Hindawi
Mobile Information Systems
Volume 2021, Article ID 9959954, 11 pages
https://doi.org/10.1155/2021/9959954

Research Article

The Implementation of Deep Reinforcement Learning in E-Learning and Distance Learning: Remote Practical Work

Abdelali El Gourari,[1] **Mustapha Raoufi**,[2] **Mohammed Skouri**,[1] **and Fahd Ouatik**[1]

[1]*Laboratory of Physics High Energy and Astrophysics, Faculty of Sciences Semlalia, Cadi Ayyad University, Marrakech, Morocco*
[2]*Laboratory of Materials Energy and Environment, Faculty of Sciences Semlalia, Cadi Ayyad University, Marrakech, Morocco*

Correspondence should be addressed to Abdelali El Gourari; abdelali.elgorari@uca.ac.ma

Received 24 March 2021; Revised 24 May 2021; Accepted 11 June 2021; Published 25 June 2021

Academic Editor: Salvatore Carta

The world has seen major developments in the field of e-learning and distance learning, especially during the COVID-19 crisis, which revealed the importance of these two types of education and the fruitful benefits they have offered in a group of countries, especially those that have excellent infrastructure. At the Faculty of Sciences Semlalia, Cadi Ayyad University Marrakech, Morocco, we have created a simple electronic platform for remote practical work (RPW), and its results have been good in terms of student interaction and even facilitating the employment of a professor. The objective of this work is to propose a recommendation system based on deep quality-learning networks (DQNs) to recommend and direct students in advance of doing the RPW according to their skills of each mouse or keyboard click per student. We are focusing on this technology because it has strong, tremendous visibility and problem-solving ability that we will demonstrate in the result section. Our platform has enabled us to collect a range of students' and teachers' information and their interactions with the learning content we will rely on as inputs (a large number of images per second for each mouse or keyboard click per student) into our new system for output (doing the RPW). This technique is reflected in an attempt to embody the virtual teacher's image within the platform and then adequately trained with DQN technology to perform the RPW.

1. Introduction

In 2017, Ouatik et al. [1] created an electronic platform for remote practical work that can describe the concept, methodology, and the results of the advanced step supported by the E-lab project of L'AUF (L'Agence Universitaire de la Francophonie). The goal of this platform is combining e-technologies and e-pedagogies to create online undergraduate courses in engineering like practical experience. This project is a part of the improvement and development of the distance learning system (e-learning), especially remote laboratories, which are new technologies that allow learners or researchers to create and conduct scientific experiments and deepen their experimental knowledge in a remote lab through the web. The good and important characteristics of this e-learning system are using and respecting the educational and pedagogical standards followed in teaching and the performance of this system concerning the speed and precision of interaction with the laboratory. Moreover, this system must make the work and the tasks easier for the laboratory manager in matters of preparing and managing the access to experiments and manipulation.

Machine learning is one of the most important branches of artificial intelligence, which is used in a range of fields and disciplines [2–4], including the field of education that we will focus on [5]. The AI is divided into three branches: supervised learning, unsupervised learning, and reinforcement learning. Some of these types are employed and used according to what we want to do and the forms presented. Reinforcement learning (RL) focuses on examining actions that gain a maximal value of cumulative reward from the environment. Moreover, RL utilizes a trial-and-error learning process to achieve its goals. This unique feature has been confirmed to be an advanced approach to building a

human-level agent [6]. In 1992, Mahadevan and Connell built a dynamic robot based on RL named OBELIX that learned how to push boxes [7]. In 1996, the Sarcos humanoid DB was constructed by Schaal to learn the pole-balancing task [8]. An RL method was proposed to control the dynamic walking of a robot without prior knowledge of the environment [9]. In [10], Büchler et al. employed RL to train a robot to play table tennis, and also, in [11], Riedmiller et al. applied a batch RL to prepare crucial skills for a soccer-playing robot. Finally, the RL has become one of the biggest common methods for building an autonomous agent; the shortcomings of traditional learning approaches restrain it from dealing with complex problems [12]. Recent integrations of deep learning methods with RL have improved the performance of existing RL methods considerably. To create a smart agent, the researchers combined deep learning with RL to obtain the deep reinforcement learning that is virtually unbeatable in a series of video Atari games [13, 14]. To extend the success of deep RL, Google's DeepMind subsidiary created AlphaGo, a program that beat one of the best professional Go players of all time, in 2016 [15]. Also, Carta et al. [16] applied a deep quality-learning networks (DQNs) method in efficient stock market predictions. Another paper published by Carta et al. [17] presented a step toward such a task by proposing an ensemble of the deep Q-learning classifiers with different experiences with the environment. Additionally, Google, Uber, and Tesla are hastening the research on deep RL to design the new generation of smart self-driving cars. DQNs [18] connect the dots between deep neural networks with RL by subduing the intractable problems in traditional RL methods. Particularly, DQNs leverage a convolutional neural network (CNN) to analyse input images and use these CNNs to approximate the Q-value function [12]. In other words, the goal of DQNs is to minimize the loss function of a CNN as we will show this expression ahead. As a result, DQNs create a smart agent that outperforms the best RL method so far in the test series of Atari games.

