Combo-Action: Training Agent for FPS Game with Auxiliary Tasks

Shiyu Huang, Hang Su, Jun Zhu,* Ting Chen*
Dept. of Comp. Sci. & Tech., BNRist Center, State Key Lab for Intell. Tech. & Sys.,
Institute for AI, THBI Lab, Tsinghua University, Beijing, 100084, China
hsy17@mails.tsinghua.edu.cn; {suhangss, dcszj, tingchen}@tsinghua.edu.cn

Abstract

Deep reinforcement learning (DRL) has achieved surpassing human performance on Atari games, using raw pixels and rewards to learn everything. However, first-person-shooter (FPS) games in 3D environments contain higher levels of human concepts (enemy, weapon, spatial structure, etc.) and a large action space. In this paper, we explore a novel method which can plan on temporally-extended action sequences, which we refer as Combo-Action to compress the action space. We further train a deep recurrent Q-learning network model as a high-level controller, called supervisory network, to manage the Combo-Actions. Our method can be boosted with auxiliary tasks (enemy detection and depth prediction), which enable the agent to extract high-level concepts in the FPS games. Extensive experiments show that our method is efficient in training process and outperforms previous state-of-the-art approaches by a large margin. Ablation study experiments also indicate that our method can boost the performance of the FPS agent in a reasonable way.

Introduction

Deep reinforcement learning (DRL) has shown great success in many games, including the computer Go game (Silver et al. 2016), Atari games (Mnih et al. 2013), etc. Besides the 2D games (e.g., Go and Atari), applying DRL to first-person-shooter (FPS) games in an adversarial 3D environment (Kempka et al. 2016; Lample and Chaplot 2016) has attracted attention, in which a player fights against other computer agents or human players. Compared with the 2D games, FPS games show a multitude of challenges since the additional spatial dimension not only introduces notions of partial observability and occlusions, but also causes complications due to viewpoint variance and more unpredictable actions of the enemies. Moreover, this task also involves a wide variety of actions and skills, such as navigating through a map, collecting items, fighting enemies, etc. ViZDoom (Kempka et al. 2016) is a RL research platform which allows researchers to develop agents to play the Doom game with the screen buffer and game variables. Many efforts have been paid on developing AI bots to learn a strategy via data-driven methods since the release of ViZDoom, e.g., Arnold (Lample and Chaplot 2016), F1 (Wu and Tian 2017) and IntelAct (Dosovitskiy and Koltun 2016).

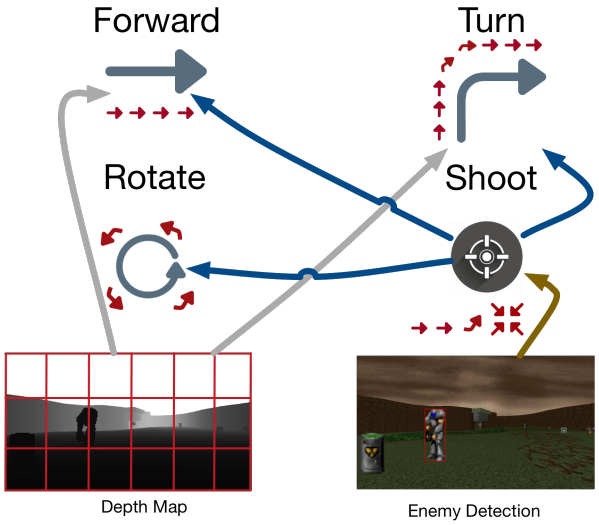


Figure 1: An illustration of the Combo-Action in ViZDoom. We define four Combo-Actions (Forward, Turn, Rotate, Shoot) in this paper. Each Combo-Action is a temporal abstraction of the primitive actions. A depth prediction model and an enemy detection model are used for corresponding Combo-Actions. Instead of using the whole depth map, we divide it into a 3×6 grid and calculate the average depth value for each cell. We further train a DRQN model, called supervisory network, as a high level controller for the Combo-Actions.

Challenges

Even much work has been done recently, there still remain many problems in building agents for the FPS game.

Large Action Space: In general, there are numerous primitive actions in FPS games for an agent to interact with the environment, which can be categorized into on-off actions and delta actions. On-off actions only contain bi-

*J. Zhu and T. Chen are corresponding authors.

nary states, while delta buttons have continuous values. Moreover, the combinations of actions increase exponentially with time steps. This makes the action space very large, and results in the inefficient training process. Previous work (Lample and Chaplot 2016; Wu and Tian 2017; Dosovitskiy and Koltun 2016) selectively abandons many actions to compress the action space, which results in performance loss, e.g., their agents can't change visual angle vertically, making the agent unable to shoot the enemy on a different horizontal plane.

