Controlling Self-Driving Race Cars with Deep Neural Networks

University of Osnabrück

Department of Neuroinformatics

Bachelor's Thesis

Author:
Christoph Stenkamp

Supervisors:
Prof. Dr. Gordon Pipa
Leon Sütfeld

Osnabrück,
August 10, 2017

Abstract

This Thesis will be written in the next two months, and I'm pretty scared about that.

Preface

This document was written as the author's bachelor thesis at the department of neuroinformatics at the University of Osnabrück during summer 2017 and is an original and independent work by the author Christoph Stenkamp.

Christoph Stenkamp
Osnabrück, August 10, 2017

Acknowledgements

Thanks to my parents, Marie, my supervisors, and my friends. . . .

"There are no surprising facts, only models that are surprised by facts; and if a model is surprised by the facts, it is no credit to that model."

Eliezer Yudkowsky

Contents

Abstract **i**

Preface **ii**

Acknowledgements **iii**

1 Reinforcement Learning **1**
1.1 Reinforcement Learning Problems 1
1.2 Temporal difference Learning 3

2 Chapter Title Here **5**
2.1 Main Section 1 5
2.1.1 Subsection 1 5
2.1.2 Subsection 2 5
2.2 Main Section 2 6

3 Program Architecture **7**
3.1 Program Design 7
3.1.1 Subsection 1 7
3.1.2 Subsection 2 7
3.2 Main Section 2 7

A Frequently Asked Questions **8**
A.1 How do I change the colors of links? 8

Declaration of Authorship **9**

List of Figures

List of Tables

List of Algorithms

1 How to write algorithms 5

List of Abbreviations

The abbreviations used throughout the work are compiled in the following list below. Note that the abbreviations denote the singular form of the abbreviated words. Whenever the plural forms is needed, an s is added. Thus, for example, whereas ANN abbreviates *artificial neural network*, the abbreviation of *artificial neural networks* is written ANNs.

ANN	**A**rtificial **N**eural **N**etwork
CNN	**C**onvolutional (artificial) **N**eural **N**etwork
CPU	**C**entral **P**rocessing **U**nit
DDPG	**D**eep **D**eterministic **P**olicy **G**radient - Network
DQN	**D**eep-**Q**-**N**etwork
GUI	**G**raphical **U**ser **I**nterface

List of Symbols

For my friends, family, and especially Marie.

Chapter 1

Reinforcement Learning

As the task as hand was not only to provide a reinforcement learning agent, but also to convert a game itself into something the agent can successfully play, I will in this chapter go into detail about Reinforcement Learning in general, to give insights into why I did what I did. Also, I will try to keep this stuff as general as possible, getting into detail when speaking about the used algorithms. [The sense of this chapter is to give an intro of MDPs and RL. It shall also go into enough details on how to specify an MDP such that an RL agent can learn on it, because a big part of the work was exactly that. It's supposed to end with SARSA and Q-learning as the two Ideas on how to perform RL]

1.1 Reinforcement Learning Problems

Machine Learning can mainly be subdivided into three main categories: Supervised Learning, Unsupervised Learning, and Semi-supervised learning. The first deals with direct classification or regression using labelled data (i.e. it uses pairs of datapoints with their corresponding category or value). In unsupervised learning, no such label exists, and the data must be clustered into meaningful parts without any knowledge, by for example grouping objects by similarity of their properties.
What will be mainly considered in this thesis will be a certain kind of semi-supervised learning: *Reinforcement learning*. In Reinforcement Learning (**RL**), instead of labels for the data, there is a *weak teacher*, which provides feedback to the actions the agent took.

The metaphor behind RL is that of an *agent* and an *environment*. The agent makes observations in the environment (its input), takes actions (output) and receives rewards. In contrast to the classical ML approaches, in RL the agent is also responsible for exploration, as he has to acquire his knowledge actively. Thus, a reinforcement learning problem is given if the only way to collect information about the model (the environment) is by interacting with it. As the environment does not explicitly provide actions the agent has to perform, its goal is to figure out the actions maximizing its cumulative reward until a training episode ends.

