1

Comparison of Q-learning and Deep Q-learning to solve the mountain car problem

Michal Zdrojewski, Oliver Williams, Tim Verheijen, Pedro Tan and Yashjit Gangopadhyay
Vrije Universiteit Amsterdam

Abstract

Reinforcement learning (RL) has undergone exciting developments in recent times, with the application of deep learning algorithms to existing RL techniques greatly expanding the scope of problems that can be solved. In this paper we discuss applicability of the classical RL algorithm, Q-learning and its more sophisticated counterpart, deep Q-learning in context of a simple OpenAI Gym MountainCar-v0 problem. In addition we compare and contrast these algorithms discussing some of their limitations. We report that even for a simple problem and with a less rigorous hyperparameter search strategy, deep Q-learning outperforms Q-learning in terms of scoring and learning time.

Keywords: Reinforcement Learning, Q-learning, Deep Q-learning

1. Introduction

Reinforcement learning (RL) describes a subset of machine learning (ML) whose objective is to maximise rewards while performing tasks. It significantly differs from the two other main branches of ML: supervised and unsupervised learning. Supervised learning uses known outputs to establish a set of model parameters to consistently achieve correct input-output pairs. Unsupervised learning is an exploratory technique that seeks to find relationships within input data without prior knowledge of labels and desired outputs. Both require pre-existing data sets and learning is performed offline. In contrast, an RL agent can be initialised without a prior model of the world and learn while performing tasks based on rewards, which is a metric of performance defined by the environment [1]. By observing states, actions and rewards, it updates its parameters to develop a strategy or policy on how to act in the environment. As this approach to learning is general and model-free, it is being increasingly applied to learn complex tasks without clear analytical solutions, such as robotic control and playing games, with clear demonstrations of success in recent years [2].

OpenAI Gym is an open-source environment for development and application of RL algorithms. It contains a repository of game environments of varying complexity, each with reward parameters to optimise and clearly defined metrics for 'solving' an environment. The objective of this work is to compare the performance of two RL algorithms, Q-learning and Deep Q-learning, at learning a classic control task using the OpenAI Gym MountainCar-v0 game.

2. Task overview

2.1 Game description and reward functions

MountainCar-v0 is a popular problem in reinforcement learning often used to evaluate RL algorithms [3, 4]. It is a 2-dimensional game in which car (agent) is placed between two hills of differing heights. The goal of the game is to reach the finish line on the highest hill as fast as possible. However, the car does not have enough power to drive straight uphill, and requires swinging left and right to build momentum.

The game implements a negative reward scheme: for every frame that the car does not reach the goal. it receives a reward of -1, and only when it reaches the top it receives a reward of 0. The episode is terminated when the goal is reached or after 200 steps. Therefore, the score function can simply be modelled as:

$$Score = -m \tag{1}$$

Where m corresponds to the number of steps taken to complete the episode and $0 < m < 200$. As documented in OpenAI gym, the game is considered solved when the car completes 100 episodes with an average score above -110.

2.2 State and action spaces

Game states are represented by 2-dimensional vectors containing the position x and velocity v, where $-1.2 \leq x \leq 0.6$ and $-0.07 \leq v \leq 0.07$. The agent begins at a point $-0.6 \leq x \leq -0.4$ and $v = 0$ (Fig. 1). To the left at ($x = -1.2$) there is an inelastic barrier and to the right ($x = 0.5$) is the flag signalling the goal. The agent has three discrete actions: no action, move left, and move right. Note that 'no action' can mean either idling or moving with momentum.

3. Implementation

3.1 Q-learning

Q-learning is a model-free reinforcement learning algorithm developed by Chris Watkins in 1989 [5]. This algorithm requires a representation of discrete states and actions and assigns a Q-value to each state-action combination. These values allow the agent to make decisions based on the quality of actions for the current state and expected future rewards. Q-values are stored in a look-up tensor known as a Q-table. All entries are initialised at zero or around zero with dimensions equal to the discrete observation $\times$ action space. At each step after taking an action a from state s, the agent receives a corresponding reward $r(s, a)$, a new state s' and the Q-table is updated as in equation (2) [6]:

$$Q_{new}(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left[r(s, a) + \gamma \max_{a'} Q(s', a') \right] \tag{2}$$

