Deep Reinforcement Learning with VizDoom First-Person Shooter*

Dmitry Akimov[1] and Ilya Makarov[1][0000−0002−3308−8825]**

National Research University Higher School of Economics, Moscow, Russia
deakimov@edu.hse.ru,iamakarov@hse.ru

Abstract. In this work, we study deep reinforcement algorithms for partially observable Markov decision processes (POMDP) combined with Deep Q-Networks. To our knowledge, we are the first to apply standard Markov decision process architectures to POMDP scenarios. We propose an extension of DQN with Dueling Networks and several other model-free policies to training agent using deep reinforcement learning in VizDoom environment, which is replication of Doom first-person shooter. We develop several agents for the following scenarios in VizDoom first-person shooter (FPS): Basic, Defend The Center, Health Gathering. We compare our agent with Recurrent DQN with Prioritized Experience Replay and Snapshot Ensembling agent and get approximately triple increase in per episode reward. It is important to say that POMDP scenario close the gap between human and computer player scenarios thus providing more meaningful justification for Deep RL agent performance.

Keywords: Deep Reinforcement Learning, VizDoom, First-Person Shooter, DQN, Double Q-learning, Dueling

1 Introduction

First-person shooter is a type of video games, in which a computer player avatar under human control competes with other agents and human players, while having visual representation if a human sees from avatar's eyes perspective. The simplest elements of playing FPS are passing the maze, collecting bonuses and cooperating and fighting other players with a ranged weapon. FPS games are demanding for players' skills and are hard to train for supervised AI models with passive learning.

Recently, models of supervised active learning, such as Reinforcement Learning (RL), achieve state-of-art results in many computer vision related tasks, such as robotics, path planning and playing computer games on human level. FPS in RL framework is presented as complex state environment with finite control actions and certain goal, for example, to maximize kill-death ratio during one game

** The work was supported by the Russian Science Foundation under grant 17-11-01294 and performed at National Research University Higher School of Economics, Russia.

session or episode, which then may be divided to subgoals, such as map navigation, health pack collection, weapon use or enemy detection. Learning agent policy in 3D FPSs using RL is computationally hard: rewards are sparse and usually highly delayed (player not always one-shot killed). Moreover, an agent do not know complete information on the environment in order to model human decision making: enemies positions are unknown and angle of view is limited by 90-110 degrees according to human perception. The only information that Deep RL agent can use for action choice is a game screenshot (image captured from rendered scene), so even navigation in 3D maze is challenging and involves storage of known routes, planning, and proper feature extraction to learn map without knowing its navigation mesh.

We choose the VizDoom [7] as simulation environment with three core scenarios with different goals and action sets: Basic, to learn navigation and monster detection; Defend The Center, to learn accurate aiming and so, improve enemy detection and ammo preserving; and Health Gathering, to learn detecting and collecting health packs under navigation in critical environment with acid on the floor.

We combine several existing MDP models for RL agents learning in POMDP and present a new model-free Deep RL agent, which showed its efficiency compared to other Deep RL models applied for VizDoom agents.

2 Related work

Here, we give an overview of existing models following our previous study on one scenario presented in [1].

2.1 Rainbow

In 'Rainbow' paper [5], authors combined together with DQN such models, as Double Q-Learning, Dueling Architecture, Multi-step learning, Prioritized Replay, C51 and Noisy Networks for exploration. They achieve 8 times faster learning than just DQN alone in The Arcade Learning Environment [3] using MDP setting, but performance in POMDP setting is unknown.

2.2 Arnold

In 'Arnold' paper [8], authors develop Deep RL agent to play Deathmatch scenario in VizDoom environment augmenting agent with in-game features during training, enemy-detection, reward shaping for subgoals, separate networks for action and navigation, and dropout for reducing overfitting of convolutional feature extraction. Their agent substantially outperforms deterministic agent for computer players and even humans.

However, 'Arnold' agent did not exploit Q-learning extensions that could significantly improve agent's performance, and the way they training the agent goes against overall framework of Deep RL agents' training in POMDP.

2.3 DRQN with Prioritized Experience Replay, Double Q-learning and Snapshot Ensembling

Prioritized Experience Replay (PER) [11] is a way to speed-up training. Samples of Experience Replay are taken with non-uniform probabilities for each tuple $\langle o, a, r, o' \rangle$ in accordance to higher loss values cause they carry more information than those with low loss value, which was proved to work well in Atari games.

