Playing football game using AI agents

Koyel Datta Gupta
Department of Computer Science, Maharaja Surajmal Institute of Technology, Janakpuri, New Delhi, India - 110058
koyel.dg@msit.in

Nishthavan Dahiya
Department of Computer Science, Maharaja Surajmal Institute of Technology, Janakpuri, New Delhi, India - 110058
nishthavandahiya7@gmail.com

Mayank Dabas
Department of Computer Science, Maharaja Surajmal Institute of Technology, Janakpuri, New Delhi, India - 110058
mayankdabas2401@gmail.com

Pratish Pushparaj
Department of Computer Science, Maharaja Surajmal Institute of Technology, Janakpuri, New Delhi, India - 110058
pratish.pushparaj16@gmail.com

2022 International Conference on Emerging Techniques in Computational Intelligence (ICETCI) | 978-1-6654-7941-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICETCI55171.2022.9921374

***Abstract*—A lot of effort has been put into training AI agents to play games like chess, connect-4 etc where they excelled and taught us new ways to approach problems. Furtheron, recent progress in this field is accelerated by broadening horizons and taking on complex environments like GO. Football is also such a complex setting which requires the agent to learn intricate concepts like passing, shooting, dribbling etc and develop tactics to maximize chances of winning. And hence we take on the game of football using AI agents. We use the google reinforcement learning environment to train and evaluate our agents. We solve this problem of playing football by training two different agents namely Deep Q Networks and Light GBM, where Deep Q Network is a self-learning algorithm based on reinforcement learning and Light GBM is a supervised learning algorithm and dataset for this algorithm is extracted through kaggle.**

Keywords—Google research football, Artificial intelligence (AI), Reinforcement learning (RL), Deep learning (DL), Machine learning (ML), Deep Q Networks, LightGBM

I. INTRODUCTION

The main goal of training AI agents to play games is to learn the environment, overcome complicated tasks and maximize rewards which allows us to test our new ideas and algorithms in a secure manner. These algorithms further have real-world applications in self-driving cars, automation in industries, robotics etc. But, many deterministic and near deterministic games like Chess, Connect-4 etc may not allow us to explore the full potential of state-of-art reinforcement and machine learning algorithms whereas, games like football provides a very challenging environment for testing algorithms to their full potential.

Hence, we take the game of football that is played among two opposite teams each having 11 players and a spherical ball and the team scoring the most goals wins the game. Google reinforcement learning football environment [1] shown in Fig 1 consists of all the rules of conventional football and provides us with a complex setting to train and test out agents. At a given time our agent has to operate a single player in the game either the player with the possession of the ball while attacking or the player closest to the ball while defending[2]. There are 19 actions to choose for the agent ranging from movement actions to passing and shooting actions through which the agent can beat the defense and score goals or stop the attack from scoring goals.

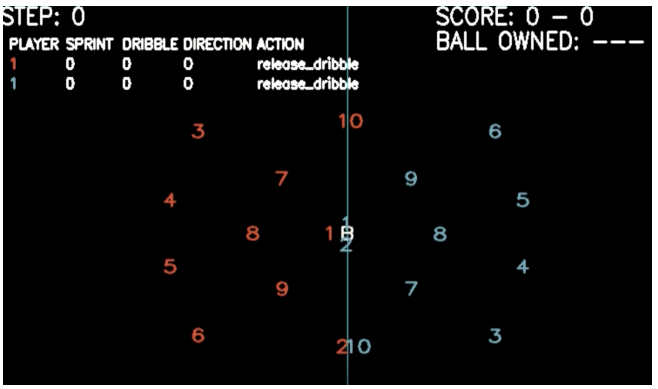

Fig 1. Google Reinforcement learning football environment

In this research, our contribution will be:

- Research and development of two agents for playing football namely, Deep Q networks and LightGBM based on reinforcement learning (RL) and supervised learning respectively.
- Collection of the dataset from kaggle for LightGBM.
- Both Deep Q Networks and lightGBM are made to play against each other for Comparative analysis of their performance.

