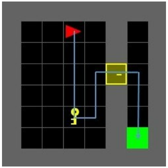
KEY AND DOOR WITH DQN

Vihang Patil
23rd April 2024
LIT AI LAB
Institute for Machine Learning
Slides: Andreas Radler and Maximilian Beck

Update

- Passing criteria for the course:
- Secure at least 50 points (Must)
- Pass minimum 2 out of 3 exercises (Must)

Find Key and Open Door with DQN

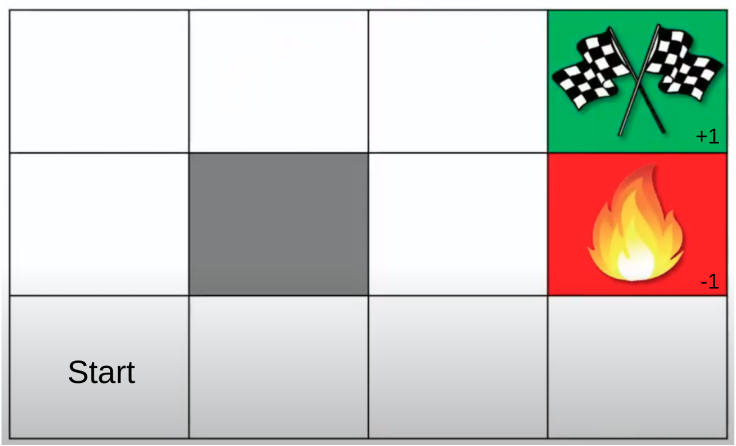


Minigrid Webpage

Agenda:

- Value and Q-functions
- Deep Q-learning Algorithm
- Tasks for DQN exercise

Value and Q functions - Intuition

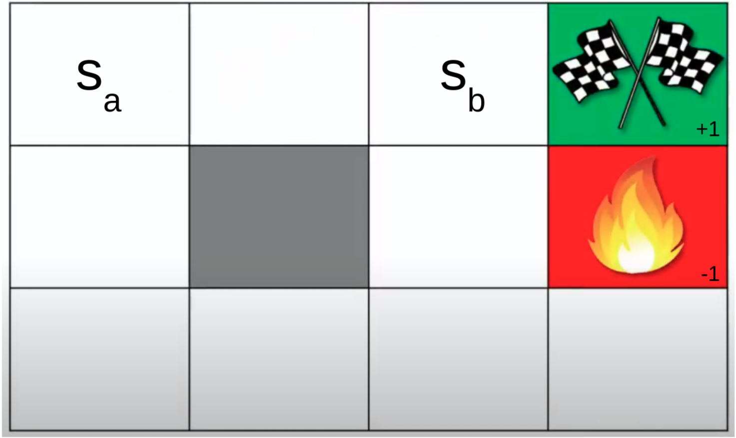


Value function

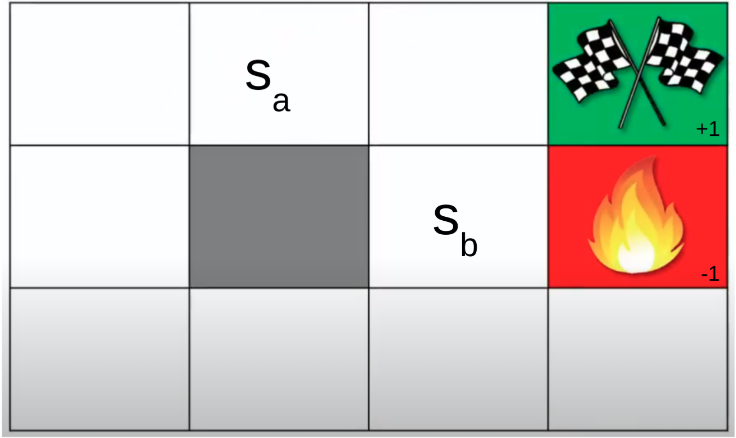
- Estimates how good it is for the agent to be in a given state
- Formally: "Expected return of a state s when following the policy π"
- Depends on:
  - Current policy π
  - Environment transition dynamics
  - Reward function
  - Discount factor gamma

$$V_\pi(s) = \mathrm{E}_\pi \left[R_t \middle| s_t = s \right] = \mathrm{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s \right]$$

