# A Sketchy Report for Experiments on Google Research Football Project

Li Haoyu(1900012730@pku.edu.cn) Dong Xinran(1900013018@pku.edu.cn)
Class 5 of Design and Analysis of Algorithms, Peking University, 2021

## INTRODUCTION

Supervised, unsupervised and reinforcement learning (RL) are the three basic machine learning paradigms, which are widely used in the field of data mining, computer vision, natural language processing, bio-metrics and so on. RL differs from Supervised and unsupervised learning mainly because it neither needs labels to be presented in advance, nor sub-optimal actions to be explicitly corrected. According to these characteristics, RL is more studied in disciplines such as game theory and operations research, focusing on how intelligent agents ought to take actions in an environment to maximize the reward function. Concerning this thesis is only a report for a class project, we simply eliminated explanations of notions and related works, mainly taking account of our experiment results instead.

Our project is based on Google Research Football (GF). GF is a novel RL environment based on open-source game called Gameplay. Google Research creates GF for training their RL models and other more profound research purposes, and they even have held a GF contest on Kaggle months ago.

GF has many scenarios of distinct difficulties. For our project, we are asked to complement one of 3 scenarios below (Last modified on 19:48 BJT, May $24^{th}$, 2021):

(1)RL empty goal;

(2)Rule-based 3 vs 1 with keeper / RL penalty kick;

(3)RL 3 vs 1 with keeper.

Although methods to finish the 3 scenarios are totally different, we have to finish them one by one mainly because the difficulties of the 4 scenarios are sharply increased. We have reflected to our assistant that it is impossible to go straight to handle the $3^{rd}$ scenario unless we simply others codes on Kaggle because it takes even more than one day to check if our codes are correct (Because we can not use GPU sources, we perform all the experiments on a machine with 4 core CPUs. Unfortunately, the $3^{rd}$ scenario is still remained to be a spare choice after modifying. So the only way to overcome the $3^{rd}$ scenario for us is to confirm our model works on the $1^{st}$ scenario in advance).

We have learned basic algorithms Q-learning and Policy Gradient (PG) as fundamental knowledge and have studied improved algorithms such as Deep Q Network (DQN), Dueling Double DQN (D3QN), and Proximal Policy Optimization (PPO). We implemented all above algorithms for GF in pytorch by referring other open-source codes. We unable to completely borrow existing codes on Kaggle or official builtin examples mainly because they relies on other frames such as baseline and baseline3, but we point out the notes on Kaggle help us set up corresponding environments and give us inspiration to finish this project.

We conduct the $2^{nd}$ scenario (i.e. rule-based version) for inspirations and the baseline of this work. Because we are learning RL for the first time (Neither Introduction to AI nor this class has taught a systematic knowledge of RL), we wonder what a performance is a normal performance for RL. Therefore, we conducted the $2^{nd}$ one for comparation purpose. In general, rule-based version has more prior knowledge given by human, while RL version has a more sophisticated control when encountering different situations. Given enough time, RL version can also converge to a high level goal rate. However, for time constraints for training, rule-based version seems to be much better than a RL version. Therefore, we take one more step, trying to combine the advantage of the rule-based version and the RL version, adopting a rule-assisted version. We perform a few rough but deterministic operations at the beginning of the game. These operations use our human's prior knowledge and cannot said to be wrong under any random adjustment of environment. After these deterministic operations, we use RL algorithms to deal with various kinds of special cases. Experimental results show that rule-assisted algorithm significantly reduces the training time.

To summarize, we listed our research procedure below (Fig. 1):

(1) We implemented DQN, D3QN, PG, and PPO on the scenario RL Empty Goal Close, finding that D3QN and PPO outperforms DQN and PG.

(2) We eliminated DQN and PG, using D3QN and PPO to train the model on the scenario RL 3 vs 1 ith Keeper. For time constraints, we use the same parameters when we deal with scenario RL Empty Goal Close. Experiments are not that satisfying.

(3) We conducted a rule based version as comparation, finding that exploration by human is much faster and more effective than machine at the beginning of the game.

(4) We combine the advantage of the rule-based version and the RL version, adopting a rule-assisted RL solution.

