dortmund university

Deep Q-Network (DQN)

Was kann DQN?

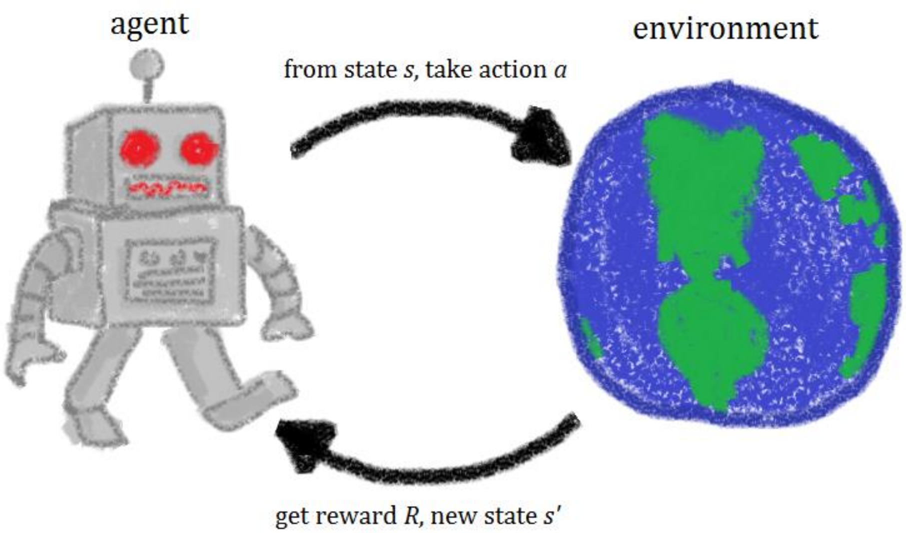

Detection image

Breakout

Was ist DQN?

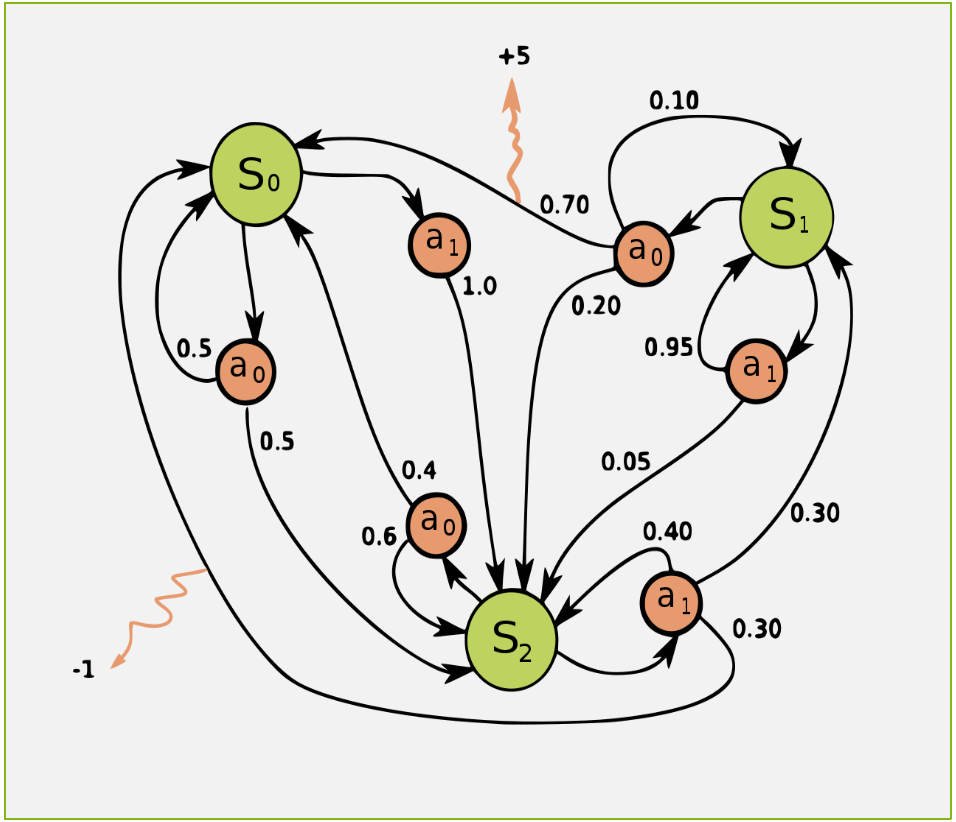
Agent-Environment Loop

- Agent perceives the state of the environment
- Agent takes an action
- Environment transitions to a new state
- Environment provides a reward to the agent



Markov Decission Process..

