# K-armed Bandit Problems

Diego Ascarza-Mendoza

Escuela de Gobierno y Transformación Pública

# Outline

- The focus of our course will be on ’Tabular Solution Methods’. We only have a quarter...
- I expect you though to study on your own ’Approximate methods’ if you ever want to use this for something more meaningful.
- Tabular methods are useful when state and action spaces are small enough for the approximate VF to be represented as tables.
- In this case, we can often find exact solutions most of the time. While the scope of these methods may be limited, we gain a deeper understanding of the core ideas of RL.

- Remember again, in RL we use training information that evaluates actions rather than instructs by giving correct actions $\rightarrow$ need for exploration.
- In this session, we learn about a special case of a RL problem where there is a single state. This problem is called *bandit problems*.
- This is a non-associative setting with an evaluative feedback problem.

**Goals:**

1. Introduce basic learning methods.
2. Take a step closer to the full RL problem with a single-state problem.

# K-armed Bandit Problem

- You are faced repeatedly with a choice among $k$ different options or actions.
- After each choice, you receive a numerical reward chosen from a stationary probability distribution that depends on your chosen action.
- The objective is to maximize the expected total reward over a specified time period or a set of steps.
- Each action selection is like a play of one of those slot machine's levers, and the rewards are the payoffs from hitting the jackpot.
- You can also think of this as a doctor choosing several treatments and each reward is the survival or well-being of the patient.

- Each of the $k$ actions has an expected or mean reward given that that action is selected.
- We call this the *value* of that action.
- Denote the action selected on time step $t$ by $A_t$, and the corresponding reward as $R_t$.
- Then, the value of an arbitrary action $a$ is denoted by $q_*(a)$ and is given by:

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

- If you knew $q_*(a)$ the problem is boring. We assume then that we don't know the value of actions. We have to estimate them!
- We denote the estimated value of action $a$ in time step $t$ as $Q_t(a)$. We want $Q_t(a)$ to be **as close as possible** to $q_*(a)$

# Greedy actions

- At any time step, there is at least one action whose estimated value is the greatest.
- We call these the *greedy* actions.
- When we select these actions, we say that we are *exploiting* our current knowledge of the values of the actions.

# Exploration

- When we select nongreedy actions, we say that we are *exploring*.
- Exploring allows you to improve your estimate of the nongreedy action's value.
- Exploitation $\rightarrow$ maximize expected reward in one step.
- Exploration may produce the greater total reward in the long run. If you have the opportunity to play multiple times, it is beneficial to explore.
- Exploration $\rightarrow$ low return in the short run but higher in the long run (discover better actions).
- You **CAN NOT** exploit and explore at the same time: dilemma between exploitation and exploration.

# Exploration and exploitation

- Whether it is better to explore or exploit depends on a lot of stuff ...
- There are many sophisticated methods for balancing them for particular mathematical formulations.
- For the problems we attack, these methods usually make assumptions that we can not verify.
- Therefore, we worry only about balancing them at all. This need for balance is something that arises in RL.

# Outline

- We want to estimate values of actions and make action selection decisions. We call these methods *action-selection methods*.
- We take the mean reward of an action when selected as its true value.
- A natural way to estimate this is by averaging rewards actually received:

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

- If the denominator is zero, $Q_t(a)$ takes some default value, such as 0.
- By LLN, when the denominator tends to infinity, $Q_t(a) \to q_*(a)$.
- This is the *sample-average* method for estimating action values.

## Greedy action selection

- The simplest action selection rule is to select one of the actions with the highest estimated value.
- That is: choose the greedy actions, or if you have more than one, choose randomly one of them.
- The greedy action selection method is written as:

$$A_t \equiv arg\max_a Q_t(a), \tag{3}$$

- Greedy action selection always exploits current knowledge to maximize immediate reward.

# $\epsilon$-GREEDY METHODS

- Greedy selection spends no time at all sampling apparently inferior actions to see if they might really be better.
- A simple alternative is to behave greedily ’most of the time’, but with probability $\epsilon$, instead select randomly and independently from among all the rest of the options.
- This near-greedy action selection rules are called $\epsilon$-*greedy methods*.
- What is the nice thing about this? In the limit, every action will be sampled infinite number of times.
- This guarantees that $Q_t(a) \rightarrow q_*(a)$.
- This implies that the probability of selecting an optimal action converges to greater than $1 - \epsilon$ (near certainty).

