# Notes on RL

Haosheng Zhou

August, 2022

# Contents

**This note refers to the book *Reinforcement Learning: an introduction, by R.S. Sutton and A.G. Barto*. All plots are generated by Python and all codes can be found at https://github.com/Haosheng-Zhou/RL_code.**

# Introduction

The essential point of reinforcement learning (RL) is the engagement of **reward**. The final objective of RL is to achieve the highest possible reward over a period of time. However, greedy strategy that takes the most reward for the time being is always not the best thing to do since short-sighted actions may cause loss in long term. This can be modelled by introducing an **environment** and a set of **actions** that can change the **states** of the environment. Different rewards are then given based on different states and actions. The states, actions and rewards at time $t$ are noted $S_t, A_t, R_t$ as random variables, and $s_t, a_t, r_t$ denote the values of the random variables respectively. In each turn, one chooses an action to take that results in the change of the state and the rewards one would receive in the next turn. The hard point is to enable the computer to think in a long-term sense. Since short-term decisions should be ignored, a **value function** is always needed to indicate the long-term benefits of a certain action at a certain state and time.

The possible problems we have to face in RL include: too large state space (like chess or even continuous state space), how to estimate the value functions (since there's no fixed definition of a value function, it may have to depend on specific tasks), how to exhibit the learning process (what is the model learning, actually it should learn a thing called **policy**, what's a policy and how shall the model learn it?) etc.. As a result, there are quite many problems to solve and let's begin with a very simple example.

# Bandit Problems with No State Transitions

## 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. $k$ different actions have their respective mean of reward, called their **values**. The mean reward for action $a$ is denoted $q_*(a)$, then

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

The reward of taking action $a$ is just a random number distributed following $N(q_*(a), 1)$, the rewards of the actions across time are assumed to be independent.

Unfortunately, we don't know about these exact values (otherwise this problem is trivial since we would always choose the option with the highest value according to the law of large numbers). However, it's still not too bad because we might conclude from the past experience and construct estimates for these values. The estimates for $q_*(a)$ at time $t$ is denoted $Q_t(a)$. It's natural that when we are at time $t$, we look through all $Q_t(a)$ for varying $a$ to find an action whose estimated value is the greatest, called a **greedy** action.

From the definition of a value function, we know that a greedy action provides us with the greatest long-term value based on current knowledge, so shall we necessarily take it? The answer is NO because our knowledge can evolve with time. If we **exploit** (choose the greedy action) all the time and do not explore (choose the non-greedy action), we would not be able to update our knowledge on all the non-greedy actions. This would possibly result in the short-sighted effect that a potentially better long-term choice is never discovered. However, exploring too much is also not what we expect since the loss in reward would be too much. So **the trade-off between exploitation and exploration** is one of the crucial problems in RL.

Let's look at how we shall construct the estimated value function $Q$ to our current knowledge. $Q_t(a)$ should provide an estimate for the mean reward received choosing action $a$ based on experience until time $t$. It's natural that sample mean would be used to replace population mean, with

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

where $\mathbb{I}$ stands for the indicator function. Such estimate is formed as a quotient of the sum of rewards when action $a$ is taken over the times when action $a$ is taken. By the strong law of large numbers (SLLN)

$$\begin{aligned} Q_t(a) &\overset{a.s.}{\to} \frac{\mathbb{E}(R_i \mathbb{I}_{A_i=a})}{\mathbb{P}(A_i = a)} && (3)\\ &= \mathbb{E}(R_i|A_i = a) && (4)\\ &= q_*(a)\ (t \to \infty) && (5) \end{aligned}$$

so this construction is perfectly reasonable and it's called the **sample average** method for estimating action values.

The greedy action is then formulated as:

$$A_t = \arg\max_a Q_t(a) \tag{6}$$

## $\varepsilon$-greedy Methods

To set space for exploration to happen, **$\varepsilon$-greedy methods** are introduced. In each turn, behave greedily with probability $1-\varepsilon$ and explore with probability $\varepsilon$. When exploring, uniformly take one action from all possible actions to update the knowledge. Refer to Fig. 1 for advantages of exploration in long term.

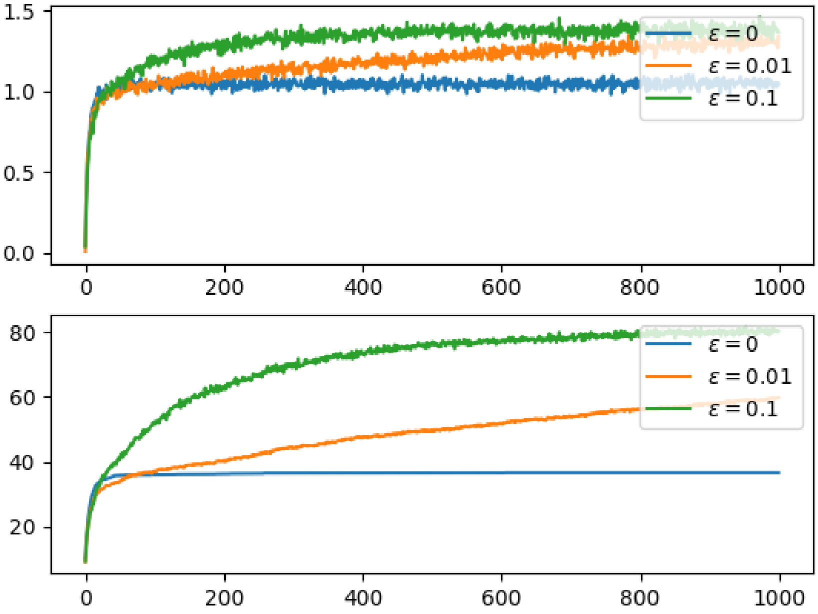


Figure 1: Advantage of $\varepsilon$-greedy strategy over greedy strategy

A $k$ bandit-model with $k = 10$, values of actions are generated following $N(0,1)$ and rewards of action $a$ are generated following $N(q_*(a), 1)$. Such model is considered from step 1 to 1000.
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 $\varepsilon$ 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.

Some remarks on the sample average method for estimating value functions would be that $Q_t(a)$ can actually be written in an **incremental form**. Assume that $R_1, ..., R_k$ is a sequence of rewards derived when choosing action $a$ before time $t$ ($R_i$ is the reward derived choosing action $a$ for the i-th time which is before time $t$), then

$Q_t(a) = Q_{k+1} \overset{def}{=} \frac{R_1+...+R_k}{k}$ where $Q_{k+1}$ has the meaning of the sample average of the rewards of the first $k$ times action $a$ is chosen. Then it's easy to notice:

$$Q_{k+1} = \frac{(k-1)Q_k + R_k}{k} = Q_k + \frac{1}{k}(R_k - Q_k) \tag{7}$$

which means that we don't have to look through all history of actions and rewards each time we have seen action $a$, past calculated $Q_k$ and reward $R_k$ shall suffice for the update of the estimated value $Q_{k+1}$. As a result, we only have to maintain $N(a)$, a count of appearances for each action $a$, and $Q(a)$, the estimate for value of action $a$ to the current knowledge. Then the **simple bandit algorithm** is just:

- Choose an action $a$ based on the estimated value $Q$ with $\varepsilon$-greedy strategy.
- Figure out the reward $r$ for such action $a$.
- Add 1 to N(a) since action $a$ appears for one more time.
- Update $Q(a)$ with incremental form $Q(a) = Q(a) + \frac{1}{N(a)}(r - Q(a))$.

Note that this incremental form also provides the intuition that we are adjusting the old estimate $Q_k$ in the direction of its error $R_k - Q_k$ by a **step size factor** $\alpha = \frac{1}{k}$. As a result, when it comes to non-stationary bandit problems, i.e. the reward distribution changes over time, it provides intuition that we may change the step size factor from $\frac{1}{k}$ to a fixed constant $\alpha$ 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 $\alpha_n(a) = \frac{1}{n}$, satisfying the well-known convergence condition that $\sum_n \alpha_n(a) = \infty, \sum_n \alpha_n^2(a) < \infty$ (this provides the if and only if 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 simplicity and effectiveness. Refer to Fig. 2 for the experiment for **non-stationary bandit problem with constant step size** compared to that with sample mean method.

## Optimistic Start

Just to mention, some tricks for picking up special initial values of estimated value $Q$ may apply for stationary bandit problems, like the **trick of optimistic start**. The intuition is that in our model the distribution of value has mean 0. If we could start with an optimistic estimated value (set the initial values of $Q$ as big positive numbers), then after picking up an action to take and figuring out the reward such action brings with, the bandit algorithm would realize that the actual reward is far lower than the optimistic start provided. The bandit algorithm would then be disappointed toward such an action and is thus forced to try other actions that still have the optimistic start. The optimistic start trick actually forces the bandit algorithm to explore at the very beginning, similar effect to that of $\varepsilon$-greedy strategy. Refer to Fig. 3 for more details. The crucial point here is that although such trick works well

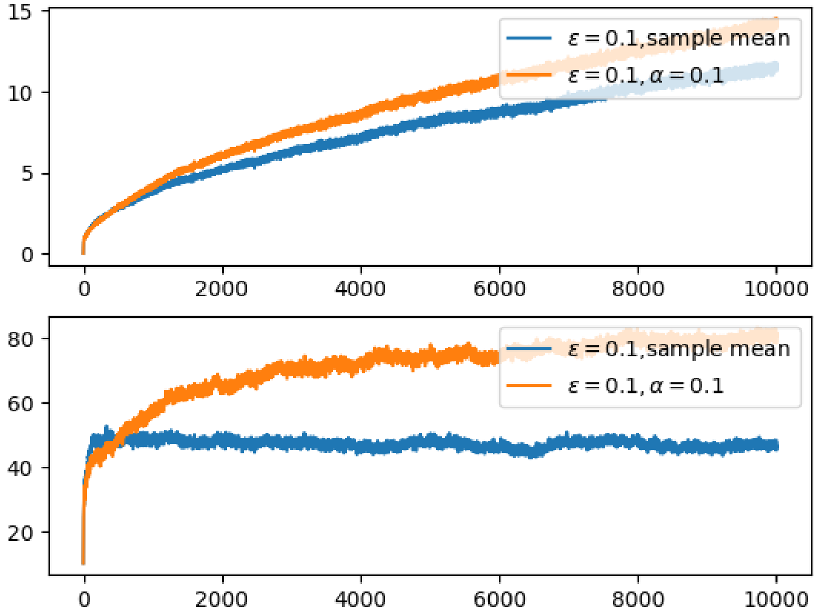


Figure 2: An non-stationary bandit problem

A $k$ bandit-model with $k = 10$, values of actions are generated following $N(0, 1)$ but will change over time. Each value will have an independent $N(0, 0.01)$ random number added to it on each step (as if the values themselves are exhibiting random walks).The rewards of action $a$ are generated following $N(q_*(a), 1)$. Such model is considered from time 1 to 10000. 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. The percentage of optimal action for sample mean method does not increase for a very long time period and the gap between the average rewards enlarges as time passes.

for stationary problems, the performance is not ensured for non-stationary problems and we cannot depend on them. Despite the existence of those tricks, the way to set initial values of $Q$ should not be a matter of concern.

### Unbiased Constant Step Size Trick

The last discussion on the construction of step size methods focuses on a new trick combining the advantages of sample mean method and constant step size. As we have seen in previous context, the sample mean method has the advantage of not producing initial bias (SLLN tells us that all initial bias goes away if there's a long enough time), but it does not work well for non-stationary problems. On the other side, the constant step size method works well

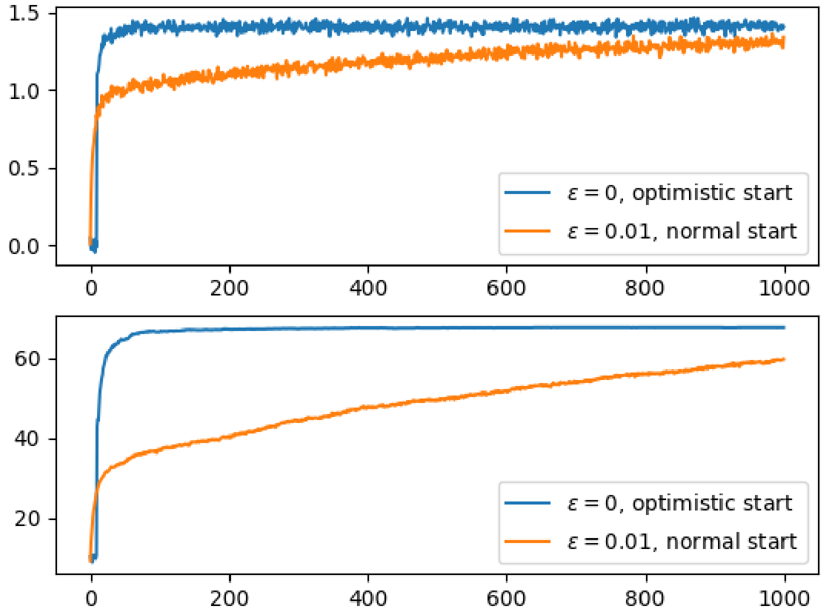


Figure 3: Trick of optimistic start

A stationary $k$ bandit-model with $k = 10$, values of actions are generated following $N(0, 1)$. The rewards of action $a$ are generated following $N(q_*(a), 1)$. Such model is considered from time 1 to 1000.
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 $\varepsilon$-greedy strategy ($\varepsilon = 0.01$) and normal start (for any action $a$, initial values $Q_1(a) = 0$).

for non-stationary problems, but it suffers from initial bias. To see this, do the following calculations:

$$\begin{aligned}
Q_{k+1} &= (1-\alpha)Q_k + \alpha R_k & (8)\\
&= (1-\alpha)[Q_{k-1} + \alpha(R_{k-1} - Q_{k-1})] + \alpha R_k & (9)\\
&= (1-\alpha)^2 Q_{k-1} + (1-\alpha)\alpha R_{k-1} + \alpha R_k & (10)\\
&= ... & (11)\\
&= (1-\alpha)^k Q_1 + \alpha \sum_{i=1}^{k} (1-\alpha)^{k-i} R_i & (12)
\end{aligned}$$

and it's clear that $Q_{k+1}$ is still a weighted average of $Q_1, R_1, ..., R_k$. If the initial estimate $Q_1$ is largely biased, $Q_{k+1}$ would be affected even if $k$ is pretty large.

Here's where the **unbiased constant step size trick** makes a difference. From its name, we can see that such trick combines the advantage of sample mean method and the constant step size method to treat non-stationary problems and maintain its unbiasedness.

The step size to process the n-th reward for a particular action is formulated as:

$$\beta_n \overset{def}{=} \frac{\alpha}{\bar{o}_n} \tag{13}$$

where $\alpha > 0$ is a constant step size and

$$\begin{aligned} \bar{o}_0 &\overset{def}{=} 0 && (14)\\ \bar{o}_n &\overset{def}{=} \bar{o}_{n-1} + \alpha(1-\bar{o}_{n-1}) && (15) \end{aligned}$$

Let's show its unbiasedness by the following calculations:

$$\begin{aligned} Q_{k+1} &= (1-\beta_k)Q_k + \beta_k R_k && (16)\\ &= (1-\beta_k)[Q_{k-1} + \beta_{k-1}(R_{k-1} - Q_{k-1})] + \beta_k R_k && (17)\\ &= (1-\beta_k)(1-\beta_{k-1})Q_{k-1} + (1-\beta_k)\beta_{k-1}R_{k-1} + \beta_k R_k && (18)\\ &= ... && (19)\\ &= (1-\beta_k)...(1-\beta_1)Q_1 + \sum_{i=1}^{k} \beta_i(1-\beta_{i+1})...(1-\beta_k)R_i && (20) \end{aligned}$$

Let's look into the coefficient of $Q_1$ which is $(1-\beta_1)...(1-\beta_k)$:

$$(1-\beta_1)...(1-\beta_k) = \left(1 - \frac{\alpha}{\bar{o}_1}\right)...\left(1 - \frac{\alpha}{\bar{o}_k}\right) = 0 \tag{21}$$

since $\bar{o}_1 = \alpha$.

Is it still a weighted average?

$$\begin{aligned} \sum_{i=1}^{k} \beta_i(1-\beta_{i+1})...(1-\beta_k) &= \alpha \sum_{i=1}^{k} \frac{1}{\bar{o}_i}\left(1 - \frac{\alpha}{\bar{o}_{i+1}}\right)...\left(1 - \frac{\alpha}{\bar{o}_k}\right) && (22)\\ &= \alpha \sum_{i=1}^{k} \frac{1-\alpha}{\bar{o}_{i+1}}\left(1 - \frac{\alpha}{\bar{o}_{i+2}}\right)...\left(1 - \frac{\alpha}{\bar{o}_k}\right) && (23)\\ &= ... && (24)\\ &= \alpha \sum_{i=1}^{k} (1-\alpha)^{k-i}\frac{1}{\bar{o}_k} && (25)\\ & && (26) \end{aligned}$$

Note that we can actually solve out $\bar{o}_n$ with its recursion formula:

$$\bar{o}_n - 1 = (1-\alpha)(\bar{o}_{n-1} - 1) \tag{27}$$

$$\bar{o}_n = 1 - (1-\alpha)^n \tag{28}$$

So

$$\begin{aligned}
\sum_{i=1}^{k} \beta_i (1-\beta_{i+1})...(1-\beta_k) &= \frac{\alpha}{1-(1-\alpha)^k} \sum_{i=1}^{k} (1-\alpha)^{k-i} && (29)\\
&= \frac{\alpha}{1-(1-\alpha)^k} \frac{1-(1-\alpha)^k}{\alpha} && (30)\\
&= 1 && (31)\\
& && (32)
\end{aligned}$$

As we can see, it's a subtle design that on one hand we keep the constant step size $\alpha$ and on the other hand we maintain the sequence $\bar{o}_n$ as if it's a seuqence of estimate for values when a reward of 1 is derived in each step. The quotient of these two quantities then gives the unbiased constant step size trick.

See Fig. 4 for experiments running this trick on non-stationary bandit problems.

## Upper-Confidence-Bound Action Selection (UCB)

In the context above, discussion are organized within the scope of $\varepsilon$-greedy strategy, in which 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 specified. This leads to the **upper-confidence-bound action selection (UCB)** where a regularization term is added to the estimated value as a modification of the greedy strategy. The details are provided below:

$$A_t = \arg\max_a \left( Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}} \right) \tag{33}$$

where $\log t$ is the natural logarithm, $c > 0$ is a hyperparameter for the degree of exploration and $N_t(a)$ denotes the number of appearances of action $a$ before time $t$. The intuition of the term $\sqrt{\frac{\log t}{N_t(a)}}$ is that as time goes by, the exploration term is getting the dominating position slowly enough but can still grow into an unbounded term, and actions rarely picked up are preferred to be explored. UCB actually exhibits a quantitative trade-off between the estimated value (how much possible importance the action has) and the exploration need (how urgent it is to explore this action). UCB works pretty well for stationary bandit problems (probably the best), but for the general RL case, it can't work well practically (UCB has something to do with the logarithm asymptotic total regret). Refer to Fig. 5 for the experiments conducted with UCB.

