

# Lecture 6: Q-Learning and DQN

Viliam Lisý & Branislav Bošanský

Artificial Intelligence Center  
Department of Computer Science, Faculty of Electrical Eng.  
Czech Technical University in Prague

[viliam.lisy@fel.cvut.cz](mailto:viliam.lisy@fel.cvut.cz)

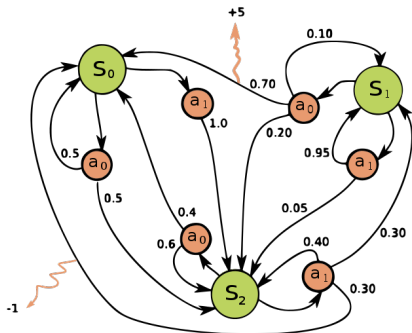
March, 2022

# Plan of today's lecture

- 1 RL algorithms in tabular representation for unknown MDP
- 2 Scaling up with Neural Networks
- 3 DQN algorithm and its application to Atari games

## Standard model for Reinforcement Learning problems

- $S$  – states
- $R$  – rewards
- $A$  – actions
- Discrete steps  $t = 0, 1, 2, \dots$
- Environment *dynamics*

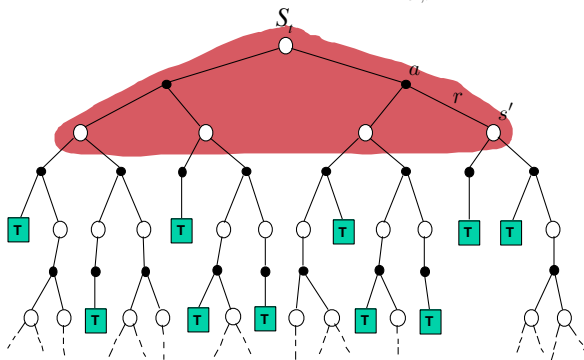


Source: Waldoalvarez @ wikimedia

$$p(s', r | s, a) \leftarrow Pr\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\}$$

# Dynamic Programming

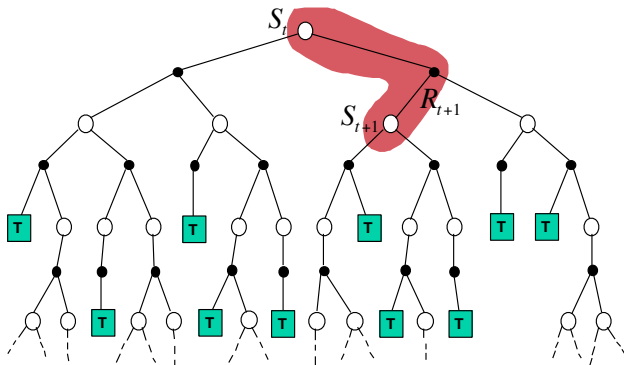
$$V(S_t) \leftarrow E_{\pi} [R_{t+1} + \gamma V(S_{t+1})] = \sum_a \pi(a|S_t) \sum_{s',r} p(s',r|S_t,a) [r + \gamma V(s')]$$



(Based on slides shared by R. Sutton)

# Simplest Temporal Difference Method

$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

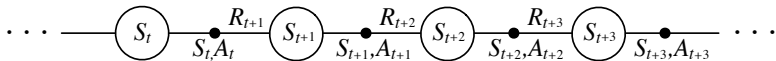


(Based on slides shared by R. Sutton)

# Learning An Action-Value Function

---

Estimate  $q_\pi$  for the current policy  $\pi$



After every transition from a nonterminal state,  $S_t$ , do this:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

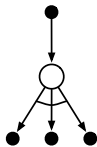
If  $S_{t+1}$  is terminal, then define  $Q(S_{t+1}, A_{t+1}) = 0$

# Q-Learning: Off-Policy TD Control

---

One-step Q-learning:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$



Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

Initialize  $S$

Repeat (for each step of episode):

Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

Take action  $A$ , observe  $R, S'$

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$ ;

until  $S$  is terminal

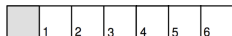
# Q-Learning Example

$$(2, \rightarrow) \Rightarrow -0.1$$

$$(3, \leftarrow) \Rightarrow -0.1$$

$$(2, \leftarrow) \Rightarrow -0.1$$

$$(1, \rightarrow) \Rightarrow -0.11$$



$\alpha = 0.1$ ,  $R = -1$  for each step

Default:  $(*, *) \Rightarrow 0$

$$(2, \leftarrow) \Rightarrow -0.09$$

$$(1, \leftarrow) \Rightarrow -0.1$$

$$(2, \leftarrow) \Rightarrow -0.191$$

$$(1, \leftarrow) \Rightarrow -0.19$$

Repeat (for each step of episode):

Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

Take action  $A$ , observe  $R$ ,  $S'$

$$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$S \leftarrow S'$ ;

until  $S$  is terminal



## Q-Learning: Off-Policy TD Control

One-step Q-learning:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$



Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$   
Repeat (for each episode):

Initialize  $S$

Repeat (for each step of episode):

Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

Take action  $A$ , observe  $R, S'$

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$S \leftarrow S'$ ;

until  $S$  is terminal

## $\epsilon$ -Greedy Action Selection

In greedy action selection, you always exploit

In  $\epsilon$ -greedy, you are usually greedy, but with probability  $\epsilon$  you instead pick an action at random (possibly the greedy action again)

This is perhaps the simplest way to balance exploration and exploitation

Algorithm  $\epsilon$ -Greedy:

Initialize, for  $a = 1$  to  $k$ :

$$Q(a) \leftarrow 0$$

$$N(a) \leftarrow 0$$

Repeat forever:

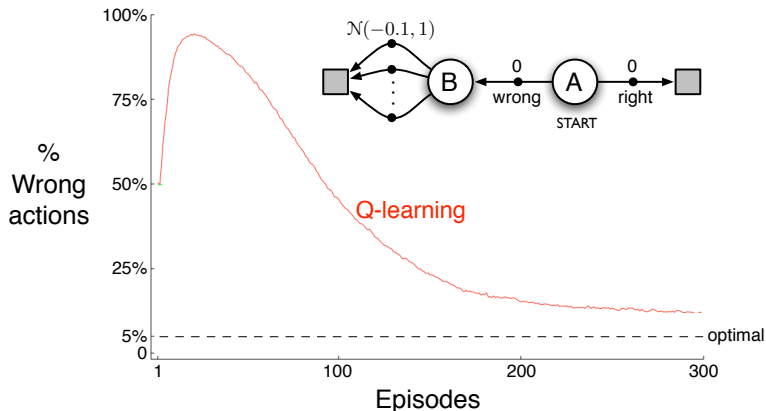
$$A \leftarrow \begin{cases} \arg \max_a Q(a) & \text{with probability } 1 - \epsilon \quad (\text{breaking ties randomly}) \\ \text{a random action} & \text{with probability } \epsilon \end{cases}$$

$$R \leftarrow \text{bandit}(A)$$

$$N(A) \leftarrow N(A) + 1$$

$$Q(A) \leftarrow Q(A) + \frac{1}{N(A)} [R - Q(A)]$$

# Maximization Bias



**Tabular Q-learning:** 
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

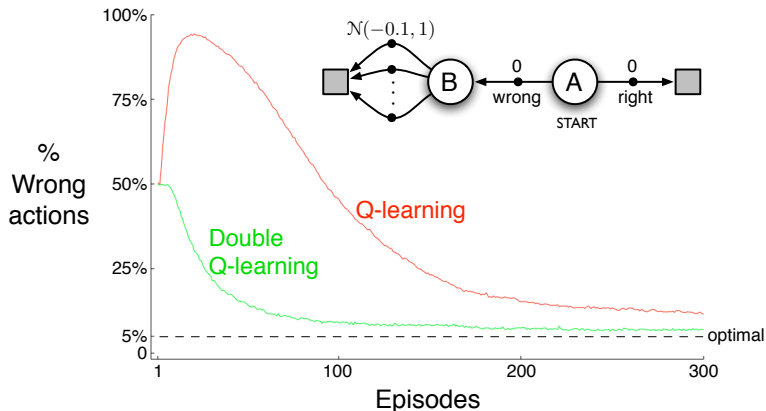
A solution to mitigate the maximization bias by van Hasselt [2010]

