Fredrik Gustafsson

Jakob Nyberg

Sara Hedar

UPPSALA
UNIVERSITET

Building an Environment and Training a Learning Agent in Unity

Natural Computation Methods for Machine Learning

Uppsala

May 16, 2019

Abstract

Hejsan jag är ett abstract.

Titelförslag: Building an Environment and Training an Agent in Unity using ML-Agents

Contents

Glossary

ML Machine Learning

RL Reinforcement Learning

PO Proximal Policy Optimisation

TRPO Trust Region Policy Optimisation

1 Introduction

We want to design programs which can learn. When designing an artificially intelligent system it is intuitive to look at the one source of intelligence humans know of, the natural world. An example of this are *perceptrons* which are an abstraction of nerve cells [7]. One way of learning is to imitate, which is how we learn language as children. Another way of learning is to simply interact with something until it becomes familiar, such as when riding a bike. It is difficult to learn how to ride a bike simply by imitation or reading. The way one learns is to simply try over and over. Learning to ride a bike in this way is analogous to how *reinforcement learning* works.

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 [12]. RL 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". 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 [12].

In 2016 an algorithm based on RL 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 [10]. 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 Machine Learning

According to Judith Hurwitz and Daniel Kirsch machine learning (ML) is the subject of devising and using algorithms to improve the accuracy of predictive models [3]. Three subfields of machine learning are mentioned below.

Supervised learning Is used to find patterns in labeled data. An example of a supervised learning problem would be classifying animals.

Unsupervised learning In which a ML algorithm tries to classify unlabelled data based on clusters. Social media utilises unsupervised learning as an example since it doesn't require labeling of the data.

Reinforcement learning The ML algorithm learning in this case by trial and error instead of being trainer with a data set.

2.2 Reinforcement Learning

Reinforcement learning can be thought of as learning what to do in a situation, learning by interaction. It is defined by characterising a problem and not the solution method [12]. In order to learn how to solve the problem the `agent` interacts with an `environment` and gets a reward. A reinforcement learning problem contains four main components according to R.Sutton and A. Barto [12].

Reward function What defines the goal of the problem. It is what maps each state, or state-action pair, to a reward, it is what defines good and bad actions. The reward is given for each action the agent takes and becomes an instant feedback.

Value function Defines how much rewards the agent can expect to accumulate in the future, how good is the state in the long run. While rewards are the direct inputs value is what matters when a choice of action is made, since an action leading to high value will lead to a higher reward. Values can only be estimated, and re estimated, from the sequence of observations that the agent makes over its training time.

Policy What determines how the agent acts in a situation. It is what maps the perceived state to the action which the agents takes in said state.

2.3 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 [14] 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.

2.4 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 [12]. 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.

2.5 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 [8]. 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 [9]. 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 [8]. 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 [8]. They also claim that the algorithm performs well on a collection of benchmark tasks, such as Atari.

3 Unity and ML-Agents

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

Unity provides an open source machine learning toolkit called ML-Agents [4]. ML-Agents utilises Unity as a platform for creating and interacting with simulated environments. A simulated environment is defined using the Unity Editor and linked C# scripts, a developer can then interact with the environment through a Python API. The toolkit also provides benchmark example environments and baseline algorithm. An example of such an environment is GridWorld and a benchmark algorithm is PPO.

A ML-Agents learning environment include the entities Agent, Brain and Academy [4]. The agent gathers observations from the environment and affects the environment by executing actions. Each agent is linked to a single Brain, but multiple agents can be connected to the same Brain. The Brain provides the Agent with a policy and thus is responsible for its decision making. The Academy is the component which is responsible for the coordination of the simulation and provide the communication with the Python API. A visualisation of the connection between the Learning Environment, the Python API, the Academy, Brains and Agents is provided in figure 1.