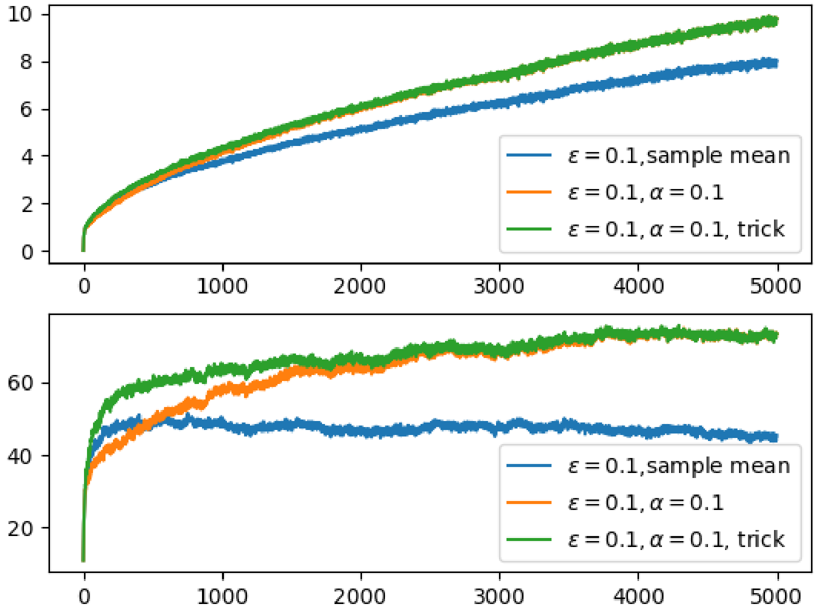


Figure 4: Unbiased constant step size trick for non-stationary bandit problem

A $k$ bandit-model with $k = 10$, values of actions are generated following $N(0, 1)$ but will change over time. Each value will have an independent $N(0, 0.01)$ random number added to it on each step (as if the values themselves are exhibiting random walks).The rewards of action $a$ are generated following $N(q_*(a), 1)$. Such model is considered from time 1 to 5000.
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, the trick works pretty well and is faster in convergence than the constant step size model.

## 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 values. However that's not the only approach since we can also learn a numerical **preference** for each action $a$ at time $t$, denoted $H_t(a)$. The preference works in a different way from the estimates for values that actions with larger preferences are more likely to be chosen. In other words, the preference is updated throughout the whole process and actions are selected only based on preferences and an estimate for value is no longer required.

Since the preferences are real numbers, a natural question is that how can we know which action to take at time $t$ given the preferences for each action. The answer would be: turn the real numbers into probabilities using the softmax function. This is a common practice to take in machine learning fields and the deeper level explanation can be given as a connection to the Boltzmann distribution.

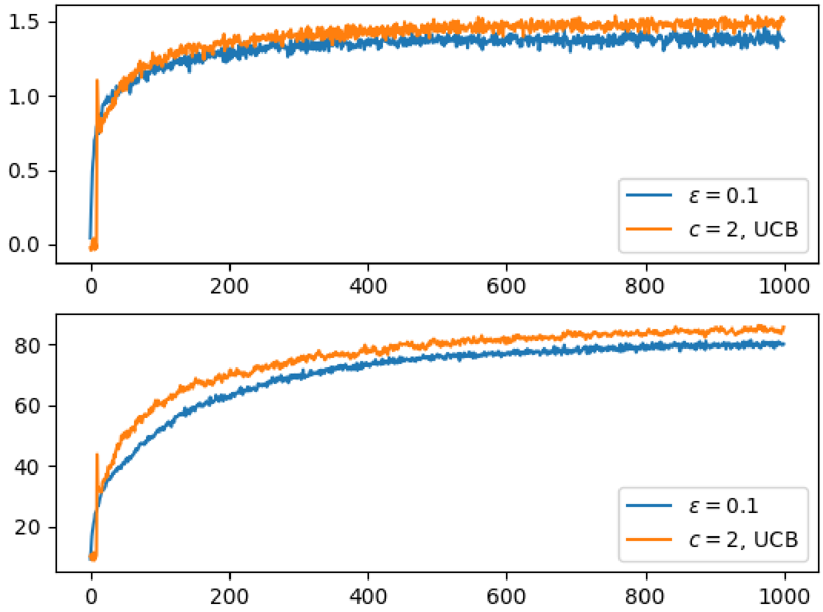


Figure 5: UCB for stationary bandit problems

A stationary $k$ bandit-model with $k = 10$, values of actions are generated following $N(0, 1)$. The rewards of action $a$ are generated following $N(q_*(a), 1)$. Such model is considered from time 1 to 1000.
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, UCB with $c = 2$ performs better than $\varepsilon$-greedy strategy with $\varepsilon = 0.1$ on stationary bandit problems. Note the spikes in the beginning period of the UCB learning curves, it's sign of exploration among different actions when the exploration term dominates the estimated value term.

$$\pi_t(a) \overset{def}{=} \mathbb{P}\left(A_t = a\right) \tag{34}$$

$$\overset{def}{=} \frac{e^{H_t(a)}}{\sum_b e^{H_t(b)}} \tag{35}$$

where $\pi_t(a)$ denotes the probability of taking action $a$ at time $t$ knowing all preferences. It's then obvious that the action with the highest $\pi_t(a)$ would be selected.

The only problem to solve is how to update the preference. As is expected, at the very start, all preferences are set as a same value (e.g. as 0), indicating that there's no preferences on any of the actions. As the learning process begins, preferences have to be adjusted based on the information we have already had in order to ensure that we would be able to make better choice the next time.

The update of preference forms the **gradient bandit algorithm**:

$$H_{t+1}(a) = H_t(a) + \alpha \left(R_t - \overline{R}_t\right)\left(\mathbb{I}_{A_t=a} - \pi_t(a)\right) \tag{36}$$

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

A direct explanation on this update formula 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 (positive effect). The more likely it is for action $a$ to be selected at time $t$, the more its preference shall be increased. The more reward it receives for action $a$ to be selected at time $t$, the more its preference shall be increased. 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 disadvantages. So where does such update formula come from?

The answer is gradient! Actually, we hope to view the update as a direct **gradient ascent** method in order to maximize the expected reward received.

$$H_{t+1}(a) = H_t(a) + \alpha \frac{\partial \mathbb{E} R_t}{\partial H_t(a)} \tag{37}$$

Now to figure out what $\mathbb{E}R_t$ is, note that

$$\begin{aligned}
\mathbb{E}R_t &= \sum_a \mathbb{P}\left(A_t = a\right)\mathbb{E}(R_t|A_t = a) && (38)\\
&= \sum_a \pi_t(a) q_*(a) && (39)
\end{aligned}$$

However, note that $q_*(a)$ is the unknown value we wish to learn, so such gradient ascent method can't be implemented practically. Let's still calculate the partial derivative to see what we can get.

$$\begin{aligned}
\frac{\partial \mathbb{E}R_t}{\partial H_t(a)} &= \sum_x q_*(x) \frac{\partial \pi_t(x)}{\partial H_t(a)} && (40)\\
&= \sum_x q_*(x) \frac{\partial \frac{e^{H_t(x)}}{\sum_b e^{H_t(b)}}}{\partial H_t(a)} && (41)\\
&= \sum_x q_*(x) \frac{\mathbb{I}_{a=x} e^{H_t(x)} \sum_b e^{H_t(b)} - e^{2H_t(x)}}{\left(\sum_b e^{H_t(b)}\right)^2} && (42)\\
&= \sum_x q_*(x) \pi_t(x) \left(\mathbb{I}_{a=x} - \pi_t(a)\right) && (43)
\end{aligned}$$

Now note that we can add a baseline to this partial derivative by using the special probability structure of $\pi_t$:

$$\sum_x \pi_t(x) = 1 \tag{44}$$

$$\sum_x \frac{\partial \pi_t(x)}{\partial H_t(a)} = \sum_x \pi_t(x)\mathbb{I}_{x=a} - \sum_x \pi_t(x)\pi_t(a) \tag{45}$$

$$= \pi_t(a) - \pi_t(a) = 0 \tag{46}$$

By adding an arbitrary baseline $B_t$ (to be selected afterwards), we get

$$\frac{\partial \mathbb{E}R_t}{\partial H_t(a)} = \sum_x \left(q_*(x) - B_t\right) \frac{\partial \pi_t(x)}{\partial H_t(a)} \tag{47}$$

$$= \sum_x \pi_t(x)\left(q_*(x) - B_t\right)\left(\mathbb{I}_{a=x} - \pi_t(a)\right) \tag{48}$$

Note that the partial derivative has a natural structure as an expectation w.r.t. the action $A_t$ by viewing $\pi_t(x)$ as its probability masses:

$$\frac{\partial \mathbb{E}R_t}{\partial H_t(a)} = \mathbb{E}_{A_t \sim \pi_t}\left[\left(q_*(A_t) - B_t\right)\left(\mathbb{I}_{A_t=a} - \pi_t(a)\right)\right] \tag{49}$$

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

$$H_{t+1}(a) = H_t(a) + \alpha \frac{\partial \mathbb{E}R_t}{\partial H_t(a)} \tag{50}$$

$$= H_t(a) + \alpha \mathbb{E}_{A_t \sim \pi_t}\left[\left(q_*(A_t) - B_t\right)\left(\mathbb{I}_{A_t=a} - \pi_t(a)\right)\right] \tag{51}$$

with value $q_*(A_t)$ unknown and baseline $B_t$ not specified yet. Note that this update is very close to the **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 such expectation (true gradient), it's totally possible to **randomly sample directions such that the expectation goes along the true gradient (expectation)**. Now we know that $q_*(A_t) = \mathbb{E}(R_t|A_t)$, so

$$\mathbb{E}_{A_t \sim \pi_t}\left[\left(q_*(A_t) - B_t\right)\left(\mathbb{I}_{A_t=a} - \pi_t(a)\right)\right] = \mathbb{E}_{A_t \sim \pi_t}\mathbb{E}\left[\left(R_t - B_t\right)\left(\mathbb{I}_{A_t=a} - \pi_t(a)\right)\middle| A_t\right] \tag{52}$$

$$= \mathbb{E}_{A_t \sim \pi_t, R_t}\left[\left(R_t - B_t\right)\left(\mathbb{I}_{A_t=a} - \pi_t(a)\right)\right] \tag{53}$$

and we find that $(R_t - B_t)(\mathbb{I}_{A_t=a} - \pi_t(a))$ would be a reasonable sample of the gradient with the required expectation.

So far, we have actually proved that **the gradient bandit algorithm in an instance of stochastic gradient ascent**! Its **general form** for any baseline $B_t$ is as follows:

$$H_{t+1}(a) = H_t(a) + \alpha\left(R_t - B_t\right)\left(\mathbb{I}_{A_t=a} - \pi_t(a)\right) \tag{54}$$

and the most important property of such method is that

$$\mathbb{E}_{(A_t,R_t)}[(R_t - B_t)(\mathbb{I}_{A_t=a} - \pi_t(A_t))] = \frac{\partial \mathbb{E} R_t}{\partial H_t(a)} \tag{55}$$

In practice, it's hard and unnecessary to figure out which baseline $B_t$ works the best, and practice proves that $B_t = \overline{R}_t$ 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 $B_t = 0$ is not acceptable and behaves poorly in practice. Refer to Fig. 6,7 for experiments conducted on bandit problems with gradient bandit algorithms showing a comparison between a past-average-baseline scheme and an no-baseline scheme. (There might be some problems with these two figures since the optimal percentage should be expected to be much higher than shown. However, I can't found any bugs in my code. So the readers are welcome to do simulations on their own and contact me if they produce different figures.)

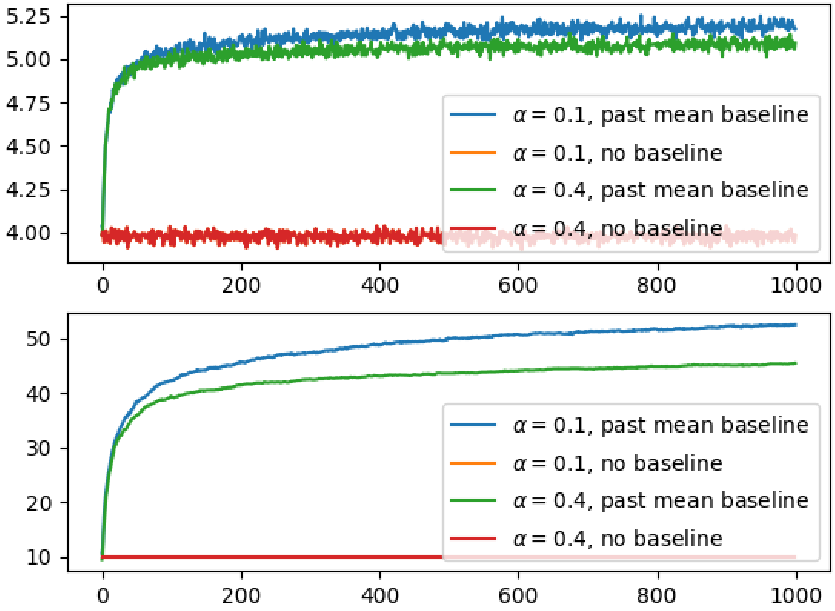


Figure 6: Gradient bandit algorithm for stationary bandit problems

A stationary $k$ bandit-model with $k = 10$, values of actions are generated following $N(4, 1)$. The rewards of action $a$ are generated following $N(q_*(a), 1)$. Such model is considered from time 1 to 1000.
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, gradient bandit with baseline $B_t = \overline{R}_t$ performs much better than that with no baseline $B_t = 0$ especially when the mean of the values are biased away from 0.

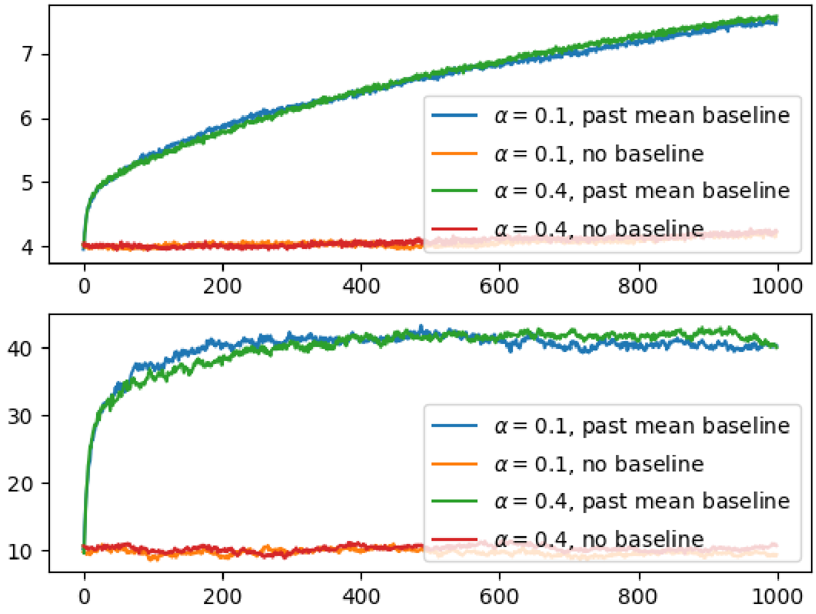


Figure 7: Gradient bandit algorithm for non-stationary bandit problem

A $k$ bandit-model with $k = 10$, values of actions are generated following $N(4, 1)$ but will change over time. Each value will have an independent $N(0, 0.01)$ random number added to it on each step (as if the values themselves are exhibiting random walks).The rewards of action $a$ are generated following $N(q_*(a), 1)$. Such model is considered from time 1 to 1000. 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, gradient bandit with baseline $B_t = \overline{R}_t$ performs much better than that with no baseline $B_t = 0$ especially when the mean of the values are biased away from 0.

## Summary

In the context above, non-associative search is discussed, where there's only one state and the set of available actions does not change. However, general RL problems are always associative, and associativity is very different from the non-stationarity mentioned above since non-stationarity still belongs to the category of non-associative search tasks. By introducing different states and different available actions for each state, the action-value methods won't work well generally. That's why for general RL problems, a **policy** is trained instead. The policy tells us what's the best action to take in each state. Nevertheless, the methods discussed above will provide us with inspirations of new methods that work in more general situations. The $\varepsilon$-greedy strategy, UCB are about the trade-off between exploitation and exploration, the constant step size and the non-biased constant step size trick are widely used in non-stationary cases, and the gradient bandit method provides new perspectives on the preferences of actions as an instance of the stochastic gradient ascent.

# Markov Decision Process (MDP)

In the last section, we have looked into RL problems that have only one state and multiple actions. Action-value methods can be applied to estimate the value of each action and select the action with the highest estimated value. Gradient bandit method, as an instance of stochastic gradient ascent method, updates the preferences such that the preferences maximize the expectation of the reward.

However, the introduction of different states brings with much complexity. The set of available actions may depend on the state, thus affecting the selection for the optimal action. Under such circumstances, the introduction of MDP helps to set up the general RL framework.

## Basic Settings

The MDP adopts the agent-environment interaction, where the **agent** means the learner or decision maker and the **environment** means everything other than the agent. The logic is that on seeing the state the agent is at and the reward the agent receives, the agent makes an action that changes the environment, causing updates in the state and reward. The state $s \in \mathscr{S}$, with $\mathscr{S}$ to be the set of all possible states. The action $a \in \mathscr{A}(s)$, with $\mathscr{A}(s)$ to be the set of all possible actions given state $s$ (Note that the set of available actions may depend on the state). The reward $r \in \mathscr{R} \subset \mathbb{R}$. MDP is generally formed as a sequential decision-making process, generating a sequence $S_0, A_0, R_1, S_1, A_1, R_2, ...$ of states, actions and rewards, where $R_{t+1}$ is actually the reward for selecting the action $A_t$.

Only **finite MDP** is in our scope, for which the sets of states, actions and rewards are all finite. It's then obvious that we can denote the set of all possible actions as $\mathscr{A}$ now, not depending on the state $s$ and it remains finite. The **dynamics** of MDP decides how a sequence is generated and how states, actions and rewards come into play, and it's assumed to be time homogeneous.

$$p(s', r|s, a) \overset{def}{=} \mathbb{P}\left(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\right) \tag{56}$$

The "Markov" in the name of MDP actually refers to the fact that given the present, i.e. $S_t, A_t$, the future, i.e. $R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, ...$ is independent of the past, i.e. $S_0, A_0, ..., S_{t-1}, A_{t-1}$. Here some notations are introduced as functions of the dynamics.

$$\begin{aligned} p(s'|s, a) &\overset{def}{=} \mathbb{P}\left(S_t = s'|S_{t-1} = s, A_{t-1} = a\right) && (57) \\ &= \sum_r \mathbb{P}\left(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\right) && (58) \\ &= \sum_r p(s', r|s, a) && (59) \end{aligned}$$

gives the **state-transition probabilities**.

$$\begin{aligned}
r(s,a) &\overset{def}{=} \mathbb{E}\left(R_t|S_{t-1}=s,A_{t-1}=a\right) && (60)\\
&= \sum_r r\cdot\mathbb{P}\left(R_t=r|S_{t-1}=s,A_{t-1}=a\right) && (61)\\
&= \sum_r r\cdot\sum_{s'} p(s',r|s,a) && (62)
\end{aligned}$$

gives the **expected rewards for state-action pairs**.

$$\begin{aligned}
r(s,a,s') &\overset{def}{=} \mathbb{E}\left(R_t|S_{t-1}=s,A_{t-1}=a,S_t=s'\right) && (63)\\
&= \sum_r r\cdot\mathbb{P}\left(R_t=r|S_{t-1}=s,A_{t-1}=a,S_t=s'\right) && (64)\\
&= \sum_r r\cdot\frac{p(s',r|s,a)}{\mathbb{P}\left(S_t=s'|S_{t-1}=s,A_{t-1}=a\right)} && (65)\\
&= \frac{\sum_r r\cdot p(s',r|s,a)}{\sum_r p(s',r|s,a)} && (66)
\end{aligned}$$

gives the **expected rewards for state-action-next-state triples**.

