IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 20, Issue 1, Ver. III (Jan.- Feb. 2018), PP 23-32
www.iosrjournals.org

Enhanced model-Free deep Q-Learning Based control

Samer I. Mohamed[1], Youssef Youssf[2]
[1]*(Computer Engineering Department, Faculty Of Engineering/ October University For Modern Sciences And Arts, Egypt)*
[2]*(Computer Engineering Department, Faculty Of Engineering/ October University For Modern Sciences And Arts, Egypt)*
Corresponding Author: Samer I. Mohamed

Abstract: *There are many challenges currently faced by the humanoid robot' models to achieve the objective or target planned for them. One of these challenges is the trade-off between the ability to accurately mimic the human body and uptime. While some of these models like ASIMO has very accurate degree to mimic the bipedal walking gait used by human due to high actuations, it consumes high power. Through our proposed model we introduce a model-free deep Q-learning algorithm (DQN) that doesn't simulate the bipedal walking gait used by human based on predetermined sequence or modelling via supervised learning. On counter, our proposed model learns from interactions/interfacing with the surrounding environment by applying actions on the robot and observing the reward from that action to make an under-actuated robot able to balance and walk forward, backwards, sideways, and rotate in place using only 4 actuators. It takes quite long time at the start of the learning process where robot is learning by trial and error but once learned it becomes able to perform and reach the target goal without being fall down. Results achieved from the proposed simulated model show better performance in terms of ability to reach target without being fall and corresponding power consumption.*

Keywords - *Deep Q-learning, Humanoid robots, Multilayer Perceptron Deep Neural Networks.*

Date of Submission: 19-01-2018 Date of acceptance: 17-02-2018

I. Introduction

Robotics and robot science becomes currently a core and building block for modern life, it found its way through different dimensions in our day to day actions. To accommodate with the high pace of current modern life needs, robotics R&D is in continuous improvement over the last decades. Despite this rapid rate of improvement, most robots designs are direct programmed, enabling them to only do few programmed or predefined and dedicated tasks with high cost [10]. To enable robots satisfy these modern life needs, artificial intelligence and learning algorithms subject to continuous development and improvement to enable robots from achieving their goal of ease the human day-to-day tasks. Although these robots can outperform humans on certain tasks, they cannot do any tasks other that what they were originally programmed to do. Therefore when we think of a better way to assist humans, what is better at interacting with an environment designed to serve humans, than a human and give the ability to robot to learn and build his knowledge base from the interactions with the environment. Through our proposed system, humanoid robot with under actuation is introduced as the human body fits to act well in a day-to-day environment which are already designed to be used by humans. By applying our system methods, the robot will be able to learn how to balance, detect its goal and walk towards it without having any predefined sequence, model or even supervised learning like other models exist currently either in the literature or industrial market. This will not limit our proposed model to perform specific set of defined tasks, but rather learn any task from interactions with surrounding environment. This makes our proposed model outperforms other existing models in the literature where learning algorithm of some of them based on supervised learning [2] that takes specific sequence of events to perform a task, while others like ATLAS [5] uses unsupervised learning with model-based learning. The proposed system of that intelligent humanoid robot's constituents are: the sensory system with a suitable processing unit, the lower body learning algorithm (DQN) that makes the robot able to learn how to balance while walking consists of two integrated blocks (Q-Learning and Neural Network [1]) which will be our main focus in this paper, and finally the Convolution Neural Network (CNN) that takes inputs from the camera to detect the goals in order to move towards them from one place to another. It is only in recent years that manufacturers are making robotics increasingly available and attainable to the public and there are four previous systems that are under the category of humanoid robots with artificial intelligence like ASIMO [2], NAO [3], DARWIN [4] and ATLAS [5].

II. Background

There are many different robotics models currently where we can illustrate some of them as relevant to our research like ASIMO [2], or Advanced Step in Innovative Mobility is a humanoid robot which recognizes and detects obstacles in the environment around it and finds the shortest path to a specific goal or target. NAO [3] is an autonomous humanoid robot, NAO has 25 Degrees of Freedom (DoF) that enables him to move and learn from the environment by trial and error. DARWIN-OP [4] is a small humanoid robot platform with reinforcement learning to learn how to walk and balance. ATLAS [5] is a humanoid robot which considered one of the few models that applied Deep Q-Network (DQN) to learn how to walk as humans. ATLAS uses the Q-learning algorithms to learn how to pick the optimized actions towards goal.The key value behind the Q-learning, is that it tries to get the state-action pair that makes the agent closer to the goal using not only the current data/state but data/states from the past to conclude the state in the future or the next state using the Q-value that maximizes specific evaluation function. And while Q-Learning objective is to find the optimum action to take towards the goal, the Neural Network helps to find the correlation between the input state or data gathered from the sensors and actions. This can be done by feeding the input sensor data to train the Neural Network to produce the optimum action in this non-linear problem for the robot infinite action space. Output from the neural network is back propagated afterwards using the gradient descent which is the derivative of evaluation function (difference between the output and the expected value from the labelled data or reward function). The main advantage of neural network that makes DQN best fit for solving our problem of robot control is its ability to learn the importance of particular features within a dataset [8]. Its main strength lies in that it operates on continuous data, meaning that even if the network has not been trained on a particular input, if the input is close to values it was trained on, then the output will be close to the output for the trained values or become more general and the other way around when the network is over-trained then it will become too specialized to detect specific features only and will not be able to detect other than what it is trained for.Our proposed system is different from the applied mentioned ones in many aspects, first the under actuation concept that results in lower cost and power consumption; and second, using model-free DQN algorithm. Our proposed learning algorithms are achieved based on reinforcement learning (Deep Q-Networks) and convolution neural network. Our proposed system for building a humanoid robot uses model-free DQN to learn how to balance and walk by applying optimized actions on the surrounding environment and getting reward in return that is mapped to these actions. The DQN algorithm selects these optimized action to optimize specific value function towards specific goal. The reminder of the paper is organized as follows: the experimental and computational details of the proposed system will be covered in section 2, section 3 will cover the testing results and experimental analysis for the proposed system and finally the conclusion for the overall proposed system will be concluded in section 4.

