Q-Learning, SARSA

Reinforcement Learning

Roberto Capobianco

Sapienza Università di Roma

Recap

Policy Iteration

- Outputs policies at every iteration: $\{\pi_0, \pi_1, \pi_2 ... \pi_T\}$
- Different from Value Iteration that was outputting values

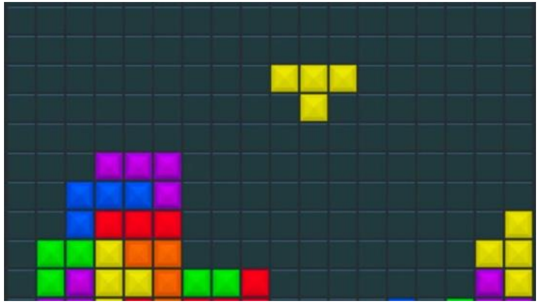
Procedure:

1. Start with a random guess π_0 (can be deterministic or stochastic)
2. For t=0,...,T:
   a. Do **policy evaluation** and compute $Q^{\pi t}$ for all s,a
   $$Q^{\pi}(s_t,a) = r_t+\gamma\mathbb{E}_{s'\sim p(.|s,a)}[V^{\pi}(s')]$$
   b. Do **policy improvement** as $\pi_{t+1}=\text{argmax}_a Q^{\pi t}(s,a)$ for all s

This algorithm only makes progress, and the performance progress of the policy is monotonic

Approximate Policy Iteration

What if the state-space is large or continuous and we cannot do exact or iterative policy evaluation for all states?



Approximate Policy Iteration

- Outputs policies at every iteration: $\{\pi_0, \pi_1, \pi_2 ... \pi_T\}$

Procedure:

1. Start with a random guess π_0
2. For t=0,...,T:
   a. Do **policy evaluation** and compute $Q^{\wedge\pi t}$ ~~for all s,a~~

$$Q^{\wedge\pi t}(s_t,a) = r_t + \gamma \mathbb{E}_{s' \sim p(.|s,a)}[V^{\wedge\pi}(s')]$$

   a. Do **policy improvement** as $\pi_{t+1} = \text{argmax}_a Q^{\wedge\pi t}(s,a)$ ~~for all s~~

   argmax is still doable, we can still enumerate actions or discretize them

Approximate Policy Evaluation

We build an **approximation V^**$^{\pi}$ of the true value function V$^{\pi}$

If the approximation is close to the true value, then the optimal policy will be close-to-optimal

Approximation for large state-spaces is needed to generalize among states and avoid looking at the whole S

We use a function approximator

e.g., linear approximators, neural nets, non-parametric, etc.

Approximate Policy Evaluation

To be fair, we can directly approximate Q, so let’s do that

Note that this also means that we can also get rid of the assumption of knowing the MDP

Data and Least Square Regression

To be fair, we can directly approximate Q, so let's do that

What do we need?

DATA $D = \{s_i, a_i, y_i\}_{i=1}^{N}$ with y being our label!

with those we can then use least-square regression to extract a function Q in the family of functions

$Q: S \times A \rightarrow [0, 1/(1-\gamma)]$

$$\mathrm{argmin}_{Q \text{ in } \mathcal{Q}} \sum_{i=1}^{N} (Q(s_i, a_i) - y_i)^2$$

SAPIENZA UNIVERSITÀ DI ROMA

Supervised Learning: Regression

Given a ***data distribution D*** from which we sample points x_i and labels $y_i=f(x_i)+\epsilon_i$, with $\mathbb{E}[\epsilon_i]=0$ and $|\epsilon_i|\leq c$, we want to approximate f using a finite set of data (dataset):

Empirical Risk Minimizer

$$\hat{f} = \mathrm{argmin}_{\hat{f} \text{ in } F} \sum_{i=1}^{N} (\hat{f}(x_i)-y_i)^2$$

with $F=\{\hat{f}: X \to \mathbb{R}\}$

We can generalize under the same data distribution

$\mathbb{E}_{x\sim D}(\hat{f}(x)-f(x))^2 \leq \delta$ with δ small

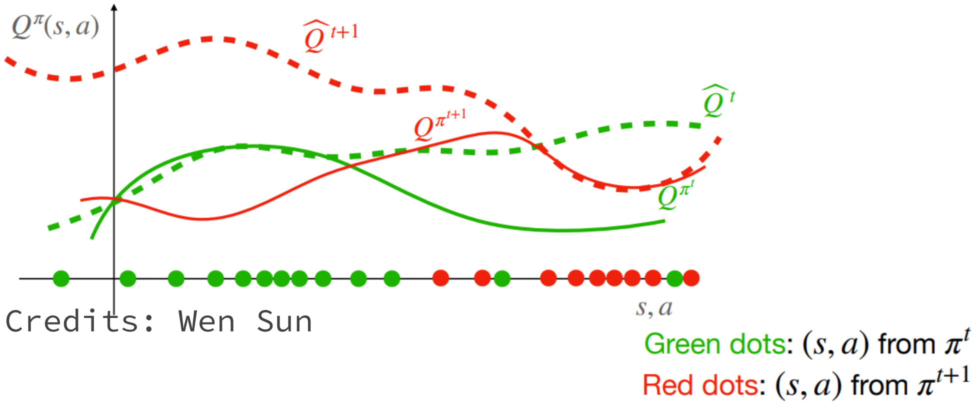
$\mathbb{E}_{x\sim D'}(\hat{f}(x)-f(x))^2$ can be huge!

