NPV-DQN: Improving Value-based Reinforcement Learning, by Variable Discount Factor, with Control Applications

Gabor Paczolay, Istvan Harmati

Department of Control Engineering, Budapest University of Technology and Economics, Magyar tudósok krt. 2, I building, H-1117 Budapest, Hungary, paczolay@iit.bme.hu; harmati@iit.bme.hu

Abstract: Discount factor plays an important role in reinforcement learning algorithms. It decides how much future rewards are valued for the present time-step. In this paper, a system with a Q value estimation, based on two distinct discount factors are utilized. These estimations can later be merged into one network, to make the computations more efficient. The decision of which network to use, is based on the relative value of the maximum value of the short-term network, the more unambiguous the maximum is, the more probability is rendered to the selection of that network. The system is then benchmarked, on a cartpole and a gridworld environment.

Keywords: reinforcement learning; DQN; NPV; NPV-DQN

1 Introduction

Reinforcement learning is becoming a highly researched field in the domain of artificial intelligence, as more and more problems can be solved using it, due to increases of available computing power.

In this paper, an economical, theory-based idea, is visited utilizing discounted rewards. The discounted rewards are considered to be similar to the Net Present Value, with a steeper discount meaning a more risk-evading policy. Two networks with different discount factors are applied, where the short-term network is exploited when the long-term system is indecisive, to ensure that the system gets rewards easier. This method can speed up or even improve learning, as it will be seen from the results herein.

The first mentionable and the most well-known reinforcement algorithm was invented by Watkins [23]. However, in those days, reinforcement learning algorithms did not see a lot of real-world usage. This abundance was due to the requirement of the state space table, which required that all state-action value pairs had to be stored in memory, leading to an explosion of memory usage when the method is used for problems with large state spaces. This problem was remedied by the utilization of artificial neural networks: Mnih et al. created the DQN algorithm and have beaten human players in Atari games [10]. As convergence was a problem in previous, not well-performing trials of neural networks, this system also has some tricks to ensure convergence. First, to process motion, it used a couple of consequential frames together, it utilized an experience replay buffer to hold previous memories, and it utilized a target network. Van Hasselt et al. created the Double Q-learning algorithm, where two value functions are utilized, and each experience updates one of the value estimators, and the estimators then update the another one [21]. Wang et al. introduced the Dueling DQN architecture, where one estimator approximated the value of the state, while the other estimated the action advantage [22]. These two pieces of information were then aggregated to receive the Q values. Schaul et al. modified the experience replay buffer to hold also loss values, and the system prioritized the samples with higher loss values [19]. Bellemare et al. used the value approximation distribution as an output, and while only using the expected values gained from the distributions, it vastly outperformed the original DQN algorithm [1]. While the previous method had the disadvantage of giving a lower and upper bound to the processable Q values, this was corrected by Dabney at al. with Quantile Regression [2]. Hessel et al. combined many of the previously mentioned researches, leading to even better results, the name of this new version is called Rainbow-DQN [6]. Lavet et al. progressively increased the discount factor to reduce learning steps [3].

The full state is not always possible to observe, for example due to the imperfection of the sensors. In this case, the Markov Decision Process becomes a Partially Observable Markov Decision Process, and the solution of these systems requires memory. Hausknecht et al. created a solution to this problem by utilizing a Long Short-Term Memory (LSTM) on the inputs [5]. Li et al. merged Supervised Learning with Reinforcement Learning, with the former containing recurrent components, to deal with POMDPs [8].

Research on control can also be mentioned regarding the research as the proposed algorithm is also intended to solve control problems. Reda et al. designed a hybrid supervised- and reinforcement learning algorithm for vehicle automation [17]. Unguritu et al. modeled an anti-lock braking system mathematically and then designed a controller for that [20]. Zamfirache et al. combined an actor-critic algorithm with the Grey Wolf Optimizer algorithm, to eliminate the main drawbacks of the gradient descent algorithm [24]. Preitl et al. applied Iterative Feedback Tuning algorithms in the design of fuzzy control systems [16]. Haidegger et al. modeled a telesurgery system and suggested several control options for that [4]. Precup et al. presented a new framework for 2DOF controllers for integral processes of servo systems [15]. Németh created an enhanced cruise control system with Linear Parameter-Varying control and an optimization-based supervisor [11].

The algorithm that we propose relies on having two action-value functions with different discount factors to introduce a conditional greediness to the DQN algorithm. This modification made it possible to improve upon the original algorithm, making our proposed algorithm favorable. In which our system is different from other systems is the economical background of the discount factor and the switching algorithm.

