Applying Q(λ)-learning in Deep Reinforcement Learning to Play Atari Games

Seyed Sajad Mousavi
National University of Ireland, Galway
s.mousavi1@nuiglaway.ie

Michael Schukat
National University of Ireland, Galway
Michael.schukat@nuigalway.ie

Patrick Mannion
National University of Ireland, Galway
p.mannion3@nuigalway.ie

Enda Howley
National University of Ireland, Galway
ehowley@nuigalway.ie

ABSTRACT

In order to accelerate the learning process in high dimensional reinforcement learning problems, TD methods such as Q-learning and Sarsa are usually combined with eligibility traces. The recently introduced DQN (Deep Q-Network) algorithm, which is a combination of Q-learning with a deep neural network, has achieved good performance on several games in the Atari 2600 domain. However, the DQN training is very slow and requires too many time steps to converge. In this paper, we use the eligibility traces mechanism and propose the deep Q(λ) network algorithm. The proposed method provides faster learning in comparison with the DQN method. Empirical results on a range of games show that the deep Q(λ) network significantly reduces learning time.

Categories and Subject Descriptors

• **Computing methodologies → Sequential decision making**

General Terms

Algorithms, Performance, Experimentation

Keywords

Reinforcement learning, Deep learning, Temporal difference methods, Q(λ)-learning

1. INTRODUCTION

Reinforcement learning [1, 2] is a suitable framework for sequential decision making problems where an agent makes a sequence of observations of its environment and make decisions based on them. To this end, many reinforcement learning methods have been developed [1, 3]. Two of the most popular and successful temporal difference [4] reinforcement learning algorithms are Q-learning [5] and Sarsa (stands for state, action, reward, state and action) [6]. The methods have been applied to a wide range of problems ranging from control and robotic problems [7] to resource allocation [8] and cloud computing [9]. However many real world problems have very large state spaces and delayed rewards i.e. high dimensional problems with sparse rewards. For these problems, the naïve structure of these methods is not very efficient. If these algorithms do converge, the learning process is slow and requires a large number of time steps.

To deal with high dimensional reinforcement learning tasks and to speed up the learning process, many solutions such as hierarchical reinforcement learning [10, 11] and eligibility traces [3, 4] have been proposed. Eligibility traces are one the most commonly used mechanisms of reinforcement learning. The use of eligibility traces can significantly increase learning speed. In order to obtain this performance increase, a basic temporal difference (TD) method was combined with eligibility traces, called TD (λ) [4, 12] . Combining these methods bridged the gap between TD learning and Monte Carlo methods, thus making it possible to take advantage of the strength of each algorithm. The λ parameter controls after how many steps (e.g. n steps) the backup should be made. In fact, the value of λ for the eligibility traces determines the balance between TD and Monte Carlo methods.

Recent research on deep learning and reinforcement learning have led to introduce a novel method called the deep q-network (DQN) [13, 14] which is a combination of the Q-learning algorithm and convolutional neural networks [15] which are a type of deep neural network. DQN has been tested within the Atari 2600 computer games environment. In many games the DQN's strategy outperformed the human player and achieved state of the art performance on several games with the same network architecture (hyper-parameters). However, applying this method to real world problems, such as robotics, is very challenging. This is because performing a large number of training episodes to collect samples is resource consuming and in many cases not even possible. Other combinations of reinforcement learning and deep neural nets are therefore needed to alleviate this problem.

One of most important extension of the simple Q-learning algorithm (1-step Q-learning) is Q (λ)-learning [5, 16]. Q (λ)-learning combines Q-learning and TD(λ). The Q (λ)-learning algorithm performs significantly better than the naive Q-learning algorithm on a number of tasks [1, 4]. This is due to enhanced performance that eligibility traces mechanism provides i.e. considering a temporary history of a set of transitions such as previously observed states and taken actions.

In this paper, we build on the idea of the eligibility traces, in particular the Q(λ)-learning algorithm. We extend this method to a more general setting by utilizing deep neural networks as a function approximation (similar to the DQN method). This deep neural network is used to estimate Q values in order to speed up the learning process. We propose a new algorithm called Deep Q(λ)-Network (DQ(λ)N). A range of Atari 2600 games will be used as a testbed to evaluate the proposed DQ(λ)N algorithm.

