Deep Learning for Video Game Playing

Niels Justesen, Philip Bontrager, Julian Togelius, and Sebastian Risi

***Abstract*—In this paper, we review recent deep learning advances in the context of how they have been applied to play different types of video games such as first-person shooters, arcade games, and real-time strategy games. We analyze the unique requirements that different game genres pose to a deep learning system and highlight important open challenges in the context of applying these machine learning methods to video games, such as general game playing, dealing with extremely large decision spaces and sparse rewards.**

***Index Terms*—Algorithms, learning, machine learning algorithms, multilayer neural network, artificial intelligence.**

I. Introduction

APPLYING artificial intelligence (AI) techniques to games is now an established research field with multiple conferences and dedicated journals. In this paper, we review recent advances in deep learning (DL) for video game playing and employed game research platforms while highlighting important open challenges. Our motivation for writing this paper is to review the field from the perspective of different types of games, the challenges they pose for DL, and how DL can be used to play these games. A variety of review articles on DL exist [39], [81], [126], as well as surveys on reinforcement learning (RL) [142] and deep RL [87]; here, we focus on these techniques applied to video game playing.

In particular, in this paper, we focus on game problems and environments that have been used extensively for DL-based Game AI, such as Atari/ALE, Doom, *Minecraft*, *StarCraft*, and car racing. Additionally, we review the existing work and point out important challenges that remain to be solved. We are interested in approaches that aim to play a particular *video game* well (in contrast to board games such as Go, etc.), from pixels or feature vectors, without an existing forward model. Several game genres are analyzed to point out the many and diverse challenges they pose to human and machine players.

It is important to note that there are many uses of AI in and for games that are not covered in this paper; Game AI is a large and diverse field [38], [93], [99], [170], [171]. This paper is focused on DL methods for playing video games well, while there is plenty of research on playing games in a believable, entertaining, or human-like manner [59]. AI is also used for modeling players' behavior, experience or preferences [169], or generating game content such as levels, textures, or rules [130]. DL is far from the only AI method used in games. Other prominent methods include Monte Carlo tree search [18] and evolutionary computation [90], [115]. In what follows, it is important to be aware of the limitations of the scope of this paper.

The rest of this paper is structured as follows. Section II gives an overview of different DL methods applied to games, followed by the different research platforms that are currently in use. Section IV reviews the use of DL methods in different video game types. Section V gives a historical overview of the field. We conclude this paper by pointing out important open challenges in Section VI and a conclusion in Section VII.

II. DL in Games Overview

This section gives a brief overview of neural networks and machine learning in the context of games. First, we describe common neural network architectures followed by an overview of the three main categories of machine learning tasks: supervised learning, unsupervised learning, and RL. Approaches in these categories are typically based on gradient-descent optimization. We also highlight evolutionary approaches, as well as a few examples of hybrid approaches that combine several optimization techniques.

A. Neural Network Models

Artificial neural networks (ANNs) are general-purpose functions that are defined by their network structure and the weight of each graph edge. Because of their generality and ability to approximate any continuous real-valued function (given enough parameters), they have been applied to a variety of tasks, including video game playing. The architectures of these ANNs can roughly be divided into two major categories: feedforward networks and recurrent neural networks (RNNs). Feedforward networks take a single input, for example, a representation of the game state, and outputs probabilities or values for each possible action. Convolutional neural networks (CNNs) consist of trainable filters and are suitable for processing image data such as pixels from a video game screen.

RNNs are typically applied to time-series data, in which the output of the network can depend on the network's activation from previous time steps [82], [165]. The training process is similar to feedforward networks, except that the network's previous hidden state is fed back into the network together with the next input. This allows the network to become context-aware

Manuscript received December 7, 2017; revised April 9, 2018, July 24, 2018, and December 7, 2018; accepted January 7, 2019. Date of publication February 13, 2019; date of current version March 17, 2020. The work of N. Justesen was supported by the Elite Research travel grant from The Danish Ministry for Higher Education and Science. *(Corresponding author: Niels Justesen.)*

N. Justesen and S. Risi are with the IT University of Copenhagen, 2300 Copenhagen, Denmark (e-mail: njustesen@gmail.com; sebastian.risi@gmail.com).

P. Bontrager and J. Togelius are with New York University, New York, NY 11201 USA (e-mail: philipjb@nyu.edu; julian@togelius.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TG.2019.2896986

by memorizing the previous activations, which is useful when a single observation from a game does not represent the complete game state. For video game playing, it is common to use a stack of convolutional layers followed by recurrent layers and fully connected feedforward layers.

The following sections will give a brief overview of different optimization methods, which are commonly used for learning game-playing behaviors with deep neural networks. These methods search for the optimal set of parameters to solve some problems. Optimization can also be used to find hyperparameters, such as network architecture and learning parameters, and is well studied within DL [12], [13].

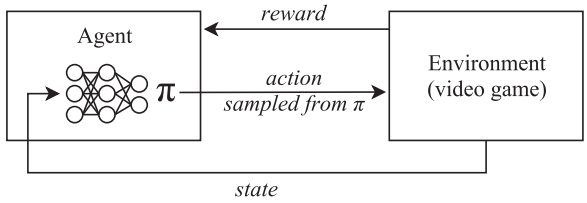

Fig. 1. RL framework where an agent's policy π is determined by a deep neural network. The state of the environment, or an observation such as screen pixels, is fed as input to the agent's policy network. An action is sampled from the policy network's output π, whereafter it receives a reward and the subsequent game state. The goal is to maximize the cumulated rewards. The RL algorithm updates the policy (network parameters) based on the reward.

B. Optimizing Neural Networks

1) Supervised Learning: In supervised learning, a model is trained from examples. During training, the model is asked to make a decision, for which the correct answer is known. The error, i.e., difference between the provided answer and the ground truth, is used as a loss to update the model. The goal is to achieve a model that can generalize beyond the training data and thus perform well on examples it has never seen before. Large datasets usually improve the model's ability to generalize.

In games, these data can come from play traces [16] (i.e., humans playing through the game while being recorded), allowing the agent to learn the mapping from the input state to output actions based on what actions the human performed in a given state. If the game is already solved by another algorithm, it can be used to generate training data, which is useful if the first algorithm is too slow to run in real time. While learning to play from existing data allows agents to quickly learn best practices, it is often brittle; the data available can be expensive to produce and may be missing key scenarios the agent should be able to deal with. For gameplay, the algorithm is limited to the strategies available in the data and cannot explore new ones itself. Therefore, in games, supervised algorithms are often combined with additional training through RL algorithms [133].

Another application of supervised learning in games is to learn the state transitions of a game. Instead of providing the action for a given state, the neural network can learn to predict the next state for an action–state pair. Thus, the network is essentially learning a model of the game, which can then be used to play the game better or to perform planning [45].

2) Unsupervised Learning: Instead of learning a mapping between data and its labels, the objective in unsupervised learning is to discover patterns in the data. These algorithms can learn the distribution of features for a dataset, which can be used to cluster similar data, compress data into its essential features, or create new synthetic data that are characteristic of the original data. For games with sparse rewards (such as *Montezuma's revenge*), learning from data in an unsupervised fashion is a potential solution and an important open DL challenge.

A prominent unsupervised learning technique in DL is the *autoencoder*, which is a neural network that attempts to learn the identity function such that the output is identical to the input [80], [117]. The network consists of two parts: an encoder that maps the input x to a low-dimensional hidden vector h, and a decoder that attempts to reconstruct x from h. The main idea is that by keeping h small, the network has to learn to compress the data and, therefore, learn a good representation. Researchers are beginning to apply such unsupervised algorithms to games to help distill high-dimensional data to more meaningful lower dimensional data, but this research direction is still in its early stages [45]. For a more detailed overview of supervised and unsupervised learning, see [39] and [126].

3) RL Approaches: In RL, an agent learns a behavior by interacting with an environment that provides a reward signal back to the agent. A video game can easily be modeled as an environment in an RL setting, wherein players are modeled as agents with a finite set of actions that can be taken at each step, and the reward signal can be determined by the game score. Fig. 1 depicts the RL framework.

In RL, the agent relies on the reward signal. These signals can occur frequently, such as the change in score within a game, or it can occur infrequently, such as whether an agent has won or lost a game. Video games and RL go well together since most games give rewards for successful strategies. Open-world games do not always have a clear reward model and are thus challenging for RL algorithms.

A key challenge in applying RL to games with sparse rewards is to determine how to assign credit to the many previous actions when a reward signal is obtained. The reward $R(s)$ for state s needs to be propagated back to the actions that lead to the reward. Historically, there are several different ways this problem is approached, which are described in the following. If an environment can be described as a Markov decision process, then the agent can build a probability tree of future states and their rewards. The probability tree can then be used to calculate the utility of the current state. For an RL agent, this means learning the model $P(s'|s,a)$, where P is the probability of state s' given state s and action a. With a model P, utilities can be calculated by

$$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a)U(s')$$

where γ is the discount factor for the utility of future states. This algorithm, known as adaptive dynamic programming, can converge rather quickly as it directly handles the credit assignment problem [142]. The issue is that it has to build a probability tree over the whole problem space and is, therefore, intractable for large problems. As the games covered in this work are considered "large problems," we will not go into further detail on this algorithm.

Another approach to this problem is temporal difference (TD) learning. In TD learning, the agent learns the utilities U directly based off of the observation that the current utility is equal to the current reward plus the utility value of the next state [142]. Instead of learning the state transition model P, it learns to model the utility U for every state. The update equation for U is

$$U(s) = U(s) + \alpha(R(s) + \gamma U(s') - U(s))$$

where α is the learning rate of the algorithm. The above equation does not take into account how s' was chosen. If a reward is found at s_t, it will only affect $U(s_t)$. The next time the agent is at s_{t-1}, then $U(s_{t-1})$ will be aware of the future reward. This will propagate backward over time. Likewise, less common transitions will have less of an impact on utility values. Therefore, U will converge to the same values as are obtained from ADP, albeit slower.

There are alternative implementations of TD that learn rewards for state–action pairs. This allows an agent to choose an action, given the state, with no model of how to transition to future states. For this reason, these approaches are referred to as model-free methods. A popular model-free RL method is Q-learning [162], where the utility of a state is equal to the maximum Q-value for a state. The update equation for Q-learning is

$$Q(s,a) = Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)).$$

In Q-learning, the future reward is accounted for by selecting the best-known future state–action pair. In a similar algorithm called State–Action–Reward–State–Action (SARSA), $Q(s,a)$ is updated only when the next a has been selected and the next s is known [118]. This action pair is used instead of the maximum Q-value. This makes SARSA an *on-policy* method in contrast to Q-learning, which is *off-policy*, because SARSA's Q-value accounts for the agent's own policy.

Q-learning and SARSA can use a neural network as a function approximator for the Q-function. The given Q-update equation can be used to provide the new "expected" Q-value for a state–action pair. The network can then be updated as it is in supervised learning.

An agent's policy $\pi(s)$ determines which action to take given a state s. For Q-learning, a simple policy would be to always take the action with the highest Q-value. Yet, early on in training, Q-values are not very accurate, and an agent could get stuck always exploiting a small reward. A learning agent should prioritize exploration of new actions, as well as the exploitation of what it has learned. This problem is known as a multiarmed bandit problem and has been well explored. The ϵ-greedy strategy is a simple approach that selects the (estimated) optimal action with ϵ probability and otherwise selects a random action.

One approach to RL is to perform gradient descent in the policy's parameter space. Let $\pi_\theta(s,a)$ be the probability that action a is taken at state s given parameters θ. The basic policy gradient algorithm from the REINFORCE family of algorithms [164] updates θ using the gradient $\nabla_\theta \sum_a \pi_\theta(s,a)R(s)$, where $R(s)$ is the discounted cumulative reward obtained from s and forward. In practice, a sample of possible actions from the policy is taken, and it is updated to increase the likelihood that the more successful actions are returned in the future. This lends itself well to neural networks, as π can be a neural network and θ the network weights.

Actor-critic methods combine the policy gradient approach with TD learning, where an actor learns a policy $\pi_\theta(s,a)$ using the policy gradient algorithm, and the critic learns to approximate R using TD learning [143]. Together, they are an effective approach to iteratively learning a policy. In actor-critic methods, there can either be a single network to predict both π and R or two separate networks. For an overview of RL applied to deep neural networks, we suggest the article by Arulkumaran *et al.* [2].

4) Evolutionary Approaches: The optimization techniques discussed so far rely on gradient descent, based on differentiation of a defined error. However, derivative-free optimization methods such as evolutionary algorithms have also been widely used to train neural networks, including, but not limited to, RL tasks. This approach, often referred to as neuroevolution (NE), can optimize a network's weights, as well as their topology/architecture. Because of their generality, NE approaches have been applied extensively to different types of video games. For a complete overview of this field, we refer the interested reader to our NE survey paper [115].

Compared to gradient-descent-based training methods, NE approaches have the benefit of not requiring the network to be differentiable and can be applied to supervised learning, unsupervised learning, and RL problems. The ability to evolve the topology, as well as the weights, potentially offers a way of automating the development of neural network architecture, which currently requires considerable domain knowledge. The promise of these techniques is that evolution could find a neural network topology that is better at playing a certain game than existing human-designed architectures. While NE has been traditionally applied to problems with lower input dimensionality than typical DL approaches, recently, Salimans *et al.* [121] showed that evolution strategies (ESs), which rely on parameter exploration through stochastic noise instead of calculating gradients, can achieve results competitive to current deep RL approaches for Atari video game playing, given enough computational resources.

