A Brief Survey of Deep Reinforcement Learning

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, Anil Anthony Bharath

Abstract—**Deep reinforcement learning is poised to revolutionise the field of AI and represents a step towards building autonomous systems with a higher level understanding of the visual world. Currently, deep learning is enabling reinforcement learning to scale to problems that were previously intractable, such as learning to play video games directly from pixels. Deep reinforcement learning algorithms are also applied to robotics, allowing control policies for robots to be learned directly from camera inputs in the real world. In this survey, we begin with an introduction to the general field of reinforcement learning, then progress to the main streams of value-based and policy-based methods. Our survey will cover central algorithms in deep reinforcement learning, including the deep Q-network, trust region policy optimisation, and asynchronous advantage actor-critic. In parallel, we highlight the unique advantages of deep neural networks, focusing on visual understanding via reinforcement learning. To conclude, we describe several current areas of research within the field.**

I. Introduction

One of the primary goals of the field of artificial intelligence (AI) is to produce fully autonomous agents that interact with their environments to learn optimal behaviours, improving over time through trial and error. Crafting AI systems that are responsive and can effectively learn has been a long-standing challenge, ranging from robots, which can sense and react to the world around them, to purely software-based agents, which can interact with natural language and multimedia. A principled mathematical framework for experience-driven autonomous learning is reinforcement learning (RL) [115]. Although RL had some successes in the past [119, 109, 51, 79], previous approaches lacked scalablity and were inherently limited to fairly low-dimensional problems. These limitations exist because RL algorithms share the same complexity issues as other algorithms: memory complexity, computational complexity, and in the case of machine learning algorithms, sample complexity [113]. What we have witnessed in recent years—the rise of deep learning, relying on the powerful *function approximation* and *representation learning* properties of deep neural networks—has provided us with new tools to overcoming these problems.

The advent of deep learning has had a significant impact on many areas in machine learning, dramatically improving the state-of-the-art in tasks such as object detection, speech recognition, and language translation [59]. The most important property of deep learning is that deep neural networks can automatically find compact low-dimensional representations (features) of high-dimensional data (e.g., images, text and audio). Through crafting inductive biases into neural network architectures, particularly that of hierarchical representations, machine learning practitioners have made effective progress in addressing the curse of dimensionality [11]. Deep learning has similarly accelerated progress in RL, with the use of deep learning algorithms within RL defining the field of "deep reinforcement learning" (DRL). The aim of this survey is to cover both seminal and recent developments in DRL, conveying the innovative ways in which neural networks can be used to bring us closer towards developing autonomous agents.

Deep learning enables RL to scale to decision-making problems that were previously intractable, i.e., settings with high-dimensional state and action spaces. Amongst recent work in the field of DRL, there have been two outstanding success stories. The first, kickstarting the revolution in DRL, was the development of an algorithm that could learn to play a range of Atari 2600 video games at a superhuman level, directly from image pixels [71]. Providing solutions for the instability of function approximation techniques in RL, this work was the first to convincingly demonstrate that RL agents could be trained on raw, high-dimensional observations, solely based on a reward signal. The second standout success was the development of a hybrid DRL system, AlphaGo, that defeated a human world champion in Go [108], paralleling the historic achievement of IBM's Deep Blue in chess two decades earlier [15] and IBM's Watson DeepQA system that beat the best human Jeopardy! players [26]. Unlike the handcrafted rules that have dominated chess-playing systems, AlphaGo was comprised of neural networks that were trained using supervised and reinforcement learning, in combination with a traditional heuristic search algorithm.

DRL algorithms have already been applied to a wide range of problems, such as robotics, where control policies for robots can now be learned directly from camera inputs in the real world [63, 64], succeeding controllers that used to be hand-engineered or learned from low-dimensional features of the robot's state. In a step towards even more capable agents, DRL has been used to create agents that can meta-learn ("learn to learn") [25, 133], allowing them to generalise to complex visual environments they have never seen before [25]. In Figure 1, we showcase some of the domains that DRL has been applied to, ranging from playing video games [71] to indoor navigation [142].

Video games may be an interesting challenge, but learning how to play them is not the end goal of DRL. One of the driving forces behind DRL is the vision of creating systems that are capable of learning how to adapt in the real world. From managing power consumption [120] to picking and stowing objects [64], DRL stands to increase the amount of physical tasks that can be automated by learning. However, DRL does not stop there, as RL is a general way of approaching optimisation problems by trial and error. From designing state-of-the-art machine translation models [143] to constructing new optimisation functions [65], DRL has already been used to approach all manner of machine learning tasks.

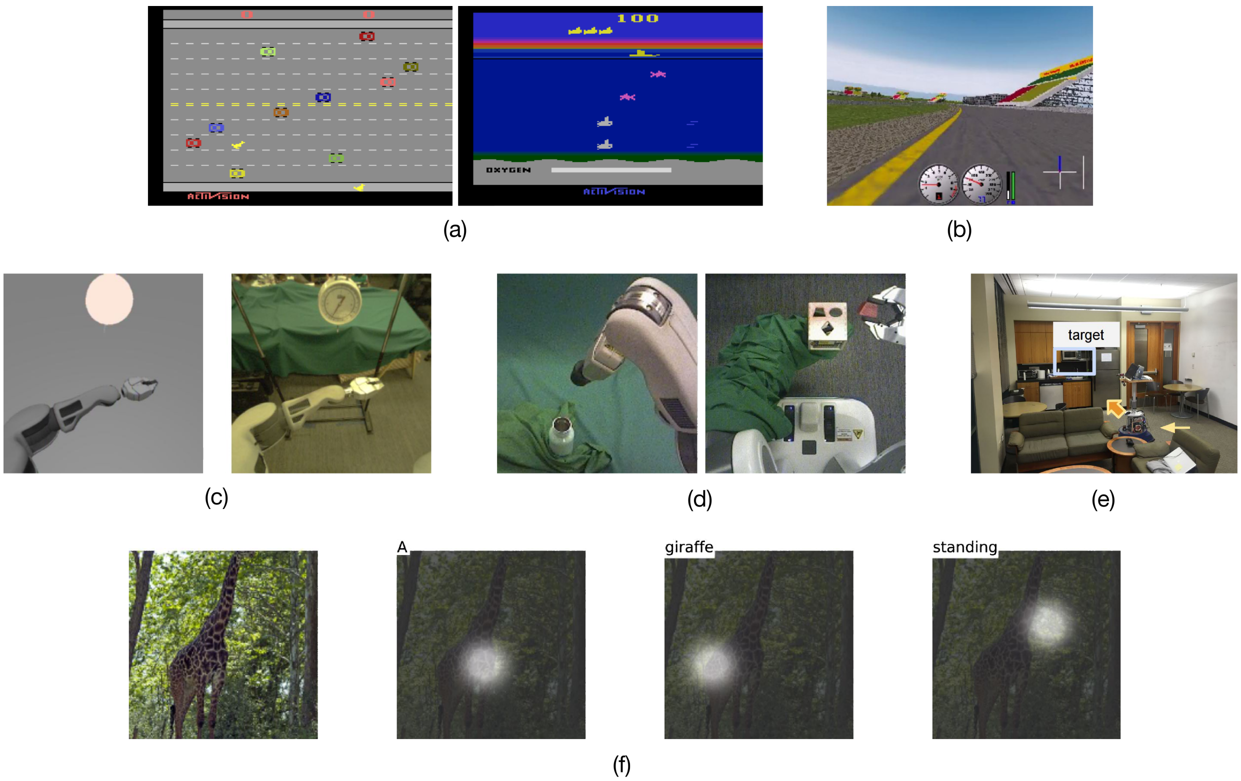

Fig. 1. A range of visual RL domains. **(a)** Two classic Atari 2600 video games, "Freeway" and "Seaquest", from the Arcade Learning Environment (ALE) [8]. Due to the range of supported games that vary in genre, visuals and difficulty, the ALE has become a standard testbed for DRL algorithms [71, 81, 35, 103, 112, 134, 72]. As we discuss later, the ALE is one of several benchmarks that are now being used to standardise evaluation in RL. **(b)** The TORCS car racing simulator, which has been used to test DRL algorithms that can output continuous actions [53, 67, 72] (as the games from the ALE only support discrete actions). **(c)** Utilising the potentially unlimited amount of training data that can be amassed in robotic simulators, several methods aim to transfer knowledge from the simulator to the real world [18, 97, 124]. **(d)** Two of the four robotic tasks designed by Levine et al. [63]: screwing on a bottle cap and placing a shaped block in the correct hole. Levine et al. [63] were able to train visuomotor policies in an end-to-end fashion, showing that visual servoing could be learned directly from raw camera inputs by using deep neural networks. **(e)** A real room, in which a wheeled robot trained to navigate the building is given a visual cue as input, and must find the corresponding location [142]. **(f)** A natural image being captioned by a neural network that uses reinforcement learning to choose where to look [141]. By processing a small portion of the image for every word generated, the network can focus its attention on the most salient points. Figures reproduced from [8, 67, 124, 63, 142, 141], respectively.

And, in the same way that deep learning has been utilised across many branches of machine learning, it seems likely that in the future, DRL will be an important component in constructing general AI systems [57].

II. Reward-driven Behaviour

Before examining the contributions of deep neural networks to RL, we will introduce the field of RL in general. The essence of RL is learning through *interaction*. An RL agent interacts with its environment and, upon observing the consequences of its actions, can learn to alter its own behaviour in response to rewards received. This paradigm of trial-and error-learning has its roots in behaviourist psychology, and is one of the main foundations of RL [115]. The other key influence on RL is optimal control, which has lent the mathematical formalisms (most notably dynamic programming [9]) that underpin the field.

In the RL set-up, an autonomous *agent*, controlled by a machine learning algorithm, observes a *state* $\mathbf{s}_t$ from its *environment* at timestep t. The agent interacts with the environment by taking an *action* $\mathbf{a}_t$ in state $\mathbf{s}_t$. When the agent takes an action, the environment and the agent transition to a new state $\mathbf{s}_{t+1}$ based on the current state and the chosen action. The state is a sufficient statistic of the environment and thereby comprises all the necessary information for the agent to take the best action, which can include parts of the agent, such as the position of its actuators and sensors. In the optimal control literature, states and actions are often denoted by $\mathbf{x}_t$ and $\mathbf{u}_t$, respectively.

