Report

Lecture ,,Fundamentals of Machine Learning" WS 2018

Juan Antonio Ruiz Leal
Carlos Alberto Rios Rubiano
hr229@ix.urz.uni-heidelberg.de
sy226@ix.urz.uni-heidelberg.de

March 20, 2019

Abstract

1 Motivation and overview

2 State

In order to construct the states, we found three set of crucial parameters to describe the situations in the game. The first set, related to the avaliable cells to move, is constructed by means of an array of four boleans.

2.1 States construction:

2.1.1 Avaliable cells array (ACA)

The first set, a boolean one, provides the information of the available cells surrounding the agent. If a bit is on, that means this direction is free to move. We provide some examples for better understanding, the first column indicates the available moves, the second the abbreviation of the direction and the third one the related set. Showed in the next table 1:

Table 1: Avalible moves, abbreviation and related boolean array.

Avalible moves	Abbreviation	First set examples
Up	U	$(1, 0, 0, 0)$
Down	D	$(0, 1, 0, 0)$
Left	L	$(0, 0, 1, 0)$
Right	R	$(0, 0, 0, 1)$
Up/Left	UL	$(1, 1, 0, 0)$
Down/Left	DL	$(0, 1, 1, 0)$
Down/Right	DR	$(0, 1, 0, 1)$
Up/Right	UR	$(1, 0, 0, 1)$
Up/Left/Down	URD	$(1, 1, 0, 1)$
.... etc	.... etc	.... etc

2.2 States construction: Dysfunctional version

In this section we describe the first attempt we made for the construction of the state, which we had several difficulties with, and we did not achieve the expected solution.

The unsuccessful results are due to the selection of the parameters that make up the state. The second array set were calculated from the regions surrounding the agent as well. We realized that these parameters are correlated with each other, therefore we find solutions that never converged. In order to show that, we define the regions definition in the next subsection:

2.2.1 Region definition

In this subsection, we show the definition of the regions. A region is defined based on the current position of the agent and the possible directions where the agent can go to. We defined 8 possible regions, Up, Down, Left, Right, Up/Left, Down/Left, Down/Right and Up/Right. The regions we described before are depicted in figure 1.

2.2.2 Normalized potential rewards

We show the regions regarding the second set, which make up the state. Each element, have the *normalized potential rewards*(NPR) in each region, by mean of the ω_{R_*}. And the weights ω_{R_*}, are elements to measure the potential reward to acquire, in all those eight regions described in figure 1.

Table 2: How looks like the second array. Where the ω_{R_*} is the weight related to the NPR in each region.

Second array of the state:
$(\omega_{R_U}, \omega_{R_D}, \omega_{R_L}, \omega_{R_R}, \omega_{R_{UL}}, \omega_{R_{DL}}, \omega_{R_{DR}}, \omega_{R_{UR}})$

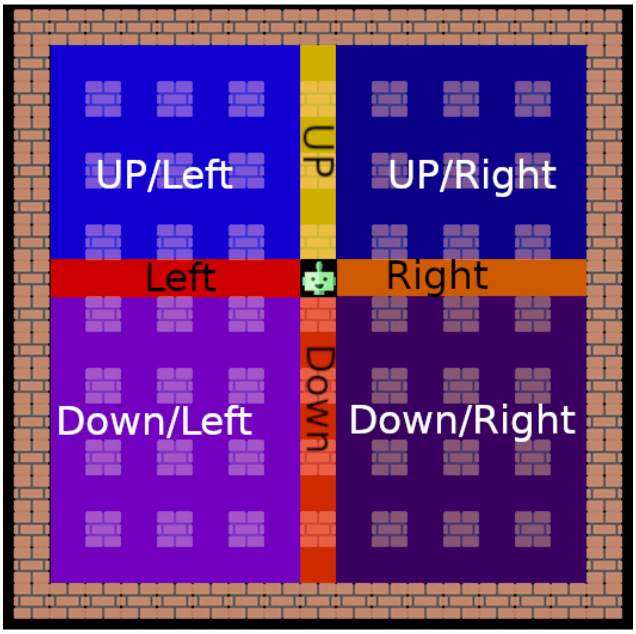

Figure 1: Definition of the regions in the maze.