5) Hybrid Learning Approaches: More recently, researchers have started to investigate hybrid approaches for video game playing, which combine DL methods with other machine learning approaches. Alvernaz and Togelius [1] and Poulsen *et al.* [113] experimented with combining a deep network trained through gradient descent feeding a condensed feature representation into a network trained through artificial evolution. These hybrids aim to combine the best of both approaches as DL methods are able to learn directly from high-dimensional input, while evolutionary methods do not rely on differentiable architectures and work well in games with sparse rewards. Some results suggest that gradient-free methods seem to be better in the early stages of training to avoid premature convergence, while gradient-based methods may be better in the end when less exploration is needed [139].

Another hybrid method for board game playing was AlphaGo [133] that relied on deep neural networks and tree search methods to defeat the world champion in Go, and [36] that applies planning on top of a predictive model.

In general, the hybridization of *ontogenetic* RL (such as Q-learning) with *phylogenetic* methods (such as evolutionary algorithms) has the potential to be very impactful, as it could enable concurrent learning on different timescales [153].

III. Game Genres and Research Platforms

The fast progression of DL methods is undoubtedly due to the convention of comparing results on publicly available datasets. A similar convention in game AI is to use game environments to compare game playing algorithms, in which methods are ranked based on their ability to score points or win in games. Conferences like the IEEE Conference on Computational Intelligence and Games run popular competitions in a variety of game environments.

This section describes popular game genres and research platforms, used in the literature, that are relevant to DL; some examples are shown in Fig. 2. For each genre, we briefly outline what characterizes that genre and describe the challenges faced by algorithms playing games of the genre. The video games that are discussed in this paper have to a large extent supplanted an earlier generation of simpler control problems that long served as the main RL benchmarks but are generally too simple for modern RL methods. In such classic control problems, the input is a simple feature vector, describing the position, velocity, and angles. Popular platforms for such problems are rllab [29], which includes classic problems such as pole balancing and the mountain car problem, and MuJoCo (multijoint dynamics with contact), a physics engine for complex control tasks such as the humanoid walking task [152].

A. Arcade Games

Classic arcade games, of the type found in the late seventies' and early eighties' arcade cabinets, home video game consoles, and home computers, have been commonly used as AI benchmarks within the last decade. Representative platforms for this game type are the Atari 2600, Nintendo NES, Commodore 64, and ZX Spectrum. Most classic arcade games are characterized by movement in a 2-D space (sometimes represented isometrically to provide the illusion of 3-D movement), heavy use of graphical logics (where game rules are triggered by the intersection of sprites or images), continuous-time progression, and either continuous-space or discrete-space movement. The challenges of playing such games vary by game. Most games require fast reactions and precise timing, and a few games, in particular, early sports games such as *Track & Field* (Konami, 1983) rely almost exclusively on speed and reactions. Many games require prioritization of several co-occurring events, which requires some ability to predict the behavior or trajectory of other entities in the game. This challenge is explicit in, e.g., *Tapper* (Bally Midway, 1983) but also in different ways part of platform games such as *Super Mario Bros.* (Nintendo, 1985) and shooters such as *Missile Command* (Atari Inc., 1980). Another common requirement is navigating mazes or other complex environments, as exemplified clearly by games such as *Pac-Man* (Namco, 1980) and *Boulder Dash* (First Star Software, 1984). Some games, such as *Montezuma's Revenge* (Parker Brothers, 1984), require long-term planning involving the memorization of temporarily unobservable game states. Some games feature incomplete information and stochasticity; others are completely deterministic and fully observable.

The most notable game platform used for DL methods is the Arcade Learning Environment (ALE) [10]. ALE is built on top of the Atari 2600 emulator Stella and contains more than 50 original Atari 2600 games. The framework extracts the game score, 160×210 screen pixels, and the RAM content that can be used as input for game playing agents. ALE was the main environment explored in the first deep RL papers that used raw pixels as input. By enabling agents to learn from visual input, ALE thus differs from classic control problems in the RL literature, such as the Cart Pole and Mountain Car problems. An overview and discussion of the ALE environment can be found in [91].

Another platform for classic arcade games is the Retro Learning Environment (RLE) that currently contains seven games released for the Super Nintendo Entertainment System (SNES) [15]. Many of these games have 3-D graphics, and the controller allows for over 720 action combinations. SNES games are thus more complex and realistic than Atari 2600 games, but RLE has not been as popular as ALE.

The General Video Game AI (GVG-AI) framework [116] allows for easy creation and modification of games and levels using the video game description language [122]. This is ideal for testing the generality of agents on multiple games or levels. GVG-AI includes over 100 classic arcade games each with five different levels.

B. Racing Games

Racing games are games where the player is tasked with controlling some kind of vehicle or character so as to reach a goal in the shortest possible time or as to traverse as far as possible along a track in a given time. Usually, the game employs a first-person perspective or a vantage point from just behind the player-controlled vehicle. The vast majority of racing games take a continuous input signal as steering input, similar to a steering wheel. Some games, such as those in the *Forza Motorsport* (Microsoft Studios, 2005–2016) or *Real Racing* (Firemint and EA Games, 2009–2013) series, allow for complex input including gear stick, clutch, and handbrake, whereas more arcade-focused games such as those in the *Need for Speed* (Electronic Arts, 1994–2015) series typically have a simpler set of inputs and thus lower branching factor.

A challenge that is common in all racing games is that the agent needs to control the position of the vehicle and adjust the acceleration or braking, using fine-tuned continuous input, so as to traverse the track as fast as possible. Doing this optimally requires at least short-term planning, one or two turns forward.

Fig. 2. Screenshots of selected games and frameworks used as research platforms for research in DL.

If there are resources to be managed in the game, such as fuel, damage, or speed boosts, this requires longer term planning. When other vehicles are present on the track, there is an adversarial planning aspect added, in trying to manage or block overtaking; this planning is often done in the presence of hidden information (position and resources of other vehicles on different parts of the track).

A popular environment for visual RL with realistic 3-D graphics is the open racing car simulator TORCS [168].

C. First-Person Shooters (FPSs)

More advanced game environments have recently emerged for visual RL agents in an FPS. In contrast to classic arcade games such as those in the ALE benchmark, FPSs have 3-D graphics with partially observable states and are thus a more realistic environment to study. Usually, the viewpoint is that of the player-controlled character, though some games that are broadly in the FPS categories adopt an over-the-shoulder viewpoint. The design of FPS games is such that part of the challenge is simply fast perception and reaction, in particular, spotting enemies and quickly aiming at them. But there are other cognitive challenges as well, including orientation and movement in a complex 3-D environment, predicting actions and locations of multiple adversaries, and in some game modes also team-based collaboration. If visual inputs are used, there is the challenge of extracting relevant information from pixels.

Among FPS platforms are *ViZDoom*, a framework that allows agents to play the classic FPS Doom (id Software, 1993–2017) using the screen buffer as input [73]. *DeepMind Lab* is a platform for 3-D navigation and puzzle-solving tasks based on the *Quake III Arena* (id Software, 1999) engine [6].

D. Open-World Games

Open-world games such as *Minecraft* (Mojang, 2011) or the *Grand Theft Auto* (Rockstar Games, 1997–2013) series are characterized by very nonlinear gameplay, with a large game world to explore, either no set goals or many goals with unclear internal ordering, and large freedom of action at any given time. Key challenges for agents are exploring the world and setting goals, which are realistic and meaningful. As this is a very complex challenge, most research uses these open environments to explore RL methods that can reuse and transfer learned knowledge to new tasks. Project Malmo is a platform built on top of the open-world game *Minecraft*, which can be used to define many diverse and complex problems [65].

E. Real-Time Strategy (RTS) Games

Strategy games are games where the player controls multiple characters or units, and the objective of the game is to prevail in some sort of conquest or conflict. Usually, but not always, the narrative and graphics reflect a military conflict, where units may be, e.g., knights, tanks, or battleships. The key challenge in strategy games is to lay out and execute complex plans involving multiple units. This challenge is in general significantly harder than the planning challenge in classic board games such as Chess mainly because multiple units must be moved at any time and the effective branching factor is typically enormous. The planning horizon can be extremely long, where actions that are taken at the beginning of a game impact the overall strategy. In addition, there is the challenge of predicting the moves of one or several adversaries, who have multiple units themselves. RTS games are strategy games, which do not progress in discrete turns, but where actions can be taken at any point in time. RTS games add the challenge of time prioritization to the already substantial challenges of playing strategy games.

The *StarCraft* (Blizzard Entertainment, 1998–2017) series is without a doubt the most studied game in the RTS genre. The Brood War API (BWAPI)[1] enables software to communicate with *StarCraft* while the game runs, e.g., to extract state features and perform actions. BWAPI has been used extensively in game AI research, but, currently, only a few examples exist where DL has been applied. TorchCraft is a library built on top of BWAPI that connects the scientific computing framework Torch to *StarCraft* to enable machine learning research for this game [145]. Additionally, DeepMind and Blizzard (the developers of *StarCraft*) have developed a machine learning API to support research in *StarCraft II* with features such as simplified visuals designed for convolutional networks [157]. This API contains several mini-challenges, while it also supports the full 1v1 game setting. *μRTS* [104] and *ELF* [151] are two minimalistic RTS game engines that implement some of the features that are present in RTS games.

F. Team Sports Games

Popular sports games are typically based on team-based sports such as soccer, basketball, and football. These games aim to be as realistic as possible with life-like animations and 3-D graphics. Several soccer-like environments have been used extensively as research platforms, both with physical robots and 2-D/3-D simulations, in the annual Robot World Cup Soccer Games

[1] http://bwapi.github.io/

(RoboCup) [3]. *Keepaway Soccer* is a simplistic soccer-like environment, where one team of agents try to maintain control of the ball while another team tries to gain control of it [138]. A similar environment for multiagent learning is *RoboCup 2-D Half-Field-Offense (HFO)*, where teams of two to three players either take the role as offense or defense on one half of a soccer field [50].

G. Text Adventure Games

A classic text adventure game is a form of interactive fiction, where players are given descriptions and instructions in text, rather than graphics, and interact with the storyline through text-based commands [144]. These commands are usually used to query the system about the state, interact with characters in the story, collect and use items, or navigate the space in the fictional world.

These games typically implement one of three text-based interfaces: parser-based, choice-based, and hyperlink-based [54]. Choice-based and hyperlink-based interfaces provide the possible actions to the player at a given state as a list, out of context, or as links in the state description. Parser-based interfaces are, on the other hand, open to any input, and the player has to learn what words the game understands. This is interesting for computers as it is much more akin to natural language, where you have to know what actions should exist based on your understanding of language and the given state.

Unlike some other game genres, like arcade games, text adventure games have not had a standard benchmark of games that everyone can compare against. This makes a lot of results hard to directly compare. A lot of research has focused on games that run on Infocom's Z-Machine game engine, an engine that can play a lot of the early classic games. Recently, Microsoft has introduced the environment TextWorld to help create a standardized text adventure environment [25].

H. OpenAI Gym and Universe

OpenAI Gym is a large platform for comparing RL algorithms with a single interface to a suite of different environments including ALE, GVG-AI, MuJoCo, Malmo, ViZDoom, and more [17]. OpenAI Universe is an extension to OpenAI Gym that currently interfaces with more than a thousand Flash games and aims to add many modern video games in the future.[2]

IV. DL Methods for Game Playing

This section gives an overview of DL techniques used to play video games, divided by game genre. Table II lists DL methods for each game genre and highlights which input features, network architecture, and training methods they rely upon. A typical neural network architecture used in deep RL is shown in Fig. 3.

A. Arcade Games

The ALE consists of more than 50 Atari games and has been the main testbed for deep RL algorithms that learn control policies directly from raw pixels. This section reviews the main advancements that have been demonstrated in ALE. An overview of these advancements is shown in Table I.

TABLE I
Human-Normalized Scores Reported With Various Deep RL Algorithms in ALE on 57 Atari Games Using the *30 no-ops* Evaluation Metric

Results	Mean	Median	Year and orig. paper
DQN [161]	228%	79%	2013 [97]
Double DQN (DDQN) [161]	307%	118%	2015 [155]
Dueling DDQN [161]	373%	151%	2015 [161]
Prior. DDQN [161]	435%	124%	2015 [122]
Prior. Duel DDQN [161]	592%	172%	2015 [122]
A3C [63]	853%	N/A	2016 [96]
UNREAL [63]*	880%	250%	2016 [63]
NoisyNet-DQN [56]	N/A	118%	2017 [35]
Distr. DQN (C51) [9]	701%	178%	2017 [9]
Rainbow [56]	N/A	223%	2017 [56]
IMPALA [30]	958%	192%	2018 [30]
Ape-X DQN [61]	N/A	434%	2018 [61]

References in the first column refer to the paper that included the results, while the last column references the paper that first introduced the specific technique. Note that the reported scores use various amounts of training time and resources, thus not entirely comparable. Successors typically use more resources and less wall-clock time. *Hyperparameters was tuned for every game leading to higher scores for UNREAL.

