Proceedings of the 37th Chinese Control Conference
July 25-27, 2018, Wuhan, China

V–D D3QN: The Variant of Double Deep Q-Learning Network with Dueling Architecture

Ying Huang, GuoLiang Wei, YongXiong Wang
School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai 200093, P. R. China
E-mail: wyxiong@usst.edu.cn

Abstract: The fashionable DQN algorithm suffers from substantial overestimations of action-state value in reinforcement learning problem, such as games in the Atari 2600 domain and path planning domain. To reduce the overestimations of action values during learning, we present a novel combination of double Q-learning and dueling DQN algorithm, and design an algorithm called Variant of Double dueling DQN (V–D D3QN). We focus on the idea behind V–D D3QN algorithm and propose the feasible idea of using two dueling DQN networks to reduce the overestimations of action values during training, and the specific approach is to randomly select one dueling DQN network at each time step to update its parameters, by exploiting the remaining dueling DQN network to determine the update targets. And then we do our experiments in the customized virtual environment– gridmap. Our experiments demonstrate that our proposed algorithm not only reduces the overestimations more efficiently than Double DQN(DDQN) algorithm, but also leads to much better performance on route planning domain with great generalization ability of the new and rapidly changing environments.

Key Words: Deep Reinforcement Learning, Path Planning, Overestimations, Double Q-learning

1 Introduction

The goal of reinforcement learning[10, 12] is to learn good policies for sequential decision problems by maximizing its cumulative future rewards. Q-learning[16] is a popular reinforcement learning algorithm, which is known to learn over-optimistic action values because it includes a maximize operator when determining its update targets. Overestimations of Q-learning processes are first investigated in Thrun S et al. 1993 [13] and it shows that there are upper bounds for them. Then Van Hasselt et al. 2015 [4] noted that it is also possible to derive a lower bound of overestimations under certain conditions and any method can invite value inaccuracies during learning, simply due to the initially unknown ground-truth Q-values. In previous works, overestimations have always been attributed to insufficiently flexible function approximation[10, 13, 14] and noise[4]. And the performance of the recent DQN algorithms[4, 7, 15] in the Atari domain revealed that the overestimations of action values indeed occur at scale in practice.

Until now, there are two ways to reduce the overestimations of action values: one is the Averaged–DQN algorithm[1], which reduces the Target Approximation Error directly, and the another is the Double Q-learning algorithm[3, 4] which can decouple the action selection operation from the action evaluation operation.

In this paper, we introduce a V–D D3QN algorithm that combines the double Q-learning and DQN with dueling architecture in a novel way to reduce overestimations. We demonstrate that the proposed algorithm can not only reduce the overestimations more efficiently in the reinforcement learning problem, but also can achieve the state-of-the-art performance on the route planning domain. Additionally, we provide experimental results on customized virtual route planning testbed – gridmap.

We conclude the main contributions of this paper as follows:

1) A novel extension to the DDQN algorithm called V–D D3QN algorithm which stabilizes training process and upgrades the performance of learned policies by using a novel combination of Double Q-learning and dueling DQN.

2) Experiments in customized gridmap environment show that the proposed V–D D3QN algorithm can obtain state-of-the-art results on the route planning domain.

3) Experiments reveal that the proposed V–D D3QN algorithm possesses great generalization ability of the unfamiliar and rapidly changing environments.

4) By analyzing the experimental results, we draw a conclusion that: with enough training steps, the two dueling DQNs in V–D D3QN algorithm are equivalent in both training and evaluating process.

2 Background

Q-learning is a popular reinforcement learning algorithm that is proposed in Watkins et al. 1993 [16] and can be used to solve the Markov Decision Processes[10, 12]. To solve sequential decision-making problems, we learn estimates that are bootstrapped from optimal action values, which are defined as the expected value of total future rewards when taking the action a and following the policy π thereafter. Under a given policy π, the ground-truth value of an action a in state s is

$$Q_\pi(s,a) = \mathbb{E}[r_1 + \gamma r_2 + ...|s_0 = s, a_0 = a, \pi], \quad (1)$$

where $\gamma \in [0,1]$ is a discount factor that compromises the importance between immediate reward and future rewards, and the $\mathbb{E}[x]$ is to find the expectation of random variable x. The optimal value is $Q^*(s,a) = \max_\pi Q_\pi(s,a)$. Usually, we use tubular Q-learning algorithm to learn the estimates for the optimal action values[16]. But in practice, we generally use function approximation methods[10, 14] to learn a parameterized value function $\boldsymbol{Q}(s,a;\boldsymbol{\theta_t})$ that can represen-

