Final Report - Using Reinforcement Learning To Play Bomberman

Elias Rieger (sv235)
elias.rieger@stud.uni-heidleberg.de
Etienne Stock (uc235)
etienne.stock@stud.uni-heidelberg.de
Philipp Nazari (iy232)
philipp.nazari@stud.uni-heidelberg.de

March 17, 2022

Abstract

We trained and tested two different models to do reinforcement learning on the videogame classic "Bomberman". The first uses a simple deterministic Q-table while the second consists of a Regression Forest. Both of our models work with hand crafted features, as this turned out to be the superior approach to f.e. PCA. We found that while the first model is superior to the second one, it is limited in it's capibilities as it makes decisions very deterministically. It does not have much freedom in finding it's way of playing and reading the game.
We also tried different learning algorithms like simple Q-learning, n-step-Q-learning and SARSA-learning, and went with simple Q-learning at the end as it gave the best results.

1 Introduction

TODO: HIER ÜBERALL SO GEILE VERWEISE EINFÜGEN, SO PROFESSIONELL MIT e.g. [5].

Reinforcement learning is a branch of machine learning which is used to learn models that can interact with, and change the environment. A Reinforcement Learning model is active, it observes its environment, makes predictions about its future state and takes corresponding actions to adapt to those predictions. At any time, which is practically quantized, the model is in a certain state. After one time quantum elapsed, the model can take a new state. The central goal of such a model trained using Reinforcement Learning is to maximize its so called "value function" or "Q function", which measures the amount of success a model will have when doing a state transformation.

TODO: WENN WIR PLATZ FÜLLEN MÜSSEN: HIER EIN BISSCHEN THEORIE MACHEN, Z.B. BELLMANN EQUATION USW.

In our application we will use Reinforcement Learning to train two agents to play the game Bomberman (1983). To do so we implemented two different models; the first one uses a simple Q-Table with handcrafted features, the second one a Regresson Forest, also with handcrafted features.

Our approach was to not directly tackle the problem of winning the game against arbitrary opponents, but rather tackled the task stept by step. Our agents went through three phases; in the first phase it focused on collecting coins on a crate- and

opponent-free map. In the second phase it then had to use bombs to uncover coins from crates. Once an agent mastered this task it advanced into the third phase, in which he had to play against and kill opponents.

In Section 2 we will first give a detailed overview over the models we used, the features we designed and the rewards we shaped. In the following sections 3 and 4 we will then go on to talk about our different training and testing strategies and approaches. Here we will set a focus on the way we conducted performance analysis by generating detailed statistics in the different phases our models went through. We will also touch the important topic of hyper parameter optimization.

2 Model Setup

2.1 Model Design

2.2 Feature Design

We tried our luck with PCA, but as it turns out the linear dimension reduction was not powerfull enough for our applications. Since we achieved very good results with our hand crafted features, we decided to just go with those.

We used symmetries!!

2.3 Reward Shaping

3 Training

Trainingsmethoden beschreiben. Also erst deterministisch, dann exploration phase und dann normale exploitation.

We tried Q-learning, n-step-Q-learning and SARSA learning It went...

We used the model trained for the old task and trained it also for the new task...

3.1 Statistics and Analysis

Für die bedachte und nachhaltige Entwicklung unserer Agenten hat es sich als äußerst wichtig herausgestellt, das Training in den unterschiedlichen Stadien mit Statistik zu analysieren. Die visuelle Darstellung ermöglicht eine vereinfachte Analyse und zielgerichtete Behebung von Fehlern.

Um die notwendige Statistik zu generieren, auf die wir in den nächsten Absätzen noch genauer eingehen werden, haben wir uns dazu entschieden wichtige Spielinformationen wie beispielsweise den score am Ende eines Spiels oder die am Ende einer Runde ausgeführten events zu loggen, und die logfiles dann mit einem script (zunächst mit Hilfe von Bash, dann von Python) zu analysieren und die wichtigen Informationen zu extrahieren. Am Ende einer Trainingssession werden dann die entsprechenden Grafiken erstellt. Dabei handelt es sich unter anderem um eine Reihe von PyCharts, die für jedes Spiel die action-distribution darstellen und so einen schnellen Überblick über die relativen action-Häufigkeiten ermöglichen (figure ???). So lässt sich beispielsweise beobachten, dass die relative Anzahl der illegal moves mit fortschreitendem training abnimmt (figure ???)

Da jedoch nicht nur die relative Häufigkeit von bestimmten Aktionen interessant sein kann, sondern auch das Verhalten der absoluten Häufigkeiten im Verlauf mehrerer Spiele, werden auch hier entsprechende Grafiken erstellt. In Figure ??? wird eine Übersicht dieser Grafiken gezeigt.

Um den Erfolg unseres Agenten zu tracken werden außerdem Grafiken erstellt, die Platzierung und finalen score zeigen.

Dies vorgestellten Statistiken verwenden wir dann auch, um Hyperparameter optimization zu betreiben. Wir optimieren die ... alpha sowie die ... gamma. Für unterschiedliche Konstellationen werden wir Platzierung und finalen score nach einer trainingsperiode und dann einer testperiode untersuchen, um die optimalen Parameter zu finden.

3.2 Symmetries

4 Testing

4.1 Statistics and Analysis

4.2 Hyperparameter Optimization

Durch Berücksichtigung der vierzähligen Rotationssymmetrie (as explaned in Section **??**) können wir die Q-Table mit vierfacher Geschwindigkeit updaten. Dies ermöglich ein vierfach schnelleres Training, sodass wir uns dafür entschieden haben mit brute force alle möglichen Kombinationen von gamma und alpha durchzuprobieren, wobei gamma und alpha auf einem grid auf dem Intervall [0, 1] liegen dessen spacing wir vorher festlegen.

Wir trainieren nun für jede der berücksichtigten hyperparemeter-Konstellationen ein Modell und testen dieses danach über eine feste anzahl von Spielen. Dann berechnen wir die durchschnittliche Positionierung des Agenten während der tests und wählen jene Hyperparameter, die die durchschnittliche Positionierund minimieren. Im Falle einer Ambiguität

wählen wir die Konstellation, die den durchschnittlichen score maximiert.

Hier gibt es theoretisch noch Verbesserungsbedarf. Anstatt den durchschnittlichen score nur im Falle eines Unentschiedens nach mittlerer Positionierung zu berücksichtigen, könnte man eine gewisse range an durchschnittlichen Positionierungen definieren, die dann auf den durchschnittlichen score untersucht werden. Denn eine sehr wenig schlechtere durchschnittliche Positionierung kann durchaus Zufall sein, obwohl der er erreichte score überdurchschnittlich groß ist. Aus Zeitgründen haben wir uns diesem Problem zunächst nicht gewidmet, da es andere dringendere Probleme gab.

5 Conclusion

The first model is superior to the second one, but also strictly limited by the human integrating it. We thus very likely just found a local optimum and didn't really use machine learnings full potential. It would thus be interesting to do some further work on less deterministic/rule based agents in the future.