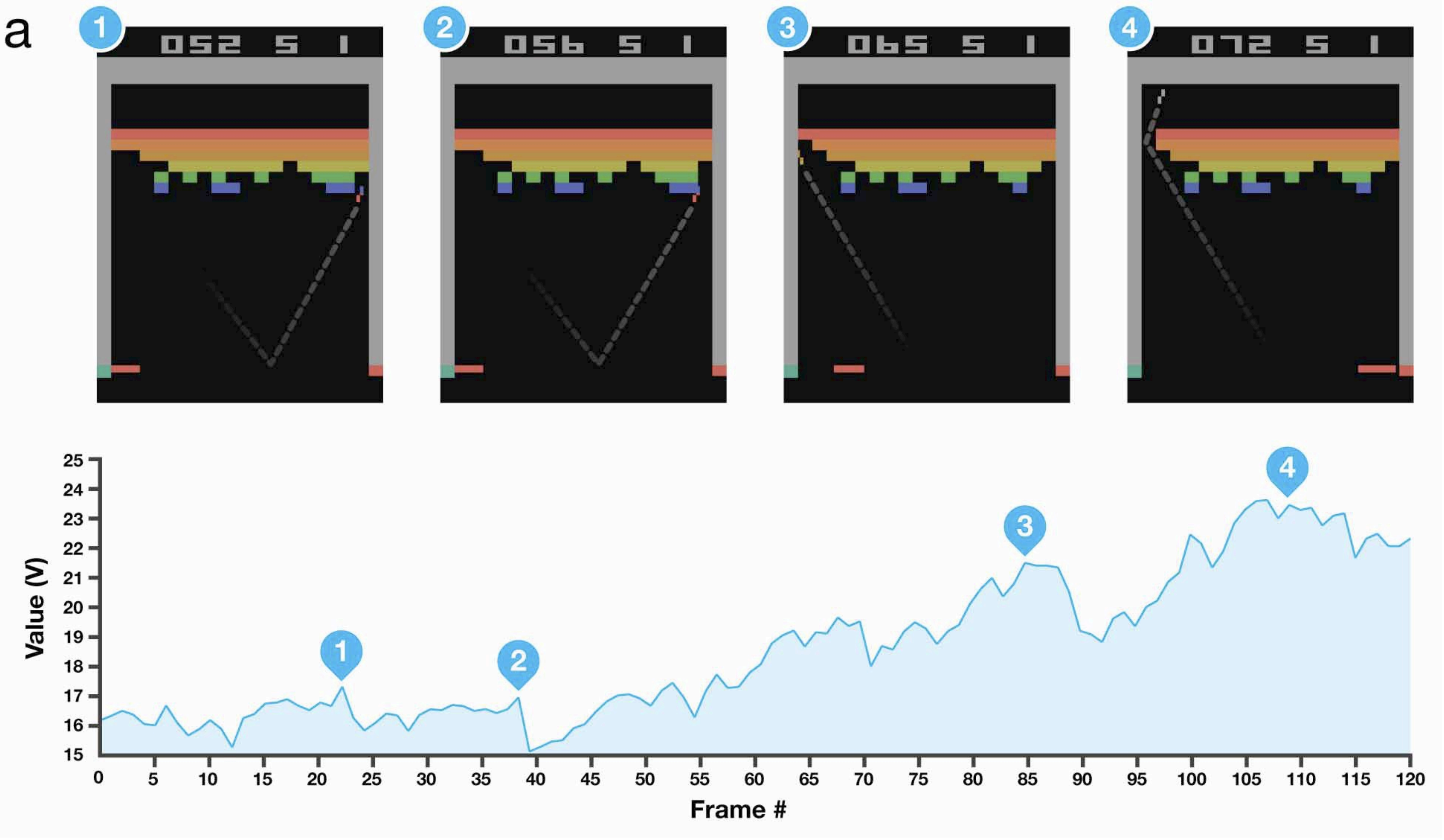
HSLU Lucerne University of Applied Sciences and Arts

Function Approximation Methods



Reinforcement Learning
December 1, 2022

Function Approximation

- Approximate the state- or action-value function by a parametrized function
- The objective (or loss function) is to minimize the mean square error between the value function and its approximation

$$\overline{\mathrm{VE}}(\mathbf{w}) = \sum_{s \in \mathcal{S}} \mu(s) \left[v_\pi(s) - \hat{v}(s, \mathbf{w}) \right]^2$$

Stochastic gradient descent

Assume that in each step we observe a state s and its (true) value under the policy.