The reason that we focus on deep reinforcement learning is its high efficiency with which we can solve some problems that are sometimes difficult. This technology includes all the problems involved in making its series of decisions. In the learning process, it helps the learner interact and integrate into the lesson and even the exam as the student performs a set of procedures to solve a problem. All of these actions and behaviours are then collected and analysed by the reinforcement-learning-based system to inform the student about the action they have taken whether they were right or not. Another reason is its algorithms, which have begun to produce very good results and sometimes excellent results in many challenging environments. This is where it can be said that deep RL is increasingly developing and is beginning to show new algorithms that will give it a great potential to solve future difficult challenges. Our first problem is about how to employ deep reinforcement learning techniques in the learning process and how they can help students improve their cognitive levels, guide them well, and teach them how to interact with their RPW in a real-world manner. The second problem is about designing new smart electronic-interface-based DQNs including the virtual teacher (agent) interacting with a custom environment that can help our students integrating. To define exactly the contribution in this paper, we adopt a framework for promoting learning, and we will focus on the implementation of DQN approaches that have a significant impact on building a smart agent that can do the RPW intelligently in a very short period of time; after the training of this agent, we will give it to our students to teach them how to do their RPW.

2. Proposed System Architecture

As our problem is about how to employ deep reinforcement learning techniques in the learning process and how they can help students improve their cognitive levels, guide them well, and interact with them in a real-world manner, we have proposed an illustration that shows this technology's work plan and its use in education in general. Figure 1is divided into three important parts.

2.1. Part 1. This is the part of making the structure of educational content, which is the main essence of the process and the success of the process because, as is usually known in social media and television channels, if the structure and content of the presentation to people is good, people will interact with it in a great way and considered it, so the focus must be on this aspect.

2.2. Part 2. This is the part that is related to the student. Our main goal is to help students and provide all the requirements they need to achieve their goals, so we, as designers, are responding and getting out in the simplest way to do this; as can be observed, for example, in Figure 2, a student interacts with a simple environment appearing on their computer screen. A circuit consisting of a set of electrodes allows the student to press on an existing circuit breaker for, for example, electrical current or power calculations. It is only the image of the electronics field that can replace this image with another field of mechanics, optics, etc. and do the same operation as required.

2.3. Part 3. This is a statement about the pictures of a fictitious teacher, factory, or robot that guides students by giving them a set of suggestions, which are in the form of written texts, sound, and pictures or a set of movements that inspire the students what they will do.

To understand deeply parts 2 and 3 on how do they work in our educational process, we will give the following explanation:

(i) Input images: they appear to the students, whose expressions are about exercises, testing, or images of system-based text

(ii) Deep Q-learning networks they are the part related to processing of these pictures and extracting

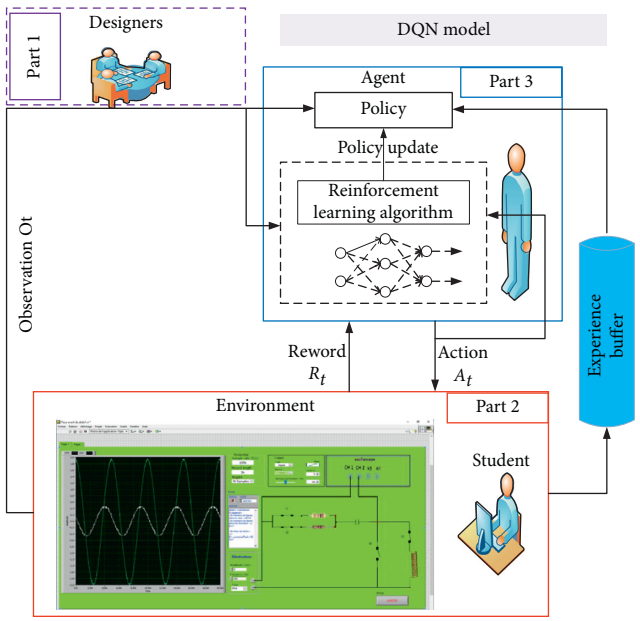

Figure 1: Architecture overview of the education process using deep reinforcement learning.

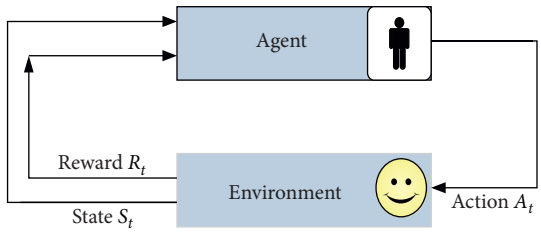

Figure 2: The interaction between the agent and environment.

similar elements and links that link these pictures depending on system weights

(iii) Possible action is taken: it is the most important stage for the learners, through which they take the actions and decisions they deem right to obtain a good reward from their environment, which is in the form of points, or the proportion of their successor falls in the substance to which they are presented, and the environment also tells them what to do to succeed or get a good score and a good reward

This technology helps the students to interact with the platform and the environment with which they interact. The agent who can make decisions directs the students to the things they must do and things that they do not need to do so that they can understand the subject. The agent considers these workers. It is added that these workers may take place at different times, as it may be simultaneous or not simultaneous, allowing the student to take the right time to study. It also allows students to be attracted to integrate into the learning process by exchanging dialog and interacting with this platform, which will provide it with things to do that.

3. Methods

3.1. Reinforcement Learning: Agent-Environment. Reinforcement learning is a subfield of machine learning and an autonomous self-education system that learns primarily through trial and error, performs work to increase rewards, and examines how an agent learns to achieve goals in a complex and uncertain environment to achieve the best results.

As seen in Figure 2, the agent is in the current state of S_t, taking action A_t, interacting with and responding to the environment, and redoing $S_{t+1}$and rewarding R_{t+1}; given its state and current reward, the agent chooses the next action and repeats it until its environment is resolved and terminated.

