

# Model Predictive Control and Reinforcement Learning

## – Lecture 05: RL with Function Approximation –

Joschka Boedecker and Moritz Diehl

University of Freiburg

School on Model Predictive Control and Reinforcement Learning  
Freiburg, 7-16 October 2026

universität freiburg



NTNU

Norwegian University of  
Science and Technology

Yesterday: Planning by dynamic programming, solve a *known, discrete* MDP.

## Policy Iteration

Alternate **evaluating** the value function  $v_\pi$  and **improving** the policy  $\pi$  to convergence.

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi^* \xrightarrow{E} V^*$$

## Value Iteration

Evaluate just once and combine it with the policy improvement step.

$$\begin{aligned} V_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma V_k(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s'} P(s' | s, a) [r(s, a) + \gamma V_k(s')] \end{aligned}$$

Dynamic programming is optimal, but it requires knowledge of the dynamics and is too computationally expensive → Approximate Dynamic Programming

$$V_{k+1}(s) = \sum_{s', r} P(s'|s, a) [r(s, a) + \gamma V_k(s')]$$

In this lecture:

- ▶ **Local state updates:** We update only the visited states  $s, s'$  instead of solving the entire system at once.
- ▶ **No model required:** The transition model  $P(s'|s, a)$  (and the reward function  $r(s, a)$ ) is not needed; we sample  $s', r$  directly from interaction.
- ▶ **Function approximation:** Instead of a tabular  $V$  we use function approximators.

**We will repeat temporal difference methods!**

And we will maximize rewards  $r(s, a)$  instead of minimizing costs  $\ell(s, a)$ , and define  $R_{t+1} := r(s_t, A_t)$ .

## Temporal Difference Methods

- Model-free Learning

- TD Prediction (learning  $V^\pi$  and  $Q^\pi$ )

- TD Control (learning  $\pi^*$ )

## RL with Function Approximation

- Incorporating Function Approximation in RL

- Semi-gradient Methods

- Deep Q-Networks (DQN)

## Summary

## Temporal Difference Methods

- Model-free Learning

- TD Prediction (learning  $V^\pi$  and  $Q^\pi$ )

- TD Control (learning  $\pi^*$ )

## RL with Function Approximation

- Incorporating Function Approximation in RL

- Semi-gradient Methods

- Deep Q-Networks (DQN)

## Summary



Model-free prediction and control. Estimate/ optimize the value function of an *unknown* MDP using Temporal Difference Learning.

- ▶ TD is model-free: no knowledge required about MDP dynamics
- ▶ TD methods learn from episodes of *experiences*  
*experiences* = sequences of states, actions, and rewards
- ▶ TD learns from *incomplete* episodes by bootstrapping
- ▶ Bootstrapping: update estimated based on other estimates without waiting for a final outcome (update a guess towards a guess)



- ▶ Goal: learn the state-value function  $V^\pi$  for a given policy  $\pi$

$$S_0, A_0, R_1, \dots, S_T \sim \pi$$

- ▶ Recall: the *return* is the total discounted reward

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$$

- ▶ Recall: the value function is the expected return

$$V^\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma V^\pi(S_{t+1}) | S_t = s]$$

- ▶ Idea: estimate  $V^\pi$  from experience by averaging the returns observed after visits to that state  $\rightarrow$  Use empirical mean return instead of expected return
- ▶ Estimating  $V^\pi$  directly from  $G_t$  leads to *Monte Carlo methods* (not in the focus of this school). Estimating  $V^\pi$  from  $R_{t+1} + \gamma V^\pi(S_{t+1})$  leads to *Temporal Difference methods*.

