Optimizing UAV control through reinforcement learning

Samuel Knight

MInf Project (Part 2) Report

Master of Informatics
School of Informatics
University of Edinburgh

2020

Abstract

Reinforcement learning has typically focused on training agents to solve complex games. Finding novel tasks to bridge the gap between theoretical and physical applications is an important pursuit. To this end, this paper explores the application of reinforcement learning to a simulated task involving the control of base-stations mounted on unmanned aerial vehicles (UAVs). The task is motivated by the real problem of providing network coverage after disaster scenarios. Intelligent control of the UAV base stations is required to achieve reliable network coverage for the users on the ground. Comparing actor-critic and off-policy deep Q-learning approaches, all algorithms consistently outperform a heuristic benchmark. The deep Q-learning approach outperforms the actor-critic methods and achieves a median SINR increase of 9 dB compared to the baseline.

Acknowledgements

Thank you to all who helped make this possible.

Table of Contents

1 Introduction **7**

2 Related Work **9**
- 2.1 Reinforcement Learning . . . 9
- 2.2 UAV Control . . . 10

3 Reinforcement Learning Primer **13**
- 3.1 Value and Policy Based Methods . . . 14
- 3.2 Deep Reinforcement Learning . . . 15
- 3.3 Algorithms . . . 16
  - 3.3.1 A3C . . . 16
  - 3.3.2 A2C . . . 17
  - 3.3.3 ADQN . . . 17

4 Simulation Environment **19**
- 4.1 State Space . . . 20
- 4.2 Action Space . . . 20
- 4.3 Network Model . . . 20
- 4.4 User Mobility Model . . . 21

5 Methodology **23**
- 5.1 Algorithm Implementation . . . 23
- 5.2 Network Architecture . . . 24
- 5.3 Reward Function . . . 24
- 5.4 Training . . . 26
- 5.5 Testing . . . 26

6 Performance Evaluation **27**
- 6.1 Greedy Baseline . . . 27
- 6.2 A3C . . . 27
  - 6.2.1 CartPole . . . 28
  - 6.2.2 UAV Environment . . . 29
- 6.3 A2C . . . 30
  - 6.3.1 CartPole . . . 30
  - 6.3.2 UAV Environment . . . 31
- 6.4 ADQN . . . 32

6.4.1 CartPole 32
6.4.2 UAV Environment 33
6.5 Evaluation 35

7 Conclusion **39**
7.1 Future work 39

Bibliography **41**

A Learning Algorithms **45**

Chapter 1

Introduction

With the rise in availability and affordability of drones, or "unmanned aerial vehicles" (UAVs), many applications for these devices have been established ranging from photography (Li and Yang, 2012) to package delivery (Anbaroglu, 2017). An important application of UAVs, stemming from their high mobility, is their usage in disaster scenarios. In this context, UAVs can serve as mobile base stations providing wireless connectivity, and consequently a method of communication, to victims and first responders after a disaster.

The focus of this paper is the application of reinforcement learning algorithms on a simulated domain where UAVs aim to provide network coverage to mobile users. Reinforcement learning has traditionally been applied to solve games. Algorithms have been shown to achieve super-human performance on a number of tasks ranging in complexity from Backgammon to Go (Tesauro, 1995) (Silver et al., 2017). The application of reinforcement learning algorithms to real-world problems is less popular, as the assumptions made by these algorithms do not often hold in real-world settings. Efforts to bridge the gap between toy simulations and physical applications are therefore important for progress in the area. The simulated environment, developed by Li et al. (2019), consists of a number of UAV base stations which provide network coverage to users "on-the-ground". The users move around according to a motion model which aims to replicate the behaviour of users in disaster scenarios. The actions taken by the base stations constitute the learned control policy of a reinforcement learning algorithm, and can be optimized through a number of approaches.

In Li et al. (2019) an actor-critic reinforcement learning approach was used to learn the control policy. They chose the asynchronous advantage actor-critic (A3C) algorithm (Mnih et al., 2016) due to its low memory and hardware costs: it can train on a single CPU rather than multiple GPUs. A3C showed superiority to a greedy baseline achieving a median SINR increase of 5 dB. In disaster scenarios, the ability to achieve reliable network connections for those affected is key and any increase in the signal quality could improve these connections (Tiefenbacher, 2019). To this end, this paper implements a number of improvements to the learning algorithm specifically, updating its implementation to a new machine learning framework and removing asynchronous updates – this approach is called advantage actor critic (A2C). These variations have

been shown to outperform the standard A3C algorithm (Dhariwal et al., 2017). Additionally an off-policy learning algorithm, asynchronous deep Q-learning (ADQN) (Mnih et al., 2016), has been implemented as another baseline to evaluate against the on-policy actor-critic methods.

It is important to identify any properties of the environment that can have an impact on the complexity of the reinforcement learning problem. As both UAV base stations and users change the state of the environment through their actions, and the reinforcement learning algorithm controls only the actions of the base stations, the task should be understood as a multi-agent problem. Many reinforcement learning algorithms, including those used in this paper, have convergence properties that depend on core assumptions. Multi-agent problems can conflict with these assumptions leading to difficulties in the learning process. The assumptions and the problems that occur when they are broken will be explained in Chapter 3. Re-framing this domain as a multi-agent problem and identifying the difficulties associated with this will hopefully encourage diligence when applying reinforcement learning algorithms to similar environments.

The implementations of all algorithms investigated in this paper have been open-sourced as a reference to facilitate further research, and interest, in deep reinforcement learning. The code has been written in Python 3, using a cutting-edge version of Google's popular machine learning framework, Tensorflow (Abadi et al., 2015). The updated framework, Tensorflow 2, offers a functional API which greatly increases the readability and structure of machine learning implementations compared to its older, declarative API. The reference implementations interface directly with OpenAI's gym framework: a common interface into a number of reinforcement learning environments, ranging from toy problems to complex continuous control tasks (Brockman et al., 2016).

The achievements of this paper are as follows:

- Implemented A3C, A2C and ADQN algorithms.
- Open-sourced reference implementations for the above algorithms.
- Identified the multi-agent properties of the environment. Presented complications that this can cause on learning performance.
- Evaluated the performance of the algorithms on the UAV domain.

All the code written for the paper can be found at the following link:
`https://github.com/SamKnightGit/DRL_UAV_CellularNet/tree/py3`

Chapter 2

Related Work

2.1 Reinforcement Learning

Reinforcement learning can broadly be categorized into two approaches: policy optimization and value optimization. Policy optimization involves directly updating the policy using a quantifiable measure of the policy's utility while value optimization involves learning the value of states in the environment and taking actions to maximize this value. Policy gradient approaches – policy optimization using the gradient of rewards with respect to policy parameters – were heavily popularized in Sutton et al. (1999). The paper proved that policy optimization, given appropriate function approximation, will converge to a locally optimum policy. Sutton et al. (1999) also laid the groundwork for modern actor-critic algorithms suggesting that the policy can be represented by a function approximator, specifically a neural network with outputs representing action probabilities.

Modern policy gradient methods include Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) and Proximal Policy Optimization (PPO) (Schulman et al., 2017). TRPO presents a surrogate objective function which, when minimized, can guarantee monotonic policy improvement. Schulman et al. (2015) demonstrate that TRPO performs well on a variety of simulated tasks without the need for hyperparameter optimization. Schulman et al. (2017) introduce a simpler policy gradient method, PPO, which aims to capture the benefits of TRPO while achieving better sample complexity. PPO uses a clipped surrogate objective function which punishes large, potentially disruptive, policy updates to promote convergence. PPO achieves better performance than TRPO and a number of other policy gradient algorithms on many continuous control environments while requiring lower training times.

Deep reinforcement learning has risen in popularity since Deepmind's paper on Deep Q Networks (DQN) (Mnih et al., 2013). Implementing a reinforcement learning framework which utilized deep neural networks, they achieved state-of-the-art performance on many virtual Atari tasks using raw pixels as inputs. To stabilize learning by reducing the correlation of updates, Mnih et al. (2013) used an experience replay buffer to sample experiences randomly during training. Another method of decorrelation was

proposed in Mnih et al. (2016): using a number of asynchronous learning agents in parallel environments. This approach avoided the limitations of experience replay, namely high memory cost and the requirement of off-policy learning algorithms. This paper also introduced the asynchronous advantage actor-critic (A3C) algorithm, which utilized an actor-critic architecture to train agents with asynchronous updates.

The actor-critic architecture has been utilized in many modern deep reinforcement learning, policy gradient approaches. Wang et al. (2016) combines actor-critic deep reinforcement learning with the aforementioned experience replay, introducing actor-critic with experience replay (ACER). Experience replay is added to reduce the high sample complexity of actor-critic methods and is shown to outperform A3C in terms of task completion and sample efficiency on both discrete and continuous environments (Wang et al., 2016). Haarnoja et al. (2018) propose using an off-policy actor-critic algorithm which uses a maximum entropy approach, pushing the learning agent to achieve high performance while acting as randomly as possible. This method achieves state-of-the-art performance on a number of continuous control benchmarks as well as achieving high stability across different environments. Finally, Gu et al. (2016) address the high sample complexity of deep reinforcement learning approaches by proposing Q-Prop, an algorithm which combines the benefits of on-policy and off-policy methods. Using an off-policy critic, Q-Prop reduces the variance inherent in policy-gradient approaches, lowering sample complexity. Empirically, Q-Prop demonstrates increased sample efficiency and stability compared to TRPO and Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2016), a policy gradient algorithm, in a number of continuous control tasks respectively.

2.2 UAV Control

Autonomous unmanned aerial vehicles (UAV) navigation and control has become a popular research area with the rise in availability of commodity UAVs. Pham et al. (2018) applied reinforcement learning to a physical UAV navigation task in an unknown environment. They propose a variant of the Q-learning algorithm (Watkins and Dayan, 1992), which incorporates the use of a PID controller for executing the actions output by the learning agent. Koch et al. (2018) note that PID controllers do not perform optimally in unpredictable environments and investigate the use of modern reinforcement learning algorithms in a custom simulated environment. Implementing and evaluating PPO, TRPO and DDPG on a continuous control task, the paper concludes that while TRPO and DDPG suffer from unstable oscillations, PPO outperforms the PID controller (Koch et al., 2018).

