Playing FPS Game with Reinforcement Learning

Tianshi Xie
Computer Science, Troy University
Troy, United States
txie@troy.edu

Figure 1 *www.nvidia.coms*

Abstract— Our goal is to train a robot or an agent to play a video game like a human. How does the human play a video game? When we play a video game, first we will receive the visualizing information and then translate into the sensory information. We can assume if an enemy fire a bullet at you, how do you do? Pressing the button to avoid, against or using some skills such as magic shield to protect yourself. Someone may say that I will stay there and choose nothing. Yes, you can, but you will lose in the game. So, that's how the human play a video game. How does the robot to play a video game? First, it will receive some information and translate into some actions such as pressing the button. And then get a feedback from the game machine we also call it reward. According this reward, the robot will know whether the last action is good or bad. Finally reprocessing the information to generate a better action next time. Applying deep reinforcement learning to Atari game has outperformed all previous method, even surpassed a human expect. However, most of these methods usually are used in 2d environment that hardly simulate the real world. This paper will

present an amazing architecture that robot can play an FPS 3d video game like a human.

Keywords—Reinforcement Learning.
Convolutional Neural Network. Recurrent Neural Network. LSTM. Q-Learning

- Reinforcement learning(RL) is an area of machine learning illuminated by behavior. Putting a robot or an agent into an unknown environment so to maximize the reward which is the feedback from the environment after doing some actions.

- A convolutional neural network (CNN, or ConvNet) is a class of deep neural networks that can efficiently recognize the visual imagery.

- A recurrent neural network(RNN) is a class of neural network which connect the last state to deal with a sequence efficiently.

- Long short-term memory(LSTM) is a recurrent neural network(RNN) structure that remembers values over arbitrary intervals.

- Q-Learning is algorithm by using bellman equation to maximize the sum of total rewards

I. Introduction and Related Work

Deep reinforcement learning has an excellent performance in learning control policies from high-dimensional sensory input. Particularly in DQN, which has proved to be effective in playing Atari games, in surpassing some human experts such as AlphaGo.

Although DQN is enough to deal with fully observable states. It will not available to the partially observable states since the agent need to remember last states to be able to choose optimal actions. Not long ago, some experts have been attempted to solve this problem by using deep recurrent neural network, especially a Long Short Term Memory network. The effect of recurrent neural network is efficient because of that RNN is good at remember previous information.

In this paper, we will train a robot to play a FPS 3D game. That is more complex than Atari games since it includes a lot of actions, for instance, Identifying and beating enemy, Picking up items, and navigation.

Our method is to only use the pixels on the game as input data for the agent. The agent will process two parts: navigation (exploring the map to collect items and find enemies) and action (fighting enemies when they are observed), and uses separate networks for each phase of the game. Furthermore, the agent infers high-level game information, such as the presence of enemies on the screen, to decide its current phase and to improve its performance.

Background

A. Deep Q-Networks:

Reinforcement learning perform a policy for an interaction between an agent and an unknow environment. Taking every step, an agent receives a state from the environment, and then generate an action according to a policy and a reward.

It's usually do through a function to estimate the Q value.

$$Q^*(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q^*(s',a') | s,a\right]$$

Q star means this action in the states has the highest action value, and then the agent will select this action as the current action. How to calculate the Q value? We are going to use this policy which is the current reward r plus a Bellman equation. The most interesting part of the Q learning is that it will estimate the q value of next state first, and the result as the parameter in the policy, and then output the q value of current state. Υ is a discounted factor that to control the future reward.

We will use a neural network in DQN. we will modify the weight constantly to obtain an estimate Q value which is close to the Q star value. In fact, Q star is a label, Q Θ is an estimate value. Our goal is to find Θ like $Q_\Theta(s; a) \approx Q\_(s; a)$. In other words, the loss which is the difference between Q_Θ and Q_ is nearly equal to 0.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for episode = $1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Figure 2 https://www.cs.toronto.edu/

Figure 2 show that is the DQN's Basic algorithm. First, we will initialize the memory D and random weights. To the episode, it presents how many times the agent plays the game. For t equal one to T, it means how many frames the agent has played. If the agent wins or loss in the game, it will break current loop and restart a new game as mean well the episode increments. In each frame, the agent will select an action according to a factor e. we usually set this variable e to 0.9. we will random a value and compare the current value with e. the condition is that: with probability e select a random action at. Otherwise select an action which has the highest value from the memory D we mentioned above. Next, through the action into the game environment to get a feedback we also call it reward that will make the agent to know whether the last action is good or bad. According to the feedback, we will generate a sequence consist of current state, current action, current reward, and next state, and then store it into the memory D. After collecting memory, the agent begins to learn the experience from the memory according the Q-Learning algorithm in the Deep neural network. First sample random minibatch of transitions from D. Second, perform a gradient descent step to update the weights which are the most valuable we will get finally.

