https://doi.org/10.3311/PPee.36789

Periodica Polytechnica Electrical Engineering and Computer Science, 68(4), pp. 335–343, 2024

Improving Reinforcement Learning Exploration by Autoencoders

Gabor Paczolay[1*], Istvan Harmati[1]

[1] Department of Control Engineering, Budapest University of Technology and Economics, Magyar Tudósok körútja 2., H-1117 Budapest, Hungary
[*] Corresponding author, e-mail: paczolay@iit.bme.hu

Received: 29 February 2024, Accepted: 09 June 2024, Published online: 01 July 2024

Abstract

Reinforcement learning is a field with massive potential related to solving engineering problems without field knowledge. However, the problem of exploration and exploitation emerges when one tries to balance a system between the learning phase and proper execution. In this paper, a new method is proposed that utilizes autoencoders to manage the exploration rate in an epsilon-greedy exploration algorithm. The error between the real state and the reconstructed state by the autoencoder becomes the base of the exploration-exploitation rate. The proposed method is then examined in two experiments: one benchmark is the cartpole experiment while the other is a gridworld example created for this paper to examine long-term exploration. Both experiments show results such that the proposed method performs better in these scenarios.

Keywords

reinforcement learning, DQN, autoencoders, exploration, AutE-DQN

1 Introduction

Reinforcement learning is a quickly emerging field in these days. As computing performance increases, more and more problems are possible to solve by this domain, which is deliberately useful for tasks with either lacking prior knowledge, huge mathematical complexity or unplannable uncertainty.

Due to the initially unknown elements during reinforcement learning, an efficient exploration is needed to discover the policies that are as close to the optimal policy as possible. An efficient exploration policy makes the agents explore while there is still a measurable possibility to find newer optimal policies, and tries to exploit the already acquired knowledge to get as high rewards as possible. This problem is examined in this paper and a new novel theory is proposed.

Probably the most influential algorithm, Q-learning, was created by Watkins [1]. Traditional Q-learning algorithms had a state-action table, increasing with the dimension of the state-space, which limited their usage on larger or continuous state-spaces. Mnih et al. made a huge advancement with Q-learning by successfully applying artificial networks on the problem [2]. The algorithm, DQN, or Deep Q-Learning, also had a so-called experience replay, a buffer containing previous memories, for better convergence, as well as multiple frames were fed to the network of the Atari games it played to provide a better understanding of motion. Afterwards, this algorithm was improved by other researchers. Van Hasselt et al. created a DQN variant with two separate Q-value estimators, the experiences are learning one of the estimators, and the two estimators update each other [3]. Wang et al. created a so-called dueling method with the state-values and the state-dependent action advantages [4]. Schaul et al. created prioritizations in the experience replay to improve convergence [5]. Bellemare et al. took the value distribution into consideration [6]. Hessel et al. combined several previously mentioned improvements for better result [7].

As the full state is not always known, for example due to the limited possibility to explore the state due to the sensors, sometimes the Markov Decision Process problem becomes a Partially Observable Markov Decision Process. To solve such a problem, the system needs to have memory. Hausknecht et al. [8] solved this issue by applying a Long Short-Term Memory network (LSTM) on the inputs. Li et al. [9] merged Supervised Learning with Reinforcement Learning, with the former containing recurrent components, to deal with POMDPs.

Cite this article as: Paczolay, G., Harmati, I. "Improving Reinforcement Learning Exploration by Autoencoders", Periodica Polytechnica Electrical Engineering and Computer Science, 68(4), pp. 335–343, 2024. https://doi.org/10.3311/PPee.36789

Exploration was also researched in the recent past. Stadle et al. [10] made experiments with Thompson sampling and Boltzmann exploration and uses autoencoders to alter the reward. Oh et al. [11] created two systems that are able to predict the action-dependent frames that are useful for control. Osband et al. [12] uses a bootstrapped DQN also reliant on Thompson sampling to measure uncertainty in neural networks for better exploration. Fortunato et al. introduced parametric noise to the deep reinforcement learning agents' weights to let them explore the environment better [13]. Kulkarni et al. [14] created Hierarchical DQN that is a framework for temporal decomposition of the value function. Houthooft et al. [15] implemented curiosity-driven reinforcement learning via bayesian neural networks. Bellemare et al. [16] unified count-based exploration and intrinsic motivation by transforming the counts to exploration bonuses.

