Lecture 1: Introduction to Reinforcement Learning

Tianpei Xie

Dec 27th., 2017

Contents

1 Reinforcement Learning

The study of machine learning can be divided into three categories:

1. **Supervised learning**. Learning to ***predict*** and ***generalize***. In other words, the task of learning is to infer a mapping between covariates (data) and responses (target) so that the error/cost is minimized. Each example is a description of situation (sample) together with a specification (label) of the correct action the system should take to that action. Supervised learning is an **error-correction** process. It is an one-step prediction. Human label is used as **instructive feedbacks**.

2. **Unsupervised learning**. Learning to *represent* and *discover* the hidden structure of data. A proper representation of data facilitates knowledge discovery and improves the performance of prediction. It also helps in visualization, storage and communication.

3. **Reinforcement learning**. Learning from ***interaction***. As compared to above approaches, reinforcement learning studies *goal-directed learning from interaction*. The term 'learning' means learning to map *situations* to *actions* so as to maximize the reward. Also it is often unrealistic to obtain all examples of desired behavior that are both correct and representative of all situations in which the agents have to act. Reinforcement learning is a **trial-and-error** process and it is a multi-step prediction. It cares about future rewards in multiple steps. On the other hand, reinforcement learning only optimize the future rewards, where the rewards are used as **evaluative feedbacks**.

In reinforcement learning, the learner is not told which actions to take, but instead must discover which actions yield the most reward by *trying* them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics – **trial-and-error search** and **delayed reward** – are the two most important distinguishing features of reinforcement learning. Supervised learning algorthim enforces each step to be as instructed whereas the reinforcement learning agent only cares if the end goal is reached, and does not care the process. In supervised learning, we care about the **target** predicted by the optimized learning model. In reinforcement learning, we care about the **process** of getting it. RL is not to be confused with ***online learning*** [Cesa-Bianchi and Lugosi, 2006, Shalev-Shwartz et al., 2012], a method of machine learning in which data becomes available in a *sequential* order and is used to update the best predictor for future data at each step. Both supervised learning and RL method can be online learning.

The basic theory behind reinforcement learning is dynamic systems theory, specifically, the optimal control of partially-known Markov decision processes. The idea is to capture the most important aspects of the real problem facing a *learning agent* interacting over time with its *environmen*t to achieve a goal.

One of the challenges that arise in reinforcement learning, and not in other kinds of learning, is the *trade-off* between ***exploration*** and ***exploitation***. To obtain a lot of reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing reward. But to discover such actions, it has to try actions that it has not selected before. The agent has to *exploit* what it has already experienced in order to obtain reward, but it also has to *explore* in order to make better action selections in the future. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task. The agent must try a variety of actions and *progressively* favor those that appear to be best. On a stochastic task, each action

must be tried many times to gain a reliable estimate of its expected reward. The **exploration-exploitation dilemma** has been intensively studied by mathematicians for many decades, yet remains unresolved.

- **exploitation**: the agent maximizes the rewards based on past experience
- **exploration**: the agent interacts with environments for better actions in future

Another key feature of reinforcement learning is that it explicitly considers the *whole* problem of a goal-directed agent interacting with an uncertain environment. It not only considers the learning sub-problem but also how it fits into a larger picture of decision making and *planning*. A reinforcement learning agent is a complete, *model-interactive*, *goal-seeking* object and usually operates facing significant uncertainties about the environment. When reinforcement learning involves planning, it has to address the interplay between planning and real-time action selection, as well as the question of how environment models are acquired and improved. When reinforcement learning involves supervised learning, it does so for specific reasons that determine which capabilities are critical and which are not.

Terminologies in reinforcement learning:

- **Agent**: an object that is able to *sense* the state of its environment to some extent and must be able to *take actions* that *affect* the state. The agent also must have a goal or goals relating to the state of the environment.
- **Environment**: the surrounding field of the agent that is related to the problem to be solved. Environment also provides feedback regarding actions of an agent and defines the goal of the agents. The situation/information of environment that is sensible to the agent is referred as *state* of environment. As it interacts with agents, the state of environment changes.
- **Policy**: defines the learning agent's way of behaving at a given time. Roughly speaking, a policy is a mapping from *perceived states* of the environment to *actions* to be taken when in those states. In some cases the policy may be a simple function or lookup table, whereas in others it may involve extensive computation such as a search process. The policy is the ***core*** of a reinforcement learning agent in the sense that it alone is sufficient to determine behavior. In general, policies may be stochastic.
- A ***reward*** *signal* defines the goal in a reinforcement learning problem. On each time step, the environment sends to the reinforcement learning agent a single number called the *reward*. The agent's sole objective is to maximize the ***total reward*** it receives ***over the long run***. The reward signal thus defines what are the good and bad events for the agent. The reward signal is the primary basis for altering the policy; if an action selected by the policy is followed by low reward, then the policy may be changed to select some other action in that situation in the future. In general, reward signals may be *stochastic* functions of the state of the environment and the actions taken.
- **Returns** at t is the cumulative rewards starting at time t. Return is the measure of rewards in long run.
- ***Value function***: specifies what is good ***in the long run***, whereas the reward signal specifies what is good in an ***immediate*** sense. Roughly speaking, the value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state.

  Rewards are in a sense **primary**, whereas values, as predictions of rewards, are **secondary**.

Without rewards there could be no values, and the only purpose of estimating values is to achieve more reward. Nevertheless, s *values* with which we are most concerned when making and evaluating decisions. Action choices are made based on *value judgments*. The optimal actions should bring about states of highest values, not highest reward. Unfortunately, it is much harder to determine values than it is to determine rewards. Rewards are basically given directly by the environment, but values must be estimated and re-estimated from the sequences of observations an agent makes over its entire lifetime. The *central role* of ***value estimation*** is arguably the most important thing we have learned about reinforcement learning over the last few decades.

- ***Model***: describes and mimics the behavior of the environment, or more generally, that allows inferences to be made about how the environment will behave. For example, given a state and action, the model might predict the resultant next state and next reward. Models are used for *planning*, by which we mean any way of deciding on a course of action by considering possible future situations before they are actually experienced.

  Methods for solving reinforcement learning problems that use models and *planning* are called ***model-based*** methods, as opposed to simpler ***model-free*** methods that are explicitly *trial-and-error* learners – viewed as almost the *opposite* of planning.