Authors of [12] combined PER with Double Q-learning and Snapshot Ensembling and tested their agent in VizDoom Defend The Center scenario. The authors train enemy detector and Q-function in a joint manner; this model was chosen for baseline in our case. Deep Reinforcement Learning improvements for MDP achieved super-human performance in many games [10,9]. However, this improvements had not been considered in POMDP setting before [1]. We combine such frameworks from MDP with POMDP to improve state-of-the art methods considering several scenarios for FPS according to VizDoom DRL competition.

3 Deep Reinforcement Learning Overview

General reinforcement learning goal is to learn optimal policy for an agent, which maximizes scalar reward by interacting with the environment.

At each time step t an agent observes the state s_t of the environment, makes a decision about the best action a_t according to its policy π and gets the reward r_t. This process well known as *Markov Decision Process* (MDP) denoted as a tuple (S, A, R, T), where S indicates a finite state space, A indicates a finite action space, R is a reward function, which maps pair $(s, a) \in (S, A)$ into stochastic reward r, and last but not least T is a transition kernel: $T(s, a, s') = P(S_{t+1} = s' | S_t = s, A_t = a)$.

Discounted return from state s_t is expressed by the following formula:

$$G_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i},$$

where $\gamma \in (0, 1)$ is a discount factor reducing the impact of reward on previous steps. Choosing γ depends on game duration, providing advanced strategies for long game sessions when trained with greater values of gamma, while small gamma provides short-term rewards.

In order to choose the actions, an agent uses a policy $\pi = P(a|s)$. We call a policy π^* *optimal* if it maximizes expected discounted reward :

$$\pi^* = \operatorname*{argmax}_{\pi} \mathbb{E}_{\pi}(G_t)$$

We consider *Q-learning* as core method for training RL agents.

3.1 Q-learning

To measure the quality of a given policy π one can use action-value function Q^π defined as:

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[G | s_0 = s, a_0 = a \right] \tag{1}$$

Expectation of Q-value is computed over all possible action and states reward with the policy π started from state **s** and performed action **a** and then following its policy. If the true Q-function (Q^*) is given for us, we can derive optimal policy by taking action a that maximizes Q for each state s: $a = \text{argmax}'_a Q(s,a')$

To learn Q for the optimal policy we use *Bellman equation* (2):

$$Q^\pi(s,a) = r(s,a) + \gamma \max_{a'} Q^\pi(s',a') \tag{2}$$

In [17], authors proved that sequential assignment converges from any Q to optimal Q^* if there are only finite number of actions and states, and all possible combinations are repeated in a sequence. The problem lies in the following procedure: if we start learning from some initialization of Q-function, we will learn nothing due to max operator, leaving no space for exploration. To overcome this issue, sampling actions with probabilities $p(a_i) = \frac{\exp Q(s,a_i)}{\sum_j \exp Q(s,a_j)} = softmax(a_i)$ (Boltzmann approach) or *epsilon-greedy* sampling were proposed.

In what follows, we consider the problem of approximating Q-function with deep neural networks when we have infinite number of states.

Deep Q-Network In [10], authors presented an agent achieving super-human performance in Atari games, starting the era of Deep RL achievements. Authors use two-layer Convolutional Neural Network (CNN) as feature extractor and add two fully-connected layers to estimate Q-function. The input of this network is screenshot from the game, and the output is Q-function, one per action. The proposed model can not directly apply the rule (2) and instead compute *Temporal Difference* error (TD):

$$TD = Q(s_i,a_i) - (r_i + \gamma \max_{a'} Q(s'_i,a')) \tag{3}$$

and then minimize square of it. Authors used *online network* to estimate $Q(s_i,a_i)$ term and *target network* to estimate $\max_{a'} Q(s'_i,a')$ term. Online network was trained via backpropagation, while value produced by the target network were fixed and updated in periodic manner. The training for online network parameters θ and target network parameters $\tilde{\theta}$ was made via the following loss:

$$L = \sum_i \left(Q(s_i,a_i;\theta) - (r_i + \gamma \max_{a'} Q(s'_i,a';\tilde{\theta}) \right)^2, \tag{4}$$

where sum is computed over a batch of samples, via which the network propagate the error. To form a batch authors in [10] propose to use *experience replay*, which contains tuples $\langle s,a,r,s' \rangle$ of state, performed action, reward and next state. The

main idea is to estimate $Q(s, a)$ only for the performed actions, rather than to all possible actions.