In this paper, first we take a look at the theoretical background. After this, a summary of the underlying theory is given to our algorithm. Later, we show the used benchmarks for our tests, then we explain our experiments and the results obtained by them. At the end, we conclude our work and give suggestions on future research possibilities.

2 Theoretical background

2.1 Markov Decision Processes

Markov Decision Process is a discrete-time process for decision making modeling. Fig. 1 shows the basics of this mathematical framework [17]. It has the following elements: states of the whole environment, selectable actions by the agent, transition probabilities between the states with respect to the actions and rewards given to the agents [18]. Every timestep the process has the same method: starting at a specific state (s), it has an available action space. From that, the agent selects and action (a), and based on the state-action pair, it will receive a reward (r), then it arrives in a new state (s'). A stochastic process is called Markovian if

$$P\left(a^t = a \mid s^t, a^{t-1}, \ldots, s^0, a^0\right) = P\left(a^t = a \mid s^t\right), \quad (1)$$

which can be described such as state transitions depend only on the last state and the currently selected action. Due to this, only these two are important in the decision of the following state [19]. P is, in fact, a probability of the transitions.

The policy, a state-action assignment, is very important in Markov Decision Processes. Agents are trying to seek for an optimal one which can maximize the return, the sum of discounted expected rewards. Discount in this case means that agents prefer an immediate reward against one in the future, thus a coefficient determines how better a reward is with respect to the same amount of reward in the next state. A solution would mean a policy that is able to maximize the reward reached by the agent. To find a general solution for the policy, one have to seek for a fixed point of the *Bellman equation* via iterative search. The Bellman equation has the following form:

$$v\left(s, \pi^*\right) = \max_a \left(r\left(s,a\right) + \gamma \sum_{s'} p\left(s' \mid s, a\right) v\left(s', \pi^*\right)\right), \quad (2)$$

where $r(s, a)$ is the reward gained from selecting action a in state s, γ is the coefficient deciding how much more important are rewards of the present in comparison with the future rewards, and $p(s'|s, a)$ is the transition probability function. It is concluded from this equation that if the agent is familiar with the dynamics of the environment, meaning that it knows P and r, it can find the optimal values.

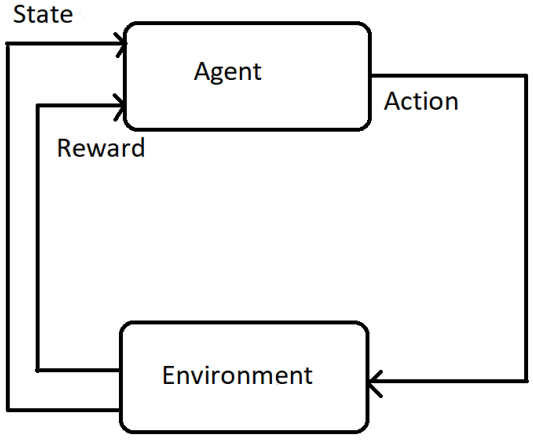

Fig. 1 Markov Decision Process

2.2 Reinforcement learning

In the case of not knowing either the reward function or the state transition probabilities, the previously discussed Markov Decision Process problem becomes a Reinforcement learning problem. When this happens, the agent tries to familiarize itself in the environment by trying actions and makes an environment model based on the rewards it receives.

The two main types of reinforcement learning are called value-based and policy-based. In the former, certain values are associated with the explorable states or its selectable actions. These values can be described as the achievable rewards when the playing agent transitions to a specific state or selects an action in a state.

The most known value-based reinforcement learning method is called **Q-learning**. In this case, the action-values are called Q-values, and they are linked to each state-action pairs of the system, and are the estimation of the cumulated reward one can achieve by selecting the aforementioned action from the specific state. The update of the Q-values is executed by the Eq. (3) [1]:

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

where α is the learning rate and γ is the discount for the reward. This algorithm is a temporal difference algorithm, and the followed policy is choosing the action that would currently maximize the Q-function in the current state.

Regarding the second type of reinforcement learning called policy-based reinforcement learning, the actions are calculated via a function of the state with a set of parameters. Regarding this kind, the most common 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 $r(\tau)$. We obtain the following payoff function:

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

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

$$\theta_{k+1]} = \theta_t + \alpha\Delta J\left(\theta_t\right) \tag{5}$$

Let's take a look on the advantages and disadvantages of policy-based reinforcement learning. While value-based systems cannot map huge or continuous action spaces due to the values rendered to each action, policy-based methods are able to do so due to the parametrization. Stochastic systems are also better suited to policy-based methods. On the contrary, it owns a disadvantage that it might stuck in local maxima much easier that value-based methods.