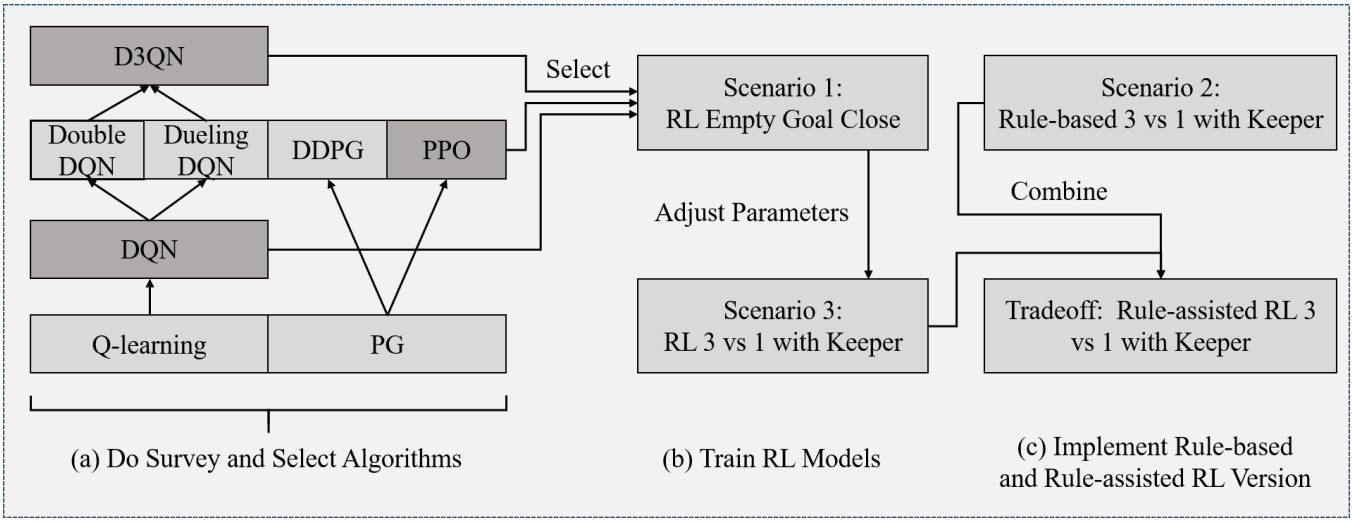


**Figure 1: Our Research Procedure**

Experiments show that it can converge more quickly with a higher goal rate.

We have released all of our codes at Github https://github.com/382308889/GFProject.

## ALGORITHM

In this part, we introduce PPO as an example.

Proximal policy optimization (PPO) is a new method based on policy gradient methods for reinforcement learning, which alternate between sampling data through interaction with the environment, and optimizing a "surrogate" objective function using stochastic gradient ascent. Whereas standard policy gradient methods perform one gradient update per data sample, we use PPO to enable multiple epochs of minibatch updates. PPO has some of the benefits of trust region policy optimization (TRPO), but they are much simpler to implement, more general, and have better sample complexity (empirically). Previous studies have shown that PPO outperforms other online policy gradient methods, and overall strikes a favorable balance between sample complexity, simplicity, and wall-time.

## Policy Gradient

The goal of reinforcement learning is to find an optimal behavior strategy for the agent to obtain optimal rewards. The policy gradient methods target at modeling and optimizing the policy directly. The policy is usually modeled with a parameterized function respect to $\theta$, $\pi_\theta(a|s)$. The value of the reward (objective) function depends on this policy and then various algorithms can be applied to optimize for the best reward. The reward function is defined as:

$$\nabla\overline{\mathcal{R}}_\theta = \mathbb{E}_{\tau\sim p_\theta(\tau)}[\mathcal{R}(\tau)\nabla\log p(\tau)].$$

## Proximal Policy Optimization

At the policy gradient methods, we use $\pi_\theta$ to collect data. When $\theta$ is updated, we have to sample training data again. Considered that $\theta'$ is fixed while using the sample from $\pi_{\theta'}$ to train $\theta$, we can reuse the sample data. So we have

$$\nabla\overline{\mathcal{R}}_\theta = \mathbb{E}_{\tau\sim p_{\theta'}(\tau)}[\frac{p_\theta(\tau)}{p_{\theta'}(\tau)}\mathcal{R}(\tau)\nabla\log p(\tau)].$$