In the classical RL approach, the environment is divided into discrete time steps. If that is the case, the environment corresponds to a *Markov Decision Process* (**MDP**). Formally, a MDP is a 5-tuple $< S, A, P, R, \gamma >$, consisting of the following:

$$
\begin{aligned}
S &- \text{set of states } s \in S \\
A &- \text{set of actions } a \in A \\
P_a(s, s') &- \text{transition probability function from state } s \text{ to state } s' \text{ under action } a \\
R_a(s, s') &- \text{ reward function for action } a \text{ in state } s \text{ if the environment moves to } s' \\
\gamma &- \text{discount factor for future rewards}
\end{aligned}
$$

Though in general both the state transition function and the reward function may be indeterministic, I will refer to the result of a state transition at discrete point in time t as $s_{t+1} := P(s_t, a_t)$ and to the result of the reward function as $r_t := R(s_t, a_t)$. If no point in time is explicitly specified, it is assumed that all variables use the same t.

While an *offline learner* gets as input the problem definition in the form of a complete MDP, where the only task left is to classify actions yielding high rewards from actions giving suboptimal results, the task for an *online reinforcement learning* agent is a lot harder, as it has to learn the MDP itself via trial and error. In the process of reinforcement learning, the agent will encounter states s of the environment, performing actions a. The future state s_{t+1} of the environment may be indeterministic, but depends on the history of previous states s_0 ... s_t as well as the action of the agent a_t. It is assumed that the Markov property holds, which means that a state s_{t+1} depends only on the current state s_t and currenct action a_t.

Throughout interacting with the environment, the agent receives rewards r, depending on his action a as well as the state of the environment s. In many RL problems, the full state of the environment is not known to the agent, and it only perceives an observation depending on the environment: $o_t := o(s_t)$[1]. This is referred to as *partial observability*. Additionally, the agent knows when a final state of the environment is reached, terminating the current training episode. An episode consists for the agent thus of a sequence of observations, actions and rewards:

$$Episode := ((s_0, a_0, r_0), (s_1, a_1, r_1), (s_2, a_2, r_2), ...)$$

.

In the process of reinforcement learning, the agent tries to perform as well as possible in the previously unknown environment. For that, it uses an action-policy, mapping states to actions: $\pi(s) = a$. In general, this policy may also be stochastic. As the agent does not have supervised data for what actions are the ground truth, it must learn some kind of measure for the value of being in a certain state or performing a certain action. The commonly used measure for the value of a state can be calculated by the immediate reward this state gives, summed with the discounted future reward the agent will archieve by continuing to follow his policy from this state on:

$$V^\pi(s_t) := \sum_{t'=t}^{t_t} (\gamma^{t'-t} * r_{t'}) \tag{1.1}$$

Where t_t refers to a terminal state, marking the end of a training episode.
Using the discounted future reward is useful because in an indeterministic environment it gets less likely that the agent actually reaches this state, and to make the agents perform good actions as quickly as possible.

[1]From now on, when I mean the state of the environment, I will explicitly refer to it as s_e, while reserving s for the agent's obvervation of the enviroment $o(s_e)$

The actual, underlying Value of a state s is defined as $V^*(s_t) := max_\pi V^\pi(s_t)$, which is the value of the state using the best possible policy. It can be interpreted as the maximally archievable reward starting in state s. While *passive reinforcement learning* simply tries to learn the Value-function without the need of action selection, an *active reinforcement learner* tries to estimate a good policy, using which those high-value states are actually reached.

If the value of every state is known, then the optimal policy can be defined as the one archieving maximal value for every upcoming state: $\pi^* := argmax_\pi V^\pi(s) \forall s \in S$. However, for an agent, there are two practical problems with this approach, namely that neither the successor function $P_a(s, s')$, nor the reward function $R_a(s)$ are known to the agent. While so-called *model-based* reinforcement learning tries to learn both of those explicitly to reconstruct the entire MDP, *model-free* agents use a different measure of quality: the *Q-value*. It represents the value of performing action a_t in a state s_t, afterwards following the policy π.

$$Q^\pi(s_t, a_t) := r_t + \gamma * V^\pi(s_{t+1}) \tag{1.2}$$

The underlying Q-value Q^* is correspondingly $Q^*(s, a) = r_t + \gamma * V^*(s_{t+1}) = max_\pi(r_t + \gamma * V^\pi(s_{t+1}))$ When an agent knows the Q-value for each action of a state, it can easily infer the optimal action in state s_t as $a_t^* := argmax_{a_t}(Q(s_t, a_t))$ and thus the optimal policy π^*, guaranteeing maximum future reward at every state. The goal of a model-free reinforcement agent is thus to get a maximally precise estimate of Q^*, yielding maximal reward for every state. For that, it does not need to explicitly learn the reward- and transition function, but instead can model only the Q-function. Its policy is then to simply always take the action yielding the highest value for every state. In RL settings with a highly limited amount of discrete states and actions, the respective Q-function estimate can be specified as a lookup table, whereas for areas of interest, the function is calculated using a kind of nonlinear function approximator.