II. RELATED WORKS

A. A Reinforcement Learning (RL) Environment Developed by Google Research

This paper discusses an open-source football learning environment where AI agents can learn and master to play the game of football [1]. This environment is a sophisticated 3-D simulator of the game and it is very easy to customize according to need as it is open-source as well. It combines two functions that can be used for rewarding: SCORING and CHECKPOINT. SCORING goes hand in hand with a natural prize where each side gets a +1 prize for scoring and a prize of -1 when the opposing side scores the goal. CHECKPOINT adds a SCORING reward for extra contribution to bringing the ball closer to the goal of the

opponent in a disciplined manner. Behaviour found in single agents are various passes, dribbles and moving speed. The goal for the benchmarks of football is winning the game against the enemy fed to the engine. The engine mimics wide-ranging football features, such as fouls, penalties, offsides, goals, corner kicks etc. These features are expected to be useful in exploring scientific challenges like self-play, sparse rewards, and model-based Reinforcement Learning (RL). The results show that the complexity of the environment greatly affects the complexity of training and the goal difference between them. Finally, they concluded that the Football benchmarks offer interesting research reference issues and that a great deal of improvement scope is there, mainly in terms of overall performance on complex benchmarks.

B. *Football using Machine Learning to improve current strategies of attacking play*

This paper tackles the limitation of conventional analysis of football gameplay by using video footage to derive tactics. It explores the prospect of machine learning to analyse football especially attacking play while considering the challenges faced currently and further deriving future tactics that may help gain deeper insights [3]. Machine learning algorithms offers much more better and automatic quantitative analysis than experimental methods because algorithms like Supervised learning are able to extract same & richer experimental data. Further, there are two different types of input data i.e event and tracking data. Event data mainly consist of recognition of patterns of teams and their characteristics & also identification of main performance measures that increase chance of success. The Tracking data was more related to ongoing processes like effective passes, probability of scoring goals without possession of ball etc. Even after the advantage of the event and tracking data there were many unpredictable scenarios like collisions between players, unpredictable movement etc. which posses big challenges in getting a good accuracy of information. So, further projects should focus on these challenges and work on reducing these errors. Finally, it is possible to integrate machine learning with team coaches which will help them to understand more interacting variables and provide them with practical and rich information at fast pace. However, there are problems that needs to be overcome like presenting the best and to point information to analyse which can be done by multi-disciplinary approaches like setting up research group of computer science and sports scientists who are competent in the game to extract relevant information.

III. METHODOLOGY

The given section gives an in-depth account of our approach of developing agents to play the simulated football game. The section also provides a detailed description of the architecture and working of our agents, along with the description of the environment and dataset used to train the agent.

A. *Environment*

We have used Google's developed environment of football, an open-source reinforcement learning (RL) environment [1]. The whole game is divided into two halves of 1500 steps each, and each team is assigned a half randomly but does not swap their halves after 1500 steps to keep things simple. For convenience, our agent controls only one player, either the player with the ball or the closest player to the ball. Our environment has 115 states or observations which are returned to our agent. Based on the current state, our agent needs to choose an optimal action out of 19 possible actions to improve its current state.

B. *Data Set*

For Light GBM, which is a supervised learning algorithm, we have extracted episode replays from Kaggle's API [4]. Each episode has 115 states/observations mapped to its corresponding label denoted 'y', representing the action taken by the model in the given state. Each episode came out to be of size 21 megabytes, and we have trained our agent on 100 such episodes making the training data of 2.1 gigabytes.

C. *Training and Evaluation Environment*

We have used Nvidia 3060 RTX with an intel i5 11th gen processor and 16GB RAM for training and evaluation of our agents.