In the training process, product the next action by using greedy strategy: with a probability _ the next action is selected randomly, and with probability 1-E according to the network best action. In practice, it is common to start with E = 1 and to progressively decay E.

Why do we need to select action randomly? Because the agent can obtain new knowledge in a unknown environment by random action. The probability E will decay with the ability of agent become more powerful. So, in the real game test, it is common to set E = 0 since we do not need the random action at all. Only according to the neural network to get the best result in the game since the random action will lead to fail in the game.

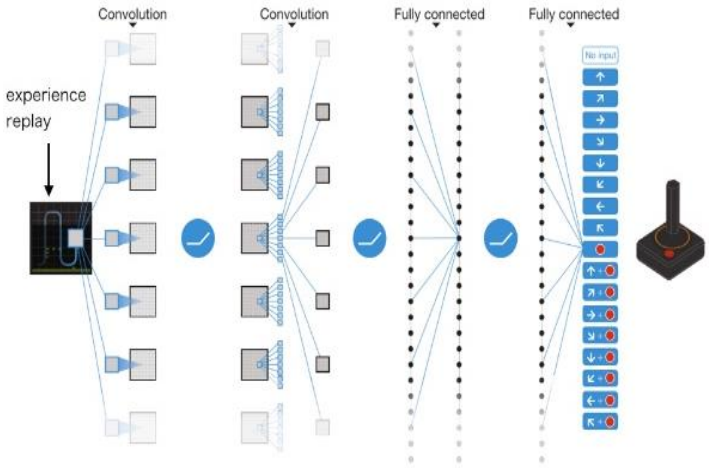

Figure3 https://www.google.com/search

B. Deep Recurrent Q-Networks

An agent usually cannot infer the full state of the system by only receive one state of the game environment each step since the game still contain some hidden states. In such case, we have to introduce the Deep Recurrent Q-Networks(DRQN) whose theory is that using formula Q (o_t, h_{t-1}, at),

where it is an extra input returned by the network at the previous step, that represents the hidden state of the agent. Here we didn't use the original model of RNN, but LSTM which is more powerful than the basic version can efficiently avoid gradient vanishing, gradient explosion according to his structure:

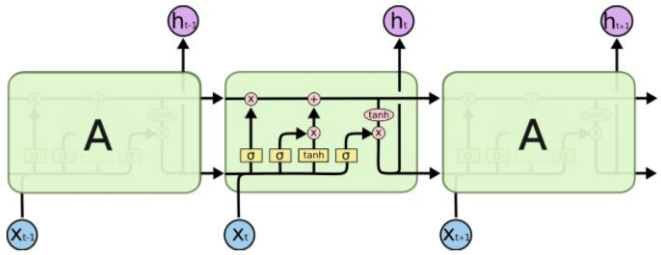

Figure 4

This architecture consists of three gates: forget gate, input gate, and output gate can delay the decay in memory.

So, we use the basic DRQN model to train the agent play FPS game. We found a strange phenomenon that the agent was firing randomly.

Model

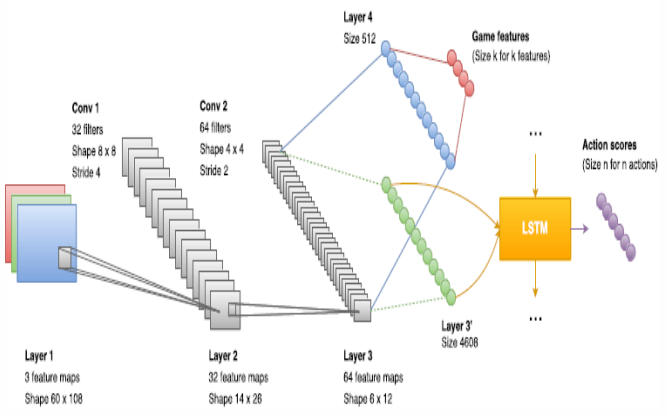

Figure 5 https://www.google.com/search

The final input is a 60*108 region of the image that roughly captures the playing area. The first hidden layer convolved 32, 8 * 8 with stride 4, the second hide layer convolved 64 6*12 with stride 2. The output of the second hide layer is divide into two parts. The first one is feed to the game feature. The second one is feed to a LSTM.