## Construction of Return

MDP is set up with a **reward hypothesis**, i.e. we want to maximize the expectation of the cumulative sum of the rewards. Since the rewards are constructed and given for each RL problem, the design of rewards is always subtle. A good design can inform the computer of the real objective and clarify what one really hope to achieve.

RL tasks are basically separated into two kinds based on their time structures: **episodic tasks** and **continuing tasks**. Episodic tasks see the end of the environment for each episode, and different episodes are independent. For such tasks, a stopping time $T$ is often used to denote the random time of the end of the current episode. Continuing tasks bring with more difficulties since the task shall never end. To ensure that "the expectation of the cumulative sum of the rewards" is well-defined, **discounting techniques** are introduced to compute the expected discounted return.

$$G_t \overset{def}{=} \sum_{k=0}^{T}\gamma^k R_{t+k+1} \tag{67}$$

where $\gamma$ is the discount rate, $T$ can take value as $\infty$ and $G_t$ is called the **return** at time $t$. **NOTE:** $R_{t+1}$ is actually the reward given based on state $S_t$ and action $A_t$, so the return at time $t$ actually contains information of the reward based on state and action at time $t$.

Similarly to what we have done in the last section, the return can be written in its incremental form for the simplicity of realization.

$$G_t = R_{t+1} + \gamma G_{t+1} \tag{68}$$

Actually, such discounted return works for both episodic tasks and continuing tasks by adding an absorbing state to the episodic task that always generates reward 0. That's the reason why we won't deal with these two kinds of problems separately. **NOTE**: Return is totally different from reward since return takes long-term rewards into consideration. We will see that it's much more reasonable to use returns to define value functions than rewards because the motivation of value functions is just to measure the long-term value of a certain action.

## Policy and Value Functions

After defining the return that focuses on the long-term discounted reward, the estimation of value functions is naturally built upon the return. However, before introducing the definitions of value functions, it has to be clarified that we are actually training something called **policy** in the RL problem. The policy $\pi$ is actually a conditional distribution, telling us the probability taking any action given the current state.

$$\pi(a|s) = \mathbb{P}\left(A_t = a | S_t = s\right) \tag{69}$$

If the current state is $S_t$ and actions are selected according to policy $\pi$, the the expectation of $R_{t+1}$ in terms of $\pi$ and the dynamics $p$ would be:

$$\begin{aligned}
\mathbb{E}R_{t+1} &= \sum_r r \cdot \mathbb{P}\left(R_{t+1} = r\right) && (70)\\
&= \sum_r r \cdot \sum_{s,a} \mathbb{P}\left(S_t = s, A_t = a\right) \mathbb{P}\left(R_{t+1} = r | S_t = s, A_t = a\right) && (71)\\
&= \sum_r r \cdot \sum_{s,a} \pi(a|s) \mathbb{P}\left(S_t = s\right) \cdot \sum_{s'} p(s', r|s, a) && (72)\\
& && (73)
\end{aligned}$$

Since we already know that the current state is $S_t$, the conclusion is that (Exercise 3.11)

$$\mathbb{E}R_{t+1} = \sum_r r \cdot \sum_a \pi(a|S_t) \cdot \sum_{s'} p(s', r|S_t, a) \tag{74}$$

Now let's introduce the notions of general value functions. Value functions are tightly related to the policy $\pi$ chosen, and should also depend on the state or action. The **state value function for policy** $\pi$ is defined as the expectation of return conditional on the given current state:

$$v_\pi(s) \overset{def}{=} \mathbb{E}_\pi(G_t|S_t = s) \tag{75}$$

$$= \mathbb{E}_\pi\left(\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle|S_t = s\right) \tag{76}$$

Another variant of value function has action involved and is called **the action value function for policy** $\pi$:

$$q_\pi(s,a) \overset{def}{=} \mathbb{E}_\pi(G_t|S_t = s, A_t = a) \tag{77}$$

$$= \mathbb{E}_\pi\left(\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle|S_t = s, A_t = a\right) \tag{78}$$

Let's provide a formula for $v_\pi$ in terms of $q_\pi, \pi$. (Exercise 3.12)

$$v_\pi(s) = \mathbb{E}_\pi(G_t|S_t = s) \tag{79}$$

$$= \sum_a \mathbb{P}\left(A_t = a|S_t = s\right)\mathbb{E}_\pi(G_t|S_t = s, A_t = a) \tag{80}$$

$$= \sum_a q_\pi(s,a)\pi(a|s) \tag{81}$$

Similarly, provide a formula for $q_\pi$ in terms of $v_\pi, p$. (Exercise 3.13)

$$q_\pi(s,a) = \mathbb{E}_\pi(G_t|S_t = s, A_t = a) \tag{82}$$

$$= \mathbb{E}_\pi(R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a) \tag{83}$$

$$= \mathbb{E}_\pi(R_{t+1}|S_t = s, A_t = a) + \gamma\mathbb{E}_\pi(G_{t+1}|S_t = s, A_t = a) \tag{84}$$

$$= \sum_r r\cdot\sum_{s'} p(s',r|s,a) + \gamma\sum_{s'}\mathbb{E}_\pi(G_{t+1}|S_{t+1} = s', S_t = s, A_t = a)\mathbb{P}\left(S_{t+1} = s'|S_t = s, A_t = a\right) \tag{85}$$

$$= \sum_r r\cdot\sum_{s'} p(s',r|s,a) + \gamma\sum_{s'} v_\pi(s')\sum_r p(s',r|s,a) \tag{86}$$

$$= \sum_{r,s'}(r + \gamma v_\pi(s'))p(s',r|s,a) \tag{87}$$

Note that $\mathbb{E}_\pi(G_{t+1}|S_{t+1} = s', S_t = s, A_t = a) = \mathbb{E}_\pi(G_{t+1}|S_{t+1} = s')$ is due to the Markov property since $G_{t+1}$ is a function of $R_{t+2}, R_{t+3}, ....$ Recognize the time $t+1$ as present, the states and actions before time $t+1$ are the past and the rewards after time $t+1$ are the future, so they are independent.

To understand these two value functions deeper, let's consider some other formulations.

$$
\begin{aligned}
q_\pi(s,a) &= \mathbb{E}_\pi\left(\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle| S_t=s, A_t=a\right) && (88)\\
&= \sum_{k=0}^{\infty}\gamma^k \mathbb{E}_\pi\left(R_{t+k+1}\middle| S_t=s, A_t=a\right) && (89)\\
& && (90)
\end{aligned}
$$

Notice that $\mathbb{E}_\pi\left(R_{t+k+1}\middle| S_t=s, A_t=a\right)$ has nothing to do with $t$ since the MDP is Markov and time-homogeneous and the policy does not change with time:

$$
\begin{aligned}
&\mathbb{E}_{\pi,S_0,A_0}\left(R_{t+k+1}\middle| S_t, A_t\right) && (91)\\
&= \mathbb{E}_{\pi,S_0,A_0}\left(R_{k+1}\circ\theta_t\middle|\mathscr{F}_t\right) && (92)\\
&= \mathbb{E}_{\pi,S_t,A_t}\left(R_{k+1}\right) && (93)\\
&\mathbb{E}_\pi\left(R_{t+k+1}\middle| S_t=s, A_t=a\right) = \mathbb{E}_{\pi,S_t=s,A_t=a}\left(R_{k+1}\right) && (94)
\end{aligned}
$$

The same thing happens for $q_\pi$ since $v_\pi(s)=\sum_a q_\pi(s,a)\pi(a|s)$. As a result, we conclude that both the state value function and the action value function are **time-invariant**.

After stating the properties and connections between value functions, let's briefly talk about estimating value functions in practice. A direct thought is to do **Monte Carlo** simulations since $v_\pi, q_\pi$ both have the structure as conditional expectations. By adopting policy $\pi$ to do simulations, multiple chains $S_0, A_0, R_1, ...$ would be derived. For each of these chains, pick out the realizations of states equal to $s$ (here assume that we have $S_t = s$), then $G_t$, the return at time $t$ can be computed since all rewards afterwards are known. By taking the sample average of all $G_t$ such that $S_t = s$, an estimate for $v_\pi(s)$ is then constructed. Similar operations work for estimating $q_\pi(s,a)$, the only difference is that we have to make sure both the state and the action at a certain time meet the restrictions.

Despite the help of Monte Carlo, many problems exist for value function estimations. One might assert that such estimation may take too long time, which often happens when a certain state $s$ has very little possibility appearing under policy $\pi$, so the appearance of $s$ cannot be empirically observed, resulting in the failure of Monte Carlo. One might also assert that when there are too many states, Monte Carlo will do a poor job since the convergence of Monte Carlo greatly depends on the size of samples. If each state is only distributed with a small number of samples, the estimation will be largely biased. Besides, it's also impossible to keep record of $v_\pi(s)$ for each state $s$ and policy $\pi$, so some parametric skills would be required. That's why there's still much work to do in the field of estimating value functions.

The last thing to say is the **Bellman Equation**. It's an equation exhibiting the structure of value functions and can be widely used in the estimation of value functions. The Bellman equation w.r.t. $v_\pi$ is provided below with the use of Markov property:

$$
\begin{align}
v_\pi(s) &= \mathbb{E}(G_t|S_t = s) \tag{95}\\
&= \mathbb{E}(R_{t+1}|S_t = s) + \gamma\mathbb{E}(G_{t+1}|S_t = s) \tag{96}\\
&= \sum_r r \cdot \mathbb{P}\left(R_{t+1} = r|S_t = s\right) + \gamma\sum_{s',a}\mathbb{P}\left(S_{t+1} = s', A_t = a|S_t = s\right)\mathbb{E}(G_{t+1}|S_{t+1} = s', S_t = s, A_t = a) \tag{97}\\
&= \sum_r r \cdot \sum_a \mathbb{P}\left(A_t = a|S_t = s\right)\mathbb{P}\left(R_{t+1} = r|S_t = s, A_t = a\right) + \gamma\sum_{s',a}\mathbb{P}\left(S_{t+1} = s', A_t = a|S_t = s\right)\mathbb{E}(G_{t+1}|S_{t+1} = s') \tag{98}\\
&= \sum_r r \cdot \sum_a \pi(a|s) \cdot \sum_{s'} p(s', r|s, a) + \gamma\sum_{s',a}\mathbb{P}\left(S_{t+1} = s'|S_t = s, A_t = a\right)\mathbb{P}\left(A_t = a|S_t = s\right)v_\pi(s') \tag{99}\\
&= \sum_r r \cdot \sum_a \pi(a|s) \cdot \sum_{s'} p(s', r|s, a) + \gamma\sum_{s',a}\sum_r p(s', r|s, a)\pi(a|s)v_\pi(s') \tag{100}\\
&= \sum_a \pi(a|s)\sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{101}
\end{align}
$$

The similar operations provide the Bellman equation w.r.t. $q_\pi$ with the use of the relationship between $v_\pi$ and $q_\pi$(Exercise 3.17):

$$
\begin{align}
q_\pi(s, a) &= \mathbb{E}(G_t|S_t = s, A_t = a) \tag{102}\\
&= \mathbb{E}(R_{t+1}|S_t = s, A_t = a) + \gamma\mathbb{E}(G_{t+1}|S_t = s, A_t = a) \tag{103}\\
&= \sum_r r \cdot \mathbb{P}\left(R_{t+1} = r|S_t = s, A_t = a\right) + \gamma\sum_{s'}\mathbb{P}\left(S_{t+1} = s'|S_t = s, A_t = a\right)\mathbb{E}(G_{t+1}|S_{t+1} = s', S_t = s, A_t = a) \tag{104}\\
&= \sum_r r \cdot \sum_{s'} p(s', r|s, a) + \gamma\sum_{s'}\sum_r p(s', r|s, a)v_\pi(s') \tag{105}\\
&= \sum_r r \cdot \sum_{s'} p(s', r|s, a) + \gamma\sum_{s'}\sum_r p(s', r|s, a) \cdot \sum_{a'}\pi(a'|s')q_\pi(s', a') \tag{106}\\
&= \sum_{r,s'} p(s', r|s, a)[r + \sum_{a'}\gamma\pi(a'|s')q_\pi(s', a')] \tag{107}
\end{align}
$$

As a result, given policy $\pi$ and MDP dynamics $p$, the two Bellman equations are linear systems in $v_\pi(s)$ and $q_\pi(s, a)$. For a small RL model, these value functions can be derived easily and accurately.

## Gridworld Model as an Example

Let's now try to set up a simple RL task called **Gridworld** (Example 3.5). All possible states are the 25 grids in the $5 \times 5$ 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. All the other actions has reward 0. Nevertheless, there are two special grids (1,4) and (3,4) (coordinates are counted starting from 0 to 4). 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.

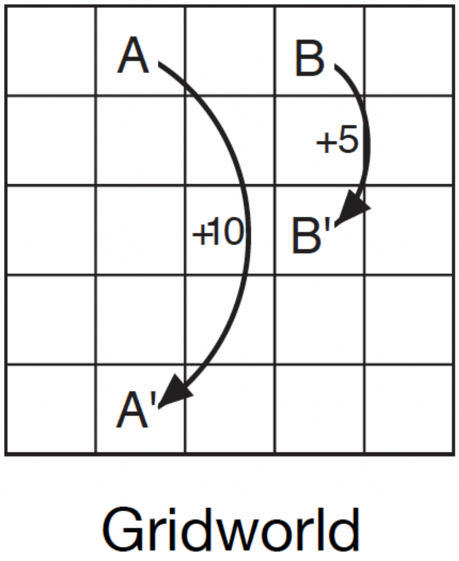
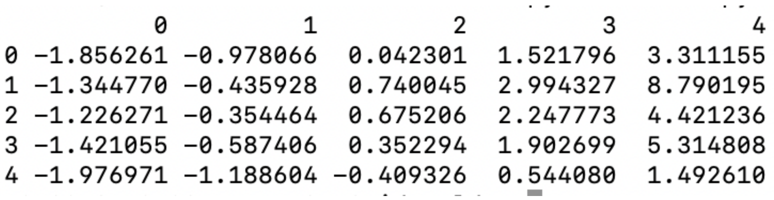


