1 Introduction

1.1 Controlling Dynamical Systems

Dynamical systems have long been of great interest to engineers and scientists due to their ability to readily describe real world phenomena. They describe how a system changes through geometric space with time and, in principle, their trajectories can be predicted solely from their initial state. In recent years advances in computing power have allowed numerical methods to solve non-linear systems with relative ease. However, often a precise solution is of less import than a prediction of the long term qualitative behaviour of the system. Much effort has been made to find methods to describe this long-term behaviour such as those made by Lyapunov in stability theory [1].

A corollary of the importance of dynamical systems is that influencing their behaviour is particularly useful. Highly non-linear models are difficult to control optimally and robustly, and the few mathematical techniques developed to deal with these can only handle very small subcategories of problems [1, 2]. The study of more general techniques to solve complex control problems has recently come to the forefront of the field with the advent of machine learning techniques such as reinforcement and deep learning. Many of these are not aimed specifically at control problems and are often designed to play games. This project looks at adapting one such algorithm - AlphaZero - from playing board games to solving general control problems such as the control of an aircraft in flight or the control of under-actuated robotics by starting with a simple inverted pendulum.

1.2 Control Theory

The first attempts at controlling dynamical systems came from classical control theory, which consisted of a series of "cut-and-try" techniques based largely on adjusting the gains of PID controllers and lead/lag compensators until satisfactory closed loop dynamics were achieved [4].

It wasn't until the unification of the calculus of variations, classical control, random process theory, and linear/non-linear programming by Bellman in the 1950s [4] that truly optimal control was discovered. Optimal control consists of finding a control law for a

specified optimality criterion and can be thought of as a non-linear feedback law, $u(t) = -K(t)x(t)$ based on the initial state, $x(0)$. In discrete time, an optimal control law can be found via *dynamic programming* (DP). DP is a systematic procedure for transforming an optimisation over a sequence of h inputs into h minimisations over 1 input (but for all states). The dynamic programming equations are:

$$V(x_k, k) = \min_{u_{k:h-1}} \left(\sum_{i=k}^{h-1} c(x_i, u_i) + J_h(x_h) \right) \tag{1.1}$$

$$= \min_{u_k} \Big(c(x_k, u_k) + V(x_{k+1}, k+1) \Big) \tag{1.2}$$

$$u_k^* = \operatorname*{argmin}_{u_k} \Big(c(x_k, u_k) + V(x_{k+1}, k+1) \Big) \tag{1.3}$$

$$\tag{1.4}$$

Where $c(x_i, u_i)$ is the cost as a function of the state and input at time i, $J_h(x_h)$ is the terminal cost, and $V(x_k, k)$ is the value function.

This enables a backwards recursion to find a sequence of value functions. This can be solved for the linear case with quadratic costs analytically (known as the linear quadratic regulator, LQR) or, if the system is non-linear, via gradient descent. However, over a finite horizon this is essentially open-loop control. Optimal control of this form has been used to control complex dynamical systems such as spaceflight and aileron folding on aircraft [5, 6]. A closed loop extension to this is Model Predictive Control (MPC), which employs a receding horizon rather than a finite or infinite horizon. MPC can therefore easily deal with plant disturbances and uncertainties, constraints, indefinite horizons and can also be extended to get a control law for non-linear systems. MPC has recently been shown to work in trajectory control for interplanetary rovers [7]. Optimal control is limited in that it requires sophisticated models of the environment/plant, and it generally struggles with highly non-linear models (state of the art is currently linearisation about the predicted trajectory). Furthermore, it is only feasible to "grid" up to 5/6 dimensions in discrete cases [2].

The inverted pendulum is a seminal problem in control theory. It is inherently unstable, under-actuated, dynamically simple, yet highly non-linear making it an ideal teaching aid. For the inverted pendulum system, a standard method of controlling the pole is to use swing-up control followed by LQR. Swing-up control aims to find a homoclinic trajectory through energy shaping to drive the pendulum to the unstable equilibrium. The region near the equilibrium can be approximated as linear and therefore LQR can optimally stabilise the pendulum near it. This is a particularly good method for the inverted pendulum, but it is difficult to generalise to more difficult systems [8].

