Advanced Atari Agent boosted by enhanced Deep Q-learning Network

1st Hangzheng Lin
ECE Department
ZJU-UIUC Institute
Haining, China
hangzheng.17@intl.zju.edu.cn

2nd Jiayuan Huang
ECE Department
ZJU-UIUC Institute
Haining, China
jiayuan.17@intl.zju.edu.cn

3rd Xiyue Zhu
ECE Department
ZJU-UIUC Institute
Haining, China
xiyue.17@intl.zju.edu.cn

***Abstract*—We train a Atari agent using deep Q-learnging Algorithm (DQN) with Experience Replay Memory and Target/Local Q Network mechanism. We try to improve the performance by adding history, decolorization, Model Structure Modification and ϵ-adaptation features. Our results are limited by limited time and computational resources we have. However, we still find the relation between the computation required and reward latency by comparing our performance between different games.**

***Index Terms*—Reinforcement Learning, Deep Q-learning, Neural Network, RAM inputs**

I. Introduction

Reinforcement learning can solve many application problems in artificial intelligence which require algorithms to make decisions and perform actions at every moment. For example, in Go(Weiqi), each move requires deciding where to place the pieces on the board in order to overcome the opponent as much as possible; for the automatic driving algorithm, it is necessary to determine the current driving strategy according to the road conditions to ensure the safe driving. There is a common feature for such problems: to make decisions and actions according to the current conditions. Although the traditional reinforcement learning theory has been improved in the past few decades, it is still difficult to solve the complex problems in the real world.

To meet this challenge, deep reinforcement learning has become a new research hotspot in the field of artificial intelligence and attracts much attention. It combines the perceptual ability of deep learning with the decision-making ability of reinforcement learning in a generic form and enables direct control from raw input to output through end-to-end learning. Deep reinforcement learning methods have made substantial breakthroughs in many tasks that require the perception of high-dimensional raw input data and decision control. Deep Q-learning Network [1] is the first end-to-end model without manual extraction of states and features. It integrates deep learning and reinforcement learning. It can solve the decision problem in the complex environment. The method is universe and can be applied to various problems.

Among many topics in improving the performance of deep RL, reward shaping has always been an interesting track to follow. Generally, reward shaping is proposed to solve the following two problems in reinforcement learning: a. to solve the long-term credit assignment problem, which means that the agent can only get substantial reward value after the end of a certain game level, and the reward value at other times is zero. As a result, the agent cannot recognize the contribution of a certain action to the final reward in a certain state and cannot identify the key action; b. to encourage appropriate exploration especially when the optimal solution requires much exploration while the agent is not likely to try for the sparse reward.

In the Arcade Learning Environment [2], which provides an interface for a variety of Atari 2600 play environments and designed to be engaging and challenging for human players, there are various works proposed to train an intelligent agent with reward shaping method. Exploration By Random Network Distillation (RND) [3] and Never Give Up (NGU) [4] introduced intrinsic reward to encourage exploration. Hindsight [5], on the other hand, starts directly from long-term credit assignment questions and calculates the importance of the current action based on future status and rewards.

However, almost all the models mentioned above adopts game screens as input and use Convolution Neural Network to extract the feature embedding, although with various methods. In this paper, we proposed to use RAM input which contains some data structures storing the state and value information. In other words, we assume RAM can be regarded as the extracted feature embedding of the current game state, which contains much information and can be better utilized to train the model.

II. Related Work

A. Deep Q-learning Network & Its related algorithms

The DQN algorithm [1] is the pioneering work of deep reinforcement learning. DQN uses a deep convolutional neural network to fit the Q function, which is called the Q network. The input of the network is the processed game screen (the last 4 frames of the game screen), and the output is the Q function value of various actions performed in this state. The overall network structure is shown in Figure 1.

Given a state, the current neural network is used for prediction to obtain the Q function of all actions, and then an action is selected to execute according to the strategy, and the next state and return value are obtained, so as to construct training samples.