In our case, the RL task is about training an agent (the virtual teacher) that interacts with its environment (computer screen of the electronic circuit). The agent turns up at different scenarios known as states by performing actions (right puts of the component, wrong puts of the component, right click, wrong click, and outside click). Actions lead to rewards (rewards are assigned based on the result of these actions. If the agent can click true or put true, it calls for a positive reward. However, foul click or wrong put is a negative reward, but outside click does not have anything; there is a sequence of actions required. The main point of RL is learning to carry out these sequences and maximizing the reward) that could be positive and negative. The purpose of our agent is to maximize the total reward through an episode. This episode happens between the first state and the last or terminal state during the environment. We enhance the agent to learn to perform the perfect actions by experience.

3.2. Q-Learning. This agent-driven action may be weak and insufficient to solve, so we use a new technology called Q-learning, which is particularly a reinforcement learning method. This technology can be used to find solutions quickly and ideally, the aim of which is to learn the work plan that tells the agent what action to take in any circumstances that do not demand a model of the environment. The quality (Q) is used to show how useful it is for a particular action to be paid for in the future, and to express what we have said in the past mathematically, we will use the mathematical equivalent known as the Bellman equivalent that embodies the most right and easiest map for getting the most rewards. We will formalize our strategy as follows [19]:

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a), \tag{1}$$

where the Q-value submitted from being at state $S$and implementing action $A$is the immediate reward $R(s,a)$plus the highest Q-value possible from the next state (s'). γ here is the discount factor that controls the contribution of rewards further in the future. $Q(s',a)$again depends on $Q(s'',a)$that will then have a coefficient of gamma squared. So, the Q-value depends on Q-values of future states as shown in the following equation:

$$Q(s,a) \longrightarrow \gamma Q(s',a) + \gamma^2 Q(s'',a) \cdots \gamma^n Q\left(s^{''\cdots n},a\right). \quad (2)$$

Adjusting the value of gamma will reduce the donation of future rewards. Since this is a recursive equation, we can start with making arbitrary presumptions for all Q-values. With nimbleness, it will collect to the optimal policy. In feasible situations, the following is performed as an update [20]:

$$Q_{t+1}(S_t, A_t) \longleftarrow Q_t(S_t, A_t) + \alpha_t(S_t, A_t) \times \left[R_{t+1} + \gamma \max_a Q_t(S_{t+1}, A_t) - Q_t(S_t, A_t)\right], \quad (3)$$

where α is the learning rate. This simply sets to what extent newly acquired information ignores old information.

3.3. Deep Q-Learning. Reinforcement learning has achieved many notable successes in difficult decision making and in solving more complex problems of reality. However, these algorithms require it to be returned as good data before it can achieve a reasonable performance. Sometimes, RL is weak and not optimal, so researchers have resorted to something else called deep Q-networks (DQNs) [21], another technique that combines deep learning with RL, as shown in Table 1 and Figure 3.

In a DQN, we use a neural network to approximate the Q-value function. The state is offered as the input, and the Q-value of all possible actions is produced as the output. The comparison between Q-learning and DQN is illustrated below.

Of course, this technology proved to be good for problems with difficulties and more complex environments. They found that it provided good results that gave a very good quality to the system that adopted that technology.

The steps implicated in RL using a DQN are as follows:

(i) All the experiences are stockpiled by the user in memory

(ii) The next action is fixed by the maximum output of the Q-network

(iii) The loss function is the mean squared error of the portend Q-value and the goal Q-value, Q^*

This is a regression problem. However, we do not know the goal or actual value here as we are transacting with an RL problem. The following is the Q-value update equation obtained from the "Bellman" equation:

$$R_{t+1} + \gamma \max_a Q_t(S_{t+1}, A_t). \quad (4)$$

Equation (4) represents the goal. We can discuss that it is predicting its value, but the term $R_{t+1}$is the equitable true reward; the network is going to update its penchant using backpropagation to converge.

3.4. Deep RL versus Deep Learning. We understood how neural networks can help the agent learn the perfect actions. However, there is a challenge when we contrast deep RL to deep learning:

Nonstationary target: let us go back to the algorithm for deep Q-learning where W is the weight of the network [20].

In Algorithm 1, the goal is continuously changing in every iteration, but in deep learning, the goal variable does not change. Hence, the training is stable, which is just not true for RL. To brief, we often count on the policy or value functions in RL to sample actions. However, this is frequently changing as we continuously learn what to explore. During our work (testing the model), we get to know more about the ground truth values of states and actions, and hence, the output is also changing. Therefore, we attempt to learn to map for a constantly changing input and output (Figure 4).

(i) Target network:

Although the same network is calculating the predicted value and the goal value, there could be a lot of offshoot between them. So, instead of using one neural network for learning, we can use two neural networks as illustrated in Figure 4. We could use a detached network to estimate the target. This network has the same architecture as the function approximate but with frosted parameters. At every iteration, the parameters from the prediction network are reproduced to the target network. This leads to more constant training because it keeps the goal function fixed [19, 21].

(ii) Experience replay:

(a) A casual sample of batch size is chosen from the experience replay buffer

(b) The Bellman equation is used

(c) $Q(s,a) = r(s,a) + \gamma \max_{a'} Q(s',a)$

(d) $Q(s',a')$is obtained from our network by predicting on next state

(e) The network is fitted to the fresh batch

3.5. The DQN Algorithm with Experience Replay. We are going to begin with initializing our replay memory to some capacity that we choose. Then, we are also going to initialize our Q-network just with our random weights; then, we will play an episode that is going to be our training episode, and according to the function, we will initialize our state, using the starting computer screen pixels at the beginning of each episode. We go through a preprocessing step to get our actual input state. For each time step for a work that we are currently doing with a small probability, a random action is

Table 1: Q-learning.

