Jordi Llatser Torres

EXPLORING THE APPLICATION OF NEURAL NETWORKS IN COMPLEX CARD GAMES

COMPUTER ENGINEERING FINAL DEGREE PROJECT

Directed by:

Dra. Aïda Valls Mateu
Dr. José Rafael Escorcia Gutiérrez

Doble Grau d'Enginyeria Informàtica i Biotecnologia

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2023

Resum.

L'objectiu d'aquest treball ha estat desenvolupar una eina basada en Intel·ligència Artificial que aprengui a fer decisions en un joc de cartes tan complex com el Magic: The Gathering. El treball està centrat en una part del joc on s'ha de triar una carta entre un conjunt de màxim de 15 d'un set de aproximadament 300 cartes, tenint en compte les seleccions que s'hagin fet prèviament. S’han utilitzat mètodes d’aprenentatge automàtic basats en xarxes neuronals, aprofitant les dades de jugadors reals fent aquestes mateixes seleccions. S’ha dissenyat un model de representació de les dades del joc i s’ha implementat una xarxa neuronal usant el llenguatge de programació Python.

En aquest projecte s’ha aconseguit entrenar una intel·ligència artificial en la tasca prèviament definida, aconseguint imitar la selecció de cartes escollides pels jugadors usant un conjunt de dades de testeig a partir de jugades conegudes. El sistema ha estat també avaluat per alguns jugadors, obtenint uns bons resultats qualitatius.

Resumen.

El objetivo de este trabajo ha sido desarrollar una herramienta basada en Inteligencia Artificial que aprendiese a tomar decisiones en un juego de cartas tan complejo como el Magic: The Gathering. El trabajo está centrado en una parte del juego donde se debe elegir una carta entre un conjunto de máximo de 15 de un set de aproximadamente 300 cartas, teniendo en cuenta las selecciones que se hayan realizado previamente. Se han usado métodos de aprendizaje automático basados en redes neuronales, aprovechando los datos de jugadores reales haciendo estas mismas selecciones. Se ha diseñado un modelo de representación de los datos del juego y se ha implementado una red neuronal usando el lenguaje de programación Python.

En este proyecto se ha conseguido entrenar una inteligencia artificial en la tarea previamente definida, consiguiendo imitar la selección de cartas elegidas por jugadores usando un conjunto de datos de testeo a partir de jugadas conocidas. El sistema también ha sido evaluado por algunos jugadores, obteniendo buenos resultados cualitativos.

Abstract.

The aim of this work has been to develop a tool based on Artificial Intelligence to learn how to make decisions in a card game as complex as Magic: The Gathering. The project is focused on a part of the game where a card must be chosen from a set of up to 15 cards from the global game set of about 300 cards, considering the selections that have been previously made. Machine learning methods based on neural networks have been used, taking advantage of data from real players making these same selections. A game data representation model has been designed and a neural network has been implemented using the Python programming language.

In this project we have managed to train an artificial intelligence in the previously defined task, managing to imitate the card selection made by players using a testing dataset from known plays. The system has also been evaluated by some players, obtaining good qualitative results.

Index

Table Index

Index de figures

1 Introduction

Artificial Intelligence (AI) is a computer science field that recently has earned a lot of popularity with the recent boom of technologies using it. From chatbots that simulate human speech with vast knowledge about everything, to self-driving cars, AI is evolving and becoming a part of our daily lives.

AI can be defined as the discipline that builds computer programs that perform tasks that would normally need human intelligence. In the early days of AI research, these systems were very limited, but with the advance in technology and data availability, new machine learning algorithms have evolved AI to a higher level.

One task that requires deep knowledge and large training is the playing of certain card games. In this bachelor project, we have explored the possibilities of training an AI system for one of the most complex card games that exists nowadays: Magic: The Gathering.

Magic: The Gathering (also called Magic or MTG) is a tabletop and digital collective card game created in 1993, where players build decks and use those decks to play versus each other. It is a multiplayer game that has been evolving through the years, with different formats and ways to play it, and with more than 25000 unique cards, it has become the most popular trading card game.

There are multiple ways to play Magic: The Gathering (called, formats [1]). There are many formats, but they can be classified in 2 ways: Constructed and Limited. Constructed formats are the formats where players play with decks previously constructed (hence the name). The specific format changes the card pool allowed, but in any Constructed tournament, players build the deck in advance of the tournament. Limited formats, on the other hand, players need to build decks after opening a specific number of Magic product, usually unopened packs of cards, also called boosters, and then build a deck with that limited pool of cards. Limited formats bring an extra gameplay experience, as it is not just playing with a deck versus an opponent, there is also the card selection and deck building problem.

The most popular limited format is the one called booster draft [2], or just draft. To play this format, players select cards from boosters to then build a deck to play with. These boosters are packs of 15 cards, and these cards are usually from the same set, and a set is the general name of the group of cards that are released as a new product. A set usually consists of 250-300 cards, that can be new or can be old cards that get reprinted as new versions.

To be more specific, a player who wants to draft first joins a pod (a group of usually 8 players that draft together. Then each player gets 3 booster packs, and starts the drafting loop [3]:

1. Open a pack.
2. Select a card from the pack and add it to your pool of cards.
3. If there are no cards left, go to step 4, otherwise to step 5.
4. If one of your 3 packs is still closed, go back to step 1, otherwise the draft has ended.
5. Pass the cards to the player next to you.
6. Get the cards passed to you from the other player next to you and go back to step 2.

After finishing these steps, the player has a pool of cards that have to be used to build a deck to play versus the other players. And while these steps are quite easy to understand, they are more complex than it seems. Step n2 is the most crucial and complex here, the card selection process. A player must evaluate the cards in the pack, that are a subset of between 1-15 cards of the set he is playing, and choose the best card based on his criteria, taking in consideration what the cards do, and how well they work with the cards previously selected that are already in the pool. Learning good criteria and making the best decisions is hard, mostly because you never know the information of your next possible picks, plus the criteria can change between sets, and even more importantly, there is not always a best decision, sometimes there are multiple options that can be correct. Players usually study the set to evaluate how good each card is and follow basic strategies like B.R.E.A.D (Bombs, Removal, Evasion, Aggro, Dirt), common strategies developed as basic knowledge for draft players. But in the end, most players end up doing trial and error to see which cards are better, or which ones work best together, to tune their strategy for the current set.

This project is focused on developing an Artificial Intelligence that will learn and will be used on the card selection process of a draft game.

As a fellow Magic: The Gathering player, I know it is a hard game to learn from zero, therefore, creating a tool that could help new players in the card selection process would be very useful to increase their engagement and enjoyment during the play.

1.1. Objectives

From the idea of designing and constructing a program based on Artificial Intelligence to learn to play Magic: The Gathering, we decided to focus the work on one of the difficult tasks of this game: the draft card selection. Then, the objectives of this work are the following:

1. Study the state of the art in AI-based methods for card selection in MTG.
2. Design a training system with Machine Learning techniques that can learn from thousands of real selections made by people.
3. Implement the system using state-of-the-art Machine Learning technologies for the management of large datasets.
4. Evaluate the system, so that the model obtained must be accurate enough to help real players make better decisions about the game.

1.2 Structure of this document

The rest of the document is organized as follows.

- Chapter 2 presents the main theoretical concepts about the techniques that will be used in this bachelor project, mainly about Machine Learning and Neural Networks.
- Chapter 3 is devoted to presenting projects where artificial intelligence and Magic: The Gathering are related, mainly about similar projects to my objective.
- Chapter 4 explains the data that will be used for this project.
- Chapter 5 discusses about the model that is built for this bachelor project.
- Chapter 6 presents the implementation of the project, and the final product.
- Chapter 7 shows the tests and results gotten with this project
- Chapter 8 talks about the conclusions reached with this project.

2 Terminology and definitions

This chapter is devoted to explaining all the terminology, tools and concepts related to the project.

2.1 Machine Learning

Machine learning [4] is a subfield of Artificial Intelligence that focuses on the development of algorithms and models that enable computers to learn and make predictions or decisions without being explicitly programmed. It involves the use of statistical techniques to enable systems to automatically learn from data, identify patterns, and make informed predictions or decisions. Machine learning algorithms are used in a wide variety of applications such as in medicine, computer vision, email filtering, where it is difficult to develop conventional algorithms to perform the tasks. Machine learning focuses on prediction based on properties previously learned from data.

There are three main approaches that correspond to the learning system used to train the computer programs, and those are supervised learning, unsupervised learning, and reinforcement learning.

2.1.1 Unsupervised Learning

Unsupervised learning is a type of machine learning where the algorithm is trained on an unlabeled dataset, meaning it is given input data without any corresponding output labels. The goal is for the algorithm to discover patterns or relationships within the data without any predefined notions of what it should find.

Unsupervised learning algorithms aim to group similar data points together or find underlying structures or patterns in the data. Common techniques in unsupervised learning include clustering algorithms, such as k-means clustering, and dimensionality reduction techniques, such as principal component analysis (PCA).

2.1.2 Reinforcement Learning

Reinforcement leaning [4] is a type of machine learning where a computer program interacts with a dynamic environment with a specific goal in mind. And while performing the task it gets a reward based on the performance and tries to maximize this reward.

The program learns based on this feedback in form of rewards or penalties, and the goal is to discover the optimal strategy that maximizes the reward score. A simple example of a use of reinforced learning is teaching an algorithm to learn to play the classic game snake, with rewards each time it grows and penalties each time it dies, with the objective of surviving until it fills the whole board, that wins the game.

2.1.3 Supervised Learning

Supervised learning is a type of machine learning where the algorithm is trained on a labeled dataset, meaning it is given input data with corresponding output data. The goal is to learn a set of “rules” that map inputs to outputs, to later apply it to unseen data.

In supervised learning, the dataset is divided into two main parts: the training set and the test set. The training set is used to train the algorithm, and after being trained, the algorithm is evaluated using the test set. These are the three main steps needed to get a successful machine learning algorithm using supervised learning: training, testing and validation.

Training is the part where the training set is given to the algorithm, and it uses the inputs and outputs given to adjust its internal parameters to minimize the error between the predicted output and the actual output.

Testing is the part where the testing set is given to the algorithm, and it uses the inputs to get a prediction on the output.

Validation is the part where the model is evaluated on its performance. It uses the outputs of the testing and compares them to the real outputs of the testing set to give a score. Some of the most usual ways to evaluate are the following:

- Accuracy: Accuracy measures the proportion of correct predictions made by a model out of the total number of predictions. It is a straightforward method and easy to understand and interpret, but it is not always the best decision, since the dataset could be imbalanced and the model could just predict 100% accuracy the main class while ignoring the minority, while still having a high accuracy.
- Precision: Precision measures the proportion of true positive predictions out of all positive predictions. It focuses on the correctness of positive predictions and is calculated as true positives divided by the sum of true positives and false positives. Precision is useful when the cost of false positives is high.
- Recall: Recall, also called Sensitivity or True Positive Rate, measures the proportion of true positive predictions out of all actual positive instances. It focuses on capturing all positive instances and is calculated as true positives divided by the sum of true positives and false negatives. Recall is important when the cost of false negatives is high.
- F-score: This score is the harmonic mean of precision and recall. It provides a balance between precision and recall and is useful when both false positives and false negatives need to be minimized.

2.2 Artificial Neural Networks

Artificial Neural Networks (ANN) are computational models inspired by the structure and functioning of biological neural networks in the human brain. They are composed of interconnected artificial neurons, also called nodes or units, organized in layers. ANNs are designed to learn from data and make predictions or decisions by adjusting the strength of connections (weights) between neurons.

Each neuron in an ANN receives input signals, applies a mathematical transformation to these inputs, and produces an output signal. The output of one neuron serves as input to the neurons in the subsequent layer until the final output is generated. During training, ANNs adjust the weights based on a learning algorithm, such as backpropagation, to minimize the difference between predicted and actual outputs.

2.3 Multi-Layer Perceptron Classifier

A perceptron is an artificial neuron, it is the basic unit of a neuronal net, and it can also be called node or neuron. The function of a perceptron is to do operations with the inputs to get an output that can be used to study characteristics. It usually does linear operations with the inputs, and it is used in multiples to build complex learning structures.

A Multi-Layer Perceptron is a common type of artificial neural network. It consists of a net of multiple perceptrons or nodes that are classified in layers (Figure 1). A specific node is connected to all the other nodes from the layers he has on both sides. It is a supervised learning algorithm that learns a function $f(\cdot) : R^m \rightarrow R^o$ where m is the number of dimensions for input and o is the number of dimensions for the output.

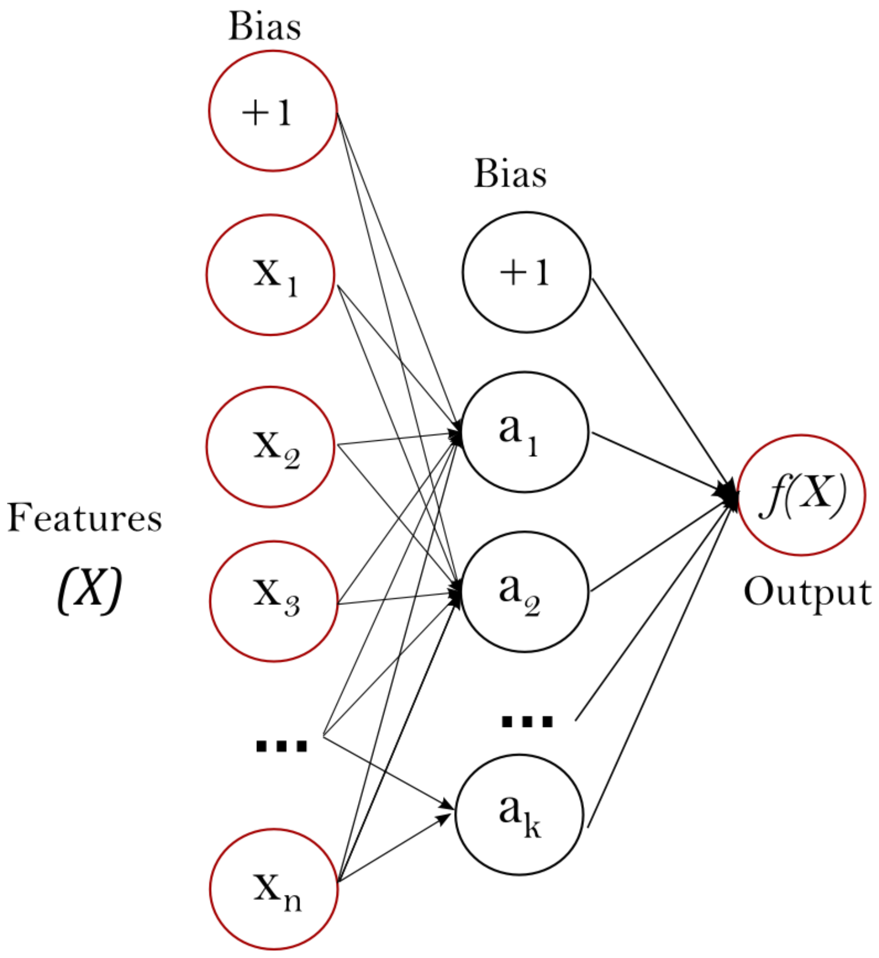


Figure 1. One Hidden Layer MLP (from sklearn)

The advantage of using a Multi-Layer Perceptron is the capability to learn non-linear models, and the capability to learn in real-time using *partial_fit*, useful for big databases

A MLPClassifier is a type of Multi-Layer Perceptron with a discrete output, composed by a predefined set of classes, for example, it can be spam or not spam, selected or not selected, cat, fish or dog. This is the kind of multi-layer perceptron we will use in this project.

