Online Knapsack Problem Using Reinforcement Learning

Ilies Ghanzouri
Department of Mechanical Engineering
Stanford University
ghanz@stanford.edu

Abstract

We ask whether reinforcement learning can be used to solve a classic online problem. The online knapsack problem is a generalized version of the 0-1 knapsack problem where items are revealed sequentially and a decision has to be made. An item could either be accepted into the knapsack or rejected before observing the next one. The goal is to maximize the total value of the items inside the knapsack without violating the capacity constraint. Our approach [1] uses model-free Q learning to determine the optimal policy.

1 Introduction

Reinforcement learning (RL) is an area of machine learning where an agent learns how to behave in an environment by performing an action and seeking high rewards. Large amounts exploration of the environment is required to find the optimal state action value $Q(s, a)$. RL has recently become a topic of interest due to its ability to solve problems without knowledge of the underlying dynamics. Model free approach are used over a large set of applications including control theory, natural language processing, image recognition, and medical diagnoses. A very popular model-free reinforcement learning algorithm is Q-learning which operates by applying incremental estimation to the Bellman equation :

$$Q(s,a) = Q(s,a) + \alpha\big(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big)$$

A model-free approach eliminates the need for a state transition model and generalization allows the agent to approximate the optimal action at a given state. While RL has been successful in many use cases, it depends on environments in which it can obtain informative rewards as it takes a certain policy. In fact, it may be difficult to learn more than a trivial policy in online knapsack problem unless future object sizes and values are know to some degree.

We seek to apply reinforcement learning methods to determine an optimal policy on a classic online optimization problem. Online optimization refer to optimization problems where we have no or incomplete knowledge of the future (online). These kind of problems are denoted as online problems and are seen as opposed to the classical optimization problems where complete information is assumed (offline). After performing Q-learning on the online knapsack, we compare our solution to the corresponding offline algorithm solution which is necessarily optimal as it knows the entire input in advance.

[1] Find code at https://github.com/ghanzouri/AA228-CS238-Online-Knapsack-Problem-Using-Reinforcement-Learning

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

2 Approach

2.1 Problem definition

Suppose we have knapsack with capacity S and a sequence of n items, represented as a sequence of value-size pairs $\{(v_i, s_i)\}_{i \in 1,\dots,n}$. The items arrive sequentially and an immediate decision has to be made : each item must be irrevocably accepted into the knapsack or rejected as soon as it arrives before observing the next one. The objective is to maximize the total value of the items inside the knapsack without violating the capacity constraint.

To avoid learning a trivial policy, our approach consists on having our agent learn the optimal policy over an environment. In fact, a trivial policy would not require use to use MDP solution techniques. A trivial policy might be to always place the current item in the knapsack or to wait until we see an exceptionally valuable object and skip over every other item.

2.2 RL formulation

Suppose there n incoming items arriving in the sequence and the knapsack capacity is S.

State space: At time $i \in [n]$, the agent sees the state $(i, n, v_i, s_i, c_i, S_i, R_i)$, where v_i and s_i are the value and size of the current item i, c_i the remaining capacity, S_i and R_i are the list of previously selected and rejected items up to item i.

Actions: The agent can choose to either Accept or Reject the current item. If $c_i < s_i$ then the agent must Reject the current item. Else, the action is drawn using an ϵ-greedy algorithm with decay α.

Reward: If $c_i > s_i$ and the action is Accept, then the reward is v_i, else reward is 0.

Architecture:

Q-Learning is a model-free Reinforcement Learning algorithm. Reinforcement Learning is widely used in contexts where agents need to develop a state-action policy to maximize some long-term reward accumulation. Being model-free, Q-Learning does not require building explicit representations of the transition and reward models. Q-learning involves applying incremental estimation of the action value function $Q(s, a)$. The update is derived from the action value form of the Bellman equation :

$$Q(s,a) = Q(s,a) + \alpha\big(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big)$$

where s, a, r are the state, action, and reward observed in a single example, is the learning rate, and is a discount factor. Since Q Learning is an online algorithm, it needs a way to balance exploration vs. exploitation. For that purpose, we used an ϵ-greedy algorithm. The latter method allows to maintain a constant amount of exploration, the ϵ is decayed over time with a decay factor α.

2.3 Linear programming offline formulation

As a baseline of comparison of our reinforcement learning results, we use linear programming to obtain an optimal offline policy. The optimal policy is obtained through linear programming using a modified version of Algorithm 7.10 from the textbook [1]. Given a set of n items numbered from 1 to n, each with a weight s_i and a value v_i, along with a maximum weight capacity S,
maximize $\sum_{i=a}^{n} v_i x_i$
subject to $\sum_{i=a}^{n} s_i x_i \leqslant S$ and $x_i \in \{0, 1\}$
Here x_i represents the number of instances of item i to include in the knapsack. The problem is to maximize the sum of the values of the items in the knapsack so that the sum of the weights is less or equal to the knapsack's capacity.

