League of Legends Ban/Pick Predictor

Shanchun Yi, Ramzi Dajani, Henry Leung

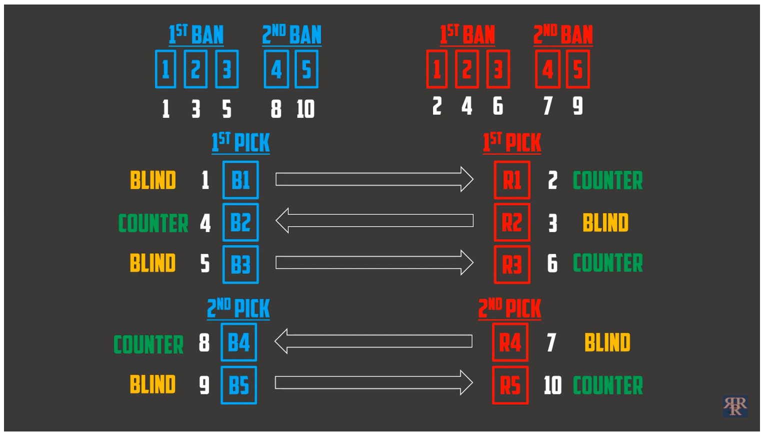
University of Toronto
Department of Computer Science
`{shanchun.yi, ramzi.dajani, henr.leung}@mail.utoronto.ca`

Abstract

In the present scene of competitive League of Legends, the skill levels of professional players are converging to a certain bottleneck. In other words, players are mostly equally skilled technically and mechanically. It might be tempting to elevate a team's win rate and its competitiveness by creating a model that can predict the seemingly best Ban/Pick option during the drafting phase. Since the drafting process can be abstracted to a zero-sum game, we trained a reinforcement learning model, specifically, a policy function that shall return the most optimal Pick/Ban. A simple neural network trained on data from the competitive scene will help infer the win rate of a match by the team composition and in turn construct the reward function for the reinforcement learning model.

1 Introduction

The goal of our project was to build a model that predicts ban/picks during a League of Legends champion drafting phase. Below is a diagram that illustrates the ban/pick phase for a competitive LoL game.

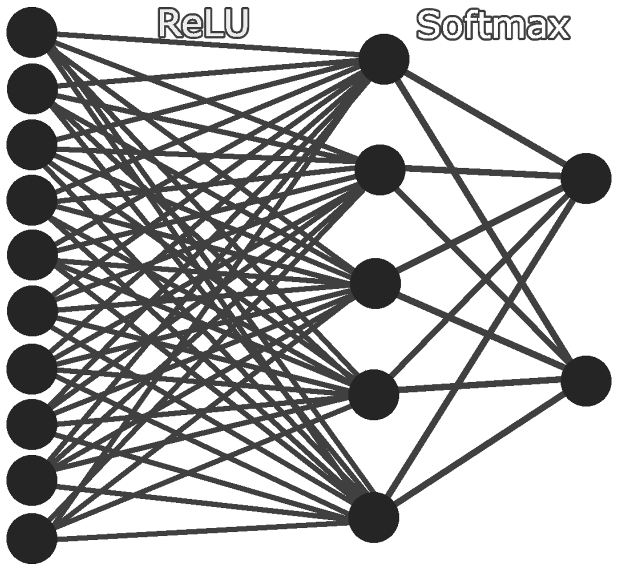


Essentially, teams alternate between bans and picks for two phases. For example, in our diagram the selection would go as following: 1st ban by Blue Team, 1st pick by Red Team, 2nd ban by Red Team, 2nd pick by red Team etc. If a champion is banned, then neither team can choose it. If a champion is picked by one team, then the other team cannot pick it. No repetitive Ban/Pick is allowed. Our model aims to make the best selection at each stage such that our final team composition has the best win rate. Further, the model can be easily generalized to other games where character selections are involved.

2 Methods and Algorithms

2.1 Win Rate Predictor

We implemented a neural network predictor, which when given a configuration of 10 champions should predict the win rates of both teams. The neural network is implemented using one hidden layer containing five units. The NN is implemented using PyTorch and is based off of data from professional tournaments. We use ReLU between the first and the second layer, followed by a Softmax function between the second and third layer. The hyperparameters of the NN include a learning rate of 0.005, 1000 epochs, and a batch size of 16. Note that the NN does not predict the optimal pick for each team at a given point, as that is up to the reinforcement learning algorithm.