So far, we've figured out the method will sample the data from $\theta'$ in each iteration and use the data to train $\theta$ many times. According to the formula $\mathbb{E}_{x\sim p}[f(x)] = \mathbb{E}_{x\sim q}[f(x)\frac{p(x)}{q(x)}]$, we can modify the expression:

$$\begin{aligned}\nabla\overline{\mathcal{R}}_\theta &= \mathbb{E}_{\tau\sim p_{\theta'}(\tau)}[\frac{p_\theta(\tau)}{p_{\theta'}(\tau)}\mathcal{R}(\tau)\nabla\log p(\tau)]\\ &= \mathbb{E}_{(s_t,a_t)\sim\pi_\theta}[A^\theta(s_t,a_t)\nabla\log p_\theta(a_t^n|s_t^n)]\\ &= \mathbb{E}_{(s_t,a_t)\sim\pi_{\theta'}}[\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}A^\theta(s_t,a_t)\nabla\log p_\theta(a_t^n|s_t^n)].\end{aligned}$$

Given a new sign $J^{\theta'}(\theta)$, we've got the expression:

$$J^{\theta'}(\theta) = \mathbb{E}_{(s_t,a_t)\sim\pi_{\theta'}}[\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}A^{\theta'}(s_t,a_t)].$$

After optimizing this expression, we can draw a new method named PPO. The specific steps of PPO are as follows:

- Initial policy parameters $\theta^0$.
- In each iteration:

(1) Using $\theta^k$ to interact with the environment to collect $\{s_t, a_t\}$ and compute advantage $A^{\theta^k}(s_t, a_t)$;

(2) Find $\theta$ optimizing $J_{PPO}(\theta)$ :

$$J^{\theta^k}_{PPO}(\theta) = J^{\theta^k}(\theta) - \beta KL(\theta, \theta^k),$$

$$J^{\theta^k}(\theta) \approx \sum_{(s_t,a_t)} \frac{p_{\theta(a_t|s_t)}}{p_{\theta^k}(a_t|s_t)} A^{\theta^k}(s_t, a_t);$$

(3) If $KL(\theta, \theta^k) > KL_{\max}$, increase $\beta$;

(4) If $KL(\theta, \theta^k) < KL_{\min}$, decrease $\beta$. Here is our pseudo-code:

**Algorithm 1: PPO, Actor-Critic Style.**

```
for iteration in {1, 2, ...} do
    for actor in [1..N] do
        Run Policy π_θold in environment for T
          timesteps.
        Compute advantage estimates Â_1, ..., Â_T
    end
    Optimize surrogate L wrt θ with K epochs and
      minibatch size M ⩽ NT.
    θ_old ← θ
end
```

Above all, we have introduced proximal policy optimization, a family of policy optimization methods that use multiple epochs of stochastic gradient ascent to perform each policy update. These methods have the stability and reliability of trust-region methods but are much simpler to implement, requiring only few lines of code change to a vanilla policy gradient implementation, applicable in more general settings and have better overall performance.

## PERFORMANCE EVALUATION

### Experimental Setup

In this section, we present our experimental settings and how to run our open-source codes on your own Linux machine.

•**Installation**: Now we assume you have installed git and python 3.6 or higher. Our project is based on GF, so you should choose your download directory and install required packages of GF (see offcial website https://github.com/google-research/football) at first. We are not responsible for its long and bumpy installation process, where everyone who wants to use GF cannot escape. Furthermore, pytorch, numpy are all used in our codes. For pytorch, you should run "pip3 install torch torchvision torchaudio". For numpy, you can simply run "python3 -m pip install numpy". To download our codes, run command "git clone https://github.com/382308889/GFProject.git". Then you can run "python3 (★)" to train the model. For example, you can run "python3 GFD3QN" for a D3QN version (GFPPO is slightly different, see README). If you want to run with render, you can simply set the *render* = *True*.

Note: *3 vs 1 with Keeper* is the default mode in our codes, so if you want to try other scenarios, you should modify our codes in a few steps below: (1) Save the given models in a safe place (or it will be covered after your training), then Open the source code with your editor. (2) Replace all "3 vs 1 with Keeper" to any scenario you want (such as "academy empty goal close"). (3) Save the file and run "python3 (★)" at last.

•**Computation Platform**: We perform all the experiments on a machine with 4 core CPUs (Intel (R) Core (TM) i7-10510U CPU @ 1.80GHz) and 8.0 MB DRAM memory. The CPU core has 256KB L1 cache, 1.0MB L2 cache, and 8MB L3 cache. For some constrains, we cannot apply GPUs to train our model.