Throughout exploration of the environment, the agent collects more information of it, continually updating its estimate Q^π. For that, it uses samples from the history of the environment

1.2 Temporal difference Learning

Throughout the process of reinforcement learning, the agent continually improves its estimates Q^π of Q^*. An optimal solution would be to calculate the Loss of the current Q-estimate as the squared difference $(Q^\pi\text{-}Q^*)^2$, to perform gradient descent in order to minimize that difference. However, as Q^* is unknown to the that is not possible. Instead, a Q-learning agent performs iterative approximation, using the information about the environment, to continually update its estimates of Q^*.
Because of equation 1.2 and the fact that

$$V^\pi(s_t, a_t) = max_{a_t} Q^\pi(s_t, a_T) \tag{1.3}$$

it is possible to define the Q-function recursively as

$$Q^\pi(s_t, a_t) = r_t + \gamma * max_{a_t+1} Q^\pi(s_{t+1}, a_{t+1}) \tag{1.4}$$

The recursion in this formula allows for a technique called *temporal difference learning*: At time $t + 1$, the agent can compare his estimate of the Q-function of the last step, $Q^{\pi}(s_t, a_t)$, with a new estimate using the new information it gained from the environment: r_{t+1} and s_{t+1}. Because of the newly gained information, the new estimate will be closer to the actual function Q^* than the original value.

The new knowledge about the environment can be incorporated in two different ways. For the first method, the agent samples a full tuple of $< s_t, a_t, r_t, s_{t+1}, a_{t+1} >$ from the environment, to then calculate the temporal difference error at iteration i as $L_i := \big[r_t + \gamma * Q_i^{\pi}(s_{t+1}, a_{t+1}) - Q_i^{\pi}(s_t, a_t))^2$ This algorithm of calculating the temporal difference error is called *SARSA* [SOURCE], and it is an example of *on-policy* learning. In on-policy learning, the agent uses his own policy in every estimate of the Q-value.

In contrast to SARSA stands *Q-learning*. Q-learning does not need to sample the action a_{t+1}, as it calculates the Q-update at iteration i using the best action in state s_{t+1}:

$$L_i := \big[r_t + \gamma * max_{a_{t+1}}(Q_i^{\pi}(s_{t+1}, a_{t+1})) - Q_i^{\pi}(s_t, a_t)\big]^2 \tag{1.5}$$

allowing to calculate a Loss defined as

$$L := r + \gamma max \tag{1.6}$$

which in turn allows for gradient descent, updating the parameters of the Q-function in the direction of the newly acquired knowledge.

Thus, when using the update rule <rule>, the agent gains continually better updates of Q* (as proven in source). The update rule in an online-learning, model-free, active RL agent is Temporal difference learning: the agent compares what he thought would happen with what actually happened. For that,

Unterschied zwischen on-policy und off-policy algorithmen, example für former ist sarsa, example für den Latter is Q-learning, und dabeischreiben dass der Unterschied ist dass off-policy alogirhtmen das argmax nehmen, on-policy algorithmen ihr eigenes resultat, und dann fußnote dass double-q ne slight deviation davon ist.

Chapter 2

Chapter Title Here

2.1 Main Section 1

Data: this text
Result: how to write algorithm with LaTeX2e
initialization;
while *not at end of this document* **do**
    read current;
    if *understand* **then**
        go to next section;
        current section becomes this one;
    else
        go back to the beginning of current section;
    end
end

Algorithm 1: How to write algorithms

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam ultricies lacinia euismod. Nam tempus risus in dolor rhoncus in interdum enim tincidunt. Donec vel nunc neque. In condimentum ullamcorper quam non consequat. Fusce sagittis tempor feugiat. Fusce magna erat, molestie eu convallis ut, tempus sed arcu. Quisque molestie, ante a tincidunt ullamcorper, sapien enim dignissim lacus, in semper nibh erat lobortis purus. Integer dapibus ligula ac risus convallis pellentesque.

2.1.1 Subsection 1

Nunc posuere quam at lectus tristique eu ultrices augue venenatis. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam erat volutpat. Vivamus sodales tortor eget quam adipiscing in vulputate ante ullamcorper. Sed eros ante, lacinia et sollicitudin et, aliquam sit amet augue. In hac habitasse platea dictumst.

