On-Policy Control with Approximation and Deep Q Networks (DQN)

Reinforcement learning – LM Artificial Iintelligence (2022-23)

Alberto Castellini
University of Verona

Summary

- Introduction
- Episodic Semi-Gradient Control
- Deep Q-Networks

Introduction

Introduction

- **Goal:** solve the **control problem** with parametric **approximation** of the **action-value function**

$$\hat{q}(s, a, \mathbf{w}) \approx q_*(s, a)$$

where $\mathbf{w} \in \mathbb{R}^d$ is a finite dimensional weight vector.

- We first restict our attention on the **on-policy** and **episodic** case

- We feature the **semi-gradient Sarsa algorithm**, the natural **extension** of **semi-gradient TD(0)** to
  - **action values**
  - **on-policy control**

Episodic Semi-Gradient Control

Episodic Semi-gradient Control

- The **extension** of state-value function approximators $\hat{v}(s,\mathbf{w})$ **to action-value function approximators** $\hat{q}(s, a, \mathbf{w})$ is **straightforward**

- **State-value functions:** training examples in the form $S_t \mapsto U_t$
- **Action-value functions:** training examples in the form $S_t, A_t \mapsto U_t$

- The update target U_t can be any approximation of $q_\pi(S_t, A_t)$ including the usual backed-up values, such as
  - The full Monte Carlo return G_t
  - The Sarsa return

Episodic Semi-gradient Control

- The **general gradient-descent update** for action-value prediction is

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\Big[U_t - \hat{q}(S_t, A_t, \mathbf{w}_t)\Big]\nabla\hat{q}(S_t, A_t, \mathbf{w}_t)$$

- For the **one-step Sarsa** method it is

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\Big[R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)\Big]\nabla\hat{q}(S_t, A_t, \mathbf{w}_t)$$

- We call this method **episodic semi-gradient one-step Sarsa**

- For a **constant policy** it **converges** as TD(0) and with the same error bound (see previous lecture)

$$\overline{\mathrm{VE}}(\mathbf{w}_{\mathrm{TD}}) \;\leq\; \frac{1}{1-\gamma}\min_{\mathbf{w}}\overline{\mathrm{VE}}(\mathbf{w})$$

Episodic Semi-gradient Control

Control methods: we need to couple

- Methods for **action-value prediction**
- Methods for **policy improvement and action selection**

- If the **action set is discrete and not too large** then we can use **techniques developed in the previous lecture**

Idea:

- For each **action** *a* of the current **state** S_t we compute $\hat{q}(S_t, a, \mathbf{w}_t)$
- Then we find the greedy action $A_t^* = \arg\max_a \hat{q}(S_t, a, \mathbf{w}_t)$
- **Policy improvement** is then performed by changing the estimation policy to a **soft-approximation** of the greedy policy, e.g., the ε-greedy policy

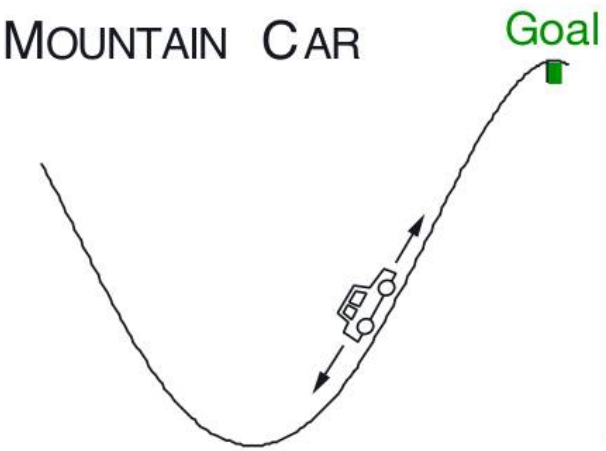
Episodic Semi-gradient Control

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

Input: a $\boxed{\text{differentiable action-value function parameterization } \hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}}$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize $\boxed{\text{value-function weights } \mathbf{w} \in \mathbb{R}^d}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
    $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
    Loop for each step of episode:
        Take action A, observe R, S'
        If S' is terminal:
            $\boxed{\mathbf{w} \leftarrow \mathbf{w} + \alpha \big[R - \hat{q}(S, A, \mathbf{w}) \big] \nabla \hat{q}(S, A, \mathbf{w})}$
            Go to next episode
        Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
        $\boxed{\mathbf{w} \leftarrow \mathbf{w} + \alpha \big[R + \gamma \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w}) \big] \nabla \hat{q}(S, A, \mathbf{w})}$
        $S \leftarrow S'$
        $A \leftarrow A'$