- We can compute the mean of a sequence  $x_1, x_2, \dots$  incrementally:

$$\begin{aligned}\mu_k &= \frac{1}{k} \sum_{j=1}^k x_j \\ &= \frac{1}{k} \left( x_k + \sum_{j=1}^{k-1} x_j \right) \\ &= \frac{1}{k} \left( x_k + (k-1) \frac{1}{k-1} \sum_{j=1}^{k-1} x_j \right) \\ &= \frac{1}{k} (x_k + (k-1) \mu_{k-1}) \\ &= \mu_{k-1} + \frac{1}{k} (x_k - \mu_{k-1})\end{aligned}$$



- ▶ Thus, we can update  $V$  incrementally by:

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)),$$

where  $\frac{1}{N(s)}$  is the state-visitation counter

- ▶ Instead  $\frac{1}{k}$ , we can use step size  $\alpha$  to calculate a running mean:

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

## Monte Carlo Update

Update value  $V(S_t)$  towards the *actual* return  $G_t$ .

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

$\alpha$  is a step-size parameter.

## Simplest temporal-difference learning algorithm: TD(0)

Update value  $V(S_t)$  towards the *estimated* return  $R_{t+1} + \gamma V(S_{t+1})$ .

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

- ▶  $R_{t+1} + \gamma V(S_{t+1})$  is called the *TD target*
- ▶  $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$  is called the *TD error*

## Tabular TD(0) for estimating $v_\pi$

Input: the policy  $\pi$  to be evaluated

Algorithm parameter: step size  $\alpha \in (0, 1]$

Initialize  $V(s)$ , for all  $s \in \mathcal{S}^+$ , arbitrarily except that  $V(\text{terminal}) = 0$

Loop for each episode:

    Initialize  $S$

    Loop for each step of episode:

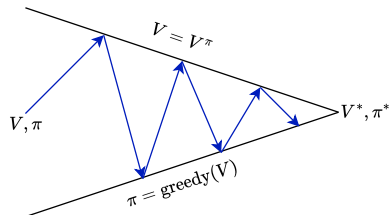
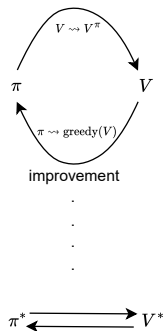
$A \leftarrow$  action given by  $\pi$  for  $S$

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

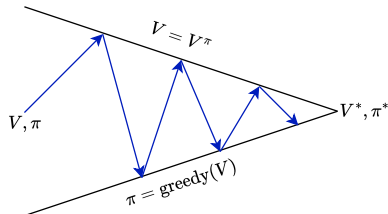
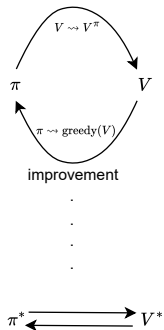
$V(S) \leftarrow V(S) + \alpha [R + \gamma V(S') - V(S)]$

$S \leftarrow S'$

    until  $S$  is terminal



- Policy Evaluation: estimate  $V^\pi$
- Policy Improvement: greedy



- Temporal Difference Policy Evaluation:  $V \approx V^\pi$
- Policy Improvement: greedy?



- ▶ Greedy policy improvement over  $V(s)$  requires a model of the MDP

$$\pi(s) = \arg \max_{a \in \mathcal{A}} \sum_{s'} P(s'|s, a) [r(s, a) + \gamma V(s')]$$

- ▶ Greedy policy improvement over  $Q(s, a)$  is model-free

$$\pi(s) = \arg \max_{a \in \mathcal{A}} Q(s, a)$$

Generalized Policy Iteration with action-value function:

- ▶ Monte Carlo Policy Evaluation:  $Q \approx Q^\pi$
- ▶ Policy Improvement: greedy?



