CMPSC 448: Machine Learning

Lecture 16. Reinforcement Learning and Bandits

Rui Zhang

Fall 2021

PennState

What types of ML are there?

Supervised Learning

Unsupervised Learning

Reinforcement Learning

Outline

- Introduction to Reinforcement learning
- Multi-armed Bandits
- Markov Decision Processes (MDP)
  - Dynamic Programming when we know the world
- Learning in MDP: When we don't know the world
  - Monte Carlo Methods
  - Temporal-Difference Learning (TD): SARSA and Q-Learning

Note: All of the lectures are tabular methods; we will only briefly discuss the motivation of function approximation methods (e.g., DQN, policy gradient, deep reinforcement learning)

What is reinforcement learning?

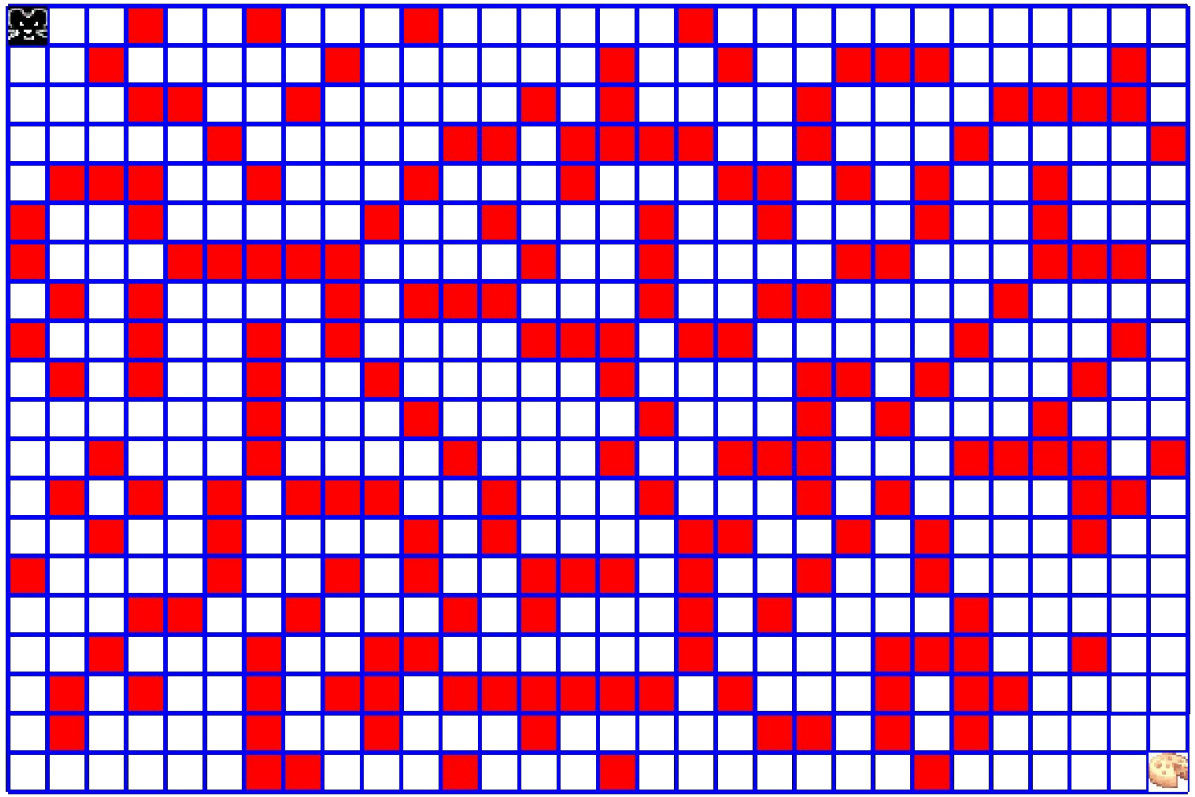
How to build agents that learn behaviors in a dynamic world?