This work is supported by National Natural Science Foundation (NNSF) of China under Grant 61673276.

t values for all the possible action-state pairs in a given state s, because the action space and the state space are usually too large to separately learn values for all action-state pairs. The standard Q-learning with function approximation method updates the parameters in Q function after taking action a_t in state s_t, and then obtain the immediate reward r_{t+1} and observe the successor state s_{t+1}. The update formula is as follows

$$\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} + \eta(y_t^Q - \boldsymbol{Q}(s_t, a_t; \boldsymbol{\theta_t}))\nabla_{\boldsymbol{\theta_t}}\boldsymbol{Q}(s_t, a_t; \boldsymbol{\theta_t}), \quad (2)$$

where η is a scalar step size and the update target y_t^Q for Q-learning is defined as

$$y_t^Q = r_{t+1} + \gamma \max_{a'} \boldsymbol{Q}(s_{t+1}, a'; \boldsymbol{\theta_t}). \quad (3)$$

2.1 Deep Q-Networks

A deep Q-network (DQN) is a multi-layered neural network that is used to approximate action value function, and in a given state s it outputs a vector of action values $\boldsymbol{Q}(s, \cdot; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are the parameters of the neural network. For a n-dimensional state space and an action space whose capacity is m, the neural network is a mapping function from $\mathbb{R}^n$ to $\mathbb{R}^m$. Two significant ingredients of DQN algorithm are the experience replay technique and the use of dual network[7], which means to use an online network and a target network. The target network, with parameters $\boldsymbol{\theta}^-$, is the same as the online network whose parameters are $\boldsymbol{\theta}$, except that its parameters are copied every τ steps from the online network. So we update parameters of online network at each time step, and the parameters of target network are updated in every τ steps from the online network and then are kept fixed on all other steps[7]. And the target network is introduced to compute the update targets, so we can derive the update target for DQN at each step from formula (4)

$$y_t^{DQN} = r_{t+1} + \gamma \max_{a'} \boldsymbol{Q}(s_{t+1}, a'; \boldsymbol{\theta_t^-}). \quad (4)$$

As for the technique of experience replay, the observed transitions are stored in a buffer and then are randomly sampled to update the online network as formula (2). The two techniques mentioned above stabilize the training of DQN.

2.2 Double Q-learning

The maximize operator in formula (3) and (4) use the same Q-values to select and evaluate the greedy action. The consistency in action selection and action assessment makes it more prone to select the overestimated values as the estimates of true action values. To prevent this, Double Q-learning[3] decouples the action selection from action evaluation. The Double Q-learning algorithm has two Q-networks separately with weights $\boldsymbol{\theta^A}$ and $\boldsymbol{\theta^B}$. To obtain the update target at each step, it randomly choose one set of weights to determine the greedy policy in the given state s_{t+1} and let the remaining set of weights to determine its corresponding action value. Accordingly, the target of Double Q-learning to update Q-network A can be written as follows

$$y_t^{Q^A} = r_{t+1} + \gamma \boldsymbol{Q}(s_{t+1}, arg\max_{a'} \boldsymbol{Q}(s_{t+1}, a'; \boldsymbol{\theta^A}); \boldsymbol{\theta^B}). \quad (5)$$

Formula (5) shows that we are still using the current online Q-network A to choose the greedy policies, and then using Q-network B to estimate the value of the corresponding greedy policies. Conversely, if we want to update the parameters of Q-network B, then swap the roles of Q-network A and Q-network B. So the update targets for the second set of weights $\boldsymbol{\theta^B}$ can be written as following

$$y_t^{Q^B} = r_{t+1} + \gamma \boldsymbol{Q}(s_{t+1}, arg\max_{a'} \boldsymbol{Q}(s_{t+1}, a'; \boldsymbol{\theta^B}); \boldsymbol{\theta^A}). \quad (6)$$

Double Q-learning stores two Q functions, and each Q function is updated with Q-values from the remaining Q function for the next state.