Value function - $V(s_a)$ vs. $V(s_b)$



Value function - $V(s_a)$ vs. $V(s_b)$

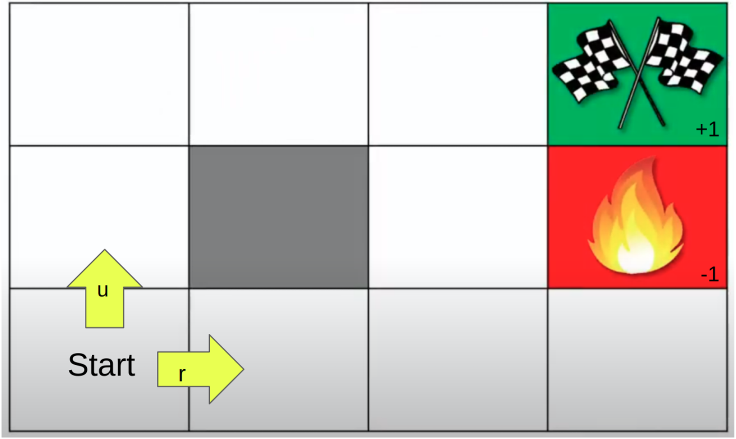


Q function

- Estimates how good it is to perform a given action in a given state
- Formally: "Expected return of taking the action a in state s, and following policy π afterwards"
- Remark: there is a direct connection between optimal V and Q functions
  - $V^*(s) = \max_a Q^*(s, a)$

$$Q_\pi(s, a) = \mathrm{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a \right]$$

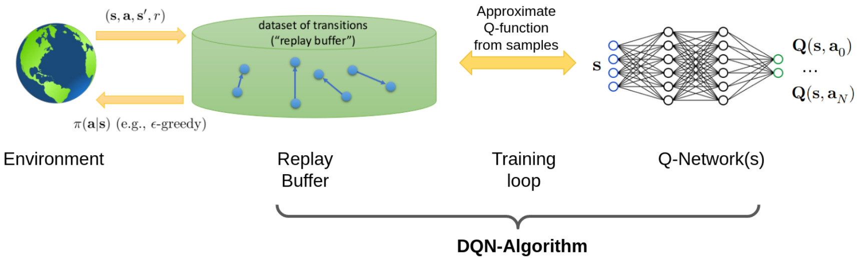
Q function - $Q(start, u)$ vs. $Q(start, r)$



Optimal V and optimal Q functions

- optimal Q-function Q^*
  - Formally: "The Q-function that has the highest value over all policies"
  - $Q^*(s, a) = \max_\pi Q_\pi(s, a)$
  - If we have the optimal Q-function we can extract the optimal policy for each state s by choosing action $a = \operatorname{argmax}_{a'} Q(s, a')$
- Q-learning aims to **learn** the **optimal Q function**
  - “DQN” is an algorithm for “Deep Q-Learning”

DQN Overview



Replay Buffer

- A buffer with limited capacity which stores transitions
- Why: Breaks the correlation in data, reduces variance of the update signal, makes the data distribution more stationary
- Transition: $(s_t, a_t, r_t, s_{t+1}, d_t)$
- Implementation:
  - buffer.add($s_t, a_t, r_t, s_{t+1}, d_t$) -> should add transition to the memory
  - buffer.sample() -> should sample a minibatch from the memory

Training loop: Minimize Temporal Difference (TD) error

Goal: learn the optimal Q-function $Q_\theta(s_t, a_t)$

1. Fill replay buffer B with transitions using some policy
2. Sample a batch $\{(s_i, a_i, r_i, s_i')\}_{i=1\ldots N}$ from B
3. Compute targets: $y_i = r_i + \gamma \max_{a'} Q_{\theta'}(s_i', a')$*
4. Use squared loss for the targets: $L_i = (Q_\theta(s_i, a_i) - y_i)^2$
   TD-error: $\delta_i = y_i - Q_\theta(s_i, a_i)$
5. Update Q-function with gradient descent: $\theta_{new} = \theta - \alpha \frac{dL_i}{d\theta}$
6. Update target network parameters θ' from time to time

