

Deep Learning for Video Game Playing

Niels Justesen, Philip Bontrager, Julian Togelius, Sebastian Risi

DRAFT VERSION. Send feedback and comments to: noju@itu.dk

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 or 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.

I. INTRODUCTION

Applying AI techniques to games is now an established research field, with multiple conferences and dedicated journals. In this article, we review recent advances in deep learning for video game playing and employed game research platforms while highlighting important open challenges. A main motivation for writing this article is to review the field from the perspective of **different types of games**, the challenges they pose for deep learning, and how deep learning can be used to play these games. A variety of different review articles on deep learning exists [26], [57], [92], as well as surveys on reinforcement learning [104] and deep reinforcement learning [63], here we focus on these techniques applied to video games.

In particular, in this article, 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 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 we are not covering in this article; AI and games is a large and diverse field [124], [123], [67], [25], [72]. In this paper, we focus on deep learning methods for playing video games well, but there is also plenty of research for playing games in a believable, entertaining or human-like manner [38]. Furthermore, AI is commonly used for tasks that do not involve playing the game, such as modeling players' behavior, experience or preferences [122], or generating game content such as levels, textures or rules [94]. Deep learning is also far from the only **AI method that has applications in games**, other prominent methods include Monte Carlo Tree Search [12] and evolutionary computation [85], [66]. In what follows, it is important to be aware of the limitations of the scope of this article.

The paper is structured as follows: The next section gives an overview of different deep learning 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 and Section V gives a historical overview of the field. We conclude the paper by pointing out important open challenges in Section VI and a conclusion in Section VII.

II. DEEP LEARNING OVERVIEW

Machine learning is traditionally divided into three different types of learning: supervised learning, unsupervised learning, and reinforcement learning. Additionally, one could also use stochastic optimization approaches like evolutionary computation for learning. All of them can be relevant to training deep networks to play games. In this section, we give a brief overview of these approaches, and also of hybrid approaches that combine one or several of these methods with each other or with other methods.

A. Supervised Learning

In supervised training of **artificial neural networks (ANNs)**, **an agent learns by example** [56], [86]. An agent is asked to make a **decision** for which the correct answer is already known. After the decision is made, an error function is used to determine the difference between the provided answer and the true one; this is used as a loss to update the model. Trained over a sufficiently large set of data, the agent should learn a general model that allows it to do well on inputs that have not been seen before.

The architectures of these neural networks can roughly be divided into two major categories: **feedforward and recurrent neural networks (RNN)**. Feedforward networks take a single input, for example, a representation of the game state, and select an output from a predefined set of possible outputs. Famously this is done with image classification, where an image is provided and a label pertaining to what is in the image is output. Feedforward networks with convolution, or convolutional neural networks (CNN), have been the most successful way to process images and this type of architecture will likely be relevant if raw image data from a game is going to be used as an input.

CNNs are a type of feedforward network [59]. They are typically **organized in layers where each layer transforms the data provided and passes it on to the next layer**. In a CNN, some layers consist of a number of small, trainable filters. These filters are convolved across the output of the previous layer. At each location of the filter over the output,

the dot product is taken as inputs for the next layer. These dot products are normally passed through an activation function first. Generally, these filters each learn to respond to certain features in the data, allowing the network to efficiently classify objects.

RNNs are typically applied to time series data, where the next output of the ANN is likely dependent on the previous ones [119], [58]. The training process is similar, except that the network's previous output is fed back into the network together with the next input. This allows the network to maintain the context of where it is in the series. This ability is useful in games where the available state information does not represent the complete state of the game and the context of past actions is important. Typically if an RNN is used with image data, the image is first preprocessed with a convolutional network and then converted into a vector that can be feed into the RNN.

While supervised learning has shown impressive results in a variety of different domains, it requires a large amount of training data that often has to be curated by humans. In games, this data can come from play trace data [10], i.e. humans playing through the game while being recorded. Through supervised learning, the agent can then 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.

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. This way, other algorithms can be used to determine the best action to take. While this method is able to use the game to generate as much data as necessary, it does not directly solve the challenge of playing a game well.

Supervised learning can be very effective, given that the correct solutions are known. However, it can also be very labor intensive and sometimes infeasible to collect enough training data. In cases when there is no training data available (e.g. playing an unknown game), or the available training data is insufficient, other training methods such as unsupervised or reinforcement learning are often applied. Reinforcement learning can also be applied to further improve on a policy that was learned through supervised training.

B. 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 is characteristic of the original data.

Within deep learning, there are several different techniques that allow unsupervised learning. One of the most prominent is the autoencoder, which is a neural network that attempts to output a copy of its input [56], [86]. The network consists of two parts: an encoder that maps the input to a hidden vector

h , and a decoder that constructs the copy from h . The loss is based on how well the copy matches the original, therefore no label is needed. The main idea is that by keeping h small, the network has to learn to compress the data and therefore learn a good representation.

So far unsupervised techniques in video games have mostly been applied in conjunction with other algorithms. Unsupervised learning algorithms can be used to generate predictions of future states of the game, which can then be fed into the main algorithm.

C. Reinforcement Learning Approaches

In reinforcement learning (RL) for games, an agent interacts with its environment to learn to play the game. The goal is to learn a policy, i.e. which action to take in each step in order to reach some desirable state. This situation often arises in video games, in which a player has a finite set of actions that can be taken at each step and the sequence of moves determines how well the player does.

For RL to work, the agent requires a reward signal, i.e. that some actions are followed by a positive or negative reward (r_t is the reward for step t). The rewards can be propagated back to all the actions that resulted in the reward, known as the return R , which is calculated as follows: $R = \sum_{t=0}^{\infty} \gamma^t r_{t+1}$, where γ is the discount factor defined $0 < \gamma \leq 1$. The goal of the agent is to learn a policy π , which determines the action a that maximizes the reward at each step t .

In value-based model-free RL, the policy is to select the action that maximizes the expected return, or value, for a given state. This method uses a function, $Q(s, a)$, which represents the expected value of R for a given state, s , and action. This method of reinforcement learning then tries to learn Q directly while leaving the policy fixed and is appropriately called Q-learning [117]. The basic learning algorithm for $Q(s_t, a_t)$ is:

$$Q(s_t, a_t) + \alpha_t \left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right),$$

where α_t is the learning rate at time t .

For deep learning applications, the Q-function is normally represented by a neural network. For each step, the parameters are updated to minimize the loss. The loss function is represented in the learning algorithm as $(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$.

There are other methods of RL that learn the policy directly. These are known as policy gradient methods. A policy gradient method represents the policy as π_θ where θ represents the parameters of the policy. In deep learning, the policy is a neural network and θ represents the weights and biases. A policy gradient method updates θ to optimize the expected reward of a state-action pair. So, where Q-learning predicts the value of each action, this directly predicts the action. The output can be used directly without any post processing.

The basic policy gradient algorithm from the REINFORCE family of algorithms [118] updates θ using the gradient $\nabla_\theta \log \pi_\theta(a|s) R_t$. This is the gradient of the policy with respect to the result at time t . To calculate R_t , the Q-function can be used that estimates R ; the Q-function can be trained as mentioned above. Using the Q-function to estimate R makes

this an actor-critic model where the actor updates θ and the critic updates Q [105].