The rest of this article is organized as follows. In Section 2 and 3, we introduce the problem setting and a technical background of reinforcement learning and deep Q-learning, respectively. Then in Section 4, we present DQ(λ)N algorithm and describe how it works. In Section 5, we empirically demonstrate that proposed method performs better than DQN on a range of Atari 2600 games. Finally, in Section 6 we will draw conclusions based on these results.

2. BACKGROUND

The goal of a reinforcement learning (RL) agent is to estimate the optimal policies or the optimal value function of a Markov decision process (MDP) in an unknown environment. If the state and action spaces are finite, then the problem is called a finite MDP. Similar to much of literature that has assumed a finite MDP environment, we also consider finite MDPs.

A RL problem modelled as a Markov decision process is described as follows: The learning agent interacts with the environment, through its sensors, by performing actions and receiving observations and rewards. The interaction is continued until reaching the terminal state or a termination condition is met. A MDP is a five-tuple (S, A, γ, T, R), where S is the set of states in the state space, A is the set of actions in the action space, $0 \leq \gamma \leq 1$ is the discount factor, T is the transition function, which $T(s, \mathrm{a}, s')$ denoting the probability of reaching next state s' from s by taking action a at time step t and R is the reward function with $\mathrm{R}(s, a)$ denoting the expected reward from taking action a in state s at time step t. The aim of the learning agent is to learn an optimal policy π, which defines the probability of selecting action a in state s, so that with following the underlying policy the sum of the discounted rewards is over time maximized. The expected discounted return R_t at time t is defined as follows:

$$R_t = E\{r_t, \gamma r_{t+1}, \gamma^2 r_{t+2} + \cdots\} = E\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k}\right] \quad (1)$$

Where $E[.]$ expectation with respect to the reward distribution and $r_t \in \mathbb{R}$ is a scalar reward obtained at step t. With regard to the transition function and the expected discounted immediate rewards, which are the essential elements for specifying dynamics of a finite MDP, action-value function $Q^{\pi}(s, a)$ is defined as follows.

$$Q^{\pi}(s,a) = E_{\pi}[R_t | \mathrm{s}_t = \mathrm{s}, \mathrm{a}_t = \mathrm{a}] = E_{\pi}[\textstyle\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = \mathrm{s}, a_t = \mathrm{a}] \quad (2)$$

The action-value function $Q_{\pi}(s, a)$ for an agent is the expected return achievable by starting from state s, s ∈ S, and performing action a, $a \in A$ and then following policy π, where π is a mapping from states to actions or distributions over actions.

Due to the recursive property of the the equation (2), the formula can be rewritten as follows:

$$\begin{aligned} Q_{i+1}{}^{\pi}(s,a) &= E_{\pi}\left[r_t + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = \mathrm{s}, a_t = \mathrm{a}\right] \\ &= E_{\pi}[r_t + \gamma Q_i{}^{\pi}(s_{t+1} = s', a_{t+1} = a') | s_t = \mathrm{s}, a_t = \mathrm{a}] \quad (3) \end{aligned}$$

Which is used as the update rule of the estimation of value function at ith iteration.

The optimal policy, π^*, is a policy that maximizes $Q^{\pi}(s, a)$ and as a result, an optimal value function $Q^*(s, a)$. An iterative update for estimating the optimal value function is defined as follows:

$$Q_{i+1}(s,a) = E_{\pi}[r_t + \gamma\, max_{a'} Q_i(s', a') | \mathrm{s}, \mathrm{a}] \quad (4)$$

Where it is implicit that s, s' ∈ S and a, a' ∈ A. The iteration converges to the optimal value function, Q^* as $i \to \infty$ and called value iteration algorithm [1].

A well-known form of temporal difference learning [4] for estimating Q^{π} for a given policy π is the Q-learning algorithm, introduced by Watkins [5]. In many real world tasks, state and action spaces are too large and the use of a table of all $Q(s, a)$ values (Q-table lookup representation) is inefficient. To address this, the function approximation technique is utilized to estimate the value function [17]. Thus, the value function is parameterized $Q(s, a; \theta)$ with parameter vector θ. Usually gradient-descent methods are used to learn parameters by trying to minimize the following loss function of mean-squared error in Q values:

$$L(\theta) = E\left[\left(r + \gamma\, max_{a'} Q(s', a'; \theta) - Q(s, a; \theta)\right)^2\right] \quad (5)$$

Where $r + \gamma\, max_{a'} Q(s', a'; \theta)$ is the target value. Typically, for optimizing the loss function above the stochastic gradient descent method is used. Hence, in the Q-learning algorithm, the parameters are updated as follow:

$$\theta_i = \theta_{i-1} + \alpha\left(y_i - Q(s, a; \theta_i)\right) \frac{\partial Q(s,a;\theta_i)}{\partial \theta_i} \quad (6)$$

Where it is implicit that $y_i = r + \gamma\, max_{a'} Q(s', a'; \theta_{i-1})$ is the target value for iteration i and α is a scalar learning rate.