II. DIVIDE AND CONQUER

A. *Game feature augmentation*

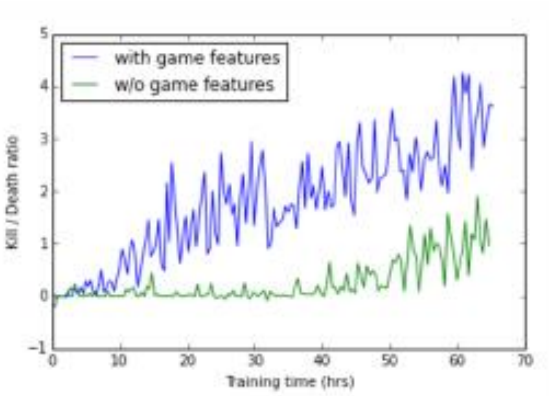

Figure6

We assume that the agent was not able to identify enemies. So, we create an interface in the game engine which can indicate whether there are visible objects in each frame or not (an object can be a gun, pack, enemy, bullet, and so on). Although we cannot use the internal information at test time, it absolutely can be used in training time.

In the original model, an LSTM was feed by the output of a CNN with a single output for each action according to the current frame and its previous information. Sharing the convolutional layers is the key in improving the performance of the model.

In this experiment: spending a few hours for this model to achieve a 90% accuracy in detecting whether this entity is enemy or not. And then, some features which often involves entity's position**,** presence of entity will be inject into the LSTM, resulting in accelerated training.

B. *Training*

1) *Reward shaping*

How to define the reward is the key in training, which can tell the robot whether the last action is good or not in the reinforcement learning. Defining the score as a reward is very difficult for the robot to learn some useful actions. However, reward may be delayed and are not the result of some actions. For example, to get a score reward requires the robot detect the map to find an enemy and fire a bullet at enemy. It's difficult for the robot to learn which action is corresponding to what reward. To deal with this problem of delayed rewards, we propose reward shaping, we will modify the reward function to involve some detailed reward to accelerate the learning process:

- Collecting object which are health package, gun and bullet will get a positive reword.
- Blood lose which is standing on the dangerous area or is coming under fire will get a negative reward.
- Firing or loosing bullet will get a negative reward.

In addition to that we mentioned above. It's very useful to give a small positive reward according to the distance the robot travelled. This will result in detecting the map fast.

2) Frame skip

In this method, the robot is not going to receive an image every frame because it will lead to image delay especially in HD game. We introduce a technique that is called Frame skip, which means the robot receive an image every k frame that can reduce the computation efficiently. However, the problem is that a higher frame will lower the performance, otherwise, to lower the player experience and training speed. The number of frame skip is equal to 4 will be the best balance.

3) Sequential updates

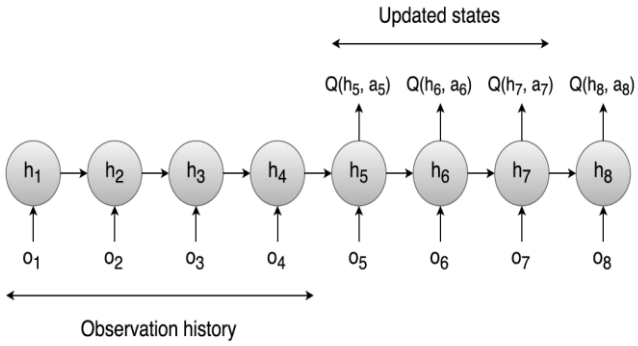

Figure7

The original method is to sample a sequence of observations from the replay memory and update all the states in the sequence. To improve the performance, we propose another approach that only considering the states which provide enough history to be updated. In fact, the first states will be updated from a non-existent history. Updating these states might lead to inexact updates.

