Current Advancements in Deep Reinforcement Learning

Henrik Hain

Institute for Program Structures and Data Organization (IPD)
Advisor: Dipl.-Ing. Daniel Zimmermann

With the successful triumph of deep neural networks in computer vision and the subsequent breakthroughs in the field of deep Q-learning by Mnih et al. beginning in 2013, the field of deep reinforcement learning came more and more into focus and achieved remarkable results in robotics, gaming, health care, and many other areas of research. This paper reviews recent advances of large influence in the field of deep reinforcement learning and introduces the necessary foundations for understanding those advancements.

1 Introduction

Reinforcement learning (RL) is concerned with teaching an agent to be able to reach a goal by feedback-driven exploration and exploitation of its environment. RL is clearly distinguishable from supervised and unsupervised machine learning, though RL utilizes methods from latter areas. Methods used in RL are historically more interwined with approaches used in the fields of psychology, neuroscience and optimal control strategy research. In the last few years successful application of RL to complex problems from various fields has increased significantly. The fusion of recent advancements in deep learning (DL) with RL principles, therefore coining the term deep reinforcement learning (DRL), has lead to exciting breakthroughs in fields such as gaming, robotics, or health care where novel DRL algorithms achieve human, or even super human performance. Advancements in DL have lead to breakthroughs like deep Q-networks and memory replay, which allowed to efficiently and effectively train an agent to play a set of Atari Games at a professional human level [19]. More recently Googles AlphaGo even beat 18-time world champion Lee Sedol in four out of five games of Go in 2016 [29] and AlphaStar was evaluated against the full game of Starcraft and other human players, being rated as grand master above 99.8% of all human players [32]. This success is not least due to the promising fusion of DL and RL, but according to Yuxi Li [12] also because of the synergistic effects of openly available big data sets, low cost, high performance computing power, as well as specialized stable software driving the fields of DL and DRL. DL at the core as revitalizing catalyst of this astounding

progress in the field of reinforcement learning, have lead the MIT to select DL and RL as game changing technologies in 2013's and 2017's MIT Technology Review. Silver et al. [29] even came up with the formula that artificial intelligence (AI) equals DL plus RL. Li [12] mentions the possibility to elevate the time consuming and error prone task of manually creating domain specific feature representations with the help of deep learning. The main reason for this capability of deep learning is that deep neural networks excel in extracting lower dimensional deep hierarchical feature representations of high dimensional input data being trained in an end-to-end fashion via gradient descent and back propagation. As the purpose of this paper is to give an overview on more recent advancements in DRL, while introducing the necessary foundations of reinforcement learning in parallel, it is structured as follows. Initially foundations and main core elements of RL are introduced in section 2, beginning with a neural network primer in subsection 2.1. Subsequently deep learning is presented in subsection 2.2, followed by important core elements of reinforcement learning in subsection 2.3. Section 3 highlights recent advancements in an analytical way and sketches used approaches that lead to significant improvements in specific DRL sub areas. An overview on hyper parameter optimization is given in section 4. A selection of DRL solutions, utilizing formerly introduced advancements, is discussed in section 5 and subsequent subsections. Finally, this paper closes with an outlook and discussion of the covered material in section 6.

2 Foundation and Core Elements

2.1 Neural Networks

Neural networks (NN) are a class of non-linear, non-parametric statistical models [8] used for supervised, self-supervised, and unsupervised machine learning tasks like, e.g classification, function approximation, pattern completion, and representation learning. Historically NN are used for numerous sub tasks within the reinforcement learning setting [2, 24, 31], e.g. for directly learning actions in a continuous state space. NN can be categorized as inductive, sub symbolic models, performing well in both incremental and non-incremental environments [16]. A neural network consists of several neurons arranged in individual layers. In the standard case of a fully connected feed forward neural network, each neuron of layer $n-1$ is connected to each neuron of layer n by an edge of weight w. Though, connectivity between neurons of different layers is usually an architectural decision. Layers located between the input and output layers are called hidden layers. Informally, hidden layer neurons can be considered as classifiers or property computers. Figure 1 shows a single neuron consisting of a sum function that calculates the scalar product between an input vector x and a weight vector w. The calculated scalar product serves as input to the non-linear sigmoid activation function. Training a neural network corresponds to learning the individual weights w or the weight matrix $\boldsymbol{W}$. Figure 2 shows a so-called "vanilla neural net" [8] with a single hidden layer. The net consists of an input layer with the input vector $x = [x_1, x_2, x_3, ..., x_{p-1}, x_p]^T$, a single hidden layer represented by the vector $z = [z_1, z_2, z_3, ..., z_m]^T$ and the output layer $y = [y_1, y_2, ..., y_k]^T$. To train a neural network means to minimize a loss function that quantifies the difference

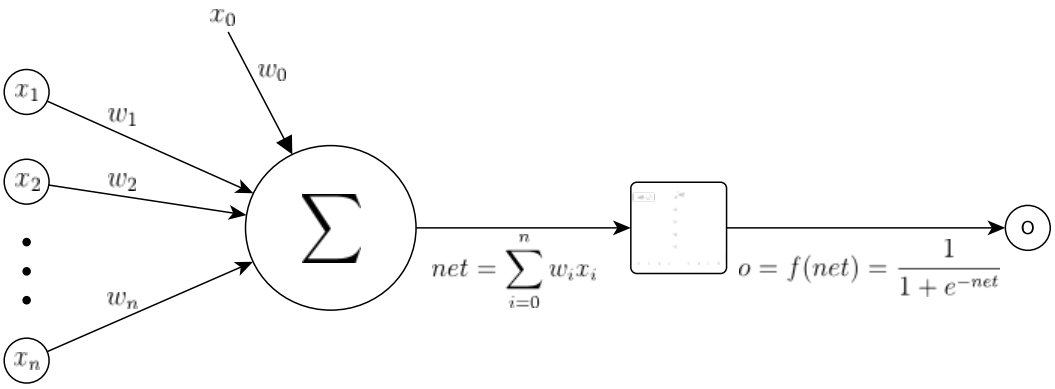


Figure 1: Schematic of a single neuron with a sigmoid activation function $f = \frac{1}{1+e^{-\sum_{i=0}^{n} x_i w_i}}$.