- Train two action-value functions  $Q_1$  and  $Q_2$
- Do Q-learning on both, but
  - never on the same time steps ( $Q_1$  and  $Q_2$  are independent)
  - pick  $Q_1$  or  $Q_2$  at random to be updated on each step
- If updating  $Q_1$  use  $Q_2$  for the value of the next state:

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \left( R_{t+1} + Q_2(S_{t+1}, \arg \max_a Q_1(S_{t+1}, a)) - Q_1(S_t, A_t) \right)$$

- Action selection can use a combination of  $Q_1$  and  $Q_2$

# Maximization Bias Mitigated



## Double Q-learning:

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma Q_2(S_{t+1}, \arg \max_a Q_1(S_{t+1}, a)) - Q_1(S_t, A_t) \right]$$

Create a program that would learn to play any Atari game



Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). [The arcade learning environment: An evaluation platform for general agents](#). *Journal of Artificial Intelligence Research*, 47, 253-279.

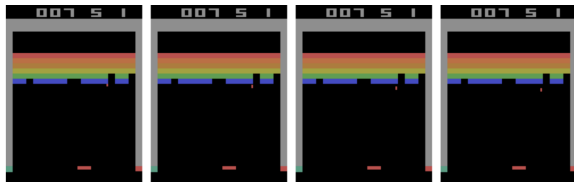
“This work bridges the divide between high-dimensional sensory inputs and actions, resulting in the first artificial agent that is capable of learning to excel at a diverse array of challenging tasks.”



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, 529-533.

States:

- Using four consecutive frames as state:



(Source: Greg Surma @ medium.com)

- Reduction of image size:  $210 \times 160 \times 3 \rightarrow 84 \times 84 \times 1$

Actions:

- $2 \times 8$  directions of the joystick + button

Transitions are taken directly from a game emulator

Rewards:

- Based on the game score
- Any score increase  $\rightarrow +1$ , any score decrease  $\rightarrow -1$

# How big is the MDP?

Assume we would quantise the colours to just black and white.  
The number of possible states of the MDP is then:

$$2^{84 \times 84 \times 4} = 2^{28224} \approx 10^{8496}.$$

There are estimated  $10^{80}$  atoms in the observable universe.  
Hence, a tabular representation of the  $q$  function may not work.

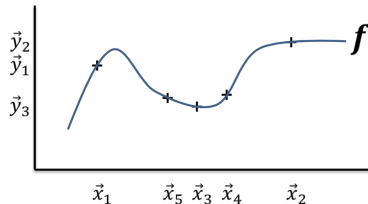


A useful tool for AI, which is **not** a focus of this course

Supervised learning = fitting a (high dimensional) function

For a data set  $(\vec{x}_i, \vec{y}_i)$ , find a function  $f$  that minimizes:

$$\frac{1}{n} \sum_i \|f(\vec{x}_i) - \vec{y}_i\|.$$



For example,  $f(2) = 2, f(3) = 3, f(4) = 4, f(5) = 5$ .

Q function is just a high-dimensional function approximable by a Neural network.

$$q(s, a) : \mathbb{R}^{28 \times 28} \times \mathcal{A} \rightarrow \mathbb{R}$$

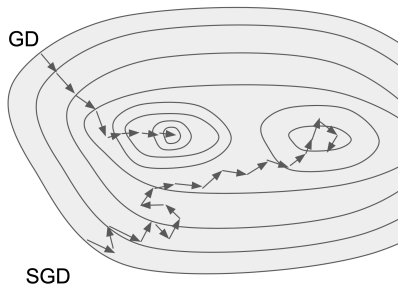
# Stochastic Gradient Descent

Dataset  $D = (\vec{x}_i, \vec{y}_i)$

Neural network  $f_w$  with weights  $w \in \mathbb{R}^m$

Loss:  $l(D, w) = \frac{1}{|D|} \sum_i \|f_w(\vec{x}_i) - \vec{y}_i\|$ .

