Kiel University
Christian-Albrechts-Universität zu Kiel

Faculty of Engineering

Autonomous Learning

Installing your programming environment

Simon Reichhuber

May 27, 2020

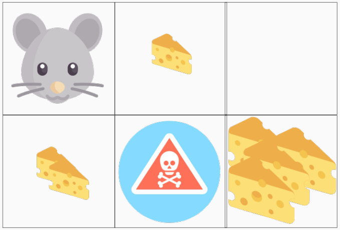
University of Kiel, Summer Term 2020

Table of contents

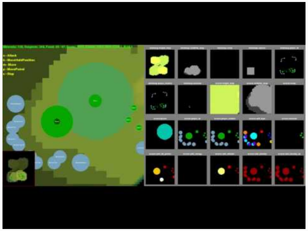
Submissions

Topics

- Reinforcement Learning



- StarCraft II API (Blizzard & Google)



Submissions

- Upload your solution to the link of your Group ($\leftarrow$ posted in the forum)
- Submit a zip-file containing the code and an explanatory video: *<Group name>_Assignement<Assignement no.>.zip*

Linux

Recommendation: use Linux

- Besides, some specific problems visualisations cannot be switched off on Windows/Mac
- Usually packages are easier to install on Linux

SUBMISSIONS II

Remarks to coding style

- It is not forbidden to use object-oriented programming.
- Use folders to increase readability (e.g. assignement1, task1, etc.)
- Comment important code
- Use telling variable names, like *update_state_values()* (not *do_calcuation()*, or *get_result()*, etc.)

Important: Use debugging tools of your IDE

Install PySC2

- DeepMind's Python Framework
- Reinforcement Learning Environment
    - receive Observation
    - take Action
- github: www.github.com/deepmind/pysc2

DEEPMIND

- Knowledge limited to the scope of human players
- No chance against simple KIs
- For hard coding KIs too complex

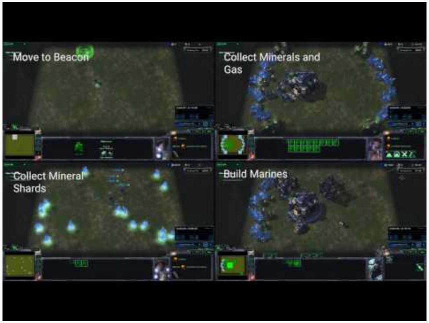
Stand 2019:
AlphaStar wins 10:1 against professional StarCraft 2 players

Minigames

- Isolated view of certain elements of the game
- Focused scenarios on small maps
- Test/Learn actions and/or game mechanics

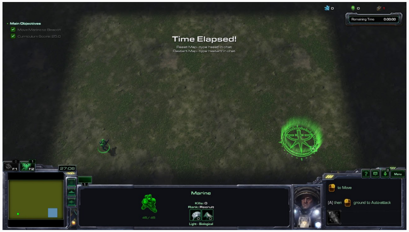
Minigames

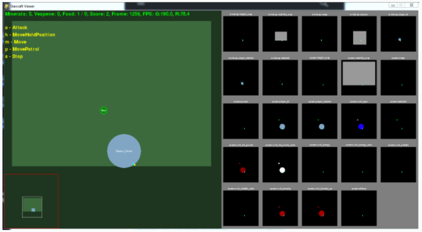
- MoveToBeacon
- CollectMineralShards
- FindAndDefeatZerglings
- DefeatRoaches
- DefeatZerglingsAndBanelings
- CollectMineralsAndGas
- BuildMarines



Minigames for assignements

Apply different Reinforcement Learning algorithms on the minimap MoveToBeacon





Minerals: 0, Vespene: 0, Food: 1 / 0; Score: 2, Frame: 1256, FPS: G:190.0, R:78.4

a - Attack
h - MoveHoldPosition
m - Move
p - MovePatrol
s - Stop

Units

Marine:

```
def get_marine(obs):
    marine = next(unit for unit in obs.observation.
        feature_units
                  if unit.alliance == features.
                      PlayerRelative.SELF)
    return marine
```

Beacon:

```
def get_beacon(obs):
    beacon = next(unit for unit in obs.observation.
        feature_units
                  if unit.alliance == features.
                      PlayerRelative.NEUTRAL)
    return beacon
```

Units

Position:

```python
def getCoord(unit):
    x = unit.x
    y = unit.y
    return x, y
```

Actions

Keine Aktion:

```
return actions.FUNCTIONS.no_op()
```

Select Unit:

```
return actions.FUNCTIONS.select_army("select")
```

Move Selected Units:

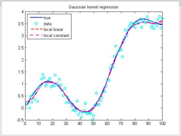
```
return actions.FUNCTIONS.Move_screen("now", (x, y))
```

Reinforcement Learning

Machine Learning

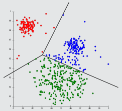
Supervised Learning

→ problem-driven (Regression / Classifiction)



