TEK 5040/9040 Reinforcement Learning (II)

Narada Warakagoda

Value function estimation (Temporal difference method)

- Disadvantage of Monte Carlo method (previous lecture):
  - Need to unroll the whole trajectory
- Temporal difference (TD) method
  - Need to unroll one time step
  - Generalization of incremental update (previous lecture)

Generalization of incremental update

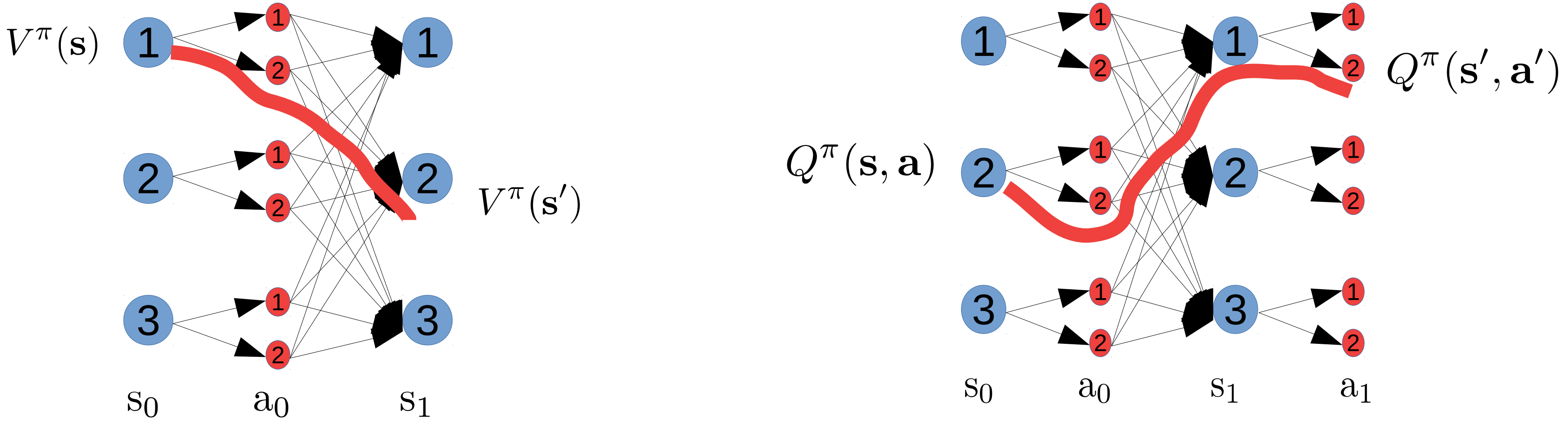
- Incremental update rule (exact)
- $$V_t(\mathbf{s}) \leftarrow V_{t-1}(\mathbf{s}) + \frac{1}{t}(R_t - V_{t-1}(\mathbf{s}))$$
- Approximate update rule with some constant
- $$V_t(\mathbf{s}) \leftarrow V_{t-1}(\mathbf{s}) + \alpha(R_t - V_{t-1}(\mathbf{s}))$$
- Use a "better" target
  - To get Return, we need to unroll the whole trajectory
  - Instead, take one step and use the Bellman like prediction as the target

Bellman equations

- Idea: The value of your starting point is the reward you expect to get from being there plus the value of the point you land next.

$$V^{\pi}(\mathbf{s}) = \underset{\substack{a\sim\pi \\ s'\sim P}}{E}\left[r(\mathbf{s},\mathbf{a}) + \gamma V^{\pi}(\mathbf{s}')\right]$$

$$Q^{\pi}(\mathbf{s},\mathbf{a}) = \underset{\mathbf{s}'\sim P}{E}\left[r(\mathbf{s},\mathbf{a}) + \gamma \underset{\mathbf{a}'\sim\pi}{E}\left[Q^{\pi}(\mathbf{s}',\mathbf{a}')\right]\right]$$



Temporal Difference method for V(s)

