Modeling Magic: the Gathering

Michael Faulkner MJF5@WILLIAMS.EDU
Computer Science, Physics
Austin Osborn ADO2@WILLIAMS.EDU
Computer Science, Studio Art, Physics

1. Introduction

1.1 Abstract

In this project we sought to apply machine learning models to a strategic trading card game which both of us enjoy: Magic: the Gathering. The focus of this analysis is on constructing a data-informed bot which can make intelligent choices when constructing a deck from a limited card pool, specifically the format known as draft. Using the substantial public datasets available on 17Lands.com, we have accomplished the construction of a neural network which is capable of mimicking average human behavior with a test accuracy of 65.3%, a value which is higher than anticipated due to reasonable people disagreeing on what the correct choice is in such a complex game. We have also done work towards augmenting, or potentially surpassing, this model through the utilization of domain knowledge and side models trained on predicting win rate of a given deck and whether or not a given card in the pool makes the final deck.

1.2 Background

Magic: the Gathering is a multiplayer card game with many different ways to play. One of the popular formats is called LIMITED where players receive a small supply of cards from which they make a deck. Matches are then played one-on-one, using the decks that the players made. There are several variants of limited where the specific way players obtain their card pool is different. One of the most popular variants is DRAFT, which is typically played with a group of 8 people. In a draft, the 8 players are seated in a circle around a table, and each player begins by opening a booster pack of 15 cards (of around 330 possible cards with different rarities). Each player chooses a card from among them to add to their card pool and then passes the remaining to the player on their left (what this looks like on the online client can be seen in Figure 1. Each player then chooses one of the 14 cards they received from the player on their right and passes the remaining 13 to their left. This process repeats until all cards from the first pack have been selected. This then happens 2 more times with new packs. After drafting 3 packs, each player will have 45 cards that they will then use (in combination with an unlimited supply of the 5 types of basic resource cards) to construct a 40 card deck. Typically these decks will include about 23 of the 45 cards an individual drafted, with any given card's inclusion in the deck being dictated by a combination of the card's characteristics and what synergistic combinations of cards are

Figure 1: Example of a pack in a draft

available within a player's card pool. The decisions that players need to make at each step of this process and the ultimate quality of the deck they are able to create is dependent on a synthesis of an enormous amount of information. We seek to create a tool that will perform comparably to high level players. That is, we seek to make a bot which can make intelligent decisions about what cards to take while drafting, what cards should be included in a deck, and how well a deck can be expected to perform.

This problem, and problems similar to it have been investigated in various other works. A couple papers which had goals similar to ours within the domain of Magic: the Gathering were: Ward et al. (2021) who followed a similar approach to building a drafting bot as we did and Bjørke and Fludal (2017) who examined the deck building side of the game, which though we only touch on at the end of the project, mirrors many of the challenges we face in building an accurate draft bot.

2. Preliminaries

An artificial neural network is a machine learning model that approximates a function that takes in an input vector X and produces an outcome Y. A neural network is arranged in layers where each layer consists of a set of neurons $n_1, \ldots, n_i$ that each have edges

to all of the neurons in the next layer. We associate a weight with each of these edges that is used during a forward pass through the network to calculate subsequent neurons' values by passing a linear combination of the previous layer's values through a non-linear activation function. Thus, we compute the value of a neuron during a forward pass as $n = F(\theta_1 n_1 + \theta_2 n_2 + \cdots + \theta_i n_i)$ where F is a non-linear function that allows the network to encode more complex relationships. In our networks, we use the rectified linear unit

$$ReLU(x) \equiv \begin{cases} 0 & x \leq 0 \\ x & 0 < x \end{cases}$$

as our nonlinear activation for our inner layers. For our final layer of classification tasks, we use the softmax function

$$\text{SOFTMAX}(X)_i \equiv \frac{e^{x_i}}{\sum_{i=1}^{n} e^{x_i}}$$

which normalizes each value to be a positive probability that properly sums to one. The final layer of the network is its approximation for the variable Y.

