School of Informatics

Informatics Research Review
Value-Based Model-Free Deep Reinforcement Learning Algorithms and Applications

B225254 - Kyriakos Kyriakou
January 2023

Abstract

This review, explores the topic of value-based model-free deep reinforcement learning (DRL). It begins by providing an overview of the basic concepts of reinforcement learning. Thereafter, it delves into the specific area of value-based model-free DRL and discusses the popular algorithms that have been proposed, including DQN, Double DQN, Dueling DQN, C51, PER, Noisy Nets, Multi-step DQN, and Rainbow DQN. Furthermore, the study examines the strengths and weaknesses of each algorithm in terms of their efficiency and stability. Additionally, we dive in their application in various fields such as autonomous driving, natural language processing, and robotics. The literature review concludes with a summary and discussion on the future direction of research in this field and how this review can be useful for researchers exploring this area.

Date: Tuesday 7$^{\text{th}}$ November, 2023

Supervisor: Dr. Borislav Ikonomov

1 Introduction

Reinforcement Learning (RL) is a type of machine learning that enables systems to learn on their own. It involves training artificial intelligence (AI) agents to learn by taking actions in an environment and receiving punishments or rewards for those actions. Agents learn to maximize a reward signal through a process known as trail and error [1, 2]. An environment consists of many states, where each of these states comes with rewards. Agents interact with the states of the environment, and receive evaluative feedback based on the actions they take until they find the optimal path that will give them the maximum rewards [3, 1, 2, 4]. This is also known as optimal policy. Policies are used to determine the actions that agents should take in any given state. With this reward-driven behaviour, agents can learn from experiences and get better over time [3].

In the past, reinforcement learning has been limited to low-dimensional problems, as the RL algorithms carry memory and computational complexity issues [5, 2]. Over the last few years, a field called deep reinforcement learning (DRL) has gained a lot of popularity. This field combines reinforcement learning with deep learning techniques [6]. Neural networks can be used innovative to develop autonomous agents that are capable to handle decision-making problems that previously were impossible by enabling them to act in large scale high-dimensional environments [5].

Deep reinforcement learning has been applied to wide range of tasks and industries. One of the breakthroughs of DRL was the advent of an algorithm that was able to learn how to play a variety of Atari 2600 video games at a phenomenal level simply by analyzing raw pixel data from images [5, 7]. Furthermore, another major milestone for DRL is the development of AlphaGo by DeepMind that beat a world champion in Go board game [5, 8]. Other application of deep reinforcement learning has been seen in robotics where robots are now capable of taking decisions just from camera input [5]. Moreover, it has been applied in autonomous driving where cars can be controlled, and path planning and trajectory can be optimized better [9]. Finally new self-improving Chatbots based on DRL are being developed that achieve real user conversations [10].

There are two existing techniques in deep reinforcement learning. The first one is model-based DRL, where we can simulate actions in the environment using a learned model [5]. A model is used to predict the future states of the environment given the current state and actions, something that enables long-term planning in the environment [5]. The second is model-free DRL approach, in which agents learn directly from the actions they take in the environment [5]. Model-based technique is a more efficient way but holds many complexities as it's difficult to model whole environments [5]. On the other hand, model-free DRL is applied in scenarios that we don't have full observation of the environment and we train agents to plan ahead [11, 5]. Model-free approach is a more popular study in DRL as it is particularly useful in complex environments and can handle high-dimensional continuous environments. Additionally, deep neural networks and hardware advances have given new more effective ways at learning complex patterns and modeling complex problems [8, 5, 2, 3]. Within model-free DRL there are two different type of methods for solving tasks. The first approach is known as Value-Based. Value-based methods guarantee the best results by iterative computing the best state value function [5]. The second is the Policy-Based approach. With this approach, algorithms directly search for the best policy which defines the best behaviour of an agent in an environment [5, 2, 3]. Recent studies showed that value-based methods are more sample efficient than policy-based methods, as the don't require estimating a separate value function in addition to learning the

optimal policy [5, 11, 2, 3]. This is well-suited for environments where data are expensive to collect. In addition, policy-based methods are also more prone to instability than value-based methods. That's happening because policy-based methods do not rely on estimates of the value function, something that can be prone to errors [5, 11, 2, 3, 12].

In this review, we will focus on algorithms and applications of value-based model-free deep reinforcement learning. We will first examine the algorithms that have been developed for problem solving in value-based model-free deep reinforcement learning. By examining that, we aim to understand the strength and weaknesses of different algorithms, and identify areas for improvement and future research. We will explore the most popular algorithms including SARSA, Q-learning, DQN, Dueling DQN, Double DQN, Prioritized Experience Replay, Rainbow DQN, C51, Noisy Nets, and Multi-step learning. We will then compare the strengths and weaknesses in terms of efficiency and stability of the different value-based model-free DRL approaches, and briefly discuss their potential for solving real-world problems in a variety of industries such as self-driving cars, NLP, and robotics. By answering these research questions, this review aims to provide a comprehensive overview of the current state of the field, and identify areas for future research and development.

2 Literature Review

2.1 Reinforcement Learning Basics

An agent in RL interacts with the environment over a period of time t, and at each state s_t receives a reward $r(s_t)$ based on an action α_t in a state space S and an action space A [2]. It selects those actions based on a policy $\pi(\alpha_t|s_t)$, which according to the environment dynamics is based on the reward of the current state $r(s_t)$ and the transition to the next one $r(s_{t+1})$[2]. The transition dynamics are formulated as $T(s_{t+1}|s_t, \alpha_t)$ [5]. The agent's goal is to maximize the expected long-term return (find the optimal policy), that is computed based on a reward discounted function $R_t = \sum_{n=0}^{\infty} \gamma^n r_{t+n}$ [2, 5, 3]. The discounted factor γ, is used to adjust the relative importance of rewards received at different timestamps t, as we want to reduce the impact that future rewards have on our value estimates [5, 11].

