

18-661 Introduction to Machine Learning

Reinforcement Learning

Spring 2020

ECE – Carnegie Mellon University

Announcements

- Homework 7 is due on Friday, April 24. You may use a late day if you have any left.
- Wednesday's lecture will be a set of four guest mini-lectures from Samarth Gupta, Jianyu Wang, Mike Weber and Yuhang Yao. **Please attend!**
- Recitation on Friday will be a review for the final exam.
- Practice final (multiple choice only) on Monday, April 27.
- **Final exam, Part I:** Wednesday, April 29 during the usual course time. We will email students with a timezone conflict about starting and finishing 1 hour earlier. Closed book except one double-sided letter/A4-size handwritten cheat sheet.
- **Final exam, Part II:** Take-home exam from Friday, May 1 (evening) to Sunday, May 3. Open everything except working with other people, designed to take 2 to 3 hours.

The final will cover all topics in the course.

There will be less emphasis (40%) on pre-midterm topics.

- **Pre-midterm topics:** MLE/MAP, Linear regression, Bias-variance tradeoff, Naive Bayes, Logistic regression, SVM
- **Non-parametric methods:** Nearest neighbors, Decision trees
- **Ensemble methods:** Bagging, Random forests, AdaBoost
- Neural networks
- **Unsupervised learning:** Clustering, EM algorithm, PCA/ICA
- **Online and reinforcement learning**

Homework, midterm, recitation, and mock final questions are all good practice for the final.

1. Online Learning Recap
2. Reinforcement Learning
3. Markov Decision Processes
4. Finding RL Policies

Online Learning Recap

What Is Online Learning?

Online learning occurs when **we do not have access to our entire training dataset when we start training**. We consider a **sequence** of data and update the predictor based on the latest sample(s) in the sequence.

- Stochastic gradient descent
- Perceptron training algorithm
- Multi-armed bandits

Online learning has **practical advantages**:

- Helps us **handle large datasets**.
- Automatically **adapts models** to changes in the underlying process over time (e.g., house prices' relationship to square footage).

What if we don't get full feedback on our actions?

- Spam classification: we know whether our prediction was correct.
- Online advertising: we only know the appeal of the ad shown. We have *no idea* if we were right about the appeal of other ads.
- Partial information often occurs because we observe **feedback from an action taken on the prediction**, not the prediction itself.
 - Analogy to linear regression: instead of learning the ground truth y , we only observe the value of the loss function, $l(y)$.
 - Makes it very hard to optimize the parameters!
- Often considered via **multi-armed bandit** problems.

Evaluating Online Learning Models

How do we evaluate online learning models?

- Previously, we measured the loss of our model on test data—but the model changes over time! For model parameters β :
 $\beta_1 \rightarrow \beta_2 \rightarrow \dots \beta_t \rightarrow \dots$
- Define **regret** as the cumulative difference in loss compared to the best model in hindsight. For a sequence of data (x_t, y_t) :

$$R(T) = \sum_{t=1}^T [l(x_t, y_t, \beta_t) - l(x_t, y_t, \beta^*)]$$

Evaluate the loss of each sample using the model parameters at that time, compared to the best parameters in hindsight.

- Usually, we want the regret to decay sub-linearly over time.
- Implies that the **regret per iteration decays to zero**: eventually, you will recover the optimal-in-hindsight model.

An Example Problem

Can we learn which slot machine gives the most money?



\$1
\$0
\$0



\$1
\$4
\$0
\$2
\$1
\$3
\$5



\$1
\$0
\$1
\$2

Bandit Formulation

We can play multiple rounds $t = 1, 2, \dots, T$. In each round, we **select an arm** i_t from a fixed set $i = 1, 2, \dots, n$; and **observe the reward** $r(i_t)$ that the arm gives.

Arm 1



Arm 2



Arm 3



Objective: Maximize the total reward over time, or equivalently minimize the regret compared to the best arm in hindsight.