Gradient descent:  $w' = w - \alpha \frac{\partial l(D, w)}{\partial w}$



Mini-batched version of the loss function:

For a uniformly selected subset of data  $\tilde{D} \subset D$  called a minibatch define the approximate loss:  $\hat{l}(\tilde{D}, w) = \frac{1}{|\tilde{D}|} \sum_i \|f_w(\vec{x}_i) - \vec{y}_i\|$

and update:  $w' = w - \alpha \frac{\partial \hat{l}(\tilde{D}, w)}{\partial w}$ .

It works, because  $\mathbb{E} \hat{l}(\tilde{D}, w) = l(D, w)$ .

**Algorithm 1: deep Q-learning with experience replay.**

Initialize replay memory  $D$  to capacity  $N$

Initialize action-value function  $Q$  with random weights  $\theta$

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  $\varepsilon$  select a random action  $a_t$

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

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  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $D$

Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $D$

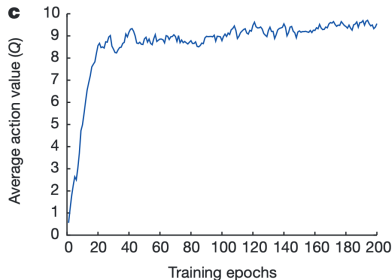
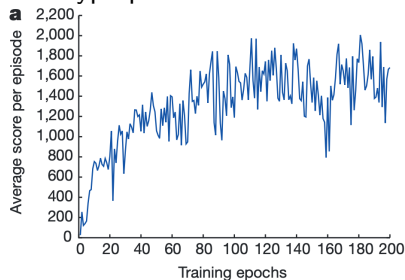
Set  $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  with respect to the network parameters  $\theta$

Every  $C$  steps reset  $\hat{Q} = Q$

**End For**

Trained for each game separately, but using the same architecture and hyperparameters.

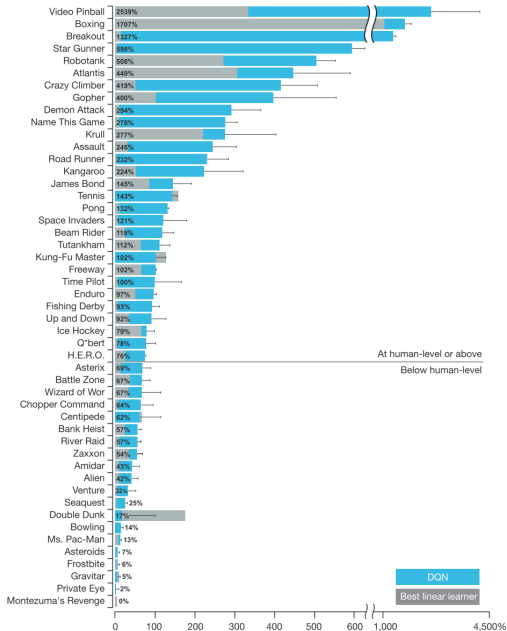


Convergence curve for "Space Invaders". One epoch is 520k frames.  $\epsilon = 0.05$ .

Training details: Minibatch size 32; exploration scaled from 1.0 to 0.1 over 1M frames and then fixed; overall 50M frames of training (38 days); replay buffer for 1M most recent frames. Probably 10 days of training per game and agent (not reported).

Breakout  
Space Invaders

# Results Relative to an Expert Human



<https://forms.gle/9nMjNQpEygtYs3XA6>  
(also available from courseware list of lectures)

RL can solve huge MDPs without their explicit knowledge.  
Key components of RL algorithms are policy evaluation and policy improvement.

Just using these steps on whole state space leads to

- policy iteration
- value iteration.

These algorithms are not super fast, but extremely versatile.

- Updates of just selected states
- Minimal / stochastic updates of policy and values
- Function approximation
- Endless modifications explored in RL literature