Solving a custom gym environment - Deep reinforcement learning final project

Lorenz Leisner lleisner@uos.de
Dario Holkieslich dholkieslich@uos.de
Tom Pieper tpieper@uos.de

September 2022

1 Abstract

This is the final project for the course Deep Reinforcement Learning. We built a Gym Environment of the well known Chrome Dino Game from scratch and implemented different reinforcement learning algorithms in order to solve it and compare their relative performance. The algorithms featured are deep q-learning as well as double deep q-learning and proximal policy optimization.

2 Introduction and Motivation

One of the most compelling and fascinating aspects of reinforcement learning and AI in general, is that you can throw it at an incredible range of problems and just let it do its thing. That is why, instead of just building a new model for an already existing gym environment, we wanted to create our own little problem. Jump and Run games are widely popular and, to our surprise, were not actually featured yet in the gym or petting-zoo library, so we decided on that to be our game-genre. As an well known classic in that domain, and its relative simplicity compared to other games, the Chrome Dino Game seemed like a good starting point for a first reinforcement learning project.

3 Gym Game Environment

3.1 Description

The environment is an implementation of the classic google chrome dino jump and run game as a gym environment. The Dino agent has to learn to dodge incoming obstacles, which are either a cactus of varying width and height that has to be jumped over, or a bird that has to be jumped over or ducked under, depending on its height.

3.2 Action Space

There are three discrete actions available: do nothing, jump and duck.

3.2.1 Continuous vs discrete action space

We first thought about working with a continuous action space (e.g. jump strength and duck height), which could have been handled by the PPO as well, but later decided it wasn't bringing any value to the environment.

3.3 Observation Space

The state is a 14-dimensional vector: the x and y coordinates as well as width and height of the player(dino), the bird and the cactus. Additionally, a float value for the current game speed and a boolean indicating whether the Player is in the air or not.

3.3.1 Observation space overhaul

The observation space is a gym.spaces.Dict with Boxes for the the Dino, Bird, Cactus, Speed and IsJumping. We are using the gym.wrappers.FlattenObservation for training, which results in Observation Shape of (14,). Initially, we just worked with a simple observation space of shape (5,) (agent height, distance to closest obstacle, obstacle height, game speed, is jumping) but later then switched over to this more detailed approach using Wrappers.

3.4 Rewards

The agent receives a standard reward of +1 for every step it stays alive. Though that should be sufficient for training the agent, we experimented around by giving the agent additional rewards to see if that would improve performance. We tried using the following rewards as aid: Jumping is penalized with a negative reward of -0.1 to incentivise staying on the ground, as jumping limits future actions. Passing an obstacle (jumping over it or ducking under it), rewards an extra +3 points, to easily counter the negative reward for jumping.

3.5 Termination

The episode finishes if the Dino (agent) collides with any obstacle on its path (bird or cactus).

4 Building different Models

Our project idea from the get-go featured a PPO algorithm implementation, as it is one of the most used used algorithms in DRL today. That is because of its wide range of possible applications and adjustability, as well as its stable performance.

However, as it became clear during development that our environment would only feature a discrete action space, we decided to also train a Deep Q Network in the environment, and compare the results.

4.1 Double deep q-learning

As presented before, our Environment features a continuous observation space and a discrete action space. For this sort of deep reinforcement learning problem, the natural choice is a deep q network. With a Neural Network handling the high-dimensional sensory input, and using q-learning for training. To reduce the overestimation bias common in q-learning, we also use a double deep q-learning approach, with one Network choosing the action and the other computing the q-values. The Q-Network consists of two Dense layers, each with 256 neurons using the ReLu activation function, as well the output layer with 3 (number of possible actions) Neurons. The optimization employed to train the network is Adam.

4.1.1 Hyperparameters

Whilst we played around with different settings during our experiments, the final model was trained with the following Hyperparameters: the discount was set to 0.99, the learning rate to 0.00025. In the DDQN, the number of steps between target network updates is 10. The size of the replay buffer is 1M. The memory gets sampled to update the network with minibatches of size 64. The exploration policy is an epsilon greedy policy with the epsilon decreasing by a factor of 0.996 from 1 to 0.01.

4.2 Proximal Policy Optimization

PPO algorithms are more forgiving in hyperparameter initializations and generally work well on a wide variety of reinforcement learning tasks. So even if our DQN model wouldn't be able to solve the environment, we had high hopes for the PPO Agent.

