8 Introduction to Reinforcement Learning

8.1 Reinforcement Learning

Reinforcement learning (RL) can be considered as one of the three basic machine learning paradigms, alongside supervised learning (e.g., regression and classification) and unsupervised learning (e.g., clustering) discussed previously. The goal of RL is for the algorithm, a software *agent*, to learn to make a sequence of decisions, called *policy*, for a specific task in a given environment, for the purpose of receiving the maximum rewards from the environment.

Different from supervised learning for either regression or classification based on a given training dataset composed of a set of i.i.d. sample points $\mathbf{x}_n$ labeled by y_n, $(n = 1, \cdots, N)$, RL is a learning method without any examples of optimal behaviors. Instead, RL learns on its own without relying on any labeled training dataset explicitely informing the agent what the correct responses are while interacting with the environment. On the other hand, different from the unsupervised methods that learns passively from the given dataset, RL is a trial-and-error learning method, in which the agent actively interacts with and learns from the given environment.

Specifically, RL as a sequential method depends on the dynamics of the environment which is modeled by a *Markov decision process (MDP)*, a stochastic system of multiple states with probabilistic state transitions and rewards. If the MDP model of the environment in terms of its the state transition and reward probabilities is known, the agent only needs to find the optimal policy in terms of what action to take at each state of the MDP to achieve maximum accumulated rewards as the consequence of its actions. The task in this *model-based* case is called *planning*. On the other hand, if MDP of the environment is unknown, the agent needs to learn the environment by repeatedly running the dynamic process of the MDP based on some initial policy and gradually evaluate the received rewards and improve the policy to eventually reach optimality. The task in this *model-free* case is called *control*.

Depending on the action taken by the agent in the current state s, the system transits to the next state s' following certain transition probability distribution. The RL is to choose the optimal action in the current state, to maximize the long-term expected reward, as the feedback of the environment. This is typically carried out by *dynamic programming (DP)*, a class of algorithms seeking to

Cut "discussed previously." Readers already know what's been discussed previously, and if they don't, they can find out from the table of contents.

This first sentence is too long to easily follow. Consider breaking it into several shorter sentences.

Define "sequential method."

"Stochastic" has been used multiple times in the previous chapters, so it shouldn't be defined here, but it *should* be defined the first time it's used.

simplify a complex problem by breaking it up into a set of sub-problems which can be solved recursively.

This is a very long sentence that would be clearer if broken up into shorter sentences.

The figure below illustrates the basic concepts of RL, where the agent makes a decision in terms of the action a_t to take in the current state s_t by following some policy $\pi(a|s)$, while the environment makes a transition from the current state s_t to the next state s_{t+1} by following the transition probability, a conditional probability $p(s'|s,a)$ for the next state $s_{t+1} = s'$ given the curent state $s_t = s$ and the action a_t taken by the agent, and provides a reward r_{t+1}.

8.2 Markov Decision Process

"Markov decision process" is a countable noun (that is, there are multiple different Markov decision processes), which means that it either needs to be "a Markov decision process," or else it needs to be pluralized. The easiest way to rewrite the first sentence would probably be, "Markov decision processes (MDPs) are the mathematical framework for reinforcement learning."

Markov decision process (MDP) is the mathematical framework for reinforcement learning. To understand MDP, we first consider the [illegible] concept of *Markov process* or *Markov chain*, a stochastic model of a system that can be characterized by a set of states $S = \{s_1, \cdots, s_N\}$ of size $|S| = N$ (cardinality of S). The dynamics of the system in terms of its state transition is modeled by the transition probability $P_{ij} = P(s_j|s_i)$ from the current state $s_t = s_i$ to the next state $s_{t+1} = s_j$ with $i, j = 1, \cdots, N$. The system is assumed to be memoryless, i.e., its future is independent of the past history $h_t = \{s_t, s_{t-1}, \cdots, s_0\}$, given the present s_t:

$$P(s_{t+1}|h_t) = P(s_{t+1}|s_t, s_{t-1}, \cdots, s_0) \tag{8.1}$$

For example, if the state of a helicopter is described by its linear and angular position and velocity, based on which its future position and velocity can be completely determined, then it can be modeled by a Markov process; but if the state is only described by its position, then its position in the past is needed for determining its future (e.g., to find its velocity), and it is not a Markov process.

All transition probabilities can be organized as an $N \times N$ state transition matrix:

The composition of this matrix should be explained. P_{ij} is the probability of transitioning from state i to state j.

$$\mathbf{P} = \begin{bmatrix} P_{11} & \cdots & P_{1N} \\ \vdots & \ddots & \vdots \\ P_{N1} & \cdots & P_{NN} \end{bmatrix} \tag{8.2}$$

As the transition probabilities from any current state s_i to the N possible next states $s_1, \cdots, s_N$ have to sum up to 1, we have

$$\sum_{j=1}^{N} P_{ij} = \sum_{j=1}^{N} P(s_j|s_i) = 1, \qquad (i = 1, \cdots, N) \tag{8.3}$$

Any matrix satisfying this property is called a *stochastic matrix*.

In the following, our discussion is mostly concentrated on the the general transition from the current state $s_t = s$ to the next state $s_{t+1} = s'$ with transition probability $P(s_{t+1} = s'|s_t = s) = P_{ss'}$.

This example needs significantly more description. What are the states? Where do the transition probabilities come from? Without the story behind the example, this is just a matrix of abstract numbers that do not help readers to understand the concept.

Example Weather in Los Angeles:

$$\mathbf{P} = \begin{bmatrix} 0.7 & 0.2 & 0.1 \\ 0.4 & 0.3 & 0.3 \\ 0.5 & 0.2 & 0.3 \end{bmatrix} \tag{8.4}$$

We next consider a *Markov reward process (MRP)*, represented by a tuple $\langle S, P, R, \gamma \rangle$ of four elements for the states, state transition probabilities, rewards, and the *discount factor*. Here the *reward* R can be considered as a feedback signal to the agent at each time step of the sequence of state transitions, indicating how well it is doing with respect to the overall task. The reward can be negative, as a penalty, for situations to be avoided. The performance of the agent is measured by the sum of rewards accumulated over all time steps of the Markov process, weighted by the discount factor $\gamma \in [0,\ 1]$ that discounts future rewards. If γ is close to 0, then the immediate reward is emphasized (short-sighted or greedy), but if γ is close to 1, then rewards in the future steps will be almost as valuable as immediate ones (far-sighted).

This makes it sound like the reward is greedy, but actually it's the algorithm that's greedy.

The sequence of state transitions of an MRP from an initial state s_0 to a terminal state s_T is called an *episode*, and the number of time steps T is called *horizon*, which can be either finite or infinite.

Both the reward r and the next state s' resulting from arriving at the current state s are considered as random variables with joint probability $p(s', r|s) = P(s_{t+1} = s', r_{t+1} = r|s_t = s)$, based on which both the state transition probability and state reward probability can be found by marginalization:

$$p(s'|s) = P(s_{t+1} = s'|s_t = s) = \sum_{r \in R} p(s', r|s), \qquad P(r|s) = \sum_{s' \in S} p(s', r|s) \tag{8.5}$$

Note that conventionally the reward r received after arriving at state s_t is denoted by r_{t+1}, instead of r_t. The index $s' \in S$ for the summation over all states will be abbreviated to s' in the subsequent discussion. The expectation of the reward at s is

$$r(s) = E[r_{t+1}|s_t = s] = \sum_r rP(r|s) = \sum_r r \sum_{s'} P(s', r|s) \tag{8.6}$$

We define the *return* G_t at each step $s = s_t$ as the accumulate reward, the sum of the immediate reward after arriving at the current state $s = s_t$ and all delayed rewards in the future states upto a terminal state s_T with reward r_T at the end of the episode, discounted by γ:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1} \tag{8.7}$$

We further define the *state value* function $v(s)$ at state $s = s_t$ as the expectation of the return, which can be further expressed recursively in terms of the values

$v(s')$ of all possible next states $s' = s_{t+1}$:

$$\begin{aligned} v(s) &= E[\, G_t | s_t = s \,] = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | s_t = s] \\ &= E[r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \cdots) | s_t = s] \\ &= E[r_{t+1} | s = s_t] + \gamma E[G_{t+1} | s_{t+1} = s',\, s_t = s] \\ &= r(s) + \gamma \sum_{s'} P(s'|s) v(s') \end{aligned} \tag{8.8}$$

where $r(s)$ is given in Eq. (8.6). The value of a terminal state s_T at the end of an episode is zero, as there will be no next state and thereby no more future reward.

This sentence is mostly restating the sentence that came before the equation, which undercuts the importance of the new information being conveyed: that the equation allows the value to be calculated without explicitly using future rewards.

This equation is called the *Bellman equation*, by which the value $v(s)$ at current state s is expressed recursively in terms of the immediate reward $r(s)$ and the values $v(s')$ of all possible next states, without explicitely invoking all future rewards. Based on this *bootstrap* idea of the Bellman equation, a multi-step MRP problem can be expressed recursively as a subproblem concerning only a single state transition from s to s', as a backward induction to find current state value from all future ones. For this reason, the Bellman equation plays an essential role in future discussion of some important algorithms in reinforcement learning.

The Bellman equation in Eq. (8.8) holds for all states $s_1, \cdots, s_N$, and the resulting N equations can be expressed in vector form as:

$$\begin{aligned} \mathbf{v} &= \begin{bmatrix} v(s_1) \\ \vdots \\ v(s_N) \end{bmatrix} = \begin{bmatrix} r(s_1) \\ \vdots \\ r(s_N) \end{bmatrix} + \gamma \begin{bmatrix} P(s_1|s_1) & \cdots & P(s_N|s_1) \\ \vdots & \ddots & \vdots \\ P(s_1|s_N) & \cdots & P(s_N|s_N) \end{bmatrix} \begin{bmatrix} v(s_1) \\ \vdots \\ v(s_N) \end{bmatrix} \\ &= \mathbf{r} + \gamma \mathbf{P} \mathbf{v} \end{aligned} \tag{8.9}$$

P has already been defined.

where $\mathbf{P}$ is called the *stochastic matrix*. Solving this linear equation system of size N we find

We already know that the system is of size N.

$$\mathbf{v} = (\mathbf{I} - \gamma \mathbf{P})^{-1} \mathbf{r} \tag{8.10}$$

Such a solution exists as matrix $\mathbf{I} - \gamma\mathbf{P}$ is invertible. This can be shown by noting that all eigenvalues of the stochastic matrix $\mathbf{P}$ are not greater than 1: $|\lambda| \leq 1$ (Section A.9), and all eigenvalues of $\gamma\mathbf{P}$ are smaller than 1: $\gamma\lambda < 1$, i.e., the eigenvalue of $\mathbf{I} - \gamma\mathbf{P}$ is greater than zero: $1 - \gamma\lambda > 0$. Consequently the determinant $\det(\mathbf{I} - \gamma\mathbf{P})$ of this coefficient matrix, as the product of all its nonzero eigenvalues, is non-zero and therefore matrix $\det(\mathbf{I} - \gamma\mathbf{P})$ is invertible.

"Dynamic programming" has already been defined.

Alternatively, the Bellman equation in Eq. (8.8) can also be solved iteratively by the general method of *dynamic programming (DP)*, which solves a multi-stage planning problem by backward induction and find the value function recursively. We first rewrite the Bellman equation as

$$\mathbf{v} = \mathbf{r} + \gamma \mathbf{P} \mathbf{v} = B(\mathbf{v}) \tag{8.11}$$

where $B(\mathbf{v}) = \mathbf{r} + \gamma\mathbf{P}\mathbf{v}$ defined as a vector-valued function (an operator)

applied to the vector argument $\mathbf{v}$. Now the Bellman equation can be solved iteratively from an arbitrary initial value, such as $\mathbf{v}_0 = \mathbf{0}$:

$$\mathbf{v}_{n+1} = B(\mathbf{v}_n) = \mathbf{r} + \gamma \mathbf{P}\mathbf{v}_n \tag{8.12}$$

"Fixed point" and "contraction function" have already been defined, and should not be italicized.

This iteration will always converge to the root of the equation, the *fixed point* of function $B(\mathbf{v})$, as it is a *contraction mapping* first introduced in Section 1.3, that satisfies:

$$\begin{aligned} ||B(\mathbf{v}_i) - B(\mathbf{v}_j)|| &= ||\mathbf{r} + \gamma\mathbf{P}\mathbf{v}_i - (\mathbf{r} + \gamma\mathbf{P}\mathbf{v}_j)\,|| = \gamma||\mathbf{P}(\mathbf{v}_i - \mathbf{v}_j)|| \\ &\leq \gamma||\mathbf{P}||\;||\mathbf{v}_i - \mathbf{v}_j|| = \gamma||\mathbf{v}_i - \mathbf{v}_j|| < ||\mathbf{v}_i - \mathbf{v}_j|| \end{aligned} \tag{8.13}$$

That $B(\mathbf{v})$ is a contraction can also be proven by showing the norm of its Jacobian matrix, its derivative with respect to its vector argument $\mathbf{v}$, is smaller than 1. We first find the Jacobian matrix

$$\mathbf{J}_B = \frac{d}{d\mathbf{v}}B(\mathbf{v}) = \frac{d}{d\mathbf{v}}(\mathbf{r} + \gamma\mathbf{B}\mathbf{v}) = \gamma\mathbf{P} \tag{8.14}$$

and then find its p-norm with $p = \infty$ (equivalent to $p = 1,\ 2$), the maximum absolute row sum:

$$||\mathbf{J}_B||_{p=\infty} = ||\gamma\mathbf{P}||_\infty = \gamma||\mathbf{P}||_\infty = \gamma \max_{1\leq i\leq N} \sum_{j=1}^{N} |P_{ij}| = \gamma < 1 \tag{8.15}$$

The last equality is due to Eq. (8.3), i.e., $||\mathbf{P}||_\infty = 1$.

This explanation would benefit from a concrete real-world example to illustrate the differences between actions, states, state transitions, and policies.

In summary, the Bellman equation in Eq. (8.8) can be solved by either of the two methods in Eqs. (8.10) and (8.12), so long as $\gamma < 1$. When the size of state space $|S| = N$ is large, the complexity $O(N^3)$ is too high, then the iterative method may be more efficient.

This is a run-on sentence. End the first sentence after "denoted by pi," and make everything after that its own sentence.

Based on the definition of MRP, we further define *Markov decision process (MDP)* as an MRP with certain decision-making rule called *policy*, denoted by π, following the policy the agent takes one of the actions $a \in A(s)$ available at each state s to control the state transition of the process. An MDP can therefore be described by a tuple $< S, P, R, A, \gamma >$, in which A is the set of all actions. The policy π can be either stochastic denoted by $\pi(a|s) = P(a_t = a|s_t = s)$ as the conditional probability of taking action a in state s, or deterministic denoted by $a = \pi(s)$. A policy is *soft* if any of the actions available at a state is possible to be taken, i.e., $\pi(a|s) > 0$ for all $a \in A(s)$ and all $s \in S$. In particular, a policy is ϵ-soft if $\pi(a|s) \geq \epsilon/|A(s)|$ for some small value of ϵ.

