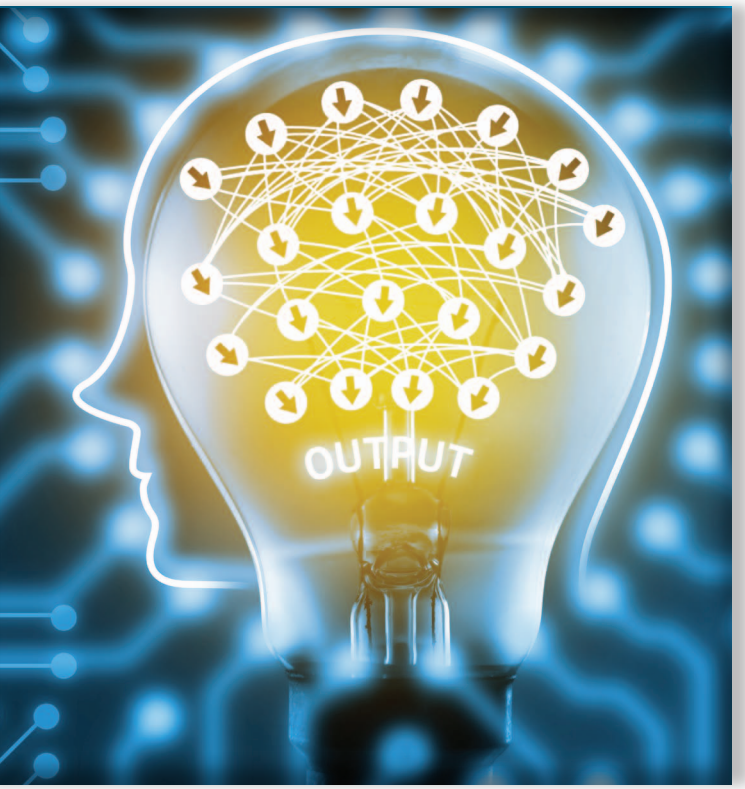
Kai Arulkumaran, Marc Peter Deisenroth,
Miles Brundage, and Anil Anthony Bharath

Deep Reinforcement Learning

A brief survey



Digital Object Identifier 10.1109/MSP.2017.2743240
Date of publication: 13 November 2017

Deep reinforcement learning (DRL) is poised to revolutionize the field of artificial intelligence (AI) and represents a step toward building autonomous systems with a higher-level understanding of the visual world. Currently, deep learning is enabling reinforcement learning (RL) to scale to problems that were previously intractable, such as learning to play video games directly from pixels. DRL 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 RL, then progress to the main streams of value-based and policy-based methods. Our survey will cover central algorithms in deep RL, including the deep Q-network (DQN), trust region policy optimization (TRPO), and asynchronous advantage actor critic. In parallel, we highlight the unique advantages of deep neural networks, focusing on visual understanding via RL. To conclude, we describe several current areas of research within the field.

Introduction

One of the primary goals of the field of AI is to produce fully autonomous agents that interact with their environments to learn optimal behaviors, 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 RL [78]. Although RL had some successes in the past [31], [53], [74], [81], previous approaches lacked scalability 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 [76]. 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 [39]. 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 [7]. Deep learning has similarly accelerated progress in RL, with the use of deep-learning algorithms within RL defining the field of 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 toward developing autonomous agents. For a more comprehensive survey of recent efforts in DRL, we refer readers to the overview by Li [43].

Deep learning enables RL to scale to decision-making problems that were previously intractable, i.e., settings with high-dimensional state and action spaces. Among 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 [47]. 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 [73], paralleling the historic achievement of IBM's Deep Blue in chess two decades earlier [9]. Unlike the handcrafted rules that have dominated chess-playing systems, AlphaGo comprised neural networks that were trained using supervised learning and RL, 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 [41], [42], succeeding controllers that used to be hand-engineered or learned from low-dimensional features of the robot's state. In Figure 1, we showcase just some of the domains that DRL has