D. Evolutionary Approaches

Another approach to training neural networks is based on evolutionary algorithms. This approach often referred to as neuroevolution (NE), can optimize the networks' weights as well as their topology. 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 both supervised and reinforcement learning problems. While NE approaches have been traditionally applied to problems with lower input dimensionality than typical deep learning approaches, recently Salimans et al. [89] showed that evolution strategies, which rely on parameter-exploration through stochastic noise instead of calculating gradients, can achieve results competitive to current deep RL approaches, given enough computational resources. For a complete overview of the application of NE in games, we refer the interested reader to our recent survey paper [85].

E. Hybrid Approaches

More recently researchers have started to investigate hybrid approaches for video game playing, which combine deep learning methods with other machine learning approaches. Both Alvernaz and Togelius [1] and Poulsen et al. [83] 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: deep learning methods are able to learn directly from high-dimensional raw pixels values, while evolutionary methods do not rely on differentiable architectures and work well in games with sparse rewards.

A seminal hybrid method for board game playing was AlphaGo [97], which relied on deep neural networks and tree search methods to defeat the world champion in Go.

In general, the hybridization of *ontogenetic* RL methods (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 [111].

III. GAME GENRES AND RESEARCH PLATFORMS

The fast progression of deep learning methods is undoubtedly helped by the fact that researchers can make comparisons on publicly available datasets, as they make it easy to evaluate new methods to the state-of-the-art. Similarly, game playing algorithms can be compared in game environments, in which methods are ranked based on their ability to score points in the games. Conferences like the IEEE Conference on Computational Intelligence in Games run popular competitions in a variety of these game environments (e.g. VizDoom, StarCraft).

This section describes popular game genres and research platforms, used in the literature, that are relevant to deep

learning; some examples are shown in Figure 1. 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 reinforcement learning benchmarks. In such classic control problems, the input is a simple feature vector, describing the position, velocity, and angles etc. A popular platform for such problems is rllab [19], which includes classic problems such as pole balancing and the mountain car problem. MuJoCo (Multi-Joint dynamics with Contact) is a physics engine for complex control tasks such as the humanoid walking task [110].

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 two-dimensional space (sometimes represented isometrically to provide the illusion of three-dimensional 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. Very 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 deep learning methods is the **Arcade Learning Environment (ALE)** [6]. ALE is built on top of the Atari 2600 emulator Stella and contains a total of 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. This platform was the main environment explored in the first deep reinforcement learning papers that used raw pixels as input. By enabling agents to learn from visual input, ALE thus differs from classic control problems in the reinforcement learning literature, such as the Cart Pole and Mountain Car problems. A more recent 2D-game like environment is MazeBase that

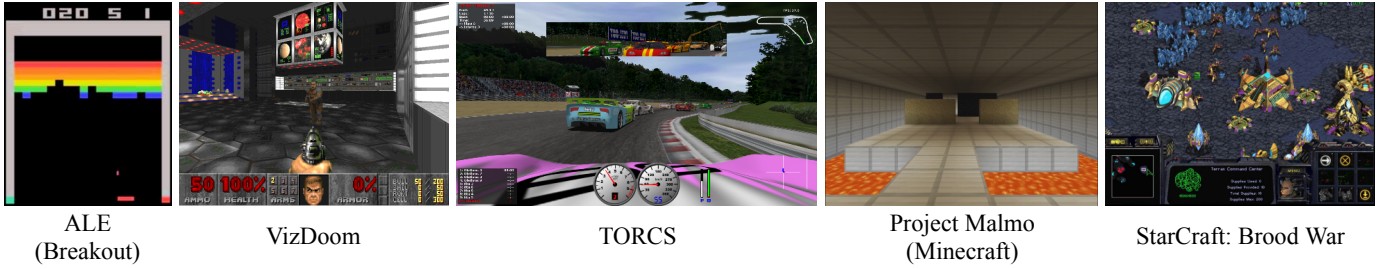


Fig. 1. Screenshots from selected games used as platforms for research in deep learning.

enables implementation of a diverse set of arcade-like games [102].

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) [9]. Many of these games have 3D graphics and the controller allows for over 720 action combinations. SNES games are thus more complex and realistic than Atari 2600 games, while this environment has not been as popular as ALE.

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 a 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. 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 reinforcement learning with realistic 3D graphics is the **open racing car simulator TORCS** [121].

C. First-Person Shooters (FPS)

More advanced game environments have recently emerged for visual reinforcement learning agents in a First-Person

Shooters (FPS)-type setting. In contrast to classic arcade games such as those in the ALE benchmark, FPSes have 3D 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 three-dimensional 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 added challenge of extracting relevant information from pixels.

Among FPS platforms are **VizDoom**, a framework that allows agents to play the classic first-person shooter *Doom* using the screen buffer as input [50]. **DeepMind Lab** is a platform for 3D navigation and puzzle-solving tasks based on the *Quake III Arena* engine [2].

D. Open-World Games

Open-world games such as *Minecraft* or *Grand Theft Auto V* are characterized by very non-linear 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 use these open environments to explore reinforcement learning methods that can reuse and transfer learned knowledge to new tasks. Project Malmö is a platform built on top of the open-world game *Minecraft*, which can be used to define many diverse and complex problems [43].

E. Real-time Strategy 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 classical

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 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. Real-time Strategy Games (RTS) 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* game series is without a doubt the most studied game in the Real-Time Strategy (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 deep learning has been applied. TorchCraft is a recent library built on top of BWAPI that connects the scientific computing framework Torch to *StarCraft* to enable machine learning research for this game [106]. 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 [114]. This API contains several mini-challenges while it also supports the full 1v1 game setting. Two minimalistic RTS game engines worth mentioning are *RTS* [77] and *ELF* [109], which implements some of the features that are present in RTS games.

F. OpenAI Gym & Universe

OpenAI Gym is a large platform for comparing reinforcement learning algorithms with a single interface to a suite of different environments including ALE, MuJoCo, Malmo, ViZ-Doom and more [11]. OpenAI Universe is a recent extension to the OpenAI Gym that currently interfaces with more than a thousand Flash games and aims to interface with many modern video games in the future².

IV. DEEP LEARNING METHODS FOR GAME PLAYING

This section gives an overview of deep learning techniques used to play video games, divided by game genre. An overview of the methods described in this section is shown in Table II and a typical neural network architecture used in Deep RL in Figure 2.

A. Arcade Games

The Arcade Learning Environment (ALE) has been the main testbed for deep reinforcement learning algorithms that learn control policies directly from raw pixels. This section reviews the main advancements that have been demonstrated in ALE in chronological order. Deep Q-Network (DQN) was the first learning algorithm that showed human expert level control in Atari games [70]. The algorithm was tested in seven Atari

2600 games and outperformed previous approaches, such as Sarsa with feature construction [3] and neuroevolution [34], as well as a human expert on three of the games. DQN is an *off-policy* reinforcement learning algorithm based on Q-learning, where the model learns a function $Q^\pi(s, a)$ that estimates the expected return of taking action a in state s using its own policy π . 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 [65], where experiences are stored in a replay memory and randomly sampled when the network is updated. This enables the algorithm to reuse past experiences, and learn from uncorrelated experiences, which reduces the variance of the updates. DQN was later extended with a separate target Q-network which parameters are held fixed between individual updates and was shown to achieve above human expert scores in 29 out of 49 tested games [71].