- Update equation $V_t(\mathbf{s}) \leftarrow V_{t-1}(\mathbf{s}) + \alpha(V_{t-1}^{\text{target}}(\mathbf{s}) - V_{t-1}(\mathbf{s}))$
- Approximate target with Bellman equation

$$V^{\pi}(\mathbf{s}) = \underset{\substack{a\sim\pi \\ s'\sim P}}{E}\left[r(\mathbf{s},\mathbf{a}) + \gamma V^{\pi}(\mathbf{s}')\right] \approx \left[r(\mathbf{s},\mathbf{a}) + \gamma V^{\pi}(\mathbf{s}')\right]$$

$$\Downarrow$$

$$V_{t-1}^{\text{traget}}(\mathbf{s}) \approx \left[r(\mathbf{s},\mathbf{a}) + \gamma V_{t-1}(\mathbf{s}')\right]$$

- Final update equation

$$V_t(\mathbf{s}) \leftarrow V_{t-1}(\mathbf{s}) + \alpha(\left[r(\mathbf{s},\mathbf{a}) + \gamma V_{t-1}(\mathbf{s}')\right] - V_{t-1}(\mathbf{s}))$$

  - $V_t(\mathbf{s})$ = value function estimate at time t for state $\mathbf{s}$
  - $r(\mathbf{s},\mathbf{a})$= reward for taking action $\mathbf{a}$ at state $\mathbf{s}$
  - $\mathbf{s}'$ = landed state after taking action $\mathbf{a}$ from state $\mathbf{s}$

TD method for V(s) II

- Can be viewed as evolving a table

t=0	t=1	t=2	t=3
V(A)	V(A)	V(A)	V(A)
V(B)	V(B)	V(B)	V(B)
V(C)	V(C)	V(C)	V(C)
V(D)	V(D)	V(D)	V(D)

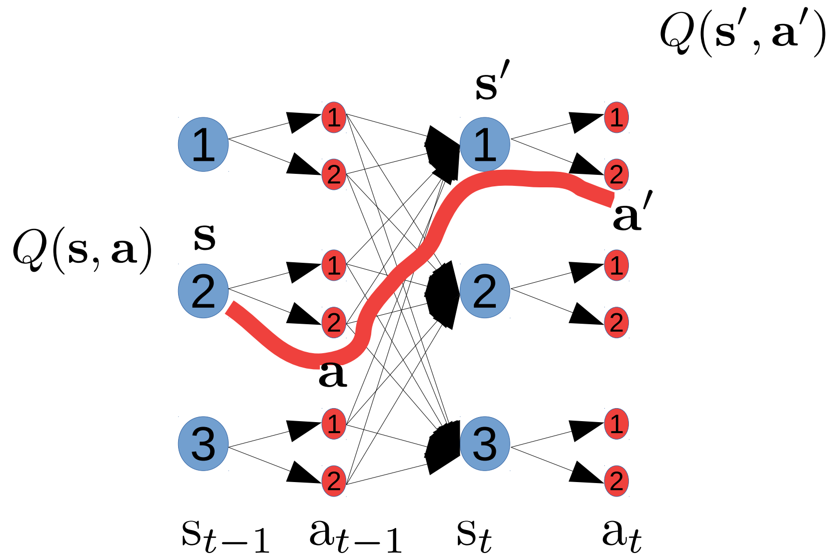
- Algorithm:
  - Initialize V(s) for all s randomly except V(terminal)=0
    - Loop for each episode
      - Choose a state S
      - Loop for each step
        - Sample an action A based on state S and the given policy
        - Take action A, observe reward R and next state S’
        - Update $V(S) \leftarrow V(S)) + \alpha[R + \gamma V(S') - V(S)]$
        - Update $S \leftarrow S'$

TD method for Q(s,a)