D. *Methods*

We have developed two different agents to play the football game based on Deep Q networks and Light GBM. Deep Q networks are based on reinforcement learning which is a self-learning algorithm. On the other hand, Light GBM is an advanced gradient boosting machine learning framework based on decision trees and is a supervised learning algorithm. The description of architecture and working of both models are mentioned below in detail.

1. *DEEP Q NETWORKS*

Deep Q Networks comes under reinforcement learning and is an improvement and extension of Q-Learning. In reinforcement learning, an agent acts in an environment, and we may or may not know how will the environment react, which is described by a model based on the action taken by the agent.[5] The agent remains in any one of the many states, denoted by 's', and can move from one state to another denoted by 's'' by taking one of many possible actions represented by 'a'. The state the agent ends up in after taking action depends upon transition probability 'P'. Once the agent takes action, it receives a reward 'R'. The agent devises a policy defined by $\pi(s)$, which acts as a guideline to choose the optimal action in a state 's' to maximise the total reward. Each state has a value function V(s) associated with it to quantify how good that particular state is. The agent tries to learn value function and an efficient policy.

The agent interacts with the environment by taking the sequence of actions over a series of time steps t = 1, 2, 3, …, T [6]. During this process, our agent explores the environment to learn an optimal policy which helps our agent to choose an optimal action to maximize the reward. The interaction sequence of state, action, and reward at a certain time step 't' is represented by s_t, a_t, and r_t, respectively, and is described by one episode. This sequence ends at the terminal state S_T. With the help of the Bellman equation, the value functions are decomposed into immediate reward and discounted future values. The equation is a follows:

$$Q(s, a) = r(s, a) + \gamma * \Sigma P(s, a, s')\max_{a'} Q(s', a') \quad (1)$$

Where,

- Q(s, a): action-value pair
- r(s, a): reward in state ‘s’ after taking action ‘a’
- γ: Discount factor
- P(s, a, s'): transition probability from state ‘s’ to state ‘s'’ by taking action ‘a’
- $\max_{a'} Q(s', a')$: optimal Q function for future rewards

Discount factor $\gamma \in [0, 1]$ is used to penalise future rewards. There are many reasons to employ the discount factor in the equation. The most important among all is as follows:

- High level of uncertainty of future rewards
- Future rewards are not as beneficial as immediate rewards
- Acts as a mathematical convenience
- There is no need to worry about the infinite loops in the state transition graph

Deep Q networks are built and improved upon Q-learning. Q-learning is based on off-policy temporal differences and works as follows within one episode:

1. We first initialise time step, t = 0
2. Start from initial state S_0
3. At time step ‘t’, the agent chooses an action, say a_t, according to optimal Q-value
4. After choosing the action, the agent will receive a reward R_{t+1} and shifts to state S_{t+1}
5. The agent updates the current Q-value with the help of temporal difference
6. Finally, the agent increments the current time step by one, i.e., t = t + 1 and repeats from the third step

Although theoretically speaking, an agent based on Q learning can memorise all state-action pairs from a table, also known as a Q table. But practically, it is computationally expensive when state and action space is enormous. To deal with this problem, one can use a function approximator in order to approximate Q-values. But, Q-learning based agents suffer from divergence and instability in combination with non-linear Q values function approximation.

Deep Q-Network is responsible for improving this shortcoming of the Q learning based agent by introducing Experience replay and updating target periodically. Experience replay stores all the episodes into a replay memory and random samples of experience tuples are drawn out during the Q-learning updates. By employing experience replay, we improve the efficiency of the data and get rid of correlations in the data. Besides that, the agent also updates the target values periodically.

a) Agent’s Architecture

Our DQN agent consists of three components: model, replay buffer and an exploration method. These aspects of our agent are explained in detail below.

b) Model

Google research football environment returns observation as an array of size 115 passed as an input to a neural network with three layers. The neural network will output an estimated value of each possible action, and the agent will choose a move with the highest estimated value most of the time. The agent uses two different copies of the model values to make its learning more stable. The first model is known as the “Action” model, which is updated after every training step and is responsible for estimating the value of actions. The second copy is known as the target model, which is updated after copying the weights of the value models. The target model is updated less frequently to keep the agent’s training stable. The below figure shows the architecture of the model in detail.