4.2.1 Hyperparameters

Similar to the DQN, the final model was trained with a learning rate of 0.0025, a discount factor of 0.99, and a batch size of 64. The model features a horizon range of 30, with the number of epochs set to 30 as well. The clipping range is set to 0.2. The generalized advantage estimator (GAE lambda) is 0.95. As our own experiments were inconclusive, the final settings were adopted from the respective paper. (Schulman et al. 2017)

5 Results

5.1 Training the network

We trained the DQN, the DDQN and the PPO Agents for 200 episodes each, which took about 5 hours of training time in total. Although we can report some spikes in training, where the Agents were able to get over the first obstacle consistently for a few episodes, those spikes were not stable at all and the model was not able to improve upon the progress it had made.

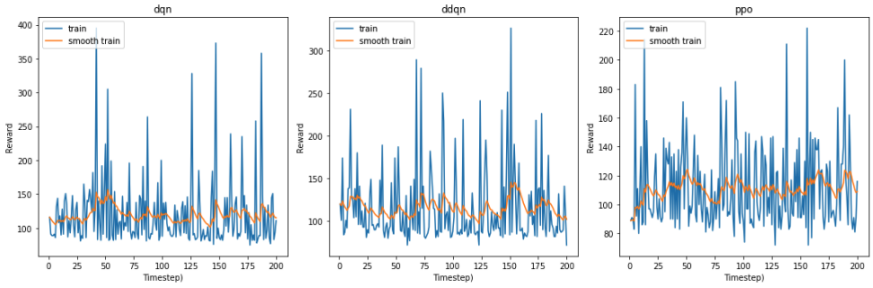

Figure 1: training scores

5.2 Comparing the models

Other than the training time, which was considerably shorter for the PPO algorithm, there was not much difference in performance across the different Agents.

6 Discussion

Unfortunately, training did not go as planned. Not one of the agents was able to beat the environment. We are not sure why that is. We tested the Agents on other environments (lunar lander and cartpole) in which they performed just fine and were able to learn the game mechanics. We tested a wide range of possible hyperparameters as suggested in the respective papers for the deep q network as well as for the ppo (Schulman et al., 2017, van Hasselt et al., 2015), again without any success.

The obvious conclusion for us at that point was, that something had to be wrong with the environment itself. So we did a complete overhaul of how our observation space worked, thinking maybe the agent did not get enough information to make good decisions. Before, our observation space consisted of just 4 values: the player height, the distance to the next obstacle, that obstacles' height, and the game speed. So now we are passing basically all the information to the agent that is available: every relevant objects' x, y, width and height, the game speed and even a helper boolean variable to indicate whether the agent is in the air. We also considered giving it yet another boolean variable indicating

whether an obstacle was in critical range of the player, but we thought that this would defeat the purpose of building an AI Agent all together, as we would basically be telling it when to jump.

Next, we tried different reward functions in order to help the agent get over the obstacles by reward shaping: Instead of getting just a reward of +1 every step (which should get the agent to try to stay alive as long as possible, ergo beating the game), we gave it a reward of +10 for overcoming an obstacle, a reward of -0.1 for being in the air (as this cancels out its ability to jump) and a reward of -10 for terminating. As this did not yield any new results either, we decided to leave out any additional reward shaping and stick to the basic reward of +1 every step.

We also experimented on the game with ourselves as players, making the game much easier to play in general, by increasing jump height and decreasing rate at which objects would spawn in. At one point we also removed all birds from the game, so the agent would only have to learn to jump, instead of both jumping and ducking. All without any effect.

Whilst we are still unsure why our experiment failed, it should be noted that due to some complications in our group's schedule, we never managed to train the Agents for longer than 500 episodes. Whilst this was more than enough to beat (or get some noticeable improvement on) the lunar lander environment, on which we had tested the Agents before, it might just not have been enough training time to beat the chrome dino environment. The respective Papers we were relying upon during implementation all had training times of about 50M episodes, which was obviously not feasible for us.

7 References

- van Hasselt, H., Guez, A., Silver, D., (2015). Deep Reinforcement Learning with Double Q-learning. https://arxiv.org/abs/1509.06461
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O. (2017) Proximal Policy Optimization Algorithms. https://arxiv.org/abs/1707.06347
- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. https://arxiv.org/abs/1312.5602
- Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., Silver, D. (2017). Rainbow: Combining improvements in deep reinforcement learning. https://arxiv.org/abs/1710.02298
- Bellemare, M. G., Naddaf, Y., Veness, J., Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. https://arxiv.org/abs/1207.4708

- Puigdomenech Badia, A., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, D., Blundell, C. (2020). Agent57: Outperforming the atari human benchmark. https://arxiv.org/abs/2003.13350