Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

An Investigation into Deep Reinforcement Learning

Jack Cassidy
14320816

March 5, 2018

A Final Year Project submitted in partial fulfilment
of the requirements for the degree of BAI (Computer Engineering)

Declaration

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: ______________________ Date: ______________________

Abstract

A short summary of the problem investigated, the approach taken and the key findings. This should be around 400 words, or less.

This should be on a separate page.

Acknowledgements

Thanks Mum!

You should acknowledge any help that you have received (for example from technical staff), or input provided by, for example, a company.

Contents

List of Figures

List of Tables

Nomenclature

A	Area of the wing	m^2
B		
C	Roman letters first, with capitals...	
a	then lower case.	
b		
c		
Γ	Followed by Greek capitals...	
α	then lower case greek symbols.	
β		
ϵ		
TLA	Finally, three letter acronyms and other abbreviations arranged alphabetically	

If a parameter has a typical unit that is used throughout your report, then it should be included here on the right hand side.

If you have a very mathematical report, then you may wish to divide the nomenclature list into functions and variables, and then sub- and super-scripts.

Note that Roman mathematical symbols are typically in a serif font in italics.

1 Introduction

1.1 Motivation

Machine Learning (ML) and Artificial Intelligence (AI) in 2018 are subjects that are almost unique in their ability to permeate into nearly every sphere, community and space in today's society. From the research community to the business world and the public eye through extensive media coverage, ML is certainly becoming more and more of a de facto part of our everyday lives. Businesses employ recommender systems to suggest new products to their customers and predict the rise and fall of stock prices using function approximators like Deep Learning. Traditional home appliances are now outdated in favour of smarter, IoT systems that learn our habits and provide a more tailored experience.

ML is a broad umbrella term, encapsulating a variety of different approaches. Most ML tasks can be classified as either supervised or unsupervised learning. Deep Learning is fast becoming a popular and powerful technique in supervised learning, involving teaching artificial neural networks to approximate any function, given enough training data. Reinforcement Learning (RL), another subset of supervised learning, is a branch of ML that perhaps receives less public attention but is nonetheless believed to be set to revolutionize the field of AI (1). Recent breakthroughs in the application of Deep Learning to RL algorithms has spawned the exciting research field of Deep Reinforcement Learning (DRL) which has produced to date unparalleled results in various AI domains, such as defeating the world champion Go player (2).

There are a growing number of RL methods and algorithms, such as Monte-Carlo, Q-Learning, SARSA and Policy Search (3). More recently, the advent of DRL has brought about adaptations to existing algorithms to expand their use to multi-dimensional observations spaces such as pixel information, a notable example being Deep Q-Learning (4). It is easy to become overwhelmed with all of these offerings when exploring the RL space. The motivation behind this project is to demystify the state of the art of RL.

1.2 Objectives

The objectives of this project are threefold.

1. Research the development and state of the art of RL.
2. Build a system to evaluate the performance of three state of the art DRL algorithms by collecting a series of metrics while applying each algorithm to a selection of Atari 2600 video games.
3. Carry out the experiments, obtaining values for game score, survival time and model loss. Compare and contrast the different algorithms using the metrics recorded.

The system is given no prior knowledge of how each game works and there is no change in the underlying architecture of the solution when applied to different games, all while maintaining a high level of performance. The aim for the system is to be a general solution, that it can be expanded to work for any number of games and algorithms in the future with ease of implementation. The algorithms used are Deep Q-Learning, Double Q-Learning and Dueling Q-Learning.

1.3 Research Methods

This project takes a *case study* based approach to the experimentation. The first phase of the project involves building the system to the specification outlined previously. The second phase treats each game entered into the system as an individual case study. The game ROM is given as input to the system. The game is simulated by a third party emulator of our choosing (discussed in chapter 3), from which the system extracts greyscale frames to learn from. The output of the system is an action that it has chosen to be optimal, selected from the discrete vector of possible actions as defined by the game's control scheme. This control scheme is not provided to the system, it determines it dynamically with each game. The action is fed back into the emulator and the cycle continues up to a terminating signal.

1.4 Report Overview

Chapter 2 gives some necessary background information. It will discuss the current state of the art of RL with particular interest in how it is being applied to video games, as well as the technologies and tools being used in research today and for this project.