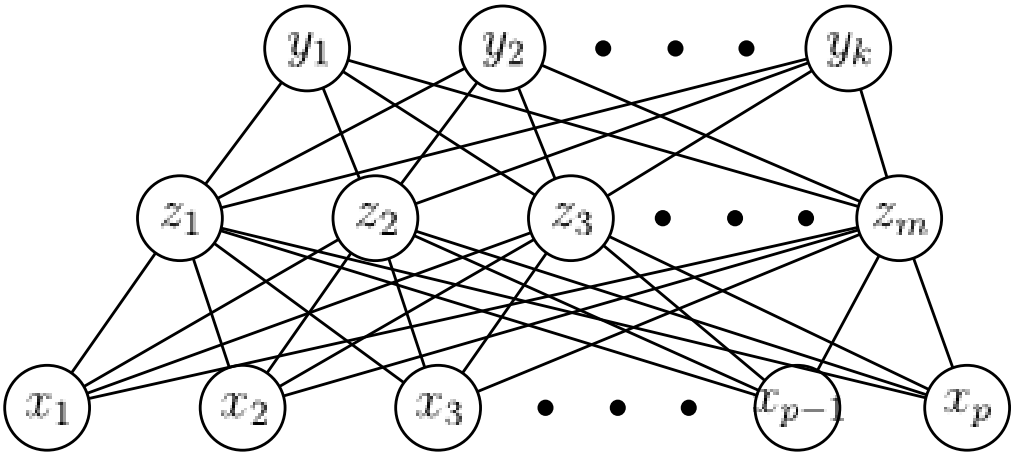


Figure 2: Schematic of a single hidden layer, feed-forward neural network.

between an expected output target vector t and an output o. In a supervised machine learning setting, training a neural network means to adjust a weight matrix $\boldsymbol{W}$ in such a way, that a loss function l, which quantifies the difference between the input vector t and an output o, is minimized. This is achieved by gradient descent on the error function and back-propagation of the corrected error portion to the individual weights. For the case of a mean square error function, this is obtained by calculating the gradient of $E(w) = \frac{1}{2|D|}\sum_{d\in D}(t_d - o_d)$, that is $\nabla E(w) = [\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_a}]$ and applying the learning rule $w \leftarrow w + \Delta w$, respectively $\Delta w = -\eta \nabla E(w)$ with learning rate η. Back propagation of the error is then performed by applying the back propagation update algorithm, given by the common delta rule presented in equation 1

$$\delta_j = \begin{cases} o_j(1-o_j), & \text{if } \sum \delta_k w_{jk} \notin output \\ o_j(1-o_j)(t_j - o_j) & \text{if } j \in output \end{cases} \tag{1}$$

A neural network has multiple hyper parameters that can be optimized such as, the number of layers to use, which type of activation function, or the kind of loss function. Hyper parameter settings are typically optimized with the help of k-fold cross-validation. Hyper parameter optimization is discussed in more detail in section 4. When a neural network uses several hidden layers for the purpose of learning feature hierarchies from, e.g. raw sensory inputs such as camera images, we speak of deep learning, which since the breakthrough of Mnih et al. ,in form of utilizing DNN for Q-learning [19], is used increasingly in the RL context.

2.2 Deep Learning

One of the main reasons for the success of deep neural networks is that they can be successfully used for facilitating or even completely eliminating the error-prone process of manual feature extraction, usually conducted by domain experts. The property of implicitly learning deep feature hierarchies makes deep neural networks very useful for the problem of reinforcement learning, e.g. an agent being able to learn effective deep hierarchical abstractions (model) of the environment the agent operates in, e.g. for the purpose of planning. Typically, DL involves many hidden layers because it is assumed that data of hierarchical structure is best represented by a similarly hierarchical model architecture. Although the idea to use neural networks for the purpose of image recognition is not new [11], it was not until 2012 that the usage of deep learning approaches increased significantly as Krizhevsky et al. [10] used a deep convolutional neural network (DCNN) approach, called AlexNet, for the challenging ImageNet Large Scale Visual Recognition Competition (ILSVRC), outperforming other approaches by a large margin. DCNN are especially useful for extracting lower-dimensional feature hierarchies from high-dimensional sensory inputs, e.g. from camera images, eventually leading to the development of deep Q-networks [18] in 2015 and their successors mentioned in section 3 . A DCNN is a type of feed forward network comprised of sequentially ordered convolutional, pooling, and fully connected layers. A DCNN uses convolution layers for performing filtering operations on raw input data, e.g. multiplication or accumulation, to drastically reduce the number of required parameters and to achieve computational feasibility. To further speed up

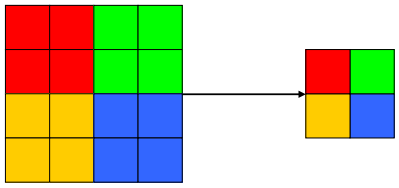


Figure 3: Functional schema of a max-pooling ($max(.)$) layer with $2 * 2$ filter and stride 2

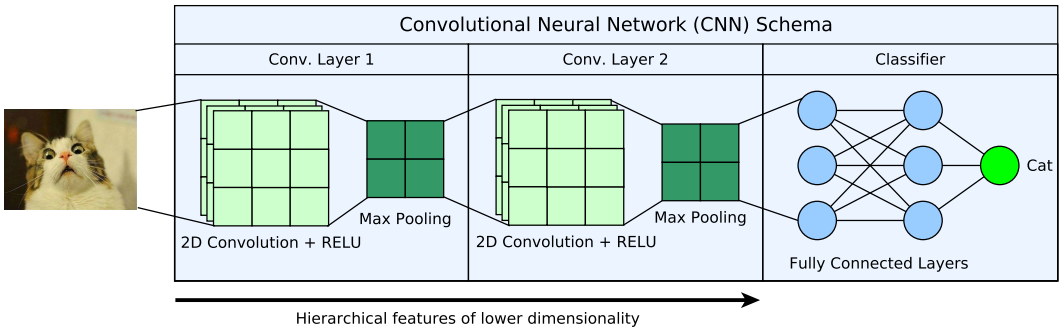


Figure 4: Schematic of a convolutional neural network with two convolutional layers, two max pooling layers and a fully connected feed forward classifier.

computation and avoid saturation, often a rectified linear unit $g = max(0, x)$ (RELU) is used as activation function. To increase robustness against noise and disorder, pooling layers are used, which apply a dimensionality reducing sum, average or maximum function to image patches. Figure 3 shows the schematic functionality of a max pooling layer calculating a pooled feature map by applying $max(.)$ with $2 * 2$ filter and stride 2. Figure 4 shows the schematic of a DCNN with 2 convolution layers where each convolution layer has its own pooling layer and a fully connected feed forward neural network classifier for cat recognition following. This type of network architecture is advantageous since the network can be trained end-to-end in one pass and be applied directly to the raw input data. Another type of deep neural networks are recurrent neural networks (RNN), where connections between nodes form a directed temporal graph and all levels share the same weight w when unfolded. RNNs are typically used to process sequential inputs, e.g. speech or continuous writing input data. As RNN are not able to retain information over time due to the vanishing gradient issue, Hochreiter and Schmidhuber proposed long short term memory networks LSTM [9], whereas more recently Chung et al. proposed the introduction of gated recurrent units [5] in 2014, by utilizing gating approaches for modifying information in recurrent nodes.