Deep Q-network (DQN) was the first learning algorithm that showed human expert-level control in ALE [97]. DQN was tested in seven Atari 2600 games and outperformed previous approaches, such as SARSA with feature construction [7] and NE [49], as well as a human expert on three of the games. DQN is based on Q-learning, where a neural network model learns to approximate $Q^{\pi}(s, a)$ that estimates the expected return of taking action a in state s while following a behavior policy μ. A simple network architecture consisting of two convolutional layers followed by a single fully-connected layer was used as a function approximator.

A key mechanism in DQN is *experience replay* [89], where experiences in the form $\{s_t, a_t, r_{t+1}, s_{t+1}\}$ are stored in a replay memory and randomly sampled in batches when the network is updated. This enables the algorithm to reuse and learn from past and uncorrelated experiences, which reduces the variance of the updates. DQN was later extended with a separate target Q-network, the parameters of which are held fixed between individual updates, and was shown to achieve above human expert scores in 29 out of 49 tested games [98].

Deep recurrent Q-learning (DRQN) extends the DQN architecture with a recurrent layer before the output and works well for games with partially observable states [51].

A distributed version of DQN was shown to outperform a nondistributed version in 41 of the 49 games using the Gorila architecture (general RL architecture) [100]. Gorila parallelizes actors that collect experiences into a distributed replay memory, as well as parallelizing learners that train on samples from the same replay memory.

One problem with the Q-learning algorithm is that it often overestimates action values because it uses the same value function for action selection and action evaluation. Double DQN, based on double Q-learning [46], reduces the observed overestimation by learning two value networks with parameters θ and θ' that both use the other network for value estimation, such that the target $Y_t = R_{t+1} + \gamma Q(S_{t+1}, \max_a Q(S_{t+1}, a; \theta_t); \theta'_t)$ [155].

[2]https://universe.openai.com/

TABLE II
OVERVIEW OF DL METHODS APPLIED TO GAMES

Game	Method	Input			Architecture				Training					Miscellaneous				
		Features	Pixels	Text	CNN	RNN	Ext. Memory	MLP	Supervised	Q-learning	Actor-critic	ES	GA	Auxiliary Learning	Hierarchical	Intrinsic Motivation	Transfer Learning	Distributed
Atari 2600 (ALE)	DQN [97], [98]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	○
	DRQN [51]	○	●	○	●	●	○	○	○	●	○	○	○	○	○	○	○	○
	UCTtoClassification [42]	○	●	○	●	○	○	○	●	○	○	○	○	○	○	○	○	○
	Gorila [100]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	●
	Double DQN [155]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	○
	Prioritized DQN [122]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	○
	Dueling DQN [161]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	○
	Bootstrapped DQN [106]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	○
	A3C / A2C [96]	○	●	○	●	●	○	○	○	○	●	○	○	○	○	○	○	●
	ACER [160]	○	●	○	●	○	○	○	○	○	●	○	○	○	○	○	○	●
	Progressive Networks [119]	○	●	○	●	○	○	○	○	○	●	○	○	○	○	○	●	●
	UNREAL [63]	○	●	○	●	●	○	○	○	○	●	○	○	●	○	○	○	●
	Scalable Evolution Strategies [120]	○	●	○	●	○	○	○	○	○	○	●	○	○	○	○	○	●
	Distributional DQN (C51) [9]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	○
	NoisyNet-DQN [35]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	○
	NoisyNet-A3C [35]	○	●	○	●	○	○	○	○	○	●	○	○	○	○	○	○	●
	Rainbow [56]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	○
	ACKTR [166]	○	●	○	●	○	○	○	○	○	●	○	○	○	○	○	○	●
	Deep GA [138]	○	●	○	●	○	○	○	○	○	○	○	●	○	○	○	○	●
	NS-ES / NSR-ES [24]	○	●	○	●	○	○	○	○	○	○	●	○	○	○	○	○	●
	IMPALA [30]	○	●	○	●	○	○	○	○	○	●	○	○	○	○	○	○	●
	TRPO [127]	○	●	○	●	○	○	○	○	○	●	○	○	○	○	○	○	●
	PPO [128]	○	●	○	●	●	○	○	○	○	●	○	○	○	○	○	○	●
	DQfD [57]	○	●	○	●	○	○	○	●	●	○	○	○	○	○	○	○	○
	Ape-X DQN [61]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	●
	Ape-X DQfD [112]	○	●	○	●	○	○	○	●	●	○	○	○	○	○	○	○	●
Ms. Pac-Man	Hybrid Reward Architecture (HRA) [156]	●	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	○
Montezuma's Revenge	Hierarchical-DQN (h-DQN) [77]	●	●	○	●	○	○	○	○	●	○	○	○	○	●	●	○	○
	DQN-CTS / DQN-PixelCNN [8]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	●	○	○
Racing	Direct Perception [21]	●	●	○	●	○	○	○	●	○	○	○	○	○	○	○	●	○
	Deep DPG (DDPG) [88]	○	●	○	●	○	○	○	○	○	●	○	○	○	○	○	○	○
	A3C [96]	○	●	○	●	○	○	○	○	○	●	○	○	○	○	○	○	●
Doom	DQN [73]	○	●	○	●	○	○	○	○	●	○	○	○	○	○	○	○	○
	A3C [167]	○	●	○	●	○	○	○	○	○	●	○	○	○	○	○	○	●
	DRQN [79]	○	●	○	●	●	○	○	○	●	○	○	○	●	○	○	○	○
	DQN + SLAM [14]	○	●	○	●	○	○	○	●	●	○	○	○	○	○	○	○	○
	Direct Future Prediction (DFP) [28]	●	●	○	●	○	○	○	○	○	○	○	○	○	○	○	●	●
Minecraft	H-DRLN [149]	●	●	○	●	○	○	○	○	●	○	○	○	○	●	○	○	○
	RMQN / FRMQN [102]	○	●	○	●	●	●	○	○	●	○	○	○	○	○	○	●	○
	TSCL [92]	○	●	○	●	●	○	○	○	○	○	○	○	○	○	○	○	○
StarCraft	Zero Order [154]	●	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○
	IQL [33]	●	○	○	○	●	○	○	○	●	○	○	○	○	○	○	○	○
	BiCNet [111]	●	○	○	○	●	○	○	○	○	●	○	○	○	○	○	○	○
	COMA [32]	●	○	○	○	●	○	○	○	○	●	○	○	○	○	○	○	○
	Macro-action SL [68]	●	○	○	○	○	○	●	●	○	○	○	○	○	●	○	○	○
	Macro-action CNNFQ/PPO [147], [140]	●	●	○	●	○	○	●	○	○	●	○	○	○	●	○	○	●
RoboCup Soccer (HFO)	DDPG + Inverting Gradients [52]	●	○	○	○	○	○	●	○	○	●	○	○	○	○	○	○	○
	DDPG + Mixing policy targets [53]	●	○	○	○	○	○	●	○	○	●	○	○	○	○	○	○	○
2D billiard	Object-centric prediction [36]	●	●	○	●	●	○	○	●	○	○	○	○	○	○	○	○	○
Text adventure games	LSTM-DQN [101]	○	○	●	○	●	○	○	○	●	○	○	○	○	○	○	○	○
	DRRN [54]	○	○	●	○	○	○	●	○	●	○	○	○	●	○	○	○	○
	Affordance Based Action Selection [37]	○	○	●	○	○	○	●	○	●	○	○	○	●	○	○	○	○
	Golovin [75]	○	○	●	○	●	○	○	○	○	○	○	○	○	○	○	○	○
	AE-DQN [173]	○	○	●	●	○	○	○	○	●	○	○	○	●	○	○	○	○

We refer to *features* as low-dimensional items and values that describe the state of the game such as health, ammunition, score, objects, etc. *MLP* refers to a traditional fully connected architecture without convolutional or recurrent layers.

Another improvement is *prioritized experience replay* from which important experiences are sampled more frequently based on the TD error, which was shown to significantly improve both DQN and double DQN [123].

Dueling DQN uses a network that is split into two streams after the convolutional layers to separately estimate state value $V^\pi(s)$ and the action advantage $A^\pi(s,a)$, such that $Q^\pi(s,a) = V^\pi(s) + A^\pi(s,a)$ [161]. Dueling DQN improves double DQN and can also be combined with prioritized experience replay.

Double DQN and dueling DQN were also tested in the five more complex games in the RLE and achieved a mean score

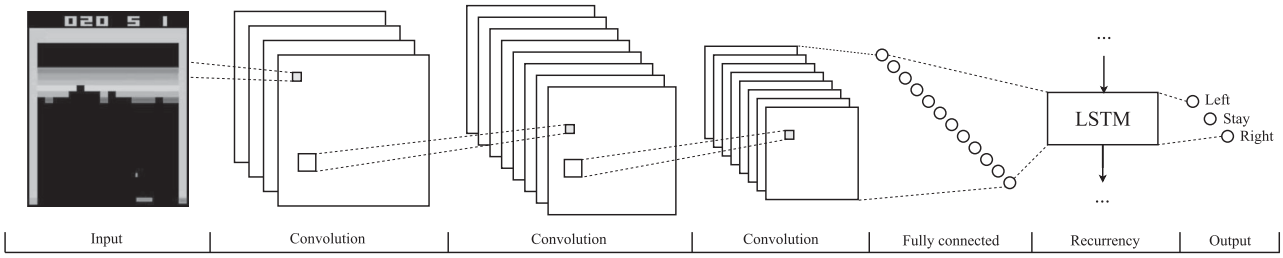

Fig. 3. Example of a typical network architecture used in deep RL for game playing with pixel input. The input usually consists of a preprocessed screen image, or several stacked or concatenated images, which is followed by a couple of convolutional layers (often without pooling), and a few fully connected layers. Recurrent networks have a recurrent layer, usually an LSTM, after the fully connected layers. The output typically consists of one unit for each unique combination of actions in the game, and actor-critic methods also have one for the state value $V(s)$. Examples of this architecture, without a recurrent layer and with some variations, are [9], [24], [30], [35], [56], [96]–[98], [100], [106], [120], [121], [123], [139], [155], [160], [161], and [166], and examples with a recurrent layer are [51], [63], and [96].

of around 50% of a human expert [15]. The best result in these experiments was by dueling DQN in the game *Mortal Kombat* (Midway, 1992) with 128%.

Bootstrapped DQN improves exploration by training multiple Q-networks. A randomly sampled network is used during each training episode, and *bootstrap masks* modulate the gradients to train the networks differently [106].

Robust policies can be learned with DQN for competitive or cooperative multiplayer games by training one network for each player and play them against each other in the training process [146]. Agents trained in the multiplayer mode perform very well against novel opponents, whereas agents trained against a stationary algorithm fail to generalize their strategies to novel adversaries.

Multithreaded asynchronous variants of DQN, SARSA, and actor-critic methods can utilize multiple CPU threads on a single machine, reducing training roughly linear to the number of parallel threads [96]. These variants do not rely on a replay memory because the network is updated on uncorrelated experiences from parallel actors, which also helps stabilize *on-policy* methods. The asynchronous advantage actor-critic (A3C) algorithm is an actor-critic method that uses several parallel agents to collect experiences that all asynchronously update a global actor-critic network. A3C outperformed prioritized dueling DQN, which was trained for 8 days on a GPU, with just half the training time on a CPU [96].

An actor-critic method with experience replay (ACER) implements an efficient trust region policy method that forces updates to not deviate far from a running average of past policies [160]. The performance of the ACER in ALE matches dueling DQN with prioritized experience replay and A3C without experience replay, while it is much more data efficient.

A3C with progressive neural networks [120] can effectively transfer learning from one game to another. The training is done by instantiating a network for every new task with connections to all the previous learned networks. This gives the new network access to knowledge already learned.

The advantage actor-critic (A2C), a synchronous variant of A3C [96], updates the parameters synchronously in batches and has comparable performance while only maintaining one neural network [166]. Actor-critic using Kronecker-factored trust region (ACKTR) extends A2C by approximating the natural policy gradient updates for both the actor and the critic [166]. In Atari, ACKTR has slower updates compared to A2C (at most 25% per time step) but is more sample efficient (e.g., by a factor of 10 in Atlantis) [166]. Trust region policy optimization (TRPO) uses a *surrogate* objective with theoretical guarantees for monotonic policy improvement, while it practically implements an approximation called *trust region* [128]. This is done by constraining network updates with a bound on the Kullback–Leibler (KL) divergence between the current and the updated policy. TRPO has robust and data-efficient performance in Atari games, while it has high memory requirements and several restrictions. Proximal policy optimization (PPO) is an improvement on TRPO that uses a similar *surrogate* objective [129], but instead uses a soft constraint (originally suggested in [128]) by adding the KL divergence as a penalty. Instead of having a fixed penalty coefficient, it uses a clipped surrogate objective that penalizes policy updates outside some specified interval. PPO was shown to be more sample efficient than A2C and on par with ACER in Atari, while PPO does not rely on replay memory. PPO was also shown to have comparable or better performance than TRPO in continuous control tasks while being simpler and easier to parallelize.

Importance weighted actor–learner architecture (IMPALA) is an actor-critic method, where multiple learners with GPU access share gradients between each other while being synchronously updated from a set of actors [30]. This method can scale to a large number of machines and outperforms A3C. Additionally, IMPALA was trained, with one set of parameters, to play all 57 Atari games in ALE with a mean human-normalized score of 176.9% (median of 59.7%) [30]. Experiences collected by the actors in the IMPALA setup can lack behind the learners' policy and thus result in *off-policy* learning. This discrepancy is mitigated through a *V-trace* algorithm that weighs the importance of experiences based on the difference between the actor's and learner's policies [30].