Lack of Prior Knowledge: Humans can learn throughout their lives and can utilize prior knowledge to complete new tasks quickly. However, reinforcement learning algorithms often learn a new task from scratch, which makes them requiring far more experience than humans during training. Although large amounts of research seeks to improve the sample efficiency of reinforcement learning algorithms, there are few studies in incorporating prior knowledge into reinforcement learning. In FPS games, for example, it is vital to recognize some basic concepts(enemy, weapon, spatial structure, etc.). But it is hard to extract such information in a single end-to-end RL model.

Disharmonious Actions: Disharmonious actions often occur in previous trained agents, i.e., actions are not meaningful between step to step. For example, sometimes the agent will turn left and right repeatedly and remain where it is. Previous work (Wu and Tian 2017) only tried to relieve this problem by manually detecting this situation in test period and could do nothing for the RL model.

Our Proposal

To address the aforementioned issues, we develop a novel method that can plan on temporally-extended action sequences, which we refer as Combo-Action. We trained a deep recurrent Q-learning network(DRQN) as a supervisory network to manage the Combo-Action. Our method enables the reinforcement learning algorithm to be boosted with auxiliary tasks and prior knowledge.

Combo-Action: We propose a kind of micro-action, called Combo-Action, in this paper. The Combo-Action is built on a series of primitive actions, which can complete a specific sub-task. These action combinations are adopted to RL training, which compresses the action space sharply and allows us to obtain the optimal value function within a practical time and memory limitation. This method also guides the agent for a better exploration during training.

Auxiliary Tasks: Previous methods prefer to use an end-to-end neural network to play the FPS game. However, a single model is hard to handle a complex task. Our method develops extra two sub-tasks simultaneously, i.e., enemy detection task and depth prediction task. This decoupling makes the debugging process to be more intuitionistic. Moreover, the auxiliary networks extract high-level concepts from the observation, which provides useful information to the executing of Combo-Action.

Our method can alleviate disharmonious-action problem by defining reasonable Combo-Actions. The priori knowledge in Combo-Actions can emit more reasonable primitive actions. Interestingly, experiment shows that even the random choosing of Combo-Actions can yield not-bad performance.

Supervisory Network: To manage the switch between different Combo-Actions, a high-level controller should be applied. In this paper, we use an LSTM (Hochreiter and Schmidhuber 1997) based recurrent neural network for the Q-learning model. Our supervisory network can work harmonically with other auxiliary networks during test period.

Contributions: The contributions of our work are as follows: (1) Our method can compress the original action space sharply, which improves the training efficiency and exploration ability. (2) Our method can fuse priori knowledge and basic concepts into the RL, which reduces the training difficulty and boosts the performance of the trained agent. (3) Our method can alleviate disharmonious-action problem by defining reasonable Combo-Actions for the FPS game.

Background

In this section, we briefly review the deep Q-learning and deep recurrent Q-learning network. We also present some work related to our method and the efforts made in the FPS game AI research field.

Deep Q-learning

Deep Q-learning can learn a policy by interacting with the environment. At each step, the agent obtains current state s_t of the environment, gives out an action according to its policy, and receives a reward r_t. The goal of the Q-learning algorithm is to maximize the expected sum of discounted rewards $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, where T is the terminating time, and $\gamma \in [0, 1]$ is a discount factor. The action value function, called Q-function, takes two inputs: state s and action a, and returns the expected future reward: $Q^{\pi}(s, a) = \mathbb{E}[R_t | s_t = s, a_t = a]$. In Deep Q-learning(Sutton, Barto, and others 1998), a neural network parameterized by θ is used as an estimate of the optimal Q-function. To optimize the Q-function, the temporal difference error is taken as the loss function:

$$L(\theta) = \mathbb{E}_{s,a,r,s'}[(Q_{target} - Q_{\theta}(s, a))^2], \tag{1}$$

where $Q_{target} = r + \max_{a'} Q_{\theta}(s', a')$.

Deep Recurrent Q-learning Network(DRQN): Typically, the task for reinforcement learning should be Markovian. However, the observation(e.g. partial field of vision in 3D FPS game) for the agent is not Markovian, and this is considered as a partially observable Markov decision process(POMDP). To allay this problem, a memory module is often required, which can be used to store the history information. (Hausknecht and Stone 2015) introduced the Deep Recurrent Q-Networks(DRQN). DRQN applies a recurrent neural network into DRL, and LSTM (Hochreiter and Schmidhuber 1997) is often used on the top of the normal DQN model. In our project, we use DQRN as our basic reinforcement learning model. We present some related work in following sub-sections.