2.3 Double Deep Q-Network

To reduce the overestimations of action values in the training process of DQN algorithm, Van Hasselt et al. 2015 proposed DDQN algorithm and noted that it is also possible to derive a lower bound of overestimations under certain conditions[4]. Also, it noted that any method can invite value inaccuracies during learning, simply due to the initially unknown ground-truth Q-values[4]. And those estimates can further deteriorate if we bootstrap from action values that are already overoptimistic. In DDQN algorithm, the target network in the DQN architecture[7] provides a natural candidate for the second Q function requested by Double Q-learning. Then the update target for DDQN algorithm is

$$y_t^{DDQN} = r_{t+1} + \gamma \boldsymbol{Q}(s_{t+1}, arg\max_{a'} \boldsymbol{Q}(s_{t+1}, a'; \boldsymbol{\theta_t}); \boldsymbol{\theta_t^-}). \quad (7)$$

It is worth noting that, unlike Double Q-learning, here the role of the online network and target network are stationary and then the target network is updated just as DQN does.

3 Variant-Double DQN with Dueling Architecture

Van Hasselt et al. 2015 [4] has proposed the minimal possible change to DQN towards Double Q-learning[3] which is called DDQN. But we still wonder if there are algorithms that are superior to DDQN in solving overestimations problem. Given this doubt, we propose a novel algorithm called V–D D3QN which is a simple variant to DDQN in the mode of integration of Double Q-learning and dueling DQN, specific details see Table 1. To further stabilize the training process, we use the DQN with dueling architecture[15] which has two separate estimators: one estimator is used to estimate the state value function $V(s)$ and the other is used to represent the state-dependent action advantage function $\boldsymbol{A}(s, a)$, and then the outputs of the two separate estimators can be integrated as following

$$\begin{aligned} &\boldsymbol{Q}(s, a; \boldsymbol{\omega}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = V(s; \boldsymbol{\omega}, \boldsymbol{\beta}) + \\ &(\boldsymbol{A}(s, a; \boldsymbol{\omega}, \boldsymbol{\alpha}) - \frac{1}{|\mathcal{A}|}\sum\nolimits_{a'} \boldsymbol{A}(s, a'; \boldsymbol{\omega}, \boldsymbol{\alpha})), \end{aligned} \quad (8)$$

where we make the last hidden convolution layer of original DQN[7] evenly be divided into two separate fully-connected streams, one stream outputs a scalar $V(s; \boldsymbol{\omega}, \boldsymbol{\beta})$, and the remaining stream outputs a $|\mathcal{A}|$-dimensional vector $\boldsymbol{A}(s, a; \boldsymbol{\omega}, \boldsymbol{\alpha})$. Here, $\boldsymbol{\omega}$ denotes the parameters of the convolution layers, while $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are the parameters of the two separate fully-connected streams[15]. Fig. 1 shows the architecture of the proposed V–D D3QN. The dueling DQN network A (**top**) and dueling DQN network B (**bottom**), which are two DQNs with dueling architecture, are independent in their structure and are trained together to represent two Q-value functions.