- Update equation

$$Q_t(\mathbf{s},\mathbf{a}) \leftarrow Q_{t-1}(\mathbf{s},\mathbf{a}) + \alpha([r(\mathbf{s},\mathbf{a}) + \gamma Q_{t-1}(\mathbf{s}',\mathbf{a}')] - Q_{t-1}(\mathbf{s},\mathbf{a}))$$

$$\begin{aligned}
Q_t(\mathbf{s},\mathbf{a}) &= \text{action value function at time } t \text{ evaluated on state } \mathbf{s} \text{ and action } \mathbf{a} \\
r(\mathbf{s},\mathbf{a}) &= \text{reward when taking action } \mathbf{a} \text{ from state } \mathbf{s} \\
\mathbf{s}' &= \text{state landed when taking action } \mathbf{a} \text{ from state } \mathbf{s} \\
\mathbf{a}' &= \text{action taken from state } \mathbf{s}'
\end{aligned}$$



TD method for Q(s,a) II

- State-Action-Reward-State-Action (SARSA) Algorithm
  - Initialize Q(s,a) for all s and a arbitrarily except for Q(.,a)=0 for terminal states
    - Loop for each episode
      - Choose S
      - Sample an action A based on S and policy
      - Loop for each step
        - Take action A, observe reward R and next state S'
        - Sample A' based on S' and policy
        - Update $Q(\mathbf{S},\mathbf{A}) \leftarrow Q(\mathbf{S},\mathbf{A}) + \alpha(R + \gamma Q(\mathbf{S}',\mathbf{A}')] - Q(\mathbf{S},\mathbf{A}))$
        - Update state and action $\mathrm{S} \leftarrow S' \quad A \leftarrow A'$

TD method Q(s,a) III

Action (1,2,3,4)

State (A,B,C)

Q(A,1)	Q(A,2)	Q(A,3)	Q(A,4)
Q(B,1)	Q(B,2)	Q(B,3)	Q(B,4)
Q(C,1)	Q(C,2)	Q(C,3)	Q(C,4)

t=t

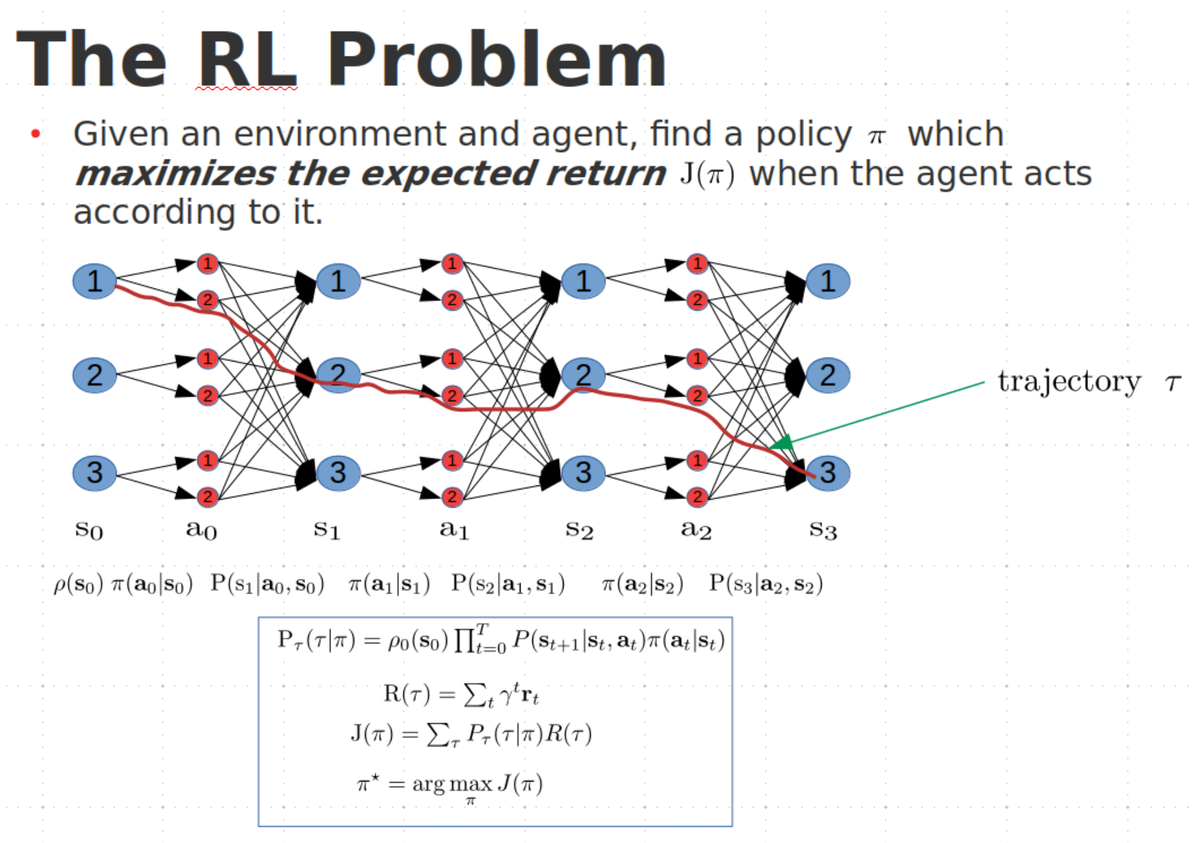
Q(A,1)	Q(A,2)	Q(A,3)	Q(A,4)
Q(B,1)	Q(B,2)	Q(B,3)	Q(B,4)
Q(C,1)	Q(C,2)	Q(C,3)	Q(C,4)