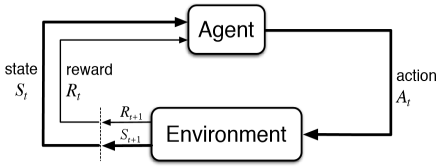

Figure 1: Basic Structure of Reinforcement Learning [3, 13]

2.1.1 Markov Decision Process

Markov Decision Process (MDP) is a framework that is used in RL to formulate sequential decision making problems within an environment [5, 2, 3]. It is represented as $MDP = (S, A, r, T, \gamma)$ [3]. The key property for the Markov process is underlined in the fact that only the next state is affected by the current one [5]. Agents' new decisions lie only on the s_{t-1}, instead of the whole S [5]. All we want to achieve is to maximize the expected reward from all states by finding the best policy $\pi^* = arg_\pi max\, E[R|\pi]$ [5, 3, 2, 1]. There are two three main approaches

to solve RL problems. We can use value-based methods, policy search methods, and actor-critic methods which combine the previous two [5]. In this study we will focus only on the value-based approach.

2.1.2 Value-Based methods

Value-based methods involve determining the anticipated outcome (value) of a particular state [5, 3, 2]. A state-value method $V^{\pi}(s)$ is the predicted outcome from a starting state s and adhering to policy π from then on [5]. The ideal/optimal policy π^*, is associated with a specific state-value method V^*, and conversely, the optimal state-value method can be determined by the optimal policy [5, 1]. The value function is defined by $V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t R_{t+1} | S_0 = s]$ [1, 2]. Value-based methods are divided into two categories: policy-based and off-policy methods (shown in Figure 2 at the appendix A). Policy-based or on-policy methods learn the policy π directly from the state-action mapping [5, 1, 11]. In contrast, off-policy methods tend to learn a value function or a quality function (Q-function), which is mapping from states and actions the expected returned reward [5, 1, 11]. Later, based on the selected action, the best return from the value function is used to derive the policy [5, 11, 1, 2].

2.1.3 SARSA

SARSA (state–action–reward–state–action) algorithm is an on-policy method that is mapped to state-action pairs [1]. It's an easy algorithm to implement, but comes with the disadvantage that is not very efficient at scaling continuous environment problems [3]. SARSA algorithm can be really effective on discrete environments (finite set of states) [11]. SARSA finds the optimal policy π^* using the rule $Q(s_t, \alpha_t) \leftarrow Q(s_t, \alpha_t) + \alpha[r_t + \gamma Q(s_{t+1}, \alpha_{t+1}) - Q(s_t, \alpha_t)]$ in state-action pairs [2, 14]. It continuously search greedily for the optimal behavior π in order to estimate the quality function Q_{π} [14]. The algorithm is given below Algorithm 1.

Algorithm 1 SARSA, adapted from Li, and Sutton and Bart (2018) [2, 14]

Output: Action value function Q
Initialize: All states S and the action values Q_s for each state to 0
for *each environment interaction*
    Initialize: state s_t
    for *each state step* in the environment
        If s_t is NOT terminal
            $\alpha \leftarrow$ action obtained by Q for state s_t
            take action α and perceive current reward r_t, and next state s_{t+1}
            $\alpha_{t+1} \leftarrow$ action obtained by Q for state s_{t+1}
            Update action value $Q(s_t, \alpha_t) \leftarrow Q(s_t, \alpha_t) + \alpha[r_t + \gamma Q(s_{t+1}, \alpha_{t+1}) - Q(s_t, \alpha_t)]$
            Update state $s_t \leftarrow s_{t+1}$, and action $\alpha \leftarrow \alpha_{t+1}$
    end
end

2.1.4 Q-Learning

Q-learning algorithm was firstly introduced by Watkins and Dayan (1992) and it is an off-policy method [15]. It helps agents learn the optimal policy π^* directly by interacting with the environment and experiencing the outcomes of actions [15, 14]. It learns the value function Q

using the rule $Q(s_t, \alpha_t) \leftarrow Q(s_t, \alpha_t) + \alpha[r_t + \gamma \max_{\alpha_{t+1}} Q(s_{t+1}, \alpha_{t+1}) - Q(s_t, \alpha_t)]$ regardless of the policy being executed [14, 15, 2]. State-action values are stored in a Q table which helps agents to choose the best actions (getting max) [3]. Agents can trade-off exploitation and exploration when using Q learning [3]. When choosing actions based on the maximum q-value Q^* we focus on exploiting. In opposition to that, when agents select actions more randomly we focus more on exploring [3, 15]. For each state-action pair, Q^* provides the highest possible expected return that can be attained through any policy π [14]. Consequently, Q-learning is more efficient and robust than SARSA as it uses experience memory to make choices and always selects the best rewarding actions. However, it's worth noting that SARSA may perform better in situations where the optimal policy is not known.

Algorithm 2 Q-learning, adapted from Li, and Sutton and Bart (2018) [2, 14, 15]

Output: Value function V
Initialize: All states S and the action values Q_s for each state to 0
for *each environment interaction*
    Initialize: state s_t
    for *each state step* in the environment
        If s_t is NOT terminal
            $\alpha \leftarrow$ action obtained by Q for state s_t
            take action α and perceive current reward r_t, and next state s_{t+1}
            Update value $Q(s_t, \alpha_t) \leftarrow Q(s_t, \alpha_t) + \alpha[r_t + \gamma \max_{\alpha_{t+1}} Q(s_{t+1}, \alpha_{t+1}) - Q(s_t, \alpha_t)]$
            Update state $s_t \leftarrow s_{t+1}$
    end
end

2.2 Value-based model-free DRL algorithms

In this section, the recent research advancements in model-free value-based DRL will be reviewed, including the most popular algorithms, such as DQN and its variants, as well the challenges and limitations of these approaches. Deep neural networks can be used to approximate RL components such as optimal policy π^*, optimal value functions V^* and Q^*, and model functions for state transitions T and rewards R straight from visual inputs with high-dimensional space [2, 5, 6].