- ▶ We have to ensure that each state-action pair is visited a sufficient (infinite) number of times
- ▶ Simple idea:  $\epsilon$ -greedy
- ▶ All actions have non-zero probability
- ▶ With probability  $\epsilon$  choose a random action, with probability  $1 - \epsilon$  take the greedy action.

$$\pi(a \mid s) = \begin{cases} \frac{\epsilon}{|\mathcal{A}|} + 1 - \epsilon & \text{if } a = \arg \max_{a' \in \mathcal{A}} Q(s, a') \\ \frac{\epsilon}{|\mathcal{A}|} & \text{otherwise} \end{cases}$$



- ▶ We want to learn the optimal policy, but we have to account for the problem of *maintaining exploration*
- ▶ We call the (optimal) policy to be learned the *target policy*  $\pi$  and the policy used to generate behaviour the *behaviour policy*  $b$
- ▶ We say that learning is from data *off* the target policy – thus, those methods are referred to as *off-policy learning*

## Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size  $\alpha \in (0, 1]$ , small  $\varepsilon > 0$

Initialize  $Q(s, a)$ , for all  $s \in \mathcal{S}^+$ ,  $a \in \mathcal{A}(s)$ , arbitrarily except that  $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

    Initialize  $S$

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

    Loop for each step of episode:

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

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

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

$S \leftarrow S'; A \leftarrow A';$

    until  $S$  is terminal

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

Algorithm parameters: step size  $\alpha \in (0, 1]$ , small  $\varepsilon > 0$

Initialize  $Q(s, a)$ , for all  $s \in \mathcal{S}^+$ ,  $a \in \mathcal{A}(s)$ , arbitrarily except that  $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

    Initialize  $S$

    Loop for each step of episode:

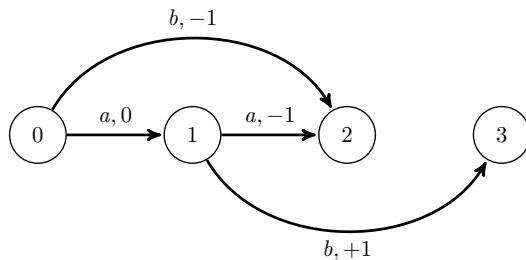
        Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)

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

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

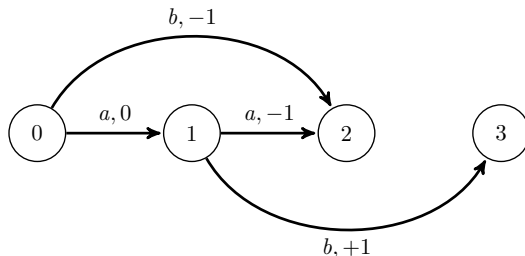
$S \leftarrow S'$

    until  $S$  is terminal



$\text{traj}_1 : 0 \rightarrow 1 \rightarrow 2$   
 $\text{traj}_2 : 0 \rightarrow 1 \rightarrow 3$   
 $\text{traj}_3 : 0 \rightarrow 1 \rightarrow 2$

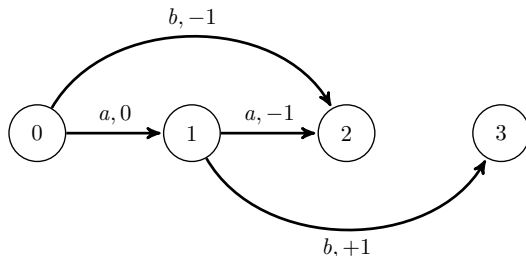
## Example: SARSA vs Q-learning Updates



Episodes (edges: action, cost):  $0 \xrightarrow{a} 1 \xrightarrow{a} 2$ ,  $0 \xrightarrow{a} 1 \xrightarrow{b} 3$ ,  $0 \xrightarrow{a} 1 \xrightarrow{a} 2$ .

**Question:** with  $Q = 0$ ,  $\alpha = 1$ ,  $\gamma = 1$ , what is  $Q(0, a)$  after the three episodes with SARSA and with Q-learning?

## Example: SARSA vs Q-learning Updates



Episodes (edges: action, cost):  $0 \xrightarrow{a} 1 \xrightarrow{a} 2$ ,  $0 \xrightarrow{a} 1 \xrightarrow{b} 3$ ,  $0 \xrightarrow{a} 1 \xrightarrow{a} 2$ .

