Playing FPS games with Deep Reinforcement Learning

Lennart Faber (s2500253) [a] Rohit Malhotra (s3801128) [a]
Yifei Chen(p285442) [a] Yuzhe Zhang(p186609) [a]

[a] *University of Groningen*

Abstract

This paper aims at comparing multiple Deep Reinforcement Learning methods to play first person shooting game in 3D environment. We describe 3 setups, which are Deep Q-Networks (DQN) with Experience Replay, Deep Recurrent Q-Networks (DRQN) with augmented features and Asynchronous Advantage Actor-Critic (A3C). We use CNN for feature extraction. Out of these three setups A3C outperformed the other two.
Keywords: Deep Reinforcement Learning, Deep Q-Learning, A3C, DRQN, DQN, Deep Learning, Doom, FPS-games.

1 Introduction

With the steady increase of computing power and recent developments in deep learning techniques [1], it has become apparent that computers are capable of performing more and more tasks that were previously believed to be too complex for non-humans too complete. As games require considerable strategies, experience and skills, they have always risen much interest from researchers and much work has been done on training agents to play games like Atari[2], AlphaGo[3] and Doom[4]. In this paper, we discuss how algorithms based on a combination of Convolutional Neural Networks (CNNs) [5] and Reinforcement Learning (RL) [6] are used to train agents that are capable of matching and even outperforming humans in playing first-person shooter (FPS) games. These Deep Reinforcement Learning (DRL) [7] networks have produced impressive results when used to play simple video games before [8], but researchers are still facing an interesting challenge in playing FPS games since agents do not have complete knowledge about states.

In this project, we realize learning algorithms to train agents to play a FPS game called DOOM[1], which was originally released in 1993. Since it was one of the first FPS games ever released, its relatively simple environment and controls provide a setting in which DRL algorithms can be trained and evaluated with reasonable computational costs. The architectures utilized in this project are Deep Q Networks (DQN), Deep Recurrent Q-Networks (DRQN) with Augmented Features[9], and Asynchronous Actor-Critic Agents (A3C)[10]. We realize these architectures, and conduct comparison among them by measuring the rate of kill and death and with the training time. A CNN is used to extract features from each frame of the game and these features are input vectors to the networks.

2 Method

2.1 Simulator

Doom is an First person shooter (FPS) game developed by Id-software. Research on Doom has become a hot area of research using techniques like deep reinforcement learning or visual reinforcement learning.

[1] https://en.wikipedia.org/wiki/Doom_(franchise)

ViZDoom which is based on ZDoom provides us the game mechanics. It is a Doom-based platform which is developed for implementing deep learning techniques or methods [11].

2.2 DQN

A deep Q-Network is the combination of CNN with a Q-network. A Q-network is the implementation of Q-learning using multi-layer fully connected network. We use CNN to extract features from a 2D/3D image, then we flatten these features to a 1D input vector which becomes input to a fully connected network. Let's say our game has n possible actions, then the output layer of our DQN will have n units, each one will correspond to each action played in the current state. Action played will be the one that corresponds to the output unit with highest value(Q-value) or one returned by the softmax method. Hence in each state s_t of the game:

- The prediction is $Q(s_t, a_t)$, where a is the action choosen by argmax or softmax.
- target is $r_t + \gamma \max_a Q(a, s_{t+1})$.
- The loss is squared of this temporal difference.

$$Loss = (r_t + \gamma \max_a Q(a, s_{t+1}) - Q(s_t, a_t))^2/2 \tag{1}$$

Then this loss is backpropagated through the network and weights are calculated.

Experience Replay: As it is highly likely that the state s_t is highly correlated with the state s_{t+1}, so by using the batch of correlated states and then backpropagating the loss, the network will not learn much. This could be improved if we store last m experiences of the agent, where m is a large number and experience is defined as tuple of (state,action,reward,next state). These last m transitions is called Experience Replay and storage of them is called Experience Replay Memory. Then from this Experience replay we randomly draw batches of transitions to make our updates.

2.3 DRQN with Augmented Features

The simple DQN model assumes that at every step agent will receive the full information about the current state. But usually that is not the case as screen buffer can not represent the complete state, there can still be hidden variables. Specially in the FPS game like Doom, the agents field of view is limited, centered around its position.