2.2.1 Deep Q-Network

DQN algorithm was first introduced by Volodymyr Mnih, et. al in 2013, and finalized in 2015 [16, 7]. The papers [16, 7] proposed a new approach to Q-Learning, where deep neural networks were used to approximate the action value function Q. This approach was able to achieve human-level performance on several Atari games, making it one of the biggest breakthroughs in deep reinforcement learning [16, 7, 2, 5, 3]. The Q-network takes as inputs monochrome consecutive frames of the game, and combines them over a period of time. These frames go through a series of convolutional layers to identify and extract spatiotemporal characteristics [5, 16]. It is trained to minimize the difference between predicted action values α and the target action values [5, 16, 7]. One of the key innovations of DQN is it's ability to use a replay buffer that stores previous experiences, consisting of the current state, action, reward, and next state $(s_t, \alpha_t, r_t, s_{t+1})$ [5, 16, 7]. This allows the algorithm to learn from a more diverse set of experiences and handle problems with massive state space, something that reduces the amount

of interactions needed, and also helps to the learning process stabilization [16, 5, 2]. Loss is minimized by using stochastic gradient descent to optimize the weights of the Q-network with respect to the parameter θ [17, 2, 16, 18]. DQN's algorithm biggest flaw is that it has a tendency to sometimes overestimate the values linked with certain actions, which is used to estimate the state-action value function Q, making it less stable [3, 7, 2]. More specifically, the max operator is utilized in such a way that the same set of values are employed to both pick and assess an action [2].

Algorithm 3 DQN, adapted from Luong, et. al in 2019, and Li in 2018 [2, 17, 16]

Input: game score and image pixels
Output: value-action function Q
Initialize: cyclic memory D to capacity N
Initialize: action-value function Q with random weights θ & target $\hat{Q}$ with random weights $\theta^{'}$
for *episode=1 to T*:
Choose with ϵ probability random action α_t or select $\alpha_t = argmaxQ^*(s_t, \alpha_t; \theta)$
Execute action α_t and observe reward r_t and next state $st + 1$
Store transition $(s_t, \alpha_t, r_t, s_{t+1})$ in D
Sample random minibatch of transitions $(s_j, \alpha_j, r_j, s_{j+1})$ from D
Perform a gradient descent step w.r.t. the network parameter θ to decrease loss:
$L \leftarrow \left[Q(s_t, \alpha_t; \theta) - (r_t + \gamma max(\hat{Q}(s_{t+1}, \alpha_{t+1}; \theta)))\right]^2$
Resetting $\hat{Q} = Q$ after K steps
end

2.2.2 Double Deep Q-Network

Double DQN algorithm was proposed by Hasselt et al. (2015) to address the problem of overestimation in Q-Learning and DQN [19, 2, 20]. They suggested evaluating the optimal policy based on the current network, but utilizing the target network to calculate its value [19]. The new proposed algorithm changes the Loss function as follows:

$$Loss = [Q(s_t, \alpha_t; \theta) - (r_t + \gamma \hat{Q}(s_{t+1}, argmax_{\alpha_{t+1}} Q(s_{t+1}, \alpha_{t+1}; \theta); \theta_{target}))]^2$$

Two action-value functions Q_{online} and $\hat{Q}_{target}$ are used . The Q_{online} is used for selecting the actions, and the $\hat{Q}_{target}$ is used for evaluating the maximum future reward [19, 2]. Using a separate target network $\hat{Q}_{target}$ for the evaluation helped improved the estimations of the Q-values and improved the overall stability of the learning process [17, 19, 5]. The most critical limitation of Double DQN is its sample efficiency making the algorithm slow, because the Q_{online} part of the algorithm requires updating the network after each step in the environment [19].

2.2.3 Dueling Deep Q-Network

Dueling DQN algorithm was first introduced by Wang et al. in 2015 [21]. The algorithm introduced a new way to estimate the action value function Q for all actions [21]. It separates the estimation of the value function $V(s)$ and the action function $A(s, \alpha)$ [21, 17]. The first stream estimates the state value function and how good it is given a state s, and the other one estimates the action advantage function on how much better taking a specific action α is compared to the average action in state s [17, 21]. At the end, the algorithm combines the two streams to calculate the Q action-value function for each action [21]. The Q value function for

each action is computed as the sum of both: $Q(s_t, a_t) = V(s_t; \beta) + A(s_t, \alpha_t; \alpha)$ [2, 21, 17]. The α and β are the parameters of the two streams respectively. The streams $A(s, \alpha)$ and $V(s)$ are applied using two single fully connected layer networks [17]. The loss function of the algorithm is the mean-squared error and is defined as follows:

$$Loss = \Big[Q(s_t, \alpha_t; \alpha, \beta) - (r_t + \gamma max_{\alpha_{t+1}}(\hat{Q}(s_{t+1}, \alpha_{t+1}; \alpha, \beta))\Big]^2$$

Dueling DQN addresses again the problem of overestimation of Q-values in the DQN algorithm and improves its performance. Its strength lies in its ability to solve large action problems more efficiently, but only when a good state representation is available [21, 20].