- Agent-oriented learning
- learning by interacting with an environment to achieve a goal
- more natural, realistic, and ambitious than other kinds of machine learning

RL is a general-purpose framework for decision-making

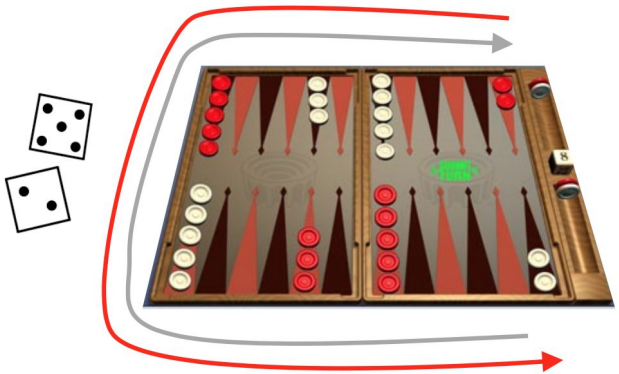
- RL is for an agent with the capacity to act
- Each action influences the agent's future state
- Success is measured by a scalar reward signal
- Goal: select actions to maximize the future reward
- The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them.
- 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.

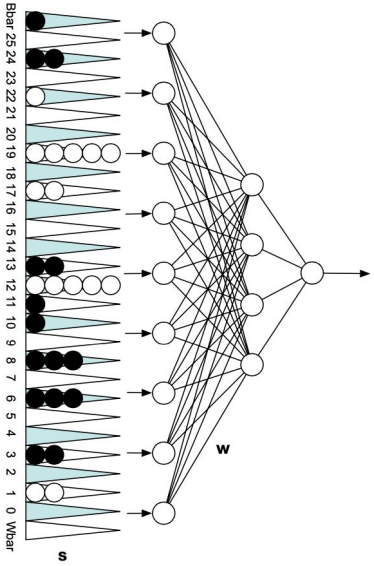
Maze



https://www.samyzaf.com/ML/rl/qmaze.html

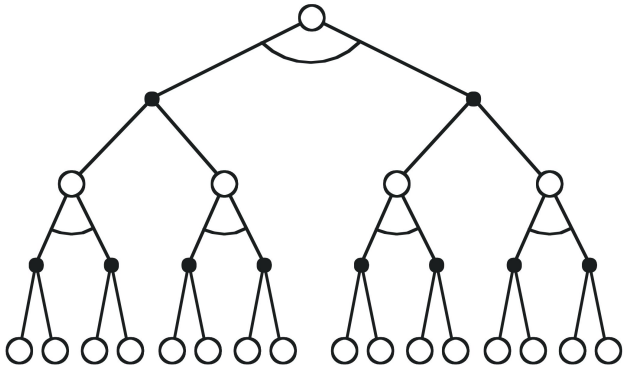
TD-Gammon





estimated state value (≈ prob of winning)