The best sequence of actions is determined by the *rewards* provided by the environment. Every time the environment transitions to a new state, it also provides a scalar reward r_{t+1} to the agent as feedback. The goal of the agent is to learn a *policy* (control strategy) π that maximises the expected *return* (cumulative, discounted reward). Given a state, a policy returns an action to perform; an *optimal policy* is any policy that maximises the expected return in an environment. In this respect, RL aims to solve the same problem as optimal control. However, the challenge in RL is that the agent needs to learn about the consequences of actions in the environment by trial and error, as, unlike in optimal control, a model of the state transition dynamics is not available to the agent. Every interaction with the environment yields information, which the agent uses to update its knowledge. This *perception-action-*

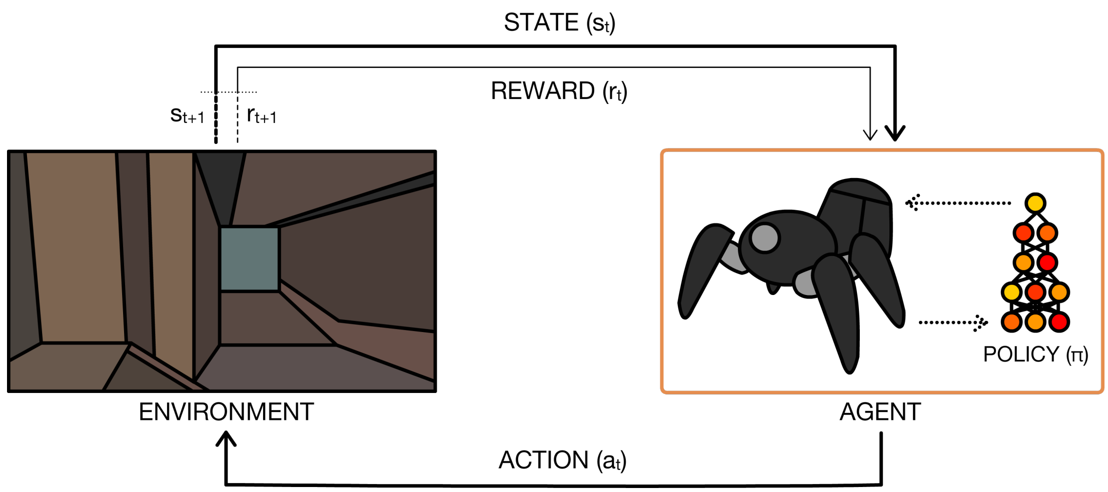

Fig. 2. The perception-action-learning loop. At time t, the agent receives state $\mathbf{s}_t$ from the environment. The agent uses its policy to choose an action $\mathbf{a}_t$. Once the action is executed, the environment transitions a step, providing the next state $\mathbf{s}_{t+1}$ as well as feedback in the form of a reward r_{t+1}. The agent uses knowledge of state transitions, of the form $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, r_{t+1})$, in order to learn and improve its policy.

learning loop is illustrated in Figure 2.

A. Markov Decision Processes

Formally, RL can be described as a Markov decision process (MDP), which consists of:

- A set of states $\mathcal{S}$, plus a distribution of starting states $p(\mathbf{s}_0)$.
- A set of actions $\mathcal{A}$.
- Transition dynamics $\mathcal{T}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ that map a state-action pair at time t onto a distribution of states at time $t+1$.
- An immediate/instantaneous reward function $\mathcal{R}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$.
- A discount factor $\gamma \in [0, 1]$, where lower values place more emphasis on immediate rewards.

In general, the policy π is a mapping from states to a probability distribution over actions: $\pi : \mathcal{S} \to p(\mathcal{A} = \mathbf{a}|\mathcal{S})$. If the MDP is *episodic*, i.e., the state is reset after each episode of length T, then the sequence of states, actions and rewards in an episode constitutes a *trajectory* or *rollout* of the policy. Every rollout of a policy accumulates rewards from the environment, resulting in the return $R = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$. The goal of RL is to find an optimal policy, π^*, which achieves the maximum expected return from all states:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}[R|\pi] \tag{1}$$

It is also possible to consider non-episodic MDPs, where $T = \infty$. In this situation, $\gamma < 1$ prevents an infinite sum of rewards from being accumulated. Furthermore, methods that rely on complete trajectories are no longer applicable, but those that use a finite set of transitions still are.

A key concept underlying RL is the Markov property, i.e., only the current state affects the next state, or in other words, the future is conditionally independent of the past given the present state. This means that any decisions made at $\mathbf{s}_t$ can be based solely on $\mathbf{s}_{t-1}$, rather than $\{\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_{t-1}\}$. Although this assumption is held by the majority of RL algorithms, it is somewhat unrealistic, as it requires the states to be *fully observable*. A generalisation of MDPs are partially observable MDPs (POMDPs), in which the agent receives an observation $\mathbf{o}_t \in \Omega$, where the distribution of the observation $p(\mathbf{o}_{t+1}|\mathbf{s}_{t+1}, \mathbf{a}_t)$ is dependent on the current state and the previous action [45]. In a control and signal processing context, the observation would be described by a measurement/ observation mapping in a state-space-model that depends on the current state and the previously applied action.

POMDP algorithms typically maintain a *belief* over the current state given the previous belief state, the action taken and the current observation. A more common approach in deep learning is to utilise recurrent neural networks (RNNs) [138, 35, 36, 72, 82], which, unlike feedforward neural networks, are dynamical systems. This approach to solving POMDPs is related to other problems using dynamical systems and state space models, where the true state can only be estimated [12].

B. Challenges in RL

It is instructive to emphasise some challenges faced in RL:

- The optimal policy must be inferred by trial-and-error interaction with the environment. The only learning signal the agent receives is the reward.
- The observations of the agent depend on its actions and can contain strong temporal correlations.
- Agents must deal with long-range time dependencies: Often the consequences of an action only materialise after many transitions of the environment. This is known as the (temporal) *credit assignment problem* [115].

We will illustrate these challenges in the context of an indoor robotic visual navigation task: if the goal location is specified, we may be able to estimate the distance remaining (and use it as a reward signal), but it is unlikely that we will know exactly what series of actions the robot needs to take to reach the goal. As the robot must choose where to go as it navigates the building, its decisions influence which rooms it sees and, hence, the statistics of the visual sequence captured. Finally, after navigating several junctions, the robot may find itself in a dead end. There are a range of problems, from learning the consequences of actions, to balancing exploration versus exploitation, but ultimately these can all be addressed formally within the framework of RL.

III. Reinforcement Learning Algorithms

So far, we have introduced the key formalism used in RL, the MDP, and briefly noted some challenges in RL. In the following, we will distinguish between different classes of RL algorithms. There are two main approaches to solving RL problems: methods based on *value functions* and methods based on *policy search*. There is also a hybrid, *actor-critic* approach, which employs both value functions and policy search. We will now explain these approaches and other useful concepts for solving RL problems.

A. Value Functions

Value function methods are based on estimating the value (expected return) of being in a given state. The *state-value function* $V^\pi(\mathbf{s})$ is the expected return when starting in state $\mathbf{s}$ and following π henceforth:

$$V^\pi(\mathbf{s}) = \mathbb{E}[R|\mathbf{s}, \pi] \tag{2}$$

The optimal policy, π^*, has a corresponding state-value function $V^*(\mathbf{s})$, and vice-versa, the optimal state-value function can be defined as

$$V^*(\mathbf{s}) = \max_\pi V^\pi(\mathbf{s}) \quad \forall \mathbf{s} \in \mathcal{S}. \tag{3}$$

If we had $V^*(\mathbf{s})$ available, the optimal policy could be retrieved by choosing among all actions available at $\mathbf{s}_t$ and picking the action $\mathbf{a}$ that maximises $\mathbb{E}_{\mathbf{s}_{t+1}\sim\mathcal{T}(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a})}[V^*(\mathbf{s}_{t+1})]$.

In the RL setting, the transition dynamics $\mathcal{T}$ are unavailable. Therefore, we construct another function, the *state-action-value* or *quality function* $Q^\pi(\mathbf{s}, \mathbf{a})$, which is similar to V^π, except that the initial action $\mathbf{a}$ is provided, and π is only followed from the succeeding state onwards:

$$Q^\pi(\mathbf{s}, \mathbf{a}) = \mathbb{E}[R|\mathbf{s}, \mathbf{a}, \pi]. \tag{4}$$

The best policy, given $Q^\pi(\mathbf{s}, \mathbf{a})$, can be found by choosing $\mathbf{a}$ greedily at every state: $\operatorname{argmax}_\mathbf{a} Q^\pi(\mathbf{s}, \mathbf{a})$. Under this policy, we can also define $V^\pi(\mathbf{s})$ by maximising $Q^\pi(\mathbf{s}, \mathbf{a})$: $V^\pi(\mathbf{s}) = \max_\mathbf{a} Q^\pi(\mathbf{s}, \mathbf{a})$.

Dynamic Programming: To actually learn Q^π, we exploit the Markov property and define the function as a Bellman equation [9], which has the following recursive form:

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{\mathbf{s}_{t+1}}[r_{t+1} + \gamma Q^\pi(\mathbf{s}_{t+1}, \pi(\mathbf{s}_{t+1}))] \tag{5}$$

This means that Q^π can be improved by *bootstrapping*, i.e., we can use the current values of our estimate of Q^π to improve our estimate. This is the foundation of Q-learning [136] and the state-action-reward-state-action (SARSA) algorithm [94]:

$$Q'(\mathbf{s}_t, \mathbf{a}_t) = Q^\pi(\mathbf{s}_t, \mathbf{a}_t) + \alpha\delta, \tag{6}$$

where α is the learning rate and $\delta = Y - Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$ the temporal difference (TD) error; here, Y is a target as in a standard regression problem. SARSA, an *on-policy* learning algorithm, is used to improve the estimate of Q^π by using transitions generated by the behavioural policy (the policy derived from Q^π), which results in setting $Y = r_t + \gamma Q^\pi(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})$. Q-learning is *off-policy*, as Q^π is instead updated by transitions that were not necessarily generated by the derived policy. Instead, Q-learning uses $Y = r_t + \gamma \max_\mathbf{a} Q^\pi(\mathbf{s}_{t+1}, \mathbf{a})$, which directly approximates Q^*.

