Designing an Artificial Intellingence for a Complex Card Game

Interim Report

psyjc5 Joel Cox

COMP4029

I hereby declare that this dissertation is all my own work, except as indicated in the text.

Table of Contents

Introduction

This project is about developing an Artificial Intelligence algorithm to learn to play a complex card game, Magic: The Gathering (MtG). MtG is a popular trading card game that pits two players against each other in the roles of duelling wizards. The players are able to summon creatures and cast spells in an attempt to reduce the opposing players life total down to 0. Cards are used to represent these spells in the game, and players can make their own choice of which cards to include in their decks.

Forge[5] is an open source project to provide a digital environment to play MtG. The goal of the project is to develop an AI that can be released for use alongside Forge in place of its current AI, which is lacking in certain areas. In this way, the project is for the Forge community and user-base.

There is currently a lack of competitive AI opponents for MtG players, and whilst the game is primarily played between two human players, it would be useful for players to have an AI opponent available to train against and benchmark themselves against. As well as this, it would be interesting to know if an AI will be able to beat professional MtG players. Whilst this project will likely not be able to achieve such a feat, I would hope that it proves an AI algorithm capable of presenting a challenge to at least more experienced players.

This report will outline the background and methodologies used to develop the project, as well as detailing design choices that impacted the project implementation.

Aim

The aim of this project is to provide an AI that can learn to play Magic: The Gathering such that it provides a fun, challenging, and valuable opponent for human players.

Objectives

1. To produce an AI capable of playing a game of Magic: The Gathering within the Forge client.
2. The AI should be able to assess the current game state, and evaluate possible moves based upon this assessment.
3. The AI should choose what it considers the most favourable move from this evaluation.
4. The AI should be able to "learn" from playing games, and become gradually better performing.

Background and Motivation

There has been a recent surge in studies looking at developing learning AIs for games, both digital and physical. Some good examples of this are AlphaGo[1] for the board game Go, as well as Deep Blue[2] for Chess; an example of a digital game AI is OpenAI Five[3] for the video game Dota 2. These AI are developed with different pursuits in mind, but most look into training AI for real world application, using game contexts as substitutes due to their complex states and decision making requirements.

Magic: The Gathering (MtG) is a strategic card game between two players, who take the roles of duelling wizards, using a variety of spells represented by cards to reduce the other's life total to

zero. Unlike traditional card games, MtG uses custom made cards in place of traditional playing cards. The game also makes use of imperfect information, in a similar vein to Bridge, where the exact contents of an opponent's hand are usually unknown. Each player uses a custom deck of cards made up from a selection of thousands of cards, which makes it hard to guess a new opponent's cards until they are played or otherwise seen.

I find MtG an interesting game due to its variety in cards, and the variety of play-styles this wide selection provides, with many cards altering how the game is played significantly. This variety also makes it an interesting case for developing AI to play MtG, as a set strategy will not always provide similar results against opposing strategies. Another facet to MtG is due to its imperfect information, there is commonly no computable optimal move, as unknown cards might make a usually optimal move sub-optimal, and vice versa.

Recent studies have shown that MtG is Turing Complete[4]; this is not directly relevant to developing an AI for the game, but it does mean that MtG is more computationally complex[4] than Chess and Go, and speculated to be the most computationally complex game in literature[4]. The cards required to induce a Turing machine state in the game will likely not be used in this project, due to complexity issues, as well as being an unrealistic scenario. However, the ability to prove MtG is Turing Complete shows the robustness of the game's logic.

This is the inspiration for this project: Is it possible to write a learning AI for MtG? The game contains complex states, as well as decision making around unknown variables, leading to a challenging environment for traditional AI. There has been research into using Monte Carlo Tree Search for card selection in order to play MtG[7], but in this example they severely limit the game's complexity in order to simplify the problem. I hope to be able to develop an AI that can play with fewer limits, although some complexity and card variety – though not as much as [7] – will still be sacrificed in order to keep development time within reason.

A lot of game AIs use various types of decision trees in combination with neural networks including AlphaGo, Deep Blue, and others[6][8]. This is the approach I will likely be taking, using tree search to determine what neural network is appropriate for the current game state. The most challenging aspect of this project would be defining appropriate inputs and structure for the neural network(s).

