Analysis on Reinforcement Learning Methods on Navigation in a Dynamic Environment

Callum Hendry: 100970932
callumhendry@cmail.carleton.ca

Jason Dunn: 101140828
jason.dunn@carleton.ca

Ujan Sen: 101171605
ujansen@cmail.carleton.ca

Uvernes Somarriba: 101146733
uvernes.somarribacast@cmail.carleton.ca

1 Introduction

2 Methods

2.1 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) was introduced in 2017 [1] and has achieved widespread success in the realm of reinforcement learning. It was a successor to Trust Region Policy Optimization (TRPO) introduced in 2015 [2] which used stochastic gradient ascent by using a trust region constraint to regulate between the old policy and the new updated policy. PPO is considered state-of-the-art and is the default reinforcement learning algorithm used at OpenAI because of its ease of use and significantly better and faster performance than its counterparts.

The biggest difference between PPO and TRPO is that PPO uses a clipping function which effectively ensures that the new learned policy does not deviate more than a certain amount from the old policy. This is done by first calculating two different sets of surrogates for the policy network. The objective for PPO is calculated as follows [1]:

$$L_\theta = \mathbb{E}_t[min(r_t(\theta)A_t, CLIP(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)]$$

For the surrogates, the ratios are first calculated between the predicted actions at the current state given the old policy and the new predicted actions. The first surrogate is calculated by weighting the ratios by the calculated advantages. The second surrogate is calculated in a similar way except the ratios are first clipped between 1 - ϵ, and 1 + ϵ where epsilon is defined as the clip ratio, and then weighted with the advantages. The loss is then defined to be the minimum of these surrogates.

Advantages are calculated using Generalized Advantage Estimation (GAE) introduced in 2015 [3]. GAE is an improvement upon traditional advantage estimation methods because of the introduction of the λ term which allows a trade-off between bias and variance. A λ of 0 reduces GAE to a one-step estimation which is just standard advantage estimation whereas a λ of 1 considers rewards infinitely into the future. It takes into consideration the temporal difference of not only the immediate rewards but also the expected future rewards. The formula for GAE calculation is as follows [3]:

$$\hat{A}_t^{GAE} = \sum_{k=0}^{\infty}(\gamma.\lambda^k).\delta_{t+k}$$

- $\hat{A}_t^{GAE}$ is the GAE at time step t
- γ is the discount factor
- λ is the GAE parameter, the tradeoff between bias and variance
- k is the time step offset

Preprint. Under review.

- δ_{t+k} is the advantage at time step $t + k$ calculated using the one step standard advantage calculation

The intuition behind choosing PPO is the same as TRPO: "being safe". Since the loss function is defined as the minimum of the two surrogates, the objective becomes a lower bound of what the agent knows is possible. It approaches the task being a pessimist, which has often times proved to be more beneficial than being optimistic with little chances of recovery. TRPO's objective function achieves a similar thing but is quite different in computation [2].

$$L_\theta = \mathbb{E}_t[D_{KL}(\pi_{\theta_{old}}(.|s_t)||\pi_\theta(.|s_t))] \leq \delta$$

The benefit PPO gives over TRPO is the usage of clipped ratios and the surrogates. TRPO enforces a strict trust region constraint where the KL-divergence between old policies and new policies is small enough, within a parameter δ leading to a second-order optimization problem [2][4]. PPO effectively does the same thing using the clipped ratio and taking the minimum of the surrogate losses resulting in a first-order optimization problem. This leads to PPO utilizing fewer computation resources while providing better results.

The PPO implementation was performed using the library stable_baselines3 available here. It ran with default arguments as described in the documentation.

2.1.1 Environmental Considerations

The environment was modified and the results for each modification of the environment is mentioned in the Results section.

- The targets were reduced from multiple to single.
- Running traffic lights, making u-turns, and going out of bounds were always penalized instead of it being probabilistic
- All penalized actions penalized to the same amount

3 Results

In this section we will discuss the results for the various implemented algorithms:

Table 1: Results for PPO

Target	Reward Type	Reward Scaling[1]	Agent Score[2]	Random Score[2]	Train Time[3]
Single	Deterministic	Same	888	-396	15
Single	Deterministic	Different[4]	935	-369	15
Single	Probabilistic	Different	920	-334	20
Multiple	Deterministic	Same	205	-405	20
Multiple	Deterministic	Different	379	-390	20
Multiple	Probabilistic	Different	255	-410	30

4 Discussion

From the results, it is clear that for PPO, the more complex the environment becomes, the worse the agent performs. This is expected as it is quite difficult to find an optimal solution to our problem considering how the environment behaves. For the environment dynamic when there is a single target, there does not seem to be large of a difference in performance. The higher score for different reward scaling compared to same can be attributed to the fact that when it makes mistakes, the penalty is

[1] Only negative rewards are altered, positive rewards remain the same

[2] Scores are out of 1000 for single target

[3] In minutes

[4] Stop at green: -2; Stop at node that is not traffic light: -2; Everything else: -5

lower in the different reward scaling scenario, which is coincidentally also the case for the multiple targets scenario. It is worth noting that the total score for multiple targets is difficult to say since the number of targets is not fixed but it is more than 1000 since there is always at least one target.

An interesting idea would be to fix the traffic lights instead of randomly initializing them. This seems quite intuitive and in fact, could even be considered closer to reality since if the environment is a representation of an urban city setting and we consider that it is the same city, traffic lights and their locations will not suddenly change. And even if they do change, they should remain consistent for a significant time once they do. Perhaps this is an avenue worth exploring where traffic lights are initially fixed but can randomly change after a certain amount of time steps.

Another interesting observation we found was that when there were varying penalties, sometimes the agent would go out of bounds. This could be due to the fact that going out of bounds had a lower penalty than running traffic lights or making a u-turn. Moreover, in the multiple target scenario, the agent performs significantly worse when penalties are probabilistic which makes sense since this is a difficult environment dynamic to learn. Whenever probabilistic elements are introduced into the environment dynamics, it makes training significantly tougher since even though it might revisit a state and take the same action as it did last time, it might receive a different reward.

Furthermore, it is worth noting that multiple targets yield worse results than singular targets. This is expected because of the fact that when there are multiple targets, there is no way to encode direction, which was something we were able to do when there was a single target. For example, if there was a single target, we had a 1D array indicating if the target was to the up, down, left, right of the agent. But when there are multiple scattered targets, this sort of information is difficult to encode. However, given more time and resources, we would like to perhaps train another agent that has the task of selecting the next target our agent should try to reach effectively reducing the target space to a singular target again. This could be achieved via multiple methods with the most naïve method being just picking the target closest (in terms of Manhattan distance) to where the agent is. A further, more refined method would be allocating each target with a priority and then choosing the next target based on the priority. A further refinement would include a combination of the two.

Given more time and resources, we would also like to test out more scenarios including various combinations of the scenarios mentioned in 2.1.1. We would also like to test various reinforcement learning algorithms and their performance if the targets had priorities assigned to them. We would also like to add pedestrians and random events in the environment and evaluate performance.

References

[1] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. arXiv preprint arXiv:1707.06347.

[2] Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2017). Trust Region Policy Optimization. arXiv preprint arXiv:1502.05477.

[3] Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2018). High-Dimensional Continuous Control Using Generalized Advantage Estimation. arXiv preprint arXiv:1506.02438

[4] Wang, Y., He, H., Wen, C., & Tan, X. (2020). Truly Proximal Policy Optimization. arXiv preprint arXiv:1903.07940.