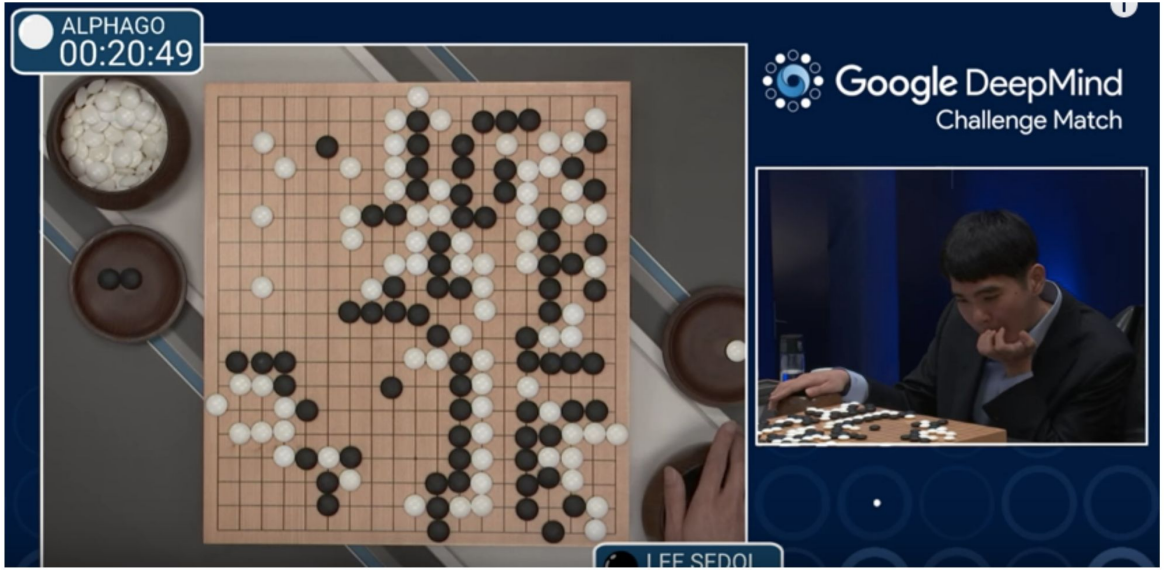
Action selection by a shallow search



Start with a random Neural Network
Play millions of games against itself (i.e., self-play)
Learn a value function from this simulated experience
Six weeks later it's the best player of backgammon in the world
Originally used expert handcrafted features, later repeated with raw board positions

AlphaGo

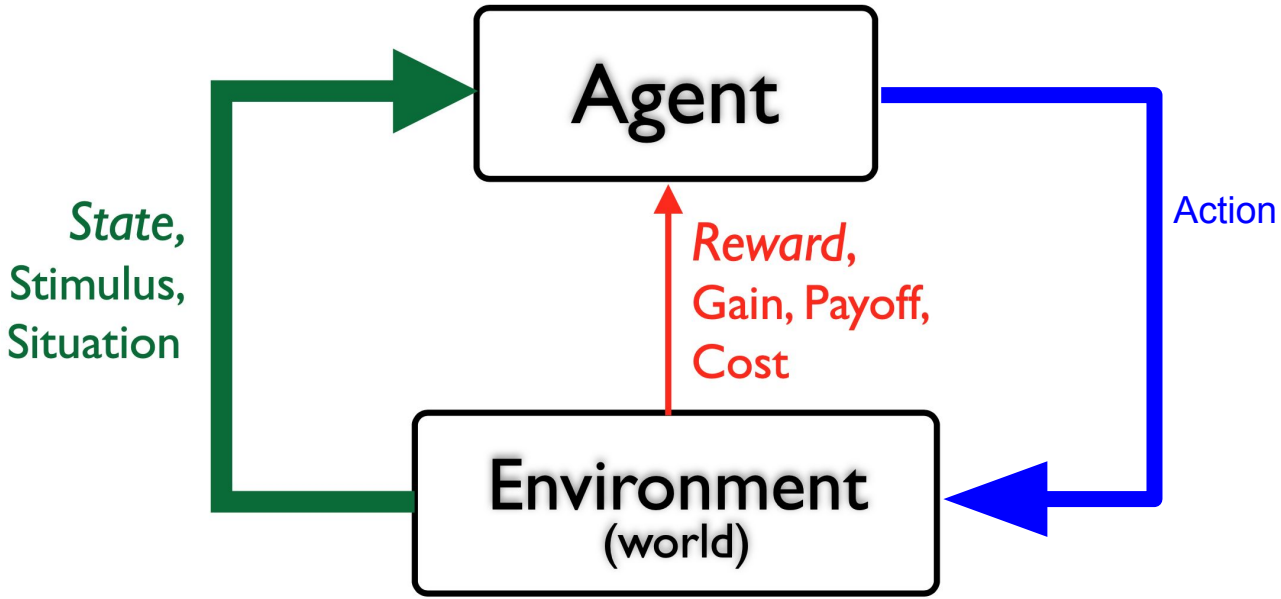
In October 2015, AlphaGo became the first computer Go program to beat a human professional Go player without handicaps on a full-sized 19×19 board.



Monte Carlo Tree Search, learning policy and value function networks for pruning the search tree, trained from expert demonstrations, self play, and Tensor Processing Unit

The RL interface between Agent and Environment

Agent is in a **state**, takes an **action**, gets some **reward** for the pair of (state, action), and goes to a **new state**!



RL Terms

A set of **States**

- These are the possible positions of our mouse within the maze.

A set of **Actions** available in each state

- This is {forward, back} in a corridor and {forward, back, left, right} at a crossroads.

Transitions between states

- For example, if you go left at a crossroads you end up in a new position. These can be a set of probabilities that link to more than one possible state (e.g. when you use an attack in a game of Pokémon you can either miss, inflict some damage, or inflict enough damage to knock out your opponent).

Rewards associated with each transition

- In the robot-mouse example, most of the rewards are 0, but they're positive if you reach a point that has water or cheese and negative if you reach a point that has an electric shock.

