Deep Q-Learning

The update rule of the Q-learning algorithm is given by

$$Q_{updated}(s,a) = Q(s,a) + \alpha\left[R(s,a) + \gamma\, max_{a'}Q\left(s'\right) - Q(s,a)\right] \tag{1}$$

with learning rate α, discount factor γ and the actual reward $R(s,a)$. The action-value function $Q(a,s)$ returns the expected reward of the model choosing a specific action at a given state.

Since the term $max_{a'}Q\left(s'\right)$ determines the action-value function of the current state, the difference of the target $Y = R(s,a) + \gamma\, max_{a'}Q\left(s'\right)$ and the predicted value $Q(s,a)$ describes the deviation of the momentary best choice from the prediction of the current model. The goal is to minimize this temporal difference error. This can be reached e.g. by minimizing the squared loss $(Y - Q(s,a))^2$. [7]

The state-value function $Q(s,a)$ is represented by a table with dimensions $|s|\times|a|$. Since in practise there are too many possible states to store the whole matrix or to get enough training data, the action-value function can instead be learned by regression. For function approximation we use a neural network and the error is minimized by gradient descent. [5]

Training

Before or while he learns to bomb crates away, the agent has to learn not to blow himself up.

Related Work

Learning how to play Bomberman with Deep Reinforcement and Imitation Learning. Ícaro Goulart, Aline Paes, and Esteban Clua, 2019 [3]

Outplaying humans in bomberman using Deep Q-learning. Frank de Morrée, Ida Stoustrup and Iestyn Watkin, 2019 [2]

Multi-Agent Strategies for Pommerman. Dhruv Shah, Nihal Singh, Chinmay Talegaonkar, 2019 [8]

Pommerman Fight: A Detailed Analysis of Reinforcement Algorithms. Bin Chen, Jiarong Qiu, Sairam Kamal Raj, Shuwei Shi, Wei Cheng, 2020 [1]

Beating Bomberman with Artificial Intelligence. Juarez Monteiro, Roger Granada, Rafael C. Pinto, Rodrigo C. Barros, 2018 [6]

Exploration Methods for Connectionist Q-Learning in Bomberman. Joseph Groot Kormelink, Madalina M. Drugan and Marco A. Wiering, 2018 [4]

MORE GENERAL

[9]

Notes: What we might do to improve our implementation

→ Jetzt und/ oder Verbesserungsvorschläge in Diskussion

Possible network adjustments

- Normierungen innerhalb des Netzes, good feature design (perceptive field figure 1, or multiple layers for whole game field containing Walls, Bricks, Players, Danger(Bomb+Explosion), Enemies, Coins)

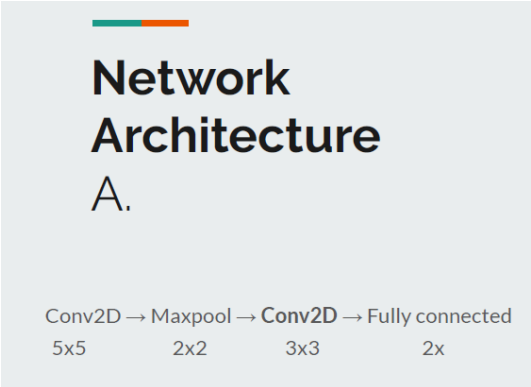
Figure 1: Feature design with perceptual field in [8]

Feature	Dimension
Enemy positions	$p \times p$
Rigid wall blocks	$p \times p$
Wooden wall blocks	$p \times p$
Passage blocks	$p \times p$
Bomb life	$p \times p$
Active flame blocks	$p \times p$
Bombs	$p \times p$
Power-ups	$p \times p$
Ammunition count	1
Bomb strength	1
Kick	1
Total	$(p \times p \times 8) + 3$

Table 1. Choice of features for a Pommerman agent with a perceptual state of size $p \times p$

- Lecture Notes: feature pre-processing via PCA
- dropout algorithm
- different activation function
- more layers, smaller getting layers (how others did in Figure 2 and 3)

Figure 2: Architecture from [2]



Rewards

(how others did in Figure 4, 5)

Reward clipping

"How do we assign rewards? Reward assignment varies for each game. In some games, we can assign rewards such as +1 for winning, -1 for loss, and 0 for nothing, but in some other games,

Figure 3: Architecture from [8]

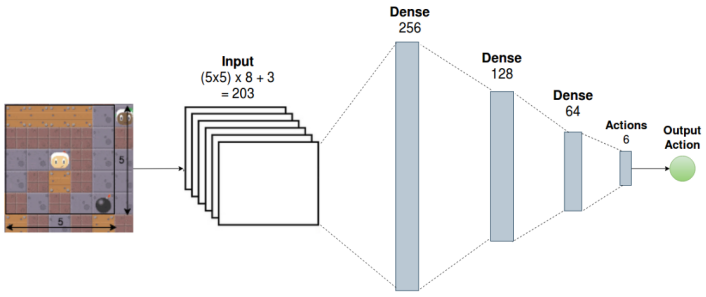


Figure 4: From [3]

Table 1: Reward Function

Event	Reward
Kill a player	100
Break a wall	30
Perform action	-1
Perform impossible action	-2
Die	-300

Figure 5: From [2]

```
self.rewards = {
            "kaboomed_brick":       50,
            "kaboomed_player":      200,
            "lost_life":            -45,
            "died":                 0,
            "invalid_move":         -5,
            "spawned_bomb":         -1,
            "do_nothing":           -3,
            "valid_move":           -1,
            "invalid_spawn_bomb":   -5 }
```

we have to assign rewards such as + 100 for doing an action and +50 for doing another action. To avoid this problem, we clip all the rewards to -1 and +1." [7]

Target network

"We are using the same Q function for calculating the target value and the predicted value. In the preceding equation, you can see the same weights are used for both target Q and predicted Q. Since the same network is calculating the predicted value and target value, there could be a lot of divergence between these two. To avoid this problem, we use a separate network called a target network for calculating the target value." [7]

Bibliography

[1] Bin Chen, Jiarong Qiu, Sairam Kamal Raj, Shuwei Shi, and Wei Cheng. Pommermanfight: A detailed analysis of reinforcement algorithms.

[2] Frank de Morrée, Ida Stoustrup, and Iestyn Watkin. Outplaying humans in bomberman using deep q-learning, 2019.

[3] Ícaro Goulart Faria Motta França, Aline Paes, and Esteban Clua. *Learning How to Play Bomberman with Deep Reinforcement and Imitation Learning*, pages 121–133. 11 2019.

[4] Joseph Groot Kormelink, Madalina Drugan, and Marco Wiering. Exploration methods for connectionist q-learning in bomberman. 01 2018.

[5] Ullrich Köthe. Fundamentals of machine learning, 2021.

[6] Juarez Monteiro, Roger Granada, Rodrigo Barros, and Rafael Pinto. Beating bomberman with artificial intelligence. 10 2018.

[7] Sudharsan Ravichandiran. *Hands-on reinforcement learning with Python*. Packt Publishing, Birmingham, UK, 2018. Includes bibliographical references. - Description based on online resource; title from title page (Safari, viewed August 2, 2018).

[8] Dhruv Shah, Nihal Singh, and Chinmay Talegaonkar. Multi-agent strategies for pommerman, 2017.

[9] Jordi Torres. Deep q-network (dqn)-i, 2020.