictly better than $\left(\hat{\pi}^{t+1}(S_{t+1}), \hat{\pi}^{t+1}(S_{t+2}), ...\right)$ for the same given S_{t+1}, consider

$$\left(\hat{\pi}^t(S_{t+1}), \hat{\pi}^t(S_{t+2}), ...\right) \tag{128}$$

as an action sequence for the optimization at time $t+1$, a contradiction. Conversely, if $\left(\hat{\pi}^{t+1}(S_{t+1}), \hat{\pi}^{t+1}(S_{t+2}), ...\right)$ is strictly better than $(\hat{\pi}^t(S_{t+1}), \hat{\pi}^t(S_{t+2}), ...)$, consider

$$\left(\hat{\pi}^t(S_t), \hat{\pi}^{t+1}(S_{t+1}), \hat{\pi}^{t+1}(S_{t+2}), ...\right) \tag{129}$$

as an action sequence for the optimization at time t, it produces strictly better results than $(\hat{\pi}^t(S_t), \hat{\pi}^t(S_{t+1}), \hat{\pi}^t(S_{t+2}), ...)$ for the same given S_t, a contradiction. Although this example is far from a rigorous proof, the advantage of thinking backwardly in a Markovian control problem is quite obvious.

Remark. *The argument above can be extended to a time-inhomogeneous feedback function and formalized into* ***the dynamic programming principle (DPP) for value function****. As we can see, the original optimization problem can always be decomposed into problems of the same type but with smaller scale. This matches the idea of dynamic programming (DP) algorithm. As a result, we shall expect to see DP algorithms when it comes to the numerical solution of RL problems.*

From the motivation of value function introduced above, the value function shall have a backward structure in time with its value depending on the policy π one picks. The **state value function for policy** π is defined as

$$v_\pi(s) \overset{def}{=} \mathbb{E}_\pi(G_t|S_t = s) = \mathbb{E}_\pi\left(\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle|S_t = s\right) \tag{130}$$

while **the action value function for policy** π is defined as

$$q_\pi(s,a) \overset{def}{=} \mathbb{E}_\pi(G_t|S_t = s, A_t = a) = \mathbb{E}_\pi\left(\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle|S_t = s, A_t = a\right) \tag{131}$$

those two value functions are typically called the V and Q function respectively in RL literature (if one has possibly seen Q-learning or V-learning before).

The two value functions for the same policy π has close connections. v_π can be represented in terms of q_π

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi(G_t|S_t = s) && (132)\\ &= \sum_a \mathbb{P}_\pi\left(A_t = a|S_t = s\right)\mathbb{E}_\pi(G_t|S_t = s, A_t = a) && (133)\\ &= \sum_a \pi(a|s)\cdot q_\pi(s,a) && (134)\end{aligned}$$

q_π can also be represented in terms of v_π

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi(G_t|S_t = s, A_t = a) && (135)\\ &= \mathbb{E}_\pi(R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a) && (136)\\ &= \mathbb{E}_\pi(R_{t+1}|S_t = s, A_t = a) + \gamma\mathbb{E}_\pi(G_{t+1}|S_t = s, A_t = a) && (137)\\ &= \sum_r r\cdot\sum_{s'} p(s',r|s,a) + \gamma\sum_{s'}\mathbb{E}_\pi(G_{t+1}|S_{t+1} = s', S_t = s, A_t = a)\mathbb{P}\left(S_{t+1} = s'|S_t = s, A_t = a\right) && (138)\\ &= \sum_r r\cdot\sum_{s'} p(s',r|s,a) + \gamma\sum_{s'} v_\pi(s')\sum_r p(s',r|s,a) && (139)\\ &= \sum_{r,s'}[r + \gamma\cdot v_\pi(s')]\cdot p(s',r|s,a) && (140)\end{aligned}$$

where the equality $\mathbb{E}_\pi(G_{t+1}|S_{t+1} = s', S_t = s, A_t = a) = \mathbb{E}_\pi(G_{t+1}|S_{t+1} = s')$ is due to the Markov property.

Since current time t appears inside the definition of v_π, q_π, it's natural to ask if v_π, q_π have dependence on t. The answer is NO since the state process is a time-homogeneous Markov chain and our policy is also time-homogeneous. The proof of this fact is left to the readers.

Remark. *Keep in mind that both value functions have the structure as conditional expectation. When it comes to numerically estimating v_π, q_π, it's thus natural to think about Monte Carlo which only requires experience but not any knowledge of the underlying model. As a result, Monte Carlo methods will be the first model-free methods we will introduce in a later context.*

Bellman Consistency Equation

The Bellman consistency equation works as characterizations of v_π, q_π and is crucial for the estimation of value functions. The Bellman consistency equation for v_π is given below

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi(G_t|S_t = s) = \mathbb{E}_\pi(R_{t+1}|S_t = s) + \gamma\mathbb{E}_\pi(G_{t+1}|S_t = s) && (141)\\
&= \sum_r r \cdot \mathbb{P}_\pi\left(R_{t+1} = r|S_t = s\right) + \gamma\sum_{s',a}\mathbb{P}_\pi\left(S_{t+1} = s', A_t = a|S_t = s\right)\mathbb{E}_\pi(G_{t+1}|S_{t+1} = s', S_t = s, A_t = a) && (142)\\
&= \sum_r r\sum_a \pi(a|s)\sum_{s'} p(s', r|s, a) + \gamma\sum_{s',a}\mathbb{P}\left(S_{t+1} = s'|S_t = s, A_t = a\right)\mathbb{P}_\pi\left(A_t = a|S_t = s\right)v_\pi(s') && (143)\\
&= \sum_r r\sum_a \pi(a|s)\sum_{s'} p(s', r|s, a) + \gamma\sum_{s',a}\sum_r p(s', r|s, a)\cdot\pi(a|s)\cdot v_\pi(s') && (144)\\
&= \sum_a \pi(a|s)\sum_{s',r} p(s', r|s, a)[r + \gamma\cdot v_\pi(s')] && (145)
\end{aligned}$$

the interpretation of this equation can be provided through the observation that the RHS consists of the sum of two terms, the one containing r as immediate reward and the one containing $\gamma \cdot v_\pi(s')$ as the discounted future reward.

Similarly, a Bellman consistency equation can be derived for q_π using the relationship between v_π and q_π proved above.

$$\begin{aligned}
q_\pi(s, a) &= \sum_{r,s'}(r + \gamma\cdot v_\pi(s'))p(s', r|s, a) && (146)\\
&= \sum_{r,s'} p(s', r|s, a)\left[r + \gamma\sum_{a'}\pi(a'|s')q_\pi(s', a')\right] && (147)
\end{aligned}$$

with policy π and MDP dynamics p given, the two Bellman consistency equations are both linear systems in terms of the value function. As a result, v_π, q_π can both be computed exactly for an RL problem where $|\mathscr{S}|, |\mathscr{A}|$ are relatively small.

But wait, is the solution to the Bellman consistency equation necessarily unique? If the uniqueness of the solution fails, we have to identify which solution is the true value function so the Bellman consistency equation would not be a useful characterization of the value function. Luckily, the solution to Bellman consistency equation always has **existence and uniqueness** property.

Let's prove here only for v_π and the case for q_π is left to the readers. Assume $\mathscr{S} = \{s_1, ..., s_n\}$, and denote $v = [v_\pi(s_1), ..., v_\pi(s_n)]^T$ we rewrite the Bellman consistency equation in matrix form $Av = b$ where

$$A = \begin{bmatrix}
\gamma\sum_a \pi(a|s_1)p(s_1|s_1, a) - 1 & \gamma\sum_a \pi(a|s_1)p(s_2|s_1, a) & ... & \gamma\sum_a \pi(a|s_1)p(s_n|s_1, a)\\
\gamma\sum_a \pi(a|s_2)p(s_1|s_2, a) & \gamma\sum_a \pi(a|s_2)p(s_2|s_2, a) - 1 & ... & \gamma\sum_a \pi(a|s_2)p(s_n|s_2, a)\\
... & & & \\
\gamma\sum_a \pi(a|s_n)p(s_1|s_n, a) & \gamma\sum_a \pi(a|s_n)p(s_2|s_n, a) & ... & \gamma\sum_a \pi(a|s_n)p(s_n|s_n, a) - 1
\end{bmatrix} \quad (148)$$

it suffices to prove that A has full rank. Notice that

$$\begin{aligned}
&\left|\gamma\sum_a \pi(a|s_1)\cdot p(s_2|s_1,a)\right| + ... + \left|\gamma\sum_a \pi(a|s_1)\cdot p(s_n|s_1,a)\right| && (149)\\
&= \gamma\sum_a \pi(a|s_1)[1-p(s_1|s_1,a)] && (150)\\
&= \gamma - \gamma\sum_a \pi(a|s_1)\cdot p(s_1|s_1,a) && (151)\\
&< 1-\gamma\sum_a \pi(a|s_1)\cdot p(s_1|s_1,a) = \left|\gamma\sum_a \pi(a|s_1)\cdot p(s_1|s_1,a)-1\right| && (152)
\end{aligned}$$

matrix A shows strict diagonal dominance. Gershgorin circle theorem tells us that every eigenvalue of A lies within at least one of the Gershgorin disks (on the complex plane) centered at A_{ii} with radius $\sum_{j\neq i}|A_{ij}|$ for $i\in\{1,2,...,n\}$. Matrix A has no zero eigenvalue due to its strict diagonal dominance so it has full rank, this concludes the proof.

Optimal Policy and Optimal Value Functions

In RL problem, the agent interact with the environment by specifying a policy π to generate actions, and our goal is to find the "best" policy that generates the maximum amount of return. A natural question to ask is that how do we compare two different policy and if the "best" policy exists.

Since we have already defined the state value function v_π, it's intuitive to say that for two policy π, π', π' is no worse than π iff $\forall s \in \mathscr{S}, v_\pi(s) \leq v_{\pi'}(s)$. This provides a partial ordering on the space of policy Π denoted as

$$\pi \leq \pi' \iff \forall s \in \mathscr{S}, v_\pi(s) \leq v_{\pi'}(s) \tag{153}$$

the partial ordering holds iff $v_{\pi'}$ dominates v_π on every state s pointwisely. Clearly, if there exists a policy $\pi^* \in \Pi$ such that $\forall s \in \mathscr{S}, v_{\pi^*}(s) = \sup_{\pi\in\Pi} v_\pi(s)$, then such policy π^* is the optimal policy and v_{π^*} is expected to be the optimal value function as a pointwise supreme of all v_π. However, the existence of this optimal policy π^* is totally unclear at this point. Before arguing the existence of optimal policy, we first define the **optimal state/action value function** as a pointwise supreme

$$\forall s \in \mathscr{S}, \forall a \in \mathscr{A}, v^*(s) = \sup_{\pi\in\Pi} v_\pi(s), q^*(s,a) = \sup_{\pi\in\Pi} q_\pi(s,a) \tag{154}$$

and then argue that this pointwise supreme can actually be attained by some $\pi^* \in \Pi$.

Remark. *The difficulty in the existence of π^* lies in the difference between global optimality and local optimality. It's clear that the optimal value function defined above is in the sense of local optimality since supreme is taken pointwisely. However, the partial ordering on Π is in the sense of global optimality that π^* shall dominate the performance of any policy π. Hence, it shall be a little bit surprising to the readers that local and global optimality coincides under the setting of RL.*

More surprisingly, we shall show below that there even exists a deterministic version of this optimal policy π^* and its optimality is actually among the set of all path-dependent policy.

In the context below, we shift our gears temporarily to include path-dependent policy. To clarify, what we have defined above as a policy is actually a randomized Markovian policy that maps a state to a probability distribution on the action space

$$\pi : \mathscr{S} \to \mathscr{P}(\mathscr{A}) \tag{155}$$

so whenever S_i, S_j take the same value, $\pi(S_i), \pi(S_j)$ are two same probability distribution on the action space. The more general randomized path-dependent policy, on the other hand, allows the difference in probability distribution even if the current state is the same. To be more specific,

$$\begin{cases} A_0 \sim \pi(\cdot|S_0) \\ A_1 \sim \pi(\cdot|S_0, A_0, R_1, S_1) \\ ... \\ A_t \sim \pi(\cdot|S_0, A_0, R_1, ..., S_t) \end{cases} \tag{156}$$

the probability distribution of A_t not only depends on current state S_t but also depends on the history $S_0, A_0, R_1, ..., R_t$. Let Π denote the collection of all randomized path-dependent policy for the time being, we are going to show that the optimal policy π^* is Markovian and deterministic, but its optimality holds among the whole space Π.

Before entering the proof of the main theorem, we first prove a lemma saying that on knowing the state at a certain time, the history actually does not make any difference toward the optimal value.

Lemma 6. *Let* Π *denote the collection of all randomized path-dependent policy, then*

$$\forall s, a, r, s', \sup_{\pi\in\Pi} \mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^t R_{t+1} \middle| (S_0, A_0, R_1, S_1) = (s, a, r, s') \right] = \gamma \cdot v^*(s') \tag{157}$$

Proof. Consider the offset policy $\pi_{(s,a,r)}$ defined as

$$\pi_{(s,a,r)}(A_t|S_0 = s_0, A_0 = a_0, R_1 = r_1, ..., S_t = s_t) \tag{158}$$

$$= \pi(A_{t+1}|S_0 = s, A_0 = a, R_1 = r, S_1 = s_0, A_1 = a_0, R_2 = r_1, ..., S_{t+1} = s_t) \tag{159}$$

given that $S_0 = s, A_0 = a, R_1 = r$ has been observed. Simply speaking, $\pi_{(s,a,r)}$ is offsetting π by one time step to match the observation $S_0 = s, A_0 = a, R_1 = r$, but it follows exactly π starting from time 1.

Use the Markov property to translate one time step,

$$\sup_{\pi\in\Pi} \mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^t R_{t+1} \middle| (S_0, A_0, R_1, S_1) = (s, a, r, s') \right] = \sup_{\pi\in\Pi} \mathbb{E}_{\pi_{(s,a,r)}} \left[\sum_{t=1}^{\infty} \gamma^t R_t \middle| S_0 = s' \right] \tag{160}$$

notice that for $\forall s, a, r$, when π traverses through Π, the offset policy $\pi_{(s,a,r)}$ also traverses through Π so

$$\sup_{\pi\in\Pi} \mathbb{E}_{\pi_{(s,a,r)}} \left[\sum_{t=1}^{\infty} \gamma^t R_t \middle| S_0 = s' \right] = \gamma \sup_{\pi\in\Pi} \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \middle| S_0 = s' \right] = \gamma \cdot v^*(s') \tag{161}$$

concludes the proof. ☐

Theorem 3. ***(Existence of Optimal Policy)*** *There exists a* ***deterministic Markovian optimal policy*** π^* *such that*

$$\forall s \in \mathscr{S}, v_{\pi^*}(s) = v^*(s) \tag{162}$$

where the optimality holds on Π*, the collection of all* ***randomized path-dependent policy****.*

Proof. The deterministic Markovian optimal policy is constructed as

$$\tilde{\pi}(s) = \arg\sup_{a\in\mathscr{A}} \mathbb{E}\left[R_1 + \gamma \cdot v^*(S_1) | S_0 = s, A_0 = a\right] \tag{163}$$

i.e. when the current state is s, always put all probability mass on the action that maximizes the sum of immediate reward and discounted maximum future reward.

Now let's prove that it induces the optimal value function. Obviously $\forall s \in \mathscr{S}, v^*(s) \geq v_{\tilde{\pi}}(s)$. On the other hand,

$$\begin{aligned}
\forall s \in \mathscr{S}, v^*(s) &= \sup_{\pi\in\Pi} \mathbb{E}_\pi \left(R_1 + \mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^t R_{t+1} \Big| S_0 = s, A_0, R_1, S_1\right] \Big| S_0 = s\right) &(164)\\
&\leq \sup_{\pi\in\Pi} \mathbb{E}_\pi \left(R_1 + \sup_{\pi'\in\Pi} \mathbb{E}_{\pi'} \left[\sum_{t=1}^{\infty} \gamma^t R_{t+1} \Big| S_0 = s, A_0, R_1, S_1\right] \Big| S_0 = s\right) &(165)\\
&= \sup_{\pi\in\Pi} \mathbb{E}_\pi \left(R_1 + \gamma \cdot v^*(S_1) | S_0 = s\right) &(166)
\end{aligned}$$

from the lemma proved above. From the definition of $\tilde{\pi}$, it's clear that

$$\forall s \in \mathscr{S}, v^*(s) \leq \mathbb{E}_{\tilde{\pi}} \left(R_1 + \gamma \cdot v^*(S_1) | S_0 = s\right) \tag{167}$$

on the RHS there is still v^* so we can apply this inequality iteratively

$$\begin{aligned}
\forall s \in \mathscr{S}, v^*(s) &\leq \mathbb{E}_{\tilde{\pi}} \left(R_1 + \gamma \cdot \mathbb{E}_{\tilde{\pi}} \left(R_2 + \gamma \cdot v^*(S_2) | S_1\right) | S_0 = s\right) &(168)\\
&= \mathbb{E}_{\tilde{\pi}} \left(R_1 + \gamma \cdot R_2 + \gamma^2 \cdot v^*(S_2) | S_0 = s\right) \leq ... &(169)\\
&\leq v_{\tilde{\pi}}(s) &(170)
\end{aligned}$$

concludes the proof. □

Remark. *Obviously, optimal policy exists but is not necessarily unique since there might be multiple actions that attain* arg sup *in the definition of* $\tilde{\pi}$ *above. Nevertheless, we don't care about which optimal policy we derive that much as long as they share the same optimal value function.*

The theorem above depends on the fact that time horizon is infinite in our MDP so the agent faces exactly the same game at each time step. If this is not the case, e.g. the finite time horizon setting in bandit problems, Markovian policy is not guaranteed to be optimal among all path-dependent policy.

Coming back to the RL problem, we restrict ourselves back to Markovian policy π but not path-dependent policy as illustrated above. The theorem proves the fact that

$$v^* = v_{\pi^*} \tag{171}$$

it's thus natural to expect

$$q^* = q_{\pi^*} \tag{172}$$

also holds for action value function. The proof can be done through the connection between v_π and q_π. Recall that

$$\forall s \in \mathscr{S}, a \in \mathscr{A}, q_\pi(s,a) = \sum_{r,s'} (r + \gamma \cdot v_\pi(s')) \cdot p(s', r|s, a) \tag{173}$$

take sup w.r.t. π on both sides

$$\begin{aligned}
\forall s \in \mathscr{S}, a \in \mathscr{A}, q^*(s,a) &= \sum_{r,s'} r \cdot p(s', r|s, a) + \gamma \sup_\pi \sum_{r,s'} v_\pi(s') \cdot p(s', r|s, a) && (174)\\
&\leq \sum_{r,s'} r \cdot p(s', r|s, a) + \gamma \sum_{r,s'} \sup_\pi v_\pi(s') \cdot p(s', r|s, a) && (175)\\
&= \sum_{r,s'} [r + \gamma \cdot v_{\pi^*}(s')] \cdot p(s', r|s, a) && (176)\\
&= q_{\pi^*}(s,a) && (177)
\end{aligned}$$

by definition, $\forall s \in \mathscr{S}, a \in \mathscr{A}, q^*(s,a) \geq q_{\pi^*}(s,a)$, we conclude

$$q^* = q_{\pi^*} \tag{178}$$

In the following context, we always use v^*, v_{π^*} and q^*, q_{π^*} without any specification. This conclusion seems trivial but it actually means the equivalence between local and global optimality, an important structure of RL problem.

Bellman Optimality Equation

Keep in mind that in RL problem, the goal is to find the optimal policy π^* so that we can generate the optimal action sequence. However, calculating the exact optimal policy is hard and often impossible in practice. That's why people think of finding characterization for the optimal value functions v^*, q^* to see if it's possible to first estimate optimal value functions and then derive the optimal policy from the optimal value functions. Those characterizations for v^*, q^* are called Bellman optimality equations.

Since v^*, q^* are value functions for the optimal policy, one can directly state that they satisfy the Bellman consistency equations. Take $\pi = \pi_*$ in Bellman consistency equation to get

$$v^*(s) = \sum_a \pi^*(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma \cdot v^*(s')] \tag{179}$$

$$q^*(s, a) = \sum_{r,s'} p(s', r|s, a) \cdot \left[r + \gamma \sum_{a'} \pi^*(a'|s') \cdot q^*(s', a') \right] \tag{180}$$

as Bellman optimality equations. However, those equations are not useful since π^* is appearing. Recall that the reason we are finding characterizations for v^*, q^* is to solve out the optimal value functions without even knowing the optimal policy. How do we deal with this paradoxical situation?

Let's start with the relationship between v^* and q^* that

$$\begin{aligned}
\forall s \in \mathscr{S}, v^*(s) &= \sup_\pi v_\pi(s) && (181)\\
&= \sup_\pi \sum_a \pi(a|s) \cdot q_\pi(s, a) && (182)\\
&\le \sup_\pi \sum_a \pi(a|s) \cdot q^*(s, a) && (183)\\
&= \sup_a q^*(s, a) && (184)
\end{aligned}$$

the last equation is due to the fact that $\pi(\cdot|s)$ is a probability distribution, so for any policy π, $\sum_a \pi(a|s) \cdot q_*(s, a) \le \max_a q_*(s, a)$ and such upper bound can be attained.

Remark. *Let's check both sides of this inequality. LHS is $v^*(s)$, the optimal value starting from state s. RHS is $\sup_a q^*(s, a)$, first take the optimal value starting from state s, action a and then optimize w.r.t. action a. However, since we are currently under the deterministic optimal policy π^*, the optimization w.r.t. action a on the RHS shall give us exactly $\pi^*(s)$ (otherwise it's a contradiction with the optimality of π^*). As a result, we expect the inequality to be an actual equality here.*

Let's state the point we make inside the remark as the policy improvement theorem and then use this theorem to prove the equality.

Theorem 4 (Policy Improvement Theorem)**.** *Fix state s, if π, π' are any two deterministic policy such that $q_\pi(s, \pi'(s)) \ge v_\pi(s)$, then $v_{\pi'}(s) \ge v_\pi(s)$. Moreover, if the condition is a strict inequality, then the conclusion is also a strict inequality.*

Proof. Since π' is deterministic,

$$\mathbb{E}_{\pi'}[R_{t+1}+\gamma\cdot v_\pi(S_{t+1})|S_t=s]=\sum_a \pi'(a|s)\cdot\mathbb{E}_{\pi'}[R_{t+1}+\gamma\cdot v_\pi(S_{t+1})|S_t=s,A_t=a] \tag{185}$$

$$=\mathbb{E}[R_{t+1}+\gamma\cdot v_\pi(S_{t+1})|S_t=s,A_t=\pi'(s)] \tag{186}$$

start from the condition provided

$$v_\pi(s)\le q_\pi(s,\pi'(s)) \tag{187}$$

$$=\mathbb{E}[R_{t+1}+\gamma\cdot v_\pi(S_{t+1})|S_t=s,A_t=\pi'(s)] \tag{188}$$

$$=\mathbb{E}_{\pi'}[R_{t+1}+\gamma\cdot v_\pi(S_{t+1})|S_t=s] \tag{189}$$

iteratively use the condition $v_\pi(s)\le q_\pi(s,\pi'(s))$ and the calculation for $q_\pi(s,\pi'(s))$ above,

$$\mathbb{E}_{\pi'}[R_{t+1}+\gamma\cdot v_\pi(S_{t+1})|S_t=s]\le\mathbb{E}_{\pi'}[R_{t+1}+\gamma\cdot q_\pi(S_{t+1},\pi'(S_{t+1}))|S_t=s] \tag{190}$$

$$=\mathbb{E}_{\pi'}\left[R_{t+1}+\gamma\cdot\mathbb{E}_{\pi'}\left(R_{t+2}+\gamma\cdot v_\pi(S_{t+2})|S_{t+1}\right)\Big|S_t=s\right] \tag{191}$$

$$=\mathbb{E}_{\pi'}\left[R_{t+1}+\gamma\cdot R_{t+2}+\gamma^2\cdot v_\pi(S_{t+2})\Big|S_t=s\right] \tag{192}$$

$$\le ... \tag{193}$$

$$\le\mathbb{E}_{\pi'}\left[R_{t+1}+\gamma R_{t+2}+\gamma^2 R_{t+3}+...\Big|S_t=s\right] \tag{194}$$

$$=\mathbb{E}_{\pi'}[G_t|S_t=s]=v_{\pi'}(s) \tag{195}$$

concludes the proof. □

Theorem 5 (Compact Form of Bellman Optimality Equation)**.**

$$\forall s\in\mathscr{S}, v^*(s)=\sup_a q^*(s,a) \tag{196}$$

Proof. Prove by contradiction, assume that $\exists s_0\in\mathscr{S}, v^*(s_0)>\sup_a q^*(s_0,a)$, then set $\pi=\pi^*,\pi'(s)=\arg\sup_a q^*(s,a)$ in the policy improvement theorem (there always exists a deterministic optimal policy), we conclude that

$$v_{\pi'}(s_0)>v^*(s_0) \tag{197}$$

a contradiction with the definition of v^*. □

Remark. *The policy improvement theorem provides important insight on how to improve a policy. Basically, if π is the policy we want to improve, define deterministic policy*

$$\forall s\in\mathscr{S},\pi'(s)=\arg\max_a q_\pi(s,a) \tag{198}$$

then π' is guaranteed to be no worse than π.

The compact form of Bellman optimality equation (BOE) enables us to characterize v^*, q^* without explicitly introducing π^*.

Theorem 6 (Bellman Optimality Equation)**.**

$$v^*(s) = \sup_a \sum_{s',r} p(s', r|s, a) \cdot [r + \gamma \cdot v^*(s')] \tag{199}$$

$$q^*(s, a) = \sum_{s',r} p(s', r|s, a) \cdot \left[r + \gamma \cdot \sup_{a'} q^*(s', a')\right] \tag{200}$$

Proof. From the compact form of BOE,

$$\begin{aligned} v^*(s) &= \sup_a q^*(s, a) &(201)\\ &= \sup_a \mathbb{E}_{\pi^*}[R_{t+1} + \gamma \cdot G_{t+1}|S_t = s, A_t = a] &(202)\\ &= \sup_a \sum_{s',r} p(s', r|s, a) \cdot [r + \gamma \cdot v^*(s')] &(203)\\ &= \sup_a \mathbb{E}[R_{t+1} + \gamma \cdot v^*(S_{t+1})|S_t = s, A_t = a] &(204) \end{aligned}$$

similarly

$$\begin{aligned} q^*(s, a) &= \mathbb{E}_{\pi^*}[R_{t+1} + \gamma \cdot G_{t+1}|S_t = s, A_t = a] &(205)\\ &= \sum_{s',r} p(s', r|s, a) \cdot [r + \gamma \cdot v^*(s')] &(206)\\ &= \sum_{s',r} p(s', r|s, a) \cdot \left[r + \gamma \cdot \sup_{a'} q^*(s', a')\right] &(207) \end{aligned}$$

concludes the proof. ☐

Remark. *BOE is a non-linear equation without referring to any specific policy. It remains a problem although if the solution to BOE always exists and is unique. Besides, a nice approach to solving BOE needs to be found due to its nonlinearity.*

Bellman Optimality Operator as Contraction Mapping

Let's check the existence and uniqueness of the solution to BOE. We only investigate the BOE for v^* and similar conclusion holds for q^*. Let's rewrite the BOE for v^* in terms of the Bellman optimality operator (BOO) T so that

v^* is the fixed point of this operator

$$\forall s \in \mathscr{S}, v^*(s) = Tv^*(s) \tag{208}$$
$$T : \mathbb{R}^{|\mathscr{S}|} \to \mathbb{R}^{|\mathscr{S}|} \tag{209}$$
$$Tv(s) = \sup_a \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s, A_t = a] \tag{210}$$

if T is a contraction mapping under some norm on $\mathbb{R}^{|\mathscr{S}|}$, then according to Banach fixed point theorem, the fixed point exists and is unique, proves the existence and uniqueness of the solution to BOE.

Theorem 7 (BOO as Contraction Mapping)**.** *Consider normed space* $(\mathbb{R}^{|\mathscr{S}|}, ||\cdot||_\infty)$*, and the operator* T *defined above, then* $\exists 0 \le k < 1$ *such that* $\forall v, v' \in \mathbb{R}^{|\mathscr{S}|}, ||Tv - Tv'||_\infty \le k||v - v'||_\infty$.

Proof.

$$||Tv - Tv'||_\infty = \sup_s |Tv(s) - Tv'(s)| \tag{211}$$
$$= \sup_s \left| \sup_a \mathbb{E}[R_{t+1} + \gamma \cdot v(S_{t+1}) | S_t = s, A_t = a] - \sup_a \mathbb{E}[R_{t+1} + \gamma \cdot v'(S_{t+1}) | S_t = s, A_t = a] \right| \tag{212}$$
$$\le \sup_{s,a} |\mathbb{E}[\gamma \cdot v(S_{t+1}) | S_t = s, A_t = a] - \mathbb{E}[\gamma \cdot v'(S_{t+1}) | S_t = s, A_t = a]| \tag{213}$$
$$= \gamma \sup_{s,a} \mathbb{E}\left(|v(S_{t+1}) - v'(S_{t+1})| \middle| S_t = s, A_t = a \right) \tag{214}$$
$$\le \gamma ||v - v'||_\infty \tag{215}$$

since $0 \le \gamma < 1$, it's a contraction mapping. □

At this point, we see that BOE is a nice characterization of optimal value functions. We will come back to this contraction mapping property later when we talk about DP methods solving RL problems.

An Example of BOE

Exercise 3.22 in Barto Sutton book provides a nice example for BOE. The MDP is provided in Fig. 7. There are three state, the only two policies for choice are π_l, π_r, the policy that always choose to go left and the policy that always choose to go right. For different values of γ, how does the optimal policy change?

Let's write out the Bellman optimality equation for this model first. The top state is state s_1, the state at the left lower corner is state s_2 and the state at the right lower corner is state s_3. When at s_2 or s_3, only one action can be chosen and when at state s_1, two actions can be chosen from.

$$v^*(s) = \max_a \sum_{s', r} p(s', r | s, a) \cdot [r + \gamma \cdot v^*(s')] \tag{216}$$

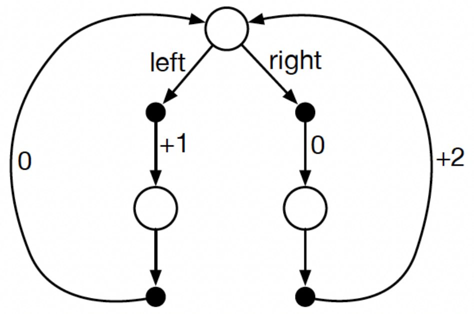


Figure 7: Another MDP

MDP with only two deterministic policies π_l, π_ra available.

for states s_2, s_3,

$$v^*(s_2) = \gamma \cdot v^*(s_1) \tag{217}$$

$$v^*(s_3) = 2 + \gamma \cdot v^*(s_1) \tag{218}$$

BOE for state s_1 contains a maximum

$$v^*(s_1) = \max\{1 + \gamma \cdot v^*(s_2), \gamma \cdot v^*(s_3)\} \tag{219}$$

find that the critical value of γ is such that

$$1 + \gamma^2 \cdot v^*(s_1) = 2\gamma + \gamma^2 \cdot v^*(s_1) \tag{220}$$

$$\gamma = \frac{1}{2} \tag{221}$$

When the discount rate $\gamma = \frac{1}{2}$, both actions achieves the maximum in the Bellman optimality equations, so both π_l and π_r are optimal policies. When $0 \leq \gamma < \frac{1}{2}$, π_l is the optimal policy. When $\frac{1}{2} < \gamma < 1$, π_r is the optimal policy.

Remark. *When solving RL problem, v_π and q_π for a specific policy π are not of main interest although they can be easily characterized by the linear Bellman consistency equation. On the other hand, how do we figure out the optimal policy after estimating the optimal value functions v^*, q^*?*

If q^* is known, the best actions to take at current state s is just ***the actions*** *a* ***that attains*** $\max_a q^*(s,a)$. *If v^* is known, the best action to take at current state s is* ***the action*** *a* ***that attains the maximum in the BOE****. Notice that the optimal policy constructed in those two ways are both deterministic. This would possibly cause a lack of exploration so we shall keep in mind to encourage exploration when those greedy policies are taken in the algorithm.*

A Toy Model: Gridworld

All possible states in this problem are the 25 grids in the 5×5 square and all possible actions at each state are going north, south, west, east. If we are already at the boundary of the square and choose the action that steps outside the square, the location is unchanged with a reward -1 (so it's not preferred) while all other actions have reward 0.

Nevertheless, there are two special grids $(1,4)$ and $(3,4)$ (row and column indices are similar to that in coordinate axis and starts from zero). At $(1,4)$, all actions have reward 10 and the state is transitioned immediately to $(1,0)$. At $(3,4)$, all actions have reward 5 and the state is transitioned immediately to $(3,2)$. Refer to Fig. 8 for a graph illustration of Gridworld. Gridworld is a continuing task and no specific ending criterion of the task is assumed.

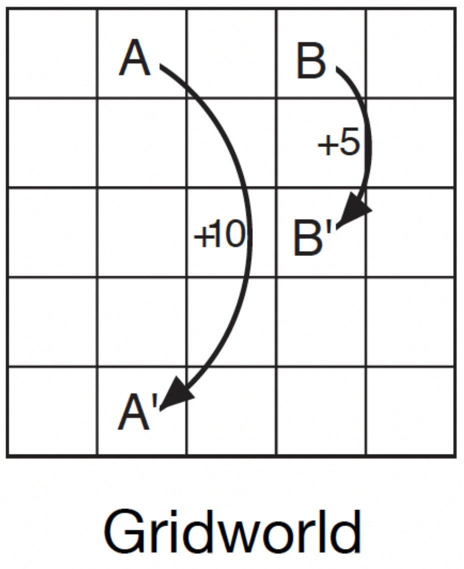
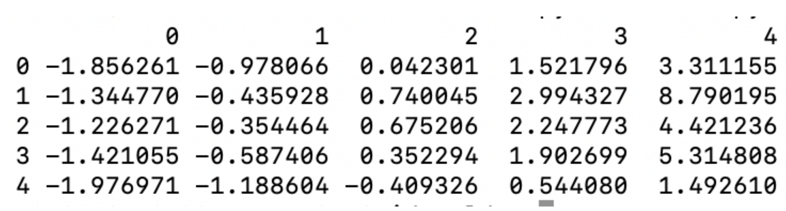


Figure 8: Graph illustration of the Gridworld model

Note that the x-coordinates increase from left to right and the y-coordinates increase from below to above. In other words, (0,0) stands for the most left-down block.

Let's investigate the state value function v_π for this model given the policy π to be the equiprobable policy, i.e. four actions always have $\frac{1}{4}$ of being taken at each state. Using Monte Carlo, we get the estimated v_π in Fig. 9.

As expected, state $(1,4)$ and $(3,4)$ has the highest state values among all states. The states next to $(1,4)$ and $(3,4)$ also benefit from the fact that they have larger probabilities of transiting into these two special states. When it comes to states far away from these two high-value states and next to the boundary, the value is the lowest since it's very likely for these states to go outside the boundary and receive reward -1. Notice that $(1,4)$ has a value slightly lower than its reward 10 and $(3,4)$ is having a value slightly higher than its reward 5. This is due to the fact that although (1,4) has a very high immediate reward, the next state would be (1,0), which is a state on the boundary, a lower-value state. For (3,4), the next state must be (3,2), a much better state, not on the boundary and closer to



```
          0         1         2         3         4
0 -1.856261 -0.978066  0.042301  1.521796  3.311155
1 -1.344770 -0.435928  0.740045  2.994327  8.790195
2 -1.226271 -0.354464  0.675206  2.247773  4.421236
3 -1.421055 -0.587406  0.352294  1.902699  5.314808
4 -1.976971 -1.188604 -0.409326  0.544080  1.492610
```

Figure 9: Monte Carlo estimate for state value function of the Gridworld model

Monte Carlo with 1000 samples. For each sample, state, action and reward are being simulated until time 20000. The discount rate is $\gamma = 0.9$.

these higher-value states.

Let's try to verify that the Bellman consistency equation holds for the state value function we get numerically

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \cdot [r + \gamma \cdot v_\pi(s')] \tag{222}$$

let's check for the state $s = (2, 2)$ and its four neighboring states

$$\begin{aligned} v_\pi((2,2)) &= 0.6752 && (223)\\ RHS &= \frac{1}{4} \times 0.9 \times [v_\pi((2,3)) + v_\pi((2,1)) + v_\pi((3,2)) + v_\pi((1,2))] && (224)\\ &= \frac{0.9}{4} \times (2.25 - 0.35 + 0.35 + 0.74) && (225)\\ &= 0.67275 && (226)\end{aligned}$$

also for the state $(0, 0)$ on the boundary

$$\begin{aligned} v_\pi((0,0)) &= -1.8563 && (227)\\ RHS &= \frac{1}{4} \times [0.9 \times [v_\pi((0,1)) + v_\pi((1,0)) + v_\pi((0,0)) + v_\pi((0,0))] - 2] && (228)\\ &= -1.859 && (229)\end{aligned}$$

for the higher-value state $(1, 4)$

$$\begin{aligned} v_\pi((1,4)) &= 8.79 && (230)\\ RHS &= 10 + 0.9 \times v_\pi((1,0)) && (231)\\ &= 8.794 && (232)\end{aligned}$$

verifies the Bellman consistency equation.

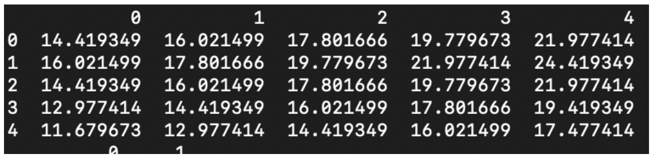
Let's see how optimal value functions are found in Gridworld and how optimal policy is constructed. Recall that BOO is a contraction mapping and the optimal value function is the fixed point, why not use fixed point iteration whose convergence is guaranteed to approximate v^*? With the value in initial iteration v_0 constructed, the fixed point iteration has

$$\forall s \in \mathscr{S}, v_{n+1}(s) = \sup_a \sum_{s',r} p(s',r|s,a) \cdot [r + \gamma \cdot v_n(s')] \tag{233}$$

we expect to see

$$\forall s \in \mathscr{S}, v_n(s) \to v^*(s)\ (n \to \infty) \tag{234}$$

refer to Fig. 10 for the experiment on Gridworld for the approximation result for v_*.

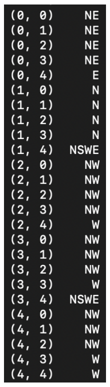


```
           0          1          2          3          4
0  14.419349  16.021499  17.801666  19.779673  21.977414
1  16.021499  17.801666  19.779673  21.977414  24.419349
2  14.419349  16.021499  17.801666  19.779673  21.977414
3  12.977414  14.419349  16.021499  17.801666  19.419349
4  11.679673  12.977414  14.419349  16.021499  17.477414
```

Figure 10: Fixed point iteration approximation for the optimal state value function v^* of the Gridworld model

The discount rate $\gamma = 0.9$. It's natural for (1,4) to have the highest optimal state value. Fixed point iteration of Bellman optimality equation is guaranteed to converge because of the contraction mapping property.

Based on the optimal state value function derived, it's easy for us to figure out the deterministic optimal policy. The results are demonstrated in Fig. 11.



```
(0, 0)    NE
(0, 1)    NE
(0, 2)    NE
(0, 3)    NE
(0, 4)     E
(1, 0)     N
(1, 1)     N
(1, 2)     N
(1, 3)     N
(1, 4)  NSWE
(2, 0)    NW
(2, 1)    NW
(2, 2)    NW
(2, 3)    NW
(2, 4)     W
(3, 0)    NW
(3, 1)    NW
(3, 2)    NW
(3, 3)     W
(3, 4)  NSWE
(4, 0)    NW
(4, 1)    NW
(4, 2)    NW
(4, 3)     W
(4, 4)     W
```

Figure 11: Optimal Policy in Gridworld model

The first column exhibits the 25 states and the second column exhibits the best actions at that state.
N stands for North, i.e. moving upward; S stands for South, i.e. moving downward; W stands for West, i.e. moving leftward; E stands for East, i.e. moving rightward.
Multiple best actions may exist at some states, e.g. at (1,4) all four actions have the same consequence, so all four actions are the best actions at state (1,4).

Connection between BOE and DPP*

Please read this section only if you are familiar with stochastic control problem. One might be able to find that BOE is closely connected with the dynamic programming principle (DPP) in the setting of stochastic control problem. View $\{S_t\}$ as state process and π as a deterministic Markovian policy (to match the setting of stochastic control problem), then π is actually the control process and r is the running reward. DPP tells us that

$$\forall s \in \mathscr{S}, v^*(s) \overset{DPP}{=} \sup_{\pi} \mathbb{E}_{A\sim\pi(\cdot|s),S'\sim P(\cdot|s,a)} \left[r(s,A) + \gamma \cdot v^*(S')\right] \tag{235}$$

$$= \sup_{\pi} \mathbb{E}_{S'\sim P(\cdot|s,\pi(s))} \left[r(s,\pi(s)) + \gamma \cdot v^*(S')\right] \tag{236}$$

$$= \sup_{\pi} \mathbb{E}_{S'\sim P(\cdot|s,\pi(s))} \left[r(s,\pi(s)) + \gamma \cdot \sup_{a'\in\mathscr{A}} q^*(S',a')\right] \tag{237}$$

if we fix the current state s and denote $\pi(s) = a$ as the action taken according to policy π, then the sup w.r.t. π is actually the sup w.r.t. action a

$$v^*(s) = \sup_{a\in\mathscr{A}} \mathbb{E}_{S'\sim P(\cdot|s,a)} \left[r(s,a) + \gamma \cdot \sup_{a'\in\mathscr{A}} q^*(S',a')\right] = \sup_{a\in\mathscr{A}} \mathbb{E}_{S'\sim P(\cdot|s,a)} \left[r(s,a) + \gamma \cdot v^*(S')\right] \tag{238}$$

gives the BOE w.r.t. optimal state value function v^*.

