TAI: TAnktrouble reInforcement learning model based on Deep Q-Networks

Written by XMU students[1]
Bo Wang, Jiefeng Wang, Yibo Miao, Yue Hong

[1]Department of Computer Science, School of Informatics, Xiamen University

Abstract

With the development of the game industry, the game AI is invented and becomes more and more important for game iteration. In the view of game companies, game AI bots can help them find the weakness and verify the fairness of the game design. In the view of players, game AI bots can play and compete with human players, increasing their willingness. In our work, we propose TAI model for implementing the Tank Trouble game AI bot based on Deep Q-Network (DQN) and its variants. To obtain a higher win rate, we make a corresponding improvement to the training network and design a proper reward function based on our game to make the training process more efficient and lead to a better training result. Our model has a preliminary effective win rate increase after training on the Tank Trouble game.

Introduction

Many game companies are dedicated to designing superior and complicated Artificial Intelligence (AI) for their game. Adding with game AI elements appropriately, players not only experience more pleasure but also enhance their operability of the game. Not just traditional chess game AIs which are famous as 'AlphaGo', mull real-time multiplayers' battle games are designing their own AI-based on a much more complicated deep learning framework.

Unfortunately, the design for game AI is considered one of the most crucial business secrets for these game companies. It is hard for us to learn from these excellent and stunning designs. If we consider learning and designing relative game artificial intelligence, we need to find a suitable game. This game needs to meet several conditions in our consideration:

- The game supports multi-user battles and requires appropriate strategies.
- The game has fewer control parameters for simplifying model size.
- We can obtain backend data quickly.

Generally, Deep reinforcement learning (DRL) can be classified into two main classes: value function-based methods and policy-based methods. DQN is the most typical model of the value function-based method. DQN is very suitable for most strategy real-time multiplayer battle games. Thus, we are supposed to design a game AI for 'TankTrouble' based DQN and improve this model to a large extent.

In TAI model, we chose an open-source project named 'TankTrouble'.Two players control these two tanks respectively Players need to aim at their enemy and avoid bullets as much as possible. Reinforcement learning (RL) successfully works in many games AI designs. We improve our model based on DQN and apple it in 'TankTrouble'. We modify Q-Network based Deep Double Q-Network (DDQN) algorithm and speed up the training process of DQN based Dueling DQN algorithm and other tricks.

However, the mechanics of this game are relatively complicated, such as the bounce of bullets in complex terrain. How to let the DRL model learn complex game mechanics is a problem. In addition, the DQN model needs to design different rewards according to different game rules. The network structure also needs to change accordingly.

The contributions of TAI are as follows:

- We designed and implemented the first DRL game AI for 'TankTrouble'.
- We improved the original DQN model and compared the effects of various tricks on the model.
- We learn how to design rewards and adjust game strategy through experiments.

Related Work

Deep reinforcement learning (DRL) has been successfully applied to training auto game players (Mnih et al. 2013; Wu 2019). Deep Q-Network(DQN) outperformed humans on about 2600 Atari games in 2013(Mnih et al. 2013). In addition, in 2019, Juewu developed by Tencent that has two layers applying attention mechanism achieved a 48% winner rate(Wu 2019). What's more, (Torrado et al. 2018) realizes an interface GVGAI to the OpenAI Gym environment, a widely used way of connecting agents to reinforcement learning problems. To the best of our knowledge, from the algorithm perspective, DRL can be classified into two main classes: value function-based methods and policy-based methods.