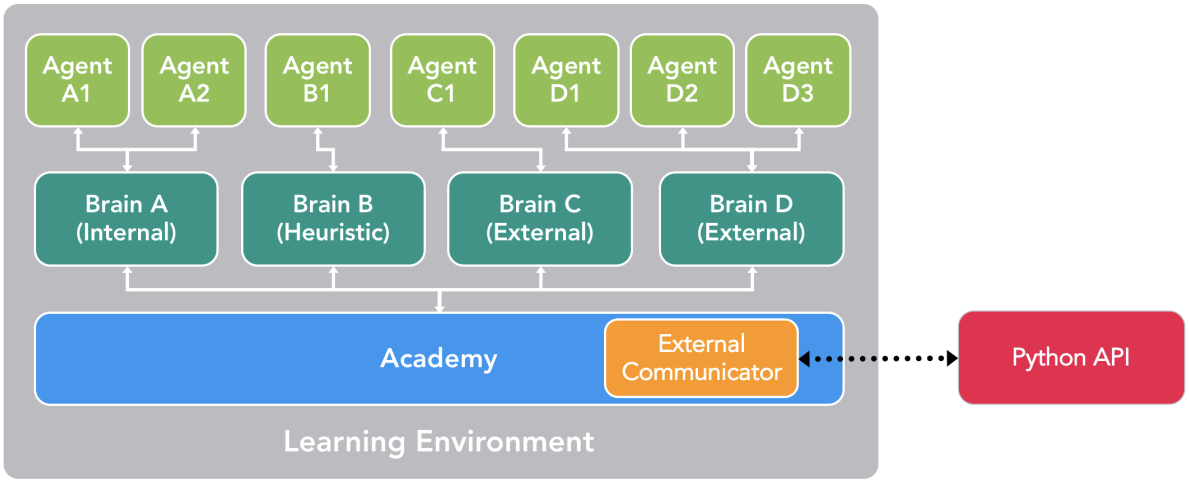

Figure 1: Visualisation of the connection of a Learning Environment, the Python API, the Academy, Brains and Agents [4].

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 was 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 $\frac{1}{4}$ part of the map and one in the lower $\frac{1}{4}$. Image **??** shows the design of the maze when looking at it from above, the green square is the target blue circle is the agent.

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 **??** shows the design from above.

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 **??** below.

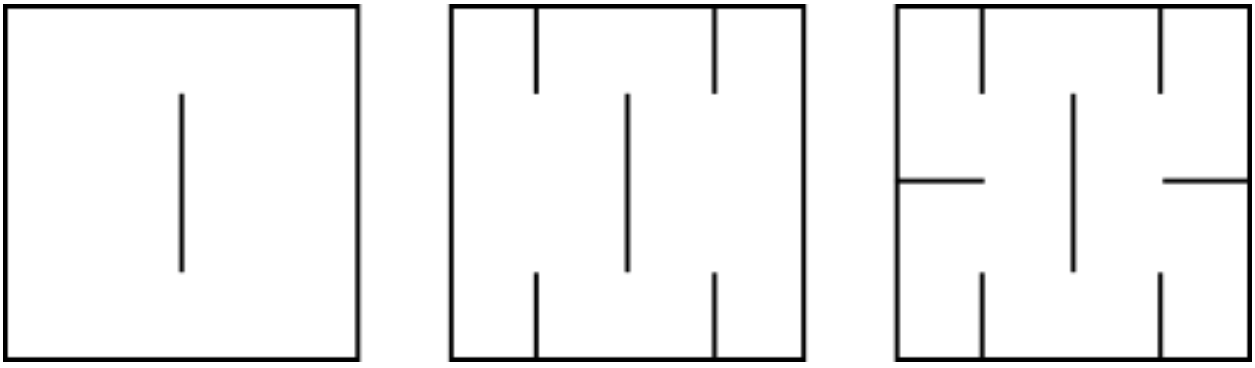

Figure 2: Maze designs

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, its own velocity and could sense its surroundings by utilising ray casting. However do note that it knew the position of the target but not that it was the target, at least not before training. When using ray casting the agent sends out rays in order to be able to detect objects. In this case the agent had to send out rays to detect the walls and the target. In order to decide in which directions the rays would be sent out the 360° surrounding was discretised in eight parts each covering 45°. The state observation therefore had 36 inputs. 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.