If $D' \neq D$

SAPIENZA UNIVERSITÀ DI ROMA

Oscillation from Distribution Change

We cannot guarantee anymore monotonic improvement!



Credits: Wen Sun

Our estimation is only good under $d_{\mu_0}^{\pi}$ and to make sure we have monotonic improvement we need a strong coverage assumption

Data Generation

---

2 steps:

1. **Roll-in**
2. Roll-out & compute supervision targets

We want to sample our $(s,a) \sim d^{\pi}_{s0}(s,a) = (1-\gamma)\sum^{\infty}_{h=0}\gamma^h \mathbb{P}^{\pi}_h(s,a;s_0)$

- Sample h from $\gamma^h(1-\gamma)$, thus committing to a specific $\mathbb{P}^{\pi}_h(s,a;s_0)$
- Follow π for h timesteps starting from $s_0 \sim \mu_0$ and get s_h, a_h

Data Generation

---

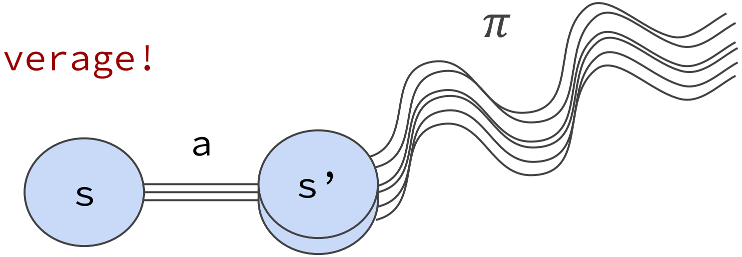
2 steps:

1. Roll-in
2. **Roll-out & compute supervision targets**

Given s, a, how do we estimate $Q^{\pi}(s,a)$?

$$Q^{\pi}(s_t,a_t) = \mathbb{E}[\sum^{\infty}_{h=0}\gamma^h r_h \,|\, (s_0,a_0)=(s_t,a_t), a_{h+1}=\pi(s_h), s_{h+1}\sim p(.|s_h,a_h)]$$

Sample many times and average!



Data Generation

---

2 steps:

1. Roll-in
2. **Roll-out & compute supervision targets**

Given s, a, how do we estimate $Q^{\pi}(s,a)$?

- Start at s,a
- Repeat:
  - Get r(s,a)
  - With probability 1-γ terminate and return $y=\sum \gamma^h r_h$
  - Execute action and get in s'

$D = \{s_i, a_i, y_i\}_{i=1}^{N}$

End Recap

SAPIENZA
UNIVERSITÀ DI ROMA

Data Generation

---

2 steps:

1. Roll-in
2. **Roll-out & compute supervision targets**

Given s, a, how do we estimate $Q^\pi(s,a)$?

$$Q^\pi(s_t,a_t) = \mathbb{E}[\textstyle\sum^\infty_{h=0}\gamma^h r_h | (s_0,a_0)=(s_t,a_t), a_{h+1}=\pi(s_h), s_{h+1}\sim p(.|s_h,a_h)]$$

$$Q^\pi(s_t,a) = r_t+\gamma\mathbb{E}_{s'\sim p(.|s,a)}[V^\pi(s')]$$

Data Generation

---

2 steps:

1. Roll-in
2. **Roll-out & compute supervision targets**

Given s, a, how do we estimate $Q^\pi(s,a)$?

$$Q^\pi(s_t,a_t) = \mathbb{E}[\sum^{\infty}_{h=0}\gamma^h r_h | (s_0,a_0)=(s_t,a_t), a_{h+1}=\pi(s_h), s_{h+1}\sim p(.|s_h,a_h)]$$

Monte-Carlo method: estimate this through sampling, execute until termination and then average many roll-outs to compute our estimate

Data Generation

---

2 steps:

1. Roll-in
2. **Roll-out & compute supervision targets**

Given s, a, how do we estimate $Q^{\pi}(s,a)$?

$$Q^{\pi}(s_t,a_t) = \mathbb{E}[\textstyle\sum^{\infty}_{h=0}\gamma^h r_h | (s_0,a_0)=(s_t,a_t), a_{h+1}=\pi(s_h), s_{h+1}\sim p(.|s_h,a_h)]$$

Monte-Carlo method: estimate this through sampling, execute until termination and then average many roll-outs to compute our estimate. **Note: True MC cannot be applied if infinite horizon, but we can adapt using the 'trick' shown in previous class**

Monte-Carlo Update

$$Q^{\pi}(s_t,a_t) = \mathbb{E}[\textstyle\sum^{\infty}_{h=0}\gamma^h r_h \,|\, (s_0,a_0)=(s_t,a_t), a_{h+1}=\pi(s_h), s_{h+1}\sim p(.|s_h,a_h)]$$

Monte-Carlo method: estimate this through sampling, execute until termination and then average many roll-outs to compute our estimate

Compute target: $y = G$

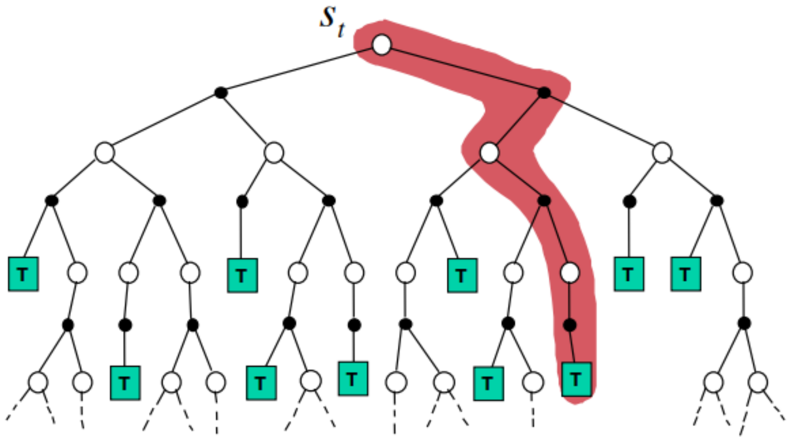
- G is the return
- The return is the sum of rewards $\sum_{h=0}^{\text{Termination}}\gamma^h r_h$

