AI Planning for Autonomy

11. Reinforcement Learning

How to learn without a model

Tim Miller

Winter Term 2017

Agenda

Agenda

Planning and Learning

So far, the AI Plannning course:

- Planning required a **complete** description of the world/problem/environment.
- If we allow **stochasticity** but still **fully known** environments (e.g. Backgammon), we arrive at a Stochastic Planning problem (MDP).
- If we deal with an **unknown** environment, which can only be **learned through experience**, we get a Machine Learning problem.

Reinforcement Learning $\approx$ Learning + Planning

Scenarios

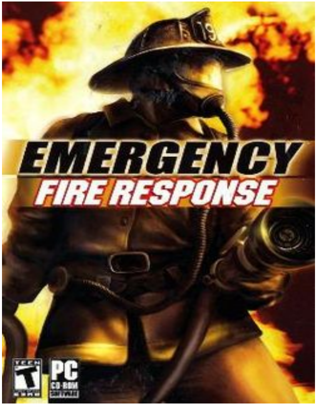


Figure: Emergency Respose

Figure: Bridge Card Game

$\rightarrow$ What these two scenarios have in common?

Special case of (PO)MDP where Probability and Reward distributions are unknown.

Scenarios

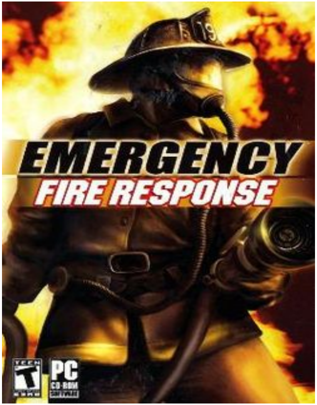


Figure: Emergency Respose

Figure: Bridge Card Game

$\rightarrow$ What these two scenarios have in common?

Special case of (PO)MDP where Probability and Reward distributions are unknown.

Estimating Probability and Reward Distribution

- Learn from past experience, E.x. collected data
- Use a simulator to gain experience. E.x. Fire Simulation Models, Computer Bridge Game Engine, Go Engine, etc.

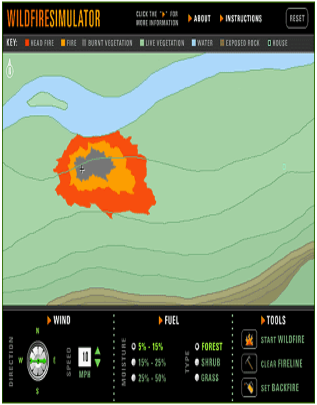


Figure: WildFire Simulator

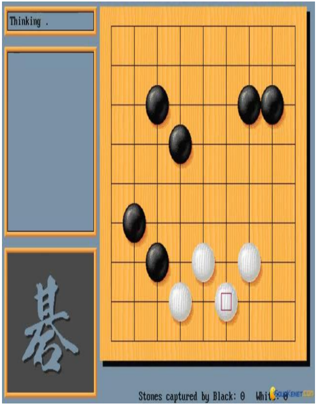


Figure: Go Simulator

Key MCTS and RL

$\rightarrow$ Monte Carlo Tres Search samples a **Policy Tree** through experience. Recomputes Tree for evey new state.

$\rightarrow$ Reinforcement Learning learns a **full policy mapping states to actions** through experience.

Applications of RL

- Checkers (Samuel, 1959)
  first use of RL in an interesting real game
- (Inverted) Helicopter Flight (Ng et al. 2004)
  better than any human
- Computer Go (AlphaGo 2016)
  AlphaGo beats Go world champion Lee Sedol 4:1
- Atari 2600 Games (DQN & Blob-PROST 2015)
  human-level performance on half of 50+ games
- Robocup Soccer Teams (Stone & Veloso, Reidmiller et al.)
  World's best player of simulated soccer, 1999; Runner-up 2000
- Inventory Management (Van Roy, Bertsekas, Lee & Tsitsiklis)
  10-15% improvement over industry standard methods
- Dynamic Channel Assignment (Singh & Bertsekas, Nie & Haykin)
  World's best assigner of radio channels to mobile telephone calls
- Elevator Control (Crites & Barto)
  (Probably) world's best down-peak elevator controller
- Many Robots
  navigation, bi-pedal walking, grasping, switching between skills, . . .
- TD-Gammon and Jellyfish (Tesauro, Dahl)
  World's best backgammon player. Grandmaster level

