UNDERSTANDING OPTIMIZATION *in* REINFORCEMENT LEARNING

An Empirical Study of
Algorithms and their Hyperparameters

Jan Ole von Hartz

May 2019

Submitted in partial fulfillment of the requirements
for the degree of Bachelor of Science

to the

Machine Learning Lab
Department of Computer Science
Technical Faculty
University of Freiburg

Work Period
28. 02. 2019 – 28. 05. 2019

Examiner
Prof. Dr. Frank Hutter

Supervisor
Raghu Rajan

DECLARATION

I hereby declare, that I am the sole author and composer of my thesis and that no other sources or learning aids,other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.I also hereby declare that my thesis has not been prepared for another examination or assignment, either in its entirety or excerpts thereof.

Place, date	Signature

This thesis was typest using the incredible template created and released into the public domain by Eivind Uggedal, found at https://github.com/uggedal/thesis.

Hereafter follow the original remarks to his thesis.

This thesis was typeset using the LaTeX typesetting system originally developed by Leslie Lamport, based on TeX created by Donald Knuth.

The body text is set 12/14.5pt on a 26pc measure with Minion Pro designed by Robert Slimbach. This neohumanistic font was first issued by Adobe Systems in 1989 and have since been revised. Other fonts include Sans and Typewriter from Donald Knuth's Computer Modern family.

Typographical decisions were based on the recommendations given in *The Elements of Typographic Style* by Bringhurst (2004).

The use of sidenotes instead of footnotes and figures spanning both the textblock and fore-edge margin was inspired by *Beautiful Evidence* by Tufte (2006).

The guidelines found in *The Visual Display of Quantitative Information* by Tufte (2001) were followed when creating diagrams and tables. Colors used in diagrams and figures were inspired by the *Summer Fields* color scheme found at http://www.colourlovers.com/palette/399372

ABSTRACT

In machine learning, praxis is by far more delicate than theory: not just different algorithms and random seeds, but also the usage of different optimizers and settings of their hyperparameters yield dramatically different results, to the point of convergence versus non-convergence. Deep reinforcement learning is especially difficult due to the strongly moving loss landscape, with agents not learning the desired behavior because of getting stuck in local optima. While there exists some rough ideas, the exact effects of different optimizers (such as Adam) and their hyperparameters on the returns of the agent, as well as the stability in training and evaluation, are not yet well understood. Using the well known DQN algorithm for discrete and DDPG algorithm for continuous environments, we investigate the influence of different optimizers and settings of their hyperparameters on the agents performance, as well as compare the optimizers' sensitivity to hyperparameters and their stability in training. Using BOBH, a hyperparameter optimization method, we further try to make a step towards hyperparameter agnostic deep reinforcement learning, to try to avoid the pitfalls of hyperparameter settings.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

TODO

Jan Ole von Hartz
Freiburg, Germany
May 2019

INTRODUCTION

1

According to Sutton and Barto (2018) the field of *reinforcement learning* studies learning "how to map situations to actions—so as to maximize a numerical reward signal", and refers - besides the field of study - to both to the problem class itself, as well as a class of solution methods. The commonly used formalism is that of an *agent* interacting with an *environment* via a fixed set of *actions* and only observing 1. a numerical reward signal and 2. the state of the environment or an observation which describes part of it.

1.1 FOCUS

In this thesis, we focus on the analysis of adaptive gradient-based optimization methods (Adam and AdamW) in deep reinforcement learning using well-established value-based (DQN) and actor-critic (DDPG) methods in a popular and computationally simple control environment (Cartpole), a popular and more complex set of video game environments (Atari Arcade Environment) and a set of of more challenging physical control tasks (Mujoco)[1]. We deem it important that our groundwork is generalized to more optimization methods, learning algorithms and environments, but we had to limit the scope of this thesis due to constraints on the volume of work.

1. For an introduction of and reference to all these methods and environemnts, see chapter 4.

1.2 MOTIVATION

Recent Progress In recent years, *reinforcement learning* hast not only seen a surge of research, but has also produced a number of remarkable results, such as agents learning to control a helicopter (Ng et al., 2006), locomotion tasks (Heess et al., 2017), in-hand manipulation of objects (OpenAI et al., 2018) and playing games[2] such backgammon (Tesauro, 1995), Atari video games (Mnih et al., 2013) and Go (Silver et al., 2016, 2017). Just during the creation of this thesis, a team of agents trained via reinforcement learning won a tournament against the 2018 world champions in the complex multiplayer online battle arena computer game DOTA 2 (Statt, 2019). All this is in part due to the increasing application of so called *deep reinforcement learning* techniques, the use of *deep neural networks* in reinforcement learning.

2. Often on what is called a "superhuman" level, generating a lot of media coverage for the field.

New Problems While allowing for great scientific progress, the use of deep neural networks also introduces many new and opaque problems, some of which we try to tackle in thesis. For one, solving machine learning problems usually involves the selection of a as well suited as possible function from a class of functions by minimizing some loss function defined over the output of said function for a set of data points. Hence, the training of a deep neural network has in its core a *numerical optimization problem*: finding a suited parametrization of the network to achieve the best possible performance (the lowest possible loss) on unseen data by using a given data set as an approximation of the general data distribution and optimizing the networks performance on it. This is usually done via *gradient-based methods*, such *gradient descent* and variations of it.

Trial-and-Error Reinforcement learning however introduces additional problems; unlike in *supervised learning*, the training data is not given, but must be produced by agent by engaging with the environment in a trial-and-error fashion. Since the deep neural networks are used by the agent to e.g. approximate a function that estimates the reward of a certain action in a certain state and these estimations change drastically over time, the numerical optimization of these networks is very unstable.

Local Optima A systematic problem of gradient-based methods is that unlike analytical methods they are generally[3] only suited to find local optima, not global optima. In reinforcement learning the loss landscape is usually very rough and provides plenty of local optima to get stuck in.

3. Global optima are only found for convex functions.

Methods and Hyperparameters Furthermore, it is not well understood, how different (adaptive) gradient-based methods[4] compare in reinforcement learning, especially since they are also somewhat sensitive to the settings of their hyperparameters. A common call thus is for the development of hyperparameter agnostic algorithms - algorithms that adapt their hyperparameters themselves during runtime (Henderson et al., 2018a).

4. Such as the popular Adam algorithm and AdamW, a variation of it.

Performance Estimation The automatic hyperparameter optimization introduces further challenges, since it requires for a cost-effective estimation of the performance of a given configuration - something that is not given in reinforcement learning, where the training of an agent is usually very costly[5].