t=t+1

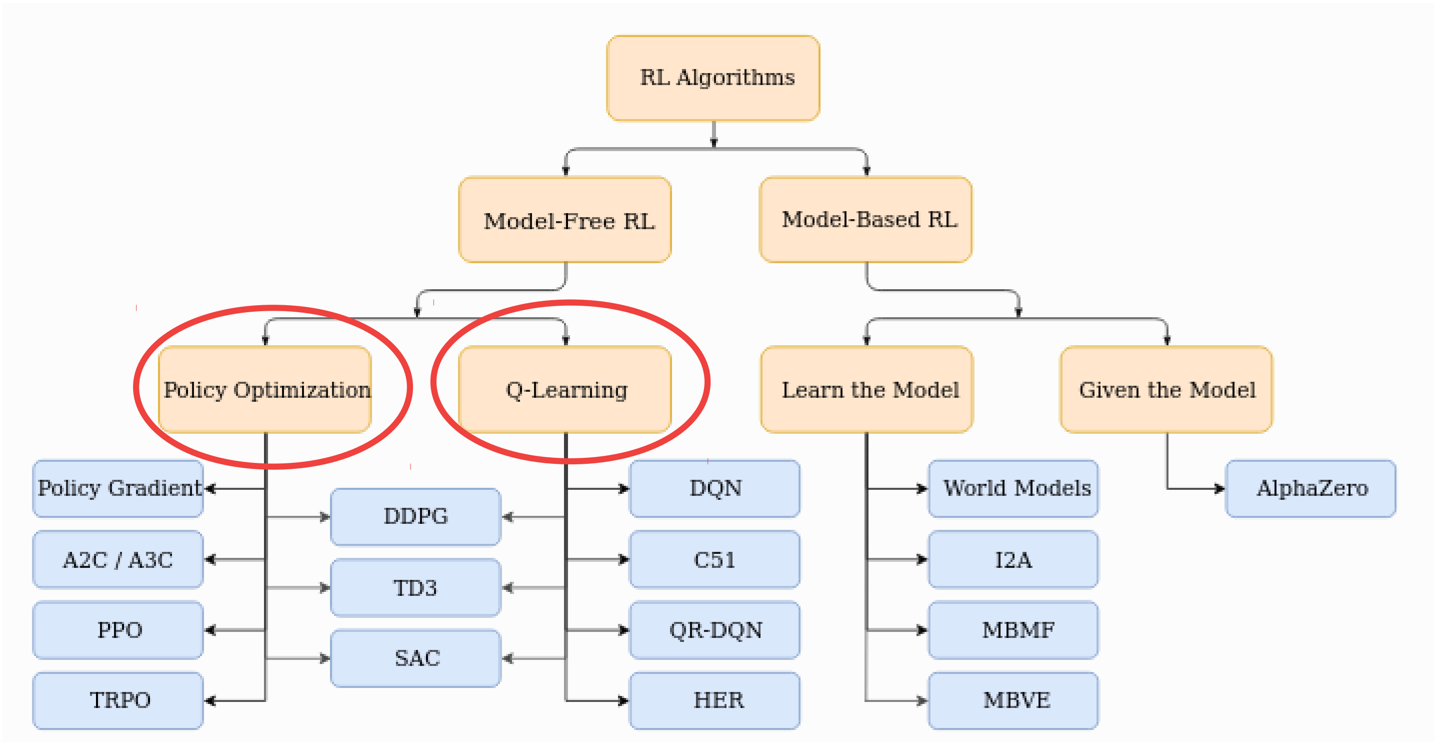
TD method as Q-table evolution

Learning the Policy

- The main goal of reinforcement learning is to learn the "optimal" policy
- The optimal policy would maximize the expected return



Taxonomy of RL algorithms



https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

Q-Learning

- Very similar to SARSA-algorithm and evolves a Q(s,a) table
- However:
  - SARSA
    - Has the goal of estimating Q(s,a)
    - is on-policy (i.e. Q-values are estimated using the given policy)
  - Q-learning
    - Has the goal of estimating optimal policy
    - Is off-policy (i.e. Q-values are estimated using a varying policy)

Q-Learning algorithm

- Algorithm returns the optimal Q -value

$$Q^\star(\mathbf{s},\mathbf{a}) = \max_\pi \underset{\tau\sim\pi}{E}\left[R(\tau)|\mathbf{s_0}=\mathbf{s},\mathbf{a}_0=\mathbf{a}\right]$$

- Then the optimal policy is defined by $\mathbf{a}^\star(\mathbf{s}) = \arg\max_{\mathbf{a}} Q^\star(\mathbf{s},\mathbf{a})$