Remark. *In the stochastic control problem where the control is Markovian with finite time horizon and we are maximizing the reward, DPP of value function is formed as*

$$V(t,x) = \sup_{\alpha\in\mathscr{A}_t} \sup_{\tau\in\tau_{t,T}} \left\{\mathbb{E}\left[\int_t^\tau f(s, X_s^{t,x,\alpha}, \alpha_s)\, ds + V(\tau, X_\tau^{t,x,\alpha})\right]\right\} \tag{239}$$

$$= \sup_{\alpha\in\mathscr{A}_t} \inf_{\tau\in\tau_{t,T}} \left\{\mathbb{E}\left[\int_t^\tau f(s, X_s^{t,x,\alpha}, \alpha_s)\, ds + V(\tau, X_\tau^{t,x,\alpha})\right]\right\} \tag{240}$$

where $\tau_{t,T}$ is the collection of all stopping time between t and T, f is the running reward. As an analogue to BOE, view $V(t,x)$ as $v^*(s)$, view $\int_t^\tau f(s, X_s^{t,x,\alpha}, \alpha_s)\, ds$ as $r(s,a)$, $V(\tau, X_\tau^{t,x,\alpha})$ as $\gamma \cdot v^*(S')$ with exactly the same interpretation in control theory.

Since DPP gives rise to Hamilton-Jacobi-Bellman equation (HJBE), actually BOE can be understood as a discretized version of HJBE.

Finite Time Horizon MDP*

The following section uses different notations from above, it's added just for the purpose of completeness.

So far we have been focusing on infinite horizon MDP and it turns out that there are many good properties, e.g. there exists a stationary deterministic optimal policy and the Bellman optimality equations characterize the optimal value functions. However, in finite horizon MDP the settings change. Firstly, we do not introduce the discount rate any longer since the sum of finitely many real numbers must be finite. Secondly, time dependence is introduced for transition kernel and reward function. Now the transition kernel is denoted P_h at time step h, similarly the reward function is r_h so at each time step we maintain a different transition kernel and reward function.

Remark. *One might be interested in the reason why time dependence in only introduced in the finite horizon case. Intuitively, if MDP has infinite horizon, at each time step one is facing a completely same problem as before. However, when MDP has finite time horizon, that is not the case since one has to manage the cost of exploration.*

Note that the time dependence of value functions in the finite horizon setting is also one of the main reasons that time-dependent finite horizon MDP is not often used in practice.

By the reasoning above, the value functions under general policy π are naturally formed as

$$V_h^\pi(s) \overset{def}{=} \mathbb{E}_\pi\left[\sum_{t=h}^{H-1} r_t(S_t,A_t)\middle| S_h = s\right] \tag{241}$$

$$Q_h^\pi(s,a) \overset{def}{=} \mathbb{E}_\pi\left[\sum_{t=h}^{H-1} r_t(S_t,A_t)\middle| S_h = s, A_h = a\right] \tag{242}$$

where the time horizon is denoted $0,1,...,H-1$. This is exactly the same setting as that in the stochastic control problems, to organize value functions backwardly. Our objective is still to find the optimal policy π that maximizes $V_0^\pi(s)$ for given initial state s.

The Bellman optimality equation now also has time structure, provides characterization of the optimal value functions

$$V_h^*(s) \overset{def}{=} \sup_{\pi\in\Pi} V_h^\pi(s) \tag{243}$$

$$Q_h^*(s,a) \overset{def}{=} \sup_{\pi\in\Pi} Q_h^\pi(s,a) \tag{244}$$

with $V_H = 0, Q_H = 0$ defined at the tail (since the game has already ended).

Theorem 8. ***(Bellman Optimality Equation, Finite Horizon)*** *Q_h is the optimal value function iff*

$$Q_h(s,a) = r_h(s,a) + \mathbb{E}_{S'\sim P_h(\cdot|s,a)} \max_{a'\in\mathscr{A}} Q_{h+1}(S',a') \tag{245}$$

and the deterministic optimal policy at time h is given by the greedy policy w.r.t. Q_h^* that

$$\pi^*(s,h) = \pi_{Q^*}(s,h) \overset{def}{=} \arg\max_{a\in\mathscr{A}} Q_h^*(s,a) \tag{246}$$

Proof. We provide the whole proof for this theorem here mimicking the proof for the infinite horizon case. Notice that the theorem of the existence of optimal policy can be generalized here to prove that there must exist a deterministic optimal policy taking action $\pi^*(s,h)$ at time step h.

Let's first prove the compact form that

$$\forall h \in [H], s \in \mathscr{S}, V_h^*(s) = \max_{a\in\mathscr{A}} Q_h^*(s,a) \tag{247}$$

where $[H] = \{0, 1, ..., H-1\}$. By the law of total probability,

$$V_h^*(s) = V_h^{\pi^*}(s) = \sum_{a\in\mathscr{A}} \mathbb{P}_{\pi^*}(A_h = a|S_h = s) \cdot Q_h^{\pi^*}(s,a) \le \max_{a\in\mathscr{A}} Q_h^*(s,a) \tag{248}$$

on the other hand, consider π_a as a policy that always takes action a at the first time step (time step h) but follows π^* afterwards, we have

$$\forall a \in \mathscr{A}, V_h^*(s) \ge V_h^{\pi_a}(s) = Q_h^{\pi^*}(s,a) = Q^*(s,a) \tag{249}$$

so the connection between two optimal value functions is proved.

Now let's prove that Q_h^* satisfies the Bellman optimality equation

$$\begin{aligned}
Q_h^*(s,a) &= \max_\pi Q_h^\pi(s,a) &(250)\\
&= r_h(s,a) + \max_\pi \mathbb{E}_\pi\left[\sum_{t=h+1}^{H-1} r_t(S_t,A_t)\Big|S_h = s, A_h = a\right] &(251)\\
&= r_h(s,a) + \max_\pi \sum_{s'\in\mathscr{S}} P_h(s'|s,a)\cdot\mathbb{E}_\pi\left[\sum_{t=h+1}^{H-1} r_t(S_t,A_t)\Big|S_h = s, A_h = a, S_{h+1} = s'\right] &(252)\\
&= r_h(s,a) + \max_\pi \sum_{s'\in\mathscr{S}} P_h(s'|s,a)\cdot V_{h+1}^\pi(s') &(253)\\
&= r_h(s,a) + \max_\pi \mathbb{E}_{S'\sim P_h(\cdot|s,a)} V_{h+1}^\pi(S') &(254)\\
&= r_h(s,a) + \mathbb{E}_{S'\sim P_h(\cdot|s,a)} V_{h+1}^*(S') &(255)\\
&= r_h(s,a) + \mathbb{E}_{S'\sim P_h(\cdot|s,a)} \max_{a'\in\mathscr{A}} Q_{h+1}^*(S',a') &(256)
\end{aligned}$$

gets proved.

Conversely, if the Bellman optimality equation holds for Q_h, consider π_Q as the greedy policy w.r.t. Q_h defined as

$$\pi_Q(s,h) \overset{def}{=} \underset{a\in\mathscr{A}}{\arg\max}\, Q_h(s,a) \tag{257}$$

so it's easy to verify that

$$\begin{aligned} Q_h^{\pi_Q}(s,a) &= r_h(s,a) + \mathbb{E}_{S'\sim P_h(\cdot|s,a)} V_{h+1}^{\pi_Q}(S') && (258)\\ &= r_h(s,a) + \mathbb{E}_{S'\sim P_h(\cdot|s,a)} \sum_{a'\in\mathscr{A}} \pi_Q(a'|S',h+1)\cdot Q_{h+1}^{\pi_Q}(S',a') && (259)\\ &= r_h(s,a) + \mathbb{E}_{S'\sim P_h(\cdot|s,a)} Q_{h+1}^{\pi_Q}(S',\pi_Q(S',h+1)) && (260)\\ &= r_h(s,a) + \mathbb{E}_{S'\sim P_h(\cdot|s,a)} \max_{a'\in\mathscr{A}} Q_{h+1}^{\pi_Q}(S',a') = Q_h(s,a) && (261) \end{aligned}$$

so $Q = Q^{\pi_Q}$ is actually just the value function w.r.t. the policy π_Q. Now in order to prove that π_Q is the optimal policy, it suffices to prove that for any non-stationary deterministic policy π', $Q_h^{\pi_Q} - Q_h^{\pi'} \geq 0$.

From the Bellman consistency equation for finite horizon MDP proved above that

$$\begin{cases} V_h^\pi(s) = \sum_{a\in\mathscr{A}} \mathbb{P}_\pi(A_h = a|S_h = s)\cdot Q_h^\pi(s,a) \\ Q_h^\pi(s,a) = r_h(s,a) + \mathbb{E}_{S'\sim P_h(\cdot|s,a)} V_{h+1}^\pi(S') \end{cases} \tag{262}$$

we rebuild the representation of Q_h^π for fixed horizon $h \in [H]$ as a vector in $\mathbb{R}^{|\mathscr{S}|\cdot|\mathscr{A}|}$ that

$$\begin{cases} Q_h^\pi = r_h + P_h V_{h+1}^\pi \\ Q_h^\pi = r_h + P_h^\pi Q_{h+1}^\pi \end{cases} \tag{263}$$

where $(P_h^\pi)_{(s,a),(s',a')} \overset{def}{=} (P_h)_{(s,a),s'}\pi(a'|s',h)$. This leads to the fact that

$$\begin{aligned} Q_h^{\pi_Q} - Q_h^{\pi'} &= r_h + P_h^{\pi_Q} Q_{h+1}^{\pi_Q} - r_h - P_h^{\pi'} Q_{h+1}^{\pi'} && (264)\\ &= P_h^{\pi_Q} Q_{h+1}^{\pi_Q} - P_h^{\pi'} Q_{h+1}^{\pi'} && (265)\\ &= P_h^{\pi_Q}(Q_{h+1}^{\pi_Q} - Q_{h+1}^{\pi'}) + (P_h^{\pi_Q} - P_h^{\pi'}) Q_{h+1}^{\pi'} && (266) \end{aligned}$$

with the first term on RHS to have positive components since $Q_{h+1}^{\pi_Q} - Q_{h+1}^{\pi'} \geq 0$ by the definition of π_Q and each row of $P_h^{\pi_Q}$ is a probability distribution. For the second term on the RHS, it's also positive since

$$\begin{aligned} [(P_h^{\pi_Q} - P_h^{\pi'})Q_{h+1}^{\pi'}]_{(s,a)} &= \sum_{s'\in\mathscr{S}, a'\in\mathscr{A}} ([P_h^{\pi_Q}]_{(s,a),(s',a')} - [P_h^{\pi'}]_{(s,a),(s',a')})\cdot Q_{h+1}^{\pi'}(s',a') && (267)\\ &= \sum_{s'\in\mathscr{S}} (P_h)_{(s,a),s'}\cdot Q_{h+1}^{\pi'}(s',\pi_Q(s',h)) - \sum_{s'\in\mathscr{S}} (P_h)_{(s,a),s'}\cdot Q_{h+1}^{\pi'}(s',\pi'(s',h)) && (268)\\ &= \mathbb{E}_{S'\sim P_h(\cdot|s,a)}[Q_{h+1}^{\pi'}(S',\pi_Q(S',h)) - Q_{h+1}^{\pi'}(S',\pi'(S',h))] \geq 0 && (269) \end{aligned}$$

by noticing that π_Q, π' are both deterministic policies and that by the definition of π_Q again, $Q_{h+1}^{\pi'}(S', \pi_Q(S', h)) \geq Q_{h+1}^{\pi'}(S', \pi'(S', h))$. So we have proved that π_Q is the optimal policy and Q must be the optimal value function. $\square$

Remark. *The proof above is very similar to the case for infinite horizon MDP and the only difference lies in the different formulations (time dependency) of Bellman consistency equation and Bellman optimality equation stated in the proof.*

Similarly, if we restrict ourselves to deterministic policy π, RL actually becomes a discrete-time finite horizon stochastic control problem where the policy is just the control process and it's time dependent. By applying the DPP again, we see that

$$V_h^*(s) \overset{DPP}{=} \sup_\pi \mathbb{E}_{\pi, A\sim\pi(\cdot|s,h), S'\sim P_h(\cdot|s,A)}[r(s,A) + V_{h+1}^*(S')] \tag{270}$$

$$= \sup_{a\in\mathscr{A}} \mathbb{E}_{S'\sim P_h(\cdot|s,a)}[r(s,a) + V_{h+1}^*(S')] \tag{271}$$

since π is deterministic and we denote $a = \pi(s,h)$, the sup taken w.r.t. all deterministic policies is just the sup taken w.r.t. all possible actions. Therefore, we derive the Bellman optimality equation for state value function V^*. To see the consistency with the Bellman optimality equation for Q^* above, take maximum w.r.t. action a to see

$$V_h^*(s) = \max_{a\in\mathscr{A}} Q_h^*(s,a) = \max_{a\in\mathscr{A}} \left\{ r_h(s,a) + \mathbb{E}_{S'\sim P_h(\cdot|s,a)} V_{h+1}^*(S') \right\} \tag{272}$$

so even if in the time-dependent and finite horizon case, it's still consistent with DPP.

Dynamic Programming (DP) Methods

The DP methods are the most direct algorithms for RL problem. The DP methods are **model-based**, i.e. it depends on the knowledge of the MDP transition kernel p. The state space is required to be not that large such that sweeping through all the states would be practically feasible.

Policy Evaluation

The policy evaluation task refers to the estimation of value functions v_π, q_π given policy π. Let's take the evaluation of v_π as an example here, the evaluation of q_π can be done similarly. Recall that v_π is characterized by Bellman consistency equation

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \cdot [r + \gamma \cdot v_\pi(s')] \tag{273}$$

it's a linear system w.r.t. $v_\pi(s)$ and it has a unique solution. Shall we organize this equation in the matrix form to solve for v_π? That's definitely an efficient way to derive v_π but we can also use iterative methods to solve this linear equation.

Define the operator $T : \mathbb{R}^{|\mathscr{S}|} \to \mathbb{R}^{|\mathscr{S}|}$

$$Tv(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \cdot [r + \gamma \cdot v(s')] \tag{274}$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma \cdot v(S_{t+1})|S_t = s] \tag{275}$$

the Bellman consistency equation is

$$v_\pi(s) = Tv_\pi(s) \tag{276}$$

so v_π is the fixed point of operator T. Similar to the Bellman optimality operator, T is a contraction mapping

$$||Tv - Tv'||_\infty \leq \gamma ||v - v'||_\infty, 0 \leq \gamma < 1 \tag{277}$$

fixed point iteration always converges. As a result, the iterative policy evaluation is given by

$$v_{n+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \cdot [r + \gamma \cdot v_n(s')] \tag{278}$$

and we expect to see

$$\forall s, v_n(s) \to v_\pi(s) \ (n \to \infty) \tag{279}$$

For action value function q_π similar arguments hold, write out the Bellman consistency equation

$$q_\pi(s,a) = \sum_{r,s'} p(s',r|s,a) \cdot \left[r + \gamma \sum_{a'} \pi(a'|s') \cdot q_\pi(s',a') \right] \tag{280}$$

and the iterative policy evaluation

$$q_{n+1}(s,a) = \sum_{r,s'} p(s',r|s,a) \cdot \left[r + \gamma \sum_{a'} \pi(a'|s') \cdot q_n(s',a') \right] \tag{281}$$

define the operator T such that

$$Tq(s,a) = \sum_{r,s'} p(s',r|s,a) \cdot \left[r + \gamma \sum_{a'} \pi(a'|s') \cdot q(s',a') \right] \tag{282}$$

$$= \mathbb{E}_\pi \left[R_{t+1} + \gamma \sum_{a'} \pi(a'|S_{t+1}) \cdot q(S_{t+1},a') \middle| S_t = s, A_t = a \right] \tag{283}$$

it's still a contraction mapping

$$\forall s,a, |Tq(s,a) - Tq'(s,a)| = \gamma \left| \mathbb{E}_\pi \left[\sum_{a'} \pi(a'|S_{t+1}) \cdot (q(S_{t+1},a') - q'(S_{t+1},a')) \middle| S_t = s, A_t = a \right] \right| \tag{284}$$

$$\leq \gamma ||q - q'||_\infty \cdot \mathbb{E}_\pi \left[\sum_{a'} \pi(a'|S_{t+1}) \middle| S_t = s, A_t = a \right] \tag{285}$$

$$= \gamma ||q - q'||_\infty \tag{286}$$

so we expect to see

$$\forall s,a, q_n(s,a) \to q_\pi(s,a) \ (n \to \infty) \tag{287}$$

Remark. *When setting initial values for iterative value iteration, if the task is episodic, the terminal state must be assigned zero value to indicate that the game ends once it hits the terminal state.*

Policy Iteration

Policy evaluation is discussed above for the purpose of conducting policy iteration. From the policy improvement theorem, for any policy π, $\pi'(s) = \arg\max_a q_\pi(s,a)$ is always a deterministic policy guaranteed to be no worse than π. Such π' is typically called the **greedy policy w.r.t.** π and this has provided a way to iteratively improve the policy until the policy is optimal. We only need to start from any policy π, do policy evaluation to get v_π or q_π and then calculate the greedy policy w.r.t. π as an improvement.

The policy iteration for v_π is provided in Alg. 2 and the policy iteration for q_π is provided in Alg. 3.

Algorithm 2 v_π-Policy Iteration

```
Input: Initial policy $\pi_0$
1: repeat
2:     Policy Evaluation
3:     repeat
4:         $v_{n+1}(s) = \sum_a \pi_k(a|s) \sum_{s',r} p(s',r|s,a) \cdot [r + \gamma \cdot v_n(s')]$
5:     until $||v_{n+1} - v_n|| \leq \varepsilon$
6:     $v_{\pi_k} = v_{n+1}$
7:     Policy Improvement
8:     $\pi_{k+1}(s) = \arg\max_a \sum_{s',r} p(s',r|s,a) \cdot [r + \gamma \cdot v_{\pi_k}(s')]$
9: until $||v_{\pi_k} - v_{\pi_{k+1}}|| < \delta$
Output: Optimal policy $\pi^* = \pi_{k+1}$
```

- Repeat until $\pi_k \to \pi_*$ happens. Note that setting the stopping criteria as $\pi_k = \pi_{k+1}$ is not sufficient since it's possible for the optimal policy to jump back and forth between two different optimal policies. Make use of the uniqueness of v_* and a good stopping criteria would be $||v_{\pi_k} - v_{\pi_{k+1}}||_\infty < \varepsilon$.

Algorithm 3 q_π-Policy Iteration

```
Input: Initial policy $\pi_0$
1: repeat
2:     Policy Evaluation
3:     repeat
4:         $q_{n+1}(s,a) = \sum_{r,s'} p(s',r|s,a) \cdot [r + \gamma \sum_{a'} \pi(a'|s') \cdot q_n(s',a')]$
5:     until $||q_{n+1} - q_n|| \leq \varepsilon$
6:     $q_{\pi_k} = q_{n+1}$
7:     Policy Improvement
8:     $\pi_{k+1}(s) = \arg\max_a q_{\pi_k}(s,a)$
9: until $||q_{\pi_k} - q_{\pi_{k+1}}|| < \delta$
Output: Optimal policy $\pi^* = \pi_{k+1}$
```

Remark. *The stopping criterion for the whole algorithm requires the value function v_{π_k} to be close enough to $v_{\pi_{k+1}}$ instead of π_k being close enough to π_{k+1}. This is due to the fact that optimal policy is not unique so it's possible to see the jump back-and-forth between two different optimal policy. However, different optimal policy corresponds to* *the same unique optimal value function in RL problem. That's why the stopping criterion is based on value function instead of the policy.*

Some theoretical analysis can be conducted to show the convergence and the convergence rate of this algorithm. However, those are standard numerical analysis and are not of practical interest so we do not mention it here. It's worth noting that the convergence of policy iteration is guaranteed by the policy improvement theorem. Recall that if π_k is not yet optimal, the greedy policy w.r.t. π_k must be a strict improvement.

Value Iteration

Actually, we have already used value iteration techniques in the Gridworld example. It's just a fixed point iteration of the non-linear Bellman optimality equations to directly get v^* or q^* and then use the optimal value function to construct the optimal policy. For the purpose of completeness, we present the value iteration w.r.t. v^* here in Alg. 4.

Algorithm 4 v^*-Value Iteration

```
Input: Initial estimate of $v^*$ denoted $v_0$
1: repeat
2:     $v_{n+1}(s) = \sup_a \sum_{s',r} p(s',r|s,a) \cdot [r + \gamma \cdot v_n(s')]$
3: until $||v_{n+1} - v_n|| < \varepsilon$
4: $v^* = v_{n+1}$
5: $\pi^*(s) = \arg\max_a \sum_{s',r} p(s',r|s,a) \cdot [r + \gamma \cdot v^*(s')]$
Output: Optimal policy $\pi^*$
```

For q^*-value iteration, replace fixed point iteration with

$$q_{n+1}(s,a) = \sum_{s',r} p(s',r|s,a) \cdot \left[r + \gamma \cdot \sup_{a'} q_n(s',a')\right] \tag{288}$$

and the construction of optimal policy as

$$\pi^*(s) = \arg\max_a q^*(s,a) \tag{289}$$

the convergence of value iteration is guaranteed by BOO being a contraction mapping.

Remark. *The value iteration seems to overperform the policy iteration on time complexity but we would like to mention here the idea and possible generalization of policy iteration. Policy iteration can be understood as the cooperation between the policy and the value function in the sense that the policy acts (decides which action to take) and the value function criticizes (tells the policy how to improve itself). As time goes by, the policy and the value function are both converging to the optimal. This is exactly the first motivation of actor-critic policy gradient algorithm we will introduce later.*

DP methods are efficient for small-scale simple RL problems. However, it's a **model-based** algorithm depending on the knowledge of p. Unfortunately, this drawback is fatal for solving RL problems in real life since in most cases we don't have any information about the transition kernel. When we are playing a new game, we never know a priori how the game is designed and what we are expected to do in the game. It's from our **experience** that we learn partially about the model and eliminate the bad actions. The emphasis on experience without knowing the underlying model gives natural rise to Monte Carlo methods introduced in a later context.

Example: Gambler's Problem

A gambler gambles on a sequence of coin flips. The game ends if the gambler has no money (fails) or if the gambler has got 100 (wins). For each flip, the gambler decide an integer as the stake. If the coin comes up heads, he wins as much as his stake. If the coin comes up tails, he loses all stakes. Note that the sum of the stake and the amount of money he owns now has to be no larger than 100.

Let's first set up the problem as an MDP. The reward is given based on if the gambler wins the game, i.e. reward is 0 for all transitions and 1 iff the gambler wins the game at last. The state space is $\mathscr{S} = \{0, 1, ..., 99, 100\}$ indicating the amount of money the gambler currently owns before putting down the stake. Note that $0, 100$ are two special terminal states, 0 always has state value 0 and 100 always has state value 1, to be consistent with the reward. The action space depends on the current state $\mathscr{A}(s) = \{0, 1, ..., \min\{s, 100 - s\}\}$.

Under the undiscounted setting where $\gamma = 1$, state value function is

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\mathbb{I}_{\text{win}}|S_t = s] = \mathbb{P}_\pi(\text{win}|S_t = s) \tag{290}$$

has the interpretation as the probability of winning given policy π and the current state s. We set up a changeable parameter for this game as p_h, the probability of getting a head in a single coin flip. It's interesting for us to see how the optimal policy of the gambler changes as p_h changes.

To use DP methods, let's first find out the transition kernel p. If heads appears, $s' = s + a$ and $r = 1$; if tails appears, $s' = s - a$ and $r = 0$. The end of the game comes in two scenarios: when we throw a head and $s' = 100$ or when we throw a tail and $s' = 0$. As a result,

$$p(s + a, 0|s, a) = p_h \text{ (if } s + a < 100) \tag{291}$$

$$p(s + a, 1|s, a) = p_h \text{ (if } s + a = 100) \tag{292}$$

$$p(s - a, 0|s, a) = 1 - p_h \tag{293}$$

$$p(s, 0|s, 0) = 1 \tag{294}$$

Through value iteration method, we get the optimal value function in Fig. 12, 13, 14 for different values of p_h and all optimal actions are printed for each state.

Remark. *The numerical experiment produces very interesting results. Those results might seem counter-intuitive at the first glance but can be well-explained from a probabilistic perspective.*

When $p_h > \frac{1}{2}$, we have positive expected payoff from the gamble so SLLN is working towards us. Recall that SLLN needs a large enough number of attempts to work, that's why the optimal policy is to force the game to last longer. The evidence is that action 1 is always the best action for all states.

When $p_h = \frac{1}{2}$, it's a completely fair game and all actions are indifferent.

When $p_h > \frac{1}{2}$, SLLN is working against us so the optimal policy is to shorten the lasting time of the game. That's why when we are at state 50 we will bet everything we have to end the game immediately. Surprisingly, when we are at state 51, betting 49 and betting 1 are as good as each other (think about the explanation). This provides us with the optimal policy looking like a binary tree.

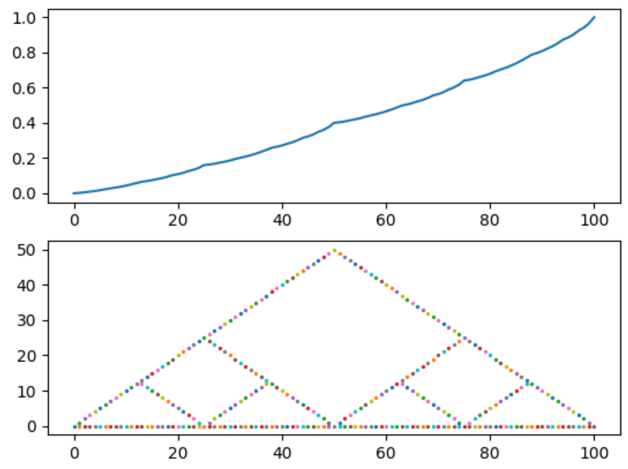


Figure 12: Optimal state value function and optimal policy for gambler's problem with $p_h = 0.4$

The upper subplot is showing $v^*(s)$. The lower subplot is showing π^*.

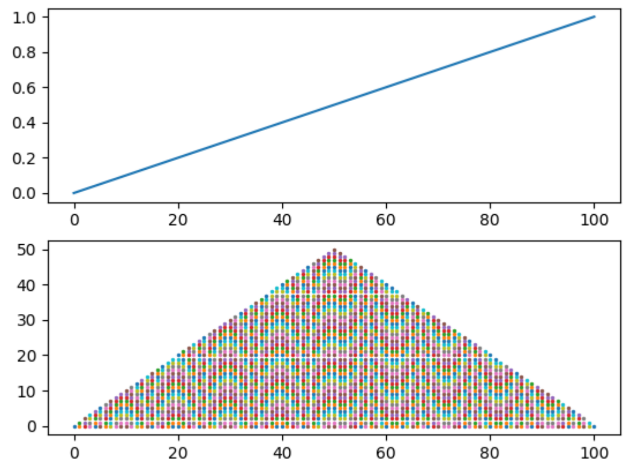


Figure 13: Optimal state value function and optimal policy for gambler's problem with $p_h = 0.5$

The upper subplot is showing $v^*(s)$. The lower subplot is showing π^*.

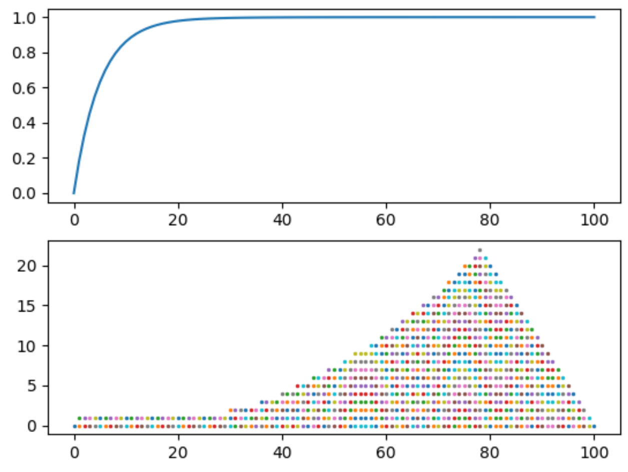


Figure 14: Optimal state value function and optimal policy for gambler's problem with $p_h = 0.55$

The upper subplot is showing $v^*(s)$. The lower subplot is showing π^*.

Analysis for Value Iteration*

The following section has different notations and is provided here for the purpose of completeness. Denote the infinite horizon MDP as $M = (\mathscr{S}, \mathscr{A}, P, r, \gamma)$ (without specification we consider infinite horizon case) and $L(P, r, \gamma)$ denotes the total number of bits required to specify M. Of course a feasible method should have polynomial time to figure out the optimal policy and the time complexity shall be a function of $|\mathscr{S}|, |\mathscr{A}|, L(P, r, \gamma), \gamma$. A method is called strongly polynomial if the time complexity is a polynomial in everything with no dependence on $L(P, r, \gamma)$.

The idea of value iteration is to find out Q^* and to construct the optimal policy as the greedy policy w.r.t. Q^*. In order to figure out Q^*, notice that it is the fixed point of the Bellman optimality operator $\mathscr{T}$ so it's natural to apply fixed point iteration. As what we have shown in the previous context, $\mathscr{T}$ is actually a contraction mapping under infinity norm.

Lemma 7. ***($\mathscr{T}$ as Contraction Mapping)***

$$\forall Q, Q' \in \mathbb{R}^{|\mathscr{S}|\cdot|\mathscr{A}|}, ||\mathscr{T}Q - \mathscr{T}Q'||_\infty \leq \gamma ||Q - Q'||_\infty \tag{295}$$

Combine with the fact that $\gamma < 1$, we know that the fixed point uniquely exists and the fixed point iteration converges to Q^*. Now we can apply the same error analysis techniques as those in the fixed point iteration to get the following error bounds.

Remark. *Notice that we have actually proved that although the optimal policy is often not unique, all different optimal policies share the* ***unique optimal value function*** *Q^*, V^* from the uniqueness of the fixed point.*

Lemma 8. ***(Q-Error Amplification)***

$$\forall Q \in \mathbb{R}^{|\mathscr{S}|\cdot|\mathscr{A}|}, V^{\pi_Q} \geq V^* - \frac{2||Q - Q^*||_\infty}{1-\gamma}\vec{1} \tag{296}$$

Proof. It follows from

$$V^*(s) - V^{\pi_Q}(s) = Q^*(s, \pi^*(s)) - Q^*(s, \pi_Q(s)) + Q^*(s, \pi_Q(s)) - Q^{\pi_Q}(s, a) \tag{297}$$

then apply Bellman consistency equation to write the second difference as an expectation in terms of V and derive the bound that

$$\forall s \in \mathscr{S}, V^*(s) - V^{\pi_Q}(s) \leq 2||Q - Q^*||_\infty + \gamma ||V^* - V^{\pi_Q}||_\infty \tag{298}$$

□

Remark. *This lemma tells us that if we adopt the greedy policy w.r.t. Q, when Q is very close to Q^*, V^{π_Q} is also very close to V^*. So the sub-optimality gap in V of the greedy policy can be bounded by the sub-optimality gap in Q. This will help us figure out the shrinking rate of the sub-optimality gap in V if the greedy policy is taken.*

Theorem 9. ***(Q-Value Iteration Convergence)*** *Set up the fixed point iteration $Q^{(0)} = 0, Q^{(k+1)} = \mathscr{T} Q^{(k)}$ and set $\pi^{(k)} = \pi_{Q^{(k)}}$ as the greedy policy in the k-th iteration, then for error tolerance $\varepsilon > 0$ such that $V^{\pi^{(k)}} \geq V^* - \varepsilon \vec{1}$ we need $k \geq \frac{\log \frac{2}{(1-\gamma)^2 \varepsilon}}{1-\gamma}$ number of iterations to guarantee such error tolerance.*

Proof. By the lemma above,

$$\frac{2||Q^{(k)} - Q^*||_\infty}{1-\gamma} \leq \varepsilon \tag{299}$$

implies $V^{\pi^{(k)}} \geq V^* - \varepsilon \vec{1}$ and by contraction mapping, $||Q^{(k)} - Q^*||_\infty \leq \gamma^k ||Q^{(0)} - Q^*||_\infty$ so we just need to ensure that

$$\frac{2\gamma^k}{1-\gamma} ||Q^{(0)} - Q^*||_\infty \leq \varepsilon \tag{300}$$

notice that the reward is always between 0 to 1 so $||Q^*||_\infty \leq \frac{1}{1-\gamma}$ and such k suffices to ensure $\frac{2\gamma^k}{(1-\gamma)^2} \leq \varepsilon$. $\square$

Remark. *To specify the MDP, we need $L(P, r, \gamma)$ number of bits so the $V^{\pi^{(k)}}$ from value iteration has no difference from the true V^* under the machine accuracy level if the difference is no more than $\varepsilon = 2^{-L(P,r,\gamma)}$. That's why the time complexity of value iteration is $O\left(|\mathscr{S}|^2 |\mathscr{A}| \frac{L(P,r,\gamma) \log \frac{1}{1-\gamma}}{1-\gamma}\right)$ and it's not strongly polynomial. Notice that we would operate the value iteration for $O\left(\frac{L(P,r,\gamma) \log \frac{1}{1-\gamma}}{1-\gamma}\right)$ iterations but in each iteration we have to apply the Bellman optimality operator that results in $O(|\mathscr{S}|^2 |\mathscr{A}|)$ calculations since for each state s and action a, the calculation of the expectation $\mathbb{E}_{S' \sim P(\cdot|s,a)} \max_{a' \in \mathscr{A}} Q(S', a')$ takes $O(|\mathscr{S}|)$ time.*

Of course, value iteration can be generalized to apply for finite horizon MDP, and one just have to change the update to the time dependent Bellman optimality equation. It's still a contraction mapping and the greedy policy shall be formed at each time step.

Analysis for Policy Iteration*

Policy iteration consists of two parts, policy evaluation and policy improvement. The idea is to start from some policy π_k, evaluate the value function Q^{π_k} and then form $\pi_{k+1} = \pi_{Q^{\pi_k}}$ as the greedy policy w.r.t. Q^{π_k} to be a better one. It's natural to prove that the policy is improving as proceeding in the policy iteration.

Lemma 9. ***(Policy Improvement)***

$$Q^{\pi_{k+1}} \geq \mathscr{T}Q^{\pi_k} \geq Q^{\pi_k}, ||Q^{\pi_{k+1}} - Q^*||_\infty \leq \gamma ||Q^{\pi_k} - Q^*||_\infty \tag{301}$$

Proof. $\mathscr{T}Q^\pi \geq Q^\pi$ directly follows from the Bellman consistency equation. Since π_{k+1} is greedy w.r.t. Q^{π_k}, by Bellman consistency equation again $Q^{\pi_k} \leq r + \gamma P^{\pi_{k+1}} Q^{\pi_k}$ and iterative application combined with the power series expansion of $(I - \gamma P^{\pi_{k+1}})^{-1}$ proves $Q^{\pi_{k+1}} \geq Q^{\pi_k}$. Apply such conclusion, we can prove $Q^{\pi_{k+1}} \geq \mathscr{T}Q^{\pi_k}$.

The reason we want to build the relationship between not only $Q^{\pi_k}, Q^{\pi_{k+1}}$ but also $\mathscr{T}Q^{\pi_k}$ is that it makes our life easier when building up the error shrinking rate since Q^* is the fixed point

$$||Q^{\pi_{k+1}} - Q^*||_\infty \leq ||\mathscr{T}Q^{\pi_k} - \mathscr{T}Q^*||_\infty \leq \gamma ||Q^{\pi_k} - Q^*||_\infty \tag{302}$$

by the contraction mapping. □

Theorem 10. ***(Policy Iteration Convergence)*** *Let π_0 be any initial policy to start the policy iteration stated above. For error tolerance $\varepsilon > 0$ such that $Q^{\pi^{(k)}} \geq Q^* - \varepsilon \vec{1}$ we need $k \geq \frac{\log \frac{1}{(1-\gamma)\varepsilon}}{1-\gamma}$ number of iterations to guarantee such error tolerance.*

Proof. By the lemma above,

$$||Q^{\pi_k} - Q^*||_\infty \leq \gamma^k ||Q^{\pi_0} - Q^*||_\infty \leq \frac{\gamma^k}{1-\gamma} \tag{303}$$

so we just need to ensure that

$$\frac{\gamma^k}{1-\gamma} \leq \varepsilon \tag{304}$$

and $k \geq \frac{\log \frac{1}{(1-\gamma)\varepsilon}}{1-\gamma}$ suffices. □

Remark. *A similar argument to that made for value iteration gives the following time complexity for policy iteration $O\left((|\mathscr{S}|^3 + |\mathscr{S}|^2|\mathscr{A}|)\frac{L(P,r,\gamma)\log\frac{1}{1-\gamma}}{1-\gamma}\right)$. Notice that in each iteration policy evaluation requires the calculation $Q^{\pi_k} = (I - \gamma P^{\pi_k})^{-1} r$ so it takes time $O(|\mathscr{S}|^3)$ to invert the matrix and time $O(|\mathscr{S}|^2|\mathscr{A}|)$ to compute the matrix product since π_k is deterministic (sparse structure of P^{π_k}).*

The difference here is that the number of policies will not exceed $|\mathscr{A}|^{|\mathscr{S}|}$ so we can get rid of the $L(P,r,\gamma)$ in the time complexity. For a fixed discount factor γ, policy iteration is strongly polynomial with time complexity $O\left((|\mathscr{S}|^3 + |\mathscr{S}|^2|\mathscr{A}|) \cdot \min\left\{\frac{|\mathscr{A}|^{|\mathscr{S}|}}{|\mathscr{S}|}, \frac{|\mathscr{S}|^2|\mathscr{A}|\log\frac{|\mathscr{S}|^2}{1-\gamma}}{1-\gamma}\right\}\right)$. Here $\frac{|\mathscr{S}|^2|\mathscr{A}|\log\frac{|\mathscr{S}|^2}{1-\gamma}}{1-\gamma}$ is the maximum number of iterations *needed to find the optimal policy under fixed* γ *and* $\frac{|\mathscr{A}|^{|\mathscr{S}|}}{|\mathscr{S}|}$ *is the maximum number of policies we will need to check, taking a minimum gives the number of iterations needed in the worst scenario.*

Monte Carlo (MC) Methods

MC is used to estimate expectation from empirical experience. Notice that v_π, q_π, v^*, q^* all have the structure of conditional expectation, this fits perfectly with MC and the model-free requirement in RL.

First-visit MC

The motivation of MC method comes from the observation in policy iteration method that the model-based property is caused by policy evaluation but not policy improvement. As a result, if we can estimate $v_\pi(s)$ only from experience (MC), then policy iteration can be extended to a model-free algorithm. Since we have already had experience dealing with estimating value function through Monte Carlo, the first-visit MC algorithm is easy for us to understand. To clarify, the visit of an action or a state just means the appearance of an action or a state in the simulation. A first visit of state-action pair (s, a) just means the first time a simulation generates state-action pair $S_t = s, A_t = a$. First-visit MC gets its name because only the return at the first visit to a state s is used to build estimation. This guarantees that all MC samples are *i.i.d.*. Refer to Alg. 5 for the details.

Algorithm 5 First-visit Monte Carlo

```
Input: Initial policy $\pi_0$
 1: repeat
 2:     Policy Evaluation
 3:     repeat
 4:         Generate sample following policy $\pi$: $S_0, A_0, R_1, S_1, ...$
 5:         Find out first-visit time $t_f(s,a) = \arg\min_t \{(S_t, A_t) = (s,a)\}$ for each state-action pair $(s,a)$
 6:         Record the return of state-action pair $(s,a)$ at time $t_f(s,a)$
 7:     until Enough samples are generated
 8:     For each state-action pair $(s,a)$, take average of the returns recorded as an estimate for $q_{\pi_k}(s,a)$
 9:     Policy Improvement
10:     $\pi_{k+1}(s) = \arg\max_a q_{\pi_k}(s,a)$
11: until $||q_{\pi_{k+1}} - q_{\pi_k}|| < \varepsilon$
12: $\pi^* = \pi_{k+1}$
Output: Optimal policy $\pi^*$
```

It's quite obvious that first-visit MC is wasting quite a lot of samples if the same state-action pair (s, a) repeatedly appears in the simulation. The every-visit MC method solves this problem by collecting the return of every visit to (s, a), not only the first visit. However, in every-visit MC the samples are no longer *i.i.d.*. Luckily, the convergence of Monte Carlo still holds and the error still shrinks at rate $n^{-\frac{1}{2}}$, the same as normal Monte Carlo.