In DQN, deep networks and reinforcement learning were successfully combined by using a convolutional neural net

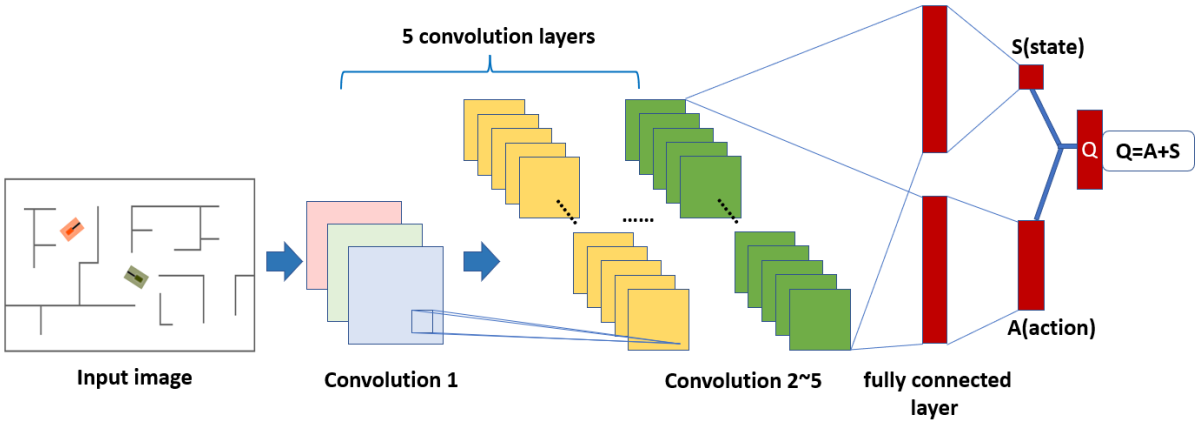

Figure 1: The Proposed Architecture of TAI Network

to approximate the action values for a given state S_t(which is fed as input to the network in the form of a stack of raw pixel frames). At each step, based on the current state, the agent selects an action ϵ-greedily with respect to the action values, and adds a transition (S_t,A_t, R_{t+1},Y_{t+1},S_{t+1}) to a replay memory buffer, that holds the last million transitions. The parameters of the neural network are optimized by using stochastic gradient descent to minimize the loss. (Yoon and Kim 2017) proposed a DQN approach to train a fighting game AI, and (Takano et al. 2018) proposed a method for implementing a fighting game AI using Hybrid Reward Architecture(HRA). Besides, (Takano et al. 2019) train a general fighting game AI from self-play games and overcome the drawback while maintaining the DQN AI's strong points. The above methods have achieved good results in fight game AI, like Tank Trouble.

Mnih et al. (Mnih et al. 2013) combined the convolutional neural network with the traditional RL. The combination of the Q learning algorithm produces the Deep Q-Network (DQN) model. This model is used to process the vision-based work in the control task. It is pioneering work in the DRL field.

Van Hasselt et al (Hasselt, Guez, and Silver 2015) based on double Q-learning (Narayanan and Jurafsky 2007), proposed Deep Double Q-Network (DDQN) algorithm. There are two sets of parameters in double-Q learning: θ^+ and θ^-. Among them, θ is used to select the corresponding maximum Q value. The action, θ^- is used to evaluate the Q value of the best action. Action selection and strategy evaluation are separated, which reduces the overestimation Q value risks. Therefore, DDQN uses the current value network parameter θ to select the optimal action, and uses the parameter θ^- of the target value network to evaluate the optimal action. Dueling DQN is a variant network of DQN that is easy to implement. Dueling DQN improves network structure based on DQN. With this different kind of network structure, Dueling DQN can be trained faster than the original DQN.

DQN recycles attention and sets a neural network (NN) to calculate value function to improve learning stability. However, it has limits in processing input with arbitrary length and is poor in handling continuous high-dimension datasets. One policy-based method called DPG accepts continuous high-dimension problems (Mnih et al. 2013). DPG learns policy during the process instead of optimizing a value function. But low converge speed makes DPG impractical. A combined method named the Actor-Critic algorithm maintains two NNs. One is an Actor that trains to learn policies, the other is Critic aiming to judge the policies. Some representatives of Actor-Critic are DDPG, A2C and A3C (Lillicrap et al. 2019; Mnih et al. 2016). Those methods are able to carry on in a parallel fashion.

