Infinitely-Armed Bandit Algorithms

Matthew Faulkner **Jon Krause** **Daniel Rosenberg**

Abstract

We have been considering a variation on the standard K-armed bandit problem where the number of arms is infinite. Here we present our progress and plans for the future.

1 Background

In many situations, the number of arms on a K-armed bandit is actually very large. For example, in the case of choosing an ordering of N ads, the number of possible arms to pull is actually $N!$, which is far too large for any standard finite-armed bandit algorithm to handle. Generalizing, it may be the case that the number of arms on a bandit is actually infinite. Examples of this include optimizing parameters of some algorithm or classifier and maximizing a noisy function over a domain containing an infinite number of points, such as a unit hypercube.

The algorithms designed to handle an infinite-armed bandit can be partitioned into those algorithms which merely discretize the domain into a finite number of points, and those which actually take into consideration every point. One natural question to ask is: are these infinite-armed bandit algorithms worth it, or is it more advantageous to stick with a discretization of the domain into a finite number of points and run a standard finite-armed bandit algorithm on them?

2 Algorithms

2.1 Discretization Algorithms

The simplest approach that one can take when tackling an infinite number of arms is to merely pick out some finite number of arms and run a standard finite-armed bandit algorithm on them. This raises the following questions:

- How many arms should be chosen?
- How should the arms be chosen?

The answer to the first question is algorithm- and problem-specific. For example, consider the case where the reward function $r(x) = x$ defined on the interval $[0, 1]$. With no noise, the ϵ-greedy strategy will converge to the correct arm immediately after sampling each arm once, and thus it is beneficial to choose a very large number of arms. A UCB1 or Exp3 strategy, though, will take significantly longer to converge, so if convergence time is an issue, perhaps fewer arms should be chosen at the expense of some accuracy. On the other hand, for a needle-in-a-haystack type of reward function, we clearly want to include more arms so that we have a better chance of achieving the optimal value.

Concerning how the arms themselves are chosen, two obvious strategies are to either choose them randomly or to choose them in a grid (at regular intervals in the 1D case). Choosing the arms in a deterministic way means that, for a fixed number of arms, we can construct a reward function that makes the algorithm behave terribly. Even given an arbitrary number of arms, by choosing a reward

Figure 1: Here we see a many armed bandit.

function with a maximum near an endpoint of an interval in $\mathbb{R}$, for example, we can ensure that the algorithm would never do too well. For these reasons, we opt to go with a random choice of arms.

In any case, merely discretizing the domain into a finite number of arms inctroduces a bias, as our effective hypothesis class (each of the arms) probably no longer contains the optimal hypothesis.

2.2 Zooming Algorithm

If you actually want to consider the infinite number of arms, you need to have certain information about the relationship between them. For instance, arms that are *close* should produce similar results. The Zooming Algorithm is defined to work in any example where the arms form a metric space. In each phase, certain arms are chosen to be *active*, and the algorithm chooses which arm to play from these active arms.

The Zooming Algorithm is composed of multiple phases, each of which is composed of $2^{i_{ph}}$ rounds, where i_{ph} is the current phase number. In a given round, you *activate* an arm if that arm is not covered by another arm. Each arm covers a radius defined by $r_t(v) := \sqrt{8 * i_{ph}/(2 + n_t(v))}$ where v is the active arm, and $n_t(v)$ is the number of times a given arm has been chosen at time t. Each time an arm is played, its radius shrinks, and at the beginning of each round, you *activate* arms until you have a complete covering using a *covering oracle*. This oracle can either return an uncovered arm, or state that there is no such arm. After the space is covered, you play the arm with the optimal index, defined as $I_t(v) := \mu_t(v) + 2 * r_t(v)$

A point of concern to us with this algorithm is that it does not seem to remember anything from previous phases when a new phase starts. The only piece of information it maintains is the phase number, which influeces the confidence radius and index, which changes the balance between exploration and exploitation. It seems like it may be better to start on a later phase, given knowledge of the specific problem. Aditionally, the *covering oracle* becomes very complicated once you move into more advanced spaces.

2.3 Hierarchical Optimistic Optimization

Some very informative text

3 Artificial Data Description

For now, we have just been testing the algorithms on reward functions from $\mathbb{R}$ to $\mathbb{R}$. We currently have support for polynomials of arbitrary degree, binomial functions, composing two arbitrary functions, and adding noise in the form of multiplying the unbiased result by a random number in a given range.

4 Possible Applications

One issue with bandit algorithms is that there are relatively few situations where a pure bandit algorithm is applicable. Rather, is it typically the case that there is some side information (e.g. a search term) that can be used in determining which arm to choose. In that case, this turns into a contextual bandit problem, which the algorithms we have implemented are not suited for. It might be possible to adapt these algorithms for the contextual bandit problem.

5 Conclusion

In conclusion, we did some stuff, and will do more.

6 Future Work

hgjfjjh

References

[1] Robert Kleinberg, Aleksandrs Slivkins, Eli Upfal, "Multi-Armed Bandits in Metric Spaces"