	S0	S1	S2	S3	S4
A0	+2.1	+3.4	+5.6	+6.6	+2.4
A1	+1.1	+2.2	+4.5	+2.1	+1.7

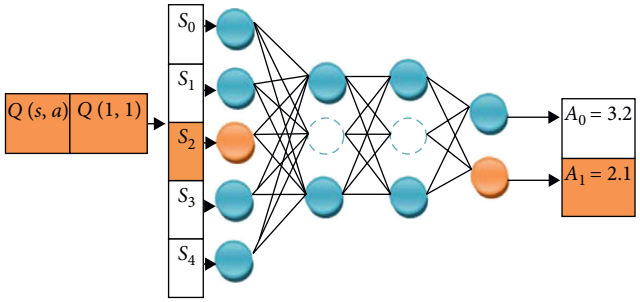

Figure 3: Deep Q-learning.

selected, so one thing that is important in algorithm 2 [22] to have sufficient exploration; that is why we want to make sure that we are sampling different parts of the state space; otherwise, we would select from the greedy action from the current policy. So, most of the time, we will take the greedy action that we think is a good policy of the type of actions that we want to take dates that we want to see, and with a small probability, we will sample something random. We will take this action $A_t$and observe the next reward $R_{t+1}$and the next state S_{t+1}, and we will make the transition and store it in our replay memory M that we are building up, then we are going to train the network a little bit (do experience replay); after that, we will take a small of a random mini-batch of translations from the replay; in other words, we are going to perform a gradient descent tape on this experience replay until getting our fall training loop. Also, we carried out sampling of minibatches experience replay to update the weights of our Q-networkas shown in Algorithm 2.

4. Simulation Results

We implemented the proposed model and environment in Python 3.8.3 and LabVIEW 2017 programming language. Table 2 summarizes the experimental setup of our proposed model. All experiments and results of the system are carried out using Intel(R) Xeon(R) CPU E5-2603 v4 @ 1.70 GHz processor with 24 GB memory. To recommend our system, we used a deep reinforcement learning technique. Moreover, we used Jupyter Notebook as the library and framework.

5. Database Description and Visualization

The database used in this study is constructed from the electronic platform from 2017 to 2020. Our 18,000 students were divided into different groups. We collected a large number of images, as each click taken by the student was accompanied by a picture taken and stored in each student's file. Every day, more than 20 million images were collected where they were sufficient to train our agent. We closely followed the experimental setup of the DQN [6] using the same preprocessing and network architecture. We preprocessed the 210×160 RGB images by downsampling them to 84×84 and extracting the luminance channel.

5.1. Creating an Environment. The reinforcement learning environment for our case is an electric circuit consisting of some interrupters and electronic components of which the agent can click on one and also can move, add, or delete some components. The training goal is to click on the right interrupter or put the component in a suitable place. The observations from the environment are the image of the circuit and the case of the interrupters, and the reward is +1 for the right click on the right interrupter and perfect position of the component, −1 for the wrong click on the wrong interrupter and bad position, and 0 for nonclick, click in other place, or no movement.

To get the observations and actions, we can use critic value function representation to make the DQN agent approximate the reward. So, about our environment, the critic value function is a deep neural network with one input (image) and one output (perfect action).

5.2. Creating the Model. The options used to build the model are shown in Table 3.

5.3. Train Agent. To train our agent, we specify the training options using the following options:

We are going to run our program at most 8,000 episodes, with each episode lasting at most 800 time steps; then, we stop the training when the agent receives an average cumulative reward greater than −1,500 over the default window length of 15 consecutive episodes. At this level, the agent can open or close the tight interrupter or put one of the electronic components.

6. Results

In these experiments, we drilled for a total of 20 million frames and used a replay memory of 3 million mostly new frames. Our simulation includes two figures: the first was running an epsilon-greedy policy with an epsilon of 0.07 for 800 steps and starting of −4000 episode reward to get around −900, and the second was running an epsilon-greedy policy with an epsilon of 0.06 for 800 steps, but here, the total reward was starting of −4500 episode reward to get also around −900, as a shown in Figures 5 and 6.

Figures 5 and 6 show how the median total reward grows during training in our environment. The averaged reward plot is quite noisy, giving one the effect that the learning algorithm is not making constant progress. Another, more stable, metric is the policy's rating action-value function Q, which tools up an estimate of how much discounted reward the agent can secure by following its policy from any given state. We collect a steady set of states by operating a random policy before training starts and track the median of the maximum (the top for each state is possessed over the possible actions) presaged Q for these states.

Start with $Q_0(S, A, W)$for all (S, A)
  Initialize the state S
  For $t = 1$, N do (for each episode N)
    Simple action A, get next state (S')
    If $(S')$is terminal:
      Target = $R(S, A, S')$
      Sample new initial state (S')
      Else:
        target = $R(S, A, W) + \gamma \max Q_t(S', A', W')$
        $\varnothing_{t+1} \longleftarrow \varnothing_t - \alpha \nabla_{\varnothing} E_{S' \sim P(S' \vdots S,A)}^{a'} [(Q_{\varnothing}(S, A) - \text{target}(S'))^2]_{\varnothing=\varnothing_t}$
      $S \longleftarrow S'$
    End if
  End for

Algorithm 1: The algorithm of deep Q-learning.