2.1 Q(λ)-LEARNING

To accelerate the learning process in reinforcement learning tasks, TD(λ) (TD learning with eligibility traces) methods [4] are incorporated into the Q-learning algorithm. This results in Q(λ)-learning method. The eligibility traces consider a temporary history of a set of transitions such as previously observed states and taken actions. In TD(λ) the backup is made after n steps not after every one step. The amount of n is controlled by λ ϵ [0, 1] parameter (e.g. in TD(0), the backup is made after each one step). The eligibility trace of each state-action pair in the process of action-value learning becomes large after visiting the state-action pair and decreases as the state-action pair is not visited. When we use function approximation instead of Q-table lookup to estimate Q values, a trace is considered for each component of the parameter vector θ and there is no single trace corresponding to a state [1]. Thus, TD(λ) updates the vector θ as follows:

$$\theta_i = \theta_{i-1} + \alpha \delta_i e_i \quad (7)$$

Where $\delta_i = y_i - Q(s, a; \theta_i)$ is TD error and $e_i = \gamma \lambda e_{i-1} + \frac{\partial Q(s,a;\theta_i)}{\partial \theta_i}$ is its eligibility value. Note that when $\lambda = 0$, the TD(λ) update is the TD(0) update.

There are two main approaches that combine the eligibility traces and Q-learning (i.e. to Q(λ)). These are different at dealing with exploratory (non-greedy) actions: First is Watkins's Q(λ) [5], where the eligibility traces are set to zero whenever an non-greedy action is taken (i.e. learning is stopped after each non-greedy action selected), and second is Peng's Q(λ) [16], where there is no difference between non-greedy and greedy actions.

3. DEEP Q-LEARNING

A deep Q learning Network (DQN) [13, 14] gets the benefits of deep learning for abstract representation in learning an optimal policy. The DQN algorithm incorporates a deep neural network function approximator with Q-learning and outputs legal action values for a given state. Using model-free reinforcement learning algorithms such as Q-learning algorithm with non-linear function

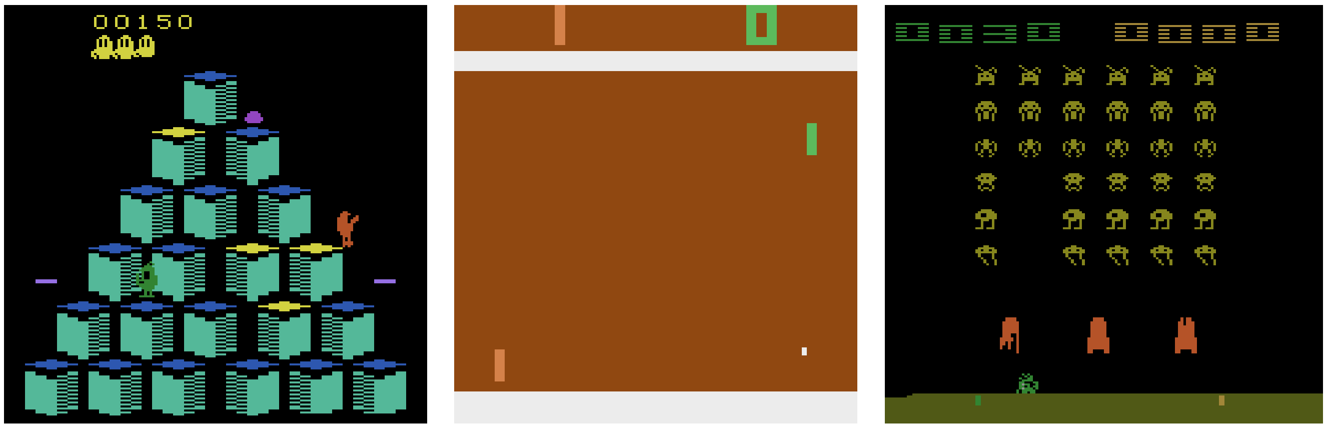


Figure 1: Three frames of 3 Atari 2600 games: Q*bert, Pong and Space Invaders, respectively.