- ***exploration-exploitation tradeoff***: the fundamental tradeoff for all reinforcement learning algorithms. Both exploration and exploitation require the agent to spend time and resources, while fail to balance them will result in agent's poor learning performance.

  - ***exploitation***: the agent need to ***optimize*** its policy and evaluation estimation performance based on accumulated experience by far.

  - ***exploration***: the agent need to ***explore*** new actions and visit previously unvisited states in order to acquire new experience and search potential better policy, even if choosing these actions are not based on an optimal policy.

  The agent's goal is to exploit but without exploration it cannot gain enough feedbacks and experience about the environment, which would cause the agent to stuck in its **comfort zoon**. On the other hand, pure exploration without exploitation would cause the agent to not learn anything.

There are approaches that do not belong to reinforcement learning. For instances, various *evolutionary methods* including genetic algorithms and genetic programming, simulated annealing and other optimization methods have been used to search optimal policy without appealing to value functions. If the *space of policies is sufficiently small*, or can be structured so that good policies are common or easy to find – or if a lot of time is *available for the search* – then evolutionary methods can be effective. In addition, evolutionary methods have advantages on problems in which the learning agent *cannot sense the complete state* of its environment. On the other hand, evolutionary methods do not interact with the environment but to evaluate the 'life-time' behavior of many non-learning agents. Without the details of individual behavioral interactions, evolutionary methods ignores many useful structures that benefits the reinforcement learning. Evolutionary methods also do not use the fact that the policy they are searching for is a function from states to actions; they do not notice which states an individual passes through during its lifetime, or which actions it selects.

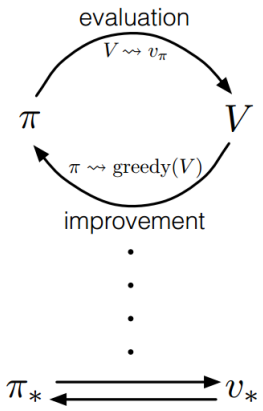


Figure 1: Generalized Policy iteration

2 Summary of methods learned

We summarize our learning in the book [Sutton and Barto, 2018]. Please comes back after finishing learning the main materials.

2.1 From Multi-armed Bandit to MDP

The reinforcement learning methods we are discussing are based on Markov Decision Process. Simpler than MDP, we have the Multi-armed Bandit problem.

- **Multi-Armed Bandit problem (MAB)** (*lecture 2*): in the classical Multi-armed bandit problem, there is only one state or it is **stationary**. Therefore, we **only care about the *short-term gain***, i.e. the rewards. The goal is to select the action $a \in \{1, \dots, k\}$ with highested **expected rewards**:

  $$q_*(a) := \mathbb{E}\left[R_t \,|\, A_t = a\right].$$

  The solution for MAB is to compute the average of sample rewards. We use the incremental updates

  $$Q_{n+1} := Q_n + \alpha_n(R_n - Q_n) \quad \text{where } \alpha_n := \frac{1}{n} \tag{1}$$
  $$\Rightarrow \textbf{new estimate} = \textbf{old estimate} + \text{stepsize} \cdot (\text{target} - \text{old estimate})$$

  The optimal action is selected via greedy or ϵ-greedy algorithm based on the estimated expected reward for each action.

- **The Markov Decision Process (MDP)** (*lecture 3*): A MDP is a discrete-time stochastic control process. It is defined by the tuple $(T, \mathcal{S}, \mathcal{A}, \mathcal{R}, P(S_{t+1}, R_{t+1}|S_t, A_t))$, where T defines the set of decison epochs, $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, which is determined by the state selected, $\mathcal{R}$ is the reward set. The conditional distribution $P(S_{t+1}, R_{t+1}|S_t, A_t))$ follows the **Markov property**, i.e. the reward and next state is determined **completely** by its immediate predecessor state and action. $p(s', r|s, a)$ defines the dynamic of the process. $\pi : \mathcal{S} \to \mathcal{A}$ is the *policy*, which is what we cared about the most.

The process modeled by MDP is inherently dynamic and **non-stationary**. The objective of the agent is to find ***optimal policy*** π that *maximizes* the **long-term gain**, i.e. the cumulative future rewards, or **expected returns**. To evalute the performance of a policy π, we introduce the value function

$$v_\pi(s) := \mathbb{E}_\pi \left[\sum_{\tau=0}^{T-t-1} \gamma^\tau R_{\tau+t+1} \middle| S_t = s \right];$$
$$q_\pi(s,a) := \mathbb{E}_\pi \left[\sum_{\tau=0}^{T-t-1} \gamma^\tau R_{\tau+t+1} \middle| S_t = s, A_t = a \right].$$

As we can see, the reinforcement learning based on MDP has two major steps:

- **Policy evaluation (Prediction)**: The task of policy evaluation is to **estimate *value function*** $\hat{v}(s)$ and $\hat{q}(s,a)$ based on rewards collected in the sequence $(S_t, A_t, R_{t+1}, S_{t+1}, \ldots)$ as well as previous estimates (**boostrapping**)
- **Policy improvement (Control)**: The task of control is to find an ***optimal policy*** π_* given the value function from policy evaluation.

The **generalized policy iteration (GPI)** (*lecture 4*) defines an iterative procedure that loops between policy evaluation and policy improvement. Through this procedure, the agent is able to adapt and improve its policy by interactions with environment.

The theory of reinforcment learning overlaps with the ***Stochastic Control theory*** [Puterman, 2014] or ***Optimal Control*** **theory**, where *action* is referred as *control* and the *environment* is the ***dynamic system*** under control. The difference is that stochastic control theory focus more on model-based learning.