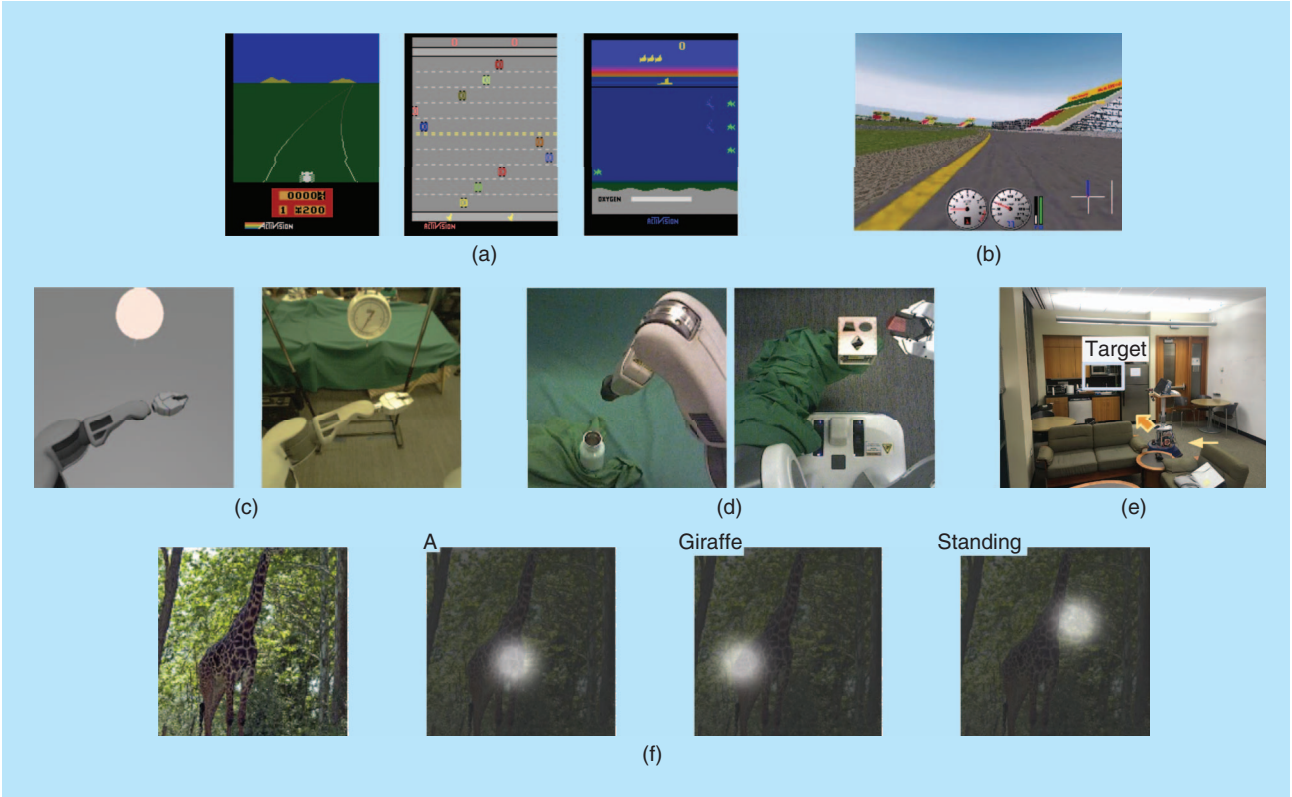


FIGURE 1. A range of visual RL domains. (a) Three classic Atari 2600 video games, *Enduro, Freeway,* and *Seaquest,* from the Arcade Learning Environment (ALE) [5]. Due to the range of supported games that vary in genre, visuals, and difficulty, the ALE has become a standard test bed for DRL algorithms [20], [47], [48], [55], [70], [75], [92]. The ALE is one of several benchmarks that are now being used to standardize evaluation in RL. (b) The TORCS car racing simulator, which has been used to test DRL algorithms that can output continuous actions [33], [44], [48] (as the games from the ALE only support discrete actions). (c) Utilizing 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 [11], [64], [84]. (d) Two of the four robotic tasks designed by Levine et al. [41]: screwing on a bottle cap and placing a shaped block in the correct hole. Levine et al. [41] 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 [100]. (f) A natural image being captioned by a neural network that uses RL to choose where to look [99]. (b)–(f) reproduced from [41], [44], [84], [99], and [100], respectively.

been applied to, ranging from playing video games [47] to indoor navigation [100].

Reward-driven behavior

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 behavior in response to rewards received. This paradigm of trial-and-error learning has its roots in behaviorist psychology and is one of the main foundations of RL [78]. The other key influence on RL is optimal control, which has lent the mathematical formalisms (most notably dynamic programming [6]) that underpin the field.

In the RL setup, an autonomous agent, controlled by a machine-learning algorithm, observes a state $\mathbf{s}_t$ from its environment at time step 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 maximizes the expected return (cumulative, discounted reward). Given a state, a policy returns an action to perform; an optimal policy is any policy that maximizes the expected return in the 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-learning loop is illustrated in Figure 2.

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 $\pi: \mathcal{S} \rightarrow p(\mathcal{A} = \mathbf{a} \,|\, \mathcal{S})$. 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^* that achieves the maximum expected return from all states:

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

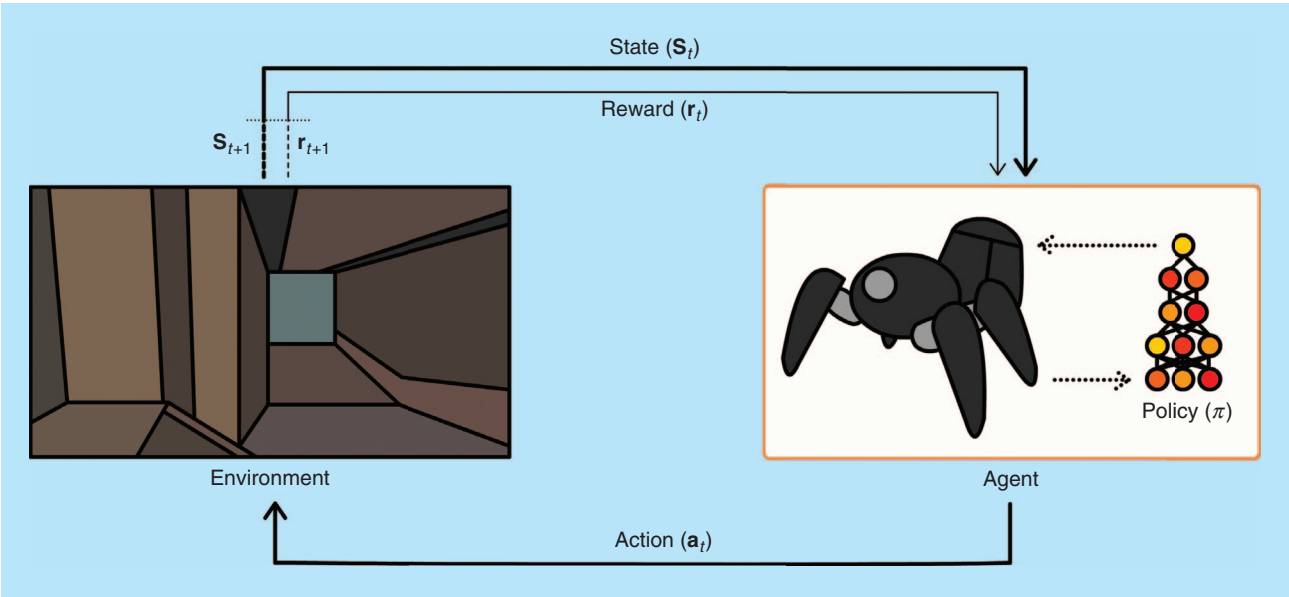


FIGURE 2. The perception-action-learning loop. At time t, the agent receives state $\mathbf{s}_t$ from the environment. The agent uses its policy to choose an action $\mathbf{a}_t$. Once the action is executed, the environment transitions a step, providing the next state, $\mathbf{s}_{t+1}$, as well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

It is also possible to consider nonepisodic 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—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 generalization 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 $p(\mathbf{o}_{t+1} \mid \mathbf{s}_{t+1}, \mathbf{a}_t)$ is dependent on the current state and the previous action [27]. 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 utilize recurrent neural networks (RNNs) [20], [21], [48], [96], which, unlike feedforward neural networks, are dynamical systems.