III. Proposed System Design

3.1: System overview:

The learning algorithm Deep Q Network (DQN) composed of two main parts the Q-learning [7] algorithm and the neural network [8]. Firstly, the Q-learning algorithm which is a specific case of Reinforcement Learning (RL) algorithm learns by trial and error through taking actions and observing the reward reserved based on that action, as visible in" (1)". Q (state, action) = R (state, action) + discount factor * Max Q (Next State, all Actions) (1)We can conclude from the above Q-learning equation that when the robot is in state "s (t)" and got an action "a (t)" that move it to state "s (t+1)" it receives a reward "r (t)" based on if that action is good or bad. The algorithm learns also to maximize the received reward r(t) which we can derive from the term [discount factor *max Q(s (t+1), a)]. On the other hand, there are other types of machine learning algorithms like supervised and unsupervised learning [9]. These learning algorithms require previous training data for the system to be trained, however in Reinforcement Learning (RL) algorithms, the learning algorithm learns by trial and error. RL is agent-centered where agent performs action on the surrounding environment and get either positive or negative reward based on the action and the reward function criteria R(t). The Q-Learning (QL) is special version of the RL algorithms, which uses the maximum function over the action value function Q(t) to choose the next actions, unlike other RL algorithms which use the summation function. One of the other main benefits of the QL algorithm, is that it absorbs and accommodates large search space using nonlinear function approximation methods like neural networks.

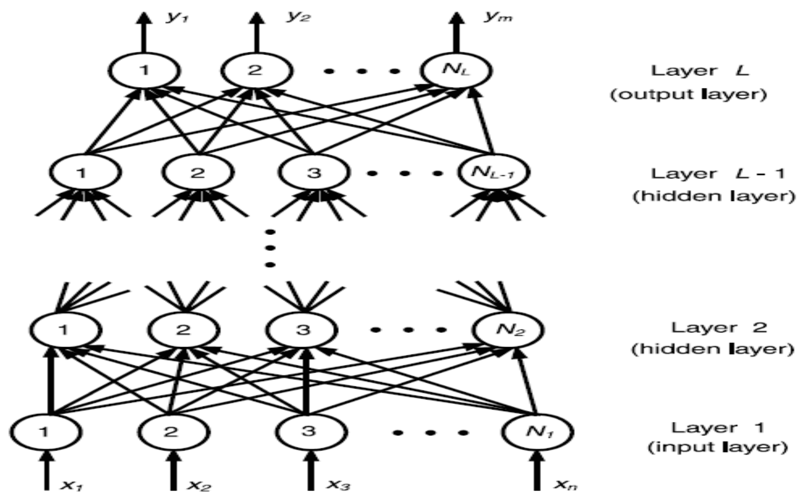


Fig. 1. Deep Neural Network with L Hidden layers

One of the main drivers to use Q-learning in our proposed model is because it is an off-policy learning algorithm that can learn by re-using experience generated from old policies which will be saved in the replay or historical memory or register. Secondly, the neural network in our system required by the RL algorithm acts as a look up table to store states and respective actions pairs. The robot search space is very large, as each change in any motor angle considered as a new state. Neural network acts as a value function approximation for the robot state, because the nonlinear problem of humanoid robots walking. The complexity of this non-linear problem related to how many parameters to be considered while making a walking robot like the robot's weight, inertia, balancing, PID control parameters and how large and changeable the environment around him, cannot be solved with linear value function approximation. We used a deep neural network [10] like in figure 1 to handle and solve this nonlinear problem via function approximation. Neural network in the DQN utilizes more than one hidden layer to be able to handle the walking and balancing process in the robot infinite search space.

3.2: System description:

The input data gathered from the robot sensors (camera and Inertia Measurement Unit (IMU)) constitutes the state vector that feed into the learning algorithm (DQN) of the proposed system [15]. This represents the robot current state and how it perceived the surrounding environment. The learning algorithm uses this data and process it to conclude the next state which will be mapped in terms of motor angles to move the robot towards the goal. The event sequence of this scenario is as follows:

1. Get the initial state S (t) of the robot from the sensory system.
2. DQN predicts an action A (t) based on the state S (t).
3. Convert the actions from the DQN algorithm to motor angles and execute them on the mechanical system (Robot).
4. Produce the new state S (t+1) to maximize the value function to reach the goal.
5. Evaluate this sequence of initial state S (t), action A (t) and new state S (t+1) through a reward function R (t).
6. The reward function gives the DQN algorithm a positive or negative reward based on whether the action was good or not.
7. DQN then learns how to predict better by training over and over achieving higher rewards.

IV. System description

The proposed DQN neural network structure is a 2 layers' feedforward neural network each layer contains 250 neurons, we used this architecture because it produces better results while training the robot, although 1-layer neural network is sufficient enough to approximate most learning problems.

3.1.1 Neural network hyper-parameters description

Network hidden layers count, There is direct proportion between the number of hidden layers, and the computation time, the probability to learn in large state space environments increases with moderate number of hidden layers. That is why we found through different iterations that 2 hidden layers is the best fit for our learning model.

1- Number of neurons per hidden layer, while there are different methods for determining the optimized number of neurons per each hidden layers. The best fit figure subject to trial and error method. There is also a direct proportion between the number of neurons and the probability of overfitting, also this extends to the computation time which is also directly proportion with the number of neurons. We reach the conclusion that 250 neurons in each hidden layer fit for our learning strategy and timelines.

2- Replay memory size, There is direct proportion between the replay memory size and the probability of learning because the capacity to store more learning and historical data increases. We select memory that fits for 1000 episodes.

3- Learning rate, this is very challenging parameter as increasing the learning rate, may lead to overshoot the goal. With infinite search space, we must select lower value for the learning rate so that it won't stuck in a local minimum.

4- Regularization factor, Critical to avoid overfitting which is irrelevant in our proposed design, as the search state space is infinite and it's hard to over fit the data or optimized action/goal.