Reinforcement Learning with Temporal Abstractions

Temporally extended actions have proven very useful in speeding up learning process, ensuring robustness and fusing prior knowledge into AI systems (Sutton, Precup, and Singh 1999; Precup 2000; He, Brunskill, and Roy 2010; Tessler et al. 2017). (Precup 2000) proposed the *options* framework, which involves abstractions over the space of actions and extends traditional MDP setting to a semi-Markov decision process (SMDP). (He, Brunskill, and Roy 2010) defined the Macro-Actions to partially observable Markov decision process(POMDP). (Bacon, Harb, and Precup 2017) proposed a method that can learn *options* autonomously from data. (Arulkumaran et al. 2016) trained a supervisory network to manager the "option heads" on the policy network. (Frans et al. 2017) used the "mete-learning" concept to construct an end-t-end hierarchical RL algorithm. Our Combo-Action is inspired by these ideas of temporal abstractions and we further incorporate the supervised signals into the building of Combo-Actions.

Reinforcement Learning with Auxiliary Tasks

Although reinforcement learning algorithms are trained with reward signals from the environment, it's interesting to study how to use the supervised signals to help the training process. (Mirowski et al. 2016) used two auxiliary tasks, i.e., depth prediction and loop closure classification to help the navigation task. They illustrated that the performance was dramatically improved via these additional auxiliary tasks. (Bhatti et al. 2016) used SLAM and Faster-RCNN (Ren et al. 2015) to boost the inputs of the observation for reinforcement learning algorithm. (Lample and Chaplot 2016) augmented the deep Q-learning model via training RL and object prediction simultaneously. Instead of using auxiliary tasks for inputs or outputs of the RL algorithms, our auxiliary tasks(detection and depth prediction) cooperatively work with Combo-Action.

Reinforcement Learning for FPS Game

Early attempts of building FPS AI players focused on the manually-designed rule-based approaches (van Waveren 2001), which is not robust and time-consuming to tune the rules in many complicated situations. Recently, researchers have deployed deep reinforcement learning into 3D first-person shooter (FPS) games, e.g., the Doom game (Kempka et al. 2016). Arnold (Lample and Chaplot 2016) used game frames and trained an action network using Deep Recurrent Q-learning and a navigation network with DQN, it outperformed the built-in AI of the Doom game. IntelAct (Dosovitskiy and Koltun 2016) modeled the Doom AI bot training in a supervised manner by predicting the future values of game variables (e.g., health, amount of ammo, etc) and acting accordingly. F1 (Wu and Tian 2017) combined the Asynchronous Advantage Actor-Critic (A3C) model with curriculum learning to train the bot step by step. However, most of these works implement the algorithm with primitive actions, without the ability to extract a variety of semantic concepts and abstractions (enemy position, environment space, etc.). It makes the decision space large and sparse, yielding the learning process with low efficiency. Our proposed method can extract meaningful concepts with auxiliary networks from the environment and yield more powerful performance.

Methodology

In this section, we first introduce our method in a general form, and then illustrate how we design Combo-Actions for the FPS game and how we train each part in this framework.

Framework Overview

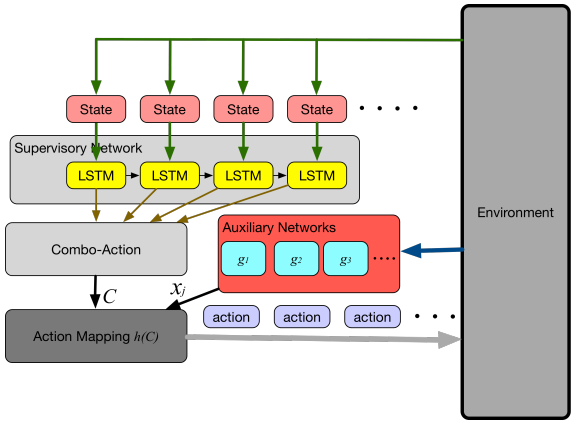


Figure 2: The overall architecture of our method. The framework contains three main parts, i.e., a supervisory network, auxiliary networks and the action mapping function. The supervisory network is a high-level controller for the Combo-Action, and auxiliary networks deal with the environment information. The action mapping function can map the Combo-Action and auxiliary network's outputs into a series of primitive actions.

Figure 2 shows the architecture of our method. ViZDoom provides an interactive environment for an agent to get information from the environment and post actions to control its behavior. Our goal is to improve the performance of the agent with auxiliary tasks.