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 [35]. A distributed version of DQN was shown to outperform a non-distributed version in 41 of the 49 games using the Gorilla architecture (General Reinforcement Learning Architecture) [73]. Gorilla 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. *Double DQN*, based on double Q-learning [31], reduces the observed overestimation by learning two value networks, which use each other as a target network when being updated [113].

Another improvement came from *prioritized experience replay*, which samples important experiences more frequently from the replay memory based on the TD-error, and shows a significant improvement to both DQN and Double DQN [90].

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 $Q^\pi(s, a) = V^\pi(s) + A^\pi(s, a)$ [116]. Dueling DQN outperforms Double DQN and can also be combined with prioritized experience replay.

Double DQN and Dueling DQN was also tested in the five more complex games in the RLE and achieved a mean score around 50% of a human expert [9]. The best result in these experiments was by Dueling DQN in the game *Mortal Combat* with 128%.

Bootstrapped DQN improves the exploration policy and thus the training time 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 [79].

Multi-threaded asynchronous variants can utilize multiple CPU threads on a single machine and thus reduces the training roughly linear to the number of parallel threads [69]. These variants do not rely on a replay memory because several parallel actors will update the policy based on uncorrelated experiences. This feature can stabilize off-policy methods such as Sarsa and actor-critic methods. The Asynchronous

¹<http://bwapi.github.io/>

²<https://universe.openai.com/>

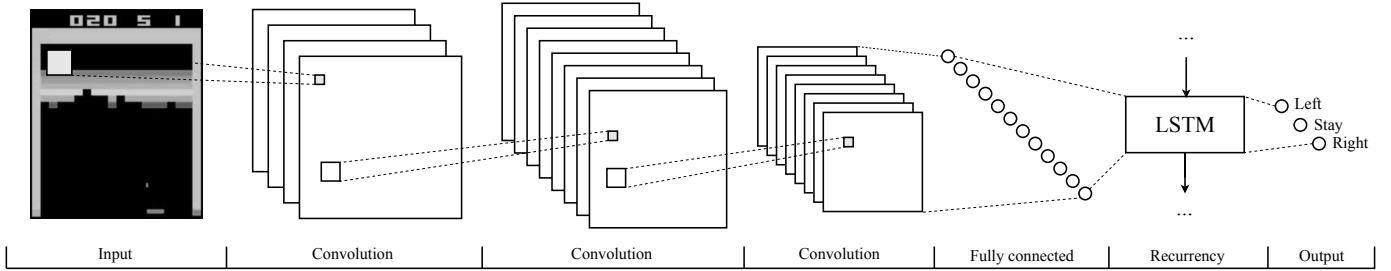


Fig. 2. An example of a typical network architecture used in deep reinforcement learning for game-playing. The input usually consists of a preprocessed screen image, or several concatenated images, which is followed by a couple of convolutional layers without pooling, and a few fully connected layers. Some recurrent networks have a recurrent layer after the fully connected layers. The output layer typically consists of one unit for each unique combination of actions in the game, and for actor-critic methods such as A3C, it also has one for the state value $V(s)$.

Advantage Actor-Critic (A3C) algorithm is an actor-critic method, which uses several parallel agents to explore the environment 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 [69].

A3C with progressive neural networks [88] can effectively transfer learning from one game to another and is done by instantiating a network for every new task it encounters. Each new network is instantiated with connections to all the previous learned networks, which gives access to general knowledge already learned.

The *UNREAL* (UNsupervised REinforcement and Auxiliary Learning) algorithm is similar to A3C but uses a replay memory from which it learns auxiliary tasks and *pseudo-reward functions* concurrently with the A3C loss [41]. *UNREAL* only shows a small improvement over vanilla A3C in ALE, but it shows larger improvements in other domains (see Section IV-D).

A variant of DQN called *C51* takes a distributional perspective on reinforcement learning by treating $Q(s, a)$ as an approximate distribution of returns instead of a single approximate expectation [5]. 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 it is, the better are the results, and with 51 atoms (hence the name C51), it achieved mean scores in ALE almost comparable to *UNREAL*.

Evolution Strategies (ES), a black-box optimization algorithm that relies on parameter-exploration through stochastic noise instead of calculating gradients, and was found to be highly parallelizable with a linear speedup in training time when more CPUs are used [89]. ES was trained on 720 CPUs for one hour and outperformed A3C (which was trained for 4 days) in 23 out of 51 games, while ES used between $3\times$ and $10\times$ as much data due to its high parallelization. However, the ES experiments were not run for several days, thus their full potential is currently unknown.

A few approaches also demonstrate how supervised learning can be applied to arcade games. In Guo et al. [28] 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

Results	Mean	Median	Year and orig. paper
DQN [116]	228%	79%	2013 [70]
Double DQN [116]	307%	118%	2015 [113]
Dueling DQN [116]	373%	151%	2015 [116]
Prior. DQN [116]	435%	124%	2015 [90]
Prior. Duel DQN [116]	592%	172%	2015 [90]
A3C [41]	853%	N/A	2016 [69]
UNREAL [41]	880%	250%	2016 [41]
C51 [5]	701%	178%	2017 [5]

TABLE I
COMPARABLE HUMAN-NORMALIZED SCORES OF DEEP REINFORCEMENT LEARNING ALGORITHMS TESTED IN ALE USING THE *30 no-ops* EVALUATION METRIC. ALGORITHMS ARE IN HISTORICAL ORDER. REFERENCES IN THE FIRST COLUMN REFER TO THE PAPER THAT INCLUDED THE RESULTS, WHILE THE LAST COLUMNS REFERENCE THE PAPER THAT INTRODUCED THE SPECIFIC TECHNIQUE.

shown to outperform DQN. Rusu et al. [87] used policy distillation to train a network to mimic one or several other policies (from several games), supervised on experiences from a replay memory, which can reduce the size of the network and sometimes also improve the performance. Oh et al. [76] showed that the encoding-transformation-decoding network architecture can be used to learn a frame prediction model from a dataset generated by a DQN agent, where after the model can be used to improve exploration in a retraining phase. 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 pre-training of a policy network, which ultimately can improve learning [96].

B. Montezuma's Revenge

Environments with sparse feedback remain an open challenge for reinforcement learning. 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 4,367 points and it is clear that the methods presented so far are unable to deal with environments with such sparse rewards. A

few methods aim to overcome these challenges with promising results.

Hierarchical-DQN (h-DQN) [53] 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 *meta-controller*, 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 with goals defining states in which the agent *reaches* (collides with) a certain type of object. This method, therefore, relies on some object detection mechanism.

Pseudo-counts have been used to provide intrinsic motivation in the form of exploration bonuses when unexpected pixel configurations are observed and can be derived from CTS density models [4] or neural density models [80]. Density models assign probabilities to images, and a model’s pseudo count 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) [4]. Interestingly, the results were less impressive when the CTS density model was combined with A3C (A3C-CTS) [4].

C. Racing Games

There are generally two paradigms for vision-based autonomous driving highlighted in Chen et al. [15]; (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 [15]. Direct perception was trained on recordings of 12 hours of human driving in TORCS and the trained system was able to drive in very diverse environments. Interestingly, the network was also able to generalize to real images.