This sentence is confusingly worded. It would probably be clearer if broken into several shorter sentences.

If an action can't be taken at state s, how is it "available" at that state? The definition of "available" needs to be made explicit.

As there are $|A(s_n)|$ possible actions to take in state $s_n \in S$, the total number of policies is $|A(s_1)|\cdots|A(s_N)| = \Pi_{n=1}^{N}|A(s_n)|$. If the number of available actions $|A|$ is the same for all $|S|$ states, then the total number of policies is $|A|^{|S|}$.

Clarify that this is the number of *deterministic* policies for an MDP. There are potentially infinite possible probabilistic policies.

Following a given policy from an initial state s_0 will result in a sequence of state transitions called a *trajectory* of the episode:

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \cdots s_T \tag{8.16}$$

of which the state visited at the t-th step denoted by s_t can be any of the N

It might be better simply to use different notation for the t-th state visited.

states in $S = \{s_1, \cdots, s_N\}$ of the MDP, not to be confused with the t-th state in S. Note that in an episode some of the states in S may be visited multiple times, while some others may never be visited. Also note that due to the random nature of the MDP, following the same policy may not result in the same trajectory.

Clarify that you're referring to stochastic MDPs.

Define v(s0) before you state the goal: "The value v(s0) at the start state is the expected value of the sum of those accumulated rewards."

Along the trajectory of an episode, an accumulation of discounted rewards from all states visited will be received. Our goal is to find the optimal policy as a sequence of actions $a_0, a_1, a_2, \cdots$ for a given MDP model of the environment in terms of its state transition dynamics and rewards, so that the value $v(s_0)$ at the start state s_0, the expectation of the sum of all discounted future rewards, is maximized. This optimization problem can be solved by the method of dynamic programming, and such a process is called *planning*.

Rewrite for active voice: "Along the trajectory of an episode, an MDP will accumulate discounted rewards from all states visited."

"Planning" has already been defined.

In an MDP, both the next state s' and reward r are assumed to be random variables with joint probability $p(s', r|s, a) = P(s_{t+1} = s', r_{t+1} = r|s_t = s, a_t = a)$ conditioned on the action a and the previous state s. The state transition probability becomes

We already know what a policy is and what an MDP model is. This sentence is much clearer if those definitions are cut: "Our goal is to find the optimal policy for a given MDP such that the v(s0) is maximized."

$$p(s'|s, a) = P(s_{t+1} = s'|s_t = s, a_t = a) = \sum_{r \in R} p(s', r|s, a) \tag{8.17}$$

and the expected reward r received after arriving at the current is

$$r(s, a) = E[r_{t+1}|s_t = s, a_t = a] = \sum_r r \; p(r|s, a) = \sum_r r \sum_{s'} p(s', r|s, a) \tag{8.18}$$

Following a specific random policy $\pi(a|s)$, the agent takes an action $a \in A(s)$ to transit from the current state $s_t = s$ to the next state $s_{t+1} = s'$ with probability

$$p_\pi(s'|s) = P(s_{t+1} = s'|s_t = s) = \sum_{a \in A(s)} \pi(a|s) \; p(s'|s, a) \tag{8.19}$$

satisfying $\sum_{s'} P_\pi(s'|s) = 1$ and the reward in state s:

$$r_\pi(s) = E_\pi[r_{t+1}|s_t = s] = \sum_{a \in A(s)} \pi(a|s) \; r(s, a) \tag{8.20}$$

Overly formal/academic language. This could be more directly stated: "There are two important functions with respect to a given policy pi of an MDP."

where E_π denotes the expectation with respect to a certain policy $\pi(a|s)$. The summation over all available actions $a \in A(s)$ in state s will be abbreviated to a in the following.

We further define two important functions with respect to a given policy π of an MDP:

- The *state value function* $v_\pi(s)$ is the expected return at each state s of the MDP while following policy π, similar to the value function of an MRP as $v(s)$ in Eq. (8.8), but now treated as a function of action a as well as state

This sentence would be clearer if broken into several smaller sentences.

s:

$$\begin{aligned} v_\pi(s) &= E_\pi[G_t|s_t = s] \\ &= \sum_a \pi(a|s) \left(r(s,a) + \gamma \sum_{s'} P(s'|s,a)\, v_\pi(s') \right) \\ &= \sum_a \pi(a|s)\, q_\pi(s,a) \end{aligned} \tag{8.21}$$

where $q_\pi(s,a)$ is defined below.

- The *state-action value function* $q_\pi(s,a)$ is the expected return of taking a specific action a (irrelevant to policy π) in state s, and then following π in all subsequent states:

$$\begin{aligned} q_\pi(s,a) &= E_\pi\left[G_t|s_t = s, a_t = a\right] \\ &= r(s,a) + \gamma \sum_{s'} P(s'|s,a)\, v_\pi(s') \\ &= r(s,a) + \gamma \sum_{s'} P(s'|s,a) \sum_{a'} \pi(a'|s')\, q_\pi(s',a')) \end{aligned} \tag{8.22}$$

where $v_\pi(s')$ is recursively represented as a weighted sum of $q_\pi(s',a')$ based on Eq. (8.21).

The state-action value $q_\pi(s,a)$ plays a more important role than $v_\pi(s)$ as it allows the freedom of taking any action independent of a given policy π, thereby allowing the opportunity to improve an existing policy, e.g., by taking a *greedy* action to maximize the value $q_\pi(s,a)$. The state-action value function $q_\pi(s,a)$ is often abbreviated as the action value or *Q-value* for convenience in future discussions.

As a function of the state-action pair (s,a), the state action value $q_\pi(s,a)$ can be represented as a table of $|S|$ rows each for one of the states s, and $|A|$ columns each for one of the actions a. The Q-value for each state-action pair stored in the table can be updated iteratively by various algorithms for learning the Q-values.

In particular, if the policy is deterministic with $\pi(a|s) = 1$ and $a = \pi(s)$, then we have

$$P_\pi(s'|s) = P(s'|s,a), \qquad r_\pi(s) = r(s,a) \tag{8.23}$$

and the Bellman equations in Eqs. (8.21) and (8.22) become the same:

$$v_\pi(s) = r(s,a) + \gamma \sum_{s'} P(s'|s,a)\, v_\pi(s') = q_\pi(s, \pi(s)) \tag{8.24}$$

same as the Bellman equation of an MRP in Eq. (8.8), and can be solved iteratively if $\gamma < 1$, s as in Eq. (8.12).

The figure above illustrates the Bellman equations in Eqs. (8.21) and (8.22), showing the bootstrapping of the state value function in the dashed box on the left, and that of the action value function in dashed box on the right. Specifically

"Greedy" has already been defined.

Sometimes you hyphenate "state-action" and sometimes you don't. Either is fine, but it should be consistent.

This description of "bootstrapping" doesn't make it clear why this method is a bootstrapping method and others aren't. Defining bootstrapping when it's first used will clear this up.

here the word bootstrapping means the iterative method that updates the estimated value at a state s based on the estimated value of the next state s'. After taking one of the actions $a \in A(s)$ available in state s based on policy $\pi(a|s)$, an immediate reward $r(s, a)$ is received. However, which state the MDP will transit into as the next state s' is random depending on $P(s'|s, a)$ of the environment but independent of the policy. This uncertainty is represented graphically by a dashed circle called a *chance node* associated with the action value $q_\pi(s, a)$ as the sum of the immediate reward $r(s, a)$ and the expected value of the next state, the weighted average of values $v(s')$ of all possible state $s' \in S$.

This sentence is confusing. The point you're trying to make is that once the policy has decided on an action, there's still uncertainty about the next state, but that's not immediately clear from the language.

8.3 Model-Based Planning

"Of the environment" doesn't add much information and makes the sentence harder to understand.

Given a complete MDP model in terms of $p(s', r|s, a)$ of the environment, the goal of model-based planning is to find the optimal policy π^* that achieves the maximum value:

$$v^*(s) = \max_\pi v_\pi(s), \qquad \pi^*(s) = \arg\max_\pi v_\pi(s) \geq \pi \quad \forall \pi \tag{8.25}$$

based on the partial order that compares different policies according to the corresponding values:

$$\begin{aligned} \text{If} \quad & v_{\pi_1}(s) \geq v_{\pi_2}(s) \geq \cdots v_{\pi_k}(s), \quad \forall s \in S \\ \text{Then} \quad & \pi_1 \geq \pi_2 \geq \cdots \geq \pi_k \end{aligned} \tag{8.26}$$

The problem of policy optimization can be addressed based on two subproblems: *policy evaluation* to measure how good a policy is, and *policy improvement* to get a better policy for higher values.

Why do you define |S| as a separate variable N, when you don't define |A| as a separate variable?

One way to solve this optimization problem is to use brute-force search to enumerate all $|A(s)|$ available policies at each of the $N = |S|$ states $s \in S$. If we assume the numbers of available actions in all states are the same for simplicity, i.e., $A(s_1) = \cdots = A(s_N) = A$, then the computational complexity of this method is $O(|A|^{|S|})$, likely to be too high for such a brute-force search to be practical if N is large.

"Greedy" has already been defined.

A more efficient and therefore more practical way to optimize the policy is to do it iteratively. Consider a simpler task of improving upon an existing deterministic policy π at a single transition step from the current state s to a next state s'. Instead of taking the action $a = \pi(s)$ dictated by the policy, we can improve it by taking the *greedy action* that maximizes the action value function in Eq. (8.22), while subsequent steps still follow the old policy π. Now we get a new policy:

$$\pi'(s) = \arg\max_a q_\pi(s, a) = \arg\max_a \left[r(s, a) + \gamma \sum_{s'} P(s'|s, a) v_\pi(s') \right] \tag{8.27}$$

so that the action value is higher than that of the old one:

$$q_\pi(s,\pi'(s)) = \max_a q_\pi(s,a) = \max_a \left[r(s,a) + \gamma \sum_{s'} P(s'|s,a) v_\pi(s') \right] \geq q_\pi(s,\pi(s)) = v_\pi(s) \tag{8.28}$$

where the last equality is due to Eq. (8.24) for a deterministic policy.

While it is obvious that this greedy method will result in a higher return for the single state transition from s to s', we can further prove the following *policy improvement theorem* stating that following this greedy method in all subsequent states, we will get a new policy π' that achieves higher values than the those of the old one π at all states:

$$v_{\pi'}(s) \geq v_\pi(s), \qquad \forall s \in S \tag{8.29}$$

The proof is by recursively applying the greedy method to replace the old policy $\pi(s)$ by the new one $\pi'(s)$ for a higher value for each of the subsequent steps one at a time:

$$\begin{aligned}
v_\pi(s) &\leq q_\pi(s,\pi'(s)) = \max_a q_\pi(s,a) = \max_a \left[r(s,a) + \gamma \sum_{s'} P(s'|s,a) v_\pi(s') \right] \\
&= r(s,\pi'(s)) + \gamma \sum_{s'} P(s'|s,\pi'(s)) v_\pi(s') \\
&\leq r(s,\pi'(s)) + \gamma \sum_{s'} P(s'|s,\pi'(s)) \max_{a'} q_\pi(s',a') \\
&= r(s,\pi'(s)) + \gamma \sum_{s'} P(s'|s,\pi'(s)) \left[r(s',\pi'(s)) + \gamma \sum_{s''} P(s''|s',\pi'(s')) v_\pi(s'') \right] \\
&= r(s,\pi'(s)) + \gamma r(s',\pi'(s)) + \gamma \sum_{s'} P(s'|s,\pi'(s)) \left[\gamma \sum_{s''} P(s''|s',\pi'(s')) v_\pi(s'') \right] \\
&\leq \cdots \\
&= r(s,\pi'(s)) + \gamma r(s',\pi'(s')) + \gamma^2 r(s'',\pi'(s'') + \cdots \\
&= v_{\pi'}(s)
\end{aligned} \tag{8.30}$$

i.e., $\pi'(s) \geq \pi(s)$.

The optimal state value $v^*(s)$ and action value $q^*(s,a)$ can therefore be achieved by replacing the weighted average over all possible actions at each state (s and s') in the Bellman equations in Eqs. (8.21) and (8.22) by their maximum:

$$v^*(s) = \max_a q^*(s,a) = \max_a \left(r(s,a) + \gamma \sum_{s'} P(s'|s,a) v^*(s') \right) \tag{8.31}$$

and

$$q^*(s,a) = r(s,a) + \gamma \sum_{s'} P(s'|s,a) v^*(s') = r(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} q^*(s',a') \tag{8.32}$$

Eqs. (8.31) and (8.32) are the *Bellman optimality equations*, which should hold for all N states in $S = \{s_1, \cdots, s_N\}$ and can be written in vector form:

$$\mathbf{v}^* = \max_a (\mathbf{r}_a + \gamma \mathbf{P}_a \mathbf{v}^*) \tag{8.33}$$

where

$$\mathbf{v}^* = \begin{bmatrix} v^*(s_1) \\ \vdots \\ v^*(s_N) \end{bmatrix}, \quad \mathbf{r}_a = \begin{bmatrix} r(s_1, a) \\ \vdots \\ r(s_N, a) \end{bmatrix}, \quad \mathbf{P}_a = \begin{bmatrix} P(s_1|s_1, a) & \cdots & P(s_N|s_1, a) \\ \vdots & \ddots & \vdots \\ P(s_1|s_N, a) & \cdots & P(s_N|s_N, a) \end{bmatrix} \tag{8.34}$$

This nonlinear equation system can be solved iteratively to for $\mathbf{v}^*$:

$$\mathbf{v}^*_{n+1} = \max_a (\mathbf{r}_a + \gamma \mathbf{P}_a \mathbf{v}^*_n) = B(\mathbf{v}_n) \tag{8.35}$$

This iteration is to be used for policy evaluation in the algorithm *value iteration* below for finding the optimal policy. This iteration will converge as function $B(\mathbf{v})$ is a contraction mapping with $\gamma < 1$:

$$\begin{aligned} ||B(\mathbf{v}_i) - B(\mathbf{v}_j)|| &= || \max_{a_i} (\mathbf{r}_{a_i} + \gamma \mathbf{P}_{a_i} \mathbf{v}_i) - \max_{a_j} \left(\mathbf{r}_{a_j} + \gamma \mathbf{P}_{a_j} \mathbf{v}_j\right) || \\ &\leq || \max_{a_i} (\mathbf{r}_{a_i} + \gamma \mathbf{P}_{a_i} \mathbf{v}_i - \mathbf{r}_{a_i} - \gamma \mathbf{P}_{a_i} \mathbf{v}_j) || \\ &= \max_{a_i} \gamma ||\mathbf{P}_{a_i} (\mathbf{v}_i - \mathbf{v}_j)|| \leq \max_{a_i} \gamma ||\mathbf{P}_{a_i}|| \, ||\mathbf{v}_i - \mathbf{v}_j|| \\ &= \gamma ||\mathbf{v}_i - \mathbf{v}_j|| < ||\mathbf{v}_i - \mathbf{v}_j|| \end{aligned} \tag{8.36}$$

where $||\mathbf{P}|| = ||\mathbf{P}||_\infty = 1$ is the p-norm ($p = \infty$) of a stochastic matrix, which is known to be 1.