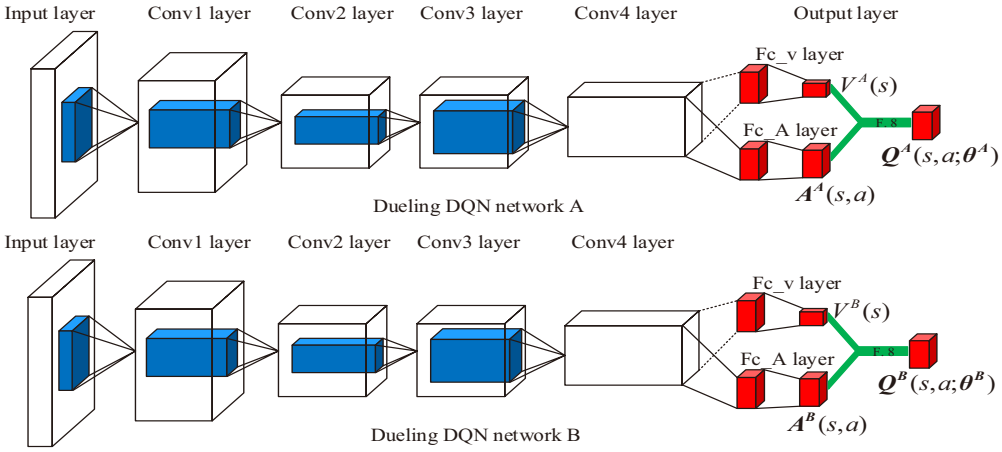

Fig. 1: Our used network architecture

Table 1: Algorithm 1

1:Initialize two dueling DQN networks $\boldsymbol{Q^A}(s,a;\boldsymbol{\theta^A})$ and $\boldsymbol{Q^B}(s,a;\boldsymbol{\theta^B})$ with random weights $\boldsymbol{\theta_0^A}$ and weights $\boldsymbol{\theta_0^B}$.
2: Initialize experience replay buffer $\mathcal{B}$ to capacity N.
3: Initialize exploration procedure as ϵ_t -greedy policy.
4: for episode =1, $Num\_episodes$ do
5: Reset the environment and preprocess the input sequence s_0.
6: **for** t=1, max_length **do**
7: With probability ϵ_t select a random action a_t, otherwise select action greedily on the averaged outputs of the two dueling DQNs.
8: Execute action a_t in emulator and observe reward r_{t+1} and the next observation s_{t+1}.
9: Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in ER buffer $\mathcal{B}$.
10: Sample random mini-batch of transitions $(s_j, a_j, r_{j+1}, s_{j+1})$ from $\mathcal{B}$.
11: Choose randomly either *UPDATE(A)* or *UPDATE(B)*.
12: **if** *UPDATE(A)* **then**
13: Define $a_A^* = arg\max_{a'} \boldsymbol{Q^A}(s_{j+1}, a'; \boldsymbol{\theta_t^A})$.
14: Set

$$y_t^{V-DD3QN(A)} = \begin{cases} r_{j+1}, s_{j+1}\ is\ terminal\ state \\ r_{j+1} + \gamma \boldsymbol{Q^B}(s_{j+1}, a_A^*; \boldsymbol{\theta_t^B}), else \end{cases}.$$

15: Perform an Adam gradient descent step on $(y_t^{V-DD3QN(A)} - \boldsymbol{Q^A}(s_t, a_t; \boldsymbol{\theta_t^A}))^2$.
16: **else if** *UPDATE(B)* **then**
17: Define $a_B^* = arg\max_{a'} \boldsymbol{Q^B}(s_{j+1}, a'; \boldsymbol{\theta_t^B})$.
18: Set

$$y_t^{V-DD3QN(B)} = \begin{cases} r_{j+1}, s_{j+1}\ is\ terminal\ state \\ r_{j+1} + \gamma \boldsymbol{Q^A}(s_{j+1}, a_B^*; \boldsymbol{\theta_t^A}), else \end{cases}.$$

19: Perform an Adam gradient descent step on $(y_t^{V-DD3QN(B)} - \boldsymbol{Q^B}(s_t, a_t; \boldsymbol{\theta_t^B}))^2$.
20: **end if**
21: $s_t \leftarrow s_{t+1}$
22: **end for**
23: **end for**

At each time step, we randomly choose a dueling DQN in Fig.1 to be online network and make the remaining one act as the target network, then select the greedy policies according to the average outputs of the online network and target network, and in the end we employ the update target bootstrapped from the current target network to update the parameters of current online network and keep the weights of current target network fixed. At each time step, the online network is selected from dueling DQN A and B randomly and can be different between time steps. In V–D D3QN, the roles dueling DQN A and B play may varying between every two steps. Therefore, V–D D3QN algorithm is innovative in the way that combines Double Q-learning and dueling DQN.

V–D D3QN has two dueling DQNs, separately representing two Q functions: $\boldsymbol{Q^A}(\cdot;\boldsymbol{\theta^A})$ and $\boldsymbol{Q^B}(\cdot;\boldsymbol{\theta^B})$. Here, $\boldsymbol{\theta^A} \equiv (\boldsymbol{\omega^A}, \boldsymbol{\alpha^A}, \boldsymbol{\beta^A})$ and $\boldsymbol{\theta^B} \equiv (\boldsymbol{\omega^B}, \boldsymbol{\alpha^B}, \boldsymbol{\beta^B})$ are respectively the overall weights of dueling DQN A and B. At each step in an episode, we execute an action a_t which is produced by ϵ_t-greedy policy, then the agent gets the immediate reward r_{t+1} and next observation s_{t+1} from environment. In the meanwhile, we store the transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in experience replay buffer $\mathcal{B}$. The most creative point is that the agent then randomly choose one dueling DQN to update with the mini-batch sampled transitions, just as Double Q-learning algorithm does. Each Q function is updated with an action value from the remaining Q function for the next state. In this way, the dueling DQN and Double Q-learning are integrated naturally. In Table 1, the action a_A^* in line 13 is the greedy action in state s_{j+1}, according to the Q function $\boldsymbol{Q^A}(s,a;\boldsymbol{\theta_t^A})$. We use the action value $\boldsymbol{Q^B}(s_{j+1}, a_A^*; \boldsymbol{\theta_t^B})$ to update $\boldsymbol{Q^A}(s,a;\boldsymbol{\theta^A})$, instead of the value $\boldsymbol{Q^A}(s_{j+1}, a_A^*; \boldsymbol{\theta_t^A})$. A similar update is used for $\boldsymbol{Q^B}(s,a;\boldsymbol{\theta^B})$, using a_B^* and $\boldsymbol{Q^A}(s_{j+1}, a_B^*; \boldsymbol{\theta_t^A})$.Since $\boldsymbol{Q^B}(s,a;\boldsymbol{\theta^B})$ is updated symmetrically on the same problem, but with a different set of experience samples, those estimations can be considered as unbiased estimates for action values in given state.