Chapter 3 outlines the architecture of the system and the rationale behind certain design decisions.

Chapter 4 will discuss the components of the experiment evaluation. It will give a greater elaboration of the project's objectives, a description of the experimental setup, and a discussion of the results.

Chapter 5 closes the project with a conclusion of all that has been discussed, an outline of what has been achieved from both an objective and personal point of view and finally a suggestion for future work.

2 Background

This chapter will give some background information to the key concepts that need to be understood for this project. It will begin with a short introduction of ML and RL, how Deep Learning has revolutionized the field, and a discussion on the current state of the art of DRL. The Q-Learning algorithm will be explained in detail, as it is a prerequisite to understand the DRL algorithms Deep Q-Learning, Double Q-Learning and Duel Q-Learning used in the implementation of this project. Finally we will provide a discussion of the various tools and technologies that were considered for use in the building of the system.

2.1 Machine Learning

2.1.1 Introduction

ML is a broad umbrella term for various methods of giving computer systems the ability to 'learn' to complete some task efficiently using training and validation data, without being explicitly programmed to do so. Instead of following a programmed set of instructions to make a prediction, the ML system constructs a model that is a function approximated to some real world problem. ML tasks can be divided into two categories; supervised and unsupervised learning. In supervised learning, the dataset provides the correct output prediction, 'labelled' data, the system should make. The system can use the input/output pairs to iteratively learn the best prediction, using a combination of some generic error function, such as the *Mean Squared Error*, and the *Back Propagation* algorithm (5) to update the model. In unsupervised learning, the dataset does not contain any output data points, 'unlabelled' data, hence it is more difficult to gauge the performance of an unsupervised ML algorithm. Unsupervised learning is generally used in the clustering of data into classes.

2.1.2 Development of Machine Learning in Video Games

In order to claim that an AI agent achieves general competency, it should be tested in a set of environments that provide a suitable amount of variation, are reflective of real world problems the agent might encounter, and that were created by an independent party to remove experimenter's bias (6). In this way, video games provide an effective test-bed for efficiently studying general AI agents as they can provide all of these requirements. Although the application of ML generated AI to video games may seem novel, the end goal is not to produce agents for defeating world champion chess players, but to take these general agents and extend them to more pressing problems to humanity, of which there are endless possibilities.

The first application of notoriety to use computing to play a game arose in the research paper "Programming a Computer for Playing Chess, 1950" (7), where mathematician Claude Shannon developed an autonomous chess-playing system. In that paper, author Shannon also highlighted the point that although the application of such a solution may seem unimportant;

> "It is hoped that a satisfactory solution of this problem will act as a wedge in attacking other problems of a similar nature and of greater significance"

Claude designed a strategy that, even at the time, was infeasible as it would take more than 16 minutes to make a move.

Fast forward to 1997, IBM developed "Deep Blue," a network of computers purpose built to play chess at an above-human level. It is renowned as the first AI system to defeat a world champion chess player, Garry Kasparov under normal game regulations. There is an air of controversy surrounding the feat, as IBM denied any chance of a replay after Kasparov claimed that IBM cheated by actively programming moves into Deep Blue as the game was in play.

In more recent times, British AI research company DeepMind published the paper "Playing Atari with Deep Reinforcement Learning, 2013" (4) in which they achieved far and above human-level performance on a selection of 52 Atari2600 games with their Deep Q-Learning algorithm. DeepMind have since applied their techniques to modern, real time, strategy game StarCraft2 (8). This application was most impressive, as StarCraft2 is a game of imperfect information. This means that not all aspects of the game are given to the player. In the context of StarCraft2, the game map is not completely observable, the two players of the game cannot see what the other is doing. All other exploits into the application of ML in video games up to then had been on games of perfect information.

2.2 Reinforcement Learning

2.2.1 Introduction

RL is another case of ML tasks, which can come under the supervised and unsupervised learning categories. It is a general way of approaching optimisation problems by trial and error. An agent carries out actions in an environment, moving from one state to a new state and is given some positive or negative numerical reward. This is known as the *perception-action-learning loop*.