Although DQN showed human-like (or even better) performance in Atari playing, it has certain drawbacks on games requiring complicated strategies, and also show slow and unstable training [16], while also overestimating of expected reward [15]. Below, we consider extensions of DQN, which can stabilize and improve training process overcoming issues mentioned above.

Double Q-learning Conventional Q-learning suffers from *max* operator in (2) and agent always overestimates obtained reward. There is simple solution called *Double Q-learning* [15] that is to replace $max_{a'}Q(s, a)$ with

$$Q(s', \operatorname*{argmax}_{a'} Q(s'a); \theta) \tag{5}$$

leading to faster and more stable training process.

Dueling Networks The *Dueling network* [16] decompose Q-function: $Q(s, a) = V(s) + A(s, a)$, where $V(s)$ is *value* and $A(s, a)$ is advantage. Estimating each of them in its own stream significantly boost training performance; Q-function is computed by following rule:

$$Q(s, a) = V(s) + A(s, a) - \frac{1}{N_a} \sum_{j}^{N_a} A(s, a_j) \tag{6}$$

3.2 Recurrent Q-learning

Q-learning is trained under fully observable environment state for each step, which is usually not the case for practical applications. In many cases, we can *observe* or trust only part of the environment state, leading to *Partially Observable Markov Decision Process* (POMDP) setting. POMDP is a tuple (S, A, R, T, Ω, O), where first four items comes from MDP, Ω indicates *observation* set and O indicates *observation function*: $O(s_{t+1}, a_t) = p(o_{t+1}|s_{t+1}, a_t)$, $\forall o \in \Omega$. An agent makes a decision by making actions in the environment and receiving new observations; its belief distribution in optimal next state is based on distribution of the current state.

Deep Recurrent Q-Network Without complete information on state s_t in POMDP, it is impossible to use DQN-like approach to estimate $Q(s_t, a_t)$. However, there is a simple solution to adapt previous solutions to POMDP: an agent is assigned with a memory h_t and approximates $Q(s_t, a_t)$ by using observations and memory from previous step $Q(o_t,\ h_{t-1}, a_t)$. One of suitable solutions is recurrent neural network eliminating observation o_t by Long-Short Term Memory block [6] $h_t = LSTM(o_t, h_{t-1})$. Such networks are called Deep Recurrent Q-Network (DRQN) [4].

Experience replay in POMDP setting contains tuples $\langle o, a, r, o' \rangle$ denoting observation, performed action, reward and next observation. It is necessary to sample consecutive observations from experience replay in order to use memory information, which is usually based on last observations.

Multi-step learning DQN is trained on a single time step, while DRQN should be trained on a sequence, which immediately get us to idea how to extend temporal difference (3) in loss function to n-step temporal difference [13]:

$$TD(n) = Q(s_i, a_i) - \\ - (r_i + \gamma r_{i+1} + \ldots + \gamma^{n-1} r_{i+n-1} + \gamma^n \max_a Q(s_{i+n}, a')) \quad (7)$$

This method also trains faster, especially for games with sparse and delayed rewards, but hyper-parameter n of steps number should be tuned [14].

Distributional RL Q-function as an expectation of discounted reward may be estimated in statistical way by learning distribution of discounted reward [2] with probability masses adjusted with discrete support z with N atoms, $z_i = V_{min} + i * \Delta z, i = 0, \ldots, N$, where $\Delta z = (V_{max} - V_{min})/(N-1)$ and (V_{min}, V_{max}) represent admissible value interval. Authors denote the value distribution as Z, then distributional version of (2) holds: $Z(x, a) \stackrel{D}{=} R(x, a) + \gamma * Z(X', A')$. Authors proposed loss function and exact algorithm called Categorical 51 (C51), where 51 is the number of atoms z_i. An agent trained using C51 is more expressive and converges to a better policy.