Example: Mountain Car



- **Actions**: full throttle forward (+1), full throttle reverse (-1), zero throttle (0)
- **Reward**: -1 at each step (until the car reaches the goal and the episode terminates)

- Simplified physics:

$$x_{t+1} \doteq bound\big[x_t + \dot{x}_{t+1}\big]$$

$$\dot{x}_{t+1} \doteq bound\big[\dot{x}_t + 0.001A_t - 0.0025\cos(3x_t)\big]$$

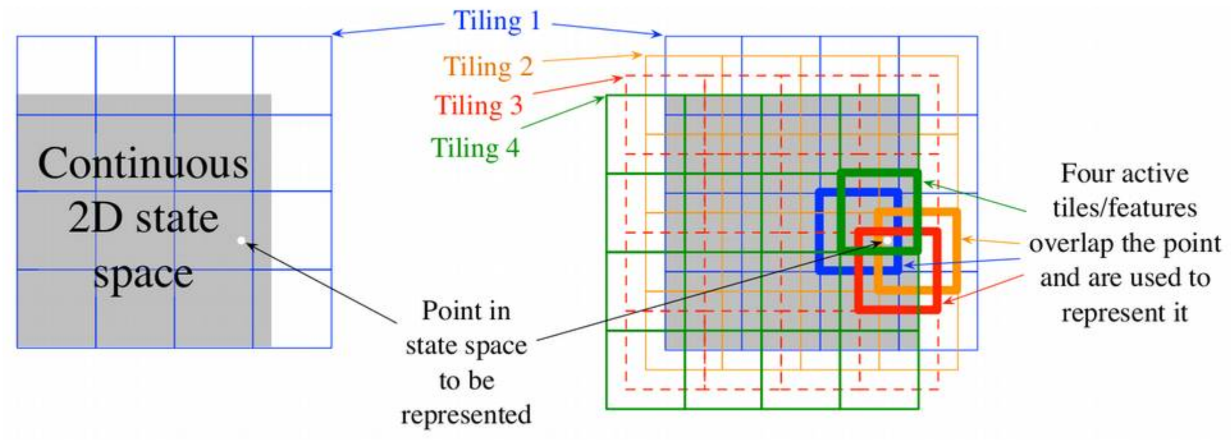
with *bound* operator

$$-1.2 \leq x_{t+1} \leq 0.5 \text{ and } -0.07 \leq \dot{x}_{t+1} \leq 0.07$$

- Episodes start in a random position $x_t \in [-0.6, -0.4)$ with zero velocity

Example: Mountain Car

- The two **continuous state variables** are converted to binary features using grid tiling (8 tilings, each tile covers 1/8th of the bounded distance in each dimension and asymmetrical offset as described in Section 9.5.4 of SutBut)