Given the complete information of the MDP model in terms of $p(s', r|s, a)$, we can find the optimal policy π^* using either *policy iteration (PI)* or *value iteration (VI)* based on the DP method.

- **Policy Iteration**

  Carry out the following two processes iteratively from some arbibrary initial policy π until convergence when π no longer changes:

  – Policy evaluation:

    Given a deterministic policy π, Eq. (8.21) becomes the Bellman equation in Eq. (8.24), a linear equation system of $N = |S|$ equations of N unknowns $v_\pi(s_1), \cdots, v_\pi(s_N)$, which can be solved iteratively as in Eq. (8.12) to find values at all states:

    $v^\pi_{n+1}(s) = r(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) v^\pi_n(s') \qquad \forall s \in S$

  – Policy improvement:

    Given $v_\pi(s)$ based on policy π, get a better policy π' by the greedy method as in Eq. (8.27):

    $\pi'(s) = \arg\max_a [r(s, a) + \gamma \sum_{s'} P(s'|s, \pi(s)) v_\pi(s')] \qquad \forall s \in S$

  We see that in each round of the policy iteration, two intertwined and interacting processes, the policy evaluation and policy improvement, are carried out alternatively, one depending on the other, and one completing

before the other starts. The value functions $v_\pi(s)$ at all states are sysmeticaly evaluated before they are updated, and the policy $\pi(a|s)$ at all states is updated before it is evaluated. While these two processes chase each other as moving target iteratively and eventually converge to the optimal policy π^* achieving the optimal value v^*. This approach is said to be *synchronous*, as shown here:

$$\pi_o \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \cdots \xrightarrow{I} \pi^* \xrightarrow{E} v_{\pi^*} = v^* \tag{8.37}$$

The pseudo code for the algorithm is listed below:

Initialize:

$v(s) \in \mathbb{R},\ v(s) = 0$ if s is terminal node, $\pi(s) \in A(s),\ \ \forall s \in S$;

Set tolerence *tol* (small positive value);

repeat

Policy Evaluation (find $v(s)\ \ \forall s \in S$):

repeat

$\delta = 0$

for $\forall s \in S$:

$v'(s) = r(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) v(s')$

$\delta = \max(\delta, |v(s) - v'(s)|)$

until $\delta < tol$

Policy Improvement (by taking greedy action):

$Done = T$

for each $s \in S$

$\pi'(s) = \arg\max_a [r(s, a) + \gamma \sum_{s'} P(s'|s, \pi(s)) v(s')]$

if $\pi'(s) \neq \pi(s),\ \ Done = F$

end for

until *done*

- **Value Iteration:**

This sentence is overly long and awkwardly worded. Everything after the first "inner iteration" can be cut, and it will make the sentence clearer.

One drawback of the policy iteration method is the high computational complexity of the inner iteration for policy evaluation inside the outer iteration for policy evaluation. To speed up this process, we could terminate the inner iteration early before convergence. In the extreme case, we simply combine policy evaluation and improvement into a single step, resulting in the following value iteration contains two steps:

– Find the optimal values:

Based on some initial value $v_0(s)$, solve the Bellman optimality equation in Eq. (8.31) by iteration in Eq. (8.35):

$$v_{n+1}(s) = \max_a \left[r(s, a) + \gamma \sum_{s'} P(s'|s, a) v_n(s') \right] \qquad \forall s \in S \tag{8.38}$$

– Find the optimal policy:

Find $\pi^*(s)$ that maximizes $v^*(s)$:

$$\pi^*(s) = \arg\max_a \left[r(s,a) + \gamma \sum_{s'} P(s'|s,a) v_n(s') \right] \qquad \forall s \in S \quad (8.39)$$

Here is the pseudo code for the algorithm:

Initialize: $v(s) = 0 \ \ \forall s \in S$
repeat
  $\delta = 0$;
  for each $s \in S$
    $v'(s) = \max_a [r(s,a) + \gamma \sum_{s'} P(s'|s,a) v(s')]$
    $\delta = \max(\delta, |v'(s) - v(s)|)$
    $v(s) = v'(s)$
  end for
until $\delta < tol$
$\pi^*(s) = \arg\max_a [r(s,a) + \gamma \sum_{s'} P(s'|s,a) v(s')]$

In the PI method considered above, the value functions at all states are evaluated before the policy is updated for all states, as illustrated below:

This synchronous policy iteration can be modified so that the values at some or even one of the states are evaluated in-place before other state values are updated, and the policy at some or even one of the states are updated in-place before that at other states, resulting in an *asynchronous* method called *general policy iteration (GPI)*, to be discussed in detail in the next sections on model-free control.

This sentence is repetitive and difficult to follow. Consider instead: "This synchronous policy iteration can be modified so that the values at some states are evaluated in-place before the other state values are updated. The same thing can be done for policy evaluation. This results in...""

If this is referring to a figure, name the figure explicitly.

Since this section immediately follows, there's no need to state what's going to be discussed in it.

8.4 Model-Free Evaluation and Control

The previously discussed dynamic programming methods find the optimal policy based on the assumption that the MDP model of the environment is completely available, i.e., the dynamics of MDP in terms of its state transition and reward mechanism are known. A given policy π can then be evaluated based on the transition probabilities $p(s'|s,a)$, and improved based on greedy action selection at each state. Such a model-based optimization problem, called *planning*, involves no learning of the environment.

"Planning" has already been defined.

Now we consider the optimization problem with an unknown MDP model of the environment. In this model-free problem, called *control*, the state transition and reward probabilities of the MDP model are unknown, the value functions can no longer be calculated directly based on Eqs. (8.21) and (8.22) as in model-based planning. Instead, now we need to learn the MDP model by sampling the environment, i.e., by repeatedly running a large number of episodes of the stochastic process of the environment while following some given policy, and estimate the value functions as the average of the actual returns received during the sampling process. This method is generally referred to as *Monte Carlo (MC)*

"Control" is a modifier for "problem"—this should be, "In this model-free problem, called a *control problem*..."

This sentence is long. It would be clearer if broken into several shorter sentences.

method. At the same time, the given policy can also be improved to gradually approach optimality.

This approach is called *general policy iteration (GPI)*, as illustrated in Eq. (8.40) below, similar to the *policy iteration (PI)* algorithm for model-based planning illustrated in Eq. (8.37). In general, all model-free control algorithms are based on GPI by which the two alternating processes of policy evaluation and policy imprvement are carried out iteratively. This GPI can also be similarly illustrated in Fig. **??** for the GI.

Based on context, I think that "GI" refers to policy iteration, but previously that's been referred to as "PI"; GI has never been defined.

$$\pi_o \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \xrightarrow{I} \cdots \xrightarrow{I} \pi^* \xrightarrow{E} q_{\pi^*} = v^* \tag{8.40}$$

While GPI and GI illustrated in respectively in Eqs. (8.40) and (8.37) may look similar to each other, there are some essential differences between the two, as listed below.

The reference to the equations is awkward and makes the sentence hard to parse. Since both equations are relatively recent, it's probably fine to leave out the reference entirely.

It's probably not necessary to list this in bullet format.

- First, in model-free control the policy improvement is based on the action-value function $q_\pi(s,a)$, instead of the state value function $v_\pi(s)$ in model-based planning. This is because without the a specific MDP model the state value function can no longer be calculated, while the action value function can still be estimated based on the actual rewards received while sampling the environment. Similar to how we improve the policy by taking a greedy action to achieve a higher state value in model-based planning, here we improve the policy by taking an action different from that dictated by the given policy π to achieve a higher action value based on some ϵ-greedy method, as discussed below.
- Second, as estimating the action value by the MC method requires running a large number of episodes while the sampling of the environment, the computational complexity for policy evaluation in the model-free case is much higher than that in model-based planning. To speed up this process in model-free control, the iterative policy evaluation is no longer fully carried for it to converge to the true action value of the policy. Instead, in model-free control, the iterative update of the action value is carried out only once, followed immediately by the next phase of policy improvement. Such an estimated action value is innevitably very noisy, but due to the much reduced complexity, we can afford to run a large number of sample episodes while sampling the envirionment.
- Third, in the PI process of model-based planning, the action value is found based on all available actions at each state (Eq. (8.22)) during policy evaluation, and we only need to *exploit* the greedy action at each state to find the policy that achieves the maximum action value $q_\pi(s)$ (Eq. (8.27)) during policy improvement. However, in model-free control, we have to learn the action value $q_\pi(s,a)$, as a function of action $a \in A(s)$ as well as state s by sampling the environment. Such an estimated action value based only on some partial sample data is inevitably noisy, especially in the early

"Action-value" is sometimes hyphenated and sometimes not. Either is fine, but it should be consistent.

stage of learning when many of the states have not yet been visited yet. We therefore need to *explore* all actions at each state to better learn the value function, as well as to exploit the greedy action to improve the policy.

This sentence is very long. It would be clearer if broken into several shorter sentences.

This issue of *exploration* versus *exploitation* can be addressed by the ϵ-*greedy method* to make a proper trade-off between the exploitation of the greedy action and the exploration of other non-greedy actions, so that the agent can be exposed to all possible state-action pairs and gradually learn the action value function while at the same time also improve the policy being followed. In this method, we define $\epsilon \in [0,\ 1]$ as the probability to explore any randomly chosen action $a \in |A(S)|$ out of all actions available in state s, each with an equal probability $\epsilon/|A(s)|$, and $1-\epsilon$ is the probability of taking the greedy action as one of the $|A(s)|$ actions, with probability $1-\epsilon+\epsilon/|A(s)|$.

It's difficult to keep track of which probability goes with which action.

$$\pi'(a|s) = \begin{cases} \arg\max_a q(s,a) & Pr = 1-\epsilon+\epsilon/|A(s)| \\ \text{any } a \in A(s) & Pr = \epsilon/|A(s)| \end{cases} \tag{8.41}$$

3/6 is 1/2.

For example, if $|A(s)| = 4$ and $\epsilon = 2/3$, the greedy action may be chosen with probability $1-\epsilon+\epsilon/|A(s)| = 3/6$, while each of the remaining three non-greedy actions may be chosen with probability $\epsilon/|A(s)| = 1/6$. Such a policy describes the behavior of the agent, and is therefore called *behavior policy*, different from the policy being followed.

It's not clear before this that sigma can change throughout the algorithm. Having a sentence to establish that first will make this part easier to understand.

We further note that a larger value of ϵ (close to 1) can be used to emphasize exploration at the early stage of the iteration when many state-action pairs have not been visited yet, while progressively smaller ϵ values (approaching to 0) can be used to emphasize exploitation for higher values in the later stage of the iteration when the action value function has been more reliably learned. For example, we can let $\epsilon_k = 1/k$ be inversely proportional to the current number of iterations k, so that the larger ϵ values at the early stage of the iteration allow more state-action pairs to be explored, while smaller ϵ values in the later stage of the iteration encourage exploitation of the greedy actions. To the limit $k \to \infty$ and $\epsilon \longrightarrow 0$, the ϵ-greedy method approaches absolute greedy method.

Similar to how we proved the policy improvement theorem in Eq. (8.30) stating that the greedy policy $\pi'(s)$ achieves state value $v_{\pi'}(s)$ no lower than $v_\pi(s)$ achieved by the original policy π, i.e., $\pi'(s) \geq \pi(s)$, here we can also prove the same policy improvement theorem stating that the ϵ-greedy policy π' achieves action value $q_{\pi'}(s,a)$ no lower than $q_\pi(s,a)$ by the original policy π:

$$\begin{aligned} q_\pi(s,\pi'(s)) &= \sum_a \pi'(a|s) q_\pi(s,a) \\ &= \frac{\epsilon}{|A(s)|} \sum_a q_\pi(s,a) + (1-\epsilon) \max_a q_\pi(s,a) \\ &\geq \frac{\epsilon}{|A(s)|} \sum_a q_\pi(s,a) + (1-\epsilon) \sum_a \frac{\pi(a|s) - \epsilon/|A(s)|}{1-\epsilon} q_\pi(s,a) \end{aligned} \tag{8.42}$$

where the inequality is due to the fact that the maximum value of $q_\pi(s,a)$ is no

less than the average of action values $q_\pi(s,a)$ over all $a \in A(s)$ weighted by some normalized coefficients adding up to 1:

$$\sum_a \frac{\pi(a|s) - \epsilon/|A(s)|}{1-\epsilon} = \frac{1}{1-\epsilon}\sum_a \left(\pi(a|s) - \frac{\epsilon}{|A(s)|}\right) = \frac{1}{1-\epsilon}(1-\epsilon) = 1 \quad (8.43)$$

Continuing the equation above we further get

$$\begin{aligned} q_\pi(s,\pi'(s)) &\geq \frac{\epsilon}{|A(s)|}\sum_a q_\pi(s,a) - \frac{\epsilon}{|A(s)|}\sum_a q_\pi(s,a) + \sum_a \pi(a|s) q_\pi(s,a) \\ &= \sum_a \pi(a|s) q_\pi(s,a) = v_\pi(s) \end{aligned} \quad (8.44)$$

Specify the figure by number.

The iterative process of the GPI method for model-free control is illustrated in the figure below, similar to the PI method for model-based planning based planning illustrated in Fig. **??**, but with the state value $v_\pi(s)$ replaced by the action value $q_\pi(s,a)$, and the greedy method replaced by ϵ-greedy method.

Also, here the estimated action value $\tilde{q}$ is obtained with only one iteration for policy evaluation, instead of a much more accurate estimate that could otherwise be obtained if the iterative evaluation was fully carried out to its convergence (the top line). At the same the time, based on the estimated $\tilde{q}$ policy improvement is carried out by the ϵ-greedy method (the bottom line).

I don't understand what this sentence is saying.