End-to-end reinforcement learning algorithms such as DQN cannot be directly applied to continuous environments such as racing games because the action space must be discrete and with a relatively low dimensionality. Instead, policy gradient methods, such as actor-critic [17] and Deterministic Policy Gradient (DPG) [98] can learn policies in high-dimensional and continuous action spaces. Deep DPG is a policy gradient method that implements both a replay memory and a separate target network, which are both essential improvements to DQN, and was used to train a CNN end-to-end in TORCS from images [64].

The aforementioned A3C methods have also been applied to the racing game TORCS using only pixels as input [69]. In those experiments, rewards were shaped as the agent’s velocity on the track, and after 12 hours of training A3C reached a score between roughly 75% and 90% of a human tester in tracks with and without opponents bots.

While most approaches to training deep networks from high-dimensional input in video games have been based on some form of gradient descent, a notable exception is the approach by Koutník et al. [52], who evolved Fourier-type coefficients that encoded a recurrent network with over 1 million weights. 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. [50] 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 2016³, a number of participants submitted pre-trained neural network-based agents that competed in a multi-player deathmatch setting. Both a *limited* competition was held, in which bots competed in known levels, and a *full* competition that included bots competing in unseen levels. The winner of the limited track used a CNN trained with A3C using reward shaping and curriculum learning [120]. 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 [7]. 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 [55]. Position inference and object mapping from the screen and depth-buffers using Simultaneous Localization and Mapping (SLAM) have also shown to improve DQN in Doom [8].

The winner of the full deathmatch competition implemented a *Direct Future Prediction* (DFP) approach that was shown to outperform DQN and A3C [18]. The architecture used in DFP has three streams: one for the screen pixels, one for lower-dimensional measurements describing the agents 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 was shown to generalize to unseen goals at test time.

Navigation in 3D environments is one of the important skills required for FPS games and has been studied extensively. A CNN+LSTM network was trained with A3C extended with additional outputs that predict the depth of pixels as well as loop closure prediction, which showed significant improvements [68].

The *UNREAL* algorithm, based on A3C, implements an auxiliary reward prediction task that trains the network to also predict the immediate subsequent future reward from a

³<http://vizdoom.cs.put.edu.pl/competition-cig-2016>

TABLE II

OVERVIEW OF DEEP LEARNING METHODS APPLIED TO GAMES. AUX. = AUXILIARY. EM = EXTERNAL MEMORY. WE REFER TO *features* AS LOW-DIMENSIONAL ITEMS AND VALUES THAT DESCRIBE THE STATE OF THE GAME SUCH AS HEALTH, AMMUNITION, SCORE, OBJECTS, ETC. GLOBAL REFERS TO A FULL VIEW OF THE VISIBLE GAME, LOCAL IS AN AGENT’S PERCEPTION OF THE GAME, AND SHARED ARE MULTIPLE LOCAL AGENT VIEWS COMBINED.

Game(s)	Method	Network architecture	Input	Output
Atari 2600	DQN [70]	CNN	Pixels	Q-values
	DQN [71]	CNN	Pixels	Q-values
	DRQN [35]	CNN+LSTM	Pixels	Q-values
	UCTtoClassification [28]	CNN	Pixels	Action predictions
	Gorila [73]	CNN	Pixels	Q-values
	Double DQN [113]	CNN	Pixels	Q-values
	Prioritized DQN [90]	CNN	Pixels	Q-values
	Dueling DQN [116]	CNN	Pixels	Q-values
	Bootstrapped DQN [79]	CNN	Pixels	Q-values
	A3C [69]	CNN+LSTM	Pixels	Action probabilities & state-value
	UNREAL (A3C + aux. learning) [41]	CNN+LSTM	Pixels	Act. pr., state value & aux. predictions
	Scalable Evolution Strategies [89]	CNN	Pixels	Policy
	C51 [5]	CNN	Pixels	Q-values
Montezuma’s Revenge	H-DQN [53]	CNN	Pixels	Q-values
	DQN-CTS [4]	CNN	Pixels	Q-values
	DQN-PixelCNN [80]	CNN	Pixels	Q-values
Racing	Direct perception [15]	CNN	Pixels	Affordance indicators
	Deep DPG [64]	CNN	Pixels	Action probabilities & Q-values
	A3C [69]	CNN+LSTM	Pixels	Action probabilities & state value
Doom	DQN [50]	CNN+pooling	Pixels	Q-values
	A3C + curriculum learning [120]	CNN	Pixels	Action probabilities & state value
	DRQN + aux. learning [55]	CNN+GRU	Pixels	Q-values & aux. predictions
	DQN + SLAM [8]	CNN	Pixels & depth	Q-values
	DFP [18]	CNN	Pixels, features & goals	Feature prediction
Minecraft	H-DRLN [108]	CNN	Pixels	Policy
	RMQN/FRMQN [75]	CNN+LSTM+EM	Pixels	Q-values
StarCraft micromanagement	Zero Order [112]	Feed-forward ANN	Local & global features	Q-values
	IQL [22]	CNN+GRU	Local features	Q-values
	BiCNet [82]	Bi-directional RNN	Shared features	Action probabilities & Q-values
	COMA [21]	GRU	Local & global features	Action probabilities & state value
2D billiard	Object-centric prediction [24]	CNN+LSTM	Pixels & forces	Velocity predictions
Text-based games	LSTM-DQN [74]	LSTM+pooling	Text	Q-values

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% [41].

The ability to transfer knowledge to new environments can reduce the learning time and in some cases is crucial to learn extremely challenging tasks. Transfer learning can be achieved by pre-training a network on similar environments with simpler tasks or by using random textures during training [14].

The *Intrinsic Curiosity Module* (ICM), which consists of several neural networks, computes an intrinsic reward at 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 [81].

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 [108]. H-DRLN uses a variation of policy distillation [87] 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 [27]. 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 [75]. Using RMQN and FRMQN the agent learns to influence an external memory based on visual perceptions, which in turn influences the selected actions.

F. Real-Time Strategy 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. Real-Time Strategy (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 do not have an in-game scoring system and the only reward in the game 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 thus sub-problems are often studied individually.

For the simplistic RTS platform μ RTS a CNN was trained as a state evaluator using supervised learning on a generated

data set and used in combination with Monte Carlo Tree Search [100]. This approach performed significantly better than previous evaluation methods. μ RTS is however very simplistic and it would be non-trivial to apply this approach to more complex RTS games.

StarCraft has been a popular game platform for AI research, but so far only with a few deep learning approaches. Deep learning methods for StarCraft have mostly focused on micromanagement, i.e. unit control, and have so far ignored other aspects of the game. The problem of delayed rewards in StarCraft can be circumvented by focusing on micromanagement in combat scenarios; here rewards can be shaped as the difference between damage inflicted and damage incurred between states, giving immediate feedback [112], [22], [82], [21]. States are often described locally relative to a single unit, which is extracted from the game engine, and actions are likewise also relative to that specific unit. If agents are trained individually it is difficult to know which agents contributed to the global reward, a problem known as credit assignment [13]. One approach to learning behaviors of individual units in StarCraft 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 [112]. This strategy was able to learn successful policies for armies of up to 15 units, for which other algorithms have had difficulties.

Independent Q-learning (IQL) simplifies the multi-agent RL problem as agents learn a policy for controlling units individually while treating other agents as they were part of the environment [107]. 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 [22].

The *Multiagent Bidirectionally-Coordinated Network* (BiC-Net) implements a vectorized actor-critic framework based on a bi-directional RNN, with one dimension for every agent, and outputs a sequence of actions [82]. This network architecture is unique to the other approaches as it can handle an arbitrary number of units of different types with a single network.