2.3.1 Backpropagation

Backpropagation is the algorithm that will be used to train the MLPClassifier. This algorithm works by first predicting the outputs based on the input data X and comparing the predicted output with the actual output Y that is given to the function to train. When comparing this error, you get an error, the difference between the predicted and the actual output, and then you send back this error to each neuron in a backward direction, and then the gradient of the error is calculated with respect of each weight of the neurons, modifying the weights for the next iteration (Figure 2).

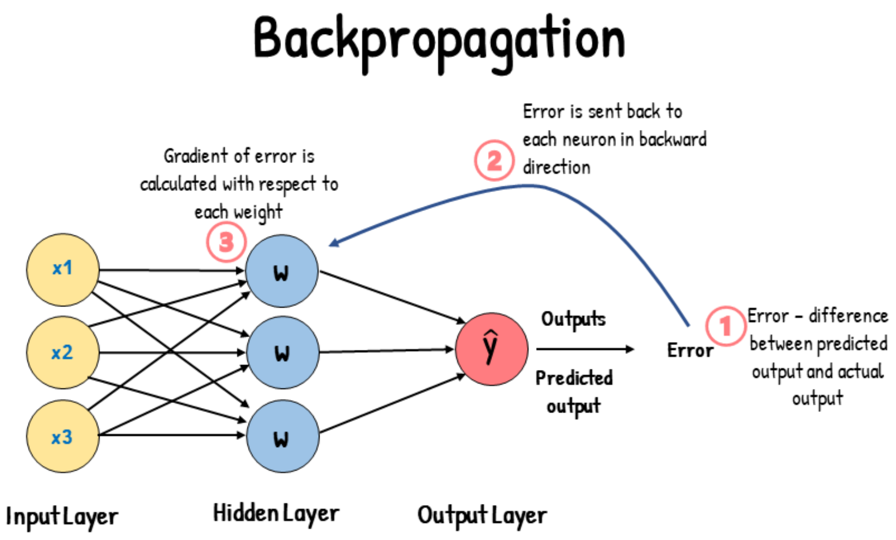

Backpropagation
Error is sent back to each neuron in backward direction
2
Gradient of error is calculated with respect to each weight
3
x1
x2
x3
w
w
w
ŷ
Outputs
Predicted output
Error
1
Error - difference between predicted output and actual output
Input Layer
Hidden Layer
Output Layer

Figure 2. Backpropagation (from analyticsvidhya.com)

2.4 Deep Learning

Deep Learning [5] is a subfield of machine learning that focuses on training artificial neural networks with multiple hidden layers. These networks, often referred to as Deep Neural Networks (DNNs), have the capability to learn complex representations and hierarchical features directly from raw input data, eliminating the need for manual feature engineering.

Deep Learning has brought significant advancements to various domains, including computer vision, natural language processing, speech recognition, and more. Deep Neural Networks excel at extracting high-level and abstract features from vast amounts of data, leading to state-of-the-art performance in numerous tasks.

One specific type of DNN is the multi-layer perceptron, which consists of multiple hidden layers. By incorporating multiple hidden layers, a Deep Neural Network has the capacity to discover intricate features and patterns in the input data, enabling it to capture more nuanced representations and enhance its learning capabilities. This depth allows DNNs to model highly complex relationships and achieve superior performance compared to shallow neural networks.

3 Related work

There have been many projects that used AI for different tasks related to MTG. Most of these projects are related to creating new cards. An example would be Urza’s AI system [6], or MTG Hivemind project [7]. This kind of projects study different properties of the cards such as the text, cost, color… and then generate new cards creating the card text, then assigning colors and costs related to the text, plus name, art, and other characteristics of the cards.

Outside these projects, there are not many that focus on the gameplay part of Magic: The Gathering. But there are two that provide different approaches to drafting.

The first paper [8] presents 5 different software agents that learn to draft, using different techniques to then evaluate them and comparing them. This article is not focused on artificial intelligence, and while it uses it in 2 models, the other 3 agents use other heuristics as randomly selecting cards, selecting cards based on the card characteristic called rarity, or by selecting based on a score each card has that was given by a human.

The second paper [9] uses AI and explains how it does. In this paper, 10 linear models are trained with data specific for each of the MTG archetypes, or color combinations played while drafting. With each model specified on a possible final deck, it can classify the cards on how good they are in each archetype. And with that it calculates a bias based on how much each card forces you to play a specific archetype and uses everything to select a card based on the picks the previous selected cards.

This last paper provides an interesting approach to how to tackle the problem, some ideas that could be used, but also give some interesting highlights about specific problems of that their approach. An interesting problem that this article exposes is what is called bad human behavior. This happens when people take an enticing card just to have it for future uses and not for the draft, making it a sub-optimal choice or even a bad one. This can also happen with other picks, where players can pick good cards but maybe not the optimal ones for that spot or be in a spot where multiple options are correct.

With these related works, we can see the current progress and uses of AI in Magic: The Gathering, plus some problems this project might face. This has helped in the design of the solution proposed in this project and in the evaluation.

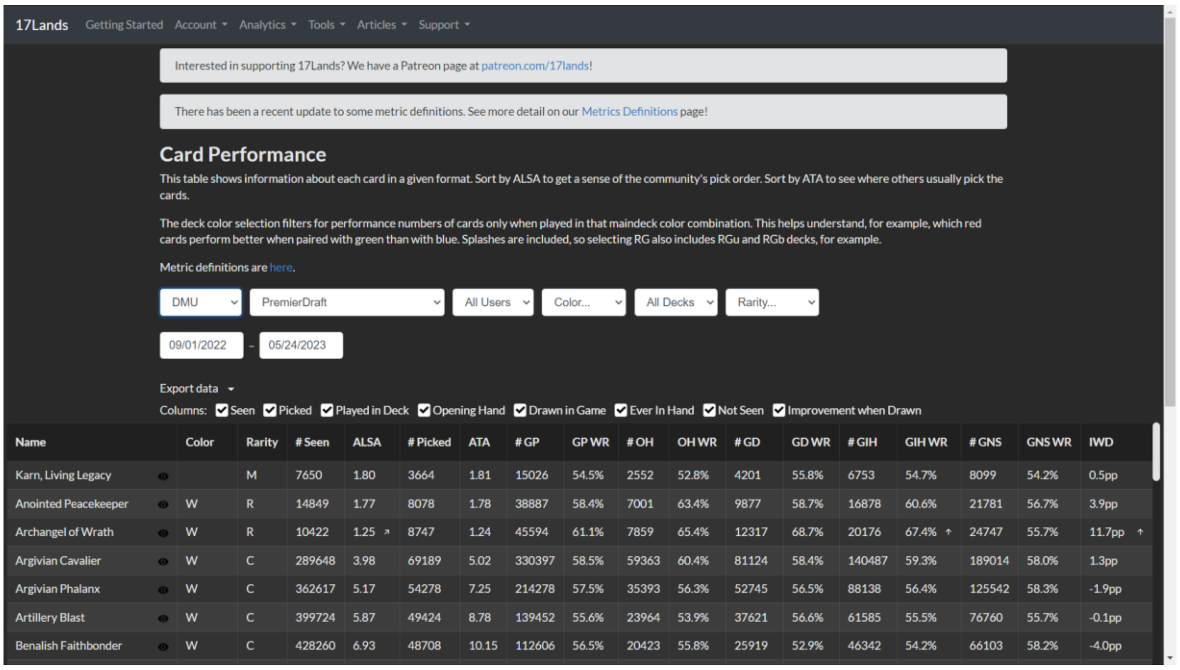
4 Data

In this chapter, the dataset used for constructing the drafting model is presented. As usual in AI systems, having a large set of data with examples of human behavior is required to train a system that mimics, in this case, the player behavior of drafting. A group of data analysts who love Magic: The Gathering have created a webpage to collect read drafting actions done by players. It is explained below.

4.1 17lands.com

17lands.com [10] is a webpage specialized in collecting data from limited formats of MTG (including draft, the one we are interested), and in their own words "The mission of 17Lands is to help the Magic: The Gathering community improve at limited. We aggregate and openly share data to help players understand their own performance, to generate insights that can only come from a larger communal pool of data, and to provide data-driven content."

17lands.com offers a tracker that gets data from the players who use it while playing in the app Magic: The Gathering Arena, the free official online platform to play. The tracker only works on computers, but gathers data of players drafts, game and replay data. This data is already analyzed to some extent in their webpage (Figure 3), but it is also free to download as datasets [11]. The datasets are already separated by expansion, format, and draft – game – replay. For training and testing reasons that will be explained later, we will use the data from Dominaria United Premier Draft.



Interested in supporting 17Lands? We have a Patreon page at patreon.com/17lands!

There has been a recent update to some metric definitions. See more detail on our Metrics Definitions page!

Card Performance

This table shows information about each card in a given format. Sort by ALSA to get a sense of the community's pick order. Sort by ATA to see where others usually pick the cards.

The deck color selection filters for performance numbers of cards only when played in that maindeck color combination. This helps understand, for example, which red cards perform better when paired with green than with blue. Splashes are included, so selecting RG also includes RGu and RGb decks, for example.

Metric definitions are here.

DMU | PremierDraft | All Users | Color... | All Decks | Rarity...

09/01/2022 – 05/24/2023

Export data

Columns: ✓ Seen ✓ Picked ✓ Played in Deck ✓ Opening Hand ✓ Drawn in Game ✓ Ever In Hand ✓ Not Seen ✓ Improvement when Drawn

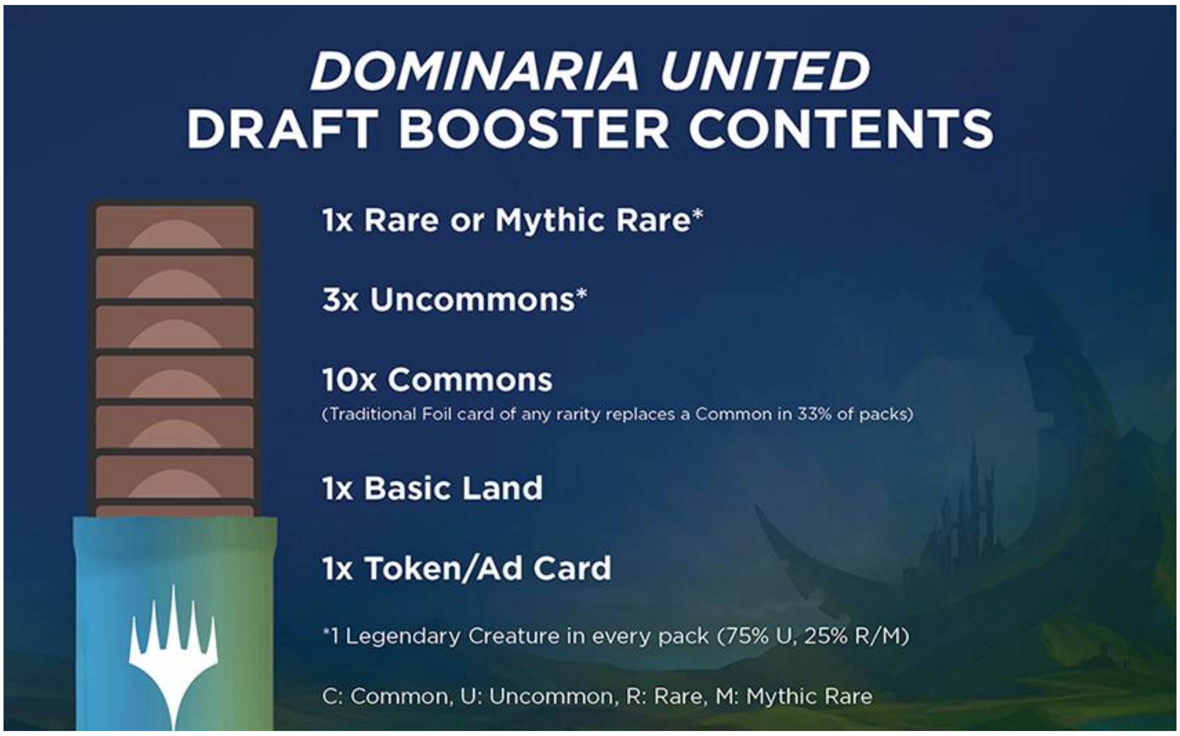
Name	Color	Rarity	# Seen	ALSA	# Picked	ATA	# GP	GP WR	# OH	OH WR	# GD	GD WR	# GIH	GIH WR	# GNS	GNS WR	IWD
Karn, Living Legacy		M	7650	1.80	3664	1.81	15026	54.5%	2552	52.8%	4201	55.8%	6753	54.7%	8099	54.2%	0.5pp
Anointed Peacekeeper	W	R	14849	1.77	8078	1.78	38887	58.4%	7001	63.4%	9877	58.7%	16878	60.6%	21781	56.7%	3.9pp
Archangel of Wrath	W	R	10422	1.25 ↗	8747	1.24	45594	61.1%	7859	65.4%	12317	68.7%	20176	67.4% ↑	24747	55.7%	11.7pp ↑
Argivian Cavalier	W	C	289648	3.98	69189	5.02	330397	58.5%	59363	60.4%	81124	58.4%	140487	59.3%	189014	58.0%	1.3pp
Argivian Phalanx	W	C	362617	5.17	54278	7.25	214278	57.5%	35393	56.3%	52745	56.5%	88138	56.4%	125542	58.3%	-1.9pp
Artillery Blast	W	C	399724	5.87	49424	8.78	139452	55.6%	23964	53.9%	37621	56.6%	61585	55.5%	76760	55.7%	-0.1pp
Benalish Faithbonder	W	C	428260	6.93	48708	10.15	112606	56.5%	20423	55.8%	25919	52.9%	46342	54.2%	66103	58.2%	-4.0pp

Figure 3. Screenshot of 17lands processed data

4.2 Dominaria United

Dominaria United is the 93rd MTG expansion, and the draft data of this set is being used to train the model. This is a normal MTG set comprised of 286 unique cards, being 101 commons, 80 uncommons, 60 rares, 20 mythic rares, 20 basic lands and 5 extra promo cards.