4.3 Hyperparameter Tuning

To train an agent capable of manoeuvring the defined learning environment the hyperparameters were tuned. This was done by changing one parameter at a time and keeping the value which resulted in the best performance. Five linearly spaced values on the intervals

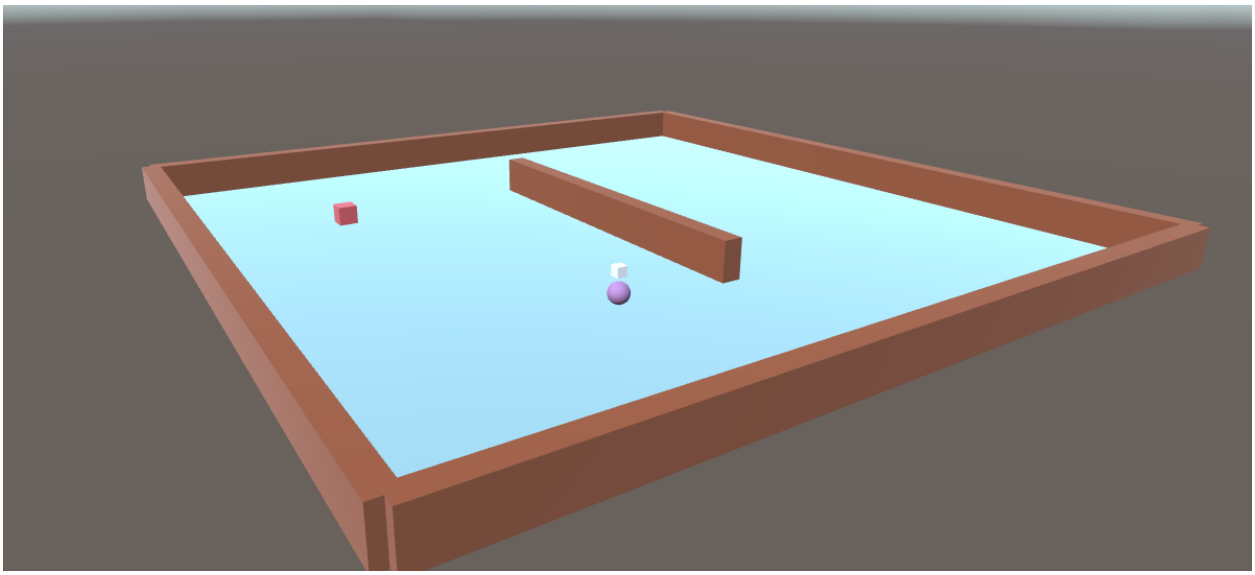

Figure 3: Instance of a single maze. The purple ball is the agent and the red cube is the goal.

