Reinforcement Learning

Math390 Project Report – Group R1

Bessant R., Croissant L., Davis J., Lang C., O'Donnell M.

University of Lancaster

November 10, 2017

Abstract

Artificial Intelligence is becoming an ever more present topic of public discourse. However, this discourse is rarely led from a technical perspective, hence it suffers contamination from inflated statements about the state of research in AI and the various fields working towards it. In this paper, we will present an overview of one learning paradigm amongst several that populate Machine Learning. This overview will aim to help the reader understand the strengths and limitations of one of the most promising imitations of human intelligence. It is hoped that this paper will help them develop a rounded view of Reinforcement Learning, general enough to understand its history, motivation, basic formalism, contemporary applications, and limitations. After a brief introduction to Machine Learning, we will focus on the various approaches that historically formed Reinforcement Learning before going into a detailed motivational analysis via examples. Afterwards, we will introduce the mathematical framework and apply this knowledge to the analysis of AlphaGo.

Contents

I. Introduction to Reinforcement Learning

i. Machine Learning and Problem Solving

The world is rife with problems to solve. Our human brains have a remarkable natural capacity to recognise and solve problems, derived from millions of years of evolution. From cracking nuts, a problem other primates and animals can solve, to taming the energy of the atom, which only a trace percentage of us understand. These range from the most empirical to the most abstract, from the most mundane to the most mathematical. Mathematics has undeniably been key in solving a large part of humanity's problems and challenges. From physics to climatology, equations underpin modern science. However, this computational power, no matter how great it's impact, was still limited by the human hand solving the equations. But in the crucible of the most devastating conflict in our history, this paradigm would change.

From the ashes of one era, another was born. The computer and Moore's law would bring about a change in computational paradigms. Numerical computations become almost entirely trivial, and soon the computer's speed and accuracy were unfathomable. Notwithstanding this, the problem solving itself remained a human chore.

Over the last 60 years, a new field has blossomed at the confluence of Mathematics, Computer Science, Psychology, and Neurology. This field envisions a computer solving problems in place of man. To paraphrase Arthur Samuel, who in 1959 bestowed the name Machine Leaning onto it[1], this field aims to "give a computer an ability without explicitly programming it". This in practice means that a computer driving a car should be able to learn how to recognise a car on the motorway, behave in a desirable manner around it, and then to generalise its learning to be able to recognise new cars and respond to them in settings it has never encountered before.

While Machine Learning's goal was and still is ambitious, it has been a successful and rich area of research. In the last decade it has also become a very publicised field, as companies started to exploit the huge volumes of data they own. Following Data Science, Big Data and other such buzzwords, Machine Learning seems poised to take the world by storm, as Artificial Intelligence becomes an ever more public issue and as tech giants engage in a tech race. This is not an accident; while Machine Learning is not a new field, it only recently has been able to deploy its potential on large datasets. This is due to increased ease of computation, second-generation products, and the huge increase in available data as the field broadened from theoretical to applied research.

The field has opened into three main learning paradigms tailored to solving different problems: Unsupervised, Supervised, and Reinforcement[2].

Unsupervised Learning is the most general, and requires only a dataset. It focuses on finding patterns of association between data-points, for example by dividing them into "clusters". The name "unsupervised" derives from the designer having no need to oversee the training in any way. Supervised Learning is the one that any reader is most likely to be familiar with. It includes both linear and logistic regressions, and many applications of neural networks. It requires some target variable (y_i in standard linear regression) and some explanatory variables ($x_{i,j}$). These algorithms then learn associations between the y_i and $x_{i,j}$ in order to later predict the y value of new observations. It is called "supervised" as the designer has to provide the true label of the target (i.e. the y_i) so that the algorithm can gauge quality of associations.

What makes Reinforcement Learning (RL) different from the two previous cases is that is interactive. While other methods are designed to extract information from a data set, Reinforcement Learning is designed to make its own decisions about what to learn and when. The idea is that the RL agent, with a goal in mind, will take what it considers the best action in its environment. It will then analyse the consequences of its action to determine if it was beneficial or not. Using this "reward" it will update its analysis of the best action at the next time step. This concept sounds very general and not specifically statistical in nature since this description of learning did not arise from mathematics, and this generality is the first key to understanding the history of Reinforcement Learning. Instead, it arose from the theory of animal learning.

ii. Trial and Error Learning

While the word "reinforcement" itself only appeared in 1927, the psychology research behind it can be traced back to the late 19th century. While Woodworth[3] [1] credits Alexander Bain (1850s) and Conway Lloyd Morgan (1894), who coined the term "trial-and-error", with early ideas in the field. A succinct and elegant formulation is found in the "Law of Effect"[4]. In the section he dedicates to the "Laws of Behaviour in General", Thorndike[4] takes interest in the way animals learn from their environment and past actions, formulating the aforementioned Law:

> "*Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will be more likely to recur; those which are accompanied or closely followed by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that, when it recurs, they will be less likely to occur. The greater the satisfaction or discomfort, the greater the strengthening or weakening of the bond.*"

Thorndike also states a "Law of Exercise":

> *Any response to a situation will, other things being equal, be more strongly connected with the situation in proportion to the number of times it has been connected with that situation and to the average vigor and duration of the connections.*

The connection mentioned here is what is now called association in learning, that is to say the ability to connect cause and consequence. Of interest here is the connection between an action taken in the past in interaction with an environment, and the response (interpretable as a reward or punishment) received.

With hindsight these two principles can be seen as a good basis for computer learning too. Simply put, the Law of Effect would encourage the agent to repeat actions which give it reward and abandon those that give penalty, while the Law of Exercise encourages the agent to use the solution that most reliably triggers a reward or that is expected to trigger the strongest reward. These are the tenets of psychological learning theory that underlie Reinforcement Learning, with the term "reinforcement" itself appearing with the English translations of Ivan P. Pavlov's work on canine salivation reflexes from 1903 onwards[2].