To deal with such situation we use the Deep Recurrent Q-Netoworks (DRQN), which will not estimate the $Q(s_t, a_t)$, but insted they will estimate $Q(o_t, h_{t-1}, a_{t-1})$, where h_{t-1} is an extra input, which was returned by the network at the previous step. h_{t-1} can represent the hidden variables. We have used an f Long-Short-Term Memory Network (LSTM)

The Deep Recurrent Q-Networks with Augmented Features are based on Deep Reinforcement Learning and it makes use the advantages of Long-Short-Term Memory Networks (LSTM) and Convolutional Neural Networks (CNN). As shown in Fig 1, the input image is fed into convolutional layers and then the output is split to two streams, one of which is flatted and fed into the LSTM to obtain the scores for various of actions, while the other stream is fed into two full connected network (Layer 4 and the right layer) and it gives k game features, e.g., whether the visible entity is an enemy or not, and the position of the enemy. The architecture uses the same CNN in DRQN to capture the relevant game information, which enhances the accuracy of DRQN considerably while costs little additional running time.

2.4 A3C

The algorithm, which is called asynchronous advantage actor-critic(A3C), is one of the asynchronous methods for deep reinforcement learning described in [10]. It is a state of the art method, in which we have several agents, each one interacting with its own copy of the environment. Let us assume we have

[2]Figure taken from [9]

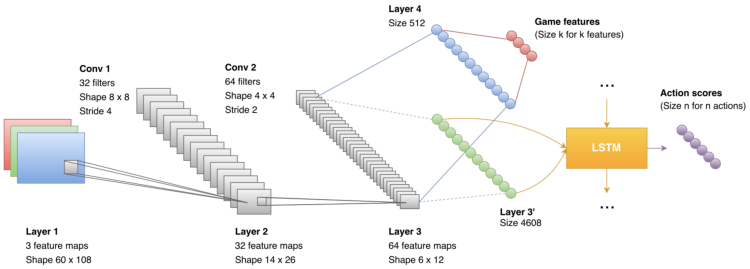

Figure 1: DRQN [2]

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ *and* θ_v *and global shared counter* $T = 0$
// Assume thread-specific parameter vectors θ' *and* θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial\,(R - V(s_i;\theta'_v))^2/\partial\theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

Figure 2: A3C algorithm [3]

n agents $A_1, A_2, ...A_n$. Each agent shares two networks, the actor network and the critic network. Role of the critic is to evaluate the value of the present state($V(s)$) and the role of the actor is to evaluate the policy (a set of action probability outputs using Q values of the actions in the present state)$\pi(s)$. Decision is made by the actor. At each epoch of the training for an agent, it takes most recent values of the weights of the network and uses the actor to play for n steps. Over these n steps, it collects all the values of the new states, rewards, etc. This is called the experience of the agent. After the n steps agents uses this collected information to calculate the discounted return(R) and advantage(A), and use those to calculate value and policy losses(2,3). We also calculate the entropy(H) of the policy(π).

$$ValueLoss = \sum (R - V(s))^2 \tag{2}$$

$$PolicyLoss = -log(\pi(s)) * A(s) - \beta * H(\pi) \tag{3}$$

The combined value and the policy loss is used to update the network weights. As the time for observing hence the time for update for each agent will not be same hence it is has name asynchronous.

The A3C process: In order to understand the update process of the algorithm, let us define θ as parameters for actor network and θ_v as parameters for critic network. Pseudo code of the algorithm is given in the Figure 2. Figure 3 clearly describes three As of the A3C:

- **Asynchronous**: There are several agents and all are observing different environments and all are asynchronous as each is playing at different time. This speedups the training as more work is getting done in parallel. As experience of each agent is independent of each other, therefore overall experience available for training becomes more diverse. Also if any unlucky agent that starts to get stuck in a sub-optimal policy, then updates from the other agents will not let that happen.

[3] Figure taken from [10]

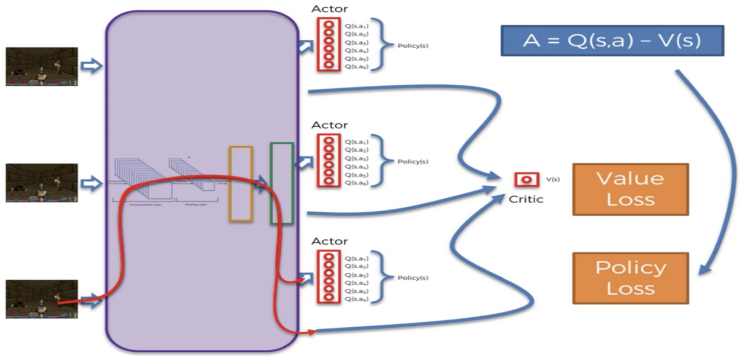

Figure 3: A3C, this figure describes shared actor and critic network between agents.[4]