An extraordinary feature of neural network structures is that they can be combined and trained end-to-end using the back propagation algorithm. This characteristic makes them suitable for a wide range of applications in reinforcement learning.

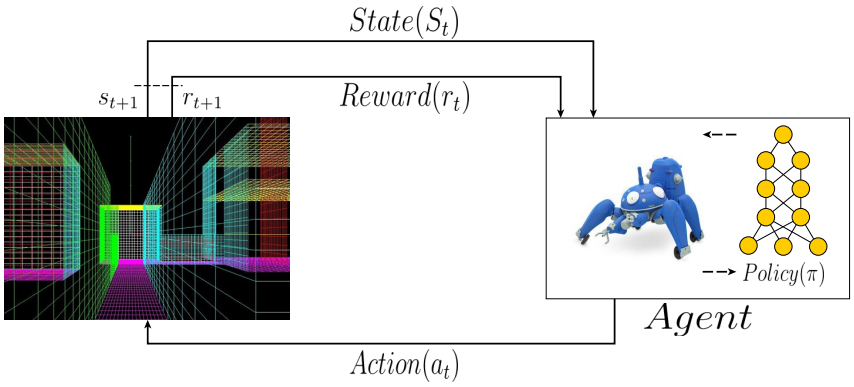


Figure 5: Adapted schematic of the perception-action-learning loop from Arulkumaran et al. [1]

2.3 Reinforcement Learning

Reinforcement learning is the mapping of situations to actions for reaching a specified goal. In a computational environment this corresponds to the maximization of a numerical reward signal according to [31]. As optimal actions to reach a goal are usually not known in advance, an agent, or actor, has to experiment in the given environment. Experimentation can be broken down into *exploration* and *exploitation*, e.g. execute possible actions to find those with the highest reward. The most difficult case occurs when actions influence not only immediate but also subsequent rewards.

Experimentation and result, respectively "trial-and-error search and delayed reward" [31] are key properties distinguishing reinforcement learning from other types of machine learning. Thus, one can identify *four main elements* of reinforcement learning. Firstly, a *policy* representing the behavior of an agent, mapping sensory perception of the environment to possible actions in an environment at a given state. Secondly, an immediate *reward signal* at a single time step from the environment, where it is the objective of the agent to maximize the total reward received.

Thirdly, a *value function* specifying the total amount of reward expected from a given state, therefore defining the value of a state. Finally, an optional *model* of the environment resembling the behavior of the environment, allowing the agent to utilize inference and planning methods. More formally, a RL agent in a discrete state and action space is interacting, or experimenting, with, an environment, or model of an environment, over time, perceiving a state s_t from a state space S at each time step t and reacting by selecting an action a_t from action space A according to policy $\pi(a_t|s)$, transitioning to state s_{t+1}. The agent receives a reward r_t according to $R(s,a)$, with $r_t \in \mathbb{R}$, in case a model is given, or else according to a probability $P(s_{t+1}|s_t, a_t)$. In non-episodic cases, rewards are weighted by a discount factor γ, assuming earlier rewards have less impact on the current state. This leads to the expected, discounted return function $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+l}$ with discount rate $\gamma \in (0, 1]$, depicted as perception-action-loop in figure 5.

An agent in a RL problem operates on the basis of state and action evaluation rather than receiving instructions on what to do. This characteristic stands in contrast to supervised and unsupervised machine learning. Thus an agent needs to balance search for better

actions against exploiting actions already known. This is a well known problem from statistical decision theory and formalized as k-armed bandit problem, which naturally develops to the full reinforcement learning problem.

In a k-armed bandit problem, each action (k) is associated with an expected reward, denoting the value of the action, or more formally $q_*(a) = \mathbb{E}(R_t|A_t = a)$, where $q_*(a)$ is the reward under the condition that a is selected. Maintaining a set of action value estimates, at any time step there is at least one action with the greatest action value estimate, called greedy actions. Selection of only greedy actions ($A_t = max_a\ Q_t(a)$) lead an agent to never explore the environment for actions of possibly greater value, thus action selection usually follows an ϵ-greedy approach where, with the probability $1 - \epsilon$, an action is chosen at random from the set of possible actions. This form of k-armed bandit is appropriate for stationary reward probabilities, where in case of non-stationary bandit problems, the incremental update rule takes the form $Q_{n+1} = Q_n + \gamma[R_n - Q_n]$, similar to the discounted return function introduced above. Subsequently, there are more sophisticated methods for balancing exploration vs. exploitation, including optimistic initial value setting, upper-confidence-bound (UCB) action selection, and gradient bandit algorithms as 'stochastic approximation to gradient ascent, eventually leading to contextual bandits essentially performing an associative search.

The more complex problem of reinforcement learning itself is formalized within the problem of the finite Markov decision process (MDP). In a MDP, R_t and S_t correspond to a discrete probability distribution adhering to the Markov property according to

$$p(s', r|s, a) = Pr(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a), \tag{2}$$

which defines the dynamics for a MDP for all $s', s \in S, r \in R$, and $a \in A(s)$ in form of a 4-argument function $p : S \times R \times S \times A \rightarrow [0, 1]$ [31]. Value functions, or functions of state-action pairs, are involved in almost all reinforcement learning algorithms. Value functions estimate the expected return, or future rewards to be expected, given that an agent is in a distinct state and what actions it will take. Subsequently value functions are interwoven with policies, which describe certain ways of acting w.r.t. the value of this particular action. Sutton describes a policy as "... a mapping from states to probabilities of selecting each possible action." [31]. Within the MDP framework, the value function of policy π in a state s is defined as

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s], \text{for all } s \in S, \end{aligned} \tag{3}$$

and denotes the expected, accumulative, and discounted future reward. Subsequently, the action-value function for policy π, quantifying the value of taking an action a in a state s, eventually following the policy π given by

$$\begin{aligned} q_\pi(s, a) &\doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_t + k + 1|S_t = s, A_t = a], \end{aligned} \tag{4}$$

where $v_*(s) = max_\pi\ v_\pi(s) = max_a q_{\pi^*}(s,a)$ denotes the maximum state value achievable by any policy for state s [12]. Equation 5 introduces the Bellman form for v_π, highlighting the fundamentally recursive property of value functions used throughout the field of RL and dynamic programming. The Bellman equation expresses the relationship between the value of a state and the values of its successor states through summation over all values of a, s', and r by calculating the probability $\pi(a|s)p(s',r|s,a)$ used as weight for the quantity in brackets for each triple [31]

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \sum_{a\in A} \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')], \text{for all } s \in S. \end{aligned} \tag{5}$$

Optimality or, retrieving the highest reward, is reached by choosing a greedily at every state which leads to the following recursively defined Bellman optimality equation 6 without referring to any distinct policy

$$\begin{aligned} v_*(s) &= \max_{a\in A(s)} q_{\pi_*}(s,a) \\ &= \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')], \end{aligned} \tag{6}$$

and subsequently to the action value function

$$\begin{aligned} q_*(s,a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_t + 1, a')|S_t = s, A_t = a] \\ &= \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} q_*(s',a')]. \end{aligned} \tag{7}$$

Temporal difference (TD) learning addresses the problem of predicting a quantity that actually depends on a future value of the signal to predict and is central to the problem domain of RL. This means that TD learning produces estimates based on other estimates, therefor they bootstrap. They are beneficial as they do not require a model of the environment, as necessary for dynamic programming (DP) solution methods, furthermore they have an advantage over Monte Carlo solutions, as they are implemented in a fully incremental fashion and do not have to wait for the end of a learning episode [31]. That means In RL TD learning refers to learning methods of value functions [30] resulting in Q-learning [34], being an off-policy learner and state-action-reward-state-action (SARSA) [31] being an on-policy learner.