Proposed Solution

Environment Setup

Game Recreation Based on (Mohammadjavadpirhadi, Tmoj1428 2018), we recreate the game. In the open-source game, the operations of movement and rotation are determined by the duration of the player's key pressing, and the direction is unified into the interface. And it generates two tanks whose locations are from a fixed specific array at the beginning of the game or when the game is reset. The key idea here is that 1) split the operations of movement and rotation into four actions, in order to execute the subsequent simulation actions by limiting the scale. (At one action, movement is limited by a fixed distance, and rotation is limited by a fixed angle.) 2) generate two tanks randomly on the map, while not colliding with each other and the wall. 3) remake the mechanism of collision detection and reflection against the wall to make the game more accurate, otherwise, subsequent bugs will cause errors in the model. 4) redefine the size of maps, tanks, and bullets, as well as the speed of tanks and bullets, so as to facilitate the training of deep reinforcement learning models.

OpenAi gym Environment Gym is a simulation platform for research and development of reinforcement learning-related algorithms. We converted our imitation tank trouble

game into an OpenAi environment for reinforcement learning training. Following is three basic interfaces:

1. space. There is always action_space in gym env. In our env, action_space represents the types of actions that can be taken. There are six types of actions in our environment, namely, (0: fire, 1: forward, 2: backward, 3: left, 4: right, 5: NON_ACTION).
2. step. Step refers to the action in the simulation game, and return four values accordingly: observation, reward, done, info.
   (a) Observation represent the current state of the object, such as position information, angle information, etc. In our work, the current state means the current frame.
   (b) Reward: amount of reward achieved by the previous action. In out work, a variety of different rewards are set, which will be introduced in subsequent chapters.
   (c) Done: whether it's time to reset the environment. In our work, when any tank is destroyed by a bullet, done is true, otherwise done is false.
   (d) Info: diagnostic information useful for debugging. In our work, info is used to print bullet information.
3. render. Render is a rendering engine, and the game can run without a render. But render is used to visually display the state of objects in the current environment, and also to facilitate our code debugging. In our work, render returns the current state.

Improved methods based DQN

We use DQN and two improved models based on DQN to train our tank bot. The basic DQN is composed of a reply memory and two networks. The basic idea of DQN is to collect the game experience from the environment and use these experiences to train a neural network that can compute a Q-value function. The network can be trained by gradient descent.

The Q-value function $Q(s, a)$ is used for evaluating the agent's action. The goal of the agent is to interact with the environment by selecting actions in a way that maximizes future rewards R. So the Q-value action is defined as $Q(s, a) = max_\pi E\{R_t|s_t = t, a_t = a, \pi\}$, where π is a policy mapping sequences to actions (or distributions over actions) and t means the future reward getting at time t.

The basic steps of model construction is depict as follows:

1. We utilize a replay memory to remember the experience provided by the agent observed from the environment. Each step a tuple $e_t = (s_t, a_t, r_t, a_t + 1)$ is stored in a dataset $D = e_1, \ldots, e_n$
2. After collecting experience,the agent will select and execute a new action. With probability of ϵ, the action will be chosen according to the neural network, in other case, it will be chosen at random.
3. Every time new action was selected, we update the policy network using the last 16 transition sampled from the replay memory. For every 10 episode, we update the target network. The target network is used for fixing parameters.

DQN Different from vanilla DQN(Hasselt, Guez, and Silver 2015), the DQN model we used in our tank game did not sample memory randomly but used the last 16 images as input to train our model. The time-series relationship is quite different with previous games(Wu 2019; Lillicrap et al. 2019). Due to the reason that the bullet will bounce, there exist various chances of the next movement even the bullets locates at the same position. Random choice may disrupt the model's experience. Based on the above assumption, we determined to sample the last 16 frames to feed our model.

During our practice, we found that delayed rewards made it difficult for the model to learn how to win the game, so we tried to increase the variety of rewards to guide the model to learn how to play the game. One of our proposed rewards is to give rewards based on the inter-direction between two models. Give the model a one-point bonus if our model faces the enemy head-on, and a -1 point penalty if it turns its back on the enemy. Other angles give a linearly varying bonus depending on the angle as equation (1).

$$reward_1 = -\frac{1}{90}\theta + 1 \tag{1}$$

, where θ is the angle between two tanks. It is also possible to use a alternative reward equation $reward = cos(\theta)$.