Where $Q(s, a)$ is the current Q-value for a state-action pair, $Q_{new}(s, a)$ is the updated value, α is a learning rate, γ is the discount factor and $\max_{a'} Q(s', a')$ is the maximum Q-value for action a' in s'. The discount factor is a hyperparameter between 0 and 1, that sets the

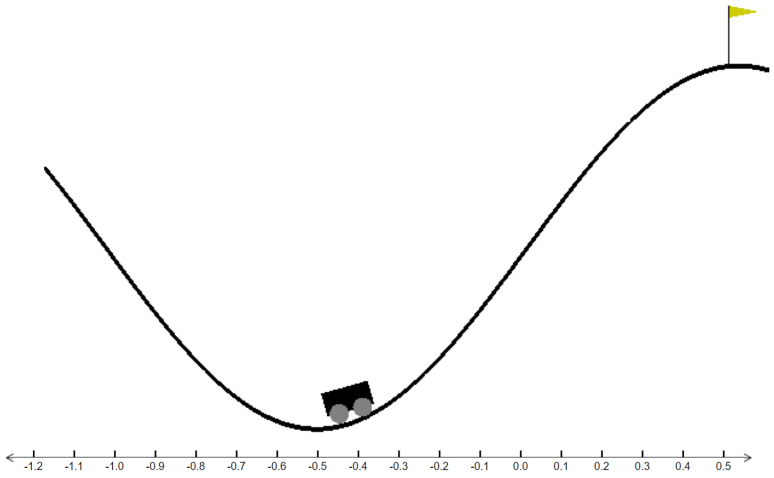

Figure 1: An image rendering a starting position of MountainCar-v0. and the game-defined distances. The car begins around position -0.5 and completes the episode upon reaching position 0.5 or after 200 steps

importance of future expected rewards over immediate rewards. The learning rate sets by how much values are updated based on new information.

3.2 Exploration-exploitation strategies in Q-learning

If the agent makes a decision on each step based only on the values of the Q-table, it will adopt a strictly greedy exploitation policy. Exploration by taking random actions is important to converge to an optimal strategy, otherwise the agent might only follow the first local improvement it reaches. In this work an epsilon-greedy (ϵ) policy was used as there is evidence that it stimulates convergence to strategies with good performance [7]. In ϵ-greedy policies, the agent chooses to either explore the environment (perform a random action) with probability ϵ or exploit what it learned performing the action with highest Q-value ($argmaxQ(s, a)$). Two variations of ϵ-greedy policy were chosen: (1) Exponential decay of ϵ over time, starting with value of 1 and decreasing exponentially with a factor of 0.98 until a minimum of 0.01. (2) A reward based ϵ decay (RBED) where ϵ was only decayed if $\max(x_{episode}) > -0.25$. This strategy favours exploration at the beginning of training and assigns more responsibility to the agent depending on its performance, converging a greedy policy [8].

3.3 Representing the observation space (OS)

One important limitation of Q-learning is that the size of the Q-table required to represent every possible state-action combination frequently requires an infeasible amount of memory. Additionally, if Q-tables are too large (matrix sparsity), many Q-values might not be updated enough, even after thousands of iterations. Even though mountain car is a simple environment, it has a continuous 2D observation space which cannot be represented in a

Table 1: Overview of hyperparameters grid-search. Discrete OS size indicate number of discretised positions and velocities, respectively. Reward for the goal were set to 0 (default) or 1000. For the ϵ regime, exp is exponential decay, RBED is reward based ϵ decay.

Search	GD1	GD2	GD3	GD4	GD5	GD6	GD7	GD8
Discrete OS size	$(18, 14)$	$(18, 14)$	$(18, 14)$	$(18, 14)$	$(36, 28)$	$(36, 28)$	$(36, 28)$	$(36, 28)$
r_{goal}	0	0	1000	1000	1000	1000	0	0
ϵ regime	exp	RBED	exp	RBED	RBED	exp	RBED	exp

Q-table. The mountain car OS was discretised into (18, 14) or (36, 28) bins, corresponding to the number of possible values for position and velocity, respectively.