**Question:** with  $Q = 0$ ,  $\alpha = 1$ ,  $\gamma = 1$ , what is  $Q(0, a)$  after the three episodes with SARSA and with Q-learning?

|                           | episode 1 | episode 2 | episode 3 |
|---------------------------|-----------|-----------|-----------|
| $Q(1, a)$                 | 1         | 1         | 1         |
| $Q(1, b)$                 | 0         | -1        | -1        |
| $Q(0, a)$ with SARSA      | 0         | 0         | 1         |
| $Q(0, a)$ with Q-learning | 0         | 0         | -1        |

## Temporal Difference Methods

- Model-free Learning

- TD Prediction (learning  $V^\pi$  and  $Q^\pi$ )

- TD Control (learning  $\pi^*$ )

## RL with Function Approximation

- Incorporating Function Approximation in RL

- Semi-gradient Methods

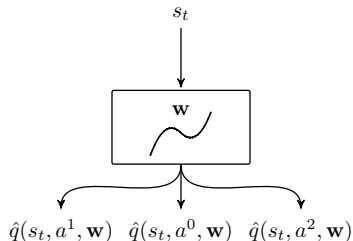
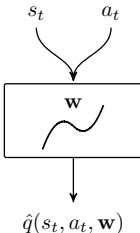
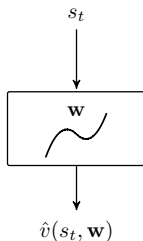
- Deep Q-Networks (DQN)

## Summary



- ▶ Up to this point, we represented all elements of our RL systems by tables (value functions, models and policies)
- ▶ If the state and action spaces are very large or infinite, this is not a feasible solution
- ▶ We can apply function approximation to find a more compact representation of RL components and to generalize over states and actions
- ▶ Reinforcement Learning with function approximation comes with new issues that do not arise in Supervised Learning – such as non-stationarity, bootstrapping and delayed targets

- Here: we estimate value-functions  $V^\pi(\cdot)$  and  $Q^\pi(\cdot, \cdot)$  by function approximators  $\hat{v}(\cdot, \mathbf{w})$  and  $\hat{q}(\cdot, \cdot, \mathbf{w})$ , parameterized by weights  $\mathbf{w}$



- But we can also represent models or policies



We can use different types of function approximators:

- ▶ Linear combinations of features
- ▶ Neural networks
- ▶ Decision trees
- ▶ Gaussian processes
- ▶ Nearest neighbor methods
- ▶ ...

Here: We focus on differentiable FAs and update the weights via gradient descent.



- ▶ Represent state  $s$  by feature vector  $\mathbf{x}(s) = (x^1(s), x^2(s), \dots, x^d(s))^\top$
- ▶ Weight vector  $\mathbf{w} = (w^1, w^2, \dots, w^d)^\top$
- ▶ These features can also be non-linear functions/combinations of state dimensions
- ▶ Linear methods approximate the value function by a linear combination of these features

$$\hat{v}(s, \mathbf{w}) = \mathbf{w}^\top \mathbf{x}(s) = \sum_{i=1}^d w^i x^i(s)$$

- ▶ Therefore,  $\nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w}) = \mathbf{x}(s)$

We want to update our weights w.r.t. the *Mean Squared Value Error* of our prediction:

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

However, we do not have  $V^\pi(S_t)$ .

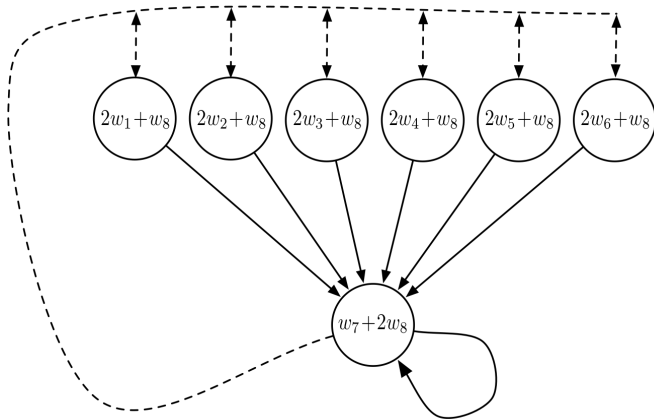
## Gradient MC

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [G_t - \hat{v}(S_t, \mathbf{w})] \nabla \hat{v}(S_t, \mathbf{w})$$