4 Scenarios

In this section we describe scenarios, on which we have tested our agents. For each scenario we provide general information, such as reward design and agent's goal and ways for agent to reach the goal. We also provide screen resolution and frameskip settings and show a screenshot for each scenario.

4.1 Basic

Player spawns in a wide room with its back to one of the long walls. On the opposite side monster spawns with 1 health point, in a random position along the wall (see Figure 1). Only 3 buttons (and $2^3 = 8$ actions) are available for player: attack, move left, move right. The goal is to navigate and hit a monster. Agent receives 100 reward for hitting, -5 reward for missing and -1 for living, this forces player to kill the monster as fast as possible. Basic scenario is the simplest in VizDoom environment, and mainly used to test correctness of algorithms, so even agent with big frameskip can converge to a good policy.

We decide to use frameskip 10 and simplified version of neural network as well as smaller amount of training epochs and steps per epoch for this scenario. We also resize game screenshot from base 640×480 resolution to 45×30 and convert it to grayscale.

Fig. 1: A screenshot of Basic scenario

4.2 Defend the center

In the second scenario, an agent with pistol and limited amount of ammo respawns in the center of a circular room and is able only to turn left or right and fire (see Figure 2). Also, there are five melee enemies that spawn at random location against the wall and make their way towards the agent in the center. When agent kills one of them, another spawns at random location. Agent's goal is to kill as many enemies as possible, it receives +1 reward for killing enemy and -1 for death. For the agent death is inevitable, because it has a limited number of ammo. Although reward shaping may cause positive effect on agent performance we decide not to use it in this scenario.

This scenario is a good example of POMDP: agent can observe only 110 degrees and can only guess about where enemies are located. For the agent, it is important to make accurate aiming, so we change screen resolution 400×225 ($16 : 9$ scale), resize down to 108×60 and reduce frameskip to 4.

Fig. 2: A screenshot of Defend The Center scenario

4.3 Health gathering

In the third scenario, agent spawns in square room with dangerous substance on the floor making constant damage each time step. Agent has no access to its health supporting POMDP setting. Among possible actions are turn left, turn right and move forward. There are health kits placed on the floor, which collected increase agent's health helping to survive (see Figure 3). The following reward shaping was used [1]: agent receives +1 for every step, -100 if it dies, and

+ *amount of healing*, which is equal to $health(o_t) - health(o_{t-1})$ if this value is bigger than zero, where $health(o)$ represents the agent health in the game. An episode ends if the agent dies or if 2100 tics done. Agent trained with frameskip 4 and screen resolution of 108×60.

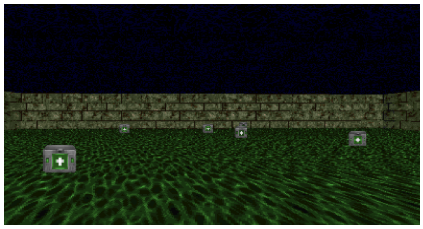


Fig. 3: A screenshot of Health Gathering scenario

5 Proposed Approach

We use two agents as a baseline: DQN and DRQN with LSTM unit. Next, we consider two modifications. The first is DRQN with Dueling Network, Double Q-Learning, and dropout with rate=0.5 (D4RQN). The second is combination of C51 with Multi-step (C51M).

Since Basic scenario is very simple for agent to learn, we decide to use the same feature extractor for every scenario with its architecture presented in Table 1.

Input size	Basic	Input size	Other
$30 \times 45 \times 1$	CR, n=8, k=6, s=3	$60 \times 108 \times 3$	CR, n=32, k=8, s=4
$9 \times 14 \times 8$	CR, n=8, k=3, s=2	$14 \times 26 \times 32$	CR, n=64, k=4, s=2
$4 \times 6 \times 8$		$6 \times 12 \times 64$	CR, n=64, k=3, s=1
		$4 \times 10 \times 64$	

Table 1: Convolutional feature extractor for different scenarios. CR denotes Convolution + Relu, n denotes number of convolution filters, k denotes kernel size, s denotes stride.

After convolutional layers, the feature map is reshaped to vector and is feeded into dense layers. DQN network has dense layer with 128 units in Basic scenario and 512 in other, both, with Relu activation. Both DQN and DRQN has one more dense layer at the end with *number of actions* units and linear activation.