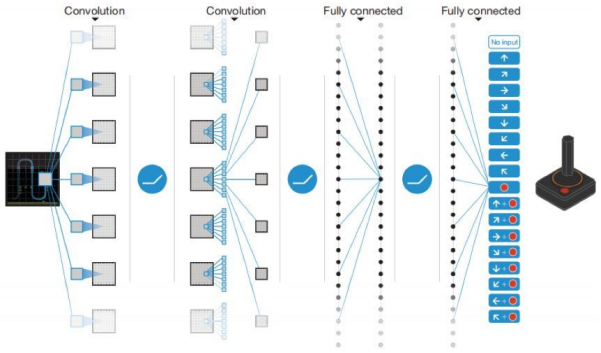

Fig. 1. DQN Network Structure

Different from the ordinary supervised learning, the training samples here are generated dynamically through the continuous execution of actions. In order to solve the correlation between the training sample, and the probability distribution of the sample is not fixed, used the experience playback mechanism, particular way is, the action structure of the training sample first store into a big collection, Q in training the network every time some samples randomly from the set as the training sample, to break the correlation between samples.

The loss function consists of the output value of the neural network and the updated Q value during each iteration of learning. It is the error between the output value of the neural network and the estimated value of Q function, which is the same as the updated term in Q learning:

$$L(\theta) = E((r + \gamma max_{a'} Q(s^{'}, a^{'}; \theta) - Q(s, a; \theta))^2) \quad (1)$$

The DQN algorithm was evaluated with the Atari game and achieved better performances among many games than average human scores. However, there are still some issues remaining, for example, the same maximum strategy deployed in evaluation and decision for Q value results in over-estimated Q value, which means the agent may take action based on impossible observation. To meet this challenge, Double DQN (DDQN) algorithm [6] is proposed. There are two different sets of parameters in DDQN, θ and θ^{-}. θ is used to select the action that corresponds to the maximum Q value, and θ^{-} is used to evaluate the Q value of the optimal action. These two sets of parameters separate action selection from strategy evaluation, reducing the risk of overestimating Q value. DDQN uses parameter of the current value network to select the optimal action, and evaluates the optimal action with parameter - of the target value network. The experimental results show that DDQN can estimate the value of Q function more accurately, which makes the training algorithm and the strategy obtained by training more stable. Therefore, the $error$ part in Equation 1 is now replaced with the following expression.

$$r + \gamma max_{a'} Q(s^{'}, a^{'}; \theta^{-}) - Q(s, a; \theta) \quad (2)$$

To better evaluate the Q value, the DQN based on competition structure [7] was proposed. The main improvement is to replace the full connection layer after CNN convolution layer with two branches, one of which fits the state value function and the other fits the action dominance function. Finally, the output values of the two branches are added together to form the value of the Q function. Experiments show that the improved method can estimate the value function more accurately.

The deep neural network in DQN is convolutional neural network, which does not have long-term memory ability. To meet this challenge, a DQN-RNN algorithm [8] was proposed, which integrated cyclic neural network (RNN). This method adds a loop layer (LSTM unit) after the convolution layer of CNN, which can remember the previous information.

B. Reward Shaping Algorithms

Exploration By Random Network Distillation (RND) [3] divides rewards into intrinsic reward and extrinsic reward. Extrinsic reward is equivalent to original reward, while intrinsic reward is calculated by designing two networks and calculating their MSE output, as shown below:

$$r_t^i = \left\| f(s_{t+1}; \theta) - f(s_{t+1}^{'}; \theta) \right\|^2 \quad (3)$$

where $f(s_{t+1}; \theta)$ represents a randomly initialized neural network with fixed parameters (not changing with training), and $f(s_{t+1}^{'}; \theta)$ also represents a randomly initialized neural network but its parameters change through training.

Therefore, this loss is used as intrinsic reward, i.e., the agent will give a larger intrinsic reward when the state it sees is very different from the state it has participated in the training. Meanwhile, with the training of data samples, the network $f(s_{t+1}; \theta)$ that minimizes this MSE is trained to familiarize the agent with the existing state and reduce the intrinsic reward for the existing state.

Never Give Up (NGU) [4] adopts a similar strategy: it also introduced the intrinsic reward. In NGU, the intrinsic reward r_t^i is designed to meet two requirements: (1) In the same episode, new access to already visited state is discouraged for smart weights; (2) In different episodes, the agent is not encouraged to visit the state that has been visited for many times, that is, the agent is not allowed to bump into the place that has tasted the benefits without exploring.