The earliest developments in Machine Learning were achieved with this "trial and error" approach in mind[2], often using mechanical contraptions. These developments spread to digital programming in the 1950s and to the then new Artificial Neural Networks (ANNs). Research on pattern recognition with ANNs at the time considered learning from a "trial and error" perspective[7], but objectively steered pattern recognition towards Supervised Learning. In their report, Woodrow and Hoff detail how they program their ANN in a clearly supervised setting: "*During a training phase, crude geometric patterns are fed to the machine by setting the toggle switches in a 4x4 input array. Setting another toggle switch tells the machine whether the desired output for the particular input pattern is +1 or -1*". As research in pattern recognition and neural networks increased in the 1960s and 1970s, there was a dearth of new research on pure reinforcement methods.

However, there were significant developments during this period such as the work of Donald Michie[8], who applied reinforcement to problems like Tic-Tac-Toe and the balancing of a broom on a moving cart, the latter in direct competition with the work of Supervised Learning researchers like Woodrow and Smith[9].

Besides the applications to specific problems, there is a more theoretical sub-field that had a very heavy influence on Reinforcement Learning and helped renew interest in the field in the late 1970s and 1980s. This sub-field focuses on learning automata, algorithms designed to learn by reinforcement revolving around the famous "k-armed Bandit" problem. This problem is non-associative, unlike those above, and relies only on learning how to pick the optimal action amongst k possible ones. The name illustrates this as it refers

[1] According to Sutton and Barto[2].

[2] While his early work [5] doesn't record "reinforcement" and focuses on physiology of digestion, the term can be found starting in his later work on conditional reflexes[6].

to a "multi-armed bandit" environment with k slot machines, each of which is a "one-armed bandit". In this case, each arm has an underlying reward distribution, unknown to the agent, which must balance exploring new arms, improving the inference on known arms, and playing its estimated best arm.

These automata saw research in Statistics but also in Engineering, Economics and Game Theory, and form one of the key components of contemporary Reinforcement Learning methods and research. To understand contemporary work however, we need to also look at another branch of research that would come to complement "trial and error" learning.

iii. Optimal Control and Temporal Difference

Control Theory is a part of Systems Engineering which deals with the designing of "controllers" to optimally manage a dynamically changing system. This may not seem related and in fact often did not pertain to learning *sensu stricto*[2], but designing an "Optimal Controller" to optimise (very much in the numerical sense) some controllable aspect of an environment with respect to a value function (called the "optimal return function") begins to show resemblance to Reinforcement Learning. The specific approach that is of interest to Reinforcement Learning is called dynamic programming. It requires a value function to be specified and then attempts optimisation by dividing the problem into smaller sub-routines to be optimised.

Most importantly, developments in the field of Control Theory led to the formulation by Bellman of the discrete stochastic control problem. As the name suggests, this version includes a discrete action-space, all actions of which interact with a stochastic process composed of multiple states. This framework of decision over a Markov Chain environment is aptly called a Markovian Decision Process (MDP)[10]. The exact mathematical details are left to section III.ii., as they are not required to understand the convergence of Optimal Control Theory to Reinforcement Learning.

One can see already that MDP that are empirically best solved by Dynamic Programming have the properties we found in an associative learning framework in section I.ii. Indeed, it has an environment (which may be partially unknown) with which it interacts in order to minimise (or maximise) some quantity linked to the system as an iterative process. This begins to resemble the goal of maximising a reward while interacting with an unknown environment, but approaches it not so much from a learning perspective so much as from an optimal algorithmic one. There is another important set of methods that will come to reinforce trial and error and Optimal Control Theory and merge into modern Reinforcement Learning. These Temporal Difference methods are a smaller domain than the ones previously discussed, but by no means an inconsequential one. The core of this approach is to use the difference in successive estimates of a quantity to improve an agent.

In the later 1970s, Reinforcement Learning became once more a topic of great research interest and this renewed interest would bring about modern Reinforcement Learning. Work by Werbos[11] was influential in bringing together trial and error and Optimal Control Theory. He refers to the work done by Samuel on his Checkers program[1] which lies at the very root of Machine Learning, whilst also bringing in the influences of Minsky and his references to "Secondary Reinforcers"[12], the neuroscience term that relates reinforcement by a stimulus that has been correlated with a primary reinforcer to Optimal Control Theory.

Amongst the flurry of new research in Reinforcement Learning, the efforts to finally amalgamate the remaining methods culminated in the development of Q-learning[13] in 1989, unifying Temporal Difference and Optimal Control. Successful applications of Reinforcement Learning brought more publicity to the field in the 1990s, for instance the TD-Gammon algorithm[14] for playing Backgammon, which combined Q-learning with an ANN for functional evaluation. This heuristic of combining computationally effective methods with a core Reinforcement Learning algorithm is still used for example by AlphaGo[3].

[3] See section IV.

II. Motivating Examples

i. Human Learning

To motivate the detailed mathematical exploration of Reinforcement Learning, we will provide in this section a series of examples aimed at illustrating how we can develop a framework for this without mathematical precision. We wish to offer in this section a clear line of thought to help the reader understand the development of Reinforcement Learning, as we believe that the key to understanding Machine Learning is a representational vision. While for many readers a good representation can be derived from the mathematical formulæ, the authors believe that examples can lead to a better intuitive understanding that will allow the reader to discuss and criticise the material. To begin, we will illustrate how Reinforcement Learning can be formulated from neuroscience and the psychology of learning, as outlined in section I.ii. First, we will re-examine Pavlov's work on the conditioning[4] of animal behaviour with the use of association.