approximators such as neural networks, causes some instability issues and might lead to divergence [18]. The reasons for these issues are as follows: 1) Consecutive states in reinforcement learning tasks have correlation. 2) The underlying policy of the agent is changing frequently, because of slight changes in Q-values. To cope with these problems, the DQN provides some solutions which improve the performance of the algorithm significantly. For the problem of correlated states, DQN uses the previously proposed experience replay approach [19]. In this way, at each time step, the DQN stores the agent's experience (s_t, a_t, r_t, r_{t+1}) into a date set D, where s_t, a_t, and r_t are the state, selected action and received reward, respectively and s_{t+1} is the state at the next time step. To update the network, the DQN utilizes stochastic minibatch updates with uniformly random sampling from the experience replay memory (previous observed transitions) at training time. This neglects strong correlations between consecutive samples. The instability problem of the policy is solved with a target Q-network. The network is trained with the target Q-network to obtain consistent Q-learning targets by keeping the weight parameters (θ^-) used in the Q-learning target fixed and updating them periodically every N steps through the parameters of the main network, θ. The target value of the DQN is represented as follows:

$$y_i = r + \gamma \, max_{a'} Q(s', a'; \theta^-{}_{i-1}) \quad (8)$$

Where θ^- is parameters of the target network.

4. DEEP Q(λ)-LEARNING (DEEP Q(λ) NETWORK)

We consider a naïve type of Watkins's Q(λ)-learning, although there are other variations of Q(λ) such as Peng's Q(λ), to combine with deep learning. The naïve type is similar to Watkins's Q(λ), but the eligibility traces are not set to zero on non-greedy actions. With regard to TD(λ) we propose the following update rule for the vector θ of the proposed algorithm which we refer DQ(λ)N:

$$\theta_i = \theta_{i-1} + \alpha \delta_t e_t \quad (9)$$

$$e_i = \gamma \lambda e_{i-1} + \frac{\partial Q(s, a; \theta_i)}{\partial \theta_i} \quad (10)$$

$$\delta_i = y_i - Q(s, a; \theta_i) \quad (11)$$

Where $y_i = r + \gamma \, max_{a'} Q(s', a'; \theta^-{}_{i-1})$ is the target value, which is the same as for DQN.

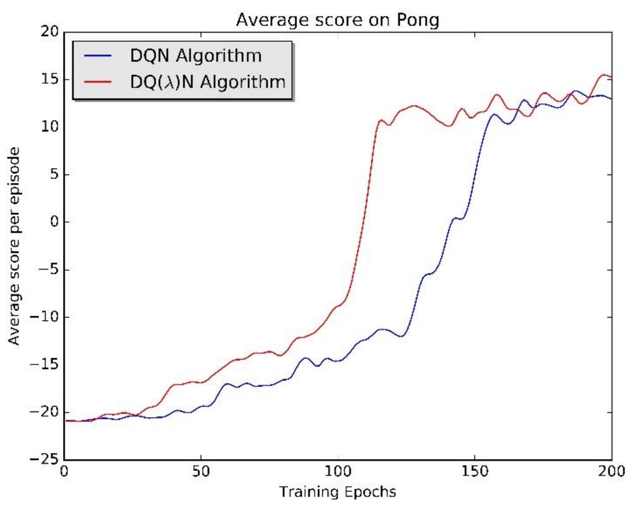
Comparing the above equations with equation (7), outlines the key difference between the DQ(λ)N and the DQN approach. These two approaches are similar as they both calculate the target value using a target network with the weights $\theta^-{}_{i-1}$. The target network is updated based on the main network periodically. To prevent divergence in parameters an experience replay mechanism [19] is applied [14].