Challenges in RL

It is instructive to emphasize 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 materialize after many transitions of the environment. This is known as the (temporal) *credit assignment problem* [78].

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 is 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.

RL 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 that employs both value functions and policy search. Next, we will explain these approaches and other useful concepts for solving RL problems.

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 subsequently:

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

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 maximizes $\mathbb{E}_{\mathbf{s}_{t+1} \sim \mathcal{T}(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a})}[V^*(\mathbf{s}_{t+1})]$.

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 onward:

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

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 maximizing $Q^\pi(\mathbf{s}, \mathbf{a})$: $V^\pi(\mathbf{s}) = \max_\mathbf{a} Q^\pi(\mathbf{s}, \mathbf{a})$.

Dynamic programming

To actually learn Q^π, we exploit the Markov property and define the function as a Bellman equation [6], 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 [94] and the state-action-reward-state-action (SARSA) algorithm [62]:

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

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 behavioral 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 generalized 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 minimizing 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), generalized policy iteration allows for interleaving the steps, such that progress can be made more rapidly.

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 is done in the TD(λ) algorithm [78]. 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

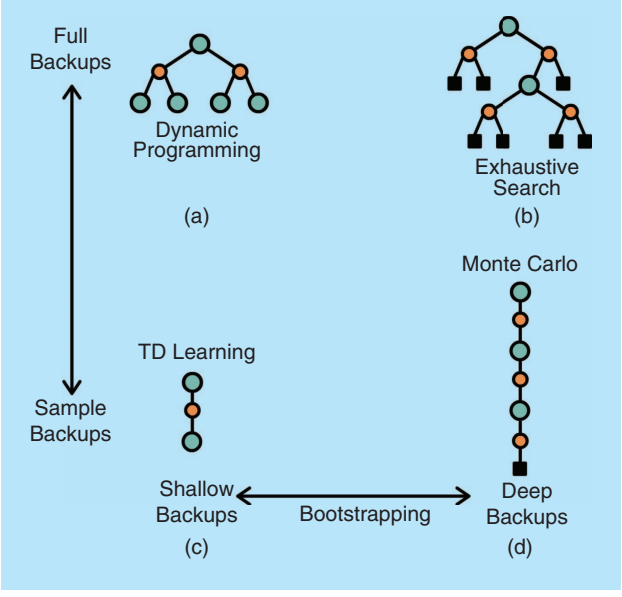


FIGURE 3. Two dimensions of RL algorithms based on the backups used to learn or construct a policy. At the extremes of these dimensions are (a) dynamic programming, (b) exhaustive search, (c) one-step TD learning, and (d) Monte Carlo approaches. Bootstrapping extends from (c) one-step TD learning to *n*-step TD learning methods [78], with (d) pure Monte Carlo approaches not relying on bootstrapping at all. Another possible dimension of variation is (c) and (d) choosing to sample actions versus (a) and (b) taking the expectation over all choices. (Figure recreated based on [78].)

an entire spectrum of RL methods based around the amount of sampling utilized.

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 $A^{\pi} = Q^{\pi} - V^{\pi}$ and is also closely related to the baseline method of variance reduction within gradient-based policy search methods [97]. The idea of advantage updates has been utilized in many recent DRL algorithms [19], [48], [71], [92].

Policy search

Policy search methods do not need to maintain a value function model but directly search for an optimal policy π^{*}. Typically, a parameterized policy π_{θ} is chosen, whose parameters are updated to maximize the expected return $\mathbb{E}[R \mid \theta]$ using either gradient-based or gradient-free optimization [12]. Neural networks that encode policies have been successfully trained using both gradient-free [17], [33] and gradient-based [22], [41], [44], [70], [71], [96], [97] methods. Gradient-free optimization can effectively cover low-dimensional parameter spaces, but, despite some successes in applying them to large networks [33], 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, while 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 [65], while more complex methods, such as compressed network search, impose additional inductive biases [33]. Perhaps the greatest advantage of gradient-free policy search is that it can also optimize nondifferentiable policies.

Policy gradients

Gradients can provide a strong learning signal as to how to improve a parameterized policy. However, to compute the expected return (1) we need to average over plausible trajectories induced by the current policy parameterization. This averaging requires either deterministic approximations (e.g., linearization) or stochastic approximations via sampling [12]. Deterministic approximations can be only 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 [97]. 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:

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

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 [97], but there are many more options available [71].

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 4. The "actor" (policy) learns by using feedback from the "critic" (value function). In doing so, these methods tradeoff variance reduction of policy gradients with bias introduction from value function methods [32], [71].

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 is that actor-critic methods utilize a learned value function. For this reason, we will later discuss actor-critic methods as a subset of policy gradient methods.

Planning and learning

Given a model of the environment, it is possible to use dynamic programming over all possible actions [Figure 3(a)], sample trajectories for heuristic search (as was done by AlphaGo [73]), or even perform an exhaustive search [Figure 3(b)]. Sutton and Barto [78] define *planning* as any method that utilizes 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 [14], [55], [75], sometimes simpler, more data-efficient methods are preferable [19]. These considerations also play a role in actor-critic methods with learned value functions [32], [71].

Searching directly for a policy represented by a neural network with very many parameters can be difficult and can suffer from severe local minima.

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 nonparametric methods [7]. 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^*.