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state},\cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., ϵ-greedy)
        Take action A, observe R, S'
        $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
        $S \leftarrow S'$
    until S is terminal

Find the maximum Q-value over all actions

How to choose actions?

- There is no policy to sample actions from (We are looking for a policy)
- Use the Q-values to sample actions
- ϵ -greedy policy
  - Select the best action $\mathbf{a}^\star = \arg\max_{\mathbf{a}} Q(\mathbf{S}, \mathbf{a})$ with probability 1-ϵ (Exploitation)
  - Select any of all other actions with probability $\epsilon/(K-1)$ where K is the number of possible actions (Exploration)
- ϵ -soft policy
  - Select any of all actions with probability at least ϵ/K

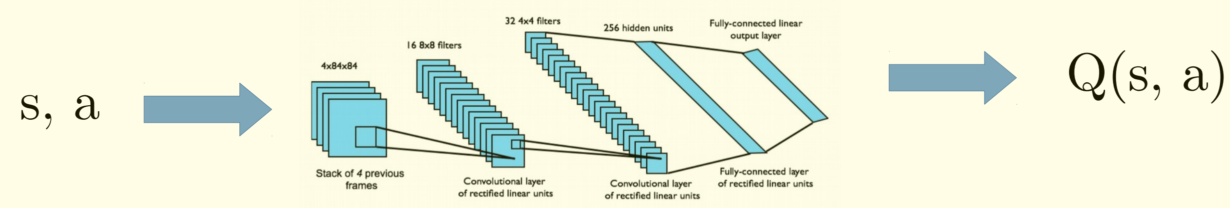
Deep Q-learning

- Main idea:
  - Replace the Q-table with a deep neural network
- Advantage:
  - Better function approximation