The main task in model-free control is to evaluate the state and action-value functions given in Eqs. (8.21) and (8.22) while following a given policy π without a specific MDP model. This is done by sampling of the dynamic process of the environment, and then estimating the value functions as the average of the actual returns received by the agent. Specifically, we run a large number of episodes (all assumed to terminate) of the MDP model of the environment by taking actions $a_t = \pi(s_t)$ in each state s_t, $(t = 1, \cdots, T)$:

$$s_0 \to (a_1, r_1, s_1) \to (a_2, r_2, s_2) \to \cdots \to (a_T, r_T, s_T) \quad (8.45)$$

and estimate the value functions based on the averaged returns G_t actually received from these episodes.

"Trajectory" has already been defined.

Such a sequence of states visited is called a *trajectory*, and the trajectories of different episodes are in general different from each another due to the random nature of the environment. In the following sections we will consider specific algoithms for the implementation of the general model-free control discussed above.

To prepare for specific discussion of the GPI methods, we first consider a general problem of the estimation of the value of a random variable x as the running average of its samples beging collected in real time. Specifically the average $\hat{x}_n$ based on previous n samples $x_1, \cdots, x_n$ is updated incrementally upon receiving

a new sample x_{n+1}:

$$\begin{aligned}\hat{x}_{n+1} &= \frac{1}{n+1}\sum_{i=1}^{n+1} x_i = \frac{1}{n+1}\left(x_{n+1} + \sum_{i=1}^{n} x_i\right) \\ &= \frac{1}{n+1}(x_{n+1} + n\hat{x}_n) = \hat{x}_n + \frac{1}{n+1}(x_{n+1} - \hat{x}_n)\end{aligned} \tag{8.46}$$

This running average can be generalized to

$$\hat{x}_{n+1} = \hat{x}_n + \alpha(x_{n+1} - \hat{x}_n) \tag{8.47}$$

This equation can be interpreted as

$$\begin{aligned}\text{newEstimate} &= \text{oldEstimate} + \text{stepSize} \times (\text{target} - \text{oldEstimate}) \\ &= \text{oldEstimate} + \text{stepSize} \times \text{error} \\ &= \text{oldEstimate} + \text{increment}\end{aligned} \tag{8.48}$$

"Stochastic gradient descent" has already been defined.

and considered as one iterative step in the method of *stochastic gradient descent (SGD)*, for minimizing the squared error $\varepsilon = (x_{n+1} - \hat{x}_n)^2/2$ between the old average $\hat{x}_n$ and the latest sample x_{n+1}, and the second term is the gradient $d\varepsilon/d\hat{x}_n$ weighted by the step size $\alpha \in [0,\ 1]$, which controls how samples are weighted differently. In the extreme case when $\alpha = 1$, $\hat{x}_n = x_n$ with no contribution from any previous samples; on the other hand when α is close to 0, $\hat{x}_n$, the most recent samples has little contribution. We can therefore properly adjust α to fit our specific need. For example, if we gradually reduce α, then the estimated average will become stabilized while enough samples have been collected. On the other hand, if we let $0 < \alpha < 1/(n+1)$, the more recent samples are weighted more heavily than the earlier ones. Such a strategy is suitable when estimating parameters of a nonstationary system, such as an MDP with varying behaviors when the policy is being modified continuously.

This makes it sound like similar things happen when alpha is large and small.

Specifically in model-free control Eq. (8.47) can be used to iteratively update the estimated value functions while sampling the environment. Both the state and action values the expected return are estimated as the average of all previous sample returns, and they are updated incrementally when a new sample return G_{n+1}, the target, becomes available:

$$\begin{aligned}v_{n+1}(s) &\Leftarrow v_n(s) + \alpha(G_n - v_n(s)) \\ q_{n+1}(s,a) &\Leftarrow q_n(s,q) + \alpha(G_n - q_n(s,a))\end{aligned} \tag{8.49}$$

as we will see in the following sections.

8.4.1 Monte Carlo (MC) Algorithms

We first consider a simple problem of evaluating an existing policy π in terms of its value function $v_\pi(s) = E[\,G_t\,]$ at state $s = s_t$, the expectation of return G_t,

"The expectation of return Gt" makes this sentence hard to follow. Cut it or reword the sentence into two shorter sentences for more clarity.

which can be obtained in the model-based case by the Bellman equation in Eq. (8.24):

$$v_\pi(s) = r(s, \pi(s)) + \gamma \sum_{s'} p(s'|s, \pi(s)) v_\pi(s') \tag{8.50}$$

based on $v_\pi(s')$ of all possible next states each weighted by the corresponding transition probability $p(s'|s, a)$, which is no longer available now in the model-free case. We can still find the value function based on the MC method by running multiple sample episodes in the environment, and estimate $v_\pi(s)$ as the average of the actual returns G_t, which in turn can be calculated as the sum of discounted rewards $r_{t+1}, \cdots, r_T$ from the current state $s = s_t$ onward to the terminal state s_T, found at the end of the episode (assumed to have finite horizon).

This sentence is hard to follow because of how long it is. Consider cutting into several shorter sentences.

This method has two versions, *first-visit* and *every-visit*, depending on whether only the first or every visit to a state in the trajectory of an episode is counted in calculation of the return. The pseudo code below is for first-visit version due to the if statement, which can be removed for the every-visit version.

Input: a given policy π to be evaluated
Initialize: $G_t = [\ ]$ (empty lists) for all $t = 0, \cdots, T-1$
loop (for each episode)
  run episode to the end while following π to get sample data: $(s_0, a_0, r_1),\ (s_1, a_1, r_2), \cdots, (s_{T-1}, a_{T-1}, r$
  $G = 0$
  for $t = T-1, T-2, \cdots, 0$ (for each step)
    $G = \gamma G + r_{t+1}$
    if $S_t \notin \{S_0, \cdots, S_{t-1}\}$ (first visit)
      $G_t = [G_t, G]$ (append G to list G_t)
$V(S_t) = average(G_t),\ \forall t = 0,\ , \cdots, T-1$

The text runs into the margin and off the page.

In the every-visit case, all sample returns G_t from multiple visits to state s_t are used, although they may be correlated and not independent of each other, i.e., they are not necessarily i.i.d. samples and their average may be biased. This problem can be avoided in the first-visit case, where only the returns of the first visit to each state are counted as independent samples drawn from the same distribution, and their average is not biased. However, as only a fraction of the sample points collected is used in the calculation of the returns, the cost of this unbiased estimation is its high variance, which can be reduced only if a large number of episodes are used to get enough samples for a more statistically reliable average. This is the typical trade-off between variance and bias errors.

This makes it sound like we're deciding to designate the first visits as i.i.d., when in fact the first visits are inherently i.i.d., which is why the method works. This section might also benefit from an explanation of why the first visits are i.i.d.

In the code above, the function *average* finds the average of all elements of a list, only when all sample returns are available at the end of each episode. A more computationally efficient method (in terms of both space and complexity) is to find the average incrementally as in Eq. (8.49). Based on this incremental average, policy evaluation in terms of both the state and action values can also be carried out by the code below:

Initialize $v(s) = 0,\ \forall s$

loop (for each episode)
run episode following π to get $G_t,\ \forall t$
for $t = 0, \cdots, T-1$ (for each step)
if s_t is visited the first time
$$v(s_t) = v(s_t) + \alpha(G_t - v(s_t))$$

and

Initialize: $q(s,a) = 0,\ \forall (s,a)$
loop (for each episode)
run episode following π to get $G_t,\ \forall t$
for all (s,a) pairs visited
if (s,a) is visited the first time
$$q(s,a) = q(s,a) + \alpha(G_t - q(s,a))$$

The *if* condition can be removed for every-visit version of the algorithm.

We next consider the MC algorithm for model-free control, based on the generalized policy iteration of alternating and interacting policy evaluation and policy improvement. Specifically, while sampling the environment by following an existing policy π the action value $q_\pi(s)$, instead of the state value $v_\pi(s)$, is gradually learned and the policy is gradually improved at the same time. This iteration will converge to the optimal policy at the limit when the number of iterations goes to infinity.

The pseudo code for this algorithm is listed below.

Input: $\epsilon > 0,\ \alpha > 0$, and a policy π
Initialize: $q(s,a) = 0,\ \forall (s,a),\ \pi = \epsilon - \text{soft}$ (arbitrarily)
loop (for each episode)
run episode following π to get $G_0, \cdots, G_{T-1}$
for $t = 0, \cdots, T-1$
if (s_t, a_t) is visited the first time
$$q(s_t,a_t) = q(s_t,a_t) + \alpha(G_t - q(s_t,a_t))$$
$$a^* = \arg\max_a q(s_t,a_t)$$
for $\forall a \in A(s_t)$
$$\pi(a|s_t) = \begin{cases} 1 - \epsilon + \epsilon/|A(s_t)| & \text{if } a = a^* \\ \epsilon/|A(s_t)| & \text{else} \end{cases}$$

Here the ϵ-greedy policy $\pi(a|s)$ is based on action value function $q(s,a)$ (maximized by the greedy action). Through out the iteration, the policy is modified following the action value function, while at the same time the action value function is modified based on the policy, until the process converges to the desired optimality.

We note that the optimal policy $\pi(s)$ obtained by the algorithm above is not completely deterministic due to the ϵ-greedy approach, but it is said to be near-deterministic as the greedy action is favored over other non-greedy actions.

Italicize "near-deterministic."

8.4.2 Temporal Difference (TD) Algorithms

The temporal difference (TD) method is a combinaion of the MC method considered above and the bootstrapping DP method based on the Bellman equation. The main difference between the TD and MC methods is the target, the return G, in the incremental average in Eq. (8.49) for estimating either the state or action value functions. While in the MC method the target is the actual return G_t calculated at the end of the episode when all subsequent rewards $r_{t+1}, \cdots, r_T$ are available, here in the TD method the target, called *TD target*, is the sum of the immediate reward r_{t+1} and the previouosly estimated value $v_\pi(s_{t+1})$ at the next state:

$$G_t = r_{t+1} + \gamma v_\pi\ (s_{t+1}) \tag{8.51}$$

We see that the TD method is an in-place bootstrapping method, and it makes more efficient use of the sample data and updates more frequently the value functions being estimated and the policy being improved at every step of an episode, instead of at the end of the episode as in the MC method. Also, it can be used even if the dynamic process of the environment is non-episodic with infinite horizon.

Again, we first consider the simpler problem of policy evaluation. As in the MC method, we estimate the value function $v_\pi(s)$ as the average of the actual returns G found by running multiple episodes while sampling the environment. Substituting this TD target into the running average in Eq. (8.49) for the value function above, we get

$$\begin{aligned} v_\pi(s) &\Leftarrow v_\pi(s) + \alpha(G - v_\pi(s)) \\ &= v_\pi(s) + \alpha(r + \gamma v_\pi(s') - v_\pi(s)) = v_\pi(s) + \alpha\delta \end{aligned} \tag{8.52}$$

where δ, called the *TD error*, is the difference between the TD target and the previous estimate:

$$\delta = G - v_\pi(s) = r + \gamma v_\pi(s') - v_\pi(s) \tag{8.53}$$

and $v_\pi(s')$ is the current estimatee of the value of the next state (bootstrapping). It can be shown that the iteration of this TD method converges to the true value $v_\pi(s)$, if the step size is small enough.

Here is the pseudo code for policy evaluation using the TD method, based on parameters γ and α:

Input: a given policy π to be evaluated
Initialize: $v(s)$ for all $s \in S$ arbitrarily, $v(s) = 0$ for terminal state s, $\alpha \in (0,\ 1]$
loop (for each episode)
    $s = s_0$
    while s is not terminal (for each step)
        take action $a = \pi(s)$, get reward r and next state s'
        $v(s) = v(s) + \alpha[r + \gamma v(s') - v(s)]$
        $s = s'$

This is a "garden path" sentence—its grammar is ambiguous, leading the reader to think that it's going to say something other than it is. In this case, it makes it sound like the "updates" is a noun, when actually it's a verb. It can be fixed by rewording: "The TD method is an in-place bootstrapping method. It makes more efficient use of the sample data, and it updates the estimates of the value functions and policy at every step of an episode, instead of at the end of the episode as in the MC method."

The parenthetical isn't doing much here. If readers already have a deep understanding of bootstrapping, they'll likely already recognize what's happening; if they don't, it doesn't shed much light on the concept of bootstrapping or of temporal difference.

As the TD algorithm updates the value function at every step of the episode, it uses the sample data more frequently and efficiently, and therefore has lower variance error than the CM method that updates the estimated value function at the end of each episode. On the other hand, the TD method may be biased when compared to the first-visit MC method, due to the arbitrary initialization of the value functions.

These sentences would both be clearer if cut into several shorter sentences.

Based on the TD method for model-free policy evaluation, we now further consider the TD method for model-free control to gradually learn the optimal policy by updating the Q-values of all state-action pairs (s, a) estimated iteratively as the running average of the sample Q-values at each time step of an episode while sampling the environment.

The Q-values for all state-action pairs can be stored in a state-action table which is iteratively updated by running many episodes of the unknown MDP by the TD method to gradually approach the maximum Q-values achievable by taking each action at each state, i.e., the optimal action is the one with the highest Q-value.

"Such as a sigma-greedy policy" confuses this concept somewhat. As you elaborate below, an algorithm is on-policy even if the policy it's following is greedy. Cutting this clause would make the point clearer.

This method has two different flavors, the *on-policy* algorithm, which updates the Q-value by following the current policy such as an ϵ-greedy policy, and the *off-policy* algorithm, which updates the Q-value by taking actions different from the current policy such as the greedy action. In particular, if the policy currently being followed is greedy (instead of ϵ-greedy), the two algorithms are the same.

- **State-Action-Reward-State-Action (SARSA)**

  The Q-value $q(s, a)$ of each state-action pair (s, a) is estimated as the running average of the return G, the sum of the immmediate reward r and the action value $q(s', a')$ of the next state based on action a' dictated by the current policy (e.g.,ϵ-greedy):

Again, specifying "sigma-greedy" seems to confuse things more than it clarifies.

$$\begin{aligned} q(s, a) &\Leftarrow q(s, a) + \alpha(G - q(s, a)) \\ &= q(s, a) + \alpha(r + \gamma q(s', a') - q(s, a)) \end{aligned} \tag{8.54}$$

  This algorithm is called *SARSA*, as it updates the Q-value based on the current state s and action a, the immediate reward r, and the next state s' and action a'. The pseudo code of SARSA algorithm is listed below.