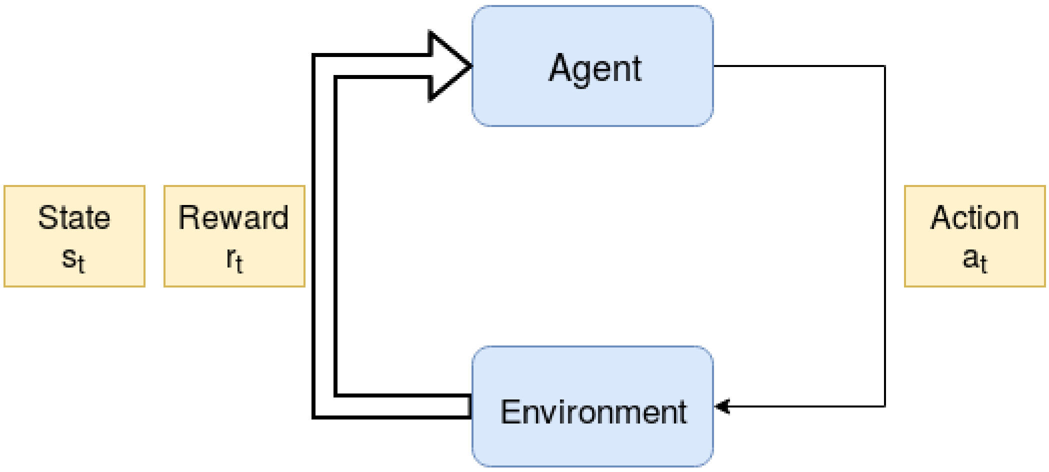

Figure 2.1: The perception-action-learning loop

RL is an interesting method of accomplishing ML tasks, as the agent can be given no prior information about it's environment or the task, and it can learn based solely on trial and error, reward and punishment. There are 3 main parts to a RL problem setup.

1. An agent follows a *policy* π, a rule that maps a state to an action.
2. A *reward function* $R(s, a)$, that gives an immediate value to an action a taken by the agent to transition from state s to s'
3. A *state value function* $V(s)$ that measures 'how good it is' to be in a given state. It assigns a value to the cumulate reward an agent can expect to gain by being in a state and following a policy through all subsequent states. We can define this as the *discounted cumulative reward*:

$$V(s) = E(\sum_{t=0} \gamma^t R(s_t, a_t) | s_0 = s) \qquad \forall s \in S$$

Where γ is a discount factor $[0, 1]$ and we choose $a_t = \pi(s_t)$. The objective of a RL problem is learning the optimal policy π^*, that for any given state will point the agent to the most favourable action so as to maximize it's cumulative reward, $V^*(s) = \max_\pi V(s)$. RL algorithms such as Q-Learning are used to find this optimal policy.

2.2.2 Markov Decision Process

In a more formal setting, it is a soft assumption that RL problems qualify as a Markov Decision Process (MDP) and can be modelled as such (1). MDP's display the Markov Property; that the conditional probability distribution of future states is dependant only on the current state and totally independent of all past states. A MDP consists of:

- A finite set of states S
- A finite set of actions A
- A transition function $P(s, a, s') = P(s_{t+1}|s_t, a_t)$, a model mapping current state, action pairs to a probability distribution of potential future states.
- An immediate reward function $R(s, a)$

Again, an MDP seeks to find the optimal policy π^*. We define π^* as

$$\pi^* = argmax_a\{\sum_{s'} P(s, a, s')(R(s, a) + \gamma V(s')\}$$

2.2.3 Model-Free Learning

Not all RL problems are provided with a transition function $P(s, a, s')$. In fact, it is more often than not that we cannot express the agent's environment with a model. Such a scenario is called *model-free learning*, where the agent must learn the optimal policy without the use of a transition function to guide it on which action to take. Instead it must devise some other way of modelling it's environment, such as building a 'memory' of actions and rewards based on experiences and deriving an optimal policy from these experiences. The downside to this is the potentially large amounts of auxiliary space needed to store the experiences. This is where algorithms such as Q-Learning are used.

