Assignment 3: Solving The Bounded Knapsack Problem With Deep Reinforcement Learning

Max van den Hoven 1750461
Dean Bakens 1026726
Alwin Verbeeten 1223096
Johan de Langen 0995417

27 June 2023

Introduction

In this assignment we train deep reinforcement learning agents to perform well on the bounded knapsack problem. The bounded knapsack problem extends the classical combinatorial problem by putting limits on the number of times an item may be selected. As such, every item $i \in \{1, ..., N\}$ has weight w_i, value v_i, and limit l_i. The goal of the bounded knapsack problem is to maximize the total value of the knapsack while satisfying the item and knapsack weight limits.

The remainder of this report is structured as follows. First, we describe the bounded knapsack environment provided by OR-GYM [Hubbs et al., 2020a], characterizing the observation and action spaces, and the reward mechanism. Following this, we describe the two reinforcement learning algorithms that we use to train our agents, being A2C [Mnih et al., 2016] and PPO [Schulman et al., 2017]. Next, we describe several hyperparameters of the learning algorithms, as well as those of the neural networks that are used as function approximators. We discuss the values that we test, as well as the impact of each parameter on model performance. Finally, we interpret the results obtained after training, specifically investigating the three best agent configurations for both A2C and PPO.

Environment

The bounded knapsack problem is readily provided as a reinforcement learning environment using the OR-GYM package. The KNAPSACK-V2 environment [Hubbs et al., 2020b] provides the agent with the current state of the items and the knapsack, including item weights, values, and limits, as well as knapsack capacity and current weight. More formally, the observation space is as a vector with the following elements:

$$(w_1, ..., w_N, C, v_1, ..., v_N, W, l_1, ..., l_N, 0)\,,$$

where $w_i \in \{1, ..., 100\}$, $v_i \in \{0, ..., 100\}$, and $l_i \in \{1, ..., 10\}$ denote the weight, value, and limit of item i respectively, and $W \in \mathbb{N}$ denotes the current weight in the knapsack. C denotes the maximum capacity of the knapsack, which is fixed to 300 for this assignment. The final element (0) is a placeholder value used by OR-GYM.

Every time step, the agent must choose which element to include in the knapsack. This results in a DISCRETE action space of size 200, corresponding to the number of elements N, which is fixed to 200 for this assignment. Whenever an item i is chosen, its weight w_i is added to the current weight of the knapsack, i.e., $W \leftarrow W + w_i$. If the weight limit of the knapsack is not violated, i.e., if $W + w_i \leq C$, the value v_i of the item is emitted as the reward. Otherwise, a reward of zero is emitted and the episode is terminated.

It is important to note that we slightly modify the environment provided by OR-GYM by normalizing the observation space to the range $[0, 1]$, as this generally improves the learning ability of neural networks. To this end, we divide w_i, v_i, l_i, and W by their maximum values, which are 100, 100, 10, and C (300) respectively. Furthermore, we set C equal to zero in the emitted observation, as it provides no extra information compared to the normalized current knapsack weight. We define a hyperparameter NORMALIZE_ENV to control whether or not the environment is normalized using a GYM wrapper class. Note that we do not explicitly normalize the reward, as we expect the effect to be minimal. This assumption may be tested in a future version however.

Algorithms

To train our agents we use the Stable Baselines 3 package [Raffin et al., 2021], which readily implements many popular reinforcement learning algorithms. In this assignment, we use the A2C and PPO algorithms, as these are very widely used in practice.

A2C [Mnih et al., 2016], or Advantage Actor-Critic, combines elements of both policy-based (actor) and value-based (critic) methods. The actor network selects actions based on the current policy, while the critic network estimates the value function to evaluate the quality of the chosen actions. A2C uses the advantage function to measure the relative value of each action compared to the average expected value.

PPO [Schulman et al., 2017], or Proximal Policy Optimization, can be seen as an extension of A2C, as it introduces a clipping mechanism to prevent large policy updates, ensuring that the new policy does not deviate significantly from the old policy. In this way, PPO prevents over-exploration, meaning that more reliable steps can be taken towards a good policy.

We define a hyperparameter ALGORITHM_CLASS to control which learning algorithm is used for our agents, taking values in {A2C, PPO}. While not a traditional hyperparameter, this makes our grid search more practical.

Training and Hyperparameter Optimization

In order to find the best combination of hyperparameters for each algorithm, we perform an exhaustive grid search using a small number of candidate values for our chosen hyperparameters. Our chosen hyperparameters are summarized in Table 1, including the possible values for each parameter. Every combination of hyperparameters is trained for a maximum of 100,000 timesteps. Additionally, we perform an evaluation step every 10,000 timesteps, where the mean reward of the current agent is evaluated over 100 episodes in a separate evaluation environment. The agent with the best mean evaluation reward is saved as a checkpoint, meaning that the best model may be found before 100,000 timesteps have elapsed.