2.1.2 Subsection 2

Morbi rutrum odio eget arcu adipiscing sodales. Aenean et purus a est pulvinar pellentesque. Cras in elit neque, quis varius elit. Phasellus fringilla, nibh eu tempus venenatis, dolor elit posuere quam, quis adipiscing urna leo nec orci. Sed nec nulla auctor odio aliquet consequat. Ut nec nulla in ante ullamcorper aliquam at sed dolor. Phasellus fermentum magna in augue gravida cursus. Cras sed pretium lorem. Pellentesque eget ornare odio. Proin accumsan, massa viverra cursus pharetra, ipsum nisi lobortis velit, a malesuada dolor lorem eu [**mnih_human-level_2015**] neque. [**bojarski_end_2016**]

2.2 Main Section 2

Sed ullamcorper quam eu nisl interdum at interdum enim egestas. Aliquam placerat justo sed lectus lobortis ut porta nisl porttitor. Vestibulum mi dolor, lacinia molestie gravida at, tempus vitae ligula. Donec eget quam sapien, in viverra eros. Donec pellentesque justo a massa fringilla non vestibulum metus vestibulum. Vestibulum in orci quis felis tempor lacinia. Vivamus ornare ultrices facilisis. Ut hendrerit volutpat vulputate. Morbi condimentum venenatis augue, id porta ipsum vulputate in. Curabitur luctus tempus justo. Vestibulum risus lectus, adipiscing nec condimentum quis, condimentum nec nisl. Aliquam dictum sagittis velit sed iaculis. Morbi tristique augue sit amet nulla pulvinar id facilisis ligula mollis. Nam elit libero, tincidunt ut aliquam at, molestie in quam. Aenean rhoncus vehicula hendrerit.

Chapter 3

Program Architecture

3.1 Program Design

The program was written by the author of this work and is licensed under the GNU General Public License (GNU GPLv3). Its source code is attached in the appendix of this work and additionally can be found digitally on the enclosed CD-ROM. The machine learning part was written in PYTHON, using the TENSORFLOW-library [**abadi_tensorflow:_2015**].

3.1.1 Subsection 1

Nunc posuere quam at lectus tristique eu ultrices augue venenatis. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam erat volutpat. Vivamus sodales tortor eget quam adipiscing in vulputate ante ullamcorper. Sed eros ante, lacinia et sollicitudin et, aliquam sit amet augue. In hac habitasse platea dictumst.

3.1.2 Subsection 2

Morbi rutrum odio eget arcu adipiscing sodales. Aenean et purus a est pulvinar pellentesque. Cras in elit neque, quis varius elit. Phasellus fringilla, nibh eu tempus venenatis, dolor elit posuere quam, quis adipiscing urna leo nec orci. Sed nec nulla auctor odio aliquet consequat. Ut nec nulla in ante ullamcorper aliquam at sed dolor. Phasellus fermentum magna in augue gravida cursus. Cras sed pretium lorem. Pellentesque eget ornare odio. Proin accumsan, massa viverra cursus pharetra, ipsum nisi lobortis velit, a malesuada dolor lorem eu [**mnih_human-level_2015**] neque. [**bojarski_end_2016**]

3.2 Main Section 2

Sed ullamcorper quam eu nisl interdum at interdum enim egestas. Aliquam placerat justo sed lectus lobortis ut porta nisl porttitor. Vestibulum mi dolor, lacinia molestie gravida at, tempus vitae ligula. Donec eget quam sapien, in viverra eros. Donec pellentesque justo a massa fringilla non vestibulum metus vestibulum. Vestibulum in orci quis felis tempor lacinia. Vivamus ornare ultrices facilisis. Ut hendrerit volutpat vulputate. Morbi condimentum venenatis augue, id porta ipsum vulputate in. Curabitur luctus tempus justo. Vestibulum risus lectus, adipiscing nec condimentum quis, condimentum nec nisl. Aliquam dictum sagittis velit sed iaculis. Morbi tristique augue sit amet nulla pulvinar id facilisis ligula mollis. Nam elit libero, tincidunt ut aliquam at, molestie in quam. Aenean rhoncus vehicula hendrerit.

Appendix A

Frequently Asked Questions

A.1 How do I change the colors of links?

The color of links can be changed to your liking using:

`\hypersetup{urlcolor=red}`, or

`\hypersetup{citecolor=green}`, or

`\hypersetup{allcolor=blue}`.

If you want to completely hide the links, you can use:

`\hypersetup{allcolors=.}`, or even better:

`\hypersetup{hidelinks}`.

If you want to have obvious links in the PDF but not the printed text, use:

`\hypersetup{colorlinks=false}`.

Declaration of Authorship

I, Christoph Stenkamp, hereby certify that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other university.

signature

city, date