•**Scenario Settings**: We perform our experiments on scenarios described in Section 1. For the *Empty Goal Close* scenario, we conduct a comprehensive analysis on its correctness vs training times with different parameter settings. For other scenarios, we only show our experimental performances of our trained or rule-based model.

### Implementation and Results

*1. Academy Empty Goal Close.* .

• Goal Rate vs. Training Rounds vs. Parameter $\gamma$ setting (PPO) (Fig. 2 (a))

We implement this scenario only to examine our codes and evaluate the effects of parameters. In this part, we focus on discount rate $\gamma$ as an example. We conduct PPO algorithm and find that when the parameter $\gamma \in [0, 0.9)$, the goal rate is very low. So then we adjust $\gamma$ to 0.9, 0.99, 0.999 and 1 to analyze the effect of $\gamma$. As shown in Fig. 2 (a), when $\gamma = 0.9$, the agent is so short signed that it cannot learn a long term strategy. While when we set $\Gamma = 1$, the agent may be also mislead by indeterministic future, and it is not recommended by RL tutorials. When $\gamma = 0.9$ or 0.99, however, the goal rate can quickly converged to a high value (almost 100%). We study referred papers *The Nuts and Bolts of Deep RL Research (John Schulman et al.) and Reinforcement Learning An Introduction (Richard S. Sutton et al.)* to find theoretical supports, finding $\gamma$ is recommended to be set as $(\frac{1}{10})^{\frac{1}{t}} \approx 1 - \frac{1}{t}$, where $t$ is the future steps that we consider. That's because the settlement reward must be greater than the daily reward, i.e. $t\hat{r} \geqslant Q_t = \sum_{i=1}^{t} r_i \gamma^i \approx \hat{r}\frac{1}{1-\gamma}$. $t = \frac{1}{1-\gamma}$ is called effective time horizon formula. Official example codes by GF also correspond this formula.

In the end, we choose $\gamma = 0.993$, learning rate as 0.0008 when we train a more difficult scenario. We use the same method on algorithms DQN and D3QN: After a long long and quite boring process of parameter adjustment, we choose appropriate parameter settings for these algorithms and the result is shown in next part.

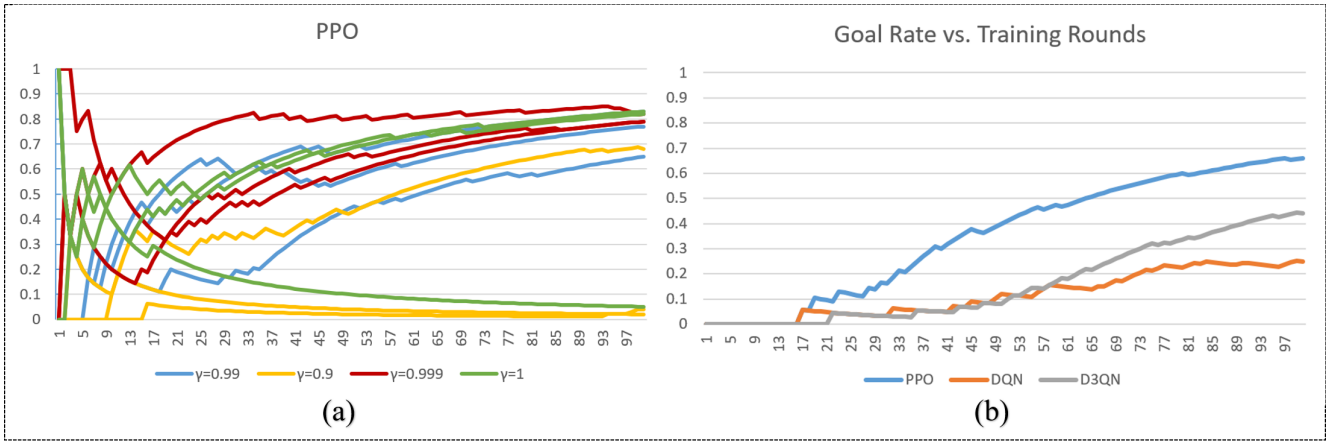


**Figure 2: Experiments on Scenario 1**

• Goal Rate vs Training Times vs Different RL Algorithms (Fig. 2 (b))

After we select the approximate optimal parameter settings of DQN and D3QN, we compare the training effects under algorithms DQN, D3QN, and PPO. We find that D3QN significantly outperforms DQN. So we decide to use D3QN (derived from Q-learning) and PPO (derived from PG) to conduct future experiments.