Sapienza Università di Roma

Monte-Carlo Update

$$Q^{\pi}(s_t,a_t) = \mathbb{E}[\sum^{\infty}_{h=0}\gamma^h r_h | (s_0,a_0)=(s_t,a_t), a_{h+1}=\pi(s_h), s_{h+1}\sim p(.|s_h,a_h)]$$

Monte-Carlo method: estimate this through sampling, execute until termination and then average many roll-outs to compute our estimate



Credits: David Silver

Monte-Carlo Update

$$Q^{\pi}(s_t,a_t) = \mathbb{E}[\sum_{h=0}^{\infty}\gamma^h r_h | (s_0,a_0)=(s_t,a_t), a_{h+1}=\pi(s_h), s_{h+1}\sim p(.|s_h,a_h)]$$

Monte-Carlo method: estimate this through sampling, execute until termination and then average many roll-outs to compute our estimate

If we use a function approximator, just do **regression**

$$\mathrm{argmin}_{Q \text{ in } Q} \sum_{i=1}^{N}(Q(s_i,a_i)-y_i)^2$$

Monte-Carlo Update

$$Q^{\pi}(s_t,a_t) = \mathbb{E}[\sum^{\infty}_{h=0}\gamma^h r_h | (s_0,a_0)=(s_t,a_t), a_{h+1}=\pi(s_h), s_{h+1}\sim p(.|s_h,a_h)]$$

Monte-Carlo method: estimate this through sampling, execute until termination and then average many roll-outs to compute our estimate

If we use a function approximator, just do **regression**

Regression will converge to the mean of the returns!

$$\mathrm{argmin}_{Q \text{ in } Q} \sum_{i=1}^{N}(Q(s_i,a_i)-y_i)^2$$

SAPIENZA
UNIVERSITÀ DI ROMA

Monte-Carlo Update

$$Q^{\pi}(s_t,a_t) = \mathbb{E}[\textstyle\sum^{\infty}_{h=0}\gamma^h r_h \mid (s_0,a_0)=(s_t,a_t), a_{h+1}=\pi(s_h), s_{h+1}\sim p(.\mid s_h,a_h)]$$

Monte-Carlo method: estimate this through sampling, execute until termination and then average many roll-outs to compute our estimate

If state space is not huge, why should we do approximation if we can rely on a table?

Monte-Carlo Update

$$Q^{\pi}(s_t,a_t) = \mathbb{E}[\sum^{\infty}_{h=0}\gamma^h r_h|(s_0,a_0)=(s_t,a_t),a_{h+1}=\pi(s_h),s_{h+1}\sim p(.|s_h,a_h)]$$

Monte-Carlo method: estimate this through sampling, execute until termination and then average many roll-outs to compute our estimate

If state space is not huge, why should we do approximation if we can rely on a table?

How do we do tabular updates?

Moving Average

We want to compute our estimate as a mean and update it with new data coming from agent’s experience

Moving Average

We want to compute our estimate as a mean and update it with new data coming from agent's experience

Definition of a mean:

$$\mu_k = 1/k \sum_0^{k-1} x_k$$

Moving Average

We want to compute our estimate as a mean and update it with new data coming from agent's experience

Definition of a mean:

$$\mu_k = 1/k\sum_0^{k-1}x_j = 1/k(x_k + \sum_0^{k-2}x_j) = 1/k(x_k + (k-1)\mu_{k-1})$$

Moving Average

We want to compute our estimate as a mean and update it with new data coming from agent's experience

Definition of a mean:

$$\mu_k = 1/k\sum_0^{k-1}x_j = 1/k(x_k + \sum_0^{k-2}x_j) = 1/k(x_k + (k-1)\mu_{k-1}) =$$

$$\boldsymbol{\mu_{k-1} + 1/k(x_k - \mu_{k-1})}$$

Moving Average

In non-stationary problem we may want to forget (a bit) the past (i.e., compute a running mean)

Definition of a mean:

$$\mu_k = 1/k\sum_0^{k-1}x_j = 1/k(x_k + \sum_0^{k-2}x_j) = 1/k(x_k + (k-1)\mu_{k-1}) =$$

$$\mu_{k-1} + 1/k(x_k - \mu_{k-1})$$

$$\mu_{k-1} + a(x_k - \mu_{k-1})$$

Tabular Updates

We now got a general form to do our updates in tabular form:

$$p_{k-1} + \alpha(y_k - p_{k-1})$$

- p_{k-1} is our current estimate
- α is between 0 and 1
- y_k is our target or label

MC Updates

Tabular:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(G_i - Q(s,a))$$

Function Approximator:

$$\text{argmin}_{Q \text{ in } Q} \sum_{i=1}^{N} (Q(s_i,a_i) - G_i)^2$$

MC Updates

Tabular:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(G_i - Q(s,a))$$

we do this at every termination

Function Approximator:

$$\operatorname{argmin}_{Q \text{ in } Q} \sum_{i=1}^{N} (Q(s_i,a_i) - G_i)^2$$

we store data and update in batch after a while or do online learning (at every datapoint - less stable)

MC Pros & Cons

Weaknesses:

- Needs some sort of termination

MC Pros & Cons

Weaknesses:

- Needs some sort of termination
- Depends on many random actions, transitions, rewards

High variance!