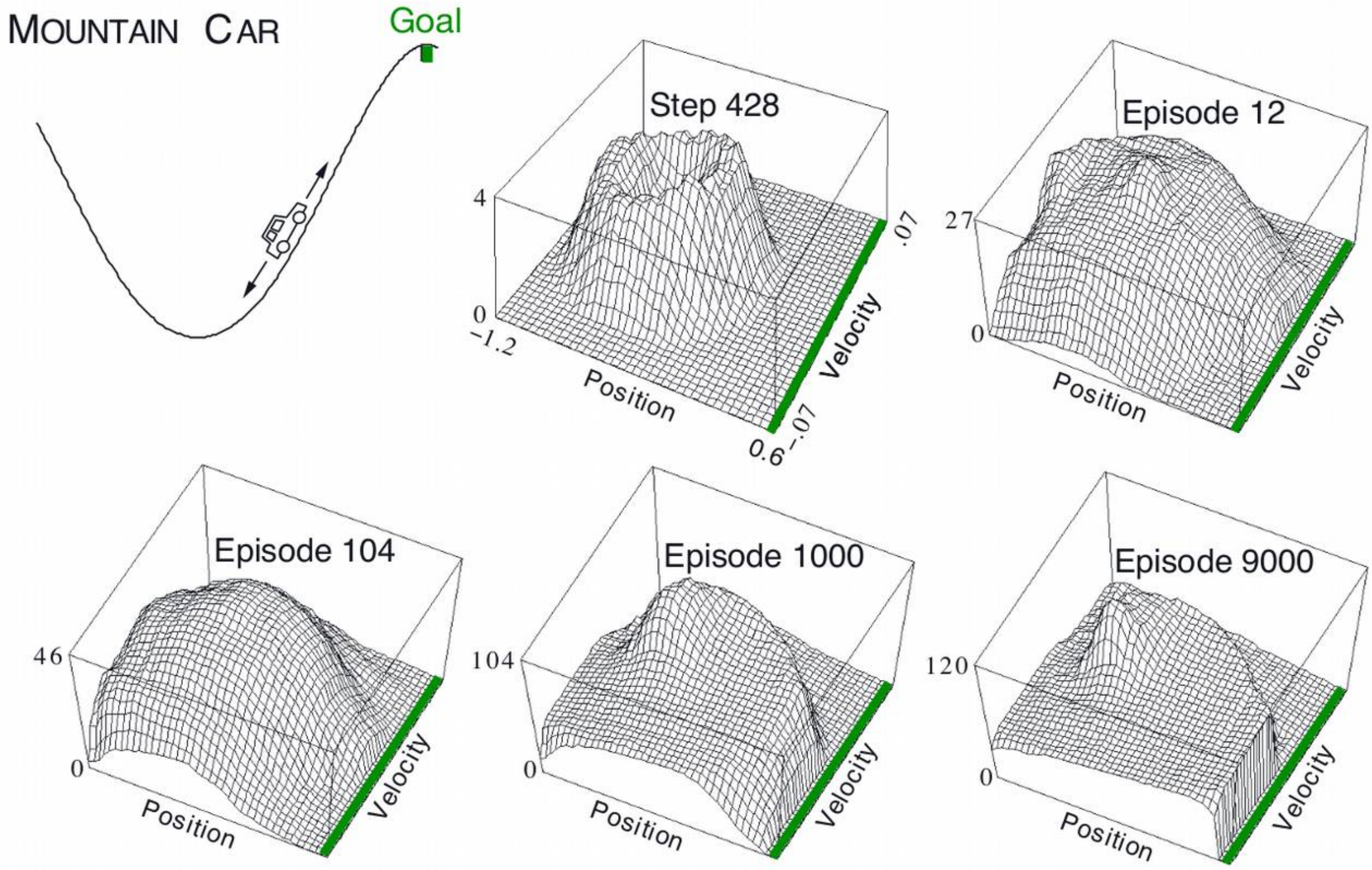


- The **feature vectors** created by tile coding are then **combined linearly** to approximate the **action-value function**

$$\hat{q}(s, a, \mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s, a) = \sum_{i=1}^{d} w_i \cdot x_i(s, a)$$