Deep reinforcement learning approaches have also been applied to the UAV control task. Huang et al. (2019) consider UAVs acting as wireless base stations and train a DQN for optimizing UAV navigation. Taking inputs from a massive multiple input multiple output (MIMO) the DQN outperforms several benchmark control strategies. Li et al. (2017) combine the policy-gradient DDPG algorithm with a PID controller

to train a control policy for following a target. To balance the inherent instability of the RL algorithm on a real-world environment they propose a novel architecture: the policy network outputs high level actions which are mapped to low-level UAV commands by the PID controller (Li et al., 2017). Additionally, to reduce instability in the learning process, supervised pre-training of the convolutional neural network layers used for perception is implemented. A comparison of pre-trained vs non pre-trained approaches shows that this bootstrapping of the policy network leads to faster and more stable convergence.

This work expands on the research done by Li et al. (2019) in which they implement the A3C algorithm to train a UAV control policy in a simulated environment. Li et al. (2019) report that the reinforcement learning approach exceeds the performance of a greedy benchmark. The performance of the algorithms is quantified using a cumulative distribution function of the SINR attained by all users. The A3C approach is reported to achieve a 5dB higher median SINR than the benchmark.

This dissertation aims to address two main limitations of the research done by Li et al. (2019): that only a single learning algorithm was explored and that the paper does not explicitly address the multi-agent aspects of the environment in relation to learning. The performance of different types of learning algorithms on unique domains can elucidate aspects of the algorithms and the environment that were not previously considered (Hausknecht and Stone, 2016). To this end, A2C and ADQN algorithms are implemented to explore the performance differences of off-policy and on-policy algorithms on the domain. Additionally, the multi agent aspects of the environment are explicitly addressed and the consequences these have on convergence guarantees are explained.

Chapter 3

Reinforcement Learning Primer

Reinforcement learning (RL) is the pursuit of optimal behaviour through interaction with an environment. Typically, an agent or autonomous learner will receive feedback for its action in the form of a "reward" which guides the agent towards a goal. At each moment in time (a time-step), the agent perceives the state of the environment, receives a reward for the previous action it took and issues an action for the current time-step. States, actions and rewards are often stochastic quantities, complicating the learning process. The RL problem is commonly formulated as a finite Markov decision process (MDP) which can be fundamentally described by the following property:

Given the current state (s) and action (a) the probability of the next state (s') and its associated reward (r) is conditionally independent of all previous states and actions:

$$p(s', r \,|\, s, a) = p(s_{t+1} = s', r_{t+1} = r \,|\, s_t = s, a_t = a) \tag{3.1}$$

Where t represents the time-step.

An MDP typically consists of the 4-tuple $< \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P} >$:

- $\mathcal{S}$ is the set of states of the environment
- $\mathcal{A}$ is the set of actions the agent can take
- $\mathcal{R}$ is the reward function, outputting the reward for taking action a in state s. $\mathcal{R}(s,a) = \mathbb{E}[r_{t+1} | s_t = s, a_t = a]$
- $\mathcal{P}$ is the transition function, outputting the probability of transitioning from state s to s' when taking action a. $\mathcal{P}(s,a,s') = \mathbb{P}[s_{t+1} | s_t = s, a_t = a]$

In many environments, including the UAV environment investigated in this paper, actions are deterministic meaning that the transition function is trivial: for a given state-action pair, the function will always map to the same next state with a probability of 1.

The formulation of RL problems into an MDP provides a framework for evaluating the convergence guarantees of various algorithms. In many environments, especially

ones involving multiple agents, the fundamental Markov assumption is broken (Laurent et al., 2011). In the UAV environment, the state space represents the locations of the base stations and users. However, as the agent only controls the actions of the base stations (and knows nothing about the mobility model of the users) the Markov assumption is broken: for a given state-action pair, the next state could differ depending on the current configuration of the users' mobility model. This assumption breaking, while important to keep in mind when investigating learning and convergence in multi-agent environments, does not prevent learning on the domain. To the contrary, many reinforcement learning algorithms which break the Markov assumption have successfully solved tasks ranging in complexity from block-pushing (Sen et al., 1994) to elevator control (Crites and Barto, 1995).

Behaving optimally, from a RL perspective, can be achieved by maximizing the expected sum of future rewards. In order to do this an agent must learn to quantify the expected rewards of states and actions and take actions to maximize this. An agent's policy, π represents a probability distribution over actions given states, and fully describes its behaviour in the environment. The two main types of approaches for maximizing future rewards are value based and policy based methods.

3.1 Value and Policy Based Methods

Value based reinforcement learning relies heavily on the notion of a Q-Value or Action-Value. The Q-Value function, $Q^{\pi}(s,a)$, gives a measure of the expected reward for taking action a, in state s and following the policy π thereafter. Typically the Q-Value function is represented as a neural network with parameters θ. By exploring the environment and receiving rewards the network aims to converge to the optimal Q-Value function. The network "learns" by minimizing a loss which captures the distance between the current Q-Value estimate and an approximation of the optimal estimate. A value based reinforcement learning agent formulates a policy which selects the action with the highest Q-Value in a given state, referred to as the greedy policy. The inability of the agent to learn a stochastic policy is one of the main downsides of pure value based learning.

Policy based reinforcement learning aims to maximize expected rewards by directly optimizing the agent's policy, π. Similar to value based methods a function approximator such as a neural network with parameters θ is often used to represent the policy. To update the policy towards optimal behaviour, a function must be used to quantify and compare the performance of various policies. Sutton et al. (1999) propose an example policy update using:

$$\Delta\theta \approx \alpha\frac{\delta\rho}{\delta\theta} \tag{3.2}$$

Where α is a step-size parameter and ρ a policy performance measure

Contrary to value based learning, one of the main advantages to policy based learning is the resulting stochastic policy. A stochastic policy allows an agent to perform optimally in environments where stochastic actions are required as well as those in

which deterministic actions are optimal. To understand why the latter case is true, consider an environment where an agent can take one of two actions: move up or move down. If the agent is rewarded for moving up, a successful policy can be represented deterministically, always move up, as well as stochastically, move up with probability 0.999 repeating.

Actor-critic methods combine value based and policy based approaches into a unique framework. The actor represents the policy of the agent while the critic represents the learned value function. The critic is updated as described in the section above, while the actor utilizes the critics value estimate to scale its policy update. Modern actor-critic approaches employ a neural network function approximator to parameterize the actor and the critic. Practically, as is suggested by Mnih et al. (2016), it is common to use a shared network with two separate output layers, for the policy and value respectively.

3.2 Deep Reinforcement Learning

Deep reinforcement learning is the usage of deep neural networks in a reinforcement learning problem. Deep RL was first popularized by Mnih et al. (2013) where a convolutional neural network (CNN) was used to approximate the Q-Value function for a number of atari games. In the DQN algorithm, the network receives state inputs and outputs the Q-Value of each possible action from the current state, see Figure 3.1. The depth of the network allows it to capture non-linearity, finding subtleties in the input that have an impact on the resulting Q-values.

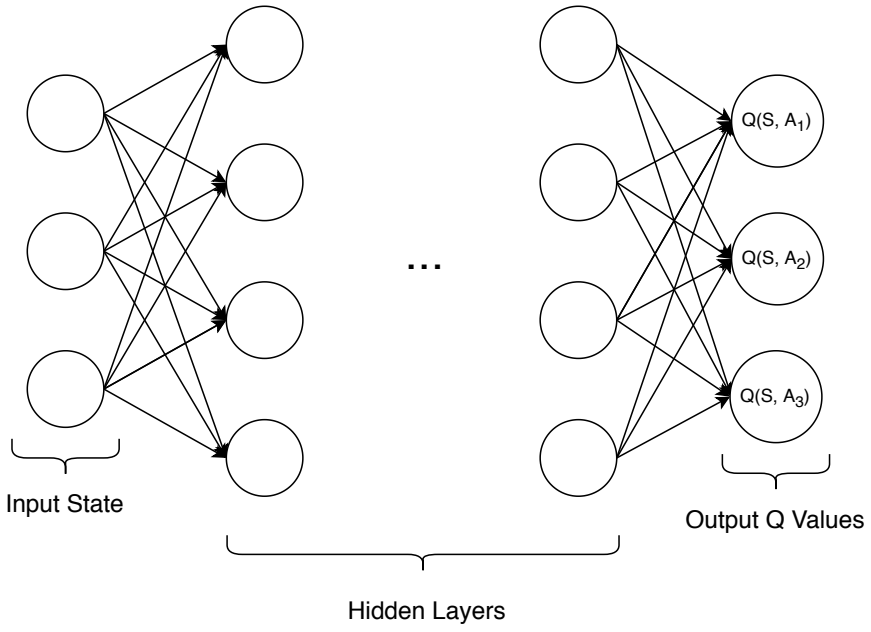


Figure 3.1: An example deep Q network modelled in multi-layer perceptron style. Outputs are Q Values for each possible action given the current state as input.

One of the primary issues faced by neural networks in the context of RL is that they

cannot decorrelate inputs. This manifests as a problem when training on an environment where varied sequential actions are required to accomplish a goal. The neural network is repeatedly updating its parameters as it trains in the environment. The most recent experiences therefore bias the parameters the most. We can highlight this problem by considering a simple environment where an agent must move up in the first time-step and then right in the following five time-steps to reach the goal. Having reached the goal, the network will be heavily biased towards moving right due to its recent experiences and upon resetting the environment will likely try to move right in the initial state even though this is sub-optimal.