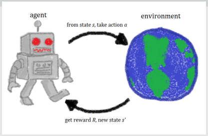
Unsupervised Learning

→ data-driven (Clustering)



Reinforcement Learning

→ react to environment


agent
environment
from state s, take action a
get reward R, new state s'

Definition

Kiel University
Christian-Albrechts-Universität zu Kiel

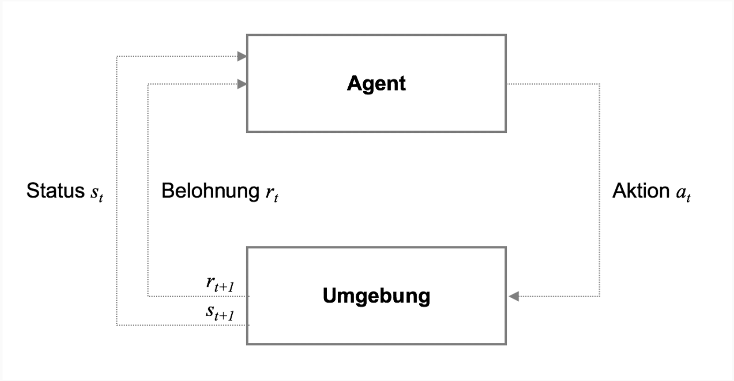
Faculty of Engineering

Reinforcement Learning

”Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics—trial-and-error search and delayed reward—are the two most important distinguishing features of reinforcement learning.”

Richard S. Sutton and Andrew G. Barto

COMPONENTS



1. Agent
2. Environment
3. State
4. Action
5. Reward

Exploration-Exploitation Dilemma

Goal

Find relation of action and expected reward and maximise rewards by taking several actions.

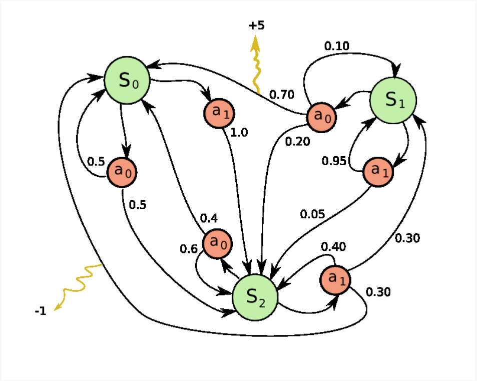
Problem: Follow known strategies (exploitation) or explore new unkown strategies which might result in a bigger reward?

$\Rightarrow$ Exploration-Exploitation Dilemma

Markov Decision Process

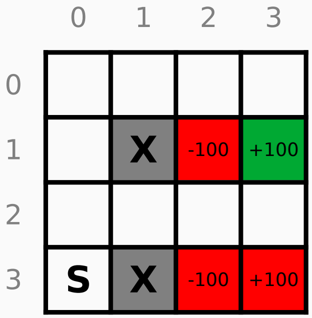
Markov Property

The future is independent of the past if you know the presence.



Beispiel

- 4 actions
- Each action reults in a reward of -2
- Coloured states are final states with a specific reward (reward 100/-100)



- With probability 0.8 the agent moves to the intended state; with probability 0.1 it takes one of the other two orthogonal actions.

Markov Decision Process

Notation

- $S = s_1, \dots, s_n$ is the set of all states
- $A = a_1, \dots, a_n$ is the set of all actions
- $T : S \times A \times S \to [0, 1]$ is the transition function. $T(s, a, s')$ represents the probability that an agent in state s moves to s' by taking action a.
- $R : S \times A \times S \to \mathbb{R}$ is the reward function. $R(s, a, s')$ assigns the transition from state s to state s' by taking action a a reward.

Markov Decision Process

Notation

- $\pi(s)$ is a policy that maps the state s to an optimal action.
- $\gamma \in [0, 1]$ is the discount factor. All possibly future rewards are multiplied after each action with γ. By this, the agent shall prefer short paths. With $\gamma = 1$ the length of the path is arbitrary if the reward is equal along the paths. With $\gamma = 0$ only the reward of the next step is taken into account.

Markov Decision Process

V-Value

$V^*(s)$ $=$ Estimated total reward for an agent die der Agent starting at a specific state s and taking the optimal actions afterwards.

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma * V^*(s')]$$

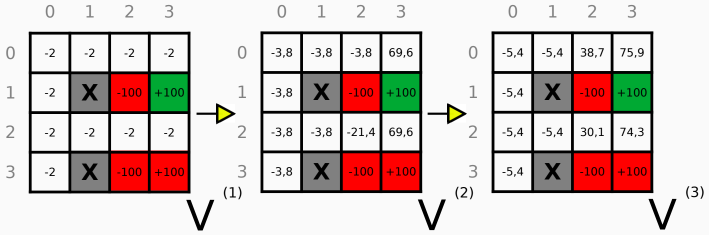
Meaning

What is the value of a state s?