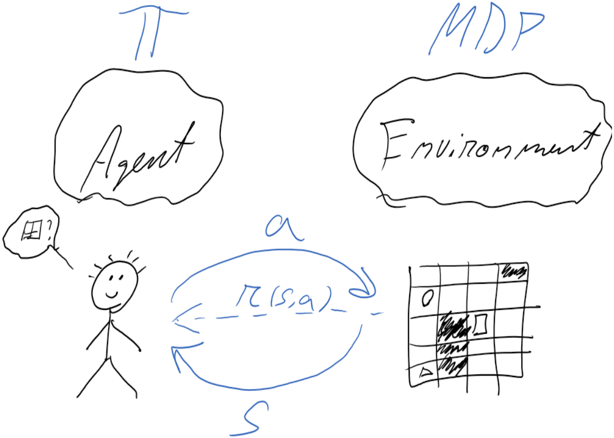
Agenda

Intuition

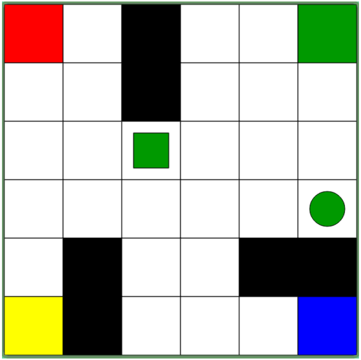
Assumptions:

- Agent does **not** know the Environment (MDP),
- Agent **experieces** the Environment **interacting** with it,
- Environment **reveals** to agent in the form of a **state** s,
- Agent **influences** the environment with an **action** a, and **recieves feedback** as a **reward** $r(s, a)$ explaining the effect of action a applied to state s.

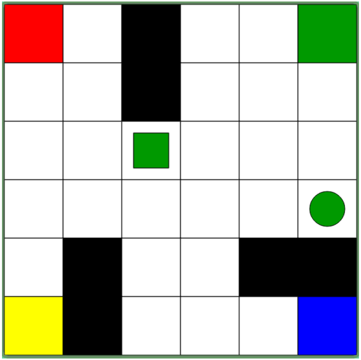
Graphic Intuition



Example

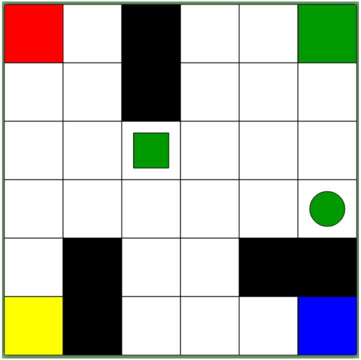


Example



- What was the process you took?
- What did you learn?
- What assumptions did you use?

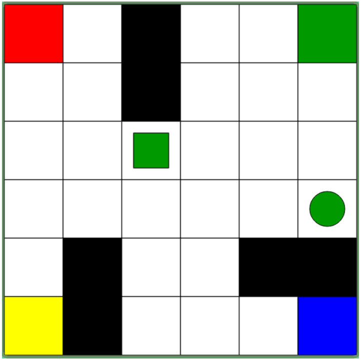
Example



- What was the process you took?
- What did you learn?
- What assumptions did you use?

$\rightarrow$ Imagine how hard it is for a computer that doesn't have any assumption!

Example



- What was the process you took?
- What did you learn?
- What assumptions did you use?

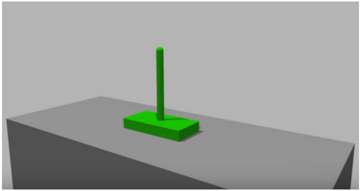
→ Imagine how hard it is for a computer that doesn't have any assumption!

Another Example: Enviornment Design

Imagine you are the environment and you want the **agent** to avoid the **pole falling** beyond certain angle.

Question

Which **reward** function should use the environment?