The novelty of this work, is a double discount factor, with a specific action selector. The algorithm can be utilized for any reinforcement learning tasks that use discrete actions, such as control tasks, like high-level control of Unmanned Aerial Vehicles (UAV) or quantized exploration or patrol tasks, where it has the advantage with respect to non-learning control tasks that the a priori environment knowledge is not required. Otherwise, reinforcement learning is now widely utilized for Large Language Model (LLM) fine-tuning.

In this paper, first the theoretical background is mentioned. Then, the theory of the proposed idea is mentioned, followed by the experiments conducted. In the end, the results are discussed and the conclusions of the research are made with the future work to be conducted later.

2 Theoretical Background

2.1 Markov Decision Processes

Markov Decision process is a discrete-time framework for decision making optimization problems, also applicable for reinforcement learning. Figure 1 shows the basics of this framework [12]. In an MDP the following elements are defined: states that the environment can hold, actions the agent can choose, transition probabilities describing which the agent will be in when selecting a specific action in a specific state, and rewards to be optimized [7]. At each timestep, the agent starts at a specific state s, selects an action a from the available action space, and based on the former two and the transition probability P, it gets into a new state s' and receives a reward r. The Markov property is defined on the stochastic process if the following holds:

$$P(\boldsymbol{a^t} = \boldsymbol{a}|\boldsymbol{s^t}, \boldsymbol{a^{t-1}}, \dots, \boldsymbol{s^0}, \boldsymbol{a^0}) = P(\boldsymbol{a^t} = \boldsymbol{a}|\boldsymbol{s^t}) \quad (1)$$

This means that P, the probability of the transitions, only depend on the last state and the currently selected action, thus only these two are important in the decision of the following state [14]. This only holds for the traversal of the MDP, during training, other samples can also be used.

Policies are a mapping of each state $\boldsymbol{s}$, to the actions $\boldsymbol{a}$, and they are very important in Markov Decision Processes, as agents are trying to find the optimal policy. This optimality means that it minimalizes the sum of the discounted expected rewards, or the return. This discount means that agents tend to prefer a short-term reward to the longer-term ones, and a coefficient decides the discount product between each step. A solution of the policy would mean one that maximizes the reward reachable by the agent, and to find one, one has to find a fixed point of the Bellman equation via iterative search. The Bellman equation looks like the following:

$$v(\boldsymbol{s},\pi^*) = max_a(r(\boldsymbol{s},\boldsymbol{a}) + \gamma \textstyle\sum_{s'} \boldsymbol{p}(\boldsymbol{s'}|\boldsymbol{s},\boldsymbol{a})v(\boldsymbol{s'},\pi^*)) \quad (2)$$

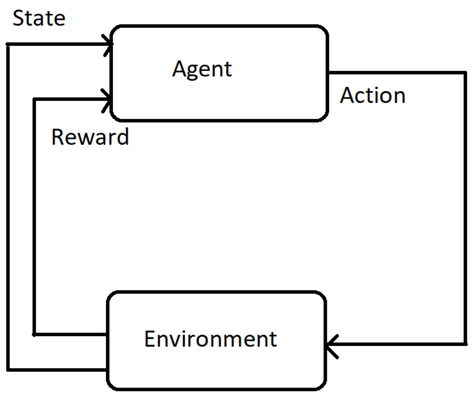

Figure 1
Markov Decision Process

In this equation. $r(s,a)$ is the reward rendered to the appropriate state-action pair, γ is the discount factor favoring short-term rewards, and $p(s'|s,a)$ is the transition probability function. It can be seen that if the agent knows P and r, thus it is familiar enough with the environment, it can find the optimal values.

2.2 Reinforcement Learning

Reinforcement learning is one of the three subtypes of machine learning, next to supervised and unsupervised learning. It is one subtype of Markov Decision Process when the state transition probabilities or the payoff function is not known, thus, the Bellman optimality equation is not enough for a solution. In the reinforcement learning case, the agent interacts with the environment to get more familiar with it, it selects specific (random or policy-based) actions and receives appropriate rewards.

There are two main kinds of reinforcement learning: one is called value-based and the other is policy-based.

In the case of value-based reinforcement learning, all states or state-action pairs are receiving a value. This value is rendered to the expected return from the state or by selecting a specific action in a state. The well-known value-based reinforcement learning method is Q-learning [23]. This algorithm renders values, the so-called Q-values to all state-action pairs, which is, in fact, the expected return by selecting the action in the state. The Q-value update works by the following equation:

$$Q(\boldsymbol{s'},\boldsymbol{a}) \leftarrow (1-\alpha)\cdot Q(\boldsymbol{s},\boldsymbol{a}) + \alpha\cdot\left(r+\gamma\cdot \max_{a'} Q(\boldsymbol{s'},\boldsymbol{a'})\right) \tag{3}$$

where α is the learning rate and γ is the discount for the reward. The policy of the algorithm is to always select the action in a state that has the highest Q-value for the state-action pair.