The weights θ to associate with each edge can be learned by minimizing an appropriate loss function $L(\theta)$ using stochastic gradient descent. For regression tasks where we seek to approximate a numerical value, we use the Mean Squared Error which is defined as

$$L(\theta) \equiv \frac{1}{n} \sum_{i=1}^{n} (y_i - \widehat{y}_i)^2$$

where y_i is a true value and $\widehat{y}_i$ is the network's prediction for y_i. For classification tasks where we seek to predict an output as one of k distinct classes, we use the Negative Log Likelihood which is defined as

$$L(\theta) \equiv -\sum_{i=1}^{N} \log(\widehat{p}(y_i | x_{i1}, \ldots, x_{id}; \theta))$$

where $\widehat{p}(y_i | x_{i1}, \ldots, x_{id}; \theta)$ denotes the network's probability of predicting that the output is y_i given a sample input $X_i = x_{i1}, \ldots, x_{id}$ and its current values for the parameters θ. Stochastic gradient descent updates the values of θ by backpropogating the derivatives from the loss function to the input parameters, updating them as each sample is put through the network to minimize computational overhead and avoid overfitting.

Because neural networks are strong learners, it is a common practice to employ additional techniques to regularize them to prevent overfitting. We employ dropout when training our networks in order to improve our models' generalizability. Dropout proceeds by generating a vector of Bernoulli values for each parameter within the network at each step of stochastic gradient descent and multiplying the parameters by it, effectively deactivating neurons at random. This process makes the network more robust and less dependent on any single pathway.

For our implementation we used the pytorch deep learning library, which provides efficient tools for training models on the GPU (Paszke et al., 2019). Additionally, we use a utility function from Scikit-Learn that helps translate between strings and a one-hot-encoding representation for them (Pedregosa et al., 2011).

3. Data

3.1 Origin

All of our data was obtained from 17Lands.com which is a website that provides a game plugin that aggregates and synthesizes data from users as they play the primary digital version of Magic—Magic: the Gathering Arena. The public datasets they provide are substantial, with the main file we used (a .csv supplying information about decisions users made while drafting) containing over 7,000,000 rows and over 700 columns. This file being too large to store in local memory led to a bevy of challenges.

3.2 Pre-Processing

We had to do several rounds of pre-processing in order to get our data into a usable state, the two main motivating factors behind these steps were: removing unusable data which could lead to errant behaviour down the line, and minimizing the memory footprint as much as possible.

Since the dataset was originally compiled from scraped logs of a computer game, it is unsurprising that there were erroneous entries scattered throughout the dataset. These errors manifested as: incomplete drafts and cards missing from packs or pools. Each erroneous entry needed to be taken out of the dataset, and since each group of 42 consecutive rows of the dataset made up a single draft, if there was an error in one row, the whole draft needed to be removed.

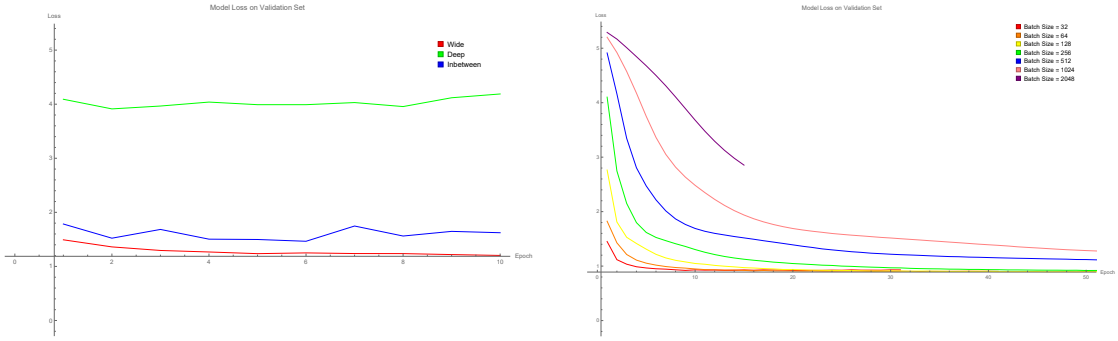
In addition to filtering out faulty entries, there were a significant number of columns in the dataset that possessed data that was irrelevant to the models we would be training (such as id, timestamp, etc.). Removing these extra columns not only helped ensure all of the data the neural network would have access to would be relevant, but also helped lower the memory footprint of the dataset since with over a billion cells in the dataset, it was impossible (on the machines we were working with) to store the entire file in local memory at once.

