CS230

Item Prediction in *Dota 2*

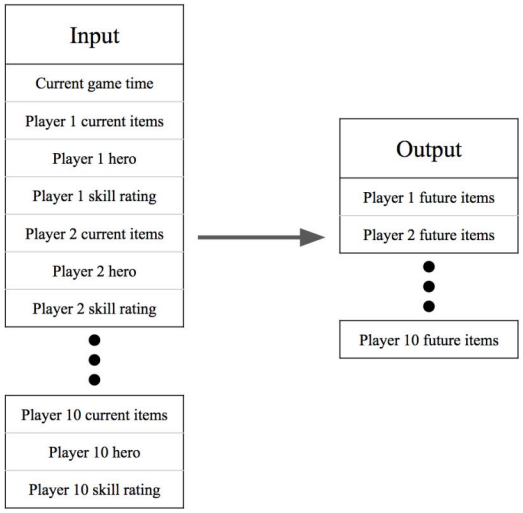
Jack Scott Department of Computer Science
Stanford University
jackk314@stanford.edu

Abstract

I present a fully connected neural network with two hidden layers that predicts which items players will purchase in the popular competitive online game *Dota 2*, given the current state of the game. The model is trained off of information about 49,867 amateur *Dota* matches.

1 Introduction

Dota 2 is a wildly popular online five-on-five video game with a rich competitive scene. In every game of *Dota*, players steadily accumulate "gold" over time from a number of sources, which can then be spent on "items" that make their characters stronger as the game progresses. Items are just as important as skill and strategy in determining the winner of the many battles that occur throughout each game. This makes choosing the right items a crucial part of *Dota*, and a difficult one, too: there are well over a hundred items to choose from, and different combinations can have different advantages for any given situation. At high levels of play, players must predict which items their opponents will buy and formulate their strategy accordingly. My algorithm seeks to assist in the latter task: the input is the current items each player has, which character (called "heros" in *Dota*) they have chosen to play, the current elapsed game time, and a skill rating of each player, and the output is the probabilities of each player buying each item by the end of the game.



I've chosen to only feed into the algorithm information about the game that would be available to a player during a normal game. This means that an interface using this network could give real-time

in-game predictions to a user without needing to consult extra information besides what is already given to the player by the game. For example, the algorithm should not be able to use information about exactly how much gold each opponent has, since this information is hidden in-game.

2 Related work

Because of the game's notorious difficulty and complexity, *Dota* has received lots of attention from machine learning researchers. The most common objectives in these papers are predicting the outcome of matches based on information available at the beginning of the game, and recommending hero choices during the character selection phase of the game. Just in past quarters of Stanford's machine learning classes alone, there have been two algorithms for prediction match outcomes [1,2], and one for hero recommendations [3], and outside of Stanford people have written numerous papers on the same subjects [4, 5, 6]. However, the biggest machine learning project for Dota 2 comes from Valve themselves, the creators of the game. Two months ago, Valve released the "Dota Plus" which is a premium subscription service that, among many other features, uses machine learning to suggest item purchases throughout the game [7]. While it may seem like a seriously unfair advantage to have access to a machine learning algorithm that can help make such important decisions, at high levels of play, the competitive edge that Dota Plus and all the other machine learning tools that provide similar services give is negligible. They are useful mostly for teaching newer player what the standard choices are for any given situation, but will very rarely give novel suggestions that an experienced player cannot come to on their own. Factors like the user's familiarity with an item and whether they understand why a certain item is optimal are often more important than knowing which item is "best" for the situation.

This is why I propose a different approach to all of these tools. Instead of trying to suggest which item the *user* should buy to help them win, my algorithm predicts items the *opponent* will buy. This offers a number of advantages. Firstly, this gives the user the power in deciding what to do given this information, rather than just telling them what decisions to make. This approach also avoids any issues of credit assignment – it's hard to tell how much a single item purchase by one of ten players actually affected the game's end result, but if we are just predicting opponent purchases, there is no such subtlety, since the opponent either bought a item or they didn't. This tool should also continue to be useful in higher skill levels, since it can be difficult even for very high level players to keep track of the decisions that five opposing players might make, as opposed to the relative ease of focusing on one's own item trajectory.

