Deep Reinforcement Learning based Video Games: A Review

Kamal A. ElDahshan
Mathematics Department
Faculty of Science
Cairo, Egypt
dahshan@gmail.com

Hesham Farouk
Computers and Systems Dept.
Electronic Research Institute
Cairo, Egypt
Heshamali68@hotmail.com

Eslam Mofreh
Mathematics Department
Faculty of Science
Cairo, Egypt
eslammofreh@azhar.edu.eg

2022 2nd International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC) | 978-1-6654-6677-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/MIUCC55081.2022.9781752

***Abstract*—Video game development is getting increasingly effective as AI paradigms advance. Deep Reinforcement Learning (DRL) is a promising artificial intelligence (AI) approach. It broadens the reinforcement learning paradigm by making use of the complicated representation of deep neural networks. According to recent research, DRL performs extraordinarily well across a variety of sectors, including healthcare, video games, robotics, finance, and medicine. In this article, we provide a complete review of current and cutting-edge research developments in deep reinforcement learning in video games. This review starts by discussing the principles of reinforcement and deep reinforcement learning. Then investigate and assess essential strategies such as value-based, policy gradient, and model-based algorithms in this work. Finally, discusses a set of research challenges of DRL in video games.**

Keywords—Deep Learning, DQN, Policy Gradient, Deep Reinforcement Learning

I. INTRODUCTION

Game development requires a significant amount of time and effort. Game environments, narrative, and character behavior are methodically crafted, requiring teamwork among graphic designers, storytellers, and software engineers. The behavior of a game is typically a sophisticated blend of ingrained in the form of traditional code and more responsive behavior in the form of enormous sets of rules. In recent years, high-density data in machine learning technologies have fiddled rule-based systems obsolete. Unity has investigated a number of these technologies, including deep learning in content production and deep reinforcement learning in game development. Such difficulties hold a great deal of promise for machine learning and artificial intelligence.

Machine learning is being used in game creation at a rapid rate. Many techniques to video game development have been created that cover a wide range of problematic areas. Machine Learning is an excellent tool for developing more realistic landscapes, fascinating challenges, one-of-a-kind content, and researching player behavior. Instead of depending on pre-written scripts, it enables a gaming agent to respond to activities in real-time. A video game that use Machine Learning to assist the agent might help the agent react dynamically and adjust the game environment based on the game agent's decisions and actions. This enables video agents using Machine Learning to interact with the player in unique ways.

Machine learning algorithms focus on two main areas: playing against human creatures and creating effective game material for game players. Controlling a non-character player who is capable of playing against humans may be a difficult task, especially in a fast-changing gaming environment. The operator must react in near real-time, which necessitates rapid and effective planning. Machine Learning algorithms are capable of addressing the aforementioned difficulties in a variety of ways. 1. Possession of a Non-Character Agent 2. Modeling of Complicated Environments 3. Dynamic and Real-time Interaction 4. Content Creation on Autopilot.

Deep learning (DL) has made significant progress in computer vision in recent years. Deep reinforcement learning (DRL) helps agents to make all-through decisions in a high-dimensional state space, significantly improving the generalization and scalability of classic reinforcement learning (RL) paradigms [1].

DRL has made major progress in video games, including Dota2, ViZDoom, Atari, StarCraft, and others. There are various related works that introduce these achievements in this sector.

Zhang et al. [2] reviewed DRL research history, concentrating on the Monte Carlo method, the Q-Learning algorithm, and the deep Q network. DeepMind advanced contextual bandits by publishing results from a system that combines deep neural networks with large-scale reinforcement learning in early 2015. This system can master a broad range of Atari 2600 games to superhuman levels using only raw pixels and scores as inputs. DeepMind took the exploration vs. exploitation debate to new heights. While contextual bandits have little capacity to learn behavior, deep reinforcement learning may learn action sequences that optimize a future cumulative reward. In other words, they may be taught to engage in long-term value-maximizing conduct (LTV). LTV appeared in a variety of Atari games as strategy development, which is often reserved for human players. Consider how long it would take a chicken to learn to cross a big highway while collecting gift presents without being killed by an incoming truck. We used a broad reinforcement learning method similar to that used in the DeepMind trials to give the chicken a positive reward for picking up a gift box and a negative incentive for being struck by a truck. We also provided the chicken four movement options: left, right, forward, and backward. The chicken achieves superhuman abilities in just under six hours of training, utilizing simply raw pixels and score as input, as well as some relatively simple instructions.

Fenjiro and Benbrahim [3] emphasizes major characteristics, critical approaches, and a wide range of applications while offering an update on current DRL advancements. Arulkumaran et al. [4] provide a high-level overview of DRL, encompassing central algorithms and a wide range of visual RL domains.

978-1-6654-6677-6/22/$31.00 ©2022 IEEE

The sections that follow will be structured as follows. Section-2 will provide a quick review of typical reinforcement learning and deep reinforcement learning methods used in video games. Following that, in section-3, the platforms employed in video game research will be examined. Finally, section-4 will provide an overview of the DRL problems in preparation for future contributions.