MC Pros & Cons

Weaknesses:

- Needs some sort of termination
- Depends on many random actions, transitions, rewards
- Needs complete sequences of returns

MC Pros & Cons

Weaknesses:

- Needs some sort of termination
- Depends on many random actions, transitions, rewards
- Needs complete sequences of returns

Strengths:

- Unbiased

MC Pros & Cons

Weaknesses:

- Needs some sort of termination
- Depends on many random actions, transitions, rewards
- Needs complete sequences of returns

Strengths:

- Unbiased
- Good convergence properties also with function approx

MC Pros & Cons

Weaknesses:

- Needs some sort of termination
- Depends on many random actions, transitions, rewards
- Needs complete sequences of returns

Strengths:

- Unbiased
- Good convergence properties also with function approx
- Not very sensitive to initialization

Data Generation

---

2 steps:

1. Roll-in
2. **Roll-out & compute supervision targets**

Given s, a, how do we estimate $Q^{\pi}(s,a)$?

$$Q^{\pi}(s_t,a_t) = \mathbb{E}[\textstyle\sum^{\infty}_{h=0}\gamma^h r_h \mid (s_0,a_0)=(s_t,a_t), a_{h+1}=\pi(s_h), s_{h+1}\sim p(.\mid s_h,a_h)]$$

$$\mathbf{Q^{\pi}(s_t,a) = r_t+\gamma\mathbb{E}_{s'\sim p(.\mid s,a)}[V^{\pi}(s')]}$$

Data Generation

---

2 steps:

1. Roll-in
2. **Roll-out & compute supervision targets**

Given s, a, how do we estimate $Q^\pi(s,a)$?

$$\mathbf{Q^\pi(s_t,a) = r_t+\gamma\mathbb{E}_{s'\sim p(.|s,a)}[V^\pi(s')]}$$

Monte-Carlo uses the actual return. In Temporal Difference we use an estimated return: our current V

Data Generation

---

2 steps:

1. Roll-in
2. **Roll-out & compute supervision targets**

Given s, a, how do we estimate $Q^{\pi}(s,a)$?

$$Q^{\pi}(s_t,a) = r_t + \gamma \mathbb{E}_{s' \sim p(.|s,a)}[V^{\pi}(s')]$$

Temporal Difference method: exploits the Markov property and, as a result, it’s more efficient than MC in Markov environments (and viceversa)

Data Generation

---

2 steps:

1. Roll-in
2. **Roll-out & compute supervision targets**

Given s, a, how do we estimate $Q^\pi(s,a)$?

$$Q^\pi(s_t,a) = r_t + \gamma \mathbb{E}_{s' \sim p(.|s,a)}[V^\pi(s')]$$

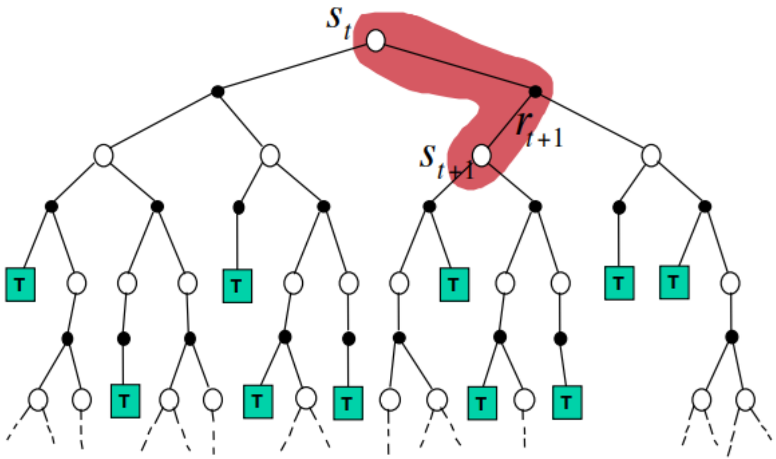
Bootstrapping: an estimate of the next state value is used instead of the true next state value

Temporal Difference method: estimate this through sampling, update our estimate towards the current reward and the current estimated return (*bootstrapping*) from incomplete episodes

Temporal Difference Update

$$Q^{\pi}(s_t, a) = r_t + \gamma \mathbb{E}_{s' \sim p(.|s,a)}[V^{\pi}(s')]$$

Temporal Difference method: estimate this through sampling, update our estimate towards the current reward and the current estimated return (*bootstrapping*) from incomplete episodes



Credits: David Silver

TD Updates

Tabular:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_i+\gamma Q(s',.) - Q(s,a))$$

$r_i+\gamma Q(s',.) - Q(s,a)$ is called *TD error* (δ)

Function Approximator:

$$\text{argmin}_{Q \text{ in } Q} \sum_{i=1}^{N} (Q(s_i,a_i)-r_i+\gamma Q(s',.))^2$$

TD Updates

Tabular:

?