To find Q^* from an arbitrary Q^π, we use *generalised policy iteration*, where policy iteration consists of *policy evaluation* and *policy improvement*. Policy evaluation improves the estimate of the value function, which can be achieved by minimising TD errors from trajectories experienced by following the policy. As the estimate improves, the policy can naturally be improved by choosing actions greedily based on the updated value function. Instead of performing these steps separately to convergence (as in policy iteration), generalised policy iteration allows for interleaving the steps, such that progress can be made more rapidly.

B. Sampling

Instead of bootstrapping value functions using dynamic programming methods, Monte Carlo methods estimate the expected return (2) from a state by averaging the return from multiple rollouts of a policy. Because of this, pure Monte Carlo methods can also be applied in non-Markovian environments. On the other hand, they can only be used in episodic MDPs, as a rollout has to terminate for the return to be calculated. It is possible to get the best of both methods by combining TD learning and Monte Carlo policy evaluation, as in done in the TD(λ) algorithm [115]. Similarly to the discount factor, the λ in TD(λ) is used to interpolate between Monte Carlo evaluation and bootstrapping. As demonstrated in Figure 3, this results in an entire spectrum of RL methods based around the amount of sampling utilised.

Another major value-function based method relies on learning the *advantage* function $A^\pi(\mathbf{s}, \mathbf{a})$ [3]. Unlike producing absolute state-action values, as with Q^π, A^π instead represents relative state-action values. Learning relative values is akin to removing a baseline or average level of a signal; more intuitively, it is easier to learn that one action has better consequences than another, than it is to learn the actual return from taking the action. A^π represents a relative advantage of actions through the simple relationship $A^\pi = V^\pi - Q^\pi$, and is also closely related to the baseline method of variance reduction within gradient-based policy search methods [139]. The idea of advantage updates has been utilised in many recent DRL algorithms [134, 34, 72, 104].

C. Policy Search

Policy search methods do not need to maintain a value function model, but directly search for an optimal policy

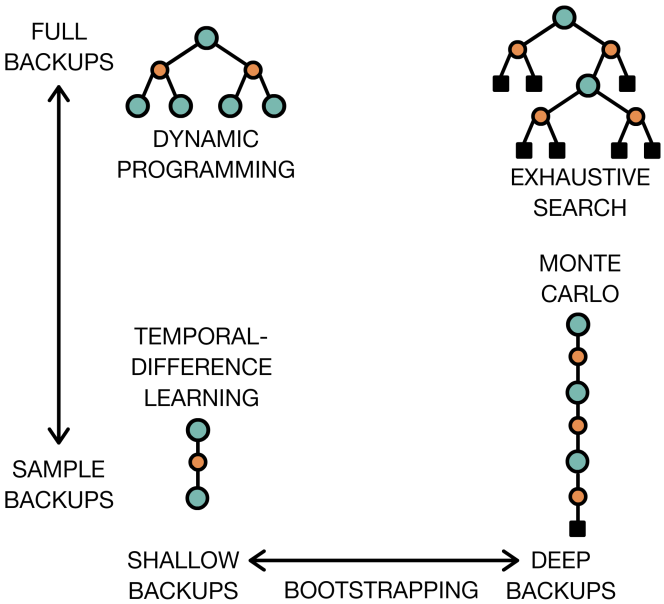

Fig. 3. Two dimensions of RL algorithms, based on the *backups* used to learn or construct a policy. Along the bottom is 1-step TD learning, n-step TD learning [115], and pure Monte Carlo approaches. Along the side is sampling actions versus taking the expectation over all choices. Recreated from [115].

π^*. Typically, a parameterised policy π_θ is chosen, whose parameters are updated to maximise the expected return $\mathbb{E}[R|\theta]$ using either gradient-based or gradient-free optimisation [21]. Neural networks that encode policies have been successfully trained using both gradient-free [32, 19, 53] and gradient-based [139, 138, 37, 67, 103, 104, 63] methods. Gradient-free optimisation can effectively cover low-dimensional parameter spaces, but despite some successes in applying them to large networks [53], gradient-based training remains the method of choice for most DRL algorithms, being more sample-efficient when policies possess a large number of parameters.

When constructing the policy directly, it is common to output parameters for a probability distribution; for continuous actions, this could be the mean and standard deviations of Gaussian distributions, whilst for discrete actions this could be the individual probabilities of a multinomial distribution. The result is a stochastic policy from which we can directly sample actions. With gradient-free methods, finding better policies requires a heuristic search across a predefined class of models. Methods such as evolution strategies essentially perform hill-climbing in a subspace of policies [98], whilst more complex methods, such as compressed network search, impose additional inductive biases [53]. Perhaps the greatest advantage of gradient-free policy search is that they can also optimise non-differentiable policies.

Policy Gradients: Gradients can provide a strong learning signal as to how to improve a parameterised policy. However, to compute the expected return (1) we need to average over plausible trajectories induced by the current policy parameterisation. This averaging requires either deterministic approximations (e.g., linearisation) or stochastic approximations via sampling [21]. Deterministic approximations can only be applied in a model-based setting where a model of the underlying transition dynamics is available. In the more common model-free RL setting, a Monte Carlo estimate of the expected return is determined. For gradient-based learning, this Monte Carlo approximation poses a challenge since gradients cannot pass through these samples of a stochastic function. Therefore, we turn to an estimator of the gradient, known in RL as the REINFORCE rule [139], elsewhere known as the score function [29] or likelihood-ratio estimator [31]. The latter name is telling as using the estimator is similar to the practice of optimising the log-likelihood in supervised learning. Intuitively, gradient ascent using the estimator increases the log probability of the sampled action, weighted by the return. More formally, the REINFORCE rule can be used to compute the gradient of an expectation over a function f of a random variable X with respect to parameters θ:

$$\nabla_\theta \mathbb{E}_X[f(X;\theta)] = \mathbb{E}_X[f(X;\theta)\nabla_\theta \log p(X)]. \qquad (7)$$

As this computation relies on the empirical return of a trajectory, the resulting gradients possess a high variance. By introducing unbiased estimates that are less noisy it is possible to reduce the variance. The general methodology for performing this is to subtract a baseline, which means weighting updates by an advantage rather than the pure return. The simplest baseline is the average return taken over several episodes [139], but there are many more options available [104].

Actor-critic Methods: It is possible to combine value functions with an explicit representation of the policy, resulting in actor-critic methods, as shown in Figure 6. The "actor" (policy) learns by using feedback from the "critic" (value function). In doing so, these methods trade off variance reduction of policy gradients with bias introduction from value function methods [52, 104].

Actor-critic methods use the value function as a baseline for policy gradients, such that the only fundamental difference between actor-critic methods and other baseline methods are that actor-critic methods utilise a *learned* value function. For this reason, we will later discuss actor-critic methods as a subset of policy gradient methods.

D. Planning and Learning

Given a model of the environment, it is possible to use dynamic programming over all possible actions (Figure 3, top left), sample trajectories for heuristic search (as was done by AlphaGo [108]), or even perform an exhaustive search (Figure 3, top right). Sutton and Barto [115] define *planning* as any method which utilises a model to produce or improve a policy. This includes *distribution models*, which include $\mathcal{T}$ and $\mathcal{R}$, and *sample models*, from which only samples of transitions can be drawn.

In RL, we focus on learning without access to the underlying model of the environment. However, interactions with the environment could be used to learn value functions, policies, and also a model. Model-free RL methods learn directly from interactions with the environment, but model-based RL methods can simulate transitions using the learned model, resulting in increased sample efficiency. This is particularly important in domains where each interaction with the environment is expensive. However, learning a model introduces extra complexities, and there is always the danger of suffering from model errors, which in turn affects the learned policy. Although deep neural networks can potentially produce very complex and rich models [81, 112, 27], sometimes simpler, more data-efficient methods are preferable [34]. These considerations also play a role in actor-critic methods with learned value functions [52, 104].

E. The Rise of DRL

Many of the successes in DRL have been based on scaling up prior work in RL to high-dimensional problems. This is due to the learning of low-dimensional feature representations and the powerful function approximation properties of neural networks. By means of representation learning, DRL can deal efficiently with the curse of dimensionality, unlike tabular and traditional non-parametric methods [11]. For instance, convolutional neural networks (CNNs) can be used as components of RL agents, allowing them to learn directly from raw, high-dimensional visual inputs. In general, DRL is based on training deep neural networks to approximate the optimal policy π^*, and/or the optimal value functions V^*, Q^* and A^*.

Following our review of RL, the next part of the survey is similarly partitioned into value function and policy search methods in DRL. In these sections, we will focus on state-of-the-art techniques, as well as the historical works they are built upon. The focus of the state-of-the-art techniques will be on those for which the state space is conveyed through visual inputs, e.g., images and video. To conclude, we will examine ongoing research areas and open challenges.

IV. Value Functions

The well-known function approximation properties of neural networks led naturally to the use of deep learning to regress functions for use in RL agents. Indeed, one of the earliest success stories in RL is TD-Gammon, a neural network that reached expert-level performance in Backgammon in the early 90s [119]. Using TD methods, the network took in the state of the board to predict the probability of black or white winning. Although this simple idea has been echoed in later work [108], progress in RL research has favoured the explicit use of value functions, which can capture the structure underlying the environment. From early value function methods in DRL, which took simple states as input [93], current methods are now able to tackle visually and conceptually complex environments [71, 103, 72, 82, 142].

A. Function Approximation and the DQN

We begin our survey of value-function-based DRL algorithms with the deep Q-network (DQN) [71], pictured in Figure 4, which achieved scores across a large range of classic Atari 2600 video games [8] that were comparable to that of a professional video games tester. The inputs to the DQN are four greyscale frames of the game, concatenated over time, which are initially processed by several convolutional layers in order to extract spatiotemporal features, such as the movement of the ball in "Pong" or "Breakout." The final feature map from the convolutional layers is processed by several fully connected layers, which more implicitly encode the effects of actions. This contrasts with more traditional controllers that use fixed preprocessing steps, which are therefore unable to adapt their processing of the state in response to the learning signal.