In the case of on-policy SARSA the TD prediction method

$$Q(s_t, A_t) \leftarrow Q(S_t, A_t) + [\alpha\ R_{t+1} + \gamma\ Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)], \tag{8}$$

which learns an action-value function by estimating $q_\pi(s,a)$ for policy π and for all states s and actions a. Whereas in off-policy Q-learning the TD control method is specified as

$$Q(s_t, A_t) \leftarrow Q(S_t, A_t) + [\alpha\ R_{t+1} + \gamma \max_a\ Q(S_{t+1}, a) - Q(S_t, A_t)], \tag{9}$$

directly approximating the optimal action-value function q_*, independently from any policy being followed.

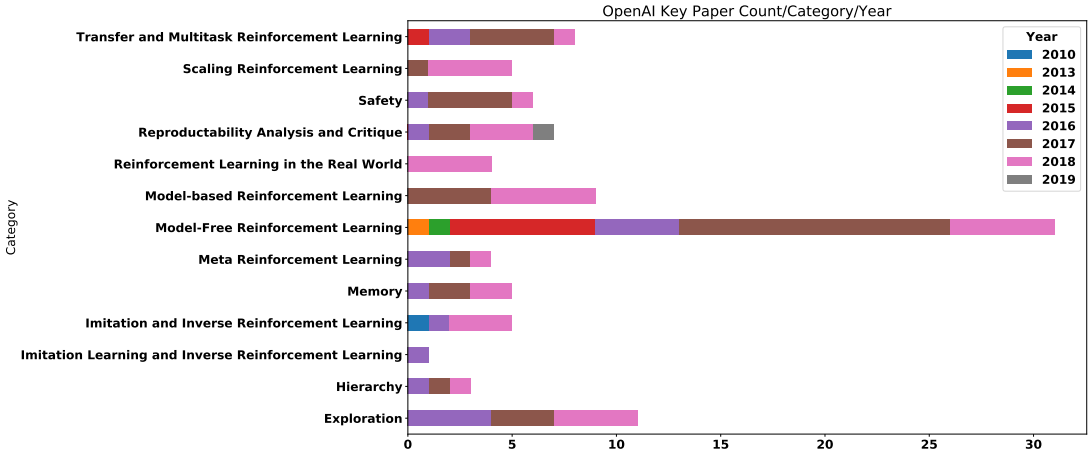
Table 1: OpenAI reinforcement learning categories and sub categories.

Category	**Sub Category**
Model-Free RL	Deep-Q-Learning
	Policy Gradients
	Deterministic Policy Gradients
	Distributional RL
	Policy Gradients with Action-Dependent Baselines
	Path-Consistency Learning
	Other Directions for Combining Policy-Learning and Q-Learning
	Evolutionary Algorithms
Exploration	Intrinsic Motivation
	Unsupervised RL
Transfer and Multitask RL	
Hierarchy	
Memory	
Model-Based RL	Model is Learned
	Model is Given
Meta-RL	
Scaling RL	
RL in the Real World	
Safety	
Imitation Learning and Inverse Reinforcement Learning	
Reproducibility, Analysis, and Critique	

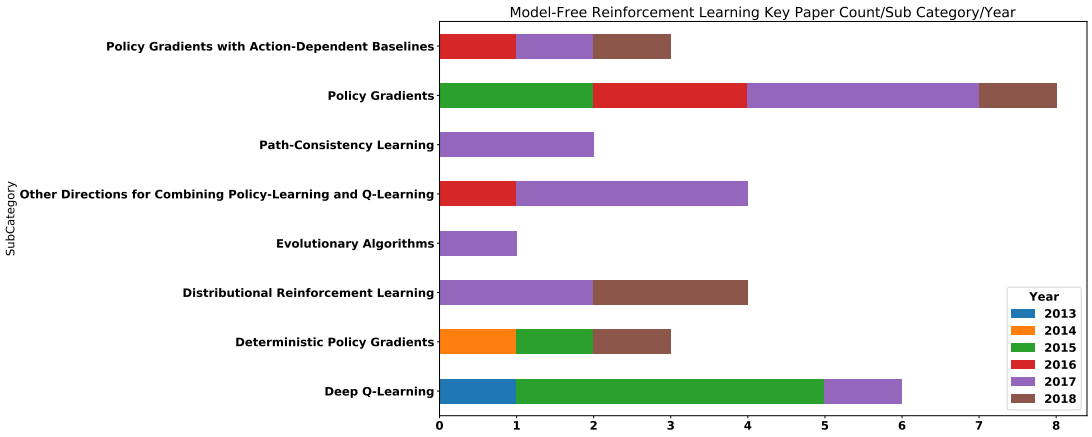
3 Deep Reinforcement Learning

Both "Deep Reinforcement Learning - A brief survey" by Arulkumaran et al. from 2017 [1] and "Deep Reinforcement Learning: An Overview" by Yuxi Li 2018 [12] already give an up-to-date, excellent overview on current advancements in deep reinforcement learning. For this reason we have decided to apply a semi empirical approach to the evaluation of novel deep reinforcement research, thus we have retrieved the curated list of 100 key papers of reinforcement learning as of 2020-01-05 from OpenAI [1], assuming that this selection covers research of far-reaching influence already. The curators of OpenAI identify thirteen distinct categories and twelve subcategories of reinforcement learning, which are listed in table 1, subsequently assigning current research to those categories. Afterwards we enriched the fetched data with meta information in form of 'cited by' and 'citing' counts, with the help of bibtex. We assume that the 'cited by' count is an indicator of the extent of influence of the current work, respectively the sum of those counts, within each main