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_i + \gamma Q(s', .) - Q(s,a))$$

Function Approximator:

$$\text{argmin}_{Q \text{ in } Q} \sum_{i=1}^{N} (Q(s_i, a_i) - r_i + \gamma Q(s', .))^2$$

TD Updates

Tabular: We will see it later

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_i + \gamma Q(s',.) - Q(s,a))$$

Function Approximator:

$$\mathrm{argmin}_{Q \text{ in } Q} \sum_{i=1}^{N} (Q(s_i,a_i) - r_i + \gamma Q(s',.))^2$$

TD Updates

Tabular:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_i+\gamma Q(s',.) - Q(s,a))$$

we do this at every timestep

Function Approximator:

$$\text{argmin}_{Q \text{ in } Q} \sum_{i=1}^{N} (Q(s_i,a_i) - r_i + \gamma Q(s',.))^2$$

still we store data and update in batch after a while or do online learning (at every datapoint - less stable), but many more data-points than MC with same experience

TD Pros & Cons

Weaknesses:

- Sensitive to initial value

TD Pros & Cons

Weaknesses:

- Sensitive to initial value
- Biased estimate of Q^π

it would be unbiased if our target was $r_i+\gamma Q^\pi(s',.)$ with the true Q^π instead of the estimated one

TD Pros & Cons

Weaknesses:

- Sensitive to initial value
- Biased estimate of Q^π

Strengths:

- Can learn at every step, from incomplete sequences and in continuing tasks easily

more efficient than MC

TD Pros & Cons

Weaknesses:

- Sensitive to initial value
- Biased estimate of Q^π

Strengths:

- Can learn at every step, from incomplete sequences and in continuing tasks easily
- Depends on just one action instead of a sequence like MC

less variance

TD Pros & Cons

Weaknesses:

- Sensitive to initial value
- Biased estimate of Q^π

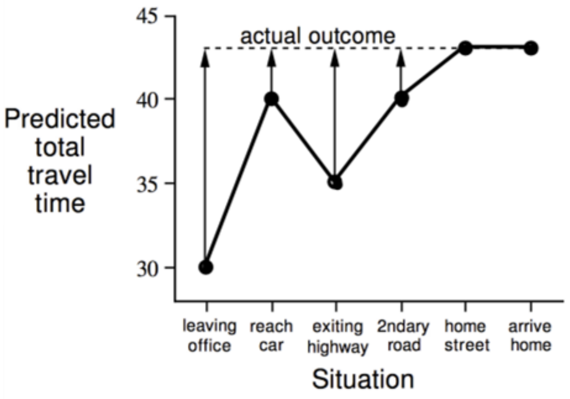
Strengths:

- Can learn at every step, from incomplete sequences and in continuing tasks easily
- Depends on just one action instead of a sequence like MC
- Convergences but not always if function approx

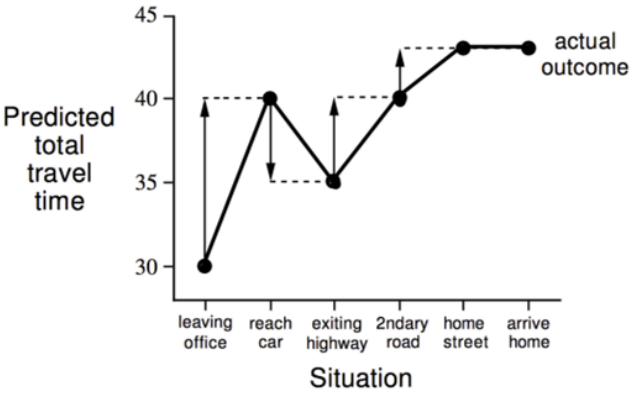
TD & MC Example: Driving Home

State	Elapsed Time (minutes)	Predicted Time to Go	Predicted Total Time
leaving office	0	30	30
reach car, raining	5	35	40
exit highway	20	15	35
behind truck	30	10	40
home street	40	3	43
arrive home	43	0	43

Changes recommended by Monte Carlo methods (α=1)



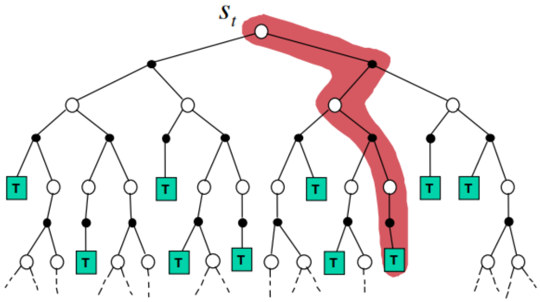
Changes recommended by TD methods (α=1)



