Solving the Harvest CPR Appropriation Problem with Policy Gradient Techniques

Alessio Falai
M.Sc. in Artificial Intelligence
University of Bologna
alessio.falai@studio.unibo.it

Abstract

This report explains methodology and results for the final project in the Autonomous and Adaptive Systems course, held by professor Mirco Musolesi for the M.Sc. in Artificial Intelligence at the University of Bologna (academic year 2020 - 2021). In this work, we tackle a Common Pool Resource appropriation problem often referred to as Harvest or Gathering, in which independently learning agents aim to collect as many apples as possible, while also trying to keep the CPR flow healthy. In the original work, agents are trained with value-based RL methods, while in this work we rely on policy gradient techniques such as VPG, TRPO and PPO. Moreover, we implement a form of cooperation through Gifting, a Social Learning scheme in which agents can deliberately be generous and gift each other with collected resources.

1 Common Pool Resources

The term Common Pool Resource (CPR) usually refers to renewable natural resources that are freely accessible by agents in an environment. There are two main components in every CPR domain: the stock, which represents the amount of core resources; and the flow, which represents the amount of fringe units. Intuitively, the resource itself is defined by the CPR stock and whenever the stock gets depleted the CPR flow at the next timestep will be null. So, the flow is a consequence of the stock and represents how many resources can re-grow based on how many resources we have on the stock. After such resources have re-grown, they "flow" into the stock and the natural cycle goes on until agents are able to maintain the correct equilibrium between stock and flow.

CPR problems are mainly categorized in two buckets: provision, which deals with stock supply; and appropriation, which deals with allocation of the flow. The core issue in CPR appropriation is about over-appropriation, i.e. when an agent takes too many resources from the stock. Over-appropriation has negative consequences both for the agent itself, as collecting resources now may impact resource collection in the future, and for all the other agents, as whenever someone takes one resource from the pool all other agents have one resource less to collect.

Modeling CPR appropriation problems with artificial agents poses great challenges as such problems tend to have what's called a socially-deficient Nash equilibrium, meaning that the choice to appropriate is tipically dominant w.r.t. leaving the resource where it is to allow for re-growth. This is because the choice to restrain leads to no individual benefit, but to the cost of CPR exploitation by others. Thus, having a socially deficient Nash equilibrium means that if agents were to collaborate, each one of them would probably obtain far more resources than if they were to act in a completely independent way.

2 The commons game

The commons game, usually referred to as Harvest [7] or Gathering [4], is a simple grid-like environment in which agents have to collect apples (resources)[1]. The catch is that the apple re-growth rate depends on the spatial configuration of the uncollected apples (stock): for each empty location l, $p_t(L)$ is the probability that a new apple will grow at location l in the next time-step $t+1$, where L gives the number of already spawned apples in a ball of radius 2 centered around position l:

$$p_t(L) = \begin{cases} 0 & \text{if } L = 0 \\ 0.01 & \text{if } L = 1 \text{ or } 2 \\ 0.05 & \text{if } L = 3 \text{ or } 4 \\ 0.01 & \text{if } L > 4 \end{cases}$$

At time-step t, each agent i sees a local observation $o_i^{(t)} \in \mathbb{R}^{3\times20\times21}$ that spans 20 grid squares ahead and 10 grid squares on each side of its position (spread over 3 channels to form an RGB image). Such observation is dependent on the agent's location and orientation (out of the 4 cardinal directions N, E, S, W) on the grid. Each agent can then select the best action (according to its policy) out of 7 possibilities: step forward, step backward, step left, step right, rotate left, rotate right and stand still. The goal is to maximize overall resource collection without depleting the stock: beware that each agent receives a reward of $+1$ when they collect an apple and no other reward is given.

2.1 Tagging and gifting

In the original environment, agents have an additional action available: tagging. Agents may tag one another with a time-out (or laser) beam: the effect of being hit with the beam is that such agents would be removed from the game for D time-steps ($D = 25$ in the original implementation). Tagging has no direct rewards or punishments, but it gives agents the possibility to effectively lower the environment population, thus giving them more time to secure resources.