# Outline

# The 10-armed Testbed

- Let's assess the effectiveness of greedy and $\epsilon$-greedy action-value methods with a numerical example.
- Suppose we generate 2000 randomly generated $k$-armed bandit problems with $k = 10$.
- For each problem, action values $q_*(a)$, $a = 1, ..., 10$, were selected from Gaussian distribution with mean $0$ and variance 1.
- When the learning method is applied by selecting action $A_t$ at time step $t$, $R_t$, was selected from a normal distribution with mean $q_*(A_t)$ and variance 1.
- We measure the performance of any method by examining how its behavior improves with experience over $1000$ time steps when applied to one of the bandit problems (this is a run).
- Repeat this for $2000$ independent runs, each with a different bandit problem.
- What do we get?

# Plot of performance

## Greedy method vs $\epsilon$-greedy methods

- We can see that at the very beginning, the greedy method improved slightly faster than the others.
- However, the greedy method performs significantly worse in the long run than the other methods (stuck with suboptimal methods).
- This can also be observed when examining the fraction of times that the greedy method identifies the optimal action.
- The higher $\epsilon$, the more exploration there is, and it usually finds the optimal action earlier.
- The advantage of $\epsilon$-greedy over greedy methods depends on the task. The higher the variance of rewards, the greater the advantage (the greater the need for exploration). RL requires a balance between exploration and exploitation.

# Outline

# Incremental Implementation

- Action-value methods estimate action values as sample averages of observed rewards.
- It would be costly to store every single observed reward. There are more efficient ways of computing this.
- Concentrate in a single action. Let $R_i$ be the reward received after the $i$th selection of this action.
- Let $Q_n$ denote the estimate of its action value after it has been selected $n-1$ times:

$$Q_n = \frac{R_1 + R_2 + ... + R_{n-1}}{n-1},$$

- We don't have to keep track of all the history of returns:

$$\begin{aligned} Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} r_i = \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\ &= \frac{1}{n}\left(R_n + (n-1)\frac{1}{(n-1)}\sum_{i=1}^{n-1} R_i\right) \\ &= Q_n + \frac{1}{n}\left[R_n - Q_n\right], \end{aligned} \tag{4}$$

# Incremental Implementation

- This updating rule will be used through the course. The general form is:

$$\text{New Estimate} \leftarrow \text{Old Estimate} + \text{Step Size} \underbrace{[\text{Target} - \text{Old Estimate}]}_{error}, \tag{5}$$

- Target is presumed to indicate a desirable direction in which to move, but it can be noisy.
- Notice that the step-size parameter used in the incremental method $\frac{1}{n}$ changes from time step to time step.
- We will denote the step-size parameter by $\alpha$ or $\alpha_t(a)$.

# Algorithm

- **Initialize:** For each action $a = 1$ to $k$, set:
  - $Q(a) \leftarrow 0$ (Estimated reward)
  - $N(a) \leftarrow 0$ (Action count)
- **Loop forever:**

$$A \leftarrow \begin{cases} \arg\max_a Q[a] & \text{with probability} \quad 1-\epsilon \quad \text{break ties randomly} \\ \text{a random action} & \text{with probability} \quad \epsilon \end{cases}$$

- R $\leftarrow$ $bandit(A)$.
- Update:
  - $N[A] \leftarrow N[A] + 1$
  - $Q[A] \leftarrow Q[A] + \frac{1}{N[A]}(R - Q[A])$

# Outline

# Tracking a Nonstationary Problem

- The methods discussed before are appropriate for stationary bandit problems.
- By stationary, we mean problems where reward probabilities do not change over time.
- If problems are nonstationary, it makes sense to give more weight to recent rewards than to long-past rewards.
- One way of doing this is by having a constant step-size parameter $\alpha$:

$$Q_{n+1} \doteq Q_n + \alpha \left[R_n - Q_n\right], \tag{6}$$

This will result in $Q_{n+1}$ being a weighted average of past rewards and the initial estimate $Q_1$:

$$Q_{n+1} = Q_n + \alpha \left[R_n - Q_n\right] = \alpha R_n + (1-\alpha)Q_n$$

By using the expression for $Q_n$:

$$= (1-\alpha)^n Q_1 + \sum_{1=1}^{n} \alpha(1-\alpha)^{n-i} R_i. \tag{7}$$

# Outline

# Optimistic Initial Values

- You can clearly see that all the discussed methods depend on $Q_1(a)$.
- Usually, this does not bring serious bias problems.
- Downside is that initial estimates must be picked by the user.
- Upside is that they provide an easy way to supply some prior knowledge about what level of rewards can be expected. They can also **encourage exploration!**.
- Suppose we are very optimistic and set $Q_1(a) = +1$. Let's compare $\epsilon$-greedy method with $Q_1(a) = 0$ with greedy method and the optimistic initial value.
- We can show that in this case, greedy method works better because of disappointment and need for exploration.
- It is far from being useful approach though in nonstationary problems. If task changes, this methods cannot help. Beginning of time occurs only once! This critique apply to sample-average methods as well that treats beginning of time as special event.

# Outline

# Upper-Confidence-Bound Action Selection

- Exploration is needed because there is uncertainty about accuracy of action-value estimates.
- While $\epsilon$-greedy action helps to explore, it would be nice to select among non-greedy actions according to their potential of being optimal.
- This can be done by taking into account how close their estimates are to being maximal and the uncertainties in those estimates:

$$A_t \doteq \arg\max_a \left[ Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right], \tag{8}$$

- Expression in square root is measure of uncertainty in the variance of our estimates.
- $c$ determines the confidence level.
- The more time has passed (t) and the fewer times an action has been chosen $N_t(a)$, the more uncertainty there is.

# Upper-Confidence-Bound Action Selection

- UCB methods perform well but it is more difficult than $\epsilon$-greedy to extend beyond bandits to the more general RL problem.
- These methods are usually hard to employ in nonstationary problems or where there are large state spaces and approximations are needed.

# Gradient Bandit Algorithms

- We have studied methods that estimate action values and use those estimates to select actions.
- There is another way of doing this. Now we consider learning a numerical *preference* for each action $a$, which we denote $H_t(a)$.
- The larger the preference, the more often that action is taken. The preference has no interpretation!.
- All that matters is the relative preference of one action over another.
- Preference are determined according to a soft-max (Gibbs) distribution:

$$Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a), \tag{9}$$

- We initialize this algorithm with $H_1(a) = 0 \ \forall a$.
- The natural learning algorithm for this setting is based on the idea of stochastic gradient ascent:

$$\begin{aligned} H_{t+1}(A_t) &\doteq H_t(A_t) + \alpha(R_t - \overline{R}_t)(1 - \pi_t(A_t)), \quad \text{and} \\ H_{t+1}(a) &\doteq H_t(a) - \alpha(R_t - \overline{R}_t)\pi_t(a), \quad \forall a \neq A_t, \end{aligned} \tag{10}$$

- $\alpha$ is a step-size parameter, $\overline{R}_t \in \mathbb{R}$ is the average of all rewards through and including time $t$ (benchmark).
- If reward is higher than baseline, the changes of taking $A_t$ in the future is increased. Non selected actions move in the opposite.

# Outline

- We have considered sofar only nonassociative tasks: no need to assocaite different actions with different situations.
- In these settings, we just try to find a single best action if task is stationary or track the best one if we are in a nonstationary problem.
- In more general RL problems, there is more than one situation and what we want to learng is a **policy**.
- A policy is a mapping from situations to actions that are best in those situations.

# Contextual Bandits

- Imagine for instance there are $K$-armed bandit tasks and that on each step you confront one of these chosen at random.
- Imagine also that when a bandit task is selected, you are given some clue about its identity (but not its action values).
- Now you can learn a policy associating each task, signaled, for instance, by the color you see.
- With a policy you can do much better. What we described is an example of *associative search* task.
- What is going to be different from the associative k-armed bandit relative to the full RL problem is that here actions only affect the present.

# Summary

- There are different ways to balance exploration and exploitation.
- $\epsilon$-greedy methods choose randomly a small fraction of the time, and UCB methods choose deterministically but achieve exploration by looking at potential.
- Gradient bandit algorithms estimate preferences instead of action values.
- Optimistic initial values helps a lot the greedy method.
- Hard to compare these methods as they depend on parameters. However, it seems that UCB can win in a lot of situations.