Fundamentals of Machine Learning - Report

Kevin Stein, Lukas Winiwarter

March 15, 2019

The code can be found .... We used Neural Networks and decided for the Actor Critic Algorithm.

Contents

1 Introduction

2 Implementation

2.1 Deep Q Learning

https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html

2.2 Actor Critic

https://github.com/pytorch/examples/blob/master/reinforcement_learning/actor_critic.py

2.2.1 State Representation

All in one Matrix

Binary mapping

2.2.2 Network Architecture

3 Training the Agent

We trained our agent in several sub scenarios of the final game, according to the suggestions on the exercise sheet. Each scenario was played until the agent had a high percentage of a successful game, where all coins were collected or the round was survived. For each scenario the rewards were adjusted to enforce wanted behavior. A summary of all rewards is shown in Table 3.

Scenario 1: Empty map with coins

We started training on an empty field, where the coins are directly visible. The goal of this scenario is to enable the agent to understand the map and find valid actions. Visible coins should be collected and the round survived. We therefore reward feasible actions with a positive reward and punish invalid actions. Waiting is punished as well to encourage movement. The collection of a coin is the main goal in this scenario and therefore highly rewarded. At this stage, the placement of a bomb is tricky to handle. Even though, there is no need for bombs and they bare a high chance of self killing, we need them later in the game. Punishing a self kill teaches our agent not to use bombs at all and it won't learn it again. To solve this, we award points for explosions and deduct them again if the agent is killed by the very same. Hence, every time the agent manages to outrun an explosion it is rewarded. The goal is to create an agent that is highly aggressive and makes excessive use of bombs, while outrunning the explosions. Prediciting and avoiding explosions is a crucial skill to master the game.

This scenario was also used to test different network architectures, algorithms and state representations. We spend approximately half of the total time of the project on this scenario. In the beginning it was unclear, weather our implementation was correct, the network architecture capable of playing the game and the rewards suitable for the task. Especially the number of rounds the agent needed to learn something was unpredictable, so we might have abandoned possibly working, but slow approaches. We started with the Deep-Q-Learning approach but did not see any agent that was able to produce the wanted behavior. First, there are many parameters to tune other than finding the right rewards: `Learning_Rate` , `Batch_Size`, `Gamma`, iterations after which the target net is updated (`Target_Update`), and the exploration / exploitation parameter ϵ. Especially the static randomness in this approach made it hard for the agent to outrun bombs, since this does not allow for errors. Different to the Actor-Critic implementation, where known situations lead to certain decisions and unknown situations are explored randomly, ϵ is static for a whole game. So having a high ϵ leads to not being able to outrun bombs and by the time ϵ becomes lower, every bomb so far has killed the agent and it won't use bombs anymore. In the end every agent we tried ended in this state where no more bombs would be planted.

Hence, we implemented the Actor-Critic algorithm. This has the advantage that every action is chosen with a certain probability according to the result of the state propagating through the network. More known states lead to sharper probability distributions, while exploration still happens for unknown, new states. Additionally this algorithm does

not rely on a replay memory, which removes the hyper parameters `Batch_Size` and `Target_Update`. We explored this approach and tried different layer sizes of our network. We also changed from the single matrix state representation to the binary representation. Finally, after XXXXX iterations of the game, the network described in XXXXX showed the desired behavior. We did not try Deep-Q-Learning with this Network and the binary state input due to time constraints.

Scenario 2: Crate density: 20 %

Having an aggressive agent, we can now teach it to blow up crates and find coins. We set the density to 20% such that all coins are now hidden. We lower the reward for a feasible action, introduce new rewards for destroying crates and finding coins, and remove the reward for the explosion. Now the agent only receives reward when a crate is in the way of the bomb but not for the explosion itself.

Scenario 3: Crate density: 75 %

In the next Scenario we increase the crate density to the final value. This makes the situation more difficult for the agent, since the space to outrun an explosion becomes very small and has to be considered before placing a bomb. At the beginning of each round, bombs can not be placed in a corner spots anymore. We increased the penalty on self killing and also feasible actions are not rewarded anymore. Waiting now has the same reward as moving. The only positive rewards left are for destroying crates, finding coins and collecting them.

Scenario 4: Game against simple agents

Finally, that our agent is able to navigate the map and collect coins reliably, we introduce opponents. For this task we used the sample agent provided with the exercise. Since our agent is used to avoid explosions it does not get killed easily and only needs to learn to fight back. We move the negative reward from self killed to any case where the agent dies and introduce a high reward for actively eliminating opponents. The reward for surviving a round becomes significant.

Event / Rewards	Scenario 1	Scenario 2	Scenario 3	Scenario 4
Moved	10	1	-1	-1
Waited	-10	0	-1	-1
Interrupted	-10	-10	-10	-10
Invalid Action	-20	-20	-20	-20
Bomb Dropped	10	1	1	1
Bomb Exploded	200			
Crate Destroyed		100	100	100
Coin Found		200	200	200
Coin Collected		1000	1000	1000
Killed Opponent				10000
Killed Self	-10	-200	-350	
Got killed				-350
Opponent Eliminated				
Survived Round	40	40	40	1000

4 Summary and possible improvements