In order to let agents cooperate with each other, we rely on Social Learning [3], a new class of algorithms that enables agents to reshape the reward function of other agents with the goal of promoting cooperation and achieving higher global rewards in mixed-motive games. The form of Social Learning we opted for is Gifting, a peer rewarding strategy which allows agents to reward other agents as part of their action space. Beware that the rewards that agent i obtains from receiving a gift are independent from those it receives by acting upon the environment. With this extension, agents are equipped with an additional gifting beam, such that all agents within the beam range receive a gift from such agent: the gift g (defaults to $g = 1$) is equally split among all tagged agents. Different gifting mechanisms have been implemented:

1. Zero-sum: imposes no restriction on the number of times an agent can pick the gifting action within an episode (each time it sends a gift of value g, it incurs an immediate penalty of $-g$);
2. Fixed Budget: agents are allocated a fixed "budget" of size B at the start of each episode, which is decremented by g each time the agent chooses to gift (once the budget is depleted, gifting is no longer available as an action until the end of the episode);
3. Replenishable Budget: each agent starts with an initial budget $B = 0$, which is incremented as a function of the rewards it collects from the environment (for every 2 apples collected, the budget is increased by 1).

Thus, if the laser beam consists of an indirect punitive action, the gifting beam is a direct rewarding action.

[1]The environment is re-implemented from scratch to be compatible with OpenAI Gym [2], as the original code is not publicly available.

2.2 Social outcome metrics

Since the returns of a single agent are poor indicators of the group's behavior and performance, authors in [7] introduced 4 social metrics: Efficiency (U), measuring average returns; Sustainability (S), measuring the average time at which environmental rewards are collected; Equality (E), measuring the amount of impurity in resource collection; and Peace (P), measuring the average number of agents that are untagged at any given point. Given N agents, T maximum steps in an episode and $R_i = \sum_{t=1}^{T} r_i^{(t)}$ return of agent i in an episode, social outcome metrics are computed as follows:

$$\begin{aligned}
U &= \frac{1}{N}\sum_{i=1}^{N} R_i \\
E &= 1 - \frac{\sum_{i=1}^{N}\sum_{j=1}^{N} |R_i - R_j|}{2N\sum_{i=1}^{N} R_i} \\
S &= \frac{1}{N}\sum_{i=1}^{N} |\{t \mid r_i^{(t)} > 0\}| \\
P &= \frac{NT - \sum_{i=1}^{N}\sum_{t=1}^{T} \mathbb{I}\{\text{agent } i \text{ timed-out on step } t\}}{N}
\end{aligned} \tag{1}$$

3 Experiments

In this section we show results obtained by running multiple experiments[2]. In particular, for the single-agent case we only rely on RLlib's DQN implementation, while for the multi-agent case we test both RLlib's DQN and our custom policy gradient implementations. Given the computational complexity of the environment in a multi-agent scenario, running experiments for more than 1000 episodes is prohibitive and would require more resources: thus, we limit ourselves to such number of episodes and observe outcomes up to that point. Additionally, we test the correctness of our VPG, TRPO and PPO methods on the standard Cartpole environment, where we reach the reward threshold (200) to consider the problem as solved with all algorithms. Moreover, we rely on PPO to train multiple agents on the Harvest environment, both with and without the gifting action and always with the tagging action enabled.

Regarding DQN, the only difference w.r.t. the original algorithm is that we rely on its double dueling variant, instead of using the vanilla version. Moreover, the exploration strategy is ϵ-greedy with a linear decay from 1.0 to 0.1 over the course of the episode, while all the other hyperparameters are left as RLlib's defaults. For TRPO $\beta = 1.0$ and $d_{targ} = 0.01$, while for PPO $c_1 = 1.0$, $c_2 = 0.0$ and $\epsilon = 0.2$.