Policy

- A mapping from states to actions aiming to maximize its cumulative reward (how to map situations to actions).

Notations

We will follow the following notation for known and unknown variables:

Known (lower case)

- States $s_1, s_2, \ldots,$
- Actions $a_1, a_2, \ldots,$
- Rewards $r_1, r_2, \ldots,$

Variables (upper case)

S_t : state at time t

A_t : action taken at time t

R_{t+1} : reward at time t for action A_t in state S_t

Dynamics

State transition probability. May or may not be known. Could be deterministic or random

$$\mathbb{P}(S_{t+1}|S_t, A_t)$$

Dynamics

State transition probability. May or may not be known. Could be deterministic or random

$$\mathbb{P}(S_{t+1}|S_t, A_t)$$

Distribution over rewards. May or may not be known. Could be deterministic or random

$$\mathbb{P}(R_{t+1}|S_t, A_t)$$

Dynamics

State transition probability. May or may not be known. Could be deterministic or random

$$\mathbb{P}(S_{t+1}|S_t, A_t)$$

Distribution over rewards. May or may not be known. Could be deterministic or random

$$\mathbb{P}(R_{t+1}|S_t, A_t)$$

Goal is to learn a policy π which is a function whose input is a state and output is an action (might be randomized)

$$\pi : S_t \longrightarrow A_t$$

Outline

- Introduction to Reinforcement learning
- Multi-armed Bandits
  - Formulation
  - Regret
  - Action-Value Methods
  - ε -greedy action selection
  - UCB action selection
  - Gradient bandits
- Markov Decision Processes (MDP)
- Learning in MDP: When we don't know the world

Multi-armed Bandits

The simplest reinforcement learning problem

One state (no state transition probabilities)

Actions: k levers (arms), each action is associated with a reward

Policy is to sequentially choose arms to maximize cumulative reward

The k-armed bandit problem

On each of an infinite sequence of time steps, $t = 1, 2, 3, \ldots$
you choose an action A_t from k possibilities, and receive a real-valued reward R_t

The reward depends only on the action taken; it is identically, independently distributed given the action:

$$q_*(a) \doteq \mathbb{E}[R_t | A_t = a], \quad \forall a \in \{1, \ldots, k\}$$

true values

These true reward distribution is unknown. Nevertheless, you must **maximize total reward** (equivalent to **minimize total regret**)

You must both try actions to learn their values (**explore**), and prefer those that appear best (**exploit**)

Regret

Goal: minimize the REGRET:

$$\sum_{t=1}^{T} \textbf{[mistakes by leaner]} - \sum_{t=1}^{T} \textbf{[mistakes by best expert in hindsight]}$$

- Low regret means that we do not lose much from not knowing future events.
- We can perform almost as well as someone who observes the entire sequence and picks the best prediction strategy in hindsight
- We cannot compute regret (because this requires knowing the best arm), but we use it to analyze our algorithm
- We can also compete with changing environment

Examples

If rewards are deterministic and known

- policy: pull arm with highest reward (exploitation)

Examples

If rewards are deterministic and known

- policy: pull arm with highest reward (exploitation)

If rewards are deterministic and unknown

- policy: try each arm (exploration), then use best one (exploitation)

Examples

If rewards are deterministic and known

- policy: pull arm with highest reward (exploitation)

If rewards are deterministic and unknown

- policy: try each arm (exploration), then use best one (exploitation)

If rewards are random and known

- policy: take action with highest expected reward

Examples

If rewards are deterministic and known

- policy: pull arm with highest reward (exploitation)

If rewards are deterministic and unknown

- policy: try each arm (exploration), then use best one (exploitation)

If rewards are random and known

- policy: take action with highest expected reward

If rewards are random and known

- policy: Explore by trying each arm 10,000 times to estimate the rewards, and then exploit. But here exploration is too long and pre-determined.

Action-value methods

Methods that learn action-value estimates and nothing else.

$$Q_t(a) \approx q_*(a), \quad \forall a$$ ***action-value estimates***

For example, estimate action values as sample averages:

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbf{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbf{1}_{A_i=a}}$$

The exploration/exploitation dilemma

Suppose you form estimates

$$Q_t(a) \approx q_*(a), \quad \forall a$$

action-value estimates

Define the greedy action at time *t* as

$$A_t^* \doteq \arg\max_a Q_t(a)$$

If $A_t = A_t^*$ then you are exploiting
If $A_t \neq A_t^*$ then you are exploring

You can't do both, but you need to do both

You can never stop exploring, but maybe you should explore less with time. Or maybe not.