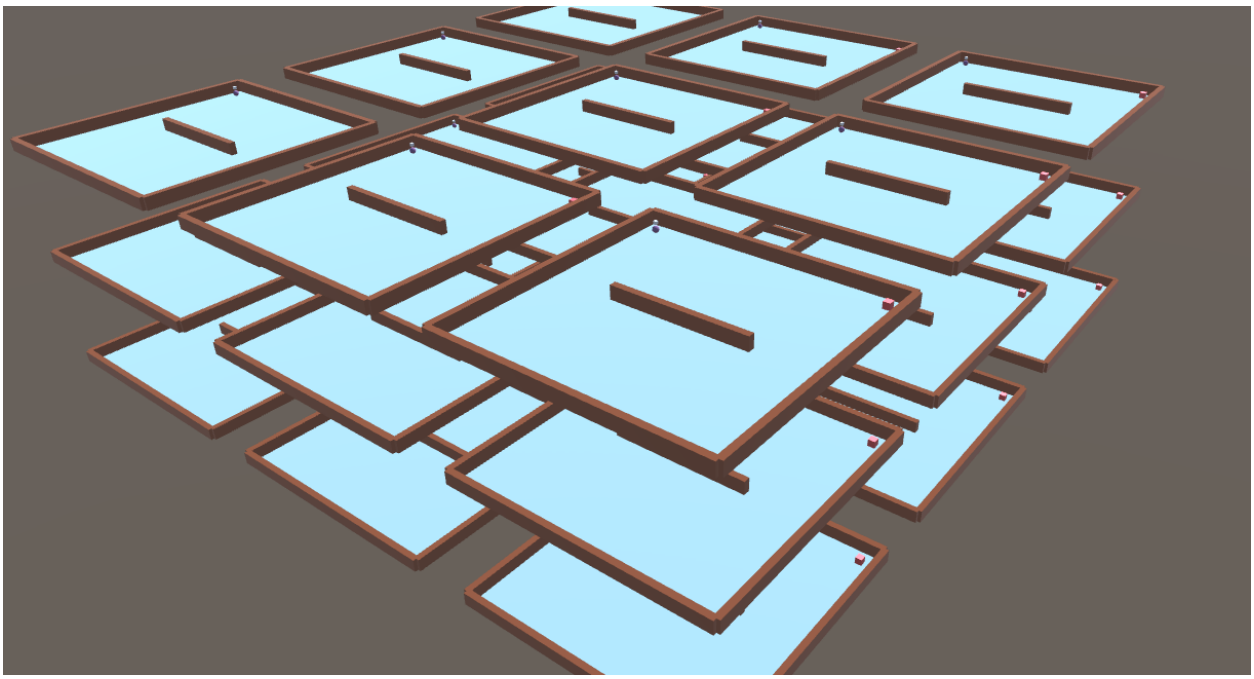

Figure 4: 27 mazes being used for training in parallel. The purple balls are agents and the red cubes are goals.

suggested in the documentation [5], for each parameters was tested. The exceptions being number of layers which had three test values and number of epochs whose interval was changed from 3-10 to 3-11. Batch Size and Buffer Size was also chosen to keep a fixed ratio. Since beta and the learning rate had intervals with difference on the scale of magnitudes five logarithmic spaced values was chosen.

Gamma Discount factor, how far into the future should the agent care about rewards. Suggested range: $0.8 - 0.995$. Default: 0.99.

Lamda The parameter which determines how much the agent uses its current value estimate when calculating an updated value estimate. Suggested range: $0.9 - 0.95$. Default: 0.95.

Buffer Size How many experiences (observations, actions and rewards) should be collected before updating the model. Suggested range: $2048 - 409600$. Default: 10240

Batch Size How many experiences used for each iteration of gradient descent update. Suggested range: $512 - 5120$. Default: 1024.

Number of Epochs The number of passes through the experience buffer during gradient descent. Suggested range: $3 - 10$. Default: 3.

Learning Rate The strength of each update step. Suggested range: $10^{-5} - 10^{-3}$. Default: 3.0e-04

Time Horizon How many steps, per agent, of experience to collect before adding it to the buffer. If the limit is reached before the end of the episode a value estimate is used to predict the total expected reward from the current state. Suggested range: $32 - 2048$. Default: 64.

Beta The strength or the entropy regularisation, higher value makes the policy more random. Suggested range: 1.0e-04-1.0e-02. Default: 5.0e-3.

Epsilon Is the threshold for the divergence between two policy updates. Suggested range: $0.1 - 0.3$. Default: 0.2.

Normalise If the observation vector inputs are normalised or not, the normalisation is based on the running average and variance. Default: False.

Number of Layers How many hidden layers are present in the model. Suggested range: $1 - 3$. Default: 2.

