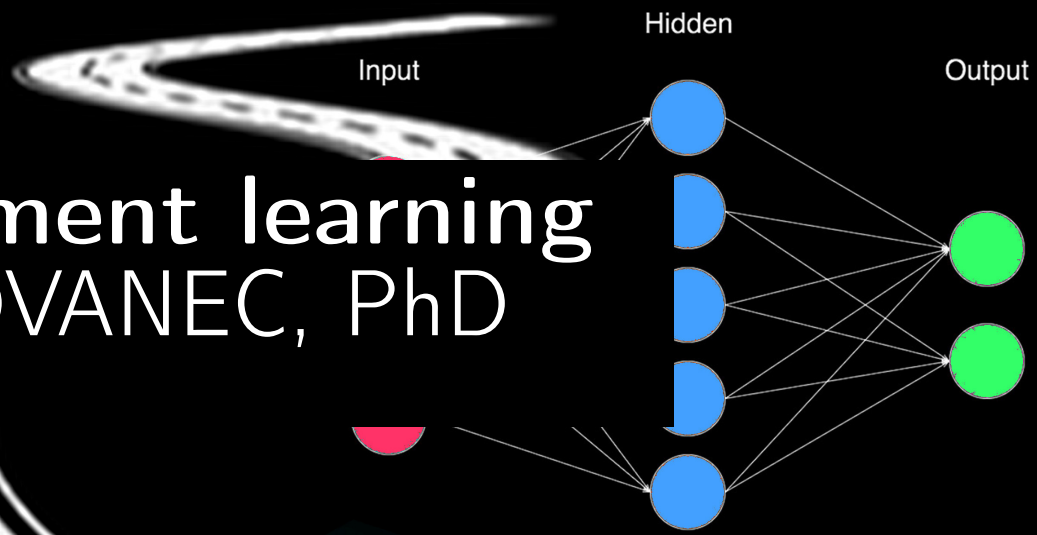
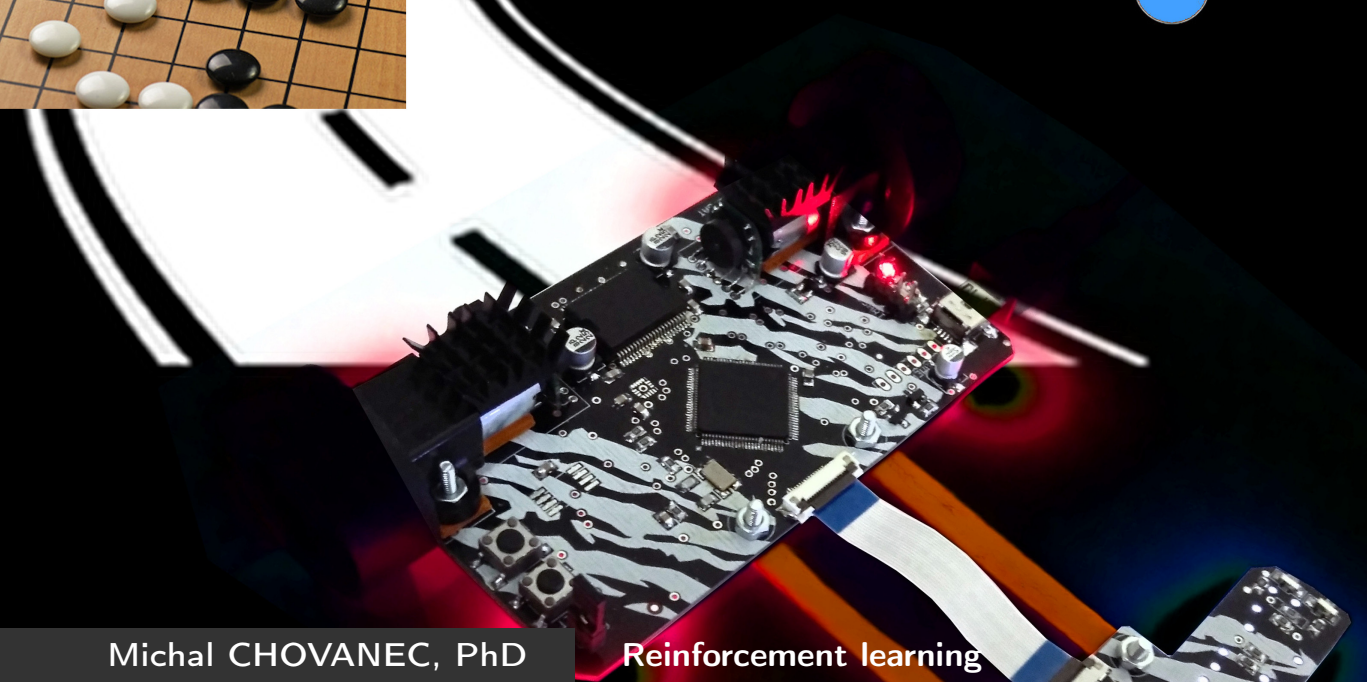
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \lambda \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$$