Q-Learning is a model free RL algorithm. At each state, the agent calculates an immediate reward, based solely on the current state and action taken, and the *quality* of taking an action a in state s and following a policy π thereafter, called the Q-Value, which is defined as:

$$Q(s, a) = E(\sum_{t=0} \gamma^t R(s_t, a_t) | s_0 = s, a_0 = a) \qquad \forall s \in S$$

Where we have chosen a_0 arbitrarily and choose all subsequent $a_t = \pi(s_t)$ thereafter. As the agent explores all states multiple times and experiments with different actions, the corresponding Q-values are saved and updated in a data structure, hence an optimum policy can be derived by finding the optimum Q-values $Q^*(s, a)$ for all states after a predetermined number of iterations or until the policy is 'good enough'. The update step for a Q-Value is defined as:

$$Q(s_t, a_t) \leftarrow (1 - \alpha) Q(s_t, a_t) + \alpha(r_t + \gamma max_a Q(s_{t+1}, a))$$

Where α is a hyperparameter called the *learning rate* chosen in the range $(0, 1)$. As $t \to \infty$, $Q_t \to Q_t^*$, we converge on an optimum solution. The data structure used to store the agent's experiences can be referred to as the Q-Matrix. It is a $|S| x |A|$ sized matrix, that is the total number of states x total number of available actions.

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Figure 2.2: An example Q-Matrix. 0 indicates an unexplored state, action pair

The agent follows the algorithm 1, detailed below. After a suitable number of iterations and exploration, the Q-Matrix becomes a 'map' for the agent, whereby it can look up the action with the highest Q-Value for any state (9). Q-Learning is a straight-forward, elegant solution to a RL task. However, for an environment space of high dimensionality, such as an array of RGB pixels from an image, the Q-Matrix becomes infeasibly large in the S

dimension and increasingly sparse, as only a small percentage of the total available state, action pairs will be visited. As a worked example, imagine a robot that is using Q-Learning to find a path from it's current position to some exit room. If the robot takes 210x160, 8-bit colour space, RGB photos of it's surroundings to represent a state, the S dimension becomes $256^{210x160x3}$ in size. A solution to the dimensionality problem was proposed by DeepMind, in the paper "Playing Atari with Deep Reinforcement Learning, 2013" (4) which will be discussed later in this chapter.

Algorithm 1 Q-Learning Algorithm

procedure BUILDING Q-MATRIX
    Set α and γ parameters.
    Initialize Q-Matrix to zero.
    repeat
        while Goal/terminal state not reached **do**
            Select a randomly from all possible actions in current state
            Consider going to state s_{t+1} from state s using action a
            Get maximum Q-Value from s_{t+1} considering all possible actions
            $Q(s,a) \leftarrow (1-\alpha)Q(s_t,a_t) + \alpha(r_t + \gamma max_a Q(s_{t+1},a))$
        end while
    until Policy good enough
end procedure
procedure USING Q-MATRIX
    $s \leftarrow$ initial state
    while Goal/terminal state not reached **do**
        $a \leftarrow max_a Q(s,a)$
        $s \leftarrow s_{t+1}$ taking action a
    end while
end procedure

2.2.4 Exploration vs. Exploitation

There is a fundamental issue in any RL task, where the agent must choose between taking an instantaneous reward by exploiting the policy, or taking a random action to explore the environment in search of a potentially higher long term reward. This problem is illustrated by a well-known RL problem known as the k-armed bandit problem.

> "The agent is in a room with a collection of k gambling machines (each called a 'one armed bandit' in colloquial English). The agent is permitted a fixed number of pulls, h. Any arm may be pulled on each turn. The machines do not require a deposit to play; the only cost is in wasting a pull playing a suboptimal machine. When arm i is pulled, machine i pays off 1 or 0, according to some underlying probability parameter p_i, where payoffs are independent events and the p_i's are unknown. What should the agent's strategy be?" (10)

The amount of time the agent spends in the environment is one factor that can be taken into consideration when making this decision. In general, the longer the agent spends in the environment, the less impact taking an exploratory approach, sometimes towards a sub-optimal policy, will have on the end policy.

One solution to this dilemma is to take an *epsilon greedy policy*. At each state, the agent takes a random action with a probability of ϵ and an action from the policy with a probability of $(1 - \epsilon)$. ϵ is linearly reduced at each iteration to some predetermined floor value. This way, the agent will spend more time exploring at the start of it's interaction with the environment, and will then (hopefully) converge to an optimal policy as it progresses.