Counterfactual multi-agent (COMA) policy gradients is an actor-critic method with a centralized critic and decentralized actors that address the multi-agent credit assignment problem with a counterfactual baseline computed by the critic network [21]. COMA achieves state-of-the-art results, for decentralized methods, in small combat scenarios with up to ten units on each side.

Deep learning has also recently been applied to learn macro-management tasks from game logs with supervised learning to predict strategic choices performed by humans in StarCraft [46]. The trained network was integrated as a module for an existing bot with promising results, outperforming the game's built-in bot.

G. 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 [24]. This predictive model could then be used for planning actions in the game.

A similar predictive approach was tested in a 3D game-like environment, using the Unreal Engine, where ResNet-34 [36] (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 [62]. Residual networks implement so-called shortcut connections that skip layers, which can improve learning in very deep networks.

H. Text-based Games

Text-based games, in which both states and actions are presented as text only, are a special video game genre. A network architecture called LSTM-DQN [74] 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-based games.

In many games with multiple agents, such as team based first-person shooters or strategy games, it is critical to communicate in order to solve problems where each agent only has partial information about the state. A deep distributed recurrent Q-network (DDRQN) architecture was used to train several agents to learn a communication protocol to solve the multi-agent *Hats* and *Switch* riddles [23]. 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.

In Section IV-B we reviewed RL methods based on intrinsic motivation for the hard Atari 2600 game Montezuma's Revenge. An instruction-based reinforcement learning approach that uses both a CNN for visual input and RNN for text-based instruction inputs managed to achieve a score of 3,500 points. Instructions were linked to positions in rooms and agents were rewarded when they reached those locations [48], 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 commands from text-based instructions in a 2D maze-like environment called XWORLD, such as walking to and picking up objects, after having learned the language of a *teacher* [125]. A RNN-based language module was connected to a perception module implementing a CNN, which were both connected to an action module used to select actions and a recognition module that learns the teacher's language in a question answering process.

V. HISTORICAL OVERVIEW OF DEEP LEARNING IN GAMES

The previous section discussed deep learning 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 deep learning methods that are reviewed in the previous section. Many of these methods are inspired or directly build upon previous methods, while some are applied to different game genres and again others are tailored to specific types of games.

Figure 3 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 [70] was very influential as an algorithm that uses gradient-based deep learning for pixel-based video game playing and was originally applied to the Atari benchmark. (Though note earlier approaches with less success such as [?], and successful non-gradient-based methods [85].) Double DQN [31] and Dueling DQN [116] are early extensions that use multiple networks to improve estimations. DRQN [35] uses a recurrent neural network as the Q network. Prioritized DQN [90] is another early extension and it adds improved experience replay sampling. Bootstrapped DQN [79] builds off of Double DQN with a different improved sampling strategy.

Gorila was the first asynchronous method based on DQN [73] and was followed by A3C [69] which uses multiple asynchronous agents instead of experience replay. This was further extended at the end of 2016 with UNREAL [41], which incorporates work done with auxiliary learning to handle sparse feedback environments.

Other techniques used on Atari include the C51 algorithm, which is based on DQN but changes the Q function. Other extensions are specifically for Montezuma’s revenge. Montezuma’s Revenge 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 [4]. These methods were published in 2016.

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 [120] proposes using curriculum learning with A3C. DRQN + Auxiliary Learning [55] extends DRQN by adding additional rewards during training. DQN + Slam [8] combines techniques for mapping unknown environments with DQN.

DFP [18] is the only approach that is not extending an Atari technique. Like UCT To Classification [28] for Atari, Object-centric Prediction [24] for Billiard, and Direct Perception [15] 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 Deep DPG [64], which extends DQN for continuous controls.

Of the major work on StarCraft, half of it has roots in Q-learning. The challenge in StarCraft is to handle many agents

and opponents; as can be seen in the figure, work in StarCraft based on Q-learning started in late 2016. IQL [22] extends DQN Prioritized DQN by treating all other agents as part of the environment. COMA [21] extends IQL by calculating counterfactual rewards, the marginal contribution each agent added. The other two methods, biCNet [82] and Zero Order Optimization [112], are reinforcement learning-based but are not derived from DQN.

Some work published 2016 extends DQN to play Minecraft [108]. At around the same time, techniques were developed to make DQN context aware and modular to handle the large state space [75].

DQN was applied to text-based games in 2015 [74] with no further work in this area.

The historic overview indicates that arcade games were the main environment used to develop and improve DQN and A3C which later have been applied to more complex games with various variations. Interestingly, while A3C outperforms algorithms based on DQN, more complex algorithms for Montezuma’s Revenge, StarCraft and Minecraft still rely on DQN.

VI. OPEN CHALLENGES

While deep learning and especially deep RL methods have shown remarkable results in video game playing, a multitude of important open challenges remains, which we review in this section. Indeed, looking back at the current state of research 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.

A. General Video Game Playing

Being able to solve a single problem does not make you intelligent; nobody would say that Deep Blue or AlphaGo [97] possess general intelligence, as they cannot even play Checkers (without re-training), much less make coffee or tie their shoe laces. To learn generally intelligent behavior, you need to train on not just a single task, but many different tasks [60]. 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 [91]. Yet, the vast majority of work on deep learning 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 adapt the network architecture to a different type of game.

Progress on the problem of playing multiple games includes progressive neural networks [88], 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. Another promising approach is elastic weight consolidation [51], in which multiple Atari games