Another apportionment of reinforcement learning is based on the question whether it is following a model. In model-based reinforcement learning, the learning is basically a tuning of a suitable model. In model-free systems, however, there is no need for a model during the learning. Model-based methods usually require less training samples to achieve similar performance as the model-free ones, but this performance is heavily determined by the model that is trained. The two methods can be combined to achieve better results by training a model-based system first to ensure convergence, then utilizing a model-free method to fine-tune the results.

2.3 Deep Reinforcement Learning

In the case when the reinforcement learning algorithm is aided by a neural network as a function approximator, the problem is called deep reinforcement learning [20].

The building blocks of a neural network are called neurons. These are similar to the biological neurons that can be found in the brain, and their function can be characterized by the Eq. (6):

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

In this case, x corresponds to the input vector, w is the weight vector, and the dot product of these two are taken. b is the bias, which ensures that the dot product can be shifted from the value zero. It can also be characterized as a neuron input connecting to the value of constant one. The function marked with $Act()$ is called activation function, this allows the system to have the nonlinearity that is required to make any kind of predictions. The algorithm for tuning the variables w and b is called backpropagation. In this case, the partial derivative errors of the inputs are calculated by starting at the final error and propagating backwards through previous layers up until the input vector.

Vanishing gradients pose a significant problem in deep neural networks. In this case, several layers are stacked after each other, and the gradients of training decrease at every layer, thus layers that are closer to the input vector do not train expectedly. Due to this issue, one needs to find an appropriate activation function that is less prone to vanishing gradients. In this short summary of deep learning, three activation functions are shown. A simple but frequent activation function is the so-called sigmoid or logistic activation function:

$$y = \frac{1}{1+e^{-x}} \tag{7}$$

Due to its small gradients when moving away from the zero point, gradients vanish easily in this activation function. The Rectified Linear Unit (ReLU) solves this problem by setting a constant gradient in the positive area of the function, and is characterised by the Eq. (8):

$$\begin{aligned} y &= x \quad \text{if} \quad x > 0 \\ y &= 0 \quad \text{if} \quad x \le 0 \end{aligned} \tag{8}$$

The result aforementioned activation functions were only determined by the outcome of the singular neurons. On the other hand, the output of the softmax activation function is determined by the output of the whole layer of the network. The function is made such that the outputs are always nonnegative, while the sum of the outputs are

equal to one. This system is commonly used for classification problems, but in reinforcement learning, the action space probability distributions are made from it. The behavior is described by the Eq. (9):

$$y = \frac{e^{x_i}}{\sum_j e^{x_j}}, \tag{9}$$

where x are the inputs. Finally, let's talk about the differences between traditional and deep reinforcement learning systems. Generally, deep reinforcement learning has worse convergence than traditional algorithms due to the approximations of the states, which are not accurate, although several improvements exist to overcome this obstacle. However, the multitude of advantages have made deep reinforcement algorithms to be usually used. State table is no longer required, as it is approximated by neural networks, so systems with huge or continuous state spaces can be modeled.

2.4 DQN algorithm

The Deep Q Network (DQN) algorithm can be thought of as an adaptation of the Q-learning algorithm for deep reinforcement learning. It is suited for discrete action space, and the state space has no limitations. The input of the network is the state, while the output vector has as much elements as the action space. The output values are equal to the approximation of the Q-values for all of the actions, that is, the achievable reward from the state by taking the specific action. This method makes it possible to compute all of the Q-values of a state in only one inference run. The algorithm can be seen in Algorithm 1.

As mentioned before, deep reinforcement learning methods have the disadvantage of not converging as easily as regular reinforcement learning methods do. To have better results, an experience replay is utilized. The experience replay buffer holds the tuples of the states, selected actions, obtained rewards, the booleans of terminal states and the states where the agent have transitioned, marked as (s, a, r, d, s'). Upon learning (usually after episodes) an array of tuples are sampled from the buffer, and the learning is performed on this array. This ensures that earlier memories also appear in the learning, thus making more diversity and making the algorithm to converge better.

Another invention to achieve better convergence is the target model. During learning, not the normal model is used for inference, but a so-called target model, a model similar to the original model but it does not change that much as the original one. The updating of the target model is performed in some periods, and the update can be a hard update by cloning the values of the original model, or a soft update by approaching the target model's values to the original model's.

