Active Reinforcement Learning

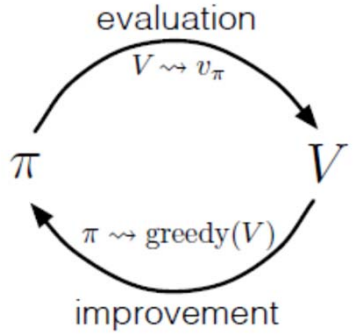
- So far, we have assumed agent with a policy
  - We try to learn how good it is
- Now, suppose agent must learn a good policy (optimal)
  - While acting in uncertain world

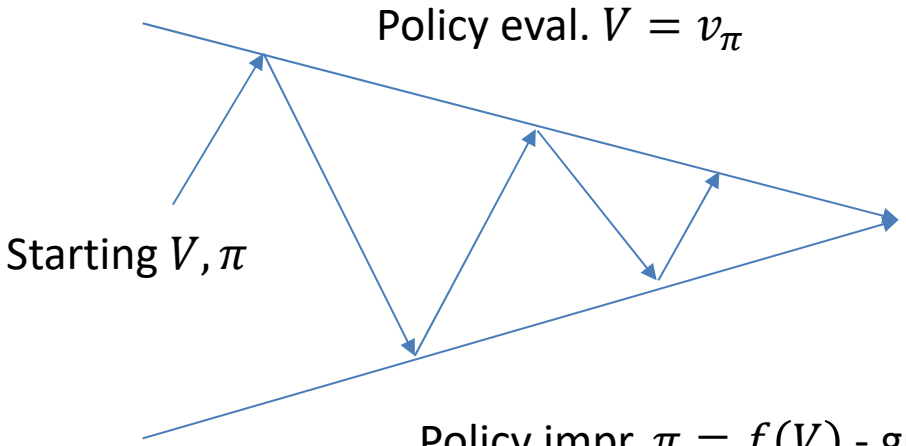
Active Reinforcement Learning

- On-Policy
  - Learn while following a policy
- Off-Policy
  - Learn outside the policy

Policy Iteration

- Policy Evaluation
  - Estimate the value function
- Policy Improvement
  - Find a way to improve the policy





$\pi_* \rightleftharpoons v_*$

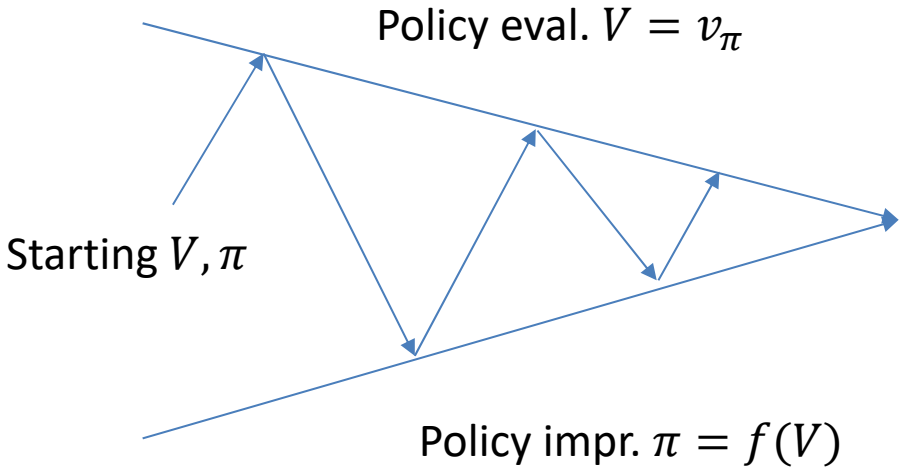
V^*, π^*

Policy impr. $\pi = f(V)$ - greedy function in policy iteration

Policy Iteration

- Policy Evaluation
  - Estimate the value function: MC-Evaluation?
- Policy Improvement
  - Find a way to improve the policy?

Model free!
How can we improve policy?



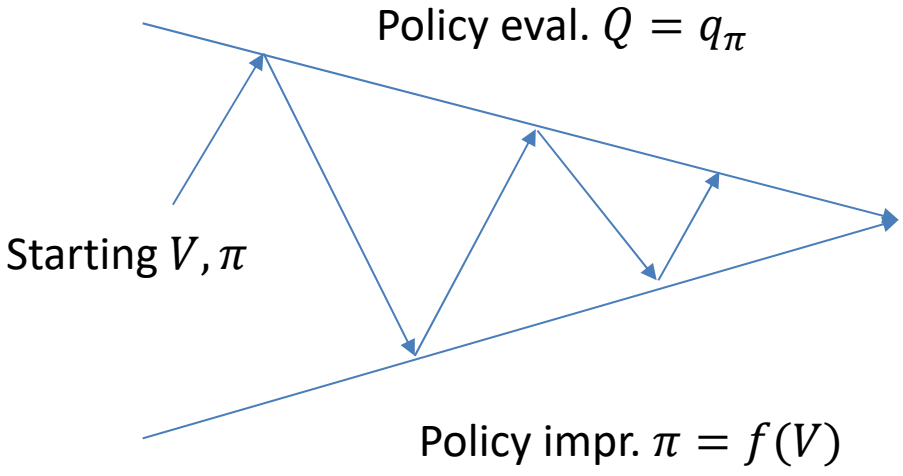
$$\pi'(s) = argmax_{a \in A} R_s^a + P_{ss'}^a V(s')$$

V^*, π^*

Policy Iteration

- Policy Evaluation
  - Estimate the value function: MC-Evaluation?
- Policy Improvement
  - Find a way to improve the policy?

Model free!
Better to use $Q(s,a)$ …



$\pi'(s) = argmax_{a \in A}\, Q\,(s,a)$

Q^*, π^*

… but is the greedy improvement a good choice?

Multi-Armed Bandit Problem

- Action selection:
  - Decide which machines to play

Greedy selection strategy?

1. $R(left) = 0$
2. $R(right) = 1$
3. $R(right) = 2$
4. $R(right) = 1$

We keep going with right, but left was not explored enough

Which is the best strategy?
- We need to explore and exploit, balancing exploration and exploitation

This is a typical RL problem

Exploration versus Exploitation

- Two reasons to take an action in RL
  - **Exploitation**: To try to get reward. We exploit our current knowledge to get a payoff.
  - **Exploration**: Get more information about the world. How do we know if there is not a pot of gold around the corner.

- To explore we typically need to take actions that do not seem best according to our current model.

- Managing the trade-off between exploration and exploitation is a critical issue in RL

- Basic intuition behind most approaches:
  - Explore more when knowledge is weak
  - Exploit more as we gain knowledge

Exploration

- Simple exploration strategy: ϵ-Greedy
  - With prob ϵ select a random action
  - With prob $1-\epsilon$ select the greedy action

$$\pi(a \mid s) = \begin{cases} \frac{\epsilon}{m} + 1 - \epsilon & \text{If } a = argmax_{a \in A} Q(s,a) \\ \frac{\epsilon}{m} & \text{otherwise} \end{cases}$$

Theorem

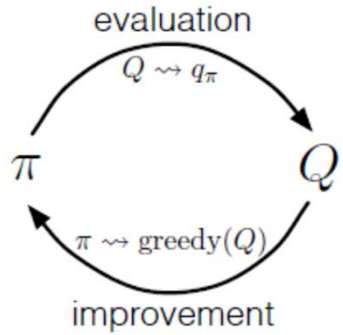
For any ϵ-Greedy policy π, the ϵ-Greedy policy π' with respect to q_π is an improvement, i.e. $v_{\pi'}(s) \geq v_\pi(s)$

$$q_\pi\big(s, \pi'(s)\big) = \sum_{a \in A} \pi'(a|s) q_\pi(s,a) = \frac{\epsilon}{m} \sum_{a \in A} q_\pi(s,a) + (1-\epsilon) \max_{a \in A} q_\pi(s,a) \geq$$

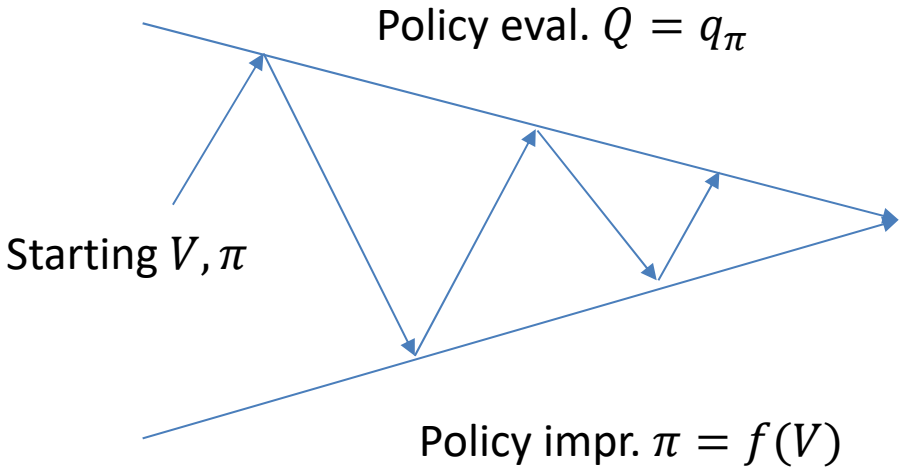
$$\sum_{a \in A} q_\pi(s,a) + (1-\epsilon) \sum_{a \in A} q_\pi(s,a) \frac{\left(\pi(a|s) - \frac{\epsilon}{m}\right)}{1-\epsilon} = v_\pi(s)$$

Policy Iteration

- Policy Evaluation
  - Estimate the value function: MC-Evaluation?
- Policy Improvement
  - Find a way to improve the policy?