II. DEEP REINFORCEMENT LEARNING

A. Reinforcement Learning

Reinforcement Learning (RL) is one of the machine learning approach that is concerned with making a series of decisions until an objective is reached. RL trains an intelligent agent by trial and error, rewarding good behavior and penalizing poor behavior. Unlike supervised learning, RL does not give direct supervision by providing the agent with an objective to reach, but rather teaches the agent how to achieve the aim. The primary distinction between supervised and unsupervised learning is that reinforcement learning learns through interactions with the environment and approximates some return on these activities to help reinforce its judgements in that environment. The entire system can be represented as a discrete-time Markov decision process (t=0,1,2,3,...), in which an agent observes the state of the environment s t at time step t and acts on that observation. When the agent completes the activity in time step t+1, the environment rewards it with a numerical reward r_{t+1}, and the environment state changes from s_t to s_{t+1} a result, a whole interaction can be described as a series of events known as a trajectory:

$$\tau = s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, \ldots\ldots$$

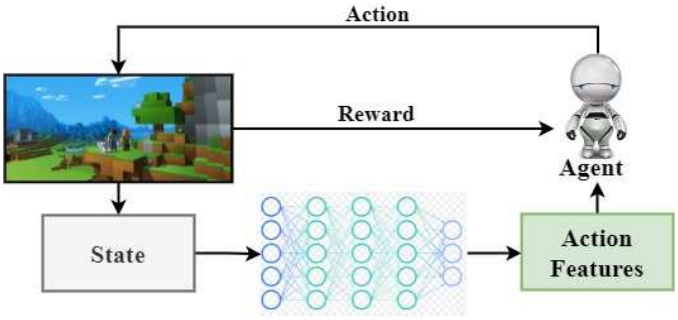

Figure.1 A DRL framework diagram for video games. The action features are automatically extracted using a deep learning model supplied with input data from the video game. Agents undertake actions based on these features and transition environments to the next

As illustrated in figure-1, a typical DRL framework [1] utilizes a Markov Decision Process (MDP) (S, A, γ, P, R) in which an agent interacts with the environment by performing an action a_t in a given state s_t at time t. The agent receives a reward r_t and travels to a different state in the environmentss_{t+1}. The goal is to maximize the overall predicted benefit gained while interacting with the environment.

$$V_\pi(s) = E_\pi\left[\sum_i^\infty \gamma^i r_{i+1}\,; s_t = s\right] \qquad (1)$$

The expected return for each state-action combination is represented by the action value function (Q-function). It specifies a value for each action in a given state, resulting in a mapping from states to actions. Equation-2 demonstrates how to define the Q-function.

$$Q_\pi(s,a) = E^\pi[\sum_i^\infty \gamma^i r_{i+1}\,; s_t = s, a_t = a] \qquad (2)$$

Such that Q (s, a) is the Q-values for state, s, and action, a, which equals the immediate reward plus discounted future rewards.
The relationship between V and Q values can be stated in equation-3.

$$V(s) = max_{a\in A} Q(s,a) \qquad (3)$$

The aforementioned Q-value is used in Q-learning algorithm. Q-learning is a tabular based algorithm iterating over every state in the state space aiming to obtain a map from the set of states to the set of actions.
The Q-learning utilizes Bellman equation in approximating the Q-value through using formula in equation-4.

$$Q(s,a) \longleftarrow r + max_{a'\in A} Q(s',a') \qquad (4)$$

Such that s', a' are next state and action that leads to max Q-value.

B. Deep Reinforcement Learning Algorithms:

Deep Learning techniques have been incorporated alongside RL to form Deep Reinforcement Learning (DRL). Tabular Q-learning works well when the state space is small. It does not appear to be employed in more realistic settings, such as Atari games, with unbounded state space. Because of the infinite state space, which is always a set of images representing the game states, this situation can be handled by using a non-linear representation between the states and actions and treating it as a regression problem in which the states are fed to a model that outputs an action probability distribution. A Deep Convolutional Neural Network (CNN) is the appropriate answer for such problems.

DRL approaches are broadly classified into two types; 1) value-based and 2) policy-based.

1) Value-Based:

The value-based class of algorithms seeks to generate a value function that can then be used to build a policy. One of the most basic value-based algorithms is the Q-learning algorithm. It utilizes the Q-value function, and the Q learning algorithm applies the Bellman equation of the Q-value function. The Deep Q-network (DQN) [14], is the most well-known DRL model. It employs a neural network as a function approximator. It directly learns rules from high-dimensional inputs. It accepts raw pixels and returns a value function to predict future rewards, as shown in figure-2 (ii). With the experience replay strategy, DQN breaks the sample correlation and stabilizes the learning process with a target Q-network. When playing ATARI games from pixels, the DQN approach, use neural networks as function approximators to achieve superhuman level control. There are many uses of DQN in the context of video games development [5-7]

Several improvements to the DQN have been created to manage different difficult parts of the action values estimation by altering the model structure, loss function, or sampling strategy of the experience reply, with the primary objective of enhancing processing speed and model action values estimation. Equation-5 describes the DQN algorithm's updating rule.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t)), \quad (5)$$

where $Q(s_t, a_t)$ is the Q-values for state, s_t, next state, s_{t+1}, immediate reward, R_{t+1}, and action, a_t, at time t.

Zhikang and Masahito [8], improved DQN convergence by using Fitted Value Iteration (FVI) to cast the DQN and using a customized loss function that is the maximum loss over the Mean Squared Bellman Error (MSBE) and the Mean Square Error.

Hasselt et al. [9] used twofold estimation of the target to solve the problem of DQN overestimation of action values. They demonstrated that if there are errors in the estimation of Q-values, max will be biased, resulting in overestimation of Q-values. Double Q-learning is used in a Double Deep Q-Network, or Double DQN, to reduce overestimation by dividing the target's max operation into action selection and action assessment. We assess the greedy strategy using the online network, but we quantify its value using the target network. The modification is the same as for DQN, however the target Y^{target} has been changed to be as given by equation-6.

$$Y^{target} = R_{t+1} + \gamma\, Q'(s_{t+1}, a), \quad (6)$$
$$where\ a = max_{a \in A} Q'(s_{t+1}, a)$$

The update rule differs slightly from the DQN algorithm and is given by equation-7:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(R_{t+1} + \gamma\, Q'^{(s_{t+1}, a)} - Q(s_t, a_t)\right), \quad (7)$$

where Q and Q′ are two action-value estimators and α is the update ratio or learning rate.