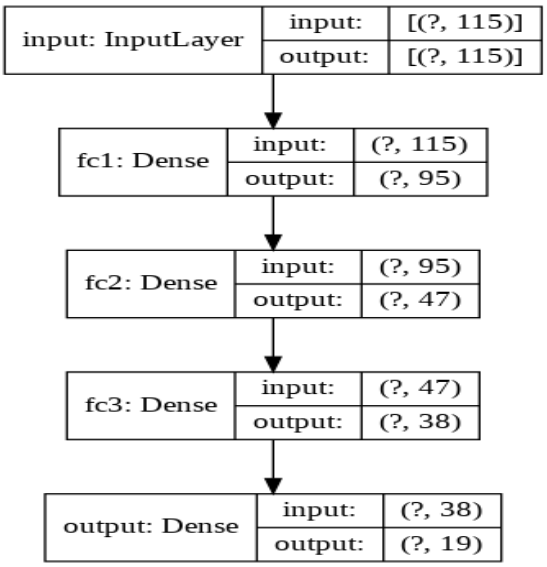

Fig 2. Model Architecture

c) Replay Buffer

The replay buffer is responsible for storing state, action and rewards. Once the agent is updated, the agent creates a training set by sampling a batch from the replay buffer. This replay buffer is of utmost importance as it helps us reduce correlation in the data and improve the working of the neural network.

d) Exploration Method

The exploration method is the final component of our agent, and we have employed epsilon greedy to achieve it. Our agent chooses a move from all the possible actions if the value of an action is greater than the value of epsilon. The value of epsilon was set to a high value initially, but it slowly moved to a much smaller value over time.

2. *LIGHT GBM*

Light GBM is a decision tree based gradient boosting framework used for classification, ranking and varieties of other machine learning tasks. Unlike different gradient boosting algorithms, Light GBM splits the decision tree leaf wise with the best fit. Light GBM can achieve far better accuracy than any other boosting algorithm because it can reduce loss more efficiently by splitting the decision tree leaf wise [7]. Light GBM utilises two techniques known as

Gradient-based One Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) [8]. These two techniques make Light GBM overcome the limitations posed by histogram-based algorithms, utilised by traditional Gradient Boosting Decision Tree (GBDT) frameworks. Together they make Light GBM more efficient than other GBDT.

Gradient-based One Side Sampling (GOSS) retains more significant gradient instances as these are more crucial to information gain and discard instances with smaller gradients. This keeps the information gain estimation more accurate than uniform and randomly sampled data. On the other hand, Exclusive Feature Bundling (EFB) bundles exclusive features into a single feature, thus improving the training speed of the framework without impacting the accuracy negatively.

The agent has trained on a dataset with 115 states. Each data point is mapped to its corresponding label denoted 'y', which represents the action taken by the model in the given state. Light GBM creates a decision based on this dataset, splits it leaf-wise, and chooses a leaf representing an action with minimum loss. Leaf-wise growth of the tree will increase the complexity of the model, which risks overfitting, but we have tackled it by keeping the max_depth parameter to the default value.

IV. Result

A. *Training*

TABLE I. Training of Deep Q Network

Number of Games	300
Time Taken	4 hours
Sustained CPU Usage (%)	52%

Deep Q Network agent training was done as per table 1. Further for the individual testing of agent performance it was played with run_right agent which always chooses right run operation. They played 10 games together out of which 4 games were won by deep q network agent and 6 games ended up being drawn. A reward vs episode graph is shown in fig 3 which was generated as a result of training.