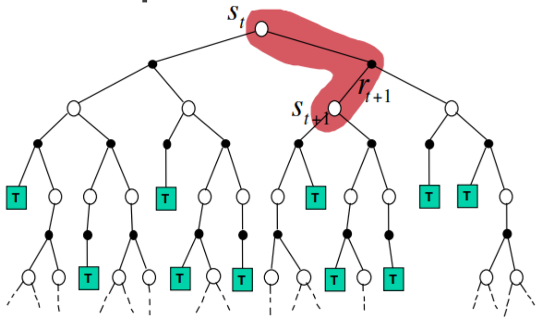
SAPIENZA
UNIVERSITÀ DI ROMA

MC vs TD vs DP

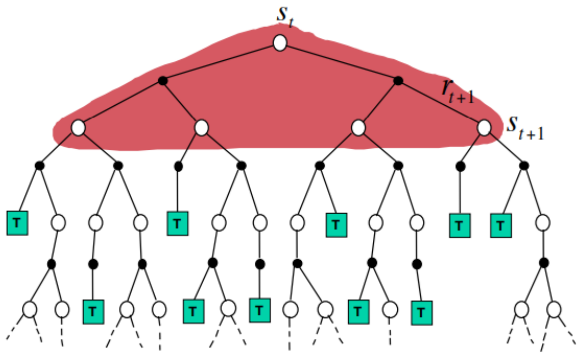
Monte-Carlo



Temporal Difference



Dynamic Programming

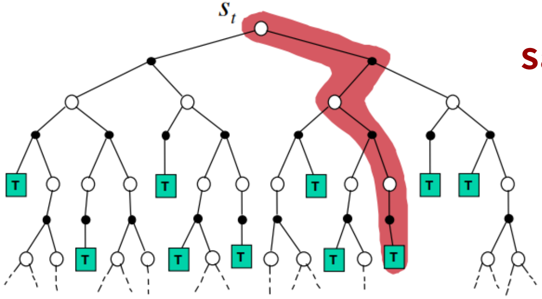


SAPIENZA UNIVERSITÀ DI ROMA

Credits: David Silver

MC vs TD vs DP

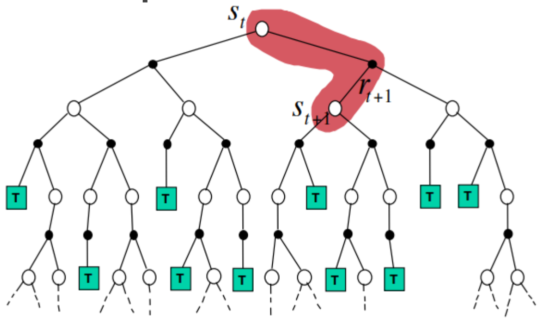
Monte-Carlo



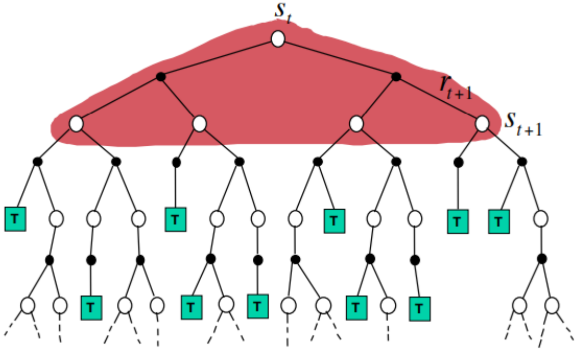
Sampling

Sampling

Temporal Difference



Dynamic Programming

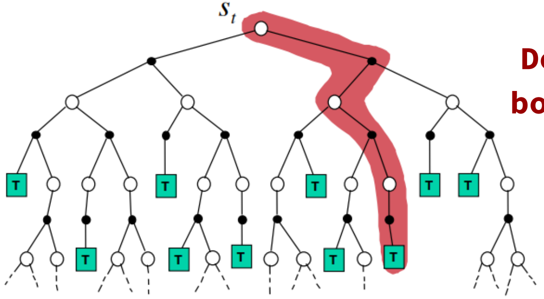


Compute expectation

Credits: David Silver

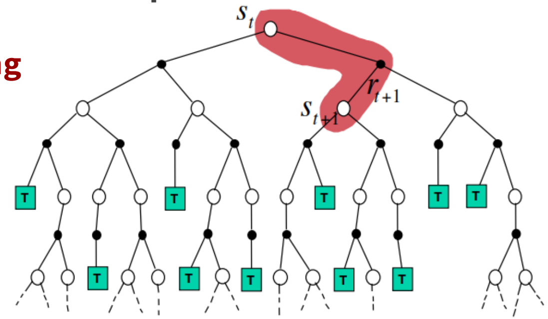
MC vs TD vs DP

Monte-Carlo



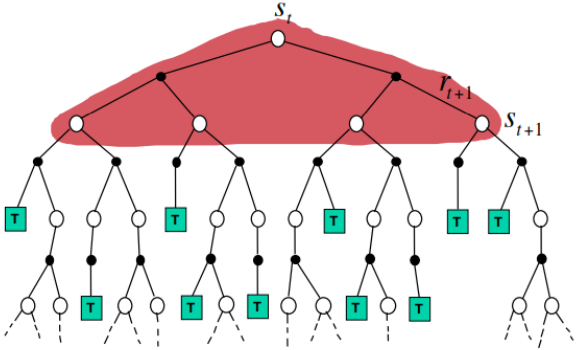
Does not bootstrap

Temporal Difference



Bootstrapping

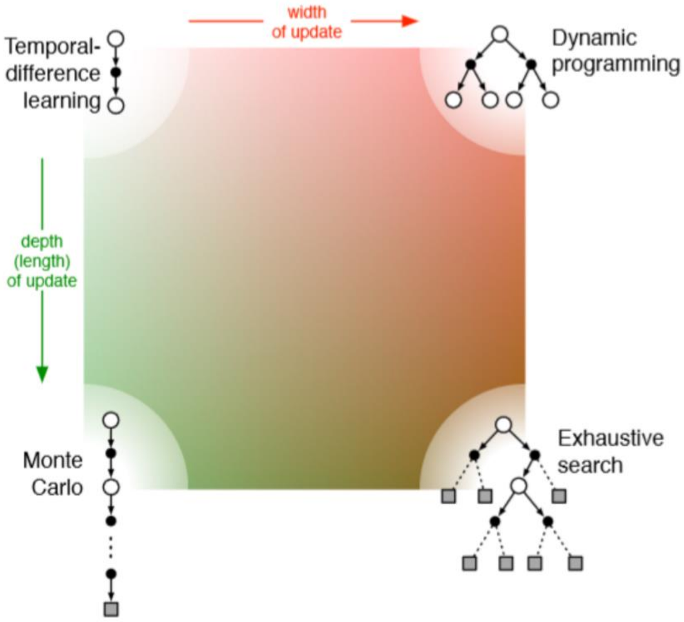
Dynamic Programming



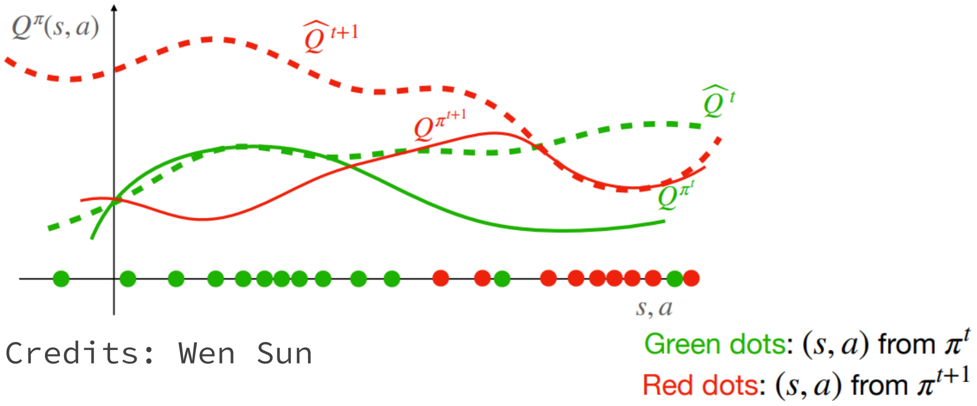
Bootstrapping

Credits: David Silver

MC vs TD vs DP


width
of update
Temporal-
difference
learning
Dynamic
programming
depth
(length)
of update
Monte
Carlo
Exhaustive
search

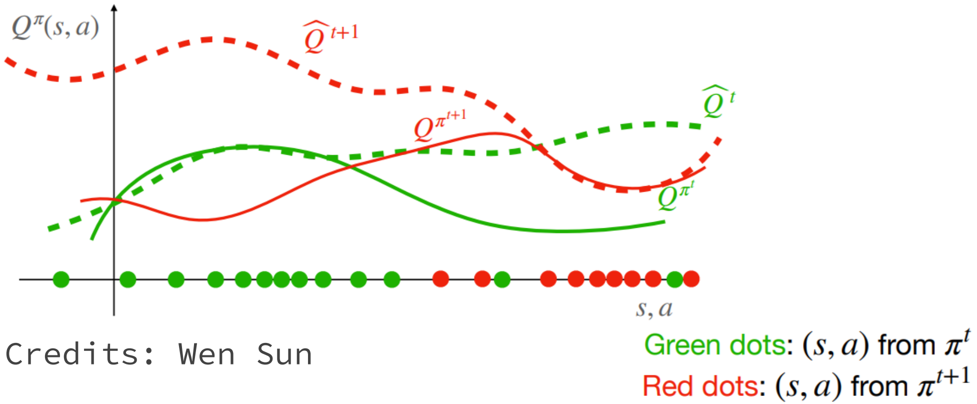
Exploration



Credits: Wen Sun

Remember? we need a strong coverage assumption

Exploration



Credits: Wen Sun

Simplest idea: instead of only being greedy with respect to Q, try all actions with some probability

ϵ-Greedy Exploration

Simplest idea: instead of only being greedy with respect to Q, try all actions with some probability

- probability 1-ϵ choose the greedy action (do argmax)
- probability ϵ choose a random action

This handles the **exploration-exploitation trade-off**

Suppose m act

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a\in\mathcal{A}}{\operatorname{argmax}}\, Q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

ϵ-Greedy Policy Improvement