  Initialize: $q(s, a) = 0$ for all (s, a), $\alpha \in (0,\ 1]$, $\epsilon > 0$, denote ϵ-greedy policy $pi(a|s)$ by π

  loop (for each episode)

    $s = s_0$

    get action a according to π based on $q(s, a)$

    while s is not terminal (for each step)

      take action a, get reward r and next state s'

      get action a' according to π based on $q(s', a)$

      $q(s, a) = q(s, a) + \alpha[r + \gamma q(s', a') - q(s, a)]$

      $s = s',\ a = a'$

As a variation of SARSA, the *expected SARSA* updates the Q-value at state s based on the expected Q-value of the next state s', the weighted average of the Q-values resulting from all possible actions, instead of only one action. Consequently, the estimated Q-values by the expected SARSA have lower variance than SARSA, and a higher learning rate α can be used to speed up the learning process.

$$q(s,a) = q(s,a) + \alpha\left(r + \gamma \sum_a \pi(a|s')q(s',a) - q(s,a)\right) \tag{8.55}$$

- **Q-Learning**

  Same as SARSA, the Q-learning algorithm also estimates the Q-value $q(s,a)$ of each state-action pair (s,a) as the running average of the return G, the sum of the immediate reward r and the action value $q(s',a')$ of the next state based on the greedy action a' that maximizes the next state value $q(s',a')$, different from that dictated by the current policy.

$$q(s,a) \Leftarrow q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - q(s,a)) \tag{8.56}$$

  The pseudo code of SARSA algorithm is listed below.

  Initialize: $q(s,a)\ \ \forall s \in S,\ \forall a \in A(s)$ arbitrarily ($q(s,a) = 0$ for terminal s),
  $\alpha \in (0,\ 1]$, $\epsilon > 0$

  loop (for each episode)

  Initialize state s

  while s is not terminal (for each step)

  take action a according to π based on $q(s,a)$, get reward r and next state s'

  $q(s,a) = q(s,a) + \alpha[r + \gamma \max_{a'} q(s',a)' - Q(s,a)]$

  $s = s'$

Here is the comparison of the MC and TD methods in terms of their pros and cons:

This is not a pro or con of either method; it's just a statement of what they do.

- The MC method estimates the state value $v_\pi(s_t)$, the expected return, by the true return G_t obtained at the end of the episode, i.e., the estimated value is updated once every episode;

  The TD method estimates $v_\pi(s_t)$ as the sum of the immediate reward r_{t+1} and the estimated state value at the next state $v_\pi(s')$, i.e., it is a bootstrap method, and the estimated value is updated at every step of every spisode.
- MC can only learn episodic (terminating) environments with complete episodes; while TD can learn continuing (non-terminating) environments of incomplete episodes.

- MC estimates $v_\pi(s_t)$ is based on sample returns G_t, and it is unbiased, while TD uses the bootstrap approach to find the TD target based on sample data that are not necessarily i.i.d., and it is more sensitive to the initial guess of the value functions, it is more biased.
- MC is based on the sample returns G_t affected by many random events (state transitions, actions, and rewards), and in particular the first-visit version of the MC method only makes use of the sample data from the first visit of a state, it does not use the available data efficiently and it has high variance, while on the other hand the TD method is based on only one random variable, the estimated return, and it makes more frequent and efficient use of the sample data, it has lower variance.
- In the MC method, especially the first-visit version of it, the estimated value functions is unbiased; on the other hand, as the TD method is based on the bootstrap strategy and relies more strongly on the initial guess of the value functions being estimated, it tends to be more biased.

This is a run-on sentence. Cutting it into several sentences would fix it: "...from the first visit of a state. It does not use the available data efficiently, and it has high variance. On the other hand, the TD method is based on only one random variable, the estimated return. It makes more frequent and efficient use of the sample data, and it has lower variance

Here is a summary of the dynamic programming (DP) method for model-based planning, and the Monte-Carlo (MC) and time difference (TD) methods for model-free control:

Is this a figure? If so, name the figure explicitly.

8.4.3 TD(λ) Algorithm

The MC and TD methods considered previously can be unified by the n-step TD(λ) method that spans a spectrum of which the MC and TD methods are two special cases at the opposite extremes.

Recall that both MC and TD algorithms updates iteratively the value functions being estimated and the policy being improved based on Eq. (8.49), but they estimate the target G_t in the equation differently. In an MC algorithm, the target is the actual return $G_t = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{T-t-1} r_T$, the sum of discounted rewards of all future steps upto the terminal state s_T, available only at the end of each episode. On the other hand, in a TD algorithm, the target is $G_t = r_{t+1} + \gamma v_\pi(s)t + 1)$ the sum of the immeidate reward r_{t+1} available at each step of the episode, and the discounted value of the next state, based on bootstrapping. We therefore see that an MC algorithm updates the value functions and policy once every episode, while a TD algorithm updates once every step in the spisode.

As a trade-off between the MC and TD methods, the n-step TD algorithm can be considered a generalization of the TD method, where the target in Eq. (8.49) is an *n-step return*, the sum of the discounted rewards in the n subsequent states and the discounted value function at the following state s_{t+n} with $1 \leq n < \infty$:

$$G_{t:t+n} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n v_\pi(s_{t+n}) \tag{8.57}$$

Especially, when

- $n = 1$: the 1-step return is the sum of the immediate reward and the estimated value of the next state, the same as the TD target in the TD method:

$$G_{t:t+1} = r_{t+1} + \gamma v(s_{t+1}) \tag{8.58}$$

- $n = T - t$: the n-step return is the sum of the discounted rewards from all future states to the terminal state at the end of the episode, i.e., it is the return r_t defined in Eq. (8.7), the same as in the MC method:

$$G_{t:t+n} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T + v_\pi(s_{t+n}) = G_t \tag{8.59}$$

  where the value function of the terminal state $v_\pi(s_{t+n}) = v_\pi(s_T) = 0$ is zero.
- $T - t < n < \infty$: All states s_n beyond the terminal state s_T with $n > T - t$ remain the same as the terminal state with value $v(s_T) = 0$ and return G_t, same as in the MC method.

We see that all these n-step returns form a spectrum with the MC and TD methods at the two ends.

Based on these n-step returns of different n values, the n-step TD algorithm can be further generalized to a more computationally advantageous and therefore more useful algorithm called TD(λ), by which the MC and TD algorithms are again unified as two special cases at the opposite ends of a spectrum.

We first define the λ-return G_t^λ as the weighted average of all n-step returns for $n = 1, 2, \cdots, \infty$:

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n} \tag{8.60}$$

where $0 \leq \lambda < 1$. Note that the weights decay exponentially, and they are normalized due to the coefficient $1 - \lambda$:

$$(1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} = (1 - \lambda) \sum_{n=0}^{\infty} \lambda^n = \frac{1 - \lambda}{1 - \lambda} = 1 \tag{8.61}$$

The λ-return G_t^λ defined above can be expressed in two summations, the sum of the first $T - t - 1$ n-step returns $G_{t:t+1}, \cdots, G_{t:t+T-1}$, and the sum of all subsequent n-step returns $G_{t:T} = \cdots = G_{t:\infty} = G_t$:

$$\begin{aligned} G_t^\lambda &= (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + (1 - \lambda) \sum_{n=T-t}^{\infty} \lambda^{n-1} G_t \\ &= (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t, \qquad (0 \leq t \leq T) \end{aligned} \tag{8.62}$$

where the coefficient of G_t in the second term is the sum of coefficients in the second summation:

$$(1 - \lambda) \sum_{n=T-t}^{\infty} \lambda^{n-1} = (1 - \lambda) \sum_{m=0}^{\infty} \lambda^m \lambda^{T-t-1} = \lambda^{T-t-1} \tag{8.63}$$

where $m = T - t$. We see that in Eq. (8.62) all n-step returns $G_{t:t+n}$ are weighted by exponentially decaying coefficient λ^{n-1}, except the true return G_t which is weighted by λ^{T-t-1}.

Again consider two special cases:

- $\lambda = 0$: all terms in Eq. (8.62) are zero, except the first one with $n = 1$:

$$G_t^{\lambda=0} = \lambda^{n-1} G_{t:t+n}\Big|_{n=1} = 0^0 \, G_{t:t+1} = r_{t+1} + \gamma v(s_{t+1}) \tag{8.64}$$

  This is simply the TD target, i.e., TD(0) is the TD method.
- $\lambda = 1$: the first summation is zero and

$$G_t^{\lambda=1} = G_t \tag{8.65}$$

  i.e., TD(1) is the MC method.

We therefore see that TD(λ) is a general algorithm of which the two special cases TD(0) and TD(1) are respectively the TD and MC methods.

In Eq. (8.62), G_t^λ is calulated as the weighted sum of all n-step returns $G_{t:t+n}$ upto the last one G_t available at the end of the episode. This summation can be truncated to include fewer terms before reaching the terminal state at the end of the episode:

$$G_{t:h}^\lambda = (1-\lambda) \sum_{n=1}^{h-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{h-t-1} G_{t:h}, \qquad (0 \le t < h \le T) \tag{8.66}$$

Specially when $h = T$, $G_{t:h}^\lambda = G_t^\lambda$ same as before, and when $h = t+1$, $G_{t:h}^\lambda = G_{t:t+1} = r_{t+1} + \gamma v(s_{t+1})$, same as the TD target.

Based on λ-return, the iterative update of the value function $v_\pi(s_t)$ in Eq. (8.52) can be modified to:

$$v_\pi(s_t) = v_\pi(s_t) + \alpha(G_t^\lambda - v_\pi(s_t)) = v_\pi(s_t) + \alpha\delta_t \tag{8.67}$$

where

$$\delta_t = G_t^\lambda - v_\pi(s_t) \tag{8.68}$$

This iteration is called the *forward view* of TD(λ), which is similar to the MC method, as they are based on either G_t or G_t^λ, available only at the end of each episode.

An alternative method is the *backward view* of TD(λ), which can be shown to be equivalent to the forward-view of TD(λ), but it is similar to the TD(0) method, as they are both based on the immediate reward r_{t+1} available at each time step of the episode, and therefore more computationally convenient.

Specifically, the backward view of TD(λ) is based the *eligibility trace* $e_t(s)$ for each of the states, which decays exponentially upon each state transition

$$e_t(s) = \gamma\lambda e_{t-1}(s) \qquad (\forall s \in S) \tag{8.69}$$

Same as before where? A lot of different variations of algorithms have been covered in this chapter; be specific.

"Same as X" is a phrasing you use a lot. It isn't quite grammatically right. It should either be "the same as," or preferably, "just as" or simply "as."

Define what the backward view of TD(lambda) is before explaining what it's equivalent/similar to.

An explanation of why these methods are called the forward and backward view would help to clarify the methods and their differences.

but is boosted by an increment 1 to become $e_t(s) = e_t(s)+1$ if $s = s_t$ is currently visited.

Now the iterative update of the estimated value function in Eq. (8.52) is modified so that *all* states, instead of only the one currently visited, are updated, but to different extents based on $e_t(s)$:

$$v_\pi(s) = v_\pi(s) + \alpha\delta_t e_t(s), \qquad \forall s \in S \tag{8.70}$$

where the TD error δ_t is the same as in TD(0), given in Eq. (8.53):

$$\delta_t = r_{t+1} + \gamma v_\pi(s_{t+1}) - v_\pi(s) \tag{8.71}$$

The eligibility trace $e_t(s)$ as defined above is motivated by the frequency and recency of the visites to each state. If a state s has been more frequently and recently visited compared to others, its $e_t(s)$ is greater than others and its value function $v_\pi(s)$ will be updated by a greater increment than others.

Here is the pseudo code for the backward view of the TD(λ) methd for policy evaluation:

Input: policy π to be evaluated
Initialize: $v(s) = 0 \ \ \forall s \in S$
loop (for each episode)
    $s = s_0$
    $e(s) = 0, \ \forall s \in S$
    while s is not terminal (for each step)
        take action a based on $\pi(a|s)$, get reward r and next state s'
        $\delta = r + \gamma v(s') - v(s)$
        $e(s) = e(s) + 1$
        for all s
            $v(s) = v(s) + \alpha\delta e(s)$
            $e(s) = \gamma\lambda e(s)$
        $s = s'$

Note that values at all states are updated, but to different extents depending on $e(s)$, different from the TD algorithm where only the value at the state currently visited is updated.

Again consider two special cases:

- $\lambda = 0$, $e(s) = 0$ for all states except the current s being visited, i.e., TD(λ) becomes TD(0), the same as the TD method.
- $\lambda = 1$, then $e(s)$ is scaled down by a factor $\gamma < 1$ for all states except the current s being visited, i.e., TD(λ) becomes TD(1), the same as the MC method. However, different from the MC method that updates the value function $v(s)$ at the end of each spisode, here $v(s)$ is still updated at every step of the episode due to the backward view of the method.

This makes it sound like TD(lambda) and vanilla TD both become TD(0) at lambda = 0, which is of course not the case.

In these cases, does e(s) take its value in addition to lambda, or because of lambda? As it stands, it's not clear.

This sentence has a dangling participle that makes it unclear that the two algorithms correspond to the SARSA and Q-learning algorithms. A rewrite to fix the problem might be: "Here are two algorithms based on eligibility traces that correspond to the SARSA and Q-learning algorithms based on TD(0)."

This backward view of the TD(λ) method can be applied to model-free control when the state value function is replaced by the action value function. Corresponding to the SARSA and Q-learning algorithms based on TD(0) in the previous section, here are the two algorithms based on eligibility traces:

- SARSA(λ) algorithm:
  Initialize: $q(s,a) = 0,\ \forall(s,a),\ \alpha \in (0,\ 1],\ \epsilon > 0$, denote ϵ-greedy policy $pi(a|s)$ by π
  loop (for each episode)
    $s = s_0$
    $e(s,a) = 0,\ \forall(s,a)$
    get action a according to π based on $q(s,a)$
    while s is not terminal (for each step)
      $e(s,a) = e(s,a) + 1$
      take action a, get reward r and next state s'
      get action a' according to π based on $q(s',a)$
      $\delta = r + \gamma q(s',a') - q(s,a)$
      for all (s,a)
        $q(s,a) = q(s,a) + \alpha\delta\, e(s,a)$
        $e(s,a) = \gamma\lambda e(s,a)$
      $s = s',\ a = a'$
- Q(λ) algorithem:
  Initialize: $q(s,a) = 0,\ \forall(s,a),\ \alpha \in (0,\ 1],\ \epsilon > 0$, denote ϵ-greedy policy $pi(a|s)$ by π
  loop (for each episode)
    $s = s_0$
    $e(s,a) = 0,\ \forall(s,a)$
    get action a according to π based on $q(s,a)$
    while s is not terminal (for each step)
      $e(s,a) = e(s,a) + 1$
      take action a, get reward r and next state s'
      get action a' according to π based on $q(s',a)$
      $a^* = \arg\max_b q(s',b)$
      $\delta = r + \gamma q(s',a^*) - q(s,a)$
      for all (s,a)
        $q(s,a) = q(s,a) + \alpha\delta\, e(s,a)$
        if $a' = a^*$ **then** $e(s,a) = \gamma\lambda e(s,a)$ **else** $e(s,a) = 0$
      $s = s',\ a = a'$

Note that Q(λ) algorithm is different from the SARSA(λ) algorithm in two ways. First, the action value is updated based on the greedy action a^*, instead of the a by policy π; second, if the ϵ-greedy policy π happens to choose a random non-greedy action $a' \neq a^*$ with probability ϵ to explore rather than exploiting a^*, the eligibility traces of all states are reset to zero. There are other different versions of the algorithm which do not reset these traces.