Algorithm 1 DQN

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target model $\hat{Q}$ with weights $\theta^- = \theta$

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 ϵ select a random action a_t

    otherwise select $a_t = argmax_a Q(\Phi(s_t), a; \theta)$

    Execute action a_t and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\Phi_{t+1} = \Phi(s_{t+1})$

    Store transition $(\Phi_t, a_t, r_t, \Phi_{t+1})$ in D

  end for

  Sample random minibatch of transitions $(\Phi_j, a_j, r_j, \Phi_{j+1})$ from D

  Set $y = \begin{cases} r_j & \text{if episode ends at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}\left(\Phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$

  Perform a gradient descent step on $(y_j - Q(\Phi_j, a_j; \theta))^2$ with respect to the network parameters θ

  Every C steps reset target model $\hat{Q} = Q$

end for

2.5 Autoencoders

Autoencoders are neural networks where the desired output of the function approximator on the training set is the input. The networks have two distinct parts: one encoder and one decoder. First, the encoder encodes the input to the so-called latent space, then the decoder reconstructs the original value from the latent space. The latent space is a vector with different dimensions from the original input. Fig. 2 shows an autoencoder.

When the latent space is smaller than the input space, the encoder compresses the input in a way that it is decompressable for the decoder. Apart from the compression

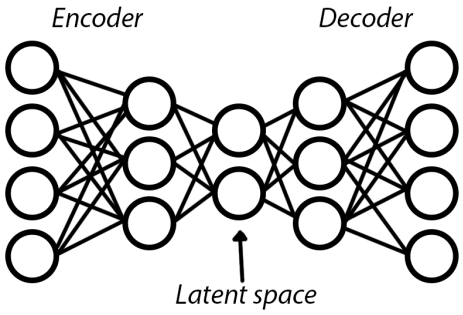

Fig. 2 Autoencoder

tasks, several other duties can be done by these systems, as the decoder will always decompress the latent space such that the output is "similar" to the training set. For example, it can be used for anomaly detection, where we can assume, that if there is a significant difference between the input and the output, there is an anomaly. It is also suitable for interpolation, for example between two faces. IT can also be used for data generation if we feed the network with random noise. If the training input is a distorted version of the input (and in this case, the output), the networks can be used for data reconstruction, for example, for de-noising or coloring a grayscale image. When the latent space is bigger than the original input space, the encoding is sparse, hence we call them as sparse autoencoders.

3 The proposed concept

The algorithm's hypothesis is based on the functioning of autoencoders. In general, an autoencoder's decoder tries to estimate the input based on the latent space. The encoder and the decoder are learned via the training set.

The starting point is that the autoencoder's estimates are 'closer' to the input data when it is equal to or similar to the training set, due to the fact that the autoencoder would be able to learn the characteristics of the training inputs that were introduced to the network. This closeness can be measured as an error, for example a Mean Squared Error.

The proposed idea is that we can set the training set of the autoencoder to be the set of states that the reinforcement learning algorithm is trained on. In reality, this means the batches of samples from the replay buffer which are the same samples as the ones used in the DQN training, just without the other data in the tuple. The other data are not required for the method in this case, as only the states are the ones that the autoencoder system is trained on. In the Markov Decision Process loop, during the action selection the current state is fed to the autoencoder. We can assume that if the autoencoder has been trained on the same (or a similar) input as the state the system is currently visiting, the error would be significantly lower compared to the states that are completely new to the autoencoder, and thus also compared to the states on which the reinforcement learning has been not trained on. Also, based on the general behavior of the autoencoders, we can assume that we can find an autoencoder architecture such that as the number of training increases on one state, the related error decreases.

Being able to make a distinction between previously 'visited' (or in this case, 'trained' is a more precise definition) states, especially without directly recording the total trajectory, can make a difference in the reinforcement learning process. We propose the idea that it can be used as a tool to set the exploration-exploitation chance ratio, or in a shorter manner, epsilon.

The functioning of the proposed concept can also be derived from the method of how the human mind works [21]. The theory of active inference states that the brain is continuously making an inner replica of the environment based on the beliefs and is selecting actions such that the action is leaning towards the most stable state, or in other words, towards the state that is inferred with the least reconstruction error. There have been researches upon this type of learning but as of the creation of this paper, utilizing the full theory have not been successful enough to be used in real-world scenarios. We use only the modeling part and put it into a reinforcement learning algorithm.