To avoid this problem, having a backpropagation on the errors from states O_h to O_n, where h is the minimum history size of a state and O_n is used for O_{n-1} state. The Figure 6 show that setting the minimum history size is equal to 4, and updating 5 states here.

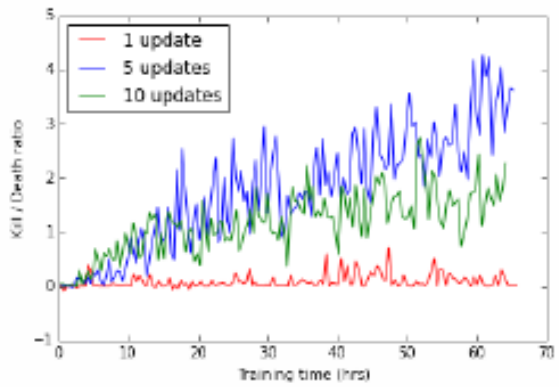

Figure8

We try to increase or decrease the update number, and found that is difficult to get a good policy. Finally, we found the number of updated state is 5

to be a best tradeoff in this architecture and the result is shown in Figure7.

.

III. FUTURE WORK

As a matter of fact, after understanding the whole architecture which has three deep neural networks, we can use TensorFlow which has an excellent performance in Deep neural network to implement it.

A. Create the first neural netwok CNN to detected whether there are enemy in current frame.

We can use the internal variable in the game environment to help the robot to identify whether the entity is enemy or not in the training time. And then it can work well in the testing time.

The output of this convolutional neural network is divided into two parts. One is used for game features, another one is shared by Deep Q-Learning Network(DQN) and Deep Recurrent Q-Learning Network.

B. Adding the second neural network DQN

To perform some basic task such as explore the map, and pick up some items.

C. Adding the third neural network DRQN

To deal with advanced task which is fighting enemies.

D. Improving the performance

We will improve the performance from two aspects. One is overestimate, another is speeding up training.

- DDQN

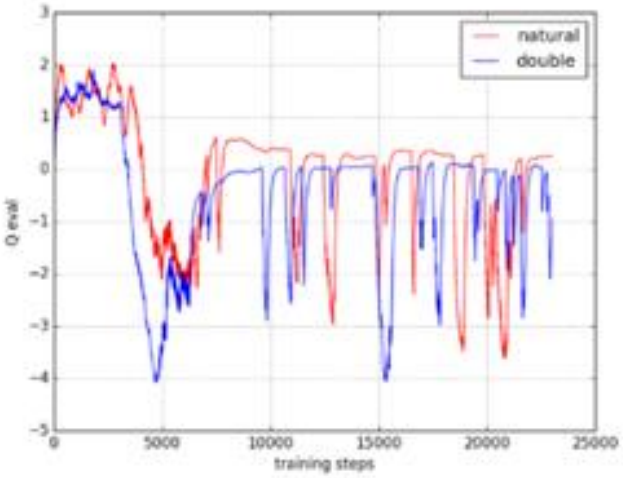

Figure9

Figure8 is shown that the original one is the red line which has high estimate value. After using Double DQN, the result is the blue line which is lower than the red one.

- Prioritized Experience Replay DQN

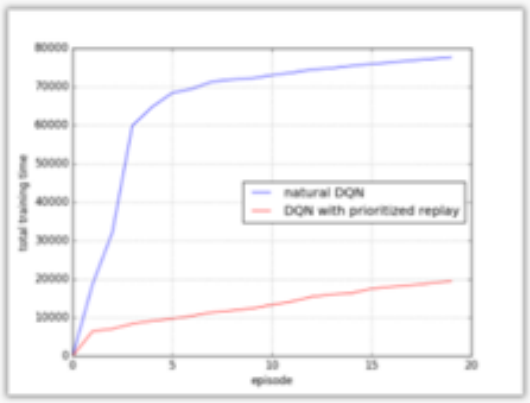

Figure10

Figure9 is shown that the original one is the blue line which has the high cost time. After using Prioritized experience replay, we get an excellent result which is the red line.