Mnih et al. (2013) handle this by using a replay buffer, a memory of state, action, reward tuples which are sampled from randomly to train the network. This allows decorrelation of inputs at an increased memory cost. Another method for decorrelating inputs is with an asynchronous approach, as proposed by Mnih et al. (2016). By instantiating a number of parallel environments, where agents update a shared global network asynchronously, the network receives a more generalized set of inputs to train on. Asynchronous methods avoid the memory and computational burden of traditional DQN approaches and can be trained on commodity CPUs (Mnih et al., 2016).

3.3 Algorithms

All the algorithms explored in this paper utilize deep reinforcement learning. However, the architecture of the neural networks and the way the networks are optimized differ across the algorithms. A3C and ADQN follow an asynchronous approach, while A2C is synchronous. The general structure of these two approaches is laid out in Figure 3.2 and Figure 3.3 respectively. The full pseudocode for each algorithm is provided in appendix A.

3.3.1 A3C

A3C stands for asynchronous advantage actor-critic. It is an actor-critic approach which uses a number of workers exploring the environment in parallel (asynchronously) to learn. As shown in 3.2, each asynchronous worker holds its own local copy of the actor and critic network and the environment. The workers take actions dictated by their actor networks and calculate policy gradients based on the values provided by the critic.

The policy gradients in A3C are scaled by an estimation of the value of each state (Mnih et al., 2016). The quantity used for this scaling is called the advantage and is calculated as follows:

$$A(a_t, s_t) = Q(a_t, s_t) - V(s_t) \tag{3.3}$$

The advantage measure represents the increased benefit of taking the action a from the state s compared to the average value from that state. Using the advantage rather than

the raw value produced by the critic reduces the variance of the estimated return, while keeping the estimate unbiased (Mnih et al., 2016).

After a set number of time-steps the local workers send their gradients to the global network for updating. After the global network has updated its weights based on the local gradients, its weights are copied to the local network of the worker. This ensures that after updating, a worker has the most up-to-date copy of the global network, and implicitly the cumulative information of all other workers up to that point. One important bi-product of the asynchronous updates is that one worker's update can overwrite the work done by another (Mnih et al., 2016). However, given each agent is exploring its own copy of the environment, and will therefore likely be experiencing different states, overwriting updates seldom occur.

3.3.2 A2C

A2C, advantage actor-critic, is the synchronous variant of A3C. One of the key downsides of A3C is that the local networks across the workers often hold out-of-date, or stale, weights. Stale weights occur as the workers only update their local networks periodically and therefore only get access to the information from other workers at this time. A2C, as visualised in Figure 3.3, uses a single coordinator which holds multiple copies of the environment. At each time-step it takes actions across all the environments and updates the global network based on the resulting gradients. Thus, a diverse range of experiences is still available during learning. Additionally, as the coordinator holds the global network, the network always acts with the most up-to-date weights possible.

3.3.3 ADQN

Asynchronous deep Q network (ADQN) implement a variant of the classic DQN approach that first popularized deep reinforcement learning (Mnih et al., 2013). On an architectural level the ADQN algorithm functions identically to A3C, with parallel workers updating a global network asynchronously. However, ADQN differs by using a single neural network to approximate the Q function. Instead of a critic which provides information on how the policy should be updated, ADQN uses a target network. The ADQN target network is used to calculate the main network's loss by acting as a predictor of the "true" Q value at any given state (Mnih et al., 2016).

In order to calculate the target Q-Value the following canonical formula is used:

$$Q_{target} = r_{t+1} + \gamma \times \max_{a} Q(s_{t+1}, a; \theta^{target}) \tag{3.4}$$

Here γ represents a discount factor and θ parameterizes a neural network.

However, Hasselt et al. (2015) proved both theoretically and empirically that the canonical formula is prone to overestimation of Q-values. This is due to the maximum operator which uses the same value for evaluation and selection of an action. They propose an alternative method which selects actions using the value from the main network and evaluates the action using the value from the target network. This manifests in the following update formula:

$$Q_{target} = r_{t+1} + \gamma \times Q(s_{t+1}, \underset{a}{\operatorname{argmax}}\, Q(s_{t+1}, a; \theta^{main}); \theta^{target}) \tag{3.5}$$

Q-learning following the update rule described in Equation 3.5 was coined Double Q-learning by Hasselt et al. (2015). This Double Q-learning update was used in the ADQN algorithm due to its demonstrated performance benefits compared to the original Q-learning update (Hasselt et al., 2015).

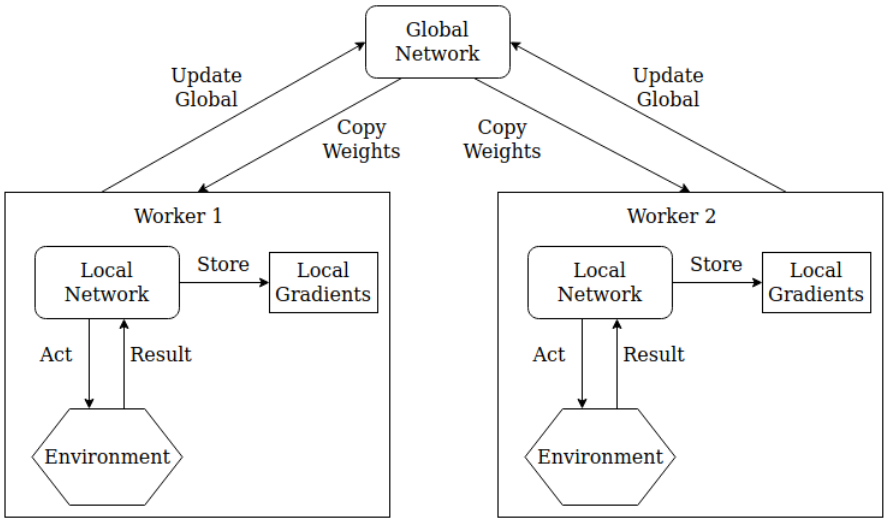


Figure 3.2: The asynchronous architecture used by A3C and ADQN algorithms assuming two asynchronous workers. The architecture functions with an arbitrary number of workers.

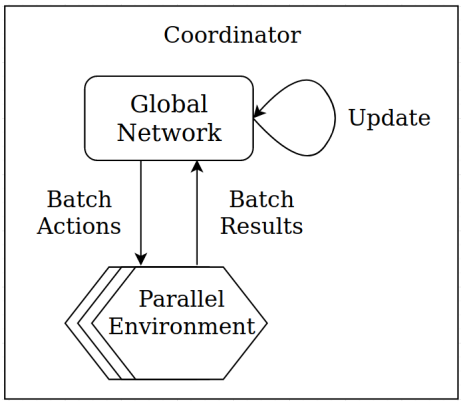


Figure 3.3: The synchronous architecture used by the A2C algorithm. A single coordinator updates the network via exploration of a set of parallel environments.

Chapter 4

Simulation Environment

The simulation environment introduced by Li et al. (2019) consists of moving UAVs and users, and models the network connectivity between these entities. The UAVs are situated at a fixed distance of 10m above the users. At each time-step the UAVs and users can move in one of four cardinal directions or remain where they are. All actors in the environment execute their actions simultaneously: in a single timestep there is no ordering of the actions taken by the different actors. The UAVs take actions based on the policy learned by a reinforcement learning algorithm while the users move according to a defined mobility model.

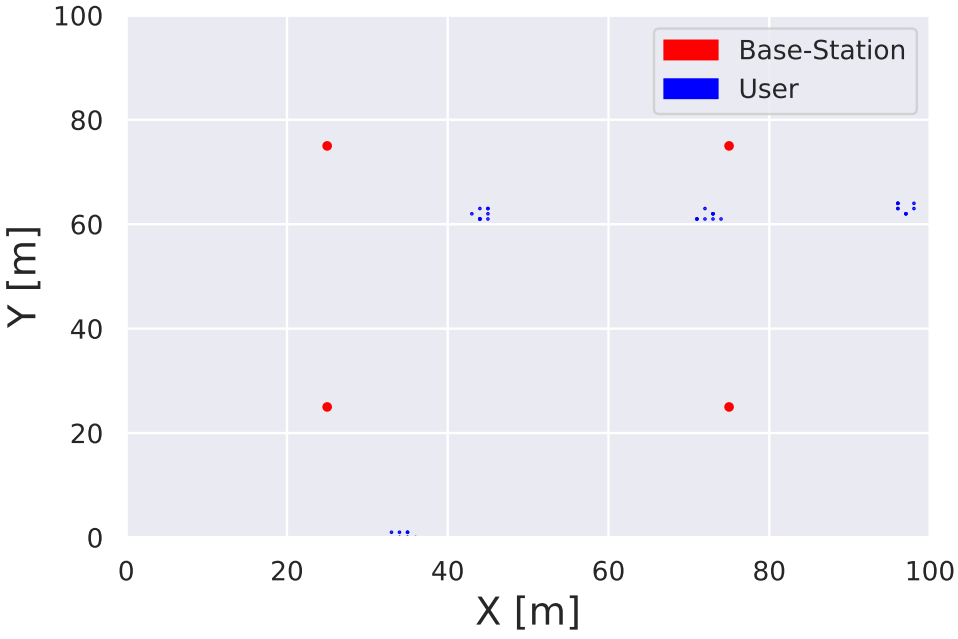


Figure 4.1: A snapshot of the UAV environment with 4 base stations and 4 user-groups

4.1 State Space

The state space is a representation of the environment that the reinforcement learning algorithm receives at each time-step. Typically, state spaces with higher dimensionality convey more information about the environment to the algorithm at the cost of longer training times. In this paper the state space was chosen in an attempt to maximize spatial information whilst preserving training efficiency.

The state space consists of two occupancy grids which represent the locations of the base stations and users respectively. Each occupancy grid is an N by N matrix where N represents the length and width of the arena. Each cell in the grid has a value of 1 if it is occupied by a UAV or user and a value of 0 if it is empty. By providing both UAV and user locations in a grid structure, the spatial relationships between states are explicitly captured.

The relationship between the size of the arena and the state space result in a performance bottleneck. Increasing the arena size, results in quadratic growth in the state space, which in turn increases the number of parameters in the neural network. A larger weight space requires more training time to find solutions to the task. For this reason the arena size was constrained to 100m.