5- Discount factor, should be in the range between 0 and 1. Shouldn't be 0 to avoid ignoring completely the future rewards, and shouldn’t be 1 to ensure the value of the future actions being considered is decreasing over time.

3.1.2 State representation:

The state representation helps to identify the state of the robot at each step as shown in figure.2.

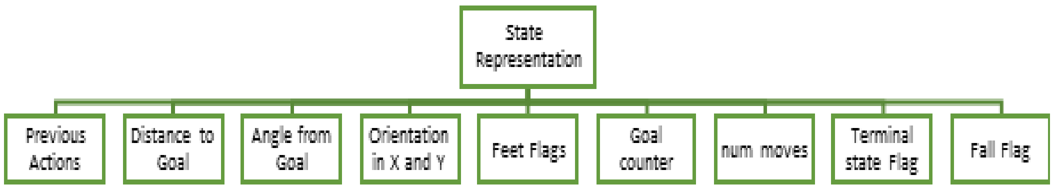


Fig. 2. Proposed model State representation

The real robot sensory (Camera, and Inertial measurement unit (IMU)) inputs (Distance to goal, Orientation values, and angle from goal) are mapped to the Q-Learning state representation. The detailed description of these state elements can be described as follows:

[1] **Previous Actions**, refers to the last 10 actions considering action as an array with 6 elements such that each element represents a motor out of 6 motors. Motor angle represented by a value of -1, 0 or 1 to control the motor state either to increase, maintain or decrease the speed/state.

[2] **dist_to_goal_flag**, flag that indicates how close the robot to the goal in meters. It is binary encoded relative to a range of distances. i.e. if the distance to goal is > 0.25 m then dist_to_goal_flag = [1 0 0 0] else if the distance to goal is >0.20 m then dist_to_goal_flag = [0 1 0 0] and so on until dist_to_goal_flag = [0 0 0 0] which means that the robot has reached the goal

[3] **angle_away_from_goal_flag**, indicates if the robot is facing the goal or there is a deviation in the angle between the robot and the goal which is the orientation around the Z-axis. The angle value is first rounded then it is encoded to a 9-bit binary number. i.e. 360 will be [101101000].

[4] **Orientation_X_Flag and orentation_Y_flag,** are the orientation of the robot around the X-axis and the Y-axis respectively. Orientation X indicates the inclination of the robot to the front or the back. Orientation Y indicates the inclination of the robot to one of its sides. Orientation X and Y values are binary encoded into 8 bits.

[5] **Feet_X_flag**, is a flag to indicate the side distance between the 2 feet of the robot. The separation between the 2 feet is encoded into 4-bit binary number. With the max separation is 40 cm and the minimum separation is 6 cm.

[6] **Feet_Y_flag**, indicates which foot is in the front and the distance between the 2 feet. The feet y value is encoded in 6 binary bits. I.e. if the feet y value is > 20 cm and the right foot is in the front then feet_Y_flag = [1 0 0 0 0 0] if the left foot was in the front then feet_Y_flag = [0 1 0 0 0 0].

[7] **Feet_Z_flag**, indicates which foot is on the ground and which foot is not. It is encoded in 2 binary bit numbers. I.e. if the right foot is on the ground and the left foot is above the ground then feet Z flag will be wither 0 or 1 where 1 indicates that a foot is above the ground.

[8] **Goal_counter_state**, indicates how many iterations the robot managed to stay above the goal and maintain its stable state. This number is encoded in 5 binary bits. Because the max number of iterations the robot need to maintain above the goal is 20.

[9] **Num_moves_state**, is a counter that indicates how many iterations the robot took to reach a terminal state also it is an indication for the time taken for the robot to reach the terminal state as each action take 0.1 second so maximum time for an episode is 0.1*500 iteration = 50 second. This number is encoded in 9 binary bits. Because the max number of iterations in our state is 500.

[10] **Terminal_flag**, is a 1-bit binary flag indicates weather the robot has reached a terminal state or not. The terminal states are 3 states, first state is when robot reaches the goal, second when robot reaches the max number of iterations allowed which is 500, and third state is when robot loses its balance and falling. In any of these 3 states the terminal flag is sit to 1.

3.1.3 Reward function representation

The reward function gives the robot an insight of how much each action has been taken, was good or bad which is the main idea behind the reinforcement learning algorithm. We are using some methods to judge the result of each action taken by the robot [12] as shown in figure 3. Based on the time taken for the robot to reach a terminal state, the distance between the robot and the goal, and the angle between the robot and the goal, whether the robot reached the goal or not, whether the robot close his legs or not and whether the robot fall or not. All rewards are weighted and summed then normalized so the maximum reward is 1 and the minimum reward is -1.

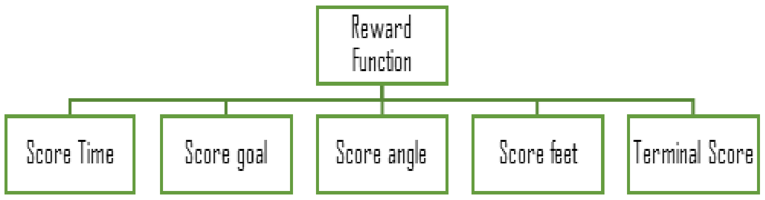


Fig. 3. Proposed model reward function

[1]. **Score Time**, is the time reward to identify how much time the robot needed to reach the goal. It starts with a big value at the beginning and decays gradually over time which is the number of iterations till it reaches zero at maximum number of iterations of 500 iteration per training episode. This gives the robot the sense of time so he could reach the goal in a minimum number of iterations to maximize the incoming reward.