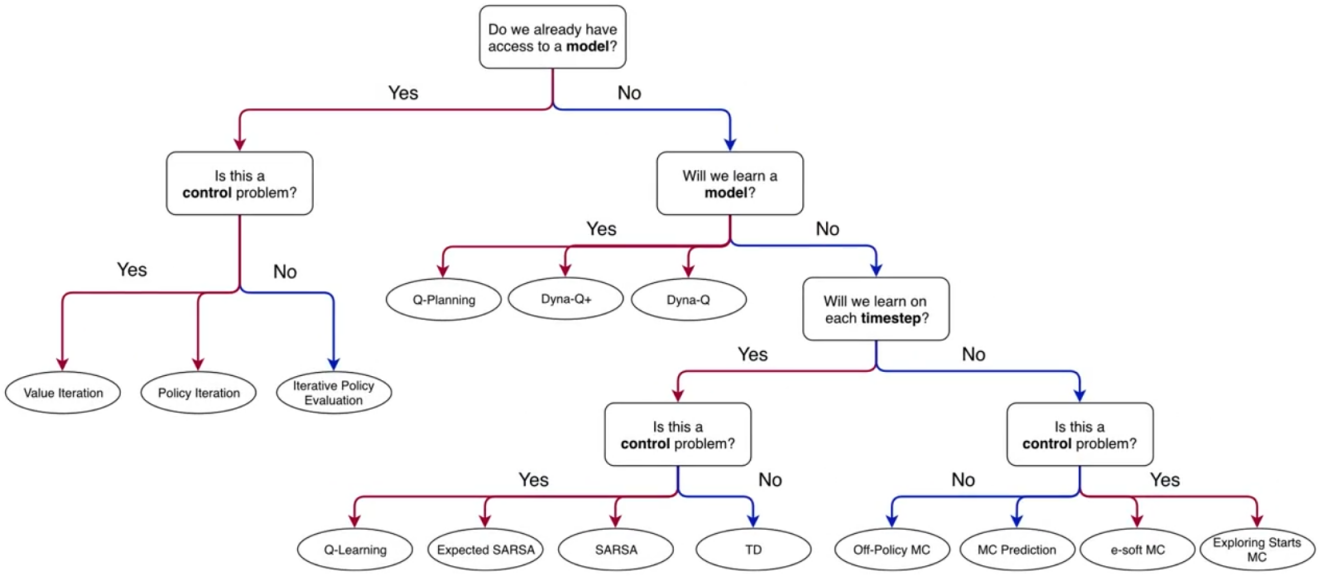
2.2 Tabular methods vs. Function approximation

The most important concepts in reinforcement learning is the **value function**. A value function is a special **data structure** that **store** the **long-term gain locally** and is *accessible* at each time step of learning. Through value function, we formulate our problem into a series of **overlapping subproblems with common structure**. At each timestep, given state/state-action pair, the **learning objective of each subproblem** is defined using value function. As the learning process continues, the **value function** will **approach to the ultimate learning goal** of our agents. Therefore, an agent that exploit its policy choice over **local** estimate of value function is able to reach the **global optimal** policy in asympototic sense. With value function, we can design a recusive algorithm conveniently with linear time.

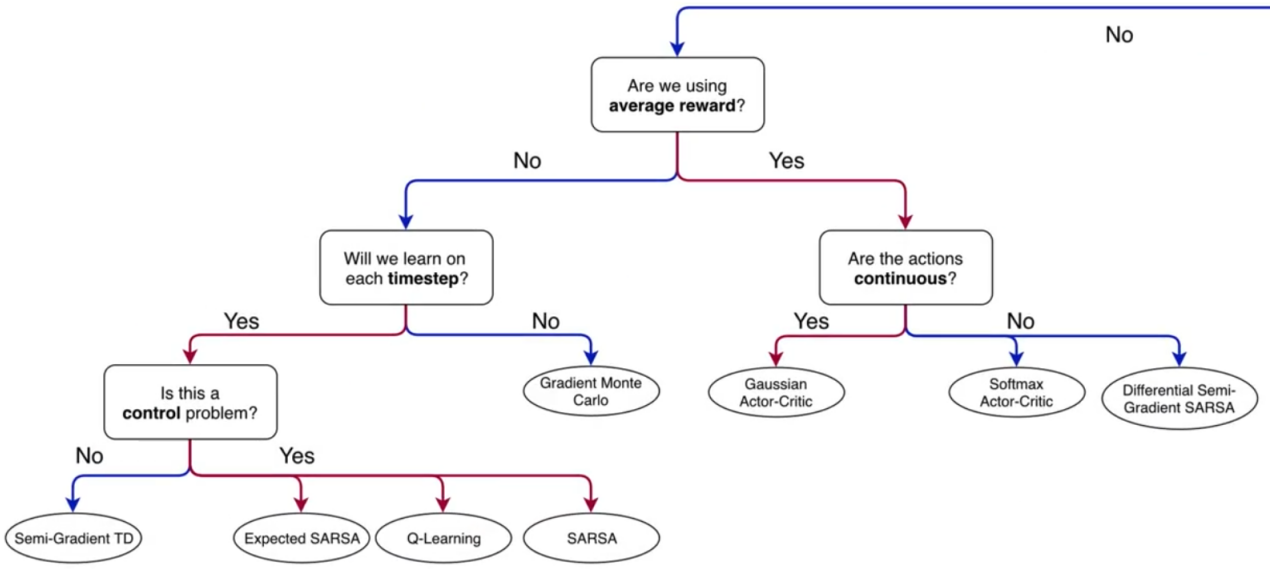
Depending on how to represent the value function, the algorithms fall into two categories. See Figure 2 for a summary of all of these methods.

2.2.1 Tabular methods

Tabular methods(lecture 4-7): The simplest way to represent and store the value function is via a table.



(a)



(b)

Figure 2: Tabular methods (a) and Function approximation methods (b) diagram

A **tabular representation** of value function allows us to **maintain** and **update** the *value* of each state or state-action pair **independently**. This type of presentation maximize the **discrimination** of the value function, but results in zero **generalization**. Generalization of value function means that updating value of one state would affect the value of other states.

Tabular representation is suitable if $|\mathcal{S}|$ and $|\mathcal{A}|$ is relatively small. For many real world problem, a tabular representation for value of all actions and states are not feasible.

We split our algorithm for tabular methods into algorithms for prediction and control:

- **Control** (lecture 4): The policy improvement strategy is **fixed**. According to **Policy improvement theorem** for tabular methods, it is guaranteed that a greedy policy that maximize the action-value function is able to improve upon existing policy by increasing the value v_π.

  Theorem 2.1 *(**Policy improvement theorem**) Let π and π' be any pair of deterministic policies such that, for all $s \in \mathcal{S}$,*

  $$q_\pi(s, \pi'(s)) \geq v_\pi(s). \tag{2}$$

  *Then the policy π' must be **as good as, or better than**, π. That is, it must obtain greater or equal expected return from all states $s \in \mathcal{S}$:*

  $$v_{\pi'}(s) \geq v_\pi(s). \tag{3}$$

  Moreover, if there is strict inequality of (2) at any state, then there must be strict inequality of (3) at that state.