Model free!
Better to use $Q(s,a)$ …

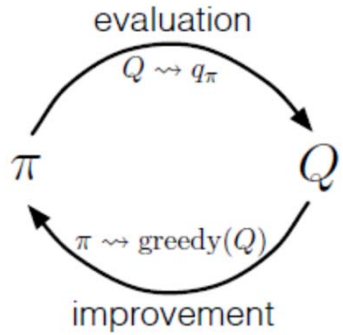


$\pi'(s) = argmax_{a \in A} Q\ (s,a)$

Q^*, π^*

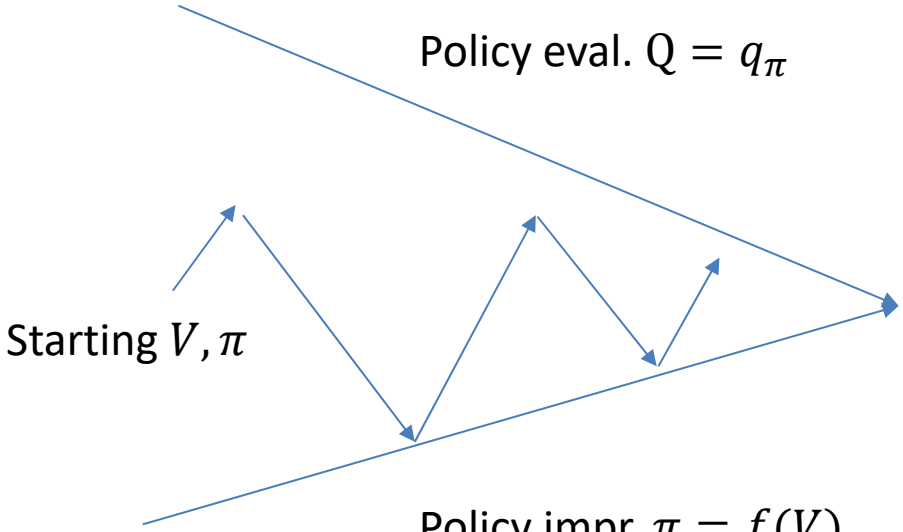
Policy Iteration

- Policy Evaluation
  - Estimate the value function: MC-Evaluation?
- Policy Improvement
  - Find a way to improve the policy?



Model free!
Better to use $Q(s,a)$ …

$$\pi'(s) = argmax_{a \in A}\, Q\,(s,a)$$



Q^*, π^*

Policy evaluation can be less frequent

GLIE Exploration

- Exploration policy greedy in the limit of infinite exploration (GLIE) satisfies the following two properties:
  - 1. If a state is visited infinitely often, then each action in that state is chosen infinitely often (with probability 1).

$$\lim_{t\to\infty} N_t(s,a) = \infty$$

  - 2. In the limit (as t → ∞), the learning policy is greedy with respect to the learned Q-function (with probability 1).

$$\lim_{t\to\infty} \pi_t(a \mid s) = 1 \; (i.e., a = argmax_{a' \in A} Q(s,a'))$$

  - For instance, ϵ-Greedy with $\epsilon_t = \frac{1}{t}$

GLIE MC

- Learning with MC
  - Sample k-th episode from π: $\{S_1, A_1, R_2, \dots, S_T\}$
  - For each state and action in the episode:

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G_t - Q(S_t, A_t))$$

  - Improve the policy with ϵ-Greedy: $\epsilon_k = \frac{1}{k}$

$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

Theorem

The MC-GLIE converges towards the optimal action-value function

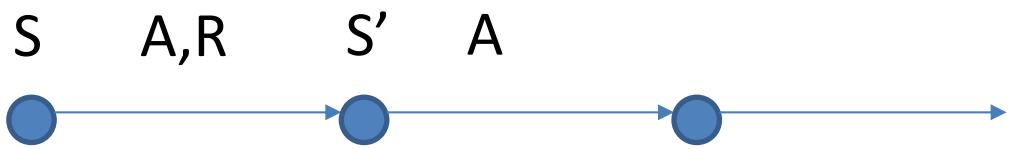
MC vs TD Learning

- TD Learning advantages:
  - On-line learning (no termination)
  - Incomplete sequences
  - Exploits Bellman
  - Low variance

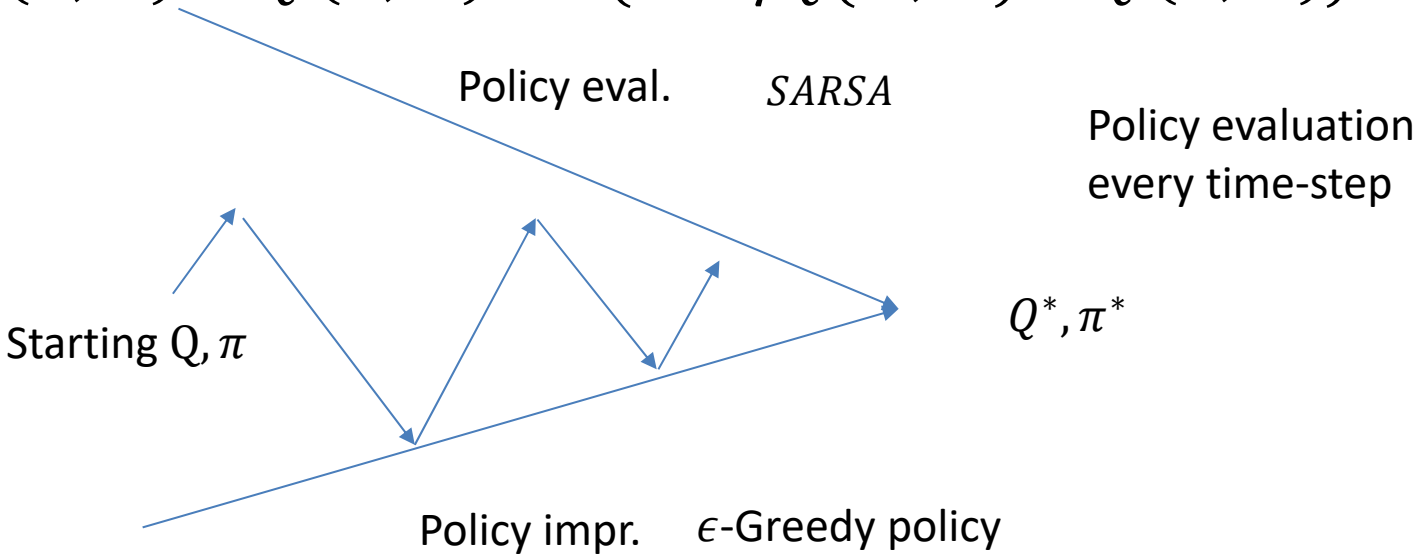
- Use TD Learning instead of MC Learning
  - Evaluation of $Q(s, a)$
  - Policy improvement with ϵ-greedy
  - Update every step (not after each episode)

TD Learning

- Use TD Learning instead of MC Learning
  - Evaluation of $Q(s, a)$
  - SARSA action-value update



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



TD-Sarsa

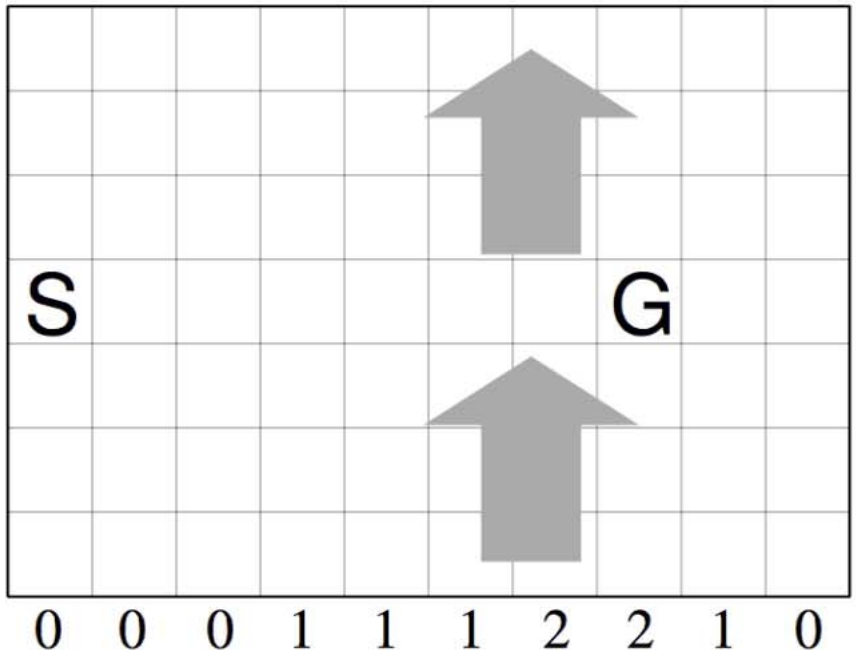
Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $Q(s, a) \leftarrow Q(s, a) + \alpha\left[r + \gamma Q(s', a') - Q(s, a)\right]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

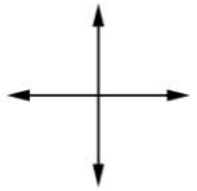
Q(s,a) is usually represented as a look-up table

Theorem
The TD-GLIE converges towards the optimal action-value function under the following conditions:

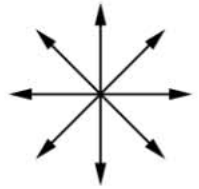
- GLIE sequence of policies $\pi_t(a \mid s)$
- Robbins-Monro sequence of step α_t: $\lim_{t\to\infty} \sum_{t\in T} \alpha_t = \infty, \sum_{t\in T} \alpha_t^2 < \infty$