We can use stochastic gradient-descend (SGD) by adjusting the weights vector to minimize the error in the observed examples

$$\begin{aligned}
\mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha\nabla\left[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)\right]^2 \\
&= \mathbf{w}_t + \alpha\left[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)\right]\nabla\hat{v}(S_t, \mathbf{w}_t)
\end{aligned}$$

TD(0) Prediction

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input:

a policy π

a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$ with parameters $\mathbf{w}$

($\hat{v}(\text{terminal}, \cdot)$) must be 0)

a step size parameter $\alpha > 0$

Initialize:

$\mathbf{w}$ arbitrarily

Loop for each episode)

Initialize $\mathcal{S}$

Loop for each step of the episode:

$A \leftarrow$ action given by π for S

Take action A, observe R, S'

$\mathbf{w} \leftarrow \mathbf{w} + \alpha[R + \gamma\hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})]\nabla\hat{v}(S, \mathbf{w})$

$S \leftarrow S'$

until S is terminal

Semi-gradient Sarsa for estimating $\hat{q} \approx q_*$

Input:

a differentiable function $\hat{q} : \mathcal{S} \times A \times \mathbb{R}^d \to \mathbb{R}$ with parameters $\mathbf{w}$

a step size parameter $\alpha > 0$, small $\epsilon > 0$

Initialize:

$\mathbf{w}$ arbitrarily

Loop for each episode)

Initialize S

Choose A as a function of $\hat{q}(S, ., \mathbf{w})$ (e.g., ϵ-greedy)

Loop for each step of the episode:

Take action A, observe R, S'

If S' is terminal:

$\mathbf{w} \leftarrow \mathbf{w} + \alpha[R - \hat{q}(S, A, \mathbf{w})]\nabla\hat{q}(S, A, \mathbf{w})$

Go to next episode

Choose A' as a function of $\hat{q}(S', ., \mathbf{w})$ (e.g., ϵ-greedy)

$\mathbf{w} \leftarrow \mathbf{w} + \alpha[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})]\nabla\hat{q}(S, A, \mathbf{w})$