Figure 8: Graph illustration of the Gridworld model

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 $v_\pi$ for this model given the policy $\pi$ to be the equiprobable policy, i.e. four actions always have $\frac{1}{4}$ of being taken at each state. By adopting the Monte Carlo method mentioned above, we get the estimate for state value function $v_\pi$ in Fig. 9.



```
          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

1000 times of simulations have been done to form the estimates. Within each simulation, state, action and reward are being generated until time 20000. The discount rate $\gamma = 0.9$.

By comparing this matrix with the Figure 3.2 in the book, we conclude that the Monte Carlo estimation works pretty well. Let's first try to explain the structure of $v_\pi$. As can be expected, the entry at (1,4) and (3,4) has the highest state values among all states, which is natural since the rewards for reaching these two states are the only positive rewards in the model. As expected, the states next to (1,4) and (3,4) also benefit from the fact that they have larger probabilities of transiting into these two states. When it comes to states far away from these two high-value states and next to the boundary (like column 0), the value is the lowest since it's very likely for these states to go outside the boundary and receive reward -1. However, the interesting is that (1,4) is having 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 low-value state. For (3,4), the next state must be (3,2), a much better state, not on the boundary and closer to these high-value states.

Secondly, let's try to verify that the Bellman equation holds for such state value function

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \tag{108}$$

Let's first pick out the state $s = (2,2)$ and it's four neighboring states to see whether the Bellman equation holds.

$$\begin{aligned}
LHS &= 0.6752 &(109)\\
RHS &= \frac{1}{4} \times 0.9 \times [v_\pi(s' = (2,3)) + v_\pi(s' = (2,1)) + v_\pi(s' = (3,2)) + v_\pi(s' = (1,2))] &(110)\\
&= \frac{0.9}{4} \times (2.25 - 0.35 + 0.35 + 0.74) &(111)\\
&= 0.67275 &(112)
\end{aligned}$$

the results on two sides are very close to each other, showing that Bellman equation works.

Let's verify the equation for another state (0,0) on the boundary:

$$\begin{aligned}
LHS &= -1.8563 &(113)\\
RHS &= \frac{1}{4} \times [0.9 \times (v_\pi(s' = (0,1)) + v_\pi(s' = (1,0)) + v_\pi(s' = (0,0)) + v_\pi(s' = (0,0))) - 2] &(114)\\
&= -1.859 &(115)
\end{aligned}$$

still quite close.

At last, verify the equation for (1,4), the high-value state.

$$\begin{aligned}
LHS &= 8.79 &(116)\\
RHS &= 10 + 0.9 \times v_\pi(s' = (1,0)) &(117)\\
&= 8.794 &(118)
\end{aligned}$$

still holds.

Last but not least, we would like to mention that Gridworld is a continuing task since no specific ending criterion of the task is assumed. As a result, adding a same constant to all rewards won't affect the relative values of any states under any policies. However, this does not hold for episodic tasks since episodic tasks may have different ending time under different policies, resulting in different discounted values for such constant.

## A Proof for the Uniqueness of the Solution to Bellman Equations

Let's first consider the Bellman equation for state value function, write it in the matrix form and assume that there are altogether $n$ different states $s_1, ..., s_n$, the coefficient matrix is

$$A = \begin{bmatrix} \gamma\sum_a \pi(a|s_1)p(s_1|s_1,a) - 1 & \gamma\sum_a \pi(a|s_1)p(s_2|s_1,a) & ... & \gamma\sum_a \pi(a|s_1)p(s_n|s_1,a) \\ \gamma\sum_a \pi(a|s_2)p(s_1|s_2,a) & \gamma\sum_a \pi(a|s_2)p(s_2|s_2,a) - 1 & ... & \gamma\sum_a \pi(a|s_2)p(s_n|s_2,a) \\ ... & & & \\ \gamma\sum_a \pi(a|s_n)p(s_1|s_n,a) & \gamma\sum_a \pi(a|s_n)p(s_2|s_n,a) & ... & \gamma\sum_a \pi(a|s_n)p(s_n|s_n,a) - 1 \end{bmatrix} \tag{119}$$

where $p(s'|s,a) = \mathbb{P}\left(S_{t+1} = s'|S_t = s, A_t = a\right)$. Then notice the fact that

$$\gamma\sum_a \pi(a|s_1)p(s_2|s_1,a) + ... + \gamma\sum_a \pi(a|s_1)p(s_n|s_1,a) \tag{120}$$

$$= \gamma\sum_a \pi(a|s_1)[1 - p(s_1|s_1,a)] \tag{121}$$

$$= \gamma - \gamma\sum_a \pi(a|s_1)p(s_1|s_1,a) \tag{122}$$

$$< 1 - \gamma\sum_a \pi(a|s_1)p(s_1|s_1,a) \tag{123}$$

since $\gamma \in (0,1)$. This shows that the matrix $A$ is strictly diagonally dominant. A simple application of the Gershgorin circle theorem then conclude that $A$ must be invertible. As a result, we conclude that **the state value function $v_\pi$ is the unique solution to its Bellman equation given policy $\pi$ and MDP dynamics**.

## Optimal Policy and Optimal Value Functions

Now let's take a look to the properties of optimal value functions. State value function $v_\pi$ naturally induces a partial ordering on the set of all policies defined as

$$\pi \leq \pi' \iff \forall s, v_\pi(s) \leq v_{\pi'}(s) \tag{124}$$

The interesting fact is that the **optimal policy** $\pi_*$, the policy better than or equal to any other policy, must exist, though not necessarily unique (easy to raise counterexamples).

We first accept this fact and define the **optimal state value function** as the pointwise maximum w.r.t. all policies at each state.

$$\forall s, v_*(s) \overset{def}{=} \max_\pi v_\pi(s) \tag{125}$$

and similarly the **optimal action value function**

$$\forall s, a, q_*(s,a) \overset{def}{=} \max_\pi q_\pi(s,a) \tag{126}$$

The optimal policy can be constructed easily as a splicing of different "good" policies:

$$\forall s, \pi_* \overset{def}{=} \arg\max_\pi v_\pi(s) \tag{127}$$

For each state, the distribution $\pi_*(a|s)$ is chosen to be the one that achieves the maximum of the state value function at state $s$. To verify whether such construction satisfies the former definition:

$$\forall s, \pi, v_{\pi_*}(s) \geq v_\pi(s) \tag{128}$$

$$\forall \pi, \pi_* \geq \pi \tag{129}$$

**NOTE:** The construction of such optimal policy $\pi_*$ actually only depends on $v_\pi$, which is accessible regardless of the existence of $\pi_*$, so we are not doing a circulation proof here.

There's no doubt that by definition, the optimal state value function is just the state value function of the optimal policy.

$$v_* = v_{\pi_*} \tag{130}$$

We would also expect that the optimal action value function is just the action value function of the optimal policy. The following provides a proof for this fact.

Recall the property that

$$q_\pi(s,a) = \sum_{r,s'} (r + \gamma v_\pi(s'))p(s', r|s,a) \tag{131}$$

gives the following equation by taking maximum w.r.t. $\pi$ on both sides

$$\begin{aligned} q_*(s,a) &= \sum_{r,s'} r \cdot p(s', r|s,a) + \gamma \max_\pi \sum_{r,s'} v_\pi(s')p(s', r|s,a) && (132) \\ &\leq \sum_{r,s'} r \cdot p(s', r|s,a) + \gamma \sum_{r,s'} \max_\pi v_\pi(s')p(s', r|s,a) && (133) \\ &= \sum_{r,s'} (r + \gamma v_{\pi_*}(s'))p(s', r|s,a) && (134) \\ &= q_{\pi_*}(s,a) && (135) \end{aligned}$$

Note that by definition, $q_*$ is the pointwise maximum of $q_\pi$, so $q_*(s,a) \geq q_{\pi_*}(s,a)$. Conclude

$$q_* = q_{\pi_*} \tag{136}$$

These two properties seem to be trivial but are actually important since they tell us that $v_*, q_*$ can be achieved by the optimal policy $\pi_*$, i.e. $v_*, q_*$ are tight upper bounds of $v_\pi, q_\pi$.

Now that it's clear to us that $v_*, q_*$ are value functions for the optimal policy, they have to follow the Bellman equations mentioned above. Take $\pi = \pi_*$ to get the following:

$$v_*(s) = \sum_a \pi_*(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')] \tag{137}$$

$$q_*(s,a) = \sum_{r,s'} p(s',r|s,a)[r + \sum_{a'} \gamma \pi_*(a'|s') q_*(s',a')] \tag{138}$$

These are called Bellman optimality equations. However, they are not that useful since $\pi_*$ still appears in these equations and there's no way for us to directly figure out what $\pi_*$ is. Therefore, we hope to do some transformations to the Bellman optimality equations so that there's no reference to any specific policy.

First notice the relationship between $v_*$ and $q_*$:

$$\begin{aligned} v_*(s) &= \max_\pi v_\pi(s) && (139)\\ &= \max_\pi \sum_a \pi(a|s) q_\pi(s,a) && (140)\\ &\leq \max_\pi \sum_a \pi(a|s) q_*(s,a) && (141)\\ &= \max_a q_*(s,a) && (142) \end{aligned}$$

here the last equation is due to the fact that $\sum_a \pi(a|s) = 1, \pi(a|s) \geq 0$, so for any policy $\pi$, the sum $\sum_a \pi(a|s) q_*(s,a)$ can't be greater than $\max_a q_*(s,a)$ and that such upper bound can be attained. To see why $v_*(s) \geq \max_a q_*(s,a)$ holds, let's first state the **policy improvement theorem**. This theorem works for deterministic policies and gives a way to improve a policy w.r.t. its state value. For any deterministic policy $\pi$, i.e. $\pi(a|s)$ can only take values as either 0 or 1, $\pi(s)$ denotes the action to take with probability 1 at state $s$.

**Theorem 1.** *If $\pi, \pi'$ are any two deterministic policies and $\forall s, q_\pi(s, \pi'(s)) \geq v_\pi(s)$, then $\forall s, v_{\pi'}(s) \geq v_\pi(s)$. Moreover, if the condition is a strict inequality, then the conclusion is also a strict inequality.*

*Proof.* First let's state a fact that $\mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = \pi'(s)] = \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$. It's due to the fact that policy $\pi'$ at state $s$ would result in a deterministic action of $\pi'(s)$.

$$\begin{aligned} \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] &= \sum_a \pi'(a|s) \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a] && (143)\\ &= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = \pi'(s)] && (144) \end{aligned}$$

$$
\begin{aligned}
\forall s, v_\pi(s) &\leq q_\pi(s, \pi'(s)) && (145)\\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = \pi'(s)] && (146)\\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] && (147)\\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1}))|S_t = s] && (148)\\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2})\Big|S_{t+1}, A_{t+1} = \pi'(S_{t+1})]\Big|S_t = s] && (149)\\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2})\Big|S_t = s] && (150)\\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3})\Big|S_t = s] && (151)\\
&\leq ... && (152)\\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...\Big|S_t = s] && (153)\\
&= \mathbb{E}_{\pi'}[G_t|S_t = s] = v_{\pi'}(s) && (154)
\end{aligned}
$$

It's an application of the law of iterated expectation and the most important step is to transform the terms into the expectation w.r.t. $\pi'$.

□

As a result, set $\pi = \pi_*$, a deterministic optimal policy (if exist). We can find out that $\forall s, a, q_{\pi_*}(s, a) \leq v_{\pi_*}(s)$. Otherwise, there always exists a strict improvement of the optimal policy $\pi$ which is deterministic, a contradiction with the definition of the optimal policy. This proves the fact that $\forall s, \max_a q_*(s, a) \leq v_*(s)$.

So the last question left is: why will there always be a deterministic optimal policy? Since optimal policy always exists, take $\pi_*$ as one of them so we can have access to $v_*, q_*$. Construct policy $\pi'$ as

$$\pi'(s) = \arg\max_a q_*(s, a) \tag{155}$$

Under such a deterministic policy, the action that results in the greatest action value is always taken for the current state. From the policy improvement theorem, we know that $\pi'$ is better than any other deterministic policies. However, optimal policy $\pi_*$ can be seen as a convex combination of some deterministic policies, so $v_{\pi'} \geq v_*$. As a result, $v_{\pi'} = v_*$ and $\pi'$ is also an optimal policy. That ends the whole proof. (actually one can see that policy improvement theorem holds for any policy $\pi$ even if it's not deterministic)

## Bellman Optimality Equations

The **compact form of Bellman optimality equation** we have concluded is just

$$v_*(s) = \max_a q_*(s, a) \tag{156}$$

This equation does not necessarily hold if the $*$ is replaced by a general policy $\pi$ and it discloses the structure of the optimal value functions. From this compact form we can infer the frequently used **Bellman optimality equations**.

$$\begin{aligned}
v_*(s) &= \max_a q_*(s,a) && (157)\\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1}+\gamma G_{t+1}|S_t=s,A_t=a] && (158)\\
&= \max_a \sum_{s',r} p(s',r|s,a)[r+\gamma v_*(s')] && (159)\\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1}+\gamma v_*(S_{t+1})|S_t=s,A_t=a] && (160)
\end{aligned}$$

which is a non-linear equation w.r.t. $v_*(s)$ with no relevance to any policy. Such Bellman optimality equation can be solved theoretically knowing the MDP dynamics and the discount rate but it's always hard to solve practically.

Similarly, the Bellman optimality equation for action value function is:

$$\begin{aligned}
q_*(s,a) &= \mathbb{E}_{\pi_*}[R_{t+1}+\gamma G_{t+1}|S_t=s,A_t=a] && (161)\\
&= \sum_{s',r} p(s',r|s,a)[r+\gamma v_*(s')] && (162)\\
&= \sum_{s',r} p(s',r|s,a)[r+\gamma \max_{a'} q_*(s',a')] && (163)
\end{aligned}$$

## Bellman Optimality Equations as Fixed Point Equation for Contraction Mapping

Just to mention, we would expect that the Bellman optimal equation for state value function has **a unique solution**, since optimal policy $\pi_*$ always exists and different optimal policies would share the same $v_*(s)$. To see this point from another point of view, we can make use of the contraction mapping theorem (Banach fixed point theorem) to conclude.

Recall the equation

$$\begin{aligned}
v_*(s) &= \max_a \sum_{s',r} p(s',r|s,a)[r+\gamma v_*(s')] && (164)\\
&= \max_a \mathbb{E}[R_{t+1}+\gamma v_*(S_{t+1})|S_t=s,A_t=a] && (165)
\end{aligned}$$

and organize it in the fixed point form

$$\begin{aligned}
v_*(s) &= Tv_*(s) && (166)\\
T :&\mathbb{R}^{|\mathscr{S}|} \to \mathbb{R}^{|\mathscr{S}|} && (167)\\
Tv(s) &= \max_a \mathbb{E}[R_{t+1}+\gamma v(S_{t+1})|S_t=s,A_t=a] && (168)
\end{aligned}$$

In order to prove the existence and uniqueness of the fixed-point, the contraction mapping theorem immediately comes to our mind. As a result, we only have to find a metric on $\mathbb{R}^{|\mathscr{S}|}$ (this is a finite-dimensional space since the state value function can be seen as a finite dimensional vector) such that $T$ is a contraction mapping. Luckily, such metric can be induced by the infinity norm $||\cdot||_\infty$. The next lemma proves that such operator $T$ is actually a contraction mapping under the metric induced by $||\cdot||_\infty$.

**Lemma 1.** *Consider metric space* $\left(\mathbb{R}^{|\mathscr{S}|}, ||\cdot||_\infty\right)$, *and the operator* $T$ *defined above, then* $\exists 0 \leq k < 1$ *such that* $\forall v, v' \in \mathbb{R}^{|\mathscr{S}|}, ||Tv - Tv'||_\infty \leq k||v - v'||_\infty$.

*Proof.*

$$\begin{aligned}
||Tv - Tv'||_\infty &= \sup_s |Tv(s) - Tv'(s)| && (169)\\
&= \sup_s |\max_a \mathbb{E}[R_{t+1} + \gamma v(S_{t+1})|S_t = s, A_t = a] - \max_a \mathbb{E}[R_{t+1} + \gamma v'(S_{t+1})|S_t = s, A_t = a]| && (170)\\
&\leq \sup_s \max_a |\mathbb{E}[\gamma v(S_{t+1})|S_t = s, A_t = a] - \mathbb{E}[\gamma v'(S_{t+1})|S_t = s, A_t = a]| && (171)\\
&= \sup_s \max_a \gamma \mathbb{E}[|v(S_{t+1}) - v'(S_{t+1})|\Big|S_t = s, A_t = a] && (172)\\
&\leq \gamma \sup_s \max_a ||v - v'||_\infty && (173)\\
&= \gamma ||v - v'||_\infty && (174)
\end{aligned}$$

□

As a result, the contraction mapping theorems holds and the Bellman optimality equation for state value function has a unique solution $v_*(s)$, i.e. the Bellman optimality equation is the characterization of the optimal value functions.

## Summary

It should be clear for now that when we are solving a RL problem, we do not care about $v_\pi$ and $q_\pi$ for a specific policy $\pi$ since there are uncountably many policies $\pi$. In other words, although for any fixed policy $\pi$ Bellman equations offer an easy way to solve out $v_\pi, q_\pi$ since the equations are linear, it's unrealistic for us to find out the optimal value functions $v_*, q_*$ by applying their definitions. Instead, the non-linear Bellman optimality equation is of our true concern. It has nothing to do with policy and only requires knowing the MDP dynamics $p$.

So what shall we do after knowing/estimating the optimal value functions $v_*, q_*$? If we know $q_*$, the best actions to pick at current state $S_t$ are just the actions $a$ such that $q_*(S_t, a)$ is maximized. If we know $v_*$, the best actions to pick at current state $S_t$ are all the actions $a$ such that the maximum in the Bellman optimality equation $v_*(S_t) = \max_a \sum_{s',r} p(s', r|S_t, a)[r + \gamma v_*(s')]$ is attained. It's easy to see that the two ways of selecting best actions give the same set of actions and any convex combination of those actions gives an optimal policy $\pi_*$ (that's why $\pi_*$ is not unique).

In brief, what we have done above can be concluded into following items:

- Estimating $v_*, q_*$ is the core task in RL

- $v_*, q_*$ satisfies non-linear Bellman optimality equations that can be solved if MDP dynamics $p$ is known
- After getting estimates of $v_*, q_*$, the RL problem is solved since the best action at each turn can be figured out

While the concepts of MDP are all clear, many problems remain unsolved. Directly solving Bellman optimality equations by fixed point iteration is always not practically feasible since MDP dynamics are always unknown, especially in cases where the reward itself is random given the states and actions. Nevertheless, the method won't work for an MDP with too many states even if $p$ is known. In fixed point iteration schemes, the vector has its length equal to the number of states, and for games like chess, there are at least $3^{361}$ number of states! As a result, further studies are necessary to develop various methods for the estimation of optimal policy/optimal value functions.

## Gridworld Model as an Example

Let's see how optimal value functions are found in the **Gridworld** model and figure out the best actions at each state. Although no methods have been introduced to estimate optimal value function $v_*$, we have mentioned above that **the solution to the Bellman optimality equation can be viewed as the fixed point of a contraction mapping**. So why not use fixed point iteration to approximate $v_*$?

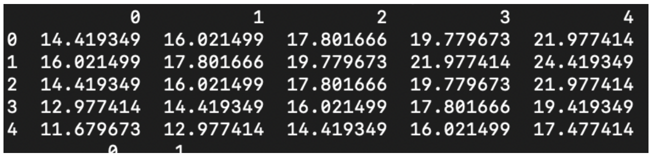
An initial value $v_0(s)$ would be constructed, and the fixed point iteration goes like:

$$\forall s \in \mathscr{S}, v_{n+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_n(s')] \tag{175}$$

It would be expected that

$$\forall s \in \mathscr{S}, v_n(s) \to v_*(s) \ (n \to \infty) \tag{176}$$

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

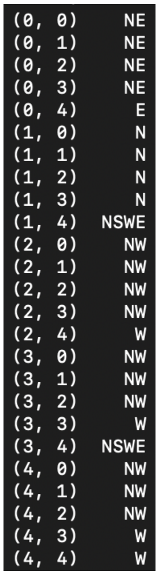


```
           0          1          2          3          4
0  14.419349  16.021499  17.801666  19.779673  21.977414
1  16.021499  17.801666  19.779673  21.977414  24.419349
2  14.419349  16.021499  17.801666  19.779673  21.977414
3  12.977414  14.419349  16.021499  17.801666  19.419349
4  11.679673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

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 we have proved.