Bandit Formulation

Arm 1



Arm 2



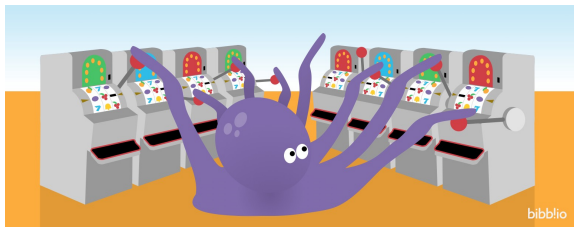
Arm 3



- The reward at each arm is **stochastic** (e.g., 0 with probability p_i and otherwise 1).
- Usually, the rewards are i.i.d over time. The **best arm** is then the arm with highest expected reward.
- We cannot observe the reward of each arm (the entire reward function): we just know the reward of the arm that we played.

Online ads example: arm = ad, reward = 1 if the user clicks on the ad and 0 otherwise

Exploration vs. Exploitation Tradeoff



Which arm should I play?

- Best arm observed so far? (exploitation)
- Or should I look around to try and find a better arm? (exploration)

We need both in order to maximize the total reward.

The ϵ -Greedy Algorithm

Very simple solution that is easy to implement. The idea is to exploit the best arm, but **explore a random arm ϵ fraction of the time**.

- Try each arm and observe the reward.
- Calculate the empirical average reward for each arm i :

$$\overline{r(i)} = \frac{\text{total reward from pulling this arm in the past}}{\text{number of times I pulled this arm}} = \frac{\sum_{t:j(t)=i} r_t}{T_i},$$

where $j(t) = i$ indicates that arm i was played at time t , r_t is the reward, and T_i is the number of times arm i has been played.

- With probability $1 - \epsilon$, play the arm with highest $\overline{r(i)}$. Otherwise, choose an arm at random.
- Observe the reward and re-calculate $\overline{r(i)}$ for that arm.
- Repeat steps 2-4 for T rounds.

The UCB1 Algorithm

Very simple policy from Auer, Cesa-Bianchi, and Fisher (2002) with $O(\log T)$ regret. The idea is to **try the best arm**, where “best” includes exploration and exploitation.

- Try each arm and observe the reward.
- Calculate the UCB (upper confidence bound) index for each arm i :

$$UCB_i = \overline{r(i)} + \sqrt{\frac{2 \log T}{T_i}},$$

where $\overline{r(i)}$ is the empirical average reward for arm i and T_i is the number of times arm i has been played.

- Play the arm with the highest UCB index.
- Observe the reward and re-calculate the UCB index for each arm.
- Repeat steps 2-4 for T rounds.

Thompson Sampling

Which arm should you pick next?

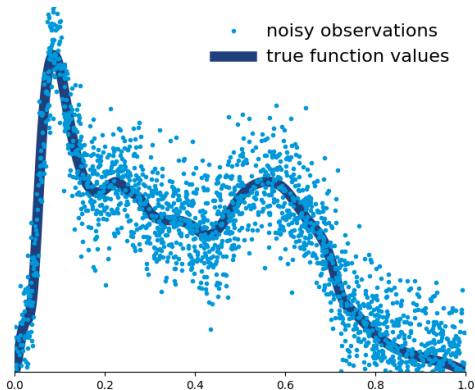
- **ϵ -greedy**: Best arm so far (exploit) with probability $1 - \epsilon$, otherwise random (explore).
- **UCB1**: Arm with the highest UCB value.

Thompson sampling instead fits a Gaussian distribution to the observations for each arm.

- Assume the reward of arm i is drawn from a normal distribution.
- Find the posterior distribution of the expected reward for each arm: $\mathcal{N}(\overline{r(i)}, (T_i + 1)^{-1})$.
- Generate synthetic samples from this posterior distribution for each arm, which represent your understanding of its average reward.
- Play the arm with the highest sample.

Continuous Bandits

- So far we have assumed a finite number of discrete arms.
- What happens if we assume continuous arms?



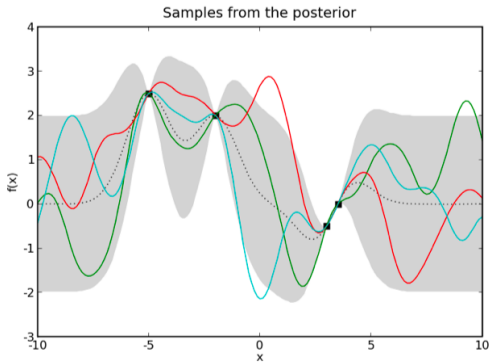
x-axis is the arm, y-axis is the (stochastic) reward.