Zhang and Liu [10], proposed sampling each n steps and computing goal of action values to speed up DQN. They also addressed the DQN overestimation problem by inventing Semi-Double DQN, an approach that combines DQN and Double DQN.

Dueling DQN architecture uses two different streams to explicitly segregate the representation of state values and action state-dependence advantages as shown in figure-2 (iii). The first stream computes the state-value function $V(s)$ whereas the second computes the action-advantage function $A^{\pi}(s, a)$. These streams' outputs are combined to form output $Q^{\pi}(s, a)$ such that $Q^{\pi}(s, a) = V^{\pi}(s) + A^{\pi}(s, a)$. The main motivation for this architecture is because in some games, knowing the value of each action at each time step is redundant. The authors use the Atari game Enduro as an example, where you don't have to know which action to take until a collision is imminent.

Fortuato et al. [11], have proposed NoisyNet DQN which has treated the agent exploration-exploitation in a different way; the agent explores the environment, it gives noise to the output layer of the network. NoisyNet DQN is a DQN variation that drives exploration by using fully connected layers with noisy parameters. As a result, the parametrized action-value function is now a random variable. It demonstrates how the agent's policy's induced stochasticity can be leveraged to promote efficient exploration. A noisy parameter θ is defined by $\theta := \mu + \Sigma \odot \epsilon$, where Σ and μ are trainable parameter vectors, and ϵ is a vector of zero mean noise. As a result, the loss function is now specified in terms of Σ and μ and the optimization is now done in terms of Σ and μ. ϵ is derived from factorized Gaussian noise and sampled.

Prioritized Experience Reply (PER) prioritizes samples depending on their relevance, such as when projected Q-values in the future state differ from those in the present state, which results in a high level of significance. This priority can result in a loss of variety, which can be mitigated by stochastic prioritization, as well as bias, which can be mitigated by importance sampling.

Zhu et al. [12] proposed an enhanced Dueling DQN network with prioritized experience replay to tackle the delayed convergence of the regular DQN.

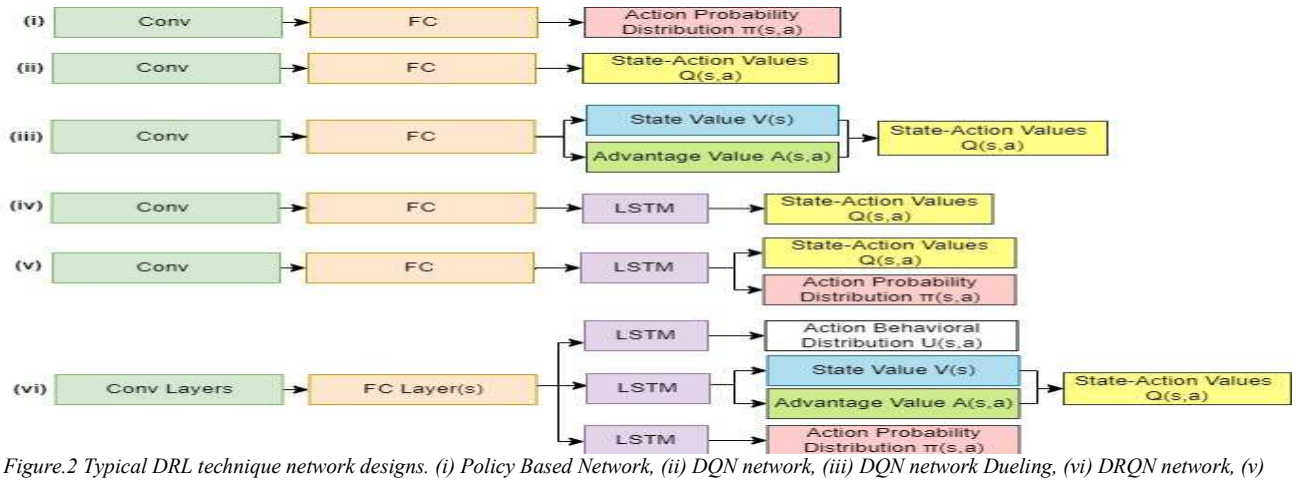

Figure.2 Typical DRL technique network designs. (i) Policy Based Network, (ii) DQN network, (iii) DQN network Dueling, (vi) DRQN network, (v) Actor-critic network, (iv) Reactor network.

The stochastic sampling approach interpolates between pure greedy prioritizing and uniform random sampling. The probability of being sampled increases monotonically as the priority of a transition increases, with a non-zero probability even for the lowest-priority transition. The probability of sampling transition i is defined as: $(i) = \frac{p_i^{\propto}}{\sum_k p_k^{\propto}}$,
where $p_i > 0$ is the i[th] transition priority and $\propto$ determines the amount of prioritization used (with $\propto = 0$ for the uniform case). Prioritized replay increases bias because it modifies the distribution in an unplanned way, changing the solution to which the estimates converge. Using importance-sampling (IS) weights, we can correct this bias: $w_i = (\frac{1}{N} \cdot \frac{1}{P(i)})^{\beta}$
If β = 1, this fully accounts for non-uniform probability P(i). By utilizing $w_i \delta_i$ instead of δ_i - weighted IS rather than conventional IS - these weights can be integrated into the Q-learning update. We always normalize weights by 1/w for stability reasons, such that they only scale the update downwards.
To maintain maximizing bias in DQN, Zhu and Rigotti [13] presented an unbiased set of estimators derived from two independent and unbiased estimators.