*2. RL 3 vs 1 with Keeper.* .

• Goal Rate vs Test Rounds (PPO) (Fig. 3)

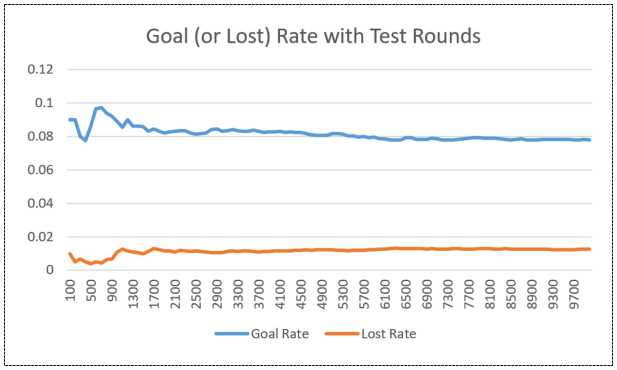


**Figure 3: Goal (or Lost) Rate of PPO**

After training 120,000 rounds, we find the goal rate under D3QN performs quite terrible (There is no difference from a random algorithm even we trained it all night! As a consequence, we don't paste our experimental result on D3QN.) Due to the lack of GPU resources, it costs at least one day to check if a modified parameter work. So we ultimately abandoned the idea to adjust parameters once and for all. We analyzed this strange phenomenon and guess that's because parameter fits *Empty Goal Close* doesn't necessarily mean it fits *3 vs 1 with Keeper*.

The process to train PPO is quite interesting. For the first several rounds of training, we select a CNN with 64 dimensions. However, experiments cannot be claimed satisfying after training: Our testing results show that the average goal rate is only about7.8%, the average lost rate is over 1.3%, and the average goal time is about 33.9 steps, see Fig. 3). We checked its performance in render mode, only to find the agent always learns two naive strategies: (1) Run to right until it meets the goal. (2) Pass the soccer to his teammates at very beginning and his teammate shoots immediately. (Fig. 6 (a) (b)).

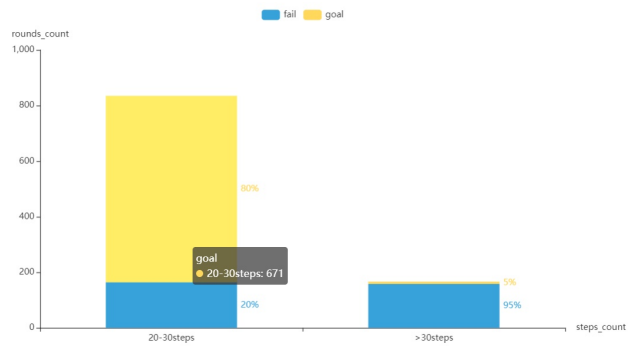


**Figure 4: Relations Between Round, Step and Goal**

We guess the reason why the agent chooses 2 strategy randomly is because the CNN dimension is not big enough to keep its choice stable. So as we keep training this scenario, we adopted our assistant's advice on the dimensions of CNN. After the dimension is set as 512, we find the the agent gradually trends to choose a deterministic strategy, i.e. pass the ball to his teammate first (In previous experiments the agent

always choose a distribution of the two strategy or even converge to the first strategy in the end). After training 100,000 rounds, the agent significantly outperforms our expectation. As is shown in Fig. 4, its goal rate has become 67.7% and the lost rate has become almost 0.002%. Besides the average goal time is around 26.9 steps with variance 2.54 according to our statistic analysis (see goalresult.csv at GitHub). To be more specific, 83.9% rounds of a game are finished in 20 to 30 steps, while 80% of them win this round of game. At the same time, if a round has prolonged to 30 steps, it probably (about 95% probability) cannot goal. Detailed analysis of distinct actions is shown in Fig. 5. However, it's still not clear if this performance is "well done" or just "fairly good" without comparation. So we implement the Rule-based 3 vs 1 with Keeper scenario as a comparation, which surprisingly improves our work in the end.