Fig. 3 shows a drawing about the basics of the theory. The current state is fed to an autoencoder that is trained on previous states on which the DQN algorithm was trained. The weighted average (where the weights are based on a priori knowledge of the system) is taken of the error between the current and reconstructed state, where the weights have to be tuned to match the most important state variables based on the problem. Then, a constant product is taken of the average to match the problem, this scales the exploration-exploitation rate to the 0..1 interval.

To conclude, the proposed idea solves the problem of getting to know how well-known and well-practiced a state is. This should make the system perform better when there is a newly explored, distinct state or state ensemble which is not yet practiced.

4 Experiments

After setting the grounds of the algorithm, its justification, the required modifications and the limitations of the theory need to be examined in experiments. Algorithm 2 shows the proposed algorithm, where some parts will be different for the following two experiments. The loss of the autoencoder was the mean squared error between the sampled states and the inferred ones.

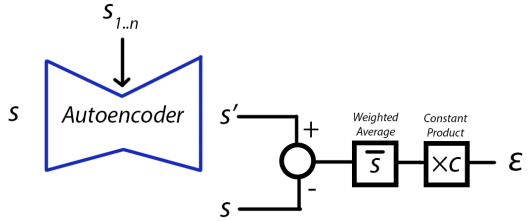

Fig. 3 Theory of the research

Algorithm 2 AutE-DQN

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

Initialize autoencoder A with random weights θ_a

for episode = 1, M do

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\Phi_1 = \Phi(s_1)$

for $t = 1$, T **do**

calculate $\epsilon a = F((s_t - A(s_t)) \cdot w)$ where F is defined as Eq. (10) or Eq. (11)

With probability $\max(c \cdot \epsilon, \epsilon_a)$ select a random action a_t

otherwise select $a_t = \text{argmax}_a Q(\Phi(s_t), a; \theta)$

Execute action a_t and observe reward r_t and image x_{t+1}

Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\Phi_{t+1} = \Phi(s_{t+1})$

Store transition $(\Phi_t, a_t, r_t, \Phi_{t+1})$ in D

end for

Sample random minibatch of transitions $(\Phi_j, a_j, r_j, \Phi_{j+1})$ from D

$$\text{Set } y = \begin{cases} r_j & \text{if episode ends at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}\left(\Phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$$

Perform a gradient descent step on $(y_j - Q(\Phi_j, a_j; \theta))^2$ with respect to the network parameters θ

Perform a gradient descent step on $((s_j - A(s_j, \theta_a)) \cdot w)^2$ with respect to the network parameters θ_a

Every C steps reset $\hat{Q} = Q$

end for

The first and most important observation to make is that the weighting of the errors based on the states makes a measurable improvement on the performance. Thus, in both of the environments, it was utilized. In the algorithm, the weighting is denoted by . These weights were designed such that they sum to one.

4.1 Cartpole

The first experiment was to use the error of the autoencoder as the greedy factor in the action selection process. For this experiment, a Cartpole environment with discrete actions was used. The Cartpole environment used is a discrete-action benchmark environment, first used by Barto [22]. It is a useful benchmark due to the fact that it is related to classical control (representing a physical pole and a cart which is also possible to build), but it is more difficult in the discrete setting. Its observation space consists of four elements: the position and the velocity of the cart, in addition to the angle and the angular velocity of the pole. The action space consists of two actions: pushing the cart either to the left or to the right. The reward is +1 for every step taken, and the termination step is either at step 200 or the moment when the pole angle is greater than ±12 ° or the cart position is greater than ±2.4. For this experiment, the epsilon utilized by us is equal to the loss of the autoencoder, that is, that $F()$ is equal to the weighted mean squared error, so

$$F(x) = x^2 \tag{10}$$

The state weights w utilized by us for the Cartpole environment were 1/28 for the position and 9/28 for the other three state elements. This is due to the fact that the position itself has only a slight impact on the behavior of the system. The epsilon coefficient, c, is set to one.

This experiment was conducted with an autoencoder encoder layer dimension of [3, 2] and DQN layers of (32, 16). This experiment was run on 10 random seeds.

