Comparison between Action-Value Method and Gradient Bandit Algorithm for k-armed Bandit Problem

Minheng Chen (Student no. 09020119)

***Abstract*—In this article, I design some algorithm for soving the *k*-armed bandit problem, and make some experiments on the stationary case and nonstationary case respectively. We compare the performance of the action-value method and the gradient bandit algorithm. Besides, a fixed step size algorithm based on action-value method is implemented for nonstationary case. I find that the gradient bandit algorithm is much more suitable for stationary case and the action-value method performs much better on the nonstationary case. What's more, the fixed step size ϵ-greedy method performs quite well in non-steady-state scenarios,and we do believe there is still a lot of room for improvement in these algorithms**

I. Problem Statement

Stationary Case. Consider a 10-armed bandit problem, where the action set is denoted as $\mathcal{A} := \{1, 2, \cdots, 10\}$. Suppose that the true value of each action is set as

$$q_*(a) = \xi, \quad a \in \mathcal{A},$$

where $\xi \sim \mathcal{N}(0, 1)$. At time t, the reward R_t of selecting action a is generated according to a normal distribution $\mathcal{N}(q_*(a), 2)$.

Design two algorithms for solving this problem, one is based on the action-value method and the other one is based on the gradient bandit algorithm. The pseudocode of your algorithm should be given. Compare the performance of your two algorithms by numerical simulation results. Some discussion on the results are required.

Nonstationary Case. Consider a 10-armed bandit problem, where the action set is denoted as $\mathcal{A} := \{1, 2, \cdots, 10\}$. Initially, at $t = 0$, the true value of each action is set as

$$q_*(a, 0) = \xi, \quad a \in \mathcal{A},$$

At time t, the true reward R_t of selecting action a is generated according to a normal distribution $\mathcal{N}(q_*(a, t), 2)$. The true value of each action is updated as

$$q_*(a, t + 1) = q_*(a, t) + \eta, \quad a \in \mathcal{A},$$

where $\eta \sim \mathcal{N}(0, 0.1)$.

Design two algorithms for solving this problem, one is based on the action-value method and the other one is based on the gradient bandit algorithm. The pseudocode of your algorithm should be given. Compare the performance of your two algorithms by numerical simulation results. Some discussion on the results are required.

Further Discussion. Try to modify your previous algorithms with improved performance. Some necessary illustration and numerical simulation results should be given.

II. Algorithm Design

A. Overview

The *k*-armed bandit problem has incited interest in economic applications, for it captures an intriguing trade-off between information acquisition (experimenting to find the option that pays the most) and exploitation (choosing the option that pays the most according to current knowledge). It has been used to model economic problems such as job search, consumer behavior and market pricing, research and development, adoption of new technology, and collective experimentation (Strulovici, 2010). The classic bandit model assumes that the decision maker (DM) has a unique prior belief about the payoff distributions. However, the model is often used to study the optimal experimentation with a novel option about which the DM has little information. Thus, the classic unique-prior assumption can be too strong in many cases. For example, workers searching for a new job might not know the exact distribution of the matching quality (Nishimura and Ozaki, 2004); or farmers considering for a new technology might not know the distribution of its productivity.

This classic problem epitomizes a core trade-off in online learning and reinforcement learning more broadly: should we explore (exploration) to try new possibilities, or should we (exploitation) stick to the best known choice so far? ?In the multi-armed bandit problem, exploring means playing slot machines that you haven't played yet, which may cost you too much time and money on machines that don't pay well; The machine with the best yield, which in turn may cost you the chance to find a better machine. And similar choices can be seen everywhere in daily life: when you go to a restaurant, do you still struggle with whether to order a familiar dish or a new dish? To go to a place, do you take the old road you know or choose a new road? The trade-off between exploration and persistence is the core of online learning.

In this article, we implement two classic algorithms for solving the multi-armed bandit problem, **Gradient Bandit** and **ϵ-Greedy** method. We evaluated their performance on the above problem conditions, and modify the previous algorithm with improved performance.