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 1990s [81]. 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 [73], progress in RL research has favored the explicit use of value functions, which can capture the

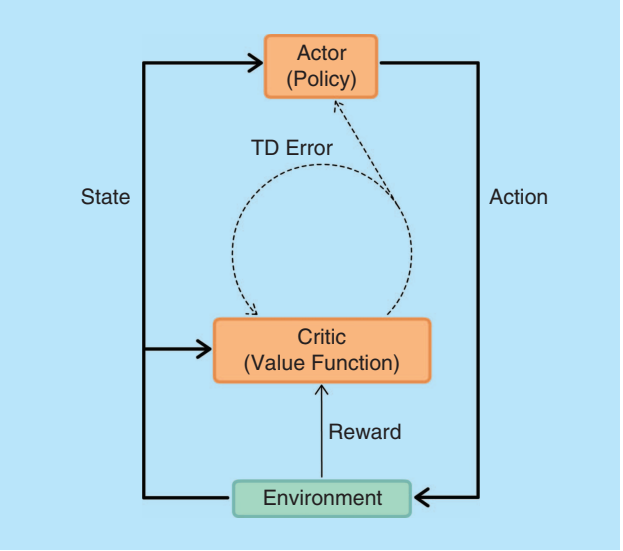


FIGURE 4. The actor-critic setup. The actor (policy) receives a state from the environment and chooses an action to perform. At the same time, the critic (value function) receives the state and reward resulting from the previous interaction. The critic uses the TD error calculated from this information to update itself and the actor. (Figure recreated based on [78].)

structure underlying the environment. From early value function methods in DRL, which took simple states as input [61], current methods are now able to tackle visually and conceptually complex environments [47], [48], [70], [100].

Function approximation and the DQN

We begin our survey of value-function-based DRL algorithms with the DQN [47], illustrated in Figure 5, which achieved scores across a wide range of classic Atari 2600 video games [5] that were comparable to that of a professional video games tester. The inputs to the DQN are four gray-scale frames of the game, concatenated over time, which are initially processed by several convolutional layers 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 (NFQ) iteration—involved training a neural network to return the Q-value given a state-action pair [61]. 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 [38]. 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 [47] is closely related to the model proposed by Lange et al. [38] 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, $\text{argmax}_\mathbf{a} Q^\pi(\mathbf{s}, \mathbf{a})$, 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 maximizing 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 behavior of CNNs in object recognition tasks, we can also inspect what parts of its view the agent considers important (see Figure 6).

The true underlying state of the game is contained within 128 bytes of Atari 2600 random-access memory. However, the DQN was designed to directly learn from visual inputs (210×160 pixel 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 [83] by the use of two techniques: experience replay [45] 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 number of interactions needed with the environment, but batches of experience can 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. While the original DQN algorithm used uniform sampling [47], later work showed that prioritizing samples based on TD errors is more effective for learning [67].

The second stabilizing method, introduced by Mnih et al. [47], 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

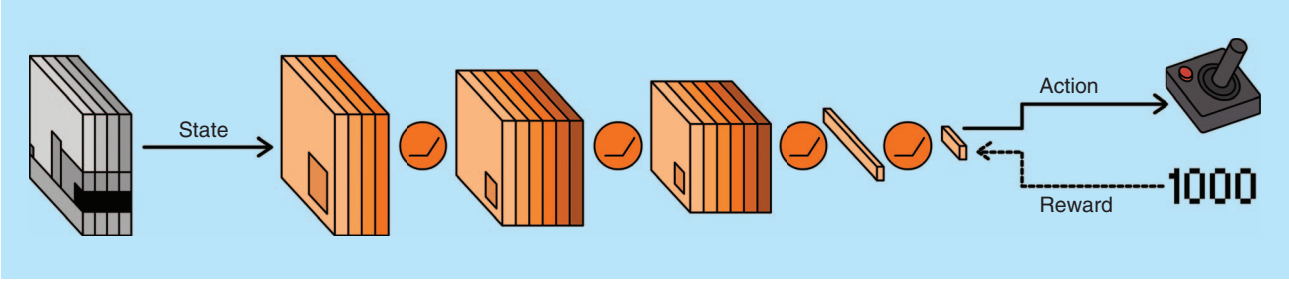


FIGURE 5. The DQN [47]. The network takes the state—a stack of gray-scale frames from the video game—and processes it with convolutional and fully connected layers, with ReLU nonlinearities in between each layer. At the final layer, the network outputs a discrete action, which corresponds to one of the 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.

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 [19], [44], [50], [93].

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. In [86], van Hasselt 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 [86]. While 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 [87].

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 [92]. Rather than having to come up with accurate Q-values for all actions, the duelling DQN [92] 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 prioritized experience replay [67] 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. [19] led them to modify the DQN with a convex advantage layer that extended the algorithm to work over sets of continuous actions, creating the normalized 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 [19].

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 favor of evolutionary algorithms [17], [33], 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 optimization methods, they can be used to optimize arbitrary, nondifferentiable models and naturally allow for more exploration in the 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 [33]. 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 [65].

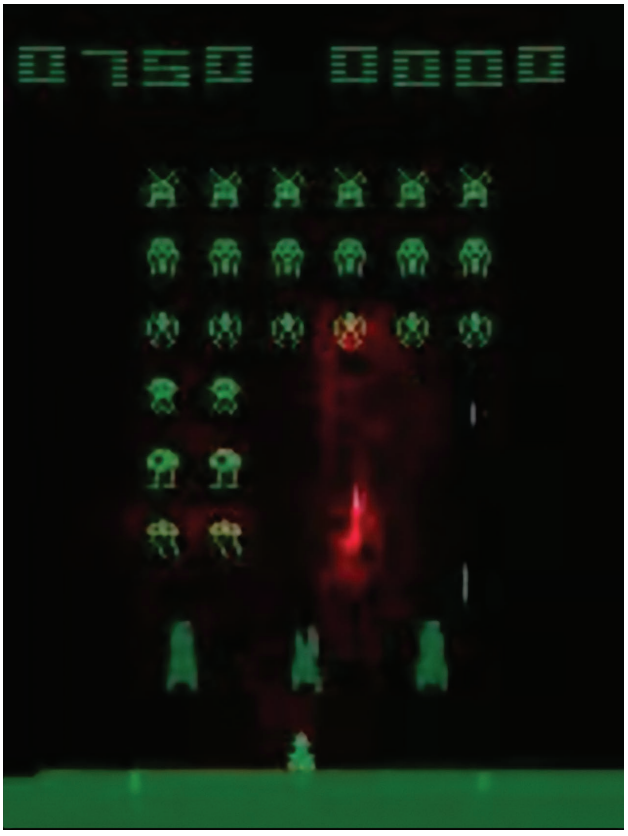


FIGURE 6. A saliency map of a trained DQN [47] playing *Space Invaders* [5]. By backpropagating the training signal to the image space, it is possible to see what a neural-network-based agent is attending to. In this frame, the most salient points—shown with the red overlay—are the laser that the agent recently fired and also the enemy that it anticipates hitting in a few time steps.