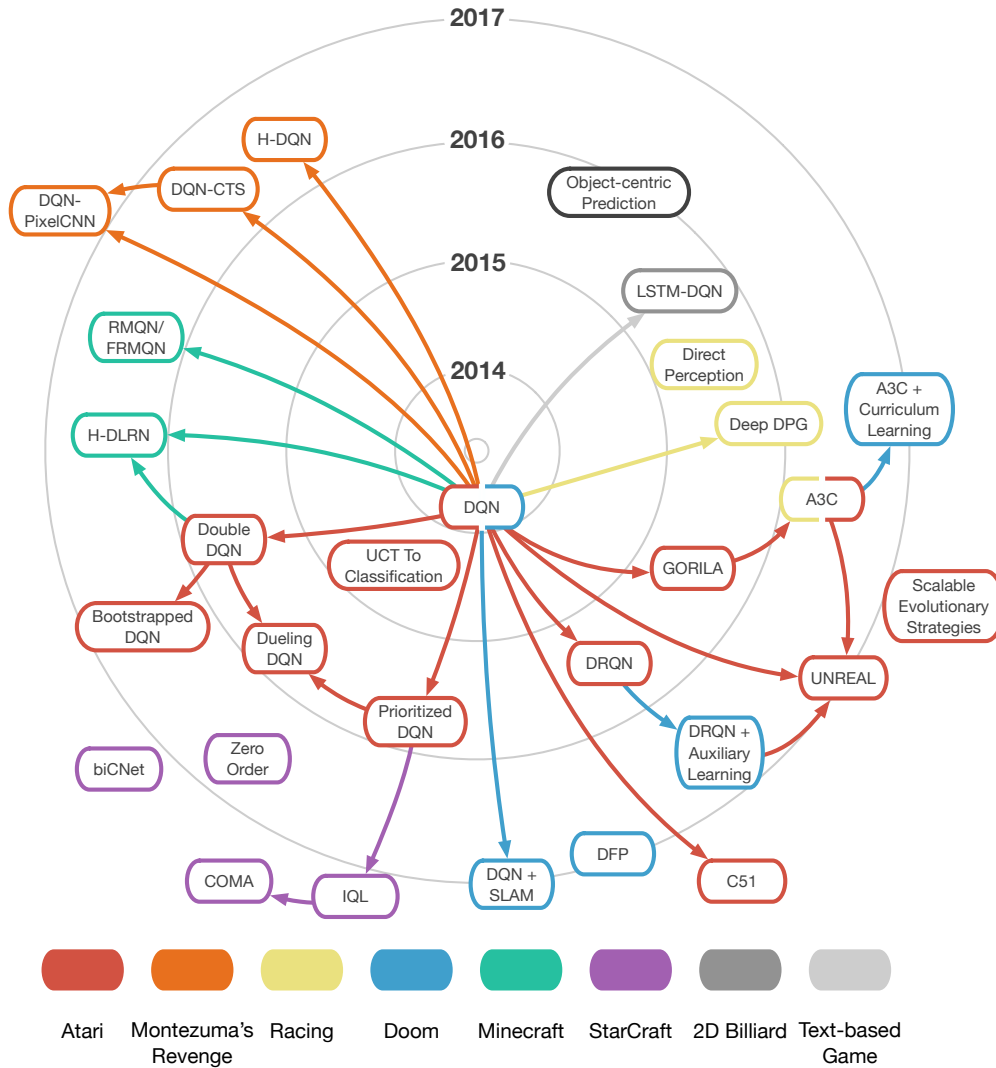


Fig. 3. Influence diagram of the deep learning techniques discussed in this paper. Each node is an algorithm while the color represents the game benchmark. The distance from the center represents the date that the original paper was published on arXiv. The arrows represent how techniques are related. Each node points to all other nodes that used or modified that technique. Influences not discussed in this paper are not represented. For details on the papers that introduced each method see Section V.

can be learned sequentially without catastrophic forgetting by protecting weights from being modified that are important for previously learned tasks. Yet another promising approach is PathNet, where 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 [20].

In the future it will be important to extend these methods to learn to play multiple games, even if those games are very different—current approaches focus on different (known) games in the ALE framework.

One important avenue for this kind of research will probably be the new Learning Track of the GVGAI competition, where some initial work has already been done [54]. Unlike in the ALE framework, where there is only a set number of games, the GVGAI framework has a potentially unlimited set of games.

It is possible that significant advances on the multi-game problem will come from outside deep learning. In particular, the recent Tangled Graph representation, a form of genetic programming, has shown promise in this task [49].

B. Games with very sparse rewards

Games such as Montezuma’s Revenge that are characterized by sparse rewards still pose a challenge for most Deep RL approaches. While recent advances that combine DQN with intrinsic motivation can help [4], games with very sparse rewards are still a challenge for current deep RL methods. There is a long history of research in intrinsically motivated reinforcement learning [?], [?] as well as hierarchic reinforcement learning which might be useful here [?], [?]. The Project Malmö environment, based on Minecraft, provides an excellent venue for creating tasks with very sparse rewards where agents need to set their own goals.

C. Multi-agent Learning

Current approaches of deep RL are mostly concerned with training a single agent. A few exceptions exist where multiple agents have to cooperate [61], [22], [112], [82], [21], 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 multi-agent 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.

D. Computational resources

Related to the previous challenge, if multiple agents in a large open-world are controlled by many deep networks, computational speed becomes a concern. Here methods that aim to make the networks computationally more efficient by either creating smaller network [40] or pruning the networks after training [32], [29] could be useful. Of course, improvements in processing power in general or for neural networks specifically will also be important. It is also not yet feasible to train networks in real-time to adapt to changes in the game or to fit the player's playing style, something which could be useful in the design of new types of games.

E. 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 that 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 the games industry is that such methods based are inherently difficult to control, which could result in unwanted NPC behaviors (e.g. an NPC in a game 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. In the future, it will be 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 [124]. Here the level of fun or engagement the player experiences while playing the game is a crucial component. One use of deep learning for game playing in the game production process is for game testing, where artificial agents are used for testing that levels are solvable or difficulty is appropriate. Deep learning might see its most prominent use in the games industry not for playing games, but for generating game content [94] based on training on existing content [103], or for modeling player experience [122].

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 deep learning. It remains to be seen whether these techniques will also be embraced by the development arms of these companies or their customers.

F. 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 these tools skill 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 could be one promising direction in this area. This approach has been extensively studied in the context of neuroevolution [85], and also in the context of video games, allowing non-expert users to breed behaviors for Super Mario [99]. Recently a similar preference based approach has been applied to deep RL method [16], allowing agents to learn Atari games based on a combination of learning human preferences and deep RL.

G. Creating new types of video games

DL could potentially offer a way to create completely new types of games, which might be important for adoption of these techniques in the game industry. 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 in games, meaning that games have been designed to not need AI. Designing new games *around* AI can help breaking out of these limitations. While evolutionary algorithms and neuroevolution in particular [85] have allowed the creation of completely new types of games, DL based on gradient descent has not been explored in this context yet. Neuroevolution, including evolution of neural nets, is a core mechanic in games such NERO [101], Galactic Arms Race [33], Petalz [84] and EvoCommander [42]. Compared to EAs, one challenge with gradient descent-based DL is that the structures are often limited to having mathematical smoothness (i.e. differentiability), which could make it more challenging to create interesting and unexpected outputs.

H. Lifetime Adaptation

While 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 a FPS map. However, most current DL techniques would require expensive re-training 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 is nowhere near that required

by common deep learning methods. This challenge is related to the wider problem of few-shot learning, and also to the problems of transfer learning and general video game playing, and solving it will be important to create more believable and human-like NPCs.

I. 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 has been an active area of research for some time, with two different competitions focused on human-like behavior, namely the *2k BotPrize* [37], [38] and the Turing Test track of the *Mario AI Championship* [95]. Most of the entries to these competitions are based on various non-neural network techniques, but some entries have used evolutionary training of deep neural nets to generate human-like behavior [93], [78].

J. Agents with adjustable performance level

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 both 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 play styles 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 [39]. However, this concept has not been implemented using deep learning, and it is still unclear how to realize the planning depth control using deep learning.

K. Learning models of games

Much work on deep learning 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 [124]. 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. [30] that learns a simple game engine of Super Mario Bros. from gameplay data. Kansky et al. [47] 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 [47]. A trained schema network thus provides a probabilistic model that can be used for planning, and is able to perform zero-shot transfer to similar environment in Breakout variations to those used in training.

L. 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 order of magnitudes larger. While recent advances in evolutionary planning have allowed real-time and long-term planning in games with larger branching factors to [44], [115], [45], how we can scale Deep RL to such levels of complexity is an important open challenge. Learning heuristics with deep learning in these games to enhance search algorithms is also a promising direction.

VII. CONCLUSION

This paper reviewed deep learning methods applied to game playing in video games of various genres including; arcade, racing, first-person shooters, open-world, real-time strategy, physics, and text-based games. Most of the reviewed work is within end-to-end model-free deep reinforcement learning, where a convolutional neural network learns to play directly from raw pixels by interacting with the game. Some of the reviewed work also demonstrate that supervised learning can be used to learn from game logs to imitate humans from the agents own interactions to learn about a model of the environment. For simple games, such as many arcade games, the reviewed methods can achieve above human-level performance, while there are many open challenges in more complex games.