Remark. *First-visit Monte Carlo is the first* ***model-free*** *algorithm we have seen so far, in the sense that it does not require any knowledge on transition kernel p. Be careful that this does not mean there is no underlying p, it exists (still in the framework of MDP) but is just unknown. Although it's model-free, this algorithm requires a simulator, i.e. free access to the interaction with the environment. Practically, this is the case when we are playing chess since all the rules are known while this is not the case when we are building an online recommendation system.*

Despite the improvement to every-visit Monte Carlo, this algorithm is still ***sample inefficient*** *since each policy evaluation requires a lot of simulations and multiple times of policy evaluation is required for the policy to converge to the optimal policy. Another problem is the* ***lack of exploration****. Every-visit MC requires the experience of visiting certain state-action pair. If some state-action pair has a small probability of appearing, every-visit MC is likely to miss it, which results in* $q_\pi(s, a)$ *being a bad estimate for the action value.*

Last but not least, we want to remind the reader that MC is carried out on q_π *instead of* v_π *since the policy improvement step requires the knowledge of the model if carried out on* v_π*, but this is not the case for* q_π*!*

MC Control for ε-Soft Policy

To improve the first-visit MC, we noticed the two main problems: lack of exploration and sample inefficiency. Let's first solve the second problem by requiring the method not to do Monte Carlo in each policy evaluation step to estimate q_π.

The idea comes from general policy iteration (GPI). Now that policy evaluation costs too much time, why don't we improve the policy before the whole policy evaluation procedure is completed? Since policy and value function are actor and critic, why do we always have to wait for the critic to be strong enough and then improve the actor? In other words, GPI is just saying that actor and critic and grow stronger simultaneously in time. This sound reasonable but is there any evidence to argue that GPI makes sense? Actually there is, it's just the policy iteration and value iteration that we have seen in DP methods. Recall the value iteration for v^*

$$v_{n+1}(s) = \max_a \sum_{s',r} p(s',r|s,a) \cdot [r + \gamma \cdot v_n(s')] \tag{305}$$

and the policy evaluation for v_π

$$v_{n+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \cdot [r + \gamma \cdot v_n(s')] \tag{306}$$

if we replace the policy π in the policy evaluation with a greedy deterministic policy

$$\pi'(s) = \arg\max_a \sum_{s',r} p(s',r|s,a) \cdot [r + \gamma \cdot v_n(s')] \tag{307}$$

it directly becomes the value iteration! In other words, the value iteration can be viewed as the GPI version of policy iteration because it evaluates and improves the policy at the same time. The same strategy can be applied in MC methods that for a single MDP trajectory from simulation, one time of policy improvement can be done instead of waiting for all the samples to be generated.

Let's shift gears to deal with the issue of exploration. It's clear that policy is defined as a randomized decision rule in RL in order to encourage exploration, but the greedy policy is deterministic which goes against this motivation. As a result, an analogue of ε-greedy strategy is constructed called the **ε-soft policy**. This policy distributed probability mass $\frac{\varepsilon}{|\mathscr{A}(S_t)|}$ to all other non-greedy actions and probability mass $1 - \varepsilon + \frac{\varepsilon}{|\mathscr{A}(S_t)|}$ to the greedy action $A^* = \arg\max_a Q(S_t, a)$ where Q is an estimate of q_π.

Combining those two ideas, we get the MC control for ε-soft policy as Alg. 6

Remark. *The policy improvement step makes use of policy improvement theorem and notice that this theorem is still true if the improved policy is an ε-soft policy. Let π' be ε-soft greedy policy w.r.t. π, then*

$$q_\pi(s, \pi'(s)) = \frac{\varepsilon}{|\mathscr{A}(s)|} \sum_a q_\pi(s,a) + (1-\varepsilon) \max_a q_\pi(s,a) \tag{308}$$

$$\geq \frac{\varepsilon}{|\mathscr{A}(s)|} \sum_a q_\pi(s,a) + \sum_a \pi(a|s) \cdot q_\pi(s,a) - \frac{\varepsilon}{|\mathscr{A}(s)|} \sum_a q_\pi(s,a) = v_\pi(s) \tag{309}$$

Algorithm 6 First-visit Monte Carlo Control with ε-soft Policy

```
Input: Initial policy $\pi_0$
1: $\pi = \pi_0$
2: repeat
3:     Generate sample following policy $\pi$: $S_0, A_0, R_1, S_1, ..., R_T$
4:     for $t = T-1, T-2, ..., 0$ do
5:         Record return $G_t$ if it is the first visit to state-action pair $(S_t, A_t)$
6:         Update $Q(S_t, A_t)$, the estimate of value function
7:         Greedy action $A^* = \arg\max_a Q(S_t, a)$
8:         Update $\varepsilon$-soft policy $\pi(a|S_t)$, taking action $A^*$ with probability $1-\varepsilon+\frac{\varepsilon}{|\mathscr{A}(S_t)|}$, any other action with
           probability $\frac{\varepsilon}{|\mathscr{A}(S_t)|}$
9:     end for
10: until Enough iteration is done
11: $\pi^* = \pi$
Output: Optimal policy $\pi^*$
```

so $\pi' \geq \pi$. It's still an improvement of the policy.

Despite the great improvement we have made on the vanilla first-visit MC, this algorithm only produces optimal policy π^* to be optimal among all ε-soft policy (since in the procedure of the algorithm π^* is always ε-soft) while the space of all possible policy is much larger than the space of all ε-soft policy. Further improvement needs to be made but it seems that we encounter some kind of bottleneck at this point, a trade-off between exploration (want the policy to contain enough randomness) and exploitation (want it to be close to the greedy policy).

Off-Policy MC Control

Actually there exists a smart way to avoid the trade-off, which depends on realizing that we were always by default using the policy we want to optimize to simulate MDP trajectory. Could they possibly be separate? That is exactly the idea of off-policy method. The policy we use to simulate MDP is called the **behavior policy** denoted b while the policy we want to optimize is called the **target policy** denoted π. To clarify, **on-policy** learning refers to the case where $b = \pi$ and **off-policy** learning refers to the case where b, π are not necessarily the same. Note that off-policy learning is a milestone in RL and is considered the key to learning most of the practical dynamics in real life.

Let's start with a naive setting of off-policy algorithm that b, π has no connection with each other. Even in this case, we don't want those two policy to be too different, i.e. they shall satisfy the **coverage condition**

$$\forall s, a, \pi(a|s) > 0 \Longrightarrow b(a|s) > 0 \tag{310}$$

the support of b shall contain the support of π, otherwise there's no gain in exploration for off-policy learning. The MDP trajectory will be generated following b but our ultimate goal is to optimize π, as a result, we need the relationship between $q_b(s,a)$ and $q_\pi(s,a)$ to turn the simulation result into what we truly want to optimize

$$\mathbb{E}_\pi(G_t|S_t = s, A_t = a) = \int h(r_{t+1}, r_{t+2}, ..., r_T) \cdot f_{\pi,s,a}(r_{t+1}, s_{t+1}, a_{t+1}, ..., r_T)\, dr_{t+1}\, ds_{t+1}\, da_{t+1} ... dr_T \tag{311}$$

where $G_t = h(R_{t+1}, R_{t+2}, ..., R_T)$ for some deterministic function h and $f_{\pi,s,a}(r_{t+1}, s_{t+1}, a_{t+1}, ..., r_T)$ is the likelihood of MDP conditional on taking policy $\pi, S_t = s, A_t = a$. It's clear that

$$\frac{f_{\pi,s,a}(r_{t+1}, s_{t+1}, a_{t+1}, ..., r_T)}{f_{b,s,a}(r_{t+1}, s_{t+1}, a_{t+1}, ..., r_T)} = \frac{p(s_{t+1}, r_{t+1}|s,a) \cdot \pi(a_{t+1}|s_{t+1}) \cdot p(s_{t+2}, r_{t+2}|s_{t+1}, a_{t+1}) \cdot ... \cdot p(r_T|s_{T-1}, a_{T-1})}{p(s_{t+1}, r_{t+1}|s,a) \cdot b(a_{t+1}|s_{t+1}) \cdot p(s_{t+2}, r_{t+2}|s_{t+1}, a_{t+1}) \cdot ... \cdot p(r_T|s_{T-1}, a_{T-1})} \tag{312}$$

$$= \prod_{k=t+1}^{T-1} \frac{\pi(a_k|s_k)}{b(a_k|s_k)} \tag{313}$$

does not depend on MDP but only depends on the two policy, this is defined as the **importance sampling ratio** (it matches the importance sampling in MC)

$$\rho_{t+1:T-1} = \prod_{k=t+1}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)} \tag{314}$$

and we immediately see that

$$\mathbb{E}_\pi(G_t|S_t = s, A_t = a) = \mathbb{E}_b(\rho_{t+1:T-1} G_t|S_t = s, A_t = a) \tag{315}$$

allows us to represent $q_\pi(s,a)$.

Remark. *As an analogue, the coverage condition is the absolute continuity for measures, the importance sampling ratio is likelihood ratio, a Radon-Nikodym derivative so what we are doing is just the change of measure.*

As a result, adopting the every-visit MC setting, after simulating each trajectory of MDP following policy b, let $T(s,a)$ denote the collection of all times when state-action pair (s,a) is visited. The estimated $q_\pi(s,a)$ is formed as

$$Q(s,a) = \frac{\sum_{t\in T(s,a)} \rho_{t:T-1} G_t}{\sum_{t\in T(s,a)} \rho_{t:T-1}} \tag{316}$$

Remark. *One might be wondering why the normal MC estimator*

$$Q(s,a) = \frac{\sum_{t\in T(s,a)} \rho_{t:T-1} G_t}{|T(s,a)|} \tag{317}$$

is not used here. That's because the normal MC estimator empirically has a dramatically higher variance due to the sensitivity in the ratio ρ. After scaling with the sum of the ratio, the variance shrinks to zero asymptotically when the returns are bounded.

Adopting the same idea in MC control, policy evaluation and improvement are conducted simultaneously, π is formed as a greedy policy w.r.t. Q while b is taken as any soft policy (fully randomized). Luckily, all the updates of ρ, Q can still be conducted in an incremental form. Refer to Alg. 7 for the details.

Algorithm 7 Off-policy MC Control

```
Input: Initial policy $\pi_0$
 1: $\pi = \pi_0$
 2: repeat
 3:   Take $b$ as any soft policy
 4:   Generate sample following policy $b$: $S_0, A_0, R_1, S_1, ..., R_T$
 5:   $W = 1$, importance sampling ratio
 6:   $\forall s, a, C(s,a) = 0$, cumulative sum of importance sampling ratio
 7:   for $t = T-1, T-2, ..., 0$ do
 8:     Record return $G_t$ for every visit to the state-action pair $(S_t, A_t)$
 9:     Update $C(S_t,A_t) = C(S_t,A_t) + W$
10:     Update $Q(S_t,A_t) = Q(S_t,A_t) + \frac{W}{C(S_t,A_t)}[G_t - Q(S_t,A_t)]$
11:     Update with greedy policy $\pi(S_t) = \arg\max_a Q(S_t,a)$
12:     if $A_t \neq \pi(S_t)$ then
13:       Break the loop
14:     else
15:       Update $W = W\frac{1}{b(A_t|S_t)}$
16:     end if
17:   end for
18: until Enough iteration is done
19: $\pi^* = \pi$
Output: Optimal policy $\pi^*$
```

Remark. *Since π is greedy, it's deterministic, if $A_t \neq \pi(S_t)$, then W becomes zero and nothing can be learnt from the same trajectory, that's why we directly break the loop. Otherwise, $\pi(A_t|S_t) = 1$ so we replace the numerator with 1 when updating W.*

Note that this algorithm can be easily modified into a general version where the optimal policy π is not necessary deterministic. One only needs to change the update of π according to some criterion, change the update of W into $W = W\frac{\pi(A_t|S_t)}{b(A_t|S_t)}$ and remove the if condition.

We comment on off-policy MC control that it's model-free and opens the door of evolutionary off-policy learning algorithm. However, it always learns from the tail of the experience and it's very likely that it misses some of the states and actions that tend to appear more in the early stage of the MDP. This causes serious deceleration of learning especially for long-episode games. Moreover, all MC methods have the common disadvantage that one has to wait for the end of one episode, after which learning can be done. This problem is not significant for some games with short length of episode but does matter for games like continuing tasks that does not have a terminal time.

Example: Blackjack

The blackjack game aims to get as close to 21 as possible with poker cards. All face cards are 10 and the ace can be 1 or 11. When the game starts, the dealer and the player each gets 2 cards, the dealer has one card facing up and another facing down. If player gets 21, he wins immediately except the case where the dealer also gets 21. The player can choose to hit (draw an additional card) or stand (stop drawing card). The player loses the game as soon as the sum exceeds 21 and when he stands the dealer begins to do the same thing. The dealer loses the game as soon as the sum exceeds 21 and when he stands he compares the sum with the player to see whether it's a draw/win/lose. The player and the dealer can decide what value each ace takes, either 1 or 11. It's quite obvious that if there is an ace in the first two cards, at least one of them should work as 11 since there is no risk of exceeding 21 if the sum is less than 12.

Blackjack can be formulated as a undiscounted episodic finite MDP. Although it's actually a two-player game, by fixing the dealer's strategy, it becomes a single player game, thus fitted into the setting of single-agent RL. Let's assume that the dealer takes the conservative strategy that he only stands on any sum greater or equal to 17. Rewards are only given based on draw/win/lose to be 0/1/-1 respectively. Assume that there are infinitely many cards provided in the casino (so we cannot calculated how many cards of a certain type are left). An ace is said to be usable if it works as 11 and nonusable if it works as 1.

Let's formulate this blackjack problem in terms of an MDP. The state space is

$$\mathscr{S} = \{(s,u,d)|s,u,d \in \mathbb{N}, 12 \leq s \leq 21, 0 \leq u \leq 1, 1 \leq d \leq 10\} \tag{318}$$

where s stands for the sum of cards in the player's hand, u is the indicator if the player has a usable ace and d stands for the first card the dealer is showing to us. Note that s has minimal value 12 because when the sum is less than 12, the player hits without any risk. The action space always contains two actions

$$\mathscr{A} = \{\text{hit}, \text{stand}\} \tag{319}$$

assume that we are adopting the policy π that is to stand if and only if the sum is 20 or 21, which is a very risky strategy, Fig. 15 and 16 shows the surface of the estimated v_π in two cases depending on whether the player has a usable ace.

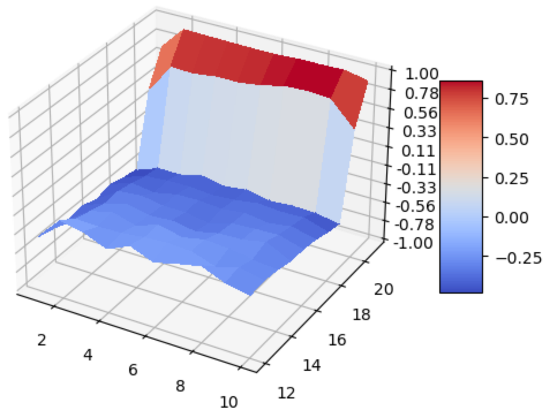


Figure 15: State value function estimated by Monte Carlo when the player has usable Ace

The x-axis stands for the first card of the dealer facing up (1 means an ace), the y-axis stands for the sum of cards in the player's hands, the z-axis stands for the estimated state value $v_\pi(s)$ for such state tuple. 1000000 iterations for Monte Carlo estimation, the estimation is coarse.

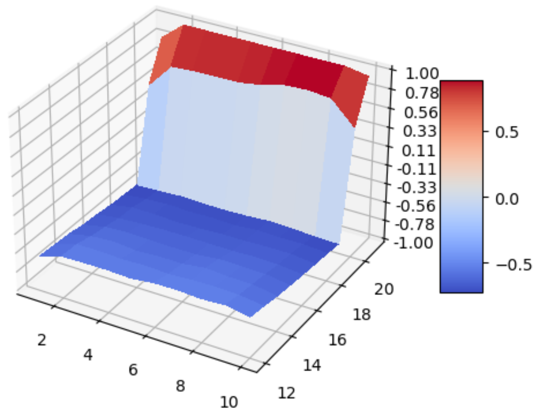


Figure 16: State value function estimated by Monte Carlo when the player has no usable Ace

The x-axis stands for the first card of the dealer facing up (1 means an ace), the y-axis stands for the sum of cards in the player's hands, the z-axis stands for the estimated state value $v_\pi(s)$ for such state tuple. 1000000 iterations for Monte Carlo estimation, the estimation is pretty good.

Some observations can be made based on these two figures. The state value function has a sudden jump when the player's sum is at 20 since the player continues hitting unless the sum is no less than 20. As a result, when the player is at a state less than but close to 20, it's often the case that he will go busted and lose the game. On the other hand, if he successfully reaches state 20, he stops hitting and has a large chance of winning. There is also a drop in the state value where the first card of the dealer is an ace. This is natural since an ace for the dealer is more likely to cause an immediately winning (the probability of drawing 10 is $\frac{4}{13}$ in Blackjack). Notice that in Blackjack, the first-visit MC is actually the same as every-visit MC since the state never repeats in a single game.

At last, we solve out the optimal policy of the player in Blackjack given that the dealer is using the 17-hit strategy, which is shown in Fig. 17 and 18.

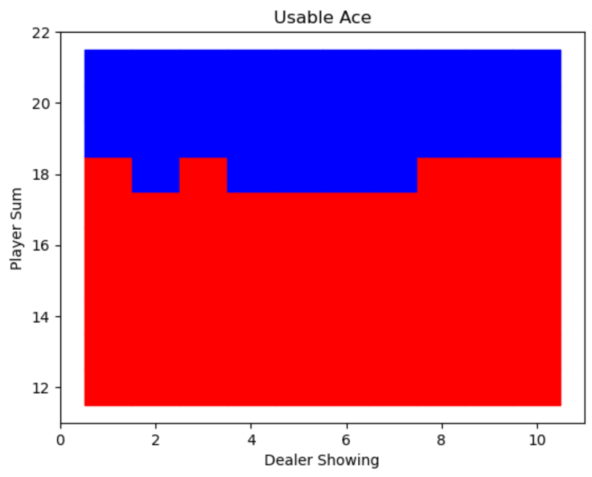


Figure 17: Optimal Policy when the player has usable Ace

Blue block corresponds to stand and red block corresponds to hit.

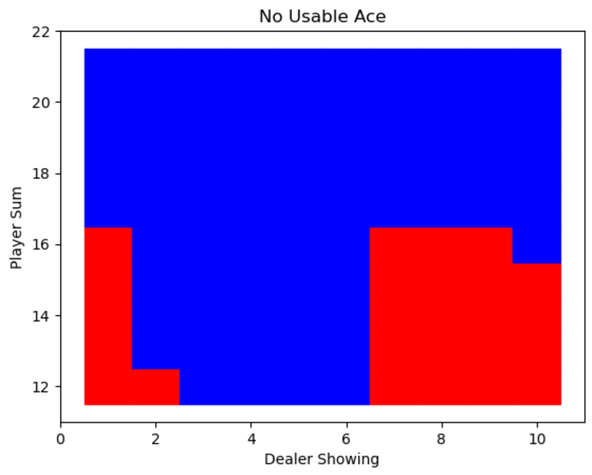


Figure 18: optimal policy when the player has no usable Ace

Blue block corresponds to stand and red block corresponds to hit.

When it comes to the detail of implementation, we use the on-policy MC control with ε-soft policy stated above. Simulation is repeatedly conducted until each state-action pair is visited for at least 100 times to ensure that there is enough exploration. Note that the optimal policy we get is an estimated one and is not exactly optimal, but it's very close to the exact optimal policy.

To interpret our result, when the player has usable ace, he has more freedom to hit since the usable ace protects him from getting busted. In particular, when the dealer shows an ace or a card with large number, the player shall be more risky just in case the dealer has a larger overall point. On the other hand, when the player has no usable ace, he shall be conservative and wait for the dealer to get busted. However, if the dealer shows an ace or a card with large number, the player has to be risky otherwise he is very likely to lose the game.

Last but not least, we remark that off-policy MC control does not behave well in this Blackjack task and requires much more time to converge compared to the on-policy version. This shall not be surprising and is always the case in practice. Generally, off-policy methods are much more powerful but the cost to pay is a longer training time and a more complicated learning algorithm. In the next section, we are going to use the idea from temporal difference learning to put up some really useful off-policy method. We also remark that in this problem the dealer is assumed to always be using the fixed policy. If the dealer changes his policy, we have to relearn the optimal policy. Moreover, if the dealer also changes his policy based on the feedback, that becomes a multi-agent RL (MARL) problem which is much more complicated and we can only expect the convergence to the Nash equilibrium.

Temporal Difference (TD) Methods

Although MC methods are model-free, they are offline in the sense that one always has to wait for the simulation of one episode to end before the learning process. Is there an online version of RL method that could possibly do better than MC? The idea comes from TD method using characterizations or consistent conditions of value function to form temporal difference.

TD(0) Method

Starting from the policy iteration idea, our first TD method only needs to find a way to do policy evaluation since policy improvement can always be done (in a model-based way for v_π and in a model-free way for q_π). Given policy π, consider evaluating v_π, DP method is based on the Bellman consistency equation

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \cdot [r + \gamma \cdot v_\pi(s')] \tag{320}$$

MC method is based on the definition

$$v_\pi(s) = \mathbb{E}_\pi(G_t|S_t = s) \tag{321}$$

it's clear that what is important is actually how we interpret the value function, but not how to calculate mathematically. DP method adopts the perspective of fixed point iteration based on the knowledge of the model (bootstrapping, use current estimate instead of the true v_π), MC method adopts the perspective of viewing value function as conditional expectation (sampling). TD method, on the other hand, values the **iterative consistency structure** of the value function, i.e. TD method views the value function as

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma \cdot v_\pi(S_{t+1})|S_t = s] \tag{322}$$

where v_π appears on both sides of the equation. This equation has the iterative consistency structure in terms of v_π, saying that if we estimate v_π with V at time t, then it's expected that V satisfies

$$\mathbb{E}_\pi[R_{t+1} + \gamma \cdot V^t(S_{t+1}) - V^t(S_t)|S_t = s] = 0 \tag{323}$$

such that it exactly matches the true value function v_π. How do we calculate this conditional expectation is a model-free way? We shall use MC to do the sampling. However, LHS and RHS always does not match in practice and there is error called the **temporal difference (TD) error**

$$\delta_t = R_{t+1} + \gamma \cdot V^t(S_{t+1}) - V^t(S_t) \tag{324}$$

Remark. *The TD error gets its name since it's the error in the estimate V^t that is made at time t and is evaluated based on the state at time $t+1$, i.e. δ_t is only available at time $t+1$ since it contains S_{t+1}. Nevertheless, notice* *that the TD error can be evaluated in a model-free way using MC!*

Our objective in policy evaluation, of course, is to make TD error as small as possible, when $\delta_t > 0$, we want the estimate of $V^t(S_t)$ to increase and when $\delta_t < 0$, we want the estimate $V^t(S_t)$ to decrease. As a result, a natural update is provided as

$$V^{t+1}(S_t) = V^t(S_t) + \alpha[R_{t+1} + \gamma \cdot V^t(S_{t+1}) - V^t(S_t)] \tag{325}$$

where one makes some improvement from V^t to V^{t+1} on the state value of S_t. It's clear at this point that TD method is bootstrapping and requires sampling at the same time, i.e. it **combines DP and MC method** to some extent. Refer to Alg. 8 for the simplest TD(0) method as policy evaluation for v_π using exactly the idea mentioned above.

```
Algorithm 8 TD(0) for policy evaluation $v_\pi$
Input: Given policy $\pi$, hyperparameter $\alpha > 0$
1: repeat
2:   Initial state $S_0$
3:   for $t = 0, 1, 2, ..., T$ do
4:     Select action $A_t$ following $\pi$
5:     Take action $A_t$, observe reward $R_{t+1}$ and state transition to $S_{t+1}$
6:     TD update $V(S_t) = V(S_t) + \alpha[R_{t+1} + \gamma \cdot V(S_{t+1}) - V(S_t)]$
7:   end for
8: until Enough iteration is done
Output: $V$ as estimate for $v_\pi$
```

It's immediately seen that different from that in MC where one waits till the end of the whole episode before learning, TD method learns as soon as new experience comes in, showing its **online** feature.

Remark. *If the estimate $V^t \equiv V$ does not change within a single episode (e.g. MC method), the estimation error can be decomposed into the sum of TD errors*

$$G_t - V(S_t) = \delta_t + \gamma(G_{t+1} - V(S_{t+1})) = ... = \sum_{k=t}^{T-1} \gamma^{k-t}\delta_k \tag{326}$$

Numerical experiments show that TD method converges faster than MC method if both are compared under batch training (after each new episode, all episodes seen so far are repeatedly presented to the algorithm as a batch). Why does TD method do better than MC? There is an example showing the different interpretation of MC and TD for the same set of experience in Barto Sutton book.

Assume that we are provided the following experience in MDP with action selected according to some policy π that does not depend on state (in some context it's called a Markov reward process), the experience provided are

$$\{A, 0, B, 0\}, \{B, 1\}, \{B, 1\}, \{B, 1\}, \{B, 1\}, \{B, 1\}, \{B, 1\}, \{B, 0\} \tag{327}$$

where A, B are states and $0, 1$ are rewards. TD and MC both agree that the state value of B is $\frac{6}{8}$ based on the experience but when it comes to the state value of A they provided totally different estimates.

For MC method, it checks the number of appearance of state A and check the return at state A which is zero, so it claims that the state value of A is zero. On the other hand, for TD method, it updates $V(A)$ as

$$V(A) + \alpha[0 + \gamma \cdot V(B) - V(A)] \tag{328}$$

setting $\gamma = 1, \alpha = 1$, plug in $V(B) = \frac{6}{8}$ we see that after TD update $V(A) = V(B) = \frac{6}{8}$.

To explain this phenomenon, MC directly focuses on the experience, saying that after reaching to state A, we have only seen return taking value zero once so we shall believe that A has state value zero. TD focuses on the MDP state transition, saying that we have observed that A must transit to B, since B has state value $\frac{6}{8}$, we shall believe that A also has state value $\frac{6}{8}$. MC only focuses on the existing experience while TD tries to figure out the intrinsic logic of existing experience and generalize it onto future experience. From a high-level point of view, that's the reason TD is doing better than MC empirically.

More specific analysis on TD methods and a proof of convergence for TD(0) can be found in the well-known paper ***Learning to Predict by the Methods of Temporal Differences by Richard Sutton***.

SARSA

Let's try to build a model-free RL algorithm based on TD idea. To be model-free, we have to evaluate q_π instead of v_π, luckily it's easy to find an iterative consistency structure also for q_π

$$q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma \cdot q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \tag{329}$$

as a result, if we estimate q_π with Q, the TD error will be formed as

$$R_{t+1} + \gamma \cdot Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \tag{330}$$

we shall increase $Q(S_t, A_t)$ if the TD error is positive and decrease $Q(S_t, A_t)$ if the TD error is negative. The algorithm is called SARSA with its name directly meaning state-action-reward-state-action as the standard procedure of the algorithm, refer to Alg. 9 for the details.

Algorithm 9 SARSA: on-policy TD control

```
Input: Hyperparameter $\alpha > 0$
 1: Initial estimate of $Q$
 2: repeat
 3:     Initial state $S_0$
 4:     Select action $A_0 = \arg\max_a Q(S_0, a)$ in an $\varepsilon$-greedy way
 5:     for $t = 0, 1, 2, ..., T$ do
 6:         Take action $A_t$, observe reward $R_{t+1}$ and state transition to $S_{t+1}$
 7:         Select action $A_{t+1} = \arg\max_a Q(S_{t+1}, a)$ in an $\varepsilon$-greedy way
 8:         TD update $Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \cdot Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$
 9:     end for
10: until Enough iteration is done
Output: $Q$ as estimate for $q^*$
```

SARSA basically first estimates Q and then forms the optimal policy as the greedy policy w.r.t. Q. Since the policy used in the simulation is the one that is ε-greedy w.r.t. Q, the behavior policy matches with the target policy and it's an on-policy method. Even if SARSA still suffers from a lot of problems, at least now we have some method which is **model-free and online**. We refer the interested reader to ***Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms by Satinder Singh et. al.*** for an analysis on the convergence of SARSA.

Q-learning

One of the main breakthroughs in RL is the Q-learning algorithm propose by Watkins in 1989. Instead of finding iterative consistency structure for q_π, Q-learning finds such structure for q^*, which is possible since we have the Bellman optimality equation

$$q^*(s,a) = \sum_{s',r} p(s',r|s,a) \cdot \left[r + \gamma \cdot \max_{a'} q^*(s',a')\right] \tag{331}$$

the transition kernel p can be absorbed into the expectation so it can be evaluated through MC

$$q^*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \cdot \max_{a'} q^*(S_{t+1},a') \middle| S_t = s, A_t = a\right] \tag{332}$$

resulting in the TD error if we estimate q^* by Q

$$R_{t+1} + \gamma \cdot \max_a Q(S_{t+1},a) - Q(S_t,A_t) \tag{333}$$

this provides the Q-learning algorithm, refer to Alg. 10 for details.

Algorithm 10 Q-learning: off-policy TD control

```
Input: Hyperparameter $\alpha > 0$
1: Initial estimate of $Q$
2: repeat
3:     Initial state $S_0$
4:     for $t = 0, 1, 2, ..., T$ do
5:         Select action $A_t = \arg\max_a Q(S_t, a)$ in an $\varepsilon$-greedy way
6:         Take action $A_t$, observe reward $R_{t+1}$ and state transition to $S_{t+1}$
7:         TD update $Q(S_t,A_t) = Q(S_t,A_t) + \alpha[R_{t+1} + \gamma \cdot \max_a Q(S_{t+1},a) - Q(S_t,A_t)]$
8:     end for
9: until Enough iteration is done
Output: $Q$ as estimate for $q^*$
```

The Q-learning is off-policy. Distinguishing on-policy and off-policy methods is not easy in the TD setting where the target policy is not clear. In this context, we say that on-policy methods improve the current policy while off-policy methods learn the optimal policy directly. To be specific, in SARSA, when learning Q, we are assuming that we will stick to the sam strategy π in the future. This presents an improvement on the current policy so it's on-policy. On the other hand, in Q-learning, our goal is to learn the optimal value function directly so it's off-policy.

Remark. *On-policy and off-policy methods fit with different situations. On-policy methods learn safer strategy, has better online performance and converges faster while off-policy methods can utilize experience replay can is more likely to find the optimal policy. On-policy methods fit more with physical robot training while off-policy methods fit more with a simulator.*

Q-learning is one of the most important results in RL due to its numerical superiority over other methods and its extension in DeepRL has still been heavily used in these days.

Improvements on Q-learning

Several improvements on Q-learning has been proposed motivated by different empirically observed problems of Q-learning.

The **experience replay** is introduced in the paper ***Self-improving reactive agents based on reinforcement learning, planning and teaching by Longji Lin***. This technique is motivated by the fact TD learning is a slow process of temporal credit assignment, i.e. the way to assign credit or blame to each individual situation to adjust its decision making. As a matter of fact, the convergence speed of Q-learning is typically slow. On the other hand, the Q-learning algorithm directly throws away the experience after using it only for once, which is wasteful since some experiences are rare and costly to obtain. This gives natural rise to the experience replay mechanism memorizing past experiences and repeatedly presenting those experiences to Q-learning.

When it comes to the details of experience replay, the vanilla experience replay just fills in the replay buffer with all the experience one gets and randomly sample a batch of experience to train the agent. This does not cause any harm to the tabular Q-learning but might cause issue for the convergence of DQN (presented later). A possible remedy is to make sure that only positive experience leaves a deeper impression for Q-learning. With the "experience" referring to the tuple $(S_t, A_t, R_{t+1}, S_{t+1})$, we only replay the experience where the action is a policy action, i.e. A_t has a large probability of being chosen according to the currently trained policy and the current state S_t. That is to say, we set up a threshold and only replay the experience if $\pi(A_t|S_t)$ exceeds the threshold. To prevent over-fitting issues, i.e. Q-learning learns a certain experience for too many times and loses its generality, after each episode the agent replays a fixed amount of the most recent episodes. Recent episodes are exponentially more likely to be chosen in experience replay.

Remark. *When one sees the term "experience replay" in RL literature, it often means the vanilla experience replay, without taking into account the policy action or the preference on more recent experiences. One can customize the frequency of sampling from the replay buffer, the frequency of updating parameters etc.*

The second important idea is called **double Q-learning** from ***Double Q-learning by Hado van Hasselt***. The motivation is based on the **maximization bias** that inside the TD update for Q, the maximum among all actions always provides an upward bias, especially in a stochastic environment. This fact can be observed through an easy example where all action values are zero but the estimate Q has some positive values and some negative values due to randomness. In this case, $\max_a Q(S_{t+1}, a)$ might always be positive, causing an overestimation in the action value.

To solve this problem, we have to realize that maximization bias appears because the same Q is used as an estimate of the action value and to compute the maximum. The double Q-learning method maintains two versions of the estimate for action value function denoted Q_1, Q_2. At each time step, one of them is responsible as an estimate of the action value and the other one is responsible for computing the maximum, e.g. when Q_1 works as the estimate, it shall be updated in Q-learning and the maximum shall be calculated as $Q_2(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a))$.

Remark. *To get the intuition of why double Q-learning works, assume that the true action value is always zero and that for fixed state s, all components of $\{Q_1(s,a)\}_{a\in\mathscr{A}(s)}, \{Q_2(s,a)\}_{a\in\mathscr{A}(s)}$ are from a probability distribution* *symmetric w.r.t. zero respectively. For simplicity, we assume the independence between all components within* Q_1 *and* Q_2*, also the independence between* Q_1 *and* Q_2*.*

It's clear that $\max_a Q_1(s,a), \max_a Q_2(s,a)$ *are both positive, causing the maximization bias. But if we set* $a^* = \arg\max_a Q_1(s,a)$*, it's clear that* $a^* \sim U(\mathscr{A}(s))$ *is uniformly distributed on the action space. From the independence of* a^* *and* Q_2*, we see that*

$$\mathbb{E}Q_2(s,a^*) = \mathbb{E}[\mathbb{E}(Q_2(s,a^*)|Q_2)] = \mathbb{E}\sum_{a\in\mathscr{A}(s)} Q_2(s,a)\cdot\mathbb{P}(a^*=a|Q_2) \tag{334}$$

$$= \mathbb{E}\sum_{a\in\mathscr{A}(s)} Q_2(s,a)\cdot\mathbb{P}(a^*=a) \tag{335}$$

$$= \frac{1}{|\mathscr{A}(s)|}\sum_{a\in\mathscr{A}(s)} \mathbb{E}Q_2(s,a) = 0 \tag{336}$$

since $\forall a, Q_2(s,a)$ *is symmetrically distributed w.r.t. zero. The maximization bias is gone.*

Note that it's not a good idea to always assign Q_1 for the calculation of maximum and Q_2 for estimating the action value since Q_1 will never get updated, resulting in a large underfitting error when evaluating the maximum. Instead, a natural way is to provide equal opportunity for Q_1, Q_2 to exchange their roles, i.e. at each time step, Q_1 has $\frac{1}{2}$ probability responsible for the calculation of the maximum and $\frac{1}{2}$ probability responsible for estimating the action value. The complete algorithm of double Q-learning with experience replay is presented below as Alg. 11.

Algorithm 11 Double Q-learning with experience replay

```
Input: Hyperparameter $\alpha > 0$
1: Initial estimate of $Q$
2: repeat
3:     Initial state $S_0$
4:     for $t = 0, 1, 2, ..., T$ do
5:         Select action $A_t = \arg\max_a \{Q_1(S_t,a) + Q_2(S_t,a)\}$ in an $\varepsilon$-greedy way
6:         Take action $A_t$, observe reward $R_{t+1}$ and state transition to $S_{t+1}$
7:         Update the replay buffer
8:         Sample a batch of past experience $\{(s_j, a_j, r_j, s'_j)\}_j$ from the buffer
9:         TD update, with probability $\frac{1}{2}$, conduct one of the following updates
```

$$\begin{cases} Q_1(s_j,a_j) = Q_1(s_j,a_j) + \alpha[r_j + \gamma\cdot Q_2(s'_j, \arg\max_a Q_1(s'_j,a)) - Q_1(s_j,a_j)] \\ Q_2(s_j,a_j) = Q_2(s_j,a_j) + \alpha[r_j + \gamma\cdot Q_1(s'_j, \arg\max_a Q_2(s'_j,a)) - Q_2(s_j,a_j)] \end{cases} \tag{337}$$

```
10:    end for
11: until Enough iteration is done
Output: $Q_1, Q_2$ both as estimates for $q^*$
```

Fitted Q-iteration (FQI)

So far we have been focusing on tabular RL problems, in which the state space and action space are assumed to be finite and always have a quite small size. However, practical RL problems like chess, Go etc. has a very large state space with a finite action space. As an example, in Go we have 361 crossings and each crossing can hold a black piece or a white piece or nothing, approximately the state space contains 3^{361} elements, such a large number that it's impossible to sweep through all the states. Oppositely, there are not a lot of available actions, e.g. in Go if we hold black, at most we have 361 actions to take, dropping a new black piece on any of the crossings. As a result, in order to make Q-learning useful in practice, it has to be able to deal with a **continuous state space**. The fitted Q-iteration (FQI) was proposed under this context to combine the idea of Q-learning with other prediction methods in machine learning.

To get out of the tabular setting, it suffices to introduce a parametrized estimator $Q_\theta(s,a)$ with parameter θ. Whenever state-action pair (s,a) is observed, we throw it into the estimator to get the predicted action value. Recall the vanilla Q-learning update $Q(S_t,A_t) = Q(S_t,A_t)+\alpha[R_{t+1}+\gamma\cdot\max_a Q(S_{t+1},a)-Q(S_t,A_t)]$, let's take the learning rate $\alpha=1$ for the purpose of simplicity so

$$Q(S_t,A_t) = R_{t+1}+\gamma\cdot\max_a Q(S_{t+1},a) \tag{338}$$

write down the FQI version

$$Q_\theta(S_t,A_t) = R_{t+1}+\gamma\cdot\max_a Q_\theta(S_{t+1},a) \tag{339}$$

and keep in mind that our objective now is to update the parameter θ such that Q_θ is close to q^*. However, this causes the difficulty that LHS and RHS shares exactly the same θ. This situation is fine when the parametrized estimator is e.g. a linear estimator in θ but otherwise it provides an implicit equation w.r.t. θ and is hard to solve. But wait, why don't we bootstrap? It's unnecessary to solve out the accurate θ immediately but we can deal with this problem in an iterative way. That is to say, the FQI update shall look like

$$Q^n_\theta(S_t,A_t) = R_{t+1}+\gamma\cdot\max_a Q^{n-1}_\theta(S_{t+1},a) \tag{340}$$

where Q^n_θ denotes the estimate of action value in the n-th iteration parametrized by θ. In other words, we first calculate $y = R_{t+1}+\gamma\cdot\max_a Q^{n-1}_\theta(S_{t+1},a)$ using the old parameter θ, then do the regression $Q^n_\theta(S_t,A_t)=y$ to update the parameter θ.

Remark. *The parametrized estimator has a lot of different choices, e.g. the KNN predictor, the decision tree predictor, the neural network etc. They all fit well into the FQI framework. When it comes to using neural network (NN), deep learning starts to come into RL and we defer the discussion of DQN (deep Q-network) into a later context.*

The complete algorithm of FQI is provided in Alg. 12.

Remark. *The FQI we provided here is model-free, off-policy but **offline** since it requires us to first prepare the*

Algorithm 12 Fitted Q-Iteration (FQI)

```
Input: A pre-determined parametrized prediction model $Q_\theta$
 1: Initial estimate of $Q$
 2: repeat
 3:     Initial state $S_0$
 4:     for $t = 0, 1, 2, ..., T$ do
 5:         Select action $A_t$ according to some given policy $\pi$ (e.g. uniformly random on the action space)
 6:         Take action $A_t$, observe reward $R_{t+1}$ and state transition to $S_{t+1}$
 7:     end for
 8: until Enough training samples are generated
 9: Initial parameter $\theta$
10: repeat
11:     Evaluate $y = R_{t+1} + \gamma \cdot \max_a Q_\theta(S_{t+1}, a)$
12:     Fit $Q_\theta(S_t, A_t) = y$ to update $\theta$
13: until Enough number of iteration is done
Output: $\theta$ such that $Q_\theta$ is the estimate for $q^*$
```

training samples before the learning process begins. In practice, the training of FQI is conducted in a batched style, i.e. each FQI iteration is done only for a subset of all the training samples simultaneously. As a result, the fitting operation is also carried out in a batched style, which is more efficient.

However, it's easy to propose an online version of FQI. Instead of fitting the batch of training samples after observing the MDP trajectories, one use gradient descent w.r.t. θ so that it's possible to update θ after a decision is made at each time step.

Besides the online feature, FQI is still sweeping through all the actions in the action space (maximum w.r.t. action a in Q-learning update) so it fails to solve RL problem with a large action space.

Deep Q-Network (DQN)

DQN is the simplest but one of the most effective DeepRL algorithms we will learn. The idea is simple: replace the tabular estimate for Q with a neural network and update the parameter of the NN through minimizing a loss function.