B. Action-value(ϵ-greedy) Method

Let's analyze the algorithm for estimating the value of an action in detail. We use these value estimates for action selection, and this class of methods is collectively referred to as "action-value methods". As mentioned earlier, the true value of an action's value is the expected payoff from choosing this action. Therefore, a natural way is to estimate the value of actions by computing the average of the actual payoffs:

$$Q_t(a) = \frac{\sum_{i=0}^{t-1} R_i * \mathbb{1}\{A_i = a\}}{\sum_{i=0}^{t-1} \mathbb{1}\{A_i = a\}} \tag{1}$$

Among them, $\mathbb{1}\{A_i = a\}$ represents a random variable, and its value is 1 when the predicate action is a, otherwise it is 0. We define $Q_t(a)$ as some default value when the denominator is 0, such as $Q_t(a)$=0. When the denominator tends to infinity, $Q_t(a)$ will converge to $q_*(a)$ according to the law of large numbers. We refer to this method of estimating action values as a sampled averaging method, since each estimate is the average of a sample of associated returns. Of course, this is only one way of estimating action values, and it's not necessarily the best way. We continue with this simple estimation method and discuss how to use estimates to select actions.

The simplest action selection rule is to choose the action with the highest estimated value, the greedy action defined in the previous section. If there are multiple greedy actions, choose one arbitrarily, such as randomly picking. We denote this greedy action selection method as:

$$A_t(t) = arg\max_a Q_t(a) \tag{2}$$

where $arg\,max_a$ is the action a that maximizes the value of $Q_t(a)$. The greedy action of choice always maximizes immediate payoff using current knowledge. This approach simply doesn't take the time to try out obviously bad moves to see if they would actually be better. A simple alternative to the greedy policy is to behave greedy most of the time, but occasionally (say with a small probability ϵ) choose randomly from all actions with medium probability in a manner independent of the action-value estimate. We refer to methods using such nearly greedy selection rules as ϵ-greedy methods. An advantage of this type of approach is that if the time instants can be infinitely long, each action will be sampled an infinite number of times, ensuring that all $Q_t(a)$ converge to $q_*(a)$. This of course also means that the probability of choosing the optimal action will converge to be greater than $1-\epsilon$. close to a deterministic choice. However, this is only an asymptotic guarantee, and little has been said about the practical effects of such methods. The following is the pseudo-code flowchart of the algorithm.

Algorithm 1 ϵ-greedy Algorithm for *k*-armed Bandit Problem

Initialization: The reward distribution R_k, $step = 1$, ϵ, the total reward $\mathbb{R}$, action list $a(i)$, reward list $\mathbb{R}(i)$
while actions do not converge **do**
  $e \leftarrow random(0,1)$
  Get $R_i(step)$ at step
  if $e < \epsilon$ **then**
    Select an action **a** randomly
    Get action a reward$R_a(step)$
    Add to total reward $\mathbb{R} \leftarrow \mathbb{R} + R_a(step)$
    Update action list $a(step) \leftarrow$ **a**
    Update reward list $\mathbb{R}(step) \leftarrow R_a(step)$
    $step \leftarrow step + 1$
  else
    Compute current action-value for each action $V_i(step)$
    a $\leftarrow arg\,max_a\, V_i(step)(imeanstheaction)$
    Get action a reward$R_a(step)$
    Add to total reward $\mathbb{R} \leftarrow \mathbb{R} + R_a(step)$
    Update action list $a(step) \leftarrow$ **a**
    Update reward list $\mathbb{R}(step) \leftarrow R_a(step)$
    $step \leftarrow step + 1$
  end if
  if the action converge **then**
    break
  else
    Continue
  end if
end while

C. Gradient Bandit Method

We consider learning a numerical preference function $\theta_t(a)$ for each action a. The larger the preference function, the more frequently the action is chosen, but the concept of preference function is not presented in the sense of "benefit". Only the relative preference of one action over another is important, and if we add 1000 to each action's preference function, then for action probabilities determined according to the following softmax distribution (Gibbs or Boltzmann distribution) has no effect:

$$Pr\{A_t = a\} = \frac{e^{\theta_t(a)}}{\sum_{b=1}^{k} e^{\theta_t(b)}} = \pi_t(a) \tag{3}$$

where $\pi_t(a)$ is a new and important definition, which is used to represent the probability that action a is selected at time t. The initial values of all preference functions are the same (eg. $\theta_1(a) = 0, \forall a$), so each action has the same probability of being chosen.