The UNsupervised REinforcement and Auxiliary Learning (UNREAL) algorithm is based on A3C but uses a replay memory from which it learns auxiliary tasks and *pseudoreward functions* concurrently [63]. UNREAL only shows a small improvement over vanilla A3C in ALE, but larger improvements in other domains (see Section IV-D).

Distributional DQN takes a distributional perspective on RL by treating $Q(s, a)$ as an approximate distribution of returns instead of a single approximate expectation for each action [9]. The distribution is divided into a so-called set of atoms, which determines the granularity of the distribution. Their results show that the more fine-grained the distributions are, the better are the results, and with 51 atoms (this variant was called C51), it achieved mean scores in ALE almost comparable to UNREAL.

In *NoisyNets*, noise is added to the network parameters, and a unique noise level for each parameter is learned using gradient descent [35]. In contrast to ϵ-greedy exploration, where an agent samples actions either from the policy or from a uniform random distribution, NoisyNets use a noisy version of the policy to ensure exploration, and this was shown to improve DQN (NoisyNet-DQN) and A3C (NoisyNet-A3C).

Rainbow combines several DQN enhancements: double DQN, prioritized replay, dueling DQN, distributional DQN, and NoisyNets, and achieved a mean score higher than any of the enhancements individually [56].

ESs are black-box optimization algorithms that rely on parameter exploration through stochastic noise instead of calculating gradients and were found to be highly parallelizable with a linear speedup in training time when more CPUs are used [121]. Seven hundred and twenty CPUs were used for 1 h, whereafter ESs managed to outperform A3C (which ran for four days) in 23 out of 51 games, while ES used three to ten times as much data due to its high parallelization. ESs only ran a single day, and thus, their full potential is currently unknown. Novelty search is a popular algorithm that can overcome environments with deceptive and/or sparse rewards by guiding the search toward novel behaviors [84]. The ES has been extended to use novelty search (NS-ES), which outperforms the ES on several challenging Atari games by defining novel behaviors based on the RAM states [24]. A quality-diversity variant called NSR-ES that uses both novelty and the reward signal reach an even higher performance [24]. NS-ES and NSR-ES reached worse results on a few games, possibly where the reward function is not sparse or deceptive.

A simple genetic algorithm with a Gaussian noise mutation operator evolves the parameters of a deep neural network (deep GA) and can achieve surprisingly good scores across several Atari games [139]. Deep GA shows comparable results to DQN, A3C, and ES on 13 Atari games using up to thousands of CPUs in parallel. Additionally, random search, given roughly the same amount of computation, was shown to outperform DQN on four out of 13 games and A3C on five games [139]. While there has been concern that evolutionary methods do not scale as well as gradient-descent-based methods, one possibility is separating the feature construction from the policy network; evolutionary algorithms can then create extremely small networks that still play well [26].

A few supervised learning approaches have been applied to arcade games. In [42], a slow planning agent was applied offline, using Monte Carlo tree search, to generate data for training a CNN via multinomial classification. This approach, called *UCTtoClassification*, was shown to outperform DQN. Policy distillation [119] or actor-mimic [108] methods can be used to train one network to mimic a set of policies (e.g., for different games). These methods can reduce the size of the network and sometimes also improve the performance. A frame prediction model can be learned from a dataset generated by a DQN agent using the encoding–transformation–decoding network architecture; the model can then be used to improve exploration in a retraining phase [103]. Self-supervised tasks, such as reward prediction, validation of state–successor pairs, and mapping states and successor states to actions, can define auxiliary losses used in pretraining of a policy network, which ultimately can improve learning [132].

The *training objective* provides feedback to the agent, while the *performance objective* specifies the target behavior. Often, a single reward function takes both roles, but, for some games, the performance objective does not guide the training sufficiently. The hybrid reward architecture (HRA) splits the reward function into n different reward functions, where each of them is assigned a separate learning agent [156]. The HRA does this by having n output streams in the network, and thus n Q-values, which are combined when actions are selected. The HRA was able to achieve the maximum possible score in less than 3000 episodes.

B. Montezuma's Revenge

Environments with sparse feedback remain an open challenge for RL. The game *Montezuma's Revenge* is a good example of such an environment in ALE and has thus been studied in more detail and used for benchmarking learning methods based on intrinsic motivation and curiosity. The main idea of applying intrinsic motivation is to improve the exploration of the environment based on some self-rewarding system, which eventually will help the agent to obtain an extrinsic reward. DQN fails to obtain any reward in this game (receiving a score of 0) and Gorila achieves an average score of just 4.2. A human expert can achieve 4367 points, and it is clear that the methods presented so far are unable to deal with environments with such sparse rewards. A few promising methods aim to overcome these challenges.

Hierarchical-DQN (h-DQN) [77] operates on two temporal scales, where one Q-value function $Q_1(s, a; g)$, the *controller*, learns a policy over actions that satisfy goals chosen by a higher level Q-value function $Q_2(s, g)$, the *metacontroller*, which learns a policy over intrinsic goals (i.e., which goals to select). This method was able to reach an average score of around 400 in *Montezuma's Revenge*, where goals were defined as states in which the agent *reaches* (collides with) a certain type of object. This method, therefore, must rely on some object detection mechanism.

Pseudocounts have been used to provide intrinsic motivation in the form of exploration bonuses when unexpected pixel configurations are observed and can be derived from context tree switching (CTS) density models [8] or neural density models [107]. Density models assign probabilities to images, and a model's pseudocount of an observed image is the model's change in prediction compared to being trained one additional time on the same image. Impressive results were achieved in *Montezuma's Revenge* and other

hard Atari games by combining DQN with the CTS density model (DQN-CTS) or the PixelCNN density model (DQN-PixelCNN) [8]. Interestingly, the results were less impressive when the CTS density model was combined with A3C (A3C-CTS) [8].

Ape-X DQN is a distributed DQN architecture similar to Gorila, as actors are separated from the learner. Ape-X DQN was able to reach state-of-the-art results across the 57 Atari games using 376 cores and one GPU, running at 50K FPS [61]. Deep Q-learning from demonstrations (DQfD) draws samples from an experience replay buffer that is initialized with demonstration data from a human expert and is superior to previous methods on 11 Atari games with sparse rewards [57]. Ape-X DQfD combines the distributed architecture from Ape-X, and the learning algorithm from DQfD using expert data and was shown to outperform all previous methods in ALE, as well as beating level 1 in *Montezuma's Revenge* [112].

To improve the performance, Kaplan *et al.* augmented the agent training with text instructions. An instruction-based RL approach that uses both a CNN for visual input and the RNN for text-based instruction inputs managed to achieve a score of 3500 points. Instructions were linked to positions in rooms, and agents were rewarded when they reached those locations [71], demonstrating a fruitful collaboration between a human and a learning algorithm. Experiments in *Montezuma's Revenge* also showed that the network learned to generalize to unseen instructions that were similar to previous instructions.

Similar work demonstrates how an agent can execute text-based commands in a 2-D maze-like environment called XWORLD, such as walking to and picking up objects, after having learned a *teacher's* language [172]. An RNN-based language module is connected to a CNN-based perception module. These two modules were then connected to an action-selection module and a recognition module that learns the teacher's language in a question answering process.

C. Racing Games

There are generally two paradigms for vision-based autonomous driving highlighted in [21]: 1) end-to-end systems that learn to map images to actions directly (behavior reflex); and 2) systems that parse the sensor data to make informed decisions (mediated perception). An approach that falls in between these paradigms is *direct perception*, where a CNN learns to map from images to meaningful affordance indicators, such as the car angle and distance to lane markings, from which a simple controller can make decisions [21]. Direct perception was trained on recordings of 12 h of human driving in TORCS, and the trained system was able to drive in very diverse environments. Amazingly, the network was also able to generalize to real images.

End-to-end RL algorithms such as DQN cannot be directly applied to continuous environments such as racing games because the action space must be discrete and with relatively low dimensionality. Instead, policy gradient methods, such as actor-critic [27] and deterministic policy gradient (DPG) [134] can learn policies in high-dimensional and continuous action spaces. Deep DPG (DDPG) is a policy gradient method that implements both experience replay and a separate target network and was used to train a CNN end-to-end in TORCS from images [88].

The aforementioned A3C methods have also been applied to the racing game TORCS using only pixels as input [96]. In those experiments, rewards were shaped as the agent's velocity on the track, and after 12 h of training, A3C reached a score between roughly 75% and 90% of a human tester in tracks with and without opponent bots, respectively.

While most approaches to training deep networks from high-dimensional input in video games are based on gradient descent, a notable exception is an approach by Koutník *et al.* [76], where Fourier-type coefficients were evolved that encoded a recurrent network with over one million weights. Here, evolution was able to find a high-performing controller for TORCS that only relied on high-dimensional visual input.

D. First-Person Shooters

Kempka *et al.* [73] demonstrated that a CNN with max-pooling and fully connected layers trained with DQN can achieve human-like behaviors in basic scenarios. In the Visual Doom AI Competition 2016,[3] a number of participants submitted pretrained neural-network-based agents that competed in a multiplayer deathmatch setting. Both a *limited* competition, in which bots competed in known levels, and a *full* competition that included bots competing in unseen levels were held. The winner of the limited track used a CNN trained with A3C using reward shaping and curriculum learning [167]. Reward shaping tackled the problem of sparse and delayed rewards, giving artificial positive rewards for picking up items and negative rewards for using ammunition and losing health. Curriculum learning attempts to speed up learning by training on a set of progressively harder environments [11]. The second-place entry in the limited track used a modified DRQN network architecture with an additional stream of fully connected layers to learn supervised auxiliary tasks such as enemy detection, with the purpose of speeding up the training of the convolutional layers [79]. Position inference and object mapping from pixels and depth buffers using simultaneous localization and mapping (SLAM) also improve DQN in Doom [14].

The winner of the full deathmatch competition implemented a *direct future prediction* (DFP) approach that was shown to outperform DQN and A3C [28]. The architecture used in DFP has three streams: one for the screen pixels, one for lower dimensional measurements describing the agent's current state, and one for describing the agent's goal, which is a linear combination of prioritized measurements. DFP collects experiences in a memory and is trained with supervised learning techniques to predict the future measurements based on the current state, goal, and selected action. During training, actions are selected that yield the best-predicted outcome, based on the current goal. This method can be trained on various goals and generalizes to unseen goals at test time.

[3]http://vizdoom.cs.put.edu.pl/competition-cig-2016

Navigation in 3-D environments is one of the important skills required for FPS games and has been studied extensively. A CNN long short-term memory (LSTM) network was trained with A3C extended with additional outputs predicting the pixel depths and loop closure, showing significant improvements [95].

The UNREAL algorithm, based on A3C, implements an auxiliary task that trains the network to predict the immediate subsequent future reward from a sequence of consecutive observations. UNREAL was tested on fruit gathering and exploration tasks in OpenArena and achieved a mean human-normalized score of 87%, where A3C only achieved 53% [63].

The ability to transfer knowledge to new environments can reduce the learning time and can in some cases be crucial for some challenging tasks. Transfer learning can be achieved by pretraining a network in similar environments with simpler tasks or by using random textures during training [20]. The distill and transfer learning (Distral) method trains several worker policies (one for each task) concurrently and shares a distilled policy [149]. The worker policies are regularized to stay close to the shared policy, which will be the centroid of the worker policies. Distral was applied to DeepMind Lab.

The intrinsic curiosity module, consisting of several neural networks, computes an intrinsic reward each time step based on the agent's inability to predict the outcome of taking actions. It was shown to learn to navigate in complex Doom and Super Mario levels only relying on intrinsic rewards [110].

E. Open-World Games

The hierarchical deep reinforcement learning network (H-DRLN) architecture implements a lifelong learning framework, which is shown to be able to transfer knowledge between simple tasks in *Minecraft* such as navigation, item collection, and placement tasks [150]. The H-DRLN uses a variation of policy distillation [119] to retain and encapsulate learned knowledge into a single network.

Neural turing machines (NTMs) are fully differentiable neural networks coupled with an external memory resource, which can learn to solve simple algorithmic problems such as copying and sorting [40]. Two memory-based variations, inspired by NTM, called recurrent memory Q-network (RMQN) and feedback recurrent memory Q-network (FRMQN), were able to solve complex navigation tasks that require memory and active perception [102].

The teacher–student curriculum learning (TSCL) framework incorporates a teacher that prioritizes tasks, wherein the student's performance is either increasing (learning) or decreasing (forgetting) [92]. TSCL enabled a policy gradient learning method to solve mazes that were otherwise not possible with a uniform sampling of subtasks.

F. RTS Games

The previous sections described methods that learn to play games *end-to-end*, i.e., a neural network is trained to map states directly to actions. RTS games, however, offer much more complex environments, in which players have to control multiple agents simultaneously in real time on a partially observable map. Additionally, RTS games have no in-game scoring, and thus, the reward is determined by who wins the game. For these reasons, learning to play RTS games end-to-end may be infeasible for the foreseeable future, and instead, subproblems have been studied so far.

For the simplistic RTS platform, μRTS, a CNN was trained as a state evaluator using supervised learning on a generated dataset and used in combination with Monte Carlo tree search [4], [136]. This approach performed significantly better than previous evaluation methods.