Windy Gridworld Example





standard moves

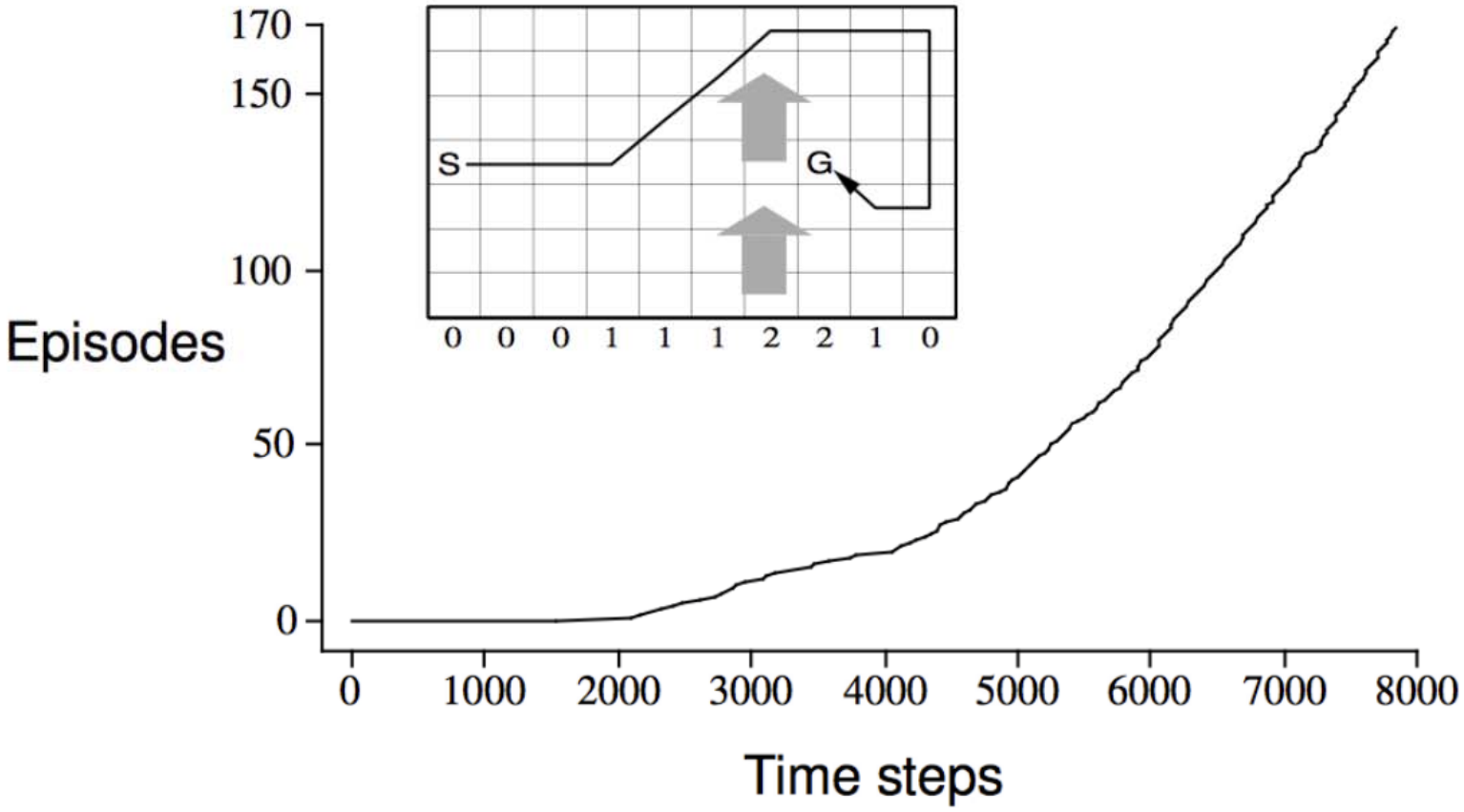


king's moves

undiscounted, episodic, reward = –1 until goal

Windy Gridworld Example

ϵ-greedy Sarsa with $\epsilon = 0.1$ and $\alpha = 0.5$, init values $Q(s, a) = 0$



Completed episodes vs time steps

n-step Sarsa

n-step version of SARSA

$n = 1, \quad q_t^1 = R_{t+1} + \gamma Q(S_{t+1})$ SARSA

$n = 2, \quad q_t^2 = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+1})$

…

$n = \infty, \quad q_t^\infty = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^T R_T$ MC

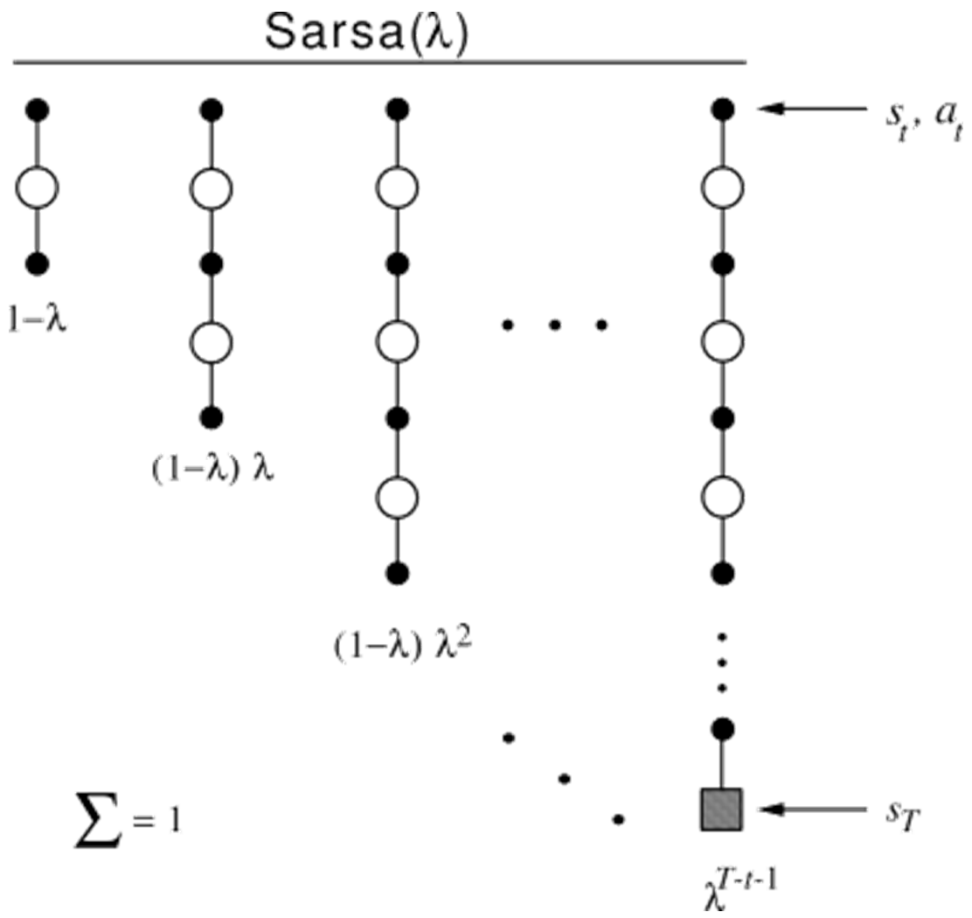
n-step Q-return

$$q_t^n = R_{t+1} + \gamma R_{t+2} + \gamma^{n-1} Q(S_{t+n})$$

n-step SARSA update towards the n-step Q-return

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(q_t^n - Q(S_t, A_t))$$

Forward View Sarsa λ



$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(q_t^\lambda - Q(S_t, A_t))$$

$$q_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} q_t^n$$

Backward View Sarsa λ

Eligibility trace in an online algorithm

Sarsa(λ) has one eligibility trace for each state-action pair

$$E_0(s,a) = 0$$
$$E_t(s,a) = \gamma\lambda E_{t-1}(s,a) + \mathbf{1}(S_t = s, A_t = a)$$

$Q(s,a)$ updated for every state and action:

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha\delta_t E_t(s,a)$$

Backward View Sarsa λ

Eligibility trace in an online algorithm

Sarsa(λ) has one eligibility trace for each state-action pair

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat (for each episode):
  $E(s,a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
  Initialize S, A
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$
    $E(S, A) \leftarrow E(S, A) + 1$
    For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
      $Q(s,a) \leftarrow Q(s,a) + \alpha \delta E(s,a)$
      $E(s,a) \leftarrow \gamma \lambda E(s,a)$
    $S \leftarrow S'$; $A \leftarrow A'$
  until S is terminal

Equivalent to forward view

Off-Policy Learning

Evaluate target policy $\pi(a \mid s)$ to compute $v_\pi(s)$ or $q_\pi(s, a)$ while following another policy $\mu(a \mid s)$

- Learn from observing humans or other agents
- Learn from past policies, re-use experience form old policies
- Learn the *optimal* policy while following an *exploration* policy
- Learn multiple policies while following one policy

Off-Policy Learning

Evaluate target policy $\pi(a \mid s)$ to compute $v_\pi(s)$ or $q_\pi(s, a)$ while following another policy $\mu(a \mid s)$

- Importance sampling
- Q-learning

Off-Policy Learning

Evaluate target policy $\pi(a \mid s)$ to compute $v_\pi(s)$ or $q_\pi(s, a)$ while following another policy $\mu(a \mid s)$

- Importance sampling

MC Off-policy with importance sampling

- Importance along the whole episode

$$G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)\pi(A_{t+1}|S_{t+1}) \dots \pi(A_T|S_T)}{\mu(A_t|S_t)\mu(A_{t+1}|S_{t+1}) \dots \mu(A_T|S_T)} G_t$$

- Update towards the correct return

$$V(S_t) \to V(S_t) + \alpha(G_t^{\pi/\mu} - V(S_t))$$

- Not practical, too high variance

Off-Policy Learning

Evaluate target policy $\pi(a \mid s)$ to compute $v_\pi(s)$ or $q_\pi(s, a)$ while following another policy $\mu(a \mid s)$

- Importance sampling

TD Off-policy with importance sampling

- Importance sampling correction at each step

$$V(S_t) \rightarrow V(S_t) + \alpha\left(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}(R_{t+1} + \gamma V(S_{t+1})) - V(S_t)\right)$$

Off-Policy Learning

Evaluate target policy $\pi(a \mid s)$ to compute $v_\pi(s)$ or $q_\pi(s, a)$ while following another policy $\mu(a \mid s)$

- **Q-Learning approach** [Watkins, 1989]
  - No importance sampling
  - Next action using the behavior policy μ
  - Assess alternative successor action with policy π
  - Update $Q(S_t, A_t)$ considering the alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t))$$

Q-Learning

Evaluate target policy $\pi(a \mid s)$ to compute $v_\pi(s)$ or $q_\pi(s,a)$ while following another policy $\mu(a \mid s)$

- The target policy π is greedy with respect to $Q(s,a)$

$$\pi(S_{t+1}) = argmax_{a'} Q(S_{t+1}, a')$$

- The behavior policy μ is ϵ –greedy with respect to $Q(s,a)$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, argmax_{a'} Q(S_{t+1}, a')) - Q(S_t, A_t))$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a') - Q(S_t, A_t))$$

Theorem

The Q-Learning converges towards the optimal action-value function with GLIE and

$$\lim_{T\to\infty} \sum_{t=1}^{T} \alpha_t = \infty \quad \text{and} \quad \lim_{T\to\infty} \sum_{t=1}^{T} \alpha_t^2 < \infty$$

Q-Learning

1. Start with initial Q-function (e.g. all zeros)
2. Take an action according to an explore/exploit policy (should converge to greedy policy, i.e. GLIE)
3. Perform TD update

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a') - Q(S_t, A_t))$$

   Q(s,a) is current estimate of optimal Q-function.