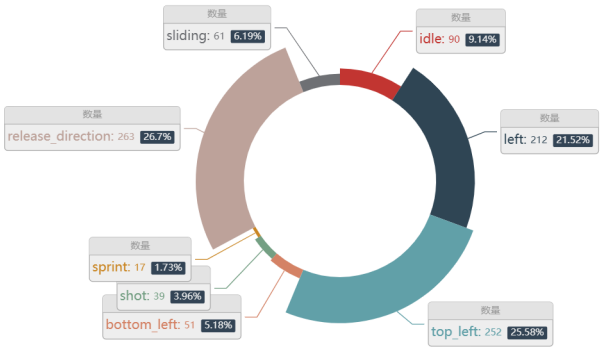


**Figure 5: Action Count**

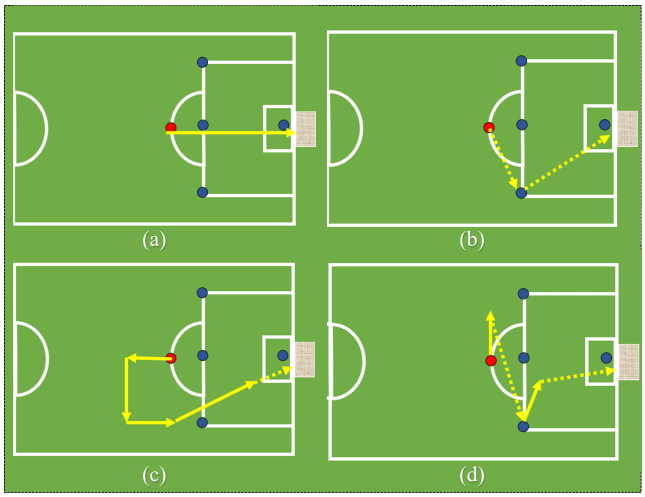


**Figure 6: Feasible Strategies**

*3. Rule-based 3 vs 1 with Keeper.* .

• Goal Rate vs Test Rounds (Rule-based) (Fig. 7)

We implemented a rule-based version of GF, finding that if the player, is in a same straight line with the anti-player and the keeper, it is relatively easier for keeper to defend. So in this scenario, we let the player run to the opposite direction to the anti-player at first. After our teammate get in the way of the anti-player, we let the player avoid him and then attack the goal. (Fig. 6 (c)) Rule-based algorithm performs a more than 85% goal rate, which is much higher than our RL algorithms (Fig. 7).

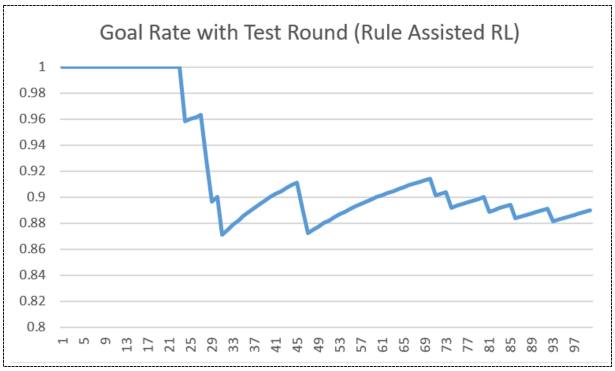


**Figure 7: Goal Rate of Rule-based Algorithm**

*4. Rule-assisted RL 3 vs 1 with Keeper.* .

• Goal Rate vs Test Rounds (Rule-assisted RL) (Fig. 8)

Inspired by the Rule-based scenario, we want to combine the advantage of the RL and rule-based algorithm. In general, original RL is easy to fall in local optimum and never tries a low-probability but better strategy. Considering that, we let our agent do some deterministic steps at first, then we train the model after these steps. In this way we manually reduce the action space at the beginning of the game. And we obtain a relative better result that significantly outperforms pure RL strategies. In this experiment, we let the agent do the following operations at the beginning: (1) Go top for $0 \sim 24$ steps. (2) The player pass the soccer to his teammate at the $25^{th}$ step. Further more, to speed up the training process, we disabled all operations that go towards left. Our experimental results are shown in Fig. 8. The goal rate of the Rule-assisted algorithm has become 80% in the end. In conclusion, we adopt a relative reasonable action at the very beginning of the game, while use RL to handle the remaining steps. It greatly helps to reduce the space to search for a pure RL algorithm, and slightly improves the goal rate.