## Semi-gradient TD(0)

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})] \nabla \hat{v}(S_t, \mathbf{w})$$

# Baird's Counterexample



$$\pi(\text{solid}|\cdot) = 1$$

$$b(\text{dashed}|\cdot) = 6/7$$

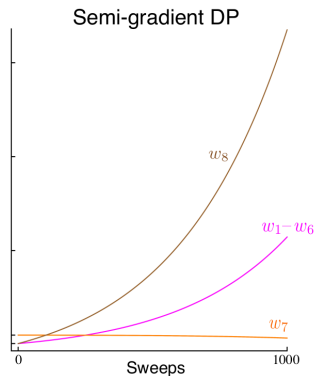
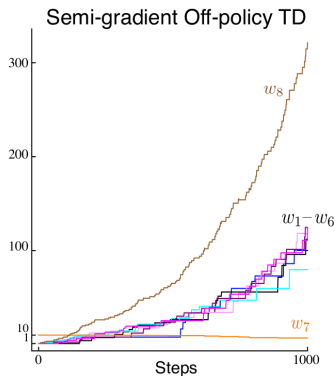
$$b(\text{solid}|\cdot) = 1/7$$

$$\gamma = 0.99$$

The reward is 0 for all transitions, hence  $v_\pi(s) = 0$ . This could be exactly approximated by  $\mathbf{w} = \mathbf{0}$ .

## Semi-gradient DP

$$\mathbf{w} \leftarrow \mathbf{w} + \frac{\alpha}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} (\mathbb{E}[R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) | S_t = s] - \hat{v}(s, \mathbf{w})) \nabla \hat{v}(s, \mathbf{w})$$





The combination of

- ▶ Function Approximation,
- ▶ Bootstrapping and
- ▶ Off-policy Learning

is known as the *Deadly Triad*, since it can lead to stability issues and divergence.



DQN provides a stable solution to deep RL:

- ▶ Use experience replay
- ▶ Sample minibatches (as opposed to full Batches)
- ▶ Freeze target Q-networks
- ▶ Optional: Clip rewards or normalize network adaptively to sensible range



To remove correlations, build data set from agent's own experience

- ▶ Take action  $A_t$  according to  $\epsilon$ -greedy policy
- ▶ Store transition  $(S_t, A_t, R_{t+1}, S_{t+1})$  in replay memory  $D$
- ▶ Sample random mini-batch of transitions  $(S, A, S')$  from  $D$
- ▶ Optimize MSE between Q-network and Q-learning targets, e.g.

$$L(\mathbf{w}) = \mathbb{E}_{(S,A,R,S') \sim D} [(R + \gamma \max_{a'} Q(S', a', \mathbf{w}) - Q(S, A, \mathbf{w}))^2]$$

To avoid oscillations, fix parameters used in Q-learning target

- ▶ Compute Q-learning targets w.r.t. old, fixed parameters  $\mathbf{w}^-$

$$R + \gamma \max_{a'} Q(S', A', \mathbf{w}^-)$$

- ▶ Optimize MSE between Q-network and target network

$$L(\mathbf{w}) = \mathbb{E}_{(S, A, R, S') \sim D} [(R + \gamma \max_{a'} Q(S', a', \mathbf{w}^-) - Q(S, A, \mathbf{w}))^2]$$