D4RQN and C51M has dropout layer after convolutions with rate = 0.5 followed by LSTM layer with 128 units for Basic and 512 units for other scenarios. D4RQN splits computation into two streams: *value* stream and *advantage* stream by dense layers with 1 and *number of actions* units with linear activation.

Outputs from streams are combined by formula (6) and targets during optimization are picked according to (5) thus combining Double Q-learning and Dueling Networks.

There are atoms in C51M algorithm supporting discounted reward distribution. Each atom has probability, calculated as softmax over atoms. We split LSTM output into two streams: *value* stream with *number of atoms* units and *value* stream with *number of actions* linear layers, each with *number of atoms* units. For each atom, these streams are combined by formula (6) and then softmax function applied. To compute Q-value from this distribution we use formula: $Q(s,a) = \sum_i^{n\_atoms} z_i p_i(s,a;\theta)$, where z_i and p_i is the i-th atom support and probability and θ represents network parameters. An illustration of the network architecture for C51M algorithm is presented in Figure 4.

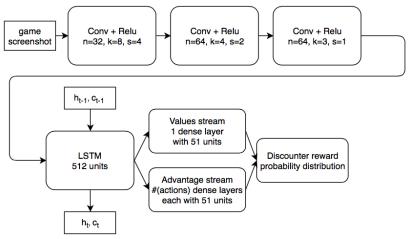


Fig. 4: Neural network for C51M architecture

We choose 21 atoms for C51M for Basic (so it could be called C21M instead) and 51 atoms as in the original algorithm for other scenarios. Next, we set the values for V_{min} and V_{max} in atom support. For Basic scenario, we set these values to -5 and +15, because it is relatively simple and reward can take only 3 values: -15, -10 and 95. So, it is normal if we clip maximum reward to 15 to balance it with negative reward. For Defend The Center, we set $V_{min} = -5$ and $V_{max} = +15$. In Health Gathering scenario [1], we set these values to -5 and 195. The maximum possible reward equals to maximum episode length which is 2100 and can not be precisely calculated for shaped reward.

6 Experiment design

For Basic scenario, we organize training as follows: first agent makes one step in environment and then samples observation history from experience replay. We set batch size to 32 and sequence length to three, with first frame used for hidden state estimation and last two frames to train on. Each agent trains 21 iterations of 2000 gradient descent steps, we call such iteration *epoch*. Before epoch starts we update target networks' weights. We define epsilon to 1.0 at the start of the training and linearly decay it at each epoch end down to 0.01 in the final epoch.

After each epoch we test our agent setting epsilon to 0.01 and plot mean episode reward over 100 episodes.

We set experience replay size to 10^4 for Basic scenario and to 10^5 for other. For sampling sequences of consecutive observations from experience replay we choose only those containing just the last one as a terminal. It was important to reduce number of terminal states for Health Gathering scenario, because agent receives large negative reward if it dies on the last observation. For all the other scenarios we used different parameters: we increase batch size and sequence length to 128 and 10, respectively. We use first four observations to update agent's memory and last six to train on. We increase number of gradient steps per epoch to 8000 and number of steps before sampling from experience replay to 15. We also reduce learning rate to 0.0002.

7 Results

For each scenario, we measure TD loss at each gradient descent step (stability), measuring per-episode reward changes during training (learning speed) and obtained reward after each training epoch (performance of agents). For each of Figures 5,6 we visualize C51M (light-blue), DQN (dark-blue), DRQN (orange) and D4RQN (red) agent training results. For Figure 7 the visualization slightly changes due to shift of colors made by mistake: C51M (dark-blue), DQN (red), DRQN (light-blue) and D4RQN (orange). For TD loss plots we visualize C51M separately, so that we could see the difference between this model and other agent architectures.

7.1 Basic

TD loss of each agent, except C51M, is shown on Figure 5a. All values smoothed by exponential moving average with parameter $\alpha = 0.99$. Normal behavior of TD loss includes slow increasing or decreasing. TD loss may jump down when target network updated. Closer to end of training, loss function should decrease, which signals that agent learned something meaningful. From Figure 5a we can see that DRQN and D4RQN trains stable and converges, but DQN has not converged. So, even without looking on reward plots we may assume now that DQN has poor performance.