ε-greedy action selection

In greedy action selection, you always exploit

In ε-greedy, you are usually greedy, but with probability ε you instead pick an action at random (possibly the greedy action again)

This is perhaps the simplest way to balance exploration and exploitation

ε-greedy action selection

A simple bandit algorithm

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Repeat forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

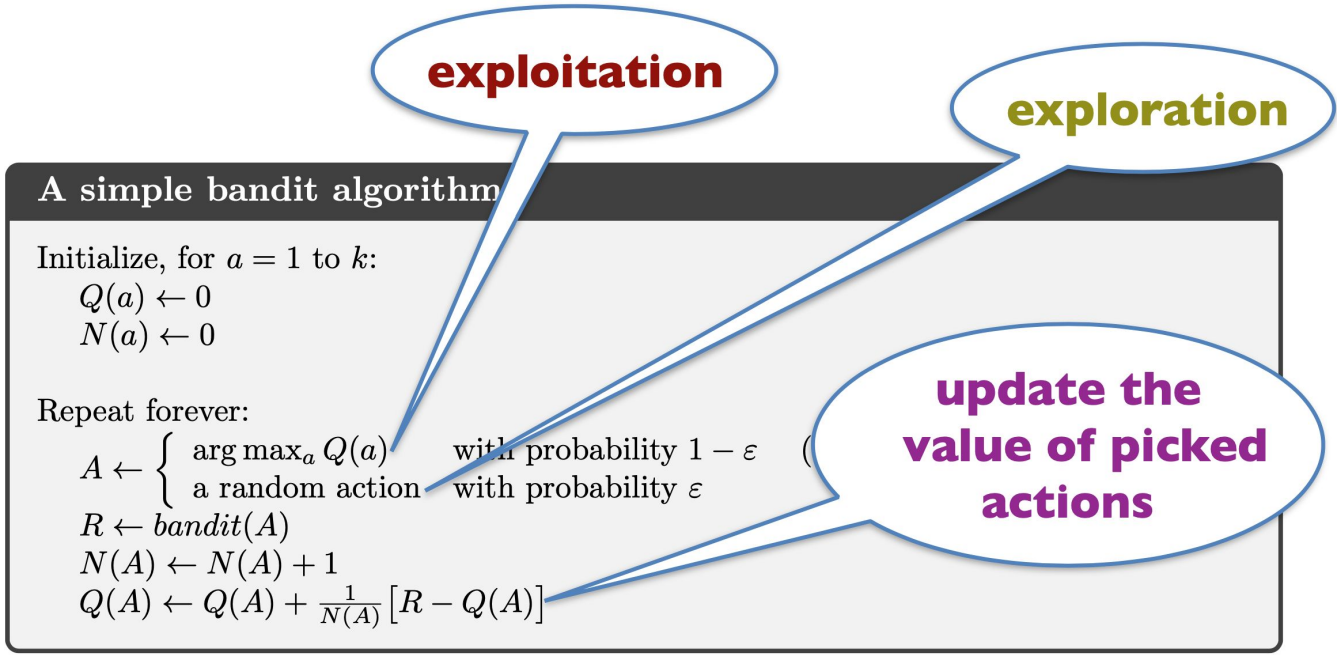
$R \leftarrow bandit(A)$

$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\big[R - Q(A)\big]$

Exploration is needed because there is always uncertainty about the accuracy of the action-value estimates.

ε-greedy action selection



Exploration is needed because there is always uncertainty about the accuracy of the action-value estimates.

Incremental Implementation

To simplify notation, let us focus on one action

- We consider only its rewards, and its estimate after $n-1$ rewards

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

How can we do this incrementally (without storing all the rewards)?

$$\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\
&= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\Big(R_n + (n-1)Q_n\Big) \\
&= \frac{1}{n}\Big(R_n + nQ_n - Q_n\Big) \\
&= Q_n + \frac{1}{n}\Big[R_n - Q_n\Big],
\end{aligned}$$

From Averaging to Learning Rule

To simplify notation, let us focus on one action

- We consider only its rewards, and its estimate after $n-1$ rewards

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

How can we do this incrementally (without storing all the rewards)?