In the 1890's, Pavlov conducted experiments measuring the quantity of saliva secreted by dogs upon presentation with a bowl of food. His work showed that salivation is an unconditioned response in dogs, i.e. that this is an instinctive response that they do not need to learn. The next protocol consisted of conditioning of the dogs to the ringing of a bell. The ringing of a bell now preceded the serving of food to the dogs for a given period of time. After this, Pavlov observed that dogs salivated in response to the sound of the bell, even when no food was in sight. This meant that the bell had become a conditioned stimulus, since this response was learned as the sound was now associated with food. This association can be viewed as a simple prediction apparatus by the dog. Having observed that the sound of the bell is strongly correlated with feeding, the dog has learnt to use the sound of the bell to predict imminent feeding.

To think of a mechanical set-up, consider an agent (an analogue of the dog's brain) whose objective is to detect that the dog is going to be fed and subsequently react to the environment (i.e. by salivating before food arrives and not in other cases). Then consider the signal from the bell ringing to be a factor covariate, with levels 1 or 0. Under the training phase (when the food is presented to the dog while the bell is rung), the agent will form a strong association between the ringing factor and the arrival of food. It will then use this predictor to salivate earlier than otherwise, thus optimising the meal. This analogy is fallible like all analogies; it remains difficult from such a thought-experiment to discern the components of a RL agent from each other. However, it shows a few aspects: here we see that the agent forms a predictive model of the world to aid in its actions, and that it learns on-line, while it performs the actions and not in one batch at the beginning. We will attempt to make the distinctions appear clearer to the reader as we gradually move from animal to animal-inspired but purely artificial learning methods.

Childhood learning is very much a trial and error approach. As a child, we grow in our ability to learn and modify behaviours based on the positive and negative outcomes of our actions, so the rewards or punishments. One sees a child repeat a task several times and failing, until by incremental corrections it manages to achieve its goal. While it is not entirely clear what constitutes the border between instincts at birth and learned behaviours, it can be seen that tasks such as riding a bike that would appear too recent to have been adapted to by evolution are probably purely learned. How to devise a reward function should be our first concern to formalise an RL agent to the problem of riding a bike. Setting aside the complexity of simulating the environment, we would like the reward to be a measure that scales with the distance travelled without falling. Thus, asymptotically, a maximum reward will be equivalent to being able to drive the bike an arbitrary distance without accident. Now that we have a reward for each "action", we should specify what the action is. In this case "action" is misleading, as in fact we would consider that at each try the learner implements a set of rules or "policy" about what actions the various muscles in its body should take in order to move. Based on the reward obtained, it would then tweak a subset of the actions in the policy and compare the difference in rewards at the next step to select the correct changes to its policy over time. While this apparatus is very simple and still not shockingly mechanical, it already begins to resemble

[4]Pavlovian conditioning is the use of an existing reflex response to engineer a learned reflex into an animal.

an algorithm. A similar problem[8], which has been mentioned earlier, consisting of balancing a broomstick on a moving cart, which you will see is quite similar with a household appliance in place of a child to be balanced on a moving support, is a historically important topic. It is somewhat of a benchmark for competing methods of problem solving via learning. Whilst childhood learning is a fascinatingly efficient process, which displays the ability to memorise, learn, and generalise on an incredibly large list of problems and topics, it is also fundamentally complex and not wholly understood. This leads us to examine smaller examples of trial and error learning in adult humans with rewards and punishments from interacting with the environment, so that we can identify approaches to trial and error learning and problems to which it can be applied.

ii. Games as a Proof of Concept

There has been a lot of application of Reinforcement Learning to the learning of optimal strategies in games. This is very much motivated by the similarities with human learning. Indeed we, during our lifetimes, come to learn a lot of games, and we do so as we play them. We learn how to play games to impressive strategical levels, but without ever having to memorise all possible combinations of play. Two important aspects of games make them a good application for Reinforcement Learning: their formal rules, which can be modelled and understood easily by mathematics and computers, and their inherent reward construction of loss, draw, and victory. While Game Theory had previously established many theoretical results on games, the interest of Reinforcement Learning is to learn the same optimal results with no supervision and then to extend this to games whose complexity is too great for a simply theoretical treatment. In this report we present two such games, in this section with Tesauro's TD-Gammon[14] and in section IV with DeepMind's AlphaGo programs.

Before delving into TD-Gammon, the authors would like to briefly pause and discuss the role of heuristics in contemporary game-playing RL agents. You have most likely heard a lot about neural networks, Deep Learning and AlphaGo in the last year. We would like to help untangle this to allow the reader to see how the next examples (which are very much predecessors of AlphaGo) developed into DeepMind's work. A simple heuristic for playing games is called "look-ahead". It is very much a human heuristic, as we tend to evaluate the future states of play conditioned on each action we can take this turn in order to determine which leads to the "best" outcome. In a more formal setting, algorithms tend to use tree-search methods to evaluate the future possible positions of the board based on the actions it is considering, and pick the move that goes down the branch with the highest associated probability of winning. This is an excellent heuristic for games as it compels the learner to look at long term reward while retaining very high computational efficiency due to the tree structure. However, this is still a heuristic design, and not a necessary component of pure Reinforcement Learning. In section IV.iii the details of the most up to date methods are given. Deep Learning is simply an architectural choice in the way to order individual artificial neurones in an ANN. Whilst it has received a lot of press for its successes in recent years, it is important to note that it is applied to Reinforcement Learning without being an inherent part of it, much like tree-searches. Neural networks have been used for Reinforcement Learning for some time, and they are only a support method for function evaluation, a very complex set of estimation methods for a function $f : X \mapsto Y$.