Based on the idea of stochastic gradient ascent, a natural learning algorithm is proposed. At each step, after choosing to do A_t and earning a payoff R_t, the preference function will be updated as follows:

$$\theta_{t+1}(A_t) = \theta_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A_t)), \tag{4}$$

$$\theta_{t+1}(a) = \theta_t(a) - \alpha(R_t - \bar{R}_t)(\pi_t(A_t)), \text{for all } a \neq A_t \tag{5}$$

where α is a number greater than 0, indicating the step size. $R \in \mathbb{R}$ is the average of all returns at time t, which can be calculated step by step. The $\bar{R}_t$ item serves as a benchmark item for comparing benefits. If the payoff is higher than it, then the probability of choosing action A_t in the future increases, otherwise the probability decreases and the probability of unselected actions being chosen increases. The pseudocode flowchart of the algorithm is also given below

Algorithm 2 Gradient Bandit Method for *k*-armed Bandit Problem

Initialization: The reward distribution R_i, $step = 1$, α, the total reward $\mathbb{R}$, action list $a(i)$, reward list $\mathbb{R}(i)$, estimate list $Q(i)$
while actions do not converge **do**
  Get $R_i(step)$ at step
  Select an action **a** randomly
  Compute current probability for each action $Q(step)$
  action probablity $p \leftarrow$ **softmax**$(Q(step))$
  Get action $a \leftarrow p$
  Get action a reward$R_a(step)$
  Add to total reward $\mathbb{R} \leftarrow \mathbb{R} + R_a(step)$
  Update action list $a(step) \leftarrow$ **a**
  Update reward list $\mathbb{R}(step) \leftarrow R_a(step)$
  $step \leftarrow step + 1$
  Caculate $\bar{R}_t$
  Update $Q(i)$ from Eq.4 and Eq.5
  if the action converge **then**
    break
  else
    Continue
  end if
end while

D. Further Discussion for Nonstationary Case

The averaging approach we have discussed so far is appropriate for stationary bandit problems, that is, bandit problems where the probability distribution of payoffs does not change over time. However, this method is not suitable if the probability of the slot machine's payoff is changing over time. As mentioned earlier, we often encounter non-stationary reinforcement learning problems. In this case, it is reasonable to give more weight to recent returns than to returns from the long past. One of the most popular methods is to use a fixed step size. For example, the incremental update Eq.1 for updating the mean $Q_t(a)$ of t-1 past returns can be changed to

$$Q_{t+1}(a) = Q_t(a) + \alpha * \mathbb{1}\{A_i = a\}(R_t - Q_t(a)) \tag{6}$$

where the step size parameter $\alpha \in (0, 1]$ is a constant. This makes $Q_{t+1}(a)$ a weighted average of past returns and initial estimates $Q_1(a)$

Sometimes it is convenient to vary the step size parameter step by step over time. Let $\alpha_n(a)$ denote the step size parameter used to process the reward received after the n_th choice of action a. As we note, choosing $\alpha_n(a) = \frac{1}{n}$ will result in a sampled average, which is

guaranteed to converge to the true value by the law of large numbers. However, convergence is of course not guaranteed for any $\{\alpha_n(a)\}$ sequence. A well-known result in stochastic approximation theory gives the condition needed to guarantee a convergence probability of 1:

$$\sum_{n=0}^{\infty} \alpha_n(a) = \infty \text{ and } \sum_{n=0}^{\infty} \alpha_n^2(a) < \infty \tag{7}$$

where the first condition is to ensure that there is a large enough step size to eventually overcome any initial conditions or random fluctuations. The second condition guarantees that the final step size becomes small to guarantee convergence.

Note that both convergence conditions are satisfied in the case of the sampled average $\alpha_n(a) = \frac{1}{n}$, but not in the constant step size parameter $\alpha_n(a) = \alpha$. In the latter case, the second condition is not satisfied, indicating that the estimate never fully converges, but instead varies with the most recent gains. As we mentioned earlier, this is what we want in non-stationary environments, and problems in reinforcement learning are often non-stationary in fact. Furthermore, sequences of step size parameters that satisfy the Eq.7 often converge very slowly, or require extensive tuning to obtain a satisfactory convergence rate.

