Reinforcement Learning

Summary

Fabian Damken

July 24, 2022

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

In this course we will look at lots of methods from the domain of *reinforcement learning (RL)*. RL is an approach for agent-oriented learning where the agent learns by repeatedly acting with the environment and from rewards. Also, it does not know how the world works in advance. RL is therefore close to how humans learn and tries to tackle the fundamental challenge of artificial intelligence (AI):

"The fundamental challenge in artificial intelligence and machine learning is learning to make good decisions under uncertainty." (Emma Brunskill)

RL is so general that every AI problem can be phrased in its framework of learning by interacting. However, the typical setting is that at every time step, an agent perceives the state of the environment and chooses an action based on these perceptions. Subsequently, the agent gets a numerical reward and tries to maximize this reward by finding a suitable strategy. This procedure is illustrated in Figure 1.1.

1.1 Artificial Intelligence

The core question of AI is how to build "intelligent" machines, requiring that the machine is able to adapt to its environment and handle unstructured and unseen environments. Classically, AI was an "engine" producing answers to various queries based on rules designed by a human expert in the field. In (supervised) machine learning (ML), the rules are instead learned from a (big) data set and the "engine" produces answers based on the data. However, this approach (leaning from labeled data) is not sufficient for RL as demonstrations might be imperfect, the correspondence problem, and that we cannot demonstrate everything. We can break these issues down as follows: supervised learning does not allow "interventions" (trial-and-error) and evaluative feedback (reward).

The core idea leading to RL was to not program machines to simulate an adult brain, but to simulate a child's brain that is still learning. RL formalizes this idea of intelligence to interpret rich sensory input and choosing complex actions. We know that this may be possible as us humans do it all the time. This lead to the RL view on AI depicted in Figure 1.1 and is based on the hypothesis that learning from a scalar reward is sufficient to yield intelligent behavior (Sutton and Barto, 2018).

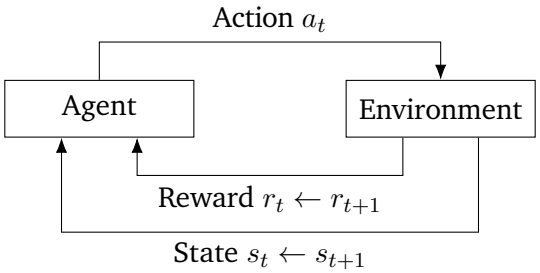


Figure 1.1: The Reinforcement Learning Cycle

	actions *do not* change the state of the world	actions change the state of the world
no model	(Multi-Armed) Bandits	Reinforcement Learning
known model	Decision Theory	Optimal Control, Planning

Table 1.1: Problem Classification

1.2 Reinforcement Learning Formulation

RL tries to *maximize the long-term reward* by finding a strategy/policy with the general assumption that it is easier to assess a behavior by specifying a cost than specifying the behavior directly. In general, we have the following things different to most (un)supervised settings:

- no supervision, but only a reward signal
- feedback (reward) is always delayed and not instantaneous
- time matters, the data is sequential and by no means i.i.d.
- the agent's actions influence the subsequent data, i.e., the agent generates its own data

In addition to this, RL is challenged by a numerous complicated factors and issues, e.g., dynamic state-dependent environments, stochastic and unknown dynamics and rewards, exploration vs. exploitation, delayed rewards (how to assign a temporal credit), and very complicated systems (large state spaces with unstructured dynamics). For designing an RL-application, we usually have to choose the state representation, decide how much prior knowledge we want to put into the agent, choose an algorithm for learning, design an objective function, and finally decide how we evaluate the resulting agent. By all these decisions, we want to reach a variety of goals, e.g., convergence, consistency, good generalization abilities, high learning speed (performance), safety, and stability. However, we are usually pretty restricted in terms of computation time, available data, restrictions in the way we act (e.g., safety constraints), and online vs. offline learning.

This sounds like a lot and, in fact, is! We therefore often limit ourselves onto specific (probably simpler) sub-problems and solve them efficiently under some assumptions. Some common flavors of the RL problem are, for instance:

- *Full:* no additional assumptions, the agent can only probe the environment through the state dynamics and its actions; the agent has to understand the environment
- *Filtered State and Sufficient Statistics:* assumption of a local Markov property (i.e., the next state only depends on the current state and action, and not on the past), decomposable rewards (into specific time steps); we can show that every problem is a (probably infinite) instance of this assumption, but how to filter the state to get such properties?
- *Markovian Observable State:* assume that we can observe the state fulfilling the Markov property directly
- *Further Simplifications:* contextual bandits (the dynamics do not depend on the action or the past and current state at all); bandits (only a single state)

We can summarize the different RL-like problems in a matrix, see Table 1.1.

1.2.1 Components

To solve an RL problem, we need three ingredients:

1. Model Learning
    - we want to approximate and learn the state transfer using methods from supervised learning
    - need to generate actions for model identification
    - estimation of the model or the model's parameters
2. Optimal Control/Planning
    - generation of optimal control inputs
3. Performance Evaluation

1.3 Wrap-Up

- why RL is crucial for AI and why all other approaches are ultimately doomed
- background and characteristics of RL
- classification of RL problems
- core components of RL algorithms

2 Preliminaries

In this chapter we cover some preliminaries that are necessary for understanding the rest of the course. Note that most of this content is dense and should be used as a reference throughout this course as oppose to an actual introduction to the topic.

2.1 Functional Analysis

Definition 1 (Normed Vector Space)**.** A *normed vector space* is a vector space $\mathcal{X}$ over X equipped with a *norm* $\|\cdot\| : \mathcal{X} \to \mathbb{R}$ that has the following properties:

1. $\|x\| \geq 0$ for all $x \in \mathcal{X}$ and $\|x\| = 0$ iff $x = 0$ (non-negativity)
2. $\|\alpha x\| = |\alpha|\|x\|$ for all $\alpha \in X$ and $x \in \mathcal{X}$ (homogenity)
3. $\|x_1 + x_2\| \leq \|x_1\| + \|x_2\|$ for all $x_1, x_2 \in \mathcal{X}$ (triangle inequality)

For the rest of this course we usually use real finite-dimensional vectors spaces $\mathcal{X} = \mathbb{R}^d$, $d \in \mathbb{N}^+$, the L_∞-norm $\|\cdot\|_\infty$, and (weighted) L_2-norms $\|\cdot\|_{2,\rho}$.

Definition 2 (Complete Vector Space)**.** A vector space $\mathcal{X}$ is *complete* if every Cauchy sequence[1] in $\mathcal{X}$ has a limit in $\mathcal{X}$.

Definition 3 (Contraction Mapping)**.** Let $\mathcal{X}$ be a vector space equipped with a norm $\|\cdot\|$. An operator $T : \mathcal{X} \to \mathcal{X}$ is called an *α-contraction mapping* if $\exists \alpha \in [0, 1) : \forall x_1, x_2 \in \mathcal{X} : \|Tx_1 - Tx_2\| \leq \alpha\|x_1 - x_2\|$. If only $\exists \alpha \in [0, 1] : \forall x_1, x_2 \in \mathcal{X} : \|Tx_1 - Tx_2\| \leq \alpha\|x_1 - x_2\|$, T is called *non-expanding*.

Definition 4 (Lipschitz Continuity)**.** Let $\mathcal{X}$ and $\mathcal{Y}$ be vector spaces equipped with norms $\|\cdot\|_X$ and $\|\cdot\|_Y$, respectively. A function $f : \mathcal{X} \to \mathcal{Y}$ is called *Lipschitz-continuous* if $\exists L \geq 0 : \forall x_1, x_2 \in \mathcal{Y} : \|f(x_1) - f(x_2)\|_Y \leq L\|x_1 - x_2\|_X$.

Remark 1. *Obviously, every contraction mapping is also Lipschitz-continuous with Lipschitz-constant $L \triangleq \alpha$ and is therefore continuous. Also, the product of two Lipschitz-continuous mappings is Lipschitz-continuous and therefore $T^n = T \circ \cdots \circ T$ is Lipschitz-continuous, too.*

Definition 5 (Fixed Point)**.** Let $\mathcal{X}$ be a vector space equipped and let $T : \mathcal{X} \to \mathcal{X}$ be an operator. Then $x \in \mathcal{X}$ is a *fixed point* of T if $Tx = x$.

Theorem 1 (Banach Fixed Point Theorem)**.** *Let $\mathcal{X}$ be a complete vector space with a norm $\|\cdot\|$ and let $T : \mathcal{X} \to \mathcal{X}$ be an α-contraction mapping. Then T has a unique fixed point $x^* \in \mathcal{X}$ and for all $x_0 \in \mathcal{X}$ the sequence $x_{n+1} = Tx_n$ converges to x^* geometrically, i.e., $\|x_n - x^*\| \leq \alpha^n\|x_0 - x^*\|$.*

[1] This section is already overflowing with mathematical rigor compared to the rest of the course, so we will skip the definition of a Cauchy sequence here.

2.2 Statistics

This section introduces some concepts of statistics, but you should

2.2.1 Monte-Carlo Estimation

Let X be a random variable with mean $\mu = \mathbb{E}[X]$ and variance $\sigma^2 = \mathrm{Var}[X]$ and let $\{x_i\}_{i=1}^n$ be i.i.d. realizations of X. We then have the *empirical mean* $\hat{\mu}_n = \frac{1}{n}\sum_{i=1}^n x_i$ and we can show that $\mathbb{E}[\hat{\mu}_n] = \mu$ and $\mathrm{Var}[\hat{\mu}_n] = \sigma^2/n$. Also, if the sample size n goes to infinity, we have the *strong* and *weak law of large numbers,* respectively:

$$P\big(\lim_{n\to\infty} \hat{\mu}_n = \mu\big) = 1 \qquad\qquad \lim_{n\to\infty} P\big(|\hat{\mu}_n - \mu| > \epsilon\big) = 0$$

Also, we have the *central limit theorem:* no matter the distribution of P, its mean value converges to a normal distribution, $\sqrt{n}(\hat{\mu}_n - \mu) \xrightarrow{D} \mathcal{N}(0, \sigma^2)$.

2.2.2 Bias-Variance Trade-Off

When evaluating/training a ML model, the error is due to two factors (illustrated in Figure 2.1):

- *bias,* i.e., the distance to the expected prediction
- *variance,* i.e., the variability of a prediction for a given data point

In general, we want to minimize both, but we can only minimize one of them! This is known as the *bias-variance trade-off.*

2.2.3 Important Sampling

If we want to estimate the expectation of some function $f(x)$ for $x \sim p(x)$, but cannot sample from $p(x)$ (which is often the case for complicated models), we can instead use the following relation(s):

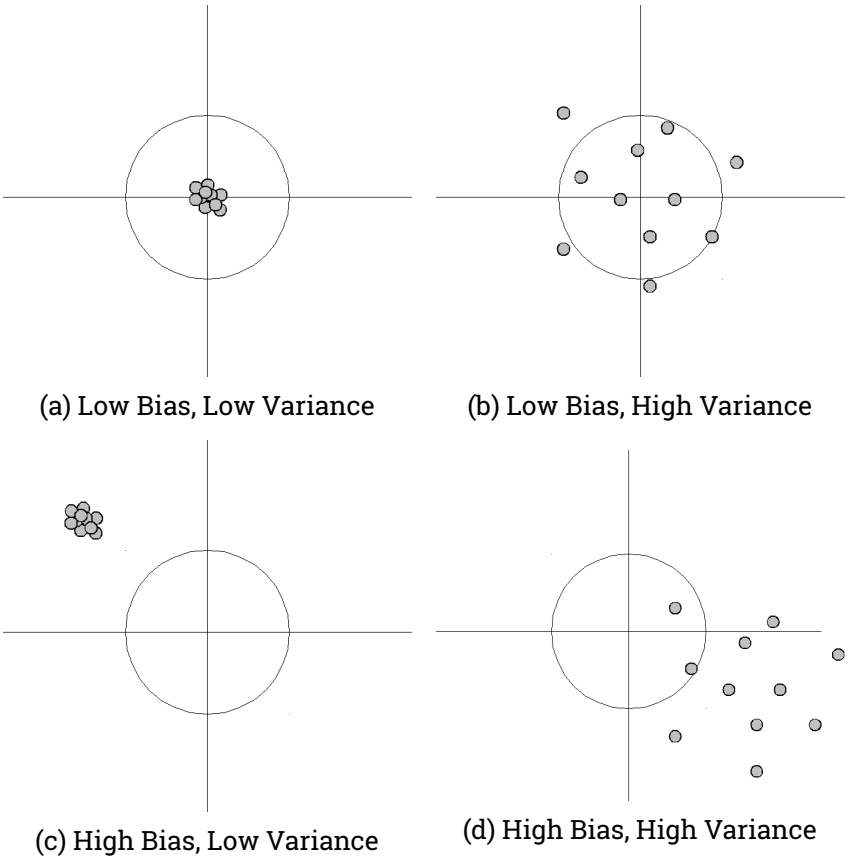
$$\mathbb{E}_{x\sim p}\big[f(x)\big] = \sum_x f(x)p(x) = \sum_x f(x)\frac{p(x)}{q(x)}q(x) = \mathbb{E}_{x\sim q}\left[f(x)\frac{p(x)}{q(x)}\right]$$
$$\mathbb{E}_{x\sim p}\big[f(x)\big] = \int f(x)p(x)\,\mathrm{d}x = \int f(x)\frac{p(x)}{q(x)}p(x)\,\mathrm{d}x = \mathbb{E}_{x\sim q}\left[f(x)\frac{p(x)}{q(x)}\right]$$

and sample from a surrogate distribution $q(x)$. This approach obviously has problems if q does not cover p sufficiently well along with other problems. See Bishop, 2006, Chapter 11 for details.

2.2.4 Linear Function Approximation

A basic approximator we will need often is the linear function approximator $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x})$ with weights $\boldsymbol{w}$ and features $\boldsymbol{\phi}(\boldsymbol{x})$. As the weights are optimized and the features are designed, we have lots of variability here. Actually, constructing useful features is the influential step on the approximation quality. Most importantly, features are the only point where we can introduce interactions between different dimensions. A good representations therefore captures all dimensions and all (possibly complex) interaction.

We will now go over some frequently used features.



(a) Low Bias, Low Variance

(b) Low Bias, High Variance

(c) High Bias, Low Variance

(d) High Bias, High Variance

Figure 2.1: Bias-Variance Trade-Off; Source: Bernhard Thiery (CC BY-SA 3.0)

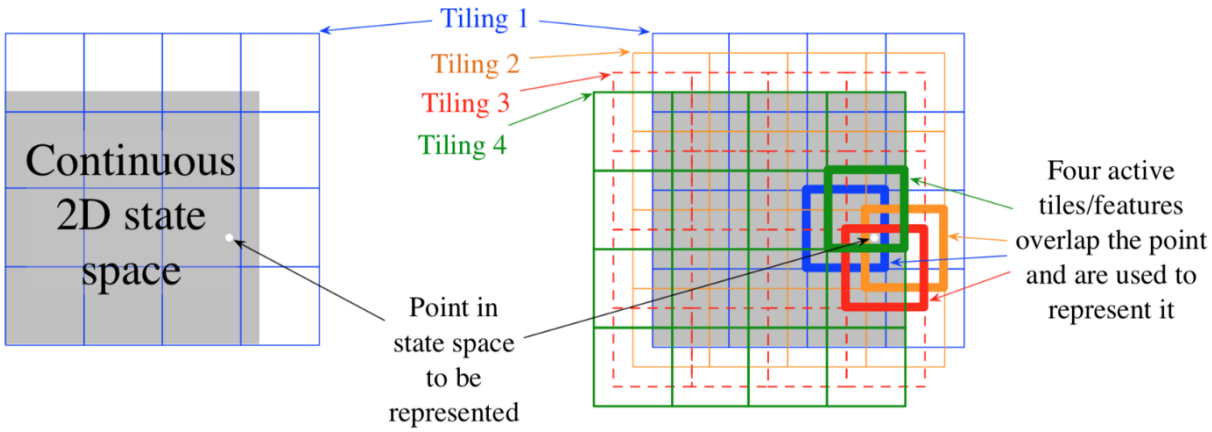


Figure 2.2: Tile Coding; Source: `https://towardsdatascience.com/reinforcement-learning-tile-coding-implementation-7974b600762b`

Polynomial Features *Polynomial features* are particularly simple and capture the interaction between dimensions by multiplication. For instance, the first- and second-order polynomial features of a two-dimensional state $\boldsymbol{x} = (x_1, x_2)^\top$ are:

$$\boldsymbol{\phi}_{P1}(\boldsymbol{x}) = (1, x_1, x_2, x_1 x_2)^\top \qquad \boldsymbol{\phi}_{P2}(\boldsymbol{x}) = (1, x_1, x_2, x_1 x_2, x_1^2, x_2^2, x_1 x_2^2, x_1^2 x_2, x_1^2, x_2^2)$$

However, the number of features grows *exponentially* with the dimension!

Fourier Basis Fourier series can be used to approximate periodic functions by adding sine and cosine waves with different frequencies and amplitudes. Similarly, we can use them for general function approximation of functions with bounded domain. As it is possible to approximate any even function with just cosine waves and we are only interested in bounded domains, we can set this domain to positive numbers only and can therefore approximate any function. For one dimension, the n-th order *Fourier (cosine) basis* is

$$\phi_m(x) = \cos(\pi m \tilde{x}), \quad m = 0, 1, \ldots, n.$$

and $\tilde{x}$ is a normalized version of x, i.e., $\tilde{x} = (x - x_{\max})/(x_{\max} - x_{\min})$.

Coarse Coding *Coarse coding* divides the space into M different regions and produced M-dimensional coding features for which the j-th entry is 1 iff the data point lies withing the respective region; all values the data point does not lie in are 0. Features with this codomain are also called *sparse*.

Tile Coding *Tile coding* is a computationally efficient form of coarse coding which use square *tilings* of space. It uses N tilings, each composed of M tiles. The features "vector" is then an $N \times M$ matrix where a single value is 1 iff x lies inside the tile and 0 otherwise. Figure 2.2 shows an illustration of this coding.

Radial Basis Functions *Radial basis functions (RBFs)* are a generalization of coarse coding where the features are in the interval $(0, 1]$. A typical RBF is the Gaussian

$$\phi_j(\boldsymbol{x}) = \exp\left\{-\frac{\|\boldsymbol{x} - \boldsymbol{c}_j\|_2^2}{2\sigma_j^2}\right\}$$

with center $\boldsymbol{c}_j$ and bandwidth σ_j^2.

Neural Networks A very powerful alternative to hand-crafting features are *neural networks (NNs)*. By stacking multiple layers of learned features, they are very powerful prediction machines.

2.2.5 Likelihood-Ratio Trick

Suppose we need to differentiate the expectation of some function $f(x)$ w.r.t. θ where $x \sim p_\theta(\cdot)$. However, we cannot directly calculate $\mathbb{E}_{x\sim p_\theta}[f(x)]$ or "differentiate through sampling." Instead, we can use the identity

$$\frac{\mathrm{d}}{\mathrm{d}z}\log h(z) = \frac{h'(z)}{h(z)} \qquad \Longrightarrow \qquad f'(z) = h(z)\,\frac{\mathrm{d}}{\mathrm{d}z}\log h(z)$$

to reformulate the derivative of the expectation as

$$\frac{\partial}{\partial\theta}\mathbb{E}_{x\sim p_\theta}[f(x)] = \int f(x)\frac{\partial}{\partial\theta}p_\theta(x)\,\mathrm{d}x = \int f(x)\left(\frac{\partial}{\partial\theta}p_\theta(x)\right)p_\theta(x)\,\mathrm{d}x = \mathbb{E}_{x\sim p_\theta}\left[f(x)\,\frac{\partial}{\partial\theta}p_\theta(x)\right].$$

While this is a very powerful approach, the gradient estimator exhibits high variance!

2.2.6 Reparametrization Trick

Suppose we need to differentiate the expectation of some function $f(x)$ w.r.t. θ where $x \sim p_\theta(\cdot)$. However, we cannot directly calculate $\mathbb{E}_{x\sim p_\theta}[f(x)]$ or "differentiate through sampling." Instead, we reformulate the expectation with a function $x = g_\theta(\varepsilon)$ that separates the random components ε from the deterministic ones θ such that we can reparameterize the expectation as

$$\mathbb{E}_{x\sim p_\theta}[f(x)] = \mathbb{E}_\varepsilon\big[f\big(g_\theta(\varepsilon)\big)\big].$$

For instance, if $p_\theta(x) = \mathcal{N}(\mu_\theta, \sigma_\theta^2)$ is a Gaussian, $g_\theta(\varepsilon) = \mu_\theta + \sigma_\theta\varepsilon$ with $\varepsilon \sim \mathcal{N}(0,1)$. We can now simply use the chain rule to take the derivative w.r.t. θ. Compared to the likelihood-ratio trick, this estimator has less variance!

2.3 Miscellaneous

Finally, this section contains all the stuff that does not fit into the categories before.

2.3.1 Useful Integrals

The following hold for a distribution $p_\theta(x)$:

$$\int \frac{\partial}{\partial\theta}p_\theta(x)\,\mathrm{d}x = 0 \qquad\qquad \int \frac{\partial}{\partial\theta}\log p_\theta(x)\,\mathrm{d}x = \int \frac{\frac{\partial}{\partial\theta}p_\theta(x)}{p_\theta(x)}\,\mathrm{d}x = 0$$

The first identity can be shown by swapping the integral and derivative and using the normalization condition of probability densities. For the second we use integration by parts with $f' = \frac{\partial}{\partial\theta}p_\theta(x)$, for which $f = 0$ due to the first integral. Hence, the second follows.

3 Markov Decision Processes and Policies

3.1 Markov Decision Processes

3.1.1 Continuous State-Action-Space

3.1.2 Example

3.2 Markov Reward Processes

3.2.1 Return and Discount

3.2.2 Value Function

Bellman Equation

3.2.3 Example

3.3 Markov Decision Processes

3.3.1 Policies

Value Functions

Bellman Expectation Equation

Bellman Operator

Optimality

Bellman Optimality Equation

Bellman Optimality Operator

3.3.2 Example

3.4 Wrap-Up

4 Dynamic Programming

4.1 Finite Horizon DP

4.2 Policy Iteration

4.2.1 Policy Evaluation

4.2.2 Policy Improvement

4.2.3 Using the Action-Value Function

4.2.4 Examples

4.3 Value Iteration

4.3.1 Principle of Optimality

4.3.2 Convergence

4.3.3 Example

4.4 Policy vs. Value Iteration

4.5 Efficiency

5 Monte-Carlo Algorithms

5.1 Policy Evaluation

5.2 Example

6 Temporal Difference Learning

6.1 Temporal Differences vs. Monte-Carlo

6.1.1 Bias-Variance Trade-Off

6.1.2 Markov Property

6.1.3 Backup

6.2 Bootstrapping and Sampling

6.3 TD(λ)

6.3.1 n-Step Return

6.3.2 λ-Return

6.3.3 Eligibility Traces

6.4 Example

6.5 Wrap-Up

7 Tabular Reinforcement Learning

7.0.1 Monte-Carlo Methods

Generalized Policy Iteration

Greediness and Exploration vs. Exploitation

ϵ-Greedy Exploration and Policy Improvement

Monte-Carlo Policy Iteration and Control

GLIE Monte-Carlo Control

7.0.2 TD-Learning: SARSA

Convergence

n-Step

Eligibility Traces and SARSA(λ)

Example

7.1 Off-Policy Methods

7.1.1 Monte-Carlo

7.1.2 TD-Learning

Importance Sampling

Q-Learning

Convergence

Example

7.2 Remarks

7.3 Wrap-Up

8 Function Approximation

8.1 On-Policy Methods

8.1.1 Stochastic Gradient Descent

8.1.2 Gradient Monte-Carlo

. . . with Linear Function Approximation

8.1.3 Semi-Gradient Methods

. . . with Linear Function Approximation

8.1.4 Least-Squares TD

Semi-Gradient SARSA

8.2 Off-Policy Methods

8.2.1 Semi-Gradient TD

8.2.2 Divergence

8.3 The Deadly Triad

8.4 Offline Methods

8.4.1 Batch Reinforcement Learning

8.4.2 Least-Squares Policy Iteration

8.4.3 Fitted Q-Iteration

8.5 Wrap-Up

9 Policy Search

9.1 Policy Gradient

9.1.1 Computing the Gradient

Finite Differences

Least-Squares-Based Finite Differences

Likelihood-Ratio Trick

9.1.2 REINFORCE

Gradient Variance and Baselines

Example

9.1.3 GPOMDP

9.2 Natural Policy Gradient

9.3 The Policy Gradient Theorem

9.3.1 Actor-Critic

9.3.2 Compatible Function Approximation

Example

9.3.3 Advantage Function

9.3.4 Episodic Actor-Critic

9.4 Wrap-Up

10 Deep Reinforcement Learning

10.1 Deep Q-Learning: DQN

10.1.1 Replay Buffer

10.1.2 Target Network

10.1.3 Minibatch Updates

10.1.4 Reward- and Target-Clipping

10.1.5 Examples

10.2 DQN Enhancements

10.2.1 Overestimation and Double Deep Q-Learning

10.2.2 Prioritized Replay Buffer

10.2.3 Dueling DQN

10.2.4 Noisy DQN

10.2.5 Distributional DQN

10.2.6 Rainbow

10.3 Other DQN-Bases Methods

10.3.1 Count-Based Exploration

10.3.2 Curiosity-Driven Exploration

10.3.3 Ensemble-Driven Exploration

10.4 Wrap-Up

11 Deep Actor-Critic

11.1 Surrogate Loss

11.1.1 Kakade-Langford-Lemma

11.1.2 Practical Surrogate Loss

11.2 Advantage Actor-Critic (A2C)

11.3 On-Policy Methods

11.3.1 Trust-Region Policy Optimization (TRPO)

Practical Implementation

Conjugate Gradient

11.3.2 Proximal Policy Optimization (PPO)

11.4 Off-Policy Methods

11.4.1 Deep Deterministic Policy Gradient (DDPG)

11.4.2 Twin Delayed DDPG (TD3)

11.4.3 Soft Actor-Critic (SAC)

11.5 Wrap-Up

12 Frontiers

12.1 Partial Observability

12.2 Hierarchical Control

12.2.1 The Options Framework

12.3 Markov Decision Processed Without Reward

12.3.1 Intrinsic Motivation

12.3.2 Inverse Reinforcement Learning

12.4 Model-Based Reinforcement Learning

12.5 Wrap-Up