1.3 Reinforcement Learning

Two further generalisations to dynamical systems and optimal control as defined in eq. (1.1) are stochastic dynamics and indefinite horizons (i.e. episodic tasks). This discrete time stochastic control process is known as a Markov Decision Process (MPD). In MDPs the cost function is often written as a reward function and, due to the indefinite nature of the process, the value function for the next step is discounted (where $\lambda \approx 0.9$ typically):

$$V(x_k) = \max_{u_k}\left(\sum_{i=k}^{\infty} \lambda^{i-k} r(x_i, u_i)\right) \tag{1.5}$$

$$= \max_{u_k} \mathbb{E}\Big[r(x_k, u_k) + \lambda V(x_{k+1})\Big] \tag{1.6}$$

$$u_k^* = \underset{u_k}{\operatorname{argmax}}\, \mathbb{E}\Big[r(x_k, u_k) + \lambda V(x_{k+1})\Big] \tag{1.7}$$

Reinforcement learning (RL) aims to learn the optimal policy, $\pi^*(x_k)$ ($= u^*(x_k)$ in control) of an MDP. This differs from optimal control in its inherent stochastic nature and therefore can lead to intractable search spaces. A solution to this is to learn form sample trajectories. Algorithms such as Q-Learning, SARSA and DYNA have recently had great success in control applications such as, their use in controlling mobile robots [10, 11]. Furthermore, the advent of neural networks has led to the extension of these to functional approximations from tabula-rasa methods, making the control highly non-linear dynamical systems possible. Notably, Deepmind's recent success with training a robot to gently manipulate objects [12], would not be possible to reproduce using classical or modern control techniques due to dimensional problems.

1.4 AlphaZero

AlphaGo Zero is a revolutionary Reinforcement Learning algorithm that achieved superhuman performance in the game of go, winning 100–0 against the previously published, champion-defeating AlphaGo. It's successor, AlphaZero, is a generalised version that can achieve superhuman performance in many games. There are two key sub-algorithms that form the basis of their success: Monte-Carlo tree search (MCTS) for policy improvement, and a deep CNN for the neural policy and value network. Policy iteration is then implemented through self-play. AlphaGo Zero and AlphaZero differ only in the latter's use of a single neural network that is updated iteratively, rather than evaluated against the previous one, and by not taking advantage of the symmetries of the games.

A key feature of AlphaZero is that it only requires the ability to simulate the environment. It does not need to be told how to win, nor does it need an exact model of the

system dynamics, $p(s_{t+1}|s_t, a_t)$, as this can be learnt through self-play. Furthermore, the algorithm often "discovers" novel solutions to problems, as shown by *move 37* in a game of Go against the reigning world champion, Lee Sedol. This makes it particularly suitable for learning to control complex dynamical systems where approximate simulations can be made.

1.4.1 Self Play and The Neural Network

Figure 1.1a shows how self-play is performed in AlphaZero. A game, known from now on as an episode, $\{s_1, ..., s_T\}$ is played and for each state, s_t, a Monte-Carlo Tree Search is performed, guided by the current policy f_θ. The MCTS outputs an improved action probability mass function ($p.m.f$), $\boldsymbol{\pi}_t = [p(a_1|s_t), p(a_2|s_t), ..., p(a_N|s_t)]$. The next move is then selected by sampling from $\boldsymbol{\pi}_t$. The agent that plays the final move then gets scored (e.g in chess $z \in \{-1, 0, 1\}$ for a loss, draw or win respectively). The game score, z, is then appended to each state-action pair depending on who the current player was on that move to give training examples of the form $(s_t, \boldsymbol{\pi}_t, (-1)^{\mathbb{I}(winner)}z)$.