We are just interested in the first 261 cards that are the ones we can open during a draft. This draft booster packs are made of 1x Rare or Mythic rare card, 3 Uncommons, 10 commons, plus a basic land and a token/ad card. While drafting, players will only use the first 14 cards, so basically only the rare/mythic card, the 3 uncommons and the 10 commons are relevant when drafting (Figure 4). Being mythic rare, rare, uncommon or common just changes the ratio you will see the card while playing, and usually the rare and mythic rare cards are better cards and have this rarity, so they are seen less, and people do not have to worry as much while playing.


DOMINARIA UNITED
DRAFT BOOSTER CONTENTS

1x Rare or Mythic Rare*

3x Uncommons*

10x Commons
(Traditional Foil card of any rarity replaces a Common in 33% of packs)

1x Basic Land

1x Token/Ad Card

*1 Legendary Creature in every pack (75% U, 25% R/M)

C: Common, U: Uncommon, R: Rare, M: Mythic Rare

Figure 4. Contents of a Dominaria United Draft Booster (from Wizards of the Coast)

4.3 The Data Format

The dataset that stores the draft information includes one row per draft pick, with information about what the previous picks were, as well as some extra information about how the matches for the specific draft went, and the user win-rate. The details of the fields available are given in Table 1.

expansion	The 3-letter acronym that indicates the set. In our case it is DMU that represents Dominaria United.
event_type	The type of limited format this data is from. Can be PremierDraft, TradDraft, Sealed, QuickDraft. In our case it is PremierDraft.
draft_id	A hexadecimal number with 32 digits. A code representing the specific draft.
draft_time	The date and time of the draft.
rank	The rank of the player. This is a classification used in-game. Winning games increases your rank, while losing

	lowers it. In this spot we will see NaN (not ranked), or one of the ranks that ordered are: bronze, silver, gold, platinum, diamond, mythic
event_match_wins event_match_losses	The number of wins and losses of the draft after the player has played it. Both elements are related. There are 2 types of limited events, one where you play until you get 7 wins or 3 losses, and another kind where you play 3 matches without caring about wins and losses. The type of event is related to event_type. The numbers will follow these rules related to the event.
pack_number pick_number	These numbers are related to from what pack and what pick is the card you are getting now. Pack numbers go from 1-3 and pick numbers from 1-15. Example: If you are picking the third card from the second pack, it is pack 2 pick 3, or also noted as P2P3
pick_maindeck_rate	The number of games the user played with the picked card in their maindeck, divided by the total number of games the user played (with that draft pool). Maindeck are the 40+ cards that are played from the pool
pick_sideboard_in_rate	The number of games the user played with the picked card in their sideboard, divided by the total number of games the user played (with that draft pool). Sideboard is made from the cards of the pool not played in maindeck
user_game_win_rate_bucket	The approximate user win-rate of the format with the expansion from expansion parameter, rounded to 2 decimals. Win-rate is the number of victories divided by the total number of games
user_n_games_bucket	Number of games played by user in the format with the expansion from expansion parameter. Rounded to closest from following list: [1, 5, 10, 50, 100, 500, 1000, 5000]
pack_card_NAME	Grouped multiple entries, each one with a name of a card instead of NAME. Indicates with a 1 if the card NAME is present on the pack to pick it, or a 0 otherwise.
pool_NAME	Grouped multiple entries, each one with a name of a card instead of NAME.

	Information about previous picks
pick	NAME of the card selected between the cards that were in the pack, the ones with a 1 in pack_card_NAME

Table 1. Dataframe information

From all these available information, in this work, wwe will use only the data in the variables *pack_card_NAME* and *pool_NAME* as input features of our model and *pick* variable as output. Our goal is to see if with small set of input data, we can train a sufficiently good model for predicting the selected card (pick).

The rest of the data values could be used to add additional information or to filter or curate the dataset. Here I will name different ways we could exploit these additional variables to try to get better results, also mentioning some limitations to take into account:

- use RANK to filter bad players. Players that play a lot and are good usually get high ranks, so we could use this to select data from experienced players. There is a downside that we can lose good data, since maybe there is a good player that has not played in a while and has lost rank this way, or if a good player has created a new account, and that one will have a low rank to start even though the player could really be in the highest rank.
- EVENT_MATCH_WINS and EVENT_MATCH_LOSSES represent how good a player has played a specific draft. This can also be used to filter bad deck choices, but again it is not an optimal choice, since with the game variance, you can just get unlucky and face opponents with better decks than yours and loose because of that, just get unlucky and loose too, or the other way around, make horrible draft decisions and still get lucky and win.
- PICK_MAINDECK_RATE tells if the deck is played in the draft or not. This can be used to filter important picks and non-important picks, since the most important cards are usually the ones played. This has the disadvantage of probably not helping enough for the last picks of a pack, where a player is usually forced to pick bad cards that will not be played. Another disadvantage is that sometimes players start building for a specific archetype but find another color open and decide to change strategies, building a different deck with more quality. This can happen and switching lanes as it is known is a good thing that you have to sometimes do, but if we filter with pick_maindeck_rate when this happens, the first picks that probably are correct for the draft would be ignored.
- USER_GAME_WIN_RATE_BUCKET tells the percentage of games win by the player the draft data is from. This can be a good metric to filter the data, but again presents some problems. If a player has played a low number of games, this metric is not correctly balanced. In the extreme case that a player has only played a game and won the win-rate would be 100%, but that does not accurately show the skill. This can be complemented with USER_N_GAMES_BUCKET to make sure a player has played a decent number of games before evaluating the WR.

An example with two sets of data is shown in table 2. Some rows related to pack cards and to pool cards that only had 0s have been omitted to reduce the size of the table.

index	0	1
expansion	DMU	DMU
event_type	PremierDraft	PremierDraft
draft_id	562c07bcca0342d79365485503d153e4	562c07bcca0342d79365485503d153e4
draft_time	2022-08-31 00:44:45	2022-08-31 00:44:45
rank	NaN	NaN
event_match_wins	0	0
event_match_losses	3	3
pack_number	0	0
pick_number	0	1
pick	Ivy, Gleeful Spellthief	Pixie Illusionist
pick_maindeck_rate	1	1
pick_sideboard_in_rate	0	0
pack_card_Pixie Illusionist	0	1
pack_card_Sunlit Marsh	1	0
pack_card_Battlewing Mystic	0	1
pack_card_Shore Up	1	0
pack_card_Heroic Charge	1	0
pack_card_Prayer of Binding	0	1
pool_Ivy, Gleeful Spellthief	0	1
pack_card_Battle-Rage Blessing	1	0
pack_card_Sacred Peaks	0	1
pack_card_Rulik Mons, Warren Chief	0	1
pack_card_Sunbathing Rootwalla	1	0
pack_card_Gaea's Might	0	1
pack_card_Idyllic Beachfront	1	0
pack_card_Phyrexian Missionary	1	0
pack_card_Mesa Cavalier	0	1
pack_card_Ivy, Gleeful Spellthief	1	0
pack_card_Soaring Drake	1	0
pack_card_Coalition Skyknight	1	0
pack_card_Tidepool Turtle	0	1
pack_card_Jaya's Firenado	1	0
pack_card_Take Up the Shield	0	1
pack_card_Floriferous Vinewall	1	0
pack_card_Meria's Outrider	0	1
pack_card_Shadow Prophecy	0	1
pack_card_Molten Tributary	0	1
pack_card_Bog Badger	1	0
pack_card_Coral Colony	1	0
pack_card_Tangled Islet	0	1
user_n_games_bucket	1	1
user_game_win_rate_bucket	0	0

Table 2. Extract of draft dataset

5 Model definition and design

In this chapter, the model developed in this Bachelor thesis is presented. The design requirements for the construction of the card selection model and the decision made are explained and justified.

5.1 Input and Output

The inputs and outputs of an AI model are one of the most important parts of it. They determine what data must be used to make the automatic decision, and what kind of output the model gives.

In our case, the input should be information related to the pack of cards, in order to know the available options to choose. We have considered two different data representation options:

1. The first possibility is to introduce only the cards in the pack. However, this option presents the downside of having different inputs each time, since in each pack you pick from 15 to 1 card because each turn there is one less card in the pack. This approach would also need to code each card in a unique way so that our model recognizes each card and knows enough information about it to be able to choose. This notation may be overcomplicated for the machine learning methods.
2. The second possibility is to have a fixed set of input Boolean values, each one representing the presence or absence of one of the possible cards in the set, so that we have a 1 if the corresponding card is in the current pack, or 0 if it is not. This data representation format is, in fact, similar to the actual codification of cards in the draft dataset we will use (Table 2). This way there is always the same number of inputs since we always pass all the cards, and there is no need to code each individual card since they are identified by having a starting node each. With this as input format, we can fix to have always N inputs where N is the number of cards, and the inputs would be the columns from the database named *pack_card_NAME*, where *NAME* is the specific name of each card.

After considering the advantages and drawbacks of each of these two approaches, we have chosen the second one.

Regarding the output, the main result required is that we want to know the selected card. We could code each card in a unique way so that only 1 node as output is enough, but as before this is extra work. If we use the same approach as with the inputs, we can have in the output layer a node for each card and have 1 in the node that represents the selected card and a 0 in each other card. This would be N outputs where N is the number of cards. Output may be also a probability of selecting that card for each card choice, instead of a binary value.

Having N inputs and N outputs may be an appropriate format, but after a careful analysis of the game, it is maybe too simple. When a player drafts, the player not only chooses based on the cards in the pack, but also chooses knowing the cards he previously selected, trying to build a more synergistic deck. To add this information, we propose to follow the same approach as before and add another set of inputs, to give each card in the pool a node to indicate how many copies of that card are in the pool. This would mean again having N nodes where N is the number of cards, so each card has one input node and this input corresponds to *pool_NAME* that has a number between 0 (no cards) and x that is the max number of copies of a card you can get.

In summary, the proposal of this work is a model with 2N input nodes, one set of N for the possible cards in the pack, and another set of N for the possible cards in the pool, and an output of N nodes, one for each possible card that could be selected.

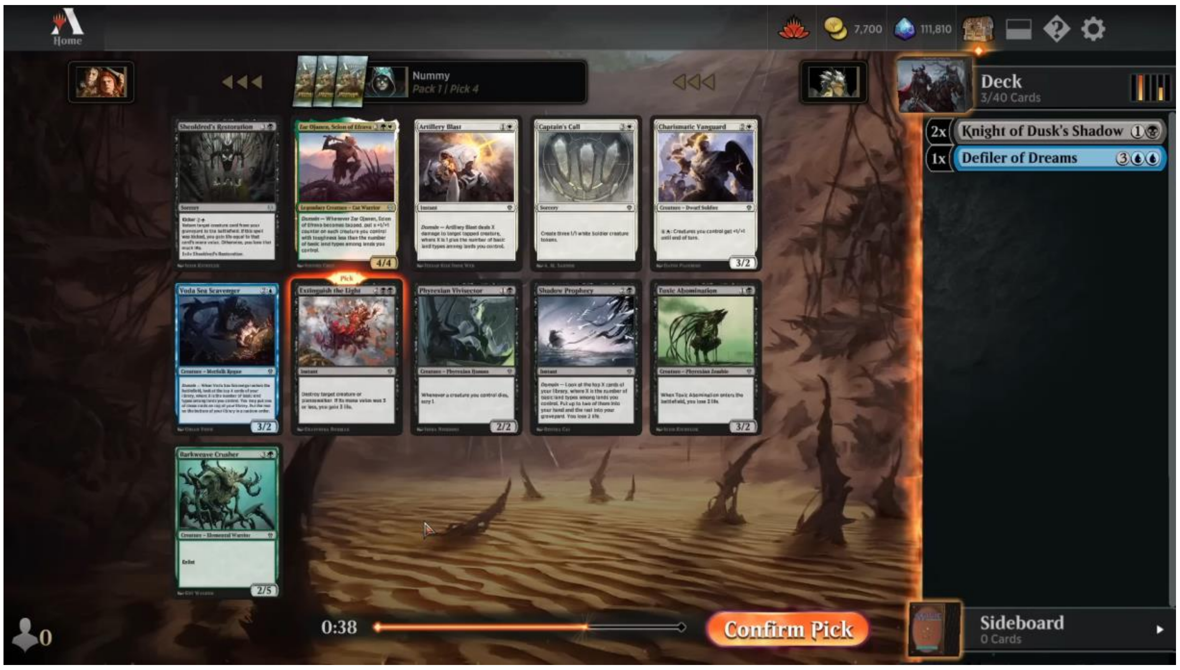


Figure 5. Pack exemple (screenshot from MTG Arena)

In this example (Figure 5) we are shown the selection of pack 1 pick 4. With the proposed input, our model would get each card in the main screen (pack) with a 1 in the respective node, and a 0 in all the other *pack_card_NAME*, and the cards on the right (pool) would also be given with the corresponding number, and a 0 in all the other *pool_NAME*. And as output we would get an array of 0s for each card except in the node of the selected card (highlighted in orange). The input would look like this, with the inputs with 1 and the output marked:

Input_card		Input_pick		Output	
pack_card_Academy Loremaster	0	pool_Academy Loremaster	0	Academy Loremaster	0
pack_card_Academy Wall	0	pool_Academy Wall	0	Academy Wall	0
pack_card_Adarkar Wastes	0	pool_Adarkar Wastes	0	Adarkar Wastes	0
pack_card_Aether Channeler	0	pool_Aether Channeler	0	Aether Channeler	0
pack_card_Aggressive Sabotage	0	pool_Aggressive Sabotage	0	Aggressive Sabotage	0
pack_card_Ajani, Sleeper Agent	0	pool_Ajani, Sleeper Agent	0	Ajani, Sleeper Agent	0
pack_card_Anointed Peacekeeper	0	pool_Anointed Peacekeeper	0	Anointed Peacekeeper	0
pack_card_Archangel of Wrath	0	pool_Archangel of Wrath	0	Archangel of Wrath	0
pack_card_Argivian Cavalier	0	pool_Argivian Cavalier	0	Argivian Cavalier	0
pack_card_Argivian Phalanx	0	pool_Argivian Phalanx	0	Argivian Phalanx	0
pack_card_Aron, Benalia's Ruin	0	pool_Aron, Benalia's Ruin	0	Aron, Benalia's Ruin	0
pack_card_Artillery Blast	1	pool_Artillery Blast	0	Artillery Blast	0
pack_card_Astor, Bearer of Blades	0	pool_Astor, Bearer of Blades	0	Astor, Bearer of Blades	0
pack_card_Automatic Librarian	0	pool_Automatic Librarian	0	Automatic Librarian	0
pack_card_Baird, Argivian Recruiter	0	pool_Baird, Argivian Recruiter	0	Baird, Argivian Recruiter	0
pack_card_Balduvian Atrocity	0	pool_Balduvian Atrocity	0	Balduvian Atrocity	0

pack_card_Balduvian Berserker	0	pool_Balduvian Berserker	0	Balduvian Berserker	0
pack_card_Balmor, Battlemage Captain	0	pool_Balmor, Battlemage Captain	0	Balmor, Battlemage Captain	0
pack_card_Barkweave Crusher	1	pool_Barkweave Crusher	0	Barkweave Crusher	0
pack_card_Battle-Rage Blessing	0	pool_Battle-Rage Blessing	0	Battle-Rage Blessing	0
pack_card_Battlefly Swarm	0	pool_Battlefly Swarm	0	Battlefly Swarm	0
pack_card_Battlewing Mystic	0	pool_Battlewing Mystic	0	Battlewing Mystic	0
pack_card_Benalish Faithbonder	0	pool_Benalish Faithbonder	0	Benalish Faithbonder	0
pack_card_Benalish Sleeper	0	pool_Benalish Sleeper	0	Benalish Sleeper	0
pack_card_Bite Down	0	pool_Bite Down	0	Bite Down	0
pack_card_Blight Pile	0	pool_Blight Pile	0	Blight Pile	0
pack_card_Bog Badger	0	pool_Bog Badger	0	Bog Badger	0
pack_card_Bone Splinters	0	pool_Bone Splinters	0	Bone Splinters	0
pack_card_Bortuk Bonerattle	0	pool_Bortuk Bonerattle	0	Bortuk Bonerattle	0
pack_card_Braids's Frightful Return	0	pool_Braids's Frightful Return	0	Braids's Frightful Return	0
pack_card_Braids, Arisen Nightmare	0	pool_Braids, Arisen Nightmare	0	Braids, Arisen Nightmare	0
pack_card_Broken Wings	0	pool_Broken Wings	0	Broken Wings	0
pack_card_Captain's Call	1	pool_Captain's Call	0	Captain's Call	0
pack_card_Caves of Koilos	0	pool_Caves of Koilos	0	Caves of Koilos	0
pack_card_Chaotic Transformation	0	pool_Chaotic Transformation	0	Chaotic Transformation	0
pack_card_Charismatic Vanguard	1	pool_Charismatic Vanguard	0	Charismatic Vanguard	0
pack_card_Choking Miasma	0	pool_Choking Miasma	0	Choking Miasma	0
pack_card_Citizen's Arrest	0	pool_Citizen's Arrest	0	Citizen's Arrest	0
pack_card_Cleaving Skyrider	0	pool_Cleaving Skyrider	0	Cleaving Skyrider	0
pack_card_Clockwork Drawbridge	0	pool_Clockwork Drawbridge	0	Clockwork Drawbridge	0
pack_card_Coalition Skyknight	0	pool_Coalition Skyknight	0	Coalition Skyknight	0
pack_card_Coalition Warbrute	0	pool_Coalition Warbrute	0	Coalition Warbrute	0
pack_card_Colossal Growth	0	pool_Colossal Growth	0	Colossal Growth	0
pack_card_Combat Research	0	pool_Combat Research	0	Combat Research	0
pack_card_Contaminated Aquifer	0	pool_Contaminated Aquifer	0	Contaminated Aquifer	0
pack_card_Coral Colony	0	pool_Coral Colony	0	Coral Colony	0
pack_card_Crystal Grotto	0	pool_Crystal Grotto	0	Crystal Grotto	0
pack_card_Cult Conscript	0	pool_Cult Conscript	0	Cult Conscript	0
pack_card_Cut Down	0	pool_Cut Down	0	Cut Down	0
pack_card_Danitha, Benalia's Hope	0	pool_Danitha, Benalia's Hope	0	Danitha, Benalia's Hope	0
pack_card_Deathbloom Gardener	0	pool_Deathbloom Gardener	0	Deathbloom Gardener	0
pack_card_Defiler of Dreams	0	pool_Defiler of Dreams	1	Defiler of Dreams	0
pack_card_Defiler of Faith	0	pool_Defiler of Faith	0	Defiler of Faith	0
pack_card_Defiler of Flesh	0	pool_Defiler of Flesh	0	Defiler of Flesh	0
pack_card_Defiler of Instinct	0	pool_Defiler of Instinct	0	Defiler of Instinct	0
pack_card_Defiler of Vigor	0	pool_Defiler of Vigor	0	Defiler of Vigor	0
pack_card_Destroy Evil	0	pool_Destroy Evil	0	Destroy Evil	0
pack_card_Djinn of the Fountain	0	pool_Djinn of the Fountain	0	Djinn of the Fountain	0
pack_card_Drag to the Bottom	0	pool_Drag to the Bottom	0	Drag to the Bottom	0
pack_card_Dragon Whelp	0	pool_Dragon Whelp	0	Dragon Whelp	0
pack_card_Eerie Soultender	0	pool_Eerie Soultender	0	Eerie Soultender	0

pack_card_Elas il-Kor, Sadistic Pilgrim	0	pool_Elas il-Kor, Sadistic Pilgrim	0	Elas il-Kor, Sadistic Pilgrim	0
pack_card_Electrostatic Infantry	0	pool_Electrostatic Infantry	0	Electrostatic Infantry	0
pack_card_Elfhame Wurm	0	pool_Elfhame Wurm	0	Elfhame Wurm	0
pack_card_Elvish Hydromancer	0	pool_Elvish Hydromancer	0	Elvish Hydromancer	0
pack_card_Ertai Resurrected	0	pool_Ertai Resurrected	0	Ertai Resurrected	0
pack_card_Ertai's Scorn	0	pool_Ertai's Scorn	0	Ertai's Scorn	0
pack_card_Essence Scatter	0	pool_Essence Scatter	0	Essence Scatter	0
pack_card_Evolved Sleeper	0	pool_Evolved Sleeper	0	Evolved Sleeper	0
pack_card_Extinguish the Light	1	pool_Extinguish the Light	0	Extinguish the Light	1
pack_card_Fires of Victory	0	pool_Fires of Victory	0	Fires of Victory	0
pack_card_Floriferous Vinewall	0	pool_Floriferous Vinewall	0	Floriferous Vinewall	0
pack_card_Flowstone Infusion	0	pool_Flowstone Infusion	0	Flowstone Infusion	0
pack_card_Flowstone Kavu	0	pool_Flowstone Kavu	0	Flowstone Kavu	0
pack_card_Forest	0	pool_Forest	0	Forest	0
pack_card_Founding the Third Path	0	pool_Founding the Third Path	0	Founding the Third Path	0
pack_card_Frostfist Strider	0	pool_Frostfist Strider	0	Frostfist Strider	0
pack_card_Furious Bellow	0	pool_Furious Bellow	0	Furious Bellow	0
pack_card_Gaea's Might	0	pool_Gaea's Might	0	Gaea's Might	0
pack_card_Garna, Bloodfist of Keld	0	pool_Garna, Bloodfist of Keld	0	Garna, Bloodfist of Keld	0
pack_card_Geothermal Bog	0	pool_Geothermal Bog	0	Geothermal Bog	0
pack_card_Ghitu Amplifier	0	pool_Ghitu Amplifier	0	Ghitu Amplifier	0
pack_card_Gibbering Barricade	0	pool_Gibbering Barricade	0	Gibbering Barricade	0
pack_card_Goblin Picker	0	pool_Goblin Picker	0	Goblin Picker	0
pack_card_Golden Argosy	0	pool_Golden Argosy	0	Golden Argosy	0
pack_card_Griffin Protector	0	pool_Griffin Protector	0	Griffin Protector	0
pack_card_Guardian of New Benalia	0	pool_Guardian of New Benalia	0	Guardian of New Benalia	0
pack_card_Hammerhand	0	pool_Hammerhand	0	Hammerhand	0
pack_card_Haughty Djinn	0	pool_Haughty Djinn	0	Haughty Djinn	0
pack_card_Haunted Mire	0	pool_Haunted Mire	0	Haunted Mire	0
pack_card_Haunting Figment	0	pool_Haunting Figment	0	Haunting Figment	0
pack_card_Herd Migration	0	pool_Herd Migration	0	Herd Migration	0
pack_card_Hero's Heirloom	0	pool_Hero's Heirloom	0	Hero's Heirloom	0
pack_card_Heroic Charge	0	pool_Heroic Charge	0	Heroic Charge	0
pack_card_Hexbane Tortoise	0	pool_Hexbane Tortoise	0	Hexbane Tortoise	0
pack_card_Hurler Cyclops	0	pool_Hurler Cyclops	0	Hurler Cyclops	0
pack_card_Hurloon Battle Hymn	0	pool_Hurloon Battle Hymn	0	Hurloon Battle Hymn	0
pack_card_Idyllic Beachfront	0	pool_Idyllic Beachfront	0	Idyllic Beachfront	0
pack_card_Impede Momentum	0	pool_Impede Momentum	0	Impede Momentum	0
pack_card_Impulse	0	pool_Impulse	0	Impulse	0
pack_card_In Thrall to the Pit	0	pool_In Thrall to the Pit	0	In Thrall to the Pit	0
pack_card_Inscribed Tablet	0	pool_Inscribed Tablet	0	Inscribed Tablet	0
pack_card_Island	0	pool_Island	0	Island	0
pack_card_Ivy, Gleeful Spellthief	0	pool_Ivy, Gleeful Spellthief	0	Ivy, Gleeful Spellthief	0
pack_card_Jaya's Firenado	0	pool_Jaya's Firenado	0	Jaya's Firenado	0
pack_card_Jaya, Fiery Negotiator	0	pool_Jaya, Fiery Negotiator	0	Jaya, Fiery Negotiator	0

pack_card_Jhoira, Ageless Innovator	0	pool_Jhoira, Ageless Innovator	0	Jhoira, Ageless Innovator	0
pack_card_Jodah's Codex	0	pool_Jodah's Codex	0	Jodah's Codex	0
pack_card_Jodah, the Unifier	0	pool_Jodah, the Unifier	0	Jodah, the Unifier	0
pack_card_Join Forces	0	pool_Join Forces	0	Join Forces	0
pack_card_Joint Exploration	0	pool_Joint Exploration	0	Joint Exploration	0
pack_card_Juniper Order Rootweaver	0	pool_Juniper Order Rootweaver	0	Juniper Order Rootweaver	0
pack_card_Karn's Sylex	0	pool_Karn's Sylex	0	Karn's Sylex	0
pack_card_Karn, Living Legacy	0	pool_Karn, Living Legacy	0	Karn, Living Legacy	0
pack_card_Karplusan Forest	0	pool_Karplusan Forest	0	Karplusan Forest	0
pack_card_Keldon Flamesage	0	pool_Keldon Flamesage	0	Keldon Flamesage	0
pack_card_Keldon Strike Team	0	pool_Keldon Strike Team	0	Keldon Strike Team	0
pack_card_King Darien XLVIII	0	pool_King Darien XLVIII	0	King Darien XLVIII	0
pack_card_Knight of Dawn's Light	0	pool_Knight of Dawn's Light	0	Knight of Dawn's Light	0
pack_card_Knight of Dusk's Shadow	0	pool_Knight of Dusk's Shadow	2	Knight of Dusk's Shadow	0
pack_card_Lagomos, Hand of Hatred	0	pool_Lagomos, Hand of Hatred	0	Lagomos, Hand of Hatred	0
pack_card_Leaf-Crowned Visionary	0	pool_Leaf-Crowned Visionary	0	Leaf-Crowned Visionary	0
pack_card_Leyline Binding	0	pool_Leyline Binding	0	Leyline Binding	0
pack_card_Lightning Strike	0	pool_Lightning Strike	0	Lightning Strike	0
pack_card_Liliana of the Veil	0	pool_Liliana of the Veil	0	Liliana of the Veil	0
pack_card_Linebreaker Baloth	0	pool_Linebreaker Baloth	0	Linebreaker Baloth	0
pack_card_Llanowar Greenwidow	0	pool_Llanowar Greenwidow	0	Llanowar Greenwidow	0
pack_card_Llanowar Loamspeaker	0	pool_Llanowar Loamspeaker	0	Llanowar Loamspeaker	0
pack_card_Llanowar Stalker	0	pool_Llanowar Stalker	0	Llanowar Stalker	0
pack_card_Love Song of Night and Day	0	pool_Love Song of Night and Day	0	Love Song of Night and Day	0
pack_card_Magnigoth Sentry	0	pool_Magnigoth Sentry	0	Magnigoth Sentry	0
pack_card_Meria's Outrider	0	pool_Meria's Outrider	0	Meria's Outrider	0
pack_card_Meria, Scholar of Antiquity	0	pool_Meria, Scholar of Antiquity	0	Meria, Scholar of Antiquity	0
pack_card_Mesa Cavalier	0	pool_Mesa Cavalier	0	Mesa Cavalier	0
pack_card_Meteorite	0	pool_Meteorite	0	Meteorite	0
pack_card_Micromancer	0	pool_Micromancer	0	Micromancer	0
pack_card_Molten Monstrosity	0	pool_Molten Monstrosity	0	Molten Monstrosity	0
pack_card_Molten Tributary	0	pool_Molten Tributary	0	Molten Tributary	0
pack_card_Monstrous War-Leech	0	pool_Monstrous War-Leech	0	Monstrous War-Leech	0
pack_card_Mossbeard Ancient	0	pool_Mossbeard Ancient	0	Mossbeard Ancient	0
pack_card_Mountain	0	pool_Mountain	0	Mountain	0
pack_card_Nael, Avizoa Aeronaut	0	pool_Nael, Avizoa Aeronaut	0	Nael, Avizoa Aeronaut	0
pack_card_Najal, the Storm Runner	0	pool_Najal, the Storm Runner	0	Najal, the Storm Runner	0
pack_card_Negate	0	pool_Negate	0	Negate	0
pack_card_Nemata, Primeval Warden	0	pool_Nemata, Primeval Warden	0	Nemata, Primeval Warden	0
pack_card_Nishoba Brawler	0	pool_Nishoba Brawler	0	Nishoba Brawler	0
pack_card_Phoenix Chick	0	pool_Phoenix Chick	0	Phoenix Chick	0
pack_card_Phyrexian Espionage	0	pool_Phyrexian Espionage	0	Phyrexian Espionage	0
pack_card_Phyrexian Missionary	0	pool_Phyrexian Missionary	0	Phyrexian Missionary	0