Backpropagation through stochastic functions

The workhorse of DRL, however, remains backpropagation. The previously discussed REINFORCE rule [97] allows neural networks to learn stochastic policies in a task-dependent manner, such as deciding where to look in an image to track [69] or caption [99] objects. 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 policysearch methods in recent years, having many applications outside of traditional RL domains.

Compounding errors

Searching directly for a policy represented by a neural network with very many parameters can be difficult and 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 [40]. This approach effectively biases the search toward a good (local) optimum. GPS works in a loop, by optimizing policies to match sampled trajectories and optimizing trajectory distributions to match the policy and minimize costs. Levine et al. [41] 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, provide one of the seminal works in DRL.

A more commonly used method is to use a trust region, in which optimization steps are restricted to lie within a region where the approximation of the true cost function still holds. By preventing updated policies from deviating too wildly from previous policies, the chance of a catastrophically bad update is lessened, and many algorithms that use trust regions guarantee or practically result in monotonic improvement in policy performance. The idea of constraining each policy gradient update, as measured by the Kullback–Leibler (KL) divergence between the current and proposed policy, has a long history in RL [28]. One of the newer algorithms in this line of work, TRPO, has been shown to be relatively robust and applicable to domains with high-dimensional inputs [70]. To achieve this, TRPO optimizes a surrogate objective function—specifically, it optimizes an (importance sampled) advantage estimate, constrained using a quadratic approximation of the KL divergence. While TRPO can be used as a pure policy gradient method with a simple baseline, later work by Schulman et al. [71] introduced generalized advantage estimation (GAE), which proposed several, more advanced variance reduction baselines. The combination of TRPO and GAE remains one of the state-of-the-art RL techniques in continuous control.

Actor-critic 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 full returns and/or TD errors. They can benefit from improvements in both policy gradient methods, such as GAE [71], and value function methods, such as target networks [47]. In the last few years, DRL actor-critic methods have been scaled up from learning simulated physics tasks [22], [44] to real robotic visual navigation tasks [100], directly from image pixels.

One recent development in the context of actor-critic algorithms is deterministic policy gradients (DPGs) [72], which extend the standard policy gradient theorems for stochastic policies [97] to deterministic policies. One of the major advantages of DPGs is that, while 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. [72] 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, which utilized neural networks to operate on high-dimensional, visual state spaces [44]. In the same vein as DPGs, Heess et al. [22] devised a method for calculating gradients to optimize stochastic policies by "reparameterizing" [30], [60] the stochasticity away from the network, thereby allowing standard gradients to be used (instead of the high-variance REINFORCE estimator [97]). 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)) models. Later work proceeded to integrate DPGs and SVGs with RNNs, allowing them to solve continuous control problems in POMDPs, learning directly from pixels [21]. 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. 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 be efficiently distributed over both processing cores in a single central processing unit (CPU), and across CPUs in a cluster of machines. Using a distributed system, Nair et al. [51] 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 [48], 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 stabilizes 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. [100], 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 [93], which adds off-policy bias correction to A3C, allowing it to use experience replay to improve sample complexity. Others have attempted to bridge the gap between value and policy-based RL, utilizing theoretical advancements to improve upon the original A3C [50], [54]. Finally, there is a growing trend toward exploiting auxiliary tasks to improve the representations learned by DRL agents and, hence, improve both the learning speed and final performance of these agents [26], [46].

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 in making RL data efficient and in trading off exploration and exploitation. After tackling exploration strategies, we shall then address hierarchical

RL (HRL), which imposes an inductive bias on the final policy by explicitly factorizing 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, we will look at multiagent systems, which have their own special considerations.

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 [29]) to speed up learning. Model learning plays an important role in reducing the number 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 [90], 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 [55], [91], [95]. 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 [14]. 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 [75].

Although deep neural networks can make reasonable predictions in simulated environments over hundreds of time steps [10], they typically require many samples to tune the large number of parameters they contain. Training these models often requires more samples (interaction with the environment) than simpler models. For this reason, Gu et al. [19] train locally linear models for use with the NAF algorithm—the continuous equivalent of the DQN [47]—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.

Exploration versus exploitation

One of the greatest difficulties in RL is the fundamental dilemma of exploration versus exploitation: When should the agent try out (perceived) nonoptimal actions to explore the environment (and potentially improve the model), and when should it exploit the optimal action to make useful progress? Off-policy algorithms, such as the DQN [47], 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 toward 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) to better preserve momentum [44].

The observation that temporal correlation is important led Osband et al. [56] to propose the bootstrapped DQN, which maintains several Q-value "heads" that learn different values through a combination of different weight initializations 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. [85] 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" [36]. The idea behind UCB is to pick actions that maximize $\mathbb{E}[\mathbb{R}] + \kappa\sigma[R]$, 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. While 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 [4].

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 [68]. There have been several DRL algorithms that try to implement intrinsic motivation via minimizing model prediction error [57], [75] or maximizing information gain [25], [49].

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 time-step actions), policies could also run other policies (multitime-step "actions") [79]. This approach allows top-level policies to focus on higher-level goals, while subpolicies are responsible for fine control. Several works in DRL have attempted HRL by using one top-level policy that chooses between subpolicies, where the division of states or goals in to subpolicies is achieved either manually [1], [34], [82] or automatically [2], [88], [89]. One way to help construct subpolicies is to focus on discovering and reaching goals, which are specific states in the environment; they may often be locations, to which an agent should navigate. Whether utilized with HRL or not, the discovery and generalization of goals is also an important area of ongoing research [35], [66], [89].