Based on the optimal state value function derived, it's easy for us to figure out the best actions to take at each state as stated above. The results are demonstrated in Fig. 11 (this is exactly the same as Figure 3.5 in the textbook).



```
(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: Best actions at each state of the 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).

## Another MDP Model to Illustrate the power of Bellman Optimality Equations

Let's look at exercise 3.22 for a different MDP, shown in Fig. 12. The only two policies for choice are $\pi_l, \pi_r$, the policy that always choose to go left and the policy that always choose to go right. For different values of $\gamma$, 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.

$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')] \tag{177}$$

Let's first write out the Bellman optimality equations for states $s_2, s_3$:

$$v_*(s_2) = \gamma v_*(s_1) \tag{178}$$

$$v_*(s_3) = 2 + \gamma v_*(s_1) \tag{179}$$

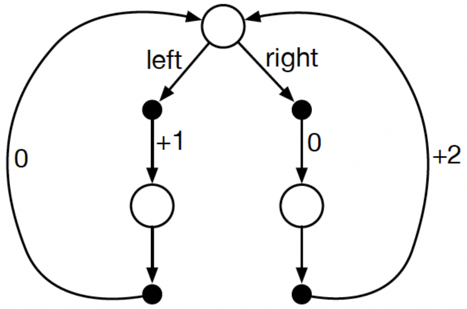


Figure 12: Another MDP

There's 3 states, but decisions only have to be made at the top state, where there are two actions to choose from. Deterministic rewards are labelled above edges. There's only two deterministic policies $\pi_l, \pi_r$.

The Bellman optimality equations for state $s_1$ has a maximum inside:

$$v_*(s_1) = \max\{1 + \gamma v_*(s_2), \gamma v_*(s_3)\} \tag{180}$$

Now, solve the Bellman optimality equations to find that the critical value of $\gamma$ is $\gamma_*$ such that $1 + \gamma_* v_*(s_2) = \gamma_* v_*(s_3)$.

$$1 + \gamma_*^2 v_*(s_1) = 2\gamma_* + \gamma_*^2 v_*(s_1) \tag{181}$$

$$\gamma_* = \frac{1}{2} \tag{182}$$

This is telling us that when the discount rate $\gamma = \frac{1}{2}$, both actions achieves the maximum in the Bellman optimality equations, so both $\pi_l$ and $\pi_r$ are optimal policies. When $0 \leq \gamma < \frac{1}{2}$, only the action of going left achieves the maximum in the Bellman optimality equations, so only $\pi_l$ is the optimal policy. When $\frac{1}{2} < \gamma < 1$, only $\pi_r$ is the optimal policy.

# Dynamic Programming (DP) Methods

The DP methods are direct and natural extensions of the properties we have proved above. In this section regarding DP methods, it's assumed that the MDP dynamics $p$ is always known and that there are not too many states (so sweeping through all states would be practically feasible).

## Policy Evaluation

The policy evaluation task refers to the task that we hope to get value functions $v_\pi, q_\pi$ given a policy $\pi$. Let's first deal with the task of getting $v_\pi$ from scratch.

It's natural to immediately recall the Bellman equations

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \tag{183}$$

We are happy to see that this equation is a linear system w.r.t. $v_\pi(s)$ and it has a unique solution. This equation certainly can be organized in its matrix form and be solved easily. However, the Bellman equation here provides us with a natural iteration scheme of equation solving.

Note that this equation is actually a fixed-point equation for $v_\pi(s)$, so why not use fixed point iteration? Let's first do some analysis to find out whether fixed point iteration is a good scheme for this problem. (Similar to what we have done for Bellman optimality equations)

Set the operator $T : \mathbb{R}^{|\mathscr{S}|} \to \mathbb{R}^{|\mathscr{S}|}$ to be that

$$\begin{aligned} Tv(s) &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v(s')] && (184)\\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v(S_{t+1})|S_t = s] && (185) \end{aligned}$$

The Bellman equation becomes

$$v_\pi(s) = Tv_\pi(s) \tag{186}$$

Now view $T$ as a linear operator on metric space $\left(\mathbb{R}^{|\mathscr{S}|}, ||\cdot||_\infty\right)$,

$$\begin{aligned} ||Tv(s) - Tv'(s)||_\infty &= \gamma ||\mathbb{E}_\pi[v(S_{t+1}) - v'(S_{t+1})\Big| S_t = s]||_\infty && (187)\\ &\le \gamma ||v - v'||_\infty && (188) \end{aligned}$$

Once again it's a contraction mapping since $0 \le \gamma < 1$, that's why the fixed point iteration will always converge. The **iterative policy evaluation** is stated as:

$$v_{n+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_n(s')] \tag{189}$$

and we would expect to see

$$\forall s, v_n(s) \to v_\pi(s)\ (n \to \infty) \tag{190}$$

So what about the action value function $q_\pi$? Can we still use the similar method and will the convergence still be guaranteed? Recall the Bellman equation for $q_\pi$:

$$q_\pi(s,a) = \sum_{r,s'} p(s',r|s,a)[r + \sum_{a'} \gamma\pi(a'|s')q_\pi(s',a')] \tag{191}$$

with the **iterative policy evaluation**:

$$q_{n+1}(s,a) = \sum_{r,s'} p(s',r|s,a)[r + \sum_{a'} \gamma\pi(a'|s')q_n(s',a')] \tag{192}$$

Set the operator $T : \mathbb{R}^{|\mathscr{S}|} \times \mathbb{R}^{|\mathscr{A}|} \to \mathbb{R}^{|\mathscr{S}|} \times \mathbb{R}^{|\mathscr{A}|}$ to be that

$$\begin{aligned} Tq(s,a) &= \sum_{r,s'} p(s',r|s,a)[r + \sum_{a'} \gamma\pi(a'|s')q(s',a')] && (193)\\ &= \mathbb{E}_\pi[R_{t+1} + \gamma \sum_{a'} \pi(a'|S_{t+1})q(S_{t+1},a')|S_t = s, A_t = a] && (194) \end{aligned}$$

The Bellman equation becomes

$$q_\pi(s,a) = Tq_\pi(s,a) \tag{195}$$

Now view $T$ as a linear operator on metric space $\big(\mathbb{R}^{|\mathscr{S}|\times|\mathscr{A}|}, ||\cdot||_\infty\big)$,

$$\begin{aligned} ||Tq(s,a) - Tq'(s,a)||_\infty &= \gamma||\mathbb{E}_\pi[\sum_{a'} \pi(a'|S_{t+1})[q(S_{t+1},a') - q'(S_{t+1},a')]\Big|S_t = s, A_t = a]||_\infty && (196)\\ &\le \gamma||q - q'||_\infty \mathbb{E}_\pi[\sum_{a'} \pi(a'|S_{t+1})\Big|S_t = s, A_t = a] && (197)\\ &= \gamma||q - q'||_\infty && (198) \end{aligned}$$

For the same reason, $T$ is a contraction mapping, convergence is ensured and we would expect to see

$$\forall s, a, q_n(s,a) \to q_\pi(s,a)\ (n \to \infty) \tag{199}$$

One last remark is that when setting initial values for the iteration, if the task is an episodic task, the terminal state must be given value 0 (means that no reward will be further provided). For all states in continuing tasks or normal states in episodic tasks, this restriction does not exist.

## Policy Iteration

As stated at the section of the policy improvement theorem, for any policy $\pi$, $\pi'(s) = \arg\max_a q_\pi(s,a)$ is always a deterministic policy as the improvement of $\pi$ and it's thus called the **greedy policy w.r.t.** $\pi$. In the case where $\pi$ is already the optimal policy, $\pi'(s) = \arg\max_a q_*(s,a)$ has also been proved to be the optimal policy, though not necessarily the same. As a result, this provides a way to iteratively improve the policy until the policy is optimal.

For any initial policy $\pi_0$, policy evaluation is applied to figure out $v_{\pi_0}$. After that, the greedy policy $\pi_1$ is constructed w.r.t. $q_{\pi_0}$ as an improvement of $\pi_0$. Iteratively, the policy iteration ensures that the policy is always being strictly improved until it becomes the optimal policy and the iteration stops. There's one question left: how to figure out the greedy policy w.r.t. $\pi$ from $v_\pi$ instead of $q_\pi$? The answer is easy since $q_\pi$ can always be represented by $v_\pi$ and the dynamics $p$ as $q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$.

To conclude, the **policy iteration for** $v_\pi$ proceeds like:

- For the current policy $\pi_k$
- Policy evaluation to get $v_{\pi_k}$: $v_{n+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_n(s')]$ as a fixed point iteration based on the Bellman equations, have to wait for the convergence of $v_n$ to $v_{\pi_k}$
- Policy improvement to get a new improved deterministic policy $\pi_{k+1}$ with $\pi_{k+1}(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$
- Repeat the operations above until convergence $\pi_k \to \pi_*$ 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 $||v_{\pi_k} - v_{\pi_{k+1}}||_\infty < \varepsilon$.

The convergence of such algorithm is ensured completely by the policy improvement theorem. If $\pi$ is not optimal, the greedy policy w.r.t. $\pi$ must be a strict improvement.

Actually, $q_\pi$ can also be used to conduct policy iteration and the frame is very much alike. The **policy iteration for** $q_\pi$ proceeds like:

- For the current policy $\pi_k$
- Policy evaluation to get $q_{\pi_k}$: $q_{n+1}(s,a) = \sum_{r,s'} p(s',r|s,a)[r + \sum_{a'} \gamma \pi(a'|s') q_n(s',a')]$ as a fixed point iteration based on the Bellman equations, have to wait for the convergence of $q_n$ to $q_{\pi_k}$
- Policy improvement to get a new improved deterministic policy $\pi_{k+1}$ with $\pi_{k+1}(s) = \arg\max_a q_\pi(s,a)$
- Repeat the operations above until convergence $\pi_k \to \pi_*$ 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 $q_*$ and a good stopping criteria would be $\forall s, ||q_k(s,\cdot) - q_{k+1}(s,\cdot)||_\infty < \varepsilon$.

## Value Iteration

Actually, we have already made use of value iteration techniques in the Gridworld example for Bellman optimality equations. It's just a fixed point iteration of the non-linear Bellman optimality equations and we will directly get $v_*$. The algorithm proceeds like:

$$\forall s \in \mathscr{S}, v_{n+1}(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_n(s')] \tag{200}$$

The convergence is ensured by the property of contraction mapping proved above. Similarly, $q_*$ can be derived according to its Bellman optimality equations:

$$\forall s \in \mathscr{S}, a \in \mathscr{A}(s), q_{n+1}(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} q_n(s',a')] \tag{201}$$

The advantage of value iteration method is that it costs less time than policy iteration generally since policy iteration has to do policy evaluation within the loop.

## Car Rental Example

Two locations for a rental company. When a customer comes, if a car is available, rent out and get 10 (dollars), if no car available, lose business. Cars are available the day after returned. Overnight movings between two locations are allowed, pay 2 per car. The numbers of rental requests per day in the first and second locations are $P(3), P(4)$ random variables, and the number of returns per day are $P(3), P(2)$ random variables. There can be no more than 20 cars at each location at any time (additional cars just disappear), overnight movings for at most 5 cars.

First let's turn this problem into a MDP by specifying the states, actions and rewards. The rewards are just the amounts of profit and the time is counted by days. The states are tuples consisting of the number of cars in the first location and the number of cars in the second location at the end of the day before overnight moving happens, i.e. $\mathscr{S} = \{(s_1, s_2)|s_1, s_2 \in \mathbb{Z}, 0 \leq s_1, s_2 \leq 20\}$. The actions are the decisions made about how many cars (net) to move from one location to another, i.e. $\mathscr{A} = \{-5, -4, ..., -1, 0, 1, ..., 5\}$ where 1 means 1 car is moved from the first to the second location and negative integers means moving in the opposite direction. If the action space is written in a form with a dependency on the current state, $\mathscr{A}(s_1, s_2) = \{a|a \in \mathscr{A}, -s_2 \leq a \leq s_1\}$.

The action changes the state is the way that if the current state is $S_t = (S_t^1, S_t^2)$, the action is $A_t$, the number of rental requests on day $t$ for location $i$ is $RT_t^i$ and the number of returned cars on day $t$ for location $i$ is $RE_t^i$, then the next state would be affected by overnight movings, next day's rented cars and today's returned cars. On the next morning, there will be

$$(\min\left\{S_t^1 - A_t + RE_t^1, 20\right\}, \min\left\{S_t^2 + A_t + RE_t^2, 20\right\}) \tag{202}$$

cars in the first and the second locations. As a result, the rental will bring with overall revenues

$$10 \times [\min\left\{\min\left\{S_t^1 - A_t + RE_t^1, 20\right\}, RT_{t+1}^1\right\} + \min\left\{\min\left\{S_t^2 + A_t + RE_t^2, 20\right\}, RT_{t+1}^2\right\}] \tag{203}$$

Note that a number of $-2|A_t|$ would be added to the reward of this turn as the cost of overnight moving. As a result, the reward is

$$R_{t+1} = 10 \times [\min\left\{\min\left\{S_t^1 - A_t + RE_t^1, 20\right\}, RT_{t+1}^1\right\} + \min\left\{\min\left\{S_t^2 + A_t + RE_t^2, 20\right\}, RT_{t+1}^2\right\}] - 2|A_t| \tag{204}$$

and the next state should be

$$RENTED_{t+1}^1 = \min\left\{\min\left\{S_t^1 - A_t + RE_t^1, 20\right\}, RT_{t+1}^1\right\} \tag{205}$$
$$RENTED_{t+1}^2 = \min\left\{\min\left\{S_t^2 + A_t + RE_t^2, 20\right\}, RT_{t+1}^2\right\} \tag{206}$$
$$S_{t+1} = (\min\left\{S_t^1 - A_t + RE_t^1, 20\right\} - RENTED_{t+1}^1, \min\left\{S_t^2 + A_t + RE_t^2, 20\right\} - RENTED_{t+1}^2) \tag{207}$$

This car rental problem is organized as a continuing finite MDP. Although it's presented in the textbook as an example of policy iteration methods, I think it would already be difficult to figure out the MDP dynamics $p$ despite the simplicity of this problem. The difficulty lies in the boundary conditions, e.g. the maximum number of cars allowed, the comparison between available cars and rental requests, and the randomness of rewards and state evolution, e.g. both the rewards and the next state involves Poisson random variables in a non-linear way.

As we have expected, if the model is simple and the dynamics $p$ can be easily derived, e.g. the Gridworld model, DP methods work pretty well and are extremely simple to implement. However, for slightly more complicated problems, it's practically impossible to compute $p$, and DP methods can do nothing without knowing $p$.

## Gambler's Problem

The gambler's problem is much nicer for DP methods to apply. 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, wins as much as his stake. If the coin comes up tails, lose all stakes. Note that the gambler's stake has to satisfy the condition that the amount of money he owns now plus the stake does not surpass 100.

To specify the problem as a MDP, the reward is NOT the amount of money BUT whether the gambler wins the game (reward is 0 on all transitions and 1 iff gambler wins the game) and the time is the number of turn. The state set would be $\mathscr{S} = \{0, 1, ..., 99, 100\}$ indicating the amount of money that gambler currently owns before putting down the stake. Note that here $0, 100$ are two special terminal states, 0 is always given state value 0 and 100 is always given state value 1 (consistent with the reward setting). The set of actions is $\mathscr{A}(s) = \{0, 1, ..., \min\{s, 100 - s\}\}$.

Under such undiscounted setting, state value function $v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\mathbb{I}_{win}|S_t = s] = \mathbb{P}_\pi(win|S_t = s)$ has the meaning of the **probability** of winning given policy $\pi$ and the current amount of money. The only parameter for this game is $p_h$, the probability to get heads in a single coin flip.

This game is an **episodic, nondiscounted**, finite MDP and its dynamics $p(s',r|s,a)$ can be easily found. If heads appears, $s'=s+a$ and $r=1$ together with the end of the game come if and only if $s'=100$. If tails appears, $s'=s-a$ and $r=0$, and it's the end of the game if $s'\le 0$.

$$p(s+a,0|s,a) = p_h \ (s+a<100) \tag{208}$$
$$p(s+a,1|s,a) = p_h \ (s+a=100) \tag{209}$$
$$p(s-a,0|s,a) = 1-p_h \tag{210}$$
$$p(s,0|s,0) = 1 \tag{211}$$

The optimal state value function and the optimal policy derived using value iteration method is exhibited in Fig. 13, 14, 15 for different values of $p_h$. Note that the optimal policy is not necessarily unique, so all possible best actions are printed out for each state.

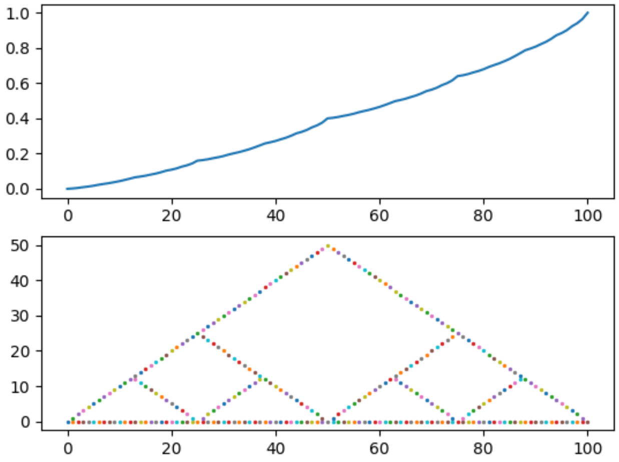


Figure 13: Optimal state value function and optimal policy for gambler's problem with $p_h = 0.4$

For the upper subplot, the x-axis is the state $s$ and the y-axis is the state value $v_*(s)$.
For the lower subplot, the x-axis is the state $s$ and the y-axis is the best action at this state, i.e. the action that achieves maximum in Bellman optimality equation for $v_*$.

Note that the lower subplot of Fig. 13 contains the deterministic optimal policy exhibited in Figure 4.3 in the textbook. As stated in the earlier context, this enables us to find the set of all optimal policies. (just have to make sure that if $\pi_*(a|s)>0$ for some state $s$ then action $a$ achieves the maximum in the Bellman optimality equation for $v_*$)

The interesting fact is that when $p_h<\frac{1}{2}$, the best actions appear like the lower subplot in Fig. 13, while if $p_h=\frac{1}{2}$, all available actions are best actions, and the best actions appear as a left-tailed biased shape if $p_h>\frac{1}{2}$.

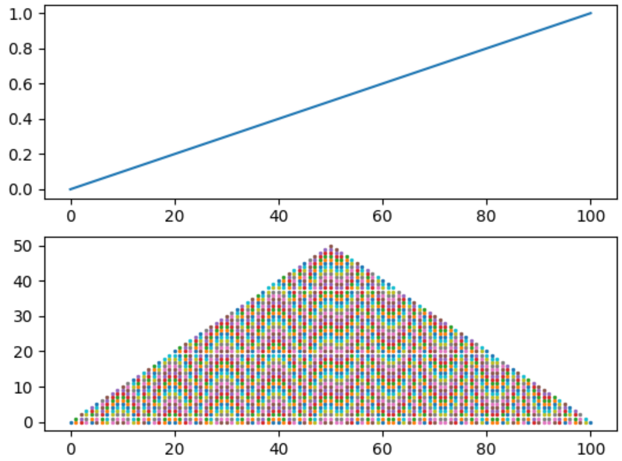


Figure 14: Optimal state value function and optimal policy for gambler's problem with $p_h = 0.5$

For the upper subplot, the x-axis is the state $s$ and the y-axis is the state value $v_*(s)$.
For the lower subplot, the x-axis is the state $s$ and the y-axis is the best action at this state, i.e. the action that achieves maximum in Bellman optimality equation for $v_*$.

For $p_h = \frac{1}{2}$, the expected return of each gamble is always 0 regardless of the action chosen. As a result, all actions are "fair bets" and there should be no preferences.