(The New Action Value = The Old Value) + The Learning Rate × (The New Information − the Old Information)



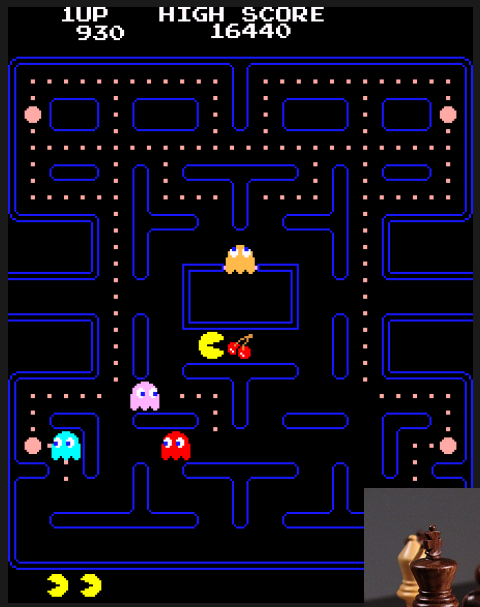
Reinforcement learning

Michal CHOVANEC, PhD



Reinforcement learning

- learn from punishment and rewards
- learn to play a game with unknow rules



Reinforcement learning

1. supervised learning based on human player

$$action = F(observation)$$

$$loss = \left(required\_action - F(observation)\right)^2$$

2. RL method

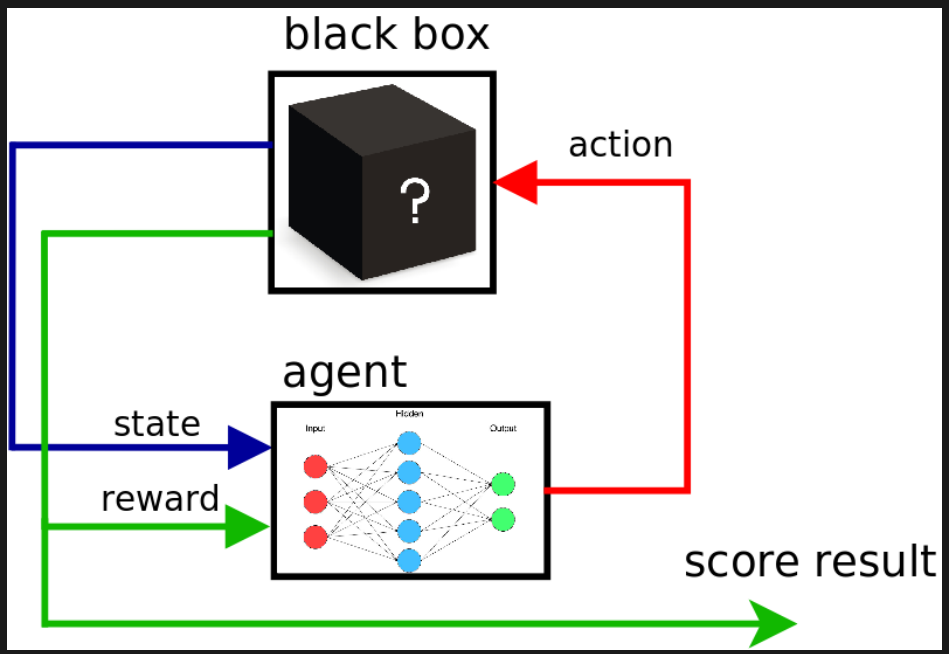
$$action = F(observation)$$

$$loss = \left(required\_action - F(observation)\right)^2$$

required_action = ?

Reinforcement learning

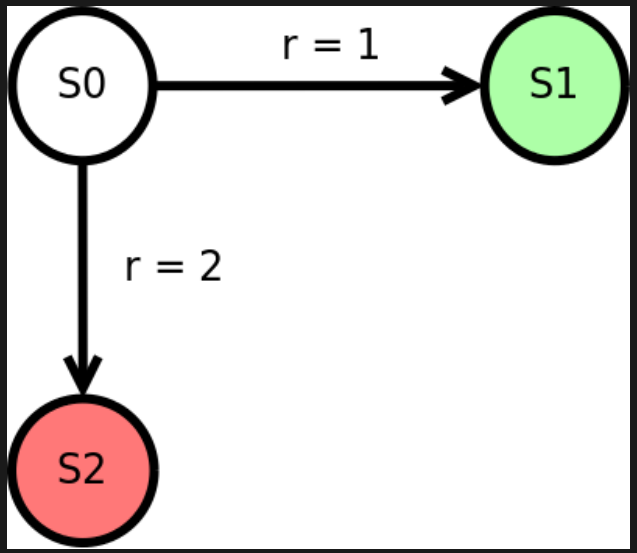
- obtain **state**
- choose **action**
- **execute** action
- obtain **reward**
- learn from **experiences**



Making decisions

two possible strategies

- strategy 1 : S0->S1, score = 1.0
- strategy 2 : S0->S2, score = 2.0



$$Q(s, a) = R(s, a)$$

where

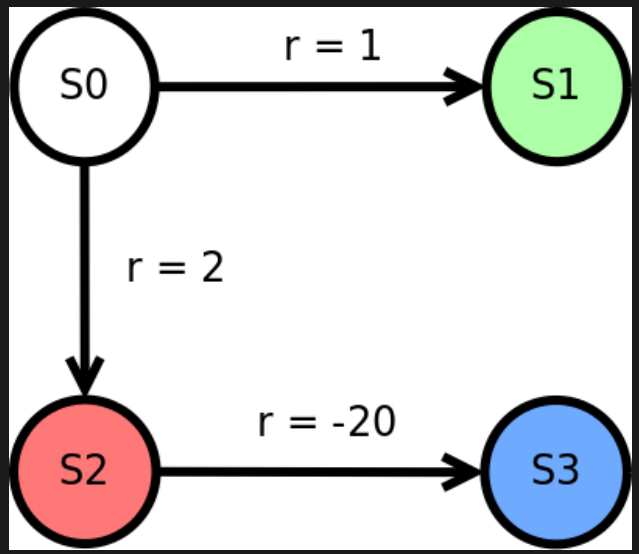
s is state

a is action

Making decisions

two possible strategies, greedy = trap

- strategy 1 : S0->S1, score = 1.0
- strategy 2 : S0->S2->S3, score = 2.0 + (-20.0) = -18