[1] https://spinningup.openai.com/en/latest/spinningup/keypapers.html



(a) RL main research area publications counts

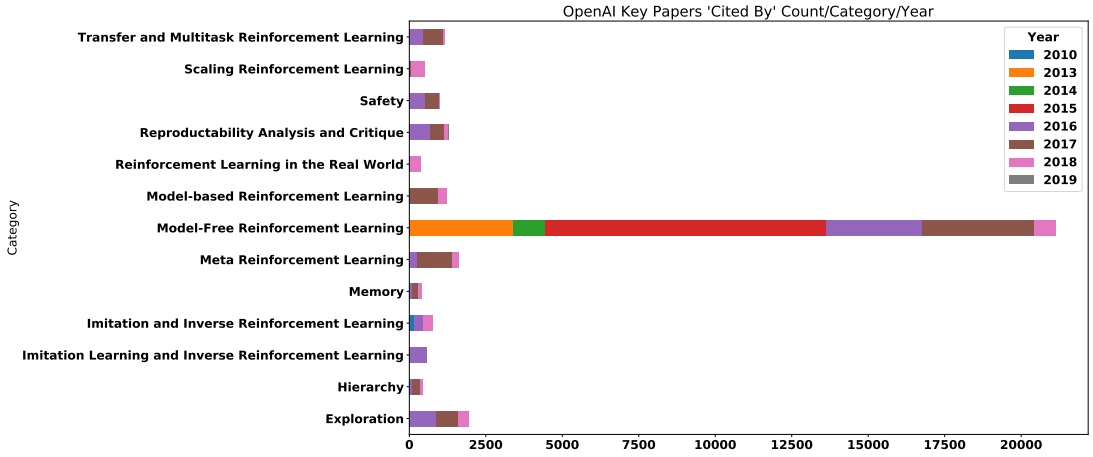


(b) RL model free research sub area publications counts

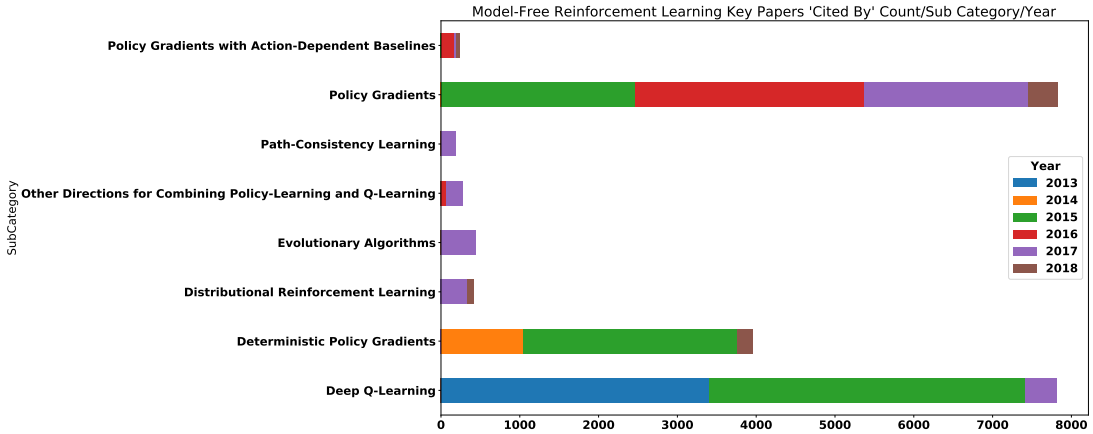
Figure 6: (a) depicts publication counts for all main research areas between 2010 and 2019. (b) depicts publication counts of model-free research sub areas between 2013 and 2018.

and sub category, a measure of influence regarding those areas. This analysis has been carried between 2010 and 2019. Finally, we make an informed assumption about where the field of deep reinforcement learning will focus and where it will move to together with an overview of the top influential key papers. The key paper data underlying our conclusions can be downloaded from [2]. Throughout the period considered, research is clearly focused on the main area of *model-free reinforcement learning* receiving over 20000 citations, with the next three most frequently cited main areas being *exploration*, *meta reinforcement learning*, and *transfer and multitask reinforcement learning*, not quite reaching 6000 citations together. The main category *model-free reinforcement learning* consists of eight sub categories, whereby the main research focus is divided equally between *deep Q-learning* and *policy gradients*, with third place being *deterministic policy gradients*. Table 2 lists the top ten most cited papers in reinforcement learning in descending order . The

[2]https://gist.github.com/hhain/cbdbcca13b9585c81fefd7d8f8825c64



(a) RL main research are 'cited-by' counts



(b) RL model-free research sub area 'cited-by' counts

Figure 7: (a) depicts 'cited-by' counts for all main research areas between 2010 and 2019. (b) depicts 'cited-by' counts of model-free research sub areas between 2013 and 2018.

Table 2: Top ten most cited papers in deep reinforcement learning.

Title	Cited by	Year
Playing Atari with Deep Reinforcement Learning	3401	2013
Continuous Control With Deep Reinforcement Learning	2707	2015
Asynchronous Methods for Deep Reinforcement Learning	2634	2016
Trust Region Policy Optimization Algorithms	1788	2015
Deep Reinforcement Learning with Double Q-learning	1599	2015
Proximal Policy Optimization Algorithms	1579	2017
Model-Agnostic Meta-Learning for Fast Adaption of Deep Networks	1156	2017
Deterministic Policy Gradient Algorithms	1045	2014
Prioritized Experience Replay	1005	2015
Dueling Network Architectures for Deep Reinforcement Learning	897	2015

remainder of this section gives an overview of the introduced achievements of the papers from table 2 in chronological order.

Playing Atari with Deep Reinforcement Learning by Mnih et al. [19] marked the beginning of the advance of deep learning methods in RL by introducing Deep-Q networks (DQN) and experience replay, allowing them to play a range of Atari 2600 games at a professional level through directly learning optimal control from raw screen pixels. The first approach to achieve this goal, they utilized a DCNN (see section 2.2) as non-linear function approximator to estimate the action value function by minimizing a sequence of loss functions $L_i(\theta_i)$" [19], changing at each iteration i. This approach lead to the loss function introduced in equation 10,

$$L_i = \mathbb{E}_{s,a\sim\rho(.)}[(y_i - Q(s,a;\theta_i))^2] \tag{10}$$

, and the corresponding gradient, shown in equation 11, used during optimization via stochastic gradient descent.

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(.);s'\sim\epsilon}[(r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i))\nabla_{\theta_i} Q(s,a;\theta_i)] \tag{11}$$

Secondly, they used a technique called *experience replay* by persisting the experience of the agent into a separate data set at each time step and pooled into a replay memory from which they sample randomly during Q-learning minibatch updates, increasing data efficiency and breaking correlation between samples. The algorithm Mnih et al. developed is an *model-free*, *off-policy* learner, solving the learning task by learning spatio-temporal features directly and following an ϵ-greedy strategy.