Where:

$$\omega_{R_*} \in [0, 1] \tag{1}$$

2.2.3 Normalized potential danger

The third set, is similar to the second one. Each element takes into acount the *Normalized Potential Danger* (NPD) in each region. And each weights ω_{D_*}, is a measure of the potential danger in all those eigth regions.

Similary to the table 2, we show how looks like the **NPD** float array.

Table 3: How looks like the second array. Where the $\omega_{R_{D_*}}$ is the weight related to the **NPD** in each region.

How looks like the second array of the state:
$(\omega_{D_U}, \omega_{D_D}, \omega_{D_L}, \omega_{D_R}, \omega_{D_{UL}}, \omega_{D_{DL}}, \omega_{D_{DR}}, \omega_{D_{UR}})$

Where:

$$\omega_{R_*} \in [0, 1] \tag{2}$$

2.2.4 Drop a bomb (DB): Feasibility and usefulness

To complete the state, we additionally add an array of two float values, regarding the situations where is feasible and useful to drop a bomb. The first value is a measure of the surrounding situation with the crates, and the second value is a measure of the proximity of the opponents.

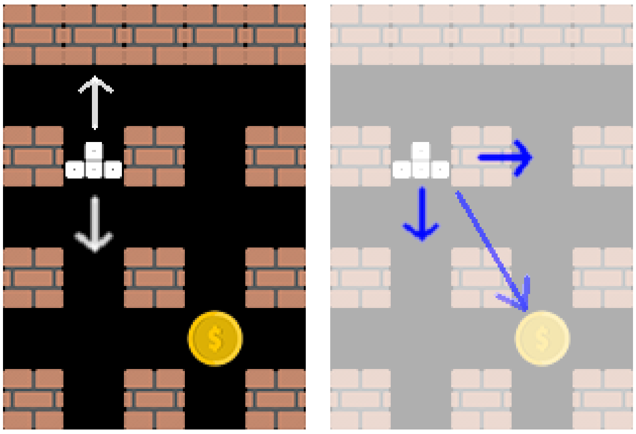

Figure 2: Example of state formation, of ACA and NPR simplified: In the left side the avalible moves is showed, then ACA is: (1100), and in the right side, the direction of probable reward (DPR), and is given by: (0101). from operate with the $\wedge$ operator, we get the $NPR = (1100) \wedge (0101) = (0100)$

Table 4: How looks like the third array. Where the $\omega_{R_{D_{Cr}}}$ is the weight that measure the usefulness to get coins by blowing up crates, and $\omega_{R_{D_O}}$ the feasible of killing an opponent

How looks like the fourth array of the state:
$(\omega_{B_{Cr}}, \omega_{B_O})$

Where:

$$\omega_{B_*} \in [0, 1] \tag{3}$$

2.2.5 Summary of state components

In order to summarise the construct of the state, we describe the features selected. Which is made up of the four arrays shown in tables 1,2,3 and 4:

2.3 Second attempt: Simplifying the state

In order to avoid correlated features, and achieve a functional version to complete the task 1 (2.4). We simplify the NPR, in a binary version.

The new NPR array is made up by four Booleans, where we calculate the nearest coin direction to operate the ACA with.

Using the *AND* operator $\wedge$. We construct the *NPR* array, a picted example is showed in figure 2. And in case of $ACA \wedge DPR = (0, 0, 0, 0)$, we chose randomly the *NPR* form the composed *ACA* possibilities

Table 5: State summarized: Unsuccessfully attempt

Abbreviation	Description	Elements	Info	Type
ACA	Avalible cells	4	Directions	boolean
NPR	Potential rewards	8	Regions	float
NPD	Potential danger	8	Regions	float
DB	Feasibility of throwing a bomb	4	Info crates and opponents	float