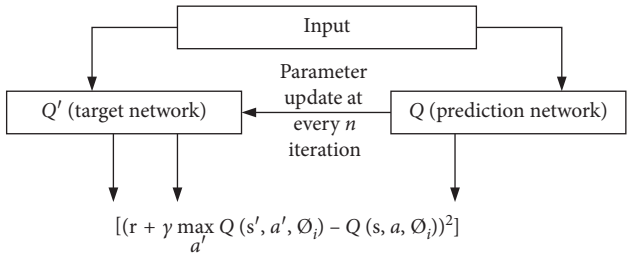

Figure 4: Estimating the target.

The epsilon starts with value 1 and decays every 800 steps with a rate of 0.987.

(i) We select random number [0, 1] n

(ii) If $n <$ epsilon, we choose a random action

(iii) Else, we choose action using our network

6.1. Visualizing the Value Function. Figure 7 shows a conception of the learned value function on the circuit sequent. This figure shows that the presaged value puts the components in the right position to close the circuit (point F1). The agent then closed all the interrupters of the circuit, and the portend value peaks when the graph appears (point F4). Additionally, the value falls to nearly its original value after the agent takes the suitable measurement (point F8). This method can pick up how the value function evolves for a reasonable ganglion sequence of events.

The first section of the results (Figure 8) is the analysis of the actions; here, basically, they are trying to show how many actions at each frame need to be taken in order to find the object; obviously, the less number of the actions shows the better efficiency. So, the proposed method as reported in 93% of the times finds the right actions in less than 5 actions at each frame in order to find the object, which is what we are going to see in Figure 9 that supports the previous claim because the number of right actions is more than that of other actions. This is basically saying that things we want are going to be found at each frame one after another without a comparable amount of other actions.

From Figure 9, we note that the right-click rate for the interrupter and the placement of electronic elements in the right place by the agent ranges from 91% to 94%, indicating that the system has been accurately and rightly learned, whereas the error rate by the agent is only 15% for the faulty click on the interrupter, putting the electronic elements in the wrong place, and pressure elsewhere. So, training the system well and obtaining satisfactory results depend on studying the problem situation well and selecting the variables used in the system properly and in proportion to what is required. For example, in Figure 5 when selecting $\alpha = 0.07$, the agent began collecting reward starting at −4000, whereas in Figure 6, $\alpha = 0.06$, the collection reward process did not begin until −4500. Therefore, we can conclude that having a high-precision, better-functioning, less-complex, and fast-paced system at the same time must select all system variables rightly and in proportion to what is required.

When we created the system environment, we have limited the agent's reward to a value equal to −100, which is the value by which we can say that it has learned well. Through Figures 5 and 6, the agent moved from −4,000 to −1,000 as in Figure 4 and from −4,500 to −1,000 as in Figure 6, which means that the agent collected enough rewards to get the target. To reinforce what we said earlier, in the bottom two images of Figure 7, we note that the agent has reached the goal that we want, which is the right put of the electronic elements and the right click on the interrupter. In addition, Figure 9 shows this, where we note that the rate of the right click and the right put is 94%, while the wrong click and the wrong put is only 7%.

Table 4 shows the comparison between different studies related to our work. In this table, we compare the presented result with 4 recent research articles on the implementation of DQN and RL techniques, and it shows the proposed result has an accuracy output between 17% and 99% in various methods that has a better consequence. Another literary review proposed by Mystakidis et al. [26] focused on the

Initialize memory M to capacity C
Initialize value function Q (action) with random weights
For episode = 1, N do (for each episode N)
Initialize state $S_1 = (X_1)$ (starting computer screen pixels) at the beginning of each episode $\varnothing_1 = \varnothing(S_1)$
//Pre-process and feed the computer screen (state S)to our DQN,
//which will regress the Q-values of all possible actions in the state.
For $t = 1$, T do
Chose an action using the epsilon-greedy policy.
//With the prospect epsilon, we chose a random action a and with probability 1-epsilon,
//chose an action that has a maximum Q-value, such as $A = \arg\max(Q(S, A, W))$
Execute action $A_t$in emulator and observe reword $R_t$and image X_{t+1}
Set $S_{t+1} = S_t$, A_t, $X_{t+1}$and pre-process $\varnothing_{t+1} = \varnothing(S_{t+1})$
Store the transition $(\varnothing_t, A_t, R_t, \varnothing_{t+1})$in M
Sample random mini-batch of the transitions $(\varnothing_i, A_i, R_i, \varnothing_{i+1})$from M
Set $Y_i = \begin{cases} R_i \text{ from terminal } \varnothing_{i+1} \\ R_i + \gamma \max_{a'}(\varnothing_{i+1}, A', W) \text{ from non} - \text{terminal } \varnothing_{i+1} \end{cases}$
figure the loss function $\mathscr{L} = (R + \gamma \max_{a'} Q(S', R', W') - Q(S, A, W))^2$
//which is just the squared difference between goal Q and predicted Q.
Do gradient descent with respect to our actual network parameters in order to reduce the loss function.
After every "N" iteration, copy our actual network weights to the goal network weights.
End for
End for

Algorithm 2: The DQN algorithm and experience replay.

Table 2: The development system information.

Component	Description
Programming language	Python 3.8.3 and LabVIEW 2017
Operating system	Windows 10, 64 bit
GPU	NVIDIA NVS 315
Library and framework	Jupyter Notebook
CPU	Intel(R) Xeon(R) CPU E5-2603 v4 @ 1.70 GHz
Memory	24 GB
Recommendation Modules	Deep reinforcement learning
Optimization algorithm	Model-free optimization

Table 3: The options used to build our model.