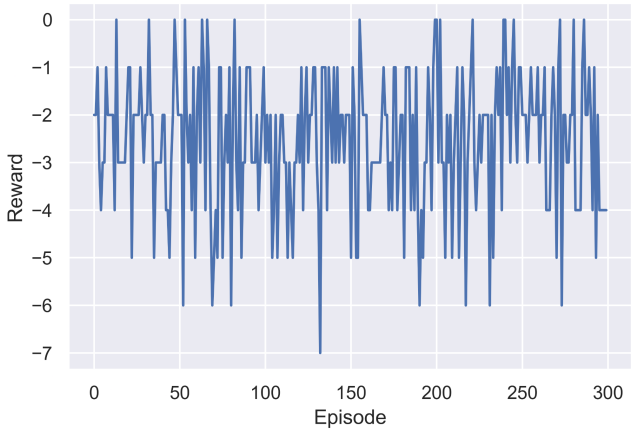

Fig 3. Reward Vs Episodes

TABLE II. Training of LightGBM

Training Data Size	2.1 GigaBytes
Time Taken	30 Minutes
Sustained CPU Usage (%)	58%

Further, LightGBM training is done as per table 2. And it played 10 games against the same run_right agent for individual testing out of which all of the 10 games were won by LightGBM.

B. *Deep Q Network Vs LightGBM*

Finally, the Deep Q Network agent and lightGBM agent were made to play against each other for 25 games for comparative analysis and generalizing results. LightGBM performed better by winning 18 out of 25 games played giving it a winning accuracy of 72% whereas Deep Q network won 2 games having a winning accuracy of 8% and 5 games ended up being drawn between the two agents.

V. Conclusion

We aimed to develop AI agents to play football video games efficiently. For this purpose, we used Google's developed environment of football, an open-source reinforcement learning (RL) environment. We created two agents based on Deep Q Networks (DQN), a self-learning algorithm, and Light GBM, a framework based on gradient boosting algorithms. During the comparative analysis of the two models, we saw Light GBM had a winning accuracy of 72%, and Deep Q Network had 8%. Although Light GBM dominated the DQN agent, we observed that DQN had a lot better possession rate than Light GBM. The agent based on DQN learned and implemented concepts like passing, dribbling, and defending more effectively than Light GBM but was struggling to shoot and score the goal. This behaviour of DQN can be attributed to its lack of more training due to our limited computation power. It was the main reason the DQN agent's winning accuracy is unimpressive than Light GBM, as Light GBM was trained on highly refined data. Based on our observation, It can be concluded that the overall performance of DQN can further improve by increasing its buffer memory and allowing it to play more games. Fig 3 Reward vs Episodes also backs up our conclusion as we can observe a lot of fluctuation in the graph, which denotes that our agent is still exploring and learning the environment. Provided sufficient computation resources, DQN will indeed outperform Light GBM.

References

[1] K. Kurach, A. Raichuk, P. Stanczyk: "Google Research Football: A Novel Reinforcement Learning Environment" [Accessed: Aug. 25, 2021]

[2] K. Kurach, A. Raichuk, P. Stanczyk: "Introducing Google Research Football: A Novel Reinforcement Learning Environment" [Accessed: Aug. 26, 2021]

[3] M. Herold, F. Goes, S. Nopp, P. Bauer: "Machine learning in men's professional football: Current applications and future directions for improving attacking play" [Accessed: Aug. 28, 2021]

[4] Kaggle Competitions: "Google Research Football with Manchester City F.C." [Accessed: Aug. 25, 2021]

[5] Weng, L : A (Long) Peek into Reinforcement Learning [Accessed: Nov. 25, 2021]
[6] Larsen, N. “Why is a target network required?” [Accessed: September 24, 2021]
[7] Analytics Vidhya : LightGBM vs XGBOOST: Which algorithm takes the crown [Accessed: Nov. 25, 2021]
[8] Geeksforgeeks: “LightGBM (Light Gradient Boosting Machine)” [Accessed: November 20, 2021]