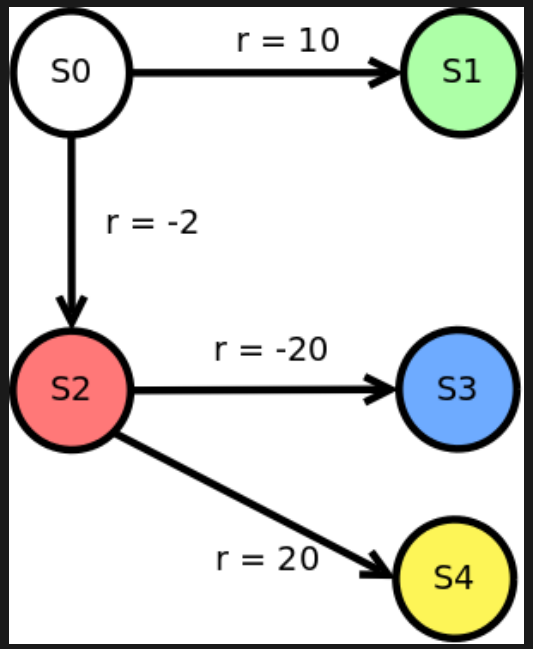


$$Q(s, a) = R(s, a) + FutureReward$$

Making decisions

three possible strategies

- strategy 1 : S0->S1, score = 10.0
- strategy 2 : S0->S2->S3, score = -2.0 + (-20.0) = -22
- strategy 3 : S0->S2->S4, score = -2.0 + (20.0) = 18