2.2.4 Prioritized Experience Replay

The paper [22] by Tom Schaul et al. (2016) proposed a modification to the experience replay mechanism used in DQN called Prioritized Experience Replay (PER) [22, 2, 17]. In the original DQN, the reinforcement learning agent can recall and utilize previous experiences, also known as transitions, through the use of the experience replay mechanism [2, 17, 7]. The new PER modifies the mechanism, and the agent can assign a priority value to each experience [22, 17]. Thereafter, the agent can sample the experiences for learning according to those priority values. As a consequence, the agent can learn to focus on the most informative experiences [22]. The algorithm starts by collecting transitions and storing them in the memory, and then for each learning step, it samples a batch of transitions from the replay memory according to their priority values, using a probability distribution defined as:

$$P(i) = \frac{p(i)^\alpha}{\sum_k^N p(k)^\alpha}$$

The N in the equation is the number of transitions in the replay memory and α is a hyper-parameter that controls the degree of prioritization [22]. The priority for the i-th transition is defined as $p(i) = |\delta(i)| + \epsilon$ [22]. Transitions that are newly acquired are given the highest priority when added to the replay buffer, giving an emphasis to the most recent transitions [17]. The PER algorithm uses the magnitude of the temporal difference error [23] for the learning potential of a transition (as a loss function) [22]. The loss function is defined as:

$$Loss = \Big[Q(s_t, \alpha_t; \theta) - (r_t + \gamma max_{\alpha_{t+1}}(\hat{Q}(s_{t+1}, \alpha_{t+1}; \theta))\Big]^\omega,$$

where The hyper-parameter ω controls the shape of the probability distribution used to sample transitions in the replay buffer [17, 22]. PER enables a faster and a more stable learning by focusing on the most informative experiences, but at the same time, calculating the priority scores for each experience can lead to a computationally expensive process [22].

2.2.5 Categorical Deep Q-Network (C51)

Bellemare et al. in 2017 [24] proposed C51 algorithm, that uses a categorical distribution over Q-values to represent the action-value function Q [24, 25]. More specifically, the Q function is represented as a categorical probability distribution over a set of discrete support points, instead of single scalar value that the original DQN and Q-learning use [24, 7, 15]. This make the algorithm able to estimate better the Q-values in problems with high degree of stochasticity [24]. Using a distribution of probabilities for each state-action pair leads to capturing with more certainty the Q-values [25]. Therefore, C51 provides a more accurate and robust estimations of

the Q-values, and also handles the over-estimation problem of DQN, outperforming the previous algorithms in many Atari games [24]. The algorithm is using the same network for updating the Q-network parameters to get trained as DQN, by getting experiences from the transitions $(s_t, \alpha_t, r_t, s_{t+1})$ of the agents. However, C51 Q-network outputs a probability distribution that represents the possible Q-values (set of support points), rather than a scalar value [7, 24]. The set of support points usually are determined by the problem designer according to the problem. Given that set of z points $z = (z_1, z_2, .., z_n)$, the predicted probability distribution for a state-action (s_t, α_t) is defined as $Q(s_t, \alpha_t, z) = P(Z = z|S = s_t, A = \alpha_t)$ [24]. Then the Q-value for a given (s_t, α_t) is given as as $Q(s_t, \alpha_t) = \sum_{z_n} z_i Q(s_t, \alpha_t, z_i)$ [24, 25]. A cross-entropy loss function is used for C51 represented as [18]:

$$Loss = -\sum_{z_n} \hat{Q}(s_t, \alpha_t, z_i) log(Q(s_t, \alpha_t, z_i))$$

The main weakness of the C51 is its sensitivity on choosing hyper-parameters such as the set of support points, making it hard to find the optimal settings for a given problem [24].

2.2.6 Noisy Nets for DQN

Noisy Nets were introduced by the authors in [26] paper. The authors suggested a different way of exploring instead of the traditional ϵ-strategy, called noisy nets [26]. Noisy nets use a noise function to add random noise to the neural network and weights [17]. In more detail, it adds a Gaussian noise to the weights of the network at each training epoch [17]. This allows the agent to make new decisions and explore more due the introduced uncertainty to the weights [26]. This is done without affecting the actual wanted weights values. After each training step, the added noise is re-sampled to allow the network to converge to the optimal policy π^* [26, 17]. The noise are added to the weights with the following: $w \leftarrow w + \nu$, where w are the weights and the ν the noise from Gaussian distribution [26]. The learning procedure is using a new proposed loss function defined as [17]:

$$Loss = E[Q(s_t, \alpha_t, \nu_t; \theta) - (r_t + \gamma max_{\alpha_{t+1}}(\hat{Q}(s_{t+1}, \alpha_{t+1}, \nu_{t+1}; \theta))]$$

Setting the noise ν hyper-parameter is considered the difficult task of the noisy nets as with a high selected noise, the optimal policy might be not found, and with small, the exploration might not be enough [26]. With a good noise, noisy nets will activate a new ability of exploring more efficiently than other traditional methods [26, 27].

2.2.7 Multi-step Deep Q-Learning

Mnih et all. in 2016 [28] proposed a new way of learning the Q-values. In the multi-step learning procedure, the agent considers multiple steps ahead of the action-value function Q, instead of seeing only one step ahead as in the normal DQN [28]. That, makes the agent capable of learning a more optimal sequence of state-rewards to better estimate the long-term rewards of its actions using discount factor γ. Therefore, multi-step DQN can improve the stability and sample efficient of the Q-learning algorithm. The new quality function is defined as [17]:

$$Q(s_t, \alpha_t) = r_t + \sum_{t=1}^{n} \gamma_t r_t + \gamma_{n+1} max_\alpha Q(s_{t+1}, \alpha_{t+1})$$

The loss function for multi-step learning remains the same with the original DQN [7]. Choosing a wrong number of a step size (how many steps will be consider for Q-value) can lead to a more unstable learning [28].