A forerunner of the DQN—neural fitted Q iteration (NFQ)—involved training a neural network to return the Q-value given a state-action pair [93]. NFQ was later extended to train a network to drive a slot car using raw visual inputs from a camera over the race track, by combining a deep autoencoder to reduce the dimensionality of the inputs with a separate branch to predict Q-values [58]. Although the previous network could have been trained for both reconstruction and RL tasks simultaneously, it was both more reliable and computationally efficient to train the two parts of the network sequentially.

The DQN [71] is closely related to the model proposed by Lange et al. [58], but was the first RL algorithm that was demonstrated to work directly from raw visual inputs and on a wide variety of environments. It was designed such that the final fully connected layer outputs $Q^\pi(\mathbf{s}, \cdot)$ for all action values in a discrete set of actions—in this case, the various directions of the joystick and the fire button. This not only enables the best action, $\mathrm{argmax}_{\mathbf{a}}\, Q^\pi(\mathbf{s}, \mathbf{a})$, to be chosen after a single forward pass of the network, but also allows the network to more easily encode action-independent knowledge in the lower, convolutional layers. With merely the goal of maximising its score on a video game, the DQN learns to extract salient visual features, jointly encoding objects, their movements, and, most importantly, their interactions. Using techniques originally developed for explaining the behaviour of CNNs in object recognition tasks, we can also inspect what parts of its view the agent considers important (see Figure 5).

The true underlying state of the game is contained within 128 bytes of Atari 2600 RAM. However, the DQN was designed to directly learn from visual inputs (210×160px 8-bit RGB images), which it takes as the state $\mathbf{s}$. It is impractical to represent $Q^\pi(\mathbf{s}, \mathbf{a})$ exactly as a lookup table: When combined with 18 possible actions, we obtain a Q-table of size $|\mathcal{S}| \times |\mathcal{A}| = 18 \times 256^{3 \times 210 \times 160}$. Even if it were feasible to create such a table, it would be sparsely populated, and information gained from one state-action pair cannot be propagated to other state-action pairs. The strength of the DQN lies in its ability to compactly represent both high-dimensional observations and the Q-function using deep neural networks. Without this ability tackling the discrete Atari domain from raw visual inputs would be impractical.

The DQN addressed the fundamental instability problem of using function approximation in RL [123] by the use of two techniques: experience replay [68] and target networks. Experience replay memory stores transitions of the form $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, r_{t+1})$ in a cyclic buffer, enabling the RL agent to sample from and train on previously observed data offline. Not only does this massively reduce the amount of interactions needed with the environment, but batches of experience can

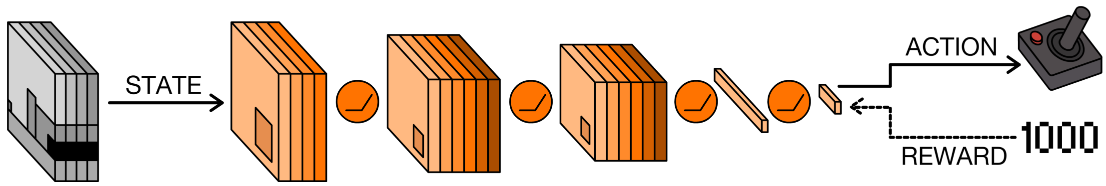

Fig. 4. The DQN [71]. The network takes the state—a stack of greyscale frames from the video game—and processes it with convolutional and fully connected layers, with ReLU nonlinearities in between each layer. At the final layer, the network outputs a discrete action, which corresponds to one of the possible control inputs for the game. Given the current state and chosen action, the game returns a new score. The DQN uses the reward—the difference between the new score and the previous one—to learn from its decision. More precisely, the reward is used to update its estimate of Q, and the error between its previous estimate and its new estimate is backpropagated through the network.

be sampled, reducing the variance of learning updates. Furthermore, by sampling uniformly from a large memory, the temporal correlations that can adversely affect RL algorithms are broken. Finally, from a practical perspective, batches of data can be efficiently processed in parallel by modern hardware, increasing throughput. Whilst the original DQN algorithm used uniform sampling [71], later work showed that prioritising samples based on TD errors is more effective for learning [100]. We note that although experience replay is typically thought of as a model-free technique, it could actually be considered a simple model [128].

The second stabilising method, introduced by Mnih et al. [71], is the use of a target network that initially contains the weights of the network enacting the policy, but is kept frozen for a large period of time. Rather than having to calculate the TD error based on its own rapidly fluctuating estimates of the Q-values, the policy network uses the fixed target network. During training the weights of the target network are updated to match the policy network after a fixed number of steps. Both experience replay and target networks have gone on to be used in subsequent DRL works [34, 67, 135, 76].

B. Q-Function Modifications

Considering that one of the key components of the DQN is a function approximator for the Q-function, it can benefit from fundamental advances in RL. van Hasselt [126] showed that the single estimator used in the Q-learning update rule overestimates the expected return due to the use of the maximum action value as an approximation of the maximum *expected* action value. Double-Q learning provides a better estimate through the use of a double estimator [126]. Whilst double-Q learning requires an additional function to be learned, later work proposed using the already available target network from the DQN algorithm, resulting in significantly better results with only a small change in the update step [127].

Yet another way to adjust the DQN architecture is to decompose the Q-function into meaningful functions, such as constructing Q^π by adding together separate layers that compute the state-value function V^π and advantage function A^π [134]. Rather than having to come up with accurate Q-values for all actions, the duelling DQN [134] benefits from a single baseline for the state in the form of V^π, and easier-to-learn relative values in the form of A^π. The combination of the duelling DQN with prioritised experience replay [100] is one of the state-of-the-art techniques in discrete action settings. Further insight into the properties of A^π by Gu et al. [34] led them to modify the DQN with a convex advantage layer that extended the algorithm to work over sets of continuous actions, creating the normalised advantage function (NAF) algorithm. Benefiting from experience replay, target networks and advantage updates, NAF is one of several state-of-the-art techniques in continuous control problems [34].

V. Policy Search

Policy search methods aim to directly find policies by means of gradient-free or gradient-based methods. Prior to the current surge of interest in DRL, several successful methods in DRL eschewed the commonly used backpropagation algorithm in favour of evolutionary algorithms [32, 19, 53], which are gradient-free policy search algorithms. Evolutionary methods rely on evaluating the performance of a population of agents. Hence, they are expensive for large populations or agents with many parameters. However, as black-box optimisation methods they can be used to optimise arbitrary, non-differentiable models and naturally allow for more exploration in parameter space. In combination with a compressed representation of neural network weights, evolutionary algorithms can even be used to train large networks; such a technique resulted in the first deep neural network to learn an RL task, straight from high-dimensional visual inputs [53]. Recent work has reignited interest in evolutionary methods for RL as they can potentially be distributed at larger scales than techniques that rely on gradients [98].

A. Backpropagation through Stochastic Functions

The workhorse of DRL, however, remains backpropagation. The previously discussed REINFORCE rule [139] allows neural networks to learn stochastic policies in a task-dependent manner, such as deciding where to look in an image to track [102], classify [70] or caption objects [141]. In these cases, the stochastic variable would determine the coordinates of a small crop of the image, and hence reduce the amount of computation needed. This usage of RL to make discrete, stochastic decisions over inputs is known in the deep learning literature as *hard attention*, and is one of the more compelling uses of basic policy search methods in recent years, having many applications outside of traditional RL domains.

One of the notable new methods in policy search is trust region policy optimisation (TRPO), which guarantees monotonic improvement in the policy by preventing it from deviating too

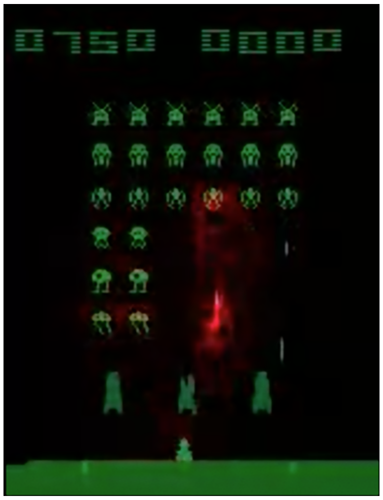

Fig. 5. Saliency map of a trained DQN [71] playing "Space Invaders" [8]. By backpropagating the training signal to the image space, it is possible to see what a neural-network-based agent is attending to. In this frame, the most salient points—shown with the red overlay—are the laser that the agent recently fired, and also the enemy that it anticipates hitting in a few time steps.

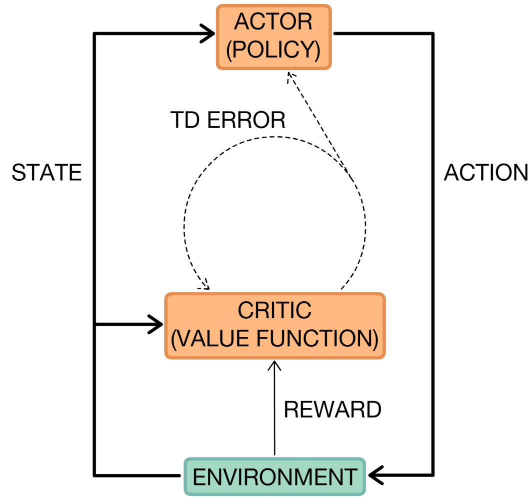

Fig. 6. Actor-critic set-up. The actor (policy) receives a state from the environment and chooses an action to perform. At the same time, the critic (value function) receives the state and reward resulting from the previous interaction. The critic uses the TD error calculated from this information to update itself and the actor. Recreated from [115].

wildly from previous policies [103]. On top of standard policy gradient methods, TRPO uses the notion of a trust region, which restricts optimisation steps to within a region where the approximation of the true cost function still holds. The idea of constraining policy gradient updates was explored earlier through the use of "natural gradient" updates [46] and also by means of the Kullback-Leibler (KL) divergence [48, 88]. In contrast with previous works, TRPO constrains each policy update to a fixed KL divergence from the current policy inducing the action conditional $p(\mathbf{a}|\mathbf{s})$, which is more feasible for use with current networks. Later work by Schulman et al. [104] introduced generalised advantage estimation (GAE), proposing more advanced variance reduction baselines for policy gradient methods. The combination of TRPO and GAE remains one of the state-of-the-art RL techniques in continuous control.