Description of the Work

The end result of the project should be a functioning AI algorithm that can be selected within the open source Forge client, and provide a challenging game experience to the player. It should be able to assess the current game state and choose an action in game to its benefit, and should be capable of making decisions that lead to a victory state. It should also be able to learn from playing games, reinforcing good actions and decisions, so that it can perform better in future games.

So far I have looked at possible methodologies to use, and evaluated which can be used effectively to achieve the project goal. I have designed a structure which will be detailed below. This is the result of considering several options and choosing the features that I considered the most effective based on other research.

Related Work and Methodologies

There have been numerous studies into teaching AI to play traditional board games, AlphaGo[1] and Deep Blue[2] being more successful examples of this. There have also been studies specific to writing an AI to play MtG[7], but these usually involved heavy restrictions on what could be played, severely limiting the game's complexity. This project aims to create an AI that can play within a less restricted version of the game.

The company behind MtG have several of their own digital clients for playing the game, the two most notable being MtG Online[11] and MtG Arena[12], however developing for these platforms is not a plausible route. MtG Online does not currently have an AI built-in, and so play is entirely between human players. MtG Arena has primitive AI for teaching the game to new players, but not a competitive AI for playing against, and so gameplay is again mainly between human players. Both games feature anti-cheat measures, which prevent "botting", or automating inputs, and so developing an AI for these platforms is not plausible. However, the lack of in-depth AI in both of these games indicate that this is a hard problem which the game developers believe to be too difficult to be worth implementing.

Two alternative open-source digital clients include Cockatrice[13] and Xmage[14]. Cockatrice lacks a rules engine for game, and relies on the players communicating their intentions correctly. Thus an AI algorithm would not be able to know what to do, as all actions must be performed manually. Xmage is a viable alternative to Forge, being both open source and having a full rules engine. However, Xmage is used more often for online games between human players, whereas Forge is strictly against AI opponents, so it makes more sense to develop for the latter.

Forge[5] is an open source project to provide a digital environment to play MtG. It currently features a basic AI with a function set to get game information. This AI is not that proficient, nor complex, and can be frustrating or dull to play against if a player is sufficiently skilled. An AI that is more challenging would be more engaging to advanced players, and is something I think worth developing for this reason, and this is what I've decided to do for my project.

Tensorflow[10] is a python library written to aid in the creation and use of neural networks. I will be using this library as writing my own neural network framework would be cumbersome and difficult. Tensorflow also makes use of Nvidia CUDA technology, which can utilise GPU processing power to speed up the machine learning process. Tensorflow is used for a wide variety of machine learning implementations, namely image classification and reinforcement learning algorithms.

Methodology

There are many methods used within AI research, and a wide spread of them are present in game playing research. Here I will outline some of the possible technologies usable for the project, and discuss why they might be plausible options.

Convolutional Networks

Convolutional networks using an operation called convolution that takes parts of an image as groups and performs some operator function – the convolution – to them. This outputs either a value or set of values that can be used to identify and classify features within an image.

For some games, such as Go[6], or Chess, convolutional networks can be used to recognise board patterns and compare these to standard board states, which the AI can use to make a decision. This

image/pattern recognition approach can work well for board games that have a rigid board structure, e.g. a square grid, as patterns are likely to appear and repeat themselves between games. As well as this, the position and orientation is meaningful in these games, and so this provides more focused patterns or board states to be recognised.

A good example of using convolutions is in [6], where they detail the way that common Go play patterns impacted the shapes of convolutions. They attempt to combat “Ladders” by using diagonal convolutions to recognise if a Ladder is escapable or not, by looking for an additional piece that can break the Ladder.

MtG does not have a rigid board structure; it does not matter what order cards are positioned on the table, nor in the hand. This means that an image based algorithm would not suit the AI, as it would not be able to read any meaningful data from the images.

