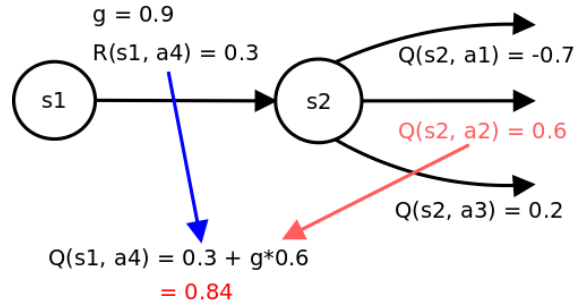


Deep Q networks

Michal Chovanec, michal.nand@gmail.com

1 Q-learning



Obr. 1: Q learning

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

where

s is state

a is action

s' is next state

a' is best action in next state

R is reward function

$\gamma \in \langle 0, 1 \rangle$ is discount factor

2 Deep Q network - DQN

Approximate $Q(s, a)$ using neural network as $\hat{Q}(s, a; w)$, where w are learnable network parameters
resulted Q value using 1

$$\hat{Q}(s, a; w) = R + \gamma \max_{\alpha'} \hat{Q}(s', \alpha'; w) \quad (2)$$

error to minimize

$$E = R + \underbrace{\gamma \max_{\alpha'} \hat{Q}(s', \alpha'; w)}_{\text{target value}} - \underbrace{\hat{Q}(s, a; w)}_{\text{predicted value}} \quad (3)$$

weights gradient

$$\Delta w = \eta E \nabla_w \hat{Q}(s, \alpha, w) \quad (4)$$

There is changing target value, $\hat{Q}(s, a; w)$ depends on w which is also changing during training, and leads to unstable learning.

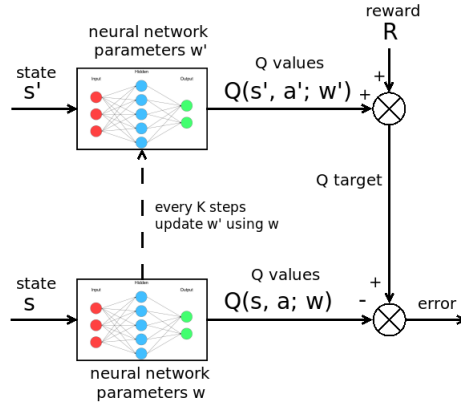
Solution - fix w , using temporary network with w' weights. This leads to deep Q learning equation

$$\hat{Q}(s, a; w) = R + \gamma \max_{\alpha'} \hat{Q}(s', \alpha'; w') \quad (5)$$

and every K steps update $w' \leftarrow w$, usually after training epoch. For agent decision making, the network w' is used. Finally, network error can be computed as

$$E = R + \gamma \max_{\alpha'} \hat{Q}(s', \alpha'; w') - \hat{Q}(s, a; w) \quad (6)$$

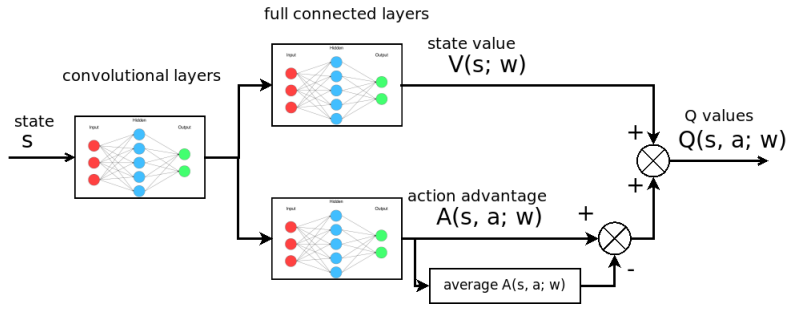
Principle of DQN can be demonstrated on image 2 (Q-learning γ and \max operator are removed for simplicity).