In essence, to implement a Reinforcement Learning algorithm for complex and computationally intensive methods, it will often be beneficial to use well established work in other parts of Machine Learning as the building blocks of implementation, while the methodology of learning remains reinforcement: online, non-supervised, and interactive. With this distinction between the learning paradigm and implementation choices we hope well established, we will now analyse two applications, a simple Tic-Tac-Toe [5] and the TD-Gammon algorithm. Board games such as these are often used to test machine learning algorithms as they offer complexity and the opportunity to learn, plus they have clearly defined reward and punishment within the games.

When programming an RL agent to solve simple games we must look for the optimal outcome of many

[5] Detailed in Sutton and Barto[2].

moves and not just what might give a short-term reward. Thus, we can view it as the equation[6]

$$V(s) = V(s) + \alpha\Big[V(s') - V(s)\Big]$$

where s is the current state and s' is the state after the next move, and $V(s)$ is the estimated value of s where α is a small positive fraction called the step-size parameter.

This describes a trial and error search with the possibility of a delayed reward, where the RL agent can use this value function method to evaluate individual states and sacrifice short-term losses for the greatest long-term reward. Though the game Tic-Tac-Toe is a simple one, it is a good basis to see how Reinforcement Learning works in practice with the application of this equation since it has relatively few states. The subsequent step is then to scale this to larger problems.

Temporal Difference Learning is a Reinforcement Learning method that is based off predictions. An infamous example of this is TD-Gammon: a neural network that was designed to learn the game of Backgammon using Reinforcement Learning. Not only was this a revelation in the world of Backgammon as it was the strongest Backgammon playing program at the time, it was even more significant for the field of Machine Learning. The program trained itself by playing games within its neural networks until it began to learn elementary strategies, and after even more training games it started to excel.

As we will also mention later in section IV.i, programs such as TD-Gammon and AlphaGo offer an opportunity to look at games differently, to the extent that human players study the gameplay of these programs' matches for the inspirational moves that have brought about changes of technique in some of the top players in the world.

This introduction to RL agents should help in understanding the way a problem is learned and optimally solved. The following section will offer an overview of the mathematical foundations which are required to be able to navigate RL literature independently.

III. Mathematical Framework

i. Modelling the Environment

For this section, a background in stochastic processes should not be necessary though it will prove helpful. Before describing the Markovian Decision Process (MDP) framework, the authors would like to outline the key elements we need to include.

Since the environment of our agent is most likely too complex to model with certainty, we would like to use a stochastic setting where there is a finite number of states between which the environment can move. We would therefore also like to have a finite number of actions that our learner can take. Because our environment is considered random, we will want to model the rewards as random variables.

Another important distinction is wether environment is fully observable or only partially observable. Fully observable means the agent is at every step aware of all the information about the environment, and could for instance be a game board. The partially observable setting implies the agent does not have all the information about the environment, but has some element it can use to make inference. This could, for example, be because of the cost of measuring the environment, or of technical limitations such as the line of sight of a camera used by a driverless car. We will not detail distinctions in this paper, however the reader should keep this in mind when approaching new problems.

Let us establish now the notation we will use in this section. The environment will consist of states s belonging to a state-space $\mathcal{S}$, taking actions a from an action-space $\mathcal{A}$, and obtaining rewards r belonging to a set $\mathcal{R}$. Since these will be sampled at random we will denote S, A and R the random variables associated with states and rewards[7]. We will often want to refer to the probability of moving to a state s at time

[6] Provided in Chapter 1 of Sutton and Barto[2].

[7] Following conventional notation we use upper-case letters for random variables (e.g. S_t) and lower-case letters for realisations (e.g. s_t).

t, which we denote $P(S_t = s)$. In this section we will assume that time is discrete and that each index t belongs to $\mathcal{T}$ an interval in $\mathbb{N}$.

To model our environment we will further assume that the Markov property holds and make use of a Markov Chain. A stochastic process which has states $s \in \mathcal{S}$ is a Markov Chain if, it follows the Markov property:

$$\forall s \in \mathcal{S} : P\left(S_{t+1} = s | S_t, S_{t-1}, \cdots, S_1, S_0\right) = P\left(S_{t+1} = s | S_t\right) \ . \tag{1}$$

The Markov Property states that the changes of the process only depend on the last state the process was in. This property can be extended to an acting agent obtaining random rewards R, if we simply consider the joint conditional probability of moving into a state s_{t+1} and obtaining a reward r_{t+1} given sequences of past states, rewards, and actions.

ii. Markovian Decision Processes (MDP)

We can reformulate the Markov property to define a MDP as any decision process where the following holds:

$$\forall (s, r) \in \mathcal{S} \times \mathcal{R} : P\left(S_{t+1} = s', R_{t+1} = r' | S_t, A_t, R_t, \cdots, R_1, S_0, A_0\right) = P\left(S_{t+1} = s', R_{t+1} = r' | S_t, A_t\right) \ . \tag{2}$$

The specification that a Markovian Decision Process must follow the Markov property means the set of one-step transition probabilities[8] $P_t(s', r'|s, a) := P\left(S_{t+1} = s', R_{t+1} = r' | S_t = s_t, A_t = a_t\right)$ completely describes our system at time $t \in \mathcal{T}$. Note that we use the subscript t to indicate that this probability might not be the same at each time step. If it is, then this process is termed homogenous.

A MDP has a stochastic environment composed of states which react to the agent's actions, and rewards that are given to the agent based on its actions. However we consider these to be random variables and that the system has no memory, as it depends only on its current state to determine its next state (by the Markov Property). This fulfils our requirements.