4.2 Action Space

The actions of the UAVs are chosen from the four cardinal directions and the option to remain stationary. These movements are conducive to the grid nature of the environment. Given 5 possible movements for each UAV, the action space is given by 5^n where n is the number of UAVs. The exponential growth of the action space with the number of UAVs necessitates a limited number of UAVs for the learning task.

4.3 Network Model

The network is modeled on the communication between base stations "mounted" on each UAV and users who act as clients to these base stations. The base stations implement a bare-bones LTE protocol to simplify the network model. The network assumes an appropriate routing mechanism is implemented between the base stations. The core measure of the quality of signals between a user and a base station is the down-link signal to noise plus interference ratio (SINR). For a particular user, u, this quantity is defined as:

$$SINR(u) = \frac{P}{N+I} \tag{4.1}$$

Where P is the power of the incoming signal, N is the noise of the signal and I is the interference power of other base station signals.

The power terms in the above equation can be further decomposed into the transmit power of the base station and the channel gain between the base station and the user. The channel gain is calculated by taking a linear combination of the antenna gain,

shadow fading and the path loss. For simplicity the path loss is chosen to be a free-space model, assuming that there exists a line-of-sight, obstacle-free path between the user and the base stations. The path loss L is computed following the 3GPP cellular model for urban scenarios:

$$L = \alpha + \beta \log(D) \tag{4.2}$$

Where D represents the Euclidean distance between the base station and the user, and α and β represent standard coefficients.

4.4 User Mobility Model

The users move according to a defined mobility model. First introduced by Hong et al. (1999), the Reference Point Group Mobility model (RPGM) allows for situations such as large-scale disaster recovery to be modelled accurately. This model clusters users together in groups with each user moving around their respective groups' center-point (or reference). Every time-step each user is moved by the summation of the direction vector of the group reference and a random motion vector. This leads to random motion within groups, with the groups themselves moving cohesively. RPGM is a more structured mobility model than simple random-walk type models and allows for a more realistic way to test the deployment of ad hoc networks, such as the network created by the UAV base stations (Hong et al., 1999).

The number of user "groups" are made equal to the number of UAVs in the environment. This choice allows for a one-to-one mapping between UAVs and user groups, hopefully eliciting maximum coverage scenarios where each UAV learns to situate itself at a unique group's center-point. Group way-points are generated randomly, resulting in unique movement of the user groups across episodes.

Chapter 5

Methodology

The first step towards the analysis of reinforcement learning on the UAV domain was the implementation of the relevant algorithms. Surveying the current state-of-the-art, the tools created by OpenAI are well tested and provide useful functionality for the development and testing of learning algorithms. OpenAI, a research organization focusing on artificial intelligence, provides a suite of open-source learning environments and reference implementations of reinforcement learning algorithms. Their learning environments are packaged under a tool called "gym", which exposes a standardized API to various domains and has become increasingly popular in the reinforcement learning community.

OpenAI's algorithm implementations depend on the first version of Google's popular machine learning framework Tensorflow (Abadi et al., 2015). Tensorflow 1 makes use of a declarative programming API, where computational operations are declared up-front and invoked implicitly later in the program. This creates a large barrier for those unfamiliar with the framework, as the execution logic does not flow in a functional manner, as in traditional programs. To remedy this, Google recently released Tensorflow 2, a functional version of Tensorflow. By utilising Tensorflow 2, the implementations presented in this paper aim to achieve higher clarity for those new to reinforcement learning while maintaining feature parity with OpenAI's implementations.

5.1 Algorithm Implementation

A3C, A2C and ADQN were all implemented using Tensorflow 2 (Abadi et al., 2015). Tensorflow 2 provides a high level API called Keras which allows neural network architectures to be constructed layer-by-layer in a readable manner. The core elements of the algorithm implementations were based on the implementations provided by OpenAI (Dhariwal et al., 2017) and a tutorial on reinforcement learning with Tensorflow 2 released on Tensorflow's blog (Yuan, 2018). A results-oriented approach was followed, whereby the implementations were tested on a simple "gym" environment to establish confidence in their functionality. The gym environment chosen was "CartPole-v1", which involves balancing a pole on a cart (see Figure 5.1). This environment was cho-

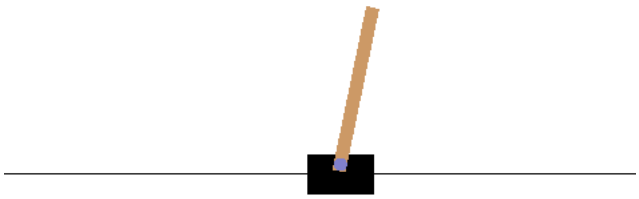


Figure 5.1: The 'CartPole-v1' environment. The agent tries to keep the pole above the horizontal line by applying force to either end of the cart.

sen due to its simplicity, which allowed for fast training and testing, and similarity to the target domain with respect to discrete action and state spaces.

5.2 Network Architecture

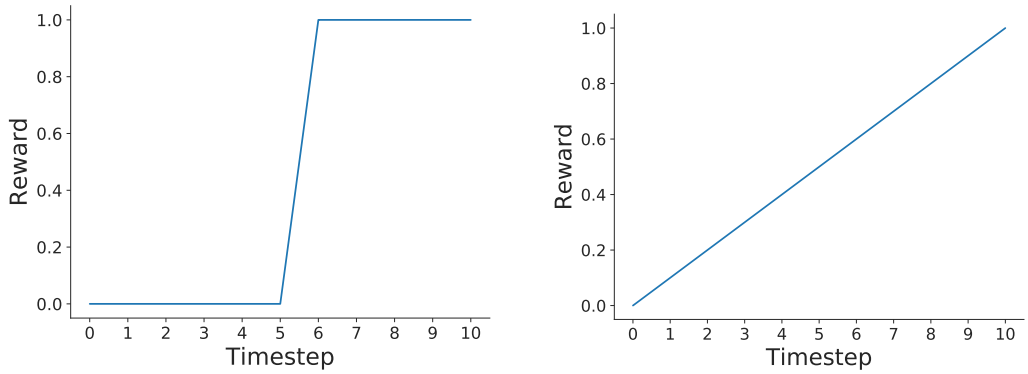
A multi-layer-perceptron (MLP) architecture was chosen for the neural networks due to the ease of implementation, and the low granularity of the input: the state spaces encode high-level features of the environment. For the actor-critic algorithms the actor and critic networks consist of two hidden layers each with 100 nodes. This architecture differs from many other reference implementations where the actor and critic share a number of hidden layers. This is particularly relevant for networks with convolutional layers, where the first few layers extract the same set of simple features such as basic shapes. For the q-learning architecture a single MLP with two hidden layers with 100 nodes is used. The parameters used for the algorithms are described in Table 5.1.

Parameter	Value
Learning Rate	0.0001
Discount Factor	0.9
Number of Workers	16
Gradient Normalisation Value	1.0
Network Update Frequency	50
Number Training Episodes	1000

Table 5.1: Learning Parameters

5.3 Reward Function

Designing a reward function for the UAV environment is a non-trivial task. A basic reward function could constitute assigning a reward of 1 for every connected user, where a user is connected if it has an SINR value above some threshold. However, trivial reward functions like this are not necessarily conducive to algorithm convergence, as the



(a) Step-wise Reward Function (b) Smooth Reward Function

Figure 5.2: Comparison of a step-wise and a smooth reward function. The step-wise reward increases when a user becomes connected while the smooth reward function increases based on the SINR of the user.

agents do not receive rewards consistently for moving towards the goal. For example, with this reward function, if a UAV moves closer to a user but fails to raise its SINR above the threshold, the reward would not increase, and thus the agent would not be aware that it took an appropriate action. The quality of a reward function to adequately reflect movement towards the end goal heavily relates to its shape.

Typically, a smooth reward function is preferred to a step-wise one, as the reward signal to the agent is clearer. Considering the simple scenario described above, with a single UAV moving towards one user, we can visualise a continuous and step-wise reward. Figure 5.2 shows the hypothetical shape of the step-wise reward function compared to a smoother reward function which reflects the SINR of the user directly.

With these considerations in mind, the simplest reward function with an appropriate shape was chosen. This manifested as an extension of the smooth reward function above, for an arbitrary number of users and base stations. The reward given to the reinforcement learning agent is equal to the mean SINR of all the users in the environment. It can be reasoned that this reward function allows for the optimal behaviour of the base stations (each base station following a different cluster of users) as clustering of base stations would be reduced by their interference, which would in turn reduce the SINR of nearby users. The reward function R is formulated as follows:

$$R = \frac{\sum_{u \in U} SINR(u)}{|U|} \tag{5.1}$$

Where U is the set of users and $SINR$ is a function which returns the SINR value for the given user.

5.4 Training

Training the reinforcement learning algorithms involved running 1000 episodes, each consisting of 200 time-steps, with 10 random seeds. The set of random seeds was fixed for the various algorithms to allow for appropriate performance comparisons. Across the training episodes 10 checkpoints of model states were saved to allow adequate testing of the learning behaviour. All algorithms were trained on an AMD Ryzen 5 3600 processor with 12 logical cores.

To measure training performance a moving average reward R was calculated at each episode:

$$R = 0.99 \times R + 0.01 \times \text{Episodic Reward} \tag{5.2}$$

This provides a measure of training convergence, as the average episodic reward will increase as the algorithm converges to solving the task.

A greedy heuristic approach to solving the problem presented by Li et al. (2019) is used as a baseline for comparing the performance of the other algorithms. At each step this baseline moves all UAVs in the direction which maximizes the mean SINR of all the users in the next time-step. It is considered a greedy baseline to solving the task as it only considers the best possible action for the next time-step.

5.5 Testing

To test the learned behaviour of the algorithms, the 10 checkpoint models for each random seed were applied to a unique UAV environment. This testing environment contains the same number of users as in training, but is constructed with a single user movement trace (across tests the users will move identically). This movement trace is necessary to ensure fair comparisons across the algorithms. For the ADQN algorithm, random actions were not taken, forcing the agent to take the greedy action (the action it deems best) in each time-step.