Let the Combo-Action space be $\mathcal{C}$ and the original action space be $\mathcal{A}$. Let the output of supervisory network be $C_t = f_\theta(s_t)$, where $C_t \in \mathcal{C}$ is a Combo-Action that the supervisory network $f_\theta(\cdot)$ chooses under state s_t at time step t, and θ is the parameters of the network. The θ is learned from scratch using reinforcement learning algorithm. There is a mapping function which can map the Combo-Action C to a series of primitive actions. We define the mapping as $\{a_i\}_t = h(C_t; g_1(s_t), g_2(s_t), \cdots), i = 1, 2, 3...$, where $g_j(\cdot)$ is an auxiliary network. $g_j(\cdot)$ is a supervised trained neural network which can provide useful information for $h(\cdot)$. The output $g_j(\cdot)$ can be formalized as a vector x_j, and x_j can be considered as the parameters of the mapping function $h(\cdot)$. The posted actions $\{a_i\}$ will be temporally scheduled by $h(\cdot)$ and sent to the environment for more sophisticated control.

Combo-Actions Design for FPS Game

In this section, we introduce the Combo-Actions for ViZDoom . A Combo-Action $C \in \mathcal{C}$ is a kind of macro actions built on a set of primitive actions and it is the output of the supervisory network. We first define three Combo-Actions: **Forward(F), Turn(T) and Rotate(R)**. We also define a **Shoot(S)** Combo-Action, which is applied to every time step. We also design two auxiliary networks: detection network and depth prediction network. ViZDoom provides APIs to generate the ground truths for both tasks, which enables training networks with supervised manner. Figure 1 shows the example of Combo-Actions in ViZDoom.

Detection The recognition of enemies is quite important for FPS games. We design a convolutional neural network to detect the enemy in the game. The network takes RGB images as input and outputs predicted bounding boxes of enemies. Our detection algorithm, named as RPNmini enables end-to-end training and satisfies the real-time requirement while maintaining high average precision.

Depth Prediction There are kinds of maps in the ViZDoom and the textures can be various in a scene. The structures or textures make little sense for the movement of the game player. The depth map can offer enough spatial information for the navigation. We design a small convolutional neural network to predict the depth of current visual input. This network takes RGB images as input and outputs predicted depth map. To simplify this task, the depth map is separated into 18 parts equably with 3 rows and 6 columns, and Figure 1 shows the example of the partition.

Forward This Combo-Action means the agent need keep moving forward. The depth values at middle two columns are used to calculate the number of steps the agent should execute for the 'move forward' action. The bigger the depth values are, the longer steps will be applied.

Turn This Combo-Action means the agent need turn a certain degree to change its direction. The depth map is used to calculate how many degrees the agent should turn. The agent will always turn to the most commodious area. In the implementation, the agent will execute four 'turn-90-degree' actions to capture the full vision of the environment, and then choose out the direction with maximal depth value.

Rotate Although Forward and Turn are enough for agent's movement, FPS games are partially observed for the agent and sometimes the agent gets injured without finding any enemy in its direction. The Rotate Combo-Action is designed to help the agent to find the enemy out of its current visual field. This Combo-Action lets the agent turn 360 degree to scout the environment with executing four fixed 'turn-90-degree' actions. Once the agent finds enemies, it will switch to Shoot Combo-Action. Experiment shows that this Combo-Action can improve the performance of the agent to a certain extent.

Shoot Shoot is the most important Combo-Action in FPS game playing, because it directly decides how many scores the agent can get. The detection model is used for this Combo-Action. Once there is an enemy, other Combo-Actions will be stopped, and the agent will target to the enemy based on the bounding box and it will fire the gun when the cross-hair within in the bounding box. When there is no enemy detected, this Combo-Action will do nothing.

Aiming fast-moving enemy ahead is a common skill in FPS game playing. To add this skill to the Shoot Combo-Action, we record the action history of the agent, and when we detect out that it turns to one direction repeatedly, we double the turning degree for ahead aiming. Experiment shows that aiming ahead can provide a performance boost.

Supervisory Network A supervisory network is trained to manage the Combo-Actions. There are three Combo-Actions which can be chosen by the supervisory network, i.e., Forward, Turn and Rotate, and the Shoot Combo-Action is applied to the every step. We use a recurrent neural network to construct the deep Q-learning algorithm. At each step, the supervisory network takes an image and two game variables(healthy value and ammo number) as inputs and outputs a hidden state and the Q-value for each Combo-Action. The hidden state is then fed to next step and the Combo-Action with maximal Q-value will be executed.

Training Detection Network

Detection Dataset: Training a high performance detection model requires a large dataset. We collect a set of labeled images for training and testing from the ViZDoom environment. We also flip the images for data augmentation. In total, we generate $30,000$ RGB images together with object labels. There is only one class of game object in the dataset– enemy. The resolution of the image is 576×1024 pixels. We then split our dataset into 3 partitions: *Train*: *Validate* :*Test*, with ratios 70%:20%:10%.