When using NN to approximate action value, notice that the action space has to be finite since Q-learning is based on BOE of q^* which contains the term $\max_a Q(S_{t+1}, a)$. As a result, DQN allows us to deal with large state space but not large action space due to the difficulty of formulating this maximum among all possible actions. With a finite action space given, DQN accepts the state as input of the NN and action value Q as the output of the NN. Since there are $|\mathscr{A}|$ actions available, the input dimension of the NN shall match the dimension of the state while the output dimension shall be equal to $|\mathscr{A}|$, i.e., outputting action value estimates $Q(s, a)$ for all possible actions $a \in \mathscr{A}$ simultaneously when a single state $s \in \mathscr{S}$ is the input.

However, the plain DQN suffers from the so-called **deadly triad** in RL, meaning that off-policy learning, bootstrapping, and function approximation altogether results in the instability of the algorithm. Q-learning is off-policy, bootstrapping (it's TD method) and DQN uses NN to approximate action value function so it exactly hits the deadly triad. That's why extra improvements are needed to make DQN a practically useful algorithm.

Remark. *To explain intuitively why the deadly triad exists, take DQN as an example. Imagine that state s only transits to a similar state s' under the only action a while multiple actions are available at state s'. The existence of function approximation implies that for similar states s, s', and similar actions a, a', they have similar action values $Q(s, a) \approx Q(s', a')$. The off-policy attribute determines that the experience from the behavioral policy does not match exactly with the target policy one is updating, e.g. in Q-learning the target policy is greedy but the behavioral policy is ε-greedy. Bootstrapping means that estimates are used in place of actual rewards and complete returns. As a result, with those combined together, if in the last episode one observes $Q(s', a') = k$ and one is currently getting reward 1 by taking (s, a), then Q-learning updates $Q(s, a) = k + 1$ to encourage taking (s, a). Function approximation comes into play that in the next episode, $Q(s', a') \approx k + 1$ so Q-learning updates $Q(s, a) = k + 2$ once more. In this procedure, $Q(s, a)$ blows up, causing the instability issue.*

The vanilla **replay buffer** is typically equipped for a better performance. In addition, DQN has essential difference from tabular Q-learning that in tabular Q-learning one updates a single action value after one step. However, in DQN one is updating all the action values after one step (since all the NN parameters are updated), which might cause instability issues like ”catastrophic forgetting”, i.e., DQN suddenly craps and starts to take bad actions. It's worth noting that this phenomenon still happens when the stepsize hyperparameter is sufficiently small. The **target network** adopts the idea that it is an out-of-date copy of the DQN, e.g., aligned to the DQN per 1000 steps taken. In this case, the target network has the general idea how well DQN is currently doing. Based on this intuition, adding a target network to compute the TD target allows DQN to explore based on what it has already achieved instead of retraining itself how to play the entire game after every move.

We provide the complete procedure of DQN as Alg. 13.

```
Algorithm 13 Deep Q-network (DQN)
Input: DQN with parameter $\theta$, a replay buffer
 1: Target network parameter $\theta' = \theta$
 2: repeat
 3:     Initial state $S_0$
 4:     for $t = 0, 1, 2, ..., T$ do
 5:         Select action $A_t = \arg\max_a Q_\theta(S_t, a)$ with exploration (e.g., $\varepsilon$-greedy)
 6:         Take action $A_t$, observe reward $R_{t+1}$ and state transition to $S_{t+1}$
 7:         Store the experience $(S_t, A_t, R_{t+1}, S_{t+1})$ in the replay buffer
 8:         Sample a random minibatch of experiences $\left\{(s_j, a_j, r_j, s'_j)\right\}_j$ from the replay buffer for learning
 9:         Evaluate $y_j = r_j + \gamma \cdot \max_a Q_{\theta'}(s'_j, a)$ as the TD target
10:         Compute the loss $\sum_j (y_j - Q_\theta(s_j, a_j))^2$ and do SGD to update $\theta$
11:     end for
12:     Update $\theta'$ after a number of steps (through copying $\theta$ or Polyak averaging)
13: until Enough number of iteration is done
Output: Updated DQN as the estimate for $q^*$
```

Example: CartPole

The CartPole example has a cart with a pole on the top binded to the cart. When the cart moves, the pole tilts left or right according to the physics law and the objective is to always keep the pole vertical. In this example, the state $s = (s_c, v_c, s_p, v_p) \in \mathbb{R}^4$ has its components s_c to be the position of the cart v_c the velocity of the cart, s_p the position (angle) of the pole, v_p the angular velocity of the pole. There are only two actions: move the cart left or move it right. The agent receives reward one at each time step if the pole is close to vertical and the game ends if the pole falls to horizontal level.

We first prepare the training samples following a uniform policy, FQI has been used with a simple KNN predictor. Surprisingly, after 20 FQI iterations, it can achieve a very good performance. Refer to the notebook for the codes and the video showing the performance of the agent trained by FQI, DQN and DQN with target network.

Convergence of Q-learning

Unlike many other RL methods for which convergence results are not well established, the convergence of the vanilla Q-learning can be proved quite easily thanks to the tools in stochastic optimization theory. We are going to introduce two main perspectives here for the convergence proof. One perspective is making use of action replay process to relate the Q-learning procedure to the optimal action value function of another particularly constructed MDP. The other perspective is making use of a more advanced tool from stochastic optimization theory, and the proof is greatly simplified. The main result is provided in theorem 11.

Theorem 11. *For a finite MDP with deterministic and bounded reward function r, the Q-learning algorithm guarantees the almost sure convergence to the optimal action value function q^* as long as the* ***Robbins-Monro condition***

$$\sum_t \alpha_t(s,a) = \infty, \sum_t \alpha_t^2(s,a) < \infty \tag{341}$$

holds for $\forall (s,a) \in \mathscr{S} \times \mathscr{A}$ almost surely. Here $\alpha_t(s,a)$ denotes the step size taken at time t when the current state-action pair is (s,a) and it's assumed to always take value in $[0,1]$.

Remark. *The Robbins-Monro condition is the fundamental condition for step size under which a stochastic optimization algorithm converges. The part $\sum_t \alpha_t^2(s,a) < \infty$ means that step size cannot be too large such that the update gain stability. The part $\sum_t \alpha_t(s,a) = \infty$ means that step size cannot be too small such that the update has the ability to overcome stochastic perturbation and correct itself.*

As an important remark, the necessary condition for Robbins-Monro condition to hold is that all state-action pairs are visited infinitely often on the infinite time horizon. This emphasizes the importance of exploration once again from a different point of view.

The first proof of convergence is carried out under the action replay process. We refer the readers to the nice paper ***An elementary Proof that Q-learning converges almost surely by Matthew Regehr and Alex Ayoub*** for the complete details. Here we only provide the basic idea of the proof since this action replay technique cannot be generalized to proving the convergence of other methods. What is more important behind the proof, is the general theorem of stochastic optimization we will mention in the later context.

Let's introduce some notations before we start. Let $\mathscr{T}_{(s,a)}$ denote the collection of all random times the MDP visits the state-action pair (s,a). The action replay process is another MDP with state space $\hat{\mathscr{S}} = \mathscr{S} \times \mathbb{N}$ and the same action space as the original one. Its transition kernel is denoted as $\hat{p}$, given by

$$\forall t' \in \mathscr{T}_{(s,a)} \cap [0,t), \hat{p}((S_{t'+1},t')|(s,t),a) = \alpha_{t'} \cdot \prod_{\tau \in \mathscr{T}_{(s,a)} \cap (t',t)} (1-\alpha_\tau) \tag{342}$$

basically, the distribution of this MDP is associated with the step size one has taken in the past. $\hat{p}((S_{t'+1},t')|(s,t),a)$ can be understood as the likelihood of action a being replayed at time $t'+1$ seeing state $S_{t'+1}$ given that (s,a) has been visited at time t' and that in a future time t the state-action pair (s,a) has been visited again. Heuristically, a larger $\alpha_{t'}$ means a larger update on the action value of (s,a) at time t' so there is a larger probability retaking action

a' at time $t'+1$. Conversely, when (s,a) is visited between time $t'+1$ and t, a larger α_τ means a larger update on the action value of (s,a) at time τ. Since we are given that (s,a) has been revisited at time t, a larger α_τ means that one has to make up for more missing information in (s,a), which means that before making up for the missing information the likelihood of replaying the action should be smaller. The reward of this new MDP is

$$\hat{r}((s,t),a) = r(s,a) \cdot \sum_{t' \in \mathscr{T}_{(s,a)} \cap [0,t)} \hat{p}((S_{t'+1}, t')|(s,t),a) \tag{343}$$

so one receives more reward if the action a is more likely to be replayed and of course the reward does not exceed that of the original MDP.

What's the connection between this action replay MDP and Q-learning? Let's denote $Q_t(s,a)$ as the estimated action value of (s,a) at time t updated using step size $\alpha_1, ..., \alpha_t$, it can be proved that

$$\hat{q}^*((s,t),a) = Q_t(s,a) \tag{344}$$

the optimal action value function $\hat{q}^*$ of this new MDP coincides with the Q-learning iterates! More than that, the limit of this constructed MDP coincides with the original MDP in the sense that

$$\hat{r}((s,t),a) \to r(s,a), \sum_{t' \in \mathscr{T}_{(s,a)} \cap [0,t)} \hat{p}((s',t')|(s,t),a) \to p(s'|s,a)\ (t \to \infty) \tag{345}$$

if Robbins-Monro condition holds (from Robbins-Monro theorem). The action replay MDP bridges the gap between $Q_t(s,a)$ and $q^*(s,a)$ so a triangle inequality type estimate can be built, proving the convergence of Q-learning.

The second perspective provides an easy proof of Q-learning based on the following theorem from stochastic optimization

Theorem 12. *If process $\{\Delta_t\}$ takes value in $\mathbb{R}^n$ and*

$$\Delta_{t+1} = (1-\alpha_t)\Delta_t + \alpha_t F_t \tag{346}$$

where $\{\Delta_t\}$ is adapted to the filtration $\{\mathscr{F}_t\}$, if the following conditions hold

$$\alpha_t \in [0,1], \sum_t \alpha_t = \infty, \sum_t \alpha_t^2 < \infty \tag{347}$$

$$\exists 0 < \gamma < 1, ||\mathbb{E}(F_t|\mathscr{F}_t)||_\infty \le \gamma ||\Delta_t||_\infty \tag{348}$$

$$\exists C > 0, Var(F_t|\mathscr{F}_t) \le C(1 + ||\Delta_t||_\infty^2) \tag{349}$$

then $\Delta_t \overset{a.s.}{\to} 0\ (t \to \infty)$.

Let's rewrite the update of Q-learning

$$Q_{t+1}(S_t, A_t) - q^*(S_t, A_t) = (1-\alpha_t)[Q_t(S_t, A_t) - q^*(S_t, A_t)] + \alpha_t[r(S_t, A_t) + \gamma \cdot \max_a Q_t(S_{t+1}, a) - q^*(S_t, A_t)] \tag{350}$$

the form matches that in the theorem above. It suffices to identify Δ_t as $Q_t(S_t, A_t) - q^*(S_t, A_t)$ and F_t as $r(S_t, A_t) + \gamma \cdot \max_a Q_t(S_{t+1}, a) - q^*(S_t, A_t)$. We only need to bound the conditional expectation and conditional variance from above.

$$\begin{aligned}\mathbb{E}(F_t|\mathscr{F}_t) &= \mathscr{T}Q_t(S_t, A_t) - q^*(S_t, A_t) &(351)\\ &= \mathscr{T}Q_t(S_t, A_t) - \mathscr{T}q^*(S_t, A_t) &(352)\\ &\le \gamma \cdot ||Q_t - q^*||_\infty = \gamma \cdot ||\Delta_t||_\infty &(353)\end{aligned}$$

where $\mathscr{T}$ is the Bellman optimality operator and has been proved to be a contraction mapping under infinity norm. At last, verify the condition for the conditional variance,

$$\begin{aligned}Var(F_t|\mathscr{F}_t) &= Var\left(\gamma \cdot \max_a Q_t(S_{t+1}, a)|\mathscr{F}_t\right) &(354)\\ &= \gamma^2 \cdot Var\left(\max_a Q_t(S_{t+1}, a)|\mathscr{F}_t\right) &(355)\\ &\le \gamma^2 \cdot \mathbb{E}\left(\left[\max_a Q_t(S_{t+1}, a)\right]^2 |\mathscr{F}_t\right) &(356)\\ &= \gamma^2 \cdot \sum_s p(s|S_t, A_t) \cdot \left[\max_a Q_t(s, a)\right]^2 &(357)\\ &\le \gamma^2 \cdot \sum_s p(s|S_t, A_t) \cdot ||\Delta_t + q^*(S_t, A_t)||_\infty^2 &(358)\end{aligned}$$

since the reward function is bounded, q^* is uniformly bounded for all state-action pairs and it's obvious that $\exists C > 0, Var(F_t|\mathscr{F}_t) \le C(1 + ||\Delta_t||_\infty^2)$. This proves the convergence of Q-learning.

Remark. *The key takeaway here is to realize that stochastic optimization theory is the right tool to use when it comes to proving the convergence of an algorithm in a stochastic environment. The essential objective of stochastic optimization is to investigate the stochastic recurrence relationship that has the self correction property, i.e. corrects itself and converges to the right limit under stochastic perturbations. We present a* ***fundamental result by Dvoretzky*** *here for the purpose of completeness.*

Theorem 13 (Dvoretzky)**.** *Consider X_1 as arbitrary random variable,*

$$X_{n+1} = T_n(X_1, ..., X_n) + W_n \tag{359}$$

where $\{X_n\}$ is adapted to the filtration $\mathscr{F}_n$ and $W_n \in \mathscr{F}_{n+1}$. If the following conditions are satisfied

$$\mathbb{E}(W_n|X_1, ..., X_n) = 0, \sum_n \mathbb{E}W_n^2 < \infty \tag{360}$$

$$|T_n - x^*| \leq \max\{\alpha_n, (1+\beta_n)|X_n - x^*| - \gamma_n\} \tag{361}$$

$$\alpha_n, \beta_n, \gamma_n \geq 0, \alpha_n \to 0, \sum_n \beta_n < \infty, \sum_n \gamma_n = \infty \tag{362}$$

then $X_n \overset{a.s.}{\to} x^*$ $(n \to \infty)$.

Typically x^* is taken as zero and X_n is taken as the error process to prove the convergence of the algorithm. We shall see stochastic optimization theory coming into play a lot of times in the future when we want to prove the convergence of some algorithm, but the Dovretzky's theorem is always one of the most important tools to keep in mind.

Policy Gradient Method (PG)

Policy gradient methods focus on updating the policy π instead of forming an estimate on the value function. This is especially useful when one has to deal with a **large action space**. Recall that we have seen FQI as an extension of Q-learning that deals with large state space but it's still infeasible for large action space. In order to explore through a large enough family of policy, we parametrize the policy as $\pi(a|s,\theta)$, denoting the dependence of the policy on the parameter $\theta \in \mathbb{R}^{d'}$. It suffices to find a way to update θ such that $\pi(a|s,\theta)$ converges to the optimal policy $\pi^*(a|s)$.

Policy Gradient Theorem

We assume that the initial state $S_0 \sim \nu$ follows initial distribution ν. The optimal parameter θ shall maximize the expected return at time 0, i.e. $\sum_{s_0\in\mathscr{S}} \nu(s_0)\cdot\mathbb{E}_\pi(G_0|S_0=s_0)$. As a result, it suffices to maximize $\mathbb{E}_\pi(G_0|S_0=s_0)$ for each state $s_0\in\mathscr{S}$ since μ exhibits no dependence on θ. We define

$$J(\theta) = \mathbb{E}_\pi(G_0|S_0=s_0) \tag{363}$$

and our goal is to find the optimal parameter θ that maximizes $J(\theta)$. The policy gradient theorem below provides an important representation of the gradient of $J(\theta)$ w.r.t. θ.

Theorem 14 (Policy Gradient Theorem)**.**

$$\nabla J(\theta) \propto \sum_{s\in\mathscr{S}} \mu(s) \sum_{a\in\mathscr{A}} q_\pi(s,a)\nabla\pi(a|s,\theta) \tag{364}$$

*where μ is the **on-policy distribution** defined as*

$$\eta(s) = \sum_{k=0}^{\infty} \gamma^k \mathbb{P}_\pi\left(S_k = s|S_0=s_0\right), \mu(s) = \frac{\eta(s)}{\sum_{s'\in S}\eta(s')} \tag{365}$$

which is a probability measure on $\mathscr{S}$.

Proof. All gradients below are taken w.r.t. θ that

$$\begin{aligned}\nabla J(\theta) &= \nabla \sum_{a\in\mathscr{A}} \pi(a|s_0,\theta) q_\pi(s_0,a) &(366)\\ &= \sum_{a\in\mathscr{A}} \left[\nabla\pi(a|s_0,\theta)q_\pi(s_0,a) + \pi(a|s_0,\theta)\nabla q_\pi(s_0,a)\right] &(367)\end{aligned}$$

use the relationship between q_π and v_π that $q_\pi(s,a)=\sum_{s',r}p(s',r|s,a)\cdot[r+\gamma\cdot v_\pi(s')]$,

$$\nabla J(\theta)=\sum_{a\in\mathscr{A}}\left(\nabla\pi(a|s_0,\theta)q_\pi(s_0,a)+\pi(a|s_0,\theta)\nabla\sum_{s',r}p(s',r|s_0,a)\cdot[r+\gamma\cdot v_\pi(s')]\right) \tag{368}$$

$$=\sum_{a\in\mathscr{A}}\left(\nabla\pi(a|s_0,\theta)q_\pi(s_0,a)+\pi(a|s_0,\theta)\gamma\sum_{s'}p(s'|s_0,a)\cdot\nabla v_\pi(s')\right) \tag{369}$$

since $J(\theta)=v_\pi(s_0)$, this provides a characterization of ∇v_π. We use this equation iteratively to replace the gradient on the RHS,

$$\nabla v_\pi(s_0)=\sum_{a\in\mathscr{A}}\nabla\pi(a|s_0,\theta)q_\pi(s_0,a)+\gamma\sum_{a\in\mathscr{A}}\pi(a|s_0,\theta)\sum_{s'}p(s'|s_0,a)\cdot \tag{370}$$

$$\sum_{a'\in\mathscr{A}}\left(\nabla\pi(a'|s',\theta)q_\pi(s',a')+\gamma\pi(a'|s',\theta)\sum_{s''}p(s''|s',a')\cdot\nabla v_\pi(s'')\right) \tag{371}$$

We denote $f(s)=\sum_{a\in\mathscr{A}}\nabla\pi(a|s,\theta)\cdot q_\pi(s,a)$, this allows us to write the equation above in a simpler way that

$$\nabla v_\pi(s_0)=f(s_0)+\gamma\sum_{a\in\mathscr{A}}\pi(a|s_0,\theta)\sum_{s'}p(s'|s_0,a)\sum_{a'\in\mathscr{A}}\nabla\pi(a'|s',\theta)q_\pi(s',a') \tag{372}$$

$$+\gamma^2\sum_{a\in\mathscr{A}}\pi(a|s_0,\theta)\sum_{s'}p(s'|s_0,a)\sum_{a'\in\mathscr{A}}\pi(a'|s',\theta)\sum_{s''}p(s''|s',a')\cdot\nabla v_\pi(s'') \tag{373}$$

$$=f(s_0)+\gamma\sum_{a\in\mathscr{A},s'}f(s')\pi(a|s_0,\theta)p(s'|s_0,a) \tag{374}$$

$$+\gamma^2\sum_{s''}\sum_{s'}\sum_{a\in\mathscr{A}}\pi(a|s_0,\theta)p(s'|s_0,a)\sum_{a'\in\mathscr{A}}\pi(a'|s',\theta)p(s''|s',a')\cdot\nabla v_\pi(s'') \tag{375}$$

$$=f(s_0)+\gamma\sum_{s'}f(s')\mathbb{P}_\pi\left(S_1=s'|S_0=s_0\right) \tag{376}$$

$$+\gamma^2\sum_{s'',s'}\mathbb{P}_\pi\left(S_1=s'|S_0=s_0\right)\mathbb{P}_\pi\left(S_2=s''|S_1=s',S_0=s_0\right)\nabla v_\pi(s'') \tag{377}$$

$$=f(s_0)+\gamma\sum_{s'}f(s')\mathbb{P}_\pi\left(S_1=s'|S_0=s_0\right)+\gamma^2\sum_{s''\in S}\mathbb{P}_\pi\left(S_2=s''|S_0=s_0\right)\nabla v_\pi(s'') \tag{378}$$

Keep doing such iteration, we will see that

$$\nabla v_\pi(s_0)=\sum_{k=0}^\infty\sum_s f(s)\gamma^k\mathbb{P}_\pi\left(S_k=s|S_0=s_0\right) \tag{379}$$

$$=\sum_s\sum_{k=0}^\infty\gamma^k\mathbb{P}_\pi\left(S_k=s|S_0=s_0\right)\sum_{a\in\mathscr{A}}\nabla\pi(a|s,\theta)\cdot q_\pi(s,a) \tag{380}$$

$$=\sum_s\eta(s)\sum_{a\in\mathscr{A}}\nabla\pi(a|s,\theta)\cdot q_\pi(s,a) \tag{381}$$

according to the definition of η. We have the representation that

$$\nabla J(\theta) = \left(\sum_{s'} \eta(s')\right) \cdot \sum_{s} \mu(s) \sum_{a\in\mathscr{A}} \nabla\pi(a|s,\theta) \cdot q_\pi(s,a) \tag{382}$$

$$\propto \sum_{s} \mu(s) \sum_{a\in\mathscr{A}} \nabla\pi(a|s,\theta) \cdot q_\pi(s,a) \tag{383}$$

since $\sum_{s'} \eta(s')$ has no dependence on the state.

$\square$

Remark. *To understand the meaning of $\eta(s), \mu(s)$, let's first consider the episodic case where $\gamma = 1$. Then*

$$\eta(s) = \sum_{k=0}^{\infty} \mathbb{P}_\pi\left(S_k = s | S_0 = s_0\right) \tag{384}$$

is the expected amount of time MDP has spent at state s. As a result, $\mu(s)$ is the expected percentage of time MDP has spent in state s.

One might be wondering if the normalization is well-defined, i.e. if $\sum_{s\in S} \eta(s) < \infty$, let's do some calculations assuming that the time horizon is T. Then

$$\sum_{s} \eta(s) = \sum_{s} \sum_{k=0}^{T-1} \mathbb{P}_\pi\left(S_k = s | S_0 = s_0\right) = \sum_{k=0}^{T-1} \sum_{s} \mathbb{P}_\pi\left(S_k = s | S_0 = s_0\right) = T < \infty \tag{385}$$

in episodic games. As an analogue, the general definition of $\eta(s)$ is just a discounted version.

This concept is also used in the approximation of value function where one forms the approximated value function as $\hat{v}(s,w)$ with weight w as parameter. The mean square error of value function approximation is given by

$$\overline{VE}(w) \overset{def}{=} \sum_{s} \mu(s) \left[v_\pi(s) - \hat{v}(s,w)\right]^2 \tag{386}$$

as a weighted average w.r.t. the on-policy distribution and one typically seeks to choose the optimal weight w to minimize this error.

REINFORCE

The policy gradient theorem gives natural rise to the algorithm REINFORCE. Since we want to maximize $J(\theta)$, from gradient ascent we shall do

$$\theta \leftarrow \theta + \alpha \nabla J(\theta) \tag{387}$$

where $\alpha > 0$ is a positive hyperparameter. By policy gradient theorem,

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_{a \in \mathscr{A}} \nabla \pi(a|s,\theta) \cdot q_\pi(s,a) \tag{388}$$

with the proportion constant to be absorbed into α, the update is provided as

$$\theta \leftarrow \theta + \alpha \sum_s \mu(s) \sum_{a \in \mathscr{A}} \nabla \pi(a|s,\theta) \cdot q_\pi(s,a) \tag{389}$$

Two problems remain in this update scheme. Firstly, μ is unknown in the model-free setting. Secondly, we don't want to maintain a tabular estimate for action value $q_\pi(s,a)$.

The first problem can be solved by noticing that μ has close connection to the invariant measure of the MDP under policy π. If $\gamma = 1$, Kolmogorov's cycle trick tells us

$$\mu(s) = \frac{\sum_{k=0}^{\infty} \mathbb{P}_\pi\left(S_k = s|S_0 = s_0\right)}{\sum_{s \in S} \sum_{k=0}^{\infty} \mathbb{P}_\pi\left(S_k = s|S_0 = s_0\right)} \tag{390}$$

is just an invariant measure of the MDP given some extra conditions (e.g. s_0 is recurrent).

Remark. *We circumvent the first problem by noticing that as long as we are following policy π, the empirical distribution of the state just provides us with the on-policy distribution.*

We calculate for the case where $\gamma = 1$,

$$\sum_s \mu(s) \sum_{a \in \mathscr{A}} \nabla \pi(a|s,\theta) \cdot q_\pi(s,a) = \mathbb{E}_\pi \sum_{a \in \mathscr{A}} \nabla \pi(a|S_t,\theta) \cdot q_\pi(S_t,a) \tag{391}$$

where the structure of expectation appears. To deal with the action value $q_\pi(S_t,a)$, notice that it is again an expectation $q_\pi(S_t, a) = \mathbb{E}_\pi(G_t|S_t, A_t = a)$. With a logarithm trick applied, we see that

$$\begin{aligned}
\mathbb{E}_\pi \sum_{a\in\mathscr{A}} \nabla\pi(a|S_t,\theta)\cdot q_\pi(S_t,a) &= \mathbb{E}_\pi \sum_{a\in\mathscr{A}} \pi(a|S_t,\theta)\frac{\nabla\pi(a|S_t,\theta)}{\pi(a|S_t,\theta)}\cdot q_\pi(S_t,a) && (392)\\
&= \mathbb{E}_\pi \mathbb{E}_{A_t\sim\pi(\cdot|S_t,\theta)}\left(\frac{\nabla\pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}\cdot q_\pi(S_t,A_t)\middle|S_t\right) && (393)\\
&= \mathbb{E}_\pi\left(\frac{\nabla\pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}\cdot q_\pi(S_t,A_t)\right) && (394)\\
&= \mathbb{E}_\pi\left[\nabla\log\pi(A_t|S_t,\theta)\cdot\mathbb{E}_\pi(G_t|S_t,A_t)\right] && (395)\\
&= \mathbb{E}_\pi\left[G_t\cdot\nabla\log\pi(A_t|S_t,\theta)\right] && (396)
\end{aligned}$$

The gradient ascent update of θ is given by

$$\theta \leftarrow \theta + \alpha\cdot\mathbb{E}_\pi\left[G_t\cdot\nabla\log\pi(A_t|S_t,\theta)\right] \tag{397}$$

although we do not know the value of the exact gradient, it's easy to sample from a distribution whose expectation is equal to the exact gradient. By adopting the stochastic gradient ascent idea presented in bandit gradient algorithms, we get the REINFORCE update that

$$\theta \leftarrow \theta + \alpha\cdot G_t\cdot\nabla\log\pi(A_t|S_t,\theta) \tag{398}$$

when $\gamma = 1$.

Remark. *For the case where $\gamma\in(0,1)$, the problem we are facing at stage $k+1$ is always equivalent to the same problem we are facing at stage k with all rewards discounted by γ. This is a consequence that we are in the infinite-horizon setting with Markov property to hold. As a result, we would perform the same update at stage $k+1$ with the only difference as to add another discount factor γ, i.e.*

$$\theta \leftarrow \theta + \alpha\cdot\gamma^t G_t\cdot\nabla\log\pi(A_t|S_t,\theta) \tag{399}$$

is the update performed at time t within each episode.

The complete REINFORCE algorithm is provided in Alg. 14.

Remark. *The policy can be parametrized in a lot of different ways. We list some of the most useful ones here for reference. When one has a finite action space, the direct parametrization is*

$$\pi(a|s,\theta) = h(s,a),\ \ h(s,a)\geq 0, \sum_a h(s,a) = 1 \tag{400}$$

assigning a value to each state-action pair. The softmax policy uses the same idea as that from the bandit gradient

Algorithm 14 REINFORCE

```
Input: Hyperparameter $\alpha > 0$
 1: Initial parameter $\theta$ for parametrized policy $\pi(a|s,\theta)$
 2: repeat
 3:     for $t = 0, 1, 2, ..., T$ do
 4:         Select action $A_t \sim \pi(\cdot|S_t, \theta)$
 5:         Take action $A_t$, observe reward $R_{t+1}$ and state transition to $S_{t+1}$
 6:     end for
 7:     Construct returns $\{G_t\}$ for the episode above
 8:     for $t = 0, 1, 2, ..., T$ do
 9:         $\theta = \theta + \alpha \cdot \gamma^t G_t \cdot \nabla \log \pi(A_t|S_t, \theta)$
10:     end for
11: until Enough iterations are done
Output: $\theta$ such that $\pi(a|s,\theta)$ is the estimate for $\pi^*$
```

algorithm, i.e.

$$\pi(a|s,\theta) = \frac{e^{h(s,a,\theta)}}{\sum_{a'\in\mathscr{A}} e^{h(s,a',\theta)}} \tag{401}$$

while such policy can also be parametrized using neural network with a softmax output layer that

$$\pi(a|s,\theta) = \frac{e^{\phi_\theta(s,a)}}{\sum_{a'\in\mathscr{A}} e^{\phi_\theta(s,a')}} \tag{402}$$

where ϕ denotes the mapping given by an NN.

Take the softmax policy as an example,

$$\pi(a|s,\theta) = \frac{e^{h(s,a,\theta)}}{\sum_{a'\in\mathscr{A}} e^{h(s,a',\theta)}} \tag{403}$$

the preference can be chosen as a linear function in θ, i.e.

$$h(s,a,\theta) = \theta^T x(s,a) \tag{404}$$

under such setting,

$$\nabla \log \pi(a|s,\theta) = x(s,a) - \sum_{a'\in\mathscr{A}} \pi(a'|s,\theta) \cdot x(s,a') \tag{405}$$

can be explicitly calculated. For neural softmax policy, $\nabla \log \pi(a|s,\theta)$ can be calculated by back propagation through auto-differentiation which is supported by most deep learning packages like Pytorch.

Notice that one of the greatest advantages of REINFORCE is that it also works for **large action space**, in which case all the parametrization of the policy above no longer works. This is because $\pi(\cdot|s)$ is now a continuous probability distribution. In this case, the parametrization is typically organized as the density of a continuous probability distribution. For example,

$$\pi(a|s,\theta) = \frac{1}{\sqrt{2\pi}\sigma_\theta(s)} e^{-\frac{(a-\mu_\theta(s))^2}{2\sigma_\theta(s)^2}} \tag{406}$$

where $\mu_\theta, \sigma_\theta$ are functions in the state variable parametrized w.r.t. θ. They can be chosen as functions linear in θ, NN with parameter θ etc. As an example, we consider

$$\mu_\theta(s) = \theta_\mu^T x_\mu(s), \sigma_\theta(s) = e^{\theta_\sigma^T x_\sigma(s)} \tag{407}$$

where x_μ, x_σ are features extracted based on state s. Easy calculation tells us that

$$\nabla_{\theta_\mu} \log \pi(a|s,\theta) = \frac{(a-\mu_\theta(s))}{\sigma_\theta^2(s)} \cdot x_\mu(s) \tag{408}$$

$$\nabla_{\theta_\sigma} \log \pi(a|s,\theta) = \left(\frac{[a-\mu_\theta(s)]^2}{\sigma_\theta^2(s)} - 1\right) x_\sigma(s) \tag{409}$$

so we can perform REINFORCE update.

REINFORCE with Baseline

The general suffering of REINFORCE is the high variance and the sample inefficiency. To resolve the high variance issue, similar to what we have done with gradient bandit methods, we add a baseline to the REINFORCE update. It's naturally expected that such baseline does not interfere with the REINFORCE update being an instance of stochastic gradient ascent but greatly accelerates the convergence of the algorithm.

From policy gradient theorem, we observe

$$\sum_s \mu(s) \sum_{a\in\mathscr{A}} \nabla\pi(a|s,\theta)\cdot q_\pi(s,a) = \sum_s \mu(s) \sum_{a\in\mathscr{A}} \nabla\pi(a|s,\theta)\cdot [q_\pi(s,a)-b(s)] \tag{410}$$

this is because

$$\sum_s \mu(s) \sum_{a\in\mathscr{A}} \nabla\pi(a|s,\theta)\cdot b(s) = \sum_s \mu(s)\cdot b(s)\cdot\nabla 1 = 0 \tag{411}$$

thus we have the REINFORCE update with baseline b that

$$\theta \leftarrow \theta + \alpha\cdot\gamma^t[G_t - b(S_t)]\cdot\nabla\log\pi(A_t|S_t,\theta) \tag{412}$$

notice that such baseline can only depend on the current state but not the current action as shown above.

One natural choice of the baseline is the state value function, i.e. $b(S_t)=\hat{v}(S_t,w)$ for a parametrized estimate $\hat{v}$ for v_π with parameter w. In the algorithm, we shall update both parameters θ and w based on our experience. The update of w aims at minimizing the weighted mean square error as previously defined

$$\min_w \overline{\text{VE}}(w) = \sum_s \mu(s)\left[v_\pi(s)-\hat{v}(s,w)\right]^2 \tag{413}$$

using gradient descent. Compute the gradient w.r.t. w that

$$\nabla_w\overline{\text{VE}}(w) = \sum_s \mu(s)\nabla\left[v_\pi(s)-\hat{v}(s,w)\right]^2 \tag{414}$$

$$= 2\sum_s \mu(s)\left[\hat{v}(s,w)-v_\pi(s)\right]\cdot\nabla_w\hat{v}(s,w) \tag{415}$$

$$\propto \mathbb{E}_\pi\left[(\hat{v}(S_t,w)-v_\pi(S_t))\cdot\nabla_w\hat{v}(S_t,w)\right] \tag{416}$$

here we adopt the same assumption on initial distribution as that in REINFORCE, then

$$\nabla_w\overline{\text{VE}}(w) \propto \mathbb{E}_\pi\left[(\hat{v}(S_t,w)-\mathbb{E}_\pi(G_t|S_t))\cdot\nabla_w\hat{v}(S_t,w)\right] \tag{417}$$

$$= \mathbb{E}_\pi\left[(\hat{v}(S_t,w)-G_t)\cdot\nabla_w\hat{v}(S_t,w)\right] \tag{418}$$

the gradient once again has the structure of the expectation so the stochastic gradient descent update of w for state value approximation is

$$w \leftarrow w + \alpha \left[G_t - \hat{v}(S_t, w)\right] \cdot \nabla_w \hat{v}(S_t, w) \tag{419}$$

where $\alpha > 0$ is the step size parameter.

The complete details of REINFORCE with baseline is provided below in Alg. 15.

Algorithm 15 REINFORCE with baseline

```
Input: Hyperparameter $\alpha^\theta, \alpha^w > 0$
 1: Initial parameter $\theta$ for parametrized policy $\pi(a|s, \theta)$
 2: Initial parameter $w$ for parametrized state value approximate $\hat{v}(s, w)$
 3: repeat
 4:     for $t = 0, 1, 2, ..., T$ do
 5:         Select action $A_t \sim \pi(\cdot|S_t, \theta)$
 6:         Take action $A_t$, observe reward $R_{t+1}$ and state transition to $S_{t+1}$
 7:     end for
 8:     Construct returns $\{G_t\}$ for the episode above
 9:     for $t = 0, 1, 2, ..., T$ do
10:         $w = w + \alpha^w \left[G_t - \hat{v}(S_t, w)\right] \cdot \nabla_w \hat{v}(S_t, w)$
11:         $\theta = \theta + \alpha^\theta \cdot \gamma^t [G_t - \hat{v}(S_t, w)] \cdot \nabla_\theta \log \pi(A_t|S_t, \theta)$
12:     end for
13: until Enough iterations are done
Output: $\theta$ such that $\pi(a|s, \theta)$ is the estimate for $\pi^*$,$w$ such that $\hat{v}(s, w)$ is the estimate for $v^*$
```

REINFORCE with baseline has a much smaller variance than the vanilla REINFORCE and notice that it's a model-free algorithm but it's offline. This is because the algorithm learns after an episode has been generated.

Remark. *The parametrization of $\hat{v}(s, w)$ is basically similar to that of the policy. One can set it as linear function in w, set up a NN with parameter w etc. An empirical guideline for setting two learning rates α^θ, α^w is that $\alpha^w = \frac{0.1}{\mathbb{E}_{S_t \sim \mu} ||\nabla \hat{v}(S_t, w)||^2}$ but it's actually mostly problem-dependent.*

Actor-Critic Methods

Recall that we turn the offline Monte Carlo methods into the online Q-learning methods by bootstrapping, i.e. integrating temporal difference error into the learning procedure. In the field of RL, we call the policy as actor and value function as critic since policy generates action (acts) and value function measures how good a policy is (criticize). In previous context, we have considered value-based (critic-only) method like Q-learning and policy-based (actor-only) method like REINFORCE. However, it's clear that both kinds of methods have their own pros and cons. Value-based method use TD error and has low variance, but cannot scale to large action space. On the other hand, policy-based method can deal with large action space but suffer from high variance and sample inefficiency. Well, why don't we combine both parts to set up actor-critic method that share the advantage of both kinds of methods?

Starting from building up the temporal difference error, recall that

$$v_\pi(s) = \mathbb{E}_\pi(R_{t+1} + \gamma \cdot v_\pi(S_{t+1})|S_t = s) \tag{420}$$

if $\hat{v}(s, w)$ still denotes a parametrized approximation of $v_\pi(s)$, we expect to see the TD error being formulated as

$$R_{t+1} + \gamma \cdot \hat{v}(S_{t+1}, w) - \hat{v}(S_t, w) \tag{421}$$

As a result, the simplest actor-critic method is online thanks to the bootstrapping idea, i.e. replacing return G_t with $R_{t+1} + \gamma \cdot \hat{v}(S_{t+1}, w)$, the best estimate we have so far. The complete details of the algorithm are provided in Alg. 16.

Algorithm 16 One-step actor-critic

```
Input: Hyperparameter $\alpha^\theta, \alpha^w > 0$
 1: Initial parameter $\theta$ for parametrized policy $\pi(a|s, \theta)$
 2: Initial parameter $w$ for parametrized state value approximate $\hat{v}(s, w)$
 3: repeat
 4:   for $t = 0, 1, 2, ..., T$ do
 5:     Select action $A_t \sim \pi(\cdot|S_t, \theta)$
 6:     Take action $A_t$, observe reward $R_{t+1}$ and state transition to $S_{t+1}$
 7:     TD error $\delta_t = R_{t+1} + \gamma \cdot \hat{v}(S_{t+1}, w) - \hat{v}(S_t, w)$
 8:     $w = w + \alpha^w \delta_t \nabla_w \hat{v}(S_t, w)$
 9:     $\theta = \theta + \alpha^\theta \gamma^t \delta_t \nabla_\theta \log \pi(A_t|S_t, \theta)$
10:   end for
11: until Enough iterations are done
Output: $\theta$ such that $\pi(a|s, \theta)$ is the estimate for $\pi^*$,$w$ such that $\hat{v}(s, w)$ is the estimate for $v^*$
```

Notice that actor-critic methods refer to those that estimate the policy and the value at the same time., in which sense the REINFORCE with baseline is also an actor-critic method. The difference is that Alg. 16 is online but REINFORCE with baseline is not.

Natural Policy Gradient (NPG)

NPG was first proposed in ***A Natural Policy Gradient by Sham Kakade*** and improvements follow in the subsequent work. NPG is a method that is inspired by information theory and geometry, but has an unbelievable simple form and is effective in practice.

Let's start with the motivation of NPG that in PG methods we always parametrize the policy $\pi(a|s,\theta)$ with parameter θ, in other words, we have a family of probability measure induced by parameter θ if s is seen as a fixed state. In the context above, we have been focusing on the gradient ascent in the direction $\nabla J(\theta)$ to maximize $J(\theta)$, but let's first think about how do we argue that along the gradient direction we have the steepest ascent. From scratch, we perturb θ a little bit to get $\theta+h$ and we hope to maximize $J(\theta+h)$ for h to be infinitesimal, i.e. under the constraint $h^Th=\varepsilon$ for some fixed small $\varepsilon>0$. As a result, we have the following optimization problem

$$\begin{cases}\max_h J(\theta+h)\\ s.t.\ h^Th=\varepsilon\end{cases}\tag{422}$$

we use the first-order approximation $J(\theta+h)=J(\theta)+[\nabla J(\theta)]^Th+o(|h|)$ to replace the objective function assuming that $\nabla J(\theta)\neq 0$ so the first-order term dominates the remainder. This optimization problem turns into

$$\begin{cases}\max_h[\nabla J(\theta)]^Th\\ s.t.\ h^Th=\varepsilon\end{cases}\tag{423}$$

Lagrange multiplier method tells us immediately that h shall have the same direction as $\nabla J(\theta)$. The derivation is reasonable and is the foundation of steepest ascent/descent but in our case things are a little bit different since what we are parametrizing is a probability measure, and the space of measure is not flat. It turns out that we can equip the space of measure with the Riemannian metric induced by the metric tensor G, i.e. $||h||_G=\sqrt{h^TG(\theta)h}$ for infinitesimal h at θ where $G(\theta)$ is an SPD matrix depending on θ (describing the local curvature at θ). In this case, the problem

$$\begin{cases}\max_h[\nabla J(\theta)]^Th\\ s.t.\ h^TG(\theta)h=\varepsilon\end{cases}\tag{424}$$

has the optimal h to be in the same direction as $[G(\theta)]^{-1}\nabla J(\theta)$. This example tells us that the steepest ascend direction is no longer along the gradient in a curved space.