We can store a running sum and count (and divide), or equivalently:

$$Q_{n+1} = Q_n + \frac{1}{n}\Big[R_n - Q_n\Big]$$

This is a standard form for learning/update rules we will frequently use

$$NewEstimate \leftarrow OldEstimate \; + \; StepSize \; \Big[Target - OldEstimate\Big]$$

Standard stochastic approximation convergence condition

Sometimes it is convenient to vary the step-size parameter from step to step.

$\alpha_n(a)$: the step-size parameter used to process the reward received after the n-th selection of action a.

If $\alpha_n(a) = \frac{1}{n}$, it results in sample average, which will converge to the true action value by law of large numbers.

But of course, convergence is not guaranteed for all choices of $\alpha_n(a)$

A well-known result in stochastic approximation theory gives us the conditions required to assure convergence with probability 1:

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty \qquad \text{and} \qquad \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty$$

The first condition is required to guarantee that the steps are large enough to eventually overcome any initial conditions or random fluctuations. The second condition guarantees that eventually the steps become small enough to assure convergence.

Standard stochastic approximation convergence condition

Sometimes it is convenient to vary the step-size parameter from step to step.

$\alpha_n(a)$: the step-size parameter used to process the reward received after the n-th selection of action a.

If $\alpha_n(a) = \frac{1}{n}$, it results in sample average, which will converge to the true action value by law of large numbers.

But of course, convergence is not guaranteed for all choices of $\alpha_n(a)$

A well-known result in stochastic approximation theory gives us the conditions required to assure convergence with probability 1:

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty \qquad \text{and} \qquad \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty$$

$\alpha_n \doteq \frac{1}{n}$ Yes; $\alpha_n \doteq \frac{1}{n^2}$ No, because it is too small.

if $\alpha_n \doteq n^{-p}, \quad p \in (0,1)$

then convergence is at the optimal rate:

$O(1/\sqrt{n})$

Optimistic initial values to Encourage Exploration

All methods so far depend on $Q_1(a)$
So far we have used $Q_1(a) = 0$

Suppose we initialize the action values *optimistically* ($Q_1(a) = 5$), we can encourage models to try all the arms.

Upper Confidence Bound (UCB) action selection

- ε-greedy action selection forces the non-greedy actions to be tried, but indiscriminately, with no preference for those that are nearly greedy or particularly uncertain.
- It would be better to select among the non-greedy actions according to their potential for actually being optimal, taking into account both **how close their estimates are to being maximal** and **the uncertainties in those estimates**.
- One effective way of doing this is to select actions according to the upper confidence bound:

$$A_t \doteq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}} \right]$$

  - Estimate an upper bound on the true action values
  - Select the action with the largest estimated upper bound
  - A clever way of reducing exploration over time

Action-Value vs Numerical Preference

We consider learning a **numerical preference** for each actio $H_t(a) \in \mathbb{R}$
The action probabilities follows softmax distribution:

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a),$$

This is similar to a classification problem where classes are actions.
Then, we can use stochastic gradient descent :)

Gradient Bandit Algorithm

Then, we can use stochastic gradient descent :)

On each step, after selecting action A_t and receiving the reward R_t

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha\big(R_t - \bar{R}_t\big)\big(1 - \pi_t(A_t)\big), \qquad \text{and}$$
$$H_{t+1}(a) \doteq H_t(a) - \alpha\big(R_t - \bar{R}_t\big)\pi_t(a), \qquad \text{for all } a \neq A_t,$$

where $\alpha > 0$ is a step-size parameter, and $\bar{R}_t \in \mathbb{R}$ is the average of the rewards up to but not including time t (with $\bar{R}_1 \doteq R_1$), which can be computed incrementally as described

The $\bar{R}_t$ term serves as a baseline with which the reward is compared.

If the reward is higher than the baseline, then the probability of taking A_t in the future is increased, and if the reward is below baseline, then the probability is decreased.
The non-selected actions move in the opposite direction.