Bayesian Optimization

Assume that $f(x)$ is drawn from a Gaussian process

$$\{f(x_1), f(x_2), \dots, f(x_n)\} \sim N(\mathbf{0}, \mathbf{K})$$

where \mathbf{K} is a kernel function, e.g., $K_{ij} = \exp(-\|x_i - x_j\|_2^2)$.



Reinforcement Learning

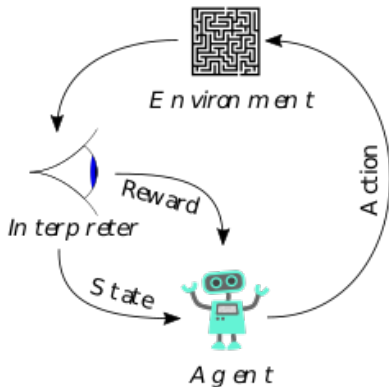
Reinforcement vs. Online Learning

Reinforcement learning occurs when we take actions so as to maximize the expected reward, given the current state of the system.

- Our actions may influence future states.
- We want to consider the total future reward, not just the current reward.
- Must navigate the exploration-exploitation tradeoff: can only observe rewards and future states once an action is taken.
- Generally formulated as an online learning problem.

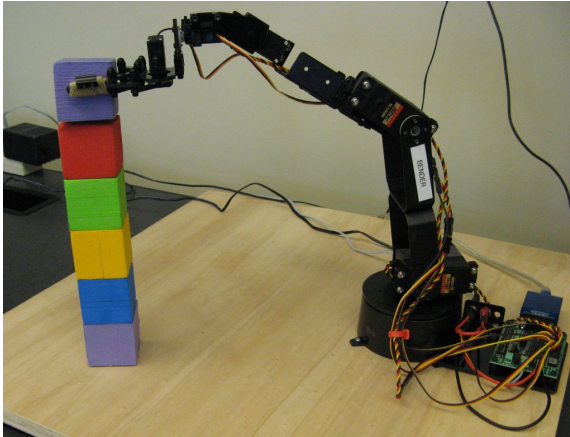
Reinforcement learning reinforces the agents' decisions over time by observing the reward and state that results from taking different actions.

Overview of Reinforcement Learning



Reinforcement learning is also sometimes called **approximate dynamic programming**. It can be viewed as a type of **optimal control theory** with no pre-defined model of the environment.

Grasping an Object



Reinforcement learning can be applied to many different areas.

- **Robotics:** in which direction and how fast should a robot arm move?
- **Mobility:** where should taxis go to pick up passengers?
- **Transportation:** when should traffic lights turn green?
- **Recommendations:** which news stories will users click on?
- **Network configuration:** which parameter settings lead to the best allocation of resources?

Similar to multi-armed bandits, but with a notion of **state** or **context**.

Reinforcement Learning Formulation

Consider a sequence of time $t = 1, 2, \dots, T$. At each time t , the **agent** experiences a **state** $s(t) \in S$ (which may be represented as a vector of real numbers). We sometimes call the state the “**environment.**”

- Position of objects that a robot wants to grasp.
- Number of passengers and taxis at different locations in a city.

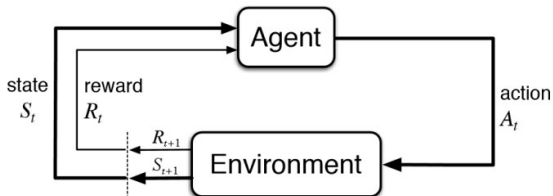
At each time t , the agent takes an **action** $a(t)$, which is chosen from some feasible set $A(t)$. It then experiences a **reward** $r(a(t), s(t))$.

- Direction in which a robot/taxi moves.
- Reward if an object is grasped/passenger is picked up.

The **next state** (at time $t + 1$) is a (probabilistic) function of the current state and action taken: $s(t + 1) \sim \sigma(a(t), s(t))$.