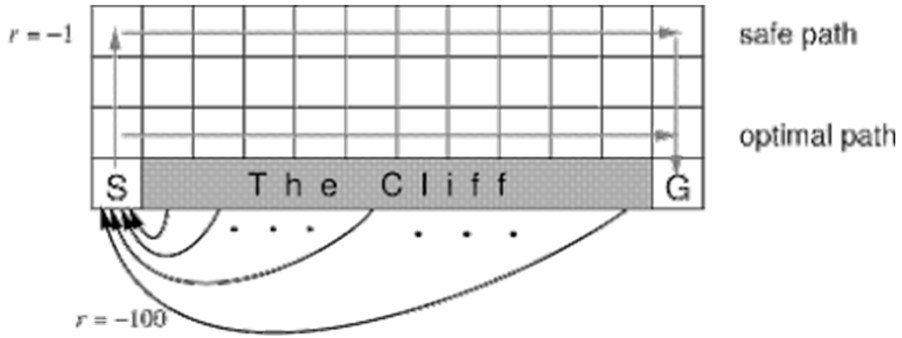
4. Goto 2

- Does not require model since we learn Q directly
- Uses explicit |S|x|A| table to represent Q
- Explore/exploit policy directly uses Q-values

SARSA vs Q-Learning

Cliff Walking (undiscounted, episodic task)

- ϵ-greedy policy with $\epsilon = 0.1$
- Q-learning off-policy, but more risky



Explore/Exploit Policies

- Boltzmann Exploration policy
  - Select action a with probability,

$$\Pr(a \mid s) = \frac{\exp\left(Q(s,a)/T\right)}{\sum_{a' \in A} \exp\left(Q(s,a')/T\right)}$$

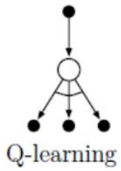
  - T is the temperature. Large T means that each action has about the same probability. Small T leads to more greedy behavior.
  - Typically start with large T and decrease with time

Expected Sarsa

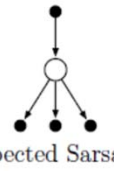
Analogous to Q-Learning, but update with respect to the expected value instead of the maximum over next state actions

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

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

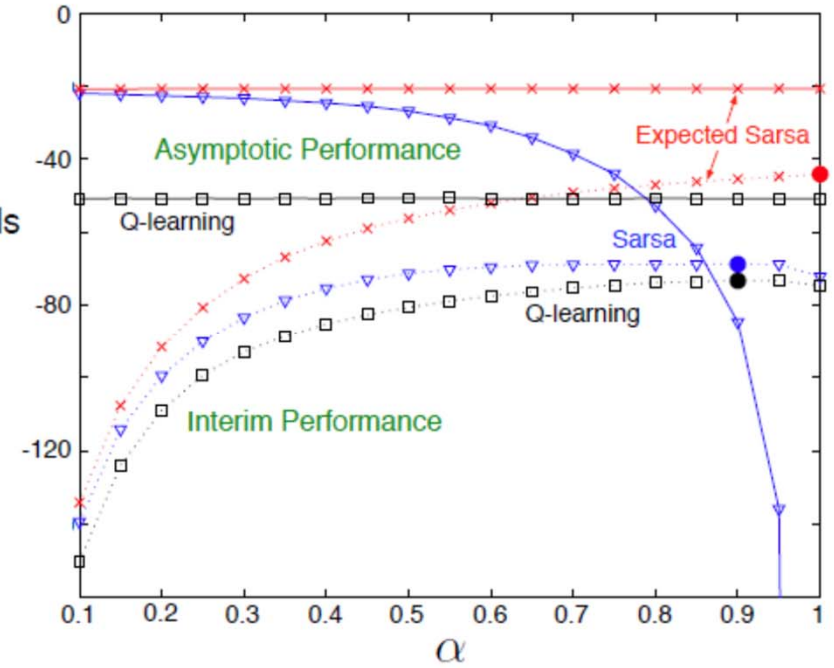


Q-learning



Expected Sarsa

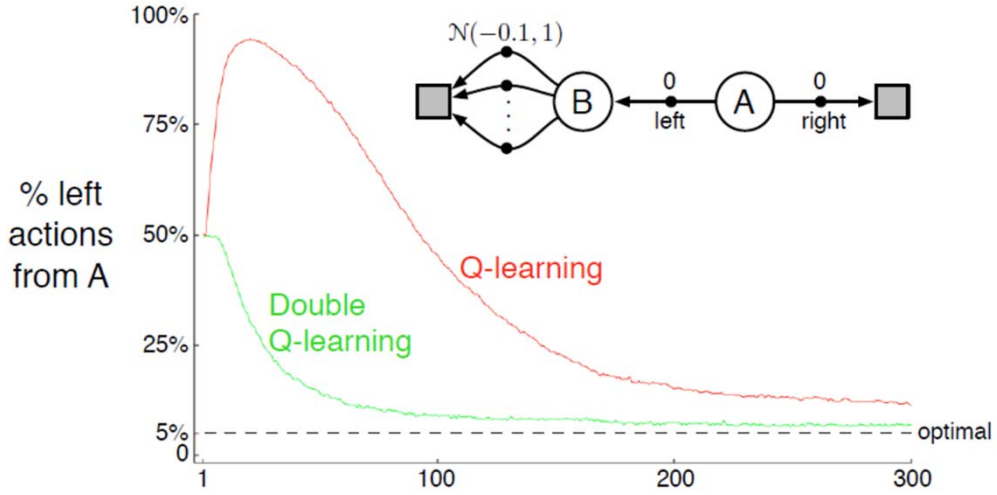
Sum of rewards per episode



Cliff Walk
ϵ-greedy policy with $\epsilon = 0.1$
asymptotic aver over 100000 eps, interim over first 100 [van Seijen et al. 2009]

Double Learning

Maximization bias: a maximum over estimated values is used as an estimate of the maximum value, which can lead to a positive bias



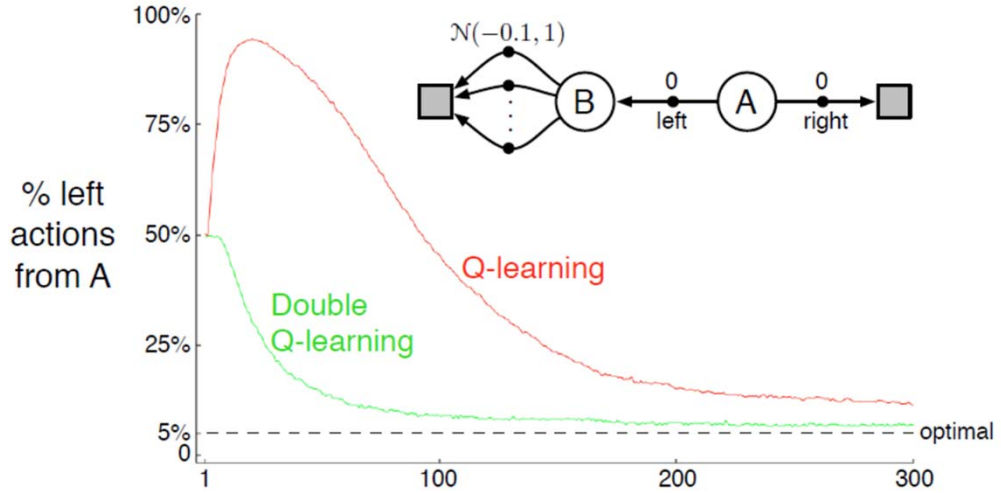
ϵ-greedy policy with $\epsilon = 0.1$, with right reward is 0, with left rewards mean is -0.1, but left may be preferred during the learning process

The same samples are used both to determine the maximizing action and to estimate its value. Divide the plays in two sets and use them to learn two independent estimates

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha(R_{t+1} + \gamma Q_2\big(S_{t+1}, argmax_{a'} Q_1(S_{t+1}, a')\big) - Q_1(S_t, A_t))$$

Double Learning

Maximization bias: a maximum over estimated values is used as an estimate of the maximum value, which can lead to a positive bias



Initialize $Q_1(s,a)$ and $Q_2(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily
Initialize $Q_1(\textit{terminal-state},\cdot) = Q_2(\textit{terminal-state},\cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q_1 and Q_2 (e.g., ε-greedy in $Q_1 + Q_2$)
    Take action A, observe R, S'
    With 0.5 probabilility:

$$Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \operatorname{argmax}_a Q_1(S',a)\big) - Q_1(S,A)\Big)$$

    else:

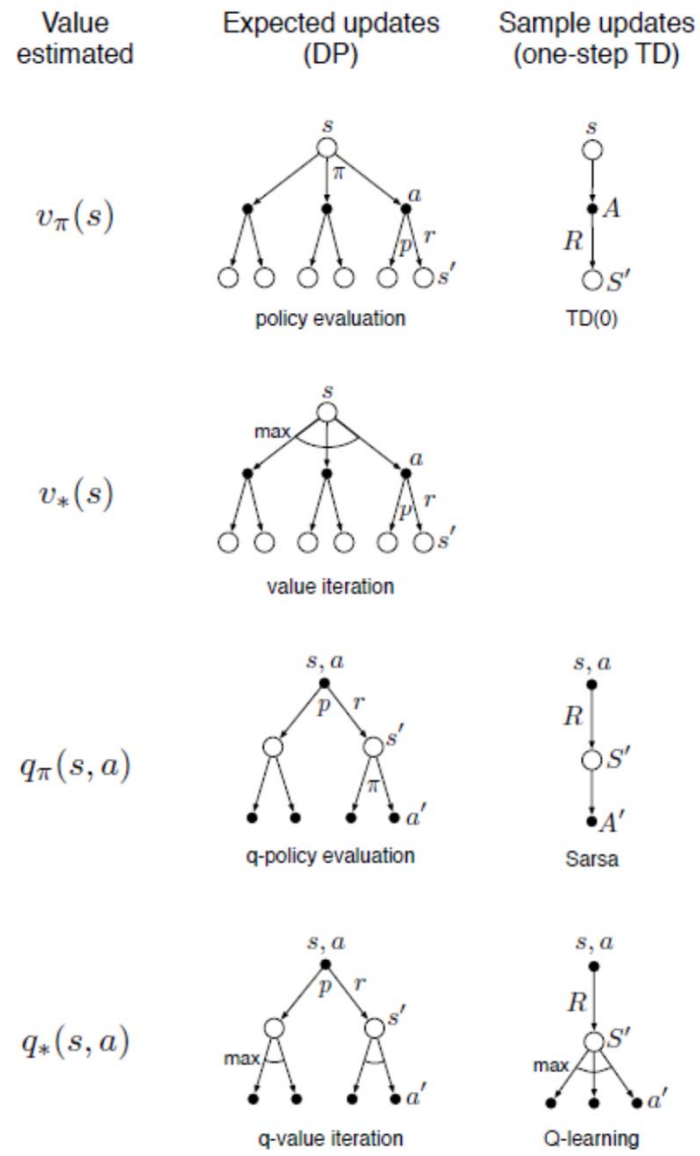
$$Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \operatorname{argmax}_a Q_2(S',a)\big) - Q_2(S,A)\Big)$$