Rainbow DQN is an expanded DQN proposed by Hessel et al. [14], which integrates numerous enhancements into a single learner. It employs Double Q-Learning to combat overestimation bias, Prioritized Experience Replay to prioritize key transitions. It uses Dueling networks and employs a multi-step learning method. It also instead of using the expected return, it employs distributional reinforcement learning and for exploration, it employs noisy linear layers.

2) Policy-Based:

The key distinction between value-based and policy-based approaches is that in value-based methods, we construct a value function that is then utilized to produce the policy, either explicitly or implicitly. With contrast, there is no need to determine the value function in policy-based techniques, and we learn the policy immediately. In policy-based approaches, the value function is essentially non-existent. Furthermore, value-based strategies would fail in the presence of a continuous or huge discrete action space since it would be difficult to define the greedy policy. Policy-based algorithms that learn a parameterized policy that allows actions to be done without validating the Q-values have been presented as a solution to this problem. Policy-based strategies can be used to learn particular probabilities for carrying out actions.

Policy gradients are a policy-based strategy. We employ a stochastic policy $\pi_\theta(\mathrm{a}|\mathrm{s})$ parameterized by some θ for policy gradients, which gives us the likelihood of taking action an in state s. We utilize a neural network to calculate this policy. This policy network uses a frame from our game to generate a probability distribution for the actions (for discrete actions). The stochastic policy $\pi_\theta(\mathrm{a}|\mathrm{s})$ is computed by our policy network, where θ are the neural network parameters. We need a quality function to tell us how good our policy is in order to improve it. This policy score function is reliant on the environment we're in, but the predicted reward is a good indicator of policy soundness in general. The policy score $J(\theta)$ is defined as the expected total reward following policy π_θ and is given by equation-

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T} r_{t+1} |\pi_\theta] \quad (8)$$

As a result, the best parameter vector θ^* for this policy will maximize the predicted reward value, such as

$$\theta^* = argmax_\theta J(\theta)$$

By continually calculating the gradient $\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}[\sum_{t=0}^{\infty} r_t |\pi_\theta]$, policy gradient approaches optimize the predicted total reward. The policy gradient can be expressed in a variety of ways, all of which have the form represented by equation-9.

$$\nabla_\theta J(\theta) = \sum_{t=0}^{\infty} \Omega_t \nabla_\theta \log \pi_\theta(a_t|s_t), \quad (9)$$

where s_t is the state at time t, $\pi_\theta(a_t|s_t)$ is the probability of occurrence of s_t , a_t and Ω_t can take one of the following:

1. Trajectory total reward $(\sum_{t=0}^{\infty} r_t)$
2. Reward following action a_t $(\sum_{t\prime=t}^{\infty} r_t)$
3. Base-lined Reward following action a_t $(\sum_{t\prime=t}^{\infty} r_{t\prime} - b(s_t))$
4. State-action value function $(Q^\pi(s_t, a_t))$
5. Advantage function $(A^\pi(s_t, a_t))$
6. Temporal Difference (TD) residual $(r_t + V^\pi(s_{t+1}) - V^\pi(s_t))$

Such that:

$$V^\pi(s_t) = \mathbb{E}_{\substack{s_{t+1:\infty}, \\ a_t:\infty}}[\sum_{l=0}^{\infty} r_{t+l}], \quad (10)$$

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t), \quad (11)$$

$$Q^\pi(s_t, a_t) = \mathbb{E}_{\substack{s_{t+1:\infty}, \\ a_{t+1}:\infty}}[\sum_{l=0}^{\infty} r_{t+l}] \quad (12)$$

a) On-Policy Policy Gradient:

In on-policy algorithms, the target policy is used to sample actions, and the same policy is used to optimize for. REINFORCE and typical actor-critic algorithms are examples of on-policy techniques. In the off-policy approach, actions are sampled using behavior policy, and a different target policy is used to optimize for in the off-policy method.

The REINFORCE Algorithm:

Monte Carlo sampling could be used to solve Equation-9. Several trials are simulated according to a predefined policy, and data from these experiments is gathered via Monte Carlo sampling. As a result, the REINFORCE algorithm [15] is often referred to as the Monte Carlo policy gradient approach, or simply Monte Carlo Policy Gradient. Using its current policy, the agent collects the trajectory of a single episode and uses it to update the policy parameter. REINFORCE is updated in an ad hoc way since generating a sample space necessitates the use of a full trajectory.

Algorithm 1: REINFORCE

function REINFORCE
  Initialize arbitrarily
  for each episode $\{(\boldsymbol{s_1}, \boldsymbol{a_1}, \boldsymbol{r_2}), \ldots, (\boldsymbol{s_{T-1}}, \boldsymbol{a_{T-1}}, \boldsymbol{r_T})\} \sim \boldsymbol{\pi_\theta}$ **do**
    for t = 1 to T-1 **do**
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \propto \boldsymbol{\nabla_\theta} \boldsymbol{\log \pi_\theta(s_t, a_t)}\ \boldsymbol{v_t}$
    end for
  end for
  return $\boldsymbol{\theta}$
end function

where v_t is the unbiased estimate sample of $Q_{\pi_\theta}(s_t, a_t)$, and $\propto$ is the step size.

The REINFORCE technique is not employed in practice, despite considerable mathematical simplifications and computational advancements. This is due to the REINFORCE method's abnormally broad variation of the gradient. The method in which the REINFORCE awards are applied may explain why there is such a wide range of variety. REINFORCE employs absolute incentives, thus the payouts might vary greatly depending on the Monte Carlo experiment. As a result, the calculated gradient has an extremely large variance.