The details of training process are illustrated in Table 1. As for the evaluating process, we must consider the relationship between the trained dueling DQN A and B since they are updated symmetrically. And we will find out whether the two trained dueling DQNs are equivalent through experiments on the route planning domain.

4 Gridmap

4.1 Gridmap Environment

The gridmap (see Fig. 2 left) is a variant of environment Gridworld (see Fig. 2 right), which has a start, a goal and lots of obstacles. In this problem, we aim to reach the goal as fast as the agent can, and avoid all the obstacles at the same time. The gridmap problem is a common RL testbed, it not only allows the experience replay buffer to accommodate all possible state-action pairs but also makes it easy to accurately calculate the optimal value function[4].

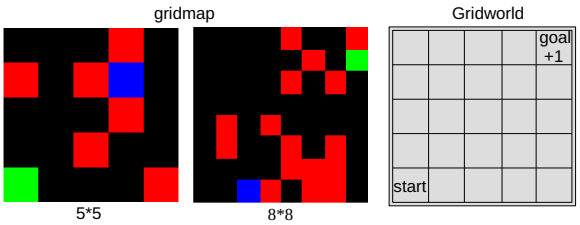

Fig. 2: Left: gridmap with different size; right: Gridworld

4.2 Environment Setup

In our experiments on the gridmap problem, the state space contains pairs of points from a 2D discrete grid ($S = \{(x, y)\}_{x,y\in 1,\ldots,84}$), and the action space includes four actions that correspond to steps in each compass direction. The agent interacts with the environment through raw pixel figures and one-hot feature actions. The agent starts at the blue block and sets the green block as the goal we want to reach. Among the gridmap, the red blocks are the obstacles which the agent should avoid. In our experiments, a reward of +1 is obtained if the agent reaches the goal and a reward -1 is obtained if the agent crashes into the obstacles or the walls. Otherwise, every step forward the agent wins a reward of -0.01. When we are training the agent, we randomly place the start, the goal and the obstacles for every episode, so the trained agent is robust and has great generalization ability of the new and rapidly changing environments.

5 Experiments

5.1 Structure and Hyper-parameters

In this section, we show that the V–D D3QN algorithm improves over DQN and DDQN both in accuracy of predicted Q-values and quality of policies. We design our path planning agents in customized gridmap environment and train them with the same architecture in Fig. 1 and same hyper-parameters setting in Table 2 across all algorithms.

We closely follow the network architecture used in paper [4] and [15] with a little change. Our network architecture is a convolution neural network with four convolution layers followed by rectified linear units[8] and two fully-connected hidden layers followed by linear units, and these four convolution layers are respectively convolving with 8×8 convolution kernel and stride 4, 4×4 kernel and stride 2, 3×3 kernel and stride 1, 7×7 kernel and stride 1. Here, $|\mathcal{A}| = 4$.

5.2 Results on Overestimations

Fig. 3 shows the averaged predicted Q-values during the training processes of the V–D D3QN,DQN and DDQN agents. These curves are obtained by training in one million episodes under the same conditions of map size 8×8 with DQN, DDQN and V–D D3QN algorithms and are the average curves of 10 separate training processes.

In our experiments, we reflect the overestimations in DQN by the averaged predicted Q-values[1] without true Q-values, since we know that the overestimations are indeed exists in process that involved Q-learning. Here, the averaged predicted Q-value estimates are computed regularly during training with full evaluation phases of length $T = 125,000$ steps as DDQN [4] does. By comparing the red learning curve of V–D D3QN to the green curve of D-DQN and blue curve of DQN in Fig. 3, we know that V–D D3QN algorithm can consistently reduce the averaged predicted Q-values of current policy more efficiently, which representing the fact that V–D D3QN algorithm reduces overestimations more efficiently than DDQN algorithm.