- **Advantage**: It is defined as the difference between the Q value of the action taken in the state and value of the state.

$$A = Q(s, a) - V(s) \tag{4}$$

  It represents how much better the action was as compared to the expected. Since $Q(s, a)$ is estimated as discounted return(R) over n steps hence 5 defines advantage(A).

$$AdvantageEstimate = R - V(s) \tag{5}$$

- **Actor-Critic**: Policy loss is related to the actor network and the value loss is related to the critic network. So critic helps agent to update the values more intelligently than without the value estimate.

2.5 Experimental Setup

Environment: We have used Python with PyTorch [5] as deep learning framework. Training and evaluation of A3C algorithm is done on CPU (Intel Core i5, 4300U), because of shared memory constraints. Training and evaluation of DQN and DRQN with augmented features is done on GPU (NVIDIA [GeForce GTX 1070 Ti]).

Game: We have trained and evaluated deathmatch scenario of Doom. Training and evaluation are done on the same maps, with the map id 1. In deathmatch scenario we have allowed 9 action buttons which are move forward or backward, turn left or right, move left or right , speed, crouch and shoot. Based on these buttons there are 53 possible actions, which are combination of these. We have trained and evaluated with 8 built in bots and it has only "enemy" as a feature. Rewards are defined as follows: 'BASE REWARD': 0.0, 'DISTANCE': 0.0, 'KILL': 5.0, 'DEATH': -5.0, 'SUICIDE': -5.0, 'MEDIKIT': 1.0, 'ARMOR': 1.0, 'INJURED': -1.0, 'WEAPON': 1.0, 'AMMO': 1.0, 'USEAMMO': -0.2.

Training and Evaluation: We have trained DQN and DRQN methods with RMSProp [6] optimizer and A3C with Adam[7]. Gamma value was 0.99 and learning rate was 0.001. The size of replay memory for DQN was 1000000 with batch size of 32 for DQN and DRQN. For A3C number of steps for calculating discounted reward(R) was 20. After every 20,000 transitions model was evaluated for 15 mins and weights were saved. Number of asynchronous agents in A3C were 4.

We used ϵ-greedy policy during the training for DQN and DRQN, where ϵ was linearly decreased from 1 to 0.1 over the first million steps, and then fixed to 0.1. We used a screen resolution of and resized each frame to 42x42 image before passing it to the model for all the setups.

[4] From https://www.udemy.com/artificial-intelligence-az/
[5] https://pytorch.org/
[6] https://en.wikipedia.org/wiki/Stochastic$_gradient_descentRMSProp$
[7] https://en.wikipedia.org/wiki/Stochastic$_gradient_descentAdam$

3 Results

Evaluation metric: for evaluation in deathmatch scenario we use Kill to death (K/D) ratio. A Kill is number of enemy bots killed and death is total number of deaths and suicides. A suicide is considered when an agent shoots too close to itself, within the blast radius of the weapon. Since suicides are part of deaths so they are a good way to penalize when agent is shooting randomly.

4 Discussion

References

[1] L. Deng, "A tutorial survey of architectures, algorithms, and applications for deep learning," *APSIPA Transactions on Signal and Information Processing*, vol. 3, 2014.

[2] Wikipedia contributors, "Atari games — Wikipedia, the free encyclopedia," 2019, [Online; accessed 4-July-2019]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Atari_Games&oldid=899276958

[3] ——, "Alphago — Wikipedia, the free encyclopedia," 2019, [Online; accessed 4-July-2019]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=AlphaGo&oldid=903968936

[4] ——, "Doom (franchise) — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=Doom_(franchise)&oldid=902247873, 2019, [Online; accessed 4-July-2019].

[5] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[6] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[9] G. Lample and D. S. Chaplot, "Playing FPS games with deep reinforcement learning," *CoRR*, vol. abs/1609.05521, 2016. [Online]. Available: http://arxiv.org/abs/1609.05521

[10] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783

[11] K. Adil, F. Jiang, S. Liu, A. Grigoriev, B. Gupta, and S. Rho, "Training an agent for fps doom game using visual reinforcement learning and vizdoom," *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 12, 2017.