Hidden Units How many neurons each fully connected layer have. Suggested range: $32 - 512$. Default: 128.

5 Results

Order of testing: gamma, lambda, buffer/batch, number of layers

Table 1: The result from training sessions where the parameter Gamma was varied.

Gamma	0.8000	0.8488	0.8975	0.9463	0.9950
Accumulative Reward	-33	-33.6	-31.78	-32.51	-19.88

Table 2: The result from training sessions where the parameter Lambda was varied.

Lambda	0.9000	0.9125	0.9125	0.9250	0.9500
Accumulative Reward	-14.30	-8.152	-33.3	-11.58	-22.02

Table 3: The result from training sessions where the batch and buffer size was varied.

Buffer/Batch	5120/512	16640/1664	28160/2816	39680/3968	51200/5120
Accumulative Reward	-28.45	-16.84	-31.21	-31.99	-30.77

As all of the tested combination gave a lower accumulated reward than the default value 10240/1024 the default value was kept.

Table 4: The result from training sessions where the number of hidden layers was varied.

Number of Layers	1	2	3
Accumulative Reward	-26.62	-6.576	-19.35

Table 5: The result from training sessions where the number of epochs was varied.

Epochs	3	5	7	9	11
Accumulative Reward	-	-	-	-	-

Table 6: The result from training sessions where the number of hidden units per fully connected layer was varied.

Number of hidden units	32	152	272	392	512
Accumulative Reward	-	-	-	-	-

Table 7: Learning Rate

Learning Rate	1.000e-05	3.160e-05	1.000e-04	3.162e-04	1.000e-03
Accumulative Reward	-	-	-	-	-

Table 8: Time Horizon

Learning Rate	32	536	1040	1544	2048
Accumulative Reward	-	-	-	-	-

Table 9: β

β	1.0e-04	3.0e-04	1.0e-03	3.2e-03	1.0e-2
Accumulative Reward	-	-	-	-	-

Table 10: ε

ε	0.01	0.15	0.20	0.25	0.30
Accumulative Reward	-	-	-	-	-

Table 11: Normalisation

Normalisation	Ture	False
Accumulative Reward	-	-

6 Related Work

The idea to implement a environment and train an agent came from the recent advances within the field with agents trained to play video games such as Super Mario [6].

MuJoCo is another engine which has been used for reinforcement learning with as an example the DeepMind Control Suite [13]. However MuJoCo requires a license to use and is not open source. Another way to implement environments used for reinforcement is to use OpenAi Roboschool which has another physics engine but environments similar to those found in MuJoCo [11].

PRatar vi om att träna saker så finns dessa exempel

MuJuCo is another tooolkit for machine learnign

7 Discussion

7.1 Conclusions and Future Work

We have successfully set up an learning environment using Unity which consists of three mazes with a goal and a ball. We have managed to train the ball to reach the goal.

We have also set up a semi-automatic system of training where we can test many different parameters.

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] Judith Hurwitz and Daniel Kirsch. *Machine Learning For Dummies®,IBM Limited Edition*. Hoboken, NJ: John Wiley & Sons, Inc, 2018.

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

[5] A. Juliani et al. *Training with Proximal Policy Optimization*. https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-PPO.md. 2018.

[6] Fabio Pardo, Vitaly Levdik, and Petar Kormushev. "Goal-oriented Trajectories for Efficient Exploration". In: *arXiv preprint arXiv: arXiv:1807.02078 [cs.LG]* (2018).

[7] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

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

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

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

[11] Mario Srouji, Jiang Zhang, and Ruslan Salakhutdinov. "Structured Control Nets for Deep Reinforcement Learning". In: *arXiv preprint arXiv: arXiv:1802.0311 [cs.LG]* (2018).

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

[13] Yuval Tassa et al. "DeepMind Control Suite". In: *arXiv preprint arXiv: arXiv:1801.00690 [cs.AI]* (2018).

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

Appendix