However, this reward only works in an empty environment. When there is a wall blocking the enemy, we need to add a special negative bonus to block it, as equation (2) shows.

$$reward_2 = reward_1 + 0.5 \times N_p \tag{2}$$

, where the reward is from (1), N_p stands from the pixel numbers of the wall that blocks shooting the enemy. N_p can be calculated by the Bresenham's line algorithm(Bresenham 1977), which is illustrated by Figure 2.

In summary, the final reward works as follows,

$$reward = \begin{cases} 100 & Candy\ tank\ dead \\ -100 & Green\ tank\ dead \\ cos(\theta) + 0.5 \times N_p & Others \end{cases} \tag{3}$$

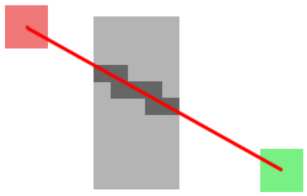

Figure 2: The illustration of N_p

DDQN The first improved model is called DDQN. The computation of Q value is improved in this model. If the estimate of the Q function is inaccurate, it causes an overestimation every time max is taken, so two Q networks are used to solve this problem, one network chooses the optimal action, the other estimates it.

Dueling DQN The second improved model is called Dueling-DQN. Generally, we estimate Q value in the Q table directly while we need to estimate S value and A value during implementing Dueling-DQN. S value can be regarded as the average Q value in a certain state. A value can be calculated by Q value and S value with the limitation of average value. The sum of the S value and the A value is the Q value in the Q table of DQN which leads to the average of the A value being zero. leads to the average of A value is zero.

In the Q network of DQN, it can be understood that a curve is used to fit the Q value of the Q table. Now we take a cross-section to indicate the Q value of each action when we take a certain state. However, when we need to update the Q value of an action, we will directly update the Q network to increase the Q value of this action. With the limitation of average on A value in Dueling-DQN, we will update the S value firstly when we update the network. Because the S value is the average Q value, the adjustment of the average means updating Q values with one S value at one time. As the result of this adjustment strategy, it not only updates the Q value of a certain action but also adjusts the Q values of all actions for this state when the network is updated. In this way, we can update more values fewer times.

TAI The implementation of TAI can be divided into three parts. The first part is the same as ordinary DQN to process and learn data. The second part is calculating the S value to train the network estimating the average. The third part is calculating the A value like the S value. Then we normalize the A value which increases the limitation of the average of the A value is zero. The processing of normalization is calculating the average of the A value and then subtracting the average value from the A value.

The implementation of TAI only needs to modify the network architecture of the Q network. In the meantime, TAI can implement other DQN techniques, such as experience playback, fixed network, double network structures computing targets, etc.

Model Implementation

Preprocessing Procedure The original TankTrouble game is not suitable for training a good DQN model. The original frame size is 1920×1080 pixels, which can be computationally demanding. Moreover, the complex map is too hard for our agent to learn. So we need to use some preprocessing strategies. First, we resize the input frames to 400×600 to reduce the input dimensionality. Then we design a set of maps differing from difficulty value. The different maps create a friendly environment for the tank agent to learn gradually.

TAI Network Architecture The deep learning network in TAI includes five convolutional layers.The architecture is showed as Figure 1. Each convolutional layer is followed by a Batch Normalization layer and an active layer using the LeakyReLU activation function. In the first convolutional layer, the number of input channels in the input image is 3 and the number of output channels is 16. In the second convolutional layer, the number of input channels is 16 and the number of output channels is 32. In the third convolutional layer, the number of input channels is 32 and the number of output channels is 32. In the fourth convolutional layer, the number of input channels is 32 and the number of output channels is 64. In the fifth convolutional layer, the number of input channels is 64 and the number of output channels is 64. All the filter in the convolutional layer has a filter size of 3×3 with a stride of 2. After the fifth convolutional layer, there are two linear layers training S value and A value respectively. The output dimension is equal to the number of actions in the Tanktrouble Game DQN agent.