$S \leftarrow S'$

$A \leftarrow A'$

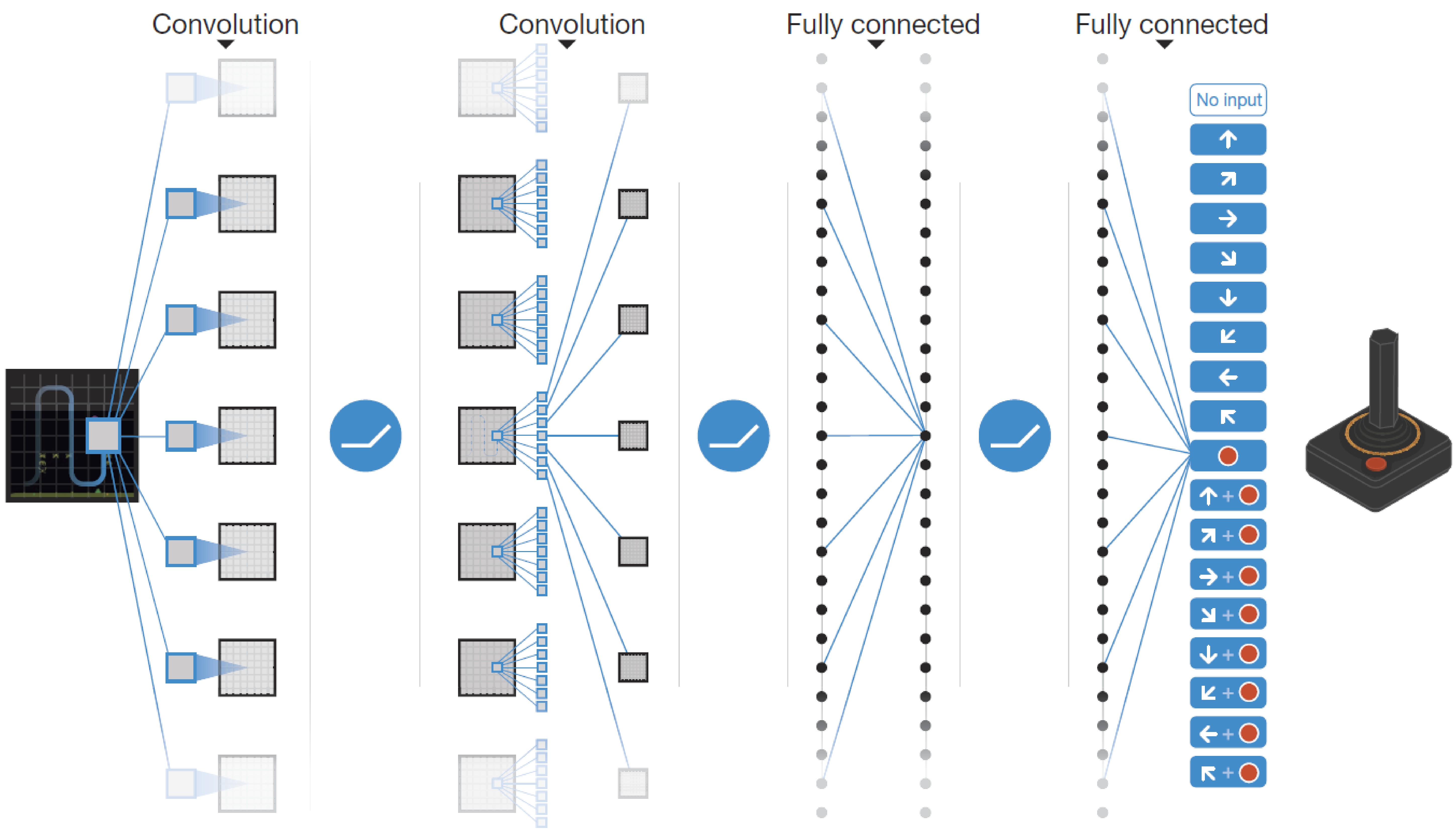
Deep Q-Learning

Deep Q-Learning uses a deep Q network (DQN), that estimates the action value function

- Uses experience replay
- Updates the action-values iteratively towards target
- Target values are only updated periodically

$$J(\mathbf{w}_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\Big[(r + \gamma \max_{a} \; Q(s', a', \mathbf{w}_i^-) - Q(s, a, \mathbf{w}_i)^2\Big]$$

Deep Q-Learning: Network Configuration



Deep Q-Learning with experience replay

Initialize:
  the replay memory D to capacity N
  action-value function Q with random weights $\mathbf{w}$
  target action-value function $\hat{Q}$ with weights $\mathbf{w}^- = \mathbf{w}$
Loop for each episode:
  Initialize S_1
  For every step $t = 1, T$ in the episode:
    Choose A_t as a function of $Q(S_t, ., \mathbf{w})$ (e.g., ϵ-greedy)
    Take action A_t, observe R_t, S_{t+1}
    Store transition (S_t, A_t, R_r, S_{t+1}) in D
    Sample a random minibatch of transitions (S_j, A_j, R_j, S_{j+1}) from D

$$y_j = \begin{cases} R_j & \text{if } S_{j+1} \text{ is terminal} \\ R_j + \gamma \max_{A'} \hat{Q}(S_{j+1}, A', \mathbf{w}^-) & \text{otherwise} \end{cases}$$

    Perform a gradient step on $(y_j - Q(S_j, A_j, \mathbf{w}))^2$ with respect to $\mathbf{w}$
    Every C steps reset $\hat{Q} = Q$