Searching directly for a policy represented by a neural network with very many parameters can be difficult and can suffer from severe local minima. One way around this is to use guided policy search (GPS), which takes a few sequences of actions from another controller (which could be constructed using a separate method, such as optimal control). GPS learns from them by using supervised learning in combination with importance sampling, which corrects for off-policy samples [62]. This approach effectively biases the search towards a good (local) optimum. GPS works in a loop, by optimising policies to match sampled trajectories, and optimising trajectory distributions to match the policy and minimise costs. Initially, GPS was used to train neural networks on simulated continuous RL problems [61], but was later utilised to train a policy for a real robot based on visual inputs [63]. This research by Levine et al. [63] showed that it was possible to train visuomotor policies for a robot "end-to-end", straight from the RGB pixels of the camera to motor torques, and, hence, is one of the seminal works in DRL.

B. Actor-Critic Methods

Instead of utilising Monte Carlo returns as baselines for policy gradient methods, actor-critic approaches have grown in popularity as an effective means of combining the benefits of policy search methods with learned value functions, which are able to learn from TD errors. They can benefit from improvements in both policy gradient methods, such as GAE [104], and value function methods, such as target networks [71]. In the last few years, DRL actor-critic methods have been scaled up from learning simulated physics tasks [37, 67] to real robotic visual navigation tasks [142], directly from image pixels.

One recent development in the context of actor-critic algorithms are deterministic policy gradients (DPGs) [107], which extend the standard policy gradient theorems for stochastic policies [139] to deterministic policies. One of the major advantages of DPGs is that, whilst stochastic policy gradients integrate over both state and action spaces, DPGs only integrate over the state space, requiring fewer samples in problems with large action spaces. In the initial work on DPGs, Silver et al. [107] introduced and demonstrated an off-policy actor-critic algorithm that vastly improved upon a stochastic policy gradient equivalent in high-dimensional continuous control problems. Later work introduced deep DPG (DDPG), which utilised neural networks to operate on high-dimensional, visual state spaces [67]. In the same vein as DPGs, Heess et al. [37] devised a method for calculating gradients to optimise stochastic policies, by "reparameterising" [50, 92] the stochasticity away from the network, thereby allowing standard gradients to be used (instead of the high-variance REINFORCE estimator [139]). The resulting stochastic value gradient (SVG) methods are flexible, and can be used both with (SVG(0) and SVG(1)) and without (SVG(∞)) value function critics, and with (SVG(∞) and SVG(1)) and without (SVG(0)) learned models. Later work proceeded to integrate DPGs and SVGs with RNNs, allowing them to solve continuous control problems in POMDPs, learning directly from pixels [36]. Together, DPGs and SVGs can be considered algorithmic approaches for improving learning efficiency in DRL.

An orthogonal approach to speeding up learning is to exploit parallel computation. In particular, methods for training networks through asynchronous gradient updates have been developed for use on both single machines [91] and distributed systems [20]. By keeping a canonical set of parameters that are read by and updated in an asynchronous fashion by multiple copies of a single network, computation can efficiently be distributed over both processing cores in a single CPU, and across CPUs in a cluster of machines. Using a distributed system, Nair et al. [77] developed a framework for training multiple DQNs in parallel, achieving both better performance and a reduction in training time. However, the simpler asynchronous advantage actor-critic (A3C) algorithm [72], developed for both single and distributed machine settings, has become one of the most popular DRL techniques in recent times. A3C combines advantage updates with the actor-critic formulation, and relies on asynchronously updated policy and value function networks trained in parallel over several processing threads. The use of multiple agents, situated in their own, independent environments, not only stabilises improvements in the parameters, but conveys an additional benefit in allowing for more exploration to occur. A3C has been used as a standard starting point in many subsequent works, including the work of Zhu et al. [142], who applied it to robotic navigation in the real world through visual inputs.

There have been several major advancements on the original A3C algorithm that reflect various motivations in the field of DRL. The first is actor-critic with experience replay [135], which adds Retrace(λ) off-policy bias correction [75] to A3C, allowing it to use experience replay in order to improve sample complexity. Others have attempted to bridge the gap between value and policy-based RL, utilising the theoretical advancements to improve upon the original A3C [76, 80, 105]. Finally, there is a growing trend towards exploiting auxiliary tasks to improve the representations learned by DRL agents, and, hence, improve both the learning speed and final performance of these agents [66, 43, 69].

VI. Current Research and Challenges

To conclude, we will highlight some current areas of research in DRL, and the challenges that still remain. Previously, we have focused mainly on model-free methods, but we will now examine a few model-based DRL algorithms in more detail. Model-based RL algorithms play an important role for data-efficient RL, but also in trading off exploration with exploitation. After tackling exploration strategies, we shall then address hierarchical RL (HRL), which imposes an inductive bias on the final policy by explicitly factorising it into several levels. When available, trajectories from other controllers can be used to bootstrap the learning process, leading us to imitation learning and inverse RL (IRL). For the final topic specific to RL, we will look at multi-agent systems, which have their own special considerations. We then bring to attention two broader areas—the use of RNNs, and transfer learning—in the context of DRL. We then examine the issue of evaluating RL, and current benchmarks for DRL.

A. Model-based RL

The key idea behind model-based RL is to learn a transition model that allows for simulation of the environment without interacting with the environment directly. Model-based RL does not assume specific prior knowledge. However, in practice, we can incorporate prior knowledge (e.g., physics-based models [47]) to speed up learning. Model learning plays an important role to reduce the amount of required interactions with the (real) environment, which may be limited in practice. For example, it is unrealistic to perform millions of experiments with a robot in a reasonable amount of time and without significant hardware wear and tear. There are various approaches to learn predictive models of dynamical systems using pixel information. Based on the deep dynamical model [131], where high-dimensional observations are embedded into a lower-dimensional space using autoencoders, several model-based DRL algorithms have been proposed for learning models and policies from pixel information [81, 137, 132]. If a sufficiently accurate model of the environment can be learned, then even simple controllers can be used to control a robot directly from camera images [27]. Learned models can also be used to guide exploration purely based on simulation of the environment, with deep models allowing these techniques to be scaled up to high-dimensional visual domains [112].

Although deep neural networks can make reasonable predictions in simulated environments over hundreds of timesteps [17], they typically require many samples to tune the large amount of parameters they contain. Training these models often requires more samples (interaction with the environment) than simpler models. For this reason, Gu et al. [34] train locally linear models for use with the NAF algorithm—the continuous equivalent of the DQN [71]—to improve the algorithm's sample complexity in the robotic domain where samples are expensive. It seems likely that the usage of deep models in model-based DRL could be massively spurred by general advances in improving the data efficiency of neural networks.

B. Exploration vs. Exploitation

One of the greatest difficulties in RL is the fundamental dilemma of *exploration versus exploitation*: When should the agent try out (perceived) non-optimal actions in order to explore the environment (and potentially improve the model), and when should it exploit the optimal action in order to make useful progress? Off-policy algorithms, such as the DQN [71], typically use the simple ϵ-greedy exploration policy, which chooses a random action with probability $\epsilon \in [0, 1]$, and the optimal action otherwise. By decreasing ϵ over time, the agent progresses towards exploitation. Although adding independent noise for exploration is usable in continuous control problems, more sophisticated strategies inject noise that is correlated over time (e.g., from stochastic processes) in order to better preserve momentum [67].

The observation that temporal correlation is important led Osband et al. [83] to propose the bootstrapped DQN, which maintains several Q-value "heads" that learn different values through a combination of different weight initialisations and bootstrapped sampling from experience replay memory. At the beginning of each training episode, a different head is chosen, leading to temporally-extended exploration. Usunier et al. [125] later proposed a similar method that performed exploration in policy space by adding noise to a single output head, using zero-order gradient estimates to allow backpropagation through the policy.

One of the main principled exploration strategies is the *upper confidence bound* (UCB) algorithm, based on the principle of "optimism in the face of uncertainty" [56]. The idea behind UCB is to pick actions that maximise $\mathbb{E}[\mathbb{R}] + \kappa\sigma[R]$, where $\sigma[R]$ is the standard deviation of the return and $\kappa > 0$. UCB therefore encourages exploration in regions with high uncertainty and moderate expected return. Whilst easily achievable in small tabular cases, the use of powerful density models has allowed this algorithm to scale to high-dimensional visual domains with DRL [7]. UCB is only one technique for trading off exploration and exploitation in the context of Bayesian optimisation [106]; future work in DRL may benefit from investigating other successful techniques that are used in Bayesian optimisation.

UCB can also be considered one way of implementing *intrinsic motivation*, which is a general concept that advocates decreasing uncertainty/making progress in learning about the environment [101]. There have been several DRL algorithms that try to implement intrinsic motivation via minimising model prediction error [112, 86] or maximising information gain [73, 42].

C. Hierarchical RL

In the same way that deep learning relies on hierarchies of features, HRL relies on hierarchies of policies. Early work in this area introduced *options*, in which, apart from *primitive actions* (single-timestep actions), policies could also run other policies (multi-timestep "actions") [116]. This approach allows top-level policies to focus on higher-level *goals*, whilst *subpolicies* are responsible for fine control. Several works in DRL have attempted HRL by the use of one top-level policy that chooses between subpolicies, where the division of states or goals in to subpolicies is achieved either manually [1, 121, 54] or automatically [2, 129, 130]. One of the ways to help with constructing subpolicies is to focus on discovering and reaching goals, which are specific states in the environment; they may often be locations, which an agent should navigate to. Whether utilised with HRL or not, the discovery and generalisation of goals is also an important area of ongoing research [99, 55, 130].