2.2.8 Rainbow Deep Q-Network

DeepMind team in their paper (2017) came up with a brilliant research to combine several improvements of DQN [27]. This proposed algorithm is named Rainbow DQN, and it's considered as the state-of-the-art innovation for value-based model-free DRL. Rainbow DQN achieves outstanding performance on a wide range of Atari games [27]. The above analyzed algorithms are combined in the rainbow DQN [27]. More precisely, it uses Double-DQN to overcome the over-estimation problem, PER to help the agent focus on transitions that are more important, Dueling DQN to handle the trade-off between the value and action advantage streams, multi-step learning to allow the agent get better estimations for the complex long-term rewards, noisy nets to encourage exploration withing the environment, and finally categorical-DQN to boost the confidence of the uncertain returns [26, 28, 24, 22, 21, 19, 7]. While the Rainbow DQN achieves the best results between the DQN variants, it's computational expensive to train [27]. Figure 3 at the appendix B displays a graphical comparison of the algorithms taken from [27].

2.3 Comparison of the value-based model-free DRL algorithms

In this part, a comparison of the performance of the above value-based model-free DRL algorithms will be examined. Table 1 analyzes the biggest strengths and limitations of each algorithm in terms of their efficiency and stability. Efficiency refers to how well and fast an algorithm can learn and make decisions with some given computational resources, and stability refers to how well the algorithm can maintain its performance over time and in different conditions [5]. Note that for different problems, the efficiency and stability of the algorithms is subject to change.

Algorithms	Strengths	Limitations
DQN	Simple and easy to implement algorithm, making it efficient in terms of computational cost and time. Uses experience replay, which improves stability by breaking the correlation between consecutive samples.	Sensitive to the choice of hyper-parameters, making it unstable. Prone to overestimation of values, making it less efficient in terms of convergence and final performance.
Double DQN	Addresses the problem with overestimation present in DQN by decoupling action selection from action evaluation, improving the stability and efficiency of the algorithm. Simple to implement and can be easily combined with other techniques for boosting performance.	Sensitive to the choice of hyper-parameters, making it unstable. Can be computationally more expensive than DQN due to requiring two separate Q-value networks.
Dueling DQN	Learns the optimal policy faster and more efficiently than DQN by decoupling the estimation of the value state and the estimation of the action advantage function. Handles states where actions not available, making it more stable.	Can be affected by the choice of hyper-parameters, decreasing stability – prone to overestimating. Can be computationally expensive as it requires handling two Q-networks (more costly).
PER	Prioritizing transitions that are more important makes the learning more efficient. Not getting "tricked" by consecutive transitions improves the stability.	Sensitive to the choice of hyper-parameters, especially assigning the importance weighting – affects stability. Maintaining a priority queue and computing the importance makes it computationally more expensive than using the regular cyclic buffer.
C51	Complex environments can be handled by representing Q-values with probability distribution making it more efficient than DQN. It decreases the variance of the Q-value estimates setting it to be more stable and robust.	Requires more computational power than DQN to compute the distributions. Stability is affected by choosing wrong number of the support set.
Multi-step DQN	Improves efficiency by using more steps of future rewards in the Q-value update (faster learning). Considers long-term rewards making it more robust and stable.	Computational power and memory needed to remember the multi-step return. Sensitive to the choice of step number, leading to instability if chosen incorrectly.
Noisy Nets	Exploring is done more efficiently by adding noise to the parameters of Q-network. Introduces stochasticity to the agents making the learning more stable.	Choosing the hyper-parameters, more particularly the noise added, can affect stability. Computationally more expensive than DQN due to the noise addition.
Rainbow DQN	Combines multiple improvements leading to a faster learning and better performance. More stable as it can handle different type of environments.	Extensive hyper-parameter tuning is required for specific tasks making the algorithm unstable. By far the most slow and computational expensive to implement, due to the complex design of combining the improvements.

Table 1: Comparison of different variants of DQN in terms of efficiency and stability

2.4 Applications of value-based model-free DRL

2.4.1 Natural Language Processing

Deep reinforcement learning is very commonly used in the field of NLP, applied in tasks including generation, question answering, translation, and conversations [29, 2]. Deep neural networks are used to approximate the action-value Q-functions, and learn policies that map inputs of words (states) to outputs e.g. generation (actions) [30]. The agent is represented by a language model, that takes states (sentences, words) and selects actions (add/delete/replace words) in a language processing environment [30]. That environment will determine the rewards after each interaction if the current taken action was correct. Recent advances in DRL and deep learning (transformer-based networks and LSTMs) led to a significant progress in solving sequence-to-sequence problems [30]. Input sentences are fed in LSTM or transformer models to learn different representations, and then DQN algorithms are applied to generate the output sentence [30]. Applied application that uses such techniques is the state-of-the-art Chat bot of openAI, Chat-GPT, capaple of understanding and generating human-like text in a wide range of topics [31].

2.4.2 Autonomous Driving

Value-Based model-free DRL methods have been applied to autonomous driving tasks including: controller (breaking/steering/accelerating), optimization, dynamic path planning, and trajectory optimization, by learning policies for complex navigation tasks that map the current state of the vehicle to an action. [9, 32]. Data including camera images, LiDARs, and GPS are used to capture vehicle states [9]. Then, these data are combined with deep neural networks to approximate action-value Q-functions for taking high reward actions [9]. The authors in [33, 34] proposed lane keeping and simple vehicle interaction assistance using DQN via a 3D car simulator called *TORCS*. Moreover, a motion planning for predictions, behaviours and trajectory estimation of vehicles using DQN algorithms and heuristics was suggested by the authors in [35]. These methods have shown to be effective in learning a safe and efficient policy for driving, however, there are many challenges to overcome such as the safety of these autonomous systems in real-world scenarios and not in simulations [9].

2.4.3 Robotics

Deep Q-Networks have been adapted to many tasks in the field of robotics. Some of them involve robotics grasping, navigation, and control [36, 37]. Value-based DRL techniques and sensors that capture the robots' motion enables the them to learn to perform these tasks. A robot manipulation example uses a DQN that gets states from the environment such as robot position, velocity, acceleration, target pose, and vision information, and outputs actions for controlling the robot [37]. Another example applies DQN for object grasping through multi-step reasoning [36]. Finally, the authors in [38] trained a mobile robot for grasping and stacking blocks tasks using deep Q-learning. There is still much to be explored in the field of robotics in order to make fully intelligent robots, but DRL opens many paths for exploration and researching [39].