Algorithm 1 summarizes the proposed deep Q(λ)-learning method, where the vector e contains the trace vector for each component of the parameter vector θ, corresponding to the eligibility traces [1]. For λ = 0, the algorithm is DQ(0)N that is the same as the DQN.

```
Algorithm 1: Deep Q(λ)-learning
initialize $\theta$ with random values
initialize replay memory $\boldsymbol{M}$ with capacity N
for each episode repeat:
    initialize $e = 0$
    initialize $\boldsymbol{s}$
    for each step in the episode repeat:
        choose action $\boldsymbol{a}$ according to $\boldsymbol{\varepsilon}$-greedy policy
        take action $\boldsymbol{a}$, observe reward $\boldsymbol{r}$ and next state $\boldsymbol{s'}$
        store transition $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{s'})$ in $\boldsymbol{M}$
        $\boldsymbol{s} \leftarrow \boldsymbol{s'}$
        $\boldsymbol{b} \leftarrow$ sample a sequence of transitions from the replay memory, $\boldsymbol{M}$
        if $\boldsymbol{s_b}$ (as the last state in the sequence) == terminal:
            $\boldsymbol{y} \leftarrow \boldsymbol{0}$
        else:
            $\boldsymbol{y} \leftarrow \boldsymbol{max_a Q(s_b, a; \theta^-)}$
        for each transition $(\boldsymbol{s_j}, \boldsymbol{a_j}, \boldsymbol{r_j}, \boldsymbol{s'_j})$ in reverse(b) repeat:
            $\boldsymbol{y} \leftarrow \boldsymbol{r_j + \gamma y}$
            $\boldsymbol{e} \leftarrow \boldsymbol{\gamma \lambda e} + \frac{\boldsymbol{\partial Q(s_j, a_j; \theta)}}{\boldsymbol{\partial \theta}}$
            $\boldsymbol{\delta} \leftarrow \boldsymbol{y - Q(s_j, a_j; \theta)}$
            $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta + \alpha \delta e}$
    until s is terminal
```

5. EMPIRICAL RESULTS

In this section, we present the performance results of the DQ(λ)N algorithm and show how it performs better than the DQN in terms of its rate of learning. The proposed method was evaluated on 3 Atari 2600 games in the Arcade Learning Environment (ALE) [20]. The ALE presents an environment that emulates the Atari 2600 games. It provides a very challenging environment for reinforcement learning that has high dimensional visual input which is partially observable. It presents a range of interesting



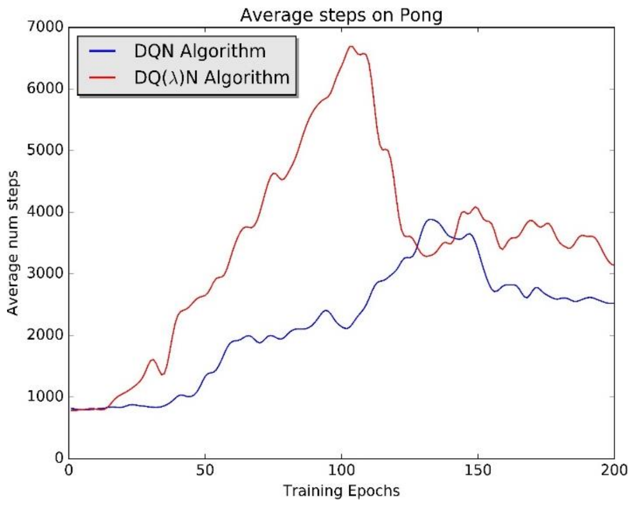
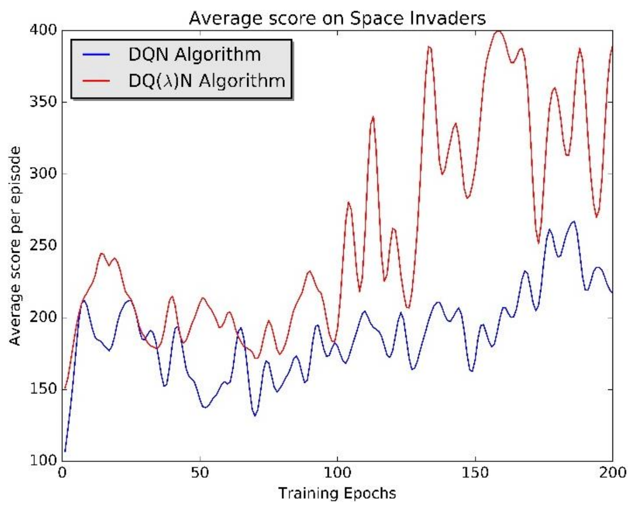