Branching Decision Trees

Decision trees are used frequently[1][8] to determine what state a game is in, so that an AI can make choices specific to that scenario. This helps in games where actions can change in priority as the game progresses; in Real Time Strategy (RTS) video games, it is common to have resource gathering phases before combat phases, and so an AI would need to be able to determine if it should aim for increasing resources or using them to fight.

An example of the difficulties of using Tree Search algorithms, such as Monte Carlo tree search, can be see in a study on writing an AI for the RTS game Starcraft[15], where the amount of complexity within the game is shown. Monte Carlo tree search is also used by the AlphaGo[1] study, where they iterate through multiple solutions using a function to maximise an “action value”, and requires a huge 48 CPUs and 8 GPUs to process 40 search threads. This kind of resource is not available for this project, so tree search is not a practical option.

For MtG, decision trees seems like a sensible idea: the game features a resource system, which can be used to create stronger combat units. The game also plays differently depending on a player’s strategy, and so identifying that strategy could be important to maximising success. An issue with decision trees, though, is that they can be slow to iterate through possible game states. This is because with each additional possible game state, the number of possible paths increases exponentially, and the processing power required is too large, as evidenced by the AlphaGo requirements above. This issue is particularly pertinent within MtG, as the board states can vary greatly between games, and strategies can appear very similar to start, and only become apparent as different several turns into the game.

Neural Networks

Neural networks are very common within AI development[1][2][6], and can be used in conjunction with other techniques, such as convolutional matrices, to either classify or provide outputs for a given set of inputs. These networks are good at being trained for a particular task, through reinforcement learning, where the network attempts to perform its purpose, e.g. classification, and by comparing its answers to training data of correct answers, can self-correct its decision variables to improve itself on the next run. This self-correction algorithm is known as back-propagation[16].

Back-propagation takes the current weights in the neural network, and reinforces the weights in order to align the output value(s) with the training data target values. This weight change usually

effects a gradient descent approach, in order to encourage gradual change to prevent outliers from breaking a model.

For this project, a neural network could be useful, as reinforcement learning is a desired part of the final AI, and a neural network is an efficient way to do this, as the reinforcement can be automated with sufficient training data. However, neural networks require specific inputs, and so deciding what inputs these should be can be difficult. In the case of MtG, it is not especially apparent what all the inputs should be, due to the large quantity of possible inputs available. This includes player information, such as life totals or cards in hand, as well as information about certain cards, properties of cards on the board, and cards in decks. Inputting all of this information into a network might result in a network that is difficult to train, and not very good at playing or learning to play. To ensure that I understand the network and don't over convolute the problem, I have chosen to select certain inputs, instead of inputting all possible inputs.

Design

The current design as is follows: a card selection AI, that uses attributes about cards in hand, as well as limited board state information, to choose which card to play when available.

This will be achieved by writing a neural network in python using the tensorflow library. This network will take the inputs from the Forge client, and return a value to the client mid-game. The program will also save the scores and inputs matching those scores to potentially be used as training data in the future. The forge client also would make use of two evaluative functions to provide additional inputs to the neural network.

The Forge client is very complex, and features a wide set of classes to deal with the MtG rules engine. Fortunately the code base is well written and thus working out an injection point to put my project's behaviour was made easier. Part of the reason the project is designed around a card score based selection is down to how the Forge client currently has AI for its games.

Decision Inputs

After consideration of different ways to read the game state, I decided that a card selection method would be the best way forward. This is because ultimately, actions are performed by playing cards, and so rating which card is best to play is the most sensible way to choose an action. In order to determine which card to play, certain inputs are needed. These include what kind of card each card is, as well as attributes about the card, like what effects it has when played, or if it will have effects later in the game, e.g. a creature card can provide benefits to a player over multiple turns, until it dies. This card selection AI will be a neural network, take inputs that describe a card, as well as limited board state information, and output a score for the card. This score is generated based on how effective the card would be to play in the current moment, with higher scores reflecting a better play. Since training data is unavailable for the initial tries, any successful games will have their scores saved as good data, and lost games will have a chance to have their scores saved as well, in case good decisions were made despite losing the game.

In order to input a card's attributes, I will have an algorithm that uses the Forge client's card database API, which contains all the relevant attributes and information, and saves it into a state that can be input into the neural network. The algorithm will give a value between 0 and 1 for each

possible attribute, and this will reflect what qualities or effects a card has, and how much value each attribute is to the card as a whole.