The neural network, $f_\theta(s)$ takes a board state, s, and outputs a $p.m.f$ over all actions and the expected outcome, $(\boldsymbol{p}_\theta, v)$ (fig. 1.1). The networks are initialised with $\theta \sim \mathcal{N}(0, \epsilon)$, where ϵ is small. The neural network is trained to more closely match the MCTS-informed action-$p.m.f$.s, $\boldsymbol{\pi}_t$, and the expected outcome (state-value), v_t to z.

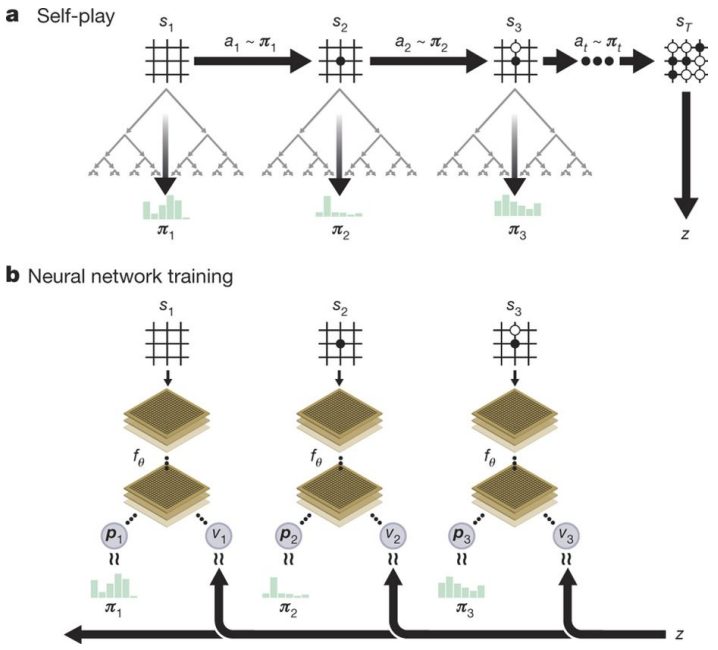

Figure 1.1: A schematic showing how self-play and policy training are performed. Taken from [13].

The loss function for the neural network is given by:

$$\mathcal{L} = (z - v)^2 - \boldsymbol{\pi} \cdot log(\boldsymbol{p}) + c||\theta||^2 \tag{1.8}$$

For chess, the input state consists of an image stack of 119 8x8 planes representing the board state at times $\{t, t-1, ..., t-8\}$, and planes representing repetitions, colours and castling etc. The output action $p.m.f$ is a 8x8x73=4672 vector representing every possible move from each piece, illegal moves are then masked. The output value, $v \in (-1, 1)$. The network itself consists of an input convolutional block and two separate "heads". The policy head has *softmax* activation function, pre-

ceded by series of *ReLU* linear layers and batch normalisation layers. The value head also has this but with a *tanh* output activation. The input convolutional block consists a single convolutional layer followed by 19 to 39 residual blocks (depending on the game).

1.4.2 Monte Carlo Tree Search

Figure 1.2 depicts the steps involved in a monte-carlo tree search (MCTS) iteration, and are described below.

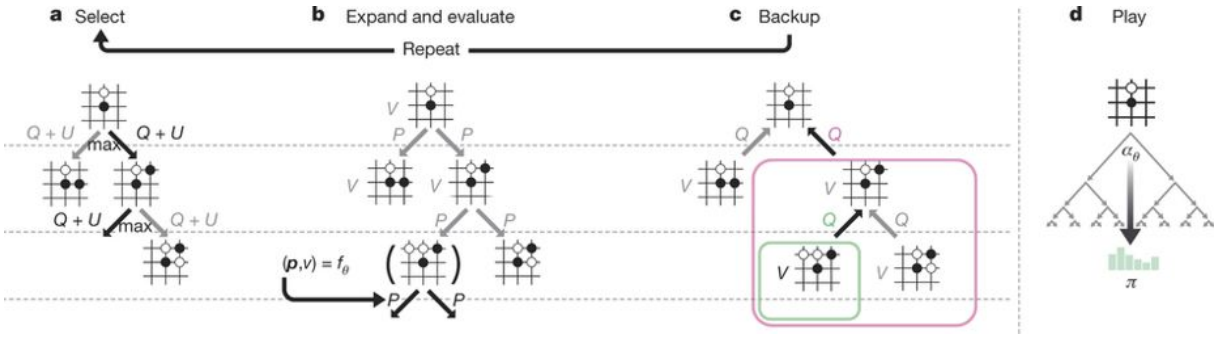

Figure 1.2: A schematic outlining the steps involved in a monte-carlo tree search. Taken from [13].