Though a DQN is able to solve the problem of RL in high-dimensional raw input spaces, it does not address the problem of an equally high-dimensional continuous action space because a_t would require an optimization at every timestep that proves infeasible for most continuous action environments. To address this problem Lillicrap et al. [13] transfer Deep Q-Learning to the domain of continuous action control by introducing the deep deterministic policy gradient (DDPG) algorithm, an *actor-critic* algorithm based on *deterministic policy gradients* (DPG), which allows them to operate in a non-episodic continuous action space. Similar to Mnih et al. [19] a replay buffer is used to decrease sample correlation and utilize hardware optimizations which allows for mini-batch learning. In contrast to DQN the target network is modified for the actor-critic approach using "soft" target updates, instead of a direct weight copy mechanism. By copying both actor and critic network as target networks together with the delayed weight update rule $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$ with $\tau << 1$ stability of learning is improved while learning speed is decreased. To account for different physical units in input data, Lillicrap et al. utilize a technique called batch normalization so that each dimension in a minibatch has unit mean and variance. DDPG, being an off-policy algorithm, has the benefit that the challenging task of exploration can be handled independently from the learning algorithm itself. The exploration policy μ' is sampled from a noise process N suiting the environment. They evaluate the algorithm on 20 classical simulated physics tasks, without changing the used hyper parameters in between.

In the paper "Asynchronous Methods for Deep Reinforcement Learning" [17] Mnih et al. address the training time problem with the help of an asynchronous gradient decent

actor-critic method, allowing for parallelization of the learning task, eventually leading to two-fold training speed-up on a single multi-core CPU within the Atari game domain. Instead of using a replay buffer to address non-stationary and strongly correlated data, multiple agents are executed in parallel on multiple instances of the environment. This approach enables a larger spectrum of on- and off-policy approaches, e.g. *actor-critic* and *n-step* methods or *SARSA*, to be used efficiently. Instead of using multiple separate machines for asynchronous actor-learners, Mnih et al. use multiple CPU threads on a single machine, therefor reducing network communication overhead and additionally allows to employ *Hogwild!* [22] for training updates, eventually leading to the best performing asynchronous advantage actor-critic (A3C) approach.

Schulman et al. [26] introduced the "Trust Region Policy Optimization" (TRPO) algorithm with guaranteed monotonic policy optimization improvements, while being similar to the usually used policy gradient methods. This is achieved by taking the largest performance improving step possible, while adhering to a KL-Divergence constraint defining the minimal distance between the old and new policy probability distributions.

Overestimation of action values, due to the max operator in standard Q-learning and DQN, was addressed by Hasselt et. al [7] by generalizing the idea of tabular Double Q-learning to the domain of non-linear large-scale function approximators via adapting DQN, outperforming DQN on serveral games from the Atari domain. This is achieved through decomposition of the max operation by using the target network to estimate the greedy policy of the online network, therefor evaluating it. Thus Y_t^{DQN} is replaced with

$$Y_t^{DoubleDQN} = R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{max}\ Q(S_{t+1}, a; \theta_t), \theta_t^-) \tag{12}$$

In 2017, Schulman et al. [25] proposed a new family of policy gradient methods, using an alternating approach, called proximal policy optimization (PPO). This approach alternates between experimentation within the environment and optimizing an artificial stochastic objective function via gradient ascent. Additionally, the objective function enables the use of multi-epoch minibatch updates.

Meta learning was addressed by Finn et al. [6] by introducing a model-agnostic meta-learning algorithm (MAML) being compatible with any model trained with gradient descent from various problem domains, e.g. classification, regression, and reinforcement learning. They reduce the necessity of large training data sets by parameter value training of the model itself, which allows for both good generalization performance and fine-tuning on a specific learning task. This form of rapid adaption is frequently formalized as few-shot learning, with the goal to adapt quickly to novel tasks only with the help of a few data points. Obviously, meta learning is closely related to the area of hyperparameter optimization and automatic machine learning (AutoML) reviewed in section 4.

Efficiency of the policy gradient estimation problem was addressed by Silver et al. [27] in 2014 by introducing deterministic policy gradients for reinforcement learning for the problem area of continuous action spaces. Deterministic policy gradients are desirable because they are the expected gradient of the action value function on the one hand and can be calculated efficiently on the other.

Schaul et al. [23] doubted the effectiveness of uniform sampling at random from the replay memory and proposed experience prioritization for weighting important transitions

higher to increase learning effectiveness, leading the algorithm to outperform DQN with standard replay memory in 41 out of 49 games. In prioritized experience replay (PER) transitions with a high expected influence on the learning process are more frequently replayed than transitions with less expected influence. The expected learning influence is measured by the order of magnitude of the corresponding TD error. The introduced sampling diversity is addressed with a combination of stochastic prioritization, while the introduced bias is corrected through importance sampling.

With dueling network architectures Wang et al. [33] presented a novel NN architecture using one estimator for learning the state value function and one for the *state-dependent action advantage function*, with the benefit to, without introducing change to the foundational RL algorithm, generalize action learning. The benefit of the dueling network architecture is, that it is able to learn the value of a state without having the evaluate the influence of each single action for each state. The neural architecture for representation learning follows the standard architecture of the DQN, but subsequently followed by two streams of fully connected layers instead of a single one. One stream is constructed with the purpose of providing the value function, while the other stream provides the advantage function. The final output, a set of Q values, is calculated by the combination of both streams. It is important to mention that the function that combines the outputs of both streams is implemented as part of the network itself, which means that only backpropagation is required to train the dueling network.

4 Hyper Parameter Optimization

Hyper parameter optimization (HPO) deals with the optimal selection of a number of learning parameters of a machine learning process which are not part of the learning process itself, but control learning behavior as well as learning capacity. In DRL there are numerous hyper parameters depending on the combination RL algorithms, DL procedures, and the environment or model. In RL typical hyper parameters to account for are the discount factor γ and the exploration rate ϵ and in combination with deep learning the set of hyper parameters extends by the learning rate η, the number of layers, the number of neurons per layer, the kind of pooling function in case of a DCNN, among others that depend on the specific procedure setup. Including the problem of selecting appropriate algorithms, the problem of *Combined Algorithm Selection and Hyperparameter optimization (CASH)* as generalization of the HPO problem is defined as follows.