D. Imitation Learning and Inverse RL

One may ask why, if given a sequence of "optimal" actions from expert demonstrations, it is not possible to use supervised learning in a straightforward manner—a case of "learning from demonstration". This is indeed possible, and is known as *behavioural cloning* in traditional RL literature. Taking advantage of the stronger signals available in supervised learning problems, behavioural cloning enjoyed success in earlier neural network research, with the most notable success being ALVINN, one of the earliest autonomous cars [89]. However, behavioural cloning cannot adapt to new situations, and small deviations from the demonstration during the execution of the learned policy can compound and lead to scenarios where the policy is unable to recover. A more generalisable solution is to use provided trajectories to guide the learning of suitable state-action pairs, but fine-tune the agent using RL [39].

The goal of IRL is to estimate an unknown reward function from observed trajectories that characterise a desired solution [78]; IRL can be used in combination with RL to improve upon demonstrated behaviour. Using the power of deep neural networks, it is now possible to learn complex, nonlinear reward functions for IRL [140]. Ho and Ermon [41] showed that policies are uniquely characterised by their *occupancies* (visited state and action distributions) allowing IRL to be reduced to the problem of measure matching. With this insight they were able to use generative adversarial training [33] to facilitate reward function learning in a more flexible manner, resulting in the generative adversarial imitation learning (GAIL) algorithm. GAIL was later extended to allow IRL to be applied even when receiving expert trajectories from a different visual viewpoint to that of the RL agent [111]. In complementary work, Baram et al. [4] exploit gradient information that was not used in GAIL to learn models within the IRL process.

E. Multi-agent RL

Usually, RL considers a single learning agent in a stationary environment. In contrast, multi-agent RL (MARL) considers multiple agents learning through RL, and often the non-stationarity introduced by other agents changing their behaviours as they learn [14]. In DRL, the focus has been on enabling (differentiable) communication between agents, which allows them to co-operate. Several approaches have been proposed for this purpose, including passing messages to agents sequentially [28], using a bidirectional channel (providing ordering with less signal loss) [87], and an all-to-all channel [114]. The addition of communication channels is a natural strategy to apply to MARL in complex scenarios and does not preclude the usual practice of modelling co-operative or competing agents as applied elsewhere in the MARL literature [14]. Other DRL works of note in MARL investigate the effects of learning and sequential decision making in game theory [38, 60].

F. Memory and Attention

As one of the earliest works in DRL the DQN spawned many extensions. One of the first extensions was converting the DQN into an RNN, which allows the network to better deal with POMDPs by integrating information over long time periods. Like recursive filters, recurrent connections provide an efficient means of acting conditionally on temporally distant prior observations. By using recurrent connections between its hidden units, the deep recurrent Q-network (DRQN) introduced by Hausknecht and Stone [35] was able to successfully infer the velocity of the ball in the game "Pong," even when frames of the game were randomly blanked out. Further improvements were gained by introducing *attention*—a technique where additional connections are added from the recurrent units to lower layers—to the DRQN, resulting in the deep attention recurrent Q-network (DARQN) [110]. Attention gives a network the ability to choose which part of its next input to focus on, and allowed the DARQN to beat both the DQN and DRQN on games, which require longer-term planning. However, the DQN outperformed the DRQN and DARQN on games requiring quick reactions, where Q-values can fluctuate more rapidly.

Taking recurrent processing further, it is possible to add a differentiable memory to the DQN, which allows it to more flexibly process information in its "working memory" [82]. In traditional RNNs, recurrent units are responsible for both performing calculations and storing information. Differentiable memories add large matrices that are purely used for storing information, and can be accessed using differentiable read and write operations, analagously to computer memory. With their key-value-based memory Q-network (MQN), Oh et al. [82] constructed an agent that could solve a simple maze built in Minecraft, where the correct goal in each episode was indicated by a coloured block shown near the start of the maze. The MQN, and especially its more sophisticated variants, significantly outperformed both DQN and DRQN baselines, highlighting the importance of using decoupled memory storage. More recent work, where the memory was given a 2D structure in order to resemble a spatial map, hints at future research where more specialised memory structures will be developed to address specific problems, such as 2D or 3D navigation [84]. Alternatively, differentiable memories can be used as approximate hash tables, allowing DRL algorithms to store and retrieve successful experiences to facilitate rapid learning [90].

Note that RNNs are not restricted to value-function-based methods but have also been successfully applied to policy search [138] and actor-critic methods [36, 72].

G. Transfer Learning

Even though DRL algorithms can process high-dimensional inputs, it is rarely feasible to train RL agents directly on visual inputs in the real world, due to the large number of samples required. To speed up learning in DRL, it is possible to exploit previously acquired knowledge from related tasks, which comes in several guises: transfer learning, multitask learning [16] and curriculum learning [10] to name a few. There is much interest in transferring learning from one task to another, particularly from training in physics simulators with visual renderers and fine-tuning the models in the real world. This can be achieved in a naive fashion, directly using the same network in both the simulated and real phases [142], or with more sophisticated training procedures that directly try to mitigate the problem of neural networks "catastrophically forgetting" old knowledge by adding extra layers when transferring domain [96, 97]. Other approaches involve directly learning an alignment between simulated and real visuals [124], or even between two different camera viewpoints [111].

A different form of transfer can be utilised to help RL in the form of multitask training [66, 43, 69]. Especially with neural networks, supervised and unsupervised learning tasks can help train features that can be used by RL agents, making optimising the RL objective easier to achieve. For example, the "unsupervised reinforcement and auxiliary learning" A3C-based agent is additionally trained with "pixel control" (maximally changing pixel inputs), plus reward prediction and value function learning from experience replay [43]. Meanwhile, the A3C-based agent of Mirowski et al. [69] was additionally trained to construct a depth map given RGB inputs, which helps it in its task of learning to navigate a 3D environment. In an ablation study, Mirowski et al. [69] showed the predicting depth was more useful than receiving depth as an extra input, lending further support to the idea that gradients induced by auxiliary tasks can be extremely effective at boosting DRL.

Transfer learning can also be used to construct more parameter-efficient policies. In the student-teacher paradigm in machine learning, one can first train a more powerful "teacher" model, and then use it to guide the training of a less powerful "student" model. Whilst originally applied to supervised learning, the neural network knowledge transfer technique known as *distillation* [40] has been utilised to both transfer policies learned by large DQNs to smaller DQNs, and transfer policies learned by several DQNs trained on separate games to one single DQN [85, 95]. This is an important step if we wish to construct agents that can accomplish a wide range of tasks since training directly on multiple RL objectives at once may be infeasible.

H. Benchmarks

One of the challenges in any field in machine learning is a standardised way of evaluating new techniques. Although much early work focused on simple, custom MDPs, there shortly emerged control problems that could be used as standard benchmarks for testing new algorithms, such as the Cartpole [5], Acrobot [22] and Mountain Car [74] domains.

However, these problems were limited to relatively small state spaces, and therefore failed to capture the complexities that would be encountered in most realistic scenarios. Arguably the initial driver of DRL, the ALE provided an interface to Atari 2600 video games, with code to access over 50 games provided with the initial release [8]. As video games can vary greatly, but still present interesting and challenging objectives for humans, they provide an excellent testbed for RL agents. As the first algorithm to successfully play a range of these games directly from their visuals, the DQN [71] has secured its place as a milestone in the development of RL algorithms. This success story has started a trend of using video games as standardised RL testbeds, with several interesting options now available .ViZDoom provides an interface to the Doom first-person shooter [49], and echoing the popularity of e-sports competitions, ViZDoom competitions are now held at the yearly IEEE Conference on Computational Intelligence in Games. Facebook's TorchCraft provides an interface to the StarCraft real-time strategy game, presenting challenges in both micromanagement and long-term planning [117]. In an aim to provide more flexible environments, DeepMind Lab was developed on top of the Quake III Arena first-person shooter engine [6], and Microsoft's Project Malmo exposed an interface to the Minecraft sandbox game [44]. Both environments provide customisable platforms for RL agents in 3D environments.

Most DRL approaches focus on discrete actions, but some solutions have also been developed for continuous control problems. Many DRL papers in continuous control [103, 37, 67, 72, 4, 111] have used the MuJoCo physics engine to obtain relatively realistic dynamics for multi-joint continuous control problems [122], and there has now been some effort to standardise these problems [24].

To help with standardisation and reproducibility, most of the aforementioned RL domains and more have been made available in the OpenAI Gym, a library and online service that allows people to easily interface with and publicly share the results of RL algorithms on these domains [13].

VII. Conclusion: Beyond Pattern Recognition

Despite the successes of DRL, many problems need to be addressed before these techniques can be applied to a wide range of complex real-world problems [57]. Recent work with (non-deep) generative causal models demonstrated superior generalisation over standard DRL algorithms [72, 96] in some benchmarks [8], achieved by reasoning about causes and effects in the environment [47]. For example, the schema networks of Kanksy et al. [47] trained on the game "Breakout" immediately adapted to a variant where a small wall was placed in front of the target blocks, whilst progressive (A3C) networks [96] failed to match the performance of the schema networks even after training on the new domain. Although DRL has already been combined with AI techniques, such as search [108] and planning [118], a deeper integration with other traditional AI approaches promises benefits such as better sample complexity, generalisation and interpretability [30]. In time, we also hope that our theoretical understanding of the properties of neural networks (particularly within DRL) will improve, as it currently lags far behind practice.

To conclude, it is worth revisiting the overarching goal of all of this research: the creation of general-purpose AI systems that can interact with and learn from the world around them. Interaction with the environment is simultaneously the advantage and disadvantage of RL. Whilst there are many challenges in seeking to understand our complex and ever-changing world, RL allows us to choose how we explore it. In effect, RL endows agents with the ability to perform experiments to better understand their surroundings, enabling them to learn even high-level causal relationships. The availability of high-quality visual renderers and physics engines now enables us to take steps in this direction, with works that try to learn intuitive models of physics in visual environments [23]. Challenges remain before this will be possible in the real world, but steady progress is being made in agents that learn the fundamental principles of the world through observation and action. Perhaps, then, we are not too far away from AI systems that learn and act in more human-like ways in increasingly complex environments.