    $S \leftarrow S'$
  until S is terminal

DP vs TD

Relationship between DP and TD

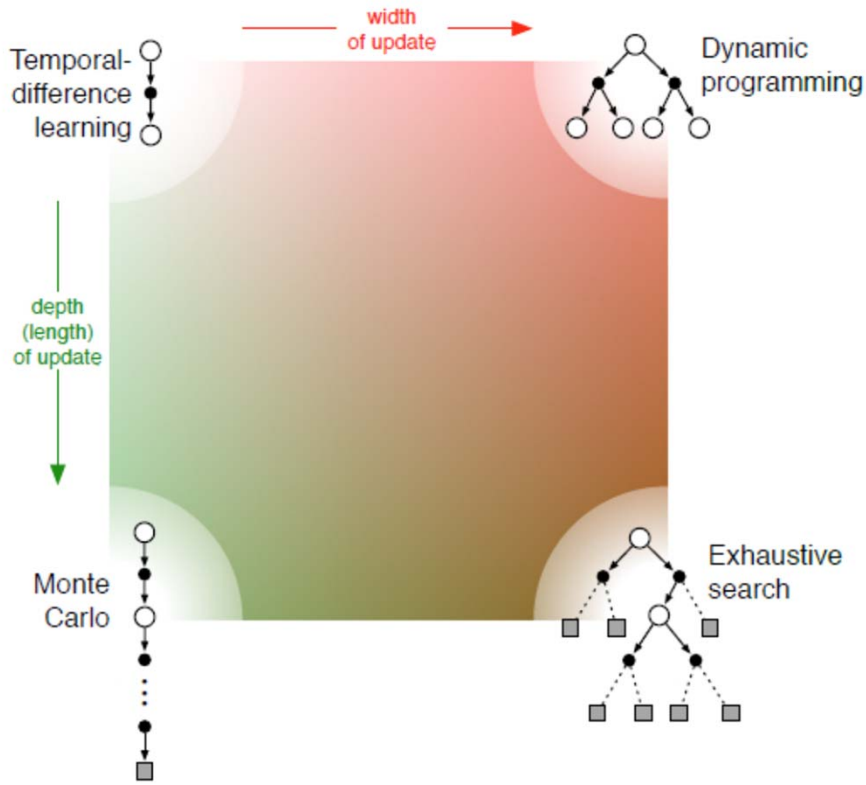
- Belman Expectation for $v_\pi(s)$
  - Iterative Policy Evaluation (DP)
  - TD Learning (Sampling)
- Belman Expectation for $q_\pi(s,a)$
  - Q-Policy Iteration (DP)
  - Sarsa (Sampling)
- Belman Optimality for $q^*(s,a)$
  - Q-Value Iteration (DP)
  - Q-Learning (Sampling)



DP vs TD

Relationship between DP and TD

- Belman Expectation for $v_\pi(s)$
  - Iterative Policy Evaluation (DP)
  - TD Learning (Sampling)
- Belman Expectation for $q_\pi(s,a)$
  - Q-Policy Iteration (DP)
  - Sarsa (Sampling)
- Belman Optimality for $q^*(s,a)$
  - Q-Value Iteration (DP)
  - Q-Learning (Sampling)



Large Scale RL

Large problems

- Backgammon: 10^{20}stati
- Go: 10^{170}stati
- Helicopter: continuous

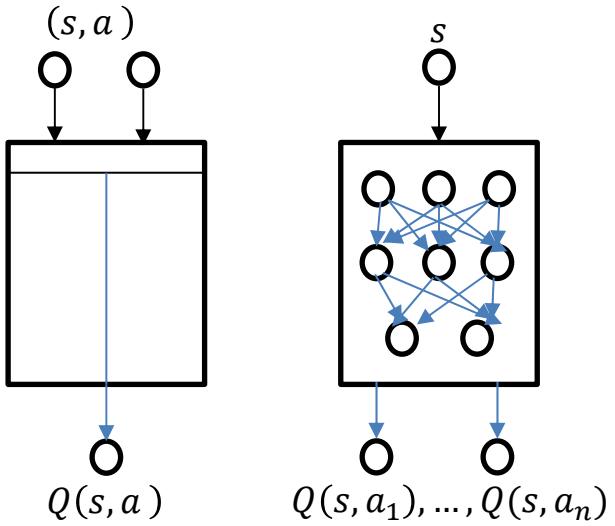
Efficient approximation of value functions and policies

Estimates with a value function approximation:

- $\hat{v}(s, w) \approx v_\pi(s)$
- $\hat{q}(s, a, w) \approx q_\pi(s, a)$
- $\dim(w) \ll |S|$

Differentiable function approximations:

- Linear combination of features
- Neural Networks
- …



- Non-stationarity target f, not indep. identically distributed data, on-line, non-supervised learning
- More complex RL, many state affected by parameter changes
- but effective, may also work with partially observable problems

Gradient Descent

Let $J(w)$ be a differentiable function of parameter vector w

The gradient of $J(w)$ is

$$\nabla_{\mathrm{w}} J(w) = \begin{pmatrix} \frac{\partial J(w)}{\partial w_1} \\ \cdots \\ \frac{\partial J(w)}{\partial w_n} \end{pmatrix}$$

- To find the local minimum of $J(w)$ adjust the parameters in the direction of the gradient vector

$$\Delta w = -\frac{1}{2}\alpha \nabla_w J(w)$$

- Find vector that minimize the mean-squared error between $\hat{v}(s,w)$ and $v_\pi(s)$

$$\mathrm{J}(w) = E_\pi[\big(v_\pi(s) - \hat{v}(s,w)\big)^2]$$

- Stochastic gradient descent (SGD) samples the gradient (update on samples)

$$\Delta w = \alpha\big(v_\pi(s) - \hat{v}(s,w)\big)\nabla_{\mathrm{w}}\, \hat{v}(s,w) \qquad \mathrm{w}_{\mathrm{t+1}} = \mathrm{w}_{\mathrm{t}} - \Delta w$$

Incremental Prediction

The value function $v_\pi(s)$ is not available

- Substitute $v_\pi(s)$ with an estimate
- For MC learning

$$\Delta w = \alpha\big(G_t - \hat{v}(S_t, w)\big)\nabla_{\mathrm{w}}\, \hat{v}(S_t, w)$$

- For TD(0) learning

$$\Delta w = \alpha\big(R_{t+1} + \gamma\hat{v}(S_{t+1}, w) - \hat{v}(S_t, w)\big)\nabla_{\mathrm{w}}\, \hat{v}(S_t, w)$$

- In MC is like generating “training data”, converges to the local optimum
- In TD is analogous and converges (close) to the local optimum

Incremental Prediction

The value function $v_\pi(s)$ is not available

- Substitute $v_\pi(s)$ with an estimate
- For MC learning

$$\Delta w = \alpha\big(G_t - \hat{v}(S_t, w)\big)\nabla_{\mathrm{w}}\, \hat{v}(S_t, w)$$

Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$

Initialize value-function weights $\mathbf{w}$ as appropriate (e.g., $\mathbf{w} = \mathbf{0}$)
Repeat forever:
  Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
  For $t = 0, 1, \ldots, T-1$:
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[G_t - \hat{v}(S_t, \mathbf{w})\big]\nabla \hat{v}(S_t, \mathbf{w})$

Incremental Prediction

The value function $v_\pi(s)$ is not available

- Substitute $v_\pi(s)$ with an estimate

- For TD(0) learning (semi-gradient method)

$$\Delta w = \alpha\big(R_{t+1} + \gamma\hat{v}(S_{t+1}, w) - \hat{v}(S_t, w)\big)\nabla_{\mathrm{w}}\,\hat{v}(S_t, w)$$

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

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$

Initialize value-function weights $\mathbf{w}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose $A \sim \pi(\cdot|S)$
    Take action A, observe R, S'
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
    $S \leftarrow S'$
  until S' is terminal