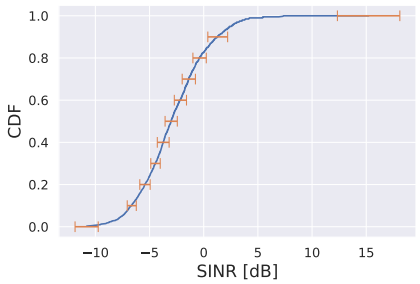
The key evaluation metrics for testing are the episodic reward and the cumulative distribution of the SINR across the episode. The cumulative distribution metric allows the median SINR experienced by the users to be identified while the reward gives a measure of the mean SINR.

Chapter 6

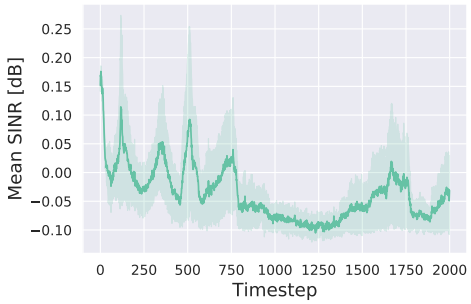
Performance Evaluation

6.1 Greedy Baseline

As the greedy baseline is a heuristic approach, it does not undergo a training process like the reinforcement learning algorithms. Testing the approach on the UAV environment produced baseline measures for the cumulative distribution function (CDF) and mean SINR of the users. The baseline was run 10 times on the testing environment, and the results were averaged across these 10 runs, to give a confidence interval to the results. These results are shown in Figure 6.1.



(a) Cumulative distribution function of SINR of all users with error bars representing one standard deviation. Median SINR is -3.00.



(b) Mean SINR of all users with shaded area representing one standard deviation.

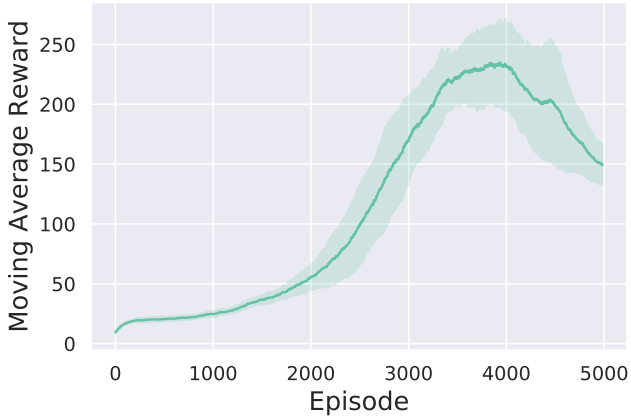
Figure 6.1: Baseline measures of SINR of users with greedy baseline.

6.2 A3C

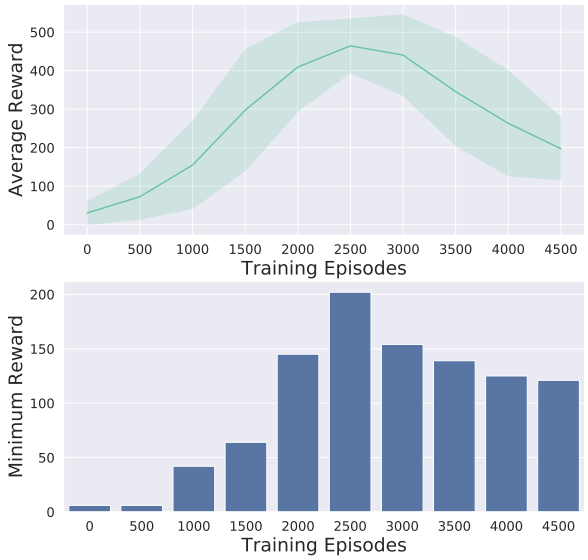
The asynchronous actor-critic algorithm (A3C) was implemented on the UAV domain by Li et al. (2019). The implementation of this algorithm was re-written in Tensorflow 2 to investigate any performance benefits in moving to the newer framework. Initially, the algorithm was tested for convergence on the CartPole task and, once this was achieved, applied to the UAV environment.

6.2.1 CartPole

The algorithm demonstrated convergence on the CartPole task, achieving a peak average score of 230 after 5000 training episodes. The CartPole task is defined as "solved" when an agent achieves an average reward of 195 over 100 consecutive trials (Brockman et al., 2016). Looking at the test performance curves we can see that this is achieved for all random seeds after 2500 training episodes. After 2500 training episodes the average testing reward reaches 454 and the minimum reward across testing episodes is 202. At peak performance 7.37% of testing episodes reach a score of 498, the maximum score for the CartPole setup, demonstrating that the algorithm does not fully master the task.



(a) Moving average reward of A3C during training, aggregated over 10 random seeds. Shaded area represents one standard deviation.



(b) Average reward aggregated over 10 random seeds and minimum reward during testing. Minimum reward showcases that the algorithm solves the task after 2500 training episodes.

Figure 6.2: Moving average reward and testing performance of A3C on cartpole task.

The algorithm showcases degraded performance after 2500 training episodes, dropping to an average reward of 200 per episode. This decrease in performance most probably stems from a key implementation detail of A3C: as the agents operate asynchronously, at any given time the parameters of the actor and critic networks across agents will differ. This manifests in a reduction in the information shared across the agents, and thus a potential decrease in the learning robustness. Learning robustness is especially important when the agent reaches a high level of performance as robustness prevents unstable updates to the policy and value networks which would result in sub-optimal behaviour (Nikishin et al., 2018). The exploration of the A2C algorithm will test this robustness hypothesis by removing the asynchronous agents from the equation.

6.2.2 UAV Environment

Applying A3C to the UAV environment, the training convergence of the older and newer implementations was investigated. As can be seen by Figure 6.3, the moving average training reward converges to approximately 39 and 36 for the old and new implementations respectively. While the newer version of A3C converges to a lower mean reward it remains within a single standard deviation of the older implementations mean reward. This provides weight to the claim that the average rewards between the two implementations remain similar, as would be expected for functionally identical algorithms.

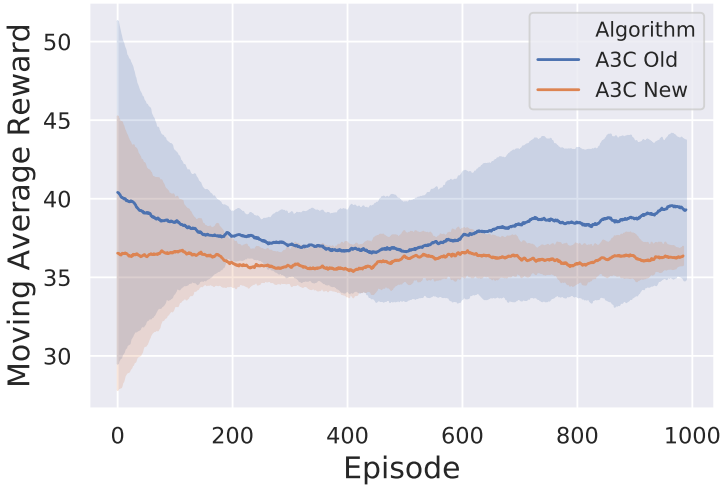
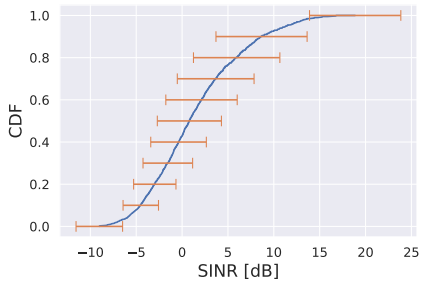
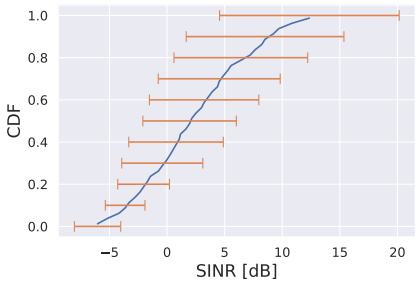


Figure 6.3: Moving average training reward for old A3C algorithm implemented in Li et al. (2019) and an updated A3C implementation in Tensorflow 2.

Investigating the results of the two algorithms on the test environment, the algorithms report similar performance metrics. In Figure 6.4, the cumulative distribution function of the SINR across the test episode shows that the updated A3C algorithm reports a 1.35 dB increase in median SINR. However, the significance of this increase is again diminished by the fact that both functions stay within one standard deviation of each other. The reward across testing displayed in Figure 6.5 shows comparable performance across the algorithms. Both converge to a mean reward of around 0 db across the 2000 timesteps.



(a) Cumulative Probability Distribution of SINR of all users trained with older A3C algorithm. Error bars represent one standard deviation. Median SINR is 0.80.



(b) Cumulative distribution function of SINR of all users trained with updated A3C algorithm. Error bars represent one standard deviation. Median SINR is 2.15.

Figure 6.4: Median SINR testing performance of A3C algorithms on UAV task.

(a) Mean SINR of users during testing, trained with older A3C algorithm. Shaded area represents one standard deviation.

(b) Mean SINR of users during testing, trained with updated A3C algorithm. Shaded area represents one standard deviation.

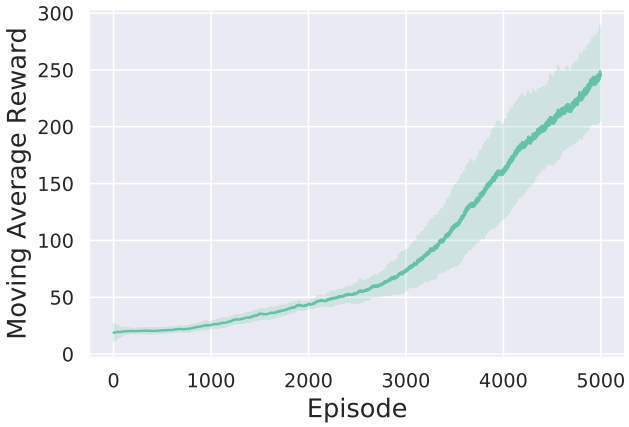
Figure 6.5: Mean SINR testing performance of A3C algorithms on UAV task.