4.2 Gridworld

The next experiment was to check the performance of the proposed algorithms in situations where we can assume that it should be better suited to the situation than the traditional DQN algorithm due to its traits. To achieve this, we created an environment where there are always possibilities of achieving more rewards by exploration. In theory, this means that the proposed algorithm should behave better even when the algorithms with general epsilon-decay work with only a small amount of exploration. The environment suited to this situation is a simple grid-world environment. The agent starts at coordinate (0,0), while the collectibles appear at (±3, ±3). After a time, the collectibles change position, this timing is (3, 4, 5 6) steps respectively, one after each other. After the collectible is collected, a new one appears at the next position in the clockwise order. The state consists of five elements: the x and y position of the agent, the number of previously collected collectibles, and the two states of timing: one corresponding to which state are we from the four states, and one corresponding to the number of steps taken since the last change of timing. The actions are the movements in the four directions. Fig. 4 shows the basics of the grid-world environment.

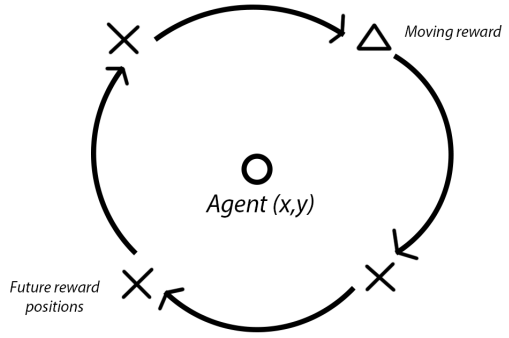

Fig. 4 The gridworld environment

For this experiment, the w weights were set that the x, y and the steps since the last timing change were 1/15, the timing state was 1/5, and the number of collectibles taken was 3/5. The c coefficient for our epsilon was 0.001, this number is such low compared to the previous experiment because here we do not take the normal mean squared error for $F()$, but we take its square root, so here

$$F(x) = \sqrt{x^2} \tag{11}$$

This ensures that there is a balance between the high spikes at the start of the learning and the learning improvement later. Our ϵ_a is compared to a normal epsilon with a constant decay, and the maximum of these is taken. This also ensures a constant high-exploration learning without being unable to exploit the policy in the later stages of the learning.

This experiment was run with autoencoder layer dimensions of (8, 16). The DQN layers, just like in the Cartpole experiment, were (32, 16). This experiment was also run on 10 random seeds.

5 Results

Now let us see the results of the two previously mentioned experiments.

5.1 Cartpole results

Let us start with the Cartpole experiment. Fig. 5 shows the corresponding results. As it can be seen, the algorithm that utilized only the proposed autoencoder exploration even surpasses the original epsilon-decay algorithm. While the learning itself is significantly slower in the first 40 episodes, our algorithm gets higher rewards almost always in the following stages.

Fig. 6 shows the average epsilon values throughout the experiment. It can be seen that at the beginning, the epsilon value is very high, but unlike the regular epsilon-decay algorithm, it gets higher when it encounters new experiences.

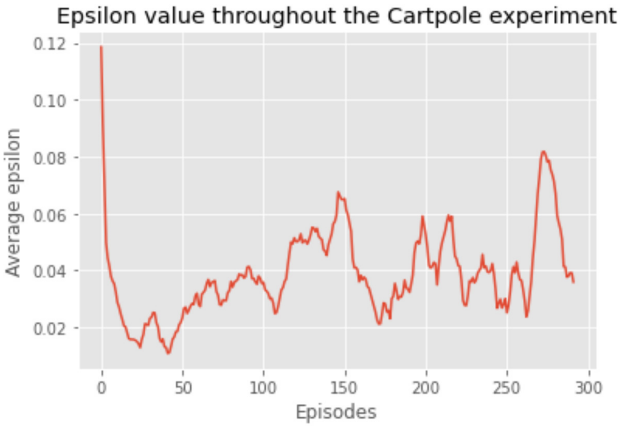

Fig. 6 Cartpole epsilon results

5.2 Gridworld results

Now let's look at the Gridworld experiment. As it can be seen from Fig. 7, our algorithm has significantly had better performance throughout the whole period of 2000 episodes. The regular DQN with epsilon decay has only approached the average reward of our algorithm between episodes 500 and 1200, then it has started to perform worse. Our algorithm, however, has learned more rapidly than the normal DQN and has achieved its peak performance much faster than the other one.

6 Conclusion

As the results prove it, the proposed modifications of the DQN algorithm are an improvement of the epsilon-greedy algorithm in cases where newly explorable states occur in the later stages of the learning. It is also a fine replacement for general cases where there is no real exploitation of the explorable states of late learning.