Now that we have described the MDP setting, it should be simple for the reader to see how to derive quantities holding information about the system and the agent's behaviour. In general, this will involve using the law of total probability to derive summations over $\mathcal{S}$, $\mathcal{A}$, or $\mathcal{T}$. We provide as an example the derivation of the expected reward from taking action a_t whilst in state s_t:

$$\begin{aligned} r_E(a_t, s_t) &= \mathbb{E}\left[R_t | S_t = s_t, A_t = a_t\right] \\ &= \sum_{r \in \mathcal{R}} r \times P(R_t = r | S_t = s_t, A_t = a_t) \\ &= \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(r, s' | s_t, a_t) \ . \end{aligned} \tag{3}$$

iii. Valuation and Decision Making

The agent's policy is a map $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ such that $\pi(a, s) = P(A_t = a | S_t = S)$. For each combination of state and action, the policy gives the probability our agent, whilst in that state, will take that action. The policy is what determines the actions our agent takes, and its gradual modification is the learning. The reinforcement comes from the re-evaluation after each time step of the policy based on the reward obtained.

We define the value function $\nu_\pi(s)$ of a policy π which gives an indication of how valuable it is to be in state $s \in \mathcal{S}$ for a learner using policy π. Since we are in a stochastic setting, it is beneficial to view this as the expected reward obtained starting in state s with policy π, so that

$$\nu_\pi(s) := \mathbb{E}_\pi\left[\Psi_{\mathcal{R}}(t) | S_t = s\right] \ . \tag{4}$$

[8] From state s to state s' and receiving reward r' while taking action a.

In a simplest case, we take $\Psi_{\mathcal{R}}(t) = \sum_{\tau>t} R_\tau$, the sum of future rewards. But in many cases one might want to re-weight the value or R_τ as a function of τ. For example, if we had a trading algorithm then recieving a small reward now leads to more opportunity for investment, so we might want the value of rewards to decrease with time. This can be done using a discount factor $\delta \in]0,1[$ with $\Psi_{\mathcal{R}}(t) = R_{t+1} + \sum_{\tau>t} \delta^{\tau-t} R_\tau$, where $\Psi_{\mathcal{R}}(t)$ denotes the function of rewards whose expectation is to be maximised as it uses a time step t as an argument in determining its sum component, and it is dependent on the random rewards $\mathcal{R}$. We hope that this makes it clear that it may depend on the increment of time, and that it is a random quantity. We also use $\mathbb{E}_\pi$ to denote the expectation computed under the assumption that the agent applies policy π.

More interesting than the value function at a state s is the action-value function q_π, under policy π, for an action a in a state s, which builds on the value[9] function to measure the value of the next possible actions. It is defined as:

$$q_\pi(s,a) := \mathbb{E}_\pi \left[\Psi_{\mathcal{R}}(t) | S_t = s, A_t = a_t\right] = \nu_\pi(s') \text{ where } a_t(s) = s' \,. \tag{5}$$

As we can see, the action value of (s,a) is the value of the state that the action a takes the agent to. We can use the action value to pick an optimal decision from state s, by choosing $\arg\max_{a\in\mathcal{A}} q_\pi(s,a)$.

Two ways to maximise the policy, i.e. to learn the policy yielding the highest reward, are included in the appendix: value-iteration and policy-iteration. Both of which fundamental methods for solving MDPs and have been taken from the textbook "Introduction to Machine Learning" by Alpaydin[15].

iv. Exploration vs. Exploitation

Whilst Reinforcement Learning excels in its interactivity with an unknown environment, it can not perform perfectly due to a trade-off between exploration and exploitation. In almost every practical situation, the partially known environment will need to be explored to gather information about it, such as which actions seem to trigger which rewards. While the agent wants to exploit the highest-value action, it must explore actions with unknown or suboptimal rewards first in order to do so. This is the linchpin of algorithmic research in Reinforcement Learning. Algorithms are compared based on how well they perform on this trade-off, which is inherently tied to how well they maximise reward.

Consider a setting where there are k possible actions which each have a reward distribution[10]. We will assume we play n turns. We will compare two simple algorithms in this fixed setting: Explore-Then-Commit and ϵ-greedy. The Explore-Then-Commit strategy is probably the most simple algorithm one could devise. It simply plays each action a fixed number of times m, then takes the action of maximum average reward over its exploration phase and plays it the remaining $n - km$ times. This is is problematic for two reasons: the algorithm can choose the wrong action to exploit if m is too small, and it will waste a lot of actions exploring if m is too large.

At each opportunity for action, the ϵ-greedy agent has a small $0 < \epsilon < 1$ random chance of selecting one of the current sub-optimal actions, otherwise it simply takes the action whose average rewards is higher. Note that the interesting algorithm here uses ϵ as a decreasing function of t instead of a constant. Then we can have an algorithm which will explore frequently early on, building a robust estimate of the means whilst exploiting efficiently as it goes. It will however re-evaluate its estimate of the best action over time, which means it will avoid exploiting the wrong action asymptotically.

However, there are even better algorithms, such as the UCB algorithm for stochastic bandits. This algorithm only explores actions which appear to have low reward if there is large uncertainty in the estimate. Setting aside the theoretical examples, we will now focus on a real-life example which has been widely reported upon in the media: Google DeepMind's AlphaGo.

[9] It is important the reader note here that we are moving from reward to value in the lexicon.

[10] This is a similar setting to a stochastic k-armed bandit.

IV. Case Study: AlphaGo

i. The Game of Go

Go is a two-player strategy board game of great complexity that has been played for millennia, originating in China. Played on a square Go board, the aim of the game is to gain a larger 'territory", i.e. surround a larger part of the board with your stones or 'Go pieces" which are placed on the intersections of the board's grid.

AlphaGo is a computer program built specifically to play the game Go, and is the first to defeat a professional Go player and a Go world champion. It is also the first program to receive 9 Dan Pro Rank, the highest rank possible for a professional Go player.

The game of Go was considered a milestone for AI as one of its "grand challenges" due to having such a great complexity; there are more possible positions on the Go board than the number of atoms in the universe. Before AlphaGo's extraordinary triumph, any attempt at conquering the game of Go with AI had been comparable only to amateur human players. The concept of a program such as this defeating professionals was inconceivable.