pack_card_Phyrexian Rager	0	pool_Phyrexian Rager	0	Phyrexian Rager	0
pack_card_Phyrexian Vivisector	1	pool_Phyrexian Vivisector	0	Phyrexian Vivisector	0
pack_card_Phyrexian Warhorse	0	pool_Phyrexian Warhorse	0	Phyrexian Warhorse	0
pack_card_Pilfer	0	pool_Pilfer	0	Pilfer	0
pack_card_Pixie Illusionist	0	pool_Pixie Illusionist	0	Pixie Illusionist	0
pack_card_Plains	0	pool_Plains	0	Plains	0
pack_card_Plaza of Heroes	0	pool_Plaza of Heroes	0	Plaza of Heroes	0
pack_card_Prayer of Binding	0	pool_Prayer of Binding	0	Prayer of Binding	0
pack_card_Protect the Negotiators	0	pool_Protect the Negotiators	0	Protect the Negotiators	0
pack_card_Queen Allenal of Ruadach	0	pool_Queen Allenal of Ruadach	0	Queen Allenal of Ruadach	0
pack_card_Quirion Beastcaller	0	pool_Quirion Beastcaller	0	Quirion Beastcaller	0
pack_card_Radha's Firebrand	0	pool_Radha's Firebrand	0	Radha's Firebrand	0
pack_card_Radha, Coalition Warlord	0	pool_Radha, Coalition Warlord	0	Radha, Coalition Warlord	0
pack_card_Radiant Grove	0	pool_Radiant Grove	0	Radiant Grove	0
pack_card_Raff, Weatherlight Stalwart	0	pool_Raff, Weatherlight Stalwart	0	Raff, Weatherlight Stalwart	0
pack_card_Ratadrabik of Urborg	0	pool_Ratadrabik of Urborg	0	Ratadrabik of Urborg	0
pack_card_Relic of Legends	0	pool_Relic of Legends	0	Relic of Legends	0
pack_card_Resolute Reinforcements	0	pool_Resolute Reinforcements	0	Resolute Reinforcements	0
pack_card_Rith, Liberated Primeval	0	pool_Rith, Liberated Primeval	0	Rith, Liberated Primeval	0
pack_card_Rivaz of the Claw	0	pool_Rivaz of the Claw	0	Rivaz of the Claw	0
pack_card_Rona's Vortex	0	pool_Rona's Vortex	0	Rona's Vortex	0
pack_card_Rona, Sheoldred's Faithful	0	pool_Rona, Sheoldred's Faithful	0	Rona, Sheoldred's Faithful	0
pack_card_Rulik Mons, Warren Chief	0	pool_Rulik Mons, Warren Chief	0	Rulik Mons, Warren Chief	0
pack_card_Rundvelt Hordemaster	0	pool_Rundvelt Hordemaster	0	Rundvelt Hordemaster	0
pack_card_Runic Shot	0	pool_Runic Shot	0	Runic Shot	0
pack_card_Sacred Peaks	0	pool_Sacred Peaks	0	Sacred Peaks	0
pack_card_Salvaged Manaworker	0	pool_Salvaged Manaworker	0	Salvaged Manaworker	0
pack_card_Samite Herbalist	0	pool_Samite Herbalist	0	Samite Herbalist	0
pack_card_Scout the Wilderness	0	pool_Scout the Wilderness	0	Scout the Wilderness	0
pack_card_Sengir Connoisseur	0	pool_Sengir Connoisseur	0	Sengir Connoisseur	0
pack_card_Serra Paragon	0	pool_Serra Paragon	0	Serra Paragon	0
pack_card_Shadow Prophecy	1	pool_Shadow Prophecy	0	Shadow Prophecy	0
pack_card_Shadow-Rite Priest	0	pool_Shadow-Rite Priest	0	Shadow-Rite Priest	0
pack_card_Shalai's Acolyte	0	pool_Shalai's Acolyte	0	Shalai's Acolyte	0
pack_card_Shanna, Purifying Blade	0	pool_Shanna, Purifying Blade	0	Shanna, Purifying Blade	0
pack_card_Sheoldred's Restoration	1	pool_Sheoldred's Restoration	0	Sheoldred's Restoration	0
pack_card_Sheoldred, the Apocalypse	0	pool_Sheoldred, the Apocalypse	0	Sheoldred, the Apocalypse	0
pack_card_Shield-Wall Sentinel	0	pool_Shield-Wall Sentinel	0	Shield-Wall Sentinel	0
pack_card_Shivan Devastator	0	pool_Shivan Devastator	0	Shivan Devastator	0
pack_card_Shivan Reef	0	pool_Shivan Reef	0	Shivan Reef	0
pack_card_Shore Up	0	pool_Shore Up	0	Shore Up	0
pack_card_Silver Scrutiny	0	pool_Silver Scrutiny	0	Silver Scrutiny	0
pack_card_Silverback Elder	0	pool_Silverback Elder	0	Silverback Elder	0

pack_card_Slimefoot's Survey	0	pool_Slimefoot's Survey	0	Slimefoot's Survey	0
pack_card_Smash to Dust	0	pool_Smash to Dust	0	Smash to Dust	0
pack_card_Snarespinner	0	pool_Snarespinner	0	Snarespinner	0
pack_card_Soaring Drake	0	pool_Soaring Drake	0	Soaring Drake	0
pack_card_Sol'kanar the Tainted	0	pool_Sol'kanar the Tainted	0	Sol'kanar the Tainted	0
pack_card_Soul of Windgrace	0	pool_Soul of Windgrace	0	Soul of Windgrace	0
pack_card_Sphinx of Clear Skies	0	pool_Sphinx of Clear Skies	0	Sphinx of Clear Skies	0
pack_card_Splatter Goblin	0	pool_Splatter Goblin	0	Splatter Goblin	0
pack_card_Sprouting Goblin	0	pool_Sprouting Goblin	0	Sprouting Goblin	0
pack_card_Squee, Dubious Monarch	0	pool_Squee, Dubious Monarch	0	Squee, Dubious Monarch	0
pack_card_Stall for Time	0	pool_Stall for Time	0	Stall for Time	0
pack_card_Stenn, Paranoid Partisan	0	pool_Stenn, Paranoid Partisan	0	Stenn, Paranoid Partisan	0
pack_card_Strength of the Coalition	0	pool_Strength of the Coalition	0	Strength of the Coalition	0
pack_card_Stronghold Arena	0	pool_Stronghold Arena	0	Stronghold Arena	0
pack_card_Sulfurous Springs	0	pool_Sulfurous Springs	0	Sulfurous Springs	0
pack_card_Sunbathing Rootwalla	0	pool_Sunbathing Rootwalla	0	Sunbathing Rootwalla	0
pack_card_Sunlit Marsh	0	pool_Sunlit Marsh	0	Sunlit Marsh	0
pack_card_Swamp	0	pool_Swamp	0	Swamp	0
pack_card_Tail Swipe	0	pool_Tail Swipe	0	Tail Swipe	0
pack_card_Take Up the Shield	0	pool_Take Up the Shield	0	Take Up the Shield	0
pack_card_Talas Lookout	0	pool_Talas Lookout	0	Talas Lookout	0
pack_card_Tangled Islet	0	pool_Tangled Islet	0	Tangled Islet	0
pack_card_Tattered Apparition	0	pool_Tattered Apparition	0	Tattered Apparition	0
pack_card_Tatyova, Steward of Tides	0	pool_Tatyova, Steward of Tides	0	Tatyova, Steward of Tides	0
pack_card_Tear Asunder	0	pool_Tear Asunder	0	Tear Asunder	0
pack_card_Temporal Firestorm	0	pool_Temporal Firestorm	0	Temporal Firestorm	0
pack_card_Temporary Lockdown	0	pool_Temporary Lockdown	0	Temporary Lockdown	0
pack_card_Territorial Maro	0	pool_Territorial Maro	0	Territorial Maro	0
pack_card_The Cruelty of Gix	0	pool_The Cruelty of Gix	0	The Cruelty of Gix	0
pack_card_The Elder Dragon War	0	pool_The Elder Dragon War	0	The Elder Dragon War	0
pack_card_The Phasing of Zhalfir	0	pool_The Phasing of Zhalfir	0	The Phasing of Zhalfir	0
pack_card_The Raven Man	0	pool_The Raven Man	0	The Raven Man	0
pack_card_The Weatherseed Treaty	0	pool_The Weatherseed Treaty	0	The Weatherseed Treaty	0
pack_card_The World Spell	0	pool_The World Spell	0	The World Spell	0
pack_card_Thran Portal	0	pool_Thran Portal	0	Thran Portal	0
pack_card_Threats Undetected	0	pool_Threats Undetected	0	Threats Undetected	0
pack_card_Thrill of Possibility	0	pool_Thrill of Possibility	0	Thrill of Possibility	0
pack_card_Tidepool Turtle	0	pool_Tidepool Turtle	0	Tidepool Turtle	0
pack_card_Timeless Lotus	0	pool_Timeless Lotus	0	Timeless Lotus	0
pack_card_Timely Interference	0	pool_Timely Interference	0	Timely Interference	0
pack_card_Tolarian Geyser	0	pool_Tolarian Geyser	0	Tolarian Geyser	0
pack_card_Tolarian Terror	0	pool_Tolarian Terror	0	Tolarian Terror	0
pack_card_Tori D'Avenant, Fury Rider	0	pool_Tori D'Avenant, Fury Rider	0	Tori D'Avenant, Fury Rider	0
pack_card_Toxic Abomination	1	pool_Toxic Abomination	0	Toxic Abomination	0

pack_card_Tribute to Urborg	0	pool_Tribute to Urborg	0	Tribute to Urborg	0
pack_card_Tura Kennerüd, Skyknight	0	pool_Tura Kennerüd, Skyknight	0	Tura Kennerüd, Skyknight	0
pack_card_Twinferno	0	pool_Twinferno	0	Twinferno	0
pack_card_Urborg Lhurgoyf	0	pool_Urborg Lhurgoyf	0	Urborg Lhurgoyf	0
pack_card_Urborg Repossession	0	pool_Urborg Repossession	0	Urborg Repossession	0
pack_card_Urza Assembles the Titans	0	pool_Urza Assembles the Titans	0	Urza Assembles the Titans	0
pack_card_Uurg, Spawn of Turg	0	pool_Uurg, Spawn of Turg	0	Uurg, Spawn of Turg	0
pack_card_Valiant Veteran	0	pool_Valiant Veteran	0	Valiant Veteran	0
pack_card_Vanquisher's Axe	0	pool_Vanquisher's Axe	0	Vanquisher's Axe	0
pack_card_Vesuvan Duplimancy	0	pool_Vesuvan Duplimancy	0	Vesuvan Duplimancy	0
pack_card_Viashino Branchrider	0	pool_Viashino Branchrider	0	Viashino Branchrider	0
pack_card_Vineshaper Prodigy	0	pool_Vineshaper Prodigy	0	Vineshaper Prodigy	0
pack_card_Voda Sea Scavenger	1	pool_Voda Sea Scavenger	0	Voda Sea Scavenger	0
pack_card_Vodalian Hexcatcher	0	pool_Vodalian Hexcatcher	0	Vodalian Hexcatcher	0
pack_card_Vodalian Mindsinger	0	pool_Vodalian Mindsinger	0	Vodalian Mindsinger	0
pack_card_Vohar, Vodalian Desecrator	0	pool_Vohar, Vodalian Desecrator	0	Vohar, Vodalian Desecrator	0
pack_card_Volshe Tideturner	0	pool_Volshe Tideturner	0	Volshe Tideturner	0
pack_card_Walking Bulwark	0	pool_Walking Bulwark	0	Walking Bulwark	0
pack_card_Warhost's Frenzy	0	pool_Warhost's Frenzy	0	Warhost's Frenzy	0
pack_card_Weatherlight Compleated	0	pool_Weatherlight Compleated	0	Weatherlight Compleated	0
pack_card_Wingmantle Chaplain	0	pool_Wingmantle Chaplain	0	Wingmantle Chaplain	0
pack_card_Wooded Ridgeline	0	pool_Wooded Ridgeline	0	Wooded Ridgeline	0
pack_card_Writhing Necromass	0	pool_Writhing Necromass	0	Writhing Necromass	0
pack_card_Yavimaya Coast	0	pool_Yavimaya Coast	0	Yavimaya Coast	0
pack_card_Yavimaya Iconoclast	0	pool_Yavimaya Iconoclast	0	Yavimaya Iconoclast	0
pack_card_Yavimaya Sojourner	0	pool_Yavimaya Sojourner	0	Yavimaya Sojourner	0
pack_card_Yavimaya Steelcrusher	0	pool_Yavimaya Steelcrusher	0	Yavimaya Steelcrusher	0
pack_card_Yotia Declares War	0	pool_Yotia Declares War	0	Yotia Declares War	0
pack_card_Zar Ojanen, Scion of Efrava	1	pool_Zar Ojanen, Scion of Efrava	0	Zar Ojanen, Scion of Efrava	0
pack_card_Zur, Eternal Schemer	0	pool_Zur, Eternal Schemer	0	Zur, Eternal Schemer	0

Table 3. Full Inputs and outputs for specific pack

5.2 Neural Network architecture

The basic structure of the proposed neural network for solving this problem is a multi-layer perceptron. This structure has been selected for its capacity to learn complex patterns using multiple inputs and outputs.

The proposed architecture of the neural network will follow the structure shown in Figure 5, with fully connected neurons (i.e., perceptrons). Therefore, each perceptron is connected to all the perceptrons of the next row in a feedforward way, which means without forming loops and the information can only move forward, from the input layer, through the hidden layers to the output layer.

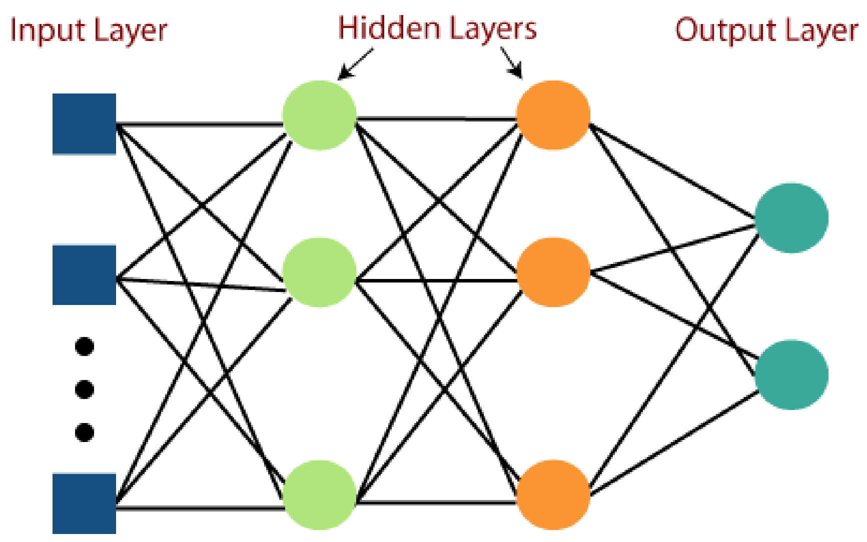
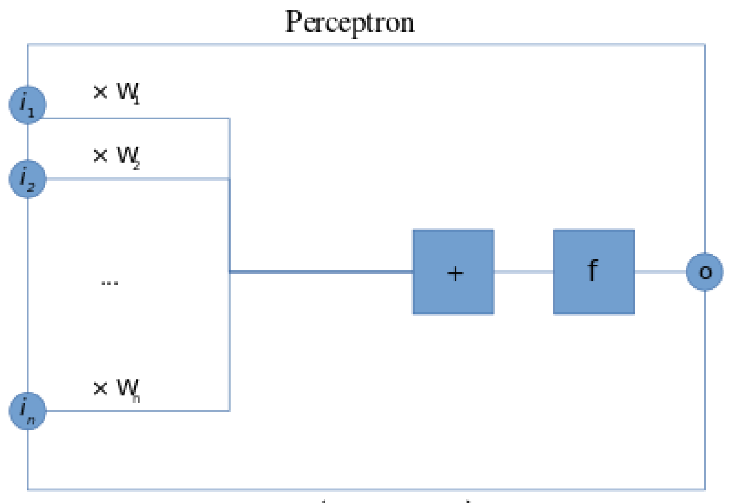


Figure 6. Layers in a MLP

Each perceptron takes inputs, then multiplies each input for a weight related to that input, adds all inputs together plus a weight, then applies an activation function and the output for the specific perceptron is ready to continue through the system, or get output if it is the last layer.



$$o = f\left(\sum_{k=1}^{n} i_k \cdot W_k\right)$$

Figure 7. Perceptron structure with output formula (from Mat the w)

In the following sections, we will detail the different elements of the neural network architecture that define the final architecture of the proposed system: activation function, the number of hidden layers and the loss functions.