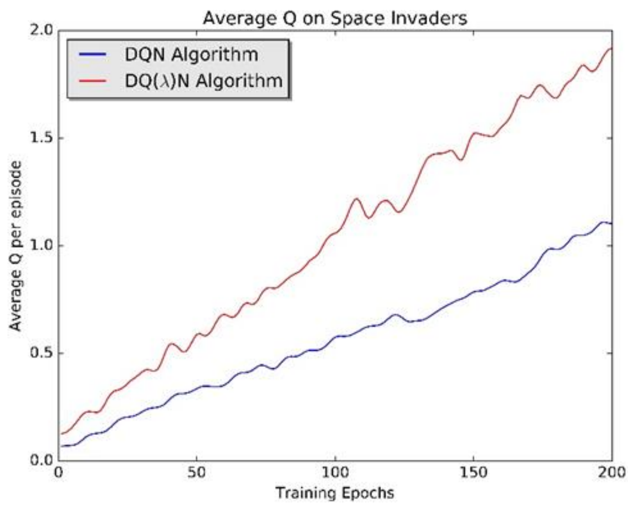
Figure 2: A comparison of performance of average score and steps per episode of the proposed algorithm with λ = 0.7 and the DQN on the game Pong. One epoch corresponds to 10 episodes and each score is an average of running an ϵ-greedy policy, with ϵ = 0.05 for 5 episosdes.

games that new methods can be tested. For our experiments we selected 3 Atari games: Q*bert ,Pong and Space Invaders, as shown in Figure 1. The goal of a RL algorithm is to learn a specific optimal policy to play each of the games just by using raw pixels frames as input.

The network architecture that we used is similar to Mnih et al [14]. It contains three hidden fully convolutional layers [21] and a fully-connected hidden layer. The output layer is a fully-connected linear layer with a number of output neurons corresponding to each action in the game. The network computes Q values of the individual action of the input state, where each state is a stack of 4 frames recently observed by the agent (to see more details refer to [14]).

Evaluation of learned policies by the agent was conducted every 10 episodes by running an ϵ-greedy policy with ε = 0.05 for 5 episodes and averaging the resulting scores and steps. The networks were trained for 200 epochs (each epoch 10 episode considered) and the size of the replay memory was 500, 000. All weights of the networks were updated by the RMSProp optimizer [22] with a learning rate of α = 0.00025 and a momentum of 0.95. The target network was updated after each 10000 steps. Training for all the games was done without changing in the network architecture and all hyper-parameters settings. The rest of settings were the same as those utilized in [14].