AlphaGo shook the foundations of the traditional styles of playing in its matches against human professional players, making moves that were unheard of in the hundreds of years of established knowledge on Go strategies. Described as having a "unique perspective that it brings to every game"[16], players have pored over the moves that AlphaGo made in its revolutionary games, studying them to enhance their own Go playing abilities. Though we will not discuss the gameplay in this paper, we will look at the Reinforcement Learning behind AlphaGo's choice of moves that had a monumental effect on this iconic game.

ii. Mechanics of AlphaGo

What constitutes AlphaGo[17] is a valid question. To start from the trivial, it is a computer program designed by researchers at DeepMind (owned by Google since 2014) in 2015. It consists of deep convolutional neural networks, a state-of-the-art architecture for extremely large scale computation on difficult learning problems such as computer vision. These networks take on as input the 19×19 board, and are trained using Supervised Learning to predict the next best move by watching recorded human professional matches. This provides initial estimates for the policy, which are then improved using Reinforcement Learning, with the machine playing against a previous version of its policy. In each game, the steps of the playing phase hold no reward, while a victory or loss carry rewards $+1$ and -1 respectively.

In more detail, AlphaGo has three components; a policy network, a value network and a Monte Carlo Tree Search (MCTS) algorithm. The policy network is the one which is trained as described above, and it is simply a neural network which evaluates π using an incredibly complex function based on its training. The value network is trained next, using a simulated dataset of thirty million game board arrangements and observing whether they led to victory when the policy was applied against itself[11]. This sample allows the value network to output a single value given any state; whether it thinks it will win or lose.

The Monte Carlo Tree Search component is slightly different. MCTS is, as the name suggests, a tree-search algorithm. It is a tree structure where each node is a state of the board, and each branch consists of an action taken on the previous node. Therefore, at each node the next possible states are the nodes beneath it and the branches to these are the possible actions. The MCTS will search from a root (the current state) down the tree, and helps the rest of the program decide which is the next best move by evaluating which branches (i.e. successive actions) are most likely to yield victory, whilst gaining computation efficiency. While MCTS is a very interesting topic, and they can be used as Reinforcement Learning methods, we will not go into further detail in this paper. More information about the detailed mechanics of AlphaGo and how it works can be found in the papers by its creators at DeepMind. We will include one figure courtesy of their publication[17].

[11] There is a clear winner and loser as each board arrangement also has information about who played into it by placing the last stone.

The neural network training follows a "pipeline"-like structure, which is illustrated in Figure 1.

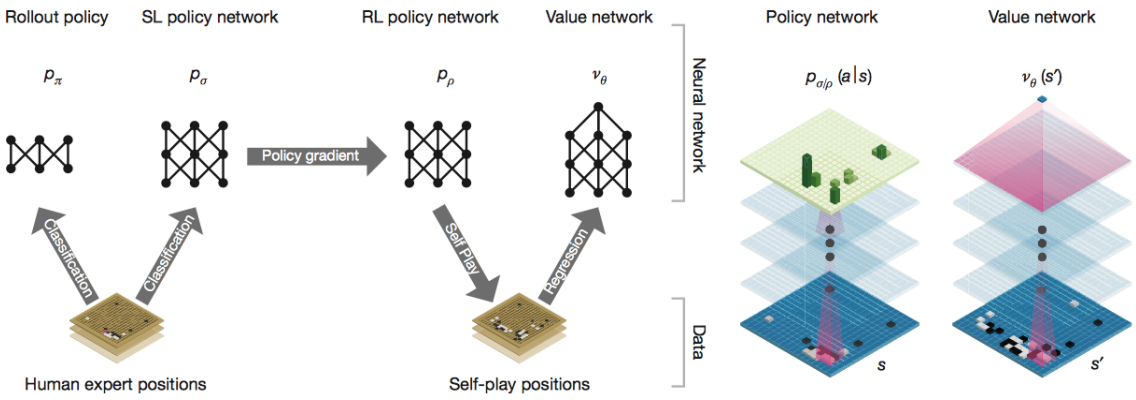

Figure 1: Simplified training pipeline for AlphaGo.

As we see on the right of Figure 1, the network applies policy gradient[12] to the supervised training step to improve the policy, which it then applies against itself to determine the value of each state. The policy network takes into account the state of the board (in blue) and suggests various moves with various probabilities represented by the height of the green columns, whilst the value network (in pink) outputs a single value assigned to state s' of the board.

AlphaGo has an advantageous approach to playing since its moves always focus on the long-term reward, calculated with the Reinforcement Learning value function at each move in the game. The short-term reward can be disregarded if the value function gives a larger probability of winning despite the short-term losses, whereas human players might overlook this in favour of small apparent victories in the game.

iii. AlphaGo Zero

This case study highlights the significance of Reinforcement Learning as a means to produce self-learning software capable of exceeding human ability. It is at the forefront of Reinforcement Learning research, with new versions of the initial program still being released, such as AlphaGo Zero[18] just a few months ago.

The Zero version removes the Supervised Learning program and learns only through self-play. It is initiated only with a knowledge of the rules of the game. Despite this, it has exceeded the ELO rating of all previous AlphaGo versions and human world champions in a matter of weeks. Whilst its structure remains basically the same as outlined above, the key changes made were:

- The input now only uses the position of the black and white stones or "Go pieces" and not an image of the board itself
- The value and policy networks have been combined into a single more efficient neural network
- Improvements made to the MCTS algorithm used have further boosted its computational efficiency.