[2]. **Score goal**, is the distance between the robot and the goal in meters. The robot needs to minimize the distance to goal there for reaching the goal to maximize its incoming reward. We use equation (3) to calculate the score goal and weight its score. Target_init is the initial distance to goal in the beginning of the episode. Goal_dist is the calculated distance to goal from the simulator at each step. The constant 50 is the weight of this reward in the total reward.
score_goal = ((target_init - goal_dist) / target_init) * 50 (3)

[3]. **Score angle**, is the angle away from goal in degrees the robot needs to orient himself towards the goal so he could move towards the goal. We use equation (4) to identify the angle between the robot and the goal. The constant 25 is the weight of this reward in the total reward.score_angle = ((360 - angle_away_from_goal_value) / 360) * 25 (4)

[4]. **Score feet**, is the separation between the 2 feet. The robot has to keep his feet separated to keep his balance and not to fall. The robot took - 0.1 (normalized reward) if he closed his legs.

[5]. **Terminal flag**, the robot reaches terminal state at 3 possibilities at each possibility he receives a

different reward. The 3 states are reaching the goal, reaching the maximum number of iterations of 500 iterations per episode or falling on the ground. If he reached the goal the robot take a maximum normalized reward of 1 else if he falls on the ground or he couldn’t reach the goal before the end of the episode he receives a minimum normalized reward of -1. If he reached the goal at the end of the episode he receives 1 reaching the goal reward override the maximum number of iteration reward. All these reward are summed to 100 then normalized from -1 to 1.

3.1.4 Evaluation function representation:

We use an evaluation function to evaluate the robot behavior based on the average it gets in an evaluation period of 10 episodes. The evaluation takes place every 20 thousand iterations to ensure there is a change in the behavior. The evaluation function runs the robot learning for 10 episodes and average the score of these 10 episodes then saves the neural network parameters that were responsible for this reward as the best network. If there is a best network saved before the process compares the reward of the two networks, if the new best network is higher in score it replaces the old best network. This allows us to have the best network if we want to start leaning again from that state or if we want to run it as a standalone network to control the robot dynamics.

3.1.5 Experience Replay

Experience Replay is a method we proposed with the DQN algorithm to make the learning more efficient. The value of the experience replay is to handle the online learning algorithm which learns step by step where each state represented by the steps and the actions the robot is executing. This makes each transition correlated to the previous transitions which will lead the neural network to fail in generalizing and learning the problem of humanoid walking. Experience replay offer a solution to this problem by saving the transitions in a replay memory of like 1 million transition capacity then picking a mini batch from the replay memory with as small number of random transitions i.e. 32 transitions. Each transition in the replay memory consists of the current state, the action taken from that state, the reward received from that state and action pair and the next state reached after performing that action i.e. {S(t), A(t), R(t), S(t+1)}.
The process goes as follows

- The robot starts to do random actions for the first 50 thousand iterations. We chose 50K based on our testing to enable the robot to have enough time to explore the world around it.
- These random transitions are saved in the replay memory.
- The learning starts after these number of iteration we sample a mini batch of 32 random transition from the replay memory to break the correlation.
- Feed that patch to the neural network to learn from it.
- Even after start learning transitions are still being saved into the replay memory replacing old transitions if the replay got full.
- We keep repeating this process till the algorithm converges.

V. System Design And Implementation

The proposed DQN algorithm pseudo code and the flowchart can be described as follows in figures 4 and 5. The pseudo code shows how the DQN algorithm repeats the iterations as long as robot not fall and starts from state S(t) with random action A(t) and observes the new state S(t+1) and reward received R(t) to maximize specific evaluation function Q(t). The main idea of our proposed system is that it learns to walk from only exploring randomly by doing random actions then getting a reward based on that action, so we taught a neural network how to walk to a goal in front of the robot that is 0.25 meters away using normal random exploration, then we make another new generation that explores from the previous generation neural network by doing feedforward and get the predicted best action from that learnt network and also keeping 1% random exploration. This allows the new neural network to learn not from random actions but from a learnt network which makes the task of the new generation easier and also allows the new generation to learn more things that the previous generation couldn’t learn in other words every new generation is evolving above the previous generation. This speeds up the learning process and increases the optimization and accuracy of the learning process to achieve the goal/target quickly compared to other models in the literature. We will show in the testing section that we The proposed DQN algorithm pseudo code and the flowchart can be described as follows in figures 4 and 5. The pseudo code shows how the DQN algorithm repeats the iterations as long as robot not fall and starts from state S(t) with random action A(t) and observes the new state S(t+1) and reward received R(t) to maximize specific evaluation function Q(t). The main idea of our proposed system is that it learns to walk from only exploring randomly by doing random actions then getting a reward based on that action, so we taught a neural network how to walk to a goal in front of the robot that is 0.25 meters away using normal random exploration, then we make another new generation that explores from the previous generation neural network by doing feedforward and get the predicted best action from that learnt network and also keeping 1% random exploration. This allows the new neural network to learn not from random actions but from a learnt network which makes the task of the new generation easier and also allows the new generation to learn more things that the previous generation couldn't learn in other words every new generation is evolving above the previous generation. This speeds up the learning process and increases the optimization and accuracy of the learning process to achieve the goal/target quickly compared to other models in the literature. We will show in the testing section that we managed to teach a network to walk to goals on its side at angle 45 and on different distances from that network that only reached a goal that is in front of it and on a distance of 0.25 meters.