3 Dataset and Features

The dataset came from information about 50,000 public *Dota* matches provided by user *devinanzelmo* on Kaggle [8]. I first eliminated matches shorter than 15 minutes: a *Dota* game lasts anywhere between about 20 and 90 minutes, so matches less than 15 minutes long were probably ended short by one or more players leaving the game at the beginning, thus making data from the match useless – this left me with 49,867 matches. Then, for each match, I chose a random point in its duration to extract features from. A single training example contains information about the match at that randomly chosen point, and the label is the set of items that each player ended up buying after that point in the game. I used 8 random points for each match, so the total number of training examples was $49,867 \cdot 8 = 398,936$, which I split into different sets like so:

Train	Dev	Test
359,042	19,946	19,946

One training example from the dataset is given below:

Input:

Player	Hero	Current Items	Skill Rating
1	Undying	Arcane Boots	25.21
2	Arc Warden	Maelstrom, Boots of Travel	25.45
⋮	⋮	⋮	⋮
10	Techies	Soul Ring, Tranquil Boots	24.73

Label:

Player	Future Items
1	*None*
2	Mjollnir, Orchid Malevolence, Nullifier
⋮	⋮
10	Aghanim's Scepter

4 Methods

The final network itself is a simple fully-connected neural network with two hidden layers. There are 95 possible items each player could buy and 10 players in each game, so the network calculates 950 different probabilities for each training example. A fitting loss function is then the average of the binary crossentropy loss over all 950 predictions:

$$\mathcal{L} = \frac{1}{950} \sum_{i=1}^{950} -(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$$

For activation functions the algorithm uses Leaky ReLU with the following formula:

$$f(x) = 0.1x \text{ if } x < 0, \text{ otherwise } x$$

This loss function was minimized using Adam optimization [9] for batch gradient descent. A sigmoid activation is used for the final layer in order to get predictions between 0 and 1.

Various regularization techniques were implemented: batch normalization between fully connection layers, dropout with probability 0.1, and ℓ_1 regularization with a weight of 5×10^{-7}.

One reason for such a simple network is scalability: while this project only used $49,867$ matches, a commercial implementation of a tool using the network could take advantage of the huge amount of publicly available match data for *Dota* matches, like the "data dump" provided by *OpenDota*, which contains data for almost 1.2 *billion* matches spanning five years [8]. Another advantage of a quickly trainable network comes from the dynamic nature of the game itself. *Dota* receives balances changes to its characters several times each month, which can drastically change the decisions that players make, meaning that any neural network must keep up to date by training with data from recent games. Even outside of actual changes to in-game values, *Dota*'s metagame is in constant flux, as attitude to certain strategies shift and new discoveries are made.

I briefly investigated using a Recurrent Neural Network in order to capture the notion of sequences of item purchases through the game. However, results were lackluster. This makes sense, because intuitively it shouldn't matter the order in which each player bought their current items: the decisions players make after a certain point in time should rely only on the current state of the game, rather than its history.

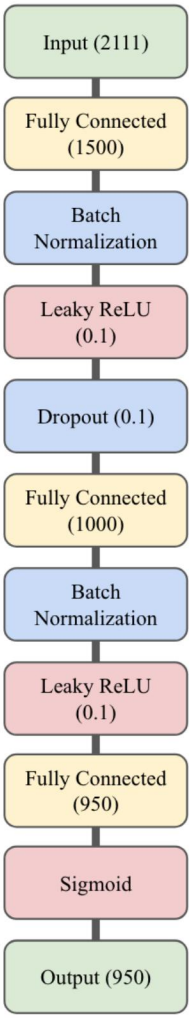


Figure 1: Structure of the network

5 Experiments/Results/Discussion

The primary metric I used during testing was AUC, which is the area under the curve defined by the values of false positive rate and true positive rate for varying output thresholds. This is most fitting because the probability that a player will buy one specific item out of 95 is usually quite low, even if some items are much more likely that others in a given situation. Thus, we care more about the relative confidence of the algorithm rather that its accuracy given a specific cutoff value.

Below are some of the experiments for various hyperparameters.

- **Batch Size**
  The gradient descent uses a batch size of 1024, which was chosen by increasing the batch

size until the GPU reached maximum utilization according to the `nvidia-smi` command, ensuring that maximum parallelism is achieved for quick training.

- **Hidden Layer 1 Size**
  To test different sizes for the first layer, I trained for 300 epochs to ensure that larger networks would have time to train the greater number of weights. For these experiments, I used a Leaky ReLU with 0.01 leakiness, batch normalization, 0.1 dropout, and a second hidden layer size of 1000.

Layer 1 Size	1000	1250	**1500**	1750	2000
Train AUC	0.895	0.900	**0.909**	0.911	0.912
Dev AUC	0.848	0.848	**0.854**	0.849	0.852

  A layer size of 1500 seemed to strike the best balanced between bias, variance, and training time.