8.5 Value Function Approximation

All previously considered algorithms are based on either the state values function $v_\pi(s)$ for each state s or the action value function $q_\pi(s,a)$ for each state-action pair. As these function values can be considered as either a 1-D or 2-D table, the algorithms based on such function values are called tabular methods. However, when the number of states and the number of actions in each state are large (e.g., the game Go has 10^{170} states), the tabular methods are no longer suitable due to the unrealistic table sizes. In such a case, we can consider approximating the state value function $v(s)$ or action-value function $q(s,a)$ by a function $\hat{v}(s,\mathbf{w})$ or $\hat{q}(s,a,\mathbf{w})$, parameterized by a set of parameters as the components of vector $\mathbf{w}$. By doing so the state or action value functions can be represented more conveniently in terms of a much smaller number of parameters in $\mathbf{w}$.

This approach can be considered as a regression problem to fit a continuous function $\hat{v}(s,\mathbf{w})$ or $q(s,a,\mathbf{w})$ to a set of discrete and finite data points $v_\pi(s)$ or $q_\pi(s,a)$ obtained by sampling the MDP of the environment, so that all points in the state space, including those for unvisited states or state-action pairs never actually observed during the sampling process, can be represented.

Examples of such value function approximators include

- linear combination of all features
- multi-layer neural networks
- decision trees

Same as in regression problems, we desire to find the optimal parameter $\mathbf{w}$ for the modeling function $\hat{(v)}_\pi(s,\mathbf{w})$ so that the following objective function, the mean squared error between the approximated values and the sample values, is minimized:

$$J(\mathbf{w}) = \frac{1}{2}E_\pi[(v_\pi(s) - \hat{v}(s,\mathbf{w}))^2] \tag{8.72}$$

where the expectation E_π is with respect to all states visited while following some policy π. The gradient vector of $J(\mathbf{w})$ is

$$\begin{aligned}\nabla J(\mathbf{w}) &= \frac{d}{d\mathbf{w}}J(\mathbf{w}) = \frac{1}{2}E_\pi\left[\frac{d}{d\mathbf{w}}\left[(v_\pi(s) - \hat{v}(s,\mathbf{w}))^2\right]\right] \\ &= -E_\pi\left[(v_\pi(s) - \hat{v}(s,\mathbf{w}))\nabla\hat{v}(s,\mathbf{w})\right]\end{aligned} \tag{8.73}$$

and the optimal parameter vector $\mathbf{w}^*$ can be found iteratively by the gradient descent method

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \Delta\mathbf{w} = \mathbf{w}_n - \alpha\nabla J(\mathbf{w}) = \mathbf{w}_n + \alpha E_\pi\left[(v_\pi(s) - \hat{v}(s,\mathbf{w}))\nabla\hat{v}(s,\mathbf{w})\right] \tag{8.74}$$

where α is the step size or learning rate, and

$$\Delta\mathbf{w} = -\alpha\nabla\mathbf{J}(\mathbf{w}) = \alpha E_\pi\left[(v_\pi(s) - \hat{v}(s,\mathbf{w}))\nabla\hat{v}(s,\mathbf{w})\right] \tag{8.75}$$

If the method of stochastic gradient descent (SGD) is used, i.e., the parameter

$\mathbf{w}$ is updated based on only one data point at each iteration instead of the expectation of the data points, then E_π for the expectation can be dropped:

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \Delta\mathbf{w} = \mathbf{w}_n + \alpha(v_\pi(s) - \hat{v}(s, \mathbf{w}))\nabla\hat{v}(s, \mathbf{w})) \tag{8.76}$$

This sentence would be clearer if it were cut into several shorter sentences.

Here the true value function $v_\pi(s)$, the expected return, is unknown and needs to be estimated by the actual return G_t at each state $s = s_t$ based on the MC method, or $r_{t+1} + \gamma\hat{v}_\pi(s', \mathbf{w})$ in terms of the immediate reward r_{t+1} and the estimated value $\hat{v}_\pi(s', \mathbf{w})$ of the next state s' based on the TD method, by sampling the environment. Such an approach can be considered as a supervised learning based on labeled training data set: $\{(s_t, G_t), \forall t\}$ for the MC method, or $\{(s_t, r_{t+1}, \forall t\}$ for xthe TD method.

Is supervised learning just one way of viewing the approach? Or is the approach an example of supervised learning? If the latter, rephrase to: "Such an approach is an example of supervised learning..."

In the following, we will consider the special case where the state value $v_\pi(s)$ is approximated by a linear function

In the following what?

$$\hat{v}(s, \mathbf{w}) = \sum_{i=1}^{d} x_i w_i = \mathbf{w}^T\mathbf{x}(s) \tag{8.77}$$

based on the assumption that the value function of each state can be approximated as a linear combinatioin of a set of features in the feature vector $\mathbf{x}(s) = [x_1(s), \cdots, x_d(s)]^T$ for state s, weighted by the corresponding weights in the weight vector $\mathbf{w} = [w_1, \cdots, w_d]^T$ for all states.

Introducing another variable to represent N makes this less clear. If you're not going to use *d* for anything later, this could just be "an N-dimensional binary feature vector."

As a simple example, if we represent each of the $N = |S|$ states s_n by a $d = N$ dimensional binary feature vector $\mathbf{x}(s_n) = [x_1(s_n), \cdots, x_N(s_n)]^T$ of all zero components $x_i(s_n) = 0$ except the n-th one $x_n(s_n) = 1$, then the weight vector can be $\mathbf{w} = [w_1, \cdots, w_N]^T = [v_\pi(s_1), \cdots, v_\pi(s_N)]^T$, and we have $\hat{v}_\pi(s_n, \mathbf{w}) = \mathbf{w}^T\mathbf{x}(s_n) = v_\pi(s_n)$. In this case, each state s is represented by its value function $v_\pi(s)$ the same as in all previous discussions.

The objective function for the mean squared error of such a linear approximating function is

$$J(\mathbf{w}) = \frac{1}{2}E_\pi[(v_\pi(s) - \hat{v}(s, \mathbf{w}))^2] = \frac{1}{2}E_\pi[(v_\pi(s) - \mathbf{w}^T\mathbf{x}(s))^2] \tag{8.78}$$

and its gradient is

$$\nabla J(\mathbf{w}) = \frac{d}{d\mathbf{w}}J(\mathbf{w}) = -E_\pi\left[(v_\pi(s) - \mathbf{w}^T\mathbf{x}(s))\mathbf{x}(s)\right] \tag{8.79}$$

If all data points in terms of state s represented by $\mathbf{x}(s)$ and the corresponding return $G(s)$ as a sample of the true value $v_\pi(s)$ have already been collected, then the value function approximation can be carried out as an off-line algorithm in a batch manner, the same as in the linear least squares linear regression problem discussed in a previous chapter, and the optimal weight vector that minimizes the squared error

To make the transition into the equation clear, include w* above: "...and the optimal weight vector w* that minimizes the squared error."

$$\mathbf{w}^* = \arg\max_{\mathbf{x}} \sum_s [G(s) - \hat{v}_\pi(s, \mathbf{w})]^2 \tag{8.80}$$

can be found by the same pseudo inverse method

$$\mathbf{w}^* = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{G} \tag{8.81}$$

where $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_N]$ is a matrix containing all N samples for the states and $\mathbf{G} = [G_1, \cdots, G_N]^T$ is a vector containing the corresponding returns.

However, as reinforcement learning is typically an online problem, the parameter $\mathbf{w}$ for the approximator needs to be modified in real-time whenever a new piece of data becomes available during the sampling of the environment. In this case, we can find the gradient descent method, based on the gradient vector of $J(\mathbf{w})$:

$$\begin{aligned}\nabla J(\mathbf{w}) &= \frac{d}{d\mathbf{w}} J(\mathbf{w}) = -E_\pi(v_\pi(s) - \hat{v}(s, \mathbf{w}))\nabla\hat{v}(s, \mathbf{w}) \\ &= -E_\pi(v_\pi(s) - \mathbf{w}^T\mathbf{x}(s))\mathbf{x}(s)\end{aligned} \tag{8.82}$$

and the optimal parameter $\mathbf{w}^*$ can be found iteratively

$$\begin{aligned}\mathbf{w}_{n+1} &= \mathbf{w}_n + \Delta\mathbf{w} = \mathbf{w}_n + \alpha E_\pi\left[v_\pi(s) - \hat{v}(s, \mathbf{w})\right]\nabla\hat{v}(s, \mathbf{w}) \\ &= \mathbf{w}_n + \alpha E_\pi\left[v_\pi(s) - \mathbf{w}^T\mathbf{x}(s)\right]\mathbf{x}(s)\end{aligned} \tag{8.83}$$

where $\Delta\mathbf{w}$ is the increment in each iteration:

$$\Delta\mathbf{w} = \alpha E_\pi\left[v_\pi(s) - \hat{v}(s, \mathbf{w})\right]\nabla\hat{v}(s, \mathbf{w}) = \alpha E_\pi\left[v_\pi(s) - \mathbf{w}_n^T\mathbf{x}(s)\right]\mathbf{x}(s) \tag{8.84}$$

If stochastic gradient descent is used, then $\mathbf{w}$ is updated whenever a new sample data point $\mathbf{x}(s_t)$ is available at a time step t, then the expectation E_π in the equations above can be dropped.

We note that the expectation of the estimated $\mathbf{w}$ found by this SGD method is the same as that obtained by full GD method, so this iteration will convergence to the global minimum of the objective function $J(\mathbf{w})$ in Eq. (8.78) as it is a quadratic function with only one minimum which is global.

Specifically, the state value function is unknown and can be estimated in several different ways, similar to the corresponding algorithms discussed before.

- **MC method**

  The value function $v_\pi(s)$ is estimated by the actual return G_t for each state s_t visited in an episode, obtained only at the end of each episode while sampling the environment:

  $$\Delta\mathbf{w} = \alpha[G_t - \hat{v}(s_t, \mathbf{w})]\nabla\hat{v}(s_t, \mathbf{w}) = \alpha[G_t - \mathbf{w}^T\mathbf{x}(s_t)]\mathbf{x}(s_t) \tag{8.85}$$

  Here is the pseudo code for the algorithm based on the MC method, similar to that for value evaluation algorithms discussed previously:

  Initialize $\mathbf{w} = \mathbf{0}$

  loop (for each episode)

  run current episode to the end to get $G_t,\ t = 1, \cdots, T$

  for $t = 1, \cdots, T$

  if s_t is visited the first time

$$\mathbf{w} = \mathbf{w} + \alpha[G_t - \mathbf{w}^T\mathbf{x}(s_t)]\mathbf{x}(s_t)$$

end if

end for

end loop

- **TD(0) method**

  The value function $v_\pi(s)$ is estimated by the TD target, the sum of the immediate reward r_{t+1} and the approximated value of the next state s_{t+1}, at each time step of each episode while sampling the environment:

$$\Delta\mathbf{w} = \alpha[r_{t+1} + \gamma\mathbf{w}^T\mathbf{x}(s_{t+1}) - \mathbf{w}^T\mathbf{x}(s_t)]\mathbf{x}(s_t) \tag{8.86}$$

  Here is the pseudo code for the algorithm based on the TD method, similar to that for value evaluation algorithms discussed previously:

  Initialize $\mathbf{w} = \mathbf{0}$

  loop (for each episode)

    for each time step in current episode

      take action $a = \pi(s)$, get reward r and next state s'

      $\mathbf{w} = \mathbf{w} + \alpha[r + \gamma\mathbf{w}^T\mathbf{x}(s') - \mathbf{w}^T\mathbf{x}(s)]\mathbf{x}(s)$

    end for

  end loop

- **TD(λ) method**

  The value function $v_\pi(s)$ is estimated based on λ-return G_t^λ given in Eq. (8.62). As discussed before, TD(λ) has two flavors:

  – Forward view: same as MC method, except the true return G_t is replaced by λ-return G_t^λ available only at the end of each spisode:

$$\Delta\mathbf{w} = \alpha[G_t^\lambda - \mathbf{w}^T\mathbf{x}(s_t)]\mathbf{x}(s_t) \tag{8.87}$$

  – Backward view: similar to TD(0), this is based on the immediate reward r_{t+1} available at every step of each episode:

$$\delta_t = \alpha[r_{t+1} + \gamma\mathbf{w}^T\mathbf{x}(s_{t+1}) - \mathbf{w}^T\mathbf{x}(s_t)]\mathbf{x}(s_t) \tag{8.88}$$

$$e_t = \gamma\lambda e_{t-1} + \mathbf{x}(s_t), \qquad \Delta\mathbf{w} = \alpha\delta_t e_t \tag{8.89}$$

Given a policy π for an MDP, we define a probability distribution $d(s)$ over all states visited according to π, satisfying

$$\sum_s d(s) = 1 \tag{8.90}$$

and the following balance equation:

$$d(s') = \sum_s \sum_a \pi(a|s)p(s'|s,a)d(s) \tag{8.91}$$

As both $d(s')$ and $d(s)$ represent the same distribution, they must be identical.

Given the probability distribution $d(s)$, the mean squared error of the value function approximation for a policy π can be expressed as

$$MSE(\mathbf{w}) = \sum_s d(s)[v_\pi(s) - \hat{v}_\pi(s, \mathbf{w})]^2 \tag{8.92}$$

It can be shown that the MC method for the linear value function approximation will converge to the optimal weight vector $\mathbf{w}_{MC}$ that minimizes the mean squared error above:

$$MSE(\mathbf{w}_{MC}) = \min_{\mathbf{w}} \sum_s d(s)[v_\pi(s) - \hat{v}_\pi(s, \mathbf{w})]^2 \tag{8.93}$$

This should be capitalized according to the rules for titles (all words are capitalized, except for small ones like "a," "the," and "and").