$$Q(s, a) = R(s, a) + max(FutureRewards)$$

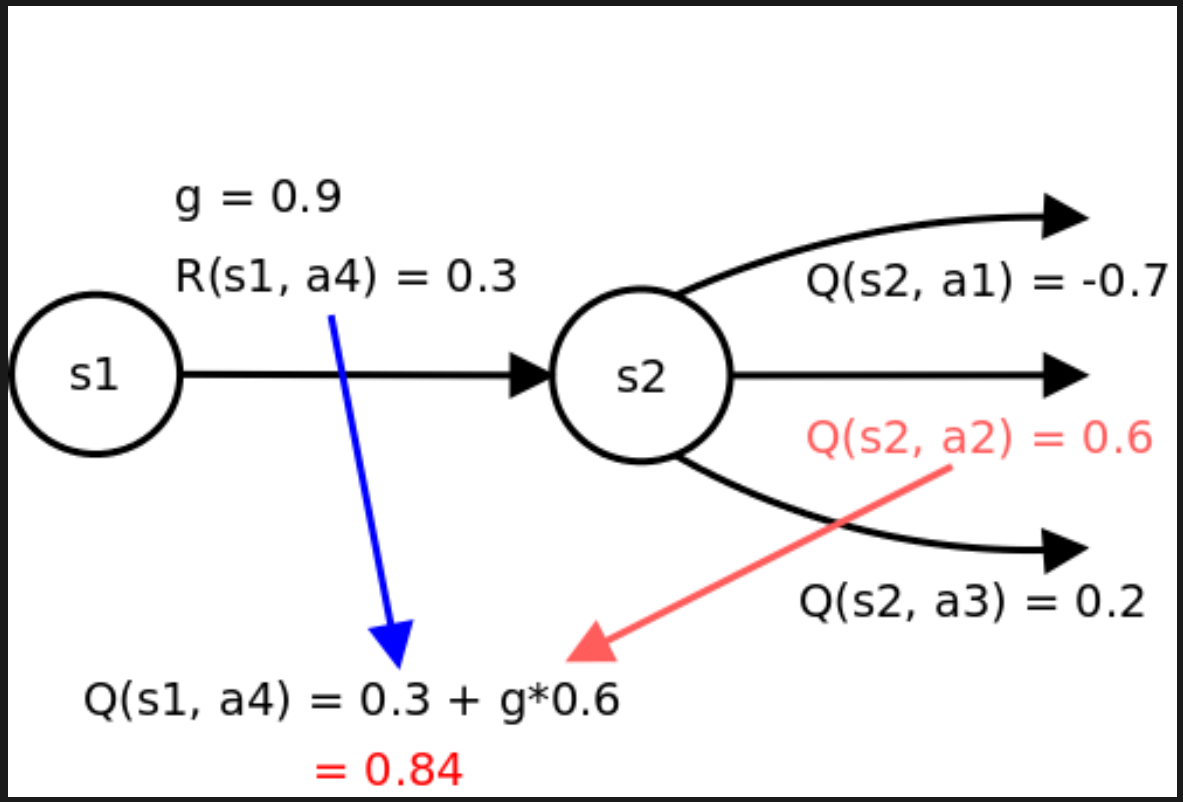
Q learning

$$Q(s,a) = R(s,a) + \gamma \max_{a'} Q(s',a')$$

where
s is state
a is action
s' is next state
a' is best action in next state
$R(s,a)$ is reward
$\gamma \in \langle 0,1 \rangle$ is discount factor

Q learning

$$Q(s,a) = R(s,a) + \gamma \max_{a'} Q(s',a')$$



Q learning - stochastic

$$Q(s,a) = R(s,a) + \gamma \max_{a'} Q(s',a')$$