- **Prediction**(lecture 4-7): depending on our **knowledge** on the **environment/model**, we split again it into two classes of algorithms:
  - **Model-based methods (Planning)** (lecture 4): when we have full access to model *dynamic* $p(s', r|s, a)$ for all s, a, r, s'. In this case, we have **Bellman equation** and **Bellman optimality equation** (lecture 3) for v_π, q_π, v_*, q_*. The **dynamic programming (DP)** algorithm make ***expected updates*** based on Bellman equation.

$$\begin{aligned} v_\pi(s) &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma v_\pi(s') \right], \quad \forall\ s \in \mathcal{S} && (4)\\ &= \mathbb{E}\left[R_{t+1} + \gamma\, v_\pi(S_{t+1}) | S_t = s \right] && (5)\\ q_\pi(s, a) &= \sum_{s', r} p(s', r|s, a) \left[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a') \right] && (6)\\ &= \mathbb{E}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a \right] && (7) \end{aligned}$$

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a \right] && (8)\\ &= \max_{a \in \mathcal{A}(s)} \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma v_*(s') \right] \quad s \in \mathcal{S} && (9)\\ q_*(s, a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a' \in \mathcal{A}(s)} q_*(S_{t+1}, a') \Big| S_t = s, A_t = a \right] && (10)\\ &= \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma \max_{a' \in \mathcal{A}(s)} q_*(s', a') \right] \quad \forall\ s \in \mathcal{S},\ a \in \mathcal{A}(s) && (11) \end{aligned}$$

There are **policy iteration methods** or **value iteration methods** based on whether or not we need to complete multiple *sweeps* all states at each iteration or just one sweep of policy evaluation and one sweep for policy improvement, respectively.

– **Model-free methods (Learning)** (lecture 5-7): normally, the agent has little knowlege on the environment. The only way to learn is via **interation with environment**, i.e. at given observed state, generate a sample action and then collect sample rewards and observe sample next states. Compared to model-based learning method, model-free algorithms requires much less knowledge and much less resources. This is achieved at the expense of losing some efficiency via additional exploration and accumulation.

  The model-free algorithms is also called **sample-based methods**. It incrementally update the value function estimate using old estimate, and some **error term**. The error term is the main difference for these algorithms. For instance,

  * **Monte Carlo (MC) methods (lecture 5)** use directly the **sample return** in the error term. In order to obtain the sample return, one has to wait until the episode ends to update the value estimate and thus to update the policy. The advantage is that the value estimator from Monte Carlo methods is **unbiased** but with **high variance** and thus **slow learning**. MC methods can be applied to **off-policy** setting using **importance sampling**, where the ratio between target policy and behavior policy is used as importance weights. The ***sample updates*** for MC are

$$V(S_t) \leftarrow V(S_t) + \alpha_t \left(G_t - V(S_t)\right)$$
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t \left[G_t - Q(S_t, A_t)\right],$$

    where G_t is the sample return accumulated starting at $t+1$ until the end of episode.

  * **Temporal Difference (TD) learning (lecture 6)** use boostrapping to estimate the return based on the immediate reward plus the estimate of value at immediate next state. TD learning can update the value estimator at each time step, thus it is much **faster** in learning with much **lower variance**. On the other hand, it introduces additional **bias** since the target is affected by the quality of value estimator at its successor state.

$$V(S_t) \leftarrow V(S_t) + \alpha_t \left[\hat{G}_t - V(S_t)\right].$$
$$\textbf{TD(0)} \quad \hat{G}_t = R_{t+1} + \gamma V(S_{t+1}).$$

    For action-value function (control), the ***sample updates*** are

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t \left[\hat{G}_t - Q(S_t, A_t)\right].$$
$$\textbf{Sarsa (on-policy)} \quad \hat{G}_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$$
$$\textbf{Q-Learning (off-policy)} \quad \hat{G}_t = R_{t+1} + \gamma \max_{a' \in \mathcal{A}(S_{t+1})} Q(S_{t+1}, a')$$
$$\textbf{Expected Sarsa (off-policy)} \quad \hat{G}_t = R_{t+1} + \gamma \sum_{a'} \pi(a'|S_{t+1}) Q(S_{t+1}, a').$$

2.2.2 Function Approximation

Function approximation (lecture 8-10): Tabular representation of value function or policy distribution is limited to small state space $|\mathcal{S}|$ and small action space $|\mathcal{A}|$. Base on the **curse of dimensionality**, as the dimension of state increases, the size of state space increases exponentially. For very large state space or action space, the only feasible solution is via function approximation.

The performance of function approximation is measured via its ability for **generalization** and **discrimination**:

- ***Generalization***, which means updating value of one state would affect the value of other states. That is, the change *generalizes* from that state to affect the values of many other states. Aggregate all states would easily achieve high generalization but with no discrimination.
- ***Discriminization***, which means the ability to make the value of two states different. If value of each states are represented independently, such as tabular methods, the discrimination is maximized but there is no generalization.

Compared to tabular representation, function approximation **encodes** the *value function* or the *policy distribution* into a set of **parameters**, i.e. $\hat{v}(s) \rightarrow \hat{v}(\boldsymbol{s}, \boldsymbol{w})$ or $\hat{q}(s, a) \rightarrow \hat{q}(\boldsymbol{s}, a, \boldsymbol{w})$. The **dimension of parameters** is much less than the size of table when values of all states/action-states are stored. It is thus an **efficient representation** compared to tables. **Tabular representation** can be seen as **linear approximation** of value function using **one-hot encoding**.

On the other side, however, the loss of discrimination power would lead to the **loss of *policy improvement theorem***. It is no longer true that if we change the policy to improve the discounted value of one state then we are guaranteed to have improved the overall policy in any useful sense. That is, we **no longer** have a **fixed strategy** to obtain optimal policy given the value function. The good news is that by using **parameterized policy distribution**, we can learn optimal policy based on ***policy gradient theorem*** using **policy gradient methods**.

This leads to the change in each step of the Generalized Policy Iteration.

- **Prediction** (lecture 8-9): Compared to Tabular methods, the prediction task is very **similar** for function approximation methods. The main difference is that we do not directly update the value at each state/state-action pair. Instead, we incrementally **updates the parameter** $\boldsymbol{w}$ or the value function.