for each pair of state *s* and action *a*

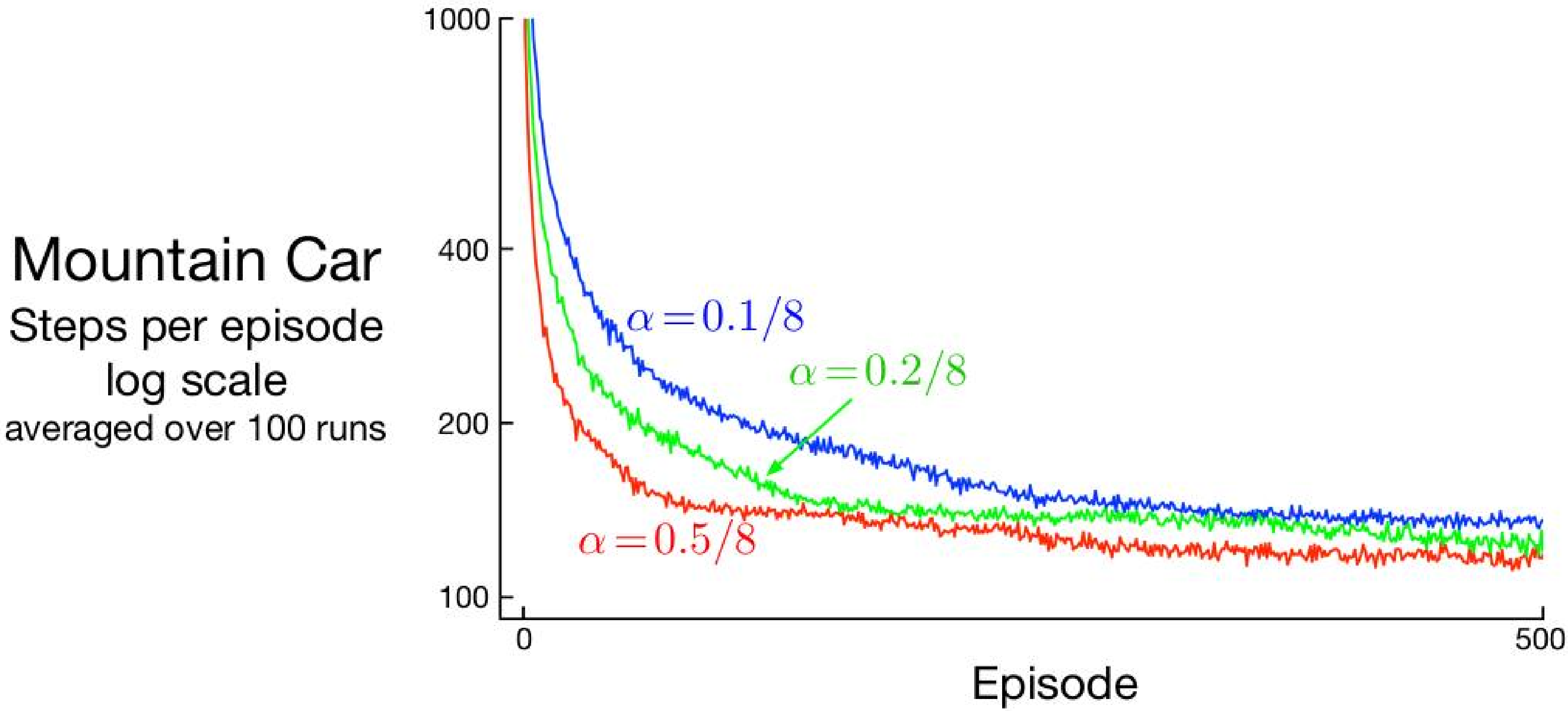
Example: Mountain Car



Cost-to-go function $(-\max_a \hat{q}(s, a, \mathbf{w}))$ learned during one run

- Initial action values were all zero (optimistic, true values are negative) causing extensive exploration even with null

Example: Mountain Car



Learning curves for **semi-gradient Sarsa** with **tile-coding** function approximation and ε-greedy action selection

References

- R. S. Sutton, A. G. Barto. Reinforcement learning, An Introduction. Second edition. Chapter 10

Deep Q Networks

ANNs for value function approximation in RL

- **Multi-layer ANNs** have been used for **function approximation** in RL since **1986**, when the **backpropagation** algorithm became popular as a method for learning internal representations (Rummelhart et al, 1986)

- Striking results have been obtained by **coupling RL** and **backpropagation** by Tesauro and colleagues with **TD-Gammon** and **WATSON** (Tesauro et al., 1994; Tesauro et al., 2012)

- In **2013**, Mnih and colleagues of **Google DeepMind** developed the first RL agent, called **Deep Q Network (DQN)** merging **Q-learning** and **deep convolutional ANNs** achieving **human level performance** in **Atari** games

- As TD-Gammon, **DQN** uses a **semi-gradient** form of a **TD** algorithm with gradients computed by **backpropagation** but DQN uses Q-learning instead of TD(λ)

Deep Q Networks

- **Basic idea: to use deep neural networks as a non-linear function approximator for the action value function in a semi-gradient form of Q-learning**