RPNmini: Similar to FasterRCNN (Ren et al. 2015), our detection model, named RPNmini, divides the input image into an $M \times N$ grid and assigns k anchors for each cell. During training, we match the default anchors to the ground truths with the best jaccard overlap (Erhan et al. 2014). Each grid cell predicts k bounding boxes (each bounding box has 4 values to indicate its location) and each bounding box predicts $C + 1$ classes, where C is the number of object types, and the extra one class is for background. The predictions for bounding boxes can be encoded as an $M \times N \times k \times 4$ tensor and the predictions for classes can be encoded as an $M \times N \times (C + 1)$ tensor. Our experiments show that RPNmini can have 30 times speed-up during inference without degradation in performance compared with other baselines (Ren et al. 2015; Huang and Ramanan 2017).

Training Loss: Our model is learned from scratch without any pre-trained weights for initialization. The overall objective loss function is a weighted sum of the localization loss and the classification loss:

$$L_{obj} = \frac{1}{N}(L_{class} + \lambda_{loc} L_{loc}), \tag{2}$$

where N is the batch size, L_{loc} is the sum-squared error in bounding box prediction and L_{class} is the softmax loss in classification. We also add a hyper-parameter $\lambda_{loc} = 0.5$ to adjust the localization error.

Layer #	1	2	3	4	5	6	7	8
	C3×3×32s1	MP2×2s2	C3×3×16s1	MP2×2s2	C3×3×8s1	MP2×2s2	FC4608×128	FC128×18

Table 1: The architecture of depth prediction model. C3×3×32s1 = convolutional layer with 3×3 kernel, stride 1 and number of output planes 32. MP2×2s2 = MaxPooling layer with 2×2 kernel, stride 2. FC4608×128 = fully connected layer with input size 4068, output size 128. Each convolutional and fully connected layer is followed by a ReLU, except for the last output layer. Dropout (Srivastava et al. 2014) with ratio 0.5 is used during training.

Fast Inference: In practical application, we only use the bounding boxes whose confidence is over 0.998. To accelerate this inference time, we first filter out the bounding boxes whose confidence is below 0.998 and whose range is out of the size of the image. This step can remove most of the uncorrelated bounding boxes and improve precision.

Training Depth Prediction Network

The depth prediction network is a tiny convolutional neural network with two fully connected layer on the top. It takes a RGB image as input. The size of the image is 144×256 pixels. For each depth map, we first normalize it between $[0, 1]$, then divide it into a 3×6 grid and calculate the average depth value for each cell. As a result, 18 values are used as the ground truths for training the depth prediction network. The depth prediction task can be formalized as a regression problem, and the objective loss is the Mean Squared Error (MSE) in depth prediction:

$$L_{depth} = \frac{1}{N} \sum_{i}^{N} \sum_{j}^{18} (y_i^j - f_{depth}(s_i)^j)^2, \tag{3}$$

where N is the batch size, y_i is the ground truth for image s_i, j is the index of the j-th depth value and $f_{depth}(\cdot)$ is the depth prediction function. Table 1 shows the architecture of the depth prediction model. Dropout (Srivastava et al. 2014) with ratio 0.5 is used during training. In this paper, we collected 10,000 images for training and use extra 2000 images as validation dataset.

Training Supervisory Network

We train a deep recurrent Q-learning Network(DRQN) as our supervisory network. The DRQN model gets one image(60×108 pixels) and two game variables(healthy value and ammo number) as inputs at each step and outputs the selection of Combo-Action. We use experience replay (Lin 1993) to offline training the model. The agent trajectories are stored in a replay memory, and the Q-learning updates are done on randomly sampled batches of experiences from the replay memory. Specifically, we use a history length of 5 to update the LSTM module. ϵ-greedy strategy is used during training: with a probability ϵ the next action is selected randomly, and with probability $1 - \epsilon$ according to the network's choice. We set ϵ starting from 1 and then progressively decaying to 0.1. The ground truths of bounding boxes and depth map can be directly used during training, so the auxiliary networks are only used during test. We follow the DRQN architecture setting in Arnold (Lample and Chaplot 2016).

Reward Shaping: Reward shaping (Ng, Harada, and Russell 1999) has to been shown to be an effective trick for RL training in a complicated environment. We found it helpful to give the agent a positive reward proportional to the displacement the agent makes, which pushes it to explore the environment. We also give positive reward to agent when picking up useful items(health, weapons and ammo). We give negative reward when it looses health and positive reward when it finds enemies. These two rewards encourage it to encounter more enemies. The rewards will be summed up during the executing period of the Combo-Action and the overall rewards will be given to the Combo-Action when the Combo-Action is finished or stopped. We summarize the rewards used in this paper as bellow:

- positive reward for finding new enemies.
- positive reward for object pickup (health, weapons and ammo)
- negative reward for loosing health
- positive reward proportional to the displacement it makes.