	Layers	Sizes	Normalization
Image observation	Image input layer	(84, 84, 4)	None
State of the interrupter and the electronic corposant	Image input layer	(1, 1)	None
	Fully connected layer	500	
	ReLU layer	—	
	Fully connected layer	400	
Action	Image input layer	(1, 1)	None
	Fully connected layer	400	
Output	Addition layer	—	—
	ReLU layer	—	—
	Fully connected layer	1	—

effectiveness of e-learning along with the factors and conditions that lead to climate change when using social virtual reality environments (SVREs) in distance higher education. The authors researched the cognitive, social, and emotional aspects of deep and meaningful learning (DML) and studied its weight at SVREs. The results suggested that the impact of DML weight on SVRE use could provide real experiences, simulations, and thoughtful participatory challenges as well as the positive impact of the DQN technique on our platform.

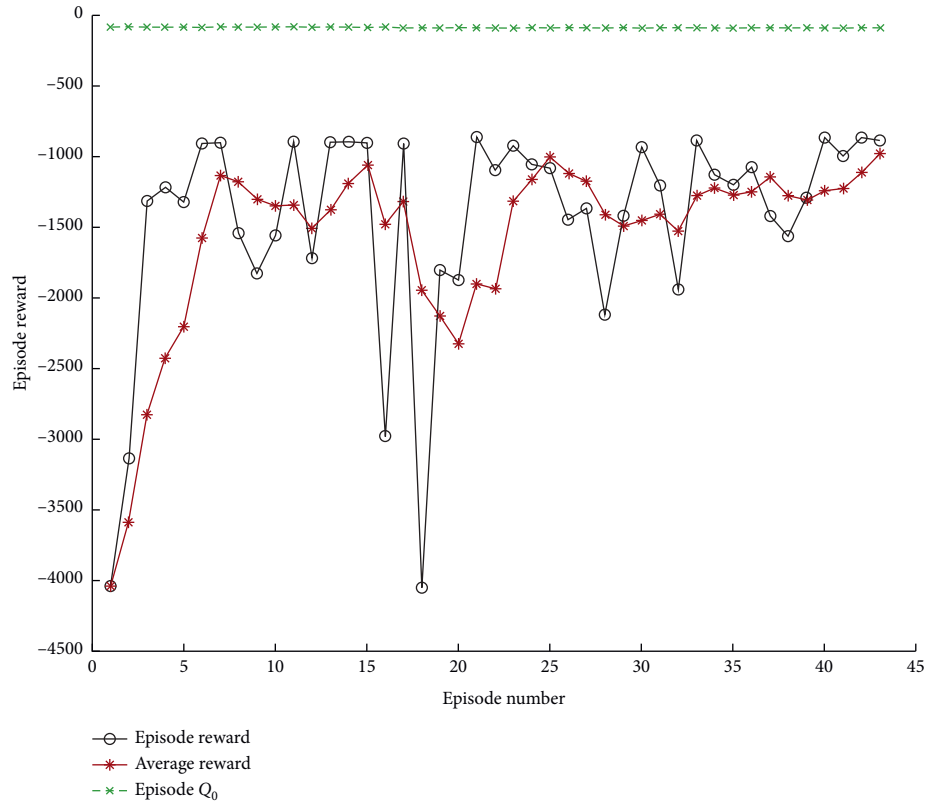

Figure 5: Episode reward for our environment with a DQN agent with an epsilon-greedy = 0.07.

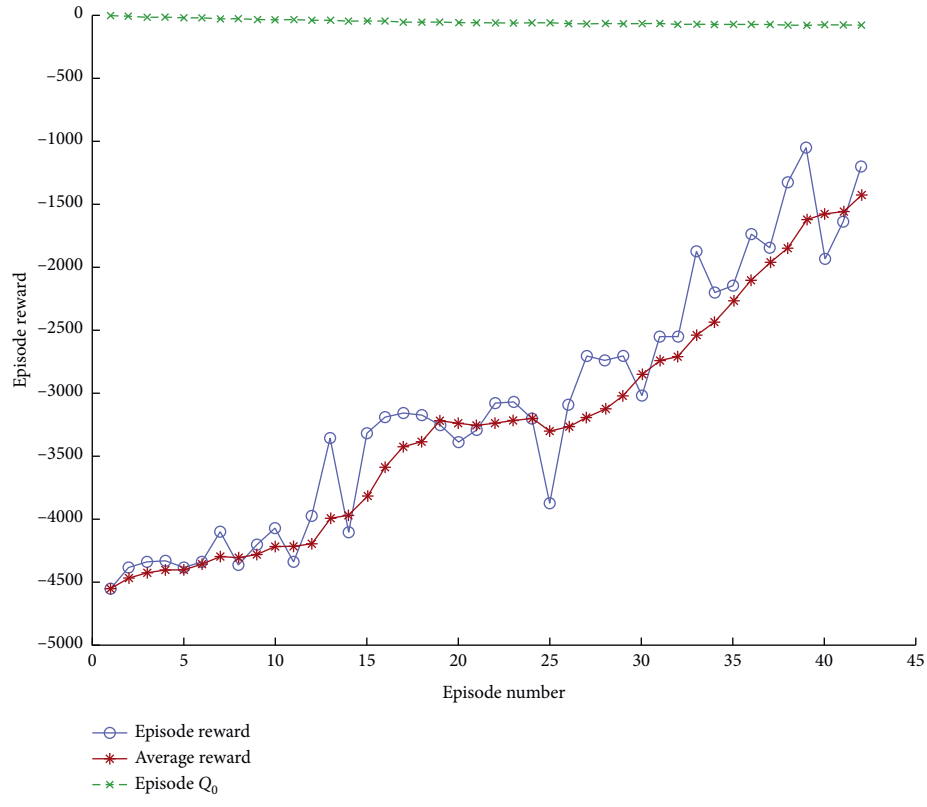

Figure 6: Episode reward for our environment with a DQN agent with an epsilon-greedy = 0.06.