Value Iteration

- Temporally limited MDP
- $V^{(k)}$ = Total reward, only *k* steps left

$$V^{(k)}(s) = \max_{a \in A} \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma * V^{k-1}(s')]$$



$V^{(1)}$

	0	1	2	3
0	-2	-2	-2	-2
1	-2	X	-100	+100
2	-2	-2	-2	-2
3	-2	X	-100	+100

$V^{(2)}$

	0	1	2	3
0	-3,8	-3,8	-3,8	69,6
1	-3,8	X	-100	+100
2	-3,8	-3,8	-21,4	69,6
3	-3,8	X	-100	+100

$V^{(3)}$

	0	1	2	3
0	-5,4	-5,4	38,7	75,9
1	-5,4	X	-100	+100
2	-5,4	-5,4	30,1	74,3
3	-5,4	X	-100	+100

Value Iteration - State: $(0,3)$, k=2

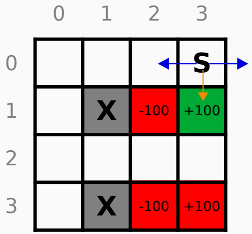
Aktion: South

$$0.8 \times (-2 + 0.9 \times 100) = 70.4$$

$$0.1 \times (-2 + 0.9 \times -2) = -0.38$$

$$0.1 \times (-2 + 0.9 \times -2) = -0.38$$

$$\sum_{s' \in S} [70.4; -3.04; -3.04] = 69.6$$



Aktion: West $= 5,38$; Aktion: North $= -3,8$; Aktion: East $= 5,38$

$$\max_{a \in A} [69,6; 5,38; -3,8; 5,38] = 69,6$$

$$V^{(2)}(0,3) = 69,6$$

Value Iteration

- max Reward of the MDP nach bounded above
- V^k in the limit $k \to \infty$ converges
- For sufficiently large k it holds $V^{(k)} = V^* \pm \epsilon$

Policy Extraction

- From the v-values V^* one can derive a optimal policy π^*
- For each state take the action that maximises the reward

$$\pi^*(s) = \operatorname*{argmax}_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma * V^*(s')]$$

	0	1	2	3
0	49,5	59,1	70,1	82,6
1	41,0	X	-100	+100
2	33,6	28,2	34,9	76,3
3	27,0	X	-100	+100

	0	1	2	3
0	⇒	⇒	⇒	⇓
1	⇑	X	⊗	⊗
2	⇑	⇒	⇒	⇑
3	⇑	X	⊗	⊗

Q-Learning

IMPERFECT KNOWLEDGE

- Value iteration and policy extraction requires knowledge about $T(s, a, s')$ and $R(s, a, s')$
- Agent has to follow its own strategy from state to state
- Externally, the agent receives a reward from the environment

Temporal Difference Learning

- Initially, randomly chosen V values $\rightarrow$ iterative improvement
- After every step, the agent analyses the reward and the possible future rewards and adapts its V value estimation approximatively.

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$$

Q-Values

Q-Values

$Q(s, a)$ is the quality of action a in state s

$$Q^*(s,a) = \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

explicit actualisation of the policy can be unnecessary if it is immediately learned implicitely (on-policy):

$$V(s) = \max_{a \in A} Q(s,a)$$

$$\pi(s) = \operatorname*{argmax}_{a \in A} Q(s,a)$$

Q-Learning

Q-Learning

= Temporal Difference Learning auf Q-Values

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)]$$

Q-Tabelle

- Q function as table
- For each Q value a single entry
- Initialise the values (randomly, with max reward, or zero)

States \ *Actions*	NORTH	EAST	SOUTH	WEST
(3,0)	31	26,4	25,7	26,4
(2,0)	37,6	28,1	26,5	32,3
(2,1)	28,1	32,2	28,1	31,2
(2,2)	-69,9	39,6	-69,9	1,5
⋮	⋮	⋮	⋮	⋮

ϵ-Greedy Exploration

- Exploration strategy
- Hyperparameter $0 \leq \epsilon \leq 1$
- On every step, the agent takes a random action with probability ϵ
- ϵ decreases over training time

Pseudocode

```
Initialize $Q(s, a)$ arbitrarily;
repeat
    Initialize $s$;
    repeat
        Chose $a$ from $s$ using policy derived from $Q$
          ($\epsilon$-greedy);
        Take action $a$, observe $r$, $s'$;
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a \in A} Q(s', a) - Q(s, a)]$;
        $s \leftarrow s'$
    until $s$ is terminal;
until $Q$ is converged;
```

Algorithm 1: Q-Learning

Bibliography

Bilder

1. https://medium.freecodecamp.org/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe
2. https://github.com/deepmind/pysc2
3. https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/19564/versions/1/screenshot
4. https://2s7gjr373w3x22jf92z99mgm5w-wpengine.netdna-ssl.com/wp-content/uploads/2018/02/reinforcement-learning.png
5. https://towardsdatascience.com/reinforcement-learning-demystified-markov-decision-processes-part-1-bf00dda41690
6. https://hciweb.iwr.uni-heidelberg.de/system/files/private/downloads/541645681/damma learning-report.pdf