2.2 Reinforcement Learning

The reinforcement learning algorithm we choose to implement is policy gradient. Policy gradient works given the Markov assumption, where all information is encapsulated in the current state. In other words, policy, rewards and transition are independent of the past states given the current state. Now we define policy, and reward function

$$\pi_\theta(a_t|s_t)$$
$$r(a_t|s_t)$$

which outputs the probability, reward respectively of the action a_t given state s_t at time t. A rollout can be defined as a playout of the game, $\tau = (s_0, a_0, s_1, a_1 ... a_{T-1}, s_{T-1})$ The probability of such rollout given the Markov assumption is

$$p(\tau) = p(s_0)\pi_\theta(a_0|s_0)...p(s_T|s_{T-1}, a_{T-1})\pi_\theta(a_T|s_T)$$

To optimize the policy function, we need to maximize the expected reward

$$\mathbb{E}_{p(\tau)}[r(\tau)]$$

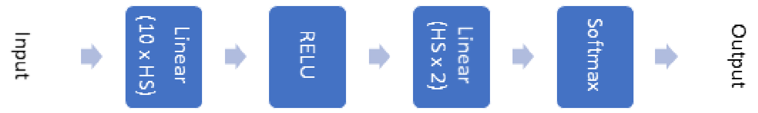
After some chunky math, we discovered that [1]

$$\frac{\partial}{\partial\theta}\mathbb{E}_{p(\tau)}[r(\tau)] = \mathbb{E}_{p(\tau)}\left(r(\tau)\sum_{t=0}^{T}\frac{\partial}{\partial\theta}\log\pi_\theta(a_t|s_t)\right)$$

To update the parameter iteratively, we can do the following algorithm [1]

```
Sample rollout $\tau = (s_0, a_0, s_1, a_1 ... a_{T-1}, s_{T-1})$;
$r(\tau) = \sum_{t=0}^{T} r(s_t|a_t)$;
for $t = 0, ..., T$ do
    $\theta \leftarrow \theta + \alpha r(\tau) \frac{\partial}{\partial \theta} \log \pi_\theta(a_t|s_t)$
end
```

To approximate the policy function, we use a simple neural network that is modelled below.



where SS is the `state_size`, AS is the `action_size` and HS is the `hidden_size`. To simply the definition of the environment, we chose to only let the model learning the picks. Consequently, each state will be represented by a vector of length 10 (5 per team). The action a state can perform is to pick a champion from the champion pool. Therefore, `action_size` is equal to the number of champions there are. This is also a result of a environment simplification, note that in real competitive setting, picks cannot be repeated.
The reward function is defined as followed, let

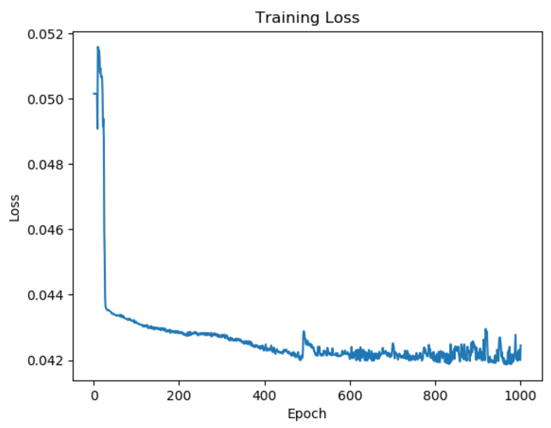
$$r_\tau = \begin{cases} 1 & \text{if WR\_Pred predicts team A victory} \\ 0 & \text{if WR\_Pred predicts team B victory} \end{cases}$$

where `WR_Pred` is the win rate predictor from the previous section.