StarCraft has been a popular game platform for AI research, but so far only with a few DL approaches. DL methods for *StarCraft* have focused on micromanagement (unit control) or build-order planning and has ignored other aspects of the game. The problem of delayed rewards in *StarCraft* can be circumvented in combat scenarios; here, rewards can be shaped as the difference between damage inflicted and damage incurred [32], [33], [111], [154]. States and actions are often described locally relative to units, which is extracted from the game engine. If agents are trained individually, it is difficult to know which agents contributed to the global reward [19], a problem known as the multiagent credit assignment problem. One approach is to train a generic network, which controls each unit separately and search in policy space using zero-order optimization based on the reward accrued in each episode [154]. This strategy was able to learn successful policies for armies of up to 15 units.

Independent Q-learning (IQL) simplifies the multiagent RL problem by controlling units individually while treating other agents as if they were part of the environment [147]. This enables Q-learning to scale well to a large number of agents. However, when combining IQL with recent techniques such as experience replay, agents tend to optimize their policies based on experiences with obsolete policies. This problem is overcome by applying *fingerprints* to experiences and by applying an importance-weighted loss function that naturally decays obsolete data, which has shown improvements for some small combat scenarios [33].

The multiagent bidirectionally-coordinated network (BiCNet) implements a vectorized actor-critic framework based on a bidirectional RNN, with one dimension for every agent, and outputs a sequence of actions [111]. This network architecture is unique to the other approaches, as it can handle an arbitrary number of units of different types.

Counterfactual multiagent (COMA) policy gradients is an actor-critic method with a centralized critic and decentralized actors that address the multiagent credit assignment problem with a counterfactual baseline computed by the critic network [32]. COMA achieves state-of-the-art results, for decentralized methods, in small combat scenarios with up to ten units on each side.

DL has also been applied to build-order planning in *StarCraft* using macro-based supervised learning approach to imitate human strategies [68]. The trained network was integrated as a module used in an existing bot capable of playing the full game with an otherwise hand-crafted behavior. Another macro-based approach, here using RL instead of SL, called convolutional neural-network-fitted Q-learning (CNNFQ), was trained with

double DQN for build-order planning in *StarCraft II* and was able to win against medium-level scripted bots on small maps [148]. A macro action-based RL method that uses PPO for build-order planning and high-level attack planning was able to outperform the built-in bot in *StarCraft II* at level 10 [141]. This is particularly impressive as the level-10 bot cheats by having full vision of the map and faster resource harvesting. The results were obtained using 1920 parallel actors on 3840 CPUs across 80 machines and only for one matchup on one map. This system won a few games against platinum-level human players but lost all games against diamond-level players. The authors report that the learned policy "lacks strategy diversity in order to consistently beat human players" [141].

G. Team Sports Games

DDPG was applied to RoboCup 2-D HFO [51]. The actor network used two output streams: one for the selection of discrete action types (dash, turn, tackle, and kick) and one for each action type's 1–2 continuously valued parameters (power and direction). The *inverting gradients* bounding approach downscales the gradients as the output approaches its boundaries and inverts the gradients if the parameter exceeds them. This approach outperformed both SARSA and the best agent in the 2012 RoboCup. DDPG was also applied to HFO by mixing *on-policy* updates with one-step Q-learning updates [53] and outperformed a hand-coded agent with expert knowledge with one player on each team.

H. Physics Games

As video games are usually a reflection or simplification of the real world, it can be fruitful to learn an intuition about the physical laws in an environment. A predictive neural network using an object-centered approach (also called fixations) learned to run simulations of a billiards game after being trained on random interactions [36]. This predictive model could then be used for planning actions in the game.

A similar predictive approach was tested in a 3-D game-like environment, using the Unreal Engine, where ResNet-34 [55] (a deep residual network used for image classification) was extended and trained to predict the visual outcome of blocks that were stacked such that they would usually fall [86]. Residual networks implement *shortcut connections* that skip layers, which can improve learning in very deep networks.

I. Text Adventure Games

Text adventure games, in which both states and actions are presented as text only, are a special video game genre. A network architecture called LSTM-DQN [101] was designed specifically to play these games and is implemented using LSTM networks that convert text from the world state into a vector representation, which estimates Q-values for all possible state–action pairs. LSTM-DQN was able to complete between 96% and 100% of the quests on average in two different text adventure games.

To be able to improve on these results, researchers have moved toward learning language models and word embeddings to augment the neural network. An approach that combines RL with explicit language understanding is deep reinforcement relevance net (DRRN) [54]. This approach has two networks that learn word embeddings. One embeds the state description, and the other embeds the action description. Relevance between the two embedding vectors is calculated with an interaction function such as the inner product of the vectors or a bilinear operation. The relevance is then used as the Q-value, and the whole process is trained end-to-end with deep Q-learning. This approach allows the network to generalize to phrases not seen during training, which is an improvement for very large text games. The approach was tested on the text games Saving John and Machine of Death, both choice-based games.

Taking language modeling further, Fulda *et al.* explicitly modeled language affordances to assist in action selection [37]. A word embedding is first learned from a Wikipedia Corpus via unsupervised learning [94], and this embedding is then used to calculate analogies such as *song is to sing* as *bike is to* x, where x can then be calculated in the embedding space [94]. The authors build a dictionary of verbs, noun pairs, and another one of object manipulation pairs. Using the learned affordances, the model can suggest a small set of actions for a state description. Policies were learned with Q-learning and tested on 50 Z-machine games.

The Golovin Agent focuses exclusively on language models [75] that are pretrained from a corpus of books in the fantasy genre. Using word embeddings, the agent can replace synonyms with known words. Golovin is built of five command generators: general, movement, battle, gather, and inventory. These are generated by analyzing the state description, using the language models to calculate and sample from a number of features for each command. Golovin uses no RL and scores comparable to the affordance method.

Most recently, Zahavy *et al.* have proposed another DQN method [173]. This method uses a type of attention mechanism called action elimination network (AEN). In parser-based games, the action space is very large. The AEN learns, while playing, to predict which actions that will have no effect for a given state description. The AEN is then used to eliminate most of the available actions for a given state and after which the remaining actions are evaluated with the Q-network. The whole process is trained end-to-end and achieves similar performance to DQN with a manually constrained actions space. Despite the progress made for text adventure games, current techniques are still far from matching human performance.

Outside of text adventure games, natural language processing has been used for other text-based games as well. To facilitate communication, a deep distributed recurrent Q-network (DDRQN) architecture was used to train several agents to learn a communication protocol to solve the multiagent *Hats* and *Switch* riddles [34]. One of the novel modifications in DDRQN is that agents use shared network weights that are conditioned on their unique ID, which enables faster learning while retaining diversity between agents.

V. Historical Overview of DL in Games

The previous section discussed DL methods in games according to the game type. This section instead looks at the

development of these methods in terms of how they influenced each other, giving a historical overview of the DL methods that are reviewed in the previous section. Many of these methods are inspired from or directly build upon previous methods, while some are applied to different game genres and others are tailored to specific types of games.

Fig. 4 shows an influence diagram with the reviewed methods and their relations to earlier methods (the current section can be read as a long caption to that figure). Each method in the diagram is colored to show the game benchmark. DQN [97] was very influential as an algorithm that uses gradient-based DL for pixel-based video game playing and was originally applied to the Atari benchmark. Note that earlier approaches exist but with less success such as [109], and successful gradient-free methods [115]. Double DQN [155] and dueling DQN [161] are early extensions that use multiple networks to improve estimations. DRQN [51] uses an RNN as the Q-network. Prioritized DQN [123] is another early extension, and it adds improved experience replay sampling. Bootstrapped DQN [106] builds off of double DQN with a different improved sampling strategy. Further DQN enhancements used for Atari include: the C51 algorithm [9], which is based on DQN but changes the Q function; Noisy-Nets which make the networks stochastic to aid with exploration [35]; DQfD which also learns from examples [57]; and Rainbow, which combines many of these state-of-the-art techniques together [56].

Gorila was the first asynchronous method based on DQN [100] and was followed by A3C [96], which uses multiple asynchronous agents for an actor-critic approach. This was further extended at the end of 2016 with UNREAL [63], which incorporates work done with auxiliary learning to handle sparse feedback environments. Since then, there has been a lot of additional extensions on A3C [35], [120], [160], [166]. IMPALA has taken it further with focusing on a single trained agent that can play all of the Atari games [30]. In 2018, the move toward large-scale distributed learning has continued and advanced with Ape-X [61], [112].

Evolutionary techniques are also seeing a Renaissance for video games. First, Salimans *et al.* showed that ESs could compete with deep RL [121]. Then, two more papers came out of Uber AI: one showing that derivative-free evolutionary algorithms can compete with deep RL [139], and an extension to ES [24]. These benefit from easy parallelization and possibly have some advantage in exploration.

Another approach used on Atari around the time DQN was introduced is TRPO [77]. This updates a surrogate objective that is updated from the environment. Later, in 2017, PPO was introduced as a more robust and simpler surrogate optimization scheme that also draws from innovations in A3C [129]. Some extensions are specifically for *Montezuma's revenge*, which is a game within the ALE benchmark, but it is particularly difficult due to sparse rewards and hidden information. The algorithms that do best on *Montezuma* do so by extending DQN with intrinsic motivation [8] and hierarchical learning [77]. Ms. Pac-Man was also singled out from Atari, where the reward function was learned in separate parts to make the agent more robust to new environments [156].

Doom is another benchmark that is new as of 2016. Most of the work for this game has been extending methods designed for Atari to handle richer data. A3C + curriculum learning [167] proposes using curriculum learning with A3C. DRQN + auxiliary learning [79] extends DRQN by adding additional rewards during training. DQN + SLAM [14] combines techniques for mapping unknown environments with DQN.

DFP [28] is the only approach that is not extending an Atari technique. Like UCT To Classification [42] for Atari, Object-centric Prediction [36] for Billiard, and Direct Perception [21] for Racing, DFP uses supervised learning to learn about the game. All of these, except UCT To Classification, learn to directly predict some future state of the game and make a prediction from this information. None of these works, all from different years, refer to each other. Besides Direct Perception, the only unique work for racing is DDPG [88], which extends DQN for continuous controls. This technique has been extended for RoboCup Soccer [52] [53].

Work on *StarCraft* micromanagement (unit control) is based on Q-learning started in late 2016. IQL [33] extends DQN-Prioritized DQN by treating all other agents as part of the environment. COMA [32] extends IQL by calculating counterfactual rewards, the marginal contribution each agent added. BiCNet [111] and zero-order optimization [154] are RL based but are not derived from DQN. Another popular approach is hierarchical learning. In 2017, it was tried with replay data [68], and in 2018, state-of-the-art results were achieved by using it with two different RL methods [141], [148].

Some work published in 2016 extends DQN to play *Minecraft* [150]. At around the same time, techniques were developed to make DQN context-aware and modular to handle the large state space [102]. Recently, curriculum learning has been applied to *Minecraft* as well [92].

DQN was applied to text adventure games in 2015 [101]. Soon after, it was modified to have a language-specific architecture and use the state–action pair relevance as the Q-value [54]. Most of the work on these games has been focused on explicit language modeling. Golovin agent and affordance-based action selection both use neural networks to learn language models, which provide the actions for the agents to play [37], [75]. Recently, in 2018, DQN was used again paired with an AEN [173].

Combining extensions from previous algorithms have proven to be a promising direction for DL applied to video games, with Atari being the most popular benchmark for RL. Another clear trend, which is apparent in Table II, is the focus on parallelization: distributing the work among multiple CPUs and GPUs. Parallelization is most common with actor-critic methods, such as A2C and A3C, and evolutionary approaches, such as Deep GA [139] and ESs [24], [121]. Hierarchical RL, intrinsic motivation, and transfer learning are promising new directions to explore to master currently unsolved problems in video game playing.