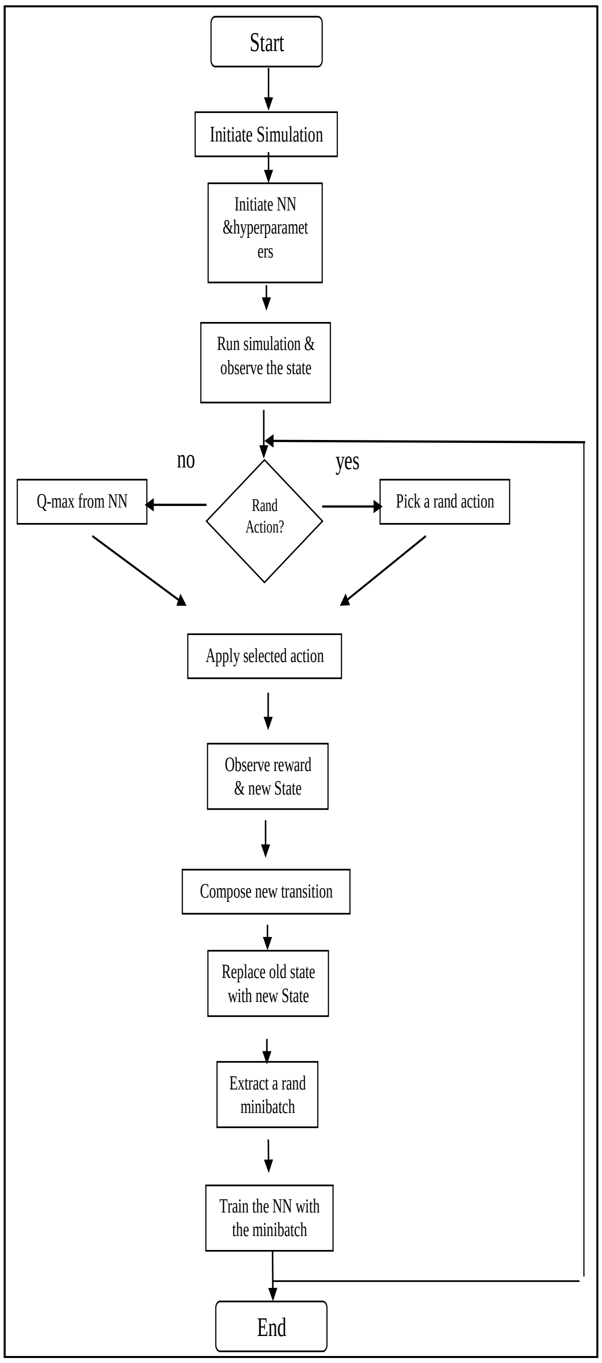
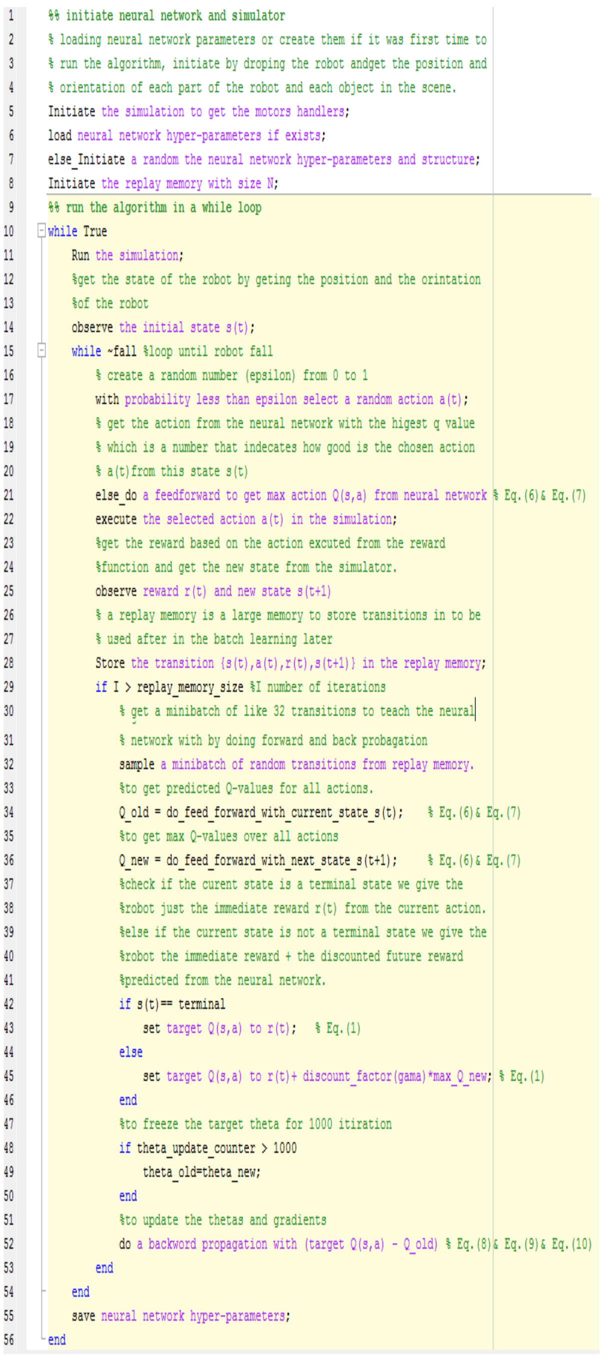

Start
Initiate Simulation
Initiate NN &hyperparameters
Run simulation & observe the state
no
yes
Q-max from NN
Rand Action?
Pick a rand action
Apply selected action
Observe reward & new State
Compose new transition
Replace old state with new State
Extract a rand minibatch
Train the NN with the minibatch
End