The substantial fluctuation was caused mostly by our inability to predictably and precisely determine which behaviors contributed to the reward in a given trajectory. As a result, we were unable to positively shift the gradient such that the subsequently amended policy could encourage the action with the highest reward while narrowing the gradient to penalize the less rewarding ones.

Actor-Critic:

Figure 2 displays the policy model and the value function, which are both critical components of policy gradient (v). Learning the value function as well as the policy adds a lot of sense to the learning technique since valuing actions may aid in policy updates by, for example, decreasing gradient variance in vanilla policy gradients. That is exactly what the Actor-Critic approach does. In Actor-Critic techniques, the policy gradient methodology is utilized in conjunction with TD learning, with the actor responsible for learning a policy $\pi_\theta(a_t|s_t)$ using the policy gradient and the critic responsible for learning how to approximate R using TD-learning. Actor-critic approaches are made up of two models that can share parameters: Critic changes the value function parameters ω, which might be action-value $Q^\omega(s_t, a_t)$ or state-value $V^\omega(s_t)$ depending on the algorithm (s). The policy parameters θ for $\pi_\theta(a_t|s_t)$ are updated by the actor in the direction recommended by the critic.

One-Step Bootstrapped Return - It is a single stage bootstrapped return, as represented by equation-13 that estimates the return using a bootstrapped value estimate of the next state in the trajectory.

$$G_t \cong R_{t+1} + \gamma V^\omega(s_{t+1}) \tag{13}$$

And the policy gradient is given is derived using equation-14:

$$\nabla_\theta \mathbb{E}_{\pi_\theta}[r(\tau)] = \mathbb{E}_{\pi_\theta}[(\sum_{t=0}^{\infty} (R_{t+1} + \gamma V^\omega(s_{t+1}) - V^\omega(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t))] \tag{14}$$

The critic objective function is given by: $J(\omega) = \frac{1}{2}(R_{t+1} + \gamma V^\omega(s_{t+1}) - V^\omega(s_t))^2$ and the gradient is given by equation-15.

$$\nabla_w J(\omega) = R_{t+1} + \gamma V^\omega(s_{t+1}) - V^\omega(s_t) \tag{15}$$

Other multi-threaded asynchronous SARSA, DQN, and Actor-Critic approaches can leverage multiple CPU threads, which reduces training time roughly linearly with the number of parallel threads [16]. The network is constantly being updated based on uncorrelated experiences from parallel actors. These versions do not rely on a replay memory, which helps to stabilize on-policy techniques.

b) Off-Policy Policy Gradient:

REINFORCE and the vanilla form of the actor-critic technique are both on-policy. The training samples in on-policy are collected in line with the target policy, which is the same policy that we are aiming to optimize for. Off-policy techniques, on the other hand, provide a variety of extra advantages. The off-policy approach requires no whole trajectories and may reuse any past episodes for much-increased sampling efficiency. The sample collection follows a distinct set of rules than the aim policy, allowing for greater research.

The behavior policy that is used for gathering samples is labelled as β(a|s), and it is a well-known policy (predefined as a hyper-parameter). The reward across the state distribution defined by this behavior policy is calculated using the objective function defined in equation-16.

$$J(\theta) = \sum_{s\in S} d^\beta(s) \sum_{a\in A} Q^\pi(s,a)\, \pi_\theta(a|s) = \mathbb{E}_{s\sim d^\beta}[\sum_{a\in A} Q^\pi(s,a)\, \pi_\theta(a|s)], \tag{16}$$

where $d^\beta(s)$ is the stationary distribution of the behavior policy β; such that: $d^\beta(s) = lim_{t\to\infty} P(S_t = s|S_0, \beta)$ and Q^π is the action-value function when applying policy π.

$$\nabla_\theta J(\theta) = \mathbb{E}_\beta[\frac{\pi_\theta(a|s)}{\beta(a|s)} Q^\pi(s,a) \nabla_\theta \log \pi_\theta(a|s)], \tag{17}$$

where $\frac{\pi_\theta(a|s)}{\beta(a|s)}$ is the importance weight.

Asynchronous Advantage Actor-Critic, or A3C, is a conventional policy gradient technique that emphasizes simultaneous training. The value function is taught to the critics in A3C, while several actors are taught in parallel and periodically synced with global parameters. As a consequence, A3C has been tuned for parallel training.

Because A2C is an asynchronous, deterministic variant of A3C, the initial character "A" of A3C has been removed to create A2C. Because each A3C agent communicates with be parameters individually, it is conceivable that particular agents are interacting with policies from various versions, resulting in a less optimal aggregated update. To address the inconsistency, an A2C coordinator waits for all parallel actors to complete their tasks before changing global parameters, and parallel actors then resume using the same policy in the next iteration. The synchronized gradient update improves training consistency and may speed faster convergence. As shown in equation-18, the first aspect of A2C is that it employs n-step updating, which is a compromise between MC and TD.

$$\nabla_\theta \mathbb{E}_{\pi_\theta}[r(\tau)] = \mathbb{E}_{s_t\sim p^\pi, a_t\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(s_t, a_t) (\sum_{k=0}^{n-1} (\gamma^k r_{t+k+1} + \gamma^n V^\omega(s_{t+n+1}) - V^\omega(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t))] \tag{18}$$

At the end of each episode, the MC adjusts the value of an action by utilizing the reward to-go (sum of earned rewards) R. (s,a). TD immediately updates the action with the immediate reward r(s,a,s′) and approximates the remaining with the value of the next state V(s).

The rest of the rewards are approximated with the value of the state visited n steps later by n-step.

PAAC [17] is a novel DRL parallelization framework that allows several actors to learn the policy on a single system.