The development of AlphaGo demonstrates how promising research into Reinforcement Learning is, and just how fast it is advancing. Whilst the resources behind the project are immense, it is after all owned by Google, one can not argue with the success it has had. AlphaGo has conquered a hallmark challenge of Machine Learning, being the first ever program to master the game of Go. The progress made since AlphaGo Lee defeated Lee Sedol, eighteen times Go world champion, in March 2016, shows that it has achieved super human capacity. Moreover, the fact that the latest version of AlphaGo surpassed all previous ones, while only playing itself, shows that Reinforcement Learning can be a highly effective

[12] See appendix A.

learning method. Reinforcement Learning is a dynamic field, and currently we must simply wait and see what progress is made next.

V. Closing Remarks

i. Limitations

Whilst Machine Learning and AI have made huge strides in the last seventy years, no research has yet approached a true General-Purpose Intelligence. Machine Learning contains many problem-solving methods, some of which succeed on remarkable ranges of problems, but no single method, architecture or engineering feat has so far been able to succeed at all known categories of problems. Research such as Reinforcement Learning has shown that it is possible for a computer to achieve super-human results without supervision but not independently. Indeed, whilst AlphaGo Zero may well be on the verge of invincibility at Go, it is unable of learning anything but Go. AlphaGo's design is inherently (from its input structure to its program rules for the game) bound to the problem of playing and winning Go.

It has been shown in this paper that Reinforcement Learning methods have mastered many games. However, building an agent capable of the plasticity of the human brain is still a daunting challenge. It is quite possible that it will happen, in time, and that Artificial General Intelligences will arise, outclassing humans at every aspect of thought. However, in the authors' opinion we should not consider this an immediate or even existential threat. Simply put, we are too far away from imitating the abilities of the human mind.

As the reader now knows, cognitive sciences have heavily contributed to the development of Machine Learning, even before the neural network had become a staple method. However, the current state of cognitive sciences and evolutionary biology does not explain all the mechanisms in our brain, some of which might be in future the basis for self-organising, self-structuring and self-improving general purpose neural networks, whose architecture would arrange itself independently to a new problem.

This is pure speculation, but even speculation on this topic quickly runs into daunting problems. For instance, if we had such a neural network which the engineer could not modify, how could it identify what constitutes a problem? Why would what essentially amounts to an artificial brain motivate itself to start recognising shapes, or taking actions? Where would such impetus come from?

Following the tradition of imitating the animal case, we also rapidly run into philosophical problems. Our neural network would need some kind of supreme reward-punishment function, akin to a natural selection for machines or perhaps a notion of pleasure and pain, which would appear to require a clear distinction between the network's self and non-self. Here we have seemingly drifted to the philosophical considerations of body and mind as perhaps necessary to each other.

We hope that by this short digression we have conveyed to the reader that the considerations of true Artificial Intelligence are still wrapped in a thick shell of unanswered conundrums and profound questions about what it means to be conscious. We hold that this would incline the reader to agree that a rogue super-intelligence is not a serious immediate threat. For another perspective on what needs to be done for artificial intelligence, from fifty-six years ago no less, we recommend the remarkable paper by Minsky[12], *Steps Towards Artificial Intelligence*[13].

ii. Ethical Considerations

A more pressing issue is the debate about applying specific purpose programs such as driverless cars and the associated ethical considerations. This is also a hotly debated topic in the public sphere, but in the authors' opinion has suffered from being dragged down by the lack of technical considerations. Some topics that are often brought up are:

- Neural networks are "black-box" algorithms and can not be held accountable or scrutinised for bias

[13] We must thank Sutton and Barto for this recommendation, which, as they put it, "is well worth reading today"[2], to which we could not agree more.

- Autonomous computer programs should not be able to have life or death power
- Automation poses a threat to society due to a collapse in employment.

In reviewing these ethical problems, we must begin by conceding that neural networks are indeed black-boxes and can not be directly interpreted. It is also true that they often find a very different set of features to examine than we would like. They have a bad tendency to take into account very complex patterns seemingly at random, and not edges or shades as we would hope. However, in many situations involving a human decision maker, it is very hard to determine what exactly led to a decision and what biases the decider may have, explicitly or implicitly. Not to mention the fact that the human brain is, however much we wish to believe we are purely rational beings, a black-box too.

There is the risk of any Machine Learning algorithm performing poorly and appearing to develop bias, but we also know that no such bias is inherent in the algorithms, as they are independent of the data used to implement them. These apparent biases arise due to the imperfect world in which they must be implemented. It is common knowledge that mathematical models are imperfect and rely on sets of assumptions and parameters. This is particularly true for large scale computationally intensive models like those in Machine Learning in general, and Reinforcement Learning in particular. It is important that the people implementing models are able to successfully diagnose issues and check for assumptions. This is common practice and very well documented for simple models like linear regression, and an informed scientific debate about education and research in this vein appears to us much more valuable. Such research exists[19], however it does not make the headlines as it does not beat anyone at Go. Ultimately though, this is going to be on the shoulders of engineers in companies implementing the software, not academics, so we must ensure that such knowledge is available and used in implementation contexts.

To address the second item, we won't argue whether or not computers should hold life and death power; this is a very valid extension of the question of whether any person or group should hold life and death power over another. Instead, we will look specifically at the case of self-driving cars. You are most likely familiar with a common principle of robotics, Asimov's three laws:

1. "A robot may not injure a human being or, through inaction, allow a human being to come to harm."
2. "A robot must obey orders given it by human beings except where such orders conflict with the First Law."
3. "A robot must protect it's own existence as long as such protection does not conflict with the First or Second Law."

In the case of self-driving cars in a situation where at least one car is going to receive damage enough to end the lives of the people inside, one could argue the best answer would be to allow the car with fewer people in to take the worst of the impact. However, this implies the driverless car must decide who lives, which is an ethical conundrum. In a survey[22], 34.2% of people agreed to self-sacrifice if it meant a lower death rate, but can we apply this to real life by letting the cars make these decisions for us?