  The task of prediction with *function approximation* is essentially a ***supervised learning*** **problem**. Given $(S_t, A_t, \hat{G}_t)$, the objective is to find parameterized function $q_{\boldsymbol{w}} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ that minimize the **mean squared value error** (VE). The **target** $\hat{G}_t$ could be the *sample returns* (MC) or the *TD target* (TD). The optimal paramter $\boldsymbol{w}_*$ can be found via online algorithms such as stochastic gradient desent or semi-gradient methods:

  - **Gradient Monte Carlo algorithm** via **stochastic gradient descent (SGD)**:

$$\begin{aligned} \boldsymbol{w}_{t+1} &\leftarrow \boldsymbol{w}_t + \alpha \left[\hat{G}_t - \hat{v}(S_t, \boldsymbol{w}_t)\right] \nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w}_t) \\ \boldsymbol{w}_{t+1} &\leftarrow \boldsymbol{w}_t + \alpha \left[\hat{G}_t - \hat{q}(S_t, A_t, \boldsymbol{w}_t)\right] \nabla_{\boldsymbol{w}} \hat{q}(S_t, A_t, \boldsymbol{w}_t) \end{aligned} \tag{12}$$

    where $\hat{G}_t = G_t$ is the sample return obtained at end of each episodes.

– **Temporal difference learning** via **semi-gradient methods**:

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \alpha\delta_t \nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w}_t) \tag{13}$$

where $\delta_t = R_{t+1} + \gamma\hat{v}(S_{t+1}, \boldsymbol{w}_t) - \hat{v}(S_t, \boldsymbol{w}_t)$ is the **TD error**.

For action-value function, we have

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \alpha\, \delta_t\, \nabla_{\boldsymbol{w}} \hat{q}(S_t, A_t, \boldsymbol{w}_t)$$

$$\textbf{SARSA} \quad \delta_t = [R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w}_t) - \hat{q}(S_t, A_t, \boldsymbol{w}_t)] \tag{14}$$

$$\textbf{Q-Learning} \quad \delta_t := \left[R_{t+1} + \gamma \max_{a'} \hat{q}(S_{t+1}, a', \boldsymbol{w}_t) - \hat{q}(S_t, A_t, \boldsymbol{w}_t)\right] \tag{15}$$

$$\textbf{Expected Sarsa} \quad \delta_t = \left[R_{t+1} + \gamma \sum_{a'} \pi(a'|S_{t+1})\hat{q}(S_{t+1}, a', \boldsymbol{w}_t) - \hat{q}(S_t, A_t, \boldsymbol{w}_t)\right] \tag{16}$$

Note that the problem we are solving in policy evalution/prediction has violated many **assumptions** in supervised learning:

– While the supervised learning assume the samples are ***independent*** *identically distributed*, this is **not true** for policy evalution task with function approximation. Since successive states $(S_t, A_t, G_t)_{t\geq\tau}$ follows a MDP, which are naturally correlated.

– Most supervised learning methods are not designed for **online learning** setting, while the reinforcement learning required online learning ability as rewards are obtained sequentially.

– By using **boostrapping** in the **target** like TD learning, we make the target depend on the approximated function evaluted at successor state. This introduce additonal **bias** as well as the ***positive feedback loop*** caused by the **error** of value function approximation as well as **asympototic dependencies**. This makes the target function dynamic and changing during the learning process.

Since the assumption of supervised learning methods are violated, many theoretical **convergence guarantees do not hold**. In fact, with boostrapping in target, many complex models such as neural networks are likely to **diverge**. This is especially true in off-policy learning such as Q-learning, which is called **Off-policy Divergence**.

We also learned two special classes of function approximation: **linear methods** and **artificial neural network**.

– **Linear methods** is the simplest function approximation. The key is to design a good feature mapping for state.

$$\hat{v}(s, \boldsymbol{w}_t) = \sum_k w_{k,t}\phi_k(s) = \langle \boldsymbol{w}_t\,,\, \boldsymbol{\phi}(s)\rangle \tag{17}$$

where $\boldsymbol{\phi}$ is the **feature mapping**. Popular choice of feature mapping functions includes **functional basis** such as Fourier Basis, RBF, RKHS. We also generalize the one-hot encoding to have **coarse coding**, **tile coding** etc.

– **Artificial neural network (ANN)**. **Deep learning architectures** vary a lot for different applications and state definition. For instance, if the state is image, then

Convolutional Neural Network (CNN) is preferred. Deep learning can be applied to Q-learning to have the ***Deep Q-Network (DQN)*** [François-Lavet et al., 2018] algorithm. Check other deep RL models in [François-Lavet et al., 2018]

ANN is highly **non-linear**, **non-convex** model, which makes it very *sensitive* to the change of target. Thus, boostrapping in target is very likely to cause the learning algorithm diverge.

- **Control** (lecture 10): As discussed above, policy improvement theorem does not hold for function approximation. However, by using **parameterized policy distribution** $\pi \to \pi(a|\boldsymbol{s}, \boldsymbol{\theta})$, we can find another way to find optimal policy: **policy gradient method**.

  In fact, we have the **policy gradient theorem** which gives an exact formula for how performance is affected by the policy parameter that does not involve derivatives of the state distribution. Like how policy improvement theorem leads to a fixed strategy in Tabular methods, this policy gradient theorem provides a theoretical foundation for all policy gradient methods.