Fig. 4 shows the discounted return and success rate in all the training processes of gridmap with size 8×8. These curves are the average curves of 10 separate training processes. The red learning curves for V–D D3QN are much better to the curves for DDQN and DQN. This indicates that V–D D3QN algorithm not only brings about more accurate value estimates for all possible state-action pairs but also leads to more outstanding policies than DDQN does.

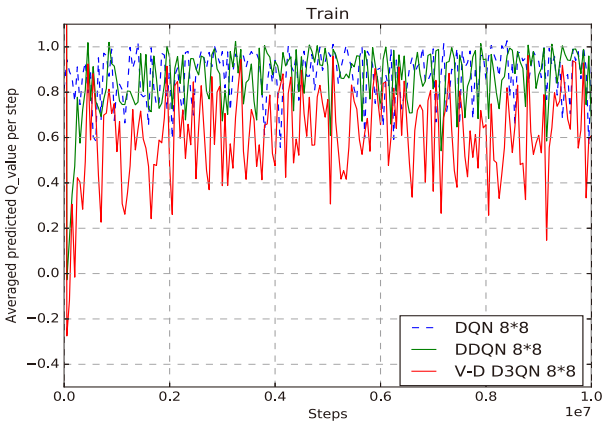

Fig. 3: Averaged estimated Q-values in training process

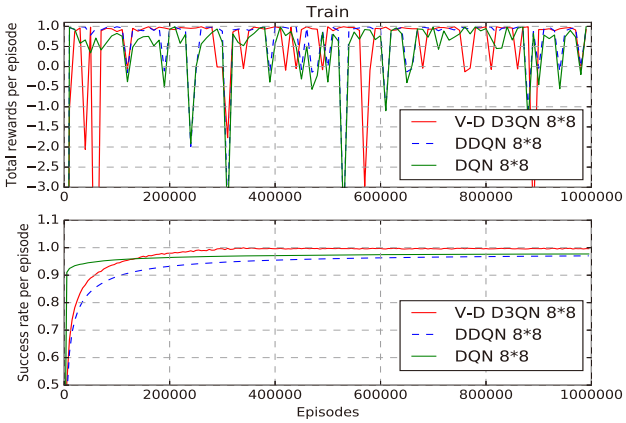

Fig. 4: The discounted return (**top**) and success rate (**bottom**) per episode in training processes with size 8×8

Table 2: The hyper-parameters we use in this paper

Hyper-parameter Name	Value	Detailed Meaning
Mini-batch	32	Number of sampled transitions each step
τ	4	Update frequency of the Target Network in DQN
γ	0.99	Discounted factor
Anneling steps	10000	The steps ϵ change from 1 to 0.1
pre-train steps	10000	The steps the agent need to pre-train and fill experience replay buffer $\mathcal{B}$
N	50000	Capacity of buffer $\mathcal{B}$
α	$1e^{-4}$	Learning rate for gradient descent
φ	0.001	Proportion of the parameters of Target network close to the Online networks parameters
ϵ_t	[0.1,1]	The probability of agent randomly choose an action when we are training agents
ϵ	0.01	The probability of agent randomly choose an action when we are evaluating agents

5.3 Quality of the Learned Policies

We now demonstrate how much V–D D3QN helps in terms of policy quality by evaluating the trained agents on the gridmap. To assess the robustness and the generalization ability of the trained agents, we evaluate our trained agents of three algorithms respectively in 10000 episodes in which the starts, the goals and the obstacles are all placed randomly. Whats more, we evaluate all the trained agents with both ϵ-greedy exploration and greedy policy in both sizes of 5×5 and 8×8. Here we consider three comparisons: V–D D3QN vs. DDQN, V–D D3QN vs. DQN, and DDQN vs. DQN.