Reinforcement Learning Formulation

- At each time t , the **agent** experiences a **state** $s(t)$. We sometimes call the state the “**environment**.”
- At each time t , the agent takes an **action** $a(t)$, which is chosen from some feasible set A . It then experiences a **reward** $r(a(t), s(t))$.
- The **next state** (at time $t + 1$) is a (probabilistic) function of the current state and action taken: $s(t + 1) \sim \sigma(a(t), s(t))$.



Objectives of Reinforcement Learning

We choose the actions that **maximize the expected total reward**:

$$R(T) = \sum_{t=0}^T \mathbb{E}[r(a(t), s(t))], \quad R(\infty) = \sum_{t=0}^{\infty} \mathbb{E}[\gamma^t r(a(t), s(t))].$$

We *discount* the reward at future times by $\gamma < 1$ to ensure convergence when $T = \infty$. The expectation is taken over the probabilistic evolution of the state, and possibly the probabilistic reward function.

A **policy** tells us which action to take, given the current state.

- **Deterministic policy**: $\pi : S \rightarrow A$ maps each state s to an action a .
- **Stochastic policy**: $\pi(a|s)$ specifies a probability of taking each action $a \in A$ given state s . We draw an action from this probability distribution whenever we encounter state s .

Example: Robot Movements

			+1
			-1
START			

actions: UP, DOWN, LEFT, RIGHT

UP

80% move UP
10% move LEFT
10% move RIGHT



- Reward of + 1 if we reach [4,3] and -1 if we reach [4,2]; -0.04 for taking each step
- What action should we take at state [3,3]? **RIGHT**
- How about at state [3,2]? **UP**

Key Challenges of Reinforcement Learning

- The relationship of future states to past states and actions, $s(t+1) \sim \sigma(a(t), s(t))$, must be learned.
- Partial information feedback: the reward feedback $r(a(t), s(t))$ only applies to the action taken $a(t)$, and may itself be stochastic. Moreover, we may not be able to observe the full state $s(t)$ (more on this later).
- Since actions affect future states, they should be chosen so as to maximize the total future reward $\sum_{t=0}^{\infty} \gamma^t r(a(t), s(t))$, not just the current reward.

We can address these challenges by **formulating RL using Markov decision processes**.

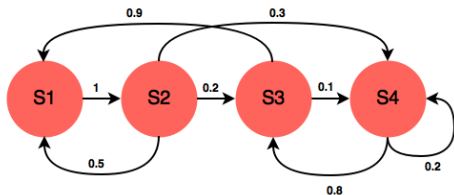
Markov Decision Processes

Finite-State, Discrete-Time Markov Chains

Suppose that $S = \{x_1, x_2, \dots, x_n\}$: these are the possible state values. Then if $s(t) = x_i$, we can represent the probability distribution of $s(t + 1)$ as:

$$\mathbb{P}(s(t + 1) = x_j) = p_{i,j}, \quad (1)$$

where the probabilities $p_{i,j}$ are called **transition probabilities**. That is, they give the probability that the state i at the current time will transition to state j at the next time.

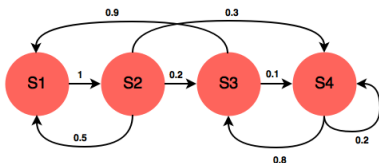


Transition Matrices

A Markov chain can be represented with a **transition matrix**:

$$P = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,n} \\ \vdots & \ddots & \vdots & \vdots \\ p_{n,1} & p_{n,2} & \cdots & p_{n,n} \end{bmatrix} \quad (2)$$

Each entry (i, j) is the probability of transitioning from state i to state j .



$$P = \begin{bmatrix} & S1 & S2 & S3 & S4 \\ S1 & 0 & 1 & 0 & 0 \\ S2 & 0.5 & 0 & 0.2 & 0.1 \\ S3 & 0.9 & 0 & 0 & 0.1 \\ S4 & 0 & 0 & 0.8 & 0.2 \end{bmatrix}$$

Markov chains are special types of **Markov processes**, which extend the notion of a Markov chain to possibly infinite numbers of states and continuous time.