3.4 Hyperparameter selection strategy

Q-learning utilises several hyperparameters in its implementation:

- The value of rewards and penalties (r)
- The learning rate (α)
- The rate of decay of the epsilon function (ϵ)
- The value of the discount function (γ)
- The observation space (OS) (number of state-action pairs)
- The number of discretised actions

As a trade-off between time and computational cost, a grid search was performed with few dimensions: the OS representation (2), the reward system (2) and ϵ regime (2), as shown in Table 1. The learning rate α was fixed at 0.2, the discount factor γ at 0.995 and the number of episodes at 10000. Along with the default reward scheme, a second reward system was developed that gives a reward of 1000 once the car reaches the goal, making the final reward much higher than the default.

3.5 Deep Q-learning

Instead of representing Q-values in a large look-up tensor, they can be estimated through the use of deep Q-learning (DQL). Since neural networks can approximate continuous functions, they can be fed environment states and output Q-values for each action [2]. This allows to compute optimal state Q-value relationships while bypassing the problem of memory representation. A deep Q-network (DQN) with parameters θ estimates Q-values at iteration m and uses the temporal difference error as the loss function (equation 3).

$$L_m(\theta_m) = \mathbb{E}_{s'}[(y_i - Q(s, a; \theta_m))^2] \tag{3}$$

Where:

$$y_i = [r(s, a) + \gamma \max_{a'} Q^*(s', a'; \theta_{m-1})|s, a] \tag{4}$$

By differentiating this function with respect to θ, we get that a minimum loss is obtained when the Bellman equation is satisfied resulting in:

$$Q(s, a; \theta_m) = \mathbb{E}_{s'}[r(s, a) + \gamma \max_{a'} Q(s', a'; \theta_{m-1})] \tag{5}$$

For DQN a feed-forward neural network (FFNN) was used to estimate Q-values. The chosen architecture of the FFNN is outlined in Table 2.

3.6 Experience replay

Successive states have a high temporal correlation that can lead to model overfitting. Experience replay with random sampling was used to remove temporal correlations in state sequences [9]. A replay memory of game transitions $e_t = s_t, a_t, r_t, s_{t+1}$ was continually updated as the model gained more experience and stored in a replay buffer, $D = e_1, e_2, \ldots, e_t$. Q-values were updated at every step in mini-batches of 50 samples. Older transitions were overwritten by latest experiences after the memory buffer reached a limit of 50000.

4. Results

4.1 Q-learning

As shown in Table 1, eight models of Q-learning agents were trained with different hyperparameters. Based on initial results, no visible difference was seen between the two ϵ regimes for training and performance. Performance is viewed as the maximum score consistently achieved during the training.

Fig. 2 shows the maximum, minimum and average scores achieved during each of the exp ϵ runs in bins of 100 episodes. Reward plots clearly indicate evidence for learning after around 1000 episodes. Notably, none of the runs came close to an average reward above -110, which is the OpenAI gym criteria to consider this environment solved. The scores exhibit a high degree of seemingly random fluctuations after 2000 episodes and do not appear to converge to only increasing rewards. Moreover, agents do not seem to adhere to an optimal policy and even move to sub-optimal policies (lower average scores) over training. This behaviour is unexpected and performance should converge over much larger training runs.

If a minimum score is above -200, the agent successfully reached the goal in all of the last 100 episodes. While this is arguably a good performance that was detected in many

Table 2: Architecture of the neural network used for the deep Q-learning experiments

Layer type	Nodes	Activation
Input	2 (Position and Velocity)	–
Fully Connected 1	50	ReLU
Fully Connected 2	50	ReLU
Output	3 (Q-value for each action)	Linear