Four metrics are used to measure the path planning performance[9], including **success rate** – the probability of successfully reach the goal without hitting any obstacles (Higher means better); **path difference** – the average path difference between the predicted path and the shortest path (Lower means better), here the shortest path is found by A^* algorithm; **expected reward** – the average discounted reward in an episode(Higher means better);and **average steps** – the average steps of all evaluating episodes cost to reach the goal (Lower, better). The overall result is showed in Table 3, where **Succ Rate**, **Path Diff**, **Exp R** and **Ave Step** are the abbreviations for the four mentioned metrics in order. As depicted by the **bold face in Table 3**, the policies learned by V–D D3QN agents are better than DDQN and DQN do, except for the path difference (specific reasons are elaborated in subsection **Result Analysis**). Moreover, Fig. 5 plots the success rate while evaluating all learned agents of gridmap with size 8×8. In Fig. 5, every curve is the average of 100 curves of the evaluating process in 10,000 independent episodes. Fig. 5 tells a truth: under the same exploration conditions, V–D D3QN agents learn the generally best and relatively more stable control strategies, and under the same algorithm, the average performance of evaluating with the ϵ-greedy exploration policy($\epsilon = 0.01$) is better than the performance of evaluating with greedy strategy($\epsilon = 0$).

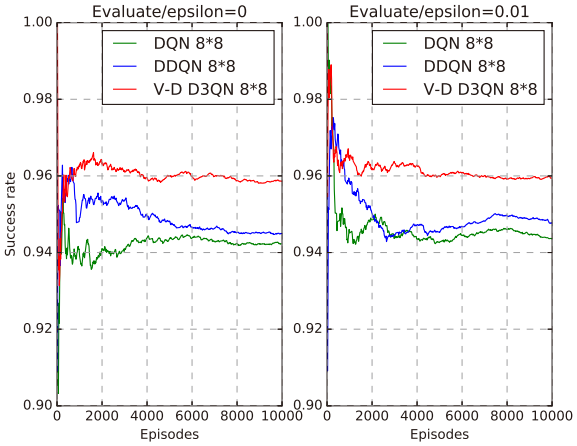

Fig. 5: Average success rate in evaluating processes with size 8×8 of V–D D3QN, DDQN and DQN agents(ϵ=0 and 0.01)

In V–D D3QN algorithm, we use the average Q-outputs of the two dueling DQN networks to select the actual action at each step when training. Theoretically, as long as there are enough training steps, two equivalent Q functions can be obtained. Thus, we must figure out whether the two trained dueling DQNs are equivalent and which network should we use to make our decisions when we are evaluating the trained V–D D3QN agents. Should the agent use the combined output of the two dueling DQNs, or randomly choose a dueling DQN to make decisions?

Through experiments in the same conditions, the results are showed above in Table 4, and we figure out that the V–D D3QN agent randomly chooses a dueling DQN to make decisions (**left columns**) achieves better performance than V–D D3QN agent that uses the combined output of the two dueling DQNs to make decisions (**right columns**). Summarize these results, we conclude that with enough training, the two dueling DQNs in V–D D3QN algorithm are approximately equivalent in both training and evaluating process.

Table 3: Gridmap path planning performance comparison for V–D D3QN vs. DDQN vs. DQN

Algorithm	**DQN**				**DDQN**				**V–D D3QN**			
Epsilon	ϵ=0		ϵ=0.01		ϵ=0		ϵ=0.01		ϵ=0		ϵ=0.01	
Map Size	5×5	8×8	5×5	8×8	5×5	8×8	5×5	8×8	5×5	8×8	5×5	8×8
Succ Rate	99.31%	94.24%	99.36%	94.36%	99.43%	94.49%	99.50%	94.75%	99.85%	95.87%	**99.86%**	**95.94%**
Path Diff	**0.015**	0.071	0.018	0.071	0.016	0.071	0.020	**0.070**	0.018	0.072	0.021	0.071
Exp R	0.956	0.836	0.944	0.839	0.958	0.840	0.946	0.832	0.954	0.836	**0.966**	**0.850**
Ave Step	4.72	11.61	4.76	11.43	4.64	11.40	4.65	11.13	4.35	10.43	**4.29**	**10.39**

Table 4: Gridmap path planning performance comparison for Random network vs. Combined Network

Algorithm	**V–D D3QN**							
Action select	**Random network**				**Combined Network**			
Epsilon	ϵ=0		ϵ=0.01		ϵ=0		ϵ=0.01	
Map Size	5×5	8×8	5×5	8×8	5×5	8×8	5×5	8×8
Succ Rate	99.85%	95.87%	**99.86%**	**95.94%**	99.51%	94.95%	99.61%	95.01%
Path Diff	0.018	0.072	0.021	0.071	**0.016**	0.075	0.019	**0.067**
Exp R	0.954	0.836	**0.966**	**0.850**	0.961	0.815	0.945	0.846
Ave Step	4.35	10.43	**4.29**	**10.39**	4.45	11.25	4.49	10.93