## CONCLUSION

In this report, we implement algorithms DQN, D3QN, PPO on Google Football Environment and test their effects on scenarios *Empty Goal Close, Rule-based 3 vs 1 with Keeper*

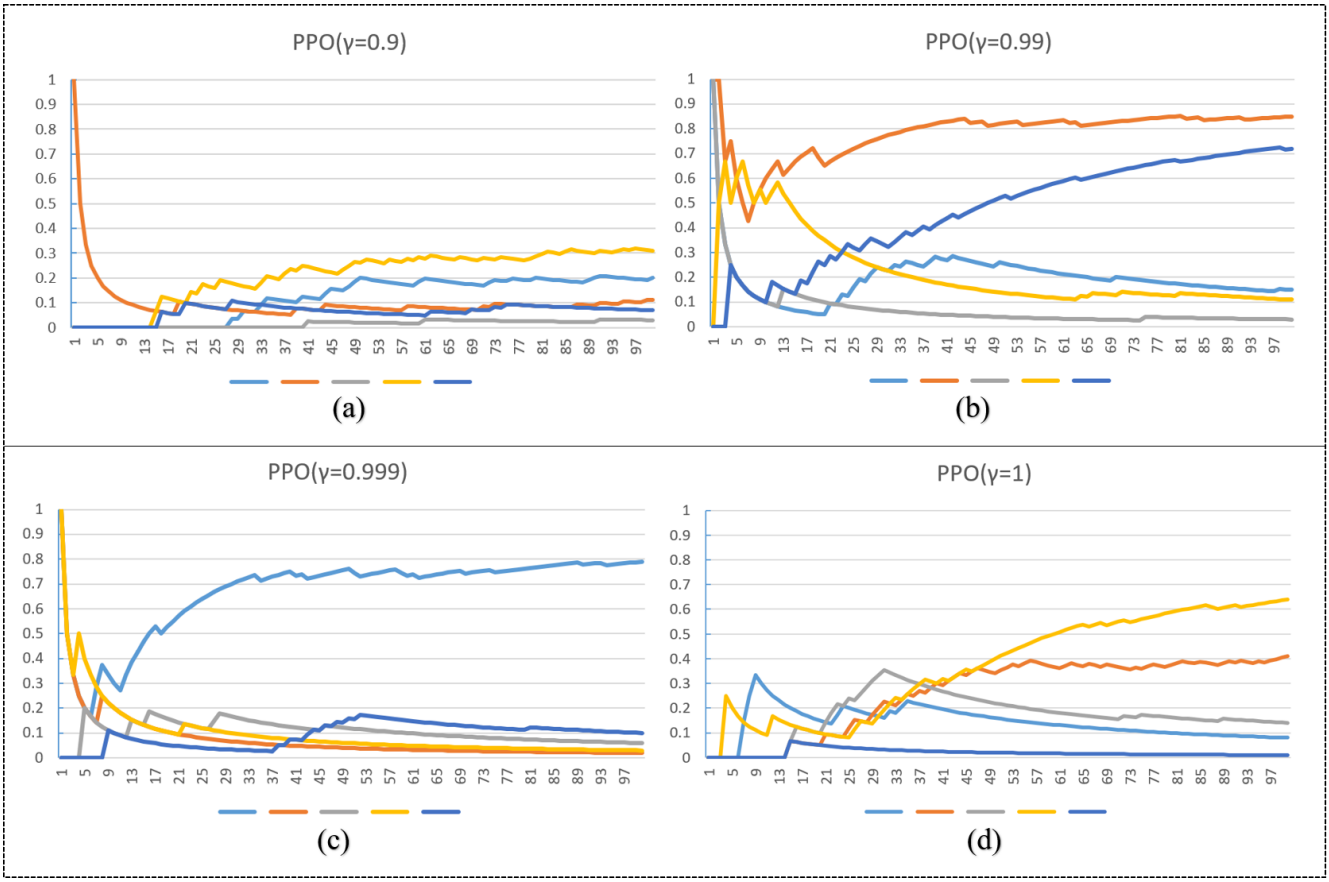


**Figure 8: Converge Speed with Different Discount Rate $\gamma$**

and *RL 3 vs 1 with Keeper*. We choose appropriate parameter settings according to both empirical and theoretical reasons and complete a Rule-assisted version in the end. Experimental results show that PPO outperforms DQN and D3QN to some extends on scenario *Empty Goal Close* and *RL 3 vs 1 with Keeper*. We have released our codes at https://github.com/382308889/GFProject.

