Super nice title of the story how AI drives humankind to better place

Karol Szurkowski

University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark
kaszu19@student.sdu.dk

Abstract. A brief summary of your manuscript XXXXXXXXXXX.

1 Introduction

Ludo is a board multiplayer game, which is mostly played by 2 or 4 players. The objective of the game is to win the race around the board by moving one out of four pieces to finally reach the goal area with all four player's pieces. The first player to put all of their pieces in their goal area ends the game and wins. During the race the movements are dictated by dice rolls and few special rules apply, one of which is knocking out other opponent's pieces. In many cultures the game is known under the different names, like indian "Pachisi" or german "Mate, Don't Get Angry", which have slight different rules. In this paper the focus was put on the english rules of the Ludo game, which will be described later in rewards section.

In modern times, when human labour is often replaced by automation, we can even use artificially intelligent agents to play the game and, as the german name of the game suggests, trying not to get mad. One of the possible approaches that could be succesfully used to obtain an agent with high win-rate is behaviour cloning of the recorded human expert to train the neural network to make it copy expert's moves. This approach may either need a lot of games played or further optimization of the neural network. This paper presents the unsupervised approach based on Reinforcement Learning (RL). The main characteristics of the approach is training an agent by trial and error in not necessarily known environment.

Popular approach using RL would be to implement Q-learning algorithm, which allows the agent to evaluate the quality of each possible action given current state and choose the best move. As the state-space of the Ludo game was found to be approximately 10^22 which is slightly larger than that of Backgammon, as shown in the paper [1] searching the state space to find the optimal policy would take a lot of iterations. Due to possibility of not discovering every state because of the size of the state-space, in this paper, the multilayer perceptron network (MLP) was implemented to estimate the quality of the action given current state, was used.

An overview of the rest of the paper is as follows: in section 2...; section 3.1...; section 3.2 ...; in section 4 finally

2 Deep Reinforcement Learning approach

2.1 Reinforcement Learning

As mentioned before RL is an unsupervised approach of training an agent performing some actions in an environment, changing the environment and getting a reward for the action taken.

To make sure if the RL methods can be used to solve a problem the first thing to make sure is checking if the state can described with accordance to the Markov Property [4]. It means that the state representation must include information about all aspects of the past agent–environment interaction that make a difference for the future - difference in the rewards obtained and actions taken. In case of the Ludo game the requirements for the property are met which means that the problem of solving the game can be described as Markov decision process (MDP).

Moreover , because of the nature of the board game it is known that the sets of states, actions, and rewards all have a finite number of elements. That means that there exist a finite number of the state, actions and rewards combination which is what enables the problem to be solved by RL.

In method implemented in this paper, choice of the action should be understood as choice which piece of the current player's to move given the roll of the dice and knowing the state of the board. Knowing the current state s_t of the environment (board) agent chooses an action a_t, executes it and receives the immediate reward r_t. The goal of the agent is to choose the actions which gives the highest discounted cumulative reward, also known as return $R_{t_0} = \sum_{t=t_0}^{\infty} \gamma^{t-t_0} r_t$. γ here represents the discount factor which makes sure that the rewards from near future contribute more to the return than ones taken in further turns.

Following choices of actions in a certain fashion is understood as the policy $\pi(s)$ of an agent. The policy should be specific about what to do in any state - should map the state to the desired action in any possible situation found in the whole state-space. Q-learning algorithm finds an optimal policy $\pi^*(s)$ in the sense of choosing the action that potentially will return the maximum expected value of the total reward. In other words the policy of Q-learning is to choose the actions which will provide the best rewards.

The goal of training the RL agent is to discover whole state-space and obtain rewards for the actions taken in current state representation. Knowing immediate reward $r(s, a)$ that can be obtained from the given state s, we can assign a value to this state action pair $Q(s, a)$. As the agent will play more games, it will discover more states and the rewards, then will update the values of the state and thus will change it's policy π.

The goal of training the agent is to explore different states and find the optimal policy given by equation

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}(Q^*(s, a)) \tag{1}$$

Instead in this paper we get the optimal policy by approximating the optimal Q^* with an multilayer perceptron network (MLP).

Since nerual networks need supervision for proper training still the value of the action pair needs to be calculated following the Bellman equation.

$$Q^{\pi}(s,a) = r_t + \gamma Q^{\pi}(s_{t+1}, \pi(s_{t+1})) \tag{2}$$

Detailed pipeline of training the model is presented after the description of used state representation and reward function.

2.2 State representation

As described before the representation of the environment is called the state, denoted S and it has to contain enough information to make sure that there should exist one possible reward for one state. That means that the state representation can not be ambiguous and should be condensed with the information.