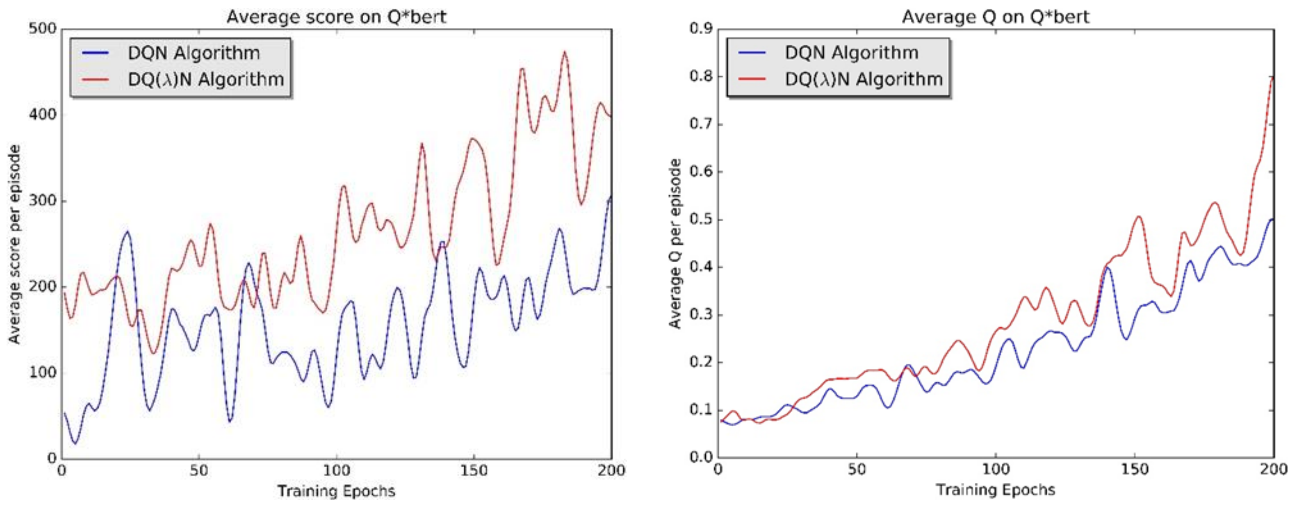


Figure 3: Th first column shows a comparison of performance of average score per episode of the proposed algorithm and the DQN on Q*bert and Space Invaders games respectively. For each game, one epoch corresponds to 10 episodes and each score is an average of running an ϵ-greedy policy, with $\epsilon = 0.05$ for 5 episosdes.The secound column shows the average of the predicted Q per episode for the DQ(λ)N with λ = 0.7 and the DQN when the agent select greedy actions during training process on Q*bert and Space invaders games, respectively.

5.1 Results

In order to validate our approach we compare it with the deep Q network. The results presented in Figures 2 and 3 show the performance results of the proposed DQ(λ)N algorithm and the DQN. The graphs present the average total reward and steps collected by the agent, also the average of the predicted Q during training phase on 3 games: Pong, Q*bert and Space Invaders. As expected, our results demonstrate the accelerated learning provided by the DQ(λ)N. The left plots in Figure 2 and Figure 3 show the faster convergence of DQ(λ)N compared to the DQN. This is particularly evident in the Pong game (Figure 2), where we can see that the learning rate of DQ(λ)N is clearly better. In this case, it was revealed that the proposed method could reach the optimal average score approximately 1.5 times faster than the DQN and significantly better (paired t-test, $p < 0.05$) average scores were obtained during training period. The second metric that we consider is the average total steps needed per episode by an agent during training. The right plot of Figure 2 shows the average number of steps taken by DQ(λ)N agent increases in early epochs but then decreases to a similar number of steps as the DQN. It is evident that the DQ(λ)N takes more steps than the DQN. This may appear to be a negative feature of the proposed DQ(λ)N as a lower number of steps is desirable. On the contrary, we argue that this reveals a key advantage feature of our method. The left plot of Figure 2 demonstrates that the algorithm is consistently progressing, in this case in the Pong game. Having a higher number of steps initially in comparison to the DQN, indicates that the DQ(λ)N has learned faster and tries to hit the ball to avoid of getting negative reward. This is why more steps are required initially for DQ(λ)N.

As described by Mnih et al. [13] another metric for evaluating a reinforcement learning agent is the policy's estimated Q value, which computes the received discounted reward while the agent follows a certain policy. The right plots of Figure 3 illustrate that the average predicted Q value by our method increases over time at a faster rate than that of the DQN. This reflects that the model is learning gradually in stable manner that is also significantly faster when compared to the DQN algorithm.

To further analysis, a paired t-test was conducted to compare the received average total reward in the proposed method and the DQN on three Atari 2600 games: Pong, Q*bert and Space Invaders. As shown in Table 1, the DQ(λ)N gave significantly higher ($p < 0.05$) average reward for each game. These results suggest that the proposed method can achieve more scores in the early stage of learning and as a result speeding up in learning process.

Table 1: Paired t-test results comparing the DQ(λ)N and the DQN algorithms on the received average total reward.

Game	*Number of epochs*	*t-statistic*	*p-value*
Pong	100	-10.064	0.000
Q*bert	100	-2.696	0.008
Space Invaders	100	-2.967	0.003

6. CONCLUSION

This paper proposed the combination of TD (λ) learning, in particular Q(λ)-learning and a deep neural network. We extended the DQN algorithm to take into account eligibility traces. This novel combination, Deep Q(λ) Network (DQ(λ)N), allowed us to take advantage of the DQN algorithm and the eligibility trace mechanism in order to further accelerate the learning process. The proposed method was compared to the DQN method by testing the algorithm on 3 Atari 2600 games. Empirical results confirm that DQ(λ)N can learn the satisfactory control policies in fewer number of trials (i.e. speeding up the learning process) in comparison to DQN. This was observed for all 3 games. In this work, we investigated one TD(λ) learning algorithm, the naïve form of Q(λ) learning. A natural direction for future work would be to incorporate the other variations of TD(λ) [5, 6, 16, 23] or the least square based methods with the possibility of eligibility trace mechanism [24] such as the least-squares temporal difference (LSTD(λ)) [25], the least-squares policy evaluation (LSPE(λ)) [26, 27], etc. into deep neural networks and establish which method performs best.