## ACKNOWLEDGEMENT

First, we would like to show our gratitudes to our assistant Luo Hao. Without his kindness and patient instructions, we cannot complement our experiments in such a short period of time. We shall extend our thanks to Professor Lu Zongqing for his high-level guidance. Our sincere appreciation also goes to the Yan Hongyu, a student of Turing Class 19 whose research insteret is reinforcement learning, we thank him for taking part in our discussion and giving us practical advice. Last but not least, we have referred numerous relative existing codes on Github during our work, we will list them in our References.

## DIVISION OF LABOUR

Li Haoyu is responsible for the architecture of the project, finishing algorithms including DQN, D3QN, Rule-based, Rule-assisted Reinforcement Learning Algorithms and parts of PPO. The training tasks is mainly finished by Li, including last versions of models that released at GitHub. Li has also written all parts of the report except ALGORITHM, as well as doing experiments and drawing figures except Fig. 4, 5. Typesetting of the report is also finished by Li. Li is also responsible for the GitHub repository, and made all parts of the midterm report except the mathematical analysis part.

Dong Xinran investigates algorithms such as D3QN and PPO, giving us a theoretical support during this project. Notes and most of codes in GFPPO are also written by her. She finishes the ALGORITHM part of the report and gives a speech of the mathematical analysis at our midterm report. Furthermore, Dong takes part in our experiments and evaluates the performance of PPO and draws some relative pictures(i.e. Fig. 4, 5). Ultimate videos and README part at GitHub are also made by her.

## REFERENCE

.


[1]. https://zhuanlan.zhihu.com/p/128484325

[2]. https://zhuanlan.zhihu.com/p/34089913

[3]. Barhate, Nikhil. Minimal PyTorch Implementation of Proximal Policy Optimization. GitHub 2021. https://github.com/nikhilbarhate99/PPO-PyTorch

[4]. https://zhuanlan.zhihu.com/p/345353294

[5]. Google Research, Brain Team. Google Research Football: A Novel Reinforcement Learning Environment. AAAI'20.

[6]. Jiayi Weng, Huayu Chen, Alexis Duburcq, Kaichao You, Minghao Zhang, Dong Yan, Hang Su, Jun Zhu. Tianshou. GItHub 2020. https://github.com/thu-ml/tianshou

[7]. Volodymyr Mnih et al. Human-level control through deep reinforcement learning. https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf

[8]. Richard S. Sutton, David McAllester, Satinder Singh, Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation.https://papers.nips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf

[9]. John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov. Proximal Policy Optimization Algorithms. https://arxiv.org/pdf/1707.06347.pdf

[10]. https://www.kaggle.com/mlconsult/best-open-rules-bot-score-1020-7

[11]. https://github.com/ChintanTrivedi/rl-bot-football?source=post_page-----d53f9afffbf6----------------------

[12]. https://awesomeopensource.com/project/Khrylx/PyTorch-RL

[13]. https://spinningup.openai.com/en/latest/

[14]. https://github.com/nikhilbarhate99/PPO-PyTorch

[15]. https://github.com/TianhongDai/google-football-pytorch

[16]. https://awesomeopensource.com/project/TianhongDai/reinforcement-learning-algorithms

[17]. Hado van Hasselt, Arthur Guez, David Silver. Deep Reinforcement Learning with Double Q-learning. https://arxiv.org/pdf/1509.06461.pdf

[18]. Ziyu Wang, Tom Schaul SCHAUL, Matteo Hessel MTTHSS, Hado van Hasselt HADO, Marc Lanctot LANCTOT, Nando de Freitas. Dueling Network Architectures for Deep Reinforcement Learning. https://arxiv.org/pdf/1511.06581.pdf.

[19]. https://pythonhealthcare.org/2020/07/11/duelling-double-deep-q-learning-a-simple-hospital-model/

[20]. Ying Huang, GuoLiang Wei, YongXiong Wang. V-D D3QN: the Variant of Double Deep Q-Learning Network with Dueling Architecture. IEEE 2018.

[21]. https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html