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\Big(R(s,a) + \gamma \max_{a'} Q(s',a')\Big)$$

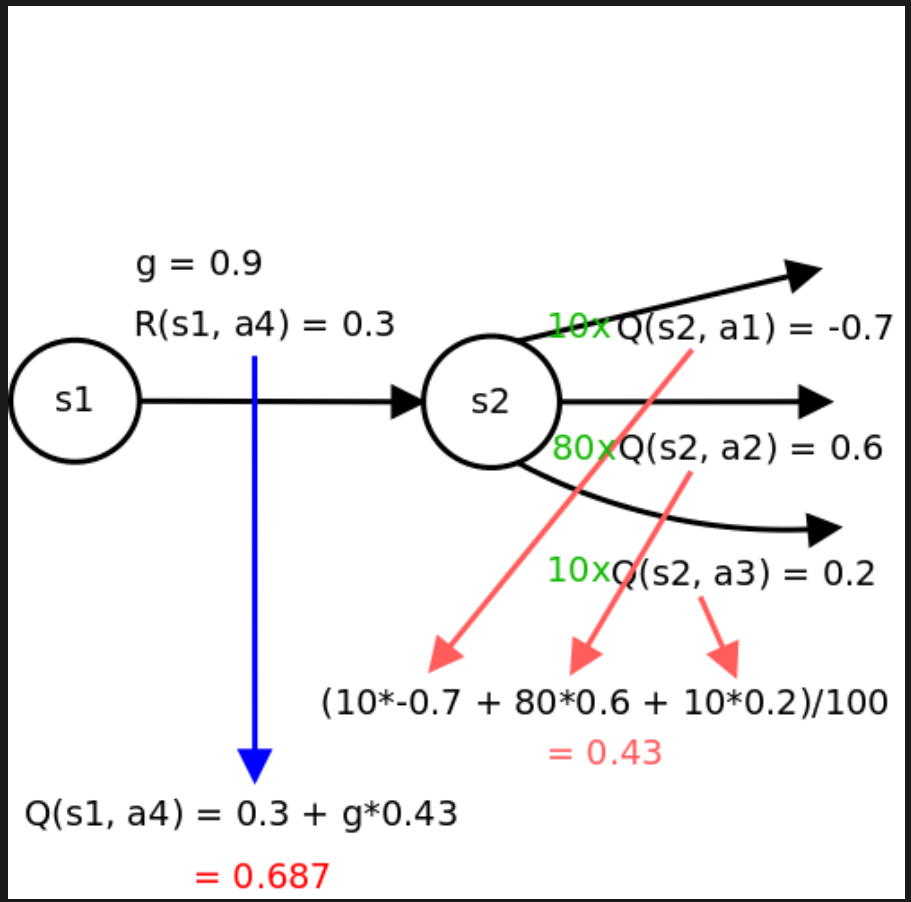
Q learning

- **obtain reward**
  reward = env.get_reward();
- **update state**
  state_old = state;
  state = env.get_observation();
- **select action**
  action_old = action;
  action = select_action(Q(state));
- **process learning**
  Q(state_old, action_old)+= α(reward + γ $\max_{a'}$Q(state, a') - Q(state_old, action_old));
- **execute action**
  env.action(action);