7. REFERENCES

[1] R. S. Sutton, and A. G. Barto, Introduction to Reinforcement Learning: MIT Press, 1998.

[2] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” Journal of artificial intelligence research, vol. 4, pp. 237-285, 1996.

[3] A. G. Barto, R. S. Sutton, and C. W. Anderson, “Neuronlike adaptive elements that can solve difficult learning control problems,” IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-13, no. 5, pp. 834-846, 1983.

[4] R. S. Sutton, “Learning to Predict by the Methods of Temporal Differences,” Machine Learning, vol. 3, no. 1, pp. 9-44, 1988.

[5] C. J. C. H. Watkins, “Learning from Delayed Rewards,” King's College, Cambridge, Cambridge, UK, 1989.

[6] G. A. Rummery, and M. Niranjan, On-Line Q-Learning Using Connectionist Systems, 1994.

[7] J. Kober, J. A. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” The International Journal of Robotics Research, vol. 32, no. 11, pp. 1238-1274, September 1, 2013, 2013.

[8] D. Vengerov, “A reinforcement learning approach to dynamic resource allocation,” Engineering Applications of Artificial Intelligence, vol. 20, no. 3, pp. 383-390, 2007.

[9] M. Duggan, J. Duggan, E. Howley, and E. Barrett, “An Autonomous Network Aware VM Migration Strategy in Cloud Data Centres.”

[10] A. G. Barto, and S. Mahadevan, “Recent Advances in Hierarchical Reinforcement Learning,” Discrete Event Dynamic Systems, vol. 13, no. 4, pp. 341-379, 2003.

[11] S. S. Mousavi, B. Ghazanfari, N. Mozayani, and M. R. Jahed-Motlagh, “Automatic abstraction controller in reinforcement learning agent via automata,” Applied Soft Computing, vol. 25, pp. 118-128, 12//, 2014.

[12] G. Tesauro, "Practical issues in temporal difference learning," Reinforcement Learning, pp. 33-53: Springer, 1992.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing Atari With Deep Reinforcement Learning,” vol. NIPS Deep Learning Workshop, 2013.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529-533, 02/26/print, 2015.

[15] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation Applied to Handwritten Zip Code Recognition,” Neural Computation, vol. 1, no. 4, pp. 541-551, 1989.

[16] J. Peng, and R. J. Williams, “Incremental multi-step Q-learning,” Machine Learning, vol. 22, no. 1-3, pp. 283-290, 1996.

[17] R. S. Sutton, A. M. David, P. S. Satinder, and Y. Mansour, “Policy Gradient Methods for Reinforcement Learning with Function Approximation,” pp. 1057--1063, 2000.

[18] J. N. Tsitsiklis, and B. Van Roy, “An analysis of temporal-difference learning with function approximation,” IEEE transactions on automatic control, vol. 42, no. 5, pp. 674-690, 1997.

[19] L.-J. Lin, “Reinforcement learning for robots using neural networks,” Carnegie Mellon University, 1993.

[20] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, “The arcade learning environment: an evaluation platform for general agents,” J. Artif. Int. Res., vol. 47, no. 1, pp. 253-279, 2013.

[21] Y. LeCun, and Y. Bengio, "Convolutional networks for images, speech, and time series," The handbook of brain theory and neural networks, A. A. Michael, ed., pp. 255-258: MIT Press, 1998.

[22] T. Tieleman, and G. Hinton, Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning, Technical report, 2012.

[23] H. van Seijen, and R. S. Sutton, "True Online TD (lambda)." pp. 692-700.

[24] S. J. Bradtke, and A. G. Barto, “Linear least-squares algorithms for temporal difference learning,” *Machine Learning*, vol. 22, no. 1-3, pp. 33-57, 1996.

[25] J. A. Boyan, "Least-squares temporal difference learning." pp. 49-56.

[26] D. P. Bertsekas, and S. Ioffe, “Temporal differences-based policy iteration and applications in neuro-dynamic programming,” *Lab. for Info. and Decision Systems Report LIDS-P-2349, MIT, Cambridge, MA*, 1996.

[27] H. Yu, "Convergence of least squares temporal difference methods under general conditions." pp. 1207-1214.