A final use of pre-processing in our project was applying domain knowledge to create sample parameters that reflect characteristics humans consider important when evaluating a deck. In particular, we sought to explicitly incorporate information about the color distribution of cards in a pool and to provide information about the aggregate card quality by providing the averaged win rate of the cards in the pool. Collecting this data required manually extracting it from the 17Lands.com website as it was excluded from the primary data file we downloaded.

4. Training And Validation Of Models

4.1 Baseline Logistic Regression

In order to establish a baseline model to make draft picks. We trained a neural network that transformed its input parameters directly into an output prediction, using no hidden layers. This minimalist network consists of just a linear transformation being passed into a



(a) Choosing model architecture

(b) Tuning batch size

Figure 2: Process for determining shape and hyperparameters

SOFTMAX transform to produce output probabilities. This combination is ordinarily referred to as logistic regression.

We trained a model using this architecture, providing the pack number, pick number, cards in the pack, and cards in your pool as input features, and used a Negative Log Likelihood loss function in order to train the model on correctly predicting the card chosen by a human. This model was trained using stochastic gradient descent with a learning rate of 0.01. During the training process the validation accuracy of this model was 63.5%. In the end, it achieved a testing accuracy of 63.3%.

4.2 Neural Network

In order to improve upon our baseline, we trained several more-complex neural networks. These models took the same input features as our logistic regression model, and we continued to use stochastic gradient descent with a learning rate of 0.01. In order to maintain reasonable training times, we limited the size of the models we explored to between 750,000 and 1,000,000 parameters. We primarily explored 3 model architectures: a wide but shallow network, a deep but narrow network, and one in-between. Our wide network had one hidden layer that contained 775 neurons, resulting in a model with 754,850 parameters. Our deep network had 5 hidden layers each with 325 neurons, resulting in a model with 844,675 parameters. The in-between network had two hidden layers each with 500 neurons, resulting in a model with 987,000 parameters. We trained each of these models for 10 epochs using stochastic gradient descent in order to identify the most promising model. The model's convergence on the validation set can be seen in figure 2a. After identifying that the wide model was performing the best, we then tuned the batch size in order to achieve the best model possible. The convergence of various models with varying batch size can be seen in figure 2b. From our validation testing, the best performing model was the wide model, trained with a batch size of 128. It achieved a validation accuracy of 65.7%.

4.3 Trying to Improve upon Naive Neural Network

Having achieved a better result than we were expecting with our neural network model, we sought to construct a different model which would not constrained to directly attempting to imitate humans. Our hope was for the computer to produce a novel, and successful approach to drafting. We have not yet able to create a model which meets this ideal, but our attempts have been informative. In order to disconnect our model from directly copying humans, we had to identify a new target for the loss function to replace our previous one that measured deviation from human picks. We decided that a reasonable target to pursue for this model would be to train on identifying the win-rate of a deck, and using this model we hoped we would be able to identify what card should be chosen from a pack by how it would affect the predicted win rate.

This new model was trained on a dataset of decks with 330 parameters as input which held an encoding of cards in the pack as well as metadata about the cards chosen. We included 5 additional input parameters that applied domain knowledge about the importance of color distribution by providing relative prevalence of each of the 5 colors of cards that were in the deck. We additionally included the mean win-rate of cards in the deck as a heuristic for the quality of the cards in the deck. The outcome Y was chosen to be the win-rate that the deck ended up with, so the neural network's final layer was a single one point. This model was trained with a learning rate ranging from 0.001-0.05. Due to the necessary preprocessing, this model had less data to draw from ("only" $\sim 200,000$ lines), so we gave it $> 1,000,000$ tunable parameters. Despite the complexity of the model, we were unable to achieve any reasonable predictive power from this model, and we were forced to consider other options.

We built upon this initial idea by training another network on the cards a player had in their deck and trying to predict the decks performance within the draft via bucketing the outcomes based on whether or not they achieved < 3 wins, between 3 and 5 wins, or > 5 wins. The best performing model we were able to achieve with these features was a logistic regression. This model achieved a validation accuracy of 44.4%. We then used this model to select cards from packs by selecting the card that maximized the difference between the predicted probabilities of achieving > 5 wins and < 3 wins. This model agreed with the human choice 41.4% of the time in our validation set. Despite having a reasonable amount of agreement, the deviations the model made were often nonsensical instead of insightful.