Hindsight Credit Assignment [5] is proposed by Google in 2019, which proposes two "hindsight" methods, one based on future state arrival and the other based on future reward value. After receiving substantial rewards, the agent is redistributed to every (s_t, a_t) that it has experienced for reward shaping.

C. RAM as Inputs

The authors of the ALE environment [2] have been doing experiments with RAM as inputs. Instead of taking the Atari game screen frames as input, the agent directly observes the Atari machine's memory. However, the RAM inputs method did not performance well even if it is based directly on the raw game state, which is perhaps because that back then they mainly used linear function approximator instead of the deep neural network.

III. Methodology

The models are trained based on the gym [9] template, which simulates the Atari games and returns the feedback for each step/action. Here we proposed two kinds of models, one for the CNN method, and the other is based on RAM information. In both methods, we initialize all the parameters randomly. The model will play the game M steps, and store this M pair experience $E = (S_t, S_{t+1}, a, R)$ as the initial memory. Generally, M is set as 1e5, which is the maximum size of memory. After that, every time the model simulates B steps in the game and replaces the earliest B results in the memory. B is the batch size of each training. Then B samples in the memory will be picked randomly as the current training samples.

$$a_{j+1} = \begin{cases} a_{out}, \text{if } r > \epsilon_j \\ a_{random}, \text{else} \end{cases} \tag{4}$$

The ϵ-greedy is applied to control the exploration ratio. Only when the random number r is larger than the ϵ_j, the next action of the game will apply the model output a_{out}, otherwise, it will be a random action a_{random}, which aims to explore new states of the game.

$$\epsilon_{j+1} = \begin{cases} r_d * \epsilon_j, \text{if } \epsilon_j > \epsilon_{\text{min}} \\ \epsilon_{\text{min}}, \text{else} \end{cases} \tag{5}$$

After every epoch training finishes, the ϵ value will be timed by a number < 1, which let it approach the ϵ_{min} value as the training time increase. That is, the exploration ability will be reduced and stable after a certain number of training sessions.

In both methods, there are two models θ_i and θ_i^- that have the same parameters but updated at different times. θ_i^- is the target model we want to train and the θ_i model is the temporal model. Initially, their parameters are identically initialized. During the training process, the reward will be given to the target model and combined with the quality of its proposed action. Then this value will be compared with the temporal model and use MSE to get the loss. The backpropagation aims to make the temporal model similar to the target model that combined with the current reward. After n episodes, the parameters of the temporal model will be copied to the target model and repeat the training. The loss function:

$$L = E_{(s_t, s_{t+1}, a, r)}[(r + \gamma max Q(s_{t+1}, a_{t+1}; \theta^-) - Q(s_t, a; \theta))^2] \tag{6}$$

The L is the loss, E is the experience used for calculating. maxQ is obtained by the target model with intput as s_t. γ is the discount factor of the next step.

A. CNN

Compared with classical DQN, the proposed CNN has three changes.

- In the classical DQN method, a convolution layer followed by fully connected layers is applied to get the proposed action. Here we tried a different convolution structure, small size kernel, and deeper layers to train the model.

$$Q_{j+1}^{(s,a)} = Q_j^{(s,a)} + \alpha[R_j + \gamma \text{max} Q_j^{(s',a')} - Q_j^{(s,a)}] \tag{7}$$

The Q value of each state is updated by the bellman function shown in the Equation 7. aplha is the update rate and gamma is the proportion of previous time.

- The history information is added during the learning. In the classical DQN, every training sample is about one frame of the game. Nonetheless, some information, like the velocity, of the state will be missed in this way.

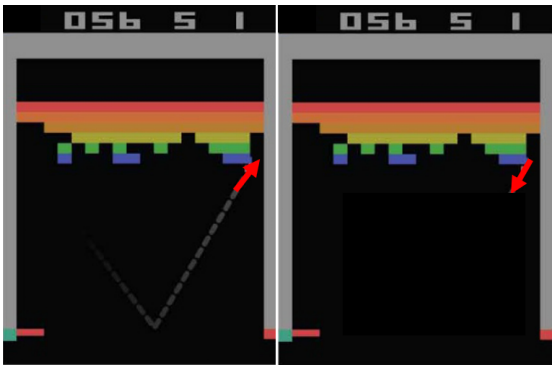

Fig. 2. Velocity information confusion.

