Fredrik Gustafsson

Jakob Nyberg

Sara Hedar

UPPSALA
UNIVERSITET

Title

Natural Computation Methods for Machine Learning

Uppsala

May 9, 2019

Abstract

Hejsan jag är ett abstract.

Contents

1 Introduction

According to Richard Sutton and Andrew Barto reinforcement learning is a way to understand and automate goal-directed decision making and learning, from a computational perspective. It puts a higher emphasis on learning by directly interacting with the environment by an agent when compared to other approaches. Before reinforcement learning became a field of its own in the late 1980s there existed two main fields, that of learning by trial and error and the problem of optimal control. Optimal control is used here to refer to the problem of designing a way to maximise or minimise some measure of a dynamical system and came to use in the late 1950s. Learning by trial and error on the other hand goes back to at least 1911 when Edward Thorndike described "the effect of reinforcing events on the tendency to select actions". It is based on behaviours observed in nature and how animals can learn, e.g. differentiate between punishments and rewards. These two fields joined together with the field of temporal-difference methods which tries to solve prediction problems from previous experience. From this the field of reinforcement learning as we know it today was created [8].

Fast forward to 2016 and an algorithm based on reinforcement learning was able to beat the European Go champion. This was seen as an impressive feat due to Go being seen as the most challenging classic game due to the enormous search space [7]. Something which is interesting to investigate is how well a reinforcement learning algorithm, in this case proximal policy optimisation, generalises. How well an algorithm trained with an easier environment perform on a harder and vice versa. An example of this would be to train a robot in an environment and then control the robot in real life where due to random element the real world environment could become harder to navigate.

In order to investigate the ability to generalise different environments was implemented in the game engine Unity. Unity provides built in tools such as physics engine, graphics shaders, scripting API:s and a way to implement tensorflow models. Hence time could be used to investigate performance instead of building the environments. Unity is also free for personal and academic use which was another reason as to why it was used.

2 Theory

2.1 Reinforcement Learning

Reinforcement learning, as a problem, contains two main components: The *agent* and the *environment* [8]. The agent acts upon the environment in some way and tries to optimise some reward function. The reward function depends on what the developer wants the agent to do. One example would be for balancing a beam where the reward could be given as long as the beam does not fall down.

2.2 Learning Environments

Many reinforcement learning solutions intend to solve physical real world problems, such as controlling robots or driving cars.

Creating real-life environment for training and testing can be expensive, as well as hard to replicate for other researchers. An industrial robot arm can be very costly and constructing an entire track to test a self-driving car is not viable for most institutions.

A solution to this problem is to instead use virtual learning environments, simulated in computers. The OpenAI group has released a number of environments, referred to as *gyms*, which serve as a standardised set of benchmarks for RL agents [1].

There have been issues raised about the use of virtual environments. In their 2018 paper about benchmarking RL algorithms [9] Amy Zhang et. al raise the question whether it is viable to test increasingly advanced learning algorithms on environments that, compared to the real world, are very simple. This may cause the agents to overfit and perform poorly when applied in real life. They propose some solutions to this issue, such as injecting different forms of natural noise into the environment. For example, if the agent uses visual input to play a game, the background could be changed after each round.

2.3 Policy Gradient Methods

The the context of reinforcement learning a policy is a mapping from a state to the probability of choosing each possible action in that state [8]. Let $\pi(a|s)$ denotes the probability of choosing action a in state s while following the policy π.

Policy Gradient Methods learn a parameterised policy based on the gradient of a scalar performance measure. The policy parameter is updated on each step in the direction of an approximated gradient. Policy Gradient Methods provide handling of continuous action spaces.

2.4 Proximal Policy Optimisation

Proximal Policy Optimisation (PPO) is a family of gradient methods for reinforcement learning proposed by the non-profit artificial intelligence research company OpenAI in 2017 [5]. PPO is a first order optimisation method which makes a "pessimistic" estimate of the performance of the policy by "clipping" the so called probability rations. The method alternates between sampling data and optimisation of the already sampled data.

The probability ratio is defined as

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|a_t)}. \tag{1}$$

In the Trust Region Policy Optimisation (TRPO) algorithm this is the objective function but subjected to a constraint on the policy update [6]. The objective function of PPO is

$$\hat{\mathbb{E}}_t\big[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\big], \tag{2}$$

where ϵ is a hyperparameter and $\hat{A}_t$ is an estimate of the advantage function at time step t [5]. This objective function gives the objective function of TRPO a lower bound and a penalty for large updates in the policy, without using a constraint.

The authors of the paper which introduces PPO claims that the algorithm has the stability and reliability of TRPO, which is a second order method, and is simpler to implement [5]. They also claim that the algorithm performs well on a collection of benchmark tasks, such as Atari.

3 Tools

3.1 Unity

A game engine is a software development kit which is used as a foundation for building different games [2]. Unity is a game engine for 2D and 3D game development. The game engine provide features as physics simulation, normal maps and dynamic shadows.

Unity provides an open source machine learning toolkit called ML-Agents [3]. Unity is utilised as a simulation platform where the , which allows for to provide the learning enviro By utilising Unity as a simulation platform the complexity of the learning

3.1.1 ML-Agents

ML-Agents is a machine learning platform provided by Unity [4].

4 Implementation

4.1 Environment

In order to investigate how well the models are able to generalise three environments were built in Unity. Each design was based upon a flat surface surrounded by walls, preventing the agent from falling of the surface. The environment had one target for the agent to reach. Once the target had been reached the agent and target was reset to one of 64 positions, taking care to not put them at the same place. The agent was given a positive reward once it reached the target. However the agent also received a small negative reward at each step that it was searching for the target.

