TorchKart

Jacob Terkuc

Methods

Methods from Artificial Intelligence Used

TorchKart uses several methods from artificial intelligence in its final implementation.

Proximal Policy Optimization (PPO): The agent uses PPO, which is made up of an 'actor', which decides what actions the agent will make, and a 'critic' which estimates the state's value. When an Agent or set of agents completes the number of steps defined by the `--num-steps` flag, the policy is updated `--update-epochs` times. The 'proximal' part of PPO means that, unlike with DQN where the exploratory actions are random, the exploratory actions are made around actions that the agent knows are good. This 'proximality' of new actions means that the learning is much more stable, and the agent can complete a course in significantly less total steps.

Frame Stacking: To give the agent a sense of 'momentum' in a movement-based game such as Mario Kart, the agent is fed 8 observation frames that are 'stacked'. These stacked frames are fed into the network. As a result of this implementation, the agent was able to better compensate for environment factors such as wide turns and was much less likely to hit walls after a few million steps.

Feature Normalization: The values fed from memory into the network are normalized between [-1, 1]. This normalization drastically improves performance, as a value such as speed [0, 67] would have much less effect than a value like orientation [0, 65535] when applied to the network unnormalized.

Validation Strategy or Experiments

The main datapoint I was interested in minimizing was the completed race time. Between all my experiments tweaking the reward function, hyperparameters or features, the goal has eventually been to continue to see a downwards trend in the completed race time.

Other datapoints were collected between experiments such as episodic length and reward, as well as episode final lap. These were not chosen as an accurate measure because an agent can get a relatively high reward while not completing a race (it found a bug to exploit, for example).

One experiment that ended up working well was asynchronous resetting for agents. Basically, in the first iterations the n agents would all start at the same time, progress for 4k steps. A better strategy I found was to keep track of an individual agent's progress, and if it stagnated for a certain time (in this case, 60 frames) the single agent would be reset to the checkpoint, along with a –20 reward.

This resetting strategy significantly decreased the number of steps before an agent completed a race, in one attempt reaching ~100M+ steps without completing a race down to ~5M.

Increasing the number of stacked frames from 4 to 8 also helped after implementing drifting. It allowed the model to see that the curve of the road was changing, and to more proactively begin a drift around a corner.

Results

A human benchmark for a completed race (3 laps around “Luigi’s Circuit”) was set at 124 seconds.

When evaluating the performance of DQN and PPO, the main metric used was the ‘Completed Race Time’. For DQN, the agent was never able to finish a lap, let alone a race, so it was found to be an ineffective solution for the given environment.

The first successful implementation of PPO had high run-to-run variance in the completed race time and found a minimum of ~ 140 seconds.

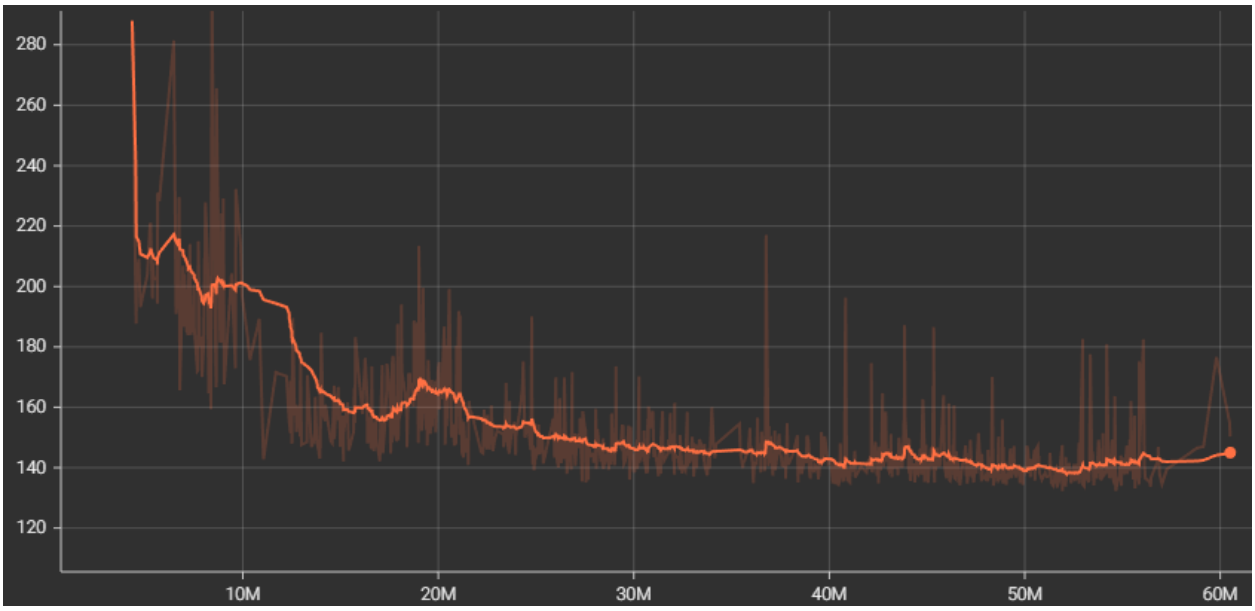

Figure 1. ‘completed_race_time’ for first implementation of PPO Agent

Further optimizations reduced the run-to-run variance, as well as leading the agent to finding a new minimum. These optimizations included optimizing the normalization of feature inputs to the network, implementing frame stacking, and reducing the complexity of the reward function. This implementation found a minimum of ~132 seconds.