- $States(S)$
- $Actions(A)$
- $transition\ Function$:
  $P(s'|s,a)$
- $Reward\ Function$:
  $R(s,a,s')$



.. and its return

The return G_t is the total accumulated reward from timestep $\boldsymbol{t}$ onwards.

Defined as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Policy

A Policy π is a function that defines the behavior of an agent by specifying the probability of taking each action a in state s

$$\pi(a|s) = P(a_t = a|s_t = s)$$

The goal of a policy is to maximize the expected return starting from each state

Action value function

Action value function $q_\pi(s, a)$ under policy π is the expected return after taking action a in state s following π

$$q_\pi(s, a) = E_\pi[G_t | s_t = s, a_t = a]$$

Bellman‘s expectation equation..

.. for action value functions:

$$q_\pi(s,a) = E[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1)} | s_t = s, a_t = a]$$

Bellman's optimality equation..

.. for action value functions:

$$q_*(s, a) = E[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a')|\, s_t = s, a_t = a]$$

Where $\boldsymbol{q_*}$ is the optimal action value function.

Q-Learning

- Model-free reinforcement learning algorithm
- Aims to find the optimal action value function $\boldsymbol{q_*}$
- Update rule is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q\big(s_{t+1,} a_{t+1}\big) - Q(s_t, a_t)]$$

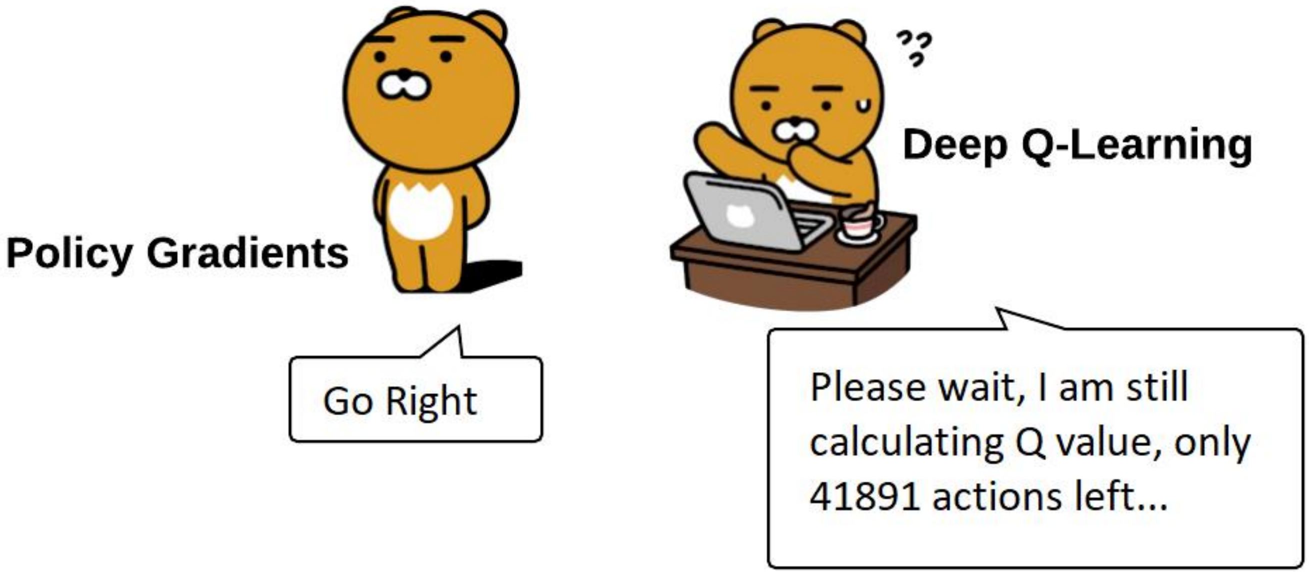
$$Q(s_t, a_t) \leftarrow (1 - \alpha) * Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q\big(s_{t+1,} a_{t+1}\big)]$$