REFERENCES

- [1] S. Alvernaz and J. Togelius. Autoencoder-augmented neuroevolution for visual doom playing. In *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*. IEEE, 2017.
- [2] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.
- [3] M. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.
- [4] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.

- [5] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*, 2017.
- [6] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47:253–279, 2013.
- [7] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.
- [8] S. Bhatti, A. Desmaison, O. Miksik, N. Nardelli, N. Siddharth, and P. H. Torr. Playing doom with slam-augmented deep reinforcement learning. *arXiv preprint arXiv:1612.00380*, 2016.
- [9] N. Bhonker, S. Rozenberg, and I. Hubara. Playing snes in the retro learning environment. *arXiv preprint arXiv:1611.02205*, 2016.
- [10] M. Bogdanovic, D. Markovikj, M. Denil, and N. De Freitas. Deep apprenticeship learning for playing video games. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.
- [11] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.
- [12] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavenor, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.
- [13] Y.-H. Chang, T. Ho, and L. P. Kaelbling. All learning is local: Multi-agent learning in global reward games. In *NIPS*, pages 807–814, 2003.
- [14] D. S. Chaplot, G. Lample, K. M. Sathyendra, and R. Salakhutdinov. Transfer deep reinforcement learning in 3d environments: An empirical study.
- [15] C. Chen, A. Seff, A. Kornhauser, and J. Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.
- [16] P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences, 2017.
- [17] T. Degris, P. M. Pilarski, and R. S. Sutton. Model-free reinforcement learning with continuous action in practice. In *American Control Conference (ACC)*, 2012, pages 2177–2182. IEEE, 2012.
- [18] A. Dosovitskiy and V. Koltun. Learning to act by predicting the future. *arXiv preprint arXiv:1611.01779*, 2016.
- [19] Y. Duan, X. Chen, R. Houthoofd, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016.
- [20] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.
- [21] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. Counterfactual multi-agent policy gradients. *arXiv preprint arXiv:1705.08926*, 2017.
- [22] J. Foerster, N. Nardelli, G. Farquhar, P. Torr, P. Kohli, S. Whiteson, et al. Stabilising experience replay for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1702.08887*, 2017.
- [23] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson. Learning to communicate to solve riddles with deep distributed recurrent q-networks. *arXiv preprint arXiv:1602.02672*, 2016.
- [24] K. Fragkiadaki, P. Agrawal, S. Levine, and J. Malik. Learning visual predictive models of physics for playing billiards. *arXiv preprint arXiv:1511.07404*, 2015.
- [25] L. Galway, D. Charles, and M. Black. Machine learning in digital games: a survey. *Artificial Intelligence Review*, 29(2):123–161, 2008.
- [26] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.
- [27] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.
- [28] 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 *Advances in neural information processing systems*, pages 3338–3346, 2014.
- [29] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pages 1379–1387, 2016.
- [30] M. Guzdial, B. Li, and M. O. Riedl. Game engine learning from video. In *International Joint Conference on Artificial Intelligence (IJCAI 2017)*, 2017.
- [31] H. V. Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.
- [32] B. Hassibi, D. G. Stork, et al. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, pages 164–164, 1993.
- [33] E. J. Hastings, R. K. Guha, and K. O. Stanley. Automatic content generation in the galactic arms race video game. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(4):245–263, 2009.
- [34] M. Hausknecht, J. Lehman, R. Miikkilainen, and P. Stone. A neuroevolution approach to general atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):355–366, 2014.
- [35] M. Hausknecht and P. Stone. Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*, 2015.
- [36] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.
- [37] P. Hingston. A new design for a turing test for bots. In *Computational Intelligence and Games (CIG)*, 2010 IEEE Symposium on, pages 345–350. IEEE, 2010.
- [38] P. Hingston. *Believable Bots: Can Computers Play Like People?* Springer, 2012.
- [39] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis. Generative agents for player decision modeling in games. In *FDG*, 2014.
- [40] 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. *arXiv preprint arXiv:1602.07360*, 2016.
- [41] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.
- [42] D. Jallo, S. Risi, and J. Togelius. Evocommander: A novel game based on evolving and switching between artificial brains. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(2):181–191, 2017.
- [43] M. Johnson, K. Hofmann, T. Hutton, and D. Bignell. The malmo platform for artificial intelligence experimentation. In *International joint conference on artificial intelligence (IJCAI)*, page 4246, 2016.
- [44] N. Justesen, T. Mahlmann, and J. Togelius. Online evolution for multi-action adversarial games. In *European Conference on the Applications of Evolutionary Computation*, pages 590–603. Springer, 2016.
- [45] N. Justesen and S. Risi. Continual online evolution for in-game build order adaptation in starcraft. In *The Genetic and Evolutionary Computation Conference (GECCO)*, 2017.
- [46] N. Justesen and S. Risi. Learning macromanagement in StarCraft from replays using deep learning. In *Computational Intelligence and Games, 2017. CIG 2017. IEEE Symposium on*. IEEE, 2017.
- [47] K. Kinsky, T. Silver, D. A. Mély, M. Eldawy, M. Lázaro-Gredilla, X. Lou, N. Dorfman, S. Sidor, S. Phoenix, and D. George. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. *arXiv preprint arXiv:1706.04317*, 2017.
- [48] R. Kaplan, C. Sauer, and A. Sosa. Beating atari with natural language guided reinforcement learning. *arXiv preprint arXiv:1704.05539*, 2017.
- [49] S. Kelly and M. I. Heywood. Multi-task learning in atari video games with emergent tangled program graphs. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 195–202. ACM, 2017.
- [50] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. *arXiv preprint arXiv:1605.02097*, 2016.
- [51] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, page 201611835, 2017.
- [52] J. Koutník, G. Cuccu, J. Schmidhuber, and F. Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1061–1068. ACM, 2013.
- [53] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 3675–3683, 2016.
- [54] K. Kuanusont, S. M. Lucas, and D. Perez-Liebana. General video game ai: Learning from screen capture. *arXiv preprint arXiv:1704.06945*, 2017.
- [55] G. Lample and D. S. Chaplot. Playing fps games with deep reinforcement learning. *arXiv preprint arXiv:1609.05521*, 2016.
- [56] Y. Le Cun. *Modèles connexionnistes de l'apprentissage*. PhD thesis, Paris 6, 1987.