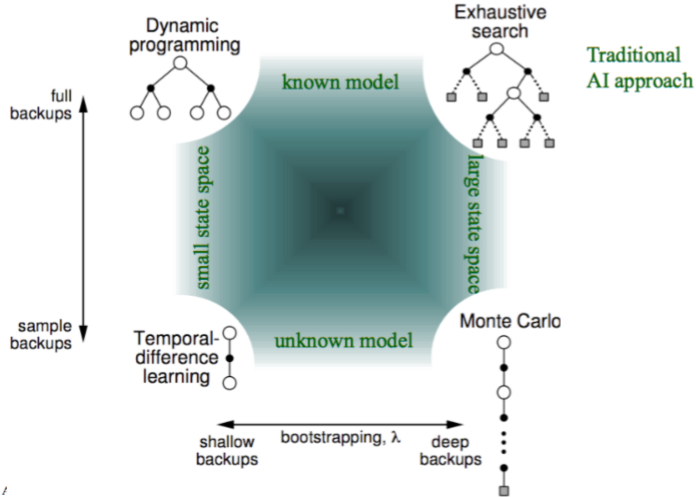
$\rightarrow$ See how it works! . . . and similarly See how it works for real!

Evaluating Learners

You can judge your algorithm given:

- Value of the policy in terms of expected reward
- Computational Time
- Experience Complexity (time): How much data/interactions it needs

Approaches to AI Planning and Learning



R. S. Sutton and /

Agenda

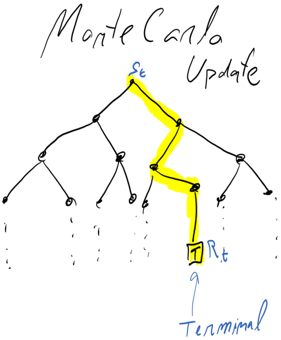
1. Motivation
2. Reinforcement Learning
3. TD(λ)
4. Control TD
5. Conclusion

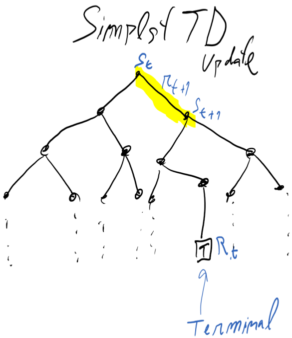
Temporal Difference Learning

Use **experience** to solve the prediction problem.

- Combination of Monte Carlo (MC) ideas and Dynamic Programming (DP):
  - MC: can **learn from raw** experience, **without a model**,
  - DP: can **update estimates** based on **other learned estimates**, no waiting for final outcome (bootstrap).

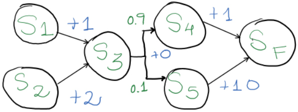
Predict Expected sum of discounted rewards, by learning over time given $< s, a, r >^*$ sequences.





Example of TD

Given the following Markov Chain:



$$V(s) = \begin{cases} 0, & \text{if } s = s_F \\ E[r + \gamma V(s')], & \text{otherwise} \end{cases}$$

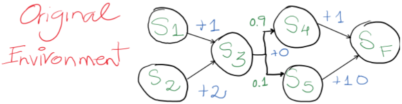
$\rightarrow$ What's the value of $\mathbf{V}(\mathbf{s_F})$ and $\mathbf{V}(\mathbf{s_i})$ for $i = 0, \ldots, 5$?

For simplicity, assume the **discount factor** $\gamma = 1$.

Example of TD: From Data

Data Sequences
(episodes)

1. $S_1 \xrightarrow{+1} S_3 \xrightarrow{+0} S_4 \xrightarrow{+1} S_F$
2. $S_1 \xrightarrow{+1} S_3 \xrightarrow{+0} S_5 \xrightarrow{+10} S_F$
3. $S_1 \xrightarrow{+1} S_3 \xrightarrow{+0} S_4 \xrightarrow{+1} S_F$
4. $S_1 \xrightarrow{+1} S_3 \xrightarrow{+0} S_4 \xrightarrow{+1} S_F$
5. $S_2 \xrightarrow{+2} S_3 \xrightarrow{+0} S_5 \xrightarrow{+10} S_F$



→ What's the value of $\mathbf{V}(\mathbf{s_1})$ after 3 Episodes?
→ What's the value of $\mathbf{V}(\mathbf{s_1})$ after 4 Episodes?