To make sure the state is unambiguously represented in this paper the state contain information of the full board. That means it contains the information about the pieces of every of the four players in a game, similar to the state used in the paper [1]. Full board is described by 240 float variables, 60 for each player in the rising order. 60 variables were used because there are 58 tiles of the race track, one tile for the safe home (starting position) and one tile for the safe goal position, which can be observed in figure 1 representing the board of the game. Each variable represents the percentage of the player's pieces being on that specific tile - values range from 0 till 1. For example, during the start of the game, 3rd player was first to roll 6 and let his piece out of the home to the first tile the whole state will consist of 240 variables presented in the list [1 59x0, 1 59x0, 0.75 0.25 58x0, 1 59x0], where the commas separate the tiles of different players and 59x0 represents 59 zeros in a row.

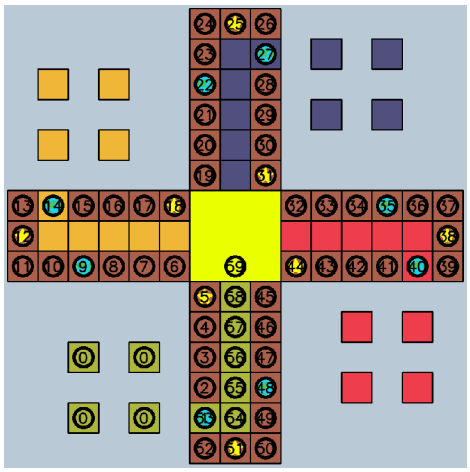

Fig. 1. Board of the game representing the 60 tiles seen from one player. The state representation carries the information about the tiles of 4 players described with 240 variables.

Representation described above enables simple access to the position of every piece of the player which is used to calculate the immediate reward given to an agent after performing an action. Also it can be easily extended with the actions taken to represent the input to the MLP network.

2.3 Reward function

When agent performs an action a_t, it changes the state of the environment from s_t to s_{t+1} and receives an immediate reward r_t. Assigning the correct reward should directly influence the performance of the agent and shape it's behaviour, i.e. change the agent's policy.

Rewards are given for choosing certain actions allowed by the rules of the game, when using different rules it may be worth to delete the rewards for forming blockades, moving on stars that teleport the piece, or globes that prevents the piece from being knocked out. Below the list of the rewards used in the paper is presented.

- 1 for winning a game,
- 0.25 for releasing a piece from safe HOME,
- 0.2 for forming a blockade on piece which is surrounded by enemy pieces,
- 0.15 for knocking out an opponent's piece,
- 0.1 for moving the piece that is the closest to the goal, but not in the safe zone,
- 0.05 for forming the blockade,
- 0.12 for getting on a globe (safe zone),
- 0.17 for using a star,
- -0.25 for loosing a piece in the next turn,
- -1 for losing a game.

Those rewards can be accumulated, e.g. when at the same time the action will move the furthest piece on the star the total reward from this move should equal 0.27. If an action does not fall in one of these situations put on the list above, no reward is given.

2.4 Architecture and training of the MLP

Similar to [2] and [1] the network approximating the Q function was fully connected multilayer perceptron neural network with one hidden layer. The network was implemented using machine learning library PyTorch.

To estimate the $Q(s, a)$ with a neural network, it is needed for its input to consist the information of transitioning from the previous to the next state with visible action taken. That is why the input of 240 variables got appended with additional 2 variables which indicate the number of tile from which piece moved and the number of tile that the piece got to after the action. Then the number of tiles (ranging from 0 to 59) is divided by 59 to ensure that all the values in the input of the neural network range between 0 and 1.

Architecture of the network is described below:

- input: vector of 242 variables: 240 represent state and 2 represent the change of the piece position,
- 1 hidden layer of 20 neurons with symmetric sigmoid activation function
- 1 variable as an output which is the estimate of $Q(s, a)$ with rectified linear activation unit.

When using the MLP to predict the $Q(s, a)$ what actually matters is the weights of the network which are updated during to training because of the back-propagation of the error. In the situation when the network is not yet fully trained, the initial weights have a big influence on the output, since they are not fit for the use of the MLP. That is why it is beneficial to pretrain the network. This can be accomplished with recording the human expert data, as done in this implementation, by saving all the necessary information about the game and later calculate the rewards got every action and use them to obtain the initial training dataset to estimate $\hat{Q}^*(s, a)$.