  Theorem 2.2 *(**Policy gradient theorem**) For both the episodic case and continuing case under ergodic MDP, the objective functions $\mathcal{R}(\boldsymbol{\theta})$ are defined as in* (**??**) *and* (**??**) *respectively. Then*

$$\nabla_{\boldsymbol{\theta}} \mathcal{R}(\boldsymbol{\theta}) \propto \sum_s \mu_{\pi(\boldsymbol{\theta})}(s) \sum_a \nabla_{\boldsymbol{\theta}} \pi(a|\boldsymbol{s}, \boldsymbol{\theta}) q_\pi(\boldsymbol{s}, a) \tag{18}$$

$$= \mathbb{E}_{\boldsymbol{s} \sim \boldsymbol{\mu}_{\pi(\boldsymbol{\theta})}} \left[\sum_a \nabla_{\boldsymbol{\theta}} \pi(a|\boldsymbol{s}, \boldsymbol{\theta}) q_\pi(\boldsymbol{s}, a) \right] \tag{19}$$

$$= \mathbb{E}_{\boldsymbol{s} \sim \boldsymbol{\mu}_{\pi(\boldsymbol{\theta})}} \left[\mathbb{E}_{a \sim \pi(a|\boldsymbol{s}, \boldsymbol{\theta})} \left[\nabla_{\boldsymbol{\theta}} \log \pi(a|\boldsymbol{s}, \boldsymbol{\theta}) q_\pi(\boldsymbol{s}, a) \right] \right], \tag{20}$$

  where $\boldsymbol{\mu}_\pi$ is the limiting state distribution under π, $\mu(s) = \lim_{t \to \infty} Pr\{S_t = s | S_0, \pi\}$, (or on-policy distribution under policy π). In particular, the gradient of objective does not depend on the gradient of state distribution $\nabla \boldsymbol{\mu}$. For ergodic MDP with average reward objective, the equation (18) *is exact. For episodic task, the constant of proportionality is the average length of an episode.*

  We have the following **policy gradient methods**:

  - **REINFORCE** (**stochastic gradient ascent** with **Monte Carlo** methods):

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha \, G_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t, \boldsymbol{\theta}), \tag{21}$$

    where G_t is the sample return obtained at the end of each episode.

  - **REINFORCE** with **baseline**: adding baseline function (function of state) helps to reduce variance in REINFORCE

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha \, (G_t - \hat{v}(S_t, \boldsymbol{w}_t)) \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t, \boldsymbol{\theta}), \tag{22}$$

    and the weight for value function is updated using Gradient Monte Carlo method (12).

  - **Actor-Critic methods**: See Figure 3. In actor-critic methods, both value estimation and policy optimization is done iteratively based on **TD error**. The actor produce optimal policy given value and the critic evalute the policy by updating the value, then the actor optimize the policy based on new value.

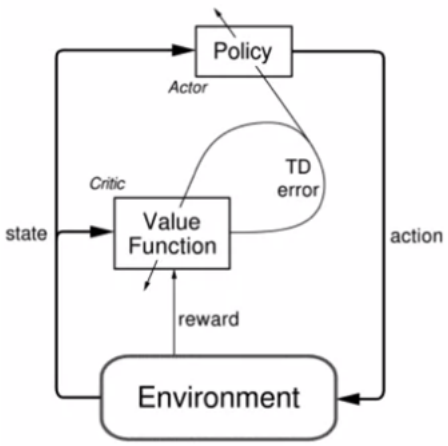


Figure 3: The Actor-Critic methods

* **Actor**: the role of an **actor** is to ***update the policy distribution*** via policy gradient algorithm. The updates for *actor* is shown below

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha_{\boldsymbol{\theta}} \underline{\delta_t} \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t, \boldsymbol{\theta}) \tag{23}$$

  where

$$\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}_t) - \hat{v}(S_t, \boldsymbol{w}_t)$$

  is the **TD error**. The update in **actor** is to **increase** the **value function** under new policy.

* **Critic**: Given the learned policy $\pi(a|\boldsymbol{s}, \boldsymbol{\theta})$, the role of a **critic** is to ***evalute and update* the *value* of the policy** and used it as a **feedback** for actor's performance. As shown in Figure 3, the same TD error δ_t is used by **semi-gradient methods** (e.g. TD(0)) for function apprximation.

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \alpha_{\boldsymbol{w}} \underline{\delta_t} \nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w}_t) \tag{24}$$

  The updates in **critics** is to adjust the value function to **match** the target value.

There are several commonly used parameterized policy distribution:

* When $\mathcal{A}$ is **finite discrete**, and $\boldsymbol{s} \in \mathcal{S}$, we can ues the **soft-max function** to approximate the policy function

$$\pi(a|\boldsymbol{s}, \boldsymbol{\theta}) = \frac{\exp\left(h(\boldsymbol{s}, a, \boldsymbol{\theta})\right)}{\sum_{a'} \exp\left(h(\boldsymbol{s}, a', \boldsymbol{\theta})\right)} \tag{25}$$

$$\log \pi(a|\boldsymbol{s}, \boldsymbol{\theta}) = h(\boldsymbol{s}, a, \boldsymbol{\theta}) - \log \sum_{a'} \exp\left(h(\boldsymbol{s}, a', \boldsymbol{\theta})\right) \tag{26}$$

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} \log \pi(a|\boldsymbol{s}, \boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} h(\boldsymbol{s}, a, \boldsymbol{\theta}) - \underline{\mathbb{E}_{\pi(a|\boldsymbol{s},\boldsymbol{\theta})}\left[\nabla_{\boldsymbol{\theta}} h(\boldsymbol{s}, a, \boldsymbol{\theta})\right]} \\ &= \nabla_{\boldsymbol{\theta}} h(\boldsymbol{s}, a, \boldsymbol{\theta}) - \sum_{a'} \pi(a'|\boldsymbol{s}, \boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} h(\boldsymbol{s}, a', \boldsymbol{\theta}) \end{aligned} \tag{27}$$

* When $\mathcal{A} \subset \mathbb{R}^k$ is **continuous space**, and $\boldsymbol{s} \in \mathcal{S}$, $\boldsymbol{\theta} = [\boldsymbol{\theta}_\mu, \boldsymbol{\theta}_\sigma]$

$$\pi(a|\boldsymbol{s}, \boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi}\sigma(\boldsymbol{s}, \boldsymbol{\theta}_\sigma)} \exp\left(-\frac{(a - \mu(\boldsymbol{s}, \boldsymbol{\theta}_\mu))^2}{2\sigma(\boldsymbol{s}, \boldsymbol{\theta}_\sigma)^2}\right) \tag{28}$$

where

$$\mu(\boldsymbol{s}, \boldsymbol{\theta}_\mu) = \langle \boldsymbol{\theta}_\mu \, , \, \boldsymbol{\phi}_\mu(\boldsymbol{s}) \rangle$$
$$\sigma(\boldsymbol{s}, \boldsymbol{\theta}_\sigma) = \exp(\langle \boldsymbol{\theta}_\sigma \, , \, \boldsymbol{\phi}_\sigma(\boldsymbol{s}) \rangle)$$

