

Heidelberg University  
Institute of Computer Science

Project report for the lecture Fundamentals of Machine  
Learning

# Reinforcement Learning for Bomberman

[https://github.com/nilskre/bomberman\\_rl](https://github.com/nilskre/bomberman_rl)

Team Member: Felix Hausberger, 3661293,  
Applied Computer Science  
eb260@stud.uni-heidelberg.de

Team Member: Nils Krehl, 3664130,  
Applied Computer Science  
pu268@stud.uni-heidelberg.de

## Abstract

## Plagiarism statement

We certify that this report is our own work, based on our personal study and/or research and that we have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication.

We also certify that this report has not previously been submitted for assessment in any other unit, except where specific permission has been granted from all unit coordinators involved, or at any other time in this unit, and that we have not copied in part or whole or otherwise plagiarized the work of other students and/or persons.

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Fundamentals and Related Work</b>	<b>2</b>
<b>3</b>	<b>Approach</b>	<b>4</b>
3.1	Reinforcement Learning Method and Regression Model . . . .	4
3.2	Training process . . . . .	4
<b>4</b>	<b>Experimental results</b>	<b>4</b>
<b>5</b>	<b>Conclusion</b>	<b>4</b>
<b>A</b>	<b>Appendix</b>	<b>I</b>
A.1	Appendix A . . . . .	I

## List of Abbreviations

<b>DQN</b>	Deep-Q-Networks
<b>MDP</b>	Markov Decision Process

# 1 Introduction

## 2 Fundamentals and Related Work

Q-Learning is a known off-policy and model-free approach to train an agent based on temporal difference in an environment that can be modeled as a Markov Decision Process (MDP). An agent therefore does not necessarily use the policy it is trained for and does not know the transition probabilities and rewards in the MDP beforehand. Equation 1 shows the iterative update formula for the Q-values that an online model uses to choose the right action [1].

$$Q_{k+1}(s, a) = (1 - \alpha)Q_k(s, a) + \alpha(r + \gamma \max_{a'} Q_k(s', a')) \quad (1)$$

The problem with conventional Q-Learning is that in most of the cases the state dimension is far too high to explore and model the MDP entirely in foreseeable future. To deal with this problem the Q-values need to be approximated using a regression model. Deep neural networks have proven to be highly applicable for this task, which leads to the term of *Deep-Q-Learning* and respectively *Deep-Q-Networks* (DQN) for such network architectures. DQNs use the vectorized numerical state as its input and outputs the predicted Q-values. It learns through backpropagating the temporal difference error over each step for a single neuron. Note that for the Q-Learning algorithm a backpropagation is done after every step of the simulation. To avoid correlation between succeeding experiences an experience buffer is used to randomly sample a training batch in each step. In the following papers regarding DQN architectures shall be introduced as well as papers dealing with the bomberman environment for reinforcement learning.

Paper [2] tackles the problem that the *max* operator in 1 often leads to overoptimistic value estimates as the DQN uses the same Q-values to both select and evaluate an action. It therefore changes the iterative update formula to 2.

$$Q_{k+1}(s, a) = (1 - \alpha)Q_k(s, a) + \alpha(r + \gamma Q'_k(s', \arg \max_{a'} Q_k(s', a'))) \quad (2)$$

Now a target DQN is used to separate the determination of the greedy policy, which is still done by the online network, from the Q-value estimation. Using this double DQN approach results in less overestimated Q-values and therefore better policies by more accurate Q-value estimates. It also makes the learning process more stable and reliable. The weights are copied from the online network to the target network after a fixed amount of episodes.

Another optimization was introduced in paper [4]. It changes the architecture of the DQN by splitting it into two separate value streams. One stream estimates the state value function and the other one the state-dependent action advantage function. Both streams are then combined again using a special aggregating layer to produce an estimate of the Q-values (see Figure 1).

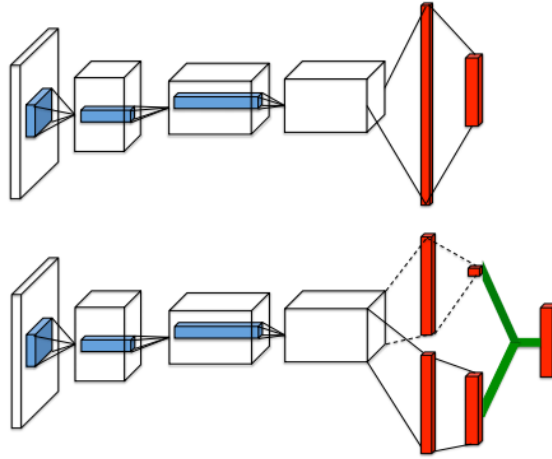


Figure 1: Architecture of a normal DQN (top) compared to a dueling DQN (bottom)

The state-dependent action advantage function is defined as

$$A^\pi(s, a) = A^\pi(s, a) - V^\pi(s) \quad (3)$$

and measures the importance of each action. The special aggregating layer uses ?? to estimate the Q-values while also tackling the issue of identifiability.

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + (A(s, a; \theta, \alpha) - \frac{1}{|A|} \sum_{a'} A(s, a'; \theta, \alpha)) \quad (4)$$

Using the dueling architecture approach an agent can learn which states are valuable independently of each action, which is especially useful in case actions do not influence the environment in any useful way. This leads to a better policy evaluation in the presence of many similar-valued actions. Also when looking at the last layer of a conventional DQN (see Figure 1) it usually becomes much more sparse and biased. As the state value is modeled as a

single neuron in the dueling architecture, learning the state value function becomes much more efficient.

There are several other improvements mentioned in the rainbow paper [3] like multi-step learning, distributional reinforcement learning and noisy nets, but these are out of the scope for this small research project.

## **3 Approach**

### **3.1 Reinforcement Learning Method and Regression Model**

- Our model
- Our exploration method

### **3.2 Training process**

- Experience buffer

## **4 Experimental results**

## **5 Conclusion**

# A Appendix

## A.1 Appendix A



## References

- [1] Aurélien Géron. *Praxiseinstieg Machine Learning mit Scikit-Learn und TensorFlow*. O'REILLEY, 2018. ISBN: 978-3-96006-061-8.
- [2] Hado van Hasselt, Arthur Guez, and David Silver. “Deep Reinforcement Learning with Double Q-learning”. In: *arXiv* (2015). URL: <https://arxiv.org/abs/1509.06461>.
- [3] Matteo Hessel et al. “Rainbow: Combining Improvements in Deep Reinforcement Learning”. In: *arXiv* (2017). URL: <https://arxiv.org/abs/1710.02298>.
- [4] Ziyu Wang et al. “Dueling Network Architectures for Deep Reinforcement Learning”. In: *arXiv* (2016). URL: <https://arxiv.org/abs/1511.06581>.