Example of TD: Updating Estimates Incrementally

$$V_T(s_1) = \frac{(T-1)V_{T-1}(s_1) + R_T(s_1)}{T}$$

$$= \frac{(T-1)}{T} V_{T-1}(s_1) + \frac{1}{T} R_T(s_1)$$

$$= V_{T-1}(s_1) + \underline{\alpha_T} \left(R_T(s_1) - V_{T-1}(s_1)\right)$$

Where $\alpha_T = \frac{1}{T}$ → learning rate

TD: Properties of Learning Rates

$$\lim_{T \to \infty} V_T = V^*(s)$$

If 1. $\sum_T \alpha_T = \infty$

2. $\sum_T \alpha_T^2 < \infty$

Properties of learning rate

Ex: $\frac{1}{T^{1/2}} > \alpha_T \geq \frac{1}{T}$

Rule of thumb

When power of denominator of (2) is bigger than 1, then is going to converge

- (1) guarantees γ_T is **big enough**, so that you do not stop learning until the limit, and you keep moving towards V^*,
- (2) guarantees that γ_T is **not too big**, so you get rid of the noise.

TD(λ) for estimating V^*

Initialize $V(s)$ arbitrarily and $e(s) = 0$, for all $s \in \mathcal{S}$
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    $a \leftarrow$ action given by π for s
    Take action a, observe reward, r, and next state, s'
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $e(s) \leftarrow e(s) + 1$
    For all s:
      $V(s) \leftarrow V(s) + \alpha\delta e(s)$
      $e(s) \leftarrow \gamma\lambda e(s)$
    $s \leftarrow s'$
  until s is terminal

- $e(s)$, stands for the Elegibility function (initially all states are ineligible for updates)
- γ is the discount factor, α is the learning rate, δ changes only the Elegibility function.

TD(1) example

Initialize $V(s)$ arbitrarily and $e(s) = 0$, for all $s \in \mathcal{S}$
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    $a \leftarrow$ action given by π for s
    Take action a, observe reward, r, and next state, s'
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $e(s) \leftarrow e(s) + 1$
    For all s:
      $V(s) \leftarrow V(s) + \alpha \delta e(s)$
      $e(s) \leftarrow \gamma \lambda e(s)$
    $s \leftarrow s'$
  until s is terminal

$$\begin{array}{lcccc} \textit{Episode} & s_1 \xrightarrow{r_1} & s_2 \xrightarrow{r_2} & s_3 \xrightarrow{r_3} & s_F \\ e(s) & 1 & 0 & 0 & \end{array}$$

$$\Delta V_T(s1) \leftarrow \alpha(r_1 + \gamma V_{T-1}(s_2) - V_{T-1}(s_1))$$

TD(1) example

Initialize $V(s)$ arbitrarily and $e(s) = 0$, for all $s \in \mathcal{S}$
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    $a \leftarrow$ action given by π for s
    Take action a, observe reward, r, and next state, s'
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $e(s) \leftarrow e(s) + 1$
    For all s:
      $V(s) \leftarrow V(s) + \alpha\delta e(s)$
      $e(s) \leftarrow \gamma\lambda e(s)$
    $s \leftarrow s'$
  until s is terminal

$$\begin{array}{lccccccc} \textit{Episode} & s_1 & \xrightarrow{r_1} & s_2 & \xrightarrow{r_2} & s_3 & \xrightarrow{r_3} & s_F \\ e(s) & \gamma & & 1 & & 0 & & \end{array}$$

$$\Delta V_T(s1) \leftarrow \alpha(r_1 + \gamma V_{T-1}(s_2) - V_{T-1}(s_1)) \; + \; \gamma\alpha(r_2 + \gamma V_{T-1}(s_3) - V_{T-1}(s_2))$$
$$\Delta V_T(s2) \leftarrow 0 \qquad\qquad + \; \alpha(r_2 + \gamma V_{T-1}(s_3) - V_{T-1}(s_2))$$