Linear Case: Feature Vectors

We can represent the state as a feature vector

$$\boldsymbol{x}(S) = \begin{pmatrix} x_1(S) \\ \dots \\ x_n(S) \end{pmatrix}$$

- Value function as a linear combination of features

$$\hat{v}(s, w) = \boldsymbol{x}(S)^T w = \sum_j x_j(S) w_j$$

- Objective function is quadratic in parameters

$$\mathrm{J}(w) = E_\pi[(v_\pi(s) - X(S)^T w)^2]$$

- Gradient descent converges on global optimum
- Update rule is simple

$$\nabla_w \hat{v}(s, w) = \boldsymbol{x}(S)$$

$$\Delta w = \alpha\big(v_\pi(s) - \hat{v}(s, w)\big)\boldsymbol{x}(S)$$

Table Lookup Features

Table lookup is a special case of linear value function approximation

- Table lookup features

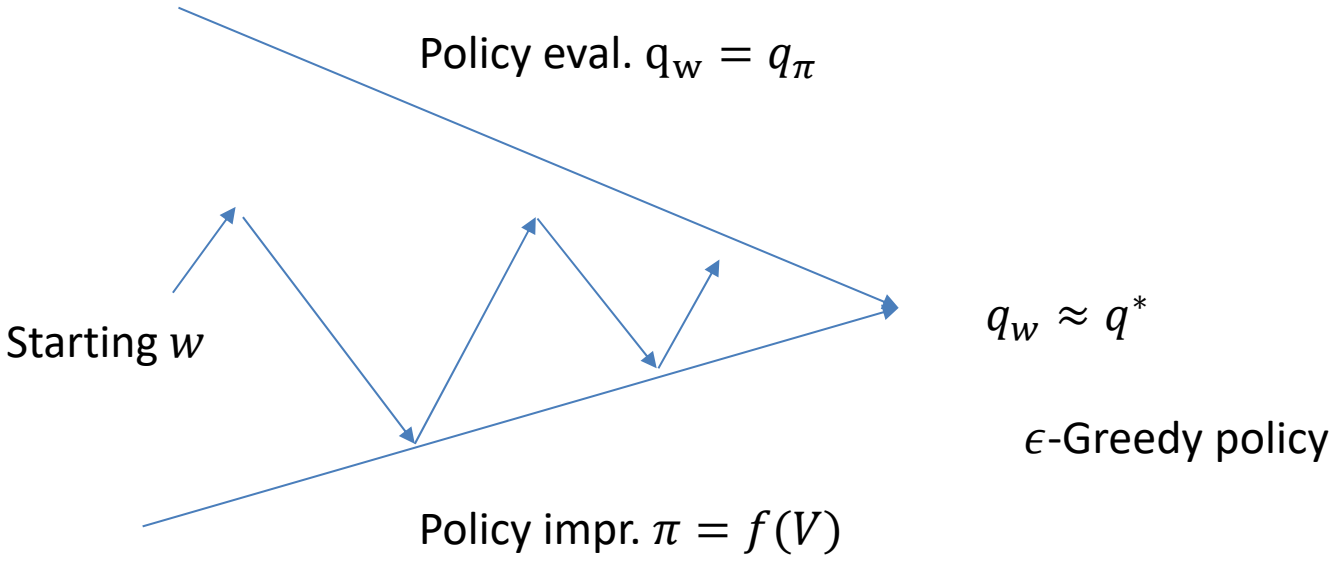
$$\boldsymbol{x}^{table}(S) = \begin{pmatrix} 1\,(S = s_1) \\ \dots \\ 1\,(S = s_n) \end{pmatrix}$$

- Parameter vector gives value of each individual state

$$\hat{v}(s, w) = \boldsymbol{x}^{table}(S)^T w = \sum_j x_j(S) w_j$$

Control with Function Approximation

- Policy Evaluation
  - Approximate $\hat{q}(s, a, w) \approx q_\pi(s, a)$
- Policy Improvement
  - ϵ-greedy



Action-Value Function Approx

Approximate the action-value function

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

- Minimize the mean-squared error

$$\mathrm{J}(w) = E_\pi[\big(q_\pi(s, a) - \hat{q}(s, a, w)\big)^2]$$

- Stochastic gradient descent samples the gradient

$$\Delta w = \alpha\big(q_\pi(s, a) - \hat{q}(s, a, w)\big)\nabla_{\mathrm{w}}\hat{q}(s, a, w)$$

Action-Value Feature Vectors

We can represent the state as a feature vector

$$\boldsymbol{x}(S,A) = \begin{pmatrix} x_1(S,A) \\ \dots \\ x_n(S,A) \end{pmatrix}$$

- Value function as a linear combination of features

$$\hat{q}(s,a,w) = \boldsymbol{x}(S,A)^T w = \sum_j x_j(S,A) w_j$$

- Objective function is quadratic in parameters

$$\mathrm{J}(w) = E_\pi[(q_\pi(s,a) - X(S,A)^T w)^2]$$

- Gradient descent converges on global optimum
- Update rule is simple

$$\nabla_w \hat{q}(S,A,w) = \boldsymbol{x}(S,A)$$

$$\Delta w = \alpha\big(q_\pi(s,a) - \hat{q}(s,a,w)\big)\boldsymbol{x}(S,A)$$

Incremental Control

- Substitute $q_\pi(s)$ with an estimate

- For MC learning

$$\Delta w = \alpha\big(G_t - \hat{q}(S_t, A_t, w)\big)\nabla_{\mathrm{w}}\, \hat{q}(S_t, A_t, w)$$

- For TD(0) learning

$$\Delta w = \alpha\big(R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, w) - \hat{q}(S_t, A_t, w)\big)\nabla_{\mathrm{w}}\, \hat{q}(S_t, A_t, w)$$

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

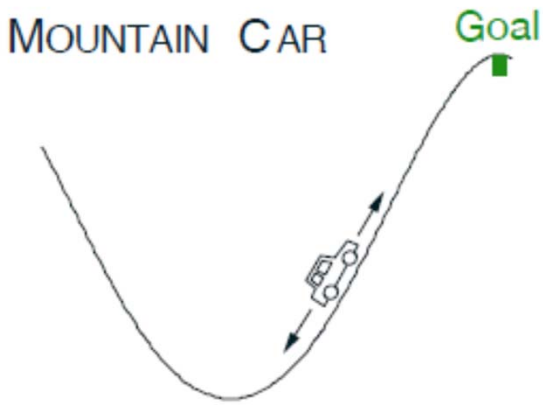
Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$

Initialize value-function weights $\mathrm{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathrm{w} = \mathbf{0}$)
Repeat (for each episode):
  $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    If S' is terminal:
      $\mathrm{w} \leftarrow \mathrm{w} + \alpha\big[R - \hat{q}(S, A, \mathrm{w})\big]\nabla\hat{q}(S, A, \mathrm{w})$
      Go to next episode
    Choose A' as a function of $\hat{q}(S', \cdot, \mathrm{w})$ (e.g., ε-greedy)
    $\mathrm{w} \leftarrow \mathrm{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathrm{w}) - \hat{q}(S, A, \mathrm{w})\big]\nabla\hat{q}(S, A, \mathrm{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$

Mountain Car Task

Underpowered car, gravity is stronger than the car's engine, the only solution is to first move away from the goal and up the opposite slope on the left.

Reward -1 on all time step until the goal is not reached



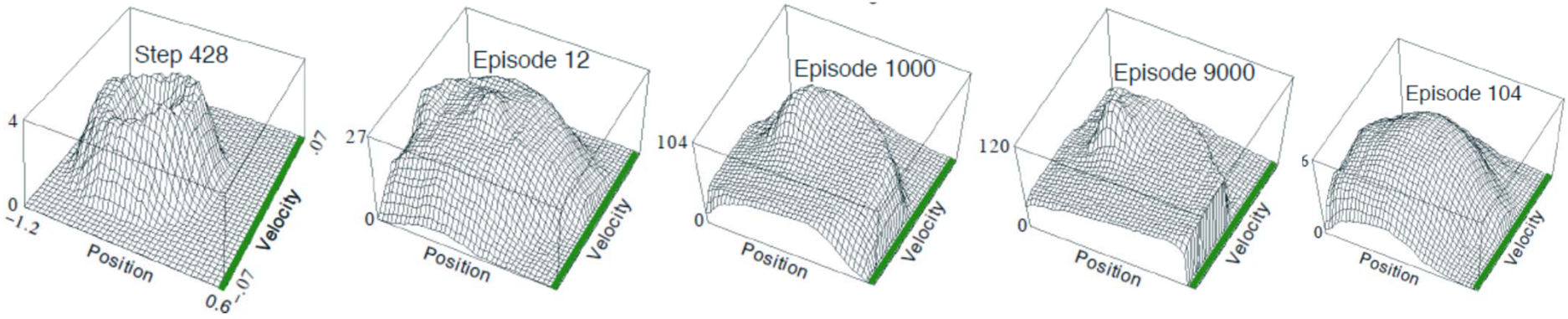
Three actions: forward, reverse, zero
Simplified physics:

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

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

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

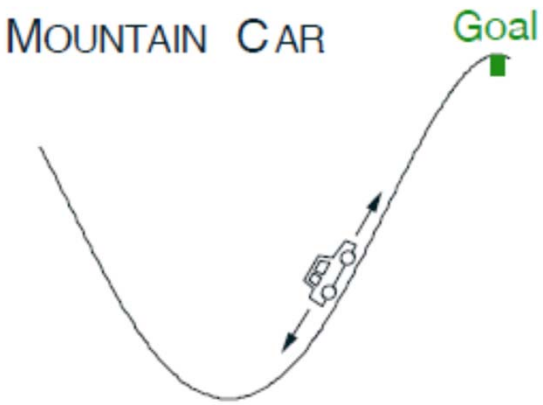
Velocity reset to zero when the left bound is reached
Each episode from a random position and zero velocity
Linear approximation $\hat{q}(s,a,\mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s,a) = \sum_i w_i \cdot x_i(s,a),$



Mountain Car Task

Underpowered car, gravity is stronger than the car's engine, the only solution is to first move away from the goal and up the opposite slope on the left.

Reward -1 on all time step until the goal is not reached



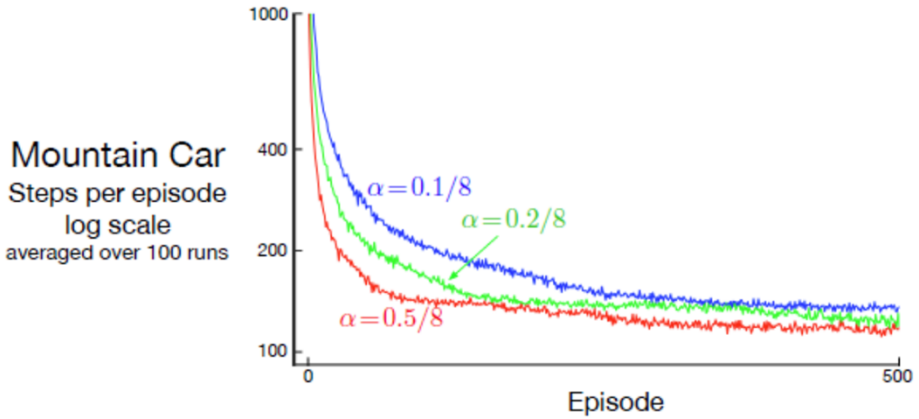
Three actions: forward, reverse, zero
Simplified physics:

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

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

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

Velocity reset to zero when the left bound is reached
Each episode from a random position and zero velocity
Linear approximation $\hat{q}(s,a,\mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s,a) = \sum_i w_i \cdot x_i(s,a),$