SARSA learning

$$\Delta Q(s,a) = \alpha\Big(R(s,a) + \gamma Q(s',a') - Q(s,a)\Big)$$

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\Big(R(s,a) + \gamma Q(s',a')\Big)$$



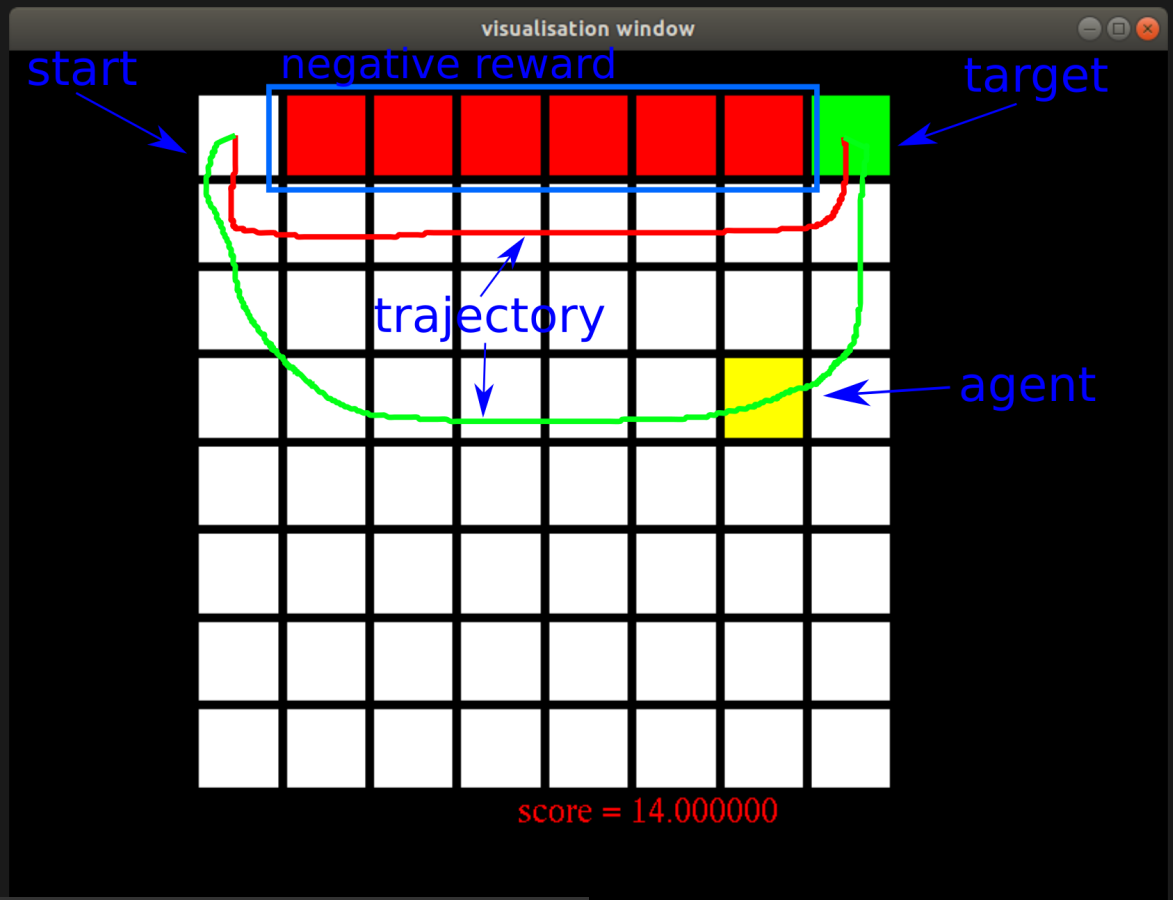
SARSA learning

- **obtain reward**
  reward = env.get_reward();
- **update state**
  state_old = state;
  state = env.get_observation();
- **select action**
  action_old = action;
  action = select_action(Q(state));
- **process learning**
  Q(state_old, action_old)+= α(reward + γQ(state, action) - Q(state_old, action_old));
- **execute action**
  env.action(action);