The maze designed to be the easiest had one extra wall in the middle. Said wall had a length of half that of the surface and was centred in the middle. This lead to there being two ways past it one in the upper $\frac{1}{4}$ and one in the lower $\frac{1}{4}$. Image 1 below shows the design of the maze when looking at it from above, green is the target blue circle is the agent.

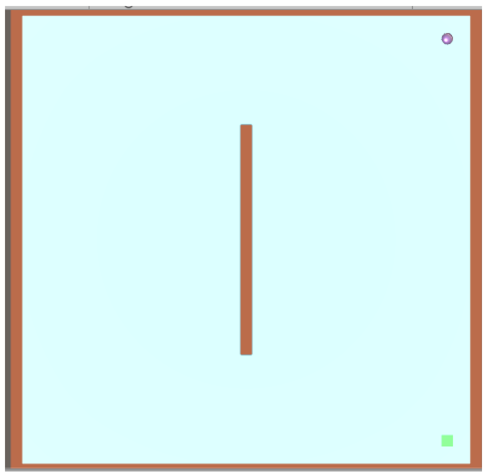

Figure 1: Maze designed to be easy

The intermediate was designed to be fairly challenging for the agent. Another four walls added, in addition to the middle wall in the easiest design. These four walls each had a length equal to $\frac{1}{4}$ of the surface. They where placed $\frac{1}{8}$ from the side walls and such that they created one small corridor in each corner. Figure 2 shows the design from above.

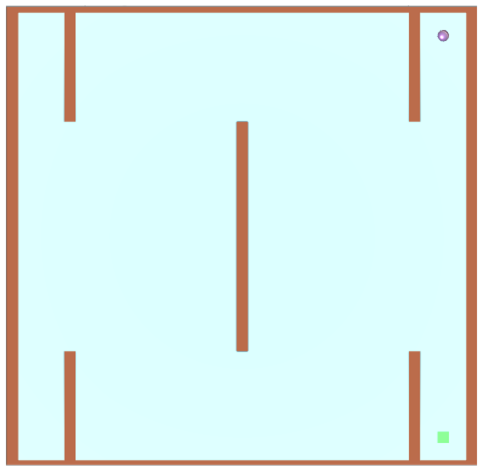
Figure 2: Maze designed to be difficult for the agent to solve

The third design meant to be the hardest had another two walls added to it, seven interior walls in total. The two new walls once again had a length equal to $\frac{1}{4}$ of the surface and was positioned at the middle of the side walls. The four walls which previously, in the intermediate maze, formed corridors where moved to a position $\frac{1}{4}$ away from the side walls. This then created four rooms, one in each corner, with the middle wall still there. The design is shown from above in figure 3 below.

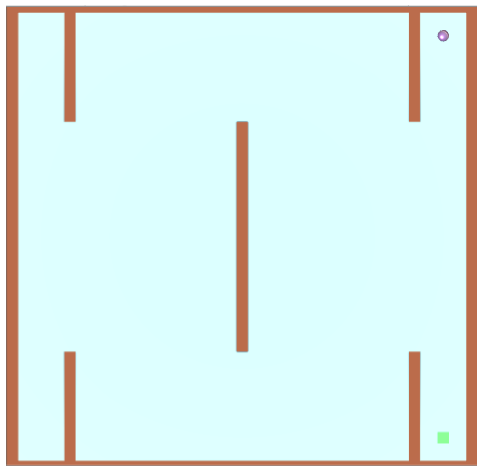
Figure 3: Maze designed to be hard for the agent to solve

4.2 Agent

An agent was trained on the environments presented in the section above, 4.1. The agent knew the position of the target, its own position and its own velocity. However do note that it knew the position of the target but not that it was the target, at least not before training. The state observation therefore had eight values. The state space was set to continuous, since position and velocities both are continuous in real life. The agent could, from its observations, control its velocity in the y and z direction in order to move around the environment.

The brain is the asset which processes the inputs in order to control the agent. It was trained by letting the agent explore the environment in order to find the target. The end result from a training is a Tensorflow model which can then be used for the decision making. However since the agent could get stuck in a corner, a wall or just in a local area the maximum amount of steps, after which the agent was reset, was set to 2000.

5 Results

6 Related Work

7 Discussion

7.1 Conclusions and Future Work

References

[1] Greg Brockman et al. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[2] Jason Gregory. *Game engine architecture*. 2nd ed. AK Peters/CRC Press, 2017.

[3] Arthur Juliani et al. "Unity: A General Platform for Intelligent Agents". In: *arXiv preprint arXiv: arXiv:1809.02627 [cs.LG]* (2018).

[4] Micheal Lanham. *Learn Unity ML-Agents - Fundamentals of Unity Machine Learning: Incorporate New Powerful ML Algorithms Such As Deep Reinforcement Learning for Games*. Birmingham: Packt Publishing, Limited, 2018.

[5] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[6] John Schulman et al. "Trust Region Policy Optimization". In: *arXiv preprint arXiv:1502.05477v5 [cs.LG]* (2015).

[7] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature 529, 484-489* (2016).

[8] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* 2nd ed. Cambridge, Mass: MIT press, 2018.

[9] Amy Zhang, Yuxin Wu, and Joelle Pineau. "Natural Environment Benchmarks for Reinforcement Learning". In: *arXiv preprint arXiv:1811.06032* (2018).

Appendix