TD(1) example

Initialize $V(s)$ arbitrarily and $e(s) = 0$, for all $s \in \mathcal{S}$
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    $a \leftarrow$ action given by π for s
    Take action a, observe reward, r, and next state, s'
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $e(s) \leftarrow e(s) + 1$
    For all s:
      $V(s) \leftarrow V(s) + \alpha\delta e(s)$
      $e(s) \leftarrow \gamma\lambda e(s)$
    $s \leftarrow s'$
  until s is terminal

$$\begin{array}{lcccc} \textit{Episode} & s_1 \xrightarrow{r_1} & s_2 \xrightarrow{r_2} & s_3 \xrightarrow{r_3} & s_F \\ e(s) & \gamma & 1 & 0 & \end{array}$$

$$\Delta V_T(s1) \leftarrow \alpha(r_1 + \gamma \cancel{V_{T-1}(s_2)} - V_{T-1}(s_1)) \;+\; \gamma\alpha(r_2 + \gamma V_{T-1}(s_3) - \cancel{V_{T-1}(s_2)})$$

$$\Delta V_T(s2) \leftarrow 0 \qquad\qquad + \;\alpha(r_2 + \gamma V_{T-1}(s_3) - V_{T-1}(s_2))$$

TD(1) example

Initialize $V(s)$ arbitrarily and $e(s) = 0$, for all $s \in \mathcal{S}$
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    $a \leftarrow$ action given by π for s
    Take action a, observe reward, r, and next state, s'
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $e(s) \leftarrow e(s) + 1$
    For all s:
      $V(s) \leftarrow V(s) + \alpha\delta e(s)$
      $e(s) \leftarrow \gamma\lambda e(s)$
    $s \leftarrow s'$
  until s is terminal

Episode	$s_1 \xrightarrow{r_1}$	$s_2 \xrightarrow{r_2}$	$s_3 \xrightarrow{r_3}$	s_F
$e(s)$	γ	1	0	

$$\Delta V_T(s1) \leftarrow \alpha(r_1 + \gamma r_2 + \gamma^2 V_{T-1}(s_3) - V_{T-1}(s_1))$$
$$\Delta V_T(s2) \leftarrow \alpha(r_2 + \gamma V_{T-1}(s_3) - V_{T-1}(s_2))$$

TD(1) example

```
Initialize V(s) arbitrarily and e(s) = 0, for all s ∈ S
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        a ← action given by π for s
        Take action a, observe reward, r, and next state, s′
        δ ← r + γV(s′) − V(s)
        e(s) ← e(s) + 1
        For all s:
            V(s) ← V(s) + αδe(s)
            e(s) ← γλe(s)
        s ← s′
    until s is terminal
```

$$\begin{array}{ll} \textit{Episode} & s_1 \xrightarrow{r_1} s_2 \xrightarrow{r_2} s_3 \xrightarrow{r_3} s_F \\ e(s) & \gamma^2 \quad\quad \gamma \quad\quad 1 \end{array}$$

$$\Delta V_T(s1) \leftarrow \alpha(r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 V_{T-1}(s_F) - V_{T-1}(s_1))$$

$$\Delta V_T(s2) \leftarrow \alpha(r_2 + +\gamma r_3 + \gamma^2 V_{T-1}(s_F) - V_{T-1}(s_2))$$

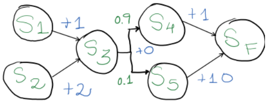
$$\Delta V_T(s3) \leftarrow \alpha(r_3 + \gamma V_{T-1}(s_F) - V_{T-1}(s_3))$$

TD(1) From Data

Data Sequences
(episodes)

1. $S_1 \xrightarrow{+1} S_3 \xrightarrow{+0} S_4 \xrightarrow{+1} S_F$
2. $S_1 \xrightarrow{+1} S_3 \xrightarrow{+0} S_5 \xrightarrow{+10} S_F$
3. $S_1 \xrightarrow{+1} S_3 \xrightarrow{+0} S_4 \xrightarrow{+1} S_F$
4. $S_1 \xrightarrow{+1} S_3 \xrightarrow{+0} S_4 \xrightarrow{+1} S_F$
5. $S_2 \xrightarrow{+2} S_3 \xrightarrow{+0} S_5 \xrightarrow{+10} S_F$

Original Environment



→ What's the value of $\mathbf{V}(\mathbf{s}_2)$ based on $TD(1)$?
→ What's the value of $\mathbf{V}(\mathbf{s}_2)$ based on Maximum Likelihood ?
→ Does TD(1) converge to ML with finite amount of data?