- **Infinite states:** For instance, if the state is the temperature in this room.
- **Continuous time:** For instance, if the state is the velocity of a robot arm.
- In both cases, we can still define transition probabilities, due to the **Markov property**.

The **Markov property** says that the state evolution is memoryless: the probability distribution of future states depends only on the value of the current state, not the values of previous states.

Motivating the Markov Property

The memorylessness of the Markov property significantly simplifies our predictions of future states (and thus, future rewards).

- **Robotic arms:** What is the probability that a block moves from point A to point B in the next 5 minutes? Does it matter where the block was 5 minutes ago? 10 minutes ago?
- **Taxi mobility:** What is the probability there will be, on average, 10 taxis at SFO in the next 10 minutes? Does this depend on the number of taxis that were there 10 minutes ago? 20 minutes ago?

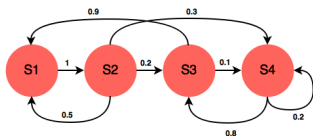
Introducing Decision Variables

Markov decision processes make state transitions dependent on user actions.

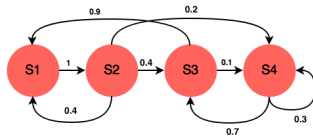
Markov processes represent transitions from state $s(t)$ to $s(t + 1)$, but they do not include actions taken by the user.

$$p_{i,j} \rightarrow p_{i,j}^a,$$

where a is a given action. Note that the transition probabilities $p_{i,j}^a$ must still be learned over time.



Action 1



Action 2

Partial Observability

In some scenarios, we may not know the full state of the system:

- **Robotic arms:** Position of the blocks outside the robot's field of view.
- **Taxi mobility:** Locations of all other taxis.
- **Recommendations:** Users' political leanings influence their susceptibility to ads.

Even though we can't see the full system state, we know that these "hidden" state variables exist. We call these **partially observable Markov decision processes** (POMDPs). In general, these are handled by **inferring the distribution of possible state variables** based on the observable variables.

Markov Decision Processes in RL

- At each time t , the **agent** experiences a **state** $s(t)$. We sometimes call the state the “**environment**.”
- At each time t , the agent takes an **action** $a(t)$, which is chosen from some feasible set A . It then experiences a **reward** $r(a(t), s(t))$.
- The **next state** (at time $t + 1$) is a (probabilistic) function of the current state and action taken: $s(t + 1) \sim \sigma(a(t), s(t))$.

The state-action relationships are just a Markov decision process!

$\sigma(a(t), s(t))$ is given by the transition probabilities $p_{s(t),j}^{a(t)}$, where j are the possible states.

Finding RL Policies

The State-Value Function

The **state-value function** of a given policy $\pi : S \rightarrow A$ gives its expected future reward when starting at state s :

$$V_{\pi}(s) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(\pi(s(t)), s(t)) \mid s(0) = s \right].$$

The expectation may be taken over a stochastic policy and reward as well as the Markov decision process (MDP) of the state transitions.

- The action $a(t)$ at any time t is determined by the policy, $\pi(s(t))$.
- Due to the Markov property of the underlying MDP, the optimal policy at any time is only a function of the last observed state.

Finding an Optimal Policy

$$V_{\pi}(s) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(\pi(s(t)), s(t)) \mid s(0) = s \right].$$

We want to find the policy π that **maximizes the state-value function**.

This is easier said than done:

- We might not know the underlying Markov decision process, requiring us to balance exploration and exploitation.
- Even if we knew the underlying process, exhaustive search and analytical solutions do not scale to infinitely many timeslots or possible states.

The Action-Value Function

Consider a variant of $V_\pi(s)$:

$$Q_\pi(a, s) = \mathbb{E} \left[r(a, s) + \underbrace{\sum_{t=1}^{\infty} \gamma^t r(\pi(s(t)), s(t))}_{\text{use policy } \pi} \mid s(0) = s, a(0) = a \right]$$

The Q function gives the expected value of taking an action a at time 0, and then following a given policy π . How is this function related to the optimal policy?

- Suppose we know the value of $Q_{\pi^*}(a, s)$, for each action a and state s .
- Then, we maximize the reward by choosing the action $a^*(s)$ at state s so as to maximize $Q_{\pi^*}(a, s)$. But this gives us π^* !