5.4 Results Analysis

Overall, the results suggest that V–D D3QN algorithm reduces the overestimations more efficiently than DDQN does, and leads to significantly improved performance. Although V–D D3QN algorithm reduces the overestimation more efficiently than DDQN algorithm and the success rate of the learned strategy is higher than DDQN algorithm, it sometimes sacrifices the optimality of the path. Thats to say, the trained V–D D3QN agents cant always find out the optimal path and sometimes takes some detours. I think this may owe to the randomness of the networks training and evaluating process. And the different sources of two consecutive actions may lead to the less optimal path when planning.

With regards to the success rate, since the starts, the goal and the obstacles of all episodes are randomly placed in gridmap when training, so the state space is very spacious and limited millions of training episodes cant guarantee to traverse all possible states at least once. Therefore, the path planning agent is sometimes trapped in a local dead zone and hovering in several positions. Fig. 6 shows some successful and failed episodes in gridmap problem.

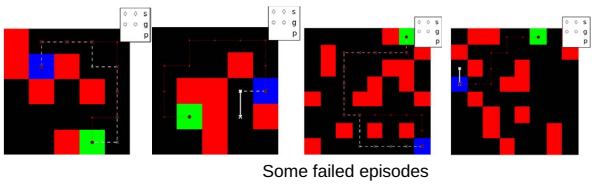

Fig. 6: Some successful episodes and failed episodes in gridmap with sizes 5×5 and 8×8

6 Conclusion and Future Directions

This paper has three major contributions. First, we propose the V–D D3QN algorithm that combines dueling DQN with Double Q-learning in a novel way. Second, experiments with the route planning problem in customized gridmap environment demonstrate the proposed algorithm can not only reduce the overestimations of action values more efficiently,but also find better policies than DDQN. Finally, we reveal that V–D D3QN algorithm has great generalization ability of rapidly changing environments and can obtain new state-of-the-art results on the route planning domain.

Variant-Double Q-learning can be easily combined with other DQN variants[11] and other algorithms, such as Count-based exploration[2] and Deep Residual Network[5]. In future work, we would like to combine some other reinforcement learning algorithms[10, 12], like SARSA and Actor-Critic, with the variant-double operator and Deep Neural Network. Moreover, we want to find methods that can further reduce the overestimations of action values.

References

[1] Anschel O, Baram N, Shimkin N. Averaged–dqn: Variance reduction and stabilization for deep reinforcement learning,in *International Conference on Machine Learning*. 2017: 176-185.

[2] Bellemare M, Srinivasan S, Ostrovski G, et al. Unifying count-based exploration and intrinsic motivation, in *Advances in Neural Information Processing Systems*. 2016: 1471-1479.

[3] Hasselt H V. Double Q-learning, in *Advances in Neural Information Processing Systems*. 2010: 2613-2621.

[4] Van Hasselt H, Guez A, Silver D. Deep Reinforcement Learning with Double Q-Learning, in *AAAI*. 2016: 2094-2100.

[5] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition,in *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016: 770-778.

[6] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning. *arXiv preprint arXiv*:1312.5602, 2013.

[7] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning. *Nature*, 2015, 518(7540): 529-533.

[8] Nair V, Hinton G E. Rectified linear units improve restricted boltzmann machines,in *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010: 807-814.

[9] Niu S, Chen S, Guo H, et al. Generalized Value Iteration Networks: Life Beyond Lattices. *arXiv preprint arXiv*:1706.02416, 2017.

[10] Sutton R S, Barto A G. *Reinforcement learning: An introduction*. Cambridge: MIT press, 1998.

[11] Schaul T, Quan J, Antonoglou I, et al. Prioritized experience replay. *arXiv preprint arXiv*:1511.05952, 2015.

[12] Szepesvri C. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 2010, 4(1): 1-103.

[13] Thrun S, Schwartz A. Issues in using function approximation for reinforcement learning,in *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*. 1993.

[14] Tsitsiklis J N, Van Roy B. An analysis of temporal-difference learning with function approximationTechnical. *Report LIDS-P-2322)*.Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 1996.

[15] Wang Z, Schaul T, Hessel M, et al. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv*:1511.06581, 2015.

[16] Watkins C J C H, Dayan P. Q-learning. *Machine learning*, 1992, 8(3-4): 279-292.