VI. Open Challenges

While DL has shown remarkable results in video game playing, a multitude of important open challenges remain, which we review here. Indeed, looking back at the current state of research

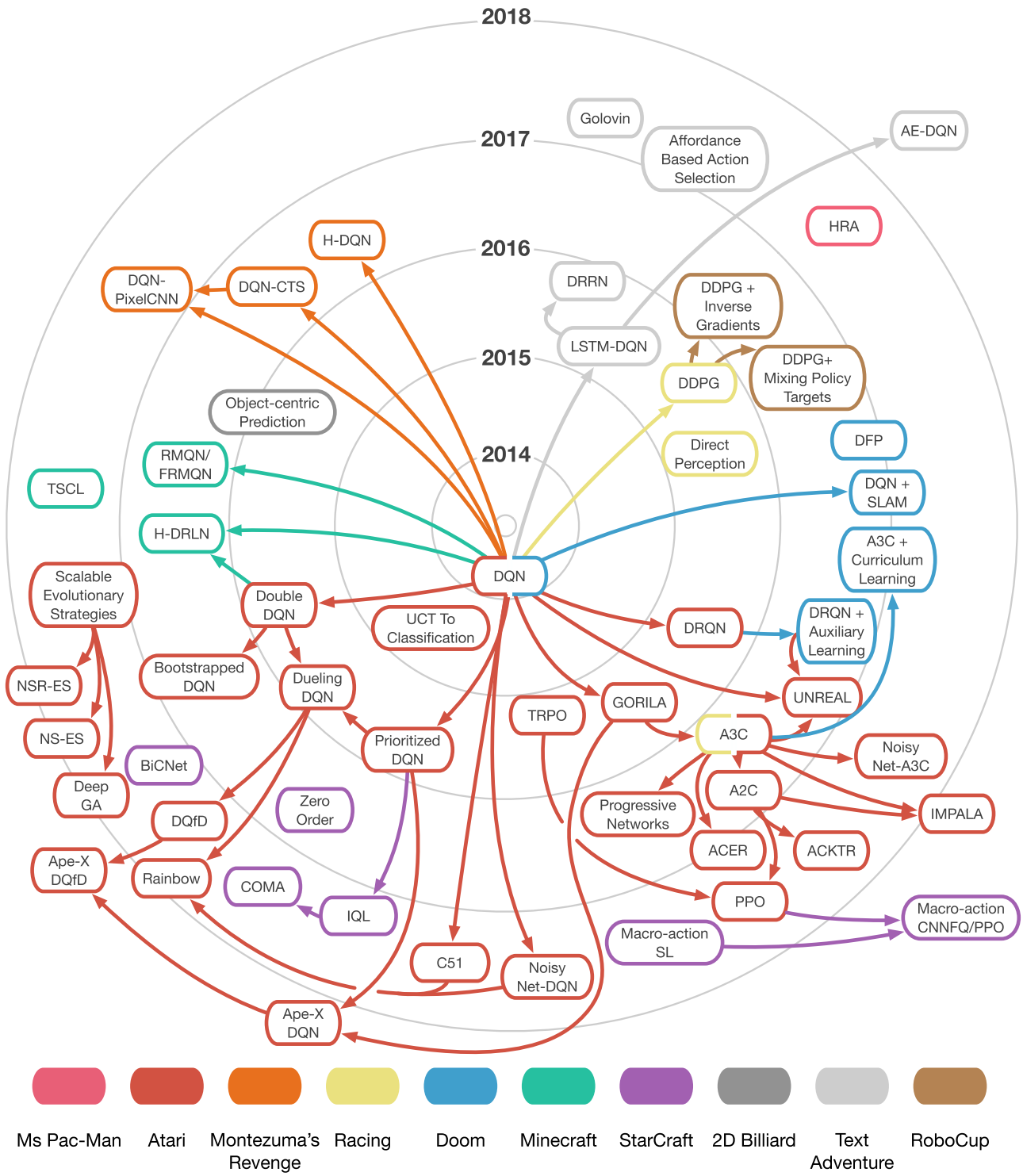

Fig. 4. Influence diagram of the DL techniques discussed in this paper. Each node is an algorithm while the color represents the game benchmark. The distance from the center represents the date that the original paper was published on arXiv. The arrows represent how techniques are related. Each node points to all other nodes that used or modified that technique. Arrows pointing to a particular algorithm show which algorithms influenced its design. Influences are not transitive: if algorithm a influenced b and b influenced c, a did not necessarily influence c.

from a decade or two in the future, it is likely that we will see the current research as early steps in a broad and important research field. This section is divided into four broad categories (agent model properties, game industry, learning models of games, and computational resources) with different game-playing challenges that remain open for DL techniques. We mention a few potential approaches for some of the challenges, while the best way forward for others is currently not clear.

A. Agent Model Properties

1) General Video Game Playing: Being able to solve a single problem does not make you intelligent; nobody would say that Deep Blue or AlphaGo [133] possess general intelligence, as they cannot even play Checkers (without retraining), much less make coffee or tie their shoelaces. To learn generally intelligent behavior, you need to train on not just a single task, but many different tasks [83]. Video games have been suggested as ideal environments for learning general intelligence, partly because there are so many video games that share common interface and reward conventions [124]. Yet, the vast majority of work on DL in video games focuses on learning to play a single game or even performing a single task in a single game.

While deep RL-based approaches can learn to play a variety of different Atari games, it is still a significant challenge to develop algorithms that can learn to play any kind of game (e.g., Atari games, *DOOM*, and *StarCraft*). Current approaches still require significant effort to design the network architecture and reward function to a specific type of game.

Progress on the problem of playing multiple games includes progressive neural networks [120], which allow new games to be learned (without forgetting previously learned ones) and solved quicker by exploiting previously learned features through lateral connections. However, they require a separate network for each task. Elastic weight consolidation [74] can learn multiple Atari games sequentially and avoids catastrophic forgetting by protecting weights from being modified that are important for previously learned games. In PathNet, an evolutionary algorithm is used to select which parts of a neural network are used for learning new tasks, demonstrating some transfer learning performance on ALE games [31].

In the future, it will be important to extend these methods to learn to play multiple games, even if those games are very different—most current approaches focus on different (known) games in the ALE framework. One suitable avenue for this kind of research is the new learning track of the GVGAI competition [78], [116]. GVGAI has a potentially unlimited set of games, unlike ALE. Recent work in GVGAI showed that model-free deep RL overfitted not just to the individual game, but even to the individual level; this was countered by continuously generating new levels during training [69].

It is possible that significant advances on the multigame problem will come from outside DL. In particular, the recent tangled graph representation, a form of genetic programming, has shown promise in this task [72]. The recent IMPALA algorithm tries to tackle multigame learning through massive scaling, with somewhat promising results [30].

2) Overcoming Sparse, Delayed, or Deceptive Rewards: Games such as *Montezuma's Revenge* that are characterized by sparse rewards still pose a challenge for most of the deep RL approaches. While recent advances that combine DQN with intrinsic motivation [8] or expert demonstrations [57], [112] can help, games with sparse rewards are still a challenge for current deep RL methods. There is a long history of research in intrinsically motivated RL [22], [125], as well as hierarchical RL, which might be useful here [5], [163]. The Project Malmo environment, based on *Minecraft*, provides an excellent venue for creating tasks with very sparse rewards, where agents need to set their own goals. Derivative-free and gradient-free methods, such as ESs and genetic algorithms, explore the parameter space by sampling locally and are promising for these games, especially when combined with novelty search as in [24] and [139].

3) Learning With Multiple Agents: Current deep RL approaches are mostly concerned with training a single agent. A few exceptions exist, where multiple agents have to cooperate [32], [33], [85], [111], [154], but it remains an open challenge how these can scale to more agents in various situations. In many current video games such as *StarCraft* or *GTA V*, many agents interact with each other and the player. To scale multiagent learning in video games to the same level of performance as current single agent approaches will likely require new methods that can effectively train multiple agents at the same time.

4) Lifetime Adaptation: While nonplayer characters (NPCs) can be trained to play a variety of games well (see Section IV), current machine learning techniques still struggle when it comes to agents that should be able to adapt during their lifetime, i.e., while the game is being played. For example, a human player can quickly change its behavior when realizing that the player is always ambushed at the same position in an FPS map. However, most of the current DL techniques would require expensive retraining to adapt to these situations and other unforeseen situations that they have not encountered during training. The amount of data provided by the real-time behavior of a single human are nowhere near that required by common DL methods. This challenge is related to the wider problem of few-shot learning, transfer learning, and general video game playing. Solving it will be important to create more believable and human-like NPCs.

5) Human-Like Game Playing: Lifetime learning is just one of the differences that current NPCs lack in comparison to human players. Most approaches are concerned with creating agents that play a particular game as well as possible, often only taking into account the score reached. However, if humans are expected to play against or cooperate with AI-based bots in video games, other factors come into play. Instead of creating a bot that plays perfectly, in this context, it becomes more important that the bot is believable and is fun to play against, with similar idiosyncrasies we expect from a human player.

Human-like game playing is an active area of research with two different competitions focused on human-like behavior, namely, the *2k BotPrize* [58], [59] and the Turing Test track of the *Mario AI Championship* [131]. Most entries in these competitions are based on various nonneural network techniques, while some used evolutionary training of deep neural networks to generate human-like behavior [105], [127].

6) Adjustable Performance Levels: Almost all current research on DL for game playing aims at creating agents that can play the game as well as possible, maybe even "beating" it. However, for purposes of game testing, creating tutorials, and demonstrating games—in all those places where it would be important to have human-like game play—it could be important to be able to create agents with a particular skill level. If

your agent plays better than any human player, then it is not a good model of what a human would do in the game. At its most basic, this could entail training an agent that plays the game very well and then find a way of decreasing the performance of that agent. However, it would be more useful to be able to adjust the performance level in a more fine-grained way, so as to, for example, separately control the reaction speed or long-term planning ability of an agent. Even more useful would be to be able to ban certain capacities of playstyles of a trained agent, so as to test whether, for example, a given level could be solved without certain actions or tactics.

One path to realizing this is the concept of *procedural personas*, where the preferences of an agent are encoded as a set of utility weights [60]. However, this concept has not been implemented using DL, and it is still unclear how to realize the planning depth control in this context.

7) Dealing With Extremely Large Decision Spaces: Whereas the average branching factor hovers around 30 for Chess and 300 for Go, a game like *StarCraft* has a branching factor that is orders of magnitudes larger. While recent advances in evolutionary planning have allowed real-time and long-term planning in games with larger branching factors to [66], [67], and [159], how we can scale deep RL to such levels of complexity is an important open challenge. Learning heuristics with DL in these games to enhance search algorithms is also a promising direction.

B. Game Industry

1) Adoption in the Game Industry: Many of the recent advances in DL have been accelerated because of the increased interest by a variety of different companies such as Facebook, Google/Alphabet, Microsoft, and Amazon, which heavily invest in its development. However, the game industry has not embraced these advances to the same extent. This sometimes surprises commentators outside of the game industry, as games are seen as making heavy use of AI techniques. However, the type of AI that is most commonly used in the games industry focuses more on hand-authoring of expressive NPC behaviors rather than machine learning. An often-cited reason for the lack of adoption of neural networks (and similar methods) within this industry is that such methods are inherently difficult to control, which could result in unwanted NPC behaviors (e.g., an NPC could decide to kill a key actor that is relevant to the story). Additionally, training deep network models require a certain level of expertise, and the pool of experts in this area is still limited. It is important to address these challenges to encourage a wide adoption in the game industry.

Additionally, while most DL approaches focus exclusively on playing games as well as possible, this goal might not be the most important for the game industry [171]. Here, the level of fun or engagement the player experiences while playing is a crucial component. One use of DL for game playing in the game production process is for game testing, where artificial agents test that levels are solvable or that the difficulty is appropriate. DL might see its most prominent use in the games industry not for playing games, but for generating game content [130] based on training on existing content [140], [158], or for modeling player experience [169].

Within the game industry, several of the large development and technology companies, including Electronic Arts, Ubisoft, and Unity, have recently started in-house research arms focusing partly on DL. It remains to be seen whether these techniques will also be embraced by the development arms of these companies or their customers.

2) Interactive Tools for Game Development: Related to the previous challenge, there is currently a lack of tools for designers to easily train NPC behaviors. While many open-source tools to training deep networks exist now, most of them require a significant level of expertise. A tool that allows designers to easily specify desired NPC behaviors (and undesired ones) while assuring a certain level of control over the final trained outcomes would greatly accelerate the uptake of these new methods in the game industry.

Learning from human preferences is one promising direction in this area. This approach has been extensively studied in the context of NE [115], and also in the context of video games, allowing nonexpert users to breed behaviors for Super Mario [135]. Recently, a similar preference-based approach was applied to deep RL method [23], allowing agents to learn Atari games based on a combination of human preference learning and deep RL. Recently, the game company King published results using imitation learning to learn policies for playtesting of Candy Crush levels, showing a promising direction for new design tools [41].

3) Creating New Types of Video Games: DL could potentially offer a way to create completely new games. Most of today's game designs stem from a time when no advanced AI methods were available or the hardware too limited to utilize them, meaning that games have been designed to not need AI. Designing new games *around* AI can help to break out of these limitations. While evolutionary algorithms and NE in particular [115] have allowed the creation of completely new types of games, DL based on gradient descent has not been explored in this context. NE is a core mechanic in games such as NERO [137], Galactic Arms Race [48], Petalz [114], and Evo-Commander [64]. One challenge with gradient-based optimization is that the structures are often limited to having mathematical smoothness (i.e., differentiability), making it challenging to create interesting and unexpected outputs.

C. Learning Models of Games

Much work on DL for game-playing takes a model-free end-to-end learning approach, where a neural network is trained to produce actions given state observations as input. However, it is well known that a good and fast forward model makes game playing much easier, as it makes it possible to use planning methods based on tree search or evolution [171]. Therefore, an important open challenge in this field is to develop methods that can learn a forward model of the game, making it possible to reason about its dynamics.

The hope is that approaches that learn the rules of the game can generalize better to different game variations and show more

robust learning. Promising work in this area includes the approach by Guzdial *et al.* [44] that learns a simple game engine of Super Mario Bros. from gameplay data. Kansky *et al.* [70] introduce the idea of Schema Networks that follow an object-oriented approach and are trained to predict future object attributes and rewards based on the current attributes and actions. A trained schema network thus provides a probabilistic model that can be used for planning and is able to perform zero-shot transfer to variations of Breakout similar to those used in training.

D. Computational Resources

With more advanced computational models and a larger number of agents in open worlds, computational speed becomes a concern. Methods that aim to make the networks computationally more efficient by either compressing networks [62] or pruning networks after training [43], [47] could be useful. Of course, improvements in processing power in general or for neural networks specifically will also be important. Currently, it is not feasible to train networks in real time to adapt to changes in the game or to fit players' playing styles, something which could be useful in the design process.