In policy-based reinforcement learning, the policy is a parametrized function, which has the states as inputs and the actions, or the distribution of the actions as output. The most utilized policy-based method is policy gradient, where a set of θ parameters characterize a policy π_θ, and the agent seeks to obtain the maximal expected reward for a specific trajectory, marked here by r(τ). We obtain the following payoff function:

$$J(\boldsymbol{\theta}) = E_{\pi_\theta}[r(\tau)] \tag{4}$$

The process of tuning the parameters is performed with respect to the gradient of the payoff function:

$$\boldsymbol{\theta}_{k+1} = \theta_t + \alpha\Delta J(\boldsymbol{\theta_t}) \tag{5}$$

It is worth mentioning the advantages and disadvantages of policy-based reinforcement learning. They have smaller bias but higher variance compared to value-based algorithms. Due to this reason, policy-gradient methods find local optima better, but they find it harder to arrive in global optima. Due to the parametrization, they can also map action spaces that have action spaces that have high dimension count or are even continuous, and stochasticity is better handled.

2.3 Deep Reinforcement Learning

Whenever the decision making of a reinforcement learning agent is aided by a neural network, the problem domain becomes a deep reinforcement learning problem [13].

Neural networks are function approximators that consist of biologically-inspired building blocks, the neurons. They can be described by the following equation:

$$y = Act\left(\sum \boldsymbol{wx} + \boldsymbol{b}\right) \tag{6}$$

In this equation, x is the input vector, w contains the weights of the neural network, and the dot product is taken from the aforementioned two. b is the bias scalar, which ensures that the result of the block can be different from zero in the case of having zero as an input. The bias element can also be described as a weight that is connected to the constant one scalar. The Act function is the activation function, that introduces nonlinearity to the system, without this, the system would only be able to solve linear problems. The selection of the activation function strongly depends on the problem type, the depth of the network and the precision-computing efficiency tradeoff. The w and b variables are trained by backpropagation, where the partial derivative errors of the inputs are calculated starting by a forward-propagation to get the final error, then backpropagating the errors through the layers up until the input vector.

The difference between traditional and deep reinforcement learning systems should also be mentioned. They have the advantage of the omission of the state space table, thus environments of huge, continuous or image-based action spaces can all be solved with them. However, deep reinforcement learning algorithms have worse convergence than traditional learning algorithms due to the fact that the output values are only approximations. This disadvantage is being solved by several tricks that makes these systems converge.

2.4 DQN Algorithm

The architecture of the traditional Q-learning method is not suitable for neural systems, due to the fact that all Q-value estimations need a forward pass on the neural network, making the system slow. This disadvantage is cured by the architecture of Deep Q-Network, where the output dimension is equal to the number of the available actions (making it only suitable for systems with discrete action spaces), and they each represent a Q-value estimation for the input state, making the system calculate all Q-values of the state in one forward inference. The pseudocode for the algorithm can be seen in Algorithm 1.

As mentioned during the theoretical background introduction, deep reinforcement learning methods do not converge as easily as traditional ones, and several improvements has to be made for the algorithms to ensure convergence. One of these improvements is called the experience replay, which is a buffer that holds the last batch of tuples of states, selected actions, received rewards, the terminal state Booleans and the 'next states', the states where the agent have transitioned after taking the action in the previous state, marked as (***s, a, r, d, s'***). During training, which can take place after either a specific number of episode steps of after an episode has ended, a batch of these tuples are sampled from the buffer, and this batch is used for the training. With this technique, earlier memories also appear in the training samples, making the samples more diverse and making the algorithm converge with higher probability.

Another trick to make the system converge is called target model. During the learning phase, this is utilized for inference of the next states, and this model is similar to the original model but is updated with less frequency (called hard update) or with smaller steps (called soft update).

The algorithm works in the following way: first, the replay memory and the models are initialized. Then, the algorithm receives a state, for which it selects the action with the maximal Q-value of the state with an ϵ-greedy strategy. After taking that action and observing the reward and the next state, the corresponding elements are saved to the replay buffer. During training, a batch is sampled from the replay buffer, and the target Q-value is calculated based on (7), for which a gradient descent step is done. The target model is updated every C steps.