5.2.1 Activation Function

An activation function is a function that decides if the neuron should be activated, or if the output can be ignored since it is not important for the neural network. Additionaly, in some cansed the activation function is used to modify the input so it is within a range.

There are multiple activation functions, but the most basic are the following (Table 4):

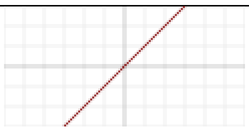
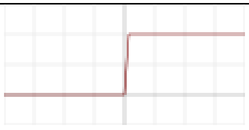
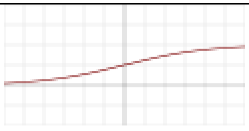
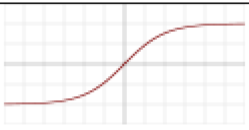
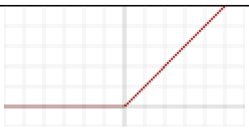
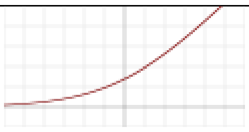
Name	Plot	Function	Range
Identity		x	$(-\infty, \infty)$
Binary step		$\begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$	$\{0, 1\}$
Logistic/Sigmoid		$\frac{1}{1-\ e^{-x}}$	$(0, 1)$
Hyperbolic Tangent		$\frac{e^x - e^{-x}}{e^x + e^{-x}}$	$(-1, 1)$
Rectified Linear Unit (ReLU)		$\max{(0, x)}$	$\{0, \infty)$
Softplus		$\ln{(1 + e^x)}$	$(0, \infty)$

Table 4. Activation Functions

These are the basic activation functions, there are also some extra a bit more complex that combine 2 of those or make some other changes, but all of these are only applied to a node at a time, modifying only a specific output.

Outside these functions, there is another one called argmax that is useful when predicting an output. This is a special function because instead of just taking as an input the value of a node, it takes as an input all the values from the nodes and makes the operation using all of them. This function gives the node with the highest output value a 1 and all the others a 0, selecting the label that matches the most with the prediction.

Of the options available, the ReLU function is the best option for the problem we are considering. The cost of applying a max() function is comparable to a basic arithmetic function, so it has a low cost while still being useful deactivating certain nodes. And since our objective does not have a linear output rather a binary one, we do not need to worry about continuity. And for the final layer, the output layer, we can use the logistic activation function, so we get the output values in the range of (0, 1) to indicate the selected card.

5.2.2 Hidden Layers

Hidden layers are the layer between the input layer and the output layer. There is no scientific way to tell how many layers are needed. Different problems require different things, for example it might be useful to have a hidden layer for each element you want to be able to identify in a picture, but it usually ends up being chosen by trial and error. A neural network with 1 hidden layer is called a simple neural network, while a neural network with more than one hidden layer is called a deep neural network.

The number of nodes inside the hidden layers can also vary, providing different results but sometimes an extra node or one less does not make a difference. There are some rule-of-thumb methods to determine the number of neurons to use in a hidden layer, but it is not something exact. These are ones fairly used:

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

In our case, the first rule for a set with 300 cards would be that the number of hidden neurons should be between 600 and 300, while the second rule tells us we should use 700 neurons. This is another thing that can be tested to see if there is any difference in performance.

5.2.3 Loss function

When training a model, you need to evaluate how well the model is doing, and then adjust to fit better the data. To evaluate this, the model uses a loss function to calculate an error, and this value is used via backpropagation to modify the weights and bias of the model.

There are multiple functions to calculate the error, some of the most used ones are the squared loss $L = (a)^2$, or the absolute loss $L = |a|$. The loss functions must be selected by looking at the cases specifically and is not as flexible as the number of hidden layers or activation function.

If we look at our case, we want the output to have a format of an array filled with zeros and in only one element of the array have a 1, looking something like this:

[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, …, 0, 0, 0]

The output is in a binary state, either 0 for not selected cards, or 1 for selected card, and since we will only select one card, each array being unique for a specific card, with no repeating arrays or cards, we can tell each array is represented by a label, the card name. And with this label, our multi-layer perceptron becomes a classifier, returning as output the class that is the selected card.

With this information in mind, and that the final layer activation function is logistic, the used function to calculate the loss is Binary Cross-Entropy loss. This function uses the probability of an element being correctly assigned to a label.

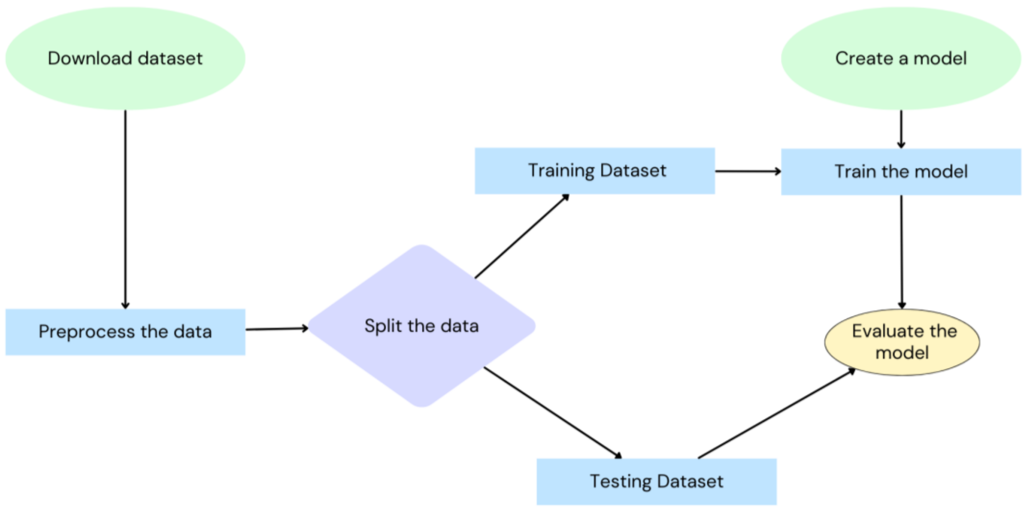
$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

It uses the negative log of the probability of being or not being the label and then makes the mean to give as error.

And we will be using the binary version of cross-entropy since we are classifying the data in different classes, that can be seen by a binary output.

5.3 Design of an AI system for MTG

The architecture of the system to implement for building a drafting system for MTG has the following structure:



Following these steps, a functional model should be created to later be used to predict card choices.

5.3.1 Data selection

From the dataset available, and presented in Chapter 3, we can take all the users or a subset. We can filter away data from bad players. Different options would be using the data that meets the criteria of having "user_game_win_rate_bucket" over 60% or restricting data only from players on specific ranks. Another way would be to filter so you only learn from decks that got at least 3 wins. But there is a problem here, and it is in Magic: The Gathering nature.

Magic: The Gathering is a game ruled by variance. It is a game of hidden information, and since each game you play you will not get all the cards you need to win, and the players need to play with what they draw. This leaves open the opportunity to win with a bad deck, lose with an amazing deck, or that a pro-player loses versus a new player, just because of the luck and variation implied in the game. But it is also thanks to this characteristic that makes it a fun game and less repetitive than other games like chess.

With this in mind, it is hard to select what is the correct data to use for training, and it is also hard to verify if the data is correct, since what a user has picked does not have to be the best choice, multiple choices can be equally good, or equally irrelevant, making it hard to evaluate. For this reason, we use all the dataset without doing any filtering.

5.3.2 Neural network structure

We propose a multi-layer perceptron classifier with 2N input nodes and N output nodes where N is the number of cards in a set. The activation function of the output layer will be a logistic function, and the loss function will be binary cross-entropy loss.

The other parts will be as a start:

- 1 hidden layer
- 400 nodes in the hidden layer
- ReLU activation function

In addition to these general parameters, for MTG, we want to choose unique card from a given fixed pack. This means picking a card from a specific subset of all the possible cards. If we negate this sentence, we must NOT choose a card NOT in the specific subset. To introduce this restriction to the neuronal network model, we propose the following operation:

Since in the input we already give an array that contains which cards can be selected and which cannot, we can use this information to mask the output, so we know for sure there is no chance a card not in the pack is selected.

This operation can be introduced in the model in the following way: we get the array with the cards in the pack, the first set of N, and multiply it with the output array before we apply the logistic function. This would be like applying a custom logistic function that not only takes as an input the actual value of the neuron, but it also takes as an extra input the original input for the card. We can relate those since each output node corresponds to a card, like the input nodes.

6 Implementation

This project has been developed using the Python language. It is one of the top languages for data analysis and AI development. In this chapter, the implementation project is explained. This implementation of the neuronal network has been done in two different ways, just as an academic exercise. First, the neuronal network model has completely programmed from scratch using the simple functions of Python. Second, anew implementation is done using Python machine learning libraries to make the code more efficient.

6.1 Basic implementation from scratch

The first implementation of the neural network propose has been done using basic functions of Python, and following the methodology explained in [12]. For this implementation, only 3 libraries were required:

- Pandas [13]: Pandas is a fast, powerful, flexible and easy to use open-source data analysis and manipulation tool, built on top of the Python programming language.
- Numpy [14]: Fast and versatile, the NumPy vectorization, indexing, and broadcasting concepts are the de-facto standards of array computing today.
- SciPy.special [15]: SciPy provides algorithms for optimization, integration, interpolation, eigenvalue problems, algebraic equations, differential equations, statistics and many other classes of problems.

With these functions imported, we can start as follows:

```
import numpy as np
import pandas as pd
import scipy.special
```

The initialization of the class is done in the following way:

```
class neuralNetwork :
    # initialise the neural network
    def    __init__(self,    inputnodes,    hiddennodes,    outputnodes,
learningrate):
        # set number of nodes in each input, hidden, output layer
        self.inodes = inputnodes
        self.hnodes = hiddennodes
        self.onodes = outputnodes

        self.weight_input_hidden = np.random.normal( 0, pow(self.hnodes,
-0.5),(self.hnodes, self.inodes))
        self.weight_hidden_output = np.random.normal ( 0, pow(self.onodes,
-0.5),(self.onodes, self.hnodes))

        self.learningrate = learningrate
        self.activation_function = lambda x : scipy.special.expit(x)
```

We create 2 weight arrays, one for the connection between the input and the hidden layer, and one for the hidden and the output layer. We also assign the learning rate for the training and the activation function.

As the goal was just building a proof-of-concept model, there are some missing components like the bias, or using a different activation function between the input layer and the hidden layer, that for now I just coded to use the logistic sigmoid function.

To train the neural network is also very easy:

```
    def train(self, inputs_list, targets_list):
        targets = np.array(targets_list, ndmin=2).T

        inputs = np.array(inputs_list, ndmin=2).T
        hidden_inputs = np.dot(self.weight_input_hidden, inputs)
        hidden_outputs = self.activation_function(hidden_inputs)
        final_inputs = np.dot(self.weight_hidden_output, hidden_outputs)
        final_outputs = self.activation_function(final_inputs)

        output_errors = binary_log_loss(targets, final_outputs)
        hidden_errors        =        np.dot(self.weight_hidden_output.T,
output_errors)
        self.weight_hidden_output       +=       self.learningrate       *
np.dot((output_errors   *   final_outputs   *   (1   -   final_outputs)),
np.transpose(hidden_outputs))
        self.weight_input_hidden       +=       self.learningrate       *
np.dot((hidden_errors   *   hidden_outputs   *   (1   -   hidden_outputs)),
np.transpose(inputs))
```

This implementation just takes the inputs, calculates the results by multiplying the inputs by the weights, applying the activation function and passing it to the next layer to do the same. Then it calculates the error using the binary_log_loss function that I defined and finalizes propagating the error so it can continue to learn with the next data.

To do the query, or the predictor, it is using the same lines of code of the first part of the train function to get the final outputs, plus the argmax function to reduce the multiple outputs to just the highest one.

This is enough to have a basic multi-layer processor, but it has some problems. Apart of not being fully complete, to train this model you have to give it the data one element at a time, making this process extremely slow. To avoid this problem, and be more efficient while training, another more professional implementation has been done by switching to using Python libraries specialized on providing AI machine learning tools.

6.2 AI Libraries

There are multiple libraries that are used for creating and training artificial intelligence models. Two of the most widespread are PyTorch and skilt-learn. The way of working of these two libraries has been studied to choose which one is more appropriate for this project.

6.2.1 PyTorch

PyTorch [16] is a machine learning framework based on the Torch library. It was originally developed by Meta AI with the first release in 2016, and some software developed using this library includes Tesla Autopilot, Uber's Pyro or Hugging Face's Transformers. And while fairly new it looks like one of the top tools on the market.

To create a model with PyTorch is extremely easy, you just need to define your own class specifying the layers you want to use using their predefined models. It looks something as simple as this:

```
class BasicNN(nn.Module):
    def __init__(self, n_input, n_output, n_hidden):
        super().__init__()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(n_input, n_hidden),
            nn.ReLU(),
            #nn.Linear(n_hidden, n_hidden),
            #nn.ReLU(),
            nn.Linear(n_hidden, n_output),
            #mascara
            nn.Sigmoid(dim=-1),
        )

    def forward(self, x):
        logits = self.linear_relu_stack(x)
        return logits
```

As you can see, you just write what layers you want to use, and also add the forward to define how you want to run your neural network, and you already have your neural network. All the other functions are already defined thanks to nn.Module, so the only thing left is training.

```
optimizer = SGD(model.parameters(), lr=0.1)
loss_func = nn.BCELoss()
print("Final bias, before optimization: " + "\n")

for epoch in range(100):

    total_loss = 0

    for iteration in range(len(inputs)):

        input_i = inputs[iteration].float()
        label_i = labels[iteration].float()

        output_i = model(input_i)

        loss = loss_func(output_i, label_i)

        loss.backward()

        total_loss += float(loss)
        optimizer.step()
        optimizer.zero_grad()

    if (total_loss < 0.0001):
        print("Num steps: "+str(epoch) + "\n")
        break



    print("Step:    "    +    str(epoch)    #+    "    model1    "+
str(model(inputs[70].float()).detach()) + " label " + str(labels[70])
               + " total loss " + str(total_loss)
               + "\n")

print("Final bias, after optimization: ")
```

This code trains the model with the data, with the given inputs and labels, and accumulates the losses to do steps using gradient descend and train.