- Related to bellman optimality equation
- Iteratively updates action value function to converge to $\boldsymbol{q_*}$

Epsilon-Greedy Policy

- Balances exploration and exploitation by choosing a random action
- Random action with ϵ and best-known action with 1 - ϵ
- Helps ensure sufficient exploration of the state-action space
- Crucial for convergence of Q-Learning to optimal action value function

Einführung in das Deep Q Learning



update rule & loss function

- Update rule

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q\big(s_{t+1,} a_{t+1}\big) - Q(s_t, a_t)]$$

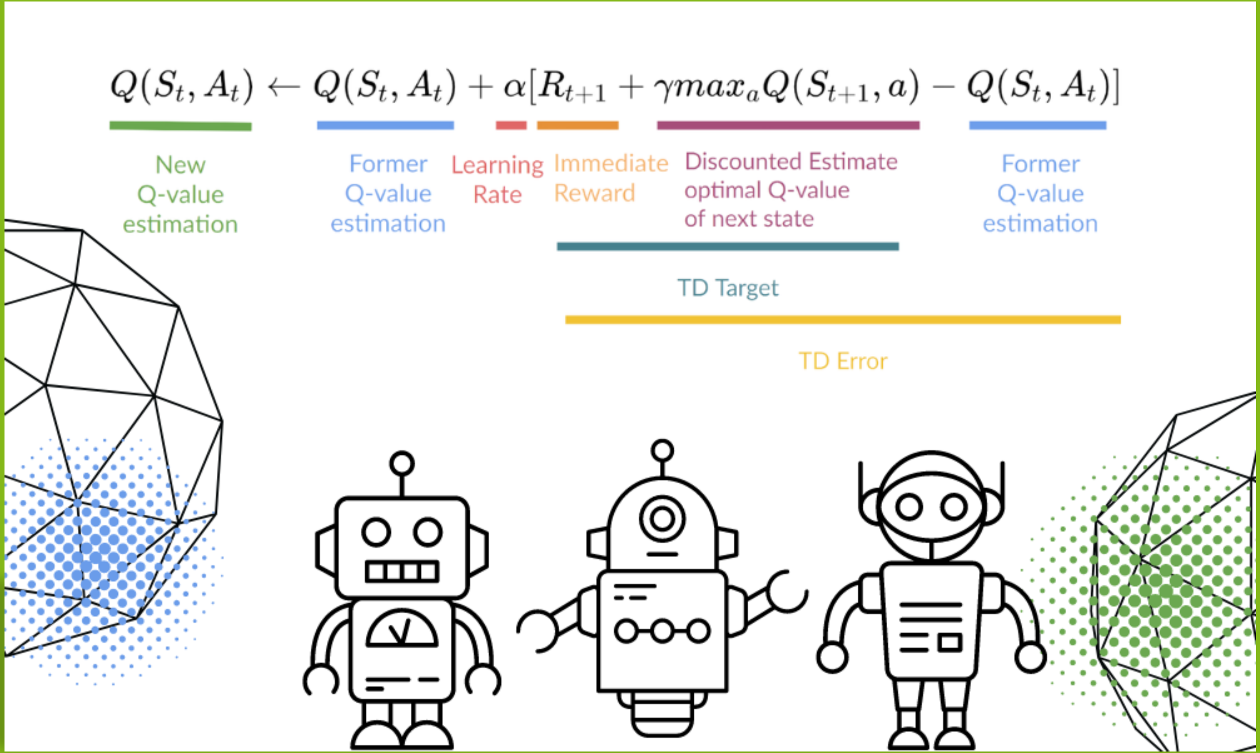
- Loss function

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim p(.)}[(y_i - Q(s_t, a_t\,; \theta_i))^2]$$

$$y_i = [r + \gamma\, max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_{i-1})]$$

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim p(.)}[((r + \gamma\, max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_{i-1})) - Q(s_t, a_t\,; \theta_i))^2]$$

Update rule a closer look!