TD(0) Rule

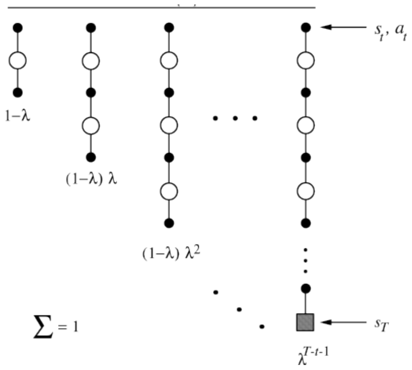
$$V_T(s) = V_T(s) + \alpha_T(r + \gamma V_T(s') - V_T(s))$$
$$V_T(s) = \mathbf{E}_{s'}[r + \gamma V_T(s')]$$

$\rightarrow$ Given **finite** amount of data (episodes), repeated **infinitely** often, then TD(0) **converges** to Maximum Likelihood (ML)

Back to TD(λ)

How to update? Look at your lambda!

- $\lambda = 0$, It's a 1-step update using current prediction of estimated discounted reward
- $\lambda = 1$, It's a all-step update using the accumulated discounted reward
- $\lambda = 1/2$, It's a weighted-step update using current prediction of discounted reward



Agenda

Sarsa(λ): On-Policy Control TD(λ)

For simplicity, next slides show Sarsa(λ), for $\lambda = 0$.

Instead of estimating V for a fixed policy/data, Control TD:

- Estimates $Q^{\pi}(s, a)$ state action pairs, for the current behavior policy π,
- Continuously updates the policy π with respect to the current estimate Q

Key Difference: Learner makes the choices of what to experience $\rightsquigarrow$ known problem?

Sarsa(λ): On-Policy Control TD(λ)

For simplicity, next slides show Sarsa(λ), for $\lambda = 0$.

Instead of estimating V for a fixed policy/data, Control TD:

- Estimates $Q^\pi(s, a)$ state action pairs, for the current behavior policy π,
- Continuously updates the policy π with respect to the current estimate Q

Key Difference: Learner makes the choices of what to experience $\rightsquigarrow$ known problem? Exploration vs. Exploitation!

```
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) ← Q(s,a) + α[r + γQ(s',a') − Q(s,a)]
        s ← s'; a ← a';
    until s is terminal
```

On-Policy: Uses the action chosen by the policy for the update!

Sarsa(λ): On-Policy Control TD(λ)

For simplicity, next slides show Sarsa(λ), for $\lambda = 0$.

Instead of estimating V for a fixed policy/data, Control TD:

- Estimates $Q^\pi(s, a)$ state action pairs, for the current behavior policy π,
- Continuously updates the policy π with respect to the current estimate Q

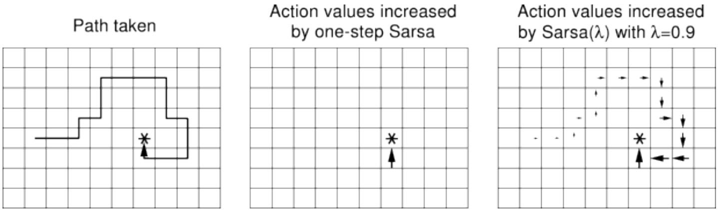
Key Difference: Learner makes the choices of what to experience $\rightsquigarrow$ known problem? Exploration vs. Exploitation!

```
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) ← Q(s,a) + α[r + γQ(s',a') − Q(s,a)]
        s ← s'; a ← a';
    until s is terminal
```

On-Policy: Uses the action chosen by the policy for the update!

Sarsa(λ): example

Sarsa(λ), uses eligibilty traces as in TD(λ). In the example below we see the effect of using lambda for $\lambda = 0$ 1-step, or $\lambda = 0.9$



Q-learning(λ): Off-Policy Control TD(λ)

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)
    $Q(s, a) \leftarrow Q(s, a) + \alpha\big[r + \gamma Q(s', a') - Q(s, a)\big]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

Figure: Sarsa Algorithm

Initialize $Q(s, a)$ arbitrarily
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

Figure: Q-Learning Algorithm

Off-Policy: Ignores the action chosen by the policy, uses the best action $\mathrm{argmax}_{a'}\, Q(s', a')$ for the update!

On-Policy **SARSA** learns action values relative to the policy it follows, while Off-Policy **Q-Learning** does it relative to the greedy policy.