5. Agents are often trained on millions of simulated timesteps using large computer clusters.

1.3 OBJECTIVE

Performance Estimation We will shine some light on these opaque problems of optimization in deep reinforcement learning by first inves-

tigating ways of reliable and cost-effective performance estimation of reinforcement learning agents as a foundation.

Comparison of Optimization Methods We will then compare different popular optimization methods in reinforcement learning to evaluate their respective performance and gain an understanding of their different characteristics, that is 1. the final performance of their produced network configuration[6] 2. the stability of these optima[7] 3. their sensitivity towards their hyperparameters and 4. their stability in training as measured by their inner state.

6. Behold the problem of **local optima** in reinforcement learning.

7. Connecting back to the problem of instable optimization due to **trial-and-error** training.

Towards Hyperparameter Agnosticism Using a hyperparameter optimization method we will automatically find well performing hyperparameter settings for the optimizers to make a step towards hyperparameter agnosticism and evaluate the practicality of this approach.

1.4 CONTRIBUTION

Cost-effective performance estimation is not only a preliminary for automatic hyperparameter setting, but will also help researchers and practitioners, especially with low computational budgets, to make early informed decisions and efficiently use their resources.

The analysis of the different optimization methods does not only deepen our understanding of the behavior of adaptive gradient-based methods in the strongly changing loss landscapes of deep reinforcement learning, but also helps with the selection of suitable algorithms and their hyperparameters.

The automatic optimization of these hyperparameters will help to advance towards the goal of a fully closed machine learning pipeline.

Generally, our results will help practitioners and researchers to focus on their core interest without the large burden that comes with the selection of optimization methods and their hyperparameters.

1.5 OUTLINE

This thesis consists of three parts.

The first one introduces necessary formalisms, concepts and establishes a consistent notation. It will do so in a logical order; starting with notation and basic theoretical notions, continuing with the realization and use of these concepts in the used algorithms, and closing with the tangible tools and computational libraries that were used[8].

8. This appraoch is motivated in the beginning of chapter 5; Henderson et al. (2018a) showed the huge influence of the code base on the final performance in deep reinforcement learning.

The second part presents the problem of performance estimation in deep reinforcement learning and the experiments we conducted.

The third part presents our comparison of the different optimizes in a practical reinforcement learning environment.

Afterwards, we will conclude our results and point out which future works needs to be done.

PART I

BACKGROUND

KEY CONCEPTS AND NOTATION

2

2.1 NOTATION

Whenever possible we distinguish between

- scalars: x
- vectors: $\vec{x}$
- matrices: X
- sets: $\mathcal{X}$
- (time) series $(\chi_n)_{n=0,1,\dots}$

For a vector $\vec{x}$ the i-th element is denoted as x_i. For a matrix X, $\vec{x}_i$ denotes the i-th column and x_{ij} the j-th value of this column. The value of a time series $(\chi_n)_{n=0,1,\dots}$ at point t is denoted as χ_t.

2.2 KEY CONCEPTS

To fully understand the content of this thesis, the reader should be familiar with some key concepts of *deep learning* - machine learning using *deep neural networks*. We will briefly reiterate the most important concepts to accomplish a common ground in notation. Unfamiliar readers are advised to revise these concepts, e.g. with Goodfellow et al. (2016). Proficient readers can safely skip this chapter. Necessary concepts from *reinforcement learning* and *deep reinforcement learning* will be introduced in the next chapter. Debutants can deepen their understanding, e.g. in Sutton and Barto (2018).

Supervised Learning Given some function class $\mathcal{G} : \mathbf{R}^d \to \mathbf{R}^o$ and a matrix of training data $X \in \mathbf{R}^{d \times n}$ with targets $Y \in \mathbf{R}^{\mathbf{o} \times \mathbf{n}}$, that are sampled from some generating data distribution X_{Gen}, Y_{Gen}, an algorithm tries to find the function $g \in \mathcal{G}$ that best describes the relation of data points $\vec{x}_i \in X_{Gen}$ with the matching targets $\vec{y}_i \in Y_{Gen}$ by using the training data and targets as an approximation of the general data distribution and minimizing some **loss function** $L : \mathbf{R}^{o \times 2} \to \mathbf{R}$ on the set of pairs of predictions $g(\vec{x}_i)$ on the training data, and training targets $\vec{y}_i$. One commonly used loss functions is the **mean squared error** $L(\vec{x}_j, \vec{y}_j) = \frac{1}{n} \sum_{i=0}^{n} (x_{ij} - y_{ij})^2$, where $n = |a| = |b|$. The learning is usually done by

defining $\mathcal{G}$ as a set of functions $g_{\vec{\theta}}$ with parameters $\vec{\theta}$ and numerically optimizing these parameters.

(Mini-) Batch Since the computation of the loss on the whole training set at once it not only costly but also not necessarily effective (Keskar et al., 2016), while the computation on a single data point leads to very noisy estimates of the gradients in numerical optimization, a common approach is to divide the training data into or to sample *(mini-) batches* from the training data and compute the numerical updates on these instead of on the full data set.

Overfitting is what happens when a machine learning model fits the training distributions more closely than justified by the underlying generating data distribution, leading to a worse generalization performance.

Gradient Descent is a numerical optimization method, suited to find local minima of some differentiable function $f : \mathbf{R}^n \to \mathbf{R}$ and global minima if f is *convex*[1]. Starting at some point $\vec{x}_0$ and with some *learning rate* α, gradient descent iteratively computes updates $\vec{x}_t = \vec{x}_{t-1} - \alpha \cdot \nabla f(\vec{x}_{t-1})$ until some stopping criterion (e.g. convergence) is reached.

1. Convexity is usually not given, but local optima give decent approximations and might overfit less.

Stochastic Gradient Descent is a stochastic approximation of gradient descent, commonly applied in machine learning. The gradients of the loss function for the parameter updates are computed on a randomly sampled minibatch, often leading to faster convergence.

(Deep) Neural Network are non-linear function approximators commonly used in recent machine learning research. They are constructed by stacking layers of nodes (neurons) that each 1. compute some linear combination of the values of the nodes of the previous layer and 2. apply some non-linear activation function. The parameters of the network (the weights and biases of the linear functions) are denoted by $\vec{\theta}$. A prediction for some data vector is made by feeding the latter into the first layer (the input layer) of the network and computing the activation of the last layer (the output layer). This is called the *forward pass*. The value of the output layer of the network f with parameters $\vec{\theta}$ for some input $\vec{x}$ is denoted as $f(\vec{x}, \vec{\theta})$. Supervised training of the network is commonly achieved by calculating the average derivative $\frac{1}{n} \sum_{i=0}^{n} \nabla_{\vec{\theta}} L(\vec{y}_i, f(\vec{x}_i, \vec{\theta}))$ of some loss function L, with respect to the network parameters $\vec{\theta}$ on some (mini-) batch X with size n and targets Y, (called *backward pass*) and applying updates to $\vec{\theta}$ via *(stochastic) gradient descent*.