Experiments

In this section, we evaluate our model with three experiments to verify the performance of the TankTrouble AI and show the experimental results.

Hyperparameters	Value
BATCH_SIZE	16
GAMMA	0.9
EPS_START	0.9
EPS_END	0.05
EPS_DECAY	200
TARGET_UPDATE	10
MEMORY_CAPACITY	300

Table 1: LIST OF HYPERPARAMETERS

Hyperparameters

We set the hyperparameters according to our empirical results. Their detail are given in Table 1. BATCH_SIZE stands for the number of frames that we feed into DQN models. GAMMA makes rewards from the uncertain far future less important for our agent than the ones in the near future that it can be fairly confident about. The probability of choosing a random action will start at EPS_START and will decay exponentially towards EPS_END. EPS_DECAY controls the rate of decay.

Training Model

We separately train models on a Tesla V100 machine for 48 hours. The metric we use to evaluate the model is the reward sum at the end of each epoch. In our experiment, there is a reward for each action performed in each epoch, and it is uncertain how much action would take to finish each round.

The reward sum for each epoch is recorded and analyzed. The experimental results are shown in Figure 3. The models have been all running 3500 epochs at local (TAI model trains 3000 epochs). And our TAI model rewards is shown by Figure 4. We will discuss the results in the Comparisons.

Results and Analysis

With training rounds increasing, DQN and DDQN show a satisfying trend in the early stage and then show a downward trend. This is most likely due to overfitting in the model. In the DQN model, the model's rewards peak at around 2500 epochs, and then the rewards began to fluctuate downward. In the DDQN model, the model's rewards peak at around

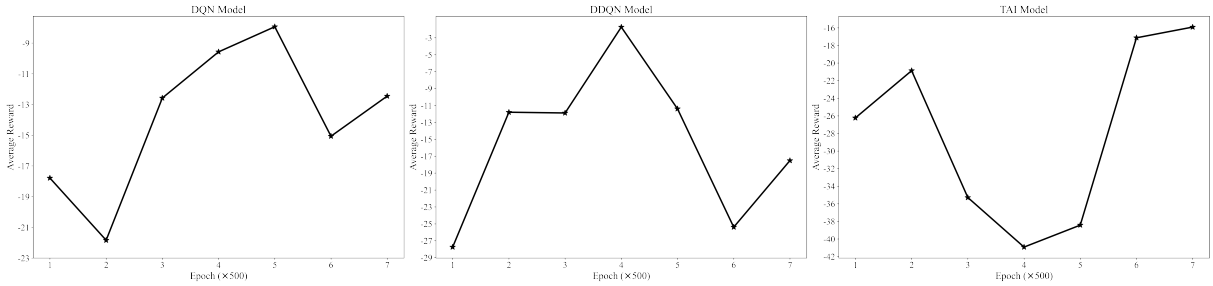

Figure 3: Rewards of DQN and its variants

2000 epochs. It shows that DDQN takes roughly the same epochs with DQN model to achieve satisfying results. However, because of the improvement of optimizer function in DQN, we can clearly observe that the rewards of the model are greatly improved compared to the DQN model. In the TAI model, we improve the network structure of DQN and DDQN. We observe that the model can achieve good training results. In the early stage of training, the quality of the model is poor, and it is difficult for the Action selected from the target-net to have a positive reward. With the training of the model and the guidance of the reward function, the model is easier to obtain higher rewards than before.

By observing the details of the results of TAI model in Figure 4, we observe that the model fluctuated randomly before 1400 rounds, and the average value of the worst rewards in the model reached -60. After 1400 rounds, although the rewards of the model was still wobbling, the reward generation of the model keeps increasing on the whole and reaches -10 after 3000 rounds. The training effect of TAI model is better than that of the first two models. Since there are too few training rounds, we will keep training the model in future Work.