Obr. 2: basic DQN principle

3 Dueling deep Q network - DDQN

$$\hat{Q}(s, a, w) = \underbrace{\hat{V}(s, w)}_{\text{value for being in state } s} + \underbrace{\hat{A}(s, a, w)}_{\text{advantage of taking action } a \text{ at state } s} \quad (7)$$



Obr. 3: dueling DQN principle

to avoid identifiability we subtract average value of A from all actions

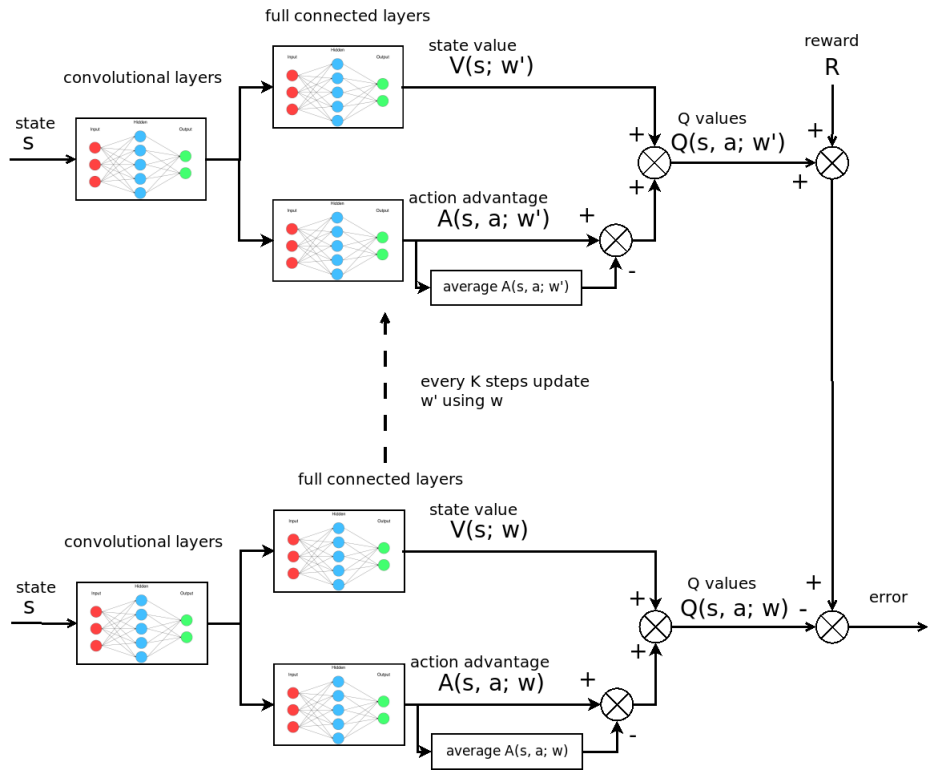
$$\hat{Q}(s, a, w) = \hat{V}(s, w) + \hat{A}(s, a, w) - \frac{1}{N_{\alpha'}} \sum_{\alpha'} \hat{A}(s, \alpha', w) \quad (8)$$

using equation 5 we obtain dueling deep Q network equation

$$\hat{Q}(s, a; w) = R + \gamma \left(\hat{V}(s', w') + \max_{\alpha'} \hat{A}(s', \alpha', w') - \frac{1}{N_{\alpha'}} \sum_{\alpha'} \hat{A}(s', \alpha', w') \right) \quad (9)$$

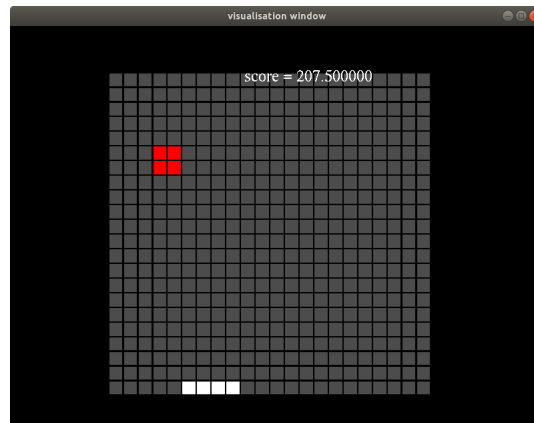
and finally the weights learning rule

$$\Delta w = \eta \left(R + \gamma \left(\hat{V}(s', w') + \max_{\alpha'} \hat{A}(s', \alpha', w') - \frac{1}{N_{\alpha'}} \sum_{\alpha'} \hat{A}(s', \alpha', w') \right) - \hat{Q}(s, a; w) \right) \nabla_w \hat{Q}(s, a; w) \quad (10)$$