When $p_h < \frac{1}{2}$, it's harder to win the bet in each turn, so whenever there's a chance to win the game directly in one turn (when state is no less than 50), it's always the best action. How shall we explain the phenomenon that there may exist multiple best actions for the same state? For example, if the current state is 51, why is action 1 also the best action? The answer might sound weird but by taking action 1, we are hoping to lose the bet this turn in order to get to state 50. After getting to state 50, we are able to take all money as stake and to see whether we will win or lose in a single bet. It's important to understand that when $p_h < \frac{1}{2}$, we hope the game to end as soon as possible since longer time causes the law of large numbers to work more effectively, lowering the possibility to win the game. At this point, it should be easy to understand why for all $p_h < \frac{1}{2}$, the set of best actions are all the same as that in Fig. 13.

To get a little deeper into the optimal policy in Fig. 13, ignoring action 0, whenever there are multiple best actions for a single state, the largest action always aims to win the bet in the turn while the smallest action always aims to lose the bet in the turn. The "critical states" are 87.5, 75, 62.5, 50, 25, 37.5, 12.5. 50 is easy to understand since we are able to bet in a single turn to see the final result of the game. 25 and 75 are also natural since they are midpoints of the intervals $[0, 50]$, $[50, 100]$ and by reaching them, it's possible for us to reach 50, reducing the number of turns in this game as much as possible. A natural question is that why don't we take the midpoint of $[0, 12.5]$ and do further bisections of the intervals? (This is a question worth thinking about, recall that the objective is to end the game as soon as possible, so it's a good idea to do further partitions and compare with the current scheme)

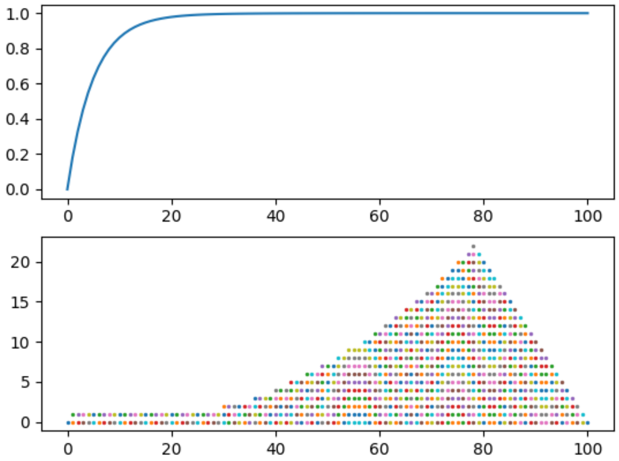


Figure 15: Optimal state value function and optimal policy for gambler's problem with $p_h = 0.55$

For the upper subplot, the x-axis is the state $s$ and the y-axis is the state value $v_*(s)$.
For the lower subplot, the x-axis is the state $s$ and the y-axis is the best action at this state, i.e. the action that achieves maximum in Bellman optimality equation for $v_*$.

When $p_h > \frac{1}{2}$, it's exactly the opposite. We hope that the game would last longer so the law of large number works. That's why taking action 1 is always the best action for all state. As long as the stake is small enough, law of large numbers finally leads us to the winning of the game. When the state gets larger, we would have more best actions since the ability to take risk also grows. It's natural to expect that different $p_h$ would result in different set of best actions for each state, since for larger $p_h$, we are more confident that we cannot lose the game and we should be willing to put more money on the desk even if we are not having much money with us for the time being.

## Summary

Mainly two simple DP methods: policy iteration and value iteration.

Policy iteration tries to find the optimal policy and optimal state value function by focusing on improving the policy. Firstly, we have to do policy evaluation to find $c_\pi$ for policy $\pi$ (fixed point iteration for Bellman equation of $v_\pi$), then policy improvement, finding a greedy deterministic policy for $v_\pi$ is found (the action that achieves maximum of $q_\pi$ for fixed current state) and updated as the new policy. Actually, it's like competition & cooperation between the policy and the value function. **Generalized policy iteration (GPI)** refers to the policy iteration that enables the the interaction between policy evaluation and policy improvement, i.e. improving the policy before policy evaluation is completely done or evaluating the new policy before policy improvement is completely done. Such methods are expected to work more efficiently and still has the convergence property.

Value iteration is a fixed point iteration of Bellman optimality equation for $v_*$. After figuring out $v_*$, the set of best actions is constructed by finding out which of those actions achieve the maximum in the Bellman optimality equation.

The DP methods are efficient for simple RL problems with small state space and concise dynamics. However, the main drawbacks are that sometimes the dynamics is too complicated to be written out in a clear style and that state space is always large and it's impossible to sweep through all states.

# Monte Carlo (MC) Methods

After introducing DP methods, it shall be quite obvious that the advantage of MC methods is that **experience** is the only thing needed. In other words, we don't have to figure out the dynamics $p$, we don't have to sweep through all states, we just need to construct the described RL process and do simulations. Experience will tell us everything we want to know. Note that since MC can deal with any expectation estimation problems and $v_\pi, q_\pi$ are both conditional expectations, MC perfectly fits with the situation in RL.

## First-visit MC

Actually, we have talked about using MC to approximate the value function $v_\pi$ for a certain policy $\pi$ in the previous context already. Fig. 9 is just produced by applying the every-visit MC method. The visit of an action or a state just means the appearance of an action or a state in the simulation process. A first visit of state $s$ just means the first time a simulation generates state $s$.

The **first-visit MC** method for estimating $v_\pi$ proceeds as:

- Do simulations for the RL problem with policy $\pi$
- For each state $s$, if it's the first time to visit state $s$, i.e. $T \overset{def}{=} \inf\{t|S_t = s\}$, take record of the return $G_T$ at that time $T$ (the simulation has to be stopped before the return can be calculated since the return is a discounted sum of future rewards)
- The sample mean of all $G_T$ recorded is just an estimate for $\mathbb{E}_\pi[G_t|S_t = s] = v_\pi(s)$

The **every-visit MC** method is just recording all visits to state $s$, getting the returns and taking sample average.

Actually, the first-visit MC is the traditional MC since different observations of the state, action, reward chains are independent and the convergence is ensured by the law of large numbers. However, the every-visit MC also has a guaranteed convergence property.

By adopting a similar scheme, $q_\pi$ can also be approximated by first-visit/every-visit MC in the sense that the state-action pair $(s, a)$ is kept track of.

## 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 stick (stop drawing card). The player loses the game as soon as the sum exceeds 21 and when he sticks the dealer begins to do the same thing. The dealer loses the game as soon as the sum exceeds 21 and when he sticks he compares the sum with the player to see whether it's a draw/win/lose.

Blackjack can be formulated as a undiscounted episodic finite MDP. Let's assume that the dealer sticks on any sum greater or equal to 17 otherwise hits (the most conservative strategy), so the problem now becomes an RL problem for the player. Rewards are given based on draw/win/lose to be 0/1/-1 respectively and any transitions within an episode receive no rewards. Assume there are infinitely many cards and ace is said to be usable if it works as 11 and nonusable if it works as 1. Note that the ace in the dealer's first two cards must work as 11 unless there are two aces that exceeds 21, and all drawn aces of the dealer work as 1. The player, however, can decide what value each ace takes. It's quite obvious that if there is ace in the first two cards of the player, at least one of them should work as 11 since there is no risk of exceeding 21 if the sum is less than 12. (when the sum is 11, if we get an ace, the ace should be nonuseable, so the sum can't exceed 21)

In detail, the state space should be $\mathscr{S} = \{(s,d)|s,d \in \mathbb{N}, 12 \leq s \leq 21, 1 \leq d \leq 10\}$, where $s$ stands for the sum of cards in the player's hand 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, we just hit without thinking. The action space contains two actions: $\mathscr{A} = \{hit, stick\}$. Assume we are adopting the policy that the player sticks if and only if the sum is 20 or 21 (a risky strategy). Let's figure out the estimated $v_\pi$ for this strategy $\pi$, refer to Fig. 17 and **??** for details.

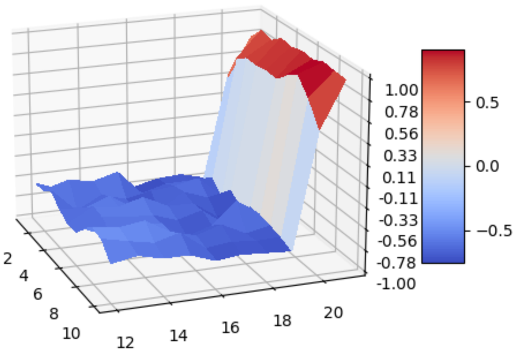


Figure 16: State value function estimated by Monte Carlo

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.
10000 iterations for Monte Carlo estimation, the estimation is coarse.

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 policy is that the player continues hitting unless the sum is no less than 20. As a result, when the player chooses to hit with a sum near 20, it's often the case that he will go busted and lose the game. It can also be seen that there's 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 conditionally likely for him to win immediately (note that the probability of drawing 10 is $\frac{4}{13}$ in Blackjack, and an ace with 10 is a Blackjack). As a result, the moment we see that the dealer has the first card as ace, we are already more likely to lose given this fact.

In the Blackjack example, the first-visit MC actually has no difference from the every-visit MC since for each

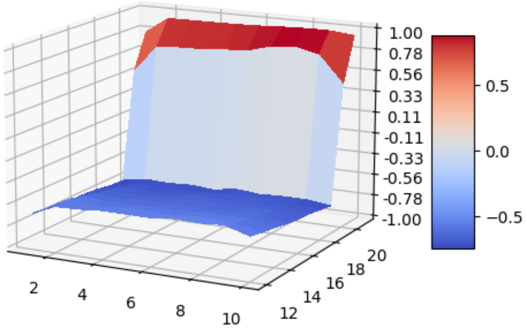


Figure 17: State value function estimated by Monte Carlo

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.
500000 iterations for Monte Carlo estimation, the estimation is pretty good.

game the state is always changing and never repeats (the player's sum is strictly increasing).

## Some Tips for MC Methods

We have mentioned above for the MC scheme estimating action value function $q_\pi$. The fact is that so far we have always been estimating the state value function or the optimal state value function to find the best action because of its simplicity. However, by recalling the procedure we would find out that searching for best actions based on $v_*$ would require us knowing the MDP dynamics $p$ (the action that achieves the maximum in the Bellman optimality equation for $v_*$). What a bad news! MC has its advantage that it does not need to know dynamics $p$ but now we are depending on $p$ once more.

However, we immediately notice that **the best action can be figured out with an estimate for $q_*$ without knowing dynamics $p$**. For each state $s$, the best action is just the action $a$ such that $q_*(s, a)$ is the largest. Besides, the MC scheme can be easily extended for the estimation of $q_\pi$ by recording both the states and the actions. That's the exact reason why we will be focusing on estimating the action value function in the following context with MC methods.

The another problem is that when evaluating value functions, there may be possibility that there's no observations of a certain action when the policy only places a very little probability on such action. That is, we have to **maintain exploration** as mentioned above to ensure that there is enough knowledge for each state-action pair. One way is to adopt the **exploring start**, i.e. each state-action pair has a positive probability of being the initial pair of state-action. By doing this, each pair is guaranteed to be visited infinitely often if we are doing infinitely many simulations.

## MC Control

Now that we want to estimate $q_*$ in order to solve the RL problem, a GPI thought can be combined with MC. Since each GPI has the policy evaluation and the policy improvement parts, MC will help us with policy evaluation (which is done by DP in the former section), and a greedy deterministic policy is still selected as the improved policy.

So now the algorithm proceeds like:

- Fix a policy $\pi$
- For each generated episode $S_0, A_0, R_1, ...$, calculate the returns at each time, then find out the time of the first visit to state-action pair $(s, a)$ and record all returns at such time
- Do the simulation above for many times as MC simulations to estimate $q_\pi(s, a)$ for each state $s$ and action $a$
- Construct the greedy deterministic policy $\pi'(s) = \arg\max_a q_\pi(s, a)$ as an improvement of $\pi$
- Iterate until $\pi \to \pi_*, q_\pi \to q_*$

Some obvious problems with this algorithm are that: (i): the algorithm generates deterministic policies as improvements, so there's a large possibility that some states or actions are never visited in the MC simulation (ii): the algorithm costs to much time by performing the complete MC simulation for each time of policy evaluation, it may take a long time to converge.

The first problem is easily solved by adding an exploring start, i.e. assign positive probability for each state-action pair to appear as the initial state and the initial action. The second problem requires inspirations coming from GPI. Since policy evaluation now costs much time, why don't we improve the policy before policy evaluation is completed? An existing example would be the value iteration method. Although we are viewing value iteration as fixed-point iteration in the previous context, let's recall the value iteration:

$$v_{n+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_n(s')] \tag{212}$$

and the policy evaluation as DP:

$$v_{n+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_n(s')] \tag{213}$$

It's natural to find that if we replace the policy $\pi$ in the policy evaluation by a greedy deterministic policy $\pi'(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r+\gamma v_n(s')]$, it just becomes the value iteration. In detail, the maximized expression $\sum_{s',r} p(s', r|s, a)[r + \gamma v_n(s')]$ is just the expression within a single iteration in the policy evaluation. To sum up, the **value iteration can be viewed as GPI** because it evaluates and improves the policy at the same time. To be specific, for policy $\pi$, policy evaluation is done only for **one step towards** $v_\pi$ (so the error is still large, since such evaluation would require a period of time to converge) and a policy improvement follows, finding the greedy policy w.r.t. such poorly estimated "value function". By recognizing that value iteration is actually one-step evaluation plus one-step improvement, it would be surprising that such method actually is a fixed point iteration and converges really quickly. However, same strategy can be applied with policy evaluation techniques replaced by MC simulation.

As a result, **the first-visit MC ES for estimating** $\pi_*$ goes like (ES for exploring start and policy evaluation is done for only one step for each policy):

- Exploring start, each state-action pair has positive probability of being selected as the initial state-action
- For a fixed policy $\pi$, generate episode $S_0, A_0, R_1, ...$, calculate the returns at each time, then find out the time of the first visit to state-action pair $(s, a)$ and record all returns at such time in a list
- Update the estimate for $q_\pi(s, a)$ as the average of all values in the list (only one-step evaluation since only one return value for $(s, a)$ is added to the list)
- Construct the greedy deterministic policy $\pi'(s) = \arg\max_a q_\pi(s, a)$ as an improvement of $\pi$
- Iterate until $\pi \to \pi_*, q_\pi \to q_*$

The difference is that originally for each policy $\pi$, a complete MC procedure is used to **accurately** evaluate $q_\pi$ and now for each policy, a single episode is generated to **push the estimate for** $q_\pi$ **toward the true** $q_\pi$ **a little bit**. Note that here a list to store the return value for each state-action pair $(s, a)$ is actually not necessary since what we care about is just the mean and it could be incrementally recorded, i.e. let $m_t(s, a)$ denote the sample mean of the return of the first-visit to $(s, a)$ up to and include time $t$, $N_t(s, a)$ denote the number of appearances of the first-visit to $(s, a)$ up to and include time $t$ and $G(s, a)$ denote the return of the first-visit to $(s, a)$ newly found. Then it's obvious that

$$m_t(s, a) = \frac{N_{t-1}(s, a)}{N_t(s, a)} m_{t-1}(s, a) + \frac{1}{N_t(s, a)} G(s, a) \tag{214}$$

This MC ES algorithm is ensured not to converge to any suboptimal policy. Otherwise, the estimate for $q_\pi$ would be very close to the action value function of that suboptimal policy. However, the greedy policy is proved to be a strict improvement if the policy is not yet optimal, providing a better policy. The convergence property of MC ES has not been proved yet, which means that may exist possibility that this algorithm does not converge.

# Policy Gradient Method

Policy gradient methods focus on updating the policy $\pi$ instead of forming any estimation on the value function. This is especially useful when one has to deal with a very large state space and action space when the estimation of value function becomes infeasible (at least one estimation has to be made for each state). In order to form a large enough family of policies to explore, one always parametrize the policy as $\pi(a|s,\theta)$ with parameter $\theta \in \mathbb{R}^{d'}$.

## Policy Gradient Theorem

The question here is how to find a way to update $\theta$ such that the policy converges to the optimal policy. Note that by assuming that the initial state is deterministic $S_0 = s_0$ and that we stick to policy $\pi(a|s,\theta)$, a good $\theta$ shall maximize

$$J(\theta) = v_\pi(s_0) = \mathbb{E}_\pi(G_0|S_0 = s_0) \tag{215}$$

i.e. the overall return of the MDP which is always our goal in RL setting. The policy gradient theorem below gives a useful representation of the gradient of $J(\theta)$ w.r.t. $\theta$.

**Theorem 2.** ***(Policy Gradient Theorem, Episodic Case)***

$$\nabla J(\theta) \propto \sum_{s\in S} \mu(s) \sum_{a\in\mathscr{A}} q_\pi(s,a)\nabla\pi(a|s,\theta) \tag{216}$$