* θ' denotes the target network, which is an older copy (from an earlier iteration) of the Q-network

DQN Pseudocode

DQN

Initialize Replay Memory B with Capacity M
Initialize Q function with network θ
Initialize Q target function with network θ'
Initialize environment: env = env.make("name_of_env")
Add preprocessing wrappers: env = wrap(env)
Initialize exploration factor ϵ, learning rate α, batch size m
discount factor γ, other hyperparameters

d: done = True for last state in a episode else False

DQN (cont.)

for $episode = 1$ to N **do**

s_t = env.reset()

while not done **do**

With probability ϵ select random action a_t

otherwise select a_t = $\text{argmax}_a Q(s_t, a; \theta)$

s_{t+1}, r_t, d_t, _ = env.step(a_t)

Store transition $(s_t, a_t, r_t, s_{t+1}, d_t)$ in B

Sample random batch from memory B: $(s_j, a_j, r_j, s_{j+1}, d_j)$

$$\text{targets: } y_j = \begin{cases} r_j & \text{For terminal } s_{j+1} \\ r_j + \gamma \cdot \max_a Q(s_{j+1}, a; \theta') & \text{For non terminal } s_{j+1} \end{cases} \quad (1)$$

Loss: $L(\theta) = (y_j - Q(s_j, a_j; \theta))^2$; Update θ: $\theta \leftarrow \theta - \alpha \cdot \nabla L(\theta)$

end while

Update target network θ': $\theta' = \tau \cdot \theta + (1 - \tau) \cdot \theta'$

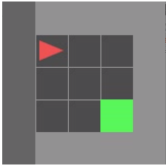
end for

Prioritized Experience Replay [Optional]

- Notion: Replace random sampling from memory with something better -> Based on some priority
- Idea: Higher priority based on TD error
  - Store TD error for each transition in a list
  - More samples with higher TD error are sampled
  - New samples are given max priority
  - Update the TD-error for samples which are sampled after update step
  - Problem: Results in lack of diversity $\rightarrow$ Overfitting when used with function approximation

See Prioritized Experience Replay paper or lecture slides for details

MinigridEmpty5x5ImgObs Environment

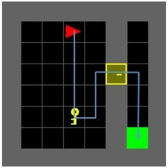


Minigrid repository

Easy environment, training takes <10 min, good for debugging

- **Observation space**: Symbolic, non-image observation (3, 7, 7)
- **Action space**: 7 discrete actions
- **Rewards**: $1 - 0.9\frac{\text{steps}}{\text{max steps}}$

MinigridDoorKey6x6ImgObs Environment (Graded Task)



Minigrid Webpage

More difficult environment, training takes 30 - 50 minutes

- **State space**: Symbolic, non-image observation (3, 7, 7)
- **Action space**: 7 discrete actions
- **Reward Range**: {0, 1}

Submission

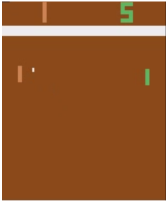
- Submission server: https://apps.ml.jku.at/challenge
- Export your trained agent as ONNX file
- Upload ONNX file to submission server
- Code is provided in Minigrid_DQN_exercise_2024.ipynb
- Submission closes on 12:59, May 20th.
- Submit your code, best model and a short report to Moodle
- Moodle submission stays open 24h after the challenge closes
- Please upload a zip file named k<studentid>.zip containing:
  - ☐ code.ipynb
  - ☐ model.onnx
  - ☐ report.pdf

JKU

Evaluation

- $< 0.5 : 0$
- $0.5 - 0.6 : 15$
- $0.6 - 0.7 : 18$
- $0.7 - 0.8 : 21$
- $>= 0.8 : 30$

Pong Environment



Pong Video

Gym Webpage

More difficult environment, training takes > 5h

After Preprocessing (see Atari Wrappers):

- **State space**: Stack of image observations (4, 84, 84)
- **Action space**: 6 discrete actions (only 3 are different)
- **Rewards**: {-1, 0, 1}