- **Leaky ReLU Leakiness**
  After choosing the hidden layer sizes, I used the same parameters but with changing leakiness.

Leakiness	0	0.01	0.02	0.05	**0.1**	0.2
Train AUC	0.902	0.909	0.908	0.910	**0.913**	0.901
Dev AUC	0.848	0.854	0.855	0.855	**0.858**	0.856

  There's not much difference between the various leakiness paramaters, as long as leakiness is positive, but a value of 0.1 gave the best AUC for the dev set.

- ℓ_1 **normalization weight**
  The model was clearly overfitting, even with batch norm and dropout, so I introduced ℓ_1 normalization to reduce variance.

Weight	10^{-7}	5×10^{-7}	10^{-6}	5×10^{-6}
Train AUC	0.912	**0.904**	0.900	0.885
Dev AUC	0.870	**0.885**	0.883	0.875

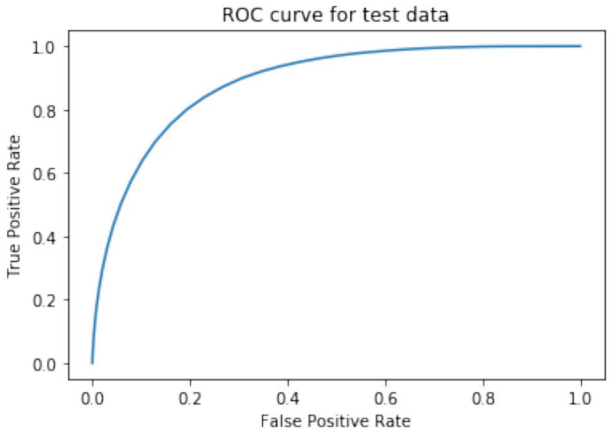
The final hyperparameters are:

- Layer sizes: 1500, 1000
- Batch Size: 1024
- ReLU Leakiness: 0.1
- Dropout Rate: 0.1
- ℓ_1 normalization weight: 5×10^{-7}
- Learning Rate: 0.01

And the final algorithm's performance was:

Train AUC	0.9037
Dev AUC	0.8854
Test AUC	0.8859

The ROC curve on the test set for varying thresholds:



6 Conclusion/Future Work

These results were actually much better than I had expected. The ROC curve might not seem very impressive; we can only achieve a true positive rate of about 60% at a false positive rate 10%, but the problem's difficulty may mean that this is fairly close to the optimal Bayes error. Players may make sub-optimal purchases, and there may be many seemingly viable item choices for a given situation, so the model's performance is quite good given the amount of randomness introduced by the human element. It takes years for players to develop an intuition for which items the opponents will buy, so this network would actually be fairly useful for newer players. Given more data, this network could let players stay one step ahead of their opponents by accurately determining what the enemy will buy, allowing the user to counter their strategies.

Besides training off more matches, some other features of the game state could perhaps be implemented to boost performance. The two most importanat ones would probably be the "hero level" of each player, which can help determine the relative strengths of each player and thus which items they are likely to buy, and the status of each team's structures (structures are important immobile units that each team must defend).

The network could also potentially be used in transfer learning for other objectives, like recommending which items to purchase, and predicting the match outcome during the game.

7 Contributions

Solo project.

References

[1] Petra Grutzik, Joe Higgins,Long Tran. *Predicting outcomes of professional DotA 2 matches*. Dec 16 2017.

[2] Kuangyan Song, Tianyi Zhang, Chao Ma. *Predicting the winning side of DotA*2. 2017.

[3] Kevin Conley, Daniel Perry. *How Does He Saw Me? A Recommendation Engine for Picking Heroes in Dota 2*. 2017.

[4] Kaushik Kalyanaraman. *To win or not to win? A prediction model to determine the outcome of a DotA2 match*. 2016.

[5] Filip Johansson, Jesper Wikström. *Result Prediction by Mining Replays in Dota 2*. 2016.

[6] Nicholas Kinkade, Kyung yul Kevin Lim. *DOTA 2 Win Prediction*. 2015.

[7] https://www.dota2.com/plus. 2018.

[8] Devin Anzelmo. https://www.kaggle.com/devinanzelmo/dota-2-matches/data. 2016.

[9] Diederik P. Kingma, Jimmy Ba. *Adam: A Method for Stochastic Optimization*. ICLR 2015. Dec 22 2014.

Libraries used:

- keras.io
- tensorflow.org
- scikit-learn.org
- pandas.pydata.org
- matplotlib.org
- docs.python-requests.org