3 Demonstration and Results

3.1 Win Rate Predictor

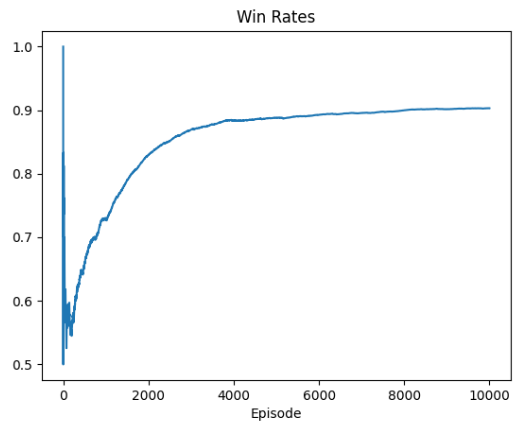
The loss used for this NN is cross entropy loss, and it's value over 1000 epochs is displayed below:



Using our data, the model is able to correctly predict the winning team about 55% of the time. It may seem only slightly higher than guessing, but that is the best that can be done considering the level of professional play seen. These players at the highest possible caliber, and with the fairness of the format of the ban/pick system, most games are merely a coin-flip at best.

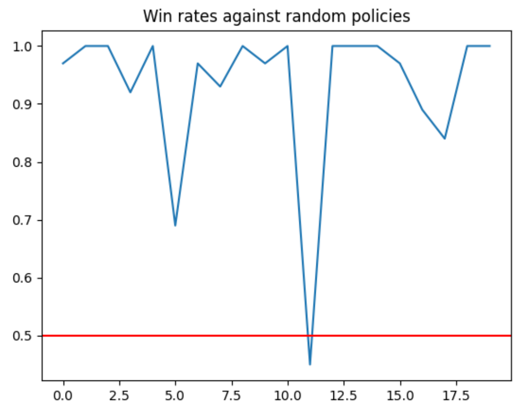
3.2 Reinforcement Learning

We initialize two Policies, `Policy_A` and `Policy_B`. The first experiment is to train `Policy_A` with respect to `Policy_B` that uses a random but static strategy. Below is the result,



`Policy_A` has learned the environment and the counter to `Policy_B` that in turns achieving an overall win rate around 0.9 at the end of the training. Note that the trained `Policy_A` might only have such a high win rate against `Policy_B`, since the counter to `Policy_B` might not work with other policies. The next goal is to train a policy function that has a high win rate in general, ie. a high win rate against any random policies.

Now, we will set up a adversarial self-play situation, where `Policy_A` optimizes while holding `Policy_B` constant, then `Policy_B` optimizes while holding `Policy_A` constant [2]. We can repeat the training for some iterations and hopefully obtain the invincible player. The trained policy is then contested by 20 players with distinct random strategies.



The results look rather promising, the trained player has over a 0.7 win rate against most of the random players. Only one random player scores a higher than 0.5 win rate against the trained player. We can now say we have created a very skilled drafting bot!

4 Discussion and Limitations

Given that the data is based off of professional tournaments, many champions in the game are not used due to their weakness, and so the NN will most likely over estimate the win rate for a team using an infrequently used champions, so it will perform poorly. As mentioned before, the players are playing at the highest caliber, and so most professional games are merely a coin-flip, not much can be done to accurately predict the winner.

Also, as more data is collected from future tournaments, we can add more matches so our win rate predictor can improve upon itself. It's also important to note that old data should be disregarded as certain champions get nerfed or buffed over different patches.

Since the policy gradient model assumes that the output of win rate predictor the absolute ground truth, the trained policy will only be effective with this particular win rate predictor. If the win rate predictor is not very accurate, the trained policy will not reflect the optimal play in the real world. The restriction on repetitive picks or picks on banned champions is omitted because policy gradient is poor at adjusting the probability of one specific action. If a reward is good/bad, policy gradient consider all actions in the sequence good/bad.

5 Conclusion

In conclusion, we achieved most of what we sought out to accomplish in our project proposal. The implementation of win rate predictor was rather successful. Although by the nature of the data, it does not produce a high accuracy. We only partially implemented the reinforcement learning environment, but the model performs surprisingly well.

6 GitHub Repo

https://github.com/xpbeyo/LoL-Draft-Predictor

References

[1] Jimmy Ba and Bo Wang. Generative models reinforcement learning. 2021.

[2] C. Gao, M. Müller, and R. Hayward. Adversarial policy gradient for alternating markov games. In *ICLR*, 2018.