Note that

$$\nabla_{\boldsymbol{\theta}_\mu} \log \pi(a|\boldsymbol{s}, \boldsymbol{\theta}) = \frac{1}{\sigma(\boldsymbol{s}, \boldsymbol{\theta}_\sigma)^2} (a - \mu(\boldsymbol{s}, \boldsymbol{\theta}_\mu)) \, \boldsymbol{\phi}_\mu(\boldsymbol{s})$$
$$\nabla_{\boldsymbol{\theta}_\sigma} \log \pi(a|\boldsymbol{s}, \boldsymbol{\theta}) = \left(\frac{(a - \mu(\boldsymbol{s}, \boldsymbol{\theta}_\mu))^2}{\sigma(\boldsymbol{s}, \boldsymbol{\theta}_\sigma)^2} - 1\right) \boldsymbol{\phi}_\sigma(\boldsymbol{s})$$

2.3 Learning vs. Planning

In many situation, we may not have *full* information on our environment, but we have *some* knowledge of it. This knowledge is maintained via a model. A ***model*** of the environment means anything that an agent can use to predict how the environment will respond to its actions. Given a state and an action, a model **produces** a prediction of the resultant next state and next reward. A model is the **inner world** of an agent, which should **match the real world** during the learning process. Planning is **preferred** when

- there lacks of **immediate frequent feedbacks** from the real world. Many environments produce ***sparse*** and ***delayed*** rewards, which prohibits agents to explore and exploit efficiently.
- the **cost** of interacting with environment and obtaining real experience is **expensive**. For example, car crushing experiments, or flight pilot training program. The cost of a failure is not recovered.
- the **size** of training data is limited. This is similar to the situtation when the generative model is preferred over discriminative model in supervised learning.
- there exists **good model** that closely **match** the real world experience. Typical example is **games** or the classical **physical experiments**, where the dynamic can be precisely modeled.

Unlike the environment that generates the ***real*** *experience*, a model is used to ***simulate*** the environment and produce ***simulated experience***. The task of agent (lecture 7) is to learn from interaction with both real experience (i.e. **direct RL**) as well as simulated experience (i.e. **indirect RL**) while maintaining and learning the model (i.e. **model learning**). The **planning** is the process of learning from simulated experience. Figure 4 shows how the **learning and planning are integrated** in the agent's system. There are two type of models:

- ***distribution models***: models produce a description of *all possible* next-state-reward pairs and their probabilities, i.e. $p(s', r|s, a)$ for all next-state-reward pairs s', r given every state s and action a. Distribution models are *stronger* than sample models in that they can always be used to produce samples. A useful class of distribution models is the ***Probabilistic***

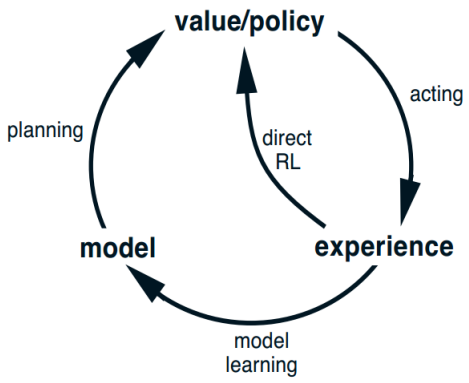


Figure 4: The generic relationship between model learning, direct RL, planning in a planning agent. Note that the value/policy can be updated either by real experience via direct RL or by simulated experience via planning. Meanwhile, the real experience will be used for model update/learning.

Graphical Model [Koller and Friedman, 2009] including the *Bayesian Networks*, *Markov Networks*, *Gaussian Graphical Models*, etc. These models use graph representation to *factorize* the global joint distribution between states, actions and rewards into several connected local factors embedded in subgraphs.

- ***sample models***: models produce just one of the possibilities, *sampled* according to the probabilities. Given probabilistic models, one can use **sampling methods** [Liu and Liu, 2001] such as *Importance Sampling*, *sequential sampling* such as *Gibs sampling*, *Markov Chain Monte Carlo (MCMC)*, or *Gradient Flow methods* etc to obtains samples of actions and states.

Within a planning agent, there are at least two roles for **real experience**:

- ***model learning***: i.e. **improve the model** to make it more accurately *match* the real environment;
- ***direct reinforcement learning (direct RL)***: i.e. directly **improve the value function** and **policy** using the kinds of reinforcement learning methods. All RL methods discussed before can be used here.

Figure 4 describes the major components in a planning agent and their interactions. Note how experience can improve value functions and policies either *directly* or *indirectly* via the model. It is the latter, which is sometimes called ***indirect reinforcement learning*** (i.e. learn via simulated experience), that is involved in ***planning***.

Figure 5 shows the architecture in ***Dyna-Q***, an architecture integrating the major functions needed in an *online planning agent*. It has the following components:

- **planning**: use random-sample one-step tabular Q-planning method.
- **direct RL**: one-step tabular Q-learning discussed in last chapter. Typically, as in Dyna-Q, the *same* reinforcement learning method is used both for learning from real experience and for planning from simulated experience. The reinforcement learning method is thus the "*final* ***common path***" for both learning and planning.
- **model-learning**: ***table-based*** and assumes the environment is ***deterministic***, i.e. given state-action pair, the output of model is **fixed** reward and next-state pair. It can be coded as

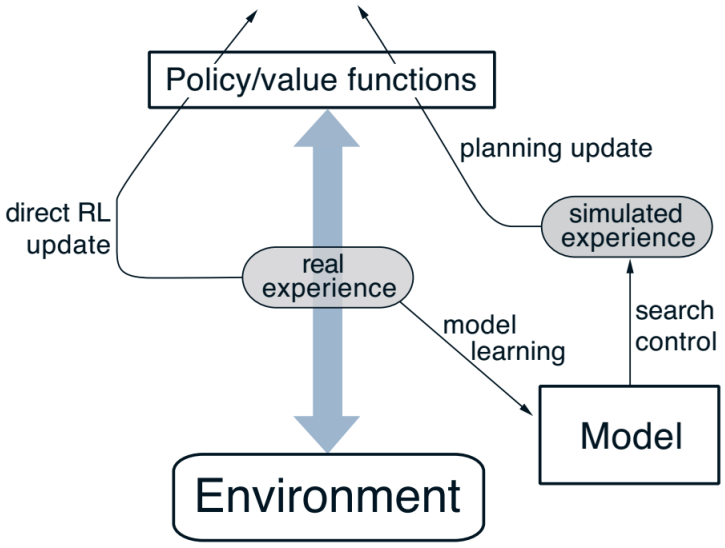


Figure 8.1: The general Dyna Architecture. Real experience, passing back and forth between the environment and the policy, affects policy and value functions in much the same way as does simulated experience generated by the model of the environment.