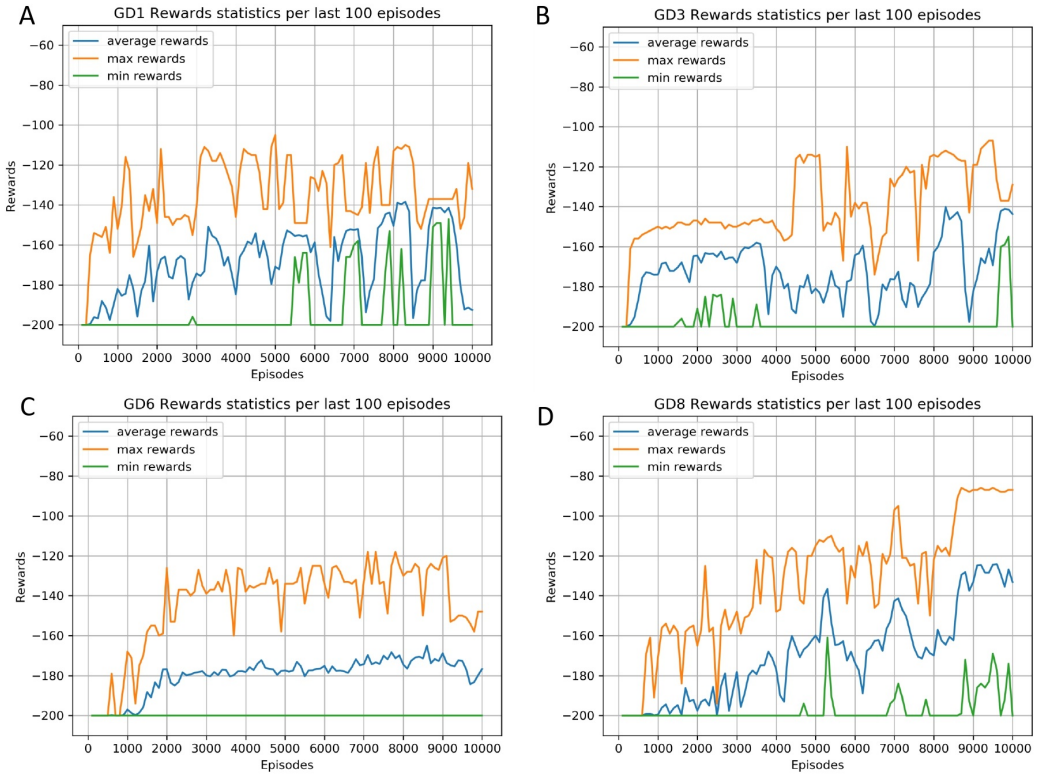

Figure 2: Reward plots representing maximum (orange), minimum (green) and average (blue) scores throughout the training for Q-learning grid search runs. Each data point is an average over the last 100 episodes.

agents, this metric is very unstable, since failure to reach the flag once is enough to decrease it to -200 over the last 100 runs. Some differences can be observed between models with different hyperparameters. The first apparent difference is that it takes longer for the more granular OS representation of (36, 28) to improve scores. For the more coarse representation (18, 14), the average starts to increase at around 500 episodes (Fig 2A, 2B), while it takes around 1000 episodes for average increases in (36, 28) representations (Fig 2C, 2D). This effect is probably due to the larger Q-table generated by a more granular OS, as more episodes are required to visit a significant number of states and populate the Q-table with meaningful values. Smaller Q-tables would are be populated faster and therefore allow the agent to reach higher scores after fewer episodes. A more granular representation of the OS does not seem to benefit performance on Mountain car that has such a simple optimal policy and smaller Q-table sizes may be preferred.

A second difference can be discerned between the two reward schemes for completion. The runs with positive r_{goal} values of 1000 (GD3-6) exhibit slightly less score oscillations. This may be due to high final rewards propagating greater values along the Q-table, leading the agents to a more stable policy. In contrast, with the default final reward of 0, changes in Q-values are more likely to significantly vary the action taken.

Due to the simplicity of the environment, we reasoned that ϵ should not be kept at high values for most of the learning, since the agent should converge to an optimal policy it should exploit. Using both our exponential and RBED ϵ regimes, epsilon decreased to the minimum of 0.01 before 200 episodes. However, it could be that random actions would still allow the agent to update poorly explored action-state combinations. To further explore this, we performed one RBED run with ϵ decreasing only until 0.2. This could potentially