Optimizing the Action-Value Function

$$Q_{\pi^*}(a, s) = \mathbb{E} \left[r(a, s) + \underbrace{\sum_{t=1}^{\infty} \gamma^t r(\pi^*(s(t)), s(t))}_{\text{use policy } \pi^*} \mid s(0) = s, a(0) = a \right]$$

- We maximize the reward by choosing the action $a^*(s)$ at state s as $a^*(s) = \arg \max_a Q_{\pi^*}(a^*(s), s)$.
- What does π^* do at time $t = 1$, and state $s(1)$? We don't care about what we did at $t = 0$...so we can pretend $t = 0$ again and just choose a to maximize $Q_{\pi^*}(a, s(1))!$
- This logic is related to Bellman's Principle of Optimality (for those familiar with dynamic programming).

Given the above insight, it suffices to learn **either** the optimal policy π^* or the optimal action-value function Q^* . When we know the MDP of the state transitions, these approaches are called **policy iteration** and **value iteration**. They depend on the **Bellman equation**:

$$Q_{\pi^*}(a, s) = r(a, s) + \underbrace{\gamma \sum_{s' \in \mathcal{S}} p_{s, s'}^a [V_{\pi^*}(s')]}_{\text{Maximum future reward}},$$

which follows from the fact that $V_{\pi^*}(s) = \max_a Q_{\pi^*}(a, s)$.

Value Iteration

Initialize $V(s)$ for each state s . Suppose we know the reward function $r(a, s)$ and transition probabilities $p_{s,s'}^a$.

- Update $Q(a, s)$:

$$Q(a, s) \leftarrow \mathbb{E}[r(a, s)] + \underbrace{\gamma \sum_{s' \in \mathcal{S}} p_{s,s'}^a V(s')}_{\text{Estimate of expected future reward}}$$

- Update $V(s) \leftarrow \max_a Q(a, s)$.
- Repeat until convergence.

Value iteration is **guaranteed to converge** to the optimal value function $V^*(s)$, and optimal action-value function $Q^*(a, s)$.

Policy Iteration

Initialize $V(s)$ and a deterministic policy $\pi(s)$ for each state s . Suppose we know the reward function $r(a, s)$ and transition probabilities $p_{s,s'}^a$.

- Update the estimate of $V(s)$:

$$V^\pi(s) \leftarrow \mathbb{E}[r(\pi(s), s)] + \gamma \underbrace{\sum_{s' \in \mathcal{S}} p_{s,s'}^{\pi(s)} V^\pi(s')}_{\text{Estimate of expected future reward}}$$

- Improve the policy at each state s :

$$\pi(s) \leftarrow \arg \max_a \left[\mathbb{E}[r(a, s)] + \gamma \sum_{s' \in \mathcal{S}} p_{s,s'}^a V^\pi(s') \right]$$

- Repeat until π converges.

Monte Carlo Methods

What if we don't know $r(a, s)$ and $p_{s,s'}^a$? Still try to evolve the policy so as to maximize $Q_\pi(a, s)$.

- **Policy evaluation:** Given a policy, observe the rewards $r(\pi(s), s)$ obtained, using Monte Carlo sampling of the possible realized states. After a sufficient amount of exploration, averaging the rewards obtained will yield a good approximation to $Q_\pi(a, s)$.
- **Policy improvement:** Once we approximate $Q_\pi(a, s)$, we can update the policy. For instance, for each state s , we play the action a that maximizes $Q_\pi(a, s)$.

Monte Carlo methods can be inefficient, as they may spend too much time evaluating a bad policy. **Temporal difference methods**, which update the state-value function $V_\pi(s)$ and action-value function $Q_\pi(a, s)$ as we observe states, can help correct this issue.

Direct Policy Search

If we parameterize the policy, then finding the optimal policy simply means finding the optimal parameter values.

- **Gradient descent:** Try to estimate the gradients of the action-value function $Q_{\pi}(a, s)$ and evolve the parameters accordingly. This can be difficult since we don't know $Q_{\pi}(a, s)$ in the first place.
- **Evolutionary optimization:** Simulated annealing, cross-entropy search, etc. are generic optimization algorithms that do not require knowledge of the gradient.