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\boldsymbol{\theta}$
Initialize target model $\hat{Q}$ with weights $\boldsymbol{\theta}^- = \boldsymbol{\theta}$
for episode = 1, $M$ do
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\Phi_1 = \Phi(s_1)$
    for $t$ = 1, T do
        With probability $\epsilon$ select a random action $a_t$
        otherwise select $\boldsymbol{a_t} = argmax_a Q(\Phi(s_t), \boldsymbol{a}; \boldsymbol{\theta})$
        Execute action $\boldsymbol{a_t}$ and observe reward $r_t$ and image $\boldsymbol{x_{t+1}}$
        Set $\boldsymbol{s_{t+1}} = \boldsymbol{s_t}, \boldsymbol{a_t}, \boldsymbol{x_{t+1}}$ and preprocess $\Phi_{t+1} = \Phi(\boldsymbol{s_{t+1}})$
        Store transition $(\Phi_t, \boldsymbol{a_t}, r_t, \Phi_{t+1})$ in D
    end for
    Sample random minibatch of transitions $(\Phi_j, \boldsymbol{a_j}, r_j, \Phi_{j+1})$ from $D$
    Set $y = \begin{cases} r_j & \text{if episode terminates at step j + 1} \\ r_j + \gamma \max_{d} \hat{Q}(\Phi_{j+1}, \boldsymbol{a'}; \boldsymbol{\theta}^-) & \text{otherwise,} \end{cases}$    (1)
    Perform a gradient descent step on $(y_j - Q(\Phi_j, \boldsymbol{a_j}; \theta))^2$ with respect to the network parameters $\boldsymbol{\theta}$
    Every C steps reset target model $\hat{Q} = Q$
end for
```

Algorithm 1
DQN

3 The Proposed NPV-DQN Algorithm

The idea for the proposed method of the paper comes from an economical background. In economy, the Net Present Value gives the value of an investment discounted to the money amount it would be valued today. The formula looks like the following:

$$NPV = \sum_{i=0}^{\infty} \frac{F_i}{(1+r_i)^i} \tag{8}$$

where F_i is the expense or return for the i-th time segment, and r_i is the capital cost, which generally means the yearly cost (usually given in a percentage) of investing it in the investment given in the calculation instead of investing it in the stock market (as an alternative that is always available) with risk equal to the risk of the aforementioned investment. Higher risk means higher capital cost; thus, the discount factor will be greater, leading to smaller values in the NPV. Thus, higher risk favors investments that have a high return in the first-time segments.

The reason of the economical introduction in this section is due to the similarities with the discount factor utilized in the return of the value calculation for reinforcement learning. A state value is given by the following formula:

$$V_i = \sum_i \gamma^i R_i \tag{9}$$

where γ is the discount factor, and R_i is the reward for the i th step.

The proposed idea is based on the fact that when the values of the long-term outcomes of a state based on the actions are undecided (meaning that there are multiple highest action-values) or close to each other, but the short-term outcomes are clearer, greediness can be introduced. This means that the agent favors actions that yield higher return on the short-term time span instead of the uncertain long-term one.

The idea can be realized by creating two similar action-value functions, and training them similarly (even on the same batch) but utilizing different γ discount factors during training. This will mean that the system will have a Q action-value function with higher γ, and a new Q' with a lower γ' discount rate. Based on the simulations, the best rate of γ' to choose is 0.7 to 0.8. Then, the system selects an action from either Q or Q', where the probability of selecting from Q' is equal to:

$$P = \left(1 - \frac{ratio(Q)}{ratio(Q) + ratio(Q\prime)}\right)^k \tag{10}$$

where k is a variable positive coefficient (taken to be 2 in the simulations) and ratio is given by the following formula (M is just a function argument):

$$ratio(M) = \frac{second(M) - min(M)}{max(M) - min(M)} \tag{11}$$

where second yields the second highest value in a list (in this case, the second highest action value).

As the usage and the training takes place at the same time elements of the system, the two action-value functions can be realized even with one network, saving the time of a second inference. Note that due to benchmarking reasons this was not done in the simulations of the paper, as in that case, the neuron count would not match the original DQN algorithm.