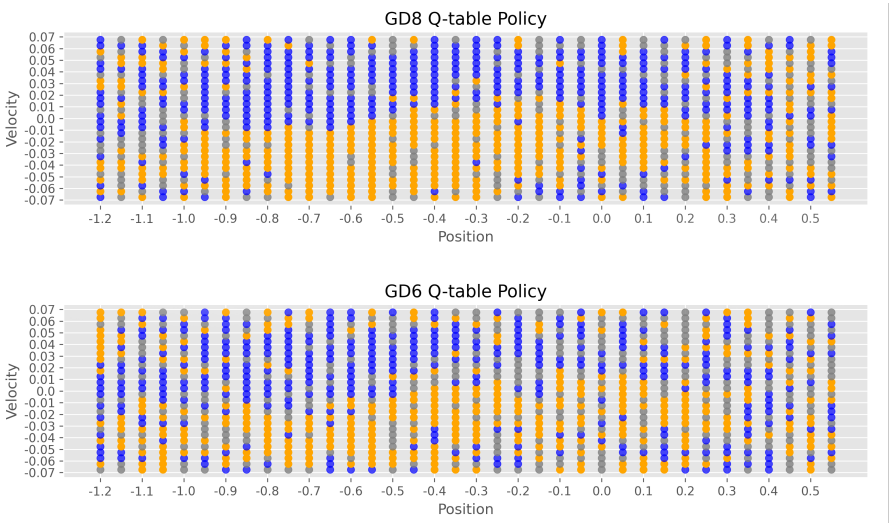

Figure 3: Q-table policies for runs 6 and 8. Coloured dots represent actions such as blue = right, orange = left, grey = no action.

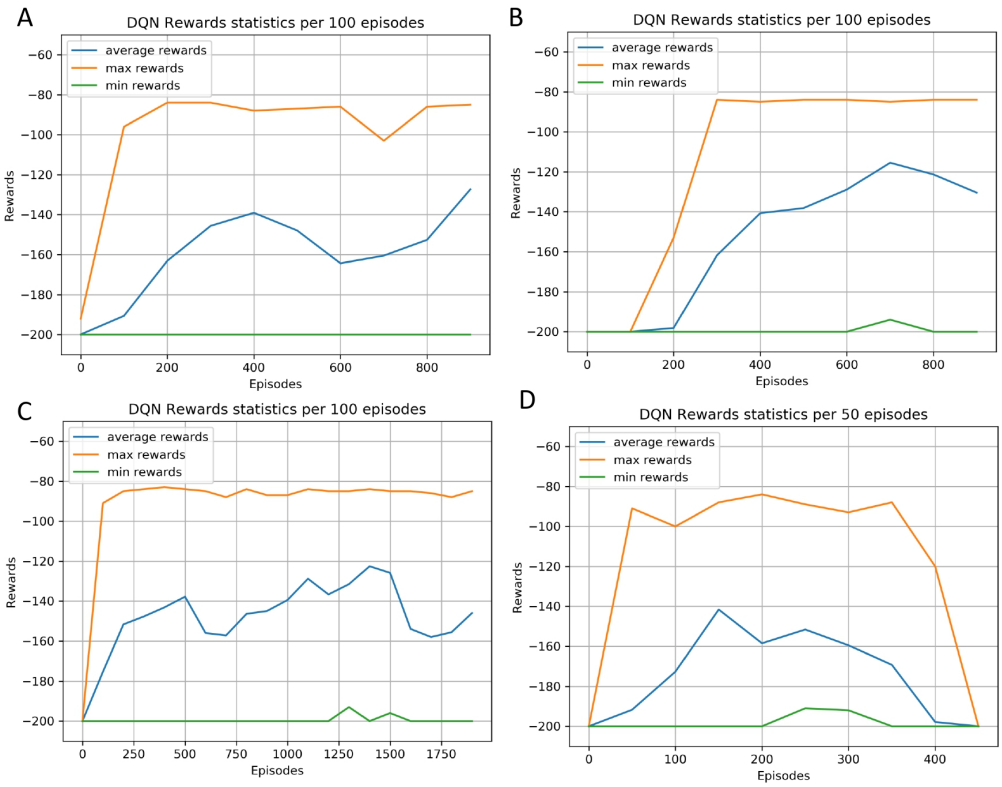

Figure 4: Reward plots representing maximum (orange), minimum (green) and average (blue) scores for deep Q-learning runs. Each data point is averaged over 100 episodes.

improve learning on very complex environments, but it only hindered performance and led to lower maximum rewards in mountain car.