Gadient

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim p(.)}[\,\big(y_i - Q(s_t, a_t\,; \theta_i)\big)\nabla_{\theta_i} Q(s_t, a_t\,; \theta_i)]$$

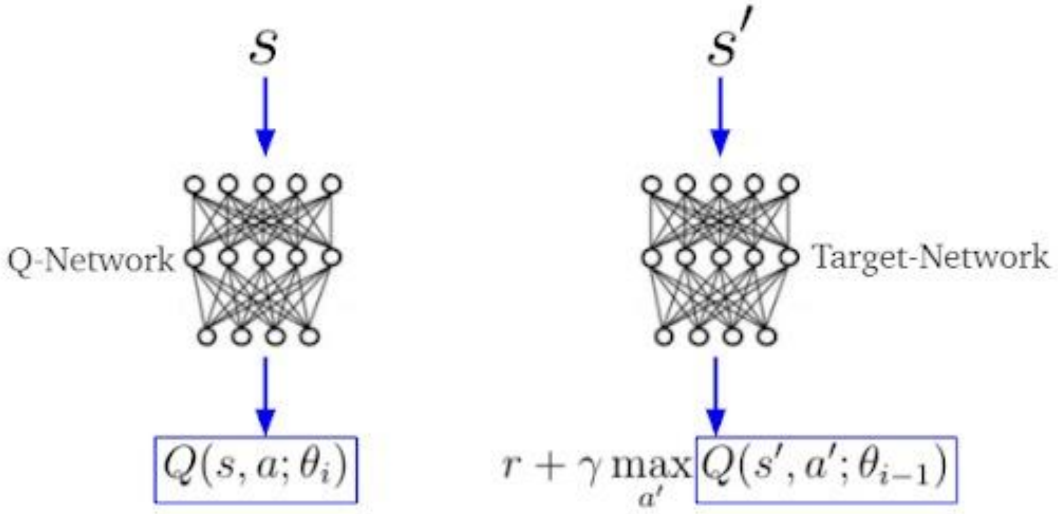
$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim p(.)}\Big[\Big(r + \gamma\, max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_{i-1}) - Q(s_t, a_t\,; \theta_i)\Big)\,\nabla_{\theta_i} Q(s_t, a_t\,; \theta_i)\Big]$$

- to minimize the loss function
- computing the gardient for updates: $\theta_{i+1} \leftarrow \theta_i$- $\alpha\, \nabla_{\theta_i} L_i(\theta_i)$

Experience Replay

- Memory of experiences
- Pick of minibatches for update of the Q-function
- Adresses three issues:
  - Data Efficiency
  - Low Correlation
  - Catastrohic forgetting

Target Q-Network



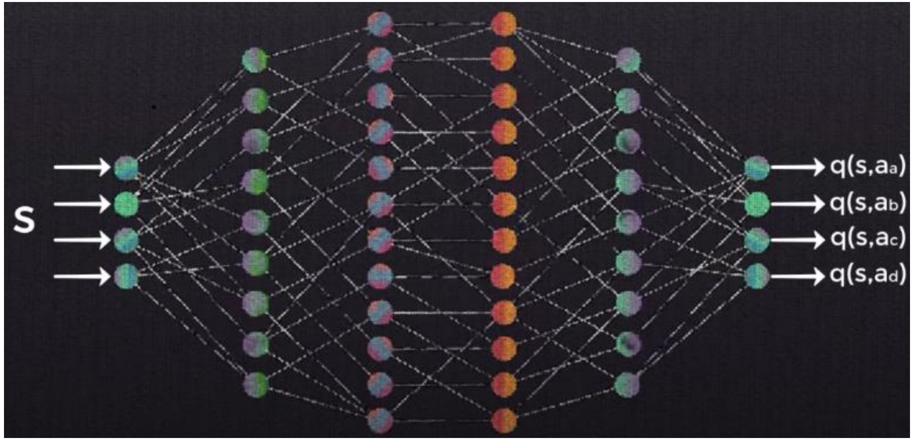
DQN Pseudocode

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Neuronal Network

- Q-function » Q-network » neuronnal network
- Weights of q-learning are weights for neuronal network
- Layers: input (states), hidden (convulational layers), output (q-values for each action)