Finally, let us look at the proposition that in some cases, the system with a lower-valued discount rate converges faster than higher-valued one. First, define the contraction operator **H** for $q: \mathcal{X} \times \mathcal{A} \longrightarrow \mathbb{R}$, as in [9] and [18]:

$$(Hq)(\boldsymbol{x}, \boldsymbol{a}) = \sum_{y \in \mathcal{X}} P_\alpha(x, y)[r(\boldsymbol{x}, \boldsymbol{a}, \boldsymbol{y}) + \gamma \max_{b \in \mathcal{A}} q(\boldsymbol{y}, \boldsymbol{b}) \tag{12}$$

And **H**’ as **H** in (12), just γ' instead of γ.

Proposition 1. If $0 < \gamma' < \gamma \leq 1$, and the reward function is greater or equal to 0, then the variance of the system with γ' is smaller that the one with γ , that is,

$$var[F'_t(\boldsymbol{x})|\mathcal{F}] \leq var[F_t(\boldsymbol{x})|\mathcal{F}] \tag{13}$$

P*roof.*

$var[F_t(\boldsymbol{x})|\mathcal{F}] =$

$= \mathbb{E}[(r(\boldsymbol{x},\boldsymbol{a},X(\boldsymbol{x}.\boldsymbol{a})) + \gamma \max_{b\in\mathcal{A}} Q_t(\boldsymbol{y},\boldsymbol{b}) - Q^*(\boldsymbol{x},\boldsymbol{a}) - HQ_t(\boldsymbol{x},\boldsymbol{a}) + Q^*(\boldsymbol{x},\boldsymbol{a}))^2] =$

$= \mathbb{E}[((r(\boldsymbol{x},\boldsymbol{a},X(\boldsymbol{x},\boldsymbol{a})) + \gamma \max_{b\in\mathcal{A}} Q_t(\boldsymbol{x},\boldsymbol{b}) - HQ_t(\boldsymbol{x},\boldsymbol{a}))^2] =$

$= var[r(\boldsymbol{x},\boldsymbol{a},X(\boldsymbol{x},\boldsymbol{a})) + \gamma \max_{b\in\mathcal{A}} Q_t(\boldsymbol{x},\boldsymbol{b})|\mathcal{F}_t]$

Similarly,

$var[F'_t(x)|\mathcal{F}] = var[r(\boldsymbol{x},\boldsymbol{a},X(\boldsymbol{x},\boldsymbol{a})) + \gamma' \max_{b\in\mathcal{A}} Q_t(\boldsymbol{x},\boldsymbol{b})|\mathcal{F}_t]$

The optimal value function is like the following:

$$V^*(x) = \max_{\mathcal{A}_t} \mathbb{E}[\textstyle\sum_{t=0}^{\infty} \gamma^t (R(X_t,A_t))|X_0 = x] \tag{2}$$

Due to the limitation we put in that limits the reward function to be $r \geq 0$, or in (14), $R(X_t,A_t) \geq 0$, we get that the value function $V^*(x) \geq 0$

As the optimal Q-function is defined as the following:

$Q^*(\boldsymbol{x},\boldsymbol{a}) = \sum_{y\in X} P_\alpha(\boldsymbol{x},\boldsymbol{y})[r(\boldsymbol{x},\boldsymbol{a},\boldsymbol{y}) + \gamma(V^*(y)]$

and P is a probability, it can be seen that $Q^*(x,a) \geq 0$.

It can be concluded that if arrays $Y \geq 0$ and $Z \geq 0$ and the scalar $0 \leq \beta \leq 1$, then

$$var[Y + \beta Z] \leq var[Y + Z] \tag{3}$$

and from that:

$var\left[r\big(\boldsymbol{x},\boldsymbol{a},X(\boldsymbol{x},\boldsymbol{a})\big) + \gamma' \max_{b\in\mathcal{A}} \mathrm{Q_t}(\boldsymbol{x},\boldsymbol{b})\middle|\mathcal{F}_t\right] \leq$

$$\leq var[r(\boldsymbol{x},\boldsymbol{a},X(\boldsymbol{x},\boldsymbol{a})) + \gamma \max_{b\in\mathcal{A}} Q_t(\boldsymbol{x},\boldsymbol{b})|\mathcal{F}_t] \tag{4}$$

which means that our initial statement is true, so:

$$var[F'_t(x)|\mathcal{F}] \leq var[F_t(x)|\mathcal{F}] \tag{5}$$

□

Proposition 2. If $0 < \gamma' < \gamma \leq 1$, and the reward function is less or equal to 0, then (13) holds.

Proof. In this case, the reward function and the optimal Q-function are both less or equal to 0. The variance of the negated array is equal to the variance of the original array, so:

$$var[-Y - \beta Z] = var[Y + \beta Z] \leq var[Y + Z] = var[-Y - Z] \tag{18}$$

and from this:

$$var[F'_t(x)|\mathcal{F}] \leq var[F_t(x)|\mathcal{F}] \tag{19}$$

holds.

4 Tests and Results

To check the usability in different conditions, the system was benchmarked on two distinct testbeds. The pseudocode of the proposed algorithm can be seen in Algorithm 2. The computational complexity of the algorithm, just like for regular neural networks, is $O(n^4)$, where n is the number of neurons of the network.

The stability of the proposed algorithm is the same as the stability of DQN, which is stable but not in all situations. The optimality is also the same as for DQN algorithms, which are converging to an optimal solution.