Observability In artificial intelligence research, an environment is called *fully observable*, if an agent that interacts with it, has full access to all relevant information about its inner state, *partly observable* else.

REINFORCEMENT LEARNING

As introduced, problems in reinforcement learning are usually framed as an *agent* interacting with an *environment* via a fixed set of *actions* and only observing 1. a numerical reward signal and 2. the state of the environment or an observation which describes part of it. This chapter introduces the mathematical formalisms and the theoretical ideas behind their practical application.

3

3.1 THE MATHEMATICAL MODEL

In large parts of the current research, as well as this thesis, instances of this problem class are formalized using *Markov decision processes* (Howard, 1960). Hence, we now define this and related notions.

3.1.1 Markov Decision Processes and Policies

Definition 1 (Markov Process) *Let $\tau = 0, 1, \ldots$ be a (possibly infinite) series of discrete time steps and $t \in \tau$. A* Markov process [1] *is a 2-tuple $(\mathcal{S}, \mathcal{P})$, where $\mathcal{S}$ is a finite [2] set of states and $\mathcal{P}$ is a state transition function $\mathcal{P}(s, s') = \mathbf{P}[s_t = s' \mid s_{t-1} = s]$, that fulfills the* Markov property*, i.e. for all states and at all times $\mathbf{P}[s_t \mid s_{t-1}, s_{t-2}, \ldots, s_0] = \mathbf{P}[s_t \mid s_{t-1}]$.* [3]

1. *Also often called a* Markov chain.

2. *We do not need the notion of infinte Markov processes here.*

3. *Intuitvely: "the future is independent of the past, given the present".*

Definition 2 (Markov Decision Process) *A* Markov decision process *is a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $(\mathcal{S}, \mathcal{P})$ is a* Markov process *with transition function $\mathcal{P}_a(s, s') = \mathbf{P}[s_t = s' \mid s_{t-1} = s, a_{t-1} = a]$ (where $a \in \mathcal{A}$), $\mathcal{A}$ is a finite set of actions, $\gamma \in [0, 1]$ is a discount factor and $\mathcal{R}$ is the expected immediate reward $\mathcal{R}_a(s, s') = \mathsf{E}[R(s_{t-1}, a_{t-1}) \mid s_{t-1} = s, s_t = s', a_{t-1} = a]$, with R being the reward received at point t, defined as a function mapping s_{t-1} and a_{t-1} to some real number.* [4]

4. *Also often defined as a function of only s_{t-1} or even s_{t-1}, a_{t-1} and s_t, depending on the problem at hand.*

Definition 3 (Policy) *A* policy *π is a mapping from a state space $\mathcal{S}$ to a set of probability distributions over an action space $\mathcal{A}$, given some state $s \in \mathcal{S}$: $s \mapsto p(\mathcal{A} = a \mid s)$.*

Reinforcement learning problems are then often modeled as interpreting a *Markov decision process M* as an environment in which an Agent *A* can take actions $a \in \mathcal{A}_M$ and observe the ensuing reward $r \in \mathbf{R}$ and state $s \in \mathcal{S}_\mathcal{M}$. The agent's goal then usually is to find an optimal policy $\pi^\star$ to maximize some cumulative function, usually the total discounted

reward $\mathrm{R} = \sum_{t=0}^{h-1} \gamma^t \cdot R(s_t, a_t)$, where $h \in \mathbf{N} \cup \{\infty\}$ is called the *horizon* of the problem. [5]

5. The horizon can be infinite.

Definition 4 (Trajectory) *Let $h \in \mathbf{N} \cup \{\infty\}$ be a (possibly infinite) Horizon, $\tau = 0, 1, \ldots, H$ a series of discrete time steps, $\mathcal{A}$ a set of actions, $\mathcal{S}$ a set of states and $\mathcal{R} \subseteq \mathbf{R}$ a set of reward signals. A trajectory then is a series $\rho = s_o, a_0, r_0, s_1, a_1, r_1, \ldots, s_{h-2}, a_{h-2}, r_{h-2}, s_{h-1}$, with $s_i \in \mathcal{S}$, $a_i \in \mathcal{A}$ and $r_i \in \mathcal{R}$ for all $i \in \tau$. A trajectory is called infinite, if h is infinite.* [6]

6. *Practically we will only concern ourselves with finite trajectories.*

The notion of the trajectory is literally used in reinforcement learning to describe the interaction of an agent with an environment over time.

3.1.2 *Value Functions and the Bellmann Equation*

Notably, a policy implies a transition distribution over the state space of a Markov decision process M. This property allows one to easily evaluate a given policy π on M, by reducing the problem of evaluation to the evaluation of a Markov process, like the following definition illustrates.

Definition 5 (State-Value Function) *The* state-value function *(often called V-function) $V^\pi(s)$ of a state $s \in \mathcal{S}$, given a policy π, is the expected return gained when starting in s and following π henceforth: $V^\pi(s) = \mathsf{E}_\pi[\sum_{t=0}^{T-1} \gamma^t \cdot R(s_t, a_t) \mid s_0 = s]$.*

Definition 6 (State-Action-Value Function) *The* state-action-value function *(often called Q-function) $Q^\pi(s, a)$ of a state $s \in \mathcal{S}$ and an action $a \in \mathcal{A}$, given a policy π, is the expected return gained when starting in s, taking action a and following π thereafter: $Q^\pi(s, a) = \mathsf{E}_\pi[\sum_{t=0}^{T-1} \gamma^t \cdot R(s_t, a_t) \mid s_0 = s, a_0 = a]$.*

Theorem 1 (Bellmann Equation) *The* state-action-value function *fulfills the following recursive property:* [7]

7. *A similar property holds for the* action-value function.

$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_a(s, s') \sum_{a' \in \mathcal{A}} \pi(a' \mid s') \cdot Q^\pi(s', a')$$

Value Iteration This famous property allows to, given the reward function R, iteratively compute the Q-function or V-function for a finite state space, a finite action space and a given policy by initializing all values as zero and then repeatedly applying the Bellmann equation until the values converge (Bellman, 1957).