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 *behavioral cloning* in traditional RL literature. Taking advantage of the stronger signals available in supervised learning problems, behavioral cloning enjoyed success in earlier

neural network research, with the most notable success being ALVINN, one of the earliest autonomous cars [59]. However, behavioral 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 generalizable solution is to use provided trajectories to guide the learning of suitable state-action pairs but fine-tune the agent using RL [23].

The goal of IRL is to estimate an unknown reward function from observed trajectories that characterize a desired solution [52]; IRL can be used in combination with RL to improve upon demonstrated behavior. Using the power of deep neural networks, it is now possible to learn complex, nonlinear reward functions for IRL [98]. Ho and Ermon [24] showed that policies are uniquely characterized 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 [18] to facilitate reward-function learning in a more flexible manner, resulting in the generative adversarial imitation learning algorithm.

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.

Multiagent RL

Usually, RL considers a single learning agent in a stationary environment. In contrast, multiagent RL (MARL) considers multiple agents learning through RL and often the nonstationarity introduced by other agents changing their behaviors as they learn [8]. In DRL, the focus has been on enabling (differentiable) communication between agents, which allows them to cooperate. Several approaches have been proposed for this purpose, including passing messages to agents sequentially [15], using a bidirectional channel (providing ordering with less signal loss) [58], and an all-to-all channel [77]. The addition of communication channels is a natural strategy to apply to MARL in complex scenarios and does not preclude the usual practice of modeling cooperative or competing agents as applied elsewhere in the MARL literature [8].

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 [37]. Recent work with (nondeep) generative causal models demonstrated superior generalization over standard DRL algorithms [48], [63] in some benchmarks [5], achieved by reasoning about causes and effects in the environment [29]. For example, the schema networks of Kanksy et al. [29] trained on the game *Breakout* immediately adapted to a variant where a small wall was placed in front of the target blocks, while progressive (A3C) networks [63] 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 [73] and planning [80], a deeper integration with other traditional AI approaches promises benefits such as better sample complexity, generalization, and interpretability [16]. 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. While 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 [13]. 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.

Acknowledgments

Kai Arulkumaran would like to acknowledge Ph.D. funding from the Department of Bioengineering at Imperial College London. This research has been partially funded by a Google Faculty Research Award to Marc Deisenroth.

Authors

Kai Arulkumaran (ka709@imperial.ac.uk) received a B.A. degree in computer science at the University of Cambridge in 2012 and an M.Sc. degree in biomedical engineering from Imperial College London in 2014, where he is currently a Ph.D. degree candidate in the Department of Bioengineering. He was a research intern at Twitter's 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) received an M.Eng. degree in computer science at the University of Karlsruhe in 2006 and a Ph.D. degree in machine learning at the Karlsruhe Institute of Technology in 2009. He is a lecturer of statistical machine learning in the Department of Computing at Imperial College London and with PROWLER.io. He was awarded an Imperial College Research Fellowship in 2014 and received Best Paper Awards at the International Conference on Robotics and Automation 2014 and the International Conference on Control, Automation, and Systems 2016. He is a recipient of a Google Faculty Research Award and a Microsoft Ph.D. Scholarship. His research is centered around data-efficient machine learning for autonomous decision making.

Miles Brundage (miles.brundage@philosophy.ox.ac.uk) received a B.A. degree in political science at George

Washington University, Washington, D.C., in 2010. He is a Ph.D. degree candidate in the Human and Social Dimensions of Science and Technology Department at Arizona State University and a research fellow at the University of Oxford's Future of Humanity Institute. His research focuses on governance issues related to artificial intelligence.

Anil Anthony Bharath (a.bharath@ic.ac.uk) received his B. Eng. degree in electronic and electrical engineering from University College London in 1988 and his Ph.D. degree in signal processing from Imperial College London in 1993, where he is currently a reader in the Department of Bioengineering. He is also a fellow of the Institution of Engineering and Technology and a cofounder of Cortexica Vision Systems. He was previously an academic visitor in the Signal Processing Group at the University of Cambridge in 2006. His research interests are in deep architectures for visual inference.

References

[1] K. Arulkumaran, N. Dilokthanakul, M. Shanahan, and A. A. Bharath, "Classifying options for deep reinforcement learning," in *Proc. IJCAI Workshop Deep Reinforcement Learning: Frontiers and Challenges,* 2016.

[2] P. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Proc. Association Advancement Artificial Intelligence*, 2017, pp. 1726–1734.

[3] L. C. Baird III, "Advantage updating," Defense Tech. Inform. Center, Tech. Report D-A280 862, Fort Belvoir, VA, 1993.

[4] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *Proc. Neural Information Processing Systems*, 2016, pp. 1471–1479.

[5] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: an evaluation platform for general agents," in *Proc. Int. Joint Conf. Artificial Intelligence*, 2015, pp. 253–279.

[6] R. Bellman, "On the theory of dynamic programming," *Proc. Nat. Acad. Sci*., vol. 38, no. 8, pp. 716–719, 1952.

[7] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: a review and new perspectives," *IEEE Trans. Pattern Anal. Mach. Intell*., vol. 35, no. 8, pp. 1798–1828, 2013.

[8] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Trans. Syst., Man, Cybern*., vol. 38, no. 2, pp. 156–172, 2008.

[9] M. Campbell, A. J. Hoane, and F. Hsu, "Deep Blue," *Artificial Intell*., vol. 134, no. 1-2, pp. 57–83, 2002.

[10] S. Chiappa, S. Racaniere, D. Wierstra, and S. Mohamed, "Recurrent environment simulators," in *Proc. Int. Conf. Learning Representations*, 2017.

[11] P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba. (2016). Transfer from simulation to real world through learning deep inverse dynamics model. *arXiv*. [Online]. Available: https://arxiv.org/abs/1610.03518

[12] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics," *Foundations and Trends in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[13] M. Denil, P. Agrawal, T. D. Kulkarni, T. Erez, P. Battaglia, and N. de Freitas, "Learning to perform physics experiments via deep reinforcement learning," in *Proc. Int. Conf. Learning Representations*, 2017.

[14] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2016, pp. 512–519.