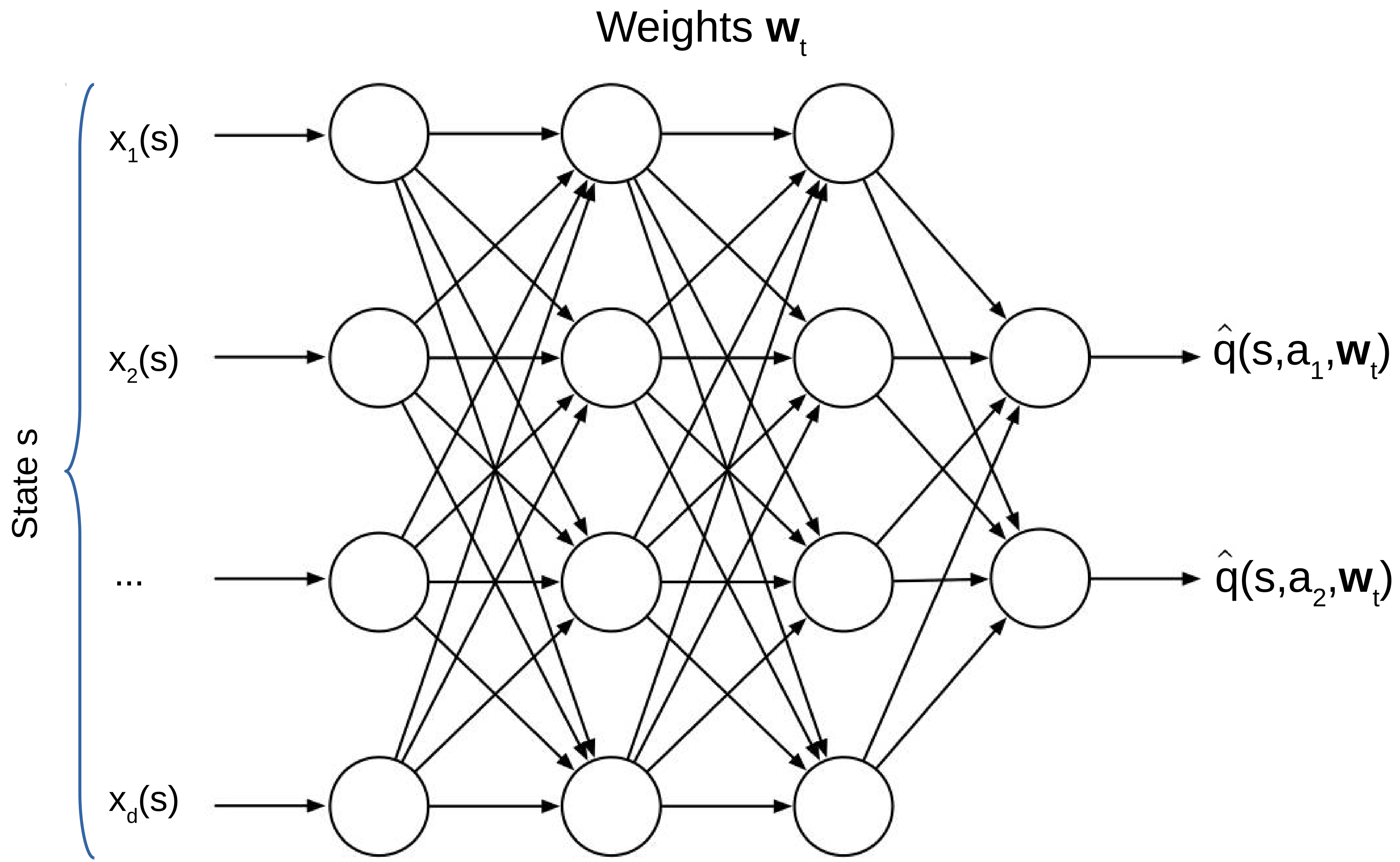
- We parametrize an approximate value function $\hat{q}(s,a,\mathbf{w}_t)$ using a **deep convolutional neural network** in which $\mathbf{w}_t$ are the parameters (weights) at iteration *t*.

- The neural network approximator is said **Q network** (e.g., see Fig. 1 of Mnih et al., 2015)

- **Input of the Q network:** raw sensor signals (current **state**). Deep NN can perform **feature construction "automatically"**, i.e., generating meaningful hierarchical abstractions in their layers

- **Output of the Q network:** estimated **optimal action values** for the input state (i.e., **one value for each action**)

Deep Q Networks



Deep Q Networks

- The **semi-gradient form of Q-learning** used by DQN to update the network's weight is

