CS 260R Metadrive Report

Colton Rowe
CS 260R, Dr. Bolei Zhou

1 Method Introduction: PPO

Proximal Policy Optimization (PPO) is an on-policy method that uses gradient clipping to keep the model parameters from changing too quickly. PPO balances the trade off between exploration and exploitation in a wide set of environments, both continuous and discrete. Because of it's efficiency, I chose to use PPO to train my agents. In on-road driving, we care less about novelty and more about avoiding crashes and getting to the destination. PPO is an on-policy algorithm, so using PPO will give the agent good driving experience in a set of environments it's likely to encounter. I initially experimented with some off-policy algorithms like TD3 and SAC, but empirically, their training curves didn't converge as well as PPO.

2 Tuning the Model

To increase the route completion of my agent, I experimented with multiple hyper parameters of the model.

2.1 Training Scenario Count

This is the number of unique scenarios the model is able to incorporate into its experience. I found that increasing the number of scenarios generally improved the performance of the model, up until a point. I tried testing scenario amounts of 50, 500, 5,000, and 50,000. The sweet spot was 5,000 - models with lower numbers of scenarios didn't have enough variance in their training data to converge past 0.45 route completion, and models with higher numbers of scenarios were took too long to start.

2.2 Clip Range

I tested clip ranges of 0.1, 0.2, 0.3, and 0.4. Intuitively, the clip range is the radius the model can step within to update its weights. Higher clip ranges give the model more flexibility with its updates, but risk "falling off a cliff," where a bad update has a fatal effect on the model. The models with higher clip ranges tended to start off slower before settling into a gradual ascent, while models with lower clip ranges started off much more quickly. Ultimately, 0.1 proved to be the best clip range in terms of performance.

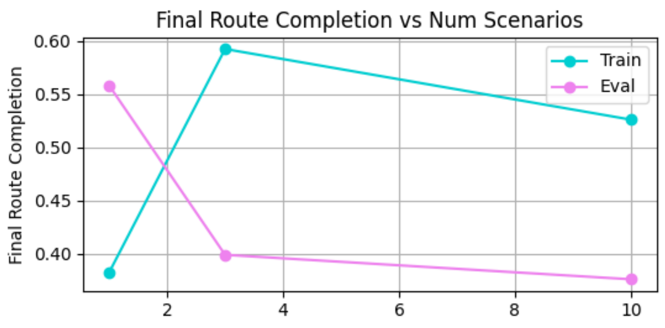
2.3 Reward Function and Gamma

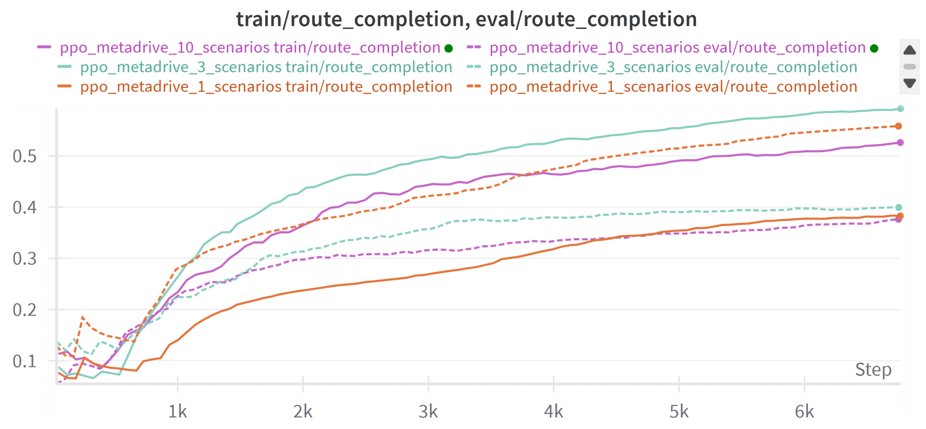
Because our agent is tested solely on route completion, it should care less about hitting obstacles and more about completing the route. In the real world, we care much more about safety when training models, but for this toy example, modifying the reward function to account for this difference in evaluation could lead to tangible results. I modified the reward function to have less of a penalty for crashes and driving off the road. This dramatically raised the ceiling for the route completion, and in testing, the model still has a very low crash and off-road percentages. I think this is because the model cares about these penalties to the point that it effects it's success, and less so that is intrinsically weary of them.

Gamma, the discount factor, effects how much the agent cares about future rewards versus immediate rewards. A discount factor closer to 1 means that the agent cares about long-term rewards more, and a discount factor closer to 0 means the model cares mostly about short term rewards. I tried discount factors of 0.99 and 0.995. As expected, the higher discount factor tended to improve the models success in the environment.