Another important piece of information is whether or not a player is on the offensive or defensive. Certain strategies lend themselves better to playing offensively, and so certain cards will be better in an offensive position. To determine this, I will write an algorithm that reads the game state values such as life totals, and identify how much of an offensive position the AI player is in. This will then feed into the neural network and contribute to scoring each card.

Implementation

The implementation will be written in part in java, and in part in python. The reasoning behind this is because the Forge client is written in java, and so writing the AI in java is practically required. The neural network part will be written in python using the tensorflow[10] library. This is because I do not have the knowledge or time to effectively write my own neural network structure for this project. To interface between the Forge client and the neural network I will use the tensorflow java library.

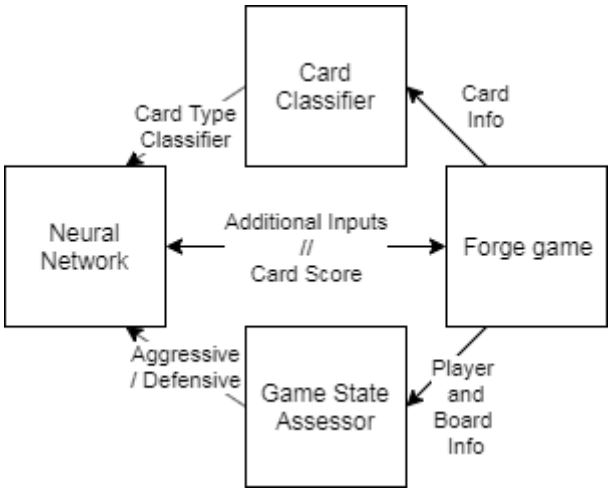

Figure 1: The information flow of the project. Data is moved from the Forge client to the evaluative functions, which return a card score.

I have so far been using Git to manage the development between two machines, only one of which is capable of using the Nvidia CUDA library to perform machine learning tasks quickly. The other machine is a laptop and so being able to develop on either is important to maintain smooth development.

Progress

So far I am on track, having spent most of the project time on investigating possible solutions for the design. I have started to implement the functions that provide inputs to the neural network, and I have setup the environment required to run the neural network and train it.

Overall progress feels slow, due to experimenting with the design of the project. The initial design of the AI used multiple neural networks, for classifying card types as well as determining the board state and the player's position in the game. However, these networks both suffered the same issue: there was no training data, and generating data for these networks would be inefficient, and so

replacing them with direct evaluative functions achieves the same goal in a less convoluted way. These functions will have drawbacks of using hard coded evaluation functions, rather than ones that can progress with the scoring network, but this drawback is mitigated in part by the ability of the network to self-balance inputs and their weights, effectively changing the inputs from these functions to be as valued as needed to perform well.

I struggled to find a good way to integrate the AI into the existing Forge client. The code base is very large, and was confusing to navigate initially. After some time examining the code I found an appropriate place to insert the card scoring AI.

I have not yet made much progress on the evaluative functions, which I plan to implement over the coming months. Once these simpler functions are complete and providing outputs, I will focus on the neural network part of the project, which will take up the majority of the time after the new year.

Task	Start	End	Duration (days)
Investigate AI within Games	10/31/19	11/20/19	20
Examine Forge Client Structure	10/31/19	11/10/19	10
Forge Client Entry Points	11/10/19	11/20/19	10
Design AI structure	11/20/19	12/04/19	14
Design Inital Classifier Function	12/04/19	12/17/19	13
Interim Report	12/04/19	12/18/19	14
Test Classifier And Modify	12/17/19	12/27/19	10
Implement Initial NN Solution (card score	12/27/19	01/09/20	13
Gather Test Data from ForgeAI	12/27/19	01/03/20	7