$$\text{Q: s} \times \mathbf{a} \rightarrow Q(\mathbf{s}, \mathbf{a})$$

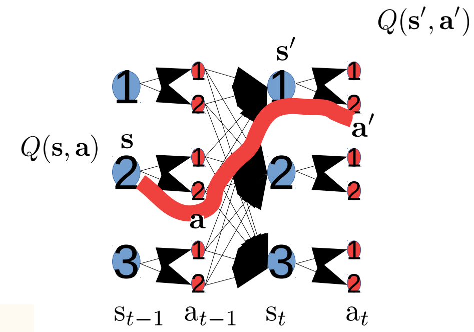
s, a → Q(s, a)

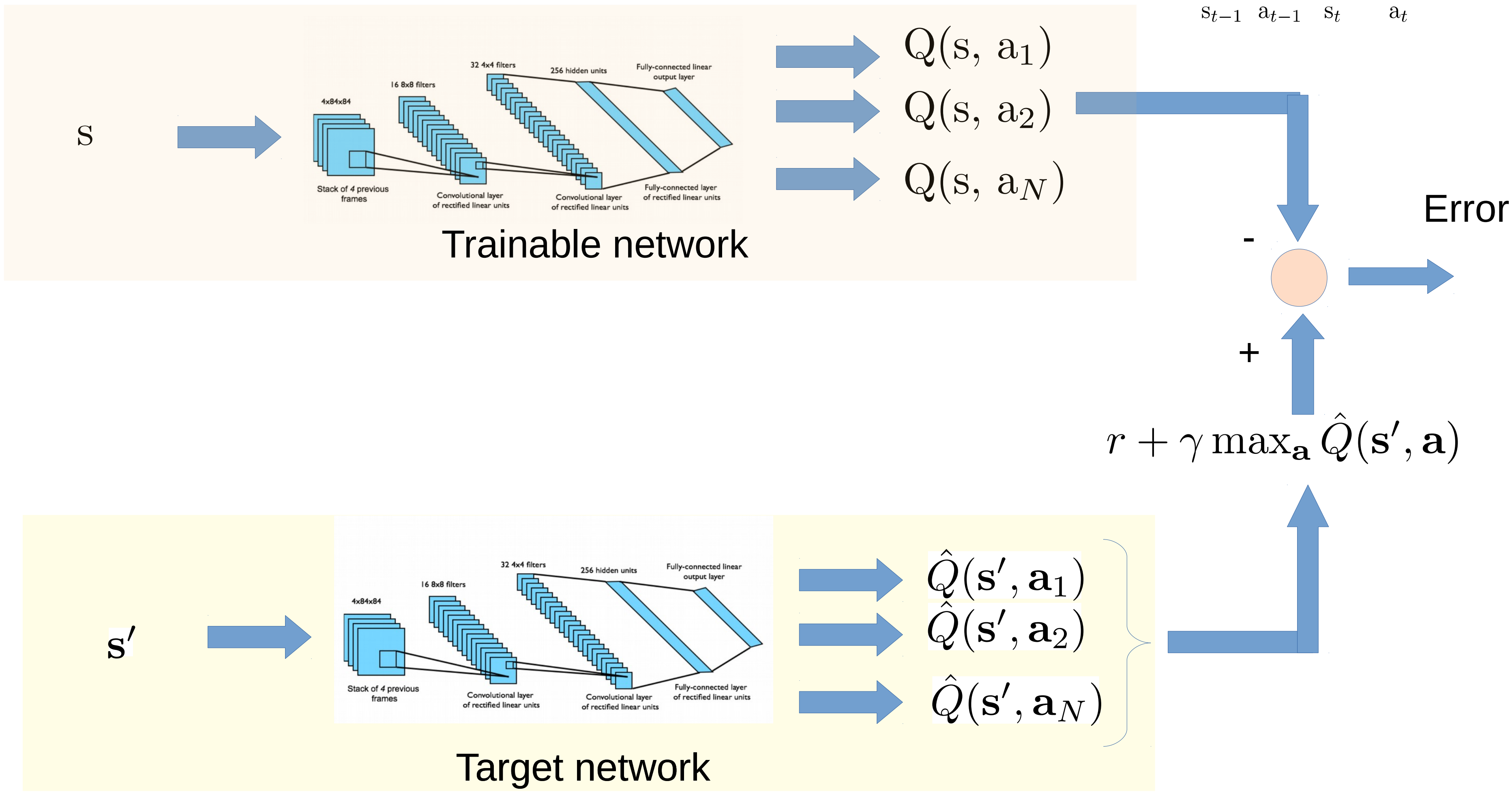
Q(A,1)	Q(A,2)	Q(A,3)	Q(A,4)
Q(B,1)	Q(B,2)	Q(B,3)	Q(B,4)
Q(C,1)	Q(C,2)	Q(C,3)	Q(C,4)