Discussion

In this section, we will present our views on our experiments. Specifically, we will present what the current model can learn, what can not learn, and why the training result is not efficient enough. Finally, we will introduce our future work.

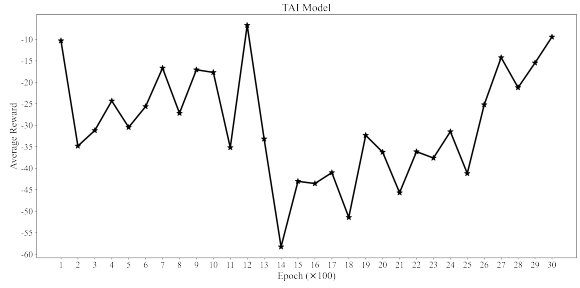

Figure 4: Reward of TAI Model

Knowledge

What can learn Based on the experiments, we find that the AI can identify the enemy's position, and adjust its position and direction for more precise aiming. What's more, the bullet's position can be learned so that the tank can dodge the bullets. In addition, instead of random firing, the AI can tune self-direction to the enemy automatically. As a result, our AI achieves a high win rate.

What can not learn However, we also found that AI can not master the reflex mechanism and uses it to kill enemies. What's more, the AI can not record the map information, namely, the wall position and direction, which would make killing enemies a lot easier.

Why can not learn The reasons for the poor model training effect are as follows:

1. The number of convolutional layers is less, and the model network is not deep enough to obtain enough high-level features from the input image.
2. The batchsize of each training is not large enough to obtain experience from a large number of images. Continuous input of multiple frames of images may lead to overfitting of the model.
3. Too few training rounds, GPU resource constraints, unable to read a large number of images into GPU, resulting in too slow training speed.

Future work

In future, we will add maps of more complex terrain and train the two tanks at the same time to achieve real-time battle. Besides, we will implement other types of DRL models for more comparison.

Conclusion

In this paper, the main work we do is to train an AI bot TAI for 'Tank Trouble'. We design our AI bot based on the DQN model and its improved algorithm. By fixing the network architecture and making changes to the Q-value function, we propose a new model which can perform more effectively on the specific game. Evaluating the models by experiments, we find our model has a preliminary effective win rate increase after training. In the future, We will try to apply this model on the more real-time battle games as Tank-trouble.

References

Bresenham, J. 1977. A linear algorithm for incremental digital display of circular arcs. *Communications of the ACM*, 20(2): 100–106.

Hasselt, H. V.; Guez, A.; and Silver, D. 2015. Deep Reinforcement Learning with Double Q-learning. *Computer ence*.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2019. Continuous control with deep reinforcement learning. arXiv:1509.02971.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T. P.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous Methods for Deep Reinforcement Learning. arXiv:1602.01783.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing Atari with Deep Reinforcement Learning. *Computer Science*.

Mohammadjavadpirhadi, Tmoj1428. 2018. https://github.com/mohammadjavadpirhadi/TankTrouble/.

Narayanan, S.; and Jurafsky, D. 2007. Advances in neural information processing systems 14: Proceedings of the 2001 conference. *Mit Press*, 159(2): 1659.

Takano, Y.; Inoue, H.; Thawonmas, R.; and Harada, T. 2019. Self-Play for Training General Fighting Game AI. In *2019 Nicograph International (NicoInt)*, 120–120.

Takano, Y.; Ouyang, W.; Ito, S.; Harada, T.; and Thawonmas, R. 2018. Applying hybrid reward architecture to a fighting game ai. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, 1–4. IEEE.

Torrado, R. R.; Bontrager, P.; Togelius, J.; Liu, J.; and Perez-Liebana, D. 2018. Deep Reinforcement Learning for General Video Game AI. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, 1–8.

Wu, B. 2019. Hierarchical Macro Strategy Model for MOBA Game AI. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01): 1206–1213.

Yoon, S.; and Kim, K.-J. 2017. Deep Q networks for visual fighting game AI. In *2017 IEEE conference on computational intelligence and games (CIG)*, 306–308. IEEE.