TD loss for C51M is presented in Figure 5b. This loss is cross-entropy between agent's prediction of distribution and actual one. Loss drops down after first epoch and target network's weight updated. After this loss starts fluctuate and slowly increase. We can observe stable training process on this plot.

Rewards obtained by agents during training can be seen at Figure 5c. We expect these rewards slowly but surely increase during training. It also can be seen that each agent has been playing different number of episodes, because in our experiments we only set up number of epochs and training steps during one epoch. It can be seen that each agent, except DQN, has similar reward growing

(a) TD loss for all agents except C51M

(b) TD loss for C51M agent

(c) Rewards per training episodes

(d) Rewards per testing episodes

Fig. 5: Basic Scenario comparison

rate, and the best one in terms of growing speed is DRQN. It is also good indicator of learning process for Basic scenario.

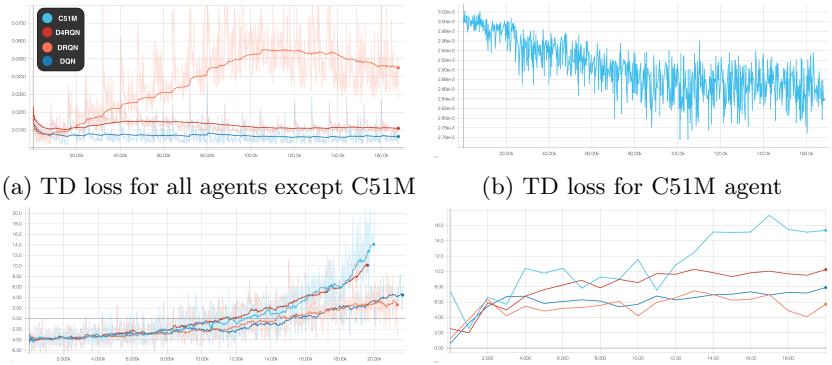
We test all agents at the end of each epoch. In test setting, we turn dropout off and set ϵ to its minimal value 0.01. We play 100 episodes with each agent and calculate mean non-shaped reward per episode. Results can be seen in Figure 5d.

From these figures, it is clear that C51M has much faster learning speed and stability as well compared to other agents. But the best policy goes to DRQN agent. The reason for this may be simplicity of scenario and dropout in combined agents. D4RQN converged to approximately same policy as C51M, but trained much slower even than DQN and is the slowest agent among these all. It converged to pretty bad policy and not capable to play this scenario well given this number of epochs.

7.2 Defend The Center

For Defend The Center scenario TD loss presented in Figures 6a and 6b. We can see similar behaviour for DRQN and D4RQN agents: they both has peaks when target network's weight updated, small variance and stable mean of TD loss. But for DQN agent loss has high variance and behaves unstable. It rapidly increases in the first half of training and starts slowly decreasing in the second half. We can assume that DQN performance is not as good as in other models. Loss for C51M has huge variance, but is stable in decreasing. We may assume that model has not converged yet but it should have good performance when trained longer.

Rewards obtained by the agents during training may be seen in Figure 6c. We can see that DQN and DRQN models trained similarly, but DQN ended up with slightly lower score. D4RQN and C51M trained much faster and converged to significantly better policies, and C51M did it faster and scored more. Also it



(a) TD loss for all agents except C51M

(b) TD loss for C51M agent

(c) Rewards per training episodes

(d) Rewards per testing episodes

Fig. 6: Defend The Center Scenario comparison

is noticeable that both agents did not stop their score improvement at the end of training and potentially may be trained further.