Details

- Data sample = $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$





Replay buffer

- Problem:
  - Target is non-stationary and targets are not independent
  -
- Solution:
  - Do not use samples $(\mathrm{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ as they arise
  - Save samples in a buffer (replay buffer)
  - Take random samples from from the buffer and train

Time indexed version of $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$

Deep Q-learning Algorithm

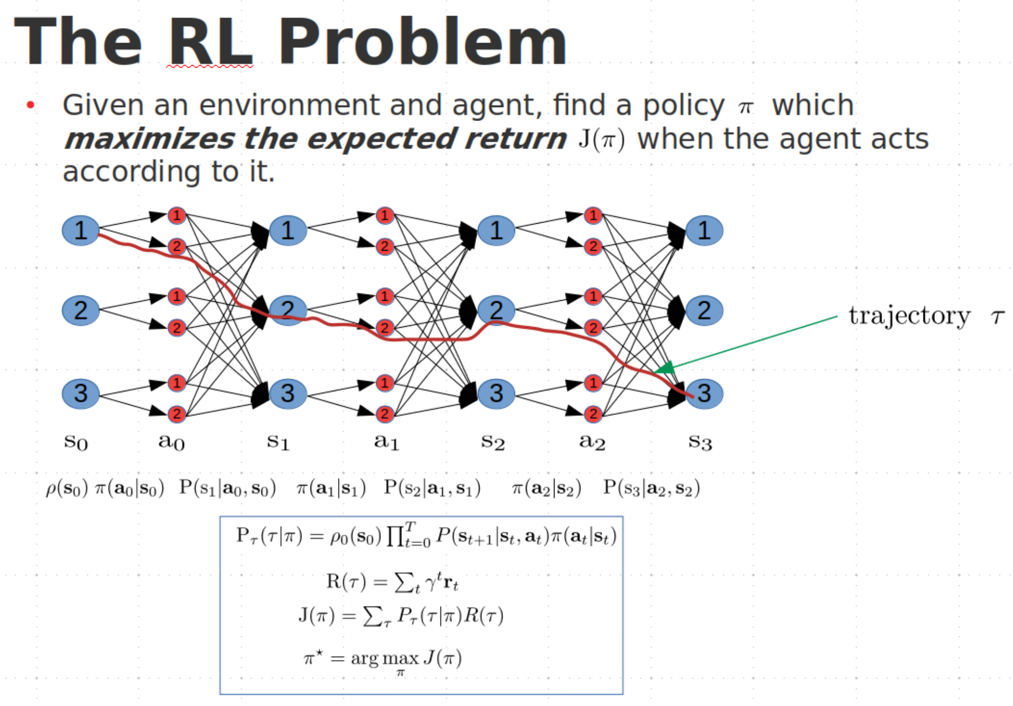
Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    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 $\left(\phi_t, a_t, r_t, \phi_{t+1}\right)$ in D
    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Policy gradients