As the picture shows in Figure 2, a single frame will not contain the velocity of the ball and the model cannot distinguish whether the ball is going up or falling down, which results in bad training. The proposed CNN model uses history to figure out the missing information. For each training step, K continue frames will be stacked together to show the temporal information.

- The RGB information in most of the games is not necessary and can be deleted. The RGB2Gray is applied in the proposed method. It converts each frame of the game from RGB to Gray before applying the convolution.

B. RAM

In the RAM-based model, the convolution layers are replaced by the RAM input, which is the information in the RAM chip during each step of the game. Compared to CNN's frame convolution, RAM information is the extracted feature itself and can be used directly. The reason is that the convolution in the CNN model aims to extract the game state feature to a lower dimension, while the RAM information has already contained these target features.

In this way, we don't need to stack multiple frames anymore because the RAM information has all the things of the current game state. For example, the velocity of the ball in game Breakout cannot be obtained by a single frame in the CNN model, but this information is already in RAM. For example, in the Figure 3, it shows the RAM information of the game Pac-Man, which contains the monster's location and plyaer's current location and lives. Then we apply the deep learning for the RAM input, the fully connection layers are the same

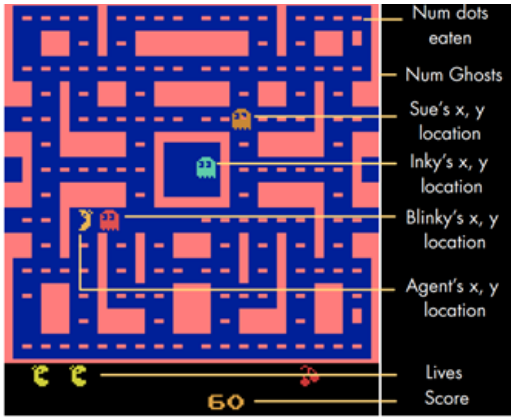

Fig. 3. RAM information of Pac-Man.

as the layers in CNN, and finally it will output score for each action for the current state.

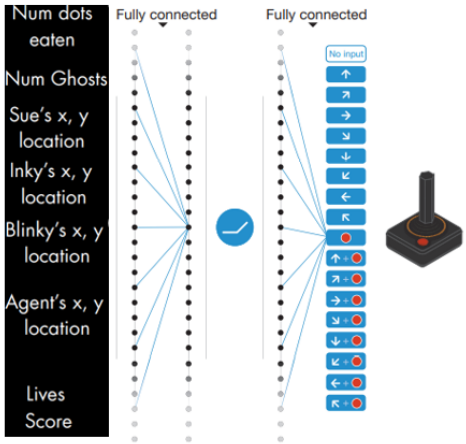

Fig. 4. RAM model structure.

IV. Experiments Settings

For the models using RAM, we used neuron network composed of four hidden layers. Suppose the len of the RAM is n, the first two layers has 4n hidden units and the following two layers has 2n units. This setting result in different structures for games which differ in action size.

For the CNN models using pictures as input, we use three layers of convolution layers. The first layer has a kernel size of 4, stride of 4 and number of output channels as 32. The second has a kernel size of 4, stride of 2 and number of output channels as 64. The third has a kernel size of 3, stride of 1 and number of output channels as 64. The convolution layers are followed by two fully connected layers, each of which has 512 hidden units.

In both of the above dqns, our reply buffer size is 10^5 bytes. The batch size when we train the network is 64. The discount factor in Bellmen equation is 0.99. Learning rate of the network is $5 * 10^-5$. We update the target network every 4 steps of training. We update our target network using soft update using parameter as 10^-3. The probability of exploring instead of going at the best action we know in the Bellmen equation is set to 1 at the start of training. It will decay by 0.005 in every step until it decays to 0.01. This value may be set to a higher value by human if the game has a significant change or the score has not been incresing for a certain number of episodes. Finally, we train our models for 20000 episodes for each game.

Also, in the validation phase, we play the game for 20 times and use the average score as the final validation score.