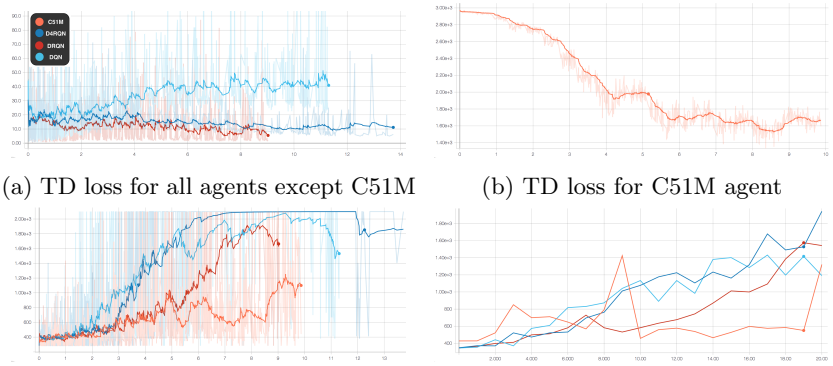
Rewards obtained during testing after each epoch are showed in Figure 6d. These plots confirm our hypothesis about agent performance in previous paragraph. Reward on this plot is mean value of total reward without shaping from 100 episodes. For each episode reward is equal to (number of kills - number of deaths), which is obviously just (number of kills - 1). So, mean number of kills can be inferred by adding one to each plot point.

C51M came off ahead D4RQN by 5 kills in average and scores roughly 16 kills. D4RQN obtains 11 kills, which is greater by 2 than DRQN and by 4 than DQN at the end of training. So, our worst models in average scores 6 kills, whereas best model in [11] scores 5, which seems quite strange and may be explained by converging of their model to bad policy or alternate image descriptors for convolutional layers representations.

We showed that in Defend The Center episode C51M was far superior. It trains faster and converges to 50% better policy in reward terms. We noticed that agents (except C51M) are not hesitating to shoot (and wasting ammo) rapidly. DQN agent learns just to turn one side and fire simultaneously, DRQN has better aiming skill but its behaviour pretty same. D4RQN aims accurately but misses a lot anyway.

Only C51M learns that saving ammunition is vital to survival and missing is bad. It starts the game by turning around, looking for enemies and shooting them down, but after some time it let enemies approach closer, because it is easier to aim and shoot at near enemy rather then at enemy far away. When ammo amount drops to small values, around 5, agent starts to wait being hit. After agent being hit screen, flash red and agent starts to search the attacker, and when it founds him then it eliminates attacker with one shoot.

This behavior is remarkable because it is similar to human: player usually start playing aggressively, but when ammo count became low, they start think



(a) TD loss for all agents except C51M

(b) TD loss for C51M agent

(c) Rewards per training episodes

(d) Rewards per testing episodes

Fig. 7: Health Gathering Scenario comparison

about safety and tries to survive as long as possible. Another reason for our admiration is that we did not use reward shaping and only signal that agent receives is +1 for kill and -1 for being killed. No additional information about ammo or health is being used. It is clearly a challenging task for RL agent to learn such behaviour without additional information.

7.3 Health Gathering

The results for Health Gathering are used from our paper [1].

TD loss for all the agents can be seen at Figures 7a and 7b. Again, all agents, except DQN, show stable learning process. Rewards obtained during training can be observed at Figure 7c (plot is constructed in relative time-scale). D4RQN obtains max score more than in half of training episodes, but other agents do not show stable performance. Test rewards presented at figure 7d. C51M has lower performance while D4RQN has highest. Unexpectedly, D4RQN obtained much higher reward while training than while testing with turned off dropout. We further study agents' performance with and without dropout at testing time. DQN does not detect health pack as well as D4RQN and may just move into a wall and die. DRQN plays pretty well, but agent's behaviour was not interpretable in terms of any reasonable humans' behaviour. C51M is good at detecting health packs, but it scores lower than D4RQN and DRQN have, because it tries to wait several frames before pick up a health pack.

7.4 Summary results

Since Health Gathering scenario forced to end after 2100 steps during training, we modified the test for this scenario set episode length to 10000 and call it Health Gathering Expanded. We add this version of scenario in Table 2 following [1] paper previously written by us.

For Defend The Center scenario reward is equal to number of kills minus one as a death penalty. In Table 2 we report pure number of kills obtained by each agent.