3 Generalization Experiment

3.1 Graphs





3.2 Discussion

Generally, the intuition is that more example scenarios leads to higher performance, but I found that to not necessarily be the case in the tests with a small number of training scenarios. In the test with only 1 scenario, the evaluation route completion is much higher than the training route completion. It's possible that this is due to random chance, maybe the training scenario was much easier or more difficult than the evaluation scenarios, and the model was able to efficiently exploit this. With higher numbers of training scenarios, the curves look more like what you'd expect, with higher training success than testing success. With these small number of scenarios though, the variance could be too high to have this trend hold. As I increased the number of scenarios by orders of 10 in subsequent tests, the results looked much more like what you would expect.

4 Final Model

The final architecture I used was PPO with a MLP Actor-Critic policy. I used a $\gamma = 0.998$, 5000 scenarios, and a clip ratio of 0.1. I decreased the penalty for crashing and off-roading by half in the value function. I was able to train this model to around step 5,500 when my computer crashed. I reloaded the agent from a checkpoint to train it for another 2,000 steps, and it began to perform better than it had been before. I suspect this might have been because the optimizer was too comfortable in the original run, so pausing and rerunning the model could have introduced necessary stochasticity into the system. This may be indicative that in the future, I should experiment with adding entropy into the system as the steps progress. This matches the intuition that the environment should get harder as the model gets better in order to continue progress - if the gradient flat lines, then the model will have a harder time improving.

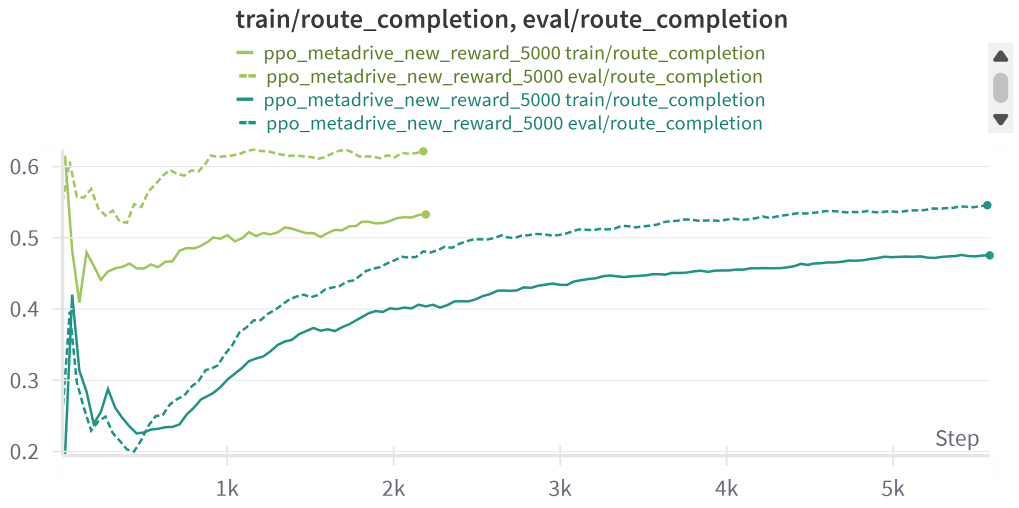


Figure 1: Testing and Evaluation performance of final model over time.

```
==================================================
EVALUATING AGENT agent_HIDDEN_UID (CREATOR: Colton Rowe, UID: HIDDEN_UID)
==================================================
Start evaluation!
==================================================
THE PERFORMANCE OF agent_HIDDEN_UID:

crash_sidewalk_rate: 0.0
crash_vehicle_rate: 0.006843042482828797
episode_cost: 4.26
episode_length: 374.47
episode_reward: 324.4400860401998
max_step_rate: 0.0
out_of_road_rate: 0.3
route_completion: 0.8830359743232336
success_rate: 0.7
```

In testing, the model tends to performs better even than the evaluation curve on the model. When the agent chooses it's action, instead of taking the maximum of it's policy, the agent samples from the distribution of the predicted action to remove determinism from its behavior. This random sampling greatly improves performance, taking it from around 45% route completion to nearly 90% route completion.

4.1 Looking forward

In the future, I would like to try a actor critic policy with a recurrent neural network (RNN) instead of an fully connected neural network (FC net). Stable Baselines 3 doesn't yet provide support for RNNs, but some community builds of SB3 has support for them. Introducing a recurrent model could let the agent perceive time-dependencies in a similar way that humans perceive the world.