Experiments

In the experiments, we investigated how the Combo-Action influences the RL training process and how each part of our method influences the performance. We hold a league match for different algorithms to display the effectiveness of our method. We also compare our detection model with other baselines to exhibit our designed model is more precise and faster. In the following experiments, all the agents are evaluated under death-match scenario:

Death-match scenario: In the death-match scenario, all agents are put into the same environment to combat against each other. The score for the agent is called **Frags**, which is defined as the total number of killings minus the number of suicides.

Combo-Action Training

The ViZDoom provides build-in agents, which can make reasonable movements in the environment. All the agents are trained and tested with build-in agents.

Combo-Action Version: We trained the Combo-Action version agent on 10 different maps with 7 build-in Doom agents. Our agent plays 10 minutes per epoch to collect training data, and the supervisory network is optimized with RMSProp algorithm and with batch size of 32.

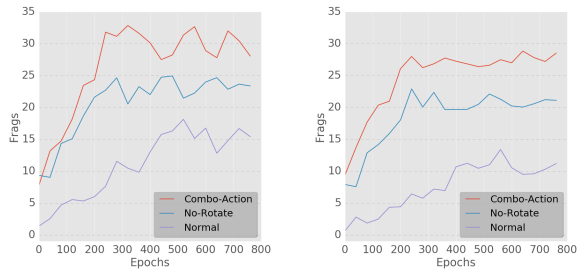
No-Rotate Version: We trained a no-Rotate version agent under the same setting, which the only difference is we drop the Rotate Combo-Action from the original Combo-Action version agent.

Frags	Deaths	Map01		Map02		Map03		Map04		Map05		Map06		Map07		Map08		Map09		Map10		Mean	
Marvin		2.3	11.0	0.5	17.6	-0.9	5.3	-2.6	22.9	0.3	8.8	2.9	19.1	2.5	14.4	-10.2	23.8	1.1	11.2	0.8	21.4	-0.33	15.55
IntelAct		7.6	11.9	2.0	20.5	**5.1**	3.8	3.4	21.0	4.6	14.0	5.2	21.5	4.5	13.1	2.8	14.6	4.5	13.1	0.3	24.3	4.0	15.78
YanShi		5.9	13.8	13.3	17.9	3.8	4.3	14.8	16.3	8.1	14.5	10.7	17.2	7.8	16.4	11.5	13.8	9.4	16.7	13.0	19.3	9.83	15.02
Arnold		8.8	**9.4**	7.7	21.4	2.2	4.6	1.4	22.4	9.9	15.4	9.9	16.0	6.8	15.1	4.0	16.9	12.3	19.7	6.6	24.0	6.96	16.49
Ours		**20.5**	9.8	**28.6**	**14.3**	2.3	**3.0**	**36.1**	**16.2**	**23.3**	**5.5**	**40.8**	**11.1**	**30.5**	**8.9**	**14.8**	**11.0**	**37.1**	**9.5**	**44.6**	**11.9**	**27.86**	**10.12**

Table 2: Our method vs previous methods in death-matches. All agents are put into the same unknown environment, and they are evaluated 10 rounds per map, 10 minutes per round. The average Frags and Deaths for each map are reported. Results show that our proposed method outperforms other methods by a large margin.

Normal Version: We also trained a normal version agent, which doesn't use Combo-Action. In the normal version, we follow the setting of the Arnold (Lample and Chaplot 2016), which only uses primitive actions.

Evaluation: We saved the agents at every 40 training epochs and evaluated them on two set of maps, and each set contains 10 maps. The first set of maps are used for training, which means the agent has seen the maps. The second set of maps are unknown to the agents. The agent will play on each map with 7 build-in agents for 10 minutes and the average Frags will be computed over each map set.



(a) (b)

Figure 3: The evaluation results along with training process. (a) The agents are test on 10 known maps with build-in Doom agents. (b) The agents are test on 10 unknown maps with build-in Doom agents. Compared with normal DRQN method, Combo-Action methods show an enormous leverage in performance and convergence speed.

Figure 3 shows the evaluation results along with training process. The agent trained with Combo-Action(Combo-Action version and No-Rotate version) achieves better performance and faster convergence speed. Interestingly, the randomly initialized agent in Combo-Action setting yields a good start line. This is because our Combo-Actions and auxiliary tasks give it some basic skills to play the FPS game. Compared with the no-Rotate version, we can tell that Rotate Combo-Action can improve the performance, and we further prove this point in Section 4.4.