Figure 5: The Dyna-Q architecture.

dictionary of dictionaries. Under this assumption, model learning is simply saving the next state S_{t+1} and reward R_{t+1} for each experienced state S_t and action A_t.

- ***search control***: the process that selects the **starting states** and **actions** for the simulated experiences generated by the model.

2.4 Reward and Average reward

- **Returns for episodic task** (lecture 3): In episodic task, the task is terminated at some time T. The return is a finite number

$$G_t = \sum_{\tau=0}^{T-t-1} R_{\tau+t+1} < \infty \tag{29}$$

- **Discounted returns for continuing task** (lecture 3): In continuing task, there is no termination and the natural sum is infinite thus improper. In order to obtain finite return, we need to add discount factor $\gamma \in [0, 1)$

$$G_t = \sum_{\tau=0}^{\infty} \gamma^{\tau} R_{\tau+t+1} < \infty \tag{30}$$

Note that the choice of γ has significant impact on the behavior of learning agents. When $\gamma \to 1$, the agent emphasize the long-term gain over shot-term gain. When $\gamma \to 0$, it does the opposite.

Unlike the tabular cases, in which the returns from each state can be separately identified and averaged, using the discounted returns for function approximation in counting tasks is

questionable, esp. when the MDP is ergodic and has become **stationary** in the long run. In **ergodic continuing task**, because all states are the same, thus the *average* of the returns will be the asympototic discount factor $\frac{1}{1-\gamma}$ times the **average reward**, or $r(\pi)/(1-\gamma)$.

- **Average reward for ergodic continuing task** (lecture 9): If a MDP is ergodic, i.e.

$$\mu_\pi(s) = \lim_{t\to\infty} \Pr\{S_t = s | A_{0:t-1} \sim \pi\},$$

which is assumed to exist for any π and to be **independent** of S_0, the state distribution $\mu_\pi(s)$ does not change over time. At this status, the return defined as the cumulative reward function above is *meaningless*, since at time t and at time $t+1$ the underling system has not changed stochastically. Instead, we just need to consider the average-reward. In the average-reward setting, the quality of a policy π is defined as the ***average rate** of **reward***, or simply **average reward**, while following that policy, which we denote as $r(\pi)$:

$$\begin{aligned} r(\pi) &= \lim_{T\to\infty} \frac{1}{T} \mathbb{E}_\pi \left[G_{0:T} \middle| S_0, A_{0:t-1} \sim \pi \right] \\ &= \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} \mathbb{E}_\pi \left[R_t \middle| S_0, A_{0:t-1} \sim \pi \right] \qquad (31) \\ &= \lim_{t\to\infty} \mathbb{E}_\pi \left[R_t \middle| S_0, A_{0:t-1} \sim \pi \right] \qquad (32) \\ &= \sum_s \mu_\pi(s) \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) r, \qquad (33) \end{aligned}$$

where the expectations are conditioned on the initial state, S_0, and on the subsequent actions, $A_0, A_1, \ldots, A_{t-1}$, being taken according to π. $\mu_\pi(s)$ is the **steady-state distribution**. We consider $\pi_* = \text{argmax}_\pi\, r(\pi)$ as the **optimal policy**, which is the basis for **policy-based learning** algorithm.

Moreover, according to MDP, the stationary distribution is the **fixed-point solution** the equation below:

$$\mu_\pi(s') = \sum_s \mu_\pi(s) \left(\sum_a \pi(a|s) \sum_{s'} p(s'|s, a) \right) \qquad (34)$$

With the average rewards, we can define the the **differential return** as the cumulative *difference* between rewards and the average reward:

$$G_t = \sum_{\tau=0}^{\infty} (R_{\tau+t+1} - r(\pi)) < \infty \qquad (35)$$

Replacing (30) or (29) with (35), we can obtain the definition of ***differential value function*** of expected differential returns given state or state-action pair: $v_\pi(s) := \mathbb{E}_\pi [G_t | S_t = s]$ and $q_\pi(s, a) := \mathbb{E}_\pi [G_t | S_t = s, A_t = a]$. The differential returns measures how well our policy behaves as compared to the average reward under a **fixed policy**. We can also reformulate the **Bellman equation** and **Bellman optimality equation** for **average reward MDP**

by removing γ and replacing reward r with differential reward $r - r(\pi)$.

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r - r(\pi) + v_\pi(s') \right], \quad \forall\ s \in \mathcal{S} \tag{36}$$

$$= \mathbb{E}_\pi \left[R_{t+1} - r(\pi) + \ v_\pi(S_{t+1}) | S_t = s \right] \tag{37}$$

$$q_\pi(s, a) = \sum_{s',r} p(s', r|s, a) \left[r - r(\pi) + \sum_{a'} \pi(a'|s') q_\pi(s', a') \right] \quad \forall\ s \in \mathcal{S},\ a \in \mathcal{A}(s) \tag{38}$$

$$= \mathbb{E}_\pi \left[R_{t+1} - r(\pi) + q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a \right] \tag{39}$$

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s', r|s, a) \left[r - \max_\pi r(\pi) + v_*(s') \right] \tag{40}$$

$$= \max_{a \in \mathcal{A}(s)} \mathbb{E} \left[R_{t+1} - \max_\pi r(\pi) + v_*(S_{t+1}) | S_t = s, A_t = a \right] \tag{41}$$

$$q_*(s, a) = \sum_{s'} \sum_r p(s', r|s, a) \left[r - \max_\pi r(\pi) + \max_{a' \in \mathcal{A}(s)} q_*(s', a') \right] \tag{42}$$

$$= \mathbb{E} \left[R_{t+1} - \max_\pi r(\pi) + \max_{a' \in \mathcal{A}(s)} q_*(S_{t+1}, a') \Big| S_t = s, A_t = a \right] \tag{43}$$

References

Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge university press, 2006.

Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, Joelle Pineau, et al. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354, 2018.

Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Jun S Liu and Jun S Liu. *Monte Carlo strategies in scientific computing*, volume 10. Springer, 2001.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Shai Shalev-Shwartz et al. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2012.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.