Definition 4.1. *(Cash) Let* $A = \{A^{(1)}, ..., A^{(R)}\}$ *be a set of algorithms, and let the hyper parameters of each algorithm have domain* $\Lambda^{(j)}$*. Further, let* $D_{train} = \{(x_1, y_1), ..., (x_n, y_n)\}$ *be a training set which is split into K cross-validation folds* $\{D^{(1)}_{valid}, ..., D^{(K)}_{valid}\}$ *and* $\{D^{(1)}_{train}, ...D^{(K)}_{train}\}$ *such that* $D^{(i)}_{train} = D_{train} \backslash D^{(i)}_{valid}$ *for* $i = 1, ..., K$*. Finally, let* $L(A^{(j)}_{\lambda}, D^{(i)}_{train}, D^{(i)}_{valid})$ *denote the loss that algorithm* A^j *achieves on* $D^{(i)}_{valid}$ *when trained on* D^{i}_{train} *with hyper parameters* λ*. Then, the Combined Algorithm Selection and Hyperparameter optimization (CASH) problem*

is to find the joint algorithm and hyper parameter setting that minimizes

$$A^*, \lambda_* \in \underset{A^{(j)} \in A, \lambda \in \Lambda^{(j)}}{argmin} \frac{1}{K} \sum_{i=1}^{K} L(A_\lambda^{(j)}, D_{train}^{(i)}, D_{valid}^{(i)}). \tag{13}$$

Given, that algorithms and procedures to be used are already selected the problem of solving CASH reduces to the somewhat simple problem of solving

$$\lambda_* \in \underset{\lambda \in \Lambda^{(j)}}{argmin} \frac{1}{K} \sum_{i=1}^{K} L(D_{train}^{(i)}, D_{valid}^{(i)}). \tag{14}$$

Traditionally, approaches like grid or random search are employed to tackle the latter optimization problem although they are often not feasible due to the high dimensionality of the hyper parameter search space. More recently, Baysian optimization is used successfully for HPO.

There are other methods for optimizing hyper parameters, e.g. those using evolutionary or genetic algorithms to optimize equation 14 as well as emerging research fields like neural architecture search and AutoML contributing to HPO. Furthermore it is important to note, that DRL istself can be utilized for HPO.

5 Current Fields of Application

5.1 Games

One of the most optimal test fields for deep reinforcement learning are games [12]. Game environments have been used for a long time to test and evaluate the performance of reinforcement learning algorithms. Games, be they classic board games like backgammon or modern strategic computer games like StarCraft, fall into one of two categories, games with perfect, and games with imperfect information. Backgammon, Go, and Space Invader are perfect information games, as game participant are able to observe the entire game state at all times. The opposite is true for games like Mahjong, Texas Hold'em Poker or StarCraft, therefore being members of the class of imperfect information games.

AlphaGo proposed by Silver et al. [28] was the first RL incorporating approach being able to win against a human European Go champion $5:0$ on the full 19×19 board and $4:1$ against Lee Sedol, the 18-time Go world champion, as the complexity in Go stems from both a huge search space size (250^{150}) and the difficulty of position evaluation this result marks an outstanding achievement in RL. AlphaGo is a combination of Monte Carlo tree search (MCTS), supervised learning, DCNN, and reinforcement learning based on a two phases, the first in form of the NN training pipeline, the second one in form of MCTS, furthermore it is comprised of a supervised policy DCNN learning from expert moves, a RL policy and a RL value network.

The sucessor AlphaGo Zero [29] differs from AlphaGo as it is an approximate policy iterator, including MCTS in the inner training loop to provide policy evaluation and policy improvement in parallel. Two distinct improvements upon AlphaGo are learning from random self-play RL, not needing human intervention and unification AlphaGo's

policy and value NN into one single network. In contrast to AlphaGo, AlphaGo is a pure reinforcement learning algorithm, neither being an unsupervised or supervised approach.

Vinyals et al. addressed the challenging task of playing the imperfect information game StarCraft II in 2019 in form of AlphaStar [32], which was the first system capable of beating a professional player 5 : 0, where AlphaStar is, like AlphaGo, a mix of supervised learning and reinforcement learning. StarCraft II poses a challenging test bed for reinforcement learning, where one need to balance short- and long-term goals in a real time environment and within a very large hierarchical action space.

5.2 Computer Vision

Automatically getting high-level insight and understanding from images and videos is the topic of the scientific, interdisciplinary field computer vision (CV). As an universal learning approach, RL can be advantageously applied to many problem domains of CV, e.g. recognition, motion analysis, scene understanding, and object affordance analysis.

For example, Cao et al. [4] proposed a face hallucination framework using DRL for generating high-resolution face images from low resolution inputs. A recurrent policy network is used for learning a policy responsible for selecting a facial part at each enhancement step, and a local enhancement network for the facial part hallucination. This approach results in selecting a new facial part by taking into account a former, already improved facial part, eventually learning an optimal hallucination path.

Brunner et al. [3] employed DRL to navigate in a 3D environment with the help of a complex tool in form of a 2D map to find the shortest way out of a maze. They decompose the complex main task of navigation into three distinct subtasks, e.g. environment observation and location mapping, map-environment understanding, and planning. The complete agent is trained as A3C agent with a combination of on-policy and off-policy losses with sample data from a replay memory.

5.3 Natural Language Processing

Natural language processing (NLP) is concerned with processing natural language data with the purpose of, e.g. understanding, information extraction, sentiment analysis, summarization of text data, among other areas of interest.

Paulus et al. [20] proposed a model for abstractive text summarization utilizing a learning method combining supervised word prediction and RL, improving performance regarding long texts from CNN/Daily Mail and the New York Times datasets.

5.4 Health Care

Healthcare, the organized provision of medical care to individuals or communities, is a very extensive field offering multiple possibilities for the practical application of DL and DRL, given the vast amount of available biomedical data which is typically unstructured, heterogeneous, complex, and poorly annotated. Miotto et al. [21] list the use of various DL models and architectures regarding data from clinical imaging, eletronic health records,

genomics, and mobile wearables, elevating the need of manual feature engineering. They cover a wide range of practically applied DL models.

Complementary Yu et al. [35] surveyed RL in healthcare from the perspective of applicability in the areas of dynamic treatment regimes for chronic diseases and critical care as well as for automated medical diagnosis and other subareas in healthcare, e.g. resource scheduling and allocation or optimal process control.

5.5 Robotics

Robotics, an interdisciplinary branch of engineering and science, is classically one of the main and original areas of application that benefits from current advancements in reinforcement learning. For example, the task to navigate in complex environments [15], fundamental to intelligent behavior, is addressed by Mirowski et al. They use a goal driven end-to-end learning approach that applies A3C on auxiliary depth information among others.

Mahler and Goldberg [14] model the task of bin picking, defined as sequentially grasping objects from an unordered heap and transporting into a packing box, as partially observable Markov decision process and collect artificial demonstrations from an algorithmic supervisor from which they learn a deep policy via fine-tuning a CNN. They achieve 94% success rate and 96% precision on average in 2192 physical simulation trials.