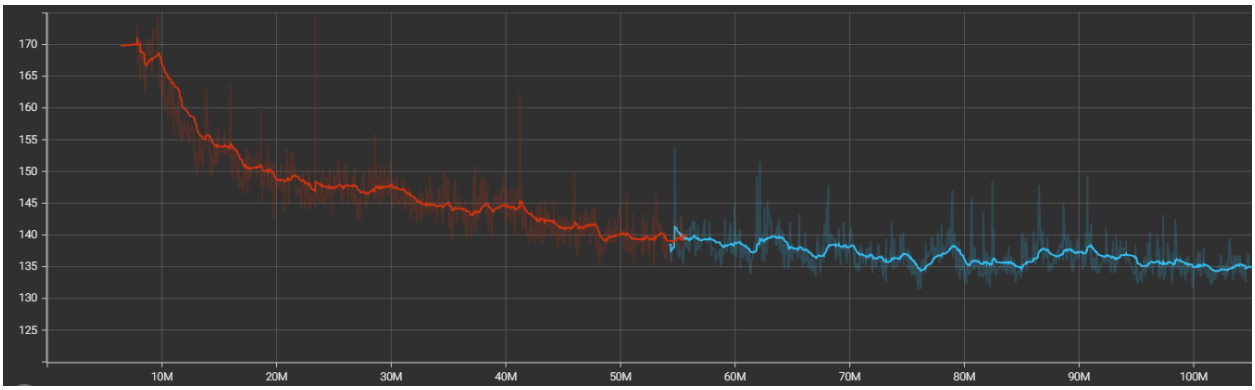

Figure 2. 'completed_race_time' for an optimized implementation of PPO Agent

An unintended consequence of this frame stacking was also that the kart remained much more 'stable' when driving on the track. Rather than constantly bouncing from side to side on the road, the agent remained much more stable on the track.

The final implementation of the PPO Agent focused on further simplifying the reward function and resetting system. The reward function removes the incentive for the agent to stay centered on the track and instead centers the reward on speed and course progress.

The resetting feature was also reworked, making it so that an agent was reset after 60 frames of inactivity, rather than 600 frames. This reduced garbage data from being introduced into the training.

This implementation found a minimum of ~122 seconds, beating the human benchmark of 124 seconds.

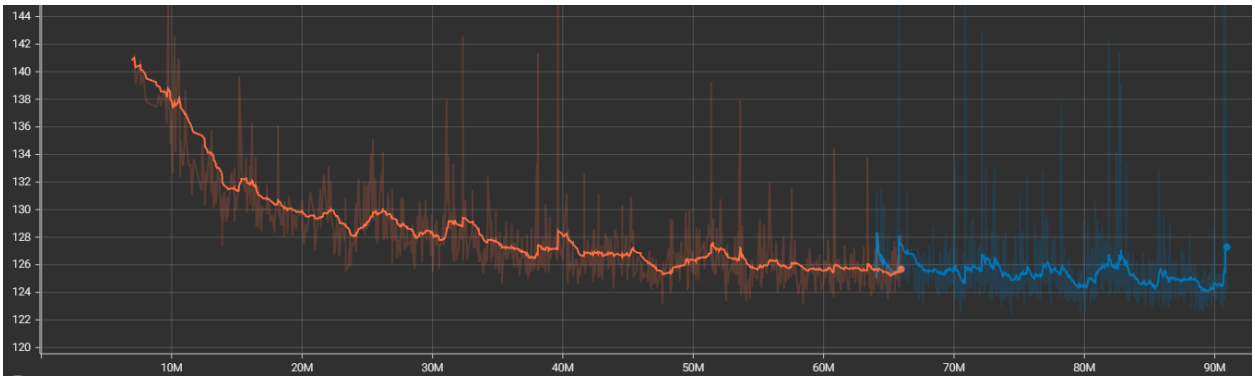

Figure 3. 'completed_race_time' with a simplified reward function for PPO Agent.

References

https://github.com/Farama-Foundation/Gymnasium

https://gymnasium.farama.org/index.html

https://github.com/pytorch/pytorch

https://docs.pytorch.org/docs/stable/index.html

https://arxiv.org/abs/1707.06347

https://github.com/n64decomp/mk64

https://n64decomp.github.io/mk64/md_docs_2doxygen__syms.html

https://github.com/kevinhughes27/TensorKart