Policy gradient approaches are effective instruments for policy improvement, however they are frequently on-policy and cannot employ off-policy data. PGQ [18], It is an innovative method that combines policy gradient with Q-learning to improve performance. PGQ is used to equalize regularized policy gradient techniques and advantage function learning algorithms. Retrace (λ) [19] combines the advantages of significance sampling, off-policy Q(λ), and tree-backup (λ) to provide low variance, safety, and performance. As demonstrated in figure-2 (vi), it combines dueling DRQN architecture with actor-critical architecture (e). Reactor [20] is an off-policy actor-critic architecture that employs a multistage return for an agent with an efficient sampling and numerical efficiency. The network generates a goal policy, an action-value Q-function, and an estimated behavioral policy, as illustrated in figure-2 (iv). The critic is trained using the off-policy multi-step Retrace technique, while the actor is trained using a leave-one-out policy gradient.

The IMPALA [21] is a revolutionary distributed DRL that has the potential to scale to thousands of computers. IMPALA utilizes a single reinforcement learning agent with a single set of parameters to handle a wide variety of tasks. This framework is an off-policy actor-critic that achieves learning stability by combining decoupled action and learning with a novel V-trace off-policy correction technique.

III. PLATFORMS FOR GAMING RESEARCH

Platforms and contests provide major contributions to the evolution of gaming AI and assist in the assessment of agents' intelligence. The Arcade Learning Environment (ALE) is the first DRL algorithm evaluation platform with an interface to a library of Atari 2600 games. Because it shows both game frames and signals such as player scores, ALE is a great framework. To aid DRL research, OpenAI has included a diverse set of reinforcement learning problems onto a platform called Gym [22], Atari, Classical Control, Algorithmic, Board games, and robots in both 2D and 3D are commonly seen in OpenAI Gym. Then there's OpenAI Universe, a game-based framework for testing and training general intelligent agents using the required tools. Gym Retro is a video game emulator wrapper with the same unified interface as Gym that enables Gym to be easily extended with a vast number of video games for RL study, including not just Atari but also Nintendo, Sega, and NEC.

The objective of General Video Game Playing is to construct an agent capable of playing a variety of video games without requiring human intervention. The General Video Game AI (GVGAI) competition [23] is being presented as a simple and open-source platform for assessing AI technologies, including DRL.

Malmo[24], a research platform for AI studies built on top of Minecraft, is a platform for AI research. It is a first-person 3D environment for multi-agent research in the Microsoft Malmo collaborative AI challenge 2017 and multi-agent RL in the MalmO competition 2018. TORCS [25] is a self-driving automobile racing simulator using DRL that incorporates both low-level and visual components.

Atari:

ALE is a development environment for constructing general intelligence agents for hundreds of Atari 2600 games. As the most extensively used DRL research testbed, a wide range of DRL techniques have consistently performed well. Machado et al. [26] investigate the ALE in the DRL research community, presenting a number of assessment methodologies as well as some main issues. Varghese et al [27] experimented with a proposed DRL approach on Atari games in OpenAI Gym.

ViZDoom:

First-person shooter (FPS) games are critical in game AI research. Khan et al [28] investigated DQN in FPS.

Doom is a classic first-person shooter game, and ViZDoom is available as a one-of-a-kind DRL testbed [29]. Agents learn from visual inputs and interact with the ViZDoom world in first-person mode.

TORCS:

TORCS is a racing game in which you may steer, brake, and accelerate. This game has more realistic graphics than Atari games, but agents must learn the mechanics of the automobile. FIGAR-DDPG may accomplish the racing task and 20 laps of the circuit, with a total reward ten times that of DDPG and significantly smoother policies. Normalized Actor-Critic (NAC) effectively normalizes the Q-function and learns an initial policy network from demonstration before improving it in a real-world environment [30].

DeepMind:

DeepMind Lab is a Quake3-based 3D first-person gaming platform that has been extended from OpenArena. DeepMind Lab has substantially superior visuals and more realistic physics than other first-person gaming platforms, making it a lot more complex platform. On a rigorous series of DeepMind lab tests, the UNREAL agent exceeds A3C by a factor of 10, averaging 87 percent expert human performance. Continuous players battle against one another. A team of learning has recently made 9 fast progress as learning agents get more powerful. In the DeepMind lab, Mankowitz et al. [31] investigate an implicit sequence of tasks with sparse rewards at the DeepMind lab to test continuous learning skills.

MOBA and Dota2:

Multiplayer Online Battle Arena (MOBA) games are based on real-time strategy in which two teams of five players must work together to beat the opponent by defeating adversaries, upgrading heroes, and eventually destroying the opponent's base. Because MOBA research is still in its early stages, there are less works available than for typical RTS games. King of Glory is the most popular mobile-end MOBA game in China (a simplified mobile version of Dota). In this game, Jiang et al. [32] use Monte-Carlo Tree Search and deep neural networks.

IV. DRL CHALLENGES IN VIDEO GAMES

1) Environment Generalization

Intelligent agents' capacity to transmit information across diverse settings is seen as a crucial feature. Multi-task learning and policy distillation have been focused on these instances in order to promote generalization performance in different settings. The generalization problem can be overcome utilizing common neural network parameters in multi-task learning, and efficiency may be boosted by transferring across related tasks. Raileanu and Fergus [33]

studied the generalization problem by separating the optimization of the policy and value function, modeling those separately using distinct networks, and imposing a loss that promotes the representation to be invariant to environmental factors. IMPALA [21] demonstrates the efficacy of multi-task reinforcement learning with less data and positive transfer between tasks.