- Goal of RL: Learn the policy
- Q-learning:
  - Approaches the problem indirectly via the optimal action value function Q and then $\mathbf{a}^\star(\mathbf{s}) = \arg\max_{\mathbf{a}} Q^\star(\mathbf{s}, \mathbf{a})$
  - More sample efficient
  - Less stable
- Policy gradients:
  - Optimizes the policy directly
  - Less sample efficient
  - More stable

Idea of policy gradient



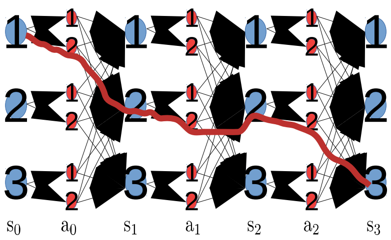
- We start with the definition of RL problem
- Assume that the policy π_θ is represented by a neural network with parameters θ
- Then $J(\pi) = J(\pi_\theta) = J(\theta)$ is a function of θ
- Optimize $J(\theta)$ with respect to the network parameters θ, by using the gradients $\nabla_\theta J(\theta)$
  - Gradient descent $\theta_{k+1} = \theta_k + \eta \nabla_\theta J(\theta_k)$

Derivation of policy gradient (background facts)

- The probability of a trajectory $\tau = (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \mathbf{s}_2, \mathbf{a}_2, \cdots)$

$$P(\tau|\theta) = \rho_0(s_0) \prod_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t).$$

$$\log P(\tau|\theta) = \log \rho_0(s_0) + \sum_{t=0}^{T} \left(\log P(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t) \right).$$



- Gradient log probability of a trajectory

$$\begin{aligned}\nabla_\theta \log P(\tau|\theta) &= \cancel{\nabla_\theta \log \rho_0(s_0)} + \sum_{t=0}^{T} \left(\cancel{\nabla_\theta \log P(s_{t+1}|s_t, a_t)} + \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \\ &= \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t).\end{aligned}$$

- Log derivative trick

$$\nabla_\theta P(\tau|\theta) = P(\tau|\theta) \nabla_\theta \log P(\tau|\theta).$$

Derivation of policy gradient

$$
\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}[R(\tau)] & \\
&= \nabla_\theta \int_\tau P(\tau|\theta)R(\tau) & \text{Expand expectation} \\
&= \int_\tau \nabla_\theta P(\tau|\theta)R(\tau) & \text{Bring gradient under integral} \\
&= \int_\tau P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)R(\tau) & \text{Log-derivative trick} \\
&= \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}[\nabla_\theta \log P(\tau|\theta)R(\tau)] & \text{Return to expectation form} \\
\therefore \nabla_\theta J(\pi_\theta) &= \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)R(\tau)\right] & \text{Expression for grad-log-prob}
\end{aligned}
$$

- Estimation of policy gradient for a set of trajectories $\mathcal{D}=\{\tau_1,\tau_2,\cdots,\tau_L\}$ as the sample mean

$$\hat{g} = \frac{1}{|\mathcal{D}|}\sum_{\tau\in\mathcal{D}}\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)R(\tau),$$

Policy gradient algorithm

- Initialize policy network parameters arbitrarily
- For iteration 1 to N
  - For episode 1 to L do
    - Choose an initial state
    - R=0
    - grad_log_array=[]
    - For step 1 to T do
      - Generate an action
      - Calculate $\nabla_\theta \log \pi_\theta(a_t|s_t)$ and append to grad_log_array
      - Perform action and accumulate the reward R=R+r
    - Multiply each element in grad_log_array by R and take sum (g_i)
  - Take the average of g_i i.e. $\hat{g} = \frac{1}{L}\sum_i g_i$
  - Update policy network parameters $\theta_{k+1} = \theta_k + \eta\hat{g}$

Interpretation of policy gradient

- Policy gradient formula

$$\hat{g} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau),$$

- If $R(\tau)$ is 1, then the gradient is just the gradient of log probability of the policy network output
- If $R(\tau)$ is positive, gradients are positively weighted and forces the trajectory to be higher probability
- If $R(\tau)$ is negative, gradients are negatively weighted and forces the trajectory to be lower probability.