2.2.5 Deep Reinforcement Learning

DRL refers to the use of deep learning algorithms within the field of RL. As mentioned previously, RL struggles with environments of high dimensionality. DRL overcomes this issue thanks to the universal function approximation property of deep neural networks and their abilities to isolate and recognize features of interest within high dimensional data and to compactly represent that high dimensional input data (1). DRL can utilize a convolutional neural network to learn a representation of the environment on behalf of the agent through high dimensional sensory input such as video data. A set of fully connected layers are then generally used to approximate the target of the underlying RL algorithm, such as $V(s, a)$, $Q(s, a)$, an action etc. The deep neural network can then be trained using an appropriate variant of the backpropagation algorithm, such as stochastic or batch gradient descent.

The event that brought DRL to the attention of the research community was from the paper mentioned earlier "Playing Atari with Deep Reinforcement Learning, 2013". DeepMind created a variant of Q-Learning called Deep Q-Learning (DQL), that achieved above-human level performance on a large selection of 52 Atari 2600 video games. They combined a convolutional neural network for feature detection, and a fully connected network to learn the Q-Values for all available actions, with ReLU activations between each layer. The network uses a standard Least Square Error loss function in training with gradient Propagation, defined as:

$$L = (y_i - Q(s, a))^2$$
$$y_i = r + \gamma max_{a^*} Q(s', a^*)$$

This architecture was named the Deep Q-network. This breakthrough successfully removes the dimensionality problem, as there is no need to keep a data structure storing all previous experiences. The deep neural network takes an array of RGB pixel information, taken as a stack of 3/4 (depending on the game) greyscale frames, as input to the convolutional network. The fully connected network outputs a vector of Q-Values for each available action in the game.