V. Results & Discussion

We trained our agent in 5 games. "Breaking-out","Ping Pong", "Montezuma Revenge", "Boxing" and "Seaquest". Among these games, the reward of "Boxing" is in short term, which means we will have nearly instant reward when we do the right action. At the other end of the spectrum, the reward of "Montezuma Revenge" comes particularly late, which is the sparse reward problem. Here are our results:

TABLE I
Results of the RAM model and baselines

GAME	Breakout	Boxing	Pong	Seaquest	Montezuma
Random play	1.7	0.1	-20.7	68.4	0
Best Linear	5.2	44	-19	664.8	10.7
SARSA	6.1	9.8	-17.4	675.5	259
Human	31.8	4.3	9.3	20182	4367
Standard DQN	401.2	71.8	18.9	5286	0
Our model	29	85.61	-9	172.4	0

As we can see from the results, our results are mostly worse than the standard dqn and better than other model from previous times. This limited result is mainly due to the limited computational resources we have. We need 170 hours for a good (not perfect) breakout game even with a good GPU.

Also, here is the score during the training process for two of the games.

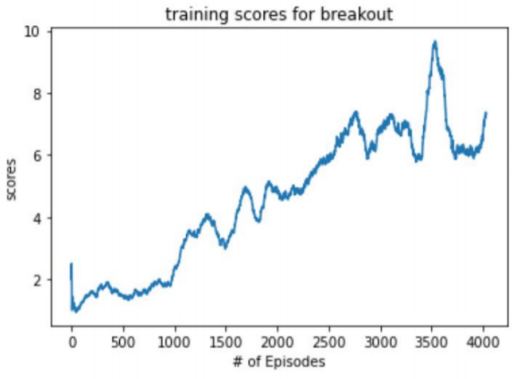

Fig. 5. training score for break out

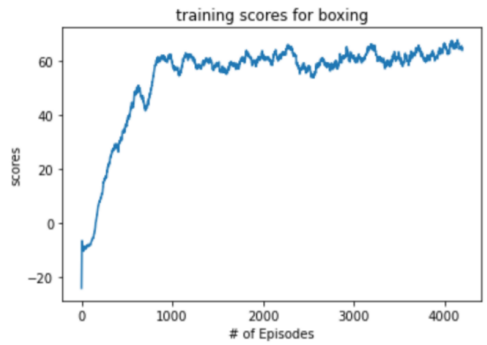

Fig. 6. training score for boxing

Since the DQN algorithm is just arbitrary exploration until there is some rewards. And we need to wait for the rewards propagate to the previous states which actually leads to that rewards through Bellmen equation. Therefore, the computation we need grows exponentially when the latency of the rewards increase. The "Boxing" game has a particularly low latency in reward, therefore it is easy to train an agent and the score converges quickly. In a game like "breakout", with some latency of rewards, it is much harder to train and the model does not converge at the end. For the "Montezuma's Revenge" game, with a much larger latency, it is nearly impossible to train a model using standard DQN. It requires more advanced algorithm or human interfere when training the model. This is why the standard DQN even with enough computational resource cannot get a good result out of it.

ACKNOWLEDGMENT

This work was supported by the Zhejiang University /University of Illinois at Urbana-Champaign Institute, and was led by Prof. Zuozhu Liu and Prof. Gaoang Wang. All the authors collaborate together to finish this work and made equal contributions.

REFERENCES

[1] Mnih Volodymyr, Kavukcuoglu Koray, Silver David, Graves Alex, Ioannis Antonoglou, W Daan, and R Martin. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*, 2013.

[2] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[3] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

[4] Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martín Arjovsky, Alexander Pritzel, Andew Bolt, et al. Never give up: Learning directed exploration strategies. *arXiv preprint arXiv:2002.06038*, 2020.

[5] Anna Harutyunyan, Will Dabney, Thomas Mesnard, Mohammad Gheshlaghi Azar, Bilal Piot, Nicolas Heess, Hado P van Hasselt, Gregory Wayne, Satinder Singh, Doina Precup, et al. Hindsight credit assignment. In *Advances in neural information processing systems*, pages 12488–12497, 2019.

[6] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[7] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[8] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*, 2015.

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.