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 $(Y - Q(s,a))^2$. [4]

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. [3]

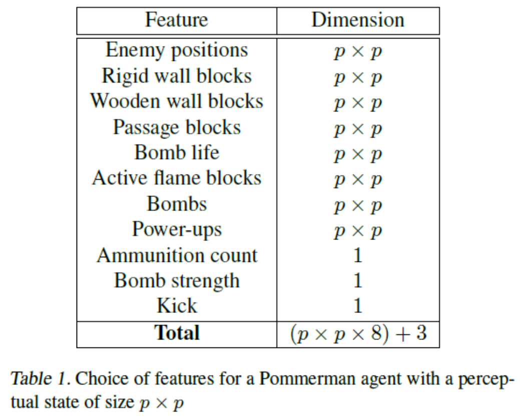
Notes: What we might do to improve our implementation

→ Jetzt und/ oder Verbesserungsvorschläge in Diskussion

Possible network adjustments

- Ich glaub convolutional layers und Normierungen ergeben keinen Sinn. Es ist eher wichtig, den feature Vektor von Hand gut zu designen

Figure 1: Feature design with perceptual field in [5]

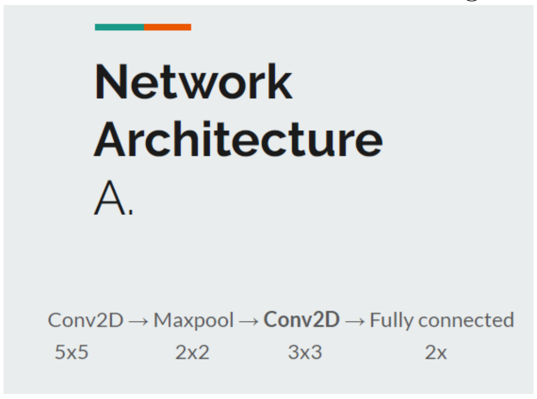


Feature	Dimension
Enemy positions	$p \times p$
Rigid wall blocks	$p \times p$
Wooden wall blocks	$p \times p$
Passage blocks	$p \times p$
Bomb life	$p \times p$
Active flame blocks	$p \times p$
Bombs	$p \times p$
Power-ups	$p \times p$
Ammunition count	1
Bomb strength	1
Kick	1
Total	$(p \times p \times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

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 [1]



Network Architecture

A.

Conv2D 5x5 → Maxpool 2x2 → **Conv2D** 3x3 → Fully connected 2x

Rewards

(how others did in Figure 4)

Batch choice

Do we want to sample some *random* batches of transitions from the buffer (batch size of Atari network is 48, # episodes = 800, learning rate = 0.001, discount factor = 0.97, input size = (88, 80, 1))? And do we want to do that only if the round (episode, = player dead) is finished?

Figure 3: Architecture from [5]

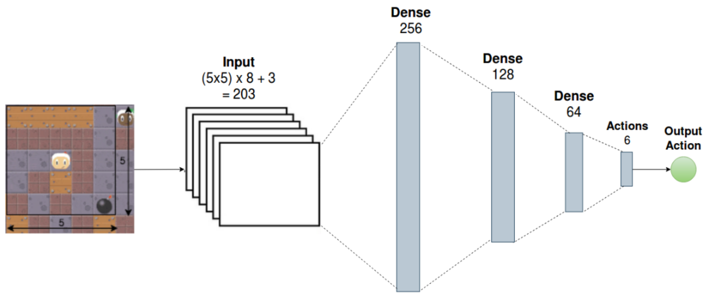


Figure 4: From [2]

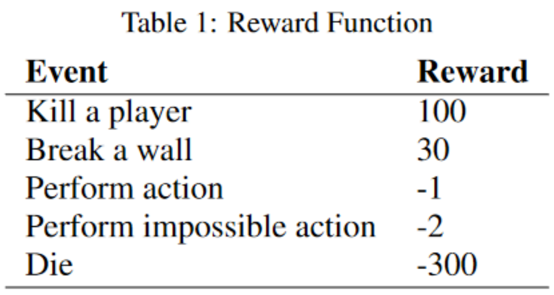


Table 1: Reward Function

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

Lecture notes: We need to sample a batch!

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, 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." [4]

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." [4]

Bibliography

[1] F. de Morrée, I. Stoustrup, and I. Watkin. Outplaying humans in bomberman using deep q-learning, 2019.

[2] c. Goulart Faria Motta França, A. Paes, and E. Clua. *Learning How to Play Bomberman with Deep Reinforcement and Imitation Learning*, pages 121–133. 11 2019.

[3] U. Köthe. Fundamentals of machine learning, 2021.

[4] S. 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).

[5] D. Shah, N. Singh, and C. Talegaonkar. Multi-agent strategies for pommerman, 2017.