[15] J. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Proc. Neural Information Processing Systems*, 2016, pp. 2137–2145.

[16] M. Garnelo, K. Arulkumaran, and M. Shanahan, "Towards deep symbolic reinforcement learning," in *NIPS Workshop on Deep Reinforcement Learning*, 2016.

[17] F. Gomez and J. Schmidhuber. "Evolving modular fast-weight networks for control," in *Proc. Int. Conf. Artificial Neural Networks*, 2005, pp. 383–389.

[18] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. Neural Information Processing Systems*, 2014, pp. 2672–2680.

[19] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep Q-learning with model-based acceleration," in *Proc. Int. Conf. Learning Representations*, 2016.

[20] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," in *Association for the Advancement of Artificial Intelligence Fall Symp. Series*, 2015.

[21] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver. "Memory-based control with recurrent neural networks," in *NIPS Workshop on Deep Reinforcement Learning*, 2015.

[22] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa, "Learning continuous control policies by stochastic value gradients," in *Proc. Neural Information Processing Systems*, 2015, pp. 2944–2952.

[23] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, A. Sendonaris, G. Dulac-Arnold, et al. (2017). Learning from demonstrations for real world reinforcement learning. *arXiv*. [Online]. Available: https://arxiv.org/abs/1704.03732

[24] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Proc. Neural Information Processing Systems*, 2016, pp. 4565-4573.

[25] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. de Turck, and P. Abbeel, "VIME: Variational information maximizing exploration," in *Proc. Neural Information Processing Systems*, 2016, pp. 1109–1117.

[26] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," in *Proc. Int. Conf. Learning Representations*, 2017.

[27] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intell*., vol. 101, no. 1, pp. 99–134, 1998.

[28] S. M. Kakade, "A natural policy gradient," in *Proc. Neural Information Processing Systems*, 2002, pp. 1531–1538.

[29] K. Kansky, T. Silver, D. A. Mély, M. Eldawy, M. Lázaro-Gredilla, X. Lou, N. Dorfman, S. Sidor, S. Phoenix, and D. George, "Schema networks: zero-shot transfer with a generative causal model of intuitive physics," in *Proc. Int. Conf. Machine Learning*, 2017, pp. 1809–1818.

[30] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *Proc. Int. Conf. Learning Representations*, 2014.

[31] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2004, pp. 2619–2624.

[32] V. R. Konda and J. N. Tsitsiklis, "On actor-critic algorithms," *SIAM J. Control Optim*., vol. 42, no. 4, pp. 1143–1166, 2003.

[33] J. Koutník, G. Cuccu, J. Schmidhuber, and F. Gomez, "Evolving large-scale neural networks for vision-based reinforcement learning," in *Proc. Conf. Genetic and Evolutionary Computation*, 2013, pp. 1061–1068.

[34] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Proc. Neural Information Processing Systems*, 2016, pp. 3675–3683.

[35] T. D. Kulkarni, A. Saeedi, S. Gautam, and S. J. Gershman, "Deep successor reinforcement learning," in *NIPS Workshop on Deep Reinforcement Learning*, 2016.

[36] T. L. Lai and H. Robbins, "Asymptotically efficient adaptive allocation rules," *Adv. Appl. Math*., vol. 6, no. 1, pp. 4–22, 1985.

[37] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, "Building machines that learn and think like people," *Behavioral Brain Sci*., pp. 1–101, 2016. [Online]. Available: https://www.cambridge.org/core/journals/behavioral-and-brain-sciences/article/building-machines-that-learn-and-think-like-people/A9535B1D745A0377E16C590E14B94993

[38] S. Lange, M. Riedmiller, and A. Voigtlander, "Autonomous reinforcement learning on raw visual input data in a real world application," in *Proc. Int. Joint Conf. Neural Networks*, 2012, pp. 1–8.

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

[40] S. Levine and V. Koltun, "Guided policy search," in *Proc. Int. Conf. Learning Representations*, 2013.

[41] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *J. Mach. Learning Res*., vol. 17, no. 39, pp. 1–40, 2016.

[42] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," in *Proc. Int. Symp. Experimental Robotics*, 2016, pp. 173–184.

[43] Y. Li. (2017). Deep reinforcement learning: An overview. *arXiv*. [Online]. Available: https://arxiv.org/abs/1701.07274

[44] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *Proc. Int. Conf. Learning Representations*, 2016.

[45] L. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Mach. Learning*, vol. 8, no. 3–4, pp. 293–321, 1992.

[46] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. Ballard, A. Banino, M. Denil, R. Goroshin, et al., "Learning to navigate in complex environments," in *Proc. Int. Conf. Learning Representations*, 2017.

[47] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[48] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Learning Representations*, 2016.

[49] S. Mohamed and D. J. Rezende, "Variational information maximisation for intrinsically motivated reinforcement learning," in *Proc. Neural Information Processing Systems*, 2015, pp. 2125–2133.

[50] O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans. (2017). Bridging the gap between value and policy based reinforcement learning. *arXiv.* [Online]. Available: https://arxiv.org/abs/1702.08892

[51] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. de Maria, V. Panneershelvam, M. Suleyman, et al., "Massively parallel methods for deep reinforcement learning," in *ICML Workshop on Deep Learning*, 2015.

[52] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *Proc. Int. Conf. Machine Learning*, 2000, pp. 663–670.

[53] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning," in *Proc. Int. Symp. Experimental Robotics*, 2006, pp. 363–372.

[54] B. O'Donoghue, R. Munos, K. Kavukcuoglu, and V. Mnih, "PGQ: Combining policy gradient and Q-learning," in *Proc. Int. Conf. Learning Representations*, 2017.

[55] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh, "Action-conditional video prediction using deep networks in Atari games," in *Proc. Neural Information Processing Systems*, 2015, pp. 2863–2871.

[56] I. Osband, C. Blundell, A. Pritzel, and B. van Roy, "Deep exploration via bootstrapped DQN," in *Proc. Neural Information Processing Systems*, 2016, pp. 4026–4034.