For any ϵ-greedy policy π, the ϵ-greedy policy π' obtained by Q^{π} is an improvement, such that $V^{\pi'} \geq V^{\pi}$ holds

ϵ-Greedy Policy Improvement

For any ϵ-greedy policy π, the ϵ-greedy policy π' obtained by Q^{π} is an improvement, such that $V^{\pi'} \geq V^{\pi}$ holds

Prove it at home

ϵ-Greedy Policy Improvement

For any ϵ-greedy policy π, the ϵ-greedy policy π' obtained by Q^π is an improvement, such that $V^{\pi'} \geq V^\pi$ holds

If we set $\epsilon = 1/k$, with k going to infinity

- we visit all state-action pairs infinitely many times
- the policy converges to a greedy policy

ϵ-Greedy Policy Improvement

For any ϵ-greedy policy π, the ϵ-greedy policy π' obtained by Q^π is an improvement, such that $V^{\pi'} \geq V^\pi$ holds

If we set $\epsilon = 1/k$, with k going to infinity

- we visit all state-action pairs infinitely many times
- the policy converges to a greedy policy

Greedy in the Limit with Infinite Exploration

ϵ-Greedy and MC

If we apply Greedy in the Limit with Infinite Exploration to MC we converge to the optimal Q*

$$\epsilon \leftarrow 1/k$$
$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

ϵ-Greedy and TD

Remember $r_i+\gamma Q(s',.)$?

How do we select .?

ϵ-Greedy and TD

Remember $r_i+\gamma Q(s',.)$?

How do we select .?

Sarsa: the target action is selected according to π (which can be eps-greedy with respect to Q)

Q-learning: the target action is greedy with respect to Q

ϵ-Greedy and TD

Remember $r_i + \gamma Q(s', .)$?

How do we select .?

Sarsa: the target action is selected according to π (which can be eps-greedy with respect to Q)

Selects the target action according to the same policy we execute

Q-learning: the target action is greedy with respect to Q

ϵ-Greedy and TD

Remember $r_i+\gamma Q(s',.)$?