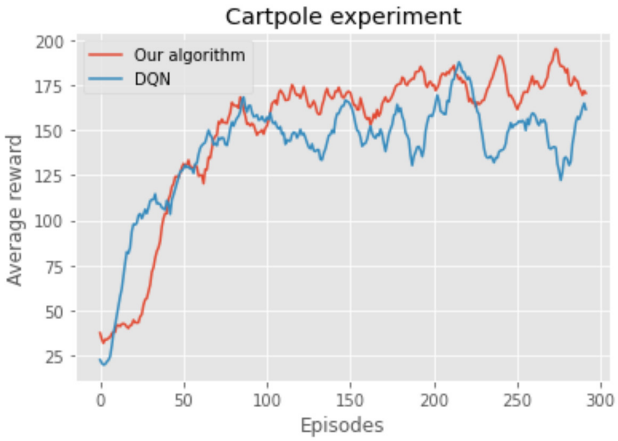

Fig. 5 Cartpole results

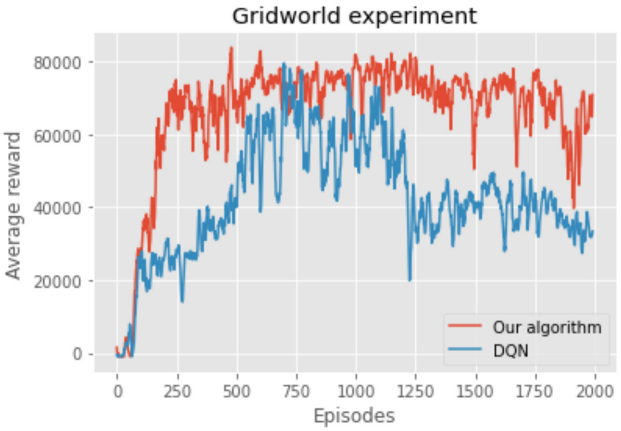

Fig. 7 Gridworld results

There are many ways to further improve upon the algorithm. For example, its relevance in image-based states can be seen, as well as in partially observable MDP-s, where only an observation is known instead of the real states. Multiagent systems might also be improved with a similar technique, to have a distinction between the agent's and the other agents' state variables. Curiosity-driven algorithms, such as [23] could also be seen for merging the intrinsic reward and the adaptive exploration rate.

Acknowledgement

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 the European Union project RRF-2.3.1-21-2022-00004 within the framework of the Artificial Intelligence National Laboratory.

References

[1] Watkins, C. J. C. H. "Learning from Delayed Rewards", PhD thesis, King's College, Oxford, 1989.

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M. "Playing atari with deep reinforcement learning", [preprint] arXiv preprint, 19 December 2013. https://doi.org/10.48550/arXiv.1312.5602

[3] Van Hasselt, H., Guez, A., Silver, D. "Deep reinforcement learning with double q-learning", In: AAAI'16: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, Phoenix, Arizona USA, 2015, pp. 2094–2100. ISBN 9781713856566 https://dl.acm.org/doi/10.5555/3016100.3016191

[4] Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., de Freitas, N., Lanctot, M. "Dueling network architectures for deep reinforcement learning", In: ICML'16: Proceedings of the 33rd International Conference on International Conference on Machine Learning, New York City, NY, USA, 2015, pp. 1995–2003. ISBN 9781510829008 https://dl.acm.org/doi/10.5555/3045390.3045601

[5] Schaul, T., Quan, J., Antonoglou, I., Silver, D. "Prioritized experience replay", In: ICLR 2016, San Juan, Puerto Rico, 2016. https://doi.org/10.48550/arXiv.1511.05952

[6] Bellemare, M. G., Dabney, W., Munos, R. "A Distributional Perspective on Reinforcement Learning", In: ICML'17: Proceedings of the 34th International Conference on Machine Learning, Sydney, NSW, Australia, 2017, pp. 449–458. ISBN 9781510855144 https://dl.acm.org/doi/10.5555/3305381.3305428

[7] Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. G., David Silver, D. "Rainbow: Combining improvements in deep reinforcement learning", In: Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2017, pp. 3215–3222. https://doi.org/10.1609/aaai.v32i1.11796

[8] Hausknecht, M., Stone, P. "Deep recurrent q-learning for partially observable mdps", [preprint] ArXiv, 11 January 2017. https://doi.org/10.48550/arXiv.1507.06527

[9] Li, X., Li, L., Gao, J., He, X., Chen, J., Deng, L., He, J. "Recurrent reinforcement learning: A hybrid approach", [preprint] ArXiv, 19 November 2015. https://doi.org/10.48550/arXiv.1509.03044