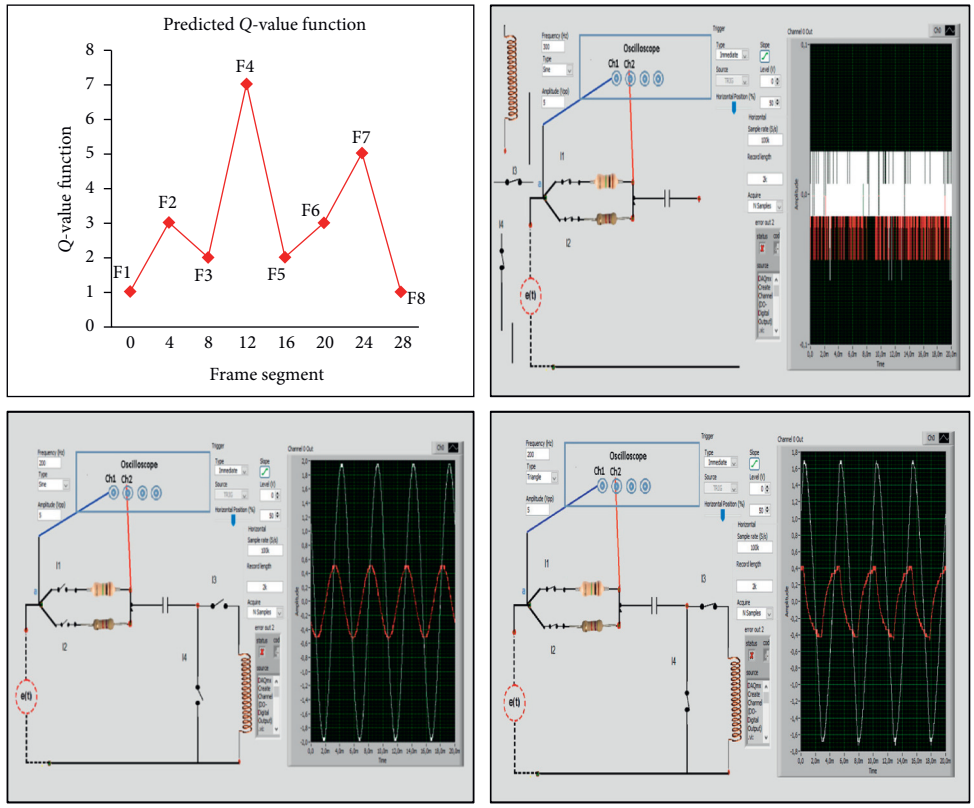

Figure 7: The left plot shows the presaged value function for 28 frames of the circuit sequent. The 3 screenshots match the frames labelled as F1, F7, and F3, respectively.

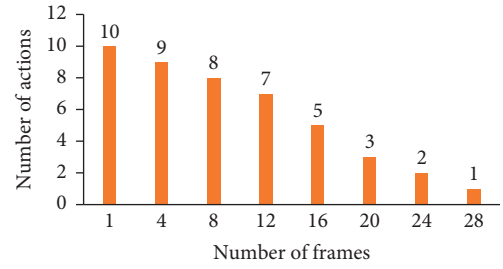

Figure 8: Number of actions per frame.

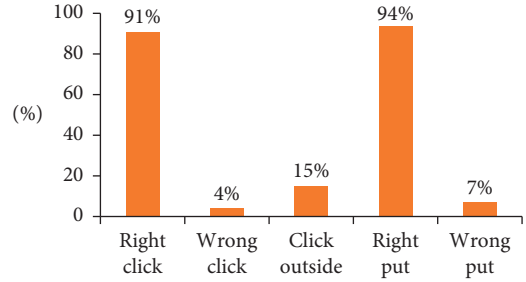

Figure 9: Select action.

Table 4: Comparison of accuracy with other studies available in the literature.

References	Location	Year	Method	Accuracy
El Fouki et al. [23]	Morocco	2017	DQN	0.99
Agrebi et al. [24]	France	2019	DQN	0.48
Shahbazi and Byun [25]	Jeju	2020	LR	0.2944
			FM	0.3125
			W&D	0.1758
			LinUCB	0.3747
			HLinUCB	0.2434
			DN	0.2657
			DDQN	0.2146
			DDQN + U	0.2824
			DDQN + U + EG	0.2118
			DDQN + U + DBGD	0.2327
This study	Morocco	2021	DQN	0.94

7. Conclusions

Reinforcement learning has become more common in recent years due to the wide spectrum of problems that can be solved through its use in control, industrial automation, robotics, health, economics, and many other areas. That is why so many researchers assert that enhanced learning will be one of the most important ways to access artificial general intelligence. This article was a starting point for learning the most important terms and approaches to the application of deep reinforcement learning in one part of the educational process to recommend and direct students in advance of doing their remote practical work according to their skills of each mouse or keyboard click per student. The first step is to understand and analyse the problem and frame the basic elements such as the agent, the environment, the quality of actions, possible situations, and reward. Then, the best approach that can solve this problem is chosen. Under each approach are many algorithms that differ in their performance and the type of problems they can solve.