*where $\mu$ is the* ***on-policy distribution*** *as a probability measure on $S$ (defined in the proof).*

*Proof.* All gradients below are taken w.r.t. $\theta$

$$\begin{aligned}
\nabla J(\theta) &= \nabla v_\pi(s_0) && (217)\\
&= \nabla \sum_{a\in\mathscr{A}} \pi(a|s_0,\theta) q_\pi(s_0,a) && (218)\\
&= \sum_{a\in\mathscr{A}} [\nabla\pi(a|s_0,\theta)q_\pi(s_0,a) + \pi(a|s_0,\theta)\nabla q_\pi(s_0,a)] && (219)
\end{aligned}$$

use the Bellman equation that $q_\pi(s,a) = \sum_{s'\in S, r\in R} p(s',r|s,a)\cdot[r+\gamma v_\pi(s')]$ proved above to see

$$\begin{aligned}
\nabla v_\pi(s_0) &= \sum_{a\in\mathscr{A}} \left(\nabla\pi(a|s_0,\theta)q_\pi(s_0,a) + \pi(a|s_0,\theta)\nabla \sum_{s'\in S,r\in R} p(s',r|s_0,a)\cdot[r+\gamma v_\pi(s')]\right) && (220)\\
&= \sum_{a\in\mathscr{A}} \left(\nabla\pi(a|s_0,\theta)q_\pi(s_0,a) + \pi(a|s_0,\theta)\gamma \sum_{s'\in S} p(s'|s_0,a)\cdot\nabla v_\pi(s')\right) && (221)
\end{aligned}$$

an iterative relationship in $\nabla v_\pi(s)$, so let's replace the gradient of value function on RHS with such formula iteratively to see

$$
\begin{aligned}
&\nabla v_\pi(s_0) && (222)\\
&= \sum_{a\in\mathscr{A}} \nabla\pi(a|s_0,\theta) q_\pi(s_0,a) + \sum_{a\in\mathscr{A}} \pi(a|s_0,\theta)\gamma \sum_{s'\in S} p(s'|s_0,a)\cdot && (223)\\
&\sum_{a'\in\mathscr{A}} \left( \nabla\pi(a'|s',\theta) q_\pi(s',a') + \pi(a'|s',\theta)\gamma \sum_{s''\in S} p(s''|s',a')\cdot \nabla v_\pi(s'') \right) && (224)
\end{aligned}
$$

for simplicity, denote $f(s) = \sum_{a\in\mathscr{A}} \nabla\pi(a|s,\theta)\cdot q_\pi(s,a)$ so

$$
\begin{aligned}
&\nabla v_\pi(s_0) && (225)\\
&= f(s_0) + \sum_{a\in\mathscr{A}} \pi(a|s_0,\theta)\gamma \sum_{s'\in S} p(s'|s_0,a)\cdot \sum_{a'\in\mathscr{A}} \nabla\pi(a'|s',\theta) q_\pi(s',a') && (226)\\
&+ \sum_{a\in\mathscr{A}} \pi(a|s_0,\theta)\gamma \sum_{s'\in S} p(s'|s_0,a) \sum_{a'\in\mathscr{A}} \pi(a'|s',\theta)\gamma \sum_{s''\in S} p(s''|s',a')\cdot \nabla v_\pi(s'') && (227)\\
&= f(s_0) + \gamma \sum_{a\in\mathscr{A}, s'\in S} f(s')\pi(a|s_0,\theta)p(s'|s_0,a) && (228)\\
&+ \gamma^2 \sum_{s''\in S}\sum_{s'\in S}\sum_{a\in\mathscr{A}} \pi(a|s_0,\theta)p(s'|s_0,a) \sum_{a'\in\mathscr{A}} \pi(a'|s',\theta)p(s''|s',a')\cdot \nabla v_\pi(s'') && (229)\\
&= f(s_0) + \gamma \sum_{s'\in S} f(s')\mathbb{P}_\pi\left(S_1 = s'|S_0 = s_0\right) && (230)\\
&+ \gamma^2 \sum_{s'',s'\in S} \mathbb{P}_\pi\left(S_1 = s'|S_0 = s_0\right)\mathbb{P}_\pi\left(S_2 = s''|S_1 = s', S_0 = s_0\right)\nabla v_\pi(s'') && (231)\\
&= f(s_0) + \gamma \sum_{s'\in S} f(s')\mathbb{P}_\pi\left(S_1 = s'|S_0 = s_0\right) + \gamma^2 \sum_{s''\in S} \mathbb{P}_\pi\left(S_2 = s''|S_0 = s_0\right)\nabla v_\pi(s'') && (232)
\end{aligned}
$$

here we use the Markov property that $\mathbb{P}_\pi\left(S_2 = s''|S_1 = s', S_0 = s_0\right) = \mathbb{P}_\pi\left(S_2 = s''|S_1 = s'\right)$ and that such MDP is time-homogeneous so the transition kernel does not depend on time $t$. If we keep doing such iteration, we will see that

$$
\begin{aligned}
\nabla v_\pi(s_0) &= f(s_0) + \gamma \sum_{s'\in S} f(s')\mathbb{P}_\pi\left(S_1 = s'|S_0 = s_0\right) + \gamma^2 \sum_{s''\in S} \mathbb{P}_\pi\left(S_2 = s''|S_0 = s_0\right)\nabla v_\pi(s'') && (233)\\
&= ... && (234)\\
&= \sum_{s\in S} f(x) \sum_{k=0}^{\infty} \gamma^k \mathbb{P}_\pi\left(S_k = s|S_0 = s_0\right) && (235)\\
&= \sum_{s\in S}\sum_{k=0}^{\infty} \gamma^k \mathbb{P}_\pi\left(S_k = s|S_0 = s_0\right) \sum_{a\in\mathscr{A}} \nabla\pi(a|s,\theta)\cdot q_\pi(s,a) && (236)\\
&= \sum_{s\in S} \eta(s) \sum_{a\in\mathscr{A}} \nabla\pi(a|s,\theta)\cdot q_\pi(s,a) && (237)
\end{aligned}
$$

define $\eta(s) \overset{def}{=} \sum_{k=0}^{\infty} \gamma^k \mathbb{P}_\pi\left(S_k = s | S_0 = s_0\right)$, and $\mu(s) \overset{def}{=} \frac{\eta(s)}{\sum_{s' \in S} \eta(s')}$ to be the on-policy distribution, a probability measure on $S$. We have the representation that

$$\nabla J(\theta) = \left(\sum_{s' \in S} \eta(s')\right) \cdot \sum_{s \in S} \mu(s) \sum_{a \in \mathscr{A}} \nabla \pi(a|s,\theta) \cdot q_\pi(s,a) \tag{238}$$

$$\propto \sum_{s \in S} \mu(s) \sum_{a \in \mathscr{A}} \nabla \pi(a|s,\theta) \cdot q_\pi(s,a) \tag{239}$$

□

**Remark.** *To illustrate the definition of $\eta(s), \mu(s)$, let's first assume that $\gamma = 1$ which is always taken in the setting of episodic games. Then*

$$\eta(s) = \sum_{k=0}^{\infty} \mathbb{P}_\pi\left(S_k = s | S_0 = s_0\right) \tag{240}$$

*is **the expected amount of time MDP has spent in state** $s$ if the deterministic initial state is $s_0$ and policy $\pi$ is taken. As a result, $\mu(s)$ is the normalization of $\eta(s)$ and it stands for **the proportion of time MDP has spent in state** $s$ if the deterministic initial state is $s_0$ and policy $\pi$ is taken. That's why it's called on-policy distribution since $\mu(s)$ actually means **how much we care for state** $s$. One might be wondering if the normalization is well-defined, i.e. whether it's true that $\sum_{s \in S} \eta(s) < \infty$, let's do some calculations by noticing that in episodic games there exists an ending criteria so the terminal time $T$ is formed as a random stopping time and the sum in the definition of $\eta(s)$ is actually from $0$ to $T-1$*

$$\sum_{s \in S} \eta(s) = \sum_{s \in S} \sum_{k=0}^{T-1} \mathbb{P}_\pi\left(S_k = s | S_0 = s_0\right) \tag{241}$$

$$= \sum_{k=0}^{T-1} \sum_{s \in S} \mathbb{P}_\pi\left(S_k = s | S_0 = s_0\right) \tag{242}$$

$$= T \tag{243}$$

*so **the proportional constant if actually** $T$ **in the episodic case** and everything works well if $T < \infty$ a.s.. At this point, we can understand that $\eta(s) \overset{def}{=} \sum_{k=0}^{\infty} \gamma^k \mathbb{P}_\pi\left(S_k = s | S_0 = s_0\right)$ is just the discounted version of $\eta(s)$ mentioned above.*

*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*

$$\overline{VE}(w) \overset{def}{=} \sum_{s \in S} \mu(s) \left[v_\pi(s) - \hat{v}(s,w)\right]^2 \tag{244}$$

*and one typically wants to choose weight $w$ to minimize this error.*

*The formulations above provide intuition for the on-policy distribution. Generally, we are not restricted to deterministic initial state any longer, assume $h(s)$ is a probability measure on $S$ such that $h(s) = \mathbb{P}\left(S_0 = s\right)$, then $\eta(s)$ is defined as*

$$\begin{aligned}
\eta(s) &\overset{def}{=} \mathbb{E}_{S_0\sim h}\sum_{k=0}^{\infty}\gamma^k\mathbb{P}_\pi\left(S_k = s|S_0\right) && (245)\\
&= \sum_{s'\in S} h(s')\sum_{k=0}^{\infty}\gamma^k\mathbb{P}_\pi\left(S_k = s|S_0 = s'\right) && (246)\\
&= \mathbb{E}_{S_0\sim h}\mathbb{E}_\pi\left(\sum_{k=0}^{\infty}\gamma^k\mathbb{I}_{S_k=s}\middle|S_0\right) && (247)\\
&= \mathbb{E}_{S_0\sim h,\pi}\sum_{k=0}^{\infty}\gamma^k\mathbb{I}_{S_k=s} && (248)\\
&= \sum_{k=0}^{\infty}\gamma^k\mathbb{P}_{S_0\sim h,\pi}\left(S_k = s\right) && (249)
\end{aligned}$$

*has **the following property** that*

$$\begin{aligned}
\forall s\in S, \eta(s) &= \mathbb{P}_{S_0\sim h,\pi}\left(S_0 = s\right) + \gamma\sum_{k=1}^{\infty}\gamma^{k-1}\mathbb{P}_{S_0\sim h,\pi}\left(S_k = s\right) && (250)\\
&= h(s) + \gamma\sum_{l=0}^{\infty}\gamma^l\mathbb{P}_{S_0\sim h,\pi}\left(S_{l+1} = s\right) && (251)\\
&= h(s) + \gamma\sum_{l=0}^{\infty}\gamma^l\sum_{s'\in S}\mathbb{P}_{S_0\sim h,\pi}\left(S_l = s'\right)\mathbb{P}_{S_0\sim h,\pi}\left(S_{l+1} = s|S_l = s'\right) && (252)\\
&= h(s) + \gamma\sum_{l=0}^{\infty}\gamma^l\sum_{s'\in S}\mathbb{P}_{S_0\sim h,\pi}\left(S_l = s'\right)\mathbb{P}_{S_0=s',\pi}\left(S_1 = s\right) && (253)\\
&= h(s) + \gamma\sum_{s'\in S}\eta(s')\mathbb{P}_{S_0=s',\pi}\left(S_1 = s\right) && (254)\\
&= h(s) + \gamma\sum_{s'\in S}\eta(s')\sum_{a\in\mathscr{A}}\pi(a|s')\mathbb{P}_{S_0=s',\pi}\left(S_1 = s|A_0 = a\right) && (255)\\
&= h(s) + \gamma\sum_{s'\in S}\eta(s')\sum_{a\in\mathscr{A}}\pi(a|s')p(s|s',a) && (256)
\end{aligned}$$

*the interpretation of this equation is that if $S_0 = s$ then it contributes 1 to $\eta(s)$ immediately, otherwise the contribution to $\eta(s)$ comes from the transition from other states back to $s$.*

## REINFORCE

Now we apply the policy gradient theorem to provide a family of policy gradient algorithms called REINFORCE. Since we want to pick the $\theta$ that maximizes $J(\theta)$, a natural idea comes from gradient ascent that $\theta$ should be updated in the way that

$$\theta \leftarrow \theta + \alpha \nabla J(\theta^t) \tag{257}$$

where $\alpha > 0$ is a positive parameter for the step size in gradient ascent. By policy gradient theorem,

$$\nabla J(\theta) \propto \sum_{s\in S} \mu(s) \sum_{a\in\mathscr{A}} \nabla \pi(a|s,\theta) \cdot q_\pi(s,a) \tag{258}$$

so very naturally the **gradient ascent** gives us the update

$$\theta \leftarrow \theta + \alpha \sum_{s\in S} \mu(s) \sum_{a\in\mathscr{A}} \nabla \pi(a|s,\theta) \cdot q_\pi(s,a) \tag{259}$$

**Remark.** *Notice that the proportional constant in the policy gradient theorem can be absorbed into the step size parameter $\alpha$ so we don't have to worry about it!*

However, we still have two problems here with this update scheme. The first problem is that we don't know $\mu(s)$ in practice since it's the on-policy distribution as the proportional of time MDP has spent staying in each state. In order to design a model-free algorithm (which is the basic requirement for the algorithm to be useful), we never want to compute $\mu(s)$ analytically. The second problem here is that we don't know $q_\pi(s,a)$, the state-action value under policy $\pi$.

The first problem is not hard to solve by noticing that $\mu$ is a probability measure so we can find a random variable that has distribution $\mu$. Let's first consider the case where $\gamma = 1$. By Kolmogorov's cycle trick,

$$\mu(s) = \frac{\sum_{k=0}^{\infty} \mathbb{P}_\pi\left(S_k = s|S_0 = s_0\right)}{\sum_{s\in S}\sum_{k=0}^{\infty} \mathbb{P}_\pi\left(S_k = s|S_0 = s_0\right)} \tag{260}$$

is the stationary measure (and stationary distribution) of Markov chain $\{S_t\}$ given terminal time $T < \infty$ and $s_0$ as a recurrent state. That is to say, by assuming that we start from the initial state following the stationary distribution $S_0 \sim \mu$, it's always true that $\forall t, S_t \sim \mu$.

**Remark.** *The policy gradient algorithm adopts the **assumption that the initial state $S_0$ already follows the stationary distribution** $\mu$. Although this seems not reasonable in practice since we are often starting from some "arbitrary" initial state, the asymptotic behavior of Markov chain ensures the convergence of the distribution $S_t \xrightarrow{d} \mu$ $(t \to \infty)$ if $\{S_t\}$ is nice enough (it's an ergodic Markov chain).*

As argued above, for the case where $\gamma = 1$ and the assumption that $S_0 \sim \mu$, we have

$$\sum_{s\in S}\mu(s)\sum_{a\in\mathscr{A}}\nabla\pi(a|s,\theta)\cdot q_\pi(s,a) = \mathbb{E}_\pi\sum_{a\in\mathscr{A}}\nabla\pi(a|S_t,\theta)\cdot q_\pi(S_t,a) \tag{261}$$

the structure as expectation of random variables! Now let's deal with the state-action value $q_\pi(S_t, a)$. Notice that

$$q_\pi(S_t,a) = \mathbb{E}_\pi(G_t|S_t, A_t = a) \tag{262}$$

$$q_\pi(S_t,A_t) = \mathbb{E}_\pi(G_t|S_t, A_t) \tag{263}$$

so it's a good idea to exploit again the structure of expectation contained in this state-action value. In order to make the sum of all actions $a \in \mathscr{A}$ an expectation w.r.t. $A_t$, we need a probability measure on the action space and naturally we notice that the policy itself is a probability measure on the action space. This leads to the transformation that

$$\begin{aligned}
\mathbb{E}_\pi\sum_{a\in\mathscr{A}}\nabla\pi(a|S_t,\theta)\cdot q_\pi(S_t,a) &= \mathbb{E}_\pi\sum_{a\in\mathscr{A}}\pi(a|S_t,\theta)\frac{\nabla\pi(a|S_t,\theta)}{\pi(a|S_t,\theta)}\cdot q_\pi(S_t,a) && (264)\\
&= \mathbb{E}_\pi\mathbb{E}_{A_t\sim\pi(\cdot|S_t,\theta)}\left(\frac{\nabla\pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}\cdot q_\pi(S_t,A_t)\middle|S_t\right) && (265)\\
&= \mathbb{E}_\pi\left(\frac{\nabla\pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}\cdot q_\pi(S_t,A_t)\right) && (266)\\
&= \mathbb{E}_\pi\left[\nabla\log\pi(A_t|S_t,\theta)\cdot\mathbb{E}_\pi(G_t|S_t,A_t)\right] && (267)\\
&= \mathbb{E}_\pi\left[G_t\cdot\nabla\log\pi(A_t|S_t,\theta)\right] && (268)
\end{aligned}$$

After solving those two problems, we see that the gradient ascent update of $\theta$ is given by

$$\theta \leftarrow \theta + \alpha\cdot\mathbb{E}_\pi\left[G_t\cdot\nabla\log\pi(A_t|S_t,\theta)\right] \tag{269}$$

although we do not know the value of the exact gradient (the expectation) here, it's easy to randomly 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 **policy gradient update** that

$$\theta \leftarrow \theta + \alpha\cdot G_t\cdot\nabla\log\pi(A_t|S_t,\theta) \tag{270}$$

**Remark.** *The importance of policy gradient algorithm is that in each step only the future discounted rewards $G_t$ and the gradient of the log policy at current state-action pair is used. As a result, we don't have to worry about **large state space or action space** any longer! Besides, policy gradient algorithm provide the possibility of **converging to a stochastic policy as the optimal policy**. In value based methods, the optimal policy is always formed as the deterministic policy putting all probability mass on the action that maximizes state-action value for the observed state.*

**Remark.** *The policy gradient update presented above only holds for $\gamma = 1$. However, if one hopes to generalize the update formula for $\gamma \in (0,1)$ we can adopt the following idea. The problem we are facing at stage $k$ is always equivalent to the same problem we are facing at stage $k+1$ with all rewards discounted by multiplying $\gamma$. As a result, we would perform the same policy gradient update at stage $k+1$ as what we have done at stage $k$ and the only difference is that there's one more discount factor $\gamma$ in the $G_t$. So the* ***discounted policy gradient update*** *is given by*

$$\theta \leftarrow \theta + \alpha \cdot \gamma^t G_t \cdot \nabla \log \pi(A_t|S_t, \theta) \tag{271}$$

***performed at time*** $t$ ***within each single episode***.

At this point, let's state the most fundamental policy gradient algorithm **REINFORCE**. The algorithm starts with an initial parameter value $\theta$ and a given form of the policy $\pi(a|s,\theta)$. For each episode, the algorithms trains the parameter $\theta$ in the following way

- Generate an episode $S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T$ following the current policy under parameter $\theta$.
- At each time $t$ from 0 to $T-1$ within this single episode, build up $G_t$ incrementally and perform the update

$$\theta \leftarrow \theta + \alpha \cdot \gamma^t G_t \cdot \nabla \log \pi(A_t|S_t, \theta) \tag{272}$$

**Remark.** *We briefly talk about how to set up the policy with a specific form $\pi(a|s,\theta)$ here. Alike the situation in the bandit gradient algorithm, the simple way is to set up the preference $h(s,a,\theta)$ for each state-action pair $(s,a)$ and form the policy as the softmax image of the preference, i.e.*

$$\pi(a|s,\theta) = \frac{e^{h(s,a,\theta)}}{\sum_{a' \in \mathscr{A}} e^{h(s,a',\theta)}} \tag{273}$$

*and the preference can be chosen as a function linear in $\theta$, i.e.*

$$h(s,a,\theta) = \theta^T x(s,a) \tag{274}$$

*under such setting, the* ***eligibility vector*** *is*

$$\nabla \log \pi(a|s,\theta) = x(s,a) - \sum_{a' \in \mathscr{A}} \pi(a'|s,\theta) \cdot x(s,a') \tag{275}$$

*In more general cases or practical usage, such policy $\pi(a|s,\theta)$ is always approximated by a neural network with $\theta$ to be the collection of all parameters of the NN. The output is ensured to be a policy by setting the output layer of the NN as a softmax layer.*

## REINFORCE with Baseline

REINFORCE makes use of the stochastic gradient ascent and the sample of the gradient has expectation as the true gradient by the policy gradient theorem proved above. However, although such sampling is not biased, there might be a huge variance that significantly slows down the convergence of the algorithm. As shown in the example of bandit gradient algorithm, adding a baseline would be a good variance reduction technique. The reason why we can add a baseline without interfering with the policy gradient update can be seen from the policy gradient theorem that

$$\sum_{s\in S}\mu(s)\sum_{a\in\mathscr{A}}\nabla\pi(a|s,\theta)\cdot q_\pi(s,a)=\sum_{s\in S}\mu(s)\sum_{a\in\mathscr{A}}\nabla\pi(a|s,\theta)\cdot[q_\pi(s,a)-b(s)] \tag{276}$$

this is because

$$\sum_{s\in S}\mu(s)\sum_{a\in\mathscr{A}}\nabla\pi(a|s,\theta)\cdot b(s)=\sum_{s\in S}\mu(s)\cdot b(s)\cdot\nabla 1=0 \tag{277}$$

embedding this baseline into policy gradient update provides the **policy gradient update with baseline** $b$

$$\theta\leftarrow\theta+\alpha\cdot\gamma^t[G_t-b(S_t)]\cdot\nabla\log\pi(A_t|S_t,\theta) \tag{278}$$

note that **such baseline can only depend on the current state but not current action** such that all previously derived conclusions still work.

In bandit gradient algorithm, we choose the baseline to be the sample mean of all past rewards. Here we can do the similar thing and set

$$b(S_t)=\mathbb{E}_{\pi^*}(G_t|S_t)=v_*(S_t) \tag{279}$$

the state value of $S_t$. Naturally, we maintain $\hat{v}(S_t,w)$ as an estimate of the state value with weight parameter $w$ and set it as the baseline. In the algorithm, we shall update both parameters $\theta$ and $w$ with the information provided simultaneously. For the update of $w$, we minimize the weighted mean square error as previously described