References

[1] Kai Arulkumaran, Nat Dilokthanakul, Murray Shanahan, and Anil Anthony Bharath. Classifying Options for Deep Reinforcement Learning. In *IJCAI Workshop on Deep Reinforcement Learning: Frontiers and Challenges*, 2016.

[2] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The Option-critic Architecture. In *AAAI*, 2017.

[3] Leemon C Baird III. Advantage Updating. Technical report, DTIC Document, 1993.

[4] Nir Baram, Oron Anschel, and Shie Mannor. Model-based Adversarial Imitation Learning. In *NIPS Workshop on Deep Reinforcement Learning*, 2016.

[5] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems. *IEEE Trans. on Systems, Man, and Cybernetics*, (5):834–846, 1983.

[6] Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. DeepMind Lab. *arXiv:1612.03801*, 2016.

[7] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying Count-based Exploration and Intrinsic Motivation. In *NIPS*, 2016.

[8] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. In *IJCAI*, 2015.

[9] Richard Bellman. On the Theory of Dynamic Programming. *PNAS*, 38(8):716–719, 1952.

[10] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum Learning. In *ICML*, 2009.

[11] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation Learning: A Review and New Perspectives. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[12] Dimitri P Bertsekas. Dynamic Programming and Suboptimal Control: A Survey from ADP to MPC. *European Journal of Control*, 11(4-5):310–334, 2005.

[13] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv:1606.01540*, 2016.

[14] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A Comprehensive survey of Multiagent Reinforcement Learning. *IEEE Trans. on Systems, Man, And Cybernetics*, 2008.

[15] Murray Campbell, A Joseph Hoane, and Feng-hsiung Hsu. Deep Blue. *Artificial Intelligence*, 134(1-2):57–83, 2002.

[16] Rich Caruana. Multitask Learning. *Machine Learning*, 28(1):41–75, 1997.

[17] Silvia Chiappa, Sébastien Racaniere, Daan Wierstra, and Shakir Mohamed. Recurrent Environment Simulators. In *ICLR*, 2017.

[18] Paul Christiano, Zain Shah, Igor Mordatch, Jonas Schneider, Trevor Blackwell, Joshua Tobin, Pieter Abbeel, and Wojciech Zaremba. Transfer from Simulation to Real World through Learning Deep Inverse Dynamics Model. *arXiv:1610.03518*, 2016.

[19] Giuseppe Cuccu, Matthew Luciw, Jürgen Schmidhuber, and Faustino Gomez. Intrinsically Motivated Neuroevolution for Vision-based Reinforcement Learning. In *ICDL*, volume 2, 2011.

[20] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large Scale Distributed Deep Networks. In *NIPS*, 2012.

[21] Marc P Deisenroth, Gerhard Neumann, and Jan Peters. A Survey on Policy Search for Robotics. *Foundations and Trends® in Robotics*, 2(1–2), 2013.

[22] Gerald DeJong and Mark W Spong. Swinging Up the Acrobot: An Example of Intelligent Control. In *ACC*, 1994.

[23] Misha Denil, Pulkit Agrawal, Tejas D Kulkarni, Tom Erez, Peter Battaglia, and Nando de Freitas. Learning to Perform Physics Experiments via Deep Reinforcement Learning. In *ICLR*, 2017.

[24] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking Deep Reinforcement Learning for Continuous Control. In *ICML*, 2016.

[25] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning. *arXiv:1611.02779*, 2016.

[26] David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, et al. Building Watson: An Overview of the DeepQA Project. *AI Magazine*, 31 (3):59–79, 2010.

[27] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep Spatial Autoencoders for Visuomotor Learning. In *ICRA*, 2016.

[28] Jakob N Foerster, Yannis M Assael, Nando de Freitas, and Shimon Whiteson. Learning to Communicate to Solve Riddles with Deep Distributed Recurrent Q-Networks. *arXiv:1602.02672*, 2016.

[29] Michael C Fu. Gradient Estimation. *Handbooks in Operations Research and Management Science*, 13:575–616, 2006.

[30] Marta Garnelo, Kai Arulkumaran, and Murray Shanahan. Towards Deep Symbolic Reinforcement Learning. In *NIPS Workshop on Deep Reinforcement Learning*, 2016.

[31] Peter W Glynn. Likelihood Ratio Gradient Estimation for Stochastic Systems. *Communications of the ACM*, 33(10):75–84, 1990.
[32] Faustino Gomez and Jürgen Schmidhuber. Evolving Modular Fast-weight Networks for Control. In *ICANN*, 2005.
[33] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *NIPS*, 2014.
[34] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous Deep Q-learning with Model-based Acceleration. In *ICLR*, 2016.
[35] Matthew Hausknecht and Peter Stone. Deep Recurrent Q-learning for Partially Observable MDPs. In *Association for the Advancement of Artificial Intelligence Fall Symposium Series*, 2015.
[36] Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. Memory-based Control with Recurrent Neural Networks. *arXiv:1512.04455*, 2015.
[37] Nicolas Heess, Gregory Wayne, David Silver, Tim Lillicrap, Tom Erez, and Yuval Tassa. Learning Continuous Control Policies by Stochastic Value Gradients. In *NIPS*, 2015.
[38] Johannes Heinrich and David Silver. Deep Reinforcement Learning from Self-play in Imperfect-information Games. *arXiv:1603.01121*, 2016.
[39] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, et al. Learning from Demonstrations for Real World Reinforcement Learning. *arXiv:1704.03732*, 2017.
[40] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. *arXiv:1503.02531*, 2015.
[41] Jonathan Ho and Stefano Ermon. Generative Adversarial Imitation Learning. In *NIPS*, 2016.
[42] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. VIME: Variational Information Maximizing Exploration. In *NIPS*, 2016.
[43] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement Learning with Unsupervised Auxiliary Tasks. In *ICLR*, 2017.
[44] Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The Malmo Platform for Artificial Intelligence Experimentation. In *IJCAI*, 2016.
[45] Leslie P Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, 101 (1):99–134, 1998.
[46] Sham M Kakade. A Natural Policy Gradient. In *NIPS*, 2002.
[47] Ken Kansky, Tom Silver, David A Mély, Mohamed Eldawy, Miguel Lázaro-Gredilla, Xinghua Lou, Nimrod Dorfman, Szymon Sidor, Scott Phoenix, and Dileep George. Schema Networks: Zero-shot Transfer with a Generative Causal Model of Intuitive Physics. In *ICML*, 2017.
[48] Hilbert J Kappen. Path Integrals and Symmetry Breaking for Optimal Control Theory. *Journal of Statistical Mechanics: Theory and Experiment*, 2005(11): P11011, 2005.
[49] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning. *arXiv:1605.02097*, 2016.
[50] Diederik P Kingma and Max Welling. Auto-encoding Variational Bayes. *arXiv:1312.6114*, 2013.
[51] Nate Kohl and Peter Stone. Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion. In *ICRA*, volume 3, 2004.
[52] Vijay R Konda and John N Tsitsiklis. On Actor-critic Algorithms. *SIAM Journal on Control and Optimization*, 42(4):1143–1166, 2003.
[53] Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez. Evolving Large-scale Neural Networks for Vision-based Reinforcement Learning. In *GECCO*, 2013.
[54] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. In *NIPS*, 2016.
[55] Tejas D Kulkarni, Ardavan Saeedi, Simanta Gautam, and Samuel J Gershman. Deep Successor Reinforcement Learning. *arXiv:1606.02396*, 2016.
[56] Tze Leung Lai and Herbert Robbins. Asymptotically Efficient Adaptive Allocation Rules. *Advances in Applied Mathematics*, 6(1):4–22, 1985.
[57] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building Machines That Learn and Think Like People. *The Behavioral and Brain Sciences*, page 1, 2016.
[58] Sascha Lange, Martin Riedmiller, and Arne Voigtlander. Autonomous Reinforcement Learning on Raw Visual Input Data in a Real World Application. In *IJCNN*, 2012.
[59] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep Learning. *Nature*, 521 (7553):436–444, 2015.
[60] Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent Reinforcement Learning in Sequential Social Dilemmas. In *AAMAS*, 2017.
[61] Sergey Levine and Pieter Abbeel. Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics. In *NIPS*, 2014.
[62] Sergey Levine and Vladlen Koltun. Guided Policy Search. In *ICLR*, 2013.
[63] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end Training of Deep Visuomotor Policies. *JMLR*, 17(39):1–40, 2016.
[64] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning Hand-eye Coordination for Robotic Grasping with Deep Learning and Large-scale Data Collection. In *ISER*, 2016.
[65] Ke Li and Jitendra Malik. Learning to Optimize. *arXiv:1606.01885*, 2016.
[66] Xiujun Li, Lihong Li, Jianfeng Gao, Xiaodong He, Jianshu Chen, Li Deng, and Ji He. Recurrent Reinforcement Learning: A Hybrid Approach. *arXiv:1509.03044*, 2015.
[67] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous Control with Deep Reinforcement Learning. In *ICLR*, 2016.
[68] Long-Ji Lin. Self-improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. *Machine Learning*, 8(3–4):293–321, 1992.
[69] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andy Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to Navigate in Complex Environments. In *ICLR*, 2017.
[70] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. Recurrent Models of Visual Attention. In *NIPS*, 2014.
[71] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level Control through Deep Reinforcement Learning. *Nature*, 518(7540):529–533, 2015.
[72] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. In *ICLR*, 2016.
[73] Shakir Mohamed and Danilo Jimenez Rezende. Variational Information Maximisation for Intrinsically Motivated Reinforcement Learning. In *NIPS*, 2015.
[74] Andrew William Moore. Efficient Memory-based Learning for Robot Control. Technical report, University of Cambridge, Computer Laboratory, 1990.
[75] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G Bellemare. Safe and Efficient Off-policy Reinforcement Learning. In *NIPS*, 2016.
[76] Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the Gap Between Value and Policy Based Reinforcement Learning. *arXiv:1702.08892*, 2017.
[77] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. Massively Parallel Methods for Deep Reinforcement Learning. *arXiv:1507.04296*, 2015.
[78] Andrew Y Ng and Stuart J Russell. Algorithms for Inverse Reinforcement Learning. In *ICML*, 2000.
[79] Andrew Y Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous Inverted Helicopter Flight via Reinforcement Learning. *Experimental Robotics IX*, pages 363–372, 2006.
[80] Brendan O'Donoghue, Remi Munos, Koray Kavukcuoglu, and Volodymyr Mnih. PGQ: Combining Policy Gradient and Q-learning. In *ICLR*, 2017.
[81] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional Video Prediction using Deep Networks in Atari Games. In *NIPS*, 2015.
[82] Junhyuk Oh, Valliappa Chockalingam, Satinder Singh, and Honglak Lee. Control of Memory, Active Perception, and Action in Minecraft. In *ICLR*, 2016.
[83] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep Exploration via Bootstrapped DQN. In *NIPS*, 2016.
[84] Emilio Parisotto and Ruslan Salakhutdinov. Neural Map: Structured Memory for Deep Reinforcement Learning. *arXiv:1702.08360*, 2017.
[85] Emilio Parisotto, Jimmy L Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep Multitask and Transfer Reinforcement Learning. *arXiv:1511.06342*, 2015.
[86] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven Exploration by Self-supervised Prediction. In *ICML*, 2017.
[87] Peng Peng, Quan Yuan, Ying Wen, Yaodong Yang, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent Bidirectionally-coordinated Nets for Learning to Play StarCraft Combat Games. *arXiv:1703.10069*, 2017.
[88] Jan Peters, Katharina Mülling, and Yasemin Altun. Relative Entropy Policy Search. In *AAAI*, 2010.
[89] Dean A Pomerleau. ALVINN, an Autonomous Land Vehicle in a Neural Network. Technical report, Carnegie Mellon University, Computer Science Department, 1989.
[90] Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adrià Puigdomènech, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural Episodic Control. *arXiv:1703.01988*, 2017.
[91] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A Lock-free Approach to Parallelizing Stochastic Gradient Descent. In *NIPS*, 2011.
[92] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *ICML*, 2014.
[93] Martin Riedmiller. Neural Fitted Q Iteration—First Experiences with a Data Efficient Neural Reinforcement Learning Method. In *ECML*, 2005.
[94] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using Connectionist Systems*. University of Cambridge, Department of Engineering, 1994.
[95] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy Distillation. *arXiv:1511.06295*, 2015.
[96] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive Neural Networks. *arXiv:1606.04671*, 2016.
[97] Andrei A Rusu, Matej Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real Robot Learning from Pixels with Progressive Nets. *arXiv:1610.04286*, 2016.
[98] Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *arXiv:1703.03864*, 2017.
[99] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal Value Function Approximators. In *ICML*, 2015.
[100] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized Experience Replay. *arXiv:1511.05952*, 2015.