2.5 Action selection and training using Deep Reinforcement Learning

describe the workflow: how to calculate the Q from rewards as training set, What batch size was used Explain the approach when training the model. Why action is taken and after training the two networks get synchronized and we use the predictions from the MLP instead of calculating the Q values

Training the DRL agent in general resembles the normal Q-learning approach but has some differences. When agent is about to select an action in an e-greedy[**egreedy**] manner, instead of calculating the current $Q(s, a)$ the estimation of the $\hat{Q}^*(s, a)$ is obtained from the MLP. In this moment, update of the values of the state-space can not be observed. Update of the network happens after the decision is taken and when there is enough training data in memory. Because of the fact that initial estimations of the Q might be far from optimal due to random decisions of actions, it is beneficial for the system to forget some of the estimates, and forget them from the memory. That is why the training of the MLP begins, when the memory has enough samples and the training can be done in batches with fixed size of the stored experiences. Moreover, after achieving big enough pool of experience, changing the weights of the network by training and constantly using the same network to estimate the $\hat{Q}^*(s, a)$ has high chances to cause instability in training [3]. To solve this issue two MLP networks with the same architectures are used. One to constantly return the estimate which is used to select an action, and the other to constantly train using the gathered experience from the batches. The estimating network is updated to the state of the constantly trained network with certain frequency.

2.6 Training the AI agent

CHANGE PARAMETERS HERE IF NEEDED

Knowing the pipeline of the DRL approach - how the actions are selected and what is the way to make the agent learn to solve the game, this section will explain the parameters used in the process of training.

We can clearly distinguish the parameters concerning calculation of the quality of given state and parameters describing training the neural network.

Classical RL parameters are: $epochs = 200$ - number of full games played to gather the data and train the agent. Due to the training being time demanding process, the agent was trained on 200 epochs. $\epsilon - from 0.95 to 0.05$ indicating how greedy is the choice of the action given current state. This parameter was tuned with having the $batch\_size$ in mind, to ensure that the actions are becoming less random with the training of the neural net. $\gamma = 0.95$ - discount factor which punishes rewards obtained in the future.

Using the MLP brings more parameters: $batch\_size = 1200$ - which specifies the number of experiences of selecting an action given current state to calculate the $Q(s, a)$. It was chosen to store experience from only around the last 14 games. $f_{MLP_{update}}$ - networks update frequency. It says how many times to train the estimating network before updating the decision-making network. $\alpha = 0.005$ is a learning-rate of the weights update. It is chosen to be small to try to prevent the learning from becoming too unstable.

results from pre-trained mlp from human data

As mentioned before to reduce the biases coming from the initial weight association to the neural network, human expert data was used to pretrain it. 4 games played by human were recorded, and the moves were used as the training data. MLP was then trained for $epochs = 200$ with the $bath_size = 50$.

results from the 200 epochs

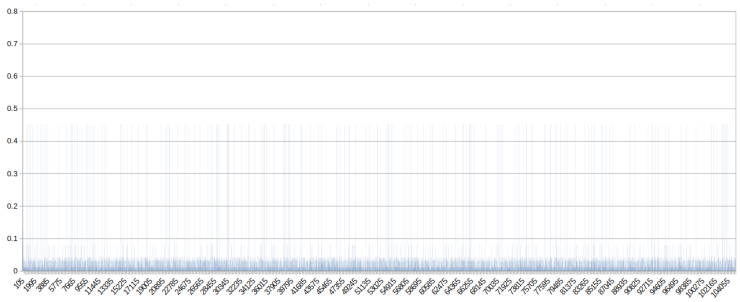

Fig. 2. Losses from pretraining the human expert data with setup containing of 200 epochs, batch of size 50 and 4 human recorded games.

is that proof for stochasticity of the learning?

As seen in the figure 2 the losses were still present even after 200 epochs. Moreover, to later check the performance of true, online training, the network was evaluated in action. With one DRL player trained on the expert data alone competing with 3 random agents, the DRL within 200 games managed to win 45 times, thus giving 22.5% win-rate.

3 Second method - to compare against

4 Results

Test of few parameters -¿ choosing the best performing model Compare DRL with other method

5 Analysis and Discussion

6 Conclusion

7 Acknowledgements

References

[1] Faisal Alvi and Moataz A. Ahmed. "Complexity analysis and playing strategies for Ludo and its variant race games". In: *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)* (2011), pp. 134–141.

[2] A. Jaramillo and Deepak Aravindakshan. "An Artificially Intelligent Ludo Player". In: 2016.

[3] Mohit Pilkhan. *Building a DQN in PyTorch: Balancing Cart Pole with Deep RL*. 2020. URL: `https://blog.gofynd.com/building-a-deep-q-network-in-pytorch-fa1086aa5435`.

[4] Richard S. Sutton and Andrew G. Barto. Introduction to Reinforcement Learning.