8.6 Control based on Function Approximation

The control algorithms based on approximated value functions also follow the general method of the general policy iteration, as illustrated below. We note that this is similar to the algorithms for model-free control illustrated in Fig. **??**, but with the action-value function $q_\pi(s, a)$ is replaced by the parameter $\mathbf{w}$ of the approximation action function $\hat{q}(s, a, \mathbf{w})$. In particular, for a linear function, we have:

If this refers to a figure, cite the figure by number.

$$\hat{q}(s, a, \mathbf{w}) = \sum_n w_n x(s, a) = \mathbf{w}^T \mathbf{x}_n(s, a) \tag{8.94}$$

where $\mathbf{x}(s, a)$ is the feature vector for the state-action pair (s, a). We need to find the optimal parameter $\mathbf{w}$ that minimizes the objective function, the mean square error of the approximation:

$$J(\mathbf{w}) = \frac{1}{2} E_\pi[(q_\pi(s, a) - \hat{q}(s, a, \mathbf{w}))^2] = \frac{1}{2} E_\pi[(q_\pi(s, a) - \mathbf{w}^T \mathbf{x}(s, a))^2] \tag{8.95}$$

with gradient vector:

$$\begin{aligned} \nabla J(\mathbf{w}) &= \frac{d}{d\mathbf{w}} J(\mathbf{w}) = -E_\pi \left[q_\pi(s, a) - \hat{q}(s, a, \mathbf{w})\right] \nabla \hat{q}(s, a, \mathbf{w}) \\ &= -E_\pi \left[(q_\pi(s, a) - \mathbf{w}^T \mathbf{x}(s, a))\mathbf{x}(s, a)\right] \end{aligned} \tag{8.96}$$

If the stochastic gradient descent method is used based on a single sample of the action value $q_\pi(s, a)$, instead of its expectation, then E_π can be dropped, and the optimal weight vector $\mathbf{w}^*$ that minimizes $J(\mathbf{w})$ in Eq. (8.95) can be learned iteratively:

$$\begin{aligned} \mathbf{w}_{t+1} &= \mathbf{w}_t + \Delta \mathbf{w} = \mathbf{w}_t - \alpha \nabla \mathbf{J}(\mathbf{w}) \\ &= \mathbf{w}_t + \alpha \left[(q_\pi(s_t, a_t) - \hat{q}_\pi(s_t, a_t, \mathbf{w}))\nabla \hat{q}_\pi(s_t, a_t, \mathbf{w})\right] \\ &= \mathbf{w}_t + \alpha \left[(q_\pi(s_t, a_t) - \mathbf{w}_t^T \mathbf{x}(s_t, a_t))\mathbf{x}(s_t, a_t)\right] \end{aligned} \tag{8.97}$$

where $\Delta \mathbf{w}$ is the increment of the update:

$$\begin{aligned} \Delta \mathbf{w} = -\alpha \nabla \mathbf{J}(\mathbf{w}) &= \alpha[q_\pi(s_t, a_t) - \hat{q}_\pi(s_t, a_t, \mathbf{w})] \nabla \hat{q}_\pi(s_t, a_t, \mathbf{w}) \\ &= \alpha[q_\pi(s_t, a_t) - \mathbf{w}_t^T \mathbf{x}(s_t, a_t)]\mathbf{x}(s_t, a_t) \end{aligned} \tag{8.98}$$

As the true Q-value $q_\pi(s,a)$ in the expression is unknown, it needs to be estimated by some target depending on the specific methods used:

- **MC method:**
  The true $q_\pi(s,a)$ is replaced by the sample return G_t as the target, obtained at the end of each episode:

$$\Delta\mathbf{w} = \alpha[G_t - \hat{q}(s_t,a_t,\mathbf{w})]\nabla\hat{q}(s_t,a_t,\mathbf{w}) = \alpha[G_t - \mathbf{w}_t^T\mathbf{x}(s_t,a_t)]\mathbf{x}(s_t,a_t) \tag{8.99}$$

- **TD(0) method:**
  The action value function $q_\pi(s,a)$ is replaced by the TD target, the sum of the immediate reward, available at each step of each spisode, and the approximated action value of the next state s_{t+1}:
  – SARSA (on-policy):

$$\begin{aligned}\Delta\mathbf{w} &= \alpha[r_{t+1} + \gamma\hat{q}(s_{t+1},a_{t+1},\mathbf{w}) - \hat{q}(s_t,a_t,\mathbf{w})]\nabla\hat{q}(s_t,a_t,\mathbf{w}) \\ &= \alpha[r_{t+1} + \gamma\mathbf{w}^T\mathbf{x}(s_{t+1},a_{t+1}) - \mathbf{w}_t^T\mathbf{x}(s_t,a_t)]\mathbf{x}(s_t,a_t)\end{aligned} \tag{8.100}$$

  Following Eq. (8.71), the TD error is defined as:

$$\delta_t = r_{t+1} + \gamma\hat{q}(s_{t+1},a_{t+1},\mathbf{w}) - \hat{q}(s_t,a_t,\mathbf{w}) \tag{8.101}$$

  then we get

$$\Delta\mathbf{w} = \alpha\delta_t\nabla\hat{q}(s_t,a_t,\mathbf{w}) \tag{8.102}$$

  – Q-learning (off-policy):

$$\begin{aligned}\Delta\mathbf{w} &= \alpha[r_{t+1} + \gamma\max_{a'}\hat{q}(s_{t+1},a',\mathbf{w}) - \hat{q}(s_t,a_t,\mathbf{w})]\nabla\hat{q}(s_t,a_t,\mathbf{w}) \\ &= \alpha[r_{t+1} + \gamma\max_{a'}\mathbf{w}^T\mathbf{x}(s_{t+1},a') - \mathbf{w}_t^T\mathbf{x}(s_t,a_t)]\mathbf{x}(s,a)\end{aligned} \tag{8.103}$$

- **TD(λ) method:**
  In the forward-view version of the TD(λ) method the action function $q_\pi(s,a)$ is approximated by λ-return G_t^λ as the target, available only at the end of each episode:

$$\Delta\mathbf{w} = \alpha[G_t^\lambda - \mathbf{w}_t^T\mathbf{x}(s_t,a_t)]\mathbf{x}(s_t,a_t) \tag{8.104}$$

  The backward-view version of the TD(λ) method based on eligibility traces is more advantageous in both space and temporal complexity as well as learning efficiency.

  We first define an eligibility trace vector which is set to zero at the beginning of the episode, but then decays

$$e_t = \gamma\lambda e_{t-1} + \nabla_w\hat{q}(s_t,a_t,\mathbf{w}) \tag{8.105}$$

$$\Delta\mathbf{w} = \alpha\delta_t e_t \tag{8.106}$$

This is not enough context to understand Eq 8.107, or what it has to do with Eq 8.70.

and Eq. (8.70)

$$v_\pi(s) = v_\pi(s) + \alpha\delta_t e_t(s), \qquad \forall s \in S \tag{8.107}$$

I assume this is a typographical error, and should be removed.

/////

Recall the TD error for the backward view of the TD(λ) method first given in Eq. (8.71):

I think you mean that the TD error for the TD(lambda) method is analogous to delta_t here, but it would be clearer if you said that outright.

$$\delta_t = (r_{t+1} + \gamma v_\pi(s_{t+1}) - v_\pi(s)) \tag{8.108}$$

In summary, here are the conceptual (not necessarily algorithmic) steps for the general model-free control based on approximated action-value function:

- Learn parameter $\mathbf{w}$ as in Eq. (8.97),
- Get the Q-values as in Eq. (8.94)
- Obtain the policy by ϵ-greedy approach as in Eq. (8.41)

These steps are also illustrated below:

$$\text{Training of } \mathbf{w} \Longrightarrow Q - value \Longrightarrow \text{Policy } \pi \tag{8.109}$$

8.7 Deep Q-learning

I think this section is a work in progress, so I won't comment on it substantively.

This is some odd spacing. Are there a few stray tabs around the word "or"?

Previously we approximated the state or action value functions by a linear function $\hat{v}_\pi(s, \mathbf{w}) = \mathbf{w}^T\mathbf{x}(s)$, or $\hat{q}_\pi(s, a, \mathbf{w}) = \mathbf{w}^T\mathbf{x}(s, a)$ parameterized by the weight vector $\mathbf{w}$ based on a set of features in $\mathbf{x}$. However, these features need to hand picked or designed based on the specific problem to solve.

Network!

8.8 Policy Gradient Methods

All RL algorithms previously considered are based on either state or action-value function and the policy is indectly derived from them by greedy or ϵ-greedy method. However, as the ultimate goal of an RL problem is to find the optimal policy that maximizes the return, it make sense to consider this as an optimization problem to directly find the optimal policy based on an objective function representing the total return to be maximized, by the gradient ascent method, as discussed in the following.

Previously we approximate the state value function $v_\pi(s)$ and action value function $q_\pi(s, a)$ by some parameterized functions $\hat{v}_\pi(s, \mathbf{w})$ and $\hat{q}_\pi(s, a, \mathbf{w})$ respectively, of which the parameter $\mathbf{w}$ can be obtained by sampling the environment, as a supervised learning process. Now we construct a model of a stochastic policy as a parameterized function:

$$\pi(a|s, \theta) = P(a|s, \theta) \tag{8.110}$$

where vector $\theta = [\theta_1, \cdots, \theta_d]^T$ represents some d parameters of the model. As the dimensionality d is typically smaller than the number of states and actions, such a parameterized policy model is suitable in cases where the numbers of states and actions are large or even continuous.

As a specific example, the policy model can be based on the soft-max function:

$$\pi(a|s,\theta) = \frac{e^{h(s,a,\theta)}}{\sum_b e^{h(s,b,\theta)}} \tag{8.111}$$

satisfying $\sum_a \pi(a|s,\theta) = 1$. Here the summation is over all possible actions, and $h(s,a,\theta)$ is the *preference* of action a in state s, which can be a parameterized function such as a simple linear function $h(s,a,\theta) = \theta^T \mathbf{x}(s,a)$, or a neural network with weights represented by θ, the same as how the value functions are approxmiated in Section 8.5. According to this policy, an action with higher preference $h(s,a,\theta)$ will have a higher probability to be chosen.

This sentence would be clearer if cut into two or three shorter sentences.

Different from how we find the parameter $\mathbf{w}$ of $\hat{q}(s,a,\mathbf{w})$ by minimizing $J(\mathbf{w})$, the mean squared error between the value function $q_\pi(s,a)$ and its model $\hat{q}_\pi(s,a,\mathbf{w})$, based on gradient descent, here we find the parameter θ of $\pi(a|s,\theta)$ by maximizing $J(\theta)$ representing the value function, the expected return, under the policy, based on gradient ascent. Such methods are therefore called *policy gradient methods*.

The value-function based methods considered previously and the policy-based methods considered here are summarized below, together with the *actor-critic method*, as the combination of the two:

- Value-based: the policy is derived by the greedy or ϵ-greedy method based on the value function learned during sampling the environment as a supervised learning process.
- Policy-based: the policy is directly learned without explicitly value functioin estimation.
- Actor-Critic: parameters for both the value function model and policy model are learned simultaneously.

This sentence doesn't quite hang together grammatically. It would probably be better if cut into two: "Policy-based RL has better convergence policies, but it may stick at a local optimum. It's effective in high-dimensional or continuous action space, and can learn stochastic policies, but it has high variance."

Better convergence properties than what?

Advantages of policy-based RL includes better convergence properties, but may stuck at local optimum, effective in high-dimensional or continuous action space can learn stochastic policies, but have high variance.

How good a policy is may be measured by different objective functions related to the values or rewards associated with the policy being evaluated, depending on the environment of the specific problem:

- In episodic environments with finite horizon, we can use the value, the expected return (sum of discounted future rewards), of the start state s_0:

$$J_1(\theta) = v_{\pi_\theta}(s_0) \tag{8.112}$$

- In continuing (online) environments with infinite horizon, we can use the

average value

$$J_{avV}(\theta) = \sum_s d_{\pi_\theta}(s) v_{\pi_\theta}(s) \tag{8.113}$$

where $d_{\pi_\theta}(s)$ is the stationary distribution of all states under policy $\pi(a|s,\theta)$.

We can find the optimal paramter θ^* for the policy model $\pi(a|s,\theta)$ by solving the maximization problem:

$$\theta^* = \arg\max_\theta J(\theta) \tag{8.114}$$

based on the gradient of $J(\theta)$:

$$\nabla_\theta J(\theta) = \frac{d}{d\theta} J(\theta) = \left[\frac{\partial J}{\partial \theta_1}, \cdots, \frac{\partial J}{\partial \theta_n}\right]^T \tag{8.115}$$

by the iterative gradient ascent method:

$$\theta_{t+1} = \theta_t + \Delta\theta = \theta_t + \alpha \nabla_\theta J(\theta) \tag{8.116}$$

Here we use t for the index of the iteration based on the assumption that the a new sample point is available at every time step of an episode while sampling the environment following policy $\pi(a|s,\theta)$.

While it is conceptually straight forward to see how gradient ascent method can be used to find the optimal parameter θ^* that maximizes $J(\theta)$, it is not easy to actually find its gradient $\nabla J(\theta)$, which depends on not only what actions to take at the states directly determined by the policy $\pi(a|s,\theta)$, but also how the states are distributed under the policy in an unknown environment. This challenge is addressed by the following *policy gradient theorem*. The proof of the theorem below leads to an expression of the gradient $\nabla_\theta J(\theta)$, which can be used to find the optimal parameter θ^* iteratively by the stochastic gradient ascend method.

Proof of policy gradient theorem:

$$
\begin{aligned}
\nabla_\theta v_{\pi_\theta}(s) &= \nabla_\theta \left(\sum_a \pi(a|s,\theta) q_{\pi_\theta}(s,a) \right) \\
&\stackrel{1}{=} \sum_a \left[\nabla_\theta \pi(a|s,\theta)\ q_{\pi_\theta}(s,a) + \pi(a|s,\theta)\ \nabla_\theta q_{\pi_\theta}(s,a) \right] \\
&\stackrel{2}{=} \sum_a \left[\nabla_\theta \pi(a|s,\theta)\ q_{\pi_\theta}(s,a) + \pi(a|s,\theta) \nabla_\theta \sum_{s'} \sum_r P(s',r|s,a)(r + v_{\pi_\theta}(s')) \right] \\
&\stackrel{3}{=} \sum_a \left[\nabla_\theta \pi(a|s,\theta)\ q_{\pi_\theta}(s,a) + \pi(a|s,\theta) \sum_{s'} \sum_r P(s',r|s,a) \nabla_\theta v_{\pi_\theta}(s') \right] \\
&\stackrel{4}{=} \sum_a \left[\nabla_\theta \pi(a|s,\theta)\ q_{\pi_\theta}(s,a) + \pi(a|s,\theta) \sum_{s'} P(s'|s,a) \nabla_\theta v_{\pi_\theta}(s') \right] \\
&\stackrel{5}{=} \sum_a \nabla_\theta \pi(a|s,\theta)\ q_{\pi_\theta}(s,a) + \sum_a \sum_{s'} \pi(a|s,\theta) P(s'|s,a) \nabla_\theta v_{\pi_\theta}(s') \\
&\stackrel{6}{=} \phi(s) + \sum_{s'} \sum_a P(s'|s,a) \pi(a|s,\theta) \nabla_\theta v_{\pi_\theta}(s') \qquad (8.117)
\end{aligned}
$$

where

1. product rule of derivative: $(uv)' = u'v + uv'$
2. $q_{\pi_\theta}(s,a) = \sum_{s'} \sum_r P(s',r|s,a)(r + v_{\pi_\theta}(s'))$
3. r is not a function of θ
4. $\sum_r P(s',r|s,a) = P(s'|s,a)$ in Eq. (8.17)
5. $\pi(a|s,\theta)$ is indpendent of s' and moved inside summation over s'
6. We have defined

You're missing a number here.

$$
\phi(s) = \sum_a \nabla_\theta \pi(a|s,\theta)\ q_{\pi_\theta}(s,a) \qquad (8.118)
$$

We note that Eq. (refPolicyGradientThm1) is a recursion by which $\nabla_\theta v_{\pi_\theta}(s)$ is expressed as a function in terms $\nabla_\theta v_{\pi_\theta}(s')$.