Policy Iteration Remember, that the goal for the agent is to learn the optimal policy $\pi^\star$, which corresponds to an optimal *state-action-value function* $Q^\star$. The optimal policy $\pi^\star$ is the one that maximizes the corresponding Q-function $Q^\star$. As the Bellmann equation illustrates, maximizing Q under policy π is achieved by greedily selecting the action a in state s that maximizes $R(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_a(s, s') \sum_{a' \in \mathcal{A}} \pi(a' \mid s') \cdot Q^\pi(s', a')$,

yielding an improved policy π': $\pi'(s,a) = \begin{cases} 1 & , \text{if } a = \arg\max_{\hat{a} \in \mathcal{A}} Q(s,\hat{a}) \\ 0 & , else \end{cases}$

Hence, just as $Q^\star$ can iteratively be computed, $\pi^\star$ can be iteratively computed as well, e.g. via *dynamic programming* (Howard, 1960).

3.2 APPLICATION OF THE MODEL

These lessons learned in the exact solution of the optimal policy via dynamic programming, can now also be applied to approximately solving this problem, as usually done in current reinforcement learning research. For this, we present two approaches that the algorithms introduced later make use of: *value base methods* and *actor-critic methods*.

3.2.1 Value Based and Actor-Critic Methods

Valued Based Methods To learn an approximation to the optimal policy $\pi^\star$ in an environment M, an agent A can thus estimate $Q^\star$ via arbitrarily initializing its estimate $\hat{Q}$ and then, using the reward signals, provided by the environment in response to the its actions [8], iteratively improve its estimates of $Q^\star$ via *value iteration*, which is at the heart of so called *value based methods*, such as *Q-learning* (Watkins and Dayan, 1992).

8. In order to get a decent estimate of $Q^\star$, the agent has to engage in a trade-off between taking the greedily selected action and exploring new actions, as well be discussed in chaper 2.

Actor-Critic Methods As the selection of the policy often happens in a greedy manner, all possible actions must be considered, which is only feasible for action spaces that are discrete and finite. For other action spaces, agents can, for example, separately estimate both the optimal V-function and the optimal policy (instead of producing the current policy via greedy selection from the current estimation of Q), which is at the heart of so called *actor-critic methods*.

Other Methods Another issue with continuous action spaces is that the runtime of the iterative update of the Q-function is at least linearly dependent on the size of the action space (see theorem 1) . This can be solved by approximating the updates of the Q-function via *Monte Carlo sampling* from the action space, which we will not discuss. Another common approach is to artificially discretize a continuous action space.

3.2.2 Practicality of the Markov Property

As noted by Arulkumaran et al. (2017), the underlying assumption of the *Markov Property* does often not hold in practical applications, since it requires full observability of the states. While algorithms that use *partly observable Markov decision processes* exist, full observability is often simply assumed as an approximation of the actual partly observable environment. An example of such an approximation can be found in

Mnih et al. (2013): using the video output of an Atari game simulator, the authors preprocessed the observations by combining four consecutive video frames into a single observation to approximate state features such as the velocity of in-game objects.

3.3 SCOPE OF THIS THESIS

In this thesis we will examine *deep reinforcement learning* algorithms based on *value based methods* and *actor-critic methods*.

We will now introduce the used algorithms and explain how they relate to the introduced formalisms and methods in order of increasing abstraction level.

ALGORITHMS AND METHODS

4

4.1 NUMERICAL OPTIMIZATION

Deep reinforcement learning usually involves the *numerical optimization* of one or more *deep neural networks* as function approximators with regards to some loss function. Numerical optimization in deep reinforcement learning usually makes use of *gradient-based methods*, since gradients can be easily computed for deep neural networks, while analytical solutions are infeasible and higher order moments very expensive to calculate. However, naive *stochastic gradient descent* is generally not the preferred method, as more advanced first-order methods that estimate higher order moments of the target function can yield significantly faster convergence without adding a lot of computational burden. We will now introduce the used optimization methods.

4.1.1 Adam: Adaptive Moment Estimation

Method

Adam (Kingma and Ba, 2014) is a first-order gradient-base optimization methods that computes "individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients" (Kingma and Ba, 2014). It does so by calculating exponential moving averages of the gradient and squared gradient of each parameter and using them to determine the individual learning rates. The individual steps of Adam can be seen in Algorithm 1. Until reaching some stopping criterion (e.g. convergence), in each time step Adam computes the current gradients, which in machine learning is usually done on some minibatch, updates its moment estimates and uses them to update the parameters it optimizes.

Popularity

Kingma and Ba (2014) have not only shown theoretical convergence properties of Adam, but also demonstrated for a range of machine learning tasks, including deep neural networks, that Adam performs en par with or even outperforms similar methods, such as RMSProp (Tieleman and Hinton, 2012) in terms of convergence of the training cost. Adam hat hence quickly gained in popularity. According to a survey conducted

Hyperparameters: step size $\alpha \in \mathbf{R}$, decay rates $\beta_1, \beta_2 \in \mathbf{R}$, division stabilization constant $\epsilon > 0$

Given: initial parameter vector $\vec{\theta}_0 \in \mathbf{R}$, objective function series $\left(f_n\left(\cdot, \vec{\theta}\right)\right)_{n=0,1,\ldots}$

Optional: schedule multiplier series $(\eta_n)_{n=0,1,\ldots}$

Init: $t \leftarrow 0$; $\vec{m}_0, \vec{v}_o \leftarrow \vec{0}$; `(time step and moment vectors)`

while *stopping criterion not met* **do**

  $t \leftarrow t + 1$;

  $\vec{g}_t \leftarrow \nabla_{\vec{\theta}} f_t(\vec{m}, \vec{\theta}_{t-1})$; `(computed on minibatch` $\vec{m}$`)`

  $\vec{m}_t \leftarrow \beta_1 \cdot \vec{m}_{t-1} + (1 - \beta_1) \cdot \vec{g}_t$; `(moment estimation)`

  $\vec{v}_t \leftarrow \beta_2 \cdot \vec{v}_{t-1} + (1 - \beta_1) \cdot \vec{g}_t^2$;

  $\hat{\vec{m}}_t \leftarrow \vec{m}_t / (1 - \beta_1^t)$; `(zero-bias correction)`

  $\hat{\vec{v}}_t \leftarrow \vec{v}_t / (1 - \beta_2^t)$;

  $\vec{\theta}_t \leftarrow \vec{\theta}_{t-1} - \eta_t \cdot \alpha \cdot \hat{\vec{m}}_t / (\sqrt{\hat{\vec{v}}_t} + \epsilon)$; `(parameter update)`

end

return θ_t

Algorithm 1: Adam. Until some predefined stopping criterion (e.g. convergence) is met, the algorithm repeats the following steps; computation of the gradients on a minibatch, update of the moment estimates and bias-correction, update of the parameters. The bias correction is introduced due to the zero-initialization of the moment estimates, see Kingma and Ba (2014). Although not proposed by Kingma and Ba (2014), Adam can also be combined with learning rate scheduling, see e.g. Loshchilov and Hutter (2017), introducing an additional factor η_t in the parameter update.

by Karpathy (2017) of 28,303 machine learning papers published on arxiv.org between 2012 and 2017, Adam was used in about one in four papers, making it the most popular optimization method by far.

Limitations

Despite its advantages and popularity, Wilson et al. (2017) found that Adam and other adaptive methods tend to generalize worse than stochastic gradient descent and provided examples on which adaptivity leads to overfitting. E.g. Reddi et al. (2018) have since proposed extensions of Adam to combat its shortcomings.

Open Questions

Despite these ongoing developments, the exact properties of Adam and the influence of its hyperparameters are not yet well understood. Heusel et al. (2017) sow the seeds by investigating Adam for *generative adversarial networks* and characterizing it as a "heavy ball with friction" - prefer-

ring flat minima over steeper ones. Henderson et al. (2018b) compared different optimizers in a reinforcement learning setting and found that "adaptive optimizers have a narrow window of effective learning rates, diverging in other cases, and that the effectiveness of momentum varies depending on the properties of the environment". These findings seem contradictory to us, as flat minima are typically thought of to be more robust than steeper ones (Hochreiter and Schmidhuber, 1997; Keskar et al., 2016) and Adam is advertised to be more robust against settings of its learning rate exactly due to the individual adaptation (Kingma and Ba, 2014). Dinh et al. (2017) also challenged the idea of flat minima generalizing better. We therefore want to continue in this line of research by investigating Adam's detailed behavior in deep reinforcement learning and explore whether hyperparameter optimization methods can solve the problems found by Henderson et al. (2018b).

Loshchilov and Hutter (2017) found another reason for the poor generalization of Adam. In practice gradient descent is often used in conjunction with L_2*-regularization* or *weight decay* (Krogh and Hertz, 1992), two popular methods to regularize network parameters to avoid overfitting. While both are equivalent for stochastic gradient descent, they are not for Adam. And while naive implementations often just apply L_2-regularization, Adam benefits far more from weight decay.

4.1.2 *AdamW: Adam with Weight Decay*

Method

While in vanilla stochastic gradient descent the update rule of the optimized parameters $\vec{\theta}$ with learning rate α for time step t on minibatch $\vec{m}$ is defined as

$$\vec{\theta}_t = \vec{\theta}_{t-1} - \alpha \cdot \nabla_{\vec{\theta}} f_t(\vec{m}, \vec{\theta}_{t-1}) \tag{4.1}$$

weight decay introduces an additional factor λ that with time exponentially decays the parameter values to regularize their growth. We follow the notation of Loshchilov and Hutter (2017) here:

$$\vec{\theta}_t = (1 - \lambda) \cdot \vec{\theta}_{t-1} - \alpha \cdot \nabla_{\vec{\theta}} f_t(\vec{m}, \vec{\theta}_{t-1}) \tag{4.2}$$

Combining this technique with the Adam algorithm (see algorithm 1), gives algorithm 2, called *AdamW* by Loshchilov and Hutter (2017). The authors showed that while for stochastic gradient descent L_2-regularization and weight decay are equivalent up to rescaling of the learning rate, this is not the case for Adam. This can easily be seen intuitively by changing line 3 in algorithm 1 to

$$\vec{g}_t \leftarrow \nabla_{\vec{\theta}} f_t(\vec{m}, \vec{\theta}_{t-1}) + \boxed{\lambda \cdot \theta_{t-1}} \tag{4.3}$$

as would to the case for L_2-regularization. Since the gradient is used afterwards for the moment estimation, $\lambda \cdot \theta_{t-1}$ has a non-linear influence on the parameter update. See Loshchilov and Hutter (2017) for a formal prove.

Hyperparameters: step size $\alpha \in \mathbf{R}$, decay rates $\beta_1, \beta_2 \in \mathbf{R}$,
division stabilization constant $\epsilon > 0$,
weight decay factor $\lambda \in [0, 1)$

Given: initial parameter vector $\vec{\theta}_0 \in \mathbf{R}$,
objective function series $\left(f_n\left(\cdot, \vec{\theta}\right)\right)_{n=0,1,\ldots}$

Optional: schedule multiplier series $(\eta_n)_{n=0,1,\ldots}$

Init: $t \leftarrow 0$; $\vec{m}_0, \vec{v}_o \leftarrow \vec{0}$; `(time step and moment vectors)`

1 **while** *stopping criterion not met* **do**

2 $\quad t \leftarrow t + 1$;

3 $\quad \vec{g}_t \leftarrow \nabla_{\vec{\theta}} f_t(\vec{m}, \vec{\theta}_{t-1})$; `(computed on minibatch` $\vec{m}$`)`

4 $\quad \vec{m}_t \leftarrow \beta_1 \cdot \vec{m}_{t-1} + (1 - \beta_1) \cdot \vec{g}_t$; `(moment estimation)`

5 $\quad \vec{v}_t \leftarrow \beta_2 \cdot \vec{v}_{t-1} + (1 - \beta_1) \cdot \vec{g}_t^2$;

6 $\quad \hat{\vec{m}}_t \leftarrow \vec{m}_t / (1 - \beta_1^t)$; `(zero-bias correction)`

7 $\quad \hat{\vec{v}}_t \leftarrow \vec{v}_t / (1 - \beta_2^t)$;

8 $\quad \vec{\theta}_t \leftarrow \boxed{(1 - \lambda)} \cdot \vec{\theta}_{t-1} - \eta_t \cdot \alpha \cdot \hat{\vec{m}}_t / (\sqrt{\hat{\vec{v}}_t} + \epsilon)$; `(parameter update)`

9 **end**

10 **return** θ_t

Algorithm 2: AdamW. Yielded by combining Adam (see algorithm 1) with weight decay (see equation 4.2).

Properties

Generalization The authors conducted experiments on a number of tasks in a *supervised* machine learning setting and found that the usage of weight decay drastically improves Adams generalization performance, while still allowing the further improvements via the use of *scheduled learning rate multipliers*. Zhang et al. (2018) confirmed these results and offered mechanisms by which these improvements are achieved.

Decoupling of Parameters Loshchilov and Hutter (2017) further showed how the usage of weight decay over L_2-regularization decouples the learning rate and regularization hyperparameter. This is an incredibly valuable lesson. Recall that Henderson et al. (2018b) found a strong dependence of the performance of reinforcement learning algorithms on the hyperparameter settings of their optimizers. Henderson et al. (2018a) thus called for the development of what they called *hyperparameter agnostic* algorithms - algorithms that adjust their hyperparameters during training. Tuning of hyperparameters is a lot easier if they can be tuned mostly independently of each other without strong cross-interaction effects. Thus, AdamW seems like an important step towards achieving this goal.

4.2 AGENTS

4.2.1 DQN: Deep Q-Networks

DQN is a value-based deep reinforcement method for discrete and finite action spaces, first proposed by Mnih et al. (2013) for the use in the *Arcade Learning Environment* (Bellemare et al., 2013), an Atari game simulator.

Basic Idea

DQN works by approximating the optimal Q-function $Q^\star$ of a given state space and action space with a deep neural network with parameters $\vec{\theta}$ that is trained using a variation of Q-learning. It estimates the optimal policy $\pi^\star$ via greedily selecting the action a in state s that maximizes its current estimate of the optimal Q-function.

The agent observes preprocessed video inputs: the raw video frames are converted from RGB values to gray-scale, down sampled and cropped to reduce the input dimensionality and thus the computational burden. Finally, four consecutive of the so produced frames are stacked to produce one observation, as to approximate full observability [1] of the environment (such as the velocity of in-game objects). This observation ϕ_t is then treated as a full state description s_t.

1. Recall that full observability is a prerequisite for a model using a Markov decision process.

The reward given to the agent to maximize, is the raw game score. At each simulated time step t, the old and new observation are bundled together with the chosen action and received reward in a *transition* $(\phi_t, a_t, r_t, \phi_{t+1})$. Additionally, the agent receives a signal when the game terminates.

Instability and Convergence in Training

As the agent learns an approximation of the optimal policy $\pi^\star$ via approximating the corresponding optimal Q-function $Q^\star$, its current estimation of $\pi^\star$ directly depends on its current estimation of $Q^\star$. Therefor the estimation of $Q^\star$ is very unstable (meaning volatile), as a change in the estimation of $Q^\star$ can trigger a change in the estimation of $\pi^\star$ and vice versa, leading to chain reactions. This is especially grave, inasmuch as that both estimates are updated at the same time, whereas the classical value iteration algorithm, that inspires value-based methods, only updates its policy once the value estimates have converged. DQN thus deploys two techniques to stabilize training (reduce volatility) and speed up convergence of the estimates: *off-policy learning* and *experience replay*.

Off-Policy Learning As DQN does not solve $Q^\star$ exactly, but merely estimates it, while at the same time already trying to maximize its rewards, there are two competing goals: the exploration of the action space

in order to find better action sequences and the exploitation of what is currently considered the best action. This is known as the *exploration-exploitation trade-off*. DQN applies a technique called *off-policy learning* to solve this problem: with some small probability ϵ it will choose a random action, while else taking the action currently favored by the greedy selection.

Experience Replay To tame rapid self-reinforcing changes in the estimation of $Q^\star$, DQN does not update $\vec{\theta}$ using the latest transition information, but maintains a *replay memory* in which all observed transitions are stored and for each update step a set of transitions is sampled at random, ensuring that they represent a diverse part of the past experience and smoothing out the updates of $\vec{\theta}$.

Learning of the Q-Function Recall from chapter 3.1.2 that the transition function of the underlying Markov process together with a policy implies a transition distribution over the state space. This is related to what Mnih et al. (2013) call the *behavior distribution* [2] $\rho(s, a)$ - a probability distribution over states s and actions a, that can be used to calculate the expected squared error between the target network and the current estimation. As the agents estimates $Q^\star(s, a)$ via a deep neural network $Q(s, a; \vec{\theta})$ with parameters $\vec{\theta}$, in each time step it needs to minimize a loss function (the expected squared error):

2. Although Mnih et al. (2013) basically assume a deterministic transition function, further simplyfing the problem.

$$L(\vec{\theta}) = \mathsf{E}_{s,a\sim\rho}[(y_i - Q(s, a, \vec{\theta}))^2]$$

with targets [3] y_i (recall the transition distribution $\mathcal{P}$, reward function R and discount factor γ from the definition of Markov decision processes):

3. See how these targets echo the Bellman equation.

$$y_i = \mathsf{E}_{s'\sim\mathcal{P}_a(s)}[R(s, a) + \gamma \cdot \max_{a'} Q(s', a'; \vec{\theta}) \mid s, a]$$

As introduced, in deep reinforcement learning optimization is solved numerically. Calculating the gradient of L with respect to $\vec{\theta}$ yields:

$$\nabla_{\vec{\theta}} L(\vec{\theta}) = \mathsf{E}_{s,a\sim\rho,s'\sim\mathcal{P}_a(s)}\left[\left(R(s, a) + \gamma \cdot \max_{a'} Q(s', a'; \vec{\theta}) - Q(s, a; \vec{\theta})\right) \cdot \nabla_{\vec{\theta}} Q(s, a; \vec{\theta})\right]$$

Finally, as deep neural networks are trained using *stochastic* gradient descent, for some minibatch $\vec{m}$ with size k, consisting of transitions $(\phi_i, a_i, r_i, \phi'_i)$, we get (assuming $\phi_i \equiv s_i$):

$$\nabla_{\vec{\theta}} L(\vec{\theta}, \vec{m}, \vec{y}) = \frac{1}{k} \sum_{i=0}^{k} \left(\underbrace{R(\phi_i, a_i) + \gamma \cdot \max_{a'} Q(\phi'_i, a'; \vec{\theta})}_{=y_i} - Q(\phi_i, a_i; \vec{\theta}) \right) \cdot \nabla_{\vec{\theta}} Q(\phi_i, a_i; \vec{\theta}) \quad (4.4)$$

Algorithm 3 shows how this estimation of the Q-function is combined with the experience replay and the off-policy learning to form DQN.

Extensions

Researchers were quick to propose and evaluate a number of extensions to DQN, such as *prioritized experience replay* (Schaul et al., 2015), *Dueling DQN* (Wang et al., 2015), *Double Q-Learning* (Van Hasselt et al., 2016), *Distributional DQN* (Bellemare et al., 2017), *Noisy DQN* (Fortunato et al., 2017) and *Rainbow* Hessel et al. (2018), a combination of the above mentioned, yielding significantly improved performances. Whereas we are mainly interested in the effects of the optimizers and thus focus on the original DQN.

4.3 HYPERPARAMETER OPTIMIZATION

As shown by Henderson et al. (2018b), reinforcement leaning algorithms are sensitive towards the setting of their hyperparameters, such as the learning rates. The authors found this to still hold true for adaptive learning rate algorithms like Adam, which are thought of being more robust due to them finding flatter minima (Heusel et al., 2017), which were traditionally considered to generalize better (Hochreiter and Schmidhuber, 1997; Keskar et al., 2016).

Unfortunately, it is common practice in reinforcement learning literature to tune hyperparameters without comprehensive reasoning, leading to hardly comparable results (Henderson et al., 2018a). It is not uncommon to find sentences like "Specifically, we evaluated two hyperparameters over our five training games and choose the values that performed best. The hyperparameter values we considered were [...] $\epsilon_{adam} \in \{1/L, 0.1/L, 0.01/L, 0.001/L, 0.0001/L\}$" (Bellemare et al., 2013). As the alert reader will remember, in Adam ϵ is simply a constant to stabilize the division by zero. To us, even tuning this constant at all seems absurd.

Hyperparameters: number of episodes m,
number of time steps per episode l,
size of replay memory n,
off-policy probability ϵ,
minibatch size b

Given: starting observation σ_0, preprocessing function ϕ,
Network Q with parameters $\vec{\theta}$,
a Markov decision process [4] $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$

Init: $\vec{\theta}$ randomly, replay memory $\mathcal{D}$ with size n

foreach *episode* $= 0 \ldots, m-1$ **do**
  $\kappa_0 \leftarrow \sigma_0$; (initialize history)
  $\phi_0 \leftarrow \phi(\kappa_o)$; (preprocessed history)
  foreach $t = 1, \ldots, l$ **do**
    sample $x \in X \sim B(\epsilon)$; (Bernoulli distributed)
    if $x = 1$ **then**
      select a_t uniformly at random ; (off-policy)
    else
      $a_t \leftarrow \arg\max_a Q(\phi(s_t), a; \vec{\theta})$; (on-policy)
    end
    $(r_t, \sigma_{t+1}) \leftarrow \text{execute}(a_t)$;
    $\kappa_{t+1} = \kappa_t, a_t, \sigma_{t+1}$;
    $\phi_{t+1} = \phi(\kappa_{t+1})$;
    $\mathcal{D}$.add($(\phi_t, a_t, r_t, \phi_{t+1})$);
    $\vec{m} \leftarrow \mathcal{D}$.sample_minibatch($b$);
    foreach $(\phi_i, a_i, r_i, \phi_{i+1}) \in \vec{m}$ **do**
      if *terminal*(ϕ_{i+1}) **then**
        $y_i \leftarrow r_i$
      else
        $y_i \leftarrow r_i + \gamma \cdot \max_{a'} Q(\phi_{i+1}, a'; \vec{\theta})$
      end
    end
    $\vec{\theta} \leftarrow \text{gradient\_descent\_step}(\vec{\theta}, \vec{m}, \vec{y})$; (see eq. 4.4)
  end
end

Algorithm 3: Deep Q-Learning with Experience Replay. [5] For each training episode a number of training time steps will be executed. Each of these consists of: randomly determining whether to take an off-policy action or not, applying that action, storing the transition in the replay memory, sampling a minibatch from the latter and making one gradient step.

4. Or an approximation of one as, again, the Markov property is just assumed as an approximation.

5. Note, how the history of the agent is related to the notion of the trajectory.

TOOLS AND LIBRARIES

Ref to henderson about code base influence etc.

5

5.1 NUMERICAL OPTIMIZATION

5.2 AGENTS

5.3 ENVIRONMENTS

5.4 HYPERPARAMETER OPTIMIZATION

5.5 VISUALIZATION AND EVALUATION

PART II

PERFORMANCE ESTIMATION

PROBLEM

6

EXPERIMENTAL SETUP

7

RESULTS

8

DISCUSSION

9

PART III

OPTIMIZER ANALYSIS

PROBLEM

10

EXPERIMENTAL SETUP

11

RESULTS

12

DISCUSSION

13

PART IV

SUMMARY

CONCLUSION

None

14

BIBLIOGRAPHY

Arulkumaran, Kai; Deisenroth, Marc Peter; Brundage, Miles; and Bharath, Anil Anthony. 2017. *A brief survey of deep reinforcement learning*. In arXiv preprint arXiv:1708.05866. Cited on p. 11.

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. jun 2013. *The Arcade Learning Environment: An Evaluation Platform for General Agents*. In Journal of Artificial Intelligence Research, vol. 47, pp. 253–279. Cited on pp. 17 and 19.

Bellemare, Marc G; Dabney, Will; and Munos, Rémi. 2017. *A distributional perspective on reinforcement learning*. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pp. 449–458. JMLR. org. Cited on p. 19.

Bellman, Richard. 1957. *A Markovian decision process*. In Journal of Mathematics and Mechanics, vol. 6, no. 5, pp. 679–684. Cited on p. 10.

Bringhurst, Robert. October 2004. *The elements of typographic style*. Hartley & Marks Publishers, Point Roberts, WA, USA, 3rd edn. ISBN 0-881-79205-5. Cited on p. d.

Dinh, Laurent; Pascanu, Razvan; Bengio, Samy; and Bengio, Yoshua. 2017. *Sharp minima can generalize for deep nets*. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pp. 1019–1028. JMLR. org. Cited on p. 15.

Fortunato, Meire; Azar, Mohammad Gheshlaghi; Piot, Bilal; Menick, Jacob; Osband, Ian; Graves, Alex; Mnih, Vlad; Munos, Remi; Hassabis, Demis; Pietquin, Olivier; et al. 2017. *Noisy networks for exploration*. In arXiv preprint arXiv:1706.10295. Cited on p. 19.

Goodfellow, Ian; Bengio, Yoshua; and Courville, Aaron. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org. Cited on p. 7.

Heess, Nicolas; Sriram, Srinivasan; Lemmon, Jay; Merel, Josh; Wayne, Greg; Tassa, Yuval; Erez, Tom; Wang, Ziyu; Eslami, SM; Riedmiller, Martin; et al. 2017. *Emergence of locomotion behaviours in rich environments*. In arXiv preprint arXiv:1707.02286. Cited on p. 1.

Henderson, Peter; Islam, Riashat; Bachman, Philip; Pineau, Joelle; Precup, Doina; and Meger, David. 2018a. *Deep reinforcement learning that matters*. In Thirty-Second AAAI Conference on Artificial Intelligence. Cited on pp. 2, 3, 16, and 19.

Henderson, Peter; Romoff, Joshua; and Pineau, Joelle. 2018b. *Where did my optimum go?: An empirical analysis of gradient descent optimization in policy gradient methods*. In arXiv preprint arXiv:1810.02525. Cited on pp. 15, 16, and 19.

Hessel, Matteo; Modayil, Joseph; Van Hasselt, Hado; Schaul, Tom; Ostrovski, Georg; Dabney, Will; Horgan, Dan; Piot, Bilal; Azar, Mohammad; and Silver, David. 2018. *Rainbow: Combining improvements in deep reinforcement learning*. In Thirty-Second AAAI Conference on Artificial Intelligence. Cited on p. 19.