Q-learning vs SARSA - cliff example

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\Big(R(s,a) + \gamma \max_{a'} Q(s',a')\Big)$$

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\Big(R(s,a) + \gamma Q(s',a')\Big)$$



Deep reinforcement learning

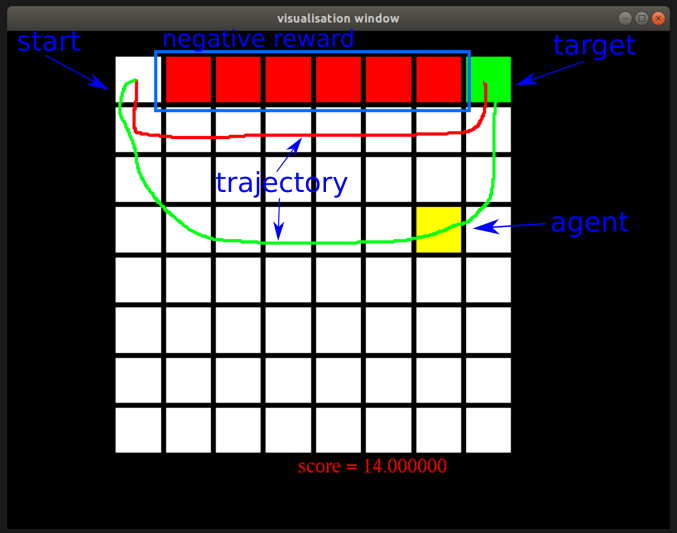
Possible states in some games

- **tic-tac-toe** 26830 states
- **2048** 44096709674720289 states
- **atoms in observable universe** 10^{82}
- **chess** 10^{120} states
- **GO** 10^{170} states

storing Q values

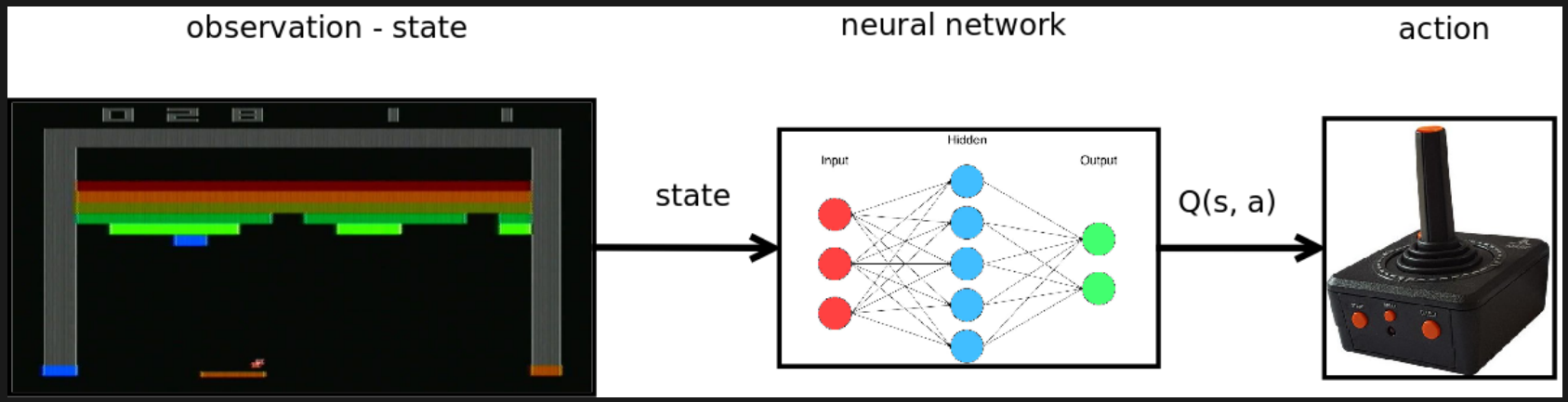
- table
- linear combination of features, $Q(s,a) = \sum_{i=1}^{N} w_i F_i(s,a)$
- neural network