5. Results

5.1 Quantitative Results

The best results we have gotten have been from tuning the neural network that minimized deviations from human picks as the loss function (achieving the 65.3% test accuracy quoted before). This occurred when the batch size was at 128 as mentioned in 4.2. This outperforms the baseline logistic regression model by 2% which might not seem like much, but is significant when the logistic regression model is already doing a good job of modeling human behavior. This speaks to the efficacy of logistic regression and neural networks to make draft decisions.

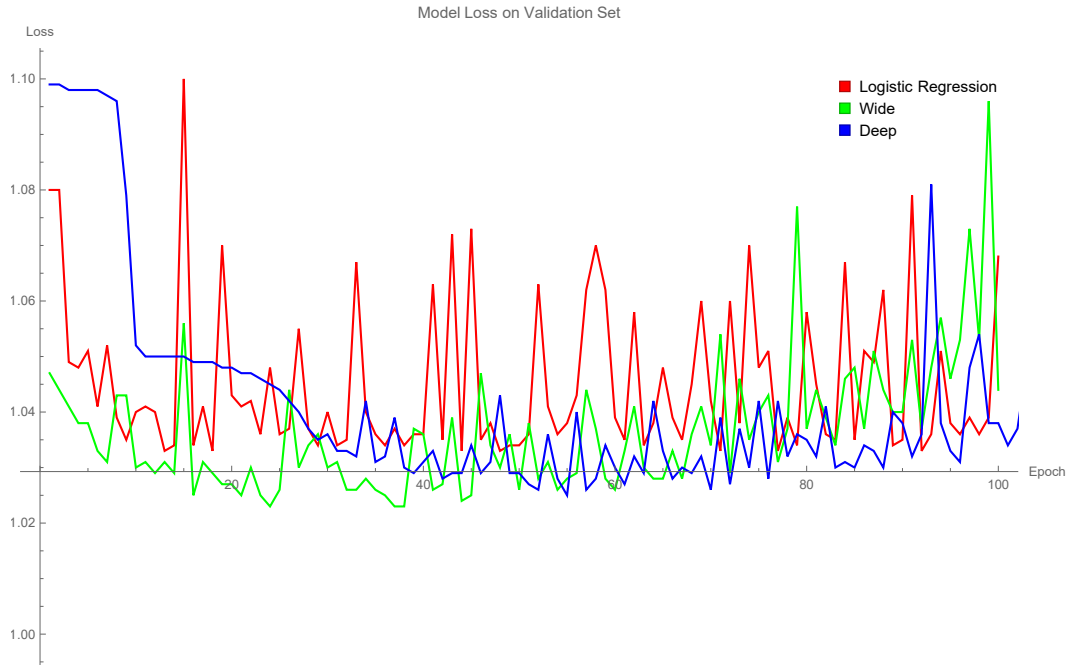


Figure 3: Performance of various iterations of the win rate model

Earlier in the process we tested a neural network on biased testing sets to see how they would generalize. This network had a test accuracy of 56.2% since it was trained on a smaller dataset with a batch size of 1. The average win-rate of a user of 17Lands is $\sim 53\%$ which is above the global average of 50% (due to the nature of Magic being a 1v1 game) as a result of the self-selecting nature of the user base of 17Lands. With this in mind we used samples exclusively from the two extremes to get biased sets: drafts done by players with $> 61\%$ win-rate and drafts done by players with $< 49\%$ win-rate. The results that we found from testing the two models on these disparate sets was: the first model successfully identified the card chosen 36.46% of the time for the good-players set and 35.26% of the time for the worse players. The decrease in accuracy compared to the unbiased tests were to be expected since they were both trained on the full spectrum of win-rate data, so they would be best at identifying the results mirroring the average user. The result of this test is interesting since both sets were $> 600,000$ rows long, and there was a $\sim 1.5\%$ improvement on the good-players set which appears to be a statistically significant result. This could imply that the pick priority of our model aligns closer with with the decisions of the better players which speaks to it's success despite being a fairly naive approach.