What G shall we choose then? It turns out that an invariant metric on the space of the parameters that parametrizes a family of probability distribution is derived by choosing $G(\theta)$ as the Fisher information matrix where the invariant property means that the metric does not depend on the parametrization but only depends on the manifold itself. As an example, let's consider the family of distribution $\mathscr{M}=\{N(\theta,1):\theta>0\}$ as a manifold in the space of probability measures. Naturally we parametrize the manifold with θ but of course we can also parametrize the manifold with η that $\mathscr{M}=\{N(e^\eta,1):\eta\in\mathbb{R}\}$. Under the Euclidean distance on $\mathbb{R}$, $\theta_1=1$ and $\eta_1=0$ corresponds to the same measure and $\theta_2 = 2$ and $\eta_2 = \log 2$ corresponds to the same measure, but $\eta_2 - \eta_1 \neq \theta_2 - \theta_1$ since this manifold is not flat. However, if we calculate Fisher information matrix $G(\theta) = 1, G(\eta) = e^{2\eta}$, then $dist(\theta_1, \theta_2) = 2 - 1 = 1$ while $dist(\eta_1, \eta_2) = \int_{\eta_1}^{\eta_2} ||d\eta||_G = \int_{\eta_1}^{\eta_2} \sqrt{G(\eta)}\, d\eta = 1$ so $dist(\eta_1, \eta_2) = dist(\theta_1, \theta_2)$ under this Fisher information metric. The parametrization no longer causes difference in the distance by taking FI as the metric tensor.

Remark. *On the space of probability measures parametrized by θ, the only invariant Riemannian metric under sufficient statistics is the one taking $G(\theta)$ as the Fisher information matrix, known as Chentsov's theorem. That's why when thinking about a family of parametrized probability measures, the Fisher information is always important to consider. The same motivation gives rise to the Jeffrey's prior $p(\theta) \propto \sqrt{I(\theta)}$ in the Bayesian setting, well-known for its invariance property.*

At this point, we realize that the NPG update is given by

$$\theta \leftarrow \theta + \alpha[F(\theta)]^{-1}\nabla J(\theta) \tag{425}$$

where $F(\theta)$ is the Fisher information matrix. However, our policy is given by $\pi(a|s,\theta)$ not only depends on θ but also depends on s. The way to deal with this situation is to first consider the FI matrix given state s that

$$[F_s(\theta)]_{ij} = \mathbb{E}_{a\sim\pi(\cdot|s,\theta)}\left(\frac{\partial \log \pi(a|s,\theta)}{\partial \theta_i}\frac{\partial \log \pi(a|s,\theta)}{\partial \theta_j}\right) \tag{426}$$

and then take an expectation w.r.t. $s \sim \mu^\pi$, the on-policy distribution corresponding to policy π (this μ^π also depends on θ, but we omit the dependence when there's no confusion). As a result,

$$F(\theta) = \mathbb{E}_{s\sim\mu^\pi} F_s(\theta) \tag{427}$$

eliminates the dependence on state s.

Remark. *For another interpretation of NPG from the information theory perspective, we claim that NPG is the process of approximatedly repeatedly solving an optimization problem under the minimum divergence principle*

$$\theta_{t+1} = \arg\max_\theta J(\theta) \ s.t. \ \mathbb{E}_{s\sim\mu^{\pi(\theta_t)}} D_{KL}(\pi(\cdot|s,\theta_t)||\pi(\cdot|s,\theta)) \leq \delta \tag{428}$$

meaning that we wish to find the best θ to maximize $J(\theta)$ under the constraint that the average difference in the policy shall be as small as possible. The correspondence is immediate if one approximates the objective function with the first-order expansion and the constraint with the second-order expansion at θ_t. Obviously, the objective function can be replaced with $J(\theta_t) + [\nabla J(\theta_t)]^T(\theta - \theta_t)$.

What about the second-order expansion of the constraint? It seems hard to differentiate the KL divergence w.r.t. the measure but there's actually a trick to compute the expansion in an easy way. Notice that $D_{KL}(\pi(\cdot|s,\theta_t)||\pi(\cdot|s,\theta)) = 0$ if $\theta = \theta_t$ so $\nabla_\theta D_{KL}(\pi(\cdot|s,\theta_t)||\pi(\cdot|s,\theta))\Big|_{\theta=\theta_t} = 0$ since it attains the minimum of the KL divergence. As a result, this KL divergence term has its value and the first-order term to be zero at θ_t so the second-order term is actually *the dominating term. We replace the KL divergence with*

$$\frac{1}{2}(\theta-\theta_t)^T\left(\nabla^2_\theta D_{KL}(\pi(\cdot|s,\theta_t)||\pi(\cdot|s,\theta))\Big|_{\theta=\theta_t}\right)(\theta-\theta_t) \tag{429}$$

the Hessian matrix of the KL divergence is

$$\nabla^2_\theta D_{KL}(\pi(\cdot|s,\theta_t)||\pi(\cdot|s,\theta)) = \nabla^2_\theta\int\log\frac{\pi(x|s,\theta_t)}{\pi(x|s,\theta)}\cdot\pi(x|s,\theta_t)\,dx \tag{430}$$

$$= -\int\nabla^2_\theta\log\pi(x|s,\theta)\cdot\pi(x|s,\theta_t)\,dx \tag{431}$$

$$= -\mathbb{E}_{X\sim\pi(\cdot|s,\theta_t)}\nabla^2_\theta\log\pi(X|s,\theta) \tag{432}$$

matches the definition of the Fisher information $F_s(\theta_t)$ when this term is evaluated at $\theta=\theta_t$. Taking expectation w.r.t. $s\sim\mu^\pi$ on both sides, we get $\mathbb{E}_{s\sim\mu^\pi}\nabla^2_\theta D_{KL}(\pi(\cdot|s,\theta_t)||\pi(\cdot|s,\theta)) = F(\theta_t)$.

As a result, the approximated optimization problem looks like

$$\theta_{t+1} = \arg\max_\theta[\nabla J(\theta_t)]^T(\theta-\theta_t)\ s.t.\ \frac{1}{2}(\theta-\theta_t)^TF(\theta_t)(\theta-\theta_t)\le\delta \tag{433}$$

a simple calculation using Lagrange multiplier gives $\theta_{t+1}=\theta_t+\alpha[F(\theta_t)]^{-1}\nabla J(\theta_t)$, the update rule for NPG.

With those interpretation in mind, we just want to inform the readers that the essential problem lies in the fact that the space of measure is not flat, which is different from the Euclidean space, and that NPG arises naturally both in terms of geometry and in terms of information theory.

Finally, we put up an algorithm based on NPG theory stated above. By adopting the same idea as that in REINFORCE, we use stochastic gradient ascend to sample a direction from a distribution whose mean is $[F(\theta)]^{-1}\nabla J(\theta)$. Therefore it's necessary to investigate the representation of $[F(\theta)]^{-1}\nabla J(\theta)$ as an expectation. The following lemma helps us with this.

Lemma 10. *Consider*

$$w^*\in\arg\min_w\mathbb{E}_{s\sim\mu^\pi,a\sim\pi(\cdot|s,\theta)}[w^T\nabla_\theta\log\pi(a|s,\theta)-A_{\pi(\cdot|\cdot,\theta)}(s,a)]^2 \tag{434}$$

where $A_{\pi(\cdot|\cdot,\theta)}(s,a)$ is an estimate for the advantage function $A(s,a)=q(s,a)-v(s)$ under policy $\pi(\cdot|\cdot,\theta)$, then

$$w^*\propto[F(\theta)]^{-1}\nabla J(\theta) \tag{435}$$

Proof. Differentiate w.r.t. w and set the gradient to zero, we get

$$\mathbb{E}[\nabla_\theta\log\pi(a|s,\theta)\nabla^T_\theta\log\pi(a|s,\theta)]w^* = \mathbb{E}[A_{\pi(\cdot|\cdot,\theta)}(s,a)\nabla_\theta\log\pi(a|s,\theta)] \tag{436}$$

from the definition of Fisher information, $\mathbb{E}[\nabla_\theta\log\pi(a|s,\theta)\nabla^T_\theta\log\pi(a|s,\theta)] = F(\theta)$ so it suffices to calculate the RHS.

Recall that in REINFORCE with baseline, the baseline is taken as the estimate for state value function, which explains the appearance of the advantage function here. From policy gradient theorem with baseline taken as state value function,

$$\nabla_\theta J(\theta) = \sum_s \mu(s) \sum_{a\in\mathscr{A}} \nabla\pi(a|s,\theta)\cdot[q_\pi(s,a) - v_\pi(s)] \tag{437}$$

$$= \mathbb{E}_{s\sim\mu^\pi, a\sim\pi(\cdot|s,\theta)} \nabla_\theta \log\pi(a|s,\theta)\cdot[q(s,a) - v(s)] \tag{438}$$

proves the conclusion.

□

With the help of the lemma above, we only need to update w and take such w as the ascend direction of θ. Luckily, such w is the minimizer of a certain expectation. By adopting the idea of stochastic gradient descend, the update of w can be carried out through

$$w \leftarrow w - \alpha^w[w^T\nabla_\theta \log\pi(a|s,\theta) - A_{\pi(\cdot|\cdot,\theta)}(s,a)]\cdot\nabla_\theta\log\pi(a|s,\theta) \tag{439}$$

and the update of θ is simply

$$\theta \leftarrow \theta + \alpha^\theta w \tag{440}$$

The complete details of this sample-based NPG is provided in Alg. 17.

Algorithm 17 Sample-based NPG

```
Input: Hyperparameter $\alpha^\theta, \alpha^w > 0$
 1: Initial parameter $\theta$ for parametrized policy $\pi(a|s,\theta)$
 2: repeat
 3:   for $t = 0, 1, 2, ..., T$ do
 4:     Initialize $w$
 5:     repeat
 6:       Generate samples of state $s$ and action $a \sim \pi(\cdot|s,\theta)$
 7:       Form advantage estimate to get $A_{\pi(\cdot|\cdot,\theta)}(s,a)$
 8:       Update $w = w - \alpha^w[w^T\nabla_\theta \log\pi(a|s,\theta) - A_{\pi(\cdot|\cdot,\theta)}(s,a)]\cdot\nabla_\theta\log\pi(a|s,\theta)$
 9:     until $w$ converges
10:     $\theta = \theta + \alpha^\theta w$
11:     Select action $A_t \sim \pi(\cdot|S_t,\theta)$
12:     Take action $A_t$, observe reward $R_{t+1}$ and state transition to $S_{t+1}$
13:   end for
14: until Enough iterations are done
Output: $\theta$ such that $\pi(a|s,\theta)$ is the estimate for $\pi^*$
```

This algorithm is model-free, on-policy and online.

Convergence of PG and NPG

The key to the convergence theory of PG and NPG method is the performance difference lemma (PDL).

Lemma 11 (PDL)**.** *For any two policy π, π', let μ^π denote the on-policy distribution of MDP following policy π, then*

$$J(\pi) - J(\pi') = \frac{1}{1-\gamma}\mathbb{E}_{s\sim\mu^\pi, a\sim\pi(\cdot|s)} A_{\pi'}(s,a) \tag{441}$$

Proof. Clearly, if we condition on $S_0 = s_0$, then

$$\begin{aligned} J(\pi) - J(\pi') = v_\pi(s_0) - v_{\pi'}(s_0) &= \mathbb{E}_\pi \sum_{t=0}^\infty \gamma^t R_{t+1} - v_{\pi'}(s_0) &(442)\\ &= \mathbb{E}_\pi \left[\sum_{t=0}^\infty \gamma^t (R_{t+1} + \gamma v_{\pi'}(S_{t+1}) - v_{\pi'}(S_t))\right] &(443)\\ &= \mathbb{E}_\pi \left[\sum_{t=0}^\infty \gamma^t (\mathbb{E}_{S_{t+1}\sim p(\cdot|S_t,A_t)}[R_{t+1} + \gamma v_{\pi'}(S_{t+1})] - v_{\pi'}(S_t))\right] &(444)\\ &= \mathbb{E}_\pi \left[\sum_{t=0}^\infty \gamma^t (q_{\pi'}(S_t,A_t) - v_{\pi'}(S_t))\right] = \mathbb{E}_\pi \left[\sum_{t=0}^\infty \gamma^t A_{\pi'}(S_t,A_t)\right] &(445) \end{aligned}$$

recall the definition $\mu^\pi(s) = (1-\gamma)\mathbb{E}_\pi \sum_{t=0}^\infty \gamma^t \mathbb{I}_{S_t=s}$, the RHS of PDL gives

$$\begin{aligned} \frac{1}{1-\gamma}\sum_s \mu^\pi(s)\sum_a \pi(a|s)A_{\pi'}(s,a) &= \mathbb{E}_\pi \sum_{t=0}^\infty \gamma^t \sum_s \mathbb{I}_{S_t=s}\sum_a \pi(a|S_t)A_{\pi'}(S_t,a) &(446)\\ &= \mathbb{E}_\pi \sum_{t=0}^\infty \gamma^t \sum_a \pi(a|S_t)A_{\pi'}(S_t,a) &(447)\\ &= \mathbb{E}_\pi \sum_{t=0}^\infty \gamma^t \mathbb{E}_{A_t\sim\pi(\cdot|S_t)} A_{\pi'}(S_t,A_t) &(448) \end{aligned}$$

concludes the proof. □

The performance difference lemma quantifies the improvement of the policy in a single PG update. To understand PDL, we raise a simple example here. Consider A_{π^*}, the advantage function for the optimal policy, it's clear by the compact form of BOE that $\forall s, \max_a A_{\pi^*}(s,a) = 0$. As a result, by PDL, for any policy π, $J(\pi) - J(\pi^*) \le 0$ proves that the optimal policy has no room for improvement. PDL will be an important tool to use when it comes to building TRPO and PPO algorithm.

To derive the global convergence of PG and NPG, the following gradient mapping domination plays an important role.

Lemma 12 (Gradient Mapping Domination)**.** *For on-policy distribution μ^{π^*}, μ^π,*

$$J(\pi^*) - J(\pi) \leq \left\| \frac{\mu^{\pi^*}}{\mu^\pi} \right\|_\infty \max_{\tilde{\pi}} < \tilde{\pi} - \pi, \nabla J(\pi) > \tag{449}$$

Proof. By PDL

$$J(\pi^*) - J(\pi) = \frac{1}{1-\gamma} \mathbb{E}_{s\sim\mu^{\pi^*}, a\sim\pi^*(\cdot|s)} A_\pi(s,a) \tag{450}$$

$$= \frac{1}{1-\gamma} \sum_{s,a} \mu^{\pi^*}(s)\pi^*(a|s)A_\pi(s,a) \tag{451}$$

$$= \frac{1}{1-\gamma} \sum_{s,a} \frac{\mu^{\pi^*}(s)}{\mu^\pi(s)} \mu^\pi(s)\pi^*(a|s)A_\pi(s,a) \tag{452}$$

conducting a change of measure from μ^{π^*} to μ^π, we see that

$$\leq \frac{1}{1-\gamma} \left\| \frac{\mu^{\pi^*}}{\mu^\pi} \right\|_\infty \max_{\tilde{\pi}} \sum_{s,a} \mu^\pi(s)\tilde{\pi}(a|s)A_\pi(s,a) \tag{453}$$

notice that $\sum_{s,a} \mu^\pi(s)\pi(a|s)A_\pi(s,a) = \sum_{s,a} \mu^\pi(s)\pi(a|s)q_\pi(s,a) - \sum_s \mu^\pi(s)v_\pi(s) = 0$, this tricks allows us to proceed

$$= \frac{1}{1-\gamma} \left\| \frac{\mu^{\pi^*}}{\mu^\pi} \right\|_\infty \max_{\tilde{\pi}} \sum_{s,a} \mu^\pi(s)[\tilde{\pi}(a|s) - \pi(a|s)]A_\pi(s,a) \tag{454}$$

use again the trick that $\sum_{s,a} \mu^\pi(s)[\tilde{\pi}(a|s) - \pi(a|s)]v_\pi(s) = 0$, we deduce

$$J(\pi^*) - J(\pi) \leq \frac{1}{1-\gamma} \left\| \frac{\mu^{\pi^*}}{\mu^\pi} \right\|_\infty \max_{\tilde{\pi}} \sum_{s,a} \mu^\pi(s)[\tilde{\pi}(a|s) - \pi(a|s)]q_\pi(s,a) \tag{455}$$

which concludes the proof from the policy gradient theorem. □

Remark. $\left\| \frac{\mu^{\pi^*}}{\mu^\pi} \right\|_\infty$ *is the distribution mismatch coefficient, showing how different the on-policy distribution of π^* and π are, it's an indicator of the intrinsic difficulty of the exploration. Intuitively, we have proved that $J(\pi^*) - J(\pi) = O(||\nabla J(\pi)||)$. That's why it's called gradient domination and this implies J is a convex-like function on the policy space.*

Following those techniques, the convergence of tabular PG method with direct parametrization is proved in ***Optimality and approximation with policy gradient methods in markov decision process by Agarwal et. al.***. The bound is saying that $J(\pi^*) - \max_{t<T} J(\pi_t) \leq C\frac{\sqrt{|\mathscr{S}||\mathscr{A}|}}{\sqrt{T}} \left\| \frac{\mu^{\pi^*}}{\mu^\pi} \right\|_\infty$. Notice that the convergence gradually loses its effect when the state space and action space grows large enough and the convergence rate in the time horizon T is of order $T^{-\frac{1}{2}}$. The rate in terms of T is improved to order T^{-1} in the paper ***On the convergence rates of policy gradient methods by Lin Xiao***.

On the other hand, global convergence of tabular NPG with softmax parametrization is proved using the policy mirror descent idea. In other words, the update of parameter θ can be translated into the update of policy $\pi(a|s,\theta)$ in a concise way. It's shown by Agarwal in the same paper that $J(\pi^*) - J(\pi_T) \leq \frac{C}{T}$. Notice that this bound is free of $|\mathscr{S}|, |\mathscr{A}|$ so the convergence rate is not affected for large state/action space and the convergence rate in T is T^{-1}. Also, NPG is robust w.r.t. distribution mismatch.

Trust Region Policy Optimization (TRPO)

NPG, despite its nice theoretical bounds, suffers from some problems empirically. The stepsize hyperparameter is a big issue in RL. In supervised learning, taking a too big step size is not an issue since subsequent data corrects it. However, in RL once we have a too large step size, the policy will become terrible and we may never be able to recover (especially in on-policy learning, since the experience is generated based on the policy). On the other hand, too small stepsize results in similar policy and may not encourage exploration to an adequate extent.

In this sense, TRPO is a remedy for the stepsize selection problem. Recall from PDL that the performance gap of two policy π, π_{old} is

$$J(\pi) - J(\pi_{old}) \propto \mathbb{E}_{s\sim\mu^{\pi}, a\sim\pi(\cdot|s)} A_{\pi_{old}}(s,a) \tag{456}$$

if we want to achieve a great enough improvement at each step moving from the old policy π_{old} to the new policy π, we naturally think about maximizing the performance gap $J(\pi) - J(\pi_{old})$. However, this expectation contains the policy π which is still unknown to us when performing the update. The trouble arises in μ^{π}, the empirical state distribution following policy π and it's hard to conduct a change of measure between μ^{π} and $\mu^{\pi_{old}}$. To avoid this difficulty, we consider instead the **surrogate loss** defined as

$$L_{\pi_{old}}(\pi) = \mathbb{E}_{s\sim\mu^{\pi_{old}}, a\sim\pi(\cdot|s)} A_{\pi_{old}}(s,a) \tag{457}$$

directly replacing μ^{π} with $\mu^{\pi_{old}}$.

Remark. *One might be wondering if this operation causes any issue and why it's reasonable. Let's check that*

$$L_{\pi}(\pi) = \mathbb{E}_{s\sim\mu^{\pi}, a\sim\pi(\cdot|s)} A_{\pi}(s,a) = 0 = J(\pi) - J(\pi) \tag{458}$$

as we have proved it in the last section. On the other hand, if we denote $\pi(\theta)$ as a policy parametrized by parameter θ, then

$$\nabla_\theta L_{\pi(\theta_0)}(\pi(\theta)) = \mathbb{E}_{s\sim\mu^{\pi(\theta_0)}, a\sim\pi(\cdot|s,\theta)} \nabla_\theta \log\pi(a|s,\theta) A_{\pi(\theta_0)}(s,a) \tag{459}$$

so

$$\nabla_\theta L_{\pi(\theta_0)}(\pi(\theta))|_{\theta=\theta_0} = \mathbb{E}_{s\sim\mu^{\pi(\theta_0)}, a\sim\pi(\cdot|s,\theta)} \nabla_\theta \log\pi(a|s,\theta) A_{\pi(\theta_0)}(s,a) \propto \nabla_\theta J(\theta)|_{\theta=\theta_0} \tag{460}$$

from the policy gradient theorem (REINFORCE with baseline). As a result, ***with parametrized differentiable policy*** *π_{old}, π*, ***$J(\pi) - J(\pi_{old})$ and $L_{\pi_{old}}(\pi)$ matches each other up to the first order when $\pi \to \pi_{old}$***. *Notice that the "proportional to" constant absorbed in $J(\pi) - J(\pi_{old})$ and $\nabla_\theta J(\theta)|_{\theta=\theta_0}$ above are both $\frac{1}{1-\gamma}$, so actually*

$$\nabla_\theta L_{\pi(\theta_0)}(\pi(\theta))|_{\theta=\theta_0} = \nabla_\theta J(\theta)|_{\theta=\theta_0} \tag{461}$$

This argues why it's reasonable to consider the surrogate loss instead. Actually replacing the on-policy distribution *does not change all terms up to the first order approximation given that the new policy is close enough to the original one.*

Notice that there is still π remaining in the expectation in the expression of $L_{\pi_{old}}(\pi)$. Luckily, this dependence on π is not a big matter since it's easy to conduct a change of measure

$$L_{\pi_{old}}(\pi) = \mathbb{E}_{s\sim\mu^{\pi_{old}},a\sim\pi_{old}(\cdot|s)}\frac{\pi(a|s)}{\pi_{old}(a|s)}A_{\pi_{old}}(s,a) \tag{462}$$

we finally get our objective function which is easy to optimize.

As stated above, the stepsize in TRPO cannot be too large due to two reasons. The first reason is the fact that RL is not supervised, the second reason is that surrogate loss approximates the performance gap well only when two policy are close to each other. As a result, we put up a constraint that the new policy π shall not be too different from the old policy π_{old} for any state s, i.e. $\max_s D_{KL}(\pi(\cdot|s)||\pi_{old}(\cdot|s)) \leq \delta$ for a given δ. However, when we are in the situation of a large state space, it's impossible to impose this constraint. Instead, we form the constraint as

$$\mathbb{E}_{s\sim\mu^{\pi_{old}}}D_{KL}(\pi(\cdot|s)||\pi_{old}(\cdot|s)) \leq \delta \tag{463}$$

making use of the fact that the on-policy distribution is empirical, enabling us to calculate under simulation. This constraint is called the **trust region constraint**, meaning that we only trust the new policy that is close enough to the last policy.

As a result, TRPO is actually solving the optimization problem

$$\begin{cases} \max_\pi \mathbb{E}_{s\sim\mu^{\pi_{old}},a\sim\pi_{old}(\cdot|s)}\frac{\pi(a|s)}{\pi_{old}(a|s)}A_{\pi_{old}}(s,a) \\ s.t.\ \mathbb{E}_{s\sim\mu^{\pi_{old}}}D_{KL}(\pi(\cdot|s)||\pi_{old}(\cdot|s)) \leq \delta \end{cases} \tag{464}$$

this looks familiar to the interpretation of NPG we have provided in the previous context! So what's the connection between NPG and TRPO? Similar to NPG, TRPO deals with the objective function by using first-order approximation and the KL constraint by using second-order approximation to turn it into a Fisher information matrix. The difference lies in the following aspects: (i): TRPO conducts the NPG update with conjugate gradient method to avoid the explicit calculation of the inverse of the Fisher information matrix (ii): TRPO performs a backtracking line search to ensure that the KL constaint is satisfied (iii): TRPO performs a backtracking line search to ensure the monotonic improvement, i.e. the maximum of the objective is positive.

In short, the parameter update of NPG and TRPO are very similar but we derive the TRPO update from a different perspective and TRPO is equivalent to NPG plus conjugate gradient plus monotonic improvement plus line search. The devils are actually in the details of implementation, one is welcome to check ***Trust Region Policy Optimization by John Schulman et. al.*** for the full details. TPRO is model-free, on-policy and online.

Poximal Policy Optimization (PPO)

TRPO requires second-order approximation for the KL constraint, which costs a lot of time in the calculation either through calculating the expectation of the outer product of score function or by calculating the Hessian of the KL divergence. The trust region constraint is also infeasible to enforce for complex policy architecture (e.g. network with stochasticity or parameter sharing between actor and critic). In addition, a great number of first-order optimizers like SGD, Adam have been coded in modules like Pytorch which dies not fit with second-order optimization. Therefore, it really makes a difference if we can get rid of the second-order approximation while maintaining a similar or better performance compared to TRPO.

Of course, the most natural thing to do is to apply Lagrange multiplier method for the optimization problem in TRPO such that

$$\max_{\pi} \mathbb{E}_{s\sim\mu^{\pi_{old}},a\sim\pi_{old}(\cdot|s)}\left[\frac{\pi(a|s)}{\pi_{old}(a|s)}A_{\pi_{old}}(s,a)-\beta D_{KL}(\pi(\cdot|s)||\pi_{old}(\cdot|s))\right] \tag{465}$$

for the multiplier $\beta > 0$. This problem is now unconstrained but it turns out that fixing β as a hyperparameter does not perform well empirically. As a result, an adaptive scheme is put on this β, adding penalty if the expected KL divergence is large and vice versa, the algorithm is called PPO-penalty.

The PPO-clip, as one of the most popular RL methods in these days, adopts an unbelievably simple way to encode the trust region. Since the KL penalty term penalizes according to the difference between π and π_{old}, and the difference between those two measures is also reflected in the Radon-Nikodym derivative $\frac{\pi(a|s)}{\pi_{old}(a|s)}$, why don't we simply do some clipping operations on this Radon-Nikodym derivative? This idea leads to the optimization problem

$$\max_{\pi} \mathbb{E}_{s\sim\mu^{\pi_{old}},a\sim\pi_{old}(\cdot|s)} \min\left\{\frac{\pi(a|s)}{\pi_{old}(a|s)}A_{\pi_{old}}(s,a), \text{clip}\left(\frac{\pi(a|s)}{\pi_{old}(a|s)};1-\varepsilon,1+\varepsilon\right)A_{\pi_{old}}(s,a)\right\} \tag{466}$$

where the clipping function

$$\text{clip}(x;1-\varepsilon,1+\varepsilon) \tag{467}$$

takes value x if $x\in[1-\varepsilon,1+\varepsilon]$, takes value $1-\varepsilon$ if $x<1-\varepsilon$ and takes value $1+\varepsilon$ if $x>1+\varepsilon$. No matter if the advantage is positive or negative, the new policy does not benefit by going far away from the old policy.

PPO-clip encodes the trust region implicitly and can be solved using first-order optimizers, which is widely applicable and efficient compared to TRPO. It's also a model-free, on-policy, online method.

Remark. *In policy gradient method, exploration is typically encouraged through an entropy bonus $\lambda H(\pi(\cdot|s,\theta))$ that we hope to see policy farther away from being deterministic. When it comes to real implementation, PPO adds up the loss from three parts: surrogate (actor) loss, critic loss and entropy bonus.*

The advantage estimation follows the details in the paper ***High-dimensional continuous control using generalized advantage estimation by John Schulman et. al.*** using a critic network. One only has to set up a critic network to approximate the state value function and the generalized advantage estimation is automatically handled in torchrl, so we don't care that much about the details of the advantage estimation here. The only thing to keep in mind is that we do need a critic network so PPO is still essentially an actor-critic method. The sketch of PPO is provided in Alg. 18.

Algorithm 18 PPO

```
1: Initial parameter $\theta$ for parametrized policy $\pi(a|s,\theta)$ (actor network)
2: Initial parameter $w$ for the state value function (critic network)
3: **repeat**
4:     Stick to policy $\pi(a|s,\theta)$ to generate $N$ episodes, with each episode of length $T$
5:     Form advantage estimate at each time step based on the critic network
6:     Update $\theta, w$ with $K$ epochs and minibatch size $M \leq NT$. The loss includes the clipped PPO objective, the
       critic network loss and the entropy bonus.
7: **until** Enough iterations are done
**Output:** $\theta$ such that $\pi(a|s,\theta)$ is the estimate for $\pi^*$
```

Reinforcement Learning with Human Feedback (RLHF)

In this section we summarize the RLHF framework used to build ChatGPT in the paper ***Training language models to follow instructions with human feedback by Long Ouyang et. al.***.

Firstly, a language model (LM) is built through supervised training. Whenever a prompt is sampled, a labeler shows the LM the desired output in a nice way. As a result, the LM is equipped with some language functionality after this supervised learning step. This step can be understood as teaching where the answer is purely generated by human.

Afterwards, we input the same prompt to the LM and collect different outputs. A labeler ranks all possible outputs from best to worst and this data is used to train a reward model. The reward model should have the ability to distribute correct reward based on the behavior of the LM, with human preferences encoded.

Finally, the parameter tuning task is organized as an RL problem. The state space consists of the distribution of all possible input token sequence, the action space consists of all tokens the LM can use to generate outputs and the policy is actually the parameter of the LM we have trained in the first step. Most crucially, the reward model we have trained in the second step encourages the parameter of the LM to achieve a higher reward matching the human preference. As a new prompt is input into the LM, a set of LM parameter generates an output, which is evaluated by the reward model and the reward is used to update the parameter of the LM through PPO.

It's worth noticing that we have omitted tons of details in the description above. Problems such as which LM to use, how to design the reward model, how human shall provide feedback to the model, how to tune so many parameters in the LM, are crucial to the performance of the whole model and requires much effort on their own. However, this shows us the usefulness and generality of RL that almost all problems containing the meaning of "strategy" or "control" can be transformed into an RL problem. Once it's transformed, we only need to care about the design of the reward and which specific RL method to use.

Topics for Deep RL Methods and Frameworks

Double Deep Q-network(DDQN)

As pointed out in the paper ***Deep Reinforcement Learning with Double Q-learning by Hado van Hasselt et. al.***, maximization bias has always been an issue for Q-learning. There is numerical evidence that maximization bias exists for DQN. This discovery means that maximization bias is not caused by the usage of a specific estimation technique but generally exists in practice (since NN is a general and nice estimation tool). In the paper, the authors prove a lower bound for the estimation error and test numerically that even if the true action values are provided, maximization bias still exists. This overoptimism results in a huge performance downgrade due to the fact that Q-learning bootstraps so the maximization bias propagates.

Remark. *On shall not confuse the maximization bias with the optimism toward uncertainty. Method like UCB uses optimism toward uncertainty to encourage exploration in the action selection step. However, the maximization bias occurs in the learning step of Q-learning, when the interaction with the environment has already ended. Besides, there is typically no uncertainty in RL problem when the reward is not randomized. As a result, optimism towards uncertainty helps improve the learnt policy but maximization bias harms the learnt policy.*

Recall the double Q-learning we have introduced in previous contexts, combine it with DQN we get DDQN. As a result, we maintain a primary network Q_θ and a target network $Q_{\theta'}$ separately. The target network is only responsible for action evaluation and the primary network is updated and used to select greedy action. In detail, the TD target is computed as

$$y_t = R_{t+1} + \gamma \cdot Q_{\theta'}(S_{t+1}, \arg\max_a Q_\theta(S_{t+1}, a)) \tag{468}$$

and the loss is formulated as

$$(y_t - Q_\theta(S_t, A_t))^2 \tag{469}$$

with an average taken among all samples (from the replay buffer) in the minibatch. It's clear that only the parameter of the primary network is updated through stochastic gradient descent and the parameter of the target network is updated through Polyak averaging, i.e., $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$. Due to numerical stability issues, τ is typically taken as a very small positive number, e.g., $\tau = 0.01$. The goal is to gradually align the parameters of those two networks.

Numerical experiments show that DDQN shrinks the maximization bias to a large extent, which greatly improves the learned policy while the performance of DQN starts to drop as maximization bias comes into effect.

Prioritized Replay

For the replay buffer, **prioritized replay** has been proposed in ***Prioritized experience replay by Tom Schaul et. al.*** to replay more often those experiences from which one has more to learn. Each experience $(S_t, A_t, R_{t+1}, S_{t+1})$ in the replay buffer has priority p_t of being sampled, with p_t to be equal to the last encountered absolute TD error

$$p_t = \left| R_{t+1} + \gamma \max_a Q_{\theta'}(S_{t+1}, a) - Q_\theta(S_t, A_t) \right| \tag{470}$$

Here $Q_{\theta'}$ is the target network with parameter θ' to be an out-of-date version of θ. Clearly, a large TD error implies that the empirical state transition is "surprising", which implies that the agent can learn more from this experience. Since the agent can learn more from recent experiences, whenever a new experience $(S_t, A_t, R_{t+1}, S_{t+1})$ is pushed into the replay buffer, we set its priority $p_t = \max_{i<t} p_i$ to be maximal.

Remark. *Unlike the vanilla replay buffer, there does not exist two exactly same experiences in the prioritized replay buffer, i.e., each experience is recorded only for once. If the same experience is visited once more, its priority will be updated to be proportional to the power of the TD error. This mechanism encourages sampling new experience that has never been learnt by the agent.*

After setting up the priority of experiences, one samples from the buffer the i-th experience with priority p_i, with probability $\frac{p_i^\alpha}{\sum_k p_k^\alpha}$, where $\alpha \in [0, 1]$ is a hyperparameter indicating the force of prioritization. However, one might notice that this changes the empirical distribution of the MDP trajectory compared to uniform sampling (when $\alpha = 0$), which is equivalent to modifying the transition kernel of the MDP. This brings with bias and might cause the convergence to a policy different from the optimal policy. However, we can conduct a change of measure (importance sampling) to create weights $w_i \propto \left(\frac{p_i^\alpha}{\sum_k p_k^\alpha}\right)^{-\beta}$. The proportional constant in w_i is chosen such that the maximum $\max_i w_i$ is equal to one and $w_i \in (0, 1]$. This is to guarantee that the step-size in SGD is no larger than its original value (since large step-size causes problem in Deep RL).

When computing the SGD update of parameter θ in DQN w.r.t. the i-th experience (s_i, a_i, r_i, s_i'), the experience only contributes w_i proportion of its original contribution, i.e.,

$$\theta \leftarrow \theta - \alpha w_i \nabla_\theta \left[r_i + \gamma \max_a Q_{\theta'}(s_i', a) - Q_\theta(s_i, a_i) \right]^2 \tag{471}$$

compared to the original one in DQN with normal experience replay that

$$\theta \leftarrow \theta - \alpha \nabla_\theta \left[r_i + \gamma \max_a Q_{\theta'}(s_i', a) - Q_\theta(s_i, a_i) \right]^2 \tag{472}$$

When $\beta = 1$, the change of measure induced by $\{w_i\}$ fully compensates the distribution used in prioritized replay, and $\beta \in [0, 1]$ is again left as a hyperparameter. However, one has to make sure that at least at the end of the training β is set as one. This is crucial for the updates to be unbiased near the end of training (asymptotically) and for the convergence to the correct value function/policy to happen.

Dueling Network

When it comes to the model that approximates the action value function, we have been using a single NN that accepts state and outputs action value for each action in DQN. However, this leads to the learning of the value of a single state-action pair upon a single experience. In addition, in most of the "stable" states s in a game, the optimal action value $q^*(s, a)$ does not vary a lot across different actions. Consider a game that one is driving a car and wants to avoid colliding into other cars, if there are cars only very far away in the front, this state has almost the same q^* value across all different actions. Based on this observation, **dueling network** is proposed as a model to better approximate the action value function in ***Dueling Network Architectures for Deep Reinforcement Learning by Ziyu Wang et. al.***

The idea of dueling network is to tear the action value into two parts: the state value and the advantage

$$q^*(s, a) = v^*(s) + A^*(s, a) \tag{473}$$

For those "stable" states, q^* is expected to be close to v^*, leaving the advantage close to zero. For those critical states, the advantage is expected to make a difference across different actions. As a result, we shall use one network to approximate state value and another to approximate the advantage, adding them up recovers the action value. One philosophy behind this dueling network structure is that state value can be approximated much better than action value, since a sample for action value requires observing a state-action pair while a sample for state value only requires observing the state. At this point, it seems that we can organize the dueling network as

$$Q_{\theta,\alpha,\beta}(s, a) = V_{\theta,\beta}(s) + A_{\theta,\alpha}(s, a) \tag{474}$$

with θ, α, β as parameters. This causes the identifiability issue, however, and does not perform well in practice. That means we might observe some weird phenomenon, such as $Q = A, V = 0$ due to the fact that we only care about the sum of V and A but put no restriction on their own structures. This causes the identifiability issue, referring to the fact that we don't know how to distribute the action value among V and A.

To solve this issue, we only need to notice the intrinsic structure of the advantage function that $\max_a A^*(s, a) = 0$, due to the compact form of BOE. With this property in mind, we modify the rule of forward propagation that

$$Q_{\theta,\alpha,\beta}(s, a) = V_{\theta,\beta}(s) + \left[A_{\theta,\alpha}(s, a) - \max_{a'} A_{\theta,\alpha}(s, a')\right] \tag{475}$$

In this case, a value of Q uniquely identifies V and A. Instead, one can use the following forward propagation

$$Q_{\theta,\alpha,\beta}(s, a) = V_{\theta,\beta}(s) + \left[A_{\theta,\alpha}(s, a) - \frac{1}{|\mathscr{A}|}\sum_{a'} A_{\theta,\alpha}(s, a')\right] \tag{476}$$

at the cost of producing a bias (since $\frac{1}{|\mathscr{A}|}\sum_{a'} A_{\theta,\alpha}(s, a') \neq 0$), it increases the stability of optimization since a mean is much smoother than the maximum. Numerical experiments generate similar behavior for those two kinds of dueling networks. To conclude, the dueling network changes the way to approximate the action value function (the model) but makes no change to the algorithm of DQN. After preprocessing inputs, the same input is delivered to the state value network that produces one output $V(s)$ and the advantage network that produces $|\mathscr{A}|$ outputs $\{A(s,a)\}_{a\in\mathscr{A}}$. Those outputs are combined in an identifiable way (mentioned above, through maximum or mean) to produce $|\mathscr{A}|$ action values.

Multi-step Learning

The multi-step learning is found to help accelerate the learning process and it's an easy generalization to the vanilla Q-learning. If we denote the forward n-step return as

$$R_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k R_{t+k+1} \tag{477}$$

then the forward n-step TD error is formed as

$$R_t^{(n)} + \gamma^n \max_a Q(S_{t+n}, a) - Q(S_t, A_t) \tag{478}$$

the multi-step version of DQN is just computing

$$y_t = R_t^{(n)} + \gamma^n \max_a Q_{\theta'}(S_{t+n}, a) \tag{479}$$

and then minimize the loss

$$(y_t - Q_\theta(S_t, A_t))^2 \tag{480}$$

where $Q_{\theta'}$ is the target network. n is left as a hyperparameter to be tuned. Multi-step learning puts off the update of the action value function due to the fact that in most of the games rewards are sparse so one-step method might not be able to update the value function in an efficient way.

Distributional RL

Distributional RL aims at approximating the distribution of return instead of its expectation as value function. The work is presented in ***A Distributional Perspective on Reinforcement Learni by Marc G. Bellemare et. al.*** It constructs the distributional Bellman optimality operator to investigate the evolution of the distribution of return under the optimal policy.

To describe the main idea, we model the distribution of the return $Z(s,a)$ given current state s and current action a. It's clear that $\mathbb{E}_{\pi^*}Z(s,a) = q^*(s,a)$. Assume $Z(s,a)$ can take N_{atoms} different discrete values. Consider the vector $z \in \mathbb{R}^{N_{atoms}}$ such that

$$z_i = v_{min} + i\Delta z, \Delta z = \frac{v_{max} - v_{min}}{N_{atoms} - 1} \tag{481}$$

as the uniform partition of the interval $[v_{min}, v_{max}]$ into N_{atoms} endpoints. All those components of z are the canonical values the return $Z(s,a)$ can take. As a result, we only need to encode a probability distribution of $Z(s,a)$ with support $\{z_1, ..., z_{N_{atoms}}\}$.

$$\mathbb{P}\left(Z_\theta(s,a) = z_i\right) = p_i = [\text{softmax}(\theta(s,a))]_i \tag{482}$$

The idea is simple: encode a feature $\theta(s,a)$ of current state and action, and a softmax transformation provides a discrete probability distribution. We denote such $Z(s,a)$ as $Z_\theta(s,a)$ due to its dependence on θ. Inspired by BOE, the distribution of $Z_\theta(s,a)$ under the optimal policy π^* should have a similar characterization called the distributional BOE.