3 Summary & Conclusion

All in all, this review has studied the area of value-based model-free deep reinforcement learning. It has been widely researched and applied in various fields, such as NLP, robotics, gaming, and many more. The DQN algorithm, proposed in 2013 by Mnih et al. [16], was one of the first successful algorithms implemented in the value-based model-free DRL field. Later, many variants of DQN have been proposed including, Double DQN, Dueling DQN, Prioritized Experience Replay (PER), Categorical DQN (C51), Noisy Nets, Multi-Step DQN, and Rainbow DQN.

Thereafter, the study has analyzed the strengths and weaknesses of the proposed algorithms compared to the baseline DQN in terms of efficiency and stability. Double DQN addresses the problem of overestimation in DQN, even though this algorithm is computationally expensive, whereas Dueling DQN improves learning by using two streams to represent the Q-values, despite the fact that's prone to overestimation. In addition, PER enhances the efficiency of learning, but can be memory expensive. C51 makes the agent more stable and robust, although it's hard to set up, while Noisy Nets enables a more efficient exploration of the environment, but requires more power. Moreover, multi-step DQN gets more accurate predictions for future rewards, even though it's sensitive to the step number selection. Finally, Rainbow DQN improves the overall performance by combining the improvements of the other DQN variants, although an extensive hyper-parameter tuning is required.

Which algorithm to choose is subjective and problem specific, therefore it's a hard to tell clearly which one is better. Based on their limitations and strengths, engineers should choose carefully the more suitable algorithm to model their problem. For example, all value-based algorithms can be applied for tasks in autonomous driving, even though Rainbow DQN is observed to be more commonly used, perhaps due to its ability to be efficient and stable at the same time. The algorithm should make fast decisions in complex environments without having oscillations in the performance for safety reasons. Furthermore, algorithms like Double DQN for better estimations and Noisy Net for better explorations, can be combined and applied to problems where we care about exploring the environment more efficiently and optimally like robot space exploration. An interesting research proposal based on this review, could be to improve the efficiency and stability of the current value-based DRL algorithms for autonomous driving, by combining and tuning them effectively, according to our task.

The review lacks on giving a depth analysis on how exactly the algorithms are implemented individually in the current applications like NLP, as the complexity of the projects does not allow for the employment of just one algorithm, but rather an integration of a number of them. Additionally, ethical and society matters have not been discussed, neither the challenges of implementing them in real-world scenarios. This review investigates only the most popular variants of DQN, thus, it misses latest development in the field.

For future research, it would be attractive to study the combinations of the value-based model-free DRL techniques with other methods, such as model-based DRL, or Actor-Critic methods. This would explore new areas, and potentially, could address some limitations of the value-based algorithms. Furthermore, new areas could be explored that apply these techniques including healthcare and finance. In conclusion, the aim of the study was not to come down in a favor of one out of the eight discussed algorithms, but rather to offer information on the most widely used value-based model-free DRL algorithms, as well as guidance for future research to decide on which algorithm (or algorithms) to choose for their specific project.

References

[1] Csaba Szepesvari. Algorithms for reinforcement learning. Draft of the lecture published in the Synthesis Lectures on Artificial Intelligence and Machine Learning series Morgan Claypool Publishers, June 9, 2009, DBLP, https://www.researchgate.net/publication/220696313_Algorithms_for_Reinforcement_Learning.

[2] Yuxi Li. Deep reinforcement learning: An overview. https://doi.org/10.48550/arXiv.1810.06339.

[3] Salim Dridih. Reinforcement learning - a systematic literature review. https://www.researchgate.net/publication/357380640_Reinforcement_Learning_-_A_Systematic_Literature_Review.

[4] Richard S. Sutton. Reinforcement learning architectures. GTE Laboratories Incorporated, Waltham, MA 02254, sutton@gte.com.

[5] Miles Brundage Anil Anthony Bharath Kai Arulkumaran, Marc Peter Deisenroth. A brief survey of deep reinforcement learning. IEEE.

[6] Juergen Schmidhuber. Deep learning in neural networks: An overview. Technical Report IDSIA-03-14, Neural Networks, Vol 61, pp 85-117, Jan 2015, https://doi.org/10.48550/arXiv.1404.7828.

[7] Koray Kavukcuoglu David Silver Andrei A Rusu Joel Veness Marc G Bellemare Alex Graves Martin Riedmiller Andreas K Fidjeland Georg Ostrovski Stig Petersen Charles Beattie Amir Sadik Ioannis Antonoglou Helen King Dharshan Kumaran Daan Wierstra Shane Legg Demis Hassabis SuttonArch, Volodymyr Mnih. Human-level control through deep reinforcement learning. Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015). https://doi.org/10.1038/nature14236.

[8] Maddison CJ Guez A Sifre L van den Driessche G Schrittwieser J Antonoglou I Panneershelvam V Lanctot M Dieleman S Grewe D Nham J Kalchbrenner N Sutskever I Lillicrap T Leach M Kavukcuoglu K Graepel T Hassabis D. Silver D, Huang A. Mastering the game of go with deep neural networks and tree search. Mastering the game of Go with deep neural networks and tree search. Nature. 2016 Jan 28;529(7587):484-9. doi: 10.1038/nature16961. PMID: 26819042.

[9] Bangalore Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad Sallab, Senthil Yogamani, and Patrick Perez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, PP:1–18, 02 2021.

[10] Elena Ricciardelli Debmalya Biswas. Self-improving chatbots based on deep reinforcement learning. Reinforcement Learning (RL) model for self-improving chatbots, specifically targeting FAQ-type chatbots, Towards Data Science, https://towardsdatascience.com/self-improving-chatbots-based-on-reinforcement-learning-75cca62debce.