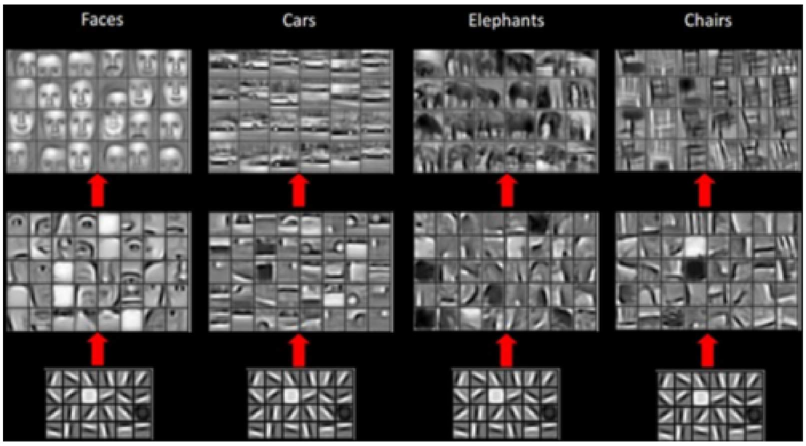

Figure 2.3: Examples of output filters from a convolution neural network

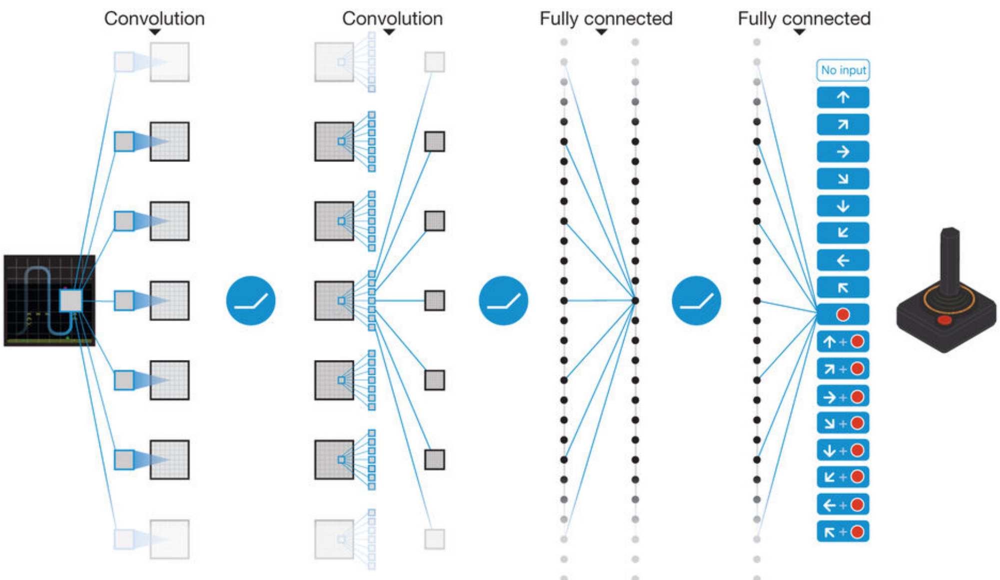

Figure 2.4: The original Deep Q-network Architecture

At the time, this breakthrough was the state of the art. Since then, many improvements and adaptations have been made on this model, by DeepMind and other research groups. Such examples include the Double Q-Learning adaptation (11) and the Dueling Q-network architecture (12). Experience replay and target networks were two techniques added to the original architecture by DeepMind to provide more stability to the learning process.

Instead of learning from the immediately previous experience when training the network, a large memory of past experiences are stored as tuples of $(s_t, a_t, r_t, s_{t+1}, term)$, where *term* is a boolean indicating if this state transition was terminal (a gameover). The network is then trained from a random sampling of past experiences from this replay memory. It was found to greatly reduce the number of interactions the agent needed to have with the environment. However, this technique *is somewhat* limited as there is no way to differentiate important experiences from unimportant ones. In the State of the Art section we will discuss an optimization to experience replay that mitigates this drawback (13).

The target network is a secondary network $\hat{Q}$ cloned from the original Q-network Q, that is used to predict the targets y_i when training Q. The weights of $\hat{Q}$ are cloned from Q every C training steps. This modification makes the algorithm more stable, as an increase to $Q(s_t, a_t)$ was often found to also increase $Q(s_{t+1}, a)$ for all a, thus also increasing the targets y_i. This can create a diverging solution in some cases. Freezing the weights makes the updates to Q and the targets y further apart, decreasing the likelihood of divergence (13).

Algorithm 2 Deep Q-Learning Algorithm with Experience Replay

Initialize replay memory D to capacity N
Initialize Q with random weights θ
for episode = 1, M **do**
    Initialize arbitrary first sequence of states
    for t = 1, T **do**
        With probability ϵ select a random action a_t, otherwise select $a_t = max_a Q(s_t, a)$
        Execute action a_t and observe reward r_t and state s_{t+1}
        Store state transition $(s_t, a_t, r_t, s_{t+1}, term)$ in D
        Sample random mini batch $(s_i, a_i, r_i, s_{i+1}, term)$ from D
        if $term = true$ **then**
            Set $y_i = r_i$
        else
            Set $y_i = \gamma max_a Q(s_{i+1}, a)$
        end if
    end for
    Perform a gradient descent step on $(y_i - Q(s_i, a_i))^2$
end for

The applications have been broadened to fields such as natural language processing and robotics. More detail on the current state of the art will be given in the next section.

2.3 State of the Art

3 Evaluation

4 Conclusion

Bibliography

[1] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *CoRR*, abs/1708.05866, 2017. URL `http://arxiv.org/abs/1708.05866`.

[2] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016. URL `http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html`.

[3] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[4] 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`.

[5] Yves Chauvin and David E Rumelhart. *Backpropagation: theory, architectures, and applications*. Psychology Press, 1995.

[6] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *CoRR*, abs/1207.4708, 2012. URL `http://arxiv.org/abs/1207.4708`.

[7] Claude E. Shannon. Xxii. programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314): 256–275, 1950. URL `https://doi.org/10.1080/14786445008521796`.

[8] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom

Schaul, Hado van Hasselt, David Silver, Timothy P. Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. Starcraft II: A new challenge for reinforcement learning. *CoRR*, abs/1708.04782, 2017. URL `http://arxiv.org/abs/1708.04782`.

[9] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3): 279–292, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992698. URL `https://doi.org/10.1007/BF00992698`.

[10] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[11] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 16, pages 2094–2100, 2016.

[12] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529, 2015.

A1 Appendix

You may use appendices to include relevant background information, such as calibration certificates, derivations of key equations or presentation of a particular data reduction method. You should not use the appendices to dump large amounts of additional results or data which are not properly discussed. If these results are really relevant, then they should appear in the main body of the report.

A1.1 Appendix numbering

Appendices are numbered sequentially, A1, A2, A3... The sections, figures and tables within appendices are numbered in the same way as in the main text. For example, the first figure in Appendix A1 would be Figure A1.1. Equations continue the numbering from the main text.