2) Exploration-Exploitation Dilemma

Exploration can aid in the acquisition of more diverse samples, whereas exploitation is a method of learning the high reward policy using valuable samples. For RL, balancing exploration and exploitation is still a key difficulty. Common exploration methods need a considerable quantity of data and are incapable of dealing with temporally prolonged investigation. In complex situations, most model-free RL methods are computationally intractable. In the training phase, parametric noise may greatly aid exploration [11]. Yao et al. [34] proposed an approach that addresses the exploration-exploitation riddle by combining optimistic exploration with weighted exploitation. Their method chooses the best action that maximizes the expected accumulative reward by considering both predicted return and future observation uniqueness.

3) Multi-Agent Environment

Multi-agent learning is critical in video games such as StarCraft. In a cooperative multi-agent scenario, the curse of dimensionality, credit assignment, and communication are all critical challenges. Cooperative learning in a multi-agent system utilizes a single learner to learn joint solutions, whereas concurrent learning uses several learners for each agent. Centralized policy training has lately emerged as a common multi-agent training paradigm. Multi-agent considers the activities of other agents and can learn complex multi-agent coordinating behavior[35]. To handle the multi-agent credit assignment problem, hypothetical multi agent policy gradients utilizes a centralized critic to estimate the action-value function and dispersed actors to optimize each agent's policies, with a hypothetical advantage function.

4) Environment Sampling

To attain human-level performance, DRL algorithms often require millions of samples. Humans, on the other hand, can quickly master extremely rewarding environmental behaviors. Most model-free DRL algorithms waste a lot of data, especially in environments with a lot of dimensions and explore space. They must interact with the environment at a high time cost in order to seek high reward events in a complicated sample space, restricting their applicability to a range of scenarios. Some data efficiency strategies, such as hierarchy and demonstration, can be used to reduce the exploratory dimension of the environment and reduce the amount of time spent on interaction.

5) Imitation Learning

Imitation learning, tries to train a policy to replicate the behavior of an expert based on samples obtained from that expert. An agent is trained to do a task by learning a mapping between observations and actions. The concept of teaching by imitation has been around for a long time, but the area has lately gained prominence due to breakthroughs in computing and sensing, as well as increased need for intelligent applications. Chen et al. [36] proposed an approach to imitation learning that aims for simplicity as well as performance. Their method use the V function to identify high-performing actions, which are subsequently used to train a policy network via imitation learning.

6) Inverse DRL

DRL has been effectively applied in domains where the reward function is well stated. However, in real-world applications, this is limited since it requires information from other domains that is not always available. Inverse DRL is a unique example of imitation learning. In autonomous driving, for example, the reward function should contain all factors such as driver behavior, gas usage, time, speed, safety, driving quality, and so on. In the real world, controlling all of these factors is hard and challenging [37].

7) Meta DRL

Meta Reinforcement Learning, in a nutshell, is meta-learning within the context of reinforcement learning. The train and test tasks are typically distinct but drawn from the same family of problems; for example, experiments in the papers included multi-armed bandits with varying reward probabilities, mazes with varying layouts, the same robots but with varying physical parameters in the simulator, and many others. DRL algorithms, as previously stated, take a significant amount of experience to learn a single task and are unable to generalize the learned policy to fresh tasks [38].

CONCLUSION

Deep reinforcement learning is a tough yet promising AI technique in gaming technology. The development of video games has benefitted from recent huge breakthroughs in artificial intelligence research. In this study, we looked at the achievements of deep reinforcement learning in video games. Different DRL techniques are examined, as well as their successful implementations in video games ranging from single-agent to multi-agent. The DRL research platforms for video games are also examined and addressed. Finally, a series of DRL issues and open paths for video games are explored.

ACKNOWLEDGMENT

This paper is based upon work supported by the Academy of Scientific Research and Technology (ASRT) under grant for a project named "Child Tut".

REFERENCES

[1] N. T. H. Le, V. S. Rathour, K. Yamazaki, K. Luu, and M. Savvides, "Deep Reinforcement Learning in Computer Vision: A Comprehensive Survey," *Artif. Intell. Rev.,* vol. 55, pp. 2733-2819, 2022.

[2] J. Zhang, C. Zhang, and W.-C. Chien, "Overview of deep reinforcement learning improvements and applications," *Journal of Internet Technology,* vol. 22, no. 2, pp. 239-255, 2021.

[3] Y. Fenjiro and H. Benbrahim, "Deep Reinforcement Learning Overview of the state of the Art," *Journal of Automation, Mobile Robotics and Intelligent Systems,* vol. 12, pp. 20-39, 12/01 2018.

[4] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Processing Magazine,* vol. 34, no. 6, pp. 26-38, 2017.

[5] X. Zhao, S. Ding, Y. An, and W. Jia, "Applications of asynchronous deep reinforcement learning based on dynamic updating weights," *Applied Intelligence,* vol. 49, no. 2, pp. 581-591, 2019/02/01 2019.

[6] Z. Ren, D. Dong, H. Li, and C. Chen, "Self-Paced Prioritized Curriculum Learning With Coverage Penalty in Deep Reinforcement Learning," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 29, no. 6, pp. 2216-2226, 2018.

[7] H. Chae, C. M. Kang, B. Kim, J. Kim, C. C. Chung, and J. W. Choi, "Autonomous braking system via deep reinforcement learning," in *2017 IEEE 20th International conference on intelligent transportation systems (ITSC)*, 2017: IEEE, pp. 1-6.

[8] Z. Wang and M. Ueda, "A Convergent and Efficient Deep Q Network Algorithm," *ArXiv,* vol. abs/2106.15419, 2021.

[9] H. v. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-Learning," presented at the Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, Phoenix, Arizona, 2016.