All neural networks are Multi-Layer Perceptrons (MLPs) with an input size of 1260 (given by the observation dimension $3 \times 20 \times 21$), 2 hidden layers with 32 units each and an output layer composed of either a single neuron (for value functions) or as many units as there are actions (for policy functions).

3.1 Single agent

In order to check the correctness of the implemented environment, we trained a single agent on the same small map (25×7) as in the original paper [7]. At the very beginning, apples have a spawn probability of 0.2 for each grid location. Results show that with such modifications the single agent is able to learn much faster to follow a greedy collection of apples: indeed, in the original paper they train for 50000 episodes, while our training procedure runs for about 1000 episodes.

3.2 Multi-agent

Table 1 shows results (in terms of social outcome metrics) obtained by training 11 agents on a 39×19 grid with 1000 maximum steps for each episode

[2]Training experiments are logged on W&B [1].

3.2.1 Value-based

3.2.2 Policy gradient

4 Results

Table 1 shows results (in terms of social outcome metrics) obtained by training 11 agents on a 39×19 grid with 1000 maximum steps for each episode. The DQN module is trained using RLlib [5], while policy gradient methods are implemented from scratch[3]. Training experiments are logged on W&B [1].

All neural networks are Multi-Layer Perceptrons (MLPs) with an input size of 1260 (given by the observation dimension $3 \times 20 \times 21$), 2 hidden layers with 32 units each and an output layer composed of either a single neuron (for value functions) or as many units as there are actions (for policy functions).

For DQN, the exploration strategy is ϵ-greedy with a linear decay from 1.0 to 0.1 over the course of the episode, while all the other hyperparameters are left as RLlib's defaults. For TRPO $\beta = 1.0$ and $d_{targ} = 0.01$, while for PPO $c_1 = 1.0$, $c_2 = 0.0$ and $\epsilon = 0.2$.

Table 1: Results

	U	E	S	P
DQN	0.0	0.0	0.0	0.0
VPG	0.0	0.0	0.0	0.0
TRPO	0.0	0.0	0.0	0.0
PPO	0.0	0.0	0.0	0.0

5 Conclusions

[3]The code is publicly available on Github.

References

[1] L. Biewald. Experiment tracking with weights and biases, 2020. URL `https://www.wandb.com/`. Software available from wandb.com.

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. URL `http://arxiv.org/abs/1606.01540`.

[3] P. Chelarescu. Deception in social learning: A multi-agent reinforcement learning perspective. *CoRR*, abs/2106.05402, 2021. URL `https://arxiv.org/abs/2106.05402`.

[4] J. Z. Leibo, V. F. Zambaldi, M. Lanctot, J. Marecki, and T. Graepel. Multi-agent reinforcement learning in sequential social dilemmas. *CoRR*, abs/1702.03037, 2017. URL `http://arxiv.org/abs/1702.03037`.

[5] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, J. Gonzalez, K. Goldberg, and I. Stoica. Ray rllib: A composable and scalable reinforcement learning library. *CoRR*, abs/1712.09381, 2017. URL `http://arxiv.org/abs/1712.09381`.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL `http://arxiv.org/abs/1312.5602`.

[7] J. Pérolat, J. Z. Leibo, V. F. Zambaldi, C. Beattie, K. Tuyls, and T. Graepel. A multi-agent reinforcement learning model of common-pool resource appropriation. *CoRR*, abs/1707.06600, 2017. URL `http://arxiv.org/abs/1707.06600`.

[8] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015. URL `http://arxiv.org/abs/1502.05477`.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL `http://arxiv.org/abs/1707.06347`.

[10] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 2000. URL `https://proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf`.

[11] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015. URL `http://arxiv.org/abs/1509.06461`.

[12] Z. Wang, N. de Freitas, and M. Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015. URL `http://arxiv.org/abs/1511.06581`.