Such methods often require temporal difference adjustments in order to converge fast enough.

Evolve the **action-value function** to find its optimal value $Q_{\pi^*}(a, s)$.

- Initialize our estimate of $Q(a, s)$ to some arbitrary value. We have dropped the dependence on π^* , as π^* is determined by $Q_{\pi^*}(a, s)$.
- After playing action a in state $s(t)$ and observing the next state $s(t+1)$, update Q as follows:

$$Q(a, s(t)) \leftarrow (1-\alpha) \underbrace{Q(a, s(t))}_{\text{old value}} + \alpha \underbrace{\left(r(a, s(t)) + \gamma \max_a Q(a, s(t+1)) \right)}_{\text{learned value}}.$$

Here α is the learning rate and $r(a, s(t))$ is our observed reward.

The term $\max_a Q(a, s(t+1))$ is our estimate of the expected future reward for the optimal policy π^* .

Q-learning has many variants: for instance, deep Q-learning uses a neural network to approximate Q .

Exploration vs. Exploitation

Given $Q(a, s)$, how do we choose our action a ?

- **Exploitation:** Take action $a^* = \arg \max_a Q(a, s)$. Given our current estimate of Q , we want to take what we think is the optimal action.
- **Exploration:** But we might not have a good estimate of Q , and we don't want to bias our estimate towards an action that turns out not to be optimal.
- **ϵ -Greedy:** With probability $1 - \epsilon$, choose $a^* = \arg \max_a Q(a, s)$, and otherwise choose a randomly. Usually, we decrease ϵ over time as additional exploration becomes less important.

Q-Learning Algorithm

Initialize our estimate of $Q(a, s)$ to some arbitrary value.

- Choose an action a using the ϵ -greedy policy.
- Observe reward $r(a, s(t))$ and state $s(t)$.
- Update Q as follows:

$$Q(a, s(t)) \leftarrow (1-\alpha) \underbrace{Q(a, s(t))}_{\text{old value}} + \alpha \underbrace{\left(r(a, s(t)) + \gamma \max_a Q(a, s(t+1)) \right)}_{\text{learned value}}.$$

- Repeat for T iterations.

Q-learning always learns an optimal policy, regardless of how a is chosen (you don't need to use ϵ -greedy)! Thus, it is an **off-policy** method.

Implementing Reinforcement Learning

- **“Offline” version:** we have access to several state-action trajectories $\{(s_0, a_0); (s_1, a_1); \dots; (s_T, a_T)\}$ that form our training data.
- **“Pre-training” version:** we have access to a simulation of the environment that tells us what the state will be when we take an action. We will train our RL algorithm on the simulator before deploying it in the real world.
- **“Fully online” version:** we can only learn about our environment by directly interacting with it.

The “pre-training” version is often used in practice, as it limits the risk of taking bad actions in deployment without requiring extensive training data.

Extensions and Variations of RL

- **Multi-agent reinforcement learning:** Suppose multiple agents are simultaneously using RL to find their optimal actions, and that one agent's actions affect another's. The agents must then learn to compete with each other.
- **Distributed reinforcement learning:** We can speed up the search for the optimal policy by having multiple agents explore the state space in parallel.
- **Hierarchical reinforcement learning:** Lower-level learners try to satisfy goals specified by a higher-level learner, which are designed to maximize an overall reward.
- **Transfer learning:** Learn how to perform a new task based on already-learned methods for performing a related one.

Types of Machine Learning

Supervised Learning

- Training data: (x, y) (features, label) samples. We want to predict y to minimize a loss function.
- Regression, classification

Unsupervised Learning

- Training data: x (features) samples only. We want to find “similar” points in the x space.
- Clustering, PCA/ICA

Reinforcement Learning

- Training data: (s, a, r) (state, action, reward) samples. We want to find the best sequence of decisions so as to maximize long-term reward.
- Robotics, multi-armed bandits

You should know:

- What a Markov decision process is (action and state variables, transition probabilities).
- What the action-value and state-value functions are.
- Differences between supervised, unsupervised, and reinforcement learning.