[10] Stadie, B. C., Levine, S., Abbeel, P. "Incentivizing Exploration In Reinforcement Learning With Deep Predictive Models", [preprint] ArXiv, 19 November 2015. https://doi.org/10.48550/arXiv.1507.00814

[11] Oh, J., Guo, X., Lee, H., Lewis, R. L., Singh, S. "Action-conditional video prediction using deep networks in Atari games", In: NIPS'15: Proceedings of the 28th International Conference on Neural Information Processing Systems, Cambridge, MA, USA, 2015, pp. 2863–2871. ISBN 9781510825024 https://dl.acm.org/doi/10.5555/2969442.2969560

[12] Osband, I., Blundell, C., Pritzel, A., Van Roy, B. "Deep exploration via bootstrapped DQN", In: NIPS'16: Proceedings of the 30th International Conference on Neural Information Processing Systems, Barcelona, Spain, 2016, pp. 4033–4041. ISBN 978-1-5108-3881-9 https://dl.acm.org/doi/10.5555/3157382.3157548

[13] Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., … Legg, S. "Noisy networks for exploration", [preprint] ArXiv, 30 June 2017. https://doi.org/10.48550/arXiv.1706.10295

[14] Kulkarni, T. D., Narasimhan, K., Saeedi, A., Tenenbaum, J. B. "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation", In: NIPS'16: Proceedings of the 30th International Conference on Neural Information Processing Systems, Barcelona, Spain 2016, pp. 3682–3690. ISBN 978-1-5108-3881-9 https://dl.acm.org/doi/10.5555/3157382.3157509

[15] Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., Abbeel, P. "Curiosity-driven exploration in deep reinforcement learning via bayesian neural networks", [preprint], ArXiv, 31 May 2016. [online] Available at: https://arxiv.org/pdf/1605.09674v1 [Accessed: 29 January 2024]

[16] G. Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., Munos, R. "Unifying count-based exploration and intrinsic motivation", In: NIPS'16: Proceedings of the 30th International Conference on Neural Information Processing System, Barcelona, Spain, 2016, pp. 1479–1487. ISBN 978-1-5108-3881-9 https://dl.acm.org/doi/10.5555/3157096.3157262

[17] Paczolay, G. Harmati, I. "A New Advantage Actor-Critic Algorithm For Multi-Agent Environments", In: 2020 23rd International Symposium on Measurement and Control in Robotics (ISMCR), Budapest, Hungary, 2020, pp. 1–6. ISBN 9781665404792 https://doi.org/10.1109/ISMCR51255.2020.9263738

[18] Littman, M. L. "Markov games as a framework for multi-agent reinforcement learning", In: ICML'94: Proceedings of the Eleventh International Conference on Machine Learning, New Brunswick, NJ USA, 1994, pp. 157–163. [online] Available at: https://courses.cs.duke.edu/cps296.3/spring07/littman94markov.pdf [Accessed: 29 January 2024]

[19] Paczolay, G., Harmati, I. "A simplified pursuit-evasion game with reinforcement learning", Periodica Polytechnica Electrical Engineering and Computer Science, 65(2), pp. 160–166, 2021. https://doi.org/10.3311/PPee.16540

[20] Paczolay, G., Harmati, I. "A2cm: a new multi-agent algorithm", Acta IMEKO, 10(3), pp. 28–35, 2021. https://doi.org/10.21014/ACTA_IMEKO.V10I3.1023

[21] Friston, K. J., Parr, T., de Vries, B. "The graphical brain: Belief propagation and active inference", Network Neuroscience, 1(4), pp. 381–414, 2017. https://doi.org/10.1162%2FNETN_a_00018

[22] Barto, A. G., Sutton, R. S., Anderson, C. W. "Neuronlike adaptive elements that can solve difficult learning control problems", IEEE Transactions of Systems, Man, and Cybernetics, SMC-13, pp. 834–846, 1983. https://doi.org/10.1109/TSMC.1983.6313077

[23] Pathak, D., Agrawal, P., Efros, A. A., Darell, T. "Curiosity-driven Exploration by Self-supervised Prediction", ICML'17: Proceedings of the 34th International Conference on Machine Learning, Sydney, NSW, Australia, 2017, pp. 2778–2787. [online] Available at: https://proceedings.mlr.press/v70/pathak17a/pathak17a.pdf [Accessed: 29 January 2024]