In order to introduce the characterization, we first have to model the transition operator under policy π as P^π, defined such that

$$P^\pi Z(s,a) \overset{d}{=} Z(S', A'), S' \sim p(\cdot|s,a), A' \sim \pi(\cdot|S') \tag{483}$$

where p is the MDP transition kernel. This P^π maps a probability distribution to another probability distribution. The distributional Bellman optimality operator under policy π is defined as

$$\mathscr{T}^\pi Z(s,a) \overset{d}{=} R(s,a) + \gamma P^\pi Z(s,a) \tag{484}$$

where scaling by γ and translating by $R(s,a)$ is defined in the sense of measure (transformation of r.v.). Notice that R has its own randomness and $P^\pi Z(s,a)$ is actually a random measure.

Recall that BOE tells us q^* is the fixed point of the Bellman optimality operator. The same logic holds here that under the optimal policy π^*, we expect to see $\mathscr{T}^{\pi^*} Z(s,a) \overset{d}{=} Z(s,a)$ for any (s,a). There are some problems we have to deal with here. Firstly, $\mathscr{T}^{\pi^*} Z(s,a)$ does not necessarily have the same support as $Z(s,a)$ so an L^2 projection Φ is needed to project the measure $\mathscr{T}^{\pi^*} Z(s,a)$ to a measure on the support $\{z_1, ..., z_{N_{atoms}}\}$. This provides the

distributional BOE

$$\Phi\mathscr{T}^{\pi^*}Z(s,a) \overset{d}{=} Z(s,a) \tag{485}$$

but how do we build up a TD algorithm based on this? To formalize the TD error, notice that we now have two measures and when it comes to measuring the distance between two measures, the natural choice is the KL divergence. As a result, we shall minimize

$$D_{KL}(\Phi\mathscr{T}^{\pi^*}Z(s,a)||Z(s,a)) \tag{486}$$

However, we don't know π^*, we proceed by writing out everything using the definition

$$\begin{aligned}\Phi\mathscr{T}^{\pi^*}Z(s,a) &= \Phi(R(s,a)+\gamma P^{\pi^*}Z(s,a)) &(487)\\ &= \Phi(R(s,a)+\gamma Z(S_{t+1},A_{t+1})),\ S_{t+1}\sim p(\cdot|S_t=s,A_t=a),\ A_{t+1}\sim\pi^*(\cdot|S_{t+1}) &(488)\end{aligned}$$

Since there is a deterministic version of π^*, it's clear that $A_{t+1}=\pi^*(S_{t+1})=\arg\max_a q^*(S_{t+1},a)$. Luckily, $q^*(s,a)=\mathbb{E}_{\pi^*}Z(s,a)$ so we can bootstrap and replace q^* with $Q_{\theta'}$, an approximation from the target network. Clearly, the action value function is the mean of $Z(s,a)$ so it can be computed quite easily, without any knowledge on π^*.

To provide a clear notation, we denote $d_t=(z,p_\theta(S_t,A_t))$ as a discrete probability distribution whose support is the collection of components of z. It has PMF $p_\theta(S_t,A_t)=\text{softmax}(\theta(S_t,A_t))$ so d_t is actually the distribution of $Z(S_t,A_t)$ condition on S_t,A_t. On the other hand, we denote

$$d'_t=(R_{t+1}\vec{1}+\gamma z, p_{\theta'}(S_{t+1},\arg\max_a Q_{\theta'}(S_{t+1},a)) \tag{489}$$

as a probability distribution with the components of $R_{t+1}\vec{1}+\gamma z$ as the support (this support has randomness if reward is randomized). The PMF of this distribution is actually the PMF of $\mathscr{T}^{\pi^*}Z(S_t,A_t)$ condition on S_t,A_t, which is calculated through the target network (parameter θ'), with the action value function calculated through

$$Q_{\theta'}(S_{t+1},a)=z\cdot p_{\theta'}(S_{t+1},a) \tag{490}$$

as the inner product between the support and the PMF, obviously. Finally, we organize the **distributional Q-learning algorithm** as the following optimization problem that on observing current state and action S_t,A_t, we want to solve

$$\min_\theta D_{KL}(\Phi d'_t||d_t) \tag{491}$$

$$\text{where }\begin{cases}d_t &= (z,p_\theta(S_t,A_t))\\ d'_t &= (R_{t+1}\vec{1}+\gamma z, p_{\theta'}(S_{t+1},\arg\max_a\{z\cdot p_{\theta'}(S_{t+1},a)\})\end{cases} \tag{492}$$

with Φ as the L^2 projection of a measure onto a measure with support $\{z_1,...,z_{N_{atoms}}\}$. Notice that similar to

DQN, the distribution of return can be approximated by NN, with the input as state and the output of dimension $N_{atoms} \times |\mathscr{A}|$, since a PMF is required for each state-action pair. Notice that we need $|\mathscr{A}|$ independent softmax transformations since the output of each action dimension needs to be normalized (as PMF). In this case, θ is just the parameter of this NN and we use target network to compute TD target (an out-of-date version similar to that in DQN). Φ is implemented as an operator that distribute probability mass of a measure at b onto a, c such that a is the largest number below b and c is the smallest number above b in the new support. The distribution of probability mass is linear in distance, i.e. if $\mathbb{P}(\{b\}) = p$, then the contribution of b to a, c is $\Phi(\mathbb{P})(\{a\}) = \frac{c-b}{c-a}p, \Phi(\mathbb{P})(\{c\}) = \frac{b-a}{c-a}p$. So far, the forward propagation of this distributional DQN can be implemented and the KL divergence of two discrete distributions can be easily calculated. After that, back propagation could be done to update the parameters θ of the network p_θ.

The author asserts that distributional RL matters because of state aliasing (sometimes the agent could not predict the reward timing, explicitly modelling the distribution makes it stable), a richer set of predictions, ability to apply for games that impose assumptions about the domain or the learning problem, and well-behaved optimization (KL divergence between discrete probability distribution). Despite the numerical improvements it brings with, distributional RL is crucial since it enables us to look into the blackbox (the model of the game) and get more information on which state and action is critical (e.g. the bimodal pattern and the tail of the distribution, which action is unrecoverably fatal, etc.).

NoisyNet

The exploration techniques like ε-greedy are commonly used, but it adds state-independent noise to the action and has a pitfall in those games where one has to make a lot of actions in order to get the first non-zero reward (e.g. Montezuma's revenge). Intuitively, those local perturbations are unlikely to lead to global behavioral patterns necessary for sufficient exploration. On the other hand, optimism in the face of uncertainty (e.g. UCB-type exploration) is efficient with theoretical guarantee but it often only applies for small state/action space and linear function approixmators (unlike NN which is nonlinear). Moreover, UCB-type exploration typically requires a set of hyperparameters to tune. To put up a universal framework for exploration in RL, NoisyNet is proposed in ***Noisy Networks for exploration by Meire Fortunato et. al.***

The main idea is to modify the model (the structure of NN) such that exploration is naturally encoded. Different from the normal NN which uses the linear mapping $y = b + Wx$ during forward propagation within two layers, NoisyNet uses the linear mapping with randomness, i.e.,

$$y = (b + Wx) + (b_{noisy} \odot \varepsilon^b + (W_{noisy} \odot \varepsilon^W)x) \tag{493}$$

where $\odot$ denotes element-wise product. In simple words, the idea is just to perturb bias b and weight W with a parametric random noise $\varepsilon^b, \varepsilon^W$. Let's assume for now that the input of this layer has dimension p and the output of this layer has dimension q, such that $b \in \mathbb{R}^q, W \in \mathbb{R}^{q\times p}, b_{noisy} \in \mathbb{R}^q, W_{noisy} \in \mathbb{R}^{q\times p}, \varepsilon^b \in \mathbb{R}^q, \varepsilon^W \in \mathbb{R}^{q\times p}$. Quoted from the original paper, "The key insight is that a single change to the weight vector can induce a consistent, and potentially very complex, state-dependent change in policy over multiple time steps.", which implements a totally different kind of exploration compared to the ε-greedy or the UCB-type exploration.

The choice of $\varepsilon^b, \varepsilon^W$ are not that hard, they are either chosen as independent standard Gaussians ($pq+q$ random noises), or as factorized Gaussians reducing the cost of random number generation. To explain the factorized Gaussian approach, just sample $p + q$ independent standard Gaussians $a_1, ..., a_p, b_1, ..., b_q$, and set

$$\varepsilon^W_{i,j} = f(b_i)f(a_j),\ \varepsilon^b_i = f(b_i),\ f(x) = \text{sgn}(x)\sqrt{|x|} \tag{494}$$

so we reduce the cost of generating random numbers. When it comes to the implementation in practice, each single layer of NoisyNet accepts input x and $\varepsilon^b, \varepsilon^W$ sampled, maps it to output y to finish the forward propagation.

When it comes to computing gradients in the backward propagation, the loss is now randomized due to the existence of $\varepsilon^b, \varepsilon^W$ so we shall minimize the expectation of such loss with the expectation taken w.r.t. the randomness in $\varepsilon^b, \varepsilon^W$. As a result, we shall minimize such expected loss to update the parameters of NoisyNet. This expectation could be calculated through Monte Carlo (by sampling several different $\varepsilon^b, \varepsilon^W$), or as a simpler approach, since NN parameters are updated through SGD, one sample of $\varepsilon^b, \varepsilon^W$ is enough to create an unbiased sample of the gradient direction in SGD. To clarify, denote the loss as $L(\theta, \varepsilon)$ with θ as the parameters of NN and ε as the noise, then we hope to minimize $\mathbb{E}_\varepsilon L(\theta, \varepsilon)$. Given ε, we know how to update θ through SGD, which is along direction $d(\theta, \varepsilon)$ such that $\mathbb{E}_d(d(\theta,\varepsilon)|\varepsilon) = -\nabla_\theta L(\theta,\varepsilon)$. Since

$$\mathbb{E}_{d,\varepsilon} d(\theta,\varepsilon) = \mathbb{E}_\varepsilon[\mathbb{E}_d(d(\theta,\varepsilon)|\varepsilon)] = -\mathbb{E}_\varepsilon \nabla_\theta L(\theta,\varepsilon) = -\nabla_\theta \mathbb{E}_\varepsilon L(\theta,\varepsilon) \tag{495}$$

the direction $d(\theta,\varepsilon)$ with two sources of randomness from d and ε is still an unbiased sample of SGD. As a result, one just has to sample one instance of ε (already done in forward propagation) and do the normal back propagation to update θ, it's still guaranteed to be an instance of SGD. This guarantees a nice compatibility with the current deep learning frameworks and one does not need to code NoisyNet from scratch.

Finally, we note that the NoisyNet is an improvement on the model to encode a universal framework of exploration, with the exploration to be state-correlated, hyperparameter-free, and applicable for any Deep RL algorithms. Interestingly, throughout numerical experiments of NoisyNet-DQN tracking the changes of W_{noisy}, in the last layer of NoisyNet, $\sum_i |(W_{noisy})_i|$ always gradually shrinks as learning proceeds but in the previous layers, it sometimes stays on the same level or even increases. This shows that "the NoisyNet-DQN agent does not necessarily evolve towards a deterministic solution as one might have expected and that NoisyNet produces a problem-specific exploration strategy as opposed to fixed exploration strategy used in standard DQN.", quoted from the paper.

Rainbow: combine improvements in value-based Deep RL

One of the best value-based RL methods is called rainbow, proposed in ***Rainbow: Combining Improvements in Deep Reinforcement Learning***. It combines all crucial improvements for value-based RL methods above and turns out to perform better than simply applying any single one of those improvements. In the following context, we briefly talk about how to combine them to construct the integrated agent.

The complete value-based RL method consists of the following parts: (i): RL and MDP framework (is it the classical RL framework, or the distributional one, or to maximize expected utility instead of expected return, etc.), (ii): function approximator (what model to use to approximate the value function), (iii): algorithm (e.g., SARSA or Q-learning), (iv): auxiliaries (e.g., replay buffer, target network etc.)

For part (i), we borrow the idea from distributional RL to learn the distribution of the return given current state and action instead of the action value function. For part (ii), we use the NoisyNet version of dueling network to approximate the distribution of the return. For part (iii), we use the multi-step Q-learning instead of the vanilla one. For part (iv), we use the prioritized replay buffer and the idea from double Q-learning to reduce maximization bias. This provides us with the framework of Rainbow.

In more details, consider the same setting as that in distributional RL, with $d_t = (z, p_\theta(S_t, A_t))$ given S_t, A_t to describe the distribution of return $Z(S_t, A_t)$. Now we rewrite the multi-step version of the distributional Q-learning that after n steps the distribution propagates to

$$d_t^{(n)} = (R_t^{(n)}\vec{1} + \gamma^n z, p_{\theta'}(S_{t+n}, \arg\max_a \{z \cdot p_{\theta'}(S_{t+n}, a)\})) \tag{496}$$

and our optimization problem is

$$\min_\theta D_{KL}(\Phi d_t^{(n)} || d_t) \tag{497}$$

where Φ is the projection of measure defined in previous contexts, $R_t^{(n)}$ is the forward n-step return. Notice that the idea of double Q-learning comes into play here, saying that we shall conduct action evaluation and figuring out greedy action with different networks. As a result, we shall modify the definition of $d_t^{(n)}$ to

$$d_t^{(n)} = (R_t^{(n)}\vec{1} + \gamma^n z, p_{\theta'}(S_{t+n}, \arg\max_a \{z \cdot p_{\theta}(S_{t+n}, a)\})) \tag{498}$$

using the online network p_θ to figure out the greedy action and the target network $p_{\theta'}$ to evaluate actions (please be careful with the slight difference here).

When it comes to the function approximators, since we are in the distributional setting, $p_\theta, p_{\theta'}$ has to output the distribution of return. Apply the idea of dueling network to approximate the return distribution with the value stream and the advantage stream. The value stream v_η is a network with state as input and outputs N_{atoms} numbers. The advantage stream a_ψ is a network with state as input and outputs $N_{atoms} \times |\mathscr{A}|$ numbers. After the state has been preprocessed (e.g. passing through convolution layers if the state is a picture) to get the state representation $f_\xi(s)$ (shared by two streams), we pass $f_\xi(s)$ through v_η and a_ψ respectively. Aggregating them in an identifiable way provides the following forward propagation rule

$$p_\theta(s,a) = \text{softmax}\left[v_\eta(f_\xi(s)) + a_\psi(f_\xi(s),a) - \frac{1}{|\mathscr{A}|}\sum_{a'\in\mathscr{A}} a_\psi(f_\xi(s),a')\right] \tag{499}$$

so p_θ accepts the state as input, and for each action a the output is a vector of length N_{atoms}. That's the way we are combining distributional RL with dueling network. Certainly, the universal exploration is implemented by integrating NoisyNet as a minor modification into this network structure. We denote θ as the collection of all those parameters, including those preprocessing parameters ξ, the value stream parameters η, the advantage stream parameters ψ, and a noisy version replication of those parameters due to the structure of NoisyNet.

Finally, we replace the vanilla replay buffer with the prioritized one. In prioritized replay for classical RL, the priority is proportional to the absolute TD error during the last visit. As an analogue, in distributional RL, KL divergence has the interpretation as the TD error. As a result, we set the priority as

$$p_t = D_{KL}(\Phi d_t^{(n)}||d_t) \tag{500}$$

with the remaining details (power law, importance sampling weights) to be exactly the same as that in prioritized replay. Notice that the replay buffer stores a single experience as $(S_t, A_t, R_t^{(n)}, S_{t+n})$ due to the fact that we are in the setting of n-step Q-learning.

As a conclusion for all value-based Deep RL methods, we provide the complete details in Alg. 19.

Remark. *Notice that some of the improvements, e.g., prioritized replay, NoisyNet, are applicable also for policy-based methods while the others, e.g., dueling network, DDQN are only applicable for value-based methods.*

Numerical experiments show that Rainbow scores 223% on learning speed while the normal DQN only scores 79%. If only one improvement is implemented, the best improvement comes from distributional DQN, scoring 164%, but there's still a large gap compared to Rainbow. If we compare the expected returns, Rainbow scores the most compared to single improvement methods on a majority of different Atari games.

Algorithm 19 Rainbow

```
Input: State preprocessing NoisyNet with parameter $\xi$, value stream NoisyNet with parameter $\eta$, advantage stream
    NoisyNet with parameter $\psi$, a replay buffer. Denote $\theta$ as the concatenation of all parameters $\xi, \eta, \psi$ and $p_\theta(s,a)$
    approximates the distribution of the return $Z(s,a)$ with its support encoded in vector $z \in \mathbb{R}^{N_{atoms}}$.
 1: Target network parameter $\theta' = \theta$
 2: repeat
 3:   Initial state $S_0$
 4:   for $t = 0, 1, 2, ..., T$ do
 5:     Sample factorized standard Gaussian noises $\varepsilon$ in NoisyNet
 6:     Forward propagation
          $$p_\theta(S_t, a) = \text{softmax}\left[v_\eta(f_\xi(S_t)) + a_\psi(f_\xi(S_t), a) - \frac{1}{|\mathscr{A}|}\sum_{a' \in \mathscr{A}} a_\psi(f_\xi(S_t), a')\right]$$
 7:     Calculate approximated action value function $Q_\theta(S_t, a) = z \cdot p_\theta(S_t, a)$
 8:     Select action $A_t = \arg\max_a Q_\theta(S_t, a)$ (no $\varepsilon$-greedy, exploration encoded in NoisyNet!)
 9:     Take action $A_t$, observe reward $R_{t+1}$ and state transition to $S_{t+1}$
10:     if $n$-step return $R_{t-n}^{(n)} = \sum_{k=0}^{n} \gamma^k R_{t-n+k+1}$ is available then
11:       Calculate $d_{t-n}^{(n)} = (R_{t-n}^{(n)}\vec{1} + \gamma^n z, p_{\theta'}(S_t, \arg\max_a \{z \cdot p_\theta(S_t, a)\}))$
12:       Calculate $D_{KL}(\Phi d_{t-n}^{(n)} || d_{t-n})$ where $d_{t-n} = (z, p_\theta(S_{t-n}, A_{t-n}))$
13:       If the experience $(S_{t-n}, A_{t-n}, R_{t-n}^{(n)}, S_t)$ is present in the replay buffer, set up its priority as $p_{t-n} =$
          $D_{KL}(\Phi d_{t-n}^{(n)} || d_{t-n})$ otherwise set its priority as maximal $\max_{k<t-n} p_k$
14:     end if
15:     Sample a random minibatch of experiences $\{(s_j, a_j, r_j, s'_j)\}_j$ from the replay buffer. The sampling distribu-
        tion assigns probability $\frac{p_i^\alpha}{\sum_k p_k^\alpha}$ to an experience with priority $p_i$, where $\alpha \in [0,1]$
16:     Compute importance sampling weights $w_j \propto \left(\frac{p_j^\alpha}{\sum_k p_k^\alpha}\right)^{-\beta}$ where $\beta \in [0,1]$ and shall be equal to 1 at the end
        of the training. Weights are normalized such that the maximum weight is equal to 1.
17:     For each experience $(s_j, a_j, r_j, s'_j)$, compute the loss $L(\theta, \varepsilon) = D_{KL}(\Phi d_{t-n}^{(n)} || d_{t-n})$ w.r.t. this experience.
        Update $\theta$ with importance sampling weights $\theta = \theta - \alpha^\theta w_j \nabla_\theta L(\theta, \varepsilon)$.
18:   end for
19:   Update $\theta'$ after a certain number of steps (through copying parameter $\theta$ or Polyak averaging)
20: until Enough number of iteration is done
Output: Parameter $\theta$ that approximates the distribution of return through $p_\theta$
```

Deep Deterministic Policy Gradient (DDPG)

Traditional value-based methods are constructed based on Q-learning so they can only deal with games with finite action space. When it comes to a continuous action space, is it still possible for us to use Q-learning? Let's start from the traditional practice in DeepRL that the policy $\pi(\cdot|s) = N(\mu_\theta(s), \sigma_\theta^2(s))$ is parameterized as a Gaussian with the mean and variance as the output of a NN. Here θ denote the parameter of the NN. Whenever we want to compute $\max_a Q(s, a)$, it's a painful and expensive sub-routine to do Gaussian optimization.

To avoid doing so, DDPG learns another function μ together with the value function where μ always maps state s to $\mu(s)$, which is the greedy action at state s. By doing this,

$$\max_a Q(s, a) \approx Q(s, \mu(s)) \tag{501}$$

solves the problem calculating the maximum for a continuous action space. As a result, we learn the policy μ and the value Q simultaneously, with the difference from traditional policy gradient methods that the parameterized policy μ is deterministic.

Algorithm 20 Deep Deterministic Policy Gradient (DDPG)

```
Input: $Q$-network $Q_\theta$, policy network $\mu_\phi$, a replay buffer
 1: Target network parameter $\theta' = \theta, \phi' = \phi$
 2: repeat
 3:   Initial state $S_0$
 4:   for $t = 0, 1, 2, ..., T$ do
 5:     Select action $A_t \sim \text{clip}(\mu_\phi(S_t) + \varepsilon; a_{\text{low}}, a_{\text{high}})$ with exploration
 6:     Take action $A_t$, observe reward $R_{t+1}$ and state transition to $S_{t+1}$
 7:     Store the experience $(S_t, A_t, R_{t+1}, S_{t+1})$ in the replay buffer
 8:     Sample a random minibatch of experiences $\{(s_j, a_j, r_j, s'_j)\}_j$ from the replay buffer for learning
 9:     Compute TD target $y_j = r_j + \gamma \cdot Q_{\theta'}(s'_j, \mu_{\phi'}(s'_j))$
10:     Compute $Q$-network loss $\sum_j [Q_\theta(s_j, a_j) - y_j]^2$ and do SGD to upudate $\theta$
11:     Compute policy network loss $\sum_j Q_\theta(s_j, \mu_\phi(s_j))$ and do stochastic gradient ascend to update $\phi$
12:   end for
13:   Update $\theta', \phi'$ after a certain number of steps (through copying parameter $\theta$ or Polyak averaging)
14: until Enough number of iteration is done
Output: Policy network $\mu_\phi$ as the greedy optimal policy
```

Then how do we update the network for Q and μ? Let's denote the Q-network as Q_θ and the policy network as μ_ϕ. In the context of DQN with a replay buffer and a target network, the Q-network loss for a single time step (experience) is formed as

$$[Q_\theta(S_t, A_t) - (R_{t+1} + \gamma \cdot Q_{\theta'}(S_{t+1}, \mu_{\phi'}(S_{t+1})))]^2 \tag{502}$$

where $Q_{\theta'}$ is the target Q-network and $\mu_{\phi'}$ is the target policy network, both organized as out-of-date copies of the original network. For the policy network μ_ϕ, since it shall maximize the Q value at each state, we update ϕ using gradient ascent for the reward function (negative loss) that

$$Q_\theta(S_t, \mu_\phi(S_t)) \tag{503}$$

Both loss functions can be computed for a batch of experience instead of a single experience in practice. For the exploration, we definitely don't want to always stick to the action provided by μ_ϕ since the policy is deterministic. The way is to introduce randomness $\varepsilon \sim N(0, \sigma^2)$ such that the action is sampled from a clipped distribution of $\mu_\phi(s) + \varepsilon$ at state s in the training procedure.

The complete details of DDPG are presented in Alg. 20. It's quite obvious that DDPG is just an extension of Q-learning to the continuous action space so it's an off-policy online algorithm. Due to the value-based nature of DDPG, it's known to be sample efficient, but it's notorious for its extreme brittleness and hyperparameter sensitivity. Refer to the paper ***Benchmarking Deep Reinforcement Learning for Continuous Control by Yan Duan et. al.*** for a comparison of the performance across different state-of-the-art DeepRL algorithms.

Twin Delayed DDPG (TD3)

Due to the hyperparameter sensitivity of DDPG and the maximization structure in updating the policy network μ_ϕ, Q-network is observed to overestimate the action values in some cases, which results in a breakdown of the deterministic policy. This is due to the fact that the favorable actions under μ_ϕ are misled by the errors in Q_θ. As a result, improvements on DDPG are made to get the TD3 method that is much more robust than DDPG.

The first improvement comes from **target policy smoothing**. In DDPG, we compute the TD target through $R_{t+1} + \gamma \cdot Q_{\theta'}(S_{t+1}, \mu_{\phi'}(S_{t+1}))$. The motivation of this improvement is to solve the failure mode where a peak caused by error in some action of Q_θ results in μ_ϕ immediately preferring the wrong action. If we smooth out Q_θ over similar actions, then this failure mode can be very well avoided. After inputting state S_{t+1} into the policy network, we do not directly use $\mu_{\phi'}(S_{t+1})$ to calculate the maximum of value function but clip the output of the policy network with some noise $\varepsilon \sim N(0, \sigma^2)$ instead.

$$a'(S_{t+1}) = \text{clip}(\mu_{\phi'}(S_{t+1}) + \text{clip}(\varepsilon; -c, c); a_{\text{low}}, a_{\text{high}}) \tag{504}$$

The clipping w.r.t. $a_{\text{low}}, a_{\text{high}}$ is just to ensure that the action is valid while the noise and the clipping w.r.t. $-c, c$ conducts the target policy smoothing.

Algorithm 21 Twin Delayed Deep Deterministic Policy Gradient (TD3)

```
Input: Two $Q$-networks $Q_{\theta_1}$ and $Q_{\theta_2}$, policy network $\mu_\phi$, a replay buffer, policy delay factor $N_{\text{delay}}$
 1: Target network parameter $\theta_1' = \theta_1, \theta_2' = \theta_2, \phi' = \phi$
 2: repeat
 3:     Initial state $S_0$
 4:     for $t = 0, 1, 2, ..., T$ do
 5:         Select action $A_t \sim \text{clip}(\mu_\phi(S_t) + \varepsilon; a_{\text{low}}, a_{\text{high}})$ with exploration
 6:         Take action $A_t$, observe reward $R_{t+1}$ and state transition to $S_{t+1}$
 7:         Store the experience $(S_t, A_t, R_{t+1}, S_{t+1})$ in the replay buffer
 8:         Sample a random minibatch of experiences $\{(s_j, a_j, r_j, s_j')\}_j$ from the replay buffer for learning
 9:         Compute the greedy action in the TD target $a'(s_j') = \text{clip}(\mu_{\phi'}(s_j') + \text{clip}(\varepsilon; -c, c); a_{\text{low}}, a_{\text{high}})$
10:         Compute TD target $y_j = r_j + \gamma \cdot \min_{i=1,2} Q_{\theta_i'}(s_j', a'(s_j'))$
11:         Compute $Q$-network loss $\sum_j [Q_\theta(s_j, a_j) - y_j]^2$ and do SGD to upudate $\theta$
12:     end for
13:     if per $N_{\text{delay}}$ steps then
14:         Sample a random minibatch of experiences $\{(s_j, a_j, r_j, s_j')\}_j$ from the replay buffer
15:         Compute policy network loss $\sum_j Q_{\theta_1}(s_j, \mu_\phi(s_j))$ and do stochastic gradient ascend to update $\phi$
16:         Update $\theta_1', \theta_2', \phi'$ through Polyak averaging
17:     end if
18: until Enough number of iteration is done
Output: Policy network $\mu_\phi$ as the greedy optimal policy
```

The second improvement comes from **clipped double Q-learning**. Since the failure of DDPG very much resembles the maximization bias, double Q-learning is a natural idea. We use two Q-networks Q_{θ_1} and Q_{θ_2}, then the TD target is formed as

$$R_{t+1} + \gamma \cdot \min_i Q_{\theta_i'}(S_{t+1}, a'(S_{t+1})) \tag{505}$$

with a minimum taken across the output of two Q-networks to fend off overestimation in Q. Both networks are updated based on the same TD target and the policy network is updated by maximizing $Q_{\theta_i}(S_t, \mu_\phi(S_t))$ for one of the Q-networks.

The third improvement is **delayed policy updates**. This means TD3 updates the policy network less often than the Q-network and updates the target network less often. This helps reduce the volatility that arises in the TD target, since TD target will be affected by a policy update. This shares the same motivation with the introduction of the target network in DQN, which is to reduce the variation in the TD target so that the policy learns something based on what it already could instead of learning from the beginning.

The complete details of TD3 are presented in Alg. 21. Obviously, TD3 is still an off-policy online algorithm. Compared to DDPG, TD3 has much lower hyperparameter sensitivity and a higher robustness level.

Maximum Entropy (MaxEnt) RL Framework and Robust RL Framework

We refer the readers to the paper ***Maximum Entropy RL (Provably) Solves Some Robust RL Problems by Benjamin Eysenbach and Sergey Levine*** for the detailed explanation. This is a paper with nice organization and explanations and I would strongly recommend those interested in MaxEnt take a look at it.

The MaxEnt RL is simply adding an entropy bonus term to the original RL objective. Let $H_\pi(A_t|S_t) = \int -\log\pi(a|S_t)\ \pi(a|S_t)\,da$ denote the entropy of the conditional distribution $\pi(\cdot|S_t)$, as a function of S_t. Notice that this is different from the traditional definition of conditional entropy, which is a deterministic number $H(X|Y) = \int\int -\log p(x|y)\ p(x,y)\,dx\,dy$. The objective of MaxEnt RL with transition kernel p and reward function r is

$$J_{\text{MaxEnt}}(\pi;p,r) = \mathbb{E}_{\pi,p}\left[\sum_{t=1}^{T} r(S_t,A_t) + \alpha H_\pi(A_t|S_t)\right] \tag{506}$$

and the agent aims to find the optimal policy to maximize this objective. Here we adopt the finite horizon setting without a discount factor but it does not cause a big problem since the policy is still time-homogeneous and one can easily recover the discounted setting by adding an absorbing state. The hyperparameter $\alpha > 0$ is called the temperature and balances the return term and the entropy term. MaxEnt is believed to provide optimal policy that is robust to perturbations in the model dynamics or the reward function.

On the other hand, robust RL framework maximizes the player's worst-case return. It's assumed that an underlying adversary exists and the game is thus formed as a two-player zero-sum game. The adversary acts first to minimize the player's return under a perturbed model dynamics $\tilde{p}$ and a perturbed reward function $\tilde{r}$. After that, the player maximizes the worst-case payoff which has already been minimized by the adversary. Denote $\tilde{\mathscr{P}}$ as the robust set for the model dynamics and $\tilde{\mathscr{R}}$ as the robust set for the reward function (of course we have to put some restrictions such that the perturbation is not that crazy), the objective in robust RL is

$$J_{\text{Robust}}(\pi;\tilde{\mathscr{P}},\tilde{\mathscr{R}}) = \min_{\tilde{p}\in\tilde{\mathscr{P}},\tilde{r}\in\tilde{\mathscr{R}}} \mathbb{E}_{\pi,\tilde{p}}\sum_{t=1}^{T}\tilde{r}(S_t,A_t) \tag{507}$$

clearly, the optimal policy in robust RL shares a similar interpretation with that in MaxEnt RL. The paper tells us the connection between those two frameworks and provide specific constructions for the robust sets.

We briefly describe the results and their proofs presented in the paper. The key lemma is the following, providing a very nice way to characterize the entropy of the policy! The proof is not hard but provides crucial interpretation, hence we also provide it here.

Lemma 13. *$\mathbb{E}_{p(x,y)}[-\log p(X|Y)] = \min_{f(x,y)}\mathbb{E}_{p(x,y)}[-f(X,Y) + \log\sum_{x'} e^{f(x',Y)}]$, so **the negative conditional entropy of X given Y matches the Frenchel conjugate of log-sum-exp** if $(X,Y)\sim p(x,y)$.*

Proof. Start from RHS and take derivative w.r.t. $f(x,y)$. Since $\mathbb{E}_{p(x,y)}f(X,Y) = \int\int p(s,t)f(s,t)\,ds\,dt$, we use functional derivative to introduce a small perturbation in f at (x,y) formed as the product of two Dirac point masses.

$$\frac{\delta}{\delta f(x,y)}\int\int p(s,t)f(s,t)\,ds\,dt = \lim_{\varepsilon\to 0}\frac{\int\int p(s,t)[f(s,t)+\varepsilon\delta_x(s)\delta_y(t)]\,ds\,dt - \int\int p(s,t)f(s,t)\,ds\,dt}{\varepsilon} \tag{508}$$

$$= \int\int p(s,t)\delta_x(s)\delta_y(t)\,ds\,dt = p(x,y) \tag{509}$$

For the second term, let's denote

$$\mathbb{E}_{p(x,y)}\log\sum_{x'}e^{f(x',Y)} = \int p_Y(t)\log\sum_{x'}e^{f(x',t)}\,dt \tag{510}$$

$$= \int p_Y(t)\log(\sum_{x'\neq x}e^{f(x',t)} + e^{f(x,t)})\,dt \tag{511}$$

this decomposition of sum is natural since we want to figure out the sensitivity w.r.t. f at (x,y). A perturbation formed as Dirac Delta is added to f at (x,y) to get

$$\frac{\delta}{\delta f(x,y)}\mathbb{E}_{p(x,y)}\log\sum_{x'}e^{f(x',Y)} = \lim_{\varepsilon\to 0}\frac{\int p_Y(t)\log(\sum_{x'\neq x}e^{f(x',t)} + e^{f(x,t)+\varepsilon\delta_y(t)})\,dt - \int p_Y(t)\log(\sum_{x'\neq x}e^{f(x',t)} + e^{f(x,t)})\,dt}{\varepsilon} \tag{512}$$

notice that at (x',t) where $x'\neq x$, there is no perturbation and at (x,t), the perturbation is only in the y component since $\delta_x(x)=1$. Using $\log(1+x)\approx x, e^x\approx 1+x$ $(x\to 0)$, we get

$$\frac{\delta}{\delta f(x,y)}\mathbb{E}_{p(x,y)}\log\sum_{x'}e^{f(x',Y)} = \lim_{\varepsilon\to 0}\frac{1}{\varepsilon}\int p_Y(t)\log\frac{\sum_{x'\neq x}e^{f(x',t)} + e^{f(x,t)+\varepsilon\delta_y(t)}}{\sum_{x'}e^{f(x',t)}}\,dt \tag{513}$$

$$= \lim_{\varepsilon\to 0}\frac{1}{\varepsilon}\int p_Y(t)\frac{e^{f(x,t)+\varepsilon\delta_y(t)} - e^{f(x,t)}}{\sum_{x'}e^{f(x',t)}}\,dt \tag{514}$$

$$= \int p_Y(t)\delta_y(t)\frac{e^{f(x,t)}}{\sum_{x'}e^{f(x',t)}}\,dt = \frac{e^{f(x,y)}}{\sum_{x'}e^{f(x',y)}}p_Y(y) \tag{515}$$

Denote $p_Y(y)=p(y)$ as the marginal likelihood of Y, the optimal $f^*(x,y)$ on the RHS satisfies

$$p(x|y) = \frac{p(x,y)}{p(y)} = \frac{e^{f^*(x,y)}}{\sum_{x'}e^{f^*(x',y)}} \tag{516}$$

taking log and expectation on both sides w.r.t. $(X,Y)\sim p(x,y)$ yields

$$-\mathbb{E}_{p(x,y)}\log p(X|Y) = -\mathbb{E}_{p(x,y)}\log\frac{e^{f^*(X,Y)}}{\sum_{x'}e^{f^*(x',Y)}} = \min_{f(x,y)}\mathbb{E}_{p(x,y)}[-f(X,Y)+\log\sum_{x'}e^{f(x',Y)}] \tag{517}$$

□

Remark. *The LHS of the lemma is the conditional entropy of $X|Y$ under $(X,Y) \sim p$, as a real number. For the RHS, let's add a negative sign so it becomes*

$$\max_{f(x,y)} \mathbb{E}_{p(x,y)}[f(X,Y) - \log \sum_{x'} e^{f(x',Y)}] \tag{518}$$

so the structure of Frenchel conjugate becomes clear.

This is the key lemma connecting MaxEnt and robust RL framework. Specifying p as the policy, X as the action and Y as the state, then we recover the entropy bonus term in the objective of MaxEnt RL. Let's state the key conclusions in the paper. The first is about the robustness in the reward function.

Theorem 15. *With finite reward function r and a soft policy π (positive probability for any state-action pair), there exists $\varepsilon > 0$ such that*

$$\forall \pi, J_{\text{MaxEnt}}(\pi; p, r) = \min_{\tilde{r} \in \tilde{\mathscr{R}}(\pi)} \mathbb{E}_{\pi,p} \sum_t \tilde{r}(S_t, A_t) \tag{519}$$

where the robust set has an explicit form

$$\tilde{\mathscr{R}}(\pi) = \left\{ \tilde{r} : \mathbb{E}_{\pi,p} \sum_t \log \int e^{r(S_t,a) - \tilde{r}(S_t,a)} \, da \leq \varepsilon \right\} \tag{520}$$

Proof. The robust set seems daunting but it is actually constructed based on the log-sum-exp in the lemma above! Directly use Lagrange multiplier method on the RHS, with $\tilde{\mathscr{R}}(\pi)$ to be strictly feasible, there exists some $\varepsilon > 0$ such that the multiplier λ can be taken as 1. This provides the unconstrained optimization

$$\min_{\Delta r} \mathbb{E}_{\pi,p} \sum_t r(S_t, A_t) - \sum_t \Delta r(S_t, A_t) + \sum_t \log \int e^{\Delta r(S_t,a)} \, da \tag{521}$$

where $\Delta r = r - \tilde{r}$. It suffices to prove $\min_{\Delta r} \mathbb{E}_{\pi,p}[-\sum_t \Delta r(S_t, A_t) + \sum_t \log \int e^{\Delta r(S_t,a)} \, da] = \sum_t \mathbb{E}_{\pi,p} H_\pi(A_t|S_t)$ so that the objective exactly equals the MaxEnt objective. Notice that

$$\min_{\Delta r} \mathbb{E}_{\pi,p}[-\sum_t \Delta r(S_t, A_t) + \sum_t \log \int e^{\Delta r(S_t,a)} \, da] = \min_{\Delta r} \sum_t \mathbb{E}_{\rho_t^\pi}[-\Delta r(S, A) + \log \int e^{\Delta r(S,a)} \, da] \tag{522}$$

where ρ_t^π is the on-policy distribution at time t under policy π, which is a measure on $\mathscr{S} \times \mathscr{A}$. Due to the lemma, the minimizer Δr of $\mathbb{E}_{\rho_t^\pi}[-\Delta r(S, A) + \log \int e^{\Delta r(S,a)} \, da]$ satisfies $\pi(a|s) = \rho_t^\pi(a|s) = \frac{e^{\Delta r(s,a)}}{\int e^{\Delta r(s,a')} \, da'}$ since policy is time-homogeneous. As a result, the same Δr attains the minimum across different time t and we can exchange the sum and the minimum. We yield

$$\sum_t \min_{\Delta r} \mathbb{E}_{\rho_t^\pi}[-\Delta r(S, A) + \log \int e^{\Delta r(S,a)} \, da] = \sum_t \mathbb{E}_{\pi,p} H_\pi(A_t|S_t) \tag{523}$$

which concludes the proof from the lemma. □

Similarly, the robustness in transition kernel p can also be established through the lemma.

Theorem 16. *Assume the dynamics have finite entropy $H_p(S_{t+1}|S_t, A_t) < \infty$ for any state-action pairs. Then there exists $\varepsilon > 0$ such that*

$$\log \min_{\tilde{p}\in\tilde{\mathscr{P}}(\pi)} \mathbb{E}_{\pi,\tilde{p}} \sum_t r(S_t, A_t) \geq J_{MaxEnt}(\pi; p, \overline{r}) + \log T \tag{524}$$

where $\overline{r}(S_t, A_t, S_{t+1}) = \frac{1}{T}\log r(S_t, A_t) + H_p(S_{t+1}|S_t, A_t)$ is a different reward function and the robust set is provided as

$$\tilde{\mathscr{P}}(\pi) = \left\{\tilde{p} : \mathbb{E}_{\pi,p} \sum_t \log \int\int \frac{p(s'|S_t, a)}{\tilde{p}(s'|S_t, a)}\, da\, ds' \leq \varepsilon\right\} \tag{525}$$

Proof. The idea is to first transform dynamics variation into reward variation. We start from the LHS and first derive a lower bound for $\log \mathbb{E}_{\pi,\tilde{p}} \sum_t r(S_t, A_t)$ that

$$\log \mathbb{E}_{\pi,\tilde{p}} \sum_t r(S_t, A_t) = \log \mathbb{E}_{\pi,p} \prod_t \frac{\tilde{p}(S_{t+1}|S_t, A_t)}{p(S_{t+1}|S_t, A_t)} \sum_t r(S_t, A_t) \tag{526}$$

$$\geq \mathbb{E}_{\pi,p} \left[\sum_t \log \frac{\tilde{p}(S_{t+1}|S_t, A_t)}{p(S_{t+1}|S_t, A_t)} + \log \sum_t r(S_t, A_t)\right] \tag{527}$$

conducting change of measure and using Jensen's inequality for $f(x) = e^x$. Let's establish a lower bound for $\mathbb{E}_{\pi,p} \log \sum_t r(S_t, A_t)$ by applying Jensen's inequality for $f(x) = \log x$.

$$= \mathbb{E}_{\pi,p} \left[\sum_t \log \frac{\tilde{p}(S_{t+1}|S_t, A_t)}{p(S_{t+1}|S_t, A_t)} + \log \frac{1}{T} \sum_t r(S_t, A_t)\right] + \log T \tag{528}$$

$$\geq \mathbb{E}_{\pi,p} \left[\sum_t \log \frac{\tilde{p}(S_{t+1}|S_t, A_t)}{p(S_{t+1}|S_t, A_t)} + \frac{1}{T} \sum_t \log r(S_t, A_t)\right] + \log T \tag{529}$$

this $\frac{1}{T}\sum_t \log r(S_t, A_t)$ terms matches the first term in the new reward $\overline{r}$.

