682 Project Milestone Report

Connor Basich
cbasich@cs.umass.edu

Ben Glickenhaus
bglickenhaus@umass.edu

1. Introduction

Deep reinforcement learning (DRL) agents have had tremendous success in recent years in game-playing domains such as Go [5], Starcraft 2 [7], DoTA2 [3], and more. A primary reason for this success has been the ability to combine the power of deep neural networks for approximating value functions, with highly efficient search-based planning techniques such as UCT [2]. In this project, we investigate a separate component present in many popular games: 'drafting'. While this framework is most notably seen in popular games like DoTA2, Magic the Gathering, fantasy sports, and others, we propose that 'drafting' is a specific instance of a more general game we call **adversarial resource allocation games**. This type of game can extrapolate to problems faced in other domains such as politics, venture capitalism, military operations, and more.

Specifically, in this project, we aim to formulate the 'drafting phase' of DoTA2 as an adversarial resource allocation game, and train a deep reinforcement learning agent to play this game using a combination of simulated bot gameplay and an extensive data of real game drafts with their results from human players. By leveraging a combination of DRL, planning, and clustering techniques, we are able to produce a drafting agent that wins **64 percent** percent of the time against **a random baseline**.

2. Problem Statement

Although we don't formalize it here due to space constraints, an adversarial resource allocation game generalizes stackelberg games. ARAG are sequential games in which each player takes turns allocating a portion of a finite pool of resources to assets. Unlike stackelberg games, in ARAG, (mixed) strategies are *not* set before the game begins by either party, and notably both players have a unique advantage - one player goes first, and the other goes last.

In this project, we specifically investigate the DoTA2 drafting phase. The DoTA2 drafting phase is comprised of two components - selecting heroes and banning heroes; selecting a hero adds the hero permanently to that team, and banning a hero prevents either team from selecting them. In this game, each team represents a player, where each move that an agent can make is to allocate (unit) resources to select a hero for their own team, or to ban the hero from further play (i.e. remove an action from the action space for both parties).

The payoff received by each agent is the probability that their team, after the draft is complete, will win; in other words, the objective of *the drafting agent* is to maximize the probability of their team composition winning against the enemy team composition. Of course, this payoff is directly determined by the agent's evaluation function of teams, which initially may be completely random. Through a combination of pretraining and simulated self-play, we train a DRL drafting agent to produce good drafts. In order to isolate the effect of the draft itself, we use identical bot agents for all heroes on both teams for the actual game simulation. We use a +1/-1 reward signal as is standard in most RL domains for training the drafting agent and learning an accurate evaluation function.

3. Technical Approach

3.1. Neural Network Architecture

Recently, Transformers [6] have had substantial success in solving tasks dependent on sequence processing, most notably outperforming traditional recurrent architectures in Natural Language Processing tasks. More recently, Parisotto et al explored the performance of Transformer-based RL agents in memory dependent domains [4]. They show that Transformers are able to match or outperform LSTMs in a variety of these domains, including more reactive tasks where memory is less important. In this project, we aim to further the body of work of understanding Transformer performance in RL tasks. However, instead of an explicitly memory dependent task, we leverage Transformers in a task whose state is inherently sequential. In this sense, the Transformer's role is more similar to the traditional NLP task, where modelling position-wise interactions of the state sequence allows the agent to construct a better "sentence" of actions.

We implement a Transformer Encoder architecture similar to BERT [1] with Positional Encoding [6]. Our sentence length is fixed to 25 to represent each pick in a DoTA 2 draft, along with 3 extra positions for special CLS (class) and SEP (separate) tokens. The CLS token allows us to implement several additional prediction heads, which take the encoded representation from the CLS index, that we use for supplemental pre-training tasks and to predict the winner of the game. The latter task is subsequently used as the value head of the RL agent, so its performance is critical for providing the agent with a good signal in order to learn an effective policy. We observe that the value head is able to predict with high confidence after each player has taken 3 or 4 turns whether it will win or lose.