After all the contributions of mathematics, it seems clearly established that it is a pillar of our society. But then we wonder how can mathematics be used to make better informed decisions about the world around us? How can we use the foundational knowledge of the discipline to establish metrics, evaluate data, process decision, and improve our society? Should the percentage chance of death on a 1km drive be used as a metric of the safety of our roads to decide which cars should be allowed on them? Should we factor in emotional considerations, and if so how? It is our belief that this line of questioning is a much more fertile and beneficial debate than the common AI concerns.

The final dilemma for all types of Machine Learning and Artificial Learning is the shift in employment. If jobs which humans currently do can be automated then the number of jobs will go down. In the past, shifts in the labour market due to increased productivity have generally been compensated by new jobs in other domains. For example, the industrial revolution raised agricultural productivity but created industry jobs for agricultural workers migrating to the cities. It is feared that the new wave of automation

spearheaded by Machine Learning, and in particular Reinforcement Learning, will destroy jobs but not create new ones.

If it is the case that Machine Learning becomes much more efficient and less costly than human labour, it could possibly begin to occupy the new jobs it creates instead of leaving them for humans. Think of Machine Learning agents writing the code for new agents instead of human software engineers. This could pose severe problems in the labour market. The future is always unpredictable, but in the medium-term we can expect the demand for technical expertise in mathematics and computer science to keep rising. Holding a debate now about fostering digital and programming skills in school children will hopefully yield a generation of young graduates in twenty years who will be ready for the high-skill digital economy and will find employment.[20]

There is a need for a healthy debate around the ethics and best-practice of Reinforcement Learning, but it is also important that we remain level-headed and do not cause suspicion about Artificial Intelligence and Reinforcement Learning. We believe it is essential to remember the positive impact of Machine Learning and the future opportunities it offers, and not cede to fear-mongering. There is a lot of work yet to be done and a lot of informed debates to be had, but we hope to have shown to the reader why Reinforcement Learning should be seen as an exciting development in science and that it should be encouraged and not feared.

A. Appendix: Optimal Policy Algorithms

Initialize $V(s)$ to arbitrary values
Repeat
    For all $s \in S$
        For all $a \in \mathcal{A}$
            $Q(s,a) \leftarrow E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a)V(s')$
        $V(s) \leftarrow \max_a Q(s,a)$
Until $V(s)$ converge

Figure 2: Pseudocode for value-iteration algorithm[15].

Initialize a policy π' arbitrarily
Repeat
$\pi \leftarrow \pi'$
Compute the values using π by
solving the linear equations
$V^{\pi}(s) = E[r|s, \pi(s)] + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V^{\pi}(s')$
Improve the policy at each state
$\pi'(s) \leftarrow \arg\max_a (E[r|s, a] + \gamma \sum_{s' \in S} P(s'|s, a) V^{\pi}(s'))$
Until $\pi = \pi'$

Figure 3: Pseudocode for policy-iteration algorithm[15].

B. References

[1] Samuel, A.L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, 3:211-229.

[2] Sutton, R., Barto, A. (1998), Reinforcement Learning: An Introduction, *MIT Press,* Cambridge, MA.

[3] Woodworth, R. S., Schlosberg, H. (1938), Experimental Psychology, *Henry Holt and Co.,* New York.

[4] Thorndike, E. L. (1911), Animal intelligence; experimental studies, *The MacMillan Company,* New York, p.244.

[5] Pavlov, I. P. (1904), Nobel Lecture: Physiology of digestion, *The Nobel Foundation*, Stockholm.

[6] Pavlov, I.P. (1927), Conditional Reflexes, *Oxford University Press*, London.

[7] Widrow, B., Hoff, M. E. (1960), Adaptive Switching Circuits, Stanford Electronics Laboratories Technical Report 1553-1 .

[8] Michie, D., Chambers, R.A. (1968) BOXES: An experiment in adaptive control. *Machine intelligence*, 2(2):137-52.

[9] Widrow, B., Smith, F.W. (1964), Pattern-recognizing control systems, *Computer and Information Sciences (COINS) Proceedings*. p312.

[10] Bellman, R. E. (1957). Dynamic Programming, *Princeton University Press*, Princeton.

[11] Werbos, P. J. (1987). Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research, *IEEE Transactions on Systems, Man, and Cybernetics*.

[12] Minsky, M. (1961) Steps towards artificial intelligence, *Proceedings of the Institute of Radio Engineers*.

[13] Watkins C.J.C.H. (1989). Learning from Delayed Rewards, Ph.D thesis, King's College, Cambridge University.

[14] Tesauro, G.J. (1995). Temporal difference learning and TD-Gammon, *Communications of the ACM*, 38:58-68.

[15] Alpaydin,E. (2014), Introduction to Machine Learning, Third Edition, *MIT Press*,Cambridge, MA.

[16] DeepMind (2015). `https://deepmind.com/research/alphago/` .

[17] David Silver., Aja Huang., Chris 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., Demis Hassabis. (2015). Mastering the game of Go with deep neural networks and tree search, `https://storage.googleapis.com/deepmind-media/alphago/AlphaGoNaturePaper.pdf` .

[18] DeepMind (2017). `https://deepmind.com/blog/innovations-alphago` .

[19] Olah, C., Mordvintsev, A., Schubert, L. (2017) Feature Visualization, *Distill*, 2017.

[20] Pavaloiu, A., Kose, U. (2017). Ethical Artificial Intelligence - An Open Question, *Journal of Multidisciplinary Developments*, 2(2), 15-27.

[21] MIT Technology Review. (2014). Do we need Asimov's laws? `https://www.technologyreview.com/s/527336/do-we-need-asimovs-laws/` .

[22] Schoettle, B., Sivak, M. (2014). A Survey of Public Opinion about Autonomous and Self-Driving Vechicles, in the U.S., the U.K. and Australia. The University of Michigan Transportation Research Institute, report No. UMTRI-2014-21.