[11] Koubaa A. Ali Mohamed N. Ibrahim H.A. Ibrahim Z.F.; Kazim M.; Ammar A. Benjdira B. Khamis A. Hameed I.A. Azar, A.T. Drone deep reinforcement learning: A review. Electronics 2021, 10, 999. https://doi.org/10.3390/electronics10090999.

[12] Yaser Faghan Puhong Duan Amir Mosavi, Pedram Ghamisi. Comprehensive review of deep reinforcement learning methods and applications in economics. https://doi.org/10.48550/arXiv.2004.01509.

[13] F. Shaikh. Simple beginner's guide to reinforcement learning its implementation. https://www.analyticsvidhya.com/blog/2017/01/introduction-to-reinforcement-learning-implementation/, 2017.

[14] Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction, second edition. The MIT Press, Cambridge, Massachusetts, London, England, 2018, 2020.

[15] PETER DAYAN CHRISTOPHER J.C.H. WATKINS. Q-learning. Watkins, C.J.C.H., Dayan, P. Q-learning. Mach Learn 8, 279–292 (May 1992), https://doi.org/10.1007/BF00992698.

[16] David Silver Alex Graves Ioannis Antonoglou Daan Wierstra Martin Riedmiller Volodymyr Mnih, Koray Kavukcuoglu. Playing atari with deep reinforcement learning. https://doi.org/10.48550/arXiv.1312.5602,2013.

[17] Shimin Gong Dusit Niyato Ping Wang Ying-Chang Liang Dong In Kim Nguyen Cong Luong, Dinh Thai Hoang. Applications of deep reinforcement learning in communications and networking: A survey. IEEE Communications Surveys Tutorials, vol. 21, no. 4, pp. 3133-3174, Fourthquarter 2019, doi: 10.1109/COMST.2019.2916583,.

[18] Begüm Demir Hichame Yessou, Gencer Sumbul. A comparative study of deep learning loss functions for multi-label remote sensing image classification. IEEE, Sep 2020, https://doi.org/10.48550/arXiv.2009.13935.

[19] David Silver Hado van Hasselt, Arthur Guez. Deep reinforcement learning with double q-learning. https://doi.org/10.48550/arXiv.1509.06461.

[20] Mohit Sewak. Deep q network (dqn), double dqn, and dueling dqn. In *Deep Reinforcement Learning*, pages 95–108. Springer, 2019.

[21] Matteo Hessel Hado van Hasselt Marc Lanctot Nando de Freitas Ziyu Wang, Tom Schaul. Dueling network architectures for deep reinforcement learning. https://doi.org/10.48550/arXiv.1511.06581.

[22] Ioannis Antonoglou David Silver Tom Schaul, John Quan. Prioritized experience replay. ICLR 2016, https://doi.org/10.48550/arXiv.1511.05952.

[23] Richard S. Sutton. Learning to predict by the methods of temporal differences. Mach Learn 3, 9–44 (1988), https://doi.org/10.1007/BF00115009.

[24] Rémi Munos Marc G. Bellemare, Will Dabney. A distributional perspective on reinforcement learning. https://doi.org/10.48550/arXiv.1707.06887.

[25] Marc G. Bellemare Clare Lyle, Pablo Samuel Castro. A comparative analysis of expected and distributional reinforcement learning. to appear in the Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, https://doi.org/10.48550/arXiv.1901.11084.

[26] Bilal Piot Jacob Menick Ian Osband Alex Graves Vlad Mnih Remi Munos Demis Hassabis Olivier Pietquin Charles Blundell Shane Legg Meire Fortunato, Mohammad Gheshlaghi Azar. Noisy networks for exploration. Jun 2017, ICML 2018, https://doi.org/10.48550/arXiv.1706.10295.

[27] Hado van Hasselt Tom Schaul Georg Ostrovski Will Dabney Dan Horgan Bilal Piot Mohammad Azar David Silver Matteo Hessel, Joseph Modayil. Rainbow: Combining improvements in deep reinforcement learning. AAAI 2017, https://doi.org/10.48550/arXiv.1710.02298.

[28] Mehdi Mirza Alex Graves Timothy P. Lillicrap Tim Harley David Silver Koray Kavukcuoglu Volodymyr Mnih, Adrià Puigdomènech Badia. Asynchronous methods for deep reinforcement learning. ICML 2016, https://doi.org/10.48550/arXiv.1602.01783.

[29] William Yang Wang, Jiwei Li, and Xiaodong He. Deep reinforcement learning for NLP. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, pages 19–21, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[30] Anabel Martin-Gonzalez Cornelius Weber Stefan Wermter Victor Uc-Cetina, Nicolas Navarro-Guerrero. Survey on reinforcement learning for language processing. Apr 2021, Artificial Intelligence Review 2022, https://doi.org/10.1007/s10462-022-10205-5.

[31] Nick Ryder Melanie Subbiah Jared Kaplan Prafulla Dhariwal Arvind Neelakantan Pranav Shyam Girish Sastry Amanda Askell Sandhini Agarwal Ariel Herbert-Voss Gretchen Krueger Tom Henighan Rewon Child Aditya Ramesh Daniel M. Ziegler Jeffrey Wu Clemens Winter Christopher Hesse Mark Chen Eric Sigler Mateusz Litwin Scott Gray Benjamin Chess Jack Clark Christopher Berner Sam McCandlish Alec Radford Ilya Sutskever Dario Amodei Tom B. Brown, Benjamin Mann. Language models are few-shot learners. May 2020, https://doi.org/10.48550/arXiv.2005.14165.

[32] John Wilhelm Balint Gyevnar Francisco Eiras Mihai Dobre Subramanian Ramamoorthy Stefano V. Albrecht, Cillian Brewitt. Interpretable goal-based prediction and planning for autonomous driving. IEEE International Conference on Robotics and Automation (ICRA), 2021, https://doi.org/10.48550/arXiv.2002.02277.