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

[58] P. Peng, Q. Yuan, Y. Wen, Y. Yang, Z. Tang, H. Long, and J. Wang. (2017). Multiagent bidirectionally-coordinated nets for learning to play StarCraft combat games. *arXiv.* [Online]. Available: https://arxiv.org/abs/1703.10069

[59] D. A. Pomerleau, "ALVINN, an autonomous land vehicle in a neural network," in *Proc. Neural Information Processing Systems*, 1989, pp. 305–313.

[60] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *Proc. Int. Conf. Machine Learning*, 2014, pp. 1278–1286.

[61] M. Riedmiller, "Neural fitted q iteration—First experiences with a data efficient neural reinforcement learning method," in *Proc. European Conf. Machine Learning*, 2005, pp. 317–328.

[62] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Dept. Engineering, Univ. Cambridge, MA, Tech. Rep. CUED/F-INFENG/TR 166, 1994.

[63] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. (2016). Progressive neural networks. *arXiv.* [Online]. Available: https://arxiv.org/abs/1606.04671

[64] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell. (2016). Sim-to-real robot learning from pixels with progressive nets. *arXiv.* [Online]. Available: https://arxiv.org/abs/1610.04286

[65] T. Salimans, J. Ho, X. Chen, and I. Sutskever. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv.* [Online]. Available: https://arxiv.org/abs/1703.03864

[66] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *Proc. Int. Conf. Machine Learning*, 2015, pp. 1312–1320.

[67] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. Int. Conf. Learning Representations*, 2016.

[68] J. Schmidhuber, "A possibility for implementing curiosity and boredom in model-building neural controllers," in *Proc. Int. Conf. Simulation Adaptive Behavior*, 1991, pp. 222–227.

[69] J. Schmidhuber and R. Huber, "Learning to generate artificial fovea trajectories for target detection," *Int. J. Neural Syst.*, vol. 2, no. 01n02, pp. 125–134, 1991. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/S012906579100011X

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

[71] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proc. Int. Conf. Learning Representations*, 2016.

[72] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. Int. Conf. Machine Learning*, 2014, pp. 387–395.

[73] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, et al., "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[74] S. Singh, D. Litman, M. Kearns, and M. Walker, "Optimizing dialogue management with reinforcement learning: Experiments with the NJFun system," *J. Artificial Intell. Res.*, vol. 16, pp. 105–133, Feb. 2002.

[75] B. C. Stadie, S. Levine, and P. Abbeel, "Incentivizing exploration in reinforcement learning with deep predictive models," in *NIPS Workshop on Deep Reinforcement Learning*, 2015.

[76] A. L. Strehl, L. Li, E. Wiewiora, J. Langford, and M. L. Littman, "PAC model-free reinforcement learning," in *Proc. Int. Conf. Machine Learning*, 2006, pp. 881–888.

[77] S. Sukhbaatar, A. Szlam, and R. Fergus, "Learning multiagent communication with backpropagation," in *Proc. Neural Information Processing Systems*, 2016, pp. 2244–2252.

[78] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[79] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intell.*, vol. 112, no. 1–2, pp. 181–211, 1999.

[80] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," in *Proc. Neural Information Processing Systems*, 2016, pp. 2154–2162.

[81] G. Tesauro, "Temporal difference learning and TD-gammon," *Commun. ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[82] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor, "A deep hierarchical approach to lifelong learning in Minecraft," in *Proc. Association for the Advancement Artificial Intelligence*, 2017, pp. 1553–1561.

[83] J. N. Tsitsiklis and B. van Roy, "Analysis of temporal-difference learning with function approximation," in *Proc. Neural Information Processing Systems*, 1997, pp. 1075–1081.

[84] E. Tzeng, C. Devin, J. Hoffman, C. Finn, X. Peng, S. Levine, K. Saenko, and T. Darrell, "Towards adapting deep visuomotor representations from simulated to real environments," in *Workshop Algorithmic Foundations Robotics*, 2016.

[85] N. Usunier, G. Synnaeve, Z. Lin, and S. Chintala, "Episodic exploration for deep deterministic policies: An application to StarCraft micromanagement tasks," in *Proc. Int. Conf. Learning Representations*, 2017.

[86] H. van Hasselt, "Double Q-learning," in *Proc. Neural Information Processing Systems*, 2010, pp. 2613–2621.

[87] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. Association for the Advancement of Artificial Intelligence*, 2016, pp. 2094–2100.

[88] A. Vezhnevets, V. Mnih, S. Osindero, A. Graves, O. Vinyals, J. Agapiou, and K. Kavukcuoglu. "Strategic attentive writer for learning macro-actions," in *Proc. Neural Information Processing Systems*, 2016, pp. 3486–3494.

[89] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "FeUdal networks for hierarchical reinforcement learning," in *Proc. Int. Conf. Machine Learning*, 2017, pp. 3540–3549.

[90] N. Wahlström, T. B. Schön, and M. P. Deisenroth, "Learning deep dynamical models from image pixels," in *Proc. IFAC Symp. System Identification*, 2015, pp. 1059–1064.

[91] N. Wahlström, T. B. Schön, and M. P. Deisenroth, "From pixels to torques: policy learning with deep dynamical models," in *ICML Workshop on Deep Learning*, 2015.

[92] Z. Wang, N. de Freitas, and M. Lanctot, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Learning Representations*, 2016.

[93] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," in *Proc. Int. Conf. Learning Representations*, 2017.

[94] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[95] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller, "Embed to control: A locally linear latent dynamics model for control from raw images," in *Proc. Neural Information Processing Systems*, 2015, pp. 2746–2754.

[96] D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber, "Recurrent policy gradients," *Logic J. IGPL*, vol. 18, no. 5, pp. 620–634, 2010.

[97] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learning*, vol. 8, no. 3–4, pp. 229–256, 1992.

[98] M. Wulfmeier, P. Ondruska, and I. Posner, "Maximum entropy deep inverse reinforcement learning," in *NIPS Workshop on Deep Reinforcement Learning*, 2015.

[99] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *Proc. Int. Conf. Machine Learning*, 2015, pp. 2048–2057.

[100] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2017, pp. 3357–3364.

SP