The first testbed was a cartpole environment with discrete action space. Its observation space holds four elements: a position and a velocity element of the cart, and the angle and angular velocity of the pole. The action space has two elements, moving the cart to the two ways along the x axis. The reward is +1 for each step when the pole is in an acceptable region (such that it does not fall), which is when the pole angle is smaller than ±12^∘ and the cart position did not leave the ±2.4 region. Given this, the goal is to maximize the time of the episodes. The termination step is the 500th step of the episode. Figure 2 shows the cartpole experiment.

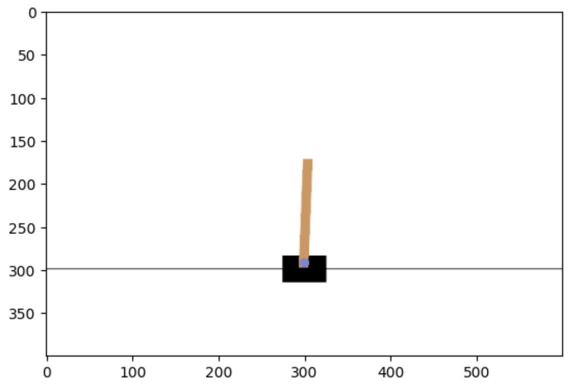

Figure 2
The cartpole environment

The second testbed was a gridworld environment. In this setting, the agent always starts at the [0,0] position, and the goal's position is at [±3,±3] (so, it can also be 4 places), starting at [+3,+3]. Then, if reached, it moves clockwise to the next position. The reward is the negative of the Euclidean norm between the agent and the current goal, that is $-\sqrt{\left(x_a - x_g\right)^2 + \left(y_a - y_g\right)^2}$, where a means agent and g means goal. This negative reward should be maximized. The agent's action space consists of the four directions of going up, left, down and right. The episode is terminated after 1000 steps or after 20 "goals" are collected. This environment can be thought of as it would be a control problem for patrol.

For both tests, the system was run for 100,000 steps and it was restarted 10 times. The replay memory had the size of 10,000, the higher gamma value was taken to be 0.99. The learning rate was 1e-4, the activation function was ReLU and the batch size was 32. The lower gamma value of the proposed network was selected to be 0.7 in the first experiment, and it was 0.7 and 0.8 for the second one. All the hyperparameters are selected to provide a good balance between convergence and learning speed.

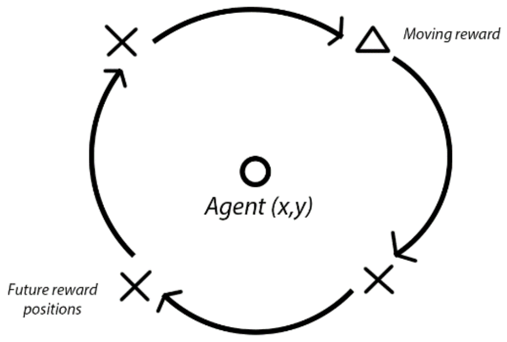

Figure 3
The gridworld environment

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\boldsymbol{\theta}$
Initialize target model $\hat{Q}$ with weights $\boldsymbol{\theta}^- = \boldsymbol{\theta}$
Initialize action-value function $Q'$ with random weights $\boldsymbol{\theta}'$
Initialize target model $\widehat{Q'}$ with weights $\boldsymbol{\theta}'^- = \boldsymbol{\theta}'$
**for** episode = 1, $M$ **do**
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\Phi_1 = \Phi(s_1)$
    **for** $t$ = 1, T **do**
        With probability $\epsilon$ select a random action $a_t$
        otherwise:
        **if** $ratio(Q') < ratio(Q)$ (based on (11)) **then**
            calculate:
```

```
            $P = (1 - \frac{ratio(Q)}{ratio(Q)+ratio(Q')})^k$   (6)
            with probability $P$ select action $\boldsymbol{a_t} = argmax_a Q'(\Phi(\boldsymbol{s_t}), \boldsymbol{a}; \boldsymbol{\theta}')$
            otherwise select $\boldsymbol{a_t} = argmax_a Q(\Phi(\boldsymbol{s_t}), a; \boldsymbol{\theta})$
        **else**
            select $\boldsymbol{a_t} = argmax_a Q(\Phi(s_t), a; \boldsymbol{\theta})$
        **end if**
        Execute action $\boldsymbol{a_t}$ and observe reward $r_t$ and image $\boldsymbol{x_{t+1}}$
        Set $s_{t+1} = \boldsymbol{s_t}, \boldsymbol{a_t}, \boldsymbol{x_{t+1}}$ and preprocess $\Phi_{t+1} = \Phi(\boldsymbol{s_{t+1}})$
        Store transition $(\Phi_t, \boldsymbol{a_t}, r_t, \Phi_{t+1})$ in D
    **end for**
    Sample random minibatch of transitions $(\Phi_j, \boldsymbol{a_j}, r_j, \Phi_{j+1})$ from $D$
        Set $y = \begin{cases} r_j & \text{if episode terminates at step j + 1} \\ r_j + \gamma \max_d \hat{Q}(\Phi_{j+1}, \boldsymbol{a}'; \boldsymbol{\theta}^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\Phi_j, \boldsymbol{a_j}; \boldsymbol{\theta}))^2$ with respect to the network parameters $\boldsymbol{\theta}$
        Set $y' = \begin{cases} r_j & \text{if episode terminates at step j + 1} \\ r_j + \gamma' \max_d \widehat{Q'}(\Phi_{j+1}, \boldsymbol{a}'; \boldsymbol{\theta}'^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $({y'}_j - Q'(\Phi_j, \boldsymbol{a_j}; \boldsymbol{\theta}'))^2$ with respect to the network parameters $\boldsymbol{\theta}'$
    Every C steps reset target model $\hat{Q} = Q$
    Every C steps reset target model $\widehat{Q'} = Q'$
**end for**
```