We further define $Pr_\pi(s \to x, k)$ as the probability of transitioning from state s to a state x after k steps following $\pi(a|s,\theta)$:

$$
s \stackrel{\pi(a|s,\theta)}{\longrightarrow} s' \stackrel{\pi(a|s',\theta)}{\longrightarrow} s'' \stackrel{\pi(a|s'',\theta)}{\longrightarrow} \cdots \stackrel{\pi(a|s^{(k-1)},\theta)}{\longrightarrow} s^{(k)} = x \qquad (8.119)
$$

with the following properties:

- $$Pr_\pi(s \to s, 0) = 1 \qquad (8.120)$$
- $$Pr_\pi(s \to s', 1) = \sum_a \pi(a|s,\theta) P(s'|s,a) \qquad (8.121)$$

- $$\sum_{s'} Pr_\pi(s \to s', 1) = 1 \tag{8.122}$$

- $$Pr_\pi(s \to s'', 2) = Pr_\pi(s \to s', 1) Pr_\pi(s' \to s'', 1) \tag{8.123}$$

Continuing Eq. (8.117) we keep rolling out the recursion of $\nabla_\theta v_{\pi_\theta}(s')$ and get:

$$\begin{aligned}
\nabla_\theta v_{\pi_\theta}(s) &= \phi(s) + \sum_{s'} \sum_{a} \pi(a|s,\theta) P(s'|s,a) \nabla_\theta v_{\pi_\theta}(s') \\
&\overset{1}{=} \phi(s) + \sum_{s'} Pr_\pi(s \to s', 1) \nabla_\theta v_{\pi_\theta}(s') \\
&\overset{2}{=} \phi(s) + \sum_{s'} Pr_\pi(s \to s', 1) \left[\phi(s') + \sum_{s''} Pr_\pi(s' \to s'', 1) \nabla_\theta v_{\pi_\theta}(s'') \right] \\
&\overset{3}{=} \phi(s) + \sum_{s'} Pr_\pi(s \to s', 1) \phi(s') + \sum_{s''} Pr_\pi(s \to s', 1) Pr_\pi(s' \to s'', 1) \nabla_\theta v_{\pi_\theta}(s'') \\
&\overset{4}{=} \phi(s) + \sum_{s'} Pr_\pi(s \to s', 1) \phi(s') + \sum_{s''} Pr_\pi(s \to s'', 2) \nabla_\theta v_{\pi_\theta}(s'') \\
&\overset{5}{=} \phi(s) + \sum_{s'} Pr_\pi(s \to s', 1) \phi(s') + \sum_{s''} Pr_\pi(s \to s'', 2) \phi(s'') + \sum_{s'''} Pr_\pi(s \to s''', 2) \nabla_\theta v_{\pi_\theta}(s''') \\
&= \cdots\cdots \overset{6}{=} \sum_{x \in S} \sum_{k=0}^{\infty} Pr_\pi(s \to x, k) \phi(x)
\end{aligned} \tag{8.124}$$

where

1. Eq. (8.121)
2. unroll $\nabla_\theta v_{\pi_\theta}(s')$
3. Eq. (8.122)
4. Eq. (8.123)
5. unroll $\nabla_\theta v_{\pi_\theta}(s'')$
6. keep unrolling recursively to infinity, and Eq. (8.120)

"And Eq. (8.120)" is unclear. To clarify, explain how Eq. (8.120) is involved.

Now the gradient of the objective $J(\theta) = v_{\pi_\theta}(s_0)$ can be written as

$$\begin{aligned}
\nabla_\theta J(\theta) = \nabla_\theta v_\pi(s_0) &= \sum_{s \in S} \sum_{k=0}^{\infty} Pr_\pi(s_0 \to s, k)\ \phi(s) \\
&= \sum_{s \in S} \eta(s)\ \phi(s) = \left(\sum_{s'} \eta(s') \right) \sum_{s \in S} \frac{\eta(s)}{\sum_{s'} \eta(s')}\ \phi(s) \\
&\propto \sum_{s \in S} \frac{\eta(s)}{\sum_{s'} \eta(s')}\ \phi(s) = \sum_{s \in S} \mu_\pi(s) \sum_{a} \nabla_\theta \pi(a|s,\theta)\ q_{\pi_\theta}(s,a)
\end{aligned} \tag{8.125}$$

Here the proportionality is introduced due to the dropping of $\sum_{s'} \eta(s')$ as a

constant independent of state s, and we also defined

$$\eta(s) = \sum_{k=0}^{\infty} Pr_{\pi}(s_0 \rightarrow s, k), \tag{8.126}$$

as the sum of all probabilities for visiting state s from the start state s_0, and

$$\mu_{\pi}(s) = \frac{\eta(s)}{\sum_{s'} \eta(s')} \tag{8.127}$$

as a normalized verion of $\eta(s)$ representing the probability distribution of visiting state s while following policy π. Eq. (8.125) can be further written as

$$\begin{aligned} \nabla_{\theta} J(\theta) = \nabla_{\theta} v_{\pi}(s_0) &\propto \sum_{s \in S} \mu_{\pi}(s) \sum_{a} \pi(a|s,\theta) \frac{\nabla_{\theta} \pi(a|s,\theta)}{\pi(a|s,\theta)} q_{\pi_{\theta}}(s,a) \\ &= \sum_{s \in S} \mu_{\pi}(s) \sum_{a} \pi(a|s,\theta) \nabla_{\theta} \ln \pi(a|s,\theta) q_{\pi_{\theta}}(s,a) \\ &= E_{\pi_{\theta}} [\nabla_{\theta} \ln \pi(a|s,\theta) q_{\pi_{\theta}}(s,a)] \end{aligned} \tag{8.128}$$

where $E_{\pi_{\theta}}$ denotes the expectation over all actions in each state s weighted by $\pi(a|s,\theta)$ and all states $s \in S$ weighted by $\mu_{\pi}(s)$.

Q.E.D.

We see that the gradient $\nabla_{\theta} J(\theta)$ is now expressed as the expectation of the action value function $q_{\pi}(s,a)$, weighted by the gradient of the logarithm of the corresponding policy $\pi(a|s,\theta)$. Now Eq. (8.116) can be further written as

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta} J(\theta) = \theta_t + \alpha E_{\pi_{\theta}} [q_{\pi_{\theta}}(s_t, a_t) \nabla_{\theta} \ln \pi(a_t|s_t,\theta)] \tag{8.129}$$

We further note that Eq. (8.125) still hold if an arbitrary bias term $b(s)$ independent of action a is included:

$$\begin{aligned} \nabla_{\theta} J(\theta) &\propto \sum_{s \in S} \mu_{\pi}(s) \sum_{a} \nabla_{\theta} \pi(a|s,\theta) (q_{\pi_{\theta}}(s,a)) \\ &= \sum_{s \in S} \mu_{\pi}(s) \sum_{a} \nabla_{\theta} \pi(a|s,\theta) (q_{\pi_{\theta}}(s,a) + b(s)) \end{aligned} \tag{8.130}$$

as

$$\sum_{a} \nabla_{\theta} \pi(a|s,\theta) b(s) = b(s) \nabla_{\theta} \sum_{a} \pi(a|s,\theta) = b(s) \nabla_{\theta} 1 = 0 \tag{8.131}$$

Now Eq. (8.116) can also be written as

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta} J(\theta) = \theta_t + \alpha E_{\pi_{\theta}} [(q_{\pi_{\theta}}(s_t, a_t) + b(s_t)) \nabla_{\theta} \ln \pi(a_t|s_t,\theta)] \tag{8.132}$$

The expectation $E_{\pi_{\theta}}$ in Eqs. (8.129) and (8.132) can be dropped if the method of stochastic gradient ascent is used, as in all algorithms below, where each iterative step is based on only one sample data point instead of its expectation.

Specifically, for a soft-max policy model as given in Eq. (8.111), we have

$$\begin{aligned}\nabla_\theta \ln \pi(a|s,\theta) &= \nabla_\theta \ln\left(\frac{e^{\theta^T \mathbf{x}(s,a)}}{\sum_b e^{\theta^T \mathbf{x}(s,b)}}\right) = \nabla_\theta \left(\theta^T \mathbf{x}(s,a) - \ln \sum_b e^{\theta^T \mathbf{x}(s,b)}\right) \\ &= \mathbf{x}(s,a) - \frac{\nabla_\theta \sum_b e^{\theta^T \mathbf{x}(s,b)}}{\sum_b e^{\theta^T \mathbf{x}(s,b)}} = \mathbf{x}(s,a) - \frac{\sum_b e^{\theta^T \mathbf{x}(s,b)} \mathbf{x}(s,b)}{\sum_b e^{\theta^T \mathbf{x}(s,b)}} \\ &= \mathbf{x}(s,a) - \sum_b \frac{e^{\theta^T \mathbf{x}(s,b)}}{\sum_b e^{\theta^T \mathbf{x}(s,b)}} \mathbf{x}(s,b) \\ &= \mathbf{x}(s,a) - \sum_b \pi(b|s,\theta)\mathbf{x}(s,b) \end{aligned} \tag{8.133}$$

We list set of popular policy-based algorithms below, based on either the MC or TD methods, generally used in previous algorithms.

- **REINFORCE (MC) policy gradient**

  In this algorithm, the action value function $q_{\pi_\theta}(s_t, a_t)$ in Eq. (8.129) is replaced by the return G_t obtained at the end of each episode while sampling the environment. Now the equation becomes:

$$\theta_{t+1} = \theta_t + \alpha \gamma^t G_t \, \nabla_\theta \ln \pi(a|s,\theta) \tag{8.134}$$

  Note that the discount factor γ^t is included as the expression for $\nabla_\theta J(\theta)$ in Eq. (8.128) assumed $\gamma = 1$ for simplicity.

  Here is the pseudo code for the algorithm:

  Initialize $\pi(a|s,\theta$

  loop (for each episode)

  At the end of episode, get G_t for each state visited

  for $t = 1, \cdots, T$

  $\theta = \theta + \alpha \gamma^t G_t \nabla_\theta \ln \pi(a_t|s_t,\theta)$

  end loop

- **Actor-Critic (TD) policy gradient**

  As its name suggests, this algorithm is based on two approximation function models, the first for the policy $\pi(a|s,\theta)$ parameterized by θ, the *actor*, same as in the REINFORCE algorithm above, and the second for the value function $\hat{v}_\pi(s,\mathbf{w})$ parameterized by $\mathbf{w}$, the *critic*, same as in Eq. (8.97).

  Specifically, in Eq. (8.132), the action value function $q_{\pi_\theta}(s_t, a_t)$ is replaced by its bootstrapping expression $r_{t+1} + \gamma \hat{v}_\pi(s', \mathbf{w})$, and the bias term $b(s)$ is replaced by the approximated value function $\hat{v}(s_t, \mathbf{w})$. Now both parameters $\mathbf{w}$ and θ can be found iteratively at every step of an episode while sampling the environment (the TD method) by stochastic gradient method with the expectation E_π dropped:

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha_w \left[(r_{t+1} + \gamma \hat{v}_\pi(s',\mathbf{w}) - \hat{v}_\pi(s,\mathbf{w}))\, \nabla_w \hat{v}_\pi(s,\mathbf{w})\right] \\ \theta_{t+1} &= \theta_t + \alpha_\theta \left[(r_{t+1} + \gamma \hat{v}_\pi(s',\mathbf{w}) - \hat{v}_\pi(s,\mathbf{w}))\, \nabla_\theta \ln \pi(a|s,\theta)\right] \end{aligned} \tag{8.135}$$

  Here is the pseudo code for the algorithm:

initialize model parameters θ and $\mathbf{w}$
initialize step sizes α_θ and α_w
loop (for each episode)
initialize $s = s_0$, $t = 0$
while s is not terminal (for each step)
take action a following $\pi(a|s, \theta)$, find reward r and next state s'
find TD error: $\delta = r + \gamma\hat{v}(s', \mathbf{w}) - \hat{v}(s, \mathbf{w})$
$\mathbf{w} = \mathbf{w} + \alpha_w \; \delta \; \nabla\hat{v}(s, \mathbf{w})$
$\theta = \theta + \alpha_\theta \gamma^t \delta \; \nabla \ln \pi(a|s, \theta)$
$s = s'$
$t = t + 1$
end while
end loop

- **Backward view of TD(λ) policy gradient**

Here is the pseudo code for the algorithm:
initialize model parameters θ and $\mathbf{w}$
initialize step sizes α_θ and α_w
initialize trace-decay rates λ_θ and λ_w
loop (for each episode)
initialize $s = s_0$, $t = 0$
initialize $\mathbf{z}_\theta = \mathbf{0}$, $\mathbf{z}_w = \mathbf{0}$
while s is not terminal (for each step)
take action a following $\pi(a|s, \theta)$, find reward r and next state s'
find TD error: $\delta = r + \gamma\hat{v}(s', \mathbf{w}) - \hat{v}(s, \mathbf{w})$
$\mathbf{z}_\theta = \gamma\lambda_\theta \mathbf{z}_\theta + \nabla\gamma^t \ln \pi(a|s, \theta)$
$\mathbf{z}_w = \gamma\lambda_w \mathbf{z}_w + \gamma^t \nabla\hat{v}(s, \mathbf{w})$
$\mathbf{w} = \mathbf{w} + \alpha_w \; \delta \; \nabla\hat{v}(s, \mathbf{w})$
$\theta = \theta + \alpha_\theta \, \delta \; \nabla \ln \pi(a|s, \theta)$
$s = s'$
$t = t + 1$
end while
end loop

Following the similar steps, the policy gradient theorem for environment with continuous state and action spaces can be also proven:

$$\nabla_\theta J(\theta) = \int_s \mu_\pi(s) \int_a \nabla_\theta \pi(a|s, \theta) \; q_{\pi_\theta}(s, a) da \; ds \tag{8.136}$$