Next, we set $\Delta r(s', s, a) = \log \frac{p(s'|s,a)}{\tilde{p}(s'|s,a)}$, changing the minimization in $\tilde{p}$ to the minimization in Δr. However, there is an extra constraint that $\tilde{p}$ is a transition kernel so $\int \tilde{p}(s'|s, a)\, ds' = 1$. As a result, we establish the constrained optimization problem of the lower bound

$$\min_{\Delta r} \mathbb{E}_{\pi,p} \left[-\sum_t \Delta r(S_{t+1}, S_t, A_t) + \frac{1}{T} \sum_t \log r(S_t, A_t)\right] + \log T \tag{530}$$

$$s.t.\ \mathbb{E}_{\pi,p} \sum_t \log \int\int e^{\Delta r(s', S_t, a)}\, da\, ds' \leq \varepsilon,\ \int p(s'|s, a) e^{-\Delta r(s',s,a)}\, ds' = 1,\ \forall (s, a) \in \mathscr{S} \times \mathscr{A} \tag{531}$$

interestingly, the density constraint does not change the optimization since one can always find $c(s, a)$ to conduct a translation on $\Delta r(s', s, a)$ such that the density constraint is satisfied without changing the objective. We apply

Langrange multiplier method and choose an ε such that the multiplier $\lambda = 1$ if $\tilde{\mathscr{P}}(\pi)$ is strictly feasible. The same trick provides an unconstrained optimization

$$\min_{\Delta r} \mathbb{E}_{\pi,p}\left[-\sum_t \Delta r(S_{t+1}, S_t, A_t) + \frac{1}{T}\sum_t \log r(S_t, A_t) + \sum_t \log \int\int e^{\Delta r(s', S_t, a)}\, da\, ds'\right] + \log T \tag{532}$$

$$= \min_{\Delta r} \sum_t \mathbb{E}_{\pi,p}\left[-\Delta r(S_{t+1}, S_t, A_t) + \log \int\int e^{\Delta r(s', S_t, a)}\, da\, ds'\right] + \log T + \mathbb{E}_{\pi,p}\frac{1}{T}\sum_t \log r(S_t, A_t) \tag{533}$$

the first term again looks similar to what we have in the lemma. A same argument interchanging minimum and sum provides

$$\min_{\Delta r} \sum_t \mathbb{E}_{\pi,p}\left[-\Delta r(S_{t+1}, S_t, A_t) + \log \int\int e^{\Delta r(s', S_t, a)}\, da\, ds'\right] \tag{534}$$

$$= \min_{\Delta r} \sum_t \mathbb{E}_{\rho_t^{\pi,p}}\left[-\Delta r(S', S, A) + \log \int\int e^{\Delta r(s', S, a)}\, da\, ds'\right] \tag{535}$$

$$= \sum_t \min_{\Delta r} \mathbb{E}_{\rho_t^{\pi,p}}\left[-\Delta r(S', S, A) + \log \int\int e^{\Delta r(s', S, a)}\, da\, ds'\right] \tag{536}$$

$$= -\sum_t \mathbb{E}_{\pi,p} \log p(A_t, S_{t+1}|S_t) \tag{537}$$

here $\rho_t^{\pi,p}$ is still the on-policy distribution of (S_{t+1}, S_t, A_t) under π, p, as a measure on $\mathscr{S} \times \mathscr{S} \times \mathscr{A}$. Notice that $\rho_t^{\pi,p}(s', a|s) = \pi(a|s)p(s'|s, a)$ is also time-homogeneous, minimum can interchange with sum. An easy decomposition tells us that $-\sum_t \mathbb{E}_{\pi,p} \log p(A_t, S_{t+1}|S_t) = \mathbb{E}_{\pi,p}\sum_t H_\pi(A_t|S_t) + \mathbb{E}_{\pi,p}\sum_t H_p(S_{t+1}|S_t, A_t)$. As a result,

$$\log \min_{\tilde{p}\in\tilde{\mathscr{P}}(\pi)} \mathbb{E}_{\pi,\tilde{p}} \sum_t r(S_t, A_t) \geq \mathbb{E}_{\pi,p}\sum_t H_\pi(A_t|S_t) + \mathbb{E}_{\pi,p}\sum_t H_p(S_{t+1}|S_t, A_t) + \log T + \mathbb{E}_{\pi,p}\frac{1}{T}\sum_t \log r(S_t, A_t) \tag{538}$$

$$= \mathbb{E}_{\pi,p}\sum_t H_\pi(A_t|S_t) + \log T + \mathbb{E}_{\pi,p}\sum_t \overline{r}(S_t, A_t) \tag{539}$$

$$= J_{\text{MaxEnt}}(\pi; p, \overline{r}) + \log T \tag{540}$$

concludes the proof. □

The key idea in those proof presented is that one first develop a technical lemma to characterize the entropy bonus term in the MaxEnt objective. After that, one designs the robust set (i.e. the constraint of the optimization problem in robust RL) to exactly match the form of the lemma. From this perspective, the construction of the robust sets and new reward $\overline{r}$ are natural since they are **a posteriori constructions**.

Remark. *The two theorems above do not require any convexity in r or p, even the boundedness of the reward function is not necessary! In the original paper, ε in the robustness of the dynamics has an explicit lower bound. If the state* *space $\mathscr{S}$ is discrete, then*

$$\varepsilon \geq T\, \mathbb{E}_{\pi,p} H_\pi(A_t|S_t) \tag{541}$$

so ***the size of the robust set should be at least as large as the policy's entropy****. Such ε in both results can be understood as the budget of the adversary. The adversary can make large changes in a few places in the reward and the dynamics or to make smaller changes to a lot of places in the reward and the dynamics.*

When there are perturbations in the dynamics, we shall use a modified reward function $\overline{r}(s,a,s') = \frac{1}{T}\log r(s,a) + H_p(S_{t+1}|S_t = s, A_t = a)$. However, the entropy term is not known in the model-free setting so practically we may assume it's approximately constant and use $\overline{r}(s,a) = \frac{1}{T}\log r(s,a)$ instead. The MaxEnt RL with reward $\overline{r}$ provides a lower bound for the objective of the robust RL with original reward function r.

By combining two results above, we easily get a result that works for simultaneous perturbations in the reward function and the dynamics. Notice that some of the proofs above in the original paper suffer from typos and mistakes so I re-presented them in my notes for clarification. There are a lot more examples and numerical works shown in the original paper that provides one with much insight on the MaxEnt RL framework.

Soft Actor-Critic (SAC)

As proved above, MaxEnt RL framework is a nice framework to use since it increases policy robustness towards perturbations in both the MDP dynamics and the reward function. To build up a practically useful algorithm based on the MaxEnt framework, let's first re-build the MDP theory after an entropy bonus term is added to the objective. For simplicity, we use the infinite time horizon setting with a discount rate. The goal is to find the optimal policy π^* that maximizes

$$\mathbb{E}_\pi\left(\sum_{t=0}^{\infty}\gamma^t[r(S_t,A_t)+\alpha H_\pi(A_t|S_t)]\right) \tag{542}$$

since the temperature α can be absorbed into the reward function, we assume $\alpha=1$. we define the state and action value function

$$v_\pi(s)=\mathbb{E}_\pi\left(\sum_{t=0}^{\infty}\gamma^t[r(S_t,A_t)+H_\pi(A_t|S_t)]\Big|S_0=s\right) \tag{543}$$

$$q_\pi(s,a)=\mathbb{E}_\pi\left(\sum_{t=0}^{\infty}\gamma^t[r(S_t,A_t)+H_\pi(A_t|S_t)]\Big|S_0=s,A_0=a\right) \tag{544}$$

where $\mathbb{E}_\pi(H_\pi(A_0|S_0)|S_0=s,A_0=a)=0$ within the action value function. The definition is similar to that in the traditional RL framework. We derive the relationship between those two value functions

$$v_\pi(s)=\mathbb{E}_{A\sim\pi(\cdot|s)}q_\pi(s,A)+H_\pi(A_0|S_0=s) \tag{545}$$

$$q_\pi(s,a)=r(s,a)+\gamma\mathbb{E}_{S'\sim p(\cdot|s,a)}v_\pi(S') \tag{546}$$

In the MaxEnt framework, the traditional policy iteration can be generalized to the **soft policy iteration**. The policy evaluation can be done similarly to that in the traditional RL but we require a new policy improvement theorem. It turns out that the policy improvement step does not generate a deterministic policy any longer, but a soft policy instead. Let's prove the following soft policy improvement theorem.

Theorem 17. *Let $\pi'\in\Pi$ be defined based on $\pi\in\Pi$ such that*

$$\forall s\in\mathscr{S},\pi'(\cdot|s)=\arg\min_{\pi''\in\Pi}D_{KL}\left(\pi''(\cdot|s)||\frac{e^{q_\pi(s,\cdot)}}{Z^\pi(s)}\right) \tag{547}$$

where $Z^\pi(s)=\sum_a e^{q_\pi(s,a)}$ is the normalization factor, then $\forall(s,a),q_{\pi'}(s,a)\geq q_\pi(s,a)$.

Proof. For fixed state s, rewrite

$$J_\pi(\pi''(\cdot|s)) = D_{KL}\left(\pi''(\cdot|s)||\frac{e^{q_\pi(s,\cdot)}}{Z^\pi(s)}\right) = \int \pi''(a|s)\left[\log \pi''(a|s) - \log \frac{e^{q_\pi(s,a)}}{Z^\pi(s)}\right] da \tag{548}$$

$$= \int \pi''(a|s)\left[\log \pi''(a|s) - q_\pi(s,a) + \log Z^\pi(s)\right] da = \mathbb{E}_{A\sim\pi''(\cdot|s)}[\log \pi''(A|S=s) - q_\pi(s,A)] + \log Z^\pi(s) \tag{549}$$

so that $J_\pi(\pi'(\cdot|s)) \leq J_\pi(\pi(\cdot|s))$ by the definition of π'. This tells us

$$\mathbb{E}_{A\sim\pi'(\cdot|s)}[\log \pi'(A|S=s) - q_\pi(s,A)] \leq \mathbb{E}_{A\sim\pi(\cdot|s)}[\log \pi(A|S=s) - q_\pi(s,A)] = -v_\pi(s) \tag{550}$$

so $H_{\pi'}(A|S=s) + \mathbb{E}_{A\sim\pi'(\cdot|s)} q_\pi(s,A) \geq v_\pi(s)$ connects policy π and π'. The improvement can be proved through iteratively applying this inequality. Let's assume that $S_0 = s, A_0 = a$ for simplicity, then

$$q_\pi(s,a) = r(s,a) + \gamma\mathbb{E}v_\pi(S_1) \tag{551}$$

$$\leq r(s,a) + \gamma\mathbb{E}H_{\pi'}(A_1|S_1) + \gamma\mathbb{E}_{\pi'}q_\pi(S_1,A_1) \tag{552}$$

$$\leq ... \leq r(s,a) + \sum_{t=1}^{\infty}\gamma^t[\mathbb{E}H_{\pi'}(A_t|S_t) + \mathbb{E}_{\pi'}r(S_t,A_t)] = q_{\pi'}(s,a) \tag{553}$$

proves the soft policy improvement theorem. □

Remark. *Due to the presence of entropy bonus in the objective, if* Π *is the whole policy space (without any constraint on the policy), then the policy improvement of* π *is a soft policy corresponding to the Boltzmann distribution induced by* q_π*. This structure is very different from that in the traditional RL framework, where the improved policy is deterministic. The KL divergence appears merely as a projection to the admissible policy space* Π*, which helps us when we are parameterizing the policy using a neural network.*

Now let's start with the SAC method. Three neural networks are constructed as $V_\psi, Q_\theta, \pi_\phi$ parameterizing the optimal state value, action value and the policy. The update of ψ is based on the consistency condition that $V_\psi(S_t)$ shall match $\mathbb{E}_{\pi_\phi}[Q_\theta(S_t,A_t) - \log \pi_\phi(A_t|S_t)]$ on knowing S_t. As a result, the loss for V-network is formed as

$$\frac{1}{2}\left(V_\psi(S_t) - \mathbb{E}_{\pi_\phi}[Q_\theta(S_t,A_t) - \log \pi_\phi(A_t|S_t)]\right)^2 \tag{554}$$

Since the loss still contains an expectation w.r.t. current action A, we calculate the gradient of such loss w.r.t. ψ as

$$\left(V_\psi(S_t) - \mathbb{E}_{\pi_\phi}[Q_\theta(S_t,A_t) - \log \pi_\phi(A_t|S_t)]\right)\nabla_\psi V_\psi(S_t) \tag{555}$$

and use the idea of SGD, sample an unbiased gradient direction

$$d_V = \left(V_\psi(S_t) - Q_\theta(S_t,a(S_t)) + \log \pi_\phi(a(S_t)|S_t)\right)\nabla_\psi V_\psi(S_t),\ a(S_t) \sim \pi_\phi(\cdot|S_t) \tag{556}$$

What is crucial is that **the action $a(S_t)$ in d_V must be sampled from the current policy network and the action a in the experience (s, a, r, s') within the replay buffer cannot be used!** This is somewhat counter-intuitive but ensures the unbiasedness of SGD.

When updating Q_θ, the consistency condition $Q_\theta(S_t, A_t) = r(S_t, A_t) + \gamma \mathbb{E} V_{\psi'}(S_{t+1})$ comes into play. Since this is the TD target, we use target V-network $V_{\psi'}$ as a common practice. The Q-network loss is formed as

$$\frac{1}{2}[Q_\theta(S_t, A_t) - r(S_t, A_t) - \gamma \mathbb{E} V_{\psi'}(S_{t+1})]^2 \tag{557}$$

the loss still contains an expectation and requires subsequent operations. Compute its gradient w.r.t. θ as

$$[Q_\theta(S_t, A_t) - r(S_t, A_t) - \gamma \mathbb{E} V_{\psi'}(S_{t+1})]\nabla_\theta Q_\theta(S_t, A_t) \tag{558}$$

we do the same thing once again, taking an unbiased sample of the gradient direction to conduct SGD along

$$d_Q = [Q_\theta(S_t, A_t) - r(S_t, A_t) - \gamma V_{\psi'}(S_{t+1})]\nabla_\theta Q_\theta(S_t, A_t) \tag{559}$$

The action A_t in d_Q is from the replay buffer and does not need to be sampled according to the current policy. This is due to the fact that the expectation in the gradient of V-network loss is w.r.t. π_ϕ but the expectation in the gradient of Q-network loss is only w.r.t. the MDP dynamics.

Finally, when updating the policy network π_ϕ, we wish to use the soft policy improvement theorem. The policy network loss is naturally formed as

$$D_{KL}\left(\pi_\phi(\cdot|S_t) || \frac{e^{Q_\theta(S_t,\cdot)}}{Z_\theta(S_t)}\right) = \mathbb{E}_{\pi_\phi}[\log \pi_\phi(A_t|S_t) - Q_\theta(S_t, A_t)] + \log Z_\theta(S_t) \tag{560}$$

take gradient w.r.t. ϕ, we get

$$\nabla_\phi \left(\mathbb{E}_{\pi_\phi}[\log \pi_\phi(A_t|S_t) - Q_\theta(S_t, A_t)]\right) \tag{561}$$

it's hard to proceed at this moment due to the dependence of the measure π_ϕ on ϕ and we are not allowed to interchange the gradient and the expectation. However, a clever **reparameterization trick** helps us here. Recall the canonical construction of the policy network is to input state s and output the mean and the variance of a Gaussian distribution such that $\pi_\phi(\cdot|s) = N(\mu_\phi(s), \sigma_\phi^2(s))$. Due to the fact that most games have bounded action spaces, a squashed Gaussian policy is used here

$$f_\phi(s, \xi) = \tanh(\mu_\phi(s) + \sigma_\phi(s) \odot \xi),\ \xi \sim N(0, I) \tag{562}$$

with μ_ϕ, σ_ϕ formed as outputs of the policy network. At this point we return to the gradient

$$\nabla_\phi \left(\mathbb{E}_{\pi_\phi}[\log \pi_\phi(A_t|S_t) - Q_\theta(S_t, A_t)]\right) = \nabla_\phi \left(\mathbb{E}_{\xi\sim N(0,I)}[\log \pi_\phi(f_\phi(S_t, \xi)|S_t) - Q_\theta(S_t, f_\phi(S_t, \xi))]\right) \tag{563}$$

$$= \mathbb{E}_{\xi\sim N(0,I)}\nabla_\phi[\log \pi_\phi(f_\phi(S_t, \xi)|S_t) - Q_\theta(S_t, f_\phi(S_t, \xi))] \tag{564}$$

the measure no longer has dependence on ϕ and the interchange is allowed.

Remark. *The reparametrization trick refers to representing the measure in a special way (e.g. standardization of Gaussian) to remove the dependence on the parameter of the measure.*

Notice that we only need one policy network since π_ϕ is implicitly encoded in terms of f_ϕ. To clarify, the policy network set up in the real implementation is an underlying NN that take state s as input and outputs $\mu_\phi(s)$ and $\sigma_\phi(s)$, which is neither f_ϕ nor π_ϕ. From the output of this NN and the sampling of $\xi \sim N(0, I)$, f_ϕ is recovered. Drawing independent samples (actions) from $\pi_\phi(\cdot|s)$ is equivalent to calling $f_\phi(s, \xi)$ for multiple independent copies of ξ. The likelihood $\pi(a|s)$ is just the likelihood of the r.v. $\tanh(\mu_\phi(s) + \sigma_\phi(s) \odot \xi)$ at a, which can be explicitly calculated since the r.v. is a transformation of Gaussian.

As a result, an unbiased sample of the gradient direction in SGD is just

$$d_\pi = \nabla_\phi[\log \pi_\phi(f_\phi(S_t, \xi)|S_t) - Q_\theta(S_t, f_\phi(S_t, \xi))] \tag{565}$$

providing all the details of SAC. Inspired by We organize everything together as Alg. 22.

Algorithm 22 Soft Actor-Critic (SAC)

```
Input: $V$-network $V_\psi$, $Q$-network $Q_\theta$, policy network $\pi_\phi$, a replay buffer
1: Target network parameter $\psi' = \psi$
2: repeat
3:   Initial state $S_0$
4:   for $t = 0, 1, 2, ..., T$ do
5:     Select action $A_t \sim f_\phi(S_t, \xi) = \tanh(\mu_\phi(s) + \sigma_\phi(s) \odot \xi),\ \xi \sim N(0, I)$
6:     Take action $A_t$, observe reward $R_{t+1}$ and state transition to $S_{t+1}$
7:     Store the experience $(S_t, A_t, R_{t+1}, S_{t+1})$ in the replay buffer
8:     Sample a random minibatch of experiences $\{(s_j, a_j, r_j, s'_j)\}_j$ from the replay buffer for learning
9:     Update $\psi$ through SGD along $d_V = (V_\psi(s_j) - Q_\theta(s_j, a(s_j)) + \log \pi_\phi(a(s_j)|s_j)) \nabla_\psi V_\psi(s_j),\ a(s_j) \sim \pi_\phi(\cdot|s_j)$
10:    Update $\theta$ through SGD along $d_Q = [Q_\theta(s_j, a_j) - r_j - \gamma V_{\psi'}(s'_j)] \nabla_\theta Q_\theta(s_j, a_j)$
11:    Update $\phi$ through SGD along $d_\pi = \nabla_\phi[\log \pi_\phi(f_\phi(s_j, \xi)|s_j) - Q_\theta(s_j, f_\phi(s_j, \xi))]$
12:  end for
13:  Update $\psi'$ through Polyak averaging per certain steps
14: until Enough number of iteration is done
Output: Policy network $\mu_\phi$ as the greedy optimal policy
```

Subsequent improvements of SAV include the introduction of clipped double Q-learning from TD3 to reduce overestimation in Q and cancelling the V-network for simplicity. However, the fundamental idea of SAC remains the same. Although SAC has the name of the actor-critic algorithm, it's actually off-policy and is closer to Q-learning under the MaxEnt framework. The only difference is that under MaxEnt framework, the improved policy in Q-learning is soft but not deterministic. SAC is found to significantly accelerate the learning process with stability over randomness and encodes natural exploration with a stochastic actor.

Evolution Strategy (ES)

As published in the paper ***Evolution strategies as a scalable alternative to reinforcement learning by Tim Salimans et. al.***, there is an application of evolution strategy in RL. Instead of thinking about the RL problem, let's view the RL problem as a black-box with input θ as the parameter of the parameterized policy π_θ (e.g. an NN) and outputs $F(\theta)$, the overall return. Our objective is just to update θ to maximize $\mathbb{E}F(\theta)$, ignoring everything that happens inside the black-box. The optimization can be done through evolution strategy, motivated by biological intuition (e.g. the genetic algorithm) that we first propose a population of possible θ (a distribution on the parameter space) and update θ towards the direction in which the expectation of the overall return grows the fastest.

In detail, we introduce a Gaussian noise centered at θ, denoted as $X \sim N(\theta, \sigma^2 I)$ and we hope to calculate $\nabla_\theta \mathbb{E}F(X)$ as the update direction of θ. The calculation can be done with a similar trick in policy gradient theorem that

$$\nabla_\theta \mathbb{E}F(X) = \nabla_\theta \int F(x) f(x,\theta)\, dx = \int F(x) \frac{\nabla_\theta f(x,\theta)}{f(x,\theta)} f(x,\theta)\, dx \tag{566}$$

$$= \mathbb{E}[F(X) \nabla_\theta \log f(X,\theta)] \tag{567}$$

where $f(x,\theta)$ is the Gaussian likelihood of $X \sim N(\theta, \sigma^2 I)$, which is also a function in θ. Simple calculation using Gaussian likelihood shows

$$\mathbb{E}[F(X)\nabla_\theta \log f(X,\theta)] = \mathbb{E}\left[F(X)(X-\theta)\frac{1}{\sigma^2}\right] \tag{568}$$

now reparameterize the r.v. $X = \theta + \sigma\varepsilon, \varepsilon \sim N(0, I)$, we see that

$$\nabla_\theta \mathbb{E}F(X) = \frac{1}{\sigma}\mathbb{E}_{\varepsilon \sim N(0,I)}\left[F(\theta + \sigma\varepsilon)\varepsilon\right] \tag{569}$$

Adopting the idea of stochastic gradient ascend, we only need to sample a batch of $\varepsilon_1, ..., \varepsilon_n \sim N(0, I)$, compute returns $F_i = F(\theta + \sigma\varepsilon_i)$ by inputting $\theta + \sigma\varepsilon_i$ into the black-box (interacting with the environment to see the empirical returns), and $\theta \leftarrow \theta + \alpha\frac{1}{n\sigma}\sum_{i=1}^n F_i\varepsilon_i$ completes the update in parameter θ. This method is extremely simple and easy to implement but produces nice numerical results.

Generally, $F(\theta)$ is not smooth in θ (e.g., discrete action, deterministic policy) so a noise ε is required for smoothing. Policy gradient methods introduce noise in the action space while ES introduce noise directly in the parameter space. As a result, ES can deal with games with long episodes and actions with long-lasting effects efficiently, in which case the variance of the policy gradient estimators are larger. Since policy gradient methods sample actions independently, the variance scales with T (REINFORCE update contains $\nabla_\theta \pi_\theta(A_t|S_t)$), while ES method shows no dependence on the time horizon T. However, ES suffers from high intrinsic dimensionality of the parameter space due to $\nabla_\theta \mathbb{E}F(X) = \mathbb{E}_{\varepsilon\sim N(0,I)}\left[\frac{F(\theta+\sigma\varepsilon)-F(\theta)}{\sigma}\varepsilon\right]$ essentially being a finite difference. Despite this restriction, ES is very efficient since it only requires forward-propagation of NN but does not do any back-propagation, which is the most time-consuming part in training a deep learning model.

Model-based RL and Planning

Model-based RL is worth investigating due to its ability to generalize to a similar problem (transition dynamics) with a different reward function. Model-based RL typically consists of two parts: approximating the model from the experience and planning. **Planning** refers to the advanced searching strategy trying to figure out the best policy in RL, assuming that one has access to the model of the environment. Here we introduce a model-based RL method called PETS presented in ***Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models by Kurtland Chua et. al.*** It's worth noting that in the context of planning, the reward function r is assumed to be known (for some specific task). If the reward function itself is unknown, one meets with a larger challenge estimating the rewards, which is typically done through inverse reinforcement learning. Estimating the reward function is not trivial since a small perturbation in the reward might result in a large change in the optimal policy.

The first part of the work focuses on approximating the MDP transition dynamics $p(s'|s,a)$ with a certain amount of experience provided. The prediction of dynamics could be done through Gaussian process, neural network etc., and the selection is crucial for the algorithm. Gaussian process works well in the low data regime (when one does not have much experience) but does not work well in high data regime due to the lack of expressivity, and the incapability scaling with dimensionality. On the other hand, NN works well in the high data regime but often overfits in the low data regime. As a result, the uncertainty in the model approximation is decomposed into two parts: aleatoric (intrinsic in the system) and epistemic (lack of data) uncertainty. To model both uncertainty, the author uses the ensemble of probabilistic NNs.

Probabilistic NN refers to an NN that takes in the state s and action a and outputs the parameter of the distribution of $s' \sim p(\cdot|s,a)$, denoted as $\mu_\theta(s,a)$ and $\Sigma_\theta(s,a)$ where θ is the NN parameter. This distribution is typically selected as a Gaussian.

Remark. *This matches the Gaussian parameterization of the policy on continuous action space. In real implementation, the NN outputs the mean and the log-variance of the Gaussian while an exponential acts on the log-variance to provide the variance (to guarantee that variance is positive). For out-of-distribution inputs (experience that has not yet been learnt by the NN), the variance always shrinks to zero or explodes due to the lack of information so a truncation is typically used.*

The loss of the NN shall be naturally designed to encourage the alignment of the NN-generated distribution and the real experience. As a result, the loss is constructed based on a maximum likelihood criterion. Denote $f_\theta(s'|s,a)$ as the likelihood seeing next state s' based on current state s and action a from the NN, the loss is

$$-\sum_t \log f_\theta(S_{t+1}|S_t, A_t) \tag{570}$$

based on which θ is updated. This probabilistic NN captures the aleatoric uncertainty. Now the MDP dynamics is modelled as a Gaussian, which is typically not the case since Gaussian is unimodal. To create a multimodal distribution, the mixture of Gaussian is a good idea, that's why an ensemble of NNs are constructed. Set up B probabilistic NNs f_{θ_b} where $b \in \{1, ..., B\}$. The input of each NN is a bootstrapped (sample with replacement, a way to generate more data in statistics) set of experience and the MDP dynamics is modelled by the mixture $f_\theta = \frac{1}{B}\sum_b f_{\theta_b}$. This ensemble setting models the epistemic uncertainty by generating more data based on what we currently have. That concludes our first task learning the MDP dynamics.

When it comes to planning, i.e., figure out the optimal policy based on the model we have derived, multiple approaches exist. One can directly parameterize the policy or parameterize the value function as what we have done in DP. Here we take the **model predictive control (MPC)** approach. The idea of MPC is to sample a sequence of actions for multiple times, evaluate the action sequence through "simulation" (using the reward function and the dynamics we have learnt), and pick the best current action to take. To be specific, we need T_P as the prediction horizon in planning (not the overall time horizon of the game). Whenever an action sequence $A_{t:t+T_P} = (A_t, ..., A_{t+T_P})$ is sampled from a certain distribution, we simulate using $S_{t+1} \sim f_\theta(S_t, A_t), S_{t+2} \sim f_\theta(S_{t+1}, A_{t+1})$ until S_{t+T_P}. The sum of reward is calculated $\sum_{\tau=t}^{t+T_P} r(S_\tau, A_\tau)$ to evaluate the action sequence and we pick the best action sequence $A^*_{t:t+T_P}$. The first action A^*_t is taken currently and we proceed to the next time step.

When simulating the state trajectory, we use particle-based propagation. This means that we will maintain p state particles S^p_t, with each particle propagated by $S^p_{t+1} \sim f_{\theta_{b(p,t)}}(S^p_t, A_t)$. In other words, the p-th particle propagates under the MDP dynamics corresponding to the $b(p,t)$-th NN, where $b(p,t)$ is an index depending on particle index p and time t. Most often, such $b(p,t)$ does not depend on p, which is the so-called trajectory sampling∞ (**TS**∞) criterion. This sampling criterion enjoys a nice separable decomposition of aleatoric and epistemic uncertainty. In simple words, assign each particle p with a NN-approximated MDP dynamics indexed by b and never change the correspondence during a trial.

Algorithm 23 Probabilistic Ensemble Trajectory Sampling (PETS): model-based MPC

```
Input: $B$ probabilistic networks $\{f_{\theta_b}\}$, an experience buffer, $T_P$ as prediction horizon, $N_{\text{sample}}$ as number of action
    sequence sampled
 1: Collect experience and fill into the experience buffer
 2: repeat
 3:     Update all $\theta_b$ based on $B$ bootstrapped experience buffer and the loss $-\sum_t \log f_{\theta_b}(S_{t+1}|S_t, A_t)$
 4:     Initial state $S_0$
 5:     for $t = 0, 1, 2, ..., T$ do
 6:         for Each action sequence $A_{t:t+T_P}$ sampled from CEM planner (loop for $N_{\text{sample}}$ times) do
 7:             Propagate $P$ state particles $\left\{(S^p_t, ..., S^p_{t+T_P})\right\}_p$ following TS$\infty$
 8:             Evaluate each action sequence with $\sum_{\tau=t}^{t+T_P} \frac{1}{P}\sum_p \sum r(S^p_\tau, A_\tau)$
 9:             Update parameters of the CEM planner
10:         end for
11:         Take action $A^*_t$ from the optimal action sequence, observe reward $R_t$ and the next state $S_{t+1}$
12:         Record the experience in the buffer
13:     end for
14: until Enough number of iteration is done
Output: Approximated MDP dynamics $\{f_{\theta_b}\}$ and a planner to generate actions
```

Another problem arises in the sampling of action sequence. Obviously, sampling from any fixed distribution is not a good choice since we want to use the information from the experience to sample actions from those distributions that are more likely to generate nice actions. A popular planner to use in this context is called **the CEM planner**, which is constructed based on the CEM (cross-entropy minimization) optimization method. We won't mention the details here but the intuition is to update the parameters of the sampling distribution such that its difference from the optimal sampling distribution is minimized. This can be done easily for exponential family distributions and in our context, we take the sampling distribution as Gaussian and update the parameter based on the top several action sequences with the highest returns. The CEM planner is wrapped up in torchrl and turns out to greatly augment the performance of the planning. One can refer to Algorithm 4.1 in ***The Cross-Entropy Method for Optimization by Zdravko I. Botev et. al.*** for the details. Keep in mind that whenever there are some complicated (NP-complete) combinatorial optimization problems (e.g. knapsack, SAT, network planning) or adaptive sampling demands in planning, CEM is always good to use.

Combining all the details together, we get the PETS method, with complete details shown in Alg. 23. PETS has better performance level then state-of-the-art model-free RL methods (like PPO, SAC). This fix the issue in asymptotic performance of previous model-based methods. Through numerical experiments, it has been verified that the probabilistic NN and the ensemble structure are both necessary for the increase in the performance. This means that capturing both sources of uncertainty is crucial for the learning.

Inverse Reinforcement Learning (IRL)

The Framework and Fundamental Methods of IRL

Before formulating IRL, let's briefly talk about the motivation. When we were kids, we learnt things from imitating adults' behavior. A similar concept in RL is called apprenticeship learning or imitation learning, which enables an agent to learn from an expert's behavior. To conduct imitation learning, a natural requirement is that we can figure out the reward function that results in expert's policy, which provides motivation for IRL. Nevertheless, IRL can help interpret why human behave in a certain pattern, and possibly such a reward function can be used to train an AI that has human intelligence (e.g. ethical rules/emotion).

In the context above, we have been thinking about how to figure out the optimal policy based on a given environment and it turns out that various methods have their own subtleties. However, IRL is considered a harder task, especially for large games, in the sense that two very different reward functions can share the same optimal policy (e.g. different by a constant multiple) while two very similar reward functions can have very different optimal policy (e.g. adding perturbation on critical state-action pairs). The sensitivity of optimal policy w.r.t. reward functions make it hard to get a reward function that is robust and useful. Moreover, the same optimal policy can be induced by a family of different reward functions, including trivial ones which one never wants to use in reality. It turns out that subjective restrictions (according to some criteria) are required to ensure that the output reward function is something meaningful.

The first paper considering IRL problem is the ***Algorithms for Inverse Reinforcement Learning by Andrew Y. Ng and Stuart Russell***. We introduce the basic formulations proposed in the paper, but in a clearer manner to avoid confusions.

The investigation starts from the tabular MDP setting when one is given the deterministic optimal policy π and wants to figure out restrictions on the reward function r. The reward function $r(s)$ is simplified to be non-randomized and only depends on the state s. The following lemma is a direct consequence of MDP theory.

Lemma 14. *Given optimal policy π, the reward function $r = r(s)$ must satisfy*

$$\forall a, (P_\pi - P_a)(I - \gamma P_\pi)^{-1} r \geq 0 \tag{571}$$

written in terms of vectors and matrices. Here the rows of P_π consist of $P_{\pi(s_1)}(s_1), ..., P_{\pi(s_n)}(s_n)$ and P_a is the state transition matrix given action a.

Proof. From policy improvement theorem, π is optimal policy iff $\forall s, \pi(s) \in \arg\max_a q_\pi(s,a)$. In other words, $\forall s, \forall a, q_\pi(s,\pi(s)) \geq q_\pi(s,a)$. Representing q_π in terms of v_π, the condition is

$$\forall s, \forall a, \sum_{s'} p(s'|s,\pi(s)) v_\pi(s') \geq \sum_{s'} p(s'|s,a) v_\pi(s') \tag{572}$$

Denote P_a as the state transition matrix given action a and $P_a(s)$ as the certain row of P_a that corresponds to the transition from state s. Then $\forall s, \forall a, P_{\pi(s)}(s) \cdot v_\pi \geq P_a(s) \cdot v_\pi$ where the value function is written as a column vector and the sum is written as a dot product. The inequality is in the sense of component-wise inequality.

Notice that v_π has a compact representation in the sense of the Bellman consistency equation that $\forall s, v_\pi(s) = r(s)+\gamma\sum_{s'} p(s'|s,\pi(s))v_\pi(s')$, i.e., $\forall s, v_\pi(s) = r(s)+\gamma\sum_{s'} P_{\pi(s)}(s)\cdot v_\pi$. Written in vector notations, $v_\pi = r+\gamma P_\pi\cdot v_\pi$ where the rows of P_π consist of $P_{\pi(s_1)}(s_1), ..., P_{\pi(s_n)}(s_n)$. The representation of state value function is thus given by $v_\pi = (I-\gamma P_\pi)^{-1}r$. At this point, we merge results to derive the condition $\forall s, \forall a, [P_\pi(s) - P_a(s)]^T(I-\gamma P_\pi)^{-1}r \geq 0$. Further simplification provides

$$\forall a, (P_\pi - P_a)(I-\gamma P_\pi)^{-1}r \geq 0 \tag{573}$$

concludes the proof. □

The simple calculation tells us the characterization of the solution set of the reward function. Unfortunately, a lot of possible r satisfies this inequality, including the trivial reward function $r\equiv 0$, which one does not hope to see since it contains no information of the incentive contained in the optimal policy.

The remedy is called the **maximum margin principle** as a subjective opinion forced onto the reward function. The motivation of this principle is to favor those reward functions that make any single deviation from the optimal policy π as costly as possible. For example, if one is playing chess and can transit to two states, s_1 that leads him to a loss and s_2 that leads him to a win. The trivial reward $r\equiv 0$ generates the same reward no matter which state the player goes to, while a reward function $r'(s_1) = 0, r'(s_2) = 1$ penalizes the player by 1 if the player deviates from the best state s_2. Maximum margin principle aims to maximize this penalty such that the reward function provides enough information distinguishing different incentives contained in the optimal policy.

The principle is formalized in the following way. Let $q_\pi(s,\pi(s))$ denote the quality of the best action at state s and $\max_{a\neq\pi(s)} q_\pi(s,a)$ the quality of the second-best action. We want to maximize $\sum_s[q_\pi(s,\pi(s)) - \max_{a\neq\pi(s)} q_\pi(s,a)]$, which is the sum of the quality margin for each state.

Remark. *One might be wondering why we are using value function to evaluate the performance of the reward function we learn in IRL. This is because calculating the difference in reward function is impossible since we don't know the true underlying reward function and a small change in reward function might correspond to a large change in the optimal policy, as mentioned above. Besides, value function is easy to estimate from experience.*

Another perspective one can adopt is the sparsity of reward function. Many games has sparse reward function that only distributed non-zero reward at the terminal of the episode. The sparsity can be easily encouraged by adding an ℓ_1-regularization term. As a result, the simplest IRL method for tabular MDP has the following objective

$$\max_{r\in\mathbb{R}^n}\sum_{i=1}^n[q_\pi(s_i,\pi(s_i)) - \max_{a\neq\pi(s_i)} q_\pi(s_i,a)] \tag{574}$$

with $q_\pi(s_i,\pi(s_i)) - \max_{a\neq\pi(s_i)} q_\pi(s_i,a) = \min_{a\neq\pi(s_i)}[P_{\pi(s_i)}(s_i) - P_a(s_i)]\cdot v_\pi = \min_{a\neq\pi(s_i)}[P_\pi(s_i) - P_a(s_i)]\cdot v_\pi$ from the Bellman consistency equation. As a result, we get the following constrained optimization problem:

$$\max_{r\in\mathbb{R}^n}\sum_{i=1}^{n}\min_{a\neq\pi(s_i)}[P_\pi(s_i)-P_a(s_i)]^T(I-\gamma P_\pi)^{-1}r-\lambda||r||_1 \tag{575}$$

$$\begin{cases}\forall a, (P_\pi-P_a)(I-\gamma P_\pi)^{-1}r\geq 0\\ \forall i, |r_i|\leq r_{\max}\end{cases} \tag{576}$$

with regularization intensity $\lambda\geq 0$. The reward function is typically assumed to be bounded by $r_{\max}$. Clearly, this **IRL for tabular MDP** is a linear programming (LP) optimization problem in r so it can be solved efficiently.

That concludes the IRL method for tabular MDP but practically we often have **a large state space**. Naturally, function approximation is introduced to approximate state value function v_π. Assuming $s\in\mathbb{R}^n$, the reward function $r:\mathbb{R}^n\to\mathbb{R}$ is assumed to have the structure

$$r(s)=\sum_{j=1}^{d}\alpha_j\phi_j(s) \tag{577}$$

where ϕ_j are known basis functions and α_j as weights to be optimized. Denote the value function v_π as the one induced by the reward function r. At this point, we re-do the procedures in the lemma above. **π is the optimal policy iff** $\forall s,\forall a, q_\pi(s,\pi(s))\geq q_\pi(s,a)$, iff

$$\forall \mathbf{s},\forall \mathbf{a}, \mathbb{E}_{\mathbf{S'\sim P_\pi(s)}}\mathbf{v}_\pi(\mathbf{S'})\geq \mathbb{E}_{\mathbf{S'\sim P_a(s)}}\mathbf{v}_\pi(\mathbf{S'}) \tag{578}$$

However, under a large state space, this condition results in infinitely many constraints. Numerically, we directly samples a large finite subset of states denoted as S_0 and only check the constraint for the states within S_0. Instead of a hard constraint, we would like to replace with a soft one adding penalty when the condition above is violated. We get the following constrained optimization problem:

$$\max_{\alpha_1,\dots,\alpha_d}\sum_{s\in S_0}\min_{a\neq\pi(s)}p(\mathbb{E}_{S'\sim P_\pi(s)}v_\pi(S')-\mathbb{E}_{S'\sim P_a(s)}v_\pi(S')) \tag{579}$$

$$s.t.\ \forall i, |\alpha_i|\leq 1 \tag{580}$$

where $p(x)=\begin{cases}x & x\geq 0\\ 2x & x<0\end{cases}$ adds extra penalty for the violation of the soft constraint. Although this method can actually run for large state space, there is expressivity issue for the reward function due to the a priori structure proposed (e.g. no way to encourage sparsity since it's pre-determined by the basis functions). Notice that v_π contains $\alpha_1,...,\alpha_d$ due to their appearance in the reward function. This is still an LP problem and can be solved efficiently.

IRL from Experience

Similar to the transition from model-based RL methods to model-free RL methods, in IRL the optimal policy is not always explicitly given, but with the experience following the optimal policy to be observable (e.g., a human expert may not be able to explain how he succeeds in doing something, but he can definitely show his skill doing the work). Similarly, the MDP dynamics is always too complicated to construct in practice. That's why we have to consider the IRL from experience when the model is not available.