Algorithm 2
NPV-DQN

Now let us discuss the results of the previous tests. Figure 4 shows the results of the cartpole experiment. It can be seen that the sum of the rewards is getting gradually higher after the 60th episode, leading to a 60% increase over the original DQN algorithm, even despite a little performance decrease after the 150th episode. Figures 5 and 6, show the results for the gridworld experiment, the former showing the first 15 episodes, while the latter showing the episodes from the 15th to the 50th. It is seen that the proposed algorithm is a faster learner, leading to higher reward values from earlier episodes, and reaching the final state 10 episodes earlier than the original DQN algorithm, after 35 episodes instead of 45. It also shows that the γ selected at 0.7 and 0.8 performs good enough as a second discount factor. However, at the 15th episode, the gamma of 0.8 is a bit behind the original one, but before and after that episode it is clearly better.

Conclusions

As it can be seen from the results, the proposed algorithm surpasses the original DQN algorithm, by a mentionable margin and also in regards of learning speed. Due to this, it is a worthwhile addition to the previous action-value based system variants and the proposed idea can be mixed with other augmentations of the DQN system.

There are, however, some ways to continue this research. As mentioned earlier, the mixing of the proposed method and other DQN variants should be considered. Methodologies to reduce computing power, by using one network, instead of two, can also be implemented. In addition, multiagent settings and utilizing short-term rewards should be explored. It could also be valuable to research how well this idea fits with actor-critic methods.

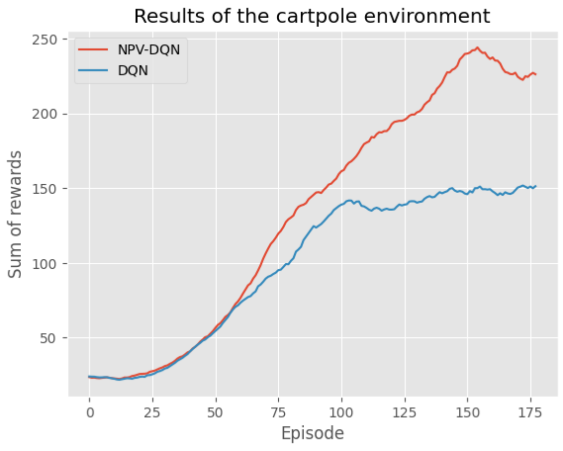

Figure 4
Results for the cartpole environment

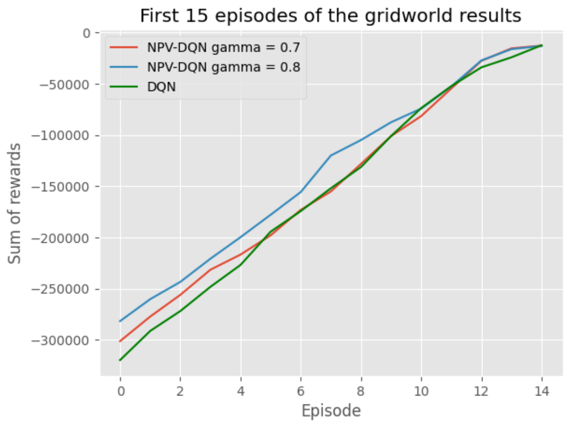

Figure 5
Results of the first 15 episodes of the gridworld environment

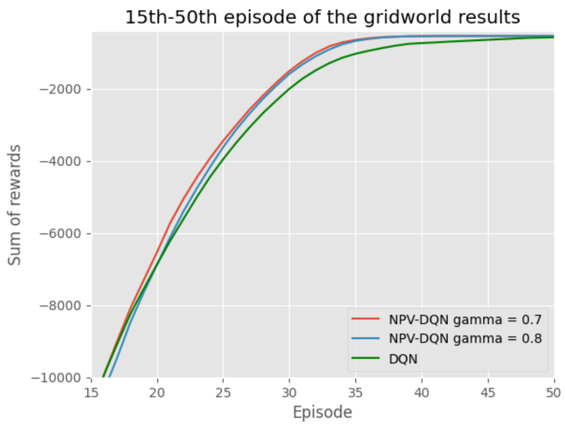

Figure 6
Results of the 15th to the 50th episodes of the gridworld environment

The research reported in this paper is part of project no. BME-NVA-02, implemented with the support provided by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund, financed under the TKP2021 funding scheme. Support by the European Union project RRF-2.3.1-21-2022-00004 within the framework of the Artificial Intelligence National Laboratory.