Figure 1.2a: A MCTS is performed at each step of an episode. The state at which the tree search starts then becomes the root state, s_{root}. From the root state, the tree search can move to an edge (s, a) by selecting an action, a. To ensure exploration, before selection a prior, $P(s,a) = (1-\epsilon)\boldsymbol{p}_{\theta,root} + \epsilon\boldsymbol{\eta}$, where $\boldsymbol{\eta} \sim Dir(0.3)$ and $\epsilon = 0.25$ is added. Each edge stores a prior action- *p.m.f.* and an initial value $\boldsymbol{p}_\theta(s_{t-1}, a_{t-1}), v(s_{t-1}, a_{t-1}) = f_\theta(s_t)$ [1]; a visit count, N(s, a); and an action-value, Q(s, a). Actions are selected by maximising an the action-value plus an upper confidence bound, which encourages exploration. The constant c (~ 1) can be increased to encourage exploration.

$$a_{selected} = \underset{\forall a}{\text{argmax}}\Big\{Q(s,a) + c \cdot \boldsymbol{p}_\theta(s,a)\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}\Big\} \tag{1.9}$$

Figure 1.2b: Once a leaf node (N(s, a) = 0) is reached, the neural network is evaluated at that state: $f_\theta(s) = (p(s,\cdot), v(s))$. The action *p.m.f.* and state-value are stored for the leaf state.

Figure 1.2c: The action-values, Q, are calculated as the mean of state-values in the subtree below that action. The state-value are then calculated as the mean of all the action-values branching from that state, and so on.

[1] note that action a_{t-1} from state s_{t-1} yields state s_t, if the system is deterministic. Therefore, $(s_{t-1}, a_{t-1}) = (s_t)$ in this setting.

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{s_{t+1}|s_t,a_t} v(s_{t+1}) \tag{1.10}$$

Figure 1.2d: After a set number of simulations (1600), the MCTS-improved action $p.m.f$s, $\boldsymbol{\pi} = p(\boldsymbol{a}|s_{root}) \propto N^{1/\tau}$, are returned where N is the visit count of each move from the root state and τ controls the sparseness of the probability mass function ($\{\tau = 0\} \rightarrow$ argmax$\{N(s,a)\}$). For self-play, $\tau = 1$ for the first 30 moves and $\tau \rightarrow 0$ thereafter. For evaluation $\tau \rightarrow 0 \; \forall t$. If $\tau \rightarrow 0$ then $\boldsymbol{\pi}$ becomes one-hot and the loss function of the neural network makes sense as a prediction of a categorical distribution.

1.4.3 Policy Iteration

AlphaZero uses a single neural network that continually updates, irrespective of whether it is better or worse than the previous network. Whereas AlphaGo Zero games are generated using the best player from all previous iterations and then only replaced if the new player wins $> 50\%$ of evaluation games played.

Evaluation is done by playing 400 or more greedy ($\tau \rightarrow 0$) games of the current best neural network against the challenging neural network. The networks are then ranked based on an elo scoring system.