Combo-Action Evaluation

Our Method vs Previous Methods: We let our trained agent combat with other state-of-the-art methods under death-match scenario. We choose 10 maps and put all agents into the same environment to play against each other. To our knowledge, we are the first in experiment to exhibit the performances of different methods under the same environment, and previous work only evaluated their agents with build-in Doom agents. To evaluate the agents, we chose 10 different maps, evaluated 10 rounds for each map and 10 minutes per round.

Baselines: (1) **Arnold** (Lample and Chaplot 2016): The agent was trained with DRQN and it took the first place in Track2 of ViZDoom AI Competition 2017. (2) **IntelAct** (Dosovitskiy and Koltun 2016): The agent was trained with future prediction and it took the first place in Track2 of ViZDoom AI Competition 2016. (3) **YanShi** (mihahauke): The agent used a two-stages structure, which contains perception module and planning module. It took the second place in Track2 of ViZDoom AI Competition 2017. (4) **Marvin** (mihahauke): The agent is trained with supervised replay of human play plus reinforcement learning. It took the forth place in Track2 of ViZDoom AI Competition 2017.

Table 2 shows the average Frags and Deaths on each map. Results show that our method gets the highest Frags and lowest Deaths on most of the maps. Compared with other end-to-end RL algorithms, our proposed method gets better performance and generalization on unknown maps.

ViZDoom AI Competition: We participated in the ViZDoom AI Competition 2018 to evaluate our agent. In the competition, all the participants need to submit their own agents and fight against each other on unknown maps. In this competition, our simplified agent achieved the first place in public-rank round and second place in private-rank round.

Detection Evaluation

Hyperparameters: We trained RPNmini for about $200,000$ steps with a batch size of 256, a momentum of 0.9 and a decay of 0.0005. Our learning rate schedule is as follows: For the first $195,000$ steps we start at a high learning rate 10^{-3}. Then we continue training with 10^{-4} for $3,000$ steps, and finally 10^{-5} for $2,000$ steps.

To evaluate the detection model, we follow the evaluation protocol of the Caltech pedestrian dataset (Dollar et al. 2012), which use ROC curves for 2D bounding box detection at overlap of 50% and 70%.

Baselines: We compare our approach with the following baselines: (1) **Faster-RCNN:** A deep neural network detec-

tor (Ren et al. 2015) using region proposals and classification pipeline, which is based on Resnet-101 (He et al. 2016). (2) **RPN+:** A deep neural network detector (Huang and Ramanan 2017) based on VGG16. All the detectors are trained and evaluated on the same dataset.

Detector	50% overlap	70% overlap	time
RPN+	47.59%	90.53%	0.63s
FasterRCNN	22.71%	48.85%	0.61s
RPNmini(ours)	**19.6%**	**36.98%**	**0.02s**

Table 3: Average miss rate of different detectors and the last column is the inference time of different methods. RPNmini not only achieves the best performance, but also use the least reference time.

Table 3 shows the average miss rate and time consuming of different detectors. We can see that our RPNmini detector achieves the best detection performance with minimal time cost. RPNmini can complete detection mission at 50 fps, which meets the real-time requirement in FPS game playing.

Ablation Investigation

In the ablation investigation, we want to answer following three questions: (1) How the Rotate Combo-Action influences the performance. (2) How the aiming ahead strategy influences the performance. (3) What if Combo-Actions are randomly chosen.

Scenario Construction: There are three questions mentioned above, so there are two options for each question and totally eight combinations. Accordingly, we construct eight different agents based on the combinations. We put all the eight agents into the same environment, which follows the death-match setting. We evaluated the agents on 10 maps with 10 round per map and 10 minutes per round. The average Frags and Hits over 100 matches are calculated. The Hits is defined as the number of effective hits the agent deals to its enemies.

		no Rotate		with Rotate	
		Frags	Hits	Frags	Hits
random	no ahead	14.87	171.98	16.50	186.65
	with ahead	13.44	153.03	17.62	189.17
normal	no ahead	19.58	205.96	20.09	214.44
	with ahead	18.74	198.83	**26.35**	**269.59**

Table 4: Average Frags and Average Hits under eight different scenarios for ablation investigation. All the agents are put into the same environment. All the agents are evaluated on 10 maps with 10 rounds per maps, 10 minutes per round.

Table 4 shows results of different scenarios. We can draw the following conclusions from the results:

(1) The Rotate Combo-Action can improve the agent's performance. The Rotate helps the agent to scout the environment and gives it more opportunities to find enemies.

(2) Aiming ahead is useful when the agent acts in reasonable manner. In some unreasonable setting, the history of the actions will mislead the aiming ahead strategy, which can result in performance degradation.

(3) Even the Combo-Actions are randomly chosen, the agent can still yield not-bad performance. This indicates that the priori knowledge in the Combo-Action gives the basic FPS playing skills to the agent. Our previous experiments also prove priori knowledge can alleviate the training difficulty for the FPS game.