VII. Conclusion

This paper reviewed DL methods applied to game playing in video games of various genres including arcade, racing, FPSs, open-world, RTS, team sports, physics, and text adventure games. Most of the reviewed work is within end-to-end model-free deep RL, where a CNN learns to play directly from raw pixels by interacting with the game. Recent work demonstrates that derivative-free ESs and genetic algorithms are competitive alternatives. Some of the reviewed work apply supervised learning to imitate behaviors from game logs, while others are based on methods that learn a model of the environment. For simple games, such as most arcade games, the reviewed methods can achieve above human-level performance, while there are many open challenges in more complex games.

Acknowledgment

The authors would like to thank the numerous colleagues who took the time to comment on drafts of this paper, including C. Tessler, D. Pérez-Liébana, E. Caballero, H. Daumé, III, J. Busk, K. Arulkumaran, M. Heywood, M. G. Bellemare, M.-P. Huget, M. Preuss, N. de Freitas, N. A. Barriga, O. Delalleau, P. Stone, S. Ontañón, T. Matiisen, Y. Fu, and Y. Hou.

References

[1] S. Alvernaz and J. Togelius, "Autoencoder-augmented neuroevolution for visual doom playing," in *Proc. IEEE Conf. Comput. Intell. Games*, 2017, pp. 1–8.

[2] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, Nov. 2017.

[3] M. Asada, M. M. Veloso, M. Tambe, I. Noda, H. Kitano, and G. K. Kraetzschmar, "Overview of RoboCup-98," *AI Mag.*, vol. 21, no. 1, pp. 9–19, 2000.

[4] N. A. Barriga, M. Stanescu, and M. Buro, "Combining strategic learning and tactical search in real-time strategy games," in *Proc. 13th AAAI Conf. Artif. Intell. Interact. Digit. Entertainment*, 2017, pp. 9–15.

[5] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dyn. Syst.*, vol. 13, no. 4, pp. 341–379, 2003.

[6] C. Beattie *et al.*, "DeepMind lab," 2016, arXiv:1612.03801.

[7] M. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," in *Proc. 24th Int. Joint Conf. Artif. Intell.*, 2015, pp. 4148–4152.

[8] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 1471–1479.

[9] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 449–458.

[10] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *J. Artif. Intell. Res.*, vol. 47, pp. 253–279, 2013.

[11] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 41–48.

[12] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proc. 30th Int. Conf. Int. Conf. Mach. Learn.*, 2013, pp. I-115–I-123.

[13] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2011, pp. 2546–2554.

[14] S. Bhatti, A. Desmaison, O. Miksik, N. Nardelli, N. Siddharth, and P. H. Torr, "Playing doom with SLAM-augmented deep reinforcement learning," 2016, arXiv:1612.00380.

[15] N. Bhonker, S. Rozenberg, and I. Hubara, "Playing SNES in the retro learning environment," 2017, arXiv:1611.02205.

[16] M. Bogdanovic, D. Markovikj, M. Denil, and N. De Freitas, "Deep apprenticeship learning for playing video games," in *Proc. Workshops 29th AAAI Conf. Artif. Intell.*, Jan. 2015.

[17] G. Brockman *et al.*, "OpenAI Gym," 2016, arXiv:1606.01540.

[18] C. B. Browne *et al.*, "A survey of Monte Carlo tree search methods," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012.

[19] Y.-H. Chang, T. Ho, and L. P. Kaelbling, "All learning is local: Multi-agent learning in global reward games," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2003, pp. 807–814.

[20] D. S. Chaplot, G. Lample, K. M. Sathyendra, and R. Salakhutdinov, "Transfer deep reinforcement learning in 3D environments: An empirical study," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016.

[21] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "DeepDriving: Learning affordance for direct perception in autonomous driving," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 2722–2730.

[22] N. Chentanez, A. G. Barto, and S. P. Singh, "Intrinsically motivated reinforcement learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2005, pp. 1281–1288.

[23] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 4302–4310.

[24] E. Conti, V. Madhavan, F. P. Such, J. Lehman, K. Stanley, and J. Clune, "Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 5032–5043.

[25] M.-A. Côté *et al.*, "TextWorld: A learning environment for text-based games," 2018, arXiv:1806.11532.

[26] G. Cuccu, J. Togelius, and P. Cudre-Mauroux, "Playing Atari with six neurons," 2018, arXiv:1806.01363.

[27] T. Degris, P. M. Pilarski, and R. S. Sutton, "Model-free reinforcement learning with continuous action in practice," in *Proc. Amer. Control Conf.*, 2012, pp. 2177–2182.

[28] A. Dosovitskiy and V. Koltun, "Learning to act by predicting the future," in *Proc. Int. Conf. Learn. Represent.*, 2017, [Online]. Available: https://sites.google.com/site/representationlearning2014/program-details/publication-model.

[29] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 1329–1338.

[30] L. Espeholt *et al.*, "IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures," in*Proc. 35th Int. Conf. Mach. Learn.*, Jul. 10–15, 2018, pp. 1407–1416.

[31] C. Fernando *et al.*, "PathNet: Evolution channels gradient descent in super neural networks," 2017, arXiv:1701.08734.

[32] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 2974–2982.

[33] J. N. Foerster *et al.*, "Stabilising experience replay for deep multi-agent reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1146–1155.

[34] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate to solve riddles with deep distributed recurrent q-networks," 2016, arXiv:1602.02672.

[35] M. Fortunato *et al.*, "Noisy networks for exploration," in *Proc. Int. Conf. Learn. Represent.*, 2018. [Online]. Available: https://sites.google.com/site/representationlearning2014/program-details/publication-model.

[36] K. Fragkiadaki, P. Agrawal, S. Levine, and J. Malik, "Learning visual predictive models of physics for playing billiards," in *Proc. Int. Conf. Learn. Represent.*, 2016. [Online]. Available: https://sites.google.com/site/representationlearning2014/program-details/publication-model.

[37] N. Fulda, D. Ricks, B. Murdoch, and D. Wingate, "What can you do with a rock? Affordance extraction via word embeddings," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, 2017, pp. 1039–1045.

[38] L. Galway, D. Charles, and M. Black, "Machine learning in digital games: A survey," *Artif. Intell. Rev.*, vol. 29, no. 2, pp. 123–161, 2008.

[39] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[40] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," 2014, arXiv:1410.5401.

[41] S. Gudmundsson *et al.*, "Human-like playtesting with deep learning," in *Proc. IEEE Conf. Comput. Intell. Games*, 2018, pp. 1–8.

[42] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang, "Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 3338–3346.

[43] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient DNNs," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 1379–1387.

[44] M. Guzdial, B. Li, and M. O. Riedl, "Game engine learning from video," in *Proc. Int. Joint Conf. Artif. Intell.*, 2017, pp. 3707–3713.

[45] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," in *Advances in Neural Information Processing Systems 31*, S. Bengio and H. Wallach and H. Larochelle and K. Grauman and N. Cesa-Bianchi and R. Garnett, Eds. New York, NY, USA: Curran Associates, 2018, pp. 2450–2462.

[46] H. V. Hasselt, "Double q-learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2010, pp. 2613–2621.

[47] B. Hassibi *et al.*, "Second order derivatives for network pruning: Optimal brain surgeon," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 1993, pp. 164–164.

[48] E. J. Hastings, R. K. Guha, and K. O. Stanley, "Automatic content generation in the galactic arms race video game," *IEEE Trans. Comput. Intell. AI Games*, vol. 1, no. 4, pp. 245–263, Dec. 2009.

[49] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, "A neuroevolution approach to general Atari game playing," *IEEE Trans. Comput. Intell. AI Games*, vol. 6, no. 4, pp. 355–366, Dec. 2014.

[50] M. Hausknecht, P. Mupparaju, S. Subramanian, S. Kalyanakrishnan, and P. Stone, "Half field offense: An environment for multiagent learning and ad hoc teamwork," in *Proc. AAMAS Adaptive Learn. Agents Workshop*, 2016.

[51] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable MDPs," in *Proc. AAAI Fall Symp. Sequential Decis. Making Intell. Agents*, Nov. 2015, pp. 29–37.

[52] M. Hausknecht and P. Stone, "Deep reinforcement learning in parameterized action space," in *Proc. Int. Conf. Learn. Represent.*, May 2016. [Online]. Available: http://www.cs.utexas.edu/users/ai-lab/?hausknecht:iclr16.

[53] M. Hausknecht and P. Stone, "On-policy vs. off-policy updates for deep reinforcement learning," in *Proc. IJCAI Workshop: Deep Reinforcement Learn.: Frontiers Challenges*, 2016.

[54] J. He *et al.*, "Deep reinforcement learning with a natural language action space," in *Proc. 54th Annu. Meeting Assoc. Comput. Linguistics*, 2016, vol. 1, pp. 1621–1630.

[55] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[56] M. Hessel *et al.*, "Rainbow: Combining improvements in deep reinforcement learning," in *Proc. AAAI*, 2018.

[57] T. Hester *et al.*, "Deep q-learning from demonstrations," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 3223–3230.

[58] P. Hingston, "A new design for a turing test for Bots," in *Proc. IEEE Symp. Comput. Intell. Games*, 2010, pp. 345–350.

[59] P. Hingston, *Believable Bots: Can Computers Play Like People?* New York, NY, USA: Springer, 2012.

[60] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, "Generative agents for player decision modeling in games," in *Proc. 9th Int. Conf. Found. Digit. Games*, 2014. [Online]. Available: http://www.fdg2014.org/papers/fdg2014_poster_05.pdf

[61] D. Horgan *et al.*, "Distributed prioritized experience replay," in *Proc. Int. Conf. Learn. Represent.*, 2018. [Online]. Available: https://sites.google.com/site/representationlearning2014/program-details/publication-model

[62] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and $<$ 0.5 MB model size," 2016, arXiv:1602.07360.

[63] M. Jaderberg *et al.*, "Reinforcement learning with unsupervised auxiliary tasks," in *Proc. Int. Conf. Learn. Represent.*, 2017. [Online]. Available: https://sites.google.com/site/representationlearning2014/program-details/publication-model.

[64] D. Jallov, S. Risi, and J. Togelius, "EvoCommander: A novel game based on evolving and switching between artificial brains," *IEEE Trans. Comput. Intell. AI Games*, vol. 9, no. 2, pp. 181–191, Jun. 2017.

[65] M. Johnson, K. Hofmann, T. Hutton, and D. Bignell, "The Malmo platform for artificial intelligence experimentation," in *Proc. Int. Joint Conf. Artif. Intell.*, 2016, pp. 4246–4247.

[66] N. Justesen, T. Mahlmann, and J. Togelius, "Online evolution for multi-action adversarial games," in *Proc. Eur. Conf. Appl. Evol. Comput.*, 2016, pp. 590–603.

[67] N. Justesen and S. Risi, "Continual online evolution for in-game build order adaptation in StarCraft," in *Proc. Genetic Evol. Comput. Conf.*, 2017, pp. 187–194.

[68] N. Justesen and S. Risi, "Learning macromanagement in StarCraft from replays using deep learning," in *Proc. IEEE Conf. Comput. Intell. Games*, 2017, pp. 162–169.

[69] N. Justesen, R. R. Torrado, P. Bontrager, A. Khalifa, J. Togelius, and S. Risi, "Illuminating generalization in deep reinforcement learning through procedural level generation." in *Proc. NeurIPS Workshop Deep Reinforcement Learn.*, 2018.

[70] K. Kansky *et al.*, "Schema networks: Zero-shot transfer with a generative causal model of intuitive physics," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1809–1818.

[71] R. Kaplan, C. Sauer, and A. Sosa, "Beating Atari with natural language guided reinforcement learning," 2017, arXiv:1704.05539.

[72] S. Kelly and M. I. Heywood, "Multi-task learning in Atari video games with emergent tangled program graphs," in *Proc. Genetic Evol. Comput. Conf.*, 2017, pp. 195–202.

[73] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, "ViZDoom: A doom-based AI research platform for visual reinforcement learning," in *Proc. IEEE Conf. Comput. Intell. Games*, 2016, pp. 1–8.

[74] J. Kirkpatrick *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proc. Nat. Acad. Sci. USA*, vol. 114, pp. 3521–3526, 2017.

[75] B. Kostka, J. Kwiecieli, J. Kowalski, and P. Rychlikowski, "Text-based adventures of the Golovin AI agent," in *Proc. IEEE Conf. Comput. Intell. Games*, 2017, pp. 181–188.

[76] J. Koutník, G. Cuccu, J. Schmidhuber, and F. Gomez, "Evolving large-scale neural networks for vision-based reinforcement learning," in *Proc. 15th Annu. Conf. Genetic Evol. Comput.*, 2013, pp. 1061–1068.

[77] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 3675–3683.

[78] K. Kunanusont, S. M. Lucas, and D. Perez-Liebana, "General video game AI: Learning from screen capture," in *Proc. IEEE Congr. Evol. Comput.*, 2017, pp. 2078–2085.

[79] G. Lample and D. S. Chaplot, "Playing FPS games with deep reinforcement learning," in *Proc. AAAI Conf. Artif. Intell.*, 2017, pp. 2140–2146.

[80] Y. Le Cun, "Modèles connexionnistes de l'apprentissage," Ph.D. dissertation, Pierre Marie Curie Univ., Paris, France, 1987.

[81] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[82] Y. LeCun *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, 1989.

[83] S. Legg and M. Hutter, "Universal intelligence: A definition of machine intelligence," *Minds Mach.*, vol. 17, no. 4, pp. 391–444, 2007.