$$\min_w \overline{\text{VE}}(w)=\sum_{s\in S}\mu(s)\left[v_\pi(s)-\hat{v}(s,w)\right]^2 \tag{280}$$

with gradient descent. Compute the gradient w.r.t. $w$ that

$$\nabla_w\overline{\text{VE}}(w)=\sum_{s\in S}\mu(s)\nabla\left[v_\pi(s)-\hat{v}(s,w)\right]^2 \tag{281}$$

$$=2\sum_{s\in S}\mu(s)\left[\hat{v}(s,w)-v_\pi(s)\right]\cdot\nabla_w\hat{v}(s,w) \tag{282}$$

$$\propto\mathbb{E}_\pi\left[\hat{v}(S_t,w)-v_\pi(S_t)\right]\cdot\nabla_w\hat{v}(S_t,w) \tag{283}$$

here we adopt the same assumption in policy gradient update that we start from initial state following the stationary distribution $S_0 \sim \mu$ to represent the sum as an expectation w.r.t. $S_t$, then

$$\nabla_w \overline{\text{VE}}(w) \propto \mathbb{E}_\pi \left[\hat{v}(S_t, w) - \mathbb{E}_\pi(G_t|S_t)\right] \cdot \nabla_w \hat{v}(S_t, w) \tag{284}$$

$$= \mathbb{E}_\pi \left[\hat{v}(S_t, w) - G_t\right] \cdot \nabla_w \hat{v}(S_t, w) \tag{285}$$

the gradient has the structure of the expectation so the stochastic gradient descent update of $w$ for state value approximation is

$$w \leftarrow w + \alpha \left[G_t - \hat{v}(S_t, w)\right] \cdot \nabla_w \hat{v}(S_t, w) \tag{286}$$

where $\alpha > 0$ is the step size parameter.

Now let's give the **REINFORCE with baseline** algorithm. It starts with an initial parameter value $\theta, w$ and a given form of the policy $\pi(a|s, \theta)$ and the state value approximation $\hat{v}(s, w)$. For each episode, the algorithms trains the parameter $\theta$ and $w$ in the following way

- Generate an episode $S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T$ following the current policy under parameter $\theta$.
- At each time $t$ from 0 to $T-1$ within this single episode, build up $G_t$ incrementally and perform the updates ($\alpha^\theta$ is the step size parameter in the update of $\theta$ and $\alpha^w$ is the step size parameter in the update of $w$)

$$w \leftarrow w + \alpha^w \cdot \left[G_t - \hat{v}(S_t, w)\right] \cdot \nabla_w \hat{v}(S_t, w) \tag{287}$$

$$\theta \leftarrow \theta + \alpha^\theta \cdot \gamma^t [G_t - \hat{v}(S_t, w)] \cdot \nabla_\theta \log \pi(A_t|S_t, \theta) \tag{288}$$

**Remark.** *There's many possible choices for the form of $\hat{v}(s, w)$. One can set it to be linear in $w$ or to set up a neural network with parameter $w$. Practically setting $\alpha^w = \frac{0.1}{\mathbb{E}_{S_t \sim \mu} ||\nabla \hat{v}(S_t, w)||^2}$ would be the best choice but setting a good $\alpha^\theta$ should depend on different problems.*

## Actor-Critic Methods

In the REINFORCE with baseline, we maintain an estimate of state-value function and update its weight $w$ with stochastic gradient descent to form a baseline. However, if a state transition from state $s_1$ to state $s_2$ happens, such state-value estimate only takes into account the state value of $s_1$ but has not yet taken into account the state value of $s_2$. By including the value estimate of both states in a state transition, we would be able to build up the **temporal difference (TD) error** as a criterion of how well we are doing in estimating the state value. Actor-critic methods allow ideas in TD methods to combine with the policy gradient ideas, which provides us with some more useful results.

The **actor** in our setting refers to the policy since it keeps generating actions while the **critic** refers to the estimated state value function since it assesses the actions taken. The main idea here is to replace the $G_t$ in REINFORCE update with $G_{t:t+1}$, the one-step return at time $t$ formed as

$$G_{t:t+1} = R_{t+1} + \gamma G_{t+1} \tag{289}$$

since $G_t = R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + ...) = R_{t+1} + \gamma G_{t+1}$. Now the unknown $G_{t+1}$ can be approximated by

$$\hat{v}(S_{t+1}, w) \tag{290}$$

so the term $G_t - \hat{v}(S_t, w)$ in REINFORCE with baseline update should be replaced with

$$R_{t+1} + \gamma\hat{v}(S_{t+1}, w) - \hat{v}(S_t, w) \tag{291}$$

which is just the TD error at time $t$.

**Remark.** *The intuitive explanation of such TD error is that $R_{t+1} + \gamma\hat{v}(S_{t+1}, w)$ is the latest and better estimate of the state value of $S_t$ while $\hat{v}(S_t, w)$ is the former estimate of the state value of $S_t$. As a result, the TD error at each time measures the error in the estimate made at that time.*

One main advantage of replacing the overall return with one-step return is that the algorithm now can be formed as totally online and incremental. In the previous REINFORCE training, one always have to wait for an episode to terminate since the overall return $G_t$ is required in the update. For one-step actor-critic methods, the update only requires $R_{t+1}, S_t, S_{t+1}$ that can all be directly observed at time $t$. As a result, now we are able to train the RL model when the game is still on play.

**Remark.** *The idea of using one-step return comes from TD learning algorithms. Although one-step return has bias, typically it enjoys much less variance and enables the algorithm to update incrementally so generally it's a better object to use compared to the overall return.*

*It's immediate that one may generalize such one-step method to $n$-step and $\lambda$-return algorithms. However, the main idea is always the same so here we just state the simplest one-step method.*

Now let's give the **one-step actor-critic** algorithm. It starts with an initial parameter value $\theta, w$ and a given form of the policy $\pi(a|s,\theta)$ and the state value approximation $\hat{v}(s,w)$. The algorithm is **totally online** so the training can be done simultaneously as the game proceeds.

- Start the episode with an initial state, set up $I \leftarrow 1$ (to record $\gamma^t$, the discount factor).
- When the current state $S$ is not the terminal state, sample an action from the policy $A \sim \pi(\cdot|S,\theta)$. Take such action $A$ so one gets reward $R$ and the state transits to $S'$.
- Set up the TD error

$$\delta \leftarrow R + \gamma\hat{v}(S',w) - \hat{v}(S,w) \tag{292}$$

  where $\hat{v}(S',w)$ is set as 0 if $S'$ is the terminal state.

- Do the following update that ($\alpha^\theta$ is the step size parameter in the update of $\theta$ and $\alpha^w$ is the step size parameter in the update of $w$)

$$w \leftarrow w + \alpha^w \cdot \delta \cdot \nabla_w \hat{v}(S_t,w) \tag{293}$$

$$\theta \leftarrow \theta + \alpha^\theta \cdot I\delta \cdot \nabla_\theta \log \pi(A_t|S_t,\theta) \tag{294}$$

- Move to the next state and do such update iteratively

$$I \leftarrow \gamma I \tag{295}$$

$$S \leftarrow S' \tag{296}$$

## Extension for Continuing Tasks

One might notice that the algorithms and theorems above only work for episodic games. However, **continuing games** are also prevalent and crucial to solve. Those games do not have a terminal criterion it's possible to last forever. For policy gradient methods, the largest difference from that in episodic case is the formation of the performance $J(\theta)$ defined as

$$J(\theta) \overset{def}{=} \lim_{h\to\infty} \frac{1}{h}\sum_{t=1}^{h} \mathbb{E}_\pi(R_t|S_0) \tag{297}$$

**Remark.** *Such performance is called* ***the average rate of reward per time step****. Intuitively, if one starts from initial state $S_0$ and follows policy $\pi$, one would get on average $\mathbb{E}_\pi(R_t|S_0)$ amount of reward at time $t-1$. $J(\theta)$ is just the asymptotic average of the expected amount of reward one can get at time $0, 1, ..., h-1$ (h is large enough). Unlike the episodic case where one always have an almost surely finite terminal time, in the continuing case the game might never stop so it's not reasonable to define $J(\theta)$ as $\mathbb{E}_\pi(G_0|S_0)$ any longer (there's typically no discount rate in continuing game setting).*

*Notice that the expectation can be calculated in the following way that*

$$\mathbb{E}_\pi(R_t|S_0) = \sum_{s\in S, a\in\mathscr{A}} \mathbb{P}_\pi\left(S_t = s, A_t = a\right) \cdot \mathbb{E}_\pi(R_t|S_0, S_t = s, A_t = a) \tag{298}$$

$$= \sum_{s\in S, a\in\mathscr{A}} \mathbb{P}_\pi\left(S_t = s\right) \cdot \pi(a|s) \cdot \sum_{s'\in S, r\in R} p(s', r|s, a) \cdot r \tag{299}$$

*now assume $S_t \sim \mu$ follows the on-policy distribution under policy $\pi$, we see that*

$$\mathbb{E}_\pi(R_t|S_0) \sim \sum_{s\in S} \mu(s) \sum_{a\in\mathscr{A}} \pi(a|s) \sum_{s'\in S, r\in R} p(s', r|s, a) \cdot r \ (t\to\infty) \tag{300}$$

*for large enough $t$ under the assumption that the Markov chain $\{S_t\}$ is ergodic since $S_t \xrightarrow{d} \mu$ $(t\to\infty)$ for stationary distribution $\mu$. Recall that we have proved in the previous context that when there's no discount factor the on-policy distribution $\mu$ is just the stationary distribution.*

*As a result, we are actually saying that*

$$\lim_{t\to\infty} \mathbb{E}_\pi(R_t|S_0) = \sum_{s\in S} \mu(s) \sum_{a\in\mathscr{A}} \pi(a|s) \sum_{s'\in S, r\in R} p(s', r|s, a) \cdot r \tag{301}$$

*and notice that the existence of such limit implies*

$$J(\theta) = \lim_{h\to\infty} \frac{1}{h}\sum_{t=1}^{h} \mathbb{E}_\pi(R_t|S_0) = \lim_{t\to\infty} \mathbb{E}_\pi(R_t|S_0) \tag{302}$$

$$= \sum_{s\in S} \mu(s) \sum_{a\in\mathscr{A}} \pi(a|s) \sum_{s'\in S, r\in R} p(s', r|s, a) \cdot r \tag{303}$$

*different characterizations of the performance $J(\theta)$.*

Correspondingly, the value functions in the continuing game setting is different from those in the episodic game. Since $J(\theta)$ stands for the average rate of reward per time step, the return $G_t$ is now formed as **the differential return, i.e. the sum of excessive future return compared to** $J(\theta)$

$$G_t \overset{def}{=} [R_{t+1} - J(\theta)] + [R_{t+2} - J(\theta)] + ... \tag{304}$$

and the value functions are defined similarly as $v_\pi(s) = \mathbb{E}(G_t|S_t = s), q_\pi(s,a) = \mathbb{E}(G_t|S_t = s, A_t = a)$. Now let's state that the policy gradient theorem still holds for such formulation so the previously stated algorithms still work for continuing tasks up to a slight modification.

**Theorem 3.** ***(Policy Gradient Theorem, Continuing Case)***

$$\nabla J(\theta) = \sum_{s\in S} \mu(s) \sum_{a\in\mathscr{A}} q_\pi(s,a)\nabla\pi(a|s,\theta) \tag{305}$$

*where $\mu$ is the stationary distribution of Markov chain $\{S_t\}$.*

*Proof.* The Bellman equation in the continuing case becomes

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi(G_t|S_t = s, A_t = a) &(306)\\ &= \sum_{s'\in S} \mathbb{P}\left(S_{t+1} = s'|S_t = s, A_t = a\right) \mathbb{E}_\pi(G_t|S_{t+1} = s', S_t = s, A_t = a) &(307)\\ &= \sum_{s'\in S} p(s'|s,a)\mathbb{E}_\pi(G_{t+1} + R_{t+1} - J(\theta)|S_{t+1} = s', S_t = s, A_t = a) &(308)\\ &= \sum_{s'\in S, r\in R} p(s',r|s,a)[v_\pi(s') + r - J(\theta)] &(309) \end{aligned}$$

now get the gradient of state value function w.r.t. $\theta$ (same calculations as before)

$$\nabla v_\pi(s) = \sum_{a\in\mathscr{A}} \left[\nabla\pi(a|s,\theta)q_\pi(s,a) + \pi(a|s,\theta)\left(-\nabla J(\theta) + \sum_{s'\in S} p(s'|s,a)\nabla v_\pi(s')\right)\right] \tag{310}$$

and represent $\nabla J(\theta)$ with $\nabla v_\pi(s)$ to see

$$\nabla J(\theta) = \sum_{a\in\mathscr{A}} \left[\nabla\pi(a|s,\theta)q_\pi(s,a) + \pi(a|s,\theta)\sum_{s'\in S} p(s'|s,a)\nabla v_\pi(s')\right] - \nabla v_\pi(s) \tag{311}$$

Since $J(\theta)$ does not depend on the state $s$, the RHS also does not depend on the state $s$ so we can take weighted average of both sides w.r.t. the stationary distribution $\mu$ to get

$$\nabla J(\theta) = \sum_{s\in S}\mu(s)\sum_{a\in\mathscr{A}}\left[\nabla\pi(a|s,\theta)q_\pi(s,a) + \pi(a|s,\theta)\sum_{s'\in S}p(s'|s,a)\nabla v_\pi(s')\right] - \sum_{s\in S}\mu(s)\nabla v_\pi(s) \tag{312}$$

$$= \sum_{s\in S}\mu(s)\sum_{a\in\mathscr{A}}\nabla\pi(a|s,\theta)q_\pi(s,a) + \sum_{s\in S}\mu(s)\sum_{a\in\mathscr{A}}\pi(a|s,\theta)\sum_{s'\in S}p(s'|s,a)\nabla v_\pi(s') - \sum_{s\in S}\mu(s)\nabla v_\pi(s) \tag{313}$$

$$= \sum_{s\in S}\mu(s)\sum_{a\in\mathscr{A}}\nabla\pi(a|s,\theta)q_\pi(s,a) \tag{314}$$

since $\sum_{s\in S}\mu(s)\sum_{a\in\mathscr{A}}\pi(a|s,\theta)\sum_{s'\in S}p(s'|s,a)\nabla v_\pi(s') = \sum_{s\in S}\mu(s)\nabla v_\pi(s)$. This is due to the fact that $\mu$ is the stationary distribution so

$$\forall s'\in S, \mu(s') = \sum_{s\in S}\mu(s)\sum_{a\in\mathscr{A}}\pi(a|s,\theta)p(s'|s,a) \tag{315}$$

□

Let's end the part for continuing task extension with the slight modification of the one-step actor-critic algorithm. Now the one-step return $G_{t:t+1}$ is formed such that

$$G_t = G_{t+1} + R_{t+1} - J(\theta) \tag{316}$$

so the estimation of the overall return by one-step return shall have one more term $\overline{R}$ included, correspondent to $J(\theta)$. As a result, the TD error shall be formed as

$$\delta \leftarrow \hat{v}(S',w) + R - \overline{R} - \hat{v}(S,w) \tag{317}$$

and one shall update $\overline{R}$ as the estimate for $J(\theta)$, for example

$$\overline{R} \leftarrow \overline{R} + \alpha^{\overline{R}}\delta \tag{318}$$

for some $\alpha^{\overline{R}} > 0$ as the step size parameter.

## Extensions for Continuous Action Parameterization

The policy gradient algorithms appear naturally to deal with problems with a large state space where action value and state value approximation becomes infeasible. However, there might also be the case where the action space is too large to be viewed as a set of finitely many elements. In such case, treating $\pi(a|s,\theta)$ as the output of softmax function acted on the preference/feature does not work well since each action is only assigned an infinitesimal small amount of probability mass.

The parameterization of actions is then necessary and the policy is always formed as a **parametric family of distribution**. Let's take the Gaussian distribution $N(\mu,\sigma^2)$ as an example. The action $A$ shall be sampled from the distribution

$$A \sim N(\mu(s,\theta),\sigma^2(s,\theta)) \tag{319}$$

where $\mu(s,\theta),\sigma(s,\theta)$ are the mean and standard deviation of the sampling Gaussian and they depend on the current state $s$ and the parameter $\theta$. In other words,

$$\pi(a|s,\theta) = \frac{1}{\sqrt{2\pi}\sigma(s,\theta)} e^{-\frac{[a-\mu(s,\theta)]^2}{2\sigma^2(s,\theta)}} \tag{320}$$

Now the parameter $\theta$ consists of two parts: $\theta_\mu$ as the parameters in $\mu(s,\theta)$ and $\theta_\sigma$ as the parameters in $\sigma(s,\theta)$. The mean can be formed as a linear function in $\theta_\mu$ and the standard deviation can be formed as the exponential of a linear function in $\theta_\sigma$ to ensure its positivity

$$\mu(s,\theta) = \theta_\mu^T x_\mu(s), \sigma(s,\theta) = e^{\theta_\sigma^T x_\sigma(s)} \tag{321}$$

where $x_\mu, x_\sigma$ are features extracted based on the states.

In **Gaussian policy parameterization**, by calculations under the settings above, the eligibility vector w.r.t. $\theta_\mu$ has the following form that

$$\begin{aligned}
\nabla_{\theta_\mu} \log \pi(a|s,\theta) &= \frac{\nabla_{\theta_\mu}\pi(a|s,\theta)}{\pi(a|s,\theta)} && (322)\\
&= \frac{\frac{\partial}{\partial \mu(s,\theta)}\pi(a|s,\theta) \cdot \frac{\partial}{\partial \theta_\mu}\mu(s,\theta)}{\pi(a|s,\theta)} && (323)\\
&= \frac{\pi(a|s,\theta) \cdot \frac{2(a-\mu(s,\theta))}{2\sigma^2(s,\theta)} \cdot x_\mu(s)}{\pi(a|s,\theta)} && (324)\\
&= \frac{(a-\mu(s,\theta))}{\sigma^2(s,\theta)} \cdot x_\mu(s) && (325)
\end{aligned}$$

and now consider the eligibility vector w.r.t. $\theta_\sigma$

$$\begin{aligned}
\nabla_{\theta_\sigma} \log \pi(a|s,\theta) &= \frac{\nabla_{\theta_\sigma}\pi(a|s,\theta)}{\pi(a|s,\theta)} &(326)\\
&= \frac{\frac{\partial}{\partial \sigma(s,\theta)}\pi(a|s,\theta)\cdot \frac{\partial}{\partial \theta_\sigma}\sigma(s,\theta)}{\pi(a|s,\theta)} &(327)\\
&= \frac{\left(-\frac{1}{\sigma(s,\theta)}\pi(a|s,\theta)+\pi(a|s,\theta)[a-\mu(s,\theta)]^2\sigma^{-3}(s,\theta)\right)\cdot \sigma(s,\theta)x_\sigma(s)}{\pi(a|s,\theta)} &(328)\\
&= \left(\frac{[a-\mu(s,\theta)]^2}{\sigma^2(s,\theta)}-1\right)x_\sigma(s) &(329)
\end{aligned}$$

so in the policy gradient update we will update $\theta_\mu, \theta_\sigma$ respectively with those eligibility vectors.

**Remark.** *Consider an example given in the Barto-Sutton book for **Bernoulli logistic unit** as a stochastic neuron-like unit with its input $x(S_t)$ as a feature vector of states and its output $A_t$ as an action taking values 0 or 1 such that $A_t \sim B(1,P_t)$. There are preferences toward two states $h(s,0,\theta), h(s,1,\theta)$ when one is at state $s$ with parameter $\theta$ and their difference is given by a weighted sum of the unit's input, i.e. $h(s,1,\theta)-h(s,0,\theta)=\theta^T x(s)$.*

*Now if the policy is formed as softmax policy based on those two preferences, then*

$$\begin{aligned}
P_t = \pi(1|S_t,\theta_t) &= \frac{e^{h(S_t,1,\theta)}}{e^{h(S_t,0,\theta)}+e^{h(S_t,1,\theta)}} &(330)\\
&= \frac{1}{1+e^{-\theta^T x(s)}} &(331)
\end{aligned}$$

*Next, we derive the Monte-Carlo REINFORCE update of $\theta$ upon the receipt of return $G_t$. By previously mentioned policy gradient update,*

$$\theta \leftarrow \theta + \alpha\gamma^t G_t \cdot \nabla \log \pi(A_t|S_t,\theta) \tag{332}$$

*where $\alpha > 0$ is some step size parameter in stochastic gradient ascent. Now let's plug in the value of $P_t$ derived above to see that*

$$\theta \leftarrow \theta + \alpha\gamma^t G_t \cdot (-1)^{A_t}[1-\pi(A_t|S_t,\theta)]e^{-\theta^T x(s)} \cdot x(s) \tag{333}$$

*after calculations for the eligibility vector. This gives the REINFORCE update without baseline.*