Derivation of gradient-bandit algorithm

The Bandit Gradient Algorithm as Stochastic Gradient Ascent

One can gain a deeper insight into the gradient bandit algorithm by understanding it as a stochastic approximation to gradient ascent. In exact *gradient ascent*, each action preference $H_t(a)$ would be incremented in proportion to the increment's effect on performance:

$$H_{t+1}(a) \doteq H_t(a) + \alpha \frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)}, \tag{2.13}$$

where the measure of performance here is the expected reward:

$$\mathbb{E}[R_t] = \sum_x \pi_t(x) q_*(x),$$

and the measure of the increment's effect is the *partial derivative* of this performance measure with respect to the action preference. Of course, it is not possible to implement gradient ascent exactly in our case because by assumption we do not know the $q_*(x)$, but in fact the updates of our algorithm (2.12) are equal to (2.13) in expected value, making the algorithm an instance of *stochastic gradient ascent*. The calculations showing this require only beginning calculus, but take several

steps. First we take a closer look at the exact performance gradient:

$$\begin{aligned}\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)}\left[\sum_x \pi_t(x) q_*(x)\right] \\ &= \sum_x q_*(x) \frac{\partial \pi_t(x)}{\partial H_t(a)} \\ &= \sum_x \big(q_*(x) - B_t\big) \frac{\partial \pi_t(x)}{\partial H_t(a)},\end{aligned}$$

where B_t, called the *baseline*, can be any scalar that does not depend on x. We can include a baseline here without changing the equality because the gradient sums to zero over all the actions, $\sum_x \frac{\partial \pi_t(x)}{\partial H_t(a)} = 0$. As $H_t(a)$ is changed, some actions' probabilities go up and some go down, but the sum of the changes must be zero because the sum of the probabilities is always one.

Next we multiply each term of the sum by $\pi_t(x)/\pi_t(x)$:

$$\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} = \sum_x \pi_t(x)\big(q_*(x) - B_t\big) \frac{\partial \pi_t(x)}{\partial H_t(a)} / \pi_t(x).$$

The equation is now in the form of an expectation, summing over all possible values x of the random variable A_t, then multiplying by the probability of taking those values. Thus:

$$
\begin{aligned}
&= \mathbb{E}\left[\big(q_*(A_t) - B_t\big) \frac{\partial \pi_t(A_t)}{\partial H_t(a)} / \pi_t(A_t) \right] \\
&= \mathbb{E}\left[\big(R_t - \bar{R}_t\big) \frac{\partial \pi_t(A_t)}{\partial H_t(a)} / \pi_t(A_t) \right],
\end{aligned}
$$

where here we have chosen the baseline $B_t = \bar{R}_t$ and substituted R_t for $q_*(A_t)$, which is permitted because $\mathbb{E}[R_t | A_t] = q_*(A_t)$. Shortly we will establish that $\frac{\partial \pi_t(x)}{\partial H_t(a)} = \pi_t(x)\big(\mathbb{1}_{a=x} - \pi_t(a)\big)$, where $\mathbb{1}_{a=x}$ is defined to be 1 if $a = x$, else 0. Assuming that for now, we have

$$
\begin{aligned}
&= \mathbb{E}\big[\big(R_t - \bar{R}_t\big) \pi_t(A_t) \big(\mathbb{1}_{a=A_t} - \pi_t(a)\big) / \pi_t(A_t) \big] \\
&= \mathbb{E}\big[\big(R_t - \bar{R}_t\big) \big(\mathbb{1}_{a=A_t} - \pi_t(a)\big) \big] .
\end{aligned}
$$

Recall that our plan has been to write the performance gradient as an expectation of something that we can sample on each step, as we have just done, and then update on each step in proportion to the sample. Substituting a sample of the expectation above for the performance gradient in (2.13) yields:

$$
H_{t+1}(a) = H_t(a) + \alpha \big(R_t - \bar{R}_t\big) \big(\mathbb{1}_{a=A_t} - \pi_t(a)\big), \qquad \text{for all } a,
$$

which you may recognize as being equivalent to our original algorithm (2.12).