Convergence of Evaluation

- On-Policy
  - MC learning
    - Converge with table lookup, linear/non-linear approx
  - TD(0)\TD(λ) learning
    - Converge with table lookup and linear approx.
  - Gradient TD
    - Converge with table lookup, linear/non-linear approx
- Off-Policy
  - MC learning
    - Converge with Table, Linear/Non-linear approx
  - TD(0)\TD(λ) learning
    - Converge with table lookup
  - Gradient TD
    - Converge with table lookup, linear/non-linear approx

Convergence of Control

- Control Algorithm
  - MC learning
    - Converge with table lookup, (linear approx)
  - TD(0)\TD(λ) learning
    - Converge with table lookup, (linear approx.)
  - Gradient TD
    - Converge with table lookup, linear approx

(chatters around the near-optimal function)

Batch RL

- Gradient descent is appealing
- It is not sample efficient
- Batch methods try to find the best fitting of the value function given the experience (“training data”)

Least Squares Prediction

- Function approximation $\hat{v}(s, w) \approx v_\pi(s)$
- Experience $D = \{\langle s_1, v_1^\pi \rangle, \ldots, \langle s_T, v_T^\pi \rangle\}$
- Parameters w for the best fitting value $\hat{v}(s, w)$
- Least squares algorithms to find the w minimizing the sum-squared error between $\hat{v}(s, w)$ and $v_\pi(s)$

$$\mathrm{LS}(w) = \sum_{t=1 \ldots T} \left[\left(v_t^\pi - \hat{v}(s_t, w) \right)^2 \right] = E_D \left[\left(v_t^\pi - \hat{v}(s_t, w) \right)^2 \right]$$

Gradient Descent with Experience Replay

- Store experience $D = \{\langle s_1, v_1^\pi \rangle, \ldots, \langle s_T, v_T^\pi \rangle\}$
- Repeat
  - Sample $\langle s , v^\pi \rangle$ from the experience D
  - Apply stochastic gradient descent update towards the target

$$\Delta w = \alpha\big(v^\pi - \hat{v}(s,w)\big)\nabla_w \hat{v}(s,w)$$

Like supervised learning

Converges to least squares solution

$$w^\pi = argmin_w LS(w)$$

Experience Replay in DQN

Deep Q-Networks uses experience replay and fixed Q-target

- Take an action a_t according to an ϵ-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in memory D
- Sample mini-batch transitions (s, a, r, s') from D
- Compute Q-learning targets w.r.t. old fixed parameters w^-
- Optimize MSE between Q-network and Q-learning targets:

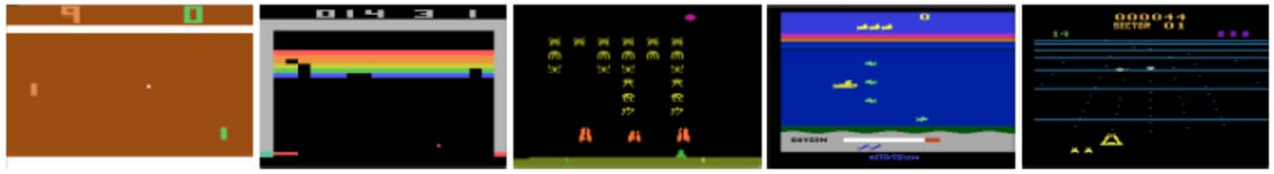
$$Loss_i(w_i) = E_{s,a,r,s' \sim D_i}\left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i)\right)^2\right]$$

- Minimize the loss with stochastic gradient descent
- After N iteration copy the actual parameters in the target
- Repeat for M episodes

- Key points to stabilize the process:
  - Decouple the policy and the learning data (mini-batch)
  - Decouple the fixed (old and frozen) Q-target and the Q-network
  - After a while switch old and new parameters …

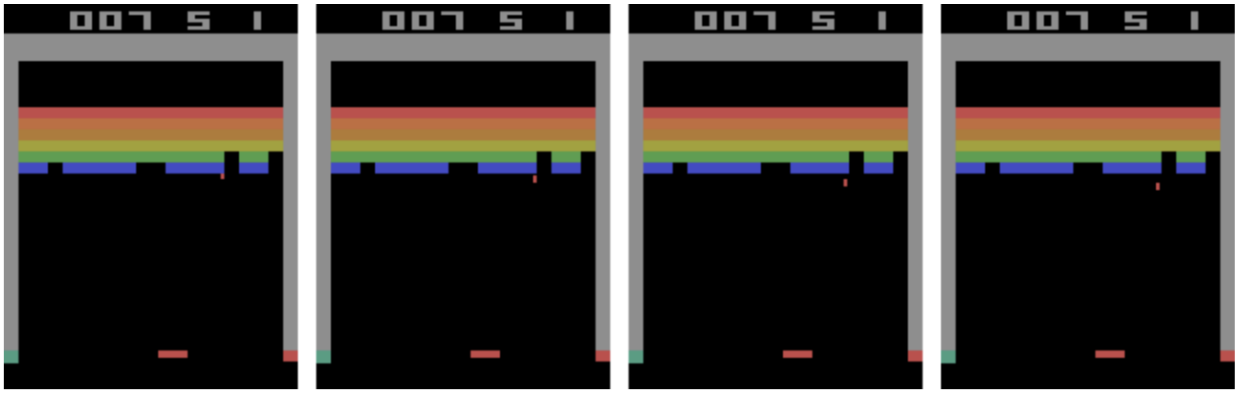
Experience Replay in DQN

Deep Q-Networks uses experience replay and fixed Q-target



Atari 2600 Playing:

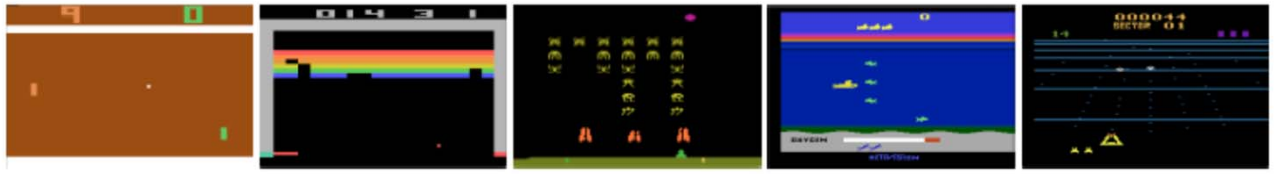
Input: 84 x 84 x 4 image produced by the preprocessing (gray-scale and down-sampling, crop), Atari frames are 210x160 pixel images, 128 color, 4 immages



Position, direction, velocity, acceleration

Experience Replay in DQN

Deep Q-Networks uses experience replay and fixed Q-target

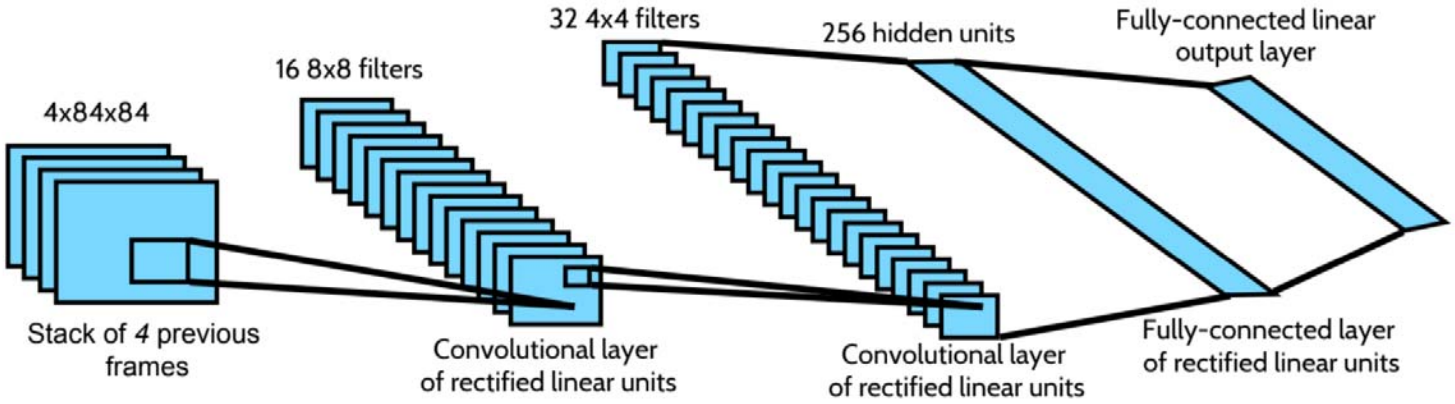


Atari 2600 Playing:

Hidden: 3 hidden conv. layers and a fully-connected hidden layer.

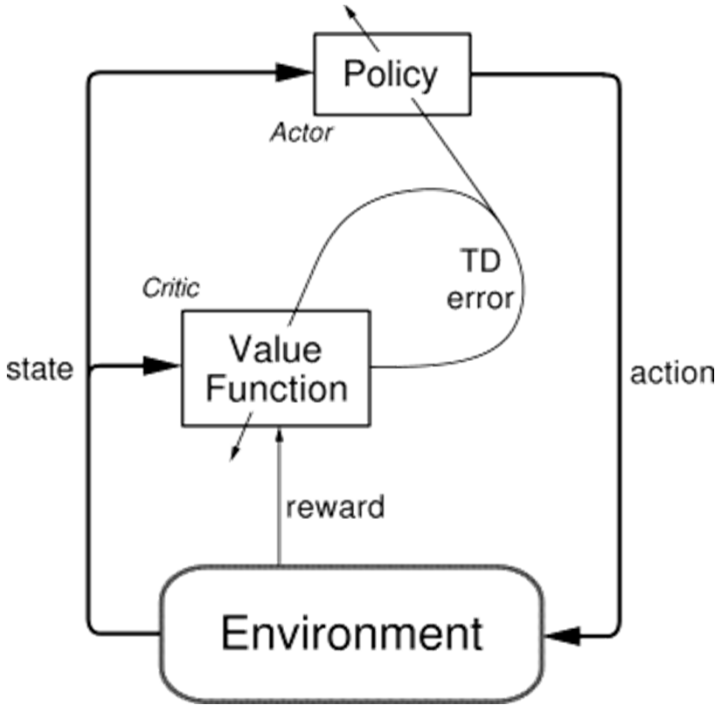
1. 16 8x8 filters, stride 4, rectifier
2. 32 4x4 filters, stride 2, rectifier
3. 256 hidden fully connected rectifier units

Output:
One for each possible action (4 to 18)



Actor Critic Method

- Policy structure (*actor)*: it selects the actions,
- Value function (*critic)*: it criticizes the actions made by the actor.