6 Conclusion and Outlook

Reinforcement learning is a thriving field, brimming with current research activities and exciting recent advancements leading to this paper. Subsequently, we have introduced RL and DRL by highlighting DRL progress in playing Atari Games [19], the game of Go [29], and StarCraft [32] as well as highlighting the role DRL has in developing artificial intelligence systems [29]. Unfortunately, due to the scope of the concepts, not all of them could be introduced in sufficient depth without going significantly beyond the scope of this work. To help with understanding DRL advancements, we have reviewed foundational core elements in section 2, specifically NN in subsection 2.1 and the basics of DL in subsection 2.2 followed by the main principles of RL in subsection 2.3. To distinguish this paper from Yuxi Li's 'Reinforcement Learning: An Overview' [12] and Arulkumaran et al.'s 'Deep Reinforcement Learning - A brief survey' [1] we conducted a semi-formal analysis regarding current key paper importance while introducing influential recent concepts in parallel in the main section 3. Afterwards, hyper parameter optimization as a key issue of high importance, spanning all formerly introduced concepts, is dealt with separately in section 4. This paper concludes with an examination of the role of reinforcement learning in several different application areas, from games to robotics, in section 5 . In the future, it is intended to conduct research on the interrelationships of the individual advances in DRL with each other and with the foundations of RL, to be able to better anticipate the interrelationships between future developments in the field of DRL.

References

[1] Kai Arulkumaran et al. “Deep Reinforcement Learning: A Brief Survey”. In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38. DOI: 10.1109/msp.2017.2743240.

[2] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming*. Belmont, MA: Athena Scientific, 1996.

[3] Gino Brunner et al. *Teaching a Machine to Read Maps With Deep Reinforcement Learning*. 2018. URL: https://aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16387/16545.

[4] Qingxing Cao et al. “Attention-Aware Face Hallucination via Deep Reinforcement Learning”. In: *CoRR* abs/1708.03132 (2017). arXiv: 1708.03132. URL: http://arxiv.org/abs/1708.03132.

[5] Junyoung Chung et al. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. cite arxiv:1412.3555Comment: Presented in NIPS 2014 Deep Learning and Representation Learning Workshop. 2014. URL: http://arxiv.org/abs/1412.3555.

[6] Chelsea Finn, Pieter Abbeel, and Sergey Levine. “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”. In: *CoRR* abs/1703.03400 (2017). arXiv: 1703.03400. URL: http://arxiv.org/abs/1703.03400.

[7] Hado van Hasselt, Arthur Guez, and David Silver. “Deep Reinforcement Learning with Double Q-learning”. In: *CoRR* abs/1509.06461 (2015). arXiv: 1509.06461. URL: http://arxiv.org/abs/1509.06461.

[8] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. 2nd ed. Springer, 2009. URL: http://www-stat.stanford.edu/~tibs/ElemStatLearn/.

[9] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[11] Yann LeCun et al. “Backpropagation Applied to Handwritten Zip Code Recognition.” In: *Neural Computation* 1.4 (1989), pp. 541–551. URL: http://dblp.uni-trier.de/db/journals/neco/neco1.html#LeCunBDHHHJ89.

[12] Yuxi Li. “Deep Reinforcement Learning: An Overview.” In: *CoRR* abs/1701.07274 (2017). URL: http://dblp.uni-trier.de/db/journals/corr/corr1701.html#Li17b.

[13] Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2015. arXiv: 1509.02971 [cs.LG].

[14] Jeffrey Mahler and Ken Goldberg. "Learning Deep Policies for Robot Bin Picking by Simulating Robust Grasping Sequences". In: *Proceedings of the 1st Annual Conference on Robot Learning*. Ed. by Sergey Levine, Vincent Vanhoucke, and Ken Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR, Nov. 2017, pp. 515–524.

[15] Piotr Mirowski et al. "Learning to Navigate in Complex Environments". In: *CoRR* abs/1611.03673 (2016). arXiv: 1611.03673. URL: http://arxiv.org/abs/1611.03673.

[16] Tom M. Mitchell. *Machine Learning*. New York: McGraw-Hill, 1997. ISBN: 978-0-07-042807-2.

[17] Volodymyr Mnih et al. "Asynchronous methods for deep reinforcement learning". In: *International Conference on Machine Learning*. 2016, pp. 1928–1937. URL: https://arxiv.org/pdf/1602.01783.pdf.

[18] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 00280836. URL: http://dx.doi.org/10.1038/nature14236.

[19] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. NIPS Deep Learning Workshop 2013. 2013. URL: http://arxiv.org/abs/1312.5602.

[20] Romain Paulus, Caiming Xiong, and Richard Socher. "A Deep Reinforced Model for Abstractive Summarization". In: *CoRR* abs/1705.04304 (2017). arXiv: 1705.04304. URL: http://arxiv.org/abs/1705.04304.

[21] Baolin Peng et al. "Composite Task-Completion Dialogue System via Hierarchical Deep Reinforcement Learning". In: *CoRR* abs/1704.03084 (2017). arXiv: 1704.03084. URL: http://arxiv.org/abs/1704.03084.

[22] Benjamin Recht et al. "Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent". In: *Advances in Neural Information Processing Systems 24*. Ed. by J. Shawe-Taylor et al. Curran Associates, Inc., 2011, pp. 693–701. URL: http://papers.nips.cc/paper/4390-hogwild-a-lock-free-approach-to-parallelizing-stochastic-gradient-descent.pdf.

[23] Tom Schaul et al. *Prioritized Experience Replay*. 2015. arXiv: 1511.05952 [cs.LG].

[24] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (2015), pp. 85–117. ISSN: 0893-6080. DOI: http://dx.doi.org/10.1016/j.neunet.2014.09.003. URL: http://www.sciencedirect.com/science/article/pii/S0893608014002135.

[25] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

[26] John Schulman et al. *Trust Region Policy Optimization*. 2015. arXiv: 1502.05477 [cs.LG].

[27] David Silver et al. "Deterministic Policy Gradient Algorithms". In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML'14. Beijing, China: JMLR.org, 2014, I–387–I–395.

[28] David Silver et al. "Mastering the Game of Go with Deep Neural Networks and Tree Search". In: *Nature* 529.7587 (Jan. 2016), pp. 484–489. ISSN: 0028-0836. DOI: 10.1038/nature16961.

[29] David Silver et al. "Mastering the game of Go without human knowledge". In: *Nature* 550 (Oct. 2017), pp. 354–. URL: http://dx.doi.org/10.1038/nature24270.

[30] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998. URL: http://www.cs.ualberta.ca/~sutton/book/the-book.html.

[31] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[32] Oriol Vinyals et al. *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/. 2019.

[33] Ziyu Wang, Nando de Freitas, and Marc Lanctot. "Dueling Network Architectures for Deep Reinforcement Learning". In: *CoRR* abs/1511.06581 (2015). arXiv: 1511.06581. URL: http://arxiv.org/abs/1511.06581.

[34] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning". In: *Machine Learning* 8.3 (May 1992), pp. 279–292. ISSN: 1573-0565. DOI: 10.1007/BF00992698. URL: https://doi.org/10.1007/BF00992698.

[35] Chao Yu, Jiming Liu, and Shamim Nemati. *Reinforcement Learning in Healthcare: A Survey*. 2019. arXiv: 1908.08796 [cs.LG].