Fig.4. Proposed model flow chart



```
%% initiate neural network and simulator
% loading neural network parameters or create them if it was first time to
% run the algorithm, initiate by droping the robot andget the position and
% orientation of each part of the robot and each object in the scene.
Initiate the simulation to get the motors handlers;
load neural network hyper-parameters if exists;
else_Initiate a random the neural network hyper-parameters and structure;
Initiate the replay memory with size N;
%% run the algorithm in a while loop
while True
    Run the simulation;
    %get the state of the robot by geting the position and the orintation
    %of the robot
    observe the initial state s(t);
    while ~fall %loop until robot fall
        % create a random number (epsilon) from 0 to 1
        with probability less than epsilon select a random action a(t);
        % get the action from the neural network with the higest q value
        % which is a number that indecates how good is the chosen action
        % a(t)from this state s(t)
        else_do a feedforward to get max action Q(s,a) from neural network % Eq.(6)& Eq.(7)
        execute the selected action a(t) in the simulation;
        %get the reward based on the action excuted from the reward
        %function and get the new state from the simulator.
        observe reward r(t) and new state s(t+1)
        % a replay memory is a large memory to store transitions in to be
        % used after in the batch learning later
        Store the transition {s(t),a(t),r(t),s(t+1)} in the replay memory;
        if I > replay_memory_size %I number of iterations
            % get a minibatch of like 32 transitions to teach the neural
            % network with by doing forward and back probagation
            sample a minibatch of random transitions from replay memory.
            %to get predicted Q-values for all actions.
            Q_old = do_feed_forward_with_current_state_s(t);     % Eq.(6)& Eq.(7)
            %to get max Q-values over all actions
            Q_new = do_feed_forward_with_next_state_s(t+1);      % Eq.(6)& Eq.(7)
            %check if the curent state is a terminal state we give the
            %robot just the immediate reward r(t) from the current action.
            %else if the current state is not a terminal state we give the
            %robot the immediate reward + the discounted future reward
            %predicted from the neural network.
            if s(t)== terminal
                set target Q(s,a) to r(t);   % Eq.(1)
            else
                set target Q(s,a) to r(t)+ discount_factor(gama)*max_Q_new; % Eq.(1)
            end
            %to freeze the target theta for 1000 itiration
            if theta_update_counter > 1000
                theta_old=theta_new;
            end
            %to update the thetas and gradients
            do a backword propagation with (target Q(s,a) - Q_old) % Eq.(8)& Eq.(9)& Eq.(10)
        end
    end
    save neural network hyper-parameters;
end
```

Fig 5. Proposed model Pseudo code

VI. System Testing

The following results are the testing results of the evolving generations and robot learning, where we took the neural network parameters and thetas from the successful agent, and used it to teach the other agents (neural networks) to walk so it could improve the behavior of the first agent. Instead of using random exploration we explored from the network that learnt already by doing feed forward with the current state. Also, we still have that 1% totally random to give it a chance to explore and ensure that it won't over fit one solution. This method let us teach the robot how to reach a goal on 1 meter from a network that only learnt to walk 0.25-meter distance and 2 other goals on his sides at an angle of 45 degree and 1.4-meter distance. In each episode, the algorithm pick one random goal for the robot reach. We tested this idea and the results were very good as the robot managed to reach the 3 goals in an acceptable number of moves. Here we reached an average normalized score (which is the total reward received at the end of each episode) of 0.5 over 7K episode. Also the increasing in the q-values means that the algorithm is predicting a high discounted reward at the end of each episode as shown in figure 6.

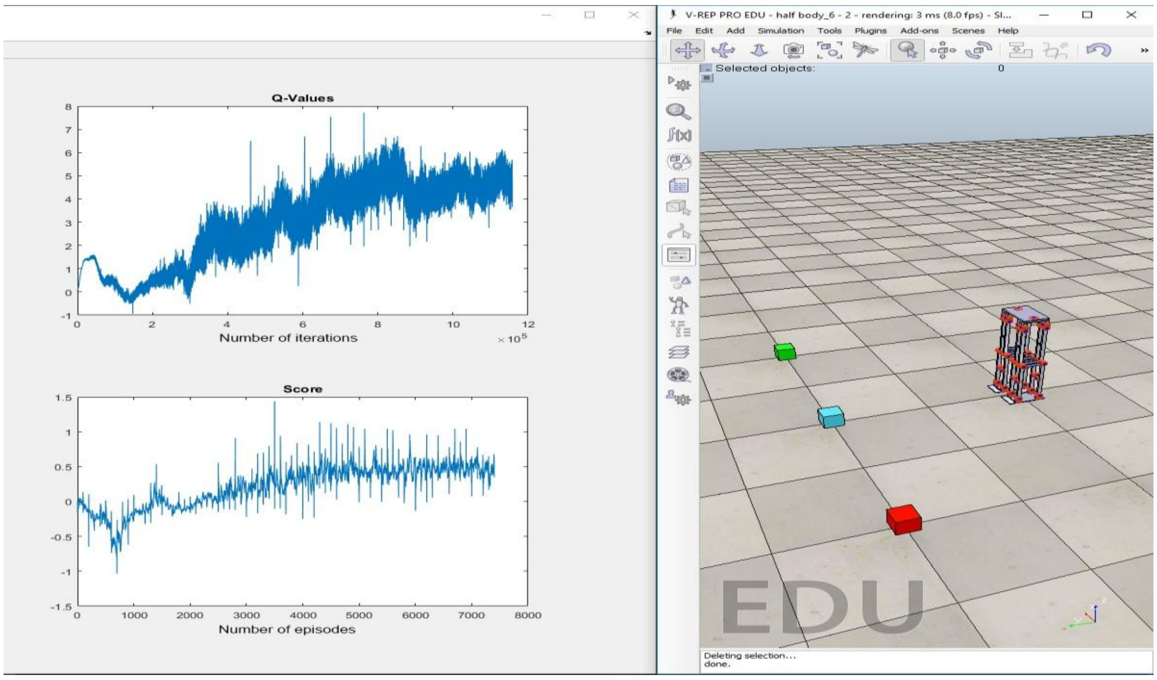

Q-Values
Number of iterations
× 10^5
Score
Number of episodes
V-REP PRO EDU - half body_6 - 2 - rendering: 3 ms (8.0 fps) - Sl...
File Edit Add Simulation Tools Plugins Add-ons Scenes Help
Selected objects: 0
EDU
Deleting selection...
done.

Fig 6. Testing Results and error function from enhanced DQN learning algorithm

In the figure 7, the robot managed to rotate and reach the red goal with an average reward of 0.5 normalized reward over 7K episode. At the same time, we were running the robot on 2 goals one in front of it and the other behind it. We have done that by reversing the actions the robot does to walk forward and make him move backward as his knee mechanism is symmetric on the 2 sides.