Doing it this way works, but again it is slow since we are getting each input individually, and while it works with a small set of data, for bigger datasets like the one I will be using is too slow. To solve this problem, I should use torch.utils.DataLoader, the tool provided by PyTorch to load data, but it has some problems with big quantities of data and I also want to look at other libraries, so let's switch and leave it as a potential future upgrade.

6.2.2 Scikit-learn (Sklearn)

Scikit-learn [17] or Sklearn is a free machine learning library for python that features classification, regression, clustering algorithms and other tools. It is built on NumPy, SciPy and matplot. It was first released in 2007 and various organisations like Booking.com, JP Morgan, Evernote, Inria, AWeber, Spotify and many more are using Sklearn.

Sklearn already has a built-in multi-layer perceptron classifier, so we can use that to build our model, by just initializing it with the specific parameters we want for our build.

```
clf = MLPClassifier(hidden_layer_sizes=(400,),
                    activation= "relu",
                    batch_size=420)
```

Sklearn has been chosen as the final library to implement the model because of the simplicity it provides, it is easy to use and it is also very robust and optimized. Plus, Sklearn also provides a lot of additional features that can be used to preprocess the data or evaluate the model.

6.3 Advanced implementation with Sklearn

With Sklearn, we have created a multi-layer perceptron with one hidden layer of 400 nodes, ReLU as the activation function between layers, and the other presets already give the logistic sigmoid activation function on the last layer, binary cross-entropy loss and everything else we want. Batch_size is another argument that it is used while training to make minibatches of the data to feed it, and I chose to use 420 as the set I am going to train has 14 cards per pack, per 3 packs, makes a total of 42 picks. And This number times 10 means each batch feeds the data of 10 drafts. This model will be called "MLPClassifier".

As an upgrade, I also wanted to apply the mask to prevent the outputs from ever picking cards not in the pack, and to do so I looked at the code for MLPClassifier since it is open source [18]. Thanks to knowing how a MLP is coded, I found where I could do the modification. The functions `_forward_pass(self, activations)` and `_forward_pass_fast(self, X, check_input=True)` are the ones used for calculating the output of the model, the first used while training for the backpropagation algorithm, and the second one to predict the values.

To add the modification, I just needed to split the input values to save the pack_card_NAME inputs for the mask, and then apply said mask. To do the split, I will use the numpy function hsplit to divide the input array in 2. We can do it this way since the inputs will be ordered to have the picks first and the cards in pool later. And to apply the mask it is a simple multiplication, and this leaves the code this way:

```
def _forward_pass(self, activations):
    """Perform a forward pass on the network by computing the values
    of the neurons in the hidden layers and the output layer.
    Parameters
    ----------
    activations : list, length = n_layers - 1
        The ith element of the list holds the values of the ith layer.
    """

    hidden_activation = ACTIVATIONS[self.activation]
```

```
        # Save inputs for mask
        packs = np.hsplit(activations[0], 2)[0]

        # Iterate over the hidden layers
        for i in range(self.n_layers_ - 1):
            activations[i   +   1]   =   safe_sparse_dot(activations[i],
self.coefs_[i])
            activations[i + 1] += self.intercepts_[i]

            # For the hidden layers
            if (i + 1) != (self.n_layers_ - 1):
                hidden_activation(activations[i + 1])

        # For the last layer
        output_activation = ACTIVATIONS[self.out_activation_]
        activations[i + 1] = activations[i + 1] * packs
        output_activation(activations[i + 1])


        return activations

    def _forward_pass_fast(self, X, check_input=True):
        """Predict using the trained model
        This  is  the  same  as  _forward_pass  but  does  not  record  the
activations
        of all layers and only returns the last layer's activation.
        Parameters
        ----------
        X : {array-like, sparse matrix} of shape (n_samples, n_features)
            The input data.
        check_input : bool, default=True
            Perform input data validation or not.
        Returns
        -------
        y_pred : ndarray of shape (n_samples,) or (n_samples, n_outputs)
            The  decision  function  of  the  samples  for  each  class  in  the
model.
        """
        if check_input:
            X  =  self._validate_data(X,  accept_sparse=["csr",  "csc"],
reset=False)

        # Initialize first layer
        activation = X

        # Save inputs for mask
        packs = np.hsplit(X, 2)[0]

        # Forward propagate
        hidden_activation = ACTIVATIONS[self.activation]
        for i in range(self.n_layers_ - 1):
            activation = safe_sparse_dot(activation, self.coefs_[i])
            activation += self.intercepts_[i]
            if i != self.n_layers_ - 2:
                hidden_activation(activation)
        output_activation = ACTIVATIONS[self.out_activation_]
        activation = activation * packs
        output_activation(activation)


        return activation
```

The advantage of implementing a class that inherits all the other functions is that it only needs the initializer, these 2 modified functions and everything else works the same. To create a MLPClassifier with the modifications is the same as before, but just with another class name:

```
clf = DraftClassifier(hidden_layer_sizes=(400,400,),
                      activation= "relu",
                      batch_size=420)
```

This one has 2 layers of 400 cards, and all the other parameters are the same as before.

To train this model, we need to use the fit function predefined by the library. This function allows us to train the model by passing all the data and all the labels, and the library automatically takes care of feeding it multiple data at the time to be more efficient based on our batch_size. There is also a similar but better option for our case that is partial_fit(). Works exactly the same as fit, but while fit you have to give it all the data at once because it resets the model each time you call fit, with partial_fit you can keep feeding the data, perfect for big datasets like ours where it is hard to load all the data to the RAM, and it is also nice to do multiple epochs (multiple trainings with the same data) to get a better fitting. This model will be called “Draft classifier”.

6.4 Downloading the data

To obtain the data, it must be downloaded from 17lands.com. To do so, we can use the requests library. This is a simple library that allows you to send HTTP/1.1 requests very easily. With this code, the dataset is requested and saved, so it can later be read.

```
import requests

card_set = "DMU"

url = "https://17lands-public.s3.amazonaws.com/analysis_data/draft_data/draft_data_public."+card_set+".PremierDraft.csv.gz"
r = requests.get(url, allow_redirects=True)

open("/content/database_"+card_set+".csv.gz", "wb").write(r.content)
```

6.5 Data Preprocessing

Before training the model, the dataset also needs to be processed to have the correct data for training.

To read the data we can load it in memory using pandas read_csv, and we can get the dataset as a TextFileReader object, to then iterate over it and get chunks of data. To do this we can read the dataset as follows:

```
all_data = pd.read_csv("/content/database_"+card_set+".csv.gz", compression="gzip", chunksize= 10500, usecols = lambda x: x not in exclude)
data = next(all_data)
```

Where in chunksize we indicate how many rows we want it to read at a time, and what we give in usecols is columns we do not need to read.

X_data or input data will be the data in the columns containing “pack” or “pool” in the name. To get this data, we can use the filter function to just get those columns as follow:

```
X_data = data.filter(regex="pack|pool", axis=1).sort_index(axis=1)
```

Y-data or labels will be the data in the column “pick”. Using the library sklearn we can give the label directly and with just telling what all the possible labels are, the model already understands and relates each label with an output array. To read this data, it is like with X_data, but filtering with “pick” instead, and we do not need the sort.

6.6 Training and Testing

With the model built and the data ready to be processed, the training can be started. The code to do so looks like this:

```
for epoch in range(1, 11):
  all_data     =     pd.read_csv("/content/database_"+card_set+".csv.gz",
compression="gzip",  chunksize= 105000,  usecols  =  lambda  x:  x  not  in
exclude)
  i = 1
  for data in all_data:
    if (i%3 != 0):
      subdata = data.loc[data["user_game_win_rate_bucket"]>=0.6]
      X_data=subdata.filter(regex="pack|pool", axis =1).sort_index(axis=1)
      Y_data = np.ravel(subdata.filter(regex="pick", axis=1))
      clf.partial_fit(X_data, Y_data, classes=cardnames)
    i = i +1
  print(epoch)
  dump(clf, "train_mplc_1layer_a_epoch_"+str(epoch)+".joblid")
```

This code trains the data for 10 epochs, first reading all the data and then iterating over it. Then it only reads the training data using the counter to see what data is for testing, and after that it applies the preprocessing. After that, the processed data is separated in X_data and Y_data and finishes by calling the partial_fit function giving the data plus all the card names so it knows all the possible outputs and in what order they should be. After training with all the training data, what is called doing an epoch, the model is saved in a file for future references using the dump function, and this process is repeated until the data is trained enough times.

With this code, if we want to train with another kind of data filtering, changing the line that defines subdata is enough. And if you want to train another model, changing the constructor is everything that must be done.

To test the models, a similar approach is followed, where first the data is read, filtered, and then given to the model. To evaluate the success of the model, the function score() is used. This function compares the model output with the given output and counts how many times the output is the same to return the mean accuracy on the given test data and labels.

```
     all_data    =    pd.read_csv("/content/database_"+card_set+".csv.gz",
compression="gzip",  chunksize= 105000,  usecols  =  lambda  x:  x  not  in
exclude)
     i = 1
     for data in all_data:
       if(i%3 == 0):
         subdata = data.loc[data["user_game_win_rate_bucket"]>=0.6]
         X_data            =            subdata.filter(regex="pack|pool",
axis=1).sort_index(axis=1)
         Y_data = np.ravel(subdata.filter(regex="pick", axis=1))
         print(clf.score(X_data, Y_data))
       i = i + 1
```

Because of the size of the dataset, the training data chunks are passed one at a time, allowing you to see their individual performance.

7 Experimentation

As a first study, I trained 4 models with only 1 layer. Two of these models were a normal MLPClassifier while the other two were the custom draft classifier model class. Two settings have been tested: (1) training the models with all the data and (2) training the models with only the data from the players that have at least a 60% Win-Rate. The testing was also done in the same way, comparing each model versus all the testing dataset and versus a testing dataset with only the players with at least a 60% Win-Rate.

The performance results obtained in terms of the scoring function are the following:

MLPC 1layer	Trained All Tested All	Trained All Tested +60%	Trained +60% Tested All	Trained +60% Tested +60%
Max	0,647147656	0,660093652	0,624217955	0,639529717
Min	0,5626	0,581229305	0,543704762	0,563603753
Mean	0,628847619	0,64301363	0,607052381	0,622720229

Table 5 Test MLPC 1 layer

Draft 1layer	Trained All Tested All	Trained All Tested +60%	Trained +60% Tested All	Trained +60% Tested +60%
Max	0,62092027	0,634079084	0,606465929	0,620209646
Min	0,557580952	0,577400662	0,537885714	0,557464128
Mean	0,607728571	0,619265853	0,592657143	0,607173503

Table 6 Test Draft Classifier 1 Layer

Comparing the results of the two classification methods, as it can be seen, in Table 5 all the values are a bit higher than in Table 6. This means that the MLPClassifier has learned to only pick cards from the pack without needing a specific step for doing it. This observation has also been proven with some qualitative tests later, so the modification in the draft classifier model is not needed, and it even slows the training and lowers the final scores.

Since the data is fed to the model in batches because of its big size, during the calculations of the score we can also evaluate how each individual batch of data is classified. And the scores show that the later batches have a higher score than the first ones. And since the data in the dataframe is ordered in chronological order, this means that the model makes similar choices to the players that were playing after having more weeks of experience and knowledge. The best and worse scores obtained this way are shown in Max-Min rows in Table 5 and Table 6.

Another important conclusion is that the models trained with all the data have better results than only training with players with 60% Win-Rate. This can either be because using more data is beneficial for the models, and players could still make the correct choices but lose anyway later, or that using this filter is not the best way to curate the data. But on the other hand, the models scored higher when testing with data from the players with 60% Win-Rate.

The scores themselves are low. Having a score of maximum 66% is not the best, but this can be attributed as players not picking the best cards always. After analyzing in detail the case, we considered that the score function could be improved. Therefore, we propose a new scoring function that takes into account the first and second choice of the model and compares them with the actual output. And using this score function, we get higher values indicating that the bot results are indeed like those from human players, but it can have some different opinions on certain spots.

Using the top 2 choices we get much higher score (Table 7), having an 84% accuracy with the MLPClassifier and an 82% accuracy with the Draft Classifier model, both being trained with all the training data.