6.3 A2C

Removing the asynchronous aspects of the A3C algorithm is referred to as advantage actor-critic (A2C). It has been shown empirically to perform equally or better than A3C and allows for large network updates to be processed on the GPU, further decreasing training times (Dhariwal et al., 2017). The A2C algorithm was trained on the CPU for both CartPole and UAV tasks due to their low space complexities and to provide a more fair comparison for the increased robustness hypothesised from the A3C results.

6.3.1 CartPole

As demonstrated in Figure 6.6, A2C converges to solving the task after 4500 episodes of training. At convergence the algorithm reaches an average testing reward of 480 with a minimum score of 252 across all testing episodes. A2C showcases mastery of the task, reaching the maximum score of 498 points across 84.3% of test episodes.



(a) Moving average reward of A2C during training, aggregated over 10 random seeds. Shaded area represents one standard deviation.

(b) Average reward aggregated over 10 random seeds and minimum reward during testing. Minimum reward showcases that the algorithm solves the task after 4500 training episodes.

Figure 6.6: Moving average reward and testing performance of A2C on cartpole task.

6.3.2 UAV Environment

Training on the UAV environment produced the moving average reward curve shown in Figure 6.7. The moving average mirrors the training curves of the A3C algorithms closely, even down to the standard deviation of the reward. Unlike in the CartPole environment, the A2C algorithm does not produce a more stable convergence curve than A3C which is likely due to the increased complexity of the environment. Plotting the CDF of the SINR experienced by all users, the median SINR reaches 4.40 and the mean SINR curves show stagnation at 0.10 dB (see Figure 6.8.

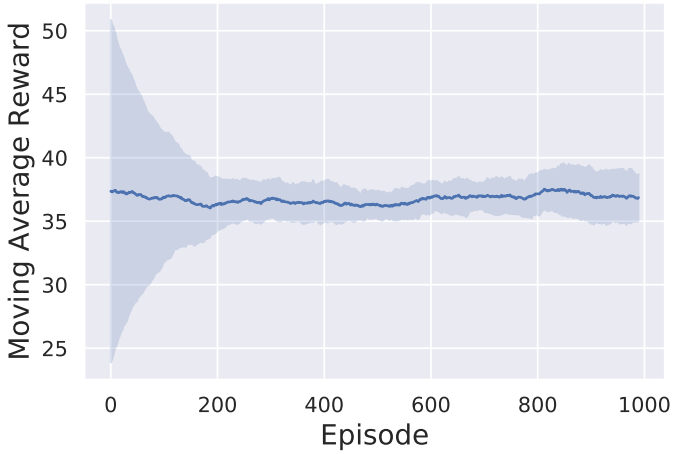
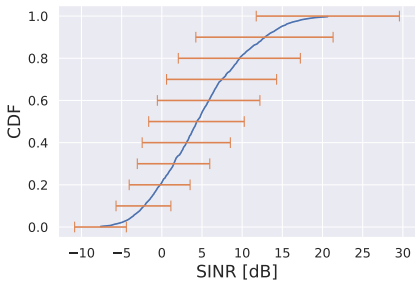


Figure 6.7: Moving average reward of A2C on the UAV environment.



(a) Cumulative distribution function of SINR of all users with error bars representing one standard deviation. Median SINR is 4.40.

(b) Mean SINR of all users with shaded area representing one standard deviation.

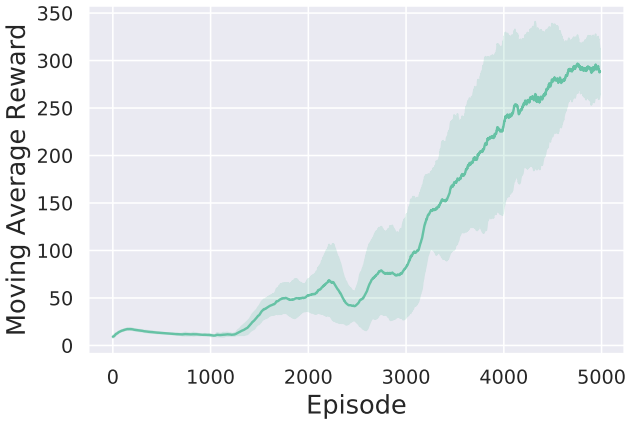
Figure 6.8: Evaluation metrics of SINR of users with the A2C algorithm.

6.4 ADQN

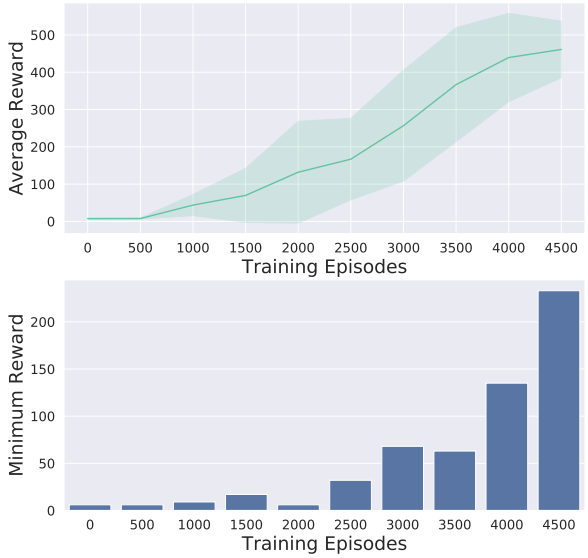
The asynchronous deep Q-network (ADQN) algorithm presents a fundamentally different approach to learning compared to the actor-critic methods described above. Utilizing a single network to represent the Q function, this algorithm allows for off-policy learning to be conducted. As the off-policy approach attempts to learn the optimal policy directly, it represents an important benchmark to compare the previous actor-critic algorithms to.

6.4.1 CartPole

Displayed in Figure 6.9, the ADQN algorithm solves the task after training for 4500 episodes. The mean average testing reward after convergence reaches 461 with a minimum score of 233. The best performing checkpoint of the model achieves the maximum score of 498 in 74.8% of episodes.



(a) Moving average reward of ADQN during training, aggregated over 10 random seeds. Shaded area represents one standard deviation.



(b) Average reward aggregated over 10 random seeds and minimum reward during testing. Minimum reward showcases that the algorithm solves the task after 4500 training episodes.

Figure 6.9: Moving average reward and testing performance of ADQN on cartpole task.

The inherent instability of the ADQN training is demonstrated with the moving average training reward. Unlike the other algorithms explored, ADQN does not iterate on a policy, instead taking the action it deems best at any given time-step or taking a random action to explore. This greedy behaviour means that small changes to the values assigned to each state can lead to large changes in the agent's actions, resulting in a non-monotonic training reward curve.

6.4.2 UAV Environment

As seen in Figure 6.10 the ADQN algorithm reports a large drop in the moving average reward across 1000 episodes of training to around 25. This behaviour indicates a degra-

dation in performance to a local minimum, from which the algorithm cannot explore enough to improve. This would typically be accompanied by low testing performance. However, as shown in Figure 6.11, the ADQN algorithm reports a median SINR value of 6.63 dB and converges to a testing reward of 0.15 dB. This testing reward is the highest achieved reward across the explored algorithms.

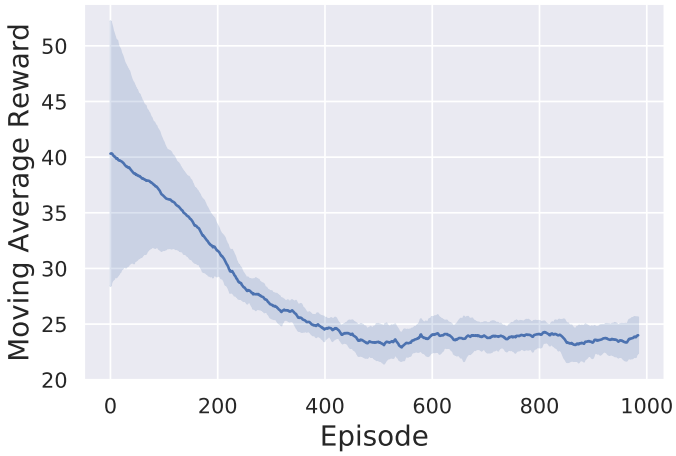
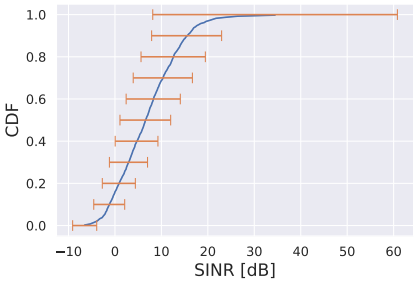
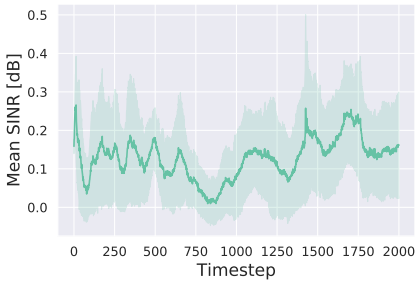


Figure 6.10: Moving average reward of ADQN on the UAV environment.



(a) Cumulative Probability Distribution of SINR of all users with error bars representing one standard deviation. Median SINR is 6.63.



(b) Mean SINR of all users with shaded area representing one standard deviation.

Figure 6.11: Evaluation metrics of SINR of users with the ADQN algorithm.