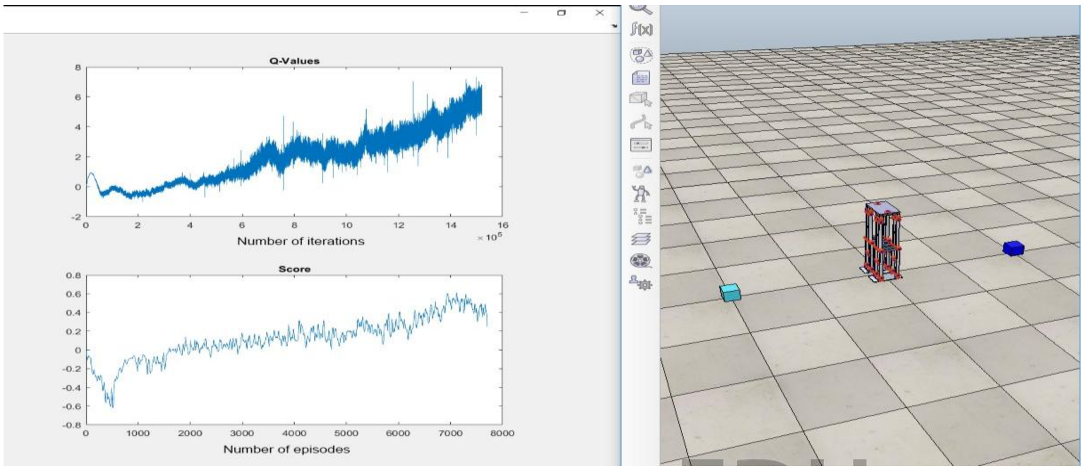
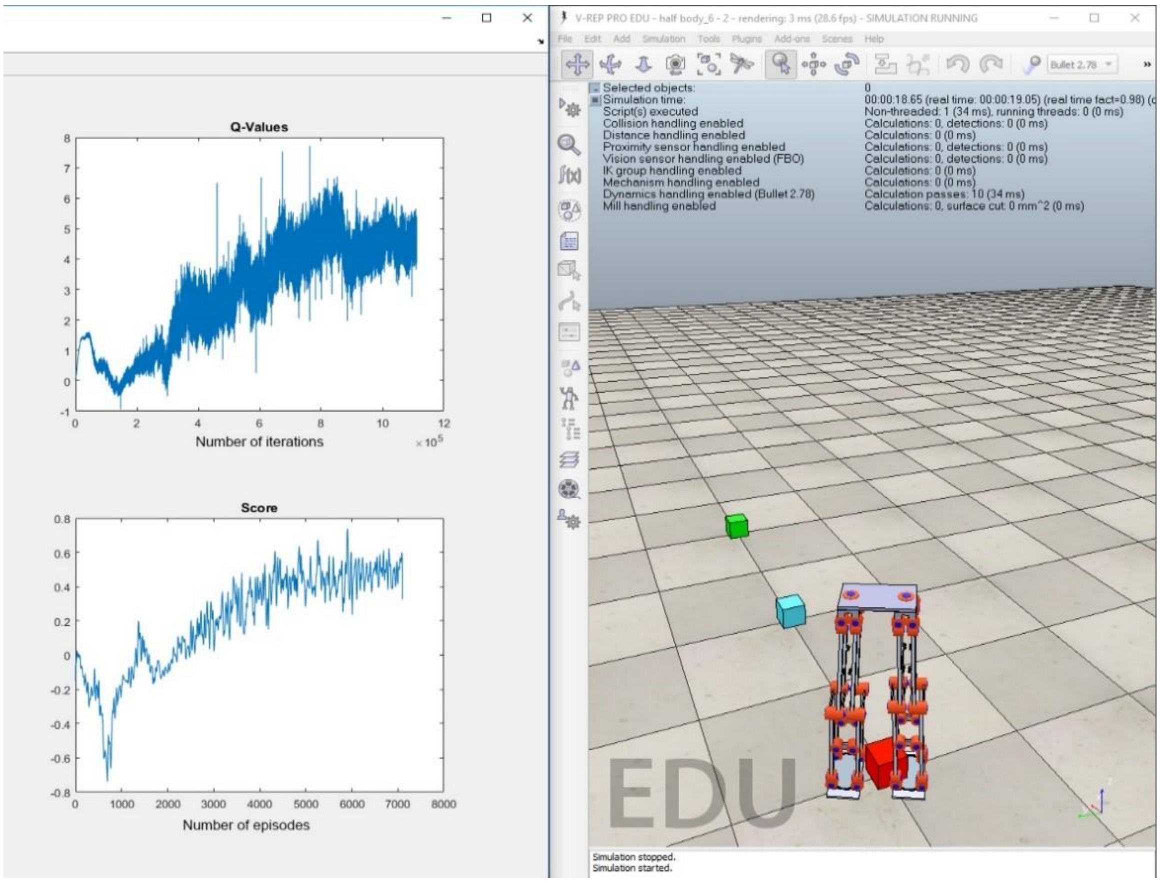


Fig 7. Testing for side goals

The algorithm gives the robot one of the 2 goals to reach randomly where the robot differentiates between the 2 goals by the angle away from goal. The robot managed to reach both goals perfectly with an average score of 0.6 normalized reward after 7.5K episode as shown also in figure 8.



```
Selected objects:                                 0
Simulation time:                                  00:00:18.65 (real time: 00:00:19.05) (real time fact=0.98) (
Script(s) executed                                Non-threaded: 1 (34 ms), running threads: 0 (0 ms)
Collision handling enabled                        Calculations: 0, detections: 0 (0 ms)
Distance handling enabled                         Calculations: 0 (0 ms)
Proximity sensor handling enabled                 Calculations: 0, detections: 0 (0 ms)
Vision sensor handling enabled (FBO)              Calculations: 0, detections: 0 (0 ms)
IK group handling enabled                         Calculations: 0 (0 ms)
Mechanism handling enabled                        Calculations: 0 (0 ms)
Dynamics handling enabled (Bullet 2.78)           Calculation passes: 10 (34 ms)
Mill handling enabled                             Calculations: 0, surface cut: 0 mm^2 (0 ms)
```

```
Simulation stopped.
Simulation started.
```

Fig 8. Testing results for backward/forward goals

VII. Conclusions

Many robots currently designed to perform a specific function or dedicated for specific application that limits its usability in different fields. In many cases robots use lots of motors to mimic the human gait which results in a very short online time and enormous power consumption with a costly model. A need for innovating more generic/broad robot that is able to learn from interactions with surrounding environment based on model-free learning drives our proposed model. The uniqueness for the proposed model is not being limited to specific application or domain but can learn and adapt to any domain of applications which makes it fit to perform a wide variety of distinct tasks. Our proposed system outperforms other systems through using an under-actuated model with less motors as possible without impacting its functionality, this will dramatically lower the cost and power consumption. And second it the ability to use the Deep Q-Learning algorithm that used in very limited number of current models like ATLAS which is model-based while our approach is model free to enable the robot to walk and balance without any pre-defined sequence of learning.