Thus it remains only to show that $\frac{\partial \pi_t(x)}{\partial H_t(a)} = \pi_t(x)\big(\mathbb{1}_{a=x} - \pi_t(a)\big)$, as we assumed. Recall the standard quotient rule for derivatives:

$$\frac{\partial}{\partial x}\left[\frac{f(x)}{g(x)}\right] = \frac{\frac{\partial f(x)}{\partial x} g(x) - f(x)\frac{\partial g(x)}{\partial x}}{g(x)^2}.$$

Using this, we can write

$$\begin{aligned}
\frac{\partial \pi_t(x)}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)} \pi_t(x) \\
&= \frac{\partial}{\partial H_t(a)} \left[\frac{e^{H_t(x)}}{\sum_{y=1}^k e^{H_t(y)}}\right] \\
&= \frac{\frac{\partial e^{H_t(x)}}{\partial H_t(a)} \sum_{y=1}^k e^{H_t(y)} - e^{H_t(x)} \frac{\partial \sum_{y=1}^k e^{H_t(y)}}{\partial H_t(a)}}{\left(\sum_{y=1}^k e^{H_t(y)}\right)^2} && \text{(by the quotient rule)} \\
&= \frac{\mathbb{1}_{a=x} e^{H_t(x)} \sum_{y=1}^k e^{H_t(y)} - e^{H_t(x)} e^{H_t(a)}}{\left(\sum_{y=1}^k e^{H_t(y)}\right)^2} && \text{(because } \tfrac{\partial e^x}{\partial x} = e^x\text{)} \\
&= \frac{\mathbb{1}_{a=x} e^{H_t(x)}}{\sum_{y=1}^k e^{H_t(y)}} - \frac{e^{H_t(x)} e^{H_t(a)}}{\left(\sum_{y=1}^k e^{H_t(y)}\right)^2} \\
&= \mathbb{1}_{a=x} \pi_t(x) - \pi_t(x)\pi_t(a) \\
&= \pi_t(x)\big(\mathbb{1}_{a=x} - \pi_t(a)\big). && \text{Q.E.D.}
\end{aligned}$$

Summary comparison of bandit algorithms

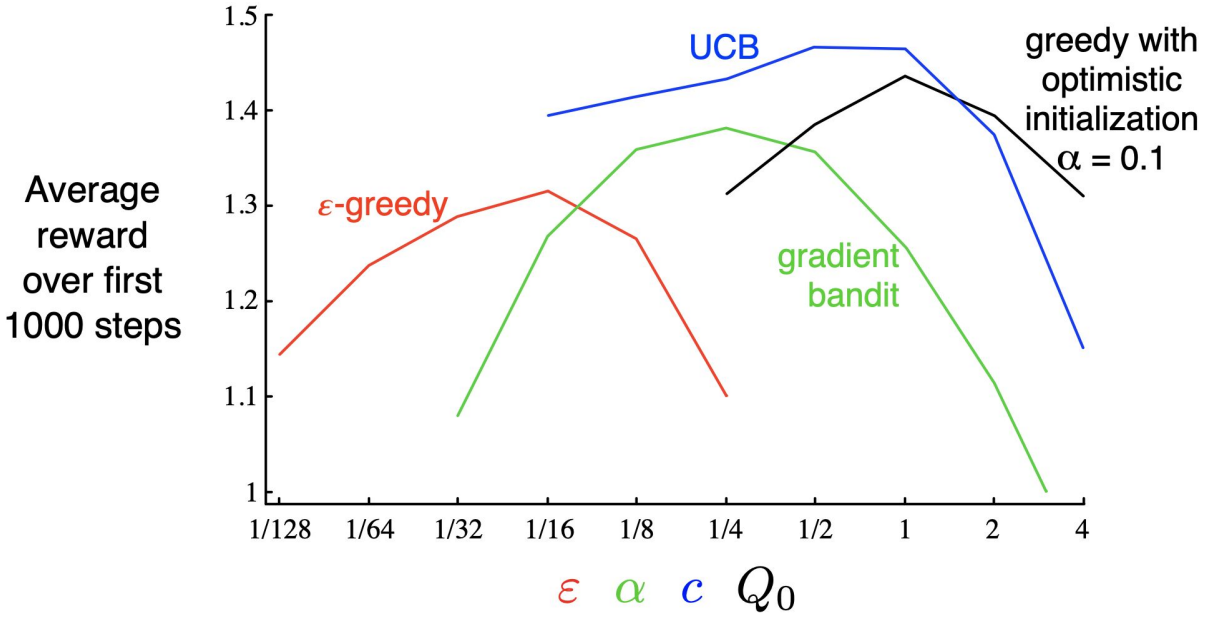


Figure 2.6: A parameter study of the various bandit algorithms presented in this chapter. Each point is the average reward obtained over 1000 steps with a particular algorithm at a particular setting of its parameter.