References

[1] M. G. Bellemare, W. Dabney, R. Munos: A distributional perspective on reinforcement learning, CoRR, abs/1707.06887, 2017

[2] W. Dabney, M. Rowland, M. G. Bellemare, R. Munos: Distributional reinforcement learning with quantile regression, CoRR, abs/1710.10044, 2017

[3] V. François-Lavet, R. Fonteneau, D. Ernst : How to discount deep reinforcement learning: Towards new dynamic strategies, CoRR, abs/1512.02011, 2015

[4] T. Haidegger, L. Kovács, R. E. Precup, B. Benyó, Z. Benyó, S. Preitl: Simulation and control for telerobots in space medicine, Acta Astronautica, 81(1), pp. 390-402, 2012

[5] M. Hausknecht, P. Stone: Deep recurrent q-learning for partially observable mdps, 2015

[6] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, D. Silver: Rainbow: Combining improvements in deep reinforcement learning, CoRR, abs/1710.02298, 2017

[7] M. L. Littman: Markov games as a framework for multi-agent reinforcement learning, In Proceedings of the Eleventh International Conference on Machine Learning, pp. 157-163, Morgan Kaufmann, 1994

[8] X. Li, L. Li, J. Gao, X. He, J. Chen, L. Deng, J, He: Recurrent reinforcement learning: A hybrid approach, 2015

[9] F. S Melo: Convergence of q-learning: A simple proof, Institute Of Systems and Robotics, Tech. Rep, pp. 1-4, 2001

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, J. Antonoglou, D. Wierstra, M. Riedmiller: Playing atari with deep reinforcement learning, arXiv preprint arXiv:1312.5602, 2013

[11] B. Németh: Providing guaranteed performances for an enhanced cruise control using robust lpv method, Acta Polytechnica Hungarica, 20(7), pp. 133-152, 2023

[12] G. Paczolay, I. Harmati: A new advantage actor-critic algorithm for multi-agent environments, In 2020 23rd International Symposium on Measurement and Control in Robotics (ISMCR), pp. 1-6, 2020

[13] G. Paczolay, I. Harmati: A2cm: a new multi-agent algorithm, ACTA IMEKO, 10:28-35, 2021

[14] G. Paczolay, I. Harmati: A simplified pursuit-evasion game with reinforcement learning, PERIODICA POLYTECHNICA-ELECTRICAL ENGINEERING AND COMPUTER SCIENCE, 65: pp. 160-166, 2021

[15] R. E. Precup, S. Preitl, E. M. Petriu, J. K. Tar, M. L. Tomescu, C. Pozna: Generic two-degree-of-freedom linear and fuzzy controllers for integral processes, Journal of the Franklin Institute, 346(10), pp. 980-1003, 2009

[16] S. Preitl, R. E. Precup, J. Fodor, B. Bede: Iterative feedback tuning in fuzzy control systems. Theory and applications, Acta Polytechnica Hungarica, 3(3), pp. 81-96, 2006

[17] A. Reda, R. Benotsmane, A. Bouzid, J. Vásárhelyi: A Hybrid Machine Learning-based Control Strategy for Autonomous Driving Optimization, Acta Polytechnica Hungarica, 20(9), pp. 165-186, 2023

[18] C. H. C. Ribeiro, Cs. Szepesvári: Q-learning combined with spreading: Convergence and results, In Proceedings of ISRF-IEE International Conference: Intelligent and Cognitive Systems, Neural Networks Symposium, pp. 32-36, Tehran, Iran, 1996

[19] T. Schaul, J. Quan, I. Antonoglou, D. Silver: Prioritized experience replay, 2015, cite arxiv:1511.05952Comment: Published at ICLR 2016

[20] M. G. Unguritu, T. C. Nichitelea: Design and assessment of an anti-lock braking system controller, Romanian Journal of Information Science and Technology, 26(1), pp. 21-32, 2023

[21] H. van Hasselt, A. Guez, D. Silver: Deep reinforcement learning with double q-learning, CoRR, abs/1509.06461, 2015

[22] Z. Wang, N. de Freitas, M. Lanctot : Dueling network architectures for deep reinforcement learning, CoRR, abs/1511.06581, 2015

[23] C. J. C. H. Watkins: Learning from Delayed Rewards, Phd thesis, King's College, Oxford, 1989

[24] I. A. Zamfirache, R. E. Precup, R. C. Roman, E. M. Petriu,. Neural network-based control using actor-critic reinforcement learning and grey wolf optimizer with experimental servo system validation, Expert Systems with Applications, 225, 120112, 2023