2.4 Greedy offline formulation

As another baseline comparison, we used an offline greedy algorithm. Given a set of n items numbered from 1 to n, each with a weight s_i and a value v_i, along with a maximum weight capacity S, we sort the items in decreasing order of value per unit of weight, such that $v_1/s_1 \geqslant \ldots \geqslant v_n/s_n$. We then proceed by inserting items into the sack if current item size is less then the remaining capacity.

3 Experiments and results

We generate an environment where a randomly-generated sequence of objects are drawn from a consistent distribution over size and value, which would allow us to make consistent comparisons. Our experiments were realised on three different distributions: Exponential, Poisson and Truncated Normal distributions. In the following results, we sample item size and value over an exponential distributions, $Exp(\lambda = 10)$ and $Exp(\lambda = 15)$ respectively. The knapsack capacity S is equals to 20. We recall that the probability density function of an exponential distribution is $f(x > 0, \lambda) = \lambda e^{-\lambda x}$ For training, we used a learning rate of $\alpha = 0.01$, a discount factor of $\gamma = 0.95$ and the number of episodes equals to $10,000$.

We begin by evaluating the performance of our Q-learning algorithm and its ability to learn, throughout episodes. We vary the exploration-tradeoff ϵ with different values. We chose a decay rate $\alpha = 0.99$. We expect that our agent learns throughout episodes and the cumulated rewards to increase. The results are presented in Figure 1. As expected, the agent manages to efficiently learn as the rewards increases through time. We notice that in Figure 1, the agent could potentially learn if the number of iteration were to increase. The Q-learning agent *does* perform better with an initial $\epsilon = 0.5$. The accumulated reward is increasingly higher than with $\epsilon = 0.3$.

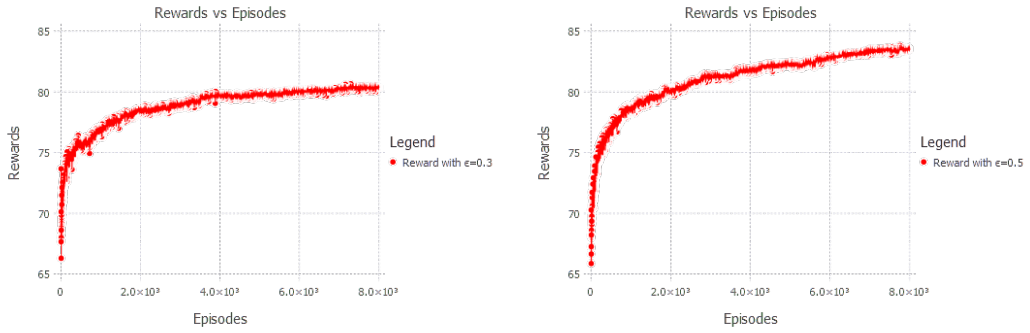

Figure 1: Accumulated reward throughout episodes with $\epsilon = 0.3$ and $\epsilon = 0.5$ respectively.

In an additional experiment, we plot and compare our learned Q-learning policy in comparison to offline solution methods as discussed in Section 2.3 and Section 2.3. Figure 2 showcases learnt policy vs the number of incoming items. As we can see, as the number of incoming items n increases, offline methods showcases that the total optimal reward increases accordingly. Our online Q-learning agent manages to mimic the increase tendency but we unfortunately notice a huge difference between online and offline policies. Other approaches such as Sarsa Lambda and Posterior Sampling were used but yielded to inferior results compared to Q-Learning. Future work are needed to better enhance our learning policy and promising methods such as Deep Q Learning are conceivable.

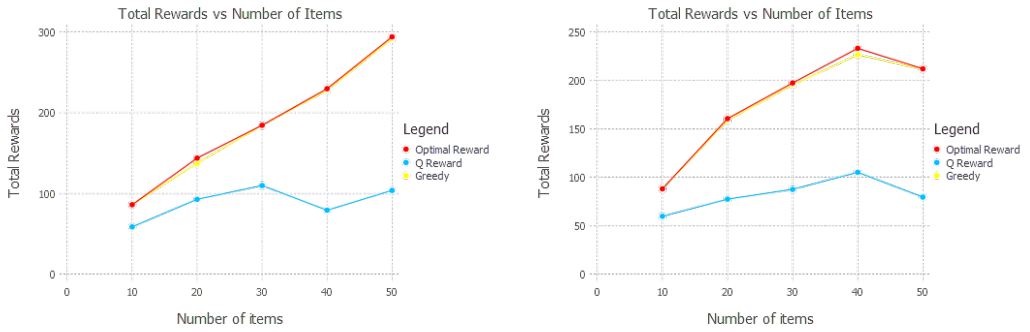

Figure 2: Optimal reward compared to our calculated Q reward and an offline greedy approach

Acknowledgments and Disclosure of Funding

I would like to thank Professor Mykel Kochenderfer and the entire AA228 course staff for providing a fantastic learning experience.

References

[1] Kochenderfer, M.J., Tim Wheeler & Kyle Wray (2020) *Algorithms for Decision Making*