Mohit Sewak

Deep Reinforcement Learning

Frontiers of Artificial Intelligence

Springer

Deep Reinforcement Learning

Mohit Sewak

Deep Reinforcement Learning

Frontiers of Artificial Intelligence

Springer

Mohit Sewak
Pune, Maharashtra, India

ISBN 978-981-13-8284-0 ISBN 978-981-13-8285-7 (eBook)
https://doi.org/10.1007/978-981-13-8285-7

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Preface

Reinforcement Learning has evolved a long way with the enhancements from deep learning. Recent research efforts into combining deep learning with Reinforcement Learning have led to the development of some very powerful *deep Reinforcement Learning* systems, algorithms, and agents which have already achieved some extraordinary accomplishment. Not only have such systems surpassed the capabilities of most of the classical and non-deep-learning-based Reinforcement Learning agents, but have also started outperforming the best of human intelligence at tasks which were believed to require extreme human intelligence, creativity, and planning skills. Some of the DQN-based agents consistently beating the best of human players at the complex game of AlphaGo are very good examples of this.

This book starts with the basics of Reinforcement Learning and explains each concept using very intuitive and easy to understand examples and applications. Continuing with similar examples, this book then builds upon to introduce some cutting-edge researches and advancements that make Reinforcement Learning outperform many of the other (artificial) intelligent systems. This book aims to not only equip the readers with just the mathematical understanding of multiple cutting-edge Reinforcement Learning algorithms, but also prepares them to implement these and similar advanced *Deep Reinforcement Learning* agents and system hands-on in their own domain and application area.

This book starts from the basic building blocks of Reinforcement Learning, then covers the popular classical DP and classical RL approaches like value and policy iteration, and then covers some popular traditional Reinforcement Learning algorithms like the TD learning, SARSA, and the Q-Learning. After building this foundation, this book introduces deep learning and implementation aids for modern Reinforcement Learning environments and agents. After this, the book starts diving deeper into the concepts of *Deep Reinforcement Learning* and covers algorithms like the deep Q networks, double DQN, dueling DQN, (deep) synchronous

actor-critic, (deep) asynchronous advantage actor-critic, and the deep deterministic policy gradient. Each of the theoretical/mathematical chapters on these concepts is followed by a chapter on practical coding and implementation of these agents' grounds-up connecting the concepts to the code.

Pune, India Mohit Sewak

Who This Book Is For?

This book will equally appeal to readers with prior experience in deep learning, who want to learn new skills in Reinforcement Learning, and to readers who have been the practitioner of Reinforcement Learning or other automation systems and who want to scale their knowledge and skills to *Deep Reinforcement Learning*. By combining the concepts from deep learning and Reinforcement Learning, we could come closer to realizing the true potential of 'general artificial intelligence'.

Besides presenting the mathematical concepts and contemporary research in the field of *Deep Reinforcement Learning*, this book also covers algorithms, codes, and practical implementation aids for both Reinforcement Learning environments and agents. This book is intended to be a guide and an aid for both the types of readers, for the ones who are interested in the academic understanding and being abreast with some of the latest advancements in *Deep Reinforcement Learning* and also for the ones who want to implement these advanced agents and systems into their own fields.

Ranging from application in autonomous vehicles to dynamic scheduling and management of production process, to intelligent maintenance of critical machineries, to driving efficiency in utility management, to making automated systems for health care, to intelligent financial trading and transaction monitoring, to aiding intelligent customer engagement, and to mitigating high-throughput cyber threats, the concepts learnt in this book could be applied to multiple fields of interest.

The code in the book is in Python 3x. The deep learning part of the code uses the TensorFlow library. Some code also uses the Keras wrapper to TensorFlow. *Deep Reinforcement Learning* wrappers like Keras-RL are also demonstrated. This book expects basic familiarization in Python with object-oriented programming concepts to enable implementation of distributed and scalable systems.

What This Book Covers?

Chapter 1—*Introduction to Reinforcement Learning*—covers the basic design of Reinforcement Learning and explains in detail the concepts like the environment, actor, state, and rewards, and the challenges in each.

Chapter 2—*Mathematical and Algorithmic Understanding of Reinforcement Learning*—builds upon a strong mathematical and algorithmic foundation to understand the internal functioning in different types of agents.

Chapter 3—*Coding the Environment and MDP Solution*—illustrates how to build a custom Reinforcement Learning environment in code over which different reinforcement agents can train and also implements the value iteration and policy iteration algorithms over a custom environment.

Chapter 4—*Temporal Difference Learning, SARSA, and Q-Learning*—covers the TD learning estimation process and the on-policy SARSA and off-policy Q-Learning algorithms along with different types of exploration mechanism.

Chapter 5—*Q-Learning in Code*—implements the Q-Learning algorithm in Python via the tabular approach using the epsilon-greedy algorithm for behavior policy.

Chapter 6—*Introduction to Deep Learning*—introduces the concepts of deep learning like layer architecture, activation, loss functions, and optimizers for the MLP-DNN and CNN algorithms.

Chapter 7—*Implementation Resources*—covers the different types of resources available to implement, test, and compare cutting-edge deep Reinforcement Learning models and environments.

Chapter 8—*Deep Q Network (DQN), Double DQN, and Dueling DQN*—covers the deep Q networks and its variants the double DQN and the dueling DQN and how these models surpassed the best of human adversaries' performance at the game of AlphaGo.

Chapter 9—*Double DQN in Code*—covers implementation of a double DQN with an online active Q network coupled with another offline target Q network with

both networks having customizable deep learning architecture, built using Keras on TensorFlow.

Chapter 10—*Policy-Based Reinforcement Learning Approaches*—covers the basic understanding of policy-based Reinforcement Learning approaches and explains the policy-gradient mechanism with the reinforce algorithm.

Chapter 11—*Actor-Critic Models and the A3C*—covers stochastic policy-gradient-based actor-critic algorithm with its different variants like the one using 'advantage' as a baseline and those that could be implemented in 'synchronous' and 'asynchronous' distributed parallel architectures.

Chapter 12—*A3C in Code*—covers the implementation of the asynchronous variant of the distributed parallel actor-critic mechanism with multiple agents working simultaneously to update the master's gradient. The agent algorithm is implemented using the TensorFlow library, using the libraries' 'eager execution' and model's 'sub-classing' features.

Chapter 13—*Deterministic Policy Gradient and the DDPG*—covers the deterministic policy-gradient theorem and the algorithm and also explains the enhancements made to enable the deep learning variant of deep deterministic policy gradient (DDPG).

Chapter 14—*DDPG in Code*—covers the implementation of the DDPG algorithm to enable the Reinforcement Learning tasks requiring continuous-action control and implements it in a very few lines of code using the Keras-RL wrapper library.

Contents

About the Author

Mohit Sewak is a Ph.D. scholar in CS&IS (Artificial Intelligence and Cyber Security) with BITS Pilani - Goa, India, and is also a lecturer on subjects like Artificial Intelligence, Machine Learning, Deep Learning and NLP for the post-graduate technical degree program. He holds several patents (USPTO & Worldwide) and publications in the field of Artificial Intelligence and Machine Learning.

Besides his academic linkages, Mohit is also actively engaged with the industry and has many accomplishments while leading the research and development initiatives of many international AI products. Mohit has been leading the Reinforcement Learning practice at QiO Technologies, the youngest player in Gartner's magic quadrant for Industry 4.0.

In his previous roles, Mohit had led the IBM Watson Commerce in India's innovation initiative in cognitive line of feature as Sr. Cognitive Data Scientist. Mohit had also been the Principal Data Scientist for IBM's global IoT products like IBM Predictive Maintenance & Quality. He had also been the advanced analytics architect for IBM's SPSS suite in India.

Mohit has over 14 years of very rich experience in researching, architecting and solutioning with technologies like TensorFlow, Torch, Caffe, Theano, Keras, Open AI, OpenCV, SpaCy, Gensim, Spark, Kafka, ES, Kubernetes, and Tinkerpop.

Chapter 1
Introduction to Reinforcement Learning

The Intelligence Behind the AI Agent

Abstract In this chapter, we will discuss what is Reinforcement Learning and its relationship with Artificial Intelligence. We would then try to go deeper to understand the basic building blocks of Reinforcement Learning like state, actor, environment, and the reward, and will try to understand the challenges in each of the aspect as revealed by using multiple examples so that the intuition is well established, and we build a solid foundation before going ahead into some advanced topics. We would also discuss how the agent learns to take the best action and the policy for learning the same. We will also learn the difference between the On-Policy and the Off-Policy methods.

1.1 What Is Artificial Intelligence and How Does Reinforcement Learning Relate to It?

Artificial Intelligence from a marketing perspective of different organizations may mean a lot of things encompassing systems ranging from conventional analytics, to more contemporary deep learning and chatbots. But technically the use of Artificial Intelligence (AI) terminology is restricted to the study and design of "**Rational**" agents, which could act "**Humanly**". Of the many definitions given by different researchers and authors of Artificial Intelligence, the criteria for calling an agent an AI agent is that it should possess ability to demonstrate "*thought-process* and *reasoning*", "*intelligent-behavior*", "*success* in terms of human performance", and "*rationality*". This identification should be our guiding factor to identify the marketing jargons from real Artificial Intelligence systems and applications from the marketing hype.

Among the different Artificial Intelligence agents, Reinforcement Learning agents are considered to be among the most advanced and very capable of demonstrating high level of intelligence and rational behavior. A reinforcement learning agent interacts with its environment. The environment itself could

M. Sewak, *Deep Reinforcement Learning*,
https://doi.org/10.1007/978-981-13-8285-7_1

demonstrate multiple states. The agent acts upon the environment to change the environment's state, thereby also receiving a reward or penalty as determined by the achieved state and the objective of the agent. This definition may look naïve, but the concepts empowering it led to the development of a many advanced AI agents to perform very complex tasks, sometimes even challenging human performance at specific tasks.

1.2 Understanding the Basic Design of Reinforcement Learning

The diagram as in Fig 1.1 represents a very basic design of Reinforcement Learning system with its "learning" and "action" loops. Here an agent as described in the above introductory definition interacts with its environment to learn to take the best possible action (a_t in the above figure) under the given state (S_t) that the environment is in at step t. The action of the agent in turn changes the state of the environment from S_t to S_{t+1} (as shown in the figure) and generates a reward r_t for the agent. Then the agent takes the best possible action for this new state (S_{t+1}), thereby invoking a reward r_{t+1} and so on. Over a period of iterations (which are referred to as experiments during the training process of the agent) the agent tries to improve upon its decision of which is the "best action" that could be taken in a given state of the environment using the rewards that it receives during the training process.

The role of the environment here is thus to present the agent with different possible/probable states that could exist in the problem that the agent may need to react to, or a representative subset of the same. To assist the learning process of the agent, the environment also gives the reward or penalty (a negative reward) corresponding to the action decisions taken by the agent in a given state. Thus, the reward is a function of both the action and the state, and not of the action alone. Which means that the same action could (and ideally should) receive a different reward under different states.

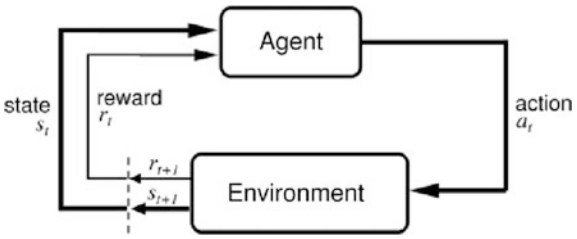

Fig. 1.1 Design for a Reinforcement Learning system

1.3 The Reward and the Challenges in Determining a Good Reward Function for Reinforcement Learning

The role of the agent, as should be obvious from the above discussion, is to take the action that pays the maximum reward in a given state. But this is not so trivial, as we will determine in this section.

1.3.1 Future Rewards

The actual reward for a right action taken in a particular state may not be realized immediately. Imagine a real-life scenario, where you have an option to go out and play now or to sit and study for your upcoming exams now and play later after the exams. Playing now may give a small adrenaline rush immediately, which in the terms of reinforcement learning could be treated as a reward if we consider it a positive outcome. Whereas studying now may seem boring in the short run (this could theoretically be also treated as a small penalty depending upon the objective) but probable may pay off very well in the long run, which may be considered as a bigger reward (for the actions decision to be taken now) but the reward is realized only in the future. Figure 1.2 shows an illustration between such present and future sense of rewards.

There are quite some well-established solutions to this problem like using a *discounting-factor* to discount the future rewards to present time (like in finance we discount the future returns/cash flows from different projects to present time to compare between projects of varying lengths and different time periods of realization of the cash flows) for comparison. We will discuss the solutions using this

Fig. 1.2 Instantaneous versus future rewards

technique later in this book, but for now our objective is to highlight the challenges pertaining to achieving the parity across rewards of varying quantum coming from different time/step spans in the future which could be attributed to the action taken in the present time/step.

1.3.2 Probabilistic/Uncertain Rewards

Another complexity in reinforcement learning is the probabilistic nature of the rewards or uncertainty in the rewards. Let us take the same example of studying now for a reward (good marks) later. Suppose we have 10 chapters in the course and we know that the questions are going to come only from six of these chapters, but we do not know from which specific six chapters the questions are going to come, and how much weightage will each of the chosen six chapters will have in the exams. Let us also assume that in a slot of 3 h that our protagonist could spend in playing an outdoor game, she could study only any one of the 10 chapters.

So even if we assume that the perceived value of the future reward is worth studying for instead of playing, we are not sure if we would be spending the present time studying a particular chapter from which there could be no questions at all. Even if we consider that the chosen chapter is an important one, we do not know the specific weightage of the marks of the questions that would come from this chapter. So the rewards from studying now would not only be realized in the future but could also be probabilistic or uncertain.

1.3.3 Attribution of Rewards to Different Actions Taken in the Past

Another important consideration is the attribution of the rewards to the specific action/s taken in the past. Continuing with the above example, suppose out of the 10 chapters or corresponding 10 slots-to-play, we decide that the protagonist/agent would randomly pick six instances where she/it would study any one of the 10 chapters (assume chosen randomly) and would play in the remaining four slots. We make specific choices with the objective being to maximize the sum total of all the rewards comprising of the small but immediate reward and larger but futuristic and probabilistic rewards.

Assume that with the choices made the protagonist/agent finally end up scoring 50% in its exam. Further assume that of the six chapters we chose to study for, the questions came from only four of them with weightages of marks as 5%, 8%, 12%, and 15%, respectively, and the remaining 50% questions came from the four chapters that we could not prepare as we decided to go to play instead (yes we were sort of unlucky). Assume that of the questions that the agent answered (worth

maximum of 50 marks) it received 40 marks (i.e., a total of 40% of the maximum achievable 100 marks in the exam).

Now, one solution for attribution of this reward (40% marks) could be that we divide this reward equally across each of the six slots where the agent decided to studying instead of playing.

Yet another solution could be that we give as much reward to each specific slot as the weightage of question that came from the chapter the agent decided to study for in that slot.

If the agent had not scored universally well across each of the chapters that came, then yet another solution could have been to reward each slot with the amount of marks that the agent scored (irrespective of the weightage of the questions) from what it studied in that particular slot. So this essentially takes into account both the weightage of marks coming from the chapter that the agent decided to study in that slot and its demonstrated performance for questions from that chapter in the exam as well.

Yet another solution could be to treat the reward for the agent as a function of the percentage of marks that we scored of the questions that came from the chapter we prepared in that slot. That is, the reward is a function of only our performance on what we decided to study. Since which chapters are going to come in the exams and what will be the weightage of each of the chapters that will appear in the exam is not in the agent's control so this method could rightly reward the protagonist with what is in her control. The reward in this approach is completely a function of the agent's performance in choosing the subjects that it can deliver the best results if it devotes the given 3 h in studying it, completely ignoring the total marks achieved in the exams.

But one problem in all these approaches is that we assume that we do not know the distribution of question weights from different chapters, and we do not want the agent to mistakenly learn the weightages just from this single example/experiment as we understand that these could be random across different experiments.

So should we actually equally divide this reward (40% marks) across all the slots in which we decided to study instead of going to play? If so, how do we reward the choice of choosing specific subjects to study where we could have performed better had we studied them as compared to some other subjects where 3 h of study might not have added that much incremental value? Another observation could be that whether there are any specific slots where studying could have been more productive than playing, for example, if we alternate across studying and playing, would a healthy body before each studying slot positively affect our performance in studies as well?

Any of the above solutions could be correct, and all of them could be suboptimal solutions could be wrong depending upon the purpose of our training. For example, if the overall purpose of the training is to score maximum marks in exams, then the last option which seems to be righteous makes no sense, and instead we should try to train an agent which scores decently well irrespective of the uncertainty in chapter/weightage selection. On the contrary, if the purpose is to select the right chapters pertaining to the strength of the protagonist then the later approach could

be better than some of the former approaches. Yet the simplest approach could be the first one to have an equal attribution to remove bias from training.

So deciding reward attribution is both an art and a science and would determine the decisions that the agent finally learns to take in different scenarios. Hence, the attribution criteria and function should be devised considering the intended behavior and objective of the agent.

1.3.4 Determining a Good Reward Function

By now you would have understood that the problem of attribution of reward is not trivial. Not only it is challenging to decide the right attribution from a domain perspective, but even a slight change in the reward function may force a distinctive behavior for the agent and the agent may decide to take different actions based on how we decide to formulate our rewards.

Compare the two examples in the previous section where we use absolute scores and percentage scores as the reward functions, respectively. The attribution with respect to which action (slot/chapter selection) receives the reward is unchanged in these two cases. What differs is the magnitude of reward given or the reward function formulation. By changing the magnitude of reward, we may differ the behavior of the agent as well as it is evident from the above example.

1.3.5 Dealing with Different Types of Reward

Beyond the attribution and reward function for the specific scenario we discussed, there is yet another challenge that we have not observed or discussed so far. How do we equate two different types of rewards?

We have only one numerical value that we could give as a reward (or penalty—i.e., a negative value for the reward). Assume even if we were to correctly standardize/normalize and apply all relevant transformations to correctly fix the reward formulation and attribution problem, how do we account for the rewards from our decisions to go to play instead of spending that time studying a particular chapter in that slot?

Playing could have a different type of reward altogether. If we were to represent it mathematically, then the rewards could have a different unit and scale. In the chosen example, may be playing instead of studying helps (pays reward to) to keep us healthy, may help us attain a good position for our team in the tournaments, etc. So how do we compare our better health, to the marks we obtained (or for that matter missed) in our exams? For the sake of simplicity, let us not get into the details of attributions of the specific hours we played to the improvement of our overall health in that week/month to answer this question for now, that is, could be a topic for another research altogether.