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \Big[\underbrace{R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t)}_{\textbf{Target value}} - \underbrace{\hat{q}(S_t, A_t, \mathbf{w}_t)}_{\textbf{Action value}} \Big] \nabla \hat{q}(S_t, A_t, \mathbf{w}_t)$$

where $\mathbf{w}_t$ is the vector of network weights, A_t is the action selected at step *t*, and S_t and S_{t+1} are the states at time *t* and *t+1* (i.e., network inputs)

- The gradient $\nabla \hat{q}(S_t, A_t, \mathbf{w}_t)$ can be computed by **backpropagation**

Deep Q Networks: problems and improvements

- **Problem:** RL is **unstable** or even **deverges** with **nonlinear function approximators** (e.g., ANNs) of the action-value function (Minh et al., 2015)

- **Causes:**
  - **C1: correlations** in the sequences of **observations** (states/features);
  - **C2:** small updates to *q* may **significantly change the policy** and change data distribution
  - **C3: correlation** between action-values $\hat{q}(S_t, A_t, \mathbf{w}_t)$ and target values $R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t)$

- **Solutions (Minh et al., 2015):**

  1) A biologically inspired mechanism for **experience replay**

  2) The usage of **two separate networks** to estimate action values in the Q-network and the target value

Deep Q Networks: experience replay

- **Idea:** Store agent experience in a replay memory then used to perform weight updates
- After each step a tuple $(S_t, A_t, R_{t+1}, S_{t+1})$ is added to the replay memory. This experience is accumulated over many episodes
- At each **step** **multiple Q-learning updates** (a **mini-batch**) are performed based on **experience** sampled uniformly at random from the replay memory
  - Q-learning is **off-policy**, it can be applied along unconnected trajectories
- **Advantages:**
  - **Reduced variance** of weight update (reduces cause **C2**)
  - The **correlation** in the sequences of observations is **eliminated** $\rightarrow$ one source instability is removed (reduces cause **C1**)

Deep Q Networks: double DQN

- **Two networks are used**. One for estimating **action values**, another for estimating **target values**

- The new update rule is

ANN 2 ANN 1

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[R_{t+1} + \gamma \max_a \tilde{q}(S_{t+1}, a, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t) \right] \nabla \hat{q}(S_t, A_t, \mathbf{w}_t)$$

- After C updates of the weights **w** of the action-value network (ANN 1) these weights are copied to the second network (ANN 2) used to compute the target values

Advantages:

- This improves stability reducing cause **C3**

Deep Q Network: Algorithm (Minh et al., 2015)

- See Algorithm 1 in (Minh et al., 2015)

Deep Q Networks: experimental settings

- In the popular works where DQN was first presented (Minh et al. 2013; Minh et al. 2015) the approach was evaluated on **49 Atari games**

- **Input**: 210x160 pixel image frames, 128 colors, 60Hz
- **Preprocessing**: images reduced to 84x84 arrays of luminecence
- **Stacked images**: the **four most recent** images were provided at each step to the agent → actual input had dimension 84x84x4
- **Network architecture**:
  - 3 hidden convolutional layers (rectifier nonlinearities act. function)

    → 32 20x20 feature maps

    → 64 9x9 feature maps

    → 64 7x7 feature maps
  - 1 fully connected hidden layer (512 neurons)
  - Output layer (18 neurons)

- **Reward**: +1 (increased game score), -1 (decreased game score), 0

Deep Q Networks: experimental setting

- ε**-greedy** policy with ε decreasing linearly over the first million frames, low value afterwards (50M frames in total, i.e., 38 days)
- **Input, output, ANN architecture and parameters** (e.g., step size, discount factor, etc.) were selected to perform well on a small selection of games, then kept **fixed for all games (generalization)**
- **Learning** was performed **independently for each game** (i.e., different parameters were learned for each game)

Deep Q Networks: results

- **Evaluations** performed on **30 sessions of each game**, each lasting up to **5 minutes** and beginning in a random initial state

- DQN performed (Minh et al. 2015)
  - **better than state-of-the-art algorithm (linear function approximation with hand-crafted features (Bellemare et al., 2013)) in 43 games**
  - **at a level comparable to professional humans in 29 games**

References

- R. S. Sutton, A. G. Barto. Reinforcement learning, An Introduction. Second edition. Chapter 16.5

- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M. (2013). **Playing atari with deep reinforcement learning.** ArXiv:1312.5602.

- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D. (2015). **Human-level control through deep reinforcement learning.** Nature, 518(7540):529–533.