How do we select .?

Sarsa: the target action is selected according to π (which can be eps-greedy with respect to Q)

Selects the target action according to the same policy we execute

Q-learning: the target action is greedy with respect to Q

Selects the target action differently from the policy we execute (which must be eps-greedy, remember?)

ϵ-Greedy and TD

Remember $r_i+\gamma Q(s',.)$?

How do we select .?

Sarsa: the target action is selected according to π (which can be eps-greedy with respect to Q)

Selects the target action according to the same policy we execute

ON-POLICY

Q-learning: the target action is greedy with respect to Q

Selects the target action differently from the policy we execute (which must be eps-greedy, remember?)

OFF-POLICY

On-Policy vs Off-Policy

On-policy: learn by what you do

Off-policy: learn by looking at someone else

- learn from observing other agents or humans
- reuse experience
- learn about optimal policy while following exploratory behaviors
- learn multiple policies while following a single policy

Sarsa

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

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)

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

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

  until S is terminal

Sarsa

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
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)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Q-Learning

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

Repeat (for each episode):

  Initialize S

  Repeat (for each step of episode):

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

    Take action A, observe R, S'

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

    $S \leftarrow S'$;

  until S is terminal

Q-Learning

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

Repeat (for each episode):

  Initialize S

  Repeat (for each step of episode):

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

    Take action A, observe R, S'

    $\boxed{Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]}$

    $S \leftarrow S'$;

  until S is terminal

Convergence of Sarsa & Q-Learning

Sarsa: if we apply Greedy in the Limit with Infinite Exploration and set the step size α for the tabular setting to a Robbins-Monro sequence we converge to the optimal Q^*

Q-Learning: converges to the optimal Q^* under the same conditions

Convergence of Sarsa & Q-Learning

Sarsa: if we apply Greedy in the Limit with Infinite Exploration and set the step size α for the tabular setting to a **Robbins-Monro sequence** we converge to the optimal Q^*

Q-Learning: converges to the optimal Q^* under the same conditions

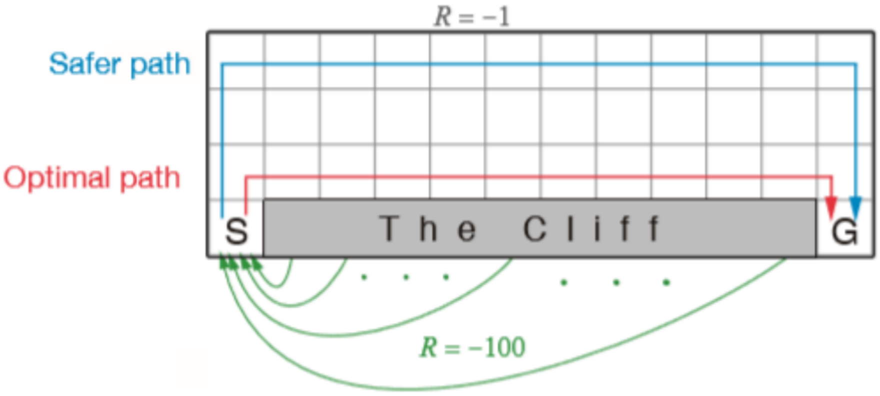
$$\sum_{n=0}^{\infty} \alpha_n = \infty \text{ and } \sum_{n=0}^{\infty} \alpha_n^2 < \infty$$

$$\text{e.g. } \alpha_n = \alpha/n \text{ for } \alpha > 0$$

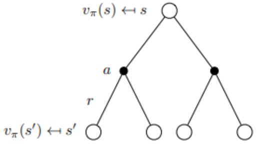
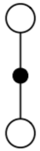
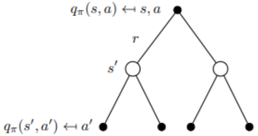
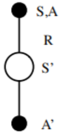
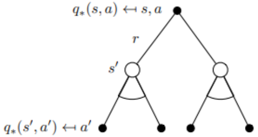
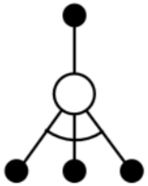
Sarsa & Q-Learning: Example

Sarsa: takes action selection into account and learns the safer path

Q-Learning: learns the optimal path independently of the action selection that, at learning time (i.e., while being eps-greedy), makes it fall in the cliff



TD, Sarsa & Q-Learning vs DP

	Full Backup (DP)	*Sample Backup (TD)*
Bellman Expectation Equation for $v_\pi(s)$	 Iterative Policy Evaluation	 TD Learning
Bellman Expectation Equation for $q_\pi(s,a)$	 Q-Policy Iteration	 Sarsa
Bellman Optimality Equation for $q_*(s,a)$	 Q-Value Iteration	 Q-Learning

Credits: David Silver

TD, Sarsa & Q-Learning vs DP

Full Backup (DP)	*Sample Backup (TD)*
Iterative Policy Evaluation $V(s) \leftarrow \mathbb{E}[R + \gamma V(S') \mid s]$	TD Learning $V(S) \xleftarrow{\alpha} R + \gamma V(S')$
Q-Policy Iteration $Q(s,a) \leftarrow \mathbb{E}[R + \gamma Q(S',A') \mid s,a]$	Sarsa $Q(S,A) \xleftarrow{\alpha} R + \gamma Q(S',A')$
Q-Value Iteration $Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S',a') \mid s,a\right]$	Q-Learning $Q(S,A) \xleftarrow{\alpha} R + \gamma \max_{a' \in \mathcal{A}} Q(S',a')$