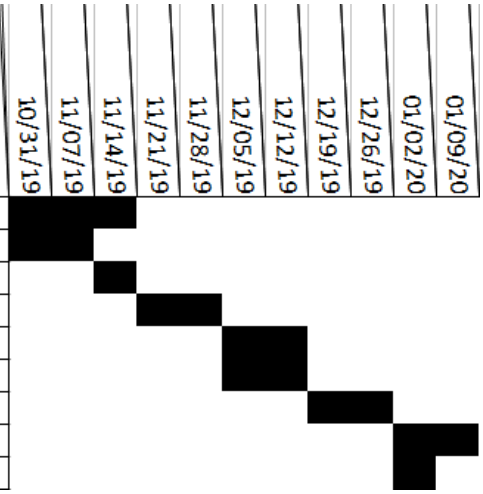

Figure 2: Projected Gantt Chart plan excerpt

Looking at the Gantt chart which I planned to follow, I can say that I am still on track. However, whether the initial plan was a good plan remains to be seen, as I would have preferred to have more developed code by this point. I expect the development to follow this plan as written until the beginning of the development of the neural network, which I expect to raise difficulties that I am not aware of, due to an unfamiliarity with the tensorflow library.

If I could change my approach, I would dedicate more time at the beginning to practice using the tensorflow library, as well as starting on the documentation, such as this report, earlier, as it would provide starting points to write about, and make the workload later on easier to manage.

Bibliography

[1] Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. and Hassabis, D. (2016). Mastering the game of Go with deep Neural Networks and tree search. *Nature*, 529(7587), pp.484-489.

[2] Campbell, M., Hoane, A. and Hsu, F. (2002). Deep Blue. *Artificial Intelligence*, 134(1-2), pp.57-83.

[3] OpenAI. (2019). *OpenAI Five*. [online] Available at: https://openai.com/blog/openai-five/ [Accessed 9 Oct. 2019].

[4] Churchill, A., Biderman, S. and Herrick, A. (2019). Magic the Gathering is Turing Complete. [online] Available at: https://arxiv.org/abs/1904.09828 [Accessed 9 Oct. 2019].

[5] Slightlymagic.net. (2019). *Forge - Slightly Magic*. [online] Available at: https://www.slightlymagic.net/wiki/Forge [Accessed 9 Oct. 2019].

[6] Barratt, J. and Pan, C. (2017). Playing Go without Game Tree Search Using Convolutional Neural Networks. [online] Available at: http://cs231n.stanford.edu/reports/2017/pdfs/603.pdf [Accessed 16 Oct. 2019].

[7] Ward, C. and Cowling, P. (2009). Monte Carlo Search Applied to Card Selection in Magic: The Gathering. [online] Available at: https://web.archive.org/web/20160528074031/http://scim.brad.ac.uk/staff/pdf/picowlin/CIG2009.pdf [Accessed 16 Oct. 2019].

[8] Silver, D. and Huang, A. (2012). Mastering the Game of Go with Deep Neural Networks and Tree Search. [online] Available at: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Publications_files/unformatted_final_mastering_go.pdf [Accessed 16 Oct. 2019].

[9] Flores. M 1999, "Who's the Beatdown?" https://web.archive.org/web/20190329100828/http://starcitygames.com/magic/fundamentals/3692_Whos_The_Beatdown.html [Accessed 1 December]

[10] tensorFlow https://www.tensorflow.org/ [Accessed 1 December]

[11] Magic: The Gathering Online https://magic.wizards.com/en/mtgo [Accessed 4 December]

[12] Magic: The Gathering Arena https://magic.wizards.com/en/mtgarena [Accessed 4 December]

[13] Cockatrice https://cockatrice.github.io/ [Accessed 4 December]

[14] Xmage http://xmage.de/ [Accessed 4 December]

[15] Uriarte, A. and Ontan˜on´, S. (2019). Improving Monte Carlo Tree Search Policies in StarCraft via Probabilistic Models Learned from Replay Data. [online] Available at: https://www.aaai.org/ocs/index.php/AIIDE/AIIDE16/paper/viewFile/13984/13601 [Accessed 4 Dec. 2019].

[16] Rumelhart, D., Hinton, G. and Williams, R. (2019). *Learning representations by back-propagating errors*.

[17] Nvidia CUDA https://developer.nvidia.com/cuda-zone [Accessed 5 December]