Sarsa vs. Q-Learning

Figure: Sarsa Algorithm

Figure: Q-Learning Algorithm

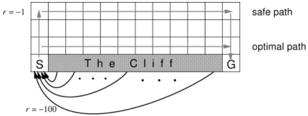


Figure: Rewards

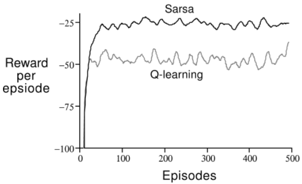


Figure: Single run results

Q: What's the effect of the ϵ exploration?

Q: how can we make SARSA converge to the optimal policy?

Sarsa vs. Q-Learning

Figure: Sarsa Algorithm

Figure: Q-Learning Algorithm

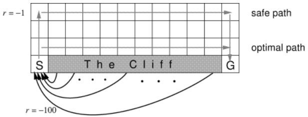


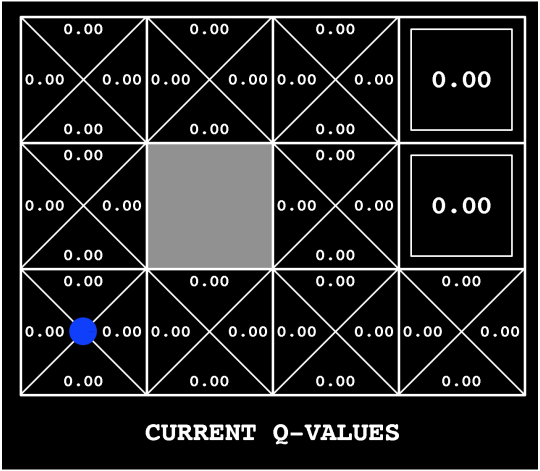
Figure: Rewards

Figure: Single run results

Q: What's the effect of the ϵ exploration?

Q: how can we make SARSA converge to the optimal policy?

Q-Learning: example



Q-learning in action!

Q-Learning: Properties

SARSA and Q-Learning converge to optimal policy, even if you are **acting suboptimally**, but require exploration!

Balance exploration:

- ϵ**-greedy**: with probability ϵ choose a random action,
- Do we know better?

Q-Learning: Properties

SARSA and Q-Learning converge to optimal policy, even if you are **acting suboptimally**, but require exploration!

Balance exploration:

- ϵ**-greedy**: with probability ϵ choose a random action,
- Do we know better? use simple exploration functions or just use **UCB1** from multi armed-bandits!

Add exploration term to $argmax_{a'}(Q(s', a') + f_{exploration}(s', a'))$

Q-Learning: Properties

SARSA and Q-Learning converge to optimal policy, even if you are **acting suboptimally**, but require exploration!

Balance exploration:

- **ϵ-greedy**: with probability ϵ choose a random action,
- Do we know better? use simple exploration functions or just use **UCB1** from multi armed-bandits!

Add exploration term to $argmax_{a'}(\mathcal{Q}(s', a') + f_{exploration}(s', a'))$

Agenda

Summary

If we **know** MDP:

- **Offline**: Value Iteration, Policy Iteration,
- **Online**: Classic Search, Monte Carlo Search Tree and friends.

If we do **not** know MDP:

- **Offline**: Reinforcement Learning
- **Online**: Classic Search, Monte Carlo Tree Search and friends.

Always think weather you need to balance Exploration and Exploitation

Once you've got your pacman Q-learning working in python, you can test it on all the evironments on OpenAI!

Toolkit for developing and testing RL algorithms

Reading

- *Introduction to Reinforcement Learning* [*Sutton and Barto*]

  Available at:

  `https://webdocs.cs.ualberta.ca/~sutton/book/the-book.html`

  Content: Great entry level book to Reinforcement Level written by the founders of the field.

- *Slides about Approximate Q-learning for PacMan*

  Available at:

  `https://www.cs.swarthmore.edu/~bryce/cs63/s16/slides/3-25_approximate_Q-learning.pdf`

  Content: Great technique if you want to use RL for the competition!

- *Deep Q-learning for Atari*

  Available at:

  `http://www.davidqiu.com:8888/research/nature14236.pdf`

  Content: Convolutional Neural Networks (NN) to estimate $Q(s, a)$. The input for the NN is the state, and the output is the esimated reward for each action.