CONCLUSION

We had illustrated that our goal is to train a robot to play a video game like a human, even surpass a

human expert. To a simple 2D game, an DQN is enough. But to a complex 3D game is impossible. So, we introduce our new model which contain three neural networks both DQN and DRQN shared an CNN and works very well. The rate of detecting enemies can reach above 90%. And also present a lot of details such as reshape reward, frame skip, sequence update and so on. In addition to what has already been said, I propose a portion that is improving the performance of this architecture from two aspects on this architecture. One is double deep Q-network (DDQN) and another is to speed up training by using prioritized experience replay.

KNOWLEDGMENT

Many thanks to Dr.Yanjun Zhao. The subject of this thesis was challenging to me. She made this experience really valuable for me. Thank you so much!

REFERENCES

[1] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In Proceedings of the 12th International Conference on Machine Learning (ICML 1995), pages 30–37. Morgan Kaufmann, 1995.

[2] Marc Bellemare, Joel Veness, and Michael Bowling. Sketch-based linear value function approximation. In Advances in Neural Information Processing Systems 25, pages 2222–2230, 2012.

[3] 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.

[4] Marc G Bellemare, Joel Veness, and Michael Bowling. Investigating contingency awareness using atari 2600 games. In AAAI, 2012.

[5] Marc G. Bellemare, Joel Veness, and Michael Bowling. Bayesian learning of recursively factored environments. In Proceedings of the Thirtieth International Conference on Machine Learning (ICML 2013), pages 1211–1219, 2013. 8

[6] George E. Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. Audio, Speech, and Language Processing, IEEE Transactions on, 20(1):30 –42, January 2012.

[7] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. In Proc. ICASSP, 2013.

[8] Matthew Hausknecht, Risto Miikkulainen, and Peter Stone. A neuro-evolution approach to general atari game playing. 2013.

[9] Nicolas Heess, David Silver, and Yee Whye Teh. Actor-critic reinforcement learning with energy-based policies. In European Workshop on Reinforcement Learning, page 43, 2012.

[10] Kevin Jarrett, Koray Kavukcuoglu, MarcAurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In Proc. International Conference on Computer Vision and Pattern Recognition (CVPR 2009), pages 2146–2153. IEEE, 2009.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems 25, pages 1106–1114, 2012.

[12] Sascha Lange and Martin Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In Neural Networks (IJCNN), The 2010 International Joint Conference on, pages 1–8. IEEE, 2010.

[13] Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, DTIC Document, 1993.

[14] Hamid Maei, Csaba Szepesvari, Shalabh Bhatnagar, Doina Precup, David Silver, and Rich Sutton. Convergent Temporal-Difference Learning with Arbitrary Smooth Function Approximation. In Advances in Neural Information Processing Systems 22, pages 1204–1212, 2009.

[15] Hamid Maei, Csaba Szepesvari, Shalabh Bhatnagar, and Richard S. Sutton. Toward off-policy ´ learning control with function approximation. In Proceedings of the 27th International Conference on Machine Learning (ICML 2010), pages 719–726, 2010.

[16] Volodymyr Mnih. Machine Learning for Aerial Image Labeling. PhD thesis, University of Toronto, 2013.

[17] Andrew Moore and Chris Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. Machine Learning, 13:103–130, 1993.

[18] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th International Conference on Machine Learning (ICML 2010), pages 807–814, 2010.

[19] Jordan B. Pollack and Alan D. Blair. Why did td-gammon work. In Advances in Neural Information Processing Systems 9, pages 10–16, 1996.

[20] Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In Machine Learning: ECML 2005, pages 317–328. Springer, 2005.

[21] Brian Sallans and Geoffrey E. Hinton. Reinforcement learning with factored states and actions. Journal of Machine Learning Research, 5:1063–1088, 2004.

[22] Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala, and Yann LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In Proc. International Conference on Computer Vision and Pattern Recognition (CVPR 2013). IEEE, 2013.

[23] Richard Sutton and Andrew Barto. Reinforcement Learning: An Introduction. MIT Press, 1998.

[24] Gerald Tesauro. Temporal difference learning and td-gammon. Communications of the ACM, 38(3):58–68, 1995.

[25] John N Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. Automatic Control, IEEE Transactions on, 42(5):674–690, 1997.

[26] Christopher JCH Watkins and Peter Dayan. Q-learning. Machine learning, 8(3-4):279–292, 1992.