A Reinforcement learning setting

A.1 Value-based methods

Value-based methods only store a value function (usually the state-action function Q), instead of explicitly storing the policy. Actions are then derived from the value function by picking the one with the best value. One of the most popular value-based algorithms is Q-learning and its neural variant DQN: a DQN (Deep Q-Network) [6] leverages neural networks from the point of view of universal function approximators, to estimate the optimal Q^* state-action function. The main idea behind DQN is to estimate Q^* with Q_θ and use the Bellman equation to update the weights θ of the network so that $Q_\theta \to Q^*$. In particular, the loss to be used is a regression one (e.g. MSE or Huber) to minimize the so-called TD (Temporal Difference) error:

$$L(\theta) = \mathbb{E}_{(s_t, a_t, r_{t+1}, s_{t+1})} \left[\left(\underbrace{r_{t+1} + \gamma \max_{a_{t+1}} Q_\theta(s_{t+1}, a_{t+1}) - Q_\theta(s_t, a_t)}_{TD} \right)^2 \right]$$

About the network architecture, the literature does not follow a clear definition, so that the term DQN is mostly linked to the whole training pipeline required to reach good estimates of the optimal Q function. For example, in the original paper [6] DeepMind researchers use an initial CNN (Convolutional Neural Network), since their input is represented by videogame frames, followed by an MLP (Multi-Layer Perceptron). Instead, the original Harvest implementation [7] only relies on the final fully-connected layers, where the number of input neurons is given by the linearization of the selected observation, the number of hidden cells and the number of hidden layers are hyperparameters, while the number of outputs is equal to the size of the action space.

Another thing to notice is that the weights used for the target versus the current (presumably non-optimal) Q values are different. In practical terms, a separate network is used to estimate the TD target. This target network has the same architecture as the function approximator but with frozen parameters. Every T steps (a hyperparameter) the parameters from the Q-network are copied to the target network. This leads to more stable training because it keeps the target function fixed (for a while). Because of this, assuming θ to represent the current network weights and θ^- the target network weights, the loss function is updated as follows:

$$L(\theta) = \mathbb{E}_{(s_t, a_t, r_{t+1}, s_{t+1})} \left[\left(r_{t+1} + \gamma \max_{a_{t+1}} Q_{\theta^-}(s_{t+1}, a_{t+1}) - Q_\theta(s_t, a_t) \right)^2 \right]$$

One important issue with standard DQNs is that they tend to overestimate Q values (which is mostly due to the $\max$ operator in the Bellman optimality equation), leading to unstable training and low quality policies. To mitigate this issue, in [11] and [12] authors propose two popular DQN variants, the double and dueling ones, respectively.

A.2 Policy gradient techniques

Policy gradient methods directly parametrize the policy and optimize it, instead of relying on proxy measures such as the Q function. The generic pipeline to follow in such algorithms is given in 1 and small adjustments span different popular techniques. For example, if we get rid of the advantage estimate and directly use returns in the loss function, we obtain REINFORCE [10]. Moreover, if we set the loss function to 2 we obtain a Vanilla Policy Gradient (VPG) implementation; if we set it to 3 we get the Trust Region Policy Optimization (TRPO) algorithm [8]; and if use loss 4 we get the Proximal Policy Optimization (PPO) method [9].

In all loss functions 2, 3 and 4 the first term is the loss of the policy network, i.e. the one that outputs log-probabilities for each action in the space, while the second term is the loss of the value network (or baseline network), i.e. the one which outputs a single value summarizing the goodness of the input state. The policy loss is a standard Cross Entropy (CE) between log-probabilities computed by the policy network (adjusted depending on the algorithm) and actual actions (A) performed by agents at trajectory collection time, while the value loss is a Mean Squared Error (MSE) one between computed values and actual returns.