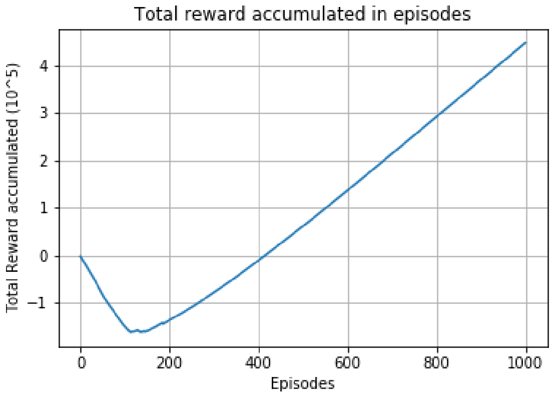

Figure 3: Total reward accumulated in episodes

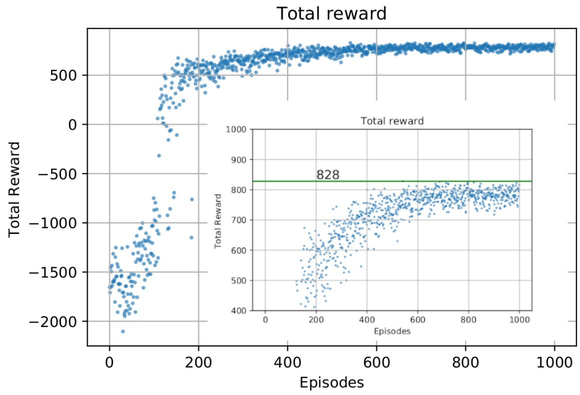

Figure 4: Total reward in episodes. Plot inside: detailed behavior (in order to show the total reward tendence))

2.4 Task 1:

On a game board without any crates, collect a number of revealed coins as quickly as possible. This task does not require dropping any bombs. The agent should learn how to navigate the board effciently.

For this task we use the state made up by the **ACA** array (see section 2.1.1) and the simplified **NRP** (see second 2.3). Summarised by the table 6.

For this task, the agent must to learn two diferent skills. The fisrt one, is the skill to avoid the invalid actions, and the second one is to chose effciently the path to the coins. In order to measure those skills, we show

2.5 Task 2:

On a game board with randomly placed crates, find all hidden coins and collect them within the step limit. The agent must drop bombs to destroy the crates. It should learn how to use bombs without killing itself, while not forgetting eficient navigation.

We tried to face the task 2 with the same strategy, but the problem was that when we performed the AND operator over the ACA and DPR the result could be $(0, 0, 0, 0)$, because the nearest coin could be in a region where the direction was blocked, which again caused that the model did not converge to a desired result.
We solved this problem by means of getting the nearest coin with respect to an unblocked direction and redefining DPR by turning on the bit in this direction.

Another problem we found when solving the task 2 was that we could not punish too much the action of killed by itself, because since the agent could die by dropping a bomb the agent avoided that action, which leaded to a situation where the agent survive the entire episode but without collecting any coin. So we decided to does not punish the action of killed by itself.

3 Model

We basically tried to use 2 models, the normal Q learning and the deep Q learning. The model used for the training was constructed based on the algorithm known as Deep Q Learning [1]

3.1 Algorithm Deep Q Learning

```
// Code
    stringToMatch = 'Score'
    matchedLine = ''

    listScore = []
    listPasos = []
    listEpsil = []
```

Table 6: State summarized

Abbreviation	Description	Elements	Info	Type
ACA	Avalible cells	4	Directions	Boolean
NPR simplified	Nearest coin	4	Directions	Boolean

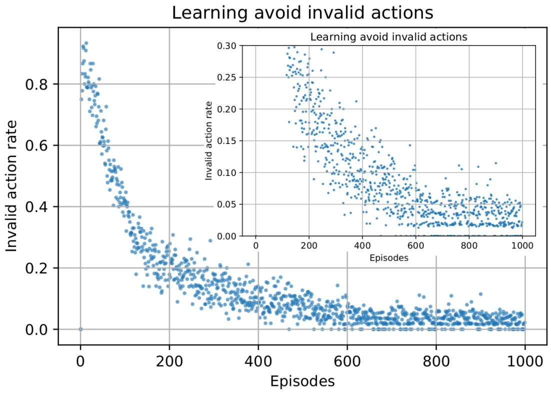

Figure 5: Meassuring the learning to avoid invalid actions. Plot inside: detailed behavior (in order to show the tendence))

```
listRewaA = []
listQMean = []
```

References

[1] Mnih, Volodymyr ; Kavukcuoglu, Koray ; Silver, David ; Graves, Alex ; Antonoglou, Ioannis ; Wierstra, Daan ; Riedmiller, Martin A.: Playing Atari with Deep Reinforcement Learning. In: *CoRR* abs/1312.5602 (2013). http://arxiv.org/abs/1312.5602 3