Hyperparameter	Possible Values
ALGORITHM_CLASS	{PPO, A2C}
GAMMA	{0.99, 0.9}
LEARNING_RATE	{0.0003, 0.001}
NORMALIZE_ENV	{True, False}
ACTIVATION_FN	{LeakyReLU, ReLU}
NET_ARCH	{[256, 128],[256, 256],[512, 256],[512, 256, 128]}

Table 1: Possible hyperparameters

Here, GAMMA indicates the discount factor to train the model with, which is equal to 0.99 by default. We also test a value of 0.9 to experiment with greedy agents that focus more on immediate reward, as this is an effective strategy in the classical knapsack problem when using the greedy method [Dantzig, 1957]. We take care not to set the discount factor too low however, as we limit the number of candidate values we evaluate.

The LEARNING_RATE hyperparameter controls the learning rate used to update the weights of the policy network, which is set to 0.0003 by default. We experiment with a higher learning rate to promote faster learning by performing bigger updates to the model. We again take care not to set the learning rate too high, as this could lead to exploding gradients or other failure modes during training.

The ACTIVATION_FN and NET_ARCH hyperparameters determine the architecture of the neural networks in a controlled manner. By default, the policy network provided by Stable Baselines 3 uses a 2-layer fully connected neural network with 64 neurons per layer and TanH activations. In general, ReLU and LeakyReLU activations often perform well [Goodfellow et al., 2016], so we limit the value of ACTIVATION_FN to these two functions. For NET_ARCH we test a set of larger network architectures to account for the relatively large observation and action spaces, consisting of 603 and 200 values respectively. Stable Baselines 3 automatically adds input and output layers with the correct number of neurons to correspond with the relevant spaces.

Results

Figure 1 shows the training and evaluation curves of the three best performing A2C agents in terms of the highest achieved mean evaluation reward over 100 epochs. At a glance, it seems that training is quite noisy, and that the model has not yet converged after 100,000 timesteps, as the slope has not flattened off. The best model only manages to achieve a mean evaluation reward of 1529, which is far below the desired threshold of 2000.

The best models use relatively large neural network architectures with the smaller learning rate, which possibly explains the need for more timesteps. It is not clear which discount factor or activation function works best, as there seems to be no clear connection between the two hyperparameters and agent performance. The NORAMLIZE_ENV hyperparameter is omitted, as configurations without normalization often failed to converge properly.

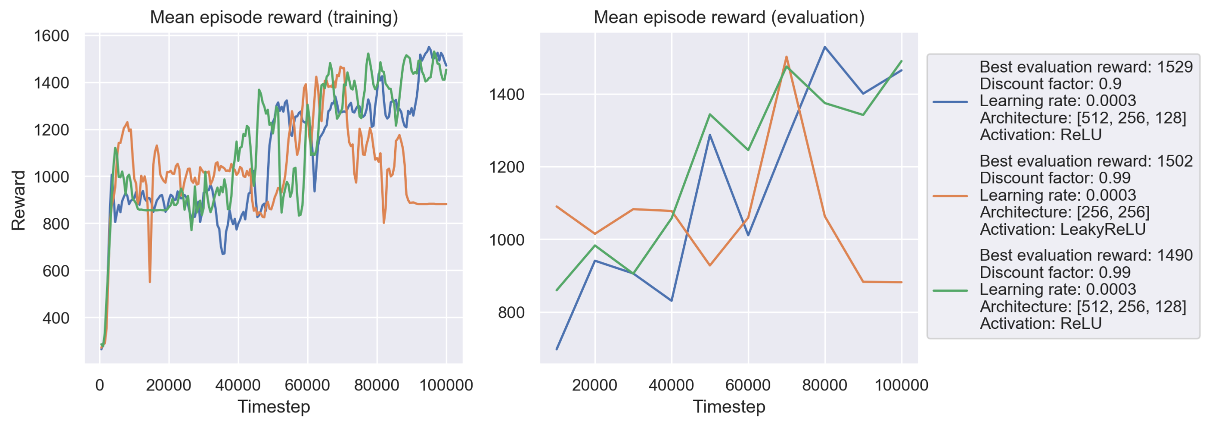

Figure 1: Learning curves for the best three models using the A2C algorithm, sorted in decreasing order of the highest achieved mean evaluation reward

The results for the PPO algorithm are shown in Figure 2, in the same fashion as for the A2C algorithm. The best hyperparameter combination for this algorithm reaches a mean episode reward of almost 3500, far higher than the desired threshold of 2600 and the best result obtained using A2C (1529). Other configurations also reached near or above a reward of 3000, indicating that the PPO algorithm works better than A2C for this problem. Compared to the A2C learning curves, PPO seems to converge much more smoothly, as indicated by the strong trend in the reward curve. Initially, the learning curves are relatively steep, followed by a trend reversal after approximately 50,000 timesteps, indicating that the algorithm has started converging towards a final policy. This indicates that agents using PPO not only converge better, but also do so using less timesteps.