Obr. 4: dueling DQN principle, full

4 Experiments



Obr. 5: example of testing game

Network hyperparameters

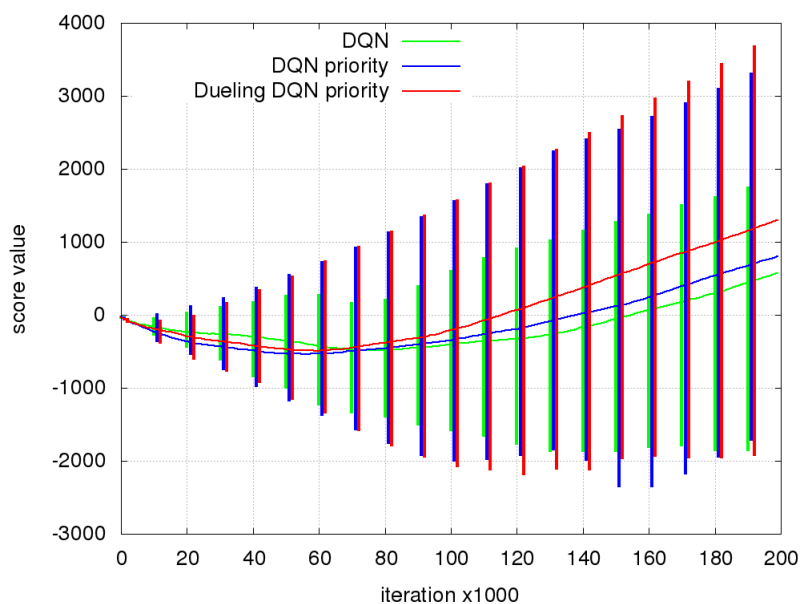
- init weights : XAVIER
- learning rate : 0.0005
- L1 regularization : 0.0000001
- L2 regularization : 0.0000001
- dropout : 0.2

RL hyperparameters

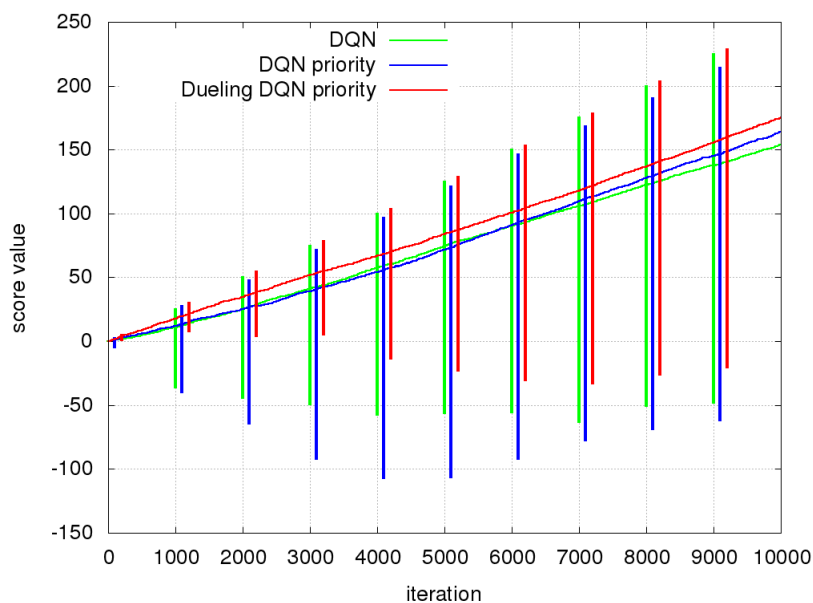
- experience buffer size : 1024
- gamma : 0.95
- epsilon training : 0.1
- epsilon testing : 0.05

layer type	input dimensions	kernel dimensions
dense convolution	22x22x1	3x3x8
dense convolution	22x22x9	3x3x8
dense convolution	22x22x17	3x3x8
dense convolution	22x22x25	3x3x8
convolution	22x22x33	3x3x32
full connected	22x22x32	actions_count

Tabuľka 1: testing network architecture



Obr. 6: training score progress



Obr. 7: testing score progress