Data Availability

The data used in this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] F. Ouatik, M. Raoufi, M. El Mohadab, F. Ouatik, B. Bouikhalene, and M. Skouri, "Modeling collaborative practical work processes in an e-learning context of engineering electric education," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 16, no. 3, pp. 1464–1473, 2019.

[2] M. A. Jallal, S. Chabaa, and A. Zeroual, "A new artificial multi-neural approach to estimate the hourly global solar radiation in a semi-arid climate site," *Theoretical and Applied Climatology*, vol. 139, no. 3-4, pp. 1261–1276, 2020.

[3] L. Tai and M. Liu, "Towards cognitive exploration through deep reinforcement learning for mobile robots," 2016, http://arxiv.org/abs/1610.01733.

[4] M. A. Jallal, A. E. Yassini, S. Chabaa, A. Zeroual, and S. Ibnyaich, "AI data driven approach-based endogenous inputs for global solar radiation forecasting," *Ingénierie des systèmes d information*, vol. 25, no. 1, pp. 27–34, 2020.

[5] A. El Gourari, M. Skouri, M. Raoufi, and F. Ouatik, "The future of the transition to E-learning and distance learning using artificial intelligence," in *Proceedings of the International Conference on e-Learning (ICEL)*, pp. 279–284, 2020.

[6] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[7] M. J. Matarić, "Reinforcement learning in the multi-robot domain," *Autonomous Robots*, vol. 4, no. 1, pp. 73–83, 1997.

[8] C. Breazeal and B. Scassellati, "Robots That Imitate Humans," *Trends in Cognitive Sciences*, vol. 6, no. 11, pp. 1–7, 2002.

[9] K. Hitomi, T. Shibata, Y. Nakamura, and S. Ishii, "Reinforcement learning for quasi-passive dynamic walking of an unstable biped robot," *Robotics and Autonomous Systems*, vol. 54, no. 12, pp. 982–988, 2006.

[10] D. Büchler, S. Guist, R. Calandra, V. Berenz, B. Schölkopf, and J. Peters, "Learning to play table tennis from scratch using muscular robots," 2020, https://arxiv.org/abs/2006.05935.

[11] M. Riedmiller, T. Gabel, R. Hafner, and S. Lange, "Reinforcement learning for robot soccer," *Autonomous Robots*, vol. 27, no. 1, pp. 55–73, 2009.

[12] N. D. Nguyen, T. Nguyen, and S. Nahavandi, "System design perspective for human-level agents using deep reinforcement learning: a survey," *IEEE Access*, vol. 5, pp. 27091–27102, 2017.

[13] T. F. G. Title, D. R. Learning, E. Telem, G. Mu, F. Advisor, and C. B. Mux, "Bachelor degree thesis," 2018.

[14] M. R. F. Mendonca, H. S. Bernardino, and R. F. Neto, "Simulating human behavior in fighting games using reinforcement learning and artificial neural networks," in *Proceedings of the 2015 14th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, vol. 0, pp. 152–159, Piauí, Brazil, November 2015.

[15] C.-S. Lee, M.-H. Wang, S.-J. Yen et al., "Human vs. Computer go: review and prospect [discussion forum]," *IEEE Computational Intelligence Magazine*, vol. 11, no. 3, pp. 67–72, 2016.

[16] S. Carta, A. Corriga, A. Ferreira, D. R. Recupero, and A. S. Podda, "A multi-layer and multi-ensemble stock trader using deep learning and deep reinforcement learning," 2020.

[17] S. Carta, A. Ferreira, A. S. Podda, D. Reforgiato Recupero, and A. Sanna, "Multi-DQN: an ensemble of deep Q-learning agents for stock market forecasting," *Expert Systems with Applications*, vol. 164, Article ID 113820, 2021.

[18] J. Wu, H. He, J. Peng, Y. Li, and Z. Li, "Continuous reinforcement learning of energy management with deep Q network for a power split hybrid electric bus," *Applied Energy*, vol. 222, pp. 799–811, 2018.

[19] A. Choudhary, "A hands-on introduction to deep Q-learning using OpenAI gym in Python," 2018, https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/.

[20] A. Nair, "Massively parallel methods for deep reinforcement learning," 2015.

[21] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI-18)*, pp. 3207–3214, New Orleans, LA, USA, February 2018.

[22] V. Mnih, "Playing atari with deep reinforcement learning," 2013.

[23] M. El Fouki, N. Aknin, and K. E. El Kadiri, "Intelligent adapted e-learning system based on deep reinforcement learning," in *Proceedings of the 2nd International Conference on Computing and Wireless Communication Systems (ICCWCS'17)*, pp. 1–6, Larache, Morocco, November 2017.

[24] M. Agrebi, M. Sendi, and M. Abed, *Deep Reinforcement Learning for Personalized Recommendation of Distance Learning*, Springer International Publishing, Berlin, Germany, 2019.

[25] Z. Shahbazi and Y. C. Byun, "Toward social media content recommendation integrated with data science and machine learning approach for e-learners," *Symmetry*, vol. 12, no. 11, pp. 1798–1822, 2020.

[26] S. Mystakidis, E. Berki, and J.-P. Valtanen, "Deep and meaningful e-learning with social virtual reality environments in higher education: a systematic literature review," *Applied Sciences*, vol. 11, no. 5, p. 2412, 2021.