Algorithm 24 IRL from Experience

```
Input: Function basis $\phi_1(s), ..., \phi_d(s)$ for the reward function, a simulator, a given optimal policy $\pi$
1: Initial parameters $\alpha_1, ..., \alpha_d$, initial policy $\pi_1$
2: for k=1,...,K do
3:     Simulate under policy $\pi_1, ..., \pi_k$ to get $M$ episodes of experiences $\left\{(S_0^{m,i}, ..., S_T^{m,i})\right\}$ for each policy $\pi_i$
4:     Set up Monte Carlo estimates for the value function $V_{\pi_i}(s_0) = \frac{1}{M}\sum_{m=1}^M \sum_{t=0}^\infty \gamma^t \sum_{j=1}^d \alpha_j \phi_j(S_t^{m,i})$
5:     Solve the following LP problem to get the new set of parameters $\alpha_1, ..., \alpha_d$:
           $$\max_{\alpha_1,...,\alpha_d} \sum_{i=1}^k p(V_\pi(s_0) - V_{\pi_i}(s_0)) \quad (581)$$
           $$s.t.\ \forall i, |\alpha_i| \leq 1 \quad (582)$$
6:     Call a model-free RL method to solve for the optimal policy $\pi_{k+1}$ given the simulator and the new reward function $r(s) = \sum_{j=1}^d \alpha_j \phi_j(s)$
7: end for
Output: $\alpha_1, ..., \alpha_d$ such that $r(s) = \sum_{j=1}^d \alpha_j \phi_j(s)$ is the reward function
```

Sticking to the linear function approximation setting, $r(s) = \sum_{j=1}^d \alpha_j \phi_j(s)$ is the reward function and our task is to update the parameters α_j. Assume, WLOG that the initial state is deterministic, denoted $S_0 = s_0$. For simplicity, we assume the access to a simulator that generates experiences according to a given policy. Provided with M episodes of experiences $\{(S_0^m, S_1^m, ..., S_T^m)\}_m$, Monte Carlo approximates the value function of any given policy π_k that

$$V_{\pi_k}(s_0) = \frac{1}{M}\sum_{m=1}^M \sum_{t=0}^\infty \gamma^t r(S_t^m) = \frac{1}{M}\sum_{m=1}^M \sum_{t=0}^\infty \gamma^t \sum_{j=1}^d \alpha_j \phi_j(S_t^m) \tag{583}$$

as a linear function in α_j. With the help of the value function, given policy $\pi_1, ..., \pi_k$, the given optimal policy π is expected to satisfy (by definition)

$$V_\pi(s_0) \geq V_{\pi_i}(s_0), \forall i \in \{1, ..., k\} \tag{584}$$

with the margin (as the difference in the quality of π and π_i) $V_\pi(s_0) - V_{\pi_i}(s_0)$ to be preferably maximized. By adopting a soft constraint instead of a hard one, we formalize IRL from experience as the following optimization problem:

$$\max_{\alpha_1,...,\alpha_d} \sum_{i=1}^{k} p(V_\pi(s_0) - V_{\pi_i}(s_0)) \tag{585}$$

$$s.t.\ \forall i, |\alpha_i| \leq 1 \tag{586}$$

where p is defined similar to above, adding extra penalty when $V_\pi(s_0) \geq V_{\pi_i}(s_0)$ is violated. This step enables us to get a new parameterized reward function r based on the policy $\pi_1, ..., \pi_k$, and it's still an LP problem.

After updating the reward, we shall add extra policies that are closer to the optimal policy. With the help of the simulator and the reward function, this problem is just a normal RL problem so we can solve for the optimal policy under the updated reward function and denote it by π_{k+1}. The algorithm proceeds iteratively until enough updates on the reward function have been conducted.

The complete details of this algorithm are presented in Alg. 24. This IRL algorithm does not require knowledge of the model and the explicit form of the optimal policy, and it's still constructed based on the maximum margin principle. This is the fundamental method in apprenticeship learning/imitation learning since it learns an optimal policy from the experience of an expert.

Maximum Causal Entropy IRL

We refer the readers to the paper ***Generative Adversarial Imitation Learning by Jonathan Ho and Stefano Ermon*** for more details. This paper is very well-organized and is one of the most important works in IRL. I recommend reading it if you are interested in IRL.

The paper starts from providing a new perspective understanding what **maximum causal entropy IRL** (MCE IRL) is doing. We stick to the original setting in the paper, considering the minimization of cost $c(s, a)$ instead of the maximization of reward $r(s, a)$ without essential difference. The paper considers the most general setting where c could be any function in state and action, not necessarily having the parameterized form assumed above. The maximum causal entropy IRL aims to solve the following optimization problem given the expert's policy π_E through experiences:

$$\max_c \left[\min_\pi \{-H(\pi) + \mathbb{E}_\pi c(S, A)\} - \mathbb{E}_{\pi_E} c(S, A)\right] \tag{587}$$

where $H(\pi) = -\mathbb{E}_\pi \sum_{t=0}^\infty \gamma^t \log \pi(A_t|S_t)$ is the discounted causal entropy of policy π and the expectation over (S, A) are taken w.r.t. ρ^π, the on-policy distribution under policy π, i.e., $\mathbb{E}_\pi c(S, A) = \mathbb{E}_\pi \sum_{t=0}^\infty \gamma^t c(S_t, A_t)$.

Remark. *The causal entropy $H(A_{0:\infty}|S_{0:\infty})$ is defined as the entropy in the distribution of actions $A_0, A_1, ...$ given the whole state trajectory $S_0, S_1, ...$, which is equivalent to saying $H(A_{0:\infty}|S_{0:\infty}) = -\mathbb{E}_\pi \log \prod_{t=0}^\infty \pi(A_t|S_t) = -\mathbb{E}_\pi \sum_{t=0}^\infty \log \pi(A_t|S_t)$. For this to be well-defined in the infinite time horizon, the discount rate is added.*

Remark. *The interpretation of MCE IRL is important to keep in mind. The term $\min_\pi \{-H(\pi) + \mathbb{E}_\pi c(S, A)\}$ is the minimum expected cost with entropy bonus. MCE IRL aims to maximize the gap between this entropy-regularized minimum expected cost and the true minimum expected cost reached by the expert. In other words, the policy that assigns probability masses as evenly as possible is preferred. This framework shares the same idea as the MaxEnt RL.*

As a result, the IRL procedure is defined as a mapping $\pi_E \mapsto \tilde{c}$ that maps the given expert's policy to the approximated cost function. Conversely, the RL procedure is defined as a mapping $RL(c) = \arg\min_\pi \{-H(\pi) + \mathbb{E}_\pi c(S, A)\}$ that maps a cost function c to the optimal policy under MaxEnt RL. The analysis starts from adding a regularization term $\psi(c)$ to the objective of MCE IRL, i.e.,

$$IRL_\psi(\pi_E) = \arg\max_c \left[-\psi(c) + \min_\pi \{-H(\pi) + \mathbb{E}_\pi c(S, A)\} - \mathbb{E}_{\pi_E} c(S, A)\right] \tag{588}$$

The regularizer ψ maps a cost function to a real number and is assumed to be a closed proper convex function. The main theoretical result of this paper is the following theorem. Since imitation learning is typically first updating the cost function (IRL) and then updating the policy (RL), the following theorem tries to figure out what one gets after applying those two steps in a row.

Theorem 18. *Denote $\rho^\pi(s, a) = \sum_{t=0}^\infty \gamma^t \mathbb{P}_\pi (S_t = s, A_t = a)$ as the on-policy measure (a constant multiple difference* *from a probability measure) under policy π, and ψ^* the Frenchel conjugate of ψ, then*

$$RL \circ IRL_\psi(\pi_E) = \arg\min_\pi \{-H(\pi) + \psi^*(\rho^\pi - \rho^{\pi_E})\} \tag{589}$$

The key insight here is that ψ-regularized MCE IRL is essentially finding a policy π whose on-policy measure matches with that of the expert, evaluated under ψ^* and a certain choice of ψ recovers the IRL algorithms in the previous literature. To be concise, **IRL is a dual of the on-policy measure matching problem**.

To be more specific and provide a proof for the theorem, we first notice that policy π and on-policy measure ρ^π are two sides of the same coin, implied by the lemma below.

Lemma 15. *Let $\mathscr{D} = \left\{\rho : \rho \geq 0, \sum_a \rho(s,a) = p_0(s) + \gamma \sum_{s',a} p(s|s',a)\rho(s',a), \forall s\right\}$ be a collection of probability measures on $\mathscr{S} \times \mathscr{A}$ where p_0 is the initial state distribution of the MDP. Then there exists a one-to-one correspondence between policy π and on-policy measure $\rho \in \mathscr{D}$.*

Proof. First prove that the on-policy measure induced by any policy π must be in $\mathscr{D}$.

$$\rho^\pi(s,a) = \sum_{t=0}^\infty \gamma^t \mathbb{P}_\pi(S_t = s, A_t = a) = p_0(s)\pi(a|s) + \gamma\pi(a|s)\sum_{t=0}^\infty \gamma^t \mathbb{P}_\pi(S_{t+1} = s), \ \forall s \in \mathscr{S}, \forall a \in \mathscr{A} \tag{590}$$

sum both sides w.r.t. a to get

$$\sum_a \rho(s,a) = p_0(s) + \gamma\sum_{t=0}^\infty \gamma^t \mathbb{P}_\pi(S_{t+1} = s) = p_0(s) + \gamma\sum_{t=0}^\infty \gamma^t \sum_{s',a} p(s|s',a)\mathbb{P}_\pi(S_t = s', A_t = a) \tag{591}$$

$$= p_0(s) + \gamma\sum_{s',a} p(s|s',a)\sum_{t=0}^\infty \gamma^t \mathbb{P}_\pi(S_t = s', A_t = a) = p_0(s) + \gamma\sum_{s',a} p(s|s',a)\rho^\pi(s',a) \tag{592}$$

so that $\rho^\pi \in \mathscr{D}$. Obviously, the constraint in $\mathscr{D}$ is just a fundamental consistency condition for any on-policy measure.

Conversely, if given $\rho \in \mathscr{D}$, consider policy $\pi_\rho(a|s) = \frac{\rho(s,a)}{\sum_b \rho(s,b)}$. Let's prove that the on-policy measure under π_ρ is exactly ρ itself and that such π_ρ is the only policy that induces ρ. Denote ρ^{π_ρ} as the on-policy measure induced by π_ρ, then

$$\rho^{\pi_\rho}(s,a) = \pi_\rho(a|s)\sum_{t=0}^\infty \gamma^t \mathbb{P}_{\pi_\rho}(S_t = s) = \pi_\rho(a|s)\sum_b \rho^{\pi_\rho}(s,b) \tag{593}$$

as a result, $\frac{\rho^{\pi_\rho}(s,a)}{\sum_b \rho^{\pi_\rho}(s,b)} = \frac{\rho(s,a)}{\sum_b \rho(s,b)}$ for $\forall s \in \mathscr{S}, a \in \mathscr{A}$ so ρ and ρ^{π_ρ} at most differ by a constant, which must be 1 since $\sum_{s,a} \rho(s,a) = \sum_{s,a} \rho^{\pi_\rho}(s,a) = 1 + \gamma$ from the constraint in $\mathscr{D}$. This proves $\rho = \rho^{\pi_\rho}$.

If two policies π_1, π_2 induce the same on-policy measure $\rho \in \mathscr{D}$, then $\rho^{\pi_1} = \rho^{\pi_2} = \rho$, which implies $\pi_1(a|s) \propto \rho^{\pi_1}(s,a) = \rho(s,a), \pi_2(a|s) \propto \rho^{\pi_2}(s,a) = \rho(s,a)$ from the proof above. As a result, $\pi_1 = \pi_2$. □

Proof of the theorem above. This proof is an application of Von-Neumann's minimax theorem.

Denote $\tilde{c} \in IRL_\psi(\pi_E), \tilde{\pi} \in RL(\tilde{c})$ and $\pi_A \in \arg\min_\pi \{-H(\pi) + \psi^*(\rho^\pi - \rho^{\pi_E})\}$. Due to the definition of Frenchel conjugate,

$$\pi_A \in \arg\min_{\pi}\max_{c}\left\{-H(\pi)+\sum_{s,a}c(s,a)[\rho^{\pi}(s,a)-\rho^{\pi_E}(s,a)]-\psi(c)\right\} \tag{594}$$

Due to the lemma above, it suffices to show that $\rho^{\pi_A}=\tilde{\rho}$ where $\tilde{\rho}$ is the on-policy measure under $\tilde{\pi}$. Let's define a function $L(\rho,c)$ such that

$$L(\rho,c)=-\overline{H}(\rho)-\psi(c)+\sum_{s,a}c(s,a)[\rho(s,a)-\rho^{\pi_E}(s,a)] \tag{595}$$

with $\overline{H}(\rho)=-\sum_{s,a}\rho(s,a)\log\frac{\rho(s,a)}{\sum_{a'}\rho(s,a')}$ to be the entropy of π_ρ. When ρ is the on-policy measure induced by π, we notice that

$$\min_{\pi}\max_{c}\left\{-H(\pi)+\sum_{s,a}c(s,a)[\rho^{\pi}(s,a)-\rho^{\pi_E}(s,a)]-\psi(c)\right\}=\min_{\rho}\max_{c}L(\rho,c) \tag{596}$$

This is because $H(\pi_\rho)=-\mathbb{E}_{\rho^\pi}\log\pi_\rho(A|S)=\overline{H}(\rho)$ due to the correspondence in the lemma.

By Von-Neumann's minimax theorem, since $\mathscr{D}$ is compact and convex, $\mathbb{R}^{|\mathscr{S}|\times|\mathscr{A}|}$ is convex, and $L(\rho,\cdot)$ is convex in ρ ($\overline{H}(\rho)$ is concave in ρ) while $L(\cdot,c)$ is concave in c ($\psi(c)$ is convex in c),

$$\min_{\rho}\max_{c}L(\rho,c)=\max_{c}\min_{\rho}L(\rho,c) \tag{597}$$

Since $\tilde{c}\in\arg\max_c\min_\rho L(\rho,c)$ and $\rho^{\pi_A}\in\arg\min_\rho\max_c L(\rho,c)$, we know that $(\rho^{\pi_A},\tilde{c})$ is the saddle point. Due to minimax duality, $\rho^{\pi_A}\in\arg\min_\rho L(\rho,\tilde{c})$. However, by definition, $\tilde{\rho}\in\arg\min_\rho L(\rho,\tilde{c})$. Since $L(\rho,\cdot)$ is strictly convex in ρ, the minimum is uniquely attained, which proves $\rho^{\pi_A}=\tilde{\rho}$. □

Remark. *Throughout the proof, we establish the $\overline{H}$ just to write the objective as a function in the on-policy measure ρ instead of the policy π. Despite the difference on the object H and $\overline{H}$ acts on, they are exactly the same under the correspondence mentioned in the lemma above.*

At last, we compute as an example when $\psi\equiv C$ is constant. It's clear that

$$\psi^*(\rho^\pi-\rho^{\pi_E})=\max_{c}\left\{\sum_{s,a}c(s,a)[\rho^{\pi}(s,a)-\rho^{\pi_E}(s,a)]\right\}-C=\begin{cases}-C & \rho^\pi=\rho^{\pi_E}\\ +\infty & \text{else}\end{cases} \tag{598}$$

As a result,

$$RL\circ IRL_\psi(\pi_E)=\arg\min_{\pi:\rho^\pi=\rho^{\pi_E}}\{-H(\pi)\}=\arg\max_{\pi:\rho^\pi=\rho^{\pi_E}}H(\pi) \tag{599}$$

provides the policy π with the maximum causal entropy that matches the on-policy measure of the expert, i.e. $\rho^{\tilde{\pi}}=\rho^{\pi_E}$. In this case, IRL is exactly an on-policy measure matching problem. If we take a look at L at this moment,

$$L(\rho, c) = -\overline{H}(\rho) - C + \sum_{s,a} c(s,a)[\rho(s,a) - \rho^{\pi_E}(s,a)] \tag{600}$$

it's linear in c so we interpret the c as the Lagrange multiplier. By doing this, L matches the Lagrange multiplier function of the following optimization problem:

$$\min_{\rho\in\mathscr{D}} -\overline{H}(\rho) - C \tag{601}$$

$$s.t.\ \rho(s,a) = \rho^{\pi_E}(s,a) \tag{602}$$

In this case, $\tilde{c}$ is the optimal solution to the dual problem of the one above and strong duality (as a special case of minimax duality) holds.

Remark. *Consider $\mathscr{C} \subset \mathbb{R}^{|\mathscr{S}|\times|\mathscr{A}|}$ as a sub-function space, for which one assumes that the cost function c lives in. Consider $\delta_{\mathscr{C}}(c) = +\infty\cdot\mathbb{I}_{c\notin\mathscr{C}}$ as an indicator function of $\mathscr{C}$ and $\delta^*_{\mathscr{C}}(\rho^\pi - \rho^{\pi_E}) = \max_{c\in\mathscr{C}}\left\{\sum_{s,a} c(s,a)[\rho^\pi(s,a) - \rho^{\pi_E}(s,a)]\right\}$. Let's see what happens if $\psi = \delta_{\mathscr{C}}$.*

$$RL \circ IRL_\psi(\pi_E) = \arg\min_\pi \left\{-H(\pi) + \max_{c\in\mathscr{C}}\left\{\sum_{s,a} c(s,a)[\rho^\pi(s,a) - \rho^{\pi_E}(s,a)]\right\}\right\} \tag{603}$$

and we can identify $\max_{c\in\mathscr{C}}\left\{\sum_{s,a} c(s,a)[\rho^\pi(s,a) - \rho^{\pi_E}(s,a)]\right\} = \max_{c\in\mathscr{C}}\left\{\mathbb{E}_\pi \sum_{t=0}^\infty \gamma^t c(S_t, A_t) - \mathbb{E}_{\pi_E}\sum_{t=0}^\infty \gamma^t c(S_t,A_t)\right\}$ as exactly the objective in traditional imitation learning. This is because we encourage the cost function to maximize the gap between expected cost under π and π_E (maximum margin principle), while RL is applied afterwards to find a policy π to under this new cost function. This is the typical procedure of imitation learning where one first applies IRL to update the cost/reward, then apply RL to learn a new policy (as presented in previous sections). As a result,

$$RL \circ IRL_\psi(\pi_E) = \arg\min_\pi \left\{-H(\pi) + \max_{c\in\mathscr{C}}\left\{\mathbb{E}_\pi \sum_{t=0}^\infty \gamma^t c(S_t, A_t) - \mathbb{E}_{\pi_E}\sum_{t=0}^\infty \gamma^t c(S_t,A_t)\right\}\right\} \tag{604}$$

so entropy-regularized imitation learning restricted on the space of cost function $\mathscr{C}$ is equivalent to first apply MCE IRL with regularizer $\psi = \delta_{\mathscr{C}}$ and then apply MaxEnt RL.

Consequently, the disadvantage of traditional imitation learning is that it cannot recover the expert-like policy if $\mathscr{C}$ is too restrictive (expressivity issue, failure encoding π_E in $\mathscr{C}$). This can be interpreted as $\psi = \delta_{\mathscr{C}}$ being a too hard threshold (non-smoothness) and that ψ does not change as π_E changes.

From the minimax duality, we see that **imitation learning is essentially figuring out the saddle point of $\overline{L}$** by minimizing w.r.t. π and maximizing w.r.t. c. The regularizer ψ plays an important role in determining the smoothness of $\overline{L}$. Ideally, ψ shall depend on π_E so that when the expert's policy changes, the regularizer would automatically adapt to it.

Generative Adversarial Imitation Learning

Motivated by the analysis above, we shall preferably choose a smooth regularizer ψ, hoping that ψ has some practical interpretations and enables the imitation of π_E exactly (unlike the case of the indicator). The choice is inspired by the generative adversarial network (GAN) where the generative network learns the distribution of real data and the adversarial network learns to distinguish the real data distribution from the distribution generated by the generative network. In this case, assume a distinguisher $D : \mathscr{S} \times \mathscr{A} \to (0, 1)$, then hopefully

$$\psi^*(\rho^\pi - \rho^{\pi_E}) = \max_D \{\mathbb{E}_\pi \log D(S, A) + \mathbb{E}_{\pi_E} \log[1 - D(S, A)]\} \tag{605}$$

as the optimal log-loss of the binary classification problem distinguishing state-action pairs generated by π and π_E. Luckily, this ψ^* is well-defined in the sense that there exists $\psi(c) = +\infty \cdot \mathbb{I}_{c\geq 0} + \mathbb{E}_{\pi_E} g(c(S, A)) \cdot \mathbb{I}_{c<0}$ for some function g. Notice that ψ has dependence on π_E, which indicates that the regularizer changes as the expert's policy changes. Moreover, this ψ^* has a correspondence with the Jensen-Shannon divergence

$$D_{JS}(\mathbb{P}, \mathbb{Q}) = D_{KL}\left(\mathbb{P}||\frac{\mathbb{P} + \mathbb{Q}}{2}\right) + D_{KL}\left(\mathbb{Q}||\frac{\mathbb{P} + \mathbb{Q}}{2}\right) \tag{606}$$

that by specifying $D(s, a) = \frac{\rho^\pi(s,a)}{\rho^\pi(s,a)+\rho^{\pi_E}(s,a)}$, we have

$$\mathbb{E}_\pi \log D(S, A) + \mathbb{E}_{\pi_E} \log[1 - D(S, A)] = -2 \log 2 + D_{JS}(\rho^\pi, \rho^{\pi_E}) \tag{607}$$

so $\psi^*(\rho^\pi - \rho^{\pi_E})$ enjoys a nice interpretation as the distance between two on-policy measures. The structure as a distance on the space of on-policy measures makes it possible to imitate the expert policy **exactly** (unlike the case when ψ is an indicator!).

Now that with a selection of ψ^*, let's build up the algorithm with the help of GAN. Clearly, the generative network generates a parameterized policy π to be as close as the expert's policy π_E. On the other hand, the discriminator network learns to distinguish two policies. From the argument above, our objective is to find the saddle point (π, D) (D as maximizer and π as minimizer simultaneously) for

$$L(\pi, D) = -H(\pi) + \mathbb{E}_\pi \log D(S, A) + \mathbb{E}_{\pi_E} \log[1 - D(S, A)] \tag{608}$$

so the discriminator network D_w shall conduct gradient ascent for its parameter w with reward $L(\pi_\theta, D_w)$. Clearly, the entropy term does not involve w and disappears after taking gradient w.r.t. w so the reward is simply $\mathbb{E}_\pi \log D_w(S, A) + \mathbb{E}_{\pi_E} \log[1 - D_w(S, A)]$, with the expectations estimated through experiences. When it comes to experiences, of course they shall be generated under policy π_θ. The parameter θ shall be updated to minimize $L(\pi_\theta, D_w)$ and only two terms: $-H(\pi_\theta)$ and $\mathbb{E}_{\pi_\theta} \log D_w(S, A)$, contain θ. The term

$$\mathbb{E}_{\pi_\theta} \log D_w(S, A) = \mathbb{E}_{\pi_\theta} \sum_{t=0}^\infty \gamma^t \log D_w(S_t, A_t) \tag{609}$$

is actually the expected loss under policy π_θ as if the cost function is $c(s,a) = \log D_w(s,a)$! Smartly, this minimization can be done by calling an existing RL method like TRPO and the gradient $\nabla_\theta H(\pi_\theta)$ can be easily calculated separately through auto-differentiation.

The complete details of the algorithm is provided in Alg. 25.

Algorithm 25 Generative Adversarial Imitation Learning

```
Input: Generative network $\pi_\theta$, discriminator network $D_w$, expert trajectories $\tau_E$
1: Initial parameters $\theta, w$
2: repeat
3:     Simulate trajectories $\tau_\theta$ under policy $\pi_\theta$
4:     Update $w$ through SGA with reward (negative loss) constructed upon $M$ trajectories of $\tau_\theta$ and $N$ trajectories
       of $\tau_E$:
       $$\frac{1}{M}\sum_{m=1}^{M}\sum_{t=0}^{\infty}\gamma^t \log D_w(S_t^{\tau_\theta^{(m)}}, A_t^{\tau_\theta^{(m)}}) + \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{\infty}\gamma^t \log[1 - D_w(S_t^{\tau_E^{(n)}}, A_t^{\tau_E^{(n)}})]$$
5:     Update $\theta$ using TRPO rule (KL-constrained natural gradient step) with cost function $c(s,a) = \log D_w(s,a)$
       and also update $\theta$ towards the direction $\nabla_\theta H(\pi_\theta)$
6: until Enough iterations are done
Output: $\pi_\theta$ as the imitation policy
```

The generative adversarial imitation learning is one of the state-of-the-art methods in imitation learning. It is sample efficient in terms of the expert's data and is a model-free method. To mention once more, the key insight lies in the saddle point interpretation of first applying IRL and then applying RL. This saddle point interpretation exactly aligns with the motivation of GAN.

Transfer RL

So far, we have seen that RL can learn the optimal policy based on the complete setting of a game or even only based on the expert's behavior (imitation learning). However, it's generally recognized that the training process in RL is costly. Motivated by this, people think of generalizing the learnt policy to work on similar tasks, which is called transfer RL, i.e., transferring existing knowledge on a task to another similar task. We hope that such transfer would directly work for the new task, or at least greatly accelerate the training. It's worth noting that transfer RL also includes meta-RL, which is a "learn-to-learn" task, i.e. learning a strategy to learn to perform the task in a faster way. This aligns with reality that kids have to attend schools to form good learning habit in order to help them learn faster and better in the future.

Modular Neural Network Policy

The method is from the paper ***Learning Modular Neural Network Policies for Multi-Task and Multi-Robot Transfer by Coline Devin et. al.*** We consider the following setting that there are multiple agents (robots) and multiple tasks. Those agents and tasks share similarities but are not exactly the same, e.g. robots have different number of joints but share a similar architecture, tasks are like opening the drawer and pushing a block. Assume that there are R robots indexed by r and K tasks indexed by k. As a result, there are $W = KR$ worlds (combinations of agents and tasks) indexed by w but through transfer RL one only needs to train them under some of the worlds and can bring the knowledge into an unseen world.

The modular neural network policy adopts a simple idea splitting the state s_w of world w into the task-specific part $s_{w,\mathscr{T}}$ and the robot-specific part $s_{w,\mathscr{R}}$. The policy for robot r and task k facing state s_w is parameterized as $N(\phi_{w_{rk}}(s_w), \Sigma)$ where ϕ is an NN. This is the common practice parameterizing policy for large action space. The difference of modular neural network policy lies in the construction of ϕ that

$$\phi_{w_{rk}}(s_w) = f_r(g_k(s_{w,\mathscr{T}}), s_{w,\mathscr{R}}) \tag{610}$$

where f_r is the robot-specific module for robot r and g_k is the task-specific module for task k. As a result, f_r, g_k are all organized as NNs so there are altogether $R + K$ modules, which is much less than $W = RK$ (number of modules if one trains each world from scratch). Once all $R + K$ modules have been trained at least for once, any combinations of the modules provides the transfer of the knowledge to an unseen world. In other words, if we face an unseen world where robot r' wants to do task k', then

$$\phi_{w'}(s_{w'}) = f_{r'}(g_{k'}(s_{w',\mathscr{T}}), s_{w',\mathscr{R}}) \tag{611}$$

provides the architecture of the policy network.

When it comes to the details, we point out that the splitting of the state s_w is done based on the interpretation of the state component. For example, robot joint state and sensor readings are considered robot-specific while images, object locations are considered task-specific. During the training, we collect experiences from all the worlds available and form the training loss as the sum of the training loss in each of the world (the specific form of the loss depends on which RL method to use). This is to encourage all the modules to jointly minimize the sum of loss in each world (to work across worlds) instead of improving the performance in each of the world (to work within worlds). At last, since transfer RL is a task of generalization, overfitting is the problem we have to deal with. The number of hidden neurons are restricted and dropout is added to the NN to mitigate overfitting issues.

The modular policy is a very simple strategy for transfer RL, but it serves typically from the perspective of engineering. In many of the virtual games, there is no clear way to distinguish agent-specific state from task-specific state and it's also hard to define the similarity between different tasks.

Progressive Neural Network

The progressive neural network is proposed in the paper ***Progressive Neural Networks by Andrei A. Rusu et. al.*** This transfer RL architecture works for a single agent across multiple tasks. However, the tasks can have arbitrary relationships and need not be similar to each other. This solves the problem of defining the similarity between tasks in order to apply transfer RL. One popular method in transfer RL from previous literature is the finetuning method, which directly takes the previously trained NN, modifies the dimension of the output layer and finetuned through backpropagation. This method might suffers from catastrophic forgetting since the previous learnt policy has been completely destroyed in the finetuning procedure.

Instead, the progressive network keeps the previously learnt policy but conducts the transfer of knowledge by adding more neurons and connecting them with previously trained policies. A progressive network starts with a single column with layers $h_1^{(1)},...,h_L^{(1)}$. The layer $h_i^{(1)}$ has n_i neurons with the number on the superscript as the index of the task and the number on the subscript as the index of the layer for each task. We train this NN to output a parameterized policy for the first task. When a new task is added, we add the second column with layers $h_1^{(2)},...,h_L^{(2)}$. Since we hope to utilize the knowledge on the first task to augment the learning on the second task, $h_i^{(2)}$ receives input from both $h_{i-1}^{(2)}$ and $h_{i-1}^{(1)}$. Importantly, when the training on the second task takes place, the parameters in the first column (for $h_1^{(1)},...,h_L^{(1)}$) are frozen (set as non-trainable) so the policy for the first task is completely preserved. This completely solves the problem of catastrophic forgetting.

In more detail, we hope to formalize the connection between $h_i^{(k)}$ and $h_{i-1}^{(1)},...,h_{i-1}^{(k-1)},h_{i-1}^{(k)}$. This is done through an adapter module formalized as below

$$h_i^{(k)} = \sigma(W_i^{(k)}h_{i-1}^{(k)} + U_i^{(<k)}\sigma(V_i^{(<k)}\alpha_i^{(<k)}h_{i-1}^{(<k)})) \tag{612}$$

where $h_{i-1}^{(<k)} \in \mathbb{R}^{n_{i-1}^{(<k)}}$ is the concatenation of $h_{i-1}^{(1)},...,h_{i-1}^{(k-1)}$, $\alpha_{i-1}^{(<k)} \in \mathbb{R}$ is a scalar for scaling purpose, and $V_i^{(<k)} \in \mathbb{R}^{n_{i-1}\times n_{i-1}^{(<k)}}$ is a projection matrix to ensure the output dimension as n_{i-1}. Here σ is the activation function, and $W_i^{(k)}\mathbb{R}^{n_i\times n_{i-1}}, U_i^{(<k)} \in \mathbb{R}^{n_i\times n_{i-1}}, V_i^{(<k)}, \alpha_i^{(<k)}$ are all trainable parameters. The adapter is organized as another NN with only one hidden layer and no bias to model the nonlinear connection between $h_i^{(k)}$ and $h_{i-1}^{(1)},...,h_{i-1}^{(k-1)},h_{i-1}^{(k)}$.

In simple words, the progressive network is a huge NN with fixed depth L but varying width according to the number of tasks trained. The nonlinear connection between $h_i^{(k)}$ and $h_{i-1}^{(1)},...,h_{i-1}^{(k-1)},h_{i-1}^{(k)}$ is organized as another NN called the adapter. Whenever a new task is being trained, the parameters for all previous trained columns will be frozen to preserve the previously leanrt knowledge.

We remark that this architecture is possible for conducting transfer analysis thanks to the fact that it exactly preserves previously learnt knowledge. One of the analysis is called average perturbation sensitivity injecting Gaussian noises at isolated points in the architecture and measuring the impact of this perturbation on the performance. A significant drop in the performance after adding the perturbation indicates the importance of the layer in transferring the knowledge. Interested readers shall refer to the original paper for the numerical analysis of sensitivity. Generally, it turns out that the transfer of knowledge does take place in progressive neural networks among different modifications for the same Atari game (e.g. change the color of the background, vertical/horizontal flip, noisy environment, etc.).

Meta-RL and RL^2

Meta-learning is understood as "learn to learn", which is motivated by natural observations that animals and humans learn things much faster than an RL agent. This is somewhat due to the fact that animals have prior knowledge on a specific task (e.g. knowledge on the physical structure) and has an intrinsic strategy to learn things instead of only focusing on the rewards. This strategic mindset is what enables animals to learn much faster, especially for complicated tasks. Meta-learning, as a result, is focusing on learning how to learn in a faster way, i.e., learning how to build up better RL methods.

We first introduce the state-of-the-art meta-learning method called RL^2 from the paper ***RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning by Yan Duan et. al.*** The paper adopts the setting of a family of MDPs denoted by $\mathscr{M}$ (a collection of different tasks) and a probability distribution on $\mathscr{M}$ (e.g., the probability we have to deal with such tasks in daily life). The objective is to build up a general RL method that maximizes the average performance for the tasks within $\mathscr{M}$ so that one does not need to design a fast RL method for each specific task.

In this sense, we shall consider a "trial" consisting of n episodes sampled from $\mathscr{M}$. Instead of maximizing the return on a single episode, maximizing the return on a single trial requires the RL method to take into consideration its generality. But how do we represent an RL method? We have parameterized the value function and the policy with NN, but parameterizing the RL algorithm requires us to integrate the structure of sequential decision making into the architecture of the model. The answer is to use recurrent neural network (RNN), as an approximation of a dynamical system, which matches the requirement of an RL algorithm. In detail, we start the first episode in a single trial with state S_0^1, compute the underlying state of RNN at time 0, denoted as h_0^1, using S_0^1. This h_0^1 contains information on the experience up to time 0, so it tells us what action A_0^1 to take as the output. The agent then send the action A_0^1 to the environment to get the reward R_1^1 and the next state S_1^1. Repeatedly, compute the underlying state h_1^1 based on $h_0^1, A_0^1, R_1^1, S_1^1$ and h_1^1 tells us what action A_1^1 to take as the output (since it contains information on the experience up to time 1). Such procedure lasts until the first episode within a single trial ends at time $T-1$. Notice that h_{T-1}^1 is then carried to the second episode, which means that h_0^2 is computed based on h_{T-1}^1 and S_0^2 as the underlying state at time 0 for the second episode. This procedure continues until the end of the trial.

Remark. *The connection between different episodes within a single trial is maintained through the sharing of the underlying state in the RNN. This forces the agent to act differently according to its belief over which MDP it is currently in, resulting in a general and fast RL method across different tasks.*

To clarify, the RNN contains nT neurons denoted $h_0^1, ..., h_{T-1}^1, h_0^2, ..., h_{T-1}^2, ..., h_{T-1}^n$ with the RNN approximated dynamical system as

$$\begin{cases} h_{t+1}^k = f(h_t^k, A_t^k, R_{t+1}^k, S_{t+1}^k) \\ A_{t+1}^k = g(h_{t+1}^k) \\ S_{t+2}^k \sim p(\cdot|S_{t+1}^k, A_{t+1}^k), R_{t+2}^k = r(S_{t+1}^k, A_{t+1}^k) \ (\textbf{MDP}) \end{cases} \tag{613}$$

As a result, all the neurons are connected as a single path (even across different episodes within a single trial), with each neuron h_{t+1}^k having inputs $A_t^k, R_{t+1}^k, S_{t+1}^k$ and output A_{t+1}^k (or a policy from which action is sampled).

With the RNN architecture, the idea of applying RL method aligns with that in RLHF. The learning process itself can be formed as a bandit problem with the "action" as the parameter of the RNN and the "reward" as the return of a single trial (calculated from simulation). This bandit problem can then be solved through some state-of-the-art RL method (e.g., TRPO) to derive the optimal parameter of the RNN. Since this procedure takes a lot of time, it's referred to as the "slow RL" procedure. By carrying out this slow RL procedure, one is able to find the optimal parameter of the RNN, which corresponds to a general "fast RL" method. This explains the title of the paper and the fundamental idea of meta-RL.

From evaluation, this meta-RL method is found to be able to scale to high-dimensional tasks (e.g. state as images), generalize to partially-observable settings (change the input of RNN neuron to O_{t+1}^k, the partial observations of the next state) and perform well for a certain task in a randomly generated environment (e.g., navigate a randomly generated maze). Possible improvements include a better RL method for parameter update of the RNN, and a better architecture under a long time horizon.

Model-agnostic Meta-learning (MAML)

MAML is a general framework that works for any model that updates its parameters θ using gradient descent w.r.t. a loss function. Given a set of tasks $\mathscr{T}$ and a probability distribution $p(\mathscr{T})$ indicating the likelihood solving each task in $\mathscr{T}$, MAML finds a general-purpose parameter θ that works well for tasks following $p(\mathscr{T})$. The idea is to use the loss function to evaluate the performance of parameter θ in solving task $\mathscr{T}_i \sim p(\mathscr{T})$ and update θ such that it minimizes the average loss across different tasks.

To be specific, we are currently using f_θ as the policy network. For a batch of tasks $\mathscr{T}_i \sim p(\mathscr{T})$ drawn from the distribution, the loss function for learning task $\mathscr{T}_i$ is denoted as $L_{\mathscr{T}_i}(f_\theta)$. As a result, one takes the gradient descend step $\theta_i' = \theta - \alpha\nabla_\theta[L_{\mathscr{T}_i}(f_\theta)]$ to learn to do task $\mathscr{T}_i$ better. In order to learn a parameter θ that works generally well for the task distribution $p(\mathscr{T})$, one measures the performance of the updated parameter θ_i' on solving $\mathscr{T}_i$ through $L_{\mathscr{T}_i}(f_{\theta_i'}) = L_{\mathscr{T}_i}(f_{\theta-\alpha\nabla_\theta L_{\mathscr{T}_i}(f_\theta)})$. On average, the loss evaluated at the updated θ_i' is $\sum_{\mathscr{T}_i\sim p(\mathscr{T})} L_{\mathscr{T}_i}(f_{\theta_i'})$ and we want to find a value of θ that minimizes this average loss.

Remark. *θ_i' is believed to show better performance than θ in doing task $\mathscr{T}_i$ but is not guaranteed to show better performance than θ in doing other tasks. This is the main difficulty in meta-learning that different tasks have different value of the parameter θ to achieve the optimal performance. In this sense, MAML uses $L_{\mathscr{T}_i}(f_{\theta_i'})$ to measure the performance of the updated model in solving task $\mathscr{T}_i$. A trade-off is made across all different tasks sampled from the distribution and θ is updated to be the parameter that shows the best average performance.*

Written as an optimization problem, MAML proposes

$$\min_\theta \sum_{\mathscr{T}_i\sim p(\mathscr{T})} L_{\mathscr{T}_i}(f_{\theta-\alpha\nabla_\theta L_{\mathscr{T}_i}(f_\theta)}) \tag{614}$$

There are multiple ways to solve this optimization problem. One of the simplest approaches is to use a gradient descent step

$$\theta \leftarrow \theta - \beta\nabla_\theta\left[\sum_{\mathscr{T}_i\sim p(\mathscr{T})} L_{\mathscr{T}_i}(f_{\theta-\alpha\nabla_\theta L_{\mathscr{T}_i}(f_\theta)})\right] \tag{615}$$

to update θ. This concludes the procedure of MAML, which is simple but effective.

Remark. *This method is called model-agnostic since it does not only apply for RL but also applies for supervised learning tasks like regression/classification. In the sense of regression, the loss function is just the mean-squared error while in the sense of classification, the loss function is just the cross entropy. For any meta-learning tasks that can be organized as a loss function differentiable in θ, MAML can always be applied.*

In the context of RL, we provide more details in terms of MAML. Clearly, f_θ is a policy network parameterized by θ and the loss is organized as the negative return $L_{\mathscr{T}_i}(f_\theta) = -\mathbb{E}_{\pi=f_\theta,p_{\mathscr{T}_i}}\sum_{t=0}^\infty \gamma^t r_{\mathscr{T}_i}(S_t, A_t)$ where $p_{\mathscr{T}_i}$ and $r_{\mathscr{T}_i}$ are the MDP transition kernel and reward function for task $\mathscr{T}_i$. We list the general scheme of MAML for RL as Alg. 26 below.

Remark. *Actually both gradient descend steps in MAML can be replaced with RL methods in the context of meta-RL. The idea is just that θ_i' is an updated version of θ towards solving task $\mathscr{T}_i$ and θ is updated to minimize average loss. For example, one can use REINFORCE update to compute θ_i' and use TRPO to compute the updated θ (similar to that in RL^2, identify θ as the action of a bandit problem).*

Notice that this algorithm requires the simulation under both f_θ and all $f_{\theta_i'}$ due to the fact that policy gradient methods are on-policy.

Algorithm 26 General scheme of Model-agnostic Meta-learning for RL

```
Input: Policy network $f_\theta$, task distribution $p(\mathscr{T})$, hyperparameters $\alpha, \beta > 0$
 1: Initial parameters $\theta$
 2: repeat
 3:     Sample a batch of tasks $\mathscr{T}_i \sim p(\mathscr{T})$
 4:     for each sampled task $\mathscr{T}_i$ do
 5:         Simulate MDP trajectories $\tau_\theta$ under policy $f_\theta$
 6:         Calculate $\theta_i' = \theta - \alpha \nabla_\theta L_{\mathscr{T}_i}(f_\theta)$ based on $\tau_\theta$ (or through RL methods like REINFORCE)
 7:         Simulate MDP trajectories $\tau_i$ under policy $f_{\theta_i'}$
 8:     end for
 9:     Update $\theta = \theta - \beta \sum_{\mathscr{T}_i \sim p(\mathscr{T})} \nabla_\theta L_{\mathscr{T}_i}(f_{\theta_i'})$ based on all the $\tau_i$ (or through RL methods like TRPO)
10: until Enough iterations are done
Output: $\pi_\theta$ as the imitation policy
```