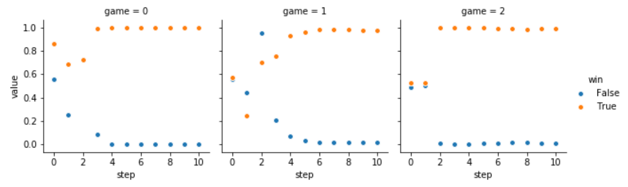

Figure 1. Predicted win probabilities by each agent from 3 randomly selected self-play games. Clearly, each agent is able to predict with high confidence after 4 turns whether it will win or lose

Notably, our Transformer-RL implementation does not make the same modifications made in [4]. We believe that our ability to pre-train using real world data effectively accomplishes the same stability improvements as their architecture modifications. This hypothesis may extend to self-play based learning techniques in general, since they effectively generate their own pre-train data. We discuss the justification for this hypothesis in the pre-train section. In future work, we would like to test this hypothesis.

3.2. Clustering

To further improve the performance of our system, we leverage the fact that there are strong interactions between DoTA 2 heroes - some heroes synergize well when on the same team, while other heroes specifically counter a hero on the opponent team. For example “IO” and “Abaddon” have abilities that greatly increase the other’s power, a fact one professional team exploited on their way to winning the largest international DoTA 2 tournament, TI, last year. Because of the interplay between heroes, drafting becomes a highly complex game of cat and mouse between the two teams - both working to achieve their goals while also countering the goals of their opponent.

To exploit these inter-hero relationships, we perform a clustering pre-processing step on the underlying attributes of one hundred thousand team compositions in the database to produce a set of clusters that represent common synergies of heroes. Since these compositions are taken from high-ELO play (i.e. games played by ‘top’ players), we argue that this represents a good proxy for ‘good’ synergies. This approach allows the model to explicitly reason about the opponent’s goals when taking an action.

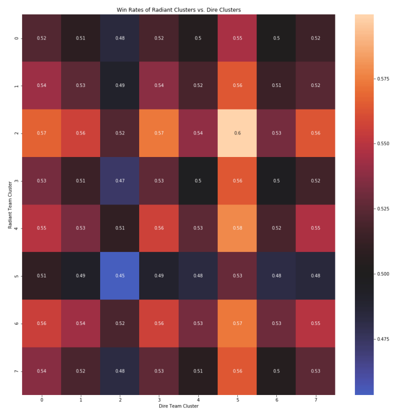

Figure 2. Win rates of different cluster matchups (read as column vs width). Here we see clear examples of the synergies discussed above. For example, cluster 2 performs extremely well versus cluster 5.

After trying several different clustering methods with various parameter selections, we settled on using KMeans clustering with eight centroids. Using these clusters, we can then train a separate component of our neural network to predict the likelihood of each cluster given a partial draft. The hidden layer output of this component is implemented as a residual block of the policy head, enabling the drafting agent to adjust its policy for the opponent’s policy. In other words, knowing the most likely composition the opponent agent is trying to play may change the relative values of different heroes, allowing our drafting agent to proactively pick and ban heroes that may have otherwise not been considered.

3.3. Planning Algorithm Selection

A fundamental component of successful game-playing agents is the planning algorithm that they use. These algorithms allow the agent to leverage the knowledge gained from their neural networks to make intelligent moves in the game. In the case of most games, the planning algorithms of choice are ‘search-based’ algorithms: algorithms which span a subset of the game-tree and select the action which maximizes some objective function on the space searched.

The classic example of search-based algorithms is the **minimax algorithm**, which in zero-sum games aims to play the move that minimizes the best value the opponent receives. While variants of this algorithm have been successful in games like chess, they have fared poorly in less symmetric games with larger spaces like Go.

A second popular search algorithm is monte-carlo tree search, which search a portion of the game tree and performs monte-carlo backups to update the heuristic value of a move for the current state. A particular variant of MCTS is the algorithm **UCT** (Upper Confidence Bounds applied to Trees) which combines classic MCTS with the UCB1 exploration/exploitation policy; this is the search algorithm that we implemented in our system for the action selection portion of the drafting game.