Another interesting comparison can be made between policy Q-tables for high and low scoring runs. As can be seen in Fig. 2, GD8 achieves consistently high scores whereas GD6 scored much lower towards the end of training. Such differences are well-reflected in the Q-tables for each run (Fig. 3). The Q-table for run 8 has much clearer separation between right and left actions depending on the velocity and position. This is intuitive, as a leftward moving car (negative velocity) around the middle position (around $x = -0.3$) should continue going left, more often than not, to gain momentum. However, in the greedy Q-table policy for GD6, there is less consistency in state-action decisions along the q-table, which explains the worse performance.

4.2 Deep Q-learning

Due to the computational cost of training even simple FFNNs, a limited number of experiments could be conducted. Using the hyperparameters $\epsilon = 1$, $\gamma = 0.995$, and a variable learning rate as dictated by the Adam optimiser, a sharp increase in maximum rewards was achieved, as well as a modest and sustained trailing average (Fig. 4). However, a high score-per-episode variability can be seen by the minimum rewards line seldom climbing above -180.

On two last DQN runs, we attempted to perform reward shaping by adding a progressive reward dependent on the height of the hill. A reward of 100 was added if the goal was reached, 20 if the position was higher than 0.25 and 10 above 0.1. Interestingly, performance improved similarly to other runs, but dropped significantly. This behaviour is illustrated in Fig. 4D.

5. Discussion

This paper aimed to evaluate and compare performances of the Q-learning and Deep Q-learning algorithms at a classic RL control task. A caveat that should be acknowledged is the innumerable possible combinations of user-defined hyperparameters and architecture choices. The scope of this work is not, however, to declare one algorithm as fitter than another, rather it is an exploratory report examining applications and their suitability for the task.

Based on the obtained results, even for a simple problem such as MountainCar-v0 a simple deep Q-learning approach seems to perform better than a Q-learning algorithm due to its ability to approximate Q-values. Not only does it learn significantly quicker than Q-learning algorithm, it also achieves higher scores much more consistently and (at least in terms of max reward) exhibits lower variation (Fig. 4). Such differences should be even more apparent in more complex environments, as can be seen from multiple examples from Atari games to biological data [10, 11]. Interestingly, in two DQN runs with reward shaping, performance dropped significantly by the end of the runs, which could indicate instability of Q-value predictions by a single neural network. Since the model is being trained upon each iteration, its predictions and reward feedback are susceptible to oscillations. The agent might have forgotten actions to reach the flag and attempted to stay within lower regions of the hill that also contained positive rewards. One way to prevent this unwanted behaviour is by using a double DQN approach with a target neural network that is only updated every k steps. This leads to a much higher stability of Q-value predictions. For more complex tasks, Q-learning without any environment simplifications becomes impractical due to a quick increase of the Q-table size. It is hoped that this comparison will allow us to make informed decisions on the preferred application of these algorithms in different problems.

Acknowledgements

We would like to thank Yaiza Barrau Alzuria for suggesting Google Colaboratory and for her advice provided over the duration of this project.

References

[1] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *arXiv preprint arXiv:1708.05866*, 2017.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[3] C. Gatti, "The mountain car problem," in *Design of Experiments for Reinforcement Learning*, pp. 95–109, Springer, 2015.

[4] A. W. Moore, "Efficient memory-based learning for robot control," tech. rep., University of Cambridge, Computer Laboratory, 1990.

[5] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.

[6] A. Gosavi, "Reinforcement learning: A tutorial survey and recent advances," *INFORMS Journal on Computing*, vol. 21, no. 2, pp. 178–192, 2009.

[7] Y. Mohan, S. G. Ponnambalam, and J. I. Inayat-Hussain, "A comparative study of policies in q-learning for foraging tasks," in *2009 World Congress on Nature Biologically Inspired Computing (NaBIC)*, pp. 134–139, 2009.

[8] A. Maroti, "Rbed: Reward based epsilon decay," *arXiv preprint arXiv:1910.13701*, 2019.

[9] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3-4, pp. 293–321, 1992.

[10] M. Mahmud, M. S. Kaiser, A. Hussain, and S. Vassanelli, "Applications of deep learning and reinforcement learning to biological data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 6, pp. 2063–2079, 2018.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.