- [57] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
- [58] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.
- [59] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [60] S. Legg and M. Hutter. Universal intelligence: A definition of machine intelligence. *Minds and Machines*, 17(4):391–444, 2007.
- [61] J. Z. Leibo, V. Zambaldi, M. Lanctot, J. Marecki, and T. Graepel. Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 464–473. International Foundation for Autonomous Agents and Multiagent Systems, 2017.
- [62] A. Lerer, S. Gross, and R. Fergus. Learning physical intuition of block towers by example. *arXiv preprint arXiv:1603.01312*, 2016.
- [63] Y. Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.
- [64] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.
- [65] L.-J. Lin. *Reinforcement learning for robots using neural networks*. PhD thesis, Fujitsu Laboratories Ltd, 1993.
- [66] S. M. Lucas and G. Kendall. Evolutionary computation and games. *IEEE Computational Intelligence Magazine*, 1(1):10–18, 2006.
- [67] R. Miikkulainen, B. D. Bryant, R. Cornelius, I. V. Karpov, K. O. Stanley, and C. H. Yong. Computational intelligence in games. *Computational Intelligence: Principles and Practice*, pages 155–191, 2006.
- [68] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.
- [69] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 2016.
- [70] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- [71] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- [72] H. Muñoz-Avila, C. Bauckhage, M. Bida, C. B. Congdon, and G. Kendall. Learning and game ai. In *Dagstuhl Follow-Ups*, volume 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.
- [73] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.
- [74] K. Narasimhan, T. Kulkarni, and R. Barzilay. Language understanding for text-based games using deep reinforcement learning. *arXiv preprint arXiv:1506.08941*, 2015.
- [75] J. Oh, V. Chockalingam, S. Singh, and H. Lee. Control of memory, active perception, and action in minecraft. *arXiv preprint arXiv:1605.09128*, 2016.
- [76] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems*, pages 2863–2871, 2015.
- [77] S. Ontanon. The combinatorial multi-armed bandit problem and its application to real-time strategy games. In *Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pages 58–64. AAAI Press, 2013.
- [78] J. Ortega, N. Shaker, J. Togelius, and G. N. Yannakakis. Imitating human playing styles in super mario bros. *Entertainment Computing*, 4(2):93–104, 2013.
- [79] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. Deep exploration via bootstrapped dqn. In *Advances In Neural Information Processing Systems*, pages 4026–4034, 2016.
- [80] G. Ostrovski, M. G. Bellemare, A. v. d. Oord, and R. Munos. Count-based exploration with neural density models. *arXiv preprint arXiv:1703.01310*, 2017.
- [81] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.
- [82] P. Peng, Q. Yuan, Y. Wen, Y. Yang, Z. Tang, H. Long, and J. Wang. Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games. *arXiv preprint arXiv:1703.10069*, 2017.
- [83] A. Precht, M. Thorhauge, M. Hvilshj, and S. Risi. DLNE: a hybridization of deep learning and neuroevolution for visual control. In *IEEE Conference on Computational Intelligence and Games (CIG 2017)*.
- [84] 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 *Aiide*, 2012.
- [85] S. Risi and J. Togelius. Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, 2015.
- [86] D. E. Rumelhart, G. E. Hinton, J. L. McClelland, et al. A general framework for parallel distributed processing. *Parallel distributed processing: Explorations in the microstructure of cognition*, 1:45–76, 1986.
- [87] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.
- [88] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.
- [89] T. Salimans, J. Ho, X. Chen, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.
- [90] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.
- [91] T. Schaul, J. Togelius, and J. Schmidhuber. Measuring intelligence through games. *arXiv preprint arXiv:1109.1314*, 2011.
- [92] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.
- [93] J. Schrum, I. V. Karpov, and R. Miikkulainen. Ut 2: Human-like behavior via neuroevolution of combat behavior and replay of human traces. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 329–336. IEEE, 2011.
- [94] N. Shaker, J. Togelius, and M. J. Nelson. *Procedural Content Generation in Games*. Springer, 2016.
- [95] N. Shaker, J. Togelius, G. N. Yannakakis, L. Poovanna, V. S. Ethiraj, S. J. Johansson, R. G. Reynolds, L. K. Heether, T. Schumann, and M. Gallagher. The turing test track of the 2012 mario ai championship: entries and evaluation. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8. IEEE, 2013.
- [96] E. Shelhamer, P. Mahmoudieh, M. Argus, and T. Darrell. Loss is its own reward: Self-supervision for reinforcement learning. *arXiv preprint arXiv:1612.07307*, 2016.
- [97] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.
- [98] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 387–395, 2014.
- [99] P. D. Sørensen, J. M. Olsen, and S. Risi. Breeding a diversity of super mario behaviors through interactive evolution. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, pages 1–7. IEEE, 2016.
- [100] M. Stanescu, N. A. Barriga, A. Hess, and M. Buro. Evaluating real-time strategy game states using convolutional neural networks. In *IEEE Conference on Computational Intelligence and Games (CIG 2016)*.
- [101] K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Real-time neuroevolution in the nero video game. *IEEE transactions on evolutionary computation*, 9(6):653–668, 2005.
- [102] S. Sukhbaatar, A. Szlam, G. Synnaeve, S. Chintala, and R. Fergus. Mazebase: A sandbox for learning from games. *arXiv preprint arXiv:1511.07401*, 2015.
- [103] A. Summerville, S. Snodgrass, M. Guzdial, C. Holmgård, A. K. Hoover, A. Isaksen, A. Nealen, and J. Togelius. Procedural content generation via machine learning (pcgml). *arXiv preprint arXiv:1702.00539*, 2017.
- [104] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.
- [105] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063, 1999.

- [106] G. Synnaeve, N. Nardelli, A. Auvolet, S. Chintala, T. Lacroix, Z. Lin, F. Richoux, and N. Usunier. Torchcraft: a library for machine learning research on real-time strategy games. *arXiv preprint arXiv:1611.00625*, 2016.
- [107] M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.
- [108] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor. A deep hierarchical approach to lifelong learning in minecraft. *arXiv preprint arXiv:1604.07255*, 2016.
- [109] Y. Tian, Q. Gong, W. Shang, Y. Wu, and L. Zitnick. Elf: An extensive, lightweight and flexible research platform for real-time strategy games. *arXiv preprint arXiv:1707.01067*, 2017.
- [110] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.
- [111] J. Togelius, T. Schaul, D. Wierstra, C. Igel, F. Gomez, and J. Schmidhuber. Ontogenetic and phylogenetic reinforcement learning. *Künstliche Intelligenz*, 23(3):30–33, 2009.
- [112] N. Usunier, G. Synnaeve, Z. Lin, and S. Chintala. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. *arXiv preprint arXiv:1609.02993*, 2016.
- [113] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100, 2016.
- [114] O. Vinyals, T. Ewalds, S. Bartunov, A. S. Georgiev, Petko Vezhnevets, M. Yeo, A. Makhzani, H. Kuttler, J. Agapiou, J. Schrittwieser, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. v. Hasselt, D. Silver, T. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, and R. Tsing. Starcraft ii: A new challenge for reinforcement learning. *manuscript*, 2017.
- [115] C. Wang, P. Chen, Y. Li, C. Holmgård, and J. Togelius. Portfolio online evolution in StarCraft. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.
- [116] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.
- [117] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.
- [118] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.
- [119] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.
- [120] Y. Wu and Y. Tian. Training agent for first-person shooter game with actor-critic curriculum learning. In *Submitted to Intl Conference on Learning Representations*, 2017.
- [121] B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner. Torcs, the open racing car simulator. *Software available at <http://torcs.sourceforge.net>*, 2000.
- [122] G. N. Yannakakis, P. Spronck, D. Loiacono, and E. André. Player modeling. In *Dagstuhl Follow-Ups*, volume 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.
- [123] G. N. Yannakakis and J. Togelius. A panorama of artificial and computational intelligence in games. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(4):317–335, 2015.
- [124] G. N. Yannakakis and J. Togelius. *Artificial Intelligence and Games*. Springer, 2017.
- [125] H. Yu, H. Zhang, and W. Xu. A deep compositional framework for human-like language acquisition in virtual environment. *arXiv preprint arXiv:1703.09831*, 2017.