6.5 Evaluation

A key takeaway from the results of the explored algorithms is that the training reward does not provide a useful indication of the performance on the test environment. The ADQN algorithm converges to a mean reward of 25 compared to the 35+ of the other algorithms, while still achieving a higher median and mean SINR on the test environment. The discrepancy in training and testing environment performance can be ascribed to the change in behaviour of the ADQN algorithm. While training, the ADQN algorithm takes random actions with a certain probability ε enabling it to explore the environment. Once the agent begins to converge to useful behaviour, these random actions can severely disrupt its policy leading to reduced average rewards. During testing, the ADQN algorithm uses the greedy policy: always taking the action it deems best from each state. As there are no random actions taken, the testing reward gives a better indication of the performance of the learned policy.

The testing performance difference between the A2C and A3C algorithms is also not borne out by the training reward. The training reward of A2C sits between the two versions of the A3C algorithm and has a similar convergence shape. However, the median testing SINR of A2C exceeds that of the best performing A3C algorithm by 2.25 dB. One difference between the algorithms that could lead to divergent behaviour is the ability for workers in A3C to overwrite each other's behaviour if updates are not sparse (Recht et al., 2011). However, if these disruptive updates were occurring we would expect to see an impact on the training reward which is not demonstrated by the data. With this in mind, it is likely that the gain in A2C testing performance is a result of the workers in the algorithm always having access to the shared experience of all other workers. While in the A3C algorithm, workers could be updating the network with gradients produced by stale weights, the network weights in A2C are fresh throughout training.

Comparing the CDF of all algorithms in Figure 6.12 there are a number of key takeaways. Firstly, all reinforcement learning algorithms significantly outperform the greedy baseline. This was an expected result due to the limited look-ahead and robustness of the baseline. As expected, the CDF curves for the A3C algorithms are closely related. Finally, there are significant median SINR increases for both the A2C and the ADQN algorithm. They achieve an increased median SINR of 7.40 and 9.63 respectively compared to the greedy baseline. If user outages were incurred at an SINR of 0 dB then the greedy baseline would incur outages from 80% of users while ADQN would incur outages from only 18% of users.

By inspecting the training times across the algorithms, the run-time complexity of training can be visualised. The training times are displayed in Figure 6.13. As shown by the times reported for the A3C implementations, upgrading to Tensorflow 2 reduced training times by an average of 5 minutes (8.85%). The ADQN algorithm is the slowest to train of the four algorithms by more than 10 minutes. This is due to the increased rate of network updates during the algorithm's execution. As stated in Chapter 3, to ensure stability in training, the target Q network must be updated with the main networks weights regularly. This occurs more frequently than the network updates in

A3C, causing the ADQN algorithm to spend more time copying network parameters.

Comparing the A3C and ADQN results to A2C, it is clear that the synchronous algorithm requires less time to train. This can be attributed to the reduced number of gradient calculations required by the A2C algorithm: A2C can gather a batch of experiences and calculate a single gradient update from this batch, while the asynchronous algorithms have to calculate separate gradients for each asynchronous worker.

Finally, the model complexity of the various algorithms is detailed in Table 6.1. As shown, the ADQN algorithm has the highest model complexity when considering the static model. However, the model complexity during training demonstrates the increased complexity incurred by the asynchronous algorithms. This is due to the local copies of the neural network required by each worker in A3C and ADQN. The model complexity is loosely coupled to the training time complexity as an increased number of parameters often translates to slower neural network updates.

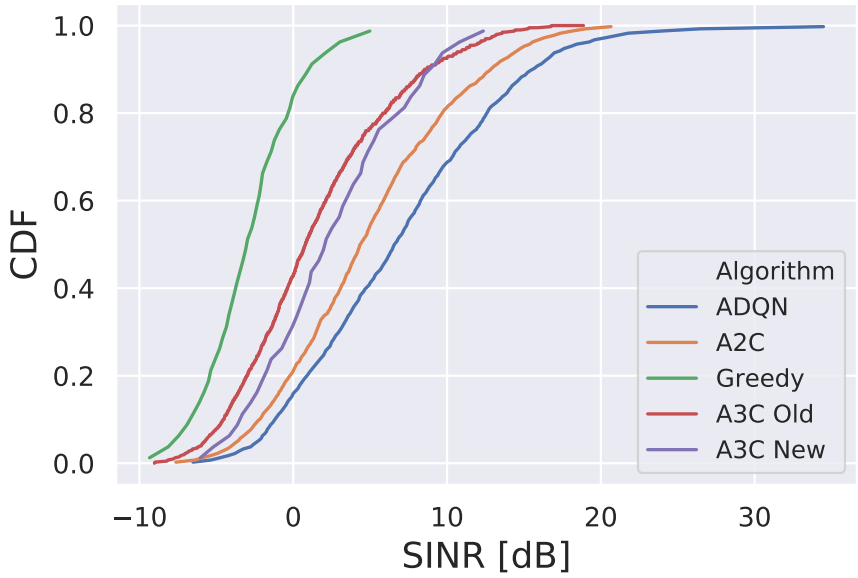


Figure 6.12: Comparison of users' SINR during testing across all algorithms.

Algorithm	Model Parameters (million)	Parameters During Training (million)
A3C	20.2	323.3
A2C	20.2	20.2
ADQN	25.6	409

Table 6.1: Algorithms and the number of learned parameters in their models. "Parameters During Training" refers to the number of parameters in memory during training.

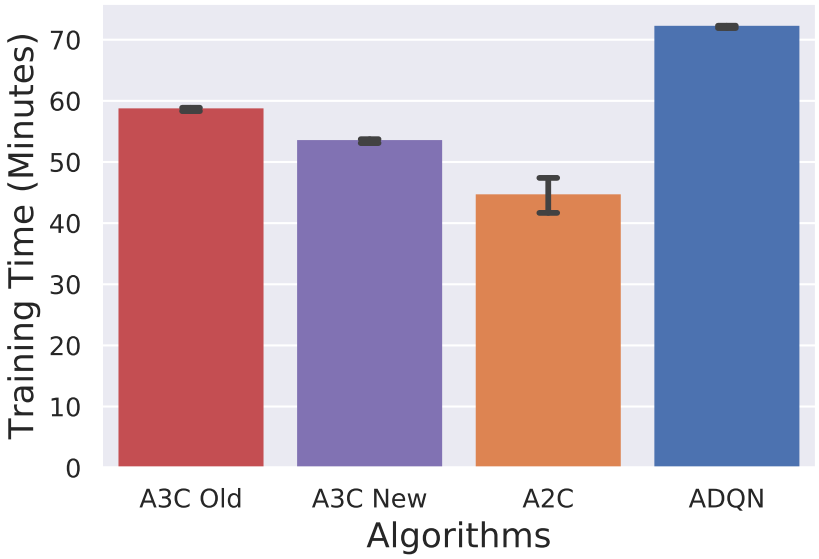


Figure 6.13: Comparison of training times for each of the algorithms.

Chapter 7

Conclusion

This paper explored the application of a number of deep reinforcement learning algorithms to a UAV control task within an environment where UAV base stations provide network coverage to users. By implementing an optimized actor critic algorithm (A2C) and exploring a popular off-policy algorithm (ADQN) this paper utilized a number of reinforcement learning methods to solve the task. The ADQN algorithm achieves the highest median SINR value during testing, surpassing a greedy baseline by over 9 dB and consequently achieving a lower user outage rate of 18% compared to 80% for the baseline with an appropriate threshold.

The reinforcement learning algorithm A2C demonstrates superior testing performance compared to the A3C algorithm. With an increased median SINR of 3.2 dB, the A2C algorithm showcases the marginal benefits of synchronous updates on performance. However, while both actor critic methods outperform the greedy baseline they perform significantly worse than the ADQN algorithm. The success of the Q-learning based approach suggests that explicit exploration of the environment can lead to better performance on the task. Additionally, the superiority of the off-policy approach suggests this family of algorithms may perform more robustly on non-Markovian environments.

Finally, the algorithms explored in this paper were all open-sourced and bench-marked against the popular "gym" environment (Brockman et al., 2016). These reference implementations will hopefully facilitate further research in the area and allow the operational details of the algorithms to be more clearly scrutinized.

7.1 Future work

There are many avenues for future exploration on the UAV environment. To provide a more realistic simulation, customized user mobility models based on data of user movement in different scenarios could be explored. Additionally, the channel model which assumes a free-space path loss could be extended to cover outdoor to indoor penetration loss as would be common for urban environments (Haneda et al., 2016). Finally, the arena size of the environment could be increased to better represent the

scale of environments where UAV base stations would be employed.

Further work to improve the performance of the explored algorithms could include optimizing the neural network architecture. As the state space represents a rough grid map of the environment, a convolutional neural network (CNN) architecture may perform better than the multi-layer perceptrons employed in this paper. CNNs have been shown to outperform MLPs on image recognition tasks due to their success at representing spatial dependencies in the input (LeCun et al., 1999).

To decrease algorithm training times further, the algorithm implementations would benefit from utilization of multi-core processing. One of the key trade-offs of Python's threading library is that while it provides a clear API for parallel processing, its threads are confined to a single core. Due to the rise in computational cores in commodity CPUs, utilizing multi-core performance would present a large increase in maximum training speed. This could be achieved with Python's multiprocessing library, or by translating the code to another language which offers multi-core parallel processing such as Go (Donovan and Kernighan, 2015), with both solutions requiring careful management of shared memory.

To exploit the multi-agent aspects of the UAV environment, an approach similar to joint action learners (JAL) could be taken. JAL is a multi-agent learning approach where each agent models the behaviour of other agents in the environment (Claus and Boutilier, 1998). By modelling the movement of the users, the learning agent could approximate the motion model being used, thus increasing the Markovian properties of the learning process.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Berk Anbaroglu. Parcel delivery in an urban environment using unmanned aerial systems: A vision paper. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-4/W4:73–79, 11 2017. doi: 10.5194/isprs-annals-IV-4-W4-73-2017.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, page 746–752, USA, 1998. American Association for Artificial Intelligence. ISBN 0262510987.

Robert H. Crites and Andrew G. Barto. Improving elevator performance using reinforcement learning. In *Proceedings of the 8th International Conference on Neural Information Processing Systems*, NIPS'95, page 1017–1023, Cambridge, MA, USA, 1995. MIT Press.