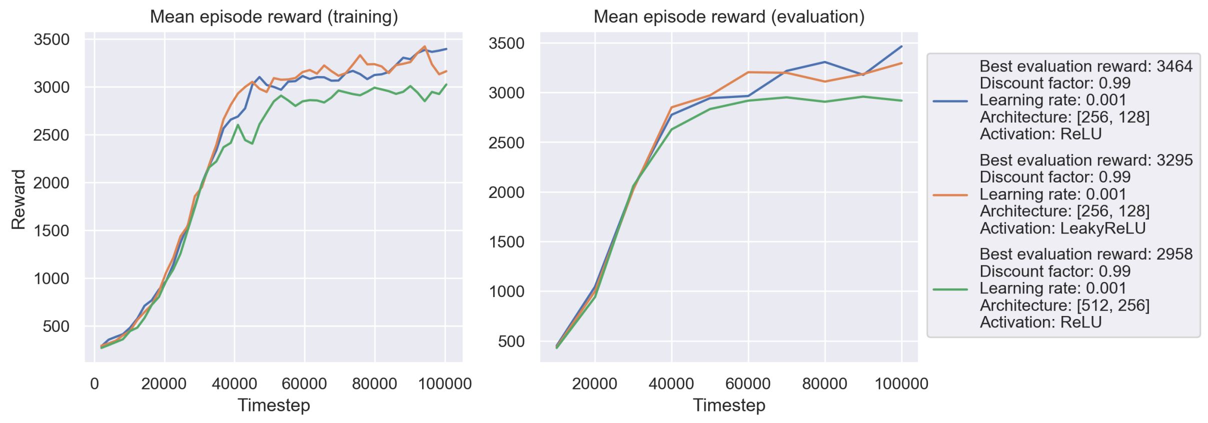

Figure 2: Learning curves for the best three models using the PPO algorithm, sorted in decreasing order of the highest achieved mean evaluation reward

Impact of Hyperparameters

From the selected hyperparameters and their respective search spaces, the most notable impact can be seen in the algorithm class itself. One can clearly conclude from the results presented and discussed above that the PPO algorithm far outperforms the A2C algorithm. This might be caused by a multitude of reasons. For instance, it might be the case that due to the limited change in policy at each epoch for PPO, it is able to converge towards optimal solutions, whereas the bigger policy changes in A2C put the policy at a place of no return, where optimal solutions can no longer be reached in a timely manner. Another reason might simply be that the PPO method is more advanced and efficient, with it being a more recent algorithm, currently being considered one of the state-of-the-art algorithms. Lastly, A2C is also known to be less data-efficient, often requiring more timesteps to converge properly. This could also be observed in our results. All in all, either one of these reasons, a combination of them or even something else might have caused the PPO algorithm to perform significantly better than the A2C algorithm.

Furthermore, normalizing the environment has also proven to have a very significant positive impact on the results. In fact, when comparing the exact same instances with the exception of tuning the hyperparameter NORMALIZE_ENV, only 1 out of 64 combinations ends up with a better best evaluation reward when not normalizing the environment, whereas for 63 out of 64 instances, the results improve. Therefore, it is recommended to normalize the environment, which is also common practice in machine learning.

Lastly, all other hyperparameters seem to have a less pronounced effect on the performance, where either of the possible values in the search space give better performance in some instances and worse performance in others. This might indicate that the effect of these hyperparameters are not so strongly correlated with the performance of the methods. However, certain combinations of these hyperparameters might work better than others for one of the algorithms as can be seen in the results. We must note however that any variations between hyperparameter configurations may be caused by random noise in the learning process, and that more extensive tests, or more evaluation samples, must be used for a more thorough comparison.

References

[Dantzig, 1957] Dantzig, G. B. (1957). Discrete-variable extremum problems. *Operations Research*, 5(2):266–277.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[Hubbs et al., 2020a] Hubbs, C. D., Perez, H. D., Sarwar, O., Sahinidis, N. V., Grossmann, I. E., and Wassick, J. M. (2020a). Or-gym: A reinforcement learning library for operations research problems.

[Hubbs et al., 2020b] Hubbs, C. D., Perez, H. D., Sarwar, O., Sahinidis, N. V., Grossmann, I. E., and Wassick, J. M. (2020b). Or-gym: A reinforcement learning library for operations research problems.

[Mnih et al., 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning.

[Raffin et al., 2021] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8.

[Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms.