Deep Reinforcement Learning Beginners Tutorial Documentation

Julian Bernhart, Robin Guth

June 23, 2019

Contents

1 Introduction

Deep Reinforcement Learning is a huge step towards the creation of an universal artificial intelligence[1]. In 2013 a company, owned by Google, called "Deep Mind", was able to create an astonishing implementation of RL[2], which was capable to play a range of retro games of the console "Atari 2600". In many cases, the AI[3] was not only able to play the games successfully, but also exceeded human performances significantly [1]. After these impressive results, it is definitely worth to take a closer look at *reinforcement learning*.
Playing games is fun, but *reinforcement learning* has more to offer, as discussed in [2]. Apart from playing video games, there are many use cases in fields like robotics, traffic control or even personalized advertisement. Apart from the Atari paper [1], multiple papers have been written about RL. Some noteworthy examples are the creation of an RL algorithm, which recommends

[1] can be used for multiple problems without notable changes
[2] Reinforcement Learning
[3] Artificial Intelligence

personalized news [3] or a survey about different applications for RL in robotics [4]. While supervised and unsupervised learning are already widely used in production, *reinforcement learning* is still in development and further research is needed. As an interesting topic with many application possibilities, there is a demand for tutorials and hands-on guides for beginners, especially students. Beginners often struggle to find a good starting point into the world of AI and specifically RL. Many existing tutorials are written for more advanced users, who already have a deeper understanding of machine learning.
For this reason, the *"Deep Reinforcement Learning Beginners Tutorial"* was created, as an effort to create an easy to understand, hands-on guide to learn RL without needing extensive knowledge about *machine learning*. After completing the tutorial, the reader will be familiar with the basics of RL and be able to create its own implementation based on the examples given.
This document serves as a documentation of the making and thought-process of the authors, as well as a presentation of the result and a critical reflection of the work. First, the basic contents of the tutorial will be explained, to give the reader a short overview about the topic of RL. Then, the initial targets will be outlined as well as the methods and materials, that were used in the making. Finally, all results will be presented and evaluated against the initial targets. This document will be concluded by the discussion of possible improvements and expansions of the final product.

2 Deep Reinforcement Learning

In this section the content of the two developed notebooks is discussed, mainly the theory behind *DRL*[4]. The reader gets to know the basic concept of software agents and RL, the deep learning aspect with the help of an exemplary neural net and work out the necessary formulas in order to translate the theory into code. Further the environment is introduced, which defines the problem, that the algorithm needs to solve and also holds all rules, which constrain the algorithm. All the theory, explanations and code can be found in full length in the enclosed notebooks:

1. Deep Reinforcement Learning Beginners Tutorial (1) - Theory
2. Deep Reinforcement Learning Beginners Tutorial (2) - Practice

2.1 Software Agent

Before the topic of RL can be explained, some basics need to be addressed in order to clarify the following explanation. For this, the term of *software agent* is introduced, which, as part of artificial intelligence, is basically a program, that acts self-sufficient to solve a task. There are three important aspects an agent should fulfill. First is autonomous action, which means that an agent needs to make decisions without external help. Further an agent needs to execute multiple actions to complete its task. This property is called proactive. Last but not least, an agent has to be reactive, which means it reacts on changes in the environment it is in. An optional ability, for an agent like this, is self-sufficient improvement. For this the agent builds up knowledge after repetitively doing its task and improves itself this way.
Basically, the part of the agent, which controls its actions, can be filled with different algorithms. In this case, this will be an implementation of RL, which fits all requirements of an agent.

2.2 Reinforcement Learning

Reinforcement Learning is considered one of the three machine learning paradigms, alongside supervised learning and unsupervised learning. The main goal of RL is to create an agent, which can navigate by actions in an environment in a way to maximize a representation of a cumulative reward. The environment is the space that contains the agent. The environment is discussed in detail later on. In order to maximize the reward, the agent has to learn by trial and error which actions most likely lead to a reward and which lead to a penalty. After some training, the agent will use its knowledge to avoid previous mistakes. It still has to explore the environment, because otherwise it can not be sure if it actually finds the global maximum or is just stuck at a local one and if there might be a better chain of actions. If this is compared to the real world, this method of learning is pretty close the human learning. If a person gets hurt

[4]Deep Reinforcement Learning

for example, the person is more likely to try to avoid the situation in the future. Still curiosity or necessity leads humans to exploring and thus maybe getting them into danger, but in the best case, will lead them to some kind of positive reward, like food for example.

2.2.1 The Concept of RL

An agent is contained by an environment. A momentary snapshot of the environment is called state. A state contains all information at a certain point in time for example the position of the agent or enemies. There is a permanent exchange of information between the agent and its environment. The agent receives the actual state and has to choose an action based on its logic. Everything the agent can use to alter its environment is called an action. Executing an action changes the environment based on a certain set of rules. After executing an action the agents receives the new state and a reward, which helps to decide whether its decision leads to a positive result. A reward can also be negative. The main goal for the agent is to learn the best action for each state, so that it can maximize the reward. Basically, an environment is just a set of states. The agent can move between states by executing actions. A good example for an environment is the universe. Humans and other living beings are agents, moving in the world. All the time they have to look at their surroundings and choose an action like moving across the street or waiting until it is free. The laws of physics restrain them, take gravity for example. All living things receive rewards like getting hurt or feeling satisfied, that is how they evaluate the actions. The environment is constantly changing, so they have to reevaluate their decisions and choose an action again. The following picture shows the whole procedure:

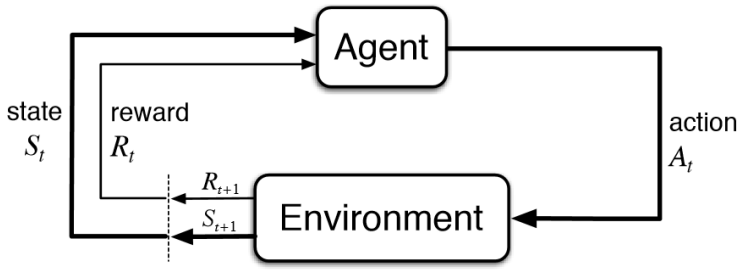

Figure 1: Concept of Reinforcement Learning

2.2.2 Deep Learning Aspect

The agent has to recognize and differentiate elements in the environment, to be able to act accordingly and choose a proper action. In the best case, the input is already reduced to some important numbers, which describe the actual state. In real applications this may not be possible or feasible with the given knowledge. Most of the time there is a picture available, for example the image of a camera or a game screen, but the input the agent receives can basically be any sensory data.
Furthermore the agent must associate the occurrence of individual elements on screen with its own actions and the subsequent reward or penalty, which may occur only after several steps in the future. Depending on the situation it may be also necessary to estimate what the next state of the game could be. There are two possible ways to use the *deep learning* aspect.

1. Image processing with a CNN[5], in order to process the image information and send this to the agent.

2. A DQN[6], in order to model the Q-function, which is discussed below.

Following is an exemplary presentation of a neural network, which is designed according to the tasks just mentioned.
There are two ways to learn in deep reinforcement learning. Either the agent can use Value Learning or Policy Learning. With Value Learning, it assigns values to each state-action pair,

[5]Convolutional Neural Network
[6]Deep Q-Learning Network

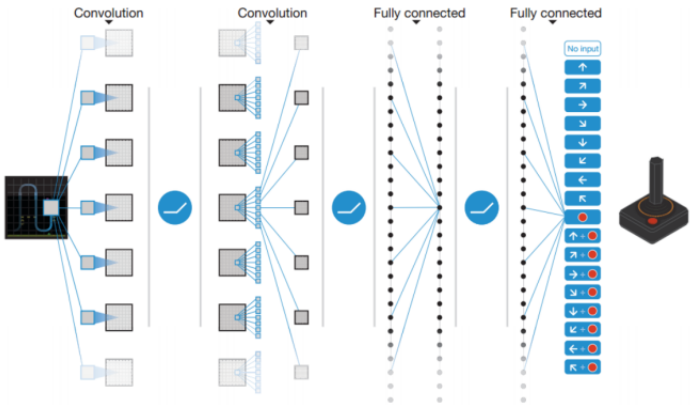

Figure 2: DQN with Convolutional Layers

which correspond to the probabilities of the pair being the best option. In Policy Learning, it will learn a strategy instead, which gives it directly the best estimated action for a given state s. In the notebook, the focus will be on Value Learning.

2.2.3 Q-Learning

The task of the agent is the maximization of a cumulative future reward. An example of this can be the score in a game. In an environment, there is no guarantee for an immediate reward after executing an action. In many cases, only specific chains of actions lead to a positive reward, so the agent needs to learn multiple moves in succession in order to fulfill the given task. For example, if the agent needs to collect an object in order to get a higher score, it may be necessary to take multiple moves to reach it. The moves in between may not increase the reward significantly or even reduce it, however they are needed to complete the task in order to be successful. So the agent may have to learn to sacrifice some rewards, to earn an overall higher reward in the end. A simple way to implement RL is Q-Learning. A numerical value gets assigned to each action which can executed per state. This value is called Q-value. It represents an estimate of which action will result in the highest reward for the actual state. This is the knowledge of the agent. In the beginning all these values are initialized by 0. Each step the agent has to choose an action for the actual state. This is either random or knowledge based, but all the actions taken, provide data the agent is trained with. The agent uses random actions to explore its environment at first. If it uses its knowledge instead, the next action is chosen by exploitation, so the action with the highest Q-value for the given state is used. The next step is the associated function. The Q-function is the name giving aspect of a Deep Q-Learning Network, while the "Q" stands for "Quality". The higher the reward, the higher the estimated Q-value and associated quality. That means this function calculates a Q-value for the actual state, which represents the best action to take. Therefore taking the highest Q-value associated with the biggest estimated future total reward $Q(s', a')$, the last gained reward r_t and a discount factor γ into account is needed, in order to make rewards in the near future more decisive for the agent. The main formula to model such a behaviour is the so called *Bellman equation*.

$$Q(s_t, a_t) = r_t + \gamma * \max_{a'} Q(s', a')$$

3 Initial Targets

As the reader is now familiar with the topic of RL, in this section, the initial targets will be discussed. Later on, these targets are critically compared to the results. Some of the presented targets may overlap.

Beginner-Friendly The main goal in the creation of this tutorial is to create an easy understandable, hands-on beginners guide to RL. This means that there need to be extra steps

to reach this target like:

- extensive, comprehensive explanations
- explanations for new terms
- accompanying images/visualizations
- easy language
- explained code examples
- simple exercises with solutions
- inviting environment
- no big obstacles without support
- easy to set up
- etc.

Topic The focus should be on the basics of RL, as well as everything the reader needs to be able to understand and write its own agents. There should be a theory part as well as accompanying code examples.

Relevant Knowledge After the completion of the tutorial, the user should be able to proceed with other tutorials or sources, without needing to start over again. The knowledge presented should contain necessary and useful basics of RL.

Easy Code As the target for the tutorial is to teach the reader how to use RL, there need to be some code examples. For this an easy programming language is needed, which is relevant for RL and not just used for examples. The code should also be commented and explained.

Combination of text and code Many tutorials in existence require the installation of additional software, like compilers for the programming language. This separates the code from the theory and explanations. For this reason, the code and text should be in the same document, so that a reader can instantly run code while reading the explanations.

Performance An important point to consider is the required computing power to train an agent. As many thousands to millions of steps are needed, this is more than most personal computers can handle in a decent amount of time. For this reason, the tutorial should be based on the usage of a cloud service to execute code. This delivers enough computational power to train an agent in a short period of time, as well as separating the execution environment from the users personal hardware, which removes the need to install software on the users pc.

Budget-Friendly & Accessible As this tutorial is aimed at beginners, who may just want to take a quick look at RL, it is important that the tutorial can be used without obstacles. For this reason, the tutorial should be freely available, as well as usable without costs or very low cost.

4 Methods and Materials

In this section, the methods and materials used in the creation of the tutorial are presented, as well as the thought process of the authors.

4.1 Python

As the programming language *Python* is dominating fields like *data science* or *machine learning*, was made with the intention of creating a very simple programming language and many libraries being written in *Python*, this seems an excellent choice for this tutorial.

4.2 Jupyter Lab

Jupyter Lab was chosen as an environment to develop and execute Python code, as well as the place to combine explanations, pictures and code into one document.
It is basically an editor for jupyter notebooks. A notebook is a document with multiple successive cells, which can be one of the following types.

Code A cell filled with *Python* code.

Markdown A cell filled with *Markdown* instructions.

Raw A cell with raw data representation, is not used generally.

The Lab provides support for the typesetting language "Markdown", which enables the authors to create stylized text sections with images and other features like table of content, web links etc. Each cell can be executed individually, to either create console output or stylized text, so a reader can be guided step by step. A section of an exemplary notebook is shown in image 3.
The usage of *Jupyter Lab* has many advantages like:

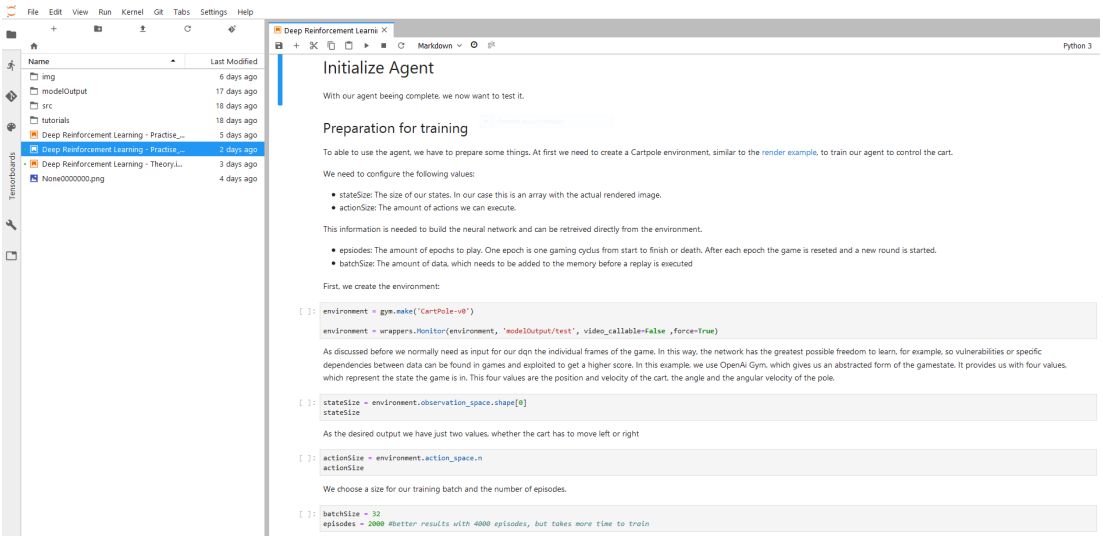

Figure 3: Jupyter Lab

Unified environment Every user has the same development environment. This reduces the probability of errors and simplifies the installation process.

Easy-to-use Jupyter Lab provides an easy to use interface.

Code integration The cell based notebooks include executable code. There is no need to copy code and the code can be placed after corresponding explanations.

Easy dependency installation Jupyter Lab supports the installation of Python libraries through simple commands.

Further information on Juypter Lab and its notebooks can be found at `https://jupyterlab.readthedocs.io/en/stable/`.

4.3 Libraries

As it is common in programming, third-party libraries provide prewritten code, so that a user does not have to implement basic tasks on its own. The following libraries are used in the notebooks:

NumPy Scientific computation in Python (n-dimensional arrays).

Tensorflow Open source machine learning library.

Keras & KerasRL High level implementations of neural networks and different RL algorithms.

OpenAI Gym Provides different machine learning environments based on games.

As OpenAI Gym is quite relevant in the world of AI, it is discussed in greater detail. The other libraries are quite common and provide basic utilities, there is no need to discuss them further.

4.3.1 OpenAI Gym

Developing *machine learning* algorithms is often neither easy to understand nor comprehensible especially for beginners. As most people are familiar with video games, these are simple and completely observable environments. OpenAI Gym is an open source framework, which delivers a range of different game environments. It provides all environmental data a RL agent needs and using an existing game makes it also easier to compare the agent to human performance or other agents and therefore the evaluation of different algorithms is easier. Another advantage is, that a developer does not need to build a dedicated testing environment and saves time.

4.4 Google Cloud

As discussed before, a normal personal computer is to slow to train an agent in a decent amount of time. For this reason, a cloud based approach is chosen.
The cloud service of Google, *Google Cloud*, provides an already preconfigured virtual machine based on the operating system *Linux*. With this, Jupyter Lab can be run on a server with more processing power and the installation process is easier, because the user just has to choose the vm[7] template and it will be automatically installed. This also eliminates the need for the user to install software on the users personal machine and makes it possible to use lower-end machines like laptops to access the vm online. With these advantages, there is also one main disadvantage: As a service of Google, the use of the vm costs money. Luckily, Google offers free credit for students after registration. If this is no option for an user, the use of a cloud is fully optional and the environment of Jupyter Lab can be installed locally without any problems. In terms of ease-of-use, the cloud service is a good choice.

4.5 Source Materials

As RL is a fairly new topic, changes may still occur. For this reason, the used source materials are mostly available online.
The base for the theory part is a freely available lecture[8] of the MIT.
The code is based on a video series about RL[9].

5 Results

In this section, the final result of the efforts to create a deep reinforcement learning tutorial is presented. First, the produced notebooks and other documents are discussed in depth. Then, these results will be compared against the initially set targets.

5.1 Products

The following documents were created in the making of the tutorial:

- Notebook1 - Theory behind RL.
- Notebook2 - Practical example with OpenAI Gym.
- Installation Guide
- Documentation
- Poster about the Documentation

[7] Virtual Machine
[8] Lecture of MIT: `https://www.youtube.com/watch?v=i6Mi2_QM3rA`
[9] Deep Q Learning Networks: `https://www.youtube.com/watch?v=0YhFoMySoVs`

5.1.1 Notebooks

Notebook 1 - Theory This notebook, is mainly used to explain the basics of RL to the reader. In the beginning, the concept of software agents is explained as well as the basic concept of RL. The usage of neural networks in RL is discussed. Two types of reinforcement learning are introduced: *value learning* and *policy learning*. One example for *value learning* is explained in depth, the so called *Q-learning*. The main formula for Q-learning is derived out of simpler formulas. Finally, an outlook is given, mainly about the second notebook.

Notebook 2 - Practice In the second part of the tutorial, the reader can finally start to program an agent. Before that, OpenAI Gym and the chosen environment is introduced. Also, the rendering of the training process is explained. The main part of this notebook is the creation of a fully functional RL agent step-by-step. The tutorial is concluded by a reflection on the things the reader learned and an outlook into further readings and tutorials.

Exercises The exercises are a comparatively small part of the notebooks. The main goal is to explain *deep reinforcement learning*. The notebooks were designed to be easily understandable with a good connection between theory and implementation. All code passages are explained in detail, so that the project can be comprehended in its entirety. The goal of the exercises is to build and understand a basic structure for a RL agent. The individual tasks are located at the points in the code where the agent can be further optimized and customized to create an entry point into a project.

5.1.2 Ease of Use

A very important target while developing the notebooks was to keep the tutorial easy-understandable and usable, as well as having a tutorial with an useful amount of information. In this section, some steps are presented, which were taken to increase the ease of use.

Installation Guide After Google Cloud and Jupyter Lab became central parts of the experience, it became clear, that a potential user may not have the needed knowledge to set up the environment. A full installation guide was created, which guides the user through the installation process with the *Google Cloud*.

Usage of "We" A tutorial should always be welcoming and invite the reader to explore the topic without to big obstacles in the way. For this reason, the notebooks are both using "we" as a way to make the user feel like he is not alone, even if there is frustration, while trying to understand the concepts.
Example: "Before we head into the world of reinforcement learning, we will have to talk about software agents." - out of notebook 1

Visualization Another important goal is to make the training epochs visible. This was not easy to achieve with the cloud and Jupyter Lab, but the final result lets the user create a gif out of a training episode, which can be viewed after the training. The following picture 4 shows a part of an created gif:

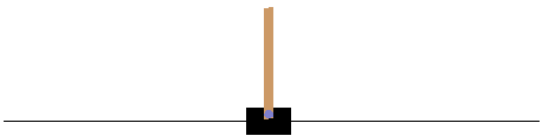

Figure 4: Visualizing the training

5.1.3 Cartpole

The game Cartpole was chosen as the environment for the agent. The goal of Cartpole is to balance a pole on a cart, which can move to the right or left. If the pole tips over the game is lost. This game is an excellent choice for an environment, as it's a really simple game, which is great for explaining things without distracting to much. In image 4 a round of Cartpole can be seen.

5.2 Initial Targets vs. Result

In this section, the initial targets are compared against the final results.

Beginner-Friendly The final product are two notebooks and the installation guide. The theory as well as the practice are explained and code is commented. Comprehensibility is a subjective opinion, but many steps were taken to assure an easy understandable tutorial. The text is accompanied by images, to illustrate explanations. There are many code examples and exercises written in Python. Finally, there is an extensive installation guide to install the whole tutorial.

Topic The initial topic was fulfilled in the final product with two notebooks, one dedicated to theory and one to practice.

Relevant Knowledge As discussed, the basics of RL are explained in the first notebook.

Easy Code As the target for the tutorial is to teach the reader how to use RL, there need to be some code examples. For this an easy programming language is needed, which is relevant for RL and not just used for examples. The code should also be commented and explained.

Combination of text and code With the usage of Jupyter Lab, code and text are successfully combined into one document.

Performance The usage of Google Cloud enables the reader to use a fully powered server to execute the agent. This target was also fulfilled.

Budget-Friendly & Accessible With the increased performance, which is reached by using Google Cloud as a server provider, the tutorial may cost a user small amounts of money, if the tutorial is run in the cloud. But as the tutorial can also be executed locally on a personal computer, this is not much of a constrain. If a user really wants to dive into the world of RL anyway, powerful hardware is needed, so this is more of a constraint of RL in general then of this tutorial.

5.3 Further Improvements

In this section, problems with the result are discussed and solutions are offered.

5.3.1 Policy Learning

The notebooks only cover value learning, this should be extended to also include policy learning, as this is a more advanced technology to train an agent. This would improve both notebooks. In notebook 1, policy learning would be introduced and explained. Theoretical advantages and disadvantages would be discussed. In the second notebook, there would be another version of the implemented agent, that would be using an implementation of policy learning. Finally, both agents would be compared and evaluated.

5.3.2 More exercises

At the moment, there are only three exercises for the reader to complete. In the future, there should be more exercises to help with the understanding as well as extending the knowledge of the reader.

5.3.3 More code & libary explanation

Another useful extensions of the tutorial would be to show more code and explain a library, which already implemented algorithms like DQN.

5.4 Conclusion

The authors were able to create a beginners tutorial, which will hopefully help many people to make their first steps into the existing world of *deep reinforcement learning*.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[2] Garychl, "Applications of reinforcement learning in real world."

[3] G. Zheng, F. Zhang, Z. Zheng, Y. Xiang, N. J. Yuan, X. Xie, and Z. Li, "Drn: A deep reinforcement learning framework for news recommendation," in *Proceedings of the 2018 World Wide Web Conference*, WWW '18, (Republic and Canton of Geneva, Switzerland), pp. 167–176, International World Wide Web Conferences Steering Committee, 2018.

[4] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.