1.5 Ranking and Elo Rating

The Elo rating system is a method of finding players' relative skills in two player games such as chess [16]. The Elo system is based upon every player having an average skill level μ_i, and variation distributed normally about their mean , i.e. $skill \sim \mathcal{N}(skill|\mu_i, \sigma)$. Therefore, the probability of one player beating the other is given by the cumulative density function (*c.d.f.*) of the difference of their Gaussians. This can be approximated as a logistic function, and written as:

$$\mathbb{E}[s_{p1}] = \frac{1}{1 + 10^{(\mu_{p2} - \mu_{p1})/400}} \tag{1.11}$$

Where 400 is a scaling factor such that the means can be used as the player rating, and s is the score. The rule for then updating each player's rating is given by:

$$\mu^{(new)} = \mu^{(old)} + 32(Score - \mathbb{E}[s]) \tag{1.12}$$

where 32 is again an arbitrary constant. New players are given an initial ranking of 1000.

1.6 Summary of Potential Benefits

The application of AlphaZero to dynamical systems a number of potential benefits above traditional optimal control or reinforcement learning. These include:

Robust Disturbance Handling. The Adversary in AlphaZero is effectively worst case scenario disturbance modelling. By incorporating this into the training process, the controller should be able to cope with situations well outside normal operating conditions.

Non-Linear Systems. Neural networks are universal function approximators and, hence with the correct architecture, therefore should be able to model any system.

General Framework. AlphaZero is a general algorithm that should be able to be applied to many different systems with minimal changes and still model the control well.

This project investigates these.

2 References

[1] M.C. Smith, I Lestas, *4F2: Robust and Non-Linear Control* Cambridge University Engineering Department, 2019

[2] G. Vinnicombe, K. Glover, F. Forni, *4F3: Optimal and Predictive Control* Cambridge University Engineering Department, 2019

[3] S. Singh, *4F7: Statistical Signal Analysis* Cambridge University Engineering Department, 2019

[4] Arthur E. Bryson Jr, *Optimal Control - 1950 to 1985*. IEEE Control Systems, 0272-1708/95 pg.26-33, 1996.

[5] I. Michael Ross, Ronald J. Proulx, and Mark Karpenko, *Unscented Optimal Control for Space Flight*. ISSFD S12-5, 2014.

[6] Zheng Jie Wang, Shijun Guo, Wei Li, *Modeling,Simulation and Optimal Control for an Aircraft of Aileron-less Folding Wing* WSEAS TRANSACTIONS on SYSTEMS and CONTROL, ISSN: 1991-8763, 10:3, 2008

[7] Giovanni Binet, Rainer Krenn and Alberto Bemporad, *Model Predictive Control Applications for Planetary Rovers*. imtlucca, 2012.

[8] Russ Tedrake, *Underactuated Robotics*. MIT OpenCourseWare, Ch.3, Spring 2009.

[9] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction (2nd Edition)*. The MIT Press, Cambridge, Massachusetts, London, England. 2018.

[10] I. Carlucho, M. De Paula, S. Villar, G. Acosta . *Incremental Q-learning strategy for adaptive PID control of mobile robots*. Expert Systems with Applications. 80. 10.1016, 2017

[11] Yuxi Li, *Deep Reinforcement Learning: An Overview*. CoRR, abs/1810.06339, 2018.

[12] Sandy H. Huang, Martina Zambelli, Jackie Kay, Murilo F. Martins, Yuval Tassa, Patrick M. Pilarski, Raia Hadsell, *Learning Gentle Object Manipulation with Curiosity-Driven Deep Reinforcement Learning*. arXiv 2019.

[13] David Silver, Julian Schrittwieser, Karen Simonyan et al, *Mastering the game of Go without human knowledge*. Nature, vol. 550, pg.354–359, 2017.

[14] David Silver, Thomas Hubert, Julian Schrittwieser et al, *A general reinforcement learning algorithm that masters chess, shogi and Go through self-pla*. Science 362:6419, pg.1140-1144, 2018.

[15] S. Thakoor, S. Nair and M. Jhunjhunwala, *Learning to Play Othello Without Human Knowledge* Stanford University Press, 2018 `https://github.com/suragnair/alpha-zero-general`

[16] HowlingPixel.com, *Elo Rating System*. `https://howlingpixel.com/i-en/Elo_rating_system` acc: 11/03/2019. published 2019