[84] J. Lehman and K. O. Stanley, "Exploiting open-endedness to solve problems through the search for novelty," in *Proc. 11th Int. Conf. Artif. Life*, 2008, pp. 329–336.

[85] J. Z. Leibo, V. Zambaldi, M. Lanctot, J. Marecki, and T. Graepel, "Multi-agent reinforcement learning in sequential social dilemmas," in *Proc. 16th Conf. Auton. Agents Multiagent Syst.*, 2017, pp. 464–473.

[86] A. Lerer, S. Gross, and R. Fergus, "Learning physical intuition of block towers by example," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 430–438.

[87] Y. Li, "Deep reinforcement learning: An overview," 2017, arXiv:1701.07274.

[88] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2016.

[89] L.-J. Lin, "Reinforcement learning for robots using neural networks," Ph.D. dissertation, School Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, 1993.

[90] S. M. Lucas and G. Kendall, "Evolutionary computation and games," *IEEE Comput. Intell. Mag.*, vol. 1, no. 1, pp. 10–18, Feb. 2006.

[91] 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. Intell. Res.*, vol. 61, pp. 523–562, 2018.

[92] T. Matiisen, A. Oliver, T. Cohen, and J. Schulman, "Teacher-student curriculum learning," in *Proc. Deep Reinforcement Learn. Symp.*, 2017.

[93] R. Miikkulainen, B. D. Bryant, R. Cornelius, I. V. Karpov, K. O. Stanley, and C. H. Yong, "Computational intelligence in games," in *Computational Intelligence: Principles and Practice*. Piscataway, NJ, USA: IEEE Comput. Intell. Soc., 2006, pp. 155–191.

[94] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013, arXiv:1301.3781.

[95] P. Mirowski *et al.*, "Learning to navigate in complex environments," in *Proc. Int. Conf. Learn. Represent.*, 2016. [Online]. Available: https://sites.google.com/site/representationlearning2014/program-details/publication-model

[96] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[97] V. Mnih *et al.*, "Playing Atari with deep reinforcement learning," in *Proc. Int. Conf. Neural Inf. Process. Syst., Deep Learn. Workshop*, 2013.

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

[99] H. Muñoz-Avila, C. Bauckhage, M. Bida, C. B. Congdon, and G. Kendall, "Learning and game AI," in *Dagstuhl Follow-Ups*, vol. 6. Wadern, Germany: Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[100] A. Nair *et al.*, "Massively parallel methods for deep reinforcement learning," 2015, arXiv:1507.04296.

[101] K. Narasimhan, T. D. Kulkarni, and R. Barzilay, "Language understanding for textbased games using deep reinforcement learning," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2015, pp. 1–11.

[102] J. Oh, V. Chockalingam, S. Singh, and H. Lee, "Control of memory, active perception, and action in Minecraft," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, vol. 48, pp. 2790–2799.

[103] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh, "Action-conditional video prediction using deep networks in Atari games," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 2863–2871.

[104] S. Ontanón, "The combinatorial multi-armed bandit problem and its application to real-time strategy games," in *Proc. 9th AAAI Conf. Artif. Intell. Interact. Digit. Entertainment*, 2013, pp. 58–64.

[105] J. Ortega, N. Shaker, J. Togelius, and G. N. Yannakakis, "Imitating human playing styles in Super Mario Bros," *Entertainment Comput.*, vol. 4, no. 2, pp. 93–104, 2013.

[106] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped DQN," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 4026–4034.

[107] G. Ostrovski, M. G. Bellemare, A. van den Oord, and R. Munos, "Count-based exploration with neural density models," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 2721–2730.

[108] E. Parisotto, J. L. Ba, and R. Salakhutdinov, "Actor-mimic: Deep multitask and transfer reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2016. [Online]. Available: https://sites.google.com/site/representationlearning2014/program-details/publication-model

[109] M. Parker and B. D. Bryant, "Neurovisual control in the Quake II environment," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 44–54, Mar. 2012.

[110] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2778–2787.

[111] P. Peng *et al.*, "Multiagent bidirectionally-coordinated nets for learning to play StarCraft combat games," 2017, arXiv:1703.10069.

[112] T. Pohlen *et al.*, "Observe and look further: Achieving consistent performance on Atari," 2018, arXiv:1805.11593.

[113] A. P. Poulsen, M. Thorhauge, M. Hvilshj, and S. Risi, "DLNE: A hybridization of deep learning and neuroevolution for visual control," in *Proc. IEEE Conf. Comput. Intell. Games*, 2017, pp. 256–263.

[114] S. Risi, J. Lehman, D. B. D'Ambrosio, R. Hall, and K. O. Stanley, "Combining search-based procedural content generation and social gaming in the Petalz video game," in *Proc. 8th AAAI Conf. Artif. Intell. Interact. Digit. Entertainment*, 2012, pp. 63–68.

[115] S. Risi and J. Togelius, "Neuroevolution in games: State of the art and open challenges," *IEEE Trans. Comput. Intell. AI Games*, vol. 9, no. 1, pp. 25–41, Mar. 2017.

[116] R. Rodriguez Torrado, P. Bontrager, J. Togelius, J. Liu, and D. Perez-Liebana, "Deep reinforcement learning for general video game AI," in *Proc. IEEE Conf. Comput. Intell. Games*, 2018, pp. 1–8.

[117] D. E. Rumelhart *et al.*, "A general framework for parallel distributed processing," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1. Cambridge, MA, USA: MIT Press, 1986, pp. 45–76.

[118] G. A. Rummery and M. Niranjan, *On-Line Q-Learning Using Connectionist Systems*, vol. 37. Cambridge, U.K.: Univ. Cambridge, 1994.

[119] A. A. Rusu *et al.*, "Policy distillation," in *Proc. Int. Conf. Learn. Represent.*, 2016. [Online]. Available: https://sites.google.com/site/representationlearning2014/program-details/publication-model

[120] A. A. Rusu *et al.*, "Progressive neural networks," 2016, arXiv: 1606.04671.

[121] T. Salimans, J. Ho, X. Chen, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," 2017, arXiv:1703.03864.

[122] T. Schaul, "A video game description language for model-based or interactive learning," in *Proc. IEEE Conf. Comput. Intell. Games*, 2013, pp. 1–8.

[123] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. Int. Conf. Learn. Represent.*, 2016.

[124] T. Schaul, J. Togelius, and J. Schmidhuber, "Measuring intelligence through games," 2011, arXiv:1109.1314.

[125] J. Schmidhuber, "Formal theory of creativity, fun, and intrinsic motivation (1990–2010)," *IEEE Trans. Auton. Mental Develop.*, vol. 2, no. 3, pp. 230–247, Sep. 2010.

[126] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, 2015.

[127] J. Schrum, I. V. Karpov, and R. Miikkulainen, "UT 2: Human-like behavior via neuroevolution of combat behavior and replay of human traces," in *Proc. IEEE Conf. Comput. Intell. Games*, 2011, pp. 329–336.

[128] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897.

[129] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, arXiv:1707.06347.

[130] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games*. New York, NY, USA: Springer, 2016.

[131] N. Shaker *et al.*, "The turing test track of the 2012 Mario AI Championship: Entries and evaluation," in *Proc. IEEE Conf. Comput. Intell. Games*, 2013, pp. 1–8.

[132] E. Shelhamer, P. Mahmoudieh, M. Argus, and T. Darrell, "Loss is its own reward: Self-supervision for reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2016.

[133] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[134] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. 31st Int. Conf. Mach. Learn.*, 2014, pp. 387–395.

[135] P. D. Sørensen, J. M. Olsen, and S. Risi, "Breeding a diversity of super Mario behaviors through interactive evolution," in *Proc. IEEE Conf. Comput. Intell. Games*, 2016, pp. 1–7.

[136] M. Stanescu, N. A. Barriga, A. Hess, and M. Buro, "Evaluating real-time strategy game states using convolutional neural networks," in *Proc. IEEE Conf. Comput. Intell. Games*, 2016, pp. 1–7.

[137] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the NERO video game," *IEEE Trans. Evol. Comput.*, vol. 9, no. 6, pp. 653–668, Dec. 2005.
[138] P. Stone and R. S. Sutton, "Keepaway soccer: A machine learning test bed," in *Proc. Robot Soccer World Cup*, 2001, pp. 214–223.
[139] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," in *Proc. 11th Annu. Conf. Genetic Evol. Comput.*, 2017, pp. 145–152.
[140] A. Summerville *et al.*, "Procedural content generation via machine learning (PCGML)," *IEEE Trans. Games*, vol. 10, no. 3, pp. 257–270, Sep. 2018.
[141] P. Sun *et al.*, "TStarBots: Defeating the cheating level builtin AI in StarCraft II in the full game," 2018, arXiv:1809.07193.
[142] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, vol. 1. Cambridge, MA, USA: MIT Press, 1998.
[143] R. S. Sutton *et al.*, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 1999, vol. 99, pp. 1057–1063.
[144] P. Sweetser, *Emergence in Games*. Boston, MA, USA: Cengage Learning, 2008.
[145] G. Synnaeve *et al.*, "TorchCraft: A library for machine learning research on real-time strategy games," 2016, arXiv:1611.00625.
[146] A. Tampuu *et al.*, "Multiagent cooperation and competition with deep reinforcement learning," *PloS One*, vol. 12, no. 4, 2017, Art. no. e0172395.
[147] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proc. 10th Int. Conf. Mach. Learn.*, 1993, pp. 330–337.
[148] Z. Tang, D. Zhao, Y. Zhu, and P. Guo, "Reinforcement learning for build-order production in StarCraft II," in *Proc. 8th Int. Conf. Inf. Sci. Technol.*, 2018, pp. 153–158.
[149] Y. Teh *et al.*, "Distral: Robust multitask reinforcement learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 4497–4507.
[150] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor, "A deep hierarchical approach to lifelong learning in minecraft," in *Proc. 31st AAAI Conf. Artif. Intell.*, San Francisco, CA, USA, Feb. 4–9, 2017, pp. 1553–1561.
[151] Y. Tian, Q. Gong, W. Shang, Y. Wu, and C. L. Zitnick, "ELF: An extensive, lightweight and flexible research platform for real-time strategy games," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 2656–2666.
[152] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 5026–5033.
[153] J. Togelius, T. Schaul, D. Wierstra, C. Igel, F. Gomez, and J. Schmidhuber, "Ontogenetic and phylogenetic reinforcement learning," *Künstliche Intell.*, vol. 23, no. 3, pp. 30–33, 2009.
[154] N. Usunier, G. Synnaeve, Z. Lin, and S. Chintala, "Episodic exploration for deep deterministic policies: An application to StarCraft micromanagement tasks," 2016, arXiv:1609.02993.
[155] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.
[156] H. Van Seijen, R. Laroche, M. Fatemi, and J. Romoff, "Hybrid reward architecture for reinforcement learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5396–5406.
[157] O. Vinyals *et al.*, "Starcraft II: A new challenge for reinforcement learning," 2017, arXiv:1708.04782.
[158] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith, and S. Risi, "Evolving Mario levels in the latent space of a deep convolutional generative adversarial network," in *Proc. Genetic Evol. Comput. Conf.*, 2018, pp. 221–228.
[159] C. Wang, P. Chen, Y. Li, C. Holmgård, and J. Togelius, "Portfolio online evolution in StarCraft," in *Proc. 12th Artif. Intell. Interact. Digit. Entertainment Conf.*, 2016, pp. 114–121.
[160] Z. Wang *et al.*, "Sample efficient actor-critic with experience replay," 2017, arXiv:1611.01224.
[161] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 1995–2003.
[162] C. J. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3/4, pp. 279–292, 1992.
[163] M. Wiering and J. Schmidhuber, "HQ-learning," *Adaptive Behav.*, vol. 6, no. 2, pp. 219–246, 1997.
[164] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3/4, pp. 229–256, 1992.
[165] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Comput.*, vol. 1, no. 2, pp. 270–280, 1989.
[166] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5285–5294.
[167] Y. Wu and Y. Tian, "Training agent for first-person shooter game with actor-critic curriculum learning," in *Proc. Int. Conf. Learn. Represent.*, 2017.
[168] B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner, TORCS, The Open Racing Car Simulator, Software, 2000. [Online]. Available: http://torcs.sourceforge.net
[169] G. N. Yannakakis, P. Spronck, D. Loiacono, and E. André, "Player modeling," in *Dagstuhl Follow-Ups*, vol. 6. Wadern, Germany: Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.
[170] G. N. Yannakakis and J. Togelius, "A panorama of artificial and computational intelligence in games," *IEEE Trans. Comput. Intell. AI Games*, vol. 7, no. 4, pp. 317–335, Dec. 2015.
[171] G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*. New York, NY, USA: Springer, 2018.
[172] H. Yu, H. Zhang, and W. Xu, "A deep compositional framework for human-like language acquisition in virtual environment," 2017, arXiv:1703.09831.
[173] T. Zahavy, M. Haroush, N. Merlis, D. J. Mankowitz, and S. Mannor, "Learn what not to learn: Action elimination with deep reinforcement learning," in *Advances in Neural Information Processing Systems 31*, S. Bengio and H. Wallach and H. Larochelle and K. Grauman and N. Cesa-Bianchi and R. Garnett, Eds. New York, NY, USA: Curran Associates, 2018, pp. 3562–3573.