- ▶ Periodically update fixed parameters  $\mathbf{w}^- \leftarrow \mathbf{w}$ 
  - ▶ hard update: update target network every  $N$  steps
  - ▶ slow update: slowly update weights of target network every step by

$$\mathbf{w}^- \leftarrow (1 - \tau)\mathbf{w}^- + \tau\mathbf{w}$$

Initialize replay memory  $D$  to capacity  $N$  and Q-network weights  $\mathbf{w}$  with  $\mathbf{w}^- = \mathbf{w}$

**for** episode  $i = 1, \dots, M$  **do**

**for**  $t = 1, \dots, T$  **do**

        Select action  $A_t$   $\epsilon$ -greedily

        Store transition  $(S_t, A_t, S_{t+1}, R_t)$  in  $D$

        Sample minibatch of transitions  $(S_j, A_j, S'_j, R_j)$  from  $D$  with batchsize  $B$

        Compute target  $y_j$  for each sample  $j = 1, \dots, B$ :

$$Y_j = \begin{cases} R_j & \text{if } S'_j \text{ is terminal} \\ R_j + \gamma \max_{a'} Q(S'_j, a', \mathbf{w}^-) & \text{else} \end{cases}$$

        Update the parameters of Q according to:

$$\nabla_{\mathbf{w}} \hat{L}(\mathbf{w}) = \frac{\alpha}{B} \sum_{j=1}^B \left( Y_j - Q(S_j, A_j, \mathbf{w}) \right) \nabla_{\mathbf{w}} Q(S_j, A_j, \mathbf{w})$$

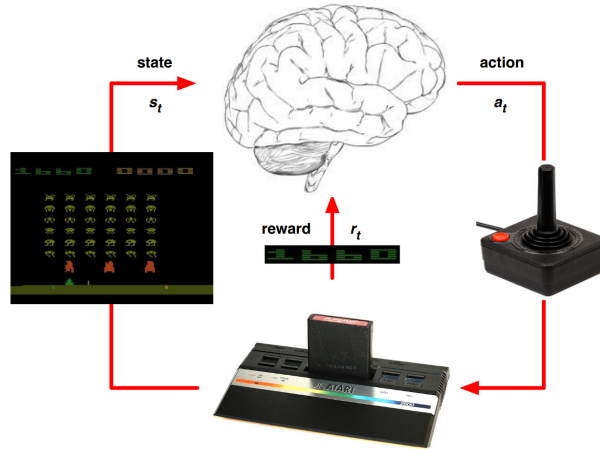
        Update target network with hard or slow update

**end**

**end**

This is your  
exercise

# Deep Q-Networks: Reinforcement Learning in Atari





- ▶ End-to-end learning of values  $Q(s, a)$  from pixels  $s$
- ▶ Input state  $s$  is a stack of raw pixels from the last 4 frames
- ▶ Output is  $Q(s, a)$  for 18 joystick/button positions
- ▶ Reward is change in score for that step

# How much does DQN help?



|                | Q-Learning | Q-Learning<br>+ Target Q | Q-Learning<br>+ Replay | DQN<br>Q-learning<br>+ Replay<br>+ Target Q |
|----------------|------------|--------------------------|------------------------|---------------------------------------------|
| Breakout       | 3          | 10                       | 241                    | <b>317</b>                                  |
| Enduro         | 29         | 142                      | 831                    | <b>1006</b>                                 |
| River Raid     | 1453       | 2868                     | 4103                   | <b>7447</b>                                 |
| Seaquest       | 276        | 1003                     | 831                    | <b>2894</b>                                 |
| Space Invaders | 302        | 373                      | 826                    | <b>1089</b>                                 |



- ▶ **Model-free learning:** Temporal Difference (TD) methods estimate value functions from experience, using bootstrapping to learn incrementally without a model of the environment.
- ▶ **Policy improvement and control:** TD control methods (e.g., Q-learning) enable learning optimal policies, often combined with exploration strategies like  $\epsilon$ -greedy and off-policy updates.
- ▶ **Function approximation and deep RL:** Large or continuous state spaces require function approximators (linear, neural networks), with techniques like DQN using experience replay and target networks for stability.