[33] Etienne Perot Senthil Yogamani Ahmad El Sallab, Mohammed Abdou. End-to-end deep reinforcement learning for lane keeping assist. Presented at the Machine Learning for Intelligent Transportation Systems Workshop, NIPS 2016, https://doi.org/10.48550/arXiv.1612.04340.

[34] Etienne Perot Senthil Yogamani Ahmad El Sallab, Mohammed Abdou. Deep reinforcement learning framework for autonomous driving. Reprinted with permission of IST: The Society for Imaging Science and Technology, sole copyright owners of Electronic Imaging, Autonomous Vehicles and Machines 2017, IST Electronic Imaging, Autonomous Vehicles and Machines 2017, AVM-023, pg. 70-76 (2017), https://doi.org/10.2352/ISSN.2470-1173.2017.19.AVM-023.

[35] Adham Ghazali Majed Jubeh Ariel Keselman, Sergey Ten. Reinforcement learning with a* and a deep heuristic. Nov 2018, https://doi.org/10.48550/arXiv.1811.07745.

[36] Chelsea Finn Mrinal Kalakrishnan Peter Pastor Sergey Levine Julian Ibarz, Jie Tan. How to train your robot with deep reinforcement learning; lessons we've learned. Journal of Robotics Research (IJRR), February 2021, https://doi.org/10.1177/0278364920987859.

[37] Philippe Zanne Michel de Mathelin Birgitta Dresp-Langley Rongrong Liu, Florent Nageotte. Deep reinforcement learning for the control of robotic manipulation: A focussed mini-review. Robotics, 2021, 10, 1, 22, https://doi.org/10.3390/robotics10010022.

[38] Timothy Lillicrap Sergey Levine Shixiang Gu, Ethan Holly. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. Oct 2016, https://doi.org/10.48550/arXiv.1610.00633.

[39] Hai Nguyen and Hung La. Review of deep reinforcement learning for robot manipulation. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 590–595, 2019.

APPENDIX

A Taxonomy of Deep Reinforcement Learning

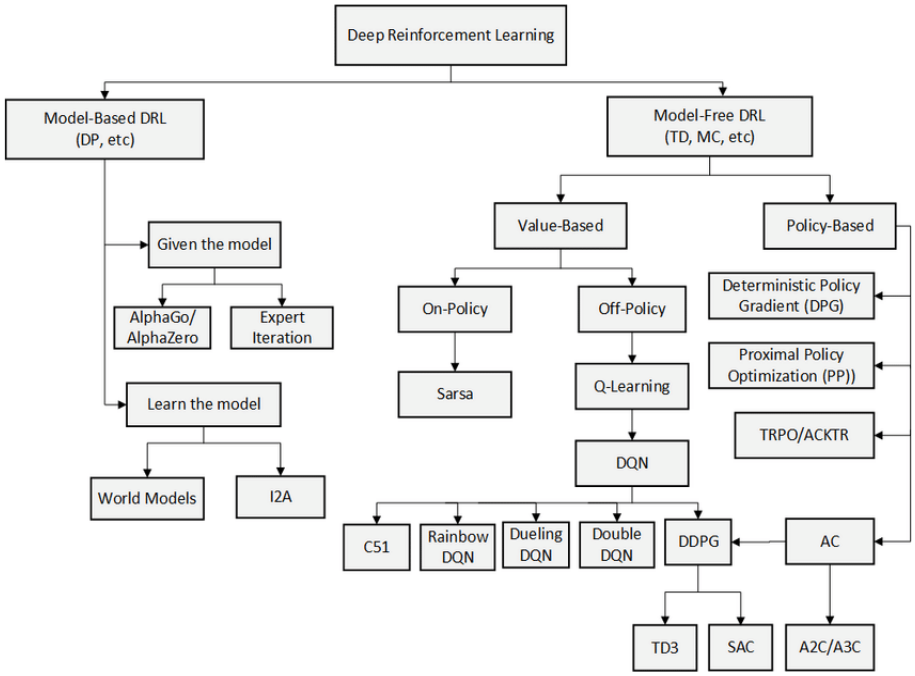

Figure 2: Categorization of Deep Reinforcement Learning techniques[11]

B DQN variants Graph Comparison

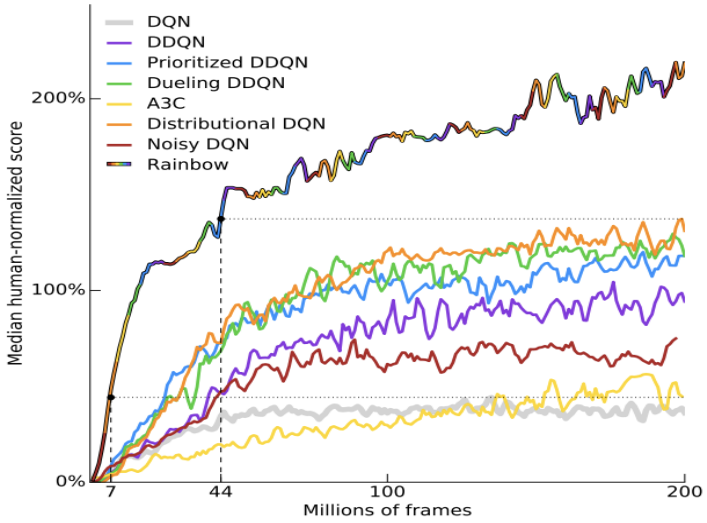

Figure 3: Median human-normalized performance across 57 Atari games. Comparison of the seven discussed algorithms is shown with the Rainbow DQN outperforming the other DQN variants. None that A3C and Distributional DQN correspond to the Multi-step learning and C51 algorithms respectively [27]