[10] Y. Zhang and C. Liu, "Accelerate deep Q-network learning by n-step backup," presented at the Proceedings of the 2nd International Conference on Innovation in Artificial Intelligence, Shanghai, China, 2018.

[11] M. Fortunato *et al.*, "Noisy Networks for Exploration," *ArXiv,* vol. abs/1706.10295, 2018.

[12] Z. Zhu, C. Hu, C. Zhu, Y. Zhu, and Y. Sheng, "An Improved Dueling Deep Double-Q Network Based on Prioritized Experience Replay for Path Planning of Unmanned Surface Vehicles," *Journal of Marine Science and Engineering,* vol. 9, no. 11, p. 1267, 2021.

[13] R. Zhu and M. Rigotti, "Self-correcting q-learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021, vol. 35, no. 12, pp. 11185-11192.

[14] M. Hessel *et al.*, "Rainbow: Combining improvements in deep reinforcement learning," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[15] R. J. Williams, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," *Machine Learning,* vol. 8, pp. 229-256, 2004.

[16] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016: PMLR, pp. 1928-1937.

[17] A. V. Clemente, H. N. Castejón, and A. Chandra, "Efficient parallel methods for deep reinforcement learning," *arXiv preprint arXiv:1705.04862,* 2017.

[18] B. O'Donoghue, R. Munos, K. Kavukcuoglu, and V. Mnih, "Combining policy gradient and Q-learning," *arXiv preprint arXiv:1611.01626,* 2016.

[19] R. Munos, T. Stepleton, A. Harutyunyan, and M. Bellemare, "Safe and efficient off-policy reinforcement learning," *Advances in neural information processing systems,* vol. 29, 2016.

[20] A. Gruslys, W. Dabney, M. G. Azar, B. Piot, M. Bellemare, and R. Munos, "The reactor: A fast and sample-efficient actor-critic agent for reinforcement learning," *arXiv preprint arXiv:1704.04651,* 2017.

[21] L. Espeholt *et al.*, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," in *International Conference on Machine Learning*, 2018: PMLR, pp. 1407-1416.

[22] G. Brockman *et al.*, "Openai gym," *arXiv preprint arXiv:1606.01540,* 2016.

[23] R. R. Torrado, P. Bontrager, J. Togelius, J. Liu, and D. Perez-Liebana, "Deep reinforcement learning for general video game ai," in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, 2018: IEEE, pp. 1-8.

[24] M. Johnson, K. Hofmann, T. Hutton, and D. Bignell, "The Malmo Platform for Artificial Intelligence Experimentation," in *IJCAI*, 2016: Citeseer, pp. 4246-4247.

[25] B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner, "Torcs, the open racing car simulator," *Software available at http://torcs. sourceforge. net,* vol. 4, no. 6, p. 2, 2000.

[26] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling, "Revisiting the arcade learning environment: evaluation protocols and open problems for general agents," *J. Artif. Int. Res.,* vol. 61, no. 1, pp. 523–562, 2018.

[27] N. V. Varghese and Q. H. Mahmoud, "A Hybrid Multi-Task Learning Approach for Optimizing Deep Reinforcement Learning Agents," *IEEE Access,* vol. 9, pp. 44681-44703, 2021.

[28] A. Khan, A. M. Khattak, M. Z. Asghar, M. Naeem, and A. U. Din, "Playing First-Person Perspective Games with Deep Reinforcement Learning Using the State-of-the-Art Game-AI Research Platforms," in *Deep Learning for Unmanned Systems*, A. Koubaa and A. T. Azar Eds. Cham: Springer International Publishing, 2021, pp. 635-667.

[29] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, "Vizdoom: A doom-based ai research platform for visual reinforcement learning," in *2016 IEEE conference on computational intelligence and games (CIG)*, 2016: IEEE, pp. 1-8.

[30] Y. Gao, H. Xu, J. Lin, F. Yu, S. Levine, and T. Darrell, "Reinforcement learning from imperfect demonstrations," *arXiv preprint arXiv:1802.05313,* 2018.

[31] D. J. Mankowitz *et al.*, "Unicorn: Continual learning with a universal, off-policy agent," *arXiv preprint arXiv:1802.08294,* 2018.

[32] D. Jiang, E. Ekwedike, and H. Liu, "Feedback-based tree search for reinforcement learning," in *International conference on machine learning*, 2018: PMLR, pp. 2284-2293.

[33] R. Raileanu and R. Fergus, "Decoupling value and policy for generalization in reinforcement learning," in *International Conference on Machine Learning*, 2021: PMLR, pp. 8787-8798.

[34] Y. Yao, L. Xiao, Z. An, W. Zhang, and D. Luo, "Sample Efficient Reinforcement Learning via Model-Ensemble Exploration and Exploitation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 30 May-5 June 2021 2021, pp. 4202-4208.

[35] G. Adams, S. Padmanabhan, and S. Shekhar, "Resolving Implicit Coordination in Multi-Agent Deep Reinforcement Learning with Deep Q-Networks & Game Theory," *ArXiv,* vol. abs/2012.09136, 2020.

[36] X. Chen, Z. Zhou, Z. Wang, C. Wang, Y. Wu, and K. Ross, "BAIL: Best-action imitation learning for batch deep reinforcement learning," *Advances in Neural Information Processing Systems,* vol. 33, pp. 18353-18363, 2020.

[37] X. Chen, L. Yao, A. Sun, X. Wang, X. Xu, and L. Zhu, "Generative inverse deep reinforcement learning for online recommendation," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 201-210.

[38] G. Qu, H. Wu, R. Li, and P. Jiao, "Dmro: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing," *IEEE Transactions on Network and Service Management,* vol. 18, no. 3, pp. 3448-3459, 2021.