Model	B	DTC	HG	HG Expanded
DQN	79.2 ± 9.8	6.91 ± 1.9	1262.0 ± 631.0	1469.6 ± 1257.0
DRQN	13.5 ± 73.5	8.53 ± 1.2	1578.1 ± 665.9	3173.5 ± 2842.3
D4RQN	59.7 ± 37.5	11.15 ± 1.32	1890.0 ± 434.3	4781.7 ± 2938.8
D4RQNd	53.9 ± 46.3	11.3 ± 1.17	2078.8 ± 144.5	9291.4 ± 2231.2
C51M	64.6 ± 24.3	16.72 ± 2.0	1451.7 ± 708.9	2466.4 ± 1766.5
C51Md	56.2 ± 43.1	16.2 ± 1.68	1593.7 ± 702.3	4138.4 ± 3634.6

Table 2: Final performance for each scenario of all trained agents in order: DQN, DRQN, D4RQN dropout off, D4RQN dropout on, C51M dropout off, C51M dropout on. Values in Table equals to $mean(R) \pm std(R)$, where R is non-shaped rewards over 100 episodes.

From Table 2 we can observe that dropout at *testing time* may increase agent's performance (Health Gathering scenario for both D4RQN and C51M), or decrease (Defend The Center for C51M, Basic for both D4RQN and C51M) or cause no effect (Defend The Center, D4RQN).

8 Conclusion

In this work, we extend results of [1] on two additional scenarios while studying new model-free deep reinforcement learning agents in POMDP settings for 3D first-person shooter. The presented agents drastically outperformed baseline methods, such as DQN and DRQN. Our agent successfully learned how to play several scenarios in VizDoom environment and show human-like behaviour. This agent can be used like backbone architecture for more challenging task, like Deathmatch scenario, which is exactly our plan for future work. Moreover, our agent could be easily combined with Action-specific DRQN [18], Boltzmann exploration [13], Prioritized Experience Replay [11], and can be modified to use in-game features as well as separate networks for action and navigation to improve further.

Acknowledgements

This work extends the results of our previous study presented on MMEDIA international conference in 2019 [1].

References

1. Akimov, D., Makarov, I.: Deep reinforcement learning in vizdoom first-person shooter for health gathering scenario. In: MMEDIA. pp. 1–6 (2019)

2. Bellemare, M.G., Dabney, W., Munos, R.: A distributional perspective on reinforcement learning. arXiv:1707.06887 (2017)
3. Bellemare, M.G., Naddaf, Y., Veness, J., Bowling, M.: The arcade learning environment: An evaluation platform for general agents. In: Proceedings of the 24th International Conference on Artificial Intelligence. pp. 4148–4152. IJCAI'15, AAAI Press (2015), http://dl.acm.org/citation.cfm?id=2832747.2832830
4. Hausknecht, M., Stone, P.: Deep recurrent q-learning for partially observable mdps. CoRR, abs/1507.06527 (2015)
5. Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., Silver, D.: Rainbow: Combining improvements in deep reinforcement learning. arXiv preprint arXiv:1710.02298 (2017)
6. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9**(8), 1735–1780 (1997). https://doi.org/10.1162/neco.1997.9.8.1735, https://doi.org/10.1162/neco.1997.9.8.1735
7. Kempka, M., Wydmuch, M., Runc, G., Toczek, J., Jaśkowski, W.: Vizdoom: A doom-based ai research platform for visual reinforcement learning. In: CIG'16. pp. 1–8. IEEE (2016)
8. Lample, G., Chaplot, D.S.: Playing fps games with deep reinforcement learning. In: AAAI. pp. 2140–2146 (2017)
9. Makarov, I., Kashin, A., Korinevskaya, A.: Learning to play pong video game via deep reinforcement learning. CEUR WP pp. 1–6 (2017)
10. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
11. Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized experience replay. arXiv preprint arXiv:1511.05952 (2015)
12. Schulze, C., Schulze, M.: Vizdoom: Drqn with prioritized experience replay, double-q learning, & snapshot ensembling. arXiv preprint arXiv:1801.01000 (2018)
13. Sutton, R.S.: Learning to predict by the methods of temporal differences. Machine learning **3**(1), 9–44 (1988)
14. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction, vol. 1. MIT press Cambridge (1998)
15. Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: AAAI. vol. 16, pp. 2094–2100 (2016)
16. Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., De Freitas, N.: Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581 (2015)
17. Watkins, C.J.C.H., Dayan, P.: Q-learning. Machine Learning **8**(3), 279–292 (May 1992)
18. Zhu, P., Li, X., Poupart, P., Miao, G.: On improving deep reinforcement learning for pomdps. arXiv preprint arXiv:1804.06309 (2018)