Q table example

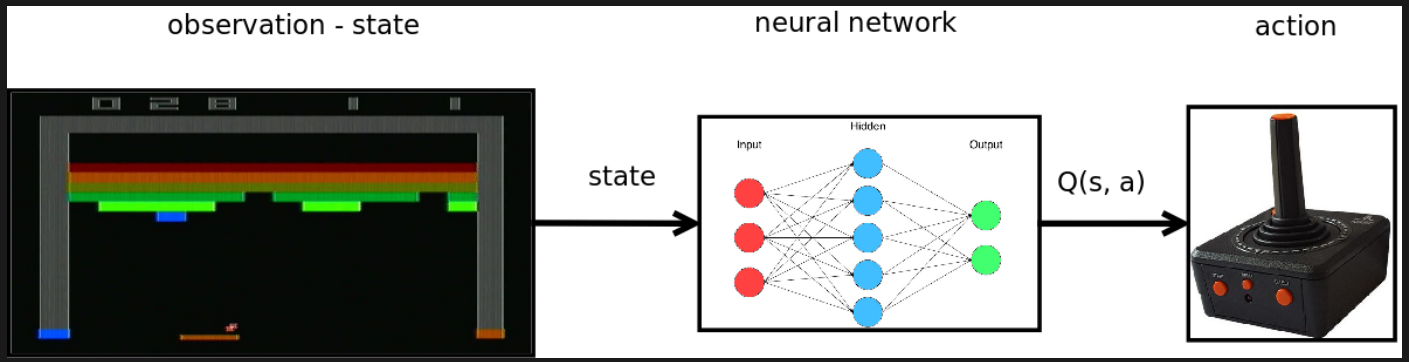


State	A0 UP	A1 LEFT	A2 DOWN	A3 RIGHT
0			0.43	-1
1, 2, 3, 4, 5, 6	T	T	T	T
7	T	T	T	T
8				0.49
9	-1			0.53
10	-1			0.59
11	-1			0.65
12	-1			0.72
13	-1			0.81
14	-1			0.9
15	1			

Neural network for $Q(s, a)$ approximation - Q networks



Neural network for $Q(s, a)$ approximation - Q networks



$$SARSA : Q(s,a) = (1-\alpha)Q(s,a) + \alpha\Big(R(s,a) + \gamma Q(s',a')\Big)$$

$$loss = \Big(required\_value - Q(state, action)\Big)^2$$

$$loss = \Big(Q(s,a) - Q'(s,a)\Big)^2$$

$$loss = \Big(R(s,a) + \gamma Q(s',a') - Q'(s,a)\Big)^2$$

Usefull links

- ImageNet Classification with Deep Convolutional Neural Networks https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
- Alex Krizhevsky web, https://www.cs.toronto.edu/~kriz/
- Deep Belief Nets in C++ and CUDA C: Volume III https://www.amazon.com/Deep-Belief-Nets-CUDA-Convolutional/dp/1530895189
- Deep Learning (Adaptive Computation and Machine Learning https://www.amazon.com/Deep-Learning-Adaptive-Computation-Machine/dp/0262035618
- Densely Connected Convolutional Networks https://arxiv.org/pdf/1608.06993.pdf
- MNIST dataset http://yann.lecun.com/exdb/mnist/
- Digital signal processing for STM32 microcontrollers using CMSIS https://www.st.com/resource/en/application_note/dm00273990.pdf
- CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs https://arxiv.org/pdf/1801.06601.pdf

Q&A

michal chovanec (michal.nand@gmail.com)

www.youtube.com/channel/UCzVvP2ou8v3afNiVrPAHQGg

github https://github.com/michalnand