Shown in Figure 3 is the progression of different architectures of the win rate model as it gets trained. Despite the domain knowledge we gave it to make informed decisions, it was not able to break 41.4%. This number, while low is more reasonable than it may seem since looking at a deck and guessing its win rate is an incredibly difficult proposition for a human to do, something we proved to ourselves by seeing if we could guess it better than the model (in the 10 attempts we made, we were not able to guess a closer value). This is due to the massive amounts of variance that exist within the game play of Magic coupled

Figure 4: Deck built from pool that a bot drafted with 7 other bots

with the large factor the skill of the person playing plays in the result. A deck that looks great could get no wins or a deck that looks terrible could win a couple games.

5.2 Qualitative Results

As soon as we had a model which was working reasonably well, we set out to what a draft bot with $> 65\%$ accuracy would look like when actually drafting. The first way we tested this was by having it suggest every pick in a actual draft on Magic: the Gathering Arena against the in-game bots (this manual data-entry system of supplying data to our model was too slow to test it against human drafters). The logs for this draft are transcribed *here*. From a domain knowledge standpoint, the picks were mostly reasonable with the main outliers from choices we would make being taking cards of high rarity over cards which would be more potent in the deck (a phenomenon known as "raredrafting", which many humans do too). To demonstrate the validity of the deck, one of us built the deck from the pool the bot generated and played several games with it - winning 6 out of 9 of those games.

Our other way of testing this was to simulate a draft with only 8 copies of our bots to see if the resulting decks look reasonable. Shown in figure 4. This draft found an open color, drafted a string sample of cards which were reasonably distributed across the various "mana values" (how many resources are required to play a given card). The rest of the decks created from this draft can be seen *here*.

We later ran the less successful win rate model through the same draft simulation to see if it would also create human looking card pools. It managed to skew the picks towards a color or combinations, but it did not have the same cohesion that the neural network was able to attain.

6. Ablation Study

In order to test our model's robustness, and identify the important we features, we performed a brief ablation study. We performed two modifications to our well-performing model that tries to imitate human picks. We wanted to test the importance of knowing which the

pack number and pick number, so we retrained the model without this information. This alternative model achieved a 63.1% test accuracy, which is slightly worse (by 2 percentage points) than our primary model. This small change indicates a relatively low importance on pack number and pick number which is unsurprising given that this knowledge could be inferred from the number of cards in your pool at a given time.

An additional experiment we performed was restricting our dataset to half of the training data. We do this to identify whether or not our model would likely be improved by collecting more data, or if we have reached the limit of what our input features allow us to know. This alternative model achieved a testing accuracy of 65.6%, slightly outperforming our primary model. This slight deviation indicates that the model has received sufficient training data and that any future gains would need to be brought about by introducing different information.

7. Discussion and Conclusion

7.1 Conclusion

Overall, the models outperformed our expectations resulting in bots which were able to draft decks which quantitatively resembled choices humans made $> 65\%$ of the time and qualitatively looked like decks that real people would draft. Although we were unsuccessful in fully implementing domain knowledge, the challenges we faced while attempting to train win rate and deck building models showed the complexity of the problem we chose to tackle.

7.2 Next Steps

We have many ideas for how to move forward from here, mainly trying to make use of the less successful win rate and deck building models as a way of weighting the naive, human mimicking neural net. We also believe that an implementation of k-means clustering to help represent the synergistic combinations of cards known as archetypes that exist within a draft format. Which, if we can model representationally, should improve our chances of making a working win-rate predictor. If the win rate and deck building models were to get to a point where we could deem them successful, an ambitious path forward would be to simulate game play of the decks created against each other with the intent of improving the draft model.

Although we are satisfied with the promising results we have attained so far, there are clearly several avenues forward wit hen analysis of a game system as complex as this.

References

Sverre Johann Bjørke and Knut Aron Fludal. Deckbuilding in magic: The gathering using a genetic algorithm. Master's thesis, NTNU, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Curran Associates Inc., Red Hook, NY, USA, 2019.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12(null):2825–2830, nov 2011. ISSN 1532-4435.

Henry N Ward, Bobby Mills, Daniel J Brooks, Dan Troha, and Arseny S Khakhalin. Ai solutions for drafting in magic: the gathering. In *2021 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2021.