	Draft Classifier all data		MLPClassifier all data	
Tested with	all data	+60% WR	all data	+60% WR
Max	0,826301353	0,836784599	0,848537615	0,863423517
Min	0,768095238	0,785389073	0,775619048	0,790114514
Mean	0,808265334	0,819659442	0,82631129	0,838778911

Table 7 Score top 2 cards in 1-layer models

To further analyze how the different models classify, I have analyzed the accuracy of each individual pick and pack, getting the following results using the models trained with all the data and tested versus data of players with 60% WR:

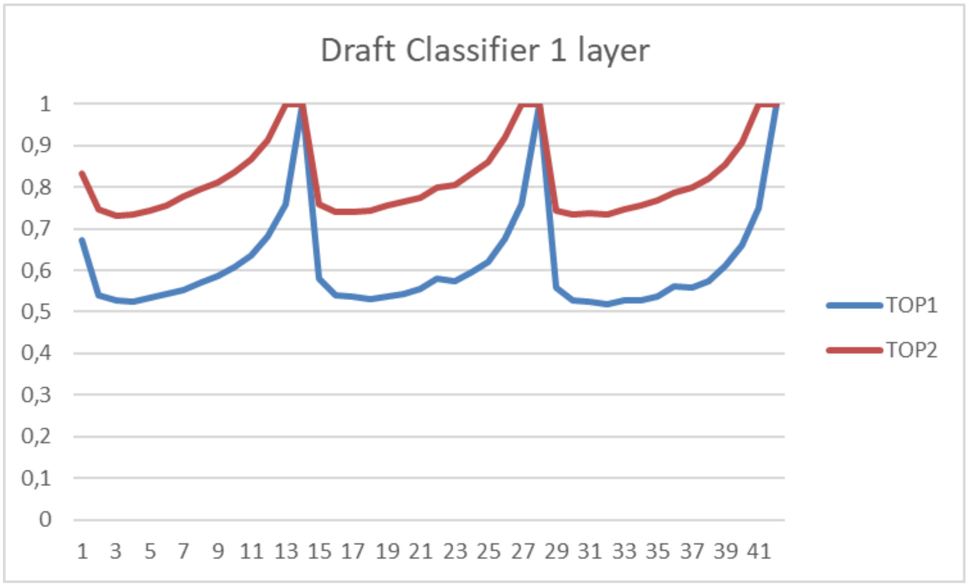


Figure 8 Score for each pick - draft classifier 1 layer

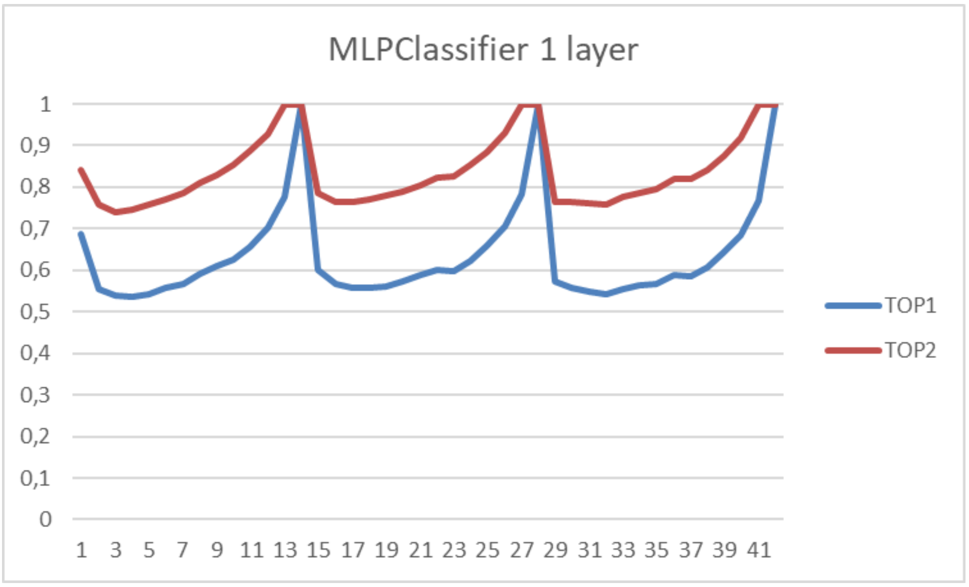


Figure 9 Score for each pick - MLPClassifier 1 layer

The highest picks are the ends of packs, where there are less cards and thus less options, making it easier for the bot to pick the same cards as the players. We can also see that the picks 2 to 5 of each pack are the ones with lower accuracy. This is because players usually take the easy choice on the first pick, and after that in picks 2-5 they might follow the first pick and try to force a deck playing that card while others are more flexible and open to picking an unrelated card that could be very good if the first card is not played, or they just pick the most expensive card in the pack for future uses outside the draft. Still, both models show a high score, being the lowest score for a specific pick just under 75% if we consider the best 2 cards.

After training models with 1 layer, the next step to evaluate is having extra layers. Because of the time cost, only two models have been trained, both using all the data, and the results obtained are shown in Table 7.

	Draft Classifier 2 layers	MLPClassifier 2 layers
All testing data top 1	0,618757143	0,638280952
All testing data top 2	0,824542857	0,838933333
60+% WR testing data top 1	0,630697789	0,653300168
60+% WR testing data top 2	0,833432241	0,849758799

Table 8 Avarage scores for 2-layered models

The scores shown in table 8 are almost identical to the ones shown in table 6, meaning that training extra layers this way is not more useful than just having one layer. This little improvement could mean that the models are learning some qualitative rules, but it needs further incentives to correctly learn that. If we look at the scores for each individual pick (Figure 10 and 11), the graphs show almost identical results to Figure 8 and Figure 9, further proving this point.

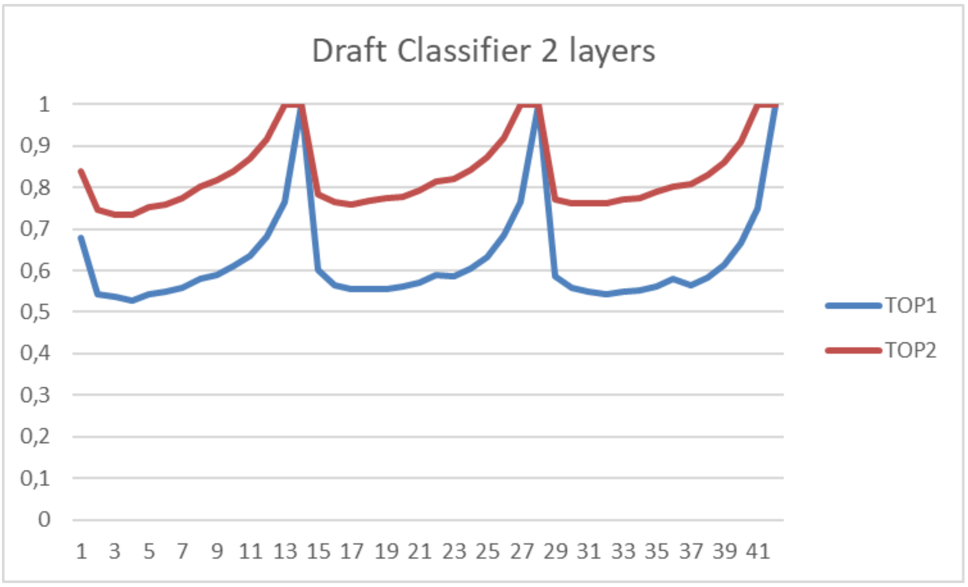


Figure 10 Score for each pick - Draft Classifier 2 layers

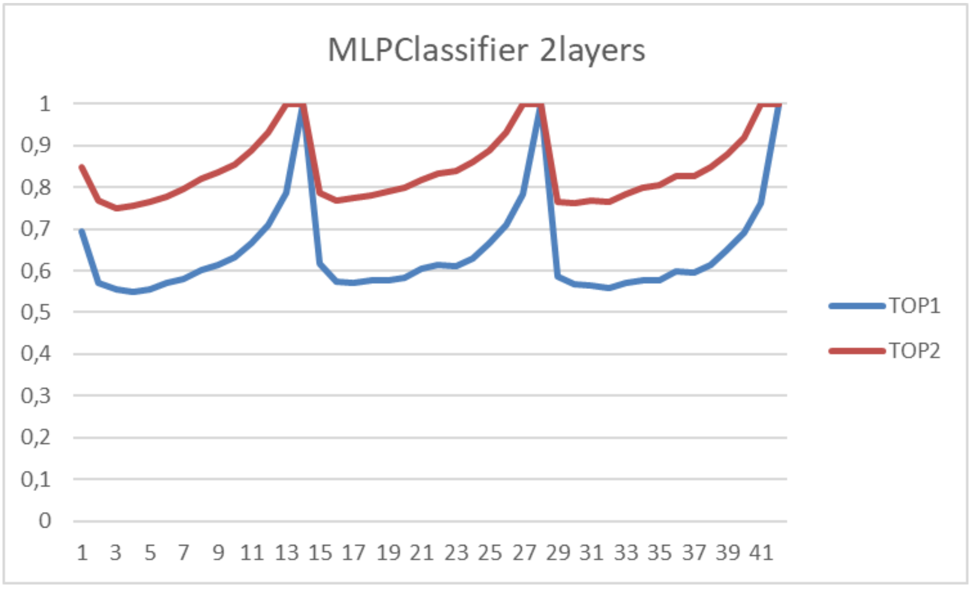


Figure 11 Score for each pick - MLPClassifier 2 layers

As a player, I can also do some qualitative tests. To do so, I tested the models with some new drafts, and evaluating the card choices. I have also shared these results with other players I know getting positive comments about it.

https://www.17lands.com/site_draft_replay/e2600ed0dbdd401281b84d81ff50ab46

This is one of the drafts I tested and asked for evaluation on other players. While sometimes the bot does not choose the best card in the pack, or picks more cards classified as *interaction* than what it is usually recommended to play, the model keeps choosing the two main colors when possible, as shown on Figure 12 where the model is picking red or white cards when possible because the cards in the pool (in screenshot shown as possible maindeck) are in those colors.

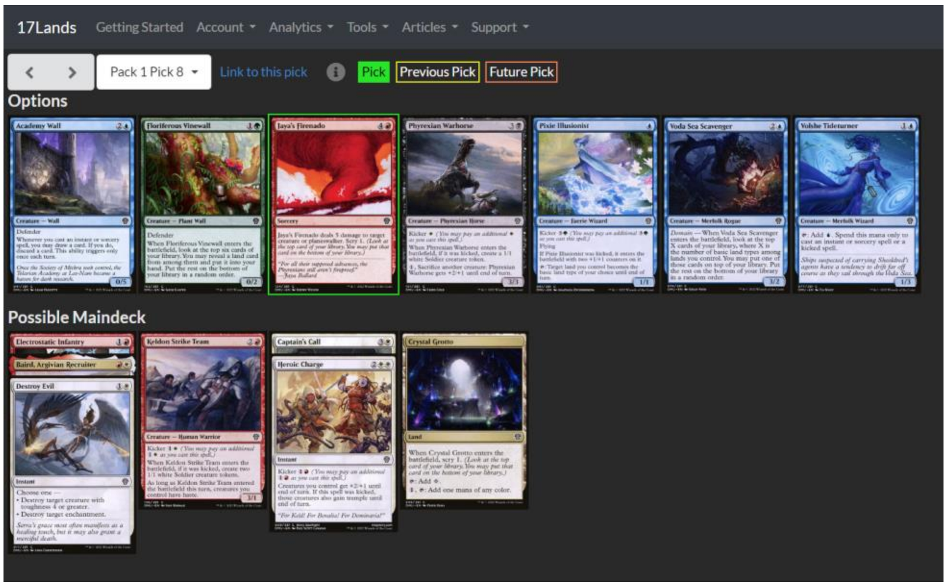


Figure 12 Real Draft Screenshot (screenshot from 17lands log)

While it is not optimal, it still follows basic drafting rules and makes good choices, better than what new players usually do, and it can be a good starting point or guide as it currently is for new players trying to learn to draft.

8 Conclusions

After finishing the work done in this project, the project has been completed fulfilling the proposed objectives:

1. I have conducted a comprehensive study on the state of the art in AI-based methods for card selection in Magic: The Gathering (MTG). This analysis allowed us to gain a deep understanding of the existing approaches and identify potential avenues for improvement in this challenging task.
2. I have designed a model with Machine Learning techniques to learn from the thousands of real card selections made by human players. It is based on Neural Networks. The input and output format has been carefully studied and a proposal based on binary codification of the cards in the pack and in the pick has been made.
3. The model has been implemented using state-of-the-art Machine Learning technologies. A first and simpler implementation of a neural network and its training has been done from scratch. Later, to improve speed and performance, an implementation with Skitlearn library has been done.
4. Different neural network models have been trained using the large datasets of human records. A multi-scale network model and a modified model for drafting into a fixed subset of cards have been studied. Results show that the standard multi-scale model has been also able to learn autonomously that the selected card must be one in the available pick.
5. Two different types of datasets have been used, including and excluding information for poorly performing players. As conclusion, it has been found that using all data gives more information to the model and so it produces better results.
6. The system has been evaluated both in a quantitative way and in a qualitative way, obtaining satisfactory results. A performance of 85% accuracy has been obtained while checking the top 2 options for the model. While the training could still be optimized to obtain better qualitative results, the models obtained work in a satisfying way and can be used to help new players of MTG.

The experience acquired in this project let me say that I think that neural networks are a good approach for building this kind of support tool for new players of complex card games. The methods proposed in this work could be also useful not only in the trained set but also in any other MTG set including future ones, plus in other similar games where drafting mechanics exist such as heartstone. A key point to building such systems is the availability of data for training, which fortunately was the case in MTG, thanks to 17lands.com.

As future work, there are 2 main points that can be worked. The first one would be the data preprocessing and include the gameplay data to give each input a specific weight so when training a model, the most useful data is taken with extra consideration, and the lesser important data, for example the data of losses, gets less importance. This would make the model learn more from the good choices and less about the bad ones, and probably also being able to use less data to train the models, since each individual data would bring more information. The second main point would be using more layers and trying to teach it more complex behaviors and see how better it can get. This would need the use of the previous explained data preprocessing, and a different kind of evaluation, since if it becomes better than an average player, trying to compare the choices of the model with the choices of average users could give a low accuracy, but because the humans do worse choices. And it would also need more computational resources to do the training and preprocess in a reasonable time.

Another extra project that could be done related with this is to make an overlay that reads the data from the online game while it is playing to predict and help the player make the choices without needing extra work from the player. This would help with the use of the project and would make it more accessible.

Finally, my personal evaluation of the work done is a good one. I’m satisfied with the results obtained both qualitative and quantitative ones, and I think it can be easily trained with future sets for further use. I’m happy with the results obtained and with the knowledge I learned doing this project, that will be useful for me in a future.

9 References

[1] https://magic.wizards.com/en/formats [formats] May 2022

[2] https://magic.wizards.com/en/formats/booster-draft [information about booster draft] May 2022

[3] https://youtu.be/fUqPxSYPfrA [draft explanation] May 2022

[4] M. Kubat, An Introduction to Machine Learning. Springer International Publishing, 2017

[5] Subramanian Chandramouli, Saikat Dutt, Amit Kumar Das. Machine Learning. Pearson Education India, 2018

[6] Sutton, R. S., & Barto, A. G. Reinforcement Learning: An Introduction. MIT Press. 2018

[7] LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. Nature 521, 436–444 (2015). https://doi.org/10.1038/nature14539

[8] https://www.urzas.ai/ [similar projects] May 2022

[9] https://medium.com/@lukbebalduke/mtg-hivemind-artificial-intelligence-designing-magic-372530640cc1 [similar projects] May 2022

[10] Henry N. Ward, Daniel J. Brooks, Dan Troha, Bobby Mills, Arseny S. Khakhalin. AI solutions for drafting in Magic: The Gathering. Proceeding os the 2021 IEEE Conference on Games, Copenhagen, Denmark 2021. https://ieeexplore.ieee.org/document/9619100

[11] Ryan Saxe, Bot Drafting the Hard Way: A New Drafting Model for Archetypes and Staying Open. Post in DraftSim blog. https://draftsim.com/ryan-saxe-bot-model/ [similar projects] 2020 (accessed May 2022).

[12] https://www.17lands.com [17lands goal] March 2022

[13] https://www.17lands.com/public_datasets [dataset] March 2022

[14] Tariq Rashid, Make Your Own Neural Network, CreateSpace, 2016.

[15] hthttps://pandas.pydata.org/docs/ [documentation] March 2022

[16] https://numpy.org/doc/stable/ [documentation] March 2022

[17] https://docs.scipy.org/doc/scipy/index.html [documentation] March 2022

[18] https://pytorch.org/ [documentation] March 2022

[19] https://scikit-learn.org/stable/ [documentation] April 2022

[20] https://github.com/scikit-learn/scikit-learn/blob/364c77e04/sklearn/neural_network/_multilayer_perceptron.py#L761 [MPLClassifier Code] April 2022