[101] Jürgen Schmidhuber. A Possibility for Implementing Curiosity and Boredom in Model-building Neural Controllers. In *SAB*, 1991.
[102] Jürgen Schmidhuber and Rudolf Huber. Learning to Generate Artificial Fovea Trajectories for Target Detection. *Intl. Journal of Neural Systems*, 2(01n02):125–134, 1991.
[103] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust Region Policy Optimization. In *ICML*, 2015.
[104] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional Continuous Control using Generalized Advantage Estimation. In *ICLR*, 2016.
[105] John Schulman, Pieter Abbeel, and Xi Chen. Equivalence Between Policy Gradients and Soft Q-Learning. *arXiv:1704.06440*, 2017.
[106] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando de Freitas. Taking the Human out of the Loop: A Review of Bayesian Optimization. *Proc. of the IEEE*, 104(1):148–175, 2016.
[107] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic Policy Gradient Algorithms. In *ICML*, 2014.
[108] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529(7587):484–489, 2016.
[109] Satinder Singh, Diane Litman, Michael Kearns, and Marilyn Walker. Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System. *JAIR*, 16:105–133, 2002.
[110] Ivan Sorokin, Alexey Seleznev, Mikhail Pavlov, Aleksandr Fedorov, and Anastasiia Ignateva. Deep Attention Recurrent Q-network. In *NIPS Workshop on Deep Reinforcement Learning*, 2015.
[111] Bradley C Stadie, Pieter Abbeel, and Ilya Sutskever. Third Person Imitation Learning. In *ICLR*, 2017.
[112] Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing Exploration in Reinforcement Learning with Deep Predictive Models. In *NIPS Workshop on Deep Reinforcement Learning*, 2015.
[113] Alexander L Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L Littman. PAC Model-free Reinforcement Learning. In *ICML*, 2006.
[114] Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning Multiagent Communication with Backpropagation. In *NIPS*, 2016.
[115] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
[116] Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112(1–2):181–211, 1999.
[117] Gabriel Synnaeve, Nantas Nardelli, Alex Auvolat, Soumith Chintala, Timothée Lacroix, Zeming Lin, Florian Richoux, and Nicolas Usunier. TorchCraft: A Library for Machine Learning Research on Real-Time Strategy Games. *arXiv:1611.00625*, 2016.
[118] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value Iteration Networks. In *NIPS*, 2016.
[119] Gerald Tesauro. Temporal Difference Learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.
[120] Gerald Tesauro, Rajarshi Das, Hoi Chan, Jeffrey Kephart, David Levine, Freeman Rawson, and Charles Lefurgy. Managing Power Consumption and Performance of Computing Systems using Reinforcement Learning. In *NIPS*, 2008.
[121] Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J Mankowitz, and Shie Mannor. A Deep Hierarchical Approach to Lifelong Learning in Minecraft. In *AAAI*, 2017.
[122] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A Physics Engine for Model-based Control. In *IROS*, 2012.
[123] John N Tsitsiklis and Benjamin Van Roy. Analysis of Temporal-difference Learning with Function Approximation. In *NIPS*, 1997.
[124] Eric Tzeng, Coline Devin, Judy Hoffman, Chelsea Finn, Xingchao Peng, Sergey Levine, Kate Saenko, and Trevor Darrell. Towards Adapting Deep Visuomotor Representations from Simulated to Real Environments. In *WAFR*, 2016.
[125] Nicolas Usunier, Gabriel Synnaeve, Zeming Lin, and Soumith Chintala. Episodic Exploration for Deep Deterministic Policies: An Application to StarCraft Micromanagement Tasks. In *ICLR*, 2017.
[126] Hado van Hasselt. Double Q-learning. In *NIPS*, 2010.
[127] Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-Learning. In *AAAI*, 2016.
[128] Harm Vanseijen and Rich Sutton. A Deeper Look at Planning as Learning from Replay. In *ICML*, 2015.
[129] Alexander Vezhnevets, Volodymyr Mnih, Simon Osindero, Alex Graves, Oriol Vinyals, John Agapiou, and Koray Kavukcuoglu. Strategic Attentive Writer for Learning Macro-actions. In *NIPS*, 2016.
[130] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal Networks for Hierarchical Reinforcement Learning. *arXiv:1703.01161*, 2017.
[131] Niklas Wahlström, Thomas B Schön, and Marc P Deisenroth. Learning Deep Dynamical Models from Image Pixels. *IFAC SYSID*, 48(28), 2015.
[132] Niklas Wahlström, Thomas B Schön, and Marc P Deisenroth. From Pixels to Torques: Policy Learning with Deep Dynamical Models. *arXiv:1502.02251*, 2015.
[133] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to Reinforcement Learn. *arXiv:1611.05763*, 2016.
[134] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling Network Architectures for Deep Reinforcement Learning. In *ICLR*, 2016.
[135] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample Efficient Actor-Critic with Experience Replay. In *ICLR*, 2017.
[136] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8 (3-4):279–292, 1992.
[137] Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images. In *NIPS*, 2015.
[138] Daan Wierstra, Alexander Förster, Jan Peters, and Jürgen Schmidhuber. Recurrent Policy Gradients. *Logic Journal of the IGPL*, 18(5):620–634, 2010.
[139] Ronald J Williams. Simple Statistical Gradient-following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8(3-4):229–256, 1992.
[140] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Deep Inverse Reinforcement Learning. *CoRR*, 2015.
[141] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C Courville, Ruslan Salakhutdinov, Richard S Zemel, and Yoshua Bengio. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In *ICML*, volume 14, 2015.
[142] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning. In *ICRA*, 2017.
[143] Barret Zoph and Quoc V Le. Neural Architecture Search with Reinforcement Learning. In *ICLR*, 2017.

Kai Arulkumaran (ka709@imperial.ac.uk) is a Ph.D. candidate in the Department of Bioengineering at Imperial College London. He received a B.A. in Computer Science at the University of Cambridge in 2012, and an M.Sc. in Biomedical Engineering at Imperial College London in 2014. He was a Research Intern in both Twitter Magic Pony and Microsoft Research in 2017. His research focus is deep reinforcement learning and transfer learning for visuomotor control.

Marc Peter Deisenroth (m.deisenroth@imperial.ac.uk) is a Lecturer in Statistical Machine Learning in the Department of Computing at Imperial College London and with PROWLER.io. He has been awarded an Imperial College Research Fellowship in 2014 and received Best Paper Awards at ICRA 2014 and ICCAS 2016. He is a recipient of a Google Faculty Research Award and a Microsoft Ph.D. Scholarship. His research is centred around data-efficient machine learning for autonomous decision making.

Miles Brundage (miles.brundage@philosophy.ox.ac.uk) is a Ph.D. candidate in Human and Social Dimensions of Science and Technology at Arizona State University, and a Research Fellow at the University of Oxford's Future of Humanity Institute. He received a B.A. in Political Science at George Washington University in 2010. His research focuses on governance issues related to artificial intelligence.

Anil Anthony Bharath (a.bharath@imperial.ac.uk) is a Reader in the Department of Bioengineering at Imperial College London. He was an academic visitor in the Signal Processing Group at the University of Cambridge in 2006. He holds a B.Eng. from University College London (EEE) in 1988, and a Ph.D. from Imperial College London (Signal Processing) in 1993. He is a co-founder of Cortexica Vision Systems. His research interests are in deep architectures for visual inference.