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

Alan A.A. Donovan and Brian W. Kernighan. *The Go Programming Language*. Addison-Wesley Professional, 1st edition, 2015. ISBN 0134190440.

Shixiang Gu, Timothy P. Lillicrap, Zoubin Ghahramani, Richard E. Turner, and Sergey

Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. *CoRR*, abs/1611.02247, 2016. URL http://arxiv.org/abs/1611.02247.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL http://arxiv.org/abs/1801.01290.

Katsuyuki Haneda, Lei Tian, Yi Zheng, Henrik Asplund, Jian Li, Yi Wang, David Steer, Clara Li, Tommaso Balercia, Sunguk Lee, YoungSuk Kim, Amitava Ghosh, Timothy Thomas, Takehiro Nakamura, Yuichi Kakishima, Tetsuro Imai, Haralabos Papadopoulas, T.S. Rappaport, George Maccartney, and Arun Ghosh. 5g 3gpp-like channel models for outdoor urban microcellular and macrocellular environments. 02 2016.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015. URL http://arxiv.org/abs/1509.06461.

Matthew Hausknecht and Peter Stone. On-policy vs. off-policy updates for deep reinforcement learning. In *Deep Reinforcement Learning: Frontiers and Challenges, IJCAI Workshop*, July 2016.

Xiaoyan Hong, Mario Gerla, Guangyu Pei, and Ching-Chuan Chiang. A group mobility model for ad hoc wireless networks. In *Proceedings of the 2nd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '99, page 53–60, New York, NY, USA, 1999. Association for Computing Machinery. ISBN 1581131739. doi: 10.1145/313237.313248. URL https://doi.org/10.1145/313237.313248.

Hongji Huang, Yuchun Yang, Hong Wang, Zhiguo Ding, Hikmet Sari, and Fumiyuki Adachi. Deep reinforcement learning for uav navigation through massive mimo technique, 2019.

William Koch, Renato Mancuso, Richard West, and Azer Bestavros. Reinforcement learning for UAV attitude control. *CoRR*, abs/1804.04154, 2018. URL http://arxiv.org/abs/1804.04154.

Guillaume Laurent, Laëtitia Matignon, and Nadine Fort-Piat. The world of independent learners is not markovian. *KES Journal*, 15:55–64, 03 2011. doi: 10.3233/KES-2010-0206.

Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, page 319, Berlin, Heidelberg, 1999. Springer-Verlag. ISBN 3540667229.

Rui Li, Chaoyun Zhang, Paul Patras, Razvan Stanica, and Fabrice Valois. Learning driven mobility control of airborne base stations in emergency networks. *SIGMETRICS Perform. Eval. Rev.*, 46(3):163–166, January 2019. ISSN 0163-5999. doi: 10.1145/3308897.3308964. URL https://doi.org/10.1145/3308897.3308964.

Siyi Li, Tianbo Liu, Chi Zhang, Dit-Yan Yeung, and Shaojie Shen. Learning unmanned

aerial vehicle control for autonomous target following. *CoRR*, abs/1709.08233, 2017. URL http://arxiv.org/abs/1709.08233.

X. Li and L. Yang. Design and implementation of uav intelligent aerial photography system. volume 2, pages 200–203, 2012.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1509.02971.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. URL http://arxiv.org/abs/1602.01783.

Evgenii Nikishin, Pavel Izmailov, Ben Athiwaratkun, Dmitrii Podoprikhin, Timur Garipov, Pavel Shvechikov, Dmitry P. Vetrov, and Andrew Gordon Wilson. Improving stability in deep reinforcement learning with weight averaging. 2018.

Huy Pham, Hung La, David Feil-Seifer, and Luan Nguyen. Autonomous uav navigation using reinforcement learning. 01 2018.

Benjamin Recht, Christopher Ré, Stephen J. Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015. URL http://arxiv.org/abs/1502.05477.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Sandip Sen, Mahendra Sekaran, and John Hale. Learning to coordinate without sharing information. In *AAAI*, 1994.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017. ISSN 1476-4687. doi: 10.1038/nature24270. URL https://doi.org/10.1038/nature24270.

Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Pro-*

ceedings of the 12th International Conference on Neural Information Processing Systems, NIPS'99, page 1057–1063, Cambridge, MA, USA, 1999. MIT Press.

G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.

John P. Tiefenbacher, editor. *Natural Hazards - Risk, Exposure, Response, and Resilience*. Number 5751 in Books. IntechOpen, 2019. ISBN ARRAY(0x440c1e78). doi: 10.5772/intechopen.77841. URL https://ideas.repec.org/b/ito/pbooks/5751.html.

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Rémi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *CoRR*, abs/1611.01224, 2016. URL http://arxiv.org/abs/1611.01224.

Christopher J. C. H. Watkins and Peter Dayan. Technical note: q -learning. *Mach. Learn.*, 8(3–4):279–292, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992698. URL https://doi.org/10.1007/BF00992698.

Ramond Yuan. Deep reinforcement learning with keras eager execution, 2018. URL https://blog.tensorflow.org/2018/07/deep-reinforcement-learning-keras-eager-execution.html. Accessed: 2019-12-20.

Appendix A

Learning Algorithms

Algorithm 1 Asynchronous advantage actor-critic implementation for worker thread

Require: *Global shared actor network with weights* θ^a
Require: *Global shared critic network with weights* θ^c
Require: *Global shared episode counter* T
Require: *Max number of episodes* T_{max}
Require: *Network update frequency* T_{update}
Require: *Local actor network with weights* θ_l^a
Require: *Local critic network with weights* θ_l^c
Require: *Local step counter* $t \leftarrow 1$
  while $T \leq T_{max}$ **do**
    Reset gradients: $d\theta^a \leftarrow 0$ and $d\theta^c \leftarrow 0$
    Update local actor and critic networks $\theta_l^a \leftarrow \theta^a$ and $\theta_l^c \leftarrow \theta^c$
    $t_{start} \leftarrow t$
    Perceive current state s_t
    while s_t is not terminal and $t - t_{start} \neq T_{update}$ **do**
      Take action a_t according to policy $\pi(a_t|s_t;\theta_l^a)$
      Receive reward r_t and new state s_{t+1}
      $t \leftarrow t+1$
      $T \leftarrow T+1$
    end while
    $R \leftarrow 0$
    if s_{t+1} is non-terminal **then**
      $R \leftarrow V(s_t, \theta_l^c)$
    end if
    for $i \in \{t-1, ..., t_{start}\}$ **do**
      $R \leftarrow r_i + \gamma R$
      Accumulate actor gradients: $d\theta^a \leftarrow d\theta^a + \nabla_{\theta_l^a} \log \pi(a_i|s_i;\theta_l^a)(R - V(s_i;\theta_l^c)$
      Accumulate critic gradients: $d\theta^c \leftarrow d\theta^c + \frac{\partial (R - V(s_i;\theta_l^c))^2}{\partial \theta_l^c}$
    end for
    Asynchronous update of network θ^a using $d\theta^a$ and θ^c using $d\theta^c$
  end while

Algorithm 2 Advantage actor-critic implementation for coordinator thread

Require: *Actor network with weights* θ^a
Require: *Critic network with weights* θ^c
Require: *Episode counter* T
Require: *Max number of episodes* T_{max}
Require: *Network update frequency* T_{update}
Require: *Number of Parallel Environments* P
  while $T \leq T_{max}$ **do**
    Reset gradients: $d\theta^a \leftarrow 0$ and $d\theta^c \leftarrow 0$
    for $p \in P$ **do**
      $t \leftarrow p.t$
      $t_{start} \leftarrow p.t_{start}$
      Perceive current state s_t
      while s_t is not terminal and $t - t_{start} \neq T_{update}$ **do**
        Take action a_t according to policy $\pi(a_t|s_t;\theta_l^a)$
        Receive reward r_t and new state s_{t+1}
        $t \leftarrow t+1$
        $T \leftarrow T+1$
      end while
      if s_{t+1} is non-terminal **then**
        $R \leftarrow V(s_t,\theta_l^c)$
      end if
      if s_{t+1} is terminal **then**
        $R \leftarrow 0$
        p.reset-environment()
      end if
      for $i \in \{t-1,...,t_{start}\}$ **do**
        $R \leftarrow r_i + \gamma R$
        Accumulate actor grads: $d\theta^a \leftarrow d\theta^a + \nabla_{\theta^a} \log \pi(a_i|s_i;\theta^a)(R - V(s_i;\theta^c)$
        Accumulate critic grads: $d\theta^c \leftarrow d\theta^c + \frac{\partial (R-V(s_i;\theta^c))^2}{\partial \theta^c}$
      end for
      Update of network θ^a using $d\theta^a$ and θ^c using $d\theta^c$
    end for
  end while

Algorithm 3 Asynchronous Q-learning implementation for worker thread

Require: *Global shared main network with weights* θ
Require: *Global shared target network with weights* θ^t
Require: *Global shared episode counter* T
Require: *Max number of episodes* T_{max}
Require: *Target network update frequency* T_{target}
Require: *Main network update frequency* T_{update}
  Initialize network gradients $d\theta \leftarrow 0$
  Perceive current state s_t
  while $T \leq T_{max}$ **do**
    Take action a_t with ε-greedy policy using $Q(s_t, a_t; \theta)$
    Receive new state s_{t+1} and reward r_{t+1}
    $y \leftarrow r_{t+1}$
    if s_{t+1} is non-terminal **then**

$$y \leftarrow y + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta^t) \tag{A.1}$$

    end if
    Accumulate gradients: $d\theta \leftarrow d\theta + \frac{\partial (y - Q(s_t, a_t; \theta))^2}{\partial \theta}$
    if T mod $T_{target} == 0$ **then**
      Update target network $\theta^t \leftarrow \theta$
    end if
    if T mod $T_{update} == 0$ **then**
      Asynchronous update of network θ using gradients $d\theta$
      Zero gradients $d\theta \leftarrow 0$
    end if
    $T \leftarrow T + 1$
  end while