- Explicit representation of policy as well as value function
- Minimal computation to select actions
- Can learn an explicit stochastic policy
- Can put constraints on policies
- Appealing as psychological and neural models

- two parts of the striatum - dorsal and ventral subdivisions – may work as actor and critic
- TD error has the dual role of being the reinforcement signal for both the actor and the critic, with different influence

Actor-Critic Details

TD-error is used to evaluate actions:

$$\delta_t = r_{t+1} + V(s_{t+1}) - V(s_t)$$

If actions are determined by preferences, $p(s,a)$, as follows:

$$\pi_t(s,a) = \Pr\{a_t = a \,|\, s_t = s\} = \frac{e^{p(s,a)}}{\sum_b e^{p(s,b)}}, \qquad \text{(softmax)}$$

then you can update the preferences like this :

$$p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta \delta_t$$

p(s, a) tendency to select
(*preference* for) each action

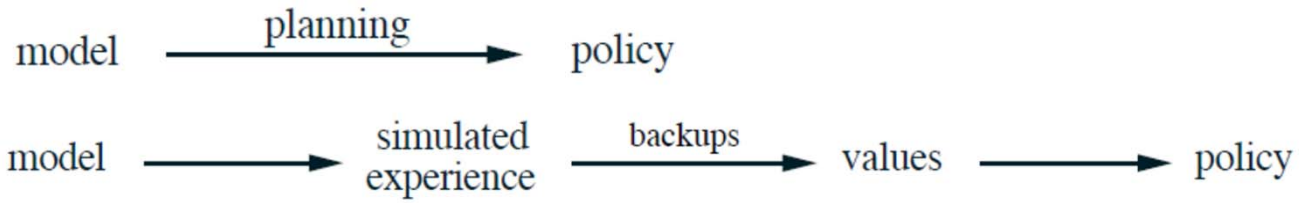
Learning and Planning

Combine Model-based and Model-free methods

Model-free
- Learn value function and action-value function via experience

Model-based
- Learn the model via experience
- Use the model to generate the value function/policy (learn from simulated experience)

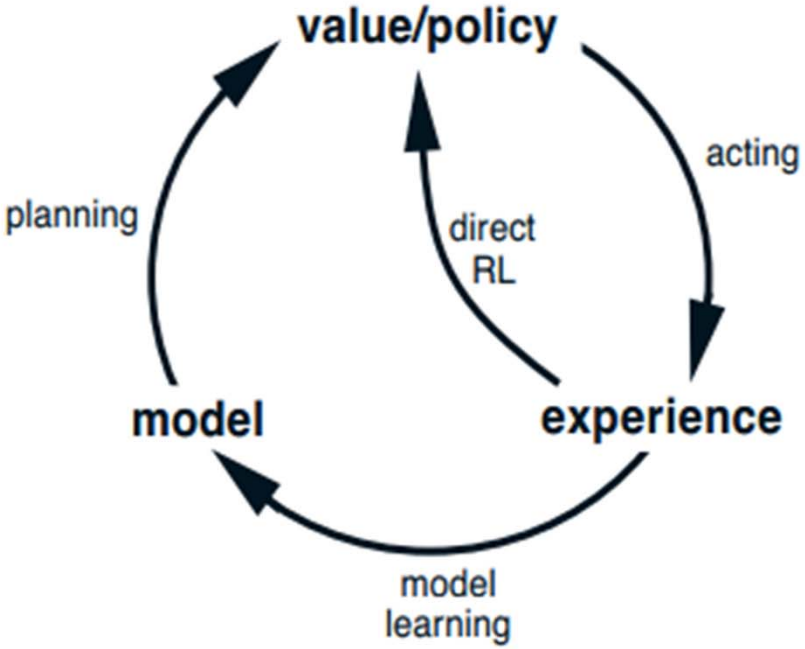
Combined (Dyna)
- Learn the model via experience
- Learn and plan via real and simulated experience

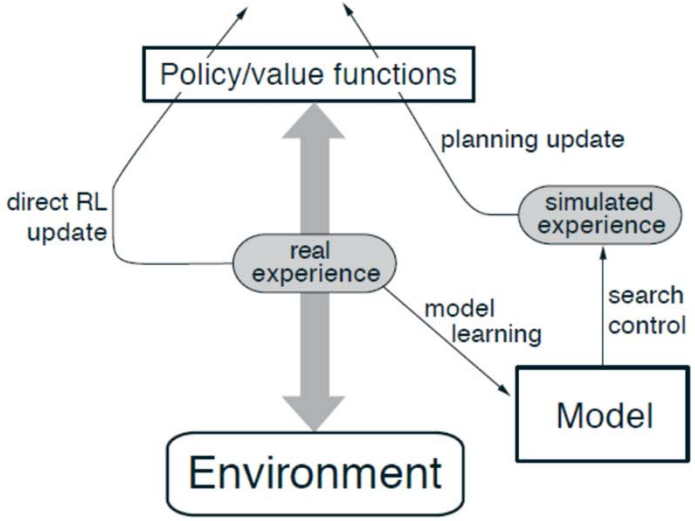


Learning and Planning

Combine Model-based and Model-free methods

Combined (Dyna)
- Learn the model via experience
- Learn and plan via real and simulated experience





Learning and Planning

Combine Model-based and Model-free methods

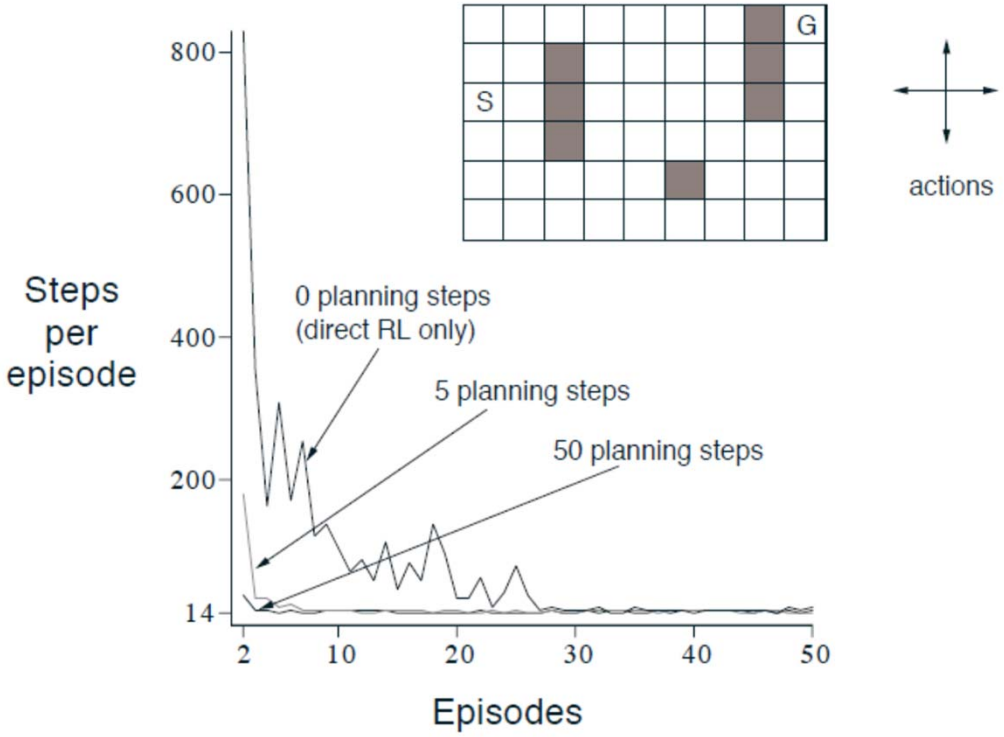
Combined (Dyna)
- Learn the model via experience
- Learn and plan via real and simulated experience

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow \epsilon$-greedy(S,Q)
(c) Execute action A; observe resultant reward, R, and state, S'
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
(f) Repeat n times:
$S \leftarrow$ random previously observed state
$A \leftarrow$ random action previously taken in S
$R, S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Learning and Planning

Combine Model-based and Model-free methods

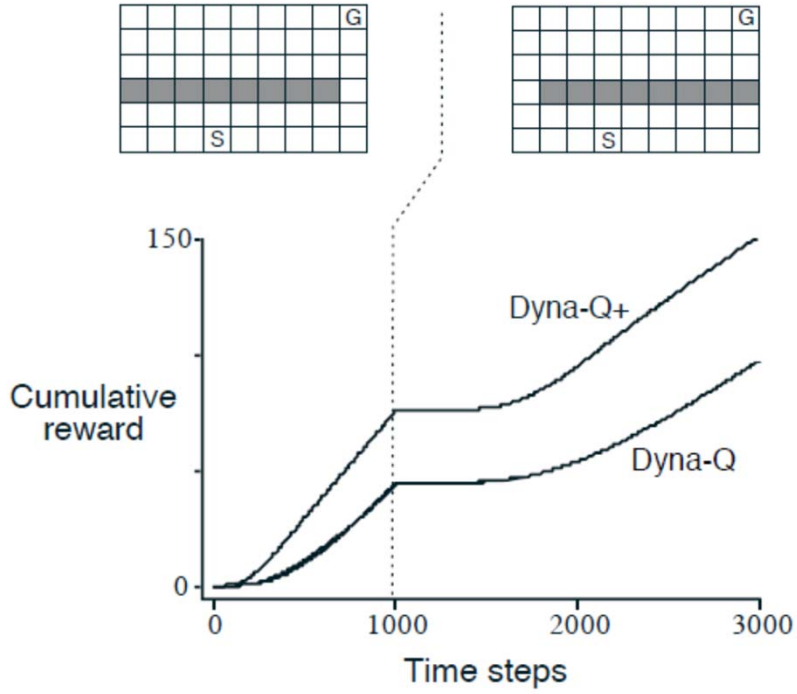
Combined (Dyna)
- Learn the model via experience
- Learn and plan via real and simulated experience



Learning and Planning

Combine Model-based and Model-free methods

Combined (Dyna)
- When the model is wrong …
- DynaQ+ has a bonus on the explorative behavior



Learning and Planning

Combine Model-based and Model-free methods

Combined (Dyna)
- When the model is wrong …
- DynaQ+ has a bonus on the explorative behavior