III. Numerical Experiments

In this section, We will quantitatively simulate and analyze the performance of the two methods mentioned above, in the stationary and nonstationary scenarios respectively.

A. Stationary Case

Under stationary conditions, we compare the convergence speed of the algorithm and the success rate of convergence to the optimal under different parameter settings.We define convergence as the same action for ten consecutive choices.We define whether convergence is successful as judging whether it finally converges on the action with the highest reward. We ran the above-mentioned algorithms one hundred thousand times respectively and got the results shown in table III-A below.

We mainly compare the performance of the ϵ-greedy algorithm when ϵ is 0.1 0.2 0.3 0.4 and the gradient algorithm when α is 0.1 and 0.2 respectively. In addition, we also added the original greedy algorithm for comparison. The experimental results show that in the ϵ-greedy algorithm, when the value of ϵ is set larger, the number of convergence steps it needs will be more, but correspondingly, its convergence success rate will be greater, that is to say, if To get better results, we must balance the expenditure on these two strategies of explore and exploit.

Alogorithm	Parameter settings	Convergence steps	Success rate(%)
ϵ-greedy	$\epsilon = 0.1$	60.74 ± 12.98	44.4
ϵ-greedy	$\epsilon = 0.2$	82.37 ± 33.18	53.7
ϵ-greedy	$\epsilon = 0.3$	136.14 ± 82.36	63.0
ϵ-greedy	$\epsilon = 0.4$	303.69 ± 240.05	73.1
greedy	/	51.33 ± 2.15	35.1
gradient	$\alpha = 0.1$	183.58 ± 106.88	82.8
gradient	$\alpha = 0.2$	103.21 ± 55.18	75.5

TABLE I
THE RESULT OF THE EXPERIMENT ON STATIONARY CASE.

In contrast, in terms of convergence speed and success rate, the gradient-based algorithm obviously performs much better. The increase of α will result in a decrease in the number of convergence steps and a decrease in the success rate of convergence.

In general, if you want the algorithm to obtain a better convergence success rate, you often need to sacrifice its convergence speed. In addition, we found that the greedy strategy does not perform as well as the gradient-based method on the steady-state *k*-armed bandit problem.

B. Nonstationary Case

For the *k*-armed bandit problem in a non-stationary state, because the rewards for each action are different at different times, our evaluation criteria and experimental methods need to be adjusted appropriately. Because the reward is changing at every step, we cannot find an optimal action in the nonstationary situation, and thus cannot calculate the convergence success rate of the action.We decided to evaluate the performance of the algorithms by looking at their total reward after a thousand steps.

The methods we adopt include the ϵ-greedy and gradient-based methods used in the nonstationary case, as well as action-value method with constant step parameter we mentioned in further discussion, where β denotes the constant step size.

Alogorithm	Parameter settings	Total reward
greedy	/	1817.42±1383.05
ϵ-greedy	$\epsilon = 0.0625$	2604.24±1213.98
ϵ-greedy	$\epsilon = 0.1$	2567.25±1154.37
ϵ-greedy	$\epsilon = 0.2$	2344.10±1072.42
gradient	$\alpha = 0.1$	2539.87±1376.43
gradient	$\alpha = 0.2$	2483.40±1426.34
ϵ-greedy(constant step)	$\epsilon = 0.0625$ β=0.1	2803.16±1045.88
ϵ-greedy(constant step)	$\epsilon = 0.1$ β=0.1	2794.42±1019.48
ϵ-greedy(constant step)	$\epsilon = 0.2$ β=0.1	2599.44±974.62

TABLE II
THE RESULT OF THE EXPERIMENT ON NONSTATIONARY CASE.

The result is shown in table III-B, and all results are the mean of 10,000 replicate experiments. In general, in an nonstationary situation, the effect of the action-value method with a fixed step parameter is far better than other methods, but it is also necessary to pay attention to the defect that this method is difficult to converge, that is to say, try not to use it in Use this method in a stationary scene.

IV. Conclusion

In this article, I design some algorithm for soving the *k*-armed bandit problem, and make some experiment on the stationary case and nonstationary case respectively. We compare the performance of the action-value method and the gradient bandit algorithm. Besides, a fixed step size algorithm based on action-value method is implemented for nonstationary case.