Algorithm 1 Generic policy gradient pipeline

```
Require: policy $\theta_p^{(0)}$, baseline $\theta_b^{(0)}$, epochs $E$, batch size $B$, max steps $M$
1: $e \leftarrow 0$
2: while $e < E$ do
3:     Build a pool of trajectories $T^{(e)} = \{t_1^{(e)}, \ldots, t_B^{(e)}\}$ by running policy $\pi(\theta_p^{(e)})$,
       with $t_i^{(e)} = [t_i^{(e_0)}, \ldots, t_i^{(e_M)}]$ and $t_i^{(e_j)} = (s_i^{(e_j)}, a_i^{(e_j)}, r_i^{(e_j)}, s_i^{(e_{j+1})})$
4:     Compute rewards-to-go $\hat{R}_i^{(e)} = [\hat{R}_i^{(e_0)}, \ldots, \hat{R}_i^{(e_M)}]$
5:     Compute values as $\hat{V}_i^{(e)} = V(s_i^{(e_j)}; \theta_b^{(e)})$
6:     Compute advantage estimates as $\hat{A}_i^{(e)} = (\hat{R}_i^{(e)} - \hat{V}_i^{(e)})$
7:     Compute log-probabilities of actions as $\hat{P}_i^{(e)} = \pi(s_i^{(e_j)}; \theta_p^{(e)})$
8:     Compute loss (2, 3 or 4) and back-propagate
9: end while
```

$$\begin{aligned} L_{VPG} = CE\left(\hat{P}_i^{(e)} \cdot \hat{A}_i^{(e)}, A\right) \\ + \alpha \cdot MSE\left(\hat{V}_i^{(e)}, \hat{R}_i^{(e)}\right) \end{aligned} \tag{2}$$

The KL divergence term in the TRPO objective is implemented through regularization and the network is trained with standard SGD, instead of relying on conjugate gradient algorithms as proposed in the original paper [8]. Given that using a penalty term instead of an hard constraint leaves the optimization problem effectively unconstrained, it is difficult to select a single value of the hyperparameter β to perform well across every instance of a RL environment. Because of this, as proposed in [9], we implement a β update strategy, to adjust the hyperparameter during learning: given the KL divergence d and a target KL divergence d_{targ}, if $d < d_{targ}/1.5$ then $\beta \leftarrow \beta/2$; otherwise, if $d > d_{targ} \times 1.5$ then $\beta \leftarrow \beta \times 2$.

$$\begin{aligned} L_{TRPO} = CE\left(\frac{\hat{P}_i^{(e)}}{\hat{P}_i^{(e-1)}} \cdot \hat{A}_i^{(e)}, A\right) \\ + \alpha \cdot MSE\left(\hat{V}_i^{(e)}, \hat{R}_i^{(e)}\right) \\ - \beta \cdot KL\left(\pi(\theta_p^{(e-1)}), \pi(\theta_p^{(e)})\right) \end{aligned} \tag{3}$$

PPO revisits and simplifies the idea used in TRPO of penalizing relatively big changes in the policy from one parameters update to the next one. To do so, it uses a custom objective that behaves differently depending on the sign of the advantage: if positive (i.e. when the selected action gets a better outcome than expected) the probability ratio is clipped at $1 + \epsilon$; if negative, the probability ratio is clipped at $1 - \epsilon$. The clipped objective removes the incentive to move the probability ratio outside of the region $[1 - \epsilon, 1 + \epsilon]$. The minimum between the unclipped and the clipped objective serves as a way to only consider changes in probability ratios that will make the objective worse. The final loss term, S, is an entropy bonus to ensure sufficient exploration.

$$\begin{aligned} L_{PPO} = CE\left(\min\left(\frac{\hat{P}_i^{(e)}}{\hat{P}_i^{(e-1)}} \cdot \hat{A}_i^{(e)}, clip\left(\frac{\hat{P}_i^{(e)}}{\hat{P}_i^{(e-1)}}, 1-\epsilon, 1+\epsilon\right) \cdot \hat{A}_i^{(e)}\right), A\right) \\ - c_1 \cdot MSE\left(\hat{V}_i^{(e)}, \hat{R}_i^{(e)}\right) \\ + c_2 \cdot S\left(\pi(\theta_p^{(e)})\right) \end{aligned} \tag{4}$$