3.4. DoTA Simulator

We received generous help from the maintainers of the *Dotaservice* GitHub Repo, Tim Zaman and Nosetrademous. They have developed a tool for running simulated DoTA bot games in Docker containers. With some modification, this tool allows us to simulate the outcomes of the drafts to train and evaluate our drafting agent. While running DoTA in Docker reduces significant graphical overhead, this is unfortunately still a computationally expensive and slow process. On our personal computers, we can run 4 games in parallel. A batch of 4 games in 4 jobs takes approximately 8-10 minutes.

4. Preliminary Experiments

4.1. Methodology

4.1.1 Data

The two most commonly played DoTA 2 draft formats are “all-pick” and “captain’s mode”. In all-pick, players may nominate characters to be banned, and then teams take turns selecting characters. Captain’s mode, on the other hand, follows a much more structured process of waves of bans and picks. We collected two datasets of high MMR public DoTA 2 games to pre-train our model before running our RL procedure. The first, larger dataset consists of ≈ 5 million all-pick games. We also collected a much smaller dataset (≈ 55000) of captain’s mode games. Both datasets were collected using the official Steam API.

We designed our RL environment to mirror the captain’s mode format. The reason for this decision is two-fold. First, professional DoTA 2 is played using a captain’s mode format. Second, and maybe more importantly, the back and forth waves of bans and picks presents much more opportunity for interesting strategy and game-theoretic interaction among agents. We collected all-pick data as well because there is substantially more of it available, and it is still useful as an initial training domain for the model.

4.1.2 Pre-training

Before training in the RL domain, our model first goes through two pre-train steps - on both the all-pick and captain’s mode datasets. These pre-training steps are critical for allowing the model to learn a useful embedding of the domain before learning strategy to act within the domain. In this sense, pre-training accomplishes a very similar goal to the rearrangement of Layer Norm layers in Parisotto et al.

4.1.3 Training (Self-play)

The training routine for the RL agent is similar to AlphaZero [7]. The agent first plays a number of games against itself, to generate a memory dataset. The agent’s neural network is then updated by sampling from the generated self-play dataset. Finally, the newly trained agent competes against its former self. If the new agent’s win rate is greater than 55 percent, the agent’s neural network is replaced with the newly trained version.

4.2. Results

In this section, we report results for a number of experiments run. First, we establish a baseline by simulating the results of two randomly acting agents competing against each other. In this setting, we observe a 50 percent win rate for both agents, as expected. This experiments is useful to confirm that the environment is essentially random without intelligent choices, thus, any statistically significant improvement over random outcomes must be due to the learned policy of the agent.

We also evaluate our model during each step of the pre-train process. We find that training for a single epoch on the all-pick dataset allows the model to learn accurate hero predictions for a partial draft. Additionally, the model is able to consistently predict the final cluster of each team composition as well as the ultimate game winner.

	Random	**All Pick**	**Captains Mode**
Win Rate	.51	.64	.60

Table 1. Winrate over 200 games of random agent, pre-trained on all pick dataset, and pretrained on captain’s mode dataset.

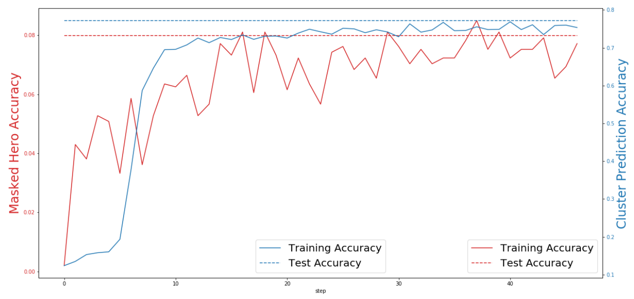

Figure 3. Train and Test prediction accuracies for the all-pick pretrain process

References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[2] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[3] OpenAI. Openai five. `https://blog.openai.com/openai-five/`.

[4] E. Parisotto, H. F. Song, J. W. Rae, R. Pascanu, C. Gulcehre, S. M. Jayakumar, M. Jaderberg, R. L. Kaufman, A. Clark, S. Noury, et al. Stabilizing transformers for reinforcement learning. *arXiv preprint arXiv:1910.06764*, 2019.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[7] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. M. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, et al. Alphastar: Mastering the real-time strategy game starcraft ii. *DeepMind Blog*, 2019.