Heusel, Martin; Ramsauer, Hubert; Unterthiner, Thomas; Nessler, Bernhard; and Hochreiter, Sepp. 2017. *Gans trained by a two time-scale update rule converge to a local nash equilibrium*. In Advances in Neural Information Processing Systems, pp. 6626–6637. Cited on pp. 14 and 19.

Hochreiter, Sepp and Schmidhuber, Jürgen. 1997. *Flat minima*. In Neural Computation, vol. 9, no. 1, pp. 1–42. Cited on pp. 15 and 19.

Howard, Ronald A. 1960. *Dynamic Programming and Markov Processes*. In . Cited on pp. 9 and 11.

Karpathy, Andrej. 2017. *Medium*. URL https://medium.com/@karpathy/a-peek-at-trends-in-machine-learning-ab8a1085a106. Cited on p. 14.

Keskar, Nitish Shirish; Mudigere, Dheevatsa; Nocedal, Jorge; Smelyanskiy, Mikhail; and Tang, Ping Tak Peter. 2016. *On large-batch training for deep learning: Generalization gap and sharp minima*. In arXiv preprint arXiv:1609.04836. Cited on pp. 8, 15, and 19.

Kingma, Diederik P and Ba, Jimmy. 2014. *Adam: A method for stochastic optimization*. In arXiv preprint arXiv:1412.6980. Cited on pp. 13, 14, and 15.

Krogh, Anders and Hertz, John A. 1992. *A simple weight decay can improve generalization*. In Advances in neural information processing systems, pp. 950–957. Cited on p. 15.

Loshchilov, Ilya and Hutter, Frank. 2017. *Fixing weight decay regularization in adam*. In arXiv preprint arXiv:1711.05101. Cited on pp. 14, 15, and 16.

Mnih, Volodymyr; Kavukcuoglu, Koray; Silver, David; Graves, Alex; Antonoglou, Ioannis; Wierstra, Daan; and Riedmiller, Martin. 2013. *Playing atari with deep reinforcement learning*. In arXiv preprint arXiv:1312.5602. Cited on pp. 1, 12, 17, and 18.

Ng, Andrew Y; Coates, Adam; Diel, Mark; Ganapathi, Varun; Schulte, Jamie; Tse, Ben; Berger, Eric; and Liang, Eric. 2006. *Autonomous inverted helicopter flight via reinforcement learning*. In Experimental Robotics IX, pp. 363–372. Springer. Cited on p. 1.

OpenAI; Andrychowicz, Marcin; Baker, Bowen; Chociej, Maciek; Józefowicz, Rafal; McGrew, Bob; Pachocki, Jakub W.; Pachocki, Jakub; Petron, Arthur; Plappert, Matthias; Powell, Glenn; Ray, Alex; Schneider, Jonas; Sidor, Szymon; Tobin, Josh; Welinder, Peter; Weng, Lilian; and Zaremba, Wojciech. 2018. *Learning Dexterous In-Hand Manipulation*. In CoRR, vol. abs/1808.00177. URL http://arxiv.org/abs/1808.00177. Cited on p. 1.

Reddi, Sashank J.; Kale, Satyen; and Kumar, Sanjiv. 2018. *On the Convergence of Adam and Beyond*. In International Conference on Learning Representations. URL https://openreview.net/forum?id=ryQu7f-RZ. Cited on p. 14.

Schaul, Tom; Quan, John; Antonoglou, Ioannis; and Silver, David. 2015. *Prioritized experience replay*. In arXiv preprint arXiv:1511.05952. Cited on p. 19.

Silver, David; Huang, Aja; Maddison, Chris J; Guez, Arthur; Sifre, Laurent; Van Den Driessche, George; Schrittwieser, Julian; Antonoglou, Ioannis; Panneershelvam, Veda; Lanctot, Marc; et al. 2016. *Mastering the game of Go with deep neural networks and tree search*. In nature, vol. 529, no. 7587, p. 484. Cited on p. 1.

Silver, David; Schrittwieser, Julian; Simonyan, Karen; Antonoglou, Ioannis; Huang, Aja; Guez, Arthur; Hubert, Thomas; Baker, Lucas; Lai, Matthew; Bolton, Adrian; et al. 2017. *Mastering the game of go without human knowledge*. In Nature, vol. 550, no. 7676, p. 354. Cited on p. 1.

Statt, Nick. 2019. *The Verge*. URL https://www.theverge.com/2019/4/13/18309459/openai-five-dota-2-finals-ai-bot-competition-og-e-sports-the-international-champion. Cited on p. 1.

Sutton, Richard S and Barto, Andrew G. 2018. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, USA, 2nd edn. ISBN 978-0-26203-924-6. Cited on pp. 1 and 7.

Tesauro, Gerald. 1995. *Temporal difference learning and TD-Gammon*. In Communications of the ACM, vol. 38, no. 3, pp. 58–69. Cited on p. 1.

Tieleman, Tijmen and Hinton, Geoffrey. 2012. *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude*. In COURSERA: Neural networks for machine learning, vol. 4, no. 2, pp. 26–31. Cited on p. 13.

Tufte, Edward R. may 2001. *The Visual Display of Quantitative Information*. Graphics Press LLC, Cheshire, CT, USA, 2nd edn. ISBN 0-961-39214-2. Cited on p. d.

———. jul 2006. *Beautiful Evidence*. Graphics Press LLC, Cheshire, CT, USA. ISBN 0-961-39217-7. Cited on p. d.

Van Hasselt, Hado; Guez, Arthur; and Silver, David. 2016. *Deep reinforcement learning with double q-learning*. In Thirtieth AAAI Conference on Artificial Intelligence. Cited on p. 19.

Wang, Ziyu; Schaul, Tom; Hessel, Matteo; Van Hasselt, Hado; Lanctot, Marc; and De Freitas, Nando. 2015. *Dueling network architectures for deep reinforcement learning*. In arXiv preprint arXiv:1511.06581. Cited on p. 19.

Watkins, Christopher JCH and Dayan, Peter. 1992. *Q-learning*. In Machine learning, vol. 8, no. 3-4, pp. 279–292. Cited on p. 11.

Wilson, Ashia C; Roelofs, Rebecca; Stern, Mitchell; Srebro, Nati; and Recht, Benjamin. 2017. *The marginal value of adaptive gradient methods in machine learning*. In Advances in Neural Information Processing Systems, pp. 4148–4158. Cited on p. 14.

Zhang, Guodong; Wang, Chaoqi; Xu, Bowen; and Grosse, Roger B. 2018. *Three Mechanisms of Weight Decay Regularization*. In CoRR, vol. abs/1810.12281. URL http://arxiv.org/abs/1810.12281. Cited on p. 16.

INDEX

APPENDICES

none