Conclusion

We have explored the method which applied Combo-Action in a famous FPS game. Our method can utilize priori knowledge and extra supervised signal to boost the ability of the agent. And the reduced action space makes the training process more efficient and let the agent behave in a more harmonious manner. Experiments show that our trained agent gains a significant performance improvement compared with previous approaches. Up to present, all the agents for ViZDoom are trained with build-in agents and recent researches (Conitzer and Sandholm 2007; Silver et al. 2017; Bansal et al. 2017) show that self-play will result in more powerful agents and reduce human biases. In the future work, we'd like to form the death-match task as a multi-agent problem and try to train the agent in a self-play scenario.

Acknowledgments

This work was supported by the National Key Research and Development Program of China (No.2017YFA0700904), NSFC projects (Nos. 61620106010, 61621136008, 61332007), the MIIT Grant of Int. Man. Comp. Stan (No. 2016ZXFB00001), Tsinghua Tiangong Institute for Intelligent Computing, the NVIDIA NVAIL Program and a Project from Siemens. We thank Dong Yan, Jia Xu and Peng Sun for inspiring discussions.

References

Arulkumaran, K.; Dilokthanakul, N.; Shanahan, M.; and Bharath, A. A. 2016. Classifying options for deep reinforcement learning. *arXiv preprint arXiv:1604.08153*.

Bacon, P.-L.; Harb, J.; and Precup, D. 2017. The option-critic architecture. In *AAAI*, 1726–1734.

Bansal, T.; Pachocki, J.; Sidor, S.; Sutskever, I.; and Mordatch, I. 2017. Emergent complexity via multi-agent competition. *arXiv preprint arXiv:1710.03748*.

Bhatti, S.; Desmaison, A.; Miksik, O.; Nardelli, N.; Siddharth, N.; and Torr, P. H. 2016. Playing doom with slam-augmented deep reinforcement learning. *arXiv preprint arXiv:1612.00380*.

Conitzer, V., and Sandholm, T. 2007. Awesome: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *Machine Learning* 67(1-2):23–43.

Dollar, P.; Wojek, C.; Schiele, B.; and Perona, P. 2012. Pedestrian detection: An evaluation of the state of the art. *IEEE transactions on pattern analysis and machine intelligence* 34(4):743–761.

Dosovitskiy, A., and Koltun, V. 2016. Learning to act by predicting the future. *arXiv preprint arXiv:1611.01779*.

Erhan, D.; Szegedy, C.; Toshev, A.; and Anguelov, D. 2014. Scalable object detection using deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2147–2154.

Frans, K.; Ho, J.; Chen, X.; Abbeel, P.; and Schulman, J. 2017. Meta learning shared hierarchies. *arXiv preprint arXiv:1710.09767*.

Hausknecht, M., and Stone, P. 2015. Deep recurrent q-learning for partially observable mdps. *CoRR, abs/1507.06527* 7(1).

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 770–778.

He, R.; Brunskill, E.; and Roy, N. 2010. Puma: Planning under uncertainty with macro-actions. In *AAAI*.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Huang, S., and Ramanan, D. 2017. Expecting the unexpected: Training detectors for unusual pedestrians with adversarial imposters. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Kempka, M.; Wydmuch, M.; Runc, G.; Toczek, J.; and Jaśkowski, W. 2016. Vizdoom: A doom-based ai research platform for visual reinforcement learning. *arXiv preprint arXiv:1605.02097*.

Lample, G., and Chaplot, D. S. 2016. Playing fps games with deep reinforcement learning. *arXiv preprint arXiv:1609.05521*.

Lin, L.-J. 1993. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science.

mihahauke. Doom agents. http://www.cs.put.poznan.pl/mkempka/misc/vdaic2017_agents/.

Mirowski, P.; Pascanu, R.; Viola, F.; Soyer, H.; Ballard, A. J.; Banino, A.; Denil, M.; Goroshin, R.; Sifre, L.; Kavukcuoglu, K.; et al. 2016. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*.

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

Ng, A. Y.; Harada, D.; and Russell, S. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, 278–287.

Precup, D. 2000. *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, 91–99.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15(1):1929–1958.

Sutton, R. S.; Barto, A. G.; et al. 1998. *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* 112(1-2):181–211.

Tessler, C.; Givony, S.; Zahavy, T.; Mankowitz, D. J.; and Mannor, S. 2017. A deep hierarchical approach to lifelong learning in minecraft. In *AAAI*, volume 3, 6.

van Waveren, J. 2001. The quake iii arena bot. *University of Technology Delft*.

Wu, Y., and Tian, Y. 2017. Training agent for first-person shooter game with actor-critic curriculum learning. In *Submitted to International Conference on Learning Representations*.