https://youtu.be/xVkPh9E9GfE

Preprocessing of game frames

1. Grayscale
2. Downsampling to 110x84
3. Crop to 84x84
4. Stack 4 frames to one

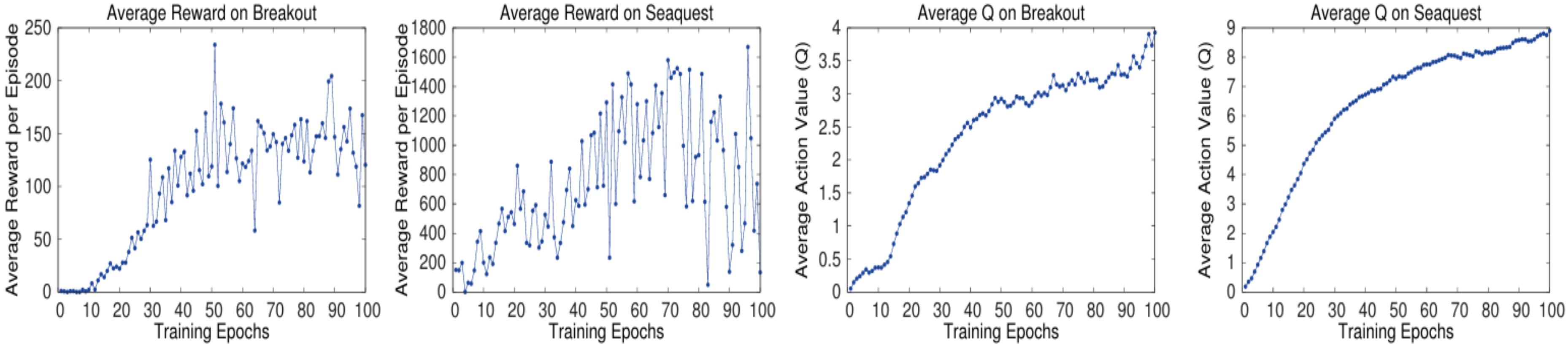
Reward Clipping

- limit the reward to interval [-1,1]
- High rewards are counterproductive for strabelized learning
- Clipping stabelizes learning

Performance Metrics

- Within a specific time periode
  - Average and maximum points

Results from the Paper



Results from the Paper

How to implement

Classes:

Agent Class (DQNAgent):

- Responsible for controlling the DQN agent.
- Methods:
  - select_action: Selects actions based on the current state.
  - train: Trains the agent using experiences stored in the replay memory.
  - update_target_network: Updates the target network periodically.

How to implement

Classes:

Neural Network Class (DQN):

- Defines the neural network architecture for approximating Q-values.
- Methods:
    - __init__: Initializes the network with state_size and action_size.
    - forward: Performs forward pass computation.
    - compute_loss: Computes the loss between predicted and target Q-values.
    - backward: Performs gradient descent.

How to implement

Classes:

Replay Memory Class:

- Manages the replay memory buffer.
- Methods:
    - add_experience: Adds experiences to the replay memory.
    - sample_batch: Samples batches of experiences for training.

Links / Sources

Picture Roboter & World :
https://commons.wikimedia.org/wiki/File:Rl_agent.png
Markov decision process:
https://upload.wikimedia.org/wikipedia/commons/thumb/a/ad/Markov_Decision_Process.svg/1280px-Markov_Decision_Process.svg.png
Update rule:
https://images.datacamp.com/image/upload/v1666973295/Q_learning_equation_3cd6652b98.png
Policy bears:
https://pylessons.com/media/Tutorials/Reinforcement-learning-tutorial/Beyond-DQN/Beyond-DQN.jpg
Target Q-network :
https://builtin.com/sites/www.builtin.com/files/styles/ckeditor_optimize/public/inline-images/4_double-deep-q-learning.png
Results from the paper:
Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller, Playing atari with deep reinforcement learning (2013), https://arxiv.org/pdf/1312.5602