References

[1] L. Bottou , " Large-Scale Machine Learning with Stochastic Gradient Descent " , NEC Labs America, Princeton NJ 08542, USA
[2] N. Srivastava, G. Hinton , A. Krizhevsky and I. Sutskever , " Dropout: A Simple Way to Prevent Neural Networks from Overfitting" , Journal of Machine Learning Research 15, Canada , June 2014 .
[3] MathWorks, (2017). Vision Toolbox: User's Guide (R2017a). Retrieved June 15, 2017 from https://www.mathworks.com/help/vision/examples/object-detection-using-deep-learning.html
[4] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research, 47:253–279, 2013.
[5] Marc G. Bellemare, Joel Veness, and Michael Bowling. Bayesian learning of recursively factored environments. In Proceedings of the Thirtieth International Conference on Machine Learning (ICML 2013), pages 1211–1219, 2013.
[6] George E. Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. Audio, Speech, and Language Processing, IEEE Transactions on, 20(1):30 –42, January 2012.
[7] Matthew Hausknecht, Risto Miikkulainen, and Peter Stone. A neuro-evolution approach to general atari game playing. 2013.

[8] Nicolas Heess, David Silver, and Yee Whye Teh. Actor-critic reinforcement learning with energy-based policies. In European Workshop on Reinforcement Learning, page 43, 2012.

[9] Takubo, T., Inoue, K., Arai, T.: Pushing an Object Considering the Hand Reflect Forcesby Humanoid Robot in Dynamic Walking, Proceedings of the 2005 IEEE International Conference on Robotics and Automation (2005).

[10] Soft Bank Robotics. (2016, Nov. 20). WHO IS NAO? Available: https://www.ald.softbankrobotics.com/en/cool-robots/nao

[11] S. Yi, B. Zhang, D. Hong and D. Lee , "Online Learning of a Full Body Push Recovery Controller for Omni-directional Walking", 2011 11th IEEE-RAS International Conference on Humanoid Robots Bled, Slovenia, October 26-28, 2011

[12] "DARPA Robotics Challenge", Defense Advanced Research Projects Agency, 2015. [Online]. Available: http://www.theroboticschallenge.org/. [Accessed 16 November 2016].

[13] "RASPBERRYPI 3 MODEL B ", [Online], [Accessed 10 April 2017] Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[14] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, London, England: The MIT Press, 2012.

[15] V. Mnih1, K. Kavukcuoglu1, D. Silver and A. Rusu1, "Human-level control through deep reinforcement learning " , Nature Vol. 518 , February 26 , 2015 .

[16] Volodymyr Mnih. Machine Learning for Aerial Image Labeling. PhD thesis, University of Toronto, 2013.

[17] Andrew Moore and Chris Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. Machine Learning, 13:103–130, 1993.

[18] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th International Conference on Machine Learning (ICML 2010), pages 807–814, 2010.

[19] Jordan B. Pollack and Alan D. Blair. Why did td-gammon work. In Advances in Neural Information Processing Systems 9, pages 10–16, 1996.

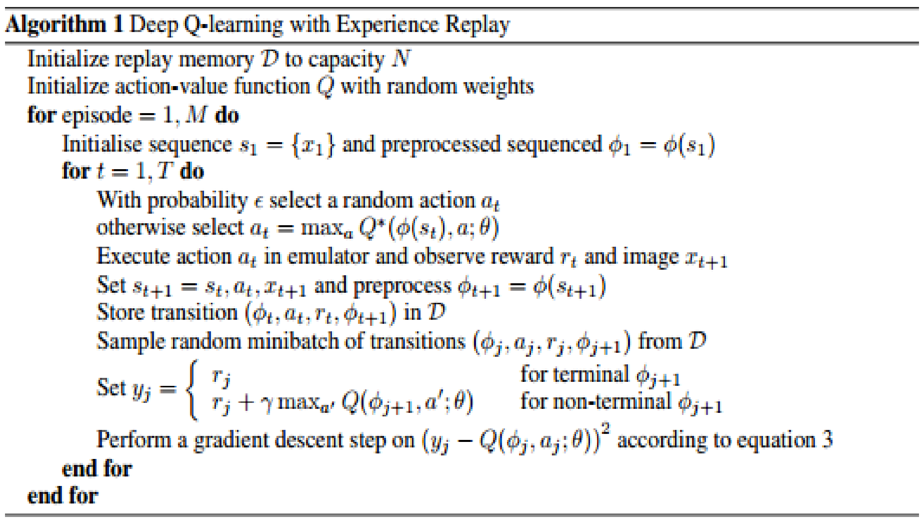
[20] Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In Machine Learning: ECML 2005, pages 317–328. Springer, 2005.

[21] Brian Sallans and Geoffrey E. Hinton. Reinforcement learning with factored states and actions. Journal of Machine Learning Research, 5:1063–1088, 2004.

[22] Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala, and Yann LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In Proc. International Conference on Computer Vision and Pattern Recognition (CVPR 2013). IEEE, 2013.

Apendecies

Pseudocode for Deep Q-Learning as implemented in Playing Atari with Deep Reinforcement Learning



Algorithm 1 Deep Q-learning with Experience Replay

```
Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights
for episode $= 1, M$ do
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ do
        With probability $\epsilon$ select a random action $a_t$
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action $a_t$ in emulator 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 $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for
```